ADVANCES IN ENCODING FOR LOGIC SYNTHESIS

by

Tiziano Villa, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M95/19

22 March 1995

ADVANCES IN ENCODING FOR LOGIC SYNTHESIS

by

Tiziano Villa, Luciano Lavagno, and Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M95/19

22 March 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Advances in encoding for logic synthesis

Tiziano Villa Luciano Lavagno Alberto L. Sangiovanni-Vincentelli

Dept. of EECS
University of California, Berkeley

To appear in **Digital Logic Analysis and Design**
Ablex Publ. Co., 1995

Abstract

In this paper we survey the recent literature on encoding problems arising from logic synthesis of combinational and synchronous sequential circuits. Encoding problems consist of assigning Boolean codes to input and output symbolic variables, so that a cost function measuring the optimality of a two-level or multi-level implementation is minimized. A successful paradigm involves optimizing the symbolic representation (symbolic minimization), and then transforming the optimized symbolic description into a compatible two-valued representation, by satisfying encoding constraints (bit-wise logic relations) imposed on the binary codes that replace the symbols. The input encoding problem is well understood. Efficient algorithms are available for two-level implementations and are under development for multi-level implementations. The output encoding problem has seen important contributions, but more work needs to be done and efficient algorithms developed.

1 Introduction

Descriptions of logic systems use variables that can take a finite number of values. They are called symbolic variables, or, in logic jargon, multiple-valued variables. It is possible to define and optimize multiple-valued logic functions and some optimization tools are available for that purpose [26]. However, most logic circuits implementations are restricted to binary values. Therefore, symbolic (multiple-valued) descriptions of logic functions at the structural level must be transformed into Boolean (two-valued) representations by replacing each symbolic entry by Boolean vectors. An assignment of Boolean vectors to symbolic entries is called an encoding.

The optimization of logic functions performed on the Boolean representation depends heavily on the encoding chosen to represent the symbolic variables.

The cost function that estimates the optimality of an encoding depends on the target implementation: two-level or multi-level. Two-level implementations optimize the number of product-terms or the area of a programmable logic array (PLA). Multi-level implementations optimize the number of literals of a technology-independent representation of the logic. Optimality may be based on more complex cost functions that take into account other criteria, like testability. It may even be the case that area becomes a secondary optimization objective as in the case of state assignment for asynchronous sequential circuits, where the main concern is the correctness of the behavior of the encoded circuit.

In this paper we concentrate on encoding problem arising from implementation of combinational and synchronous sequential circuits. Some of the solutions presented may be extendible to handle encoding problems of asynchronous sequential design.

The following optimal encoding problems may be defined.

(A) Optimal encoding of inputs of a logic function. A problem in class A is the optimal assignment of *opcodes* for a microprocessor.

(B) Optimal encoding of outputs of a logic function.

(C) Optimal encoding of both inputs and outputs (or some inputs and some outputs) of a logic function.

(D) Optimal encoding of both inputs and outputs (or some inputs and some outputs) of a logic function, where the encoding of the inputs (or some inputs) is the same as the encoding of the outputs (or some outputs). Encoding the states of a finite state machine (FSM) is a problem in class D since the state variables appear both as input (present state) and output (next state) variables. Another problem in class D is the encoding of the signals connecting two (or more) combinational circuits.

Some approaches to the encoding problem, such as those in [1] and [9], use heuristics to direct the encoding, but they cannot predict the relation between the symbolic description and the encoded optimized logic.

Another paradigm, [7], involves optimizing the symbolic representation (symbolic minimization), and then transforming the optimized symbolic description into a compatible two-valued representation, by satisfying encoding constraints (bit-wise logic relations) imposed on the binary codes that replace the symbols. This approach guarantees an upper bound on the size of the implemented machine provided all the encoding constraints are satisfied.

Encoding via symbolic minimization may be considered a three step process. The first phase consists of multiple-valued optimization. The second step is to extract constraints on the codes of the symbolic variables, which, if satisfied, guarantee the existence of a compatible Boolean implementation. The third step is assigning to the symbols codes of minimum length that satisfy these constraints, if the latter imply a set of non-contradictory bit-wise logic relations.

When the target implementation is two-level logic, the first step may consist of one or more calls [7, 6] to a multiple-valued minimizer [26], after the symbolic variables have been denoted using the positional cube notation [35, 26]. An exact algorithm to handle symbolic input and output variables was recently provided in [10]. Then constraints are extracted and a constraints satisfaction problem is set up. In the case of a multi-level logic implementation constraints are generated after performing multiple-valued algebraic factoring [21].

A variety of other applications may also generate similar constraints satisfaction problems, as in the case of synthesis for sequential testability [8], and optimal re-encoding and decomposition of PLA's [27, 11, 42, 31]. Given a PLA, it is possible to group the inputs into pairs and replace the input buffers with two-bit decoders to yield a bit-paired PLA with the same number of columns and no more product-terms than the original PLA [30] In the more general case, a single PLA is decomposed into two levels of cascaded PLA's. A subset of inputs is selected such that the cardinality of the multiple-valued cover, produced by representing all combinations of these inputs as different values of a single multiple-valued variable, is smaller than the cardinality of the original binary cover. The encoding problem consists of finding the codes of the signals between the PLA's, so that the constraints imposed by the multiple-valued cover are satisfied.

Using the paradigm of symbolic minimization followed by constraints satisfaction, the most common types of constraints that may be generated [7, 6, 10, 29, 28] are three. The first type, generated by the input variables, are *face-embedding* constraints. The two types generated by the output variables are

dominance and *disjunctive* constraints. Each face-embedding constraint specifies that a set of symbols is to be assigned to one *face* of a binary n-dimensional cube and no other symbol should be in that same face. Dominance constraints require that the code of a state must bit-wise cover the code of another state. Disjunctive constraints specify that the code of a state be expressed as the bit-wise disjunction (*oring*) of two or more other symbols.

In this paper we will survey important ideas and algorithms that marked the recent progress in this area, with emphasis towards the contributions made by our research group. When the target logic is two-level, the achievements have been noticeable. In that case, the input encoding problem has been completely understood [7] and efficient algorithms for satisfaction of input constraints are available [38, 28]. Important contributions have been made for the input and output encoding problem [6, 38, 10]. Exact solutions [10] are still very expensive for non-trivial examples and more work to find efficiently high-quality solutions is coming [29]. Efficient algorithms for satisfaction of input and output constraints are being developed [28].

When the target logic is multi-level the problem is much harder [9, 19, 14]. An important theoretical contribution for input encoding has been made by [21], but more needs to be done to model the different optimization operators available in a multi-level environment, and to take into account the presence of symbolic output variables.

The organization of this paper is as follows. In Section 2 we introduce some basic definitions. In Section 3 we present the encoding problem for optimal two-level implementations. In Section 4 we introduce the encoding problem for optimal multi-level implementations. In Sections 5 and 6 algorithms for the solutions of only input and mixed input-output constraints, respectively, are presented. In Section 7 some experimental results of programs implementing successful approaches are given, while conclusions are drawn in Section 8.

2 Definitions

2.1 Finite state machines

For definitions of two-level logic minimization we refer to [3]. Here we describe the connection between a FSM tabular description and its interpretation as a multiple-valued logic function.

A logic function may have multiple-valued (called also symbolic) input variables and symbolic output variables. A symbolic input or output variable takes on symbolic values. FSM's can be represented by state transition tables. State transitions tables have as many rows as transitions in the FSM. The rows of the table are divided into four fields corresponding to the primary inputs, present states, next states and primary outputs of the FSM. Each field is a string of characters. The primary inputs may be in boolean or symbolic form. Note that the input and output patterns may contain don't care entries. A state transition table defines a symbolic cover of the combinational component of a FSM. The rows of the state transition table are called symbolic implicants of the symbolic cover. The symbolic cover representation may be seen as a multiple-valued logic representation, where each present state mnemonic is one of the possible values of a present-state multiple-valued variable. A similar identification holds for the next states (and the proper inputs and outputs, if they are symbolic).

