Jacob A. Abraham (S'71-M'74-SM'84) received the B.Sc. degree in electrical engineering from the University of Kerala, India, in 1970, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 1971 and 1974, respectively.

He is a Professor in the Department of Electrical Engineering and the Coordinatetd Science Laboratory, University of Illinois, Urbana. His research interests include VLSI systems, computer-aided design, fault-tolerant computing, and computer architecture. He is currently Coprincipal Investigator of the Department of Defense Very High Speed Integrated Circuits (VHSIC) Phase III Project in Reliable High Performance VHSIC Systems at the University of Illinois. He is also a Consultant to industry in the areas of VLSI, testability, and fault tolerant computing.

Dr. Abraham is a member of ACM and Sigma Xi.

Optimal State Assignment for Finite State Machines

GIOVANNI DE MICHELI, MEMBER, IEEE, ROBERT K. BRAYTON, FELLOW, IEEE, AND ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, IEEE

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 (FSM's), and we consider a regular and structured implementation by means of Programmable Logic Arrays (PLA's) 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.**

I. 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 [20]. 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).

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 PLA's, 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 com-

Manuscript received March 18, 1985.

G. De Micheli and R. K. Brayton are with IBM, Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

A. Sangiovanni-Vincentelli is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

0278-0070/85/0700-0269$01.00

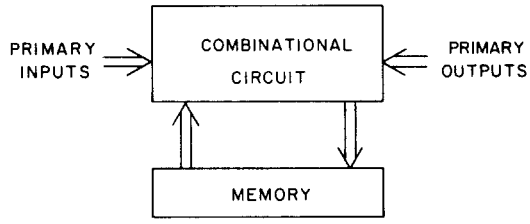


Fig. 1. Finite state machine.

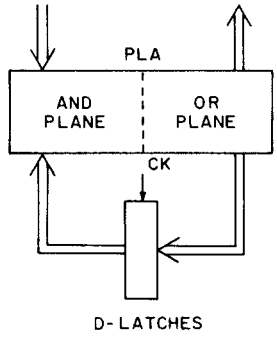


Fig. 2. Model of synchronous finite state machine using delay latches.

paction, 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 [18]. 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 FSM's 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. 2.

The computer-aided synthesis of a sequential function as a PLA-based finite state machine can be partitioned into several tasks. We refer the interested reader to [29], [10], [11], and [13] for further details. This paper addresses the *optimal state assignment* problem.

The state assignment problem has been the object of extensive theoretical research. Harmanis [16], Stearns [31], Karp [22], and Kohavi [23] 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 [1], [2] 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; and iii) his graph embedding technique was ineffective. Armstrong's technique was analyzed and enhanced in [9]. Dolotta and McCluskey [14] 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 [36], Torng [34], and Story [32]. Curtis [5], [6] considered the problem of using different types of storage elements in relation to the state assignment problem. Tracey [35] and Saucier [30] addressed the state-assignment problem in connection with race-free asynchronous machine design. Despite all these efforts, to the best of our knowledge, no computer-aided design tool for designing FSM's 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 II we show how logic minimization can be performed on a symbolic (code independent) representation of the combinational component of the FSM. In Section III we introduce a constrained encoding problem, 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 IV we present a heuristic algorithm for constructing a solution to the constrained encoding problem. We then present in Section V the software implementation of the algorithm and the experimental results achieved by this technique.

II. 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 [18], [3], and [17] for details.

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

$X = \{x_1, x_2, \cdots, x_{\|X\|}\}$	set of primary input symbols,
$Y = \{y_1, y_2, \cdots, y_{\|Y\|}\}$	set of internal states,
$Z = \{z_1, z_2, \cdots, z_{\|Z\|}\}$	set of primary output symbols,
$\delta: X \times Y \rightarrow Y$	next state function,
$\lambda: X \times Y \rightarrow Z$ ($\lambda: Y \rightarrow Z$)	output function for a Mealy (Moore) machine.

A finite 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 [17]. 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 [15] [7] and partitioning [8] are not considered in the following analysis. Under this assumption, each PLA row implements a product-term (implicant) and each column is related to an I/O of the PLA and in particular to the signals representing an encoding of the primary input an output symbols and of the present and next states. We assume *D*-latches are used to store the state information. Therefore, if n_b is the state encoding length (i.e., the number of bits used to represent the states), then $3 \times n_b$ columns of the PLA are related to the internal states. The PLA area is proportional to the product of the number of rows 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 encoding-length is related to the number of PLA columns. 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 a given code length.

The optimum solution to the state assignment problem which minimizes the PLA area can be seen as a tradeoff between the solutions to problem i) and ii) [12]. 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 [16], [1], [2] 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 feedback latches 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.

TABLE I
STATE TABLE

	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

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 combinationl 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. Symbolic covers specify combinational-logic functions by means of binary and symbolic strings.

