State merging and state splitting via state assignment: a new FSM synthesis algorithm

M.J. Avedillo
J.M. Quintana
J.L. Huertas

Indexing terms: FSM synthesis, State assignment, State reduction, Design automation

Abstract: In the paper the authors describe a state assignment algorithm for FSMs which produces an assignment of non-necessarily distinct, and eventually, incompletely specified codes. In this new approach, state reduction and state assignment are dealt with concurrently, and a restricted state splitting technique is explored. The algorithm is particularly appropriate for machines with compatibility relations among its states because the potentials of state merging are exploited during the state assignment step. The input to SMAS, the program implementing the algorithm, is a symbolic cover of the FSM. The output is a Boolean representation of both next state and output functions suitable to minimise with ESPRESSO. The machines in the MCNC benchmark set are used to test the new algorithm and to compare it with a well known state assignment program.

1 Introduction

Computer-aided synthesis of sequential circuits is an active area of research as many modern CAD VLSI systems treat the control unit of the designed circuit as a finite state machine (FSM) realised with some regular structure. Two main problems encountered in the optimal synthesis of a sequential circuit are the minimisation of the number of the machine's internal states and their assignment. The optimal solution to each of these problems is very complex and, classically, they have received independent treatment. State reduction of completely specified FSM can be achieved in $O(n \log n)$ steps [1] whereas state minimisation of incompletely specified FSM is already NP-complete [2]. This means that most probably there will never be an algorithm of less than exponential complexity and therefore heuristic techniques must be used. This problem has received renewed attention in the past few years and very efficient approaches have been developed to cope with it [3–5]. In order to realise the behaviour of a sequential machine it is necessary to represent the states by binary sequences. The complexity of the required circuits often strongly depends on the binary encoding chosen. The problem of assigning a binary representation to the control states is referred to as state assignment or state encoding. There are many approaches for solving this problem, the heuristic ones being the most promising [6]. It is well known that state reduction and state assignment are intimately related. Reducing the number of states corresponds to decreasing the number of transitions of the sequencing functions and eventually to reducing the number of prime implicants in a minimal Boolean cover of a two-level logic realisation or reducing the number of literals in a multilevel one. Moreover, a reduction of the number of states may correspond to a reduction of the number of bits that are needed for the state encoding. Consequently, the first phase in a traditional design system has been to obtain a minimal FSM and the second one its optimal design.

More recently, it has been stated that the classical assumption of considering each problem in a separated way is no longer valid for modern technologies [7]. The recommendation [7] is not to seek an FSM with the minimum number of states but to replace the stages of state minimisation and state assignment with one stage of joint minimisation and state assignment. The previously mentioned work describes an approach based on partition theory in which the classical constraint of state assignment using partitions (the product partition is the zero partition) is removed. The method has important limitations like the treatment of incompletely specified machines. In Reference 8 a state assignment algorithm using the degree of freedom that two or more equivalent states can be given the same code is sketched. Also, the need for a more general transformation of the FSM aimed at obtaining an equivalent machine that is easier to build has been pointed out [9].

In this paper, a novel approach to the concurrent state minimisation and state assignment is described. A restricted state splitting technique is also explored. The need for state splitting for optimal state assignment was pointed out as early as the 1960s. In Reference 10 state splitting is introduced to increase the possibilities of finding good assignments. Reference 11 relates the existence of a type of sequentially redundant faults with the use of state assignment algorithms which do not take into account state splitting.

2 Preliminaries

2.1 Background

For the sake of completeness some basic concepts in FSM synthesis will be briefly reviewed in this Section [12, 13]. An FSM can be formally defined as a quintuple $M = (I, S, O, \delta, \lambda)$ where I is a finite set of inputs, S is a

© IEE, 1994

Paper 1228E (C2, E3), first received 26th July 1993 and in revised form 22nd February 1994

The authors are with the Departamento de Diseño Analógico, Centro Nacional de Microelectrónica, Edif CICA, Avd. Reina Mercedes s/n, 41012 — Sevilla, Spain and the Departamento de Electrónica y Electromagnetismo, Universidad de Sevilla

finite, non-empty set of states, O is a finite set of outputs, $\delta: I \times S \rightarrow S$ is the next state function, and $\lambda: I \times S \rightarrow O (\lambda: S \rightarrow O)$ is the output function for a Mealy (Moore) machine. Either of these functions may be incompletely specified (i.e. the value of the function need not be defined for all possible inputs). We can represent any deterministic sequential function with this model. The FSM can be separated into two components: a combinational circuit and a memory. The memory stores the (binary) representation of the FSM state throughout computation whereas the combinational circuit generates machine output and next state representation as a function of the inputs and the present state.

Two internal states, S_i and S_j, of a machine M are compatible if and only if, for every input sequence applicable to both S_i and S_j, the same output sequence will be produced whenever both outputs are specified regardless of whether the initial state is S_i or S_j. A compatibility class or compatible C_i is a set of states such that members of the set are pairwise compatible. If there is an input such that S_k is the next state of S_i and S_l is the next state of S_j and $(S_k, S_l) \neq (S_i, S_j)$ then we say that (S_k, S_l) is implied by (S_i, S_j). A set of compatibles covers all states of a state table if every state is contained in at least one compatible in the set. A set of compatibles is closed if for each compatibility class in it, $C_i = \{S_{i1}, S_{i2}, \ldots, S_{im}\}$ and any input i_k, the set $\{\delta(i_k, S_{i1}), \delta(i_k, S_{i2}), \ldots, \delta(i_k, S_{im})\}$ is contained in at least one compatible of the collection. A fundamental theorem states that, given an incompletely specified state table, another state table specifying the same external behaviour corresponds to each closed set of compatibility classes which covers all internal states of the given table [14].

2.1 Previous examples

In this Section two simple examples are given. Both examples will illustrate several useful considerations. The first example shows a way of assigning the machine's states which is equivalent to the assignment of the previously reduced machine. The second describes a way of assigning the states which is equivalent to the assignment of the machine after having split some states. The use of D type memory elements is assumed in both examples.

