Boolean Matching Using Generalized Reed-Muller Forms

Chien-Chung Tsai
Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106 USA

Abstract -- In this paper we present a new method for Boolean matching of completely specified Boolean functions. The canonical Generalized Reed-Muller forms are used as a powerful analysis tool. Input permutation, as well as input and output negation for matching are handled simultaneously. To reduce the search space for input correspondence, we have developed a method that can detect symmetries of any number of inputs simultaneously. Experiments on MCNC benchmark circuits are very encouraging.

1. Introduction

One of the critical steps in technology mapping is to decide whether or not a subnetwork can be implemented by any of the library cells, perhaps with inverters on some of the input or output lines. In logic verification, descriptions of the logic from different stages of the design process are compared to check if they represent the same function. A conclusive answer to either problem requires extensive computation. This is due to the fact that, in most cases, the input variable correspondence is not known in advance. In recent years, Boolean matching has been proposed, where networks are converted to their Boolean function representations and matching is decided by the equivalence check on the appropriate functions [2], [3], [6], [7], [9], [10].

Three major equivalence classes have been discussed before. Two *n*-input Boolean functions are equivalent if one can be transformed into the other by one or more of the following transformations:

(P1) input permutations,

(P2) input negation, also known as phase assignment of input variables, and

(P3) output negation or phase assignment of the output.

Functions are *p-equivalent* under P1 and *np-equivalent* under P1 and P2. *npn-equivalent* functions allow all three transformations.

The problem addressed in this paper is stated as follows. Given two completely specified Boolean functions *f* and *g*, which have the same number of inputs, we ask if *f* and *g* are *npn*-equivalent; and, if they are, we wish to know the transformation. Note that, all of the previous work in Boolean matching check for P1 and P2 transformations separately.

Several approaches have been used to expedite the decision of equivalence. Signatures of the functions and of the individual variables are widely utilized for this purpose [3], [6], [7], [10]. Symmetry property among variables is another important characteristic [3], [6], [7], [8], [9], [10]. However, only one type of symmetry is checked and the method of checking is very inefficient.

In this paper, we propose a uniform and efficient method for solving the Boolean matching problem. Our method is based on the Generalized Reed-Muller (GRM) representations of Boolean functions. The GRM form is also used to verify four different types of symmetry [12].

For fixed polarities of all vaiables, GRM forms are canonical representations. Therefore, the number of cubes with different lengths for the function and for each variable can be used as meaningful signatures. The signatures of variables are used both for symmetry check and Boolean matching. A certain kind of reduced-ordered binary decision diagram (ROBDD) [5], [11] to represent GRMs is used in our method. With this data structure, all of our operations are efficiently carried out in a ROBDD[1] package without any extra implementation.

In the remainder of this paper, we discuss only the single-output functions. Multi-output functions are handled by treating each output function independently. For the purpose of technology mapping, the majority of library cells are single-output functions.

2. Definitions and Terminology

Let $f(x_1, x_2, ..., x_n)$ be a completely specified Boolean function. $|f|$ denotes the number of the on-set minterms of *f*. We will use t_i to represent the literal of variable x_i, t_i can be either x_i or $\bar{x}_i$. The length of a cube *p*, denoted $|p|$, is the number of literals in the cube. $S(p)$ denotes the set of variables in cube *p*.

A variable x_i is *balanced*, if $|f_{x_i}| = |f_{\bar{x}_i}|$. Otherwise, x_i is *unbalanced*. A function *f* is *neutral*, if $|f| = 2^{n-1}$. *f* is *odd*, if $|f|$ is an odd integer. Otherwise, it is *even.*

A *Boolean difference* of *f* with respect to a variable x_i, denoted $f^B_{x_i}$, is defined as $f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ..., \bar{x}_i, ..., x_n)$. It can be computed from the formula $f^B_{x_i} = f_{x_i} \oplus f_{\bar{x}_i}$. The definition of Boolean difference with respect to an arbitrary cube $p = t_i\, t_j ... t_k$ is recursively defined as:

$$f^B_p = (\ldots((f^B_{t_i})^B_{t_j})\ldots)^B_{t_k}.$$

The following properties of the Boolean difference operator follow directly from the definition (a) $f^B_{\bar{x}_i} = f^B_{x_i}$ (b) $f^B_{x_i x_j} = f^B_{x_j x_i}$. These properties imply that, for two arbitrary cubes p_1 and p_2, $f^B_{p_1} = f^B_{p_2}$, if $S(p_1) = S(p_2)$.

