A State Variable Assignment Method for Asynchronous Sequential Switching Circuits*

C. N. Liu

IBM Corporation, Yorktown Heights, N. Y.†

Abstract. This paper describes a method of state variable assignment for asynchronous sequential switching circuits. The method yields state variable assignments with a view towards minimizing the number of state variables and maximizing the operating speed of the circuit. A systematic procedure is presented. An upper bound on the number of state variables for a general r-state circuit is included. The advantages and limitations of the method are discussed.

1. *Introduction*

Sequential switching circuits are commonly classified as being either synchronous or asynchronous, depending upon whether or not the operations are synchronized with some source of fundamental frequency which regulates the entire circuit. The synchronization signal is generally called the *clock* of the circuit. In a synchronous circuit, it is possible to predict the state of the circuit at the time of any given clock period if one knows the initial state of the circuit and its logical characteristics. However, in an asynchronous circuit it is not generally possible to predict the state immediately resulting from a previous state from a knowledge of the logical characteristics of the circuit alone. The resulting state may also depend upon the relative speeds of some of the logical elements which constitute the circuit.

In describing the terminal action of a sequential switching circuit, one method commonly used is a flow table [1–3]. The columns of a flow table are associated with the possible input states to the circuit and the rows represent internal states (see Fig. 1). The entries in the table indicate the next internal state and the output state which result from the given input and internal states. For example, the underlined entry in Figure 1 indicates that if the input state is 11 and the internal state is 2 the output state is now 01, and the next internal state will be 2 (if the input remains 11).

If the internal state component of a flow table entry is found in a row associated with the same internal state, the total state (combination of input and internal state) is a stable one, and the entry may be circled to denote this fact. Once circuit action reaches a stable total state, no further changes of internal state can occur until the input state is changed.

An important step in carrying out the synthesis procedures for sequential circuits is to assign to each row of the flow table of a unique state of a set of binary valued state variables. Once this step is completed, the design problem becomes

* Received August, 1962.

† Formerly with the University of Illinois, Urbana, Ill.

INTERNAL STATE	INPUT STATE			
	00	01	11	10
1	①-00	①-00	①-00	2-00
2	1-00	②-01	②-01	3-01
3	1-00	③-11	2-01	4-11
4	1-00	④-10	3-11	④-10

FIG. 1. A completely specified flow table

identical with the design of a combinational switching circuit. We consider in this paper a method of assigning state variables for asynchronous circuits with a view towards minimizing the number of state variables and maximizing the operating speed of the final circuit.

2. *State Variable Assignment Problem*

In an asynchronous circuit, the assignment of state variables must be made so that each internal transition always leads to a definite and appropriate stable state regardless of the relative speeds of the circuit elements (we restrict our effort here to flow tables without cyclic state variable actions).

Consider the case of the 4-row flow table in Figure 1. There are four internal states to be distinguished. At least $\log_2 4$ $(=2)$ state variables are necessary. In general, for an r-row flow table, at least $\log_2 r$ state variables are necessary. Let S_0 be the smallest integer meeting the condition $S_0 \geqq \log_2 r$ for a given r. We will then have available 2^{S_0} states to assign to r rows. The requirements of a flow table are satisfied if each inter-row transition is accomplished either by a change of internal state in which only one state variable changes, or by a change of internal state in which a multiple change of state variables always leads to a definite destination.

A race occurs whenever a required internal transition involves the change of two or more state variables simultaneously. If the result of a race may lead to false operation of the circuit, we designate it as a *critical race*. Internal states which differ in only a single variable are said to be *adjacent states*. One solution to the problem then is to assign state variables so that each inter-row transition is represented by adjacent states. When this type of assignment is chosen, we call the circuit *totally sequential*. In a totally sequential circuit, only one state variable is excited at any time during a transition. If we define a unit time as the time required for one state variable to change, this time being assumed to be uniform, then an internal transition in a totally sequential circuit will take T units of time when T state variables are required to change.

Another type of state variable assignment is to allow multiple changes of variables. In this case we must be certain that every race condition is non-critical. The internal transition speed of this type of state variable assignment may be the same as that of a circuit where each internal transition requires the change of only a single state variable, because we have provided in this case all possible noncritical races for each transition. In other words, if all races are noncritical we can have a "free-running" condition in internal transitions. Concurrently excited state variables may thus be relaxed in one step. The following sections of this paper describe a method of this type of state variable assignment.

3. *Assignment Method*

Methods for assigning state variables so that the resultant circuit is totally sequential have been well known [1, 3]. We outline here a procedure to assign state variables so that concurrent operations of state variables are provided to increase the speed of the circuit [4].

We restrict here our effort to flow tables without cyclic state variable actions. Indeed, when cycling occurs, the entire set of states involved in the cycling action can be treated as one stable state and then separated with the aid of a suitable additional number of state variables. This procedure may require more state variables than would a direct attack on the entire problem, but it also often leads to a much simpler final circuit.

