CONTRIBUTIONS TO STRUCTURE THEORY OF SEQUENTIAL MACHINES

by

Wayne A. Davis, B.S.E., M.Sc.

Submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Department of Electrical Engineering
Faculty of Pure and Applied Science
University of Ottawa
Ottawa, Ontario
October, 1966

ABSTRACT

This thesis is concerned with the relationship between structure and behavior of sequential machines. Sequential machines are introduced and the fundamentals of machines, partitions and set systems are presented. A brief survey of structure theory is outlined. The problem of state assignments for combinational circuits is considered and an approach to the solution of the problem is demonstrated. Linear sequential machines are studied and methods are given for determining whether or not a machine is realizable as a linear machine. Shift registers are introduced and methods are presented for finding shift register realizations for given sequential machines. The thesis concludes with a short outline of remaining problems.

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to Dr. Janusz A. Brzozowski, of the Electrical Engineering Department, for his advice, criticism and many stimulating discussions on various parts of this thesis.

The assistance is acknowledged of Dr. Juris Hartmanis, of the Department of Computer Science of Cornell University, who provided the original impetus and ideas for the joint paper [42], the results of which appear in various portions of Chapter 4.

Gratitude is also expressed to my colleagues the graduate students in the Electrical Engineering Department, and in particular: Shanker Singh, John McEntyre and Peter Hawtrey.

The research presented in this thesis was performed in the Communications Laboratory (CL), Defence Research Telecommunications Establishment (DRTE) of the Defence Research Board under PCC D48-28-01-07. The author is grateful to Dr. Brzozowski and Prof. G.S. Glinski of the Electrical Engineering Department for making possible this co-operative research. The author is indebted to DRTE for financial

support, and particularly to: Mr. W.L. Hatton, Section Leader of the Communications Systems Section, for his unending patience, understanding and support; Dr. J.H. Meek, Superintendent of CL, for his support; Dr. J.H. Chapman, Deputy Chief Superintendent of DRTE, for his assistance; Mr. F.T. Davies, Chief Superintendent of DRTE, for his encouragement, particularly in the final phases; Mr. W.K. Nicholson, now at Stanford University, for his assistance; a host of others, too numerous to mention by name, for their patience and encouragement.

The author is particularly grateful to Mrs. Beverley Lake, who typed the final manuscript; to the multitude of typists and secretaries in CL who typed the seemingly endless number of drafts; to the various draftsmen and secretaries, who produced the diagrams.

CONTENTS

CONTENTS (Continued)

Chapter 1

INTRODUCTION

This thesis considers the relationship between structure and behavior of sequential machines. The results have application in the synthesis of digital systems. In modern design and construction of electronic systems, digital circuits are playing an ever increasing role. The use and application of digital circuits can be attributed not only to advances in digital computing, but also to the relative ease by which digital circuits may be fabricated and used. The advent of modern electronic computers has revolutionized the field of information handling and processing. Since we now have the ability of processing large amounts of data in very short periods of time, it is natural that digital systems are used not only for final processing, but also for the collection of data. In addition, this permits us now to do a limited amount of preprocessing, in order to save costly computer time. Increased speeds in digital circuits also permit them to compete effectively against analogue circuits in the field of automatic control.

The over-all result of this increased use of digital circuits has been that more and more people are synthesizing more and more systems. This, in turn, has provided a greater impetus to the switching circuit theorist to provide better and more complete algorithms for logical design. One problem which has continued to plague both the logic designer and the theorist is the problem of optimum synthesis. Synthesis is the process of converting an abstract requirement, or behavior, into a working piece of hardware, or structure, that uses a restricted set of devices. The specification is usually nebulous, very often incomplete and possibly contradictory. The first step in the process is to turn this requirement into an unambigious mathematical specification. The second step is to convert the mathematical specification into a hardware realization.

While, in general, for a given problem this may not be too difficult since a brute force approach is usually available, trouble arises when one tries to optimize the realization subject to physical restrictions. This problem has recently become more important and apparent due to technological changes, because structure is dependent on the actual circuits used and hence technology dependent. This, however, introduces a difficulty, since if synthesis techniques become too dependent on the circuits used, then they will only be useful as long as that particular set of circuits is employed. We therefore want to consider synthesis, and hence structure, in a sense that is general enough to be adaptable to changes in technology, and yet specific enough to be of practical use in a good many synthesis problems.

It should be noted that it is doubtful that there ever will exist a structure theory or synthesis procedure that is capable of handling all structures.

The behavior of a sequential machine is the relationship between the inputs and the outputs of the machine. The state behavior of a sequential machine refers to the next-state transition function of the machine. The structure of a sequential machine is the composition of the machine from its parts, i.e., the actual interconnections between building blocks. At this point we should note that while the definitions of structure and behavior in the literature [1,6,9,33,37,41,61,63] may not seem to be consistent, they in effect are, because the differences are only between the level at which the structure is considered. For example: Structure as presented by Hartmanis and Stearns [41] is the realization from "sets of smaller component machines and how these component machines have to be interconnected and how information flows in and between these smaller machines". Yet structure, as presented by Brzozowski [6], refers to the actual circuit (idealized) of the machine. Now if a machine is decomposed into two smaller submachines, then this is certainly structure in the Hartmanis and Stearns sense; however, since nothing is said about the circuit it would seem not to be structure in the Brzozowski sense. Nevertheless, if we consider circuits for both machines, i.e., structure in the Brzozowski sense, then we notice that there is indeed a difference in the structure of the two circuits. To summarize then, a structural change in the Hartmanis and Stearns sense did produce a structural change in the Brzozowski sense without actually mentioning the circuit. We con-

clude that there are various levels of structure that are useful and that all are interrelated. On the other hand, structure has been used [25] to denote the next-state function. It is felt that this is actually a behavioral description, and will be considered as such.

This thesis considers several structural problems. Following this chapter, Chapter 2 briefly summarizes the current results in structure of sequential machines, in an attempt to give the reader a perspective by which he can adequately evaluate the remaining results.

Chapter 3 presents an approach to a problem on the structure of combinational circuits in the output logic of a sequential machine. In other words, given a sequential machine, is it possible to determine an assignment for the internal variables, that minimizes the output logic. Furthermore, it is also shown that the same approach can also be applied to the process of finding an internal variable assignment which minimizes the combinational logic for the next state function.

In Chapter 4, methods are presented for realizing sequential machines as linear machines. Existing methods are summarized in detail, and their various differences and similarities are pointed out along with their short-comings. A linearity algorithm is then presented for machines having their output specified. Next, machines whose outputs are either unspecified or are nonlinear are considered. These are initially treated for one-to-one assignments and then the methods are extended to many-to-one assignments.

Chapter 5, considers realizations of sequential machines using shift registers. Initially the existing methods are outlined with their

relative merits and failings. Single binary shift registers are then investigated and methods are given for determining shift register assignments. It is proven that there exists a sequential machine that cannot be realized by a single binary shift register of finite length. Methods for determining the set systems necessary for a single shift register assignment are given. For machines in which every state has two distinct next states, a method is presented whereby a number of machines can be eliminated as being not realizable as a single shift register. It is then shown how this can be extended to all machines. Multiple shift registers are then considered and techniques are presented for determining multiple shift register assignments. The chapter concludes with a short exposition on logical design techniques for shift register assignments.

1.1 Fundamentals

In this section the basic material that is common to all chapters is presented. The reader is referred to [41,37,63] for additional details.

Definition 1.1: A sequential machine is a quintuple $M = (S,I,O,\delta,\lambda)$ where S, I, O are respectively the non-void finite sets of internal states, inputs and outputs: $\delta : S \times I \rightarrow S$ is the next-state function, and $\lambda : S \times I \rightarrow O$ is the output function.

We restrict ourselves to sequential machines that are fixed, finite, deterministic, discrete and synchronous.

Definition 1.2: A machine $M' = (S', I', O', \delta', \lambda')$ is a submachine of a machine $M = (S, I, O, \delta, \lambda)$ if, and only if $S' \subset S$, $I' \subset I$, $O' \subset O$ and $\delta' = \delta$ restricted to $S' \times I' \to S'$, $\lambda' = \lambda$ restricted to $S' \times I' \to O'$.

Definition 1.3: A machine $M' = (S', I', O', \delta', \lambda')$ is a homomorphic image of the machine $M = (S, I, O, \delta, \lambda)$ if, and only if there exist three onto mappings

$$h_1: \quad S \to S'$$

$$h_2: \quad I \to I'$$

$$h_3: \quad O \to O'$$

such that

$$h_1[\delta(S_i, x)] = \delta'[h_1(S_i), h_2(x)]$$

$$h_3[\lambda(S_i, x)] = \lambda'[h_1(S_i), h_2(x)],$$

for all $S_i \in S$, $x \in I$.

M' is said to be isomorphic to M ($M' \cong M$) if, and only if M' is a homomorphic image of M and h_1, h_2 and h_3 are one-to-one mappings.

We say that a machine M realises a reduced machine M' if and only if M' is the homomorphic image of a submachine of M.

Definition 1.4: A partition, P, over a set S, is a collection of disjoint subsets of S, called blocks, such that their set union is S.

Two states which appear in the same block of a partition P, are said to be identified by P. We denote the trivial partition where each block contains a single state (no states are identified) by 0, and the trivial partition having a single block (all states are identified) by S.

Every homomorphism of M onto M' induces a congruence relation or substitution property (S.P.) partition P on the set of states of M, if we define

$$S_i \equiv S_j(P) \text{ if, and only if, } h_1(S_i) = h_1(S_j).$$

Conversely, every S.P. partition defines a homomorphism of M onto M' with

$$S' = \{b_i | b_i \in P\},$$

$$\delta'(b_i, x) = b_j \text{ for } b_j \text{ such that}$$

$$\delta(S, x) \in b_j, \text{ and } S \in b_i,$$

and

$$h_1(S_i) = \{S_j | S_j \equiv S_i \ (P)\}.$$

For obvious reasons, we restrict ourselves to binary realizations. The problem is one of finding an assignment for the internal states of the given machine such that the resulting realization has the desired structure.

We consider two types of assignments: The first is a one-to-one assignment, where each internal state is assigned a unique binary code. The second is a many-to-one assignment where each internal state can be assigned more than one binary code. Clearly, the former is a special case of the latter; however, because the treatment is considerably simplified, we treat the two cases separately.

It has been shown [41] that one of the dangers of reducing the flowtable of a sequential machine is that we can destroy some of the structure that is exhibited by the behavior of the machine. While this does not happen in every case, as will be demonstrated in latter chapters, when it does occur we must be prepared. This means that, in order to determine a realization for a machine, we must consider homomorphic image machines, and hence many-to-one assignments.

1.1.1 Algebra of Partitions

We use the following notation for operations on sets: Given two sets a and b, their union is denoted by $(a + b)$, their sum (Modulo 2) by $(a \oplus b)$, their product (intersection) by (ab), and the complement of a with respect to the universal set I by $\bar{a}$. The empty set is denoted by φ.

Definition 1.5: If P_1 and P_2 are partitions over S, then $P_1 \leq P_2$ if, and only if, every block b_{1i} of P_1 is contained in some block b_{2j} of P_2 $(b_{1i} \subseteq b_{2j})$. $P_1 < P_2$ if $P_1 \leq P_2$ and $P_1 \neq P_2$.

Definition 1.6: The product* of two partitions $P_1 = \{b_{11}; b_{12}; \ldots; b_{1i}\}$ and $P_2 = \{b_{21}; b_{22}; \ldots; b_{2j}\}$ is a partition given by

$$P_1P_2 = \{b_{11}b_{21}; b_{11}b_{22}; \ldots; b_{1i}b_{2j}\}.$$

We can now find the product of k partitions, and it is easily shown that the product operation is associative and commutative. Since we are restricted to binary assignments, we naturally consider binary partitions.

Definition 1.7: A binary partition (bp), over a set S, is a partition having two blocks, where $S = \{S; \varphi\}$ is considered to be the trivial binary partition.

A binary partition is completely determined by one of its blocks b, since the other block is simply $\bar{b}$, the complement of b with respect to S. It is clear that the set β of all binary partition blocks over a set S, forms a Boolean algebra under the operations of sum, product and

---

* called the g.l.b. by Hartmanis [41]

complement and furthermore, β is isomorphic to the algebra of all possible binary variables for S.

For minimal assignments, where we are restricted to the smallest number of binary variables possible, we qualify the bp's that can be used.

Definition 1.8: A qualified binary partition (qbp), over a set S containing q elements, is a binary partition $P = \{b;\bar{b}\}$ such that $\#(\tilde{b}) \leq 2^{n-1}$, where $n = [\log_2 q]$, $\tilde{b} = b$ or $\bar{b}$, $\#(b)$ represents the number of elements in the block b, and $[N]$ is the smallest integer $\geq N$.

We have defined those partitiions that can be used individually in an assignment; we now consider those bp's that can be used together in a valid assignment.

Definition 1.9: m bp's $P_1, P_2, \ldots, P_m$ over a set S containing q elements, are said to be mutually consistent for a fixed $n \geq [\log_2 q]$ if, and only if,

$$\#(\tilde{b}_1 \tilde{b}_2 \ldots \tilde{b}_m) \leq 2^{n-m}$$

for all possible $\tilde{b}_j$, where $1 < m \leq n$.

This definition is equivalent to saying that the number of states in the largest block of $P_1 P_2 \ldots P_m$ is less than or equal to 2^{n-m}. We now state the fundamental requirement of all valid assignments.

Theorem 1.1: An assignment of $n \geq [\log_2 q]$ binary variables $y_1, y_2, \ldots, y_n$ for a set S is valid if, and only if, each y_i defines a binary partition P_i over S such that the n bp's so defined are mutually consistent.

Proof: [7].

Since we are dealing with nonautonomous machines we must consider total states, and consequently total state partitions.

Definition 1.10: A total state S_iI_j (or I_jS_i) is an internal state S_i and an input state I_j.

Definition 1.11: The product of a set b of internal states and a set c of input states is denoted by bc and is defined as the set of all total states S_iI_j such that $S_i \in b$, and $I_j \in c$.

We now define two operators: σ, the successor operator and π, the predecessor operator.

Definition 1.12: The successor of a total state S_aI_b is the next internal state S_k for the present internal state S_a and present input I_b, and is denoted by $\sigma(S_aI_b) = S_k$. Similarly, the predecessor of the internal state S_k is the set β of all total states S_iI_j such that $\sigma(S_iI_j) = S_k$, and is denoted $\pi S_k = \beta$.

At this point, it should be pointed out that σ and δ are essentially identical, and the only reason for using both symbols is that σ is the generalized version of δ. In other words, when σ is used it usually refers to sets of total states, while δ usually refers to single total states.

Definition 1.13: The successor of a set of total states is the union of successors of each total state. Similarly, the predecessor of a set of internal states is the union of predecessors of each internal state.

Definition 1.14: The successor of a partition $R = \{d; \bar{d}\}$ on the set of total states is defined as a grouping (or arrangement) of sets of internal states

$$\sigma R = \{\sigma d; \sigma \bar{d}\}.$$

Obviously σR is a binary partition over S only if $\sigma d = \overline{\sigma d}$.

Definition 1.15: The predecessor of an internal state partition $P = \{b;\bar{b}\}$ is defined as the total state partition

$$\pi P = \{\pi b;\ \pi\bar{b}\} .$$

1.1.2 Algebra of Set Systems

Definition 1.16: A set system $\psi = \{b_k\}$ over a set S of states, is a collection of subsets b_k of S called blocks, such that their set union is S.

Set systems differ from partitions in that they do not require the blocks to be disjoint. In Chapter 3 set systems are used as an intermediate representation of the desired partitions. Whereas in Chapters 4 and 5 by manipulating the overlapping blocks we can consider more states (i.e., by state splitting [50]) and hence image machines of the given machine. The set systems defined above differ slightly from those of Hartmanis and Stearns [41]since they do not have the restriction that $\psi_\alpha \subset \psi_\beta \rightarrow \psi_\alpha = \psi_\beta$. However, the Hartmanis and Stearns set systems are clearly included in the above definition.

Definition 1.17: If ψ_1 and ψ_2 are two set systems over S, then $\psi_1 \leq \psi_2$ if every block b_{1i} of ψ_1 is contained in some block b_{2j} of $\psi_2 (b_{1i} \subseteq b_{2j})$, and $\#(\psi_1) \geq \#(\psi_2)$, where $\#(\psi)$ represents the number of blocks in the set system ψ.

Definition 1.18: The product of two set systems

$$\mathcal{Q}_1 = \{b_{11}; b_{12}; \ldots; b_{1i}\} \text{ and}$$

$$\mathcal{Q}_2 = \{b_{21}; b_{22}; \ldots; b_{2j}\} \text{is a set system given by}$$

$$\mathcal{Q}_1\mathcal{Q}_2 = \{b_{11}b_{21}; b_{11}b_{22}; \ldots; b_{1i}b_{2j}\}.$$

As an illustration, consider $S = \{A, B, \ldots, K\}$. Then $Q = \{\overline{AC}; \overline{H,I,J}; \overline{A,E,F}; \overline{B,C,D,J,K}; \overline{E,F,G,H,J,K}\}$ is a set system, whereas $Q_2 = \{\overline{A,C,H,K}; \overline{D,E,I,J,K}; \overline{E,F,G,H,K}\}$ is not because B is not included in any block of Q_2. If $Q_3 = \{\overline{A,B,C,H,K}; \overline{D,E,I,J,K}; \overline{C,F,G,H,K}; \overline{C,K}\}$ then $Q_1Q_3 = \overline{A,C}; \overline{C}; \overline{C}; \overline{H}; \overline{I,J}; \overline{H}; \overline{A}; \overline{E}; \overline{F}; \overline{B,C,K}; \overline{D,J,K}; \overline{C,K}; \overline{C,K}; \overline{H,K}; \overline{E,J,K}; \overline{F,G,H,K}; \overline{K}\}$.

Definition 1.9: A binary set system (bsm) is a set system having two blocks.

Definition 1.20: A qualified binary set system (qsm), Q, is a binary set system over a set S containing q states such that there exists an assignment for the included states which is a qbp.

For example consider $S = \{A,B,\ldots,G\}$. $Q_1 = \{\overline{A,B,C,D}; \overline{D,E,F,G}\}$ is a qsm over S, where D is the only included state, since if we assign D to either block of Q_1, Q_1 becomes a qbp. $Q_2 = \{\overline{A,B}; \overline{B,C,D,E,F,G}\}$ is not a qsm, because if we assign the included state B to either block of Q_2, Q_2 does not become a qbp.

Definition 1.21: m bsm's over a set S containing q elements are said to be mutually consistent, for a fixed $n \geq [\log_2 q]$, if, and only if, there exists an assignment for the included states such that the bp's created by the assignment are mutually consistent for n.

As an example, $S = \{A, B, \ldots, G\}$, $Q_1 = \{\overline{A,B,D,E,F}; \overline{B,C,F,G}\}$, $Q_2 = \{\overline{C,E,F,G}; \overline{A,B,C,D,F,G}\}$ and $Q_3 = \{\overline{B,C,D,E,F,G}; \overline{A,B,D}\}$. It is seen that Q_1, Q_2, and Q_3 are bsm's. Now $Q_1Q_2 = \{\overline{E,F}; \overline{A,B,D,F}; \overline{C,F,G}; \overline{B,C,F,G}\}$, and we see that it is possible to find an assignment such that the number of states in any block of Q_1Q_2 is ≤ 2, hence Q_1 and Q_2 are mutually consistent for $n = 3$. However, $Q_2Q_3 = \{\overline{C,E,F,G}; \overline{B,C,D,F,G}; \overline{A,B,D}\}$ and it is impossible to find an assignment such that the number of states in any block of Q_2Q_3 is ≤ 2, hence Q_2 and Q_3 are not mutually consistent for $n = 3$.

Definition 1.22: σQ is any set system given by $\sigma Q \succ \{\sigma b_1; \sigma b_2; \ldots; \sigma b_r\}$ where $Q = \{b_1; b_2; \ldots; b_r\}$ is a set system.

Definition 1.23: If Q is a set system, then πQ is any set system such that some $\sigma(\pi Q) = Q$.

It should be noted that partitions are used for one-to-one assignments and set systems for many-to-one assignments.

Chapter 2

A SURVEY OF CURRENT RESULTS

The purpose of this chapter is to survey the current results in the field of structure theory for sequential machines. By structure theory we mean the synthesis of a given fixed behavior by a specific structure. While it is known that a machine with a given structure has a fixed behavior, if the behavior is given then many different structures can realize the same behavior. Significant advances have been made recently on the synthesis of sequential machines, but they are in largely uncorrelated form. This chapter attempts to provide a short comprehensive survey of the results.

Before we start to summarize any actual results it will be worthwhile to also summarize the surveys. In other words, we start by considering those papers that could, in some way or another, be called surveys of structure theory. McNaughton [61], in a survey of automata, was one of the first to point out the difference between structure and behaviour,

and to define — even though somewhat loosely — switching theory. He defines switching theory to be "the study of the structure and the relationship between behavior and structure in fixed discrete automata". Other than pointing out the distinction between switching theory and automata theory, including some rather subtle points, McNaughton dwells mainly on the behavioral aspects of automata. Crowley [9], in an unpublished paper, presents the first real survey of state assignment methods. This paper is an excellent survey of the early results on structure theory, even though it is slightly out of date now. Glushkov [33], in a very general paper, presents a brief and somewhat superficial picture of structure theory. He does, however, attempt to relate both automata and switching theory to their applications. Armstrong [1], in a short survey, classifies and lists a large number of papers on structural properties of sequential circuits. He includes a wide range of papers, but merely lists the papers and the significance without including any of the details. Nelson [64], in a survey of automata, includes algebraic methods applied to structural properties such as decomposition and feedback. Brzozowski [6]and Hartmanis [39], in a pair of complementary papers, present surveys directly related to structure theory. Brzozowski discusses, without going into detail, the following structures: feedback, linear machines, shift registers, realizations using threshold logic and logic with inherent delays. Hartmanis, on the other hand, discusses, also without going into detail, decomposition,

loop-free structure, feedback and errors, and structure exhibited by the semigroup of a machine.

Most of the papers written on the structure of sequential machines are overlapping, as would be expected; however, for the sake of clarity they are classified into the following categories reflecting the type of structure:

(a) Minimal Assignments -- includes papers having the prime aim of obtaining a best (in a cost sense) assignment, but excludes papers that attempt to achieve this aim by considering a specific type of structure, e.g., shift registers or decompositions.

(b) Decomposition -- covers papers that attempt to realize the machine by a number of smaller machines.

(c) Feedback -- includes papers concerned with the amount of feedback in the machine.

(d) Linear -- covers those papers that attempt to determine realizations using only delay elements and exclusive or gates (Mod 2 adders).

(e) Shift Registers -- includes papers having the aim of determining realizations using shift registers, and does not consider the amount of feedback.

(f) Other structures which assume specified building blocks.

Because categories (d) and (e) are covered in some detail in Chapters 4 and 5 respectively, they will not be considered any further in this chapter. Therefore, to achieve a complete picture on structure theory, the interested reader is advised to read Chapter 2 and Sections 4.1 and 5.2 together.

2.1 Minimal Assignments

A considerable amount of effort has been expended on the problem of obtaining the minimal (in some sense) structure. This problem is usually referred to as 'the assignment problem'; however, some of the results are indirect approaches, i.e., the minimal structure is found by decomposition. Since all structure problems are essentially assignment problems, we will restrict ourselves in this section to those whose prime aim is a minimal structure.

The first approach to this problem, by Hartmanis and Stearns [41], was to look for assignments that reduce the dependence between the internal state variables. This has been further advanced by Curtis, Karp (See [1]), Friedes [27]and Zahle [89].

The second approach, by Armstrong, et al., (see Crowley [9] for a more complete description) is to look for adjacencies between internal states, and to find an assignment which incorporates as many adjacencies as possible. This approach has the distinct advantage of having been programmed; however, it has the disadvantage of being inexact.

Haring [37] studied the bounds on the amount of combinational logic required by sequential machines. This permits the designer to evaluate an assignment.

Dolotta and McCluskey [23], in a slightly different approach, examine all binary partitions and by a rather complex system of weights attempt to find a minimal assignment. While this approach does have its merits, the sheer number of partitions for large machines is sufficient to restrict its use to machines with few states.

2.2 Decomposition

By a decomposition of a sequential machine we mean a realization of the machine from smaller machines. In general this is usually a loop-free decomposition, i.e., machines in parallel, cascade or both. The big advantage of decomposition is that a large machine, which may be too large to be handled by a minimal assignment algorithm, may be decomposed into several smaller machines, each of which may be small enough to permit the use of the algorithm. Decomposition also has the effect of reducing the dependence between the internal variables.

The problem of finding loop-free decompositions, for one-to-one assignments, can be considered essentially solved by Hartmanis and Stearns [41], Yoeli [85,86], and Griffiths [34]. The same problem, for many-to-one assignments, has been approached by Kohavi [50], Hartmanis and Stearns [41], Yoeli [87], and Kohavi and Smith [51],

with the result that, while advances have been made, an algorithm is as yet unavailable. Krohn and Rhodes [52] and Zeiger [90,91] have shown that by considering the semigroup of a machine it is possible to decompose any machine into permutation-reset machines. It has also been shown that a machine whose semigroup is simple cannot be decomposed.

Nichols [67] , on the other hand, considers the problem of modular decomposition, i.e., where we are given a set of fixed modules, and must determine how to realize the given machine using this set of devices. The results, unfortunately, are not in a usable form, as yet.

2.3 Feedback

Feedback is studied in sequential machines because of, among other things, its relationship to reliability and the propagation of errors as shown by Hartmanis and Stearns [41]. Sequential machines can be classified, with respect to feedback, into two categories: a) definite event machines, and b) feedback machines. A definite event machine is a machine that is realizable without feedback, and a variety of methods [4] are available for testing if a machine is definite or not. With regard to feedback machines, Friedman [28,29] has shown that every feedback machine is realizable with a single binary feedback loop. Brzozowski [5] has also developed methods for finding single-loop realizations. Since every feedback machine is realizable with one feedback loop, we can say that every sequential

machine is decomposable into a loop-free connection of a definite event machine and a single binary feedback shift register. It has been shown [13] that some machines, but not all are realizable as single binary feedback shift registers. Therefore we can further classify feedback machines into: a) single binary feedback shift register machines, and b) combination machines, i.e., a loop free connection of a definite event machine and a single binary feedback shift register.

It should be noted that a single-loop machine may be uneconomical in components, hence if economy is a criterion then a method for evaluating costs is necessary.

2.4 Miscellaneous Results

In addition to the results summarized in the preceding sections, there are a number of additional results which should be mentioned.

Iterative or cellular circuits have been studied in many forms and various configurations including infinite arrays. While not every form is applicable to the present survey, what is pertinent is the fact that sequential machines are realizable with a repetitive or iterative module. The value of this type of structure is that it is very amenable to the rapidly advancing technology of microcircuits and integrated circuits.