Using the Shannon expansion, a function can be expressed

as $f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$, or equivalently as $f = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$. By applying the identity $\bar{x}_i = x_i \oplus 1$, we can derive

(c) $f = x_i f^B_{x_i} \oplus f_{\bar{x}_i}$, or (d) $f = \bar{x}_i f^B_{x_i} \oplus f_{x_i}$.

Each equation has two terms: one contains the literal t_i and the other does not. We will call these *pole-branch* and *dc-branch*, respectively. The process of XORing the two cofactors is referred to as folding.

With each *n*-input function *f* we associate a binary *n*-dimensional *polarity vector.* An entry of the vector is 0(1) if the corresponding variable in GRM form is in the negative (positive) polarity.

For each variable x_i in *f*, the *major pole (M-pole)* is *one* if $|f_{x_i}| > |f_{\bar{x}_i}|$ and is *zero* if $|f_{x_i}| < |f_{\bar{x}_i}|$. The *minor pole (m-pole)* is the one corresponding to the smaller of the two. We will call the polarity vector *M-pole (m-pole)* with respect to *f*, denote *M (m)*, when each variable is assigned the *M*-pole (*m*-pole). Note that, for balanced variables, the *M*-pole/*m*-pole cannot be decided. The *M*-pole/*m*-pole always exists for odd functions, since every variable is unbalanced.

3. Generalized Reed-Muller Forms

3.1 General properties

A GRM form of a Boolean function is the XOR sum of cubes, in which every variable has either positive or negative, but not both, polarity in all cubes.

Consider two functions of three variables: $f(x_1, x_2, x_3) = \Sigma(2, 3, 5, 6, 7)$, $g(y_1, y_2, y_3) = \Sigma(0, 2, 3, 4, 6)$. Their representations in GRM forms are $f_U = x_1x_3 \oplus x_2 \oplus x_1x_2x_3$ and $g_V = \bar{y}_1y_2 \oplus \bar{y}_3 \oplus \bar{y}_1y_2\bar{y}_3$, where *U = (1 1 1)* and *V = (0 1 0)*.

These two functions are *np*-equivalent by matching variables as $\{x_1 \to y_2, x_2 \to \bar{y}_3, x_3 \to \bar{y}_1\}$ or $\{x_1 \to \bar{y}_1, x_2 \to y_3, x_3 \to y_2\}$. Note that the GRM forms explicitly display the np-equivalence of the two functions. The polarities of variables in either function do not change the equivalence. We say that *matching condition* or *equivalence* of GRM forms is fulfilled, if the variables (polarities ignored) in two GRMs can be matched such that all the cubes match.

The key question in our method is how to select each variable's polarity before the two functions are matched. There are 2^n possible combinations of polarities for *n* variables. Any Boolean function can be represented in 2^n GRM forms. We will use f_V to represent the GRM form of a function *f* under the polarity vector *V.* Note that the number of cubes for a function varies with different polarity vectors. The selection of polarities will determine the GRM forms that will be used for matching. The following theorems give conditions for Boolean matching of functions for which *M*-poles exist. The proofs are omitted due to space limitations.

Theorem 1: Suppose that the *M*-pole (m-pole) exist for the functions *f* and *g*. Then *f* and *g* are *np*-equivalent, if, and only if, their GRM forms under *M*-pole (m-pole) are equivalent.

Theorem 2: Let *f* be a Boolean function and $\bar{f}$ be its complement. Then, for any polarity vector *V*, we have $\bar{f}_V = f_V \oplus 1$. Let *M* and *m* be the *M*-pole and *m*-pole of *f*, respectively. Then the *m*-pole vector for $\bar{f}$ is *M* and the *M*-pole vector for $\bar{f}$ is *m*; i.e., the roles of the major pole and minor pole vectors are reversed in the complement.

To generate compatible GRMs, we apply the following rules to all functions: (1) for functions with $|f| < 2^{n-1}$, we use M-pole vector, (2) for functions with $|f| > 2^{n-1}$, we first derive $\bar{f}$ and use *M*-pole of $\bar{f}$, (3) for *neutral* functions, we generate both *M*-pole and *m*-pole GRMs of *f*.

To determine the equivalence of two functions, both functions will be transformed to their canonical GRM forms. A set of signatures, discussed later, will then be used to indicate any discrepancies. Symmetry is also checked at this stage. If all signatures match, then we can compare the actual functions in the GRM forms. At this stage, the concern is on matching of variables and cubes. After the functions are matched, the phase assignment of input variables can be decided with the comparison of polarities between the corresponding variables of the two functions. Different polarity between the corresponding variables means an inverter is needed to bring them to a common phase.