In the text of this paper, the symbol $[x]$ is used to denote the nearest integer that is larger than or equal to x.

Assume that we have an r-row flow table. Let r_i and r_j be two arbitrary rows in the flow table. Let the state variable assignments for r_i and r_j be $(y_1^i y_2^i \cdots y_n^i)$ and $(y_1^j y_2^j \cdots y_n^j)$, respectively. Suppose that a transition exists between these two rows. Assume that both y_1 and y_2 change in this transition, i.e., $y_1^i \neq y_1^j$, $y_2^i \neq y_2^j$. It is simple to show that there will be no critical races involved in this transition if $y_3^i y_4^i \cdots y_n^i = y_3^j y_4^j \cdots y_n^j \neq y_3^k y_4^k \cdots y_n^k$ componentwise, for $k \neq i \neq j$.

THEOREM 1. *In a flow table, if we consider each column separately and assign a sufficient number of state variables for each column so that no critical races exist in any column, then the combined state variable assignment, i.e., the assignment obtained by putting together all the individual assignments for the various columns, has no critical races.*

PROOF. Consider the flow table shown in Figure 2. Let us examine the column for input X_k. There are three stable entries and one unstable entry in this column. To assure no critical races for transitions within this column we must use two state variables to distinguish the three stable states. Generally, $[\log_2 n]$ state variables are required for a column with n stable entries. If we use this sufficient number of state variables for each column considered separately, then the assignment obtained by combining all the individual assignments has no critical races, because

(1) if a transition exists between rows r_1 and r_2, then their assignments will contain a nonchanging part that is different from the corresponding part in any other row assignment in the flow table;

(2) every transition within a column contains a different nonchanging part

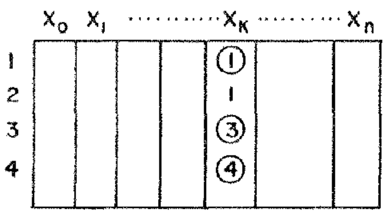

FIG. 2. Typical column of a flow table

in the assignments for the rows involved and therefore the set of intermediate unstable states in each transition is disjoint.

Definition 1. A table listing the state variable assignment for each row of the flow table is called an *assignment table*. This table is obtained, as outlined above, by considering each column separately. The assignment table is an array of 0's and 1's, and ϕ's (don't cares) if the flow table is not completely specified. Each row represents the state variable assignment for the corresponding row in the flow table. A *column* of the assignment table contains one entry from each row of the table. If there are k state variables, then there are k columns.

Definition 2. Given a column S_i of an assignment table, we define the complement of S_i, denoted by $\bar{S}_i$, as the column obtained by changing all the 0's in S_i to 1's and all the 1's to 0's. The ϕ's, if any, are not affected.

The following two definitions are due to Dolotta and McCluskey [5].

Definition 3. Given two columns, S_i and S_j, S_i will be said to *include* S_j if and only if S_j agrees with S_i wherever the latter is 1 or 0. We write this relation as $S_i \supset S_j$. It is obvious that S_i has at least as many ϕ's as S_j.

Definition 4. Column S_i will be said to *cover* column S_j if and only if either $S_j \supset S_i$ or $S_j \supset \bar{S}_i$.

THEOREM 2. *Wherever S_i covers S_j we may always discard S_j.*

PROOF. First consider the case where $S_j \supset S_i$. It is obvious that we may discard S_j. When $S_j \supset \bar{S}_i$, we can always re-assign the state variables for the corresponding column in the flow table so that the S_i column in the assignment table is complemented.

Definition 5. Given two columns, S_i and S_j, there will exist an *intersection column* of S_i and S_j if and only if S_i and S_j agree wherever both S_i and S_j are 1 or 0. This intersection column agrees with both S_i and S_j where they agree with each other, but agrees with either S_i or S_j when the other has entries ϕ.

Definition 6. In a column of a flow table, all $k-1$ unstable entries which eventually lead to the corresponding stable entry, together with the stable entry, form a *k-set*.

We now outline the procedure for making the assignment.

(1) Construct a transition diagram of the given flow table by representing each row by a node and each inter-row transition by a line joining nodes. Solid lines are used to represent those transitions which must be accomplished by a direct transition, i.e., the unstable entry must go to the stable state directly. Broken lines represent the transitions for which there are alternate routes.

(2) If the transition diagram is a complete graph, i.e., every pair of nodes is connected by a line, make the assignment according to Theorem 3 described below. Otherwise, continue to step 3.