2.2 Boolean networks

For definitions of multi-level logic minimization we refer to [4]. Here we repeat the most basic ones.

A *boolean network* is a directed acyclic graph, where each node n_i is associated with:

1. a variable y_i and

2. a representation of a logic function f_i.

An arc connects node j with node i if the representation of f_i depends on y_j (e.g. in $f_3 = y_0 + y_1\overline{y_2}$, there are arcs from n_0, n_1 and n_2 to n_3).

Each node function can either be represented in sum-of-products form or in *factored form*. A factored form is recursively defined as:

1. a literal (i.e. a boolean variable or its complement),

2. a sum of factored forms,

3. a product of factored forms.

E.g. $f = ab(c + d) + cd$ and $f = abc + (ab + c)d$ are both factored forms of the function whose sum-of-products can be $f = abc + abd + cd$. Neither factored forms nor sum-of-product forms are a canonical representation of boolean functions.

The cost of a boolean network is typically estimated as the sum over all nodes of the number of literals in a minimum (i.e. one with a least number of literals) factored form of the node function. This cost estimation has a good correlation with the cost of an implementation of the network in various technologies, e.g. standard cells or CMOS gate matrix.

3 Encoding for two-level implementations

3.1 Early work

Among the first to define input and output encoding problems for combinational networks were [5] and [24]. The former based his theory of input encoding on partitions and set systems. The latter tried to minimize the variable dependency of the output functions and studied the problem of the minimum number of variables required for a good encoding.

There is a rich early literature on the state assignment of FSM's, although it has been rarely cast as a general encoding problem. In [1], the state assignment problem was formulated as a graph embedding problem, where a graph represents adjacency relations between the codes of the states, to be preserved by a subgraph isomorphism on the encoding cube. The objective was to minimize the number of gates of the final implementation. Others, as [15, 34, 16, 12], proposed algebraic methods based on the algebra of partitions and on the criterion of reduced dependency. These approaches suffered from a weak connection with the logic optimization steps after the encoding. More recent approaches [32, 33] rely on local optimization rules defined on a control flowgraph. There rules are expressed as constraints on the codes of the internal variables and an encoding algorithm tries to satisfy most of these constraints. Unate state assignments to guarantee testability by construction were proposed first in [36]. The logic to compute the outputs and the encoding of the next state is said to be unate in a given state variable, if the output and next state functions can be expressed as sums of products where the given variable appears either uncomplemented or complemented, but not both. In [25] a case was made for a variation of unate encoding called half-hot encoding that may allow sometimes savings in the number of columns of the encoded PLA. Half-hot encodings have exactly half the total number of state variables equal 1. The penalty on the number of necessary product terms was not addressed.

3.2 Multiple-valued minimization

Advances in the state assignment problem, reported in [23, 3, 7], made a key connection to multiple-valued logic minimization, by representing the states of a FSM as the set of possible values of a single multiple-valued variable. A multiple-valued minimizer, such as [26], can be invoked on the symbolic representation of the FSM. This can be done by representing the symbolic variables using the positional cube notation [35, 26]. The effect of multiple-valued logic minimization is to group together the states that are mapped by some input into the same next-state and assert the same output. To get a compatible boolean representation, one must assign each of the groups of states obtained by MV minimization, (called face or input constraints) to subcubes of a boolean k-cube, for a minimum k, in a way that each subcube contains all and only all the codes of the states included in the face constraint. This problem is called face embedding problem. The state table of a FSM and its 1-hot encoded representation is shown in Figure 1.

It is worth mentioning that the face constraints obtained through straightforward symbolic minimization are sufficient, but not necessary to find a two-valued implementation matching the upper bound of the multi-valued minimized cover. As it was already pointed out in [6], for each implicant of a minimal (or minimum) multi-valued cover, one can compute an *expanded implicant*, whose literals have maximal (maximum) cardinality and a *reduced implicant* whose literals have minimal (minimum) cardinality. By bit-wise comparing the corresponding expanded and reduced implicant, one gets *don't cares* in the input constraint, namely, in the bit positions where the expanded implicant has a 1 and the reduced implicant has a 0. The face embedding problem with *don't cares* becomes one of finding a cube of minimum dimension k, where, for every face constraint, one can assign the states asserted to vertices of a subcube that does not include any state not asserted, whereas the *don't care* states can be put inside or outside of that subcube. One can build examples where the presence of *don't cares* allows to satisfy the input constraints in a cube of smaller dimension, than it would be possible otherwise. In Figure 2 the expanded and reduced minimized multi-valued covers of the FSM of Figure 1 are shown. Figure 3 shows the expanded and reduced implicants of the same FSM and the generation of the implied *don't care* face constraints.

3.3 Symbolic minimization

Any encoding problem, where the symbolic variables only appear in the input part, can be solved by setting up a multiple-valued minimization followed by satisfaction of the induced face constraints. However, the problem of state assignment of FMS's is only partially solved by this scheme, because the encoding of the symbolic output variables is not taken into account (e.g. the next state variable). Simple multiple-valued minimization disjointly minimizes each of the on-sets of the symbolic output functions, and therefore disregards the sharing among the different output functions taking often place when they are implemented by two-valued logic. We will see now more powerful schemes to deal with both input and output encoding.

In [6, 38] a new scheme was proposed, called symbolic minimization. Symbolic minimization was introduced to exploit bit-wise dominance relations between the binary codes assigned to different values of a symbolic output variable. The fact is that the input cubes of a dominating code can be used as don't cares for covering the input cubes of a dominated code. The core of the approach is a procedure to find useful dominance (called also covering) constraints between the codes of output states. The translation of a cover obtained by symbolic minimization into a compatible boolean representation defines simultaneously a face embedding problem and an output dominance satisfaction problem.

00	st0	st0	0	00	10000000000	10000000000	0
10	st0	st1	-	10	10000000000	01000000000	-
01	st0	st2	-	01	10000000000	00100000000	-
10	st1	st1	1	10	01000000000	01000000000	1
00	st1	st3	1	00	01000000000	00010000000	1
11	st1	st5	1	11	01000000000	00000100000	1
01	st2	st2	1	01	00100000000	00100000000	1
00	st2	st7	1	00	00100000000	00000001000	1
11	st2	st9	1	11	00100000000	00000000010	1
00	st3	st3	1	00	00010000000	00010000000	1
01	st3	st4	1	01	00010000000	00001000000	1
01	st4	st4	1	01	00001000000	00001000000	1
00	st4	st0	-	00	00001000000	10000000000	-
11	st5	st5	1	11	00000100000	00000100000	1
01	st5	st6	1	01	00000100000	00000010000	1
01	st6	st6	1	01	00000010000	00000010000	1
00	st6	st0	-	00	00000010000	10000000000	-
00	st7	st7	1	00	00000001000	00000001000	1
10	st7	st8	1	10	00000001000	00000000100	1
10	st8	st8	1	10	00000000100	00000000100	1
00	st8	st0	-	00	00000000100	10000000000	-
11	st9	st9	1	11	00000000010	00000000010	1
10	st9	st10	1	10	00000000010	00000000001	1
10	st10	st10	1	10	00000000001	00000000001	1
00	st10	st0	-	00	00000000001	10000000000	-

Figure 1: Initial and 1-hot encoded covers of FSM-1

01	01010111011	000000010001	01	00010001000	000000010001
01	01001110111	000000001001	01	00001000100	000000001001
10	00111101110	000000000101	10	00000100010	000000000101
10	00111011101	000000000011	10	00000010001	000000000011
00	01011010000	000100000001	00	01010000000	000100000001
11	11011101111	000010000001	11	01001000000	000010000001
00	00101110000	000001000001	00	00100100000	000001000001
11	10110111111	000000100001	11	00100010000	000000100001
10	11111001100	010000000001	10	11000000000	010000000001
01	11100110011	001000000001	01	10100000000	001000000001
00	10001011111	100000000000	00	10000001111	100000000000