A symbolic cover is a set of primitive elements called symbolic implicants. A symbolic implicant consists of $n \geq 2$ fields; each field is a string of characters. For our purposes, symbolic implicants have four fields ($n = 4$) corresponding to the primary inputs, present states, next states and primary outputs of the FSM respectively. We denote a symbolic implicant by the 4-tuple $i\ s\ s'\ o$. The first two fields ($i\ s$) are the symbolic implicant input part, the last two ($s'\ o$) are the symbolic implicant output part. Since we consider here the encoding of the internal states, the fields i and o are binary valued representations[1] of the primary input and primary output symbols. The fields s and s' represent a symbolic representation of the present and next states. Note that this representation can be generalized, by representing the primary inputs and outputs by symbolic fields as well.

A symbolic implicant represents a state transition if $s' = \delta(i, s)$ and $o = \lambda(i, s)$. A symbolic cover consisting of symbolic implicants representing all the state transitions is equivalent to the formal mathematical representation of a FSM.

Example 2.1: Consider the finite state machine described by the state table in Table I. The following is a

[1]In the binary-valued representation, we denote an uncomplemented variable by "1" and a complemented variable by "0." A "*" in an input-part denotes a don't care condition, a "-" in an output-part means that the implicant carries no information about that variable.

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

In general, a symbolic implicant can represent a transition from one or more states to a next-state under some input conditions. Therefore, there exist several symbolic cover representations that are equivalent among each other. A *minimum symbolic cover* is one of minimum cardinality, i.e., consisting of a minimum number of symbolic implicants. *Symbolic minimization* consists of finding such a minimum symbolic cover, i.e., is equivalent to determining a minimum sum-of-product representation independent of the encoding of the symbolic strings.

The symbolic cover representation is related to a *multiple-valued logic* representation, where each symbolic string takes a different logic value [33], [19]. 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, $\cdots$, $p-1$. This is an extension of the binary notation to a p-valued representation. The *positional cube* notation is used here [33]. A p-valued logical variable is represented by a string of p binary symbols. Value r is represented by a "1" in the rth 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 straightforward, since the latter is itself a symbolic cover and the transformation involves only symbol translations.

Example 2.2: The symbolic cover of Example 2.2 can be translated into a multiple-valued positional-cube representation by associating a positional cube 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.

The multiple-valued logic algebra, in its most common interpretation in the literature [21], is based on the work by Post [24] and on the assumption of an ordering relation among the logic values: $0 < 1 < \cdots < p-1$. The minimization of a switching function with multiple-valued outputs and represented as sum-of-products of multivalued literals depends on such an ordering [26]. Since no ordering relation among the symbolic strings is assumed in the symbolic representation of a combinational function, we perform symbolic minimization by considering any positional cube in the implicant input-parts as a different logic *value* and any positional cube in the implicant output-parts as a different logic *function*. In other words, we perform symbolic minimization by minimizing simultaneously the multiple-valued-input functions related to each next-state transition function δ_i, $i = 1, 2, \cdots, |Y|$ and each output function λ_i, $i = 1, 2, \cdots, |Z|$.

Finding a minimum multiple-valued-input cover is a computationally expensive problem. Heuristic multiple-valued-input logic minimizers, such as ESPRESSO-II [28] and MINI [19] and be used to compute a minimal (local minimum) cover. (Programs ESPRESSO-II and MINI are used in particular for binary-valued logic minimization; however in general they support multiple-valued-input minimization [3].) 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.[2] By using this representation, binary-valued minimizers, such as PRESTO [4], POP, MINI, and ESPRESSO-II, 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 a lower bound on the minimum cover can be determined [27], as shown in Table II.

[2]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 encoding do not represent existing states. The interested reader is referred to [3] and [10] for details.

TABLE II
COMPARISON OF THE MINIMAL COVER CARDINALITIES USING POP, MINI AND ESPRESSO-II AND A LOWER BOUND ON THE MINIMUM COVER CARDINALITY

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

Example 2.3: Consider the symbolic cover of Example 2.2. A minimal multiple-valued-input 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 "state-2", "state-3" and "state-7" into "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-input) 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:

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

where s represents a single state, the minimal multiple-valued-input 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 bitwise 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-input) 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 state 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. Let us suppose to know a solution to the constrained encoding problem. If the states are coded as follows:

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

then the following Boolean cover specifies the FSM combinational component:

0 1** 001 00
0 0*0 011 00
1 *00 011 10
0 011 010 01
1 001 110 10
0 001 010 10
1 010 000 00
1 011 110 01
1 110 101 00
1 101 100 00

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 [1], [16], [9]. A critical survey is presented in [10]. 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 δ_i, $i = 1, 2, \cdots, |Y|$ and output-function λ_i $i = 1, 2, \cdots, |Z|$ as a sum of a minimum number of product-terms.

