DESIGN OF ASYNCHRONOUS SEQUENTIAL CIRCUITS WITH ASYNCHRONOUS UNIT DELAYS

by

Shanker Singh, M.Sc., M.Sc. (Tech).

Submitted in the partial fulfillment of
the requirement for the degree of

Doctor of Philosophy

Department of Electrical Engineering,
Faculty of Pure and Applied Science,
University of Ottawa,
Ottawa, Ontario.

November, 1968

Approved for the Department
of Electrical Engineering

Supervisor

Chairman of the Examining

Chairman of the Department

ABSTRACT

An asynchronous unit delay is an n-input n-output sequential circuit in which the present value of the output n-tuple is equal to the value of the input n-tuple prior to the last input change. This thesis considers the problem of realization of the asynchronous unit delay and its use in the design of asynchronous circuits. Asynchronous fundamental mode flow tables have been classified as "asynchronous definite" and "asynchronous indefinite", where asynchronous definite and asynchronous indefinite properties are modifications of the definite and indefinite properties of synchronous sequential machines respectively. It is shown that a fundamental mode table is realizable as a feedback-free connection of asynchronous unit delays if and only if the table is asynchronous definite. Straight-forward methods of realizing asynchronous definite and asynchronous indefinite flow tables without critical races by circuits of asynchronous unit delays and combinational gates are developed. The use of asynchronous unit delays avoids complicated secondary assignment problems, results in circuits with very simple structure and brings closer the theories of synchronous and asynchronous sequential machines.

ACKNOWLEDGEMENTS

The author wishes to express his deep and sincere sense of gratitude to Dr. Janusz A. Brzozowski for introducing the author to asynchronous sequential circuits, providing stimulating criticisms and discussions throughout the course of this work. His assistance is most gratefully acknowledged in writing, and providing ideas for, the joint paper [6a] the results of which appear in Chapter 4.

Special thanks are due to Professor S.G.S. Shiva for his assistance in proving certain results on codes which appear in Chapter 3.

Gratitude is expressed also to Professor G.S. Glinski, Dr. C.L. Sheng, Dr. Wayne Davis and Dr. J. Kruus for their willingness to listen and offer suggestions. Sincere thanks are also due to Professor G.S. Glinski for the financial assistance through a lectureship in the Department of Electrical Engineering, University of Ottawa.

Thanks are also due to my colleagues Peter Hawtrey, V. Batra and I. Chen for their comments on the work.

Lastly, Mrs. Thibert's patience and understanding while typing the manuscript is appreciated.

The financial support of the National Research Council of Canada under Grant No. A-1617 is gratefully acknowledged.

CONTENTS

CONTENTS (Continued)

Chapter 1

INTRODUCTION

Switching machines are commonly classified as combinational or sequential depending upon whether the value of the output is completely determined by the present values of inputs or it depends upon the past values of inputs as well. A physical realization of a switching machine by means of interconnection of various switching devices is called a switching circuit. Sequential circuits have been classified as synchronous or asynchronous depending upon whether the circuit is operating under the control of a clock signal or not. It is well known that any sequential machine can be realized by a number of different types of sequential circuits.

Due to the recent advancement in the technology of microelectronic circuit components (which may be small switching circuits themselves), the accompanying decline in the circuit element costs, and with ever increasing demand for faster operating circuits, the hardware considerations and synchronous circuits may be less important than maintenance considerations and asynchronous circuits respectively. Two primary concerns of circuit design are (i) the facility of detecting and correcting faults which may occur during the operation and (ii) the maximum possible speed of operation of the circuit. Both of the above demands could be met by devising constructional methods for asynchronous circuits which could lead to overall simpler structure.

The research presented in this thesis is restricted to asynchronous sequential circuits with level inputs, level outputs and operating in fundamental mode [22] . An attempt is made to realize any asynchronous

sequential machine by building blocks which leads to simpler overall circuit structures. Since the approach followed is analogous to one generally taken in the study of synchronous sequential circuits, this thesis also attempts to bring closer the asynchronous and synchronous circuit theories.

We will be using an asynchronous circuit element called asynchronous unit delay (AUD), which corresponds to a synchronous unit-delay. It is shown that the role of asynchronous unit delay can be served by an n-input, n-output asynchronous sequential circuit in which the present value of the output n-tuple is equal to the value of the input n-tuple prior to the last input change.

Chapter 1 consists of a brief review of the basic material that is common to all the chapters in this thesis.

In Chapter 2 we present a brief survey of the important results related to the subject discussed in this thesis. We have also interpreted some of the already existing results on feedback [12] in asynchronous circuits according to different concepts available elsewhere [17].

Chapter 3 presents a brief discussion on single error correcting codes and a code in which every pair of words w_i, w_j have distance = = 1 or ≥ 3. This code has been used to obtain a reduced table of an n x n AUD with single input changes only. Also, AUD's with single and multiple input changes are described. Different approaches which lead to different circuit realizations of the AUD are discussed and a short comparison of relative merits and demerits of different circuit models is also presented.

In Chapter 4 we consider a class of asynchronous circuits called "asynchronous definite", in which the internal state is a function of only a finite number of past input values. It is shown that

every asynchronous definite table can be realized by a feedback-free connection of AUD's and combinational networks and every feedback-free circuit of AUD's and gates has a definite table. It is also shown that asynchronous definite tables can be realized by circuits which do not need inertial[1] delay elements in any of their feedback loops but some inertial delays in the output leads of each AUD are required.

Chapter 5 considers the use of AUD as a basic element in the construction of any asynchronous sequential circuit. In general circuit design, we first show that a circuit constructed with AUD's also needs inertial delay elements to ensure proper operation. Then it is shown that any fundamental mode flow table F can be realized by a circuit of one AUD, m inertial delays and combinational networks. The circuit has feedback index m, where m is the smallest integer $\geq \lceil \log_2 \max (S_i) \rceil$, S_i being the number of stable states in any input column of F. Such a circuit realization may be a good compromise between reliable circuit operation, speed of operation and maintenance cost. Lastly, we also show that every indefinite asynchronous fundamental mode table can be realized by a circuit consisting of AUD's, one inertial delay and combinational gates and with feedback index 1. This means that if AUD's are constructed with S-R flip-flops and gates and if the internal feedback in the flip-flops is not taken into consideration, then the above asynchronous indefinite circuit of AUD's has actually a feedback index of one, because an AUD can be realized by a feedback-free connection of S-R flip-flops and combinational logic.

---

1 - An inertial delay element of magnitude Δ acts like a pure delay to pulses of width $\geq \Delta$, but does not respond to pulses of width $< \Delta$.

In the last section of Chapter 5 we consider "pulse controlled asynchronous fundamental mode circuits". Each such circuit has to have an input-change detector circuit, which generates a pulse whenever any input change occurs. This pulse controls the circuit operation. We have shown that such a realization of any fundamental mode table needs only one delay element. This delay element with many S-R flip-flops can act as many inertial delay elements.

1.1 Fundamentals : [10, 18, 22, 33]

Definition 1.1 : A fundamental mode asynchronous sequential machine (FMA machine) A is a quintuple :

$$A = (X, Y, Z, M, N),$$

where $X = \{X_1, X_2, \ldots, X_n\}$, $Y = \{Y_1, Y_2, \ldots, Y_s\}$ and $Z = \{Z_1, Z_2, \ldots, Z_m\}$ are finite, non-empty sets of input states, internal states and output states, M is the next state function :

$$M : Y \times X \rightarrow (Y \cup \{-\}),$$

where " - " represents a "don't care" situation and N is the output function :

$$N : Y \times X \rightarrow (Z \cup \{-\})$$

Furthermore, Rules (i) - (v) below apply to the operation of the machine.

Before the rules of operation are given, a few definitions are necessary. A total state of an FMA machine is a pair (Y_i, X_j) where $Y_i \in Y$ and $X_j \in X$. A total state (Y_i, X_j) is stable iff

$M(Y_i, X_j) = Y_i$; otherwise it is unstable. A total state is a "don't care" state iff $M(Y_i, X_j)$ = " - " .

Rules of Operation :

(i) A must be started in a stable state

(ii) The input (state) cannot change unless the present total state is stable ; otherwise the input can change any time :

(iii) Let (Y_{i_o}, X_{i_o}) be the starting state, and let the input change from X_{i_o} to X_j. Then

(a) If $M(Y_{i_o}, X_j) = Y_{i_o}$, the total state is again stable and the input can change again.

(b) If $M(Y_{i_o}, X_j) = Y_\ell \neq Y_{i_o}$ and if the total state (Y_ℓ, X_j) is stable, the input can change again as soon as the internal state has become Y_ℓ.

(c) If $M(Y_{i_o}, X_j) = Y_\ell$ and if the total state (Y_ℓ, X_j) is unstable, then the input cannot change and the internal state of the machine becomes $M(Y_\ell, X_j) = Y_k$. This operation continues with input constant until either a stable total state has been reached in a finite time and after a finite number of changes in the internal state or until a don't care state has been reached.

(d) If $M(Y_\ell, X_j)$ = " - " further action of the machine ceases to be of interest. This is a "don't care" or "unspecified" condition.

(iv) After the initial input change any stable state can be considered as a new starting state and the above rules apply.

(v) The output value is of interest only when the total state is stable.

Sequential machines will be represented as usual by flow tables with n columns corresponding to inputs and s rows corresponding to internal states. The entry in the i^{th} row and the j^{th} column consists of the pair $(M(Y_i,X_j), N(Y_i,X_j))$. A flow table having at most one stable state per row is called primitive [11]. When each unstable entry in a flow table leads directly to a stable entry, the table is called normal [12]. Note that every normal primitive flow table defines an FMA machine, but the converse is not generally true. However, each FMA machine can be represented by a normal flow table, when multi-step transitions are replaced by one-step transitions. This is allowed since we are only interested in transitions among stable states. Also a flow table that is not primitive can be expanded to an equivalent primitive form.

Definition 1.2 : An input sequence (a sequence of values from X) is applicable [25] to an internal state Y_i of an FMA machine iff, when the machine is placed in state Y_i and the input sequence is applied, all the internal states resulting from this sequence are specified except possibly the last.

Definition 1.3 : Let F_1 and F_2 be flow tables of FMA machines. An internal state Y_i of F_1 is said to cover [25] an internal state Y_j of F_2 (written $Y_i \supseteq Y_j$) iff when any input sequence applicable to Y_j is applied to F_1 and F_2 started in Y_i and Y_j respectively, the outputs of F_1 and F_2 are identical, whenever the output of F_2 is specified.

Definition 1.4 : A flow table F_1 covers [25] a flow table F_2 ($F_1 \supseteq F_2$) iff for each internal state Y_j of F_2 there is an internal state Y_i in F_1 such that $Y_i \supseteq Y_j$.

If $F_1 \supseteq F_2$ and if F_1 has fewer internal states than F_2, then at least one state of F_1 must cover more than one state of F_2. Whenever two states Y_i and Y_j of a flow table can be covered by a single state of another flow table, then Y_i and Y_j must be "compatible" as defined below.

Definition 1.5 : Two internal states Y_i and Y_j of a flow table F are compatible [25] iff for any input sequence applicable to both Y_i and Y_j, the output sequence which results when F is initially in Y_i is the same as the output sequence which results when F is initially in Y_j, whenever both outputs are specified. It is understood that the output sequence consists only of outputs associated with stable total states.

Definition 1.6 : The flow table of an FMA machine is reduced, if no two internal states of F are compatible.

It must be noted that a given table can be covered by more than one reduced table. If there are no don't care entries the reduced table is unique.

Definition 1.7 : An asynchronous sequential circuit is a structure with p binary inputs $x_1, x_2, \ldots, x_p$, q binary outputs $z_1, z_2, \ldots, z_q$, and r binary memory elements (such as delays, feedback loops or flip-flops) with outputs $y_1, y_2, \ldots, y_r$, such that ;

$$y_i(t+\Delta) = f_i(y_1(t), y_2(t), \ldots, y_r(t) ; x_1(t), x_2(t), \ldots, x_p(t))$$

$$z_j(t) = g_j(y_1(t), y_2(t), \ldots, y_r(t) ; x_1(t), x_2(t), \ldots, x_p(t))$$

where f_i and g_j are Boolean functions realized by combinational networks, $y_i(t+\Delta)$ is the next value of $y_i(t)$ and Δ is the response time of the i^{th} memory device.

Since various specific realizations of asynchronous sequential circuits are adequately described in the literature [11, 18, 22, 33], the above general definition will suffice here, and the reader is referred to the literature for more details. Every sequential circuit defines a unique sequential machine, if one defines X to be the set of values of the p-tuple $(x_1, x_2, \ldots, x_p)$ and Y and Z are similarly defined by $(y_1, y_2, \ldots, y_r)$ and $(z_1, z_2, \ldots, z_q)$, and if the M and N functions are computed from the sets of Boolean functions $(f_1, f_2, \ldots, f_r)$ and $(g_1, g_2, \ldots, g_q)$ respectively. Note that the sequential machine defined by a circuit does not have don't care entries, i.e., is fully specified. We shall say that a circuit is a fundamental mode circuit if the corresponding machine, which the circuit represents, is the FMA machine. A circuit C is said to realize an FMA machine A iff the flow table of the machine defined by C covers the flow table of A.

If an asynchronous circuit is in a total state in which the next values of two or more internal state variables y_i differ from the present values, the circuit is said to have a race. If the final stable state reached as a result of the race depends on the order of change of the variables the race is called critical.

Another problem which can cause incorrect circuit operation is the presence of hazards. The following are the different types of hazards generally found in switching networks.

Static hazard : A combinational network is said to contain static hazard if and only if there exists a single input variable change for which the output before the change is equal to the output after the

change, and during the change a momentary false output pulse may occur. Sometimes it is also referred to as a combinational hazard. Methods of eliminating such hazards have been developed by Caldwell [11] , Huffman [19] and McCluskey [22] .

Essential hazard : A flow table is said to contain an essential hazard, if there exists a total stable state (Y_i, X_1) and a second input state X_2 which can follow X_1 such that, starting in (Y_i, X_1) the input sequence $X_2X_1X_2$ leads to a different state than the sequence X_2.

Steady state hazard : A hazard in a sequential circuit which may cause a transition to an improper stable state.

Transient hazard : These are momentary false output pulses which appear at the output terminals of the circuits.

We close this chapter with a brief discussion of partition theory [17] which will be required later.

Definition 1.8 : A partition π over a set S, is a collection of disjoint subsets of S, called blocks, such that their set union is S. If π over S has two blocks then it is called a binary partition.

Two states which appear in the same block of a partition π are said to be identified by π. We denote the trivial partition where each block contains a single state (no states are identified) by zero, and trivial partition having a single block (all states are identified) by S. We designate the block of π which contains $s \in S$ by $B_\pi(s)$. We also write $s \equiv t(\pi)$ iff s and t are contained in the same block of π, i.e. $s \equiv t(\pi)$ if $B_\pi(s) = B_\pi(t)$.

Definition 1.9 : A partition π on the set of internal states of an FMA machine is said to have the substitution property (S.P.) iff

$$Y_i \equiv Y_j(\pi) \longrightarrow M(Y_i X_k) \equiv M(Y_j X_k)\ \pi$$

for all $X_k \in X$

Definition 1.10 : The product of two partitions

$\pi_1 = \{b_{11}; b_{12}; \ldots b_{1i}\}$ and $\pi_2 = \{b_{21}; b_{22}; \ldots; b_{2j}\}$

is a partition given by

$$\pi_1, \pi_2 = \{b_{11}b_{21};\ b_{11}b_{22}; \ldots;\ b_{1i}b_{2j}\}.$$

The following definitions and theorems have been stated and proved [17] for synchronous sequential machines. We notice that they hold true for FMA machines also.

Definitions 1.11 : Given the state machines (machines without outputs)

$$A' = \{Y', X', M'\} \text{ and } A'' = \{Y'', X'', M''\}$$

With $X'' = Y' \times X'$, an output set Z and output function

$$N : Y' \times Y'' \times X' \longrightarrow (Z \cup \{-\})$$

then the serial connection of A' and A'' with the output function N is the machine

$A = A' \multimap A'' = \{Y' \times Y'',\ X',\ Z, M, N\}$

where $M((Y'_i, Y''_j), X_k) = (M'(Y'_i X_k), M''(Y''_j, (Y'_i, X_k)))$

and $N : Y' \times Y'' \times X' \longrightarrow (Z \cup \{-\})$

Definition 1.12 : The parallel connection of two machines

$A' = \{Y', X', Z', M', N'\}$ and

$A'' = \{Y'', X'', Z'', M'', N''\}$ is the machine

$A = A' // A'' = \{Y' \times Y'',\ X' \times X'',\ Z' \times Z'',\ M, N\}$

with $M((Y'_i, Y''_j), (X'_k, X''_\ell)) = (M'(Y'_i, X'_k), M''(Y''_j, X''_\ell))$

and $N((Y'_i, Y''_j), (X'_k, X''_\ell)) = (N'(Y'_i, X'_k), N''(Y''_j, X''_\ell))$.

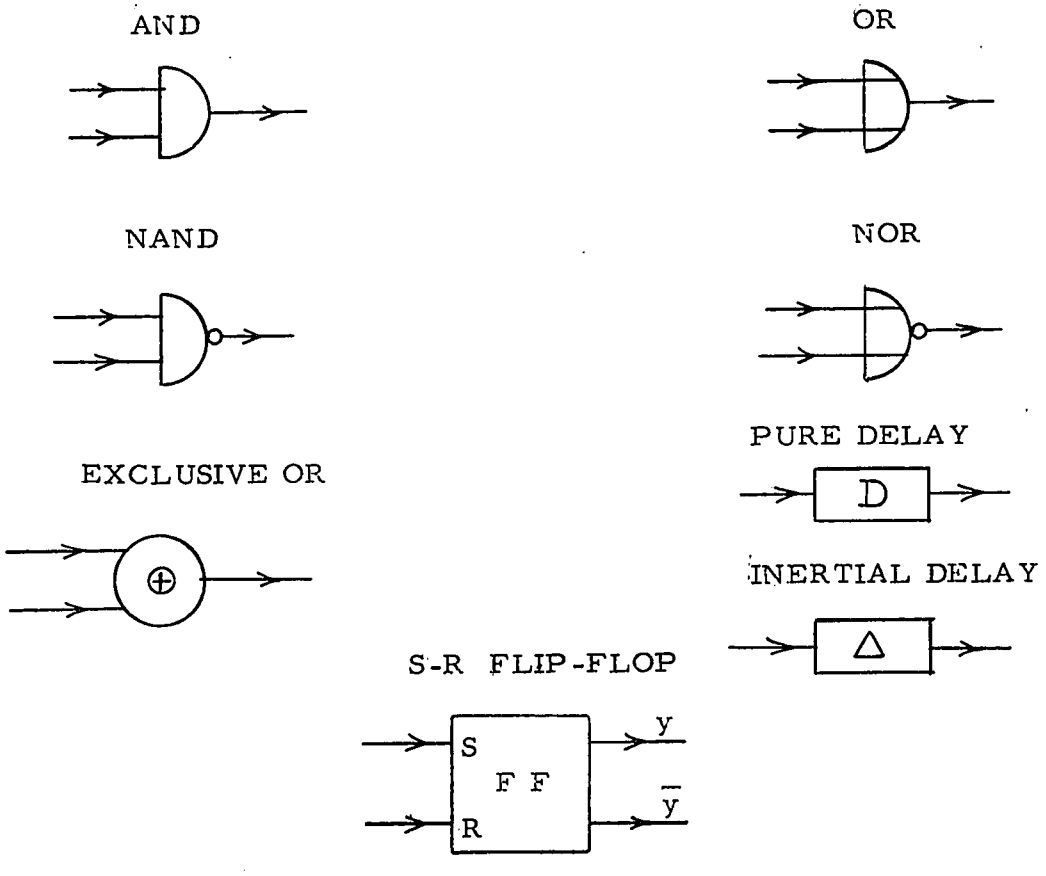
Definition 1.13 : The machine $A' \multimap A''$ is a serial decomposition of A iff $A' \multimap A''$ realizes A. Similarly the machine $A' \,//\, A''$ is a parallel decomposition of A iff $A' \,//\, A''$ realizes A.

We say that state behavior decomposition of A is non-trivial iff A' and A'' have fewer states than A.

Theorem 1.1 : A sequential machine has a non-trivial serial decomposition of its state behavior iff there exists a non-trivial S.P. partition on the set of states Y of A.

Theorem 1.2 : A sequential machine has a non-trivial parallel decomposition of its state behavior iff there exists two non-trivial S.P. partitions π_1 and π_2 on $\bigwedge^{Y \text{ of } A}$ such that $\pi_1 \cdot \pi_2 = 0$.

1.2 Notations :



Chapter 2

A SURVEY OF CURRENT RESULTS

Introduction:

Since the state assignment, the amount of feedback and the number of delay elements in an asynchronous switching circuit are important to the circuit structure, we present a brief survey of the results available in this connection. The results obtained in this thesis are also related to the structural properties of asynchronous circuits.

2.1 State assignment

In asynchronous circuits the assignment of state variables must not only provide a minimal circuit structure (in some sense), but also guarantee that each internal state transition will always lead to the correct stable state regardless of relative speeds of various circuit elements. Thus an assignment free of critical races is required and commonly, the least possible number of secondary variables is used. It is well known that such an assignment of minimum number of state variables may not always provide a structure with the minimum possible combinational logic. For example, any delayed input circuit [12] realization of an FMA machine with more than one input needs a greater number of state variables compared to its equivalent conventional circuit realization, but the delayed input circuit is often more economical in terms of hardware.

The state assignment problem for asynchronous circuits has been discussed in the literature [11, 18, 19, 20, 32] quite extensively. Most of the methods already available are based on the following two approaches.

The first approach towards the solution of the problem is to assign state variables so that each inter-row transition is represented by adjacent states. Two states are said to be adjacent if they differ in only a single variable. In a circuit of this type only one state variable is excited at any time during the transition. If we define a unit time as the time required for one state variable to change, this time being assumed uniform, then an internal state transition from one stable state to another stable state in such a circuit will take ℓ units of time, ℓ being the number of state variables required to change. Obviously such a circuit will not be able to give the full advantage of the basic speed of circuit elements, though it will operate properly. This approach has been used to obtain one-to-one as well as many-to-one state assignments and is applicable both to completely and incompletely specified flow tables. The speed of the circuit can be improved by assigning more than one binary code per state, yet keeping the number of binary variables to its minimum in particular cases. For details, the reader is referred to Caldwell [11] and Huffman [19] .

The second approach is to allow changes in more than one variable at the same time. In this case we must be certain that every race condition is non-critical. The internal transition speed of this type of state variable assignment may be the same as that of a circuit in which each internal state requires a change of only a single variable, because non-critical races can take place.

Liu [20] developed a method, based on the concurrent operation of state variables, so as to increase the speed of the circuit. His principal results can be summarized as follows:

In a fundamental mode flow table F , if we consider each column separately and assign a sufficient number of state variables for each column, then the combined state variable assignment (the assignment obtained by putting together all the individual assignment for various columns) has no critical races. This is the starting point in his procedure.

Obviously, the combined assignment will have many more state variables than any method based upon the direct attack on the entire problem. First a table, called the assignment table F_A is prepared, in which each row corresponds to the combined assignment. The table F_A is an array of 0's, 1's and -'s if F is not completely specified. A column of F_A contains one entry from each row of F_A. An assignment with a reduced number of variables is obtained, (i) by discarding those columns in F_A which are covered[2] by some other columns in F_A and (ii) by replacing columns in F_A by their intersection column[3].

The reduced assignment obtained in the above fashion often leads to a simpler final circuit, but the method does not guarantee that the reduction in the number of variables will yield a minimal solution. However, Liu also presents an upper bound assignment method in which he shows that any 2^m row flow table can always be coded with 2^m-1 variables to yield a minimum transition time assignment. This result is the same as that of Huffman [19] obtained in his general assignment method. The knowledge of such an upper bound on the number of state variables helps in evaluating how good a particular solution is in particular cases.

---

2. Given two columns p_i and $p_j \in F_A$, p_i is said to <u>include</u> p_j iff p_j agrees with p_i wherever the latter is 1 or 0. Column p_i is said to <u>cover</u> column p_j iff either p_i includes p_j or p_i includes the complement of p_j, where complement of any column p_i is obtained by changing all 0's to 1's and all 1's to 0's and leaving the don't care's unchanged.

3. Given two columns p_i and $p_j \in F_A$, there exists an <u>intersection column</u> of p_i and p_j iff p_i and p_j agree whenever both p_i and p_j are 1 or 0. This intersection column agrees with both p_i and p_j where p_i and p_j agree with each other, but agrees with either p_i or p_j when the other has don't care entries.

In a recent paper Tracy [32] further extends this class of assignments and develops three algorithms which lead to minimal state variable assignments having the property that all state transitions are completed in minimum time. He uses the theory of partitions to determine such a state assignment. His first method is based on the following result :
A row assignment allotting one state variable per row can be used for direct transition realization of normal flow tables without critical races iff, for every state transition $|Y_i, Y_j|$
(a) if $|Y_m, Y_n|$ is another transition in the same column, then at least one state variable partitions the pairs $|Y_i, Y_j|$ and $|Y_m, Y_n|$ into separate blocks,
(b) if Y_k is a stable state in the same column then at least one state variable partitions the pair $|Y_i, Y_j|$ and state Y_k into separate blocks and
(c) for $i \neq j$, Y_i and Y_j are in separate blocks of at least one state variable partition. Then from the list of all such binary partitions which satisfy (a),(b) and (c) we find partitions such that each partition from the list is $\leq \Pi$ for some τ - partition[4] Π, and an effort is made to obtain a minimum possible number of such τ - partitions.

The second method of Tracy is shorter. First he defines a k-set in a single column of a flow table, which consists of all k - 1 unstable entries leading to the same stable state Y_i and stable state itself (the k-set is denoted by kY_i). He proves that a direct transition in a k-set kY_i does not race critically with a direct transition in k-set kY_j if an assignment has been made such that at least one state variable partitions the element of kY_i and elements of kY_j into separate blocks. Hence method 2 consists of constructing a partition list from the k-sets of the

---

4 - The binary partitions $\tau_1, \tau_2 \ldots \tau_p$ induced by the state variables $y_1, y_2, \ldots y_p$ in a minimum transit time assignment are called the set of τ - partitions of that assignment.

flow table instead of from the transition pairs prescribed in the first method. From the partition list of k-sets again we find the minimum possible number of τ-partitions which cover all the partitions in the list.

His third method is similar to that of Liu. As pointed out earlier this method is very efficient only in some cases. For most of the flow tables, method 3 is easier to apply than method 2 and method 2 is easier than method 1. The upper bound methods of Liu and Huffman give an upper bound on the number of variables for method 3, method 3 gives an upper bound for method 2 and method 2 given an upper bound for method 1.

2.2 Feedback

Recently Eichelberger [12], Huffman[5] and Friedman [13] have studied the problem of realizing a given fundamental mode flow table by an asynchronous sequential circuit with minimum feedback index. The feedback index of the circuit is the minimum number of wires which must be cut to eliminate all the feedback loops. Eichelberger has treated the problem by using a design technique called "delayed input circuit" which is inherently free of critical races. Therefore one can make an unrestricted state assignment so as to minimize the feedback index. His main results can be summarized as follows : Any normal fundamental flow table containing a maximum number S of stable states in any column can be realized with feedback index m, where m is the smallest integer $\geq \lceil \log_2 S \rceil$. The advantage of delayed input circuits over conventional asynchronous circuits is that they are more economical in terms of logical hardware, if the delaying of the inputs can be done inexpensively. Delayed input circuits have the disadvantages of requiring delays of different magnitudes, restricting input changes so that changes in the input sig-

5 - Huffman, D.A., Unpublished paper, Lecture delivered at Princeton University Symposium on Switching Theory, January, 1962.

nals can only occur one at a time and finally they can only be used to realize normal fundamental mode flow tables.

Figure 2.1 represents the model of a delayed input circuit and this works properly under the following assumptions.

1) $D_1(\min) \geq 2d(\max) - d(\min)$, where d represents the delay through the combinational network.

2) $D_2(\min) \geq D_1(\max) + d(\max) - 2d(\min)$.

3) The portion of the combinational network generating y'[6] signals is free of hazards for all allowable network input changes.

4) y'_i is independent of $\tilde{y}_j$ for $i \neq j$.

5) There exists no set of values for $2n + m$ variables x, $\underline{x}$ and y for which $y_i = \tilde{\bar{y}}_i$.

Eichelberger develops this model on the concept of using controlled hazards (static hazards and delays) for designing fundamental mode asynchronous sequential circuits. The delay in the input lead performs two functions. First, it changes static hazards into controlled hazards, thus generating pulses for input level changes. Second, it provides input memory so that during a transition caused by an input change, the previous input before the change can be used to determine the next internal state.

Considering the same basic assumptions as made by Eichelberger about the different magnitudes of the delays in the circuit, we develop the delayed input model on the concept of partition theory. We proceed as follows :

Lemma 2.1 : Any n-input reduced normal primitive flow table of an FMA machine has a 2^n-block S.P. partition π_F .

