COMPUTER-AIDED SYNTHESIS OF PLA-BASED FINITE STATE MACHINES

Giovanni De Micheli
Alberto Sangiovanni-Vincentelli
Department of EECS-University of California at Berkeley
Tiziano Villa
CSELT Torino (Italy) and Department of EECS-University of California at Berkeley

Abstract: We address the optimal logic design of PLA-based Finite State Machines (FSM). Techniques related to heuristic combinational logic minimization are used to determine optimal coding of the FSM internal states. We show that if appropriate Hamming-distance requirements among state codes are preserved, reduction of the combinational logic is guaranteed. A state encoding technique satisfying these requirements and based on graph embedding in squashed hypercubes is presented. Experimental results are reported.

1. INTRODUCTION

Sequential circuits play a major role in the control part of digital systems. We address the automated synthesis of sequential logic functions in a structured VLSI design methodology. We consider sequential logic functions implemented by synchronous deterministic Finite State Machines (FSM) consisting of two distinct components: a combinational circuit implemented by a Programmable Logic Array (PLA) and a memory implemented by Delay-type registers.

In particular we consider here the problem of assigning binary codes to the internal states of a Finite State Machine. The literature is rich of papers dealing with the state-assignment problem. Here we refer to the major approaches only. Armstrong [1] introduced a set of criteria for encoding states, aiming at the minimization of the number of gates used to implement the FSM and formulated the encoding problem as a graph embedding problem. Hartmanis [2], Stearns [3] and Karp [4] developed algebraic methods based on partition theory and on a reduced dependence criterion. Dolotta and McCluskey [5] suggested a "column-based" procedure to code states.
Note that despite these efforts, to the best of our knowledge no tool for designing FSM is in use today for a time-effective state encoding of industrial digital controllers.

Armstrong's approach can in principle handle rather large machines, but it has three serious drawbacks. The first is related to the fact that the criteria suggested by Armstrong do not take into account the techniques of fast heuristic logic minimizers such as MINI [6], PRESTO [7], or ESPRESSO-II [8] in use today (Armstrongs paper appeared before the work on heuristic minimizers started). The second is that the state-assignment problem is transformed into a particular graph-embedding problem, which represents only partially the state coding problem, as shown in section 4. The third is that the graph embedding algorithm suggested by Armstrong was ineffective.

Our approach is based, as Armstrong's, on the use of distance relations among the codes of the internal states. In section 3 we show how the combinational logic can be reduced by requiring state codes to satisfy appropriate distances. Distance requirements are determined by predicting the effects of heuristic minimization of the combinational logic related to a symbolic description of the FSM, and are represented by a graph. In particular it is shown that a convenient reduction of the combinational logic is obtained if the distance between some state codes is large enough and appropriate states have adjacent codes.

In section 4 we consider the problem of assigning codes which satisfy the distance relations. Adjacent code assignment can be seen as an embedding of an adjacency graph into a boolean hypercube. Armstrong [1] and Saucier [9] represented the state assignment problem as a subgraph isomorphism problem, where a one-to-one relation (coding) is sought between the set of the states (vertices of the adjacency graph) and a subset of the boolean hypercube vertices (codes).
Note that even questioning the existence of a subgraph isomorphism is a hard problem: in particular it was shown to belong to the class of NP-complete problems [10]. Since such an isomorphism may not exists, Armstrong and Saucier relaxed some adjacency requirements and proposed heuristic techniques to embed a subgraph of the adjacency graph into the boolean hypercube. Note that a distance-preserving embedding is not even guaranteed by augmenting the dimensions of the hypercube, i.e. increasing the length of the state codes.

Our approach exploits the use of *don't care* conditions in state codes. In particular every state is coded by associating each vertex of the adjacency graph to a subcube of the boolean hypercube. This is equivalent to embed the adjacency graph into a **squashed hypercube**, i.e. a hypercube having appropriate faces squeezed into vertices [11]. Note that most of the state assignment techniques presented in the literature obtained a state coding using the minimum number of bits, because it was important to minimize the number of memory elements due to their cost. On the other hand, the area taken by the PLA is the major concern in a VLSI circuit implementation of a Finite State Machine. Minimal area PLA implementations of the FSM combinational component can be obtained by using non-minimal-length state codings i.e. fewer product-terms are often required to implement a logic function at the expense of an increased number of input/output columns. Therefore we allow non-minimal-length state codings when leading to minimal area PLAs. In this case, state coding corresponds to an embedding into a squashed hypercube of variable dimension. However bounds on code-length can be enforced when required by a particular implementation.

