Linear Cofactor Relationships in Boolean Functions

Jin S. Zhang, *Member, IEEE*, Malgorzata Chrzanowska-Jeske, *Senior Member, IEEE*, Alan Mishchenko, *Member, IEEE*, and Jerry R. Burch, *Member, IEEE*

***Abstract*—This paper describes linear cofactor relationships (LCRs), which are defined as the exclusive sums of cofactors with respect to a pair of variables in Boolean functions. These relationships subsume classical symmetries and single-variable symmetries. The paper proposes an efficient algorithm to detect LCRs and discusses their potential applications in Boolean matching, minimization of decision diagrams, synthesis of regular layout-friendly logic circuits, and detection of support-reducing bound sets.**

***Index Terms*—Boolean functions, linear cofactor relationship (LCR), logic synthesis, symmetry.**

I. Introduction

Many properties of a Boolean function can be expressed using cofactors. The cofactors are derived from the function by substituting constant values for the input variables. For example, the Boolean difference can be expressed as $f_0 \oplus f_1$, where $f_0 = f[x \leftarrow 0]$ and $f_1 = f[x \leftarrow 1]$ are the negative and positive cofactors of function f with respect to variable x, respectively. The Boolean difference equals 0 if f does not depend on variable x.

A Boolean function has symmetry if the function stays unchanged when several of its input variables are permuted. When the permutation involves two variables, it is called classical symmetry [1]. We use f_{00}, f_{01}, f_{10}, and f_{11} to represent the cofactors of f with respect to two variables, say x_i and x_j. Equation $f(\ldots x_i, \ldots, x_j \ldots) = f(\ldots x_j, \ldots, x_i, \ldots)$ can be expressed as $f_{01} = f_{10}$, or equivalently, as the linear cofactor relationship (LCR) $f_{01} \oplus f_{10} = 0$.

Many applications in electronic design automation exploit symmetries of functions to achieve better results and improve performance. Classical symmetries have a long history and many applications, such as functional decomposition in technology-independent logic synthesis [1]–[4], technology mapping [5]–[7], and binary decision diagram (BDD) minimization [8].

Single-variable symmetries [1] are derived based on similar cofactor relationships involving the other cofactor pairs: $f_{00} \oplus f_{01} = 0/1$, $f_{00} \oplus f_{10} = 0/1$, $f_{01} \oplus f_{11} = 0/1$, and $f_{10} \oplus f_{11} = 0/1$. These symmetries imply that function f remains unchanged or complemented under the constant assignment of one variable while the other variable is complemented, e.g., $f(\ldots, 0, \ldots, x_j, \ldots) = f(\ldots, 0, \ldots, x_j', \ldots)$.

Both classical and single-variable symmetries are called first-order symmetries. The notion of symmetries can be further extended [9] to include symmetries defined as invariants of the functions under the permutation/complementation of arbitrary groups of variables (rather than variable pairs). These symmetries are called higher order symmetries.

One of the benefits of classical and single-variable symmetries is that they result in functionally equivalent cofactors, which lead to merging of nodes in BDDs [10], thereby reducing the BDD size. It is well known that some functions can be more efficiently represented by functional DDs (FDDs) [11], [12] and Kronecker FDDs (KFDDs) [13]. We observed other relationships in Boolean functions, in addition to classical and single-variable symmetries, which result in merging of nodes in FDDs and KFDDs. As a result of these relationships, we can reduce FDDs and KFDDs even more. This was our initial motivation for extending the notion of symmetries to these relationships.

In this paper, we study linear (EXOR-based) relationships among any nonempty subset of the four two-variable cofactors of a Boolean function. These LCRs are sufficient conditions for the minimization of all of the abovementioned DDs. They also play an important role in representing a function's characteristics, which can be used in many optimization and synthesis steps during digital circuit design. We discuss four of the potential applications: Boolean matching, minimization of DDs, synthesis of regular layout-friendly logic circuits, and detection of support-reducing bound sets.

