A New Canonical Form for Fast Boolean Matching in Logic Synthesis and Verification

Afshin Abdollahi
University of Southern California
afshin@usc.edu

Massoud Pedram
University of Southern California
pedram@ceng.usc.edu

Abstract – *An efficient and compact canonical form is proposed for the Boolean matching problem under permutation and complementation of variables. In addition an efficient algorithm for computing the proposed canonical form is provided. The efficiency of the algorithm allows it to be applicable to large complex Boolean functions with no limitation on the number of input variables as apposed to previous approaches, which are not capable of handling functions with more than seven inputs. Generalized signatures are used to define and compute the canonical form while symmetry of variables is used to minimize the computational complexity of the algorithm. Experimental results demonstrate the efficiency and applicability of the proposed canonical form.*

I. Introduction

In this paper Boolean matching is defined as the problem of determining whether a Boolean function can be functionally equivalent to another one under a permutation of its inputs and complementation of some of its inputs. Boolean matching algorithms have many applications in logic synthesis including cell-library binding where it is necessary to repeatedly determine whether some part (cluster) of a Boolean network can be realized by any of the cells in a library [1]. Boolean matching is a critical and CPU-intensive task and therefore, there have been many efforts to effectively solve the problem [2]. Boolean functions that are equivalent under negation of inputs are N-equivalent, under permutation of inputs are P-equivalent, and under both stated conditions, are NP-equivalent [3]. An exhaustive method for Boolean matching is computationally expensive since the complexity of such an algorithm for n-variable functions is $O(n!2^n)$.

Boolean matching algorithms can be classified into two categories: pair-wise matching algorithms and algorithms based on canonical forms of functions. Pair-wise Boolean matching algorithms are based on a semi-exhaustive search where the search space is pruned by the use of some signatures which are computed from some properties of Boolean functions [2]. A signature in general is a description of (one or more) input variables of a Boolean function that is independent of the permutation or complementation of the variables of the function. To match a function against a cell library, pair-wise matching algorithms often need to perform pair-wise matching of the function with all the library cells. Therefore, these algorithms can only handle libraries with a modest size.

Boolean matching algorithms that belong to the second category compute some canonical form for Boolean functions [5] - [10]. These algorithms are based on the fact that two functions match if and only if their canonical forms are the same. Burch and Long introduced a canonical form for matching under complementation and a semi-canonical form for matching under permutation of the variables [5]. Thus, in order to handle complementation and permutation of inputs simultaneously, a large number of forms for each cell are required. Other researchers, including Wu et al. [6], Debnath and Sasao [8], and Ciric and Sechen [9] also proposed canonical forms that are applicable for Boolean matching under permutation of the variables only and do not handle complementation of inputs. Hinsberger and Kolla [7] and Debnath and Sasao [10], introduced a canonical form for solving the general Boolean matching problem. However their approach is mainly based on manipulating the truth table of the function and using a table look-up which introduces an enormous space complexity, thus limiting the algorithm to library cells with seven and fewer input variables, which is a major limitation.

In this paper a new canonical form for representing Boolean functions is introduced. The proposed canonical form for an arbitrary Boolean function is the unique Boolean function that is obtained after applying some CP transformation on the input variables. The canonical forms of NP-equivalent Boolean functions are identical. Next, an effective technique is presented for generating this canonical form. The proposed method is based on using *generalized signatures* (signatures of one or more variables) to find a *canonicity-producing* (CP) phase assignment and ordering for variables. From here on, phase assignment and ordering for variables is referred to as a *transformation* on variables. For most Boolean functions, single-variable and two-variable signatures are enough to recognize all variables (i.e., to obtain a CP transformation.) However, use of single-variable and two-variable signatures alone may not result on a canonical input transformation. In this paper it is shown that, by using generalized signatures of one or more variables, it is always possible to create a CP transformation on variables of the function.

Experimental results provided in this paper demonstrate that the proposed approach for computing the canonical form does not have the limitations of previous works; i.e. it computes the canonical form under both permutation and complementation of variables and there is no limit on number of variables. An important advantage of the proposed technique is the way it handles and uses the symmetry of variables to minimize the complexity of the algorithm compared to some of the previous approaches which are not able to consider symmetries [7][10]. Hence, the proposed technique is applicable to large libraries with cells of any number of inputs. Also it can be used for logic verification of large circuits since there is no limit on the number of input variables.

The reminder of this paper is organized as follows: In section II, definitions and terminology are introduced. In section III, CP transformations and canonical forms are defined. Section IV describes how symmetries of variables are handled and section V provides the details of computing the canonical form followed by experimental results and conclusions in sections VI and VII.

II. Preliminaries

Let X denote a vector (x_1, x_2,…, x_n) of Boolean variables and $f(X)$ a single-output completely-specified Boolean function of X. A *literal* is a variable, x_i, or its complement $\overline{x}_i$. In general a literal can be denoted as x_i^p, where p denotes the phase assignment to variable x_i i.e., $x_i^1=x_i$ and $x_i^0=\overline{x}_i$. A *minterm* is a vertex in the n-dimensional Boolean space, $\{0, 1\}^n$ and a *cube* is the conjunction of some literals. An *NP transformation*, Γ, is an onto mapping which assigns a unique phase and variable ordering to each variable in X i.e., $\Gamma(X)=T$ where $T=(t_1,t_2,\ldots, t_n)$ and $t_i= x_{\pi(i)}^{p(i)}$. Here, $\pi(i)$ denotes the position of x_i under permutation π. Similarly, $p(i)$ denotes the phase of $x_{\pi(i)}$ under phase assignment p. The *inverse* NP *transformation* Γ^{-1} applied to vector T produces vector X. For example, assume that a transformation Γ on (x_1,x_2,x_3) produces $(x_2,\overline{x}_3,x_1)$. The inverse transformation Γ^{-1} on $(x_2,\overline{x}_3,x_1)$ produces (x_1,x_2,x_3). Notice also that $\Gamma^{-1}(x_1,x_2,x_3)=(x_3,x_1,\overline{x}_2)$.