2. FINITE STATE MACHINE REPRESENTATION

Different functional FSM representations are commonly used. Most state-assignment techniques reported in the literature are based on a state-table representation, though it can be cumbersome for large uncompletely-specified machines. For this reason designers describe the machine functionality by means of flow-charts or Hardware Description Languages (HDL). Unfortunately these descriptions are not well-suited to support machine optimization techniques. For these reasons we represent the FSM functionality by means of a **symbolic cover**. The concept of symbolic cover is a generalization of the logic cover representation of combinational-logic functions [6]. Symbolic covers can be obtained from flow-charts, HDL or state tables in a straightforward way.

A symbolic cover is a set of primitive elements called **symbolic implicants**. A symbolic implicant (denoted here by a capital letter e.g. $A = \{i_A, s_A, s'_A, o_A\}$) is a set of two input and two output character strings. The two input strings represent a binary-valued representation of a primary input (i_A) and a symbolic representation of a present state (s_A). The two output strings represent the corresponding symbolic representation of the next-state (s'_A) and a binary-valued representation of the primary outputs (o_A). Note that we consider in this paper the problem of assigning binary codes to the FSM internal states only. Therefore we assume that i_A and o_A are already coded into binary strings. However i_A and o_A might describe symbolic inputs and outputs in a more general framework, where primary input and output coding is also considered. We represent binary valued variables by the symbols "1", "0" and "*", where "*" represents a *don't care* condition. States are represented symbolically by a character mnemonic string.

Example: Consider the traffic-light controller presented in [12]. The following is a symbolic implicant:

```
11*,HG,HY,10010
```

showing the a "1" in the first two primary-input lines maps state "HG" into state "HY" and asserts output 10010. The symbolic cover is the collection of the symbolic implicants representing the state transitions:

```
0**,HG,HG,00010
*0*,HG,HG,00010
11*,HG,HY,10010
**0,HY,HY,00110
**1,HY,FG,10110
10*,FG,FG,01000
0**,FG,FY,11000
*1*,FG,FY,11000
**0,FY,FY,01001
**1,FY,HG,11001
```

■

Note that a symbolic cover is a logic cover of a multiple-valued logic function [6] [13], where each state takes a different logic level and is represented by a character string. A symbolic implicant having n (m) primary input (output) bits can be seen as a (n+1)-input, (m+1)-output multiple-valued logic implicant. Definitions and properties of multiple-valued-logic covers carry over to symbolic covers as well [13].

The motivation for using a symbolic cover relies on the following points.

i) properties and operations on symbolic covers can be exploited and related to heuristic minimization algorithms for binary-valued logic functions [6],[7],[8].

ii) any logic cover of the combinational component of a FSM obtained by assigning disjoint codes to each state can be seen as a symbolic cover. Hence the technique we present can be interfaced to several FSM automated design tools, in order to implement the machine aiming specifically to a PLA-based implementation in a minimal area.

The state assignment problem consists of determining a coding map $c(\cdot)$ which transforms the state symbols into strings of binary digits. This is equivalent to transforming the symbolic cover into a binary-valued logic cover of the combinational component. Note that in general *don't care* coordinates are used in state codes, and therefore every state is assigned to a subcube of the boolean hypercube. However a coding map is **implementable** only if the states are assigned to non-overlapping regions of the boolean hypercube.

State coding affects substantially the complexity of the combinational component of a FSM, because minimal binary-valued logic covers [6] corresponding to different coding maps have different cardinalities. We consider first coding maps with no code-length bounds. The **unconstrained optimum state assignment** problem can be stated as follows:

Find an implementable coding map $c(\cdot)$ that minimizes the cardinality of the minimal logic cover of the FSM combinational component.

This is a formidable task, because it involves the search for all the minimal covers related to all possible codings! We therefore concentrate on a simpler problem and we relate optimal state coding to heuristic minimization of the logic cover [6] [8]. In particular we look for an implementable coding map which leads to a minimal logic cover having significantly fewer implicants than the original symbolic cover. Similarly a **constrained state assignment** problem can be defined by restricting the search to codings of bounded length.

3. CODE DISTANCES AND COMBINATIONAL LOGIC MINIMIZATION

We investigate in this section the relations between state assignment and the complexity of the related implementation of the combinational part of a FSM. In particular a set of rules can be obtained to determine constraints on state code distances, so that either the cardinality or the number of literals of the logic cover (or both) can be reduced. However we report here on the two major rules only.

