To the Graduate Council:

I am submitting herewith a thesis written by Wen Pai Lu entitled "An Investigation of the Hazards in Fundamental Mode Sequential Circuit Implementation Using SSI, MSI and LSI Devices." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Engineering, with a major in Electrical Engineering.

Robert E. Bodenheimer

Robert E. Bodenheimer, Major Professor

We have read this thesis
and recommend its acceptance:

Donald W. Bouldin

Accepted for the Council:

L. Evans Peck

Vice Chancellor
Graduate Studies and Research

AN INVESTIGATION OF THE HAZARDS IN FUNDAMENTAL MODE SEQUENTIAL CIRCUIT IMPLEMENTATION USING SSI, MSI AND LSI DEVICES

A Thesis

Presented for the

Master of Engineering

Degree

The University of Tennessee, Knoxville

Wen Pai Lu

August 1981

ACKNOWLEDGMENTS

I wish to extend my deepest thanks to my major professor, Dr. Robert E. Bodenheimer, for his continual advice, instruction and most of all, his personal support during the course of this research. Also, special thanks to Dr. Donald W. Bouldin and Dr. John M. Bailey for reading the manuscript and offering valuable advice and suggestion.

Appreciation is extended to the Department of Electrical Engineering, for moral support throughout this research.

Furthermore, Mr. Charles Hagan is given appreciation for editing the manuscript.

Finally, I wish to express sincere appreciation to my parents, brothers and sister, and friends for their constant encouragement throughout this research.

ABSTRACT

An extensive investigation of the essential hazard in a fundamental mode circuit was conducted using random logic. This hazard does not always exist in the circuit when it is analyzed using Unger's method. A theoretical method of detecting essential hazard was derived, and an experiment was conducted to measure the minimum delay time that would cause hazards.

The combinational part of the fundamental mode circuit was replaced by three different types of MSI and LSI devices. They are digital multiplexer, read only memory and programmable logic array. These circuits were connected to an input sequence generator which was either a microcomputer or a high frequency circuit at 909 KHz. Use of a logic analyzer allowed investigation of the combinational hazards in the MSI circuit. Sequential hazards were also considered and tested. Some possible methods of eliminating hazards in the circuit were established.

In addition, some hazards that might occur when power was turned on were also studied. Two methods of solving the power reset problem were presented, and they were applied in the experiment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Sequential functions are realized by systems of the form shown in Figure 1. These are essentially combinational circuits with some of the output signals being fed back into some of the input terminals as internal inputs. This model shows n-tuples inputs $(X_1,\ldots,X_n)$, m-tuples outputs $(Z_1,\ldots,Z_m)$ and r-tuples of present states $(y_1,\ldots,y_r)$ and next states $(Y_1,\ldots,Y_r)$. The mathematical expressions of these variables are expressed as[20]

$$Z_i = f_i(X_1,\ldots,X_n;y_1,\ldots,y_r), \quad i = 1,\ldots,m \qquad \text{(I-1)}$$

and $$Y_i = g_i(X_1,\ldots,X_n;y_1,\ldots,y_r), \quad i = 1,\ldots,r \qquad \text{(I-2)}$$

Outputs of the sequential circuit in Figure 1 are functions of the present inputs, X_i, and the present states, y_i.

Fundamental Concept

Sequential circuits are categorized as synchronous and asynchronous circuits[10] The inputs of the former circuit are allowed to change only in the interval between the clock pulses. Hence, the outputs and states of the circuit are of interest only at a fixed instant of time. The aforementioned restrictions simplify the analysis and synthesis of synchronous sequential circuits. However, some circuits do not operate with a clock. Another type of sequential circuit is required to operate without a synchronizing

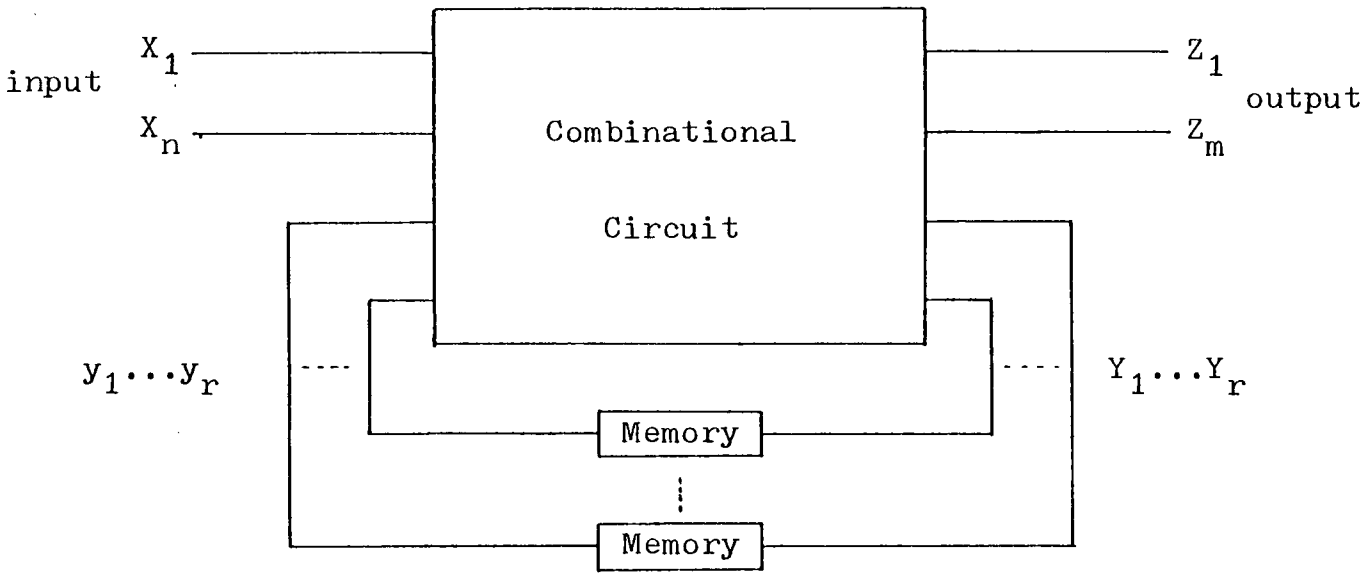


Figure 1. Basic Block Diagram of a Sequential Machine.

clock. Such circuits are used as interfaces to control interactions between two synchronous circuits operating at different speeds, such as a peripheral I/O unit and a digital computer, and are classified as asynchronous sequential circuits. Asynchronous circuit operation is divided into pulse mode and level mode (fundamental mode). Pulse mode circuits have pulse inputs and unclocked memory elements, while level mode circuits have level inputs and feedback loops.

The general model for a level mode circuit is repeated in Figure 2, but the memory elements shown in Figure 1 are replaced by delay elements in the model. In practice, the use of a physical delay line is often unnecessary since sufficient delay is present in the other circuit elements. Once an input change initiates a transition, a level mode circuit is required to reach a stable condition before the next input change is allowed to occur. The inputs of a level mode circuit are allowed to have multiple changes at a given instant of time. However, the inputs of a fundamental mode circuit are restricted in that only one variable is allowed to change at a given time. These circuits consconstitute a "clockless" or a "free-running circuit without special timing signals[6].

The flow table shown in Table I describes the operational characteristics of asynchronous sequential circuits. Columns (Is) of the flow table represent the external input

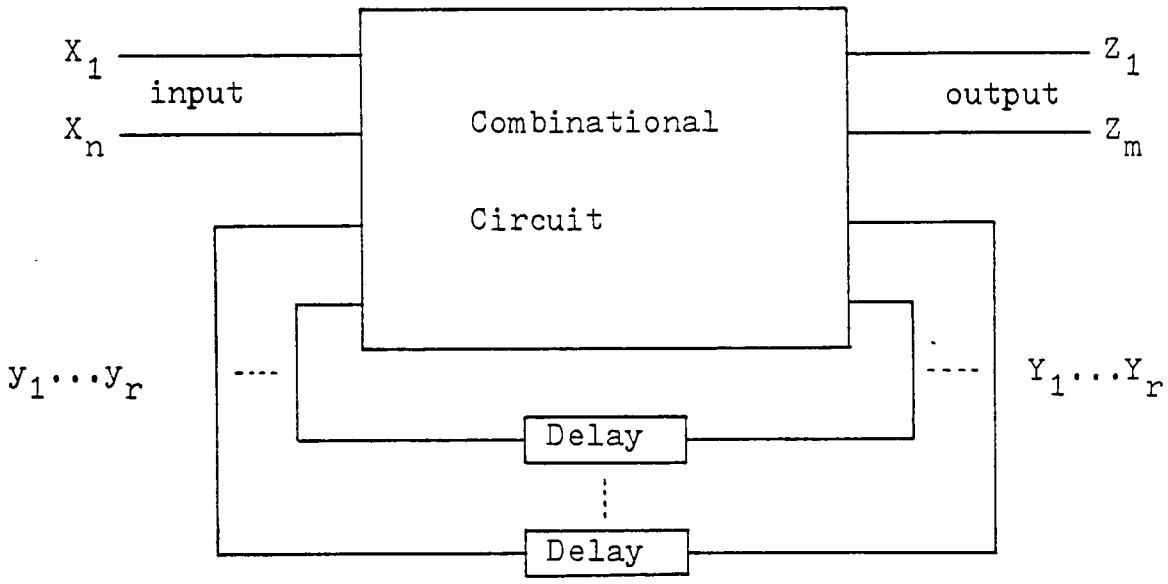


Figure 2. Basic Model of Level Mode Sequential Circuit.

TABLE I

Flow Table of a Level Mode Sequential Circuit.

Present State	Input Column I_1	I_2	I_3	I_4
1	(1)/0	2 /0	(1)/1	4 /0
2	1 /0	(2)/1	3 /0	4 /0
3	(3)/0	4 /0	(3)/0	4 /0
4	3 /0	(4)/1	1 /0	(4)/0

states of the system. Present internal states labelled 1 through 4 are placed on each separated row. This system always starts at a particular internal state under a certain input state. For example, assuming the machine starts at (1) under input column I_1. As the input state changes to I_2, the entry in the table describes a transition to state 2 . Since this internal state is not the same as the present internal state on the same row, the state is unstable. A transition takes place to the state labelled (2) . This state is the same as the present internal state on the same row, and no further transition occurs as long as the input state remains at I_2. In Table I a circled internal state represents a stable state.

A step-by-step description for designing a typical fundamental mode sequential circuit is given in Figure 3. Starting with a primitive state table as the first step, techniques are used to reduce the state table to that shown in Figure 3(b). In step 3, a state assignment is made to each row of Figure 3(b). An excitation table is formed by substituting internal states in Figure 3(b) with the corresponding coded state assignment in Figure 3(c) and is given in Figure 3(d). Next, two K-Map tables for the internal states Y_1 and Y_2 are constructed in step 5, and their corresponding logic equations are produced in Figure 3(e). Finally, a logic diagram is prepared from the expressions in Figure 3(e) and illustrated in Figure 3(f).

step 1

	00	01	11	10
a	(a)/0	b/-	-/-	f/-
b	a/-	(b)/0	c/-	-/-
c	-/-	b/-	(c)/0	d/-
d	a/-	-/-	e/-	(d)/1
e	-/-	b/-	(e)/1	d/-
f	a/-	-/-	e/-	(f)/0

(a)

step 2

	00	01	11	10
1	(1)/0	2/0	3/-	(1)/0
2	1/0	(2)/0	(2)/0	3/-
3	1/-	2/-	(3)/1	(3)/1

(b)

step 3

y_1	y_2	State
0	0	1
1	1	2
0	1	3

(c)

step 4

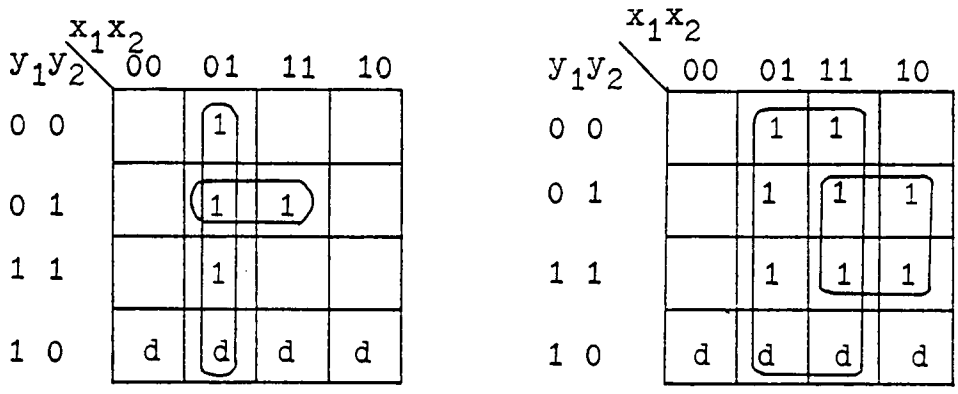
y_1y_2 \ x_1x_2	00	01	11	10
0 0	(00)/0	11/0	01/-	(00)/0
1 1	00/0	(11)/0	(11)/0	01/-
0 1	00/-	11/-	(01)/1	(01)/1

Y_1Y_2/Z

(d)

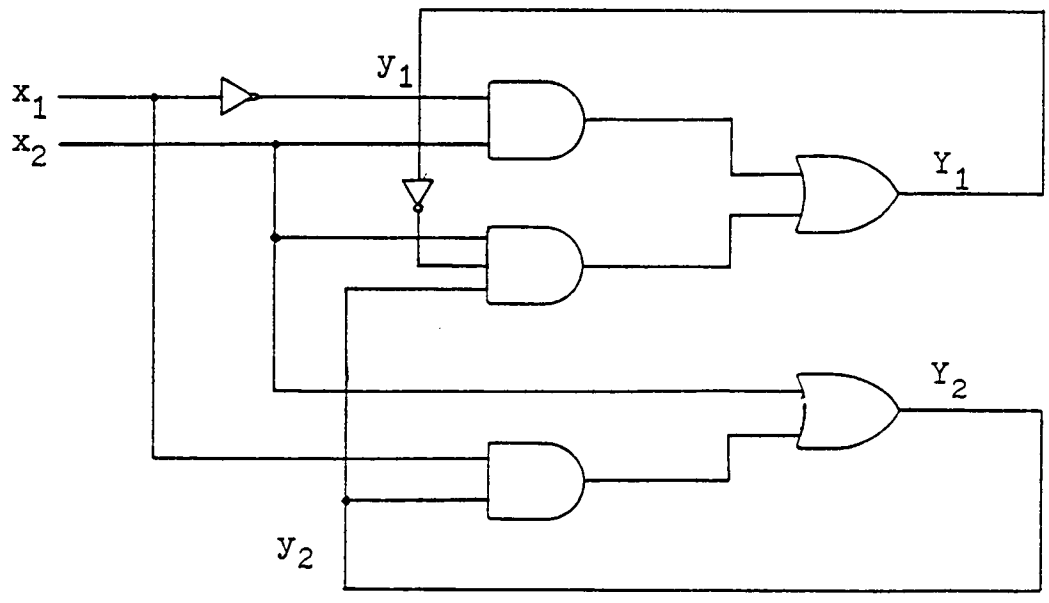
Figure 3. Procedure for Designing an Fundamental Mode Circuit.
(a) Primitive Flow Table,
(b) Reduced Flow Table,
(c) State Assignment,
(d) Excitation Table,

step 5



(e)

step 6



(f)

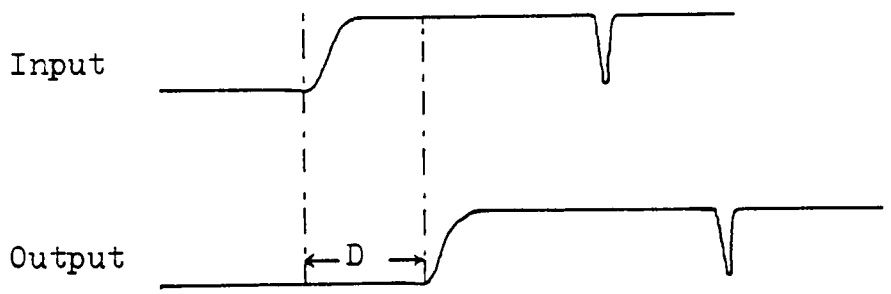
Figure 3. (Continue)
(e) K-Map and Logic Expression,
(f) Logic Diagram.

Hazard Consideration

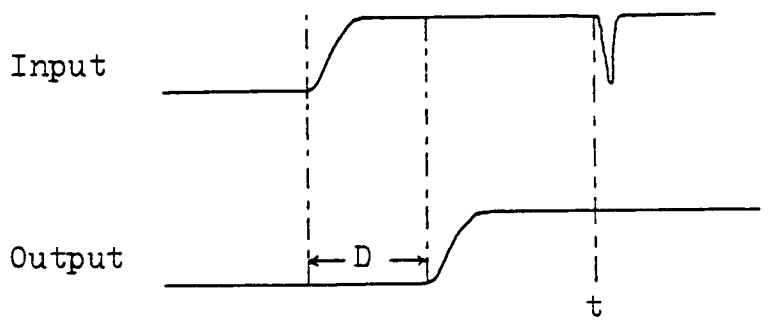
All physical devices, such as the logic gates in switching circuits, do not operate instaneously. This causes a delay of the signal passing through the gates. If the output signal is the same as the input signal but shifted by a delay time D, the delay is called a pure delay. A typical timing of a pure delay is shown in Figure 4(a). In Figure 4(b); the output switches to one for a period of time D after the input changed to one. When the input signal reaches time t in Figure 4(b), this input momentarily goes to zero and then back to one. Since this transition does not persist for a time more than D, the output response remains unchanged. This input change is filtered by a device that is analogous to a low-pass filter. This type of delay is referred to as inherent delay[33].

Many malfunctions of the level mode sequential switching circuit are caused by delays. A signal may temporarily or transiently take different logic values from those predicted by the logic equations. As a result, the delay may cause a switching circuit malfunction. This incorrect operation is termed a hazard.

Classically, there are two main types of hazard in switching circuits. A transient hazard is present if a hazard momentarily produces a false output signal following a certain input change. If it is possible for a circuit to enter the wrong internal state after certain input changes,



(a)



(b)

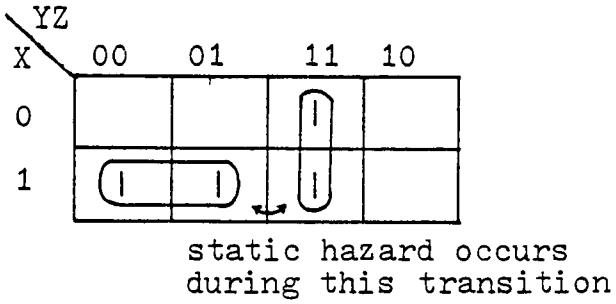
Figure 4. Two Types of Delay.
(a) Pure Delay,
(b) Inherent Delay.

then a steady state hazard exists.

Transient hazards exist in combinational circuits and they are divided into static and dynamic hazards. In general, a static hazard is related to the condition that occurs when a single input variable change produces a momentary output change when in fact no output change should occur. The system given in Figure 5 is a good example of a circuit containing a static hazard. Figure 5(a) includes both the K-Map of this system and the logic equation of the circuit. A logic diagram prepared from the equation is shown in Figure 5(b). At time t_5 in Figure 5(c), the input state (X,Y,Z) is (101). A change in Y at t_6 causes line a change to zero at t_7. When the delay time for gate 2 in Figure 5(b) is much longer than the response time of gate 1, line b does not change to one instantly. Hence, the output f changes from one to zero at t_8 and from zero to one at t_{10} in figure 5(c). This change in f is not indicated by the logic description of the network and hence is not the correct behavior of the network. This type of hazard is called a static hazard.

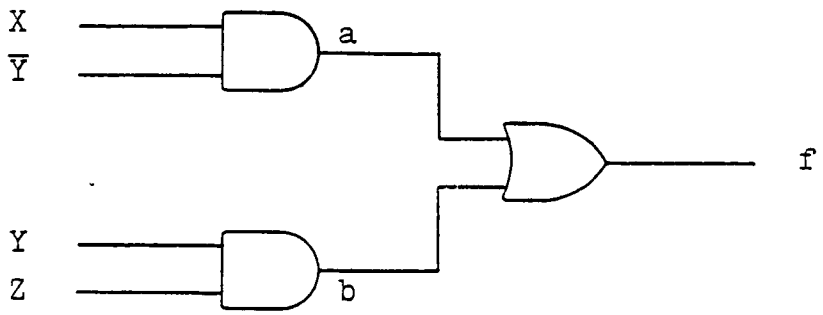
A definition of static hazard is given as follows[8]:

Definition : Let there be a state transition in a fundamental mode circuit such that one or more of the secondary variables are required to remain constant. Also, the circuit is assumed to hace unequal delay in each element. If it is possible for one or more of the constant state variables to change momentarily, then a



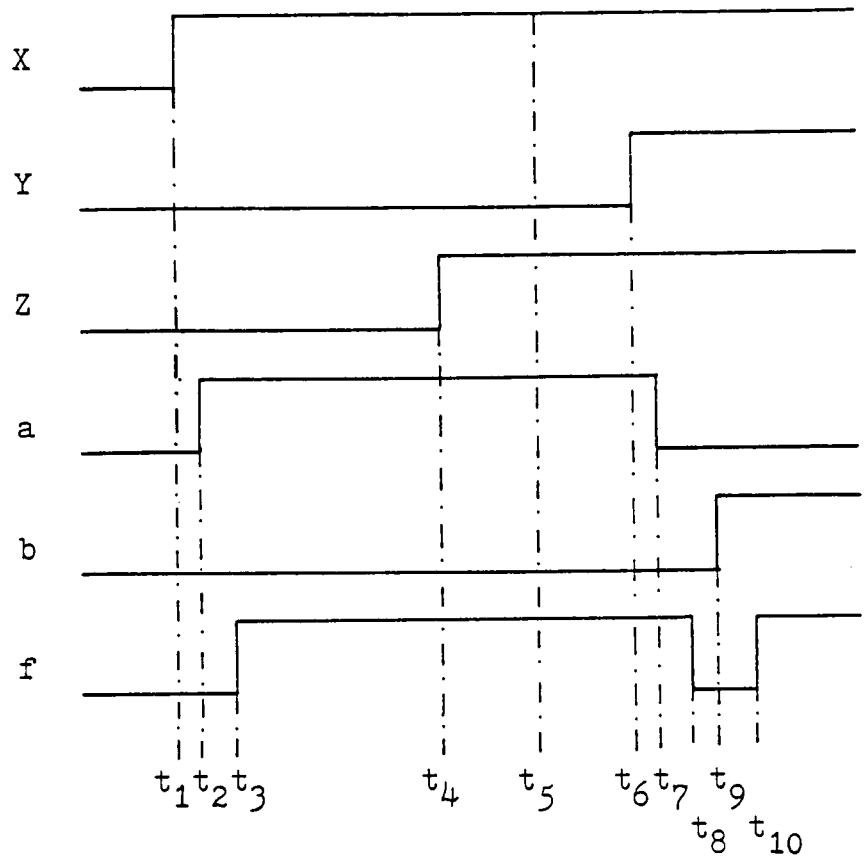
$$f = X\overline{Y} + YZ$$

(a)



(b)

Figure 5. Illustration of a static hazard.
(a) K-Map and logical equation,
(b) Logic diagram,



(c)

Figure 5. (continue) (c) Timing Diagram.

state hazard is said to exist.

In general, a static hazard is a characteristic of the combinational circuit. Figure 6 dipicts the elimination of a static hazard. This hazard can be removed by grouping all pairs of minterms with a common product.

If a static hazard occurs when the output should remain at logical 1, it is called a static 1-hazard. Obviously, the same situation can occur when the output remains at 0. In this case, it is characterized as a static 0-hazard.

Another type of hazard is the dynamic hazard. This hazard occurs when the circuit momentarily goes to the new value and back to the present value before making the permanent transition; that is, the output changes $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ instead of $0 \rightarrow 1$. The dynamic hazard, like the static hazard, causes spikes in transient operation of the combinational circuit and can also cause a malfunction in the fundamental mode circuit.

Steady hazards are called essential hazards[34] It is briefly considered as follows: In the physical circuit, the inherent delay associated with every logic element may not always be equal. This unequal delay in each path may result in one excitation change before the input change propagates to other sections of the combinational circuit. Such a hazard is diagramed in Figure 7. Figure 7(a) is the excitation table for this circuit.

