THE STATE MATRIX METHOD FOR THE SYNTHESIS OF DIGITAL LOGIC SYSTEMS

By

ROBERT L. WOODS

Bachelor of Science

Southern Methodist University

Dallas, Texas

1967

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1970

THE STATE MATRIX METHOD FOR

THE SYNTHESIS OF DIGITAL

LOGIC SYSTEMS

Thesis Approved:

Karl N. Reid

Thesis Adviser

E C Fitch

D. Durham

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those people involved in this thesis. In particular, I wish to thank Messrs. Engelland, Brun, and Gertsen for their encouragement during the earily stages of development of this synthesis philosophy.

Thanks are in order to my committee, Professors K. N. Reid, E. C. Fitch, and P. A. McCollum, and a special thanks to Dr. Reid for serving as my thesis adviser.

I also wish to thank Dr. Fitch for allowing this thesis to be published under the Basic Fluid Power Research Program even though this work was not supported by BFPR.

I want to thank Velda Davis for providing such an excellent typing service. It is greatly appreciated.

Those of you who have not undertaken a formal writing such as this cannot realize how important a good wife can be. Those who have, know what I mean when I say, thank you, Brenda.

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
	Background	2
	Development of the State Matrix Method . .	3
	Scope and Results of Study	5
II.	THE FEEDBACK SEQUENTIAL STATE MATRIX SYNTHESIS PROCEDURE	7
	Formal Matrix Representation	7
	Persistent States	11
	Memory Assignment	19
	Counting Sequences	23
	Procedure Summary	26
III.	THE STATE MATRIX SYNTHESIS PROCEDURE FOR RANDOM INPUT CIRCUITS	34
	The Primitive Flow Table	34
	Formal Matrix Representation	39
	Memory Assignment	41
	Output and Switching Conditions	43
	Procedure Summary	46
IV.	DIGITAL EQUATION SIMULATION AND THE CANONICAL FLOW TABLE	54
	Digital Equation Simulation	55
	Canonical Flow Table	61
V.	DIGITAL COMPUTER PROGRAMS	67
	Synthesis Program LOGSYN	68
	Simulation Program LOGSIM	70
VI.	SUMMARY AND CONCLUSIONS	72
	Summary	72
	Comparison to Other Techniques	73
	Suggestions for Further Study	77
A SELECTED BIBLIOGRAPHY		80

Chapter Page

APPENDIX A - THE PASSIVE MEMORY 81

APPENDIX B - LISTING OF COMPUTER PROGRAMS 90

APPENDIX C - EXAMPLE COMPUTER SOLUTIONS 104

LIST OF TABLES

Table		Page
I.	The Developing State Matrix Relation for the Sequence Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$	10
II.	The State Matrix Relation for the Sequence Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$	12
III.	The State Matrix Relation for Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$, Z_1, $\bar{Z}_1$	20
IV.	The State Matrix for a 2, 2, 1 Counter	24
V.	Primitive Flow Table for Example 3.1	36
VI.	Primitive Flow Table for Example 3.2	37
VII.	Canonical Flow Table for Example 3.2	38
VIII.	The State Matrix Relation for Example 3.1 . . .	40
IX.	The State Matrix Relation for Example 3.2 . . .	41
X.	The Unique State Matrix Relation for Example 3.2	43
XI.	The Primitive Flow Table for Example 3.3	51
XII.	Step-By-Step Development of Digital Equation Simulation	58
XIII.	The Development of the Canonical Flow Table . .	63

LIST OF FIGURES

Figure		Page
1.	Hydraulic Circuit Illustrating Notation	8
2.	Synthesized Hydraulic Circuit for Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$	16
3.	Hydraulic Implementation for Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$, Z_1, $\bar{Z}_1$	22
4.	Hydraulic Implementation for 1, $\bar{1}$, 1, $\bar{1}$, 2, 3, 1, $\bar{2}$, $\bar{1}$, $\bar{3}$	33
5.	Logic Circuit for Example 3.3	53
6.	Flow Chart for Simulation Method	56
7.	Flow Diagram for Logic Systems Program	69
8.	Passive Memory Valve	82
9.	Fluidic Memory Circuit	84
10.	Passive Memory Assignment Circuits	86
11.	Passive Memory Code Schematic	87

CHAPTER I

INTRODUCTION

The technology of switching circuit theory, although relatively young, has found great application and utility in modern design. Most of the theory has been developed for application in electrical engineering since electronics has dominated the field of computation and logic for the last dew decades.

Recent years have seen a rebirth of the use of a fluid medium to perform the logic and computation in sequential machines. The newly emerging field of fluid technology termed "fluidics" is one major reason for this rebirth of fluid logic. Since fluid power is often used as the muscle of machines, it is convenient also to use the fluid itself for the required computation in order to avoid the electrical to fluid interfaces.

To realize maximum utilization of fluid logic devices, it is necessary to develop a technology of switching circuits applicable to fluid circuits. The theory should consider the unique properties of fluid devices not only in the implementation of circuits, but also in the synthesis procedure itself. The synthesis procedure presented in this thesis does take advantage of the unique properties of devices in order to produce simple fluid circuits containing minimal hardware.

Background

Modern switching theory had its origin in 1938 when C. E. Shannon (9), of M.I.T., applied the laws of Boolean algebra to the representation of electrical switches.[1] Although this was a great advancement for combinational switching circuits, there was no formal procedure for the synthesis of sequential switching circuits until 1954 when D. A. Huffman (3) and E. F. Moore (8) independently developed the synthesis technique which is used today. This technique has gained such widespread use and application that today it is taught at every major university and is even referred to as the "classical method". The synthesis procedure presented in this thesis relies upon much of the notation of the classical method. The reader not familiar with this method, should refer to a book on classical switching theory (2), (5), (7), (8).

E. C. Fitch (2), of Oklahoma State University, was one of the first authors to apply the methods of sequential switching circuit theory to hydraulics. However, his work did not take into account any special properties of hydraulic valves except in the implementation of logic circuits.

---

[1]Numbers in parentheses refer to references in the Selected Bibliography.

Later work at Oklahoma State University by J. H. Cole (1) did consider the properties of devices in the synthesis procedure. Dr. Cole used the properties of the passive memory devices to produce extremely simple circuits for the feedback sequential type problem. This work has been a major advancement for the field even though its scope of application is limited.

G. E. Maroney (6) extended Cole's tabular method to include the random input type circuit. This method was fundamentally the same as Cole's except that the random input possibility necessarily complicated the execution of the method. This technique also utilized the passive memory effect to reduce hardware.

Development of the State Matrix Method

The state matrix synthesis procedure evolved from the assumption that the outputs are related to the inputs and the past state of the system. This relationship can be written in matrix form as:

$$\left[Z\right] = \left[M\right]\left[X\right].$$

Here, the outputs are contained in the $\left[Z\right]$ vector, the $\left[X\right]$ vector contains the inputs, and the matrix $\left[M\right]$ contains output and memory information. This binary matrix changes with time to yield different outputs representing the different states of a sequence.

Early experiments with this type of synthesis were restricted to the feedback sequential type problems because of their simplicity. A close examination of the resulting equations revealed that they were essentially identical to those obtained from Cole's method. This was very encouraging since Cole's method was known to produce valid expressions. The matrix arrangement of this method also gave insight to many of the hidden subtleties of Cole's method.

Once the rules for the synthesis of feedback sequential circuits using state matrices were defined, the method was extended to handle the random input problems. The main difference between the state matrix methods for random input and feedback sequential problems was the input vector used. The feedback sequential input vector contained only the changed input, whereas the random input vector contained the total input state.

The random input form of the state matrix synthesis procedure has since received more attention since it is the more general procedure. This form will also handle the feedback sequential problems in some respects better than the original state matrix method. Hereafter, the random input form of this method will be referred to simply as the "state matrix method", and the method using the changed input vector will be referred to as the "feedback sequential state matrix method."

Scope and Results of Study

Although the state matrix synthesis procedure is the most important item in this thesis, many other original topics have arisen from this study. The major accomplishments of this study are:

(1) The development of the feedback sequential state matrix synthesis procedure. (Chapter II)

(2) The development of the state matrix synthesis procedure for random input circuits. (Chapter III)

(3) A digital computer program to perform the state matrix synthesis procedure. (Chapter V)

(4) The development of a simulation technique to check the logical implications of digital equations. (Chapter IV)

(5) A digital computer program to perform the digital equation simulation and to formulate the implied primitive flow table. (Chapter V)

(6) The definition of a standard format for the primitive flow table. (Chapter IV)

The state matrix synthesis procedures have the following distinguishing features:

(1) The basic concepts of circuit synthesis are much easier to grasp than those of other methods.

(2) The execution of the procedure is straightforward with few or no exceptions to

established rules.

(3) The resulting digital equations have few of the usual logical complications.

(4) The procedure takes advantage of device properties to produce circuits with fast response and minimal hardware.

(5) There is virtually no limitation upon the size or length of the problems which can be handled.

The simulation method presented here provides a check upon the digital equations resulting from a synthesis procedure. Each possible input change is systematically inspected for its effect upon circuit equations and the resulting transitions are recorded in a primitive flow table. This flow table may then be compared to the original flow table which should contain identical information.

In comparing the simulated flow table to its original primitive flow table, it is convenient, if not necessary, to establish a standard flow table format. For this reason, a method similar to the simulation method is used to define the canonical flow table format.

The computer programs included in Appendix B perform the mechanics of synthesis or simulation rapidly and accurately. These programs encompass all of the defined rules and methods for the analysis of digital logic systems and can be utilized to good advantage in design work.

CHAPTER II

THE FEEDBACK SEQUENTIAL STATE MATRIX SYNTHESIS PROCEDURE

Although feedback sequential circuits are comparatively simple, they have found a large field of application in modern automation. Consequently, the synthesis of such circuits is of major importance to industrial designers.

Feedback sequential circuits are characterized by their use of a signal indicating the completion of one event to initiate the next event in a prescribed sequence. Feedback sequential circuits are automatic and, once started, require no further attention to sustain sequential action.

Formal Matrix Representation

In sequential circuits, each element is associated with one corresponding output from the logic circuit. In a hydraulic circuit this element is typically a hydraulic cylinder and the output is the fluid flow which actuates the cylinder. Since there is usually more than one element in a sequential machine, it is convenient to let Z_1 represent the output which extends cylinder one and $\overline{Z}_1$ represent the retract output for the cylinder.[1] The signal X_2 is used as an input to the logic circuit indicating the full extension of cylinder two, and the signal $\overline{X}_2$ appears when cylinder two is fully retracted. Figure 1 illustrates a physical realization of these variables. The reader who is unfamiliar with hydraulic circuit notation should refer to the literature.

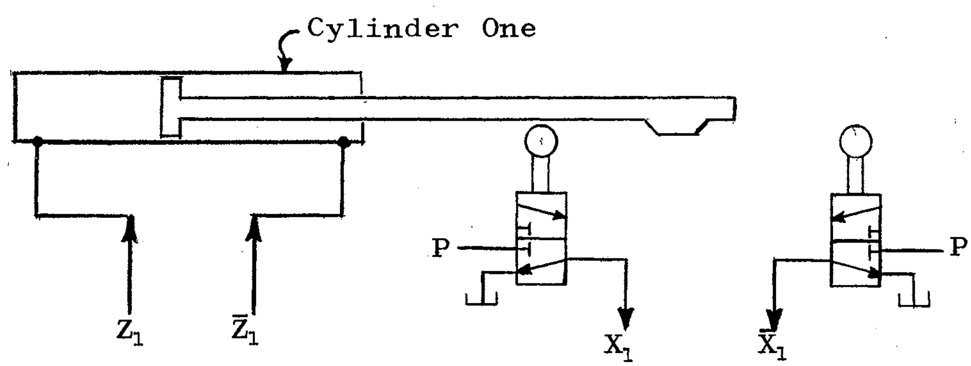


Figure 1. Hydraulic Circuit Illustrating Notation

Using this notation, a sequence involving two cylinders

---

[1]This notation is somewhat unfortunate since Z_1 is used in this chapter to specify only the <u>change</u> of cylinder one, not its continuous state. Also, Z_1 and $\overline{Z}_1$ are not perfect complements since the specification of one does not imply the other. A more appropriate notation would be ΔZ_1, etc.; however, the Z, $\overline{Z}$ notation is used here for simplicity. A similar statement is true for the inputs X and $\overline{X}$ used in this chapter only.

can be written as Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$. This implies that cylinder one extends, then cylinder two extends, cylinder one retracts, cylinder two retracts, and then the entire sequence is repeated indefinitely. Each event is initiated by the completion of the proceeding event.

The synthesis of a circuit to execute this sequence proceeds from the assumption that the required outputs from the logic circuit are related to the inputs by the matrix equation given below.

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ \vdots \\ Z_n \\ \bar{Z}_n \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ \cdot & & & \cdot \\ \vdots & & m_{ij} & \vdots \\ \cdot & & & \cdot \\ m_{n1} & \cdots & \cdots & m_{nn} \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ \vdots \\ X_n \\ \bar{X}_n \end{bmatrix} \tag{1}$$

Recall from the rules of matrix multiplication that when multiplying the matrix $[M]$ by the $[X]$ vector, every entry in the j^{th} column of $[M]$ is multiplied by the element in the j^{th} row of $[X]$. Thus, each column in $[M]$ is associated only with the corresponding input element of $[X]$.

For the sequence under consideration, the first event is the extension of cylinder one which results from the previous retraction of cylinder two. Thus, the state number 1 is entered in the matrix in the row of the Z_1 output and the column associated with the $\bar{X}_2$ input (column four). See Table I.

The next event, the extension of cylinder two, is initiated by the full extension of cylinder one. Hence, the state number 2 is entered in the Z_2 row and the X_1 column. Similarly, state 3 is in the $\bar{Z}_1$ row and the X_2 column. The sequence is completed by state 4 in the $\bar{Z}_2$ row, $\bar{X}_1$ column.

TABLE I

THE DEVELOPING STATE MATRIX RELATION
FOR THE SEQUENCE Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} \sim \begin{array}{|c|c|c|c|} \hline \quad & \quad & \quad & 1 \\ \hline & & 3 & \\ \hline 2 & & & \\ \hline & 4 & & \\ \hline \end{array} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \end{bmatrix}$$

After all state numbers are entered into Table I, the state matrix must be inspected to ensure that each state is unique and does not represent any contradictions. For this extremely simple problem, this is true and further attention is not required. Table I may now be written matrix form by placing a logical "1" for each state and a "0" elsewhere.

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \end{bmatrix} \qquad (2a)$$

The matrix in Table I is termed the <u>state matrix</u> since it only shows the states of the sequence. The matrix in Equation (2a) is termed the <u>output matrix</u> because Equation (2a) is merely a set of digital output equations in matrix notation. Writing Equations (2a) in longhand, one has:

$$\begin{aligned} Z_1 &= \bar{X}_2 \\ \bar{Z}_1 &= X_2 \\ Z_2 &= X_1 \\ \bar{Z}_2 &= \bar{X}_1 \end{aligned} \qquad (2b)$$

Note that the variables used in digital equations are Boolean or binary logic variables.

Since this introductory problem is simple and requires no memory, one could almost predict the results without the use of any formal synthesis procedure. However, further problems in this chapter illustrate the general case.

Persistent States

The problem of persistent states are prevalent in almost every feedback sequential circuit. Persistent states result when signals remain on long enough to form a contradiction. The exact cause and remedy for this can best be illustrated by an example.

Consider as example 2 the sequence Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$. The state numbers are entered into Table II in exactly the same fashion as the previous example. That is, state 1 is in the Z_1 row, $\bar{X}_2$ column. State 2 is in the $\bar{Z}_1$ row, X_1 column. The remaining state numbers are entered similarily and the resulting state matrix is shown in Table II.

TABLE II

THE STATE MATRIX RELATION FOR
THE SEQUENCE Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} \sim \begin{bmatrix} & & & 1 \\ 2 & & & \\ & 3 & & \\ & & 4 & \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \end{bmatrix}$$

If this state matrix were now converted into the output matrix by placing a "1" for the states and a "0" elsewhere, the following equations would result:

$$\begin{aligned} Z_1 &= \bar{X}_2 \\ \bar{Z}_1 &= X_1 \end{aligned} \qquad (3)$$

Reference to these equations and the state sequence in Table II reveals that cylinder one would be extended by $\bar{X}_2$ and subsequently retracted by X_1. However, at the time of retraction the extent signal $\bar{X}_2$ would still be on, because cylinder two has not been changed since its retraction. Hence, there is a contradiction because cylinder one is trying to extend and retract simultaneously. The signal which remains on creating a contradiction is called a persistent state. In this case, the persistent state is the signal $\bar{X}_2$ from state 1. This problem arises because only the changed input and the changed output are used in the state matrix relation. An event is specified only by the variables that change, not by the present state of all variables.

This condition can be alleviated by entering a shut-off memory element at the persisting state and its complement at the contradiction. The memory element should be in the "set" position prior to the persistent state and should be in the "reset" position either prior to or on the contradicting state. The complemented memory signal is not used in state signal formulation; it is only used as a reminder when it should be off or in the "reset" position.

For the problem under consideration, the persistent state 1 contradicts state 2. Consequently, state 1 must be modified with a shut-off memory element, say W_{12}. This element can then be used to shut-off the persistent signal thereby avoiding a contradiction. Further examination of

Table 2 reveals that state 3 is persisting at state 4. Hence, the memory element W_{34} is assigned to state 3. The state matrix for example 2.1 may now be written in output equation form as:

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & W_{12} \\ 1 & 0 & 0 & 0 \\ 0 & W_{34} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \end{bmatrix} \tag{4}$$

Equations (4) give all of the required output equations to sustain the desired sequential action only if the shut-off memory elements are switched at the proper times. W_{12} must be in the "set" position in order to formulate the state signal 1; therefore, it may be set prior to its state. In this case, W_{12} is set by the state signal 4 which is X_2. W_{12} must be reset either prior to, or by, state 2. Since the previous state is the persisting state, its signal may not be used to reset itself. Therefore, the contradicting state must be used to shut-off or reset the memory element. Thus, the switching conditions for W_{12} may be shown as follows:

	Set	Reset
W_{12}	X_2 State 4	X_1 State 2

The notation adopted for subscripting the W elements is quite fortunate since the subscripts of W_{12} (read W one, two) give both the persisting and the contradicting states, respectively. The switching conditions may then be stated by simply observing the subscripts. For example, the memory element W_{34} is set prior to the persistent state 3 and is reset by the contradicting state 4. Thus, the complete logic specifications for example 2.1 are:

Output Equations:

$$
\begin{aligned}
Z_1 &= \bar{X}_2 \; W_{12} \\
\bar{Z}_1 &= X_1 \\
Z_2 &= \bar{X}_1 \; W_{34} \\
\bar{Z}_2 &= X_2
\end{aligned}
\tag{5}
$$

Switching Conditions:

	Set	Reset
W_{12}	X_2	X_1
W_{34}	X_1	X_2

Before going any further into synthesis procedures, it might be helpful to demonstrate the circuit implementation for this problem. If the circuit shown in Figure 2 is not self-explanatory, the reader is advised to consult a text on fluid circuits. Refer to Figure 1 for the circuit implied by the boxes representing the cylinders.

Persistent states always occur when two events involving one cylinder are consecutive; however, the same problem arises anytime there is a possibility for a contradiction. This problem may best be illustrated by an example. Consider for example 2.2 the three cylinder sequence Z_1 , Z_2 ,

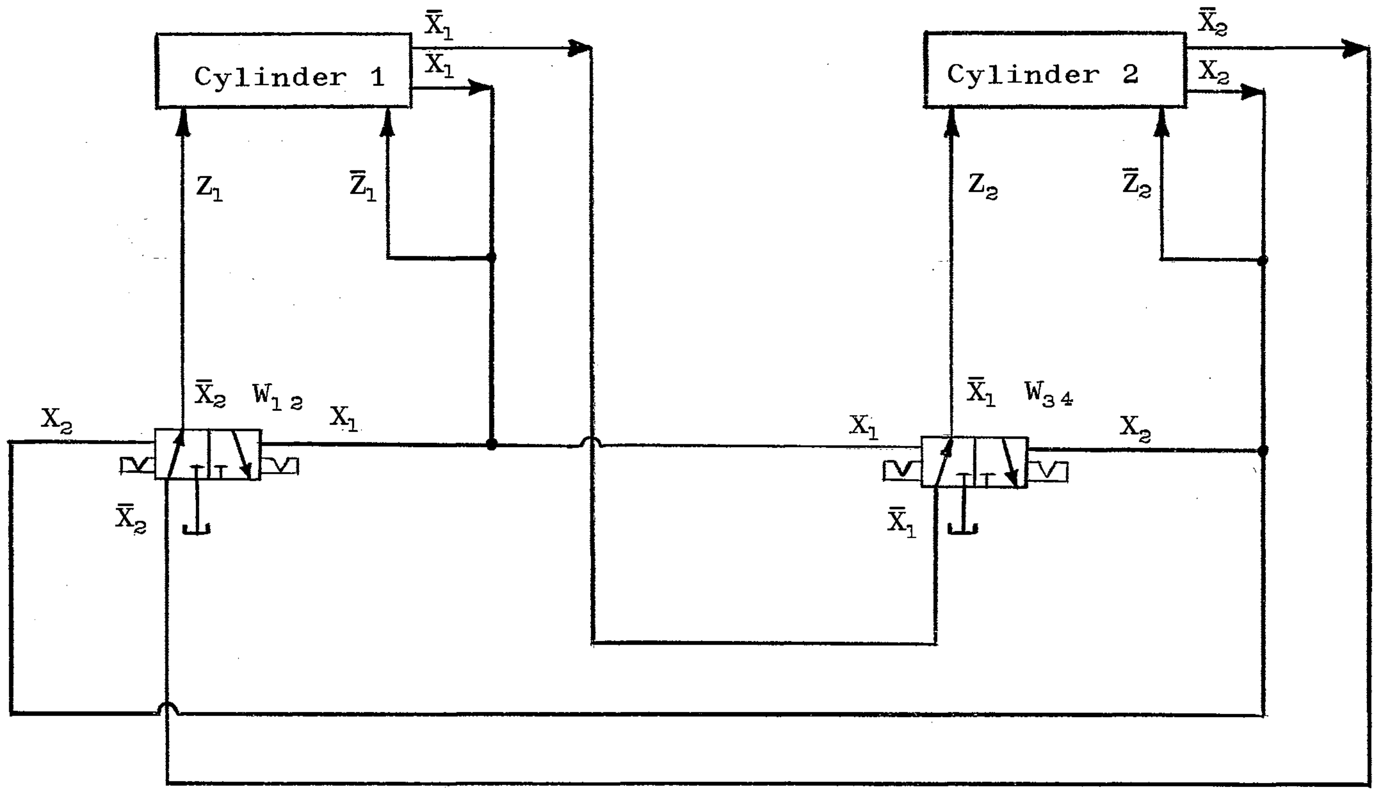


Figure 2. Synthesized Hydraulic Circuit for Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$

Z_3, $\bar{Z}_3$, $\bar{Z}_2$, $\bar{Z}_1$. Following through the sequence, it is found that Z_2 is caused by X_1 in event two. Later, in event five, $\bar{Z}_2$ is required. However, since cylinder one is not retracted between events two and five, the signal X_1 from event two is still on. Thus, state 2 is a persisting state contradicting event 5. A shut-off memory, W_{25}, is required to modify state 2. Since states 2 and 5 are not consecutive, the shut-off memory element W_{25} can be reset just prior to the contradiction, state 5, rather than by the contradiction itself. This is usually more desirable; however, the particular circuit hardware might dictate otherwise.

There are three other persistent states in this sequence. The reader is encouraged to develop the state matrix for this sequence and verify the memory assignment and switching conditions represented by Equations (6). The output matrix for the sequence Z_1, Z_2, Z_3, $\bar{Z}_3$, $\bar{Z}_2$, $\bar{Z}_1$ is:

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \\ Z_3 \\ \bar{Z}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & W_{61} & 0 & 0 \\ W_{25} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & W_{52} \\ 0 & 0 & W_{34} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \\ X_3 \\ \bar{X}_3 \end{bmatrix} \qquad (6)$$

where the switching conditions are:

	Set	Reset
W_{25}	$\bar{X}_1$	X_3
W_{34}	$X_1 \; W_{25}$	X_3
W_{61}	$\bar{X}_3 \; W_{52}$	$\bar{X}_1$
W_{52}	X_3	$\bar{X}_1$

When determining persistent states, it is convenient to <u>partition</u> the state matrix according to outputs. The two rows for Z_1 and $\bar{Z}_1$ represent the output partition one, etc. The two columns for X_2 and $\bar{X}_2$ are input partition two, etc. With the matrix partitioned in this manner, a systematic method for determining persisting states can be defined. This method requires the individual investigation of each output partition. Starting with the first entry in an output partition, each state is checked by investigating the next entry in the output partition. This next entry is always in the complementary half of the output partition. These two states are always contradictory if they are consecutive and are not within a <u>diagonal partition</u>. A diagonal partition is the four entry square formed by the intersection of an output partition and its corresponding input partition. This square will always be on the diagonal of the matrix. Two consecutive entries in a diagonal partition are not contradictory since the first event turns itself off by the next entry. For the same reason, the event in the output partition following an entry in its diagonal partition is not contradictory. States not covered by the above rule must be examined by applying the following general rule. If the next entry in the output partition is not consecutive and is not within a diagonal partition, then the complementary event of the state immediately preceding the first entry in the output partition must occur before the next entry in the output partition. In other words, the signal that initiated the first entry in the output partition must be negated or turned-off prior to the next entry in the output partition, otherwise the first entry will be a persisting state. The application of these rules is discussed in detail for the example given in the Procedure Summary.

