Exploiting Communication Complexity for Boolean Matching

Kuo-Hua Wang, TingTing Hwang, *Member, IEEE,* and Cheng Chen

Abstract— **Boolean matching is to check the equivalence of two functions under input permutation and input/output phase assignment. A straightforward implementation takes time complexity $O(n!2^n 2)$, where n is the number of variables. Various *signatures* of variables were used to prune impossible permutations by many researchers. In this paper, based on communication complexity, we also propose two signatures, *cofactor* and *equivalence* signatures, which are general forms of many existing signatures. These signatures are used to develop an efficient Boolean matching algorithm which is based on checking structural equivalence of OBDD's. Experimental results on a set of benchmarks show that our algorithm is indeed very effective in solving Boolean matching problem.**

I. Introduction

Boolean MATCHING is to check the equivalence of two functions under input permutation and input/output phase assignment (so called *NPN-class* [1]). It has been widely used in technology mapping recently [6]–[12]. Applying Boolean matching in technology mapping can improve the quality of mapped circuits and increase the mapping flexibility since it exploits implicit don't cares [2] which was not considered in traditional tree covering algorithm [3]. Moreover, it is able to shorten the mapping time when using a library containing complex gates with large input size. Boolean matching is also applied in logic verification, e.g., checking the equivalence of two circuits, and verifying the implementation of a specification.

Various methods for Boolean matching were proposed [6]–[16]. Mailhot *et al.* [6] are among the first ones to apply Boolean matching to technology mapping. They proposed an algorithm using tautology checking based on shannon decompositions. Symmetry and unateness properties were used to speed up the matching algorithm. Don't cares were considered by a lattice-based method. Savoj *et al.* [7] used smoothing and consensus operators to solve Boolean matching problem. Symmetry of variables was utilized to expedite the matching process. The techniques presented in [10] were based on computing *canonical forms* of functions. If two functions have the same canonical form then they are matched. Boolean unification and branch-and-bound techniques were adopted in [8]. The matching between two functions was checked by finding the *most general unifier* (mgu).

Yet, another group of researchers take "signature" approach to solve Boolean matching. Various signatures [9], [12], [15], [16] were defined to characterize the input variables of Boolean functions, where variables with different signatures can be distinguished from each other and many infeasible permutations can be pruned. The structure of Ordered Binary Decision Diagrams (OBDD's) [4] was also utilized for Boolean matching [11], [13], [14]. In [11], OBDD's were represented by character strings. The matching between OBDD's was checked by comparing their character string representations. In [13] and [14], Boolean matching was designed to transform one OBDD with different orderings until OBDD's of two Boolean functions are graph isomorphism (*structural equivalence*) or failure is reported.

In [13], the subgraphs of OBDD's were matched in a top-down manner (from root to terminal nodes) while in [14], in a bottom-up manner. Using OBDD structure, many infeasible permutations which cannot be identified by signatures can be pruned during the transformation process.

In this paper, we propose a Boolean matching algorithm combining the signature techniques and the transformation method. Our method is similar to that of [14]. However, in [14] only minterms count is used to select variables for transformation during matching process. Our algorithm is based on a more descriptive signatures. It can quickly prune a large number of infeasible matchings.

The remaining of this paper is organized as follows. In Section II, we define structural equivalence of OBDD's and correlate it to Boolean matching problem. Two signatures, cofactor and equivalence, based on communication complexity of Boolean functions are proposed in Section III. Some properties of these signatures are also given. In Section IV, we present a Boolean matching algorithm based on equivalence signature. Some experimental results on a set of benchmarks are shown in Section V. Finally, we give a brief conclusion.

II. Binary Decision Diagrams and Boolean Matching

In this section, we first review OBDD's and the Boolean matching problem. Then we correlate Boolean matching to structural equivalence of OBDD's.

The OBDD of a function f constructed by some variable ordering is denoted as BDD^f. Fig. 1 shows a BDD^f of $f = x_1x_2x_3$ using the variable ordering $x_1 < x_2 < x_3$. In this figure, a rectangle denotes a terminal node with logical value, a

Manuscript received April 22, 1994; revised March 30, 1995 and April 24, 1996. This work was supported by a Grant from the National Science Council of R.O.C. under Contracts NSC-81-0404-E-007-129. This paper was recommended by Associate Editor M. Fujita.

K.-H. Wang and C. Chen are with the Department of Computer Science and Information Engineering, National Chiao Tung University, HsinChu, Taiwan 30050.

T.T. Hwang is with the Department of Computer Science, National Tsing Hua University, HsinChu, Taiwan 30043.

Publisher Item Identifier S 0278-0070(96)07397-6.

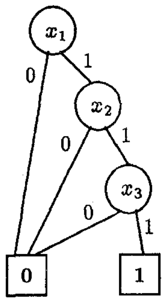


Fig. 1. BDD^f using ordering $x_1 < x_2 < x_3$.

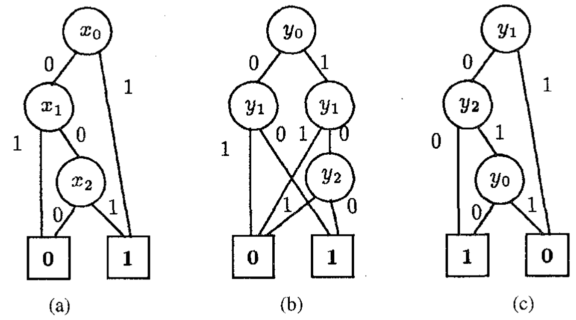


Fig. 2. Structure equivalence of BDD^f of (a) and BDD^g of (c). (a) BDD^f with input ordering of $x_0 < x_1 < x_2$. (b) BDD^g with input ordering of $y_0 < y_1 < y_2$. (c) BDD^g with input ordering of $y_1 < y_2 < y_0$.

circle denotes a nonterminal node labeled by a variable index, and two children are indicated by branches labeled 0 and 1.

Definition 2.1 (Structural Equivalence): Two OBDD's, BDD^f and BDD^g, have *structural equivalence* if 1) they are graph isomorphism, 2) labels of nonterminal nodes of two graphs have one-to-one correspondence, and 3) for all nonterminal nodes with the same index, all corresponding branches have the same values or all corresponding branches have complemented values. It is denoted as $BDD^f \equiv BDD^g$. ■

The *Boolean matching* problem can be stated as follows. Given two functions $f(X)$ and $g(Y)$, where $X = \{x_0, x_1, \cdots, x_{n-1}\}$ and $Y = \{y_0, y_1, \cdots, y_{n-1}\}$, find an *assignment* function ψ which maps x_i to a unique $y_j(\overline{y}_j)$ for each variable $x_i \in X$ such that $g(Y) = f(\psi(X))$ (or $\overline{f}(\psi(X))$.