Assume that the circuit is in the stable state (1) in

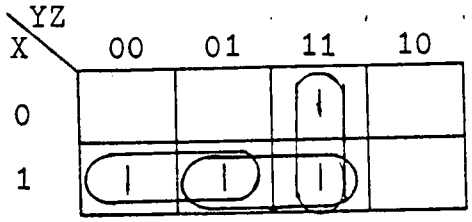


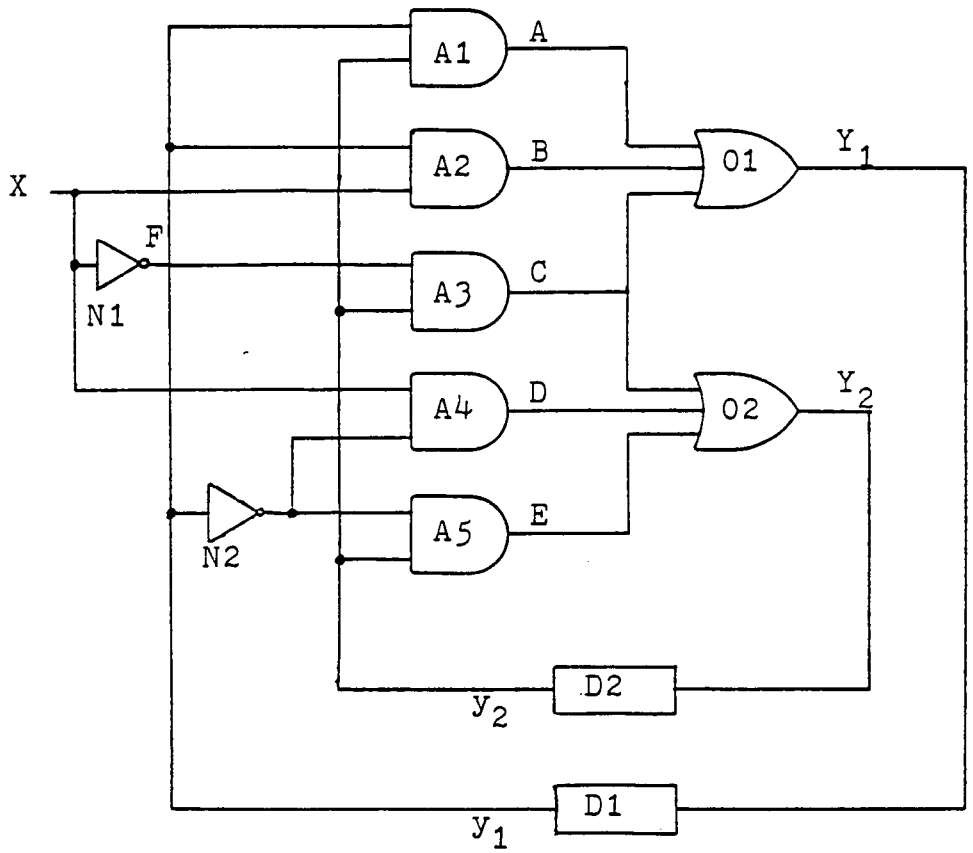
Figure 6. Elimination of Static Hazard.

y_1y_2		X = 0	X = 1	X = 0	X = 1
0 0	1	(1)	2	(00)	01
0 1	2	3	(2)	11	(01)
1 1	3	(3)	4	(11)	10
1 0	4	1	(4)	00	(10)

$$Y_1 = y_2\bar{X} + y_1X + y_1y_2$$

$$Y_2 = y_2\bar{X} + \bar{y}_1X + \bar{y}_1y_2$$

(a)



(b)

Figure 7. Circuit with Essential Hazard.
(a) Truth Table, Excitation Table and Logic Equations,
(b) Circuit Diagram.

Figure 7. A change in X from zero to one causes the output of A4 to one in Figure 7(b). This causes the output state Y_2 to become one. Since the NOT gate is assumed to have larger delay associated with it in comparison to the delay of other elements, the output of N1 remains at one as the feedback signal Y_2 reaches the other input of A3. Then, the output of A3 becomes one and causes a transition at Y_1 from zero to one. This system is now in state shown at ③ (11). When the signal X finally reaches N1, line c in Figure 7(b) switches to zero, changing Y_2 to zero and causing the circuit to stabilize at 10. It is not the expected state for the input transition when the circuit is initially at state state ① . A steady state hazard exists in the transition from ① to ② of Figure 7(a).

Another hazard due to multiple-input change can occur in sequential circuits[7]. This type of hazard is called a function hazard[4].

Critical races are another form of hazard which might appear in a sequential circuit[2,10,32]. This is the condition where two or more secondary state variables change value as a result of an input change. This hazard can be eliminated by making an appropriate state assignment[5,16, 21,30].

Problem in the Design

Hazard detection and correction are the most important considerations in asynchronous sequential circuit design.

Huffman[11]discussed the static and dynamic hazard in switching networks. Eichelberger illustrated a unified approach to the detection of hazards in both combinational and sequential circuit through the use of ternary algebra[4]. Also , Lerner[15], Misel and Kashef[17] developed many ideas for the detection and correction of hazards. In addition, modern test equipment such as a logic analyzer may be used to trace different hazards in logic circuits.

The first part of this thesis develops a new method for the investigation of the essential hazards in a level mode asynchronous circuit. This method is only applied to random logic networks. It is an extension of Unger's work on this class of hazard[34].

In modern logic design, the combinational part of a fundamental mode circuit can be constructed by using standard MSI and LSI devices. These devices are digital multiplexer/selector (MUX), read only memory (ROM) and programmable logic array (PLA).

Detection and elimination of hazards in the random logic design has been fully discussed in the literature [2, 6,11,32]. Although hazards contained in the MSI and LSI devices are very similar to those in the random logic circuit, they form a class of hazards which have not yet been analyzed. The purpose of the second part of this thesis is to study hazards in the combinational part of a fundamental mode circuit which is constructed using MUX, ROM and

PLA realizations. Some methods have also been developed to eliminate the hazards associated with these realizations.

Unless otherwise stated, this thesis considers only asynchronous fundamental mode operation; that is, only one input is allowed to change at a time. Also, hazards caused by critical races are not considered since they are assumed to have been eliminated by making a correct state assignment.

CHAPTER II

ESSENTIAL HAZARD IN RANDOM LOGIC CIRCUIT

An essential hazard in a random logic circuits was first described by Unger[34]. He presented an excellent work on the theory of designing an asynchronous sequential circuit both with and without an essential hazard. His conclusions are

1. The model sequential function contains a stable state S_o and an input variable X. Starting with the system in S_o, three consecutive changes in X bring the system to a state other than the one arrived at after the first change in X. Consequently, an essential hazard is said to exist in this sequential function.
2. If a function contains an essential hazard, then any proper circuit realization of the function must contain at least on delay element.

The following discussion assumes that the sequential circuit is free of critical race conditions as well as all static and dynamic hazards.

Discussion on the Relation to Previous Works

A block diagram describing the characteristic of delay time in a fundamental mode circuit is shown in Figure 8. Suppose a delay time δ is defined as the maximum time for a signal to propagate from the input terminal X to any output

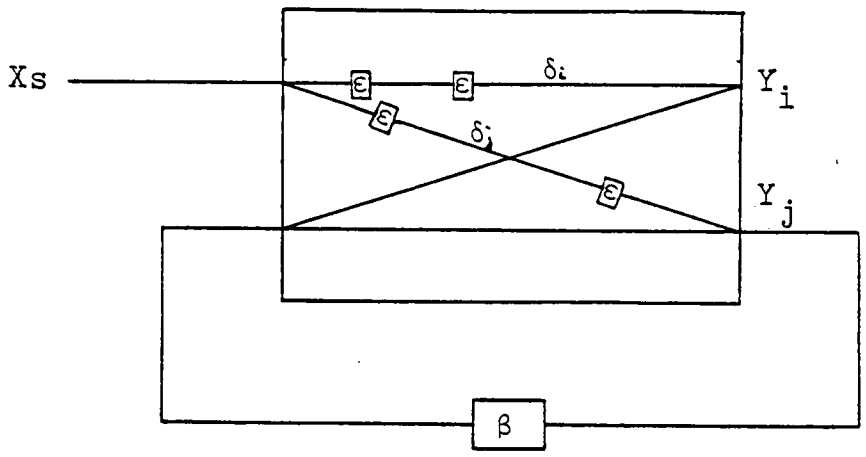


Figure 8. Characteristic of Delay Time in a Fundamental Mode Circuit.

terminal Y. As shown in Figure 8, β is defined as the magnitude of the maximum delay in the feedback path. A stray delay of value ε is assumed to be placed in each gate element of the combinational network. The successive input changes should have a minimum interval time of α. For any $\varepsilon > 0$, there exists a δ and a β. Suppose all stray delay values of a gate element in combinational network in Figure 7 are less than ε. In addition, if all delay elements βs are greater than δ (i.e., $\beta > \delta$) then the circuit would operate in accordance with a corresponding flow table. If the magnitude of β is less than the delay time δ, an essential hazard can possible occur.

Consider the circuit in Figure 9. Assuming that the delay in the feedback loops of Y_i to y_i is large compared to the delays in other branches, then the circuit is functioning correctly. This circuit was used by Unger[33] in the discussion of essential hazard. A corresponding flow table is shown in Table II. Unger gave a definition of the essential hazard and traced out the existence of three possible essential hazards in the circuit: First, as shown in Table II(a), one starts in (1) under input column (00) with X_2 changing as the relevant input variable. Second, it initializes in (2) under column (01) with X_2 changes. Finally, it begins in (4) under column (11) with X_1 as the input variable. Unger described the first essential hazard in detail[33]. He showed that such behavior is always

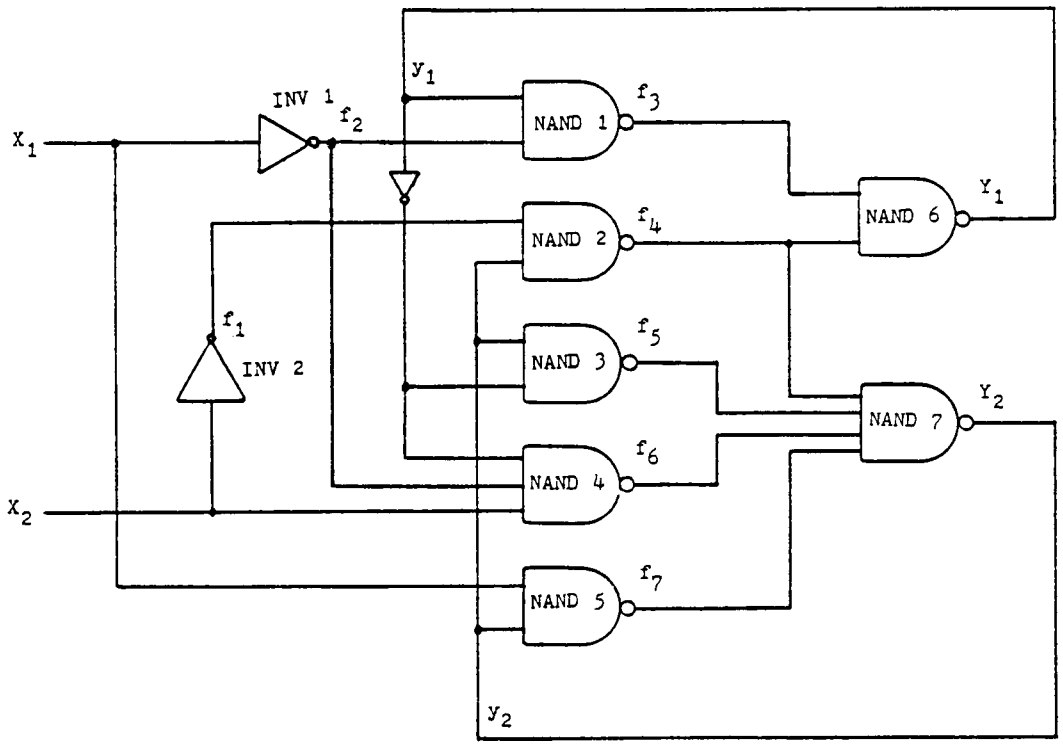


Figure 9. Circuit Diagram of a Fundamental Mode Circuit.

Table II. Table of a Fundamental Mode Circuit.

Present State [a]			Input Column 00	01	11	10
0	0	1	(1)	2	(1)	(1)
0	1	2	3	(2)	(2)	3
1	1	3	(3)	4	2	(3)
1	0	4	(4)	(4)	1	1

Present State [b]		Input Column 00	01	11	10
0	0	(00)	01	(00)	(00)
0	1	11	(01)	(01)	11
1	1	(11)	10	01	(11)
1	0	(10)	(10)	00	10

a Truth Table,
b Excitation Table.

possible in circuits having functions corresponding to the flow matrix shown in Table II.

Ternary algebra could also be applied to trace hazards contained in the circuit. This method was first introduced by Yoeli and Rinon[35] in the application to the study of static hazards in combinational network. The method was reworked and applied by Eichelberger[4] to the analysis of essential hazard in the asynchronous sequential function.

This method is divided into two steps. The first step is to determine all the feedback signals that may change as a result of the input change. The second step is to determine whether or not these feedback signals will eventually stabilize in some predetermined state.

These two steps are applied to the circuit shown in Figure 9. Lines f_1 to f_7 in Figure 9 show the output of each gate. Results are shown in Table III with different switching of inputs X_1 and X_2. The first row of each part of the table illustrates the initial input and starting state of the system. This system ended at the final input and state which given in the last row of Table III(a) to Table III(d). The problem in these cases is to determine whether changing X_1 or X_2 could result in an indeterminate final state. If this condition is met, then an essential hazard occurs in this transition. Table III(a) & (d) shows that a switching of X_2 causes the final state to become an indeterminate state. These results verify that hazards are

Table III. Transition Analysis from Logic Network, Using Ternary Algebraic Method:
(a) Input Change from 00 to 01,
(b) Input Change from 11 to 01,
(c) Input Change from 11 to 10,
(d) Input Change from 01 to 00.

X_1	X_2	y_1	y_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	Y_1	Y_2	$\bar{y}_1$
0	0	0	0	1	1	1	1	1	1	1	0	0	1
0	½	0	0	1	½	1	1	1	½	1	0	½	1
0	½	0	½	1	½	1	½	½	½	1	½	½	1
0	½	½	½	1	½	½	½	½	½	1	½	½	½
0	1	½	½	1	0	½	1	½	½	1	½	½	½
a Input Change from 00 to 01													
1	1	0	0	0	0	1	1	1	1	1	0	0	1
½	1	0	0	½	0	1	1	1	½	1	0	½	1
½	1	0	½	½	0	1	1	½	½	½	0	½	1
0	1	0	½	1	0	1	1	½	0	1	0	1	1
0	1	0	1	1	0	1	1	0	0	0	0	1	1
b Input Change from 11 to 01													
1	1	0	1	0	0	1	1	0	1	0	0	1	1
1	½	0	1	0	½	1	½	0	1	0	½	1	1
1	½	½	1	0	½	1	½	½	1	0	½	1	½
1	0	½	1	0	1	1	0	½	1	0	1	1	½
1	0	1	1	0	1	1	0	1	1	0	1	1	0
c Input Change from 11 to 10													
0	1	0	1	1	0	1	1	0	0	1	0	1	1
0	½	0	1	1	½	1	½	0	½	1	½	1	½
0	½	½	1	1	½	½	½	½	½	1	½	½	½
0	½	½	½	1	½	½	½	½	½	1	½	½	½
0	0	½	½	1	1	½	½	½	1	1	½	½	½
d Input Change from 01 to 00													

contained in these transitions. The other ternary simulation of logic in Table III(b) & (c) illustrate the reverse results because the final state reached a stable state for a change in input X.

Discussion on the Detection of Essential Hazards

The essential hazard described in the previous section is discussed further in the following section. Referring to Figure 9, suppose that a relatively large delay appears in INV2 and the circuit is initially at the state labelled (2) in Table II(b). If X_2 turns off (switches from one to zero), the output of NAND 4 is first changed to unity. However, outputs of NAND 3 and NAND 5 still remain at zero and one respectively. Then the next state Y_2, the output of NAND 7, remains unchanged. Although the large delay of INV 2 prevents the immediate change at the input of NAND 2, the unchanged Y_2 feeds back the signal to the other input of NAND 2 which still holds the output of NAND 2 at 1. The next state is not changed. When the switching signal of X_2 propagates through INV 2 and changes the output of NAND 2 to zero, Y_1, the output of NAND 6 switches to one and is connected to one input of NAND 1. The output of NAND 1 turns off to zero, but this does not affect the next state Y_1. Now, $Y_1 = Y_2 = 1$, and the system is in the stable state (11) marked 3 in Table II(B). It reaches an expected next state and shows that incorrect operation did not occur in this transition.

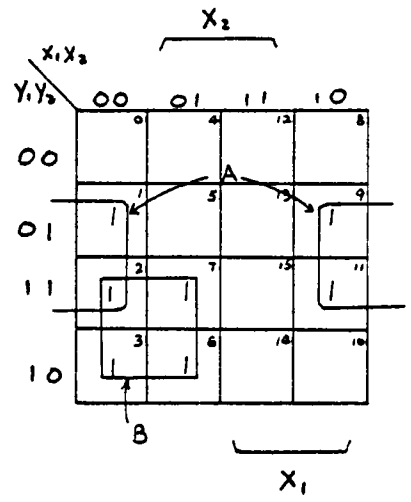
Similarly, the same assumption is made that a large delay appears in INV 1 in Figure 9. While the circuit is started at the state (10), labelled 6 in Table II(b), X_1 is turned from zero to one which is fetched to one input of NAND 5. The switching of X_1 does not change the output of NAND 5, and the next state Y_2 remains unchanged. The large delay that appears in INV 1 would hold the changing signal of X_1 for a period of time. This signal passes through INV 1, switching one input of NAND 1 and one input of NAND 4 to zero. Then, the output of NAND 1 changes to unity and a zero appears at the output of NAND 6. Although a signal changes at one input of NAND 4 due to the switching of X_1, the output of NAND 4 stays at one independently from other inputs. A feedback path of Y_1 connected to one input of NAND 1 changes its output to one. This signal Y_1 propagates the inverter INV 3, but it does not affect the output of NAND 3 and NAND 4. Therefore, the system becomes stable at $Y_1 = Y_2 = 0$. This transition reaches the expected next stable state and a hazard is not detected.

The logic behavior of Figure 9 may also make possible the use of a K-Map to trace essential hazards contained in the circuit. The flow table in Table II could be divided into two K-Maps, and the corresponding realization functions are shown in Figure 10.Few cells of 1s combined into a block of sets which are called 1-set. For example, in Figure 10(a) cells 1, 3, 9 and 11 form a 1-set labelled A and cells 2,

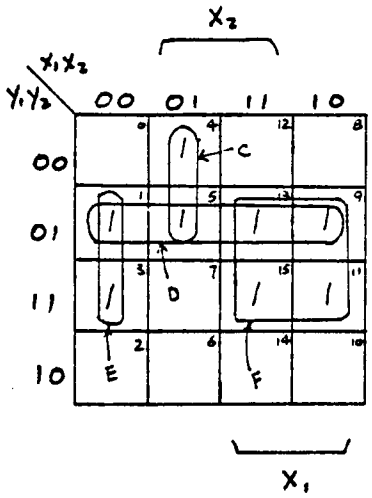
3, 6 and 7 form another 1-set labelled B. Each 1-set represents a term in the realization function (or it may also be defined as a product term). In the following discussion a large delay time is assumed to exist in the path of a signal X propagates to $\overline{X}$.

Suppose that the system initially stays at total state 0000 (represented by $X_1X_2Y_1Y_2$), denoted by cell 0 in the K-Map of Figure 10. If X_2 turns to unity, there is a transition from cell 0 to cell 4. Since a large delay is assumed in the path from X_2 to $\overline{X}_2$, a transition in the first Map in Figure 10(a) does not take place. However, a transition occurs from cell 0 to cell 4 and is shown in Figure 10(b). This transition switches Y_2 to one. If the delay is long enough for the Y_2 signal to return to y_2 before the input X_2 could be changed, a transition may jump to cell 1 instead of cell 5. Now, Y_1 switches to unity, and this effect results in a transition from cell 1 to cell 3 in both Figure 10(a) & 10(b). When the changing signal of $\overline{X}_2$ propagates through, it transits to cell 7, and turns off Y_2. The system now stabilizes at output state 10 which is not the expected output for the changing of input from 00 to 01.

In the second case, the system begins at total state 0101, represented by cell 5 in the K-Map of Figure 10. A change of X_2 from one to zero still makes the output of Y_2 remain at unity (The output Y_1 does not switch first because 1-set of A does contain $\overline{X}_2$. The $\overline{X}_2$ prevents the changing



$$Y_1 = \overline{X}_2 y_2 + \overline{X}_1 y_1$$



$$Y_2 = \overline{X}_2 y_2 + \overline{y}_1 y_2 + X_1 y_2 + \overline{X}_1 X_2 \overline{y}_1$$

Figure 10. Two K-Map of the Fundamental Mode Circuit in Figure 9.

signal of X_2 from passing through an inverter to affect the output Y_1). A transition does not occur until the changing of X_2 switches $\bar{X}_2$ in the first K-Map in Figure 10(a) which transits from cell 5 to cell 1. Thus, this transition turns the output Y_1 on. Another transition takes place and jumps to cell 3. The system is in state 11 as stated. This output is the expected output for this input change and incorrect operation was not detected.

Similarly the system is started in the total state 0110 shown in cell 6 in K-Map of Figure 10. As X_1 changes from zero to one, the output falls in cell 12 and displays the expected output for this input change. It should also be noted that an essential hazard does not occur in this transition.

From the previous discussion and analysis, only the first case (initially at (1) with a change of X_2) contains an essential hazard while the other two transitions are free of hazard. This argument shows that the possibilities of detecting the occurrence of the essential hazard by Unger's method[33] is not always the right approach. Therefore, a new definition is stated as follows:

Definition : Suppose that there exists a stable state S, an input X and several present states y in a sequential function. Starting from S, three consecutive changes in input X bring the system to a stable state other than the one after the first change in input X. By looking at the

K-Map, a change of X (but not $\overline{X}$) may affect a change in one of the present states y. Then an essential hazard is produced in this transition.

Experiment Result

An input is generated by a KIM-1 microcomputer[19]. The purpose of using a microcomputer to produce the input is that it arbitrarily may be changed to any form of inputs as needed. In addition, inputs to the circuit would arrive at the same time.

Figure 11 shows the interconnection between the KIM-1 microcomputer and the fundamental mode circuit. A program to generate this input sequence is listed in Appendix A[18]. Outputs are sent out from PA0 and PA1 of the microcomputer and fetched into inputs X_1 and X_2 of the system. The maximum frequency of this input sequence is 45.5 KHz (or a period of 22 ns).

In this experiment, a Hewlett-Packard model 1615-A logic analyzer is used to synthesize the logic behavior of any sequential circuit. The input sequence listed below was applied to the system.

X_1 : 0 0 1 1 0 0 1 1

X_2 : 0 1 1 0 0 1 1 0

This circuit would behave as follows: The system is initially at (1) under inputs Xs(00) as shown in Table II on page 23. Then it transits to (2)(2)(3)(3)(4)(1)(1) and finally stays at (1) waiting for the next input sequence.

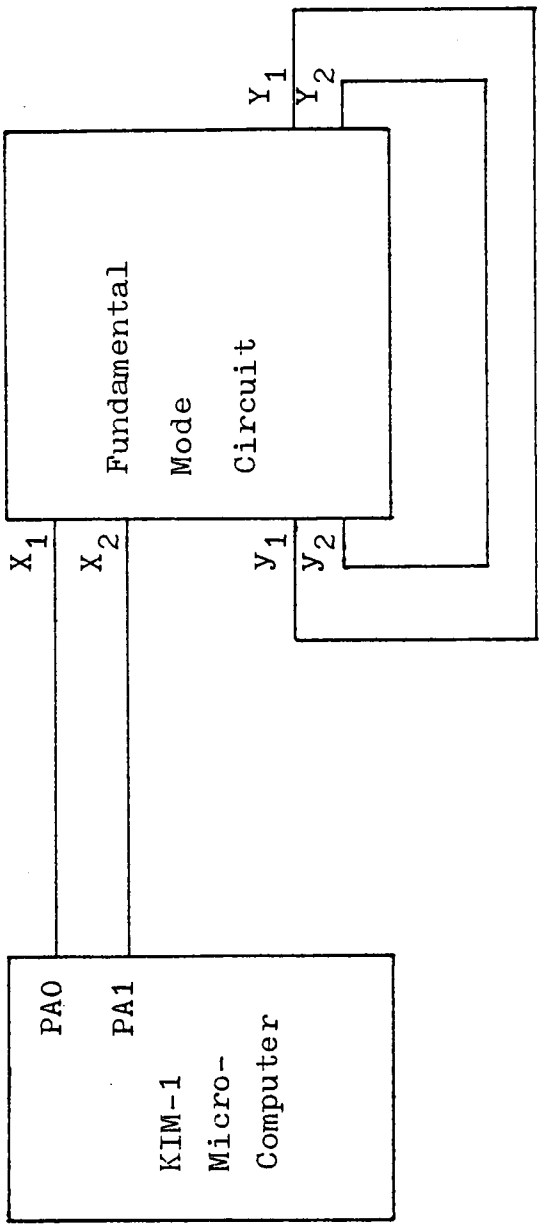


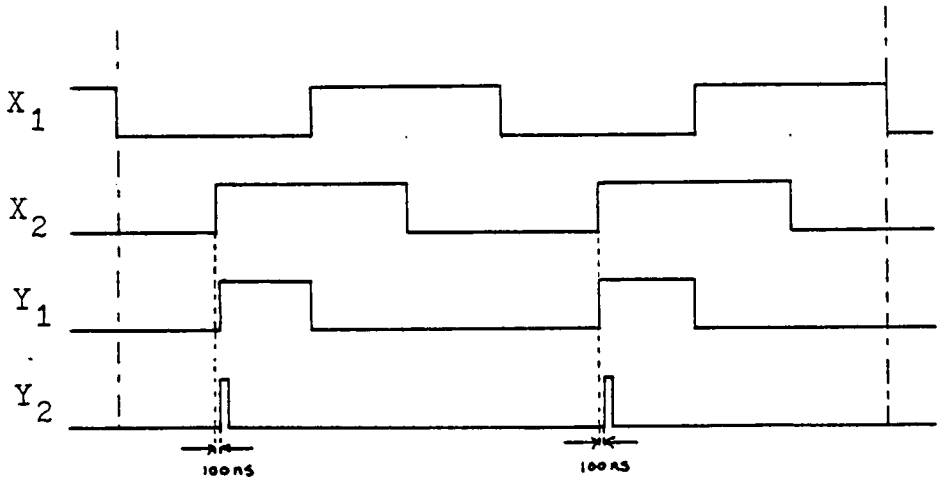
Figure 11. Interconnection Between KIM-1 Microcomputer and the Fundamental Mode Circuit.

