A Canonical-Based NPN Boolean Matching Algorithm Utilizing Boolean Difference and Cofactor Signature

JULING ZHANG[1], **GUOWU YANG**[1], **(Member, IEEE),**
WILLIAM N. N. HUNG[2], **(Member, IEEE), AND JINZHAO WU**[3]

[1]Big Data Research Center, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2]Synopsys Inc., Mountain View, CA 94043 USA
[3]Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Guangxi University for Nationalities, Nanning 530006, China

Jinzhao Wu (gxmdwjzh@aliyun.com).

ABSTRACT This paper presents a new compact canonical-based algorithm to solve the problem of single-output completely specified input negation and/or input permutation and/or output negation Boolean matching. We propose a new signature vector Boolean difference and cofactor signature vector. Our algorithm utilizes the Boolean difference, cofactor signature, and symmetry properties to search for canonical transformations. The use of symmetry and Boolean difference notably reduces the search space and speeds up the Boolean matching process compared with the algorithm proposed by Adbollahi and Pedram. We tested our algorithm on a large number of circuits. The experimental results showed that the average runtime of our algorithm 37% higher and its average search space 67% smaller compared with Adbollahi and Pedram, when tested on general circuits.

INDEX TERMS Boolean difference, boolean matching, NPN equivalent, symmetry.

I. INTRODUCTION

Judging whether two Boolean functions are equivalent under input negation and/or input permutation and/or output negation (NPN) is an important problem applied in integrated circuit design, logic synthesis, logic verification, and so on. In cryptography, affine equivalence is used to resolve S-box problem [2]. Technology mapping is the process of selecting logic gates from a library to implement a Boolean circuit [3]–[5]. However, every Boolean function has many NPN-equivalent Boolean functions. Technology mapping searches for an optimal combination of logic gates in terms of area, performance and power dissipation. In the cell-library binding process, some of the cells of a library are found to realize some part of a multiple-level representation of a Boolean function [6]. Two Boolean functions f and g are NPN equivalent if a transformation T exists that can transform f to g. In technology mapping and cell-library binding, Boolean matching is a key step.

In recent years, numerous methods have emerged to solve the NPN Boolean matching problem. The main approaches focus on the canonical-based, pairwise and SAT-based algorithms. In the canonical-based algorithms, two Boolean functions are NPN-equivalent when they have the same canonical form. An NPN-equivalent class has a canonical representative, and all the Boolean functions in an NPN-equivalent class can be transformed to this canonical representative. The canonical representative may have a maximal truth table or a minimal truth table or a maximal signature vector [1], [6]–[12].

A pairwise Boolean matching algorithm searches the correspondence relation of the variables of two Boolean functions by their signatures. In the pairwise Boolean matching process, the search operation terminated when it finds a transformation T that can transform one Boolean function into the other. The authors of [13]–[16] presented a pairwise Boolean matching algorithm.

The NPN Boolean matching problem is converted to one that involves solving Boolean satisfiability (SAT) problem. To check whether a Boolean function f is equivalent to a Boolean function g, the SAT-based algorithm first constructs a circuit with the functionality $f \oplus g$ and then generates a SAT instance (circuit CNF). Finally, it judges the Boolean

satisfiability of this circuit CNF [17]. When the circuit CNF is satisfied, f is not NPN-equivalent to g. In contrast, f is NPN-equivalent to g when the circuit CNF is not satisfied.

It is well known that the search space complexity of NPN Boolean matching is $n!2^{n+1}$ using the exhaustive method. Therefore, the exhaustive method is computationally infeasible when the number of inputs is large, and thus, it is important to reduce the search space regardless of what approach is used. Many properties of Boolean functions are used to reduce the search space, including positive unate, negative unate, cofactor and symmetry, and so on.

This paper studies canonical-based NPN Boolean matching for single-output completely specified Boolean functions. We exploit a new compact canonical form using a DC signature vector. The canonical representative of an NPN equivalent class has the maximal DC signature vector. The use of the symmetry class, independent class and Boolean difference signature dramatically reduce the search space of the canonical transformation. The experimental results reflect that using the Boolean difference signature is highly conductive for distinguishing variables. The proposed algorithm enhances the NPN Boolean matching speed and reduces the search space significantly.

The remainder of this paper is organized as follows. In Section II, we survey the state of the Boolean matching problem. Section III introduces related preliminaries and definitions used in our algorithm. Section IV presents our canonical-based algorithm in detail. We demonstrate the effectiveness of our algorithm by presenting experimental results in Section V, and we summarize our work and outline future work in Section VI.

II. RELATED WORKS

Numerous scholars have contributed to this problem in prior Boolean matching studies. The authors of [1], [6]–[12] studied the canonical-based Boolean matching algorithm and proposed many practical methods for computing the canonical representative. The authors of [7] studied the P-equivalence problem and proposed a formal framework that unified the spectral and canonical-based approaches. They found a linear transformation to improve the speed of P-equivalence matching. Chai and Kuehlmann of [8] designed a fast Boolean matching that used satisfied counts to assign the phases of the Boolean function and its variables, the sums of rows or columns to search permutation, and symmetry to refine transformations. In [6], a canonical-based algorithm was presented that used table look-ups and a tree-based breadth-first search to find the canonical representative with the smallest value in the binary representation. Ciric and Sechen [9] proposed the minimum-cost canonical form for P-equivalence matching. In [10], the authors unified multiple canonical-based approaches and proposed a new P-equivalence matching algorithm. Generally, in the canonical-based Boolean matching algorithms, the canonical representative has the maximal or minimal truth table. Abdollahi and Pedram [1] exploited a new canonical form

using signature vector to resolve the NPN Boolean matching problem. The algorithms proposed in [11] and [12] used general symmetry and higher-order symmetry to realize fast NPN Boolean classification. They classified a number of Boolean functions with 6-16 inputs.

Many Boolean matching approaches have emerged based on SAT. In a SAT-based Boolean matching algorithm, Boolean matching is converted to a SAT or UNSAT problem. The authors of [17]–[22] all studied Boolean matching using the SAT technique. The authors of [18] proposed PP-equivalence matching based on graphs, simulation and SAT; their approach can be applied to large-scale circuits. Matsunaga [19] presented a Boolean matching method for LUT-based circuits by using one-hot encoding and the CEGAR technique to speed up Boolean matching [19]. The authors of [20] exploited architectural symmetry in PLB and proposed a SAT-based Boolean matching algorithm to resolve FPGA technology mapping. Wang *et al.* [17] integrated the simulation and SAT technique, and proposed a P-equivalent Boolean matching approach.

