Jurnal Teknologi, bil. 24, Disember 1994 hlm. 45 – 59

LOGIC SIMULATOR: SEQUENTIAL LOGIC MINIMIZATION

MOHAMED OTHMAN
Department of Computer Science
University Pertanian Malaysia
43400, Serdang, Selangor
Malaysia

BAMBANG SUNARYO SUPARJO
Department of Electronic and Computer
University Pertanian Malaysia
43400, Serdang, Selangor
Malaysia

Abstract. Computer Aided Design (CAD) is a tool that comprises of a program written in Turbo Pascal ver 5.0. This CAD tool is use to minimize the states of synchronous sequential logic automatically. In completely specified case, it will give a minimum solution but in the incomplete case, it does not guarantee to produce a minimum solution.

Keywords: state–table, compatible tree, maximal compatible (MC), closure function.

1 INTRODUCTION

In sequential circuits, the outputs at any given time are functions of the external inputs as well as some stored information determined by the previous inputs. It can be described as a combination circuit with a memory section to remember the past inputs and feedback. The variables that represent the past inputs before the present input is applied, are the state variables. The clock is used in synchronous circuits only.

There are two classes of sequential circuit, synchronous and asynchronous. In synchronous type, the input, output and internal states are sampled at definite intervals of time, controlled by the fundamental clock frequency of the system. Since the clock is generally some form of square wave, synchronous circuits are often referred to as pulse circuits, the timing being done by incorporating an element. This type of logic circuit is readily applied to the processing of serial information. Furthers detail see Alamaini A. E. A. [1], Douglas Lewin [3], Hill F. J. *et. al.* [5] and Murai S., *et. al.* [6].

Generally, in synchronous logic design, there are several steps to be followed:

1. Functional description
2. State diagram
3. State table
4. State minimization
5. State assignment
6. Implementation.

The main objective of this work is to develop a CAD tool functionally as a logic simulator. With this tool, it can help the engineer to reduce the following problems such as labour reduction, timescale reduction, error reduction, design integrity and reproducible results.

2 AN ALGORITHM FOR STATE-TABLE REDUCTION

The State-Table Reduction is used instead of State Minimization, this is because the algorithm described is not guarantee in the incompletely specified case in order to produce a system requiring the minimum possible number of states. A tractable technique for determining the true minimum solution has not yet been found. In the completely specified case the algorithm always find a minimum solution.

The algorithm is easily described by means of an example, see Table 2.1 below:

Table 2.1 State table of problem

	Next state				Output			
Present State	00	01	10	11	00	01	10	11
1	×	×	6	9	×	1	×	1
2	7	×	×	×	1	×	0	×
3	×	×	×	2	×	0	×	1
4	×	4	9	×	×	×	×	×
5	8	1	×	×	1	0	0	×
6	×	1	×	3	×	×	×	1
7	3	3	×	9	×	×	0	×
8	×	5	×	4	×	×	×	0
9	7	9	4	×	0	×	1	×

Based on this example, the processes of reduction can be described in the following four subsections which outline the steps involved.

2.1 Determination of Compatible Pairs

The algorithm commences with the determination of all compatible pairs of states. This determination is an implication chart-like method which involves the following steps.

Step 1:

For each row of the table in turn search the lower rows for states with incompatible outputs. In Table 2.1, commence with state 1 and note that the outputs of state 1 are incompatible with those of states 3 and 5. A square that corresponding to the incompatible pair (3–1 square for state 1 and 3, and 5–1 square for states 1 and 5) will be marked in the implication chart as shown in Table 2.2 below:

After marking those states whose outputs are incompatible with the outputs of state 1, a search is made through the state 2 below in the state table to locate any states whose

Table 2.2 First step in formation of chart

2								
3	×							
4								
5	×							
6								
7								
8								
9								
	1	2	3	4	5	6	7	8

outputs are incompatible with those of state 2. In this case, the outputs of state 9 are incompatible with those of state 2. The corresponding square, 9–2 square is marked in the chart. The method proceeds in the obvious fashion until all pairs of states with incompatible outputs have been marked and the resulting chart is shown in Table 2.3. This complete Step 1 in the determination of incompatible pairs of states.

