KISS: A PROGRAM FOR OPTIMAL STATE ASSIGNMENT OF FINITE STATE MACHINES

Giovanni De Micheli
Robert Brayton
IBM-Thomas J. Watson Research Center, Yorktown Heights, NY 10598

Alberto Sangiovanni-Vincentelli
Dept. EECS, University of California, Berkeley, CA 94720

Abstract. We address the state assignment problem for Deterministic Synchronous Finite State Machines, implemented by Programmable Logic Arrays and feedback registers. Optimal state assignment aims at a minimal-area implementation. We present 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 encodings that allow to implement 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. A heuristic algorithm is used to construct a solution of the constrained encoding problem. The algorithm has been coded in computer program KISS, and tested on different examples of Finite State Machines. Experimental results are reported.

1. INTRODUCTION

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 **synchronous deterministic finite state machine** 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. We use clocked Delay (D) registers to store the state information.

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 Programmable Logic Array (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 computer-aided synthesis of a sequential circuit as a PLA-based finite state machines can be partitioned in several tasks [DEMI83g]. We address here the ***optimal state assignment problem*** . The state assignment problem has been the object of extensive theoretical research. A survey of the major results is reported in [DEMI83g]. 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.

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], [BRAY84b], and [HART66] for details.

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 [DEMI83c] and partitioning [DEMI83d] 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 (PLA I/O). 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. the logarithm of the number of states). 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. Symbolic covers were introduced by the authors in previous papers [DEMI83f] [DEMI84b] [DEMI84d] to specify a combinational function by means of binary and symbolic strings. In particular, states are represented by mnemonic strings, while primary inputs and outputs are represented by binary strings. (Note that primary inputs and/or outputs could be represented by mnemonic strings as well.)

Minimizing a symbolic cover is equivalent to finding a representation of the combinational component of the FSM with the minimum number of symbolic implicants. Symbolic minimization is achieved here by minimizing a multiple-valued-input logic function [SU72] [HONG74], where each symbolic string representing a present-state corresponds to a different logic value and each symbolic string representing a next-state corresponds to a different output function. Note that in the present approach we do not represent next-states by different logic values, because the minimization of a multiple-valued-output function depends on the output representation (i.e. the mapping between next-states and the values) and therefore is no longer code independent. As a consequence, the FSM combinational component is optimized with regard to the encoding of present-states only.

Finding a minimum multiple-valued cover is a computationally expensive problem. Heuristic multiple-valued logic minimizers, such as ESPRESSO-II [BRAY84a] [BRAY84b] and MINI [HONG74] can be used to compute a minimal (local minimum) cover. (Program MINI and ESPRESSO-II are used in general for binary-valued logic minimization; however they support multiple-valued minimization as well.) Experimental results have shown that ESPRESSO-II yields minimal covers that are quite close to the minimum (symbolic) cover, for problems for which the minimum (symbolic) cover can be determined. We refer the reader to [DEMI84b] and [DEMI84d] for detailed examples and further properties of symbolic minimization. Note that binary-valued logic minimizers, such as PRESTO [BROW81], POP, MINI and ESPRESSO-II, can be also used to minimize multiple-valued-input functions, by specifying an appropriate "don't care" set [BRAY84b] [DEMI83g].

Symbolic (multiple-valued) minimization groups together the states that are mapped by some input (or input combination) into the same next-state and assert the same output. We call **state group** each proper state subset having this property. 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 en-

coding 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 encoding problem is a state assignment such that the coded Boolean cover has the same cardinality as the minimal symbolic cover. We proved in [DEMI84b] that there always exist solutions to this problem. An optimal state assignment is a minimal code-length solution to the constrained encoding problem. The geometric interpretation of the optimal encoding problem is:

Find 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 .

3 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 a non-enumerative procedure. A heuristic algorithm is presented here, that constructs a solution to the constrained encoding problem. 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.

We introduce first some definitions. Let n_s be the number of states to encode, n_i the number of groups and n_b the code length. The **constraint matrix** A is a matrix: $A \in \{0,1\}^{n_i \times n_s}$ representing n_i state groups. State j belongs to group i if $a_{ij} = 1$. Note that the constraint matrix corresponds to the present-state field of a minimal multiple-valued cover of the FSM combinational component, when positional-cube notation is used [SU72] [DEMI84b]. A row of the constraint matrix is said to be **prime** if it does not represent the intersection of two or more groups. The prime rows of a constraint matrix A, denoted by A_P, have the following property: any encoding, that is a solution to the constrained encoding problem specified by A_P, is a solution to the original problem specified by A [DEMI84b].

The **encoding matrix** S is a matrix $S \in \{0,1\}^{n_s \times n_b}$ whose rows are the encodings. Our problem is to determine the encoding matrix S, given a constraint matrix A. An encoding matrix S is said to **satisfy the constraint relation** for a given A if S is a solution to the constrained encoding problem specified by A.

The encoding algorithm constructs an encoding matrix S row by row and column by column by an iterative procedure. At each step a larger set of states is considered and an encoding matrix S is computed that satisfies the constraint relation for the corresponding columns of A. For each state that is being considered, a new row, σ, is appended to S . The encoding matrix S is initialized to a 1-column matrix, and columns are appended to S (i.e. the code-length n_b is increased) only when needed to satisfy the constraint relation. The input to the algorithm is the constraint matrix A. The output is the state code matrix S having n_b columns. The selected state (or state subset) to be encoded at the current iteration of the algorithm is denoted by $\mathscr{P}$. The set of encoded and selected states is denoted by $\mathscr{E}$. The algorithm is described in Pidgin C.