Proof:

Let $\mathcal{P}$ be the minimum symbolic cover of f, where f is any next-state function δ_i, $i = 1, 2, \cdots, |Y|$ or output-function λ_i, $i = 1, 2, \cdots, |Z|$. Let $\mathcal{A}$ be a state assignment that is a solution of the related constrained encoding problem. Then there exists a sum-of-product representation of f, $\mathcal{B}$, such that each symbolic implicant of $\mathcal{P}$ is represented by one Boolean implicant of $\mathcal{B}$. Therefore $|\mathcal{P}| = |\mathcal{B}|$. For the sake of contradiction, suppose that there exists an assignment $\mathcal{A}'$, such that the corresponding sum-of-product representation of f, $\mathcal{B}'$ requires fewer product-terms, i.e., $|\mathcal{B}'| < |\mathcal{B}|$. Since the assignment $\mathcal{A}'$ is a symbolic representation of the states as well, there exists a symbolic cover of f, $\mathcal{P}'$ such that $|\mathcal{P}'| < |\mathcal{P}|$. Then $\mathcal{P}$ is not a minimum cover of f, and we have a contradiction. ■

There is still room to improve this technique. Since we perform symbolic minimization with multiple-valued-input minimization, the components of the next-state function δ_i, $i = 1, 2, \cdots, |Y|$ have disjoint on-sets [3]. 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 cardinality of each minimum cover related to each next-state and output function. In other words, state encoding transforms a minimum symbolic cover into a nonnecessarily minimum Boolean cover, because the next-states are encoded exactly as the present-states.

Example 2.5: Consider the minimal symbolic cover of Example 2.3 and in particular the following three implicants:

```
1  0000100  0100000  10
0  0000100  1000000  10
1  0010000  0000001  00
```

Consider now the encoding of Example 2.4 and let us replace the encoding in the symbolic implicants.

```
1  001  110  10
0  001  010  10
1  101  100  00
```

Note that the encoding of "state-2" is the bit-wise disjunction of the encoding of "START" and "state-7". Then the three state transitions can be represented by the following cover of cardinality two:

```
*  001  -1-  1-
1  *01  1--  --
```

because input "1" and "state-5" imply $\delta_1 = 1$ and $\delta_2 = 1$ and, therefore, a transition to "state-7." The following is a minimal Boolean cover:

```
0  1**  --1  --
0  0*0  -11  --
1  *00  -11  1-
1  110  1-1  --
1  0*1  1--  --
*  011  -1-  -1
*  001  -1-  1-
1  *01  1--  --
```

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.

By performing symbolic minimization with a multiple-valued-input minimizer, the present-state grouping does not take into account the encoding of the next states. Symbolic minimization and encoding strategies 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 FSM's. 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 product-terms 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 2.1: 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 encoded 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.

III. 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 II, 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 $\{0, 1, *, \phi\}$ 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: $A \in \{0,1\}^{n_l \times n_s}$

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

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

A row of the constraint matrix is said to be a *meet* if it represents the conjunction 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 by considering the present-field and by dropping the rows corresponding to one state only. It represents 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 conjunction between the second and the third group.

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

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

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

$$a \cdot b = \begin{cases} b, & \text{if } a = 1 \\ \phi, & \text{if } a = 0 \end{cases}$$

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

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

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

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

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

$$F = \begin{bmatrix} f_{1\cdot} \\ f_{2\cdot} \\ \cdots \\ f_{n_l\cdot} \end{bmatrix}$$

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:

$$F = A \cdot S.$$

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

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

Then the face matrix is

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

Note that $f_{i\cdot}$ is the minimum subspace containing the state encodings for group i. A geometrical representation of F is shown in Fig. 3.

Let $\bar{A}$ be the complement of the constraint matrix A, i.e., the matrix obtained from A by complementing its entries. Then $\bar{F}^i \equiv \bar{a}_{\cdot i} \cdot s_{i\cdot}$ 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:

$$\bar{F}^i \Lambda F \equiv \begin{bmatrix} \bar{f}^i_{1\cdot} \Lambda f_{i\cdot} \\ \bar{f}^i_{2\cdot} \Lambda f_{i\cdot} \\ \cdots \\ \bar{f}^i_{nl\cdot} \Lambda f_{nl\cdot} \end{bmatrix} = \Phi, \qquad \forall_i = 1, 2, \cdots, n_s$$

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

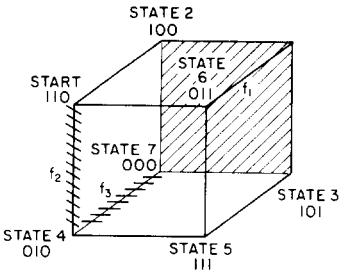


Fig. 3. Geometric representation of state encoding.

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:

$$\bar{a}_{\cdot 6} \cdot s_{6\cdot} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot [111] = \begin{bmatrix} 111 \\ 111 \\ 111 \end{bmatrix} \Lambda F$$

$$= \begin{bmatrix} 111 \\ \phi 1 \phi \\ 1 \phi \phi \end{bmatrix} \neq \Phi$$

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 satisfied 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 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 $\forall i = 1, 2, \cdots, n_s$. Let us consider now the remaining rows of A. For each remaining row i in A, let $J_i = \{j \text{ such that } a_j = 0\}$. Then $f_{ij} = 0$ $\forall j \in J_i$. Since:

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

$$\forall_j = 1, 2, \cdots, n_b$$

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

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

Proof:

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

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

Therefore, $F \Lambda \bar{F}^i = \Phi \ \forall i = 1, 2, \cdots, n_s$, 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 [10]. ■

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 [10]. This corresponds to modifying parameters n_l, 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, is a meet of A.