A pairwise Boolean matching algorithm was presented in [13] for multiple-output Boolean functions with don't care sets and applied Boolean matching to technology mapping to optimize circuits. Abdollahi [14] proposed a signature-based Boolean matching to resolve NPN-equivalent problem for single-output incompletely specified Boolean functions. They tested their Boolean matching approach on 4-10 inputs Boolean functions. The authors of [15] utilized a level-first strategy and a set of filters to reduce the search space in the pairwise Boolean matching process, while [16] unified the structural signature of Boolean functions and Shannon expansion to exploit a pairwise Boolean matching algorithm.

A few other methods exist for solving the Boolean matching problem. In [23], the authors studied NPN Boolean matching using a Walsh Spectral Decision Diagram, and achieved an efficiency higher than that of Luks' hypergraph method. Lai *et al.* [24] utilized a conflict-driven learning method to solve the multiple-output Boolean function matching problem.

III. PRELIMINARIES AND PROBLEM STATEMENT

Let $f(x_1, x_2, \cdots, x_n)$ be a single-output completely specified Boolean function and $X = (x_1, \cdots, x_n)$ be a vector of f. $|f|$ denotes the number of minterms in f, which is also called the 0th-order signature.

$X_\pi(x_1, \cdots, x_n) = (x_{\pi(1)}, \cdots, x_{\pi(n)})$ is an input permutation and $X_\varphi(x_1, \cdots, x_n) = (x_1^{\varphi(1)}, \cdots, x_n^{\varphi(n)})$ is an input negation. $x_i^{\varphi(i)} = x_i$ when $\varphi(i) = 1$, and $x_i^{\varphi(i)} = \bar{x}_i$ when $\varphi(i) = 0$. NPN equivalence is introduced by $f \equiv g \Leftrightarrow f = (g \circ X_\pi \circ X_\varphi)^\phi$, where $\varphi \in \mathcal{B}^n (\phi \in \mathcal{B})$ [25].

Definition 1 (NP Transformation): An NP transformation, T, is an onto mapping where $T = \{x_1, \cdots, x_n\} \mapsto \{x_{\pi(1)}^{\phi(1)}, \cdots, x_{\pi(n)}^{\phi(n)}\}$.

For each Boolean function f in an NPN-equivalent class with a canonical representative $\mathbb{F}$, there must be an NP

transformation $\mathcal{T}$ that can transform f to $\mathbb{F}$, (i.e., $f(\mathcal{T}X) = \mathbb{F}$ or $\overline{f(\mathcal{T}X)} = \mathbb{F}$). The $\mathcal{T}$ is called a canonical transformation. Boolean function f_1 is NPN-equivalent to f_2 if and only if they have the same canonical representative (i.e., $\mathbb{F}_1 = \mathbb{F}_2$).

A cofactor is a generalized signature of a Boolean function that has been used in many Boolean matching algorithms.

Definition 2: (Cofactor Signature): The cofactor signature of f with respect to $x_i(\overline{x}_i)$, $|f_{x_i}|(|f_{\overline{x}_i}|)$, is the onset size of $f_{x_i}(f_{\overline{x}_i})$, where $f_{x_i} = f[x_i \leftarrow 1]$ and $f_{\overline{x}_i} = f[x_i \leftarrow 0]$.

The cofactor signature of f with respect to a cube b, $|f_b|$, is the onset size of f_b, where $f_b = f[b \leftarrow 1]$. If the cube b has k variables, $|f_b|$ is denoted as the kth-order signature [1].

A Boolean function may have some independent variables; however, the value of a Boolean function is irrelevant to these independent variables. The negative cofactor and positive cofactor of a variable are used to check its independence.

Definition 3: (Boolean Difference Signature): The Boolean difference signature of f with respect to $x_i, |f'_{x_i}|$, is the onset size of f'_{x_i}, where $f'_{x_i} = f_{x_i} \oplus f_{\overline{x}_i}$ [26].

A variable x_i is said to be an independent variable when it satisfies $|f'_{x_i}| = 0$. The independence property of an independent variable does not changed under NP transformation.

Definition 4: (Variable Symmetry): The variables x_i and $x_j(\overline{x}_j)$ of a Boolean function f are symmetric if f is invariant after swapping x_i and $x_j(\overline{x}_j)$ [26].

Two NP transformations are equal when two or more symmetric variables are permuted. Therefore, symmetry is always used to reduce the search space for Boolean matching.

IV. CANONICAL-BASED BOOLEAN MATCHING

The goal of our algorithm is to find the canonical transformation $\mathcal{T}$ of a Boolean function f, which can transform f to $\mathbb{F}$ that has the maximal DC signature vector. During the search process, we search many candidate canonical transformations by sorting and grouping variables. Then, we compare the DC signature vectors of these candidate canonical transformations to find the target canonical transformation $\mathcal{T}$.

A. THE PROPOSED CANONICAL FORM

The canonical form presented in [1] has the maximal signature vector. For some Boolean functions, many variables have the same cofactor signature and are not symmetric variables. In this case, the algorithm proposed in [1] will generate multiple splitting ways, which increases the search space of canonical transformation. Therefore, to reduce the search space, further methods to distinguish these variables are required.

Although the variables of many Boolean functions have the same cofactor signatures, their Boolean difference signatures are the different. Therefore, we proposes DC signature vector that can resolve the shortcoming in [1] described above.

Definition 5: ($\prec$): Let "$\prec$" denote the lexicographic comparison of two vectors.

Definition 6: (1st DC Signature Value): The 1st Boolean Difference and Cofactor (DC) signature value of a Boolean function f with respect to x_i is a two-tuple $(|f_{x_i}|, |f'_{x_i}|)$.

The 1st DC signature values of the two variables x_i and x_j of a Boolean function f are $(|f_{x_i}|, |f'_{x_i}|)$ and $(|f_{x_j}|, |f'_{x_j}|)$, respectively. When they satisfy one of the two following cases, then $(|f_{x_i}|, |f'_{x_i}|) \prec (|f_{x_j}|, |f'_{x_j}|)$ and the relation of x_i and x_j is $x_i < x_j$.

1) $|f_{x_i}| < |f_{x_j}|$
2) $|f_{x_i}| = |f_{x_j}|$ and $|f'_{x_i}| < |f'_{x_j}|$

Definition 7: (kth-order DC Signature Value): The kth-order DC signature value of a Boolean function f with respect to a cube b, $b = x_{i1} \cdots x_{ik}$, is $(|f_{x_{i1} \cdots x_{ik}}|, |f'_{x_{i1} \cdots x_{ik}}|)$.