We call **cube** any string of characters from the set $\{0,1,*\}$. We refer the reader to [6] for definitions of cover ($\supseteq$), union ($\cup$), sharp (-) and intersection ($\cap$) between cubes. The distance $D(a,\ b)$ between two cubes a and b of equal length is the number of positions in which they differ. The Hamming distance $H(a,\ b)$ between two cubes a and b of equal length is the number of positions in which they differ and both entries are cares. Note that if s1 and s2 are two different state symbols, an implementable coding is such that $H(c(s1), c(s2)) > 0$. We define two state codes to be **adjacent**, if their Hamming distance is one, because they are adjacent vertices of a squashed cube representation.

The basic strategy for obtaining a set of relations among state codes is the following. All pairs of symbolic implicants (A, B) are examined and code distance requirements are enforced according to the following rules. When Rule 1 applies, two symbolic implicants can be coded and merged into one binary-valued logical implicant and the cover cardinality be reduced. Therefore Rule 1 is considered a "strong rule" and it is highly desirable that the related code distance requirements are satisfied. Rule 2 allows to reduce the number of literals and is considered a "weak rule" compared to Rule 1, because a reduction in size of the PLA is considered more desirable than a reduction of its complexity.

Let $A = \{i_A, s_A, s'_A, o_A\}$ be a symbolic implicant of the machine cover. We define $S(A)$ the set of states which are mapped by any input representation $i \subseteq i_A$ either into a next-state different from s'_A or into an output representation not covered by o_A or both.

Rule 1: Let $A = \{i_A, s_A, s'_A, o_A\}$, $B = \{i_B, s_B, s'_B, o_B\}$ be two symbolic implicants such that: $i_A \supseteq i_B$ and $o_A \supseteq o_B$.
Then:
$$c(s'_A) \supseteq c(s'_B).$$
and:
$$H(c(s_A) \cup c(s_B), c(s_Q)) > 0 \quad \forall s_Q \in S(A).$$

Rationale: A and B can be coded and merged into only one logic implicant, namely:
$$\{i_A\ ,\ c(s_A) \cup c(s_B)\ ,\ c(s'_A)\ ,\ o_A\}$$ ■

Rule 1 requires two different conditions on state codes: i) a covering relation between cubes $c(s'_A)$ and $c(s'_B)$ considered as output parts of binary-valued implicants; ii) a distance relation which keeps state codes $c(s_A)$ and $c(s_B)$ far from the codes of the states in $S(A)$.

Remark: If $s'_A = s'_B$ the covering requirement is automatically satisfied. Moreover if only completely specified codes are used (i.e. no *don't care* conditions are used in state codes), then $D(c(s_A), c(s_B)) = 1$ implies that $H(c(s_A) \cup c(s_B), c(s_Q)) > 0\ \forall s_Q \neq s_A$ and $s_Q \neq s_B$. This requirement is equivalent to the column adjacency rule stated by Armstrong in [1], when $i_A = i_B$ and $o_A = o_B$. Note that Rule 1 is far more general than Armstrong's rule. ■

Let $A = \{i_A, s_A, s'_A, o_A\}$ and $B = \{i_B, s_B, s'_B, o_B\}$ be two symbolic implicants such that: $s_A = s_B$ and $o_A = o_B$. We define $I(AB)$ the set *of input representations* $\{\, i \subseteq i_A \cup i_B \,\}$ *which map state s_A either into a* next-state different from s'_A or s'_B or into an output representation not covered by o_A or both.

Rule 2: Let $A = \{i_A, s_A, s'_A, o_A\}$ and $B = \{i_B, s_B, s'_B, o_B\}$ be two symbolic implicants such that: $s_A = s_B$ and $o_A = o_B$. If $I(AB) = \phi$, then:
$$H(c(s'_A), c(s'_B)) = 1.$$

Rationale: the corresponding logical implicants can be reshaped [6] as:
$$\{i_A \cup i_B,\ c(s_A),\ \tilde{c}(s'_A),\ o_A\}$$
$$\{i_A,\ c(s_B),\ c(s'_B) - c(s'_A),\ \vartheta\}$$
where ϑ is a string of "0"s and where without loss of generality $c(s'_A)$ and $c(s'_B)$ are obtained by assigning cares to the *don't care* entries of the next-state codes so that $D(c(s'_A), \tilde{c}(s'_B)) = 1$ and the "1" count in $\tilde{c}(s'_B)$ is larger than in $c(s'_A)$. Note that the second logical implicant obtained by Rule 2 has always only one *care* in the output part. Hence it may be covered by some other implicants of the logical cover. Therefore when Rule 2 applies the number of literals and possibly the logical cover cardinality are reduced. ■

Rule 2 requires an adjacency relation between cubes $c(s'_A)$ and $c(s'_B)$.

