RC 10599 (#47457) 6/25/84
Computer Science 58 pages

OPTIMAL STATE ASSIGNMENT FOR FINITE STATE MACHINES

Giovanni De Micheli
Robert Brayton

IBM T.J.Watson Research Center
Yorktown Heights, NY 10598

Alberto Sangiovanni-Vincentelli

Dept. EECS, University of California at Berkeley
Berkeley, CA 94720

ABSTRACT

Computer-Aided synthesis of sequential functions of VLSI systems, such as microprocessor control units, must include design optimization procedures to yield area-effective circuits. We model sequential functions as deterministic synchronous Finite State Machines (FSMs), and we consider a regular and structured implementation by means of Programmable Logic Arrays (PLAs) and feedback registers. State assignment, i.e. binary encoding of the internal states of the finite state machine, affects substantially the silicon area taken by such an implementation. Several state assignment techniques have been proposed in the past. However, to the best of our knowledge, no Computer-Aided Design tool is in use today for an efficient encoding of control logic. We propose an algorithm for optimal state assignment. Optimal state assignment is based on an innovative strategy: logic minimization of the combinational component of the finite state machine is applied before state encoding. Logic minimization is performed on a symbolic (code independent) description of the finite state machine. The minimal symbolic representation defines the constraints of a new encoding problem, whose solutions are the state assignments that allow the implementation of the PLA with at most as many product-terms as the cardinality of the minimal symbolic representation. In this class, an optimal encoding is one of minimal length satisfying these constraints. A heuristic algorithm constructs a solution to the constrained encoding problem. The algorithm has been coded in a computer program, KISS, and tested on several examples of finite state machines. Experimental results have shown that the method is an effective tool for designing finite state machines.

1. INTRODUCTION

Very Large Scale Integrated (VLSI) circuits require a structured and hierarchical design methodology to cope with design complexity. Automated synthesis of regular modules implementing functional components allows a decrease in the design time while ensuring electrical correctness. Unfortunately, computer-aided synthesis techniques often yield integrated circuits requiring a large amount of silicon area. Therefore automated synthesis of VLSI modules must include design optimization procedures to be effective in industrial design.

Sequential circuits play a major role in the control part of digital systems. Digital computers are very complex examples of sequential systems and involve a combination of sequential functions.

A sequential function can be represented by several models [HOPC79]. The **deterministic finite state machine** or **deterministic automaton** representation is used in the sequel and is referred to as a finite state machine (FSM) for the sake of simplicity.

Hardware implementations of finite state machines consist of two major components: a combinational circuit and a memory. The memory stores a representation of the state of the machine at any given time and the combinational circuit generates the machine primary outputs as a function of the machine state and/or the machine primary inputs (Fig. 1.1).

A customized design of the combinational component can be achieved by interconnecting logic gates. For example, the control unit of the Z8000 micro-processor was implemented by "random logic" gates. Such a design may require a small silicon area, but it is highly dependent on the particular logic function. Moreover design time is longer in comparison to other structured implementations and engineering changes may require a complete redesign.

On the other hand, digital controllers are often implemented by finite state machines whose combinational component is a Read Only Memory (ROM). The controller operation can be altered by replacing or re-programming the ROM, giving a high degree of flexibility. The control units of micro-programmed digital computers are designed according to this methodology.

The implementation of sequential functions in VLSI system design has to satisfy two major requirements:

i) regular and structured design that can be supported by computer-aided tools;

ii) size and performance of the silicon implementation.

A PLA implementation of the FSM combinational component can satisfy both requirements. Since FSM memory components, as well as PLAs, can be designed by means of regular structures, the entire FSM implementation can be regular and structured. This allows the automation of FSM-based sequential-circuit design. Moreover several techniques, like logic minimization and topological compaction, allow the design of area-effective PLA implementations. Therefore PLA-based FSM design can be optimized with regard to silicon area requirement and subsequently to switching-time performance.

The memory component of a FSM consists of a set of latches that store the machine state representation. Several types of latches (Delay (D), Toggle (T), JK) can be used [HILL81]. Some functions can be implemented more efficiently with a particular type of latch: for example counters are usually implemented by means of T-latches and generic sequential functions by D-latches. For the sake of simplicity, we focus on FSMs whose memory elements are implemented by D-latches.

The operation of VLSI systems is often synchronized to a system clock. The main goal is to keep a "race-free" design even when the circuit size is large. For this reason the model of a sequential function implementation used here is a synchronous finite state machine, as shown in Fig. 1.2.

The computer-aided synthesis of a sequential circuit as a PLA-based finite state machines can be partitioned in the following tasks: **functional design, logic design, topological design** and **physical design** [DEMI83g].

Functional design consists in defining the system behavior by means of a functional description such as a flow-chart, a Hardware Description Language (HDL) or a state table. Flow chart descriptions, such as the Algorithmic State Machine (ASM) chart [CLAR75] or the Mnemonic Documented State (MDS) diagram [FLET80], allow an algorithmic description of the sequential system and enable the designer to visualize the entire machine functionality. Hardware description languages allow the description of a sequential function as a software program. State tables are tabular descriptions of the system functionality. Optimal functional design includes **state minimization** [HART66]. State minimization reduces the number of states of a sequential system, without modifying its functionality.

Logic design consists of mapping the functional description into a logic representation in terms of logic variables. In particular, a representation of the states in terms of Boolean variables is chosen. This step is referred to as **state assignment** [HART66]. The memory configuration used to store the machine state is selected at the same time. This allows us to express the excitation maps of the memory elements by Boolean equations. Note that the complexity of the combinational component of the machine depends heavily on the state assignment and on the excitation maps of the chosen memory elements. Therefore optimal logic design of sequential functions includes optimal state assignment and selection of memory elements. These steps are tightly related to the logic minimization of the combinational component [BRAY84],[HONG74].

Topological and **physical design** are related to the definition of the FSM layout. Optimal topological design of PLA-based systems have been investigated elsewhere and are surveyed in [DEMI83g].

This paper addresses the ***optimal state assignment*** problem.

The state assignment problem has been the object of extensive theoretical research. Hartmanis [HART61], Stearns [STEA61], Karp [KARP64] and Kohavi [KOHA64] developed algebraic methods based on partition theory. Their approach was based on a reduced dependence criterion, which led to a good assignment. However no theoretical result was presented that related reduced dependencies to optimal FSM implementations. Moreover no systematic procedure was developed that could be used to encode large machines. Armstrong [ARM62a] [ARM62b] developed a method capable of coding large machines, based on a graph interpretation of the problem. However his approach was still ineffective, because i) he did not take into account the techniques of fast heuristic logic minimizers (heuristic logic minimization was not known at that time); ii) he transformed the state assignment problem into a graph embedding problem, that only partially represented the state encoding problem; iii) his graph embedding technique was ineffective. Armstrong's technique was analyzed and enhanced in [DEMI83f]. Dolotta and McCluskey [DOLO62] introduced the concept of codable columns, used for finding near-optimal solutions. Their algorithm was able to estimate the complexity of the combinational component of a FSM, as a function of partial-length state assignments. However their method was computationally efficient for small machines only. Their method was later improved by Weiner and Smith [WEIN67b], Torng [TORN68] and Story [STOR72]. Curtis [CURT69] [CURT70] considered the problem of using different types of storage elements in relation to the state assignment problem. Tracey [TRAC66] and Saucier [SAUC72] addressed the state-assignment problem in connection with race-free asynchronous machine design. Despite of all these efforts, to the best of our knowledge, no computer-aided design tool for designing FSMs is in use today for a time-effective encoding of control logic.

We present here a new technique for state assignment, based on an innovative strategy: logic minimization of the FSM combinational component is applied **before** state assignment. In Section 2 we show how logic minimization is performed on a symbolic (code independent) representation of the combinational component of the FSM. In Section 3 we introduce a constrained encoding prob-

lem, that relates the minimal symbolic representation to the class of assignments that implement the combinational component with at most as many product-terms as the cardinality of the minimal symbolic representation. In Section 4 we present a heuristic algorithm for constructing a solution to the constrained encoding problem. We then present in Section 5 the software implementation of the algorithm and the experimental results achieved by this technique.

2. SYMBOLIC COVER AND SYMBOLIC MINIMIZATION

We assume that the reader is familiar with the basic concepts and definitions of switching theory. We refer the reader to [HILL81], [BRAY84], and [HART66] for details.

Formally a FSM can be defined as a 5-tuple (X,Y,Z,δ,λ) where:

$X = \{x_1,x_2, \ldots ,x_{|X|}\}$ is the set of primary inputs;

$Y = \{y_1,y_2, \ldots ,y_{|Y|}\}$ is the set of internal states;

$Z = \{z_1,z_2, \ldots ,z_{|Z|}\}$ is the set of outputs;

$\delta: X \times Y \rightarrow Y$ is the next state function and

$\lambda: X \times Y \rightarrow Z$ $(\lambda: Y \rightarrow Z)$ is the output function for a Mealy (Moore) machine.

A finite state machine representation is said to be **incompletely-specified** if the next-state and/or output function are not specified for some input and/or present-state.

The **state assignment** , or **state encoding** problem consists of choosing a Boolean representation of the internal states of the machine. State encoding affects substantially the complexity of the FSM combinational component [HART66]. In particular, the PLA size depends heavily on the state assignment. Therefore the optimum state assignment problem for PLA-based finite state machines can be stated as follows:

Find a state assignment corresponding to a PLA implementation of minimum area

This task is formidable and some simplifying assumptions are needed. As a first step, topological compaction techniques to reduce the PLA area, such as folding [HACH82a] [DEMI83c] and partitioning [DEM83d] are not considered. Under this assumption, the PLA area is proportional to the

product of the number of rows (product-terms) times the number of columns. Both row and column cardinality depend on state encoding. The (minimum) number of rows is the cardinality of the (minimum) cover of the FSM combinational component according to a given assignment. The code-length (i.e. the number of bits used to represent the states) is related to the number of PLA columns and in particular to the number of PLA input and output columns corresponding to the present and next states. Therefore the PLA area has a complex functional dependence on state assignment. For this reason two simpler optimal state assignment problems are defined:

i) Find the assignment of minimum code length among the assignments that minimize the number of rows of the PLA.

ii) Find the assignment that minimizes the number of rows of the PLA among the assignments of given code length.

