A New Method For The State Reduction of Incompletely Specified Finite Sequential Machines

M.J. Avedillo J.M. Quintana J.L. Huertas

Dpto. de Electrónica y Electromagnetismo, Universidad de Sevilla
Dpto. de Diseño de Circuitos Análogicos,
Centro Nacional de Microelectrónica, 41012 Sevilla, Spain

ABSTRACT

A new method for the state reduction of incompletely specified Finite Sequential Machines is proposed. Fundamental theorem of minimization theory states that, given an incomplete state table, another state table specifying the same external behavior corresponds to each closed set of compatibility classes which covers all internal states of the given table. The new heuristic algorithm builds up a closed cover for a given state table selecting maximal compatibles (MCs) one by one until both covering and closure requirements are satisfied. Near-minimal solutions are so incrementally generated. The process is dynamic as the consequences of adding a particular MC are precisely determine.The new algorithm is designed for speed and has proven to be extremely valuable in situations where fast but good optimization is required. The algorithm has been programmed and results on a wide set of machines shown.

INTRODUCTION

In the framework of computer-aided synthesis of Finite Sequential Machines (FSMs), a great attention has been paid to the state assignment of a FSM, i.e., to the problem of selecting a binary representation for the internal states of the machine. It has been so because of its dramatic influence on the overall area occupation of the final circuit as well as on the time required for the design process. However, there is a previous step in the design of a FSM which also might be responsible of a significant wasting both in silicon area and CPU time, if the designer is not aware from it. We are referring to the elimination of redundant states in the description of the FSM being manipulated in the design process. A reduction of the number of states may correspond to a reduction of the number of bits that is needed for the state coding. Moreover, reducing the number of states corresponds to decreasing the number of transitions of the sequencing functions (and eventually, to reducing the number of implicants in a two level logic realization, for example). Although there is no assurance that circuit complexity reduces with state minimization [1], it has been noted that this process may become necessary and bring area minimization when starting with high-level initial descriptions which are then translated into state tables [2]. Those programs usually introduce a lot of states, which are redundant and must be eliminated in what is classically called the state reduction phase [3,4].

In this communication we will show the need of considering this previous state reduction phase, we will discuss the presently known state reduction methods and finally, we will propose a new heuristic algorithm which achieves a satisfactory compromise between the quality of the solutions and the required time to obtain such solutions.

A PRELIMINARY EXAMPLE

In order to show how the state reduction may influence the result of the design process let us consider a classical example taken from the literature and corresponding to a medium size machine [5]. *Table 1-a* shows the state table representation of an incompletely specified machine of 22 states. This machine has been proved to have an equivalent representation with only 9 states, as is shown in *Table 1-b*. *Table 2* depicts figures for the area obtained and the time invested when both state tables are implemented using a PLA and a dynamic register. An automatic state assignment program (PRASES [6]) has been used. From *Table 2* it should be clear the difference between applying the program to a reduced machine and to the original description.

PREVIOUS WORK

Several methods have been reported in the past to deal with the problem of state reduction [7-12]. In general these techniques involve two steps: 1) generation of compatibility classes (prime compatibles [9], prime closed sets [12], etc.), and 2) extraction of a minimal closed cover. There are drawbacks related to both steps. Concerning the first one, the number of those compatibility entities might be too large precluding an efficient generation. Moreover, the second step implies the solution of a minimum closed covering problem, which is known to belong to the class of NP-complete problems. The consequence is that exhaustive algorithms are impractical for VLSI circuits even when the machine size is relatively small.

Traditionally, two kinds of solutions have been proposed to deal with second step, one of them being based on covering tables, the other being based on a mathematical programming formulation. In both cases, prime compatibles were usually preferred to maximal compatibles since the former ones ensure the absolute minimality of the solution. The price to be paid is the computational cost. Tabular approaches are impractical even for small size machines. Nevertheless, they are systematically reported in textbooks, the mathematical programming approaches being only considered in a few papers [9].