Table 2.3 Chart after making all pairs of states with incompatible outputs

2								
3	×							
4								
5	×							
6								
7								
8			×		×			
9		×			×		×	
	1	2	3	4	5	6	7	8

Step 2 of the process uses the information obtained in Step 1 to identify further incompatible pairs. For each square of the chart in turn (from top to bottom and then from left to right) check whether the square has been marked or not. If the square is unmarked then get the next state pair for every input condition, and then check whether the states of the next state are compatible or not by examining the appropriate square. If that square is marked, it implies that the pair of current states are incompatible, and thus the current square must be marked.

Step 2:

Search the unmarked square of the chart and then get the next state pair for every input condition of that square. After that check if any states of the next state are compatible or not; this is done by examining the appropriate square. If its marked then it is incompatible. If any such incompatibility is found, the current square must be marked to indicate that these two states are incompatible.

Thus in term of our example, Step 2 would commence by examining the 2–1 square. Since there are no next state pair, therefore, just skip this square. This process is carry on until 9–8 square. Let say that it has come to 7–3 square, the corresponding next state pair is only one, that is 9–2 (when input is 11). Therefore, the 9–2 square will be examined whether it is marked or not. Since states 2 and 9 are incompatible, and the 9–2 square has already been marked, this implies that states 3 and 7 are incompatible, and 7–3 square must be marked. Table 2.4 shown the chart after completion of Step 2.

Step 3:

Repeat Step 2 until no new incompatibilities are revealed.

On completion of Step 3, the final chart is as shown in Table 2.5. From the chart, the unmarked square is representing the compatible states. This completes the process of determination of compatible pairs; the next stage in the algorithm requires the setting-up of compatibility trees.

Table 2.4 Chart after completion of step 2

2								
3	×							
4								
5	×							
6								
7			×		×	×		
8			×		×	×		
9		×			×		×	×
	1	2	3	4	5	6	7	8

Table 2.5 Final chart

2								
3	×							
4								
5	×							
6								
7		×			×	×		
8			×		×	×		
9		×			×	×	×	×
	1	2	3	4	5	6	7	8

2.2 Construction of Compatibility Trees

The construction of compatibility trees is used to determine the set of maximal compatibles (MC). Each tree is constructed by commencing with a single state, which form the root of the tree, and then growing branches which link pairs of compatible states, see Writh N.

[11]. In the example commence with state 1 and, by reference to final chart (Table 2.5), note that state 1 is compatible with states 2, 4, 6, 7, 8, 9 so that, with state 1 as the root, a compatibility tree can be drawn which has the six branches shown in Figure 2.1. Note that the leaves of the tree represent those states that compatible with its ancestor, state 1.

The process can now be taken further by creating the subtrees from the leaves. The creation of the subtree involves the following steps:

Step 1:

From left-most leaf, check whether it's siblings are compatible with it or not by referring to the final chart (Table 2.5).

For this example, start from state 2 and check whether it's siblings at the right-hand side of state 2, (i.e. states 4, 6, 7, 8, and 9) are compatible with state 2 or not. By referring to final chart, note that state 2 is compatible with states 4, 6 and 8, therefore from state 2, a subtree can be drawn which has three branches as shown in Figure 2.2.

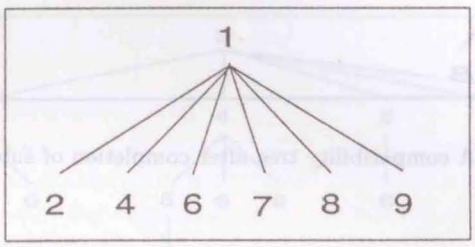

Fig. 2.1 Compatibility tree for pairs of states including state 1

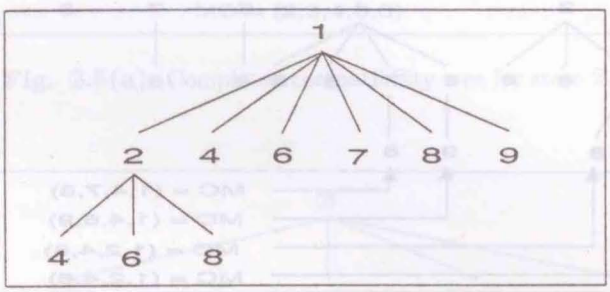

Fig. 2.2 A compatibility tree after step 1

Step 2:

By using the same manner, build another subtrees until there is no more subtree. As states 4 and 6, and states 4 and 8 are compatible, but states 6 and 8 are incompatible, therefore, the final subtree of state 2 can be drawn as shown in Figure 2.3. Note that all the leaves are compatible with its ancestors.

