VLSI Design, 2002 Vol. 14 (2), pp. 171–181

Taylor & Francis
Taylor & Francis Group

COPAS: A New Algorithm for the Partial Input Encoding Problem

MANUEL MARTÍNEZ, MARÍA J. AVEDILLO*, JOSÉ M. QUINTANA and JOSÉ L. HUERTAS

Instituto de Microelectrónica de Sevilla, Edif. CICA, Avda. Reina Mercedes s/n, Sevilla 41012, Spain

(Received 31 March 2000; Revised 23 May 2000)

Frequently, the logic designer deals with functions with symbolic input variables. The binary encoding of such symbols should be chosen to optimize the final implementation. Conventionally, this input encoding (IE) problem has been solved in a two-step process. First step generates constraints on the relationship between codes for different symbols, called group constraints. In a following step, symbols are encoded such that constraints are satisfied. This paper addresses the partial input encoding problem (PIE), a variation of the IE problem which generates codes of minimum length. The role of group constraints within the framework of the PIE problem has been questioned. This paper describes an algorithm that unlike conventional approaches, which try to maximize the number of satisfied constraints, targets the economical implementation of each input constraint. The proposed approach is based on a powerful heuristic that produces high quality results in shorter time compared to previous algorithm.

Keywords: Face constraints; Group constraint; Dichotomy; Partial satisfaction; Input encoding; Logic synthesis

INTRODUCTION

Frequently, when synthesizing integrated circuits, specification of design includes symbolic variables. A binary encoding of such symbols should be chosen to optimize the final implementation. This task is known as the encoding problem. The difficulty of the encoding problem resides in the modeling of the subsequent optimization step. In this paper, we address the input encoding (IE) problem. This is, given a function with symbolic inputs, determine a binary encoding of the symbols such that, after logic minimization, the implementation is of minimum size. IE arises in many different synthesis tasks. Examples are the encoding of mnemonic input fields of the microcode, the encoding of symbolic inputs that appear in high level descriptions or the state assignment of finite state machines.

Figure 1a shows a two-output function with a symbolic input from [1]. In this example, the encoding problem is to replace the symbols by binary representations so that the final implementation has a minimum number of product terms. An exhaustive search technique trying all possible input codes would be excessively costly, because it would require an exponential number of logic minimizations. A fundamental development in the IE problem was the work in Ref. [2] where a tabular representation of the function with symbolic inputs is symbolically minimized using two level multiple-valued minimization [3]. This minimization step generates constraints (group, input or face constraints) on the relationship between codes for different symbols. In a second step, symbols are encoded in such a way that constraints are satisfied. Satisfaction of the constraints guarantees that the optimization at the symbolic level will be preserved in the boolean domain. This two step approach has been taken in many works [4–8]. Let us summarize this strategy with the example original from Ref. [1]. The minimized symbolic representation is shown in Fig. 1b. Constraints for the encoding process are given by the symbolic implicants with more than one symbol. A boolean representation with the same number of implicants can be obtained if the codes assigned to the symbols in each group form a cube in the boolean space in such a way that it does not contain the codes of any symbol which is not in the group.

*Corresponding author. Tel.: +34-95-50-56667. Fax: +34-95-50-56686. E-mail: avedillo@imse.cnm.es

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/10655140290010088

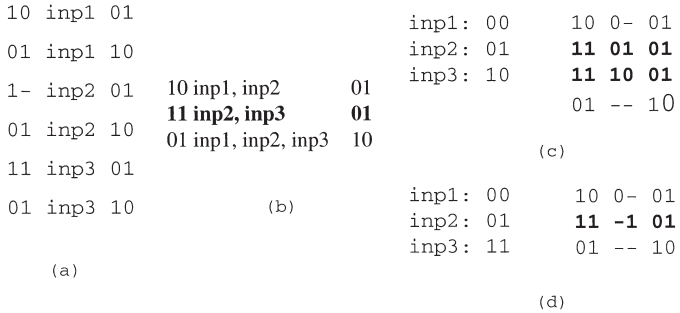

FIGURE 1 Encoding process: (a) function with a symbolic input, (b) minimized symbolic function, (c) encoding 1 and minimized boolean implementation, (d) encoding 2 and minimized boolean implementation.

Encoding 1 in Fig. 1c does not satisfy constraint (inp2, inp3) and the symbolic implicant corresponding to it is implemented with two product terms in the boolean domain (in bold in the Figure). Encoding 2 in Fig. 1d satisfies all constraints and allows implementing each symbolic implicant with a single cube. This strategy was extended to multi-level implementations with the development of multi-level multiple-valued optimization algorithms [9].

In some applications, satisfying the complete set of constraints involves such an increase of the length of the codes that gains in terms of area are not usually achieved. Because of this, many practical IE algorithms address the partial input encoding problem (PIE), a variation of the IE problem which generates codes of minimum length. Recently, logic synthesis tasks such as the functional decomposition for look-up tables based field programmable gate arrays have been modeled as PIE problems [11], which contributes to the importance of the problem.

Conventional approaches for the PIE problem try to maximize the number of satisfied face constraints, this is, the number of symbolic implicants which can be implemented with a single product term, without considering cost effective implementation of the remaining. More recent works aim at minimizing the cost of implementing the complete set of constraints. In particular, Ref. [10] has been successfully applied to different synthesis tasks such as logic decomposition or logic partitioning, but its intensive use of logic minimization makes it useless for large problems. In this paper, we propose a new algorithm for the PIE problem which also targets the economical implementation of each input constraint, but greatly improves the time performance described in Ref. [10] without degrading the quality of the results.

The rest of the paper is organized as follows. Section 2 introduces basic definitions to mathematically formulate the problem. Section 3 presents several encoding examples as a motivation for the work and summarizes previous approaches. Section 4 describes the new approach. In section 5 experimental results are shown and discussed. Finally, in section 6 we give some conclusions.

DEFINITIONS AND NOTATIONS

In this section, we review several definitions of constrained IE, originally introduced in different references [2, 4, 13].

DEFINITION 1 *Binary encoding*: given a set of symbols $S = \{S_1, S_2, \ldots, S_n\}$ and an integer k, a binary encoding of S is a-one-to one mapping $S \rightarrow \{0,1\}^k$.

Encoding can be represented as a code matrix $C \in \{0,1\}^{n \times k}$ where the ith row represents the code assigned to symbol S_i, and the jth column represents bit j of the encoding.

