SOME STATE-ASSIGNMENT TECHNIQUES UTILIZING FLIP-FLOPS FOR SEQUENTIAL MACHINE REALIZATION

by

Godfrey W.Muehle, B.A.Sc.

Submitted to the Department of Electrical Engineering in partial fulfillment of the requirements for the degree of Master of Applied Science

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa, Onterio

April, 1972

ABSTRACT

This thesis is concerned with the state assignment problem of Sequential Machines. Sequential Machines are introduced. A brief discussion of partitions and their relation to the secondary state assignment is presented. Several known techniques of state assignments are discussed in comparison to Flip-Flop Memory Elements and Delay Memory Elements. A new approach to the solution of obtaining an "optimum" realization using Flip-Flops as Memory Elements of a Sequential Machine is presented.

ACKNOWLEDGEMENTS

The author would like to express his sincere gratitude to Dr.S.R.Das, of the Electrical Engineering Department,for his advice and criticism during the preparation of this thesis.

Gratitude is also expressed to Dr. W.A. Davis who introduced me to the subject and provided me with many thought provoking questions and to my colleagues the graduate students in the Electrical Engineering Department.

The author gratefully acknowledges the support that was received from the Northern Electric Company, Limited under the Post-Graduate Bursary Plan.

CONTENTS PAGE

I INTRODUCTION

1.1 Sequential Machines and Sequential Circuits

The last decade has experienced an increased activity in the development of the theory of Sequential Machines, referred to by various authors as finite automata, finite state machines, or switching circuits with memory. This surge of interest in sequential machine theory originated from the fact, that physical devices such as electronic computers, digital systems, switching systems etc., can be modelled by sequential machines, hence, providing a mathematical model for theoretical studies. D. A. Huffman, E. F. Moore and G. H. Mealy appear to be the first authors of published papers concerning sequential machines. Each author has some specialty in his formalization of a sequential machine model. As an example, in Moore's Model [13] the sequential machine outputs are not dependent on the inputs at a particular time, in contrast to the Mealy Model [14] where the outputs are input dependent; this is also true for the model used by Huffman [15] except that he restricts himself to the application of electromechanical devices.

Before a formal definition of a sequential machine is given, we shall briefly state some general facts with appropriate configurations.

A sequential machine is the mathematical model of its corresponding sequential circuit. Let SM1 be any arbitrary sequential machine, then its flow table and state diagram are given in Figures 1.1a and b respectively.

Present States	Inputs			
	x_1	x_2	x_1	x_2
S_1	S_1	S_4	Z_1	Z_1
S_2	S_3	S_3	Z_2	Z_2
S_3	S_3	S_3	Z_2	Z_2
S_4	S_1	S_3	Z_1	Z_2
	Next States		Outputs	

FIGURE 1.1a: FLOW TABLE OF SEQUENTIAL MACHINE SM1

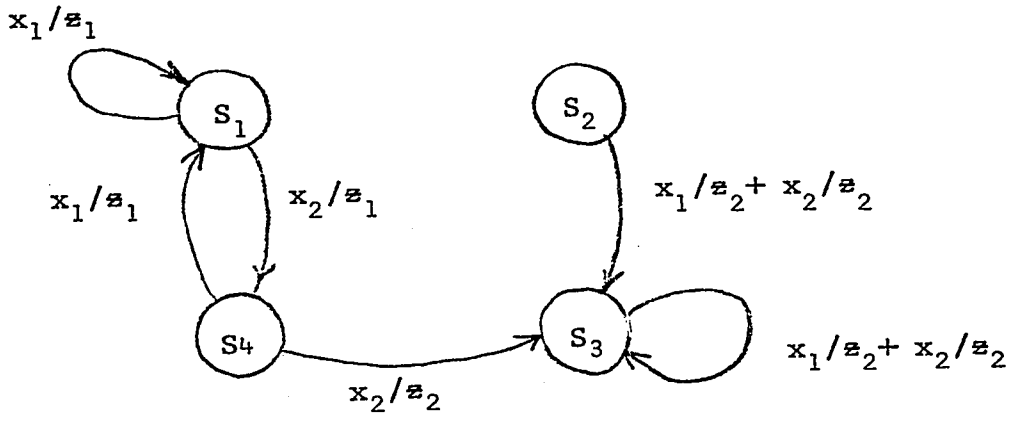

FIGURE 1.1b: STATE DIAGRAM OF SEQUENTIAL MACHINE SM1

An arbitrary code assignment of the states will convert the flow table in figure 1.1a to the modified form as shown in figure 1.1c.

Present States		Inputs			
y_1	y_2	$x_1 = 0$	$x_2 = 1$	$x_1 = 0$	$x_2 = 1$
0	0	0 0	1 0	0	0
0	1	1 1	1 1	1	1
1	1	1 1	1 1	1	1
1	0	0 0	1 1	0	1
		Y_1, Y_2, Z			

FIGURE 1.1c: FLOW TABLE OF SM1 WITH STATE ASSIGNMENT SPECIFIED

The next state equations resulting from this assignment are seen to be:

$$Y_1 = y_2 + x_2$$

$$Y_2 = y_2 + x_2 y_1$$

$$Z = y_2 + x_2 y_1$$

and when realizing these euqations we obtain the following sequential circuit:

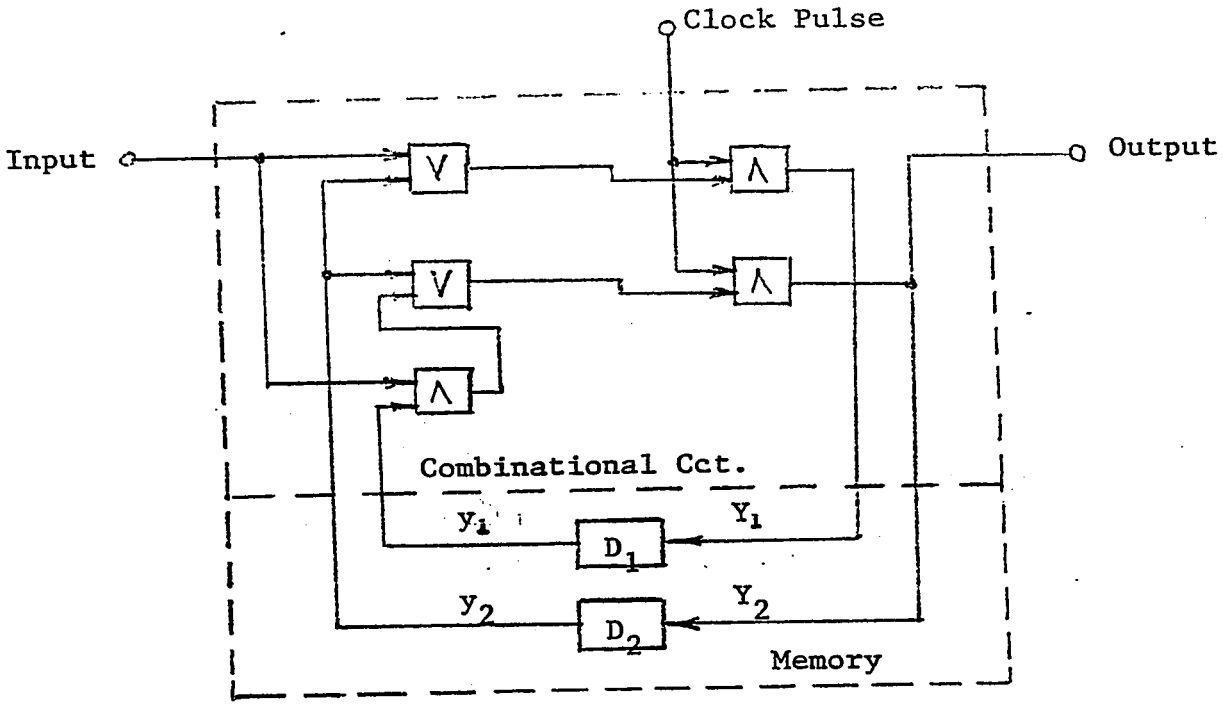

FIGURE 1.2: SEQUENTIAL CIRCUIT OF SM1

It should be obvious by now that the sequential circuit under consideration utilizes clock pulses to control the timing of changes of memory elements. These circuits are generally known as synchronous sequential circuits, and only these circuits are discussed in this thesis.

Finally, we state the Boolean equations describing the sequential circuit in Fig 1.3.

$$Y_1 = y_1(t+1) = f_1(x_1(t), x_2(t), \ldots, x_m(t), y_1(t), y_2(t), \ldots, y_n(t))$$

V = OR gate

Λ = AND gate

$$Y_i = f_i(x_1(t), x_2(t), \ldots, x_m(t), y_1(t), y_2(t), \ldots, y_n(t))$$

$$\vdots$$

$$Y_n = f_n(x_1(t), x_2(t), \ldots, x_m(t), y_1(t), y_2(t), \ldots, y_n(t))$$

$$Z_j(t) = f_j(x_1(t), x_2(t), \ldots, x_m(t), y_1(t), y_2(t), \ldots, y_n(t))$$

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

$j = 1, 2, \ldots, r$

t = time of occurrence

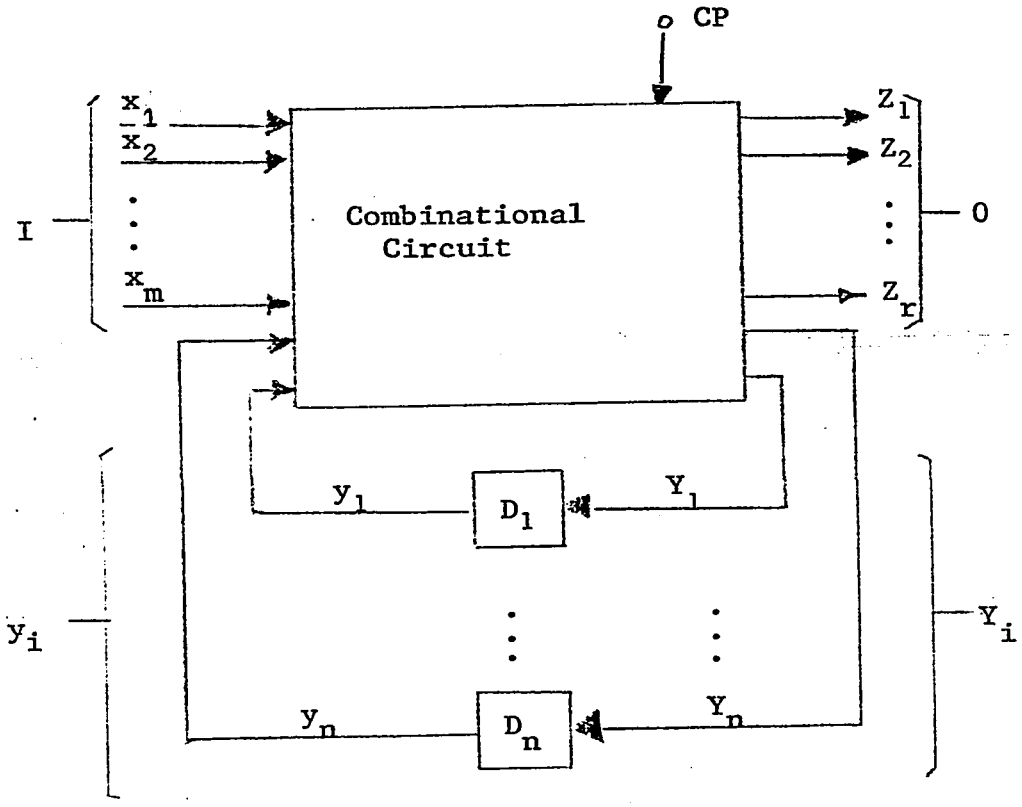

FIGURE 1.3: GENERAL FORM OF A SYNCHRONOUS SEQUENTIAL CIRCUIT

1.2 The State Assignment Problem

The essence of an economical realization of a sequential machine centers around the code assignment of the secondary variables. It is one of the most important steps, but also the most complicated step in the design procedure. Theoretically, the code assignment problem is solveable since any code assignment will lead to a properly working sequential circuit, and furthermore, to obtain the most economical design we merely examine all possible assignments on a comparative basis and select the desired one to implement the circuit. This will always provide us with the most economical sequential circuit. Unfortunately, this enumerative approach reaches monstrous dimensions and even with the aid of high-speed computers it remains impractical for all but the smallest of sequential machines. As an example, consider a sequential machine with four memory elements. All possible assignments of codes for the four memory elements would require consideration of 2.0923×10^{13} different assignments, resulting in different functions which are not all necessarily distinct.

For a given flow table containing k rows, i.e. k states, there are $\binom{2^n}{k}$ ways to select k distinct states and k! ways to present them, consequently there are

$$\binom{2^n}{k} k! = \frac{(2^n)!}{(2^n-k)!}; \qquad \text{(E 1.1)}$$

possible assignments. Two assignments are distinct if it is not possible to obtain one assignment from the other by complementing and permuting variables. The number of distinct assignment of n variables to k states is

$$\binom{2^n}{k} k! \left(\frac{1}{n!2^n}\right) = \frac{(2^n-1)!}{(2^n-k)!n!} \qquad \text{(E 1.2)}$$

Consider a machine with 16 states, then we require $n = \log_2 16 = 4$ binary memory elements and evaluating (E 1.2) one obtains $\simeq 5.448 \times 10^{10}$ distinct assignments.

Before we conclude this section, let us make one more statement concerning the number of significant Boolean functions. For n binary variables there are 2^n combinations, each one of which may be a zero or a one for a function; hence, there are 2^{2^n} functions of n variables. However, many of these functions are equivalent. For only 9 variables the total number of functions is given by a fantastic figure of [1] $2^{2^9} = 2^{512} \simeq 2.6 \times 10^{24}$.

From the preceding discussion it becomes evident, that a completely exhaustive approach to the code assignment problem is not only impractical, but in many cases impossible.

Many attempts have been made to solve the problem, but only with partial success. However, from these emerged two general ways of approach to the problem.

One method is distinguished by its inherent association with a certain weighing scheme, where a collection (or set) of states is given a weight or a score, according to some predetermined rules, and either the high scoring or the low scoring sets are being utilized to effect an assignment. Since this method examines a subset of all possible assignments and selects on a comparative basis, it is almost essential, in order to be practical, that the method be programmable on an electronic computer.

---

[1] An estimated number of "protons and electrons in the universe" is given by $N = 3/2 \times 136 \times 2^{256}$; according to A.S. Eddington, "Fundamental Theory", Cambridge University Press 1946, pp 283.