Memory Assignment

In most sequences, an element is cycled more than once, thus causing an input signal to appear more than once during the sequence. Often, this input signal will initiate a different event each time it appears. In order to determine which event is called for when that input appears, memory of previous events is required.

Consider for example 2.3 the sequence Z_1, Z_2, $\bar{Z}_1$, $\bar{Z}_2$, Z_1, $\bar{Z}_1$. The state matrix shown in Table III is constructed by entering the state numbers as previously discussed. A close examination of this sequence reveals that state 5 is a persistent state. The element W_{56} is assigned to state 5 to prevent the contradiction at state 6. This element is then entered into the output matrix for state 5. This is the only persistent state in this sequence.

TABLE III

THE STATE MATRIX RELATION FOR
Z_1 , Z_2 , $\bar{Z}_1$, $\bar{Z}_2$, Z_1 , $\bar{Z}_1$

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} \sim \begin{array}{|c|c|c|c|} \hline & 1 & & 5 \\ \hline 6 & & 3 & \\ \hline 2 & & & \\ \hline & 4 & & \\ \hline \end{array} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ X_2 \end{bmatrix}$$

Columns one and two of Table III contain more than one stable state per column. The states 6 and 2 in column one indicate that there are two separate outputs initiated by the input X_1 . One time the input signal X_1 initiates the output Z_2 ; the next time X_1 appears, the output $\bar{Z}_1$ is desired. In order to distinguish between these states, a memory element is assigned to one of these states and its complement is assigned to the other. For instance, the memory element Y_{26} is assigned to state 2 and $\bar{Y}_{26}$ is assigned state 6. In accordance with the W elements, the Y elements are subscripted to denote their associated states. The element Y_{26} is used to distinguish between states 2 and 6, and is set prior to state 2 and is reset before 6. A similar condition exists between states 1 and 4. The memory element Y_{14} is used to make each of these states unique.

The output matrix is constructed by entering all of the Y elements to distinguish between common input states. The W elements are entered at their persisting states and a "1" is entered for any stable state which does not require memory. A "0" is entered elsewhere. The resulting output matrix for example 2.3 is given by Equation (7).

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix} = \begin{bmatrix} 0 & Y_{14} & 0 & W_{56} \\ \bar{Y}_{26} & 0 & 1 & 0 \\ Y_{26} & 0 & 0 & 0 \\ 0 & \bar{Y}_{14} & 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ \bar{X}_1 \\ X_2 \\ \bar{X}_2 \end{bmatrix} \tag{7}$$

Written out, these equations are:

$$Z_1 = \bar{X}_1 Y_{14} + \bar{X}_2 W_{56}$$

$$\bar{Z}_1 = X_1 \bar{Y}_{26} + X_2$$

$$Z_2 = X_1 Y_{26}$$

$$\bar{Z}_2 = \bar{X}_1 \bar{Y}_{14}$$

The switching conditions are:

	Set	Reset
Y_{26}	$\bar{X}_1 Y_{14}$	$\bar{X}_2 W_{56}$
Y_{14}	$X_1 \bar{Y}_{26}$	X_2
W_{56}	$\bar{X}_1 \bar{Y}_{14}$	$X_1 \bar{Y}_{26}$

Figure 3 is a hydraulic implementation of the logic circuit for this sequence. The passive memory effect is

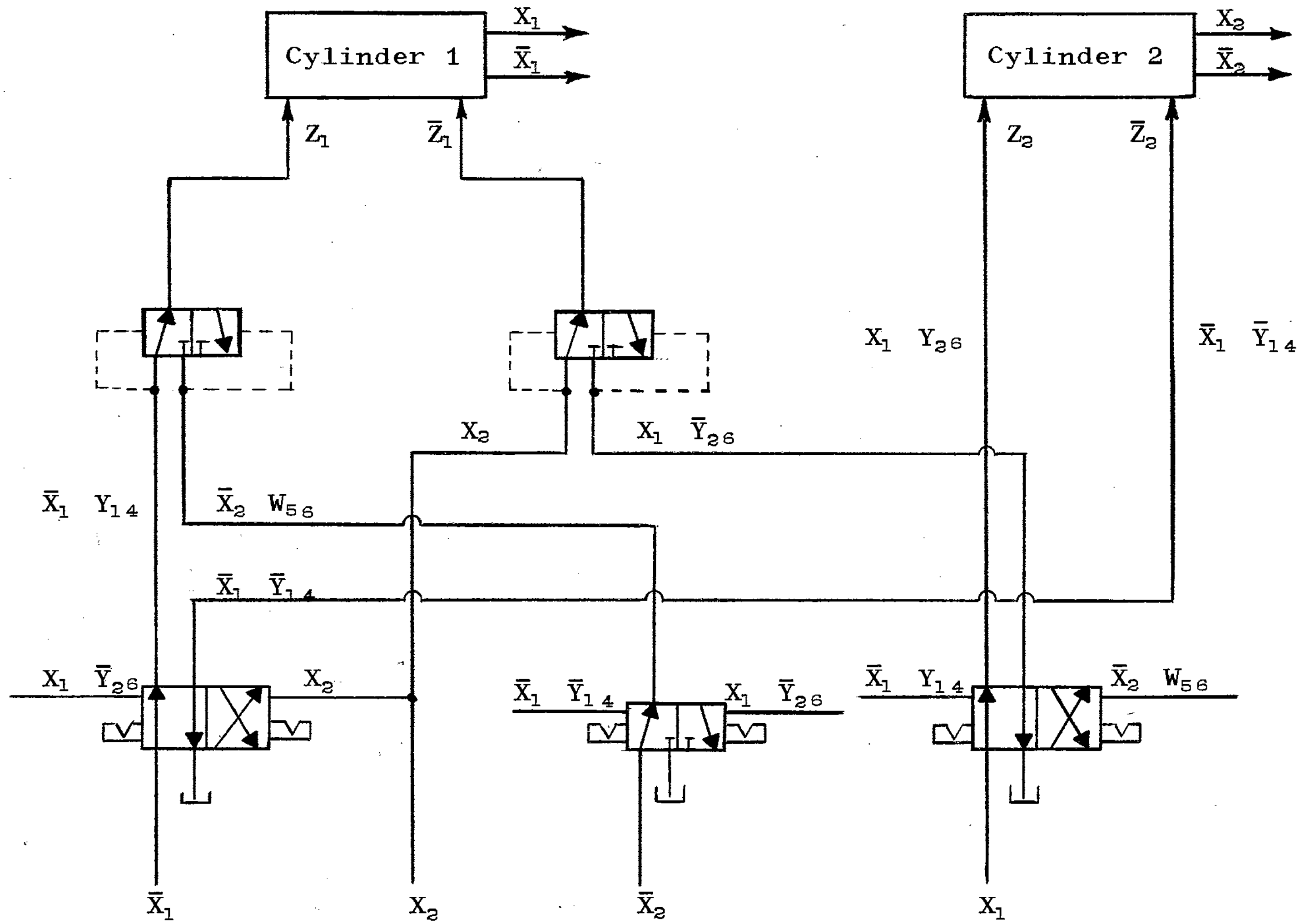


Figure 3. Hydraulic Implementation for Z_1 , Z_2 , $\bar{Z}_1$, $\bar{Z}_2$, Z_1 , $\bar{Z}_1$

utilized in this circuit to reduce circuit complexity and hardware. At this point, the reader should refer to Appendix A for a complete discussion of the passive memory effect, assignment, and implementation for hydraulic and fluidic circuits.

Counting Sequences

Counting sequences are characterized by their repetitious cycling of outputs. For example, a 2,2,1 counter cycles (i.e., extends, retracts) the first element twice, the second twice, and the third once and then repeats the sequence. Counting sequences are handled in exactly the same manner as any other automatic circuit; however, their uniqueness deserves special mention.

In synthesizing this circuit, the usual formal notation is dropped and the simplified approach is introduced. The first simplification is the omission of the output and input vectors. Instead of writing a formal state matrix relation, the rows and columns of the state matrix are labeled corresponding to their associated vectors. With this simplified approach, the state numbers representing the sequence are entered into the matrix as usual. The required memory elements are then assigned adjacent to their state number eliminating the need for rewriting the state matrix into the output matrix form. The output equations are written directly from the completed state matrix.

The first step in synthesizing the equations for example 2.4 is to enter the state numbers representing the sequence into the state matrix as shown in Table IV. This sequence is written as Z_1, $\bar{Z}_1$, Z_1, $\bar{Z}_1$, Z_2, $\bar{Z}_2$, Z_2, $\bar{Z}_2$, Z_3, $\bar{Z}_3$.

TABLE IV

THE STATE MATRIX FOR A 2,2,1 COUNTER

	X_1	$\bar{X}_1$	X_2	$\bar{X}_2$	X_3	$\bar{X}_3$
Z_1		3 Y_{35}				1 W_{12}
$\bar{Z}_1$	2 Y_{24} 4 $\bar{Y}_{24}$					
Z_2		5 $\bar{Y}_{35}$ W_{56}		7 Y_{79}		
$\bar{Z}_2$			6 Y_{68} 8 $\bar{Y}_{68}$			
Z_3				9 $\bar{Y}_{79}$ W_{910}		
$\bar{Z}_3$					10	

The next step is the determination of the existence of any persistent states. Applying the rules from page 18 to the matrix under consideration, it is found that states 1 and 2 are contradictory since they are consecutive entries within the same output partition and different input partitions. States 2 and 3 and 3 and 4 are both within diagonal partitions and, thus, are not contradictory. Since state 4 is in a diagonal partition, there is no contradiction between states 4 and 1. Entries like 5 and 6 in the second partition and 9 and 10 in the third partition are contradictory. By similar application of these rules, it can be seen that these are the only three contradictions in this sequence. The shut-off memories (W elements) are now entered into Table IV adjacent to their corresponding persistent states (e.g., W_{12} at 1, W_{56} at 5, and W_{910} at 9).

The next step of the procedure is the assignment of input memory elements (Y elements). Here, the rule is simple: whenever there is more than one state in a column, a secondary memory state must be assigned to make each state in the column unique. In Table IV there are four such columns requiring memory. The memory elements Y_{24}, Y_{35}, Y_{68}, Y_{79} are assigned to their corresponding states in accordance to Appendix A.

The last step in the procedure is the specification of the output equations and switching conditions. The output equations are written directly from the state matrix in the same manner as initially discussed. The switching conditions are determined directly from the element subscripts. The complete logical specifications for example 2.4 are given below:

Output equations:

$$
\begin{aligned}
Z_1 &= \bar{X}_1 \; Y_{35} + \bar{X}_3 \; W_{12} \\
\bar{Z}_1 &= X_1 \; Y_{24} + X_1 \; \bar{Y}_{24} = X_1 \\
Z_2 &= \bar{X}_1 \; \bar{Y}_{35} \; W_{56} + \bar{X}_2 \; Y_{79} \\
\bar{Z}_2 &= X_2 \; Y_{68} + X_2 \; \bar{Y}_{68} = X_2 \\
Z_3 &= \bar{X}_2 \; \bar{Y}_{79} \; W_{910} \\
\bar{Z}_3 &= X_3
\end{aligned}
\tag{8}
$$

Switching conditions:

	Set	Reset
Y_{24}	$\bar{X}_3 \; W_{12}$	$\bar{X}_1 \; Y_{35}$
Y_{35}	$X_1 \; Y_{24}$	$X_1 \; \bar{Y}_{24}$
Y_{68}	$\bar{X}_1 \; \bar{Y}_{35} \; W_{56}$	$\bar{X}_2 \; Y_{79}$
Y_{79}	$X_2 \; Y_{68}$	$X_2 \; \bar{Y}_{68}$
Y_{12}	X_3	$X_1 \; Y_{24}$
W_{56}	$X_1 \; \bar{Y}_{24}$	$X_2 \; Y_{68}$
W_{910}	$X_3 \; \bar{Y}_{68}$	X_3

Notice that the equations for $\bar{Z}_1$ and $\bar{Z}_2$ both reduce, thereby eliminating a memory element. This does not imply that these memory elements are not required. These two signals (states 2 and 4) must be unique since they are used to switch other memories to prepare the proper transition paths.

Procedure Summary

The procedure for the synthesis of feedback sequential digital control circuits is summarized by the following four steps:

1. Enter State Numbers - Write down the specified sequence and number each event in the sequence. Starting with the first event, sequentially enter the state numbers into the state matrix in the row corresponding to the desired output and the column corresponding to the previous event.

2. Correct Persistent States - Whenever a state signal remains on to form an extend-retract contradiction, the persistent state signal must be modified by a W memory element.

3. Assign Memory States - Whenever there is more than one state in any column of the state matrix, memory states are required to make each of these states unique.

4. Determine Output and Switching Conditions - The digital output equations are obtained from the state matrix by replacing each state number by a logical "1" and all blank entries in the matrix by "0" and then multiplying the matrix. The switching conditions are determined from the memory subscripts.

The following example encompasses all of the defined rules for the synthesis of feedback sequential logic circuits and is worked in detail as a final illustration of this synthesis procedure. The entire problem is presented on page 31 and the procedure is discussed in detail below.

First of all, the sequence is specified and written with state numbers below it, as shown on page 31. This sequence is then entered into the state matrix by placing the state numbers in the row of the desired output and the column of the present input. For example, the state number 1 is entered in the Z_1 row and the $\bar{X}_3$ column since the first event, Z_1, is initiated by the previous event which is the retraction of cylinder three. The next event is the retraction of cylinder one; accordingly, state 2 is located in the $\bar{Z}_1$ row and X_1 column. The remainder of the sequence is entered into the state matrix in the same fashion.

The next step of the procedure requires the investigation of each output partition for the possibility of persistent states. The first partition is investigated by starting with state 1. The next entry in this partition is state 2. Since this is a consecutive entry not within a diagonal partition, states 1 and 2 are contradictory and must be corrected by modifying the persistent state (state 1) with the memory element W_{12}. W_{12} is entered in the matrix adjacent to state number 1. The next entry in this partition is state 3. This entry, as well as the next, is within a diagonal partition and is not contradictory. The next entry in partition one after state 4 is state 7. Since state 4 is within a diagonal partition, its initiating signal is negated prior to the next entry (state 7). States 7 and 9 form a contradiction since event 6 has not been negated before state 9. Accordingly, the memory element W_{79} is entered by state 7. The final entry in partition one is state 1. Since event 8 is not negated before state 1, W_{91} is placed beside state 9 to correct this contradiction.

The next partition has only two states (5 and 8). It can be seen that these states do not form a contradiction since event 4 is negated by event 7. Similarly, state 8 is not persisting at state 5.

The possible contradiction in partition three (6 and 10) is eliminated since event 5 is negated by event 8. Thus, the signal causing state 6 is turned off before state 10. The state prior to state 10 (state 9) is negated before state 6 eliminating this possible contradiction.

Now that all persistent states have been corrected, the next step in the procedure is the assignment of any required memory states. Column one of the state matrix contains three states (2, 4, and 8). Each of these states must be made unique by modifying the states with the proper memory state. This is done by placing Y_{28} Y_{24} at state 2, Y_{28} $\overline{Y}_{24}$ at state 4 and $\overline{Y}_{28}$ at state 8. (Notice the double subscript notation.) Column two also contains three states, 3, 5, and 10, and the memory elements Y_{35} and Y_{310} are assigned accordingly. There are no other columns requiring memory.

The final step of the procedure is the specification of output and switching conditions. The output equations are obtained by mentally replacing each state number by the logical "1" and multiplying the matrix by the input vector.

The switching conditions for the memory elements are obtained from the element subscripts. For example, W_{12} is set prior to state 1 by state 10 and is reset by state 2. W_{79} is set by state 6 and is reset by state 8. Y_{24} is set by state 1 and reset by state 3, etc.

This problem is worked to completion on the following page.

EXAMPLE PROBLEM

Sequence:	1	$\bar{1}$	1	$\bar{1}$	2	3	1	$\bar{2}$	$\bar{1}$	$\bar{3}$
State Nos:	1	2	3	4	5	6	7	8	9	10

	X_1	$\bar{X}_1$	X_2	$\bar{X}_2$	X_3	$\bar{X}_3$
Z_1		3 Y_{310} Y_{35}			7 W_{79}	1 W_{12}
$\bar{Z}_1$	2 Y_{28} Y_{24} 4 Y_{28} $\bar{Y}_{24}$			9 W_{91}		
Z_2		5 Y_{310} $\bar{Y}_{35}$				
$\bar{Z}_2$	8 $\bar{Y}_{28}$					
Z_3			6			
$\bar{Z}_3$		10 $\bar{Y}_{310}$				

Output Equations:

$$
\begin{aligned}
Z_1 &= \bar{X}_1 Y_{310} Y_{35} + X_3 W_{79} + \bar{X}_3 W_{12} \\
\bar{Z}_1 &= X_1 Y_{28} Y_{24} + X_1 Y_{28} \bar{Y}_{24} + \bar{X}_2 W_{91} = X_1 Y_{28} + \bar{X}_2 W_{91} \\
Z_2 &= \bar{X}_1 Y_{310} \bar{Y}_{35} \\
\bar{Z}_2 &= X_1 \bar{Y}_{28} \\
Z_3 &= X_2 \\
\bar{Z}_3 &= \bar{X}_1 \bar{Y}_{310}
\end{aligned}
\tag{9}
$$

Switching Conditions:

	Set	Reset
Y_{24}	$\bar{X}_3$ W_{12}	$\bar{X}_1$ Y_{310} Y_{35}
Y_{28}	$\bar{X}_3$ W_{12}	X_3 W_{79}
Y_{35}	X_1 Y_{28} Y_{24}	X_1 Y_{28} $\bar{Y}_{24}$
Y_{310}	X_1 Y_{28} Y_{24}	$\bar{X}_2$ W_{91}
W_{91}	X_1 $\bar{Y}_{28}$	$\bar{X}_1$ $\bar{Y}_{310}$
W_{79}	X_2	X_1 $\bar{Y}_{28}$
W_{12}	$\bar{X}_1$ $\bar{Y}_{310}$	X_1 X_{28} Y_{24}

The hydraulic implementation for this circuit is shown in Figure 4. In this circuit, the actual switching signals have been replaced by the notation S_{24}, R_{35}, etc., where S_{24} denotes the "set" signal for Y_{24} from the above switching conditions.

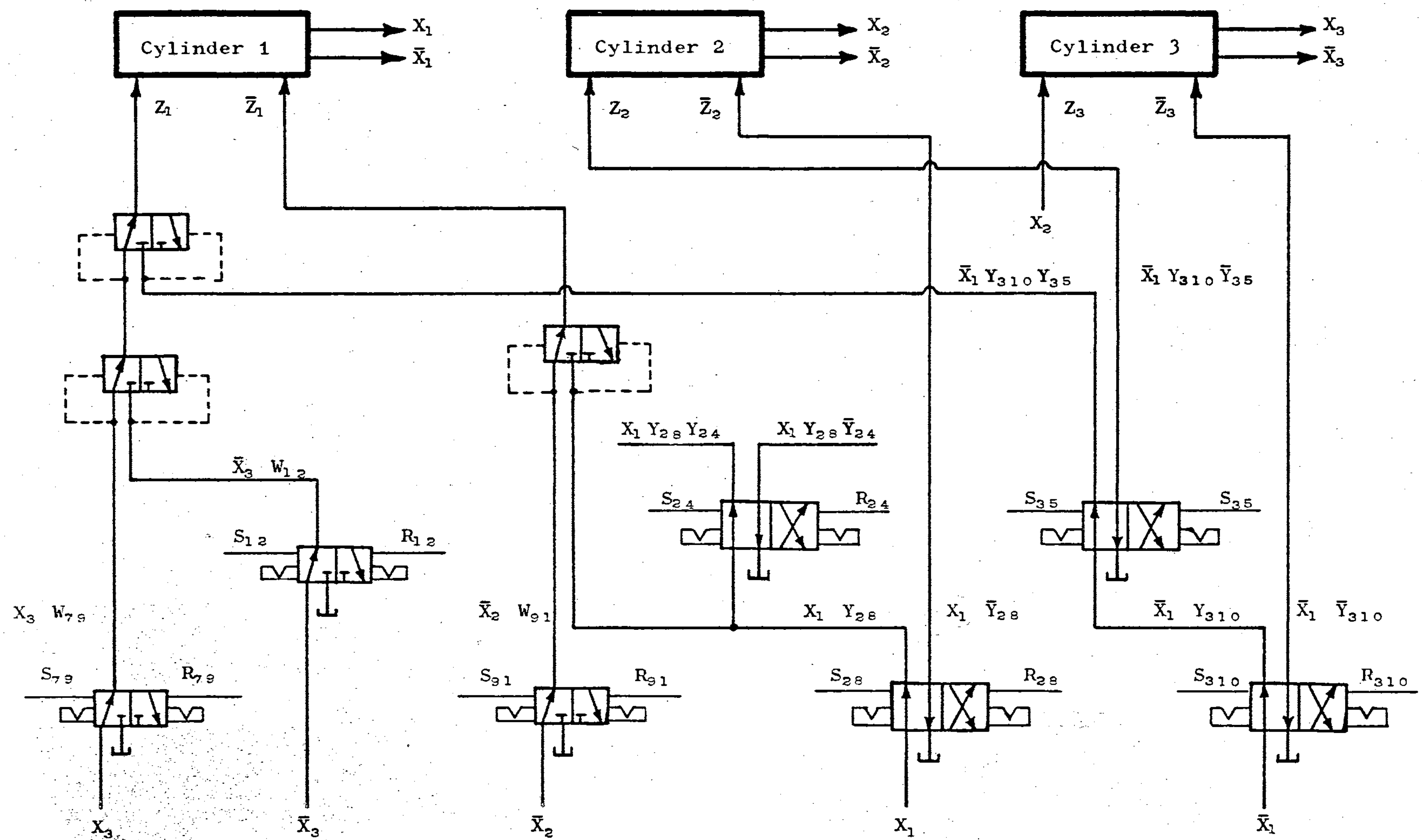


Figure 4. Hydraulic Implementation for 1, $\bar{1}$, 1, $\bar{1}$, 2, 3, 1, $\bar{2}$, $\bar{1}$, $\bar{3}$

CHAPTER III

THE STATE MATRIX SYNTHESIS PROCEDURE FOR RANDOM INPUT CIRCUITS

Unlike feedback sequential circuits, random input circuits do not anticipate the next input; consequently, every possible input change must be considered. An example of this type of circuit is the secret combination lock in which only one sequence of input changes will result in the proper output (i.e., the opening of the lock). Other sequences might result in different outputs, return to starting position, or many other conceivable situations. In any event, the response to all input change possibilities from any state in the sequence must be specified before a circuit to perform the required logic can be synthesized.

The Primitive Flow Table

The synthesis of a circuit to perform certain logic sequences must proceed from the word statement of the possible inputs and the desired response to input changes. For every input change, two things must be specified: the resulting output and the desired transition paths from that state. These specifications are most conveniently represented by the information table termed the Primitive Flow Table.

The primitive flow table contains the complete logic specifications for a problem and is arranged as follows. The columns of the table indicate all of the possible input combinations. These input states are usually labeled above each column according to the Gray code (one variable change between columns). Each row of this table represents the state of the logic system and its corresponding output, Z. Numbers with parentheses around them indicate stable states of the circuit and the unparenthesized numbers show the possible transition paths from one stable state to another.

As example 3.1, consider the primitive flow table shown by Table V. This example has two inputs, X_1 and X_2, and one output, Z_1. The table indicates that the logic circuit must provide a path from state (1) to state (2) when the input changes from "00" to "10" as indicated by the transition path numbered 2 in the first row. Also, the circuit must return from (2) to (1) by the path indicated in the second row, first column. Notice that no transition path is shown from input "00", state (1), to input "11", since this would require two inputs to be changed at exactly the same instant, which is highly improbable.