DEFINITION 2 *Encoding dichotomy*: an encoding dichotomy is a two block partition, $(B_1{:}B_2)$, of symbols such that one code bit of the symbols in B_1 is assigned 0(1) while the same code bit is assigned 1(0) for the symbols in B_2. We can think of each column of the code matrix as an encoding dichotomy.

DEFINITION 3 *Group constraint*: a group constraint gc on the set of input symbols $S = \{S_1, S_2, \ldots, S_n\}$ is a subset S' of symbols from S which must be assigned such that the minimum boolean cube containing their codes does not intersect the codes of the symbols absent from S'.

DEFINITION 4 *Seed dichotomy*: a seed dichotomy d is a disjoint two block partition, $(B_1{:}B_2)$, associated with a group constraint gc_1, such that the block B_1 contains all symbols that belongs to gc_1 and B_2 contains exactly one of the symbols that does not belong to gc_1.

Satisfaction of a group constraint is equivalent to the satisfaction of the whole set of its associated seed dichotomies [13]. A seed dichotomy, d, is satisfied if subset B_1 of d is distinguished from subset B_2 of d by at least one encoding bit. In the following, we will refer to a seed dichotomy simply as dichotomy.

Within this framework, an instance of the IE problem with n input symbols denoted $\{S_1, S_2, \ldots, S_n\}$ can be represented by an input constraint matrix L with as many rows as there are group constraints, and n columns. $L_{ij} = 1$, if the symbol S_j belongs to the i-th constraint and 0 otherwise. Given a set S of n symbols and a constraint matrix L, the IE problem consists in determining a code

matrix $C \in \{0,1\}^{n \times k}$ with minimum value of k, which satisfies L.

$\lceil \log_2 n \rceil$ and a constraint matrix L, determine $C \in \{0,1\}^{n \times s}$ which maximizes the number of dichotomy constraints satisfied.

MOTIVATION AND PREVIOUS WORK

Conventionally, the Partial IE problem is considered as a restriction of the complete one. Group constraints are generated in the same manner. Then, as there is not guarantee of the existence of an encoding of minimum length, which satisfies the constraint matrix, the encoding step is reformulated. There are two different problem statements:

(1) Given a set S of n input symbols, the integer $s = \lceil \log_2 n \rceil$, and a constraint matrix L, determine $C \in \{0,1\}^{n \times s}$ which maximizes the number of group constraints satisfied.
(2) Given a set S of n input symbols, the integer $s =$

Algorithms *i_greedy* and *i_hybrid* in NOVA [7], and CUBIC [6] are examples of the first formulation while ENCORE [8] corresponds to the second one. It has been claimed that the role of both group and dichotomy constraints within the framework of the partial IE problem should be questioned. As a motivation for our work, let us introduce two examples which illustrate the above statement.

Example 1 This example shows that two encodings which satisfy the same subset of group constraints can result in boolean implementations with different costs. Consider the function with a symbolic input shown in Fig. 2a. The minimized symbolic function and derived input constraints are shown in Fig. 2b. Figure 2c and d give

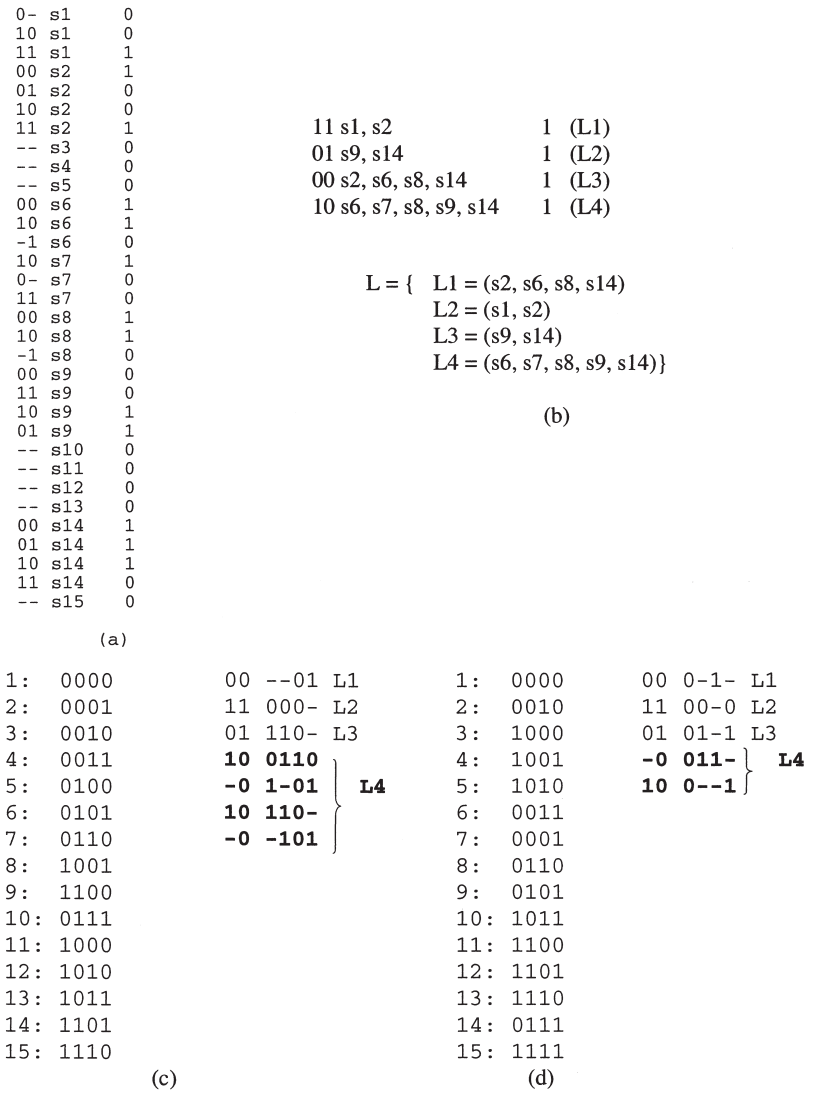

FIGURE 2 Example 1: (a) function with symbolic input, (b) minimized symbolic function and input constraints, (c) encoding 1 and minimized boolean implementation, (d) encoding 2 and minimized boolean implementation.