Figure 2: Expanded and reduced minimized covers of FSM-1

01010111011	00010001000	0-010--10--
01001110111	00001000100	0-001--01--
00111101110	00000100010	00---10--10
00111011101	00000010001	00---01--01
01011010000	01010000000	0101-0-0000
11011101111	01001000000	-10-1-0----
00101110000	00100100000	0010-1-0000
10110111111	00100010000	-01-0-1----
11111001100	11000000000	11---00--00
11100110011	10100000000	1-100--00--
10001011111	10000001111	1000-0-1111

Figure 3: Expanded and reduced implicants and don't care face constraints of FSM-1

(1)	10	st1	st2	11	(1')	-0	st1,st2	st2	11
(2)	00	st2	st2	11	(2')	0-	st2,st3	st2	00
(3)	01	st2	st2	00	(3')	10	st2,st3	st1	11
(4)	00	st3	st2	00	(4')	00	st1	st1	- -
(5)	10	st2	st1	11	(5')	01	st3	st0	00
(6)	10	st3	st1	11	(6')	11	st1,st0	st1	10
(7)	00	st1	st1	- -	(7')	11	st0,st3	st3	01
(8)	01	st3	st0	00					
(9)	11	st1	st1	10					
(10)	11	st3	st3	01					
(11)	11	st0	st0	11					

Figure 4: Covers of FSM-2 before and after symbolic minimization

Notice that any output encoding problem can be solved by symbolic minimization. Symbolic minimization was applied also in [27], where a particular form of PLA partitioning is examined, by which the outputs are encoded to create a reduced PLA that is cascaded with a decoder. However, to mimic the full power of two-valued logic minimization, another fact must be taken into account. When the code of a symbolic output is the bit-wise disjunction of the codes of two or more other symbolic outputs, the on-set of the former can be minimized by using the on-sets of the latter outputs, by redistributing some cubes. An extended scheme of symbolic minimization can therefore be defined to find useful dominance and disjunctive relations between the codes of the symbolic outputs. This is currently investigated in [29]. The translation of a cover obtained by extended symbolic minimization into a compatible boolean representation induces a face embedding, output dominance and output disjunction satisfaction problem.

In Figure 4, we show the initial description of a FSM and an equivalent symbolic cover returned by an extended symbolic minimization procedure.

The reduced cover is equivalent to the original one if we impose the following constraints on the codes of the states.

(1")	-0	0-	00	11
(2")	0-	-0	00	00
(3")	10	-0	01	11
(4")	00	01	01	- -
(5")	01	10	11	00
(6")	11	-1	01	10
(7")	11	1-	10	01

Figure 5: Encoded cover of FSM-2

Product terms (1'), (3') and (4') are consistent with the original product terms (5) and (7) if we impose $code(st1) > code(st2)$. In a similar way, product terms (2') and (5') are consistent with the original product term (8) if we impose $code(st0) > code(st2)$. The product terms (1') and (2') yield also the face constraints $face(st1, st2)$ and $face(st2, st3)$, meaning that the codes of $st1$ and $st2$ ($st2$ and $st3$) span a face of a cube, to which the code of no other state can be assigned. The previous face and dominance constraints together allow to represent the four original transitions (1), (2), (3), (4) by two product terms (1') and (2').

Product term (3') is equivalent to the original transitions (5) and (6) and yields the face constraint $face(st2, st3)$. This saving is due to a pure input encoding join effect.

Finally the product terms (6'), (7') represent the original transitions (9), (10) and (11). The next state of (11) is $st0$, that does not appear in (6') and (7'). But, if we impose the disjunctive constraint $code(st0) = code(st1) V code(st3)$, i.e. we force the code of $st0$ to be the bit-wise *or* of the codes of $st1$ and $st3$, we can redistribute the transition (11) between the product terms (6') and (7'). The product terms (6') and (7') yield has also the face constraints $face(st1, st0)$ and $face(st0, st3)$; together with the previous disjunctive constraint they allow the redistribution of transition (11).

We point out that if we perform a simple MV minimization on the original description we save only one product term, by the join effect taking place in transition (3').

An encoding satisfying all constraints can be found and the minimum code length is two. A solution is given by $st_0 = 11, st_1 = 01, st_2 = 00, st_3 = 10$. If we replace the states by the codes in the minimized symbolic cover, we obtain an equivalent Boolean representation that can be implemented with a PLA, as shown in Figure 5. Note that we replace the groups of states in the present state field with the unique face assigned to them and that product term (2") is not needed, because it asserts only zero outputs. Therefore the final cover has only six product terms.

3.4 Exact encoding with generalized prime implicants

Another procedure for output encoding has been reported in [10]. It guarantees an exact solution and it is based upon a frame different from that used in symbolic minimization and its natural extensions. A notion of generalized prime implicants (GPI's), as an extension of prime implicants defined in [22], is introduced, and appropriate rules of cancellation are given. Each GPI carries a tag with some output symbols. If a GPI is accepted in a cover, it asserts as output the intersection (bit-wise *and*) of the codes of the symbols in the tag. To maintain functionality, the coded output asserted by each minterm must be equal to the bit-wise *or* of the outputs asserted by each selected GPI covering that minterm. Given a

1101	out1	1101	(out1)	110-	(out1,out2)	110-	01
1100	out2	1100	(out2)	11-1	(out1,out3)	11-1	10
1111	out3	1111	(out3)	000-	(out4)	000-	00
0000	out4	110-	(out1,out2)				
0001	out4	11-1	(out1,out3)				
		000-	(out4)				

Figure 6: Initial cover, GPI's, encodable selection of GPI's and encoded cover of OUT-1

selection of GPI's, each minterm yields a boolean equation constraining the codes of the symbolic values. If an encoding can be found that satisfies the system of boolean equations, then the selection of GPI's is encodable. We will explain with some more detail in Section 6.2 the notion of encodability of GPI's. Given all the GPI's, one must select a minimum subset of them that covers all the *minterms* and forms an encodable cover. This can be achieved by solving repeated covering problems that return minimum covers of increasing cardinality, until an encodable cover is found, i.e. the minimum cover that is also encodable. An encodable cover yields a compatible set of face, dominance and disjunctive constraints on the codes of the symbols. In order to minimize the area of a two-level implementation, one must find an encoding satisfying all relations with a minimum number of bits. In this way the problem is reduced to the familiar paradigm of constraints satisfaction. Figure 6 shows output encoding based on GPI's with a simple example taken from [10].

4 Encoding for multi-level implementation

Automatic multi-level logic synthesis programs are now available to the IC designer ([13], [4], [2]), and sometimes a PLA implementation of the circuit does not satisfy the area/timing specifications.

A two-level encoding program, such as those described in the previous sections, can often give a very good result when multi-level realization is required, but in order to get the maximum advantages from multi-level logic synthesis we need a specialized approach.

This section describes such approaches, giving some information on the relative strengths and weaknesses.

There are two main classes of multi-level encoding algorithms:

1. estimation-based algorithms, that define a distance measure between symbols, such that if "close" symbols are assigned "close" (in terms of Hamming distance) codes it is likely that multi-level synthesis will give good results. Programs such as *mustang* [9] and *jedi* [19] belong to this class.

2. synthesis-based algorithms, that use the result of a multi-level optimization on the unencoded or one-hot encoded symbolic cover to drive the encoding process. Programs such as *mis-MV* [21, 17] and *muse* [14] belong to this class.

4.1 Mustang

Mustang uses the state transition graph to assign a weight to each pair of symbols. This weight measures the desirability of giving the two symbols codes that are "as close as possible".

Mustang has two distinct algorithms to assign the weights, one of them ("fanout oriented") takes into account the next state symbols, while the other one ("fanin oriented") takes into account the present state symbols. Such a pair of algorithms is common to most multi-level encoding programs, namely *mustang*, *jedi* and *muse*.

The fanout oriented algorithm is as follows:

1. For each output o build a set O^o of the present states where o can be asserted. Each state p in the set has a weight OW_p^o that is equal to the number of times that o is asserted in p.