A delay element is added in the path of $\overline{X}_2$ in Figure 9. The minimum delay time with the magnitude of 42.5 ns results in the occurrence of an essential hazard during the changing of X_2 from zero to one. The timing diagram of this behavior is shown in Figure 12(a). Figure 12(b) shows the waveform of the pulse in Y_2. As illustrated in Figure 12(a), when X_2 switches to one at t_1, the next state Y_1 and Y_2 would momentarily go to 11 for 60 ns and then switch to 10. As inputs continue to come in, the next state stays at 00 until the next transition of input X_2 goes from zero to one at t_2. This proves the result arrived at from the discussion in previous two sections that essential hazard would occur if there is a stable state starting with a change of input X_2 from zero to one, as shown in Table II on page 23.

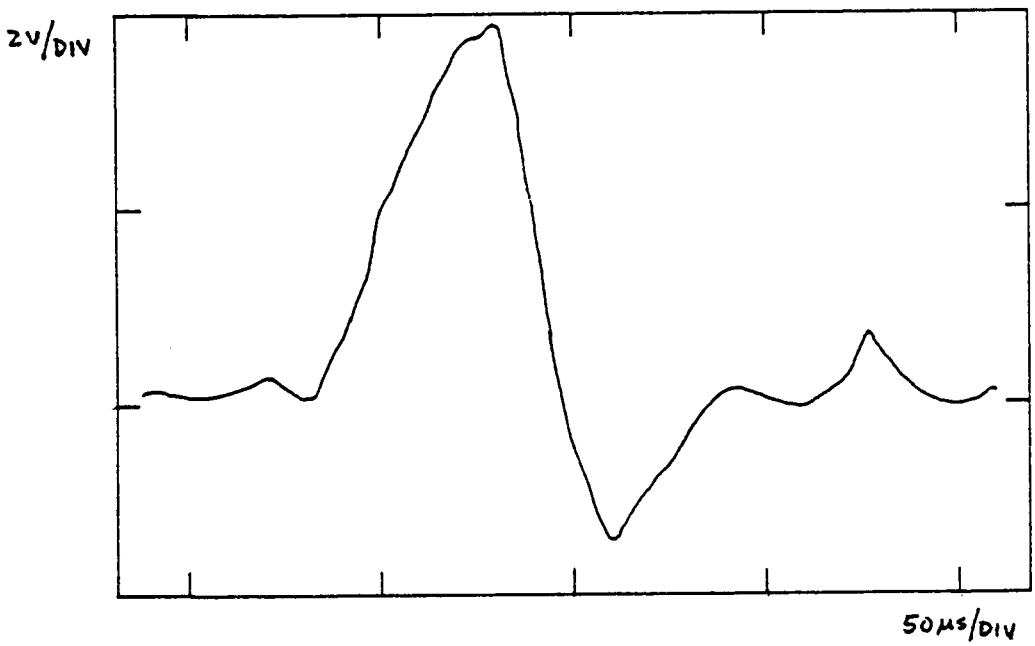
Another delay element is placed in the path of $\overline{X}1$ in Figure 9. No matter how large a delay is inserted, there is no trace to signal that an essential hazard is contained in this transition. The output state of the circuit still responds correctly. Similarly, this gives the same result if delay elements are put in the feedback paths of Y_1 and Y_2. The experiment that has been conducted illustrates that the theory described in the previous section is correct.

Other Consideration

An electric dryer was used to heat the particular gates to examine if the temperature affected the output state. As the inverter gate of X_2 (INV 2 in Figure 9) was



(a)



(b)

Figure 12. Logic Behavior of the System by Adding a Delay Element in the Path of $\bar{X}_2$.
(a) Timing Diagram, (b) Expansion of the Pulse at Y_2.

heated, the output became unstable. This output state might start from stable state (1) or stable (3) in Table II on page 23. Thus, the possibilities of the occurrence of the hazard are greatly increased when the temperature of the gates is increased. A graph of propagation delay time to logical 1 vs temperature is shown in Figure 13[28]. This graph shows that as the temperature increases, the delay time becomes longer.

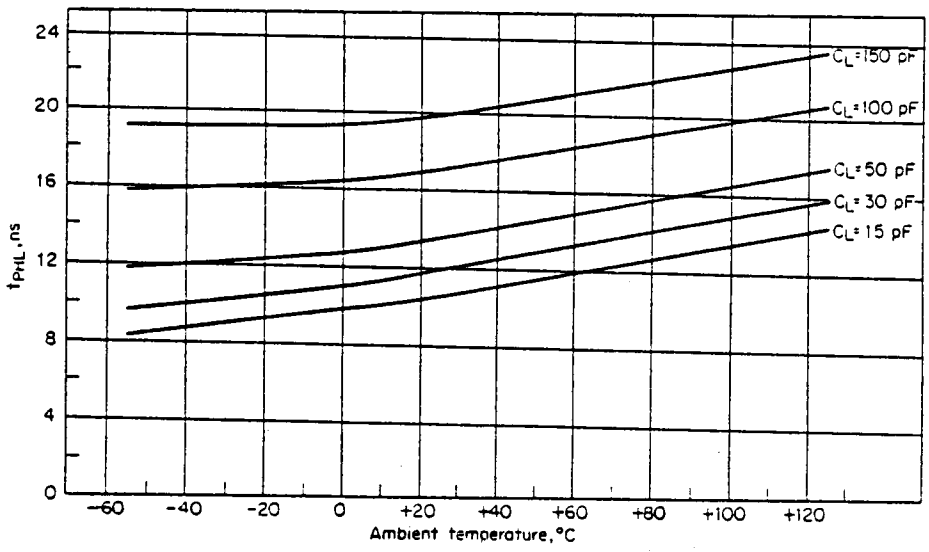


Figure 13. Typical Propagation Delay Time to Logical 1.

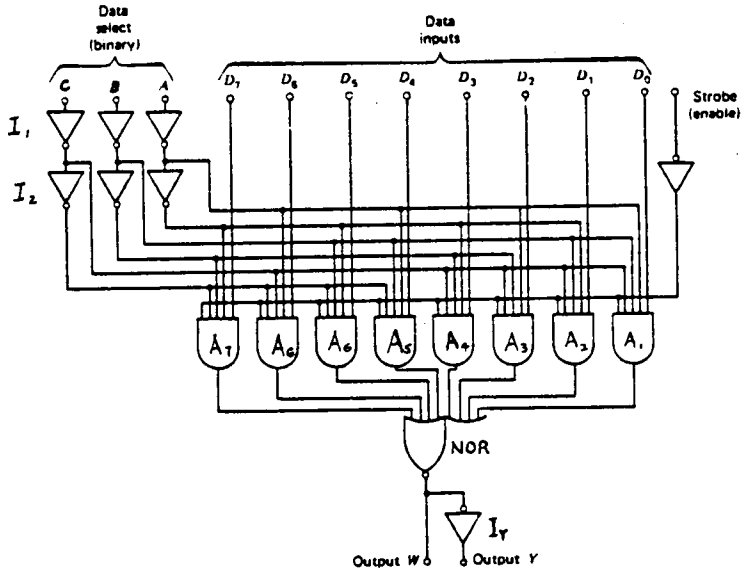
CHAPTER III

DESIGN WITH MULTIPLEXER

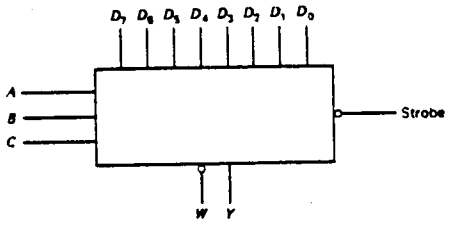
The design of a digital system has became more complicated in recent years. One cannot employ traditional logic design techniques (i.e., implementing by random logic gates) to design such large systems. Therefore, certain logic combinations are available in the form of standard products that are better alternatives than discrete gates. One should always choose to use them, and conventional design techniques should be considered a last resort – to be used only when higher levels of integration cannot be used.

One of the most generally useful MSI decvices is the digital multiplexer (MUX). Because it functions very much like a selector switch, this device is sometimes called a selector. Figure 14 shows the internal circuit, block diagram and truth table for an eight input multiplexer/ selector[29]. Three bit binary codes A, B and C determine one of the inputs to be selected and appear at the output of the circuit. They are termed select inputs. The lines labelled I_0 through I_7 are called address inputs. Quad two-input, dual four-input and one 16-input packaged multiplexers are available in the market.

In general, a multiplexer could be designed directly from the truth table (unless logic equations are first obtained). Once the truth table is derived, one could



(a)



(b)

Inputs												Outputs	
C	B	A	Strobe	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Y	W
X	X	X	1	X	X	X	X	X	X	X	X	0	1
0	0	0	0	0	X	X	X	X	X	X	X	0	1
0	0	0	0	1	X	X	X	X	X	X	X	1	0
0	0	1	0	X	0	X	X	X	X	X	X	0	1
0	0	1	0	X	1	X	X	X	X	X	X	1	0
0	1	0	0	X	X	0	X	X	X	X	X	0	1
0	1	0	0	X	X	1	X	X	X	X	X	1	0
0	1	1	0	X	X	X	0	X	X	X	X	0	1
0	1	1	0	X	X	X	1	X	X	X	X	1	0
1	0	0	0	X	X	X	X	0	X	X	X	0	1
1	0	0	0	X	X	X	X	1	X	X	X	1	0
1	0	1	0	X	X	X	X	X	0	X	X	0	1
1	0	1	0	X	X	X	X	X	1	X	X	1	0
1	1	0	0	X	X	X	X	X	X	0	X	0	1
1	1	0	0	X	X	X	X	X	X	1	X	1	0
1	1	1	0	0	X	X	X	X	X	X	0	0	1
1	1	1	0	X	X	X	X	X	X	X	1	1	0

When used to indicate an input, X = irrelevant.

(c)

Figure 14. Eight-input Multiplexer/Selector (74151). (a) Logic Diagram, (b) Block Diagram, (c) Truth Table.

choose any N-1 variables as the select inputs (where N is the number of variables in the system). The residue variable would then become the input function of the multiplexer. For example, a table with four variables A, B, C, D and one output function f is given in Table IV. If one chooses A, B, C as select inputs, then variable D is separated from different combinations of A, B and C and is given in Equation (III-1).

$$f = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + ABCD \quad \text{(III-1)}$$

This circuit can be made with an 8-input MUX. One could find the input functions required by simply making a table of the eight multiplexer input address combinations. This is shown in Table V. The "other variable" column of this table is taken directly from the logic equation (III-1). For each product term of the equation, an entry is made on the table. For example, variable $\bar{D}$ in the first term $\bar{A}\bar{B}\bar{C}\bar{D}$ is entered in the $\bar{A}\bar{B}\bar{C}$ row.

The multiplexer inputs in Figure 15 could be read directly from Table V where each row represents one address input. Since $D + \bar{D} = 1$ in Figure 15, a true logic level (+5V) is simply connected to inputs I_2 & I_3; and a False logic level (ground) is connected to I_1 and I_5 because there is nothing in row 1 and 3 in Table V.

Figure 16 shows how to work directly from the Veitch diagram rather than from the equation. The number in

Table IV. A Truth Table For Using the Multiplexer.

Input				Output
A	B	C	D	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Table V. Multiplexer Inputs to Generate Function f.

Address Input	Select Input	Other variable
I_0	$\bar{A}\bar{B}\bar{C}$	$\bar{D}$
I_1	$\bar{A}\bar{B}C$	0
I_2	$\bar{A}B\bar{C}$	$D + \bar{D}$ (or 1)
I_3	$\bar{A}BC$	$D + \bar{D}$ (or 1)
I_4	$A\bar{B}\bar{C}$	D
I_5	$A\bar{B}C$	0
I_6	$AB\bar{C}$	$\bar{D}$
I_7	ABC	$\bar{D}$

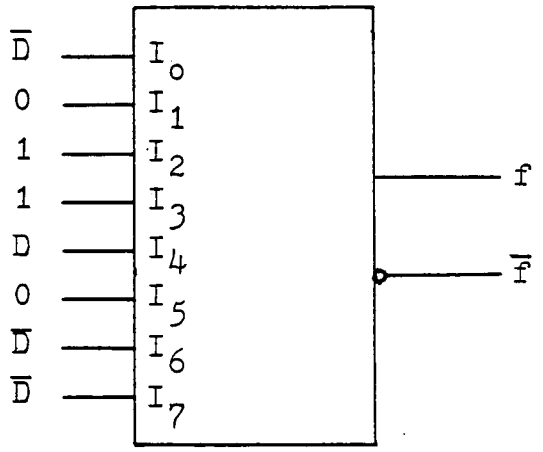


Figure 15. A Multiplexer as a Logic Function Generator.

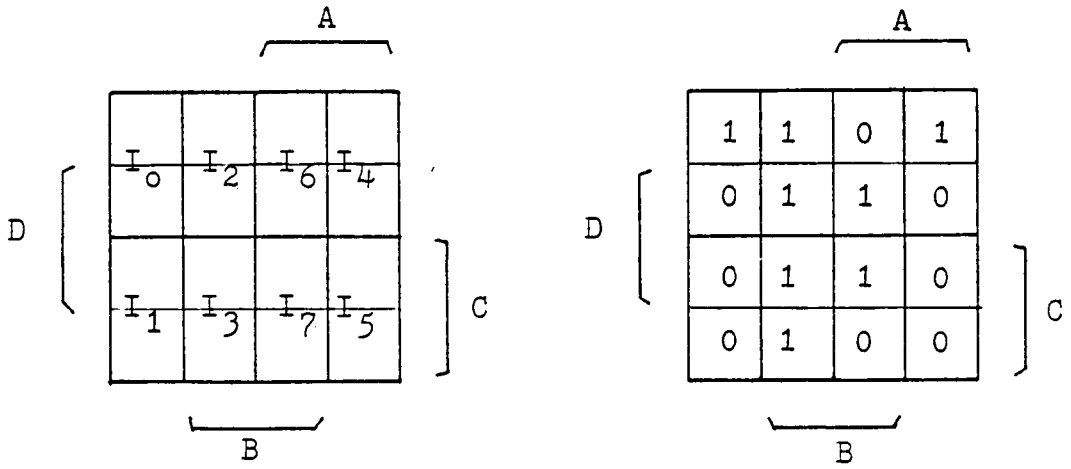


Figure 16. Veith Diagram Method.

Figure 16(a) corresponds to the multiplexer input number. The eight rectangles of Figure 16(b) indicate whether to connect D, $\overline{D}$, 1 or 0 to that address input of the multiplexer.

In addition, one may use some gates to generate "residue" functions that are connected to the multiplexer inputs. When this method is applied, M variables are chosen as select inputs (where M < N-1, N is the total number of variables in a system). As in the example above, referring to Table IV, one may "factor out" A and B from this function. This yields four separate functions of 2 variables which are connected to the address inputs. A general equation is given in Equation (III-2)

$$f_1 = I_o \cdot \overline{A} \cdot \overline{B} + I_1 \cdot \overline{A} \cdot B + I_2 \cdot A \cdot \overline{B} + I_3 \cdot A \cdot B \qquad \text{(III-2)}$$

Inputs I_o, I_1, I_2 and I_3 in Equation (III-2) contain two variables, C and D, which are different from address inputs in Table V. Those inputs produce some "residue" functions which are formed with several gates. This method might save more IC package than the previous method. However, the designer should make his own judgement to use the method which utilizes the minimum chips.

The Design Procedure

Consider the same circuit that is described in the previous chapter. A truth table shown in Table VI was set up with four inputs X_1, X_2, y_1 and y_2 where y_1 and y_2 are

Table VI. Truth Table of Asynchronous Circuit in Figure 9.

X_1	X_2	y_1	y_2	Y_1	Y_2
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	0	0
1	1	1	1	0	1

present states and outputs Y_1, Y_2 (Y_1 and Y_2 are next states). Since the purpose of this investigation is to detect hazards in multiplexers, only one MUX is used for each output function. Depending upon which three variables are chosen to connect to the address inputs, the functions (to be connected to eight address inputs) would be found from various partitions of the Veith diagram given in Figure 17. Each subdivision could be thought of as a smaller three-variable Veith diagram for the required function.

The first Veith diagram in Figure 17(a) was used in this design. Then, each partition of the Veith diagram was filled with zero and one which is shown in Figure 18(a). A table of the required input functions from the Veith diagram in Figure 18(a) was set up in Figure 18(b). Finally, these functions were simply read off the Veith diagram rows in their simplest form — just as though each row were a Veith diagram in three variables. The circuits required to generate those functions are shown in Figure 18(c). Low logic levels of inputs were connected to ground and high levels of inputs were connected to a power source (+5V). Output Y_1 was fed back into one select input, and output Y_2 was fed back into address inputs of the MUX.

Hazard Consideration in Multiplexer

The structure of the internal logic diagram in Figure 14(a) is an AND-OR logical network. Each product term of the multiplexer represents a grouping of two adjacent

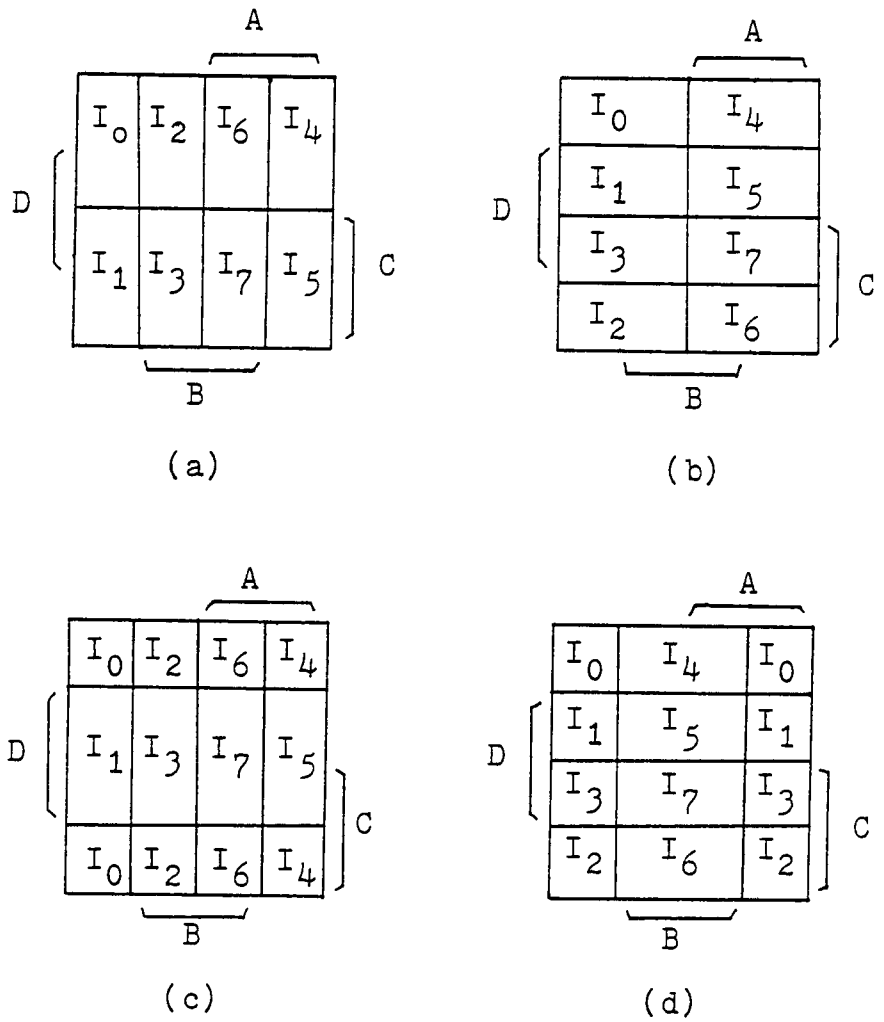
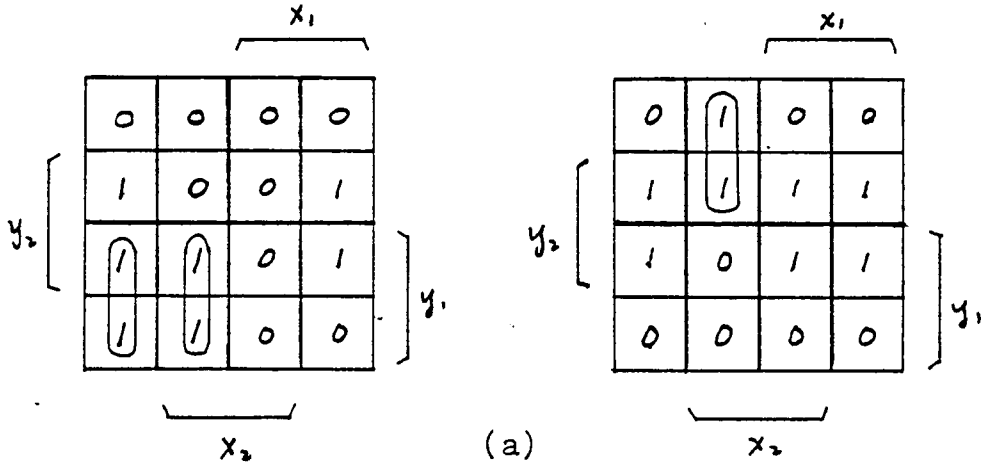


Figure 17. Four Possible Ways to Reduce Function by Three Variables (I_n=multiplexer input number): (a) A, B and C, (b) A, C and D, (c) A, B and D, (d) B, C and D.



Input	Y_1	Y_2
I_0	y_2	y_2
I_1	1	y_2
I_2	0	1
I_3	1	0
I_4	y_2	y_2
I_5	y_2	y_2
I_6	0	y_2
I_7	0	y_2

(b)

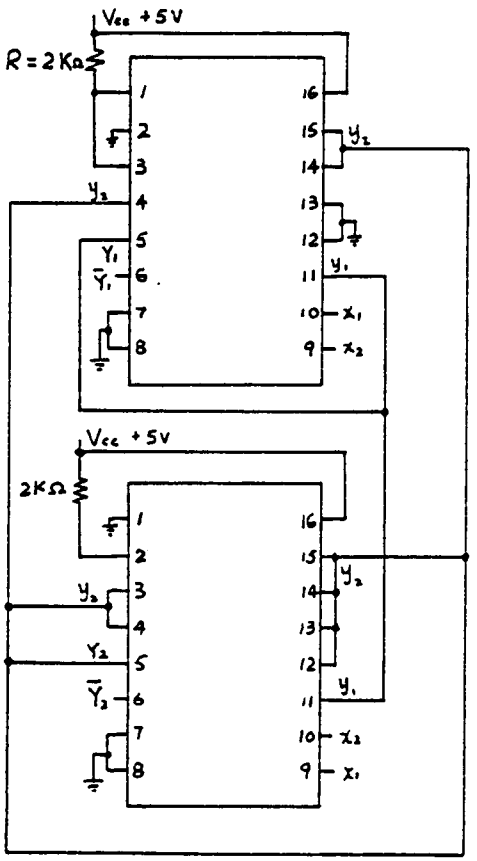


Figure 18. Procedure for Designing Level Mode Circuit With MUX.
(a) Step 1: Veith Diagram used to Generate a table of Function, (b) Step 2: Residue Function, (c) Step 3: Logic Diagram.

cells in the K-Map. For example, cell 0 and cell 1 shown in Figure 18(a) are combined to be in one product term. Two types of hazard might exist in the multiplexer. First, a static hazard may exist when a changing input requires corresponding minterms to be covered by different product terms. The other type is the essential hazard which may exist if there are unequal delays in the gates represented by those product terms.

Measurement

The propagation delay times in the gates of the multiplexer were measured and listed in Table VII. The column labelled "Delay Path" shows the path that signal travel in the multiplexer. Symbols I_1 and I_2 are INVERTER gates at select inputs in Figure 14(a) on page 38. Symbol A represents any AND gates of A_o to A_7 in Figure 14(a). The notation of each propagation delay time is labelled on the first column in Table VII. Then, the delay time of each element (or between two gates) in the MUX was calculated and listed in Table VIII. In this table, the first row shows the delay time between I_1 and I_2 in Figure 14(a) on page 38 which is derived by substracting t_{S-Y} and t_{D-Y}. Similarly, the second and third rows were derived by the same procedure. Table VIII illustrates that the maximum propagation delay time in each gate is 6 ns. This delay time is much less than the maximum delay time in any gate in the random logic circuit. Therefore, this time interval is too

Table VII. Switching Characteristic of 74151 MUX.

V_{cc} = 5V T_A = 25°C					
Notation	From	To	Transition for Propagation Delay Time	Delay Time	Data Path
t_{S-Y}	A,B or C	Y	0→1	36 ns	$I_1 \rightarrow I_2 \rightarrow A \rightarrow NOR \rightarrow I_Y$
			1→0	42 ns	$I_1 \rightarrow I_2 \rightarrow A \rightarrow NOR$
t_{S-W}	A,B or C	W	0→1	30 ns	$I_1 \rightarrow I_2 \rightarrow A \rightarrow NOR$
			1→0	43 ns	$I_1 \rightarrow A \rightarrow NOR$
t_{D-Y}	Any D	Y	0→1	46 ns	$A \rightarrow NOR \rightarrow I_Y$
			1→0	31 ns	$A \rightarrow NOR \rightarrow I_Y$
t_{D-W}	Any D	W	0→1	34 ns	A NOR
			1→0	25 ns	

Table VIII. Switching Characteristic of elements in MUX.