---

6 - y'_i represents the next value of the state variable y_i.

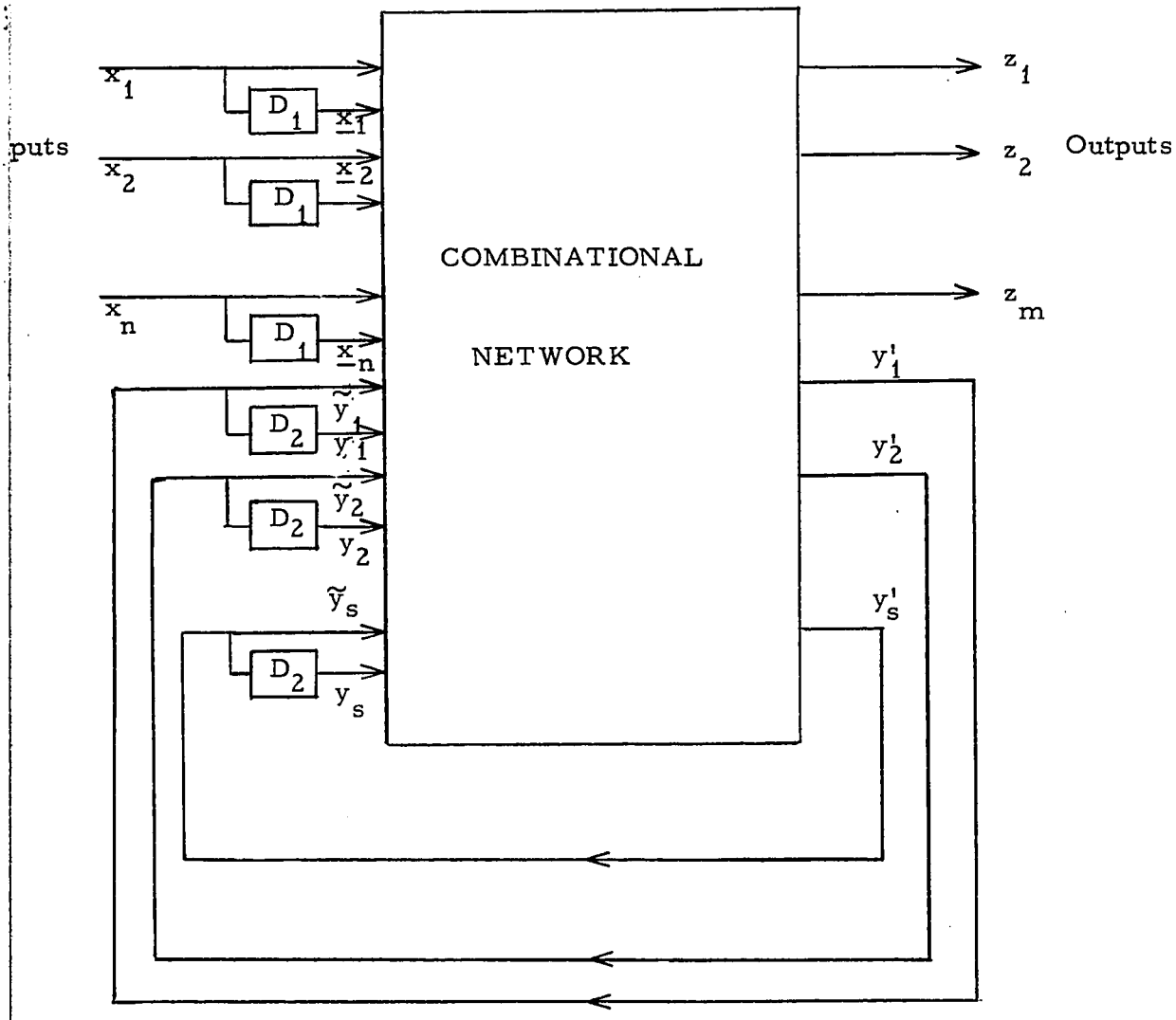


Figure 2.1 - Delayed input model

Proof : We know that in any normal primitive flow table F with n-inputs the set of stable states in any input column X_i and the set of stable states in any other input column X_j are disjoint. Therefore if we consider each such set of stable states as a block of some partition Π_F on the set of states, Π_F will be a 2^n block partition. It is quite obvious that for any $Y_i \equiv Y_j \ (\Pi_F)$ we shall have $M(Y_i, X_k) \equiv M(Y_j, X_k) \ \Pi_F$ for all X_k of F. Therefore Π_F has S.P.

Lemma 2.2 : Any n-input normal primitive flow table of FMA machine has n binary S.P. partitions and the product of these n partitions is equal to S.P. partition Π_F .

Proof : Since there are n-input variables, one can always form n two-block-partitions by choosing all the stable states in those columns of F for which a particular variable x_i is 1 for one block and 0 for the other. It is again quite obvious that all such n binary partitions will have S.P. and their product will be equal to Π_F.

As a result of Lemma 2.1 and Theorem 1.1, it is clear that any FMA machine A described by a normal primitive flow table must be realizable by a serial connection $A_1 \longrightarrow A_2$ of two smaller machines A_1 and A_2. At the same time, due to Lemma 2.2 and Theorem 1.2, A_1 is again decomposible into n 2-state machines $A_{11}, A_{12}, \ldots A_{1n}$ connected in parallel. In the case of a normal primitive flow table of any FMA machine, $A_{11}, A_{12}, \ldots A_{1n}$ always turn out to be trivial delays in the input lines.

Example : 1

	x_1x_2 00	01	11	10
1	(1)	(1)	3	4
2	(2)	(2)	4	4
3	-	2	(3)	4
4	2	1	(4)	(4)

Figure 2.2

First we describe the flow table of Figure 2.2 by its reduced normal primitive flow table as shown in Figure 2.3

	x_1x_2 00	01	11	10
1_a	(1_a)	1_b	-	4_b
1_b	1_a	(1_b)	3	-
2_a	(2_a)	2_b	-	4_b
2_b	2_a	(2_b)	4_a	-
3	-	2_b	(3)	4_b
4_a	-	1_b	(4_a)	4_b
4_b	2_a	-	4_a	(4_b)

Figure 2.3 - Normal primitive flow table of Figure 2.2

From Figure 2.3 it can be easily verified that

$\Pi_F = \{\overline{1_a, 2_a}, \overline{1_b, 2_b}, \overline{3, 4_a}, \overline{4_b}\}$ has S.P. Thus we know from Theorem 1.1 that the table of Figure 2.3 can be realized by a serial connection of 4-state and 2-state machines A_1 and A_2. The flow table F_1 for A_1 can be constructed directly from the normal primitive flow table of Figure 2.3. To construct flow table F_2 for A_2 we have to choose

	x_1x_2 00	01	11	10
A	(A)	B	C	D
B	A	(B)	C	D
C	A	B	(C)	D
D	A	B	C	(D)

$$\Pi_F = \{\overline{1_a 2_a},\ \overline{1_b 2_b},\ \overline{3,4_a},\ \overline{4_b}\} = (A,B,C,D)$$

Figure 2.4 - Flow table F_1

a 2-block partition τ_F on the states of normal primitive flow table such that

$$\Pi_F \cdot \tau_F = 0.$$

Let

$$\tau_F = \{\overline{1_a, 1_b, 3, 4_b},\ \overline{2_a, 2_b, 4_a}\} = (a,b).$$

We construct F_2 by rewriting the primitive flow table of Figure 2.3 by replacing its states by the corresponding blocks of τ_F and rearranging the information to obtain F_2 as shown in Figure 2.5.

It is obvious that F_1 can again be decomposed into two parallel machines which are trivial delays. Hence the next state equations which realize the flow table of Figure 2.2 are

$$y'_1 = x_1 \quad \text{for machine } A_{11},$$

$$y'_2 = x_2 \quad \text{for machine } A_{12}.$$

And if we assign binary code 0 and 1 to a and b respectively, we obtain $y'_3 = x_1\bar{y}_1y_2 + \bar{x}_1\bar{y}_2\bar{y}_3 + \bar{x}_1y_1\bar{y}_3 + x_1x_2y_3 + \bar{y}_1y_2y_3$ for machine A_2.

		x_1x_2 00	01	11	10
A	a	a	a	-	a
	b	b	b	-	a
B	a	a	a	b	-
	b	b	b	b	-
C	a	-	b	a	a
	b	-	a	b	a
D	a	b	-	b	a

	A00	A01	A11	A10	B10	B11	B01	B00	C00	C01	C11	C10	D10	D11	D01	D00
a	a	a	-	a	-	b	a	a	-	b	a	a	a	a	-	b
b	b	b	-	a	-	b	b	b	-	b	b	-	-	-	-	-

$$\tau_F = \{\overline{1_a, 1_b, 3, 4}, \quad \overline{2_a, 2_b, 4_a}\} = (a, b)$$

Figure 2.5 - Construction of F_2.

Therefore it seems that delayed input circuit realization of machines described by normal primitive flow tables is a very natural choice for more economical circuits in terms of logical hardware. Of course the proper operation of such circuits depends upon the relative magnitudes of different delays to be used in the input lines and feedback loops, and the delays must be bounded according to basic assumptions made by Eichelberger.

It must be noted that our approach clearly indicates that it is impossible to find an m block partition τ_F such that for any normal primitive flow table description the conditions $m < \log_2 S$ and $\Pi_F \cdot \tau_F = 0$ are simultaneously satisfied. Hence this also illustrates the statement that any normal primitive flow table F cannot be realized by a delayed input circuit C with feedback index $m < \lceil \log_2 \max(S_i) \rceil$ where S_i is the number of stable states on any column of F [12, 13].

Huffman treats the problem of realizing fundamental mode flow tables with minimum feedback index by restricting the state assignment, so that no state transitions involve critical races. He again shows that a feedback index $m \leq \lceil \log_2 \max(S_i) \rceil$ is required. He then shows that any normal fundamental mode flow table having Λ states can be realized with a feedback index $\leq m_0 + 1$, where $m_0 = \lceil \log_2 \Lambda \rceil$. Any non-normal flow table can be realized with a feedback index $\leq m_0 + 2$.

Huffman's design technique needs inertial delays. The schematic circuit diagram due to Huffman's procedure is shown in Figure 2.6. Inertial delays are necessary in the circuits designed by his approach for the following reason : It is evident from the circuit of Figure 2.6 that the next state variables y'_2 are functions of the input and the feedback variables q_i. As shown in the Figure 2.6 a feedback variable Q_i is the input to an amplifier and appears as output q_i of the amplifier. It

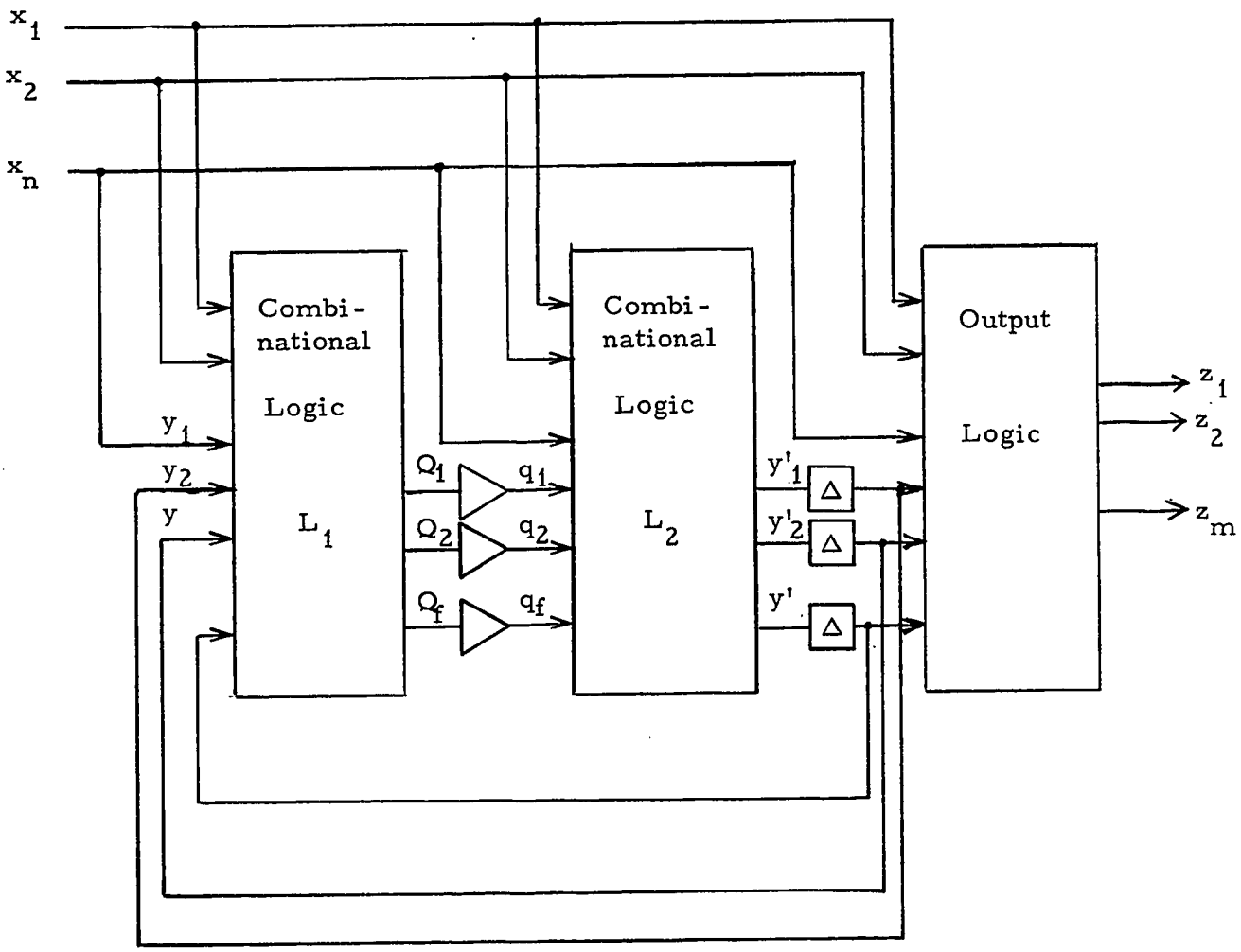


Figure 2.6 - Huffman's circuit model

is assumed that there is no delay associated within the amplifier; hence $Q_i = q_i$. Since Q_i is again a function of the present state variables and inputs, a change in the value of inputs can affect y'_j through two different paths, one through L_1 and the other more directly. Such a situation is the basic condition for a static hazard. Thus transient erroneous pulses may appear at the inputs of any state device, and therefore these must be avoided from appearing at the y_i terminals of L_1 so as to ensure the proper operation of the asynchronous circuit. This is possible only if there are inertial delays present in all state branches.

Friedman's approach to this problem also yields circuits that are similar to those of Huffman. For any given fundamental mode flow table, Friedman first finds a non-critical race-free state assignment by using a method very similar to Liu's [20] and then he shows that if the present input of F is X_i, then the knowledge of the values of all the state variables y^i, assigned for non-critical race-free assignment of states in input column X_i of F, is sufficient to determine the next stable state of F according to its flow table description. In other words the next value of any state variable $(y_k^j)'$ can be expressed as a function of those variables with superscript i when the input is X_i and hence (y_k^j) can be expressed as a sum of different functions for all inputs of F, i.e.

$$(y_k^j)' = \sum_{i=1}^{n} X_i \; g_k^{ji} \; (y^i_1 , y^i_2, \ldots y^i_{a_i}) ,$$

where n is the number of input columns of F.

His principal results can be summarized as follows:

(i) Any fundamental mode flow table can be realized as a fundamental mode circuit with feedback index $\lceil \log_2 \max (S_i) \rceil$, where S_i is the number of stable states in column X_i of the table. Unlike Eichelberger's results a flow table does not have to be normal.

(ii) Since he uses double rail inputs and concentrates on the diode-transistor logic, the circuit realization requires no inverters, and hence his method gives minimal transistor solution.

(iii) Simple R-C integrating circuits can be used as inertial delay elements for state variables.

(iv) Any fundamental mode flow table can also be realized as a nonfundamental mode circuit with feedback index one, assuming ideal inertial delays as state devices. He has also suggested a scheme, using R-C networks, diodes and a single threshold device with a feedback loop, which acts as an ideal inertial delay.

We shall return to the feedback in Chapters 4 and 5.

2.3 Delay Elements in Asynchronous Circuits

It is well known that by inserting delays in feedback loops (pure delays for circuits, without combinational hazards, inertial delays for circuits with combinational hazards), the correct operation in asynchronous circuits can be obtained despite the presence of different stray delays in the circuits. Unger [33] has shown that the state behavior of any FMA machine can be realized by a properly operating asynchronous circuit which either does not need any delay elements or needs one delay element only.

In a recent report [1] Armstrong, Friedman and Menon have further extended Unger's work by taking alternative approaches. Since some of the results obtained in Chapter 4 and 5 are also somewhat related to these results, it will be worthwhile to discuss briefly the works of the above mentioned authors.

Unger's principal results are as follows :

(a) If the table F does not have any essential hazards, then it can always be realized by a circuit without delay elements and this circuit

will be free of steady state hazards. Transient hazards may occur in some cases.

(b) If F contains an essential hazard, then any proper circuit realization of F must contain at least one delay element. If more than one input variables are permitted to change, then transient hazards may occur in these circuits. The proof is by construction.

It is shown that any flow table with essential hazards can be constructed by a circuit called a 'delay box' and the combinational network of Figure 2.7. This delay box for any number of inputs can always be realized by a circuit of one delay element only. As an illustration, his method of realization of a delay box for three variables is shown in Figure 2.8.

The delay box provides the same behavior as having a delay element in each feedback line as long as only one state variable changes at a time and there is sufficient time between changes for the delay box to become stable. Since many asynchronous circuits are designed in which only one state variable is allowed to change at any time, these restrictions on the operation of the delay box are not serious.

Although the delay box is somewhat complex and would probably be not used as a replacement for feedback delay elements, it does prove the point that one delay element is sufficient for realizing an asynchronous machine that contains essential hazards. However, in Chapter 4 we shall show that there are some FMA machines which have essential hazards, but can be realized by circuits without delay elements in their feedback lines, although such delay elements will be necessary elsewhere.

Unger's results [33] imply the possibility that the line delays (inherent delay due to stray capacitances of feedback line plus the delay due to delay element if it has been inserted in the line) may exceed the

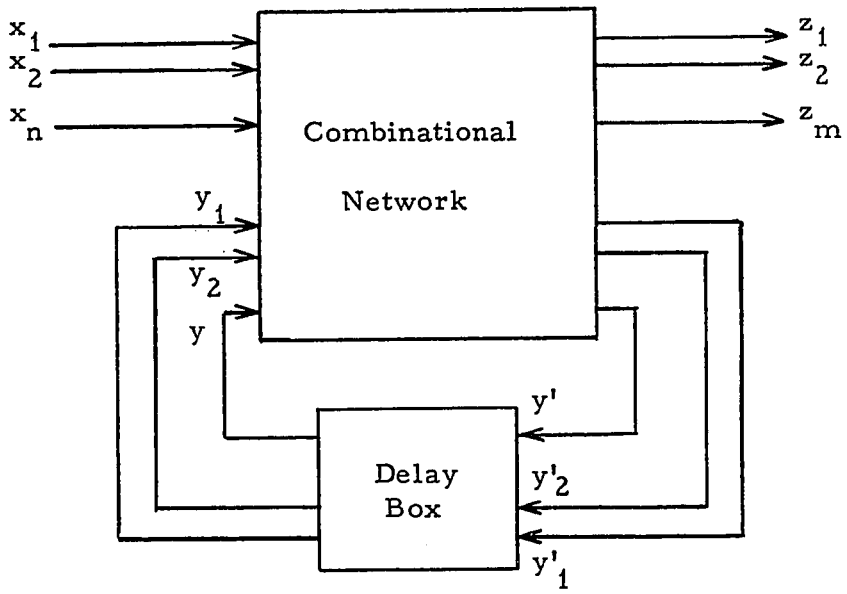


Figure 2.7

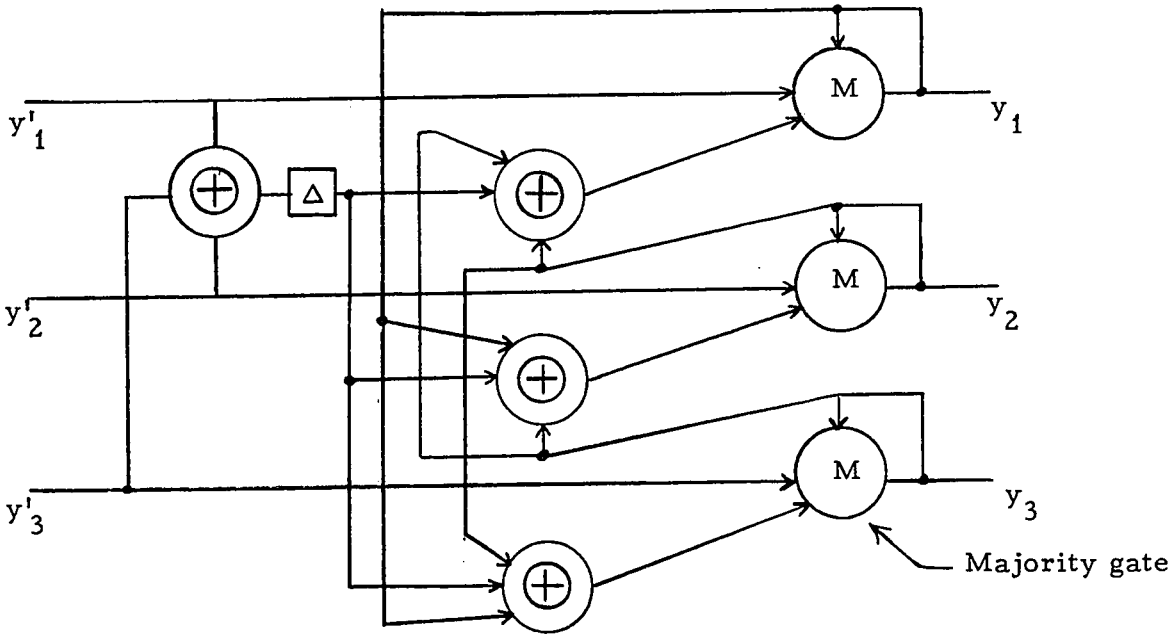


Figure 2.8

gate delays to ensure the proper operation of a circuit. Muller [23] and recently [1] Armstrong, Friedman and Menon show that if this possibility can be forbidden then any FMA machine can be realized without using any delay element, provided only single input variable changes are allowed in any transition.

For example, a lower bound on a loop delay may be $N\delta$, where N is the number of gates in the loop and δ is the minimum delay in any gate. The approach taken by Armstrong, Friedman and Menon is as follows :

If one wants to realize a circuit without inserting delay elements one must guarantee that (a) all gates should sense the change in the input variable x before a change in the state variable y, (b) there should be no transients caused by combinational hazards in y-variables which should remain fixed during any transition.

In general, the Boolean equations of a sequential circuit require both negated and unnegated input variables. Therefore, if inverters are used on the inputs, a change in x will not result in a change in $\bar{x}$ until some time δ associated with the inverter. Hence to ensure that the change in x is seen before the subsequent change in y the use of inverters must be avoided. This has been achieved by replacing first-level AND gates in the sum of products form of equations by NOR-AND pairs where necessary, thus eliminating all the appearances of $\bar{x}$. In doing so, effectively, we are taking those x-variables which must be complemented and y-variables with which x-variables are racing and passing through the same NOR gate. Because some non-zero value of line delay (due to stray capacitances) is assumed, this fixes the race so that first level gates see the x changes before a change in y. The authors of the above mentioned report suggest a systematic procedure by partitioning the literals into two sets, one containing all

complemented input variables and other uncomplemented input variables. The first set is realized as a complement of its complement, which is just the NOR of the complements of the literals in the set. To resolve the second difficulty they have suggested two procedures.

(1) A realization using a special factoring

(2) A realization with flip-flops.

A realization using special factoring

If the races on the first level gates are fixed as discussed above and a single state transition assignment [20, 33] for coding the states of the circuit is used, no transient can occur in the inputs to the second level OR gate. The transients at the output of the OR gate occur when some y' map has the configuration shown below and the 1's in the transition subcube[7] cannot be covered by a single implicant. In such a situation

	X_k	X_ℓ
Y_i	1	1
Y_j	0	1

y'

y' should remain fixed at 1. This is resolved by covering the 1's in the transition sub-cube by the 1-term $L_k\bar{Z}_k$, where L_k is the transition subcube and Z_k is the sum of all terms covering all 0 points of L_k.

---

7 - The transition subcube of the transition $(Y_iX_k)\rightarrow(Y_jX_\ell)$ is the smallest subcube in the cube of the total states containing (Y_i, X_k) and (Y_j, X_ℓ).

An $L_k \bar{Z}_k$ term must be used for each transition during which a y-variable is required to remain fixed at 1. Thus the state variable logic may be realized in the following special form :

$$y_i' = \sum_{k=1}^{n} L_k \bar{Z}_k + \sum_{i=1}^{m} T_i .$$

where T_i represent simple terms which cover those 1 vertices of y_i' not covered by L_k. Since T_i can be realized by NOR-AND pairs, hence there are no incorrect transients in y_i' .

The second method using S-R flip-flops makes use of the fact that S-R flip-flops may temporarily produce 0's on both outputs during transitions but never transient 1's. For details, the reader is referred to the report [1] .

Armstrong, Friedman and Menon in the same report also present a method for single-delay circuit realization of any normal fundamental mode flow table F. Such a circuit consists of two component circuits, a source box (or encoder) and sequential circuit C' as shown in Figure 2.9. C' does not require any delay element in its feedback loops. The source box receives the input signals and codes them as proper inputs to C'. The input signals to the source box are assumed to be levels and the time between the changes is assumed to be sufficient for all transitions in the source box and the transition of states of C' to be completed. All input changes are allowed, provided the changes are completed within a certain specified time after the first change. On detection of an input change, the source box generates a spacer at its output terminals for a period of time Δ , after which the coded output is produced. The spacer is the all-zero-tuple which appears between any two successive inputs of F. It is proved that this source box is always realizable by a circuit of one delay as shown in the Figure 2.10.

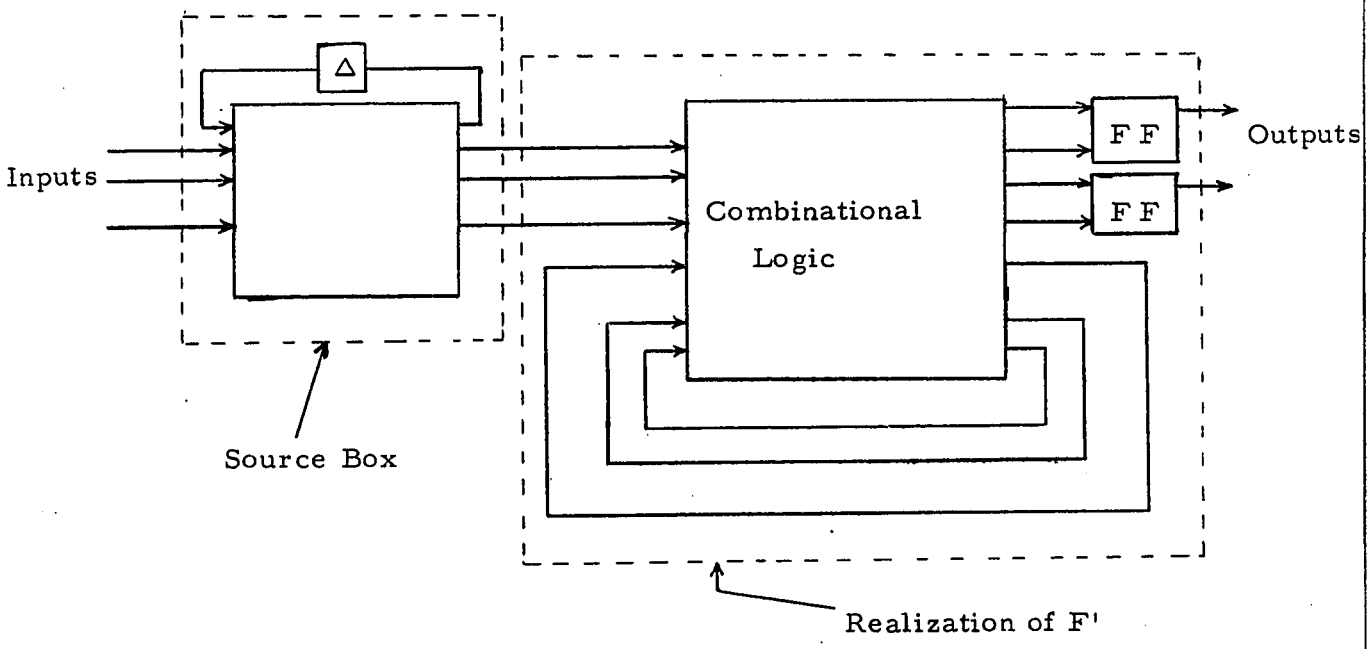


Figure 2.9.