Luccio and Graselli [9] converted the closure and covering problem into a mathematical programming one and proposed to use standard solution methods for such a kind of problems. Namely, the authors

2024/90/0000/0552$01.00©1990 IEEE

formulated this selection stage as an integer restricted linear program by associating with every prime class R_i an 0-1 integer variable r_i. R_i is selected if $r_i = 1$ and not selected if $r_i = 0$. Unfortunately, we are still dealing with a NP-complete problem. Heuristic algorithms have been developed to cope with this. These algorithms do not attempt to determine the absolute minimum, so the compatibility class set on which they are based is not so critical (it is well known that it is not guaranteed [13] that a minimum closed cover composed uniquely of maximal compatibles exists, for example). Bennets in [10] proposes a reduction algorithm based on the maximal compatible sets, MCs. Using this subset of the prime compatible sets reduces (in some cases drastically) the number of compatible candidates. The procedure consists of selecting one of the essential (or quasi-essential) MCs and attempting to satisfy its closure requirements (generating one of the smallest set of MCs that satisfies the violated closure requirements for the MC selected). The result will be a closed set of MCs that may be or may not provide full cover on the initial set of states. The procedure is repeated until a full cover is performed.

We have carried out many experiments using Bennets' approach as well as several well-stablished general solution methods for linear programming problems. *Table 3* shows some results for four different machines to illustrate our experience. The first machine, FSM1, is the one in *Table 1-a*. The second one, FSM2, has also been selected from the literature [13]. The machines FSM3 and FSM4 have been randomly generated. We have applied to those machines three different state reduction methods. The one referenced as G1 in *Table 3* corresponds to the first solution obtained by a linear programming (LP) integer problem general method [14]. We include solutions using maximal compatibles, MCs, as well as those using prime compatibles, PCs, as potential members of a minimal closed cover. The column referenced as G-H corresponds to the same previous general method with a speed-up strategy added [15]. This strategy allows intermediate solutions, perhaps no minimal, but with a bound on the difference from the minimal for the closed covering problem. This is, a given tolerance ε_0 is assumed which causes an exit if an approximate solution is found that is guaranteed to be within a distance ε_0 from the best. We also report the solution obtained by the algorithm in [10] and the minimum. The CPU times, in seconds, corresponds to a SUN workstation 3/260. Entries filled with the label "Not P.", standing for *not practical*, apply for a method in *Table 3* when the CPU time was found to be at least three orders of magnitude higher than G1.

From *Table 3* we can observe that using PCs or MCs as starting compatibility classes does not seem to be too critical if we are trying to get near minimal solutions rather than the absolute minimum. It should be clear that the solutions obtained with G-H have a bound on the difference from the minimal, but time requirements can be too large even for medium size machines. Finally, the algorithm [10] produces nice results when is applied to small FSMs but we have realized that for many machines produces worse solutions than G1. In summary, we have realized that none of these approaches is a clear candidate to be inserted on top of a design tool we are implementing for the automatic design of FSMs. Hence, we have decided to introduce a new alternative.

	I1	*I2*	*I3*	*I4*
s1	*s1,–*	*s2,2*	*–,1*	*s8,2*
s2	*–,–*	*s3,2*	*s5,1*	*s9,–*
s3	*–,–*	*s4,–*	*s6,1*	*s10,2*
s4	*–,–*	*–,–*	*–,–*	*s11,2*
s5	*s1,–*	*–,2*	*–,–*	*–,–*
s6	*s1,–*	*s3,2*	*s12,1*	*–,–*
s7	*–,–*	*s4,2*	*–,1*	*–,–*
s8	*–,–*	*s2,–*	*–,–*	*s14,2*
s9	*s7,2*	*–,–*	*–,–*	*s15,2*
s10	*s7,–*	*–,–*	*s6,1*	*s16,2*
s11	*s1,2*	*s4,1*	*s5,–*	*s17,2*
s12	*s1,2*	*s2,1*	*–,–*	*–,–*
s13	*–,–*	*s3,1*	*–,–*	*–,–*
s14	*–,–*	*–,–*	*s13,2*	*s18,2*
s15	*s7,1*	*–,–*	*s12,–*	*–,–*
s16	*s7,–*	*s2,1*	*–,2*	*–,–*
s17	*s1,1*	*s3,1*	*–,–*	*–,–*
s18	*s1,–*	*s4,–*	*–,2*	*s19,–*
s19	*–,1*	*–,1*	*s5,–*	*s20,2*
s20	*–,2*	*–,1*	*s5,–*	*s21,2*
s21	*–,2*	*–,2*	*s5,–*	*s22,2*
s22	*–,1*	*–,2*	*s5,–*	*s14,2*