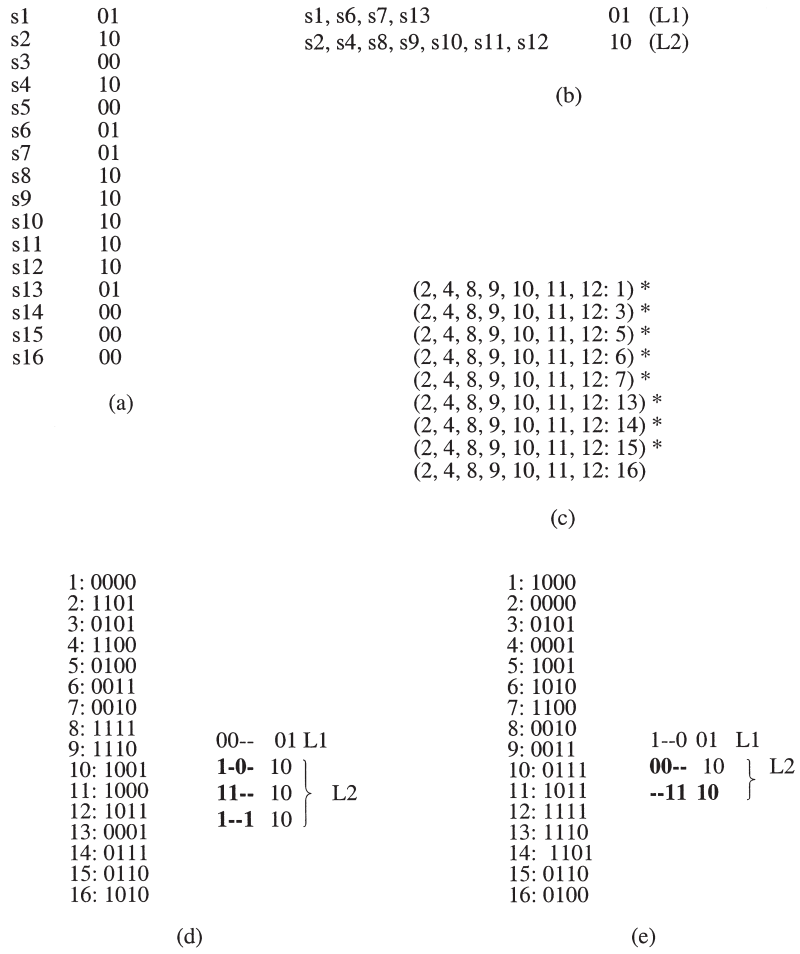

FIGURE 3 Example 2: (a) function with symbolic input, (b) minimized symbolic function, (c) seed dichotomies for *L*2, (d) encoding 1 and minimized boolean implementation, (e) encoding 2 and minimized boolean implementation.

two codes for input symbols. Both encodings satisfy group constraints *L*1, *L*2 and *L*3, and violate *L*4 (in fact this constraint cannot be satisfied with minimum code length). However, the symbolic implicant corresponding to that row is implemented with four product terms with encoding 1 (Fig. 2c) and with only two when encoding 2 is used (Fig. 2d).

Example 2 This example shows that verifying a larger number of the seed dichotomies associated with a group constraint does not mean that it produces smaller implementations. Consider the function with the symbolic input in Fig. 3a. Figure 3b shows the minimized symbolic function and Fig. 3c, the seed dichotomies for *L*2. Figure 3d shows an encoding satisfying the dichotomies marked with an asterisk in Fig. 3c and Fig. 3e an encoding that does not satisfy any dichotomy constraint. Using encoding 1 the encoded function is implemented with four product terms (Fig. 3d). Encoding 2 requires three product terms (Fig. 3e).

These examples point out that the number of satisfied constraints and/or dichotomies is not an adequate measure of the quality of an encoding. So, solving PIE problems with algorithms developed for the complete one could lead to suboptimal results. More appropriate measurements of the satisfaction of the input constraints for partial IE problems, which also consider the cost of implementing symbolic implicants associated to unsatisfied constraints, have been proposed [5,10]. In an algorithm called ENC [10], the cost of implementing unsatisfied constraints is evaluated using ESPRESSO [12]. In this work, a Boolean function is associated with each input constraint. Its on-set contains the codes of the symbols in the constraint and its off-set contains the codes of the symbols not in the constraint. The unused codes are in the dc-set. For a two-level design style, the number of product terms in a sum-of-product representation of this function after minimization gives the cost of an input constraint for a given encoding. Clearly, there is a single product term in the representation of the functions associated with satisfied constraints. For a multi-level style, the cost is given by the number of literals in a factored form (sum-of-product representation in practice) of the same minimized function.

ENC has been successfully applied to different synthesis tasks such as logic decomposition or logic partitioning. However, intensive use of logic minimization makes it impractical even for medium size

problems. Next, we briefly describe ENC in order to show the high number of logic minimization operations it requires. ENC is based on a splitting and merging strategy. The splitting phase is used to divide a given encoding problem into two smaller ones, each to be encoded using one less bit. Assuming that each subproblem is solved optimally, the solution for the original encoding problem is generated by the merging step. The splitting procedure is carried out recursively on each resulting partition until only two symbols remain and a single encoding dichotomy is generated. The merging procedure obtains an encoding of length c of the symbols of a partition S from the encodings of length $c-1$ of the subpartitions S_0 and S_1 ($S_0 \cup S_1 = S$). The merging uses as cost function the number of product terms or the literals required to implement in two levels the input constraints restricted to the symbols in S, and implies intensive application of logic minimization tools. For this, a set of candidate encoding dichotomies, D, is generated. Let $D_0(D_1)$ be the set of $c-1$ encoding dichotomies in the solution for subpartition $S_0(S_1)$, then $D = (S_0 : S_1) \cup D_0 \times D_1 \cup D_1 \times D_0$. For example, consider partitions $S_0 = \{s_0, s_1, s_2, s_4\}$ and $S_1 = \{s_3\}$ and assume $D_0 = \{(s_0 s_4 : s_1 s_2), (s_0 s_2 : s_1 s_4)\}$ and $D_1 = \{(s_3 :)\}$. The candidate dichotomies are $D = \{(s_0 s_1 s_2 s_4 : s_3), (s_0 s_3 s_4 : s_1 s_2), (s_0 s_4 : s_1 s_2 s_3), (s_0 s_2 s_3 : s_1 s_4)(s_0 s_2 : s_1 s_3 s_4)\}$. For each selection of c candidate encoding dichotomies from D which distinguishes every symbol, ESPRESSO [12] is used to evaluate the number of product terms or literals required to implement the constraints. Continuing with the example, there are $\binom{5}{3} = 10$ subsets of D with cardinality 3, 8 of them correspond to valid encodings because they distinguish every symbol. Each of these is evaluated with ESPRESSO and the best one is selected. In order to give an idea of the number of logic minimization steps required per merging step, Table I shows the number of subsets of D with cardinality c for different parameters. Solving a problem with 32 symbols involves 15 merging operations, eight with $c = 2$; four with $c = 3$; two with $c = 4$ and one with $c = 5$.

