Considering State Minimization during State Assignment[1]

Ney Laert Vilar CALAZANS

Instituto de Informática - PUCRS
Av. Ipiranga, 6681 - Prédio 16 - Porto Alegre - RS - CEP 90619-900 BRAZIL
Phone: +(55) (0) 51 3391511 Ext: 3211 Fax: +(55) (0) 51 3391564
e-mail address: calazans@music.pucrs.br

Abstract

The state minimization and the state encoding of finite state machines are complex problems arising quite frequently in VLSI *design. These problems have been treated separately for a long time. In this paper, we provide a new algorithm to perform state encoding while considering state minimization, for two-level logic implementations. The algorithm is based on a newly developed, generalized formal treatment of the encoding problem. It has been implemented as a prototype computer program and compared with approaches to solve the two problems separately. The benchmark results show that our prototype has a performance comparable to these well-established strategies in terms of area estimation and transistor count.*

1 Introduction

Encoding information as vectors of binary digits is a fundamental step of numerous problems encountered in Computer Science in general, and in computer design and VLSI design problems in particular. The *optimal* solution of any encoding problem depends on the satisfaction of a set of *constraints* as well as on objective *optimization criteria*, all of which must be defined in terms of the original problem statement.

Encoding is basically a translation process, where a set of symbols is associated to a set of Boolean vectors. Several of the general approaches to encoding in VLSI appeared as a by-product of solutions to the state encoding problem for finite state machines (FSMs), also known as state assignment (SA). Most solutions to the state assignment of FSMs assume encodings that are injective functions from the state set into a set of Boolean vectors of a given fixed length [10, 11]. Although the use of injective functions be justified for the state assignment problem alone [4], it poses severe limitations if more powerful encoding strategies are required. For example, suppose that the set of symbols to be encoded has a structure that allows the identification of equivalence classes in it. To capture this characteristic, we must allow encodings that are not injective, so that every symbol in an equivalence class can be mapped into a unique Boolean vector. A more complicated case arises when the set of symbols allows the definition of compatibility classes onto it. Here, the encodings must be allowed to be both non-injective and non-functional, such that the intersection of overlapping classes is related to more than one Boolean vector. Since equivalence and compatibility classes are so commonly found in VLSI design problems, it is useful to consider them in the scope of encoding problems. One problem where compatibility classes identification is fundamental is the state minimization (SM) of incompletely specified FSMs [5].

We provide here just an informal approach to encoding problems, aiming at the understanding of the proposed algorithm. For a thorough formal treatment of the subject, the reader should refer to [2, 3]. In Section 2, we provide a brief discussion on the SM and the SA problems. The algorithm we propose appears detailed in Section 3. The implementation and the benchmark results obtained with it are discussed in Section 4. Finally, Section 5 lists a set of conclusions, and suggests further work on the subject.

[1]This work is partially supported by the CNPq, under research grant number 520523/94-6.

2 State Minimization and State Assignment

Given an FSM, SM is the problem of finding another FSM that has the same input/output behavior of the original machine and has the least possible number of states. On the other hand, SA is the problem of finding a function that to each state of the FSM associates a *code*, so that some optimization criterion is satisfied. The satisfaction of this criterion is dependent on the way the FSM will be implemented. For instance, if the criterion is smallest silicon area, satisfying it for a two-level logic implementation is distinct from satisfying it for multilevel logic implementation. In this work, we assume a two-level logic implementation.

Traditionally, SM and SA are separate steps of sequential logic synthesis, but using such a ***serial strategy*** may prevent the obtainment of optimal state assignments [8]. In this work, we address the problem of assigning codes to states of an FSM such that state minimization is taken into account *during* the encoding process, in what we call a ***simultaneous strategy***. This is possible thanks to the fact that both problems can be expressed by means of a set of ***constraints***. The constraint classes involved in the definition of each of the problems have been related in [2], and a unified representation framework has been proposed to gather them in the definition of a new encoding problem. The solution of this problem is obtained by the ***satisfaction*** of the constraints inside the framework.