Table 1-a

	I1	*I2*	*I3*	*I4*
s1	*s1,–*	*s1,2*	*s3,1*	*s3,2*
s2	*s1,–*	*s1,2*	*s3,1*	*–,–*
s3	*s1,1*	*s1,1*	*s2,1*	*s4,2*
s4	*s1,1*	*s1,1*	*s3,2*	*s5,2*
s5	*s1,–*	*s1,–*	*–,2*	*s6,–*
s6	*–,1*	*–,1*	*s1,–*	*s7,2*
s7	*–,2*	*–,1*	*s1,–*	*s8,2*
s8	*–,2*	*–,2*	*s1,–*	*s9,2*
s9	*–,1*	*–,2*	*s1,–*	*s4,2*

Table 1-b

	PRASES			
	nv	tp	t	Area (λ^2)
#A (22 states)	5	38	5.2	1428
#B (9 states)	4	17	2.1	725

Time in seconds on a SUN 3/260

Table 2

FSM	PARAMETERS				G1[14]			
					using MCs		using PCs	
	ns	ni	#MC	#PC	ns_1	t	ns_2	time
FSM1	22	4	30	261	14	0.6	70	58.8
FSM2	9	4	10	70	6	0.1	5	0.7
FSM3	43	3	27	27	21	0.7	21	0.7
FSM4	43	3	25	25	8	0.5	8	0.25

FSM	G-H [14],[15]				B. et al. [10]		MINI MUM
	using MCs		using PCs				
	ns_3	t	ns_4	t	ns_5	time	
FSM1	11	443.5	10	Not P.	12	0.6	9
FSM2	5	0.4	4	Not P.	6	0.1	4
FSM3	9	13.9	9	13.9	8	0.4	8
FSM4	8	0.25	8	0.25	12	0.7	7

ns : *state number of the original FSM*
ni : *input number of the original FSM*
#MC : *MC number of original the FSM*
#PC : *PC number of original the FSM*
ns_i : *state number of the reduced FSM*

Table 3

A NEW APPROACH

In this section, we present the state reduction algorithm we have developed. It is described using *Pidgin-C* as follows:

```
main()
/* state reduction algorithm based on maximal
compatibles (MCs) */
{
CCSS = Get_MC(T);
Q = Get_constraints(CCSS, T);
CC = {∅};
while(CC ! = closed cover)
{  if(CC ! = closed)
   CC = + Select_CC(Violated Closure Constraints);
   else
   {CC = + Select_CC(Violated Covering
   Constraints);
      }
   }
   T' = Get_Table(CC);
}
Select_CC(Violated Type Constraints)
{  Cand = Most_restrictive(Violated Type
   Constraints);
   for(each CC in Cand)
      Compute_C();
      return(Mi maximizes C);
}
```

The state reduction is achieved by processing the state table T, from which the set of maximal compatibles, $CCSS$ is obtained. Then, a table Q with the covering and closure constraints is derived by ***Get_constraints()***. The closed cover (CC) is initialized to the empty set. CC is built up by adding MCs one by one until no closure or covering constraint is violated. To do this, if CC is not closed, procedure ***Select_CC()*** is called with the set of violated closure constraints as an argument, else it is called with the set of violated covering constraints. This procedure returns a maximal compatible, M_i, satisfying at least one of the most restrictive constraint (constraint satisfied by the smallest number of MCs) in the passed set. Procedure ***Most_restrictive()*** evaluates the most restrictive constraints. These MCs are stored in the set *Cand*. For each MC in this set, ***Compute_C()*** calculates a) the number of violated constraints satisfied by selecting M_i (PC, positive contribution), and b) the number of them that are violated as a consequence of adding M_i to CC (NC, negative contribution). Finally the MC, M_i, which maximizes C, a weighted sum of NC and PC, is returned so ending the call to procedure **Select_CC()**. we remark how if both closure and covering constraints are violated, we attempt to satisfy closure requirements firstly. These actions are repeated until CC is a closed cover for table T. Then, the reduced state table, T', is derived by **Get_Table()**. An important point in the heuristic proposed is the selection of weights in the definition of C. From our experience we conclude that an effective guideline is $C = PC - 2NC$.