The structure is essentially fixed in a one or two dimensional planar array, with variations permitted by changing the orientation of the

cell, or by altering the output function, as in a cutpoint cell. In short: iterative circuits have been studied by Hennie [44]; by Kilmer [46,47,48]; by Maitra [57], under the title of cascaded switching networks; by Sklansky [75], who called them tributary switching networks; by Minnick [62], using the title cutpoint cellular logic; by Stone and Korenjak [78], who used cellular cascades; Yooli [88]; and Haring [37].

Sequential circuit realizations using threshold logic have been considered by Gustafson, et al [35]. Here it is shown how to realize counters for any period using one threshold element for each memory element. Hadlock and Coates [36] and Masters and Mattson [59] also study realizations using threshold elements.

Since the reliability of a sequential circuit is a function of the structure of the circuit, some (certainly not all) of the recent work in this area should be mentioned. Winograd [82] and Harrison [38] have studied the error correcting capability of sequential machines. Russo [72] shows how to synthesize counters which are tolerant of single bit errors. Russo's technique has the distinction that the redundancy is inherent in the design of the circuit. Frank and Yau [24] have also considered error-correcting state assignments.

In conclusion, we have briefly summarized the current results in structure theory for sequential machines. We have covered techniques for finding minimal assignments, decomposition, and feedback, plus

a number of miscellaneous methods. Before proceeding to linear machines and shift registers, in Chapters 4 and 5 respectively, we consider state assignments for combinational circuits in the next chapter.

Chapter 3

COMBINATIONAL LOGIC ASSIGNMENTS

In this chapter the relationship between the internal state assignment and the complexity of the combinational logic is considered. Initially, we present an approach for finding a state assignment which minimizes the combinational circuitry necessary to realize a set of given output functions. It is then shown how the same approach can be used in minimizing the combinational logic realizing the next-state functions.

A solution to these problems can be applied to other problems of a similar nature: One such problem could occur in a computer [15], where the outputs of the order code decoder, i.e., the control signals necessary to operate the various functions, registers, peripheral equipment, etc., are already determined by the design of the remainder of the computer, and it is desired to make an optimal order code assignment to provide a minimal structure in the order code de-

coder. Another case could be an n bit register, whose outputs are used directly in another application; however, only $q \leq 2^{n-1}$ of the possible states of the register occur. Therefore, it is necessary to know the minimal cost to save k memory elements, and use an $(n-k)$ bit register, where $2^{n-k} \geq q$. We therefore consider the following generalized problem [18]:

Given q states, and r output functions, determine an assignment of n binary variables to the q states where $n \geq [\log_2 q]$ such that the logical network needed to realize the r functions is the minimum necessary.

This problem has been considered previously by Dolotta and McCluskey [21] and by Dolotta [20]; however, their results were mainly the derivation of methods for simplifying incompletely specified tables. The present chapter does consider incompletely specified tables, but the primary result establishes a new approach to the solution of the main problem (i.e., the input code assignment).

Since the number of distinct assignments [60], as tabulated in Table 3.1, becomes quite large for more than four states, it is impossible, in any practical problem, to consider all possible assignments, even with a high-speed digital computer. Hence, it is a decided advantage to have a method for restricting the set of all possible assignments to a subset with a manageable number of elements. The proposed approach is to first consider configurations that provide the possibility of a minimal realization and then, by using this configuration as a standard, to examine the remaining configurations in

a tree search, to either prove that no smaller realization exists, or to find a smaller realization.

Table 3.1

Number of possible assignments of n variables for q states

No. of States	No. of Variables	No. of Distinct Assignments
1	0	1
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
8	3	840
9	4	10,810,800
q	$n = [\log_2 q]$	$\frac{(2^n-1)!}{(2^n-q)!\ n!}$

The output functions will be given by a table, called the output table, as shown in Table 3.2.

Table 3.2

Example 1

S	f_1	f_2	f_3	f_4	f_5
A	0	0	ø	ø	1
B	1	0	1	ø	0
C	0	1	1	1	1
D	1	ø	0	0	0
E	1	0	ø	ø	1
F	1	0	ø	0	0
G	1	0	0	0	0
H	1	1	0	ø	1
I	1	1	0	ø	0

In this example there are 9 states to be coded, A, B, ..., I, and 5 output functions f_1, f_2, ..., f_5. An output table is called <u>incompletely specified</u> if it contains 'don't cares' (ø's), otherwise it is called <u>completely specified</u>. Initially only completely specified tables are considered, and later the method is generalized, to include incompletely specified tables.

In the following treatment only assignments where $n = [\log_2 q]$ are considered; however, the extension to larger n's is obvious and will be omitted. Furthermore, 2 level unshared diode logic is used ('and'

plus 'or' gates), the number of gates or the number of diodes determines the expense factor, and the complements of the assigned variables are assumed to be available.

3.1 Minimal Functions

To appropriately define the difference between a good realization and a poor realization (i.e., number of gates or diodes necessary) some of the basic properties of bp's and bp blocks are now established.

Definition 3.1: The union $b_k = b_1 + b_2 + \ldots + b_i$ is irredundant if and only if $b_k \neq b_1 + b_2 + \ldots + b_{j-1} + b_{j+1} + \ldots + b_i$ for any $j = 1, 2, \ldots, i$.

In other words, an irredundant union is a union in which all terms are essential. A union which is not irredundant is called redundant.

Lemma 3.1: The union $b_k = b_1 + b_2 + \ldots + b_i$ is irredundant if and only if there exists at least one state $A \in b_j$ such that $A \notin b_m$, $m = 1, 2, \ldots, j-1, j+1, \ldots, i$ for all $j = 1, 2, \ldots, i$.

Definition 3.2: An irredundant union $b_k = b_{11} + b_{21} + \ldots + b_{il}$ is minimal if and only if there does not exist a smaller $(j < i)$ subset of blocks such that $b_k = b_{12} + b_{22} + \ldots + b_{j2}$.

As an example consider $\{A, B, \ldots, I\}$ and P_1, P_2, P_3, P_4, and P_5 given by b_1, b_2, b_3, b_4, b_5 = (A,B,C,D,E); (A,B,D); (B,C,E,F,G); (B,F,H); (B,C,E,F,G,H). It is easily shown that $b_7 = b_1 + b_2 + b_3$ is

redundant for $b_7 = b_1 + b_3$. It can also be shown that $b_8 = b_2 + b_3 + b_4$ is both irredundant and non-minimal for $b_8 = b_1 + b_5$. Using the principle of duality, similar definitions and lemmas about irredundant products can be stated.

Definition 3.3: The minimum gate realization number $g(f_i)$, of an output function f_i, is the minimum number of gates necessary to realize f_i, for any assignment $y_1, y_2, \ldots, y_n$, such that $f_i \neq y_1, y_2, \ldots, y_n$.

Definition 3.4: The minimum diode realization number $d(f_i)$, of an output function f_i, is the minimum number of diodes necessary to realize f_i, for any assignment $y_1, y_2, \ldots, y_n$, such that $f_i \neq y_1, y_2, \ldots, y_n$.

In other words, for an output function f, $g(f)$, (or $d(f)$) is the non-zero number of gates (diodes) necessary to realize f in the minimal mapping of f into an n-cube. In a general solution to the problem under consideration, it will be necessary to be able to determine $g(f)$ or $d(f)$ for any f and q. An enumeration method for solving this problem was given by Dolotta [20], and by Dolotta and McCluskey [22]. However, for the purpose of this thesis, this is considered to be a separate problem. Hence, the major problem is approached on the assumption that $g(f)$ or $d(f)$ can be determined. In Table 3.3, $g(f)$'s and $d(f)$'s are tabulated for $q = 3, 4, \ldots, 16$; the table was obtained by cut and try methods. For example, consider the case where $q = 6$, $n = 3$ and $\#(b) = 3$. Here, 6 specified entries on the n-cube are given, and it is necessary to determine the minimal function distinguishing 3 of the entries. The unspecified entries can be divided so that in the completely specified case it is possible to distinguish 3, 4, or 5 ele-

ments. A single two input gate will distinguish either 2 (for an AND gate) or 6(for an OR gate) elements; however, a pair of two input gates in the form of $y_1(y_2 + y_3)$ or $y_1 + y_2y_3$ can distinguish 3 or 5 elements. Therefore d(f) = 4 and g(f) = 2.

Table 3.3

g(f) and d(f) for q = 3, 4, ..., 16

q	n	#(b)	#($\bar{b}$)	d(f)	g(f)
3	2	1	2	2	1
4	2	1	3	2	1
		2	2	6	3
5	3	1	4	2	1
		2	3	2	1
6	3	1	5	2	1
		2	4	2	1
		3	3	4	2
7	3	1	6	2	1
		2	5	2	1
		3	4	4	2
8	3	1	7	3	1
		2	6	2	1
		3	5	4	2
		4	4	6	3
9	4	1	8	2	1
		2	7	2	1
		3	6	2	1
		4	5	2	1
10	4	1	9	2	1
		2	8	2	1
		3	7	2	1
		4	6	2	1
		5	5	4	2
11	4	1	10	2	1
		2	9	2	1
		3	8	2	1
		4	7	2	1
		5	6	4	2

q	n	#(b)	#($\bar{b}$)	d(f)	g(f)
12	4	1	11	2	1
		2	10	2	1
		3	9	2	1
		4	8	2	1
		5	7	4	2
		6	6	4	2
13	4	1	12	2	1
		2	11	2	1
		3	10	2	1
		4	9	2	1
		5	8	4	2
		6	7	4	2
14	4	1	13	3	1
		2	12	2	1
		3	11	2	1
		4	10	2	1
		5	9	4	2
		6	8	4	2
		7	7	6	3
15	4	1	14	3	1
		2	13	3	1
		3	12	2	1
		4	11	2	1
		5	10	4	2
		6	9	4	2
		7	8	5	2
16	4	1	15	4	1
		2	14	3	1
		3	13	5	2
		4	12	2	1
		5	11	7	3
		6	10	4	2
		7	9	5	2
		8	8	6	3

3.2 Completely Specified Case

As an example of the completely specified case, consider Example 2 given in Table 3.4. Here $q = 7$, hence $n = 3$.

Table 3.4

Example 2

S	f_1	f_2	f_3	f_4
A	0	1	0	1
B	1	0	1	0
C	1	1	1	0
D	1	0	1	1
E	0	1	0	0
F	0	1	1	0
G	0	0	0	0

An assignment of 3 binary variables to the 7 states is to be determined, such that, the logical network realizing the 4 output functions is minimal. An example of an assignment for Example 2, not necessarily minimal , is given in Table 3.5.

Table 3.5

Assignment A for Example 2

S	y_1	y_2	y_3
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1
G	1	1	0

The functions necessary to realize the outputs (considering don't care states, and both sum of products and product of sums form) for assignment A of Example 2 are therefore:

$$f_1 = \bar{y}_1(y_3 + y_2),$$
$$f_2 = \bar{y}_3\bar{y}_1 + \bar{y}_2 y_1,$$
$$f_3 = y_3 + y_2\bar{y}_1,$$
$$f_4 = y_3 y_2 + \bar{y}_3\bar{y}_2\bar{y}_1.$$

The expense factor for assignment A for Example 2 is therefore 10 gates or 21 diodes. Now consider the systematic method.

First determine which outputs define a qbp over the set of all states. Then enumerate the sets of output qbp's which are mutually consistent. For example, it is easily shown that in Example 2, f_1, f_2

and f_3 define qbp's P_1, P_2, P_3 respectively and P_2, P_3 are the only qbp's that are mutually consistent. Therefore 5 possible approaches are possible, called assignment schemes, for finding a minimal assignment. These are:

S_1: P_2, P_3 and P_5, where $P_2P_3P_5 = 0$.

S_2: P_1, P_5 and P_6, where $P_1P_5P_6 = 0$.

S_3: P_2, P_5 and P_6, where $P_2P_5P_6 = 0$, and P_5, $P_6 \neq P_3$.

S_4: P_3, P_5 and P_6, where $P_3P_5P_6 = 0$, and P_5, $P_6 \neq P_2$.

S_5: P_5, P_6 and P_7, where $P_5P_6P_7 = 0$, and $P_5, P_6, P_7 \neq P_1, P_2, P_3$.

Since only valid minimal assignments are considered, when indicating an assignment scheme, it is natural to assume that the product of the qbp's is equal to zero. Furthermore to indicate the qbp determined by an output f_i, use P_i, and to indicate those qbp's not determined by an output use P_{r+1}, P_{r+2}, ... From Tables 3.3 and 3.4, it is seen that $g(f_1)$, $g(f_2)$, $g(f_3)$ and $g(f_4)$ are 2,2,2, and 1 respectively and that $d(f_1)$, $d(f_2)$, $d(f_3)$ and $d(f_4)$ are 4,4,4 and 2 respectively.

Definition 3.5: The minimum realization sum, $m(S_j)$, of an assignment scheme S_j, is the sum of $g(f_1)$, or $d(f_i)$, for those output functions f_i which do not determine a qbp used in the scheme, i.e., $m(f)$ is a lower bound for the realization cost.

For example, consider S_1 for Example 2, and a minimal diode realization. Since P_2 and P_3 are used in the scheme, $m(S_1) = d(f_1) + d(f_4) = 6$. Similarly, it is seen that $m(S_2)$, $m(S_3)$, $m(S_4)$ and $m(S_5)$ equal 10, 10, 10, and 14 respectively.

Once $m(S)$ has been found for each scheme S, consider a scheme having the smallest $m(S)$. It is clear that if it can be shown that a scheme having the smallest $m(S)$ is realizable with that number of diodes, then the problem is solved, since no smaller realization exists. This is not usually the case, and, in general, a tree search is used to obtain the minimal realization. The search terminates when an assignment is found such that the number of diodes used in the realization is less than or equal to any of the other alternatives. It may seem, at this point, that an exhaustive search could become quite extensive, due to the number of possible assignments involved. In practice this turns out to be false, and indeed only those assignments which have the possibility of providing a minimal realization are considered.

To demonstrate, consider Example 2. 5 assignment schemes are possible, but since $m(S_1)$ is smallest S_1 is evaluated first. S_1 uses P_2, P_3, defined by f_2, f_3, and P_5, such that $P_2P_3P_5 = 0$. A P_5 satisfying the above requirements is to be found, such that the realizations of f_1 and f_4 are minimal. Hence:

$$P_2 = \{\overline{A,C,E,F};\ \overline{B,D,G}\}$$

$$P_3 = \{\overline{B,C,D,F};\ \overline{A,E,G}\}$$

$$P_2P_3 = \{\overline{C,F};\ \overline{B,D};\ \overline{A,E};\ \overline{G}\}.$$

Therefore P_5 must separate each of the pairs CF, BD and AE in order that $P_2P_3P_5 = 0$. We now consider the additional constraints imposed upon P_5, by the functions realizing f_1 and f_4, f_1 and f_4 determine the binary partitions:

$$P_1 = \{\overline{B,C,D};\ \overline{A,E,F,G}\}$$
$$P_4 = \{\overline{A,D};\ \overline{B,C,E,F,G}\}. \quad \text{and}$$

Hence, it is desired to determine the blocks $b_1 = (BCD)$ and $b_4 = (AD)$. First, consider b_1, and the following two forms:

$$b_1 = \tilde{b}_i(\tilde{b}_j + \tilde{b}_k) \quad \text{or} \qquad (3.1)$$
$$b_1 = \tilde{b}_i + \tilde{b}_j\tilde{b}_k, \qquad (3.2)$$

where i, j and k are all different. The first of these implies that $\tilde{b}_i \supset b_1$, and the second that $b_1 \supset \tilde{b}_i$. $b_1 \supset \tilde{b}_i$ is not satisfied by b_2 or b_3, and cannot be satisfied by b_5 since both B and D cannot be in the same block of P_5. However, $b_1 \subset b_3$ and $B, D \in \bar{b}_2$. Hence, if $C \in b_5$ and $F \in \bar{b}_5$ then $b_1 = b_3(\bar{b}_2 + b_5)$. But, C and F must be separated by P_5, hence any P_5 such that $P_2P_3P_5 = 0$ will provide a realization of f_1 using 4 diodes. Now, consider b_4, and initially a 2 diode realization in the form:

$$b_4 = \tilde{b}_i\tilde{b}_j, \qquad i \neq j. \qquad (3.3)$$

This implies that $b_4 \subset \tilde{b}_i, \tilde{b}_j$. But note that $b_4 \not\subset \tilde{b}_2, \tilde{b}_3$. Hence, a 2 diode realization of f_4 is impossible with S_1. Furthermore a 3 diode realization must identify 1 or 6 states and no others, hence the next best is a 4 diode realization in the form of (3.1) or (3.2). Clearly, $b_4 \not\subset \tilde{b}_2, \tilde{b}_3$ and $b_4 \not\supset \tilde{b}_2, \tilde{b}_3$ hence either $b_4 \subset \tilde{b}_5$ or $b_4 \supset \tilde{b}_5$. Now, $b_4 \not\supset \tilde{b}_5$, since the minimum number of states in $\tilde{b}_5$ is 3. Therefore, for a 4 diode realization the form $b_4 = \tilde{b}_5(\tilde{b}_2 + \tilde{b}_3)$, where $b_4 \subset \tilde{b}_5$, and $b_4 \subset (\tilde{b}_2 + \tilde{b}_3)$ must be used. The only two possibilities for $\tilde{b}_2 + \tilde{b}_3$ are $b_2 + b_3$ and

$\bar{b}_2 + \bar{b}_3$. For $b_2 + b_3$, AD must be contained in one block of P_5, while BCEF are contained in the other. However, C and F cannot be contained in the same block of P_5 and still satisfy $P_2P_3P_5 = 0$. Hence, $b_2 + b_3$ will not work. Now consider the other alternative $\bar{b}_2 + \bar{b}_3$ = (ABDEG). In this case AD must be contained in one block of P_5, while BEG are contained in the other. Hence,

$$P_5 = \{\overline{A,C,D};\ \overline{B,E,F,G}\} \text{ or } \{\overline{A,D,F};\ \overline{B,C,E,G}\}$$

will satisfy the requirements and f_4 is realizable with 4 diodes. Assignment B for Example 2 is shown in Table 3.6.

Table 3.6

Assignment B for Example 2

S	y_1	y_2	y_3
A	1	0	1
B	0	1	0
C	1	1	1
D	0	1	1
E	1	0	0
F	1	1	0
G	0	0	0

The resulting output functions are:

$$f_1 = y_2(\bar{y}_1 + y_3),$$
$$f_2 = y_1,$$
$$f_3 = y_2,$$
$$f_4 = y_3(\bar{y}_1 + \bar{y}_2).$$

The realization uses 8 diodes, and since $8 < m(S_2)$, $m(S_3)$, $m(S_4)$, and $m(S_5)$, the realization is minimal. As a further example, consider Example 3 shown in Table 3.7.

Table 3.7

Example 3

S	f_1	f_2	f_3	f_4	f_5
A	0	1	1	0	1
B	0	1	0	1	0
C	0	0	0	0	0
D	1	1	1	0	1
E	0	1	0	0	0
F	0	1	1	1	0
G	0	1	1	0	0
H	0	1	0	0	1
I	0	1	1	0	1
J	1	1	1	0	0

It can be shown, using the same approach, that the assignment shown in Table 3.8 is minimal and uses 6 diodes.

Table 3.8

Assignment for Example 3

S	y_1	y_2	y_3	y_4
A	1	1	1	1
B	0	0	0	1
C	0	0	1	0
D	1	1	0	0
E	0	0	0	0
F	1	0	1	1
G	1	0	1	0
H	0	1	0	0
I	1	1	1	0
J	1	0	0	0

It is clear that in any problem having a large number of assignment schemes it is not necessary to enumerate all the schemes. For example, consider Example 2, where g(f) and d(f) are given in Table 3.9.

Table 3.9

g(f) and d(f) for Example 2

f	f_1	f_2	f_3	f_4
g(f)	2	2	2	1
d(f)	4	4	4	2

Since P_2 and P_3 are the only qbp's that are mutually consistent, it is clear from the table that the smallest $m(S_j)$ will be $m(S_1)$ given previously. Now it is not necessary to consider the remaining schemes until it is determined that the actual number of gates or diodes is greater than the next smallest $m(S_j)$.

3.3 Simplification of Incompletely Specified Tables

A procedure for the simplification of incompletely specified tables has been given by Dolotta and McCluskey [21], and again by Dolotta [20]. This method is slightly modified and restated in two parts. The first part, the elimination of state redundancies, is optional, since there are cases where it is necessary to have a valid assignment for all states regardless of whether any states have the same output functions or not.

A row (or column) is equivalent to another row (or column) if and only if identical outputs are equal. A row (or column) R_i is covered by another row (or column) R_j, denoted $R_i \subset R_j$, if R_j is equal to R_i for every specified entry of R_j, and R_j is said to cover R_i. The intersection $R_{i.j}$ of two rows (or columns) R_i and R_j is that row (or column) which agrees with both R_i and R_j whenever either is specified and contains ϕ everywhere else. Clearly, the intersection exists for every pair of rows (or columns) which don't disagree. Intersections can obviously be extended to any number of rows (or columns). A

maximal intersection is a largest possible intersection, i.e., $f_{i.j}$ is a maximal intersection if and only if there does not exist an f_k such that $f_{i.j.k}$ exists. The inverse $\bar{f}_k$ of a column f_k is the column obtained by inverting all specified entries of f_k. The inverse of a row certainly exists; however, since it is of no use it is not considered. Column inverses can be used, for if two outputs are given such that one is the inverse of the other, then if one of the outputs is used to determine one variable in the assignment, the other output is also available directly. On the other hand however, if neither output determines a variable in the assignment, then either both outputs are obtained independently or an inverter must be used to obtain the second output. Covers and intersections can be found using inverses. In the following procedure inverses will be considered in the column reduction procedure, but not in the row reduction procedure. As an illustration, consider the example given in Table 3.10.

Rows G and K are equivalent, $B \supset G$, the intersection of A and D is given by 1 ØØ Ø1 0000ØØ10, and $\overline{f_{13}} \supset f_{12}$.

Table 3.10

Example 4

S	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}
A	ø	ø	ø	ø	1	0	0	0	0	ø	ø	1	0
B	1	ø	1	0	ø	ø	ø	0	0	ø	ø	ø	ø
C	0	1	0	ø	ø	1	ø	ø	ø	0	ø	ø	ø
D	1	ø	ø	ø	1	0	0	0	ø	ø	ø	ø	ø
E	1	ø	ø	ø	ø	0	ø	0	0	ø	ø	1	0
F	ø	ø	1	0	1	ø	0	0	ø	ø	1	0	ø
G	1	ø	1	0	0	ø	ø	0	0	ø	1	ø	ø
H	1	ø	1	ø	0	ø	ø	0	0	0	0	ø	ø
I	1	ø	ø	ø	0	ø	ø	ø	1	0	0	ø	ø
J	0	ø	ø	ø	ø	1	ø	1	ø	0	ø	ø	ø
K	1	ø	1	0	0	ø	ø	0	0	ø	1	ø	ø
L	ø	0	0	1	1	0	ø	0	0	ø	ø	1	0
M	1	ø	1	ø	0	ø	ø	0	0	0	0	ø	ø
N	1	ø	1	0	1	ø	0	0	ø	ø	1	0	1

1. Eliminate all state redundancies (Optional):

   a) Discard all equivalent rows (i.e., represent each set of states having equivalent rows by a single row).

   b) Discard all rows that cover another row.

c) With the remaining rows form the set of maximal intersections.

d) Select a minimal set of rows that cover all rows, where if any choice exists select the row having the largest number of don't cares.

2. Eliminate all column redundancies.

a) Discard any trivial columns, ie., any column containing no 0's (or 1's).

b) Discard any column that covers another column.

c) List all inclusions of the type $f_i \supset \bar{f}_j$ for which $\bar{f}_j$ determines a qsm.

d) Form all maximal intersections that contain no primes or determine a qsm.

e) Select a minimal set of columns that cover all columns.

To illustrate the reduction procedure, consider Example 4 given in Table 3.10, where it is assumed that state reduction is permitted. Apply 1. a) and b) to find that B, F, K and M can be discarded since G = K, M = H, B ⊃ G and F ⊃ N. Form the set of maximal intersections:

ADEL:	1	0	0	1	1	0	0	0	0	∅	∅	1	0
CJ:	0	1	0	∅	∅	1	∅	1	∅	0	∅	∅	∅
DN:	1	∅	1	0	1	0	0	0	∅	∅	1	0	1
EG:	1	∅	1	0	0	0	∅	0	0	∅	1	1	0
EH:	1	∅	1	∅	0	0	∅	0	0	0	0	1	0

Then select the row reduced table given in Table 3.11 covering the original table.

Table 3.11

Row reduced table covering Example 4

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}
ADEL	1	0	0	1	1	0	0	0	0	∅	∅	1	0
CJ	0	1	0	∅	∅	1	∅	1	∅	0	∅	∅	∅
EGK	1	∅	1	0	0	∅	∅	0	0	∅	1	∅	∅
HM	1	∅	1	∅	0	∅	∅	0	0	0	0	∅	∅
I	1	∅	∅	∅	0	∅	∅	∅	1	0	0	∅	∅
FN	1	∅	1	0	1	∅	0	0	∅	∅	1	0	1

Now apply part 2. a) and b) of the procedure and find that f_{10} and f_7 are trivial, $f_2 = f_6$, $f_2 \supset f_8$, $f_{13} \supset f_3$, $f_{12} \supset f_4$. Applying 2. c) obtain:

$$\bar{f}_4 \supset f_3.$$

From 2. d) find:

$f_{\overline{11}.4}$	$f_{8.9}$
1	0
∅	1
0	0
1	0
1	1
0	0

Applying 2. e) produces the reduced table given in Table 3.12.

Table 3.12

Reduced table for Example 4

S	$f_{8.9}$	$f_{\overline{11}.4}$	f_1	f_3	f_5
A	0	1	1	0	1
C	1	∅	0	0	∅
B	0	0	1	1	0
H	0	1	1	1	0
I	1	1	1	∅	0
F	0	0	1	1	1

Note that if $f_{\overline{11}.4}$ is not used to define a variable in the assignment then both f_{11} and f_4 must be obtained independently.

3.4 Approach for Incompletely Specified Tables

For incompletely specified tables, first reduce the table with the procedure given in the preceeding section, and then apply the techniques for completely specified tables using qsm's. As an example, consider Table 3.12. It is easily seen that $f_{8.9}$, $f_{\overline{11}.4}$, f_3 and f_5 define qsm's P_1, P_2, P_3 and P_4 respectively. Furthermore, P_2, P_3, P_4 and P_1, P_2, P_4 are the only groups of 3 qsm's that are mutually consistent for n = 3.

The following minimum gate realization numbers can be found from Table 3:

f	$f_{8.9}$	f_1	f_3	f_5	f_{11}	f_4
g(f)	1	1	1	1	1	1

Clearly, the two assignment schemes having the smallest minimum realization sums are:

$$S_1: P_2, P_3, P_4,$$
$$S_2: P_1, P_2, P_4,$$

where $m(S_1) = m(S_2) = 2$.