All constraints in SM and SA express the need to separate (or not to separate) the code of two states. In SM, if two states are compatible, their codes need not be disjoint. If two states are incompatible, their codes *must* be disjoint. In SA, if a set of states lead to the same output under a same unique input, making their codes lead to a Boolean cube[2] that contain all of them and no other code of states that do not obey this condition, generates encodings with reduced logic implementations [4]. SA constraints for two-level implementations can be obtained by a technique called symbolic minimization [4]. An interesting result arising from symbolic minimization is that it generates an upper bound for the area of a two-level implementation of the combinational part of the initial FSM. Other constraint classes are discussed in [2].

We represent the above constraint classes using a two-block partition-like notation. For instance, suppose that states s_i and s_j are incompatible. This corresponds to a constraint represented by a pair $[\{s_i\}, \{s_j\}]$. The first element of the pair is called ***0-side***, while the second element is the ***1-side***. The pair means that there must be one bit position in the codes of s_i and s_j where either there is a 0 assigned to s_i and a 1 assigned to s_j, or there is a 1 assigned to s_i and a 0 assigned to s_j, to make the codes disjoint. We say that the first possibility satisfies the ***direct cube***, while the second satisfies the ***reverse cube***. Since this constraint is satisfied by respecting it in a single column of the encoding, it is called a ***local constraint***. If a constraint must be satisfied in *all* columns of an encoding, it is called a ***global constraint***. Example of global constraint is the compatibility between a pair of states. If we want them to be implemented as compatible, we must not allow any bit position where one of them is assigned 0 and the other is assigned 1. Note that this does not mean that these codes must be identical, since we may generate codes that are Boolean cubes. For example, $010-1-$ and $01-011$ are two compatible codes, where the character $-$ stands for a ***don't care***. Finally, constraints may conflict among themselves, but this issue is not addressed here.

To our knowledge, only three works have suggested the use of the simultaneous strategy to date. In the first of these, Hallbauer [7] proposes a method based on pseudo-dichotomies that avoid races in asynchronous circuits, and which tries to perform state minimization while heuristically reducing the encoding length. The second work is due to Lee and Perkowski [9], and suggests one exact method to tackle synchronous FSMs. Their method employs a branch-and-bound technique to reduce the search in the solution space. In a third work, Avedillo [1]

[2] A *Boolean cube* is a function representable by a *disjunction* or *product* of literals, where a *literal* is either a Boolean variable or its complement.

presents a heuristic method in which the encoding is generated incrementally, and which may create incompletely specified codes for the states in the original description. The subsequent combinational logic minimization step can, in this way, merge compatible states so that the equivalent of state minimization is performed.

No theoretical findings on the relationship between the SM and SA problems is provided in any of these works. Besides, the method of Hallbauer has not been submitted to benchmark tests, its efficiency being thus hard to evaluate. The second method is feasible only for very small machines (no more than sixteen states). The Avedillo's method has been extensively tested using a subset of the MCNC FSM benchmark set, comprising machines with no more than 32 states. Although reasonably efficient, the results of this method are poorer than those obtained with a serial strategy proposed in the same work.

3 The Asstuce Algorithm

The algorithm is a generalization of a procedure suggested in [11]. Consider a set of states $S = \{s_0, s_1, s_2, s_3, s_4\}$ and a set of local constraints defined over this set as follows:

$$C_l \;=\; \{c_0 = [\emptyset, \{s_1\}], c_1 = [\{s_1\}, \{s_3\}], c_2 = [\{s_4\}, \{s_1, s_2, s_3\}], c_3 = [\{s_3, s_4\}, \{s_0, s_2\}]\}.$$

Each constraint is satisfied by separating the codes of states placed on distinct sides. To simplify the discussion, the set of global constraints is empty. While treating the example, we shall ignore this fact. There are two other particularities in C_l. First, there is a constraint that has an empty 0-side. Such a constraint is always trivial in the context of the SM and SA problems, but have applications elsewhere [11]. The constraints in C_l will be represented using a matrix formulation. The ***constraint matrix*** P contains one row for each element of S, and one column for each constraint in C_l. The components p_{ij} are taken from the three-valued set $\{0, 1, -\}$, each position i of a column j of P contains a 1 if the element s_i is in the 1-side of a constraint ∂_j, contains a 0 if s_i is in the 0-side of ∂_j, and contains $-$ otherwise. Matrix P for our example is displayed below, together with an *encoding vector* ξ.