Fact. Boolean functions $f_1(X)$ and $f_2(X)$ are *NP-equivalent* exactly if there exists a transformation $\Gamma(X)=T$ such that $f_1(X)= f_2(T)$. Notice that if $f_1(X)= f_2(T)$ then $f_1(T^{-1})= f_2(X)$ where $T^{-1}=\Gamma^{-1}(X)$.

Note that NP-equivalence is an equivalence relation, partitioning the space of n-input Boolean functions to equivalence classes.

Example 1. Let $f_1(x_1,x_2,x_3)=x_1x_2+\overline{x}_1x_3$ and $f_2(x_1,x_2,x_3)=x_1x_3+\overline{x}_2\,\overline{x}_3$. Then $f_1(X)= f_2(T)$ where $X=(x_1,x_2,x_3)$ and $T=(x_2,\overline{x}_3,x_1)$. Thus, $f_1(X)$ and $f_2(X)$ are NP-equivalent. Also $f_1(T^{-1})=f_1(x_3,x_1,\overline{x}_2)=f_2(X)$.

In the remainder of this paper, for a given X, when there is no confusion, we will use T to denote the transformation, Γ, or its result, $\Gamma(X)$. A similar overloading will be used for T^{-1}.

For any function $f(X)$, let $|f(X)|$ denote the number of minterms covered by $f(X)$. The cofactor of f with respect to a literal x_i ($\overline{x}_i$) is the Boolean function obtained by setting x_i ($\overline{x}_i$) to 1 in f and is denoted by f_{x_i} ($f_{\overline{x}_i}$), which is considered as a function with the same number of input variables as f. The cofactor of f with respect to a cube, b, is the Boolean function obtained by setting all literals of b to 1 in f and is denoted by f_b, which is also considered as a function with the same number of input variables as f.

Definition: For any cube b, *generalized signature*, $|f_b|$, of function f with respect to cube b is equal to the number of minterms in the onset of function f_b. If b consists of exactly k literals, then the corresponding signature will be referred to as a k^{th} order signature.

III. Canonical Form

There are $2^n n!$ NP transformations on n variables $X=(x_1, x_2,\ldots, x_n)$ of function $f(X)$. Some of these transformations result in identical Boolean functions. Therefore, the set of Boolean functions of exactly n variables can be partitioned into a set of NP-equivalent functions. Each equivalence class is uniquely represented by a single *NP-representative function*. It is important to be able to determine the equivalence class of a given Boolean function, *f(X)*. This is in turn achieved by applying a transformation to X such that *f(T)* is the NP-representative function of that class.

Definition: The canonical form of function *f(X)* is a function, f^C, such that $f^C(X)=f(T_C)$, where T_C denotes a *canonicity-producing* (or CP) transformation. Notice that $f^C(T_C^{-1})=f(X)$.

There may be more than one such CP transformation due to variable symmetries as described in the next section. For example, for function *f(X)*, T_{C_1} and T_{C_2} may be two different CP transformations, which result in the same canonical form $f^C(X)$, i.e., $f(T_{C_1})=f(T_{C_2})=f^C(X)$. To guarantee that the canonical forms of *f(X)* and its *negation-equivalent functions* are the same, constraint $\left|f_{\bar{t}_i}\right|\leq\left|f_{t_i}\right|$ is imposed on the inverse of CP transformation, T. In addition, for computational efficiency reasons, "NE-symmetric" variables (cf. section IV) must be placed next to one another. Therefore, an order relation, '$\prec$', is defined among the transformations that satisfy these constraints.

Definition: For a transformation $T=(t_1, t_2,\ldots, t_n)$, the *signature vector* S^T is defined as:

$$S^T=(\overbrace{|f_{t_1}|,\ldots,|f_{t_n}|}^{1^{\text{st}}-\text{signatures}},\overbrace{|f_{t_1t_2}|,|f_{t_1t_3}|,\ldots,|f_{t_{n-1}t_n}|}^{2^{\text{nd}}-\text{signatures}},\ldots,\overbrace{|f_{t_1\ldots t_{n-1}}|,\ldots,|f_{t_2\ldots t_n}|}^{(n-1)^{\text{th}}-\text{signatures}},\overbrace{|f_{t_1\ldots t_n}|}^{\text{n}^{\text{th}}-\text{signature}})$$

Signature vector S^T includes the 1st-signatures followed by the 2nd- and higher order signatures up to the n^{th}-signatures. Signatures of S^T correspond to cofactors of f with respect to cubes consisting of non-empty subsets of vector T. (Only positive phases of t_i are considered; hence S^T includes 2^n-1 signatures.)

Definition: The relation '$\prec$' between T_1 and T_2 is defined based on the lexicographic comparison of their corresponding signature vectors, S^{T_1} and S^{T_2}, i.e., $T_1\prec T_2\Leftrightarrow S^{T_1}\prec S^{T_2}$.

Recall that in lexicographic comparison of two vectors, the corresponding entries are compared until an inequality is encountered. For example $(2,1,3,4)\prec(2,3,0,1)$. Notice that because $|f|$ is invariant under a CP transformation on its variables, it is not included in the signature vector.

Definition: The *maximal transformation* is a transformation which is maximal with respect to the order relation,'$\prec$' in the set of all transformations that satisfy the aforesaid constraints.

A necessary condition for this method to be valid is that when two signature vectors S^{T_1} and S^{T_2} are identical, the corresponding Boolean functions $f(T_1^{-1})$ and $f(T_2^{-1})$ will also be identical, i.e., $S^{T_1}=S^{T_2}\Leftrightarrow f(T_1^{-1})=f(T_2^{-1})$ as stated equivalently in theorem 1.