Example 1: Fig. 1*a* depicts the state table of an FSM. The closed covering and the reduced state table obtained by the application of a state reduction algorithm to the symbolic description in Fig. 1*a* are shown in Fig. 1*b*. Fig. 1*c* depicts the state transition table and the state encoding. This result can also be reached assigning the same code to states s_1 and s_2, and to the state s_4 and s_5 in the original machine. Fig. 2 shows the transition table. Note that there are input-present state combinations which belong both to the DC-set and ON-set (OFF-set) of one or several next state and/or output functions. There is no input-present state combination belonging simultaneously to the ON-set and to the OFF-set. Specifying those combinations simultaneously in the DC-set and ON-set (OFF-set) as part of the ON-set (OFF-set), the transition table in Fig. 2 becomes identical to the one in Fig. 1*c* that was obtained with the conventional two step synthesis approach: state reduction and state assignment.

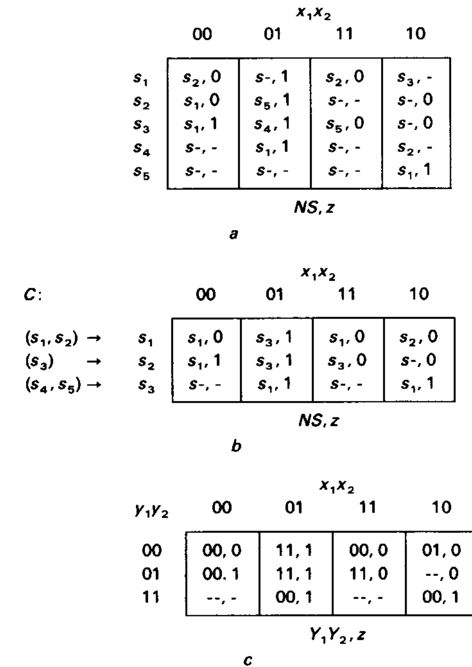
Example 2: The state table of another FSM is shown in Fig. 3*a*. An encoding Fig. 3*b*, the transition table Fig. 3*c* and the excitation and output functions requiring five product terms Fig. 3*d* are also shown.

We can split the state s_3 into two states called s_3' and s_3'' with

$$\delta(x, s_3') = \delta(x, s_3'') = \delta(x, s_3)$$

and

$$\lambda(x, s_3') = \lambda(x, s_3'') = \lambda(x, s_3)$$



a (x_1x_2; entries NS, z)

	00	01	11	10
s_1	s_2, 0	s-, 1	s_2, 0	s_3, -
s_2	s_1, 0	s_5, 1	s-, -	s-, 0
s_3	s_1, 1	s_4, 1	s_5, 0	s-, 0
s_4	s-, -	s_1, 1	s-, -	s_2, -
s_5	s-, -	s-, -	s-, -	s_1, 1

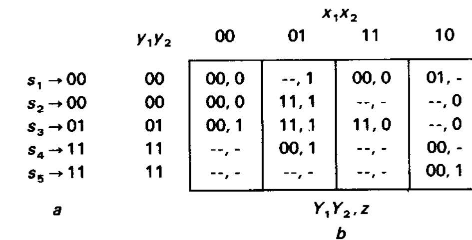
b (x_1x_2; entries NS, z)

C:		00	01	11	10
$(s_1, s_2) \rightarrow$	s_1	s_1, 0	s_3, 1	s_1, 0	s_2, 0
$(s_3) \rightarrow$	s_2	s_1, 1	s_3, 1	s_3, 0	s-, 0
$(s_4, s_5) \rightarrow$	s_3	s-, -	s_1, 1	s-, -	s_1, 1

c (x_1x_2; entries Y_1Y_2, z)

y_1y_2	00	01	11	10
00	00, 0	11, 1	00, 0	01, 0
01	00, 1	11, 1	11, 0	--, 0
11	--, -	00, 1	--, -	00, 1

Fig. 1 *FSM design with state reduction*
NS: next state, z: output
a state table
b closed covering C and reduced state table
c state assignment and transition table



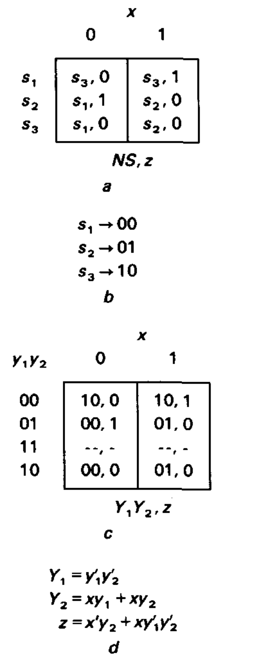
a state assignment / *b* transition table (x_1x_2; entries Y_1Y_2, z)

	y_1y_2	00	01	11	10
$s_1 \rightarrow 00$	00	00, 0	--, 1	00, 0	01, -
$s_2 \rightarrow 00$	00	00, 0	11, 1	--, -	--, 0
$s_3 \rightarrow 01$	01	00, 1	11, 1	11, 0	--, 0
$s_4 \rightarrow 11$	11	--, -	00, 1	--, -	00, -
$s_5 \rightarrow 11$	11	--, -	--, -	--, -	00, 1

Fig. 2 *FSM assignment with repeated codes*
a state assignment
b transition table

for each input value ($x = 0$ and $x = 1$). Fig. 4*a* shows the resulting state table. Both s_3' and s_3'' appear in those cells corresponding to input-present state combinations with s_3 as next state in the original description (Fig. 3*a*). This means that any of them can be used to describe the behaviour of the machine. The state transition for the encoding of Fig. 4*b* is depicted in Fig. 4*c*. Excitations and output functions (Fig. 4*d*) requires only four product terms. The logic minimiser takes advantage of the extra degree of freedom introduced by the state splitting. That is, choosing $\delta(0, 00) = 10$ and $\delta(1, 00) = 11$ produces a cheaper implementation.