Proof:

Let $F = A \cdot S$ and

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

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

$$\begin{bmatrix} \bar{a}_{\cdot k} \\ \bar{a}_{mk} \end{bmatrix} \cdot s_{k\cdot} \quad \Lambda [A' \cdot S] \neq \Phi$$

Since $[\bar{a}_{\cdot k} \cdot s_{k\cdot}] \ \Lambda \ [A \cdot S] = \Phi$, then $[\bar{a}_{mk} \cdot s_{k\cdot}] \ \Lambda f'_{m\cdot} \neq \Phi$. Therefore, $a_{mk} = 0$. Since $a_{m\cdot}$ is a meet, there exists $a_{i\cdot}$ such that $a_{ij} \geq a_{mj}, \forall j = 1, 2, \cdots, n_s$ and $a_{ik} = 0$. Since $f'_{m\cdot}$ is a subspace of $f_{i\cdot}$, then $[\bar{a}_{ik} \cdot s_{k\cdot}] \ \Lambda f_{i\cdot} \neq \Phi$, which implies $[\bar{a}_{\cdot k} \cdot s_{k\cdot}] \ \Lambda \ A \cdot S \neq \Phi$ 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 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 $S' = [S|T]$ satisfies the constraint relation, where T is any $\{0, 1\}$ matrix with n_s rows.

Proof:

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

$$[\bar{a}_{\cdot k} \cdot s'_{k\cdot}] \ \Lambda \ [A \cdot S'] \neq \Phi.$$

Then

$$[\bar{a}_{\cdot k} \cdot s_{k\cdot} \mid \bar{a}_{\cdot k} \cdot t_{k\cdot}] \ \Lambda \ [A \cdot S | \ A \cdot T] \neq \Phi$$

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

and we have a contradiction. ■

IV. An Algorithm for Optimal State Assignment

Optimal constrained encoding is a complex problem of combinatorial optimization. To date, it is not known whether an optimal solution can be computed by an nonenumerative 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 $A' = [A|\alpha]$. The nonzero 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. Unfortunately it is not always possible to determine such a matrix S'. However it is always possible to determine a binary matrix T, such that $S' = \begin{bmatrix} S|T \\ \sigma \end{bmatrix}$ 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

$$S' = \begin{bmatrix} S|T \\ \sigma \end{bmatrix}, \qquad T \in \{0\}^{n_s \times 1}$$

that satisfies the constraint relation for $A' = [A|\alpha]$ is 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 nonzero rows of A corresponding to the nonzero entries of α.

Proof:

The new face matrix is

$$F' = A' \cdot S' = [A|\alpha] \cdot \begin{bmatrix} S|T \\ \sigma \end{bmatrix} = [A \cdot S|A \cdot T] \vee [\alpha \cdot \sigma].$$

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

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

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

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

$$[a_{.i} \cdot s_{i.}] \wedge [A \cdot S] = \Phi, \qquad \forall i = 1, 2, \cdots, n_s$$

then,

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

Therefore, we need only consider the new state and in particular the rightmost 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

$$\exists S' = \begin{bmatrix} S|R|T \\ \sigma \end{bmatrix}; \; R \in \{0, 1\}^{n_s \times n_p}; \; T \in \{0\}^{n_s \times 1}$$

that satisfies the constraint relation for $A' = [A|\alpha]$, where n_p is the number of nonzero entries in α.

Proof:

Without loss of generality, let us permute the rows of

A' so that:

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

where P is a permutation matrix, the entries of α^1 are all "1" and the entries of α^0 are all "0". Let: $R \equiv A^{1T}$. Let $\sigma = [c|d|1]$, 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 $[A^1|\alpha^1]$ and $[A^0|\alpha^0]$. Consider $[A^1|\alpha^1]$ first. Since $\begin{bmatrix} R \\ d \end{bmatrix} = [A^1|\alpha^1]^T$ from Theorem 3.2, $\begin{bmatrix} R \\ d \end{bmatrix}$ satisfies the constraint relation for $[A^1|\alpha^1]$. From Lemma 3.2 $\begin{bmatrix} S|R|T \\ c|d|1 \end{bmatrix}$ satisfies the constraint relation for $[A^1|\alpha^1]$ as well. Consider now $[A^0|\alpha^0]$. Since $\bar{\alpha}^0 \cdot \sigma$ is a matrix of empty values, the face submatrix corresponding to $[A^0|\alpha^0]$ is:

$$[A^0 \cdot S'] = [A^0 \cdot S|A^0 \cdot R|A^0 \cdot T]$$

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,

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

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

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

$$S' = \begin{bmatrix} S|R|T \\ \sigma \end{bmatrix} \qquad R \in \{0, 1\}^{n_s \times n_p} \qquad T \in \{0\}^{n_s, n}$$

that satisfies the constraint relation for

$$A' = [A|\alpha_1|\alpha_2| \cdots |\alpha_n]$$

where n_p is the number of rows of A' with nonzero entries in $\alpha_1, \alpha_2, \cdots, \alpha_n$. ■

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 $a_{\cdot\mathcal{E}}$ is the subset of the columns of A corresponding to $\mathcal{E}$. The algorithm is described in Pidgin C.