Device	Derivation	Propagation Delay Time	
$I_1 + I_2$	$t_{S-Y} - t_{D-Y}$	0→1	10 ns
		1→0	11 ns
Any A + I_2	$t_{PLH\ S-Y} - t_{PHL\ S-Y}$		8 ns
I_Y	$t_{S-Y} - t_{S-W}$	0→1	6 ns
		1→0	1 ns

small to cause a glitch at the output of the multiplexer. The circuit always responds to the correct output. (The above discussion is valid only when one output function is generated.)

Another Hazard Consideration in the Multiplexer

Consider the system in Figure 19, the detailed block diagram of the circuit in Figure 18(c). Three select inputs of the multiplexer in Figure 19 are connected to X_1, X_2 and y_1. Address inputs of these two multiplexer are labelled according to the Veith diagram in Figure 18(a). Assume that input X_2 in the multiplexer A in Figure 19 has a large delay associated with it in comparison to other MUX. When the system is initialized at $X_1 = X_2 = y_1 = y_2 = 0$, a change of X_2 from zero to one takes place. This changing signal first changes the select line from I_0 to I_2 in the multiplexer labelled B and causes output Y_2 to switch to unity. Since a large delay in input X_2 prevents the changing signal of X_2 from arriving at the product term of multiplexer A in Figure 19, output Y_1 is changed to one by the present input y_2. Then, the select input chose I_3 in both multiplexers, causing Y_1 to remain at one and Y_2 to switch to zero. The circuit is stable at $y_1 = 1$ and $y_2 = 0$. In Table II on page 20, a change in input X from 00 to 01 starting from state 00 would fall at 01. Therefore, the above result shows the occurrence of a hazard in this transition. Similarly, the same result would happen when the system started at $X_1 = 0$,

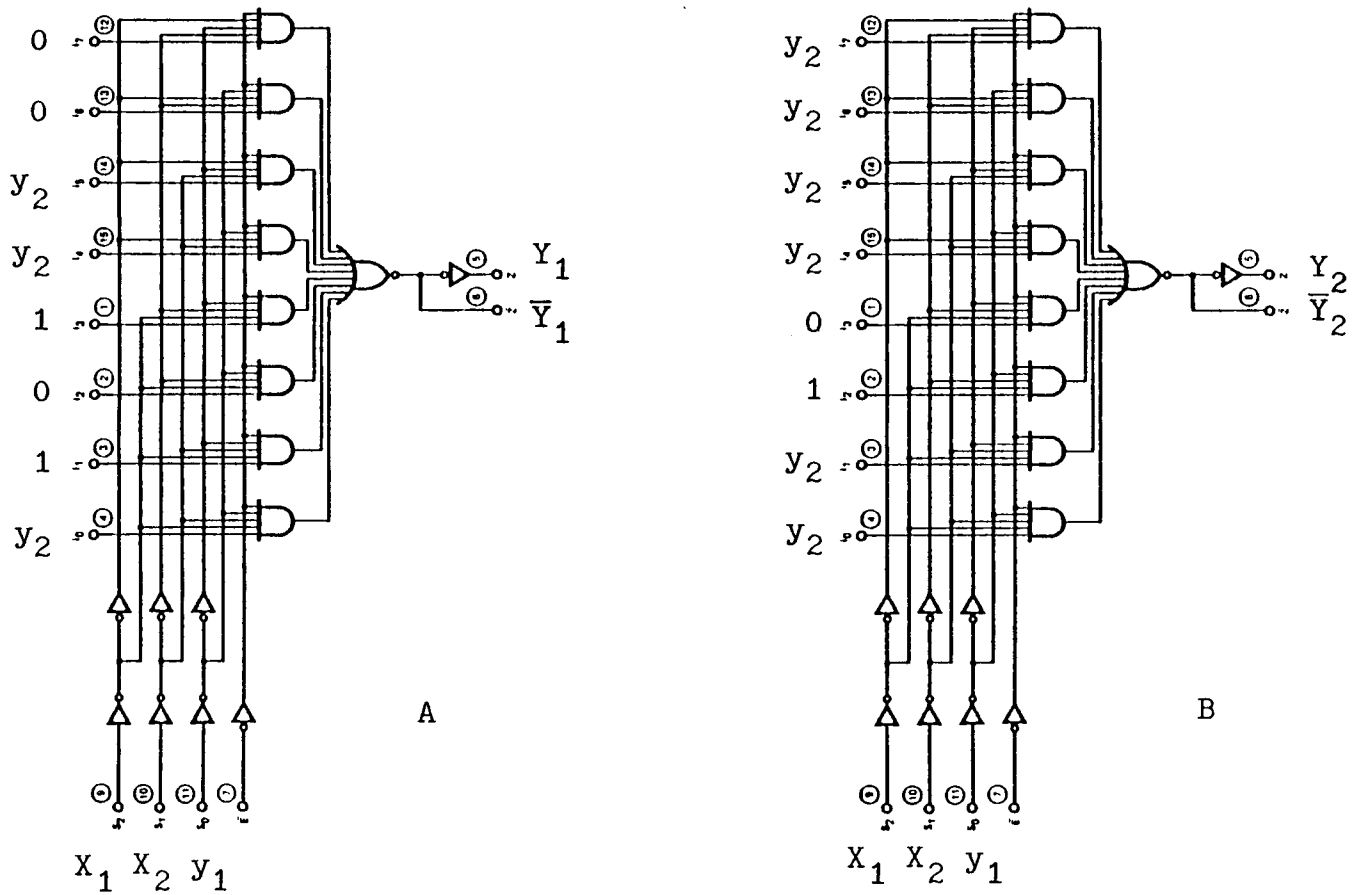


Figure 19. The Detail Block Diagram of the System.

$X_2 = 1$, $y_1 = 0$ and $y_2 = 1$, and a change of X_2 to zero with a delay at X_2 of multiplexer B. Also, if this system started at $X_1 = 0$, $X_2 = 1$, $y_1 = 1$ and $y_2 = 0$, a change in X_1 to one with a delay at X_1 of multiplexer B would produce a similiar results. Thus a hazard may occur when two or more multiplexers are used.

Experiment

The circuit diagram in Figure 18(c) was constructed. Inputs X_1 and X_2 were delivered from KIM-1 microcomputer for the same purpose described in chapter 2 for the testing of fundamental mode circuit using random logic gates. The result turned out as expected, and no trace of incorrect operation could be seen from the logic analyzer. This same input sequence was sent out from an input sequence generator circuit as described in Appendix B with an input frequency of 909 KHz (or period T = 1.15 ns). The interconnection of this circuit and the testing circuit is shown in Figure 20. Similarly, the circuit still responds without any hazards. Then, a delay element was added to the select input X_2 in multiplexer A in Figure 19. A hazard was produced with a magnitude of 50 ns in the delay element. The timing diagram is similar to the one in Figure 12 on page 34. Similarly, the other two hazards discussed in the previous section occurred with the same magnitude of delay time at either input X_1 or X_2.

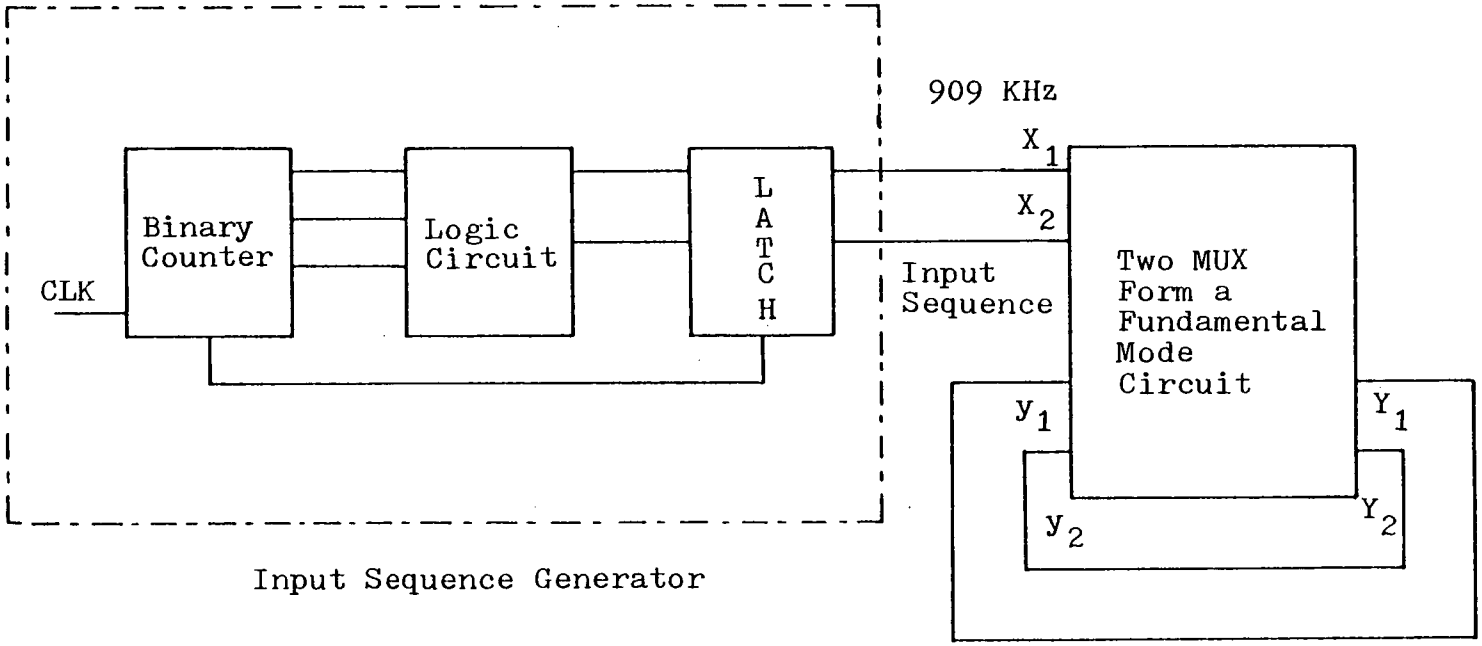
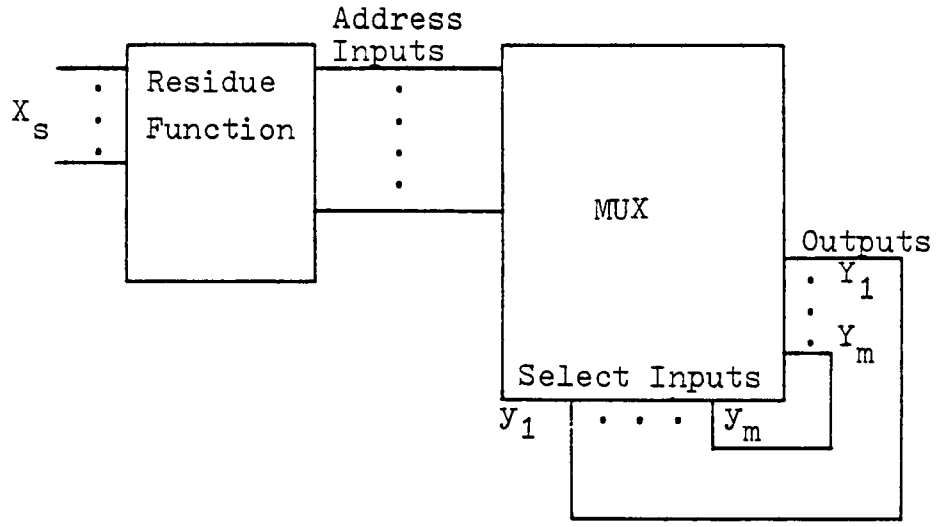


Figure 20. Interconnection Between Input Sequence Generator and the Fundamental Mode Circuit.

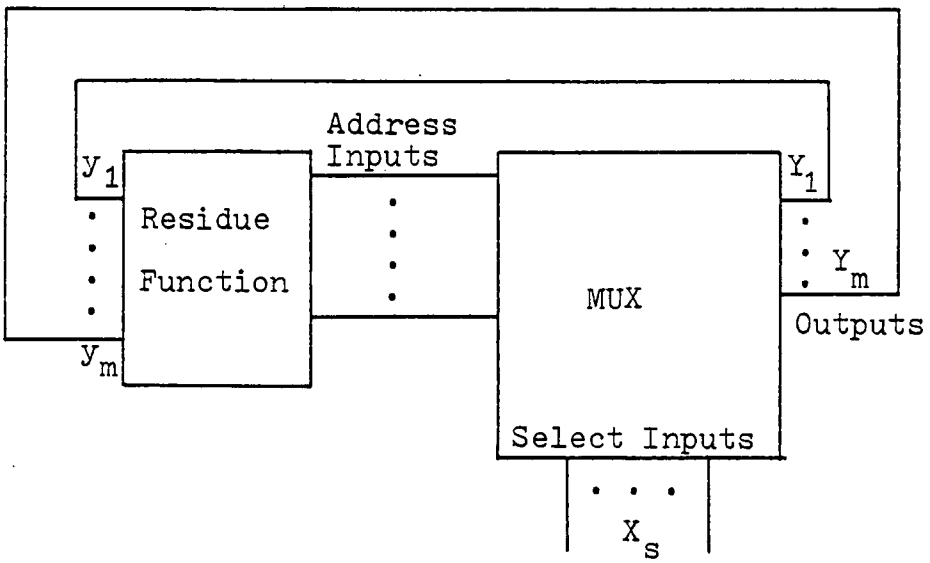
Other Considerations

As mentioned previously, the designer may use some residue functions with the multiplexer; that is, by putting some logic gates as residue functions before the address inputs. Figure 21 shows two ways to connect inputs Xs and feedback loops ys. Inputs Xs and present states ys are connected to address inputs and select inputs respectively, shown in Figure 21(a). In the other method, as illustrated in Figure 21(b), Xs are connected to select inputs, and ys are being connected to address inputs. In both cases, a residue function circuit is presented before the address inputs of the multiplexers. If the first method is applied, there must be at least one inverter for Xs. The signals of these Xs would pass through several logic gates before entering the address inputs of the multiplexers. If one of those gates has a large delay compared to other gates, the multiplexer would be first affected by feedback signals before the direct effect from input signals. Then a hazard would be noted during this transition of input X. If the second method is applied, inputs Xs would immediately affect the output before feedback signals entered the address inputs. Hence, a hazard would not presented in the circuit.

Another type of malfunction in the circuit that might happen is discussed as follows: This malfunction is due to the power set up. When power was being turned on, inputs and outputs of the multiplexers may randomly start from any state. As a result, this circuit might not begin from the



(a)



(b)

Figure 21. Two Ways of Connection with Residue Function.
(a) X_s Connect to Address Input and y_s Connect to Select Input,
(b) y_s Connect to address input and X_s Connect to Select Input.

expected initial state and a malfunction would take place. One would use the enable bits in the multiplexer in Figure 14(c) on page 38 to solve this problem. When the enable bit is at logical 1, output would always stay low. When this capability is applied, a positive pulse is generated when power is on. This would force the circuit to start from state 00 (always defined as initial state). When this method was used, the circuit functioned correctly no matter how and when the power was turned on, thus proving the validity of this method.

CHAPTER IV

DESIGN WITH READ ONLY MEMORY

Designing and implementing an asynchronous sequential circuit with read only memory (ROM) is another example of using MSI and LSI devices[14]. Since the structure of ROM is standardized, one can create many different kinds of logic circuits, just as Henry Ford created the same identical car — model T — which reduced the cost of production. According to him, one could find the identical Ford model car, but in different colors. This is called "customized standardization". When this concept is applied to the memory system in the computer and digital system, it is possible to design at the highest level of sophistication with any options that a designer wants.

Read only memories represent the ultimate solution in customized standardization: for example, one can actually specify a 1 or 0 in any of the NxM bit locations, where N is the number of words in the ROM and M is the number of bits in one word. Thus the number of possible combination is 2^{NxM}.

The development of a semiconductor ROM has improved so rapidly that a succession of faster, larger and more flexible devices have been produced. They are widely used in code decoding, character generation, micro-instruction generation and code checking/correcting. Read only memory is different from random access memory (RAM) in that it has

writing ability. However, it is also a random access device.

Even though custom mask preparation is totally automated, mask programming of ROMs is pratical only when customers use a minimum of approximately 100 identical devices. Another disadvantage of the mask-programmed ROM is that making any change is virtually impossible if an error is discovered in the design. Thus, it is important to have another kind of ROM for small production and for initial check-out and pilot production. The programmable read only memory (PROM) [31] meets the above requirements and can be manufatured by millions.

In this chapter, the fundamental knowledge of semiconductor MOS memories is discussed. It is also desirable to present the detail of designing a sequential circuit with ROM. A fair amount of discussion is devoted to the testing of PROM and the occurrence of the possible hazards in the circuit. It would be useful, of course, to assume that the hazard correction technique will continue indefinitely into the furture.

The Basic Idea of Erasable Programmable Read Only Memory and Its Technology

A PROM (or EPROM), having the same structure as ROM, is an array of selectively open and closed unidirectional contacts. It has the decoding structure example[22] shown in Figure 22 which is a 16-bits array with 4 input addresses and one output line. Two of the address lines, A_1 and A_2,

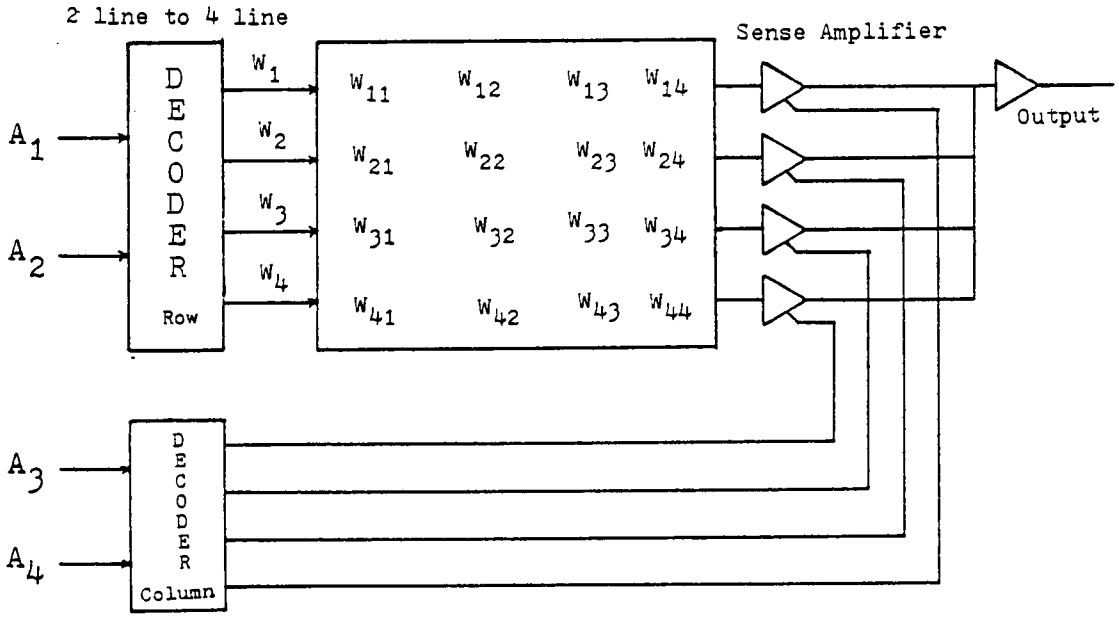
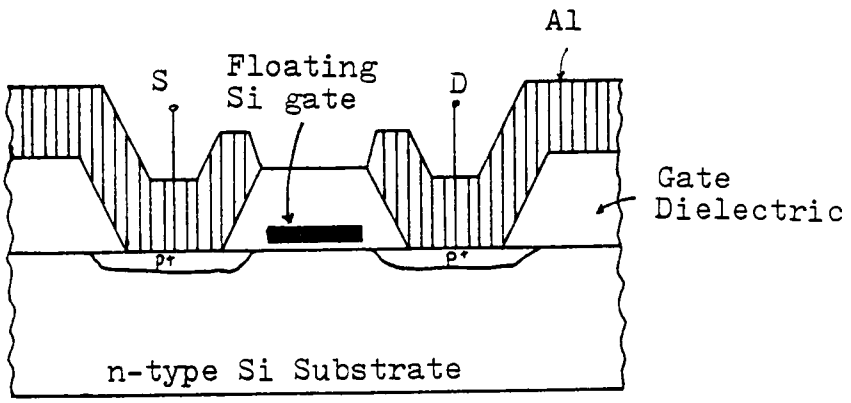


Figure 22. Realization of Decoder Logic in PROM. (4 bit x 4 bit).

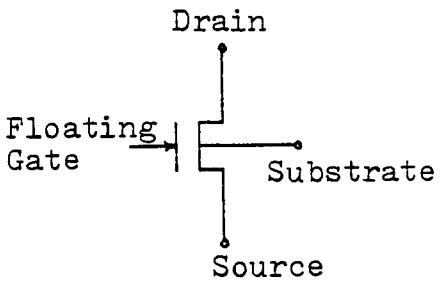
are decoded and used to energize one of the four row lines, w_1 to w_4 in Figure 22. This,in turn, activates those column lines which have a closed contact to the one selected row line. The other two lines, A_3 and A_4, are decoded and thus enable one of the column sense amplifiers.

In 1971, Intel introduced a unique erasable PROM that could erase the program information by exposure to correct wavelength and intensity of ultraviolet light. They used the technology called Floating Avalanche-injection MOS (FAMOS) as the charge storage device[9]. A cross-section of this device is shown in Figure 23(a). Figure 23(b) shows a circuit schematic for the device. It is basically similar to having a floating gate on a transistor within each storage cell. The device is essentially a silicon gate MOS field effect transistor in which no connect is made to the silicon gate as shown in Figure 23(b). This gate is electrically isolated, and a charge is transported from the source to drain to the floating gate by a programmed avalanche injection.

A junction voltage in excess of -30V applied to a p-channel FAMOS device will result in the injection of high-energy electrons from the p-n junction surface avalanche region to the floating silicon gates. The presence or absence of a gate charge (at logical 1 or 0) is sensed by measuring the source-drain conductance of the resulting transistor action. Once the applied junction voltage is removed, no discharge path is available for the accumulated



(a)



(b)

Figure 23. FAMOS Storage Cell.
(a) Cross-section,
(b) Circuit Schematic.

electrons. The floating gate is surrounded by silicon dioxide, a dielectric of very low conductivity. This trapped gate charge remains for a long periods of time. As shown in Figure 24, 70% of the initial induced charge will be retained for as long as ten years at 125°C[12].

Once the gate is charge, the charge cannot be removed by electrical pulse because the gate electrode is not electrically accessible. One must write 1s to clear the charge from the gates, and a special technique is used to meet this requirement. When the FAMOS device is illuminated by ultraviolet light having a specific wavelength and intensity, a flow of a photocurrent from the floating gate back to the silicon substrate results, thereby discharging the gate to its initial condition.

Device Used

The memory device used in the experiment is a 2708 static 8192-bits (1024x8 bit) Erasable Programmable Read Only Memory, or EPROM[28]. It is used because of its capability to erase and reprogram if a mistake is found during the testing procedure. The device is packaged in a standard 24-pin package which has a transparent lid to allow erasure by high intensity ultraviolet light. Maximum access time is 450 ns. A block diagram of 2708 EPROM is shown in Figure 25. The low order address bits (A_0-A_3) perform column (or Y) selection, while the high order address bits (A_4-A_9) perform the row (or X) selection. Internal cells in

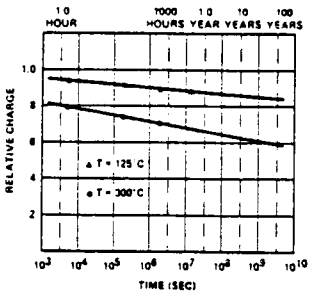


Figure 24. Charge Decay vs. Time.

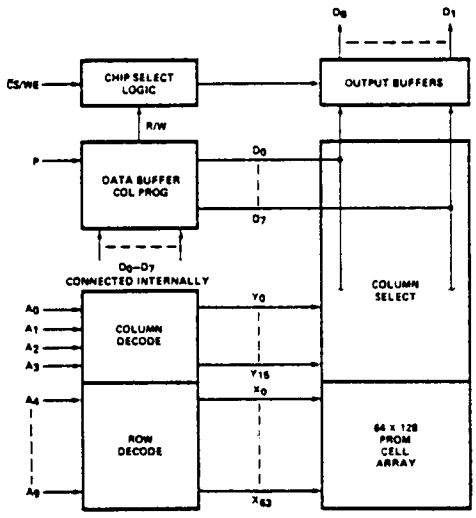


Figure 25. Detailed Block Diagram of PROM.

Figure 25 are interconnected to form a 64x128 matrix or array as shown in Figure 26. Six of the address bits A_4 to A_9 in row selection were decoded to 64 row lines. These row lines provided bias to all the top gates in a particular selected row. The column address connects 8 of the 128 column lines to their respective sense amplifier. Figure 27 illustrates the equivalent schematic of the output buffer. As in the diagram, the output buffer consists of a pair of MOS transistors connected in a push-pull configuration. The line labelled $\overline{CS}$ enables both transistors when true, whereas both output devices are turned off when $\overline{CS}$ is false, thus producing three state output operation.