Step 3:

Move to the nearest sibling at right-hand side, and build another subtree. After that repeat step 1 and step 2 on the above until it has reached to the right-most sibling.

On completion of step 3, the final form of the compatibility tree is shown in Figure 2.4.

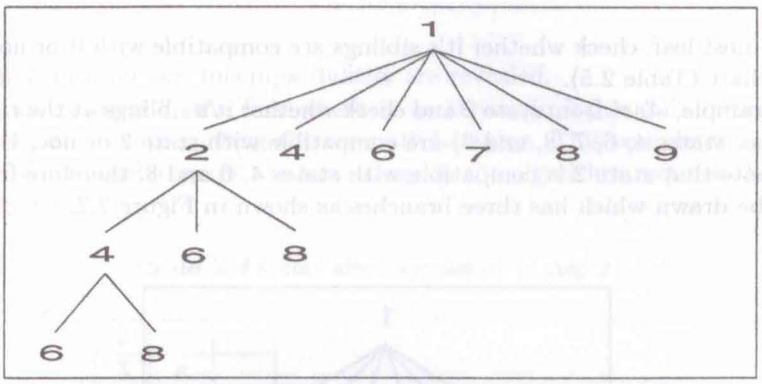

Fig. 2.3 A compatibility tree after completion of subtree of state 2

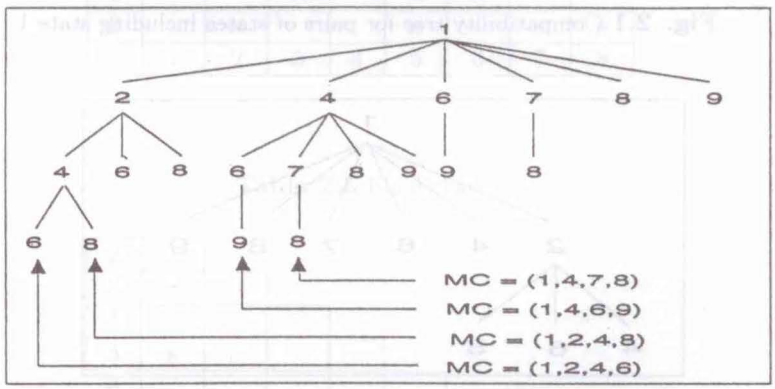

Fig. 2.4 Completed compatibility tree for state 1

After the completion of a compatibility tree, a preorder traversal search will be used to get all the MCs, see Writh N. [11]. Whenever the traversal reach to a leaf, it will record down the leaf and its ancestors as a set of compatible states, then compare it with the existed MCs. If it is not a subset of one of the MCs, automatically, it will become an MC.

For this example, the compatibility tree in Figure 2.4, has generated four MCs, which have been pointed by an arrow. The remaining leaves of the tree do not represent MCs because each represents a set of states that is a subset of one of the pointed (maximal)

sets. All MCs generated by the tree necessarily contain state 1 as the root of the tree. The procedure now continues with a search for other MCs.

The next logical step is to search for MCs containing state 2, but in carrying out the search, bear in mind that some have already been generated by means of the tree in Figure 2.4. In order to avoid generating these MCs again, the tree with root 2 is constructed using only those states located below state 2 in Table 2.5. The resulting tree is shown in Figure 2.5(a). Note that only one MC has been generated by this tree. All the other sets of compatibles are either subset of (2, 3, 4, 5, 6) or subsets of a MC generated by the tree in Figure 2.4.

The process continues by constructing compatibility tree for the remaining states. For each tree, only those states located below the root state (in Table 2.5) are considered; as explained above, this avoids duplication of MCs. The resulting compatibility tree are shown in Figure 2.5(b)–(h).

