1964

Minimum transition time state assignment methods for asynchronous sequential switching circuits

James Henry Tracey
Iowa State University

MINIMUM TRANSITION TIME STATE ASSIGNMENT METHODS FOR ASYNCHRONOUS SEQUENTIAL SWITCHING CIRCUITS

by

James Henry Tracey

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

Signature was redacted for privacy.
In Charge of Major Work

Signature was redacted for privacy.
Head of Major Department

Signature was redacted for privacy.
Dean of Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1964

TABLE OF CONTENTS

I. INTRODUCTION

A. Switching Circuits

Switching circuits are usually characterized by a "black box" as shown in Figure 1. The switching network is shown with n input lines and

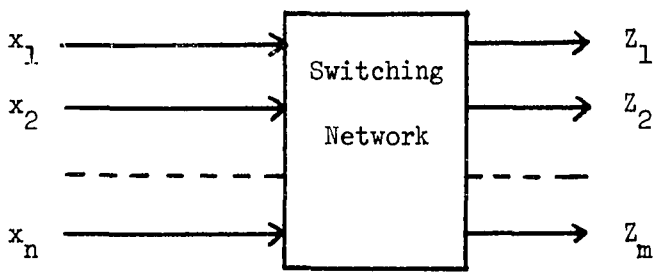


Figure 1. Block diagram of a switching circuit

m output lines. In general the input variabels $x_1, x_2, \cdots x_n$ and the output variables $Z_1, Z_2, \cdots Z_m$ may take on any finite number of values, but in this paper it will be assumed that all such variables are binary variables. The switching network of Figure 1 is described as a combinational switching network if the outputs are functions solely of the inputs. In that case one may write

$$Z_1 = f_1(x_1, x_2, \cdots x_n)$$
$$Z_2 = f_2(x_1, x_2, \cdots x_n)$$
$$\vdots$$
$$Z_m = f_m(x_1, x_2, \cdots x_n)$$

On the other hand, if the outputs depend not only on the present

input but also on past circuit states, the circuit is described as a sequential switching network. A sequential circuit is usually thought of as consisting of a combinational circuit plus feedback loops. See Figure 2. The feedback loops and their associated storage capability (usually a delay or flip-flop element) provide the necessary memory for the circuit. The feedback variables $Y_1, Y_2, \cdots Y_p$ and $y_1, y_2, \cdots y_p$ are commonly called secondary variables with the secondary excitations represented by $Y_1, Y_2, \cdots Y_p$ and the secondary states represented by $y_1, y_2, \cdots y_p$. Since the next secondary state will be the same as the present secondary excitation, it is convenient to refer to the Y's as "next state" variables and the y's as "present state"variables. The secondary circuit is said to be "stable" when the excitation is the same as the state. The setting expressions for the memory elements may be written as functions of the x's and y's. If delay is used for memory, the setting variables are the Y's themselves, and one may write

$$
\begin{aligned}
Y_1 &= g_1(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p) \\
Y_2 &= g_2(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p) \\
&\vdots \\
Y_p &= g_p(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p)
\end{aligned}
$$

If the circuit inputs and memory element inputs are gated with clock pulses the circuit is called a synchronous sequential circuit. In synchronous circuits one may imagine the clock pulses to be numbered so that the i-th input combination is the input to the circuit that is gated with the i-th clock pulse.

If no clocking is available the circuit is asynchronous. The i-th

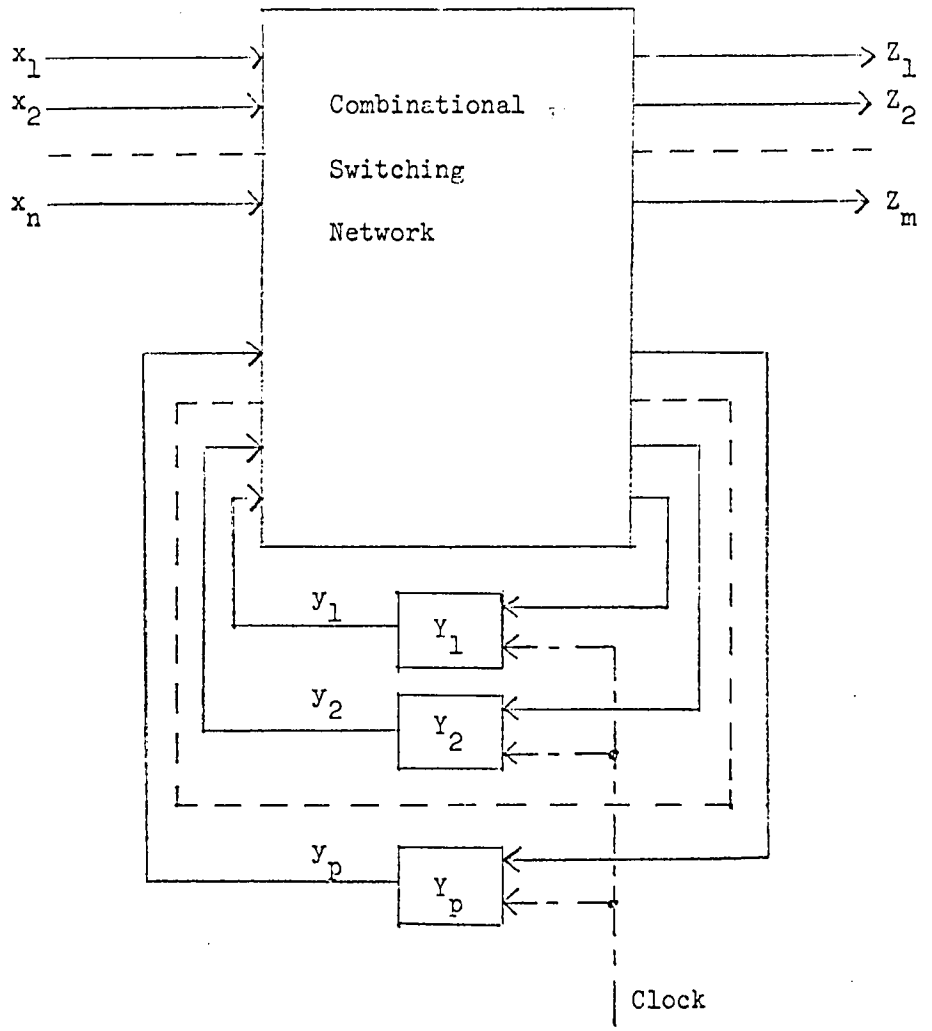


Figure 2. Model of a sequential switching circuit

input combination for an asynchronous circuit is the input after i changes in the input combination. The synchronous circuit recognizes new data each clock pulse time while the asynchronous circuit recognizes new input data only when there is a change in the data itself.

As mentioned before, the outputs of a sequential switching circuit are dependent on present input and past circuit states. For the general case then, one may write the output expressions as

$$Z_1 = h_1(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p)$$
$$Z_2 = h_2(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p)$$
$$\vdots$$
$$Z_m = h_m(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_p)$$

In terms of what has been developed thus far, the secondary state assignment problem involves coding the finite number of secondary states with combinations of $y_1, y_2, \cdots y_p$. This can be done in a trivial fashion except when constraints are placed on the assignment. In synchronous circuits the usual constraint is to make the assignment so as to minimize the cost of the combinational logic circuit. It is well known that the cost of the combinational circuit may more than double if a "poor" secondary assignment is used in place of a "good" assignment.

A primary consideration in asynchronous sequential circuits is to make a secondary assignment such that the circuit will function properly independent of variations in transmission delays of signals within the circuit. Keep in mind that clock pulses are not available here to control gating operations in the combinational circuit or in the feedback loop. Of course, the cost of the combinational circuit is also important in the

design of asynchronous sequential circuits but first consideration must be given to the elimination of what is later described as dangerous "race" conditions.

This paper is primarily concerned with secondary state assignment methods for asynchronous sequential circuits. Assignment methods will be developed to insure desired circuit action independent of variations in circuit delays. The assignment methods will also be designed so as to enable the circuit to accept data at a maximum rate.

1. Synthesis of asynchronous sequential switching circuits

In this section, an illustration of the synthesis of an asynchronous sequential switching circuit will be given. The models and techniques used here are essentially those of Huffman (5). Figure 3 is a block diagram of the circuit realization that will be referenced throughout the synthesis procedure. The model of Figure 3 is just a slight modification of that in Figure 2. Note that delays are used for memory, so the setting functions for the memory elements are the secondary excitations variables themselves.

Sequential circuit specifications are usually in the form of a word statement, list of input-output sequences or a timing diagram. The synthesis procedure can best be explained by fabricating a circuit specification and illustrating the steps leading to a final circuit synthesis. One may begin with the following specification: A sequential circuit is to have two inputs, x_1 and x_2, and one output, Z. The output, Z, is to turn on only when x_1 turns on and Z is to turn off only when x_2 turns off. Only one input variable may change state at a time.

The first step is to relate the specifications to a primitive flow

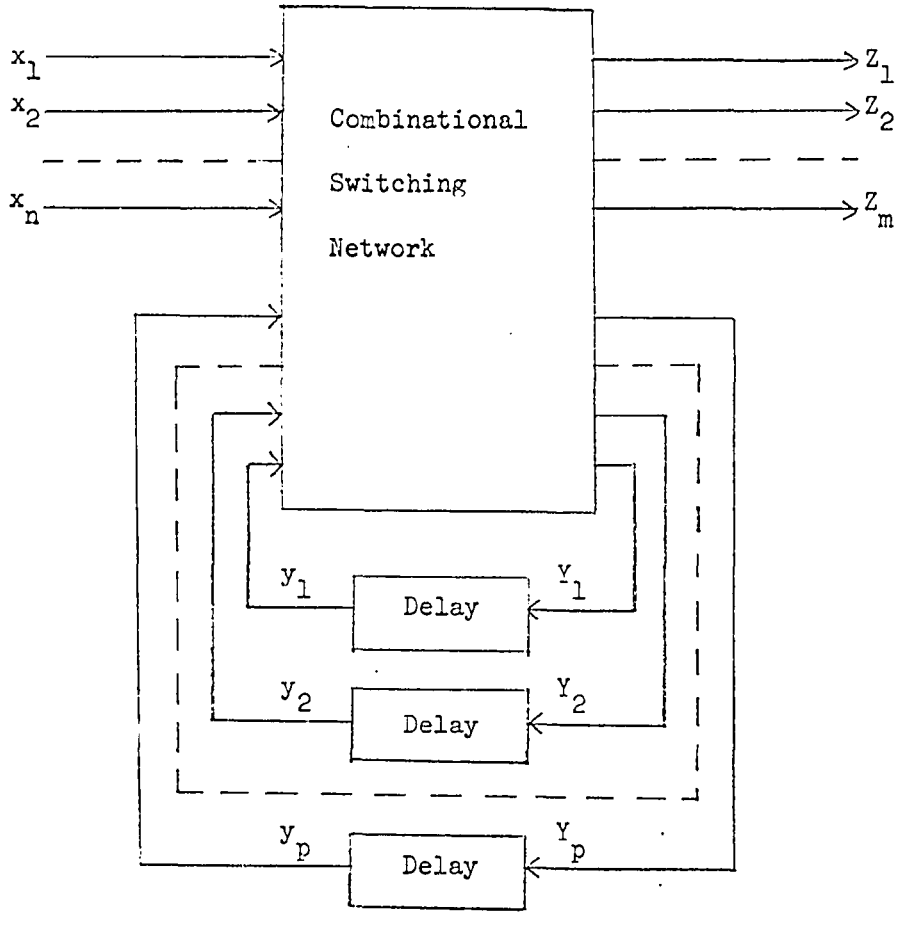


Figure 3. Model of an asynchronous sequential switching circuit

table. The primitive flow table is simply a systematic arrangement of the problem specifications. One might say that a primitive flow table is to sequential circuits as a truth table is to combinational circuits. A primitive flow table for this example is shown in Figure 4.

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

Figure 4. Primitive flow table

Each of the columns of the flow table represents an input state and each row of the table represents an internal or secondary state. The entries of the flow table (circled or uncircled) indicate the next secondary state. For this reason, a flow table is sometimes called a "next state" matrix. For this example, if the circuit is in secondary row 4 and an input combination of x_1x_2 = 01 is presented, the next secondary state will be row 6. As long as the input combination remains 01, there will be no

further secondary circuit action and for this reason we call the circled 6 in row 6 a stable state. Thus, the uncircled entries of the flow table are called unstable entries and the circled entries are called stable entries. The circuit outputs are identified with the stable entries, or states, of the table. The dash (-) entries of the table are unspecified entries. They resulted from the input restriction that only one input variable changes state at a time. Therefore, there is no need to define the circuit action for the case of two input variables changing state simultaneously. As usual, these optional entries, or "don't cares", may be filled in later with any entry one chooses. Later it will be shown that proper choices for these optional entries can be an aid in problem simplification.

It is helpful at this point to trace through a particular input sequence for the primitive flow table of Figure 4. Suppose the circuit is presently in stable state 1 with an input x_1x_2 = 00 and an output of Z = 0. Consider now an input sequence x_1x_2: 00, 10, 11, 10. When the input combination is changed to 10, motion is horizontal in the table to unstable 3. Next, the secondary circuit changes and goes from unstable 3 in row 1 to stable 3 in row 3 and there the circuit has an output of Z = 1. For the next input the circuit goes to unstable 4 and then to stable 4 with an output of Z = 1. For the last input of the sequence it goes to unstable 7, stable 7, and an output of Z = 0. Notice that the relationship between input and output for this sequence is that specified in the original problem specifications. The binary 1 is associated with "on" and the binary 0 is associated with "off".

The next step is a check for redundant stable states. Redundant

stable states are sometimes introduced inadvertently during the construction of the primitive flow table because it is not apparent that two states are actually equivalent. Systematic techniques for detecting equivalent states in a primitive flow table are well known in the literature (1,8). It will suffice here to say that two stable states are equivalent if

(1) They have the same input state, and

(2) They have the same output state, and

(3) Each transition from these states, for the same input, is either to the same state or equivalent states.