We show that LCRs can be computed as efficiently as classical symmetries. However, they are more diverse and, therefore, give a more detailed characterization of Boolean functions. This is demonstrated, for example, in Section V-A, by the comparison of classical symmetries and LCRs when applied to Boolean matching. Higher order symmetries are an orthogonal generalization of classical symmetries in relation to LCRs because neither of them subsumes the other. However, the computation of higher order symmetries is substantially harder because it requires manipulation of the matrices of cofactors of the Boolean function, which is more complex than the BDD traversal procedures proposed in this paper for the computation of LCRs.

The remainder of this paper is organized as follows. Background information on Boolean functions, DDs, and classical and single-variable symmetries are introduced in Section II. Section III presents the motivation and definitions of LCRs. An

Manuscript received June 25, 2004; revised February 21, 2005. This work was supported in part by the National Science Foundation (NSF), Grant CCR-9988402. This paper was recommended by Associate Editor S. Nowick.

J. S. Zhang and M. Chrzanowska-Jeske are with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207 USA (e-mail: jinsong@ece.pdx.edu; jeske@ece.pdx.edu).

A. Mishchenko is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: alanmi@eecs.berkeley.edu).

J. R. Burch is with the Synopsys, Inc., Hillsboro, OR 97124 USA (e-mail: jrb@synopsys.com).

Digital Object Identifier 10.1109/TCAD.2005.855951

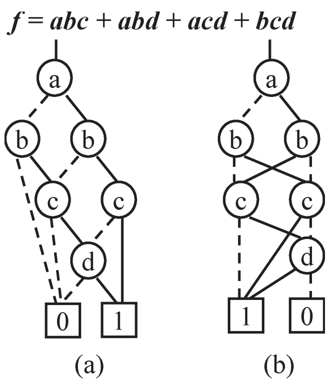


Fig. 1. (a) ROBDD, (b) FDD with nD expansions.

efficient algorithm to detect LCRs is described in Section IV. Experimental results on Microelectronics Center of North Carolina (MCNC) benchmarks demonstrate a performance improvement of this algorithm over the naïve method. Section V discusses potential applications of LCRs in Boolean matching, pseudo-KFDD (PKFDD) minimization, regular layout synthesis [14], and detecting support-reducing bound sets [15]. Conclusions are drawn in Section VI.

II. Background

A. Boolean Functions and DDs

We use $f(x_1, \ldots, x_n)$ to denote a Boolean function with input variables $x_1, \ldots, x_n$. In this paper, we only consider completely specified Boolean functions and Boolean variables.

The variables that a function f depends on are called the support of f, denoted by supp(f).

We write $f[x \leftarrow 0]$ to denote the function formed from f by replacing x with 0. This is also known as the negative cofactor of f with respect to variable x, denoted by $f_{x'}$. While x is not in the support of the resulting function, it is often convenient to treat x as an input. The positive cofactor f_x is $f[x \leftarrow 1]$.

We denote multiple substitutions similarly. For example, $f[x_i \leftarrow 0,\ x_j \leftarrow 0]$ is a cofactor of a function $f(x_1, \ldots, x_n)$ with respect to variables x_i and x_j. If x_i and x_j are clear from context, then this cofactor can be denoted by f_{00}.

The Shannon expansion of a Boolean function f is $f = x \bullet f_x + x' \bullet f_{x'}$, where "$\bullet$" denotes conjunction and "$+$" denotes disjunction. If no confusion results, we may leave the conjunction operator implicit.

The positive Davio expansion [16] (denoted Davio I or pD) for a Boolean function f is $f = f_{x'} \oplus (x \bullet (f_x \oplus f_{x'}))$, where "$\oplus$" denotes the EXOR operation. $(f_x \oplus f_{x'})$ is the Boolean difference of f with respect to variable x. We refer to $f_{x'}$ as the constant moment and $(f_x \oplus f_{x'})$ as the linear moment. The negative Davio expansion (denoted Davio II or nD) is $f = f_x \oplus (x' \bullet (f_x \oplus f_{x'}))$. Here, f_x is the constant moment and $(f_x \oplus f_{x'})$ is the linear moment.