Boolean matching of two functions can be viewed as searching structural equivalence of OBDD's. Consider two matched functions $f(X), g(Y)$ and an assignment function ψ, where y_j (or $\overline{y}_j$) $= \psi(x_i)$ for $x_i \in X$ and $y_j \in Y$. The effect of x_i on f (or $\overline{f}$) is the same as the effect y_j (or $\overline{y}_j$) on g. Assign x_i and y_j the same order index will result in structural equivalence OBDD's. Therefore, we have the following observation.

Observation 2.1: Let $f(X)$ and $g(Y)$ be two matched functions and $g(Y) = f(\psi(X))$ (or $\overline{f}(\psi(X))$. Suppose BDD^f and BDD^g are constructed by ordering α and β, respectively. If $\beta = \psi(\alpha)$ then $BDD^f \equiv BDD^g$.

We give an example to illustrate this observation.

Example 2.1: Consider two matched functions $f(X) = x_0 + \overline{x}_1 x_2$ and $g(Y) = \overline{y}_0\overline{y}_1 + \overline{y}_1\overline{y}_2$, where $g(Y) = \overline{f}(\psi(X))$, and $\psi(x_0) = y_1, \psi(x_1) = \overline{y}_2$ and $\psi(x_2) = y_0$. The BDD^f with ordering $\alpha = x_0 < x_1 < x_2$ and BDD^g with orderings $y_0 < y_1 < y_2$ are shown in Fig. 2(a) and (b), respectively. By Observation 2.1, we transform initial BDD^g to the other one using the ordering $\psi(\alpha) = y_1 < y_2 < y_0$. The resultant OBDD is shown in Fig. 2(c) which is isomorphic to BDD^f ($BDD^f \equiv BDD^g$). ■

Based on Observation 2.1, the matching of g to f can be viewed as transforming BDD^g with different variable orderings until $BDD^f \equiv BDD^g$ or failure is reported. A straightforward method for solving this problem is to enumerate all possible BDD^g using different variable orderings. Obviously, this exhaustive search is not feasible because it needs $2^n \times n! \times 2$ permutations, where n is the number of inputs. Instead, we propose a signature based algorithm for this transformation.

III. Cofactor and Equivalence Signatures

Many types of signatures have been proposed to speed up Boolean matching [9], [12], [15], [16]. These signatures were used to quickly distinguish inputs of Boolean functions. Based on communication complexity, we also propose two types of signatures which are general forms of many existing signatures. We first describe communication complexity of Boolean functions and show how to use OBDD's in computing communication complexity. Then we define two signatures - cofactor and equivalence signatures based on communication complexity. Some properties of these signatures are then presented.

A. The Communication Complexity of Boolean Functions

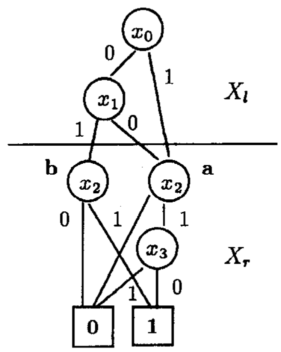
For a function $f(X)$ and its input set $X, onsize(f)$ is the size of on-set, B^X is the Boolean space spanned by X, and a *partition* of the input set X is to partition X into two disjoint sets X_l and X_r which is denoted as $\pi = (X_l, X_r)$.

Definition 3.1 (Communication Complexity): Given a function $f(X)$ and a partition $\pi = (X_l, X_r)$, the Boolean space B^{X_l} can be divided into many *equivalence classes* so that $f(m_1, X_r) = f(m_2, X_r)$ for any two elements m_1, m_2 in the same class. The number of equivalence classes is the *communication complexity* of the function f with respect to the partition π. ■

Given a function $f(X)$ and an input partition $\pi = (X_l, X_r)$, the communication complexity can be computed by counting the number of distinct row patterns in the communication matrix (essentially a truth table) obtained with respect to the input partition π. For example, consider the function $f(X) = x_2\overline{x}_3 + \overline{x}_0 x_1 x_2$ and a partition $\pi = (\{X_l = \{x_0, x_1\}, X_r = \{x_2, x_3\})$. Its communication matrix with respect to π is shown in Fig. 3(a). In this matrix, there are two equivalence classes $E_1 = x_0 + \overline{x}_1, E_2 = \overline{x}_0 x_1$ which correspond to row patterns $\mathbf{A} = \mathbf{0}-\mathbf{0}-\mathbf{1}-\mathbf{0}$ and $\mathbf{B} = \mathbf{0}-\mathbf{0}-\mathbf{1}$-$\mathbf{1}$, respectively. Using communication matrix to compute communication complexity is impractical since it needs to enumerate all $2^{|X_l|}$ rows and check their equivalences,

	$x_0x_1 \backslash x_2x_3$	00	01	10	11	
E_1	00	0	0	1	0	**A**
E_2	01	0	0	1	1	**B**
E_1	10	0	0	1	0	**A**
	11	0	0	1	0	

(a)



(b)

Fig. 3. An example of $f = x_2\overline{x}_3 + \overline{x}_0 x_1 x_2$ w.r.t. $\pi = (\{x_0, x_1\}, \{x_2, x_3\})$. (a) Communication matrix. (b) OBDD's with input ordering of $x_0 < x_1 < x_2 < x_3$.

where $|X_l|$ is the number of variables in X_l. Instead, we propose to use OBDD to compute communication complexity.

Given a Boolean function $f(X)$ and a partition $\pi = (X_l, X_r)$. Let α be a variable ordering which is constructed by the simple ordering rule: all inputs in X_l are ordered before all inputs in X_r. Then the number of nodes *direct below* X_l [5] in BDD^f constructed by ordering α is the communication complexity. Consider the same function f and the partition π in Fig. 3(a). The BDD^f constructed by variable ordering $x_0 < x_1 < x_2 < x_3$ is shown in Fig. 3(b). It has two nodes direct below X_l. The sub-OBDD's rooted at **a** and **b** correspond to pattens **A** and **B** in the communication matrix, respectively.