Definition 8: (DC Signature Vector): The DC signature vector of an n-input Boolean function f is

$$D^f = \{|f|, (|f_{x_1}|, |f'_{x_1}|), \cdots, (|f_{x_n}|, |f'_{x_n}|), (|f_{x_1 x_2}|, |f'_{x_1 x_2}|),$$
$$\cdots, (|f_{x_{n-1} x_n}|, |f'_{x_{n-1} x_n}|), \cdots, (|f_{x_1 \cdots x_{n-1}}|, |f'_{x_1 \cdots x_{n-1}}|),$$
$$(|f_{x_2 \cdots x_n}|, |f'_{x_2 \cdots x_n}|), |f_{x_1 \cdots x_n}|\},$$

which is composed of its 0th-order signature, 1st DC signature values and higher order DC signature values up to the nth-order signature.

The authors of [1] proved that each Boolean function has a unique signature vector. Therefore, each Boolean function also has a unique DC signature vector. We need not prove that here.

Definition 9: (The Canonical Representative): The canonical representative of an NPN-equivalent class $EC = \{f_1, \cdots, f_m\}$ is the Boolean function that has the maximal DC signature vector.

Assume that two Boolean functions f and g have the DC signature vectors D^f and D^g. When comparing D^f and D^g, we compare their 0th-order signature first, then their 1st DC signature values, 2nd-order DC signature values and higher DC signature values until an inequality is encountered. $D^f \prec D^g$ when the corresponding DC signature value of D^f is less than that of D^g. If and only if $D^f \prec D^g$, the order relation of f and g is $f < g$.

Definition 10: (The Maximal Canonical Transformation): The transformation $C_f = (t_1, t_2, \cdots, t_n)$ that has the maximal DC signature vector is the maximal canonical transformation.

In this paper, we use $T = (t_1, t_2, \cdots, t_n)$ to denote a candidate canonical transformation searched in a canonized process where $t_i = x_{\pi(i)}^{\varphi(i)}$. For a candidate canonical transformation T, the DC signature vector is as follows:

$$D^T = \{|f|, (|f_{t_1}|, |f'_{t_1}|), \cdots, (|f_{t_n}|, |f'_{t_n}|), (|f_{t_1 t_2}|, |f'_{t_1 t_2}|),$$
$$\cdots, (|f_{t_{n-1} t_n}|, |f'_{t_{n-1} t_n}|), \cdots, (|f_{t_1 \cdots t_{n-1}}|, |f'_{t_1 \cdots t_{n-1}}|),$$
$$(|f_{t_2 \cdots t_n}|, |f'_{t_2 \cdots t_n}|), |f_{t_1 \cdots t_n}|\}$$

In the canonized process, multiple candidate canonical transformations similar to T exist. The maximal canonical transformation C_f is the candidate canonical transformation that has the maximal DC signature vector. Consider two candidate canonical transformations $T1$ and $T2$: $T1 < T2$ when $D^{T1} \prec D^{T2}$.

For an n-input Boolean function f, $n!2^n$ NP transformations exist. The canonical transformation $\mathcal{T}$ of a Boolean function

f is the transformation that can transform f to $\mathbb{F}$ when $\mathbb{F}$ has the maximal DC signature vector in its NPN-equivalent class. We use C_f to express the maximal canonical transformation of Boolean function f found in the canonical process. The input phase assignment and reindexing transformation is $X = (x_1, \cdots, x_n)$ to $C_f = (t_1, \cdots, t_n)$, where $\mathbb{F}(C_f) = f(X)$. The relation between f and $\mathbb{F}$ can be expressed by an NP transformation $\mathcal{T}$, where $X = \mathcal{T}C_f$ [1]. Therefore, the canonical form of a Boolean function f is $f(\mathcal{T}X)$, and $f(\mathcal{T}X) = f(C_f{}^{-1}X)$.

The key step of our algorithm is sorting and grouping the variables according to their DC signature values. After each grouping, a group result is obtained. We denote G as the group result of f, and $G = \{G_1, \cdots, G_m\}$, where $m \in \{1, \cdots, n\}$. Every group G_i has one or more classes, and each class has one or more variables. We use $G_i = \{C_{i1}, \cdots, C_{ik}\}$ denote a group with k classes.

In the process of grouping variables, each group may have many classes. We categorize these as asymmetric, symmetric and independent classes.

Definition 11: (Asymmetric Class): Every asymmetric variable in a group is an asymmetric class.

Definition 12: (Symmetric Class): A variable set $C_i = \{x_{i1}, \cdots, x_{ik}\}$ and $k \in \{2, \cdots, n\}$ is a symmetric class if any arbitrary two variables in C_i are symmetric.

Definition 13: (Independent Class): A variable set $C_i = \{x_{i1}, \cdots, x_{ik}\}$ and $k \in \{1, \cdots, n-1\}$ is an independent class if every variable in C_i is an independent variable.

B. COMPUTE CANONICAL FORM

The maximal canonical transformation C_f is found after comparing all the candidate canonical transformations. We first try to use the 1st DC signature value to group the variables. When the 1st DC signature values can not resolve all the groups, we use the 2nd-order DC signature values and higher DC signature values until all the groups are resolved. Then, our algorithm generates a candidate canonical transformation.

In the matching process of two Boolean functions f and g, the algorithm first judges whether there is an output negation between them. If $|f| = |g| \wedge |f| \neq |\overline{g}|$, the phases of f and g are positive. There is no output negation and our algorithm computes the canonical form of f and g. If $|f| = |\overline{g}| \wedge |f| \neq |g|$, the phase of f is positive and the phase of g is negative. There is an output negation and our algorithm computes the canonical form of f and $\overline{g}$. The matching runtime will be double when $|f| = |\overline{g}| \wedge |f| = |g|$, because our algorithm must try two phases for f and g.

The phase of a variable is obtained by comparing the size of its positive and negative cofactors. The phase of x_i is positive when $|f_{x_i}| > |f_{\overline{x}_i}|$, and negative when $|f_{x_i}| < |f_{\overline{x}_i}|$. When $|f_{x_i}| = |f_{\overline{x}_i}|$, we need to use a higher signature to determine its phase. If the higher signature cannot determine its phase, we must try both positive and negative.

If variable x_i of a Boolean function f is an independent variable, the phase of x_i is assigned as positive when $|f_{x_i}| = |f_{\overline{x}_i}|$ because x_i is independent to the Boolean function.

If f has k independent variables and their phases cannot be determined by comparing the positive and negative cofactors, we may try two phases; which doubles the search space. Therefore, checking independent variables is a necessary operation.