A BDD for a Boolean function f is generated by successively applying Shannon expansions to all variables. A reduced-ordered BDD (ROBDD) is a BDD with the constraint that input variables appear in the same order in every path from root to leaves, and each node represents a distinct function. Fig. 1(a)

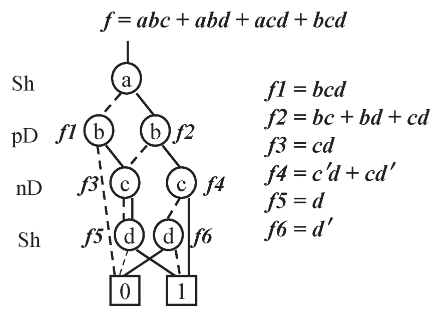


Fig. 2. KFDD for f.

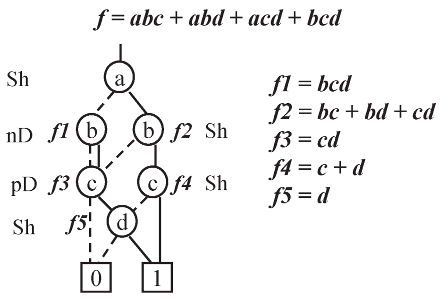


Fig. 3. PKFDD for f.

shows the ROBDD for function $f = abc + abd + acd + bcd$. The dashed lines indicate the negative cofactor of each node.

A functional DD [11], [12] is generated by successively applying either positive or negative Davio expansions to all the variables, with one type of expansion per variable. Fig. 1(b) is the FDD for f with all negative Davio expansions. The dashed lines represent the constant moments for each node.

A KFDD [13] allows both Shannon and Davio expansions in generating the DD, although only one type of expansion is allowed per variable. Fig. 2 shows KFDD for the same function f and the associated functions at each node.

A PKFDD [17] removes the constraint that only one type of expansion is allowed per variable for KFDD, thus providing more flexibility in representing a Boolean function. Fig. 3 is the PKFDD for function f with the expansion type and the associated function at each node.

B. Classical and Single-Variable Symmetry

The most basic notion of symmetry states that two variables x_i and x_j of function $f(\ldots, x_i, \ldots, x_j, \ldots)$ are symmetric if the function remains invariant when the variables are swapped, i.e., $f(\ldots, x_i, \ldots, x_j, \ldots) = f(\ldots, x_j, \ldots, x_i, \ldots)$.

For a pair of variables x_i and x_j, there are four cofactors: f_{00}, f_{01}, f_{10}, and f_{11}. The above notion of symmetry can be expressed as $f_{01} = f_{10}$, or equivalently, $f_{01} \oplus f_{10} = 0$. This symmetry is also called the nonskew nonequivalent symmetry, denoted by $x_i \mathrm{NE} x_j$. "Nonskew" refers to the right side of the equation being 0 rather than 1. "Nonequivalent" in this context means that, in each cofactor, the values of the two variables are not equivalent. Nonskew equivalent symmetry, denoted by $x_i \mathrm{E} x_j$, exists when $f_{00} \oplus f_{11} = 0$. This is a generalization where if the two variables are swapped, they must also be negated for the function to be preserved. Skew symmetry