TABLE V

PRIMITIVE FLOW TABLE FOR EXAMPLE 3.1

$X_1 X_2$ 00	10	11	01	Z_1
(1)	2	–	3	0
1	(2)	4	–	0
1	–	4	(3)	0
–	2	(4)	3	1

In Table V, the output Z_1 results when both inputs are actuated by either the path from state 2 or 3. As is the case with this example, the primitive flow table should specify every possible transition path and should form a closed loop in that there is a path back to the origin or any other state. The above example is extremely simple and requires no memory. When the sequences get larger and inputs are cycled, the need for memory arises as is shown in the next example.

Consider for example 3.2 the primary sequence 00, 10, 11, 01, 11, 10, which results in the output Z_1. The sequence 00, 01, 11 results in the Z_2 output. All other possible sequences are considered and the transition paths are shown in the completed primitive flow table, Table VI.

TABLE VI

PRIMITIVE FLOW TABLE FOR EXAMPLE 3.2

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
–	2	(3)	4	0	0
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0
1	–	8	(7)	0	0
–	9	(8)	7	0	1
1	(9)	3	–	0	0

Before synthesizing a circuit to perform the indicated logic of Table VI, it is advantageous, although not completely necessary, to administer two additional steps to the primitive flow table. First of all, the primitive flow table should be checked for the possibility of <u>redundant states</u>. Two stable states are said to be redundant if and only if they have the same input state, the same output state, and the same or equivalent transition paths. For example, the states (2) and (9) in Table VI are redundant since they have the same input (they are in the same column), the same output ($\bar{Z}_1$ $\bar{Z}_2$), and the same transition paths (1 and 3). For this reason, the row containing state (9) may be completely removed and all of the transition paths 9 may be replaced with the path indicator 2. There are no more redundancies in this table and the resulting flow table is termed the reduced primitive flow table.

Another advantageous operation on this flow table is the transformation to the canonical flow table. This operation is not completely necessary for the purposes of this chapter, so the definition and detailed discussion of it is deferred until Chapter IV. Briefly though, the basic concept is to order the states according to systematic input changes. The canonical flow table for the problem under consideration (which includes the above mentioned reduction) is shown in Table VII.

TABLE VII

CANONICAL FLOW TABLE FOR EXAMPLE 3.2

$X_1\ X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	3	0	0
1	(2)	4	–	0	0
1	–	5	(3)	0	0
–	2	(4)	6	0	0
–	2	(5)	3	0	1
1	–	7	(6)	0	0
–	8	(7)	6	0	0
1	(8)	4	–	1	0

Formal Matrix Representation

Once a problem has been completely specified and the canonical flow table has been derived, the next step is the synthesis of circuit equations to perform the required logic. This synthesis can be reduced to the determination of a unique matrix $[M]$ satisfying the relation.

$$[Z] = [M]\,[X]$$

This is a statement that the outputs $[Z]$ are related to the inputs $[X]$ and previous events. The matrix $[M]$ provides this relationship and contains memory information which defines the present state. The only difference between this matrix relation and the one used for the feedback sequential circuits is the input and output vectors used. In feedback sequential circuit synthesis, the changed input and the changed output vectors are used. For random input circuit synthesis, the input vector contains the total input state (present state of all inputs) and the output vector represents the continuous output state (present state of each output) rather than the change output.

As a first step toward constructing this matrix, the state numbers from the canonical flow table are entered into each of the output partitions. States with an output of Z_i are entered in the top half of the i^{th} output partition and states with the $\bar{Z}_i$ output are entered in the bottom half. This determines the rows in which states are entered. To determine the proper entry column, recall from the rules of matrix multiplication that each column in the matrix is multiplied only by a corresponding row of the input vector $\begin{bmatrix} X \end{bmatrix}$. Thus, a column of the matrix represents events associated with only one input state. Hence, state numbers are entered in the proper row of the output partition and in the column associated with that input state.

To illustrate the state matrix synthesis concept, consider example 3.1 as represented by Table V. This primitive flow table is entered into the state matrix by entering the stable state numbers in the row of the individual output and the column of the present input similar to the way it was done in Chapter II. This matrix is given by Table VIII.

TABLE VIII

THE STATE MATRIX RELATION FOR EXAMPLE 3.1

$$\begin{bmatrix} Z_1 \\ \bar{Z}_1 \end{bmatrix} \sim \begin{array}{|c|c|c|c|} \hline & & 4 & \\ \hline 1 & 2 & & 3 \\ \hline \end{array} \quad \begin{array}{c} X_1 X_2 \\ \begin{bmatrix} 00 \\ 10 \\ 11 \\ 01 \end{bmatrix} \end{array}$$

To obtain the output equation, replace every state number by the logical "1" and place a "0" elsewhere. Multiplying the matrix yields the result:

$$Z_1 = X_1 X_2 \tag{10}$$

The above example illustrates the basic concept of circuit synthesis using state matrices. This problem did not require memory; a more general problem requiring memory is discussed below.

As another example of circuit synthesis, consider example 3.2 represented by the canonical flow table given in Table VIII. The state numbers are entered into the matrix as described above and the result is termed the <u>state matrix relation</u>. See Table IX.

TABLE IX

THE STATE MATRIX RELATION FOR EXAMPLE 3.2

Z_1		8		
$\bar{Z}_1$	1	2	4 5 7	3 6
Z_2			5	
$\bar{Z}_2$	1	2 8	4 7	3 6

$$\sim \qquad \begin{array}{c} X_1 X_2 \\ \begin{bmatrix} 00 \\ 10 \\ 11 \\ 01 \end{bmatrix} \end{array}$$

Memory Assignment

As can be seen from Table IX, the only time the output

Z_1 appears is state 8. Since state 8 is associated with the input "10", one would be tempted to state that the output Z_1 is equal to $X_1\bar{X}_2$. However, this is not the case since state 2 also has the input "10" but does not have the output Z_1. Thus, some method to distinguish between states 2 and 8 is required. This is most conveniently done by assigning a memory state at both states. If a memory element was in the "set" position for 2 and in the "reset" position for 8, then these two states would be a unique combination of the input and memory states. This memory element may be represented by placing Y_{28} adjacent to every 2 in Table IX and its logical complement $\bar{Y}_{28}$ by states 8. This double subscript notation implies that the memory element Y_{28} is used to distinguish between states 2 and 8 and is set prior to 2 and is reset prior to 8.

A similar condition exists in column four. Although states 3 and 6 do not have differing outputs, they still required uniqueness since they have different transition paths and their signals are used to switch different memory elements. Therefore, the memory element Y_{36} is assigned to state 3 and its complement $\bar{Y}_{36}$ is assigned to state 6. States 4, 5, and 7 in column three also require memory to demand their uniqueness. The memory state Y_{47} Y_{45} is assigned to state 4, Y_{47} $\bar{Y}_{45}$ to state 5, and $\bar{Y}_{47}$ to state 7. Here again, the switching conditions are inferred by the subscripts. At this point, the reader should refer to

Appendix A for further information concerning the passive memory.

The matrix shown in Table X has all of the above memory modifications. Now, each state in this matrix has a unique representation.

TABLE X

THE UNIQUE STATE MATRIX RELATION
FOR EXAMPLE 3.2

					$X_1 X_2$
Z_1		8 $\bar{Y}_{28}$			00
$\bar{Z}_1$	1	2 Y_{28}	4 Y_{47} Y_{45} 5 Y_{47} $\bar{Y}_{45}$ 7 $\bar{Y}_{47}$	3 Y_{36} 6 $\bar{Y}_{36}$	10
Z_2			5 Y_{47} $\bar{Y}_{45}$		11
$\bar{Z}_2$	1	2 Y_{28} 8 $\bar{Y}_{28}$	4 Y_{47} Y_{45} 7 $\bar{Y}_{47}$	3 Y_{36} 6 $\bar{Y}_{36}$	01

Output and Switching Conditions

The purpose of any synthesis procedure is to give every state a unique signal representation. This signal (or variations upon this signal) is then used either as an output signal or as a switching signal for other memory elements. The above steps produce a state matrix in which every state is represented uniquely by a certain combination of input and memory states. The only remaining step is the specification of the output and switching conditions.

The output equations are obtained from the state matrix relation by replacing every state number designation in the state matrix by the logical "1" and by placing a logical "0" elsewhere. Once this substitution has been made, the resulting matrix is termed the <u>output matrix</u> since it now represents a set of digital equations rather than a state matrix relation. These equations can be rewritten in the individual equation form by multiplying the matrix by the input vector.

The final step in the synthesis procedure is the one which insures the proper circuit operation; this is the specification of when each memory element is to be switched to the proper state. These switching conditions are inferred from the element subscripts and the flow table. For example, the memory element Y_{ij} is set <u>prior</u> to the state "i" and is reset <u>prior</u> to state "j". This information is obtained from the flow table by observing the possible transition paths to states i and j. The corresponding previous states are to be used for switching signals.

As a specific example, the output and switching conditions for the problem given in Table X are as follows. The output matrix equation is:

$$
\begin{bmatrix} Z_1 \\ \bar{Z}_1 \\ Z_2 \\ \bar{Z}_2 \end{bmatrix}
=
\begin{bmatrix}
0 & \bar{Y}_{28} & 0 \quad 0 & 0 \\
1 & Y_{28} & Y_{47}\,Y_{45} + Y_{47}\,\bar{Y}_{45} + \bar{Y}_{47} & Y_{36} + \bar{Y}_{36} \\
0 & 0 & Y_{47}\,\bar{Y}_{45} & 0 \\
1 & Y_{28} + \bar{Y}_{28} & Y_{47}\,Y_{45} + \bar{Y}_{47} & Y_{36} + Y_{36}
\end{bmatrix}
\begin{array}{c} X_1 X_2 \\ \begin{bmatrix} 00 \\ 10 \\ 11 \\ 01 \end{bmatrix} \end{array}
\qquad (11a)
$$

Since the outputs Z_1 and $\bar{Z}_1$ are perfect complements, only the equations for Z_1 and Z_2 need to be specified. These are:

$$
\begin{aligned}
Z_1 &= X_1 \bar{X}_2 \ \bar{Y}_{28} \\
Z_2 &= X_1 X_2 \ Y_{47} \bar{Y}_{45}
\end{aligned}
\qquad (11b)
$$

The switching conditions as determined from the subscripts and the flow table (Table VII) are:

Y_{28}:

$$
\begin{aligned}
\text{Set} &= \text{States } 1 + 4 + 5 \\
&= \bar{X}_1 \bar{X}_2 + X_1 X_2 \ Y_{47} Y_{45} + X_1 X_2 \ Y_{47} \bar{Y}_{45} \\
&= \bar{X}_1 \bar{X}_2 + X_1 X_2 \ Y_{47} \\
\text{Reset} &= \text{State } 7 \\
&= X_1 X_2 \ \bar{Y}_{47}
\end{aligned}
$$

Y_{47}:

$$
\begin{aligned}
\text{Set} &= \text{States } 2 + 8 \\
&= X_1 \bar{X}_2 \ Y_{28} + X_1 \bar{X}_2 \ \bar{Y}_{28} \\
&= X_1 \bar{X}_2 \\
\text{Reset} &= \text{State } 6 \\
&= \bar{X}_1 X_2 \ \bar{Y}_{36}
\end{aligned}
$$

Y_{45}: Set $=$ States 2 + 8

$= X_1 \bar{X}_2$

Reset $=$ State 3

$= \bar{X}_1 X_2 \; Y_{36}$

Y_{36}: Set $=$ States 1 + 5

$= \bar{X}_1 \bar{X}_2 + X_1 X_2 \; Y_{47} \; \bar{Y}_{45}$

Reset $=$ States 4 + 7

$= X_1 X_2 \; Y_{47} Y_{45} + X_1 X_2 \; \bar{Y}_{47}$

In more compact notation, the switching conditions are:

	Set	Reset
Y_{28}	$00 + 11\ Y_{47}$	$11\ \bar{Y}_{47}$
Y_{47}	10	$01\ \bar{Y}_{36}$
Y_{45}	10	$01\ Y_{36}$
Y_{36}	$00 + 11\ Y_{47}\ \bar{Y}_{45}$	$11\ Y_{47}\ Y_{45} + 11\ \bar{Y}_{47}$

Procedure Summary

The state matrix synthesis procedure consists of the following four steps:

1. Develop Primitive Flow Table - From the word statement of the problem, construct a primitive flow table showing all possible input changes, all possible transitions, and the corresponding outputs. If desired, this flow table may then be transformed into the canonical flow table.
2. Form State Matrix - Enter the stable state numbers into the state matrix. Each state

number appears in every output partition under the proper column.

3. Assign Memory States - Whenever there is more than one stable state number in a column, make each state unique by assigning the appropriate memory state.

4. Determine Output and Switching Conditions - The output equations are obtained by replacing each state number by "1" and placing a "0" elsewhere and then multiplying the matrix. The output complement need not be specified. The switching conditions are determined from the element subscripts and previous events shown in the flow table.

As a final example of the state matrix synthesis procedure, example 3.3 is worked to completion on page 51, and each step is explained in detail below. The reader may refer to Appendix C for further example problems and their solutions.

Before working the final example, some of the formality of the method can be dropped and the shorthand notation introduced. First of all, the formal matrix representation is omitted and the rows and columns of the matrix itself are merely labeled according to their outputs and inputs. Next, the intermediate step of writing the output matrix is eliminated by mentally multiplying the matrix rather than rewriting it. As a matter of fact, the matrix representation itself can be eliminated by working directly with the primitive flow table once the reader is familiar with the technique. However, this step is not presented here.

Consider for example 3.3 a secret combination lock in which there is only one proper sequence of output actuations which will open the lock (output Z_1). Any deviation from this sequence sounds an alarm (output Z_2). The correct sequence is X_1, X_2, $\bar{X}_2$, X_2, $\bar{X}_1$; where X means actuate and hold, $\bar{X}$ means release. Even though a mistake sounds the alarm, there should be a path provided back to the origin. This primitive flow table is shown on page 51 and is not transformed into the canonical form.

Once the primitive flow table is developed, the next step is the formation of the corresponding state matrix. This is done by entering each state number in the column of the input state and the rows of the individual outputs. For the first output, all state numbers except state 6 are entered in the lower half of output partition one, since all of them have the $\bar{Z}_1$ output. State 6 is then entered into the Z_1 row of the state matrix. Next, states 1 through 6 are entered in partition two in the $\bar{Z}_2$ row and states 7 through 10 are entered in the Z_2 row. These two row partitions comprise the state matrix for this example.

The next step is the determination of memory requirements. To do this, each column, representing one combination of the inputs, is treated separately. Reference to the state matrix reveals that every column has multiple states and requires memory to make each state unique. Column one has two states, 1 and 8, requiring one memory element, Y_{18}. Y_{18} is thus entered beside every 1 in the matrix, and its complement $\bar{Y}_{18}$ is entered adjacent to states 8. Similarily, column two contains three states, 2, 4, and 9. Each of these states is made unique by assigning two memory elements, Y_{29} and Y_{24}, in accordance with Appendix A. Column three has three states and column four has two. Memory elements are assigned to these states in the same manner as above.

After the state matrix is formed and the memory requirements are entered adjacent to their respective states, the output equations are obtained by mentally replacing the state numbers with "1's" and then multiplying the matrix by the input vector. The output complements do not have to be specified. The output Z_1 appears at state 6 only. The Z_2 output appears at states 8, 9, 10, and 7.

The final step is the specification of the switching conditions; this step ensures proper circuit operation. If the double subscript notation is used to denote memory elements, the switching conditions are stated from knowledge of the subscripts and the flow table. The subscripts indicate when an element should be in the "set" or "reset" position and the flow table shows the possible transitions to these states. For example, Y_{18} is set by any state immediately preceding state 1 and is reset by states preceding state 8. From the flow table, it can be seen that the only transition path to only transition path to state 1 is from state 6. There are transition paths to state 8 from states 2, 4, 7, and 9. Thus, Y_{18} is set by state 6 and is reset by state 2, 4, 7, or 9. The element Y_{29} is set by state 1 or 8 and is reset by state 5 or 10. The remaining switching conditions are determined in the same fashion and the complete table or switching conditions is given below.

This problem is shown on the next page and the logic circuit schematic is shown in Figure 5.

TABLE XI

THE PRIMITIVE FLOW TABLE FOR EXAMPLE 3.3

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2		7	0	0
8	(2)	3		0	0
	4	(3)	7	0	0
8	(4)	5		0	0
	9	(5)	6	0	0
1		10	(6)	1	0
8		10	(7)	0	1
(8)	2		7	0	1
8	(9)	10		0	1
	9	(10)	7	0	1

The State Matrix:

	00	10	11	01
Z_1				6 Y_{67}
$\bar{Z}_1$	1 Y_{18} 8 $\bar{Y}_{18}$	2 Y_{29} Y_{24} 4 Y_{29} $\bar{Y}_{24}$ 9 $\bar{Y}_{29}$	3 Y_{310} Y_{35} 5 Y_{310} $\bar{Y}_{35}$ 10 $\bar{Y}_{310}$	7 $\bar{Y}_{67}$
Z_2	8 $\bar{Y}_{18}$	9 $\bar{Y}_{29}$	10 $\bar{Y}_{310}$	7 $\bar{Y}_{67}$
$\bar{Z}_2$	1 Y_{18}	2 Y_{29} Y_{24} 4 Y_{29} $\bar{Y}_{24}$	3 Y_{310} Y_{35} 5 Y_{310} $\bar{Y}_{35}$	6 Y_{67}

Output Equations:

$$Z_1 = \bar{X}_1 X_2 \ Y_{67}$$

$$Z_2 = \bar{X}_1 \bar{X}_2 \ \bar{Y}_{18} + X_1 \bar{X}_2 \ \bar{Y}_{29} + X_1 X_2 \ \bar{Y}_{310} + \bar{X}_1 X_2 \ \bar{Y}_{67} \tag{12}$$

Switching Conditions:

	Set	Reset
Y_{18}	$\bar{X}_1 X_2 \; Y_{67}$	$X_1 \bar{X}_2 + \bar{X}_1 X_2 \; \bar{Y}_{67}$
Y_{29}	$\bar{X}_1 \bar{X}_2$	$X_1 X_2 \; Y_{310} \bar{Y}_{35} + X_1 X_2 \; \bar{Y}_{310}$
Y_{24}	$\bar{X}_1 \bar{X}_2$	$X_1 X_2 \; Y_{310} Y_{35}$
Y_{310}	$X_1 \bar{X}_2 \; Y_{29} Y_{24}$	$\bar{X}_1 X_2 + X_1 \bar{X}_2 \; \bar{Y}_{29}$
Y_{35}	$X_1 \bar{X}_2 \; Y_{29} Y_{24}$	$X_1 \bar{X}_2 \; Y_{29} \bar{Y}_{24}$
Y_{67}	$X_1 X_2 \; Y_{310} \bar{Y}_{35}$	$\bar{X}_1 \bar{X}_2 + X_1 X_2 \; Y_{310} Y_{35} + X_1 X_2 \; \bar{Y}_{310}$

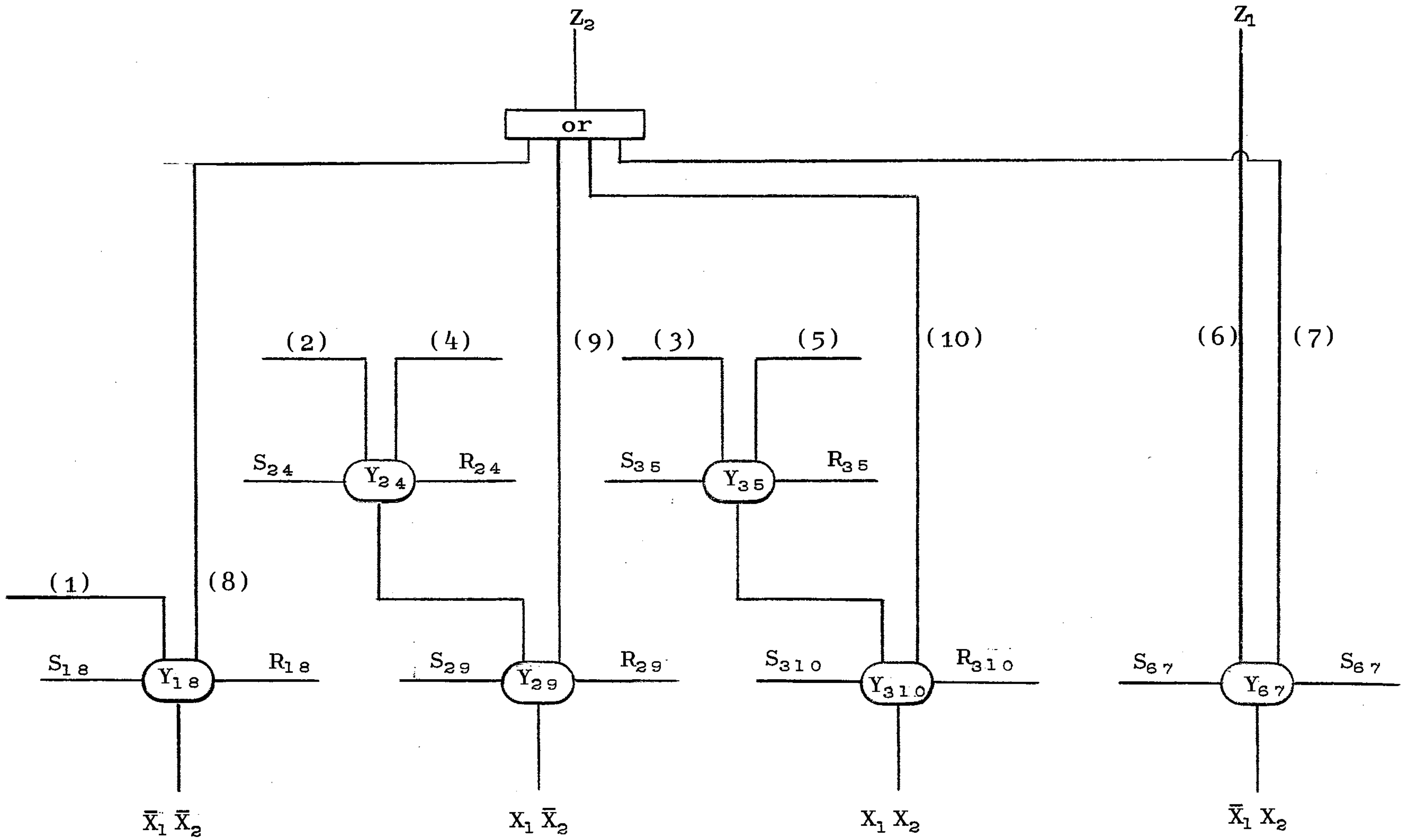


Figure 5. Logic Circuit for Example 3.3

CHAPTER IV

DIGITAL EQUATION SIMULATION AND THE CANONICAL FLOW TABLE

All synthesis procedures will produce valid equations for the representation of the specified logic when the procedure is executed correctly. However, some methods are not easily understood or require personal preference in certain steps. Often, intuitively designed circuits do not function properly or for some reason the circuit action needs to be analyzed. To do this, the implied equations of the circuit can be written.

Whether for verification or analysis, it is often necessary to check the system equations. For this reason, a systematic digital equation simulation method has been developed. This method involves the systematic excitation of the inputs to the equations to produce a primitive flow table. This simulated flow table representing the equations may then be compared to the desired circuit action to ascertain if the equations represent the required logic.

Once the simulated flow table is obtained, the task of comparing this table to the original flow table may be larger than the original task of verifying the equations if the state numbers do not coincide. For this reason, it is advantageous, if not mandatory, to define a standard format for flow tables. The canonical flow table defined in this chapter satisfies this requirement.

Digital Equation Simulation

The simulation technique presented here offers a systematic method for checking equations and in no way assumes prior knowledge of system response. The basic idea is to change one input from some base state and then observe the resulting output and memory states. If these output and memory states are different from any previously determined, then a new state is defined. If they are the same as some other state, then this new state is redundant and is replaced by its equivalent state. By extending this procedure, there finally results a closed flow table. The flow chart shown in Figure 6 illustrates the complete simulation method.