To overcome the limitations of ENC, we have developed an algorithm which also applies ESPRESSO in order to precisely measure the quality of an encoding in relation to the set of input constraints, but greatly reduces the number of logic minimization operations without degrading the quality of the results. The new algorithm can be useful in solving larger instances of the above mentioned synthesis tasks.

TABLE I Estimation of logic minimizations per merging step

c	card(D_0)	card(D_1)	card(D)	# selections
3	2	1	5	10
3	2	2	9	84
4	3	3	19	3876

c: length of encoding after merging; card(D_0)/card (D_1): number of encoding dichotomies in solutions to subproblems S_0/S_1; card(D): cardinality of the set of candidate encoding dichotomies; # selections: number of subset of set D with cardinality c.

THE NEW ALGORITHM

This section describes the new algorithm proposed for the PIE problem. It follows the splitting and merging strategy but the procedures implementing each step are completely different from ENC. It significantly speeds up the process. Main distinguishing features of our method are:

(1) The use of logic minimization at each merging step is greatly reduced by restricting the search space on the basis of an experimentally proved effective theoretical model for selecting a reduced set of good candidate encodings.
(2) The splitting of symbols is strongly coupled to the merging-selection model.
(3) The recursive splitting procedure is stopped when the number of bits to be used in the encoding subproblems is less or equal than a given bound, *bound*. At this stage, the encoding subproblems are solved to maximize the number of restricted input constraints satisfied. *Bound* is chosen so that solving the subproblems is faster than carrying out the remaining splitting and merging of solutions.

Figure 4 shows a *Pidgin-C* description of the algorithm. It starts obtaining the constraint matrix. The core of the algorithm is the recursive function *assign*. *Assign* has three arguments: a set of symbolic values, S, a constraint matrix on those symbols, L, and the code length the symbols are going to be encoded with, *nv*. When function *assigns* is first called, it takes the complete set of symbolic inputs, *Sc*, the complete constraint matrix *Lc* and fixes the code length to the minimum (code_length $= \lceil \log_2(\text{Card}(Sc)) \rceil$). In subsequent calls, *assign* deals with a subset of the symbols and a constraint matrix restricted to them. *Assign* works as follows, while the size of the given encoding problem (*nv*) is over *bound*, it generates two smaller problems to be encoded with one less bit (*generate_partition*()). Once these two subproblems are solved, a solution is obtained for the original one (*merge_selects*()). If the problem is small enough, no partitioning takes place but it is solved (*resolve*()). Now we explain in more detail the three main functions. We start with the merging step because, as mentioned above, the splitting is strongly coupled with it and so it is better to postpone the description of the partitioning phase.

Merging and Selection Phase

In this step, *merge_select*(), given encodings with ($nv-1$) bits for the symbols in S_0, $C_0 \in \{0,1\}^{\text{Card}(S_0)\times(nv-1)}$, and in S_1, $C_1 \in \{0,1\}^{\text{Card}(S_1)\times(nv-1)}$, searches for an encoding

```
COPAS()
{
(Sc, Lc) = get_constraints();
code_length = ⌈log2(Card(S))⌉;
assign (Sc, Lc, code_length);
}

assign (S, L , nv)
{
 if(nv > bound){
   (S0, S1) = generate_partition(S, L, nv));
   C0 = assign ( S0, L0,nv-1);
   C1 = assign(S1 , L1 ,nv-1);
   C = merge_select(S0, S1, C0, C1);
   return(C);
    }
 else {
   C = resolve(S, L, nv);
   return(C);}
 }
```

FIGURE 4 Pseudocode description of the new algorithm.

C with nv bits for the symbols in S, ($S = S_0 \cup S_1, C \in \{0,1\}^{\mathrm{Card}(S)\times nv}$), which heuristically minimizes the number of product terms required to implement L.

The Merging Model

The matrix C is built as shown in Fig. 5. C_1^* is obtained from C_1 by permuting and or complementing columns as will be explained later. Clearly, the code matrix C obtained in this way is a valid encoding (because it distinguishes every symbol in S) if C_0 and C_1 are valid encodings. Another interesting and attractive feature of C, proved in the Appendix, is that the number of cubes, #cubes, required to implement the constraints in L using C is less or equal to $\#\mathrm{cubes}_0 + \#\mathrm{cubes}_1$ where $\#\mathrm{cubes}_0$ ($\#\mathrm{cubes}_1$) is the number of cubes required to implement the constraints in $L_0(L_1)$ using $C_0(C_1)$. It is possible that some cubes from the two coverings merge leading to #cubes under the stated upper bound. The transformations applied to C_1 aim at reducing #cubes. This way of building matrix C is key to produce efficient implementations of unsatisfied constraints. The following example illustrates this.

Example 3 Let $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$ be a set of states to be encoded and let us suppose that the chosen partition creates the subsets: $S_0 = \{s_0, s_1, s_2, s_3, s_7, s_8\}$ and $S_1 = \{s_4, s_5, s_6, s_9, s_{10}\}$, and $L = (s_1, s_6, s_8, s_{10})$ is a constraint of S. Figure 6a shows matrices C_0 and C_1. Constraint L has been broken in $L_0 = (s_1, s_8)$ implemented in C_0 by the 1-cube 11- and $L_1 = (s_6, s_{10})$ implemented in C_1 by the 1-cube 0-0. Clearly, with $C_1^* = C_1$ two cubes are required to implement L in C: 11-0 and 0-01. However, if the matrix C_1^* shown in Fig. 6b is used, only the cube 11- is needed (Fig. 6c). C_1^* has been obtained from C_1 complementing all its columns, and interchanging second and third columns.

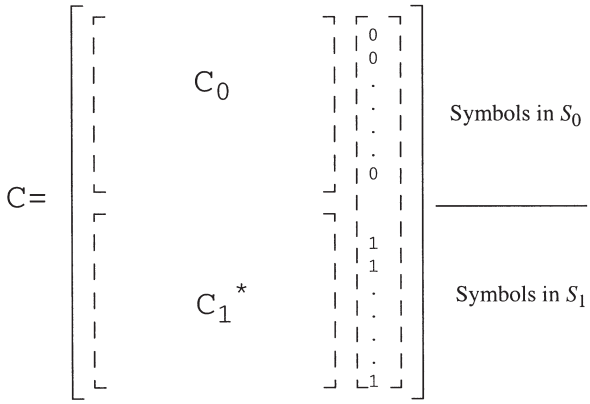

FIGURE 5 Generation of C from C_0 and C_1.