TABLE I
CLASSICAL AND SINGLE-VARIABLE SYMMETRIES

Invariant of the function (nonskew/skew)	Cofactor relationship	Name
$f(\ldots,x_i,\ldots,x_j,\ldots) \oplus f(\ldots,x_j,\ldots,x_i,\ldots) = 0/1$	$f_{10} \oplus f_{01} = 0/1$	Classical Symmetry
$f(\ldots,x_i,\ldots,x_j,\ldots) \oplus f(\ldots,x_j',\ldots,x_i',\ldots) = 0/1$	$f_{00} \oplus f_{11} = 0/1$	
$f(\ldots,0,\ldots,x_j,\ldots) \oplus f(\ldots,0,\ldots,x_j',\ldots) = 0/1$	$f_{00} \oplus f_{01} = 0/1$	Single-Variable Symmetry
$f(\ldots,1,\ldots,x_j,\ldots) \oplus f(\ldots,1,\ldots,x_j',\ldots) = 0/1$	$f_{10} \oplus f_{11} = 0/1$	
$f(\ldots,x_i,\ldots,0,\ldots) \oplus f(\ldots,x_i',\ldots,0,\ldots) = 0/1$	$f_{00} \oplus f_{10} = 0/1$	
$f(\ldots,x_i,\ldots,1,\ldots) \oplus f(\ldots,x_i',\ldots,1,\ldots) = 0/1$	$f_{01} \oplus f_{11} = 0/1$	

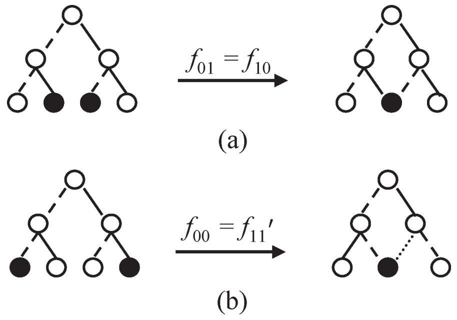


Fig. 4. Effect of symmetries on ROBDD. (a) Nonskew nonequivalent symmetry; (b) skew equivalent symmetry.

exists when two cofactors are complemented rather than equivalent to each other. For example, a Boolean function with skew equivalent symmetry is such that $f_{00} \oplus f_{11} = 1$. Skew symmetries are analogous to nonskew symmetries, except that the function is complemented, rather than preserved, when the variables are swapped. Taken together, all of the above symmetries are called classical symmetries [1].

For any two variables in a Boolean function, there are six possible cofactor pairs. Single-variable symmetries [1] are defined when the function has equivalent (nonskew symmetry) or complement (skew symmetry) relationships of the other four cofactor pairs not included in classical symmetries, e.g., $f_{00} = f_{01}$. This nonskew relationship means that the negative cofactor of f with respect to variable x_i does not depend on x_j. Table I summarizes classical and single-variable symmetries, their functional invariance, and cofactor relationships.

III. LCRs IN BOOLEAN FUNCTIONS

A. Motivations

The existence of both classical symmetries and single-variable symmetries in a Boolean function results in shared or constant nodes in the corresponding ROBDD, as illustrated in Fig. 4. The complemented edges (represented by dotted lines) are used in the DD in the case of skew symmetries. This observation allows us to reduce the size of the ROBDD.

Boolean functions can be represented by FDDs using positive or negative Davio expansions. Fig. 5 shows a fragment of an FDD with positive Davio expansions for both variables x_i and x_j. The functions at each node $f1$, $f2$, $f3$, and $f4$ are given in terms of the cofactors for variables x_i and x_j.

If variables x_i and x_j are symmetric with nonskew nonequivalent symmetry, then $f_{01} = f_{10}$. This relationship implies

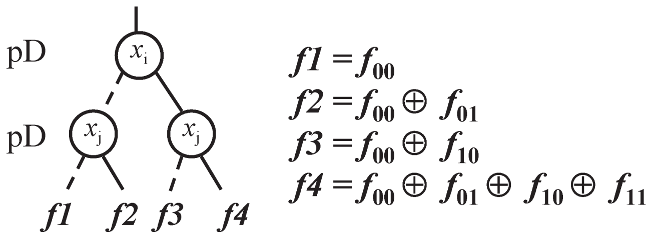

pD
pD
x_i
x_j
x_j
$f1$ $f2$ $f3$ $f4$
$f1 = f_{00}$
$f2 = f_{00} \oplus f_{01}$
$f3 = f_{00} \oplus f_{10}$
$f4 = f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11}$

Fig. 5. Partial FDD with positive Davio expansions.