Note that unlike the algorithm proposed in [10] we do not need to generate a minimum set of MCs satisfying a given subset of the constraints, this preventing us to handle a NP-complete problem.

We will use FSM2 in *Table 3* as an example to clarify how the algorithm works. The set of maximal compatibles (CCSS) for this machine is shown in *Figure 1*.

MCs for Table 2:

$M_1 = (s_4,s_5,s_7,s_8,s_9)$	$M_6 = (s_2,s_3,s_6,s_9)$
$M_2 = (s_2,s_5,s_7,s_8,s_9)$	$M_7 = (s_1,s_2,s_5,s_8)$
$M_3 = (s_2,s_3,s_7,s_8,s_9)$	$M_8 = (s_1,s_2,s_3,s_8)$
$M_4 = (s_4,s_5,s_6,s_9)$	$M_9 = (s_1,s_2,s_5,s_6)$
$M_5 = (s_2,s_5,s_6,s_9)$	$M_{10} = (s_1,s_2,s_3,s_6)$

Figure 1

We associate a 0 -1 integer variable c_i with each maximal compatible M_i so that the corresponding integer program is formulated. $c_i = 1$ if M_i is selected as a member of a closed cover set and $c_i = 0$ if M_i is not selected. These covering and closure constraints are shown in *Figure 2*.

$$-1 + c_7 + c_8 + c_9 + c_{10} \geq 0 \quad (1)$$
$$-1 + c_2 + c_3 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} \geq 0 \quad (2)$$
$$-1 + c_3 + c_6 + c_8 + c_{10} \geq 0 \quad (3)$$
$$-1 + c_1 + c_4 \geq 0 \quad (4)$$
$$-1 + c_1 + c_2 + c_4 + c_5 + c_7 + c_9 \geq 0 \quad (5)$$
$$-1 + c_4 + c_5 + c_6 + c_9 + c_{10} \geq 0 \quad (6)$$
$$-1 + c_1 + c_2 + c_3 \geq 0 \quad (7)$$
$$-1 + c_1 + c_2 + c_3 + c_7 + c_8 \geq 0 \quad (8)$$
$$-1 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 \geq 0 \quad (9)$$
$$-c_1 + c_2 + c_5 + c_7 + c_9 \geq 0 \quad (10)$$
$$-c_1 + c_8 + c_{10} \geq 0 \quad (11)$$
$$-c_1 + c_9 \geq 0 \quad (12)$$
$$-c_2 + c_4 + c_5 + c_9 \geq 0 \quad (13)$$
$$-c_2 + c_3 + c_6 \geq 0 \quad (14)$$
$$-c_2 + c_9 + c_{10} \geq 0 \quad (15)$$
$$-c_3 + c_4 + c_5 + c_9 \geq 0 \quad (16)$$
$$-c_3 + c_7 + c_8 \geq 0 \quad (17)$$
$$-c_4 + c_7 + c_9 \geq 0 \quad (18)$$
$$-c_4 + c_8 + c_{10} \geq 0 \quad (19)$$
$$-c_4 + c_5 + c_6 + c_9 + c_{10} \geq 0 \quad (20)$$
$$-c_5 + c_9 \geq 0 \quad (21)$$
$$-c_5 + c_3 + c_6 \geq 0 \quad (22)$$
$$-c_6 + c_9 \geq 0 \quad (23)$$
$$-c_6 + c_2 + c_3 \geq 0 \quad (24)$$
$$-c_7 + c_5 + c_9 \geq 0 \quad (25)$$
$$-c_7 + c_1 + c_4 \geq 0 \quad (26)$$
$$-c_7 + c_{10} \geq 0 \quad (27)$$
$$-c_8 + c_5 + c_9 \geq 0 \quad (28)$$
$$-c_8 + c_1 + c_4 \geq 0 \quad (29)$$
$$-c_9 + c_1 + c_4 \geq 0 \quad (30)$$
$$-c_9 + c_6 + c_{10} \geq 0 \quad (31)$$
$$-c_{10} + c_1 + c_4 \geq 0 \quad (32)$$
$$-c_{10} + c_2 + c_3 \geq 0 \quad (33)$$
$$-c_{10} + c_3 + c_8 \geq 0 \quad (34)$$