Note that, in our method, we do not have to consider the input negation as a separate task and perform additional computations. The input negation (similar to the output negation) is determined as a side effect of the matching condition.

3.2 Functional decision diagram

The data structure for GRM forms is called *Functional Decision Diagram(FDD)* [5]. It can be derived efficiently [5], [11] and the size is, in general, smaller than that of the conventional ROBDD. It is a binary acyclic graph in which nodes are labeled *0* or *1* and each nonterminal node is labeled with a variable. The two edges for each nonterminal node have the attribute *0* or *1*. The order in which the variables appear along each path is fixed and the graph has no isomorphic subgraphs. The root of the graph represents the function. A polarity vector is maintained with the FDD. For each nonterminal node labeled x_i, the edge corresponding to the polarity of x_i is the *pole-branch* and indicates that the corresponding literal appears in the cube. The edge with an attribute opposite to the polarity of x_i is the *dc-branch* and indicates that x_i is not in the cube. Each path which starts from the root and terminates at the terminal *one* node represents a set of cubes in the GRM form of *f*. Any missing node, corresponding to the variable x_j, in the path represents two cubes in the GRM. One cube contains x_j with the appropriate polarity and the other cube does not have x_j. Therefore, a path with *k* nonterminal nodes stands for a set of 2^{n-k} cubes in the GRM [11].

An important operation in our Boolean matching method is the equivalence checking of two GRM forms. It's execution on FDD is similar to the equivalence checking of two functions in ROBDD forms. Assume the variable orderings are matched in the two FDDs. Starting from the root, the equivalence check is recursively called at each branch of a node and terminates at the leaf nodes of the two FDDs. At each node, we check first to be certain that the variables corresponding to the nodes in the two FDDs are the same. Then the polarity of the presently processed variable x_i is retrieved from the polarity vectors for each FDD and the dc and pole branches are identified. Then we check the equivalence of the corresponding dc and pole branches with recursive calls. All operations

and FDD representations can reside in an ROBDD package.

3.3 Prime cubes in the GRM forms

A cube p is *prime*[4] in f if $f_p^B = 1$. We observe that every variable in a prime cube can assume either polarity without violating the definition of prime cubes. In [4], Csanky *et al* have proved that all the prime cubes occur in every GRM form of *f*. Polarities of the variables in prime cubes follow that of the polarity vector of the residing GRM form. The fact that all prime cubes are essential makes the set of prime cubes very unique in identifying a function.

The detection of the prime cubes is very straightforward. Csanky *et al* [4] proved that p is a prime, if, and only if, p is the only cube that contains all of *S(p)*. In other words, all the cubes in any GRM form with maximum cardinality are primes. These might not be all the primes. In the function $f = x_1 \oplus x_2x_3 \oplus x_2 \oplus x_3x_4$, x_2x_3 and x_3x_4 are both primes; x_1 is also a prime but not one of the largest cardinality. Assume that p is a prime cube, then any cube p' that satisfies $S(p') \subset S(p)$ is not a prime. Ignore all the longest primes and all the cubes that are composed of subsets of their literals. If there is any cube left, again, we will look for the longest cubes. This process will continue until all the cubes are accounted for.

4. Signatures for Boolean Matching and Symmetry Detection

Signatures are values that characterize the function or the variables in a function.

4.1 Signatures from on-set weight

The on-set weight of the function can easily be computed by traversing the ROBDD. For each variable x_i, the two weights $|f_{x_i}|$ and $|f_{\bar{x}_i}|$ are caled *positive cofactor weight (pcw)* and *negative cofactor weight (ncw)*, respectively. To ensure consistancy with the matching of GRM, all functions with $|f| > 2^{n-1}$ will have weights computed on $\bar{f}$.

On the functional level, two types of signatures derive from this source. The first, *functional weight (fw)*, is the value $|f|$. The second, *weight distribution vector (wd)*, is the set of values indicating how many different *pcw* and *ncw* pairs there are in the function.

On the variable level, the *(ncw, pcw)* pair is the signature for each variable.

Theorem 3: Let f and g be *np*-equivalent and assume that x_i and y_j are the matching variables from f and g, respectively. Then f and g have the same pairs of numbers for *pcw* and *ncw*.

4.2 Signatures from GRM form

We propose three sets of signatures that are obtained from GRM form. All the signatures described below require *O(kn)* time complexity, where k is the number of nodes in the FDD and n is the number of variables.