There are six MCs exist for this problem. Note that it is essential for the full set of compatibility trees be generated to ensure that no MCs are missed.

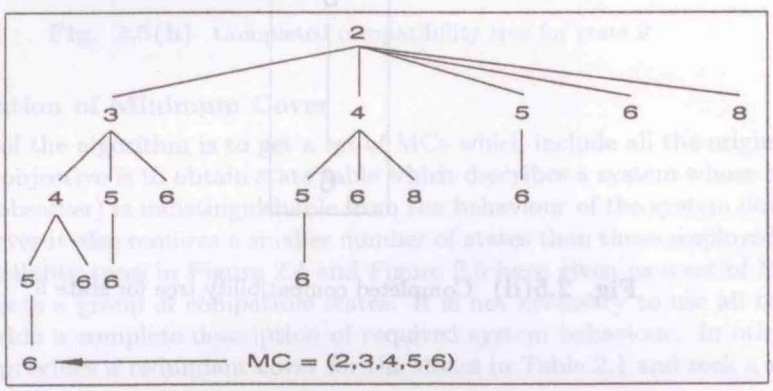

Fig. 2.5(a) Completed compatibility tree for state 2

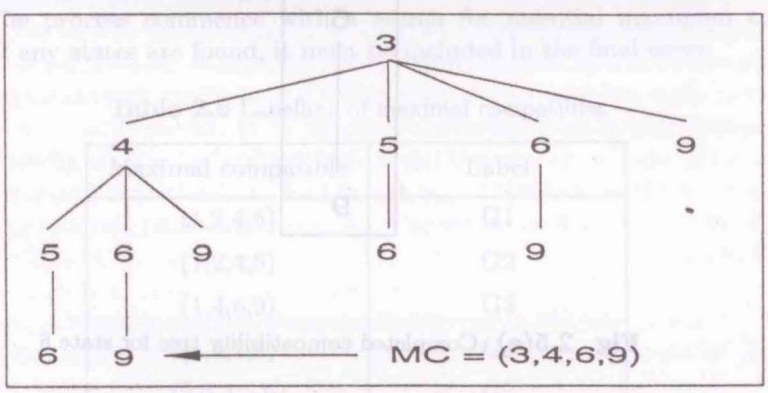

Fig. 2.5(b) Completed compatibility tree for state 3

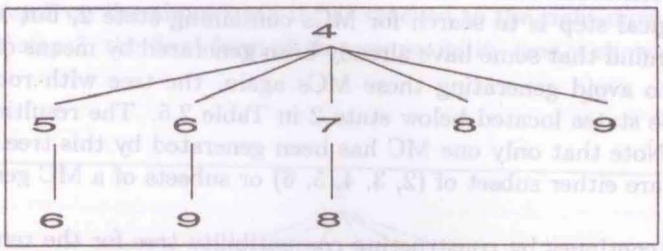

Fig. 2.5(c) Completed compatibility tree for state 4

Fig. 2.5(d) Completed compatibility tree for state 5

Fig. 2.5(e) Completed compatibility tree for state 6

Fig. 2.5(f) Completed compatibility tree for state 7

8

Fig. 2.5(g) Completed compatibility tree for state 8

9

Fig. 2.5(h) Completed compatibility tree for state 9

2.3 Determination of Minimum Cover

The next stage of the algorithm is to get a set of MCs which include all the original states. Recall that our objective is to obtain state table which describes a system whose behaviour (to the outside observer) is indistinguishable from the behaviour of the system described in Table 2.1. However it also requires a smaller number of states than those employed in Table 2.1. The compatibility trees in Figure 2.4 and Figure 2.5 have given us a set of MCs, each of which represents a group of compatible states. It is not necessary to use all these MCs in order to provide a complete description of required system behaviour. In other words, the set of MCs provides a redundant cover for the states in Table 2.1 and seek a minimum cover.

Firstly, commence by labelling the MCs. Any labelling will arbitrary assigned labels as shown in Table 2.6. The important features of the determination of a minimum cover should become clear in the following preamble.

Preamble: The process commence with a search for essential maximum compatibles (EMC's) and, if any states are found, it must be included in the final cover.