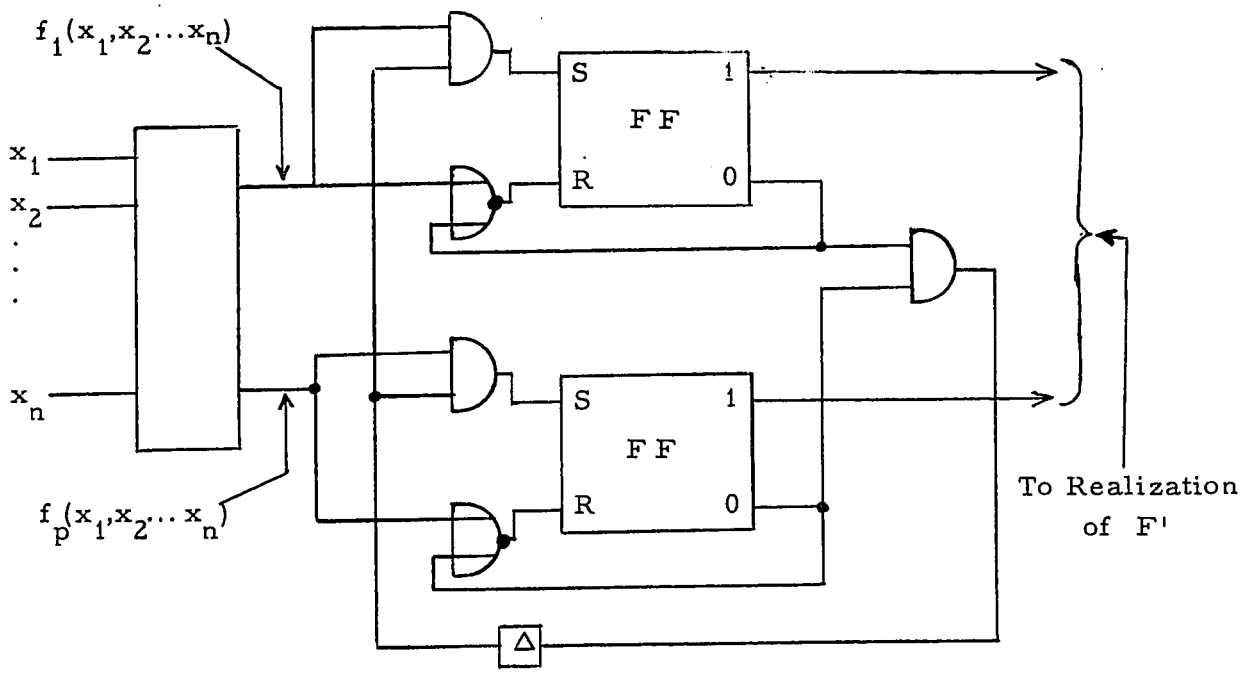


Figure 2.10.

Since for any change of input to F the source box first produces an all zero signal for time Δ, and then automatically produces a coded input corresponding to the changed value of input of F, the augmented flow table F' realized by C' is of the nature as shown in Figure 2.11. It is obvious that an augmented table F' of any F will never have any essential hazards. Therefore C' does not need any delay elements in its feedback loops.

	x_1 = 0	x_1 = 1
1	(1), 00	2, 00
2	3, 11	(2), 10
3	(3), 11	3, 01
4	1, 00	(4), 01

F

	x_1x_2 = 00	01	11	10
1	(1), 00	2, 10	-	(1), 00
2	(2), 10	(2), 10	-	3, 11
3	(3), 11	4, 01	-	(3), 11
4	(4), 01	(4), 01	-	1, 00

F'

Figure 2.11

This type of one-delay circuit has some advantages over Unger's one-delay circuit, such as :

(i) It works well for all input changes.

(ii) Its speed of switching is comparable to that of Unger's circuit.

(iii) Though the complexity of the source box increases with the number of inputs, it is more economical for the realization of tables with a large number of states compared to Unger's circuit.

Chapter 3

ASYNCHRONOUS UNIT DELAYS

Introduction :

The technological details of a circuit which realizes the behavior of an nxn AUD may vary according to the nature of logical components to be used and at the same time be too numerous to mention. Therefore, in this chapter we concentrate on the logical aspect of the circuit design, such as decomposition of the table describing an nxn AUD, possible ways of extending the circuit for higher values of n and the number of delay elements to be inserted in the circuit to ensure proper behavior of the AUD, etc. . Both the cases when the AUD constructed under (i) all input changes allowed (ii) single input changes only are considered. AUD with single input changes is studied in detail. Using single error correcting codes, we develop a systematic method of obtaining a reduced table of an nxn AUD with single input changes only. A brief account of pulse-controlled design applicable to both single input and multiple input changes is also discussed.

3.1 Single-error-correcting codes

As single-error-correcting (SEC) codes [16] form the basis for the development of a special class of codes later, we shall first review briefly certain relevant features of these SEC codes. A code B_n of word length n is SEC iff, for all $x, y \in B_n$, the (Hamming) distance $d(x,y) \geq 3$. We now proceed to present certain known results for the sake of completeness.

Lemma 3.1 : For every $n = 2^q$ a positive integer, there exists a SEC code B_{n-1} with the number $N(B_{n-1})$ of words equal to 2^{n-1-q}.

Proof : - See Reference [29].

Lemma 3.2 : Let $n = 2^q$ and let r be a positive integer such that $1 \leq r < 2^{q-1}+$. Then there exists a SEC code B_{n-1-r} with $N(B_{n-1-r})$

$$\geq \frac{2^{n-1-r}}{2^{[\log_2 (n-r)]}} = \frac{2^{n-1-r}}{2^{\log_2 n}} \quad , \text{ where } [\log_2 (n-r)] \text{ is the smallest integer } > \log_2 (n-r) \ .$$

Proof : From Lemma 3.1 we have

$$N(B_{n-1}) = 2^{n-1-q} = \frac{2^{n-1}}{2^q} = \frac{2^{n-1}}{n}$$

which incidentally is the Hamming bound.

Thus

$$N(B^*_{n-1}) = \frac{2^{n-1}}{n} \quad . \tag{1}$$

where * indicates a code with maximum possible number of words.

Now Plotkin [30] has shown that

$$N(B^*_{n-2}) \geq \frac{1}{2} \quad N(B^*_{n-1}) \ .$$

Applying this rule successively we get

$$N(B^*_{n-1-r}) \geq \frac{1}{2^r} \quad N(B^*_{n-1}) \ .$$

Using (1) we further have

$$N(B^*_{n-1-r}) \geq \frac{1}{2^r} \frac{2^{n-1}}{n} = \frac{2^{n-1-r}}{n}$$

$$= \frac{2^{n-1-r}}{2^{\log_2 n}}$$

$$= \frac{2^{n-1-r}}{2^{\lceil \log_2 (n-r) \rceil}} .$$

Theorem 3.1 : For every $n-1$ there is a SEC code with

$$N(B_{n-1}) = \frac{2^{n-1}}{2^{\lceil \log_2 n \rceil}} .$$

Proof : - Follows directly from Lemmas 3.1 and 3.2.

3.1.1 Construction of SEC Codes for any Length.

When $n-1$ is such that $n = 2^q$ then a SEC code B^*_{n-1} can easily be constructed using parity check equations [30]. The code so formed will have $\frac{2^{n-1}}{n}$ words and thus meets the Hamming bound.

From this code we can easily derive codes for shorter lengths $n-1-r$, $1 \leq r < 2^{q-1}$, in two ways which are really equivalent. The first method is as follows : In the parity check equations for $n-1$, delete any one variable x_i wherever it occurs. The resulting equations are parity checks for length $n-2$. If in these equations, a further variable x_j is deleted, we have now parity checks for length $n-3$. Clearly we can go on repeating the process till we reach the required length.

The second method is as follows : Suppose we simply take out of a B^*_{n-1}, only those words which, in some one column, have all

1's (0's) . These words form a B_{n-2} code after we delete the column containing all 1's(0's) . The process can be repeated till we reach the desired length $n-1-r$.

The above mentioned methods always give

$N(B_{n-1-r}) = \frac{2^{n-1-r}}{2^{\lceil \log_2 n \rceil}}$. This number is the same as the Hamming bound for $r = 0$. For other lengths it is the best one can do by way of linear codes ; Table 1 [34] gives the sizes of the best known codes as against best linear codes for certain lengths. In Table 1 subscript b indicates the best known.

m	3	4	5	6	7	8	9	10
$N_b(B_m)$	2	2	4	8	16	20	38	68
$N(B_m)$	2	2	4	8	16	16	32	64

Table 1

3.2 A Special Class of Codes

Definition 3.1 : Any two words are said to satisfy $d_{1,\geq 3}$ property iff, their Hamming distance is either 1 or ≥ 3.

Definition 3.2 : Any code with word length n, which satisfies the $d_{1,\geq 3}$ property, will be designated by C_n.

First let us consider the case when $|w_i \oplus w_j| = 1$ for all $i \neq j$. This case is trivial in that a code satisfying this property has exactly two words.

Second, when a code is such $|w_i \oplus w_j| \geq 3$ for all $i \neq j$, the situation is one of SEC codes which have been dealt with extensively in literature [7,8,16,27] .

Therefore, hereafter we shall assume that in every C_n we have some pairs of words satisfying $|w_i \oplus w_j| = 1$ and others $|w_i \oplus w_j| \geq 3$.

Theorem 3.2 : Let p be the number of pairs of words in a code C_n such that the words in each pair have mutual distance of 1. Let $r = N(C_n) - 2p$, where $N(C_n)$ is the number of words in C_n. Then $N(C_n) \not> \frac{2^n - r}{n}$.

Proof :- The p pairs of words "use up" $2pn$ words. The remaining r words "use up" $r(n+1)$ words. But $2pn + r(n+1) = 2n \frac{N(C_n) - r}{2} + r(n+1) = r + n N(C_n)$ should be $\not> 2^n$. This also means that $N(C_n) \not> \frac{2^n - r}{n}$.

This completes the proof.

Let $\beta(r) = \frac{2^n - r}{n}$. Clearly as r decreases, or equivalently p increases, the bound $\beta(r)$ increases. We shall show next that when $r = 0$ and $n - 1$ is a maximal length the bond $\beta(0)$ can indeed be reached.

Definition 3.3 : $C_n^{(p)}$ is a code in which $r = 0$, or equivalently $p = \frac{N(C_n)}{2}$.

Theorem 3.3 : For any n there exists a code $C_n^{(p)}$ such that $N(C_n^{(p)}) = 2\, N_{max}(B_{n-1})$, where $N_{max}(B_{n-1})$ is the number of words in the largest SEC code of length $n - 1$.

Proof : Let us suppose that we have a B_{n-1} SEC code with the maximum number of words. Without losing generality we can assume that

B_{n-1} has the zero word since any code can be converted to have zero word by adding an appropriate word to all the words of the code. Now suppose we append a 0 to every word of B_{n-1} that has an even weight and a 1 to every word that has odd weight. The resulting code D_n has a word length of n, a minimum distance 4 and minimum non-zero weight of 4. Also, $N(D_n)$ is the same as $N(B_{n-1})$.

Next let us get $\hat{D}_n$ by adding modulo 2 a word of weight 1 to each of the words of D_n. Clearly $N(\hat{D}_n) = N(D_n)$. Also, $N(D_n \cup \hat{D}_n) = 2N(D_n) = 2N(B_{n-1})$. Also, in the code $D_n \cup \hat{D}_n$ the Hamming distance is 1 or ≥ 3. Thus $D_n \cup \hat{D}_n = C_n^{(p)}$. Thus $N(C_n^{(p)}) = 2N_{max}(B_{n-1})$ if B_{n-1} is the largest SEC code.

This completes the proof.

Corollary 3.1 : For any n there exists a code $C_n^{(p)}$ such that $N(C_n^{(p)}) = 2^{n-k}$, $k \triangleq [\log_2 n]$.

Proof : The proof follows directly from Theorem 3.3 since we can always form a linear (group) code B_{n-1} with $N(B_{n-1}) = \frac{2^{n-1}}{2^{[\log_2 n]}}$.

From Corollary 3.1 it is obvious that, when $n = 2^q$, we have $N(C_n^{(p)}) = \frac{2^n}{n}$ which is the bound referred to earlier.

Lemma 3.3 : If B_{n-1} is a group, then D_n is a group and $\hat{D}_n$ is a coset of D_n.

3.3 Asynchronous Unit Delays :

Definition 3.4 : An nxn asynchronous unit delay (nxn AUD) is an asynchronous sequential circuit with n-binary inputs $x_1, x_2, \ldots, x_n$ and n binary outputs $z_1, z_2, \ldots, z_n$. At any time t the value of the output n-tuple is equal to the value of the input n-tuple prior to time $(t - \delta)$, where $(t - \delta)$ is the instant of time when the input n-tuple was last changed. In symbols :

$$Z(t) = X(t - \delta) = D(X(t)),$$

where $Z(t) = (z_1(t), z_2(t), \ldots, z_n(t))$,

$X(t) = (x_1(t), x_2(t), \ldots, x_n(t))$,

and D denotes the delay operator. Define $D^0(X(t)) = X(t)$ and $D(D(X(t)) = D^2(X(t)) = (X(t - \delta_1 - \delta_2))$ where the input changed at time $(t - \delta_1 - \delta_2)$ and $(t - \delta_1)$ etc. .

Note that the instant of time t can be chosen anytime after the inputs have attained steady state value following an input change.

In the general case all input changes are allowed in an AUD. However, the case where only single input changes are allowed will be of special interest. We shall next describe the flow tables of the two types of AUD's.

Case 1, All Input Changes Allowed :

Assume that the AUD is stable in some input column. The previous input can be any one of the $2^n - 1$ possible inputs ; hence there must be at least $2^n - 1$ states in each column, each state having the output corresponding to the previous input. In its normal primitive flow table there must be at least $2^n(2^n - 1)$ stable states or rows. This number of states is also sufficient, for suppose the present total state is the stable state (Y_i, X_j). If the input changes to X_k then $M(Y_i, X_k)$ is a stable state Y_ℓ in column X_k whose output is X_j. Thus in the normal primitive flow table all transitions have the form shown in Figure 3.1, where stable states are denoted by circled entries. Notice

that there are no unspecified next state entries. It is also easy to verify that this table is reduced for $n \geq 2$. As an example, Figure 3.2 shows the general 2x2 AUD.

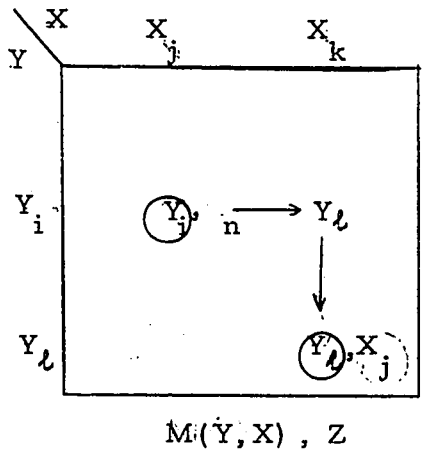


Figure 3.1

	x_1x_2 00	01	11	10
1	(1), 01	4 , -	7 , -	10 , -
2	(2), 11	4 , -	7 , -	10 , -
3	(3), 10	4 , -	7 , -	10 , -
4	1 , -	(4), 00	8 , -	11 , -
5	1 , -	(5), 11	8 , -	11 , -
6	1 , -	(6), 10	8 , -	11 , -
7	2 , -	5 , -	(7), 00	12 , -
8	2 , -	5 , -	(8), 01	12 , -
9	2 , -	5 , -	(9), 10	12 , -
10	3 , -	6 , -	9 , -	(10), 00
11	3 , -	6 , -	9 , -	(11), 01
12	3 , -	6 , -	9 , -	(12), 11

Figure 3.2 - General 2x2 AUD.

Case 2, Single Input Changes Only :

A simpler flow table for an n x n AUD is obtained, if it is assumed that only one input can change at a time. Each column must have now n stable states corresponding to n allowed previous inputs. The normal primitive flow table has $n \cdot 2^n$ rows as is illustrated in Figure 3.3 for n = 2. Because of the presence of unspecified next state entries, it is possible to obtain a reduced flow table for an n x n AUD. For the case n = 2 we obtain a table of Figure 3.4.

upon the inputs prior to X_i. Therefore $A \neq B$.

(b) If $d(X_i,X_j) = 1$, then $M(A,X_j)$ and $M(B,X_i)$ must be specified. If $M(B,X_i) = A_1 \neq A$, then obviously the output associated with A_1 is X_j which must be different from the output associated with A. Hence $A \not\simeq B$. This implies that we must have $M(B,X_i) = A$. By the same arguments $M(A,X_j) = B$. Now $A \simeq B$ because any input X_k adjacent to X_i cannot be adjacent to X_j and hence for any such X_k, if $M(A,X_k) = C$ then $M(B,X_k) =$ " - " or vice versa. Hence there is no column in F where there are conflicting entries.

(c) if $d(X_i,X_j) = 2$, then for some X_ℓ adjacent to both X_i and X_j, $M(A,X_\ell) = C$ and $M(B,X_\ell) = D \not\simeq C$ because C and D are supposed to remember X_i and X_j respectively. So there are conflicting entries in X_ℓ. Hence $A \not\simeq B$.

(d) Let C be any state reachable from A by changing input X_i to X_k and let D be another state reachable from B by changing X_j to X_ℓ. Since $d(X_i,X_j) \geq 3$, $d(X_i,X_k) = 1$, $d(X_j,X_\ell) = 1$, therefore $d(X_k,X_\ell) \geq 1$. Hence again there are no conflicting entries in any input columns of F, and $A \simeq B$. This completes the proof of the Lemma.

Corollary 3.2 : Let X_i,X_j,X_k,X_ℓ be four distinct columns of F of an n x n AUD and let A, B, C, D be distinct states stable in these columns respectively. If $d(X_i,X_j) = 1$, $d(X_k,X_\ell) = 1$, $d(X_i,X_k) \geq 3$, $d(X_i,X_\ell) \geq 3$, $d(X_j,X_k) \geq 3$, $d(X_j,X_\ell) \geq 3$, and $M(A_i,X_j) = B$, $M(B_j,X_i) = A$, $M(C,X_\ell) = D$ and $M(D,X_k) = C$, then A, B, C, D are compatible.

Because of Corollary 3.1 of Theorem 3.3 and Lemma 3.4, the size of the largest maximal set of compatible states S_c in the primitive table F of an AUD is bounded by

$\frac{2^n}{n}$. Hence the number of states in the reduced table F_R of the AUD cannot be $< \frac{n\,2^n.n}{2^n} = n^2$. We shall show that this bound is achievable for $n = 2^q$. For other values of n, we shall prove that bound $n.2^{[\log_2 n]} = n \cdot 2^k$ is achievable.

Before going into the general discussion about the use of the code $C_n^{(p)}$, consider a primitive flow table F of a 4x4 AUD. F has 2^4 columns, 4×2^4 states and each input column has 4 stable states. To obtain F_R, step by step let us proceed as follows :

1. Construct parity check equations for a SEC Code B_3.

$$x_1 \oplus x_2 = 0$$
$$x_1 \oplus x_3 = 0 .$$

From these it is obvious that $B_3 = (000, 111)$.

2. According to Theorem 3.3 $D_4 = (0000, 1111)$

3. Now choose an input X_i such that it is adjacent to any one of the inputs in D_4. Since there are 4 such choices, we can form 4 distinct $\hat{D}_4$'s by adding X_i (mod.2 each time to all the words in D_4. Thus $\hat{D}_{4,1} = (0001, 1110)$, $\hat{D}_{4,2} = (0010, 1101)$, $\hat{D}_{4,3} = (0100, 1011)$ and $\hat{D}_{4,4} = (1000, 0111)$. Obviously $C_{4,1}^{(p)} = (D_4 \cup \hat{D}_{4,1}) = (0000, 1111, 0001, 1110)$, $C_{4,2}^{(p)} = (0000, 1111, 0010, 1101)$, $C_{4,3}^{(p)} = (0000, 1111, 0100, 1011)$ and $C_{4,4}^{(p)} = (0000, 1111, 1000, 0111)$ respectively. From Lemma 3.4 and Corollary 3.2 it is known that states of F stable in columns of $C_{4,i}^{(p)}$ for i = 1 to 4 form sets of 4 compatible states $S_{c,1}$ $S_{c,2}, S_{c,3}, S_{c,4}$ such that $S_{c,1} \cap S_{c,2} \cap S_{c,3} \cap S_{c,4} = \phi$, ϕ being an empty set. In doing so we "use up" all the states stable in the columns of D_4 and one state stable from the columns of $\hat{D}_{4,i}$.

4. From part (c) of Lemma 3.4 it is also clear that no two states stable in X_i, X_j are compatible if $d(X_i, X_j) = 2$. We choose an X_j such that it is at an exact distance of 2 from any one inputs $\in D_4$. For example X_j could be 0011 or 1100. Construct coset E_4 by adding X_j mod. 2, to all columns of D_4. Now again choose another 4 adjacent columns to any one column in E_4 and add one adjacent input mod. 2 to the inputs of E_4 each time to construct $\hat{E}_{4,1} = (0010, 1101)$, $\hat{E}_{4,2} = (0001, 1110)$, $\hat{E}_{4,3} = (0100, 1011)$ and $\hat{E}_{4,4} = (1000, 0111)$ and hence $C_{4,5}^{(p)} = (E_4 \cup \hat{E}_{4,1}) = (0011, 1100, 0010, 1101)$, $C_{4,6}^{(p)} = (0011, 1100, 0001, 1110)$, $C_{4,7}^{(p)} = (0011, 1100, 0100, 1011)$ and $C_{4,8}^{(p)} = (0011, 1100, 1000, 0111)$. Again, it is clear that the states stable in $C_{4,j}^{(p)}$ for $j = 4$ to 8 form sets of 4 compatible states $S_{c,5}, S_{c,6}, S_{c,7}$ and $S_{c,8}$ such that $S_{c,1} \cap S_{c,2} \cap \ldots \cap S_{c,8} = \phi$. In doing so we have again "used up" all the 4 states stable in columns of E_4 and one stable state from the columns of $E_{4,j}$.

5. Construct cosets by adding 1001 and 0101 Mod. 2 to all the columns of D_4 and repeat step 4 to obtain $C_{4,9}^{(p)}$, $C_{4,10}^{(p)}$, $\ldots C_{4,16}^{(p)}$. It is clear that all the 16 sets of compatible states S_c's thus obtained are mutually disjoint and we have "used up" all the states of F.

The reduced table F_R of 4x4 AUD thus obtained is shown in Figure 3.5.

From this example it is clear that if B_{n-1} is constructed with parity check equations, it will be a linear code and such a code will have 2^a words. Therefore the number of compatible states in each set will be 2^{a+1}. All these sets of 2^{a+1} states will be mutually disjoint. Naturally F_R thus obtained, will also have 2^{a+1} stable states in each row.

Lemma 3.5 : Let D_n be the largest group code in the set of input columns of a primitive table F of an nxn AUD. Let S_c be a set of

$x_1x_2x_3x_4$	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
1	(1)	(1)	5	2	7	4	6	3	5	2	(1)	(1)	6	3	7	4
2	(2)	1	8	(2)	11	4	12	3	8	(2)	(2)	1	12	3	11	4
3	(3)	1	10	2	14	4	16	(3)	10	2	(3)	1	16	(3)	14	4
4	(4)	1	9	2	13	(4)	15	3	9	2	(4)	1	15	3	13	(4)
5	1	(5)	(5)	8	7	9	6	10	(5)	8	1	(5)	6	11	7	9
6	1	(6)	5	12	7	15	(6)	16	5	12	1	(6)	(6)	16	7	15
7	1	(7)	5	11	(7)	13	6	14	5	11	1	(7)	6	14	(7)	13
8	2	5	(8)	(8)	11	9	12	10	(8)	(8)	2	5	12	10	11	9
9	4	5	(9)	8	13	(9)	15	10	(9)	8	4	5	15	10	13	(9)
10	3	5	(10)	8	14	9	16	(10)	(10)	8	3	5	16	(10)	14	9
11	2	7	8	(11)	(11)	13	12	14	8	(11)	2	7	12	14	(11)	13
12	2	6	8	(12)	11	15	(12)	16	8	(12)	2	6	(12)	16	11	15
13	4	7	9	11	(13)	(13)	15	14	9	11	4	7	15	14	(13)	(13)
14	3	7	10	11	(14)	13	16	(14)	10	11	3	7	16	(14)	(14)	13
15	4	6	9	12	13	(15)	(15)	16	9	12	4	6	(15)	16	13	(15)
16	3	6	10	12	14	15	(16)	(16)	10	12	3	6	(16)	(16)	14	15

Fig. 3.5 - Reduced state table of 4 x 4 AUD.

compatible states constructed from the states of F which are stable $\in C_n^{(p)}$. Then the reduced table F_R has $n.2^k$ rows, k being $[\log_2 n]$.

Proof : Since D_n is the largest group code, therefore the number of input columns in $C_n^{(p)}$ is $N(D_n \cup D_n) = 2N(D_n) = 2^{n-k}$ (from Corollary 3.1 of Theorem 3.3). Thus a set of compatible states S_c will have 2^{n-k} states. From the previous example, it is clear that one can construct $N(S_c)$ mutually disjoint sets of 2^{n-k} compatible states. Therefore the number of rows in

$$F_R = \frac{n.2^n}{2^{n-k}} = n.2^k.$$

This proves the Lemma.

General Remarks : Consider the case when B_{n-1} is not a group code, but is the best code with maximum possible number of words. Therefore $N(B_{n-1}) \neq 2^q$, but has a number of words $> 2^q$. Hence $N(C_n^{(p)}) = N(D_n \cup \hat{D}_n) = 2N(B_{n-1})$ will not divide the total number of states $n.2^n$ in the primitive table F. Also, the reader must note that, all the sets S_c's with $2N(B_{n-1})$ states will not be mutually disjoint. However, by proceeding in a systematic way according to the construction suggested in Lemma 3.5, we can always choose at least $[31(a)]\ n + n(n-1) = n^2$ sets S_c's with $2N(B_{n-1})$ states which will be mutually disjoint. This is because D_n is a minimum-distance-four code, so we can have n distinct $\hat{D}_n$ by adding mod. 2 the n words of weight 1. Thus we can construct n S_c's corresponding to n. $C_n^{(p)}$'s. Also, by choosing $(n-1)$ words of weight 2 with exact mutual distance 2, we can construct $(n-1)$ $\hat{E}_n$'s and hence $n(n-1)$ S_c's (see steps 4 and 5, in the example of 4x4 AUD). Of course, there is a chance that we may obtain a few more sets by actually

trying all the possibilities for particular values of n. Thus the number n^2 is only a lower bound. Therefore from the remaining states $n.2^n - n^2\, 2N(B_{n-1})$ we may have to form many S_c's with significantly lesser number of states compared to $2N(B_{n-1})$. Thus we may or may not be able to obtain an F_R with lesser number of states compared to F_R when B_{n-1} is a group. Lastly, the most important factor is that there are no systematic methods known to obtain best non-group SEC codes.

For example, consider the flow table of 9x9 AUD. The best non-group code B_8 known [34] has 20 words. Thus $N(C_9^{(p)}) = 40$. Hence the largest set of compatible states has 40 states. In this case we can guarantee to have 81 mutually disjoint sets of 40 compatible states only. Therefore, we may be left with $9.2^9 - 81.40 = 1368$ states in F. Out of 1368 states we shall have to construct many compatible sets with much lesser number of states compared to $2N(B_8)$. Thus the table F_R obtained with $81 + \alpha$ rows obtained in this fashion may or may not have lesser number of rows from F_R with 144 rows obtained by group code B_8 with $N(B_8) = 16$. The reader must note that for larger values of n the group code approach leads to a far simpler systematic method of reducing the flow table of an AUD compared with any other method available.

Definition 3.5 : Any two input columns X_j, X_k in F_R of an AUD are said to be distinct iff, for at least one state Y_i in F_R, $M(Y_i, X_j) = Y_\ell$ and $M(Y_i, X_k) = Y_m$ and $\ell \neq m$. Otherwise X_j and X_k are identical.

Theorem 3.4 : A reduced table F_R with $n.2^k$ rows, $k = [\log_2 n]$, of an AUD has the following characteristics ,

(i) F_R has n stable states per column.

(ii) F_R has 2^{n-k} stable state entries in each row.

(iii) The sets of next state entries in X_i, X_j are disjount if $d(X_i, X_j) = 2$.

(iv) Let X_i be some input column in F_R in which a set of states $Y = \{ Y_1, Y_2 \ldots Y_n \}$ is stable.

Let $X_1, X_2 \ldots X_n$ be the columns adjacent to X_i. Then there is one and only one adjacent X_j in which a unique $Y_i \in Y$ is also stable.

(v) Input column X_i, X_j and X_m in F_R are identical if $d(X_i, X_j) = 4q$ $d(X_i, X_m) = 4q$ and $d(X_j, X_m) = 4q$ where $q = 0, 1, 2 \ldots$

(vi) F_R has 2^{k+1} distinct input columns.

