Optimum and Suboptimum Algorithms for Input Encoding and Its Relationship to Logic Minimization

Saeyang Yang and Maciej J. Ciesielski, *Member, IEEE*

Abstract—**A new theoretical formulation of the input encoding problem is presented, based on the concept of compatibility of dichotomies. The input encoding problem is shown to be equivalent to a two-level logic minimization. Three possible techniques to solve the encoding problem are discussed, based on: 1) techniques borrowed from classical logic minimization (generation of prime dichotomies and solving the covering problem), 2) graph coloring applied to the graph of incompatibility of dichotomies, and 3) extraction of essential prime dichotomies followed by graph coloring. The extraction of essential prime dichotomies serves the same purpose as the extraction of essential prime implicants in logic minimization, in the sense that it reduces the size of the covering/graph coloring problem. The conditions of optimality of the solutions to the input encoding problem are discussed. For near-optimum results a powerful heuristic, based on an iterative improvement technique, has been developed and implemented as a computer program Dichtomy-based symbolic Input Encoding Technique (DIET). The test results indicate that DIET compares favorably with KISS and NOVA in terms of the CPU time, is superior to both programs in terms of the encoding length, and requires considerably less memory. The new method can be applied to the input encoding of combinational logic and the state assignment of finite state machine's (FSM's) in both two-level and multilevel implementations.**

I. Introduction

INPUT encoding arises in a number of important logic synthesis problems. A typical example of input encoding in VLSI is the binary encoding of symbolic inputs that appear in high-level description of digital logic, or the encoding of mnemonic input fields of the microcode. The input encoding also appears in the state assignment of finite state machines (FSM's), i.e., sequential machines implemented as combinational logic with feedback. In this case, however, the input encoding is only a part of the encoding problem, because the states to be encoded are both inputs as well as outputs of the combinational logic. The input encoding problem consists of choosing a binary representation of the symbolic input of the digital logic. Since, in general, the encoding affects the area needed to implement the digital logic, a good input encoding is very important.

A traditional approach to the encoding problem typically involves the encoding followed by logic minimization of two-level or multilevel implementation. The encoding techniques based on this approach attempt to find an encoding that will simplify the logic minimization. Classical state assignment techniques are good examples of this approach, as they are generally directed towards the simplification of the prospective combinational component of the FSM [1], [7], [9], [11], [19]. Specifically, these techniques attempt to find the assignment that minimizes the number of product terms (rows of the PLA) among the assignments of a given (typically minimum) code length.

Manuscript received January 1, 1990. This work was supported by the University of Massachusetts at Amherst under Faculty Research Grant 1-03286. This paper was recommended by Guest Editor A. Sangiovanni-Vincentelli.

M. J. Ciesielski is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

S. Yang is with the Microelectronics Center of North Carolina, Research Triangle Park, NC 27709.

IEEE Log Number 9039376.

Recently, a new encoding technique has been proposed by De Micheli [4], [5], based on an innovative strategy: instead of trying to estimate the effect of encoding on the possible simplification of combinational logic, logic minimization is applied *before* the code assignment. That is, this technique attempts to find the assignment of minimum code length among the assignments that minimize the cover of a Boolean function. In this new approach a set of mnemonic inputs is represented by one multiple-valued variable, and the resulting multiple-valued input Boolean function is minimized using a multiple-valued minimizer. With this representation, the logic is represented in a symbolic, i.e., code independent form. This design methodology avoids the dependence on variable representation and attempts to find a mimimum representation of logic function independently of the encoding of its inputs. The optimal input encoding is then obtained by constrained input encoding, i.e., the encoding of the inputs of the digital logic so that they are compatible with the symbolic cover.

The constrained input encoding has applications in decomposition of Boolean functions. Efficient PLA decomposition techniques, proposed in [6], [22], use the constrained input encoding to re-encode the primary inputs (to define the outputs of the input decoders) in order to achieve overall PLA area reduction. It has been demonstrated that the constrained input encoding problem can be also applied to input encoding of multilevel logic [13]. Therefore, an efficient method for solving the constrained input encoding problem is highly desirable.

In this paper we present a new theoretical formulation of the constrained input encoding problem based on *compatibility of seed dichotomies*, and investigate the relationship between the input encoding and logic minimization. The organization of the paper is as follows. Section II reviews the basic concept of the constrained input encoding. A new approach to the encoding problem is presented in Section III, and the algorithm based on this approach is described in Section IV. The application of the encoding technique to state assignment of FSM, and some experimental results are shown in Section V.

II. Constrained Input Encoding: An Overview

In this section we review the basic concept of constrained input encoding based on symbolic minimization [4], [5]. This concept is illustrated by elaborating on a specific example taken from [5].

Consider a combinational circuit that implements an instruction decoder, Fig. 1. The circuit has two sets of input signals, denoted *ADDR* and *OPC*, and one set of outputs, denoted *CNTR*. These input/output signals are referred to as *symbolic input/output variables*.

The input variable *ADDR* can take one of the three possible input "values," *INDEX*, *DIR*, and *IND*, corresponding to the three types of addressing modes. Similarly, the input variable *OPC* represents four possible operation codes, *AND*, *OR*, *ADD*, and *JMP*, and the output variable *CNTR* represents four values of control signals, *CNTA*, *CNTB*, *CNTC*, and *CNTD*. The circuit can be represented by the following symbolic truth table, with inputs and outputs represented by mnemonic strings (symbols). Each row of the table specifies a symbolic output for a given combination of symbolic inputs, and is called a *symbolic implicant*:

ADDR	*OPC*	*CNTR*
INDEX	*AND*	*CNTA*
INDEX	*OR*	*CNTA*
INDEX	*JMP*	*CNTA*
INDEX	*ADD*	*CNTA*
DIR	*AND*	*CNTB*
DIR	*OR*	*CNTB*
DIR	*JMP*	*CNTC*
DIR	*ADD*	*CNTC*
IND	*AND*	*CNTB*
IND	*OR*	*CNTD*
IND	*JMP*	*CNTD*
IND	*ADD*	*CNTC*