The optimum solution to the state assignment problem which minimizes the PLA area can be seen as a trade-off between the solutions to problem i) and ii). Note that the above problems are still computationally difficult and to date no method (other than exhaustive search) is known that solves them exactly. Therefore heuristic strategies are used to approximate their solution.

A method that attempts a solution to problem i) is presented in the sequel, as an intermediate step toward the solution of the complete problem. Problem i) is referred to as the optimal state assignment problem throughout this paper. Note that most of the previous state assignment techniques attempted to solve problem ii) with minimum code length (i.e. $\log_2 |Y|$). The relevance of problem ii) was related to minimizing the number of storage elements in discrete component implementations of finite state machines. Today, optimizing the total usage of silicon area (related only weakly to the number of storage elements) is the major goal in integrated circuit implementations of PLA-based finite state machines.

The state encoding technique reported in the sequel is based on an innovative strategy: instead of trying to estimate the possible simplification of the FSM combinational component after a state assignment is chosen, logic minimization is applied **before** state assignment. For this reason, logic

minimization is performed on a symbolic (code independent) representation of the combinational component of the FSM: the **symbolic cover** . The concept of symbolic cover is a generalization of the logic cover representation of combinational-logic functions.

A symbolic cover is a set of primitive elements called **symbolic implicants** . A symbolic implicant is a set of four character strings and is denoted by the 4-tuple $i\ s\ s'\ o$. The first two character strings are the symbolic implicant input-part and the last two are the symbolic implicant output-part. The input-part represents a binary-valued representation of a primary input (i) and a symbolic representation of a present state (s). The output-part represents the corresponding symbolic representation of the next-state ($s' = \delta(i,s)$) and a binary-valued representation of the primary outputs ($o = \lambda(i,s)$). Therefore it is possible to describe a FSM by means of a symbolic cover equivalent to the formal mathematical representation. Note that this representation can be generalized by letting i and o describe symbolic inputs and outputs.

Example 2.1: Consider the finite state machine described by the state table in Fig. 2.1. The following is a symbolic implicant:

0 START state-6 00

showing that a "0" primary-input value maps state "START" into "state-6" and asserts output 00. The symbolic cover is the collection of the symbolic implicants representing the state transitions:

0 START state-6 00
0 state-2 state-5 00
0 state-3 state-5 00
0 state-4 state-6 00
0 state-5 START 10
0 state-6 START 01
0 state-7 state-5 00
1 START state-4 01
1 state-2 state-3 10
1 state-3 state-7 10
1 state-4 state-6 10
1 state-5 state-2 00
1 state-6 state-2 00
1 state-7 state-6 00

Remark 2.1: Several representations of finite state machines are used in the literature and in design. The most common are **state tables , transition graphs,** [HILL81] **flow-charts** [CLAR75] and **hardware description language** programs [BARB77], [HILL78], [DEUT83]. A state table, a flow-chart and a HDL program describing the FSM of Example 2.1 are shown in Fig. 2.1, 2.2 and 2.3 respectively. Note that all these representations are equivalent to the formal representation and their transformation into a symbolic cover is straight-forward [DEMI83g].

A symbolic cover can be considered as a logic cover of a multiple-valued logic function [SU72] [HONG74], where each state takes a different logic level and is represented by a character string. A symbolic implicant having n primary input bits and m primary output bits can be seen as a $(n + 1)$-input, $(m + 1)$-output multiple-valued logic implicant.

Several notations are used to represent multiple-valued logic covers. For example, the different logic levels can be represented by integer values : $0,1,2, \ldots, p-1$. This is an extension of the binary notation to a p-valued representation.

The **positional cube** notation is used here [SU72]. A p-valued logical variable is represented by a string of p binary symbols. Value r is represented by a "1" in the r^{th} position, all others being "0". Note that the positional cube notation allows the representation of a set of values with one string. The disjunction (multiple-valued logical OR) of several values is represented by a string having "1"s in the corresponding positions. Therefore the "don't care" value is represented by a string of "1"s and the empty value by a string of "0"s.

The transformation of a symbolic cover into a multiple-valued cover with positional cube notation is straight-forward, since the latter is itself a symbolic cover and the transformation involves only

symbol translations. Therefore definitions and properties of multiple-valued logic covers carry over to symbolic covers as well and the notations "symbolic" and "multiple-valued" are interchangeable in the sequel.

Example 2.2: The symbolic cover of Example 2.2 can be translated into a multiple-valued positional-cube representation by associating a value to each state. START is represented by 1000000, state-2 by 0100000, etc.

```
0 1000000 0000010 00
0 0100000 0000100 00
0 0010000 0000100 00
0 0001000 0000010 00
0 0000100 1000000 10
0 0000010 1000000 01
0 0000001 0000100 00
1 0000010 0100000 01
1 0000100 0100000 10
1 0001000 0000010 10
1 0000001 0000010 10
1 1000000 0001000 00
1 0100000 0010000 00
1 0010000 0000001 00
```

Minimizing a symbolic cover is equivalent to finding a representation of the combinational component of the FSM with the minimum number of symbolic implicants. Finding a minimum multiple-valued cover is a computationally expensive problem. Heuristic multiple-valued logic minimizers, such as MINI [HONG74] can be used to compute a minimal (local minimum) cover. (Program MINI [HONG74] is used in general for binary-valued logic minimization; however it supports multiple-valued minimization as well.) Alternatively, the positional-cube representation can be seen as a binary-valued encoding of a multiple-valued function. This encoding is referred to as **1-hot coding,** because each value of the multiple-valued function corresponds to one and only one binary value "1" (HIGH) in the coded representation.[1] By using this representation, binary-valued minimizers, such

[1] The 1-hot representation has a different interpretation than the positional cube notation. An appropriate "don't care" set must be specified for the 1-hot representation, to specify that n-hot encodings do not represent existing states. The interested reader is referred to [BRAY84] and [DEMI83g] for details.

as PRESTO [BROW80], POP, MINI [HONG74] and ESPRESSO-II [BRAY84], can be used to obtain minimal symbolic covers. Experimental results have shown that ESPRESSO-II yields minimal (symbolic) covers that are quite close to the minimum (symbolic) cover, for problems for which the minimum cover can be determined, as shown in table 2.1.

Example 2.3: Consider the symbolic cover of Example 2.2. A minimal symbolic (multiple-valued) cover obtained by ESPRESSO-II is the following:

```
0 0110001 0000100 00
0 1001000 0000010 00
1 0001001 0000010 10
0 0000010 1000000 01
1 0000100 0100000 10
0 0000100 1000000 10
1 1000000 0001000 00
1 0000010 0100000 01
1 0100000 0010000 00
1 0010000 0000001 00
```

Consider now the first symbolic implicant from the top:

```
0 0110001 0000100 00
```

This implicant shows that input "0" maps states "2", "3" and "7" into next-state "5" and assert output "00". A similar condition is expressed by the second and third implicants.

The example above shows that the effect of symbolic (multiple-valued) logic minimization is to group together the states that are mapped by some input (or input combination) into the same next-state and assert the same output. In other words, while the original symbolic cover is a set of implicants:

$$i \; s \; \delta(i,s) \; \lambda(i,s)$$

where s represents a single state, the minimal multiple-valued cover may contain symbolic implicants in which s represents a set of states. Each state subset having more than one element and represented by a s string is termed **state group**. Given a state assignment and a state group, the corresponding **group face** (or simply **face**) is the minimal dimension subspace containing the encodings of the

states assigned to that group (or equivalently the bit-wise disjunction of the encodings assigned to the states in that group).

The goal of the state assignment technique presented here is to group together the state encodings in binary-valued logical implicants in the same way states are grouped in the minimal symbolic (multiple-valued) cover. In particular, a state encoding is sought, such that each symbolic implicant can be coded by one binary-valued implicant. For this assignment, there exists a binary-valued cover of the FSM combinational component having as many implicants as the minimal symbolic cover.

An encoding, such that each group face contains the encodings of the states included in the corresponding group and no other state encoding, satisfies the above requirement. In fact, each encoded implicant represents exactly the state-transitions related to the corresponding symbolic implicant. For this reason, a **constrained encoding problem** is considered:

> ***Given a set of groups, find an encoding such that each group face does not intersect the code assigned to any state not contained in the corresponding group.***

In view of the previous considerations, any solution to the constrained coding problem is a state assignment such that the coded Boolean cover has the same cardinality as the minimal symbolic cover.

Example 2.4: Consider the minimized symbolic cover of Example 2.3. If the states are coded as follows:

START 100
state-2 110
state-3 011
state-4 000
state-5 001
state-6 101
state-7 010

then the following Boolean cover specifies the FSM combinational component:

0*1*	00100
0*00	10100
10*0	10110
0101	10001
1001	11010
0001	10010
1100	00000
1101	11001
1110	01100
1011	01000

The Boolean cover cardinality is the same as the minimal symbolic cover cardinality.

It is important to note that most state assignment techniques are based on state grouping and an encoding scheme based on the group information [ARMS62a], [HART62], [DEMI83f]. A critical survey is presented in [DEMI83g]. A solution to the constrained encoding problem based on symbolic minimization provides the best possible assignment, when the number of product-terms of a two-level implementation is used as a cost function.

Theorem 2.1: A state assignment that is a solution of the constrained encoding problem related to a minimum symbolic cover implements each next-state function $\delta_i, i = 1,2,...,|Y|$ and output-function $\lambda_i, i = 1,2,...,|Z|$ as a sum of a minimum number of product-terms.