There are no redundant states in the present example so one may continue.

A characteristic of the primitive flow table is that there is only one stable state to a row and hence the outputs may be directly associated with the rows of the flow table. Making a secondary assignment consists of coding the rows of the flow table with combinations of $y_1, y_2, \cdots y_p$. It would appear that the table in Figure 4 could be coded with at least three secondary variables, y_1, y_2 and y_3. Clearly, if this were done, the output would be a function solely of these secondary variables.

By a technique called merging, the output can be made to be a function of the input and secondary state. Merging usually results in a shorter flow table and a fewer number of secondary variables to code the rows. Two rows of a flow table may be merged if there are no conflicting state numbers in corresponding columns of each row. If a state number is circled in one of the merging rows, it is circled in the merged row. Here is a place where optional entries may be filled in so as to obtain an optimum merge. Generally, there is more than one way of merging the

rows of a flow table and a merger diagram is helpful in obtaining an optimum merge. A merger diagram has as its nodes the numbered rows of the flow table and shows all possible mergers of these rows. See Figure 5 for a merger diagram of the flow table in Figure 4. From the merger

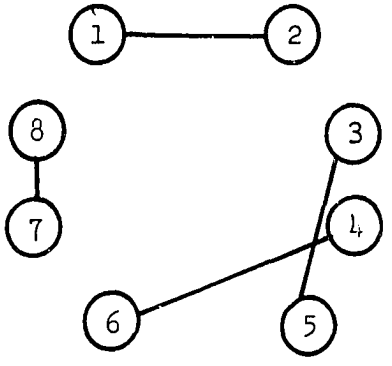


Figure 5. Merger diagram

diagram, a suitable merge is determined. Usually one seeks to reduce the number of rows in the flow table to a minimum. The idea here is that fewer rows in the flow table may result in a need for fewer variables to code the secondary states. The merged flow table for the present example appears in Figure 6.

	x_1x_2 00	01	11	10
a	(1)	(2)	4	3
b	(5)	6	4	(3)
c	1	(6)	(4)	7
d	1	2	(8)	(7)

Figure 6. Merged flow table

The merging was done with no consideration of the output and it may no longer be true, in the general case, that each row can now be associated with a particular output combination. Therefore, circuit outputs are usually not shown on the merged flow table. Notice, for this example, that the optimum merge happened to be that obtained by merging only rows with the same output. This is one case then, where it would be possible to show the output combinations on the merged flow table.

The next step of the synthesis procedure is the secondary assignment. Combinations of variables y_1, y_2, $\cdots$ y_p are assigned to distinguish the rows of the merged flow table. The problems involved in finding a satisfactory assignment will be discussed in some detail in the next section. For now, what is known to be a satisfactory assignment will be made so that the reader may continue on through the remainder of the synthesis procedure without loss of continuity. A satisfactory assignment is shown in Figure 7.

y_1y_2

Row a - 00

b - 01

c - 11

d - 10

Figure 7. Secondary assignment

After the secondary assignment is made, the excitation matrix is constructed. It was mentioned earlier that the flow table is a next state matrix. The construction of the excitation matrix then, amounts to replacing the numbered entries of the flow table with appropriate "next state" binary codes. As stated previously, the Y's are "next state" variables and therefore the internal entries of the excitation matrix are truth values for the Y's. For this reason, the excitation matrix is referred to as a Y-map. The Y-map for this example is shown in Figure 8.

	x_1x_2 00	01	11	10	
y_1y_2 00	00	00	01	01	a
01	01	11	11	01	b
11	10	11	11	10	c
10	00	00	10	10	d
	Y_1Y_2				

Figure 8. Excitation matrix or Y-map

In the next section we will show that one way of insuring proper operation of the final circuit independent of variations in transmission delays, is to excite only one secondary variable to change state at a time. Therefore, in the 00 column, row c, of Figure 8, a transition is effected from row c to row d and then to row a instead of directly from row c to row a.

The excitation expressions, Y_1 and Y_2 can be conveniently read from the Y-map since the presentation is in the form of a Karnaugh map. From Figure 8, one may write

$$Y_1 = x_2y_2 + y_1y_2 + x_1y_1$$
$$Y_2 = x_1\bar{y}_1 + \bar{y}_1y_2 + x_2y_2$$

Following the excitation matrix, an output matrix is prepared by first replacing each stable state in Figure 6 with the appropriate output combination from the primitive flow table of Figure 4. This stage of development is shown in Figure 9. If output transients are undesirable,

y_1y_2 \ x_1x_2	00	01	11	10	
00	0	0			a
01	1			1	b
11		1	1		c
10			0	0	d

Figure 9. Partly developed output matrix

the remaining locations of Figure 9 are filled in so that the output will change at most once for each change in input. If this restriction is

unnecessary in the practical application, one treats these locations as "don't care" conditions or optional entries. Figure 10 shows the completed output matrix, or what is sometimes called the Z-map, for the case of no output transients allowed. The output expression can be read directly from Figure 10 as $Z = y_2$.

y_1y_2 \ x_1x_2	00	01	11	10	
00	0	0	-	-	a
01	1	1	1	1	b
11	-	1	1	-	c
10	0	0	0	0	d
	Z				

Figure 10. Output matrix or Z-map

All that remains now is to synthesize the combinational logic for Y_1, Y_2 and Z and then complete the feedback loops in accordance with Figure 3. The complete logical design is shown in Figure 11.

Admittedly the steps of the synthesis procedure were rather brief in their explanation. For a more thorough treatment the reader is referred to the literature (1,7,8). In Figure 12 is a pictorial representation of the various parts of the synthesis procedure that have been presented.

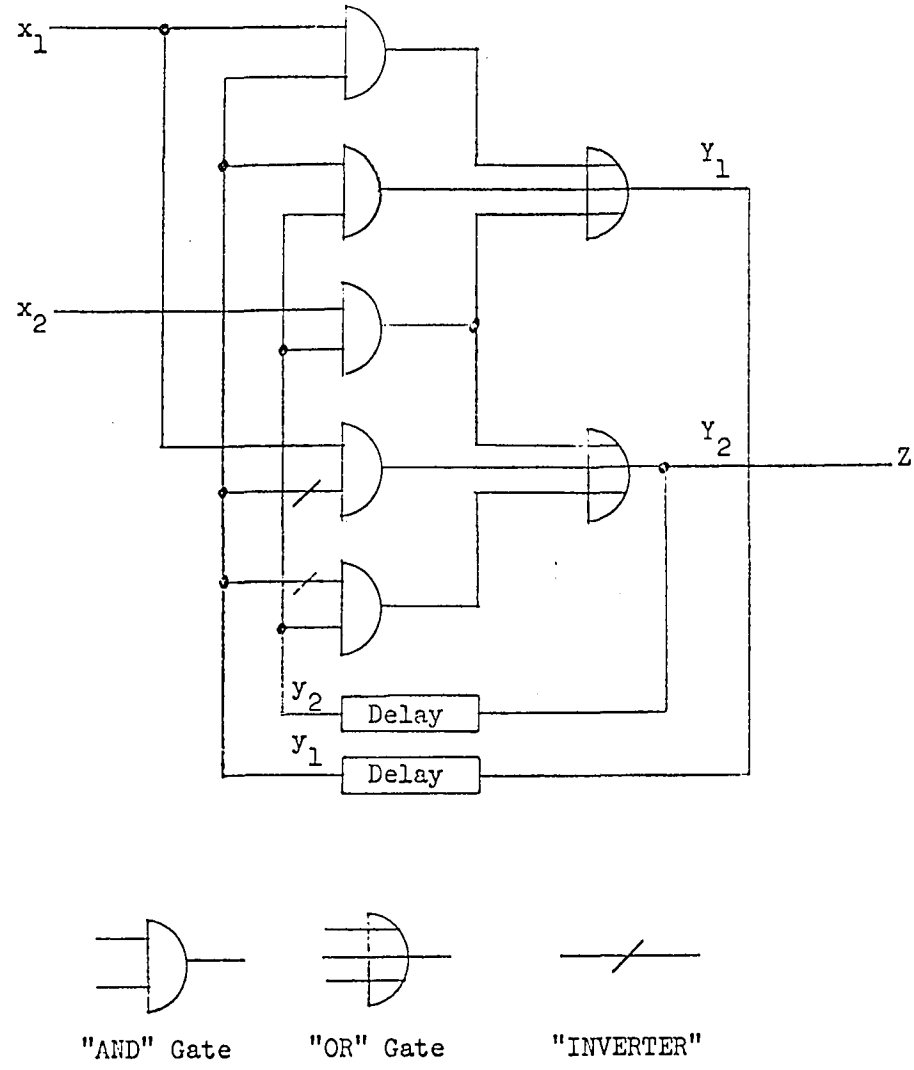


Figure 11. Logic design of synthesis example

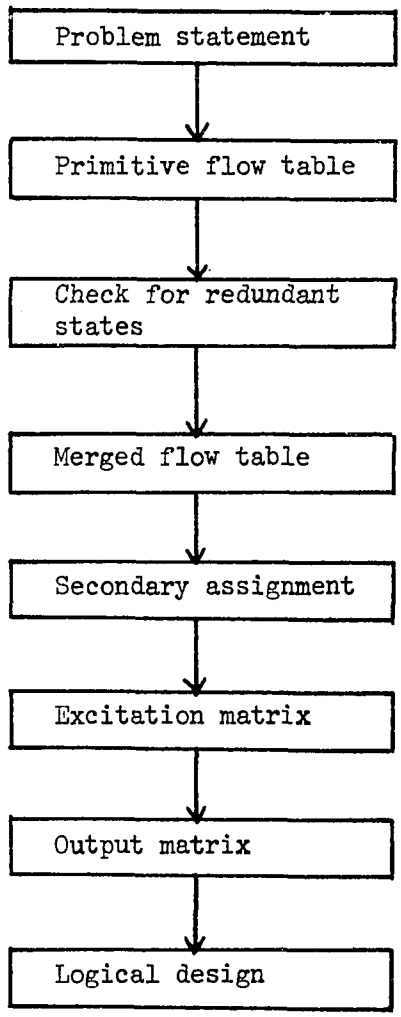


Figure 12. Synthesis procedure

2. The secondary state assignment problem

In this section a close examination will be made of the secondary state assignment problem and the constraints under which the assignment must be made. A new merged flow table, different from that used in the previous example, will be used to better illustrate the problems involved in making a satisfactory assignment.

Consider the merged flow table of Figure 13. The input combinations

	I_1	I_2	I_3	I_4	
	(1)	2	3	4	a
	5	(2)	(6)	(7)	b
	(5)	8	6	(4)	c
	1	(8)	(3)	7	d

Figure 13. Merged flow table

are labeled simply I_1, I_2, I_3 and I_4 since their binary code is of no particular interest at present. The rows that must be coded with secondary state variables are lettered a through d.

First an attempt will be made to make the same secondary assignment for this flow table that was made for the merged flow table in Figure 6, a - 00, b - 01, c - 11 and d - 10. Figure 14 shows the excitation that results from such an assignment.

y_1y_2	I_1	I_2	I_3	I_4
a - 00	00	01	10	11
b - 01	11	01	01	01
c - 11	11	10	01	11
d - 10	00	10	10	01

Figure 14. Excitation matrix

Consider now the transition from unstable 4 to stable 4 in Figure 13, which is the transition from row a to row c under input I_4 in Figure 14. When the circuit is in unstable 4 the situation is as shown in Figure 15.

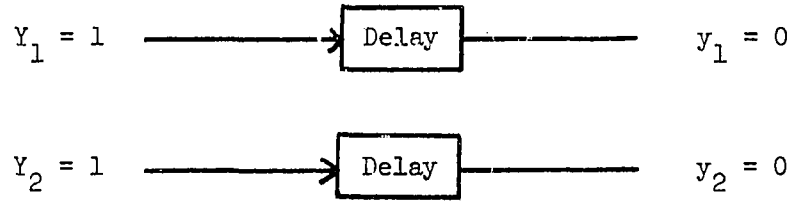


Figure 15. Excitation of secondary state variables

The secondary state is presently 00 and excitation function is in the process of exciting the secondary state to 11. The delays of Figure 15 are a part of the model in Figure 3. Practically speaking, these delays may be thought of as lumped combinational logic delay, amplification in the feedback loop to give greater than unity gain, or added delay to improve circuit performance. Now if these delays of Figure 15 are of the same magnitude, the next secondary state will be 11. However, if the delay associated with Y_1 is longer than the delay associated with Y_2, y_2 will be set to 1 before y_1 is set to a 1. That means the circuit will momentarily find itself in secondary state $y_1y_2 = 01$. Furthermore, once the secondary state becomes 01, Figure 14 shows that the excitation is changed to $Y_1Y_2 = 01$ and no further secondary action takes place. But location $y_1y_2 = Y_1Y_2 = 01$ under input I_4 in Figure 14 corresponds to stable state 7 in Figure 13. What was intended to be a transition from row a to row c has ended up in row b because of unequal transmission delays in the circuit. Such a circuit malfunction is said to be the

result of a critical race condition. It will be convenient at this point to list some definitions.

Definition 1: When the secondary excitation differs in value from the present secondary state in none of the bit positions, the circuit is stable.

Definition 2: When the secondary excitation differs in value from the present secondary state in exactly one bit position, the circuit is said to be cycling from the present secondary state to the secondary state that agrees with the present excitation.

Definition 3: When the secondary excitation differs in value from the present secondary state in two or more bit positions the circuit is said to be racing from the present secondary state to the secondary state that agrees with the present excitation.

Definition 4: If a race condition exists and unequal transmission delays can possibly cause the circuit to reach a stable state other than the one intended, the race is called a critical race.

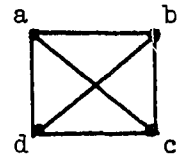
Definition 5: If a race condition exists and unequal transmission delays cannot possibly cause the circuit to reach a stable state other than the one intended, the race is called a non-critical race.