Our goal is first to minimize this table in its symbolic form, and then find the binary encoding of the input symbols of minimum possible length. The minimization of the truth table in a symbolic form can be achieved by using techniques of multiple-valued minimization [17], [3]. The symbolic cover is translated into a multiple valued cover by representing each symbolic value using positional cube notation:

ADDR	*OPC*	*CNTR*
100	1000	1000
100	0100	1000
100	0001	1000
100	0010	1000
010	1000	0100
010	0100	0100
010	0001	0010
010	0010	0010
001	1000	0100
001	0100	0001
001	0001	0001
001	0010	0010

This cover is then reduced, using multiple valued Boolean minimizer:

ADDR	*OPC*	*CNTR*
100	1111	1000
010	1100	0100
001	1000	0100
001	0101	0001
010	0011	0010
001	0010	0010

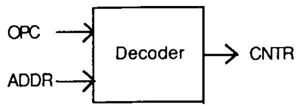


Fig. 1. Combinational circuit with symbolic inputs.

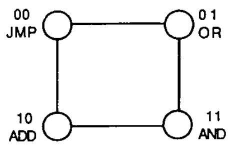


Fig. 2. Geometric representation of input encoding.

This type of minimization is referred to as a *disjoint minimization*, because the input symbols are grouped for each output symbol independently (each symbolic implicant of the cover is output-disjoint). It should be pointed out that this table could be minimized even further, if encoding of the input and output symbols were considered simultaneously. This, however, requires introducing covering relations among the binary encodings of the output symbols [5], and is not considered in this paper.

The minimized symbolic (multiple valued) cover imposes certain constraints on the binary encoding of inputs. Specifically, encoding of the inputs is needed such that the Boolean cover obtained with this encoding is compatible with the minimized symbolic cover, i.e., their cardinalities are equal.

The constraints imposed by the minimized cover specify the relationships among the binary codes chosen for the inputs. Specifically, the binary codes of the inputs should be grouped together in the same way the inputs are grouped in the minimized symbolic cover. These constraints can be conveniently expressed for each symbolic input by a constraint matrix derived from the minimized cover. In our example, two constraint matrices can be derived from the cover, one for *ADDR* and one for *OPC* inputs. Consider, for example, the matrix associated with the *OPC* variable:

$$\begin{array}{c} 1111 \\ 1100 \\ 1000 \\ 0101 \\ 0011 \\ 0010. \end{array}$$

Each column of this matrix corresponds to one input value of the input variable *OPC*, and each row represents a group of inputs that are mapped into the same output. The constraints imposed by this matrix can be expressed as follows [4]: *the minimum dimension Boolean subspace containing the encodings of inputs assigned to the same group must not intersect the code assigned to any input not contained in the corresponding group.*

The constraint matrix for the *OPC* variable is satisfied, for example, by the following encoding: *AND* = 11, *OR* = 01, *ADD* = 10, and *JMP* = 00, as shown in Fig. 2. In the next section we show how to obtain the minimum length input encoding using a new and efficient procedure based on graph theory.

III. Formulation of the Input Encoding Problem

3.1. Constraint Matrix

Let M be a matrix composed of those columns of the input part of the minimized symbolic cover that are associated with the input variable to be encoded. Each row of M represents a symbolic literal associated with a given input variable, i.e., a literal of the corresponding multiple valued variable, X. Each column of M represents one value of X:

$$M(i,j)=\begin{cases}1, & \text{if } j\text{th value of } X \text{ is present in } i\text{th literal,}\\ 0, & \text{otherwise.}\end{cases}$$

Using notation similar to that of De Micheli [4], define an *input group* to be a subset of columns of M (values of X) in row i, for which $M(i,j) = 1$. Also define a *group face* to be a minimum dimension Boolean subspace containing the binary encoding of the values of X assigned to that group. The constrained encoding problem is to assign the distinct binary codes of minimum length to each column of M, such that each group face does not intersect the code assigned to any column not in the group.

Before solving the set of constraints, the constraint matrix M can be reduced. It has been shown that all duplicate rows, as well as the rows with all 1's, can be deleted from M without affecting the encoding [4]. The resulting matrix is called a *reduced constraint matrix*, and denoted M^R. The rows with single 1's, and the *meet* rows, i.e., the rows that are conjunctions of two or more rows, need not be deleted, however, as it was done in [4]. It will be apparent from the description of our encoding procedure in the next section, that this may result in an encoding with fewer bits.

In general, the columns of the reduced constraint matrix M^R may not be distinct. Deleting duplicate columns results in further simplification of M^R. This can be done in both the input encoding of decoded PLA's, and in the encoding of FSM's, without affecting the quality of the encoding. Recall that in the decoded PLA a row of the constraint matrix represents an output of the decoder, and a column of the constraint matrix represents a product term of the decoding logic [22]. Identical columns, representing different product terms, may have the same encoding. On the other hand, in the encoding of FSM deleting the duplicate columns means assigning identical binary codes to different states of the machine. It can be shown that, if the rows with single 1's are not deleted from the encoding constraint matrix of the FSM, the reduced constraint matrix will contain identical columns if and only if the corresponding states are strongly equivalent, i.e., for each input they have the same next state and produce the same output.