Procedure 1 Compute Canonical Form

Input: f
Output: $\mathbb{F}$
   function Canonical(f)
      CT_List=NULL
      Create Boolean function f and compute $|f|$
      Compute the 1st DC signature of f
      Determine the phase of $x_1, \cdots, x_n$
      Check the independent variables
      Check the symmetric variable
      m=group(f)
      SEARCH(f, CT_list, $group$, m)
      if $|f| = 2^{n-1}$ **then**
         $f = \overline{f}$
         Compute the 1st difference signature of f
         Determine the phases of $x_1, \cdots, x_n$
         Check the independent variables
         Check the symmetry variable
         m=group(f)
         SEARCH(f, CT_list, $group$, m)
         $\mathcal{T} = C_f{}^{-1}$
      else
         $\mathcal{T} = C_f{}^{-1}$
      end if
   end function

The purpose of Procedure 1 is to compute the canonical form of a Boolean function f. The individual tasks in Procedure 1 are as follows.

1) Compute $|f|$.

2) Compute the 1st DC signature values of all variables of f.

3) The 1st DC signature value determines the phases of all variables. Variables whose phases are not determined are handled in Procedure 2. The method used is the same as that described in [1].

4) Check the symmetry of all variables.

5) Check the independent variables. If the phases of independent variables are not determined, assign the positive phase to them.

6) Group all variables by comparing their 1st DC signature values.

7) Call Procedure 2 to get the maximal canonical transformation C_f.

8) Compute canonical form by the maximal canonical transformation C_f.

Example 1: Consider a 7-input Boolean function $f(X) = x_3\overline{x}_5x_7 + \overline{x}_1\overline{x}_2x_4 + \overline{x}_1\overline{x}_2\overline{x}_4x_5\overline{x}_6 + x_3\overline{x}_4x_5\overline{x}_6 + x_2\overline{x}_3\overline{x}_4x_5\overline{x}_6 + \overline{x}_2\overline{x}_3\overline{x}_4x_5\overline{x}_6$, whose phase assignment and group variables are as follows.

Procedure 2 Canonical transformation search process

Input: f, map_list, $group$, m
Output: C_f
  function Search(f, CT_list, $group$, m)
    if D_1 **then**
      if $Empty(C_f)$ **then**
        $C_f = T$
      else if $D^{C_f} \prec D^T$ **then**
        $C_f = T$
      end if
    else
      for all G_i, $i \in (1, \cdots, m)$ **do**
        if D_2 **then**
          break
        end if
      end for
      if D_3 **then**
        Add the variables in G_i to CT_list
        SEARCH(f, CT_list, $group$, m)
      else
        for all C_{ij}, $j \in \{1, \cdots, k\}$ **do**
          Split G_i
          create node_list
        end for
        for all $node\_j$ in node_list **do**
          UPDATE_SEQUENCE($node\_j$, $group$)
          m=m+1
          if D_1 **then**
            if $Empty(C_f)$ **then**
              $C_f = T$
            else if $D^{C_f} \prec D^T$ **then**
              $C_f = T$
            end if
          else
            Add the variables in G_i to CT_list
            Update_signature(f)
            m=group(f)
            SEARCH(f, CT_list, $group$, m)
          end if
        end for
      end if
    end if
  end function

Procedure 1 computes $|f| = 46$ and assigns a negative phase to f. Then, it computes the 1st DC signature value of all the variables of f, and the results are $\{(16,28), (16,28), (30,28), (22,44), (24,44), (15,32), (30,28)\}$. The variables x_3, x_5 and x_7 are positive, and the others are negative.

Check the symmetry of every variable. There are two symmetric class $\{x_1, x_2\}$ and $\{x_3, x_7\}$. Group the variables according to the 1st DC signature value. The grouping result is $G = \{G_1, G_2, G_3\}$, and $G_1 = \{C_{11}\} = \{\overline{x}_6\}$, $G_2 = \{C_{21}, C_{22}\} = \{\{\overline{x}_1, \overline{x}_2\}, \{x_3, x_7\}\}$ and $G_3 = \{C_{31}, C_{32}\} = \{\{\overline{x}_4\}, \{x_5\}\}$.

C. SEARCHING THE CANONICAL TRANSFORMATIONS

After obtaining the initial group results, Procedure 2 begins to search candidate canonical transformations. Procedure 2 first addresses the first group G_1, then group G_2, and so on until all groups have been resolved. A group G_i is resolved when

G_i has only one class. There are three possible cases when group G_i is resolved.

1) Group G_i has only one asymmetric variable and the phase of this variable is determined.

2) Group G_i has multiple variables, an arbitrary two variables of G_i are symmetric, and the phases of all variables of G_i are determined.

3) Group G_i has one or more variables, all of which are independent variables.

Because the Boolean difference signature of an independent variable is 0 and the Boolean difference signature of dependent variable must not be 0, a group contains only independent variables when any independent variable occurs.

When searching candidate canonical formations, all the candidate canonical transformations are stored in a tree and Procedure 2 uses a depth-first search to find candidate canonical transformations. Each branch of the candidate canonical transformation tree is a candidate canonical transformation. In Procedure 2, CT_list is the candidate canonical transformation tree, which is initialized to NULL. T is the candidate canonical transformation found in the search process, and C_f is the maximal canonical transformation, which is initialized empty.

Let $G = \{G_1, G_2, \cdots, G_m\}$ be the grouping results that Procedure 1 obtained using the 1st DC signature values. The first step in Procedure 2 is to check whether all groups are resolved. Here we use the condition D_1 to denote whether all groups are resolved.

When D_1 is true, Procedure 2 finds a candidate canonical transformation T. When this occurs, Procedure 2 assigns T to C_f if C_f is empty or $D^{C_f} \prec D^T$. Then, Procedure 2 continues to search for other candidate canonical transformations until all the candidate canonical transformations have been searched.

When D_1 is false, some groups could not be resolved. At this point, Procedure 2 searches the group G_i which is an unresolved group that has the minimal sequence number. Condition D_2 becomes true when an unresolved group is found.

When group G_i (the unresolved group having the minimal sequence number) is found, it is handled in two ways.

1) Group G_i can be resolved.

The condition D_3 denotes whether group G_i can be resolved. When D_3 is true, Procedure 2 adds all the variables in G_i to the candidate canonical transformation tree and calls itself recursively.

In Example 1, the condition D_1 is false. Procedure 2 finds that group G_1 is the first unresolved group and that it can be resolved in this call. Procedure 2 adds variable $\overline{x}_6$ to the candidate canonical transformation tree, giving a new layer with a node. Then, Procedure 2 calls itself recursively.

2) Group G_i cannot be resolved.