Proof:

Let $\mathscr{S}$ be the minimum symbolic cover of f, where f is any next-state function $\delta_i, i = 1,2,...,|Y|$ or output-function $\lambda_i, i = 1,2,...,|Z|$. Let $\mathscr{A}$ be a state assignment that is a solution of the related constrained encoding problem. Then there exists a sum-of-product representation of f, $\mathscr{B}$, such that each symbolic implicant of $\mathscr{S}$ is represented by one Boolean implicant of $\mathscr{B}$. Therefore $|\mathscr{S}| = |\mathscr{B}|$. For the sake of contradiction, suppose that there exists an assignment $\mathscr{A}'$, such that the corresponding sum-of-product representation of f, $\mathscr{B}'$ requires fewer product-terms, i.e. $|\mathscr{B}'| < |\mathscr{B}|$. Since the assignment $\mathscr{A}'$ is a symbolic representation of the states as well, there ex-

ists a symbolic cover of f, $\mathcal{S}'$ such that $|\mathcal{S}'| < |\mathcal{S}|$. Then $\mathcal{S}$ is not a minimum cover of f, and we have a contradiction. ■

There is still room to improve this technique. In the symbolic cover, the components of the next-state function $\delta_i, i = 1,2,..., |Y|$ have disjoint on-sets. However, in the encoded Boolean cover, some states codes have a non-zero entry in the same position and therefore the components of the next-state functions are not necessarily disjoint. Therefore the final minimum Boolean cover of the FSM combinational component may require fewer implicants than the sum of the each minimum cover related to each next-state and output function. In other words, state encoding transforms a minimum symbolic cover into a non-necessarily-minimum Boolean cover, because the next-states are encoded exactly as the present-states.

Example 2.5: Consider the coded cover of Example 2.2. Since the encoding of state-2 is bit-wise disjunction of the encoding of START and state-7, then the transition state-5 → state-2 under primary input 1 can be represented by the two implicants:

*001 10010
10*1 01000

which also represents the transition state-5 → START under primary input 0 and the transition state-3 → state-7 under primary input 1.

The following is a minimal Boolean cover:

0*1*	00100
0*00	10100
10*0	10110
1110	01100
1*01	01000
*101	10001
*001	10010
10*1	01000

Note that the implicant representing a transition to state-4 (encoded by 000) and asserting the primary output 00 is not needed in a Boolean representation.

In the present formulation, symbolic minimization groups present-states only. Therefore the assignment optimizes the choice of present states, and the implications of the corresponding next-state encoding is neglected. State assignment techniques that take into account next-states encoding are still under investigation.

An extensive use of 1-hot encoding for finite state machines is found in industrial designs. One reason is the lack of algorithms and programs for state encoding of large FSMs. The penalty of using 1-hot coding is related to the number of variables representing the states, which grows linearly with the state set cardinality. Note that the minimum number of state variables grows with the logarithm of the state set cardinality. Moreover, 1-hot encoding does not necessarily give a minimum sum-of-product representation.

Theorem 2.2: Any state assignment that is a solution of the constrained encoding problem implements the FSM combinational component using at most as many product-terms as a 1-hot encoded implementation.

Proof:

The on-sets of the next-state functions of a 1-hot encoded cover are disjoint. Therefore the cardinality of a minimum 1-hot encoded cover is the same as the minimum symbolic cover cardinality. A solution to the constrained encoding problem allows us to define a Boolean cover with as many product-terms as the minimum symbolic cover. Moreover this cover is not necessarily a minimum cover, because the encoded next-state functions may overlap. Thus a minimum encoded cover may be found, having fewer implicants than the minimum 1-hot encoded cover. ■

Therefore a solution to the constrained encoding problem allows the implementation of the combinational component by a PLA having fewer (or at most equal number of) columns and rows than a 1-hot encoding.

Remark 2.3: In general, symbolic minimization is affected by the machine primary inputs and/or outputs. If a minimal PLA implementation is sought, primary inputs and outputs can be considered as machine input and output states and coded as well as the internal states. However interfacing a FSM to other circuit building blocks often limits the possibility of finding an optimal encoding for primary inputs and/or outputs.

3 CONSTRAINED STATE ENCODING

The minimal symbolic representation defines the constraints of an encoding problem, whose solutions are the state assignments that allow the implementation of the FSM combinational component with at most as many product terms as the cardinality of the minimal symbolic cover. According to the definitions given in Section 2, the constrained encoding problem consists of finding a state assignment such that each group face does not intersect the code assigned to any state not contained in the corresponding group. An optimal state assignment is a minimal code-length solution to the constrained encoding problem.

The geometric interpretation of the optimal encoding problem is: ***finding the minimal dimension Boolean space in which each group face is a subspace which does not intersect the encoding assigned to any state not contained in the corresponding group*** .

State assignment is restricted here to one-to-one mappings between the state set and a subset of the vertices of the Boolean hypercube, i.e. each state encoding is a 0-dimensional subspace. This restriction is motivated as follows. A 0-dimensional state assignment that is a solution to the constrained encoding problem, can be derived from a n-dimensional (n>0) solution by assigning to each state a vertex contained in the corresponding n-dimensional assignment. Therefore a 0-dimensional solution has code-length less than or at most equal to the code-length of any n-dimensional solution.

Some definitions are introduced now to allow a formal statement of the problem. The encoding problem is studied using matrix notation. Let n_s be the number of states, n_l the number of groups and n_b the code length. The matrices we consider have pseudo-Boolean entries from the set: {0,1,*,ϕ} where * represents the don't care condition (i.e. either 1 or 0) and ϕ represents the empty value (i.e. neither 1 nor 0). Conjunction and disjunction on pseudo-Boolean variables is defined as follows:

Λ	0	1	*	ϕ
0	0	ϕ	0	ϕ
1	ϕ	1	1	ϕ
*	0	1	*	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ

V	0	1	*	ϕ
0	0	*	*	0
1	*	1	*	1
*	*	*	*	*
ϕ	0	1	*	ϕ

To be consistent with the positional-cube notation, state groups are represented by a 1-0 matrix and in particular by the subset of the columns of the minimal multiple-valued cover corresponding to the present-states.

The **constraint matrix** A is a matrix: $A \in \{0,1\}^{n_l \times n_s}$

$$A = \begin{bmatrix} a_{1\bullet} \\ a_{2\bullet} \\ \cdots \\ a_{n_l\bullet} \end{bmatrix} = \left[a_{\bullet 1} \mid a_{\bullet 2} \mid \cdots \mid a_{\bullet n_s} \right]$$

representing n_l state groups. State j belongs to group i if $a_{i,j} = 1$.

A row of the constraint matrix is said to be a **meet** if it represents the intersection of two or more state groups. A row of the constraint matrix is said to be **prime** if it is not a meet.

Example 3.1: The following constraint matrix is derived from the minimal symbolic cover of Example 2.3 and represent the state groups: { state-2 , state-3 , state-7 } , { START , state-4 } and { state-4 , state-7 }.

$$A = \begin{bmatrix} 0110001 \\ 1001000 \\ 0001001 \end{bmatrix}$$

All the rows of A are prime. If row $a = 0001000$ is appended to A, then a is a meet because it represents { state-4 }, which is the intersection between the second and the third group.

The state code matrix S is a matrix $S \in \{0,1\}^{n_s \times n_b}$

$$S = \begin{bmatrix} s_{1\bullet} \\ s_{2\bullet} \\ \cdots \\ s_{n_s\bullet} \end{bmatrix}$$

whose rows are state encodings. Our problem is to determine the state code matrix S, given a constraint matrix A.

Definition 3.1: Let $a \in \{0,1\}$ and $b \in \{0,1,*,\phi\}$. The selection of b according to a is:

$$a \bullet b = \begin{matrix} b & \text{if } a = 1 \\ \phi & \text{if } a = 0 \end{matrix}$$

Selection can be extended to two dimensional arrays and is similar to matrix multiplication.

Definition 3.2: Let $A \in \{0,1\}^{p \times q}$ and $B \in \{0,1,*,\phi\}^{q \times r}$. Then

$$A \bullet B = C = \{c_{ij}\}^{p \times r}$$

where: $c_{ij} \equiv \mathrm{V}^{q}_{k=1}\ a_{ik} \bullet b_{kj}$

Selection is useful to determine group faces corresponding to a group set and a given encoding.

The face matrix $F \in \{0,1,*,\phi\}^{n_f \times n_b}$

$$F = \begin{bmatrix} f_{1\bullet} \\ f_{2\bullet} \\ \cdots \\ f_{n_f\bullet} \end{bmatrix}$$

is the matrix whose rows are the group faces. Note that the empty group corresponds to the empty face represented by n_b ϕ entries. The face matrix can be obtained by performing the selection of S according to a constraint matrix A:

$$F = A \bullet S$$

Example 3.2: Consider the constraint matrix of Example 3.1 and the state assignment represented by:

$$S = \begin{bmatrix} 100 \\ 110 \\ 011 \\ 000 \\ 001 \\ 101 \\ 010 \end{bmatrix}$$

Then the face matrix is:

$$F = A \bullet S = \begin{bmatrix} *1* \\ *00 \\ 0*0 \end{bmatrix}$$

Note that $f_{i\bullet}$ is the minimum subspace containing the state encodings for group i. A geometrical representation of F is shown in Fig. 3.1..

Let $\bar{A}$ be the complement of the constraint matrix A. Then $\bar{F}^i \equiv \bar{a}_{\bullet i} \bullet s_{i\bullet}$ is a matrix whose rows are the encoding of state i if state i does not belong to group j, else are empty values. A state encoding matrix S is a solution of a constrained encoding problem and is said to satisfy the constraint relation for a given constraint matrix A if:

$$\bar{F}^i \ \Lambda \ F \ \equiv \ \begin{bmatrix} \bar{f}^i_{1\bullet} \Lambda f_{1\bullet} \\ \bar{f}^i_{2\bullet} \Lambda f_{2\bullet} \\ \cdots \\ \bar{f}^i_{n_l\bullet} \Lambda f_{n_l\bullet} \end{bmatrix} \ = \ \Phi \quad \forall i = 1,2, \ldots, n_s$$

where Φ is the empty matrix, i.e. a matrix whose rows have at least one ϕ entry and therefore representing no point in the Boolean space.

Example 3.3: The state matrix of Example 3.2 satisfies the constraint relation. However if the 6-th row is changed to 111, the constraint relation is no longer satisfied, because the code of state-6 intersects the first face, or equivalently:

$$\bar{a}_{\bullet 6} \bullet s_{6\bullet} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \bullet [111] = \begin{bmatrix} 111 \\ 111 \\ 111 \end{bmatrix} \quad \Lambda \quad F = \begin{bmatrix} 111 \\ 1\phi\phi \\ \phi 1\phi \end{bmatrix} \neq \Phi$$

Remark 3.1: In an implementable state assignment, all state encodings must be pair-wise disjoint. This requirement can be embedded in the constraint relation when n_s groups, consisting of one different state each, are added to the problem. In this case, n_s 0-dimensional faces correspond to the n_s state codes, and any encoding satisfying the constraint relation is such that codes are disjoint from one another.