2. For each next state n build a set N^n of the present states that have n as next state. Again each state p in the set has a weight NW_p^n that is equal to the number of times that n is a next state of p (each cube under which a transition can happen appears as a separate edge in the state transition graph) multiplied by the number of state bits (the number of output bits that the next state symbol generates).

3. For each pair of states k, l let the weight of the edge joining them in the weight graph be $\sum_{n \in S} NW_k^n \times NW_l^n + \sum_{o \in O} OW_k^o \times OW_l^o$.

This algorithm gives a high weight to present state pairs that have a high degree of similarity, if similarity is measured as the number of common outputs asserted by the pair.

The fanin oriented algorithm (almost symmetric with the previous one) is as follows:

1. For each input i build a set ON^i of the next states that can be reached when i is 1, and a set OFF^i of the next states that can be reached when i is 0. Each state n in ON^i has a weight ONW_n^i that is equal to the number of times that n can be reached when i is 1, and each state n in OFF^i has a weight $OFFW_n^i$ that is equal to the number of times that n can be reached when i is 0.

2. For each present state p build a set P^p of the next states that have p as present state. Again each state n in the set has a weight PW_n^p that is equal to the number of times that n is a next state of p multiplied by the number of state bits.

3. For each pair of states k, l let the weight of the edge joining them in the weight graph be $\sum_{p \in S} PW_k^p \times PW_l^p + \sum_{i \in I} ONW_k^i \times ONW_l^i + OFFW_k^i \times OFFW_l^i$.

This algorithm tries to maximize the number of common cubes in the next state function, since next states that have similar functions will be assigned close codes.

The embedding algorithm identifies clusters of nodes (states) that are joined by maximal weight edges, and greedily assigns to them minimally distant codes. It tries to minimize the sum over all pairs of symbols of the product of the weighted distance among the codes.

The major limitation of *mustang* is that its heuristics are only distantly related with the final minimization objective. It also models only common cube extraction, among all possible multiple-level optimization operations ([4]).

4.2 Jedi

Jedi is aimed at generic symbol encoding rather than at state assignment, and it applies a set of heuristics that is similar to *mustang*'s to define a set of weights among pairs of symbols. Then it uses either a simulated annealing algorithm or a greedy assignment algorithm to perform the embedding.

The proximity of two cubes in a symbolic cover is defined as the number of non-empty literals in the intersection of the cubes. It is the "opposite" of the Hamming distance between two cubes, defined as the number of empty literals in their intersection. For example, cubes $ab\bar{c}$ and $\bar{c}d\bar{e}$ have proximity 4, because their intersection has four non-empty literals ($a, b, \bar{d}$ and e), and distance 1, because their intersection has an empty literal ($c \cap \bar{c}$).

Each pair of symbols (s_i, s_j) has a weight that is the sum over all pairs of cubes in the two-level symbolic cover where s_i appears in one cube and s_j appears in the second one of the proximity between the two cubes.

The cost function of the simulated annealing algorithm is the sum over all symbol pairs of the weighted distance among the codes.

The greedy embedding algorithm chooses at each step the symbol that has the strongest weight connection with already assigned symbols, and assigns to it a code that minimizes the above cost function.

4.3 Muse

Muse uses a multi-level representation of the finite state machine to derive the set of weights that are used in the encoding problem.

Its algorithm is as follows:

1. encode symbolic inputs and outputs with one-hot codes.
2. use *misII* ([4]) to generate an optimized boolean network.
3. compute a weight for each symbol pair (see below).
4. use a greedy embedding algorithm trying to minimize the sum over all state pairs of the weighted distance among the codes.
5. encode the symbolic cover, and run *misII* again.

The weight assignment algorithm examines each node function (in sum-of-product form) to see if any of the following cases applies (S_i denotes a state symbol, s_i denotes the corresponding one-hot present state variable, other variables denote primary inputs):

1. $s_1ab + s_2ab + \ldots$: if S_1 and S_2 are assigned adjacent codes, then the cubes can be simplified to a single cube, and we obtain a saving in the encoded network cost.
2. $s_1ab + s_2abc + \ldots$: if S_1 and S_2 are assigned adjacent codes, then the cubes can be simplified (even though they will remain distinct cubes, due to the appearance of c only in the second one) and a common cube (the common state bits and ab) can be extracted. For example, if S_1 is encoded as $c_0\overline{c_1}\overline{c_2}$ and S_2 is encoded as $c_0\overline{c_1}c_2$, the expression above can be simplified as $c_0\overline{c_1}ab + c_0\overline{c_1}c_2abc$.
3. $s_1abc + s_2abd + \ldots$: same as above, but only a common cube (the common state bits and ab) can be extracted.

For each occurrence of the above cases the weight of the state pair is increased by an amount that is proportional to the estimated gain if the two states are assigned adjacent codes. For example, if abc is extracted from $f = abcd$, $g = abce$, (cost 8 literals) then we obtain $f = hd$, $g = he$, $h = abc$ (cost 7 literals), and the gain obtained extracting h is 1.

Each gain is also multiplied by the number of distinct paths from the node to a network output. This heuristic gives a higher gain to common subexpressions that are used in many places in the network, so that their extraction gives a high reduction in the network cost. If the codes in the pair are assigned adjacent codes, then hopefully *misII* will be able to extract again useful subexpressions after the encoding.

The algorithm described above takes into account only present state symbols. Another heuristic algorithm is used to estimate the "similarity" among the next state functions. This "next-state oriented" algorithm adds to the weight of each pair of states the gain of common subexpressions that can be extracted from the functions generating that pair of next states in the one-hot encoded network. For example, if n_i denotes a one-hot next state variable and N_i the corresponding state symbol, $n_1 = abcd$ and $n_2 = abce$ have a common subexpression abc of gain 1 (see above), so the weight of the (N_1, N_2) pair is incremented by 1 due to this subexpression.

The embedding algorithm, using the weights computed above, chooses the unencoded state that has a maximum weight connection with the already encoded states and assigns to it a code that has the minimum weighted distance from the already encoded states.

Muse uses a cost function that is a closer representation of reality with respect to *mustang* and *jedi*, but there is no guarantee that the optimizations performed on the one-hot encoded network are the best ones for all possible encodings, and that *misII* will choose to perform the same optimizations when it is run on the encoded network.

4.4 Mis-MV

In order to have a satisfactory solution of the multi-level encoding problem we must have a closer view of the real cost function, the number of literals in the encoded network. The weight matrix is rather far from giving a complete picture of what happens to this cost function whenever an encoding decision is made.

Following the pattern outlined in the previous sections for the two-level case, we should perform a multi-level symbolic minimization, and derive constraints that, if satisfied, can guarantee some degree of minimality of the encoded network.

Mis-MV, unlike the previous programs, performs a full multi-level multiple-valued minimization of a network with a symbolic input. Its algorithms are an extension to the multiple-valued case of those used by *misII* (the interested reader is referred to [21, 17] for a detailed explanation of these algorithms).

Its overall strategy is as follows:

1. read the symbolic cover. The symbolic output is encoded one-hot, the symbolic input is left as a multiple-valued variable.

2. perform multi-level optimization (simplification, common subexpression extraction, decomposition) of the multiple-valued network.

3. encode the symbolic input so that the total number of literals in the encoded network is minimal (simulated annealing is used for this purpose, while extensions of constrained embedding algorithms from the two level case are being studied).

A set of theorems, proved in [21], guarantees that step 2 of the above algorithm is complete, i.e. that all possible optimizations in all possible encodings can be performed in multiple-valued mode *provided that the appropriate cost function is available.*

The last observation is a key to understand both strengths and limits of this approach: the cost function that *mis-MV* minimizes is only an approximate *lower bound* on the number of literals that the

encoded network will have (much in the same spirit as what happens in the two-level case with symbolic minimization). This lower bound can be reached if and only if all the face constraints from all the nodes in the multiple-valued network can be simultaneously satisfied in a minimum length encoding, which is not possible in general (each node has a multiple-valued function, so the constraints can be extracted as described in Section 3). This lower bound is approximate because further optimizations on the encoded network can still reduce the number of literals.