Proof :

(i) Obvious from Lemma 3.4, Part (a).

(ii) Obvious from Lemma 3.5.

(iii) Obvious from Lemma 3.4, Part (c).

(iv) Obvious from Lemma 3.4, Part (b).

(v) From parts (iii) and (iv) of this theorem it is clear that X_i, X_j with $d(X_i, X_j) = 1$ and $d(X_i, X_j) = 2$ are distinct.

If $d(X_i, X_j) = 3$, then every state stable in $X_i \in F$ can be compatible to every state stable in X_j. But F_R is constructed by forming sets of compatible states according to Corollary 2 of Lemma 3.4. Under such conditions a state compatible with a state stable in X_k with $d(X_i, X_k) = 1$ cannot be mutually compatible with any state stable in X_j. Therefore X_i, X_j are distinct in F_R.

If $d(X_i, X_j) = 4q$ then every state stable in X_i is compatible with every state stable in X_j of F. This does not violate the condition of compatibility of other states stable

in X_k and X_ℓ with $d(X_i, X_k) = 1$ and $d(X_j, X_\ell) = 1$, because $d(X_i, X_\ell) \geq 3$ and $d(X_j, X_k) \geq 3$. Hence $X_i, X_j \in F_R$ are identical. Similarly it can be proved that X_i, X_m and X_j, X_m are identical. Therefore X_i, X_j, X_m are identical.

(vi) Let $S_x^{(1)}$ be the set of input columns of F_R such that any two $X_i, X_j \in S_x^{(1)}$ have $d(X_i, X_j) = 2$. We know that each column of F_R has n stable states and the set of stable states in X_i, X_j are mutually disjoint. Now if p_1 is the number of such columns $\in S_x^{(1)}$, then $n.p_1 = n.2^k$, therefore $p_1 = 2^k$.

Similarly, if we choose some X_k adjacent to $X_i \in S_x^{(1)}$ and construct another set of input columns $S_x^{(2)}$ such that any two columns $X_k, X_\ell \in S_x^{(2)}$ have $d(X_k, X_\ell) = 2$. Now if p_2 is the number of such columns $\in S_x^{(2)}$, then $np_2 = n.2^k$, hence $p_2 = 2^k$.

Therefore the number of distinct input columns in $F_R = p_1 + p_2 = 2^{k+1}$.

This completes the proof of the theorem.

3.4 Circuit Realizations of AUD's

Theorem 3.5: A reduced table F_R of $n.2^k$ rows of an AUD has 2k binary S.P. partitions whose product $\pi_1.\pi_2.,\ldots.\pi_{2k} = 0$.

Proof: Let $S_x^{(1)}$ and $S_x^{(2)}$ be the sets of inputs of F_R as defined earlier in the proof of the part (vi) of Theorem 3.4. We know that each column $\in S_x^{(1)}$ has a set of n stable states, all these sets being mutually disjoint and their set union equal to the set of states of F_R. It is clear that one can construct many non-trivial S.P. partitions on the states of F_R such that each block of any such partition has n states

which are stable in some 2^{k-1} columns of $S_x^{(1)}$. In order that their product is a 2^k block S.P. partition $\pi^{(1)}$ one must choose at least k binary S.P. partitions.

Similarly with the help of states stable in the columns of $S_x^{(2)}$ one can again choose minimally k binary S.P. partitions such that their product is again a 2^k block partition $\pi^{(2)}$. But since columns of $S_x^{(1)}$ and $S_x^{(2)}$ have only one common stable state, the product $\pi^{(1)} \cdot \pi^{(2)} = 0$.

This completes the proof of the theorem.

Theorem 3.6 : The flow table of an n x n AUD with single input changes has no essential hazards.

Proof : Let F be the primitive flow table of an n x n AUD, and assume an essential hazard is present. Note that both $M_S(Y_i, X_k)$ and $M_S(Y_i, X_k X_j X_k)$ are stable in the same column X_k and the output associated with both these states must be X_j, because both states must remember that the previous input was X_j. Also all transitions from these two states under any X_ℓ must go to the same state stable in X_j with output X_k. Hence we have a contradiction, for the two states $M_S(Y_i, X_k)$ and $M_S(Y_i, X_k X_j X_k)$ are identical in the primitive table.

As was shown by Unger [33] the absence of essential hazards means that the flow table can be realized by a circuit without any inertial delays in any of its feedback loops.

However, there may be transient hazards in the outputs of the AUD. In order to avoid these it may be necessary to introduce inertial delays in some output leads. Clearly, transient incorrect outputs must not be allowed if the AUD outputs are to feed another circuit.

Illustrative Example : Consider the reduced flow table F_R of a 4 x 4 AUD of Figure 3.5. It has 8 distinct input columns. With the help of these 8 columns one can form a number of binary S.P. partitions on the set of states of F_R. For example the partitions given below are binary S.P. partitions whose product is equal to 0.

$$\pi_1 = \{\overline{1,2,3,4,5,8,9,10} \; ; \; \overline{6,7,11,12,13,14,15,16}\}$$

$$\pi_2 = \{\overline{1,2,3,4,7,11,13,14} \; ; \; \overline{5,6,8,9,10,12,15,16}\}$$

$$\pi_3 = \{\overline{1,2,5,6,7,8,11,12} \; ; \; \overline{3,4,9,10,13,14,15,16}\}$$

$$\pi_4 = \{\overline{1,4,5,6,7,9,13,15} \; ; \; \overline{2,3,8,10,11,12,14,16}\}$$

The reader can easily verify by mere inspection that 4 is the minimum possible number of binary partitions whose product can be equal to zero. Therefore the state behavior of 4 x 4 AUD can be realized by a parallel connection of four 2-state submachines $M\pi_1$, $M\pi_2$, $M\pi_3$ and $M\pi_4$ corresponding to binary partitions π_1, π_2, π_3 and π_4 are shown in Figure 3.6. Since these are 2-state machines, hence each of these is realizable by a single feedback loop circuit. The feedback loop need not contain any extra delay element [Theorem 3.6] . The proper outputs are obtained by additional combinational logic as shown in Figure 3.7. In the worst case, the outputs may have transient hazards. In such a situation inertial delays must be inserted in the output leads before tapping the outputs. It is obvious that such a realization is automatically critical race free. These four 2-state machines are also realizable by a feedback free connection of some combinational logic and 4 S-R flip-flops as shown in Figure 3.8.

Theorem 3.5 implies that any n x n AUD with single input changes only can be realized by a parallel connection of 2k n-input 2-state machines followed by combinational logic for outputs, where

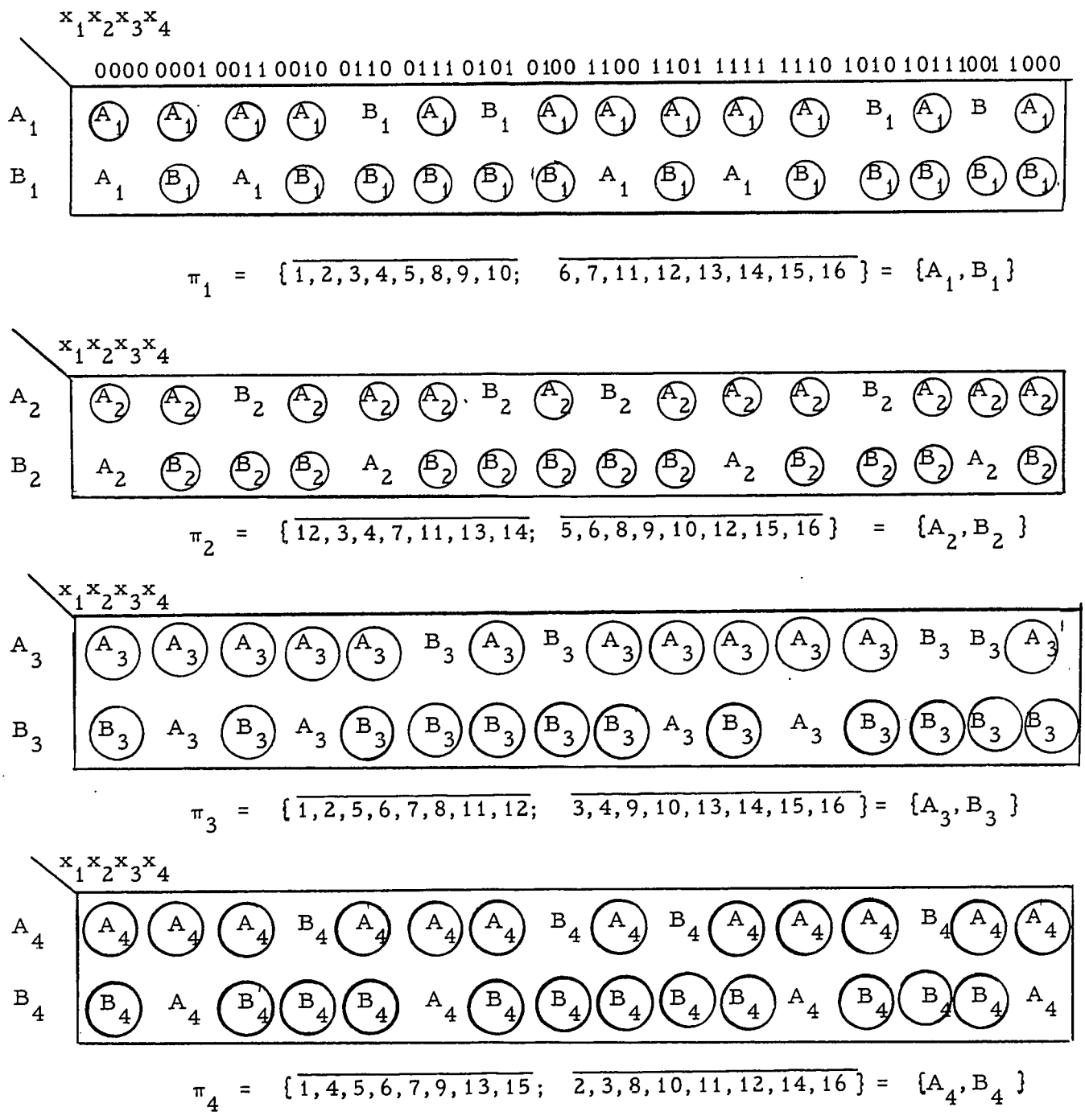

x1x2x3x4
0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000
A1
B1
π1 = {1, 2, 3, 4, 5, 8, 9, 10; 6, 7, 11, 12, 13, 14, 15, 16} = {A1, B1}
x1x2x3x4
A2
B2
π2 = {12, 3, 4, 7, 11, 13, 14; 5, 6, 8, 9, 10, 12, 15, 16} = {A2, B2}
x1x2x3x4
A3
B3
π3 = {1, 2, 5, 6, 7, 8, 11, 12; 3, 4, 9, 10, 13, 14, 15, 16} = {A3, B3}
x1x2x3x4
A4
B4
π4 = {1, 4, 5, 6, 7, 9, 13, 15; 2, 3, 8, 10, 11, 12, 14, 16} = {A4, B4}

Figure 3.6.

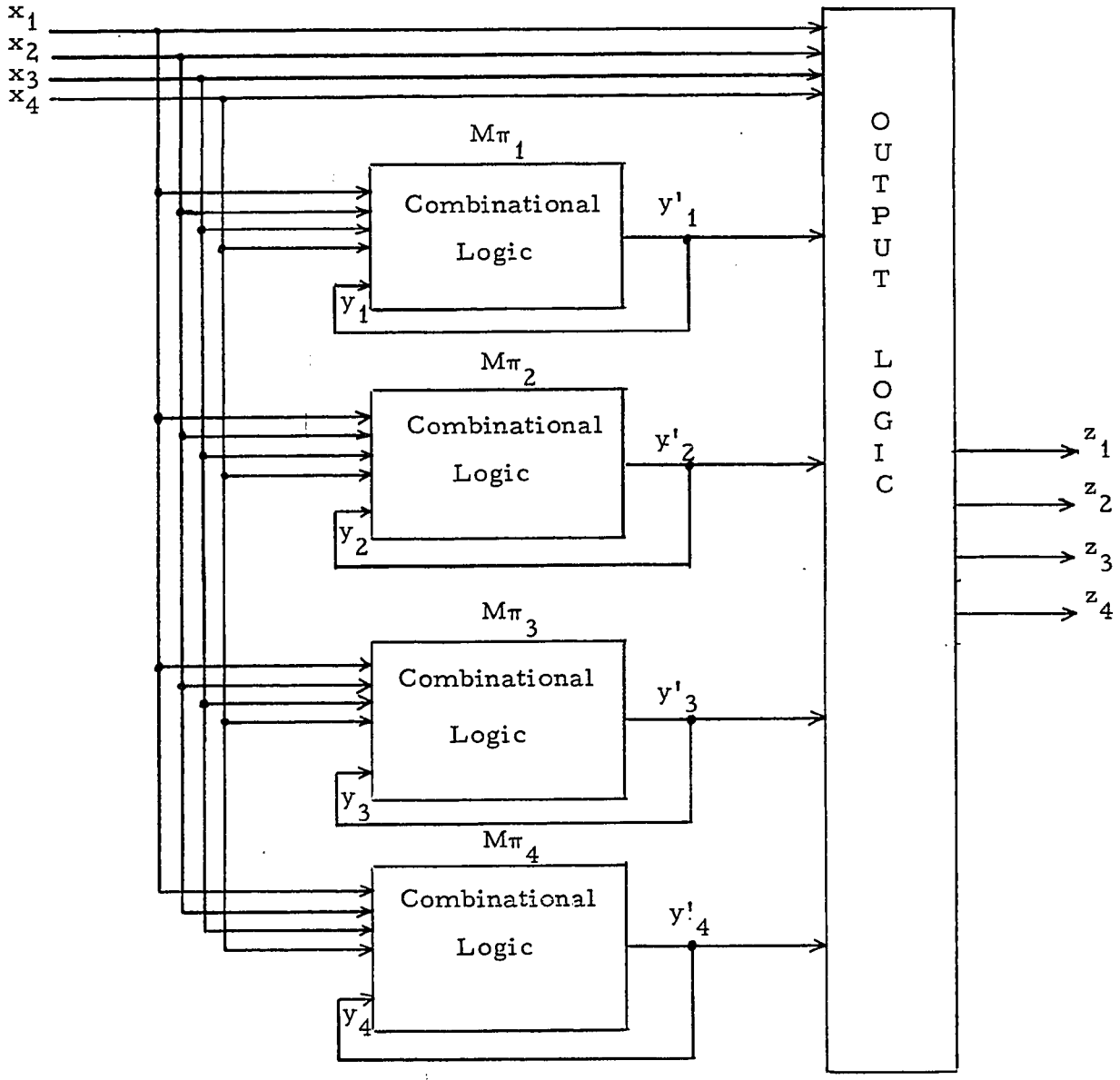
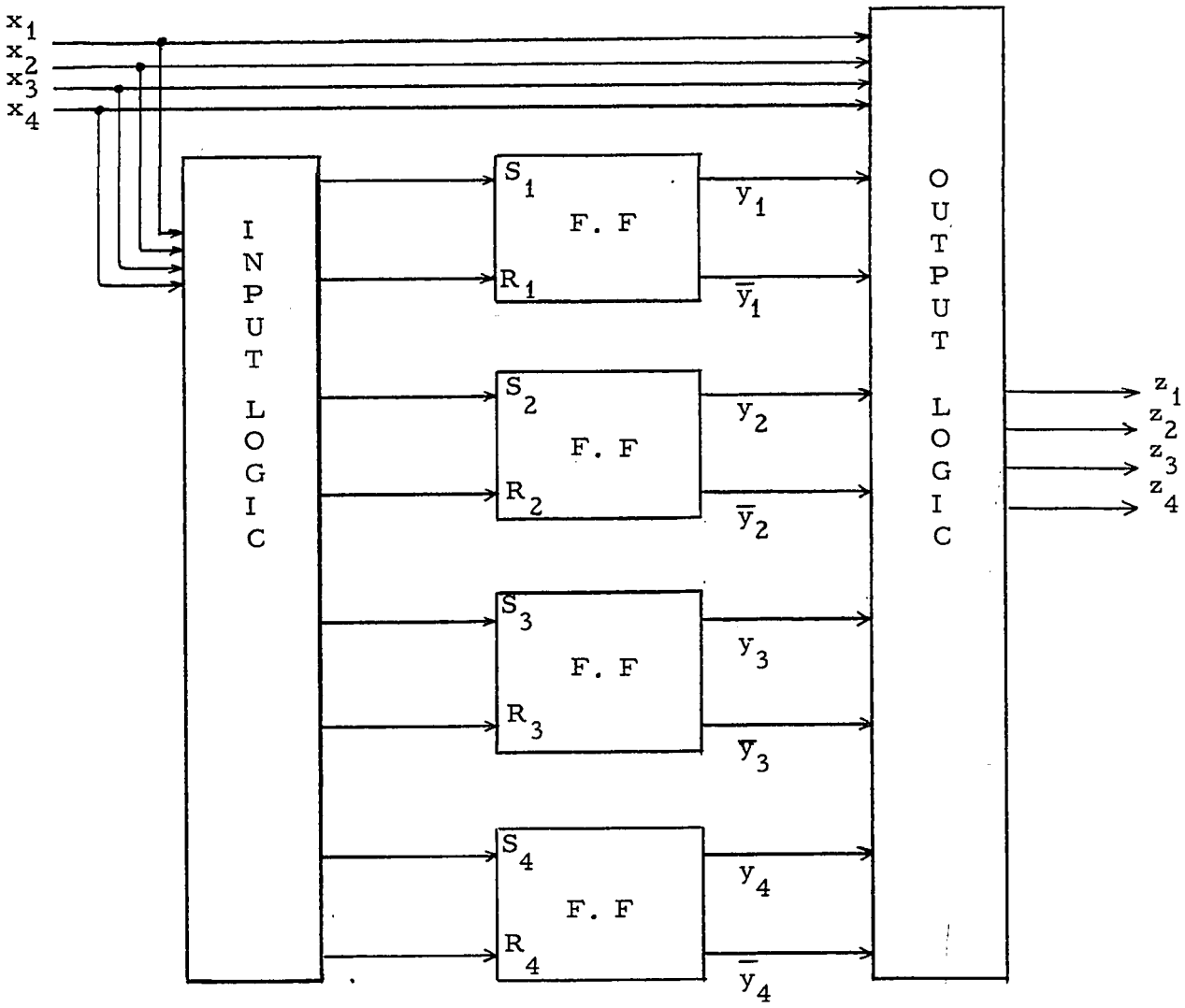


Figure 3.7 - 4x4 AUD circuit realization



$$S_1 = x_1\bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_2\bar{x}_3x_4 + \bar{x}_1x_2x_3\bar{x}_4$$

$$R_1 = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4 + x_1x_2\bar{x}_3\bar{x}_4 + x_1x_2x_3x_4$$

$$S_2 = x_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3x_4$$

$$R_2 = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2x_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3x_4 + x_1x_2x_3x_4$$

$$S_3 = x_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2x_3x_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2x_3x_4$$

$$R_3 = \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3x_4 + x_1x_2x_3\bar{x}_4$$

$$S_4 = x_1\bar{x}_2x_3x_4 + x_1x_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4$$

$$R_4 = \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_2x_3x_4 + x_1x_2x_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3\bar{x}_4$$

Figure 3.8 - Flip-Flop realization of 4x4 AUD.

$k \geq [\log_2 n]$. Obviously this realization is critical race free and no delay elements are necessary in the individual feedback loops of different smaller machines. It is interesting to note that each of these 2-state machines is also realizable by a feedback free connection of combinational logic followed by an S-R flip-flop. This is because the next state variable y'_i for any machine assumes a value 1 independently of present variable y_i or $y'_i = 1$ only if $y_i = 1$. Hence the S and R functions can be chosen as functions of inputs only..

There are many other well known asynchronous circuit design techniques available in the relevant literature [11, 12, 13, 21] for the realization of any asynchronous fundamental mode flow tables. Since all such techniques can be easily used to realize any n x n AUD with single input changes, we shall not pursue the subject any further.

Some remarks about Circuit Realization of AUD's with all input changes allowed :

Though the table of an n x n AUD, with all input changes allowed, also does not have any essential hazards, still its state behavior realization by a circuit without delay elements is not possible. This is because any input change, involving several input variables, will produce transient hazards at the next state variable y'_i terminals and these must be filtered out before they appear as present state variables y_i's and cause steady state hazards. Hence the presence of inertial delay elements in feedback loops is essential to remove these transients. Recent results due to Armstrong, Friedman and Menon [1] imply that a flow table with all input changes can be realized by a circuit having only one delay element. Hence one way to realize an AUD, with all input changes allowed, can be by using a source box [1] and a switching network without delay elements in its feedback loops. The source box is a circuit having only one delay element. Since the circuit design of the

source box varies with the number of input variables and gets quite cumbersome for higher values of n, it is not worthwhile to recommend this approach for the design of an n x n AUD, with all input changes from a practical point of view, specially when n is large.

Now we shall briefly discuss a new technique [2] to design an n x n AUD which is applicable to both single-input and multiple-input changes.

3.5 Pulse Controlled AUD Circuit :

The principle of operation of such a circuit is extremely simple. Since the present value of the output n-tuple of the circuit must be equal to the input n-tuple prior to the last input change, one can imagine a circuit where any change in the values of input shifts the original values of inputs from the input terminals to the output terminals. Therefore the circuit must be capable of performing the following operations :

(i) The circuit must detect a change in the inputs.

(ii) The circuit must retain the previous values for time D after the input has changed.

(iii) During this time D the stored values must shift from input to output terminals of the circuit.

These three operations can be easily performed by a circuit of Figure 3.9. First a change in the input values is detected by a change detector circuit and a pulse of duration t_p is produced, t_p being the minimum duration which can trigger the shift-circuit. There may be a time difference between the instant of input change and the time when the pulse appears, because of delays due to stray capacitances in the detector circuit. Let this delay be δ_c. In order that any two successive input changes can be identified by the detector as two distinct changes, these must be separated by a time interval $> \delta_c + t$. Otherwise, changes separated by time interval $\leq \delta_c + t_p$ will be recognized as a double

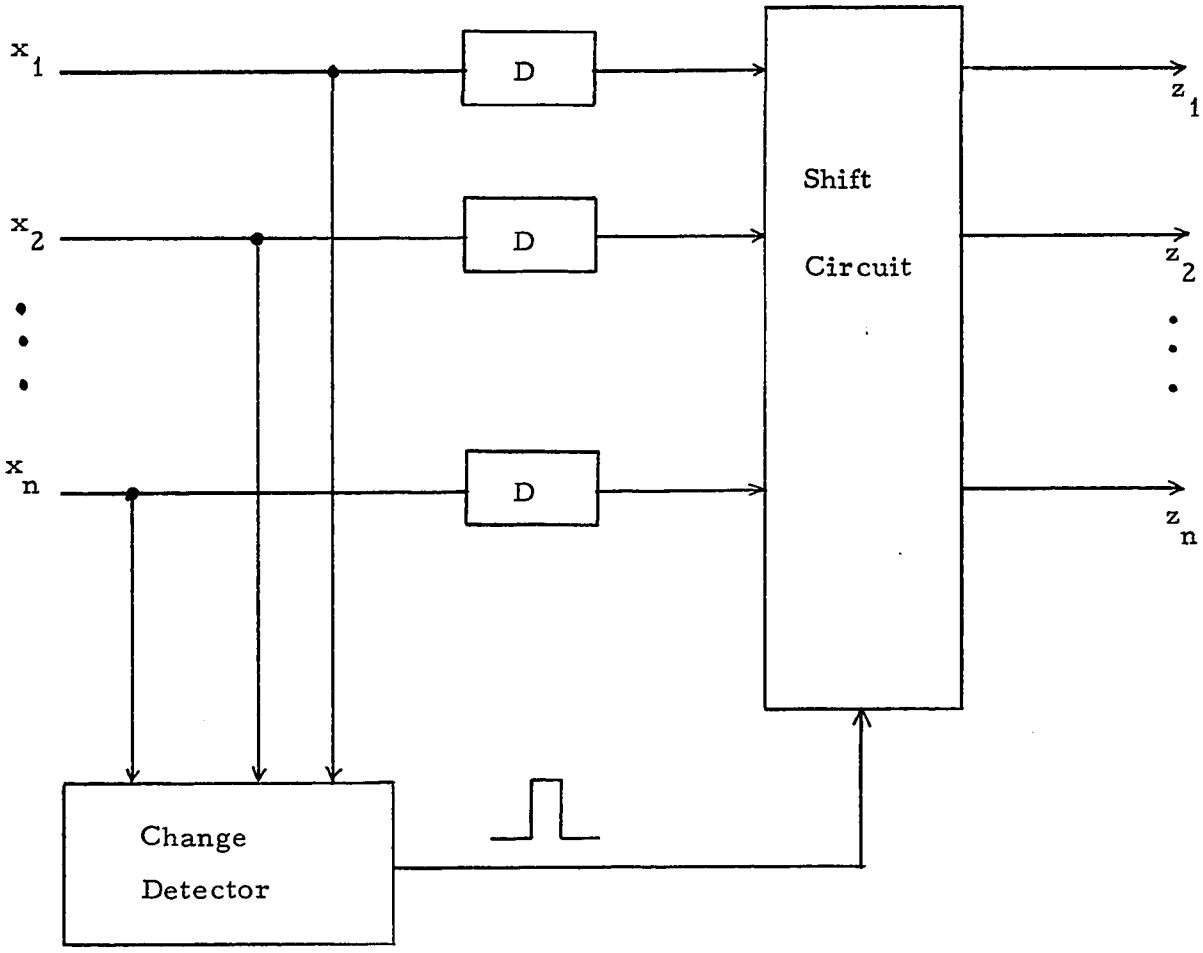


Figure 3.9

change in the values of input occuring at the instant of the first input change. The delays D in the input leads act as memory devices and retain the previous values of the input. For proper operation D must be $\geq \delta_c + t_p$. The shift circuit consists of n S-R flip-flops.

It is clear that this circuit has many advantages over others :

(i) The same circuit realizes n x n AUD's with single input changes and all input changes allowed.

(ii) The circuit can be extended for larger values of n.

(iii) For larger values of n this circuit is most economical.

(iv) The outputs are free of transient hazards.

(v) The propagation delay is uniformly the same for any input change.

Batra has designed and tested many circuits of AUD's for specific values of n. For details the reader is referred to his work [2].

Chapter 4

DEFINITE ASYNCHRONOUS SEQUENTIAL CIRCUITS

Introduction :

In a certain class of deterministic synchronous sequential machines the internal state depends only on a finite number of past inputs. Such machines are called definite. Methods of identifying definite machines have been studied by many authors [3, 26], and it is well known that every definite machine can be realized by a feedback-free circuit of unit delays and combinational gates. In this chapter we study a similar class of asynchronous machines in which the internal state is also a function of only a finite number of past input values. It will be shown that every asynchronous definite flow table is realizable by a feedback free connection of AUD's and combinational gates and every feedback-free circuit of AUD's and gates has a definite table. Thus a relation of definite tables to feedback-free circuits in synchronous theory has a very close analogy in the asynchronous case, if proper interpretation is used.

Aside from the fact that synchronous and asynchronous theories are brought closer together by this point of view, there are other advantages of considering asynchronous definite machines and their realizations using AUD's.

1. AUD is the only memory device required for the realization of the state behavior of any definite table. Although, for $n \geq 2$, the $n \times n$ AUD is a larger building block than a flip-flop and its use leads to simple overall structures of identical elements.

2. The secondary assignment problem for asynchronous definite tables is almost trivial.

3. With suitably designed unit delays and combinational circuits, the realization of a definite table is free from hazards and critical races.

4.1 Asynchronous Definite Tables

Definition 4.1 : Let F be the flow table of an FMA machine A with input restriction R. An input sequence $\bar{X}$ is valid for F iff (a) no two successive symbols in $\bar{X}$ are identical, and (b) $\bar{X}$ obeys the restriction R.

For example if A is a two input machine and R restricts the input changes to single variable changes only, then (00)(01)(11) is valid, but (01)(10) and (00)(00) are not valid for F.

Definition 4.2 : A flow table F of an FMA machine A is asynchronous definite iff there exists a $k \geq 0$ such that for all valid input sequences $\bar{X}$ with length $\ell(\bar{X}) \geq k$ and for all $Y_i, Y_j \in Y$,

$$M_S(Y_i, \bar{X}) = M_S(Y_j, \bar{X}),$$

where $M_S(Y_i, \bar{X})$ is the usual extension of the next state function M of A to sequences of arbitrary finite length except that it specifies the stable state reached when A is started in Y_i and $\bar{X}$ is applied.

The test of a given flow table F for the definite property is carried out in a manner very similar to the synchronous case. We define the input successor tree as follows [4, 5, 15, 21] :

1. The zeroth level of the tree contains one node corresponding to Y_S, the set of all internal states which are stable in some column.