Table 2.6 Labelling of maximal compatibles

Maximal compatible	Label
(1,2,4,6)	G1
(1,2,4,6)	G2
(1,4,6,9)	G3
(1,4,7,8)	G4
(2,3,4,5,6)	G5
(3,4,6,9)	G6

In this example there are two EMCs, via G4, which provides the only cover for state 7, and G5 which provides the only cover for state 5. Thus, these two MCs must be included in the final cover.

Now note that each of the EMCs is made up of a group of states and that, according to the state table (Table 2.1). Each state has four next states, which each next state corresponding to a particular input. One of the conditions for a group of state to be compatible is that the next state of each state must be in the group.

The EMC G4 has the next states shown in Table 2.7 (these next states are taken directly from the state table). The table indicates that if the group of states 1,4,7,8 is compatible, then the group (4,3,5), (6,9), and (9,4) must also be groups of compatible states.

Table 2.7 The next for EMC G4

Present state	Next state			
	00	01	10	11
1	×	×	6	9
4	×	4	9	×
7	3	3	×	9
8	×	5	×	4

The objective is to obtain a minimum state description of the system using MCs as states. MC G4 has to be included in the state description (it provides the only cover for state 7) but now from Table 2.7 that the final cover must allow the system to move from state G4 to states with other compatibility properties. There are such states as follow:

a. A state compatible with state 3, 4, 5
b. A state compatible with states 6, 9
c. A state compatible with states 4, 9

From Table 2.6, MC G5 constitutes the only state compatible with the three states 3,4, and 5. This implies that G5 should be included in the final cover and obviously it should be included because it is an EMC.

There are two MCs that include states 6,9 via G3 and G6. In addition G3 and G6 include the pair 4,9 so that inclusion of one of these MCs is the final cover (along with G5). This will allow satisfaction of the next-state requirements of the G4.

How to decide whether to select G3 or G6 for the final cover?. In general case, this kind of selection of one MC rather than another will lead to a better solution (i.e: one with fewer states). We are not aware of any other selection procedure that guarantees a minimum-state except as follows:

Rule 1. Pick up the MC that cover the largest number of states.
Rule 2. If there is more than one candidate for selection under Rule 1, then select from these candidates the MC that has the smallest closure function.

Actually, these rules were applied at the very beginning (ie: in the selection of the EMCs)

because it turns out that the order in which EMCs are selected can influence the number of MC's in the final cover.

The procedure: From the example, there are two EMCs, Rule 1 tells us to select G5 first. Having selected G5, then check the state table (Table 2.1) to determine the next-state condition implied by G5. These conditions are displayed in Table 2.8, where selection of G5 requires that the final cover contains a state in which 7 and 8 are compatible, a state in which 4 and 1 are compatible and a state in which 2 and 3 are compatible.

Table 2.8 The next for EMC G5

	Next state			
Present state	00	01	10	11
2	7	×	6	×
3	×	×	×	2
4	×	4	9	×
5	×	1	×	3
×	1	×	×	3

The first of these requirement is satisfied if selection of G4 (1,4,7,8) is selected for the final cover. The second requirement (a state in which 4 and 1 are compatible) can be satisfied by selecting any one of the MCs G1, G2, G3 or G4. The final requirement will be satisfied by selection of G5 itself.

Thus, following selection of G5 will cause the selection of G4 and either G1 or G2 or G3 or G4. This can be written as a Boolean expression as G5 ⇒ G4. (G1+G2+G3+G4) which should be read *selection of G5 implies the selection of G4 and either G1 or G2 or G3 or G4*. The right-hand side is called the *closure function* for G5. In this case, any elementary rule of Boolean algebra that X.(X + Y) = X, the closure function reduces to a single MC, G5 ⇒ G4 so that selection of G5 implies that G4 must also be selected.

Knew that G4 had to be selected (it is an EMC), but in general, the closure function indicates that other MCs, not previously considered, should also be added to the cover. In this case, the closure function tells us that G4 should be added to the cover next and, whenever an MC is added to the cover, the closure function must be drawn up in order to see if it requires the addition of further MC to the cover.