When condition D_3 is false, Procedure 2 splits group G_i. Group G_i has k classes; thus, there are k approaches for splitting it. Let group $G_i = \{C_{i1}, \cdots, C_{ik}\}$, select one class as group G_i; the other classes are in group G_{i+1}. The first

splitting approach is $P_1 = \{G_i, G_{i+1}\} = \{\{C_{i1}\}, \{C_{i2}, \cdots, C_{ik}\}\}$, the second splitting approach is $P_2 = \{G_i, G_{i+1}\} = \{\{C_{i2}\}, \{C_{i1}, C_{i3}, \cdots, C_{ik}\}\}$, and the kth splitting approach is $P_k = \{G_i, G_{i+1}\} = \{\{C_{ik}\}, \{C_{i1}, \cdots, C_{i(k-1)}\}\}$. Note when the phases of the variables of group G_i are not determined, there are $2k$ splitting approaches, because we need to try the positive and the negative.

Suppose there are k splitting approaches. These k splitting approaches are stored in the variable node_list. Then, Procedure 2 selects one splitting approach from the variable node_list in order. Let Procedure 2 selects the qth splitting approach $P_q = \{G_i, G_{i+1}\} = \{\{C_{iq}\}, \{C_{i1}, \cdots, C_{i(q-1)}, C_{i(q+1)}, \cdots, C_{ik}\}\}$, Procedure 2 calls UPDATE_SEQUENCE() to update the sequence number of the groups from G_i to G_m. the group sequence number of the variables of C_{iq} remains unchanged and the group sequence number of the variables of $\{C_{i1}, \cdots, C_{i(q-1)}, C_{i(q+1)}, \cdots, C_{ik}\}$ is incremented by one. Finally, the group sequence number of the variables of original group $G_{i+1}, \cdots, G_m$ is incremented by one, and the value of m is incremented by one.

Then, Procedure 2 checks the condition D_1. When condition D_1 is true, Procedure 2 creates a candidate canonical transformation T and compares the DC signature vector of T and C_f. When condition D_1 is false, the qth branch is added to the $|C_{iq}|$ layers as one layer with one node. In other words, $|C_{iq}|$ expresses the number of variables in C_{iq}. Procedure 2 calls Update_signature() to update the DC signature value of f and regroups the variables of $G_{i+1}, \cdots, G_m$, and calls itself recursively.

Procedure 2 starts with the first splitting approach and then processes the next splitting approach until all splitting approaches have been processed.

In Example 1, after handling group G_1, group G_2 is the next unresolved group with the minimal sequence number. Group G_2 has two symmetric classes; therefore, it can not be resolved. There are two ways to split group G_2: $P_1 = \{G_2, G_3\} = \{\{\overline{x}_1, \overline{x}_2\}, \{x_3, x_7\}\}$ and $P_2 = \{G_2, G_3\} = \{\{x_3, x_7\}, \{\overline{x}_1, \overline{x}_2\}\}$. The first branch added to the candidate canonical transformation tree has two layers with the nodes $\overline{x}_1$ and $\overline{x}_2$. The second branch added to the candidate canonical transformation tree has two layers with the nodes x_3 and x_7.

Procedure 2 selects one splitting approach in sequence and updates the DC signature values. The method for updating the DC signature is similar to that used in [1] to update signatures. In Example 1, Procedure 2 uses variable $\overline{x}_6$ to compute the 2nd-order DC signature value for the other variables. Thus, there are new DC signature values, and Procedure 2 regroups all the unresolved groups and calls itself recursively.

Our algorithm uses three strategies to reduce the search space.

1) A symmetric class has multiple variables. When we group and sort variables, all the variables of a symmetric class are regarded as a single operational object.

2) Take advantage of the independence of independent variable. When the phase of independent variables cannot

be determined, we assign positive phase to them. If we do not make use of independent variable, a independent class C_{il} is a symmetric class. When the phases of the variables in C_{il} are not determined, the number of candidate canonical transformations is doubled. When there are other symmetric classes having the same signatures to C_{il}, the number of candidate canonical transformations is grow even more.

3) The combination of the cofactor signature and Boolean difference signature can better distinguish variables.

In Example 1, after updating the DC signature using variable $\overline{x}_1$, variables $\overline{x}_4$ and x_5 are also resolved. There are two candidate canonical transformations $T1 = \{\overline{x}_6, \overline{x}_1, \overline{x}_2, x_3, x_7, \overline{x}_4, x_5\}$ and $T2 = \{\overline{x}_6, x_3, x_7, \overline{x}_1, \overline{x}_2, \overline{x}_4, x_5\}$. The candidate canonical transformation tree of Example 1 is shown in Fig. 1.
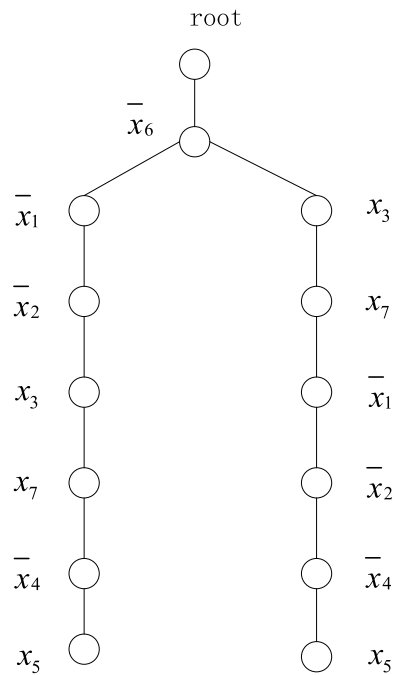

FIGURE 1. The candidate canonical transformation tree of Example 1.

For a 7-input Boolean function, there are $7!2^7$ NP transformations, while only two candidate canonical transformations exist after using our algorithm in Example 1. Through the DC signature vector comparison, the maximal canonical transformation C_f is $\{\overline{x}_6, \overline{x}_1, \overline{x}_2, x_3, x_7, \overline{x}_4, x_5\}$.

To demonstrate that our algorithm is superior to the algorithm proposed in [1], we use both algorithms to search the maximal candidate canonical transformations in Example 2.