The method may best be explained by an example. Table XII illustrates the step-by-step development of the simulation discussed below. Consider the logic represented by the following equations as derived by the classical method:

$$\begin{aligned} Z_1 &= X_1 \bar{X}_2 \ \ \bar{Y}_1 Y_2 \\ Z_2 &= \bar{X}_1 X_2 \ \ Y_1 Y_2 \\ Y_1 &= S_1 + Y_1 \bar{R}_1 \\ Y_2 &= S_2 + Y_2 \bar{R}_2 \end{aligned} \tag{13}$$

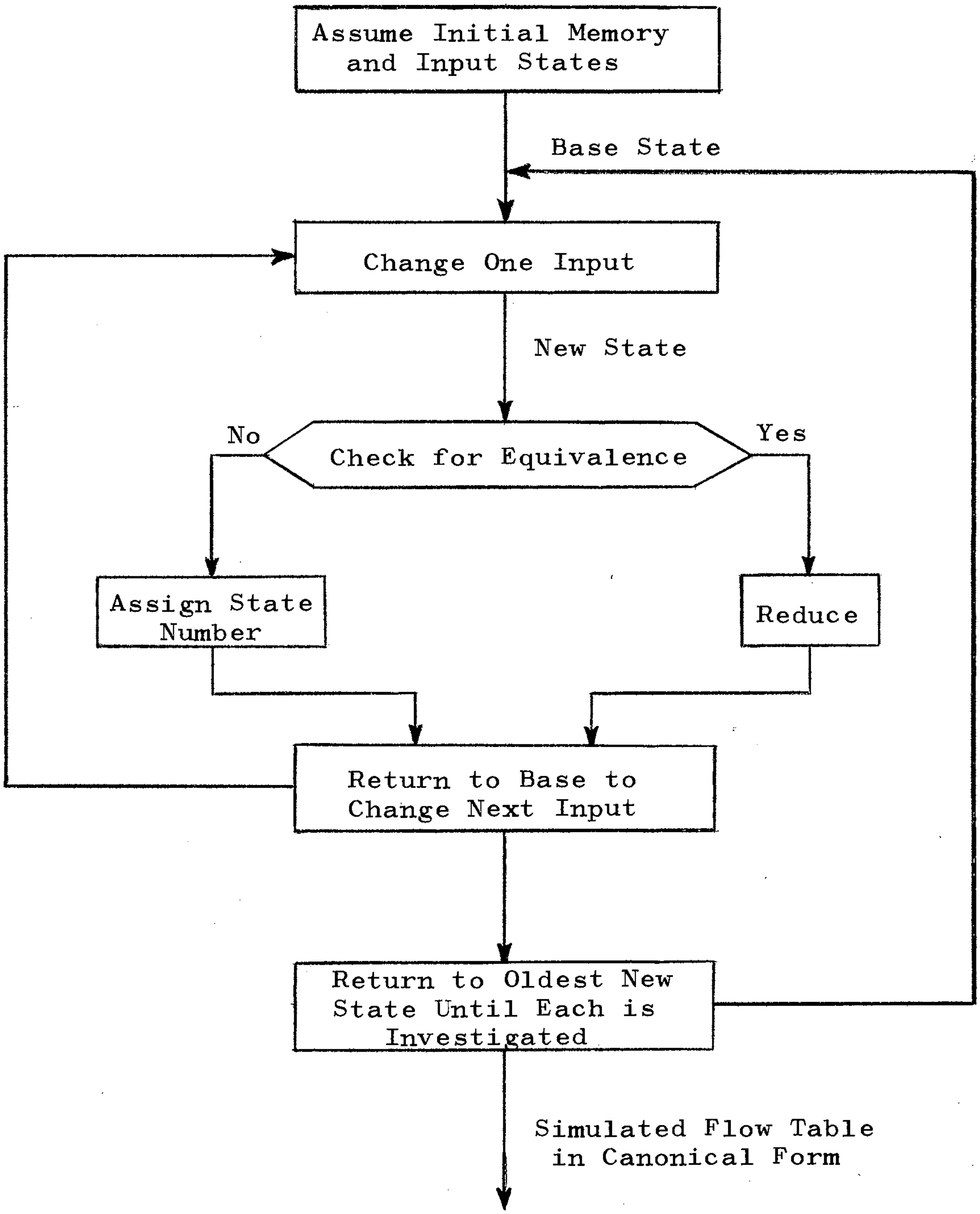


Figure 6. Flow Table for Simulation Method

Where the switching conditions are given by:

$$
\begin{aligned}
S_1 &= X_1 X_2 + X_2 Y_2 \\
R_1 &= \bar{X}_1 \bar{X}_2 + \bar{X}_2 Y_2 \\
S_2 &= X_1 \bar{X}_2 Y_1 + \bar{X}_1 X_2 Y_1 \\
R_2 &= X_1 X_2 Y_1 + \bar{X}_1 \bar{X}_2 \bar{Y}_1
\end{aligned}
\tag{14}
$$

The simulation is started by the initial assumption of a memory state and an input state. For convenience, assume that all memories are in the reset position, $(\bar{Y}_1 \bar{Y}_2)$, and that all inputs are off, (00). This state is termed the temporary base and is entered into a flow table by placing a (1) in the first row under the input column "00". The corresponding output and memory states are also indicated for this row. Starting with this state, (1), as a base, each input is excited individually to determine the system response. First, the input X_1 is excited. This defines a new state, (2), in the "10" column of Table XII (a). The transition path to stable state (2) is indicated by the unparenthesized 2 in row one. Reference to the equations reveal that the corresponding output and memory states do not change. Next, input two is changed from the base, resulting in the new state (3) in the "01" column. Again, the output and memory states remain the same. This completes the investigation from base (1) and the resulting response is indicated by Table XII (a).

The next step is to return to the oldest new state and repeat the procedure with this state as the base.

TABLE XII

STEP-BY-STEP DEVELOPMENT OF DIGITAL EQUATION SIMULATION

$X_1 X_2$ 00	10	11	01	Z_1	Z_2	Y_1	Y_2
(1)	2	–	3	0	0	0	0
	(2)			0	0	0	0
			(3)	0	0	0	0

(a) Initial Investigation

$X_1 X_2$ 00	10	11	01	Z_1	Z_2	Y_1	Y_2
(1)	2	–	3	0	0	0	0
1	(2)	4	–	0	0	0	0
			(3)	0	0	0	0
		(4)		0	0	0	0

(b) Investigation of Base (2)

$X_1 X_2$ 00	10	11	01	Z_1	Z_2	Y_1	Y_2
(1)	2	–	3	0	0	0	0
1	(2)	4	–	0	0	0	0
1	–	4	(3)	0	0	0	0
		(4)		0	0	1	0

(c) Investigation of Base (3)

TABLE XII (Continued)

$X_1 X_2$ 00	10	11	01	Z_1	Z_2	Y_1	Y_2
(1)	2	–	3	0	0	0	0
1	(2)	4	–	0	0	0	0
1	–	4	(3)	0	0	0	0
–	6	(4)	5	0	0	1	0
			(5)	0	1	1	1
	(6)			1	0	0	1

(d) Investigation of Base (4)

$X_1 X_2$ 00	10	11	01	Z_1	Z_2	Y_1	Y_2
(1)	2	–	3	0	0	0	0
1	(2)	4	–	0	0	0	0
1	–	4	(3)	0	0	0	0
–	6	(4)	5	0	0	1	0
1	–	4	(5)	0	1	1	1
1	(6)	4	–	1	0	0	1

(e) Investigations of Bases (5) and (6) and the Final Simulated Flow Table

At this point, the oldest new state is (2). With "10" as a new base, changing the first input defines a transition to the "00" column. The reader is encouraged to check both the output and switching equations to verify that the resulting output and memory states for this possible transition remain the same. The new state defined in column one is redundant since it is equivalent to (1). Hence, a transition path from (2) to (1) is indicated by a 1 entered in column one. Next, the second input is changed from the base. This defines a new state, (4), in the "11" column and the input "11" sets Y_1. This completes the investigation of (2). The result is shown in Figure XII (b).

The next base is (3) and investigations from this base reveal that both input changes describe redundant information. The first input change transfers to (4) and the second change transfers to (1). See Table XII (c).

The first input change from the next base, (4), sets Y_2, subsequently giving the output Z_2. Since this new state is not redundant, the state number (5) is assigned in the "01" column. Changing the second input from base (4) sets Y_2. $\bar{X}_2 Y_2$ resets Y_1 which results in the Z_1 output. Again, this new state is not redundant and the state number (6) is assigned to this transition. See Table XII (d).

The first input change from state (5) resets Y_2 and produces no output. This is equivalent to state (4) so no new state number is assigned. The second input change from (5) resets Y_1 and then Y_2, and has no output. This defines a transition path back to state (1).

The final state to be investigated is state (6). It can be shown that both input changes describe transitions to previously defined states. Since there are no new states to be investigated, this completes the simulation; the final simulated flow table is shown in Table XII (e).

The equations examined above were derived from the classical method. In the classical method, each memory state is assigned to a complete row. In the state matrix method, the memory elements are associated with input columns individually, not the complete row. Consequently, when simulating the state matrix equations, the particular sub-memory state associated with a column, not the total memory state, is all that needs to be considered during investigations. With this in mind, it is convenient to place the designation of the memory state beside the state number in the flow table rather than beside the complete row.

Canonical Flow Table

Considering the previously mentioned need for the canonical flow table and the simulation method discussed above, it seems reasonable to define the canonical flow table in a manner analogous to the simulated flow table. The process used here is the systematic ordering of the rows of a primitive flow table in accordance with the specified response to input changes. Starting with the origin or first stable state as a base, the state resulting from the first input change is placed in the second row. The state resulting from the second input change is placed in the third row, etc. Upon the completion of the investigation of this base, the oldest new state is then used as a base and the entire process is repeated until all rows have been reordered. The state numbers are then resequenced.

The process is best illustrated by an example. Consider the primitive flow table used in Chapter III, Table VI. The redundant state is eliminated and the reduced primitive flow table is shown in Table XIII (a).

Starting with state (1) as a base, the first input change indicates a transition path to state (2). Since state (2) is already in row two, no reordering is necessary. The second input change indicates a transition path to (7). Hence, the row containing state (7) is placed third as shown in Table XIII (b). This completes the investigation from (1).

The first input change from (2) indicates a path back to a previously ordered state, (1), requiring no reordering. The second input change indicates a path to (3). Since it happens that (3) is already in the next row, no reordering is required. See Table XIII (c).

The next base is (7). This state has transitions to states (8) and (1), respectively. Thus, state (8) is moved to the fourth row and the transition to (1) is already ordered. See Table XIII (d).

TABLE XIII

THE DEVELOPMENT OF THE CANONICAL FLOW TABLE

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
–	2	(3)	4	0	0
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0
1	–	8	(7)	0	0
–	2	(8)	7	0	1

(a) Original Primitive Flow Table

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
1	–	8	(7)	0	0
–	2	(3)	4	0	0
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0
–	2	(8)	7	0	1

(b) Initial Investigation From (1)

TABLE XIII (Continued)

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
1	–	8	(7)	0	0
–	2	(3)	4	0	0
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0
–	2	(8)	7	0	1

(c) Investigation of State (2)

$X_1 X_2$ 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
1	–	8	(7)	0	0
–	2	(3)	4	0	0
–	2	(8)	7	0	1
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0

(d) Investigation of State (7)

TABLE XIII (Continued)

X_1X_2 00	10	11	01	Z_1	Z_2
(1)	2	–	7	0	0
1	(2)	3	–	0	0
1	–	8	(7)	0	0
–	2	(3)	4	0	0
–	2	(8)	7	0	1
1	–	5	(4)	0	0
–	6	(5)	4	0	0
1	(6)	3	–	1	0

(e) Investigation of State (3)

X_1X_2 00	10	11	01	Z_1	Z_2
(1)	2	–	3	0	0
1	(2)	4	–	0	0
1	–	5	(3)	0	0
–	2	(4)	6	0	0
–	2	(5)	3	0	1
1	–	7	(6)	0	0
–	8	(7)	6	0	0
1	(8)	4	–	1	0

(f) The Completed Canonical Flow Table With Resequenced State Numbers

The reader is encouraged to investigate states (3), (8), (4), and (5) to verify that the remaining states are already in the proper order. Once the rows are in the proper order, the state numbers are then resequenced so that each stable state number corresponds to its row number. The completed canonical flow table is shown in Table XIII (f).

One further point which has not been decided at the time of this writing is the definition of an origin for the primitive flow table. The origin is usually thought of as being the state with the inputs off and having the desired sequence or logic developed from this point. However, a more meaningful definition of the origin should consider the topology of the transitions as being more important than the number of inputs or outputs that are on or off. This definition should be comprehensive enough so that an origin can be uniquely determined for any primitive flow table.

Since an origin is not defined in this chapter, the canonical flow table used here is not unique. The rows are in the proper order, but the origin or first row in the canonical flow table will be the first row given in the primitive flow table. This depends upon the designer's personal preference and will, in general, not be unique. However, for all of the cases investigated by the author, the simulated flow table has resulted with the same origin as the canonical primitive flow table, thereby presenting no problem.

CHAPTER V

DIGITAL COMPUTER PROGRAMS

The logic systems program is designed to perform either the synthesis or simulation of digital control systems. In order to perform system synthesis, the user needs only to supply the primitive flow table describing the desired logic; the computer program will then perform the necessary steps to obtain the digital equations by the state matrix synthesis procedure given in Chapter III. These equations may then be implemented to obtain a circuit containing the information represented by the primitive flow table.

With this capability, the designer does not need to know a formal synthesis procedure; he only needs to know how to write a primitive flow table, call the program, and then implement the resulting equations.

The simulation program offers a powerful tool for the analysis of digital systems. This program generates the primitive flow table implied by a set of digital equations by the method described in Chapter IV. The simulation program may be used either to confirm the validity of equations or to analyze the logical implications of existing circuits. This can be advantageous when working with intuitively designed circuits.

The FORTRAN IV source deck listed in Appendix B has been running on the WATFOR terminal of OSU's IBM 360/50 computing facility. A time-share version of the program is also available to allow users with remote teletype terminals to have access to the program from any phone line. A user's guide for the time share program will be made available under a separate cover.

Since the programs are rather lengthy and the listings given in Appendix B contain many of the details of the programs, only the philosophy of the programs is presented in the rest of this chapter. Appendix C shows both the calling information and the computer solutions to many example problems. For further details of the use of this program, see the write-up in Appendix B and the example solutions in Appendix C.

Synthesis Program LOGSYN

Subroutine LOGSYN is the executive subroutine for the synthesis of digital systems. The flow chart showing the relation of subroutines is given by Figure 7. Subroutine LOGSYN reads in the input data concerning the primitive flow table and then uses subroutine PRINT to print the original primitive flow table. This primitive flow table is then examined by subroutine EQUIV to reduce any redundant information which might be contained in the flow table. If two states are found to be redundant, one is eliminated and an indication of this reduction is printed out below the

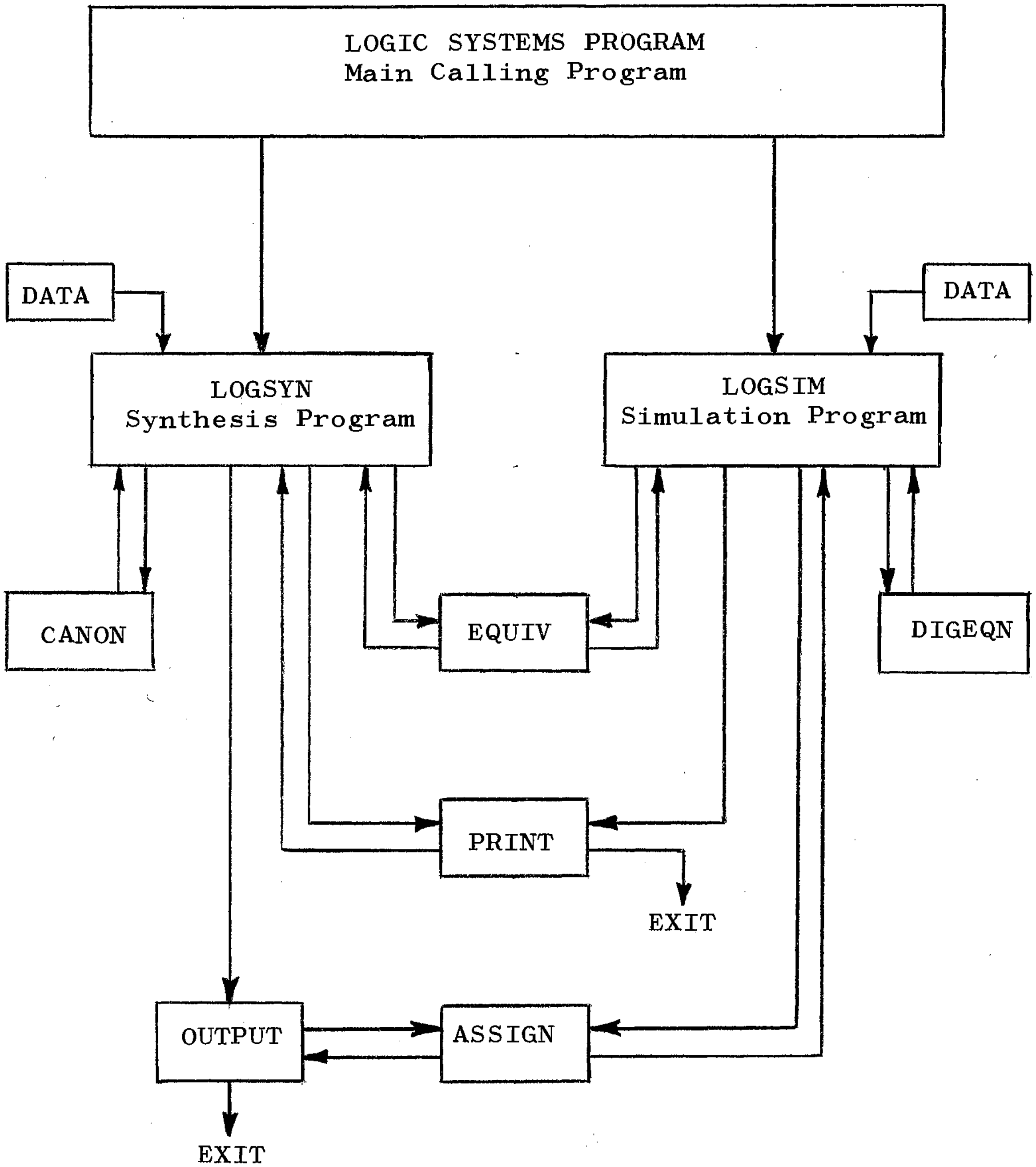


Figure 7. Flow Diagram for Logic Systems Program

original primitive flow table. This reduced primitive flow table is then put into canonical form by subroutine CANON. In this routine the rows of the primitive flow table are reordered and resequenced as described in Chapter IV. The resulting canonical flow table is then printed by subroutine PRINT.

Subroutine OUTPUT performs most of the steps required for system synthesis. In this routine, the memory requirements for each column are determined and subroutine ASSIGN is used to provide the passive memory assignment code to distinguish between stable states. After each state is made unique by the proper memory assignment, the state signals are printed. This gives the input and memory combination which describes each stable state. Next, the switching conditions required for proper circuit action are printed. The switching conditions are presented by giving the state numbers at which a switch occurs. Finally, the output equations are given by printing the states at which the individual outputs appear. This completes the synthesis procedure and the program then returns to the main calling program to exit.

Simulation Program LOGSIM

Subroutine LOGSIM is the executive program for systems simulation. As can be seen by Figure 7, this routine reads the data cards containing basic information concerning the system to be simulated. Subroutine LOGSIM then sets up a loop similar to the one shown in Figure 6 of Chapter IV. This routine changes an input according to a Gray code. The Gray code is supplied by subroutine ASSIGN. The corresponding system response is determined by subroutine DIGEQN. Subroutine DIGEQN is a subroutine supplied by the user containing the switching and output equations. The input change and the corresponding response determines a new state. This state is then checked for redundancy by subroutine EQUIV. If the new state is not equivalent to a previously defined state, a state number is assigned to this state.

This process is continued until all states have been investigated and no new information is being generated. At this point, the simulation is completed and subroutine PRINT is then used to print the simulated primitive flow table.

Appendix C contains many examples of problems solved with both the synthesis and simulation programs. The reader is referred to the appendices for further information concerning the usage and input for these computer programs.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Summary

The major effort of this thesis has been concentrated upon the development of new techniques for the synthesis and analysis of digital logic systems. The synthesis procedures are based upon the assumption that the outputs are related to the inputs. This relation can be represented by the vector matrix equation

$$\left[Z\right] = \left[M\right]\left[X\right] \tag{15}$$

Since the input vector $\left[X\right]$ and the desired output vector $\left[Z\right]$ are known, the synthesis reduces to the determination of the binary matrix $\left[M\right]$. The entries in this matrix give the relationship between the inputs and the outputs and contain memory information of previous states.

The synthesis proceeds from entering the state numbers from a statement of the desired logic or sequence into the matrix $\left[M\right]$. The memory requirements are then determined and entered into the matrix, producing the set of output equations in matrix form. Specification of the switching conditions for the memory elements completes the synthesis procedure.

The simulation technique presented here is quite helpful either to verify digital equations or to analyze existing circuits. This technique can also be used to totally redesign existing circuits by first writing the equations for the circuit, obtaining the simulated flow table, and then synthesizing the state matrix equations from this flow table. The canonical flow table is also an aid for analysis and comparison.

The digital computer programs developed to perform either systems synthesis or simulation offer a great design tool to the designer who is unfamiliar with switching circuit theory. These programs perform the steps necessary to synthesize or simulate digital systems as described in Chapters III and IV. With these programs, the designer only needs to be able to write a primitive flow table and to implement equations.

Comparison to Other Techniques

To fully evaluate the merits of this synthesis technique, a general comparison to existing techniques should be made. This technique is compared to the classical method and those methods suggested by Cole (1) and Maroney (6) on the basis of the following areas:

1. Simplicity of the Synthesis Procedure - The execution of the state matrix synthesis

procedure is much less complicated than the classical method since the merging operation, operational flow table, Karnaugh maps, etc., are eliminated. The total concepts of circuit synthesis are much easier to grasp, partially due to the use of the familiar matrix notation. In comparing to the tabular methods of Cole and Maroney, one can only compare on the basis of procedure simplicity since these methods produce essentially the same equations as the techniques presented here. The philosophy of circuit implementation is also the same. Thus, any comment made about the state matrix equations or circuits is equally applicable to those of the tabular methods.

Cole's tabular technique for the synthesis of feedback sequential circuits handles persistent states in a more straightforward manner than does the matrix method. However, the search procedure for persistent states in the matrix is more mechanical. It is felt that the synthesis concepts using the matrix notation are easier to grasp than the tabular method; but this is a matter of personal preference.

Maroney's tabular method handles random input problems in a tabular technique similar to Cole's method. The random input possibility requires multiple transition paths from states. The transitions from each state are very hard to follow in the tabular form; whereas, the primitive flow table provides a graphic display of transition paths. This causes a slight problem for involved sequences since the designer must keep much of this information in his head rather than on paper. Also, redundant states are harder to sense from the tabular technique than from the primitive flow table. Again, it is felt that the matrix synthesis concepts are easier to grasp.

2. <u>Simplicity of Circuit Implementation Procedure</u> - The state matrix and tabular synthesis procedures offer a specific step-by-step procedure for circuit implementation; whereas, the classical method does not lend itself to any set procedure.

3. <u>Circuit Complexity</u> - The number of elements required to implement a circuit is generally a good indication of the circuit complexity. Although the state matrix equations usually require more memory elements, the use of the passive memory effect reduces the total number of elements to about the same or less than that required by the classical method. However, this is not a very rigid basis for comparison

since the classical method offers such a flexibility in writing equations from the Karnaugh maps. Each designer might derive different equations from the classical method depending upon his own personal preference. Thus, to compare on this basis, the equations from the classical Karnaugh maps must be rewritten until a combination with minimum hardware is determined. This is then compared to the state matrix method.

4. <u>Other Circuit Considerations</u> - The state matrix synthesis procedure offers circuit features that are not available from the classical method. Among these are the elimination of switching hazards, cycles, and other logical complications. Another very important feature is the prepared flow path concept. In this procedure, each memory is switched prior to any input change, thus preparing all possible paths from that state. Notice that in the classical method the input change causes the switching of a memory to give the next state. The prepared flow path feature produces circuits in which the only delays are the delays caused by forming the input combination and any transmission time delay. Thus, circuit response time is at a minimum.

Another important feature stemming from the prepared flow path concept is that the passive memory elements used in this synthesis procedure are never switched when they are under power as they are in the classical method. Switching under power causes undesirable transient pulses in the circuit. This is avoided by switching the element before the passive signal appears.

Suggestions for Further Study

As is true with any study, there are many areas providing interesting further study. Among these are:

1. A Synthesis Procedure Considering Some Combination of the Total Input and Changed Inputs - The synthesis procedure for feedback sequential circuits presented in Chapter II considers only the changed input whereas the procedure for random input circuits (Chapter III) considers only the total input state. Both of these approaches have their own distinguishing merit; however, it is felt that some combination of the two concepts will consistently produce circuits having more of the desirable features of both methods.

   In the feedback sequential method, the W elements can often be replaced by "anding" another input signal to the state signal. Rules for doing this should be investigated.

Another interesting synthesis concept is the use of internal information as an auxiliary input. It seems that as more information is used as input information, the less complicated the resulting circuit. This concept has not yet been pursued.