Quite obviously, critical races should be avoided in the design of an asynchronous sequential switching circuit. It might be mentioned here that the problem of making a secondary assignment to avoid critical races does not exist for the synchronous sequential circuit since gating pulses

from a clock are available to control delay and element switching.

Assignments to eliminate all races One obvious way to avoid the introduction of critical races in the circuit design is to eliminate races altogether. This can be done by requiring that all transitions be made in cyclic or totally sequential fashion. An attempt will be made to produce such an assignment for the flow table of Figure 13.

A helpful tool in accomplishing this type of assignment is the transition diagram. Each row of the flow table is represented by a node in the transition diagram and each inter-row transition is shown by a line joining the appropriate nodes. It has been common in the literature to use solid lines for those transitions that must go directly from an unstable entry to a stable entry and broken lines for those transitions having alternate routes. Alternate routes exist when there is more than one unstable entry of the same number. A transition diagram for this example is the following:



It is clear that the number of secondary variables needed to code any n-row flow table must be greater than or equal to $\log_2 n$. One might try to code this flow table with two secondary variables. But if cycles are allowed, the transition diagram shows that one must cycle, for example, from a to b, a to c and a to d. Cycling on a transition diagram corresponds to moving between adjacent squares on a Karnaugh map. Recall that two squares are adjacent on a Karnaugh map if only one

variable changes state in moving from the one square to the other. What is really being said here then, is that if the rows of the flow table, lettered a through d, are associated with the squares of a two-variable Karnaugh map, it is required that square a to be adjacent to squares b, c and d. Obviously this is impossible. However, a satisfactory assignment can be made if one increases the number of secondary state variables to three. The Karnaugh map is such a convenient way to look at particular assignments that it will be used extensively throughout the paper. A satisfactory assignment using three secondary variables is shown in map form in Figure 16. The letters a, b, c, d correspond to the rows of the

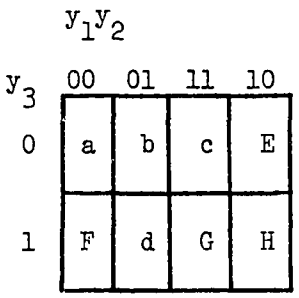


Figure 16. Karnaugh map of secondary assignment

flow table in Figure 13 and the upper case letters E, F, G, H are called "spare" secondary rows. These spares correspond to optional or "don't care" rows in the coded flow table. It may be necessary to fill the optional rows in with particular state numbers in order to effect all the transitions shown in the transition diagram. It is easy to determine whether or not a secondary assignment is satisfactory simply by tracing out on the Karnaugh map of the assignment, all transitions shown on the

transition diagram. If only cycles are allowed, transitions on the Karnaugh map must trace through adjacent squares. But squares passed through should be spare rows or the cycle may end in an undesirable state. In this example, all transitions may be made according to the following list which we will call the transitions specifications:

ab

aEc

aFd

bc

bd

cGd

This list is not unique in that the transition from row a to row c could be accomplished as aFHGc. The example flow table with the above secondary assignment and transition specifications is shown in Figure 17. A solid arrow indicates a cycle. The absence of an arrow leaving an unstable state implies a cycle directly to the stable state of the same number. It should be pointed out that in preparing and reading the transitions specifications the order of the transition is unimportant. In other words, once the transition from row a to c is specified as aEc, a transition from row c to a is specified to be cEa.

The spare states E, F, G and H may be used in more than one transition each, although there was no need to consider this possibility in our example. It can easily be shown that in general the same spare secondary row may be used for all those transition specifications that either begin or end in the same row. For example, from Figure 16 one could write

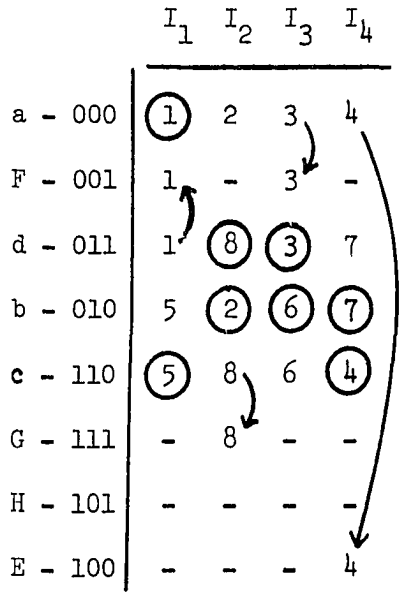


Figure 17. Flow table with secondary assignment

transition specifications aFd and aFHEc for the same flow table.

Systematic methods for determining assignments with no races are well known in the literature (1,5,8).

Assignments to eliminate only critical races In the previous section, an assignment method was discussed that allowed all transitions to be made without the introduction of any races. That was one way of assuring no circuit malfunctions due to critical races. All transitions were cyclic in nature and only one secondary state variable was excited at a time during a transition. Now, secondary assignments that allow multiple changes of secondary variables will be discussed. Remember that when more than one secondary variable is excited, a race condition exists. Therefore, one must insure that all races are non-critical. The same example flow table will be used with a different secondary

assignment to illustrate the use of non-critical races. The flow table and assignment appears in Figure 18. It will be demonstrated that all

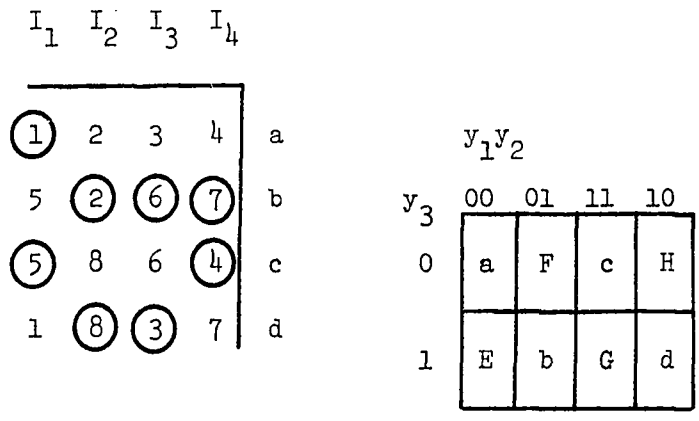


Figure 18. Flow table and secondary assignment

transitions may be accomplished with non-critical races. One sure test as to whether an assignment is workable or not is to construct the excitation matrix. Faulty assignments result in an inability to properly construct this matrix. In Figure 19 the excitation matrix for this example is shown. That part of the matrix for the transition from unstable 5 to stable 5 under input I_1 will be examined. When the circuit is in unstable 5, the present secondary state is 011 and the present excitation is 110. The excitation is different from the present state in the first and third bit positions. If y_1 and y_3 both change state simultaneously, the circuit will go directly from state 011 to state 110. However, if y_1 changes before y_3, the circuit will momentarily be in secondary state 111. Therefore, the secondary state 111 must have the capability of providing the proper excitation to carry the transition on through to stable 5. Faulty

$y_1y_2y_3$	I_1	I_2	I_3	I_4
a - 000	000	011	110	110
E - 001	000	011	011	011
b - 011	110	011	011	011
F - 010	110	011	110	110
c - 110	110	101	110	110
G - 111	110	101	011	011
d - 101	000	101	011	011
H - 100	000	101	110	110

Figure 19. Excitation matrix

assignments result in at least one case of conflicting excitations. Such is not the case here; an excitation of 110 is shown, the code associated with stable state 5, in the spare secondary row G under input I_1.

On the other hand, y_1 may change <u>after</u> y_3 and the circuit will momentarily be in spare F. Therefore, spare F under input I_1 must also show an excitation of 110.

It is interesting to note how conveniently all this information is displayed in a Karnaugh map of the assignment. Consider the same transition, b to c, on the assignment map in Figure 18. It is easy to see that c is a Hamming distance of two from b. In other words, the shortest path from b to c through adjacent squares on the map (changing one variable at a time) is two squares. The squares involved in the race from b to c are all those squares covered, moving cyclicly from b to c, over all paths of

length two. The length of the path is taken to be a count of the number of squares traversed in moving cyclicly from b to c. These four squares, a, F, G, c must all show the same excitation under input I_1 in the excitation matrix of the example. Since the transition was <u>from</u> a <u>to</u> c, they all have an excitation corresponding to secondary state c, or 110. If the transition had been <u>from</u> c <u>to</u> a, these locations would all show an excitation of 000.

The above illustration can obviously be generalized for an arbitrary race from row r_i to row r_j under input I_k for any flow table assignment. Let r_i be a Hamming distance of n from row r_j. All rows encountered in going from r_i to r_j by all paths of length n, and including rows r_i and r_j, must show as an excitation, the assigned secondary state of row r_j. Clearly, for a distance n, 2^n rows will have the same excitation.

There is an important subtlety that must be checked in an assignment utilizing races. This can best be explained with a different example. Consider an 8 row flow table with rows lettered a through h in which all transitions occur except a to b, c to d, e to f and g to h. A single column of such a flow table might look like Figure 20. Shown also in Figure 20 is a seemingly satisfactory secondary assignment and a partially constructed excitation matrix. The problem comes about in filling in a proper excitation for spare row L. The race from row c to a is non-critical but implies that the excitation in L be 0000. The race from f to g is also non-critical but implies that the excitation in L be 0101. Spare row L can be given just one excitation so the assignment is unsatisfactory if one insists on races for the transitions c to a and f to g. We say in this case, that the transition from c to a races critically

	I_i
a	(1)
b	2
c	1
d	4
e	(2)
f	4
g	(4)
h	(3)

Flow table column

y_3y_4 \ y_1y_2	00	01	11	10
00	a	J	d	K
01	L	g	M	f
11	c	N	b	P
10	Q	e	R	h

Secondary assignment

$y_1y_2y_3y_4$	I_i
a - 0000	0000
L - 0001	
c - 0011	0000
Q - 0010	0000
e - 0110	0110
N - 0111	0110
g - 0101	0101
J - 0100	0101
d - 1100	0101
M - 1101	0101
b - 1111	0110
R - 1110	0110
h - 1010	1010
P - 1011	----
f - 1001	0101
K - 1000	----

Partial excitation matrix

Figure 20. Flow table with assignment and partial excitation matrix

with the transition f to g. This is like the example of Figures 13 and 16 with cyclic specifications in that two transitions may make use of the same spare only if either the beginning or end of the transition is the same row. Thus, for the assignment in Figure 20, we could specify a pair of races as a(race)c and g(race)c but not the pair a(race)c and g(race)f

unless the a to c and g to f transitions occurred in different columns of the flow table.

In the above example of Figure 18, a secondary assignment was given that involved races for all transitions while in the previous example with the same flow table, all transitions were accomplished with cycles. It is quite possible in many problems to show a mixture of the two. For example, it is easy to see from the Karnaugh map of Figure 18 that one could choose to cycle from b to F and then to c instead of racing directly from b to c. In fact, it is quite clear that <u>any</u> transition that can be accomplished by a non-critical race can also be accomplished with just cycles, for the same assignment.

There is a significant difference between the cyclic assignment and the race assignment given for the flow table in Figure 13. The flow table and two assignments are repeated in Figure 21 for easy reference.

I_1	I_2	I_3	I_4	
(1)	2	3	4	a
5	(2)	(6)	(7)	b
(5)	8	6	(4)	c
1	(8)	(3)	7	d

y_3	y_1y_2 = 00	01	11	10
0	a	b	c	E
1	F	d	G	H

Assignment 1 (cyclic)

y_3	y_1y_2 = 00	01	11	10
0	a	F	c	H
1	E	b	G	d

Assignment 2 (race)

Figure 21. Flow table with cyclic and race assignments

Consider the transition from row c to d under input I_2 for each assignment. In Assignment 1 we must first cycle to G and then cycle to d while in Assignment 2 we can race directly from c to d. In Assignment 1, y_1 is excited at the beginning of the transition but y_3 is not excited until the circuit reaches spare row G. But in Assignment 2, both y_2 and y_3 are excited at the beginning of the transition. It is true that in Assignment 2 the circuit may momentarily be in a spare secondary state F or G, but the point is that all secondaries that are to switch have begun to switch prior to arrival at the spare. If a unit of time, T, is defined as the longest time required for any one state variable to change, the c to d transition in Assignment 1 may require a total transition time of 2T while in Assignment 2, the transition will be completed in, at most, time T. The first assignment then, requires input information to come at a rate no greater than 1/2T while the second will allow the circuit to accept information at a rate 1/T. As cycles increase in length for larger flow tables, assignments eliminating all races become less attractive in terms of circuit speed. An important aspect of the secondary state assignment problem is the development of a systematic assignment procedure that will allow all transitions to be accomplished in a minimum amount of time. This does not preclude the use of any cycles but simply says that if cycles are used, they must be of unit length.

The major contribution of this paper is the development of a minimum transition time secondary state assignment algorithm for asynchronous sequential circuits. One might wonder why assignments using only cycles are of any importance in view of the fact that both assignments in Figure 21 used the same number of secondary variables. Huffman (5) has shown

standard assignments, which we will use as upper bounds, for 2^m-row (m an integer) flow tables that require 2m - 1 secondary variables for cyclic specifications but 2^m - 1 variables for minimum transition time specifications. Therefore, if minimum number of secondary variables is a consideration, one might choose an assignment whereby all transitions, or most transitions, are accomplished with cycles. This might be particularly true for larger flow tables.

B. Summary

In summary, this section serves to introduce design problems associated with asynchronous sequential switching circuits. The secondary state assignment problem and assignment constraints were studied in some detail. The characteristics of satisfactory assignments were studied to show why one assignment might be preferable over another. General assignment methods were not discussed; assignments were stated and then analyzed. Section II will be devoted to the development of algorithms for construction of minimum transition time assignment codes.

II. PARTITION THEORY RELATED TO THE STATE ASSIGNMENT PROBLEM

A. Introduction to Partition Theory

The purpose of this section is to develop algorithms for the construction of minimum transition time secondary state assignments for asynchronous sequential circuits. A minimum code assignment will be defined as that assignment which allows all transitions to be accomplished in a minimum amount of time and uses the fewest number of secondary state variables.