Now the **optimal constrained encoding problem** can be formally stated as follows:

Find a state code matrix S with minimal number of columns that satisfies the constraint relation.

It is important to point out that there always exist matrices S satisfying the constraint relation.

Theorem 3.1: The identity state code matrix $S = I \in \{0,1\}^{n_s \times n_s}$ satisfies the constraint relation for any given constraint matrix A.

Proof:

Without loss of generality, let us consider the rows of A and F one at a time. Let us consider first the rows of A having 1 (0) entries only. The corresponding rows of F consist of * (ϕ) entries only, while the corresponding rows of $\bar{F}^i$ consists of ϕ (*) entries only $\forall i = 1,2, \ldots, n_s$. Let us consider now the remaining rows of A. For each remaining row i in A, let $J_i = \{j \text{ such that } a_{ij} = 0\}$. Then $f_{ij} = 0 \quad \forall j \in J_i$. Since:

$$\bar{f}_{ij}^{k} = \begin{cases} s_{kj} & \text{if } k \in J_i \\ \phi & \text{else} \end{cases} \qquad \forall k = 1,2, \ldots, n_s \quad \forall j = 1,2, \ldots, n_b$$

then: $f_{i\cdot} \ \Lambda \ \bar{f}_{i\cdot}^{k} = \Phi \quad \forall k = 1,2, \ldots n_s$, and the constraint relation is satisfied. ■

Theorem 3.2: Given any constraint matrix A, $S = A^T$ satisfies the constraint relation.

Proof:

Let $F = A \bullet S = A \bullet A^T$. Then $f_{jj} = 1, \ \forall j = 1,2, \ldots, n_l$. Since $\bar{f}_{jj}^{i} = \bar{a}_{ji} \bullet s_{ij} = \bar{a}_{ji} \bullet a_{ji}$, then:

$$\bar{f}_{jj}^{i} = \begin{cases} \phi & \text{if } a_{ji} = 1 \\ 0 & \text{if } a_{ji} = 0 \end{cases}$$

Therefore $F \ \Lambda \ \bar{F}^i = \Phi \quad \forall i = 1,2, \ldots n_s$, and the constraint relation is satisfied. ■

Theorem 3.3: If S satisfies the constraint relation for a given A then S' satisfies the constraint relation when S' is obtained from S by column permutation or column complementation.

Proof:

It is well known that permutation and complementation yield equivalent state assignments. This applies to the present formalism as well. A detailed proof is reported in [DEMI83g]. ■

Theorems 3.1 - 3.3 show that there exist different assignments satisfying any given constraint relation. However these assignment are seldom optimal, in the sense that encodings of shorter length can be found that satisfy the constraint relation. To construct length-efficient encodings satisfying the constraint relation, it is useful to consider a set of transformations on A and S. The constraint relation is invariant under a set of transformations on matrices A and S. In particular, addition (deletion) of rows and/or columns to (from) matrices A and S have been investigated in [DEMI83g]. This corresponds to modifying parameters n_i,n_s and n_b of the problem. We report here only the most relevant results.

Lemma 3.1: If S satisfies the constraint relation for a given A, then S satisfies the constraint relation for $A' = \begin{bmatrix} A \\ a_{m\cdot} \end{bmatrix}$, where $a_{m\cdot}$ is a meet of A.

Proof:

Let $F = A \bullet S$ and $F' = A' \bullet S = \begin{bmatrix} A \\ a_{m\cdot} \end{bmatrix} \bullet S = \begin{bmatrix} A \bullet S \\ a_{m\cdot} \bullet S \end{bmatrix} = \begin{bmatrix} F \\ f'_{m\cdot} \end{bmatrix}$.

For the sake of contradiction, suppose that there exists a state, say k, such that:

$$\left[\begin{bmatrix} \bar{a}_{\cdot k} \\ \bar{a}_{nk} \end{bmatrix} \bullet s_{k\cdot} \right] \Lambda \, [A' \bullet S] \neq \Phi$$

Since $[\bar{a}_{\cdot k} \bullet s_{k\cdot}] \Lambda [A \bullet S] = \Phi$, then $[\bar{a}_{mk} \bullet s_{k\cdot}] \Lambda f'_{m\cdot} \neq \Phi$. Therefore $a_{mk} = 0$. Since $a_{m\cdot}$ is a meet, there exists $a_{i\cdot}$ such that $a_{ij} \geq a_{mj}$, $\forall j = 1,2, \ldots, n_s$ and $a_{ik} = 0$. Since $f'_{m\cdot}$ is a subspace of $f_{i\cdot}$, then $[\bar{a}_{ik} \bullet s_{k\cdot}] \Lambda f'_{i\cdot} \neq \Phi$, which implies $[\bar{a}_{\cdot k} \bullet s_{k\cdot}] \Lambda A \bullet S \neq \Phi$ and we have a contradiction. ■

Let $A = \begin{bmatrix} A_P \\ A_M \end{bmatrix}$ be the partitioned constraint matrix, in which A_P (A_M) represents the prime (meet) rows.

Theorem 3.4: S satisfies the constraint relation for A if and only if S satisfies the constraint relation for A_P.

Proof:

(if) By Lemma 3.1.

(only if) If S satisfies the constraint relation for A, then S satisfies the constraint relation for any matrix obtained from A by dropping any number of rows. ■

Theorem 3.4 allows the construction of a a solution of a constrained encoding problem by considering an equivalent problem of smaller size, obtained by removing all the meet rows of A.

Lemma 3.2: If S satisfies the constraint relation for a given A, then $S' = [S \mid T]$ satisfies the constraint relation, where T is any $\{0,1\}$ matrix with n_s rows.

Proof:

Let $s'_{i\bullet} = [s_{i\bullet} \mid t_{i\bullet}]$ $\forall i = 1,2,\ldots,n_s$ and suppose by contradiction that $\exists k$ such that:

$$[\bar{a}_{\bullet k} \bullet s'_{k\bullet}] \wedge [A \bullet S'] \neq \Phi$$

Then:

$$[\bar{a}_{\bullet k} \bullet s_{k\bullet} \mid \bar{a}_{\bullet k} \bullet t_{k\bullet}] \wedge [A \bullet S \mid A \bullet T] \neq \Phi$$

$$\rightarrow [\bar{a}_{\bullet k} \bullet s_{k\bullet}] \wedge [A \bullet S] \neq \Phi$$

and we have a contradiction. ■

4 AN ALGORITHM FOR OPTIMAL STATE ASSIGNMENT

Optimal constrained encoding is a complex combinatorial optimization problem. To date, it is not known whether an optimal solution can be computed by an non-enumerative procedure. A heuristic algorithm is presented here, that constructs a state assignment satisfying the constraint relation. Experimental results show that the length of the encoding generated by the algorithm is reasonably short, and often equal to the minimum length solution when this is known.

The encoding algorithm constructs the state code matrix S by an iterative procedure. At each step a larger set of states is considered, and a coding matrix S is computed that satisfies the constraint relation for the corresponding columns of A.

The structure of the algorithm is the following:

STEP 1: Select an uncoded state (or a state subset).

STEP 2: Determine the encodings for that state (states) satisfying the constraint relation.

STEP 3: If no encoding exists, increase the state code dimension and go to STEP 2.

STEP 4: Assign an encoding to the selected state (states).;

STEP 5: If all states have been encoded, stop. Else go to STEP 1.

Before explaining the details of the algorithm, we show how such a procedure constructs an encoding satisfying the constraint relation. We investigate first some sufficient conditions for constructing a state code matrix. We assume that a state subset has already been coded, the encoding being represented by S, satisfying the constraint relation for a given A. We would like to encode another state, so that the constraint relation is satisfied for the constraint matrix $A' = [A \mid \alpha]$. The non-zero entries in column α are related to the groups to which the new state belongs. The problem consists of determining an augmented state code matrix S' that satisfies the augmented constraint relation. The most desirable situation is to obtain $S' = \begin{bmatrix} S \\ \sigma \end{bmatrix}$, where σ is the new state code. Unfor-

tunately it is not always possible to determine such a matrix S'. However it is always possible to determine a binary matrix T, such that $S' = \begin{bmatrix} S \mid T \\ \sigma \end{bmatrix}$ satisfies the augmented constraint relations. We show here that under certain conditions, we can determine a code σ, by increasing at most by one the code space dimension.

Theorem 4.1: Let S satisfy the constraint relation for a given A. Then there exists a matrix $S' = \begin{bmatrix} S \mid T \\ \sigma \end{bmatrix}$, $T \in \{0\}^{n_s \times 1}$ that satisfies the constraint relation for $A' = [A \mid \alpha]$ if either:

i) α is a column of "0"s;

ii) $\tilde{A}$ has a column of "1"s, where $\tilde{A}$ is the set of non-zero rows of A corresponding to the non-zero entries of α.

Proof:

The new face matrix is:

$$F' = A' \bullet S' = [A \mid \alpha] \bullet \begin{bmatrix} S \mid T \\ \sigma \end{bmatrix} = [A \bullet S \mid A \bullet T] \vee [\alpha \bullet \sigma]$$

Let us consider the following cases:

i) α is a column of "0"s

The new state does not belong to any group represented by A, and σ can be any encoding not covered by the existing faces.

Let $\sigma = [c \mid 1]$, where c is any binary vector of length n_b. Note that σ is disjoint from the other encodings represented by $[S \mid T]$, because the trailing bit is different. Since $\alpha \bullet \sigma$ is a matrix of empty values, then: $F' = [A \bullet S \mid A \bullet T]$. Moreover since S satisfies the constraint relation for A, by Lemma 3.2, $[S \mid T]$ satisfies the constraint relation for A as well. Therefore: $[\bar{a}_{.i} \bullet s'_{i.}] \wedge F' = \Phi \ \forall i = 1,2,...,n_s$. We need then only verify that: $[\bar{\alpha} \bullet \sigma] \wedge F' = \Phi$. By construction the right-most column of $\bar{\alpha} \bullet \sigma$ has all entries equal to "1" while the right-most column of F', i.e. $[A \bullet T]$, has all entries equal to "0" or ϕ. Since $1 \wedge 0 = \phi$, the last column of $[\bar{\alpha} \bullet \sigma]$ consists of all ϕ and the constraint relation is satisfied.

ii) $\tilde{A}$ has a column of "1"s.