In order to take this limitation into account, *mis-MV* computes at each step the currently optimal encoding, and uses it as an estimate of the cost of each multiple-valued node.

For example, if one denotes by $S^{\{1,2,3,4\}}$ a multiple-valued literal representing the boolean function that is true when variable S has value 1, 2 , 3 or 4, the estimated cost of $S^{\{1,2,3,4\}}$ with the codes: $e(S^{\{1\}}) = \overline{c_1}\overline{c_2}\overline{c_3}$ $e(S^{\{2\}}) = \overline{c_1}\overline{c_2}c_3$ $e(S^{\{3\}}) = \overline{c_1}c_2\overline{c_3}$ $e(S^{\{4\}}) = \overline{c_1}c_2c_3$ $e(S^{\{5\}}) = c_1\overline{c_2}\overline{c_3}$ $e(S^{\{6\}}) = c_1\overline{c_2}c_3$ would be 1, since the minimum sum of products expression for $\overline{c_1}\overline{c_2}\overline{c_3} + \overline{c_1}\overline{c_2}c_3 + \overline{c_1}c_2\overline{c_3} + \overline{c_1}c_2c_3$ with the don't cares (unused codes) $c_1c_2\overline{c_3} + c_1c_2c_3$ is c_1.

Currently *mis-MV* does not handle the output encoding problem. Its approach, though, can be extended to handle a symbolic minimization procedure similar to what is explained in section 3, and therefore to obtain a solution also to this problem.

4.5 Comparison of different methods

Programs such as *mustang* and *jedi* rely only on the two-level representation of the symbolic cover to extract a similarity measure between the context in which each pair of symbols appear. This measure is used to drive a greedy embedding algorithm that tries to keep similar symbols close in the encoded boolean space. This has clearly only a weak relation with the final objective (minimum cost implementation of a boolean network), and it makes an exact analysis of the algorithm performance on benchmark examples hard.

Some improvement can be seen in *muse*, that uses a one-hot encoding for both input and output symbols, and then performs a multi-level optimization. In this way at least some of the actual potential optimizations can be evaluated, and their gain can be used to guide the embedding, but there is no guarantee of optimality in this approach, and the output encoding problem is again solved with a similarity measure.

Full multi-level multiple-valued optimization (*mis-MV*) brings us closer to our final objective, because all potential optimizations can in principle be evaluated. The complexity of the problem, though, limits this potentiality to an almost greedy search, as in *misII*.

Still we do not have a complete solution to the encoding problem for multi-level implementation because:

1. we need to improve our estimate of the final cost to be used in multi-level multiple-valued optimization.

2. the problem of optimal output encoding must be addressed directly.

The algorithms described in this section, though, can and have been successfully used, and the path towards an optimal solution is at least clearer than before.

5 Satisfaction of input constraints

Many heuristic algorithms for heuristic satisfaction of input constraints have been published [7, 6, 11, 38, 20]. In [38] an exact solution, based on a branch-and-bound strategy, is also described. An exact

solution provides an encoding that satisfies all face constraints within a minimum code-length. A heuristic solution satisfies all constraints, but does not guarantee that the code-length is minimum. Obviously, 1-hot encoding provides always a solution, although at the price of a code-length equal to the number of symbols to encode.

In [7] the face constraints become the rows of a matrix M. The goal is finding a code matrix A, whose rows are the codes of the states. The number of columns of A is the code-length. It is observed that trivial constraints and constraints which are the bit-wise intersection of two or more constraints can be discarded (compaction criteria) and that the transpose of M is a solution to the constraint problem, i.e. $M^t = A$. In [7] a row-based encoding method was presented, where A is computed row by row, i.e. by computing the encoding of the symbols one at a time. This method fails to be effective for large examples. In [6], a column-based encoding scheme was proposed, where A is constructed column by column. The constraint matrix is compacted using the previous compaction criteria and A is constructed incrementally as transpose of the constraint matrix.

In [11] the main compaction criterion is extended to constraints obtained by bit-wise intersection of two or more rows or their bit-wise complements. Since checking for relationships between all constraints is too time- consuming, a procedure of constraint ordering is proposed. Given a particular ordering of rows of M, the code matrix A is constructed incrementally, column by column. Constraints which are already satisfied are discarded. The cost of a particular ordering is the number of columns of A. Then, the ordering of rows of the constraint matrix M is changed and A is recomputed. It is possible that a different code-length is found. An optimization problem is defined, whose objective is to find an ordering that minimizes the number of encoding bits required. A constructive heuristic algorithm and a simulated-annealing-based algorithms are proposed. The authors state that their algorithm can be generalized to the case of given code-length and the objective to optimize a weighted sum of the face constraints. Since one may not be able to satisfy all face constraints within a given code-length, a weight for each constraint quantifies the gain or loss for satisfying it or not. The weights are suggested by the cost function of the specific logic synthesis problem.

In [38] an exact solution to the face embedding problem is described and implemented. It is based on reducing the problem to embedding partially ordered sets (*posets*). A set of input constraints induces a *poset*, by ordering all the constraints according to the set inclusion relation. Also the hypercube in which the embedding takes place is represented by its underlying *face-poset*, by ordering all faces of all available dimensions according to the boolean inclusion relation. Satisfying the input constraints thus reduces to the problem of finding the minimum cube dimension, so that in the cube there is a *poset* equivalent to the one induced by the given set of input constraints. Equivalence of *posets* is based on the preservation of the inclusion and intersection operations between the elements of the *posets*. A branch-and-bound procedure for solving *poset* embedding is described. Counting arguments are used to cut the search space. Although not always computationally feasible, this exact solution, which was the first published, allowed the collection of a set of exact results against which to compare heuristic solutions. Moreover, a computationally efficient version of it, with heuristics to reduce the search space, is the core of a heuristic algorithm, *ihybrid*, also described in [38]. The algorithm *ihybrid* maximizes, for a given code-length, a weighted sum of the face constraints. It has performed very well under intensive experimentations.

In [37] it was proposed an algorithm to find correct (i.e. with no critical races) *unicode* (i.e. each state is coded with a *minterm*) single transition time (STT, i.e. all variables that must change during a transition are allowed to change simultaneously) state assignments of asynchronous sequential circuits. The problem is reduced to generating sets of disjoint 2-block partitions of the set of states (dichotomies), defining and computing maximal compatibles among them and covering the original dichotomies with a minimum number of maximal compatible dichotomies. In [43] it was noticed that a face constraint

is equivalent to a collection of dichotomies (called seed-dichotomies), one for each state that is 0 in the input constraint. Each dichotomy has in one block all states that are 1 in the input constraint, and in the other block one of the states that are 0 in the input constraint. Once we have all seed-dichotomies, we have reduced the face-embedding problem to solving a covering problem, as outlined by [37].

In [28] a formulation is provided starting from the approach of [37, 43]. Results of a prototype implementation indicate that, for almost all of the standard FSM benchmark examples, the input encoding problem can be solved exactly. This new approach can be easily extended to a weighted covering problem. An extension of the algorithm is also under investigation to handle a different cost function such as that required in multi-level input encoding, as seen in Section 4. In that case a minimum code length solution is desired where those constraints are satisfied which yield the best savings in factored form literals instead of product terms.

6 Satisfaction of input and output constraint

6.1 The problem of mixed input and output constraints

In [6] the proposed column-based heuristics handles not only face constraints, as already said, but also dominance output constraints. In [10] a column-based heuristics is presented to include also disjunctive constraints. When there are both input and output constraints, it is not possible to guarantee that an encoding always exists, In [6, 10], the algorithms that build the solution are applied only after a satisfiability check determines that the given mixed constraints are mutually compatible. The conditions of satisfiability of a set of mixed constraints given in [10] are:

1. The set of dominance constraints should not imply of two codes that each dominate the other. E.g. $code(st1) > code(st2)$, $code(st2) > code(st3)$, $code(st3) > code(st1)$.

2. For every disjunctive constraint, the set of dominance constraints should not imply of two *orred* codes, that one dominate the other. E.g. $code(st1) = code(st2) \vee code(st3)$, $code(st2) > code(st3)$, which implies $code(st1) = code(st2)$.