$$\begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{cccc} p_0 & p_1 & p_2 & p_3 \end{array} \\ \left(\begin{array}{cccc} - & - & - & 1 \\ 1 & 0 & 1 & - \\ - & - & 1 & 1 \\ - & 1 & 1 & 0 \\ - & - & 0 & 0 \end{array}\right) \end{array} \qquad \begin{array}{c} \xi \\ \left(\begin{array}{c} - \\ - \\ - \\ - \\ - \end{array}\right) \end{array}$$

The matrix P representation clarify the role of the constraints, the need to separate the codes of two subsets of states. For example, the constraint c_3 generates column p_3, which requires the separation of codes for the states $\{s_0, s_1\}$ from the codes for states $\{s_3, s_4\}$. The matrix is already a solution of the encoding problem. The optimization problem is to find another matrix that respects all constraints and that has the minimum number of columns. Our algorithm is a greedy approach to solve this covering problem through the generation of an encoding column at a time. Each column is generated to satisfy as many constraints as possible. After satisfying constraints, their columns can be dropped from P and the remaining columns become a new matrix P. We depict now one typical iteration of the algorithm.

3.1 The First Iteration

The ***encoding vector*** ξ is a column vector with length equal to the cardinality of S. It represents a single column of the final encoding Ξ. The ASSTUCE method accepts as input the initial

configuration of this vector. If none is given, it generates the *all don't cares* vector above. This is in fact the least restraining configuration that may be specified, since our approach is to produce one column encoding at a time, through the execution of a series of "moves" on the ξ vector. Each ***move*** is a change of the value of a component of ξ. The allowed changes depend on the initial value of ξ, but at any given moment, no change can make the vector contain more don't cares than in any previous step. Stated otherwise, no change from 1 or 0 to $-$ is allowed. Then, the all don't cares version of ξ is the less restraining initial configuration, a consequence of the choice of a greedy algorithm that generates column encodings without backtracking.

The First Move - From matrix P and vector ξ we produce an ***evaluation matrix*** E, which contains information about how far we are from satisfying the constraints in P using vector ξ. E is a matrix of pairs with the same dimensions as P, and the coordinate values of each pair are taken from the same three-valued set used to construct P and ξ. The rules to construct the components e_{ij} of E, given p_{ij} of P and ξ_i of ξ, appear in Table 1.

Table 1: Rules for defining the elements e_{ij} of the evaluation matrix E

$\xi_i \backslash p_{ij}$	0	1	-
0	(1,-)	(-,1)	(-,-)
1	(-,0)	(0,-)	(-,-)
-	(1,0)	(0,1)	(-,-)

Remember that a constraint in C_l is satisfied if the direct or the reverse cubes associated to it evaluate to 1. In what follows, ***to satisfy a cube*** will mean to make it to evaluate to 1. In a pair e_{ij}, the first coordinate refers to the reverse cube $\overline{c_j}$ associated to the constraint represented in column j, while the second coordinate refers to the direct cube c_j. A value 0 in any coordinate of a pair e_{ij} means that a change of the current value of the component ξ_i to 0 will make the encoding vector ξ closer to satisfy the corresponding (reverse or direct) cube. A value 1 means that a change of the current value of the component ξ_i to 1 will make the encoding vectorξ closer to satisfy the corresponding (reverse or direct) cube, and a value $-$ in any coordinate tells that it is not possible to get closer to satisfy the cube by changing e_{ij}, because it is already satisfied in position i by the current value ξ_i.

We may now compute the number of moves needed to satisfy each constraint. Since there are two ways to satisfy each constraint, there are two such numbers, and they are gathered in a vector of pairs of integers, which we call the ***distance vector*** ν. The components ν_j of ν are computed by simply counting the number of non-don't care values in a given column, and this for each coordinate. The number of presently unsatisfied constraints, u, is the number of components of ν where no coordinate is 0. Matrix E, vector ν and u, which are generated by considering P and ξ above are:

$$P^{(1)} = \begin{pmatrix} - & - & - & 1 \\ 1 & 0 & 1 & - \\ - & - & 1 & 1 \\ - & 1 & 1 & 0 \\ - & - & 0 & 0 \end{pmatrix} \quad \xi^{(1)} = \begin{pmatrix} - \\ - \\ - \\ - \\ - \end{pmatrix} \quad E^{(1)} = \begin{pmatrix} (-,-) & (-,-) & (-,-) & (0,1) \\ (0,1) & (1,0) & (0,1) & (-,-) \\ (-,-) & (-,-) & (0,1) & (0,1) \\ (-,-) & (0,1) & (0,1) & (1,0) \\ (-,-) & (-,-) & (1,0) & (1,0) \end{pmatrix}$$

$$u^{(1)} = 4 \qquad \nu^{(1)} = \begin{pmatrix} (1,1) & (2,2) & (4,4) & (4,4) \end{pmatrix}.$$

Since $u \neq 0$, we have to choose a component of vector ξ to change, in order to get the as close as possible to a column encoding that maximizes the number of satisfied constraints. To do

so, we define a ***direction matrix*** D, which carries information about in which direction a single change of a component in vector ξ can satisfy a constraint from C_l. D is a pair matrix, just like E, but with a distinct interpretation for its values, and a distinct generation method. The first and second coordinates of a pair d_{ij} in D correspond to changes to 0 and to 1, respectively. The values of the d_{ij} pairs are thus interpreted as follows:

$$\begin{cases} (1,1), & \text{if a change of } \xi_i \text{ to either 1 or 0 satisfies one of } \overline{c_j},\ c_j; \\ (0,-), & \text{if a change of } \xi_i \text{ to 0 unsatisfies one of } \overline{c_j},\ c_j; \\ (-,0), & \text{if a change of } \xi_i \text{ to 1 unsatisfies one of } \overline{c_j},\ c_j; \\ (1,-), & \text{if a change of } \xi_i \text{ to 0 satisfies one of } \overline{c_j},\ c_j; \\ (-,1), & \text{if a change of } \xi_i \text{ to 1 satisfies one of } \overline{c_j},\ c_j; \\ (-,-), & \text{otherwise.} \end{cases}$$

The configurations not shown either never happen or represent trivial cases. The first configuration arises only in a trivial case (from which the first constraint in P is an example) and in a special case. In the list above, *unsatisfies* means *turns a satisfied cube into an unsatisfied one.*

Matrix D can be obtained from an inspection of the distance vector ν and the evaluation matrix E. There are various possible situations, but only five non-trivial ones. If we have $\nu_j = (0, \geq 2)$ (resp. $\nu_j = (\geq 2, 0)$), the cube $\overline{c_j}$ (resp. c_j) is already satisfied, and any change of a component ξ_i such that $p_{ij} \neq -$ can only increase u, the number of unsatisfied constraints. If, on the other hand, we have $\nu_j = (1, \geq 2)$ (resp. $\nu_j = (\geq 2, 1)$), there is a component ξ_i that, if changed, will satisfy the cube $\overline{c_j}$ (resp. c_j). Finally, there is the special case where $\nu_j = (1, 1)$ and the direct and reverse cube associated to column j have only two non-don't cares in their three-valued representation. Then, there are exactly two possible single changes such that one can lead to the satisfaction of $\overline{c_j}$ and the other to the satisfaction of c_j. Obviously, $\nu_j = (\geq 2, \geq 2)$ implies that no single change of one component of ξ can satisfy any direct or reverse cube.

To accumulate the values of matrix D and determine the best move, we use a ***gain vector*** ω. Vector ω is a column vector of pairs of integers. Its contents are obtained as a componentwise sum of pairs over all columns of matrix D, for each D row. The sum is done as follows: if a coordinate of a pair d_{ij} is 1, add 1 to the same coordinate of ω_i; if a coordinate of a pair d_{ij} is 0, add -1 to the same coordinate of ω_i; if a coordinate of a pair d_{ij} is -, add 0 to the same coordinate of ω_i.