Design Philosophy

In general, read only memory is used in combinational circuits in which outputs are accessed by the selected input addresses. Read only memory could also be used in the sequential logic design with internal storage. As a result, outputs are a function of the internal state as inputs.

This system was clocked with the asynchronous input in the early studies[3,24]. Sholl and Yang[25] discussed the operation of the normal asynchronous circuit and created an excellent work. The basic structure of their system is shown in Figure 28. Inputs i_j are fetched in an address calculation network. They considered the next state as feedback to the address calculation network with inputs I_s. The output of the address calculation network is the address

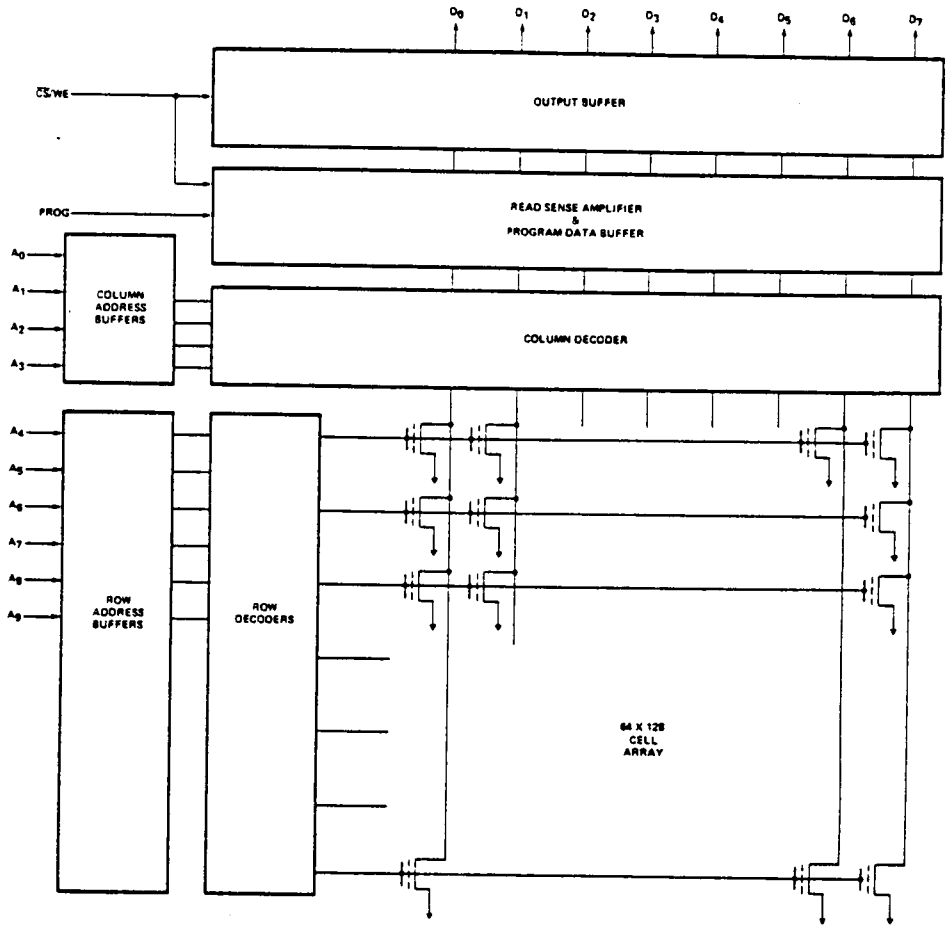


Figure 26. Expanded Block Diagram of 2708.

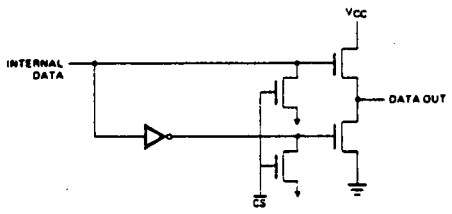


Figure 27. Output Buffer of 2708.

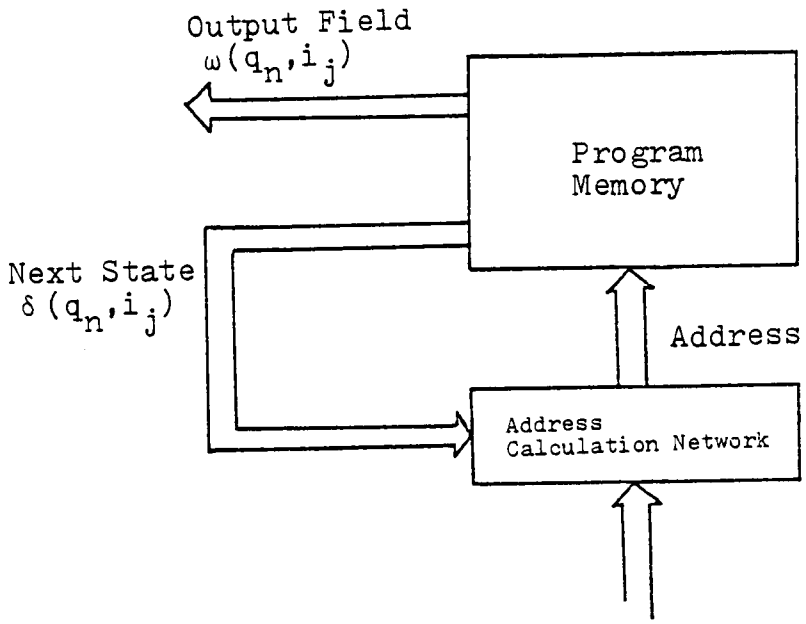


Figure 28. Basic Asynchronous Computing Structure.

input to the program memory which produced the output field $\omega(q_n, i_j)$ and the next state $\delta(q_n, i_j)$ as shown in Figure 28.

The system that will be considered here is the fundamental mode sequential circuit. The combinational part of the circuit is formed by read only memory. Designing with ROM can be done directly from truth table as shown in Table IX, instead of simplifying the logic equations in random logic design.

Each output Y in Table IX represents one function and each 1 on the table represents one canonical minterm form of the expression. For example, output function Y_1 in Table IX has seven 1s which forms minterms numbered 1, 2, 3, 6, 7, 9 and 11.

Because of the orderly nature of the truth table representation of ROM – mechanized logic functions, computer programmed design automation is quite simple. A simple computer program can thus convert logic equations directly into punched cards or paper tape for ROM or PROM programming.

EPROM Programming Procedure

An IM1010 PROM programmer was used to program the memory chip. It is a microprocessor-based universal PROM programming instrument as shown in Figure 29. This equipment has one hexadecimal keypad and a 14-digit alpha-numeric display. The keyboard contains sixteen data keys from 0 to F and three control keys, which are RESET, CLEAR and ENTER.

Table IX. ROM Table.

Address Input				Word	Output	
X_1	X_2	y_1	y_2		Y_1	Y_2
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	2	1	0
0	0	1	1	3	1	1
0	1	0	0	4	0	1
0	1	0	1	5	0	1
0	1	1	0	6	1	0
0	1	1	1	7	1	0
0	0	0	0	8	0	0
0	0	0	1	9	1	1
0	0	1	0	10	0	0
0	0	1	1	11	1	1
0	1	0	0	12	0	0
0	1	0	1	13	0	1
0	1	1	0	14	0	0
0	1	1	1	15	0	1

Figure 29. The IM1010 Universal PROM Programming Instrument: (A) Sixteen Data Keys, (B) Three Control Keys, (C) Ten Function Keys, (D) 14-digit Alpha-Numeric Display, (E) Personal Modules, (F) PROM Socket.

Ten function keys (F) on the mainframe perform fourteen different functions. The system is equipped with 32K bit standard RAM. Easily changed personality modules (E) allow the IM1010 to program a full range of programmable devices. Standard and optional interface allow a wide range of programming system configurations.

This programmer was initialized by pressing the RESET key on the front panel. Then FILL key followed by ENTER was depressed to erase all the content of the RAM. Since there are only sixteen words in Table IX, the author utilized the lower sixteen addresses of the memory to generate the function. An address input table that would program to EPROM is listed in Table X. The EPROM address starts from 000 to 00F and its corresponding input data is given on the right hand side of the table. Those data were then input to the RAM of the programmer by depressing the data input key. After all data had been entered, the FUNC key was typed to list the content of RAM to check if any error were made. Then, A 2708 EPROM was placed in the socket, and this was followed by the entering of PROG, PROM, ENTER and ENTER keys. This chip was programmed, and the programming cycle took about 98 sec. After that, the programmer automatically went into a verify cycle and -VP DONE would be displayed if no errors were discovered. Thus, the data was programmed in and EPROM was ready for testing.

TABLE X

PROGRAM TABLE

Address to PROM (in HEX)	Data Input (in Hex)
0 0 0	0 0
0 0 1	0 3
0 0 2	0 2
0 0 3	0 3
0 0 4	0 1
0 0 5	0 1
0 0 6	0 2
0 0 7	0 2
0 0 8	0 0
0 0 9	0 3
0 0 A	0 0
0 0 B	0 3
0 0 C	0 0
0 0 D	0 1
0 0 E	0 0
0 0 F	0 1

Experiment

A circuit was constructed as shown in Figure 30. Outputs Q_2 and Q_1 (labelled Y_1 and Y_2) were connected to address inputs A_1 and A_o. Inputs X_1 and X_2 were joined with address inputs A_3 and A_2. An input sequence could be generated either from the microcomputer shown in Appendix A or the input sequence generator circuit shown in Appendix B . As displayed on the logic analyzer, hazards could not be visualized from the screen. Then, the power was turned on and off repeatedly until a trace of incorrect operation was observed from the logic analyzer. This circuit started from an incorrect starting state because of the random access property of PROM. When the power was turned on, any bit in the memory matrix would be selected. Then, this signal was fed back to the inputs. If a selected output state caused the circuit to initialize from the incorrect starting state, this circuit would have a malfunction.

It is obvious that the malfunction is caused by the power set-up. The considerations described in Chapter 6 about power set up could be used to solve this problem. (since $\overline{CS}$ in memory does not force the system to start from a particular state, a method of generating a pulse at $\overline{CS}$ when the power was on could not be used.) A power reset table was set up with the original truth table as shown in Figure 31. New bit R was used to indicate the power reset pulse. When R=0, the circuit resumes normal operation.

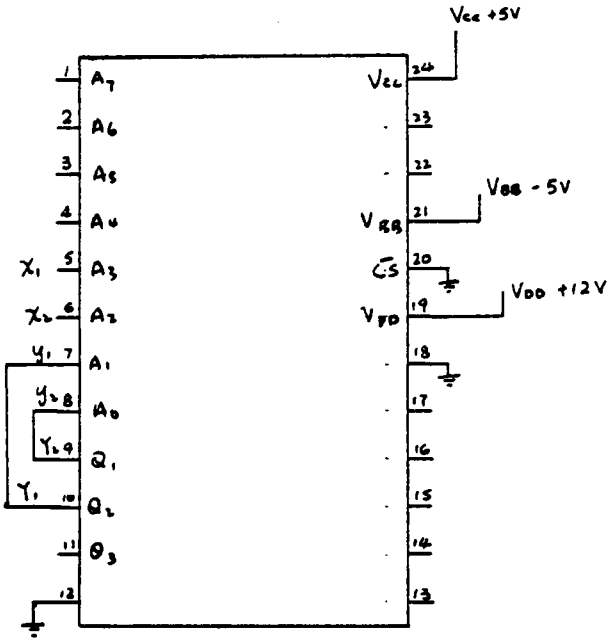


Figure 30. Circuit Assembly.

$Y_1 Y_2$ \ $X_1 X_2$	R=0 00	01	11	10	R=1 00	01	11	10
00	(00)	01	(00)	(00)	(00)	10	01	11
01	11	(01)	(01)	11	00	10	(01)	11
11	(11)	10	01	(11)	00	10	01	(11)
10	(10)	(10)	00	00	00	(10)	01	11

Figure 31. Table With Power Reset.

On the other hand, when R=1, the circuit was reset to the desired state. A pulse was generated when the power was turned on. This reset pulse R should remain at one until the output state becomes stable. Two timing diagrams for R depicting its change from one to zero are shown in Figure 32. Figure 32(a) shows that when R is reset to zero, the output state became unstable for the three input sequences A, B and C. When the fourth input sequence arrived, the output state stabilized, and it functioned correctly. Figure 32(b) shows another response when R goes to zero. The output state was also unstable in the first three input sequences. At the end of the third input sequence C as shown in Figure 32(b), outputs were not set to the starting state. As a result, an incorrect operation took place when the fourth input sequence arrived. Thus, the circuit did not respond correctly with the power reset table. This incorrect operation might due to the static and essential hazards contained in the circuit.

Hazard Testing

This circuit was first set up for testing the static hazard. Outputs Y_1 and Y_2 were not fed back to inputs y_1 and y_2 as in the circuit illustrated in Figure 30. A five bit gray code generator circuit shown in Figure 33 delivered a sequence to the address inputs of memory chips. They are R, X_1, X_2, y_1 and y_2. Input to the gray code generator was a five bit binary code. The conversion table for changing

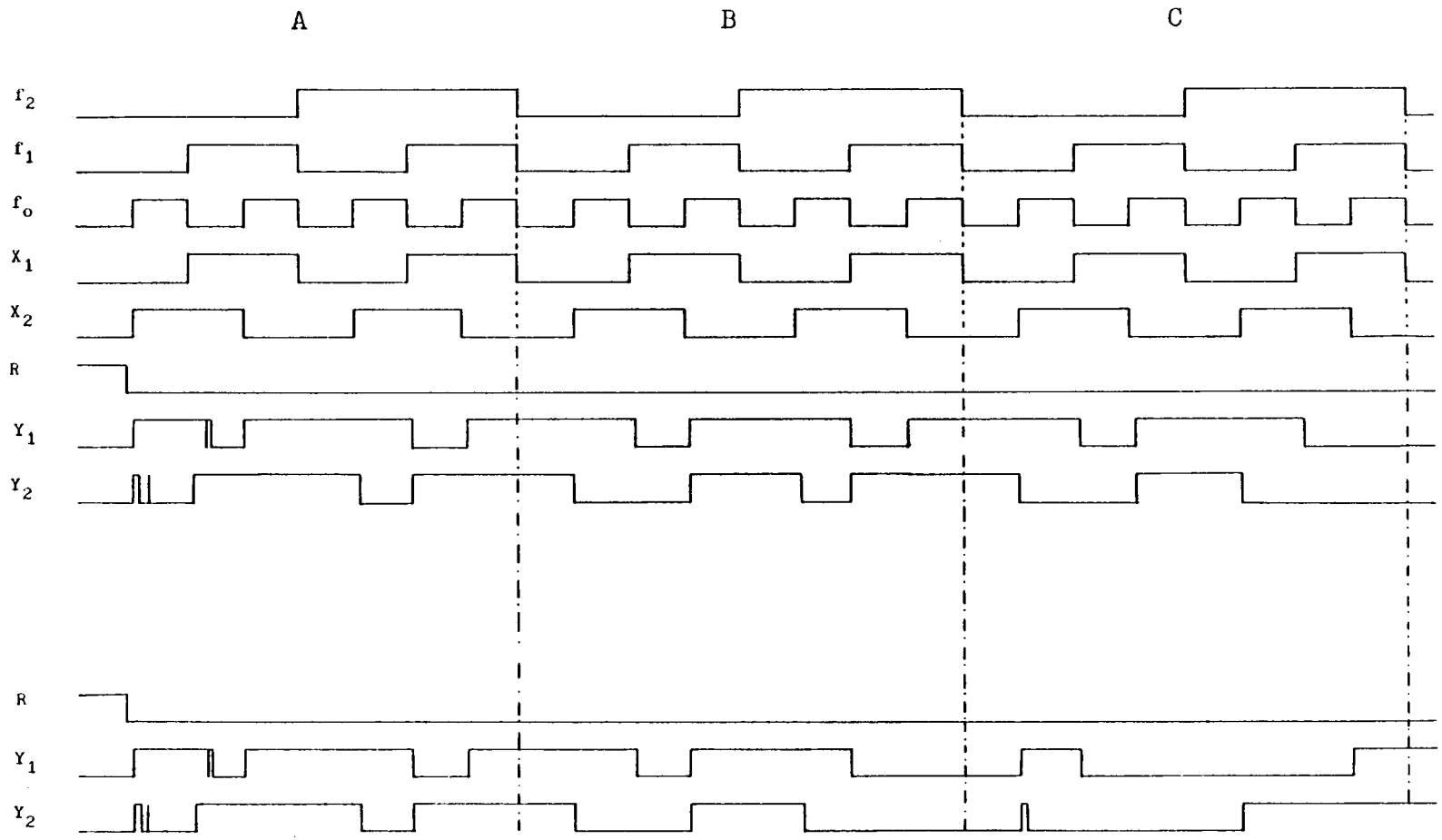


Figure 32. Timing Diagram When R Change from One to Zero.

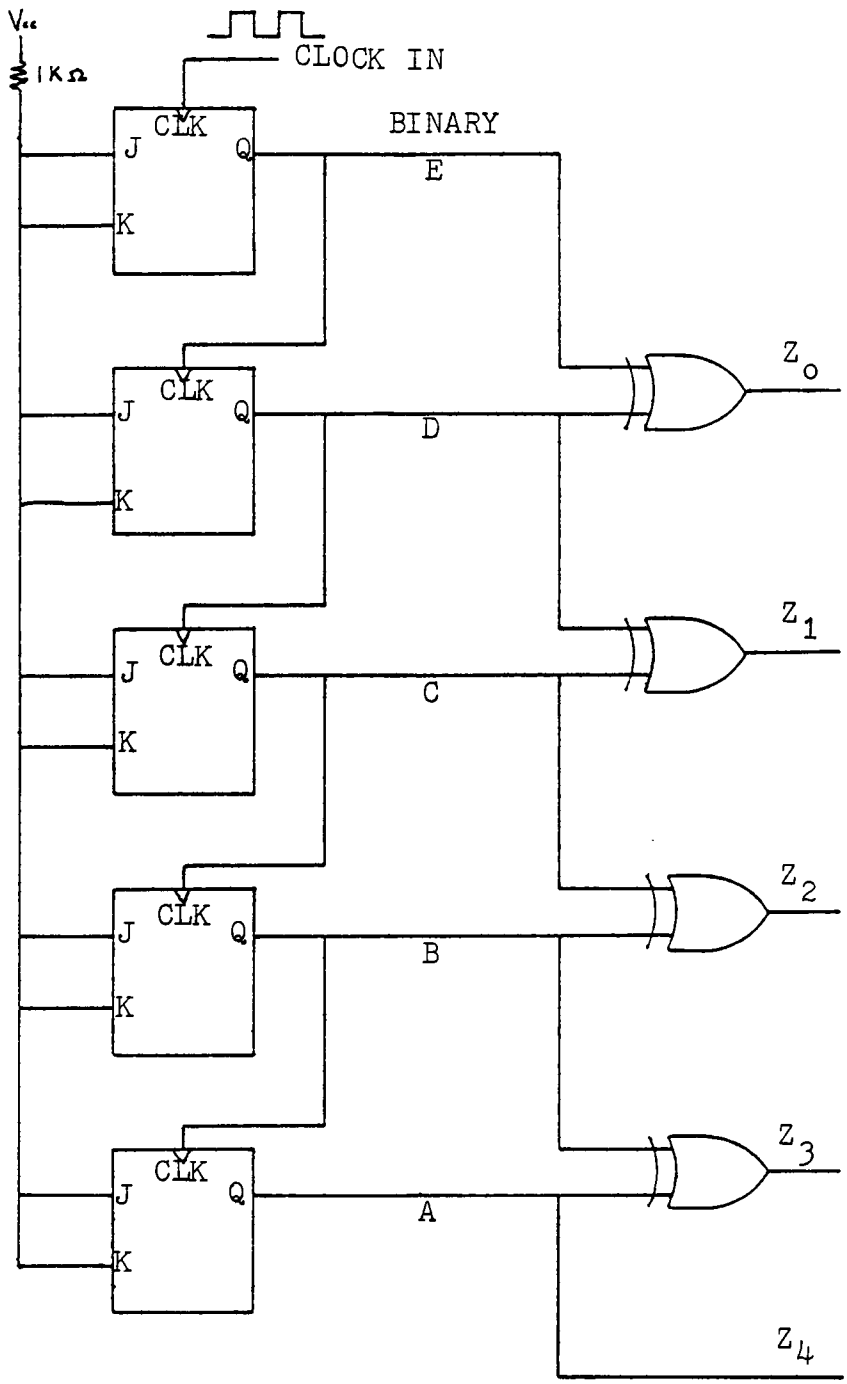


Figure 33. Five Bits Gray Code Generator.

binary to gray code is listed in Table XI. This converter was accomplished by using four Ex-OR gates. The purpose of using gray code as the input to the system was to prevent the occurence of any multiple input change. Three different cases were considered in the testing procedure. First, the truth table in Figure 30 was presented in Figure 34(a) where bit R was used to perform the reset capability. Second, a new system with five variables was established as shown in Figure 35(a). The part where R is one did not perform to initialize the circuit to the desired state. Third. the reset table (on the right hand side where R=1) is the same as its original table which is shown in Figure 36(a). As the reader should remember, the lower four bits of the addresses are the column selects of the memory matrix array, and other bits are the row selects of the array. Therefore, the most significant bit, Z_4, of the input sequence was connected to one row select and the other bits were input to the column selects.

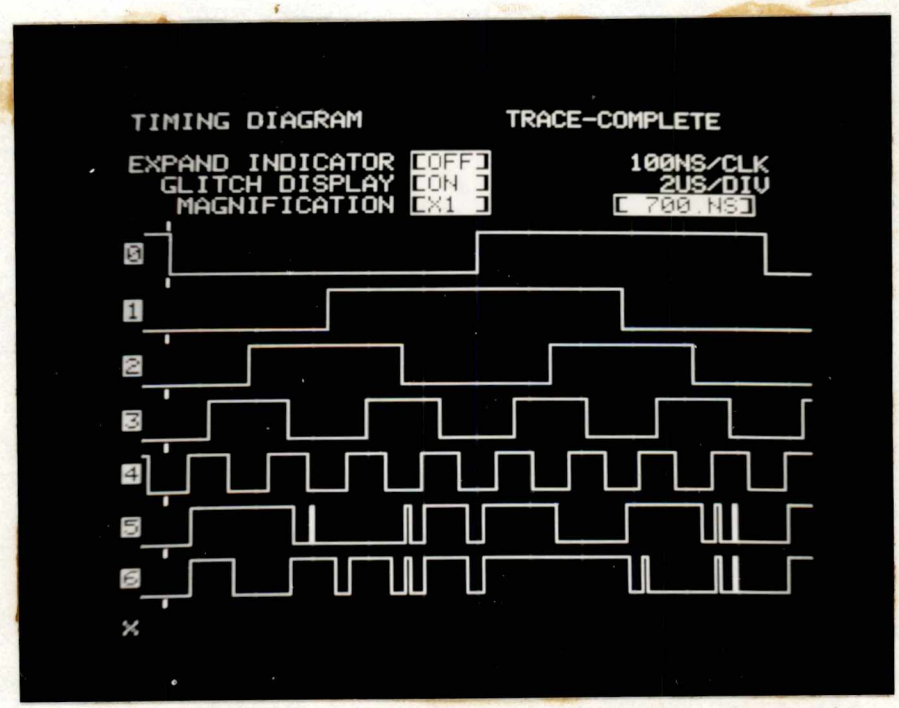
After the circuit was tested, the results shown in Figure 34(b), 35(b) and 36(b) were obtained. When ROM was used in combinational logic, it was not guaranteed to give a single output transition for a single input transition. In Figure 37, a short time (called hold time) after input change, outputs were undefined until a period equal to the ROM access time had elapsed. During this undefined interval, the ROM outputs might show noise or extra transitions.

Table XI. Binary to Gray Code Conversion Table (5 bits).