The algorithms developed in this paper are strongly based on the concept of partition theory, and therefore a brief introduction to partition theory will be a necessity. Hartmanis (4) is responsible for much of the original work in partition theory and his terminology will be used throughout this paper. Hartmanis was primarily interested in a solution to the state assignment problem for synchronous sequential circuits. It will be remembered that the state assignment problem for synchronous circuits is to find an assignment that minimizes the combinational logic requirements. As pointed out previously, the problem of avoiding critical race conditions need not exist in synchronous circuits. So the application of partition theory will be quite different here in the case of asynchronous circuits where the state assignment problem is defined to be that of obtaining assignments that avoid these critical race conditions.

1. Definitions and illustrations of partition properties

This section begins with a definition due to Hartmanis.

Definition 6: A partition π on a set S is a collection of disjoint subsets of S such that their set union is S.

The subsets of the partition π on S are called the blocks of the partition and π is described by listing these blocks. The partition $\pi = 0$ is that partition in which each block consists of a single element; the partition $\pi = I$ is that partition in which all elements are contained in one block. The partitions $\pi = 0$ and $\pi = I$ are called trivial partitions.

As an illustration of what is meant by a partition, consider an arbitrary assignment for the following flow table with rows lettered a through d:

y_1y_2

a - 00

b - 01

c - 10

d - 11

One says that the variable y_1 determines the partition $\pi_1 = \overline{a,b};\ \overline{c,d}$ and y_2 determines the partition $\pi_2 = \overline{a,c};\ \overline{b,d}$. The elements of partition π_1 are a, b, c, and d; the blocks are $\overline{a,b}$ and $\overline{c,d}$. Next, it is convenient to define some algebraic properties of partitions.

Definition 7: Partition π_2 is $\leq \pi_1$ if and only if every block of π_2 is contained in a block of π_1.

The sum of two partitions, $\pi_1 + \pi_2$, is defined as follows:

Definition 8: Two elements a and b are in the same block of $\pi_1 + \pi_2$ if and only if these elements are in the same

block of π_1 or π_2 or both.

The product of two partitions, $\pi_1 \cdot \pi_2$, is defined as follows:

Definition 9: Two elements a and b are in the same block of $\pi_1 \cdot \pi_2$ if and only if they are in the same block of π_1 and in the same block of π_2.

To illustrate the construction of the product and sum of two partitions, consider again $\pi_1 = \{\overline{a,b};\ \overline{c,d}\}$ and $\pi_2 = \{\overline{a,c};\ \overline{b,d}\}$. Then $\pi_1 \cdot \pi_2 = \{\overline{a};\ \overline{b};\ \overline{c};\ \overline{d}\} = 0$ and $\pi_1 + \pi_2 = \{\overline{a,b,c,d}\} = I$. Clearly, if the partitions $\pi_1, \pi_2, \cdots \pi_i$ are to uniquely encode each of the n elements contained in these partitions, the product of the partitions must be the trivial 0 partition. To illustrate, consider some partitions one might use to uniquely code the rows of a flow table lettered a through f. If one is concerned with partitions that can each be described by a binary variable then each partition should consist of only two blocks. At least three partitions are needed and two example assignments are shown in Figure 22. For each assignment let y_1, y_2 and y_3 describe π_1, π_2 and π_3 respectively. Since the product of the partitions used in Assignment 1 is not the 0 partition, a unique code does not result for each of the partition elements when these partitions are used to make an assignment. Such is not the case in Assignment 2.

Huffman makes a comment in his paper (5) to the effect that in a secondary assignment, the Hamming distance of any two rows is unaffected by complementation of corresponding variables in the two states. This has a clear interpretation when the assignment is thought of as consisting of a collection of partitions. For example, π_1 in Assignment 1 of Figure 22 was described by y_1 and the first block was coded with a 0, the second

$\pi_1 = \{\overline{a,b};\ \overline{c,d,e,f}\}$

$\pi_2 = \{\overline{a,c,d};\ \overline{b,e,f}\}$

$\pi_3 = \{\overline{a,b,f};\ \overline{c,d,e}\}$

$\pi_1 \cdot \pi_2 \cdot \pi_3 = \{\overline{a};\ \overline{b};\ \overline{c,d};\ \overline{e};\ \overline{f}\}$

	$y_1y_2y_3$
a	000
b	010
c	101
d	101
e	111
f	110

Assignment 1

$\pi_1 = \{\overline{a,b};\ \overline{c,d,e,f}\}$

$\pi_2 = \{\overline{a,c,d};\ \overline{b,e,f}\}$

$\pi_3 = \{\overline{a,c,e};\ \overline{b,d,f}\}$

$\pi_1 \cdot \pi_2 \cdot \pi_3 = \{\overline{a};\ \overline{b};\ \overline{c};\ \overline{d};\ \overline{e};\ \overline{f}\}$

	$y_1y_2y_3$
a	000
b	011
c	100
d	101
e	110
f	111

Assignment 2

Figure 22. Codes defined by two different sets of partitions

with a 1. One could just as well have coded the first block with a 1 and the second with a 0, which would amount to complementing the y_1 column in the assignment. In other words, there is no distinction between the partition $\pi = \overline{a,b};\ \overline{c,d,e,f}$ and the partition $\pi' = \overline{c,d,e,f};\ \overline{a,b}$. Likewise, when the partitions are used to construct the assignment, the particular order in which the partitions are introduced is of no consequence.

This corresponds to interchanging the columns of an assignment and it is clear that such manipulations have no effect on the Hamming distances of the rows. Henceforth, significantly different partitions or assignments will mean different to within complementation and permutation of the secondary variables.

B. The Assignment Problem Stated in Terms of Partition Theory

1. A theorem on minimum transition time assignments

The development of minimum transition time secondary assignment algorithms will begin with a definition, an important theorem and its corollary.

Definition 10: Consider a Huffman flow table with rows r_1, r_2, $\cdots$ r_n. A direct transition from row r_i to row r_j is a transition whereby all secondary state variables that are to undergo a change of state are excited only at the beginning of the transition.

Therefore, a direct transition must be either a race from r_i to r_j or a cycle of unit length.

Theorem 1: A direct transition from row r_i to row r_j does not race critically with a direct transition from row r_k to row r_l if and only if a secondary assignment has been made such that at least one secondary variable, y_m, describes the following partition:

$$\pi_m = \{\overline{r_i, r_j, \cdots}; \overline{r_k, r_l \cdots}\}$$

Proof: For the first part of the proof it will be assumed that π_m exists in the assignment and it will be shown that it is impossible for the transition r_i to r_j to race critically with the transition r_k to r_l.

If y_m describes the partition π_m, the assignment must be of the following general form:

	$y_1 \cdots y_m \cdots y_p$
r_i -	0
$\vdots$	
r_j -	0
$\vdots$	
r_k -	1
$\vdots$	
r_l -	1

As explained previously, the y_m column could be complemented without changing the problem since both describe the same partition. According to Definition 10, in a direct transition all secondary variables that are to change state must be excited at the beginning of the transition. Keep in mind that one is not considering here the effect of two transitions r_i to r_j and r_k to r_l occurring simultaneously, but rather the possibility of these two transitions making use of the same spare row. Now since y_m is shown to be 0 at the beginning and end of the transition r_i to r_j, it will never be excited to a 1 during the entire transition. On the other hand, y_m will be a 1 throughout the transition r_j to r_k. Therefore, independent of the Hamming distance between r_i and r_j, or r_k and r_l, and independent of the switching times of the excited variables, the two pairs of transitions will never share the same spare secondary state.

In the second part of the proof, it will be assumed that π_m does <u>not</u> exist in the assignment and it will be shown that a transition from

r_i to r_j must race critically with a transition from r_k to r_l. There are eight significantly different ways to partition the four rows r_i, r_j, r_k, and r_l with two-block partitions. These are as follows:

$$\pi_1 = \{\overline{r_i, r_j, r_k, r_l, \cdots}; \overline{\cdots}\}$$

$$\pi_2 = \{\overline{r_i, \cdots}; \overline{r_j, r_k, r_l, \cdots}\}$$

$$\pi_3 = \{\overline{r_i, r_k, r_l, \cdots}; \overline{r_j, \cdots}\}$$

$$\pi_4 = \{\overline{r_i, r_j, r_l, \cdots}; \overline{r_k, \cdots}\}$$

$$\pi_5 = \{\overline{r_i, r_j, r_k, \cdots}; \overline{r_l, \cdots}\}$$

$$\pi_6 = \{\overline{r_i, r_j, \cdots}; \overline{r_k, r_l, \cdots}\}$$

$$\pi_7 = \{\overline{r_i, r_k, \cdots}; \overline{r_j, r_l, \cdots}\}$$

$$\pi_8 = \{\overline{r_i, r_l, \cdots}; \overline{r_j, r_k, \cdots}\}$$

If π_m does not exist in the assignment, one is interested in examining a largest assignment consisting of all of the above partitions with the exception of π_6. Let y_{m1} describe π_1, y_{m2} describe π_2, etc. The following partial assignment results:

	$y_1 \cdots$	y_{m1}	y_{m2}	y_{m3}	y_{m4}	y_{m5}	y_{m7}	$y_{m8} \cdots y_p$
r_i -		0	0	0	0	0	0	0
$\vdots$								
r_j -		0	1	1	0	0	1	1
$\vdots$								
r_k -		0	1	0	1	0	0	1
$\vdots$								
r_l -		0	1	0	0	1	1	0

It can be seen that for the columns shown, the transitions r_i to r_j and r_k to r_l will share the secondary states 01000-- where the dashes represent all possible combinations of 1's and 0's. Therefore, these two

transitions race critically with one another. All possible assignments including these rows r_i, r_j, r_k and r_l and meeting the conditions of the second part of this proof can be shown to be a sub-class of the general assignment given above. Note that by simply including π_6 in this general assignment, the critical race condition is eliminated.

Of interest, is the following corollary to Theorem 1:

Corollary 1: A direct transition from row r_i to row r_j does not race critically to r_k if and only if a secondary assignment has been made such that at least one secondary variable, y_m, describes the following partition:

$$\pi_m = \{\overline{r_i, r_j, \cdots}; \overline{r_k, \cdots}\}$$

This corollary would be proved very similar to Theorem 1. Just omit from the proof of Theorem 1, row r_l, and retain only those partitions in part 2 of the proof that remain significantly different from each other. There would be a total of four partitions to consider instead of eight.

2. Construction of the partition list

It is necessary to introduce the concept of an incompletely specified partition.

Definition 11: An incompletely specified partition, π, on a set S is a collection of disjoint subsets of S such that their set union is not necessarily S but may be another subset of S. Elements of S that do not appear in π will be called unspecified or optional elements with respect to that partition.

As usual, these unspecified elements may be defined in any way one pleases and will normally be defined in such a way as to bring about a problem

simplification. It will not be necessary in this paper, to define the product and sum operations for incompletely specified partitions. Such partitions will be completely specified before any product or sum operation is performed. A modification of the previously defined property of inequality is necessary for the discussion of incompletely specified partitions.

Definition 12: Partition $\pi_2 \leq \pi_1$, where π_1 and π_2 may be incompletely specified, if and only if all elements specified in π_2 are specified in π_1 and every block of π_2 is contained in a block of π_1.

As an illustration of Definition 12, consider the following incompletely specified partitions on a set S of eight elements lettered a through h:

$$\pi_1 = \{\overline{a,b};\ \overline{c,f}\}$$

$$\pi_2 = \{\overline{a,b,d};\ \overline{c,e,f}\}$$

$$\pi_3 = \{\overline{a,d};\ \overline{c,e}\}$$

From Definition 12 it is clear that $\pi_1 \leq \pi_2$, $\pi_3 \leq \pi_2$ and $\pi_1 \not\leq \pi_3$.

Now the development of the partition list will be explained with an illustration. For an example, consider the merged flow table of Figure 23. The transitions under input I_1 are c to a, d to e and f to b. According to Theorem 1, a satisfactory assignment will have to include at least the incompletely specified partitions, $\pi_1 = \{\overline{a,c};\ \overline{d,e}\}$, $\pi_2 = \{\overline{a,c};\ \overline{b,f}\}$ and $\pi_3 = \{\overline{b,f};\ \overline{d,e}\}$. Recall that the ordering of the variables in a transition is unimportant; if the transition from a to c can be made non-critically, then so can the transition from c to a. Under

I_1	I_2	
(1)	4	a
(3)	5	b
1	(5)	c
2	(4)	d
(2)	6	e
3	(6)	f

Figure 23. Flow table for illustration of partition list

input I_2 there are transitions a to d, b to c and e to f. Therefore, the assignment must also include the partitions, $\pi_4 = \{\overline{a,d};\ \overline{b,c}\}$, $\pi_5 = \{\overline{a,d};\ \overline{e,f}\}$ and $\pi_6 = \{\overline{b,c};\ \overline{e,f}\}$. A complete listing of all the partitions that must be included in the secondary assignment will be referred to hereafter as the partition list for the flow table.

Note that these incompletely specified partitions may be set up on a per column basis since secondary transitions are always completed within a single column of the flow table. A very necessary and obvious assumption in the design of asynchronous circuits is that a particular input is always present a sufficient length of time for the circuit to complete its secondary action.