Since $\tilde{A}$ has a column of "1"s, then the intersection of the groups containing the new state contain another state as well. The new encoding σ can constructed by appending a "1" to any state encoding in that intersection.

Let c be the encoding of any state corresponding to a column of "1"s in $\tilde{A}$ and let $\sigma = [c \mid 1]$.

For this choice of σ, the new face matrix is: $F' = [F \mid \beta] = [A \cdot S \mid A \cdot T] \vee [\alpha \cdot \sigma]$, where:

$$\beta_k = \begin{cases} * & \forall k \text{ such that } \alpha_k = 1 \text{ and } a_{k\bullet} \text{ is not a row of 0s} \\ 1 & \forall k \text{ such that } \alpha_k = 1 \text{ and } a_{k\bullet} \text{ is a row of 0s} \\ 0 & \forall k \text{ such that } \alpha_k = 0 \end{cases}$$

Since S already satisfies the constraints given by A, i.e.

$$[\bar{a}_{\bullet i} \cdot s_{i\bullet}] \wedge [A \cdot S] = \Phi \qquad \forall i = 1,2, \ldots, n_s$$

then:

$$[\bar{a}'_{\bullet i} \cdot s'_{i\bullet}] \wedge F' = \Phi \qquad \forall i = 1,2, \ldots, n_s$$

Therefore we need only consider the new state and in particular the right-most column, ψ, of $\bar{\alpha} \cdot \sigma$. Since $\psi_k = \phi \;\; \forall k$ such that $\alpha_k = 1$ and $\psi_k = 1 \;\; \forall k$ such that $\alpha_k = 0$, then $\psi \wedge \beta = \Phi$ and $[\bar{\alpha} \cdot \sigma] \wedge F' = \Phi$. ■

Remark 4.1: In some cases it is not necessary to increase the code-length when adding a state to the state set. Consider for example:

$$A = \begin{bmatrix} 110 \\ 101 \\ 111 \end{bmatrix} \qquad S = \begin{bmatrix} 00 \\ 01 \\ 10 \end{bmatrix}$$

If:

$$A' = \begin{bmatrix} 1100 \\ 1010 \\ 1111 \end{bmatrix}$$

Then $S' = \begin{bmatrix} S \\ \sigma \end{bmatrix}; \sigma = 11$ satisfies the augmented constraint relation.

Let us consider now the general case, in which no assumption is made on the entries in column α. Since by Theorem 3.4 a solution to a constrained encoding problem can be computed by considering the prime rows of the constraint matrix only, we assume $A = A_P$ without loss of generality.

Theorem 4.2: If S satisfies the constraint relation for a given A, then $\exists S' = \begin{bmatrix} S \mid R \mid T \\ \sigma \end{bmatrix}$; $R \in \{0,1\}^{n_s \times n_p}$; $T \in \{0\}^{n_s \times 1}$ that satisfies the constraint relation for $A' = [A \mid \alpha]$, where n_p is the number of non-zero entries in α.

Proof:

Without loss of generality, let us permute the rows of A' so that:

$$PA' = \begin{bmatrix} A^1 \mid \alpha^1 \\ A^0 \mid \alpha^0 \end{bmatrix}$$

where P is a permutation matrix, the entries of α^1 are all "1" and the entries of α^0 are all "0".

Let: $R \equiv A^{1^T}$. Let $\sigma = [c \mid d \mid 1]$, where c is a binary vector of length n_b and $d \in \{1\}^{1 \times n_p}$.

We show that S' satisfies the constraint relation for both $[A^1 \mid \alpha^1]$ and $[A^0 \mid \alpha^0]$. Consider $[A^1 \mid \alpha^1]$ first. Since $\begin{bmatrix} R \\ d \end{bmatrix} = [A^1 \mid \alpha^1]^T$ from Theorem 3.2, $\begin{bmatrix} R \\ d \end{bmatrix}$ satisfies the constraint relation for $[A^1 \mid \alpha^1]$. From Lemma 3.2 $\begin{bmatrix} S \mid R \mid T \\ c \mid d \mid 1 \end{bmatrix}$ satisfies the constraint relation for $[A^1 \mid \alpha^1]$ as well. Consider now $[A^0 \mid \alpha^0]$. Since $\bar{\alpha}^0 \bullet \sigma$ is a matrix of empty values, the face submatrix corresponding to $[A^0 \mid \alpha^0]$ is:

$$[A^0 \bullet S'] = [A^0 \bullet S \mid A^0 \bullet R \mid A^0 \bullet T]$$

Since S satisfies the constraint relation for A^0, by Lemma 3.2, [S | R | T] satisfies the constraint relation for A^0 as well. Therefore:

$$[\bar{a}^0_{\bullet i} \bullet s_{i\bullet}] \wedge [A^0 \bullet S'] = \Phi \quad \forall i = 1,2,\ldots,n_s$$

and hence the "old states" continue to satisfy the constraints. We need then only to verify that $[\bar{\alpha}^0 \bullet \sigma] \ \Lambda \ [A^0 \bullet S'] = \Phi$. We consider again the right-most column of $[\bar{\alpha}^0 \bullet \sigma]$, which is a column of "1"s. Since the right-most column of $[A^0 \bullet S']$ is $[A^0 \bullet T]$ and $[A^0 \bullet T]$ is a column of "0" or ϕ entries, then $[\bar{\alpha}^0 \bullet \sigma] \ \Lambda \ [A^0 \bullet S'] = \Phi$. ■

Theorem 4.2 shows that a state code matrix satisfying the constraint relation can be always constructed for any sequence of states, at the price of increasing the state code dimension n_b. In general, an assignment satisfying the constraint relation can always be constructed by encoding successively arbitrary blocks of a partition of the state set. Suppose a state subset has already been encoded, and we would like to encode another subset of n states.

Corollary 4.1: Let S satisfy the constraint relation for a given A. Then there exists a matrix $S' = \begin{bmatrix} S & R \\ \sigma & T \end{bmatrix}$ $R \in \{0,1\}^{n_b \times n_p}$ $T \in \{0\}^{n_b,n}$ that satisfies the constraint relation for $A' = [A \mid \alpha_1 \mid \alpha_2 \mid \ \dots \ \mid \alpha_n]$, where n_p is the number of rows of A' with non-zero entries in $\alpha_1, \alpha_2, \ \dots \ , \alpha_n$. ■

Theorems 4.1, 4.2 and Corollary 4.1 show that a state code matrix satisfying the constraint relation can be constructed by an iterative procedure. We present now the encoding algorithm in more detail, and we show how an appropriate ordering of the states and selection of the encodings is used to keep the code-length short. The input to the algorithm is the constraint matrix A. The output is the state code matrix S having n_b columns. S is initialized to the empty matrix. The selected state (or state subset) to be encoded at the current iteration of the algorithm is denoted by $\mathcal{S}$. The set of encoded and selected states is denoted by $\mathcal{E}$ and $a_{.\mathcal{E}}$ is the subset of the columns of A corresponding to $\mathcal{E}$. The algorithm is described in Pidgin C.

ENCODING ALGORITHM

$S = \phi$;

$\mathscr{E} = \phi$;

A = **compress**(A);

do {

$\mathscr{S}$ = **state-select**;

$\mathscr{E} = \mathscr{E} \cup \mathscr{S}$;

$A' = a_{\cdot \mathscr{S}}$;

do {

$\mathscr{C}$ = **candidates**(S,A');

σ = **code-select**($\mathscr{C}$);

if($\sigma = \phi$)S = **adjoin**(S)

}

while($\sigma = \phi$);

$S = \begin{bmatrix} S \\ \sigma \end{bmatrix}$;

}

while($\mathscr{E}$ is a proper subset of the state set)

Procedure **compress**(A) returns the prime rows of A. Procedure **state-select** sorts the states according to a heuristic strategy, and returns the current state (state subset) $\mathscr{S}$ to be encoded. The constraint matrix A' represents a permutation of the columns of A corresponding to the encoded and selected states in the given order.

Procedure **candidates**(S,A') returns the set of encodings that can be assigned to $\mathscr{S}$ of the same length of those represented by S. In particular: $\mathscr{C} = \{c$ such that $\begin{bmatrix} S \\ c \end{bmatrix}$ satisfies the constraint relation for $A'\}$. Note that $\mathscr{C}$ may be empty.

The **code-select** routine returns an element of $\mathscr{C}$ according to a heuristic criterion. If $\mathscr{C}$ is empty, then **code-select** (ϕ) returns ϕ, and the dimension of the code space, n_b, has to be increased. Else, the rationale of the choice of a code σ is the following. Let $u(c)$ be the number of vertices covered by at least one face. Then $\frac{u(c)}{2^{n_b}}$ represents the "utilization" of the Boolean space of current size n_b. The higher the utilization of the Boolean space is, the higher the probability is that $\mathscr{C}$ is empty at the next iteration of the algorithm and that n_b has to be increased. Since encodings are selected so that the final code length is as short as possible, σ is chosen as: $\sigma = \arg\min u(c)$.

Procedure **adjoin**(S) is invoked when the candidate set is empty, and the code space dimension has to be increased. Let $T = \{0\}^{n_b \times 1}$, i.e. T is a column of "0"s. Let $\bar{S}$ be the subset of the columns of S different from T and t be the number of columns of S equal to T. Let $R = A^{1^T}$, where A^1 is the subset of prime rows of A having a non-zero entry in columns $a_{\cdot s}$.

```
adjoin(S)
if( t < |𝒫| )return([S | T]);
else {
        R' = set of the columns of R not already adjoined to S;
        r = column of R' with minimal 1-count;
        return([S̄ | r]);
};
```

The rationale of procedure **adjoin**(S) is the following. The code space dimension is increased by adding to S columns of T and R. Columns are added one at a time, because it is desirable to find an encoding σ while adding the fewest columns to S, i.e. by the minimum increase of the code space dimension. After a finite number of iterations through **adjoin**(S) , all the columns of R and as many T columns as $|\mathscr{P}|$ are appended to S. Since the assumptions of Theorems 4.2 and Corollary 4.1 are met, the candidate set $\mathscr{C}$ is not empty. Procedure **adjoin**(S) appends columns to S in a particular se-

quence because of the following reasons. When **adjoin**(S) appends T to S, the size of the faces not related to state (states) $\mathscr{S}$ is not increased. Moreover a state code σ is always found after one iteration through procedure **adjoin**(S), when states are encoded one at a time and the conditions of Theorem 4.1 are met. Adjoining to S the columns of R one at a time corresponds to reshaping the prime faces related to $\mathscr{S}$, i.e. the faces corresponding to the prime rows in A having a non-zero entry in $a_{.\mathscr{S}}$. Reshaping consists of adding one dimension to the state code space: the new coordinate of the state codes in a prime face is set to "1", while is set to "0" for the remaining state codes. Reshaping is performed considering one prime face at a time, and by considering first the faces involving fewest states. Since, in general, states are related to many faces, reshaping a prime face leads to a size increase of some other face. Therefore the heuristic strategy tries to increase the least the face dimensions.