ENCODING ALGORITHM

```
S = φ;
E = φ;
A = compress (A);
do {
    S = state-select;
    E = E ∪ S;
    A' = a_{·E};
    do {
        C = candidates (S, A');
        σ = code-select (C);
        if (σ = φ) S = adjoin (S)
    }
    while (σ = φ)
    S = [S; σ];
}
while (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) $\mathcal{S}$ 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 $\mathcal{S}$ of the same length of those represented by S. In particular: $\mathcal{C} = \{c$ such that $\begin{bmatrix} S \\ c \end{bmatrix}$ satisfies the constraint relation for $A'\}$. Note that $\mathcal{C}$ may be empty.

The **code-select** routine returns an element of $\mathcal{C}$ according to a heuristic criterion. If $\mathcal{C}$ 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 $u(c)/2^{n_b}$ 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 $\mathcal{C}$ 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 $T = \{0\}^{n_s \times 1}$, 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 $R = A^{1T}$, where A^1 is the subset of prime rows of A having a nonzero entry in columns $a_{\cdot\mathcal{S}}$.

```
adjoin(S)
if (t < |S|) 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 $([\bar{S}|r])$;
};

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 $|\mathbb{S}|$ are appended to S. Since the assumptions of Theorem 4.2 and Corollary 4.1 are met, the candidate set $\mathbb{C}$ is not empty. Procedure **adjoin**(S) appends columns to S in particular sequence because of the following reasons. When **adjoin**(S) appends T to S, the size of the faces not related to state (states) $\mathbb{S}$ 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 $\mathbb{S}$, i.e., the faces corresponding to the prime rows in A having a nonzero entry in $a_{\cdot\mathbb{S}}$. 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

$$a_{\cdot\mathbb{S}} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Since $n_b = 1$ and all one-dimensional codes have been designed, the candidate set is empty. Then **adjoin**(S) returns $[^{00}_{10}]$. The candidate set is still empty, because: $[^{S}_{c}]$, $c \in \{01, 11\}$ does not satisfy the constraint relation for $[A'|a_{\cdot\mathbb{S}}]$. Therefore, **adjoin**(S) is invoked a second time with $S = [^{00}_{10}]$. Then $\bar{S} = [^{0}_{1}]$ and **adjoin**(S) returns $[^{01}_{10}]$. Since the candidate set is still empty, **adjoin**(S) is invoked again and returns $[^{010}_{100}]$. 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 $\mathbb{S}$ to be encoded, and therefore at most n_s times. For every $\mathbb{S}$ the algorithm loops through the internal **while** loop until a valid code σ is found: i.e. until $[^{S}_{\sigma}]$ 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 $|\mathbb{S}|$, by Theorem 4.2 and Corollary 4.1 a valid encoding is always found in at most $n_l + n_s$ iterations. Therefore, the algorithm terminates a finite number of steps. Since every state encoding σ is selected so that $[^{S}_{\sigma}]$ 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 FSM's. On the other hand, states can be encoded one at a 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 $a_{ip} \geq a_{iq}$ $\forall i = 1, 2, \cdots, n_l$.

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 $A' = [{}^{S|T}_{\sigma}]$, $T \in \{0\}^{n_s,1}$ that satisfies the constraint relation for $A' = [A|\alpha]$, 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 of 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 nonzero entries only). Then states are ordered as follows. Let A ($\mathcal{E}$) be the subset of the columns of A corresponding to those states belonging to some group including an encoded state, i.e. A ($\mathcal{E}$) = $\{a_{\cdot i}$ 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 ($\mathcal{E}$) 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 $S = [{}^{0}_{1}]$ satisfies the constraint relation.

ITERATION 2

Now $\mathcal{E}$ = {state-4, state-7} and among the remaining states {START, state-2, state-3} have all the same column count. Then $\mathcal{S}$ = 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: $S = [{}^{00}_{10}]$. The candidate set is now

$$\mathcal{C} = \{01\} \text{ and } S = \begin{bmatrix} 00 \\ 10 \\ 01 \end{bmatrix}.$$

ITERATION 3

Now state-2 is selected:

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

The candidate set $\mathcal{C}$ = {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 $\mathcal{C} = \{101, 111\}$, $\sigma = 101$, and

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

ITERATIONS 5 AND 6

State-5 and state-6 can be assigned to any code not covered by any face. Hence $\mathcal{C} = \{001, 011\}$. 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.

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.