It remains now to find a set of partitions, $\tau_1, \tau_2, \cdots \tau_n$ that include the partitions π_1 through π_6. Only a set of two-block partitions is of interest because one can then relate each partition to a secondary variable and there will be only one essentially different way to code each partition. The coding of partitions containing more than two blocks

is an assignment problem in itself. For example, there are three essentially different ways to code a four block partition with two binary variables, just as there are three essentially different ways to code a four row flow table with two secondary variables. An optimum code has been defined to be that code with the least number of secondary variables. Therefore, an attempt will be made to include all the partitions of the partition list in a minimum number of partitions τ. Systematic methods designed for obtaining this minimum set will be discussed in the next section. For the present, consider the following partitions:

$$\tau_1 = \{\overline{a,b,c};\ \overline{d,e,f}\}$$

$$\tau_2 = \{\overline{a,c,d};\ \overline{b,e,f}\}$$

$$\tau_3 = \{\overline{a,d,e};\ \overline{b,c,f}\}$$

Clearly, π_1 and π_6 are $\leq \tau_1$, π_2 and π_5 are $\leq \tau_2$ and π_3 and π_4 are $\leq \tau_3$. Alternately, τ_1 includes π_1 and π_6, τ_2 includes π_2 and π_5, and τ_3 includes π_3 and π_4. The coding of these partitions with the secondary variables y_1, y_2 and y_3 produces the assignment shown in Figure 24. The reader may easily verify for himself from the map of the assignment that all transitions may be accomplished in a minimum amount of time without critical races. In fact, it turns out in this example that no races are needed and all transitions may be accomplished with unit cycles.

The set partitions just used for Figure 24 is not the only set of three. Partitions $\tau_1 = \{\overline{a,c};\ \overline{b,d,e,f}\}$, $\tau_2 = \{\overline{a,d,e};\ \overline{b,c,f}\}$ and $\tau_3 = \{\overline{a,b,c,d};\ \overline{e,f}\}$ work equally well and again all transitions are unit cycles. It is interesting to look at the following set of four partitions that may be used to code the same flow table even though it is not minimum:

	$y_1y_2y_3$
a -	0 0 0
b -	0 1 1
c -	0 0 1
d -	1 0 0
e -	1 1 0
f -	1 1 1

y_3 \ y_1y_2	00	01	11	10
0	a		e	d
1	c	b	f	

Figure 24. The secondary assignment

$\tau_1 = \{\overline{a,c};\ \overline{b,d,e,f}\}$

$\tau_2 = \{\overline{a,b,c,d};\ \overline{e,f}\}$

$\tau_3 = \{\overline{a,c,d,e};\ \overline{b,f}\}$

$\tau_4 = \{\overline{a,d};\ \overline{b,c,e,f}\}$

Again, all the partitions π_1 through π_6 are $\leq$ to some τ. Figure 25 shows the resulting secondary assignment. Observe that in this case there is a

	$y_1y_2y_3y_4$
a -	0 0 0 0
b -	1 0 1 1
c -	0 0 0 1
d -	1 0 0 0
e -	1 1 0 1
f -	1 1 1 1

y_3y_4 \ y_1y_2	00	01	11	10
00	a			d
01	c		e	
11			f	b
10				

Figure 25. A non-minimum secondary assignment

mixture of unit cycles and non-critical races; the transition c to a is a unit cycle but the transition from d to e is a non-critical race over a Hamming distance of two.

Next, an example will be given that makes use of Corollary 1. Consider the flow table and associated partition list shown in Figure 26.

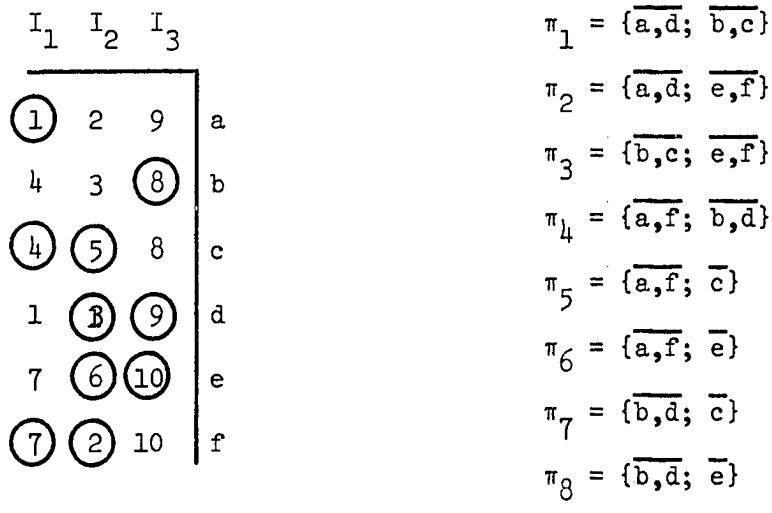


Figure 26. Flow table and partition list

This flow table is different from that of Figure 23 in that some transitions between states of Figure 26 involve no secondary circuit action; for example, there is no unstable 5 or unstable 6. Therefore, in Figure 26, one does not need to be concerned with any transition racing critically with a transition to state 5 but one must be careful that other transitions under input I_2 do not race critically to state 5. This implies the applicability of Corollary 1 and accounts for the incompletely specified partitions π_5, π_6, π_7 and π_8 in Figure 26. A minimum set of τ partitions and the corresponding assignment is shown in Figure 27.

$\tau_1 = \{\overline{a,d,f};\ \overline{b,c,e}\}$

$\tau_2 = \{\overline{a,b,d};\ \overline{c,e,f}\}$

$\tau_3 = \{\overline{a,e,f};\ \overline{b,c,d}\}$

y_3 \ y_1y_2	00	01	11	10
0	a	f	e	
1	d		c	b

Figure 27. Partitions and assignment for the flow table of Figure 26

The last example that will be considered in this section is an incompletely specified flow table in Figure 28. This example serves to

I_1	I_2	I_3	I_4	
(1)	4	7	10	a
1	5	8	(11)	b
2	(4)	(8)	(10)	c
-	(6)	(7)	11	d
1	6	8	(12)	e
(2)	(5)	-	12	f

$\pi_1 = \{\overline{a,b};\ \overline{c,f}\}$

$\pi_2 = \{\overline{a,e};\ \overline{c,f}\}$

$\pi_3 = \{\overline{a,c};\ \overline{d,e}\}$

$\pi_4 = \{\overline{a,c};\ \overline{b,f}\}$

$\pi_5 = \{\overline{b,f};\ \overline{d,e}\}$

$\pi_6 = \{\overline{a,d};\ \overline{b,c}\}$

$\pi_7 = \{\overline{a,d};\ \overline{c,e}\}$

$\pi_8 = \{\overline{a,c};\ \overline{b,d}\}$

$\pi_9 = \{\overline{a,c};\ \overline{e,f}\}$

$\pi_{10} = \{\overline{b,d};\ \overline{e,f}\}$

Figure 28. Incompletely specified flow table and partition list

illustrate the efficiency of the assignment method in making use not only of spare secondary rows, but optional entries of the original flow table as well. Optimum use is also made of what Huffman calls the k-sets of a

flow table.

Definition 13: A k-set exists in a single column of the flow table and consists of all k - 1 unstable entries leading to the same stable state, together with that stable state.

In the I_1 column of the flow table in Figure 28, there is a pair of transitions b to a and e to a. One need not consider the problem of these two transitions racing critically since both are in the same k-set and end in the same stable state. Therefore, there is no incompletely specified partition in Figure 28 that separates into separate blocks, the row pairs a,b and a,e. This is fortunate in a sence, because a single element can appear in only one block of a given partition. Since row d has an optional entry under I_1, element d is not specified in any of the partitions describing the transitions in the I_1 column. The set of τ expressions and corresponding assignment appear in Figure 29.

$\tau_1=\{\overline{a,c};\ \overline{b,d,e,f}\}$

$\tau_2=\{\overline{a,d,e};\ \overline{b,c,f}\}$

$\tau_3=\{\overline{a,b,d};\ \overline{c,e,f}\}$

y_3 \ y_1y_2	00	01	11	10
0	a		b	d
1		c	f	e

Figure 29. Partitions and assignment for the flow table in Figure 28

It is clear from the assignment that the optional entry in column I_1 will be used in the transition to stable 1 while the optional entry in column I_3 will be used in the transitions to stable 8. Observe from the map of the assignment that the transition from b to a in column I_1 may

race through d; so may the transition from e to a. But since both transitions are in the same k-set, there will be no conflict of excitations for d in the excitation matrix. For completeness, the excitation matrix is shown in Figure 30.

$y_1y_2y_3$	I_1	I_2	I_3	I_4
000	000	011	100	011
001	000	011	011	011
011	111	011	011	011
010	000	011	011	011
110	000	111	011	110
111	111	111	011	101
101	000	100	011	101
100	000	100	100	110

Figure 30. Excitation matrix for flow table of Figure 28

Summary In this section, partition lists were defined and illustrated for a variety of flow tables. Six-row flow tables were used throughout because they were about the right size to illustrate the principles involved without being too lengthy. The method, in principle, can be applied to a flow table of any size. A covering set of partitions was given in each case and the corresponding assignment examined for correctness. No mention was made of how these covering sets were obtained, other than possibly by inspection of the partition list. Systematic methods for obtaining these covering partitions will be developed in the next section.

3. Systematic reduction of the partition list

A convenient way to study the problem of systematic reduction of the partition list is to convert the partition list to the form of an

incompletely specified Boolean matrix. The conversion is straight-forward and will be illustrated with an example. Consider again the flow table of Figure 23 and its following associated partition list:

$$\pi_1 = \{\overline{a,c};\ \overline{d,e}\}$$
$$\pi_2 = \{\overline{a,c};\ \overline{b,f}\}$$
$$\pi_3 = \{\overline{b,f};\ \overline{d,e}\}$$
$$\pi_4 = \{\overline{a,d};\ \overline{b,c}\}$$
$$\pi_5 = \{\overline{a,d};\ \overline{e,f}\}$$
$$\pi_6 = \{\overline{b,c};\ \overline{e,f}\}$$

These partitions will be listed in abbreviated form as rows of the matrix. Insteand of showing $\pi_1 = \{\overline{a,c};\ \overline{d,e}\}$, the partition will be numbered according to the π subscript with just a space distinguishing the blocks of the partition as follows:

1 ac de

The columns will be the complete set of elements appearing in the partition list. Each coordinate of the matrix will contain a 1, 0 or optional entry as defined by the partition of that row. Figure 31 shows the Boolean matrix for this example.

	a	b	c	d	e	f
1 ac de	0	-	0	1	1	-
2 ac bf	0	1	0	-	-	1
3 bf de	-	0	-	1	1	0
4 ad bc	0	1	1	0	-	-
5 ad ef	0	-	-	0	1	1
6 bc ef	-	0	0	-	1	1

Figure 31. Boolean matrix formulation of partition list

Arbitrarily, the first block of each partition is coded with a 0 and the second with a 1. From the previous discussion of partition coding, it is immaterial whether the first or second block is coded with a 0, and therefore any or all of the rows of the matrix may be complemented without altering the problem description.

T. A. Dolotta and E. J. McCluskey, Jr. (2) have studied coding problems associated with incompletely specified Boolean matrices. Although their application is not the same, it will be convenient to use some of the same terminology. Some applicable definitions, with appropriate modifictions, will be given from their paper. The definitions apply equally well to columns and rows of a Boolean matrix.

Definition 14: Two columns (rows), F_i and F_j, will have an intersection of F_i and F_j, written $F_i \cdot F_j$, if and only if F_i and F_j agree wherever both F_i and F_j are specified. The intersection will be defined as a column (row) which agrees with both F_i and F_j wherever either is specified and contains optional entries everywhere else.

Definition 15: Column (row) F_i is said to include column (row) F_j if and only if F_j agrees with F_i wherever F_i is specified.

Definition 16: Column (row) F_i is said to cover column (row) F_j if and only if either F_j includes F_i, or if F_j includes the complement ($\overline{F_i}$) of F_i.

A consequence of Definition 16 is that any time one discovers two columns (rows) such that F_i covers F_j, column (row) F_i may be discarded.

A Boolean matrix is reduced by replacing pairs of columns (rows)

with their intersection as per Definition 14, discarding columns (rows) as per Definition 16 and repeating until there are no further reductions. An obvious problem in the reduction of a matrix is that if it is done on a step by step basis, and the matrix is large, it is nearly impossible to tell how to begin so as to obtain an optimum reduction. In the secondary state assignment problem for asynchronous circuits, one is usually interested in an assignment with the fewest number of secondary variables and hence fewest two-block partitions. For the matrix arrangement then, one is primarily interested in a reduction that will yield a minimum or near minimum number of rows.

Consider now some possibilities for the reduction of the Boolean matrix in Figure 31. One might choose to begin by replacing rows 1 and 3 with their intersection to give the following reduction:

	a	b	c	d	e	f
1•3	0	0	0	1	1	0
2	0	1	0	-	-	1
4	0	1	1	0	-	-
5	0	-	-	0	1	1
6	-	0	0	-	1	1

Row 1•3 does not cover any others so next replace 5 and 6 by their intersection to give

	a	b	c	d	e	f
1•3	0	0	0	1	1	0
2	0	1	0	-	-	1
4	0	1	1	0	-	-
5•6	0	0	0	0	1	1

There are no further row intersections and converting back to the partitions described by the rows of the reduced matrix we have the completely specified partitions $\tau_1 = \{\overline{a,b,c,f};\ \overline{d,e}\}$, $\tau_2 = \{\overline{a,b,c,d};\ \overline{e,f}\}$ and the incompletely specified partitions $\tau_3 = \{\overline{a,c};\ \overline{b,f}\}$, and $\tau_4 = \{\overline{a,d};\ \overline{b,c}\}$.

The codes for each of the rows a through f may be taken as the corresponding columns of the reduced matrix. No matter how the optional entries are filled in, a satisfactory minimum transition time assignment results and no two rows have the same code.

On the other hand, suppose the matrix of Figure 31 is reduced by making the following intersections:

	a	b	c	d	e	f
1•2	0	1	0	1	1	1
3•4	1	0	0	1	1	0
5•6	0	0	0	0	1	1

In the first reduction, four secondary variables were needed, but in the second, only three are needed for the assignment. While in this simple example it is easy to determine an optimum reduction, it should be fairly obvious that as the size of the matrix increases, and as optional entries increase, the optimum reduction becomes considerably more difficult to achieve just by inspection of the matrix.