3. There should be no pair of disjunctive constraints, with the same *orred* codes and different resulting codes. E.g. $code(st1) = code(st2) \vee code(st3)$, $code(st4) = code(st2) \vee code(st3)$.

4. For every disjunctive constraint, the set of dominance constraints should not imply that the resulting code of the disjunction dominate a code that dominates all the *orred* codes. E.g. $code(st1) = code(st2) \vee code(st3)$, $code(st1) > code(st4)$, $code(st4) > code(st2)$, $code(st4) > code(st3)$, which implies $code(st1) > code(st2) \vee code(st3)$.

5. For every triple $s1$, $s2$, $s3$ such that $code(s1) > code(s2)$ and $code(s2) > code(s3)$, no face constraint should require $s1$ and $s3$ in one face and $s2$ not in that face.

6. For every disjunctive constraint, no face constraint should require the *orred* codes in one face and the resulting code not in that face. E.g. given $code(s1) = code(s2) \vee code(s3)$, no face constraint should require $s2$ and $s3$ in one face and $s1$ not in that face.

A proof that they are necessary and sufficient is not given in the paper. It is easy to show that these conditions are necessary, while one can show that, stated as they are, they are not sufficient. It is not easy

to determine the minimum superset of rules that make them sufficient. For a discussion and a complete algorithm that checks satisfiability efficiently, the reader is referred to [28].

We give an overview of the constructive algorithm sketched in [10], for a set of (already shown) satisfiable mixed constraints. Face constraints are compacted and an encoding equal to the transpose of the face constraint matrix is built. Sets of unsatisfied dominance relations are satisfied by adding one bit at a time to the codes, in a greedy fashion. An unsatisfied disjunctive relation is satisfied by raising bits in the codes of the *orred* symbols. This may introduce violations of face constraints and decrease the satisfied dominance relations, so bits are appended again to satisfy the violated face constraints. Then one goes back to the dominance constraints that became unsatisfied and iterates the loop as many times as needed to satisfy all constraints. It is claimed that the procedure always converges. No experiments are available to show the efficiency and quality of the solutions.

In [38] it is described also an algorithm, *iohybrid*, that maximizes, for a given code-length, a weighted sum of clusters of face and dominance constraints, obtained by a variant of symbolic minimization. It is based on an extension of *ihybrid*, to handle also output constraints. Its performance is good, but not of consistent quality. It compares favorably to the published results of the output encoding algorithm implemented in [6], although different twists of the two algorithms do not make comparisons easy.

In [28] a uniform framework for the efficient satisfaction of input and output constraints is provided. This approach allows exact solutions as well as trade-off schemes between code-length and maximum constraints satisfaction.

6.2 The encodability problem of GPI's

In Section 3 we reviewed exact encoding based on GPI's. Here we highlight the encodability problem of GPI's, to see how it relates to the problem of input and output constraints satisfaction. The selection of a cover of GPI's induces a set of boolean equations, one for each *minterm*, stating that, for all GPI's that include it, bit-wise *oring* the intersections of the codes of symbols in the tags yields the output asserted by that *minterm*. If the collection of boolean equations has a solution, the cover of GPI's is called encodable. The solution to the boolean equations is a set of compatible face, dominance and disjunctive constraints on the codes of the symbols. The solution to the set of boolean equations is a two-step process. The first step requires choosing for each boolean equation a dominance or disjunctive constraint that makes it true. In the worst case one has to try all possible selections. The second step requires checking the compatibility of the chosen output constraints among themselves and with the face constraints of the cover. This can be done efficiently by checking the polynomial-complexity compatibility conditions of a set of mixed constraints, already given in the Section 6.1.

A formulation to both steps that finds also an encoding within a certain code-length via general boolean satisfiability is proposed, even though its computational complexity is very high.

Once a cover of GPI's has been found encodable, i.e. a set of satisfiable input and output relations has been determined, one must find an encoding satisfying all given relations with a minimum number of bits. In this way the problem is reduced to the familiar paradigm of mixed input and output constraints satisfaction, as defined in the previous subsection.

For example, the encodable selection of Figure 6 yields the following non-trivial boolean equations (generated, in order, by the *minterms* 1101, 1100, 1111):

$$(code(out1) \wedge code(out2)) \vee (code(out1) \wedge code(out3)) = code(out1) \quad (1)$$

$$code(out1) \wedge code(out2) = code(out2) \quad (2)$$

$$code(out1) \wedge code(out3) = code(out3). \quad (3)$$

To satisfy the first equation one can choose one of the three constraints:

$$code(out2) > code(out1)$$
$$code(out3) > code(out1)$$
$$code(out1) \leq code(out2) \vee code(out3).$$

The second equation has only one possible choice:

$$code(out1) > code(out2).$$

The third equation has only one possible choice:

$$code(out1) > code(out3).$$

If one chooses the disjunctive constraint to satisfy the first equation, one can verify that one gets the following compatible collection of mixed constraints:

$$code(out1) \leq code(out2) \vee code(out3)$$
$$code(out1) > code(out2)$$
$$code(out1) > code(out3).$$

The encoded cover of Figure 6 shows a minimum-length encoding that satisfies the previous set of constraints.

7 Experimental results

We report some comparisons among available state assignment programs based on the techniques discussed in the previous sections. For the experiments we used the MCNC '89 set of benchmark FSM's.

7.1 The two-level case

We report one set of experiments that compare programs for two-level state assignments.

Table 1 summarizes the results obtained running the algorithms of *NOVA* ([38]), *KISS* ([7]) and random state assignments. The results of *NOVA* were obtained running *espresso* ([26]) to obtain the input constraints and the symbolic minimizer of *NOVA* built on top of *espresso* to obtain the mixed input/output constraints, *NOVA* to satisfy the constraints on the codes of the states and of the symbolic inputs (if any), and *espresso* again to obtain the final area of the encoded FSM. The best result of the different options of *NOVA* was shown in the Table. The results of *KISS* were obtained running *espresso* to obtain the input constraints, *KISS* to satisfy the constraints on the codes of the states and of the symbolic inputs (if any), and *espresso* again to obtain the final area of the encoded FSM. The areas under random assignments are the best and the average of a statistical average of a number of different (number of states of the FSM + number of symbolic inputs, if any) random state assignments on each example. The final areas obtained by the best solution of *NOVA* average 20% less than those obtained by *KISS*, and 30% less than the best of a number of random state assignments. *NOVA* can use any number of encoding bits greater than or equal to the minimum. The best results of *NOVA* on the benchmark of Table 1 have been obtained with a minimum encoding length, but this is not always the case. *KISS* uses a code-length sufficient to satisfy

all input constraints. Since it satisfies the constraints by an heuristic algorithm it does not always achieve the minimum necessary code-length.

Notice that the lower bound provided by symbolic minimization is often larger than the best upper bound achieved by encoding the FSM's, even though the available programs model only partially the effects of output encoding. This means that output encoding is more important than input encoding on the quality of final results.

Comparisons for some of the approaches mentioned above [32, 10] have not been carried out for the lack of an available implementation.

7.2 The multi-level case

We report a set of experiments that correlate good two-level state assignment to the corresponding multi-level logic implementation, comparing against an estimation-based multi-level encoding algorithm.

Table 2 reports the number of literals after running through the standard boolean optimization script in the multi-level logic synthesis system *misII* ([4]) with encodings obtained by *NOVA*, *MUSTANG* ([9]), *JEDI* ([19]) and random state assignments. In the case of *NOVA* only the best minimum code-length two-level result was given to *misII* . *MUSTANG* was run with *-p*, *-n*, *-pt*, *-nt* options and minimum code-length. *JEDI* was run with all available options and minimum code-length [18]. In all cases *espresso* was run before *misII*. The final literal counts in a factored form of the logic encoded by *NOVA* average 30% less than the literal counts of the best of a number of random state assignments. The best (minimum code-length) two-level results of *MUSTANG*, and *JEDI* versus the best (minimum code-length) two-level results of *NOVA* are also reported. Notice that in the case of *MUSTANG* and *JEDI* the run that achieved the minimum number of cubes is not necessarily the same that achieved the minimum number of literals. In the case of *NOVA* only the best two-level result was fed into *misII*, so the data reported refer to the same minimized cover. Even though *NOVA* was not designed as a multi-level state-assignment program, its performances compare successfully with *MUSTANG*. Among the three programs, the best literal counts are often given by *JEDI*. These data show that a state assignment that gives a good two-level implementation also gives a good multi-level implementation. This is consistent with the experiments reported in [40, 41, 39].