Example 1: Consider the FSM described by the following state table:

	I_1	I_2	I_3
S_1	S_3, 1	S_4, 0	S_3, 0
S_2	S_3, 1	S_2, 1	S_1, 0
S_3	S_1, 0	S_3, 0	S_2, 1
S_4	S_1, 0	S_2, 1	S_2, 1

The FSM has a set of symbolic primary inputs $I = \{I_1, I_2, I_3\}$, and a set of symbolic state variables $S = \{S_1, S_2, S_3, S_4\}$, that are to be encoded as binary vectors. Each symbolic input and symbolic state can be represented by a multiple valued variable and represented in a positional cube notation, resulting in the following symbolic cover;

100 1000 00101
100 0100 00101
100 0010 10000
100 0001 10000
010 1000 00010
010 0100 01001
010 0010 00100
010 0001 01001
001 1000 00100
001 0100 10000
001 0010 01001
001 0001 01001.

This symbolic cover is minimized using multiple valued Boolean minimizer to obtain

100 0011 10000
010 0101 01001
001 0011 01001
100 1100 00101
010 1000 00010
001 0100 10000
001 1000 00100
010 0010 00100.

The constraint matrix M_S, associated with the state vector S, is obtained from this cover:

$$M_S=\begin{bmatrix}0&0&1&1\\0&1&0&1\\0&0&1&1\\1&1&0&0\\1&0&0&0\\0&1&0&0\\1&0&0&0\\0&0&1&0\end{bmatrix}$$

The corresponding reduced constraint matrix, M_S^R, is derived by removing duplicate rows, 0011 and 1000:

$$M_S^R=\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}$$

△

A number of heuristic algorithms to obtain minimum length encodings which satisfy the constraints matrix, were proposed [4]-[6], [21]. In [4], a row-based encoding scheme has been presented. In this scheme the encoding of all input symbols is represented by a *code matrix* S, whose rows represent individual encodings. Matrix S is constructed row by row, such that the *constraint relation* is satisfied between matrices M and S. The objective is to find a matrix S with a minimum number of columns. Another approach, using the column-based encoding scheme has been also investigated [5]. In this case the codes are constructed column by column, introducing at each step a single bit encoding of all the symbols. This encoding must satisfy both the input and the output encoding relation. A somewhat similar approach was proposed by Devadas [6] for re-encoding of outputs of the input decoder to a PLA. Recognizing that the transpose of a constraint matrix, M^T, trivially satisfies the encoding constraint, Devadas formulated the constrained encoding problem as a constrained ordering problem. In his approach, an ordering of row of M^T is sought such that the encoding constraint is satisfied by a subset of M^T with a minimum number of columns. In some cases, however, an encoding of shorter length can be obtained that is not derivable from M^T by simple reordering. This point will be illustrated in the next section.

In the remainder of this paper, we propose a more efficient method for the constrained encoding, based on the concept of compatibility of dichotomies.

3.2. Dichotomies

Our approach to the constrained encoding problem is based on the concept of compatibility of dichotomies. The notion of dichotomy was introduced for the first time by Tracey [20] in hazard-free state assignment for asynchronous FSM's.

Definition 1: The *dichotomy* is a disjoint 2-block partition of a subset of column indexes of the reduced constraint matrix M^R. The *seed dichotomy* associated with row i of matrix M^R is a disjoint 2-block partition $(l:r)$ of a subset of column indexes of M^R, such that block l contains all column indexes j, for which $M^R(i,j) = 1$, and block r contains exactly one column index, k, such the $M^R(i,k) = 0$.

Example 2: From the first row of the constraint matrix M_S^R in Example 1, the following seed dichotomies are generated:

$$0\ 0\ 1\ 1 \Rightarrow (3\ 4:1), (3\ 4:2).$$

The indexes 1, 2, 3, 4 refer to the columns S_1, S_2, S_3, S_4 of the constraint matrix. △

Definition 2: Dichotomy $D_i = (l_i:r_i)$ is said to *cover* dichotomy $D_j = (l_j:r_j)$ if $l_i \supseteq l_j$ and $r_i \supseteq r_j$, or $l_i \supseteq r_j$ and $r_i \supseteq l_j$. D_j is then said to be covered by D_i.

Definition 3: An *irredundant set of dichotomies* is a set of dichotomies such that no dichotomy is covered by other dichotomies.

Example 3: Given the constraint matrix M_S^R in Example 1, the following set of irredundant seed dichotomies is generated:

$$0\ 0\ 1\ 1 \Rightarrow (3\ 4:1), (3\ 4:2)$$
$$0\ 1\ 0\ 1 \Rightarrow (2\ 4:1), (2\ 4:3)$$
$$1\ 1\ 0\ 0 \Rightarrow (1\ 2:3), (1\ 2:4).$$

Notice that the seed dichotomies generated from the last three rows of the constraint matrix M_S^R are not included in this set as they are covered by other seed dichotomies. For example, the seed dichotomies $(1:2)$, $(1:3)$, $(1:4)$, derived from row 1000, are covered by $(2\ 4:1)$, $(1\ 2:3)$, and $(1\ 2:4)$, respectively. □

Definition 4: Two dichotomies, D_1, $= (l_1:r_1)$ and $D_2 = (l_2:r_2)$, are said to be *incompatible* if there exists an unordered pair of indexes (j,k), $j, k \in l_1$ or $j, k \in r_1$, such that $j \in l_2$ and $k \in r_2$, or if the same is true with the role of D_1 and D_2 reversed. Otherwise the two dichotomies are called *compatible*.

Example 4: In Example 3, the seed dichotomies $(1\ 2:3)$ and $(1\ 2:4)$ are compatible. Similarly, $(1\ 2:3)$ and $(3\ 4:2)$ are compatible, but $(1\ 2:3)$ and $(2\ 4:1)$ are incompatible. △

Definition 5: A *union* of two compatible dichotomies, $D_1 = (l_1:r_1)$ and $D_2 = (l_2:r_2)$, denoted $D_1 \cup D_2$, is a dichotomy, $D = (l:r)$, such that $l = l_1 \cup l_2$ and $r = r_1 \cup r_2$, or $l = l_1 \cup r_2$ and $r = r_1 \cup l_2$.