A	B	C	D	E	Z_4	Z_3	Z_2	Z_1	Z_o
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	1
0	0	0	1	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1	0
0	0	1	0	1	0	0	1	1	1
0	0	1	1	0	0	0	1	0	1
0	0	1	1	1	0	0	1	0	0
0	1	0	0	0	0	1	1	0	0
0	1	0	0	1	0	1	1	0	1
0	1	0	1	0	0	1	1	1	1
0	1	0	1	1	0	1	1	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	1	0	1	0	1	1
0	1	1	1	0	0	1	0	0	1
0	1	1	1	1	0	1	0	0	0
1	0	0	0	0	1	1	0	0	0
1	0	0	0	1	1	1	0	0	1
1	0	0	1	0	1	1	0	1	1
1	0	0	1	1	1	1	0	1	0
1	0	1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1
1	0	1	1	0	1	1	1	0	1
1	0	1	1	1	1	1	1	0	0
1	1	0	0	0	1	0	1	0	0
1	1	0	0	1	1	0	1	0	1
1	1	0	1	0	1	0	1	1	1
1	1	0	1	1	1	0	1	1	0
1	1	1	0	0	1	0	0	1	0
1	1	1	0	1	1	0	0	1	1
1	1	1	1	0	1	0	0	0	1
1	1	1	1	1	1	0	0	0	0

y_1y_2 \ x_1x_2	R=0 00	R=0 01	R=0 11	R=0 10	R=1 00	R=1 01	R=1 11	R=1 10
00	(00)	01	(00)	(00)	(00)	10	01	11
01	11	(01)	(01)	11	00	10	(01)	11
11	(11)	10	01	(11)	00	10	01	(11)
10	(10)	(10)	00	00	00	(10)	01	11

(a)

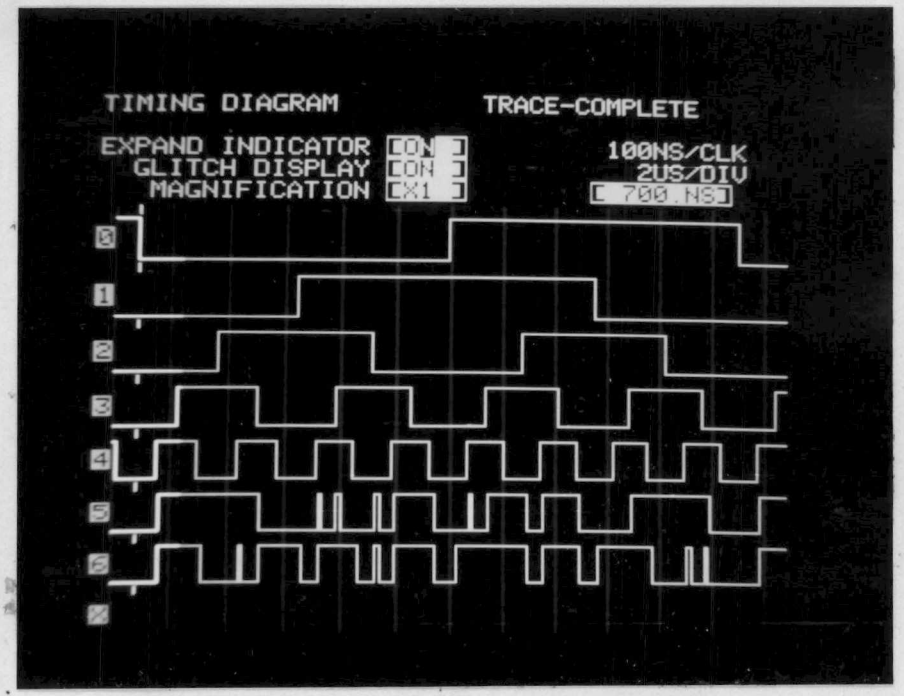


(b)

Figure 34. Test Table with Reset Capability.
(a) Excitation Table,
(b) Timing Diagram.

y_1y_2 \ x_1x_2	R=0 00	01	11	10	R=1 00	01	11	10
00	(00)	01	(00)	(00)	(00)	01	(00)	01
01	11	(01)	01	11	00	(01)	11	(01)
11	(11)	10	01	(11)	10	(11)	(11)	(11)
10	(10)	(10)	00	00	(10)	(10)	00	11

(a)

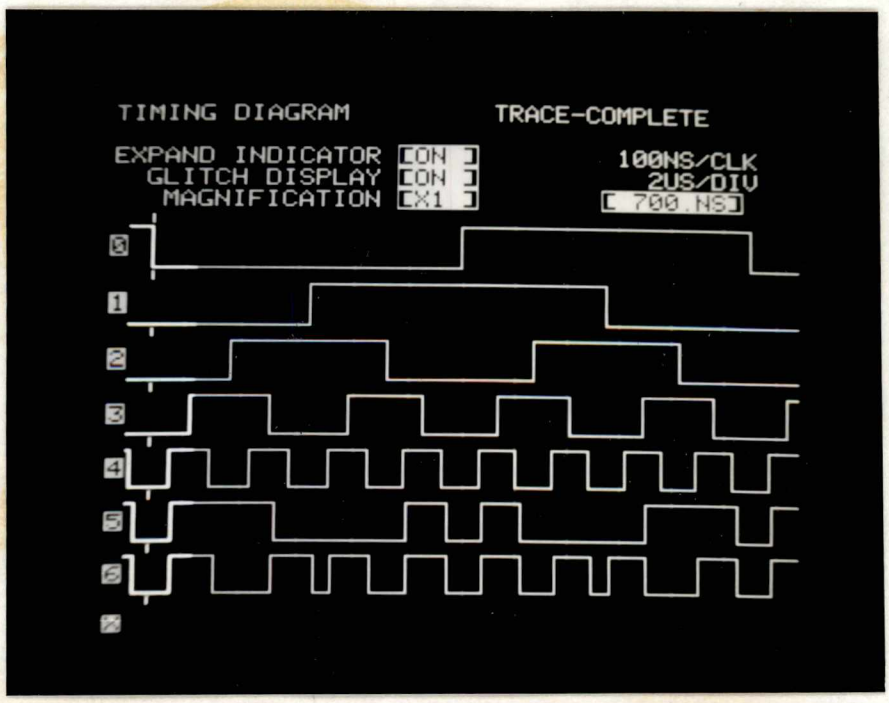


(b)

Figure 35. Test Table with a New System when R=1.
(a) Excitation Table,
(b) Timing Diagram.

y_1y_2 \ x_1x_2	R=0 00	R=0 01	R=0 11	R=0 10	R=1 00	R=1 01	R=1 11	R=1 10
00	(00)	01	(00)	(00)	(00)	01	(00)	(00)
01	11	(01)	(01)	11	11	(01)	(01)	11
11	(11)	10	01	(11)	(11)	10	01	(11)
10	(10)	(10)	00	00	(10)	(10)	00	00

(a)



(b)

Figure 36. Test Table with the Same Set of Table When R=1.
(a) Excitation Table,
(b) Timing Diagram.

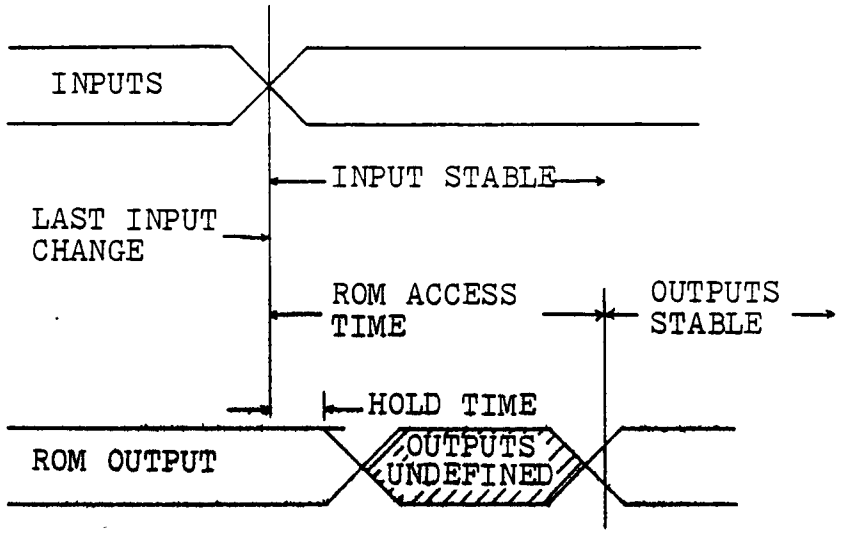


Figure 37. ROM Behavior for Combinational Logic.

These extra transitions would causes glitches at the output shown in Figures 34(b), 35(b) and 36(b). The width of these glitches is equal to the difference between access time and hold time of the memory.

If R was fixed to a certain logic level, for example, at logical 0, the output timing diagram in Figure 38 shows that glitch is not observed from the logic analyzer. The following passage gives an explaination of this phenomenon.

When R was fixed (Row address was stable), one row was selected, turning on the row line to all 128 cells in the row as shown in Figure 26 on page 64. The column address connects 8 of the 128 column lines to their respective sense amplifier. This row line provided bias to all the top gates in a particular selected row. Figure 39 shows a basic diagram in the PROM array[3]. Depending on the state of the cells, FAMOS devices would be charged for a programmed "0" labelled ⊖ or discharged, pulling the column lines down to low level for a programmed "1". If a "0" is stored in the cell, the FAMOS device in ON, and the level at the input of the sense circuit is high. These sense amplifier acts like an inverter which converts the signal to lower level. Thus, a logical 0 is at the output of the memory. A "1", which represents no charge, was stored in the cell and a low level was presented at the input of the sense circuit. Then, the output of memory would be at logical 1.

The stable signal of R fixed the logical level at the

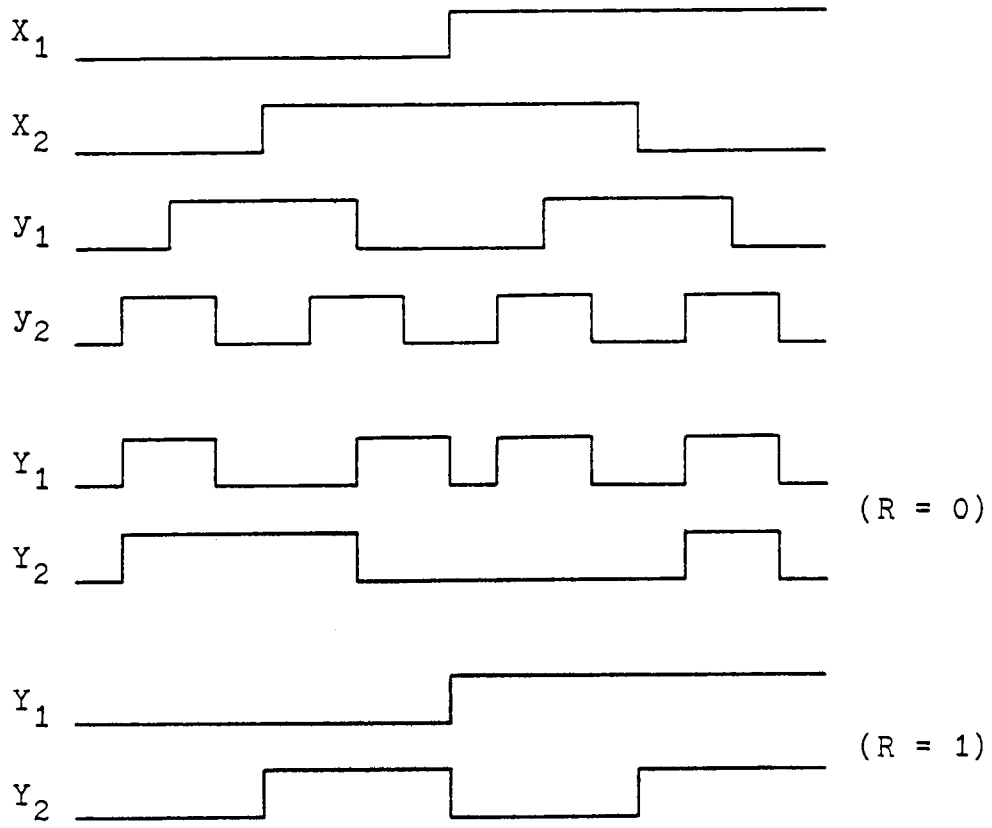


Figure 38. Timing Diagram When R is Fixed At Zero and One.

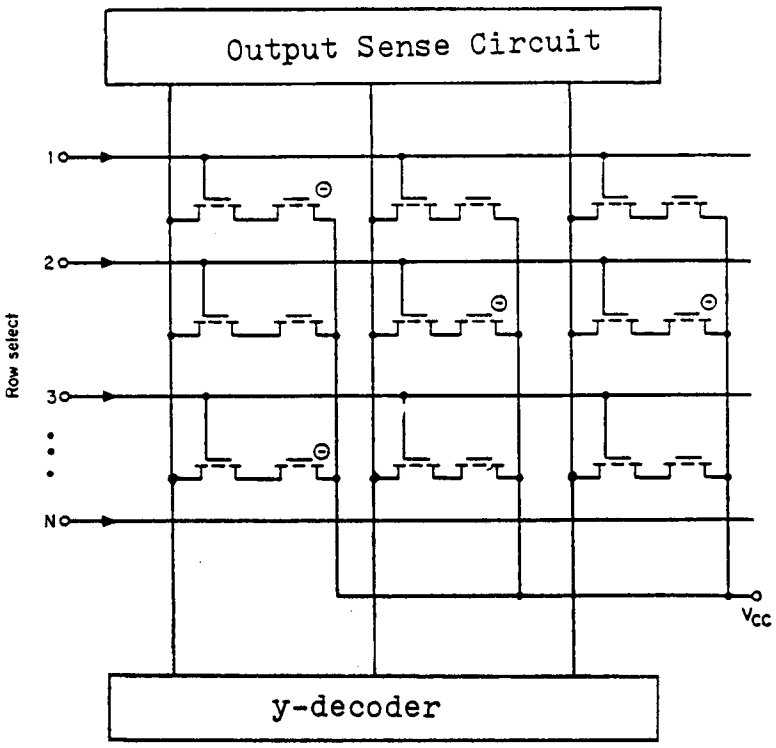


Figure 39. Charge-storage MOS for Electrical Programing the ROM: Matrix Circuit Schematic.

row lines. Thus, only the column lines were used to select the content of the memory. This meant that the changing of column lines would cause the decoder to select one of those sense circuits. The effect of changing column lines did not turn on and off the Mos transistors. Therefore, the switching time of changing column lines was too small to cause the unstable region at the ROM output and the circuit behaved correctly without any glitch.

Another interesting result is shown in Figures 34(b) and 36(b). The glitches appeared only when the output was at logical 0, and the corresponding column lines (under the same column line but with different row) produced logical 1 at the output. For example, a glitch occurs at $Z_4 = Z_3 = 0$ and $Z_2 = Z_1 = Z_o = 1$. At $Z_4 = 1$ and under the same column lines (i.e. at $Z_3 = 0$ and $Z_2 = Z_1 = Z_o = 1$) the output is at logical one. This result can be explained as follows.

Figure 40 shows a particular cell in the memory array with two different row selects but the same column lines. Suppose that the left FAMOS was charged (programmed "0") and the right one was uncharged (programmed "1"). If row A was first selected, the output of the sense circuit stayed at logical 0. When a change in row select from A to B took place (this is due to the unstable signal at row line), row A and B might temporarily selected. The level at the drain of the left MOS transistor became low and the drain of the right transistor became high. Since these two drains were connected in one line, the lower level would always

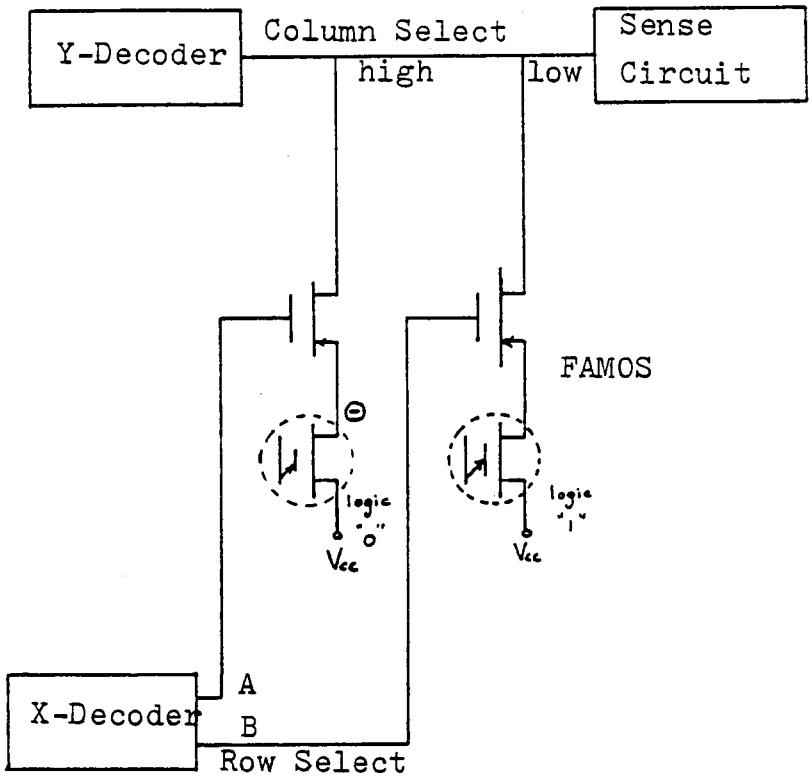


Figure 40. Internal Structure of Two Cell With Different Row Select and Same Column Select in a EPROM.

pull the high voltage to low. Thus, the output would temporarily go to one and a positive glitch was displayed.

If ROM was used to replace wired logic in the combinational part of the fundamental mode circuit, the momentarily change at the outputs would feed back to some inputs of the circuit. This resulted in an addition transition before the output became stable. The addition transition would cause the system to change to the incorrect output state and a hazard is said to be exist. This hazard is called a transit hazard in this thesis and is defined as follows:

Definition : Suppose there is a ROM in a combinational circuit and at least one row line and several column lines are input from any external signal. A change in a row line followed by a change in one column line causes an undefined signal at the output. This produces a glitch with a pulse width equal to the difference between ROM access time and hold time. Then a transit hazard exists in this transition.

This hazard permanently stayed in ROM and it could not be corrected. However, the designer could prevent the incorrect operation caused by this hazard when ROM was connected to a digital system.

Suppose that a ROM is used in a digital system. Signals from other parts of the system input to the address of the memory. An output is produced and fetched into a latch circuit until the output signal becomes stable. As a result,

the system would not be affected by a spurious pulse which caused a faulty operation. In realizing the level mode circuit, a delay element should be inserted between the next state and the present state of ROM. Since outputs of ROM are undefined for a maximum period of time equal to the ROM access time, the magnitude of the delay element should be longer than the access time. This provides a stable signal at the present state input of the level mode circuit. Thus, a transit hazard would not exist in the circuit.

CHAPTER V

DESIGN WITH PROGRAMMABLE LOGIC ARRAY

Programmable or customizable logic is another solution to the design of fundamental mode circuit impeding the development of LSI building block. The primary example is the programmable logic array. The programmable logic array, or PLA, is a universal array of logic gates that can be customized, normally by blowing a fuse, to a particular application.

Introduction

PLAs (and the newer additions to the array logic family, field programmable logic array, FPLAs) consist of a string of AND and OR gates custom configured to produce a specified output (typically eight pins), given by a specific input. A typical PLA is configured with fourteen input pins, permitting up to 65K input variations. Throughput can be as short as 14 ns. These inputs are run through an array of AND gates producing a predetermined series of product terms (typically 50-200). This is determined first by the identity of the input to the gates and second by the configuration of the gates. This output is run through a series of OR gates to produce the final output from the PLA. Buffers and inverters are also incorporated into many PLAs.

To use a PLA, the designer must think of the product terms that will be created as a result of running specific

input signals through the AND gates and the equations he will need to produce the specified terms

Using the equations necessary to generate the required product terms, the designer marks off the gates that are or are not to be activated to produce the desired output. The specification is then sent to a manufacturer in the form of a truth table, or on punched cards, and the data are fed to a computer that generates the mask needed to produce the PLA.

To get around the problems associated with the time and trouble involved in cutting the masks, FPLAs have been developed. These devices permit the individual programmable connection points to be addressed from the input and output pins and, once addressed, these points can be charged by blowing out fuses. However, all this comes at a price — to allow space for the internal addressing and programming logic, the devices offer users a limited number of product terms. For example, National's DM7575 series offer 96 product term gates; the FPLA equivalent has only 48 product terms. (Note that "field programmable" in this section does not mean erasable.)

PLAs are being used in computer design, microprogrammed microprocessors and single-chip calculator applications. In addition, one of the most straightforward applications of PLAs is in the address transformation in conventional control memories. They may also be used with registers directly in peripheral device controllers to implement the

flow chart of the state diagram. This reduces design time and permits fewer errors.

Other applications include code converters, large high speed ROMs, high speed character generators for large and unusual fronts, sequential controllers, logic function generator, and implementing decision tables.

In common with the ROM type of design, PLA-based have much greater design flesibility than random logic system. Instead of being restricted by available SSI and MSI functions, IC package count, power and fan-out limitations, and the rigidness of PC board layouts, the PLA designer can modify his system by changing the PLA.

Structure of PLA

A. Basic Structure. Although various PLAs differ slightly in organization, most of them have a structure similar to that of Figure 41. A Signetics' FPLA 82S100[27] has sixteen inputs, 48 product terms and 8 outputs. The logical structure of 82S100 is shown in Figure 42. Each AND is connected (through fuses) to each input and its complement; that is, each AND has 32 inputs. There are 48 AND gates in the circuit, meaning that the fuses can be blown to form 48 product terms. Each OR gate is connect to each AND output; that is, each OR gate has 48 inputs. These eight OR gates are used to produce eight outputs.

The internal structure of 82S100 is shown in Figure 43.

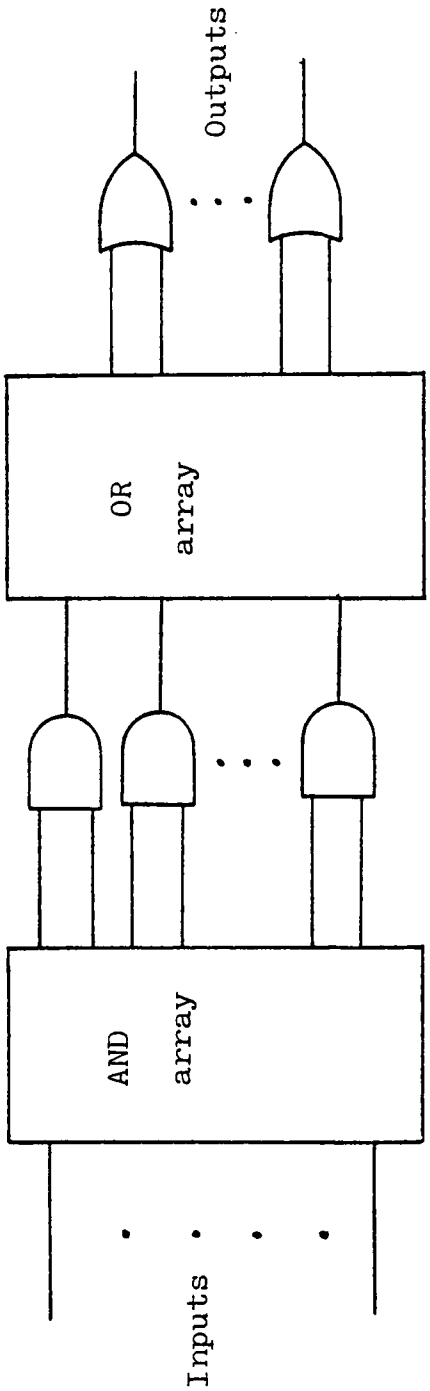
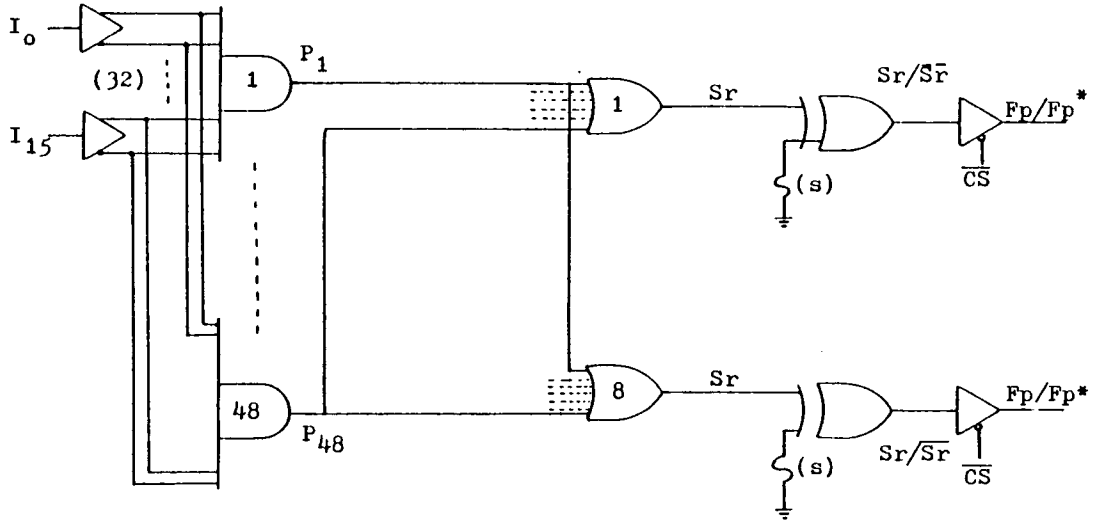


Figure 41. General Structure of a PLA.



$$Pn = \prod_{0}^{15} (KmIm + Jm\overline{Im}) \quad ; \; K = 0,1,x \text{ (don't care)} \qquad n = 0,1,2,\ldots,47$$

Where

Unprogrammed State: $Jm=Km=0; t_n=1$

Programmed State : $Jm=\overline{Km}; t_n=0$

$$Sr = P_o + P_1 + \cdots + P_n$$

$$\overline{Sr} = \overline{P}_o \cdot \overline{P}_1 \cdot \overline{P}_2 \cdot \cdots \cdot \overline{P}_n \quad @ \; S=open$$

$$Fp = \overline{CE} + Sr = \overline{CE} + (P_o + P_1 + P_2 + \ldots + P_n) \quad @ \; S=short$$

$$Fp^* = CE + \overline{Sr} = CE + (\overline{P}_o \cdot \overline{P}_1 \cdot \overline{P}_2 \cdot \ldots \cdot \overline{P}_n) \quad @ \; S=open$$

Figure 42. Logic Structure of FPLA.