From investigation of the next-state requirements of G4 (in the preamble to the solution of this problem), its closure function can be written as G4 ⇒ G5.(G3+G6). Since G5 has already been selected, decision has to be made whether to add either G3 or G6 to final cover. Rule 1 does not assist us in the selection because both G3 and G6 have the same number of states. Therefore, apply Rule 2 and draw up the closure function for each of the MCs. For G3, the next state requirements imply the need for states in which (1,4,9), (4,6,9), and (3,9) form compatible groups. Thus, the closure function for G3 is G3 ⇒ G3.(G3 + G6).G6 = G3.G6 and since looking for the next-state requirements following selection of G3, the closure function requirement can be written as G3 ⇒ G6.

Similarly, for G6 the closure function is G6 ⇒ G3.(G3 + G6).G5 = G3.G5 and since G5 has already been selected, the closure function requirement is G6 → G3.

Thus, found that if selection of G3 will cause the selection of G6 and vice versa. Hence both G3 and G6 are selected for the cover.

The cover now contains G3, G4, G5, and G6 as MCs. While G4 and G5 are both EMCs and, to satisfy their next-state requirements, the G3 and G6 had to be included in the cover. Addition of G3 and G6 to the cover does not generate any next-state requirements that cannot be satisfied by the four MCs G3, G4, G5 and G6. Consequently, these four MCs are said to form a closed group. This is the origin of the term closure function.

Having selected a closed group of MCs for the cover, now check if any states remain to be covered. In this case, the MCs G3, G4, G5 and G6 provide a complete cover and the procedure terminates.

When a closed group of MCs has been determined and the group does not provide a complete cover, the procedure continues with a search for a minimum cover of the remaining states. It can happen that one or more of the EMCs has not yet been included in the cover and the procedure should recommended with this consideration. In other cases there may be quasi-essential MCs (i.e: MCs that provide a unique cover for one of the remaining states) and the procedure would recommended with these. The third possibility is that there are neither, nor quasi-essential MCs remaining and the procedure would then recommended by implementing Rule 1 and Rule 2.

The basic steps of the procedure are as follows:

1. Select the largest MC as an initial EMC. If there are more than one candidates, then use Rule 2 to select an initial an EMC.
2. Generate the closure function for the selected MC.
3. Satisfy the closure function by adding the minimum possible number of MCs to the cover; use Rule 1 or 2 where necessary.
4. For each MC added to the cover, return to step 2.
5. Once a closed group of MCs has been found, check if all states are covered by the group and any groups previously added to the cover. If they are, go to 8.
6. Check any remaining EMCs. If any are found go to 1.
7. Select an MCs using Rule 1 or 2; go to 2.
8. End.

2.4 Relabel the State Table

The last stage of the algorithm is relabelling the state table. The MCs provide a complete and closed cover for the states of the system. Each row of the new state table represents a group of compatible states and the entries in given row are obtained by merging the corresponding entries from the original state table. For instance, G3 represents the group of states 1,4,6,9 and so in the new state table, the row corresponding to G3 is drawn up by merging the entries in rows 1,4,6 and 9 of the original state table. The new state table is shown in Table 2.9.

The final step is to relabel the new states. Note that some of the next state entries can be identified with more than one MCs (for instance, (4,6,9) can be identified with either G3 or G6). In such case, it does not matter which choice is made. Thus, there is no general and a unique relabelling scheme. Table 2.10 displays the relabelled state table from the Reduced state table as in Table 2.9.

Table 2.9 Reduced state table for example

Present state	Next state				Output			
	00	01	10	11	00	01	10	11
G3(1,4,6,9)	7	1,4,9	4,6,9	3,9	0	1	1	1
G4(1,4,7,8)	7	3,4,5	6,9	4,9	x	1	0	0
G5(2,3,4,5,8)	7,8	1,4	9	2,3	1	0	0	1
G6(3,4,6,9)	7	1,4,9	4,9	2,3	0	0	1	1

Table 2.10 Relabelled state table

Present state	Next state				Output			
	00	01	10	11	00	01	10	11
1	2	1	1	4	0	1	1	1
2	3	3	1	1	x	1	0	0
3	2	1	1	3	1	0	0	1
4	2	1	1	3	0	0	1	1

3 SYSTEM DESIGN

This system is implemented with Turbo Pascal 5.0 on an IBM PC compatible, see Turbo Pascal ver. 5.0 Reference Guide [9] and Turbo Pascal User's Guide [10]. This language was chosen because of the following reasons:

a. It is a structured high-level language
b. It has an amazing compilation speed
c. It provides a library of powerful standard units
d. It supports separate compilation using units
e. Its built-in project management performs automatic recompilation of dependent source files including units and others.