Dichotomies can be efficiently represented in the computer as binary vectors as follows. An element to be encoded (column of M) is represented by a 2-b string 10 if it appears in l_i of D_i, by 01 if it appears in r_i of D_i, and by 00 (don't care) if it does not appear in D_i. Each dichotomy is, therefore, described as a vector of 2-b strings, one for each column of M. Bitwise *OR* of such defined vectors provides a simple compatibility test for the corresponding dichotomies. If the result of such OR-ing contains a 2-b string 11, the corresponding dichotomies are incompatible. Actually this test has to be performed twice, with the role of the left and right blocks, l_i, r_i, reversed. The dichotomy is incompatible if both results contain a string 11. Otherwise, they are compatible and the result is the union of the corresponding dichotomies.

Example 5:

$$\begin{array}{rl} & \quad\ 1 \quad\ 2 \quad\ 3 \quad\ 4 \\ (1\ 2{:}3) = & (10 \quad 10 \quad 01 \quad 00) \\ (1\ 2{:}4) = & \underline{(10 \quad 10 \quad 00 \quad 01)} \\ (1\ 2{:}3\ 4) = & (10 \quad 10 \quad 01 \quad 01) \text{ compatible, union} \end{array}$$

$$\begin{array}{rl} & \quad\ 1 \quad\ 2 \quad\ 3 \quad\ 4 \\ (1\ 2{:}3) = & (10 \quad 10 \quad 01 \quad 00) \\ (2\ 4{:}1) = & \underline{(01 \quad 10 \quad 00 \quad 10)} \\ & (11 \quad 10 \quad 01 \quad 10) \text{ incompatible.} \end{array}$$

△

Let $D_k = (l_k:r_k)$ be an arbitrary dichotomy derived from the constraint matrix M. This dichotomy implies a 1-b encoding of the subset of column indexes, $l_k \cup r_k$, of matrix M. The corresponding encoding is obtained by assigning a single bit 0 (or 1) to all indexes included in block l_k, and assigning an opposite bit, 1 (or 0), to all indexes in block r_k. This encoding partially satisfies the constraint matrix in the sense that it satisfies the constraints imposed by the seed dichotomies covered by D_k.

Example 6: Dichotomy $D_i = (1\ 2:3\ 4)$ implies a 1-b encoding of columns $(1, 2, 3, 4)$. This encoding satisfies the constraints imposed by the seed dichotomies $(1\ 2:3)$, $(1\ 2:4)$, (3 4 : 1), and (3 4 : 2), covered by D_1:

	D_1
1	0
2	0
3	1
4	1 .

△

3.3. Classical Logic Minimization Approach to Encoding

Remarkably, the input encoding problem can be solved using techniques of Boolean minimization. In this section we establish the relationship between the encoding and logic minimization and use this as a basis for our formulation of the input encoding problem. First we introduce the definitions of prime, distinct, and essential dichotomies to help establish these relationships. In the following, the constrained matrix M is considered to be reduced.

Definition 6: A *prime dichotomy* is a dichotomy that is not included in any other dichotomy.

Prime dichotomies can be obtained from a set of seed dichotomies by the following procedure. A seed dichotomy is selected and combined, using a union operation, with a compatible dichotomy (seed dichotomy) from the set. The resulting new dichotomy is then added to the set and all dichotomies (seed dichotomies) covered by the new dichotomy are labeled. The new dichotomy is then used as a "seed" and combined with the next compatible dichotomy. The process is repeated until there are no more compatible dichotomies that can be added to the seed. In particular, one prime dichotomy can be obtained for each row of the constraint matrix as a union of all seed dichotomies derived from this row. A prime dichotomy $D_i = (l_i : r_i)$ derived from the constraint matrix M has the property that $l_i \cup r_i$ includes *all* indexes of matrix M. Furthermore, prime dichotomies are incompatible with each other.

To obtain a set of *all* prime dichotomies from the set of seed dichotomies, a procedure similar to that for obtaining all prime implicants from the minterms can be used [14]. First, a union of all pairs of compatible seed dichotomies is calculated, and the seed dichotomies covered by the resulting dichotomies are labeled. The process is repeated by pairing the compatible dichotomies in the new set until no more compatible pairs can be found. A set of unlabeled dichotomies constitute a complete set of prime dichotomies.

Definition 7: A *distinct seed dichotomy* is a seed dichotomy which is covered by one and only one prime dichotomy.

Definition 8: An *essential prime dichotomy* is a dichotomy which contains at least one distinct seed dichotomy.

It is interesting to notice that such defined dichotomies are equivalent to implicants of Boolean functions; a prime dichotomy is equivalent to a prime implicant, and a seed dichotomy is equivalent to a minterm (it represents the smallest entity from which the dichotomies can be constructed). In fact, the input encoding problem can now be shown to be equivalent to logic minimization.

The goal of logic minimization is to obtain the minimum cover of a Boolean function. Classical logic minimization techniques achieve this by first generating all prime implicants and then solving the covering problem, i.e., grouping the implicants into a minimum set of prime implicants that covers the function [14]. In principle, the input encoding problem can be solved using a similar approach, i.e., by creating all prime dichotomies and grouping them into a minimum set, so that each seed dichotomy is covered by some prime dichotomy in the "cover."

Theorem 1: The length of the encoding satisfying the constraint matrix M is equal to the number of prime dichotomies needed to cover all seed dichotomies of M.

Proof: By construction, each prime dichotomy is obtained as a union of some seed dichotomies. The 1-b encoding that this prime dichotomy represents satisfies the constraints imposed by the respective seed dichotomies. The encoding imposed by the set of prime dichotomies needed to cover all seed dichotomies satisfies the entire constraint matrix M. Since each prime dichotomy implies a 1-b encoding, the length of the encoding is equal to the number of prime dichotomies in the cover. □

The conclusion from this theorem is that the minimum length encoding can be obtained by finding a minimum cover of prime dichotomies.