The other method utilizes partition algebra, lattice theory and group theory to effect a secondary assignment. In this area, one also finds that machine decomposition plays a predominant role, since, whenever these methods yield a solution, the SM is decomposed into serial, parallel or cross-coupled smaller operating segments. This method works very well whenever the SM lends itself to decomposition but, unfortunately, only a small subclass of all SM's fall into that category and for the larger part of SM's these procedures are unproductive.

1.3 Algebra of Sets and Partitions

This portion of the present chapter is intended to provide the necessary mathematical background for this thesis.
Modern Algebra, in particular set theory, semi groups and lattice theory have now been fully accepted as part of sequential machine theory. A more extensive coverage can be found in [6,8].

Let S and R be non-void sets. Then a function f of S into R is defined as

$$f: S \rightarrow R$$

such that

$$f(s) = r$$

where $s \in S$ and $r \in R$.

Should a set be empty then we denote this set by $\emptyset$.

Definition 1.301:

A partition P on a set S is a collection of disjoint subsets of S such that their set union is S.

Definition 1.302:

The sets of P are called blocks of P which are denoted by B and distinguished by bars and semicolons [6].

Definition 1.303:

A binary partition (bp) [7] over a set S, is a partition having only two blocks. Partition $P = \{\overline{S};\emptyset\}$ is considered to be the trivial binary partition.

Definition 1.304:

The product of two partitions (g.l.b)[1] $P_1 * P_2 = \{B_1^i * B_2^j) \mid B_1^i \epsilon P_1$ and $B_2^j \epsilon P_2$; $i = |P_1|$ and $j = |P_2|\}$, where $|P|$ denotes the number of blocks contained in P_k. The operator * is to be interpreted as intersection and when no ambiguity arises we simple write $P_1 * P_2$ as $P_1 P_2$.

Example:

Let $S = \{a,b,c,d,e,f,g\}$

$P_1 = \{\overline{a,b,c};\ \overline{d,e,f};\ \overline{g}\}$ $P_2 = \{\overline{a,b,c,d};\ \overline{e,f,g}\}$

then

(i) $\{a,b,c\} \cup \{d,e,f\} \cup \{g\} = S$

$\{a,b,c\} \cap \{d,e,f\} \cap \{g\} = \emptyset$

(ii) P_2 is a binary partition

(iii) B_1, B_2, B_3 of P_1 are $\overline{a,b,c}$; $\overline{d,e,f}$ and $\overline{g}$ respectively

(iv) $P_1 P_2 = B_1 B_2 : B_1^1 B_1^2;\ B_1^1 B_2^2;\ B_2^1 B_2^2;\ B_3^1 B_1^2;\ B_3^1 B_2^2$

$= \{a,b,c,;\emptyset;d;e,f;\emptyset;g\}$

$= \{\overline{a,b,c};\ \overline{e,f};\ \overline{d};\ \overline{g}\}$

Definition 1.305:

A binary partition on a set of states S and $|S| = q$ being the number of states, is called a qualified binary partiton (qbp) if and only if $|\tilde{B}| \leq 2^{n-1}$, where $n = [\log_2 q]$, $|\tilde{E}|$ = the number of states in block B or $\overline{B}$, and $[\log_2 q] = N$ is the smallest integer $\geq [N]$

---

(1) The greatest lower bound (g.l.b) as applied to partitions. Reference [6] and [12]

Definition 1.306:

Let S be a set and $|S| = q$, where q is the number of elements or states is S. Let $P_1, P_2, \ldots, P$ be m binary partitions of S. If

$$\left|\tilde{B}_1 \tilde{B}_2 \ldots \tilde{B}_m\right| \leq 2^{n-m}$$

for all possible combination of $\tilde{B}_i$, $i = 1, 2, \ldots m$, where $1 < m \leq n$, then we say that the partitions $P_1, P_2, \ldots, P_m$ are mutually consistent.

These two definitions provide us with a means of selecting a subset from the set of all partitions of S, which may be useful to effect a valid assignment. A fundamental requirement for all valid assignments has been formulated in [7] and is repeated here.

Theorem 1.31:

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]

An example will demonstrate the previous statements:

Let S = {1,2,3,4,5,6,7,8} and

Let $P_1 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$

$P_2 = \{\overline{1,2,5,6};\ \overline{3,4,7,8}\}$

$P_3 = \{\overline{1,3,5,7};\ \overline{2,4,6,8}\}$

$P_4 = \{\overline{1,4};\ \overline{2,3,5,6,7,8}\}$

$$P_5 = \overline{1,3,5,6}; \overline{2,4,7,8}$$

Obviously all P_i, i = 1, ..., 5, are binary partitions. Which of these are qualified binary partitions? In accordance with definition 1.306 we find that

$$|S| = q \text{ and } n = \log_2 q = \log_2 8 = 3$$

and hence

$|B| \leq 2^{n-1} = 2^2 = 4$ will satisfy the qbp requirement.

$$|\tilde{B}_1| = (4,4)$$
$$|\tilde{B}_2| = (4,4)$$
$$|\tilde{B}_3| = (4,4)$$
$$|\tilde{B}_4| = (2,6)$$
$$|\tilde{B}_5| = (4,4)$$

where $(|B|, |\overline{B}|)$ indicates the number of states in block B and $\overline{B}$ respectively. Clearly the only non-qualified binary partition is P_4. The remaining partitions must now be checked for mutual consistency, and in the quest of brevity we will restrict ourselves to a few samples.

Consider m = 2 and in particular P_1 and P_5

then $|B_1 B_5| \leq 2$
$|B_1 B_5| \leq 2$
$|B_1 \overline{B}_5| \leq 2$
$|\overline{B}_1 \overline{B}_5| \leq 2$

where $B_1 = (1,2,3,4)$, $\overline{B}_1 = (5,6,7,8)$

and $B_5 = (1,3,5,6)$, $\overline{B}_5 = (2,4,7,8)$

Evaluating $|B_1 . B_5|$ we obtain $|B_1 B_5| = |(1,3)| = 2$ and the contition is satisfied.

Continuing this process, we will find that P_1 and P_5 are mutually consistent. This is also true for P_1P_2, P_1P_3, and P_2P_3. But definition 1.306 is not satisfied for the product of $P_2P_5$and P_3P_5, hence these qualified binary partions are not mutually consistent.

Next consider m = 3. It can easily be shown that the only three mutually consistent partitions are $P_1P_2P_3$. These three partitions will now provide a valid secondary assignment, as demonstrated below:

(1,2,3,4)→0	(1,2,5,6)→0	(1,3,5,7)→0
(5,6,7,8)→1	(3,4,7,8)→1	(2,4,6,8)→1

therefore:

	y_1	y_2	y_3
1 →	0	0	0
2 →	0	0	1
3 →	0	1	0
4 →	0	1	1
5 →	1	0	0
6 →	1	0	1
7 →	1	1	0
8 →	1	1	1

1.4 Cubes and Subcubes

Let K^n denote an n- dimensional cube and let a single vertex be called a zero cube.

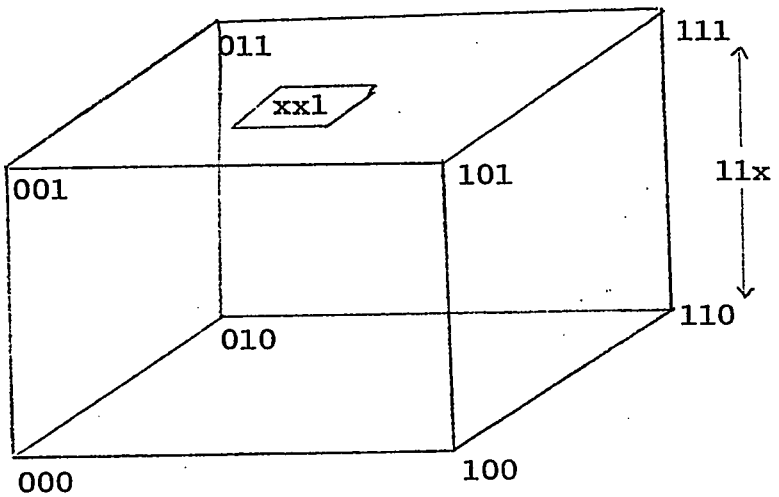

FIGURE 1.41: AN n=3 DIMENSIONAL CUBE

The first dimensional expansion of the zero cube will be a one cube or an edge. Hence a pair of adjacent zero cubes, i.e. points connected by a line, will constitute a one cube.

Examples: (refer to figure 1.41).

(i) Any vertex designated 000, 001, 010 etc. is a zero cube. There are a total of 8 zero cubes in a 3 dimensional cube.

(ii) The line joining the two adjacent zero cubes 111 and 110 is the one cube 11x, where x = {0,1}. There are a total of 12 one cubes.

(iii) The area enclosed by the four zero cubes 001, 101, 111 and 011 is a two cube and is designated xx1, where x = {0,1}. There are a total of 6 two cubes.

How do we relate the preceding concepts to Boolean Algebra? Consider a function f and let K^0 (f) denote the vertices of the n-cube which are mapped into 1. Hence, K^0 (f) represents a set of zero cubes, and similarly, K^1 (f) is to be the set of all one cubes generated by K^0 (f). Thus, if f is a function of n variables, K (f) is a union of sets of all cubes generated from f; or

$$K(f) = \bigcup_{i=0}^{n-1} K^i(f)$$

Example:

Let $f(x_1,x_2) = \bar{x}_1\bar{x}_2 + \bar{x}_1x_2 + x_1\bar{x}_2$;

then the set of zero cubes are

$K^0(f) = \{(0,0),(0,1),(1,0)\}$;

The set of one cubes are

$K^1(f) = \{(0,\phi),(\phi,0)\}$, where $\phi = \{x_i,\bar{x}_i\}$

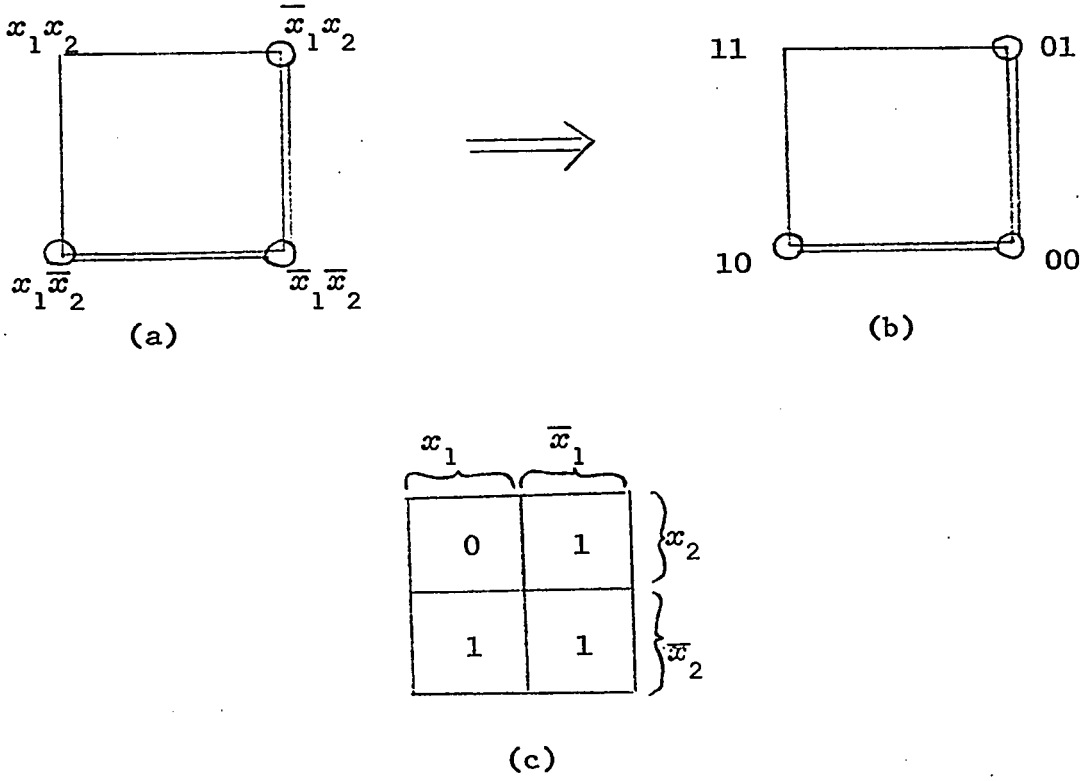

FIGURE 1.42: REPRESENTATIONS OF $f(x_1,x_2)$

(a) $f(x_1,x_2)$ drawn on a two cube (b) $f(x_1,x_2)$ drawn on a two cube, where $x_i \rightarrow 1$ and $x_i \rightarrow 0$ (c) $f(x_1,x_2)$ mapped.

Taking the highest ranking cubes, i.e., $K^1(f)=\{(0,\phi),(\phi,0)\}$ we see that all lower cubes can be imbedded into the one cubes.

$\therefore f(x_1,x_2)= \overline{x}_1+ \overline{x}_2$

This procedure, when extended to include prime subcubes and essential prime subcubes, is known as Quine's Algorithm. The map representation in figure 1.42 (c) is usually used to minimize litterals of a given $f(x_i)$.

II SOME CODING PROCEDURES

2.1 Secondary State Assignment Technique by Armstrong [4,5]

The two main features of this technique are the (p,r) mappings, to be defined shortly, and an associated numerical scoring of the (p,r) mappings. The flow table may be looked upon as a mapping of present states into next states by the inputs. A subset of these mappings is used for the secondary state assignment. Armstrong's type I and type II adjacencies [4] are the simplest types of (p,r) mappings.

We begin by showing how certain mappings are employed to obtain an efficient code assignment.

Definition 2.01:

Let the subset of next states be {Q} , the subset of present states be {P} and the subset of inputs be {I}, then the next state function is defined as

$$\delta : SXI \rightarrow S$$

and can be written symbolically as

$$\{P\} \xrightarrow{\{I\}} \{Q\}$$

Assigning the subset {P} to a p-cube of the n-cube such that all $s \in P$ are contained in the p-cube, and assigning all $s \in Q$ to a q- cube of the n-cube such that all $s \in Q$ are contained in a q-cube, where $q<n$ and $Q \not\subset P$, then the Karnaugh map representation of this

mapping will exhibit a secondary variable relationship. It has been shown [4] that a mapping will make the simplest possible contribution to at least one y-function by fulfilling two necessary conditions:

(1) The states of {P} are to be assigned to completely fill as few subcubes as possible on the n-cube, each subcube being as large as possible.