TABLE II
COFACTOR RELATIONSHIPS FOR ONE OR THREE COFACTORS

Cofactor relationships	Expansion Combinations	Equivalent cofactors
$f_{00} = 0$	pDnD	$f1 = f2$
$f_{01} = 0$	pDpD	$f1 = f2$
$f_{10} = 0$	pDpD	$f1 = f3$
$f_{11} = 0$	pDnD	$f2 = f4$
$f_{01} \oplus f_{10} \oplus f_{11} = 0$	pDpD	$f1 = f4$
$f_{00} \oplus f_{10} \oplus f_{11} = 0$	pDnD	$f2 = f3$
$f_{00} \oplus f_{01} \oplus f_{11} = 0$	nDpD	$f2 = f3$
$f_{00} \oplus f_{10} \oplus f_{01} = 0$	nDnD	$f1 = f4$

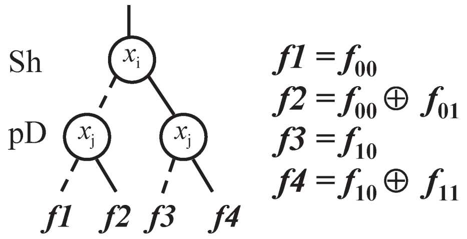


Fig. 6. Partial KFDD with Shannon and positive Davio expansions.

that $f2 = f3$, so nodes $f2$ and $f3$ can be shared in Fig. 5. Both classical nonskew nonequivalent and nonskew equivalent symmetries create shared nodes in the FDD.

Other node-sharing possibilities lead to an additional set of cofactor relationships. For example, $f1 = f4$ if and only if (iff) $f_{00} = f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11}$, or equivalently, $f_{01} \oplus f_{10} \oplus f_{11} = 0$. This cofactor relationship involves three cofactors. Similarly, $f1 = f2$ iff $f_{00} = f_{00} \oplus f_{01}$, or $f_{01} = 0$. This is a constant cofactor relationship, which also reduces FDDs.

Since an FDD can be constructed with either positive or negative Davio expansions, there are four combinations of expansion types for two variables: pDpD, nDnD, pDnD, and nDpD. Performing the same analysis above, we get the following eight cofactor relationships involving one or three cofactors [18], [19]. Table II shows these relationships, the expansion combinations, and equivalent cofactors that lead to these relationships (there could be multiple expansions that lead to the same relationship, only one is given in Table II).

KFDDs allow Shannon, positive Davio, and negative Davio expansions to be used. Fig. 6 shows the partial KFDD for variables x_i and x_j using Shannon and positive Davio expansions. The functions at each node $f1$, $f2$, $f3$, and $f4$ are given in terms of the cofactors for x_i and x_j.

It is easy to see that the cofactor relationships listed in Table II also reduce KFDDs. One condition that is missing

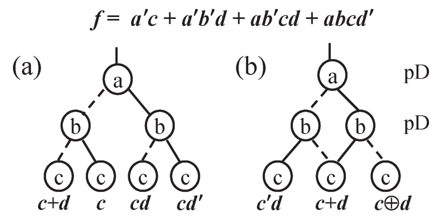


Fig. 7. (a) Partial BDD, (b) partial FDD of function f.

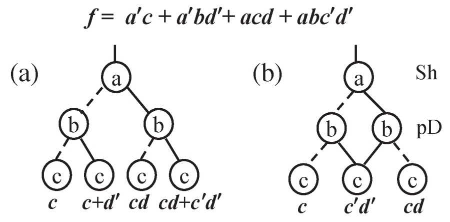


Fig. 8. (a) Partial BDD, (b) partial KFDD of function f.

is when $f2 = f4$, or equivalently, $f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11} = 0$. This is a cofactor relationship that involves all four cofactors [18], [19]. The other combinations of Shannon, as well as the positive and negative Davio expansions, lead to the same relationships.