Example 7: Consider the constraint matrix M_S^R and the irredundant set of seed dichotomies derived in Example 3: (1 2:3), (1 2:4), (3 4:1), (3 4:2), (2 4:1), and (2 4:3). From this set the following prime dichotomies are generated: (1 2:3 4), (2 4:1 3, (1 2 4:3), (1:2 3 4). The covering table is now constructed, with rows representing prime dichotomies and columns representing seed dichotomies. The covering problem is then solved by finding a minimum set of prime dichotomies which cover all seed dichotomies:

	(12:3)	(12:4)	(34:1)	(34:2)	(24:1)	(24:3)
√ (1 2:34)	X	X	X	X		
√ (2 4:1 3)					X	X
(1 2 4:3)	X					X
(1:2 3 4)			X		X	

In this case only two prime dichotomies, (1 2:3 4) and (2 4:1 3), are needed to cover all seed dichotomies. This means that only 2 b are needed for the encoding. The first bit encodes columns 1, 2 with bit 0, and columns 3, 4 with bit 1. The second bit encodes columns 2, 4 with bit 0, and columns 1, 3 with bit 1:

	D_1	D_2
1	0	1
2	0	0
3	1	1
4	1	0 .

△

The reason for not deleting the meet rows and the rows with single 1's from the constraint matrix prior to the encoding should now be apparent. If the reduced matrix has the rows with single 1's deleted, some additional bits have to be added to the encoding that satisfies this constraint matrix, so that each column of the original constraint matrix is coded by a distinct binary pattern. Thus the problem is effectively partitioned into two separate problems: 1) the encoding of rows with two or more 1's, and 2) the encoding of rows with single 1's. Clearly, even if the encoding of each problem separately is solved optimally, the optimality of the entire set cannot be guaranteed. In contrast, in the approach taken in this work *all* the constraints are considered at once, and thus it is possible to achieve better results in terms of the encoding length. Notice that in doing so the complexity of the problem does not increase significantly; the number of seed dichotomies that are generated from those rows, and which are not covered by seed dichotomies generated from other rows, is small compared to the total number of dichotomies. Including these seed dichotomies increases the degree of flexibility in choosing the best cover, and thus may result in better solutions.

A similar argument holds for the meet rows. The meet rows (which first would have to be detected as a conjunction of some other rows) do not have to be explicitly deleted from the original constraint matrix; all seed dichotomies generated from these rows will be automatically deleted as they are all covered by other seed dichotomies. Notice that our procedure may also delete partial constraints imposed by other (nonmeet) rows, if some of seed dichotomies generated from these rows are covered by other seed dichotomies. Classical input encoding techniques are unable to detect such partial constraints, as they can only delete the entire row. For these reasons our approach can give, in general, better results than those presented in [4]-[6].

3.4. A Compatible Graph Coloring Approach

In this section an alternative, graph theoretical approach to the constrained input encoding problem is presented.

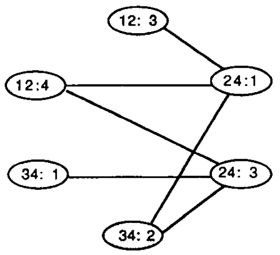
Given a constraint matrix M, a *graph of incompatibility of seed dichotomies*, $G_m(V, E)$ is constructed. $G_M(V, E)$ is a simple, nondirected graph, whose nodes V are in one-to-one correspondence with the irredundant set of seed dichotomies derived from M, and whose edges E represent the incompatible relations among the seed dichotomies. There is an edge between each pair of nodes if and only if the corresponding seed dichotomies are incompatible. The encoding satisfying the constraints imposed by the seed dichotomies can be obtained by compatible coloring of the graph $G_M(V, E)$.

Definition 9: A *compatible coloring* of graph G_M is a coloring of its nodes such that no two adjacent nodes in the graph are assigned the same color, and all the nodes with the same color are compatible.

Notice that this definition allows a node to be assigned multiple colors, as long as all nodes with the same color are compatible. Our compatible graph coloring algorithm, therefore, attempts to find sets of maximally compatible seed dichotomies. The following theorem allows us to obtain a miminum length encoding by performing compatible graph coloring of $G_M(V, E)$.

Theorem 2: The minimum encoding length satisfying the constraint matrix M is equal to the minimum number of colors in compatible coloring of $G_M(V, E)$.

Proof: By definition, compatible graph coloring finds the sets of compatible seed dichotomies and merges them into single dichotomies. Each dichotomy, associated with one color, represents a 1-b encoding that satisfies the constraints imposed by the respective seed dichotomies. Since the resulting set of dichotomies covers all seed dichotomies, this set satisfies all the constraints imposed by the constraint matrix M, and the encoding length is equal to the cardinality of this set. Therefore, the minimum encoding length is equal to the minimum number of colors. □



$D_1 = (1\ 2 : 3\ 4)$
$D_2 = (2\ 4 : 1\ 3)$

	D_1	D_2
S_1	0	1
S_2	0	0
S_3	1	1
S_4	1	0

Fig. 3. Compatible graph coloring.

Example 8: Consider again the constraint matrix M_S^R and the irredundant set of seed dichotomies as in Example 3. The graph $G_M(V, E)$ for this example and its compatible coloring is shown in Fig. 3. The two colors, corresponding to the dichotomies $(1\ 2 : 3\ 4)$ and $(1\ 3 : 2\ 4)$, give the final 2-b encoding. △

In general, the assignment of a single bit code $(0, 1)$ to the blocks l_k and r_k of dichotomy D_k during the final encoding is not arbitrary. If the encoding input symbols (columns of the constraint matrix) are outputs of the precious stage logic, e.g., input decoders to the PLA, this assignment may affect the size of the input logic (decoders) [22]. This problem, referred to as an *output phase assignment*, can be solved using techniques reported in [18].