Example 2: Consider a 6-input Boolean function $f(X) = \overline{x}_1 x_2 x_3 \overline{x}_4 + x_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 + \overline{x}_1 \overline{x}_2 \overline{x}_3 x_4 + \overline{x}_1 x_2 x_3 x_6 + \overline{x}_1 x_2 x_4 x_5 \overline{x}_6 + x_1 \overline{x}_2 x_3 \overline{x}_6 + x_1 x_2 \overline{x}_3 x_6 + \overline{x}_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 x_3 \overline{x}_4 x_6 + \overline{x}_1 \overline{x}_2 x_3 x_5 x_6 + \overline{x}_1 \overline{x}_2 x_3 x_5 \overline{x}_6 + \overline{x}_1 x_2 \overline{x}_3 x_4 \overline{x}_5 \overline{x}_6 + x_1 x_2 x_3 x_4 \overline{x}_5 \overline{x}_6 + x_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 x_5 x_6$. We use both the signature vector from [1] and the DC signature vector proposed in this paper to search the candidate canonical transformations.

1) Using the algorithm of [1], the candidate canonical transformation search process is as follows.

The algorithm from [1] computes the 1st signature, checks the symmetry and groups variables. The Boolean function f has 32 minterms, i.e., $|f| = 32$. The results of the 1st signature are $|f_{x_1}| = 13$ and $|f_{x_2}| = |f_{x_3}| = |f_{x_4}| = |f_{x_5}| = |f_{x_6}| = 16$. There are no symmetric classes. The phase of variable x_1 is negative and the phases of the other variables are not determined. According to the 1st signatures, the grouping result is $G = \{G_1, G_2\} = \{\{x_1\}, \{x_2, x_3, x_4, x_5, x_6\}\}$.

The first group G_1 can be resolved, and the variable $\bar{x}_1$ is used to compute the 2nd-order signature of the variables in G_2. The results are $|f_{\bar{x}_1 x_2}| = 9$ and $|f_{\bar{x}_1 x_3}| = |f_{\bar{x}_1 x_4}| |f_{\bar{x}_1 x_5}| |f_{\bar{x}_1 x_6}| = 10$. We know that $|f_{\bar{x}_1}| = |f| - |f_{x_1}| = 19$; therefore, we can determine the phases of these 5 variables. Variable x_2 is negative and the others are positive. But G_2 cannot be resolved: these 5 variables have the same 2nd-order signature. Group G_2 has 5 asymmetric classes, the algorithm of [1] splits G_2 and there are five ways to split it.

- $P_1 = \{G_2, G_3\} = \{\{x_2\}, \{x_3, x_4, x_5, x_6\}\}$
- $P_2 = \{G_2, G_3\} = \{\{x_3\}, \{x_2, x_4, x_5, x_6\}\}$
- $P_3 = \{G_2, G_3\} = \{\{x_4\}, \{x_2, x_3, x_5, x_6\}\}$
- $P_4 = \{G_2, G_3\} = \{\{x_5\}, \{x_2, x_3, x_4, x_6\}\}$
- $P_5 = \{G_2, G_3\} = \{\{x_6\}, \{x_2, x_3, x_4, x_5\}\}$

The algorithm of [1] traverses each possible split shown above. We select the first split approach as an example. The updated groups are $G_1 = \{x_1\}$, $G_2 = \{x_2\}$ and $G_3 = \{x_3, x_4, x_5, x_6\}$.

Group G_2 can be resolved; therefore, we use variable $\bar{x}_2$ to compute 2nd-order signature and group variables. The 2nd-order signatures of the unresolved variables are $|f_{\bar{x}_2 x_3}| = |f_{\bar{x}_2 x_4}| = |f_{\bar{x}_2 x_5}| = |f_{\bar{x}_2 x_6}| = 8$. From these results, we also know that group G_3 also cannot be resolved. There are four ways to split it as follows.

- $P_1 = \{G_3, G_4\} = \{\{x_3\}, \{x_4, x_5, x_6\}\}$
- $P_2 = \{G_3, G_4\} = \{\{x_4\}, \{x_3, x_5, x_6\}\}$
- $P_3 = \{G_3, G_4\} = \{\{x_5\}, \{x_3, x_4, x_6\}\}$
- $P_4 = \{G_3, G_4\} = \{\{x_6\}, \{x_3, x_4, x_5\}\}$

Continuing, the algorithm of [1] computes its signature and splits the group. Because variables x_2, x_3, x_4, x_5 and x_6 always have the same 2nd-order signature, the following computations will generate a total of $5 \times 4 \times 3 \times 2$ candidate canonical transformations.

2) Using the algorithm proposed by this paper, the candidate canonical transformation search process is as follows.

Our algorithm computes the 1st DC signature value of each variable, checks the symmetry, and searches the independent variables and groups variables by the 1st DC signature value.

Boolean function f has no symmetry classes or independent classes. The 1st DC signature values are $(|f_{x_1}|, |f_{x_1}'|) = (13, 64)$, $(|f_{x_2}|, |f_{x_2}'|) = (16, 36)$, $(|f_{x_3}|, |f_{x_3}'|) = (16, 52)$, $(|f_{x_4}|, |f_{x_4}'|) = (16, 20)$, $(|f_{x_5}|, |f_{x_5}'|) = (16, 12)$ and $(|f_{x_6}|, |f_{x_6}'|) = (16, 28)$.

Because $|f| = 32$, we can determine that the phase of variable x_1 is negative and the phases of others are not

determined. Our algorithm groups the variables and generates 6 groups: $G_1 = \{x_1\}$, $G_2 = \{x_3\}$, $G_3 = \{x_2\}$, $G_4 = \{x_6\}$, $G_5 = \{x_4\}$ and $G_6 = \{x_5\}$.

Group G_1 can be resolved. The other groups are not resolved because the phases of the variables in these groups have not been determined. Our algorithm uses the variable $\bar{x}_1$ to compute the 2nd-order DC signature values, which are: $(|f_{\bar{x}_1 x_2}|, |f_{\bar{x}_1 x_2}'|) = (9, 18)$, $(|f_{\bar{x}_1 x_3}|, |f_{\bar{x}_1 x_3}'|) = (10, 26)$, $(|f_{\bar{x}_1 x_4}|, |f_{\bar{x}_1 x_4}'|) = (10, 10)$, $(|f_{\bar{x}_1 x_5}|, |f_{\bar{x}_1 x_5}'|) = (10, 6)$, and $(|f_{\bar{x}_1 x_6}|, |f_{\bar{x}_1 x_6}'|) = (10, 14)$.

From the above 2nd-order DC signature values, the phases of all variables are determined. Therefore, groups G_2, G_3, G_4, G_5 and G_6 are all resolved. One candidate canonical transformation is generated: $\{\bar{x}_1, x_3, \bar{x}_2, x_6, x_4, x_5\}$.

Because $|f| = |\bar{f}|$, the algorithm of [1] and ours must both perform the same search for $\bar{f}$. However, the Boolean function f results in 240 candidate canonical transformations when computed by the algorithm of [1] but only 2 when computed by the algorithm proposed in this paper.