Remark: If $D(i_A,\ i_B) = 1$, then $I(AB) = \phi$. Therefore, if we restrict our attention to completely specified codes only, $D(c(s'_A), c(s'_B)) = 1$ implies that $H(c(s'_A), c(s'_B)) = 1$. This condition is equivalent to the row adjacency rule presented by Armstrong in [1]. ■

More complex rules can be derived by considering other relations between symbolic implicants. In particular, Rule 2 can be generalized to the case in which $o_A \supseteq o_B$ and Rule 1 be modified to the case in which $H(o_A, o_B) = 1$.

4. STATE ENCODING STRATEGIES

The rules stated in Section 3 give rise to relations among state codes which can be grouped as follows:

1) code adjacency ($H(c(s_A), c(s_B)) = 1$);

2) code covering , i.e. requiring a next-state code to cover another next-state code ($c(s'_A) \supseteq c(s'_B)$);

3) code distance, i.e. requiring the code of one state, say s_Q , to be far enough from the union of the codes of a pair of states s_A and s_B ($H(c(s_A) \cup c(s_B), c(s_Q)) > 0$).

Relations 1) and 2) are represented by a mixed weighted graph, $G(V,E,W(E))$, where the set of nodes V is in one-to-one correspondence with the set of states, and E consists of a set of directed and undirected edges. The undirected edges are related to the adjacency relations, i.e. $\{v_a,v_b\} \in E$ if $H(c(s_A), c(s_B)) = 1$; and the directed edges are related to covering relations, i.e. $(v_a,v_b) \in E$, if $H(c(s_A), c(s_B)) = 1$ and $c(s_A) \supseteq c(s_B)$. Weights are defined according to the number of times the same distance requirement occurs and to the related rule. Code distance requirements are represented by a list structure. In particular $H(c(s_A) \cup c(s_B), c(s_Q)) > 0$ is represented by s_Q pointing to the pair s_A and s_B in the list.

The problem of finding a state coding which satisfies the rules given in Section 3 can now be seen as a graph embedding problem. Let N be the dimension of a boolean hypercube B, representing the possible codes. Let $P(B)$ be the set of all subcubes contained in B. We have to determine the dimension N and an injective function $c: V \rightarrow P(B)$ such that the relations induced by the rules presented in Section 3 are satisfied. The adjacency relations are satisfied if:

$$d_G(v_i,v_j) \geq H(c(v_i),c(v_j)) \quad \forall v_i,v_j \quad v_i \neq v_j \in V$$

where $d_G(v_i,v_j)$ is the length of the shortest path in the graph between v_i and v_j and $H(c(v_i),c(v_j))$ denotes the Hamming distance of the subcubes $c(v_i)$ and $c(v_j)$. Geometrically, we would like to determine, ***if possible***, an isomorphism between the graph and a squashed hypercube, i.e. an hypercube B in which some elements of $P(B)$ collapse into a vertex. It can be proven that there always exists an integer $\hat{N}$ such that an injective map $c: B \rightarrow P(B)$ satisfying the above relations can be found. However, it is important not only to reduce the number of product terms of the FSM combinational component, but also to keep N as small as possible because N is proportional to the number of columns required by a PLA implementation.

Three optimization strategies can be followed:

1) Set N to a fixed value and find $c(\cdot)$ such that the number of code constraints violated by the encoding is minimized;

2) Find the smallest $\hat{N}$ such that all the rules are satisfied;

3) Trade-off N and the number of constraints violated.

Strategy 1 is close to the one followed by Armstrong where $N=\lceil \log_2 |V| \rceil$, i.e. the minimum number of bits needed to encode the states. Strategy 3 is the most desirable but obviously the most difficult to implement. We decided to implement strategy 2 as an intermediate step towards strategy 3. A first theoretical question to ask, when implementing strategy 2, is whether a bound on N can be found.