Figure 2

Initially, $CC = \emptyset$ and the procedure **Select_CC()** is called with the set of covering constraints ((1) – (9)) as argument. Inequality number (4) is the most restrictive violated one, as it is satisfied only by the selection of M_1 or M_4 (these MCs are the only ones that cover state

s_4). It is clear that M_1 and/or M_4 will be members of any closed cover set, including that of minimum cardinality. Then:

$$Cand = \{M_1, M_4\}$$

and C is computed for both MCs:

- for M_1

$PC(M_1)$ = 5 because M_1 = (s_4, s_5, s_7, s_8, s_9) covers five uncovered states of the original table. This is, the selection of this MC satisfies five violated inequalities. These are (4), (5), (7), (8) and (9).

$NC(M_1)$ = 3 because the consequence of selecting M_1 is the violation of inequalities (10), (11), (12) which express the closure requirements of M_1. This is, if M_1 is selected as a member of a closed cover, MCs containing compatibles (s_1, s_2, s_3), (s_2, s_5) and (s_1, s_5, s_6) will have to be included too.

$$C(M_1) = PC(M_1) - 2NC(M_1) = -1$$

- for M_4

$$PC(M_4) = 4$$
$$NC(M_4) = 3$$
$$C(M_4) = PC(M_4) - 2NC(M_4) = -2$$

M_1 is selected ($CC = \{M_1\}$) because it maximizes the parameter C. Now there are some closure constraints violated (inequalities (10), (11), (12)) and the algorithm tries to satisfy them firstly, so procedure **Select_CC()** is called for these constraints. Among this set inequality (12) is the most restrictive one and:

$$Cand = \{M_9\}$$

In this case as there is only one candidate MC we do not need to compute C. M_9 becomes a member of CC.

Now, the set of violated closure constraints is {(11), (31)} and

$$Cand = \{M_6, M_8, M_{10}\}$$
$$C(M_6) = 0$$
$$C(M_8) = 2$$
$$C(M_{10}) = -1$$

M_8 is selected. The only inequality not satisfied is (31) and the algorithm evaluates C for each of the MCs whose selection satisfies that constraint, i. e.:

$$Cand = \{M_6, M_{10}\}$$
$$C(M_6) = -1$$
$$C(M_{10}) = -1$$

Arbitrary M_6 is selected. As a consequence of adding this MC to CC, constraint (24) is violated :

$$Cand = \{M_2, M_3\}$$
$$C(M_2) = 1$$
$$C(M_3) = 1$$

M_2 is added to CC ($CC = \{M_1 M_9, M_8, M_6, M_2\}$) which now is a closed cover of the original table describing the FSM

PRACTICAL RESULTS AND COMPARISONS

We have developed REDUCES, a computer program for state reduction of Finite State Machines. This program has been written in C and consists of around 1000 lines of code. We have oriented this program to get solutions based on MCs. REDUCES has been applied to many problems; solutions given by this new algorithm are better than those supplied by other programs. Because of the lack of space, we will only show some examples herein. As we have no knowledge of any standard benchmark for state reduction we have opted for generating random machines. *Table 4* depicts the results obtained by REDUCES, the first solution given by an implementation of the algorithm in [14], the one obtained by an implementation of the algorithm in [10], and the minimum, for 30 randomly generated machines.

Concerning CPU time, REDUCES always gave either the same or better results . However, numeric figures in *Table 4* may be misleading since the three methods are very fast for these examples. What must be emphasized is that a reduction in the number of states allow us to get significant savings in time during the state assignment phase, these savings ranging in the case of the machines in *Table 4* from a few to many minutes when a SUN 3/260 and PRASES are used. Concerning the number of states, it should be clear the superior performance of the new algorithm, which always gave the best solution with the exception of FSM15 for which G1 found the absolute optimum. Nevertheless, REDUCES found a solution with only one more state.