Consider S_1, where

$$P_2 = \{\overline{A,C,H,I};\ \overline{C,B,F}\},$$
$$P_3 = \{\overline{B,H,I,F};\ \overline{A,C,I}\},$$
$$P_4 = \{\overline{A,C,F};\ \overline{C,B,H,I}\}.$$

For the output functions, it is necessary to realize $f_{8.9}$ and f_1 given by $b_1 = (CI)$ and $b_5 = (ABHIF)$ respectively. Clearly, if $\bar{b}_3 = (ACI)$ and $\bar{b}_4 = (CBHI)$ then $b_1 = \bar{b}_3\bar{b}_4$. Similarly, if $b_2 = (AHI)$ then $b_5 = b_2 + b_3$. This provides an assignment such that the combinatorial circuits require 2 gates, and since no smaller realization exists it is a minimal assignment. The complete assignment is shown in Table 3.13.

The output functions are then given by:

$$f_1 = y_2 + y_3,$$
$$f_2 = \bar{y}_3\bar{y}_4,$$
$$f_3 = y_3,$$

$$f_4 = y_2,$$
$$f_5 = y_4,$$
$$f_6 = \bar{y}_3\bar{y}_4,$$
$$f_7 = 0,$$
$$f_8 = \bar{y}_3\bar{y}_4,$$
$$f_9 = \bar{y}_3\bar{y}_4,$$
$$f_{10} = 0,$$
$$f_{11} = \bar{y}_2,$$
$$f_{12} = y_2,$$
$$f_{13} = y_3.$$

Table 3.13

A minimal assignment for Example 4

S	y_2	y_3	y_4
A	1	0	1
B	0	1	0
C	0	0	0
D	1	0	1
E	1	0	1
F	0	1	1
G	0	1	0
H	1	1	0
I	1	0	0
J	0	0	0
K	0	1	0
L	1	0	1
M	1	1	0
N	0	1	1

3.5 Application in Minimizing Next-State Logic

While this approach can also be used to determine internal state assignments that minimize the combinatorial circuits realizing the outputs of a sequential machine, some of the ideas can also be applied to minimizing the circuits that realize the next-state functions. Consider a sequential machine, where it is desired to evaluate various internal state variables in a manner analogous to those given by Dolotta [20] and by Dolotta and McCluskey [23]. For any single internal state variable given by a binary partition, by considering its predecessors and the techniques presented herein, a lower bound is established on the realization of its next-state function. For example, consider the sequential machine given in Table 3.14, and the qbp P_1, determined by $b_1 = (DE)$. If the predecessor of an internal state S_j, denoted by $\pi(S_j)$ is the set of total states that have s_j as a successor and $\pi(b)$ is the union of the predecessors of the states of b, then:

$$\pi(b_1) = 1(ACD) + 2(BE)$$

Table 3.14
A sequential machine

S \ I	1	2	Z
A	E	B	0
B	A	E	1
C	D	A	1
D	E	C	0
E	C	E	1

Now, it is necessary to evaluate P_1 in terms of the minimum amount of combinatorial circuits necessary to realize the next-state function. It is seen that the minimum would exist if the partition P_2 given by $b_2 = (BE)$ was available. Since P_1 and P_2 are mutually consistent for $n = 3$, both can be used as internal state variables, and hence the lower bound to realize the next-state function of P_1 is 3 gates or 6 diodes. In a similar manner P_2 can be evaluated and in turn each of the other internal state partitions.

3.6 Conclusion

In this chapter, the relationship between the internal state assignment and the complexity of the combinational logic has been considered. In particular, a new approach is presented for assigning a binary input code that is optimal in the sense that it requires the minimum number of gates or diodes necessary to realize the given output functions. A number of additional problems will have to be solved, however, before a complete solution is available:

a) A nonenumerative method for obtaining minimum realization numbers.

b) An algorithm for determining the actual realization cost of a given assignment scheme.

c) A computer program to handle large tables.

In addition, it is noted that Nichols and Berstein [68] have considered the inverse to the problem considered here. In other words,

Now, it is necessary to evaluate P_1 in terms of the minimum amount of combinatorial circuits necessary to realize the next-state function. It is seen that the minimum would exist if the partition P_2 given by $b_2 = (BE)$ was available. Since P_1 and P_2 are mutually consistent for $n = 3$, both can be used as internal state variables, and hence the lower bound to realize the next-state function of P_1 is 3 gates or 6 diodes. In a similar manner P_2 can be evaluated and in turn each of the other internal state partitions.

3.6 Conclusion

In this chapter, the relationship between the internal state assignment and the complexity of the combinational logic has been considered. In particular, a new approach is presented for assigning a binary input code that is optimal in the sense that it requires the minimum number of gates or diodes necessary to realize the given output functions. A number of additional problems will have to be solved, however, before a complete solution is available:

a) A nonenumerative method for obtaining minimum realization numbers.

b) An algorithm for determining the actual realization cost of a given assignment scheme.

c) A computer program to handle large tables.

In addition, it is noted that Nichols and Berstein [68] have considered the inverse to the problem considered here. In other words,

how does one find an output code which minimizes the combinational circuits realizing the outputs, when the inputs are given.

Chapter 4

LINEAR SEQUENTIAL MACHINES

Linear sequential machines are a proper subset [10] of the set of all sequential machines. Their importance is due, not only to their wide application in coding, decoding and sequence generation for communications, but also to their mathematical properties such as canonical forms and transfer functions. Significant advances have been made recently on the problem of determining whether a given sequential machine is realizable as a linear machine or not. We begin by summarizing the known methods for solving this problem. Given a sequential machine M:

1. If the output of M is given, then an algorithm [11] exists for determining whether or not M is realizable as a linear machine.

2. For any single column of M containing 2^n states, a method [24,32] is available for determining whether or not there exists an autonomous linear assignment, using n internal variables, for that column. Once a linear assignment has been obtained for any single column* it is easily determined [19,11] whether or not the assignment is also linear for the remaining columns.

3. If only the next-state behavior is given (or considered) and M:

   a) contains unit-period states,

   b) contains 2^w unit-period inputs, for $w > 0$,

   c) is strongly connected,

   d) is nonautonomous, i.e., has more than one input column,

   e) is a permutation machine, i.e., each state appears in every column of the machine, then an algorithm [76,77] is available for determining one-to-one linear state assignments.

4. If only the next-state behavior is given (or considered), then two algorithms [40] exist to determine whether or

---

* It is easily shown that every autonomous machine is trivially linear, by using a unit delay element for each internal state.

not M is realizable (one-to-one assignment) as a single binary feedback shift register having a linear feedback function.

In Sections 4.2 and 4.3 the following additional results are presented.

1. If the output is given, then a simplified algorithm is given for determining whether or not a linear realization exists for M.
2. If M is a permutation machine, and only the next-state behavior is given (or considered), then two methods are given for determining whether or not there exists a one-to-one linear state assignment for M.
3. If M is either a nonpermutation machine or a nonlinear permutation machine then methods are given for determining whether or not M is realizable as a linear machine.

4.1 Summary of Previous Results

In this Section existing methods for synthesizing linear machines are summarized. While the synthesis technique does not constitute a linearity test in every case, they are all included for completeness.

Definition 4.1: A sequential machine M is said to be linear (over a field of two elements), if and only if there exists a one-to-one state assignment

$$S \rightarrow \{(y_1, y_2, \ldots, y_n)\}, \qquad 2^{n-1} < |S| \leq 2^n,$$

for which

a) The next-state variables are linear functions of the input and present state variables

$$y'_i = \sum_{j=1}^{n} a_{ij} y_j \oplus b_i x \oplus c_i$$

for

$$i = 1, 2, \ldots, n,$$

or in matrix notation

$$Y' = YA + XB + C.$$

b) The outputs are also linear functions of the input and present state variables.

$$Z = YD + XE + F.$$

It should be noted that the constant inputs C and F above may be removed with the addition of another internal variable y_k, such that $y'_k = y_k$ (for complete details see [11]).

Huffman, in his original paper [45], presents techniques for finding a realization for a linear machine whose transfer function is given. Essentially, the result is this: Given a transfer function of the form:

$$\frac{P(D)}{Q(D)} = \frac{a_o \oplus a_1 D \oplus a_2 D^2 \oplus \ldots \oplus a_n D^n}{b_o \oplus b_1 D \oplus b_2 D^2 \oplus \ldots \oplus b_n D^n}$$

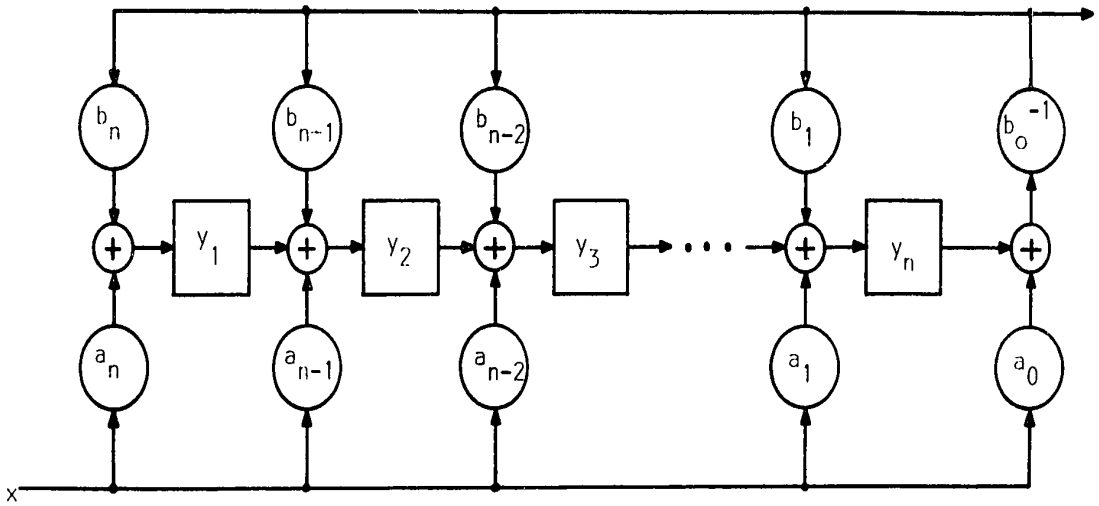

FIG. 4.1 - REALIZATION OF P(D)/Q(D)

where it is easily shown that every transfer function can be converted to this form, the linear sequential machine realizing the transfer function is shown in Fig. 4.1.

Elspas [24], and subsequently Gill and Tan [32], present methods for finding linear realizations for autonomous permutation machines containing 2^n states. The machine is expressed as a cycle set, and is linear if, and only if, the cycle set is expressible as a product of canonical linear cycle sets. This method suffers from two major drawbacks, the first of which is the fact that the method is restricted to machines having 2^n states, and the second is that machines having non-zero constant inputs [7] are not considered.

Gill [30,31] has studied the problem of synthesis of singular autonomous linear machines. He presents a synthesis technique for autonomous state graphs containing 2^n states.

Srinivasan [76,77] investigated the problem of determining whether or not a given state diagram can be realized by a linear machine. The resulting algorithm will determine a linear assignment for those machines whose state diagram:

a) contains unit-period states, i.e., states S_i such that there exists an input I_j, called a unit-period input, and $\delta(S_i, I_j) = S_i$.

b) contains 2^w unit-period inputs, for $w > 0$.

c) is not disjoint.

d) is a nonautonomous, permutation machine.

Briefly the algorithm has the following steps:

1. Code the unit-period inputs.
2. Construct all well-formed sequences of length r, where a well-formed sequence is a sequence beginning with a unit-period input,

$$r \leq n/w < r + 1, \text{ and } 2^n = q.$$

3. Choose arbitrarily a unit-period state s_o of S, and form a subset $F(r,0) = \delta(s_o, X^r_o)$. Choose a unit-period state $s_i \notin F(r,0)$, if one exists, as the initial state and form another subset $F(r,i) = \delta(s_i, X^r_i)$. Repeat until all unit-period states are exhausted.
4. If the set of states obtained in 3 does not contain all the states of S, then this set is extended. The process is rather lengthy and will not be explained in detail here.
5. Code the states.

As an illustration, consider machine M4.1 given in Table 4.1.

Table 4.1

Machine M4.1

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

Here $w = 1$, $n = 3$, therefore $r = 3$. All well-formed sequences of length 3 are:

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Starting with the unit-period state A, and applying the first four well-formed sequences we have:

$\delta(A,000) = A$
$\delta(A,001) = D$
$\delta(A,010) = F$
$\delta(A,011) = G$

Since we have not exhausted all the unit-period states, we select C and apply the last four sequences to produce:

$$\delta(C,100) = E$$
$$\delta(C,101) = H$$
$$\delta(C,110) = B$$
$$\delta(C,111) = C$$

This last step exhausted all the states, and we are left to code the states, producing the following linear assignment:

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

The next-state equations are therefore:

$$y'_1 = y_1,$$
$$y'_2 = y_1 \oplus y_2 \oplus y_3,$$
$$y'_3 = y_1 \oplus y_2 \oplus x.$$

Cohn and Even [11] studied linear machines and present an identification technique that is unique in the sense that very few restrictions are imposed on the machine. This algorithm begins by ensuring that the machine is minimal. Then, starting with the outputs of the set of states for the minimized machine, find the outputs of their successor states under a single input, i.e., as an autonomous machine. This step is repeated until

linear dependence is reached among all the new output vectors. The set of independent output vectors provides a linear state assignment, if it exists, for the autonomous machine. The algorithm then checks to ensure that the assignment holds for the remaining inputs. This algorithm represents a significant advance in that it is a linearity test for all machines whose outputs are given. Furthermore, it is shown that state-splitting does not help, i.e., if the reduced machine is non-linear with a one-to-one assignment then there does not exist a larger linear machine realizing the given machine. On the negative side there exist two disadvantages: Firstly, the algorithm may be lengthy, in that one finds successive states and then their outputs. Secondly, if the particular column chosen is linear, and all autonomous machines are linear, then a negative reply could involve a considerable amount of work.

As an example, consider machine M4.2 given in Table 4.2.

Table 4.2

Machine M4.2

	0	1	z
A	H	D	0
B	E	C	0
C	B	A	0
D	G	F	1
E	G	F	0
F	E	C	1
G	H	D	1
H	B	A	1

Applying the Cohn and Even method we produce:

	A	B	C	D	E	F	G	H	
z(0)	0	0	0	1	0	1	1	1	✓
z(1)	1	0	0	1	1	0	1	0	✓
z(2)	0	1	0	1	1	1	0	0	✓
z(3)	0	1	1	0	0	1	0	1	X

We have determined the following assignment:

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

After checking the remaining column for consistency, the assignment produces the following next-state and output equations:

$$y_1' = y_2,$$
$$y_2' = y_3 \oplus x,$$
$$y_3' = y_2 \oplus x \oplus 1,$$
$$z = y_1.$$

Hartmanis [40] presents two tests for determining the linearity of single binary shift registers. The first test assumes that the machine has a transfer function of the form P/Q where the order of Q is greater than or equal to the order of P. If this condition is not satisfied then the machine may be decomposed into a feedback free tail machine [41] and a

machine M_τ, where $\tau = M^R(0) = M^{k+1}(0)$, ($M^i(0)$, is a partition obtained from $M^{i-1}(0)$ by identifying those blocks having the same next state rows, and $M^0(0) = 0$), and M_τ satisfies the requirement.

The test is:

1. There must exist exactly two unit-period (stationary) states.
2. Find a shortest transfer sequence $x_1x_2 \ldots x_n$ between the two states and compute:
   $Q(D) = 1 \oplus (x_1 \oplus x_2)D \oplus \ldots \oplus (x_{n-1} \oplus x_n)D^{n-1} \oplus D^n$.
   Check if the state behavior of M is isomorphic to the state behavior of $M_{1/Q}$ (i.e., the machine whose transfer function is $1/Q(D)$).
3. If $M_{1/Q} \cong M$ then M has a linear next-state relation. If an output is given for M, we check if it is a linear function of the input and state variables.

The second test is almost identical to that given in Section 5.2.1, with the exception that the feedback function f must be a linear combination of the input and state variables, and therefore will not be repeated here. Both of these tests are useful, but only consider a restricted class of linear machines.

As an illustration we will apply the first test to machine M4.3 given in Table 4.3.

Step 1 of the test is satisfied, since F and H are the only unit-period states. The shortest transfer sequence from F to H is

1 1 0, and $Q(D) = 1 \oplus (x_1 \oplus x_2)D \oplus (x_2 \oplus x_3)D^2 \oplus D^3 = 1 \oplus D^2 \oplus D^3$. It is easily shown that the next-state behavior of M4.3 is isomorphic to that of $M_{1/Q}$.

Table 4.3

Machine M4.3

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

This completes the first test.

Pugsley [70,71] considers the synthesis of linear sequential functions with unit delays and exclusive or gates which involve unit delays, i.e., the unit delay is physically part of the gate. He then proves that every linear sequential function that is retrospective, i.e., not dependent on future values of the input, is realizable with some delay using two-input exclusive or delay gates, and unit delays. A problem occurs, however, when one tries to realize the function with minimum input/output delay.

Yau and Wang [84] have also studied linear machines and present a test for the linearity of machines. The major portion of their paper is

restricted to machines with 2^n states, and furthermore the linearity test is merely a reiteration of Elspas's [24] test for nonsingular autonomous machines and Gill's [30,31] techniques for singular autonomous machines extended to nonautonomous machines. In other words, the machine is tested as an autonomous machine for a single input column and the assignment checked to ensure that the same assignment holds for the other input columns. The authors were apparently not aware of Gill and Tan's [32] work and hence propagate the erroneous assumption that cycle set factorization is unique. When extending the process to machines not having 2^n states, the authors state the rather obvious conclusion that a machine is linear only if there exists a way of specifying the remainder of the machine so that it is realizable as a linear machine with 2^n states. Nothing is said about how one finds a way of specifying the remainder of the machine. As a result, the paper provides very little except for completely specified machines with 2^n states, and n-variable realizations without non-zero constant inputs.

4.2 Linear Output Machines

In this section we investigate machines with given output and give a detailed algorithm for establishing whether the machine is linear or not. It will be shown that the algorithm is applicable to one-to-one assignments. It is known [11,31a] that the extension to many-to-one assignments will not help. In other words, if a machine is non-linear, then by splitting states one cannot make it linear. The algorithm is an extension of a previously presented method.

The following theorem will permit us to reject by inspection certain flow tables as not having a linear assignment.

Theorem 4.1: For every linear sequential machine M having n delay elements and a characteristic matrix A of rank e, where $n \geq e \geq 0$, the flow table containing $q \leq 2^n$ distinct states has:

1. In any single input column:
   a) d distinct next states, where $d \leq 2^e$, and
   b) each next state cannot be repeated more than 2^{n-e} times.
2. a symmetric one-to-one correspondence between next states of any two input columns of the flow table, such that if any single state corresponds to itself, then the two columns are identical.

Proof: The theorem has been proved in [19].

Definition 4.2: The sum of two bp's $P_1 = \{b_1 : \bar{b}_1\}$ and $P_2 = \{b_2 ; \bar{b}_2\}$, is defined by:

$$P_1 \oplus P_2 = \{b_1 \oplus b_2 ; \overline{b_1 \oplus b_2}\}$$

We can similarly find the sum of k bp's, and it is easily shown that the sum operation is associative and commutative.

Definition 4.3: Let $P_1, P_2, \ldots, P_k$ be k bp's. Then $P_1, P_2, \ldots, P_k$ are (linearly) independent if, and only if

$$\bigoplus_{i=1}^{k} a_i P_1 = 0,$$

implies that $a_i = 0$ for all i, where 0 is the zero partition $\{\emptyset;S\}$, $a_i \in \{0,1\}$, $0P_i = 0$, $1P_i = P_i$.

Definition 4.4: The sum of a set c of input states and a set b of internal states is a set of total states defined by

$$b \oplus c = b\bar{c} + \bar{b}c,$$

where $\bar{b}$ is the complement of b with respect to S, and $\bar{c}$ is the complement of c with respect to I.

Definition 4.5: The sum of an input partition $Q = \{c;\bar{c}\}$ and a state partition $P = \{b;\bar{b}\}$ is defined as the total state partition

$$P \oplus Q = \{b \oplus c;\ \overline{b \oplus c}\}$$

Lemma 4.1: For every linear sequential machine, each binary output defines a bp R on the total states and $R = P \oplus Q$, where P is a bp on S and Q is a bp on I.

Proof: Every binary output z is given by a function of the form

$$z = f(y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_r).$$

Now, since each y_i defines a bp on S and each x_i defines a bp on I, and the bp's so defined are independent, z must define a bp on the total states and $R = P \oplus Q$.

Definition 4.6: An S/I k-tuple, denoted $T = P_1/Q_1, P_2/Q_2, \ldots, P_k/Q_k$, where P_1 and Q_1 are state bp's and input bp's respectively, is an ordered k-tuple of bp's such that $P_1, P_2, \ldots, P_k$ are linearly independent,

$\pi P_i = P_{i-1} \oplus Q_{i-1}$ for $i = 2,3,\ldots,k$, and πP_1 is a linear combination of $P_1,P_2,\ldots,P_k,Q_1,Q_2,\ldots,Q_k$. We now define independence among k_i-tuples.

Definition 4.7: m-S/I k-tuples $T_1,T_2,\ldots,T_m$ are linearly independent if and only if the $k_1k_2\ldots k_m$ state bp's are independent.

Lemma 4.2: If $T_1,T_2,\ldots,T_j$ are independent and P_i is a state bp that is dependent, then T_i (i.e., the S/I k_i-tuple containing P_i) is also dependent.

Theorem 4.2: For every linear sequential machine M, there exist $r > 0$ S/I k_i-tuples.

Proof: Every linear sequential machine is realizable in canonical form, i.e., either companion matrix or rational canonical form depending on whether the characteristic matrix is non-derogatory or derogatory. In either case the realization is in the form of a number of parallel shift registers where

$$y_1' = f(y_1,y_2,\ldots,y_k) + f_1(x_1,x_2,\ldots,x_r),$$
$$y_i' = y_{i-1} \oplus f_i(x_1,x_2,\ldots,x_r), \text{ for } i = 2,3,\ldots,k.$$

Since each y_i defines an internal state bp, P_i, and each $f_i(x_1,x_2,\ldots,x_r)$ defines an input bp, Q_i, there exist $r > 0$ S/I k-tuples

$$P_1/Q_1,\ P_2/Q_2,\ \ldots,\ P_k/Q_k.$$

We now apply this theorem to the problem of determining whether or not a given machine is linear or not, and produce the algorithm shown in Fig. 4.2.

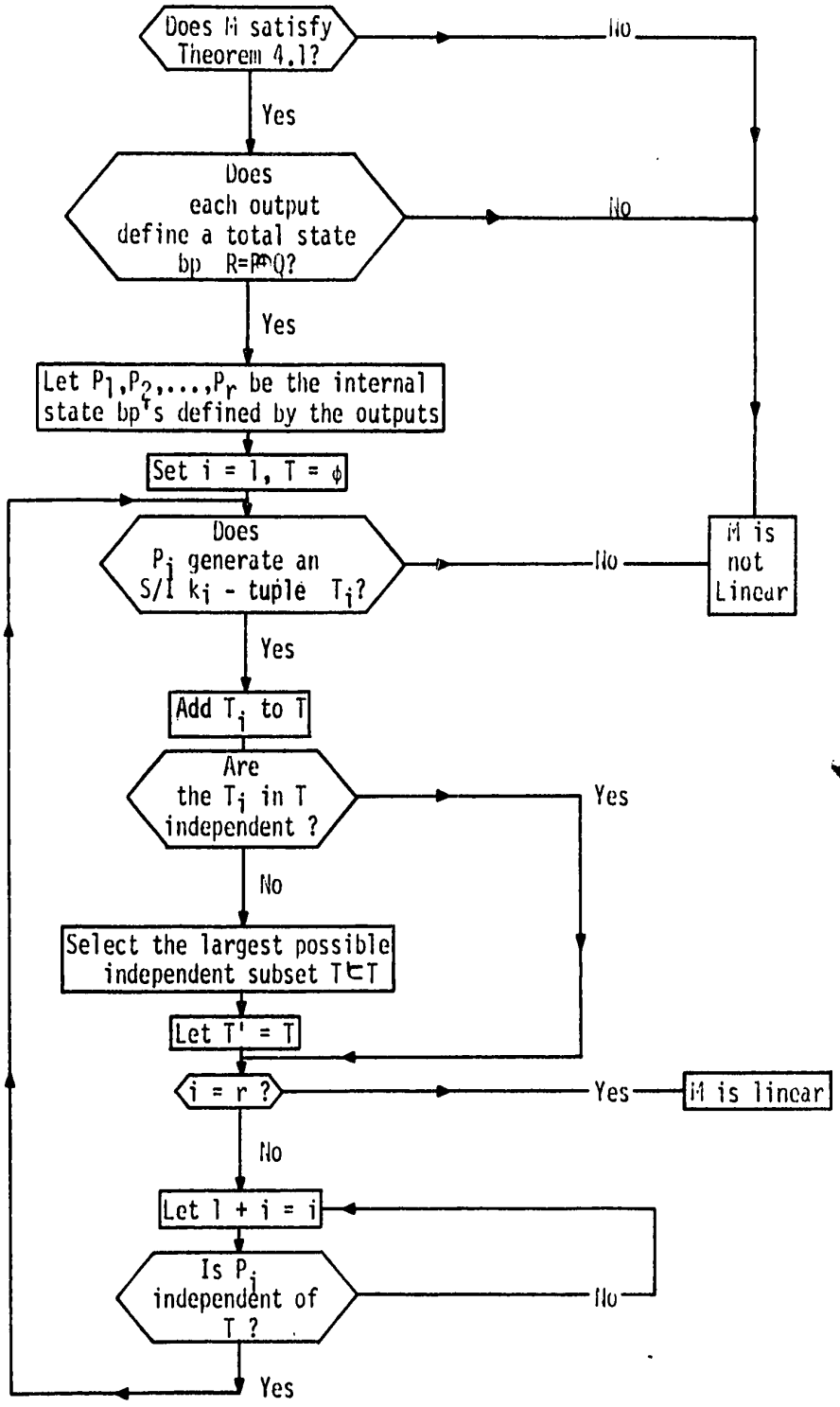

FIG. 4.2 LINEARITY ALGORITHM

To justify the algorithm, i.e., to prove that the answer produced is correct, we have the following theorem.

Theorem 4.3: If $T_1, T_2, \ldots, T_j$ are j independent S/I k-tuples that have been obtained from the output of a sequential machine M, and each output is a linear combination of the bp's contained in the T's then $T_1, T_2, \ldots, T_j$ provides a valid linear assignment for M.

Proof: Let $P_1, P_2, \ldots, P_n$ be the n state bp's contained in $T_1, T_2, \ldots, T_j$. Since each output can be expressed as a function of $P_1, P_2, \ldots, P_n$ and the input bp's, the outputs must be constant in each block of the product $P_1P_2 \ldots P_n$. Hence, if $P_1, P_2, \ldots, P_n$ do not provide a valid assignment for the states of M, then M is not reduced, since $P_1P_2 \ldots P_n$ is an S.P. partition that is output consistent. The assignment is linear from the definition of the k-tuples.

To illustrate, consider M4.4 given in Table 4.4.