4.2.1 Distributions of cubes with different lengths

For each GRM, we first compute an n-by-n matrix of *variable inclusion count, VIC* = (a_{ij}), where a_{ij} is the number of cubes of length i that contain variable x_j. At the same time, we compute incrementally the number of cubes of each length for the entire function. At the functional level, this becomes an n element vector *FC*, where each entry i contains the number of cubes in the GRM of length i. Note that if the cube *1* is in the GRM form, we need to store the information in a separate location.

The second array on the functional level is *FVC* computed from the *VIC* by summing entries of each column, so that each entry of *FVC* is the number of cubes containing variable x_i.

4.2.2 Distributions of cubes with related variables

During the same traversal of FDD as in the above subsection, we also compute an n-by-n symmetric matrix, the *incidence matrix, INC* = (a_{ij}), where a_{ij} is the number of cubes containing both variables x_i and x_j. The diagonal entry a_{ii} is *0* if single literal cube x_i is not present; else it is *1*. This is a signature set at the variable level.

A functional level signature *FINC* is also generated from *INC*. We compute an n element array by summing up each row (or column), except the diagonal entry. Each entry in *FINC* represents the total frequency of occurrences of each variable.

4.2.3 Prime cubes

In *VIC*, each column j represents the cubes of various lengths that contain x_j, for each variable x_j. The number of prime cubes, that contain x_i, is the last nonzero number in the column. This value is saved separately as an array *PCV* of all variables. On the functional level, *PC* is the total number of prime cubes. This is easily derivable from *PCV.*

There are two more matrices on the variable level: they are the versions of *VIC* and *INC* calculated only on prime cubes. They are (1)an n-by-n matrix, *PCvic* = (a_{ij}), where a_{ij} is the number of prime cubes of length i that contain x_j, and (2) an n-by-n matrix *PCinc* = (a_{ij}), where a_{ij} is the number of prime cubes that contains both variables x_i and x_j.

5. Symmetries and Linear Variables

For any pair of variables x_i and x_j, there are four cofactors, namely, $f_{\bar{x}_i\bar{x}_j}, f_{\bar{x}_ix_j}, f_{x_i\bar{x}_j}, f_{x_ix_j}$. Equivalence between any two of the four cofactors, with the choice of negating one of them, form 12 different symmetry relations (6 from both positive, 6 from negating one of the two). Theoretically, any one or more of the 12 cases can indicate certain relationships between the two variables. We can use some of them to partition the input variable set into different equivalence classes. Checking all the symmetries can be time consuming. We have discovered that four among them are very closely related and can be verified simultaneously in the GRM forms [12]. These four types of symmetries form transitive relations and are very useful for the purpose of matching.

5.1 Positive symmetry

5.1.1 The nonequivalence symmetry

A function f exhibits a nonequivalence symmetry (*NE* symmetry) in variables x_i and x_j, denoted as x_i *NE* x_j or $\{x_i, x_j\}$, if f remains invariant when the two variables are interchanged, or equivalently, if $f_{\bar{x}_ix_j} = f_{x_i\bar{x}_j}$. Note that $\{x_i, x_j\}$ is the same as $\{\bar{x}_i,$

$\bar{x}_j\}$ in terms of the definition. To detect the *NE* symmetry in the GRM form, note that $f_{\bar{x}_i x_j} = f_{x_i \bar{x}_j}$, if, and only if, $f_{\bar{x}_i \bar{x}_j} \oplus f_{\bar{x}_i x_j} = f_{\bar{x}_i \bar{x}_j} \oplus f_{x_i \bar{x}_j}$. When the polarities of x_i and x_j are the same, x_i *NE* x_j can be detected in the GRM.

5.1.2 The equivalent symmetry

When $f_{\bar{x}_i \bar{x}_j} = f_{x_i x_j}$, the function is said to exhibit equivalence symmetry (*E* Symmetry) with respect to x_i and x_j. It is denoted x_i *E* x_j, or $\{x_i, \bar{x}_j\}(\{\bar{x}_i, x_j\})$. To detect *E* symmetry in the GRM form, note that $f_{\bar{x}_i \bar{x}_j} = f_{x_i x_j}$, if, and only if, $f_{\bar{x}_i \bar{x}_j} \oplus f_{\bar{x}_i x_j} = f_{x_i x_j} \oplus f_{\bar{x}_i x_j}$. When the polarities of x_i and x_j are different, x_i *E* x_j can be detected in the GRM.

Theorem 4: If x_i *E* x_j and x_j *E* x_k, then x_i *NE* x_k [8].