Obtaining C_1^*. *The Link Matrix*

Exhaustive search of the matrix C_1^* which minimizes #cubes implies building each possible C_1^* and using ESPRESSO [12] to evaluate the product terms or literal counts required to implement the constraints with each C. If C_1 has s columns, the number of different matrixes C_1^* is $2^s s!$. For example, merging solutions to subproblems encoded with three bits ($s = 3$) to generate a four bit encoding, which corresponds to the last row of Table I, requires only 48 minimization operations. However, although with this novel merging model the number of minimizations is greatly reduced compared to ENC, exhaustive search of C_1^* is still lengthy for large machines. A method for predicting useful transformations of C_1 has been developed which experimentally has proven to produce good results. In order to determine C_1^*, a matrix, *LINK*, which has as many rows as there are columns in C_0, $(nv - 1)$, and twice the number of columns in C_1, $2(nv - 1)$, is built up, as will be explained later, such that:

LINK$[i][j]$, $1 \le j \le nv - 1$ measures the convenience of using column j of C_1 as column i of C_1^*.
LINK$[i][j]$, $nv \le j \le 2(nv - 1)$ measures the convenience of using the complement of column $(j - (nv - 1))$ of C_1 as column i of C_1^*.

Once *LINK* is available, C_1^* is obtained selecting the highest $nv - 1$ elements from *LINK* restricted to: (a) two elements from the same row cannot be selected, and (b) selecting an element from column j precludes selecting any other element from column j and from column $j \pm (nv - 1)$. These restrictions guarantee that the obtained matrix is valid.

The algorithm combines the use of the link matrix method and the application of logic minimization to improve the results. That is, using the link matrix a set of C_1^* matrices (candidate matrices) is determined instead of a single one. Then, ESPRESSO is used to select the best one among them. Experimental results in the next section show that very good results can be obtained with few candidate matrices and a consequently, a few logic minimization steps.

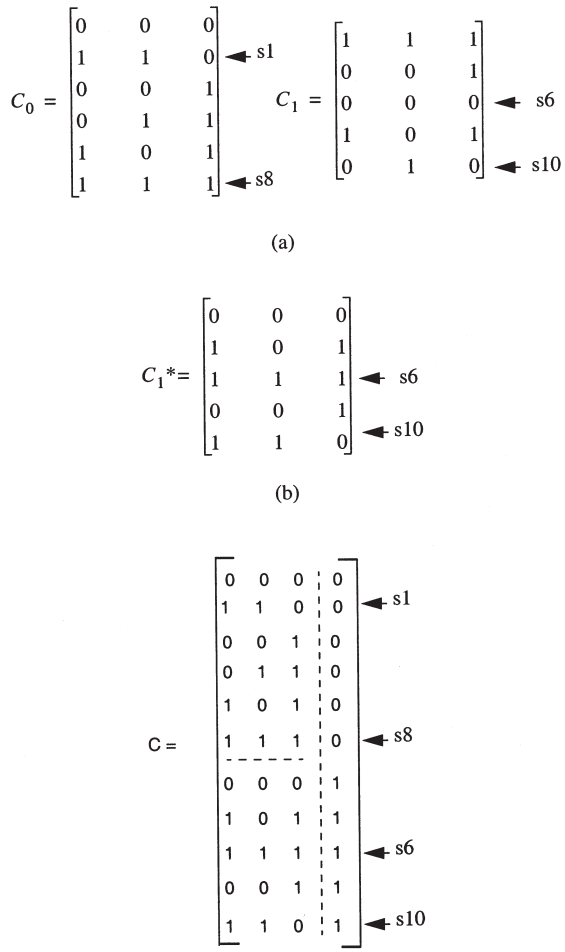

FIGURE 6 Example 3: (a) code matrices for subproblems, (b) modification of C_1, (c) code matrix for original problem.

Procedure implemented to build up *LINK* is as follows. Lets call $P_{Li,0}(P_{Li,1})$ the set of cubes implementing subconstraint Li_0 (Li_1) in $C_0(C_1)$. For each constraint Li in L, the possible merging of a cube from $P_{Li,0}$ with a cube from $P_{Li,1}$ is examined. Every $LINK[i][j]$ is incremented by m when it can contribute in building a future cube of dimension m.

Example 4 Let us follow with example 3 in order to illustrate the building of matrix *LINK*. This is: $L = \{L1 = (s_1, s_6, s_8, s_{10})\}$, $L1_0 = \{(s_1, s_8)\}$, $P_{L1,0} = \{11-\}$, $L1_1 = \{(\mathrm{sf}1s_6, s_{10})\}$ and $P_{L1,1} = \{0-0\}$. Figure 7 shows *LINK* matrix generated. For example, *LINK*[1][4] has been raised from 0 to 2 because using the complement of column 1 of C_1 as column 1 of C_1^* contributes to create the 2-cube 11- in C. In order to determine C_1^*, the highest three elements of *LINK* verifying restrictions (a) and (b) above are selected. The set of elements (1, 4), (2, 6) and (3, 5), where the first number corresponds to the row and second to the column, is one of the possible selections. This set corresponds to use the complement of column 1 of C_1 as column 1of C_1^*, the complement of column 3 of C_1 as column 2 of C_1^* and the complement of column 2 of C_1 as column 3 of C_1^*. Note that the matrix C_1^* generated is the matrix C_1^* in Fig. 6b.

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

FIGURE 7 Matrix LINK for Example 1.

Partitioning Phase

In this step *generate_partition*() takes a partial encoding problem, S, L and nv, and obtains a partition of S, $(S_0{:}S_1)$ which satisfies (i) $S_1 \cup S_0 = S$, (ii) $S_1 \cap S_0 = \varnothing$, (iii) $\mathrm{Card}(S_1) \leq 2^{nv-1}$, $\mathrm{Card}(S_0) \leq 2^{nv-1}$. These conditions guarantee that this partition can be used as an encoding dichotomy without avoiding that a minimum-length code can be derived. In the following, such a partition will be called a *valid* partition. The selection of the partition dichotomy greatly influences the quality of the solutions. A heuristic procedure has been developed which takes into account the merging-selection strategy carried out by the algorithm. The splitting process consists in applying the following rules:

Rule 1: If there is a constraint in L such that the cardinality of the set of symbols in it, Sa, is less or equal 2^{nv-1} and the cardinality of the set of symbols not in it, Sb, is less or equal 2^{nv-1}, the partition $(Sa{:}Sb)$ is chosen.