Furthermore, in order to understand the improvement given by REDUCES, it is worth comparing it with standard algorithms. Basically, the general method [14] for 0-1 integer LP problems can be viewed as a branch and bound algorithm. This is, a variation on backtrack that achieves branch pruning (preclusion) of the search tree, on the assumption that each solution has a cost associated with it and that the one of least cost is to be found. The efficiency of the preclusion techniques depends on the selective strategy being used to choose the next variable on which branching is performed. We have used the heuristic strategy described in the previous section to implement a new branch and bound algorithm that allows to preclude more nodes of the search tree. So we can obtain minimum solutions faster than with [14]. In *Table 5* we include a time comparison between the algorithm in [14] and the new one just proposed for other 30 machines. We also report the number of searched nodes in each case. Again, the superior performance of the new method should be evident.

CONCLUSIONS

A new state reduction program REDUCES has been described. It has proven to be more efficient than previous heuristic algorithms in both quality of the solutions and required time. Moreover, the strategy described above when used as the selective strategy within a general linear programming method, has shown to achieve high pruning of the search tree for integer LP problems associated with the state reduction of incompletely specified FSMs. Comparisons between the results given by our method and others previously reported have shown a clear superiority of the new algorithm.

FSM	PARAMETERS			G1 [14]		BENNETS [10]		REDUCES		MINI-MUM
	ns	ni	#MC	ns_1	t	ns_2	t	ns_3	t	
FSM1	43	3	25	17	0.4	12	0.4	8	0.2	8
FSM2	43	3	27	9	0.3	8	0.5	8	0.3	6
FSM3	43	3	27	15	0.4	17	0.6	15	0.4	15
FSM4	43	3	27	9	0.3	10	0.5	8	0.3	8
FSM5	43	3	27	7	0.3	8	0.3	7	0.3	7
FSM6	43	13	25	19	1.0	20	1.8	19	1.0	18
FSM7	43	13	25	21	1.1	20	1.6	16	0.8	11
FSM8	43	13	25	24	1.3	20	1.4	17	0.9	12
FSM9	43	13	27	15	1.4	18	1.9	12	1.1	11
FSM10	43	13	27	22	1.4	19	1.6	14	0.9	11
FSM11	43	13	27	10	0.9	20	2.1	11	0.9	10
FSM12	43	13	27	12	1.2	20	1.7	10	1.0	10
FSM13	43	13	27	27	1.5	27	2.1	27	1.5	27
FSM14	70	3	45	15	1.6	12	1.2	7	0.8	7
FSM15	70	3	45	24	1.4	28	1.5	23	1.4	22
FSM16	70	3	45	9	0.8	11	1.0	8	0.8	8
FSM17	70	3	45	11	0.8	13	0.9	10	0.8	9
FSM18	70	3	47	8	0.8	11	0.9	8	0.8	7
FSM19	70	3	47	13	0.9	17	1.0	11	0.8	10
FSM20	70	3	47	13	1.3	13	1.1	11	1.1	10
FSM21	70	3	49	19	1.8	13	1.1	9	0.9	9
FSM22	70	3	49	10	0.9	14	1.1	10	0.9	9
FSM23	70	3	49	34	2.2	18	1.2	14	0.9	12
FSM24	70	3	65	16	2.2	12	2.0	11	1.5	8
FSM25	70	3	65	15	2.5	15	2.0	10	1.7	8
FSM26	70	3	65	9	2.0	11	1.9	9	2.0	8
FSM27	70	3	65	9	1.5	10	1.8	8	1.4	8
FSM28	70	3	65	16	2.7	11	1.8	9	1.5	7
FSM29	70	3	65	16	1.7	12	1.7	10	1.7	9
FSM30	70	3	65	11	2.0	11	1.7	8	1.5	8