2. Definition of Origin for Canonical Flow Table - The canonical flow table defined in Chapter IV has a unique relationship involving the order of the rows of a primitive flow table. Any two flow tables containing the same information will always result in canonical flow tables having the same row relationships. However, the row appearing first in the table is thus far left to the designer's preference. Although this is usually acceptable, a rigorous definition for the origin or first row of the canonical flow table should be made considering only the topology of the table's transition paths. This would provide a unique format for displaying the information contained in any primitive flow table.

3. Computer Program for Feedback Sequential Synthesis - Efforts should be made to write a computer program to perform the necessary steps for the synthesis of feedback sequential circuits

as presented in Chapter II. The techniques already developed for the present program could be easily adapted to provide a program to accomplish this from a statement of the desired sequence.

4. A Logic Synthesis Procedure Considering Proportional As Well As Binary Variables - To date, the synthesis of physical systems using formal logic has been restricted to binary or digital systems. Considering the matrix synthesis philosophy presented in this thesis, it seems natural to extend this technique to include proportional or dynamic variables as well as binary variables. A proportional variable could be entered into the state matrix to modify a state in the same manner as the memory elements are in this thesis. The proportional state modifier would tell not only when to give the output but would also tell how. This "how" could be the proportional signal rather than the binary signal now used.

The author is currently engaged in investigating the possibilities of such a synthesis procedure.

A SELECTED BIBLIOGRAPHY

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

(2) Fitch, E. C. Jr. Fluid Logic. Stillwater: Oklahoma State University, 1966.

(3) Huffman, D. A. "The Synthesis of Sequential Circuits." Journal of the Franklin Institute, Vol. 257, No. 4 (1954).

(4) Jensen, D. F., et al. "Pneumatic Diaphragm Logic." Proceedings of the Fluidics Symposium. New York: ASME, 1967.

(5) Marcus, M. P. Switching Circuits For Engineers. Englewood Cliffs: Prentice-Hall, 1962.

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

(7) Miller, R. E. Switching Theory, 2 Vols. New York: John Wiley, 1965.

(8) Moore, E. F., ed. Sequential Machines: Selected Papers. Reading, Mass.: Addison-Wesley, 1964.

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

APPENDIX A

THE PASSIVE MEMORY

APPENDIX A

THE PASSIVE MEMORY

This appendix deals with the definition, description, and assignment of passive memory elements.

Definition

Any memory element which does not rely upon an active power source to retain its output state is said to be a <u>passive</u> memory element. In most cases, these devices have a mechanical memory and the logic signal is merely directed through the device according to its mechanical position. The best example of this concept is the four-way, two-position detent valve shown in Figure 8.

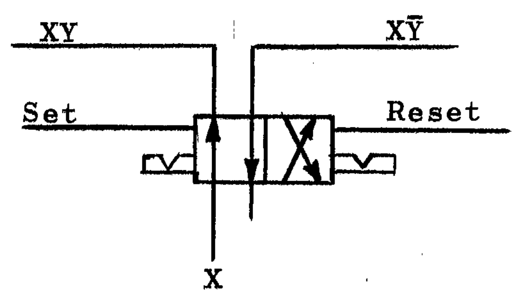


Figure 8. Passive Memory Valve

Description

This device has many salient features, most important of which is the mechanical memory. Once the device has been switched by either the set or reset signals, the device remains in that position due to the detent hold feature. The signal sent through the device does not necessarily have to be an active signal connected to the supply; this signal may be an input or logic signal which appears only occasionally.

By sending a logic signal through the device, the output XY appears only when the memory element is in the proper position (indicated by Y) "and" the logic signal X is on. The $X\overline{Y}$ signal appears only when the device is in the "reset" position "and" the signal X is on. This device holds its mechanical position to display memory characteristics and it forms two "and" combinations ($X \cdot Y$ and $X \cdot \overline{Y}$); thus, the passive memory device serves the function of three logic elements, memory and two "ands". By utilizing this effect, circuit complexity and hardware can be reduced substantially.

Another advantageous feature of this device is the complementary output. Notice that the device has two outputs, XY and $X\overline{Y}$; when one is on (pressurized) the other is off (to tank). Thus, the need for the inversion of Y to get its complement $\overline{Y}$ is eliminated.

The pneumatic diaphragm logic device (4) possesses similar mechanical memory characteristics as the valve described above.

Fluidic passive memory devices without moving mechanical parts do not exist; however, a similar savings in circuit hardware can be made by the use of the two devices shown in Figure 9. The bistable amplifier is an active memory element and its complementary outputs are fed into a passive "and". The passive "and" element has complementary outputs serving the function of two separate "ands" to form XY and X$\bar{Y}$.

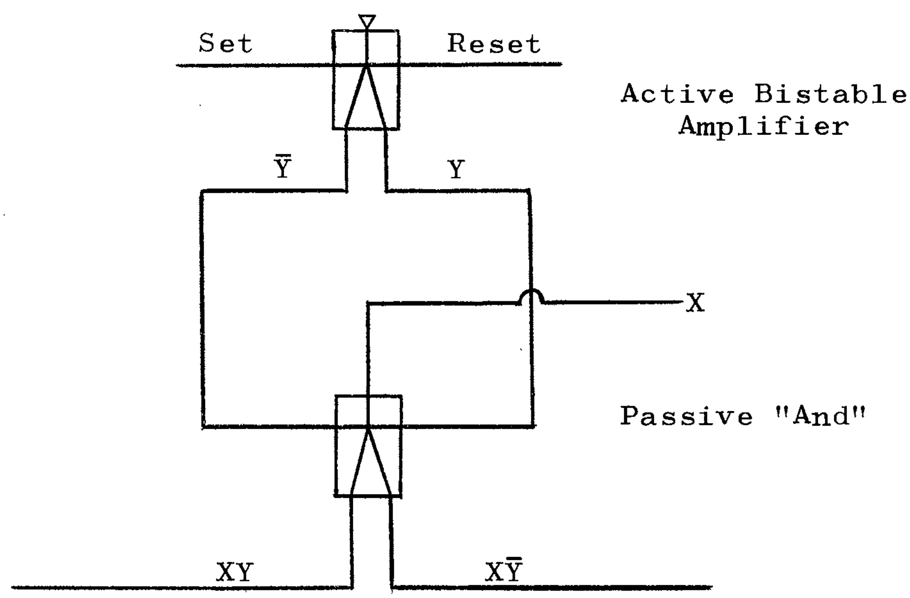


Figure 9. Fluidic Memory Circuit

The latching relay performs the analogous passive memory function in electronic circuits. However, modern technology has almost phased out the use of relays in compact logic circuits. Even so, the addition of two extra "ands" in an electronic circuit is much less costly than the same for fluid circuits. The usual bistable flip-flop integrated circuit could be built with outputs XY, and $X\bar{Y}$ instead of the usual Y, $\bar{Y}$ where X is some logic signal.

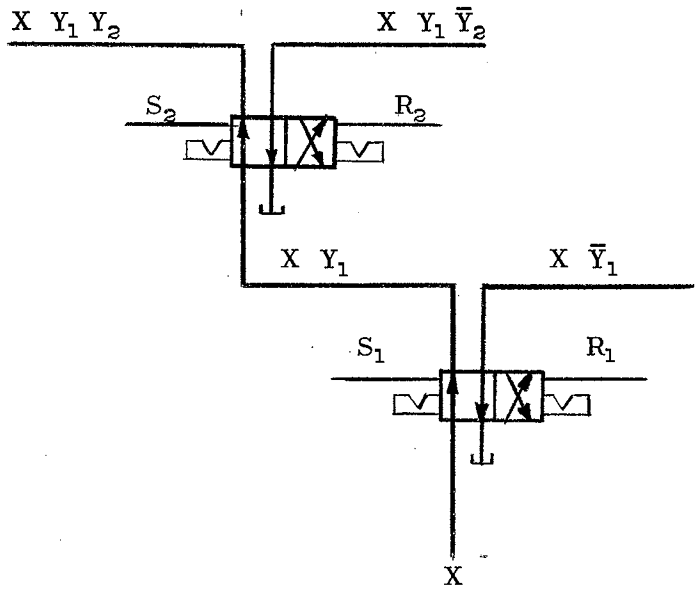
Assignment

As has been shown above, the passive memory can be used to reduce hardware when distinguishing between two states. The problem of assignment when higher orders of memory are required is discussed next. By using one more passive memory element, the circuit of Figure 8 is modified to form three unique memory states as shown by Figure 10 (a). Four unique states are obtained in Figure 10 (b) by adding one more passive memory element.

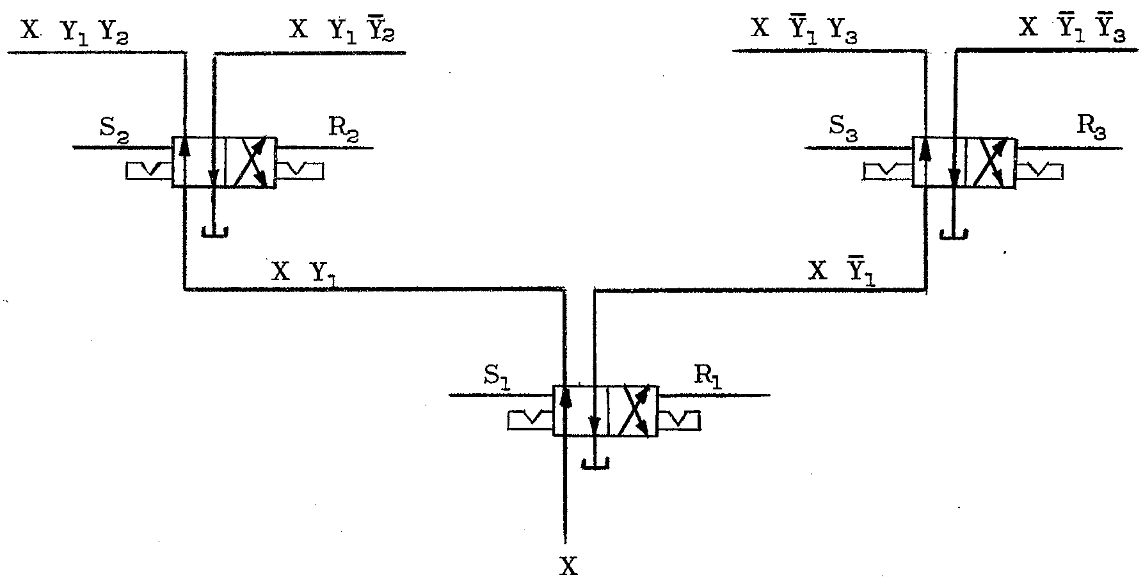
As shown by the previous discussion, each time another memory element is added, another unique passive memory state results. In general, N-1 passive memory elements describe N unique states. The assignment schematic shown in Figure 11 illustrates the passive memory code. To describe N unique states, omit all memory elements numbered above N-1.

The alternating placement of elements in the assignment code allows the proper balance of fluid power. Higher orders may be obtained in the same alternating pattern.

To illustrate the assignment technique for making each state of an input column unique, consider the three states



(a) Three Unique States, $X\ Y_1 Y_2$, $X\ Y_1 \bar{Y}_2$ and $X\ \bar{Y}_1$



(b) Four Unique States, $X\ Y_1 Y_2$, $X\ Y_1 \bar{Y}_2$, $X\ \bar{Y}_1 Y_3$, and $X\ \bar{Y}_1 \bar{Y}_3$

Figure 10. Passive Memory Assignment Circuits

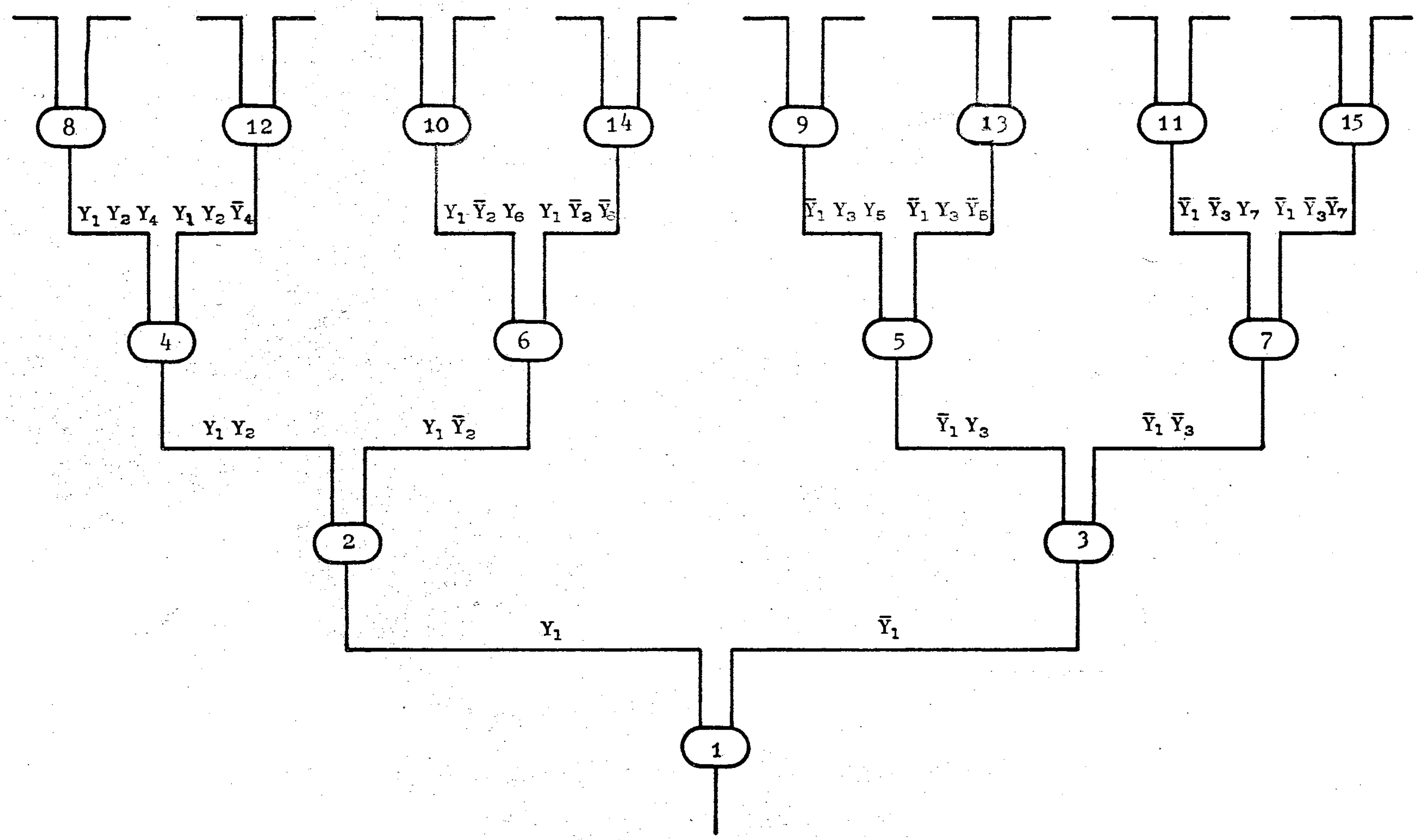


Figure 11. Passive Memory Code Schematic

1, 3, and 5. Using the double subscript notation, the memory states are assigned as follows:

(1) $Y_{15}Y_{13}$

(3) $Y_{15}\overline{Y}_{13}$

(5) $\overline{Y}_{15}$

The reader is cautioned not to confuse the double subscript notation discussed here and the single subscript notation used in Figure 11. The double subscript notation carries information of the switching conditions. For example, Y_{15} (read Y, one, five) is set prior to state 1, and is reset prior to state 5. As an example of higher order memory state assignment, consider the states 1, 3, 5, 8, 10, and 13. The assignment is as follows:

(1) $Y_{18}Y_{15}\ Y_{13}$

(3) $Y_{18}Y_{15}\ \overline{Y}_{13}$

(5) $Y_{18}\overline{Y}_{15}$

(8) $\overline{Y}_{18}Y_{813}Y_{810}$

(10) $\overline{Y}_{18}Y_{813}\overline{Y}_{810}$

(13) $\overline{Y}_{18}\overline{Y}_{813}$

The reader is encouraged to implement this circuit using Figure 11 as a guideline.

As a final note, it should be pointed out that this synthesis procedure allows every column in the state matrix to be treated independently. In this respect, each input state (or changed input) may be sent through memory elements as a passive signal.