We report two kinds of experiments to verify the validity of *mis-MV* as *input encoder*:

- compare the relative importance of the various multi-valued optimization steps.
- compare *mis-MV* with some existing *state assignment* programs, such as *JEDI* ([19]), *MUSE* ([14]), *MUSTANG* ([9]) and *NOVA* ([38]). Notice that we want to compare only the input encoding algorithms of these programs and so we need to "shut off" all effects due to the encoding of the output part, captured by purpose (these programs embody also heuristics for the output encoding problem) or by chance. Therefore we replaced the codes returned by each program in the present state *only*, while the next state was simply replaced by one-hot codes.

The experiments were conducted as follows:

- a single simplified boolean script (using *simplify* only once) was used both for multi-valued and binary valued optimization.
- the script was run twice in all cases.
- *mis-MV*:

example	random		kiss			nova		
	b-area	a-area	#bits	#cubes	area	#bits	#cubes	area
bbara	616	649	5	26	650	4	24	528
bbsse	1089	1144	6	27	1053	4	29	957
bbtas	165	215	3	13	195	3	8	120
beecount	285	293	4	11	242	3	10	190
cse	1947	2087	6	45	1756	4	45	1485
dk14	720	809	9	24	550	6	25	500
dk15	357	376	6	17	391	5	17	289
dk16	1826	1994	12	55	2035	7	54	1188
dk17	320	368	6	19	361	5	17	272
dk27	143	143	4	9	117	4	7	91
dk512	374	418	7	18	414	5	17	289
donfile	1200	1360	12	24	984	5	28	560
ex1	3120	3317	7	42	2436	6	37	2035
ex2	798	912	6	31	744	5	27	567
ex3	342	387	6	18	432	4	17	306
ex5	324	358	5	15	315	4	14	252
ex6	810	850	5	24	792	3	25	675
iofsm	560	579	4	16	448	4	15	420
keyb	3069	3416	8	47	1880	5	48	1488
mark1	760	782	5	19	779	4	17	646
physrec	1677	1741	5	34	1564	4	33	1419
planet	4896	5249	6	89	4539	6	86	4386
s1	3441	3733	5	81	2997	5	63	2331
sand	4278	4933	6	95	4655	6	89	4361
scf	19650	21278	8	140	18760	7	137	17947
scud	2262	2533	6	71	2698	3	62	1798
shiftreg	132	132	3	6	72	3	4	48
styr	5031	5591	6	91	4186	5	94	4042
tbk	5040	6114	na	na	na	5	57	1710
train11	221	241	6	10	230	4	9	153
TOTAL	65453	72002			na			51053
%	100	110			na			77

Table 1: Comparison of FSM's encoding for two-level implementation

example	jedi	mustang	nova	jedi	mustang	nova	random
	#cubes	#cubes	#cubes	#lit	#lit	#lit	#lit
bbara	24	25	24	57	64	61	84
bbsse	30	31	29	111	106	132	149
bbtas	9	10	8	21	25	21	31
beecount	12	12	10	39	45	40	59
cse	52	48	45	200	206	190	274
dk14x	29	32	26	106	117	98	164
dk15x	19	19	17	67	69	65	73
dk16x	64	71	52	225	259	246	402
donfile	33	49	28	76	160	88	193
ex1	48	55	44	250	280	215	313
ex2	35	36	27	122	119	96	162
ex3	19	19	17	66	71	76	83
keyb	52	58	48	140	167	200	256
mark1	17	19	17	66	76	86	116
physrec	39	37	33	132	159	150	178
planet	93	97	86	547	544	560	576
s1	57	69	63	152	183	265	444
sand	105	108	96	549	535	533	462
scf	147	148	137	812	791	839	890
scud	57	83	62	127	286	182	222
shiftreg	4	4	4	0	2	0	16
styr	100	112	94	508	546	511	591
tbk	57	136	57	278	547	289	625
train11	11	10	9	27	37	43	44
TOTAL	1113	1288	1033	4678	5394	4986	6407
%	107	124	100	93	108	100	130

Table 2: Experiments on FSM's encoding for two and multi-level implementation

1. *espresso* was run on the unencoded machine.
2. all or part of the first script was run in *mis-MV*'s multi-valued mode.
3. the inputs were encoded, using the simulated annealing algorithm.
4. the remaining part of the first script and the second script were run in binary-valued mode.

- *JEDI*, *MUSE*, *MUSTANG* and *NOVA*:

1. each program was run in *input oriented* mode ("-e i" for *JEDI*, "-e p" for *MUSE*, "-pc" for *MUSTANG* and "-e ih" for *NOVA*) to generate the codes.
2. the symbolic input was encoded.
3. *espresso* was run again, using the invalid states as don't cares.
4. the script was executed twice.

We performed seven experiments on each machine, four using *JEDI*, *MUSE*, *MUSTANG* and *NOVA*, and three using *mis-MV*. The experiments on *mis-MV* differed in the point of the script where the symbolic inputs were encoded (*mis-MV* can carry on the multi-level optimizing operations on a multiple-valued network or on the encoded binary-valued network):

1. at the beginning. At this point, both *mis-MV* and *NOVA* extract the same face constraints by multiple-valued minimization. The two programs get different results because of the different face constraints satisfaction strategies. *mis-MV* satisfies the face constraints with a simulated annealing algorithm that minimizes the literal count of a two-level implementation. The cost function is computed by calling *espresso* and counting the literals. *NOVA* satisfies the input constraints with a heuristic deterministic algorithm that minimizes the number of product-terms of a two-level implementation.

2. after *simplify*, to verify multiple-valued boolean resubstitution.

3. after algebraic optimization (*gkx*, *gcx*, ...), to verify the full power of *mis-MV*.

Table 3 contains the results, expressed as factored form literals.

8 Conclusions

The input encoding problem for two-level implementations has been completely understood [7] and efficient algorithms for satisfaction of input constraints are available [38, 28]. Important contributions have been made for the input and output encoding problem for two-level implementations [6, 38, 10]. Exact solutions [10] are still very expensive for non-trivial examples and more work is needed to find efficiently high-quality solutions [29]. Efficient algorithms for satisfaction of input and output constraints are currently investigated [28].

When the target logic is multi-level the problem is much harder. Several heuristic approaches have been proposed [9, 19, 14]. An important theoretical contribution for input encoding has been made by [21], but more needs to be done, especially to take into account the presence of symbolic output variables.