Table 4

FSM	PARAMETERS			GILLET [14]		REDUCES	
	ns	ni	#MC	searched nodes	t (secs.)	searched nodes	t (secs.)
FSM1	43	3	25	2915	78.0	183	4.9
FSM2	43	3	25	2997	103.1	279	9.6
FSM3	43	3	25	3593	98.1	637	17.4
FSM4	43	3	25	7191	220.6	339	10.4
FSM7	43	3	25	7323	282.2	685	26.4
FSM6	43	3	27	7339	231.6	957	30.2
FSM7	43	3	27	4439	177.5	145	5.8
FSM8	43	3	27	3739	130.7	855	29.9
FSM9	46	3	25	3659	115.2	305	0.6
FSM10	46	3	27	5559	204.0	365	13.4
FSM11	46	3	27	9807	313.0	1385	44.2
FSM12	46	3	27	387	19.1	297	14.7
FSM13	46	3	29	9393	376.2	719	28.8
FSM18	46	3	29	6093	228.1	1245	46.6
FSM15	46	3	31	5289	265.3	315	15.8
FSM26	49	3	25	8557	246.7	843	24.3
FSM17	49	3	25	5351	171.4	409	13.1
FSM18	43	13	25	4001	393.9	655	64.5
FSM19	43	13	25	3853	367.5	587	56.0
FSM20	43	13	25	7829	806.7	723	74.5
FSM21	43	13	25	19227	1906.1	815	80.8
FSM22	43	13	25	18925	1818.1	3233	310.6
FSM23	43	13	27	14895	1681	1909	215.5
FSM24	43	13	27	23057	2424	1625	170.9
FSM25	43	13	27	28671	3063	3065	327.5
FSM26	43	13	27	4911	545.5	2211	245,6
FSM27	70	3	45	1943	239.5	893	110.1
FSM28	70	3	47	6831	594.6	139	12.1
FSM29	70	3	49	174133	17510	179	18
FSM30	70	3	49	228557	21385	1741	162.9

Table 5

REFERENCES

[1] M. A. Perkowski, J. Liu: "Generation and Optimization of Finite State Machines from Parallel Program Graphs," *DIADES Research Group Report 10/89*, Dept. EE PSU.

[2] R. Rudell, A. Sangivanni-Vincentelli and G. De Micheli: "A Finite-State Machine Sunthesis System,"*Proc. 1985 Int. Symp. on Circ. and Syst., Kyoto, Japan*, pp. 647-650.

[3] F. J. Hill and G. R. Peterson: "Switching Theory and Logical Design," John Wiley and Sons, 1981.

[4] Z. Kohavi: "Switching and Finite Automata Theory". New York, McGraw Hill, 1970.

[5] S. House: "A New Rule for Reducing CC Tables," *IEEE Trans. on Computers (Short Notes)*, Nov. 1970.

[6] J.L. Huertas and J.M. Quintana: "A New Method for the Efficient State-Assignment of PLA-Based Sequential Machines," *Proc. 1988 Int. Conf. on CAD*, pp. 156-159.

[7] S. Ginsburg: "On the Reduction of Superfluous States in a Sequential Machine," The National Cash Register Company, Hawthorne, California, 1959.

[8] M. C. Paul and S. H. Unger: "Minimizing the Number of States in Incompletely Specified Sequential Switching Funtions," *IRE Trans. on Computers*, vol EC-8, pp. 356-367, Sept. 1959.

[9] A. Grasselli and F. Luccio: "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IEEE Trans. Electron. Comp.*, Vol. EC-14, pp. 350-359, June 1965.

[10] R. G. Bennetts, J. L. Washington and D. W. Lewin: "A Computer Algorithm for State Table Reductions," *IERE Radio and Electron. Eng.*, Vol. 42, pp. 513-520, Nov. 1972.

[11] N. N. Biswas: "State Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. on. Computers*, Vol. C-23, pp. 80-84, Jan. 1974.

[12] S. C. De Sarkar, A.K. Basu and A.K. Choudhury: "Simplification of Incompletely Specified Flow Tables with the Help of Prime Closed Sets," *IEEE Trans. on Computers (Short Notes)*, Vol. C-18, pp 953-956, Oct. 1969.

[13] S. H. Unger: "Asynchronous Sequential Switching Circuits," Wiley-Interscience, New York, 1969.

[14] B. E. Gillet: "Introduction to Operations Research," McGraw Hill, 1976.

[15] R. W. House, L. D. Nelson and T. Rado: "Computer Studies of a Certain Class of Linear Integers Problems," in Recent Advances in Optimization Techniques, A. Lavi and T. P. Vogl, Eds. New York: Wiley 1966.