V. KISS

KISS (Keep Internal States Simple) 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 [7] or SMILE [8] and eventually by a silicon assembler which generates the mask layout of a PLA with clocked feedback registers [11], [28] according to a given technology.

KISS performs the following tasks. First a symbolic cover is read and a positional-cube representation of the FSM combinational component is written to a temporary file. Then multiple-valued-input minimizer ESPRESSO-II [28] is invoked to minimize the cover. 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 multiple-valued cover and the encoded cover is minimized again to take advantage of the possible merging of the output parts.

In the first version of KISS, we experimented with different binary-valued logic minimizers (including ESPRESSO-II) to minimize the symbolic cover. When KISS is used in conjunction with a binary-valued minimizer, an appropriate "don't care set" is generated by the program and fed to the minimizer along with the positional-cube representation. (See [10] for details.) 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 II). For this reason ESPRESSO-II has been linked to KISS.

As an example we report the encoding constructed by KISS for the symbolic cover of Example 2.1:

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

There are two versions of program KISS. The former is coded in RATFOR (that is preprocessed into Fortran-77)

TABLE III
PARAMETERS OF SOME FINITE STATE MACHINES ENCODED BY KISS

Parameters of Some Finite State Machines Coded by KISS (Logic minimizer: ESPRESSO-II)								
FSM	*ni*	*ns*	*no*	*c*1	*c*2	*c*3	*nb*	Time (sec)
FSM 1	4	5	1	20	13	10	3	4
FSM 2	8	7	5	56	24	22	5	31
FSM 3	8	4	5	32	16	14	4	10
FSM 4	4	27	3	108	55	48	9	748
FSM 5	4	8	3	32	18	17	4	11
FSM 6	2	7	2	14	10	8	3	4
FSM 7	2	15	3	30	23	17	5	26

ni number of input symbols
ns number of internal states
no number of output symbols
*c*1 initial cardinality of the symbolic cover
*c*2 minimal symbolic cover cardinality
*c*3 minimal boolean cover cardinality
nb encoding length

TABLE IV
COMPARISON OF STATE ENCODINGS USING DIFFERENT TECHNIQUES: MINIMAL COVER CARDINALITY AND ENCODING LENGTH

	KISS		SAP		1-HOT		MIN-LENGTH	
FSM	*c*3	*nb*	*c*3	*nb*	*c*3	*nb*	*c*3	*nb*
FSM 1	10	3	10	3	13	5	13	3
FSM 2	22	5	33	6	24	7	44	3
FSM 3	14	4	18	2	16	4	24	2
FSM 4	48	9	80	22	55	27	87	5
FSM 5	17	4	25	5	18	8	26	3
FSM 6	8	3	9	3	10	7	8	3
FSM 7	17	5	26	14	23	15	23	4

*c*3 minimal boolean cover cardinality
nb encoding length

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 KISS at University of California, Berkeley, are reported in Table III along with the execution times in seconds on a VAX-UNIX[3] computer. The state tables for these finite state machines are not reported here for lack of space, but are available from the authors on request.

Table IV compares the assignments generated by the RATFOR version of KISS to those obtained using a previous approach (Program SAP [9]), 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 ceiling of $\log_2 n_s$. We would have liked to compare our results with those obtained by other computer programs. Unfortunately, to the best of our knowledge, we are not aware of any computer program for optimal state assignment in public domain, distributed and supported.

[3]VAX is a trademark of DEC corporation. UNIX is a trademark of AT&T Bell Laboratories.

TABLE V
COMPARISON OF THE LENGTH OF THE ENCODING OBTAINED BY KISS, THE MINIMUM LENGTH ENCODING AND THE MINIMUM-LENGTH ENCODING SATISFYING THE CONSTRAINTS

	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.

Table V 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 [24].

An entire control-unit of a microprocessor has been encoded by the APL version of KISS. The FSM had 93 states, 18 primary inputs and 14 primary outputs. The symbolic cover was specified by 3178 symbolic implicants. The state set was encoded by 12 bits and a minimal Boolean cover with 660 product-terms was derived. However, preliminary experiments have shown that a further reduction of the Boolean cover (25 percent) can be achieved by exploiting the encoding of the next-states. For this reason, techniques for symbolic minimization and next-state encoding are under investigation and will be added to KISS in the near future.

VI. 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 is achieved at the moment by multiple-valued-input minimization and 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. Recent re-

sults on different encoding techniques have been reported in [12]. It is interesting to explore implementations of the FSM combinational component other than PLA's. 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.

Acknowledgment

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

[1] D. B. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 466–472, Aug. 1962.

[2] D. B. Armstrong, "On the efficient assignment of internal codes to sequential machines," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 611–622, Oct. 1962.

[3] R. Brayton, G. D. Hachtel, C. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic, 1984.

[4] D. W. Brown, "A State-Machine Synthesizer—SMS," in *Des. Autom. Conf.*, pp. 301–304, Nashville, TN, Jun. 1981.

[5] H. A. Curtis, "Systematic procedures for realizing synchronous sequential machines using flip-flop memory: Part 1," *IEEE Trans. Computer.*, vol. C-18, pp. 1121–1127, Dec. 1969.