5.1.3 Mixed symmetries

The transitive condition of *E* symmetry tells us that *NE* and *E* symmetries are related. Using GRM forms, we can detect both types of symmetry by applying the same procedure. The only difference is in the polarity combinations of the two variables. Therefore, we can group variables with the two types of symmetries together; i.e., if $\{x_i, \bar{x}_j\}$ and $\{x_j, \bar{x}_k\}$, then $\{x_i, \bar{x}_j, x_k\}(\{\bar{x}_i, x_j, \bar{x}_k\})$ is a positive symmetric set of three variables.

Theorem 5: If x_i *NE* x_j and x_i *E* x_j, then x_i and x_j are both balanced variables.

This theorem sets up a necessary condition for two variables to be both *E* and *NE* symmetric.

Theorem 6: Suppose both x_i and x_j are unbalanced and both variables have *M*-pole (*m*-pole) in the polarity vector *V*. Then f_V will show the symmetry, if, and only if, x_i *NE* x_j or x_i *E* x_j.

This theorem makes the detection of positive symmetries span uniformly across unbalanced variables. The existance of *M*-pole GRM is enough to conclude any positive symmetry among multiple variables. We do not have to check the *NE* symmetry and *E* symmetry separately for each pair of variables, as the conventional method requires.

Theorem 7: Any positive symmetry occurs between x_i and x_j in *f*, if, and only if, they are symmetric in the complement $\bar{f}$.

5.1.4 Total symmetry

A function is *totally symmetric* if every pair of variables in the function is positive symmetric. This implies that every pair of variables will be symmetric in a GRM form. For functions with *M*-pole vector, the following theorem makes checking for total symmetry very simple.

Theorem 8: A function *f* is totally symmetric, if, and only if, there exists a polarity vector *V* such that for each *k*, $1 \le k \le n$, f_V either contains no cube of length *k* or it contains all cubes of length *k*.

To check for total symmetry, we only need to verify, for each *k* from *1* to *n*, whether f_M contains *0* or C_k^n cubes of length *k*, where C_k^n is the combination of *n* choose k.

5.2 Negative symmetry

5.2.1 The skew-nonequivalence symmetry

When $f_{\bar{x}_i x_j} = \overline{f_{x_i \bar{x}_j}}$, the function is said to be skew-nonequivalence symmetric(skew-*NE* symmetric) with respect to x_i and x_j. It is denoted *by* x_i *!NE* x_j. To detect skew-*NE* symmetry in the GRM form, note that $f_{\bar{x}_i x_j} = \overline{f_{x_i \bar{x}_j}}$, if, and only if, $f_{\bar{x}_i \bar{x}_j} \oplus f_{\bar{x}_i x_j} = f_{\bar{x}_i \bar{x}_j} \oplus f_{x_i \bar{x}_j} \oplus 1$. When the polarities of x_i and x_j are the same, x_i*!NE* x_j can be detected in the GRM [12].

The only difference in the GRM form between *NE* symmetry and skew-*NE* symmetry is the extra term *1* in the above discussion. This term *1* is a single literal cube x_i or x_j in the function. The detection can be done similarly to the *NE*-symmetry.

Theorem 9: Any two of the conditions x_i*!NE* x_j, x_j*!NE* x_k, and x_i *NE* x_k implies the third.

5.2.2 The skew-equivalence symmetry

When $f_{\bar{x}_i \bar{x}_j} = \overline{f_{x_i x_j}}$, the function is said to exhibit skew-equivalence symmetry (skew-*E* symmetry) with respect to x_i and x_j. It is denoted *by* x_i*!E* x_j. To detect skew-*E* symmetry in the GRM form, note that $f_{\bar{x}_i \bar{x}_j} = \overline{f_{x_i x_j}}$, if, and only if, $f_{\bar{x}_i \bar{x}_j} \oplus f_{\bar{x}_i x_j} = f_{x_i x_j} \oplus f_{\bar{x}_i x_j} \oplus 1$. When the polarities of x_i and x_j are different, x_i*!E* x_j can be detected in the GRM.

The only difference in the GRM form between *E* symmetry and skew-*E* symmetry is the extra term *1*. This term *1* is a single literal cube x_i or x_j in the function.

Theorem 10: Any two of the conditions x_i *!E* x_j, x_j *!E* x_k, and x_i *NE* x_k implies the third.

5.2.3 Mixed symmetries

Theorem 11: If x_i *!NE* x_j and x_i *!E* x_j both hold, then *f* is a *neutral* function.

This theorem sets up a necessary condition for two variables to hold both *!E* and *!NE* symmetries. Only the variables in a *neutral* function need to be checked for both symmetries.

Theorem 12: Any two of the conditions x_i *!E* x_j, x_j *!NE* x_k, and x_i *E* x_k implies the third.