APPENDIX B

LISTING OF COMPUTER PROGRAMS

```
$JOB    10050,442-44-0874
C      ****************************************************************LSP0001
C      *                                                               *LSP0002
C      *                   LOGIC  SYSTEMS  PROGRAM                     *LSP0003
C      *                                                               *LSP0004
C      *                                                               *LSP0005
C      *                        ROBERT L. WOODS                        *LSP0006
C      *                                                               *LSP0007
C      *          SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING       *LSP0008
C      *                  OKLAHOMA STATE UNIVERSITY                    *LSP0009
C      *                       DECEMBER, 1969                          *LSP0010
C      *                                                               *LSP0011
C      *                                                               *LSP0012
C      *                                                               *LSP0013
C      *     THIS PROGRAM IS DESIGNED TO PERFORM EITHER THE SYNTHESIS *LSP0014
C      *  OR SIMULATION OF DIGITAL CONTROL SYSTEMS.  FOR FURTHER      *LSP0015
C      *  INFORMATION, SEE THE M.S. THESIS "THE STATE MATRIX METHOD   *LSP0016
C      *  FOR THE SYNTHESIS OF DIGITAL LOGIC SYSTEMS".                *LSP0017
C      *                                                               *LSP0018
C      *                                                               *LSP0019
C      *                                                               *LSP0020
C      *                                                               *LSP0021
C      *SYSTEMS SYNTHESIS:  -------------------------------------------*LSP0022
C      *                                                               *LSP0023
C      *     IN ORDER TO PERFORM SYSTEM SYNTHESIS USING THE STATE     *LSP0024
C      *  MATRIX SYNTHESIS PROCEDURE, THE USER MUST USE THE FOLLOWING *LSP0025
C      *  CALLING PROGRAM.                                            *LSP0026
C      *                                                               *LSP0027
C      CALL LOGSYN                                                     *LSP0028
C      STOP                                                            *LSP0029
C      END                                                             *LSP0030
C      *                                                               *LSP0031
C      *                                                               *LSP0032
C      *     THE USER MUST ALSO SUPPLY THE FOLLOWING INFORMATION TO   *LSP0033
C      *  BE READ FROM DATA CARDS.                                    *LSP0034
C      *                                                               *LSP0035
C      *     CARD 1 - PROBLEM IDENTIFICATION                          *LSP0036
C      *              ANYTHING READ FROM THIS CARD WILL BE PRINTED IN *LSP0037
C      *              THE OUTPUT.                                     *LSP0038
C      *     CARD 2 - NI, NO, NR  =  FORMAT(3I2)                      *LSP0039
C      *              NI = NUMBER OF INPUTS                           *LSP0040
C      *              NO = NUMBER OF OUTPUTS                          *LSP0041
C      *              NR = NUMBER OF ROWS IN THE PRIMITIVE FLOW TABLE *LSP0042
C      *     CARD 3 - INPUT STATES FOR EACH COLUMN  =  FORMAT(16(4I1))*LSP0043
C      *              FOR TWO INPUTS, THE CARD SHOULD READ            *LSP0044
C      *              00  10  11  01                                  *LSP0045
C      *     CARD 4 AND SUCCESSIVE CARDS EACH CONTAIN ONE ROW OF THE  *LSP0046
C      *              PRIMITIVE FLOW TABLE AND THE CORRESPONDING      *LSP0047
C      *              OUTPUT STATE  =  FORMAT(16I4,6I1)               *LSP0048
C      *              16 COLUMNS AND 6 OUTPUTS ARE READ FROM EACH     *LSP0049
C      *              CARD.  STABLE STATES ARE INDICATED BY ADDING    *LSP0050
C      *              1000 TO THE STATE NUMBER TO DISTINGUISH THEM    *LSP0051
C      *              FROM TRANSITION PATHS.  A "DON'T CARE" OUTPUT   *LSP0052
C      *              IS INDICATED BY ENTERING A "2" INSTEAD OF A "0" *LSP0053
C      *              OR A "1".                                       *LSP0054
C      *                                                               *LSP0055
C      *                                                               *LSP0056
C      *                                                               *LSP0057
C      *                                                               *LSP0058
C      *---------------------------------------------------------------*LSP0059
C      *SYSTEMS SIMULATION:  ------------------------------------------*LSP0060
C      *                                                               *LSP0061
C      *                                                               *LSP0062
C      *     IN ORDER TO PERFORM SYSTEM SIMULATION, THE USER MUST USE *LSP0063
C      *  THE FOLLOWING CALLING PROGRAM.                              *LSP0064
C      *                                                               *LSP0065
C      CALL LOGSIM                                                     *LSP0066
C      STOP                                                            *LSP0067
C      END                                                             *LSP0068
C      *                                                               *LSP0069
C      *     THE USER MUST ALSO SUPPLY THE SUBROUTINE DIGEQN          *LSP0070
C      *  DESCRIBING THE SYSTEM EQUATIONS, AND THE FOLLOWING INFORMATION*LSP0071
C      *  TO BE READ FROM DATA CARDS.                                 *LSP0072
C      *                                                               *LSP0073
C      *     CARD 1 - PROBLEM IDENTIFICATION                          *LSP0074
C      *              ANYTHING READ FROM THIS CARD WILL BE PRINTED IN *LSP0075
C      *              THE OUTPUT.                                     *LSP0076
C      *     CARD 2 - NI, NO, NM  =  FORMAT(3I2)                      *LSP0077
C      *              THE NUMBER OF INPUTS, OUTPUTS, AND MEMORIES.    *LSP0078
C      *     CARD 3 - Y(M), X(I)  =  FORMAT(18I1,4I1)                 *LSP0079
C      *              THE INITIAL STATE OF ALL OF THE MEMORY ELEMENTS *LSP0080
C      *              AND THE INITIAL STATE OF ALL INPUTS.            *LSP0081
C      *     CARD 4 AND SUCCESSIVE CARDS EACH CONTAIN: THE NUMBER OF  *LSP0082
C      *              MEMORY ELEMENTS ASSOCIATED WITH THE JC-TH       *LSP0083
C      *              COLUMN, MC(JC,1), AND THE CORRESPONDING NUMBER  *LSP0084
C      *              DESIGNATION OF THE JM-TH MEMORY IN THE JC-TH    *LSP0085
C      *              COLUMN, MC(JC,JM+1)  =  FORMAT(I2,9I2)          *LSP0086
C      *                                                               *LSP0087
C      *                                                               *LSP0088
C      *                                                               *LSP0089
C      *                                                               *LSP0090
C      *-----------------------  ARRAY SIZES  -------------------------*LSP0091
C      *                                                               *LSP0092
C      *  COMMON STATEMENTS -                                         *LSP0093
C      *      /ALL/ - IX(NI,NC), IY(NM,NR), IZ(NO,NR), S(NR,NC)       *LSP0094
C      *      /EQN/ - X(NI), Y(NM), Z(NO), KS(NR,NC), MC(NC,NM/2+1)   *LSP0095
C      *      /OUT/ - SSC(NC,NR/2+1), SSR(NR)                         *LSP0096
C      *      /ASN/ - IG(NR/2-1,NR/2)                                 *LSP0097
C      *      /IDN/ - IDEN(20)                                        *LSP0098
C      *                                                               *LSP0099
C      *  DIMENSION STATEMENTS -                                      *LSP0100
C      *      DIGEQN - ITS(NC), MS(NM), MR(NM)                        *LSP0101
C      *      EQUIV  - IT(NC)                                         *LSP0102
C      *      OUTPUT - SET(NM,NR/2), RESET(NM,NR/2), PS(NR/2)         *LSP0103
C      *               IZS(NO,NR), JL(NM/2),IYP(MO,NR/2),INOT(MO,NR/2),*LSP0104
C      *               NY(MO=5), IO(NR=40)                            *LSP0105
C      *      PRINT  - IN3(41), IXP(NI=4), IZP(NO=6), MS(NC), MSS(NC) *LSP0106
C      *      ASSIGN - IA(NR/2), RA(NR/2)                             *LSP0107
C      *                                                               *LSP0108
C      *  NOTES -                                                     *LSP0109
C      *         - NM = NR-NC                                         *LSP0110
C      *         - MO = LOG(NR,1) + 1                                 *LSP0111
C      *         - SUBSCRIPTS SUCH AS IXP(NI=4) IMPLY THAT THE ARRAY IXP *LSP0112
C      *           IS DEFINED IN A DATA STATEMENT.                    *LSP0113
C      *                                                               *LSP0114
C      ****************************************************************LSP0115
       CALL LOGSYN                                                      LSP0116
       CALL LOGSIM                                                      LSP0117
       STOP                                                             LSP0118
       END                                                              LSP0119
```

```
      SUBROUTINE LOGSYN
C     ****************************************************************
C     *                                                              *
C     *       SUBROUTINE LOGSYN IS THE EXECUTIVE PROGRAM FOR SYSTEM   *
C     *  SYNTHESIS.  THIS PROGRAM READS THE DATA CARDS, PRINTS THE    *
C     *  ORIGINAL PRIMITIVE FLOW TABLE, CHECKS FOR ANY REDUNDANT      *
C     *  INFORMATION IN THE PRIMITIVE FLOW TABLE, REARRANGES THE ROWS *
C     *  TO FORM THE CANONICAL FLOW TABLE, PRINTS THE CANONICAL FLOW  *
C     *  TABLE, AND THEN PRINTS THE STATE, SWITCHING, AND OUTPUT      *
C     *  INFORMATION.                                                 *
C     *                                                              *
C     *                                                              *
C     *  NI = THE NUMBER OF INPUTS                                    *
C     *  NO = THE NUMBER OF OUTPUTS                                   *
C     *  NR = THE NUMBER OF ROWS IN THE PRIMITIVE FLOW TABLE          *
C     *  NC = THE NUMBER OF COLUMNS IN THE PRIMITIVE FLOW TABLE.      *
C     *  IX(I,JC) = THE STATE OF THE I-TH INPUT FOR THE JC-TH COLUMN  *
C     *  IZ(J,IR) = THE STATE OF THE J-TH MEMORY IN THE IR-TH ROW     *
C     *  S(IR,JC) = THE ENTRY IN THE IR-TH ROW AND JC-TH COLUMN OF THE *
C     *             PRIMITIVE FLOW TABLE.  STABLE STATES ARE INDICATED *
C     *             BY ADDING 1000 TO THE STATE NUMBER TO DISTINGUISH *
C     *             THEM FROM TRANSITION PATHS.                       *
C     *  SSC(JC,1) = THE NUMBER OF STABLE STATES IN THE JC-TH COLUMN  *
C     *  SSC(JC,K+1) = THE K-TH STABLE STATE IN THE JC-TH COLUMN      *
C     *  SSR(IR)   = THE STABLE STATE IN THE IR-TH ROW                *
C     *                                                              *
C     ****************************************************************
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)
      COMMON /OUT/ SSC(16,21), SSR(40)
      COMMON /IDN/ IDEN(20)
      INTEGER S, SSC, SSR
    1 FORMAT(20A4)
    2 FORMAT(3I2)
    3 FORMAT(16(4I1))
    4 FORMAT(16I4,6I1)
    5 FORMAT(1H1,30X,'LOGIC SYNTHESIS',/,27X,'FOR 'I1,' INPUTS, ',I1,
     *       ' OUTPUTS.',///)
    6 FORMAT(10X'ORIGINAL PRIMITIVE FLOW TABLE FOR'/15X,20A4///)
    7 FORMAT(1H1,9X'CANONICAL FLOW TABLE FOR'/15X,20A4///)
    8 FORMAT(/10X'WHAT IS 'I4' IN COLUMN 'I2', ROW 'I2'?'/
     *        10X'CHECK YOUR DATA FOR ERROR')
    9 FORMAT(1H1)
   10 FORMAT(1H1,9X,'NUMBER OF INPUTS = 'I2',  NUMBER OF OUTPUTS = 'I2
     *       ',  NUMBER OF ROWS = 'I2/10X'TO RUN A PROBLEM OF THIS SIZE'
     *       ', THE ARRAY SIZES MUST BE ALTERED.')
      READ(5,1) IDEN
      READ(5,2) NI, NO, NR
      ERROR = 0
      IF(NI .GT. 4) ERROR = 1
      IF(NO .GT. 6) ERROR = 1
      IF(NR .GT.40) ERROR = 1
      IF( ERROR .EQ. 1.0) WRITE(6,10) NI, NO, NR
      NC = 2**NI
      READ(5,3)((IX(I,JC),I=1,4),JC=1,NC)
      READ(5,4)((S(IR,JC),JC=1,16),(IZ(J,IR),J=1,6),IR=1,NR)
      WRITE(6,5) NI, NO
      WRITE(6,6) IDEN
      CALL PRINT(0)
      CALL EQUIV(0)
      CALL CANON
      WRITE(6,7) IDEN
      CALL PRINT(0)
      DO 21 JC=1,NC
      K = 0
      DO 20 IR=1,NR
      IS = S(IR,JC) - 1000
      IF(IS .LT. 0) GO TO 20
      K = K+1
      SSC(JC,K+1) = IS
      SSR(IR) = IS
   20 IF(S(IR,JC) .LT. 0) WRITE(6,8) S(IR,JC), JC, IR
   21 SSC(JC,1) = K
      CALL OUTPUT
      WRITE(6,9)
      RETURN
      END
```

```
      SUBROUTINE LOGSIM
C     ****************************************************************
C     *                                                              *
C     *      SUBROUTINE LOGSIM IS THE EXECUTIVE PROGRAM FOR SYSTEM   *
C     *  SIMULATION.  THIS PROGRAM READS THE REQUIRED DATA CARDS AND *
C     *  CONDUCTS THE SYSTEMATIC INPUT EXCITATION. THE RESPONSE TO   *
C     *  INPUT CHANGES IS CHECKED EACH TIME BY CALLING THE SUBROUTINE*
C     *  CONTAINING THE DIGITAL EQUATIONS.  EACH NEW STATE DEFINED IS*
C     *  CHECKED FOR REDUNDANCY AND IS ELIMINATED IF EQUIVALENT TO A *
C     *  PREVIOUSLY DEFINED STATE.  UPON COMPLETION OF THE SIMULATION,*
C     *  THE RESULTING PRIMITIVE FLOW TABLE IS PRINTED.              *
C     *                                                              *
C     *                                                              *
C     *  NI = THE NUMBER OF INPUTS                                   *
C     *  NO = THE NUMBER OF OUTPUTS                                  *
C     *  NR = THE NUMBER OF ROWS IN THE PRIMITIVE FLOW TABLE         *
C     *  NC = THE NUMBER OF COLUMNS IN THE PRIMITIVE FLOW TABLE.     *
C     *  IX(I,JC) = THE STATE OF THE I-TH INPUT FOR THE JC-TH COLUMN *
C     *  IY(M,IR) = THE STATE OF THE M-TH MEMORY IN THE IR-TH ROW    *
C     *  IZ(J,IR) = THE STATE OF THE J-TH MEMORY ELEMENT IN THE IR-TH *
C     *  S(IR,JC) = THE ENTRY IN THE IR-TH ROW AND JC-TH COLUMN OF THE *
C     *             PRIMITIVE FLOW TABLE.  STABLE STATES ARE INDICATED *
C     *             BY ADDING 1000 TO THE STATE NUMBER TO DISTINGUISH *
C     *             THEM FROM TRANSITION PATHS.                      *
C     *  MC(JC,1) = THE NUMBER OF MEMORY ELEMENTS ASSOCIATED WITH THE *
C     *             JC-TH COLUMN                                     *
C     *  MC(JC,JM+1) = THE NUMBER DESIGNATION OF THE JM-TH MEMORY    *
C     *             IN THE JC-TH COLUMN                              *
C     *                                                              *
C     ****************************************************************
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)
      COMMON /EQN/ X(4), Y(36), Z(6), KS(40,16), MC(16,19)
      COMMON /ASN/ IG(19,20)
      COMMON /IDN/ IDEN(20)
      INTEGER X, Y, Z, S
    1 FORMAT(20A4)
    2 FORMAT(3I2)
    3 FORMAT(30I1,4I1)
    4 FORMAT((19I2))
    5 FORMAT(1H1,30X'LOGIC SIMULATION'/20X'FOR 'I1' INPUTS, 'I1
     *       ' OUTPUTS, 'I2' MEMORIES.'////10X'SIMULATED FLOW TABLE FOR'
     *       /15X,20A4///)
    6 FORMAT(10X,'THE SIMULATED FLOW TABLE WILL BE LONGER THAN 40 ROWS'/
     *       10X,'THE PARTIAL FLOW TABLE IS GIVEN BELOW'/)
    9 FORMAT(1H1)
   16 FORMAT(1H1,9X'NUMBER OF INPUTS = 'I2',  NUMBER OF OUTPUTS = 'I2
     *       ',  NUMBER OF MEMORIES = 'I2/10X'TO RUN A PROBLEM OF THIS '
     *       'SIZE, THE ARRAY SIZES MUST BE ALTERED.')
   17 FORMAT(1H1,9X'THE NUMBER OF MEMORIES IN COLUMN 'I2' = 'I2/10X
     *       'TO RUN A PROBLEM OF THIS SIZE, THE ARRAY SIZES MUST BE '
     *       'ALTERED.')
      READ(5,1) IDEN
      READ(5,2) NI, NO, NM
      ERROR = 0
      IF(NI .GT. 4) ERROR = 1
      IF(NO .GT. 6) ERROR = 1
      IF(NM .GT.36) ERROR = 1
      IF(ERROR .EQ.1.0) WRITE(6,16) NI, NO, NM
      NC = 2**NI
      READ(5,3)(Y(M),M=1,NM),(X(I),I=1,4)
      READ(5,4)((MC(JC,JM),JM=1,19),JC=1,NC)
      DO 19 JC=1,NC
   19 IF(MC(JC,1) .GT. 18) WRITE(6,17) JC, MC(JC,1)
      WRITE(6,5) NI, NO, NM, IDEN
      NIS = 2**NI
      CALL ASSIGN(NIS,1)
      DO 11 J=1,NC
      JC = J
      DO 10 I=1,NI
      IF(X(I) .NE. IG(I,J)) GO TO 11
   10 CONTINUE
      GO TO 12
   11 CONTINUE
   12 IR = 1
      CALL DIGEQN
      NR = 1
      NRS = 1
      DO 13 M=1,NM
   13 IY(M,NR) = Y(M)
      DO 14 J=1,NO
   14 IZ(J,NR) = Z(J)
      DO 15 I=1,40
      DO 15 J=1,NC
      KS(1,J) = 0
   15 S(I,J) = 0
      S(NR,JC) = 1001
   20 DO 21 I=1,NI
      IX(I,JC) = IG(1,JC)
   21 X(I) = IX(I,JC)
      DO 50 I=1,NI
      X(1) = NOT(X(I)I
      CALL DIGEQN
      NR = NR+1
      NRS = NRS+1
      DO 23 J=1,NC
      DO 22 II=1,NI
      IF(X(II) .NE. IG(II,J)) GO TO 23
   22 CONTINUE
      JC = J
      GO TO 24
   23 CONTINUE
   24 IF(NR .LT. 40) GO TO 25
      WRITE(6,6)
      CALL PRINT(NM)
      RETURN
   25 S(NR,JC) = NRS+1000
      S(IR,JC) = NRS
      DO 26 M=1,NM
   26 IY(M,NR) = Y(M)
      DO 30 J=1,NO
   30 IZ(J,NR) = Z(J)
      KS(NR,JC) = 1
   50 X(1) = NOT(X(I))
      CALL EQUIV(NM)
      DO 60 JR=2,NR
      DO 60 IC=1,NC
      IF(KS(JR,IC) .EQ. 0) GO TO 60
      KS(JR,IC) = 0
      IR = JR
      JC = IC
```

```
      DO 55 M=1,NM
   55 Y(M) = IY(M,IR)
      GO TO 20
   60 CONTINUE
      DO 100 I=1,NR
      DO 90 J=1,NC
      IS = S(I,J)-1000
      IF(IS .LT. 0) GO TO 90
      DO 80 II=1,NR
   80 IF(S(II,J) .EQ. IS) S(II,J) = -I
      S(I,J) = -(I+1000)
      GO TO 100
   90 CONTINUE
  100 CONTINUE
      DO 110 I=1,NR
      DO 110 J=1,NC
  110 S(I,J) = -S(I,J)
      CALL PRINT(NM)
      WRITE(6,9)
      RETURN
      END
```

```
      SUBROUTINE DIGEQN                                                 DIGEQN01
C     ******************************************************************DIGEQN02
C     *                                                                *DIGEQN03
C     *       SUBROUTINE DIGEQN CONTAINS THE OUTPUT AND SWITCHING      *DIGEQN04
C     *  EQUATIONS NECESSARY TO PERFORM SYSTEM SIMULATION.  THE USER   *DIGEQN05
C     *  MAY FIND IT ADVANTAGEOUS TO USE THE TOTAL INPUT STATE ARRAY   *DIGEQN06
C     *  ITS(JC), INSTEAD OF FORMING THESE STATES IN HIS EQUATIONS. THE*DIGEQN07
C     *  SWITCHING EQUATIONS ARE REPEATED TO ALLOW FOR ANY INTERNAL    *DIGEQN08
C     *  STATE SWITCHING OR CYCLING.                                   *DIGEQN09
C     *       THE USER SHOULD ONLY SUPPLY THE SWITCHING EQUATIONS AND  *DIGEQN10
C     *  THE OUTPUT EQUATIONS IN THIS SUBROUTINE.                      *DIGEQN11
C     *       FUNCTION NOT MAY BE USED TO PERFORM THE LOGICAL          *DIGEQN12
C     *  COMPLEMENT                                                    *DIGEQN13
C     *                                                                *DIGEQN14
C     *                                                                *DIGEQN15
C     *  X(I)    = THE CURRENT STATE OF THE I-TH INPUT                 *DIGEQN16
C     *  Y(M)    = THE CURRENT STATE OF THE M-TH MEMORY                *DIGEQN17
C     *  Z(J)    = THE CURRENT STATE OF THE J-TH OUTPUT                *DIGEQN18
C     *  ITS(JC)= THE TOTAL INPUT STATE FOR THE JC-TH COLUMN           *DIGEQN19
C     *  MS(M)  = THE "SET" SIGNAL FOR THE M-TH MEMORY                 *DIGEQN20
C     *  MR(M)  = THE "RESET" SIGNAL FOR THE M-TH MEMORY               *DIGEQN21
C     *                                                                *DIGEQN22
C     ******************************************************************DIGEQN23
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)  DIGEQN24
      COMMON /EQN/ X(4), Y(36), Z(6), KS(40,16), MC(16,19)              DIGEQN25
      DIMENSION ITS(16), MS(36), MR(36)                                 DIGEQN26
      INTEGER X, Y, Z, S                                                DIGEQN27
    1 FORMAT(10X,'SET AND RESET SIGNALS APPEARED SIMULTANEOUSLY'/       DIGEQN28
     *       10X,'X = '4I1'  Y = '18I1)                                 DIGEQN29
      ITS( 1) = NOT(X(1))*NOT(X(2))*NOT(X(3))*NOT(X(4))                 DIGEQN30
      ITS( 2) =     X(1) *NOT(X(2))*NOT(X(3))*NOT(X(4))                 DIGEQN31
      ITS( 3) =     X(1) *     X(2) *NOT(X(3))*NOT(X(4))                DIGEQN32
      ITS( 4) = NOT(X(1))*     X(2) *NOT(X(3))*NOT(X(4))                DIGEQN33
      ITS( 5) = NOT(X(1))*     X(2) *     X(3) *NOT(X(4))               DIGEQN34
      ITS( 6) =     X(1) *     X(2) *     X(3) *NOT(X(4))               DIGEQN35
      ITS( 7) =     X(1) *NOT(X(2))*     X(3) *NOT(X(4))                DIGEQN36
      ITS( 8) = NOT(X(1))*NOT(X(2))*     X(3) *NOT(X(4))                DIGEQN37
      ITS( 9) = NOT(X(1))*NOT(X(2))*     X(3) *     X(4)                DIGEQN38
      ITS(10) =     X(1) *NOT(X(2))*     X(3) *     X(4)                DIGEQN39
      ITS(11) =     X(1) *     X(2) *     X(3) *     X(4)               DIGEQN40
      ITS(12) = NOT(X(1))*     X(2) *     X(3) *     X(4)               DIGEQN41
      ITS(13) = NOT(X(1))*     X(2) *NOT(X(3))*     X(4)                DIGEQN42
      ITS(14) =     X(1) *     X(2) *NOT(X(3))*     X(4)                DIGEQN43
      ITS(15) =     X(1) *NOT(X(2))*NOT(X(3))*     X(4)                 DIGEQN44
      ITS(16) = NOT(X(1))*NOT(X(2))*NOT(X(3))*     X(4)                 DIGEQN45
   10 ICH = 0                                                           DIGEQN46
C                                                                       DIGEQN47
C     ----------------------ENTER SWITCHING EQUATIONS-------------------DIGEQN48
C                                                                       DIGEQN49
      MS(1) = ITS(2)*Y(2)*Y(3) + ITS(4)
      MR(1) = ITS(2)*Y(2)*NOT(Y(3))
      MS(2) = ITS(1) + ITS(3)*Y(4)
      MR(2) = ITS(3)*NOT(Y(4))
      MS(3) = ITS(1)*Y(1)
      MR(3) = ITS(1)*NOT(Y(1)) + ITS(3)*Y(4)
      MS(4) = ITS(2)*Y(2)*Y(3) + ITS(4)*Y(5)
      MR(4) = ITS(2)*Y(2)*NOT(Y(3)) + ITS(2)*NOT(Y(2)) +ITS(4)*NOT(Y(5))
      MS(5) = ITS(1)*Y(1) + ITS(3)*Y(4)
      MR(5) = ITS(3)*NOT(Y(4))
C                                                                       DIGEQN50
C     ------------------------------------------------------------------DIGEQN51
      DO 15 M=1,NM                                                      DIGEQN52
      MCH = Y(M)                                                        DIGEQN53
      Y(M) = MEMORY(Y(M),MS(M),MR(M))                                   DIGEQN54
      IF(MS(M)*MR(M) .EQ. 1) WRITE(6,1) (X(I),I=1,4),(Y(J),J=1,NM)      DIGEQN55
      IF(Y(M) .NE. MCH) ICH = 1                                         DIGEQN56
   15 CONTINUE                                                          DIGEQN57
      IF(ICH .EQ. 1) GO TO 10                                           DIGEQN58
C                                                                       DIGEQN59
C     ------------------------ENTER OUTPUT EQUATIONS--------------------DIGEQN60
C                                                                       DIGEQN61
      Z(1) = ITS(2)*NOT(Y(2)) + ITS(3)*NOT(Y(4)) + ITS(4)*NOT(Y(5))
C                                                                       DIGEQN62
C     ------------------------------------------------------------------DIGEQN63
      DO 20 J=1,NO                                                      DIGEQN64
      IF(Z(J) .GT. 1) Z(J) = 1                                          DIGEQN65
   20 CONTINUE                                                          DIGEQN66
      RETURN                                                            DIGEQN67
      END                                                               DIGEQN68
```

```
      SUBROUTINE EQUIV(KNM)
C     ****************************************************************
C     *                                                              *
C     *        SUBROUTINE EQUIV SENSES ANY REDUNDANT STATES IN THE   *
C     *  PRIMITIVE FLOW TABLE AND REPLACES THESE STATES WITH THEIR   *
C     *  EQUIVALENT STATES.  THIS ROUTINE IS USED WITH EITHER THE    *
C     *  SYNTHESIS OR SIMULATION PROGRAMS.                           *
C     *                                                              *
C     *                                                              *
C     *  KNM = THE NUMBER OF MEMORY ELEMENTS TO BE CHECKED FOR       *
C     *        EQUIVALENCE DURING SYSTEM SIMULATION.  KNM = 0 FOR    *
C     *        SYSTEM SYNTHESIS.                                     *
C     *  MC(JC,1) = THE NUMBER OF MEMORY ELEMENTS ASSOCIATED WITH THE *
C     *             JC-TH COLUMN                                     *
C     *  MC(JC,JM+1) = THE NUMBER DESIGNATION OF THE JM-TH MEMORY    *
C     *             IN THE JC-TH COLUMN                              *
C     ****************************************************************
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)
      COMMON /EQN/ X(4), Y(36), Z(6), KS(40,16), MC(16,19)
      DIMENSION IT(16)
      INTEGER S
    1 FORMAT(////)
    2 FORMAT(10X,'STATE 'I3' WAS EQUIVALENT TO STATE 'I3' AND HAS BEEN'
     *       ' REMOVED.')
      IF(NR .LE. 2) RETURN
      IF(KNM .EQ. 0) WRITE(6,1)
    7 IRC = 0
      DO 70 JC=1,NC
      NR1 = NR-1
      DO 40 I1=1,NR1
      I11 = I1+1
      IF(S(I1,JC) .LE. 1000) GO TO 40
      DO 30 I2=I11,NR
      IF(S(I2,JC) .LE. 1000) GO TO 30
      DO 10 J=1,NO
      IF(IZ(J,I1) .GT. 1) IZ(J,I1) = -1
      IF(IZ(J,I1).LT.0 .OR. IZ(J,I2).LT.0) GO TO 10
      IF(IZ(J,I1) .NE. IZ(J,I2)) GO TO 30
   10 CONTINUE
      IF(KNM .EQ. 0) GO TO 12
      NMC = MC(JC,1)
      IF(NMC .EQ. 0) GO TO 12
      DO 11 JM=1,NMC
      M1 = MC(JC,JM+1)
      IF(IY(M1,I1) .NE. IY(M1,I2)) GO TO 30
   11 CONTINUE
   12 DO 13 J=1,NC
      IF(J .EQ. JC) GO TO 13
      IT(J) = S(I1,J) + S(I2,J)
      IF(S(I1,J) .EQ. 0 .OR. S(I2,J) .EQ. 0) GO TO 13
      IF(S(I1,J) .NE. S(I2,J)) GO TO 30
      IT(J) = S(I1,J)
   13 CONTINUE
      IRC = 1
      IR = S(I2,JC) - 1000
      IS = S(I1,JC) - 1000