Example 4.2: Let us consider the constraint matrix:

$$A = \begin{bmatrix} 101 \\ 110 \\ 011 \end{bmatrix}$$

and suppose that two states have been encoded. Let:

$$A' = \begin{bmatrix} 10 \\ 11 \\ 01 \end{bmatrix} \qquad S = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Note that S satisfies the constraint relation. Let $a_{.\mathscr{S}} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$. Since $n_b = 1$ and all one dimensional codes have been assigned, the candidate set is empty. Then **adjoin**(S) returns $\begin{bmatrix} 00 \\ 10 \end{bmatrix}$. The candidate set is still empty, because: $\begin{bmatrix} S \\ c \end{bmatrix}$, $c \in \{01,11\}$ does not satisfy the constraint relation for $[A' \mid a_{.\mathscr{S}}]$. Therefore **adjoin**($S$) is invoked a second time with $S = \begin{bmatrix} 00 \\ 10 \end{bmatrix}$. Then $\overline{S} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and **adjoin**(S) returns $\begin{bmatrix} 01 \\ 10 \end{bmatrix}$. Since the candidate set is still empty, **adjoin**(S) is invoked again and returns $\begin{bmatrix} 010 \\ 100 \end{bmatrix}$. Now the candidate set is not empty and $\sigma = 001$.

Theorem 4.1: The coding algorithm terminates in a finite number of steps and constructs a state code matrix satisfying the constraint relation, regardless of the ordering of the states.

Proof:

The encoding algorithm iterates through the external **while** loop as many times as the number of state subsets $\mathscr{S}$ to be encoded, and therefore at most n_s times. For every $\mathscr{S}$ the algorithm loops through the internal **while** loop until a valid code σ is found: i.e. until $\begin{bmatrix} S \\ \sigma \end{bmatrix}$ satisfies the constraint relation for matrix A'. Since **adjoin**(S) appends to S the columns of R (or a permutation of the columns of R) and as many T columns as $|\mathscr{S}|$, by Theorem 4.2 and Corollary 4.1 a valid encoding is always found in at most $n_t + n_s$ iterations. Therefore the algorithm terminates in a finite number of steps. Since every state encoding σ is selected so that $\begin{bmatrix} S \\ \sigma \end{bmatrix}$ satisfies the constraint relation for matrix A', eventually matrix S satisfies the constraint relation for matrix A. ■

State ordering is critical to obtain an encoding with a minimal number of bits. Procedure **state-select** returns the current state (or state subset) to be encoded. As far as state selection is concerned, we assume A to be irreducible: i.e. the rows of A cannot be partitioned into two or more subsets having nonzero entries only in mutually disjoint column subsets.

> Remark 4.2: If A is reducible, it can be rearranged into a block diagonal matrix, whose blocks are irreducible. Then state ordering can be achieved by considering each block at a time.

In principle, all the states could be selected at the first iteration, and a simultaneous encoding of the state set could be attempted in increasingly larger Boolean spaces. In this case an optimum solution would be constructed by an exhaustive search. However, the computational complexity of an exhaustive encoding makes it unattractive even for medium-sized FSMs. On the other hand, states can be encoded one at the time, with a considerable saving of computing time at the expense of a possible increase in code-length. An intermediate approach takes advantage of the structure of the constraint matrix.

Definition 4.1: State p **dominates** state q if $a_{ip} \geq a_{iq}$ $\forall i = 1,2, \ldots n_l$.

Definition 4.2: A **dominating set** is a maximal cardinality set of states, such that no state dominates any other state in the set.

Note that a dominating set is not necessarily unique.

Example 4.3 Let:

$$A = \begin{bmatrix} 11010 \\ 11101 \\ 00110 \end{bmatrix}$$

A dominating set consists of the states corresponding to columns {1,3,4} or {2,3,4}.

Corollary 4.2 Let S be the state code matrix representing the state encoding of a state subset including (or equivalent to) a dominating set. Let S satisfy the constraint relation for the related A. Then, for any uncoded state, there exists a matrix $S' = \begin{bmatrix} S \mid T \\ \sigma \end{bmatrix}$, $T \in \{0\}^{n_s \cdot 1}$ that satisfies the constraint relation for $A' = [A \mid \alpha]$, where α is the corresponding constraint column. ■

Corollary 4.3: If a state s belongs to every group, then any sequence of state encodings starting with state s allows the construction of S with $n_b \leq n_s$■.

Several strategies for state encoding have been explored, but two have shown to be practical for finite state machine encoding. The first requires the encoding of a dominating set at the first iteration of the algorithm. An optimum encoding is computed for the dominating set. Since in general a dominating set is much smaller than the state set, such a computation can be done in reasonable time. Thereafter states are encoded one at a time. The criterion for state ordering is the following: the

uncoded state belonging to the largest number of prime groups (highest column count in A) is selected first. The strategy tries not to increase the state space dimension. The uncoded state with highest column count in A is the one whose encoding must be covered by the intersection of the largest number of faces. Therefore the fewer states have been encoded, the higher is the likelihood of finding such an encoding without increasing n_b. For this reason, the uncoded state with highest column count in A is the "local most critical state to code" and is encoded first. By Corollary 4.2, this strategy guarantees that an encoding satisfying the constraint relation for a given A has $n_b \leq n_s$.

The second state ordering strategy is useful when the computational burden of encoding a dominating set is too high. This is obviously dependent on the finite state machine and the computation environment. According to this strategy, states are encoded one at a time. The first state that is selected is the one with highest column count in A. Note that in general this state is the best approximation of a dominating set (it is a dominating set if the corresponding column in A has non-zero entries only). Then states are ordered as follows. Let $A(\mathscr{E})$ be the subset of the columns of A corresponding to those states belonging to some group including an encoded state, i.e. $A(\mathscr{E}) = \{a_{.i}$ such that $\exists$ a row j and an encoded state e such that$a_{ji} = 1$ and $a_{je} = 1\}$. The state corresponding to the column with the highest count in $A(\mathscr{E})$ is selected first. The rationale for this choice is similar to the previous strategy, but we restrict our attention to the states "related" to the encoded ones. No theoretical upper bound on the length of the encoding can be stated when this state selection strategy is followed. However experimental results have shown only slightly longer encodings than those obtained with the previous strategy.

Example 4.4: Consider the constraint matrix of Example 3.1 related to the FSM described in Example 2.1:

$$A = \begin{bmatrix} 0110001 \\ 1001000 \\ 0001001 \end{bmatrix}$$

where the columns correspond to START, state-2, state-3, state-4 state-5 state-6 and state-7 respectively. Note that two columns have only "0" entries; i.e. the corresponding states (state-5 and state-6) do not belong to any state group. Let us consider now the first state selection strategy.

ITERATION 1

A dominating set is: { state-4, state-7 }. Then:

$$A' = \begin{bmatrix} 01 \\ 10 \\ 11 \end{bmatrix}$$

And $S = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ satisfies the constraint relation.

ITERATION 2

Now $\mathscr{E}$ = { state-4, state-7 } and among the remaining states {START, state-2, state-3} have all the same column count. Then $\mathscr{S}$ = START, and

$$A' = \begin{bmatrix} 010 \\ 101 \\ 110 \end{bmatrix}$$

The candidate set is empty, because all possible 1-bit encodings have been assigned. The procedure adjoin is invoked, and returns: $S = \begin{bmatrix} 00 \\ 10 \end{bmatrix}$. The candidate set is now $\mathscr{C} = \{01\}$ and $S = \begin{bmatrix} 00 \\ 10 \\ 01 \end{bmatrix}$.

ITERATION 3

Now state-2 is selected:

$$A' = \begin{bmatrix} 0101 \\ 1010 \\ 1100 \end{bmatrix}$$

The candidate set $\mathscr{C} = \{11\}$ and:

$$S = \begin{bmatrix} 00 \\ 10 \\ 01 \\ 11 \end{bmatrix}$$

ITERATION 4

Now state-3 is selected and:

$$A' = \begin{bmatrix} 01011 \\ 10100 \\ 11000 \end{bmatrix}$$

The candidate set is empty and adjoin returns:

$$S = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 110 \end{bmatrix}$$

Now the candidate set is $\mathscr{C} = \{101,111\}$, $\sigma = 101$ and

$$S = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 110 \\ 101 \end{bmatrix}$$

ITERATION 5 AND 6

State-5 and state-6 can be assigned to any code not covered by any face. Hence $\mathscr{C} = \{001,011\}$. State-5 is coded by 001 and state-6 by 011. The state matrix is:

$$S = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 110 \\ 101 \\ 001 \\ 011 \end{bmatrix}$$

that satisfies the constraint relation for:

$$A' = \begin{bmatrix} 0101100 \\ 1010000 \\ 1100000 \end{bmatrix}$$

Since A' is a column permutation of A, then the rows of S must be permuted accordingly to represent the encodings of the states in the original order. Moreover note that other

encodings satisfying the constraint relation can be obtained by permuting and/or complementing the columns of S. In particular the encoding reported in Example 2.4 can be obtained by permuting the first two columns.

Note that by using the second strategy for state ordering, the same encoding matrix would have been computed. At the first iteration, state-4 would have been selected and encoded by 0; at the second iteration state-7 would have selected and encoded by 1. The other states would have followed in the same sequence.