The following theorem is needed for matching in the application of technology mapping where negation of the output is allowed and has to be managed.

Theorem 13: Any negative symmetry occurs between x_i and x_j in *f*, if, and only if, it occurs in the complement $\bar{f}$.

5.3 Checking all symmetry types among variable pairs

We can verify the positive symmetry on an FDD by checking whether certain two branches are the same; i.e., point to the same sub-FDD. One branch is the one with x_i and without x_j; i.e., $f_{t_i, dc}$, similar to taking cofactor on the ROBDD, where t_i stands for the *pole*-branch of x_i and *dc* stands for the *dc*-branch of x_j. The other is f_{dc, t_j}, where *dc* stands for the *dc*-branch of x_i and t_j stands for the *pole*-branch of x_j. Negative symmetry is checked after we add (XOR) a *1* to any one of the two branches discussed above.

To check for symmetries in the GRM form, we first partition the variables by their signatures and then check symmetries only on certain candidate groups of variables.

The number of GRMs needed for symmetry detection depends on the polarity vectors we choose. For checking *NE*

and skew-*NE* symmetries, the polarities of the two variables need to be the same in the GRM; i.e., both *1* or both *0*. For checking *E* and skew-*E* symmetries, the polarities of the two variables need to be different in the GRM; i.e., *1* and *0*, or *0* and *1*. There are four possible combinations of polarities between any two variables, namely, *00, 01, 10, 11*. We need one combination from *00* and *11* and a second combination from *01* and *10*. It can be shown that any *n* vectors, where the *i*th and *(i+1)*th vectors differ only in the *i*th entry, are sufficient. These *n* polarity vectors contain, between any two variables, three out of four of the desired combinations.

5.4 Linear variables and linear functions

When a variable x_i satisfies $f_{x_i} = \overline{f}_{\bar{x}_i}$, we will obtain $f^B_{x_i} = 1$ and the function can be expressed as $f = x_i \oplus g$ or $f = \bar{x}_i \oplus g$, where the function g is independent of x_i. Variables that satisfy this condition are called *linear* in *f*. In $f = x_2 \oplus x_1\bar{x}_3$, x_2 is a linear variable.

Linear variables are very easy to detect in any GRM form, since the variable can have only one cube of length one in any GRM form. They also have strong properties. First of all, the function that contains linear variable(s) must be *neutral*. Second, linear variables are all *NE* symmetric and *E* symmetric to each other in the function. Hence, once the set of linear variables is detected, it does not have to be checked for any type of symmetry. The third property of the linear variables is that they are prime cubes by definition.

A *linear function* is of the form $f = c_0 \oplus x_1 \oplus \ldots \oplus x_n$, where $c_0 = 0$ or *1*. A linear function is always *neutral* and all dependent variables in a linear function are *balanced*. These two properties make linear functions excellent choices for breaking the balanced variables while we searching for unique GRM forms.

6. Boolean Matching Procedure

This section describes the procedure of how the *npn*-equivalent classes of functions are identified with GRM forms. For Boolean matching, our goal is to (1) differentiate every variable in a function in a unique way, and (2) apply (1) to determine whether two Boolean functions are *npn*-equivalent. If a unique GRM can be derived, then the signatures obtained from the GRM can be used for (1). Variables with identical signatures will be further checked for any symmetry. If all pairs of variables with identical signatures are symmetric in any one of the four symmetry types, no further identification on the variables is needed. Then (2) can be accomplished on the GRMs of the two functions with the variables ordered in the same way.

6.1 Deciding polarities for all variables

As descrobed in section 3.1, we do the following to assure the consistency of polarity selections of variables in the two functions. For functions with $|f| > 2^{n-1}$, we first derive $\bar{f}$, and for *neutral* functions, we need both *f* and $\bar{f}$. The *M*-pole is the choice for each unbalanced variable. If all variables have *M*-poles, then the function can be folded and a unique GRM is obtained. If balanced variables and unbalanced variables exist, then the unbalanced variables will be folded first. The function is now an XOR sum of cubes of mixed polarities. Counting the occurrences of x_i and $\bar{x}_i$ in all cubes can still show the weight unbalance or balance of the remaining variables. Note that the polarity vector obtained with this process is still consistent, as long as the rule is applied to every variable. The function can be folded with respect to the variables whose polarities were decided to obtain a unique GRM. This process is repeated if a function still has balanced variables. Now, either (1) polarities of all the variables have been decided or (2) a set of variables, with nondecreasing size, remain balanced throughout this process.