It is interesting to notice that the problems of input and output encoding for logic synthesis, both those already understood, as input encoding, or still subject of active research, as output encoding,

example	jedi	muse	mustang	nova	best mis-MV	beginning	simplify	algebraic optimization
bbara	96	99	96	106	84	84	84	85
bbsse	125	126	148	151	131	130	132	131
bbtas	34	36	37	32	31	35	31	31
beecount	56	60	65	70	56	62	56	58
cse	189	192	208	214	195	191	199	195
dk14	96	102	108	98	79	97	79	81
dk15	65	65	65	65	68	65	68	69
dk16	254	244	314	351	247	225	247	261
dk17	63	58	69	58	62	58	62	64
dk27	30	29	34	38	27	27	27	27
dk512	73	73	78	93	68	70	68	69
donfile	132	131	195	186	123	127	123	123
ex1	256	239	252	246	232	240	232	237
ex2	176	169	197	167	144	143	144	154
ex3	87	96	98	98	82	82	86	82
ex4	71	72	73	84	72	90	74	72
ex5	79	79	80	83	69	67	69	69
ex6	93	92	90	98	84	85	85	84
ex7	87	84	100	94	78	89	79	78
keyb	186	180	203	195	146	186	172	146
lion	16	16	14	16	16	16	16	16
lion9	55	55	61	43	38	40	38	38
mark1	94	92	89	105	92	90	94	92
mc	32	30	30	32	30	35	30	30
modulo12	58	72	77	71	71	71	71	71
opus	83	70	88	90	70	87	70	74
planet	453	511	538	551	466	512	466	473
s1	339	291	377	345	249	335	253	251
s1a	262	195	264	253	214	217	214	225
s8	50	52	47	48	48	52	48	48
sand	556	498	519	542	509	523	509	529
shiftreg	24	25	34	35	24	24	24	24
styr	427	418	460	501	438	442	438	473
tav	27	27	27	27	27	27	27	27
TOTAL	4724	4578	5135	5186	4370	4624	4415	4487

Table 3: Multi-level input encoding comparison

all generate collections of face-embedding, dominance and disjunctive constraints, that must be either satisfied or shown unfeasible. If satisfiable, an encoding must be found of minimum code-length. Alternatively, one can optimize an area cost function, trading-off between encoding length and gain obtained from maximum satisfaction of constraints within the given code-length. Efficient solutions to these combinatorial optimization problems are a key to high-quality encoding programs.

9 Acknowledgements

Interesting discussions with Robert Brayton, Alex Saldanha and Sharad Malik are gratefully acknowledged.

This work was supported in part by DARPA under Contract N00039-87-C-0182, by the National Science Foundation under Grant ECS 84-30435, and by grants from MICRO, AMD, ATT Bell Laboratories, Bell Communications Research, GE, Harris, HP, Hughes, Intel, Microelectronics and Computer Technology, Olivetti, Philips/Signetics, Rockwell, Silicon Computers, and Xerox.

References

[1] D. Armstrong. A programmed algorithm for assigning internal codes to sequential machines. *IRE Transactions on Electronic Computers*, August 1962.

[2] D. Bostick, G. Hachtel, R. Jacoby, M. Lightner, P. Moceyunas, C. Morrison, and D. Ravenscroft. The Boulder optimal logic design system. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1987.

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

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, November 1987.

[5] W. Davis. An approach to the assignment of input codes. *IEEE Transactions on Electronic Computers*, August 1967.

[6] G. DeMicheli. Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros. *IEEE Transactions on Computer-Aided Design*, October 1986.

[7] G. DeMicheli, R. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design*, July 1985.

[8] S. Devadas, H-T. Ma, R. Newton, and A. Sangiovanni-Vincentelli. Synthesis and optimization procedures for fully and easily testable sequential machines. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1987.

[9] S. Devadas, H-T. Ma, R. Newton, and A. Sangiovanni-Vincentelli. Mustang: state assignment of finite state machines targeting multi-level logic implementations. *IEEE Transactions on Computer-Aided Design*, December 1988.

[10] S. Devadas and R. Newton. Exact algorithms for output encoding, state assignment and four-level Boolean minimization. In *The Proceedings of the 23rd Hawaii Conference on System Sciences*, January 1990.

[11] S. Devadas, A. Wang, R. Newton, and A. Sangiovanni-Vincentelli. Boolean decomposition in multilevel logic optimization. *IEEE Journal of solid-state circuits*, April 1989.

[12] T. Dolotta and E. McCluskey. The coding of internal states of sequential machines. *IEEE Transactions on Electronic Computers*, October 1964.

[13] D. Gregory, K. Bartlett, A. DeGeus, and G. Hachtel. SOCRATES: A system for automatically synthesizing and optimizing combinational logic. In *The Proceedings of the Design Automation Conference*, 1986.

[14] G. Hachtel, X. Du, and P. Moceyunas. Algorithms for state assignment based on multilevel representations. In *The Proceedings of the 23rd Hawaii Conference on System Sciences*, January 1990.

[15] J. Hartmanis. On the state assignment problem for sequential machines - 1. *IRE Transactions on Electronic Computers*, June 1961.

[16] R. Karp. Some techniques for state assignment for synchronous sequential machines. *IEEE Transactions on Electronic Computers*, October 1964.

[17] L. Lavagno, S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli. Mis-mv: Optimization of Multi-level Logic with Multiple-valued Inputs. Submitted for publication, May 1990.

[18] B. Lin. Experiments with jedi. Private communication, October 1989.

[19] B. Lin and A. R. Newton. Synthesis of multiple level logic from symbolic high-level description languages. In *Proceedings of the International Conference on Very Large Scale Integration*, 1989.

[20] B. Lin and A. Richard Newton. A generalized approach to the constrained cubical embedding problem. In *International Conference on Computer Design*, October 1989.

[21] S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli. Encoding symbolic inputs for multi-level logic implementation. In *The Proceedings of the IFIP International Conference on VLSI*, August 1989.

[22] E. McCluskey. Minimization of Boolean functions. *Bell Laboratories Technical Journal*, November 1956.

[23] G. De Micheli, T. Villa, and A. Sangiovanni-Vincentelli. Computer-aided synthesis of pla-based finite state machines. In *The Proceedings of the International Conference on Computer-Aided Design*, September 1983.

[24] A. Nichols and A. Bernstein. State assignments in combinational networks. *IEEE Transactions on Electronic Computers*, June 1965.

[25] D. Rosenkrantz. Half-hot state assignments for finite state machines. *IEEE Transactions on Computer-Aided Design*, May 1990.

[26] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design*, September 1987.

[27] A. Saldanha and R. Katz. PLA optimization using output encoding. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1988.

[28] A. Saldanha, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Exact algorithm for satisfying input and output encoding constraints. Manuscript in preparation, April 1990.

[29] A. Saldanha, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Symbolic minimization for input and output encoding. Manuscript in preparation, April 1990.

[30] T. Sasao. Input variable assignment and output phase optimization of pla's. In *IEEE Transactions on Computers*, October 1984.

[31] T. Sasao. Application of multiple-valued logic to a serial decomposition of pla's. In *The Proceedings of the International Symposium on Multiple-Valued Logic*, June 1989.

[32] G. Saucier, M. Crastes de Paulet, and P. Sicard. Asyl: a rule-based system for controller synthesis. *IEEE Transactions on Computer-Aided Design*, November 1987.

[33] G. Saucier, C. Duff, and F. Poirot. A new embedding method for state assignment. *The Proceedings of the International Workshop on Logic Synthesis*, May 1989.

[34] R. Stearns and J. Hartmanis. On the state assignment problem for sequential machines - 2. *IRE Transactions on Electronic Computers*, December 1961.

[35] Y. Su and P. Cheung. Computer minimization of multi-valued switching functions. *IEEE Transactions on Computers*, September 1972.

[36] Y. Tohma, Y. Ohyama, and R. Sakai. Realization of fail-safe sequential machines by using a k-out-of-n code. *IEEE Transactions on Computers*, November 1971.

[37] J. Tracey. Internal state assignment for asynchronous sequential machines. *IRE Transactions on Electronic Computers*, August 1966.

[38] T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment for optimal two-level logic implementations. In *The Proceedings of the Design Automation Conference*, June 1989.

[39] W. Wolf. Recoding-derived bounds for input encoding. Manuscript in preparation, January 1990.

[40] W. Wolf, K. Keutzer, and J. Akella. A kernel-finding state assignment algorithm for multi-level logic. In *The Proceedings of the Design Automation Conference*, June 1988.

[41] W. Wolf, K. Keutzer, and J. Akella. Addendum to "a kernel-finding state assignment algorithm for multi-level logic". In *IEEE Transactions on Computer-Aided Design*, August 1989.

[42] S. Yang and M. Ciesielski. A generalized pla decomposition with programmable encoders. In *The Proceedings of the International Workshop on Logic Synthesis*, May 1989.

[43] S. Yang and M. Ciesielski. On the relationship between input encoding and logic minimization. In *The Proceedings of the 23rd Hawaii Conference on System Sciences*, January 1990.