Table 4.4

Machine M4.4

	1	2	3	z
A	E	B	C	0
B	B	E	A	0
C	B	E	A	1
D	A	C	B	1
E	E	B	C	1
F	B	E	A	0

Applying the algorithm we get:

$$\pi(CDE) = 1(AE) + 2(BCDF) + 3(AE)$$
$$= (AE) \oplus (2)$$

$$\pi(AE) = 1(ADE) + 2(BCF) + 3(BCF)$$
$$= (ADE) \oplus (23)$$

$$\pi(ADE) = 1(ADE) + 2(BCF) + 3(BCF)$$
$$= (ADE) \oplus (23).$$

The bp's produce the following assignment:

	y_1	y_2	y_3
A	0	1	1
B	0	0	0
C	1	0	0
D	1	0	1
E	1	1	1
F	0	0	0

From the assignment we note that B and F have the same assignment, and upon checking Table 4.4 we find, indeed, that B and F can be merged.

4.3 Next-State Behavior

In this section we investigate machines whose outputs are either not given or are nonlinear. In other words, we are considering only the next-state behavior of the machine. We initially consider linear permutation* machines and then extend the scope to include all machines realizable as linear machines, i.e., homomorphic image machines of linear machines.

---

* A machine M is a permutation machine if, and only if, every input permutes the set of states, that is, every next-state column contains all states of M.

4.3.1 Linear Permutation Machines

Given a permutation machine we want to determine whether or not there exists a linear assignment for the machine, i.e., is the machine linear?

Definition 4.8: Two distinct states S_i and S_j will be called a primary corresponding pair (pcp), denoted $[S_i;S_j]$, under inputs I_a and I_b, if and only if there exists a state S_k, such that either, $\sigma(S_kI_a) = S_i$ and $\sigma(S_kI_b) = S_j$, or $\sigma(S_kI_a) = S_j$ and $\sigma(S_kI_b) = S_i$.

Obviously a pcp is independent of the order in which the states are listed, i.e., $[S_i:S_j] = [S_j:S_i]$.

Lemma 4.3: In a linear sequential machine (singular or nonsingular) any two distinct pcp's under inputs I_a and I_b are disjoint.

Proof: The proof follows from Theorem 4.1.

Lemma 4.4: For every pair of distinct inputs I_a and I_b, in a flow table of a nonsingular linear sequential machine M containing q states, there exist q/2 distinct pcp's.

Proof: There are q pcp's $[\sigma(S_iI_a): \sigma(S_iI_b)]$ for $i = 1,2,\ldots,q$; however, M is nonsingular hence $\sigma(SI_a) = \sigma(SI_b) = S$ and each of the q states appears twice. Since two distinct pcp's are disjoint, there are exactly q/2 pcp's.

Obviously, from the above lemma, q is necessarily an even number for a nonautonomous linear sequential machine, which is nonsingular and fully specified.

It is also a consequence of the lemma that for any S_g of a nonsingular machine and any two inputs I_a and I_b the states $\sigma(S_g I_a) = S_i$ and $\sigma(S_g I_b) = S_j$ are distinct internal states, and that there exists a state S_h such that $\sigma(S_h I_a) = S_j$ and $\sigma(S_h I_b) = S_i$; i.e., the flow table has pattern shown in Table 4.5.

Table 4.5
Flow Table Pattern

	I_a ... I_b
S_g	S_i ... S_j
...	... ...
S_h	S_j S_i

This leads to the following definition:

Definition 4.9: Two states S_g and S_h are called a secondary corresponding pair (scp) denoted $\langle S_g : S_h \rangle$ under inputs I_a and I_b, if and only if there exists a pcp $[S_i : S_j]$ under I_a and I_b such that $\sigma(S_g I_a) = \sigma(S_h I_b) = S_i$ and $\sigma(S_h I_a) = \sigma(S_g I_b) = S_j$.

Having defined corresponding pairs (cp's) and proved some of their elementary properties, we can now state their use in finding a linear assignment.

Theorem 4.4: For every linear sequential machine M, and any two distinct inputs I_a and I_b, there exists a bp, P_c, which can be used to define a variable y_c in a valid linear assignment for M, and which separates all cp's under I_a and I_b.

Proof: See [19].

This now permits us to generalize the previously presented algorithm [19], which was restricted to minimal assignments. The complete algorithm will not be repeated; however, a brief summary follows:

1. Select any two inputs, and find all cp's.
2. Find a distinct bp, denoted P, separating all cp's.
3. Is P contained in an S/I k-tuple? If not return to step 2; otherwise continue to step 4.
4. If the state bp's contained in the k-tuples do not provide a valid assignment for the internal states, then find an S/I k-tuple independent from those already found.

This procedure suffers from the fault that a search may be required. We therefore take a slightly different approach to one-to-one assignments, and restrict the machines under consideration to strongly connected machines. A machine is <u>strongly connected</u> if, and only if, for every S_1 and S_2 in S there exists an input sequence x in I* which maps S_1 onto S_2,

$$\delta(S_1, x) = S_2,$$

where the next-state function is extended, by induction [64], to input sequences x in I*, the set of all sequences composed of elements in I. Since for a permutation machine the parts that are not strongly connected are disjoint, it is sufficient, when considering any permutation machine, to show that each strongly connected submachine is linear.

A machine is <u>uniformly connected</u> if, and only if there exists a positive integer p such that for every S_1 and S_2 in S there

exists an input sequence of length p which maps S_1 onto S_2.

Lemma 4.5: If M is a linear machine and for some a in I $\delta(S_1,a) = \delta(S_2,a)$ then $\delta(S_1,x) = \delta(S_2,x)$ for all x in I. Furthermore, if for some S_1 in S $\delta(S_1,0) = \delta(S_1,1)$ then $\delta(S_i,0) = \delta(S_i,1)$ for all S_i in S.

Thus a linear machine is either a permutation machine or it has identical next-state rows in its flow table.

Lemma 4.6: A uniformly connected, linear permutation machine M has 2^n states.

Proof: Consider a linear assignment containing $Y_0 = (00 \ldots 0)$, and let the matrix period for the characteristic matrix be k. Since M is uniformly connected, every state can be reached from every other state in kr steps. Hence if Y_0 is mapped onto Y_0' by an input sequence X of length kr, we have

$$Y_0' = Y_0 A^{kr} \oplus K_x,$$

where K_x is a constant determined by X. Since $A^{kr} = I$, we have

$$Y_0' = Y_0 \oplus K_x.$$

K the set of all possible K_x must equal Y the set of all m-tuples for M, because every state can be reached from Y_0 in kr steps. Furthermore, for any Y_1 and Y_2 in Y we have

$$Y_1 \oplus Y_2 = Y_1 \oplus K_i = Y_j,$$

where $Y_2 = Y_0 \oplus K_i$.

Hence Y is closed under $\oplus$, and is a subgroup of the set of all binary m-tuples. Hence, since the order of Y must divide 2^m, M must contain 2^n states for some $n \leq m$.

From the above proof, it is clear that for a linear permutation machine that is uniformly connected there exists a linear assignment of order n.

Lemma 4.7: A linear permutation machine whose A matrix is derogatory is not strongly connected.

Proof: Every machine with a derogatory A matrix is realizable as a number of $k > 1$ parallel shift registers. Therefore, if we can prove that the lemma holds for two fsr's, it can be extended to k parallel fsr's. Suppose that the machine is realizable as two parallel fsr's. Since A is derogatory [24] the characteristic polynomial of one of the fsr's, M_1, must divide the characteristic polynomial of the other fsr, M_2.

Since the characteristic polynomial of M_2 is factorable, M_2 can be realized by two fsr's in cascade, the first of which is isomorphic to M_1. Now since M_1 and the initial portion of M_2 are isomorphic and both dependent on the input, the over-all machine is not strongly connected, since if we start in a state where the internal variables of the isomorphic machines are equal we can never get to a state where the variables are unequal.

We now show the distinction between strongly connected and uniformly connected machines.

Lemma 4.8: Every strongly connected, linear permutation machine M that is not uniformly connected, contains an autonomous submachine.

Proof: Starting with any state S_i of S and applying the set of all sequences of length k (sufficiently large) we must reach some subset b_k of S. b_k is a nontrivial subset of S because M is not uniformly connected. Now consider the sequences of length $k + 1$, and again we reach some nontrivial subset, b_{k+1}, of S. If we continue, at some length $k + r$ we must reach b_k again. Therefore, since the b_j's for $k \leq j \leq k+r$ contain all states of S, and the transition between the b's is independent of the input, M must contain an autonomous submachine (i.e., clock [41,83]).

We now use the results of this lemma in an attempt to simplify the means of determining linear assignments for permutation machines.

Theorem 4.5: Every strongly connected, linear permutation machine M with a single binary input is realizable as an autonomous machine, which may be trivial, and a nontrivial single binary fsr, which is linear.

Proof: To prove the theorem, it is necessary to prove that every linear permutation machine that is uniformly connected, and whose characteristic matrix A is nonderogatory is realizable as a single binary fsr. Starting from any state S_i of S and applying the set of all input sequences of length k, denoted I^k, we must reach the set of all states, i.e., $\sigma(S_i, I^k) = S$. Clearly $n \leq k$, since if $n > k$ some states must have more than two successors. Now suppose $n < k$, which

means that by applying the set of all sequences of length n, from S_i we must reach $\leq 2^n$ states. Hence there must exist at least two sequences $X_n \neq X_n'$ of length n which have the same next state. Hence

$$S' = S_iA^n \oplus (x_1B \oplus C)A^{n-1} \oplus (x_2B \oplus C)A^{n-2} \oplus \ldots \oplus (x_{n-1}B \oplus C)A \oplus x_nB \oplus C =$$

$$S_iA^n \oplus (x_1'B \oplus C)A^{n-1} \oplus (x_2'B \oplus C)A^{n-2} \oplus \ldots \oplus (x_{n-1}'B \oplus C)A \oplus x_n'B \oplus C.$$

And

$$(x_1 \oplus x_1')BA^{n-1} \oplus (x_2 \oplus x_2')BA^{n-2} \oplus \ldots \oplus (x_{n-1} \oplus x_{n-1}')BA \oplus (x_n \oplus x_n')B = 0.$$

This is a contradiction, because if A is nonderogatory there cannot exist a polynomial in A, of degree $< n$, which is equal to zero. Hence $n = k$. Since there are 2^n states and n binary variables there exists a state $S_o = (00\ldots0)$. Furthermore, there exists a sequence X_n of length n such that $S_o = \delta(S_o, X_n)$. Hence

$$S_o = S_oA^n \oplus (x_1B \oplus C)A^{n-1} \oplus (x_2B \oplus C)A^{n-2} \oplus \ldots \oplus (x_{n-1}B \oplus C)A \oplus x_nB \oplus C$$

And

$$(x_1B \oplus C)A^{n-1} \oplus (x_2B \oplus C)A^{n-2} \oplus \ldots \oplus (x_{n-1}B \oplus C)A \oplus x_nB \oplus C = 0.$$

Since A is nonderogatory

$$x_iB \oplus C = 0 \quad \text{for } i = 1, 2, \ldots, n.$$

Therefore

$$S_o = S_oA \oplus x\ B \oplus C,$$

for some x.

We now assign to each state $S_i = (y_1y_2\ldots y_n)$ that n-tuple $(x_nx_{n-1}\ldots x_1)$ such that

$$S_i = \delta(S_o, X) \quad \text{and}$$

$$X = x_1x_2\ldots x_n.$$

It is easily seen that this assignment is linear and is a single binary fsr assignment for M.

This means that to find a linear assignment for a permutation machine we need only find the autonomous submachine and the remainder must be a single binary shift register, which is covered in Chapter 5.

To illustrate this approach, consider M4.6 given in Table 4.5

Table 4.6
Machine M4.5

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

It can be shown that the autonomous submachine of M4.5 is given by $P_1 = \{\overline{A,B,C,F};\ \overline{D,E,G,H}\}$. Furthermore, the single binary fsr is given by

$$P_2 = \{\overline{A,C,E,G};\ \overline{B,D,F,H}\},$$
$$P_3 = \{\overline{C,E,F,H};\ \overline{A,B,D,G}\}.$$

4.3.2 Many-to-One Assignments

In this section we investigate homomorphisms of the state behavior of linear machines in order to characterize those nonlinear machines whose state behavior can be realized by larger linear machines. Stated

differently, we are investigating those non-linear machines whose state behavior can be linearized by many-to-one assignments (i.e., with state-splitting).

It will be shown that for every machine M there exists a unique machine M_T, (to be defined) such that M is linearly realizable if, and only if M_T is a linearly realizable permutation machine. This means that if M_T is not a permutation machine, then M is not realizable as a linear machine.

We begin by considering permutation machines, and will show that a linearly realizable permutation machine must contain an autonomous submachine, and there must exist a one-to-one correspondence between any two next-state columns. We first show that uniformly connected linear permutation machines can only have linear image machines.

Theorem 4.6: If M is a uniformly connected, linear permutation machine, then all its homomorphic images M_p are linear.

Proof: Consider a linear assignment for M which contains $(0,0,\ldots,0) = Y_0$, and let the matrix period for this realization be k. Since M is uniformly connected, every state can be reached from every other state in km steps. Thus if Y is mapped onto Y' by an input string x of length km, $l(x) = km$, we have

$$Y' = Y A^{km} \oplus K_x,$$

where K_x is a constant determined by x. Since

$$A^{km} = I$$

we have

$$Y' = Y \oplus K_x.$$

Consider the block b of P which contains Y_o and let

$$C = \{K_x \mid \delta(b\ x) = b \text{ and } l(x) = km,\ m = 1,2,\ldots\}$$

Then for all Y_j in b and K_{cx} in C there exists a Y_i in b such that

$$Y_j \oplus K_{cx} = Y_i.$$

Since all the states in b can be reached from Y_o in km steps, we conclude that

$$b = C.$$

Furthermore, if K_x and K_y are in C and Y in b then

$$Y' = Y \oplus K_x \oplus K_y \text{ is in } b$$

and therefore there is a K_z in C such that

$$K_x \oplus K_y = K_z.$$

This implies that C is a linear subspace of the set of all n-tuples of M.

Consider any other block b_1 of P. This block can be reached from b in km steps and therefore there exists a K_x such that for any Y_j in b and any Y_{1i} in b_1

$$Y_j \oplus K_x = Y_{1i}.$$

But then

$$Y_{1i} \oplus K_{cx} = Y_j \oplus K_x \oplus K_{cx} = Y_i \oplus K_x = Y_{1j}$$

and we conclude that C is the set of vectors that leave the blocks of P invariant.

Since C is a linear subspace we map by a non-singular linear transformation C onto the last β variables. In this new linear assignment for M the first $\alpha - \beta$ variables are constant in each block of P and different for different blocks of P. Thus the first $\alpha - \beta$ variables yield a one-to-one state assignment for M_p. Furthermore, this is a linear assignment for M_p, since P has the substitution property and therefore the input and the first $\alpha - \beta$ variables uniquely determine the first $\alpha - \beta$ variables for the next state of M_p. Thus by just removing the last β variables from the linear realization of M we get a linear realization of M_p.

The previous theorem can be sharpened and we show next that even if a linear permutation machine is not uniformly connected its uniformly connected image machines are linear.

Theorem 4.7: If M is a linear permutation machine and its homomorphic image M_p is uniformly connected then M_p in linear.

Proof: If M does not contain a clock it is uniformly connected, and by Theorem 4.6 the image machine M_p is linear.

If M is not uniformly connected then there exists an r such that the sets of states which can be reached from S_i in r+1, r+2, ..., r+p steps

$$b_{r+1}, b_{r+2}, \cdots, b_{r+p},$$

are the blocks of an S.P. partition P_c. Furthermore, the image machine

$$M_{P_c}$$

is the largest autonomous image of M.

Consider now a linear assignment for M which contains the code Y_o and let Y_o be contained in b of P_c.
Let the matrix period be k and let

$$C = \{K_x \mid \delta(b,x) = b \text{ and } l(x) = km, \quad m = 1, 2, \ldots\}.$$

Since every state of b can be reached from Y_o in km steps we conclude that

$$C = b$$

and that b is a linear subspace which leaves the blocks of P_c invariant, i.e., for $Y_i \in b_i$ and $Y \in b$

$$(Y_i \oplus Y) \in b_i.$$

We will now show that if M_p is a uniformly connected image machine of M then M_p is linear.

Let d be the block of P which contains Y_o and let

$$C' = \{K_x \mid \delta(Y_o,x) \in d \text{ and } l(x) = km, \quad m = 1, 2, \ldots\}$$

Then C' is a linear subspace, for every d_i of P, and $Y_j \in d_i$ $K_x \in C'$ we have

$$(Y_j \oplus K_x) \in d_i$$

$$C' \subseteq d \text{ and } C' \subseteq b \text{ (of } P_c).$$

By a non-singular linear transformation we now obtain a new linear assignment in which C = b is mapped onto the first β variables and C' is mapped onto the last η variables of these first β variables. We will show that the first $\beta - \eta$ variables of this new

assignment form a linear assignment for M_P. To do this we have to show that the first $\beta - \eta$ variables are constant in the blocks of P and have different values for different blocks of P. By construction C' is the largest subspace of b which leaves the blocks of P invariant. Therefore the first $\beta - \eta$ variables (are not affected by C' and) stay constant in the blocks of P. To see that the first $\beta - \eta$ variables are different in different blocks we recall that M_P is uniformly connected and therefore every block of P can be reached from S_i in r, r+1, ... steps. This implies that if Y is in d of P, then d contains all the states of M whose codes differ from Y only in the clock variables. The same is true for the last η variables of the first β variables. Thus the states in different blocks of P must differ in the first $\beta - \eta$ variables, and we have a one-to-one assignment. Since P has S.P. and since first $\beta - \eta$ variables come from a linear assignment we conclude that M_P is a linear machine.

The next example shows that a linear permutation machine which is not uniformly connected can have non-linear image machines. Machine M4.6, given in Table 4.7, is easily shown to be a linear machine. On the other hand, the S.P. partition

$$P = \{\overline{0,1};\ \overline{2,3};\ \overline{4,5};\ \overline{6,7}\}$$
$$= \{\overline{A};\ \overline{B};\ \overline{C};\ \overline{D}\}$$

defines the image machine M4.6p shown in Table 4.8.

Table 4.7

Machine M4.6

	0	1
0	4	6
1	6	4
2	5	7
3	7	5
4	2	0
5	0	2
6	3	1
7	1	3

Table 4.8

Machine M4.6p

	0	1
A	C	D
B	D	C
C	A	A
D	B	B

Since $\delta(C,0) = \delta(C,1)$ and $\delta(B,0) \neq \delta(B,1)$ machine M4.6p is not linear because of Lemma 4.5.

We therefore need a method for determining whether a nonlinear permutation machine is the image machine of a linear machine.

Corollary 4.1: A nonlinear permutation machine M is the homomorphic image of a linear machine if, and only if:

(a) there exists a non-trivial S.P. partition P such that M_P is autonomous.

(b) there exists a symmetric one-to-one correspondence between next states in any two input columns of M.

(c) there exists a linear machine $M_{1/p}$, such that M is an image machine of the serial connection [41] of M_p and $M_{1/p}$.

Parts (a) and (b) permit the elimination of a number of permutation machines as not being linearly realizable. The problem now occurs, however, if a nonlinear machine satisfies parts (a) and (b) to determine if part (c) is satisfied. This problem, unfortunately, has not yet been completely solved.

We have considered permutation machines, and given the properties necessary to realize a permutation machine as a linear permutation machine, we now relate these results to the class of all sequential machines.

Lemma 4.9: Every linear machine may be decomposed into a parallel connection [41] of a linear permutation machine and a linear definite machine [4].

Proof: See Crowell [8].

The next example shows that a linear definite machine which is strongly connected can have non-linear image machines. Machine M4.7, given in Table 4.9 is a two stage binary shift register and thus a linear definite machine. On the other hand, the S.P. partition

$$P = \{\overline{0,1};\ \overline{2};\ \overline{3}\} = \{\overline{A};\ \overline{B};\ \overline{C}\},$$

defines the image machine M4.7p shown in Table 4.10.

Table 4.9

Machine M4.7

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

Table 4.10

Machine M4.7p

	0	1
A	A	B
B	A	C
C	A	C

Since

$$\delta(A,0) = \delta(B,0) \text{ and } \delta(A,1) \neq \delta(B,1)$$

machine M4.7p is not linear by Lemma 4.5.

Our next result shows that such machines will always have non-linear image machines. We first show that any machine, like M4.7p, which has n states and n-1 different next state rows is not linear.

Lemma 4.10: A strongly connected n state machine, for $n \geq 3$, with exactly n-1 distinct next state rows is not linear.

Proof: Since two distinct states of M are mapped onto the same state the matrix for the linear assignment must be singular. Therefore, we can transform this assignment by a non-singular linear transformation

onto another assignment in which the last variable y_α does not depend on the previous state. But then all the states in $\delta(S,0)$ have the same last variable, say $y_\alpha = 0$. Since M is strongly connected and one state is missing from $\delta(S,0)$ this state must appear in $\delta(S,1)$ and furthermore it must have $y_\alpha = 1$. But then all states in $\delta(S,1)$ must have $y_\alpha = 1$, which leads to a contradiction, since only one state can have $y_\alpha = 1$. Thus M is not linear.

Corollary 4.2: Let M be a strongly connected machine with more than two states and at least two identical next-state rows (i.e., M is not a permutation machine). Then M has a non-linear homomorphic image machine.

Proof: Our proof shows that any machine satisfying the hypothesis of the theorem can be mapped homomorphically onto a machine with more than two states and with exactly two identical next state rows, which by the proceeding lemma is non-linear.

If M has exactly two identical next state rows (and more than two states), then it is not linear. If M has more than two identical next state rows, then let P be a partition which identifies all but two of the identical next state rows. The partition P defines a homomorphism of M onto M_P. Clearly M_P must have at least three states and if it has exactly three states, then it has exactly two identical next state rows (otherwise it is not strongly connected and therefore M is not strongly connected) and is not linear. If M_P has more than two identical next state rows, then it has more than three states and we repeat the process

of taking its homomorphic images as above and by induction in a finite number of steps, we obtain a non-linear image machine.

In the preceding proof we obtained the non-linear image machine by partially collapsing the feedback-free part. If the homomorphism identifies all the states with identical next-state rows of a linear machine then the image is again linear. To make this precise we introduce some notation.

Let $M(0) = \tau_1$ be the partition on S of M which is obtained by identifying all states with identical next state rows, that is

$S_i = S_j(\tau_1)$ if, and only if, for all x in I $\delta(S_i,x) = \delta(S_j,x)$. This partition has the substitution property and defines an image machine M_{τ_1}. If M_{τ_1} has identical next-state rows we can repeat this process and arrive, in a finite number of steps, at the largest homomorphic image machine M_τ, which has no identical next-state rows. The S.P. partition τ in the notation of [41] is given by

$$\tau = M^k(0) \text{ for } k \text{ such that } M^k(0) = M^{k+1}(0).$$

Note that any machine M can be realized by a serial connection of M_τ to a feedback-free tail machine [41].

Lemma 4.11: If M is a linear machine then M_τ is also linear.

Proof: If M has identical next-state rows then the matrix is singular and by a linear transformation we can obtain a new linear assignment in which the variables of the new state do not depend on the last β, $\beta \geq 1$, variables of the old state. Thus we can remove the β variables

and obtain a linear image machine. By repeating this process a finite number of times we arrive at the linear image machine M_τ.

Combining the results with Theorem 4.7 we obtain the next theorem.

Theorem 4.8: If M is a linear machine and its image M_P is uniformly connected and has no identical next-state rows then M_P is linear.

We now consider the problem of recognizing when a nonlinear machine is a homomorphic image of a linear machine.

Theorem 4.9: Every homomorphic image machine M of a linear machine M' is such that M_τ is a linearly realizable permutation machine.

Proof: Since M is an image machine of M', there exists an S.P. partition P such that $M = M'_P$. Furthermore, let τ' be that S.P. partition on M' such that $M'_{\tau'}$ is the largest image machine having no identical next-state rows. If $P \leq \tau'$ or $P \geq \tau'$ then the theorem obviously holds. Suppose $P \not\leq \tau'$, $P \not\geq \tau'$, and that the first α variables of the linear assignment for M' define the linear permutation machine $M'_{\tau'}$ and the last $n - \alpha$ variables define a feedback free machine. Since the first α variables are independent of the last $n - \alpha$ variables, any S.P. partition on the states of M' must map the first α variables onto a linearly realizable permutation machine (the argument is similar to the proofs of Theorems 4.6 and 4.7). In addition, since M is not a permutation machine, M must have identical next-state rows (i.e., the feedback free portion is not completely collapsed).

Hence, by identifying all identical next-state rows of M to produce M_τ we conclude that M_τ is a linearly realizable permutation machine.

We can now combine these results with the observation that any feedback-free machine is realizable as a linear machine, producing the following:

Corollary 4.3: A nonlinear machine M is the homomorphic image of a linear machine if and only if M_τ is a linearly realizable permutation machine.

This result shows that the non-linear machines which can be realized by larger linear machines are exactly those machines which are obtained by a serial connection of a linearly realizable permutation machine to a feed-back-free machine. If all the non-linearity of the machine is not in its feedback-free tail part, then M_τ must contain a non-trivial clock or else no linear machine can be used to realize it.

We can make another observation;

Corollary 4.4: If M is not a permutation machine and has no identical next-state rows then M is not a homomorphic image of a linear machine.

This result permits us to construct large classes of machines which are not homomorphic images of linear machines. For example, any machine which has no identical next state rows, but which has some input which maps two states onto identical next states, is not an image of a linear machine.

An interesting by-product of the above results for linear machines is a result for logic-free machines. Brzozowski [5], has shown

that the state behavior of every definite machine is realizable without logic. Brzozowski and Hawtrey [7a]consider logic-free realizations in general and present a method for finding logic-free assignments. Since a logic-free machine is trivially linear, and it can be shown that a permutation machine is realizable without logic if and only if it is autonomous we have:

Corollary 4.5: A machine M is realizable without logic if, and only if M_τ is autonomous.

4.4 Conclusion

In summary we have:

For autonomous machines with the output specified, we have the Cohn and Even [11]algorithm and the algorithm presented in Section 4.2, each of which covers all cases. For autonomous machines where we are only interested in the next-state behavior, we have the methods of Elspas [24], Gill and Tan [32] and Gill [30,31]. These, however, are restricted to machines having 2^n states and realizations using n memory elements. It is easily shown that the next-state behavior of every autonomous sequential machine is realizable as a linear machine. In addition, it can also be shown that there exist autonomous machines with 2^n states that are not realizable as linear machines with n memory elements. A method has been presented, Brzozowski and Davis [7], for determining minimum linear assignments for autonomous machines. This method, however, suffers from the unfortunate fact that a search— which could prove to be extensive—is required.

We conclude therefore that the problem of determining the minimum linear realization for the next-state behavior of a given autonomous machine has not been solved by any of the previously mentioned work.