(3) Make a state variable assignment for each column of the flow table. The objective is to obtain an assignment table that can be reduced by Theorem 2. This may be accomplished by the following rules:

(a) Start on a column containing the largest number of stable states. Assume that there are M_i stable states in this column. $[\log_2 M_i]$ state variables will be used. Assignment will be made in this column so that a maximum number of direct transition requirements in the transition diagram is satisfied. Assign at first any arbitrary combination to a stable state in this column. Then focus attention on the states in this column which are connected to the first stable state by transition requirements in all columns. Then assign these states to be adjacent to the first one, whenever possible. Repeat this until all stable states in this column are examined.

(b) Unstable states in a column will have the same binary codings as the stable states to which they are terminated. Utilize any extra combinations provided by $[\log_2 M_i]$ state variables for elements in a k-set, i.e., assign adjacent codings to the unstable states that lead to the same stable state. This may generate covering relationships for state variable minimization.

(c) At this point switch to a different column in the flow table. The number of stable states in this column is less than or equal to that in the first column. Again use $[\log_2 M_i]$ state variables. If the number of stable states in this column is less than M_i, we may have a large number of extra combinations to work with. The strategy to apply here is to make assignments such that a maximum duplication of columns exists in the assignment table. A rule for this is to use identical codings, whenever possible, for entries in the same row of the flow table.

(d) For each of the columns containing only one stable entry we can safely set up races, or any other transition patterns which eventually reach the row containing the stable entry. Therefore, we can ignore such columns.

(4) Discard column S_j in the assignment table if it is covered by some column S_i.

(5) If intersection exists between two columns, replace both columns by the intersection.

(6) Compare the number of columns in the reduced assignment table with the upper bound given by Theorem 3 described below. If this number is greater than the bound, discard this assignment and use the assignment generated by Theorem 3.

The problem of state variable assignment can be viewed as a mapping of the rows of a flow table into the vertices of a unit n-dimensional cube. A set of vertices from an n-cube is said to be *equidistant* if the distance between every pair of vertices in this set is the same.

We now show that equidistant error-correcting codes may be applied to assign state variables for any general flow table. We establish an upper bound on the number of state variables needed for any 2^m-row flow table.

THEOREM 3. *A state variable assignment in which the row assignments correspond to an equidistant error-correcting code contains no critical races.*

PROOF. On an n-cube, there are 2^m equidistant vertices with mutual distance $d = 2^{m-1}$ for $n = 2^m - 1$[6]. Suppose there are $n = 2^m - 1$ state variables and the distance between each pair of states is $d = 2^{m-1}$. Then the transition from any arbitrary state r_1 to any other state r_2 is the change of d state variables. Consider another arbitrary state r_3. This state is at a distance d from both r_1 and r_2. Suppose a critical race in the transition from r_1 to r_2 and involving state r_3 exists. Then the $n-d$ state variables which do not change in r_1 and r_2 must have the same values as the corresponding variables in r_3. Therefore r_3 cannot be at a distance d from both r_1 and r_2. This contradicts the hypothesis.

Now it remains to check if the transition between any pair of rows, say r_1 and r_2, will race critically with the transition between another pair of rows, say r_3 and r_4. We must find out if any of the possible intermediate states between r_1 and r_2 would appear also between r_3 and r_4, or any other pair of rows. We know that an equidistant code with distance $d = 2^{m-1}$ is capable of correcting $2^{m-2}-1$ errors and detecting 2^{m-2} errors. Therefore any intermediate states at distances less than or equal to $2^{m-2}-1$ from either r_1 or r_2 cannot appear between any other pair of rows.

We now show that the intermediate states at a distance 2^{m-2} between r_1 and r_2 cannot appear between any other pair of rows. Let V be a vertex at a distance 2^{m-2} from both r_1 and r_2. Then we can represent V as shown in Figure 3. If V is also at a distance 2^{m-2} from another pair of rows, r_3 and r_4, then V must contain $4 \times 2^{m-2}$ bits. Therefore any intermediate states between a pair of rows cannot appear between any other pair of rows.

Methods for generating equidistant error-correcting codes are well-known in the literature [7]. Since an equidistant error-correcting code of 2^m message words requires $2^m - 1$ bits, the number of state variables required for a flow table of 2^m rows will be $2^m - 1$. Note that this number of state variables is sufficient for any general 2^m-row flow table. It is important to note that in this assignment all noncritical races in a transition are used to increase the speed of internal transitions.