The direction matrix D and the gain vector ω resulting for our example are:

$$\begin{array}{cc} D^{(1)} & \omega^{(1)} \\ \begin{pmatrix} (-,-) & (-,-) & (-,-) & (-,-) \\ (1,1) & (-,-) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \end{pmatrix} & \begin{pmatrix} (0,0) \\ (1,1) \\ (0,0) \\ (0,0) \\ (0,0) \end{pmatrix} \end{array} \quad \text{select } \xi_1 \to 0$$

The first coordinate of a component ω_i of ω gives the gain of changing ξ_i to 0, while the second coordinate gives the gain of a change to 1. The best choice is the one with the greatest positive gain. In our example, there are two such best choices. Either we select a change of ξ_1 to 0 or to 1. Suppose that one selection is made. It is at this point that we must verify if the change does not violate any of the global constraints. Suppose that a violation occurs. Then, the change is discarded and a second choice is tried, and so on. If no ***feasible*** change exists, none is performed, and the encoding column generated up to now is a column of the final encoding. Our example has an empty set of global constraints, and thus such conflicts will not occur here. The first choice is then arbitrarily taken, changing ξ. This is what we call the ***first move***, indicated by the exponent (1) on the denomination of the structures.

The Second Move - After the first move, the component ξ_1 is ***locked*** (indicated by the symbol $\times$ in the corresponding position of the gain vector ω), and cannot change anymore during this column generation. The computations for the next move are summarized below.

$$\overset{\xi^{(2)}}{\begin{pmatrix} - \\ 0 \\ - \\ - \\ - \end{pmatrix}} \quad E^{(2)} = \begin{pmatrix} (-,-) & (-,-) & (-,-) & (0,1) \\ (-,1) & (1,-) & (-,1) & (-,-) \\ (-,-) & (-,-) & (0,1) & (0,1) \\ (-,-) & (0,1) & (0,1) & (1,0) \\ (-,-) & (-,-) & (1,0) & (1,0) \end{pmatrix}$$

$$\nu^{(2)} = \begin{pmatrix} (0,1) & (2,1) & (3,4) & (4,4) \end{pmatrix} \qquad u^{(2)} = 3$$

$$D^{(2)} = \begin{pmatrix} (-,-) & (-,-) & (-,-) & (-,-) \\ (-,1) & (-,-) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \\ (-,-) & (-,1) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \end{pmatrix} \overset{\omega^{(2)}}{\begin{pmatrix} (0,0) \\ \times \\ (0,0) \\ (0,1) \\ (0,0) \end{pmatrix}} \quad \text{select } \xi_3 \to 1$$

The End of the Iteration - This process goes on until no more moves are possible. The impossibility of making moves arises either when all positions of ξ are locked, or when all possible moves violate a global constraint, or when only negative gains exist in ω for every non-locked position in ξ that does not violate a global constraint, indicating that any allowed change will decrease the number of satisfied constraints.

One heuristic choice that can be used is to make moves whenever one is possible, until all positions of ξ are locked, retaining the configuration of ξ that satisfied most constraints, and not necessarily the last one.

Suppose that after the second move, we make a third move, by selecting $\xi_2 \to 0$ and a fourth move selecting $\xi_4 \to 1$. There is still room for another move in the example, which is:

$$\overset{\xi^{(5)}}{\begin{pmatrix} - \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}} \quad E^{(5)} = \begin{pmatrix} (-,-) & (-,-) & (-,-) & (0,1) \\ (-,1) & (1,-) & (-,1) & (-,-) \\ (-,-) & (-,-) & (-,1) & (-,1) \\ (-,-) & (0,-) & (0,-) & (-,0) \\ (-,-) & (-,-) & (-,0) & (-,0) \end{pmatrix}$$

$$\nu^{(5)} = \begin{pmatrix} (0,1) & (2,0) & (1,4) & (1,4) \end{pmatrix} \qquad u^{(5)} = 2$$

$$D^{(5)} = \begin{pmatrix} (-,-) & (-,-) & (-,-) & (1,-) \\ (-,1) & (-,0) & (-,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \\ (-,-) & (0,-) & (1,-) & (-,-) \\ (-,-) & (-,-) & (-,-) & (-,-) \end{pmatrix} \overset{\omega^{(5)}}{\begin{pmatrix} (1,0) \\ \times \\ \times \\ \times \\ \times \end{pmatrix}} \quad \begin{matrix} \text{select } \xi_0 \to 0 \\ \text{and stop.} \end{matrix}$$