Theorem 1. *For a transformation* T^{-1} *on variables X of a function f, values of |f| and signature vector* S^T *uniquely and completely specify function f.*

Proof: Let vector F represent the values of function f in all 2^n minterms (the last column of the truth table of f.) Then the vector $S=[|f|, S^T]$ can be obtained using the matrix relation $S=A\times F$ where A is a reversible matrix of 0 and 1 entries and values and operations in this matrix relation are regarded as integers (rather than Boolean.) The proof follows from the reversibility of matrix A. Details are however omitted because of space limitation. ■

When comparing two different transformations T_1 and T_2, it is often unnecessary to completely compute all the signature vectors. More precisely, if $f(T_1^{-1})\neq f(T_2^{-1})$, then most of the time the 1st-signatures alone will determine the order between T_1 and T_2. However, if all of the 1st signatures are equal, then the 2nd-signatures should be compared. Subsequently, if the 2nd-signatures are also equal, then the 3rd-signatures must be compared, and so on until an inequality is encountered. Based on theorem 1 it is guaranteed that if $f(T_1^{-1})\neq f(T_2^{-1})$, then $S^{T_1}\neq S^{T_2}$ i.e., an inequality in some of signatures will always occur. Experimental evidence shows that in the great majority of cases, a signature inequality occurs for the low order signatures (1st and 2nd signatures.) Intuitively, the reason is that the lower order signatures depend on more minterms of the function and thus contain more information about the function. For example a 1st-signatre depends on 2^{n-1} minterms which is a half of the whole Boolean space (2^n minterms) whereas a 2nd-signatre depends on one-forth of all minterms (2^{n-2} minterms.) Hence, the 1st-signatures are the most powerful and effective signatures. The 2nd-signatures are the next most effective signatures and so on. The reader will observe that this arrangement of the proposed signature vector minimizes the computational complexity.

In this paper a number of techniques are presented to minimize the computational complexity of identifying the CP transformation. One of these techniques is to analyze and use the variable symmetries to significantly prune the search space, and thereby, avoid unnecessary computations. Another technique is to use the 1st and 2nd signatures to order (sort) the literals (variables after phase assignment) of the function. Subsequently, if the resultant ordering does not conclude a CP transformation, then the 3rd and possibly higher order signatures are used to identify a CP transformation.

IV. Symmetry Classes

Accounting for symmetry relations between variables is critical in minimizing the complexity of any Boolean matching algorithm. Hence, this paper focuses on efficient handling and utilization of symmetry relations and symmetry classes.

Definition: Given function $f(X)$, variables x_i and x_j are said to be *non-equivalent-symmetric* (NE-symmetric), denoted as x_iNEx_j, if f is invariant under swapping x_i and x_j [11] i.e., $f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n) = f(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$. Notice that x_iNEx_j if and only if $f_{x_i\bar{x}_j} = f_{\bar{x}_i x_j}$. Two variables x_i and x_j of $f(X)$ are said to be *equivalent-symmetri*c (E-symmetric), denoted as x_iEx_j, if x_iNE$\bar{x}_j$. Notice that x_iEx_j if and only if $f_{\bar{x}_i\bar{x}_j} = f_{x_i x_j}$. Two variables x_i and x_j of $f(X)$ are called *symmetric*, denoted as x_iSx_j, if x_iNEx_j or x_iEx_j.

Fact: Transitive property of the *symmetric relation* (S). If x_iSx_j and x_jSx_k, then x_iSx_k.

Since symmetric relation, S, is an equivalence relation, the set of variables, $x_1,x_2,\ldots,x_n$ can be divided into equivalent classes $C_1,C_2,\ldots,C_m$, that are referred to as *symmetry classes*. Similarly it is possible to define NE-symmetry relation and *NE-symmetry classes* (*NE-classes*) with respect to non-equivalent symmetries only.

Definition: A symmetry (or NE-symmetry) class is *maximal* if no other variable is symmetric (or NE-symmetric) to the variables of the class.

Example 2. Consider the Boolean function:

$$f(x_1, x_2, x_3, x_4, x_5) = (x_1 + \bar{x}_3 + x_4)(\bar{x}_2 + x_5).$$

Variables x_1, x_2, x_3, x_4, x_5 can be divided into two symmetry classes: $C_1 = \{x_1, x_3, x_4\}$ and $C_2 = \{x_2, x_5\}$. Note that a symmetry class (C_1) may include both NE and E symmetries (i.e., x_1Ex_3 and x_1NEx_4.)

Lemma 1. *Given a symmetry class $C=\{x_1,x_2,\ldots,x_i\}$ of function f(X), there always exists a phase assignment $P=(p_1,p_2,\ldots,p_i)$ to variables $(x_1,x_2,\ldots,x_i)$ such that $C^P=\{x_1^{p_1}, x_2^{p_2},\ldots, x_i^{p_i}\}$ is an NE-class.*

Proof: The proof is by construction. The construction algorithm (which we refer to as the **Generate-NE-symmetry**) starts by setting $C^P=\{x_1\}$, i.e., p_1=1. In the next step, if x_1NEx_2, then x_2 is added to C^P, resulting in $C^P=\{x_1, x_2\}$, i.e., p_2=1. However, if x_1Ex_2, then $C^P=\{x_1, \bar{x}_2\}$, i.e., p_2=0. At each step of the algorithm, C^P remains NE-symmetric by assigning appropriate phases to variables being added to C^P. Consider step j where j-1 literals have been included in the NE-class i.e., $C^P=\{x_1^{p_1}, x_2^{p_2},\ldots, x_{j-1}^{p_{j-1}}\}$. Now x_j is either NE-symmetric with all members of $C^P = \{x_1^{p_1}, x_2^{p_2}, \ldots, x_{j-1}^{p_{j-1}}\}$ or E-symmetric with all of them. In the first case x_j is added to C^P (p_j=1) while in the second case $\bar{x}_j$ is added to C^P (p_j=0) resulting in $C^P=\{x_1^{p_1},\ldots, x_{j-1}^{p_{j-1}}, x_j^{p_j}\}$, which is an NE-class. The algorithm continues until all variables of C are added to C^P with appropriate phases. ■

The cofactor of function f with respect to any literal in an NE-class of f, $C=\{x_1,x_2,\ldots,x_i\}$, is a unique function i.e., $f_{x_1} = f_{x_2} = \ldots = f_{x_i}$. Therefore, the cofactor of f with respect to the NE-class C can be defined as the cofactor of f with respect to any literal from C, e.g., $f_C = f_{x_1}$. The second order cofactor of f with respect to NE-classes C_1 and C_2 (i.e., $f_{C_1C_2}$) is defined similarly.

Definition: The *first-order signature* (1st-signature) of f with respect to an NE-class C is defined as $|f_C|$ whereas the *second order signature* (2nd-signature) of f with respect to NE-classes C_1 and C_2 is defined as $\left|f_{C_1C_2}\right|$. Higher order signatures are similarly defined.