2. Under input X_i there is a transition from the node Y_S to the first level node which contains all the stable states $M_S(Y_j, X_i)$, $Y_j \in Y_S$. There is one such node for each X_i.

3. The second level contains nodes corresponding to the successors of the first level states, reached by inputs which form valid input sequences considered from the zeroth level node.

The graph is terminated by one of the two rules :

(a) A level is reached in which each node corresponds to a single state or,

(b) The tree contains a path along which a node with more than one state appears twice.

It is clear that the graph is always finite. Furthermore if the graph is terminated by Rule (a), F is definite - if by Rule (b) it is not. It should be noted that this graph contains one more level than a similar graph drawn for the test of the definite property in synchronous case. However the number of input changes along any path on such a graph for asynchronous case is equal to number of levels in the graph for the synchronous case.

For example the table F of Figure 4.1 in which only single input changes are allowed yields a graph of Figure 4.2. Since the graph terminates by Rule (a), F is definite.

x_1x_2	00	01	11	10
1	(1), 0	(1), 1	-, -	4, -
2	1, -	(2), 0	-, -	3, -
3	4, -	2, -	-, -	(3), 1
4	(4), 1	1, -	-, -	3, -

Figure 4.1.

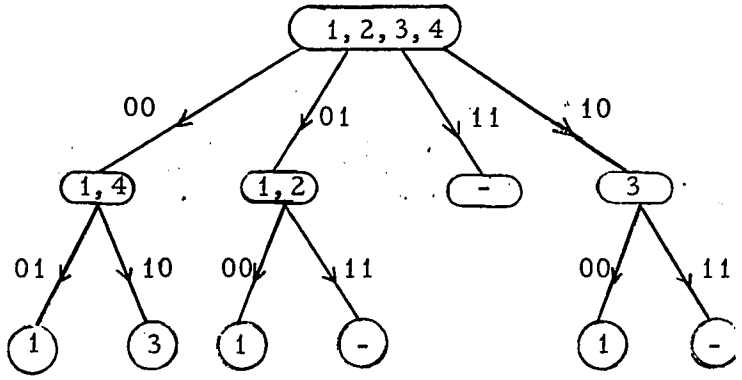


Figure 4.2

4.2 Feedback-Free Circuits

Definition 4.3 : An asynchronous circuit constructed with AUD's and combinational networks is feedback-free if starting at any point and passing through any number of AUD's and combinational networks it is not possible to return to the same point.

Note that we are not concerned with the feedback internal to the AUD, since the AUD is considered to be a basic building block.

Theorem 4.1 : Every asynchronous feedback-free circuit has a definite flow table.

Proof : Assume that any path from input to output has no more than k AUD's. If a valid sequence $\overline{X}$ of length $\geq k+1$ is applied, all the AUD outputs in the circuit are determined solely by $\overline{X}$, the initial conditions having been lost. Thus after $\overline{X}$ has been applied, the inputs and the outputs of each AUD in the circuit are determined. Now for a fixed input n-tuple, there is a unique internal state in the normal primitive flow table of an AUD with the given output. Thus the internal state of each AUD, and hence the internal state of the entire circuit are uniquely determined by $\overline{X}$. Therefore the state table of the circuit is asynchronous definite.

Theorem 4.2 : Let F be the flow table of a feedback-free asynchronous circuit. If F covers another flow table F', then F' is asynchronous definite.

Proof : It is clear that any sub-table of a definite table is definite. Now if F is definite its input successor graph terminates by Rule (a). Now if a number of states of F form a compatible and are merged to a single state for F', this can only lead to a possible faster termination of the graph of F' by Rule (a), but can never lead to Rule (b).

It follows therefore that a reduced table must be definite if it is realizable by a feedback-free circuit. We shall show now that every

definite table is realizable by a feedback-free circuit.

Definition 4.4 : An FMA machine is strongly connected if for every pair of (Y_i, Y_j) of states (stable in some columns) there exists a valid sequence $\bar{X}$ such that $M_S(Y_i, \bar{X}) = Y_j$.

Theorem 4.3 : Every definite strongly connected flow table F is realizable by a shift register of AUD's and combinational logic for outputs.

Proof : Since F is definite, its input successor tree terminates by Rule (a) at some level k. Since F is strongly connected, each internal state that is stable in some column must appear at least once in level k of the graph. Assign to each appearance of Y_i in level k the input sequence that takes the successor graph from zeroth level to that appearance of Y_i (Y_i may be represented by more than one state of the AUD register). Then a shift register of (k - 1) AUD's realizes the state behavior of F. The output can then be obtained by combinational circuits.

For example consider the table of Figure 4.3 with single input changes, and its tree in Figure 4.4. The resulting assignment is shown in Figure 4.5 and the circuit in Figure 4.6.

Example

	x_1x_2 00	01	11	10
1	(1), 00	(1), 00	9 , -	11 , -
2	(2), 00	6 , -	10 , -	(2), 10
3	(3), 11	1 , -	- , -	11 , -
4	(4), 11	6 , -	- , -	2 , -
5	3 , -	(5), 01	(5), 01	12 , -
6	1 , -	(6), 10	9 , -	- , -
7	3 , -	(7), 11	5 , -	- , -
8	4 , -	7 , -	(8), 11	(8), 01
9	- , -	5 , -	(9), 11	12 , -
10	- , -	7 , -	(10), 00	8 , -
11	2 , -	- , -	10 , -	(11), 11
12	4 , -	- , -	8 , -	(12), 10

Figure 4.3.

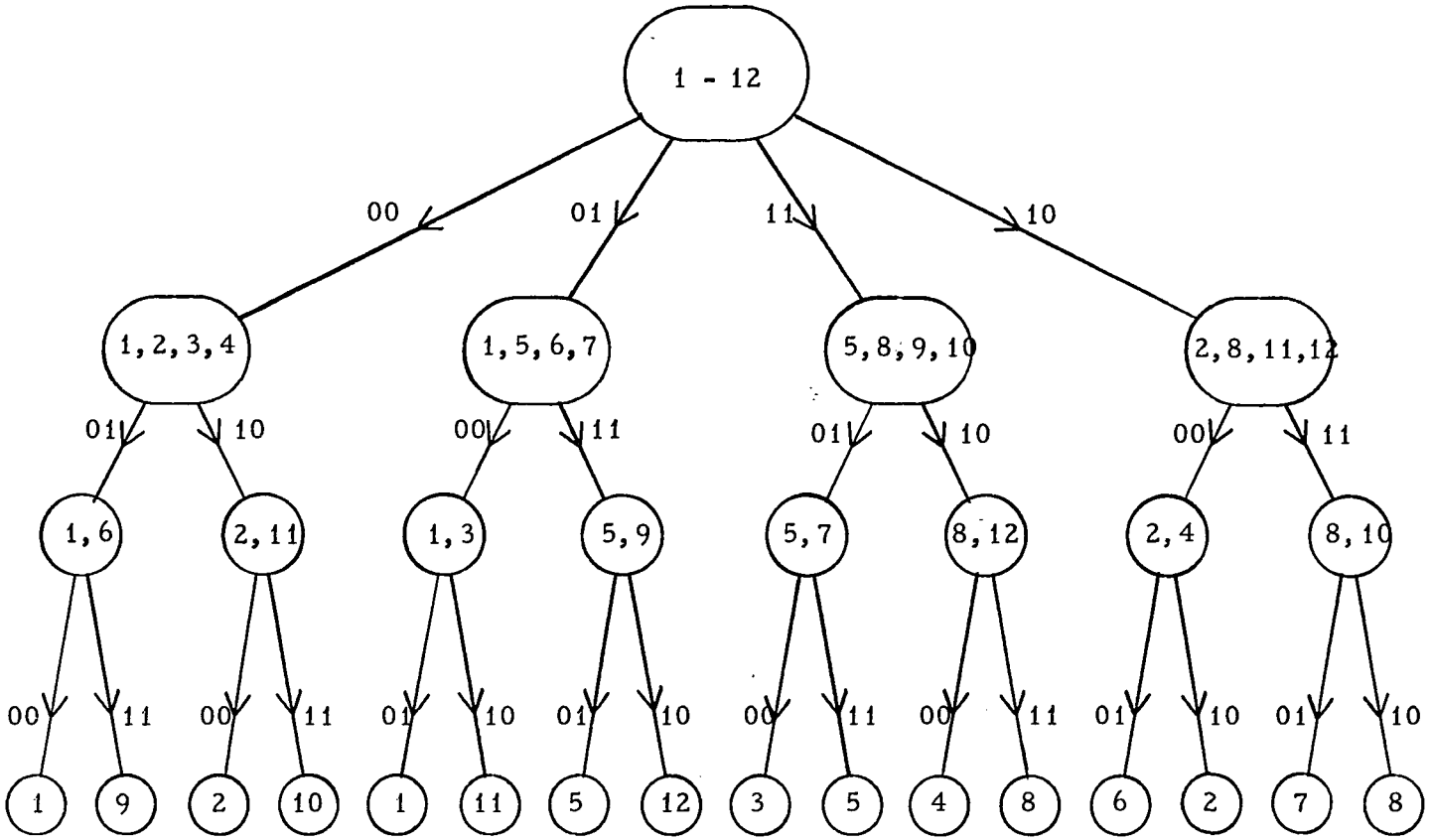

1 - 12
00
01
11
10
1, 2, 3, 4
1, 5, 6, 7
5, 8, 9, 10
2, 8, 11, 12
01
10
00
11
01
10
00
11
1, 6
2, 11
1, 3
5, 9
5, 7
8, 12
2, 4
8, 10
00
11
00
11
01
10
01
10
00
11
00
11
01
10
01
10
1
9
2
10
1
11
5
12
3
5
4
8
6
2
7
8

Figure 4.4

$D^0(X) = X$		$D(X)$		$D^2(X)$			Z	
x_1	x_2	$D(x_1)$	$D(x_2)$	$D^2(x_1)$	$D^2(x_2)$	State	z_1	z_2
0	0	0	1	0	0	1	0	0
0	1	0	0	0	1	1	0	0
0	0	1	0	0	0	2	0	0
1	0	0	0	1	0	2	1	0
0	0	0	1	1	1	3	1	1
0	0	1	0	1	1	4	1	1
0	1	1	1	0	1	5	0	1
1	1	0	1	1	1	5	0	1
0	1	0	0	1	0	6	1	0
0	1	1	1	1	0	7	1	1
1	0	1	1	1	0	8	0	1
1	1	1	0	1	1	8	1	1
1	1	0	1	0	0	9	1	1
1	1	1	0	0	0	10	0	0
1	0	0	0	0	1	11	1	1
1	0	1	1	0	1	12	1	0

$$z_1 = \bar{x}_1 \overline{D^2 x_1} + x_1 \overline{Dx_1}\,\overline{Dx_2} + x_1 Dx_2 \overline{D^2 x_1} + x_2 Dx_1 D^2 x_1 .$$

$$z_2 = x_1 \overline{Dx_2}\; D^2 x_2 + \overline{x_1}\; Dx_1 D^2 x_1 + x_1\, Dx_1 Dx_2 + Dx_1\, D^2 x_1 \overline{D^2 x_2}$$
$$+ x_2 Dx_1 Dx_2 + Dx_1 Dx_2 D^2 x_2 .$$

Figure 4.5 .

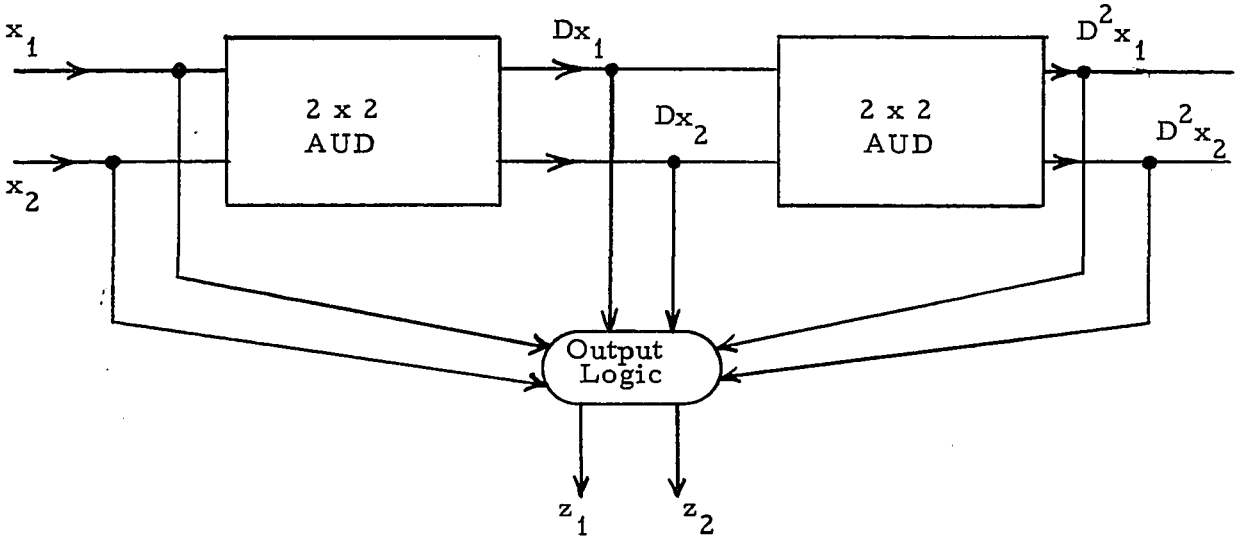


Figure 4.6

We shall now consider the general case, in which the given flow table need not be strongly connected. It is then possible that some states do not appear in the final level of the input successor tree. For example, if Y_i is stable in some column, but there are no unstable Y_i entries, then Y_i can be used as a starting state. However, once the input changes, Y_i cannot be re-entered. We shall take care of such situations by embedding every definite table in a strongly connected definite table.

Definition 4.5 : A state Y_i (stable in some input column) is a transient state if

(a) Y_i does not appear as an unstable entry of F, in which case Y_i is called a primary transient state, or

(b) All appearances of Y_i as unstable entries are next state entries of transient states of F.

Lemma 4.1 : Let F be a definite table and let Y_i be a state of F stable in some column. If Y_i does not appear in the final level of the input

successor tree, then it is a transient state.

Proof : Suppose the final level is the nth. Since Y_i appears in the 0th level, there must exist a k, $0 \leq k < n$ such that Y_i appears in the kth level, but does not appear in the $(k + 1)$ level. Thus Y_i must have a predecessor Y_j in the $(k - 1)$ st level and Y_j cannot appear in any greater level. Therefore all such Y_j are transient. The argument is then repeated for Y_j and its predecessors a finite number of times.

	x_1x_2 00	01	11	10
1	(1), 0	2	5, -	3, -
2	5, -	(2), 0	6, -	3, -
3	4, -	6, -	6, -	(3), 0
4	(4), 0	6, -	6	-, -
5	(5), 1	6, -	(5), 0	-, -
6	5, -	(6), 1	(6), 1	0, -

Figure 4.7 (a).

It is not true, however, that if Y_i is transient then it does not appear in the final level. This is illustrated in Figure 4.7, where states 1, 2, 3, 4 are transient, but 3 and 4 appear in the final level (the 3rd level). This has happened because there are don't care entries present. For example, Figure 4.7(b) indicates that applying (00)(10) to state 2 results in state 3. But this means that the don't care entry for state 5 column (10) will be state 3. Thus in fact, 3 ceases to be a transient state after the assignment is made, provided the input (10) can be applied to state 5. If this input is not allowed, then we must consider state 3 as transient.

If the don't care entries are interpreted to mean that the input is allowed, but the next state is of no consequence, then any table F in

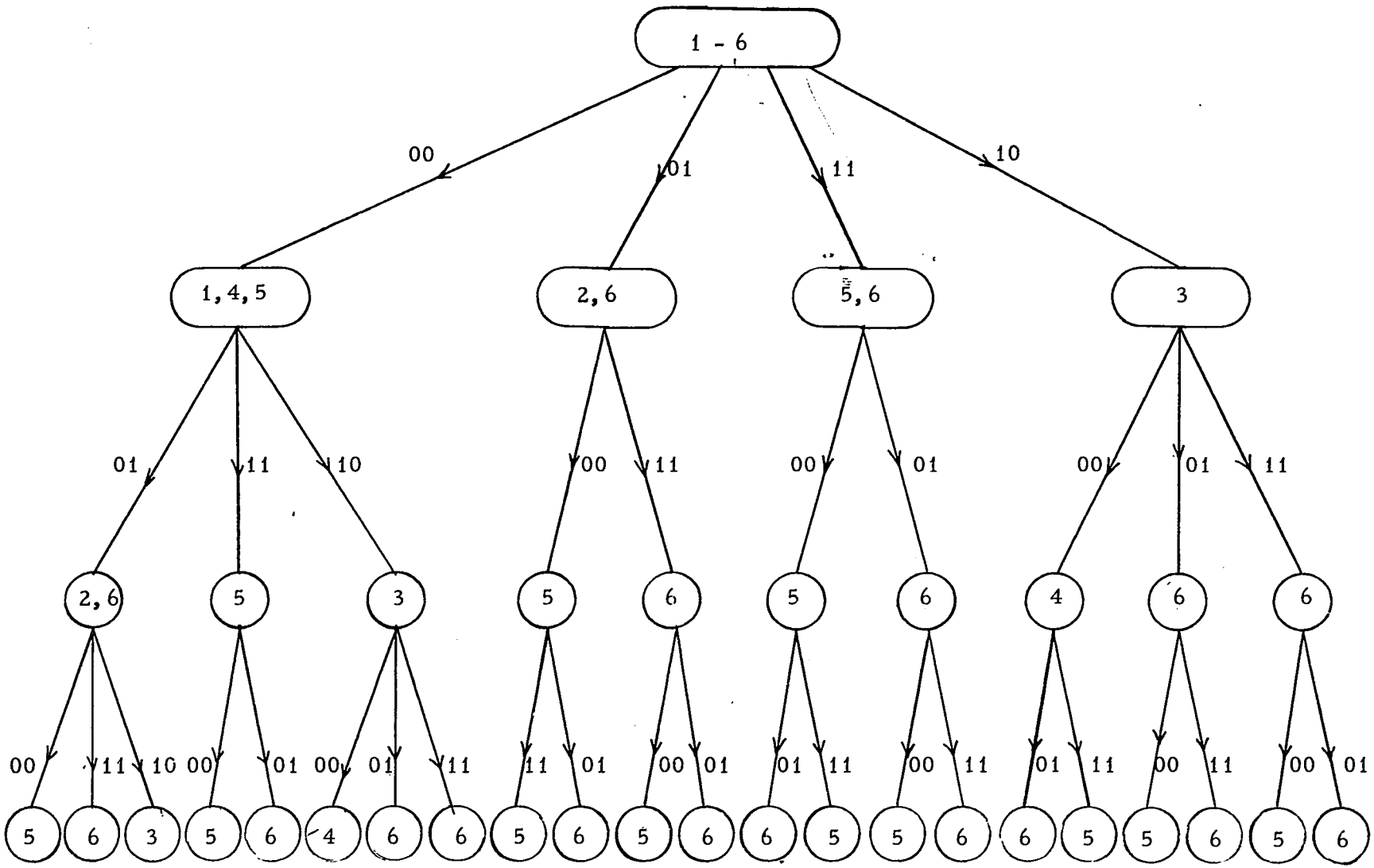

1 - 6
00
01
11
10
1, 4, 5
2, 6
5, 6
3
01
11
10
00
11
00
01
00
01
11
2, 6
5
3
5
6
5
6
4
6
6
00
11
10
00
01
00
01
11
11
01
00
01
01
11
00
11
01
11
00
11
00
01
5
6
3
5
6
4
6
6
5
6
5
6
6
5
5
6
6
5
5
6
5
6

Figure 4.7 (b)

which all states appear in the final level of the input successor graph will be strongly connected, and some don't care entries may be replaced by specific next states. We shall make this assumption in the following work.

Theorem 4.4 : Every definite table F is a subtable of a strongly connected definite table. This theorem was proved for synchronous machines in [5].

Proof : Case 1. All input changes allowed.

Suppose Y_i is a primary transient state stable in column X_j of F. For each such state Y_i add a column X'_j, in which all entries are Y_i. Now Y_i can be reached from every stable state by applying X'_j, and Y_i will appear in the final level of the input successor graph. Of course, F', the extended table, remains definite since F was definite and all the added columns have precisely one next-state entry. Since any other transient state must be a successor of a primary state, all transient states will appear in the final level. Thus F' is strongly connected and definite.

The added input columns still have to be encoded. Let $[x]$ denote the least integer $\geq x$. Suppose F has n columns and there are p primary transient states. Then F' has $m+p = m'$ columns. The minimum total number of binary inputs is $n' = [\log_2 m']$; thus $k = n' - n$ new inputs must be added. The new input n' - tuple in a column of F can consist of the original n-tuple with 0's added. In the added columns assign arbitrarily n' -tuples that have not been used for F.

Case 2. : Single input changes only :

First consider a transient state Y_i stable in column X_j which reaches a final level state Y_k stable in X_ℓ in one step i.e.

$M(Y_i, X_\ell) = Y_k$, X_j being adjacent to X_ℓ. Clearly if F has any transient states at all it must have at least one transient state with the above property. Add one new variable x_{n+1} for Y_i : In the columns of F let $x_{n+1} = 0$. Let $(X_j, 1)$ be the $(n+1)$ -tuple which is adjacent to $(X_j, 0)$ and differs only in x_{n+1}. Similarly let $(X_\ell\ 1)$ be adjacent to $(X_\ell\ 0)$. Now $(X_j, 1)$ and $(X_\ell, 1)$ are adjacent. Make all the entries in $(X_j, 1)$ equal to Y_i and those in $(X_\ell, 1)$ equal to Y_k. Then we have the situation shown in Figure 4.8 where X_j and X_ℓ are adjacent. Therefore a transition from final level state Y_k to transient state Y_i can be made using only single input changes. Also it is clear that the augmented table F' is definite and Y_i is now a final level state of F'.

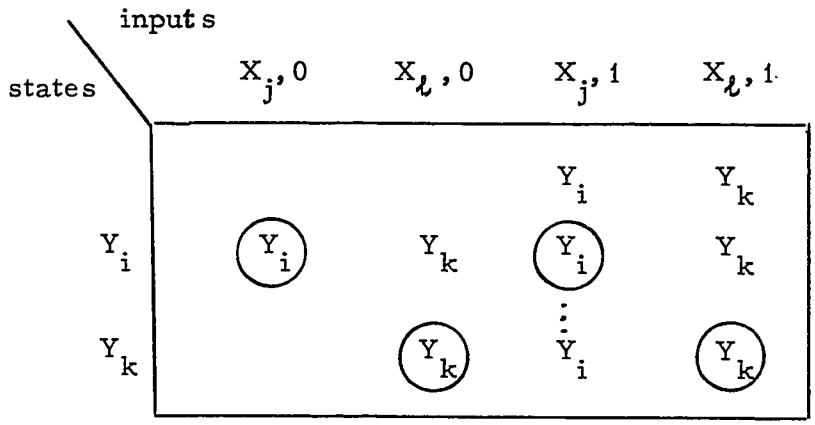


Figure 4.8

This procedure is repeated for the remaining states which reach a final state in one step. When this is done, a transient state which in F required a minimum of two steps to reach a final level state, in the augmented table requires only one step. Hence all the transient states can be treated in this fashion and a strongly connected definite table can be constructed for any definite F. This completes the proof for Case 2.

Case 1 above is straight forward. However in Case 2 one can often use fewer input variables than indicated in the above general construction. This is illustrated in the following examples. Figure 4.9 illustrates the various steps for obtaining a strongly connected definite table covering the original table.

The procedure suggests three extra input variables. However, it is possible to obtain a strongly connected definite table for the same example of Figure 4.9 with only one extra binary input instead of three. This is shown in Figure 4.10. The example in Figure 4.11 illustrates the fact that often it may be possible to add more than one state in the added input columns and yet the table can remain definite.

x_1x_2	00	01	11	10
1	(1)	-	(1)	4
2	1	(2)	1	-
3	4	-	1	(3)
4	(4)	1	(4)	(4)
5	1	(5)	4	-

(a) Given table F, single input changes.

$x_1x_2x_3$	000	010	110	100	011	001
1	(1)	-	(1)	4	2	(1)
2	1	(2)	1	-	(2)	1
3	4	-	1	(3)	2	1
4	(4)	1	(4)	(4)	2	1
5	1	(5)	4	-	2	1

(b) Making state 2 a final level state.

$x_1x_2x_3x_4$	0000	0100	1100	1000	0110	0010	1001	1101
1	(1)	-	(1)	4	2	(1)	3	(1)
2	1	(2)	1	-	(2)	1	3	1
3	4	-	1	(3)	2	1	(3)	1
4	(4)	1	(4)	(4)	2	1	3	1
5	1	(5)	4	-	2	1	3	1

(c) Handling of state 3.

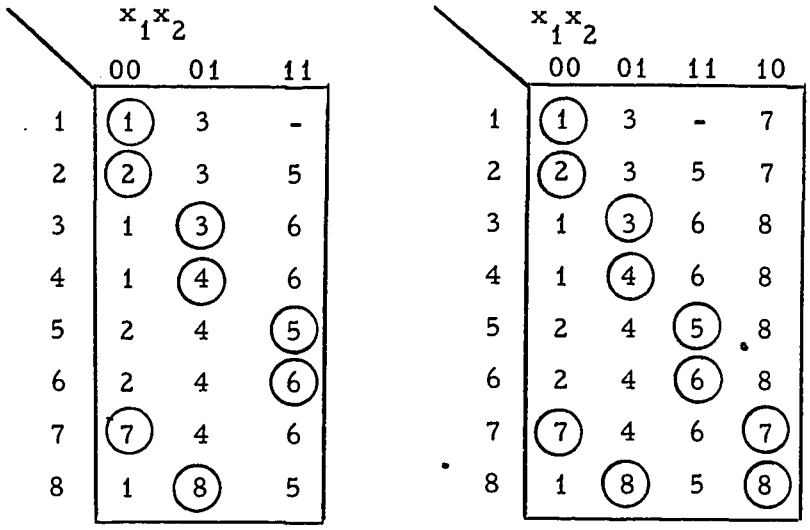
$x_1x_2x_3x_4x_5$	00000	01000	11000	10000	01100	00100	10010	11011	01001	11001
1	(1)	-	(1)	4	2	(1)	3	(1)	5	4
2	1	(2)	1	-	(2)	1	3	1	5	4
3	4	-	1	(3)	2	1	(3)	1	5	4
4	(4)	1	4	(4)	2	1	3	1	5	(4)
5	1	(5)	4	-	2	1	3	1	(5)	4

(d) Handling of state 5.

Figure 4.9 - The steps of the construction

$x_1x_2x_3$	000	010	110	100	101	001	011	111
1	(1)	-	(1)	4	3	(1)	2	5
2	1	(2)	1	-	3	1	(2)	5
3	4	-	1	(3)	(3)	1	2	5
4	4	1	(4)	(4)	3	1	2	5
5	1	(5)	4	-	3	1	2	(5)

Figure 4.10 - Embedding a definite machine into a strongly connected definite machine with fewer inputs.



	x_1x_2 00	01	11
1	(1)	3	-
2	(2)	3	5
3	1	(3)	6
4	1	(4)	6
5	2	4	(5)
6	2	4	(6)
7	(7)	4	6
8	1	(8)	5

	x_1x_2 00	01	11	10
1	(1)	3	-	7
2	(2)	3	5	7
3	1	(3)	6	8
4	1	(4)	6	8
5	2	4	(5)	8
6	2	4	(6)	8
7	(7)	4	6	(7)
8	1	(8)	5	(8)

Figure 4.11 - Adding more than one state per column

Corollary 4.1 : Every definite FMA machine is realizable by a shift register constructed with AUD's.

Theorem 4.5 : Every definite FMA machine with single input changes only can be realized by a series parallel connection of n-input 2-state definite FMA machines.

Proof : Since the nxn AUD with single input changes is realizable by a parallel connection of 2 state definite FMA machine, hence the proof is obvious.

Remarks : (1) It is evident from the previous discussion that there does not exist a non-trivial single input, strongly connected definite FMA machine. This can easily be proved by contradiction. Suppose such a machine A exists and has s states, s being ≥ 2. Then in the final level of the input successor graph of A each of its states must appear and the graph must terminate according to Rule (a).

It is clear that the graph does not have more than two nodes at any level. Therefore, if the machine is definite, there are only two states in the final level, the others being transient. But these two states can always be merged into a single state if only stable-state outputs are of interest, because they are stable in different columns. Hence the machine is either not strongly connected or it has only one state.

(2) Any asynchronous circuit obtained by introducing combinational networks in between AUD's remains definite, provided the outputs of the combinational networks do not violate the conditions set by the input restriction R for any AUD in the circuit. If the introduction of combinational logic violates R for some AUD, then the circuit operation is no longer proper.

(3) The method of realization of definite asynchronous machines suggested in this paper is not economical in all the cases :, for example, a 4 state machine with single input changes only in Figure 4.12 is realizable by a shift register of length three as compared to the 12 state machine in Figure 4.3 realizable by a register of length two. If we follow the conventional techniques, the 4 state machine can be realized with two S-R flip-flops compared to 6 S-R flip-flops in the three AUD's.