If we require that:

$$d_G(v_i,v_j)=H(c(v_i),c(v_j)) \quad \forall v_i,v_j \quad v_i \neq v_j \in V$$

we have an isometric embedding of a graph into a squashed hypercube. Graham showed in [11] that any graph $G(V,E)$ can be embedded into an hypercube of dimension $N(G)=(|V|-1)\text{diam}(G)$ where diam(G) is the diameter of G, and conjectured that the bound can be lowered to $N(G)=|V|-1$. The conjecture can be proven true for graphs belonging to some special classes ,e.g. complete graphs. The graph embedding problem arising from our formulation is a distance-bounded graph embedding. It can be reduced to an isometric embedding into a squashed hypercube by appending appropriate edges to G. Therefore there always exists a coding map $c(\cdot)$ satisfying the given requirements having $N(G)=|V|-1$.

We present in Fig. 1 the flow-chart of a heuristic algorithm for distance-bounded graph embedding, which reminds of the procedure presented in [14] for isometric embedding. The algorithm tries to minimize N and is constructs a coding using $N \leq |V|-1$ bits. Note that this is a worst-case upper bound and that the computed codes are much shorter than $|V|-1$ in many practical cases.

The algorithm applies to connected graphs. If $G(V,E,W(E))$ is disconnected, its connected components are determined first and the different groups of codes are packed together at the end. We deal here with a connected graph for the sake of simplicity.

The algorithm visits each node of the graph v_k, $k=1,\ldots,|V|$ and at the k-th step constructs a partial encoding of length $N(K)$ for v_k. It appends one bit to the codes of the nodes v_i, $i=1,\ldots,k$ only if the code length must be increased, as shown in Fig. 1. A degree of freedom of our procedure is the order of the selected nodes. We choose as first node the one which corresponds to the state with maximum number of occurrences as next-state in the symbolic cover. We map it into the origin of the coordinates of the hypercube, to maximize the occurrence of "0"s in the output part of the coded implicants. The node selected at the k-th step, v_k is adjacent to a coded vertex (i.e. adjacent to $v_j; j<k$) and has the maximum number of uncoded adjacent nodes. The rationale is that such a node has more constraints to satisfy and so it deserves higher priority in the space occupation on the hypercube.

At step k node v_k is coded as follows. Assume we have assigned partial codings of code length $N(K-1)$ to v_i, $i=1,\ldots,k-1$ so that:

$$H(c(v_i),c(v_j))=1$$

for all adjacent coded pairs v_i and v_j. Then we search for an implementable coding $c(v_k)$ of the same length with the property that:

$$H(c(v_k),c(v_j))=1$$

for all adjacent coded pairs v_i and v_j, under the constraint:

$$H(c(v_k),c(v_r)) \cup c(v_s)))>0$$

for all node pairs v_r,v_s in the list pointed by v_k and coded before step k. An exhaustive search of a feasible code would require $3^{N(K-1)}$ trials. Therefore we test only a subset of the possible trials, which are called "slight modifications" of the coding of the vertices adjacent to v_k. A slight modification is obtained by complementing one care bit ("1" or "0") of the code of a vertex adjacent to v_k.There are at most $|V|^2$ such trials.

It is possible that no implementable coding for v_k can be obtained by slight modifications. In this case the algorithm constructs the code of v_k by appending a "1" to the string of bits obtained from the logical union of the codes of all the adjacent vertices and by appending a "1" or "0" or "*" to the code of each vertex v_j coded before step k. In this way,we can always satisfy the distance requirements, but unfortunately at the expense of an increase in the code length. However, the algorithm will construct a valid encoding for G of length bounded by $1+\sum_{i=3}^{|V|} 1 = |V|-1$. Its computational complexity is $O(|V|^3)$ in the worst case.

Remark: Since a bound on the code length can be obtained by bounding the number of vertices in each connected component of G, we can partition the graph into components of bounded size by removing a subset of edges. Edge weights can be used to determine the optimal graph decomposition. ■

5. EXPERIMENTAL RESULTS AND CONCLUDING REMARKS

The algorithm has been implemented by an interactive computer program. The program reads the symbolic description of a FSM, generates the distance requirements and determines state codes. The program has been tested on a set of industrial Finite State Machines. Results are reported in Table 1 and show that the algorithm is effective in generating state codes leading to a FSM implementation with a reduced number of product-terms in the combinational component. Execution times are in the order of some seconds on a IBM 3081 computer.

A new approach based on multiple-valued logic minimization is being currently pursued in collaboration with Dr. Brayton of IBM. Preliminary experimental results show that this method can be considered a break-through in FSM synthesis.