B. Definitions of the Signatures

Definition 3.2 (Cofactor of Equivalence Class): Given a function $f(X)$ and a partition $\pi = (X_l, X_r)$. Let $E_1, E_2, \cdots, E_m$ be the equivalence classes of f with respect to π. $f_{E_i} = f(X_l = E_i, X_r)$ is defined as the *cofactor of* E_i. That is, $f_{E_i} = f(X_l = E_i, X_r)$ is the result of partially evaluating f for $X_l = E_i$. ∎

Example 3.1: Consider the function $f(X) = x_2\overline{x}_3 + \overline{x}_0 x_1 x_2$ shown in Fig. 3 and partition $\pi = (X_l = \{x_0, x_1\}, X_r = \{x_2, x_3\})$. The cofactor of equivalence class E_1 is $f_{E_1} = x_2\overline{x}_3$, where $E_1 = x_0 + \overline{x}_1$. ∎

We now define communication set.

Definition 3.3 (Communication Set): Given a function $f(X)$ and a partition $\pi = (X_l, X_r)$. Let $E_1, E_2, \cdots, E_m$ be the equivalent classes of f with respect to π. $CS_\pi^f = \{(E_i, cf_i) | cf_i = f_{E_i}$ (cofactor of E_i) for $i = 1$ to $m)\}$ is defined as the *communication set* of f with respect to the partition π. ∎

The cardinality of communication set is identical to the communication complexity. Each element in CS_π^f consists of two parts. The first component represents an equivalence class E_i and is a function of X_l. The second represents the corresponding cofactor cf_i and is a function of X_r.

Example 3.2: Consider the function $f(X)$ shown in Fig. 3 and partition $\pi = (X_l = \{x_0, x_1\}, X_r = \{x_2, x_3\})$. The communication set is $CS_\pi^f = \{(E_1, cf_1), (E_2, cf_2)\}$ where $E_1 = x_0 + \overline{x}_1, E_2 = \overline{x}_0 x_1, cf_1 = x_2\overline{x}_3$ and $cf_2 = x_2(cf_1$ and cf_2 are the sub-OBDD's rooted at **a** and **b**, respectively). ∎

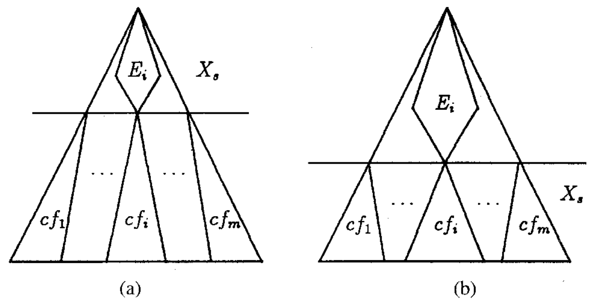


Fig. 4. Two OBBD's for (a) $COFSIG_{X_s}^f$ and (b) $EQUSIG_{X_s}^f$.

Based on the definition of communication set, we define cofactor and equivalence signatures.

Definition 3.4 (Cofactor Signature): Given a function $f(X)$ and a subset $X_s \subset X$, the *cofactor signature* of f with respect to X_s is defined as:

$$COFSIG_{X_s}^f = \{(\text{onsize}(cf_i), E_i) | (E_i, cf_i) \in CS_\pi^f\} \quad (1)$$

where $\pi = (X_s, X - X_s)$.

Definition 3.5 (Equivalence Signature): Given a function $f(X)$ and a subset $X_s \subset X$, the *equivalence signature* of f with respect to X_s is defined as

$$EQUSIG_{X_s}^f = \{(\text{onsize}(E_i), cf_i) | (E_i, cf_i) \in CS_\pi^f\} \quad (2)$$

where $\pi = (X - X_s, X_s)$. ∎

For a given subset $X_s, COFSIG_{X_s}^f$ is a signature to characterize X_s when X_s is ordered on the top part of an OBDD while $EQUSIG_{X_s}^f$ is a signature when X_s is ordered at the bottom part. Fig. 4(a) and (b) shows the OBDD's for computing $COFSIG_{X_s}^f$ and $EQUSIG_{X_s}^f$, respectively, where in Fig. 4(a) X_s is ordered first and in Fig. 4(b) the last.

By examining the communication matrix partitioned with respect to $\pi = (X_s, X - X_s)$ as shown in Fig. 3, $COFSIG_{X_s}^f$ is computed considering row pattern. The number of row patterns is the number of communication complexity with respect to $\pi = (X_s, X - X_s)$. For each element $(\text{onsize}(cf_i), E_i)$ in $COFSIG_{X_s}^f$, the first component is the number of 1's in a row pattern, and the second component is the expression for the row indexes which have the corresponding row pattern. On the contrary, if on the same communication matrix, $EQUSIG_{X_s}^f$ is computed considering column pattern. The number of column patterns is the number of communication complexity with respect to the partition $\pi = (X - X_s, X_s)$. For each element $(\text{onsize}(E_i), cf_i)$ in $EQUSIG_{X_s}^f$, the first component is the number of column indexes which have the same column pattern, and the second component is the expression for the corresponding column pattern.

Example 3.3: For the function $f(X)$ and partition π of Example 3.2 as shown in Fig. 3, $COFSIG_{X_l}^f = \{(1, x_0 + \overline{x}_1), (2, \overline{x}_0 x_1)\}$ and $EQUSIG_{X_l}^f = \{(2, \mathbf{0}), (1, \mathbf{1}), (1, \overline{x}_0 x_1)\}$. ∎

Various existing signatures are special cases of cofactor and equivalence signatures. When $|X_s| = 1, COFSIG_{X_s}^f$ is a *syndrome* signature [15] or a *cofactor signature* [16], and

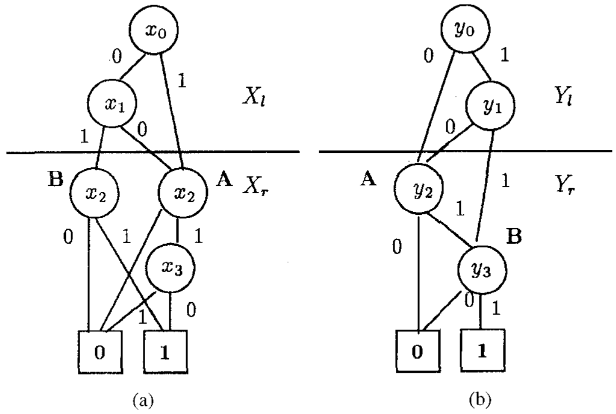


Fig. 5. (a) OBDD's of f and (b) OBDD's of g.

$EQUSIG^f_{X_s}$ is the *partner pattern* [15] or *single fault propagation weight signature* [12]. When $|X_s| = 2, COFSIG^f_{X_s}$ is the *cross signature* [9].