3.5. Extracting Essential Prime Dichotomies

When minimizing a Boolean function using classical methods, the size of the covering problem can be reduced by extracting essential prime implicants from the cover. This approach has been taken in the advanced logic minimizers, such as Espresso [2], [17], UMini [3], and Kuai-Exact [16]. It has been demonstrated that essential implicants can be extracted without actually generating all prime implicants [18], [12], [17], [15]. A similar concept can be used in our encoding problem: essential prime dichotomies can be extracted from the set of seed dichotomies to reduce the number of nodes in the graph. The extraction of essential dichotomies, however, can be performed in a much simpler way than the extraction of essential prime implicants. This is indicated by the following theorem.

Theorem 3: The row r_i of the constraint matrix represents an essential prime dichotomy if and only if there exists a seed dichotomy generated from this row which is incompatible with all seed dichotomies not covered by row r_i.

Proof: Assume there exists a seed dichotomy D, generated from row r_i, that is not compatible with any other dichotomy, except for those covered by r_i. Therefore, by definition, D is a distinct seed dichotomy, and is covered only by the prime dichotomy generated by row r_i. Thus the corresponding row must represent an essential prime dichotomy. On the other hand, if row r_i represents an essential prime dichotomy, it must contain at least one distinct seed dichotomy that satisfies the above condition. □

By construction, each dichotomy, or seed dichotomy, represents a 1-b encoding of the subset of columns of the constraint matrix. Since essential prime dichotomies are obtained by grouping compatible seed dichotomies such that at least one of them is covered only by this prime dichotomy, the encoding represented by essential prime dichotomies must be included in the final encoding vector. By extracting essential prime dichotomies, the corresponding seed dichotomies are removed from the graph $G_M(V, E)$.

Example 9: In the example in Fig. 3 there is only one essential prime dichotomy, (1 2 : 3 4). The 1-b encoding implied by this dichotomy must be included in the final encoding. After removing the seed dichotomies, (1 2:3), (1 2:4), (3 4:1), and (3 4:2), covered by this essential prime dichotomy, the graph $G_M(V, E)$ contains only two nodes, (2 4:1) and (2 4:3). The seed dichotomies represented by these nodes are compatible and, therefore, can be combined in one color, giving a rise to a single prime dichotomy, (2 4:1 3). △

3.6. Heuristic Approach to Input Encoding

In general, the optimality of the input encoding depends on compatible graph coloring of the reduced graph of incompatibility of implicants. If this graph is empty, the encoding represented by the essential prime dichotomies is optimum. Since the compatible graph coloring is a more restricted version of a general graph coloring, the problem to be solved is *NP*-hard [8]. A fast heuristic technique has been developed for efficient compatible coloring of G_M.

Definition 10: A dichotomy $D_i = (l_i : r_i)$ is said to be *balanced*, if $| \, |l_i| - |r_i| \, | < 2$. The coloring of the incompatibility graph G_M is called *balanced* if adding a seed dichotomy to a set of compatible dichotomies of the same color, such that the resulting dichotomy D_i is not balanced, is allowed only if there are no other compatible seed dichotomies that could maintain the balance.

The reason for maintaining the balance during the graph coloring can be explained as follows. A 1-b encoding implied by dichotomy $D_i = (l_i : r_i)$ allows to distinguish indexes in l_i from those in r_i. The number of additional bits that are needed to distinguish the indexes within the blocks l_i and r_i are bounded below by $\lceil \log_2 |l_i| \rceil$ and $\lceil \log_2 |r_i| \rceil$, respectively. Those additional bits must be provided by other dichotomies in the cover. Therefore, the minimum number of additional bits needed to distinguish all indexes in D_i is equal to max $\{ \lceil \log_2 |l_i| \rceil, \lceil \log_2 |r_i| \rceil \}$. In order to minimize the number of encoding length, the size of both blocks should be comparable.

Our heuristic graph coloring algorithm assigns colors to vertices of the graph of incompatibility of seed dichotomies $G_M(V, E)$ in certain order. A weight is assigned to each seed dichotomy D_i (node in the graph), that is equal to the number of dichotomies compatible with D_i. Initially all nodes are labeled "uncolored." Among all uncolored nodes, the one with the lowest weight is chosen as a seed, and a color is assigned to it. Next all the nodes that are compatible with the seed are examined as candidates for merging with the seed dichotomy. Two strategies are used for selecting the compatible node at this point. Under one strategy, called *balanced coloring*, any node that maintains balance is chosen and assigned the same color as the seed. The *weighted coloring* strategy chooses a node having the least weight. At this point the candidate set must be modified by deleting those nodes that are incompatible with the newly created dichotomy. The compatibility check is very simple, and requires bitwise OR-ing of two binary vectors representing the two dichotomies.

This procedure continues until all candidate nodes are either colored or deleted. Finally, the nodes that are already colored (with a different color) are similarly considered for merging with the current seed. The entire procedure is then repeated for the remaining uncolored nodes, in the ascending order of their weights, adding as many previously colored nodes as needed.

Notice that this coloring procedure allows a node in graph G_M to be assigned multiple colors. In this sense the compatible graph coloring is equivalent to covering by cliques of the complement of graph G_M [8]. The described procedure is similar to that applied in multiple valued logic minimization UMini [3]. The main difference is that here the nodes of the incompatiblility graph represent dichotomies rather than product implicants, and that checking the compatibility of dichotomies is much simpler than checking the compatibility of cubes (the latter one being equivalent to tautology checking).

Theorem 4: If all columns of the reduced constraint matrix M are distinct, then the binary encoding of each column obtained using our encoding scheme is distinct.