(2) The states of {Q} must be assignable to a subcube of dimension less than n.

Whenever the states $s \varepsilon Q$ are assigned to the q-subcube, then each of the (n-q) variables will have the same code bit, i.e. secondary assignment. This will provide a good partial secondary assignment which is determined by only a single mapping. To achieve a good overall assignment we must find a subset of all possible mappings such that the states can be assigned to fulfill conditions 1 and 2 for each mapping simultaneously. Furthermore, the subset must include enough mappings to result in a complete and unique secondary state assignment. This then constitutes a <u>basic set</u> and to obtain a state assignment which results in a set of y-functions having an economical form we select only particular mappings. Mappings of this type are called optimal basic sets.

An example will demonstrate the preceding features. Consider the flow table of SM1 given below:

Present States	Inputs	
	0	1
A	G	D
B	G	E
C	H	F
D	H	E
E	A	F
F	B	A
G	C	B
H	A	A

FIGURE 2.1: MACHINE SM1

Since SM1 has 8 states, n = 3; hence a 3-cube suffices for the assignment. Now consider the first 4 states A,B,C,D and the corresponding next states for input 0, i.e. G and H. Symbolically this can be written

$$\{I\}$$

$$\{P\} \rightarrow \{Q\}$$

where {P} = {A, B, C, D}; {I} = {0} and {Q} = {G, H} resulting in

$$\{0\}$$

$$\{A,B,C,D,\} \rightarrow \{G,H\}$$

P contains 4 states requiring a 2-subcube or p = 2; similarly, we see that q = 1.

If we assign the 2-subcube to 0xx, that is, the secondary variables of the four states in {P} are assigned to 011, 001, 000

and 010. The exact location of the states within the 2-subcube is irrelevant for the mapping. Now let us assign the two states G and H to a 1- subcube in the region $11x$. The resulting mapping can now be presented as

$$0$$

$$0xx \rightarrow 11x$$

This implies that the mapping contributes only one term to the y_1 and y_2 functions. Let the states within the two sets {P} and {Q} be assigned as follows:

	y_1	y_2	y_3
A → 0	0	0	
B → 0	0	1	
C → 0	1	0	
D → 0	1	1	
G → 1	1	0	
H → 1	1	1	

The remaining two states have to be assigned to the 1-subcube in the $10x$ region. Hence we let

	y_1	y_2	y_3
E → 1	0	0	
F → 1	0	1	

Entering the assignments into the flowtable of SM1 we have

Present States	Inputs 0	1
000	110	011
001	110	100
010	111	101
011	111	100
100	000	101
101	001	000
110	010	001
111	000	000

$Y_1\ Y_2\ Y_3$

FIGURE 2.2: STATE ASSIGNMENTS OF SM1

Now let us consider the mapping $0xx \xrightarrow{0} 11x$ again in consideration to the contribution of this mapping to the Y_1, Y_2 functions.

$$Y_1 = \bar{y}_1 \bar{x} + \text{other terms}$$

$$Y_2 = \bar{y}_1 \bar{x} + \text{other terms}$$

For the purpose of this example we ignore the other terms because they are not due to the mapping under consideration. Also it should

be clear now that the exact location of the states within the 2-subcube is of no consequence for this particular mapping.

From the total number of possibly mappings, almost 2^{S+I}, where S is the number of internal states and I is the number of input states, we must select a subset which will provide us with a good assignment. The following three definitions will reduce the number of mappings to be taken into account considerably.

Definition 2.02: A p,r mapping is a mapping having 2^p states in {P} and the input symbols form an r- dimensional sub-cube of I.

Definition 2.03: A selectable p,r mapping is a mapping where the states of {P} can be assigned to a p-subcube and the states of {Q} can be assigned to a q-subcube of dimension <n and {P} and {Q}can be simultaneously assigned to p- subcube and q- subcube on the n- cube.

Definition 2.04: A set of mappings is mutually compatible if the mappings comprising the set are simultaneously selectable.

One can see an analogy between mutually consistent and mutually compatible mappings if the latter comprises only mappings of a binary nature.

The following example will illustrate the meaning of the three definitions. Consider SM2 in figure 2.3 below and a partial selection of sets from SM2.

Present States	Inputs			
	x_1	x_2	x_3	x_4
A	H	A	C	A
B	B	F	D	D
C	F	A	B	B
D	A	F	B	H
E	E	A	D	H
F	H	F	A	D
G	C	G	F	A
H	D	B	C	A

FIGURE 2.3: MACHINE SM2

let $\{P_1\}=\{ABCD\}$, $\{I_1\}=\{x_1x_2\}$, $\{Q_1\}=\{ABFH\}$

$\{P_2\}=\{CD\}$, $\{I_1\}=\{x_1x_2\}$, $\{Q_2\}=\{AF\}$

$\{P_3\}=\{CDEF\}$, $\{I_3\}=\{x_3x_4\}$, $\{Q_3\}=\{ABDH\}$

According to the definition 2.02 all three partitions P_1, P_2 and P_3 and the associated partitions I_1, I_2 and I_3 are qualifying as p,r mappings, since $P_1 = 2^p$

$P_2 = 2^p$

and $P_3 = 2^p$

where, in each case, p = 2 or 1. Similarly, the input partitions form an "n" dimensional subcube of I, with r=1.

To determine the selectability of the p,r mappings, we check for the relation $|Q| < 2^{n-1}$ and that the states appearing in both {P} and {Q} form a subset. $\{P_1\}$ = {ABCD} and $\{Q_1\}$ = {ABFH} fulfill this requirement, since the common states A and B can be assigned to a common subcube of $\{P_1\}$ and {Q} in the following manner.

For $\{P_1\}$:

A → 0 1 0

B → 0 1 1

C → 0 0 x

D → 0 0 x

For $\{Q_1\}$:

A → 0 1 0

B → 0 1 1

F → 1 1 1

H → 1 1 0

This relationship is best shown on a map (see figure 2.4). It is also fairly obvious that both sets $\{P_2\}$ and $\{Q_2\}$ are

y_1y_2 \ y_3		0	1
0	0	C	D
0	1	A	B
1	1	F	H
1	0	∅	∅

FIGURE 2.4: $\{P_1\}$ AND $\{Q_1\}$ MAPPED ON THE n-TUPPLE OF {0,1}

selectable p,r mappings, since both sets can be selected without interfering with the code assignment. This is not true for $\{P_3\}$ and $\{Q_3\}$. Although, $\{P_3\}$ and $\{Q_3\}$ are p,r mappings, they are not selectable because the common state D prevents the assignment of $\{P_3\}$ to a p-cube and $\{Q_3\}$ to a q-cube on the n-cube of SM2.

With $\{P_3\}$ and $\{Q_3\}$ eliminated we now determine whether or not the remaining two mappings are simultaneously selectable, i.e. mutually compatible. Once $\{P_1\}$ and $\{Q_1\}$ have been selected and assigned according to figure 2.4 we have only to confirm that $\{P_2\}$ can be mapped into $\{Q_2\}$ without interfering with the previous assignment. They do not; consequently, the two mappings are mutually compatible. Obviously, $\{P_1\}$, $\{Q_1\}$ and $\{P_3\}$, $\{Q_3\}$ are not mutually compatible.

Armstrong [4] proposes a table look-up method for both the selectable mappings and the mutually compatible mappings, but at the same time he points out, rightly so, that the construction of such tables would be a vast undertaking.

We now turn our attention to the scoring method utilized to effect a selection of mappings which will result in a valid assignment. Under consideration are two types of scoring functions. The first one, called the individual scoring function, which states

a relation between the map vertices assigned (or consumed) and the number of codes assigned, is

$$S_c = \frac{N_A}{N_V} \qquad \text{(E2.1)}$$

where N_A = number of assignments eliminated

N_V = number of map vertices eliminated

The number of map vertices whose values are assigned is

$$N_V = (n - q) \cdot 2^{p+r} \qquad \text{(E2.2)}$$

The number of assignments eliminated is N_A, but, due to lack of a better method, it is approximated by

$$N_B = \Sigma_j (n - d_j) \qquad \text{(E2.3)}$$

where d_j is the dimension of the smallest cube containing state j. This results in a formula which is similar to (E2.1); however, it is only an approximation, and is given by

$$S_c = \frac{2^m \cdot N_B}{N_V} \qquad \text{(E2.4)}$$

where 2^m is normalizing factor and the (0,0) mappings, i.e. $N_B = 0$, are being rejected.

The second scoring method called marginal scoring, is very closely related to the individual scoring method and is given by

$$S_c = \frac{2^m \cdot \Delta N_B}{\Delta N_V} \qquad \text{(E2.5)}$$

where $\Delta N_B = (\sigma N_B)_{M+1} - (\sigma N_B)_M$

$\Delta N_V = (\sigma N_V)_{M+1} - (\sigma N_V)_M$

Here M is the number of mappings. It follows that $(\sigma N_B)_M$ is the number of code bits used up by the M selected mappings, and $(\sigma M_B)_{M+1}$ is the number of code bits used up by the M selected mappings plus the one mapping under consideration.

Whenever the flow table of a sequential machine is incomplete, i.e. the flowtable exhibits don't care states, then Armstrong proposes two basically different approaches.

In the first procedure one ignores the existence of the don't care states. Proceed to determine an assignment as discussed previously, where all the {P} subsets contain only specified states. Once the specified states have been uniquely fixed on the n-cube, the remaining vertices, representing don't care states, are then used to simplify the resulting next state functions.

The second procedure introduces the don't care states immediately which will generate mappings with mixed subsets {P}. However, for computing N_B and ΔN_B one considers only the code bits of specified states, whereas in the calculation of N_V and ΔN_V, as well as for testing mappings for compatibility, don't care states must be taken into account. Consider the fully specified sequential machine SM3 as an example.

Present States	Inputs	
	x = 0	x = 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

FIGURE 2.5: FLOWTABLE OF SM3

Next we reorder the flowtable in order to facilitate the selection of (p,r) mappings.

Present States	Inputs	
	x = 0	x = 1
A	A	B
G	A	B
B	F	D
C	F	D
D	J	E
E	E	H
F	G	C
H	C	G

FIGURE 2.6: FLOWTABLE OF SM3 IN A RE-ORDERED FORM

From this table we select the (p,r) mappings.

(3,0) Mappings:

$\{A,B,C,D,E,F,G,H\} \xrightarrow{\{0\}} \{A,F,H,E,G,C\}$ (1)

$\{A,B,C,D,E,F,G,H\} \xrightarrow{\{1\}} \{B,D,E,H,G,C\}$ (2)

(2,1) Mappings:

$\{A,G,B,C\} \xrightarrow{\{0,1\}} \{A,B,F,D\}$ (3)

$\{A,G,D,E\} \xrightarrow{\{0,1\}} \{A,B,H,E\}$ (4)

$\{A,G,F,H\} \xrightarrow{\{0,1\}} \{A,B,C,G\}$ (5)

$\{B,C,D,E\} \xrightarrow{\{0,1\}} \{F,D,H,E\}$ (6)

$\{D,E,F,H\} \xrightarrow{\{0,1\}} \{H,E,G,C\}$ (7)

(2,0) Mappings:

$\{A,G,B,C\} \xrightarrow{\{0\}} \{A,F\}$ (8)

$\{A,G,D,E\} \xrightarrow{\{0\}} \{A,H,E\}$ (9)

$\{A,G,F,H\} \xrightarrow{\{0\}} \{A,G,C\}$ (10)

$\{B,C,D,E\} \xrightarrow{\{0\}} \{F,H,E\}$ (11)

$\{B,C,F,H\} \xrightarrow{\{0\}} \{F,G,C\}$ (12)

$\{D,E,F,H\} \xrightarrow{\{0\}} \{H,E,G,C\}$ (13)

$\{A,G,C,B\} \xrightarrow{\{1\}} \{B,D\}$ (14)

$\{A,G,D,E\} \xrightarrow{\{1\}} \{B,E,H\}$ (15)

$\{A,G,F,H\} \xrightarrow{\{1\}} \{B,G,C\}$ (16)

$\{B,C,D,E\} \xrightarrow{\{1\}} \{D,E,H\}$ (17)

$\{B,C,F,H\} \xrightarrow{\{1\}} \{D,G,C\}$ (18)

$\{D,E,F,H\} \xrightarrow{\{1\}} \{E,H,C,G\}$ (19)

Continuing with this procedure of finding (p,r) mappings, we would obtain a total of 40 mappings. Obviously, these (p,r) mappings represent only a subset of all possible mappings, and from the listed (p,r) mappings we select a basic set

BASIC SET = {3,5,6,7,8,10,11,13,14,16,19}

Note that the mappings (1) and (2) are not selectable because the condition

$$|Q| < 2^n - 1$$

is not met. The other mappings are not mutually compatible; hence they are not included in the basic set. To evaluate N_B and N_V we use the following two formulae:

(1) $\sigma N_B = \Sigma\ (n - d_j)$

(2) $\sigma N_V = (-1)^h \left[\sum_{i=1}^{M_h} (n-q_i^h) \cdot 2^{p_i^h} + r_i^h\right]$

where $q_i^h = q_i^{h-1} + q_j^{h-1} - q_{ij}^{h-1}$

$p_i^h = p_{ij}^{h-1}$

$r_i^h = r_{ij}^{h-1}$

$h = 1,2,\ldots,k$

---

(1) Armstrong provides an explanation of how the two formulae were derived.

Sample calculation for (3):

STATES	SETS	n - d
A	S12	3 - 1
B	S12	3 - 1
C	S1	3 - 2
G	S1	3 - 2
F	S2	3 - 2
D	S2	3 - 2

resulting in $\Sigma(n - d_j) = 8$.

Therefore,

$$\sigma N_B = 8$$

Next calculation is N_V:

$$\sigma N_V = (-1)^0 \left[\sum_{i=1}^{1} (3 - 2) \cdot 2^2 + {}^1 \right] = 8$$

where $M_o = 1$, $h = 0$.

Now

$$S_C = \frac{2^m \cdot \sigma N_B}{\sigma N_V}$$

Letting m = 1 we have

$$S_C = \frac{2 \cdot 8}{8} = 2$$

Repeating these calculations for all mappings of the basic set, we obtain several scores associated with each mapping from which we select the lowest scoring mappings.

In the above paragraph we select (3) as our first choice and sequentially check the other mappings with the first selection and find that (5) is a likely selection for our second mapping.