It might be interesting, before continuing, to investigate the effect of column reduction. Note that in Figure 31, one may form intersections of columns $a \cdot \bar{f}$, $b \cdot \bar{d}$ and $c \cdot \bar{e}$. Let g, h and j represent these intersections respectively. After forming and substituting these intersections, the result is

	g	h	j
1	0	0	0
2	0	1	0
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	0

Row 1 covers 6, 2 covers 5 and 3 covers 4. If rows 4, 5 and 6 are discarded, the reduced matrix is

	g	h	j
1	0	0	0
2	0	1	0
3	1	0	0

The resulting partitions are

$$\tau_1 = \{\overline{g,h,j}\} = \{\overline{a,b,d};\ \overline{d,e,f}\}$$

$$\tau_2 = \{\overline{g,j};\ \overline{h}\} = \{\overline{a,c,d};\ \overline{b,e,f}\}$$

$$\tau_3 = \{\overline{g};\ \overline{h,j}\} = \{\overline{a,d,e};\ \overline{b,c,f}\}$$

To illustrate the writing of the partitions in terms of their original elements, consider from above $\tau_1 = \{\overline{g,h,j}\}$. This can be written as $\tau_1 = \{\overline{a,\bar{f},b,\bar{d},c,\bar{e}}\}$, where the lower bar means a complementation and the upper bar is the block designation. Since only two-block partitions are of interest, an element in one block may be shown as its complement in the other. Therefore one may write $\tau_1 = \{\overline{a,b,c};\ \overline{d,e,f}\}$.

The effect of column reduction is clear. Once it is decided to replace columns a and f, for instance, with the intersection $a \cdot \bar{f}$, one eliminates from further consideration any partition having elements a and f in the same block. It so happened in this example that a minimum solution could be obtained by insisting at the outset that elements a and f always be in different blocks of each assignment partition. The same was true for element pairs c,e and b,d. If instead, one lets $g = a \cdot \bar{e}$, $h = b \cdot \bar{d}$ and $j = c \cdot \bar{f}$ in the matrix of Figure 31, there is no way to reduce the number of rows to less than four. It has been shown then, what might have been suspected intuitively; column reduction may often preclude an optimum row reduction. The column reduction problem is further complicated by the fact that for larger matrices there are often many ways to reduce the number of columns and it seems impossible to predict which

column reductions will lead to the best row reduction.

Because our primary concern is a minimum row Boolean matrix and because column reduction may preclude an optimum row reduction, any further consideration of column reductions will be excluded from the remainder of this paper. Let it suffice to say that column reduction will always lead to a usable solution, sometimes a good solution, but often precludes an optimum solution.

A method will now be presented that will always lead to a minimum row reduction of a Boolean matrix. The method is similar to that developed by Unger (10) for the simplification of incompletely specified flow tables for synchronous sequential switching circuits.

Matrix Reduction Algorithm #1 First some definitions.

Definition 17: If there exists a row F_{ij} that will cover row F_i and row F_j of a Boolean matrix then F_i and F_j are said to be compatible. Otherwise F_i and F_j are incompatible.

Definition 16 still holds as a definition of what is meant by covering. An important point concerning Definition 17 is that compatibility is not a transitive relation. For example, if F_i is compatible with F_j and F_j is compatible with F_k, it does not necessarily follow that F_i is compatible with F_k. Larger compatibles may be built up from smaller ones by adding rows that are compatible with each member. A compatible which cannot be added to is called a maximal compatible.

Definition 18: A compatible is maximal if it is not a proper subset of any other compatible.

The reduction of a Boolean matrix by construction of a set of maximal compatibles will now be illustrated. For variety of example, consider

the flow table from Figure 28. The appropriate Boolean matrix is shown in Figure 32.

	a	b	c	d	e	f
1 ab cf	0	0	1	-	-	1
2 ae cf	0	-	1	-	0	1
3 ac bf	0	1	0	-	-	1
4 ac de	0	-	0	1	1	-
5 bf de	-	0	-	1	1	0
6 ad bc	0	1	1	0	-	-
7 ad ce	0	-	1	0	1	-
8 ac bd	0	1	0	1	-	-
9 ac ef	0	-	0	-	1	1
10 bd ef	-	0	-	0	1	1

Figure 32. Boolean matrix for the flow table of Figure 28

Following is a list of pairwise compatibles obtained from Figure 32:

(1,2) (1,7) (1,10) (6,7)

(2,$\overline{5}$) (2,6) (7,10)

(3,4) (3,$\overline{5}$) (3,8) (3,9) (8,9) (8,$\overline{10}$)

(4,5) (4,8) (4,9) (9,10)

($\overline{5}$,6)

From the list of pairwise compatibles one may construct the list of maximal compatibles. For example, 1 is compatible with 7, 1 is compatible with 10 and 7 is compatible with 10. Therefore (1,7,10) is a compatible and we may discard (1,7), (1,10) and (7,10). The compatible (1,7,10) is also a maximal compatible since no other member may be added to the set. A point to keep in mind is that if F_i is compatible with $\overline{F}_j$, then $\overline{F}_i$ is compatible with F_j. Following is a list of maximal compatibles for this example:

A (1,2)

B (1,7,10)

C (2,$\overline{5}$,6)

D (3,4,8,9)

E (3,$\overline{5}$)

F (4,5)

G (6,7)

H (8,$\overline{10}$)

J (9,10)

For identification purposes, the maximal compatibles are lettered A through J. It remains now to select the fewest number of maximal compatibles that will cover all the rows of the original matrix. It can be seen almost by inspection, in this example, that one should select maximal compatibles B, C and D. The partitions themselves can be determined from the intersection of the rows of each compatible. For example, compatible B corresponds to the intersection of rows 1, 7, and 10 of Figure 32. Or, one may look at partitions 1, 7, and 10 and see quickly that the partition identified is $\tau = \{\overline{a,b,d};\ \overline{c,e,f}\}$. The reduced matrix and element codes are as follows:

	a	b	c	d	e	f
1•7•10	0	0	1	0	1	1
2•5•6	0	1	1	0	0	1
3•4•8•9	0	1	0	1	1	1

Three secondary variables are needed for the assignment and a code for each secondary row is shown by the columns of the reduced matrix.

It may not always be as easy to select a minimum number of maximal compatibles from the complete list of maximal compatibles as it was in the previous example. A formal method does exist for determining a minimum set. It is essentially that introduced by Petrick (9) for the algebraic solution of prime implicant tables in the tabular method of simplifying Boolean expressions. For each row of the original Boolean

matrix, a Boolean expression is written indicating which maximal compatibles cover that particular row. Thus, in this example, row 1 may be covered by the maximal compatibles A or B, which is written in Boolean algebra as A + B. A sum is formed for each row of the matrix and the product of all these sums indicate how the entire matrix may be covered. In this example one would have the expression

$$(A + B)(A + C)(D + E)(D + F)(C + E + F)(C + G)(B + G)$$
$$(D + H)(D + J)(B + H + J)$$

The product of sums expression is converted to the sum of products expression

$$BCD + ABDFG + ADFGH + ADFGJ + ACDGH + ACDGJ + ABDEG +$$
$$ADEGH + ADEGJ + BCEFHJ + ADFGHJ$$

The sum of products expression logically states the same thing as the product of sums expression but in a different way. The number of literals in each term of the sum of products expression corresponds to the number of rows in the reduced Boolean matrix. Hence, if one desired a minimum row reduced matrix he would pick the term BCD. The maximal compatibles B, C and D are those previously selected by inspection. In some cases there is more than one minimum row reduced matrix. The algebraic solution in that case would clearly show all reductions and the designer could, because of other considerations, possibly pick one over the other. The example used above did not perhaps best illustrate the power of the algebraic solution since it turned out that the selected term of the sum of products expression was considerably smaller than the others. This would imply that the best solution could probably be determined rather easily by inspection of the list of maximal compatibles.

Matrix Reduction Algorithm #1 may be summarized with the following systematic steps:

1. Examine all pairs of rows of the incompletely specified Boolean matrix and list those pairs that are compatible. This is called a list of compatibles.
2. Enlarge each compatible from Step 1 by adding rows that are compatible with each member. For example, if the list of pairwise compatibles states that F_i is compatible with F_j, F_j is compatible with F_k and F_i is compatible with F_k, an enlarged compatible (F_i, F_j, F_k) can be formed.
3. Continue Step 2 until no compatible can be further enlarged.
4. Discard all compatibles that are either identical to other compatibles or are a proper subset of other compatibles. The remaining compatibles comprise the list of maximal compatibles.
5. Determine a least number of maximal compatibles that will cover all the rows of the original matrix. This may be done systematically as follows:
   a. Letter each maximal compatible for identification.
   b. Write the Boolean sum of products expression that logically states how the entire matrix may be covered. (See page 55).
   c. Convert the Boolean expression to product of sums form.
   d. A term from the Boolean expression containing the fewest literals describes a least number of maximal

compatibles that will cover all rows of the original matrix.

6. The maximal compatibles selected in Step 5 each describe an intersection row of the reduced matrix. Furthermore, each intersection represents one partition to be used in the secondary assignment.

It should be pointed out that W. Starrett of the Bell Telephone Laboratories has been reported by Unger (10) to have demonstrated the feasibility of programming to find a list of maximal compatibles for synchronous machines with 28 or fewer states. For matrix reduction then, one would expect to use essentially the same kind of program for matrices containing 28 or fewer rows.

A serious disadvantage of the algorithm just described is that it becomes quite lengthy for moderate increases in flow table size. For example, the author has investigated, among others, a 6-row flow table that resulted in a 14-row matrix, an 8-row flow table with a 30-row matrix and a 12-row flow table with a 60-row matrix. Determination of the pairwise compatibles alone requires an investigation of $n!/2(n-2)!$ pairs where n is the number of rows of the merged flow table. In the case of the 14-row matrix, the complete list of maximal compatibles had about 20 entries. It was not difficult to obtain what seemed to be a minimum row reduction from the list of maximal compatibles but an algebraic solution to prove it was optimum would be quite tedious by hand computation and was therefore not attempted.

The large increase in work required to obtain an optimum assignment for a moderate increase in flow table size is not surprising when one

considers the number of assignments that exist for a flow table as a function of its size. Haring (3) presents the following table:

Table 1. Number of secondary assignments as a function of the number of flow table rows

Number of rows in flow table	Number of secondary variables	Number of non-degenerate essentially different state assignments
2	2	0
3	2	3
3	3	1
4	2	3
4	3	29
4	4	34
5	3	140
5	4	1,015
5	5	2,688
6	3	420
7	3	840
9	4	10,810,800
10	4	75,675,600
16	4	54,486,432,000

As previously stated essentially different assignments are those exclusive of assignments obtained by complementation or permutation of the secondary state variables.

Another factor that makes optimum reduction of the Boolean matrix difficult to obtain for large matrices is due to the fact that no matter how many columns appear in the matrix, there are never more than four entries specified in each row. As more optional entries are introduced, considerably more possibilities must be examined in order to determine an optimum reduction.

The author's experience with the above algorithm would indicate that a list of maximum compatibles can be conveniently obtained by hand

computation when the flow table produces a Boolean matrix of about 15 rows or less. An algebraic solution to determine a minimum set of maximal compatibles becomes tedious when the list of maximal compatibles has more than 10 entries or so. Next, an algorithm that works well for the reduction of up to 60-row Boolean matrices will be developed.

Matrix Reduction Algorithm #2 We have just shown that as flow tables increase in size it becomes considerably more difficult to obtain an optimum secondary assignment, in the sense that the fewest number of secondary variables are required. At least this is certainly the case using our previous algorithm. No other algorithm is known that will handle the problem any easier and always produce an optimum assignment. Therefore, one is lead to the development of an algorithm that may be used for larger matrices, but cannot be guaranteed to always yield an optimum reduction. One would expect such an algorithm to be a series of steps leading to a solution, but with the possibility that as each step is executed, it is impossible to tell its complete effect on the final solution.

Algorithm #2 for reducing Boolean matrices is based on the assumption that for many Boolean matrices, an optimum or near optimum reduction may be obtained by removing, on a step by step basis, large groups of intersecting rows. In other words, look for a largest group of intersecting rows, represent them with their intersection, remove them from the matrix, and for the part of the matrix remaining, look again for a largest group of intersecting rows, etc. The algorithm will be stated, illustrated with an example, and then an attempt will be made to show some of the reasoning behind the steps. In the algorithm a specified entry is a 1

or 0. The optionalentry (-) is unspecified.

1. Select a column of the Boolean matrix with the largest number of specified entries and identify it with the letter A. If several columns have the same largest number of specified entries, arbitrarily select one of them.
2. Complement appropriate rows of the matrix so that all specified entries in the column selected in Step 1 agree.
3. Identify those rows that are not specified under the column selected in Step 1 with the letter B.
4. Examine each column not identified with an A and determine the difference between the number of 1's and 0's in each of these columns. Ignore for this count, those rows identified with a B or C.
5. Select the column from Step 4 that has the largest difference magnitude. Set that column to a 1 or 0, whichever was larger, and identify the column with an A. If several columns have the same largest difference, arbitrarily select one of them.
6. Examine those rows not identified with a B or C. If a row does not agree with the setting of the column in Step 5, identify that row with a C.
7. Consider those rows identified with a B and specified under the column selected in Step 5. Remove the B identification from these rows and either complement

them or not complement them so that they will agree with the selected column setting in Step 5.

8. Go back to Step 4 unless all columns are identified with an A. If all columns are identified with an A, go to Step 9.

9. All rows not identified with a C have an intersection. This intersection represents one of the partitions to be used in the assignment. Determine this intersection and remove the covered rows from the matrix. Remove all identifiers from the remaining matrix and go back to Step 1. The algorithm is ended when there are no rows remaining in the matrix.

Now the algorithm will be illustrated with an example. Consider the example flow table in Figure 33 and its corresponding Boolean matrix in Figure 34.