For nonautonomous machines with the output specified, again the Cohn and Even [11] algorithm and the algorithm in Section 4.2 cover all cases. With respect to the next-state behavior, the work of Srinivasan [76,77], Hartmanis [40] and Yau and Wang [84] covers only a restricted class of machines, for one-to-one assignments. Section 4.3 covers all cases, but does not produce an easy method for nonlinear machines having an autonomous image machine and a symmetric one-to-one correspondence between any two input columns.

Chapter 5

SHIFT REGISTER REALIZATIONS

In this chapter we consider the problem of determining shift register realizations for a given sequential machine. It is clear [3,53] that every sequential machine having q internal states is trivially realizable as $n = [\log_2 q]$ shift registers of length 1. It will be proven in Section 5.3 that there exists a machine that is not realizable as a single binary shift register. Therefore, for a given machine, the problem is to determine how few shift registers are necessary to realize the machine.

We now summarize the existing methods, which are presented in Section 5.2, for solving this problem. Given an arbitrary sequential machine M:

1. A test exists [53,55] that will determine whether or not M is realizable (one-to-one assignment) as a multi-level shift register of length $k > 1$.

2. It is possible to determine [80] whether or not M is realizable (one-to-one assignment) as a number of variable length shift registers (also multi-level).
3. An algorithm [37] is available which will find a single binary shift register realization for any single column of M.
4. Several algorithms [66,79] exist that will determine whether or not M is realizable (one-to-one assignment) as a single binary shift register.
5. An algorithm exists [65,66] for finding minimal one-to-one assignments for multiple binary shift registers of varying lengths.

In Sections 5.3, 5.4 and 5.5 the following results are presented. Given an arbitrary sequential machine M:

1. A simple straightforward way of determining the existence of a one-to-one single, binary, shift register assignment for M (without transient states).
2. A method (nonalgorithmic) for finding a single binary shift register realization for M.
3. If every state of M has two distinct next states, a method is presented for determining if M is not realizable as a single binary shift register.
4. An approach is presented for finding multiple shift register assignments (one-to-one) for M.

5. A logical design procedure is presented, using a generalized model, for determining shift register realizations.

5.1 Shift Register Properties

A single feedback shift register (fsr) of order n, having a single input x, will be considered to be in the form shown in Figure 5.1. The shift register is binary, since we have restricted ourselves to binary realizations. While other forms of fsr's have been considered, [58,80], we will restrict this discussion to the form shown.

The next state equations of a single fsr are given by:

$$y'_{i+1} = y_i, \quad \text{for } i = 1, 2, \ldots, n-1,$$
$$y'_1 = f(y_1, y_2, \ldots, y_n, x),$$

where f is termed the feedback function. If f is a function of only the input (i.e., a trivial function of the internal state variables $y_1, y_2, \ldots, y_n$), the shift register is not a feedback shift register in the strict sense. However, since the following treatment holds for either case, we will not distinguish between the two and consider both as fsr's.

Any output of an fsr is given by:

$$z = g(y_1, y_2, \ldots, y_n, x),$$

where g is termed the output function. It is clear that, if we can realize the internal state behavior of a sequential machine as an fsr such that each internal state is identified uniquely, then the realization of the output becomes a combinatorial problem, since the output is simply a

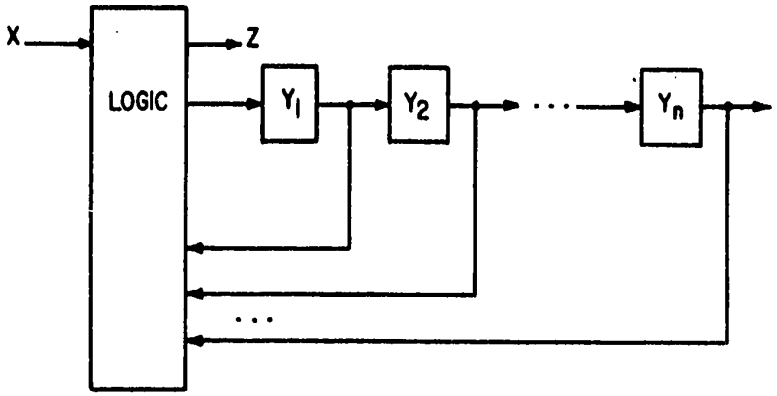

FIG. 5.1 - A SINGLE FEEDBACK SHIFT REGISTER

Boolean function of the internal state variables and the input variable. We therefore omit the output from the flow tables under consideration, and restrict ourselves to the internal state behavior of the machine.

The following properties have not only been proved [56] previously for single fsr's, but the proofs follow from the next state equations, and hence are omitted for the sake of brevity.

1. Each internal state J can have only two possible successors called primary corresponding pairs (pcp's) and denoted $[A:B]$ where $A \neq B$.
2. If $[A:B]$ and $[C:D]$ are two pcp's and $A = C$, then $B = D$.
3. Each internal state J can have only two possible predecessors called secondary corresponding pairs (scp's) and denoted $\langle A:B \rangle$ where $A \neq B$.
4. If $\langle A:B \rangle$ and $\langle C:D \rangle$ are two scp's and $A = C$, then $B = D$.
5. Each variable y_i, $1 \leq i < n$, must identify all scp's.
6. Each variable y_j, $1 < j \leq n$, must identify all pcp's.

It is clear from the last two properties that every variable y_k in an fsr, $k = 2, 3, \ldots, n-1$, must identify all corresponding pairs (cp's).

Theorem 5.1: For every fsr where $y_i' = y_{i-1}$, for $i = 2, 3, \ldots, n$, and $y_1' = f(y_1, y_2, \ldots, y_n, x)$, the bp's P_j, determined by the variable y_j for $j = 1, 2, \ldots, n$, must have the following properties:

1) P_1 must separate all pcp's and identify all scp's.
2) $P_2, P_3, \ldots, P_{n-1}$ must identify all cp's.
3) P_n must identify all pcp's and separate all scp's.

4) The blocks b_j and $\bar{b}_j$ are ordered so that $\pi b_j = b_{j-1}$, for $j = 2, 3, \ldots, n$ where πb is the union of the predecessors of the states of b.

Proof: The theorem follows directly from the definition and properties of an fsr.

Definition 5.1: The n-tuple of bp's $P_1, P_2, \ldots, P_n$, such that $\pi b_i = b_{i-1}$ for $i = 2, 3, \ldots, n$, representing a valid assignment for an fsr is called a bp n-tuple.

We will consider only machines with a binary input (i.e., two column machines) since the generalization to the multiple input case is straightforward. To illustrate, suppose we have a multiple input machine. If any state of the machine has more than two distinct successors then we can immediately say that the machine is not realizable as a single binary shift register (Property 1). If, however, no state of the multiple input machine has more than two distinct successors then we can find a binary input machine which is related to the given machine such that if the binary input machine is realizable as an fsr then both machines are realizable. Furthermore both machines will have the same state assignment, but different feedback functions. The related machine is merely that machine obtained by listing the two or less distinct successors of each state. Note that this property was used by Sheng [73] where he interchanged next-states until the machine became definite, i.e., made the feedback function trivial.

5.1.1 Properties of Multiple fsr's

In a multiple fsr the number of shift registers is termed the multiplicity of the fsr. In the multiple fsr shown in Fig. 5.2 the multiplicity is p, where the orders of the shift registers are $n_1, n_2, \ldots, n_p$ respectively. It is clear that we can extend the properties previously given for single fsr's to multiple fsr's. We have:

1. Each internal state J of a multiple fsr with multiplicity p, can have 2^p possible successors, called the primary corresponding set (pcs).
2. Each internal state J of a multiple fsr can have 2^p possible predecessors, called the secondary corresponding set (scs).

In the multiple fsr case, since we consider a pcs to be the set of states having a common predecessor and an scs to be the set of states having a common successor, then properties 5 and 6 must also hold. The proof follows directly from the proof for a single fsr and the definition of a multiple fsr, and will not be repeated. Furthermore, Theorem 5.1 also holds for the multiple fsr, but for a clearer understanding we restate the theorem for this case:

Theorem 5.2: For every multiple fsr where

$y'_{ij} = y_{i(j+1)}$, and

$y'_{i1} = f_i(y_{11}, y_{12}, \ldots, y_{pn_p}, x)$,

for $i = 1, 2, \ldots, p$ and $j = 2, 3, \ldots, n_i$ the bp's P_{kr}

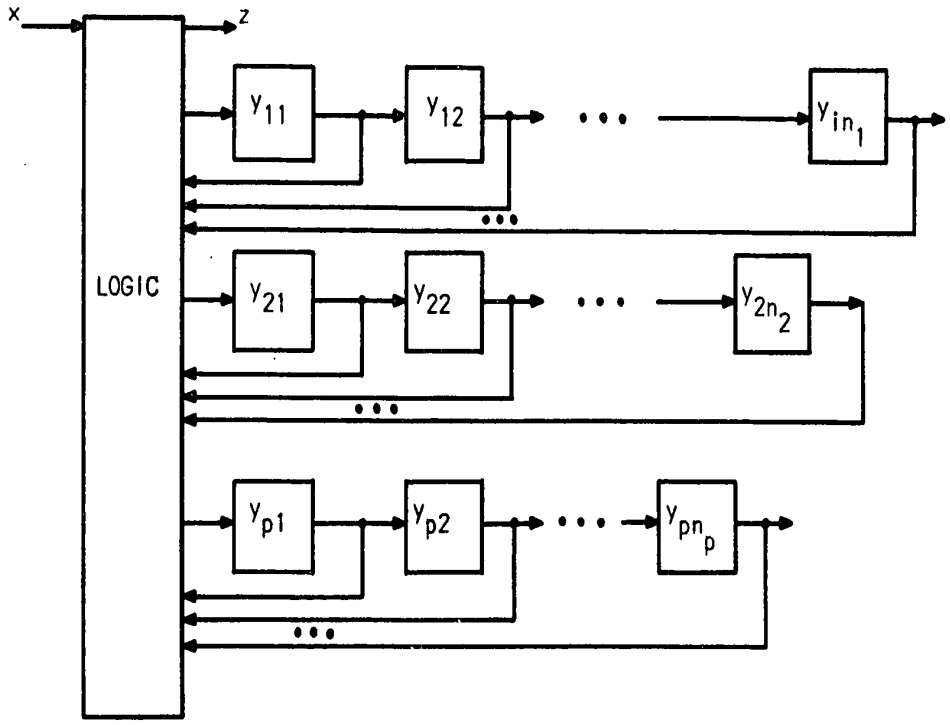

FIG. 5.2 - A MULTIPLE FEEDBACK SHIFT REGISTER

determined by the variables y_{kr} for $k = 1, 2, \ldots, p$ and $r = 1,2,\ldots,n_k$ must have the following properties:

1) $(P_{11}P_{21}\ldots P_{p1})$ must separate all pcs's.
2) $P_{11}, P_{12}, \ldots, P_{1\,(n_1-1)}, P_{21}, \ldots, P_{p\,(n_p-1)}$ must identify all scs's, for $n_i > 1$.
3) $P_{12}, P_{13}, \ldots, P_{1n_1}, P_{22}, \ldots, P_{pn_p}$ must identify all pcs's.
4) $(P_{1n_1}P_{2n_2}\ldots P_{pn_p})$ must separate all scs's.
5) The blocks b_{kr} and $\bar{b}_{kr}$ are ordered, so that $\pi b_{kr} = b_{k(r-1)}$ for $k = 1, 2, \ldots, p$ and $r = 2, 3, \ldots, n_i$.

Proof: Again the proof of the theorem follows directly from the definitions and properties.

The definition of a bp k-tuple also applies to multiple fsr's with the proviso that there exist p k-tuples of orders $n_1, n_2, \ldots, n_p$ respectively, and the $h = n_1 + n_2 + \ldots + n_p$ bp's represent a valid assignment for the entire machine.

5.2 Summary of Previous Results

C.L. Liu [53,55,79] was one of the first to publish any results on the problem of realizing sequential machines as shift registers. He initially [53,55] considers one-to-one assignments and p-nary shift registers. P_1 is defined to be the trivial 0 partition, where no states are identified. P_k for $k > 1$ is defined as follows: Define a relation T_k, for

$k = 2,3,\ldots$ over the P_{k-1} classes: for any P^i_{k-1} and P^j_{k-1} among the P_{k-1} classes, $P^i_{k-1}\ T_k\ P^j_{k-1}$ if both P^i_{k-1} and P^j_{k-1} can be reached from states that are in one P_{k-1} class after one transition. Let R_k be the relation that is the transitive extension of T_k. Let P_k be the partition over the P_{k-1} classes induced by R_k. In [53] he proves:

(a) A sequential machine is realizable as a shift register of length 2 if, and only if, any two states in the same P_2 class enter different P_2 classes after one step of transition corresponding to all input symbols.

(b) A sequential machine is realizable as a shift register of length 3 if

i) it is realizable as a shift register of length 2, and

ii) in the listing of P_3 classes, any two P_2 classes in the same P_3 class enter different P_3 classes after one step of transition.

In [55] Liu extends, without proof, the above to realizations for shift registers of length k: A sequential machine is realizable as an fsr of length k if, and only if,

(a) the machine is realizable as a shift register of length $k - 1$, and

(b) any two P_{k-1} classes in one P_k class go to different P_k classes after one step of transition.

To illustrate, consider the sequential machine M5.1 given in Table 5.1.

Table 5.1

Machine M5.1

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

It is seen that $P_2 = \{\overline{A,E};\ \overline{B};\ \overline{C};\ \overline{D}\}$, and since A and E both enter different P_2 classes after one step of transition, we can state that M5.1 is realizable as a shift register of length 2. Again, $P_3 = \{\overline{A,D,E};\ \overline{B};\overline{C}\}$ and the P_2 classes AE and D go to different P_3 classes after one step of transition, and M5.1 is realizable as a shift register of length 3. Lastly, $P_4 = \{\overline{A,C,D,E};\ \overline{B}\}$ and the P_3 classes ADE and C go to different P_4 classes after one step of transition and M5.1 is realizable as a shift register of length 4.

Once the $P_1, P_2, \ldots, P_k$ partitions are found to satisfy the necessary and sufficient conditions, an assignment procedure is presented, but will not be repeated. Liu then proceeds [55] to extend this method to many-to-one assignments by creating equivalent states, i.e., whenever the requirements fail, an equivalent machine is created, by adding states, that satisfies the requirement. Since Liu's procedure is rather long and non-algorithmic, we will not attempt to summarize it here.

In [79] Liu attempts to overcome the difficulties encountered, by giving an assignment procedure for varying length registers. Given the series of partitions $P_1, P_2, \ldots, P_{k_r}$, not all satisfying the necessary on sufficient conditions, the 'generalized assignment procedure' will be:

(a) Assign a distinct integer to each P_{k_r} class.

(b) Following the assignment procedure, assign to each state an ordered k_r-tuple.

(c) Suppose that P_i is the highest level partition that does not satisfy the necessary and sufficient condition. Then there must be states that are in different $P_{k_{(r-1)}} + 1$ class a distinct integer.

(d) Again, following the assignment procedure given above, assign to each state an ordered k_{r-1}-tuple.

(e) Repeat (c) and (d) for the next highest level partition that does not satisfy the necessary and sufficient conditions until every state can be identified by the ordered k_r-tuple, ordered k_{r-1}-tuple, etc. assigned to it.

To illustrate, consider machine M5.2 given in Table 5.2. We have

$P_2 = \{\overline{A,B};\ \overline{C};\ \overline{D,E};\ \overline{F,G};\ \overline{H,I};\ \overline{J,K};\ \overline{L}\}$

$P_3 = \{\overline{A,B,C,D,E};\ \overline{F,G,H,I};\ \overline{J,K,L}\}$

It can be shown that P_3 does not satisfy the requirements, and following steps (a) and (b) of the procedure, we produce the following assignment for the states of M5.2

A	121
B	123
C	112
D	121
E	123
F	212
G	211
H	231
I	233
J	311
K	312
L	331

Table 5.2

Machine M5.2

	0	1
A	F	G
B	H	I
C	D	E
D	F	F
E	I	H
F	A	B
G	C	C
H	J	K
I	L	L
J	C	C
K	E	D
L	K	K

Continuing with step (c), we see that P_2 classes (AB) and (DE) do not satisfy the necessary and sufficient conditions; we therefore assign the

integers 1 and 2 respectively, producing the assignment:

A	11
B	12
C	22
D	21
E	22
F	11
G	12
H	21
I	22
J	12
K	12
L	21

This concludes the process and produces a realization having one ternary shift register of length 3, and one binary register of length 2.

Liu [79] then considers the problem of finding binary assignments for sequential machines satisfying the requirements for realization as a shift register of length k. The procedure presented, however, is cumbersome and not particularly effective.

Haring, [37,79], begins by considering autonomous (i.e., single column) machines. Here he proves that an arbitrary autonomous machine is realizable as a single binary fsr. While the procedure does not necessarily determine a minimal machine, an attempt is made to keep the machine as small as possible. The main problem in an fsr realization for an autonomous machine is to determine P_1 (i.e., the first binary partition in the n-tuple $P_1, P_2, \ldots, P_n$). He first considers permutation machines*, which

* Brzozowski [5] has also shown that every autonomous permutation machine can be realized as a single binary fsr.

are characterized by a set of disjoint cycles. Starting with any state S_i of a single cycle, we partition the states in the cycle by putting S_i in $\bar{b}_1$, S_{i+1} and S_{i+2} in b_1, S_{i+3}, S_{i+4}, S_{i+5}, in b_1, etc. (where $\sigma S_j = S_{j+1}$). If the machine has two or more cycles, and none of the cycle lengths are equal, then we consider each cycle separately with the preceeding procedure, making an attempt to distribute the number of states as evenly as possible between the two blocks. Haring has proved [37,79] that this procedure will produce an fsr realization for any autonomous permutation machine that does not have two cycle lengths that are equal. For cycles of equal length, there is another consideration since in an fsr of length n, $I_2(\lambda)$* is the maximum number of equal length cycles that can occur. Incidentally, the maximum occurs when $n = \lambda - 1$. We can overcome this restriction by adding equivalent states to increase a cycle of length λ to a cycle of length $k\lambda$, where $k = 2, 3, \ldots$.

As an example, consider machine M5.3 in Table 5.3. In M5.3 we have cycles of length 4,3,3,3 and 6. Since $I_2(3) = 2$, we will have to expand one of the cycles of length 3. Let the expanded cycle be E,F,G,E',F',G',E,... as shown in Table 5.4. Starting off with the binary partition P_1 given by $b_1 = (A,D,E,E',G',H,K,M,O.P)$ we find that it is necessary to have a shift register at length 7 to realize M5.3. The assignment found is given in Table 5.5. We now turn to autonomous machines containing transient states, i.e., nonpermutation machines. For nonpermutation machines, the procedure is changed somewhat in that instead of finding P_1 first, we start by finding P_n, i.e., the last partition of the shift register.

* Where $I_2(\lambda)$ is the number of irreducible (Modulo 2) polynomials of degree λ [37,79].

Table 5.3

Machine M5.3

S	S
A	B
B	C
C	D
D	A
E	F
F	G
G	E
H	I
I	J
J	H
K	L
L	M
M	K
N	O
O	P
P	Q
Q	R
R	S
S	N

Table 5.4

Machine M5.3'

S	S'
A	B
B	C
C	D
D	A
E	F
F	G
G	E'
E'	F'
F'	G'
G'	E
H	I
I	J
J	H
K	L
L	M
M	K
N	O
O	P
P	Q
Q	R
R	S
S	N

Table 5.5

Shift Register Assignment for M5.3'

A	0	0	1	1	0	0	1
B	1	0	0	1	1	0	0
C	1	1	0	0	1	1	0
D	0	1	1	0	0	1	1
E	0	0	0	0	1	1	0
F	1	0	0	0	0	1	1
G	1	1	0	0	0	0	1
E'	0	1	1	0	0	0	0
F'	0	0	1	1	0	0	0
G'	0	0	0	1	1	0	0
H	0	1	1	0	1	1	0
I	1	0	1	1	0	1	1
J	1	1	0	1	1	0	1
K	0	0	1	0	0	1	0
L	1	0	0	1	0	0	1
M	0	1	0	0	1	0	0
N	1	1	1	1	0	0	1
O	0	1	1	1	1	0	0
P	0	0	1	1	1	1	0
Q	1	0	0	1	1	1	1
R	1	1	0	0	1	1	1
S	1	1	1	0	0	1	1

The procedure first assigns the non-transient states (ie., the contained permutation machine) to P_n, using the procedure given previously for permutation machines. If no state has more than two predecessors, then all that is necessary is that P_n separate all states having common successors.

For example, consider M5.4 given in Table 5.6. If P_5 given by $b_5 = (A,D,F,H,J,L)$ then we have an fsr realization of length 4 given by P_1,P_2,P_3,P_4,P_5 where $b_1,b_2,b_3,b_4 = (A,C,G,J,K,L)$, (B,D,E,F,H,I,K,L), (A,C,G,H,I,J), (B,F,G,J,K,L). For nonpermutation machines containing a state having more than two predecessors, then it is again necessary to split

states, i.e., find an equivalent machine such that no state has more than two predecessors.

Table 5.6

Machine M5.4

S	S'
A	B
B	A
C	B
D	C
E	C
F	A
G	F
H	G
I	G
J	F
K	H
L	H

To illustrate, we have machine M5.5 given in Table 5.7.

Table 5.7

Machine M5.5

S	S'
A	C
B	C
C	B
D	I
E	I
F	I
G	H
H	C
I	B
J	D

States C and I both have more than two predecessors, hence we must split both. One possibility is M5.5' given in Table 5.8.

Table 5.8

Machine M5.5'

S	S'
A	C'
B	C'
B'	C
C	B
C'	B'
D	I
E	I'
F	I
G	H
H	C
I	B
I'	B'
J	D

It is easily seen that M5.5 is an image machine of M5.5'. Furthermore, since no state of M5.5' has more than two predecessors, it is possible to obtain a shift register assignment with the preceeding method. This completes the procedure for finding a single binary fsr realization for any autonomous sequential machine.

Haring then attempts to generalize this procedure to non-autonomous (i.e., multi-column) machines. He has the following procedure for determining all one-to-one fsr assignments for binary input machines:

1. Derive the 4 column machine M+ with columns $X_1^{-1}X_2$, $X_2^{-1}X_1$, $X_1X_2^{-1}$ and $X_2X_1^{-1}$ from M, where: X_i^{-1} means the inverse under X_i, i.e., if the next state of S_k under X_i is S_j, then the next state of S_j under X_i^{-1} is S_k. $X_1^{-1}X_2$ is equivalent to applying X_1^{-1} and then X_2.

2. Calculate all binary S.P. partitions of $M+$.

3. Select one of the S.P. partitions P from 2, that:

   a) $P \geq M+(0) + m^*(I)$ where $m^*(I)$ is that partition on S obtained by identifying all states having a common predecessor under any input, and $M^+(0)$ is that partition on S obtained by identifying all states having a common successor under any input.

   b) P is not an S.P. partition of M.

   c) The states of the contained permutation machines of both inputs must form appropriate P_1-type partitions.

   d) If the largest block of P contains $2^{j-2} < a \leq 2^{j-1}$ states, then in order to have a valid assignment there must be at least $j-3$ other binary S.P. partitions of M^+ satisfying a,b and c and not used for a different fsr assignment.

4. Using P_k obtained in 3 above, find $P_1, P_2, \ldots, P_{k-1}$ such that $P_1 P_2 \ldots P_k = 0$.

To illustrate, consider M5.6 given in Table 5.9.

Applying step 1 of the procedure, we obtain the machine $M5.6^+$ given in Table 5.10.

Table 5.9

Machine M5.6

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

Figure 5.10

Machine $M5.6^{+}$

	$0^{-1}1$	$1^{-1}0$	01^{-1}	10^{-1}
A	B	-	-	D
B	B	AB	B	B
C	C	C	-	F
D	E	-	AD	D
E	E	DE	E	E
F	F	F	CF	F
G	G	G	G	G

We find the following binary S.P. partitions on M^{+}:

$$P_1 = \{\overline{A,B,C,D,E,F,}; \overline{G}\}$$

$$P_2 = \{\overline{A,B,D,E}; \overline{C,F,G}\}$$

$$P_3 = \{\overline{A,B,D,E,G}; \overline{C,F}\}$$

From step 3 we find:

$$m^{+}(0) = \{\overline{A,D}; \overline{B}; \overline{C,F}; \overline{E}; \overline{G}\}$$

$$m^{*}(I) = \{\overline{A,B}; \overline{C}; \overline{D,E}; \overline{F}; \overline{G}\}$$

$$M^{+}(0) + m^{*}(I) = \{\overline{A,B,D,E}; \overline{C,F}; \overline{G}\}$$

It is easily verified that neither P_1, P_2 nor P_3 are S.P. partitions on M5.6. If we consider the contained permutation machines, we see that the cycle BCD under input 0 is identified by P_1, hence P_1 is rejected under 3 c). From step 4 we eliminate P_3, and find that P_2 is acceptable and produces the following assignment:

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

Nichols [65,66] considers the problem of finding multiple shift register realizations for sequential machines. Although his procedure is restricted to one-to-one assignments, it is the only existing algorithm that determines minimal assignments for multiple shift registers. Furthermore, it has been programmed on a digital computer. The algorithm, abreviated because the complete version [65] is too long, is:

1. Find P_c, P_r and $P_c \cdot P_r$, where P_r is that partition obtained by identifying all states having common predecessors, and P_c is that partition obtained by identifying all states having common successors. (Note that all states having no predecessors or no successors are eliminated.)
2. Find all possible binary partitions P, such that $P_c \leq P$, and $m(P) \leq 2^{n-1}$.

3. For each P, found in 2, find a partition P', where $P_r \leq P'$, and the blocks of P' are the successors of the blocks of P. Omit all P such that $m(P') \leq 2^{n-1}$.

4. Denote the partitions obtained in 2 and 3 by $\{P^{L+1}\}$.

5. Find $\{P^{L+1}\}$ from $\{P^L\}$ where $P^{L+1} = P_a^L \cdot P_b^L$ and the following conditions are satisfied:

   a) $P_c \leq P_a^L \in \{P^L\}$.

   b) Every block of P_a^L maps under I into, at most, one block of P_b^L, and every block of P_b^L is mapped into by, at most, one block of P_a^L.

   c) $m(P^{L+1}) \leq 2^{n-(L+1)}$.

   d) $\#(P^{L+1}) \geq 2^{L+1}$ and $P^{L+1} \notin \{P^k\}$ for $k \leq L$, and $\#(P)$ is the number of blocks in the partition P.

6. Repeat 5 until a $\{P^L\}$ is found in which there is no P_a^L such that $P_a^L \geq P_c$. For $L \geq 2$, $\{P^L\}$ will be the set of all acceptable shift register partitions (SRP's) of minimal length L.

7. The last step of the algorithm is, in effect, an exhaustive examination of those SRP's that can be used together for a minimal assignment.

As an aid to understanding, we now apply the algorithm to M5.6, given in Table 5.9, assuming that $n = 3$.