Proof: Consider an arbitrary pair of columns, c_i and c_j, of M. If c_i and c_j have distinct binary patterns, they must differ in at least one row, say r_k. From r_k one can generate a seed dichotomy, $D_s = (l_s : r_s)$, such that $c_i \in l_s$ and $c_j \in r_s$, or vice versa. Therefore, the encoding assigned to those columns must differ in at least one bit. □

IV. Encoding Algorithm

Our compatible graph coloring procedure uses an iterative improvement technique to find the best possible encoding. The main idea of this iterative improvement technique is to split the dichotomies into smaller parts, and then reconstruct the dichotomies by repeating graph coloring with this new set. The idea of splitting the dichotomies is similar to splitting the cubes in UMini logic minimizer; the cubes in the initial cover are split in some systematic way, and then recombined to obtain a minimum Boolean cover [3]. To perform the splitting of dichotomies in a systematic way, a special type of dichotomies is introduced.

Definition 11: A *relatively essential seed dichotomy* is a seed dichotomy which is covered by only one prime dichotomy in the cover. A *relatively nonessential seed dichotomy* is a seed dichotomy which is covered by two or more prime dichotomies in the cover. A *minimal dichotomy* of a prime dichotomy, D_k, in the cover is a dichotomy that is obtained as a union of all relatively essential seed dichotomies of D_k.

Recall that our compatible graph coloring allows a node in the graph G_M to be assigned multiple colors. As a result, some prime dichotomies in the cover may overlap. The relatively nonessential dichotomies are those seed dichotomies which belong to the overlap between two or more prime dichotomies. The relatively essential dichotomies are those covered by only one prime dichotomy, and therefore, can be merged to form minimal dichotomies. The minimal dichotomies and the relatively nonessential seed dichotomies are then used as the graph nodes in the next iteration.

The following procedure describes the iterative encoding algorithm based on heuristic compatible graph coloring.

```
**Iterative_encode**
  generate an irredundant set of seed dichotomies of $M$;
  extract essential prime dichotomies;
  construct the incompatibility graph, $G_M$;
  solve the heuristic compatible graph coloring problem for $G_M$;
  find the cover = {dichotomies}, each associated with one
    color;
```

```
while no improvement in cover cardinality do
    {
    find minimal and relatively nonessential seed dichotomies;
    construct a new incompatibility graph;
    solve the heuristic compatible graph coloring problem;
    find the cover;
    }
cover = cover ∪ { essential prime dichotomies }
for each dichotomy $D_K$ in the cover do
    {
    assign binary codes to $D_k$;
    }
return (encoding).
```

In principle, this iterative improvement procedure is similar to the reduction/expansion procedure of MINI [10] and Espresso [17]. However, the iteration strategy of our algorithm is different. We actually use two types of heuristics in each pass of the compatible graph coloring algorithm: balanced coloring and weighted coloring. The two heuristics are executed alternately in each iteration, complementing each other in selecting the right nodes.

We also have a different stopping rule. The iterative procedure will not stop even if there is no improvement in the solution, and the procedure will be allowed to run from that point for a fixed number of iterations, *inum*, set up by the user. If during these iterations there is no improvement, the procedure stops. Otherwise, the iteration counter is reset. The justification for this stopping rule comes from the observation that the solution space of most of the encoding problems is very complex and has a large number of local optima. Application of this stopping rule allows us to "hill climb" to other local minima. As one pass of the heuristic graph coloring is very fast, this stopping rule does not increase execution time drastically.

V. Results

The heuristic encoding technique presented in this paper has been implemented as a computer program Dichotomy based symbolic Input Encoding Technique (DIET). The program is written in C and runs under both VMS and UNIX operating systems. DIET has been tested on more than 30 industrial FSM examples, available from MCNC benchmark set. The most significant results are reported in Table I, and compared to KISS [4] and NOVA [21]. For each tested example, the table shows the total number of states, *states*, the minimum number of bits in unconstrained encoding, *bits*, the minimum number of bits satisfying all encoding constraints, *cbits*, and the encoding length obtained with KISS, NOVA, and DIET, respectively. The encoding constraints were obtained by performing disjoint minimization using Espresso-MV.

The test results indicate that DIET compares favorably with KISS and NOVA in terms of the CPU time, and is superior to both programs in terms of the encoding length. DIET produced optimum encoding length in most of the tested examples. The optimum solutions were taken from [21]. The success rate (the ratio of the total number of optimum solutions to the total number of tested cases) was 61.1% for KISS, 62.5% for NOVA, and 88.9% for DIET. In one case (*donfile*) the encoding length was drastically shorter than those obtained with the other two programs. (The absolute minimum encoding length satisfying all constraints for this design is not known exactly—the best known solution, obtained with NOVA with *iexact* option was 11, but the search of the entire solution space could not be completed; this is to be compared to the encoding length, 7, obtained with DIET). The memory requirement of our program is significantly smaller than that of KISS. DIET is included as part of the PLA decomposition tool, PLADE [22], recently developed at the University of Massachusetts at Amherst.

TABLE I
ENCODING RESULTS

	states	*bits*	*cbits*	KISS	NOVA*	DIET
bbsse	16	4	6	6	6	6
cse	16	4	5	6	5	5
dk14	7	3	4	4	5	4
dk15	4	2	4	4	4	4
dk16	27	5	7	10	10	8
dk17	8	3	4	4	4	4
dk512	15	4	5	6	6	5
donfile	24	5	**	12	15	7
ex1	20	5	7	7	7	7
ex2	19	5	6	6	6	6
ex3	10	4	5	6	5	5
ex5	9	4	5	5	6	5
ex6	8	3	4	5	5	4
keyb	19	5	7	8	7	8
lion9	9	4	4	4	N/A	4
train11	11	4	5	6	5	5
sand	32	5	**	6	N/A	6
s1	20	5	5	5	5	5
s1a	20	5	5	5	N/A	5
styr	30	5	6	6	9	7

*used with *ihybrid* option to satisfy all constraints.
**optimum solution unknown.