Example

	x_1x_2 00	01	11	10
1	①,00	①,11	2 , -	3 , -
2	3 , -	②,10	②,01	3 , -
3	③,10	1 , -	4 , -	③,11
4	1 , -	- , -	④,00	④,01

Figure 4.12 - Machine realizable by shift register of length 3.

duced in feedback loops [33] in order to avoid improper behavior, if an essential hazard exists. The table to Figure 4.3 has an essential hazard and does require inertial delays, but the inertial delays are not in a feedback loop.

There is another type of timing problems arising in cascade connections of AUD's. Consider an asynchronous circuit C of k nxn AUD's connected in cascade. Let the propagation delay for a change of value at the input terminal x_i which causes a change in the value of output at the output terminal $D(x_j)$ of the AUD be δ_{ij} for any $1 \leq i \leq n$ and $1 \leq j \leq n$. If the AUD's in the circuit are constructed with similar logic components, one would expect the δ_{ij}'s to be approximately the same for each AUD.

In general in any AUD δ_{ij} may vary for different values of i and j depending upon the number of gates the binary signals have to go through to affect the value of the output. It is also understood that each AUD has been constructed to be free from races and hazards.

Therefore, it is clear that, as long as the time interval between any two successive input changes is $\max(\delta_{ij})$, all output changes in the AUD will have been completed before the next input change is made. Since circuit C has k AUD's, if no two successive input changes are made in the time interval $\Delta_t < k \max(\delta_{ij})$, we can be assured of the correct transitions between states of the circuit. The circuit outputs will also be correct provided the combinational output circuits are free from hazards.

Figure 4.13 illustrates a situation which could possibly occur if the inputs violate the above condition and the circuit may not operate properly. In the figure, the time intervals a and b are so small that some of the changes in Dx_1, Dx_2 may be interpreted as double changes by the next AUD.

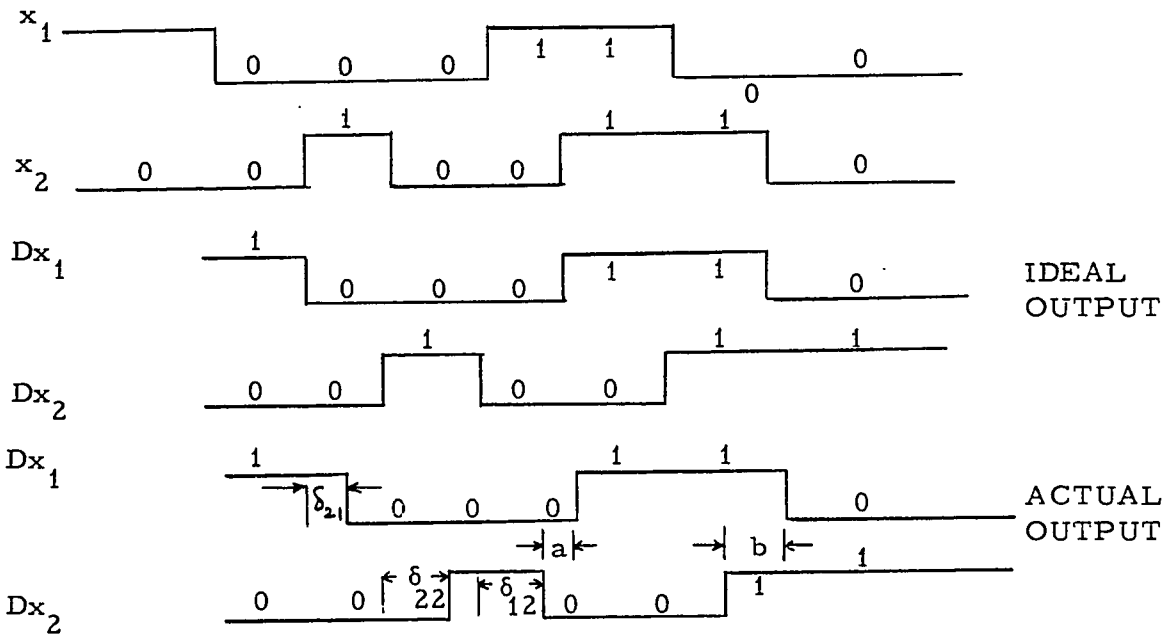


Figure 4.13 - Illustrating the effects of δ_{ij} .

Chapter 5

ASYNCHRONOUS SEQUENTIAL CIRCUITS WITH FEEDBACK

Introduction :

In Chapter 4 an n x n AUD was used as a basic circuit element in the construction of definite asynchronous circuits. Here we shall show than an n x n AUD can also be used as a circuit element in the design of any asynchronous circuit. It will be shown that circuits constructed with AUD's also need inertial delay elements and any asynchronous fundamental mode flow table F is realizable by a circuit of feedback index m with one (n + m) x (n + m) AUD, m inertial delays and a combinational network, where m is the smallest integer $\geq [\log_2 \max(S_i)]$ and S_i is the number of stable states in any column $X_i \in F$. We will also show that every fundamental mode indefinite[11] table can be realized by a circuit of feedback index 1, with k (n + 1) x (n + 1) AUD's, one inertial delay element and a combinational network. Since the circuit with feedback index 1 may require many (n + 1) x (n + 1) AUD's, therefore, from the point of view of cost, reliable operation, speed and maintenance considerations, a circuit realization with lesser number of AUD's may be a good compromise.

In the last section we will consider pulse controlled FMA circuits very briefly. Their merits over general FMA circuits will also be discussed.

---

11 - FMA machines which are not definite are called indefinite.

5.1 Finite Memory Asynchronous Circuits

The following definitions have been stated for synchronous machines in [6, 14, 15].

Definition 5.1 : Given an asynchronous sequential circuit C constructed with basic building blocks such as gates, flip-flop's, AUD's and delay elements (pure or inertial), the value of a binary signal on any wire w_i of C at any given time is a function of the state and the input at that time. A set of wires which, when cut, remove all the closed loops of C is called a feedback set. The logical value of the binary signals of the wires of feedback set defines a function of the state and the input of C. This function will be called a feedback function of C and will be denoted as f(Y;X).

Definition 5.2 : For any flow table F of an FMA machine a function of the present state and the input is a feedback function f(Y,X) of F if there exists a circuit C which realizes F and has f(Y,X) as a feedback function.

Definition 5.3 : A Finite memory FMA machine A is a machine whose output is a function of only the present value of the input and a finite number k of past inputs and output values, i.e.

$$Z(t) = N\left[X(t),\ D(X(t),\ Z(t)),\ D^2(X(t),\ Z(t)),\ \ldots\ D^k(X(t),\ Z(t))\right] .$$

where $X(t), = (x_1(t), \ldots x_n(t))$, $(X(t), Z(t))$ is the combined input-output tuple $= (x_1(t), x_2(t), \ldots x_n(t) ; z_1(t), z_2(t), \ldots, z_m(t))$ and D denotes the delay operator. Define $D^0(X(t), Z(t)) = (X(t), Z(t))$ and $D(D(X(t), Z(t)) = D^2(X(t), Z(t)) = (X(t - \delta_1 - \delta_2), Z(t - \delta_1 - \delta_2))$, where the input-output tuple was changed at time $(t - \delta_1 - \delta_2)$ and $(t - \delta_1)$ etc.. The symbol k represents the memory of A.

Definition 5.4 : Let F be the flow table of an FMA machine with input restriction R. A combined input-output sequence $\overline{X, Z}$ is valid for F iff (a) no two successive symbols in $\overline{X, Z}$ are identical and (b) $\overline{X, Z}$ obeys the restriction R and (c) when $\overline{X}$ is applied Z is the resulting output sequence.

For the moment if we assume (i) that the output of each AUD and the output of the combinational network appear instantaneously whenever the input is changed (that is, assuming zero propagation delay for signals passing through different AUD's and combinational network), and (ii) that the combinational network for outputs is hazard-free, then Definition 5.3 suggests the circuit realization as shown in Figure 5.1. The AUD's in the register consist of $n + m$ input and $n + m$ output terminals, n terminals to remember past k values of n inputs and m for the past k values of m outputs. The m input terminals of the first AUD are different because its inputs cannot be used as inputs to the combinational networks for the FMA machine. Since there are m binary outputs, therefore the circuit has $\leq m$ binary feedback loops.

The decision procedure for testing whether a given flow table F of an FMA machine has finite memory is quite similar to the synchronous case [4, 6, 15]. We define the input/output successor tree $G_0(F)$.

1. The zeroth level of the $G_0(F)$ contains one node (a_0) corresponding to Y_S the set of all internal states which are stable in some input column of F.

2. Under a distinct input/output pair X_i/Z_j there is a transition from the node (a_0) to the first level node (a_{1i}) which contains all the stable states $M(Y_j, X_i)$, $Y_j \in Y_S$ such that $N(Y_j, X_i) = Z_j$. There is one node for each X_i/Z_j. Note that since the output value is of interest only when the total state is stable, therefore input/output pairs are formed

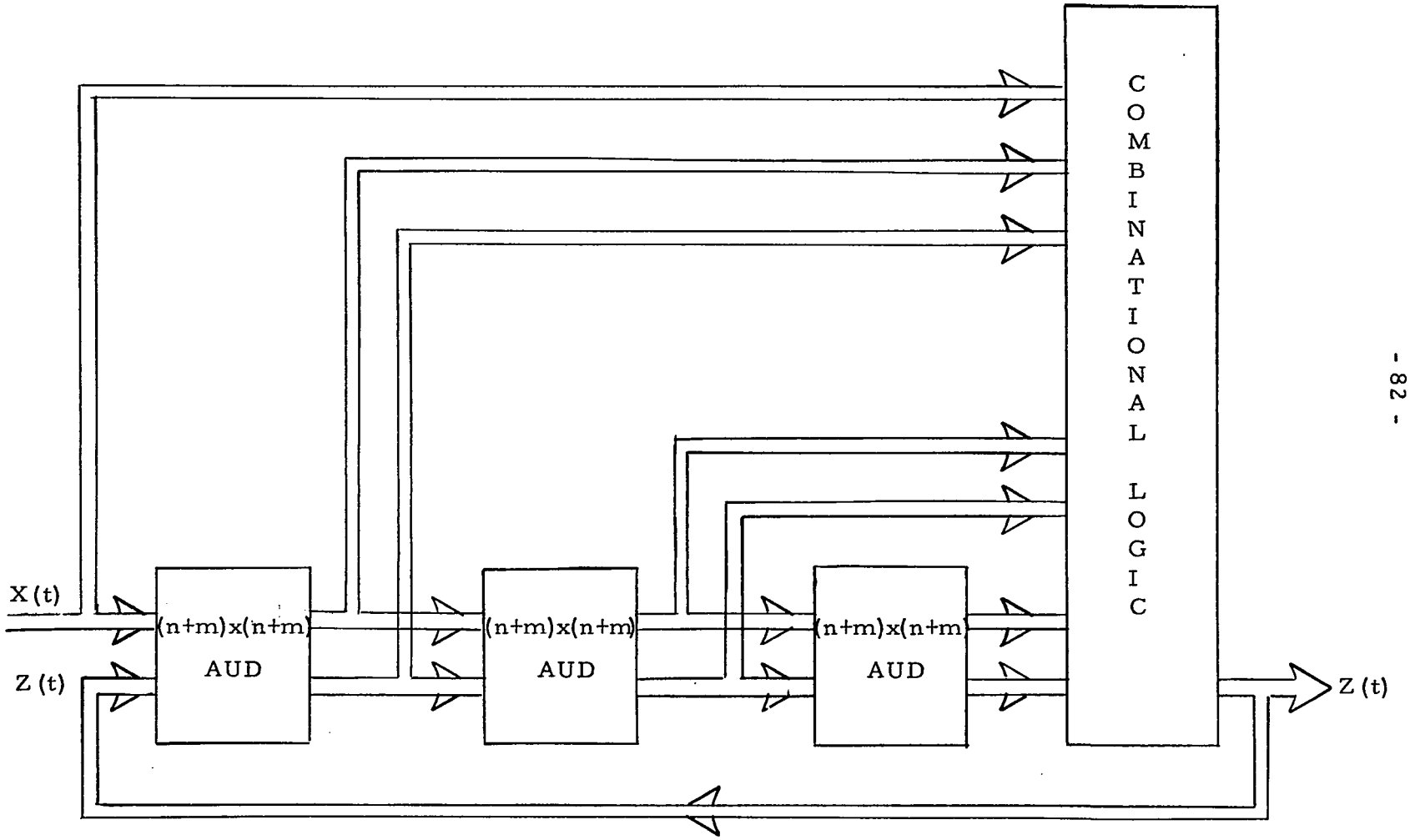


Figure 5.1 - A finite memory circuit

for the outputs associated with stable states only.

3. The second level contains nodes corresponding to the successor of the first level states reached by valid input/output sequences considered from $(a_0).G_0(F)$ is terminated according to one of the following rules :

(a) A level is reached in which all the stable states of the machine have been resolved, i.e. in the level each node corresponds to a single state, or

(b) $G_0(F)$ contains a path along which a node with more than one state appears twice.

It is obvious that the graph is always finite. Therefore, if it is terminated by Rule (a) , then FMA machine is a finite memory machine, and if by Rule (b) , then it is not.

For example, the FMA machine of Figure 5.2 yields a graph of Figure 5.3. Since the graph terminates according to Rule (a), it is a finite memory machine.

x	0	1
1	(1), 0	2 , -
2	3 , -	(2), 0
3	(3), 1	4 , -
4	- , -	5 , -
5	1 , -	(5), 1
6	(6), 1	5 , -

Figure 5.2

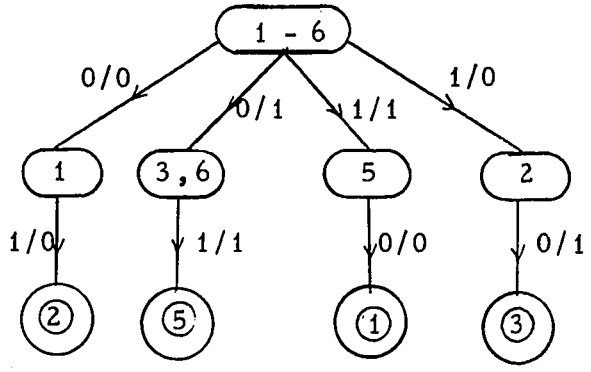


Figure 5.3

It is obvious that all FMA machines do not have finite memory property. For example, the machine of Figure 5.4(a) is not a finite memory machine. However, to realize its state behavior as a finite memory machine, we can always introduce an auxillary output Z_0 equal to a feedback function $f(Y,X)$ of F shown in the table of Figure 5.4(b). There could be many choices for selecting a suitable $f(Y,X)$ for a given table. For example, one obvious choice for any given F could be that $f(X,Y) = M(Y,X)$.

x_1x_2	00	01	11	10
1	(1),0	(1),1	3,-	4,-
2	(2),0	(2),1	4,-	3,-
3	1,-	2,-	(3),0	(3),1
4	1,-	2,-	(4),1	(4),0

Figure 5.4(a)

x_1x_2	00	01	11	10
1	1	1	1	1
2	0	0	0	0
3	1	1	1	1
4	0	0	0	0

Figure 5.4(b)

Unlike the synchronous case an input/output successor graph $G_0(F)$ cannot provide the state assignment for a given finite memory FMA machine. Since in actual practice in the circuit of Figure 5.1 the output of each AUD and the output of combinational logic will not change instantaneously with the change of input due to non-zero propagation delay in the gates used in AUD's and combinational logic, hence assumption (i) stated before cannot hold true. Also, since the output Z_j is a function of X_i, $D(X_i), \ldots D^k(X_i)$, a change in X_i can affect Z_j through many different paths, one through different AUD's and the other more directly. This is the basic condition for a combinational hazard. Thus transient erroneous pulses may appear at the output lines. Such a situation can only be avoided if the binary outputs $\{z_1, z_2, \ldots z_m\} = Z_j$ are passed through inertial delay ele-

ments Δ's before they are allowed to appear at the m input terminals of the first AUD. Therefore for any finite memory FMA machine realized by circuits as shown in Figure 5.1, inertial delays in the feedback lines will be necessary. Hence a change in X at time t will cause a change in Z appearing at the input of the first AUD will be further delayed by Δ units of time. Thus in practice an input/output successor graph of any properly operating circuit will be different from $G_0(F)$ already defined. We shall denote this modified input/output successor graph as $G_m(F)$.

Definition 5.5 : Let $G_m(F)$ be a graph of a finite memory FMA flow table F defined as follows. Let there be a transition under X_i/Z_j from node (a_0) to some first level node (a_{1i}) and a transition under X_k/Z_ℓ from (a_{1i}) to some second level node (a_{2j}) in $G_0(F)$.

(i) If $Z_j = Z_\ell$, then transition from (a_{1i}) to (a_{2j}) is direct in $G_m(F)$.

(ii) If $Z_j \neq Z_\ell$, then the transition from (a_{1i}) to (a_{2j}) is through an intermediate node a'_{2j} corresponding to a set of unstable states in $G_m(F)$.

Since every finite memory FMA table can be realized by a circuit of Figure 5.1, therefore after the initial conditions in the register are cleared, each internal state of F must be uniquely represented by the last k values of input/output-tuples stored in the register. Let Y_m be some state stable in column X_i of F and be represented uniquely by values (X_i, Z_j), $D(X_i, Z_j)$, ... $D^k(X_i, Z_j)$ in the register.

(i) Let X_i be changed to X_k such that $M(Y_m, X_k) = Y_n$, $M(Y_n, X_k) = Y_n$ for both $n = m$ and $n \neq m$, $N(Y_m, X_i) = Z_j$ and $N(Y_n, X_k) = Z_\ell$ for $j = \ell$. Correspondingly state Y_n will be represented uniquely by values (X_k, Z_j), $D(X_k Z_j)$, ... $D^k(X_k Z_j)$ in the register. Hence this transition from Y_m to Y_n on $G_m(F)$ will be represented directly

from a node (a_{1i}) of stable states to (a_{2j}) another node of stable state, because the output Z_j is same for Y_m and Y_n.

(ii) Let X_i be changed to X_k such that $M(Y_m, X_k) = Y_n$, $M(Y_m X_k) = Y_n$ for both $n = m$ and $n \neq m$, and $N(Y_n, X_k) = Z_\ell$ for $j \neq \ell$. Since output Z_j cannot be changed to Z_ℓ instantaneously with the change of input, initially the circuit of Figure 5.1 will correspond to some intermediate state Y_p represented by the values (X_k, Z_j) $D(X_k, Z_j)$,... $D^k(X_k Z_j)$ for some time $\Delta > 0$ and then eventually Z_ℓ will appear and circuit will take another sets of values (X_k, Z_ℓ), $D(X_k, Z_\ell)$, ... $D^k(X_k Z_\ell)$ corresponding to stable state Y_n in X_k. Such a transition from Y_m to Y_n on $G_m(F)$ will naturally be represented through an intermediate node a'_{2j} corresponding to some unstable state Y_p to the node (a_{2j}) of stable states corresponding to Y_n in the second level.

This implies that in $G_0(F)$ obtained for testing the finite memory machines we must insert nodes representing sets of unstable states whenever an input change is causing a change in the output.

We shall see later that $G_m(F)$ is directly useful in obtaining state assignment for the circuit realization of finite memory FMA machines. The reader must note that all the AUD's in the register of such a circuit are constructed for all input changes allowed. The AUD with single input changes can be used only when input and output separately follow the restriction of single changes only.

If a finite memory FMA machine is to be realized by AUD's with single input changes only then it is clear that the modified graph $G_m(F)$ may have more than one node of unstable states between any two levels of stable states in $G_m(F)$ after the first level.

For example the modified input/output successor graph $G_m(F)$ of finite memory machine of Figure 5.5(a) and its augmented flow table with extra unstable states are shown in Figure 5.5(b) and 5.5(c) respectively.

x_1x_2	00	01	11	10
1	(1), 00	(1), 00	5	6
2	(2), 01	(2), 01	5	6
3	(3), 11	(3), 01	5	6
4	(4), 10	(4), 10	5	6
5	4	2	(5), 01	(5), 01
6	1	3	(6), 10	(6), 10

Figure 5.5 (a)

5.2 Single AUD Asynchronous Circuits

Theorem 5.1 : Any FMA machine with a given state assignment can be realized by a circuit of feedback index m with one $(n + m)x(n + m)$ AUD and combinational network, where n and m are the number of binary inputs and state variables respectively.

Proof : Let F be the flow table of an FMA machine A. Let the feedback function $f(Y,X) = M(Y,X)$. Considering $f(Y,X)$ as an auxiliary output of F if we draw $G_o(F)$ of F, then it is clear that in the first level each node will have only one stable state, because each state of F has a unique unique binary code. As $G_0(F)$ terminates at level 1 according to Rule (a), A is a finite memory machine. Since to estimate the contribution of the AUD in any single AUD circuit we must extend the graph to second level of stable states, $G_0(F)$ is further

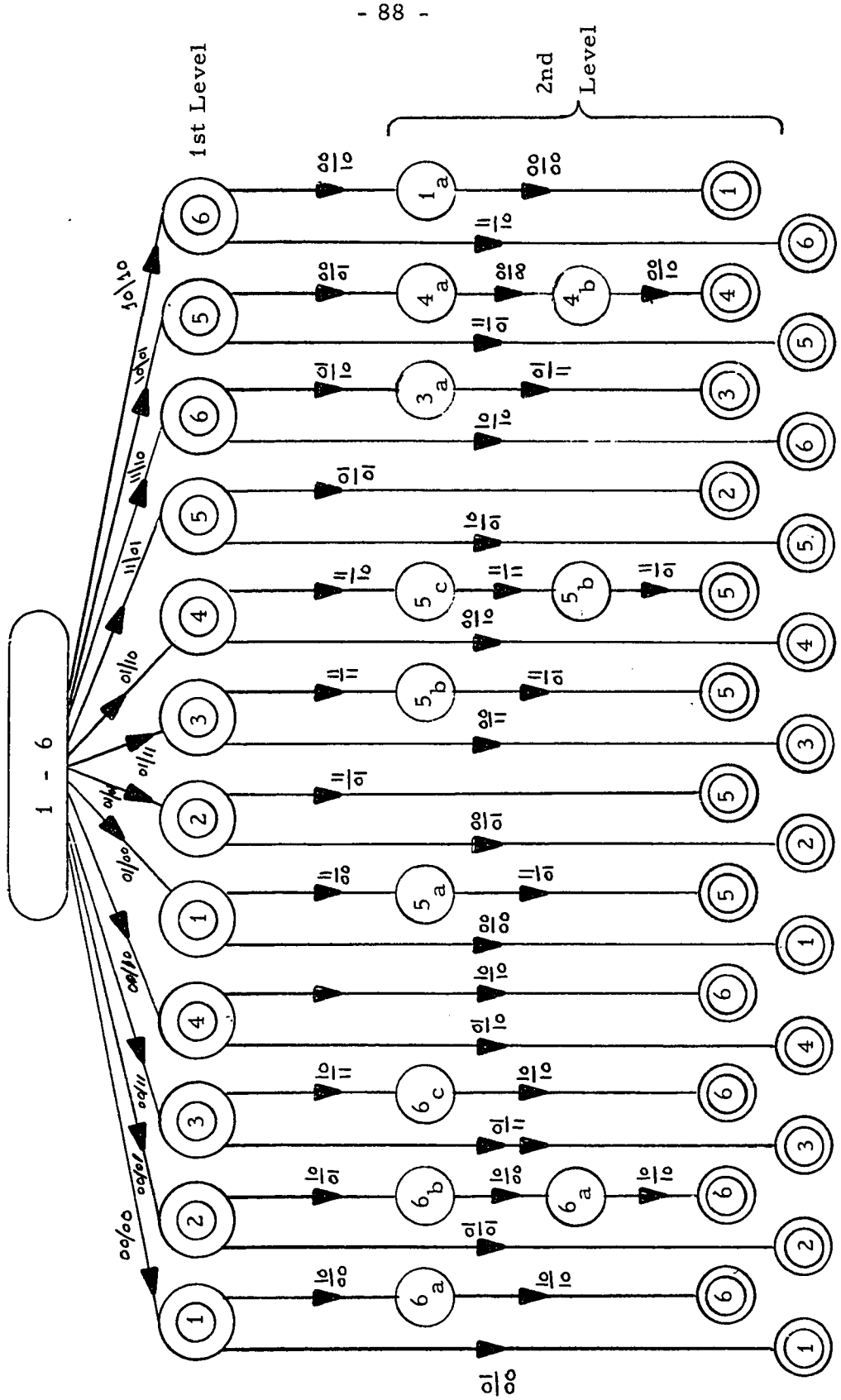


Figure 5.5 (b) - $G_m(F)$ of Figure 5.5 (a).

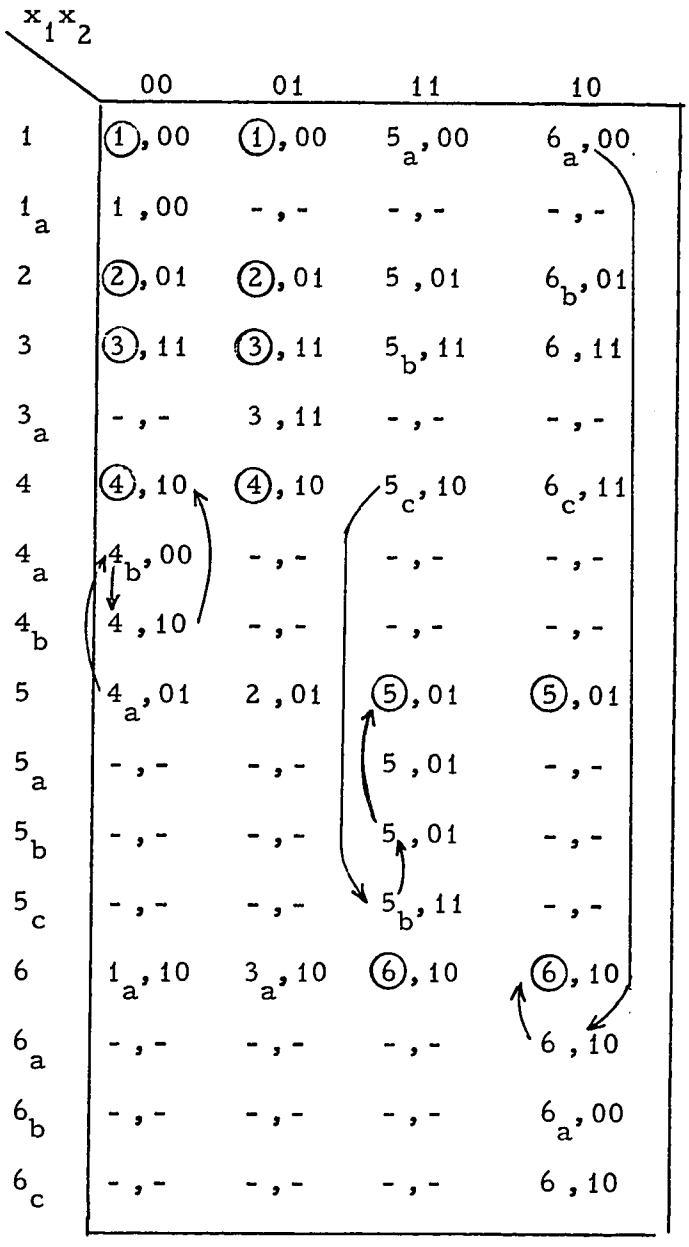


Figure 5.5(c) - Augmented table of the table of Figure 5.5(a) showing multiple changes of secondary states.