1. $P_c = \{\overline{A,D};\ \overline{B};\ \overline{C,F};\ \overline{E};\ \overline{G}\}$

   $P_r = \{\overline{A,B};\ \overline{C};\ \overline{D,E};\ \overline{F};\ \overline{G}\}$

   $P_c.P_r = 0.$

2. $P_1 = \{\overline{A,B,D};\ \overline{C,E,F,G}\}$

   $P_2 = \{\overline{A,B,D,E};\ \overline{C,F,G}\}$

   $P_3 = \{\overline{A,B,D,G};\ \overline{C,E,F}\}$

   $P_4 = \{\overline{A,C,D,F};\ \overline{B,E,G}\}$

   $P_5 = \{\overline{A,D,E};\ \overline{B,C,F,G}\}$

   $P_6 = \{\overline{A,D,E,G};\ \overline{B,C,F}\}$

   $P_7 = \{\overline{A,D,G};\ \overline{B,C,E,F}\}$

3. $P_1' = \{\overline{A,B,C};\ \overline{D,E,F,G}\}$

   $P_2' = \{\overline{A,B,C,G};\ \overline{D,E,F}\}$

   $P_3' = \{\overline{A,B,C,F};\ \overline{D,E,G}\}$

   $P_4' = \{\overline{A,B,D,E};\ \overline{C,F,G}\}$

   $P_5' = \{\overline{A,B,G};\ \overline{C,D,E,F}\}$

   $P_6' = \{\overline{A,B,F,G};\ \overline{C,D,E}\}$

   $P_7' = \{\overline{A,B,F};\ \overline{C,D,E,G}\}$

4. $\{P^1\} = \{P_1, P_2, \ldots, P_7, P_1', \ldots, P_7'\}.$

5. $P_1^2 = P_2.P_2' = \{\overline{A,B};\ \overline{C,G};\ \overline{D,E};\ \overline{F}\}$

   $P_2^2 = P_3.P_3' = \{\overline{A,B};\ \overline{C,F};\ \overline{D,G};\ \overline{E}\}$

   $P_3^2 = P_4.P_4' = \{\overline{A,D};\ \overline{B,E};\ \overline{C,F};\ \overline{G}\}$

   $P_4^2 = P_5.P_5' = \{\overline{A};\ \overline{B,G};\ \overline{C,F};\ \overline{D,E}\}$

   $P_5^2 = P_6.P_6' = \{\overline{A,G};\ \overline{B,F};\ \overline{C};\ \overline{D,E}\}$

   $P_6^2 = P_7.P_7' = \{\overline{A};\ \overline{B,F};\ \overline{C,E};\ \overline{D,G}\}$

$$\{P^2\} = \{P_1^2, P_2^2, \ldots, P_6^2\}$$
$$P_1^3 = P_3^2 \cdot P_1^2 = 0$$
$$\{P^3\} = \{P_1^3\}.$$

0 is an SRP of length 3, hence M5.6 is realizable as a single binary fsr of length 3.

Sheng [73,74] in a pair of reports also considers fsr's under the title SLYCD machines. The technique of interchanging next-states until the machine is definite is novel, but is shown not to hold for every case. Sheng also presents an algebraic technique for finding fsr assignments.

Martin and Susskind in Chapter IV of [80] consider the synthesis of sequential machines using two slightly different shift register models: bidirectional feedback shift registers and multishift feedback shift registers. Even though only one-to-one assignments are considered, the work involved in constructing the necessary partitions is prohibitive. Furthermore, the advantages gained by using these models may be disputed, since most of the reasons for studying shift registers in the first place are not necessarily applicable to the new models.

5.3 Single Binary fsr Realizations

In this section we consider the problem of realizing a given sequential machine as a single binary shift register.

5.3.1 One-to-One Assignments

We are concerned with the problem of obtaining an fsr assignment for a given sequential machine, or determining that such an assignment does not exist. Furthermore, the assignment must be one-to-one, i.e., state splitting is not allowed. Since the methods to be explained do not accommodate initial transient states, (i.e., states not appearing in any cycle), all tables will be assumed not to contain initial transient states. However, at the end of the section a method will be given for determining an assignment for initial transient states. Since we are restricted to one-to-one assignments, we use bp's and look for bp k-tuples.

If a sequential machine can be realized as a single binary fsr, then it must satisfy properties 1, 2, 3 and 4, and there must exist a bp n-tuple, where $n \geq [\log_2 q]$ and q is the number of internal states.

Definition 5.2: If $P = \{b_1; b_2; \ldots; b_r\}$ is an internal state partition and σb is the union of all successors of the states of b, then σP is that partition obtained from $\{\sigma b_1, \sigma b_2; \ldots; \sigma b_r\}$ by identifying any blocks σb_i and σb_j that are chain connected [41].

Lemma 5.1: For every nonautonomous single fsr of order n

$$0 < \sigma 0 < \sigma^2 0 < \ldots < \sigma^n 0 \leq S$$

where 0 is the trivial partition having no states identified.

Proof: $\sigma 0$ is clearly that partition that identifies only the pcp's. Since the fsr is nonautonomous, $0 < \sigma 0$. Suppose $i < n$ and $\sigma^i 0 = \sigma^{i+1} 0 \neq S$. Now each block of $\sigma^i 0$ can be uniquely represented by an n-tuple.

$$\varphi_1 \varphi_2 \cdots \varphi_i y_{i+1} y_{i+2} \cdots y_n.$$

where some of the φ's are don't cares and represent the variables which identify the states within the block. Take any pcp [A:B] and a state J such that $\sigma(J) = (AB)$. Now the i^{th} successors $\sigma^i(A)$ and $\sigma^i(B)$ of A and B must be identified by $\delta^{i+1}0$; however $\sigma^i(A)$ and $\sigma^i(B)$ differ in the $(i+1)^{st}$ variable, since A and B differ in the first variable. Hence $\sigma^i(A)$ and $\sigma^i(B)$ must be separated by $\sigma^i 0$. This is a contradiction, unless $i = n$.

The relation $0 < \sigma 0 < \sigma^2 0 < \ldots < \sigma^n 0 \leq S$ is illustrated by an example. Consider M5.7 given in Table 5.10.

Table 5.10

Machine M5.7

	0	1
A	B	B
B	D	C
C	E	F
D	G	G
E	H	I
F	J	J
G	K	K
H	A	B
I	D	D
J	F	E
K	I	H

$\sigma^0 = \{\overline{A,B};\ \overline{C,D};\ \overline{E,F};\ \overline{G};\ \overline{H,I};\ \overline{J};\ \overline{K}\}$,

$\sigma^2 0 = \{\overline{A,B,C,D};\ \overline{H,I,J};\ \overline{E,F,G};\ \overline{K}\}$,

$\sigma^3 0 = \{\overline{A,B,C,D,E,F,G};\ \overline{H,I,J,K}\}$,

$\sigma^4 0 = S$.

Definition 5.3: The composition tree of a sequential machine is a directed graph of $0, \sigma 0, \sigma^2 0, \ldots, \sigma^n 0$ where the nodes are the blocks and the branches indicate the transitions between the blocks of successive partitions.

To illustrate, the composition tree for M5.7 is shown in Fig. 5.3.

Lemma 5.2: A machine M is realizable (one-to-one assignment) as an fsr of order n if and only if there exists a binary partition $P \geq \sigma^{n-1} 0$ such that $\pi^{n-1}P, \pi^{n-2}P, \ldots, P$ is a bp n-tuple.

Proof: If $\pi^{n-1}P, \pi^{n-2}P, \ldots, P$ is a bp n-tuple then M is clearly realizable as an fsr of order n. If M is an fsr then there exist n binary partitions satisfying the requirements of Theorem 5.1. Now $\sigma 0, \sigma^2 0, \ldots, \sigma^{n-1} 0$ must identify all pcp's. Furthermore, $\sigma^{n-1} 0 \leq P$ is a binary partition, namely that one determined by y_n. Now $\pi^{n-1}P, \pi^{n-2}P, \pi^{n-3}P, \ldots, \pi P,$ must identify all scp's, and $\pi^{n-1}P, \pi^{n-2}P, \ldots, P$ is a bp n-tuple for M.

For example, consider M5.7 $P_1 = \sigma^3 0 = \{\overline{A,B,C,D,E,F,G}; \overline{H,I,J,K}\}$

$P_2 = \pi P_1 = \{\overline{A,B,C,D,H,I,J}; \overline{E,F,G,K}\}$

$P_3 = \pi^2 P_1 = \{\overline{A,B,E,G,H,I,K}; \overline{C,D,G,J}\}$

$P_4 = \pi^3 P_1 = \{\overline{A,C,E,F,H,J,K}; \overline{B,D,F,I}\}$

It is easily shown that $P_4, P_3, P_2,$ and P_1 are mutually consistent. A resulting fsr assignment for M5.7 is given in Table 5.11.

The next state equations for this assignment for M5.7 are therefore:

$$y_1' = y_2,$$

$$y_2' = y_3,$$

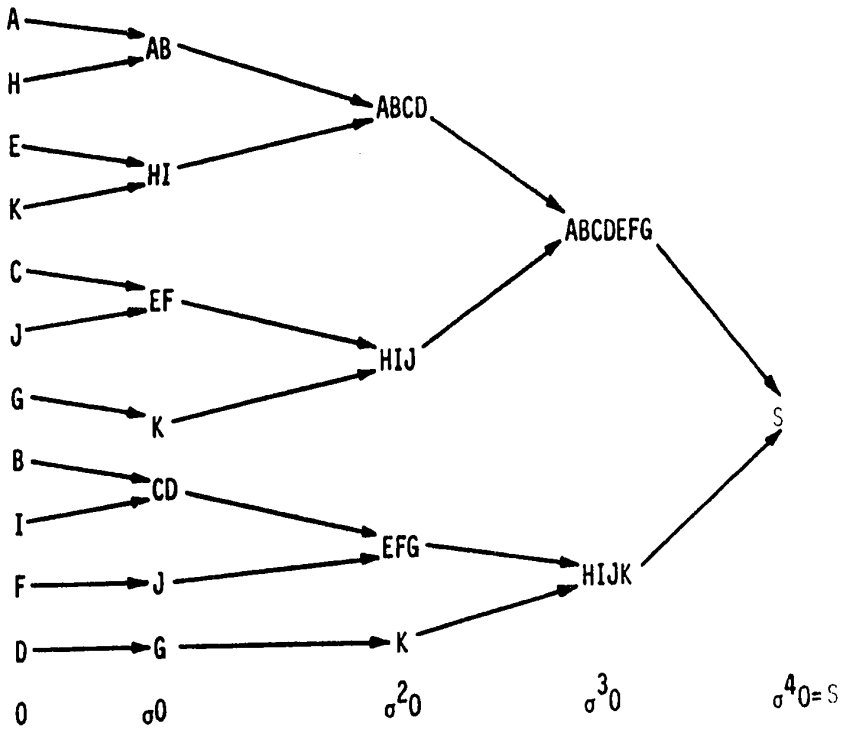

FIG. 5.3 - COMPOSITION TREE OF M5.7

$$y_3' = y_4,$$
$$y_4' = \bar{x}y_1\bar{y}_2 + \bar{y}_4y_1x + \bar{y}_3y_1\bar{x} + \bar{y}_2\bar{y}_1x + \bar{y}_3\bar{y}_1x + \bar{y}_1y_2y_3y_4\bar{x} + \bar{y}_2\bar{y}_3.$$

Table 5.11

Fsr Assignment for M5.7

	y_4	y_3	y_2	y_1
A	1	1	1	1
B	0	1	1	1
C	1	0	1	1
D	0	0	1	1
E	1	1	0	1
F	0	1	0	1
G	1	0	0	1
H	1	1	1	0
I	0	1	1	0
J	1	0	1	0
K	1	1	0	0

The procedure for all cases may not be quite as easy as that illustrated by M5.7.

For example, consider machine M5.8 given in Table 5.12.

Table 5.12

Machine M5.8

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

M5.8 has five distinct states, hence we initially consider a realization for $n = 3 = [\log_2 5]$. The composition tree for $n = 3$ of M5.8 is shown in Fig. 5.4.

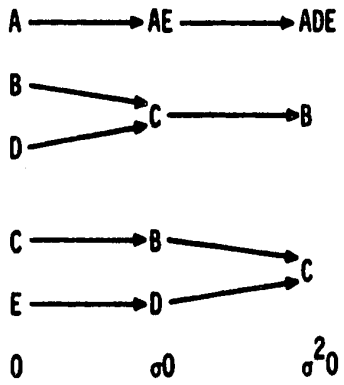

FIG. 5.4 - COMPOSITION TREE, FOR **n** = 3, OF M5.8

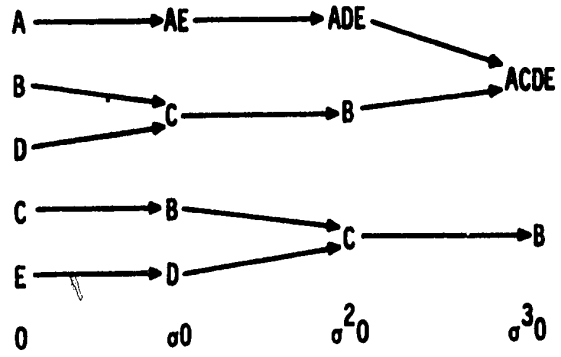

FIG. 5.5 - COMPOSITION TREE, FOR **n** = 4, OF M5.8

There are clearly three possibilities for $P \geq \sigma^2 0$, which are: P_1, P_2, P_3 given by b_1, b_2, b_3 = (A D E), (A B D E), (A C D E) respectively. It is easily seen that $\pi^2 P_1$, πP_1, P_1 are not mutually consistent, furthermore, neither are $\pi^2 P_2$, πP_2, P_2, or $\pi^2 P_3$, πP_3, P_3. We conclude therefore, that M5.8 is not realisable with a one-to-one assignment as an fsr of order 3. We can therefore consider a realization of order 4, and the composition tree of M5.8 for $n = 4$ is shown in Fig. 5.5. Since $\sigma^3 0$ is a bp, we have no choice for P, and it is easily shown that $\pi^3 P$, $\pi^2 P$, πP, P (given by (C,E), (B,D), (C), (B) respectively) is an fsr assignment for M5.8. An algorithm for determining one-to-one assignments for nonautonomous sequential machines without initial transient states is shown in Fig. 5.6.

We now turn to the problem of assignments for initial transient states. Consider a transient state S_t, of a machine M, such that $\sigma(S_t) = (S_a, S_b)$, where S_a and S_b are not necessarily distinct. S_a and S_b are obviously a pop, if they are distinct, and hence will be identified by $P_2, P_3, \ldots, P_n$ of the bp n-tuple for M. Now if we add S_t to those blocks b_i of P_i in the n-tuple for $i = 1, 2, \ldots, n-1$ such that S_a and S_b are contained in σb_i we will have a proper assignment. Furthermore, to add S_t to P_n we merely ensure that the assignment is valid.

To illustrate, consider M5.9 given in Table 5.13. We proceed exactly as if there were no transient states, but remember that $\sigma^1 0$ is not a partition in the required sense, but can be completed using the above procedure. As an example, the composition tree for M5.9 is shown

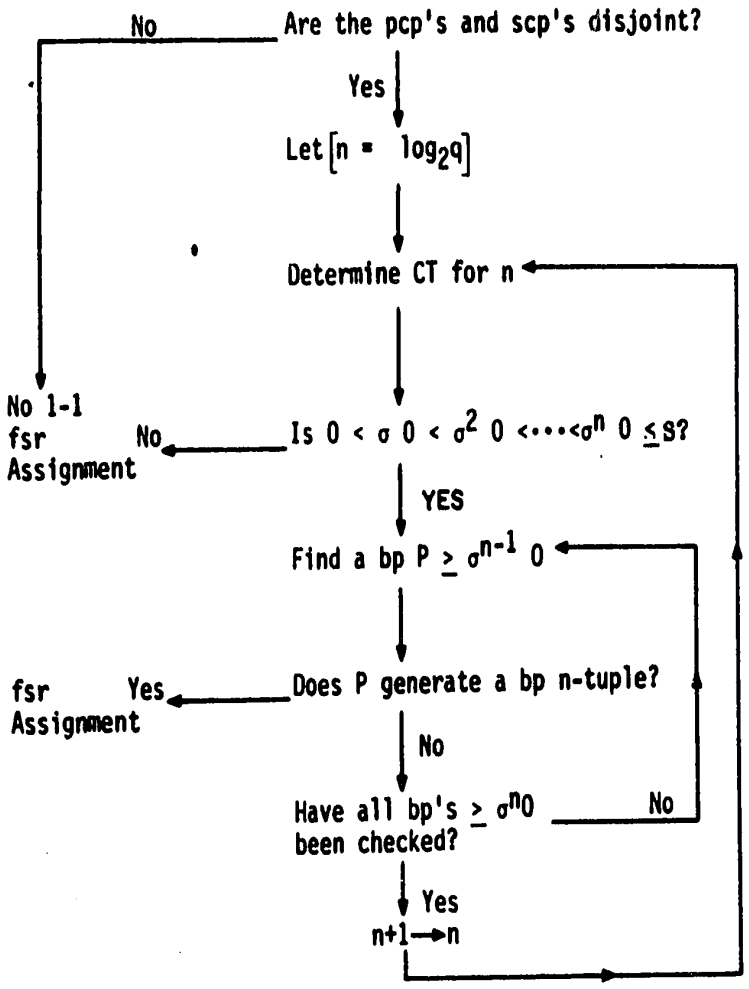

FIG. 5.6 - A ONE-TO-ONE SINGLE fsr ASSIGNMENT ALGORITHM FOR NONAUTONOMOUS MACHINES

in Fig. 5.7. The incomplete bp 3-tuple obtained from the composition tree is therefore: $\{b_1; \bar{b}_1\}$, $\{b_2; \bar{b}_2\}$, $\{b_3; \bar{b}_3\} = \{\overline{A,C,G,H}: \overline{B,D,E,F}\}$, $\{\overline{B,C,D}\ ;\ \overline{A,E,F,G}\}$, $\{\overline{A,E,D};\ \overline{B,F,G}\}$. Now H does not appear in P_2, and since D, a successor of H, appears in b_3, H must be contained in b_2. It should now be noted that C is also a successor of H, and therefore C should also be contained in b_3. H does not appear in P_3, and since $\#(b_3) = 4$, we obviously must add H to $\bar{b}_3$. The resulting complete n-tuple is: $\{\overline{A,C,G,H};\ \overline{B,D,E,F}\}$, $\{\overline{B,C,D,H};\ \overline{A,E,F,G}\}$, $\{\overline{A,C,D,E};\ \overline{B,F,G,H}\}$.

Table 5.13

Machine M5.9

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

5.3.2 Characterization and Many-to-One Assignments

We first characterize fsr's in terms of set systems. Obviously, if a sequential machine M is realizable as an fsr, then there exists a machine M' that is realizable as an fsr with a one-to-one assignment, and furthermore M is an image machine of M'.

Definition 5.4: A bsm n-tuple for a Machine M is an ordered n-tuple bsm's $Q_1, Q_2, \ldots, Q_n$ such that $\sigma Q_i = Q_{i+1}$ for $i = 1, 2, \ldots, n-1$ and $Q_1, Q_2, \ldots Q_n$ are mutually consistent.

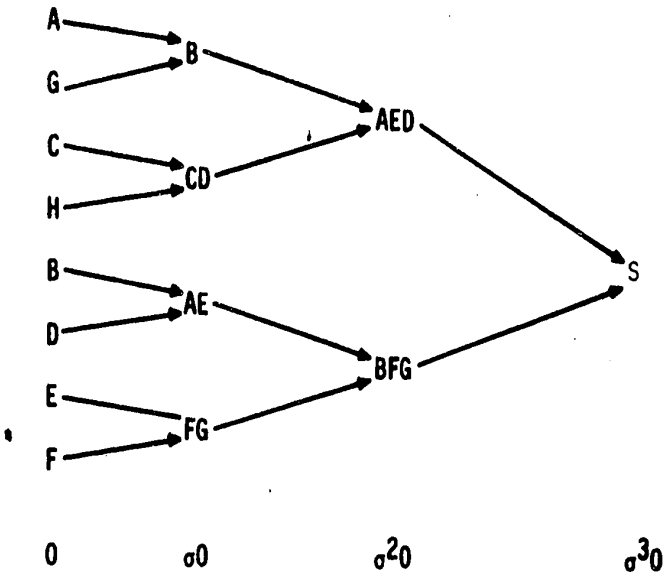

FIG. 5.7 - COMPOSITION TREE WITH TRANSIENT STATES FOR M5.9

Lemma 5.3: For every machine M realizable as a single binary fsr of order n, there exists a bsm n-tuple.

Proof: M is realizable as a single binary fsr, hence there exists a machine M', such that M' is realizable as an fsr with an one-to-one assignment and M is an image machine of M'. Therefore, there must exist a bp n-tuple for M'. If we apply the mapping, relating the states of M' to M to the bp n-tuple, we obviously get an n-tuple of bsm's. Furthermore, this is a bsm n-tuple since there exists a valid assignment for the included states.

As an example, consider M5.10 and M5.11 given in Tables 5.14 and 5.15.

Table 5.14

Machine M5.10

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

The bp 3-tuple for M5.10 is clearly $\{\overline{A,F,G,H};\ \overline{B,C,D,E}\}$, $\{\overline{A,B,C,G};\ \overline{D,E,F,H}\}$, $\{\overline{A,B,D,F};\ \overline{C,E,G,H}\}$. To illustrate bp (and bsm) n-tuples we will use the following notation:

(AFGH) ——→ (ABCG) ——→ (ABDF)

(BCDE) ——→ (DEFH) ——→ (CEGH).

The partitions are arranged in a column with the arrows indicating the transitions between blocks under the operator σ. If we apply the mapping given by the S.P. partition $P = \{\overline{A,F,G};\ \overline{B,C};\ \overline{D};\ \overline{E};\ \overline{H}\} = \{1;\ 2;\ 3;\ 4;\ 5\}$, we produce the following bsm 3-tuple for M5.11

(15) ——→ (12) ——→ (123)

(234) ——→ (1345) ——→ (1245)

Table 5.15

Machine M5.11

	0	1
1	1	2
2	1	3
3	5	4
4	4	5
5	2	1

Given an n-tuple of bsm's for a sequential machine M, how do we determine the machine M', if it exists, such that M' is realizable, with a one-to-one assignment, as a single binary fsr? There are two slightly different approaches to the problem:

1. For each state determined the possible n-tuples, or as an alternative, list states with n-tuples.
2. Starting with a state having the minimum number of possibilities, determine whether its successors are consistent,

i.e., have an admissible assignment. As we assign the successors, we then must check their successors, etc.

3. The process continues until either a consistent assignment is obtained or else it is determined that no consistent assignment is possible.

As an example, consider M5.12 given in Table 5.16, and the bam 4-tuple:

(AC) ——→(ABE) ——→ (ACDE) ——→(ABCDE)

(BDE) ——→(CDE) ——→ (ABCD) ——→(ABCE)

Table 5.16

Machine M5.12

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

The possible 4-tuples for the states are therefore (using 0 for top blocks and 1 for bottom blocks):

A: 0000
0001
0010
0011

B: 1010
1011

C: 0100
0101
0110
0111

D: 1100
1110

E: 1000
1001
1100
1101

Since B has only two possibilities and furthermore each possibility has the same successors, we shall start with B. Since $\sigma(B) = E$, we know that there must be an assignment for E of the form $\varphi 101$ where φ represents either 0 or 1. From the possibilities for E we see that there must exist an $E_1 = 1101$. Now $\sigma(E) = (C,D)$ therefore C and D must be of the form $\varphi 110$. Again, from the list of possibilities for C and D, we see that we must have $C_1 = 0110$ and $D_1 = 1110$. By continuing the process, we produce the following tabulation:

$E_1 = 1101$
$C_1 = 0110$
$D_1 = 1110$
$B_1 = 1011$
$A_1 = 0011$
$C_2 = 0111$
$A_2 = 0001$
$E_2 = 1001$
$A_3 = 0000$
$E_3 = 1000$
$D_2 = 1100$
$C_3 = 0100$

$A_4 = 0010$

$B_2 = 1010$

We therefore have M5.13 in Table 5.17 and it is easily seen that M5.12 is an image machine of M5.13 and M5.13 is realizable as an fsr with a one-to-one assignment.

Table 5.17

Machine M5.13

	0	1
A_1	A_2	E_2
A_2	A_3	E_3
A_3	A_3	E_3
A_4	A_2	E_2
B_1	E_1	E_1
B_2	E_1	E_1
C_1	B_1	A_1
C_2	B_1	A_1
C_3	B_2	A_4
D_1	C_2	C_2
D_2	C_1	C_1
E_1	D_1	C_1
E_2	D_2	C_3
E_3	D_2	C_3

As a further illustration, we may ask whether M5.12 is realizable as an fsr of order 3, using the bsm 3-tuple:

(AC) ——→ (ABE) ——→ (ACDE)

(BDE) ——→ (CDE) ——→ (ABCD)

Using the same technique we have

A: 000
001

B: 101

C: 010
011

D: 110
111

E: 100
110

We start with B since no choice exists. Therefore, we must have:

E_1 = 110

D_1 = 111

C_1 = 011

A_1 = 001

B = 101

A_2 = 000

E_2 = 100

D_2 = 110

C_2 = 010

At this point, we can stop since the process has produced both an E and a D with the 3-tuple 110. Therefore the given bsm 3-tuple does not admit a consistent assignment.

To illustrate the alternative approach, again consider M5.12 given in Table 5.16 and the bsm 3-tuple:

(ABE) ——→(ACDE)——→(ABCDE)

(CDE) ——→(ABCD) ——→(ABCE)

We have the following 3-tuples with their respective possible states:

000	AE
001	AE
010	AB
011	AB
100	CDE
101	CE
110	CD
111	C

Again we start with B since only two possibilities exist and they both have the same successors. We have:

$$E_1 = 101.$$

Now the possible successors of E_1 are 010 and 110, furthermore C must be assigned one and D the other; however, neither C and D are admissible states for 010. Hence, the given bsm 3-tuple does not admit a consistent assignment.

We now consider the problem of determining, for a given sequential machine, whether or not there exists a bsm n-tuple. While this problem is not completely solved, a number of techniques are available. We begin by showing that not every sequential machine is realizable as a single binary fsr.

Theorem 5.3: There exists a sequential machine M5.14 which is not realizable as a single binary fsr.

Proof: Let M5.14 be given by the flow table in Table 5.18.

Table 5.18

Machine M5.14

	0	1
A	B	C
B	C	A
C	A	B

Suppose that M5.14 is realizable as a single binary fsr M' of order n. Therefore there exists three nonempty sets of states {a}, {b} and {c} of M' that correspond (one-to-one) to A, B, and C respectively of M5.14. We note that the next-states of any state of M5.14 are distinct, i.e., under input 0 the next-state of A is B and under input 1 the next state of A is C, and furthermore B and C are distinct. Hence, if we start machine M' in any single state and apply the set of all input sequences of length n, in turn, we must exhaust all 2^n possible states of M'. Therefore, because M5.14 and hence M' are strongly connected, all 2^n possible states of M' must be contained in the sets {a}, {b} and {c}. However, the state 00...0 can have only two possible successors, namely 10...0 and itself, which means that one of the sets {a}, {b} and {c} of M' must have itself as a next state. This is an obvious contradiction and therefore M', and hence M5.14 is not realizable as a single binary fsr.