The result is given in tabular form below:

Y_3 \ Y_1Y_2	00	01	11	10
0		C	B	D
1		G	A	F

FIGURE 2.7: STATE ASSIGNMENT TABLE FOR SM3

For the third choice we have two possible ways of completing the assignment:

(1) {HGAF} → Y_3

(2) {EGAF} → Y_3

However, to satisfy (5), i.e.

{AGFH} → {ABCG}

we have to select (1), which completes the assignment procedure.

Y_3 \ Y_1Y_2	00	01	11	10
0	E	C	B	D
1	H	G	A	F

FIGURE 2.8: COMPLETED ASSIGNMENT OF SM3

Entering the assignment in the flowtable, we get

Present States		Inputs	
		x = 0	x = 1
A	111	111	110
B	110	101	100
C	010	101	100
D	100	001	000
E	000	000	001
F	101	011	010
G	011	111	110
H	001	010	011

FIGURE 2.9: FLOWTABLE OF SM3 WITH ASSIGNMENT

The logic equations are:

$$Y_1 = y_2$$

$$Y_2 = y_3$$

$$Y_3 = \bar{x}\,(y_2 + y_1\bar{y}_2) + \bar{x}\,(\bar{y}_1\bar{y}_2)$$

2.2 The D-M Method

The method of coding internal states of sequential machines as developed by Dolotta and McCluskey [1], is an improvement over many of the existing methods. The procedure can be programmed on

a computer, but due to its size, it becomes impractical for sequential machines with more than 16 states, even for computers. However, since the majority of sequential machines encountered in a practical situation usually have less than 16 states, the method presented does not exhibit a very serious restriction.

We now turn our attention to the development of the method.

Consider the flow table of SM given in table 2.10:

Present States	Inputs			
	00	01	11	10
1	3	1	3	2
2	4	2	3	3
3	2	4	4	4
4	2	1	4	4

FIGURE 2.10: FLOWTABLE OF SM4

The SM is fully specified and the inputs x_1 and x_2 have been pre-determined. With four internal states, one will need two secondary variables y_1 and y_2 in order to assign distinct codes to each state. It has been shown [10] that there are exactly three distinct ways of coding this table, corresponding to the following three codes:

(a)

y1	y2
0	0
0	1
1	0
1	1

(b)

y1	y2
0	0
0	1
1	1
1	0

(c)

y1	y2
0	0
1	1
0	1
1	0

Hence, only three columns can be used to code a four state machine. These are:

(1)	(2)	(3)
0	0	0
0	1	1
1	0	1
1	1	0

and are called codable columns.

Definition 2.05: A codable column for a k-row flow table is any column of 0's and 1's with the following properties:

(a) has length k

(b) the first element is a zero

(c) has at most 2^{n-1} ones

(d) has at most 2^{n-1} zeros

Clearly then, we have three possible different assignments or evaluating

$$A(k) = \frac{(2^n - 1)!}{(2^n - k)!\, n!}$$, where k is the number of states

as shown in section 1,

$$A(4) = \frac{(2^2 - 1)!}{(2^2 - 4)!\, 2!} = \frac{3!}{1!2!} = 3$$

Next, consider the codable columns as a matrix and associate the rows of the matrix with the states of the sequential machine. For SM4 we obtain the following relations:

States	Matrix
1	000
2	011
3	101
4	110

This matrix is called the Base Matrix. Similarly, by constructing matrices of the next state columns with the appropriate rows of the base matrix, a mapping is obtained from the present states, the base matrix rows, to the next states which correspond to the four columns of the flow table. Thus we can determine what each

States	Base Matrix	$\bar{x}_1\bar{x}_2$ Matrix	$\bar{x}_1 x_2$ Matrix	$x_1 x_2$ Matrix	$x_1\bar{x}_2$ Matrix
1	000	101	000	101	011
2	011	110	011	101	101
3	101	011	110	110	110
4	110	011	000	110	110

FIGURE 2.11: BASE MATRIX MAPPINGS FOR SM4

of the codable columns maps into each of the columns of the state table. This mapping provides us with a method for selecting the best coding and to facilitate this selection we construct a scoring array.

Code the column entry in octal numbers reading from bottom to top. Convert to octal label, then add each column and form an array from the sums of columns. A column containing a topmost one is complemented and the score of the column is identified with a minus sign.

Octal Labels of Base Matrix	$\bar{x}_1\bar{x}_2$ Matrix	$\bar{x}_1 x_2$ Matrix	$x_1 x_2$ Matrix	$x_1\bar{x}_2$ Matrix
3	-3	2	-0	7
5	7	6	3	-4
6	-4	4	-3	-3

FIGURE 2.12: SCORING ARRAY FOR SM4

It is obvious that if the row with base entry 3 is chosen for coding, then the next state equation will contain a complemented variable of its present state.

Now consider the 0- entry in that row. Since it is identified by a minus sign, we know immediately that all entries are 1's, and the corresponding term in the next state equation is independent of the present state variables. Hence, by assigning different weights to desirable entries and adding these, we obtain different scores of which we select the highest score.

Assign the following octal scores to the different entries:

20 for an all-zero entry

10 for an all-one entry

30 for adjacency

4 for identity with base entry

4 for complement of base entry

The scoring array for the illustrative example is shown in Figure 2.13.

Octal Labels of Base Matrix	$\bar{x}_1\bar{x}_2$ Matrix	$\bar{x}_1x_2$ Matrix	x_1x_2 Matrix	$x_1\bar{x}_2$ Matrix	Score 1	Score 2
3	-3	2	-0	7	14	34
5	7	6	3	-4	0	34
6	-4	4	-3	-3	20	-

FIGURE 2.13: SCORING ARRAY OF SM4

For our first variable we chose the entry 6 of the base matrix since it has the highest score. Before we make a selection for the second variable we must re-evaluate the scoring method, since an already chosen base entry will influence the next selection. Therefore, assign a score of 4 to the occurrence of an already chosen entry (or its complement) as an entry in the scoring array. Another important feature relates to the fact, that the more the identical terms appear in all expressions, the more economical is its implementation. Thus, we are looking for identical entries in the rows under consideration. We will assign an octal score of 20 for this feature of row similarities.

Lastly, the selected columns must be mutually consistent.

From figure 2.13 we see that both entries under score #2 are equal, so either one will be selectable provided all other requirements are fulfilled also.

On selection of base rows 6 and 3, the following coded flowtable evolves.

Present States	Inputs			
y_1y_2	00	01	11	10
0 0	11	00	11	10
1 0	01	10	11	11
1 1	10	01	01	01
0 1	10	00	01	01

FIGURE 2.14: CODED FLOW TABLE OF SM4

The resulting two-level Boolean expressions are:

$$Y_1 = \bar{x}_1\bar{x}_2y_2 + \bar{x}_1\bar{x}_2\bar{y}_1 + x_2y_1\bar{y}_2 + x_1\bar{y}_2$$

$$Y_2 = \bar{x}_1\bar{x}_2\bar{y}_2 + x_2y_1y_2 + x_1x_2 + x_1y_1 + x_1y_2$$

which need 11 gates for their implementation.

2.3 Assignment Algorithm by Schneider

In this section we consider the state assignment algorithm developed by M.I. Schneider[11]. There are three basic concepts which are fundamental to the algorithm:

(1) column adjacent groups

(2) row adjacent groups

and (3) output adjacent groups

All adjacency groups are listed, along with the frequency of occurrence of each type, and a score is derived. An ordered list of adjacencies is formed, starting with the highest score, and codes assigned in order of decreasing rank. This procedure continues until a valid assignment has been obtained, without destroying any of the higher ranked adjacency groups already satisfied. The results are quite good and the method is applicable to large machines as well.

Let us consider five definitions which are basic to the algorithm:

Definition 2.20:

Groups of states containing all states that go to the same next state for a particular input excitation will be called column groups.

Definition 2.21:

Groups of different states all of whose elements are members of a single column group and which contain 2^j states ($1 \leq j \leq n-1$) will be known as column adjacent groups.

Definition 2.22:

Pairs of states, that are next state entries for the same internal state and that are specified by input codes identical in all but one bit, will be defined as row adjacent groups.

Definition 2.23:

Groups of states containing all states, that, for a specified input excitation, have a "1" bit corresponding to a particular output line will be called output groups.

Definition 2.24:

Groups of different states, all of whose elements are members of a single output group and which contain 2^j states ($1 \leqslant j \leqslant n-1$) will be known as output adjacent groups.

The column adjacent group is the same as Armstrong's type II adjacency if j = 1, and the row adjacent group is the same as Armstrong's type I adjacency. The output group definition was formulated by Schneider.

Applying the procedure:

(i) Find:

(1) all column groups

(2) all column adjacent groups

(3) all output groups

(4) all output adjacent groups

(5) all row adjacent pairs

(ii) Compute:

(6) an adjacency total

(7) a score

(iii) Order:

(8) the adjacent groups by descending score values

(iv) Assign:

(9) codes to states

(v) Complement:

(10) code bits of the derived assignment

In connection with steps 6 and 7 one has two formulae to compute:

(a) $A_t = a_1 C_a + a_2 R_a + a_3 O_a$

(b) $S_c = b_j A_t$

where C_a = number of occurrences as column adjacencies

R_a = number of occurrences as row adjacencies

O_a = number of output adjacencies

A_t = adjacency total

S_c = Score

This necessitates the evaluation of four constants, namely: a_1, a_2, a_3 and b_j where $1<j<2^n$. Schneider determined experimentally these terms to be

$$a_1 = \frac{n}{2} = 1.5$$

$$a_2 = \frac{3n + 2m - 2}{5n + 4m - 1} \cdot \frac{n}{2} = 0.75$$

$$a_3 = 1$$

by considering the average numbers of minterms that can be merged when certain conditions, i.e. column adjacencies, row adjacencies, and output adjacencies, are satisfied. Next consider the constant b_j. The only useful relation given is, that

$$b_j + 1 \geq b_j$$

implying that the adjacent groups with the greater number of next state entries have the greater co-efficient. This is reasonable, since the selection of the adjacent group with the greater number of states, result in more minterm being merged into a single for some Y_i or Z_j function.

2.4 Assignment Approach by Hartmanis and Stearns

The state assignment approach of Hartmanis and Stearns [6] is based on selecting codes for the internal states of a machine in such a manner that each variable depends upon the smallest subset of the variables. The concept of partitions with substitution property and partition pairs are developed.

Definition 2.25:

A partition τ on the set of states of a sequential machine SM has the substitution property (SP) if for any two states S_i and S_j in the same block of τ and any input I, the sates (S_i, I) and (S_j, I) are again contained in a common block of τ, i.e.

$$\delta : \{S_i, I\} \rightarrow \{S_k\}$$

$$\delta : \{S_j, I\} \rightarrow \{S_\ell\}$$

where $S_k, S_\ell \varepsilon (B)_\tau$

Definition 2.26:

A partition pair (τ, τ^1) on the states of a sequential machine SM is an ordered pair of partitions on the set of states such that, if S_i and S_j belong to the same block of τ, then for each input I, (S_i, I) and (S_j, I) are in the same block of τ^1.

$$\delta : \{S_i, I\} \rightarrow \{S_k\}$$

$$\delta : \{S_j, I\} \rightarrow \{S_\ell\}$$

where $S_k, S_\ell \varepsilon (B)_{\tau^1}$

If $\tau = \tau^1$, then the definition becomes that of a partition with the substitution property. For both cases, methods for generating these partitions and the algebra for manipulating them are developed. Unfortunately, there is only a small subset of sequential machines where these methods yield a solution. When it does, the results are quite good and exhibit the characteristic decomposition into segments as shown in figures 2.41 and 2.42. For partition pair (τ, τ^1) the

segment dependency manifests itself in cross-coupling of inputs and outputs as shown in figure 2.43.

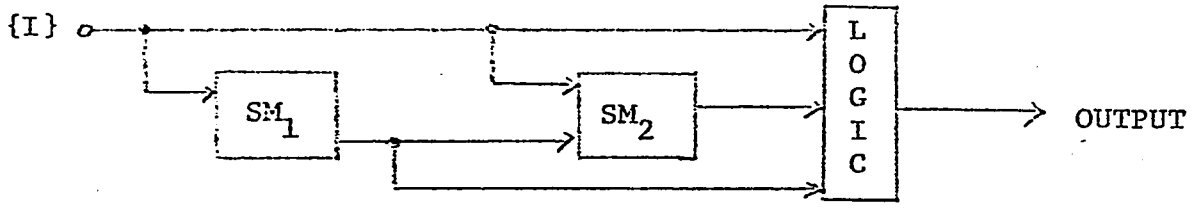

FIGURE 2.41: SERIAL DECOMPOSITION OF SM INTO SM_1 AND SM_2

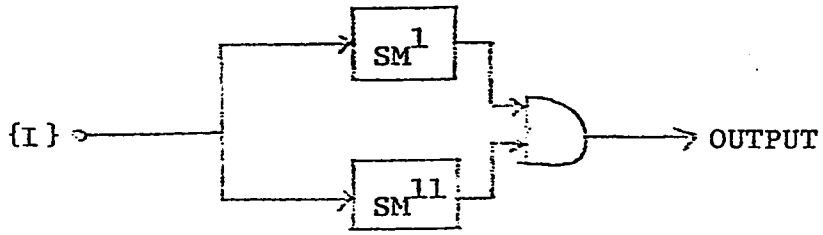

FIGURE 2.42: PARALLEL DECOMPOSITION OF SM INTO SM^1 AND SM^{11}

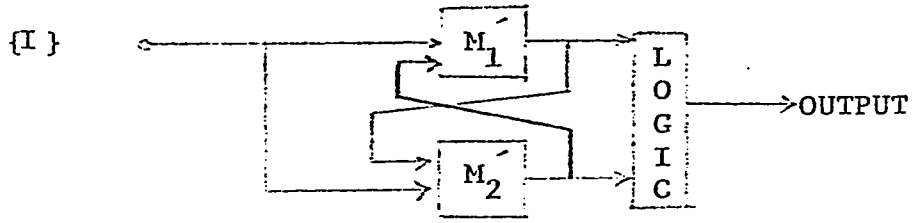

FIGURE 2.43: TYPICAL DECOMPOSITION OF SM INTO SM_1 AND SM_2 WITH PARTITION PAIRS (τ, τ^1)

III REALIZATION WITH RS FLIP-FLOP MEMORY ELEMENTS

3.1 Introduction

The most important memory element in sequential switching systems is the flip-flop. In the preceding chapter we discussed the secondary state assignment problem utilizing delay type memory elements. The delay memory elements have many advantages, and usually, simplify the assignment problem considerably. However, the majority of digital machines utilize flip-flops, and in general, they have one to three inputs and two outputs. The number of inputs depends on the particular type of flip-flop as shown in Figure 3.1

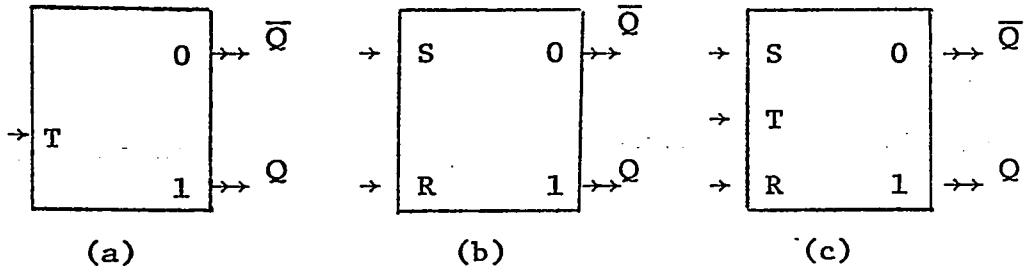

FIGURE 3.1: THREE TYPES OF FLIP-FLOPS

(a) Trigger Flip-Flop

(b) RS Flip-Flop

(c) RST Flip-Flop

We shall be only concerned with the RS and T flip-flops in the succeeding discussion, and hence, terminate any further analysis of the RST flip-flop. For the RS and T flip-flop, however, we will develope a new secondary assignment technique, which will result in some simplified circuit realization. This simplification usually manifests itself in reduced cost, fewer circuit interconnections and better reliability. However, the technique presented here has also some of the "universal" short-comings; it applies only to small set of circuits, and it gives no indication of how optimum the produced codings are. On the other hand, the technique is systematic and can be readily programmed on a computer.