V. Computing the Canonical Form

In this section, an algorithm, called **compute-CF**, for computing the canonical form of function $f(X)$ is described. The algorithm consists of several steps including classifying variables to symmetry classes, phase assignment, using 1st and 2nd signatures (and if necessary higher order signatures) to order classes.

As stated previously, **compute-CF** finds a CP transformation which is the inverse of the maximal transformation among the $2^n n!$ possible transformations. Note that there may exist more than one maximal transformation for any function f; however, all such maximal transformations result in the same unique canonical form. Recall that function f will remain invariant under permutations inside an NE-symmetry class. Therefore, instead of finding a transformation on the variables, variables are first partitioned into maximal symmetry classes, next maximal NE-classes $C_1,C_2,\ldots,C_m$ are generated by appropriate phase assignment to members of each class using the **Generate-NE-symmetry** provided in the proof of lemma 1. Next the 1st-signatures are used to construct a maximal transformation on NE-classes if possible. If the 1st-signatures do not contain enough information to generate a maximal transformation, then the 2nd-sinatures (and possibly higher order signatures) will be used to find a maximal transformation.

Example 3. Consider function f in example 2. After applying the **Generate-NE-symmetry**, the NE-classes are: $C_1 = \{x_1, \bar{x}_3, x_4\}$ and $C_2 = \{x_2, \bar{x}_5\}$. Because f is a function of five variables, there exists $2^5 5!$=3840 transformations on variables. However, since f is invariant with respect to permutations inside C_1 or C_2, there are only $2^2 2!$=8 distinct transformations, which result from transformations on NE-classes C_1 and C_2 i.e., (C_1,C_2), $(C_1,\bar{C}_2)$, $(\bar{C}_1,C_2)$, $(\bar{C}_1,\bar{C}_2)$, (C_2, C_1), $(C_2, \bar{C}_1)$, $(\bar{C}_2,C_1)$, and $(\bar{C}_2,\bar{C}_1)$.

To be concise, from this point on, we will use the term "class" to refer to "NE-class" unless stated otherwise. Each transformation on classes determines a transformation on variables (among which the maximal transformation must be identified.) For example $(C_2,\bar{C}_1)$ corresponds to the transformation $(x_2, \bar{x}_5, \bar{x}_1, x_3, \bar{x}_4)$.

By classifying variables to classes and only considering transformations on the resulting classes, variable transformations (and thus, the maximal transformation) are restricted to transformations in which members of a class are NE-symmetric and are placed next to each other.

The next step of **compute-CF** is to assign a phase to each class C_i by using the 1st-signatures of f with respect to C_i and the complement of C_i as follows. (The complement of a class C_i, as the defined before, is a class $\bar{C}_i$ consisting of complements of literals from C_i.) If $\left|f_{\bar{C}_i}\right| < \left|f_{C_i}\right|$, then the positive phase (C_i) is assigned to C_i; otherwise if $\left|f_{\bar{C}_i}\right| > \left|f_{C_i}\right|$, then the negative phase is assigned i.e., literals of C_i will be negated. In case of equality, the phase of the class is marked *undecided*. The phase of undecided classes must be determined in subsequent steps of the algorithm.

Example 4. Continuing with examples 2 and 3, the 1st-signatures of f with respect to the classes and their complements are as follows: $\left|f_{C_i}\right| = 12$, $\left|f_{\bar{C}_i}\right| = 9$, $\left|f_{C_2}\right| = 7$, $\left|f_{\bar{C}_2}\right| = 14$. Since $\left|f_{\bar{C}_1}\right| < \left|f_{C_1}\right|$, positive phase is assigned for C_1 (i.e., $C_1=\{x_1, \bar{x}_3, x_4\}$.) Since $\left|f_{\bar{C}_2}\right| > \left|f_{C_2}\right|$, negative phase is assigned to C_2 (i.e., new $C_2=\{\bar{x}_2, x_5\}$.)

Let $C_1, C_2,\ldots,C_m$ represent the resulting classes after proper phase assignment. (In case of example 4, $C_1=\{x_1, \bar{x}_3, x_4\}$ and $C_2=\{\bar{x}_2, x_5\}$.)

After phases are assigned, new classes are ordered based on their 1st-signatures. (The 1st- signatures used for ordering are values after phase assignment.) Without loss of generality, let's assume that $\left|f_{C_1}\right| \le \left|f_{C_2}\right| \le \ldots \le \left|f_{C_m}\right|$. Thus classes are ordered as (C_1,C_2,…,C_m). If the phase of a class (C_i) is undecided, since $\left|f_{\overline{C}_i}\right| = \left|f_{C_i}\right|$, the phase of C_i (positive or negative) will not affect the outcome of ordering (i.e., $\left|f_{C_i}\right|$ or $\left|f_{\overline{C}_i}\right|$ could be used for ordering). If the 1st-signatures are distinct values for C_1,C_2,…,C_m, then a unique ordering can be achieved since $\left|f_{C_1}\right| < \left|f_{C_2}\right| < \ldots < \left|f_{C_m}\right|$.

Example 5. Continuing with example 4, $f(x_1,x_2,x_3,x_4,x_5)$ = $(x_1+\overline{x}_3+x_4)(\overline{x}_2+x_5)$; the classes after phase assignment will be $C_1=\{x_1, \overline{x}_3, x_4\}$ and $C_2=\{\overline{x}_2, x_5\}$. Furthermore, $\left|f_{C_1}\right|$=12 < $\left|f_{C_2}\right|$=14. Thus, $(C_1,C_2) \equiv (x_1, \overline{x}_3, x_4, \overline{x}_2, x_5)$ will be a maximal transformation. The inverse of this maximal transformation is a CP transformation $T_C=(x_1, \overline{x}_4, \overline{x}_2, x_3, x_5)$ and the canonical form (cf. section III) is: $f^C(X)=f(T_C)=f(x_1, \overline{x}_4, \overline{x}_2, x_3, x_5)=(x_1+x_2+x_3).(x_4+x_5)$.