6. ACKNOWLEDGMENTS

This research has been sponsored by IBM, NSF under subcontract # 392741C-1, DARPA under contract # 25976 and CSELT, Italy.

REFERENCES

[1] D.B.Armstrong " A Programmed Algorithm for Assigning Internal Codes to Sequential Machines " *IRE Trans. Elect. Comp.* vol EC-11 pp 466-472 , aug 1962.

[2] J. Hartmanis " On the State Assignment Problem for Sequential Machines 1" *IRE Trans. Elect. Comp.* vol EC-10 pp 157-165, jun 1961.

[3] R.E.Stearns and J. Hartmanis " On the State Assignment Problem for Sequential Machines 2" *IRE Trans. Elect. Comp.* vol EC-10 pp 593-603, dec 1961.

[4] R.Karp " Some Techniques for State Assignment for Synchronous Sequential Machines " *IEEE Trans. Elect. Comp.* vol EC-13 pp 507-518 , oct 1964.

[5] T.A.Dolotta and E.G McCluskey " The coding of internal states of sequential machines" *IEEE Trans. Elect. Comp.* vol EC-13 pp 549-562, oct 1964.

[6] S.J.Hong,R.G.Cain and D.L.Ostapko "MINI:a Heuristic Approach for Logic Minimization" *IBM Jour. of Res. and Devel.*, vol 18, pp 443-458, sep 1974.

[7] D.Brown "A State Machine synthesizer" *Proc. 18th Des. Aut. Conf.*, Nashville, jun 1981.

[8] R.Brayton,G.D.Hachtel,C.McMullen and A.L.Sangiovanni- Vincentelli "ESPRESSO-II A New PLA Logic Minimization Program " in preparation.

[9] G.Saucier "State Assignment of Asynchronous Sequential Machines Using Graph Techniques" *IEEE Trans. Comp.* vol C-21 pp 282-288, mar 1972.

[10] M.R.Garey and D.S.Johnson "*Computers and Intractability*" W.H.Freeman and Company, San Francisco 1978.

[11] R.L.Graham and H.O.Pollak "On Embedding Graphs in Squashed Cubes" *Graph Theory and Applications* Lecture notes in mathematics, no. 303, Springer Verlag 1972.

[12] C.Mead and L.Conway "*Introduction to VLSI Systems*" Addison Wesley, 1981.

[13] S.Y.H.Su andP.T.Cheung "Computer Minimization of Multi-Valued Switching Functions" *IEEE Trans. on Comp.* vol 21, 1972, pp 995-1003.

[14] R.L.Graham and H.O.Pollak "On The Addressing Problem for Loop Switching" *Bell Syst. Tech. Jour*, vol 50 No 8 , pp 2495-2519, oct 1971.

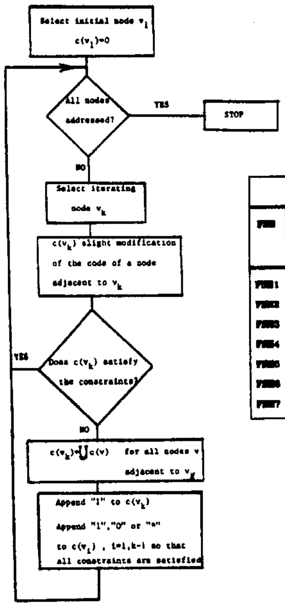
Select initial node v_1
$c(v_1)=0$
All nodes addressed?
YES
STOP
NO
Select iterating node v_k
$c(v_k)$ slight modification of the code of a node adjacent to v_k
YES
Does $c(v_k)$ satisfy the constraints?
NO
$c(v_k)=\bigcup c(v)$ for all nodes v adjacent to v_k
Append "1" to $c(v_k)$
Append "1","0" or "*" to $c(v_i)$, i=1,k-1 so that all constraints are satisfied

Fig. 1

PLA MINIMIZATION OF STATE-ENCODED FSMs						
FSM	Input bit-length	States	Output bit-length	Initial symbolic cover cardinality	Final logic cover cardinality	State-code bit-length
FSM1	4	8	1	20	8	3
FSM2	2	7	2	14	9	3
FSM3	4	8	3	32	25	5
FSM4	8	4	6	32	13	2
FSM5	8	7	6	38	25	6
FSM6	2	16	2	30	28	14
FSM7	8	6	5	46	25	3

TABLE 1