3.2 The Subsequent Iterations

After the fifth move, all positions in ξ are locked, a first column encoding has been generated, namely 00011^T, which satisfies three of the four constraints describing the problem. A new iteration is needed, since there are unsatisfied constraints left. All satisfied constraints are eliminated from matrix P, transforming it into a smaller constraint matrix, which is the input of the new iteration, together with the specified vector ξ. Note that the vector ξ need not be the

same at each step. This is useful if, for example, some symbol codes are to be preestablished. Assuming the initial ξ vector to be the same as before, the start of the new iteration appears below.

$$\begin{array}{ccc} \xi^{(1)} & & \\ \begin{pmatrix} - \\ - \\ - \\ - \\ - \end{pmatrix} & E^{(1)} = & \begin{pmatrix} (-,-) \\ (0,1) \\ (0,1) \\ (0,1) \\ (1,0) \end{pmatrix} \\ & \nu^{(1)} = & \begin{pmatrix} (4,4) \end{pmatrix} \quad u^{(1)} = 1 \end{array}$$

The associated direction matrix D would be an all don't cares column vector, and the weight vector would have only pairs of the type $(0,0)$. A danger arises in the situation where the maximum gains are 0, given an all don't cares initial encoding vector. Since no direction was given about how to choose one of the **to 0** or **to 1** moves when they have identical weights, the method may pick one arbitrarily. In the example, we have to make at least three moves before obtaining a matrix D that contains anything except don't cares. Suppose that the method picks moves arbitrarily, but deterministically, and that the moves $\xi_2 \rightarrow 1$ and $\xi_3 \rightarrow 0$ are the first and second ones. Then, there will be no further ways of satisfying the constraint. The encoding column would be added to the encoding and the iteration would repeat the same procedure *ad infinitum*, without never finding a satisfying vector for the constraint. To avoid this, every time that the maximum gains in the ω vector are zero, we choose to satisfy always the direct cubes. This ensures that the column generation always stops. For our example, we would thus obtain a vector ξ equal to -1110^T, satisfying the last constraint, and the algorithm would stop.

The final encoding respecting all constraints would be obtained from the concatenation of the two encoding columns, which gives

symbol	*code*
s_0	$0-$
s_1	01
s_2	01
s_3	11
s_4	10.

Note that the encoding is neither functional, nor injective, what we wanted to be able to do, and that it solves the problem.

3.3 ASSTUCE Method Discussion

The above method is heuristic, with no backtracking. One advantage of its application is the execution speed. Indeed, our implementation iterates on encoding generation, not only on column encoding generation, without great expense in computation time, even for big examples. However, some observations should be pointed out about the algorithm. First, several choices have to be made during the execution, like which best move to take, in which direction, and what to do when only null gains are computed, which happens frequently in the initial steps of the iteration. Second, only locally optimal solutions can be obtained, even if iteration over a first solution is possible and applied. Third, the direct matrix computation may be substituted by sparse matrix techniques to accelerate execution, which is done in the implementation.

The computational cost of the column encoding generation problem is influenced mainly by four tasks: **Task 1**, the computation of the evaluation matrix E and of the distance vector ν; **Task 2**, the computation of the direction matrix D and of the gain vector ω; **Task 3**,

the selection of the component in ω with the maximum gain; **Task 4**, the verification of the feasibility of performing a move on the selected component.

Suppose that n is the cardinality of the set of symbols S to encode, and that m is the cardinality of the local part C_l. Suppose also that the number of operations to verify if a move does not violate a global constraint is constant. Suppose finally that we use the natural choices, namely one-dimensional vectors for ν, ω and two-dimensional arrays for E, D. The, a column encoding generation consists in an iteration repeated at most n times where E, ν, D, ω are built with complexity bounded by $\mathcal{O}(n.m)$, and a best component to change is selected. We have shown that selection of the best component takes a constant number of operations $\mathcal{O}(1)$ [2]. Each move has to be verified against at most $(n-1)$ positions in ξ, giving a complexity bounded by $\mathcal{O}(n)$. A move is made and the associated component of ξ is locked, using a constant number of operations ($\mathcal{O}(1)$). The final bound on the complexity of column encoding generation is then $\mathcal{O}(n^2.m)$.

The use of special data structures reduces this complexity, such that it is a function of the number c of non-don't care components in the initial constraint matrix P, instead of a function of $m.n$ [2]. Note that c is at most $m.n$, the total number of entries in matrix P. In practice c is quite smaller than that, specially for large examples.