5. KISS

KISS is a computer program for state assignment of finite state machines. The FSM description is given as input to the program in the form of a symbolic cover. Primary inputs can be described by symbolic strings and coded as well as the internal states. KISS generates an output file containing a minimal Boolean cover of the FSM combinational component. Information about state encoding is provided on request by the user. The KISS output file can be processed by a topological compaction program, such as PLEASURE [DEMI83c] or SMILE [DEMI83d], and eventually by a silicon assembler which generates the mask layout of a PLA with clocked feedback registers [DEMI84] according to a given technology.

KISS performs the following tasks. First a symbolic cover is read and a 1-hot code representation of the FSM combinational component is written to a temporary file. The "don't care set" related to the 1-hot representation is generated and appended to the temporary file. Second a two-level binary-valued logic minimizer is invoked to minimize the cover; i.e. to perform the equivalent operation of a symbolic minimization [BRAY84]. Any two-level logic minimizer can be linked to KISS. However note that the logic minimizer performs a key role to obtain a good encoding. A partially minimized symbolic cover corresponds to a partial information about state groups and eventually to an encoding close to a binary enumeration of the states. KISS has been tested in connection with minimizers POP, MINI and ESPRESSO-II. Experimental results have shown that ESPRESSO-II outperforms the other logic minimizers and enables KISS to obtain encodings leading to the minimal-area PLA implementing the FSM combinational component. (See Table 2.1) For this reasons ESPRESSO-II has been linked to KISS.

The minimized representation defines the constraints of the encoding and the encoding algorithm constructs a state code matrix. Eventually the encoded states and state groups are replaced into the minimal symbolic cover and the encoded cover is minimized again to take advantage of the possible

merging of the output parts. As an example we report the encoding constructed by KISS for the symbolic cover of Example 2.1. in Fig. 5.1.

There are two versions of program KISS. The former is coded in RATFOR (that is preprocessed into FORTRAN-77) and consists of about 2000 lines of code. The latter is coded in APL and consists of twenty APL functions.

KISS has been tested on a set of industrial finite state machines. Some results, obtained by the RATFOR version of KISS, are reported in Table 5.1 along with the execution times in seconds on a VAX-UNIX [2] computer. Table 5.2 compares the assignments generated by KISS to those obtained using a previous approach [DEMI83f], 1-hot coding and a random assignment of minimal length. Note that the number of bits used by KISS, i.e. n_b, is slightly higher than the minimum number of bits required by any assignment. Table 5.3 compares the area estimates of the segment of the PLA depending on the state representation. Table 5.4 compares the length of the encoding constructed by KISS to the minimum length assignment and to the minimum length of an encoding satisfying the constraints [MCMU83]. These results have shown that the state assignment technique presented here is an effective tool for finite state machine design.

---

[2] VAX is a trade mark of DEC corporation. UNIX is a trade mark of Bell Laboratories

6. CONCLUDING REMARKS

We have presented a new technique for state assignment of Finite State Machines, based on symbolic minimization of the FSM combinational component and on a related constrained encoding problem. Symbolic minimization yields a minimal sum-of-product representation of the next-state transition functions, independently of the state assignment. The state assignment is a solution to the constrained encoding problem and is constructed by a heuristic algorithm. The results obtained by program KISS, which implements our strategy, compare favorably to other techniques.

There are still some open problems that emerge from our analysis and that are currently being investigated. In particular the proposed state assignment technique does not optimize the PLA area with regard to next-state encoding. Different encoding techniques can be investigated. For example: i) encoding the states with the minimal number of bits while satisfying a maximal number of constraints; ii) exploring the trade-off between this last technique (that minimizes the PLA columns) versus the algorithm presented in the previous sections (that minimize the PLA rows) to achieve minimal area. It is interesting to explore implementations of the FSM combinational component other than PLAs. Since the described technique minimizes the number of product-terms implementing the next-state functions, it reduces the implementation complexity of any other two-level logic implementation. However it would be challenging to explore how to encode a finite state machine for optimal multiple-level implementation of the combinational component.

7. ACKNOWLEDGMENTS

The authors wish to thank Curt McMullen and Tiziano Villa for some helpful discussions. Richard Rudell coded program ESPRESSO-II in the C programming language and extended the program to handle symbolic minimization of large machines.

REFERENCES

[ARMS62a] 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.

[ARMS62b] D.B.Armstrong , " On the efficient Assignment of Internal Codes to Sequential Machines ", IRE Trans. Elect. Comp. , vol. EC-11, pp. 611-622, oct. 1962.

[BARB77] M.Barbacci, D.Siewiorek, R.Gordon, R.Howbrigg and S.Zuckermann, "An Architectural Research: ISP Description, Simulation and Data Collection", Proc. Nat. Comp. Conf. vol 46, 1977.

[BRAY84] R.Brayton,G.D.Hachtel,C.McMullen and A.L.Sangiovanni- Vincentelli, "ESPRESSO-II: Logic Minimization Algorithms for VLSI Synthesis", in preparation.

[BROW81] D.W.Brown, "A State-Machine Synthesizer - SMS", Des. Autom. Conf., pp. 301-304, Nashville, jun. 1981.

[CLAR75] C.R.Clare, "Designing Logic Systems using State Machines", McGraw Hill, 1975.

[CURT69] H.A. Curtis, "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-Flop Memory: Part 1", IEEE Trans on Comput., vol. C-18, pp. 1121-1127, dec. 1969.

[CURT70] H.A. Curtis, "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-Flop Memory: Part 2", IEEE Trans on Comput., vol. C-19, pp. 66-73, jan. 1970.

[DEMI83c] G.De Micheli and A.L.Sangiovanni Vincentelli , "Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications", IEEE Trans. on Comput. Aided Des. of Int. Circ, vol. CAD-2, No. 3 pp. 167-180 jul. 1983.

[DEMI83d] G.De Micheli and M.Santomauro, "SMILE: A Computer Program for Partitioning of Programmed Logic Array", Computer Aided Design No. 2 pp. 89-97, mar. 1983 and Memorandum UCB/ERL No. 82/74.

[DEMI83f] G. De Micheli, A.Sangiovanni-Vincentelli and T.Villa, "Computer-Aided Synthesis of PLA-based Finite State Machines", Int. Conf. on Comp. Aid. Des., Santa Clara pp. 154-157, sep 1983.

[DEMI83g] G. De Micheli "Computer-Aided Synthesis of PLA-based Systems" Ph.D. Dissertation, University of California,Berkeley, 1983.

[DEMI84] G. De Micheli, M.Hoffman, A.R.Newton, A.Sangiovanni-Vincentelli "A design System for PLA-based Digital Circuits" in Advances in Computer Design Jai Press, 1984.

[DEUT83] J.T.Deutsch and A.R.Newton, "Data-flow based Behavioral-level Simulation and Synthesis", Int. Conf. on Comp. Aid. Des., pp 63-64, Santa Clara, CA sep. 1983.

[DOLO64] 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.

[FLET80] W.Fletcher, "An Engineering Approach to Digital Design", Prentice Hall, 1980.

[HART61] J. Hartmanis, "On the State Assignment Problem for Sequential Machines 1", IRE Trans. Elect. Comp. , vol. EC-10 pp. 157-165, jun. 1961.

[HART66] J.Hartmanis and R.E.Stearns, "Algebraic Structure Theory of Sequential Machines", Prentice Hall, 1966.

[HILL78] F.Hill and G.Peterson, "Digital Systems: Hardware Organization and Design", Wiley, 1981.

[HILL81] F.Hill and G.Peterson, "Introduction to Switching Theory and Logical Design", Wiley, 1981.

[HONG74] S.J.Hong,R.G.Cain and D.L.Ostapko, "MINI: a Heuristic Approach for Logic Minimization", IBM Jour. of Res. and Dev., vol. 18, pp. 443-458, sep. 1974.

[HOPC79] J. Hopcroft and J. Ullman, "Introduction to Automata Theory, Languages and Computation", Addison-Wesley 1979.

[KARP64] R.Karp, "Some Techniques for State Assignment for Synchronous Sequential Machines", IEEE Trans. Elect. Comp. , vol. EC-13 pp. 507-518 , oct. 1964.

[KOHA64] Z. Kohavi, "Secondary State Assignment for Sequential Machines", IEEE Trans. on Elect. Comp. pp. 193-203 jun. 1964.

[MCMU83] C. McMullen, private communication.

[SAUC72] G.Saucier, "State Assignment of Asynchronous Sequential Machines Using Graph Techniques", IEEE Trans on Comput. vol. C-21 pp. 282-288, mar. 1972.

[STEA61] 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.

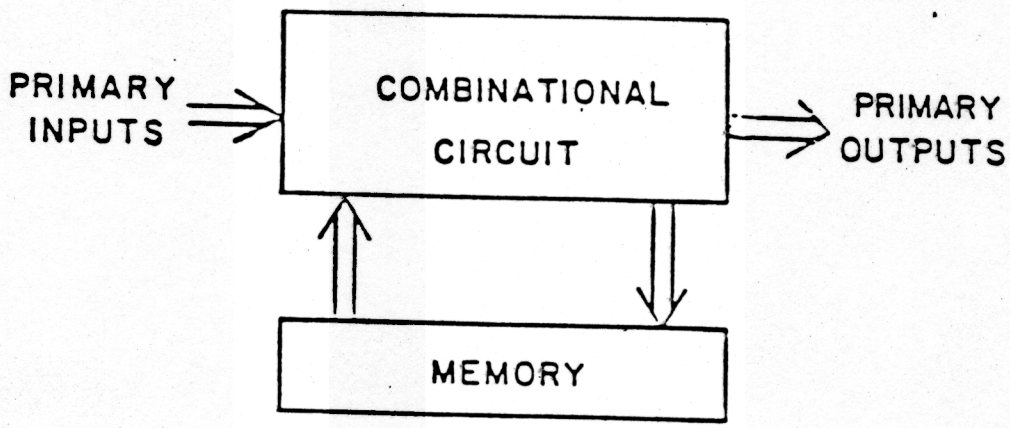
[STOR72] J.R.Story H.J.Harrison and E.A.Reinhard, "Optimum State Assignment for Synchronous Sequential Circuits", IEEE Trans. on Comp., vol. C-21 pp. 1365-1373, dec. 1972.