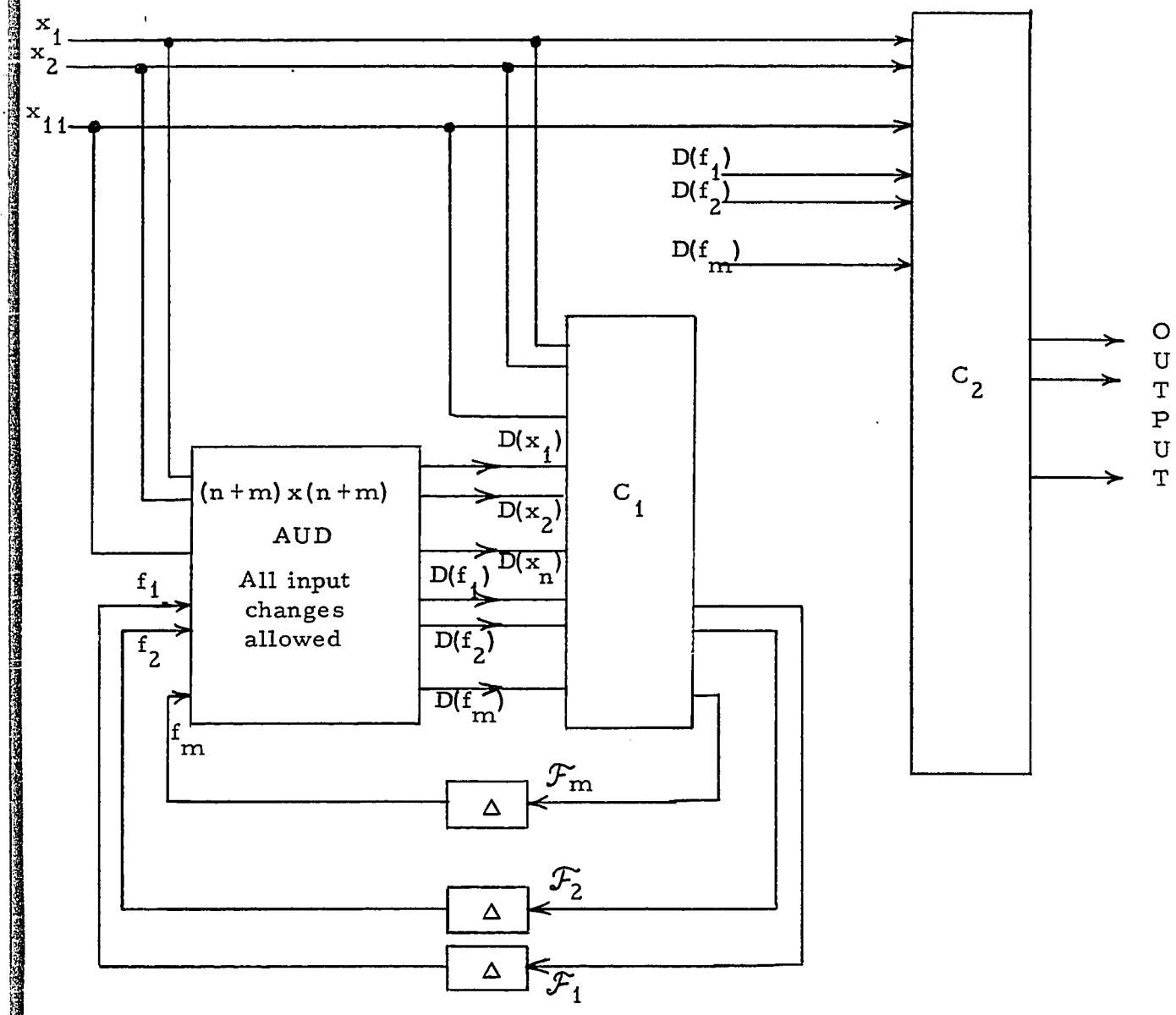


Figure 5.6 - Single AUD circuit .

extended to the second level of stable states. According to Definition 5.5, for each $G_0(F)$ one can always obtain a unique modified input/output successor graph $G_m(F)$ in which second level will have stable as well as unstable states. Assign to each appearance of Y_i stable or unstable in the second level of $G_m(F)$ the reverse of input/output sequence of length 2 required to map Y_i on the tree, because a single AUD can only remember the input-output tuple prior to the last change in its value. (Y_i may be represented by more than one state of the AUD). Therefore the state behavior of F can be realized by a circuit of one AUD and combinational network C_1 with feedback index m as shown in Figure 5.6. The actual output can then be obtained by additional combinational circuit C_2. This completes the proof of the theorem.

In future, for the sake of convenience we shall represent the present and next values of feedback function $f(Y,X)$ by f and $\mathcal{F}$ respectively.

For example, consider the table of Figure 5.7 with single input changes and its modified tree $G_m(F)$ in Figure 5.8. The resulting augmented flow table of Figure 5.7, state assignment and the circuit are shown in Figures 5.9, 5.10 and 5.11 respectively.

y_1	y_2		x_1x_2 = 00	01	11	10
0	0	1	(1)	(1)	2	4
0	1	2	1	3	(2)	(2)
1	1	3	(3)	(3)	4	2
1	0	4	3	1	(4)	(4)

Figure 5.7

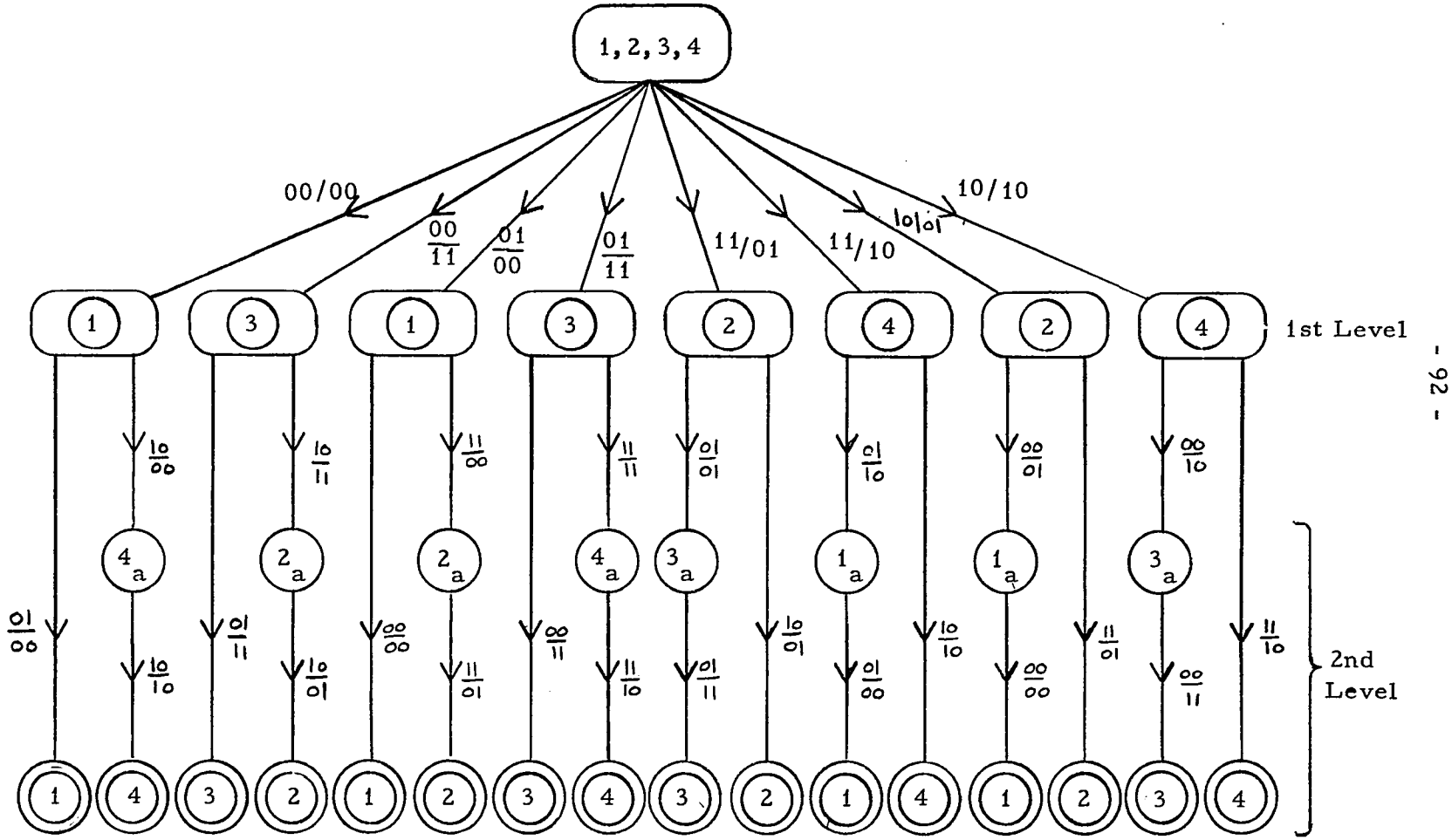

1, 2, 3, 4
00/00
00/11
01/00
01/11
11/01
11/10
10/01
10/10
1st Level
2nd Level

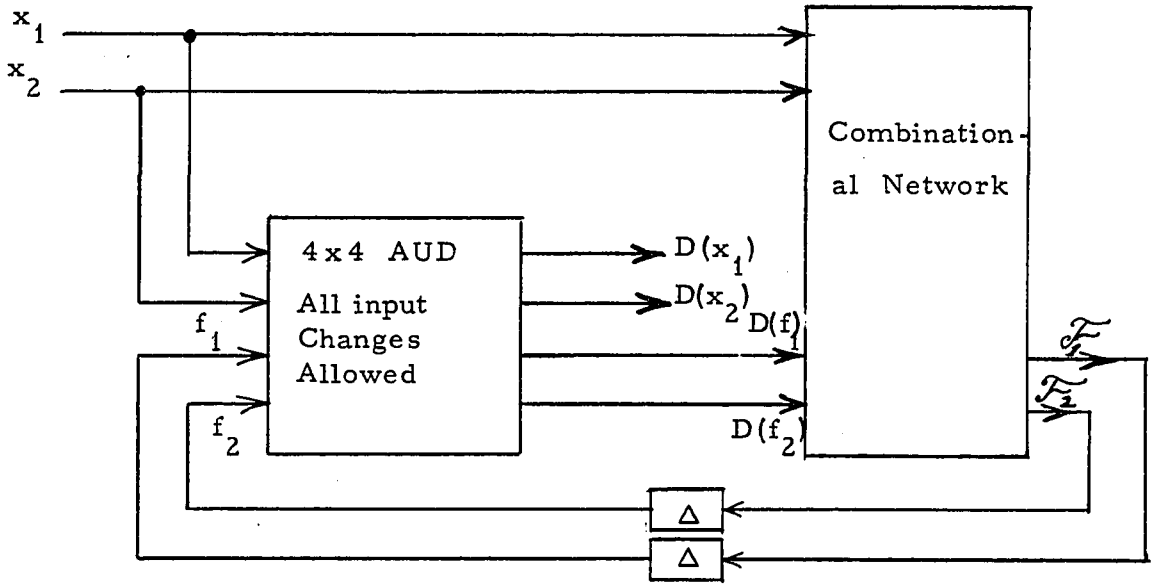
Figure 5.8 .

x_1x_2	00	01	11	10
1	(1), 00	(1), 00	2_a, 00	4_a, 00
1_a	1 , 00	1 , 00	- , -	- , -
2	1_a, 01	3_a, 01	(2), 01	(2), 01
2_a	- , -	- , -	2 , 01	2 , 01
3	(3), 11	(3), 11	4_a, 11	2_a, 11
3_a	3 , 11	3 , 11	- , -	- , -
4	3_a, 10	1_a, 10	(4), 10	(4), 10
4_a	- , -	- , -	4 , 10	4 , 10

Figure 5.9 - Augmented flow table of the table in Figure 5.7

Internal State	$D^0(X)=X$		$D(X, f)$				Present f.b. function		Next f.b. function	
	x_1	x_2	$D(x_1)$	$D(x_2)$	$D(f_1)$	$D(f_2)$	f_1	f_2	$\mathcal{F}_1$	$\mathcal{F}_2$
1	0	1	0	0	0	0	0	0	0	0
4	1	0	1	0	0	0	1	0	1	0
4_a	1	0	0	0	0	0	0	0	1	0
3	0	1	0	0	1	1	1	1	1	1
2	1	0	1	0	1	1	0	1	0	1
3_a	1	0	0	0	1	1	1	1	0	1
1	0	0	0	1	0	0	0	0	0	0
2	1	1	1	1	0	0	0	1	0	1
2_a	1	1	0	1	0	0	0	0	0	1
3	0	0	0	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	0	1	0
4_a	1	1	0	1	1	1	1	1	1	0
3	0	1	0	1	0	1	1	1	1	1
3_a	0	1	1	1	0	1	0	1	1	1
2	1	0	1	1	0	1	0	1	0	1
1	0	1	0	1	1	0	0	0	0	0
1_a	0	1	1	1	1	0	1	0	0	0
4	1	0	1	1	1	0	1	0	1	0
1	0	0	0	0	0	1	0	0	0	0
1_a	0	0	1	0	0	1	0	1	0	0
2	1	1	1	0	0	1	0	1	0	1
3	0	0	0	0	1	0	1	1	1	1
3_a	0	0	1	0	1	0	1	0	1	1
4	1	1	1	0	1	0	1	0	1	0

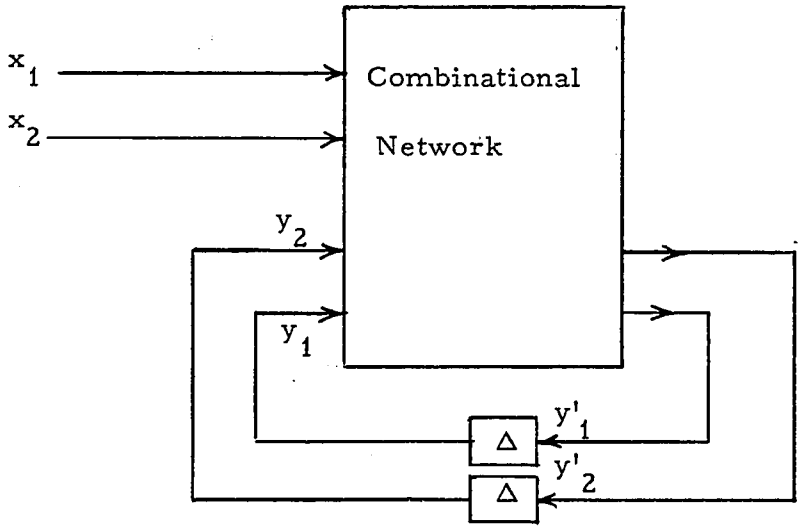
Figure 5.10.



$$\mathcal{F}_1 = \bar{x}_1\bar{x}_2\ D(f_1) + \bar{x}_1 x_2\ D(f_2) + x_1 x_2\ D(f_1) + x_1\bar{x}_2\ \overline{D(f_2)}$$

$$\mathcal{F}_2 = \bar{x}_1\bar{x}_2\ D(f_1) + \bar{x}_1 x_2\ D(f_2) + x_1 x_2\ \overline{D(f_1)} + x_1\bar{x}_2\ D(f_2)$$

Figure 5.11



$$y'_1 = \bar{x}_1\bar{x}_2 y_1 + \bar{x}_1 x_2 y_1 + x_1 x_2 y_1 + x_1\bar{x}_2\bar{y}_2$$

$$y'_2 = \bar{x}_1\bar{x}_2 y_1 + \bar{x}_1 x_2 y_2 + x_1 x_2\bar{y}_1 + x_1 x_2 y_2$$

Figure 5.12

Remarks

Comparing the Boolean expressions for $\mathcal{F}_1$ and $\mathcal{F}_2$ of the circuit of Figure 5.11 with the next state Boolean expressions for y'_1 and y'_2 for the conventional circuit realization of Figure 5.12, we find that the two sets of expressions are identical. Since the use of inertial delay elements is inevitable in AUD circuits and at the same time their presence in the circuit also fulfills the role of state devices, the role of AUD as a memory device becomes trivial.

Now we shall show generally that the role of $(n+m)$ x $(n+m)$ AUD in a single AUD circuit realizing an FMA flow table F with a given state assignment is trivial if its auxillary output function $N_0(Y,X) = f(Y,X) = M(Y,X)$.

We already know that the presence of inertial delay elements in feedback lines connecting m auxiliary outputs to m input terminals of the first AUD is essential for the proper operation of any single-AUD circuit. Let the circuit be in some stable state and let X_i and Z_{0j} be the present input and output respectively. Since $N_0(Y,X) = M(Y,X)$ and each internal state is given a unique code, the present input and present output values are sufficient to determine the total stable state $M(Y_m, X_i) = Y_m$. Let X_i be changed to X_j. If

(i) $M(Y_m, X_k) = Y_m$, then $D(X_k, Z_{0j}) = (X_i, Z_{0j})$

(ii) $M(Y_m, X_k) = Y_n$ and $M(Y_n, X_k) = Y_n$, then

first $D(X_k, Z_{0j}) = (X_i, Z_{0j})$ for time $t \leq \Delta$,

followed by $D(X_k, Z_{0\ell}) = (X_k, Z_{0j})$ for time $t \geq \Delta$,

Δ being the value of inertial delay element. Since to determine Y_n

we need not know the previous input $D(X_k)$ and since $D(Z_{0\ell}) = Z_{0j}$ is unique for each such input change, one can always replace the present state variables in the Boolean expressions for the next state variables in the conventional circuit realization by the corresponding delayed variable in the single AUD circuit. Basically, the role of a memory device has already been fulfilled by the inertial delays.

In order to obtain a non-trivial single AUD circuit, we must remove the redundancy in the memory role of the AUD and the inertial delays by reducing the number of delay elements to a minimum possible value.

Theorem 5.2 : Any flow table F of an FMA machine can be realized by a circuit of feedback index m with one $(n+m) \times (n+m)$ AUD and a combinational network, where n is the number of binary inputs and m is the smallest positive integer $\geq [\log_2 \max (S_i)]$, S_i being the number of stable states in any column X_i of F.

Proof : Let S_i be the maximum number of stable states in some X_i of F. In order to distinguish each stable state in X_i we must have at least m variables $\geq [\log_2 (S_i)]$. In other columns $X_j \neq X_i$, where the number of stable states are $\leq$ in X_i, m variables are automatically enough to distinguish one stable state from another. Therefore with the help of present input X_i and m binary variable code assigned to each total stable state we can distinguish each state of F. Now if this code is considered as auxiliary output of F and an input/output successor graph $G_0(F)$ is constructed, then such a graph will have only one total stable state in each node of the first level. The rest follows as given in Theorem 5.1. This completes the proof.

Once again, considering the previous example of Figure 5.7 with its auxiliary output equal to f(Y,X) of Figure 5.12, the resulting assignment and the circuit are as shown in Figures 5.13 and 5.14 respectively.

x_1x_2	00	01	11	10
1	0	0	0	0
2	0	0	0	0
3	1	1	1	1
4	1	1	1	1

Figure 5.12 - f(Y,X) of the table of Figure 5.7.

State	$D^0(X) = Y$		$D(X,f)$			Present f.b. Function	Next f.b. Function
	x_1	x_2	$D(x_1)$	$D(x_2)$	$D(f)$	f	$\mathcal{F}$
1	0	1	0	0	0	0	0
4	1	0	1	0	0	1	1
4_a	1	0	0	0	0	0	1
3	0	1	0	0	1	1	1
2	1	0	1	0	1	0	0
2_a	1	0	0	0	1	1	0
1	0	0	0	1	0	0	0
2	1	1	0	1	0	0	0
3	0	0	0	1	1	1	1
4	1	1	0	1	1	1	1
3	0	1	0	1	0	1	1
3_a	0	1	1	1	0	0	1
2	1	0	1	1	0	0	0
1	0	1	0	1	1	0	0
1_a	0	1	1	1	1	1	0
4	1	0	1	1	1	1	1
1	0	0	1	0	0	0	0
2	1	1	1	0	0	0	0
3	0	0	1	0	1	1	1
4	1	1	1	0	1	1	1

$$\mathcal{F} = \bar{x}_1\bar{x}_2\ D(f) + \bar{x}_1 x_2\ D(x_2)\ \overline{D(f)} + x_2\ \overline{D(x_2)}\ D(f) + x_1 D(x_2)\ D(f)$$
$$+ x_1\bar{x}_2\ \overline{D(x_2)}\ \overline{D(f)}$$

Figure 5.13

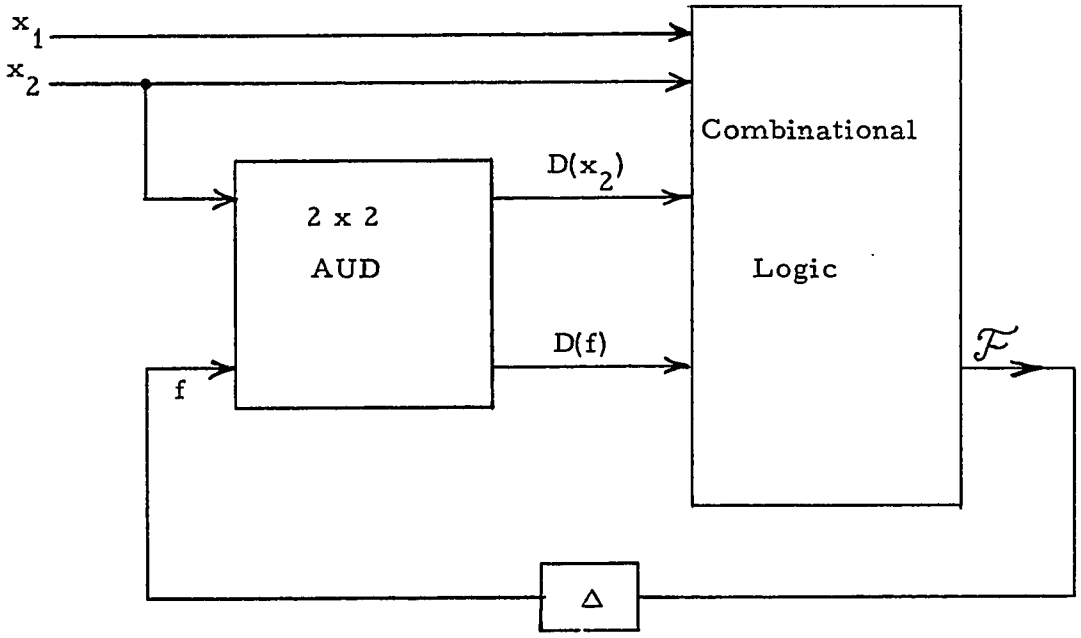


Figure 5.14 .

Because of the presence of many don't care situations, the feedback function and output functions can be expressed in a number of ways in this type of circuit realization. The presence of don't care entries also helps in obtaining minimum expressions for feedback and output functions. Since the use of inertial delay elements is inevitable one could make use of their intrinsic memory property by connecting their outputs to the combinational network in addition to the inputs of the AUD. It is obvious that this does not change the value of feedback index. This could result in even a wider choice of different minimal Boolean functions for feedback and output of a circuit, since now $\mathcal{F}_i$ and Z_j could be chosen as functions of $X, f, D(X), D(f)$. In practice this may be of some help in avoiding fan out due to excessive loading on some delayed variable terminals $D(X_i)$ or $D(f_j)$, by fixing some terms in the Boolean expressions for outputs in variables f_k and f_ℓ instead of $D(X_i)$ of $D(f_j)$.

Remarks

The circuits realized as a result of Theorem 5.2 are non-trivial and have many useful characteristics. The value of the feedback index is the same[12] as that obtained by Huffman[5], Eichelberger [12] and Friedman [13]. Inertial delay elements are necessary as in the case of circuits realized by Huffman's and Friedman's techniques, but in our realization the number of inertial delay elements required is minimum[12] $\geq [\log_2 \max (S_i)]$. The flow tables to be realized need not be normal. From the point of view of machine decomposition single AUD circuits consist of two smaller circuits, A_1, an AUD which is sequential and A_2 which is purely combinational. In general,

---

12 - This is when internal circuit structure of an AUD is not taken into consideration. That is, AUD is a basic circuit element.

since A_1 is a standard predetermined circuit the designer needs to construct A_2 only. The method suggested for the design of A_2 is always easy to apply.

5.3 Single Loop Realization of FMA Machines :

In Chapter 4, the problem of deciding whether a given FMA machine required no feedback (asynchronous definite) was solved. Here we shall discuss another anologous case in FMA machines of a recent result due to Friedman [14] in synchronous sequential machines. The result is as follows : Every synchronous sequential machine can be realized by a circuit with a single binary feedback loop. In view of our discussion of single AUD circuit realization in this chapter, this means that instead of storing m binary signals, $m \geq [\log_2 \max(S_i)]$ along with n binary inputs prior to last input/output change, it is possible to store one binary output signal along with n binary inputs for the last k input/output changes. Therefore, if for every FMA table F there exist an equivalent FMA table F_e which can be realized by a circuit of k AUD's and combinational logic with feedback index 1, an analogous result for FMA machines is also true.

The existence of such an FMA table F_e equivalent to any FMA table F can be proven as follows : Any FMA flow table belongs to a special class of tables where similar successive inputs can never occur. Cadden [10] classifies them as level input - level output (LL) tables and suggests a method of transformation of LL table to its equivalent pulse input-pulse output (PP) table. Friedman shows that for every synchronous table F' (which according to Cadden is a PP table) there exists an equivalent synchronous table F'' such that it is realizable by a synchronous circuit with feedback index 1. Therefore by following Friedman's method of deriving F'' from transformed

PP table F', one can find a binary feedback function f(Y,X) on F''. Using this f(Y,X) one can determine the present state of F'' by the last k values of input and f(Y,X) for some finite k.

In order to realize the original FMA flow table with feedback index 1, one can transform the PP table F'' with its feedback function as output again to its equivalent LL table by following Cadden's method PP to LL transformation. This LL table is the required equivalent FMA table F_e realizable by a single loop asynchronous circuit of $k(n+1) \times (n+1)$ AUD's. Since such a transformation from LL to PP and back to LL equivalent table is always possible, we can state the following theorem without furnishing the necessary formal proof.

Theorem 5.3 : Any FMA indefinite flow table F can be realized by an asynchronous circuit of feedback index 1, with $k(n+1) \times (n+1)$ AUD's and combinational gates.

Since certain details in the actual construction of asynchronous circuit do vary from its analogous synchronous case, we shall demonstrate these details with the help of the following illustrative examples.

Example

	x = 0	x = 1
1	(1)	2
2	3	(2)
3	(3)	4
4	5	(4)
5	(5)	6
6	7	(6)
7	(7)	8
8	1	(8)

	x = 0	x = 1
1	0	C
2	0	0
3	1	1
4	1	1
5	1	1
6	0	0
7	0	0
8	1	1

f(Y,X)

Figure 5.15

Consider a table and its feedback function of Figure 5.15. Assuming the feedback function as an auxiliary output function, we construct its modified input/output successor graph $G_m(F)$ as shown in Figure 5.16. In the second level of stable states of $G_m(F)$ each node contains one and only state. In the synchronous case, such a graph does provide a unique code for the internal states of any finite memory machine, but here it does not. This is because a second level of stable state $G_m(F)$ implies the presence of a single 2 x 2 AUD with single input changes only in the circuit and such an AUD has only 4 internal states, therefore it is impossible to obtain a unique code for 12 states. For example, stable state 2 and unstable state 8a will have the same code, if such a code is derived from the 2nd level $G_m(F)$.

Hence $G_m(F)$ is further extended to 3rd level of stable states. Now we assign to each appearance of Y_i (stable or unstable) in the 3rd level of $G_m(F)$ the reverse of input/output sequence of length 3 required to map on $G_m(F)$, because a 3 level $G_m(F)$ implies two 2 x 2 AUD connected in cascade and such a register stores the input/output values prior to last two changes. The assignment thus obtained is unique and is shown in the table of Figure 5.17. The corresponding circuit is in Figure 5.18.

In general, to obtain a unique state assignment from $G_m(F)$ it may be necessary to extend $G_m(F)$ to many levels of stable states beyond the level, where each node has one and only one state. This is where FMA case differs from the analogous synchronous case.

To demonstrate that there are cases when indefinite FMA tables cannot be realized by a circuit of feedback index 1 directly as was the case in the previous example consider the flow table F_4 of Figure 5.19. It is clear that at least one of the states out of 1, 2, 3, 4, 5 must be split.