Using our algorithm, the NPN Boolean matching algorithm is denoted as follows: Given two Boolean function $f(X)$ and g(X), Boolean function f is NPN-equivalent to g if the canonical form $\mathbb{F}$ of f is equal to that of g.

V. EXPERIMENTAL RESULTS

We reimplemented the algorithm of [1] to compare it with the algorithm of this paper and obtain supporting experimental evidence. The two algorithms were tested on a set of randomly generated circuits and a set of MCNC benchmark circuits. The two circuit sets include a number of NPN-equivalent circuits. We tested the NPN matching runtime and the number of candidate canonical transformations generated when searching for the canonical transformation of a Boolean function. The runtime is measured in seconds. The following experimental results were obtained on a computer equipped with 3.3 GHz CPU and 4GB RAM.

**TABLE 1.** Boolean matching results on the random circuits.

#I	#A.T of [1]	# A.C.N of [1]	# A.T of ours	# A.C.N of ours
7	0.0007	13.2	0.0005	7.3
8	0.0009	22.1	0.0004	5.3
9	0.0010	16.8	0.0007	6.1
10	0.0013	19.6	0.0004	3.6
11	0.0032	19.1	0.0020	4.3
12	0.0068	12.5	0.0048	4.5
13	0.0099	17.2	0.0043	3.7
14	0.0097	14.8	0.0062	4.2
15	0.0078	16.5	0.0044	4.2
16	0.0249	13.9	0.0135	5.0
17	0.0326	23.4	0.0208	5.9
18	0.0423	13.3	0.0261	5.0
19	0.1565	15.8	0.1070	5.8
20	0.1195	12.5	0.0974	5.6
21	0.0859	13.3	0.0708	4.9
22	0.1349	18.3	0.1186	6.8

Tables 1 and 2 show the experimental results on the random circuit and MCNC benchmark circuit sets, respectively.

[1] A.T: Average runtime, A.C.N: Average number of candidate canonical transformations

TABLE 2. Boolean matching results on the MCNC benchmark circuits.

#I	#A.T of [1]	# A.C.N of [1]	# A.T of ours	# A.C.N of ours
7	0.00215	66.7	0.00037	2.4
8	0.00989	27.1	0.00939	26.3
9	0.00163	3.0	0.00169	2.8
10	0.00098	2.1	0.00103	2.1
11	0.00283	3.2	0.00211	2.3
12	0.00177	2.7	0.00173	2.2
13	0.04242	7.8	0.02971	6.5
14	0.00713	2.1	0.00784	2.0
15	0.02730	2.4	0.02818	2.4
16	0.00238	2.9	0.00246	2.4
17	0.18872	2.8	0.19304	2.5
18	0.21707	4.4	0.21725	2.4
19	1.13960	3.4	1.04902	2.1
20	1.90159	3.8	2.07058	3.7
21	7.1248	2.0	6.6848	1.8
22	13.9547	4.5	14.3204	4.0
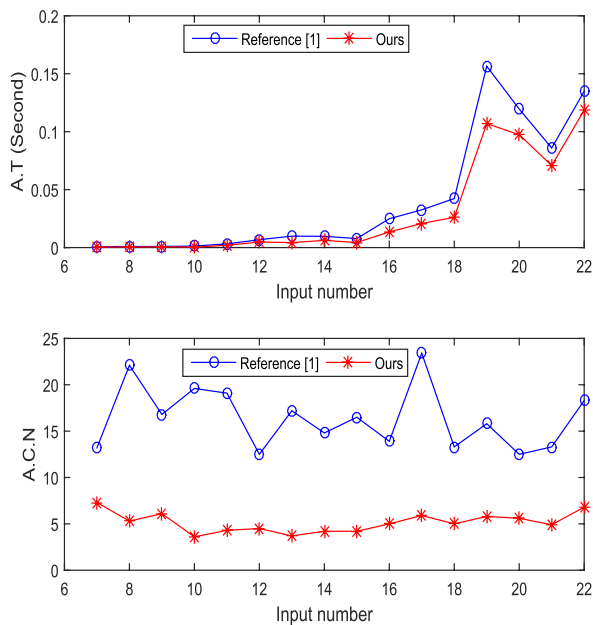
FIGURE 2. The comparison results on the equivalent random circuits.

In these two tables, the first column shows the number of inputs (#I), the second and third columns show the average matching runtime (#A.T) and the average number of candidate canonical transformations (#A.C.T) of the algorithm in [1], respectively, and the forth and fifth columns show the average matching runtime and the average number of candidate canonical transformations of our algorithm, respectively.

The experimental results listed in Table 1 show that our algorithm improves the speed of the Boolean matching process and reduces the search space effectively. In the best case, our algorithm's runtime is 70% faster and its search space is 82% smaller than that of the algorithm from [1]. On average, our algorithm improves the runtime by 37% and reduces the search space by 67%. Fig. 2 shows the results on the equivalent random circuits.

Form Table 2, we can see that the average runtime improves by 83% and the search space is reduced by 96%

compared with [1] when the number of input variables is 7. However, neither the runtime nor the search space are drastically improved in the other input cases. In majority of the circuits of the MCNC benchmark, using only cofactor signature serves to quickly identify the variables. Therefore, the average runtime of the algorithm of [1] when tested on 22 inputs is slightly faster than ours. Despite this, our algorithm improves the average runtime by 6% and reduces the average search space by 19% compared with that of [1] when tested on the MCNC benchmark circuits.

VI. CONCLUSIONS

In this paper, we propose a canonical-based Boolean matching algorithm. The proposed DC signature vector is more effective in computing canonical form than the algorithm of [1]. We test the runtime and the canonical transformation search space of Boolean matching algorithms with 7-22 inputs. The experimental results show that the average runtime of our algorithm is 37% faster and its search space is 67% smaller than those of the algorithm proposed in [1]. The algorithm proposed in this paper is highly effective at reducing the search space and enhancing the Boolean matching speed. In future work, we plan to extend this Boolean matching approach to Boolean matching with don't care set and multiple-output Boolean matching.

REFERENCES

[1] A. Abdollahi and M. Pedram, "Symmetry detection and Boolean matching utilizing a signature-based canonical form of Boolean functions," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 27, no. 6, pp. 1128–1137, Jun. 2008.

[2] Y. Zhang, G. Yang, W. N. Hung, and J. Zhang, "Computing affine equivalence classes of Boolean functions by group isomorphism," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3606–3616, Dec. 2016.

[3] B. Kapoor, "Improved technology mapping using a new approach to Boolean matching," in *Proc. Eur. Design Test Conf.*, Paris, France, Mar. 1995, pp. 86–90.