To obtain the bsm n-tuples, we proceed in a manner quite analogous to one-to-one assignments, but somewhat heuristically. It is clear, from the previous discussion, that for every sequential machine realizable as a single binary fsr of order n

$$0 \leq \sigma 0 \leq \sigma^2 0 \leq \ldots \leq \sigma^n 0 \leq S,$$

where $0, \sigma^i 0$ are set systems. We then find a bsm $Q \geq \sigma^n 0$ such that $\pi^n Q, \pi^{n-1} Q, \ldots, Q$ is a bsm n-tuple. We can also illustrate $0, \sigma 0, \ldots, \sigma^n 0$ by a composition tree; however, since the composition tree is not unique, we present a few general rules to assist in its preparation:

a) In the image machine M', there can exist only two distinct internal states S_i such that $S_i \in \sigma S_i$ [56]. Furthermore, both states must be separated by P_1 of the bp n-tuple for M'. This means that if there exist $k > 2$ states S_i such that $S_i \in \sigma S_i$ then 2 states must be separated by Q_1 of the bsm n-tuple and the remaining k-2 must be contained in both blocks of Q_1.

b) All pop's of M must be separated by Q_1.

c) Any state S_i of M such that $S_i \in \sigma^2 S_i$ and $S_i \notin \sigma S_i$ must be separated by Q_1 from the intermediate state S_j, where $S_j \in \sigma S_j$.

To illustrate the techniques we will consider a number of examples. First consider M5.11 given in Table 5.15 whose composition tree is straightforward and is given in Fig. 5.8. It is easily seen that the bsm 3-tuple

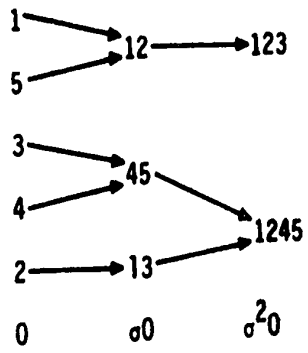

FIG. 5.8 - COMPOSITION TREE FOR M5.11

previously given for M5.11 can be obtained from Fig. 5.8. As another example, consider M5.15 given in Table 5.19.

Table 5.19

Machine M5.15

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

We obtain composition tree A given in Fig. 5.9. From Fig. 5.9 we note that A and C will not be separated by Q_1, furthermore A and B plus C and E must be separated by Q_1. Two alternatives exist and they are given in Fig. 5.10 and Fig. 5.11. It can be shown that a bsm 3-tuple cannot be obtained from Fig. 5.10 for M5.15. It can also be shown that we can obtain the following bsm 3-tuple for M5.15 from Fig. 5.11.

(ACD) ——→ (ABDE) ——→ (ABC)

(ABE) ——→ (ABC) ——→ (BCDE)

As a further method, it is possible, using an algebraic technique presented by Brzozowski [5] to find the set of all bsm's which separate the set of all pcp's. To illustrate, for M5.15 the pcp's are [D:E], [A:B] and [A:C]. From [5], the set of desired bsm's are given by:

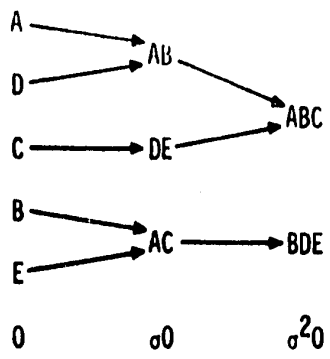

FIG. 5.9 - COMPOSITION TREE A FOR M5.15

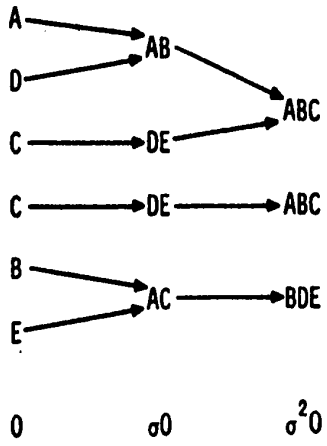

FIG. 5.10 - COMPOSITION TREE B FOR M5.15

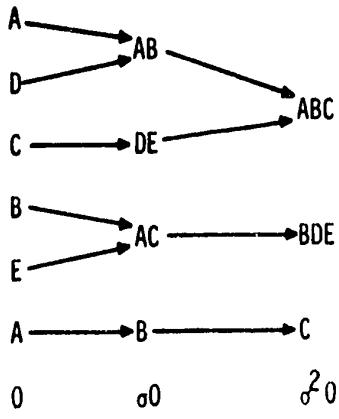

FIG. 5.11 - COMPOSITION TREE C FOR M5.15

(D/E) (A/B) (A/C)

= (AD/BE + AE/BD) (A/C)

= AD/BCE + ACD/ABE + AE/BCD + ACE/ABD.

5.3.3 Necessary Conditions for fsr Realizations

In this section we establish necessary and sufficient conditions for the realization of sequential machines as single binary fsr's, which enable us to discard, as not realizable, a number of flow tables. Since the one-to-one assignment case was easily solved, using partitions, we only consider many-to-one assignments.

Definition 5.5: A next-state distinct (nsd) machine M, is a sequential machine such that for distinct inputs I_a and I_b, and every state $S_j \in M$ we have $\sigma(S_j, I_a) \neq \sigma(S_j, I_b)$.

The following holds for any number of inputs, but for simplicity we will restrict ourselves to binary input machines.

Definition 5.6: The successor tree $\sum_n (S_i)$, of a sequential machine is an n level tree of successors of internal state S_i, where the 0th level consists of S_i, the 1st level contains $S_j \in \sigma(S_i)$, where $S_i \neq S_j$, and the ith level, i = 2, 3, ..., consists of all the successors of the (i-1)st level.

To illustrate, the successor tree $\sum_4$ (A) given in Fig. 5.12 is for machine M5.16 given in Table 5.20.

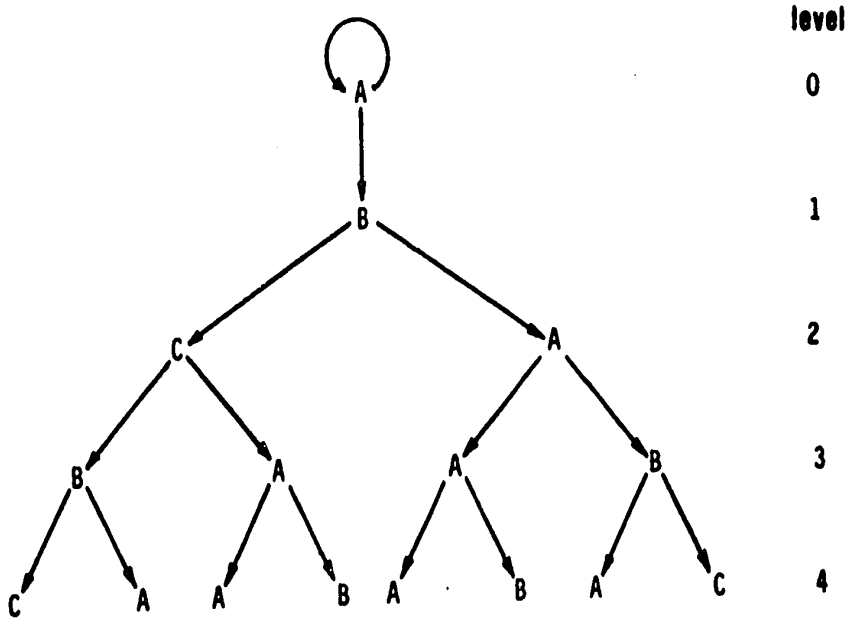

FIG. 5.12 - $\sum_4$ (A) FOR M5.16

Table 5.20

Machine M5.16

	0	1
A	A	B
B	C	A
C	B	A

Lemma 5.4: Given an nsd machine that is realizable as a single binary fsr of order n where the state S_i has the trivial assignment 00...0 or 11...1, then:

(a) $\sum_n (S_i)$ must contain 2^n distinct states.

(b) the (n + 1)st level of $\sum_{n+1}(S_i)$ must contain 2^n distinct states.

(c) the states in (a) and (b) are identical.

Proof:

(a) Let S_i be assigned the n-tuple 00...0 (a similar argument will hold for 11...1, and will not be repeated). Since we are restricted to nsd machines $\sigma(S_i)$ must consist of two (2) distinct states, namely 10...0 and 00...0. Furthermore, 00...0 is the 0th level and 10...0 is the 1st level. The second level must also consist of 010...0 and 110...0. If the kth level consists of k states, the $(k + 1)^{st}$ level must contain 2k states, for k = 1, 2, ... Since the 1st level contains 1 state, the kth level must contain 2^{k-1} states. Since the first level consists of

the n-tuple 10...0, the kth level, $k = 1, 2, \ldots, n$ must consist of n-tuples of the form $x_k x_{k-1} \cdots x_1 10 \ldots 0$, where $x_i = 0,1$. Therefore, any state in the kth level, $k = 1, 2, \ldots, n$, cannot have the same n-tuple as a state in the jth, for $j < k$, level. Hence $\sum_n(S_i)$ must contain 2^n distinct states.

(b) the nth level of $\sum_n(S_i)$ consists of 2^{n-1} distinct states having n-tuples of the form $x_{n-1} x_{n-2} \cdots x_1 1$, therefore the $(n + 1)$st level of $\sum_{n+1}(S_i)$ must consist of 2^n distinct states.

(c) Since an fsr of order n can have only 2^n distinct internal states, the states in (a) and (b) must be identical.

We now consider the consequences of this lemma. Consider $\sum_4(A)$ for M5.16 given in Fig. 5.12. If M5.16 is realizable as an fsr of order 3, then the states A appearing in the 0th, 2nd and 3rd level must be distinct. This means there must exist 4 distinct states A, 3 distinct states B and 1 distinct state C in the fsr assignment of M5.16. Now, from part (b) we see that the 4th level of $\sum_4(A)$ consists of 4 distinct states A, 2 distinct states B and 2 distinct states C. We therefore conclude that $\sum_3(A)$ is not identical to the 4th level of $\sum_4(A)$ and part (c) of the lemma is not satisfied. Hence M5.16 is not realizable as an fsr of order 3.

Definition 5.7: The canonical successor tree for an fsr of order n is given by Fig. 5.13 where the ith node represents the state having the binary n-tuple $y_n y_{n-1} \cdots y_1$ whose numerical value is i.

Note that Fig. 5.13 is essentially an unfolded version of Good's diagram [65], where each state appears twice, once in levels 0 to n and again in level n + 1.

We now state a theorem relating the canonical successor tree to the states of any machine realizable as a single binary fsr.

Theorem 5.4: A sequential machine M is realizable as a single binary fsr of order n, if and only if, there exists a mapping that preserves the successor relation from a subset of the nodes of the canonical successor tree onto the states of M.

Proof: The proof of the theorem is obvious. $\bigvee_n (S_i)$ is a graphical representation of the states of an fsr, but we can also have an analytical representation. Let T (the transition matrix) be a (q x q) square matrix of a machine representing the transitions, (where q is the number of distinct internal states in the flow table), $t_{ij} = 1$ if state S_i has a successor state S_j otherwise 0. For example, T for M5.16 is given by

$$T = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

The level vector of a successor tree is a row vector $V_k = (n_1 n_2 \cdots n_q)$ where n_i is a non-negative integer representing the number of occurrences of state S_i in the kth level of the tree.

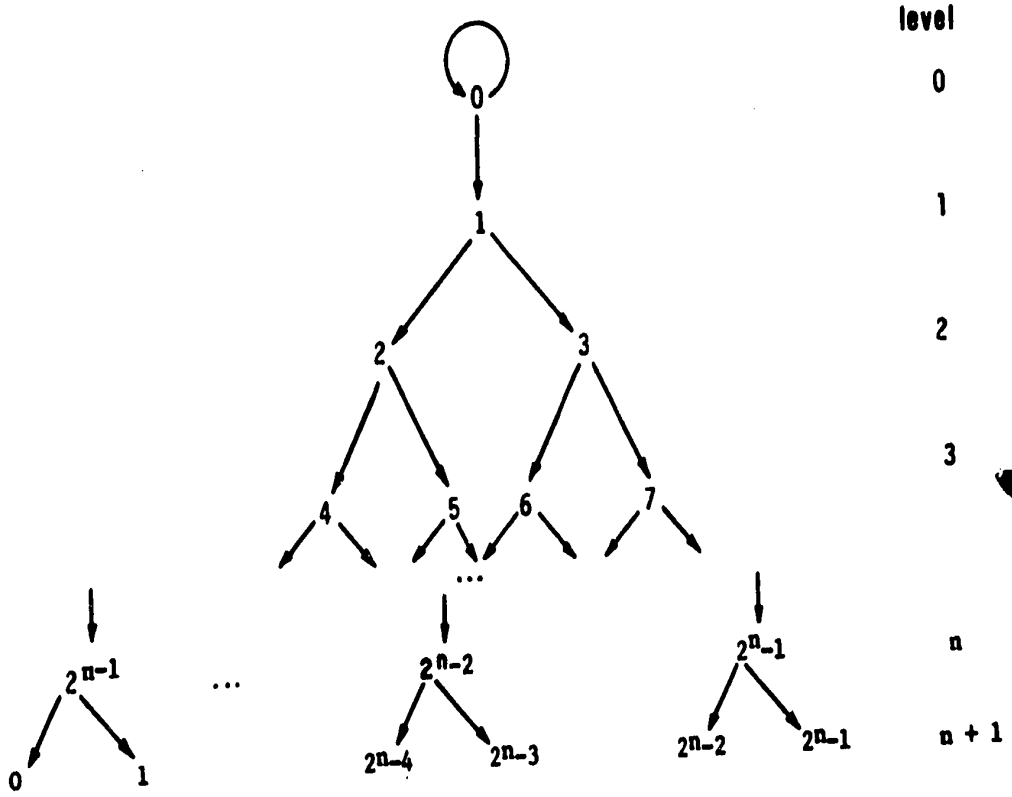

FIG. 5.13 - CANONICAL SUCCESSOR TREE

Theorem 5.5: If M is an nsd machine that is realizable as a single binary fsr then:

1. S, the set of internal states of M, contains all n-tuples.
2. There exists at least one $t_{ii} = 1$, for $i = 1, 2, \ldots, q$.
3. M is strongly connected which implies that there exists at least one 1 in each column.
4. $V_0 T^n (T - 2I) = 0$, for some $V_0 = (0\ldots010\ldots0)$, i.e., the ith column = 1, $i = 1, 2, \ldots, q$ and $t_{ii} = 1$.
5. T is singular.

Proof:

1. Starting at any state $S_i \in S$ and applying any sequence of length n, we arrive at some state $S_i \in S$. Furthermore, because every state has two distinct next states, for every distinct input sequence of length $\leq n$, we arrive at a distinct state. Now, since there are 2^n distinct input n-tuples of length n, S must contain 2^n distinct states.
2. The n-tuples 00...0 and 11...1 can have only two possible successors, one of which must be themselves, hence since S consists of all states and all states have two distinct successors, there must exist a state of M which is its own successor, i.e., $t_{ii} = 1$.
3. From 1. we note that from any state, we can get to any other state, hence M is strongly connected. Now, if M is strongly connected, then each state must have at least one

predecessor which means that there is at least one 1 in each column.

4. $V_0 = (0\ldots010\ldots0)$, i.e., a row vector with a single 1 in the ith column and 0's everywhere else.

$$V_1 = V_0(T - I)$$
$$V_2 = V_1T = V_0(T^2 - T)$$
$$V_3 = V_2T = V_0(T^3 - T^2)$$
$$\ldots$$
$$V_n = V_0(T^n - T^{n-1})$$

From Lemma 5.4:

$$V_{n+1} = V_0 + V_1 + V_2 + \ldots + V_n$$
$$= V_0 + V_0(T - 1) + V_0(T^2 - T) + \ldots + V_0(T^n - T^{n-1})$$
$$= V_0T^n$$

$$V_{n+1} = V_0(T^{n+1} - T^n)$$
$$V_0T^n = V_0(T^{n+1} - T^n)$$
$$V_0(T^{n+1} - 2T^n) = 0$$
$$V_0T^n(T - 2I) = 0$$

5. $V_0(T - 2I)T^n = 0$

Hence, $V_0T^{n+1} = 2V_0T^n$

If T is nonsingular T^{-1} exists, and is unique [2].

$$V_0T^{n+1}(T^{-1})^n = 2V_0T^n(T^{-1})^n$$
$$V_0T = 2V_0$$

Since V_0 has a single 1 in the ith position.

V_0T = the ith row of $T \neq 2V_0$.

Hence, T is singular, which completes the proof of the theorem. To illustrate the application of the theorem, consider M5.17 given in Teble 5.21.

Table 5.21

Machine M5.17

	0	1
A	A	B
B	C	A
C	B	A

It is easily seen that M5.17 is an nsd machine, and from the matrix T below, parts 2, 3 and 5 of the theorem are satisfied.

$$T = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

The only $t_{ii} = 1$, is for $i = 1$, hence V_0 must satisfy part 4. Furthermore,

$$T - 2I = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 1 & 1 & -2 \end{pmatrix}$$

Now, in order to satisfy part 4 of the theorem, and if we let

$V_0T^n = (a_1\ a_2\ a_3)$ then we have:

$$a_1 = a_2 + a_3$$
$$2a_2 = a_1 + a_3$$
$$2a_3 = a_2$$

V_0T^n is then of the form $k(3\ 2\ 1)$ where $k > 0$. We will use the following notation:

$$T^n = \begin{pmatrix} t_{11}^n & t_{12}^n & t_{13}^n \\ t_{21}^n & t_{22}^n & t_{23}^n \\ t_{31}^n & t_{32}^n & t_{33}^n \end{pmatrix}$$

hence,

$$t_{12}^n = t_{32}^n = t_{12}^{n-1} + t_{22}^{n-1}$$

$$t_{22}^n = t_{12}^n + t_{32}^n = 2t_{12}^n$$

Now t_{22}^n is even for $n = 1$ and will always be even for $n > 1$. Furthermore, t_{12}^n is odd for $n = 1$, and will always be odd for $n > 1$, since it is the sum of an even and an odd number.

$V_0T^n = (t_{11}^n\ t_{12}^n\ t_{13}^n)$, and since t_{12}^n is odd for $n > 0$, V_0T^n can never be of the form $k(3\ 2\ 1)$. Hence, part 4 of the theorem can never be satisfied and we conclude that M5.17 is not realizable as a single binary fsr.

5.3.4 Non-NSD Machines

The preceeding section was restricted to machines whose internal states have two distinct successors; however, not all sequential machines satisfy this requirement. We therefore consider this class of machine (i.e., non-nsd machines). In this case the successor tree merely indicates the possible successors for each state and does not require that both be implemented (i.e., that the actual transitions exist). Furthermore, it is easily shown that all n-tuples do not necessarily appear as states in the assignment.

Lemma 5.5: For every sequential machine M, realizable as a single binary fsr, of order n, there exists a machine M+, such that S+ contains all 2^n binary n-tuples and M is an image machine of M^+.

Proof: M is realizable as a single binary fsr, hence there exists a machine M', such that M' is realizable as an fsr or order n with a one-to-one assignment. If S' contains all n-tuples then the lemma is proved, and M+ = M'. Suppose that S' does not contain all n-tuples. We now construct M+ from M',by assigning to each n-tuple $Y_k \notin S'$ that state $S_i \in S'$,such that Y_i (the n-tuple assigned to S_i) is the n-tuple that agrees with Y_k in the longest initial portion of the n-tuple. In other words, Y_i is the n-tuple in S' such that Y_k and Y_i have the longest initial sequence of identical bits. We now have to show that Y_k and Y_i are indeed equivalent states. Suppose that Y_k and Y_i agree in the first h bits of the n-tuples. Now Y_k and Y_i have identical possible next states after any input sequence of length n-h. Hence Y_k and Y_i are not equivalent states if and only if there exists an input sequence X_α of length $x < n-h$ such that X_α when applied to Y_k and Y_i, results in two different states $Y_{xk} \neq Y_{xi}$. Suppose $Y_{xk} \in S'$,therefore there exists a state Y_β, namely the xth predecessor of Y_{xk} such that Y_k and Y_β agree in the first n-x bits. This is a contradiction, since $n-x < h$. Suppose $Y_{xk} \notin S'$,then Y_{xk} is not equivalent to Y_{xi} if and only if there exists a state Y_β such that Y_{xk} and Y_β agree in the first $\geq h + x$ bits. This implies, however, that there exists a state $S \in S'$ such that Y_k and S agree

in the first $g-x > n$ bits. Again this is a contradiction, and the lemma is proved.

We can now consider the consequences of this lemma in determining whether or not there exist fsr realizations for non-nsd machines. To illustrate the approach, consider machine M5.18 given in Table 5.22.

Table 5.22

Machine M5.18

	0	1
A	C	B
B	C	A
C	B	B

As a consequence of the lemma, we know that if M5.18 is fsr realizable, then a realization exists where one of the states A, B, or C is assigned the trivial n-tuple (00...0). Clearly, the only state of M5.18 which could possibly be assigned this n-tuple is state C. Consider a realization for $n = 3$, and hence $\sum_4(C)$ for M5.18 given in Fig. 5.14. We note that state C also appears in the 2^{nd} and 3^{rd} levels of $\sum_4(C)$. Furthermore, the C's in the 0th to 3^{rd} levels are distinct. From the lemma we know that two distinct possible successors of each state exist. We have indicated the other possible transitions in the tree by dashed lines and a circle indicating the state. In Fig. 5.14 there exists only one unassigned n-tuple in the tree $\sum_3(C)$, and there exists two distinct states A in the 4^{th} level of $\sum_4(C)$ and only one state A in $\sum_3(C)$. Hence, from lemma 5.4, if M5.18 is realizable as an fsr of length 3, we know that the

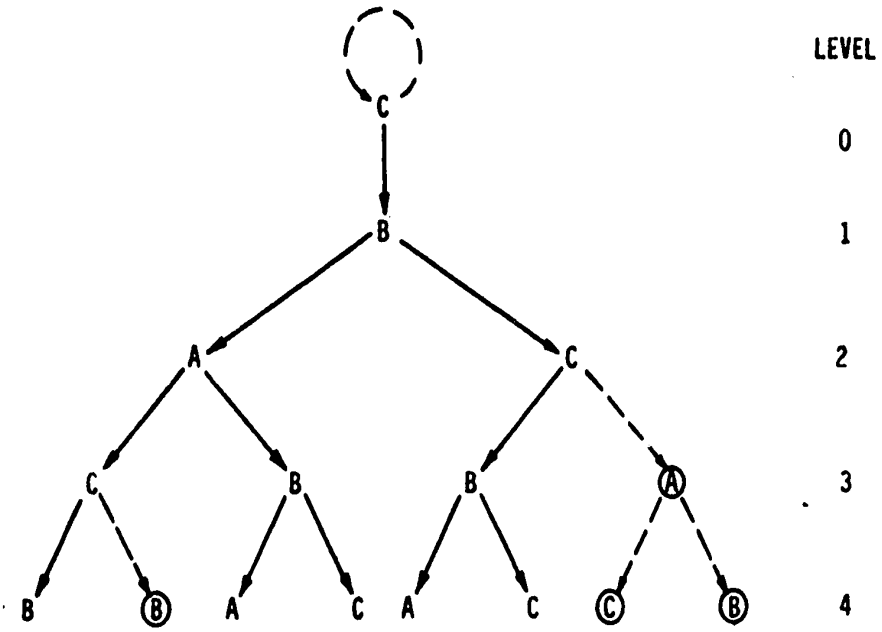

FIG. 5.14 - SUCCESSOR TREE FOR M5.18

single unassigned state must be A. By continuing the argument, we complete the tree, as shown in Fig. 5.14. Now, lemmas 5.4 and 5.5 are satisfied, and we need only satisfy Theorem 5.4. It is easily shown that Theorem 5.4 cannot be satisfied, hence, we can conclude that M5.18 is not realizable by an fsr of length 3. We can continue the argument for fsr's of length 4, 5, ...; however, for the case of non-nsd machines we do not obtain an easy method for eliminating machines as non-fsr realizable.

5.4 Multiple Binary fsr Assignments

In this section we extend some of the concepts developed for single binary fsr's to multiple fsr's. Here the object is to realize a given machine with as few shift registers as possible. We naturally assume that the machine is to be realized with at least two binary fsr's. In other words, either the machine is not realizable as a single binary fsr or else we have found a single binary fsr realization and would like to compare it with a multiple fsr realization.

5.4.1 One-to-One Assignments

From properties 1 and 2, given in Section 5.1, we see that if m is the maximum number of predecessors or successors for any state of a given machine, then the multiplicity of the machine for an fsr realization is $p \geq [\log_2 m]$.

Lemma 5.6: For every multiple fsr of order $m = \max(n_1, n_2, .., n_p)$ $0 < \sigma 0 < \sigma^2 0 < \ldots < \sigma^m 0 \leq I$.

Proof: m is the length of the longest shift register, hence the lemma follows directly from Lemma 5.1.

Lemma 5.7: A machine M is realizable (one-to-one assignment) as a multiple fsr of multiplicity p and orders $n_1, n_2, \ldots, n_p$ respectively, if and only if there exist binary partitions P_i, for $i = 1,2,\ldots,p$ such that $\sigma^{n_i-1} 0 \leq P_i$ and

$$\pi^{n_i-1} P_i, \pi^{n_i-2} P_i, \ldots, P_i \text{ is a bp } n_i\text{-tuple.}$$

Proof: The proof of the lemma follows directly from the proof of Lemma 5.2.

The problem of determining multiple fsr assignments is more complicated than that for single fsr assignments; however, we can use the same method by considering the multiple fsr as several single fsr's. To illustrate, consider machine M5.19 given in Table 5.23, and its composition tree in Fig. 5.15.