4 Implementation and Benchmark Results

The ASSTUCE method has been implemented as a computer program and compared against a serial strategy where state minimization is performed using the program STAMINA [6] and state assignment is done with the program NOVA [12]. The FSM test set used is part of the MCNC benchmarks, but only machines with at least one pair of distinct compatible states were considered. Our prototype implementation at present do not consider output constraints, and the program NOVA was parameterized to avoid their consideration (i.e. with the run-time option **-e ih**), to allow a fair comparison.

All comparison parameters are extracted from the minimized two-level combinational part of the encoded FSM. The program NOVA, as well as ASSTUCE rely on the ESPRESSO program to perform the combinational part minimization after encoding. The same statement is true for the input constraints generation step. In this way, the comparisons reflect the differences arising from the encoding strategy alone, not from side effects arising from the use of distinct minimization schemes.

The data resulting from comparing ASSTUCE and the serial strategy based on the STAMINA and NOVA programs is depicted in Table 2.

ASSTUCE and the partial encoding serial strategy based on NOVA are comparable for most parameters, with the serial strategy obtaining slightly better area results and ASSTUCE obtaining slightly sparser machines but with reduced number of transistors in it, and less product terms. The consequences of these differences is that we judge the ASSTUCE results more adapted to consider power dissipation issues in big PLAs, because of the combined effect of smaller areas corresponding to sparser PLAs. Besides, we know that sparser PLAs favor the use of topological optimization tools during the low level synthesis of the FSM [**?**].

The advantages related to ASSTUCE are a consequence of using non-functional, non-injective encodings. Cube merging is facilitated during the logic minimization step, and even if the encoding length is increased, the final result in some cases combine smaller areas with less dissipated power. However, the main issue here is to show that the formulation of the problem does not imply less efficient solutions for encoding problems, which validates the basic idea of searching for more powerful encoding methods.

Table 2: Comparison ASSTUCE × STAMINA/NOVA

FSM	a_area	n_area	sn_area	a_time	n_time	sn_time	a_pt	n_pt	sn_pt	a_cl_f	n_cl_f	sn_cl_f	a_tr	n_tr	sn_tr	a_spty	n_spty	sn_spty
s27	198	234	216	0.28	0.10	0.10	11	13	12	3	3	3	43	62	51	78.28	73.5	76.39
beecount	144	247	160	0.23	0.10	0.10	9	13	10	2	3	2	50	68	54	65.28	72.47	66.25
lion9	102	136	77	0.43	0.30	0.10	6	8	7	4	4	2	24	35	24	76.47	74.26	68.83
ex5	120	252	96	0.25	0.50	0.00	8	14	8	3	4	2	34	88	34	71.67	65.08	64.58
ex7	120	306	96	0.27	0.30	0.10	8	17	8	3	4	2	38	107	36	68.33	65.03	62.5
ex3	144	324	96	0.18	0.20	0.10	8	18	8	4	4	2	33	103	35	77.08	68.21	63.54
bbara	380	550	380	0.78	0.20	0.20	20	25	20	3	4	3	94	129	94	75.26	76.55	75.26
opus	504	448	448	1.05	0.20	0.10	18	16	16	4	4	4	139	128	123	72.42	71.43	72.54
train11	85	153	66	0.37	0.60	0.10	5	9	6	4	4	2	26	47	24	69.41	69.28	63.64
mark1	697	684	646	0.93	5.10	4.30	17	18	17	5	4	4	145	115	117	79.20	83.19	81.73
sse	972	990	1023	1.67	0.50	1.10	27	30	31	5	4	4	196	191	206	79.84	80.71	79.86
bbsse	972	990	1023	1.62	0.40	1.20	27	30	31	5	4	4	196	191	206	79.84	80.71	79.86
ex2	594	609	195	1.28	0.50	1.00	18	29	13	9	5	3	91	171	66	84.68	71.92	66.15
tma	1230	1155	1295	5.97	6.60	13.50	30	33	37	7	5	5	241	230	257	80.41	80.09	80.15
ex1	2288	2496	2132	5.37	6.50	5.00	44	48	41	5	5	5	399	422	330	82.56	83.09	84.52
tbk	1680	4620	1431	103.22	140.7	24.5	56	154	53	5	5	4	614	1423	553	63.45	69.2	61.36
scf	17420	18471	16244	603.68	105.8	59.8	130	141	124	8	7	7	1541	1469	1383	91.15	92.04	91.49
s298	16632	22464	10332	10637.90	828.8	266.6	308	624	287	14	8	8	3824	6783	2586	77.01	69.81	74.96