The above examples illustrate the state reduction and state assignment can be dealt with concurrently and points out the convenience of considering state splitting in the context of optimal synthesis of FSMs.



	x = 0	x = 1
s_1	$s_3, 0$	$s_3, 1$
s_2	$s_1, 1$	$s_2, 0$
s_3	$s_1, 0$	$s_2, 0$

NS, z

a

$s_1 \to 00$
$s_2 \to 01$
$s_3 \to 10$

b

y_1y_2	x = 0	x = 1
00	10, 0	10, 1
01	00, 1	01, 0
11	--, -	--, -
10	00, 0	01, 0

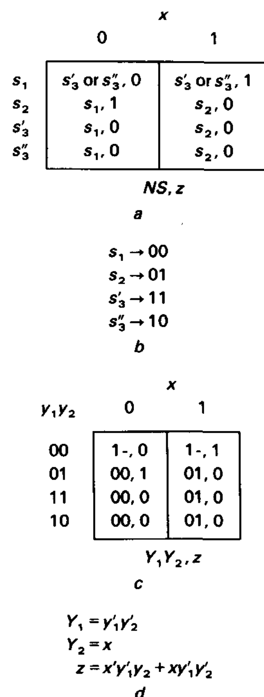
Y_1Y_2, z

c

$Y_1 = y_1'y_2'$
$Y_2 = xy_1 + xy_2$
$z = x'y_2 + xy_1'y_2'$

d

Fig. 3 *Example 2*
a state table
b state assignment
c transition table
d excitation and output functions



	x = 0	x = 1
s_1	s_3' or s_3'', 0	s_3' or s_3'', 1
s_2	$s_1, 1$	$s_2, 0$
s_3'	$s_1, 0$	$s_2, 0$
s_3''	$s_1, 0$	$s_2, 0$

NS, z

a

$s_1 \to 00$
$s_2 \to 01$
$s_3' \to 11$
$s_3'' \to 10$

b

y_1y_2	x = 0	x = 1
00	1-, 0	1-, 1
01	00, 1	01, 0
11	00, 0	01, 0
10	00, 0	01, 0

Y_1Y_2, z

c

$Y_1 = y_1'y_2'$
$Y_2 = x$
$z = x'y_1'y_2 + xy_1'y_2'$

d

Fig. 4
a split state table
b state assignment
c transition table
d excitation and output functions

3 Rationale of the new approach

Let us briefly show how the goals of state reduction and state splitting are achieved by a coding process which allows a single code to be assigned to a group of states and the use of incompletely specified codes.

Assigning a single code to a group of states is equivalent to the transformation of the FSM description by the substitution of a group of states with one single state (state reduction) and the assignment of this internal state in the new description. From classical state reduction theory we know that the states which are merged into one state must be compatible and closure constraints must be satisfied [12, 14]. These conditions are used as constraints in our encoding process. That is, the state reduction is achieved during the assignment process.

Assigning an incompletely specified code [13] (group of codes) to a single state is equivalent to the transformation of the FSM symbolic description consisting in the substitution of a single state by a group of states (state splitting) and the assignment of this new description. There are two reasons for allowing incompletely specified codes as follows.

(1) It is important to cope with the concurrent state reduction and state assignment of incompletely specified FSMs. In general, for these machines, a closed set of compatibles covering a state table (solution to the classic state reduction phase) is not disjoint. A non-disjoint gathering of states cannot be achieved via state assignment if incompletely specified codes are not allowed. Thus, concurrent state reduction and state assignment would fail to reach a part of the space of solutions, so precluding the possibility of finding some efficient ones.

(2) The Boolean cover of the machine which is supplied to the logic minimiser can have extra flexibility in this way, because some next state entries have been substituted by a group of states. The minimiser can take advantage of this by choosing each time the best state of the group to achieve minimisation. In Reference 15 it is reported that allowing 'don't care' bits in a state assignment often results in a significant reduction of the combinational component.

Nevertheless, this state splitting is restricted, because the codes assigned to the set of states resulting from the splitting of an original one are constrained to form a cube.

3.1 Basic definitions

In order to formalise the above ideas, several definitions are introduced.

Definition 1: Let a_i and b_i be two variables which can take the logic values: 0, 1, and unspecified ('–'). The intersection operator $\cap$ is defined according to following table.

$\cap$ (b_i \ a_i)	0	1	–
0	0	∅	0
1	∅	1	1
–	0	1	–

Definition 2: Let $C(S_1) = (y_{11} \cdots y_{1n})$ and $C(S_2) = (y_{21} \cdots y_{2n})$ be the codes of the states S_1 and S_2, respectively. Assume each y_{ij} can be logic 0, logic 1 or be unspecified. The intersection of the codes $C(S_1)$ and $C(S_2)$, $C(S_1) \cap C(S_2) = C_\cap = (c_{\cap 1} \cdots c_{\cap n})$, is defined as

$$c_{\cap j} = y_{1j} \cap y_{2j} \quad 1 \leqslant j \leqslant n$$

$$\text{if } \exists j \mid c_{\cap j} = \varnothing \quad \text{then } C_\cap = \varnothing$$

The meaning of $C_\cap = \varnothing$ is that the set of binary codes assigned to S_1 and the codes assigned to S_2 have no common elements.

Definition 3: Two states S_1 and S_2 have compatible assignments if the intersection of their codes, $C(S_1) \cap C(S_2)$, is not empty.

For example, assume the state S_1 has been given the code (001–) and the state S_2 the code (0010). S_1 and S_2 have compatible assignments because (001–) ∩ (0010) = (0010). Identical assignments is a particular case of compatible assignments when the codes of both states are completely specified.

Definition 4: Two states have discriminated assignments if the intersection of their codes is empty.

Definition 5: Given an FSM, an encoding of its states is valid if the encoded machine and the symbolic description are equivalent; that is, the derived logic implements the desired behaviour.

Our assignment process is equivalent to both the transformation of the symbolic description of the FSM and to the assignment of this new description. Some constraints need to be imposed on the encoding so that the behaviour of the machine is not changed by the state reduction achieved concurrently with the assignment. Translating the concepts of symbolic state reduction theory to the state assignment context and using Definitions 1 to 5, it can be stated that in order to guarantee that an encoding is valid the following constraints must be satisfied.

(1) A pair of states with compatible assignments must be compatible states or, a pair of incompatible states must be given discriminated assignments.

(2) For each input, the pairs of states which are implied by a pair of states with compatible assignments must also have compatible assignments or, the pairs of states which imply a pair of states with discriminated assignments must also be given discriminated assignments.

3.2 New state assignment problem formulation

Classically the optimal state assignment problem is formulated as

> 'Obtain a distinct single binary code for each state of the FSM so that given design constraints are satisfied'.