It is important to notice that the application of the steps of the **compute-CF** algorithm described so far and also the next steps will produce the same canonical form for NE-equivalent functions.

Example 6. Consider two Boolean functions: f_1 $(x_1, x_2, x_3, x_4, x_5)$ $=(x_1+\overline{x}_3+x_4).(\overline{x}_2+x_5)$ and $f_2(x_1, x_2, x_3, x_4, x_5)$ = $(x_1+x_2).(x_3+x_4+x_5)$. For function f_1 a CP transformation, $T_{C_1}=(x_1, \overline{x}_4, \overline{x}_2, x_3, x_5)$, and the canonical form, $f_1^C(X)=(x_1+x_2+x_3).(x_4+x_5)$, were given in example 5. For function f_2, the classes (after phase assignment) and the corresponding 1st-signatiures are: $C_1=\{x_1,x_2\}$ with $\left|f_{C_1}\right|$=14 and $C_2=\{x_3,x_4,x_5\}$ with $\left|f_{C_2}\right|$=12. Since $\left|f_{C_2}\right| < \left|f_{C_1}\right|$, the maximal transformation is $(C_2,C_1) \equiv (x_3,x_4,x_5,x_1,x_2)=T_{C_2}^{-1}$ and the canonical form of $f_2(X)$ is: $f_2^C(X) = f_2(T_{C_2}) = (x_1+x_2+x_3).(x_4+x_5)$ which is equal to the canonical form of f_1: $f_1^C(X) = f_2^C(X)$. Therefore, functions f_1 and f_2 are NP-equivalent.

In this part we consider the case where not all of the 1st-signatures are distinct, i.e., there exist two or more classes with the same 1st-signature. In this case the classes are placed in k groups such that all classes inside a group have the same 1st-signature:

$$\overbrace{C_1^1, C_2^1, \ldots, C_{m_1}^1}^{G_1}, \overbrace{C_1^2, C_2^2, \ldots, C_{m_2}^2}^{G_2}, \ldots, \overbrace{C_1^k, C_2^k, \ldots, C_{m_k}^k}^{G_k}.$$

The superscript j of a class C_i^j indicates that it belongs to the group G_j. Note that $\left|f_{C_1^j}\right| = \left|f_{C_2^j}\right| = \ldots = \left|f_{C_{m_j}^j}\right|$ and groups are ordered based on the 1st-signatures, i.e., $\left|f_{C_i^1}\right| < \left|f_{C_i^2}\right| < \ldots < \left|f_{C_i^k}\right|$. This ordering of groups imposes a restriction on the CP transformation whereby all classes inside a group G_i should precede classes of G_{i+1}. More precisely, candidates for maximal transformation are transformations that respect the ordering of the groups.

Example 7. Consider the Boolean function of a six-input multiplexer: $f = \overline{x}_5\overline{x}_6x_1 + \overline{x}_5x_6x_2 + x_5\overline{x}_6x_3 + x_5x_6x_4$. In this function there is no symmetry between variables i.e., each symmetry class contains only one variable. After doing phase assignment, ordering and grouping classes (in this case, variables), two groups G_1 and G_2 are created: $\overbrace{x_5, x_6}^{G_1:|f_{x_i}|=16}, \overbrace{x_1, x_2, x_3, x_4}^{G_2:|f_{x_i}|=20}$ where for members in G_1 the phase is undecided since $|f_{x_5}| = |f_{\overline{x}_5}| = |f_{x_6}| = |f_{\overline{x}_6}| = 16$ while for G_2 the phase is decided and the 1st-signatures are $|f_{x_i}| = 20$.

In the next step of the **compute-CF** algorithm, the 2nd-signatures are used to order classes inside individual groups and then split the groups into smaller ones based on the outcome of ordering until each group contains only one class at which point a total ordering on classes has been obtained.

We refer to a group as *unresolved* if it contains more than one class or the phases of classes in that group are undecided. (Because all classes in the group have the same 1st-signature, the phase of all classes in that group is decided or none of the phases is decided.) More precisely, if C_1^j and C_2^j belong to the same group and $\left|f_{C_1^j}\right| = \left|f_{\overline{C}_1^j}\right|$, then $\left|f_{\overline{C}_2^j}\right| = |f| - \left|f_{C_2^j}\right| = |f| - \left|f_{C_1^j}\right| = \left|f_{\overline{C}_1^j}\right| = \left|f_{C_1^j}\right| = \left|f_{C_2^j}\right|$.) Let G_u=$\{C_1^u, C_2^u, \ldots, C_{m_u}^u\}$ be the first unresolved group. Since all groups G_1, G_2,…, $G_{u\text{-}1}$ are resolved, (i.e., they contain a single class with decided phase,) the ordering of classes up to G_u is identified. (The case that G_1 is unresolved is discussed later.) Now the 2nd-signatures are used to specify the ordering inside the unresolved groups starting with G_u. Since G_1 is resolved, G_1=$\{C_1^1\}$, 2nd-signatures with respect to C_1^1 and C_i^u (i.e., $|f_{C_1^1C_i^u}|$) can be used for phase assignment (if needed) and ordering classes $C_1^u, C_2^u, \ldots, C_{m_u}^u$ (later on this step will be referred to as iteration 1.) If phase of a class C_i^u is undecided, the 2nd-signatures $|f_{C_1^1C_i^u}|$ and $|f_{C_1^1\overline{C}_i^u}|$ are compared and if $|f_{C_1^1\overline{C}_i^u}| < |f_{C_1^1C_i^u}|$, then a positive phase is assigned to C_i^u; otherwise if $|f_{C_1^1C_i^u}| < |f_{C_1^1\overline{C}_i^u}|$, then a negative phase is assigned to C_i^u. In case of equality of the 2nd-signatures, the phase of C_i^u remains undecided. Next, new values of 2nd-signatures $|f_{C_1^1C_i^u}|$ after phase assignment are used to order classes $C_1^u, C_2^u, \ldots, C_{m_u}^u$ and subsequently regroup these classes. Moore concretely, G_u is split into smaller groups such that inside each group the 2nd-signatures, $|f_{C_1^1C_i^u}|$, are equal. The same procedure (phase assignment, ordering and regrouping based on $|f_{C_1^1C_i^v}|$) is applied to all other unresolved groups G_v. Finally, the indices of new groups and corresponding classes are properly updated. If after these steps, there still exist some unresolved groups G_w, a similar procedure (called iteration 2) is applied based on 2nd-signatures with respect to C_1^2 and C_i^w (i.e., $|f_{C_1^2C_i^w}|$). If needed iterations 3,4,…,u-2 and u-1 are applied. If at iteration u, there still exists some unresolved groups and G_u itself is also unresolved, the procedure described below will be used. (This case includes the case where G_1 is unresolved.)