C. Properties of the Signatures

In this section, we present some properties of cofactor and equivalence signatures. These properties are used in our Boolean matching algorithm. First, we define the equivalence of two signatures.

Definition 3.6: Two (cofactor or equivalence) signatures $S_1 = \{(n^1_i, f_i(X)) | i = 1, 2, \cdots, m_1\}$ and $S_2 = \{(n^2_j, g_j(Y)) | j = 1, 2, \cdots, m_2\}$ are equivalent if and only if

1) $m_1 = m_2$, and
2) there exists an assignment ψ to see that each element $(n^1_i, f_i(X)) \in S_1$ corresponds to a unique element $(n^2_j, g_j(Y)) \in S_2$ where $n^1_i = n^2_j$ and $g_j(Y) = f_i(\psi(X))$ (or $\overline{f}_i(\psi(X))$).

This equivalence relation is denoted as $S_1 \equiv S_2$. ∎

Before we present Theorem 3.1, we first have Observation 3.1.

Observation 3.1: Let $CS^f_\pi = \{(E_1, cf_1), (E_2, cf_2), \cdots, (E_m, cf_m)\}$ be the communication set of $f(X)$ with respect to a partition $\pi = (X_l, X_r)$. If $g(Y) = f(\psi(X))$ (or $\overline{f}(\psi(X))$) and $\pi' = (Y_l = \psi(X_l), Y_r = \psi(Y_r))$, then $CS^g_{\pi'}$ can be obtained from CS^f_π by applying ψ to each element of CS^f_π. That is, $CS^g_{\pi'} = \{(E_i(\psi(X_l)), cf_i(\psi(X_r))$ (or $\overline{c}f_i(\psi(X_r))))| i = 1, 2, \cdots, m\}$.

We give an example to illustrate this observation.

Example 3.4: Consider two matched functions $f(X)$ and $g(Y) = f(\psi(X))$, where $\psi(x_0) = \overline{y}_1, \psi(x_1) = y_0, \psi(x_2) = y_3$, and $\psi(x_3) = \overline{y}_2$. The OBDD's of f and g are shown in Fig. 5(a) and (b), respectively. Given the partition $\pi = (\{x_0, x_1\}, \{x_2, x_3\})$ of $X, CS^f_\pi = \{(E_1, cf_1), (E_2, cf_2)\}$, where $E_1 = x_0 + \overline{x}_1, E_2 = \overline{x}_0 x_1, cf_1 = x_2\overline{x}_3$, and $cf_2 = x_2$. By Observation 3.1, $CS^g_{\pi'} = \{(E'_1, cg_1), (E'_2, cg_2)\}$, where $E'_1 = E_1(\psi(X_l)) = \overline{y}_0 + \overline{y}_1, E'_2 = E_2(\psi(X_l)) = y_0 y_1, cg_1 = cf_1(\psi(X_r)) = y_2 y_3$, and $cg_2 = cf_1(\psi(X_r)) = y_3$. ∎

Based on Observation 3.1, Theorem 3.1 is presented.

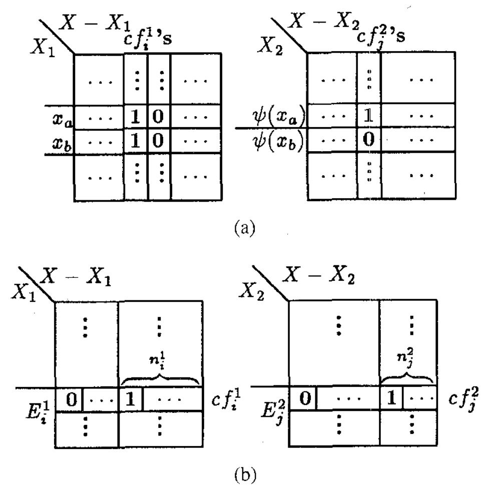


Fig. 6. The communication matrices of f w.r.t X_1 and X_2. (a) For the Case $m_1 \neq m_2$. (b) For the Case $E^1_i(\psi(X_1)) \neq E^2_j(X_2)$ and $n^1_i \neq n^2_j$.

Theorem 3.1: Two matched functions $f(X)$ and $g(Y)$, where $g(Y) = f(\psi(X))$ (or $\overline{f}(\psi(X))$). If $Y_s = \psi(X_s)$ for any subset $X_s \subseteq X$, then

1) $COFSIG^f_{X_s} \equiv COFSIG^g_{Y_s}$.
2) $EQUSIG^f_{X_s} \equiv EQUSIG^g_{Y_s}$.

Proof: Using the procedure implied in Observation 3.1, we can obtain $CS^f_{(X_s, X-X_s)}$ and $CS^g_{(Y_s, Y-Y_s)}$ for any subset $X_s \subseteq X$. This theorem follows obviously. ∎

Theorem 3.1 states the necessary condition for two functions to be matched.

Theorem 3.2: Given a functions $f(X)$. Let X_1 and X_2 be any two subsets of X. If $EQUSIG^f_{X_1} \equiv EQUSIG^f_{X_2}$ then $COFSIG^f_{X_1} \equiv COFSIG^f_{X_2}$.

Proof: We will prove this theorem using communication matrix. Recall that with respect to a given partition $\pi = (X_s, X - X_s)$, the row patterns and the column patterns of the same matrix are used to compute $COFSIG^f_{X_s}$ and $EQUSIG^f_{X_s}$, respectively. Let the matrices partitioned with respect to $\pi = (X_1, X - X_1)$ and $\pi' = (X_2, X - X_2)$ be M_1 and M_2, and $COFSIG^f_{X_1} = \{(n^1_i, E^1_i(X_1)) | i = 1, 2, \cdots, m_1\}, COFSIG^f_{X_2} = \{(n^2_j, E^2_j(X_2)) | j = 1, 2, \cdots, m_2\}$. Suppose that $COFSIG^f_{X_1} \not\equiv COFSIG^f_{X_2}$. The inequality occurs when either the size of the sets are not the same or the elements in the sets are different.