4. *Illustrative Example*

To make this presentation more complete, we offer the example shown in Figure 4. First of all, we draw the transition diagram of this flow table (Figure 5). There are three columns with four stable states in this flow table. Now focus attention on the problem of assigning state variables for one of these columns. For the 00 column, two state variables are required. Examination of the transition diagram shows that ③ should be made adjacent to ①, and ② should also be made adjacent to ①. Now make the assignment as follows: ① → 00, ② → 01, ③ → 10, ④ → 11. At this point, examine the 01 column. ⑤ is in the same row as ① in the 00 column. Therefore, choose ⑤ → 01. ⑦ is coded with 01 because this row in the 00 column has been coded with 00, and 01 would make the coding for the uncircled 7 in the second row match the corresponding coding in the 00 column. Make ⑥ → 10, ⑧ → 11 along the same line of reasoning. For the 11 column, use ⑨ → 10, ⑩ → 11, ⑪ → 00, ⑫ → 01. The last column has only two stable states. Let us use ⑬ → 01, ⑭ → 11. Extra combinations 00 and 10 can now be used in a k-set to make more duplications of columns in the assignment table. We can use either 00 or 01 for the uncircled 13's and either 10 or 11 for the uncircled 14's. Figure 6 shows the complete assignment table.

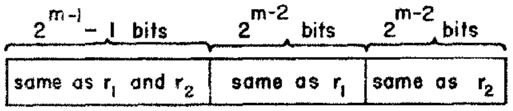

FIG. 3. Bit pattern of a vertex

00	01	11	10	
①	⑤	11	13	A
②	7	12	14	B
③	5	–	13	C
④	7	–	14	D
3	⑥	⑨	14	E
1	⑦	11	13	F
3	⑧	9	–	G
4	8	⑩	13	H
–	6	⑪	14	J
–	8	⑫	13	K
2	–	12	⑬	L
4	–	10	⑭	M

FIG. 4. Flow table for the example

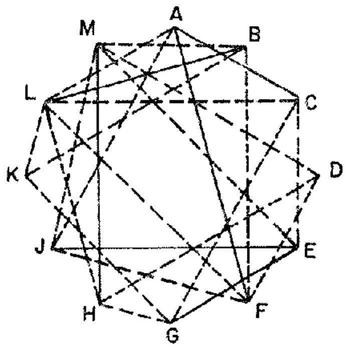

FIG. 5. Transition diagram

12	34	56	78
00	00	00	00
01	01	01	11
10	00	–	00
11	01	–	11
10	10	10	10
00	01	00	01
10	11	10	–
11	11	11	01
–	10	00	10
–	11	01	01
01	–	01	01
11	–	11	11

FIG. 6. Assignment table

There are eight columns in this table. Note that columns 1 and 5 have an intersection, and so do columns 2 and 6, 4 and 8. We may replace these pairs of columns by their corresponding intersections and obtain the reduced assignment table shown in Figure 7.

1	2	3	4	5	
0	0	0	0	0	A
0	1	0	1	1	B
1	0	0	0	0	C
1	1	0	1	1	D
1	0	1	0	1	E
0	0	0	1	0	F
1	0	1	1	1	G
1	1	1	1	0	H
0	0	1	0	1	J
0	1	1	1	0	K
0	1	0	1	0	L
1	1	1	1	1	M

FIG. 7. Reduced assignment table

5. *Conclusions*

A method has been presented for generating state variable assignments for asynchronous sequential switching circuits. This method yields assignments with provisions for concurrent changes of state variables. Reduction of the number of state variables is included in the procedure. However, as is true for most methods of this type, our procedure will be better suited to certain class of problems than to others. As the example illustrated, this method works quite well with incompletely specified flow tables.

Since the method does not do an exhaustive search, we cannot guarantee that the reduction of the number of state variables yields a minimal solution. A comparison of this solution with the upper bound calculated by Theorem 3 can give an idea of how good our solution is.

* * *

Acknowledgment. The author wishes to thank Professor F. E. Hohn of the University of Illinois, Urbana, Illinois, for his guidance and advice during the writing of the doctoral dissertation [4] on which this paper is based.

REFERENCES

1. HUFFMAN, D. A. The synthesis of sequential switching circuits. *J. Franklin Institute 257* (Mar. 1954), 161–191 and (Apr. 1954), 275–303.
2. CADDEN, W. J. Equivalent sequential circuits. *IRE Trans. CT-6* (Mar. 1959), 30–34.
3. CALDWELL, S. H. *Switching Circuits and Logical Design.* Wiley, New York, 1958.
4. LIU, C. N. The state variable assignment problem for asynchronous sequential switching circuits. Ph.D. dissertation, University of Illinois, Urbana, Ill., 1961; Digital Computer Laboratory, University of Illinois, Rept. No. 110, July 3, 1961.
5. MCCLUSKEY, JR., E. J. AND DOLOTTA, T. A. Encoding of incompletely specified Boolean matrices. Proc. Western Joint Comput. Conf. *18* (1960), 231–238.
6. ——. Error-correcting codes—a linear programming approach. *Bell System Tech. J. 38* (Nov. 1959), 1485–1512.
7. MACDONALD, J. E. Design methods or minimum-distance error-correcting codes. *IBM J. Res. Dev. 4* (Jan. 1960), 43–57.