At this point, group G_u=$\{C_1^u, C_2^u, \ldots, C_{m_u}^u\}$ has been split into two groups such that one of them contains only one class and the other group contains the rest of m_u-1 classes: $G_u \rightarrow \overbrace{C_i^u}^{G_u}, \overbrace{C_1^u, \ldots, C_{i-1}^u, C_{i+1}^u, \ldots, C_{m_u}^u}^{G_{u+1}}$. (The indices of all subsequent groups are shifted by one.) As can be seen, there are m_u ways to split the group G_u (m_u ways to specify new G_u corresponding to i=1,2,…,m_u in the above relation) and if the phase of C_i^u is undecided, then there will be two ways to resolve the group, new G_u, (positive or negative phases.) Therefore there are m_u (or $2m_u$) ways to specify and resolve the new group, G_u. All these m_u (or $2m_u$) cases need to be tracked, since it is unknown which one(s) will result in a maximal transformation. For each case, the 2nd-signatures are used to first order classes inside the unresolved groups among G_{u+1},…, G_k and then split them based on the outcome of ordering as follows.

Assume that G_u is split into two following groups: $G_u \rightarrow \overbrace{C_i^u}^{G_u},\overbrace{C_1^u,\ldots,C_{i-1}^u,C_{i+1}^u,\ldots,C_{m_u}^u}^{G_{u+1}}$, after shifting the indices of subsequent groups and making corresponding changes to superscripts of classes, new groups $G_u,G_{u+1},\ldots,G_k$ (where k represents the new number of groups) are represented as: $\overbrace{C_1^u}^{G_u},\overbrace{C_1^{u+1},C_2^{u+1},\ldots,C_{m_{u+1}}^{u+1}}^{G_{u+1}},\ldots,\overbrace{C_1^k,C_2^k,\ldots,C_{m_k}^k}^{G_k}$. At this point (iteration u) similar to the procedure explained before, for unresolved groups G_v, the 2nd-signatures $|f_{C_1^1 C_i^v}|$ are used for phase assignment, ordering and regrouping. This process will continue for all m_u (or $2m_u$) cases, recursively (cf. the **recursive-resolve** algorithm), until all groups are resolved.

All resulting transformations T_1, $T_2,\ldots,T_r$ corresponding to different cases are stored. These transformations are superior with respect to relation '$\prec$', considering only the1st and 2nd signatures because of the way they have been constructed. The maximal transformation(s) is identified among these transformations. As a result of the way in which these transformations are obtained, they all have the same set of 1st and 2nd signatures. Some of these transformations are equivalent to each other because of special types of symmetry (excluding the type of symmetry discussed in section IV.) Examples of such special symmetries include hierarchical, group and rotational symmetries [4] i.e., for some T_i and T_j , the equality $f(T_i^{-1})=f(T_j^{-1})$ may hold, in which case only one of T_i and T_j is kept and the other one is removed from the list T_1, $T_2,\ldots,T_r$. This process continues until there is no equivalency among the remaining transformations.

Experimental results show that for nearly all Boolean functions only one transformation will remain in the list; i.e., all transformations T_1, $T_2,\ldots,T_r$ are equivalent: $f(T_1^{-1})=f(T_2^{-1})=\ldots=f(T_r^{-1})$. In this case all transformations, T_1, $T_2,\ldots,T_r$ are canonical and the canonical form is $f^C(X)=f(T_1^{-1})=\ldots=f(T_r^{-1})$. In other words, in most cases, only the 1st and 2nd signatures can identify the maximal transformation(s).

Example 8. Continuing with the multiplexer function of example 7,

$$f(X) = f(x_1,x_2,x_3,x_4,x_5,x_6) = \bar{x}_5\bar{x}_6x_1 + \bar{x}_5x_6x_2 + x_5\bar{x}_6x_3 + x_5x_6x_4$$