Case 1: $m_1 \neq m_2$

W.l.o.g., we let $m_1 < m_2$. There must exist two elements $x_a, x_b \in X_1$ in the same equivalence class and $\psi(x_a), \psi(x_b) \in X_2$ belong to different classes for any assignment ψ. Since $x_a, x_b \in X_1$ are in the same equivalence class, $f(x_a, X - X_1) = f(x_b, X - X_1)$. Entries of the two rows are the same as shown in the left matrix of Fig. 6(a). However, since $\psi(x_a), \psi(x_b) \in X_2$ are in different equivalence classes and thus they have different row patterns, there must exist an entry of rows where the values of $\psi(x_a)$ and $\psi(x_b)$ are different. The right matrix of Fig. 6(a) shows the case. Now consider the column pattern to compute the equivalence signature. In the left matrix, the entries corresponding to the row index x_a and x_b of all columns will be the same whereas in the

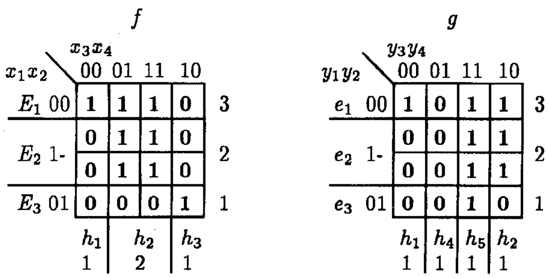


Fig. 7. A counter-example for Property 3.2.

right matrix, there exists at least one column where the entries corresponding to the row index $\psi(x_a)$ and $\psi(x_b)$ are different. Therefore, these two matrices will not have the same column patterns and $EQUSIG^f_{X_1} \not\equiv EQUSIG^f_{X_2}$. Contradict to our assumption.

Case 2: For any assignment function ψ, there exists at least one element where $(n^1_i, E^1_i) \neq (n^2_j, \psi(E^2_j))$.

Two cases for this inequality:

case i: $E^1_i(\psi(X_1)) \neq E^2_j(X_2)$.

The same argument in Case 1 can be applied.

case ii: Suppose that $E^1_i(\psi(X_1)) = E^2_j(X_2)$ and $n^1_i \neq n^2_j$. This implies that the number of 1's in the row whose row index is expression E^1_i in M_1 is different from that of the row whose row index is E^2_j in M_2. Fig. 6(b) shows the matrices, where columns which have 1's at the entry with row index E^1_i or E^2_j, are moved to the right side. The number of columns which have 1's in entries with row index E^1_i is different from that of columns which have 1's in entries with row index E^2_j. There must exist at least one element $(N^1_i, cf^1_i) \in EQUSIG^f_{X_1}$ and one element $(N^2_j, cf^2_j) \in EQUSIG^f_{X_2}$, where $N^1_i \neq N^2_j$. Therefore, $EQUSIG^f_{X_1} \not\equiv EQUSIG^f_{X_2}$. Contradict to our assumption.

Therefore, $COFSIG^f_{X_1} \equiv COFSIG^f_{X_2}$. ■

The converse of this theorem is not true. We show a counter-example in the following.

Example 3.5: Consider $f(X)$ and $g(Y)$ shown in Fig. 7. Let $X_1 = \{x_1, x_2\}$ and $Y_1 = \{y_1, y_2\}$. $COFSIG^f_{X_1} = \{(3, E_1 = \overline{x}_1\overline{x}_2), (2, E_2 = x_1), (1, E_3 = \overline{x}_1 x_2)\}$ and $COFSIG^g_{Y_1} = \{(3, e_1 = \overline{y}_1\overline{y}_2), (2, e_2 = y_1), (1, e_3 = \overline{y}_1 y_2)\}$. We have $COFSIG^f_{X_1} \equiv COFSIG^g_{Y_1}$, where x_1, x_2, x_3, and x_4 map to y_1, y_2, y_3, and y_4, respectively. However, $EQUSIG^f_{X_1} = \{(1, h_1), (2, h_2), (1, h_3)\}$ and $EQUSIG^g_{Y_1} = \{(1, h_1), (1, h_2), (1, h_4), (1, h_5)\}$. Therefore, $EQUSIG^f_{X_1} \not\equiv EQUSIG^g_{Y_1}$. ■

Theorem 3.2 says that any two subsets of variables distinguished by cofactor signatures can be distinguished by equivalence signatures. Therefore, our matching algorithm will be based on equivalence signature rather than cofactor signature.

Theorem 3.3: Given a functions $f(X)$, a subset $X_1 \subset X$ and $X_2 \subset X$, where $X_2 = \psi(X_1)$. If $EQUSIG^f_{X_1} \equiv EQUSIG^f_{X_2}$, then for every subset $X_{s1} \subset X_1$ and $X_{s2} \subset X_2$ where $X_{s2} = \psi(X_{s1}), EQUSIG^f_{X_{s1}} \equiv EQUSIG^f_{X_{s2}}$.

Proof: Let (n, cf^1) and (n, cf^2) be two equivalent elements, where $(n, cf^1) \in EQUSIG^f_{X_1}$ and $(n, cf^2) \in EQUSIG^f_{X_2}$. Suppose that $X_{s1} = X_1 - \{x_a\}$. Fig. 8(a) and (b) show the communication matrices partitioned with respect to $\pi = (X_1, X - X_1)$ and $\pi' = (X_1 - \{x_a\}, X - X_1 \cup \{x_a\})$, respectively. The new columns at the right matrix are obtained by partitioning the old column at the left with respect to $x_a = 0$ and $x_a = 1$. Similarly, for any subset X_{s1}, cf^1 can be partitioned into $2^{|X_1 - X_{s1}|}$ new subcolumns where each subcolumn corresponds to a cofactor with respect to a minterm in $X_1 - X_{s1}$. The same partition can be applied to cf^2 with respect to a minterm in $X_2 - X_{s2}$. Since $cf^1 \equiv cf^2$, the partitioned results are also the same. Therefore, $EQUSIG^f_{X_{s1}} \equiv EQUSIG^f_{X_{s2}}$. The theorem follows.

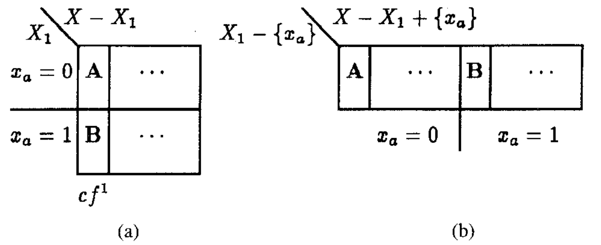


Fig. 8. The communication matrices of f (a) w.r.t $\pi = (X_1, X - X_1)$ and (b) $\pi' = (X_1 - \{x_a\}, X - X_1 + \{x_a\})$.

Based on Theorem 3.3, for a subset $X_s \subseteq X$ matched to a subset $Y_s \subseteq Y$, the larger the subset X_s is taken to compute the equivalence signature, the more efficient it is to match the remaining unmatched inputs.