VI. Conclusions

It has been demonstrated in this paper that the constraint input encoding problem is equivalent to a two-level logic minimization problem. We should point out, however, that we do not attempt to transform the encoding problem into an equivalent logic minimization problem, and then use a generic logic minimizer to solve it. Rather, we show the similarity between the two problems, so that the efficient algorithms in one problem can be used in the other problem with no, or little, modification. Each problem, however, has its own critical subproblems, which may have no logical counterpart in the other problem. For example, a typical logic minimization procedure requires frequent checking if a cube intersects an offset, or is included in the cover. These problems are computationally expensive, and are not required in the encoding problem as formulated here.

Unlike other input encoding methods the method described in this paper is well suited for iterative improvement technique. This has been achieved by breaking dichotomies into fundamental parts, from which different prime dichotomies can be reconstructed at subsequent iterations.

The relationship between the input encoding and logic minimization established in this paper should help in better understanding of a more general encoding problem, and the state assignment problem in particular. In this work the state assignment was approximated by the input encoding problem. To accurately address the state assignment problem, however, the encoding constraints imposed by the output part (next state) should be also considered. Future work in this area should, therefore, concentrate on symbolic minimization, whereby the input and output encoding are considered simultaneously. It would be also desirable to investigate the tradeoff between the encoding length and logic complexity.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive suggestions and interesting comments. They are also grateful to Bill Lin of UC Berkeley for providing them with a copy of KISS, which made it possible to compare the results of the two encoding programs.

References

[1] D. B. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 466-472, Aug. 1962.
[2] R. K. Brayton, G. D. Hatchel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Hingham, MA: Kluwer Academic, 1984.
[3] M. J. Ciesielski, S. Yang, and M. A. Perkowski, "Multiple-valued minimization based on graph coloring," in *Proc. 1989 IEEE Int. Conf. on Computer Design, Oct. 1989*; also, in Tech. Rep., TR-89-CSE-4, Dept. of Electrical and Computer Engineering, Univ. of Massachusetts, Amherst, 1989.
[4] G. DeMicheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 269-284, July 1985.
[5] G. DeMicheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 597-616, Oct. 1986.
[6] S. Devadas, A. R. Wang, A. R. Newton, and A. Sangiovanni-Vincentelli, "Boolean decomposition in multi-level logic optimization," in *IEEE Int. Conf. on Computer-Aided Design, Dig. Tech. Papers*, 1988, pp. 290-293.
[7] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential circuits," *IRE Trans. Electron. Comput.*, vol. EC-13, pp. 549-562, Oct. 1964.
[8] M. R. Garey and D. S. Johnson, *Computer and Intractability*, San Francisco, CA: Freeman, 1979.
[9] F. J. Hill and G. R. Peterson, *Introduction to Switching Theory and Logical Design*, New York: Wiley, 1974.
[10] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, pp. 443-458, Sept. 1974.
[11] Z. Kohavi, "Secondary state assignment for sequential machines," *IEEE Trans. Electron. Comput.*, pp. 193-203, June 1964.
[12] Y. S. Kuo, "Generating essential prime implicants for a Boolean function with multiple-valued inputs," *IEEE Trans. Comput.*, vol. 36, pp. 356-359, Mar. 1987.
[13] S. Malik, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Encoding symbolic inputs for multi-level logic implementations, in *Proc. Int. Workshop on Logic Synthesis*, MCNC, May 1989.
[14] E. J. McCluskey, "Minimization of Boolean functions," *Bell Syst. Tech. J.*, vol. 35, pp. 1417-1445, Nov. 1956.
[15] M. A. Perkowski and P. Wu, "A new approach to exact minimization of Boolean functions with multiple-valued inputs," Tech. Rep. 38/1988, DIADES Research Group, Dept. of Electrical Engineering, Portland State Univ., 1988.
[16] M. A. Perkowski, P. Wu, and K. A. Pirkl, "Kuai-Exact: A new approach for multi-valued logic minimization in VLSI synthesis," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 401-404.
[17] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 727-750, Sept. 1987.
[18] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, vol. C-33, pp. 879-894, Oct. 1984.
[19] G. Saucier, "State minimization of asynchronous sequential machines using graph techniques," *IEEE Trans. Comput.*, vol. C-21, pp. 282-288, Mar. 1972.
[20] J. H. Tracey, "Internal state assignment for asynchronous sequential machines," *IEEE Trans. Electron. Comput.*, pp. 551-560, Aug. 1966.
[21] T. Villa and A. L. Sangiovanni-Vincentelli, "Algorithms for state assignments for finite state machines for optimal two-level logic implementations," in *Proc. Int. Workshop on Logic Synthesis*, MCNC, May 1989.
[22] S. Yang and M. J. Ciesielski, "A generalized PLA decomposition with programmable encoder," in *Proc. Int. Workshop on Logic Synthesis*, MCNC, May 1989.

*

Maciej Ciesielski (M'85) received the M.S. degree in electronic engineering from Warsaw Technical University, Warsaw, Poland, in 1974, and the Ph.D. degree in electrical engineering from the University of Rochester in 1983.

From 1983 to 1986 he worked as a Senior Member of Technical Staff at GTE Laboratories, and in 1986 as an Assistant Professor at the University of Lowell, Computer Science Department. In 1987 he joined the University of Massachusetts at Amherst, where he is now an Assistant Professor in the Department of Electrical and Computer Engineering. His research focuses on design automation and VLSI synthesis, and in particular on CAD algorithms and tools for logic and layout synthesis and performance optimization of VLSI circuits.

*

Saeyang Yang received the B.S and M.S degrees in electronic and computer engineering from Korea University, Seoul, Korea, in 1981 and 1985, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Massachusetts at Amherst, in 1990.

Recently he joined the Microelectronic Center of North Carolina (MCNC) as a CAD researcher. At MCNC, he is responsible for the research and development of logic synthesis tools. His research interest includes logic synthesis, testing, high-level synthesis, and neural networks.