Rule 2: If there is a constraint in L such that the cardinality of the set of symbols in it, Sa, is higher than 2^{nv-1}, then the partition $(S'a{:}S'b)$ is chosen where $S'a$ is a subset of Sa with exactly 2^{nv-1} symbols in it and $S'b$ is the set of symbols not in $S'a$.

Rule 3: The partition is generated in such a way that the number of constraints whose states are divided between the two blocks (broken constraints) and whose subconstraints can be implemented by cubes of different dimension, is heuristically minimized. Only broken constraints are dealt with when obtaining C_1^*.

Rule 1 and Rule 2 take advantage of the fact that the partition dichotomy is used as an encoding dichotomy. Rule 1 guarantees the implementation of the constraint that generates the partition with a single cube. Rule 2 does not guarantee the implementation of the constraint with a minimum number of cubes but it is likely to produce cheap implementations due to the partial satisfaction achieved by the partition. Rule 3 aims at simplifying the merging selection phase and its rationale resides in the way the link matrix is built. Rule 2 is applied when Rule 1 fails to derive a valid partition, and Rule 3 when both Rule 1 and 2 fail.

Solving Problems in the Last Levels

In this step, function *resolve*() obtains a code matrix C that satisfies a maximum number of constraints given an

TABLE II Results with NOVA and COPAS

		NOVA			COPAS		
	Constraints	Cubes	Satisfied constraints	Time	Cubes	Satisfied constraints	Time
bbara	4	8	2	1.5	6	2	6.3
bbsse	5	12	3	2.2	8	3	6.3
cse	12	24	8	3.1	19	6	7.7
dk512	10	12	9	12.2	12	8	6.2
ex3	6	8	5	1.3	8	4	5.6
ex5	7	11	4	1.3	9	5	6
ex7	6	10	3	1.4	9	3	6
kirkman	25	58	9	47.4	57	9	23.9
lion9	10	10	10	1.5	10	10	6
mark1	4	6	3	17.6	5	3	9.4
opus	2	2	2	1.1	2	2	2.4
train11	11	13	10	1.7	14	8	5.8
s208	5	8	4	2.9	6	4	9.1
s420	5	8	4	2.9	6	4	9
dk16	34	43	25	136.0	51	19	23.6
donfile	24	48	8	267	47	7	29.1
ex1	11	19	8	27.8	15	7	15.8
ex2	8	10	7	2.5	12	4	12.8
keyb	33	41	26	4.8	40	25	19.3
s1	14	14	14	4.5	14	14	5.7
s1a	14	14	14	3.8	14	14	5.6
sand	7	8	6	3.8	8	6	15.3
tma	11	19	6	30.7	17	5	14.8
pma	18	30	14	90.6	33	9	24.6
styr	18	29	14	45.3	27	8	33.1
tbk	98	284	44	539	202	44	101
s820	15	17	13	12	17	13	17.5
s832	15	17	13	12.9	17	13	16.1
planet	12	12	12	39.7	13	11	26.7
s1494	29	81	16	307	61	13	68.6
s1488	29	70	17	315	58	14	68.7
scf	14	21	11	474	23	6	60.8
total		937			840		

Cube: number of product terms in a two-level implementation of input constraints; Time: time, in seconds, in a Sparcstation 10.

enough small encoding problem ($nv < bound$). Several reasons support this procedure:

(1) The differences in cube count among distinct code matrices that do not satisfy a given constraint are less significant for small problems than for large ones.
(2) Solving the subproblems is faster than carrying out the remaining splittings and subsequent mergings.
(3) Experimentally, we have found that in a high number of these problems the complete set of restricted constraints can be satisfied.

EXPERIMENTAL RESULTS

This Section shows the results obtained with COPAS, a *C* implementation of the algorithm described in this paper. Value $bound = 3$ produces the best trade-off between quality and speed. Results reported herein correspond to this choice. Experiments with a wide set of IE problems have been carried out. These problems have been generated from the IWLS'93 FSM benchmarks using a 1-hot encoding for the next state field. This is, for each FSM, we obtain a function with a symbolic input (present state field) but with all its outputs binary. These symbolic inputs are encoded using minimum code-length.

Concerning the IE problems, the figure of merit used to evaluate and compare the results obtained is the number of product terms required to implement in two level form the complete set of constraints, *cubes*. Table II summarizes the results obtained with COPAS and with a standard conventional (partial problem treated in an unified manner

TABLE III Results with ENC and COPAS

	ENC	COPAS	
	Cube	Cube	Calls
bbsse	8	8	14
cse	18	19	14
dk512	11	12	14
kirkman	58	57	14
dk16	48	51	33
donfile	39	47	33
ex1	19	15	33
s1	14	14	3
s1a	14	14	3
sand	8	8	33
styr	26	27	33
tbk	237	202	46
planet	12	13	46
scf	Out of memory	23	79
Total	512	487	

Cube: number of product terms in a two-level implementation of input constraints; calls: number of calls to the logic minimizer.

TABLE IV Results of COPAS with different merging-selection mechanisms

	$COPAS_{exhaustive}$		$COPAS_1$		COPAS	
	Cubes	Calls	Cubes	Calls	Cubes	Calls
bbara	6	48	7	0	6	14
bbsse	8	48	9	0	8	14
cse	19	48	21	0	19	14
dk512	12	48	13	0	12	14
ex3	8	48	8	0	8	14
ex5	9	48	10	0	9	14
ex7	9	48	9	0	9	14
kirkman	57	48	57	0	57	14
lion9	10	48	10	0	10	14
mark1	5	48	5	0	5	14
opus	2	48	2	0	2	1
train	14	48	16	0	14	14
s208	6	480	6	0	6	20
s420	6	480	6	0	6	20
dk16	51	480	59	0	51	33
donfile	45	480	54	0	47	33
ex1	15	480	15	0	15	33
ex2	12	480	14	0	12	33
keyb	41	480	41	0	40	46
s1	14	480	14	0	14	3
s1a	14	480	14	0	14	3
sand	8	480	8	0	8	33
tma	16	480	19	0	17	33
pma	31	480	36	0	33	33
styr	27	480	32	0	27	33
tbk	202	480	202	0	202	46
s820	17	480	17	0	17	33
s832	17	480	17	0	17	33
planet	12	4800	15	0	13	46
s1494	59	4800	67	0	61	84
s1488	55	4800	67	0	58	84
TOTAL	807		870		817	