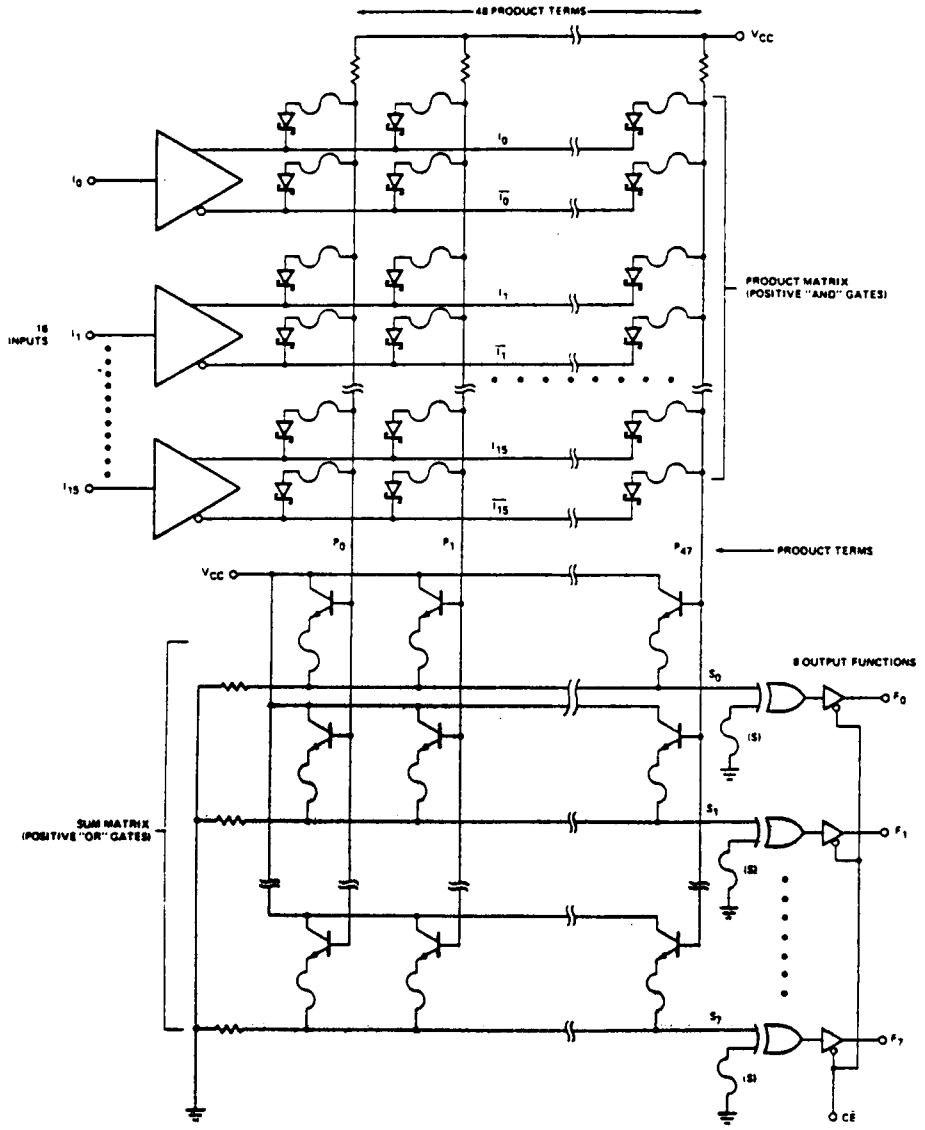


Figure 43. The Internal Structure of 82S100 FPLA. (16x48x8)

This device consists of an upper AND matrix containing 48 product term columns (P-terms), and a lower OR matrix containing 8 sum term rows (S-terms),one for each output function. Each P-term in the AND matrix is initially coupled to each input variable via two schottky diodes for programming the desired input state, and to each S-term in the OR matrix through an emitter follower with an emitter fuse. This emitter follower pulls the summing node to a high level when the P-term is activated. Each S-term in turn is coupled to its respective output via an Ex-OR gate which has programmable transmission polarity by means of an input to ground through a fusible link.

The initial unprogrammed state of 82S100 has all Ni-Cr links intact such that

1. Each P-term contains both true and complement values of every input I_m. Hence, all P-terms are in the "Null" state (always low).
2. Each S-term contains all 48 P-terms.
3. The polarity of each output is set to active HIGH. Since all P-terms are inactive, all outputs will be at a low level when the chip is enabled ($\overline{CE}$ = low), regardless of input condition.

B. <u>In Comparison to Read Only Memory</u>. The PLA is a well-organized logic structure, similar in many respects to the ROM; a set of inputs gives a related set of outputs. The number of variables in the input word does not affect the

number of variables in the output word.

In a ROM (Figure 44), all internal words are reached by a fixed decoder internal to the device. A controller with 10 input variables and one output would require 2^{10} (1024) bits of ROM storage, even if only 200 of these bits (20 words) were needed for control. The rest of the ROM is unused and would be programmed with "0s". Assuming no logic reduction could be made, the PLA design would require 20 rows to accomplish the same task. The remainder of the PLA would be available for other applications. A typical truth table stored in an 8x4 ROM is shown in Figure 45. If logic 1 is defined as the active-True state of all output functions; it is not possible to compress the truth table by eliminating inactive minterm 2, 4 and 7. Moreover, with regard to minterm 0 and 1, it is necessary to allocate two distinct storage locations to active output functions fs with a single change in input variable A_0. In this case, A_0 represents a logical Don't care (x) which cannot be directly programmed in a PROM. Instead, separate minterms $\overline{A}_2\overline{A}_1\overline{A}_0$ and $\overline{A}_2\overline{A}_1\overline{A}_0$ must be programmed.

In a PLA, storage for any unused minterm is no longer required. The necessary logic output for the inactive occurs by "default." When any programmed logic combination is present at the FPLA inputs, the corresponding address matrix column (P-term) will be pulled HIGH (logically active), forcing all outputs to their True logic state programmed in

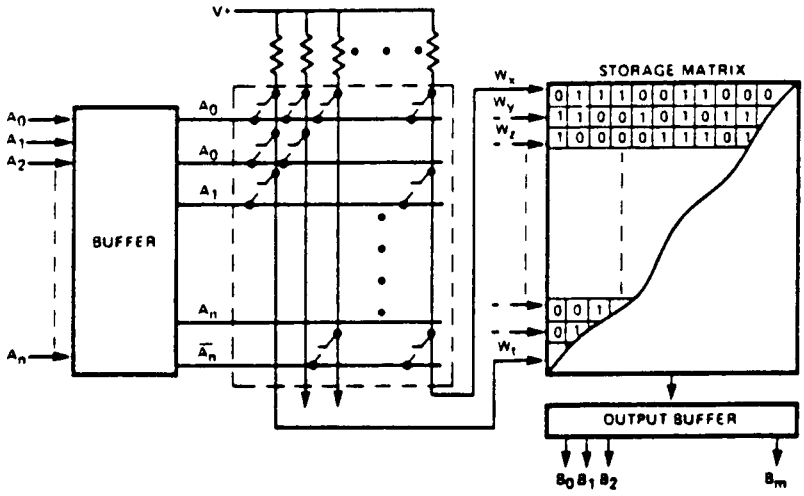


Figure 44. Simplified ROM Organization.

	ADDRESS			OUTPUTS				
M_n	A_2	A_1	A_0	O_1	O_2	O_3	O_4	
0	0	0	0	1	1	1	1	
1	0	0	1	0	0	1	1	
2	0	1	0	0	0	0	0	
3	0	1	1	0	1	1	1	
4	1	0	0	0	0	0	0	Inactive minterms
5	1	0	1	1	0	1	0	
6	1	1	0	1	1	0	0	
7	1	1	1	0	0	0	0	

Figure 45. Typical Truth Table Store in an 8x4 ROM.

the storage. Conversely, for all nonexistent logic combinations present at the FPLA inputs, all columns will remain low (logically inactive) forcing all output to the False logic state by default (the complementary logic state of their programmed active level polarity).

The address decoder of the ROM is exhaustive; that is, all possible addresses are decoded, and a word line is provided for each address. For example, a ROM with eight address lines has 2^8 or 256 words. For each additional address line, the number of words is doubled. This characteristic is important because it is related to PLA characteristics.

Like a ROM, the PLA has an address decoder and a data matrix (Figure 46); and like a ROM, the data matrix is programmed with MOS transistors at the various cross points. However, it is the differences in the address decoders that give the PLA its advantages. The address decoder is programmable in the PLA, as is the data matrix.

In ROM, each word line is selected by one and only one minterm. The number of word lines equals the number of possible address minterms. This number of word lines equals the number of possible address minterms. This exhaustive addressing should be contrasted with the nonexhaustive addressing of the PLA, where there is no inherent relationship between the number of minterms and the number of words.

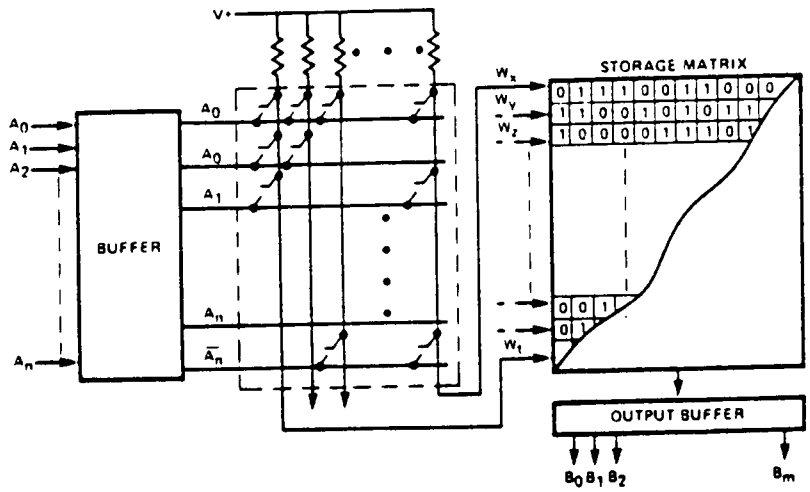


Figure 46. Typical FPLA Organization.

Because each word line may be selected on a general product term basis rather than on just a minterm basis, it is possible to select two or more words concurrently. concurrency occurs when two or more word lines have product terms that share at least one minterm (Figure 47). For example :

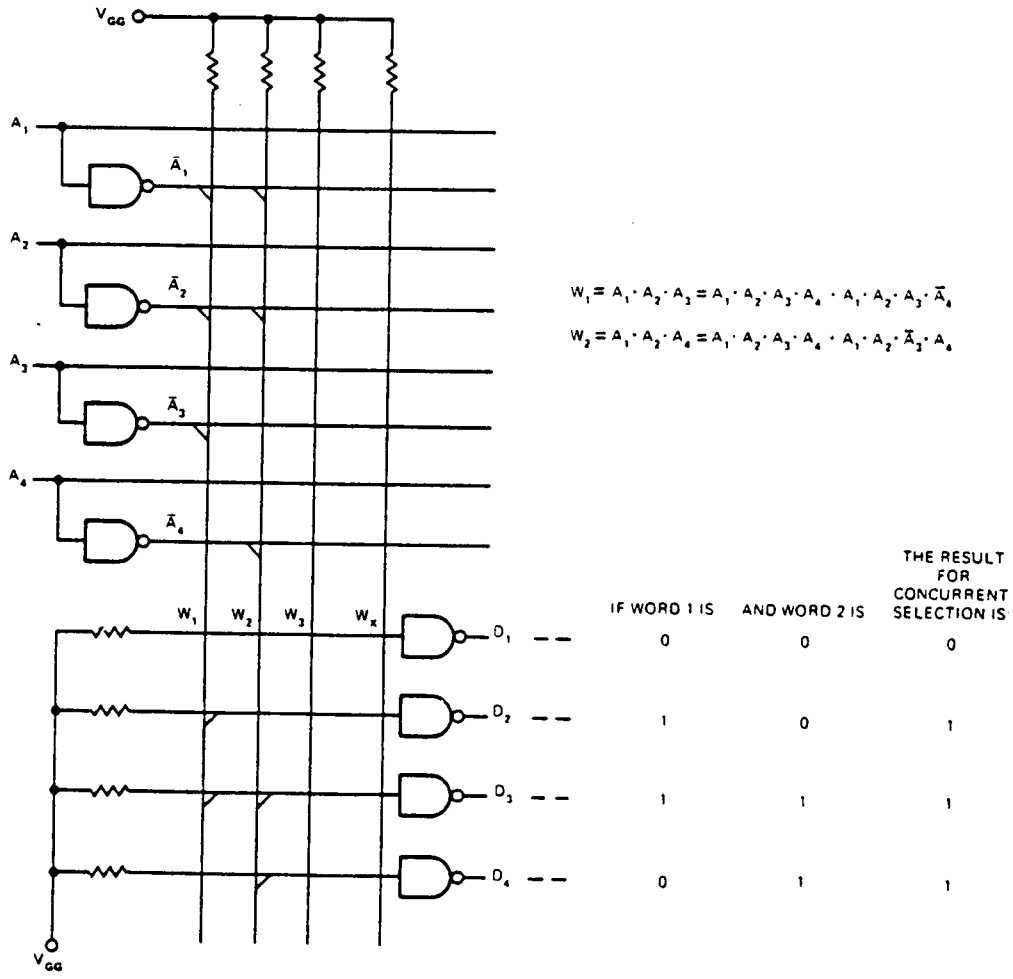
$$w_1 = A_1 A_2 A_3 = A_1 A_2 A_3 A_4 + A_1 A_2 A_3 \overline{A}_4 \qquad (V\text{-}1)$$

$$w_2 = A_1 A_2 A_4 = A_1 A_2 A_3 A_4 + A_1 A_2 \overline{A}_3 A_4 \qquad (V\text{-}2)$$

Both w_1 and w_2 have the $A_1 A_2 A_3 A_4$ minterm in common. It is possible for each word to be selected uniquely; however if an address is received with $A_1 - A_4$ in the "1" state, both words will be selected concurrently, with the result that the two words are logically ORed together in the data matrix :

	D_1	D_2	D_3	D_4
if word 1 is	0	1	1	0
and word 2 is	0	0	1	1
the result for con-current selection is	0	1	1	1

In general, when two or more words share minterms in their product term makeups, concurrcncy will occur when an address is received such that the shared minterm is true. Because of the mutual exclusivity of minterms in the ROM, concurrency cannot occur in the memory.



IF WORD 1 IS	AND WORD 2 IS	THE RESULT FOR CONCURRENT SELECTION IS
0	0	0
1	0	1
1	1	1
0	1	1

Figure 47. Programming the FPLA.

Since the addresses are programmed, the PLA can cope with certain special address conditions more easily than a ROM can. These special cases include unprogrammed addresses, single addresses for multiple words, and multiple addresses for single words.

An unprogrammed address input does not not read any word of the PLA; thus, all bits of the output, in general, remain at "0". Similarly, a single address can apply to two or more words in the PLA, producing the logical OR function of all the addressed words. For example, suppose two words with the contents 01010011 and 10100101 have the same address; the output when this address is received is 11110111, the logical OR of two words.

But when there are multiple addresses for single words, any of the different addresses produce the same output. This occurs when one or more bits of the address of a word have optional or "don't care" values instead of being specifically "1" or "0". Thus, a single word may be given a binary address of 00010xxx, where x represents an optional value. This word appears at the PLA output when any address in the binary range 000 1000 – 00010111 inclusive is sent to the PLA input. Optional values are supplied to the address of a word in the PLA simply by leaving both the true and complement input lines disconnected from the product term line in the address part of the PLA matrix.

This use of optional address bits is the basic of the PLA advantage over the ROM, which requires exhaustive decoding to achieve the same results. For the three special cases mentioned, the ROM must respond to every address. It must produce an all-0's-word for a meaningless address, corresponding to PLA's unprogrammed address. In addition, the ROM must also produce a word specifically programmed with the logical OR of two or more other words, corresponding to PLA's single address for multiple words; moreover, it must produce separate identical words for nearly identical addresses, which cannot contain optional bits in ROM. The PLA can implement several simple functions by using only part of the structure for each function; or complex functions can be implemented by connecting several PLAs in series or parallel.

Design Philosophy

When an PLA is used as a logic function generator, it is actually mechanized as an AND/OR logic structure where the 48-word addresses are decoded by 48 16-input AND gates. Each output can be defined as a sum of the logical product terms.

As a simple example, Figure 48(a) shows a small part of a PLA layout with six inputs, 13 product terms (words), and four outputs. This is an alternative notation of the logical structure of PLA in Figure 43 on page 95 (although the output line inverter has not been shown). The dots on

the regular matrix at the top can be thought of the unblown fuse of the AND inputs to the vertical product term line. These are either connected to the inverted, or uninverted input or not connected at all (for a "don't care") as specified by the customer. The small matrix at the bottom is the output OR matrix. Each dot can be thought of as an unblown fuse of OR input to generate one of the outputs.

To demonstrate how to order a PLA to mechanize logic functions, this PLA of Figure 48(a) is used to mechanize the following equations :

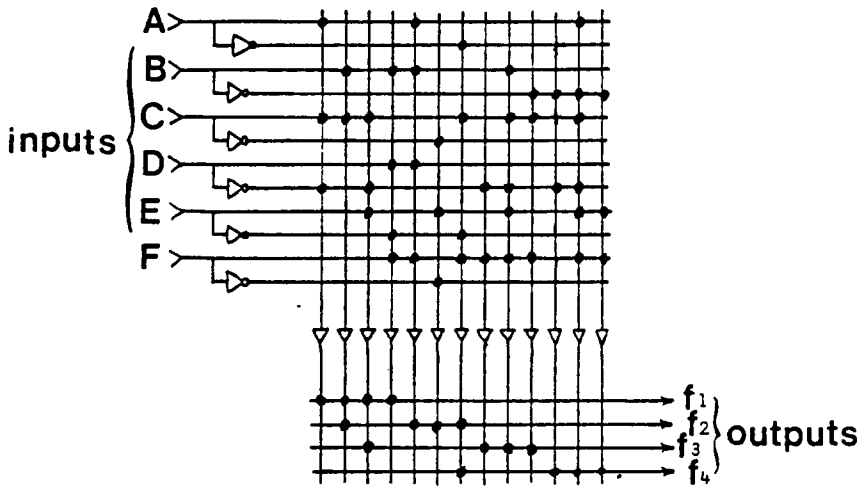
$$f_1 = AC\bar{D} + BC + C\bar{D}E + BD\bar{E}F \qquad (V\text{-}3)$$

$$f_2 = ABDF + BC + \bar{C}E\bar{F} + \bar{A}C\bar{E}F \qquad (V\text{-}4)$$

$$f_3 = \bar{D}F + BC\bar{D}EF + C\bar{D}E + \bar{B}CF \qquad (V\text{-}5)$$

$$f_4 = \bar{A}C\bar{E}F + A\bar{B}C\bar{D}EF + \bar{B}\bar{D} + \bar{B}EF \qquad (V\text{-}6)$$

Because the sum-of-product form of the equations fits the logic of the PLA exactly, one can simply enter the fun function directly on the truth table of Figure 48(b). Noting that the second term of f_2 and f_1 are both BC, one can use the same product term for both functions. Similarly, the third terms $C\bar{D}E$ of f_3 and f_1 are the same and the first term of f_4 $\bar{A}C\bar{E}F$ is the same as the fourth term of f_2; therefore, they can be combined as two different product terms. Then, it begins to mechanize the functions by entering each term in order on the truth table (Figure 48(b)). The first product term $AC\bar{D}$ is mechanized by writing 1x10xx on the first row of the table indicating that the first AND



(a)

Product Term No.	Inputs A	B	C	D	E	F	Outputs f_1	f_2	f_3	f_4
1	1	x	1	0	x	x	1	0	0	0
2	x	1	1	x	x	x	1	1	0	0
3	x	x	1	0	1	x	1	0	1	0
4	x	1	x	1	0	1	1	0	0	0
5	1	1	x	1	x	1	0	1	0	0
6	x	x	0	x	1	0	0	1	0	0
7	0	x	1	x	0	1	0	1	0	1
8	x	x	x	0	x	1	0	0	1	0
9	x	1	1	0	1	1	0	0	1	0
10	x	0	1	x	x	1	0	0	1	0
11	1	0	1	0	1	0	0	0	0	1
12	x	0	x	0	x	x	0	0	0	1
13	x	0	x	x	1	1	0	0	0	1

(b)

Figure 48. Realization of a PLA.
(a) Simplified PLA, (b) Truth Table.

gate should be connected to A, C and D and should have no connection to B, E and F. (x = don't care). A 1 in column f_1 indicates that the OR gate for output f_1 should be connected to this term. One simply continues this way until he has filled out a row for each term.

On the other hand, the table in Figure 48(b) can be easily code in an FPLA table and programmed in a device using inexpensive field equipment. The FPLA table in Table Table XII and the codes filled in the table refer to the Signetics' 82S100/101. In the product term in Table XII, there have 16 inputs (0 through 15). The H, L and - (dash) in the table represent the 1, 0 and x in Figure 48(b). If the polarity of an output function is set to active-HIGH, then an N is put in the ACTIVE LEVEL of the Table XII. Conversely, an L is put in the ACTIVE LEVEL if the polarity of a function is set to active low. In the OUTPUT FUNCTION of the table, the code Ⓐ (for ACTIVE) is used to indicate the presence of P_n (same as a 1 in f of Figure 48(b)) and a ⊙ (period) is used to indicate absence (same as a 0 in f of Figure 48(b)). Table XII shows a FPLA program table which is coding from Figure 48(b). This table is ready to use for fusing the PLA.

When more than one PLA is used , more complex logic function can be mechanized. For example, Figure 49 shows two 16-input PLAs used to generate 14 function of 20 variables. Examination of the product terms shows that none contains

TABLE XII

SIGNETICS FPLA PROGRAM TABLE

16X48X8 FPLA PROGRAM TABLE

PROGRAM TABLE ENTRIES						
INPUT VARIABLE			OUTPUT FUNCTION		OUTPUT ACTIVE LEVEL	
Im	$\overline{Im}$	Don't Care	Prod. Term Present in FP	Prod. Term Not Present in FP	Active High	Active Low
H	L	— (dash)	A	• (period)	H	L
NOTE Enter (—) for unused inputs of used P-terms.			NOTES 1. Entries independent of output polarity. 2. Enter (A) for unused outputs of used P-terms.		NOTES 1. Polarity programmed once only. 2. Enter (H) for all unused outputs	

PRODUCT TERM[1]																	
NO.	INPUT VARIABLE[1]																
	15	14	13	12	11	10	9	8	7	6	A 5	B 4	C 3	D 2	E 1	F 0
0	–	–	–	–	–	–	–	–	–	–	H	–	H	L	–	–
1	–	–	–	–	–	–	–	–	–	–	–	H	H	–	–	–
2	–	–	–	–	–	–	–	–	–	–	–	–	H	L	H	–
3	–	–	–	–	–	–	–	–	–	–	–	H	–	H	L	H
4	–	–	–	–	–	–	–	–	–	–	H	H	–	H	–	H
5	–	–	–	–	–	–	–	–	–	–	–	–	L	–	H	L
6	–	–	–	–	–	–	–	–	–	–	L	–	H	–	L	H
7	–	–	–	–	–	–	–	–	–	–	–	–	–	L	–	H
8	–	–	–	–	–	–	–	–	–	–	–	H	H	L	H	H
9	–	–	–	–	–	–	–	–	–	–	–	L	H	–	–	H
10	–	–	–	–	–	–	–	–	–	–	H	L	H	L	H	L
11	–	–	–	–	–	–	–	–	–	–	–	L	–	L	–	–
12	–	–	–	–	–	–	–	–	–	–	–	L	–	–	H	H

ACTIVE LEVEL[1]								
				H	H	H	H	
OUTPUT FUNCTION[1]								
7	6	5	4	f_1	f_2	f_3	f_4	
				A	•	•	•	
				A	A	•	•	
				A	•	A	•	
				A	•	•	•	
				•	A	•	•	
				•	A	•	•	
				•	•	A	•	
				•	•	A	•	
				•	•	•	A	
				•	•	•	A	
				•	•	•	A	

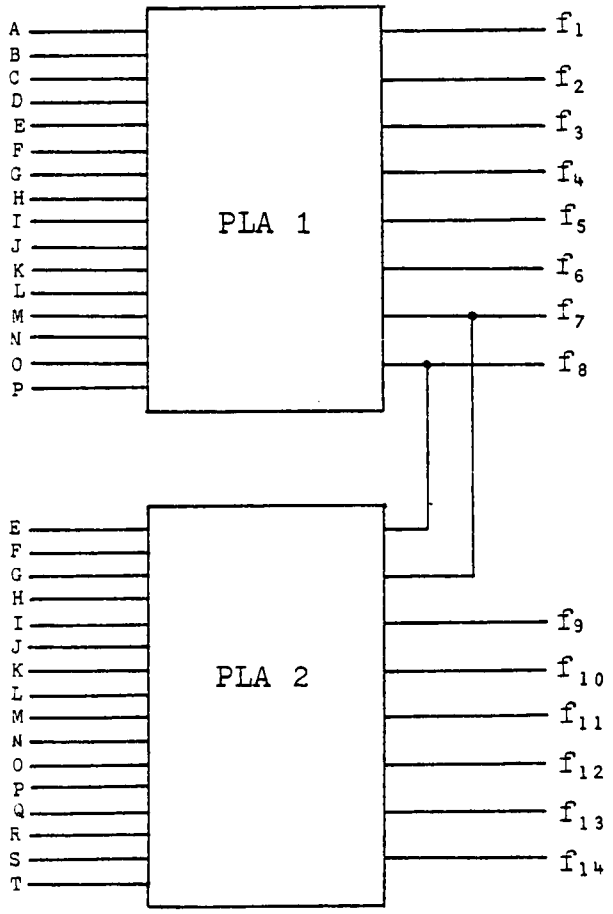


Figure 49. Multiple PLAs.

all variables. In fact, inputs A, B, C, Q, R, S and T occur in only a few places, and A, B, C, or D never occur in the same product term with Q, R, S, or T. Therefore, one can generate only product terms that do not contain Q, R, S, or T in PLA 1 and terms that do not contain A, B, C, or D in P PLA 2. Since functions f_7 and f_8 contain terms from both groups, they can connect together forming a Virtual OR. When many PLAs are used, it is useful to put functions that share the same product terms in the same package, even when the inputs to all the PLAs are the same. Again, "Virtual ORing" can be used for some functions to make it possible to use product terms from more than one PLA.