Typical design constraints are that the circuit must be constructed with a minimal amount of logic or that circuit can be built from an interconnection of smaller subcircuits to obtain superior performance, or that it can be easily testable. Most of the state assignment algorithms aim to minimise the area of an implementation of the combinational component assuming a predetermined design style (two level, multilevel).

However since a unique code, different from the codes of all other states, must be assigned to each state, the state assignment is only free to encode the states in such a way that a minimal area implementation of a given state transition graph (STG) is obtained. The problem we propose to solve is more general in the sense that a minimal area implementation of an FSM is sought. As many STGs can implement the same functionality, our approach enlarges the search space compared with classical techniques.

Transformations of the STG like state splitting and state merging are made concurrently with the binary encoding instead of in a separate synthesis step (state reduction) as in conventional design procedures. In this way the STG transformation is guided by more realistic criteria than the ones used in symbolic STG manipulations.

The problem we propose to solve can be stated as

> 'Obtain a valid assignment of the states of the FSM leading to a minimal area implementation of the machine'.

4 The algorithm

Given a state table describing the external behaviour of an FSM, a valid binary representation is built up by exploring both the potentials of state merging and state splitting. Given *ns* internal states in the symbolic description, we assume that *ns* × *nb* bits are going to be assigned to a value from the set {0, 1, –}, where *nb* is the number of bits of the generated codes, and so it is unknown until the algorithm has finished the assignment process.

Fig. 5 shows a Pidgin_C description of the algorithm.

```
SMAS (T)
{
    Discriminated_pairs = Ø;
    Identical_pairs = Ø;
    Implied_by_identical_pairs = Ø;
    Compulsory_discrimination_pairs = pairs of incompatible states;
    nb = nb_o = Initial_length();
    Sinit = s_1;
    while ( Compulsory_discrimination_pairs ! = {Ø} )
    }   (s_i, s_j) = Select_pair(Compulsory_discrimination_pairs);
        Discriminate_pair(s_i, s_j);
        Update_structures();
    }
}

Discriminates_pair(s_i, s_j)
{   Candidate_assignments = Explore_codes (s_i, s_j);
    if ( Candidate_assignments ! = {Ø} )
    {   for(each assignment ∈ Candidate_assignments)
        {   Evaluate_cost(assignment);
        }
        selects minimum cost assignment;
    }
    else
    {   Increase_code_length();
        Discriminate_pair(s_i, s_j);
    }
}
```

Fig. 5 *Algorithm's description (pseudocode)*

Initially the codes of all the states are completely unspecified, the algorithm assigns bits to 0 or 1 until the assignment is valid according to conditions 1 and 2 stated in Section 3.1. In order to describe how the algorithm works, we introduce the concept of pair of compulsory discrimination.

Definition 6: Two states form a pair of compulsory discrimination if they cannot have compatible assignments.

Incompatible states, and those pairs which imply a pair with discriminated assignments, are pairs of compulsory discrimination.

First we explain the meaning of the data structures. The list of state pairs with discriminated assignments is stored in *Discriminated_pairs* (DP); the list of state pairs with identical assignments in *Identical_pairs* (IP); the list of state pairs which are implied by at least another pair of states with identical assignments in *Implied_by_identical_pairs* (IIP) and the list of state pairs which must have discriminated assignments in *Compulsory_discrimination_pairs* (CDP).

The algorithm starts initialising the above lists. The lists DP, IP, IIP are empty. The list CDP is initialised to the set of pairs of incompatible states.

The initial length of the codes, nb_0, is determined. Since our state assignment algorithm allows state merging and state splitting, the length of the resulting encoding is not known beforehand. We start with the number of bits needed to encode as many states as there are in the maximal incompatible of highest cardinality. Afterwards, the length of the codes will be increased by one bit each time the intermediate assignment cannot be turned into a valid one with the current number of bits in the encoding. To finish the initialisation block, the first state in the symbolic description, S_1, is selected to have nb_0 bits assigned to 0.

The basic operation in the process is the discrimination of a compulsory discrimination pair (*Discriminate_pair*). This operation is repeated until the list CDP is empty. The pair (S_i, S_j) to be discriminated next is selected according to a heuristic criteria which will be explained later.

Procedure *Discriminate_pair* assigns a single bit of the code of the states S_i and/or S_j so that these two states have discriminated assignments. To achieve the discrimination, first all legal bit assignments leading to the discrimination of S_i and S_j are enumerated. Function *Explore_codes* stores them in *Candidate_assignments*.

A bit assignment is not legal if any of the following situations arises:

(*a*) there is at least a state pair (S_m, S_k), $m = i, j$; $k \neq m$, which now have identical assignments but it belongs to CDP;

(*b*) there is at least a state pair (S_m, S_k), $m = i, j$; $k \neq m$, which now have discriminated assignments but it belongs to IIP.

If either (*a*) or (*b*) occurs, the number of bits will have to be increased before a valid encoding is reached. It is important to point out that legal bit assignment does not mean that there is guarantee of the existence of a valid encoding with current code length. However, trying to discriminate a given pair using legal bit assignments helps in keeping the code length short and has proven to give good results. The lists IP and IIP are used to easy and efficiently detect that a given bit assignment is not legal.

Next, a cost function for each member of *Candidate_assignments* is evaluated and the one which minimises it is selected. Finally, if *Candidate_assignments* is empty, the function *Increase_code_length* adds one new state variable. Since the code of every state has the new bit unspecified, there is no state pair with identical codes and so the lists *Identical_pairs* and *Implied_by_identical_pairs* are emptied.

Procedure *Update_structures* updates lists DP, IP, IIP and CDP. For example, to update CDP we remove from it those pairs which have been discriminated and add those implying other pairs just discriminated. Fig. 6 summarises the update procedure.

Update of data structures: ***Update_structures***

Let (s_i, s_j) be the discriminated pair.

→ on list *Discriminated_pairs:*

(1) Addition of (s_i, s_j).

(2) Addition of pairs (s_i, s_k) or (s_j, s_k), with $k \neq i$, $k \neq j$, whose assignments have been discriminated as a consequence of assigning a bit of s_i and/or s_j.

→ on list *Identical_pairs:*

(1) Addition of pairs (s_i, s_k) or (s_j, s_k), with $k \neq i$, $k \neq j$, whose assignments are identical as a consequence of assigning a bit of s_i and/or s_j.