      IF(KNM .EQ. 0) WRITE(6,2) IR, IS
      IT(JC) = S(I1,JC)
      DO 14 I3=1,NR
   14 IF(S(I3,JC) .EQ. S(I2,JC)-1000) S(I3,JC) = S(I1,JC) -1000
      NR = NR-1
      IF(I2 .EQ. NR+1) GO TO 25
      DO 24 I4=I2,NR
      DO 20 J=1,NC
      S(I1,J) = IT(J)
      S(I4,J) = S(I4+1,J)
   20 IF(KNM .GT. 0) KS(I4,J) = KS(I4+1,J)
      DO 21 J=1,NO
   21 IZ(J,I4) = IZ(J,I4+1)
      IF(KNM .EQ. 0) GO TO 23
      DO 22 M=1,KNM
   22 IY(M,I4) = IY(M,I4+1)
   23 CONTINUE
   24 CONTINUE
   25 DO 26 J=1,NC
      S(NR+1,J) = 0
   26 KS(NR+1,J) = 0
   30 CONTINUE
   40 CONTINUE
   70 CONTINUE
      IF(IRC.NE.0 .AND. KNM.EQ.0) GO TO 7
      RETURN
      END
```

```
      SUBROUTINE CANON                                                  LSP0420
C     ******************************************************************LSP0421
C     *                                                                *LSP0422
C     *       SUBROUTINE CANON TAKES THE PRIMITIVE FLOW TABLE FROM     *LSP0423
C     *  THE SYNTHESIS PROGRAM AND REORDERS THE ROWS ACCORDING TO      *LSP0424
C     *  SYSTEMATIC INPUT CHANGES AS OUTLINED IN THE THESIS.  THE STATE*LSP0425
C     *  NUMBERS ARE THEN RESEQUENCED TO PRODUCE THE CANONICAL FLOW    *LSP0426
C     *  TABLE.                                                        *LSP0427
C     *                                                                *LSP0428
C     ******************************************************************LSP0429
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)  LSP0430
      COMMON /EQN/ X(4), Y(36), Z(6), KS(40,16), MC(16,19)              LSP0431
      INTEGER S, X                                                      LSP0432
    1 FORMAT(10X'THERE IS NO STABLE STATE IN THE FIRST ROW.')           LSP0433
      DO 10 I=1,NR                                                      LSP0434
      DO 10 J=1,NC                                                      LSP0435
   10 KS(I,J) = 0                                                       LSP0436
      DO 11 J=1,NC                                                      LSP0437
      IF(S(1,J) .LT. 1000) GO TO 11                                     LSP0438
      IR = 1                                                            LSP0439
      IRR= 1                                                            LSP0440
      JC = J                                                            LSP0441
      GO TO 20                                                          LSP0442
   11 CONTINUE                                                          LSP0443
      WRITE(6,1)                                                        LSP0444
   20 DO 22 I=1,NI                                                      LSP0445
   22 X(I) = IX(I,JC)                                                   LSP0446
      DO 50 I=1,NI                                                      LSP0447
      X(I) = NOT(X(I))                                                  LSP0448
      DO 30 J1=1,NC                                                     LSP0449
      DO 23 I1=1,NI                                                     LSP0450
      IF(X(I1) .NE. IX(I1,J1)) GO TO 30                                 LSP0451
   23 CONTINUE                                                          LSP0452
      IF(S(IRR,J1) .EQ. 0) GO TO 50                                     LSP0453
      IRI = IR+1                                                        LSP0454
      DO 26 IR1=IRI,NR                                                  LSP0455
      IF(S(IR1,J1)-1000 .NE. S(IRR,J1)) GO TO 26                        LSP0456
      IR = IR+1                                                         LSP0457
      KS(IR,J1) = 1                                                     LSP0458
      DO 24 JC1=1,NC                                                    LSP0459
      ST = S(IR,JC1)                                                    LSP0460
      S(IR,JC1) = S(IR1,JC1)                                            LSP0461
   24 S(IR1,JC1) = ST                                                   LSP0462
      DO 25 JZ=1,NO                                                     LSP0463
      ZT = IZ(JZ,IR)                                                    LSP0464
      IZ(JZ,IR) = IZ(JZ,IR1)                                            LSP0465
   25 IZ(JZ,IR1) = ZT                                                   LSP0466
      GO TO 31                                                          LSP0467
   26 CONTINUE                                                          LSP0468
   30 CONTINUE                                                          LSP0469
   31 IF(IR .GE. NR) GO TO 61                                           LSP0470
   50 X(I) = NOT(X(I))                                                  LSP0471
      DO 60 I2=2,NR                                                     LSP0472
      DO 60 J2=1,NC                                                     LSP0473
      IF(KS(I2,J2) .EQ. 0) GO TO 60                                     LSP0474
      KS(I2,J2) = 0                                                     LSP0475
      IRR = I2                                                          LSP0476
      JC = J2                                                           LSP0477
      GO TO 20                                                          LSP0478
   60 CONTINUE                                                          LSP0479
   61 DO 100 I=1,NR                                                     LSP0480
      DO 90 J=1,NC                                                      LSP0481
      IS = S(I,J)-1000                                                  LSP0482
      IF(IS .LT. 0) GO TO 90                                            LSP0483
      DO 80 I1=1,NR                                                     LSP0484
   80 IF(S(I1,J) .EQ. IS) S(I1,J) = -I                                  LSP0485
      S(I,J) = -(I+1000)                                                LSP0486
      GO TO 100                                                         LSP0487
   90 CONTINUE                                                          LSP0488
  100 CONTINUE                                                          LSP0489
      DO 110 I=1,NR                                                     LSP0490
      DO 110 J=1,NC                                                     LSP0491
  110 S(I,J) = -S(I,J)                                                  LSP0492
      RETURN                                                            LSP0493
      END                                                               LSP0494
```

```
      SUBROUTINE OUTPUT
C     ******************************************************************
C     *                                                                *
C     *       SUBROUTINE OUTPUT DETERMINES THE MEMORY REQUIREMENTS,    *
C     *  PRINTS THE STATE SIGNALS, THE SWITCHING CONDITIONS, AND THE   *
C     *  OUTPUT SIGNALS.                                               *
C     *                                                                *
C     *                                                                *
C     *  SET(K,M)   = THE M-TH STATE SIGNAL USED TO SET THE K-TH       *
C     *               MEMORY ELEMENT                                   *
C     *  RESET(K,M) = THE M-TH STATE SIGNAL USED TO RESET THE K-TH     *
C     *               MEMORY ELEMENT                                   *
C     *  PS(M)      = THE M-TH PREVIOUS STATE TO A STABLE STATE        *
C     *  SET(K,1)   = THE NUMBER OF "SET" SIGNALS FOR THE K-TH MEMORY  *
C     *  RESET(K,1) = THE NUMBER OF "RESET" SIGNALS FOR THE K-TH MEMORY*
C     *  MC(JC,1)   = THE NUMBER OF MEMORY ELEMENTS IN THE JC-TH COLUMN*
C     *                                                                *
C     *  IN THE RESULTING PRINT-OUT:                                   *
C     *       - THE STATE SIGNALS ARE TO BE SUBSTITUTED FOR THE STATE  *
C     *         NUMBERS IN THE SWITCHING AND OUTPUT EQUATIONS          *
C     *       - "*" IMPLIES THE LOGICAL "AND"                          *
C     *       - "+" IMPLIES THE LOGICAL "OR"                           *
C     *                                                                *
C     ******************************************************************
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)
      COMMON /OUT/ SSC(16,21), SSR(40)
      COMMON /EQN/ X(4), Y(36), Z(6), KS(40,16), MC(16,19)
      COMMON /ASN/ IG(19,20)
      DIMENSION SET(36,20), RESET(36,20), PS(20), IZS(6,40), JL(19),
     *          IYP(5,20), INOT(5,20), NY(5), IO(40)
      INTEGER S, SSC, SSR, SET, RESET, PS
      DATA IAB,IAN/1H ,1H_/, NY/5*4H * Y/, IO/1H ,39*1H+/
    3 FORMAT(//10X'(PASSIVE MEMORY ASSIGNMENT)'/)
    4 FORMAT(//10X'STATE SIGNALS:'/)
    5 FORMAT(15X,1H(,I2,4H)  =,4X, I1,12(A4,I2))
    6 FORMAT(15X,1H(,I2,4H)  =,3X,2I1,12(A4,I2))
    7 FORMAT(15X,1H(,I2,4H)  =,2X,3I1,12(A4,I2))
    8 FORMAT(15X,1H(,I2,4H)  =,1X,4I1,12(A4,I2))
    9 FORMAT(1H+,26X,(16(3X,A1,2X)/))
   10 FORMAT(10X'OUTPUT SIGNALS:'/)
   11 FORMAT(15X,'Z',I1,' =' 13(I3,1X,A1),/(19X,13(I3,1X,A1)))
   12 FORMAT(10X'SWITCHING CONDITIONS:'/)
   13 FORMAT(15X'Y',I2,6X'SET   =',10(I3,1X,A1),/(21X,10(I3,1X,A1)))
   14 FORMAT(24X,'RESET =',10(I3,1X,A1),/(21X,10(I3,1X,A1)))
   15 FORMAT(10X'THERE ARE NO TRANSITION PATHS TO STATE',I4)
   16 FORMAT(1X)
      NM = 0
      DO 20 J=1,NC
      MC(J,1) = LOG(SSC(J,1),2)
   20 NM = NM + MC(J,1)
      IF(NM .EQ. 0) GO TO 26
      WRITE(6,3)
      DO 25 K=1,NM
      SET(K,1) = 0
   25 RESET(K,1) = 0
   26 LM = 0
      MCL = 1
      MC1 = 0
      WRITE(6,4)
      DO 170 JC=1,NC
      MC1 = MC1 + MC(JC,1)
      L = SSC(JC,1)
      IF(L-1)170,160,30
   30 CALL ASSIGN(L,2)
      DO 100 I=1,L
      M1 = 0
      DO 35 IR=1,NR
      IF(S(IR,JC) .NE. SSC(JC,I+1)) GO TO 35
      M1 = M1+1
      PS(M1) = SSR(IR)
   35 CONTINUE
      IF(M1 .EQ. 0) WRITE(6,15) SSC(JC,I+1)
      J1 = 0
      DO 90 K=MCL,MC1
      JK = K-MCL+1
      IF(IG(JK,I) .LT. 0) GO TO 90
      J1 = J1+1
      INOT(J1,I) = IAB
      IYP(J1,I) = K
      IF(IG(JK,I) .EQ. 0) INOT(J1,I) = IAN
      NS1 = SET(K,1)
      NR1 = RESET(K,1)
      IF(IG(JK,I) .EQ. 0) GO TO 83
      SET(K,1) = NS1 + M1
      DO 82 M=1,M1
   82 SET(K,1+M+NS1) = PS(M)
      GO TO 85
   83 RESET(K,1) = NR1 + M1
      DO 84 M=1,M1
   84 RESET(K,1+M+NR1) = PS(N)
   85 CONTINUE
   90 CONTINUE
      JL(I) = J1
  100 CONTINUE
      DO 155 KK=1,L
      J1 = JL(KK)
      GO TO (150,151,152,153),NI
  150 WRITE(6,5) SSC(JC,KK+1),(IX(I,JC),I=1,NI),(NY(K),IYP(K,KK),K=1,J1)
      WRITE(6,9)(INOT(M,KK),M=1,J1)
      GO TO 155
  151 WRITE(6,6) SSC(JC,KK+1),(IX(I,JC),I=1,NI),(NY(K),IYP(K,KK),K=1,J1)
      WRITE(6,9)(INOT(M,KK),M=1,J1)
      GO TO 155
  152 WRITE(6,7) SSC(JC,KK+1),(IX(I,JC),I=1,NI),(NY(K),IYP(K,KK),K=1,J1)
      WRITE(6,9)(INOT(M,KK),M=1,J1)
      GO TO 155
  153 WRITE(6,8) SSC(JC,KK+1),(IX(I,JC),I=1,NI),(NY(K),IYP(K,KK),K=1,J1)
      WRITE(6,9)(INOT(M,KK),M=1,J1)
  155 CONTINUE
      WRITE(6,16)
      GO TO 170
  160 IF(NI .LE. 2) WRITE(6,6) SSC(JC,2),(IX(I,JC),I=1,NI)
      IF(NI .EQ. 3) WRITE(6,7) SSC(JC,2),(IX(I,JC),I=1,NI)
      IF(NI .EQ. 4) WRITE(6,8) SSC(JC,2),(IX(I,JC),I=1,NI)
      WRITE(6,16)
  170 MCL = MCL + MC(JC,1)
      IF(NM .EQ. 0) GO TO 176
      WRITE(6,12)
      DO 175 K=1,NM
      M1 = SET(K,1)
```

```
      WRITE(6,13) K, (SET(K,M+1), ID(MI+1-M),M=1,MI)                    LSP0615
      MI = RESET(K,1)                                                    LSP0616
      WRITE(6,14) (RESET(K,M+1), ID(MI+1-M),M=1,MI)                     LSP0617
  175 WRITE(6,16)                                                        LSP0618
  176 WRITE(6,10)                                                        LSP0619
      DO 185 J=1,ND                                                      LSP0620
      JZ = 0                                                             LSP0621
      DO 180 IR=1,NR                                                     LSP0622
      IF(IZ(J,IR) .NE. 1) GO TO 180                                      LSP0623
      JZ = JZ+1                                                          LSP0624
      IZS(J,JZ) = SSR(IR)                                                LSP0625
  180 CONTINUE                                                           LSP0626
      WRITE(6,11) J,(IZS(J,K),ID(JZ+1-K),K=1,JZ)                         LSP0627
  185 CONTINUE                                                           LSP0628
      RETURN                                                             LSP0629
      END                                                                LSP0630
```

```
      SUBROUTINE PRINT(KNM)                                             LSP0631
C     ******************************************************************LSP0632
C     *                                                                *LSP0633
C     *       SUBROUTINE PRINT IS USED TO PRINT THE FLOW TABLES        *LSP0634
C     *  INVOLVED IN EITHER SYNTHESIS OR SIMULATION.                   *LSP0635
C     *                                                                *LSP0636
C     ******************************************************************LSP0637
      COMMON /ALL/ NI,NO,NR,NM,NC,IX(4,16),IY(36,40),IZ(6,40),S(40,16)  LSP0638
      DIMENSION MS(16), MSS(16), IN3(41), IXP(4), IZP(6)                LSP0639
      INTEGER S                                                         LSP0640
      DATA IAB, IAP, IN3, IXP, IZP/4H    ,4H(   ),41*3H---,4*1HX,6*1HZ/ LSP0641
    1 FORMAT(11X,4(1X,A1,I1)/)                                          LSP0642
    2 FORMAT(10X,2(5X, I1),3X,6(1X,A1,I1))                              LSP0643
    3 FORMAT(10X,4(4X,2I1),3X,6(1X,A1,I1))                              LSP0644
    4 FORMAT(10X,8(3X,3I1),3X,6(1X,A1,I1))                              LSP0645
    5 FORMAT(    16(2X,4I1),3X,6(1X,A1,I1))                             LSP0646
    6 FORMAT(12X,41A3)                                                  LSP0647
    7 FORMAT(2X,41A3)                                                   LSP0648
    8 FORMAT(10X,2I6,3X,6(2X,I1))                                       LSP0649
    9 FORMAT(10X,4I6,3X,6(2X,I1))                                       LSP0650
   10 FORMAT(10X,8I6,3X,6(2X,I1))                                       LSP0651
   11 FORMAT(    16I6,3X,6(2X,I1))                                      LSP0652
   12 FORMAT(1H+, 50X,16I2)                                             LSP0653
   13 FORMAT(1H+, 62X,16I2)                                             LSP0654
   14 FORMAT(1H+, 86X,16I2)                                             LSP0655
   15 FORMAT(1H+,115X,16I1)                                             LSP0656
   16 FORMAT(1H+,10X,1H|,1X,A4,3(2X,A4),2H |)                           LSP0657
   17 FORMAT(1H+,10X,1H|,1X,A4,7(2X,A4),2H |)                           LSP0658
   18 FORMAT(3H+| ,A4,15(2X,A4),2H |)                                   LSP0659
      INC = 1+NC*2+NO                                                   LSP0660
      WRITE(6,1)(IXP(I),I,I=1,NI)                                       LSP0661
      GO TO (20,25,30,35), NI                                           LSP0662
   20 WRITE(6,2)((IX(I,JC),I=1,NI),JC=1,NC),(IZP(J),J,J=1,NO)           LSP0663
      WRITE(6,6)(IN3(I),I=1,INC)                                        LSP0664
      DO 22 IR=1,NR                                                     LSP0665
      DO 21 JC=1,NC                                                     LSP0666
      MS(JC) = S(IR,JC)                                                 LSP0667
      MSS(JC)= IAB                                                      LSP0668
      IF(S(IR,JC) .LE. 1000) GO TO 21                                   LSP0669
      MS(JC) = S(IR,JC) - 1000                                          LSP0670
      MSS(JC)= IAP                                                      LSP0671
   21 CONTINUE                                                          LSP0672
      WRITE(6, 8)(MS(JC),JC=1,NC),(IZ(J,IR),J=1,NO)                     LSP0673
      IF(KNM .NE. 0) WRITE(6,12)(IY(M,IR),M=1,KNM)                      LSP0674
   22 WRITE(6,16)(MSS(JC),JC=1,NC)                                      LSP0675
      GO TO 40                                                          LSP0676
   25 WRITE(6,3)((IX(I,JC),I=1,NI),JC=1,NC),(IZP(J),J,J=1,NO)           LSP0677
      WRITE(6,6)(IN3(I),I=1,INC)                                        LSP0678
      DO 27 IR=1,NR                                                     LSP0679
      DO 26 JC=1,NC                                                     LSP0680
      MS(JC) = S(IR,JC)                                                 LSP0681
      MSS(JC)= IAB                                                      LSP0682
      IF(S(IR,JC) .LE. 1000) GO TO 26                                   LSP0683
      MS(JC) = S(IR,JC) - 1000                                          LSP0684
      MSS(JC)= IAP                                                      LSP0685
   26 CONTINUE                                                          LSP0686
      WRITE(6, 9)(MS(JC),JC=1,NC),(IZ(J,IR),J=1,NO)                     LSP0687
      IF(KNM .NE. 0) WRITE(6,13)(IY(M,IR),M=1,KNM)                      LSP0688
   27 WRITE(6,16)(MSS(JC),JC=1,NC)                                      LSP0689
      GO TO 40                                                          LSP0690
   30 WRITE(6,4)((IX(I,JC),I=1,NI),JC=1,NC),(IZP(J),J,J=1,NO)           LSP0691
      WRITE(6,6)(IN3(I),I=1,INC)                                        LSP0692
      DO 32 IR=1,NR                                                     LSP0693
      DO 31 JC=1,NC                                                     LSP0694
      MS(JC) = S(IR,JC)                                                 LSP0695
      MSS(JC)= IAB                                                      LSP0696
      IF(S(IR,JC) .LE. 1000) GO TO 31                                   LSP0697
      MS(JC) = S(IR,JC) - 1000                                          LSP0698
      MSS(JC)= IAP                                                      LSP0699
   31 CONTINUE                                                          LSP0700
      WRITE(6,10)(MS(JC),JC=1,NC),(IZ(J,IR),J=1,NO)                     LSP0701
      IF(KNM .NE. 0) WRITE(6,14)(IY(M,IR),M=1,KNM)                      LSP0702
   32 WRITE(6,17)(MSS(JC),JC=1,NC)                                      LSP0703
      GO TO 40                                                          LSP0704
   35 WRITE(6,5)((IX(I,JC),I=1,NI),JC=1,NC),(IZP(J),J,J=1,NO)           LSP0705
      WRITE(6,7)(IN3(I),I=1,INC)                                        LSP0706
      DO 37 IR=1,NR                                                     LSP0707
      DO 36 JC=1,NC                                                     LSP0708
      MS(JC) = S(IR,JC)                                                 LSP0709
      MSS(JC)= IAB                                                      LSP0710
      IF(S(IR,JC) .LE. 1000) GO TO 36                                   LSP0711
      MS(JC) = S(IR,JC) - 1000                                          LSP0712
      MSS(JC)= IAP                                                      LSP0713
   36 CONTINUE                                                          LSP0714
      WRITE(6,11)(MS(JC),JC=1,NC),(IZ(J,IR),J=1,NO)                     LSP0715
      IF(KNM .NE. 0) WRITE(6,15)(IY(M,IR),M=1,KNM)                      LSP0716
   37 WRITE(6,18)(MSS(JC),JC=1,NC)                                      LSP0717
      WRITE(6,7)(IN3(I),I=1,INC)                                        LSP0718
      RETURN                                                            LSP0719
   40 WRITE(6,6)(IN3(I),I=1,INC)                                        LSP0720
      RETURN                                                            LSP0721
      END                                                               LSP0722
```

```
      SUBROUTINE ASSIGN(K,NA)
C     ******************************************************************
C     *                                                                *
C     *      SUBROUTINE ASSIGN PRODUCES EITHER THE PASSIVE MEMORY OR   *
C     *  GRAY ASSIGNMENT CODE.  THE PASSIVE MEMORY ASSIGNMENT CODE IS  *
C     *  USED FOR MEMORY ASSIGNMENT IN THE SYNTHESIS PROGRAM AND THE   *
C     *  GRAY CODE IS USED FOR INPUT STATES IN THE SIMULATION PROGRAM. *
C     *  THE GRAY CODE PRODUCED BY THIS SUBROUTINE HAS THE ELEMENT ON  *
C     *  THE LEFT MOST FREQUENTLY CHANGING.                            *
C     *                                                                *
C     *                                                                *
C     *  NA = OPTION SPECIFYING CODE                                   *
C     *       1 = GRAY CODE (INPUT STATES)                             *
C     *       2 = PASSIVE CODE (MEMORY ASSIGNMENT)                     *
C     *  K  = THE NUMBER OF STATES(OR COLUMNS) REQUIRED                *
C     *  IG(1,J) = THE VALUE OF THE I-TH MEMORY (OR INPUT) IN THE      *
C     *            J-TH STATE (OR COLUMN).                             *
C     *                                                                *
C     ******************************************************************
      COMMON /ASN/ IG(19,20)
      DIMENSION IA(16), RA(16)
      INTEGER RA
      IF(K .LE. 1) RETURN
      NM1 = LOG(K,NA)
      IF(NA .EQ. 2) GO TO 30
   10 DO 11 I=1,K
      DO 11 J=1,NM1
      IG(J,I) = 0
      JM = 2**(NM1-J)
      DO 11 M=1,JM,2
      AM = M
      A1 = 2**J*(AM+0.5)
      A2 = 2**J*(AM-0.5)
      IF(I.GT.A1 .OR. I.LE.A2) GO TO 11
      IG(J,I) = 1
   11 CONTINUE
      RETURN
   30 DO 31 I=1,19
      DO 31 J=1,20
   31 IG(I,J) = -1
      NRA = 1
      NR = 1
   32 M = 1
      DO 33 I=1,NRA
   33 IA(I) = 0
      IF(NRA .GT. 1) IA(1) = 1
   34 M = M*2
      DO 35 J=1,M,2
   35 IA(1+J*NRA/M) = IA(1+(J-1)*NRA/M) + M/2
      IF(M .LT. NRA) GO TO 34
      DO 37 I=I,NRA
      IK = 0
      IR1 = I
      DO 36 I4=1,IR1
   36 IF(IA(I4) .LT. IA(I)) IK = IK+1
   37 RA(I) = IA(I) + IK
      NRR = NR+NRA
      DO 39 I=1,NRA
      IM = NR-1+I
      IR = RA(I)
      DO 38 I1=IR,NRR
      I2 = NRR+IR-I1
      DO 38 M1=1,IM
   38 IG(M1,I2+1) = IG(M1,I2)
      IG(IM,IR) = 1
      IG(IM,IR+1) = 0
      IF(NR+I .EQ. K) GO TO 40
   39 CONTINUE
      NR = NRR
      NRA = NRA*2
      GO TO 32
   40 CONTINUE
      RETURN
      END
```

```
      FUNCTION LOG(K,NA)                                                LSP0797
C     *******************************************************************LSP0798
C     *                                                                 *LSP0799
C     *        FUNCTION LOG DETERMINES THE NUMBER OF MEMORIES           *LSP0800
C     *  (OR INPUTS) REQUIRED FOR K ROWS (OR COLUMNS) USING THE PASSIVE*LSP0801
C     *  MEMORY (OR GRAY) ASSIGNMENT CODE.                              *LSP0802
C     *                                                                 *LSP0803
C     *                                                                 *LSP0804
C     *  NA - OPTION SPECIFYING CODE                                    *LSP0805
C     *       1 = GRAY CODE (INPUT STATES)                              *LSP0806
C     *       2 = PASSIVE MEMORY CODE (MEMORY STATES)                   *LSP0807
C     *  K  = THE NUMBER OF ROWS (OR COLUMNS).                          *LSP0808
C     *                                                                 *LSP0809
C     *******************************************************************LSP0810
      LOG = 0                                                           LSP0811
      IF(NA .EQ. 2) GO TO 30                                            LSP0812
   10 KK = 1                                                            LSP0813
      DO 11 I=1,10                                                      LSP0814
      IF(KK .LT. K) LOG = I                                             LSP0815
   11 KK = KK*2                                                         LSP0816
      RETURN                                                            LSP0817
   30 IF(K .GE. 2) LOG = K-1                                            LSP0818
      RETURN                                                            LSP0819
      END                                                               LSP0820
```

```
      FUNCTION NOT(IA)                                                  LSP0821
C     *******************************************************************LSP0822
C     *                                                                 *LSP0823
C     *        FUNCTION NOT PERFORMS THE LOGICAL COMPLEMENT OF THE      *LSP0824
C     *  VARIABLE IA.                                                   *LSP0825
C     *                                                                 *LSP0826
C     *******************************************************************LSP0827
      IF(IA .GE. 1) NOT = 0                                             LSP0828
      IF(IA .EQ. 0) NOT = 1                                             LSP0829
      RETURN                                                            LSP0830
      END                                                               LSP0831
```

```
      FUNCTION MEMORY(IY,MS,MR)
C     ******************************************************************
C     *                                                                *
C     *       FUNCTION MEMORY CONTAINS THE SIMPLIFIED MEMORY EQUATION. *
C     *                                                                *
C     *                                                                *
C     *  MS = SET SIGNAL                                               *
C     *  MR = RESET SIGNAL                                             *
C     *  IY = PREVIOUS MEMORY STATE                                    *
C     *                                                                *
C     ******************************************************************
      MEMORY = MS + IY*NOT(MR)
      IF(MEMORY .GT. 1) MEMORY = 1
      RETURN
      END

$ENTRY
```