Cube: number of product terms in a two-level implementation of input constraints; calls: number of calls to the logic minimizee.

with the complete one) tool like NOVA [7] (*i_hybrid* algorithm). Column labeled *constraints* shows the number of group constraints for each example. Also, the number of satisfied constraints and the *cubes* counts are depicted for each method. In 17 of the 32 examples the number of satisfied constraints with the two algorithms is different. Only in one of these 17 cases, COPAS produces an encoding satisfying a higher number of constraints that the one derived with NOVA. This result is concordant with the fact that NOVA aims at maximizing the number of satisfied constraint while COPAS does not. However, when the relevant figure of merit, *cubes*, is compared better results are obtained in 17 cases with COPAS. NOVA outperforms COPAS in only six examples. Adding *cubes* for the complete benchmark, 840 is obtained for COPAS and 937 for NOVA.

Table III compares the results obtained with ENC (results taken from [10] are available only for examples included in Table III) and COPAS. Only in 4 of the 13 examples, covers obtained with each algorithm differ in more than one product term. Smaller cardinalities are produced by ENC for two of the four, namely *dk*16 and *donfile*, while COPAS obtains better results for the other two examples, *ex*1 and *tbk*. In total, the sum of *cubes* for the 13 cases is slightly smaller with COPAS. Note that ENC fails to solve example *scf*.

Concerning time performance, COPAS takes around 1 min to solve every encoding problem reported in Table II except *tbk* which consumes 100 seconds of CPU time. The superiority of COPAS is significant for IE problems with many constraints as can be seen in Table II (*tbk*, *s*1494, *s*1488). We could not carry out a comparison of times with ENC because data is not available in Ref. [10]. However, there are two reasons why COPAS should be significantly

TABLE V Results of state assignment problems with different algorithms

FSM	Tp *i_hybrid*	Time ratio *i_hybrid*	Tp *io_hybrid*	Time ratio *io_hybrid*	Tp ENCORE	Tp Hyper-Place	Time ratio Hyper-Place	Tp COPAS	Time ratio COPAS
*s*208	25	1	24	7.00				17	3.15
*s*420	25	1	24	7.23				18	3.17
*dk*16$	59	1	62	5.56	58			63	0.17
donfile $	35	1	47	3.40	18			38	0.11
ex1 $	48	1	52	15.59	45			46	0.57
*ex*2 $	29	1	44	56.92	32			32	5.12
keyb $	48	1	102	10.38	51			47	4.00
*s*1 $	80	1	75	103.91	86			81	1.30
*s*1a $	76	1	73	112.30	73			71	1.50
sand $	101	1	99	39.06	100			88	4.00
tma	33	1	35	5.21				32	0.48
pma	45	1	51	3.12				47	0.27
styr	94	1	106	27.83				99	0.73
tbk $, $$	154	1	94	8.83	129	100	0.02	54	0.19
*s*820 $$	76	1	66	54.45		75	6.92	67	1.50
*s*832 $$	72	1	64	63.61		73	6.70	69	1.25
planet$	91	1	99	75.66	90			91	0.67
*s*1494 $$	139	1	120	13.76		131	3.31	128	0.22
*s*1488 $$	133	1	119	12.81		132	3.30	121	0.24
scf $	148	1	143	56.41	140			141	0.13
total	1511		1499					1320	
total $					822			752	
total $$						511		452	

faster than ENC. First, it should be clear from the explanation in the "Motivation and previous work" section and the data in column labeled calls in Table III, that ENC implies a much higher number of calls to ESPRESSO than our approach. Second, on average, COPAS dedicates around 50% of the CPU time to run ESPRESSO. In summary, COPAS and ENC perform comparably in terms of the number of cubes required to implement the face constraints but time performance of COPAS is superior.

Experimental results shown in Table IV allow the evaluation of the different novel features from COPAS. The version of the algorithm called $\text{COPAS}_{\text{exhaustive}}$ builds up the complete set of candidate C_1^* matrices and relies on ESPRESSO to select one. Experimental results on column labeled COPAS_1 correspond to a version which generates using the LINK matrix a single matrix C_1^* at each merging step, so that calls to ESPRESSO to select among several candidates are avoided. In total, the sum of the cubes for COPAS_1 is only an 8% higher than for $\text{COPAS}_{\text{exhaustive}}$. Time consumption of $\text{COPAS}_{\text{exhaustive}}$ is between one and two orders of magnitude higher than with COPAS_1. This result validates the *LINK*-matrix selection method. Finally, results obtained with COPAS are repeated in this table so that the readers can easily appreciate the good quality-speed trade-off that it is achieved by the intermediate approach that has been implemented in the proposed tool. We remember that this intermediate approach consists in generating a reduced set of candidate matrices and using ESPRESSO to select among them.

Finally, in order to show an application of the algorithm, results for the state assignment problem are also given. Table V compares NOVA *i_hybrid*, NOVA *io_hybrid*, ENC, Hyper_Place [14] and COPAS for the state assignment problem. The number of product terms required to implement the combinational component of an IWLS'93 FSM, *tp*, is shown in this table. Also, execution times, normalized to those of NOVA, *i_hybrid* are given. COPAS compares favorably to all of them. When comparing time performance, note that COPAS' times are less than 2 min for all the machines in the table.

CONCLUSIONS

An encoding technique specifically targeting the partial input encoding problem has been developed. It is based on a recursive splitting and merging strategy. Main features of the new approach include: (1) the development of a novel merging model which guarantees an implementation of the constraints at least as efficient as in the lower levels, and the use of the partition phase with the aim of satisfying constraints; (2) a procedure for determining a set of candidate encodings for which ESPRESSO is called, avoiding intensive use of logic minimization, and (3) a strongly coupling between the splitting and merging phases. Our work shows that the splitting-merging strategy is actually a practical and competitive alternative for the partial input encoding problem, as the limitations of ENC, which has been reported to produce very good results for different synthesis tasks, have been overcome in the new algorithm, without degrading the quality of the solutions.

References

[1] Ashar, P., Devadas, S. and Newton, A.R. (1992) *Sequential Logic Synthesis* (Kluwer Academic Publishers, Dordrecht).