3.2 The RS Flip-Flop

First let us consider the operation of the RS flip-flop. The RS flip-flop, shown in figure 3.1 (b), has two inputs R = {0,1} (reset) and S = {0,1} (set) and two outputs $\overline{Q}$ = {1,0} and Q = {0,1}. If we are discussing more than one RS-FF (FF for flip-flop) we shall index them with subscripts and refer to the i^{th} FF as FF_i with inputs R_i and S_i and outputs $\tilde{Q}_i$, where i is an integer and $\tilde{Q}_i$ equals either Q_i or $\overline{Q}_i$. The flow table of an RS-FF is shown in figure 3.2.

Present State	Inputs			
	RS			
Q	00	01	11	10
0	0	1*	N.A.	0
1	0/1	1	N.A.	0*

FIGURE 3.2: RS-FF FLOW TABLE

(N.A. - NOT ALLOWED)

From the RS-FF flow table we conclude that the RS-FF is a 2 state sequential machine with

1) $\{Q\} = \{0,1\}$ - State set

2) $\{X\} = \{0,1\} \times \{0,1\}$ - Input set

3) $\{O\} = \{0,1\}$ - Output set

Let us now consider the two inputs R and S separately and obtain the corresponding flowtable of each input. Since the input R = 1 and S = 1 is not allowed we can immediately dispense with the 11 column. For R = 0 and S = 1 the RS-FF is under control of the S input and conversely, if R = 1 and S = 0 the RS-FF is under control of the R input; hence, we can disregard the respective column for which the input is 0. This provides us with a flowtable for input R or S only.

Present States	Inputs	
	R	
Q	0	1
0	0	0
1	1	0

FIGURE 3.3: R INPUT ONLY - FF FLOWTABLE

Present States	Inputs	
	S	
Q	0	1
0	0	1
1	1	1

FIGURE 3.4: S INPUT ONLY -FF FLOWTABLE

For the R input only we can write the following excitation equations:

$$\text{RESET} = QR + \overline{Q}R$$

$$= R$$

But surely, if the FF is already in the (reset) "0" state, then an input on the R lead has no effect on the FF whatsoever; hence, this condition can be treated as a "don't care" condition, and the only necessary excitation of the R input is the state change of $Q_i \rightarrow \overline{Q}_i$. Incorporating the don't care conditions in the flowtable of figure 3.3, one obtains the following flowtable.

Present States	Inputs	
	R	
Q	0	1
0	∅	∅
1	1	0

FIGURE 3.5: MODIFIED R ONLY -FF FLOWTABLE

The ∅ indicates the don't care conditions, appearing in the next state column whenever

$$\delta: \{q_i, x\} \rightarrow \{q_i\}$$

for any input x. In particular

$$\delta: \{0, \tilde{R}\} \rightarrow \{0\}$$

where $R = R$ or $\overline{R}$. This shows that the only necessary input to Reset the RS-FF is required when the FF is in a set state and any other time the excitation of the Reset input does not alter the state of the FF, and is therefore optional. Similarly, we can show that for the setting of a FF we require only

$$S_{FF} = S\overline{Q} + SQ$$

or $$S_{FF} = S$$

by discarding the don't care condition.

This discussion leads one to believe that realization of Sequential Machines with flip-flops is a special case, requiring specific techniques in order to obtain "best" realization. Harlow and Coates[2] already demonstrated that this is certainly true for the realization techniques with Partition Algebra as developed by Hartmanis and Stearns[6]. And indeed, one can show that although the Dolotta-McCluskey[1] method produces an acceptable secondary assignement for delay memory elements, this may not be necessarily true for flip-flop memory elements.

3.3 The Scoring Approach

Consider the Base Matrix mappings for SM4 as shown in Figure 2.11 and repeated here in figure 3.6.

States	Base Matrix	$\bar{x}_1\bar{x}_2$	$\bar{x}_1 x_2$	$x_1 x_2$	$x_1\bar{x}_2$
1	000	101	000	101	011
2	011	110	011	101	101
3	101	011	110	110	110
4	110	011	000	110	110

FIGURE 3.6: BASE MATRIX AND INPUT MATRIX MAPPINGS FOR SM4

The evaluation of these mappings resulted in the coded flow table below.

Present States		Inputs							
		x_1	x_2						
y_1	y_2	0	0	0	1	1	1	1	0
0	0	1	1	0	0	1	1	0	1
0	1	1	0	0	1	1	1	1	1
1	1	0	1	1	0	1	0	1	0
1	0	0	1	0	0	1	0	1	0

Y_1Y_2

FIGURE 3.7: CODED FLOW TABLE OF SM4

The expressions for RS-FF realization are as follows:

$$S_1 = \bar{x}_1\bar{x}_2\bar{y}_1 + x_1x_2 + x_1y_2$$

$$R_1 = \bar{x}_1\bar{x}_2y_1 + \bar{x}_1y_1\bar{y}_2$$

$$S_2 = \bar{x}_1\bar{x}_2\bar{y}_2 + x_2\bar{y}_1y_2 + x_1\bar{y}_1$$

$$R_2 = \bar{x}_1\bar{x}_2\bar{y}_1y_2 + x_2y_1 + x_1y_2$$

This gives a total of 14 gates and 27 literals. This may be compared with the previously derived count of 11 gates and 23 literals for realization with delay elements.

Next consider the Base Matrix Mapping for SM4 in the context of the preceding discussion by developing two separate Base Matrix Mappings: one for the setting function, and one for the resetting function, i.e. the S-Base Matrix Mapping and R-Base Matrix Mapping respectively.

Definition 3.1:

A sequential machine flow table generated from the Base Matrix entries under input set {X} is called an S-Base Matrix if don't care conditions

$$\delta: \{Q, \tilde{S}\} \rightarrow \{Q\}$$

have been identified.

Definition 3.2:

A sequential machine flow table generated from the Base Matrix entries under input set {X} is called an R-Base Matrix if the don't care conditions

$$\delta: \{\overline{Q}, \tilde{R}\} \rightarrow \{\overline{Q}\}$$

have been identified.

It should be clear, that for the S-Base Matrix the don't care condition is shown as a 1 and for the R-Base Matrix as a $\emptyset$. Obviously the two Matrices can be combined.

Definition 3.3:

An I-Base Matrix is the superposition of the S-Base Matrix and the R-Base Matrix.

Base Matrix	I - Base Matrix			
	$\bar{x}_1\bar{x}_2$	$\bar{x}_1x_2$	x_1x_2	$x_1\bar{x}_2$
000	1Ø1	ØØØ	1Ø1	Ø11
011	1*1*0	Ø*1**1*	10*1*	10*1*
101	01*1*	110	*1*10	*1*10
110	Ø*1*1	00Ø	*1**1*Ø	*1**1*Ø

FIGURE 3.8: THE COMBINED MACHINE MATRIX OF SM4

Separating the information into two distinct tables, one for the setting function, and the other for the resetting function yields the following tables.

Base Matrix	Inputs			
	$\bar{x}_1\bar{x}_2$	$\bar{x}_1x_2$	x_1x_2	$x_1\bar{x}_2$
000	101	000	101	011
011	1*1*0	0*1**1*	10*1*	10*1*
101	01*1*	*1*10	*1*10	*1*10
110	0*1*1	000	*1**1*0	*1**1*0

FIGURE 3.9: THE S-BASE MATRIX TABLE FOR SM4

Here we have indicated the condition

$$\delta: \{FF_k\ (q_i),\ x_j\} \rightarrow \{FF_k\ (q_i)\}$$

where the particular FF_k remaining in its state is indicated by a stroke ($\not{1}$) which is interpreted as a don't care condition. Similarly, we formulate the R-Base Matrix by considering only the information required to derive the resetting function for the particular FF's, i.e. we consider only the 0's of the table.

Base Matrix	Inputs			
	$\bar{x}_1\bar{x}_2$	$\bar{x}_1x_2$	x_1x_2	$x_1\bar{x}_2$
000	1Ø1	ØØØ	1Ø1	Ø11
011	110	Ø11	101	101
101	011	110	110	110
110	011	00Ø	110	110

FIGURE 3.10: THE R-BASE MATRIX TABLE FOR MS4

Again we indicate the condition of

$$\delta: \{FF_k\ (q_i),\ x_j\} \rightarrow \{FF_k\ (q_i)\}$$

by placing a stroke through the variable associated with that particular FF_k which does not change state in the transition. This leaves us now at the point of formulating a procedure for evaluating the two tables, particularly, to identify features for which the final realization is facilitated. There are certain

features which are more desirable than others and should be given precedence. To make the selection process easier, we will assign a numerical score to each feature and weigh it in accordance with the desirability. The desirable properties and the associated scoring are as follows:

Condition	Score
An all 0 or $\emptyset$ or $\not{1}$ columns entry	20
An all 1 column entry	10
An all 1 row entry	10*
For adjacency columns	10
For identity with base column	4
For complement of base column	4

* provided the # of input subsets are greater than 2

In each case we consider the don't care entries as part of the column and use it to achieve the best possible score. If we are not interested in hand calculations, there is no need to develop a particular labelling scheme but simply compare the I-Base Matrix entries (IM_{ij}) with the Base Matrix entries (BM_{ij}). In each pass we use the don't care entries of a particular (IM_{ij}) to give the best results in relation to its (BM_{ij}) entries. After the first selection the process is repeated and at this time the don't cares may be changed in order to achieve a higher score in consideration to the already selected BM column.

3.4 Hand Assignment Technique

For hand assignments we can generate a Scoring Array by assigning binary weights to the relative position of the 1 entries in the Base Matrix and adding to a column total. For every column we have at least two sums, one will be the actual calculated and the other will be the difference between the calculated and the total obtainable. The significance of the latter is, that this relates to the negated variable.

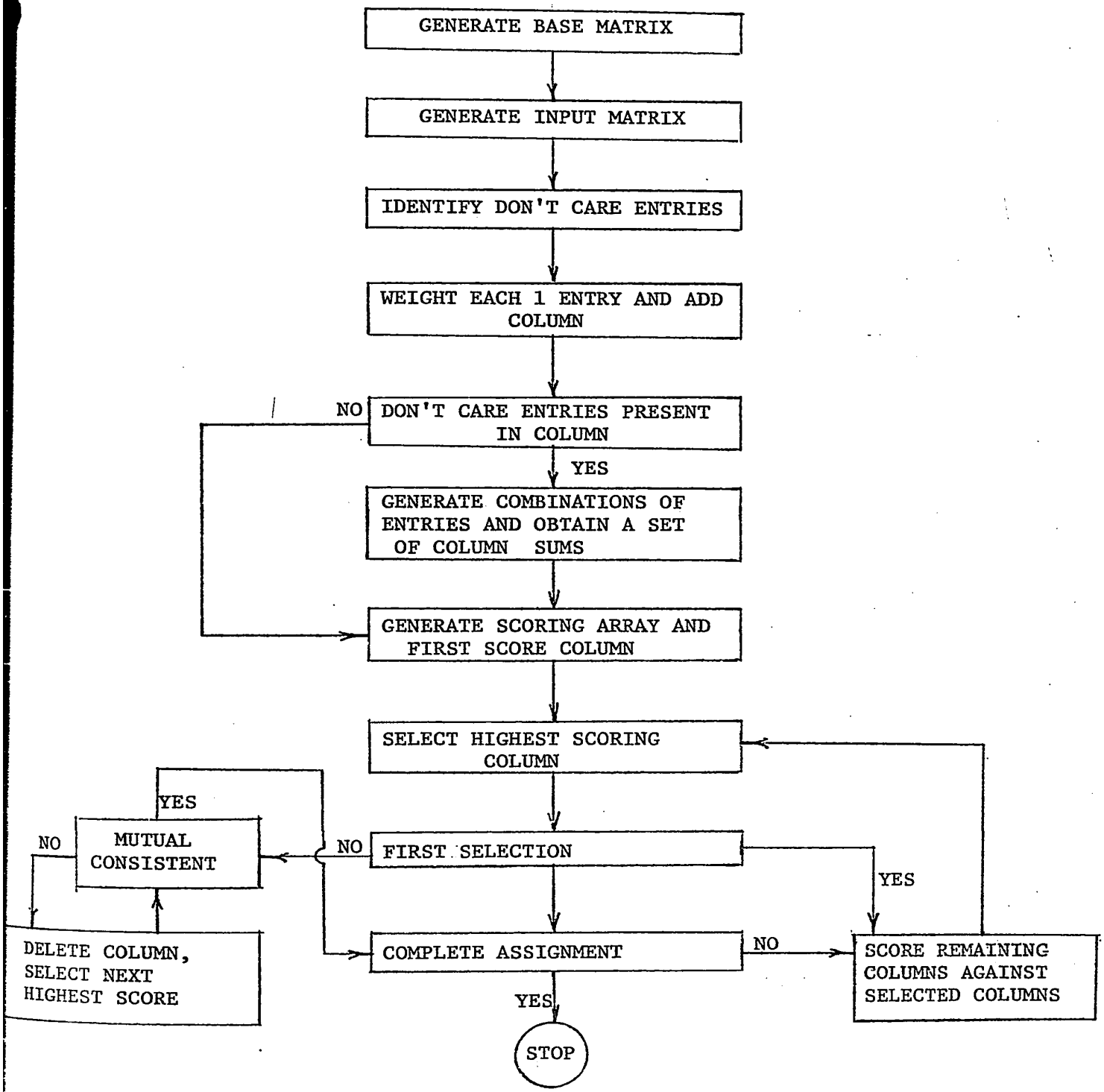

FIGURE 3.11: THE HAND ASSIGNMENT ALGORITHM

Example:

Weight	Base Matrix
1	000
2	011
4	101

Actual calculated	426
Difference	351
Total	777

Whenever the matrix contains don't care entries, combinations are generated using the don't cares. The set of sums so generated can then be used for comparison with the Base Matrix sums and the most advantageous combination can be selected.