[6] ——, "Systematic procedures for realizing synchronous sequential machines using flip-flop memory: Part 2," *IEEE Trans. Computer*, vol. C-19, pp. 66–73, Jan. 1970.

[7] G. De Micheli and A. L. Sangiovanni-Vincentelli, "Multiple constrained folding of programmable logic arrays: Theory and applications," *IEEE Trans. Computer Aided Design*, vol. CAD-2, pp. 167–180, Jul. 1983.

[8] 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; Also, Memo UCB/ERL No. 82/74, Univ. California, Berkeley.

[9] G. De Micheli, A. Sangiovanni-Vincentelli, and T. Villa, "Computer-aided synthesis of PLA-based finite state machines," in *Int. Conf. on Comp. Aid. Des.*, Santa Clara, CA, pp. 154–157, Sept. 1983.

[10] G. De Micheli, "Computer-aided synthesis of PLA-based systems," Ph.D. dissertation, Univ. California, Berkeley, 1983.

[11] G. De Micheli, M. Hoffman, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "A design system for PLA-based digital circuits," in *Advances in Computer Engineering Design*, Jai Press, 1984 (in print).

[12] G. De Micheli, "Optimal encoding of control logic," in *Int. Conf. on Circ. and Comp Des.*, Rye, NY, Sept. 1984.

[13] D. L. Dietmeyer and M. H. Doshi, "Automated PLA synthesis of the combinational logic of a DDL description," *J. Des. Aut. Fault. Tol. Comput.*, vol. 3, no. 3-4, pp. 241–257, 1979.

[14] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 549–562, Oct. 1964.

[15] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "An algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 63–77, Apr. 1982.

[16] J. Hartmanis, "On the state assignment problem for sequential machines 1," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 157–165, Jun. 1961.

[17] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, 1966.

[18] F. Hill and G. Peterson, *Introduction to Switching Theory and Logical Design*. New York: Wiley, 1981.

[19] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443–458, Sep. 1974.

[20] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.

[21] S. K. Hurst, "Multiple-valed logic- its status and its future," *IEEE Trans. Computer*, vol. C-33, no 12, pp. 1160–1179, Dec. 1984.

[22] R. Karp, "Some techniques for state assignment for synchronous sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 507–518, Oct. 1964.

[23] Z. Kohavi, "Secondary state assignment for sequential machines," *IEEE Trans. Electron. Comput.*, pp. 193–203, Jun. 1964.

[24] C. McMullen, private communication.

[25] E. L. Post, "Introduction to a general theory of elementary propositions," *Amer. J. Math.*, vol. 43, pp. 163–185, 1921.

[26] D. Rine, *Computer Science and Multiple-Valued Logic*. Amsterdam, The Netherlands: North Holland, 1977.

[27] R. Rudell, private communication.

[28] R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for multi-valued logic minimization," in *Proc. Custom Int. Circ. Conf.*, Portland, OR, May 1985.

[29] R. Rudell, A Sangiovanni-Vincentelli, and G. De Micheli, "A FSM synthesis system," in *Proc. Int. Symp. Circ, and Syst.*, Kyoto, Japan, June 1985.

[30] G. Saucier, "State assignment of asynchronous sequential machines using graph techniques," *IEEE Trans. Comput.*, vol. C-21, pp. 282–288, Mar. 1972.

[31] R. E. Stearns and J. Hartmanis, "On the state assignment problem for sequential machines 2," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 593–603, Dec. 1961.

[32] J. R. Story, H. J. Harrison, and E. A. Reinhard, "Optimum state assignment for synchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-21, pp. 1365–1373, Dec. 1972.

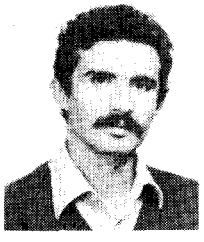
[33] S. Y. H. Su and P. T. Cheung, "Computer minimization of multi-valued switching functions," *IEEE Trans. Comput.*, vol. 21, pp. 995–1003, Dec. 1972.

[34] H. C. Torng, "An Algorithm for finding secondary assignments of synchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-17, pp. 416–469, May 1968.

[35] J. H. Tracey, "Internal state assignment for asynchronous sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 551–560, Aug. 1966.

[36] P. Weiner and E. J. Smith, "Optimization of reduced dependencies for synchronous sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 835–847, Dec. 1967.

*



Giovanni De Micheli (S'79–M'83) was born in Milano, Italy, in 1955. He received the Dr. Eng. degree (summa cum Laude) in Nuclear Engineering from the Politecnico di Milano, Italy, in 1979, the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley in 1980 and 1983, respectively.