→ on list *Implied_by_identical_pairs:*

(1) Addition of pairs implied by (s_i, s_k) or (s_j, s_k), with $k \neq i$, $k \neq j$, whose assignments are identical as a consequence of assigning a bit of s_i and/or s_j. The implication concept is applied recursively.

→ on list *Compulsory_discrimination_pairs:*

(1) Elimination of (s_i, s_j).

(2) Elimination of those pairs (s_i, s_k) or (s_j, s_k), with $k \neq i$, $k \neq j$, whose assignments have been discriminated as a consequence of assigning a bit of s_i and/or s_j.

(3) Addition of pairs (s_n, s_m) implying any pair (s_i, s_k) or (s_j, s_k), whose assignments have been discriminated as a consequence of assigning a bit of s_i and/or s_j. The implication concept is applied recursively.

Fig. 6 *Update of data structures after having discriminated the pair* (s_i, s_j)

Finally, we demonstrate that a state assignment derived with this algorithm is valid according to conditions stated in Section 3.1.

Theorem 1: Given an FSM, an encoding of its states derived with previous algorithm is a valid assignment.

Proof:

(*a*) Suppose it is not a valid assignment because there are pairs with compatible assignments which are no compatible states. This would mean an incompatible pair has not been discriminated yet and so the list of pairs of compulsory discrimination would not be empty as it is the condition for the process to finish.

(*b*) Suppose it is not a legal assignment because there are pairs which have compatible codes but they imply pairs with discriminated assignments. This is not possible because when discriminating those implied pairs, the implying pairs would have been added to the list of pairs of compulsory discrimination (see Fig. 6).

4.1 Considerations about order strategies and cost function

It is worth noting that the straight application of our algorithm does not guarantee that the length of the codes is a minimum. However, several heuristics have been developed in order to achieve efficient assignments. For example, the order in which pairs of states are discriminated and the cost function are critical. Very efficient order strategies have been introduced so that codes longer than minimum ones are not usually generated. Moreover, the

potential of state merging leads, in some cases, to assignments with a number of state variables which is less than the minimum needed to code the number of states in the initial symbolic description. The number of pairs in CDP each state belongs to is evaluated. For state S_i this number is referred to as Ind (S_i). Each state pair (S_i, S_j) in CDP has associated an index equal to the sum of Ind (S_i) and Ind (S_j). In each iteration the pair with the maximum index is discriminated.

The cost function has been designed to force the number of bits to be small. In this sense each pair of states is discriminated so that the cardinality of the resulting *Discriminated_pairs* list is a minimum. It also aims at reducing the area of final realisations by taking into account the fulfillment of the adjacencies derived from Humphrey's or Armstrong's rules [16, 17].

4.2 Example 3

As an example of how the algorithm works we use the FSM shown in Fig. 7 taken from Reference 18. To simplify the explanation only the first mentioned cost criteria is evaluated.

	000	001	010	011	100	101	110
s_1	s_1, 0	s_-, –	s_4, 0	s_5, 1	s_2, 0	s_1, –	s_-, –
s_2	s_2, 0	s_4, 1	s_1, –	s_-, –	s_1, –	s_1, 1	s_-, –
s_3	s_2, 0	s_4, 1	s_1, 1	s_-, –	s_-, –	s_-, –	s_7, 0
s_4	s_-, –	s_5, –	s_-, –	s_2, –	s_2, 0	s_-, –	s_1, –
s_5	s_2, –	s_5, –	s_1, –	s_-, –	s_2, –	s_5, –	s_1, 1
s_6	s_2, 0	s_3, –	s_-, 1	s_8, 1	s_6, 1	s_7, 0	s_-, –
s_7	s_-, –	s_3, 1	s_-, –	s_5, 1	s_-, –	s_7, 0	s_8, 0
s_8	s_1, 1	s_5, 0	s_4, 1	s_2, 0	s_2, –	s_5, –	s_1, 1

NS, z

Fig. 7 *State table for FSM in Example 3*

Initialisation phase

N, the cardinality of the largest maximal incompatible, is three and so the initial code length $nb = nb_0 = \lceil \log_2 N \rceil = 2$

Code Matrix:

s_1 00 s_2 — s_3 — s_4 —

s_5 — s_6 — s_7 — s_8 —

Discriminated_pairs = {∅}; *Identical_pairs* = {∅};
Implied_by_identical_pairs = {∅}
Compulsory_discrimination_pairs = *incompatible state pairs* = $\{(s_1, s_3), (s_1, s_6), (s_1, s_8), (s_2, s_6), (s_2, s_7), (s_2, s_8), (s_3, s_5), (s_3, s_8), (s_4, s_6), (s_4, s_7), (s_5, s_6), (s_5, s_7), (s_6, s_8), (s_7, s_8)\}$

Assignment phase

Iteration 1: The number of pairs in CDP each state belongs to is evaluated. The pair (s_6, s_8) maximises the parameters Ind (S_i) + Ind (S_j) and so this pair is selected to be discriminated in this stage.

The assignments which discriminate the states S_6 and S_8 are

(1) s_6 1– (2) s_6 0– (3) s_6 –1 (4) s_6 –0

s_8 0– s_8 1– s_8 –0 s_8 –1

The four assignments have the same cost. In every case the resulting CDP list has 12 elements. None of the four assignments implies the addition of pairs to that list. The first and third options discriminate the pairs (s_6, s_8) and (s_1, s_6). The second and fourth discriminate the pairs (s_6, s_8) and (s_1, s_6). In this case the first assignment is arbitrarily selected. This is frequent at the first stages of the procedure because there are many unspecified bits in the codes of the states. The data structures are updated and the results are

Code matrix:

s_1 00 s_2 — s_3 — s_4 —

s_5 — s_6 1– s_7 — s_8 0–

Compulsory_discrimination_pairs = $\{(s_1, s_3), (s_1, s_8), (s_2, s_6), (s_2, s_7), (s_2, s_8), (s_3, s_5), (s_3, s_8), (s_4, s_6), (s_4, s_7), (s_5, s_6), (s_5, s_7), (s_7, s_8)\}$

Iteration 2: The pair selected to be discriminated is (s_7, s_8). Candidate bit assignments are

(1) s_7 1– (2) s_7 –1 (3) s_7 –0

s_8 **0**– s_8 **0**0 s_8 **0**1

Those bits already assigned appear in bold. The cost function is evaluated for first and third assignments but not for the second one. This is because it is not legal. Clearly if this second option is taken, the states s_1 and s_8 have identical assignments but the pair (s_1, s_8) belongs to the list of pairs of compulsory discrimination. The discrimination of two states with identical codes can only be achieved by adding a new state variable.