M5.19 clearly has a multiplicity ≥ 2. Hence, if we initially consider a minimal realization we are restricted to either a) one fsr of order 3 and the second of order 1 or b) two fsr's of order 2. Upon checking the composition tree for M5.19 we find that there does not exist a bp $P \geq \sigma^2 0$ such that $\pi^2 P, \pi P, P$ are mutually consistent for $n = 4$. We are therefore limited to considering two fsr's, each of order 2, for a minimal realization. It can be shown, that there does not exist a valid

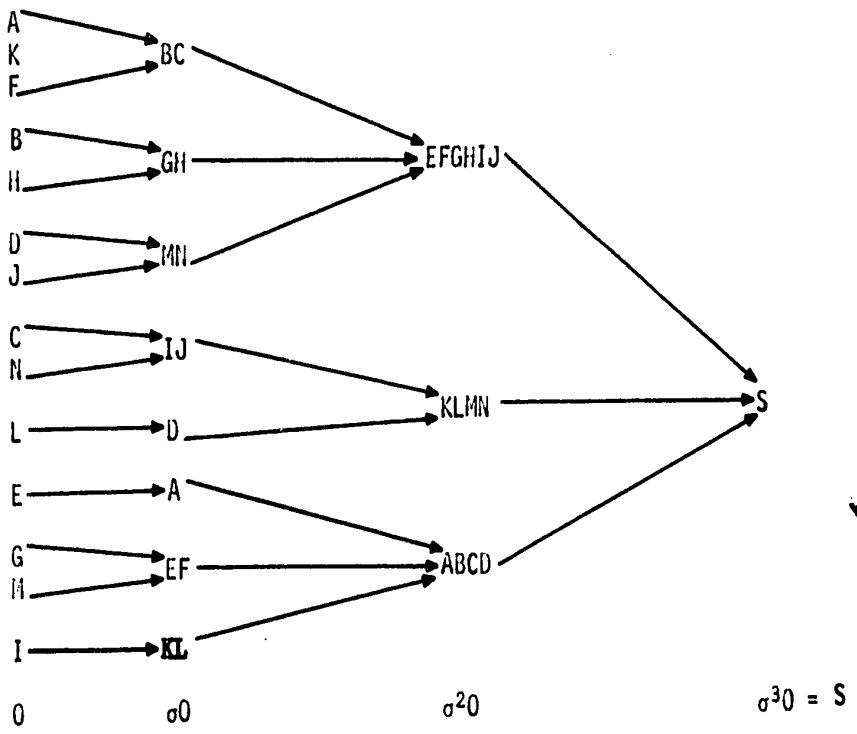

FIG. 5.15 COMPOSITION TREE FOR M 5.19

one-to-one assignment for M5.19 of 2 fsr's of order 2. We therefore must consider a realization for $n = 5$. From the composition tree we note that we cannot realize M5.19 as 2 fsr's of order 4 and 1, we therefore consider orders 3 and 2. It is easily seen that the bp 3-tuple given by

$$b_1, b_2, b_3 = (ABDFHJK), (BCGHMN), (KPGHLJ),$$

is mutually consistent for $n = 5$. Therefore, in order to complete the realization we must find a bp 2-tuple such that the 5 bp's are mutually consistent. It can be shown that one bp 2-tuple satisfying this requirement is given by $b_4, b_5 = (ABCEFHKN), (ABCGHLJ)$. A resulting assignment is given below:

Table 5.23

Machine M5.19

	0	1
A	C	B
B	H	G
C	I	J
D	N	M
E	A	A
F	B	C
G	E	F
H	G	H
I	L	K
J	N	N
K	C	B
L	D	D
M	E	F
N	J	I

	y_1	y_2	y_3	y_4	y_5
A	0	1	1	0	0
B	0	0	1	0	0
C	1	0	1	0	0
D	0	1	1	1	1
E	1	1	0	0	1
F	0	1	0	0	1
G	1	0	0	1	0
H	0	0	0	0	0
I	1	1	0	1	0
J	0	1	0	1	0
K	0	1	1	0	1
L	1	1	1	1	1
M	1	0	1	1	1
N	1	0	1	0	1

5.5 Logical Design Using Shift Registers

It is apparent, from the preceeding sections, that a considerable amount of effort has been expended, with varying degrees of success, on the problem of finding shift register assignments for sequential machines. There is, however, an inconsistency between the theoretical results and what happens in actual practice. The purpose of this section is to point out this inconsistency and to provide an actual design technique that will overcome the problem.

5.5.1 The Inconsistency, and a New Model

A synchronous shift register is generally considered to be in the form shown in Fig. 5.1, where it is assumed that the delay elements

are synchronized by a common clock signal. If it is assumed that Fig. 5.1 is the form of the realization under consideration, then it can be shown, using the methods given in Section 5.3.3, that the sequential machine M5.20, given in Table 5.24, is not realizable as a single binary fsr.

Table 5.24

Machine M5.20

	0	1
A	A	B
B	B	C
C	C	A

Under certain assumptions, however, it is posible to realize M5.20 using a single binary shift register of length 2. For example, the circuit shown in Fig. 5.16 is a realization of M5.20 using a shift register of length 2.

In this realization the feedback function is given by

$$f = y_1 \oplus y_2,$$

where $\oplus$ represents the exclusive or operation and the shift signal is given by

$$S = X.(\text{clock}).$$

This realization was brought about by gating the clock signal with the input signal to provide the common shift signal to the memory elements. This may, or may not, be possible in every case; however, if it is possible we must consider a new model, and its attendant consequences, during the pro-

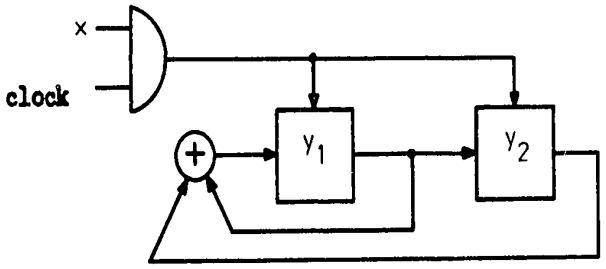

FIG. 5.16 - AN fsr REALIZATION OF M 5.20

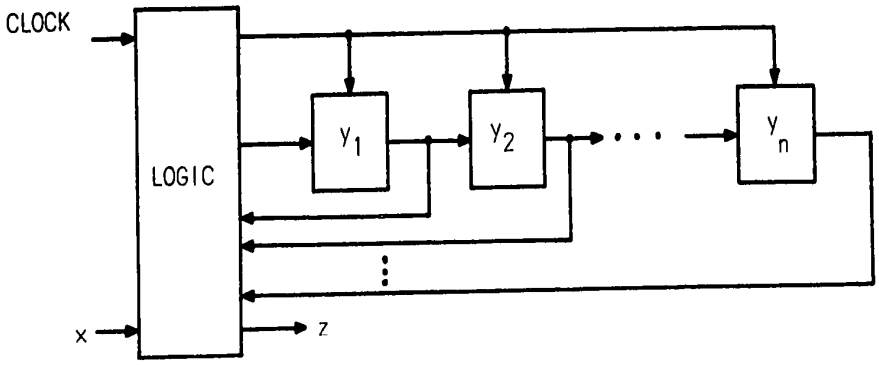

FIG. 5.17 - NEW SHIFT REGISTER MODEL

cess of logical design. This more realistic model, which includes the ability to gate the shift signal is shown in Fig. 5.17.

This improved shift register model now increases our logical design capability using shift registers. The question now occurs, by how much? Note that in an fsr with an ungated shift signal (i.e., Fig. 5.1) at each clock signal the contents of the register are shifted one place to the right, and this occurs irrespective of the contents of the register. Now, by providing the ability to gate the clock signal, the shift signal can be interrupted at any time, depending on the state of the register and the input. Furthermore, when the shift signal is interrupted, the shift register does nothing, i.e., each memory element stays in its original state. Since the shift signal is common to all memory elements, whenever the signal is interrupted the state of the register does not change. Hence, any state of the shift register can be succeeded by itself under any input. We next consider this increased capability with respect to a logical design procedure.

5.5.2 Logical Design Procedure

To determine an fsr realization, using the new model, for a sequential machine, given by its flow table, we use the following procedure:

1. Replace, in the table, each next state, which is the same as the present state, by a don't care.

2. Find an fsr realization for the incompletely specified machine, resulting from 1, using the methods given previously in this chapter.
3. Add the gating necessary to interrupt the shift signal for all those states that were removed from the table in 1.

To illustrate the procedure, we will apply it to M5.21 given in Table 5.25.

Table 5.25

Machine M5.21

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

The first step of the procedure results in the incompletely specified machine, denoted M5.21', given in Table 5.26.

Table 5.26

Machine M5.21'

	0	1
A	B	-
B	A	C
C	-	D
D	-	E
E	A	-

Now, using the methods previously given in the chapter we produce the following assignment for M5.21'.

A: 0 0 1
   0 1 0

B: 1 0 0
   1 0 1

C: 1 1 0

D: 1 1 1

E: 0 1 1

The feedback function for this assignment is therefore

$$y_1' = y_2\bar{y}_3 + \bar{y}_2x + \bar{y}_1\bar{y}_2.$$

We now produce the function for the shift signal sh which must produce an output 1 for every specified entry in the next state table and a 0 everywhere else. The truth table for the shift signal is given by:

		∅ B A C	A B E D
	y_1	0 1 0 1	0 1 0 1
	y_2	0 0 1 1	0 0 1 1
	y_3	0 0 0 0	1 1 1 1
X	0	∅ 1 1 0	1 1 1 0
	1	∅ 1 0 1	0 1 0 1

It is easily shown that

$$Sh = \bar{y}_1\bar{x} + \bar{y}_2\bar{x} + y_1x.$$

In conclusion, an inadequacy in the present techniques of logical design of using synchronous shift registers has been presented. The complete capability of synchronous shift registers are examined and a logical design technique is presented for using this inherent capability.

5.6 Conclusion

In summary, we have: For autonomous machines, Haring's algorithm [37] which determines a single binary shift register realization for every autonomous sequential machine. For nonautonomous machines the results are varied. Liu [53] has a method for multi-level shift registers, that he attempts to extend [55,79] , with very little success, to variable length shift registers and also to binary shift registers. For single binary shift register the algorithms of Haring [37], Nichols [65,66] and Section 5.3.1 all produce one-to-one assignments, if they exist. For multiple binary registers Nichols [65] has the only complete algorithm, again for one-to-one assignments, but the methods presented in Section 5.4.1 could be used to effectively simplify Nichols algorithm. Many-to-one assignments are treated by Liu [79] , Haring [37] and in Section 5.3.2, but none of these methods produce anything like an effective procedure. The results of Section 5.3.3 do permit, however, the elimination of a large class of machines as being not realizable as a single binary far. Nothing has been produced on minimal shift register realizations (many-to-one assignments). A generalized logical design procedure is presented in Section 5.5.

Chapter 6

CONCLUSION AND FURTHER PROBLEMS

In conclusion, the structure of sequential machines has been studied from several different points of view. Initially, a brief survey of structure theory was presented. State assignments for combinational circuits were examined, with the object of applying the results to the problem of determining a state assignment producing the minimum output logic. It was then shown how the same methods could be applied to the problem of finding an assignment minimizing the combinational logic realizing the next-state functions. Linear sequential machines were considered and methods were presented for determining whether or not a machine is realizable as a linear machine. Feedback shift registers were also studied, and methods were presented for determining shift register realizations for sequential machines.

All of the problems that were considered in this thesis have not been completely solved; however, these and other synthesis problems are left as problems for further research:

1. Most of the work on the synthesis of digital systems is concerned with the process of converting the mathematical specification of a machine, i.e., a flow table, into a logic circuit realizing the machine. Very little (see Armstrong [1]) has been done on the problem of converting the initial word statement into the mathematical specification. A solution to this problem, however, would provide a giant step forward in the problem of synthesizing digital systems. Furthermore, if the approach was programmed on a digital computer, then the application of such a result could revolutionize logical design.

2. The problem of the relationship between assignments and combinational circuits is still not fully understood. A formal treatment of adjacency techniques would seem to be a good way to start, since adjacency has been shown to yield reasonable results for some cases. In addition, the approach presented in Chapter 3 could be further amplified and possibly related adjacency methods, since the two clearly have something in common.

3. With respect to linear machines, the last remaining synthesis problem is to determine whether or not a machine

which satisfies parts a) and b) of Corollary 4.1 also satisfies part c) of the same theorem.

4. For shift registers, many problems still remain. The most obvious one is many-to-one assignments. No satisfactory method exists for determining the minimum shift register realization for a given machine.

5. Modular synthesis is another area where much remains to be done. In other words, given a set of modules, is it possible to determine the simplest realization using the given modules.

6. Many-to-one assignments have not been exploited to any degree with respect to the minimization of the combinational logic in a sequential machine. It would seem that this could also prove to be a fruitful area to study.

BIBLIOGRAPHY

1. Armstrong, D.B. Sequential Circuits - Part A: Structural Properties, in Weinberg: Progress in Circuit Theory. IEEE Trans., Vol. CT-11, pp. 25-27. March, 1964.

2. Birkhoff, G. and S. MacLane. A Survey of Modern Algebra. The Macmillan Co., New York, N.Y. 1953.

3. Brzozowski, J.A. Review of [53]. IEEE Trans. Vol. EC-13, p. 162. April, 1964.

4. Brzozowski, J.A. An Essay on Feedback. Electrical Engineering Dept., University of Ottawa. Technical Report No. 65-2. March, 1965.

5. Brzozowski, J.A. On Single Loop Realizations of Automata. 1965 IEEE Conference Record on S.C.T.L.D.*, pp. 84-93. October, 1965.

6. Brzozowski, J.A. Synthesis of Sequential Machines. Presented at Conference on Systems and Computer Science. The University of Western Ontario, London. September, 1965.

7. Brzozowski, J.A. and W.A. Davis. On the Linearity of Autonomous Sequential Machines. IEEE Trans. Vol. EC-13, pp. 673-679. December, 1964.

7a. Brzozowski, J.A. and P. Hawtrey. Logic-Free Realization of Sequential Machines. Electrical Engineering Department, University of Ottawa. Technical Report No. 66-9. August, 1966.

8. Crowell, R.H. Graphs of Linear Transformations Over Finite Fields. Jour. Soc. Indust. Appl. Math., Vol. 10, pp. 103-112. March, 1962.

* Switching Circuit Theory and Logical Design.

9. Crowley, T.H. Recent Developments in the State Assignment Problem. Presented at the 4th Annual Symposium on Switching Circuit Theory and Logical Design. Chicago, Ill. October, 1963.

10. Cohn, M. A Theorem on Linear Automata. IEEE Trans., Vol. EC-13, p. 52. February, 1964.

11. Cohn, M. and S. Even. Identification and Minimization of Linear Machines. IEEE Trans., Vol. EC-14, pp. 367-376. June, 1965.

12. Davis, W.A. On the Linearity of Sequential Machines. M.Sc. Thesis, E.E. Dept. University of Ottawa. September, 1963.

13. Davis, W.A. On Shift Register Realizations for Sequential Machines. 1965 IEEE Conference Record on S.C.T.L.D., pp. 71-83. October, 1965.

14. Davis, W.A. Generation of Delayed Replicas of M-sequences. Proc. IEE, Vol. 113, pp. 295-6. February, 1966.

15. Davis, W.A. An Approach for Determining Order Code Assignments for Digital Computers. Proc. 1966 British Joint Computer Conference, Publication No. 19, pp. 212-219. May, 1966.

16. Davis, W.A. Automatic Delay Changing Facility for Delayed M-sequences. Proc. IEEE, Vol. 54, pp. 913-4. June, 1966.

17. Davis, W.A. Single Shift Register Realizations for Sequential Machines. Submitted for publication in IEEE Trans. on Electronic Computers.

18. Davis, W.A. An Approach to the Assignment of Input Codes. Submitted for publication in IEEE Trans. on Electronic Computers. Also as University of Ottawa, Technical Report No. 65-18. October, 1965.

19. Davis, W.A. and J.A. Brzozowski. On the Linearity of Sequential Machines. IEEE Trans., Vol. EC-15, pp. 21-29. February, 1966.

20. Dolotta, T.A. The Coding Problem in the Design of Switching Circuits. PhD dissertation, Princeton University, Princeton, N.J. May, 1961.

21. Dolotta, T.A. and E.J. McCluskey, Jr. Encoding of Incompletely Specified Boolean Matrices. Proc. W.J.C.C., Vol. 17, pp. 231-238. May, 1960.

22. Dolotta, T.A. and E.J. McCluskey, Jr. Minimal n-Point Functions on n-Cubes. Digital Systems Lab., Dept. of the E.E., Technical Report No. 2, Princeton University, N.J., November, 1960.

23. Dolotta, T.A. and E.J. McCluskey. The Coding of Internal States of Sequential Circuits. IEEE Trans., Vol. EC-13, pp. 549-562. October, 1964.

24. Elspas, B. The Theory of Autonomous Linear Sequential Networks. IRE Trans., Vol. CT-6, pp. 45-60. Also in [45]. March, 1959.

25. Fleck, A.C. Isomorphism Groups of Automata. Jour. A.C.M., Vol. 9, pp. 469-476. October, 1962.

26. Frank, H. and S.S. Yau. Improving Reliability of a Sequential Machine by Error-correcting State Assignments. IEEE Trans., Vol. EC-15, pp. 111-113. February, 1966.

27. Friedes, A. Critical Pairs and Set Systems. IEEE Trans., Vol. EC-15, pp. 102-104. February, 1966.

28. Friedman, A.D. Feedback in Synchronous and Asynchronous Sequential Switching Circuits. PhD Dissertation, Columbia University. 1965.

29. Friedman, A.D. Feedback in Synchronous Sequential Switching Circuits. IEEE Trans., Vol. EC-15, pp. 354-367. June, 1966.

30. Gill, A. Analysis of Linear Sequential Circuits by Confluence Sets. IEEE Trans., Vol. EC-13, pp. 226-231. June, 1964.

31. Gill, A. Analysis and Synthesis of Stable Linear Sequential Circuits. Jour. A.C.M., Vol. 12, pp. 141-149. January, 1965.

31a. Gill, A. The Minimization of Linear Sequential Circuits. IEEE Trans., Vol. CT-12, pp. 292-294. June, 1965.

32. Gill, A. and C.J. Tan. The Factorization of Linear Cycle Sets. IEEE Trans., Vol. CT-12, pp. 630-632. December, 1965.

33. Glushkov, V.M. Automata Theory and Its Applications. Proc. IFIP Congress 1965, Vol. 1, pp. 1-8. May, 1965.

34. Griffiths, T.V. The Decomposition of Finite State Machines. AFCRL, L.G. Hanscom Field, Mass., Report AFCRL-63-543. November, 1963.

35. Gustafson, C.H., et.al. Synthesis of Counters with Threshold Elements. 1965 IEEE Conference Record on S.C.T.L.D., pp. 25-35. October, 1965.

36. Hadlock, F.O. and C.L. Coates. Realization of Sequential Machines with Threshold Elements. 7th Annual Symposium on Switching and Automata Theory. Berkeley, California. October, 1966.

37. Haring, D.R. Sequential Circuit Synthesis: State Assignment Aspects. M.I.T., Press. 1965.

38. Harrison, M.A. On the Error Correcting Capacity of Finite Automata. Info. and Control, Vol. 8, pp. 430-450. August, 1965.

39. Hartmanis, J. The Structure of Finite Automata. Presented at Conference on Systems and Computer Science. The University of Western Ontario, London. September, 1965.

40. Hartmanis, J. Two Tests for the Linearity of Sequential Machines. IEEE Trans., Vol. EC-14, pp. 781-786. December, 1965.

41. Hartmanis, J. and R.E. Stearns. Algebraic Structure Theory of Sequential Machines. Prentice Hall, Inc., Englewood Cliffs, N.J. 1966.

42. Hartmanis, J. and W.A. Davis. Homomorphic Images of Linear Sequential Machines. Presented at Conference on Algebraic Theory of Machines with Applications, Pacific Grove, California. 29 August - 8 September, 1966.

44. Hennie, F.C. III. Iterative Arrays of Logical Circuits. Technology Press of MIT, Cambridge, Mass., and Wiley and Sons, New York. 1961.

45. Kautz, W.H. (ed). Linear Sequential Switching Circuits: Selected Technical Papers. Holden-Day, San Francisco. 1965.

46. Kilmer, W.L. Iterative Switching Networks Composed of Combinational Cells. IRE Trans., Vol. EC-11, pp. 123-131. April, 1962.

47. Kilmer, W.L. On Dynamic Switching in One-dimensional Iterative Logic Networks. Info. and Control, Vol. 6, pp. 399-415. December, 1963.

48. Kilmer, W.L. Topics in the Theory of One-dimensional Iterative Networks. Info. and Control., Vol. 7, pp. 180-199. June, 1964.

49. Kohavi, Z. Decomposition and State Assignment for Sequential Machines. Polytechnic Inst. of Brooklyn, Report PIBMRI-1231-64. August, 1964.

50. Kohavi, Z. Secondary State Assignment for Sequential Machines. IEEE Trans., Vol. EC-13, pp. 193-203. June, 1964.

51. Kohavi, Z. and E.J. Smith. Decomposition of Sequential Machines. 1965 IEEE Conference Record on S.C.T.L.D., pp. 52-61. October, 1965.

52. Krohn, K.B. and J.L. Rhodes. Algebraic Theory of Machines. Proc. Symposium on Mathematical Theory of Automata. Brooklyn, N.Y. April, 1962.

53. Liu, C.L. kthorder Finite Automaton. IEEE Trans., Vol. EC-12, pp. 470-475. October, 1963.

54. Liu, C.L. kth-Order Finite Automaton IEEE Trans., Vol. EC-13, p. 642. October, 1964.

55. Liu, C.L. Sequential Machine Realization Using Shift Registers. Proc. 5th Annual Symp. on S.C.T.L.D., Princeton, N.J. November, 1964. (Also as: Part V of Interim Engineering Report ESL-IR-196. March, 1964.)

56. Magleby, K.B. The Synthesis of Nonlinear Feedback Shift Registers. Stanford University, Stanford, California, Report No. 6207-1. October, 1963.

57. Maitra, K.K. Cascaded Switching Networks of Two-input Flexible Cells. IRE Trans., Vol. EC-11, pp. 136-143. April, 1962.

58. Massey, J.L. and R.W. Liu. Equivalence of Nonlinear Shift Registers. IEEE Trans., Vol. IT-10, pp. 378-379. October, 1964.

59. Masters, G.M. and R.L. Mattson. Threshold Logic Synthesis of Sequential Machies. Presented at 7th Annual Symposium on Switching and Automata Theory, Berkeley, California. October, 1966.

60. McCluskey, E.J. Jr., and S.H. Unger. A Note on the Number of Internal Variable Assignments for Sequential Circuits. IRE Trans., Vol. 8, pp. 439-440. December, 1959.

61. McNaughton, R. The Theory of Automata, a Survey, in Advances in Computers, Vol. 2, FL. Alt. (ed), Academic Press, New York. 1961.

62. Minnick, R.C. Cutpoint Cellular Logic. IEEE Trans., Vol. EC-13, pp. 685-698. December, 1964.

63. Moore, E.E. Sequential Machines: Selected Papers. Addison-Wesley, Reading, Mass. 1964.

64. Nelson, R.J. Basic Concepts of Automata Theory. Proc. 20th National Conference of the ACM, pp. 138-161. August, 1965.

65. Nichols, A.J. Multiple Shift Register Realization of Sequential Machines. Lockheed Missiles and Space Co., TR No. 6-74-64-48. October, 1964.

66. Nichols, A.J. Minimal Shift Register Realizations of Sequential Machines. IEEE Trans., Vol. EC-14. October, 1965.

67. Nichols, A.J. Modular Synthesis of Sequential Machines. 1965 IEEE Conference Record on S.C.T.L.D., pp. 62-70. October, 1965.

68. Nichols, A.J. and A.J. Bernstein. State Assignments in Combinational Networks. IEEE Trans., Vol. EC-14, pp. 343-349. June, 1965.

69. Nievegelt, J. Partially Ordered Sets of Finite Automata. 1965 IEEE Conference Record on S.C.T.L.D., pp. 229-234. October, 1965.

70. Pugsley, J.H. Linear Sequential Machines Whose Logic Elements Contain Delay. Ph.D. dissertation, University of Illinois. 1963.

71. Puglsey, J.H. Sequential Functions and Linear Sequential Machines. IEEE Trans., Vol. EC-14, pp. 376-382. June, 1965.

72. Russo, R.L. Synthesis of Error-Tolerant Counters Using Minimum Distance Three State Assignments. IEEE Trans., Vol. EC-14, pp. 359-366, June, 1965.

73. Sheng, C.L. The Theory of the Single-Loop Y-Cascaded-Delay Sequential Machine I - The Definite-Event Machine Approach. Dept. of E.E., University of Ottawa, Technical Report No. 65-17. September, 1965.

74. Sheng, C.L. The Theory of the Single-Loop Y-Cascaded-Delay Sequential Machine II - Relationships with Linear Machines and Autonomous Machines. Dept. of E.E., University of Ottawa, Technical Report No. 66-4. May, 1966.

75. Sklansky, J. General Synthesis of Tributary Switching Networks. IEEE Trans., Vol. EC-12, pp. 464-469. October, 1963.

76. Srinivasan, C.V. State Diagram of Linear Sequential Machines. Jour. Franklin Inst., Vol. 273, pp. 383-418. (Also in [45]) May, 1962.

77. Srinivasan, C.V. The State Diagram of Linear Sequential Machines. Doctor of Engineering Science Dissertation, Columbia University. 1963.

78. Stone, H.S. and A.J. Korenjak. Canonical Form and Synthesis of Cellular Cascades. IEEE Trans., Vol. EC-14, pp. 852-862. December, 1965.

79. Susskind, A.K. et al. Synthesis of Sequential Switching Networks. Final Technical Documentary Report, M.I.T. Report ESL-FR-216. October, 1964.

80. Susskind, A.K. Threshold Element and The Design of Sequential Switching Networks. Rome Air Dev. Center, Griffiss AFB, N.Y., Technical Report No. RADC-TR-66-286. June, 1966.

81. Toda, I. A Theory of Linear Machines. Department of Electrical Engineering, University of California, Unpublished manuscript.

82. Winograd, S. Input Error Limiting Automata. Jour. A.C.M., Vol. 11, pp. 338-351. July, 1964.

83. Yau, S.S. Autonomous Clocks in Sequential Machines. IEEE Trans., Vol. EC-14, pp. 467-472. June, 1965.

84. Yau, S.S. and K.C. Wang. Linearity of Sequential Machines. IEEE Trans., Vol. EC-15, pp. 337-354. June, 1966.

85. Yoeli, M. The Cascade Decomposition of Sequential Machines. IRE Trans., Vol. EC-10, pp. 587-592. December, 1961.

86. Yoeli, M. Cascade-Parallel Decomposition of Sequential Machines. IEEE Trans., Vol. EC-12, pp. 322-324. June, 1963.

87. Yoeli, M. Generalized Cascade Decompositions of Automata. Journal JCM., Vol. 12, pp. 411-422. July, 1965.

88. Yoeli, M. A Group Theoretical Approach to Two-Rail Cascades. IEEE Trans., Vol. EC-14, pp. 815-822. December, 1965.

89. Zahle, T.U. On Coding the States of Sequential Machines with the Use of Partition Pairs. IEEE Trans., Vol. EC-15, pp. 249-253, April, 1966.

90. Zeiger, H.P. Loop-Free Synthesis of Finite State Machines. Ph.D. Thesis, Mass. Inst. of Tech. September, 1964.

91. Zeiger, H.P. Cascade Synthesis of Finite-State Machines. 1965 IEEE Conference Record on S.C.T.L.D., pp. 45-51. October, 1965.

V I T A

Name: Wayne Alton Davis

Born: 16 November, 1931. Fort Macleod, Alberta.

Educated:

Primary: Fort Macleod, Alberta

Secondary: Fort Macleod, Alberta.

University:

(a) The George Washington University,
Washington, D.C.

1960 Bachelor of Science in Engineering
(B.S.E.)

Mathematics Option.

(b) University of Ottawa,
Ottawa, Ontario.

1963 Master of Science
(M.Sc.)

Electrical Engineering