6.2 Adding linear function

There are cases where all variables are balanced in the function or some variables are balanced as described above. To break the balanced variables, a linear function that contains all and only the balanced variables is added to the function. The new function is used to determine the polarities of the balanced variables.

6.3 Additional GRMs

There are three cases where additional GRMs are needed.

The first case is for the output negation of *neutral* functions. If a unique GRM has been obtained, then a new GRM will be determined with the polarity of every variable reversed. This is based on Theorem 2. The signatures should be generated from both GRMs and they are compared during the Boolean matching process to decide which output phase should be used. Note that all the symmetry conditions remains the same, because of Theorem 7 and Theorem 13.

The second case that requires additional GRMs is for more symmetry checking. This is when there are variables that can not be differentiated or when we need to determine all four symmetry types for all pairs of variables. The maximum number of GRMs is *n*. Note that each new GRM can be incrementally computed from the original GRM.

The last case is when a unique GRM cannot be obtained. The problem is with the balanced variables. If none of the aforementioned methods can break the balance and the remaining balanced variables are not in any symmetric set, then some exhaustive search is needed. Instead of exhaustive permutation among the remaining subsets of variables, we can derive a minimum set of GRMs that can still manage the matching of *npn*-equivalent functions. A set of GRMs can be derived as follows: after the process of finding *M*-pole, a new function g_i is created for each difficult variable x_i by adding a single literal cube to the function *f*. After obtaining the GRM for g_i, adding x_i back to g_i will derive the GRM for *f*. This will be done for both polarities of x_i. In technology mapping, for hard-to-match functions, the set of GRMs and their signatures are computed beforehand. A function can match the library cell if it can match any one of the GRMs.

In the worst case, all variables are balanced throughout the process without any symmetry and we will need *2n* GRMs. *2n* is the upper bound in the number of GRMs needed for Boolean matching of *npn*-equivalent functions. However, this is a pessimistic upper bound since balanced variables are likely to form some type of symmetries among themselves.

7. Experimental Results

To verify the efficiency of our approach, we have tested it on a set of MCNC benchmark cases. The test was run on a DEC5000. Each MCNC benchmark was treated as a set of single output functions and tested separately. Our intention was to differentiate all variables in the functions. For logic verification, this is certainly not necessary, as long as every variable can be differentiated in one of the output functions, it would have sufficient information to order the variables for the entire circuit and the rest can be done on the reordered ROBDDs of the source and target circuits. In the cases we tested, most of the variables are differentiated in few of the output functions. As stated earlier, we do compute functional level signatures for output matching.

The program terminates when all variables are differentiated in the cofactor weight or in a unique GRM with the detection of symmetries. Additional GRMs are generated for symmetry check if some variables are not yet differentiated.

Table 1 lists the MCNC benchmark cases. Column *#I* and *#O* stand for the number of primary input and primary output, respectively. *#h* is the number of output functions that contain nondifferentiable variables. The column *time* is the average time per output function for each benchmark. Note that the vast majority of the output functions have a unique GRM. The rest of the functions with all variables differentiated have up to four GRMs.

For the benchmarks with hard output functions, we have also investigated all variables for the purpose of logic verification. Table 2, column *#hi*, shows the sizes of each subset of variables that are not differentiated in any output function. Multiple subsets of the same size are shown with the number of sets outside the parentheses.

8. Conclusion

In this paper, we proposed a new method for Boolean matching. The GRM forms of Boolean functions are used as tools for matching under input permutation, input negation and output negation. We also incorporate signatures and symmetries for Boolean matching. The signatures obtained from the GRM forms are also used in the symmetry detection. We have shown that all four types between every pair of variables can be found by checking at most *n* GRMs. The total symmetry of functions can be checked with simple arithmetic computation.

With our method, most of the *npn*-equivalent classes only need one GRMs for the purpose of Boolean matching.

Acknowledgement: This work was supported in part by the National Science Foundation under Grant MIP 9117328 and in part by AT&T Bell Laboratories and Digital Equipment Corporation through the California MICRO program.

References

[1] R.E. Bryant, "Graph-based Algorithms for Boolean Functions Manipulation", *IEEE Trans. Computers*, vol. C-35, pp. 677-691, Aug. 1986.

[2] J.B. Burch and D.E. Long, "Efficient Boolean Function Matching", *Proc. Intl. Conference. on Computer Aided Design '92*, pp. 408-411, Nov. 1992.

[3] D. I. Cheng and M. Marek-Sadowska, "Verifying Equivalence of Functions with Unknown Input Correspondence", *Proc. European Design Automation Conference '93*, pp. 81-85, Feb. 1993.