and variables are grouped as $\overbrace{x_5,x_6}^{G_1:|f_{x_i}|=16},\overbrace{x_1,x_2,x_3,x_4}^{G_2:|f_{x_i}|=20}$. Both groups G_1 and G_2 are unresolved. Group G_1 can be split into two groups in two ways: $G_1 \rightarrow \overbrace{x_5}^{G_1},\overbrace{x_6}^{G_2}$ or $G_1 \rightarrow \overbrace{x_6}^{G_1},\overbrace{x_5}^{G_2}$. For each of these cases there are two ways to assign phase to G_1. Therefore, there are four ways to specify and resolve the new G_1: $\overbrace{x_5}^{G_1},\overbrace{x_6}^{G_2}$ or $\overbrace{\bar{x}_5}^{G_1},\overbrace{x_6}^{G_2}$ or $\overbrace{x_6}^{G_1},\overbrace{x_5}^{G_2}$ or $\overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2}$. The algorithm keeps track of all these cases. Let's zoom in one of the cases, e.g., $G_1 \rightarrow \overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2}$ results in the following new grouping: $\overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2},\overbrace{x_1,x_2,x_3,x_4}^{G_3}$. Now, G_2 and G_3 are unresolved. G_2 is unresolved because the phase of x_5 is undecided. First we try to resolve G_2 (decide a phase for x_5) using 2nd-signatures $|f_{\bar{x}_6x_5}|$ and $|f_{\bar{x}_6\bar{x}_5}|$. Since $|f_{\bar{x}_6x_5}|=|f_{\bar{x}_6\bar{x}_5}|=8$ the phase of x_5 can not be decided and the group G_2 remains unresolved. Next, we try to resolve G_3 using 2nd-signatures $|f_{\bar{x}_6x_1}|=12$, $|f_{\bar{x}_6x_2}|=8$, $|f_{\bar{x}_6x_3}|=12$ and $|f_{\bar{x}_6x_4}|=8$. (Recall that the phase of literals in G_3 is decided.) Since $|f_{\bar{x}_6x_2}|=|f_{\bar{x}_6x_4}|<|f_{\bar{x}_6x_1}|=|f_{\bar{x}_6x_3}|$, the group G_3 is split into two groups: $G_3 \rightarrow, \overbrace{x_2,x_4}^{G_3},\overbrace{x_1,x_3}^{G_4}$ and the overall grouping will be $\overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2},\overbrace{x_2,x_4}^{G_3},\overbrace{x_1,x_3}^{G_4}$. At this point, G_2, G_3 and G_4 are still undecided. There are two ways to resolve G_2: $\overbrace{x_5}^{G_2}$ or $\overbrace{\bar{x}_5}^{G_2}$. Following one of these cases, for example $\overbrace{x_5}^{G_2}$, the overall grouping will be $\overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2},\overbrace{x_2,x_4}^{G_3},\overbrace{x_1,x_3}^{G_4}$ where G_1 and G_2 are resolved and G_3 and G_4 are unresolved. Now we try to resolve G_3 using 2nd-signatures $|f_{x_5x_2}|=8$ and $|f_{x_6x_4}|=12$. Since $|f_{x_5x_2}|<|f_{x_6x_4}|$, the group G_3 is split to $\overbrace{x_2}^{},\overbrace{x_4}^{}$ and similarly for G_4 since $8=|f_{x_5x_1}|<|f_{x_6x_3}|=12$, the group G_4 is split to $\overbrace{x_1}^{},\overbrace{x_3}^{}$ which result in the following ordering of resolved groups: $\overbrace{\bar{x}_6}^{G_1},\overbrace{x_5}^{G_2},\overbrace{x_2}^{G_3},\overbrace{x_4}^{G_4},\overbrace{x_1}^{G_5},\overbrace{x_3}^{G_6}$ which corresponds to the transformation T_1=($\bar{x}_6,x_5,x_2,x_4,x_1,x_3$). We only followed one case out of eight cases; (four ways to resolve G_1 multiplied by two ways to resolve G_2.)

Following all cases will result in overall eight transformation including T_1. Among other seven cases, one for example is T_2=(x_5, $\bar{x}_6$,x_2,x_1,x_4,x_3). However, it can be seen that:

$$f(T_1^{-1}) = f(x_5,x_3,x_6,x_4,x_2,\bar{x}_1) = \bar{x}_2x_1x_5 + \bar{x}_2\bar{x}_1x_3 + x_2x_1x_6 + x_2\bar{x}_1x_4 \ ,$$

$$f(T_2^{-1}) = f(x_4,x_3,x_6,x_5,x_1,\bar{x}_2) = \bar{x}_1x_2x_4 + \bar{x}_1\bar{x}_2x_3 + x_1x_2x_6 + x_1\bar{x}_2x_5 \,.$$

Since $f(T_1^{-1}) = f(T_2^{-1}) = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_4 + x_1\bar{x}_2x_5 + x_1x_2x_6$, transformations T_1 and T_2 are equivalent. All eight transformations will result in the same function, which implies that any one of them, e.g., T_1, is a maximal transformation and the canonical form is $f^C(X) = f(T_1^{-1}) = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_4 + x_1\bar{x}_2x_5 + x_1x_2x_6$.

If at this point, more than one transformation is left in the list; the 1st and 2nd signatures can no longer be utilized. For the remaining transformations T_1, $T_2,\ldots,T_l$, the *partial signature vectors* P^{T_1}, P^{T_2}, …, P^{T_l} are generated where *partial* means the 1st and 2nd signatures are excluded and only the 3rd and higher order signatures are used. For example, if $T=(t_1,t_2,\ldots,t_n)$, then

$$P^T = (\overbrace{|f_{t_1t_2t_3}|,|f_{t_1t_2t_4}|,\ldots,|f_{t_{n-2}t_{n-1}t_n}|}^{3^{rd}-\text{signatures}},\ldots,\overbrace{|f_{t_1\ldots t_{n-1}}|,\ldots,|f_{t_2\ldots t_n}|}^{(n-1)^{th}-\text{signatures}},\overbrace{|f_{t_1\ldots t_n}|}^{\text{n}^{th}-\text{signature}})\,.$$

Since for all pairs T_i and T_j among T_1, $T_2,\ldots,T_l$,, we have $f(T_i^{-1}) \neq f(T_j^{-1})$, as a result of theorem 1, all respective partial signature vectors $P^{T_1},P^{T_2},\ldots,P^{T_l}$ are different. Hence, a unique maximal transformation (with respect to partial signature vectors), T_C (which will be canonical) can be identified among T_1, $T_2,\ldots,T_l$,.