IV. The Matching Algorithm

Based on Theorem 3.1, 3.2, and 3.3, we develop a transformation based matching algorithm. By Theorem 3.2, any two subsets of variables distinguished by cofactor signature can be distinguished by equivalence signature. Therefore, equivalence signature rather than cofactor signature is used in our algorithm. Given two functions f and g, the algorithm transforms the structure of BDD^g to that of BDD^f.

Initially, for each input of f we first compute the candidate variables for matching. The candidate set is obtained using the equivalence signature for $|X_s| = 1$ and then $|X_s| = 2$. Let $f(X)$ and $g(Y)$ be two functions to be matched. Based on equivalence signature, we distinguish inputs of X into many groups $X_1, X_2, \cdots, X_m$, where the signatures of variables in the same group are equivalent. Then, the same process is applied to g so that Y is also partitioned into groups $Y_1, Y_2, \cdots, Y_m$. If X_i and Y_i have the same signature, Y_i is the candidate set for matching the variables in X_i.

Now we use an example to explain the candidate set generation in more detail. Consider the function $f(x_1, x_2, x_3, x_4) = x_1x_2 + x_1x_3 + x_1\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4$. We first compute $EQUSIG^f_{\{x_i\}}$ for each $x_i \in X$. The communication matrices with respect to inputs x_1, x_2, x_3, x_4 are shown in Fig. 9(a). $EQUSIG^f_{\{x_1\}} = \{(1, f_1), (7, f_2)\}, EQUSIG^f_{\{x_2\}} = EQUSIG^f_{\{x_3\}} = EQUSIG^f_{\{x_4\}} = \{(3, f_0), (1, f_1), (1, f_2), (3, f_3)\}$, where $f_0 = \mathbf{0}$, $f_1 = \overline{x}_i, f_2 = x_i$, and $f_3 = \mathbf{1}$. Thus, input x_1 can be distinguished from x_2, x_3, x_4. Now

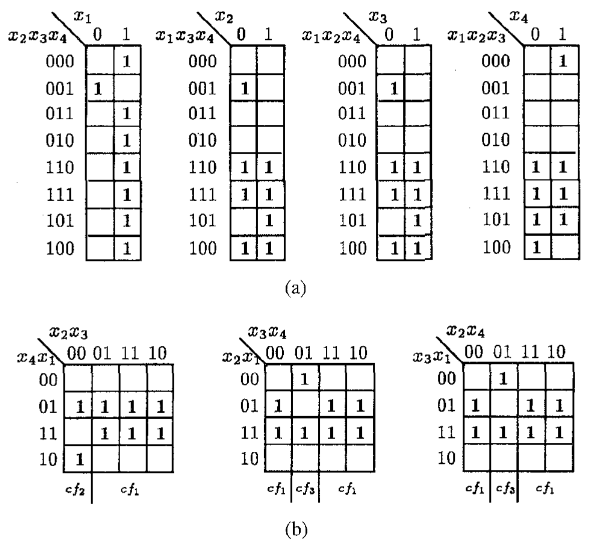


Fig. 9. The communication matrices of f w.r.t. $\pi = (X - X_s, X_s)$. (a) For $X_s = \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}$. (b) For $X_s = \{x_4, x_1\}, \{x_2, x_1\}, \{x_3, x_1\}$.

we compute signatures for $|X_s| = 2$ and $x_1 \in X_s$. The communication matrices with respect to the sets $\{x_2, x_1\}, \{x_3, x_1\}$ and $\{x_4, x_1\}$ are shown in Fig. 9(b). We obtain $EQUSIG^f_{\{x_4,x_1\}} = \{(3, cf_1), (1, cf_2)\}$ and $EQUSIG^f_{\{x_2,x_1\}} = EQUSIG^f_{\{x_3,x_1\}} = \{(3, cf_1), (1, cf_3)\}$. Together, these two signatures distinguish input x_4 from x_2, x_3. Therefore, we partition inputs to three sets $\{x_1\}, \{x_4\}$, and $\{x_2, x_3\}$.

If we continue increasing the size of X_s, we would be able to distinguish all variables of X. The same procedure can then be applied to the other target function. However, it is inefficient in that the signature computations have to be performed twice for both target functions. Instead, after generating and matching candidate sets X_i and Y_i of $f(X)$ and $g(Y)$, we proceed to transform the OBDD structure of g to that of f bottom up.

We first construct BDD^f and BDD^g using the ordering where the indistinguishable inputs are ordered before distinguishable ones. This ordering rule follows Theorem 3.3 where putting distinguishable inputs as many as possible on the bottom of BDD will fasten the distinction of unmatched inputs. Also note matching is possible only between x_i of X_i and y_i of Y_i, where Y_i is the candidate set of X_i. Therefore, variables with the same signature using $|X_s| = 1$ and $|X_s| = 2$ are grouped together on OBDD and their candidate sets are given corresponding order indexes. Fig. 10 shows the initial ordering of BDD^f and BDD^g.

Now, the variable ordering of BDD^f is held fixed, we transform BDD^g with different orderings until $BDD^f \equiv BDD^g$ or failure is reported. The transformation process on BDD^g starts with the first candidate set of indistinguishable inputs bottom up. Let X_d and Y_d be the sets of distinguishable inputs which are ordered at the bottom of BDD^f and BDD^g, and $x_0 < x_1 < \cdots < x_m$ be the ordering of the indistinguishable variables in X. Let x_i in $X_l$be the next variable to be matched. Initially, i is set to m. For the candidate set Y_l, we compute $EQUSIG^g_{Y_d \cup \{y_j\}}$ for each $y_j \in Y_l$. If there is no equivalence signature, failure of matching is reported. If there is a unique $EQUSIG^g_{Y_d \cup \{y_j\}} \equiv EQUSIG^f_{X_d \cup \{x_i\}}$, then x_i is matched to y_j. If there are more than one equivalence signatures, we select an arbitrary one for matching. More than one equivalence signatures happens when variables are symmetric or they are indistinguishable using equivalence signatures. For the former case, arbitrarily selecting one variable for matching is always correct since these variables are symmetric. For the latter case backtrack may be required.

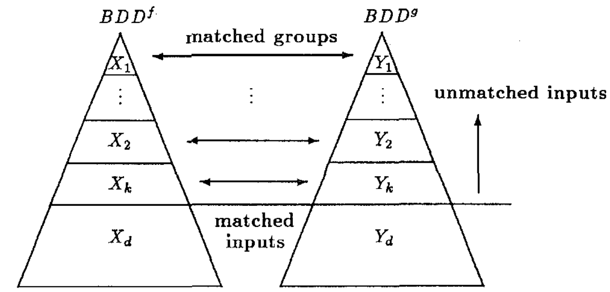


Fig. 10. The Transformed BDD^f and BDD^g.