I_1	I_2	I_3	I_4	
(1)	2	3	4	a
1	5	(6)	7	b
8	(2)	9	12	c
(8)	(13)	(3)	(7)	d
10	(11)	6	(4)	e
(10)	(5)	(9)	(12)	f

Figure 33. Flow table for algorithm illustration

		a	b	c	d	e	f
1	ab cd	0	0	1	1	-	-
2	ab ef	0	0	-	-	1	1
3	cd ef	-	-	0	0	1	1
4	ac bf	0	1	0	-	-	1
5	ac d	0	-	0	1	-	-
6	ac e	0	-	0	-	1	-
7	bf d	-	0	-	1	-	0
8	bf e	-	0	-	-	1	0
9	ad be	0	1	-	0	1	-
10	ad cf	0	-	1	0	-	1
11	be df	-	0	1	-	0	1
12	ae bd	0	1	-	1	0	-
13	ae cf	0	-	1	-	0	1
14	bd cf	-	0	1	0	-	1

Figure 34. Boolean matrix for flow table in Figure 33

The algorithm proceeds as follows:

1. Columns a, b, c and f each have nine specified entries. Select column a and identify it with an A.
2. No rows need to be complemented.
3. Identify rows 3, 7, 8, 11 and 14 with a B.
4. Counts of 1's and 0's must be made for columns b through f. In column f, for example, there is a count of zero 1's and four 0's.
5. Column f is selected, set to a 1, and identified with an A.
6. All rows not identified with a B or C agree with the setting of column f.
7. The B identification is removed from rows 3, 7, 8, 11 and 14. Rows 7 and 8 are complemented.
8. Return to Step 4.

4. Count 1's and 0's in columns b,c,d and e.
5. Column d is selected and set to a 0, since it has a maximum count difference with five 0's and three 1's.
6. Identify rows 1, 5 and 12 with a C. Notice that rows identified with a C are those that will not be covered by the partition presently being constructed.
7. No rows are identified with a B.
8. Return to Step 4.

At this point the matrix and identifiers appear as follows:

	a	b	c	d	e	f	
1	0	0	1	1	-	-	C
2	0	0	-	-	1	1	
3	-	-	0	0	1	1	
4	0	1	0	-	-	1	
5	0	-	0	1	-	-	C
6	0	-	0	-	1	-	
7	-	1	-	0	-	1	
8	-	1	-	-	0	1	
9	0	1	-	0	1	-	
10	0	-	1	0	-	1	
11	-	0	1	-	0	1	
12	0	1	-	1	0	-	C
13	0	-	1	-	0	1	
14	-	0	1	0	-	1	
	A		A		A		

If one proceeds through the algorithm until Step 9 is entered, all columns will have been identified with the letter A. All rows will have been identified with the letter C <u>except</u> rows 3, 4, 6, 7 and 9. So that the reader may follow, in case of a tie in Steps 1 or 5, the left-most column was selected. Therefore, rows 3, 4, 6, 7 and 9 should have an intersection that in turn determines a partition τ_1 to cover these rows in the matrix. This is the case and the resulting partition is $\tau_1 = \{\overline{a,c,d};\ \overline{b,e,f}\}$. The reader may easily verify that partitions 3, 4, 6, 7 and 9 of Figure 34

are included in τ_1.

Let one now go back to Step 1 of the algorithm. The matrix now consists of Figure 34 exclusive of rows 3, 4, 6, 7 and 9. The process is continued until we get the partitions and secondary assignment shown in Figure 35.

$\tau_1 = \{\overline{a,c,d};\ \overline{b,e,f}\}$

$\tau_2 = \{\overline{a,b,d,e};\ \overline{c,f}\}$

$\tau_3 = \{\overline{a,c,e};\ \overline{b,d,f}\}$

$\tau_4 = \{\overline{a,b};\ \overline{c,d,e,f}\}$

y_3y_4 \ y_1y_2	00	01	11	10
00	a			
01		e		c
11	d		f	
10		b		

Figure 35. Partitions and secondary assignment for the flow table in Figure 33

An attempt will be made now to show the reasoning behind some of the steps of the algorithm. The main theme of the algorithm is: Given an incompletely specified Boolean matrix, determine a partition that will cover the maximum or near maximum number of rows in the matrix. These covered rows are then discarded and a subset of the original matrix is considered. One way to arrive at this maximum partition is to determine one by one, the setting of each individual column, so that a maximum number of rows are covered. Or alternately, determine the column settings in such a manner that a minimum number of rows of the matrix will be discarded as the setting for each column is established. Since the

intent is to determine the column settings on a step by step basis, the outcome will be greatly dependent upon which column one starts with. Hence, in Step 1 the column is selected that will bring a maximum number of rows into consideration at the beginning of the algorithm. One would suspect also, that the outcome would be greatly dependent upon the order in which one determined the setting of the succeeding columns. Therefore, in Steps 4 and 5 one chooses that column which is most strongly associated with the already chosen columns and at the same time requires that a relatively few number of rows be excluded from further consideration. While the algorithm always attempts to find a maximum intersection in the matrix, there is obviously no guarantee that a true maximum is always produced. However, experience has indicated that at least a near maximum intersection can be obtained in each case.

An important advantage of this algorithm for the reduction of the Boolean matrix is that the steps are very systematic, programmable on a computer and capable of handling relatively large matrices. Matrices of up to 60 rows have been reduced by hand computation using this algorithm. It might be pointed out that for all examples presented thus far in this paper, application of this algorithm for matrix reduction has produced what seemed to be optimum reductions in every case.

An obvious disadvantage of the algorithm is that there exist matrices where the optimum reduction or reductions does not include the intersection of the maximum number of rows. The 60-row matrix mentioned earlier was reduced with the above algorithm to a matrix of 5 rows. The first intersection covered 27 rows. However, with a little trial and error, a reduced matrix of 4 rows was obtained and no intersection covered more

than 25 rows.

Summary Two algorithms that produce optimum or near optimum row reductions for incompletely specified Boolean matrices have just been described. Matrices of 15 rows or less can be optimally reduced by Reduction Algorithm #1, while matrices of at least 60 rows may be reduced by Reduction Algorithm #2 with no guarantee that the result is optimum. The first algorithm has the advantage of producing an optimum solution butthe disadvantage of becoming quite long and impractical for matrices of more than 15 rows and does not appear to be easily programmable. The second algorithm has been used for up to 60-row matrices, could be programmed to handle even more, but has a disadvantage of not necessarily producing an optimum reduction.

4. Assignment Method #1

A minimum transition time secondary assignment method, which will be called Assignment Method #1, is summarized in Figure 36. Each block of Figure 36 has been discussed in detail. Either Algorithm #1 or #2 may be used to reduce the Boolean matrix. The reader is aware of the advantages and limitations of each.

This assignment method is theoretically applicable for flow tables of any size, but practically speaking, it is efficient in terms of time and results for flow tables of about 10 rows or less and can become quite lengthy for hand computation when working with flow tables of 12 rows or more. Examples of merged flow tables larger than 12 rows have been rare in the literature. Recall that a merged flow table of 12 rows could correspond to a considerably longer primitive flow table. The true

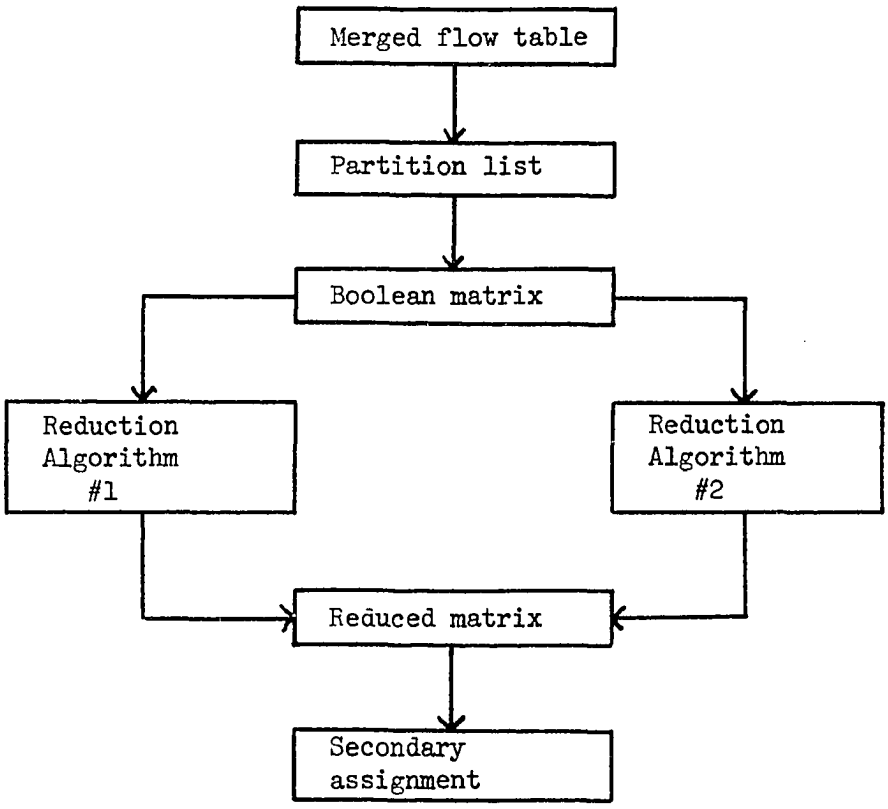


Figure 36. Summary of Secondary Assignment Method #1

limiting factor in the assignment method is the size of the incompletely specified Boolean matrix to be reduced. When Reduction Algorithm #2 is programmed, it may be possible to consider merged flow tables larger than 12 rows.

5. Assignment Method #2

It is advantageous to construct an assignment method that is shorter, although less efficient in terms of secondary variables, than Method #1. This will allow one to at least establish an upper bound on the number of secondary variables needed for a minimum transition time assignment.

Let one consider such a method in this section. It will be a modification of a method due to Liu (6). Liu does not explain his algorithm in terms of partitions and it seems to be longer and more difficult than it need be. Assignment Method #2 will be introduced with a definition and theorem.

Definition 19: A column partition is a partition constructed from a single column of a flow table with each k-set of the column appearing as a separate block. A column partition may be either completely or incompletely specified.

As an illustration of Definition 19, consider column I_1 of Figure 33. The column partition is $\pi = \{\overline{a,b};\ \overline{c,d};\ \overline{e,f}\}$. This column partition is completely specified because all elements of the set a through f are specified in the partition. Incompletely specified column partitions arise when there are optional entries in the corresponding column of the flow table.

Theorem 2: A secondary assignment constructed from all the column partitions of a flow table contains no critical races, even if all transitions are direct.

Proof: Consider a column of a flow table to contain n k-sets. These k-sets can be distinguished by the product of N_o two-block partitions where N_o is the smallest integer $\geq \log_2 n$. If the product of these N_o two-block partitions distinguishes all of the k-sets, then for each pair of k-sets, n_p and n_q, some one of these two-block partitions must contain n_p and n_q in separate blocks. Transitions can occur only within k-sets. Assume rows r_i and r_j of Theorem 1 to be in k-set n_p and rows r_k and r_l to be in k-set n_q. Now all the conditions of Theorem 1 are met. Therefore, there are no critical races in the flow table column and

all transitions may be accomplished directly.

Since races are always restricted to the columns of a flow table, it follows that critical races can be avoided in the entire flow table if all the column partitions are used to construct the secondary assignment.

Clearly, before the column partitions are coded to give the assignment, only those partitions that are essentially different should be retained. An example will be given to illustrate how efficient this assignment method may be for some particular flow tables. This example, shown in Figure 37, is one for which Caldwell (1) determines a secondary assignment but by a technique quite longer and less systematic than our Method #2.

I_1	I_2	I_3	I_4	
(1)	(5)	11	15	a
(2)	7	12	(13)	b
(3)	5	-	15	c
(4)	7	-	13	d
3	(6)	(9)	16	e
1	(7)	11	-	f
3	(8)	9	-	g
4	8	(10)	(14)	h
-	6	(11)	16	j
-	8	(12)	14	k
2	-	12	(15)	l
4	-	10	(16)	m

Figure 37. Flow table from Caldwell

The column partitions are

$$\pi_1 = \{\overline{a,f};\ \overline{b,l};\ \overline{c,e,g};\ \overline{d,h,m}\}$$

$$\pi_2 = \{\overline{a,c};\ \overline{b,d,f};\ \overline{e,j};\ \overline{g,h,k}\}$$

$$\pi_3 = \{\overline{a,f,j};\ \overline{b,k,l};\ \overline{e,g};\ \overline{h,m}\}$$

$$\pi_4 = \{\overline{a,c,l};\ \overline{b,d};\ \overline{e,j,m};\ \overline{h,k}\}$$

Note that all the column partitions are incompletely specified. Just by inspection of these column partitions it can be seen that the completely specified partition $\pi_5 = \{\overline{a,f,j};\ \overline{b,k,l};\ \overline{c,e,g};\ \overline{d,h,m}\}$ includes π_1 and π_3, and $\pi_6 = \{\overline{a,c,l};\ \overline{b,d,f};\ \overline{e,j,m};\ \overline{g,h,k}\}$ includes π_2 and π_4. Therefore, the coding of the blocks of π_5 and π_6 will result in a satisfactory minimum transition time assignment. Two secondary variables are needed to code each partition for a total of four secondary variables in the assignment. This is the minimum number of variables one could use to code any 12 row table. The secondary assignment is shown in Figure 38; y_1 and y_2 code π_5, y_3 and y_4 code π_6.

	$y_1y_2y_3y_4$
a	0 0 0 0
b	0 1 0 1
c	1 0 0 0
d	1 1 0 1
e	1 0 1 0
f	0 0 0 1
g	1 0 1 1
h	1 1 1 1
j	0 0 1 0
k	0 1 1 1
l	0 1 0 0
m	1 1 1 0

y_3y_4 \ y_1y_2	00	01	11	10
00	a	l		c
01	f	b	d	
11		k	h	g
10	j		m	e

Figure 38. Secondary assignment for the flow table in Figure 37

The above example illustrated a case where Assignment Method #2 produced a minimum transition time assignment with a minimum number of secondary variables. If Method #2 were being used in that example to obtain an upper bound on the number of secondary variables required, there would be no need to consider any other algorithm because the upper bound turned out to be the lower bound as well. Unfortunately, Method #2 does not work this well in most cases. Huffman (5) develops in his paper, a flow table for a reversible counter. The merged flow table consists of eight rows and if Method #2 is used to code the table, six secondary variables are required. However, Method #1 produces a code requiring only three secondary variables.