[SU72] S.Y.H.Su and P.T.Cheung, "Computer Minimization of Multi-Valued Switching Functions", IEEE Trans on Comput., vol 21, pp. 995-1003, 1972.

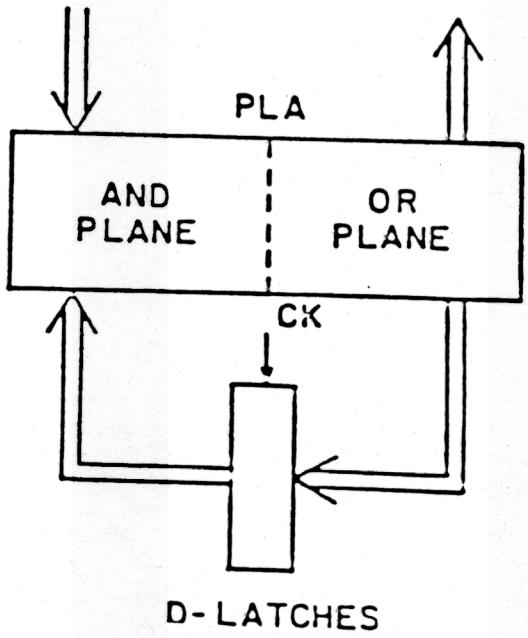
[TORN68] H.C.Torng, "An Algorithm for Finding Secondary Assignments of Synchronous Sequential Circuits", IEEE Trans on Comput., vol. C-17 pp. 416-469, may 1968.

[TRAC66] J.H.Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Trans. on Elect. Comp., vol. EC-15, pp. 551-560, aug. 1966.

[WEIN67b] P.Weiner and E.J.Smith, "Optimization of Reduced Dependencies for Synchronous Sequential Machines", IEEE Trans on Elect. Comp., vol. EC-16, pp. 835-847, dec. 1967.



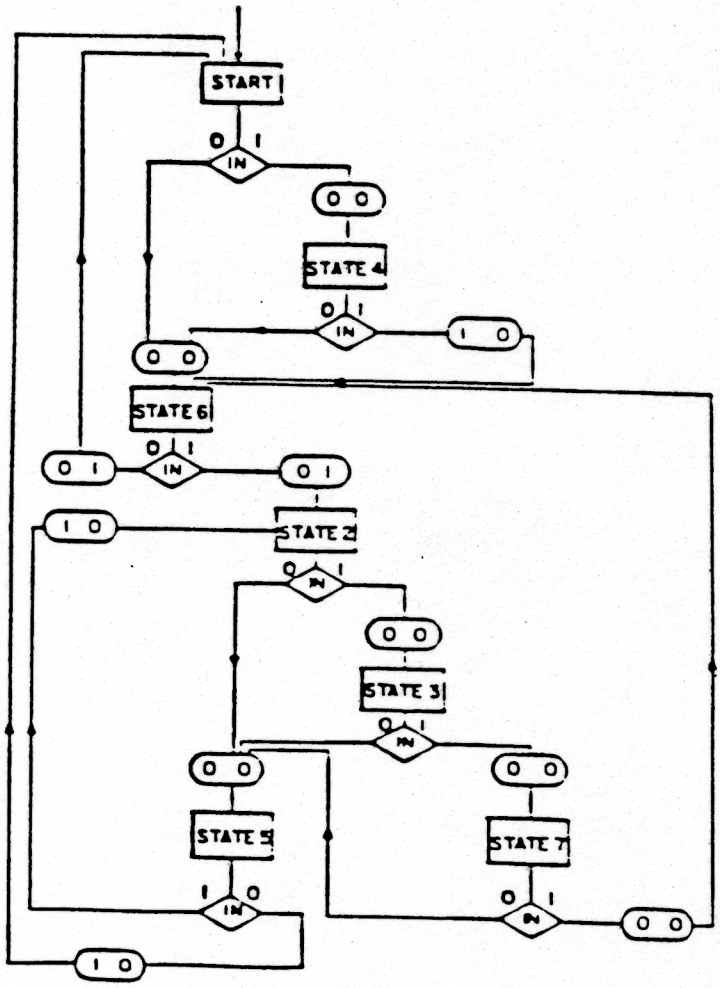
1.1 Finite State Machine.



1.2 Model of Synchronous Finite State Machine using Delay latches.

	input = 0		input = 1	
present state	next state	output	next state	output
START	state_6	00	state_4	00
state_2	state_5	00	state_3	00
state_3	state_5	00	state_7	00
state_4	state_6	00	state_6	10
state_5	START	10	state_2	10
state_6	START	01	state_2	01
state_7	state_5	00	state_6	10

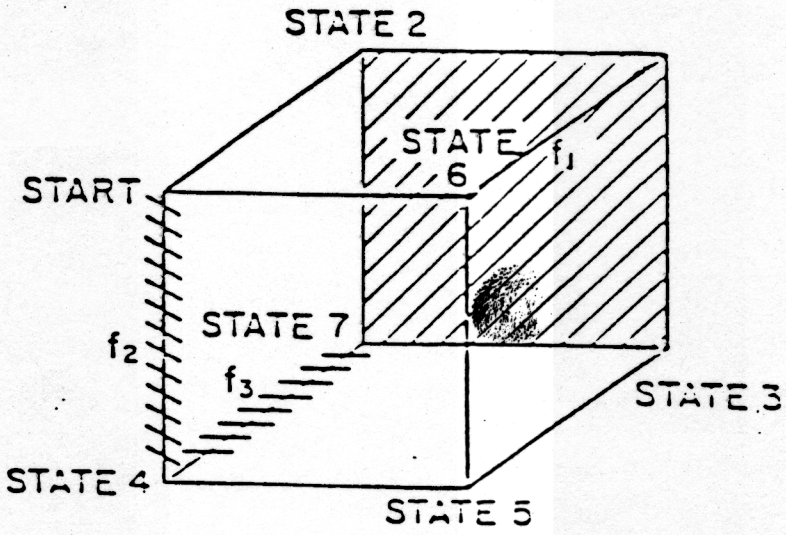
2.1 State Table.



2.2 Algorithmic State Machine representation of a FSM.

```
(which? state

 (START (if (= input 0)
        then
        (<- state_6)
        else
        (<- state_4))

     (<- output 00b))

 (state_2 (if (= input 0)
        then
        (<- state_5)
        else
        (<- state_3))

      (<- output 00b))

 (state_3 (if (= input 0)
        then
        (<- state_5)
        else
        (<- state_7))

      (<- output 00b))

 (state_4 (if (= input 0)
        then
        (<- output 00b)
        else
        (<- output 10b))

      (<- state_6))

 (state_5 (if (= input 0)
   then
   (<- START)
  else
   (<- state_2))

   (<- output 10b))

 (state_6 (if (= input 0)
   then
   (<- START)
  else
   (<- state_2))

   (<- output 01b))

 (state_7 (if (=input 0)
    then
    (<- state_5)
     (<-output 00b)
      else
    (<- state_6))

 (<- output 10b))))
```

2.3 Hardware Description Language representation of a FSM.



3.1 Geometric representation of state encoding.

STATE CODES

```
state = _START_  label =  1   code = 110
state = state_2  label =  2   code = 100
state = state_3  label =  3   code = 101
state = state_4  label =  4   code = 010
state = state_5  label =  5   code = 111
state = state_6  label =  6   code = 011
state = state_7  label =  7   code = 000
```

5.1 KISS encoding.

	POP	MINI	ESPRESSO-II	MINIMUM
FSM1	16	13	13	13
FSM2	49	27	24	24
FSM3	23	17	16	16
FSM4	108	79	55	>=54
FSM5	30	21	18	18
FSM6	13	11	10	10
FSM7	28	22	22	>=21

2.1 Comparison of minimal symbolic cover cardinalities using POP, MINI and ESPRESSO-II and the minimum cover cardinality.

LEGEND

n_i = number of inputs

n_s = number of states

n_o = number of outputs

c_1 = symbolic cover cardinality

c_2 = minimal Boolean cover cardinality

n_b = number of bits

Parameters of some Finite State Machines coded by KISS Logic minimizer: ESPRESSO-II							
FSM	n_i	n_s	n_o	c_1	c_2	n_b	time
FSM 1	4	5	1	20	9	3	4
FSM 2	8	7	5	56	21	5	31
FSM 3	8	4	5	32	12	4	10
FSM 4	4	27	3	108	29	9	748
FSM 5	4	8	3	32	16	4	11
FSM 6	2	7	2	14	7	3	4
FSM 7	2	15	3	30	17	5	26

5.1 Parameters of some Finite State Machines encoded by KISS.

Comparison of state assignments using different techniques								
	KISS		*SAP**		1-hot		minimal n_b	
FSM	c_2	n_b	c_2	n_b	c_2	n_b	c_2	n_b
FSM 1	9	3	9	3	13	5	13	3
FSM 2	21	5	33	6	24	7	44	3
FSM 3	12	4	18	2	16	4	24	2
FSM 4	29	9	80	22	55	27	87	5
FSM 5	16	4	25	5	18	8	26	3
FSM 6	8	3	9	3	10	7	8	3
FSM 7	17	5	26	14	21	15	23	4

5.2 Comparison of state encodings using different techniques: minimal cover cardinality and encoding length.

	KISS	*SAP**	1-hot	minimal n_b
FSM 1	27	27	65	39
FSM 2	105	198	168	132
FSM 3	48	36	64	48
FSM 4	261	1760	1485	435
FSM 5	64	125	144	78
FSM 6	24	27	70	24
FSM 7	85	312	330	92

5.3 Comparison of state encodings using different techniques: PLA areas.

* SAP (State Assignment Program) implemented a previous technique [DEMI83].

	Number of states	Encoding minimum length	KISS encoding length	Constrained encoding minimum length
FSM1	5	3	3	3
FSM2	7	3	5	5
FSM3	4	2	4	4
FSM4	27	5	9 (*)	7
FSM5	8	3	4	4
FSM6	7	3	3	3
FSM7	15	4	5	5

* An encoding using 7 bits has been obtained using the APL version

5.4 Comparison of the length of the encoding obtained by KISS, the minimum length encoding and the minimum-length encoding satisfying the constraints.