[2] de Micheli, G., Brayton, R.K. and Sangiovanni-Vincentelli, A.L. (1985) "Optimal state assignment of finite state machines", *IEEE Transactions on CAD* **CAD-4**, 269–285.

[3] Rudell, R. and Sangiovanni-Vincentelli, A.L. (1987) "Multiple-valued minimization for PLA optimization", *IEEE Transactions on CAD* **CAD-6**, 727–751.

[4] de Micheli, G. (1986) "Symbolic design of combinational and sequential logic circuits implemented by two-level macros", *IEEE Transactions on CAD* **5**, 597–616.

[5] Devadas, S., Wang, A.R., Newton, A.R. and Sangiovanni-Vncentelli, A. (1989) "Boolean descomposition in multilevel logic optimization", *IEEE Journal of Solid State Circuits* **24**(2), 1.

[6] B. Lin and A.R. Newton (1989). "A generalized approach to the constrained cubical embedding problem". In *Proc. International Conference on Computer Design*, VLSI computers and Processors, October 1989, pp. 400–404.

[7] Villay, T. and Sangiovanni-Vincentelli, A.L. (1990) "NOVA assignment of finite state machines for optimal two-level logic implementation", *IEEE Transactions on CAD* **CAD-9**(9), 905–923.

[8] Shi, C.J. and Brzozowski, J.A. (1993) "An efficient algorithm for constrained encoding and its applications", *IEEE Transactions on CAD* **12**(12), 1813–1826.

[9] Malik, S., Lavagno, L., Brayton, R.K. and Sangiovanni-Vincentelli, A. (1992) "Symbolic minimization of multilevel logic and the input encoding problem", *IEEE Transactions on CAD* **11**(7), 825–843.

[10] Saldanha, A., Villa, T., Brayton, R.K. and Sangiovanni-Vincentelli, A.L. (1994) "Satisfaction of input and output encoding constraints", *IEEE Transactions on CAD* **13**(5), 589–602.

[11] R. Murgai, R.K. Brayton, A.L. and Sangiovanni-Vincentelli (1994). "Optimun functional de-composition using encoding". In *Proc. 31st ACM/IEEE Design Automation Conference*, pp 408–414.

[12] Brayton, R.K., Hachtel, G.D., McMulleny, C. and Sangiovanni-Vincentelli, A.L. (1984) *Logic minimization algorithms for VLSI synthesis* (Kluwer Academic Publishers, Boston, MA).

[13] Yang, S. and Cieselski, M.J. (1991) "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization", *IEEE Transactions on CAD* **10**(1), 4–12.

[14] Liu, S., Pedram, M. and Despain, A.M. (1997) "State assignment based on two-dimensional placement and hypercube mapping", *Integration, The VLSI Journal*(24), 101–118.

APPENDIX

THEOREM 1 If using $C_0(C_1)$ a cube $c \subset \{0,1\}^{nv-1}$ contains the codes of a subset of symbols $A \subset S_0(S_1)$ and does not intersect the code of any other symbol in $S_0(S_1)$, then for C, there is a cube $c' \subset \{0,1\}^{nv}$ which contains the codes of every symbol in A, and does not intersect the code of any other symbol in S ($S_0 \cup S_1$).

Proof For C, the codes of the symbols in $A \subset S_0(S_1)$ are included in a cube c' obtained as the conjunction of c and a literal associated with the new encoding bit when dealing with partition 0, and as the conjunction of a literal associated with the new bit and the cube c'' obtained from c complementing and or renaming variables when dealing with partition 1. Clearly, c' does not include the code of any symbol in $S_1(S_0)$ as there is a 0(1) in last bit position of

any code in c' and a 1(0) in last bit position of the codes of the symbols in $S_1(S_0)$. In addition c' does not intersect the code of any other symbol in $S_0(S_1)$ that does not belong to A as the transformations applied to C_1 guarantee that c'' contains the codes in A but not any other. □

COROLLARY The number of cubes, #cubes, required to implement the constraints in L using C is less or equal to $\#\text{cubes}_0 + \#\text{cubes}_1$ where $\#\text{cubes}_0$ ($\#\text{cubes}_1$) is the number of cubes required to implement the constraints in $L_0(L_1)$ using $C_0(C_1)$.

Authors' Biographies

Manuel Martinez-Perez received the B.S. degree in Electronics from the University of Seville, Spain in 1991. Since 1994 he is in the Institute of Microelectronics at Seville (IMSE) where he is currently working toward a PhD Degree. His main research interest is logic synthesis.

Maria J. Avedillo joined the Department of Electronics and Electromagnetism at the University of Seville in 1988 as Assistant Professor, and obtained a PhD degree in 1992. Since 1995 she is Associate Professor in that Department. In 1989 she became researcher at the Department of Analog Design of the National Microelectronics Center (CNM), now Institute of Microelectronics at Seville (IMSE). She has participated in several research projects financed by the Spanish CICYT and in ESPRIT Projects. She won the KELVIN price of "The Council of the Institution of Electrical Engineers" for two articles published in 1994. Her current research interests include design of threshold logic circuits, development of CAD tools for FSM synthesis and design for testability.

José M. Quintana joined the Department of Electronics and Electromagnetism at the University of Seville in 1983 as Assistant Professor, and obtained a PhD degree in 1987. Since 1990 he is Associate Professor in that Department. In 1989 he became researcher at the Department of Analog Design of the National Microelectronics Center (CNM), now Institute of Microelectronics at Seville (IMSE). He has participated in several research projects financed by the Spanish CICYT and in the ESPRIT Projects ADCIS and AD-2000. He won the KELVIN price of "The Council of the Institution of Electrical Engineers" for two articles published in 1994. His current research interests include design of threshold logic circuits, computer arithmetic and development of CAD tools for FSM synthesis.

José L. Huertas received a PhD degree in 1973 from the University of Seville. Since 1971 he has been with the department of electronics and electromagnetism at the University of Seville, where he is a Professor. Since 1989 he has been Director of the Department of Analog Design of the National Microelectronics Center (CNM), now Institute of Microelectronics at Seville (IMSE). He has participated and led many research projects financed by CICYT, COMMET and SPRITE programs of the European Community. He is scientific advisor of a number of international journals and has won several scientific prizes. He is a Senior member of the Institute of Electrical and Electronic Engineers. His current research interests include design and test of analog/digital integrated circuits, computer-aided IC analysis and design, fuzzy logic, nonlinear microelectronics and neural networks.