The assignment (1) discriminates the pairs (s_7, s_8) and (s_1, s_7). The pair (s_7, s_8) is the only one removed from the CDP list. The discrimination of (s_1, s_7) which did not belong to CDP involves adding to that list those state pairs which imply (s_1, s_7) and have not discriminated assignments. In this case the pairs (s_3, s_4), (s_3, s_7) and (s_3, s_6) must be added to the list. The cost of this assignment is 14 (there are 14 pairs in the resulting CDP list).

The assignment (3) discriminates the pairs (s_7, s_8) and (s_1, s_8) which are then removed from CDP. It does not require the addition of any pair to that list and so its cost is 10.

Assignment (3) is chosen since it minimises the cost function. Updated data structures are

Code Matrix:

s_1 00 s_2 — s_3 — s_4 —

s_5 — s_6 1– s_7 –0 s_8 01

Discriminated_pairs = $\{(s_1, s_6), (s_1, s_8), (s_6, s_8), (s_7, s_8)\}$
Identical_pairs = {∅};
Implied_by_identical_pairs = {∅}
Compulsory_discrimination_pairs = $\{(s_1, s_3), (s_2, s_6), (s_2, s_7), (s_2, s_8), (s_3, s_5), (s_3, s_8), (s_4, s_6), (s_4, s_7), (s_5, s_6), (s_5, s_7)\}$

Iteration 3: The pair (s_5, s_3) is discriminated next. Again candidate assignments are enumerated

(1) s_3 1– (3) s_3 –1

s_5 0– cost = 6 s_5 –0 cost = 9

(2) s_3 0– (4) s_3 –0

s_5 1– cost = 15 s_5 –1 cost = 14

Table 1: Example 3 summary

	Iteration 4	Iteration 5	Iteration 6	Iteration 7
Pair to discriminate	(s_2, s_7)	(s_3, s_4)	(s_3, s_6)	(s_2, s_8)
Minimum cost discriminating assignment	s_2 0– s_7 10 cost = 6	s_3 1– s_4 0– cost = 3	s_3 11 s_6 10 cost = 1	s_2 00 s_8 01 cost = 0
DP	$DP_4 =$ $(s_1, s_6), (s_1, s_3), (s_1, s_7), (s_1, s_8), (s_2, s_3), (s_2, s_6), (s_2, s_7), (s_3, s_5), (s_3, s_8), (s_5, s_6), (s_5, s_7), (s_6, s_8), (s_7, s_8)$	$DP_5 = DP_4 \cup (s_3, s_4), (s_4, s_6), (s_4, s_7)$	$DP_6 = DP_5 \cup (s_3, s_7), (s_3, s_6)$	$DP_7 = DP_6 \cup (s_2, s_8)$
IP	$\{\varnothing\}$	$\{\varnothing\}$	(s_6, s_7)	$(s_6, s_7), (s_1, s_2)$
IIP	$\{\varnothing\}$	$\{\varnothing\}$	$(s_5, s_8), (s_1, s_2), (s_1, s_4), (s_2, s_5), (s_4, s_5), (s_1, s_5)$	$(s_5, s_8), (s_1, s_2), (s_1, s_4), (s_2, s_5), (s_4, s_5), (s_1, s_5)$
CDP	$(s_2, s_8), (s_4, s_6), (s_4, s_7), (s_3, s_7), (s_3, s_4), (s_3, s_6)$	$(s_2, s_8), (s_3, s_7), (s_3, s_6)$	(s_2, s_8)	$\{\varnothing\}$

Assignment (1) is selected and updated data structures are

Code Matrix:

s_1 00 s_2 — s_3 1– s_4 —

s_5 0– s_6 1– s_7 –0 s_8 01

Discriminated_pairs $= \{(s_1, s_6), (s_1, s_3), (s_1, s_8), (s_3, s_5), (s_3, s_8), (s_5, s_6), (s_6, s_8), (s_7, s_8)\}$
Identical_pairs $= \{\varnothing\}$;
Implied_by_identical_pairs $= \{\varnothing\}$
Compulsory_discrimination_pairs $= \{(s_2, s_6), (s_2, s_7), (s_2, s_8), (s_4, s_6), (s_4, s_7), (s_5, s_7)\}$

While CDP is not empty the algorithm follows discriminating state pairs. The remaining iterations do not introduce anything new. Table 1 presents a summary.

The state assignment obtained with the algorithm is:

s_1 00 s_2 00 s_3 11 s_4 0–

s_5 0– s_6 10 s_7 10 s_8 01

The state encoding derived with our algorithm is equivalent to the following state merging $C = \{(s_1, s_2, s_4, s_5), (s_3), (s_6, s_7), (s_4, s_5, s_8)\}$. So the coding procedure is equivalent to the generation of a symbolic description with four states and to the binary encoding of it.

A two level realisation of the FSM using this encoding requires eight product terms and two memory elements. Using the symbolic description in Fig. 7 as input to a conventional state assignment program produces an encoding with three state variables. In particular a two level realisation with 13 product terms and three memory elements was obtained with the well known state assignment tool NOVA [6].

5 Experimental results

The algorithm is particularly suited to those machines for which state merging applies and so a subset of the MCNC machines (group 1) [19], those with compatibles or equivalent states, has been used to test the algorithm. Many examples picked out from switching circuit textbooks and journal papers on minimisation have also been tried, all of them are in Reference 20 (group 2). Two figures of merit are focused on: the area occupied by the combinational component in an PLA implementation of the FSM and the time which is required for design.