Example:

Base Matrix	S-Base Matrix
000	011
011	011
101	000
426	011
351	033

The second set of figures in the S-Base Matrix section, i.e. 3, is generated by the don't care entry in position 2 from the top which has a weight of 2.

Having thus evaluated each particular column we can now rearrange this information into a scoring array.

(BM) Base Matrix	(SM) S-Base Matrix	Score
4,3	0	20
2,5	1,3	0
6,1	1,3	4

Obviously, the hand assignment relates very closely to the computer algorithm, but like so many others, the procedure becomes long and cumbersome when the Sequential Machine has more than 8 states.

3.5 Scoring Applied

The computer scoring array is an (SCM) Scoring Matrix of dimension I x C (I = Input subcubes and C = # of Columns of Base Matrix). Each element of SCM_{ic} is the accumulated score for that particular operation which, when added columnwise, results in the total score for that particular Base Column.

For SM4, the SCM has the dimension of 4 x 3 and would accumulate the following scores from the S-Base Matrix Table:

$SCM_{1,1}$ = 4 - for complement of base columns

$SCM_{2,1}$ = 20 - for all {0,1} entries

$SCM_{3,1}$ = 10 - for all {1,1} entries

$SCM_{4,1}$ = 0 - no established relationship

Adding the scores obtained from the R-Base Matrix Table:

$SCM_{1,1}$ = 4 - for identity with base

$SCM_{2,1}$ = 0 - no established relationship

$SCM_{3,1}$ = 20 - for all {0} entries

$SCM_{4,1}$ = 20 - for all {0,1} entries

giving the following first run score of

$SCM_{1,1}$ = 8

$SCM_{2,1}$ = 20

$SCM_{3,1}$ = 30

$SCM_{4,1}$ = 20

and $\Sigma SCM_{k,1} = 78, \quad k = 1,..,4$

Completing the SCM_{ic} for SM4 as described above gives

$$SCM_{i,c} = \begin{bmatrix} 8 & 30 & 4 \\ 20 & 10 & 30 \\ 30 & 10 & 20 \\ 20 & 4 & 0 \end{bmatrix}$$

$$\Sigma SCM_{i,c} = \begin{bmatrix} 78 & 54 & 54 \end{bmatrix}$$

Adding the score for an all 1 row entry of 10 to $SMC_{1,2}$ results in the final array of

$$\Sigma SMC_{i,c} = \begin{bmatrix} 78 & 64 & 54 \end{bmatrix}$$

Since $\Sigma SCM_{i,1}$ has the highest score, the first column of the Base Matrix is our first choice. With the selection of the first variable Y_1, we review the scoring by considering the Y_1 with respect to the remaining choices. Any additional score is added to the respective columns in $SCM_{i,c}$ which has now dimension (I x C-1). Continuing with the second scoring run we obtain,

$$SMC_{i,c-1} = \begin{bmatrix} 34 & 4 \\ 10 & 34 \\ 18 & 28 \\ 4 & 8 \\ 10 & - \end{bmatrix}$$

$$\Sigma SCM_{i,c-1} = \begin{bmatrix} 76 & 74 \end{bmatrix}$$

Obviously, our next selection is the second column of the Base Matrix. Now we are ready to write the logic equations and observe the results of the scoring procedures.

Present States		Inputs							
		x_1x_2							
y_1y_2		0	0	0	1	1	1	1	0
0	0	1	ø	ø	0	1	ø	ø	1
0	1	1	1	ø	1	1	0	1	0
1	0	0	1	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1

Y_1Y_2

FIGURE 3:10a: | FLOW TABLE OF SM4

$$FF_{1S} = \bar{y}_1\bar{x}_1\bar{x}_2 + x_1x_2 + \underline{y_1y_2x_1}$$

$$FF_{2S} = y_1\bar{y}_2 + \bar{y}_2x_1\bar{x}_2$$

$$FF_{1R} = y_1\bar{x}_1\bar{x}_2 + \underline{y_1y_2x_1x_2}$$

$$FF_{2R} = \underline{y_1y2_x1} + \underline{y_1y_2x_1x_2}$$

Underlined products are identical and hence the gates can be shared. This leaves us with a gate count of 11 and 20 literals.

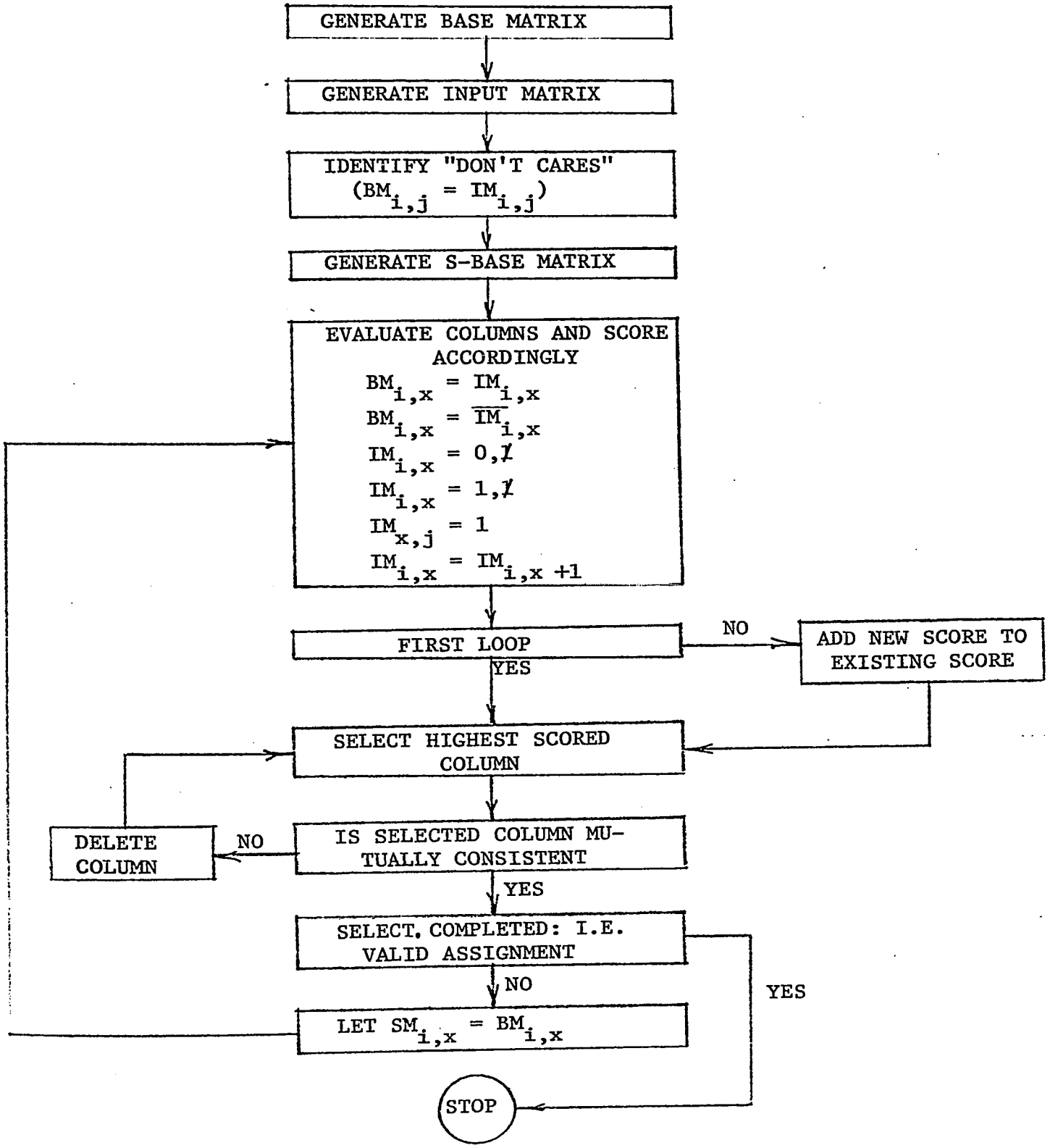

FIGURE 3.12: The Assignment Algorithm for RS-FF

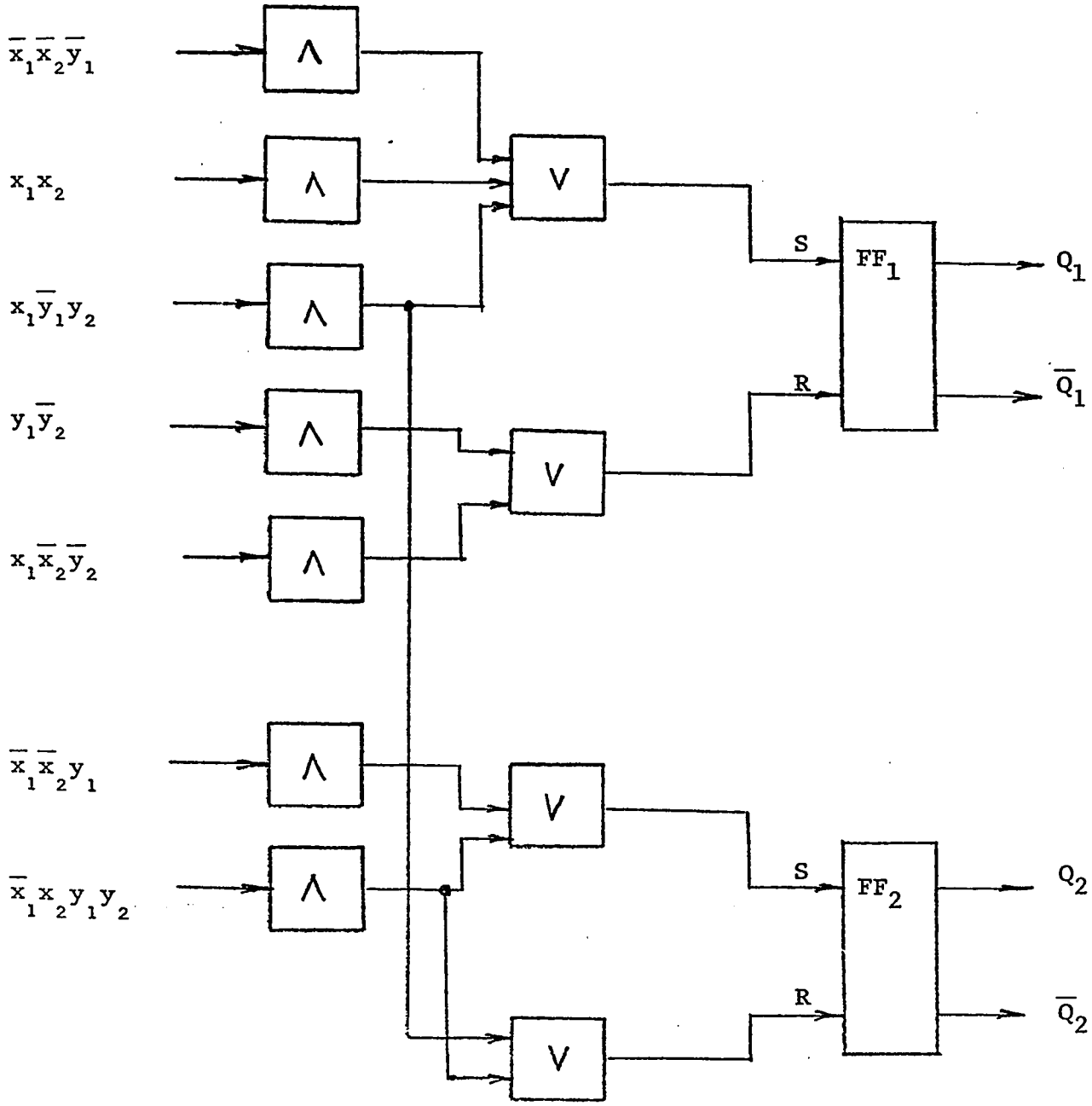

FIGURE 3.13: Circuit of SM4

Next we will apply the Hand Algorithm to SM4.

(1) Generate Base Matrix

000

011

101

110

(2) Generate Input Matrix

00	01	11	10
101	000	101	011
110	011	101	101
011	110	110	110
011	000	110	110

(3) Identify don't care entries

Base Matrix	Input Matrix			
	00	01	11	10
000	1Ø1	ØØØ	1Ø1	Ø11
011	1X0	ØXX	10X	10X
101	01X	X10	X10	X10
110	0X1	00Ø	XXØ	XXØ

(4) Weight each 1 entry and add column

For set:

12	10	6		3	4	9		0	4	0		3	4	1		2	5	1
3	5	9																

For reset:

			12	0	2		8	8	4		0	2	4		0	2	4

(5) Yes

(6) For set:

BM	00	01	11	10
12 10 6	3 4 9	0 4 0	3 4 1	2 5 1
3 5 9	6 13	6	7 12 3	6 13 3
	12		15	10
	14			14

For reset:

BM	00	01	11	10
12 10 6	12 0 2	8 8 4	0 2 4	0 2 4
3 5 9		10 9 12	3 12	12
		9 5		
		11 13		

(7) Generate Scoring Array

Base	Inputs				Scores	
	00	01	11	10	I	II
12,3	3 12	0 8,10,9,11	3,7,15 0	2,6,10,14 0	34 44 Σ 78	
10,5	4,6,12,14 0	4,6 8,9	4,12 2,3	5,13 2	34 30 Σ 64	8 4 Σ 76
6,9	9,13 2	0 4,12,5,13	1,3 4,12	1,3 4,12	34 20 Σ 54	8 12 Σ74

(8) Select highest scoring column.