In other words, the maximal transformation(s) has the maximal partial signature vector with respect to the lexicographical ordering '$\prec$' discussed in section IV, among the transformations $T_1,T_2,\ldots,T_l$. For nearly all functions, there is no need to generate all signatures for identifying the maximal transformation(s) with respect to'$\prec$'. Instead, to compare two transformations T_i and T_j, at the first step, only 3rd-signatures are generated and comparison is performed with respect to 3rd-signatures only. If the corresponding 3rd-signatures are equal, then 4th-signatures are generated and compared and this procedure continues until an inequality is encountered. As a result of theorem 1, it is guaranteed that at some point, an inequality will occur, because $f(T_i^{-1}) \neq f(T_j^{-1})$. Different steps of the proposed techniques are summarized in the following pseudo-code descriptions of **compute-CF** and **recursive-resolve** algorithms.

```
Algorithm compute-CF (f)
Input: A Boolean function $f(X)$
Output: The canonical form $f^C(X)$
Classify variables to NE-classes using generate-NE-symmetry;
Using the1st-signatures, assign phases, order and group classes to
        groups $G_1,G_2,\ldots,G_K$
recursive-resolve($G_1,G_2,\ldots,G_K$; $T$)
Set the canonical form: $f^C(X)=f(T^{-1})$
```

```
Algorithm recursive-resolve ($G_1,G_2,\ldots,G_K$; $T$)
Input: Ordered groups ($G_1,G_2,\ldots,G_K$)
Output: A transformation, $T$
 $T$=none; $i$=1; // where "$none \prec T_i$" is true for all transformations $T_i$
 while ($i<m+1$) { // m is the number of classes
    if ($G_i$ is resolved) { // $G_i=[C_i]$
        for (all unresolved groups $G_j=[C_{j1}, C_{j2},\ldots, C_{jl},\ldots]$) {
            use signatures $|f_{C_iC_{jl}}|$ to assign phase, order and split $G_i$;
            update indices of groups and classes; }
        i = i + 1;
    } else { // $G_i =[C_{i1}, C_{i2},\ldots, C_{jj},\ldots]$ is not resolved
        for (j=1; j<$|G_i|$; j++) {
            split $G_i$ to groups $[C_{ij}]$ and $[C_{i1}, C_{i2},\ldots, C_{j(j-1)}, C_{j(j+1)},\ldots]$;
            // for space limitation assume the phase of $[C_{ij}]$ is decided
            update indices of groups and classes: ($G_1,G_2,\ldots,G_K,G_{K+1}$);
            recursive-resolve ($G_1,G_2,\ldots,G_K,G_{K+1}$; $T_{TEMP}$) ;
            if ( $f(T^{-1})==f(T_{TEMP}^{-1})$ )
                continue;
            if ( $T \prec T_{TEMP}$ ) // Based on their partial signature vectors
                $T= T_{TEMP}$;}
        return; }
 }
// At this point there are $m$ groups and all of them are resolved
// T =($G_1,G_2,\ldots,G_n$); i.e., each group contains a single class
```

VI. Experimental Results

The technique presented above has been implanted as part of the SIS logic synthesis environment. To reveal the effectiveness of the proposed technique, the proposed canonical form is computed for all cells in a cell library, containing a large number of complex cells with up to 20 inputs. To asses the efficiency of the method, a large number of randomly generated logic cells with different input counts were added to the library. Figure 1 shows the worst-case and average run-times required for computing the canonical form in terms of the number of inputs; i.e., the height of the nth bars are the worst-case and average runtimes for all n-input cells.

The run-times in this Figure 1 (Y-axis) are in microseconds and include data for cells with more than five inputs. This is because run-times for cells with as few as five inputs are too small (less than 1μ-sec) to be discernible. As an example the worst-case 20-input cell was a multiplexer with four select inputs for which the algorithm takes 240 microseconds to compute its canonical form.

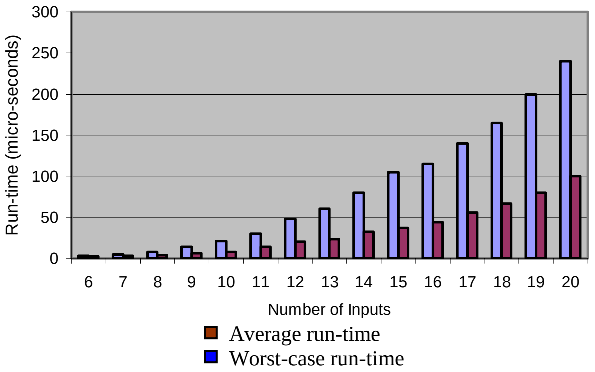

Figure 1. Worst-case and average run time to compute canonical forms

These results show a major improvement in run-time over previous approaches [9][10]. (Notice that reference [9] does not handle complementation of inputs and reference [10] entails enormous space complexity.) For nearly all of the cells in the library, the canonical forms were computed using only the 1st and 2nd signatures. Only one of the cells required the use of the 3rd signatures and none of them required the use of higher order signatures. However, the algorithm given above is complete and able to handle functions that may require the use of higher order signatures for computing the canonical form.

VII. Conclusions

A new efficient and compact canonical form was defined and an effective algorithm for computing the proposed canonical form was provided in this paper. The compactness and efficiency of the presented methods enables the approach to be applicable to a wide range of Boolean networks as apposed to previous approaches that either do not solve the problem generally or only handle functions with limited number of inputs. This paper addresses the general Boolean matching problem in which both permutation and complementation of inputs are considered. The proposed canonical form was based on using generalized signatures to obtain a CP transformation on inputs. Signatures were defined very effectively and first, most powerful signatures (that include more information about the function) are generated and used followed by less significant signatures, only if necessary. Experimental results demonstrate the efficiency of the proposed approach and it was observed, in nearly all cases 1st and 2nd signatures are enough to provide the canonical form and since handling 1st and 2nd signatures is performed efficiently by ordering variables, the proposed approach is associated with a very low computational complexity.

VIII. Acknowledgment

This work was sponsored in part by a grant from Magma Design Automation Inc.

IX. References

[1] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.

[2] L. Benini and G. De Micheli, "A survey of Boolean matching techniques for library binding," *ACM Trans. Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 193–226, July 1997.

[3] M. A. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, 1965.

[4] J. Mohnke, P. Molitor, and S. Malik, "Limits of using signatures for permutation independent Boolean comparison," *Proc. of ASP Design Automation Conf.*, pp. 459-464, 1995.

[5] J. R. Burch and D. E. Long, "Efficient Boolean function matching," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 408–411, Nov. 1992.

[6] Q. Wu, C. Y. R. Chen, and J. M. Acken, "Efficient Boolean matching algorithm for cell libraries," *Proc. IEEE Int. Conf. on Computer Design*, pp. 36–39, Oct. 1994.

[7] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," *Proc. of Design Automation Conf.*, pp. 206–211, June 1998.

[8] D. Debnath and T. Sasao, "Fast Boolean matching under permutation using representative," *Proc. ASP Design Automation Conf.*, pp. 359–362, Jan. 1999.

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

[10] D. Debnath and T. Sasao, "Efficient computation of canonical form for Boolean matching in large libraries," Proc. *ASP Design Automation Conf.*, pp. 591–596, Jan. 2004.

[11] C. R. Edwards and S. L. Hurst, "A digital synthesis procedure under function symmetries and mapping methods", *IEEE Trans. Comp.*, Vol. C-27, No. 11, pp. 985-997, NOV. 1978.