He was granted a Fulbright Scholarship in 1980, a Rotary International Fellowship in 1981, and an IBM Fellowship for VLSI in 1982 and 1983. He received a Best Paper Award at the 20th Design Automation Conference, in June 1983. He spent the fall 1981 as Consultant in residence at Harris Semiconductor, Melbourne, FL. In 1983 he was Assistant Professor at the Department of Electronics of the Politecnico di Milano. In 1984 he joined the technical staff of IBM T. J. Watson Research Center, Yorktown Heights NY, where he is currently Project Leader of the Design Automation work station group. His research interest include several aspects of the computer-aided design of integrated circuits with particular emphasis on silicon compilation, logic synthesis, optimization, and verification of VLSI circuits.

*

Robert K. Brayton (M'75–SM'78–F'81) received the B.S.E.E. degree from Iowa State University, Ames, in 1956, and the Ph.D. degree in mathematics from MIT, Cambridge, MA, in 1961.

While at MIT, he developed the first LISP compiler. He has been at the IBM T. J. Watson Research Center in Yorktown Heights, NY since 1961

spending the years 1966 at MIT and 1976 at Imperial College as a visiting professor. He was Assistant Director of Mathematical Sciences at IBM during 1971–1972. He is presently a second-level manager of the Mathematical Algorithms group. He is winner of the IEEE Best Paper award (Circuit and Systems Group), and has received two IBM Outstanding Innovation Awards and an IBM Patent Award. During 1970–1972, he was a NSF Chatauqua Lecturer on Mathematical Models and Computing. He has been a member of the NSF Advisory Panel in Mathematics, the IEEE Circuits and Systems ADCOM, and subcommittees on nonlinear networks, large-scale systems, and computer-aided design. He is the author of over 70 technical papers and two books. His fields of interest have been nonlinear networks, sparse matrices, numerical analysis, stability theory, computer-aided design, character recognition, and optimization. His most recent research has been in the construction and design of an automatic chip compiler, involving logic synthesis and minimization, state assignment, and high level design languages.

Dr. Brayton is a Fellow of the AAAS.

*

Alberto Sangiovanni-Vincentelli (M'74–SM'81–F'83), for a photograph and biography please see page 166 of this issue.

An Integrated Automated Layout Generation System for DSP Circuits

JAN M. RABAEY, MEMBER, IEEE, STEPHEN P. POPE, ROBERT W. BRODERSEN, FELLOW, IEEE

Abstract—**An integrated CAD system for the automated design of digital signal-processing (DSP) circuits for audio and telecommunication applications is described. The system uses as unique input a symbolic description of the algorithm. This representation is translated into an actual layout using a two-step process. First, the symbolic input is mapped into the target architecture, which consists basically of a set of concurrent processors and dedicated I/O circuitry. The resulting hardware configuration is compiled into a layout description through a full exploitation of the hierarchy and the modularity of the architecture, calling consecutively a tiler, a floorplanner, and a global placement and routing tool. All these layout generation tools are able to support a wide range of technologies.**

The provision of a dedicated register transfer level simulator allows for the efficient debugging and algorithmic checking of the real-time operating signal-processing algorithms.

The efficiency and the usefulness of this design methodology has been demonstrated by multiple examples. Experiments have shown that the use of these techniques can reduce the complete design process to a few months.

I. Introduction

REAL-TIME DIGITAL signal processing with its continuous data flow and its complex algorithms poses extreme computational demands which, most of the time, cannot be met by general-purpose processors or machines. This has resulted in the development of a range of dedicated signal-processing architectures with an increased data throughput as a common feature. This can be obtained through a full exploitation of the inherent parallelism in the signal-processing algorithms, using pipelining and concurrency. Increases in the computational performance can also be achieved by the use of functional units optimized for digital signal processing, as parallel multipliers, address arithmetic units, and dedicated I/O processors.

Until recently, a full utilization of these architectural ideas on a single integrated circuit has been prohibited by technological restrictions. As a result of the advances in VLSI technology, the amount of the circuitry and the complexity of the algorithms which can be implemented on a single integrated circuit has increased dramatically. This has resulted in the development of a number of single-chip solutions for digital signal-processing applications. The approaches to design these chips can be divided into general-purpose and custom designs.

The general-purpose signal processor can be considered as the signal-processing equivalent of the microprocessor and basically uses pipelining and a parallel multiplier to increase the data throughput (e.g., [1]). The advantage of this approach is the programmability, which avoids the expensive design times of the custom approach. The generality of these processors is, however, a major handicap: the processor does not fit the specifications and the requirements of a specific algorithm, such as wordlength, datapath, memory size, data throughput, and I/O requirements.

Fully custom designs on the other hand can be optimally designed to meet the requirements of a specific applica-

Manuscript received February 15, 1985; revised April 16, 1985. This work was supported in part by the Defense Advance Research Projects Agency under Contract MDA903-79-C-0429.

J. M. Rabaey is with the Electronics Research Laboratory, the University of California, Berkeley, on leave from the Katholieke Universiteit of Leuven, Leuven, Belgium.

S. P. Pope was with the University of California, Berkeley. He is now with Cyclotomics Corp., Berkeley, CA 94704.

R. W. Brodersen is with the Department of Computer Sciences, University of California, Berkeley.

0278-0070/85/0700-0285$01.00 © 1985 IEEE