Table 2 depicts the results for the first group of machines and Table 3 for the second group. The number of primary inputs (*ni*), primary outputs (*no*) and states (*ns*) are shown for each of the machines to which SMAS (C implementation of the new algorithm), NOVA [21] (*i_hybrid* algorithm*), and a synthesis procedure including state reduction (ARNES [5] + NOVA *i_hybrid*) have been applied. The number of state variables (*nv*), the number of product terms (*tp*) after logic minimisation and the size $((2ni + 3nv + no) \times tp)$ of the PLA implementing the combinational component for the assignments obtained with each program after code shifting† and minimisation with ESPRESSO [22] are also shown in Tables 2 and 3. Times are given in seconds in a SUN WorkStation 4/330. Columns with ratio of sizes, A_1 and A_2 are included with:

$$A_1 = \frac{Size_{SMAS}}{Size_{NOVA_{ihibrid}}} \quad A_2 = \frac{Size_{SMAS}}{Size_{(ARNES)+(NOVA_{ihibrid})}}$$

Both tables show that in all the cases the number of state variables in SMAS's assignments is less than or equal to the number of state variables in NOVA's assignments. Concerning group 1 machines, SMAS obtains more efficient implementations (in terms of size) than NOVA in 15 of the 17 machines in Table 2. In particular, let us focus on the set formed by *don-file*, *modulo*12, *s*1*a* and *s*8. These machines do not need to be realised as FSMs. NOVA is not able to detect it and supplies an assignment for them. This leads to realisations which are expensive in terms of area. None of the machines in Table 3 is more efficiently assigned with NOVA than it is with SMAS. On average the area savings with SMAS is around 50% for group 1 and 57% for group 2. Also the time comparison is favourable for SMAS.

* One of the NOVA's options. It achieves a very good trade-off between quality of the solutions and CPU time. It uses codes with minimum length

† Each bit position which is '1' in the code of the state appearing more times in the next state field is complemented in the codes of all the states. This 'classic' heuristic is incorporated in many state assignments tools and has been reported to lead to area savings. Clearly, this operation cannot change a valid assignment into an invalid one

Table 2: Experimental results for group 1 machines

	NOVA *i_hybrid*						ARNES + NOVA *i_hybrid*				SMAS				A_1	A_2
	ni	*no*	*nv*	*tp*	*size*	*time*	*nv*	*tp*	*size*	*time*	*nv*	*tp*	*size*	*time*		
bbara	4	2	4	25	550	1.9	3	20	380	1.5	3	21	399	0.8	0.72	1.05
bbsse	7	7	4	30	990	3.0	4	31	1023	5.2	4	28	924	5.2	0.93	0.90
beecount	3	4	3	13	247	1.1	2	9	144	1.0	3	10	190	0.3	0.77	1.32
donfile	2	1	5	35	700	394	0	0	0	0.1	0	0	0	0.1	0.00	1.00
ex1	9	19	5	48	2496	38.6	5	41	2132	34.7	5	53	2756	32.1	1.10	1.29
ex2	2	2	5	29	609	3.4	3	12	180	3.4	3	12	180	1.7	0.30	1.00
ex3	2	2	4	18	324	1.4	3	10	150	1.6	3	8	120	0.4	0.37	0.80
ex5	2	2	4	14	252	3.7	2	6	72	1.1	2	7	84	0.2	0.33	1.16
ex7	2	2	4	17	306	2.3	2	9	108	1.3	2	8	96	0.3	0.31	0.88
lion9	2	1	4	8	136	1.9	2	7	77	0.9	3	9	126	0.3	0.93	1.63
mark1	5	16	4	21	798	23.4	4	17	646	5.5	4	21	798	3.7	1.00	1.23
modulo12	1	1	4	13	195	0.9	0	0	0	0.1	0	0	0	0.1	0.00	1.00
opus	5	6	4	16	448	1.9	4	16	448	1.6	4	15	420	0.9	0.94	0.94
s1a	8	6	5	76	2812	5.0	0	0	0	12.7	0	0	0	12.7	0.00	1.00
s8	4	1	3	10	180	1.1	0	0	0	0.1	0	0	0	0.1	0.00	1.00
tbk	6	3	5	154	4620	979	4	53	1431	146.6	4	55	1485	48.2	0.32	1.03
train11	2	1	4	9	153	3.3	2	6	66	1.1	2	6	66	0.3	0.43	1.00

ni no. of input variables
no no. of outputs
tp no. of product terms
time in seconds on a workstation SUN 4/330
size $(2ni + 3nv + no) \times tp$

Table 3: Experimental results for group 2 machines

	NOVA *i_hybrid*						ARNES + NOVA *i_hybrid*				SMAS				A_1	A_2
	ni	*no*	*nv*	*tp*	*size*	*time*	*nv*	*tp*	*size*	*time*	*nv*	*tp*	*size*	*time*		
FSM1	2	1	3	8	112	1.1	2	5	55	0.8	2	4	44	0.1	0.39	0.80
FSM2	2	1	4	12	204	1.6	2	8	88	1.1	3	11	154	0.3	0.75	1.75
FSM3	2	1	3	11	154	1.2	2	5	5	0.7	2	5	55	0.2	0.36	1.00
FSM4	2	1	3	8	112	1.0	2	4	44	0.9	2	4	44	0.1	0.39	1.00
FSM5	2	1	3	9	126	1.3	2	5	55	0.9	2	6	66	0.1	0.52	1.20
FSM6	2	1	5	31	620	164	4	15	255	2.6	4	15	255	3.6	0.41	1.00
FSM7	2	1	4	15	255	6.9	2	8	88	0.8	2	7	77	0.2	0.30	0.87
FSM8	2	1	3	9	126	1.4	2	7	77	0.9	2	7	77	0.1	0.61	1.00
FSM9	3	1	4	15	255	6.1	2	7	91	0.9	2	7	91	0.4	0.36	1.00
FSM10	3	1	3	18	288	4.0	3	13	208	1.5	2	8	104	0.3	0.36	0.50
FSM11	2	1	3	10	140	1.3	2	7	77	0.9	3	6	84	0.2	0.60	1.09
FSM12	2	1	4	13	221	3.5	2	8	88	0.9	2	7	77	0.3	0.35	0.87
FSM13	3	1	3	13	208	1.4	2	8	104	0.9	2	8	104	0.1	0.50	1.00
FSM14	2	1	3	7	98	0.9	1	3	24	0.8	1	3	24	0.1	0.24	1.00
FSM15	3	1	4	11	209	5.4	2	4	52	1.0	2	4	52	0.4	0.25	1.00

ni no. of input variables
no no. of outputs
tp no. of product terms
time in seconds on a workstation SUN 4/330
size $(2ni + 3nv + no) \times tp$

Next we compare SMAS with the synthesis procedure including state reduction with ARNES and optimal state assignment with NOVA. In six out of the 17 machines in Table 2, the same area results were obtained with both approaches. In seven machines worse results were obtained with SMAS and better in four cases. For machines from Table 3, the area results were identical in eight cases, in three SMAS achieved more expensive implementations and in the remaining four SMAS obtained cheaper realisations. The CPU time invested by SMAS is smaller for both groups of machines, with the exception of FSM6.