APPENDIX C

EXAMPLE COMPUTER SOLUTIONS

```
$JOB
      CALL LOGSYN
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.1 - TABLE XI REPRESENTING EXAMPLE 3.3
 2 210
00  10  11  01
1001   2       7                                                00
   81002   3                                                    00
       41003   7                                                00
   81004   5                                                    00
       91005   6                                                00
   1      101006                                                10
   8      101007                                                01
1008   2       7                                                01
   81009  10                                                    01
       91010   7                                                01
```

LOGIC SYNTHESIS
FOR 2 INPUTS, 2 OUTPUTS.

ORIGINAL PRIMITIVE FLOW TABLE FOR
EXAMPLE C.1 - TABLE XI REPRESENTING EXAMPLE 3.3

X1 X2 00	10	11	01	Z1	Z2
(1)	2	0	7	0	0
8	(2)	3	0	0	0
0	4	(3)	7	0	0
8	(4)	5	0	0	0
0	9	(5)	6	0	0
1	0	10	(6)	1	0
8	0	10	(7)	0	1
(8)	2	0	7	0	1
8	(9)	10	0	0	1
0	9	(10)	7	0	1

CANONICAL FLOW TABLE FOR
EXAMPLE C.1 - TABLE XI REPRESENTING EXAMPLE 3.3

X1 X2 00	10	11	01	Z1	Z2
(1)	2	0	3	0	0
4	(2)	5	0	0	0
4	0	6	(3)	0	1
(4)	2	0	3	0	1
0	7	(5)	3	0	0
0	8	(6)	3	0	1
4	(7)	9	0	0	0
4	(8)	6	0	0	1
0	8	(9)	10	0	0
1	0	6	(10)	1	0

(PASSIVE MEMORY ASSIGNMENT)

STATE SIGNALS:

(1) = 00 * Y 1
(4) = 00 * $\underline{Y}$ 1

(2) = 10 * Y 2 * Y 3
(7) = 10 * Y 2 * $\underline{Y}$ 3
(8) = 10 * $\underline{Y}$ 2

(5) = 11 * Y 4 * Y 5
(6) = 11 * Y 4 * $\underline{Y}$ 5
(9) = 11 * $\underline{Y}$ 4

(3) = 01 * Y 6
(10) = 01 * $\underline{Y}$ 6

SWITCHING CONDITIONS:

Y 1 SET = 10
RESET = 2 + 3 + 7 + 8

Y 2 SET = 1 + 4 + 5
RESET = 6 + 9

Y 3 SET = 1 + 4
RESET = 5

Y 4 SET = 2 + 3 + 8 + 10
RESET = 7

Y 5 SET = 2
RESET = 3 + 8 + 10

Y 6 SET = 1 + 4 + 5 + 6
RESET = 9

OUTPUT SIGNALS:

Z1 = 10
Z2 = 3 + 4 + 6 + 8

```
$JOB
      CALL LOGSYN
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.2 - TABLE 6.4 FROM FLUID LOGIC TEXT BY E.C. FITCH
 2 112
00  10  11  01
1001   2       7                                                0
   81002   3                                                    0
       41003   9                                                0
  101004   5                                                    0
      111005   6                                                1
   1      121006                                                1
   1        1007                                                0
1008   2                                                        0
           31009                                                0
1010   4                                                        0
    1011   5                                                    1
        1012   6                                                1
```

LOGIC SYNTHESIS
FOR 2 INPUTS, 1 OUTPUTS.

ORIGINAL PRIMITIVE FLOW TABLE FOR
EXAMPLE C.2 - TABLE 6.4 FROM FLUID LOGIC TEXT BY E.C. FITCH

X1 X2

00	10	11	01	Z1
(1)	2	0	7	0
8	(2)	3	0	0
0	4	(3)	9	0
10	(4)	5	0	0
0	11	(5)	6	1
1	0	12	(6)	1
1	0	0	(7)	0
(8)	2	0	0	0
0	0	3	(9)	0
(10)	4	0	0	0
0	(11)	5	0	1
0	0	(12)	6	1

STATE 8 WAS EQUIVALENT TO STATE 1 AND HAS BEEN REMOVED.
STATE 12 WAS EQUIVALENT TO STATE 5 AND HAS BEEN REMOVED.
STATE 9 WAS EQUIVALENT TO STATE 7 AND HAS BEEN REMOVED.

CANONICAL FLOW TABLE FOR
EXAMPLE C.2 - TABLE 6.4 FROM FLUID LOGIC TEXT BY E.C. FITCH

X1 X2

00	10	11	01	Z1
(1)	2	0	3	0
1	(2)	4	0	0
1	0	4	(3)	0
0	5	(4)	3	0
6	(5)	7	0	0
(6)	5	0	0	0
0	9	(7)	8	1
1	0	7	(8)	1
0	(9)	7	0	1

(PASSIVE MEMORY ASSIGNMENT)

STATE SIGNALS:

(1) = 00 * Y 1
(6) = 00 * $\underline{Y}$ 1

(2) = 10 * Y 2 * Y 3
(5) = 10 * Y 2 * $\underline{Y}$ 3
(9) = 10 * $\underline{Y}$ 2

(4) = 11 * Y 4
(7) = 11 * $\underline{Y}$ 4

(3) = 01 * Y 5
(8) = 01 * $\underline{Y}$ 5

SWITCHING CONDITIONS:

Y 1 SET = 2 + 3 + 8
RESET = 5

Y 2 SET = 1 + 4 + 6
RESET = 7

Y 3 SET = 1
RESET = 4 + 6

Y 4 SET = 2 + 3
RESET = 5 + 8 + 9

Y 5 SET = 1 + 4
RESET = 7

OUTPUT SIGNALS:

Z1 = 7 + 8 + 9

```
$JOB
     CALL LOGSYN
     STOP
     END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.3 - 4-INPUT, 30 ROW
 4 630
00001000110001000110111010100010001110111111101110101110110010001
1001   2      10              13                              14000000
   11002   3              15                              16    100000
       21003  10       4                              17        110000
           3      181004  15               5                    111000
                       4              191005   6      17        111100
                  18               7       51006  11            011100
                              131007  19       6               8001100
   9                               7              11      161008000110
1009  20      10              13                                000000
   1       31010  18                              11            010000
              10                               61011  17      12010100
   1                               7              11      161012000111
   1              18      151013   7                            001000
   1                               7              11      161014000111
       2               41015  13      19                        101000
       2                              19              171016  12100100
           3                               5      111017  16    110100
              101018   4      13               6                011000
                          15       71019   5              16    101100
  211020   3              15                              16    100000
1021  22      10              13                              14000000
  231022   3              15                              16    100000
1023  24      10              13                              14000000
  251024   3              15                              16    100000
1025  20      10              13                              26000000
   9                              27              11      161026000100
                              281027  19       6               8001100
   1              18      291028   7                            101000
      30               41029  13      19                        101000
   11030   3              15                              16    100010
```

LOGIC SYNTHESIS
FOR 4 INPUTS, 6 OUTPUTS.

ORIGINAL PRIMITIVE FLOW TABLE FOR
EXAMPLE C.3 - 4-INPUT, 30 ROW

X1 X2 X3 X4

0000	1000	1100	0100	0110	1110	1010	0010	0011	1011	1111	0111	0101	1101	1001	0001	Z1	Z2	Z3	Z4	Z5	Z6
(1)	2	0	10	0	0	0	13	0	0	0	0	0	0	0	14	0	0	0	0	0	0
1	(2)	3	0	0	0	15	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0
0	2	(3)	10	0	4	0	0	0	0	0	0	0	17	0	0	1	1	0	0	0	0
0	0	3	0	18	(4)	15	0	0	0	5	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	4	0	0	0	19	(5)	6	0	17	0	0	1	1	1	1	0	0
0	0	0	0	18	0	0	0	7	0	5	(6)	11	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	13	(7)	19	0	6	0	0	0	8	0	0	1	1	0	0
9	0	0	0	0	0	0	0	7	0	0	0	11	0	16	(8)	0	0	0	1	1	0
(9)	20	0	10	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	3	(10)	18	0	0	0	0	0	0	0	11	0	0	0	0	1	0	0	0	0
0	0	0	10	0	0	0	0	0	0	0	6	(11)	17	0	12	0	1	0	1	0	0
1	0	0	0	0	0	0	0	7	0	0	0	11	0	16	(12)	0	0	0	1	1	1
1	0	0	0	18	0	15	(13)	7	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	7	0	0	0	11	0	16	(14)	0	0	0	1	1	1
0	2	0	0	0	4	(15)	13	0	19	0	0	0	0	0	0	1	0	1	0	0	0
0	2	0	0	0	0	0	0	0	19	0	0	0	17	(16)	12	1	0	0	1	0	0
0	0	3	0	0	0	0	0	0	0	5	0	11	(17)	16	0	1	1	0	1	0	0
0	0	0	10	(18)	4	0	13	0	0	0	6	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	15	0	7	(19)	5	0	0	0	16	0	1	0	1	1	0	0
21	(20)	3	0	0	0	15	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0
(21)	22	0	10	0	0	0	13	0	0	0	0	0	0	0	14	0	0	0	0	0	0
23	(22)	3	0	0	0	15	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0
(23)	24	0	10	0	0	0	13	0	0	0	0	0	0	0	14	0	0	0	0	0	0
25	(24)	3	0	0	0	15	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0
(25)	20	0	10	0	0	0	13	0	0	0	0	0	0	0	26	0	0	0	0	0	0
9	0	0	0	0	0	0	0	27	0	0	0	11	0	16	(26)	0	0	0	1	0	0
0	0	0	0	0	0	0	28	(27)	19	0	6	0	0	0	8	0	0	1	1	0	0
1	0	0	0	18	0	29	(28)	7	0	0	0	0	0	0	0	1	0	1	0	0	0
0	30	0	0	0	4	(29)	13	0	19	0	0	0	0	0	0	1	0	1	0	0	0
1	(30)	3	0	0	0	15	0	0	0	0	0	0	0	16	0	1	0	0	0	1	0

STATE 25 WAS EQUIVALENT TO STATE 9 AND HAS BEEN REMOVED.
STATE 14 WAS EQUIVALENT TO STATE 12 AND HAS BEEN REMOVED.

CANONICAL FLOW TABLE FOR
EXAMPLE C.3 - 4-INPUT, 30 ROW

X1 X2 X3 X4

0000	1000	1100	0100	0110	1110	1010	0010	0011	1011	1111	0111	0101	1101	1001	0001	Z1	Z2	Z3	Z4	Z5	Z6
(1)	2	0	3	0	0	0	4	0	0	0	0	0	0	0	5	0	0	0	0	0	0
1	(2)	6	0	0	0	7	0	0	0	0	0	0	0	8	0	1	0	0	0	0	0
1	0	6	(3)	9	0	0	0	0	0	0	0	10	0	0	0	0	1	0	0	0	0
1	0	0	0	9	0	7	(4)	11	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	11	0	0	0	10	0	8	(5)	0	0	0	1	1	1
0	2	(6)	3	0	12	0	0	0	0	0	0	0	13	0	0	1	1	0	0	0	0
0	2	0	0	0	12	(7)	4	0	14	0	0	0	0	0	0	1	0	1	0	0	0
0	2	0	0	0	0	0	0	0	14	0	0	0	13	(8)	5	1	0	0	1	0	0
0	0	0	3	(9)	12	0	4	0	0	0	15	0	0	0	0	0	1	1	0	0	0
0	0	0	3	0	0	0	0	0	0	0	15	(10)	13	0	5	0	1	0	1	0	0
0	0	0	0	0	0	0	4	(11)	14	0	15	0	0	0	16	0	0	1	1	0	0
0	0	6	0	9	(12)	7	0	0	0	17	0	0	0	0	0	1	1	1	0	0	0
0	0	6	0	0	0	0	0	0	0	17	0	10	(13)	8	0	1	1	0	1	0	0
0	0	0	0	0	0	7	0	11	(14)	17	0	0	0	8	0	1	0	1	1	0	0
0	0	0	0	9	0	0	0	11	0	17	(15)	10	0	0	0	0	1	1	1	0	0
18	0	0	0	0	0	0	0	11	0	0	0	10	0	8	(16)	0	0	0	1	1	0
0	0	0	0	0	12	0	0	0	14	(17)	15	0	13	0	0	1	1	1	1	0	0
(18)	19	0	3	0	0	0	4	0	0	0	0	0	0	0	20	0	0	0	0	0	0
21	(19)	6	0	0	0	7	0	0	0	0	0	0	0	8	0	1	0	0	0	0	0
18	0	0	0	0	0	0	0	22	0	0	0	10	0	8	(20)	0	0	0	1	0	0
(21)	23	0	3	0	0	0	4	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	24	(22)	14	0	15	0	0	0	16	0	0	1	1	0	0
25	(23)	6	0	0	0	7	0	0	0	0	0	0	0	8	0	1	0	0	0	0	0
1	0	0	0	9	0	26	(24)	11	0	0	0	0	0	0	0	1	0	1	0	0	0
(25)	27	0	3	0	0	0	4	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0	28	0	0	0	12	(26)	4	0	14	0	0	0	0	0	0	1	0	1	0	0	0
18	(27)	6	0	0	0	7	0	0	0	0	0	0	0	8	0	1	0	0	0	0	0
1	(28)	6	0	0	0	7	0	0	0	0	0	0	0	8	0	1	0	0	0	1	0

(PASSIVE MEMORY ASSIGNMENT)

STATE SIGNALS:

(1) = 0000 * Y 1 * Y 2
(18) = 0000 * Y 1 * $\underline{Y}$ 2
(21) = 0000 * $\underline{Y}$ 1 * Y 3
(25) = 0000 * $\underline{Y}$ 1 * $\underline{Y}$ 3

(2) = 1000 * Y 4 * Y 5 * Y 7
(19) = 1000 * Y 4 * Y 5 * $\underline{Y}$ 7
(23) = 1000 * Y 4 * $\underline{Y}$ 5
(27) = 1000 * $\underline{Y}$ 4 * Y 6
(28) = 1000 * $\underline{Y}$ 4 * $\underline{Y}$ 6

(6) = 1100

(3) = 0100

(9) = 0110

(12) = 1110

(7) = 1010 * Y 8
(26) = 1010 * $\underline{Y}$ 8

(4) = 0010 * Y 9
(24) = 0010 * $\underline{Y}$ 9

(11) = 0011 * Y10
(22) = 0011 * $\underline{Y}$10

(14) = 1011

(17) = 1111

(15) = 0111

(10) = 0101

(13) = 1101

(8) = 1001

(5) = 0001 * Y11 * Y12
(16) = 0001 * Y11 * $\underline{Y}$12
(20) = 0001 * $\underline{Y}$11

SWITCHING CONDITIONS:

Y 1 SET = 2 + 3 + 4 + 5 + 24 + 28 + 16 + 20 + 27
RESET = 19 + 23

Y 2 SET = 2 + 3 + 4 + 5 + 24 + 28
RESET = 16 + 20 + 27

Y 3 SET = 19
RESET = 23

Y 4 SET = 1 + 6 + 7 + 8 + 18 + 21
RESET = 25 + 26

Y 5 SET = 1 + 6 + 7 + 8 + 18
RESET = 21

Y 6 SET = 25
RESET = 26

Y 7 SET = 1 + 6 + 7 + 8
RESET = 18

Y 8 SET = 2 + 4 + 12 + 14 + 19 + 23 + 27 + 28
RESET = 24

Y 9 SET = 1 + 7 + 9 + 11 + 18 + 21 + 25 + 26
RESET = 22

Y10 SET = 4 + 5 + 14 + 15 + 16 + 24
RESET = 20

Y11 SET = 1 + 8 + 10 + 21 + 25 + 11 + 22
RESET = 18

Y12 SET = 1 + 8 + 10 + 21 + 25
RESET = 11 + 22

OUTPUT SIGNALS:

Z1 = 2 + 6 + 7 + 8 + 12 + 13 + 14 + 17 + 19 + 23 + 24 + 26 + 27 + 28
Z2 = 3 + 6 + 9 + 10 + 12 + 13 + 15 + 17
Z3 = 4 + 7 + 9 + 11 + 12 + 14 + 15 + 17 + 22 + 24 + 26
Z4 = 5 + 8 + 10 + 11 + 13 + 14 + 15 + 16 + 17 + 20 + 22
Z5 = 5 + 16 + 28
Z6 = 5

```
$JOB
      CALL LOGSYN
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.4 - 8 EVENT COUNTER
 1 116
0   1
1001   2                                                        0
   31002                                                        0
1003   4                                                        0
   51004                                                        0
1005   6                                                        0
   71006                                                        0
1007   8                                                        0
   91008                                                        0
1009  10                                                        0
  111010                                                        0
1011  12                                                        0
  131012                                                        0
1013  14                                                        0
  151014                                                        0
1015  16                                                        0
   11016                                                        1
```

LOGIC SYNTHESIS
FOR 1 INPUTS, 1 OUTPUTS.

ORIGINAL PRIMITIVE FLOW TABLE FOR
EXAMPLE C.4 - 8 EVENT COUNTER

X1 0	1	Z1
(1)	2	0
3	(2)	0
(3)	4	0
5	(4)	0
(5)	6	0
7	(6)	0
(7)	8	0
9	(8)	0
(9)	10	0
11	(10)	0
(11)	12	0
13	(12)	0
(13)	14	0
15	(14)	0
(15)	16	0
1	(16)	1

CANONICAL FLOW TABLE FOR
EXAMPLE C.4 - 8 EVENT COUNTER

X1 0	1	Z1
(1)	2	0
3	(2)	0
(3)	4	0
5	(4)	0
(5)	6	0
7	(6)	0
(7)	8	0
9	(8)	0
(9)	10	0
11	(10)	0
(11)	12	0
13	(12)	0
(13)	14	0
15	(14)	0
(15)	16	0
1	(16)	1

(PASSIVE MEMORY ASSIGNMENT)

STATE SIGNALS:

(1) = 0 * Y 1 * Y 2 * Y 4
(3) = 0 * Y 1 * Y 2 * $\underline{Y}$ 4
(5) = 0 * Y 1 * $\underline{Y}$ 2 * Y 6
(7) = 0 * Y 1 * $\underline{Y}$ 2 * $\underline{Y}$ 6
(9) = 0 * $\underline{Y}$ 1 * Y 3 * Y 5
(11) = 0 * $\underline{Y}$ 1 * Y 3 * $\underline{Y}$ 5
(13) = 0 * $\underline{Y}$ 1 * $\underline{Y}$ 3 * Y 7
(15) = 0 * $\underline{Y}$ 1 * $\underline{Y}$ 3 * $\underline{Y}$ 7

(2) = 1 * Y 8 * Y 9 * Y11
(4) = 1 * Y 8 * Y 9 * $\underline{Y}$11
(6) = 1 * Y 8 * $\underline{Y}$ 9 * Y13
(8) = 1 * Y 8 * $\underline{Y}$ 9 * $\underline{Y}$13
(10) = 1 * $\underline{Y}$ 8 * Y10 * Y12
(12) = 1 * $\underline{Y}$ 8 * Y10 * $\underline{Y}$12
(14) = 1 * $\underline{Y}$ 8 * $\underline{Y}$10 * Y14
(16) = 1 * $\underline{Y}$ 8 * $\underline{Y}$10 * $\underline{Y}$14

SWITCHING CONDITIONS:

Y 1 SET = 16 + 2 + 4 + 6
RESET = 8 + 10 + 12 + 14

Y 2 SET = 16 + 2
RESET = 4 + 6

Y 3 SET = 8 + 10

```
          RESET = 12 + 14

     Y 4  SET   = 16
          RESET =  2

     Y 5  SET   =  8
          RESET = 10

     Y 6  SET   =  4
          RESET =  6

     Y 7  SET   = 12
          RESET = 14

     Y 8  SET   =  1 +  3 +  5 +  7
          RESET =  9 + 11 + 13 + 15

     Y 9  SET   =  1 +  3
          RESET =  5 +  7

     Y10  SET   =  9 + 11
          RESET = 13 + 15

     Y11  SET   =  1
          RESET =  3

     Y12  SET   =  9
          RESET = 11

     Y13  SET   =  5
          RESET =  7

     Y14  SET   = 13
          RESET = 15

OUTPUT SIGNALS:

     Z1 = 16
```

```
$JOB
      CALL LOGSIM
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.5 - TABLE XII - CLASSICAL EQUATIONS
 2 2 2
0000
 2 1 2
 2 1 2
 2 1 2
 2 1 2

      MS(1) = ITS(3) + X(2)*Y(2)
      MR(1) = ITS(1) + NOT(X(2))*Y(2)
      MS(2) = ITS(2)*Y(1) + ITS(4)*Y(1)
      MR(2) = ITS(3)*Y(1) + ITS(1)*NOT(Y(1))
      Z(1) = ITS(2)*NOT(Y(1))*Y(2)
      Z(2) = ITS(4)*Y(1)*Y(2)
```

LOGIC SIMULATION
FOR 2 INPUTS, 2 OUTPUTS, 2 MEMORIES.

SIMULATED FLOW TABLE FOR
EXAMPLE C.5 - TABLE XII - CLASSICAL EQUATIONS

X1 X2 00	10	11	01	Z1	Z2	
(1)	2	0	3	0	0	0 0
1	(2)	4	0	0	0	0 0
1	0	4	(3)	0	0	0 0
0	6	(4)	5	0	0	1 0
1	0	4	(5)	0	1	1 1
1	(6)	4	0	1	0	0 1

```
$JOB
      CALL LOGSIM
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.6 - ROTATION DIRECTION SENSOR - STATE MATRIX EQUATIONS
 2 1 4
000000
 1 1
 1 2
 1 3
 1 4

      MS(1) = ITS(4)*NOT(Y(4))
      MR(1) = ITS(2) + ITS(4)*Y(4)
      MS(2) = ITS(3)
      MR(2) = ITS(1)
      MS(3) = ITS(4)
      MR(3) = ITS(2)
      MS(4) = ITS(3)
      MR(4) = ITS(1)
      Z(1) = ITS(1)*Y(1) + ITS(2)*NOT(Y(2)) + ITS(3)*NOT(Y(3)) +
     *        ITS(4)*Y(4)
```

LOGIC SIMULATION
FOR 2 INPUTS, 1 OUTPUTS, 4 MEMORIES.

SIMULATED FLOW TABLE FOR
EXAMPLE C.6 - ROTATION DIRECTION SENSOR - STATE MATRIX EQUATIONS

X1 X2 00	10	11	01	Z1	
(1)	2	0	3	0	0 0 0 0
1	(2)	4	0	1	0 0 0 0
6	0	5	(3)	0	1 0 1 0
0	8	(4)	7	1	0 1 0 1
0	8	(5)	7	0	1 1 1 1
(6)	2	0	3	1	1 0 1 0
1	0	5	(7)	1	0 1 1 1
1	(8)	4	0	0	0 1 0 1

```
$JOB
      CALL LOGSIM
      STOP
      END
```

```
DATA CARDS:
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

EXAMPLE C.2 - TABLE 6.4 FROM FLUID LOGIC BY FITCH - STATE MATRIX EQUATIONS
 2 1 5
1111100
 1 1
 2 2 3
 1 4
 1 5

      MS(1) = ITS(2)*Y(2)*Y(3) + ITS(4)
      MR(1) = ITS(2)*Y(2)*NOT(Y(3))
      MS(2) = ITS(1) + ITS(3)*Y(4)
      MR(2) = ITS(3)*NOT(Y(4))
      MS(3) = ITS(1)*Y(1)
      MR(3) = ITS(1)*NOT(Y(1)) + ITS(3)*Y(4)
      MS(4) = ITS(2)*Y(2)*Y(3) + ITS(4)*Y(5)
      MR(4) = ITS(2)*Y(2)*NOT(Y(3)) + ITS(2)*NOT(Y(2)) +ITS(4)*NOT(Y(5))
      MS(5) = ITS(1)*Y(1) + ITS(3)*Y(4)
      MR(5) = ITS(3)*NOT(Y(4))
      Z(1) = ITS(2)*NOT(Y(2)) + ITS(3)*NOT(Y(4)) + ITS(4)*NOT(Y(5))
```

LOGIC SIMULATION
FOR 2 INPUTS, 1 OUTPUTS, 5 MEMORIES.

SIMULATED FLOW TABLE FOR
EXAMPLE C.2 - TABLE 6.4 FROM FLUID LOGIC BY FITCH - STATE MATRIX EQUATIONS

X1 X2 00	10	11	01	Z1						
(1)	2	0	3	0		1	1	1	1	1
1	(2)	4	0	0		1	1	1	1	1
1	0	4	(3)	0		1	1	1	1	1
0	5	(4)	3	0		1	1	0	1	1
6	(5)	7	0	0		0	1	0	0	1
(6)	5	0	3	0		0	1	0	0	1
0	9	(7)	8	1		0	0	0	0	0
1	0	7	(8)	1		1	0	0	0	0
1	(9)	7	0	1		1	0	0	0	0

VITA

Robert Loren Woods

Candidate for the Degree of

Master of Science

Thesis: THE STATE MATRIX METHOD FOR THE SYNTHESIS OF DIGITAL LOGIC SYSTEMS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Frederick, Oklahoma, April 14, 1945, the son of Mr. and Mrs. Aldon Woods.

Education: Graduated from Frederick High School, Frederick, Oklahoma, in May, 1963; attended Oklahoma State University from 1963 to 1965; received the Bachelor of Science degree from Southern Methodist University in 1967, with a major in Mechanical Engineering; completed requirements for the Master of Science degree from Oklahoma State University in May, 1970.

Professional Experience: Technical analyst, Nuclear Research Services, Dallas, Texas, 1967; graduate teaching assistant, Southern Methodist University, 1968; graduate research assistant, Oklahoma State University, 1968-69.

Professional Organizations: Member Pi Tau Sigma.