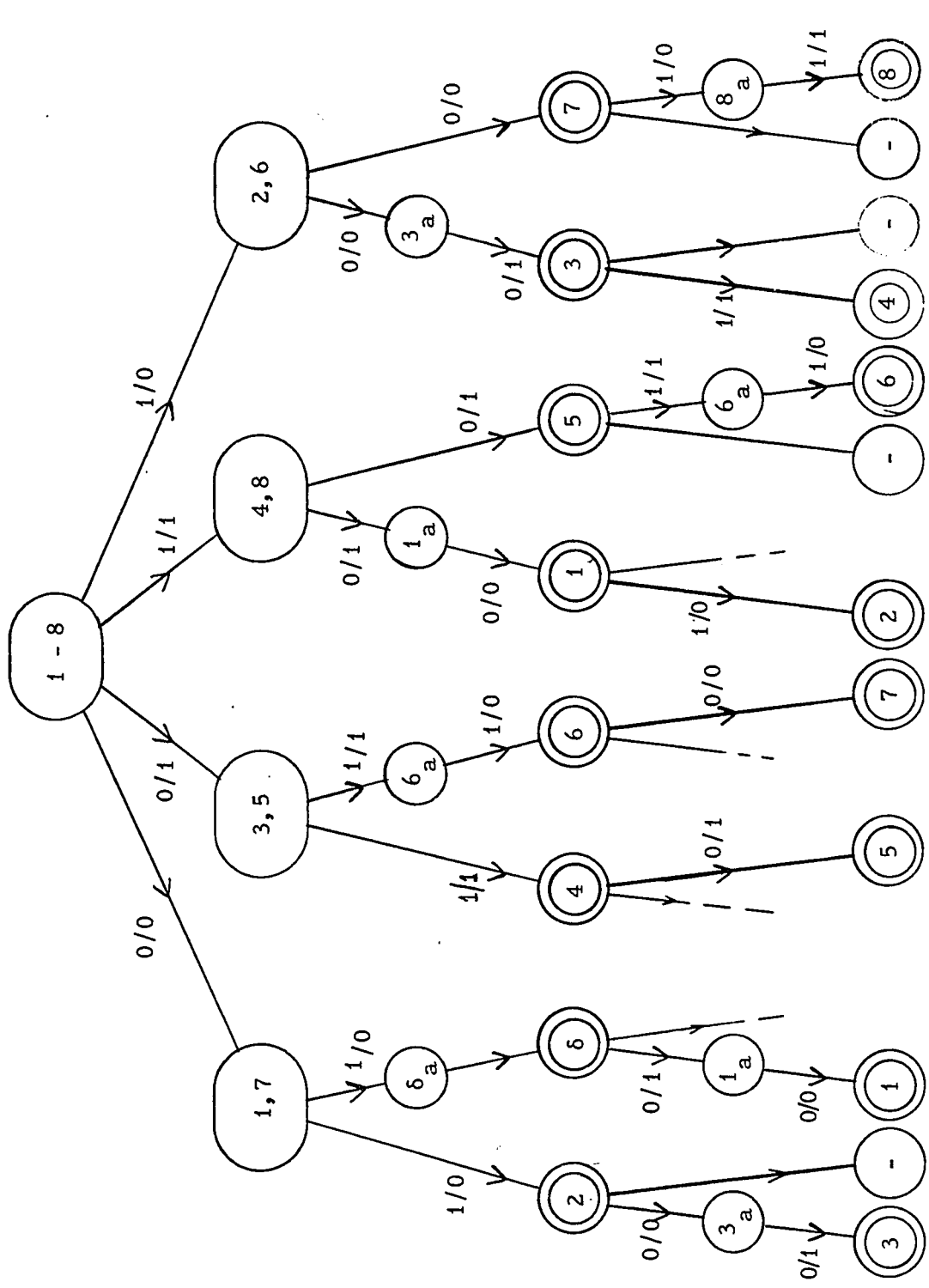

1 - 8
1,7
3,5
4,8
2,6
0/0
1/0
0/1
1/1
1/0
0/0
1/1
0/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1
0/0
1/0
0/1
1/1

Figure 5.16.

Internal State	X	D(X,f)		D^2(X,f)		Present f.b. fn.	Next f.b. fn.
	x	D(x)	D(f)	D^2(x)	D^2(f)	f	$\mathcal{F}$
3	0	0	0	1	0	1	1
3_a	0	1	0	0	0	0	1
1	0	0	1	1	1	0	0
1_a	0	1	1	1	0	1	0
5	0	1	1	0	1	1	1
7	0	1	0	1	1	0	0
6	1	1	1	0	1	0	0
2	1	0	0	0	1	0	0
6	1	1	1	0	1	0	0
6_a	1	0	1	1	1	1	0
4	1	0	1	0	0	1	1
8	1	1	0	0	0	1	1
8_a	1	0	0	1	0	0	1

$$\mathcal{F} = \overline{D(f)} \quad D \overline{(x)} + \overline{x} \; \overline{D^2(x)} + x \; \overline{D^2(f)}$$

Figure 5.17

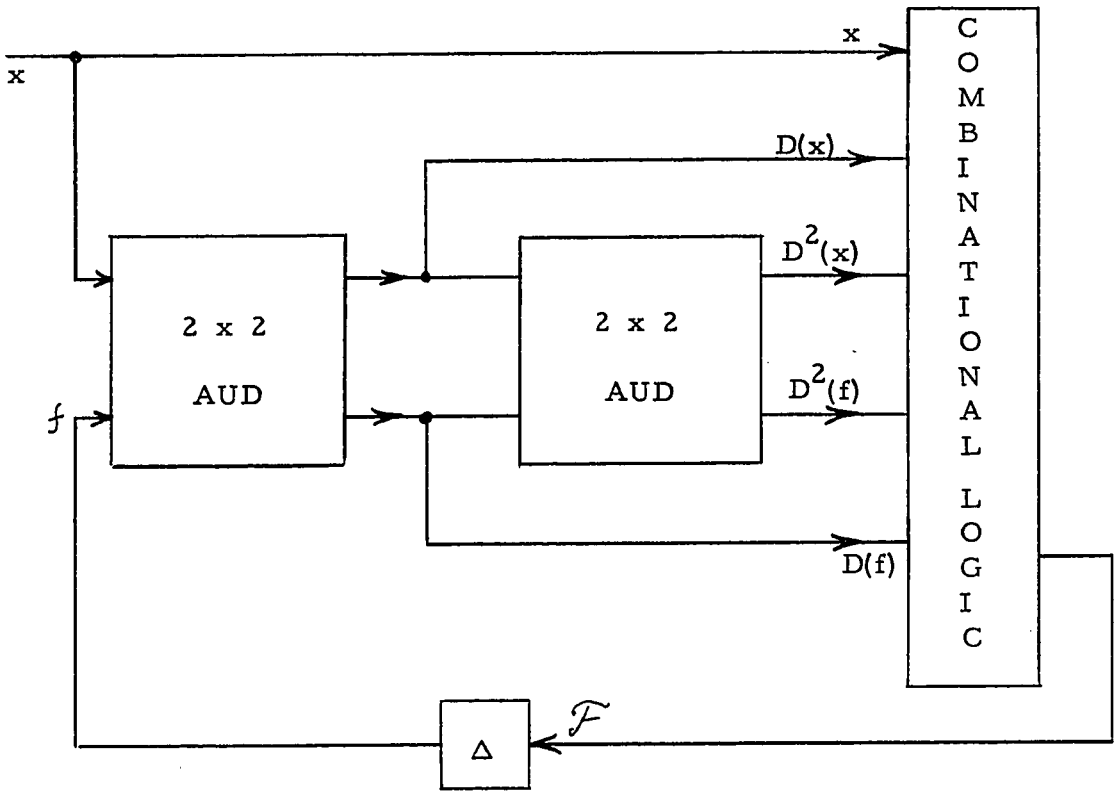


Figure 5.18

Here state 5 of F is being split to $5_a, 5_b, 5_c$ and 5_d to obtain an equivalent flow table F', Now F' can be realized by a single binary loop circuit.

Example

	x_1x_2 00	01	11	10
1	(1),00	(1),01	7 , -	6 , -
2	(2),01	(2),11	8 , -	6 , -
3	(3),10	(3),00	9 , -	6 , -
4	(4),11	(4),11	7 , -	6 , -
5	(5),00	(5),11	6 , -	10, -
6	5 , -	2 , -	(6),0	(6),0
7	- , -	3 , -	(7),01	10, -
8	- , -	4 , -	(8),11	10, -
9	- , -	1 , -	(9), 1	10,-
10	1 , -	- , -	6	(10),10

Figure 5.19

5.4 Advantages of Reducing Feedback :

From the point of view of maintenance of synchronous or asynchronous sequential circuits it would be desirable to be able to block all the feedback loops by the use of interrupt circuitry [14]. This will enable us to insert external test signals to monitor the outputs of the feedback lines and circuit outputs to detect if the circuit is operating properly. This will be advantageous only if the test circuitry is least complex and constructed with a minimum number of circuit components. Therefore, it is desirable that the number of feedback loops in the main circuit should be minimum.

x_1x_2	00	01	11	10
1	(1),0	(1),0	7,0	6,0
2	(2),0	(2),1	8,1	6,0
3	(3),1	(3),0	9,0	6,1
4	(4),1	(4),1	7,1	6,1
5_a	(5_a),1	5_b,1	-,-	10,1
5_b	5_c,1	(5_b),1	6,1	-,-
5_c	(5_c),0	5_d,0	6,1	10,0
5_d	5_a	(5_d),0	6,0	-,-
6	5_c,0	2,0	(6),0	(6),0
7	-,-	3,0	(7),0	10,0
8	-,-	4,1	(8),1	10,0
9	-,-	1,1	(9),1	10,1
10	1	-,-	6,-	(10),1

Figure 5.20 .

If we construct asynchronous circuits with AUD's and combinational network with feedback index 0 or at most 1, then it is clear that such circuits will be easiest to maintain. But such a circuit in general, may need many AUD's, therefore the cost of the circuit becomes very high. Also, the length of the asynchronous shift register may reduce the overall speed of the circuit. Therefore from the point of view of a compromise among maintenance, cost and speed we must also make a compromise between number of feedback loops and number of AUD's in the circuit. One of the many possible choices could be a single AUD circuit with feedback index equal to $[\log_2 \max (S_i)]$.

5.5 Pulse Controlled FMA Circuits

One common technique for designing digital systems makes use of master-clock circuit, which produces a periodic train of pulses. The operation of each circuit constituting the system is synchronized with these clock pulses. In the literature, if one of the input is a clock pulse, while all other inputs are levels and change asynchronously, such circuits are said to be fundamental mode clocked circuits [22]. In this section we shall show that the pulse input need not be periodic, and can be derived with the help of level signals which are allowed to change in asynchronous fashion.

The principle of operation of this type of FMA circuits is as follows :

Whenever a level input signal is changed, it is detected by a change detector circuit and one pulse per input change is generated. The generated pulse is delayed by a length of time Δ, depending upon the delay involved in the combinational logic in computing the output and the next state functions. Once the next state variables reach the steady state values, they are allowed to appear at the input terminals

of combinational logic as present state variables with the help of shift circuits operating under the control of a delayed pulse. The circuit schematic of this class of FMA circuits is shown in Figure 5.22.

Realization of an Ideal Inertial Delay

The reader must note that in doing so, we have realized an ideal inertial delay of magnitude Δ, which controls the operation of the circuit. This can also be interpreted as providing some kind of synchronization[13] to the circuit, though none of the input signals are changing in any periodic fashion.

Generally the role of the inertial delay in the asynchronous circuits is performed by a simple R-C integrating circuit, though this device is far from an ideal one. If the shape of the binary signal in the circuit is an important consideration, then a simple R-C circuit cannot be used, because an ideal inertial delay is not simply a smoothing device. If the input to an ideal inertial delay of magnitude Δ consists of a series of 1 and 0 pulses of width $\Delta - \varepsilon$ for $\varepsilon > 0$ as shown in Figure 5.23, then the output of the ideal delay must remain in its initial state unless a pulse of magnitude $\geq \Delta$ occurs. Such an operation is derivable from a circuit of Figure 5.24. The shift circuit of S-R flip-flop is the same as that of an ideal synchronous unit delay operating with external clock pulses. Here instead of periodic pulses, a single pulse generated from the input change detector circuit, which is further delayed by some arbitrary length of time Δ, is being used to control the operation of the circuit.

---

13 - For example, see Miller, "Switching Theory, Vol. 2, Sequential Circuits and Machines". Chapter 9, p. 141, John Wiley and Sons, Inc., N.Y. 1965.

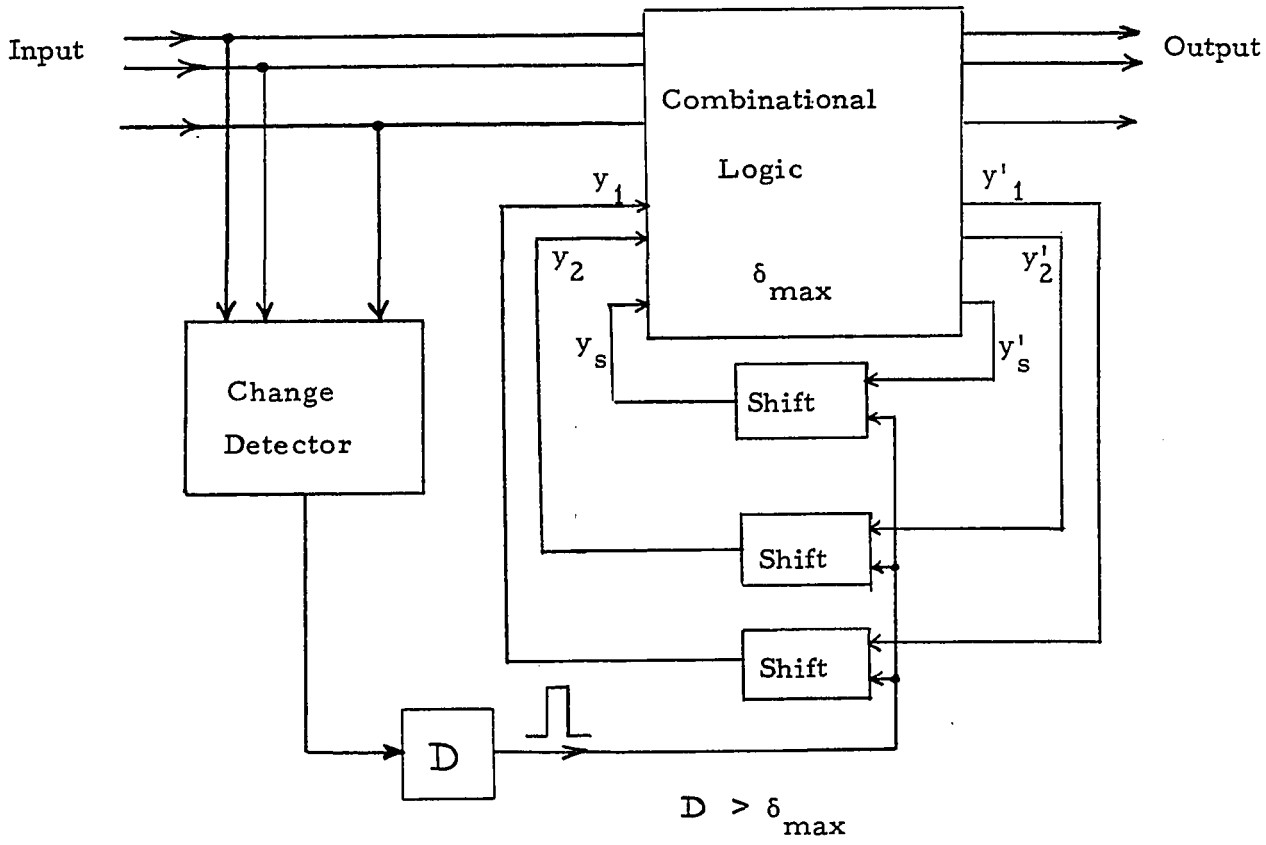


Figure 5.22 - Circuit schematic of FMA Pulse controlled circuit

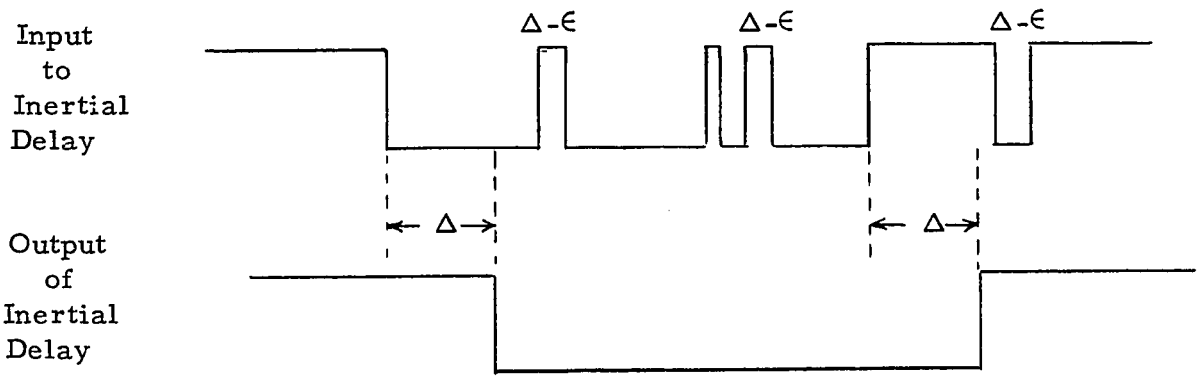


Figure 5.23.

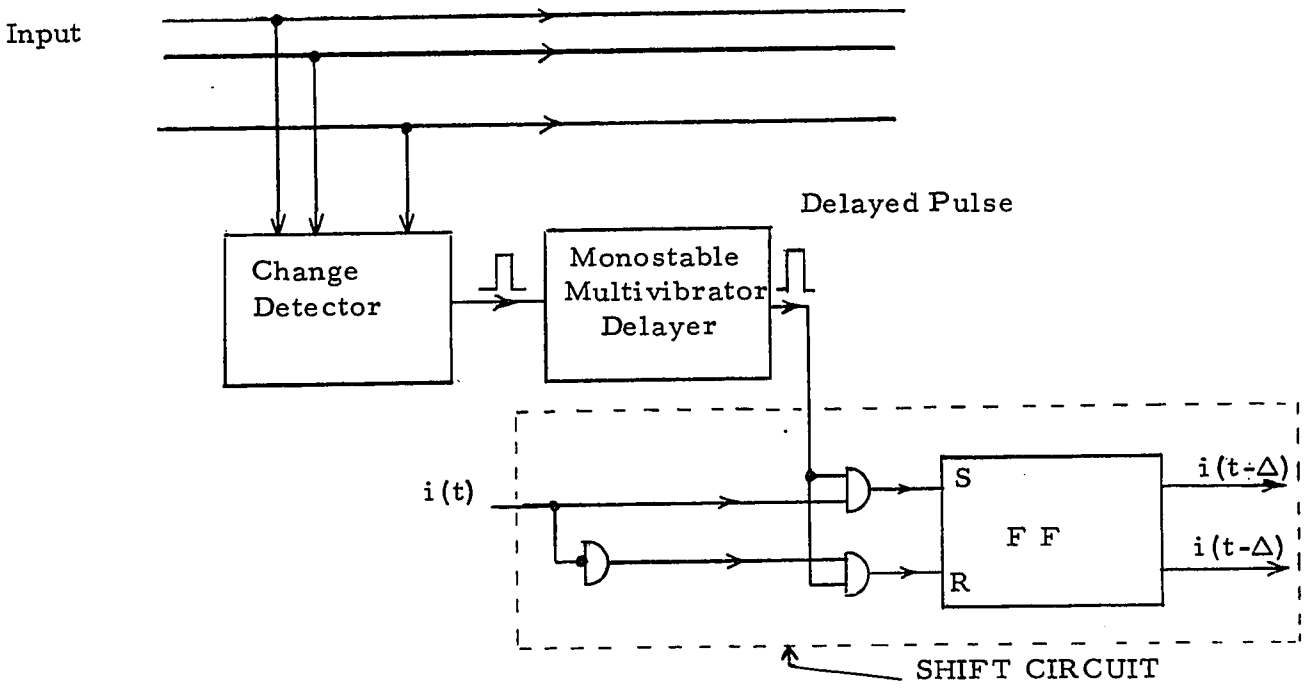


Figure 5.24.

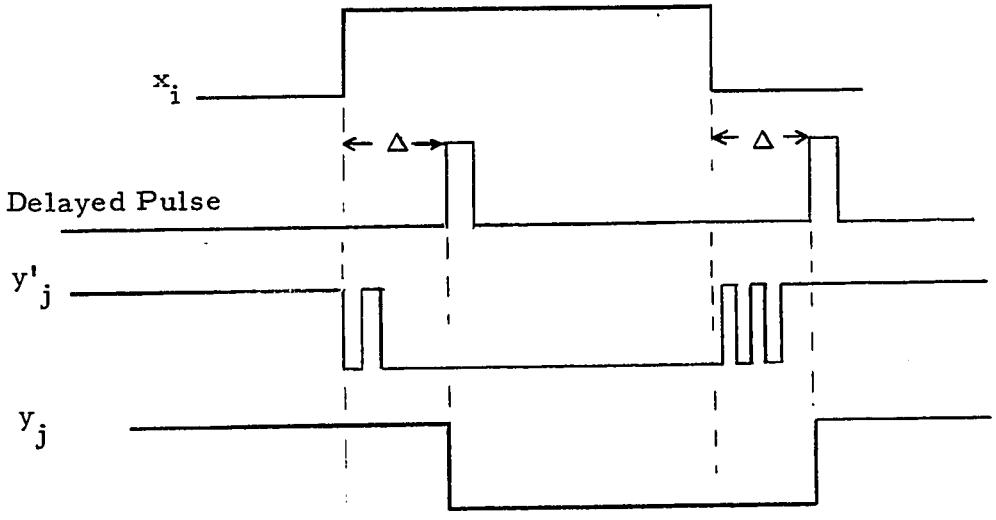


Figure 5.25.

Remarks :

Basically there is no difference between clocked fundamental mode circuits and the circuits with a change detector. Therefore asynchronous circuits with a change detector will have all the advantages of synchronous circuits with regard to (i) minimal state assignment problems, as the question of critical races does not enter in these types of circuits, (ii) transient hazards which could cause steady state hazards automatically disappear (see Figure 5.25) and (iii) circuits with all input changes can be easily designed, etc. .

Chapter 6

CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY

In this dissertation we have presented an approach to the design of asynchronous circuits operating under fundamental mode, in an attempt to bring closer the synchronous and asynchronous switching theories. In doing so, first we have described the AUD device analogous to synchronous unit delay and then prescribed methods to construct it. Then as in synchronous case, we have classified asynchronous fundamental mode flow tables as (i) definite and (ii) indefinite. It has been shown that all definite tables are realizable by feedback-free circuits of AUD's and combinational network, while circuits with AUD's realizing indefinite tables do require feedback.

It has also been shown that an AUD can be used as a basic circuit element in the general asynchronous circuit design. Any FMA table can be realized by a circuit of feedback index m with one AUD, m inertial delay elements and a combinational network, where $m \geq [\log_2 \max(S_i)]$; S_i being the number of stable states in any column of the table. As a special case, for a given state assignment of any FMA machine, if we choose the feedback function equal to its next state function, then the presence of an AUD in the circuit does not contribute anything towards its operation. We have also shown that any indefinite FMA machine is also realizable by a circuit of feedback index one with k AUD's, one inertial delay element and a combinational network.

In an attempt to obtain a reduced flow table of nxn AUD with single input changes only, we have developed a code $C_n^{(p)}$, where

every pair of words $w_i, w_j \in C_n^{(p)}$ has either the Hamming distance $d(w_i, w_j) = 1$ or ≥ 3.

Lastly, we have shown that if a change detector circuit is provided, then one delay element with many shift circuits is sufficient to realize any properly operating circuit and such a circuit has all the advantages of synchronous circuits.

The following problems may be worthy of further investigation:

(1) Development of methods to make the single loop circuit realization more economical in terms of logical hardware, where instead of lumping the whole combinational logic, it could be interspersed between AUD's. This could be quite an interesting problem, because the input restriction to the different AUD's (if AUD's to be used are designed for single input changes only) is also to be simultaneously satisfied by the interspersed logic.

(2) Extension of the present approach to non-fundamental mode asynchronous flow tables.

REFERENCES

(1) Armstrong, D.B., Friedman, A.D. and Menon, P.R., "Synthesis of asynchronous sequential circuits with minimum number of delay elements". Proc. Eighth Ann. Symp. on Switching and Automata Theory, Austin, Texas, pp. 95-105, 1967.

(2) Batra, V.K., "Design of Asynchronous Unit Delay", M.Sc. Thesis, Department of Electrical Engineering, University of Ottawa, June, 1967.

(3) Brzozowski, J.A., "Canonical regular expressions and minimal state graphs for definite events", Proc. Symp. on Mathematical Theory of Automata, Polytechnic Press of the Microwave Research Institute Symp.. series, Polytechnic Inst. of Brooklyn, N.Y., 12, pp. 529-561, 1962.

(4) Brzozowski, J.A., "An Essay on Feedback", Dept. of Elect. Engg., The University of Ottawa, Canada, Tech. Report No. 65-2 ; March, 1965.

(5) Brzozowski, J.A. and Hawtrey, P., "Logic-free realization of sequential machines, Dept. of Elect. Engg., The University of Ottawa, Canada, Tech. Rept. No. 66-9 ; August, 1966.

(6) Brzozowski, J.A., "On single loop realizations of sequential machines", Information and Control, Vol. 10, pp. 292-314, March, 1967.

(6a) Brzozowski, J.A. and Singh, S. "Definite asynchronous sequential circuits", Department of Electrical Engineering, The University of Ottawa, Canada, Tech. Report No. 67-3, Feb., 1967. To appear in IEEE Trans. on Elec. Comp. , January, 1968.

(7) Bose, R.C., and Ray-Chaudhuri, D.K., "On a class of error-correcting binary group codes", Information and Control, Vol. 3, pp. 68-79, 1960.

(8) Bose, R.C., and Ray-Chaudhuri, D.K., "Further results on error-correcting binary group codes", Information and Control, Vol. 3, pp. 279-290, 1960..

(9) Bose, R.C. and Ray-Chaudhuri, D.K., "Op. cit.", p. 289, Table I.

(10) Cadden, W.J., "Equivalent sequential circuits", IRE Trans. Circuit Theory, Vol. C.T.-6, pp. 30-34, March, 1959.

(11) Caldwell, S.H., "Switching Circuit and Logical Design", John Wiley and Sons, Inc., New York, 1958.

(12) Eichelberger, E.B., "Sequential circuit synthesis using hazards and delays", Digital System Tech. Report 19, Dept of Electrical Engineering, Princeton University, Princeton, N.J., June, 1962.

(13) Friedman, A.D., "Feedback in asynchronous sequential circuits", IEEE Trans. on Electronic Computers, Vol. EC-15, pp. 740-749, October, 1966.

(14) Friedman, A.D. "Feedback in synchronous sequential circuits", IEEE Trans. on Electronic Computers, Vol. E.C. - pp. 354-367, 1966.

(15) Gill, A., "Introduction to the Theory of Finite State Machines", McGraw-Hill Book Co. Inc. ; 1962.

(16) Hamming, R.W., "Error detecting and error correcting codes", Bell System Technical Journal, Vol. 29, pp. 147-160, April, 1950.

(17) Hartmanis, J. and Stearns, R.E., "Algebraic Structure Theory of Sequential Machines", Prentice Hall, Inc., N.J., 1966.

(18) Huffman, D.A., "The synthesis of sequential circuits", Pt. I and II, J. of Franklin Institute ; Vol. 257, pp. 161-190 and 275-303, March and April, 1954.

(19) Huffman, D.A., "A study of the memory requirements of sequential switching circuits", Research Laboratory of Electronics, Massachusetts Institute of Technology, Tech. Report 293, March, 1955.

(20) Liu, C.N., "A State variable assignment method for asynchronous sequential switching circuits", JACM, Vol. 10, pp. 209-216, April, 1963.

(21) McCluskey, E.J., "Reduction of feedback loops in sequential circuits and carry leads in iterative circuits", Proc. 3rd. Ann. Symp. on Switching Circuit Theory and Logical Design, Chicago, Ill., pp. 91-102, 1962.

(22) McCluskey, E.J., "Introduction to the Theory of Switching Circuits", McGraw-Hill Book Co., Inc., 1965.

(23) Miller, R.E., "Switching Theory, Vol 2 : Sequential Circuits and Machines", John Wiley and Sons, Inc., N.Y., 1965.

(24) Muller, D.E., "Asynchronous logics and application to information processing", Proceedings of a Symposium of the Application of Switching Theory in Space Technology, Stanford University Press, March, 1962.

(25) Paul, M.C. and Unger, S.H., "Minimizing the number of states in incompletely specified sequential switching functions", I.R.E. Trans. on Electronic Computers, Vol. E.C.-8, pp. 356-367, Sept., 1959.

(26) Pearles, M., Rabin, M.O. and Shamir, E. "The theory of definite automata", IEEE Trans. Electronic Computers, E.C.-12, pp. 233-243, June, 1963.

(27) Peterson, W.W., "Error Correcting Codes", John Wiley and Sons, N.Y., 1961.

(28) Peterson, W.W., Op. cit., pp. 11-19.

(29) Reza, F.M., "An Introduction to Information Theory" McGraw-Hill Book Co., N.Y., pp. 171-174, 1961.

(30) Reza, F.M., Op. cit., p. 447.

(31) Rotman, J.J., "The Theory of Groups : An introduction", Allyn and Bacon, Inc., Boston, 1965.

(31a) Shiva, S.G.S., "Lecture Notes on Theory of Coding with Applications", Dept. of Electrical Engineering, University of Ottawa, Ottawa, Canada.

(32) Tracy, James H., "Internal state assignment for asynchronous sequential machines", IEEE Trans. on Electronic Computers, Vol. E.C.-15, pp. 551-560, Aug., 1966.

(33) Unger, S.H., "Hazards and delays in asynchronous switching circuits", I.R.E. Trans. on Circuit Theory, Vol. C.T.-6, pp. 12-25, March, 1959.

(34) Wax, N., "On upper bounds for error detecting and error correcting codes of finite length", I.R.E. Trans. on Information Theory, Vol. IT-5, p. 170, Table I, December, 1959.

VITA

Name : Shanker Singh

Born : 5-5-1936 Agra, India

Educated :

Primary : Government Normal School, Agra

Secondary : Balwant Rajput Higher Secondary School, Agra

University :

(a) Agra College, (Agra University)

1954 Bachelor of Science

1957 Master of Science in Physics

(b) Birla Institute of Technology and Science, Pilani, India

1959 Master of Science (Technology) in Electronics