ENCODING ALGORITHM

```
S = φ;
𝓔 = φ;
A = compress(A);
do {
        𝒫 = state-select;
        𝓔 = 𝓔 ∪ 𝒫;
        A' = columns of A corresponding to 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, i.e. returns the minimal number of rows specifying completely the encoding problem.

Procedure **state-select** sorts the states according to a heuristic strategy, and returns the current state (or state subset) $\mathscr{P}$ 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 $\mathscr{P}$ 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.

Procedure **code-select**($\mathscr{C}$) 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 c is the following. The group faces corresponding to $\mathscr{E}$ depend on c. 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 $\mathscr{C}$ 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 $T = \{0\}^{n_s \times 1}$, 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 $R = A^{[T]}$, where $A^{[}$ is the subset of rows of A having a non-zero entry in columns of A corresponding to $\mathscr{P}$ and the superscript T denotes the transpose operator.

```
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. We proved in [DEMI84b] that the candidate set $\mathscr{C}$ is not empty after a finite number of iterations through **adjoin**(S). Procedure **adjoin**(S) appends columns to S in a particular sequence because of the following reasons. When **adjoin**(S) appends T to S, the size of the faces not related to state (states) $\mathscr{P}$ is not increased. Moreover a state code σ is found after one iteration through procedure **adjoin**(S), under some restrictive assumptions [DEMI84b]. Adjoining to S the columns of R one at a time corresponds to reshaping the faces related to $\mathscr{P}$, i.e. the faces corresponding to the rows in A having a non-zero entry in a column corresponding to $\mathscr{P}$. Reshaping each one of this faces consists of adding one dimension to the state code space: the new coordinate of the state codes on that face is set to "1", while is set to "0" for the remaining state codes. Reshaping is performed considering one face at a time, and by considering first the faces involving fewest states. Since, in general, states are related to many faces, reshaping a face leads to a size increase of some other face. Therefore this heuristic strategy tries to minimally increase the face sizes.

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. 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.

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 an appropriate state subset, called **dominating set** [DEMI84b], at the first iteration of the algorithm. A dominating set has the following property: if an encoding for a dominating set satisfies the constraint relation for the appropriate columns of the constraint matrix, then remaining states can be encoded one at a time by increasing at most by one the code length n_b at each iteration of the algorithm [DEMI84b]. 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_P) is selected first. The strategy tries not to increase the state space dimension. The uncoded state with highest column count in A_P 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_P is the "local most critical state to code" and is encoded first. We proved in [DEMI84b] that 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_P. Then states are ordered as follows. Let $A(\mathscr{E})$ be the subset of the columns of A_P corresponding to those states belonging to some group including an encoded state. 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 selection strategy is followed. However experimental results have shown only slightly longer encodings than those obtained with the previous strategy.

4. 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 [DEMI84a] 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 multiple-valued-input logic minimization [BRAY84b]. 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 [DEMI84b]. For this reasons ESPRESSO-II has been linked to KISS. The minimized representation defines the constraints of the encoding (i.e. the state groups) 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.

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 1 along with the execution times in seconds on a VAX-UNIX [1] computer. Table 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 3 compares the area estimates of the segment of the PLA depending on the state representation.

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 (30%) 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.

5. 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 by multiple-valued-input minimization, that yields a minimal sum-of-product representation of the next-state transition functions, independently of the state assignment. Multiple-valued-input minimization is achieved using state-of-the-art logic minimizers and specifies the constraints of an encoding problem. The state assignment is a solution to the constrained encoding problem and is constructed by a heuristic algorithm. Program KISS, which implements our strategy, gives results which are superior to previous methods for automatic state assignment.

6. 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. This work has been sponsored by NSF under contract No. ECS-8121446

REFERENCES

[BRAY84a] R.Brayton,G.D.Hachtel,C.McMullen and A.L.Sangiovanni- Vincentelli, "ESPRESSO-II: A New Logic Minimizer for Programmable Logic Arrays", Proc. Cust. Integr. Circ. Conf., Rochester, NY, pp. 370-376, may 1984.

[BRAY84b] R.Brayton,G.D.Hachtel,C.McMullen and A.L.Sangiovanni- Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

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

[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.

[DEMI84a] 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).

[DEMI84b] G.De Micheli, R.Brayton and A.L.Sangiovanni Vincentelli, "Optimal State Assignment for Finite State Machines", IBM Research Report RC 10599 and submitted for publication.

[DEMI84d] G.De Micheli "Optimal Encoding of Control Logic", Int. Conf. on Circ. and Comp. Des., Rye, NY, sep 1984.

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

[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.

[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.

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	5	7	5	55	21	5	31
FSM 3	5	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	25

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	16	2	16	4	24	2
FSM 4	29	9	80	22	65	27	87	5
FSM 5	16	4	25	5	16	5	25	3
FSM 6	3	3	9	3	10	7	8	3
FSM 7	17	5	26	14	22	15	23	4

	KISS	*SAP**	1-hot	minimal n_b
FSM 1	27	27	65	39
FSM 2	105	199	169	132
FSM 3	48	38	64	48
FSM 4	261	1780	1485	435
FSM 5	64	125	144	78
FSM 6	24	27	70	24
FSM 7	65	312	336	92

n_i = number of inputs — c_1 = symbolic cover cardinality
n_s = number of states — c_2 = minimal Boolean cover cardinality
n_o = number of outputs — n_b = number of bits

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