6 Conclusions

We have presented a state assignment algorithm which, concurrently with the assignment of binary codes in the symbolic representation, exploits the potentials of state merging and state splitting. The algorithm heuristically minimises the area of the combinational component after logic reduction. The conventional constraint of giving each state a unique distinct code have been removed and conditions under which two states can share one or more binary codes have been stated so that the initial symbolic machine and the encoded one have the same functionality. The algorithm is original and paves the way for a new treatment of state assignment.

Experimental results have been shown. The algorithm is extremely advantageous (reductions of around 50%) for the assignment of non-minimised FSMs because of the power of state merging. It achieves slightly worse results than NOVA for previously minimised machines.

Paying more attention to the cost function will lead to more efficient assignment of FSMs with a minimum number of states. More exact criteria such as input and and output constraint satisfaction [23, 24] for two level implementations are being incorporated into the cost function instead of counting state adjacencies. Finally, future work includes a BDD-based [25] version of the algorithm.

7 References

1 HOPCROFT, J.: 'An *n* log *n* algorithm for minimizing states in a finite automaton', *in* KOHAVI, (Ed.): 'Theory of machines and computation' (Academic Press, 1971), pp. 189–196

2 PFLEEGER, C.P.: 'State reduction in incompletely specified finite-state machines', *IEEE Trans. Comput.*, 1973, **C-22**, (12), pp. 1099–1102
3 HACHTEL, G.D., RHO, J.-K., SOMENZI, F., and JACOBY, R.: 'Exact and heuristic algorithms for the minimization of incompletely specified state machines'. Proceedings of the European *Design Automation* Conference, 1991, pp. 184–191
4 KANNAN, L.N., and SARMA, D.: 'Fast heuristic algorithms for finite state machine minimization'. Proceedings of the European *Design Automation* Conference, 1991, pp. 192–196
5 AVEDILLO, M.J., QUINTANA, J.M., and HUERTAS, J.L.: 'Efficient state reduction methods for PLA-based sequential circuits', *IEE Proc. E*, 1992, **139**, (6)
6 VILLA, T.: 'NOVA: state assignment of finite state machines for optimal two-level logic implementation', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1990, **9**, (9), pp. 905–924
7 LEE, E.B., and PERKOWSKI, M.A.: 'Concurrent minimization and state assignment of finite state machines'. Proceedings of the 1984 International Conference on *Syst., Man. and Cyb.*, IEEE, Halifax, October 1984
8 ASHAR, P., DEVADAS, S., and NEWTON, A.R.: 'Sequential logic synthesis' (Kluwer Academic Publishers, 1992)
9 RUDELL, R., SANGIOVANNI-VICENTELLI, A.L., and DE MICHELI, G.: 'A finite state machine synthesis system'. Proceedings of International Symp. on *Circuits and Systems*, May 1985, pp. 647–650
10 HARTMANIS, J., and STEARNS, R.E.: 'Some dangers in the state reduction of sequential machines', *Inf. Control*, 1962, **5**, pp. 252–260
11 DEVADAS, S., MA, H.-K.T, NEWTON, A.R., and SANGIOVANNI-VINCENTELLI, A.: 'Irredundant sequential machines via optimal logic synthesis', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1990, **9**, (1), pp. 8–17
12 UNGER, S.H.: 'Asynchronous sequential switching circuits' (Wiley-Interscience, New York, 1969)
13 KOHAVI, Z.: 'Switching and finite automata theory' (McGraw-Hill Book Company, New York, 1978), 2nd edn.
14 GRASSELLI, A., and LUCCIO, F.: 'Some covering problems in switching theory', *in* 'Networks and switching theory' (Academic Press, 1968)
15 TSENG, C., PRABHU, A.M., LI, C., MEHMOOD, Z., and TONG, M.M.: 'A versatile finite state machine synthesis system'. Proceedings of the 1986 International Symposium on *Circuits and Systems*, 1986, pp. 206–209
16 HUMPHREY, W.S.: 'Switching circuits with computer applications' (McGraw-Hill Book Company, New York, 1957), Section 10.5, pp. 233–252
17 ARMSTRONG, D.B.: 'A programmed algorithm for assigning internal codes to sequential machines', *IRE Trans. Electro. Comput.*, 1962, **EC-11**, pp. 466–472
18 DE SARKAR, S.C., BASUY, A.K., and CHOUDHURY, A.K.: 'On the determination of irredundant prime closed sets', *IEEE Trans. Comput.*, 1971, **C-20**, pp. 933–938
19 LISANKE, R.: 'Introduction of synthesis benchmarks'. International Workshop on *Logic Synthesis*, North Carolina, 1989
20 REUSCH, B., and MERZENICH, W.: 'Minimal coverings for incompletely specified sequential machines', *Acta Inform.*, 1986, **22**, pp. 663–678
21 VILLA, T.: 'NOVA, User's Manual'. University of California, Berkeley, October 1988
22 BRAYTON, R.K., HACHTEL, G.D., McMULLEN, C., and SANGIOVANNI-VINCENTELLI, A.L.: 'Logic minimization algorithms for VLSI synthesis' (Kluwer Academic Publishers, Boston, Massachusetts, 1984)
23 DE MICHELI, G., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.L.: 'KISS: a program for optimal state assignment of finite state machines'. Proceedings of International Conference on *Computer Aided Design*, November 1984, pp. 209–211
24 DE MICHELI, G.: 'Symbolic design of combinational and sequential logic circuits implemented by two-level macros', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1986, **CAD-5**, pp. 597–616
25 BRYANT, R.E.: 'Graph based algorithms for Boolean function manipulation', *IEEE Trans. Comput.*, 1986, **C-35**, (8), pp. 677–691