This is column I with a 78 score.

(9) Score remaining columns against selected column. This is score II under step #7.

(10) Select highest scoring column. This is column II with a 76 score.

(11) Mutually consistent: yes

(12) Assignment complete

Present States	Inputs			
y_1y_2	x_1x_2			
	0 0	0 1	1 1	1 0
0 0	1 0	0 0	1 0	0 1
0 1	1 1	0 1	1 0	1 0
1 0	0 1	1 1	1 1	1 1
1 1	0 1	0 0	1 1	1 1

FIGURE 3.14: FLOW TABLE OF SM4

Obviously, this is identical to the computer generated Flow Table of SM4. However, there are some very interesting features to be observed in the Scoring Array. Any entry in an input column can be combined with an adjacent input column entry, provided that the entry under consideration is smaller than the adjacent entry and be binary inclusive. Consider the first row in the Scoring Array under input $x_1 = 1$ and $x_2 = 0$. None of the entries {2,6,10,14} score, but there is still a simplification possible, because in column $x_1 = 1$ and $x_2 = 1$, the scoring entry {3} can be utilized to combine with the entry {2} in the adjacent column, and thereby reducing the literal count.

Furthermore, any entry in the same column but on different rows are equal terms. This means that the terms can be shared. Again consider column $x_1 = 1$ and $x_2 = 0$ and the entry {2} in row 1 and entry {2} in row 3. The first term appears in the setting function for FF_1 and the second term appears in the resetting function of FF_2. Hence the gates can be shared. Similarly, for input column $x_1 = 0$ and $x_2 = 1$ and the entry {8} in row 2 and entry {8} in row 4.

Although there is no scoring for common terms it is certainly possible to predict prior to deriving the logic operations how many terms can be shared.

IV REALIZATION WITH TRIGGER FLIP-FLOP MEMORY ELEMENTS

4.1 Introduction

We will now show that the previously developed scoring method for RS-FF requires only minor modification in order to be applicable to T-FF realization.

4.2 The Trigger Flip-Flop or Toggle

A T-FF is a two state sequential machine as specified in the T-FF flow table in Figure 4.1. The input terminal is denoted by T and the output terminals by Q and $\overline{Q}$. As previously discussed in connection with the RS-FF, trigger flip-flops will be also indexed with integers as $T\text{-}FF_i$, T_i, $\tilde{Q}_i$ for the i^{th} input or output respectively.

Present States	Inputs	
	0	1
0	0	1
1	1	0

4.1 THE TRIGGER FLIP-FLOP FLOW TABLE

As can be seen from the flow table, the T-FF will always change state whenever the input terminal is energized, i.e.

$$\delta : \{Q,X\} \rightarrow \{\overline{Q}\}$$

and

$$\delta : \{\overline{Q},X\} \rightarrow \{Q\}$$

This differs from the RS-FF where a set input would have no effect on the FF if the FF was already in the set state. These conditions were identified as don't cares in the SB matrix. In this context the T-FF has no don't care conditions available for possible simplification of the excitation function. Every map entry has to be covered, similar to the delay element technique. However, what appears as a don't care entry in the SB matrix for the RS-FF, becomes a no-change condition for the T-FF. This means that for every

$$\delta : \{FF_k \ (q_i), X\} \rightarrow \{FF_k \ (q_i)\}$$

Where $q_i = \tilde{Q}$, i.e. the 1 or 0 state, the TB matrix will contain a zero entry.

Definition 4.1:

A Trigger Base Matrix (TB matrix) is the resulting matrix from the logic "Not Equal" operation performed on the Base Matrix and the Input Matrix, i.e.

$$TB_{ij} \leftarrow BM_{ij} \neq IM_{ij}$$

Once this procedure has been performed the basic scoring method applies and although the TB matrix will never contain don't care entries the Assignment Algorithm for RS-FF can be effectively utilized by implementing the following modification.

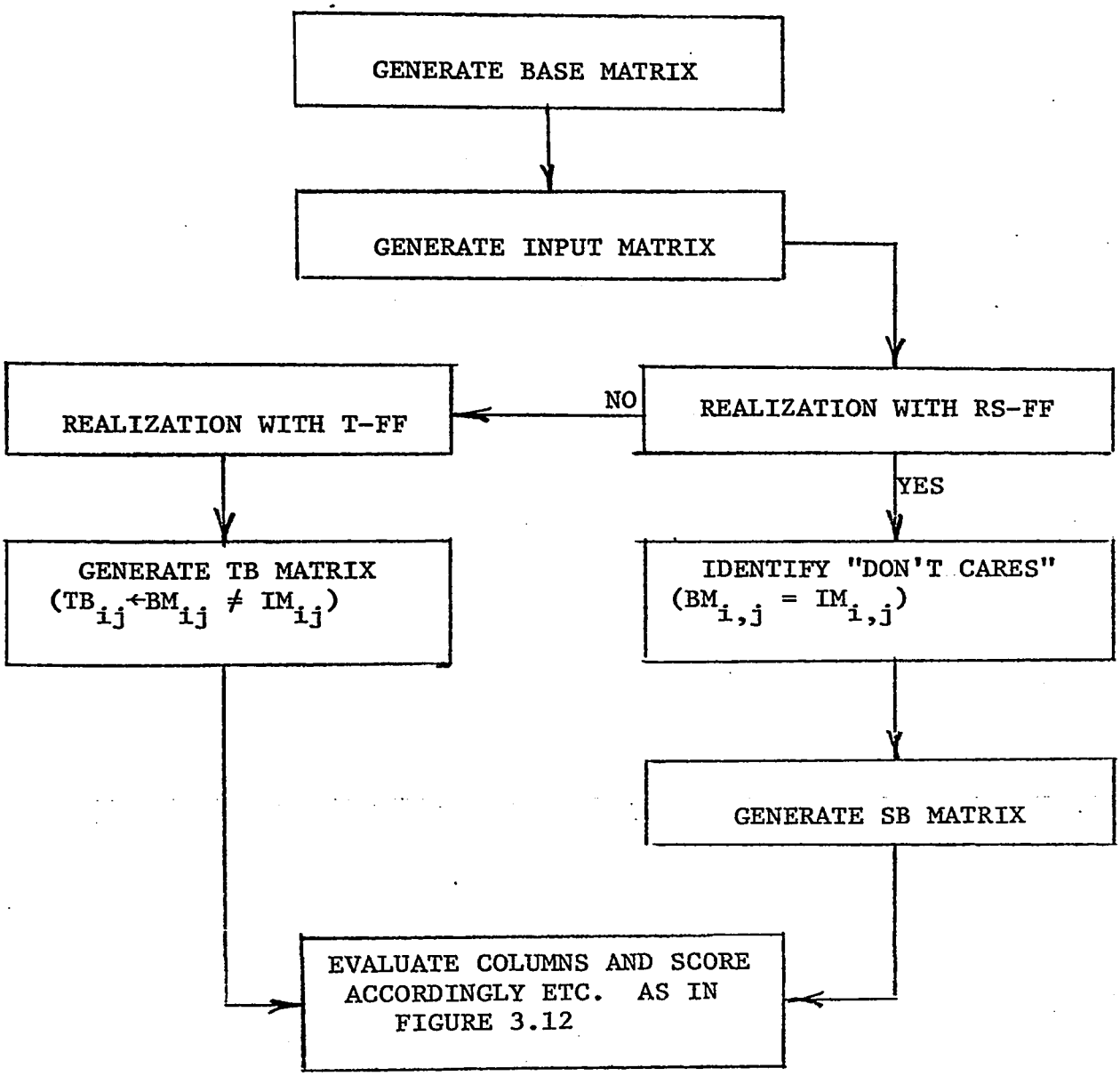

FIGURE 4.2: The Assignment Algorithm For RS-FF and T-FF

Example 4.1: Let the BM and IM matrices have the following values

$$BM = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad IM = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

then the operation

$TB_{ij} \longleftarrow BM_{ij} \neq IM_{ij}$ or $BM_{ij} \oplus IM_{ij}$, where $\oplus$ is the the exclusive OR operation, results in

$$TB = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Consider SM5 below, which has only trivial SP partitions namely {I} and {0}.

Present States	Inputs	
	0	1
1	2	3
2	2	4
3	5	1
4	5	4
5	3	5

FIGURE 4.3: FLOW TABLE OF SM5

Prior to calculating the Mm pairs for SM5, let us review a definition originally formulated in [6].

Definition 4.1:

Let τ' be a partition on the set of states of SM. Define a partition $M(\tau')$ so that $M(\tau') = \Sigma\tau_i$, where the sum is over all τ such that (τ_i,τ) is a partition pair. Define the partition $m(\tau) = \pi\tau_i$, where the product is over all τ' such that (τ,τ_i) is a partition pair. A partition pair (τ,τ_i) is an Mm pair if $\tau = M(\tau')$ and $\tau' = m(\tau)$.

Calculating the Mm pairs of SM5 we obtain:

$(T_1, T_1^1) = \{\overline{1,2}, \overline{3}, \overline{4}, \overline{5}\}\ \{\overline{3,4}, \overline{1}, \overline{2}, \overline{5}\}$

$(T_2, T_2^1) = \{\overline{1,3}, \overline{2,4}, \overline{5}\}\ \{\overline{2,5}, \overline{1,3}, \overline{4}\}$

$(T_3, T_3^1) = \{\overline{1,2,4}, \overline{3}, \overline{5}\}\ \{\overline{2,5,3,4}, \overline{1}\}$

$(T_4, T_4^1) = \{\overline{1,5}, \overline{3,4}, \overline{2}\}\ \{\overline{2,3,5}, \overline{1}, \overline{4}\}$

$(T_5, T_5^1) = \{\overline{2,3,4}, \overline{1}, \overline{5}\}\ \{\overline{2,5}, \overline{1,4}, \overline{3}\}$

$(T_6, T_6^1) = \{\overline{2,4}, \overline{1}, \overline{3}, \overline{5}\}\ \{\overline{2,5}, \overline{1}, \overline{3}, \overline{4}\}$

$(T_7, T_7^1) = \{\overline{2,5}, \overline{1}, \overline{3}, \overline{4}\}\ \{\overline{2,3}, \overline{4,5}, \overline{1}\}$

$(T_8, T_8^1) = \{\overline{3,4}, \overline{1}, \overline{2}, \overline{5}\}\ \{\overline{1,4}, \overline{2}, \overline{3}, \overline{5}\}$

$(T_9, T_9^1) = \{\overline{3,5}, \overline{1}, \overline{2}, \overline{4}\}\ \{\overline{1,3,5}, \overline{4,2}\}$

$(T_{10}, T_{10}^1) = \{\overline{4,5}, \overline{1,2}, \overline{3}\}\ \{\overline{3,4,5}, \overline{1}, \overline{2}\}$

$(T_{11}, T_{11}^{1'}) = \{\overline{1,2,4,5},\ \overline{3}\}\quad \{\overline{1},\ \overline{2,3,4,5}\}$

$(T_{12}, T_{12}^{1}) = \{\overline{1,2,3,4},\ \overline{5}\}\quad \{\overline{1,3,4},\ \overline{2,5}\}$

$(T_{13}, T_{13}^{1}) = \{\overline{1,2},\ \overline{3,4,5}\}\quad \{\overline{1,3,4,5},\ \overline{2}\}$

$(T_{14}, T_{14}^{1}) = \{\overline{1,3,5},\ \overline{2},\ \overline{4}\}\quad \{\overline{1,2,3,5},\ \overline{4}\}$

$(T_{15}, T_{15}^{1}) = \{\overline{1,5},\ \overline{2,3},\ \overline{4}\}\quad \{\overline{1,4},\ \overline{2,3,5}\}$

$(T_{16}, T_{16}^{1}) = \{\overline{2,5},\ \overline{3,4},\ \overline{1}\}\quad \{\overline{1,4,5},\ \overline{2,3}\}$

Definition 4.4:

The state partitions τ and τ' on SM are in relation $\{\tau\}\ t\ \{\tau'\}$

if

(i) τ' has two blocks

(ii) $\{\tau \cdot \tau'\}$ relation $\{\tau'\}$

(iii) For every two states s_1, s_2 such that $\tau[s_1] = \tau[s_2]$ and $\tau'[s_1] \neq \tau'[s_2]$ and for every input x, then $\tau'[\delta(s_1, x)] \neq [\delta(s_2, x)]$ when $\delta(s_1, x)$ and $\delta(s_2, x)$ are specified.

Next let us look at Harlow's [2] t-relation partitions which are

$\{\overline{1,2},\ \overline{3,4,5}\}\quad t\quad \{\overline{1,2,3,4},\ \overline{5}\}$

$\{\overline{1,2,3},\ \overline{4},\ \overline{5}\}\quad t\quad \{\overline{1,2},\ \overline{3,4,5}\}$

$\{\overline{2,3,5},\ \overline{1,4},\}\quad t\quad \{\overline{1,3,5},\ \overline{2,4}\}$

Since the t-relation partitions are specifically derived for T-FF it is obviously the assignment sought after. However, there is no systematic way of obtaining these partitions and one may spend considerable time and effort to find these partitions. Deriving the Mm pairs, on the other hand, is a straightforward procedure but once obtained, it is not obvious which of the Mm pairs will give a good T-FF assignment. This is also true for SP partitions, provided the SM has other partitions except the trivial ones.

4.3 Scoring Applied to SM5

Let us now apply the proposed method to SM5 and obtain a secondary assignment.

The Base Matrix is:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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

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

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

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

The I_1 Matrix is:

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

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

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

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

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

The I_2 Matrix is:

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

The T_1B Matrix is:

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

The T_2B Matrix is:

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

USING THE SCORING ARRAY

Row #	Base	Inputs		Scores		
		0	1	I	II	III
1	16	28	0	20	24*	-
2	8	8	2	8	8	8
3	24	20	2	8	8	8
4	4	20	5	8	8	8
5	20	8	5	4	4	4
6	12	28	7	4	12	16
7	28	0	7	24*	-	-
8	2	1	2	8	8	8
9	18	29	2	4	4	4
10	10	9	0	20	20	20
11	26	21	0	24	24	24*
12	6	21	7	4	8	8
13	14	29	5	4	4	4
14	30	1	5	4	4	4
15	22	9	7	0	4	4

FIGURE 4.4: SCORING ARRAY OF SM5

In score column I we have to choose between row 7 and 11 as both have a score of 24. The decision goes to row 7 because the weight of the base column is 28 which appears in the input 0 column twice. Should we select row 11, however, there are no other entries with

weight 26 and the next score would not change. This would bring us right back to row 7 on the second selection. As it happens, with the selection of row 7 the score column II evolves and once again one must make a decision between row 1 and 11 as both have identical weights of 24. Note that the base weight entry for row 1 is 16 which, just like row 11, will not contribute to the next score for identity with the base column.