[4] L. Csanky, M. Perkowski and I. Schaefer, "Canonical Restricted Mixed-Polarity Exclusive Sums of Products", *Proc. IEEE International Symposium on Circuits and Systems '92*, pp. 17-20, May 1992.

[5] U. Kebschull and W. Rosenstiel, "Efficient Graph-Based Computation and Manipulation of Functional Decision Diagrams", *Proc. European Design Automation Conf. '93*, pp. 278-282, Feb. 1993.

[6] Y.-T. Lai, S. Sastry and M. Pedram, "Boolean Matching using Binary Decision Diagrams with Applications to Logic Synthesis and Verification", *Proc. Intl. Conf. on Computer Design '92*, pp. 452-458, Oct. 1992.

[7] J. Mohnke and S. Malik, "Permutation and Phase Independent Boolean Comparison", *Proc. European Design Automation Conf. '93*, pp. 86-92, Feb. 1993.

[8] A. Mukhopadhyay, "Detection of Total or Partial Symmetry of a Switching Function with the Use of Decomposition Charts", *IEEE Trans. Elec. Computers*, vol. EC-16, pp. 553-557, Oct. 1963.

[9] H. Savoj, M. J. Silva, R. K. Brayton and A. Sangiovanni-Vincentelli, "Boolean Matching in Logic Synthesis", *Proc. European Design Automation Conf. '92*, pp. 168-174, Feb. 1992.

[10] U. Schlichtmann and F. Brglez, "Efficient Boolean Matching in Technology Mapping with Very Large Cell Library", *Proc. Custom Integrated Circuits Conf. '93*, pp. 3.6.1-3.6.6, May. 1993.

[11] C. Tsai and M. Marek-Sadowska, "Efficient minimization algorithms for fixed polarity AND/XOR canonical networks", *Proc. 3rd Great Lake Symp. VLSI 1993, pp. 76-79.*

[12] C. Tsai and M. Marek-Sadowska, "Detecting Symmetric Variables in Boolean Functions using Generalized Reed-Muller Forms", *accepted to IEEE International Symposium on Circuits and Systems, '94.*

TABLE 1 Results of MCNC benchmark test cases

test case	#I	#O	#h	time	test case	#I	#O	#h	time
5xp1	7	10	0	0.065	f51m	8	8	0	0.081
9sym	9	1	0	0.650	frg1	28	3	0	2.444
C17	5	2	0	0.025	frg2	143	139	36	2.387
alu2	10	6	0	0.325	i1	25	16	0	0.021
alu4	14	8	0	0.498	i6	138	67	0	0.046
apex6	135	99	17	0.193	i8	133	81	0	0.726
apex7	49	37	14	2.136	lal	26	19	0	0.086
b1	3	4	0	0.013	ldd	9	19	0	0.034
b9_n2	41	21	0	0.102	misex2	25	18	1	0.041
bw	5	28	0	0.038	misex3c	14	14	0	0.377
c8	28	18	8	0.091	parity	16	1	0	0.200
cc	21	20	0	0.020	pcle	19	9	0	0.048
cht	47	36	28	0.038	pm1	16	13	0	0.027
cm138a	6	8	0	0.008	rd84	8	4	0	0.471
cm150a	21	1	1	2.200	sao2	10	4	2	0.313
cm151a	12	2	2	0.400	sct	19	15	0	0.076
cm162a	14	5	0	0.083	t481	16	1	0	20.78
cm163a	16	5	0	0.057	tcon	17	16	0	0.011
cmb_n	16	4	0	0.029	term1	34	10	1	2.505
con1	7	2	0	0.034	ttt2	24	21	0	0.140
cordic	23	2	0	36.92	vda	17	39	0	0.318
count	35	16	0	0.159	vg2	25	8	4	100.8
cu	14	11	1	0.055	x1	51	35	0	1.110
des	256	245	84	1.302	x2	10	7	0	0.043
duke2	22	29	4	0.304	x3	135	99	17	0.209
example2	85	66	11	0.088	x4	94	71	0	0.118
f2	4	4	0	0.029	z4ml	7	4	0	0.083

TABLE 2 Sizes of non-differentiable sets of variables

test case	#hi	test case	#hi	test case	#hi	test case	#hi
apex6	(2)	cm150a	(4, 16)	duke2	0	sao2	0
apex7	(6)	cm151a	(3, 8)	example2	(2)x8	term1	(2)
c8	0	cu	(2, 4)	frg2	0	vg2	0
cht	(2)x5	des	0	misex2	0	x3	(2)