Hazard Consideration

The structure of PLA is actually an AND-OR logic structure. When it is utilized to generate a logic function, it is just the same as a random logic circuit. As a result, hazards contained in the PLA are more likely to occur than the hazards in the random logic circuit. A static hazard may exist in the PLA if there is an unequal delay time between any two product terms. The extra grouping of two adjacent 1s in different 1-sets would eliminate the static hazard. The method of investigating hazards in PLA will be the same as the one used in studying the random logic circuit; therefore, no further research will be carried out.

CHAPTER VI

POWER RESET CONSIDERATION

The power reset problem is very important in a fundamental mode circuit. Since the fundamental mode is a random access system, the user cannot be certain that the circuit will initialize from the expected starting state. As a result, this circuit may begin from a wrong state and cause the circuit to malfunction.

Basically, there are two methods to solve the power reset problem. In the first method, diagrammed in Figure 50, suppose that A, B, C and D are any four states in a state table. The user could design a circuit which always forced the system to start from a particular state, for example at state C in Figure 50. A binary code 000 (For a state table with a maximum of eight states) is decoded to this state. In this case, when the power is turned on, the system would always start from 000 (i.e., state C) and then transit to other states according to the input state changes.

This method could be done by the circuit shown in Figure 51. An RC circuit was connected to the power source followed by a schmitt trigger circuit[23]. When the power was on, an exponential pulse generated at the RC circuit. Then, the schmitt trigger circuit (Appendix C) [13] produced a positive rectangular pulse. This pulse was sent to an open collector circuit which is connected to the feedback path

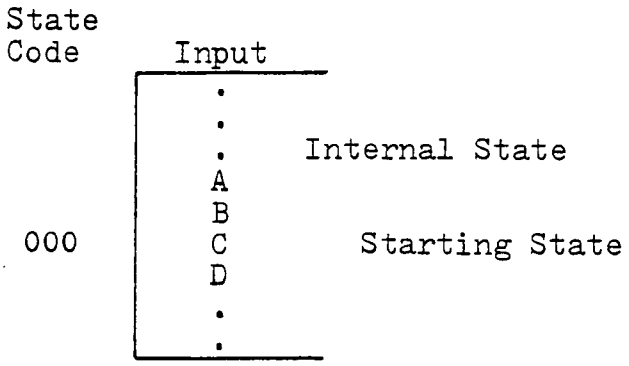


Figure 50. State Table of the First Method.

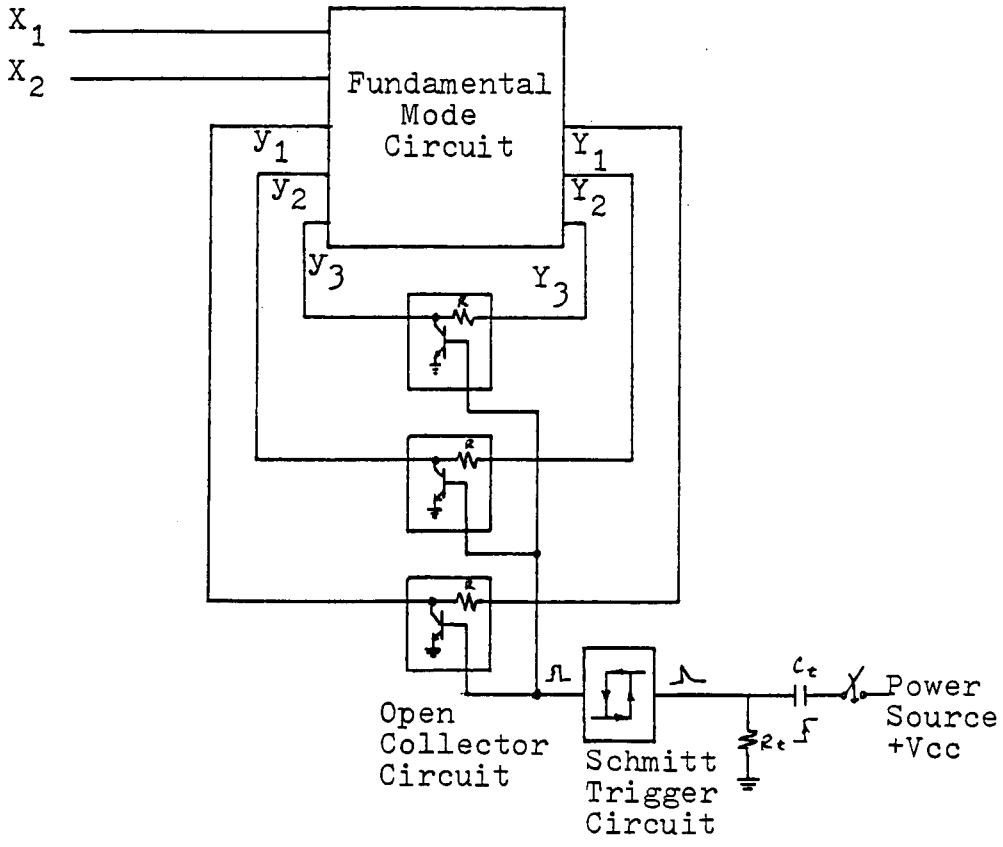


Figure 51. Circuit Diagram of the First Method.

of the fundamental mode circuit. (This open collector circuit gave a low level output when there was a high level at the input). When the level at the output of the schmitt trigger circuit changed to one, a low level output was presented at the open collector. This forced the feedback path to the input of the fundamental mode circuit to zero. Thus, the system is being initialized at state 000.

Secondly, a reset table is formed together with the original state table. Figure 52 illustrated the use of the reset table. In Figure 52, bit R is used to represent the reset pulse. When R=1, the system would jump to a particular initial state depending on the input. For example, in Figure 52, as the power is turned on, and the input state is at 00, bit R changes to 1. The system is forced to stay at state D. If the input state remains unchanged and bit R switches to zero, then the system will start from state D. On the other hand, if the input state changes to 01 when R stays at one, the system will move to state B. As R switches to zero, this system resumes at state B and transits according to the input states.

The advantage of the second method is that the system could initialize from any state depending on the input states. This is more flexible than the first method. In addition, this method could be done on the map instead of building an external circuit to control the system. (The external circuit here means the open collector circuit at

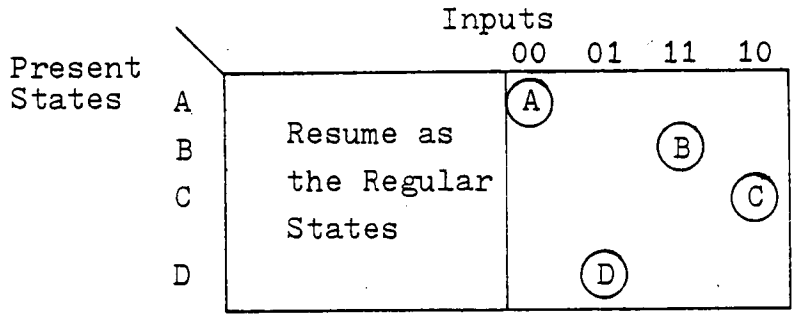


Figure 52. The State Table With Reset Capability.

the feedback path in Figure 51.)

Basically, the second method was applied in the experiment of investigating the hazards. A block diagram shown in Figure 53 illustrates the interconnection between the power reset circuit, the input sequence generator and the fundamental mode circuit. When the power was being turned on, a positive exponential pulse was generated at point A in Figure 53. The schmitt trigger circuit changed this exponential pulse to a rectangular pulse, and an inverter followed which gave a negative pulse. This negative pulse was sent to the CLR of the input generator circuit which would reset the circuit to zero. A positive pulse at point B was delivered to the R input of the fundamental mode circuit. As R switched to one, the circuit moved to the reset table which would be initialized at the starting state. (The duration of the pulse generated by the RC circuit should be long enough for the fundamental mode sequential circuit to reach a stable state.) Now, inputs of the fundamental mode circuit were at zero, and the system was at state 00. When R went to zero, the system resumed its original table and the output state transited according to the input state. Therefore, the fundamental mode circuit would always start from the expected initial state, and the circuit would respond correctly.

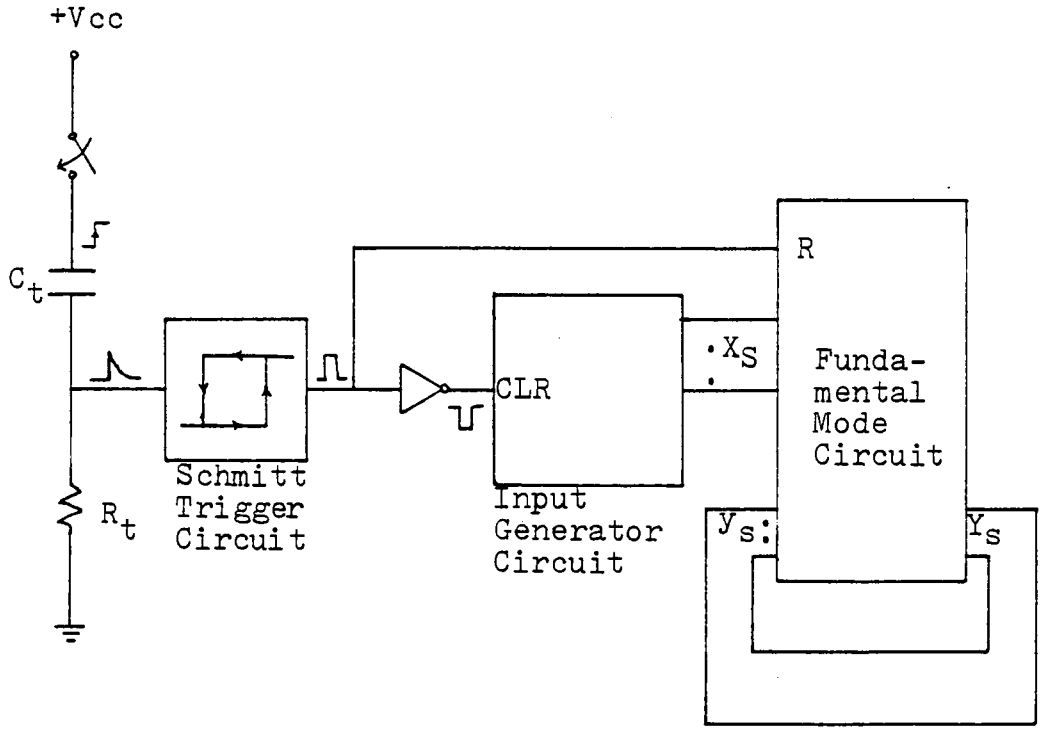


Figure 53. A Block Diagram of the Interconnection of Power Reset Circuit, Input Generator Circuit and the Fundamental Mode Circuit.

CHAPTER VII

CONCLUSION

In general, the essential hazard in random logic circuits detected by Unger's method does not always exist. One should employ a K-Map to carry on the further investigation of the essential hazard. The increase in temperature of one particular gate extended the delay time of that gate. Thus, this extension of delay time would sometimes cause the circuit to malfunction.

The hazards in the multiplexer would exist only when two or more multiplexers have been used. The delay time in one particular gate in a multiplexer could be much longer than in other gates in other multiplexer. Because of this significant difference in delay time, hazards may also exist.

When read only memory is used as the combinational circuit (column selects and row selects of the memory are used), glitches are produced at the output. These glitches would cause the circuit to malfunction when the memory is used in a fundamental mode circuit. This type of hazard is called a transit hazard and can be eliminated by two methods:

1. Adding a delay element in the feedback path.
2. Using a latch circuit at the output of the circuit.

Since the logical organization of PLA is just an AND-OR logic circuit, the hazards contained in PLA are very similar to the hazards in any two-level random logic circuit.

The power reset method has been applied to the experiment. This method always initiates the circuit to the starting state when the power is being turned on, thus providing a correct operation for power set up.

LIST OF REFERENCES

LIST OF REFERENCES

1. Blakeslee, T. R., Digital Design with Standard MSI and LSI, Wiley, New York, 1975.

2. Caldwell, S. H., Switching Circuits and Logical Design, Wiley, New York, 1958.

3. Carr, W.N. and Mize, J.P., MOS/LSI Design and Application, McGraw-Hill Book Company, New York, 1972.

4. Eichelberger, E. B., "Hazard Detection in Combinational and Sequential Switching Circuits," IBM Journal, vol. 9 vol. 9-10, pp. 90-99, March 1965.

5. Friedman, A.D., Graham, R.L. and Ullman, J.D., " "Universal Single Transition Time Asynchronous State Assignments," IEEE Trans. Comput., vol. C-18, pp. 541-548, June 1969.

6. Friedman, A.D. and Menon, P.R., Theory and Design of Switching Circuits, Computer Science Press, California, 1975.

7. Friedman, A.D. and Menon, P.R.,"Synthesis of Asynchrono Asynchronous Sequential Circuit with Multiple-Input Changes," IEEE Trans. Comput. vol. C-17, pp. 559-566, June 1968

8. Hill, F.J. and Peterson, G.R., Introduction to Switching Theory and Logical Design, Wiley, New York, 1977.

9. Hnatek, E.R., A User's Handbook of Semiconductor Memories, Wiley, New York, 1977.

10 Huffman, D. A., "The Synthesis of Sequential Switching Circuits," J. Franklin Inst., vol. 257, pp. 151-190, 275-303, March and April 1954.

11. Huffman, D. A., "The Design and Use of Hazard-free Switching Network," J. ACM, vol 4, p.47, 1957.

12. Intel Staff, Intel Memory Handbook, Intel Corporation, California, 1978.

13. Jacob, M. and Taub, H., Pulse, Digital and Switching Waveforms, McGraw-Hill, New York, 1972.

14. Kramme, F., "Standard ROM Simplify Complex Logical Design," Electronics, pp. 88-95, January 5, 1970.

15. Lerner, S. T., "Hazard Correction in Asynchronous Sequential Circuits," IEEE Trans. Electr. Comput., vol. EC-14, pp. 265-267, April 1965.

16. Liu, C. N., "A State Variable Assignment Method for Asynchrocous Sequential Switching Circuits," J. ACM, vol. 10, pp. 209-216, 1963.

17. Meisel, W.S. and Kashef, R.S., "Hazard in Asynchronous Sequential Circuit," IEEE Trans. Comput. vol. C-18, pp. 753-759, August 1969.

18. MOS Technology Staff, MCS6500 Microcomputer Family Programming Manual, MOS Technology Inc., Pennsylvania, 1975.

19. MOS Technology Staff, KIM-1 User Manual, MOS Technology Inc., Pennsylvania, 1975.

20. Nagle, H.T. Jr., Carrol, B.D. and Irwin, J.D., An Introduction to Computer Logic, Prentice-Hall Inc., New Jersey, 1975.

21. Nanya, T. and Tohma, Y., "On Universal Single Transition Time Asynchronous State Assignments," IEEE Trans. Comput., vol. C-27, pp. 781-782, August 1978.

22. Penny, W.M. and Lau, L., MOS Integrated Circuits, Van Nostrand Reinhold Company, 1972.

23. Shanghai Amateurish Industrial University Staff, Transistor Switching Circuits, Science Academic Press, Beijing, 1972.

24. Sholl, H. A., "Direct Transition Memory and Its Application in Computer Design," IEEE Trans. Comput. vol. C-23, pp. 1048-1061, October 1974.

25. Sholl, H.A. and Yang, S.C., "Design of Asynchronous Sequential Networks Using Read-Only Memories," IEEE Trans. Comput., vol. C-24, pp. 195-206, February 1975.

26. Signetics Staff, Signetics Field Programmable Logic Arrays, Signetics Corporation, California, 1976.

27. Texas Instruments Staff, Designing with TTL Integrated Circuits, McGraw-Hill, New York, 1971.

28. Texas Instruments Staff, The MOS Memory Data Book for Design Engineers, Texas Instruments Inc., Texas 1980.

29. Texas Instruments Staff, The TTL Data Book for Design Engineers, Texas Instruments Inc., Texas 1973.

30. Tracey, J. N., "Internal State Assignment for Asynchronous Sequential Machines," IEEE Trans. Electr. Comput., vol. EC-15, pp. 551-560, August 1966.

31. Uimari, D., "PROM's — a Alternative to Random Logic," Electronic Products Magazine, pp. 75-81, January 21 1974.

32. Unger, S. H., Asynchronous Sequential Switching Circuits, Wiley, New York, 1969.

33. Unger, S. H., "Hazard and Delay in Asynchronous Sequential Switching Circuits," IRE Trans. Circuit Theory, vol. CT-5, pp. 12-25, March 1959.

34. Unger, S. H., A Study of Asynchronous Logical Feedback Network, M.I.T. Res. Lab. Electron. Tech. Rept. 320, April 26, 1957.

35. Yoeli, M. and Rinon, S., "Application of Ternary Algebra to the Study of Static Hazards," J. ACM, vol. 11, p. 84, 1964.

APPENDICES

APPENDIX A

A PROGRAM FOR GENERATING A INPUT SEQUENCE

```
Address  Op Code      Mnemonic          Comments
0080     A9 0F     MAIN:LDA  #$0F      Initialize port A
0082     8D 01 17       STA  PADD      as output.
0085     A2 07          LDX  #$0F      Set # of input in seq.
0087     A4 20          LDY  GEN       Store the pointer
0089     B1 20    START:LDA  GEN       Load in Data by ptr.
008B     8D 00 17       STA  PAD       Output thru port A
008E     E6 20          INC  GEN       Move ptr to next data
0090     E4 20          CPX  GEN       End of sequence?
0092     D0 F5          BNE  START     If not, then continue
0094     84 20          STY  GEN       If yes, restore the ptr
0096     4C 89 00       JMP  START     Continue for next input

0000     00       TABLE .BYTE $00      Table for Input
0001     01             .BYTE $01      Sequence
0002     03             .BYTE $03
0003     02             .BYTE $02
0004     00             .BYTE $00
0005     01             .BYTE $01
0006     03             .BYTE $03
0007     02             .BYTE $02
0020     00        GEN  .BYTE $00      Pointer
```

APPENDIX B

INPUT SEQUENCE GENERATOR CIRCUIT

A block diagram of an input sequence generator circuit is shown in Figure 54. A high frequency clock pulse was sent in to a Divide-by-12 up counter. A block labelled CONTROL produces a clock pulse to activate the binary counter which was constracted by three J-K flip-flops. Then, a binary sequence was input to a control logic circuit and provided the required input sequence. A latch block was used to insure that the input sequence would deliver to the fundamental mode circuit at the same time. Table XIII AND XIV show the truth table used to generate the functions E, F, Z_1 and Z_2.

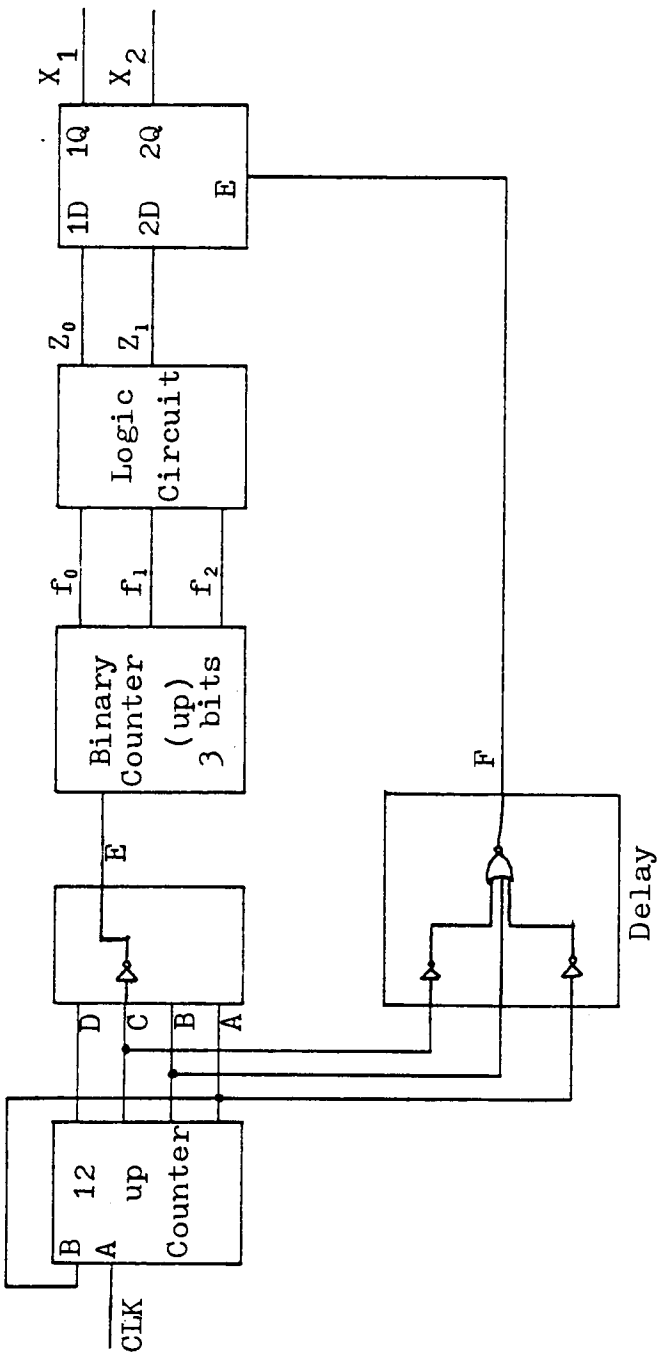


Figure 54. A Block Diagram of Input Sequence Generator Circuit.

Table XIII. Table for Generating Function E and F.

D	C	B	A	E	F
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1

$E = \overline{C}$

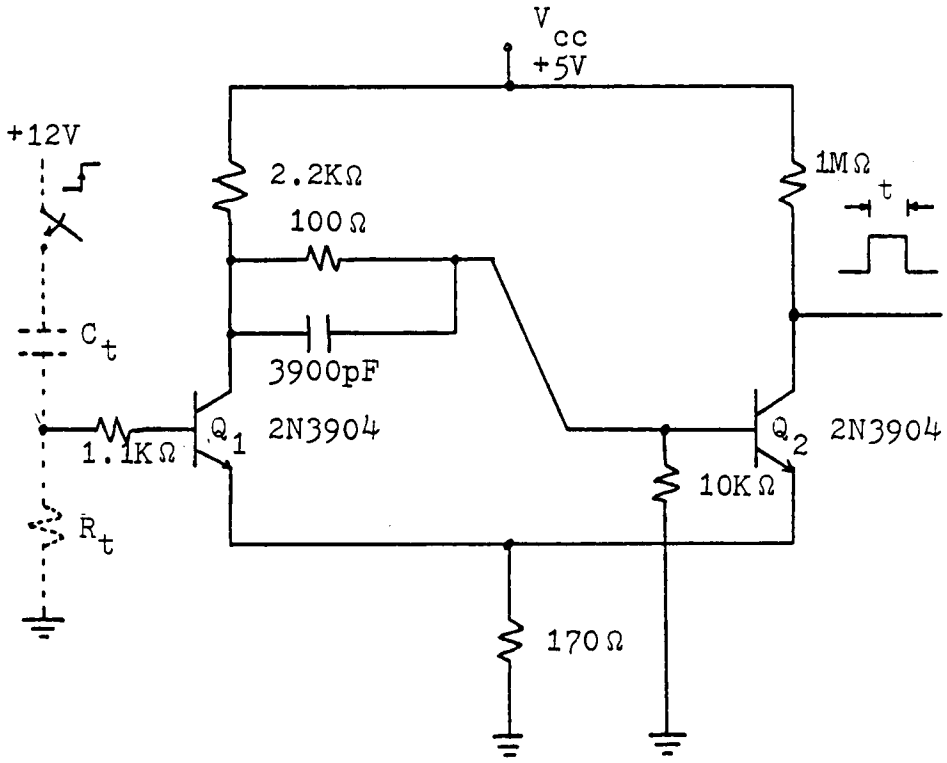
$F = \overline{A} + B + \overline{C}$

Table XIV. Table for Generating Function Z_0 and Z_1 from Binary Counter.

f_2	f_1	f_o	Z_1	Z_o
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

$Z_1 = f_1 \qquad Z_2 = f_1 + f_o$

APPENDIX C



$$t = 0.69R_tC_t$$

Figure 55. Schmitt Trigger Circuit.

VITA

Lu, Wen Pai was born in Taiwan, China, on October 10, 1957. Following graduation from The Chinese High School in Singapore, he attended University of Tennessee. In 1978, he received his Bachelor of Science with a major in Electrical Engineering.

He entered the Graduate School at the University of Tennessee in August 1978 and studied for his master degree in the Department of Electrical Engineering.

The author is a member of Institute of Electrical and Electronic Engineers and the Computer Society of that association.