After matching one variable, y_j to x_i, we set $X_d = X_d \cup \{x_i\}, Y_d = Y_d \cup \{y_j\}$, and i is decreased by one. The procedure continues until all variables in the candidate set are matched. If the candidate set can not be matched and there are backtrack points, then the procedure backtracks to the nearest point and restarts the matching procedure. Note that backtrack may occur only within each candidate set. Candidate sets are matched one by one bottom up until all variables are processed. Fig. 11 shows the matching algorithm. The inputs to this algorithm are two functions $f(X), g(Y)$ (BDD^f, BDD^g). It returns *Success* if f and g are matched; otherwise returns *Failure*. The sizes of on-sets of f and g (and $\overline{g}$) is checked at the beginning of the procedure to prune unmatched functions first. The transpositional operator [17] which restructures OBDD with different ordering is used in transforming BDD.

The time complexity of our algorithm mainly consists of three parts. The first part is the complexity of generating candidates sets. A procedure based on transpositional operator which takes time complexity $O(p^2)$ [17], where p is the size of OBDD, is used in computing equivalence signatures. The total time complexity of this part is $O(n \times (p^2 + q^2))$, where n, p, q are the number of inputs, the sizes of BDD^f and BDD^g, respectively. The second part is to construct BDD^f and BDD^g using a constrained ordering. It takes $O(n \times (p^2 + q^2))$. The last part is the time for transformation of BDD^g. The worst number of transformations is $\Sigma_{i=1}^k |X_i|!$, where k is the number of candidates sets whose size is greater than 1. For each transformation, transpositional operator [17] is applied. Summing up these three parts, the complexity of our algorithm is $O(n \times (p^2 + q^2) + (\Sigma_{i=1}^k |X_i|!) \times q^2)$. In fact, from the experiments, we find that inputs can be distinguished after the candidate set is generated for most cases. Therefore, the time complexity is $O(n \times (p^2 + q^2))$ in practice.

```
Algorithm Boolean-Matching(f(X), g(Y))
Input: f(X), X = {x_0, x_1, ..., x_{n-1}};
       g(Y), Y = {y_0, y_1, ..., y_{n-1}};
Output: return Success if f and g are matched; otherwise, return Failure;
Begin
    if (onsize(f) ≠ onsize(g(ḡ))) then
        return Failure;
    endif
    Generate candidate sets for variables in X;
    Let X_d and Y_d be the sets of distinguishable inputs;
    Construct BDD^f and BDD^g;      /*  Push down the distinguishable inputs  */
    i = m;
    while (i ≤ m) and (i ≥ 0) do
    next:
        choose an unmasked input y_j ∈ Y_l, where x_i ∈ X_l;
        if (no such an input exists) then
            unmask unmatched inputs;
            i = i + 1;           /* Backtracking */
            if (backtract to other group) then
                return Failure;
            endif;
        else
            mask y_j;
            if (EQUSIG^f_{X_d ∪ {x_i}} ≡ EQUSIG^g_{Y_d ∪ {y_j}}) then
                X_d = X_d ∪ {x_i};
                Y_d = Y_d ∪ {y_j};
                i = i − 1;           /* Next level matching */
            else
                goto next;           /* Choose next input in Y_l */
            endif;
        endif;
    endwhile;
    return Success;
End
```

Fig. 11. The *Boolean-Matching* Algorithm.

TABLE I
EXPERIMENTAL RESULTS FOR BOOLEAN MATCHING

circuits	*#in*	*#out*	$\lvert X_i \rvert = 1$	$\lvert X_i \rvert > 1$	*#errors*	*Matching_ CPU*	*BDD_const._ CPU*
5xp1	7	10	7	0	0	0.3	0.5
*act1	8	1	8	0	0	0.0	0.0
*act2	8	1	8	0	0	0.0	0.0
alupla	25	5	25	0	0	20.1	1.1
apex7	49	37	49	0	0	7.1	0.4
cm138a	6	8	6	0	0	0.0	0.0
*cm150a	21	1	3	(4,4,4,6)	4	28.1	0.1
*cm151a	12	1	2	(3,3,3)	1	2.5	0.0
cm162a	14	5	14	0	0	0.1	0.0
cm163a	16	5	16	0	0	0.1	0.0
comp	32	3	32	0	0	3.3	0.1
cordic	23	2	23	0	0	82.2	0.3
count	35	16	35	0	0	3.9	0.4
cu	14	11	14	0	0	0.7	0.2
duke2	22	29	22	0	0	9.2	0.9
f51m	8	8	8	0	0	0.2	0.3
frg1	28	3	28	0	0	5.4	1.0
lal	26	19	22	0	0	0.9	0.5
misex2	25	18	25	0	0	0.9	0.2
misex3	14	14	14	0	0	2.7	0.3
pcler8	27	17	27	0	0	0.8	0.2
pm1	16	13	16	0	0	0.1	0.2
sao2	10	4	10	0	0	4.2	0.5
seq	42	35	42	0	0	20.3	2.8
t481	16	1	0	(4,4,4,4)	1	35.6	21.5
term1	34	10	25	(5)	0	29.3	2.7
ttt2	24	21	24	0	0	0.6	0.7
unreg	36	16	36	0	0	0.8	0.3
vg2	25	4	25	0	0	19.2	0.8
z4ml	7	4	7	0	0	0.2	0.0

TABLE II
THE COMPARISON RESULTS

circuits	*Ours*		[16]	
	indist. inputs	*CPU*	*indist. inputs*	*CPU*
act1	0	0.0	0	0.1
act2	0	0.0	(2,2)	0.0
cm150a	(4,4,4,6)	5.6	(4,4,4,6)	3.1
cm151a	(3,3,3)	0.7	(3,3,3)	0.6
cordic	0	42.8	(2,2)	32.2
cu	0	0.1	(2)	0.3
lal	0	0.5	(2,2)	0.4
sao2	0	1.0	(2,2,2,2)	0.8
t481	(4,4,4,4)	8.6	(4,4,4,4)	2.6
term1	(5)	17.3	(5,5,5,2,2)	54.1
vg2	0	8.9	0	16.1

V. EXPERIMENTAL RESULTS

The proposed Boolean matching algorithm has been implemented in C language on SUN Sparcstation IPC (a 15.7 mips machine). To demonstrate the efficiency of our algorithm, circuits from MCNC benchmark set have been tested. Two circuits, act1 and act2, of *actel1* and *actel2* cells from FPGA manufacturer *Actel* were also included in the test set. For each circuit, we first constructed two OBDD's. The second OBDD was generated from the first one by permuting and renaming its input variables. Then we applied our matching algorithm to transform the second OBDD until these two OBDD's are matched.