Prefixes:
a_ : results obtained by running ASSTUCE
n_ : results obtained by running NOVA alone
sn_ : results obtained by running STAMINA followed by NOVA

Suffixes:
area : area estimate of the minimized combinational part for the encoded FSM
pt : number of product terms in the minimized combinational part of the encoded FSM
cl_f : encoding length
time : total execution time
tr : number of transistors in the minimized combinational part of the encoded FSM
spty : percentual sparsity of the minimized combinational part for the encoded FSM

5 Conclusions and Further Work

Symbols codes generated by our approach are by construction sparser than those obtained with traditional encoding methods, as a result of employing non-injective, non-functional encodings. The competitiveness of the results obtained so far with our prototype implementation indicates that we may achieve gains in power dissipation and communication complexity without compromising the area occupied by the circuit if more elaborate encoding schemes are developed.

We envisage the evolution of the present work in several directions. The first of these is to further validate the approach proposed here by obtaining more examples of encoding problems where the identification of equivalence and/or compatibility classes is fundamental to the search of optimal solutions. Second, we are presently doing research on the application of recently developed techniques for the manipulation of implicit representations of switching functions with the use of reduced ordered binary decision diagrams. We are considering the application of these techniques to the representation and satisfaction of our constraint framework. Finally, we are envisaging the application of the formal paradigm developed here to sequential logic synthesis problems for FPGAs.

References

[1] M. J. Avedillo, J. M. Quintana, and J. L. Huertas. SMAS: a program for concurrent state reduction and state assignment of finite state machines. In *Proceedings of the IEEE International Symposium on Circuits and Systems - ISCAS*, pages 1781–1784, Singapore, June 1991. The Institute of Electrical and Electronics Engineers.

[2] N. L. V. Calazans. *State minimization and state assignment of finite state machines: their relationship and their impact on the implementation.* PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, Oct. 1993.

[3] N. L. V. Calazans. Boolean constrained encoding: a new formulation and a case study. In *Proceedings of the IEEE International Conference on Computer-Aided Design - ICCAD*, San Jose, CA, Nov. 1994. The Institute of Electrical and Electronics Engineers.

[4] G. de Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design*, CAD-4(3):269–284, July 1985.

[5] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IRE Transactions on Electronic Computers*, EC-14:350–359, June 1965.

[6] G. D. Hachtel, J.-K. Rho, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. In *Proceedings of the European Conference on Design Automation - EDAC*, pages 184–191, Amsterdam, Feb. 1991.

[7] G. Hallbauer. Procedures of state reduction and assignment in one step in synthesis of asynchronous sequential circuits. In *Proceedings of the International IFAC Symposium on Discrete Systems*, pages 272–282, 1974.

[8] J. Hartmanis and R. E. Stearns. Some dangers in state reduction of sequential machines. *Information and Control*, 5:252–260, Sept. 1962.

[9] E. B. Lee and M. Perkowski. Concurrent minimization and state assignment of finite state machines. In *Proceedings of the 1984 International Conference on Systems Man and Cybernetics*, pages 248–260, Halifax, Oct. 1984.

[10] B. Lin and A. R. Newton. A generalized approach to the constrained cubical embedding problem. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors - ICCD*, pages 400–403. The Institute of Electrical and Electronics Engineers, Oct. 1989.

[11] C.-J. Shi and J. A. Brzozowski. Efficient constrained encoding for VLSI sequential logic synthesis. In *Proceedings of the European Design Automation Conference - EURO-DAC*, pages 266–271, Hamburg, Germany, Sept. 1992. IEEE Computer Society Press.

[12] T. Villa and A. Sangiovanni-Vincentelli. NOVA: state assignment of finite state machines for optimal two-level logic implementation. *IEEE Transactions on Computer-Aided Design*, 9(9):905–924, Sept. 1990.