[4] F. Mailhot and G. De Micheli, "Technology mapping using Boolean matching and don't care sets," in *Proc. Eur. Design Autom. Conf. (EDAC)*, Glasgow, U.K., Mar. 1990, pp. 212–216.

[5] M. Damiani and A. Y. Selchenko, "Boolean technology mapping based on logic decomposition," in *Proc. 16th Symp. Integr. Circuits Syst. Design (SBCCI)*, Sao Paulo, Brazil, Sep. 2003, pp. 35–40.

[6] D. Debnath and T. Sasao, "Efficient computation of canonical form for Boolean matching in large libraries," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Yohohama, Japan, Jan. 2004, pp. 591–596.

[7] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto, "A transform-parametric approach to Boolean matching," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 28, no. 6, pp. 805–817, Jun. 2009.

[8] D. Chai and A. Kuehlmann, "Building a better Boolean matcher and symmetry detector," in *Proc. Design, Autom. Test Eur. (DATE)*, Munich, Germany, Mar. 2006, pp. 1–6.

[9] J. Ciric and C. Sechen, "Efficient canonical form for Boolean matching of complex functions in large libraries," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 22, no. 5, pp. 535–544, May 2003.

[10] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto, "A unified approach to canonical form-based Boolean matching," in *Proc. 44th ACM/IEEE Design Autom. Conf. (DAC)*, San Diego, CA, USA, Jun. 2007, pp. 841–846.

[11] Z. Huang, L. Wang, and Y. Nasikovskiy, "Fast Boolean matching based on NPN classification," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Kyoto, Japan, Dec. 2013, pp. 310–313.

[12] A. Petkovska, M. Soeken, G. De Micheli, P. Ienne, and A. Mishchenko, "Fast hierarchical NPN classification," in *Proc. Int. Conf. Field Program. Logic Appl.*, Lausanne, Switzerland, Aug./Sep. 2016, pp. 1–4.

[13] K.-C. Chen and J. C.-Y. Yang, "Boolean matching algorithms," in *Proc. Int. Symp. VLSI Technol., Syst., Appl.*, Taipei, Taiwan, May 1993, pp. 44–48.

[14] A. Abdollahi, "Signature based Boolean matching in the presence of don't cares," in *Proc. 45th ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, Jun. 2008, pp. 642–647.

[15] Y.-T. Lai, S. Sastry, and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Process. (ICCD)*, Cambridge, MA, USA, Oct. 1992, pp. 452–458.

[16] J. Zhang, G. Yang, W. N. Hung, and Y. Zhang. (2017). "An efficient NPN Boolean matching algorithm based on structural signature and shannon expansion." [Online]. Available: https://arxiv.org/abs/1708.04597

[17] K. H. Wang, C. M. Chan, and J. C. Liu, "Simulation and SAT-based Boolean matching for large Boolean networks," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jul. 2009, pp. 396–401.

[18] H. Katebi and L. Igor, "Large-scale Boolean matching," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. New York, NY, USA: springer, 2010, pp. 771–776.

[19] Y. Matsunaga, "Accelerating sat-based Boolean matching for heterogeneous fpgas using one-hot encoding and cegar technique," in *Proc. Design Autom. Conf.*, Chiba, Japan, Jan. 2016, pp. 255–260.

[20] X.-Q. Wang and Y. Yang, "New approach of exploiting symmetry in SAT-based Boolean matching for FPGA technology mapping," in *Proc. IEEE Int. Conf. Veh. Electron. Saf.*, Dongguan, China, Jul. 2013, pp. 282–285.

[21] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient SAT-based Boolean matching for FPGA technology mapping," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jul. 2006, pp. 466–471.

[22] K.-H. Wang and C.-M. Chan, "Incremental learning approach and SAT model for Boolean matching with don't cares," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, Nov. 2007, pp. 234–239.

[23] J. Moore, K. Fazel, M. A. Thornton, and D. M. Miller, "Boolean function matching using walsh spectral decision diagrams," in *Proc. IEEE Dallas/CAS Workshop Design, Appl., Integr. Softw.*, Richardson, TX, USA, Oct. 2008, pp. 127–130.

[24] C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang, "Boolean matching of function vectors with strengthened learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2010, pp. 596–601.

[25] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, May 1998, pp. 206–211.

[26] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch, "Linear cofactor relationships in Boolean functions," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 25, no. 6, pp. 1011–1023, Jun. 2006.

GUOWU YANG received the B.S. degree from the University of Science and Technology of China in 1989, the M.S. degree from the Wuhan University of Technology in 1994, and the Ph.D. degree in electrical and computer engineering from Portland State University in 2005. He was with the Wuhan University of Technology from 1989 to 2001 and with Portland State University from 2005 to 2006. He is currently a Full Professor with the University of Electronic Science and Technology of China. His research interests include verification, logic synthesis, quantum computing, and machine learning. He has published over 100 journal and conference papers.
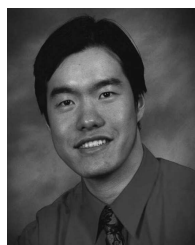
WILLIAM N. N. HUNG received the B.S. and M.S. degrees from the University of Texas at Austin in 1994 and 1997, respectively, and the Ph.D. from Portland State University in 2002 all in electrical and computer engineering, respectively.

He is currently leading technological innovation efforts on constraint-based verification and hardware-accelerated verification, such as emulation and prototyping as Synopsys Scientist (Senior Director) with Synopsys, Mountain View, CA, USA. He was with several high-tech companies, including Intel, Syncplicity and Synopsys. He has over 19 years of industrial Research and Development experience, has published over 80 journal and conference papers, and has patented numerous inventions.

Dr. Hung has served on the technical program committees of conferences, such as DAC, DATE, ICCD, CAV, FMCAD, CEC, WCCI, and others. He was the Chair of the Quantum Computing Task Force under the Emergent Technologies Technical Committee of the IEEE Computational Intelligence Society. He also served as the Co-Chair of the Logic and Circuit Track for the technical program committee of ICCD. He is currently an Associate Editor of the IEEE Transactions on CAD and the IEEE Transactions on Circuits and Systems II.

JULING ZHANG is currently pursuing the Ph.D. with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, China. Her research interests include logic synthesis, Boolean matching and information security risk assessments.

JINZHAO WU was born in 1965. He received the Ph.D. degree in science from the Institute of Systems Science, Chinese Academy of Sciences. He is currently a Professor with the Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Guangxi University for Nationalities. His research interests include the fields of formal methods, symbolic computation, and automated reasoning.

• • •