From the system flowchart, see Figure 3.1, this system provides the help and error messages, user guide, softcopy and hardcopy facilities, and screen display.

During the operation, help messages are available in any input section as a guidance so that user can use this CAD tool easily. Beside that, if any error was encountered during the operation, the system will display the corresponding error message so that the user can do the appropriate correction if possible.

The input (contents of state table) for this system can be retrieved either from input file or done manually. User can save the input data into a file (if the user keyed in the input

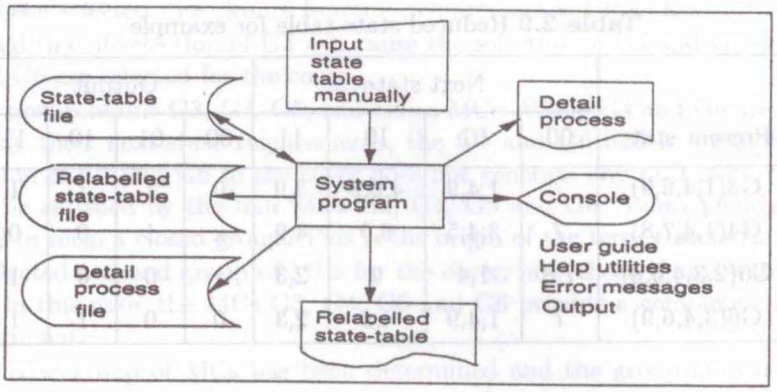

Fig. 3.1 System flowchart

manually) so that any modification can be done easily later.

If the user wants to know about the detail processing steps (algorithm) of the system, he may select the appropriate options, then view it at the required output device. The user guide is used to teach the user how to use this CAD tool.

4 CONCLUSION AND FUTURE WORKS

This work has been successful in achieving its objective and fulfil the CAD basic requirements even though the program produced have some limitations.

Further expansion of this work are; 1) Once the reduction state table has been obtained, the next step in the design procedure is to allocate a binary code to every internal state, or row, in the table so that input equations for the storage elements (JK or SR bitstables, etc) may be derived – state assignment. 2) In all practical cases (say circuits with more than five states) to attempt for determine an optimum assignment by enumeration methods is obviously impossible and some form of algorithm technique (programmed for a computer) must be used. Therefore, it is suggested that an automatic state assignment program be produced.

REFERENCES

[1] A. E. A Alamaini, *Electronic Logic System*, Prentice-Hall International, UK, 1986.
[2] Douglas Lewin, *Computer-Aided Design of Digital System*, Grane, Russak and Company Inc, New York, USA, 1977.
[3] Douglas Lewin, *Design of Logic System*, Van Nostrand Reinhold, UK, 1986.
[4] Gerald Musgrave, *Computer-Aided Design of Digital Electronic Circuits and System*, North Holland Publishing Company, Netherlands, 1979.
[5] F. J. Hill & G. R. Peterson, *Switching Theory and Logic Design*, John Wiley and Son, New York, 1981.
[6] S. Murai & H. Matsushita & K. Enomoto, *Logic Simulation Programs*, Advances in CAD for VLSI **2** (1986), 135-163.
[7] R. M. McDernott, *Computer-Aided Logic Design*, Howard W. Sams and Co. Inc., New York, 1985.
[8] E. J. McCluskey, *Logic Design Principles*, Prentice Hall Inc, New York, 1986.
[9] —, *Turbo Pascal ver 5.0 Reference Guide*, Borland International, USA, 1988.
[10] —, *Turbo Pascal ver 5.0 User's Guide*, Borland International, USA, 1988.
[11] N. Writh, *Algorithms + Data Structures=Programs*, Prentice Hall, Inc, USA, 1988.