Table I shows the experimental results. The columns with labels *#in* and *#out* show the numbers of inputs and outputs of circuits, respectively. The column $|X_i| = 1$ refers the number of inputs which could be distinguished from the other inputs. The column labeled $|X_i| > 1$ refers the sizes of candidate sets whose sizes are greater than 1 when equivalence signature for $|X_s| = 1$ and $|X_s| = 2$ are used. The column *#error* shows the number of variables which were incorrectly matched during the matching process. The columns *matching_CPU* and *BDD_const._CPU* show the running time of our matching algorithm and OBDD's construction time, respectively. The CPU time is measured in seconds by using *time* command of *MIS* [18]. The table shows that all inputs of 26 circuits could be distinguished using signatures only. Candidate-set size of only 4 circuits is bigger than 1. We also find that number of error variable selections during the matching process is very small. The reason is that when the inputs are not distinguishable, it often involves many legal assignments. For example, cm150a (cm151a) has $4! = 24(3! = 6)$ legal assignments since it is a 16 to 1 (8 to 1) multiplexer. Our algorithm is designed to choose any assignment. The running time of our matching algorithm is also short. In many cases, we have a very small amount CPU time compared to the construction time of OBDD's.

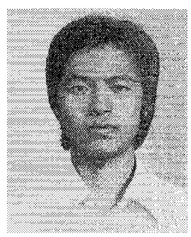
In [16], **cofactor** and **breakup** signatures were proposed to distinguish inputs. We compare our results with the results shown in [16]. Table II shows the comparison results. The column labeled *in. inputs* refers the sizes of indistinguishable inputs. This table shows our algorithm can distinguish all inputs of 6 circuits out of 9 circuits of which inputs were not all distinguished by [16]. The CPU time of [16] is measured in seconds on a SUN Sparcstation SLC (a 18 mips machine).

VI. Conclusion

We have proposed a signature based Boolean matching algorithm. It transforms OBDD's using different orderings until two target OBDD's have the same structure or failure is reported. *Equivalence* and *cofactor* signatures which are general forms of many existing signatures are presented to speed up this transformation process. Experimental results on a set of benchmarks show that our algorithm is indeed very effective in Boolean matching problem.

References

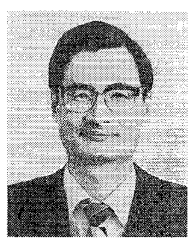
[1] S. Muroga, *Threshold Logic and its Applications.* New York: Wiley-Intersci., 1971.
[2] K. A. Bartlett, R. K. Brayton *et al.*, "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 723–740, June 1988.
[3] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in *Proc. DAC24,* June 1987, pp. 341–347,
[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.,* vol. C-35, pp. 677–691, Aug. 1986.
[5] C. L. Berman, "Circuit width, register allocation, and ordered binary decision diagrams," *IEEE Trans. Computer-Aided-Design,* vol. 10, pp. 1059–1066, Aug. 1991.
[6] F. Mailhot and G. De Micheli, "Technology mapping using boolean matching and don't care sets," in *Proc. EDAC'90*, 1990, pp. 212–216.
[7] H. Savoj, M. J. Silva, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Boolean matching in logic synthesis," in *Proc. Euro-DAC'92*, 1992, pp. 168–174.
[8] K.-C. Chen, "Boolean matching based on Boolean unification," in *Proc. Euro-DAC'93*, 1993, pp. 346–351.
[9] U. Schlichtmann, F. Brgkexz, and P. Schneider, "Efficient Boolean matching based on unique variable ordering," in *IWLS'93*.
[10] J. R. Bruch, and D. Long, "Efficient Boolean function matching," in *Proc. ICCAD'92*, 1992, pp. 408–411.
[11] K. Zhu. and D. F. Wong, "Fast Boolean matching for field programmable gate arrays," in *Proc. Euro-DAC'93*, 1993, pp. 352–357.
[12] U. Schlichtmann, F. Brglez, and M. Hermann, "Characterization of Boolean functions for rapid matching in EPGA technology mapping," in *Proc DAC'92*, 1992, pp. 374–379.
[13] Y. T. Li, S. Sarma, and P. Massoud, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," in *Proc. ICCD'92*, 1992, pp. 452–458.
[14] Y. Matsunaga, "A new algorithm for Boolean matching utilizing structural information," in *SASIMI'93*, 1993, pp. 366–373.
[15] D. I. Chen and M. Marek-Sadowska, "Verifing equivalence of functions with unknown input correspondence," in *Proc. EDAC'93*, 1993, pp. 81–85.
[16] J. Mohnke, and S. Malik, "Permutation and phase independent Boolean comparison," in *Proc. EDAC'93*, 1993, pp. 86–92.
[17] K. H. Wang, T.-T. Hwang, and C. Chen, "Restructuring binary decision diagrams based on functional equivalence," in *Proc. EDAC'93*, Feb. 1993, pp. 261–265.
[18] R. K. Brayton *et al.*, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.



Kuo-Hua Wang received the Ph.D. degree from the Institute of Computer Science and Information Engineering, National Chiao Tung University, HsinChu, Taiwan, in 1994.

His current research interests include logic synthesis and optimization, logic verification, and high-level synthesis.

TingTing Hwang (M'90) for a photograph and biography, see this issue, p. 1236.



Cheng Chen received the M.S. degree from the Institute of Electronics, National Chiao Tung University (NCTU), Taiwan, R.O.C. in 1971.

Currently he is a Professor with the Institute of Computer Science and Information Engineering, NCTU. From 1972 to 1977, he was an instructor with the Department of Computer Science, NCTU and from 1977 to 1981, he was an Associate Professor with the Department of Computer Engineering. During the academic year 1980, he was a visiting scholar with the University of Illinois at Urbana-Champaign. He then became full professor with the Department of Computer Engineering from 1981 until the present. From 1987 to 1988, he was the Chairman of the Department of Computer Engineering, NCTU. During the academic year 1988, he was a Visiting Scholar at Carnegie Mellon University, Pittsburgh, PA. Currently, he is also the Deputy Director of Microelectronic and Information System Research Center, NCTU. His research interests include computer architecture, parallel processing, compiling techniques for RISC and superscalar systems, high performance inference machines.

Professor Chen is a member of the IEEE Computer Society.