This leads to the selection of the three high scoring rows of 7, 1, and 11. Reconstructing the flow table for SM5 with the selected columns, one obtains the following coding.

Present States	Inputs	
$y_1y_2y_3$	x = 0	x = 1
0 0 0	0 0 1	1 0 0
0 0 1	0 0 0	1 0 0
1 0 0	0 1 1	1 0 0
1 0 1	0 1 0	0 0 0
1 1 1	0 1 1	0 0 0

T_1, T_2, T_3

FIGURE 4.5: SECONDARY STATE ASSIGNMENT FOR SM5

From Figure 4.5 we obtain the excitation functions for the T- flip-flops as

$$T_1 = x\,\overline{y}_1 + x\,\overline{y}_3$$

$$T_2 = \bar{x}\, y_1$$

$$T_3 = \bar{x}\,\bar{y}_2 + \bar{x}\,\bar{y}_3$$

Let us compare this with the Harlow's derivation of the T- flip-flop functions

$$T_1^1 = \bar{x}\, y$$

$$T_2^1 = x\,\bar{y}_2 + \bar{x}\,\bar{y}_1 y_2$$

$$T_3^1 = \bar{x}\,\bar{y}_2\bar{y}_3 + \bar{x}\, y_2 y_3$$

Although there is no reduction in the number of gates, the number of literals was reduced from 13 to 10. It is interesting to note that Harlow's assignment scores significantly high in the scoring array. Had we selected row 10 instead of row 11, row 10 was the next highest score after row 11, we would have obtained Harlow's T_1^1, T_2^1 and T_3^1 functions.

4.4 Some Comparison With Delay Memory Elements

Most of the published papers in secondary state assignments primarily concern themselves with unit delay elements. As mentioned previously, this simplifies the problem considerably, but it does not lead to a "good" assignment when other memory elements are utilized. Let us demonstrate the difference.

Consider the flow table of SM4. This machine structure does not possess any useful SP or PP partitions. From the application of Dolotta and McCluskey technique we obtained the next state equations.

$$Y_1 = \bar{x}_1\bar{x}_2\bar{y}_1 + x_2y_1y_2 + x_1x_2 + x_1y_2 + x_1y_2$$

$$Y_2 = \bar{x}_1\bar{x}_2y_1 + x_2\bar{y}_1y_2 + \bar{x}_1\bar{x}_2\bar{y}_2 + x_1\bar{y}_1$$

This leads to a total of 11 gates and 23 literals.

However, these equations are only valid for unit delay element realization. For RS-FF realization one would obtain four equations, two for setting and two for re-setting the RS-FF's.

$$FF_{1S} = \bar{x}_1\bar{x}_2\bar{y}_1 + x_1x_2 + x_1y_2\bar{y}_1$$

$$FF_{1R} = \bar{x}_1\bar{x}_2y_1 + x_2\bar{x}_1y_1y_2$$

$$FF_{2S} = x_1\bar{x}_2\bar{y}_2 + y_1\bar{y}_2$$

$$FF_{2R} = \bar{x}_1x_2y_1y_2 + x_1\bar{y}_1y_2$$

These would need a total of 14 gates and 26 literals for realization. It is obvious from Figure 4.6 that no timing problems exist when the circuit is realized using the above functions.

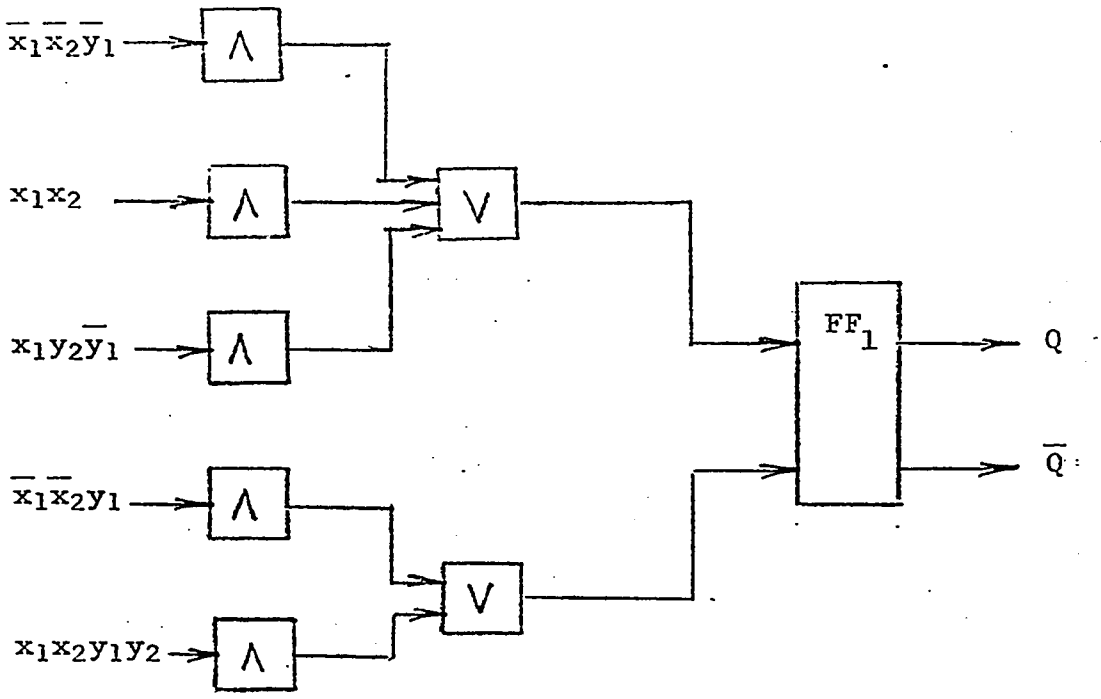

FIGURE 4.6: REALIZATION OF FUNCTIONS FF_{1S} AND FF_{1R}

There is one more alternative. We could use the equations derived for unit delay element realization, if we recognize that the relationship of

$$FF_{iS} = \overline{FF_{iR}}$$

is true for all cases, provided that there are no don't care conditions and all 1 conditions are covered. This approach introduces one additional problem, however, because the negation of the FF_{iS} function must be gated before it can be applied to the reset input of the RS-FF_{iR}. Otherwise timing and operational problems would cause the circuit to malfunction. This arrangement

increases the gate count by 2 gates per function, and consequently the total count for SM4 increases to 15 gates and 25 literals. A portion of this circuit is shown in figure 4.7.

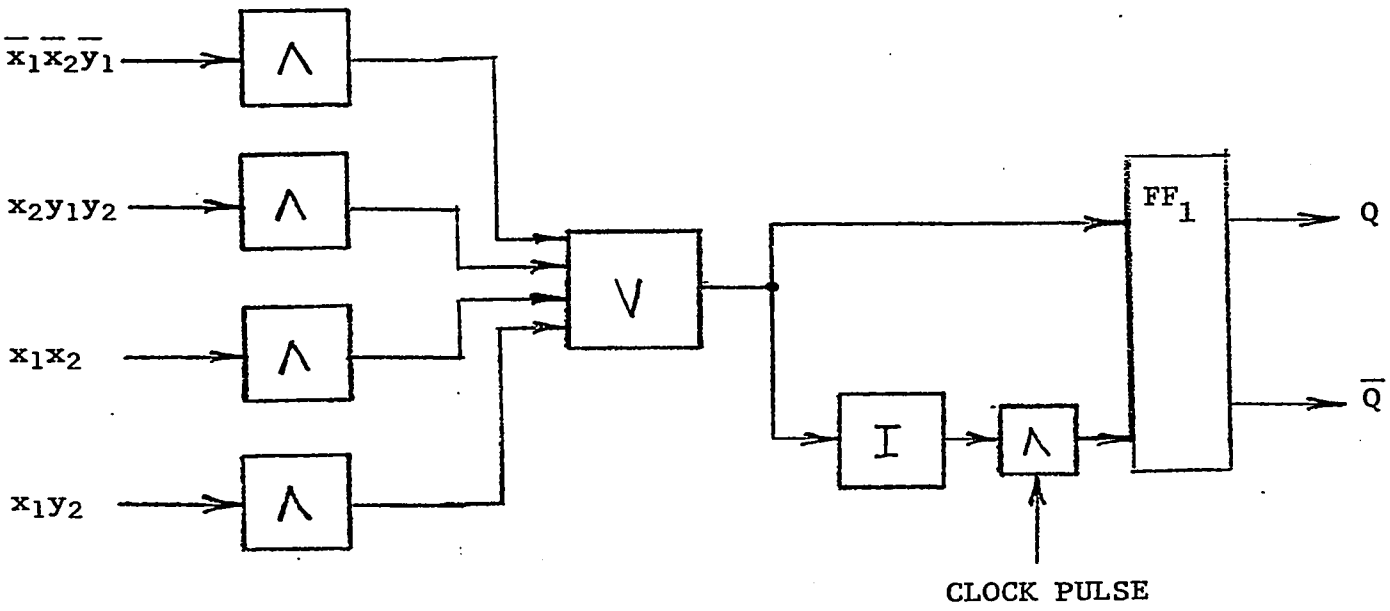

FIGURE 4.7: MODIFIED REALIZATION OF FUNCTION Y

Next let us demonstrate the difference between trigger flip-flops realizations and unit delay realizations. From the previous evaluation of SM4 we obtained the unit delay element assignment as follows:

1 → 0 0

2 → 1 0

3 → 1 1

4 → 0 1

This assignment resulted in the functions Y_1 and Y_2.

But when we derive the excitation function for the trigger flip-flops we get

$$T_1 = \bar{x}_1\bar{x}_2\bar{y}_1 + x_1\bar{y}_1\bar{y}_2 + x_1y_1y_2 + x_2y_1y_2 + \bar{x}_1\bar{x}_2\bar{y}_2$$

$$T_2 = \bar{x}_1\bar{x}_2 + x_1y_1\bar{y}_2 + x_1x_2\bar{y}_2$$

which need a total of 10 gates and 23 literals. We may compare this to the results of the proposed method, which yields the functions

$$T_1 = \bar{x}_1\bar{x}_2 + \bar{x}_1y_1y_2 + \underline{x_1y_1y_2} + x_1x_2\bar{y}_1$$

$$T_2 = y_1\bar{y}_2 + \bar{x}_1x_2y_1 + \underline{x_1y_1y_2} + x_1\bar{x}_2\bar{y}_1$$

requiring a total of 9 gates and 19 literals for implementation. Again the results are better.

V CONCLUSION AND PROBLEMS

A major objective of this study was to find an algorithm for obtaining good secondary state assignments using Flip-Flop memory elements for sequential machines. Two algorithms were presented which demonstrated the procedures to be followed for realization utilizing RS flip-flops and T flip-flops. Although the algorithms are intended to be programmed on a computer, a hand technique was also presented for small sequential machines. An APL program was written and executed on an IBM 360/67, providing the necessary information to confirm the procedure. The solution obtained from the worked problems resulted in good circuit realizations. There was no single set of rules which could be found to obtain the optimum solution for all problems.

The state-assignment algorithms presented are restricted to completely specified sequential machines. Furthermore, even the computer techniques becomes too cumbersome for large sequential machines with more than 16 states. This is probably the biggest shortcoming of this technique and should be given further consideration.

In order to apply the proposed technique to asynchronous sequential machines, one requires that each transition is accomplished either by a change of only one secondary variable or by a change of multiple secondary variables without a critical race. Neither one of these two conditions are readily identifiable without considerable changes to the proposed technique.

Earlier work on secondary state assignment techniques was primarily done for economical reasons. Since each gate consisted of several discrete components, which had to be individually fabricated and interconnected, a reduction in the number of gates represented a direct saving in cost of the circuit. In today's era of medium Scale Integration (MSI) and Large Scale Integration (LSI) the justification for obtaining "optimum" secondary state assignments has shifted from the cost reduction aspect to one of greater reliability, minimum heat dissipation and smaller chip area for a particular circuit.

BIBLIOGRAPHY

[1] T.A. Dolotta and E.J. McCluskey; The Coding of Internal States of Sequential Circuits: IEEE Trans. E.C., Vol. 13, 1964, pp. 549-563.

[2] Harlow, C. and Coates, C.L.; On the Structure of Realizations using Flip-Flop Memory Elements: Information and Control Vol. 10, 1967, pp. 159-174.

[3] Kohavi, Z.; Secondary State Assignment for Sequential Machines: IEEE Trans. E.C. June 1964, pp. 193-203.

[4] Armstrong, D.B.; On the Efficient Assignment of Internal Codes to Sequential Machines. IRE Trans. E.C., October 1962, pp. 611-622.

[5] Armstrong, D.B.; A Programmed Algorithm for Assigning Internal Codes to Sequential Machines: IRE Trans. E.C., August 1962, pp. 466-472.

[6] Hartmanis, J. and Stearns, R.E.; Algebraic Structure Theory of Sequential Machines; New York: Prentice-Hall 1966.

[7] Davis, W.A.; Contributions to Structure Theory of Sequential Machines; PHD dissertation, University of Ottawa, October 1966.

[8] Haring, D.R.; Sequential - Circuit Synthesis; State Assignment Aspects; Research Monograph No. 31, The M.I.T. Press, Cambridge, Massachusetts, 1966.

[9] Curtis, H.A.; Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-Flop Memory: Part I and Part II IEEE Trans. EC-18.

[10] Dolotta, T.A.; The Coding Problem in the Design of Switching Circuits; PHD Dissertation, Princeton University, May 1961.

[11] Schneider, M.I.; State Assignment Algorithm for Clocked Sequential Machines; Lincoln Lab Report 270, Massachusetts Institute of Technology, 1962.

[12] Kohavi, Z.; Switching and Finite Automaton Theory; New York, McGraw-Hill,1970

[13] Moore, E.F.; Gedanken - experiments on Sequential Machines; Princeton University Press, Princeton, N.J., 1956.

[14] Mealy, G.H.; A Methode for Synthesizing Sequential Circuits; B.S.T.J., Vol 34, September, 1955.

[15] Huffman, D.A.; The Synthesis of Sequential Switching Circuits; J. Franklin Inst., Vol 257, 1954.