As a further comparison, consider coding the flow table of Figure 28 with Method #2 and compare that with the assignment shown in Figure 29. The column partitions are:

$$\pi_1 = \{\overline{a,b,e};\ \overline{c,f}\}$$

$$\pi_2 = \{\overline{a,c};\ \overline{b,f};\ \overline{d,e}\}$$

$$\pi_3 = \{\overline{a,d};\ \overline{b,c,d}\}$$

$$\pi_4 = \{\overline{a,c};\ \overline{b,d};\ \overline{e,f}\}$$

It would appear that six state variables are needed to make an assignment. This number can be reduced by observing that π_2 and π_4 are covered by

$$\pi_5 = \{\overline{a,d};\ \overline{b,d,e,f}\}$$

$$\pi_6 = \{\overline{a,b,c,f};\ \overline{d,e}\}$$

$$\pi_7 = \{\overline{a,b,c,d};\ \overline{e,f}\}$$

By using these three partitions along with π_1 and π_3, an assignment can be made using five secondary variables. But Figure 29 shows an assignment

with only three secondary variables.

Interestingly enough, there is a close relationship between this assignment method and Method #1. In terms of the Boolean matrix introduced in Method #1, Method #2 can be thought of as a reduction of that Boolean matrix on a sectional basis. Each section of the matrix corresponding to the transitions in a single column of the flow table is first reduced, and then the sections are compared with one another in an attempt to achieve further reduction.

Summary Assignment Method #2 consists of constructing a minimum transition time assignment from the column partitions of a flow table. The method may be quite inefficient in terms of the number of secondary variables. But it is easy to obtain and can be useful as an upper bound on the number of variables needed to code the flow table. For some flow tables, the resultant assignment may be considered minimum or near minimum with no further investigation required.

6. Assignment Method #3

Here we consider what Liu (6) describes in his paper to be an upper bound on the number of secondary variables required for a minimum transition time assignment. Liu has shown that a minimum transition time secondary assignment in which the row assignments correspond to an equidistant error-correcting code contains no critical races. For a 2^m-row flow table (m an integer) an error-correcting code of 2^m message words is required. But the code words require $2^m - 1$ bits, which corresponds to a secondary assignment with $2^m - 1$ secondary variables. The assignment may be made independent of the flow table structure, and is usually an efficient assignment only for those flow tables where there are transitions

between all pairs of rows. Fortunately, most practical 2^m-row flow tables, except perhaps 2-row and 4-row tables, do not have transitions between all pairs or rows and hence one seldom needs to resort to this assignment method. For large flow tables the number of secondaries required approach the number of rows in the table.

What one has here then, in Assignment Method #3, is the easiest method of all to apply, but a method that tends to be very inefficient in obtaining an optimum code for most flow tables larger than four rows. As Liu points out though, it is useful as an upper bound assignment.

Caldwell (1) reports a minimum transition time assignment method due to Huffman. It is based on a row set concept with multiple codes assigned to each row of the flow table. It differs from Liu's upper bound in that each transition may be made with a change of only one secondary variable; but the assignment still requires $2^m - 1$ secondary variables for a 2^m-row flow table. Since Huffman's method is similar to Liu's in the number of variables required, Huffman's method will not be considered as a separate assignment method in this paper.

7. Incompletely merged flow tables.

Previous examples were concerned with the coding of merged flow tables. In some instances, one may be interested in an assignment for flow tables that have not been completely merged. Maley and Earle (7) show that if one merges only those rows of the primitive flow table that have the same output, it is sometimes possible to code the rows in such a manner that the output is a function of a single secondary variable, and thus one may save the entire output gating. The result is fewer logic stages and faster propagation time from circuit input to circuit output.

The assignment methods developed in this paper always yield assignments free of critical races and always assign a unique code to each row of a merged flow table. However, if the flow table is not completely merged, assignments Methods #1 and #2 will still be free of critical races but there is no guarantee that they will distinguish all rows of the flow table. Method #3 will always distinguish the rows because the assignment is made independent of the flow table structure. This will be illustrated with the example primitive flow table of Figure 39.

I_1	I_2	I_3	I_4	Z_1Z_2
(1)	2	4	3	00
1	(2)	5	-	01
1	-	6	(3)	00
1	8	(4)	7	10
1	2	(5)	7	01
1	8	(6)	3	00
1	-	5	(7)	11
1	(8)	6	-	10

Figure 39. Example primitive flow table

A merged flow table for Figure 39, subject to the additional constraint that the output of merged rows must agree, is shown in Figure 40. The application of Assignment Method #1 produces the following partitions (corresponding to the list of maximal compatibles) and algebraic solution:

	I_1	I_2	I_3	I_4	Z_1Z_2
a	(1)	2	4	3	00
b	1	(2)	(5)	7	01
c	1	8	(6)	(3)	00
d	1	8	(4)	7	10
e	1	-	5	(7)	11
f	1	(8)	6	-	10

Figure 40. Merge of the primitive flow table in Figure 39

$\pi_A = \{\overline{a,b,e};\ \overline{c,d,f}\}$ ADF + BCE + ACEF + BCDF + ABDE

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

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

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

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

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

If one selects the first term of the algebraic solution, ADF, the result is three partitions that do not distinguish all the rows of the flow table. The product of the partitions π_A, π_D, and π_F is $\pi_A \cdot \pi_D \cdot \pi_F = \{\overline{a};\ \overline{b,e};\ \overline{c};\ \overline{d};\ \overline{f}\} \neq 0$. Therefore, rows b and e will have the same code in the secondary assignment.

On the other hand, the selection of the second term in the algebraic solution, BCE, does give a set of partitions such that their product is the 0 partition. If the variables y_1, y_2 and y_3 code the partitions π_B, π_C and π_E respectively, one may write for output expressions, $Z_1 = y_2$ and $Z_2 = y_3$. If the output code had described partitions π_A and π_F above, it

might have been advantageous to choose the assignment given by the third term of the algebraic expression, even though it involves the use of an additional secondary variable.

If Matrix Reduction Algorithm #2 in Assignment Method #1 is used, one does not have available a selection of alternate assignments and therefore some trial and error may be necessary to come up with an assignment that distinguishes all rows. It is sometimes possible to complete the incompletely specified partitions that may result from Algorithm #2 and thereby arrive at a code. If this doesn't work, one may have to add partitions solely for the purpose of distinguishing some of the rows.

8. Conclusions and summary

Three minimum transition time assignment methods have been developed and illustrated. Assignment Method #1 is best in the sense that it produces codes utilizing a minimum or near minimum number of secondary variables. Its main disadvantage is that it often takes longer to apply Method #1 than the other two. Method #1 produces incompletely specified Boolean matrices of up to perhaps 60 rows for a 12-row flow table. The primary limiting factor in the application of Method #1 is the size of this Boolean matrix. Two algorithms were introduced for the purpose of systematically reducing such matrices. Matrix Reduction Algorithm #1 yields an optimum code but becomes unwieldy for hand computation in the case of matrices with more than about 15 rows. Algorithm #2 can handle matrices with up to about 60 rows but does not guarantee an optimum solution. Experience has shown, however, that an optimum solution is frequently obtained and at least a near optimum solution always results. Matrix Reduction Algorithm #2 is programmable and it is felt that computer

solutions could be obtained for matrices considerably larger than 60 rows. Merged flow tables longer than 12 rows have been rare in the literature. So even without programming, Assignment Method #1, coupled with Matrix Reduction Algorithm #2, can be conveniently used to code nearly all flow tables of current interest.

Assignment Method #2 is easier to apply, but is less efficient in terms of secondary variables, than Method #1. Method #2 utilized the column partitions of a flow table in the secondary state assignment. It was shown that the set of column partitions always produces an assignment free of critical races. The column partitions often contain more than two blocks. The coding of these partitions with more than two blocks is a state assignment problem in itself. For example, just as there are three significantly different ways to code a 4-row flow table, there are also three significantly different ways to code a 4-block partition. A "good" assignment for the column partitions may result in the sharing of secondary variables between column partitions while a "poor" assignment may not. This was illustrated on page 71 where it was discovered that three 2-block partitions could be used to cover two 3-block column partitions with the effect of reducing the number of secondary variables by one. The advantage of Method #2 is that it is relatively quick to apply. The disadvantage is that it is often difficult to determine a "good" assignment for each column partition that will result in an overall "good" assignment for the complete flow table.

Assignment Method #3 is the simplest of all to apply, but for large flow tables the resulting code uses an excessively large number of secondary state variables. For a 2^m-row flow table, $2^m - 1$ secondary

variables are required. The assignment is simply an equidistant error-correcting code, a function only of the number of rows in the flow table and can be assigned independent of the flow table structure.

A good procedure to use in obtaining an assignment is to consider Method #3 as an upper bound for Method #2 and to consider Methods #2 and #3 as upper bounds for Method #1.

The assignment methods were designed for merged flow tables. For unmerged tables, partitions from Method #1 and Method #2 do not necessarily completely specify a code for the flow table. It may be necessary in this case, to add a partition or partitions to distinguish some of the rows.

III. SUMMARY

The paper began with a brief introduction to switching circuits. The sequential circuit design procedure introduced by Huffman was illustrated with an example. Special emphasis was placed on the secondary state assignment aspect of the design procedure. For synchronous sequential switching circuits, the state assignment problem has been defined in the literature as: Given the flow table specifications for the synchronous sequential circuit, select a state assignment that results in a simplest configuration of combinational logic. But in asynchronous circuits, primary consideration must be given to the problem of obtaining assignments that avoid critical race conditions. Only asynchronous circuits have been considered in this paper.

One way to avoid critical race conditions in the design of asynchronous circuits, is to avoid races altogether. Huffman has described general secondary assignment methods that do eliminate all races. He has shown that if minimum transition time is <u>not</u> a requirement, a 2^m-row flow table can always be satisfactorily coded with 2m - 1 secondary state variables. For the case where minimum transition time <u>is</u> a requirement, Huffman describes an assignment procedure which requires 2^m - 1 secondary state variables for a 2^m-row flow table. Both of these assignment methods result in codes that may be assigned to any flow table, independent of its algebraic properties.

This paper has been mainly concerned with the development of minimum transition time assignment algorithms for asynchronous circuits. The resulting assignments are dependent upon the flow table structure. As a consequence, it is often the case that fewer secondary variables are

required to code the flow table than if Huffman's assignment was used.

Partition theory is a useful tool in the development of minimum transition time assignment methods. Theorem 1 conveniently states the necessary and sufficient conditions for such assignments in terms of the assignment partitions. On the basis of this theorem, two assignment methods were developed, Assignment Method #1 and Assignment Method #2. The third assignment method was essentially that of Huffman's and Liu's with $2^m - 1$ state variables for a 2^m-row flow table.

A characteristic of the codes resulting from the first two assignment methods is that all transitions are accomplished by either non-critical races or unit cycles. Therefore, all transitions are accomplished in a minimum amount of time. In the third assignment method, all transitions are non-critical races for Liu's general assignment method, and all are unit cycles for Huffman's.

Interestingly enough, for many flow tables, minimum transition time assignments utilize no more secondary variables than the non-minimum transition time assignments of Huffman which require 2m - 1 state variables for a 2^m-row flow table. So even when minimum transition time is not a requirement, it may be worthwhile to investigate assignments produced by Assignment Method #1 and Assignment Method #2.

It was shown that as flow tables increase in number of rows, the number of essentially different assignments that exist grows at a fantastic rate. Because of this, it seems to be the case that as one tries to achieve a minimum code for larger and larger flow tables, the amount of effort required also increases at a rapid pace. It is a characteristic of the assignment methods in this paper that those methods easy to apply

often require more than the necessary number of state variables, while those that minimize the number of variables tend to become quite long for large flow tables. An attempt was made to illustrate, with a variety of examples, this trade-off between optimum code and algorithm length.

IV. BIBLIOGRAPHY

1. Caldwell, S. H. Switching circuits and logical design. New York, N.Y., John Wiley and Sons, Inc. 1958.

2. Dolotta, T. A. and E. J. McCluskey, Jr. Encoding of incompletely specified Boolean matrices. Western Joint Computer Conference Proceedings 17: 231-238. 1960.

3. Haring, D. R. Some aspects of the state assignment problem for sequential circuits. Massachusetts Institute of Technology Electronic Systems Laboratory Report ESL-R-147. 1962

4. Hartmanis, J. On the state assignment problem for sequential machines. I. Institute of Radio Engineers Transactions on Electronic Computers EC-10: 157-165. 1961.

5. Huffman, D. A. The synthesis of sequential switching circuits. In Moore, E. F., ed. Sequential machines: selected papers. pp. 3-62. Reading, Mass., Addison-Wesley Publishing Co., Inc. 1964.

6. Liu, C. N. A state variable assignment method for asynchronous sequential switching circuits. Journal of the Association for Computing Machinery 10: 209-216. 1963.

7. Maley, G. A. and J. Earle. The logic design of transistor digital computers. Englewood Cliffs, N.J., Prentice-Hall, Inc. 1963.

8. Marcus, M. P. Switching circuits for engineers. Englewood Cliffs, N.J., Prentice-Hall, Inc. 1963.

9. Petrick, S. R. A direct determination of the irredundant forms of a Boolean function from the set of prime implicants. U.S. Air Force Cambridge Research Center Technical Report 56-110. 1956.

10. Unger, S. H. Simplification of state tables. In McCluskey, E. J., Jr. and T. C. Bartee, eds. A survey of switching circuit theory. pp. 145-170. New York, N.Y., McGraw-Hill Book Company, Inc. 1962.

V. ACKNOWLEDGMENT

The author is grateful for the advice and encouragement provided by Professor R. M. Stewart, Jr.