Figs. 7 and 8 show how FDDs and KFDDs can be reduced as a result of these cofactor relationships. In Fig. 7, variables a and b in $f = a'c + a'b'd + ab'cd + abcd'$ do not have symmetries, therefore there is no node sharing in the BDD for variables a and b. However, there is node sharing in the FDD because $f_{01} \oplus f_{10} \oplus f_{11} = 0$. Using this relationship in f between variables a and b, the FDD for f can be constructed with a specific variable order and expansion types to take advantage of this node reduction. Similarly, Fig. 8 shows that variables a and b are not symmetric in function $f = a'c + a'bd' + acd + abc'd'$, therefore there is no node sharing in the BDD. However, $f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11} = 0$. This results in a shared node in the KFDD if variables a and b are adjacent in the variable order, and Shannon and positive Davio expansions are used.

B. LCRs

Definition 1: Let $f(x_1, \ldots, x_n)$ be a Boolean function. Let i and j be integers, $1 \le i \le n$, $1 \le j \le n$. Let $g = \langle g_4, g_3, g_2, g_1, g_0 \rangle$ be a Binary vector of length 5, such that at least one of g_3, g_2, g_1, g_0 is 1. Variables x_i and x_j have LCR g in function f iff: $\forall x_1, \ldots, x_n [g_4 = g_3 f_{11} \oplus g_2 f_{10} \oplus g_1 f_{01} \oplus g_0 f_{00}]$, where f_{11}, f_{10}, f_{01}, and f_{00} are cofactors of f with respect to variables x_i and x_j. We use the symbol $\mathrm{LCR}_g(f, (x_i, x_j))$ to denote LCRs. If f and (x_i, x_j) are clear from context, we write LCR_g.

For example, if variables x_i and x_j have nonskew non-equivalent classical symmetry in function f, then $0 = 0 \bullet f_{11} \oplus 1 \bullet f_{10} \oplus 1 \bullet f_{01} \oplus 0 \bullet f_{00}$. Therefore, $\mathrm{LCR}_{\langle 0,0,1,1,0 \rangle}$, or, in hexadecimal form, LCR_{06} holds. All classical and single-variable symmetries are subsumed by LCRs.

We use a don't care "$-$" in g to represent multiple LCRs. For example, $\mathrm{LCR}_{\langle 0,0,0,1,- \rangle}$ represents both $\mathrm{LCR}_{\langle 0,0,0,1,0 \rangle}$ and $\mathrm{LCR}_{\langle 0,0,0,1,1 \rangle}$.

TABLE III
SUMMARY OF LCRs

Index	Linear Cofactor Relationships (nonskew / skew)	Linear Cofactor Class	Symbol (nonskew/ skew)
1	$f_{00} = 0/1$	LCC$_1$	LCR$_{01}$/LCR$_{11}$
2	$f_{01} = 0/1$		LCR$_{02}$/LCR$_{12}$
3	$f_{10} = 0/1$		LCR$_{04}$/LCR$_{14}$
4	$f_{11} = 0/1$		LCR$_{08}$/LCR$_{18}$
5	$f_{01} \oplus f_{10} = 0/1$	LCC$_2$	LCR$_{06}$/LCR$_{16}$
6	$f_{00} \oplus f_{11} = 0/1$		LCR$_{09}$/LCR$_{19}$
7	$f_{00} \oplus f_{01} = 0/1$		LCR$_{03}$/LCR$_{13}$
8	$f_{10} \oplus f_{11} = 0/1$		LCR$_{0C}$/LCR$_{1C}$
9	$f_{00} \oplus f_{10} = 0/1$		LCR$_{05}$/ LCR$_{15}$
10	$f_{01} \oplus f_{11} = 0/1$		LCR$_{0A}$/ LCR$_{1A}$
11	$f_{00} \oplus f_{01} \oplus f_{10} = 0/1$	LCC$_3$	LCR$_{07}$/ LCR$_{17}$
12	$f_{00} \oplus f_{01} \oplus f_{11} = 0/1$		LCR$_{0B}$/ LCR$_{1B}$
13	$f_{00} \oplus f_{10} \oplus f_{11} = 0/1$		LCR$_{0D}$/ LCR$_{1D}$
14	$f_{01} \oplus f_{10} \oplus f_{11} = 0/1$		LCR$_{0E}$/ LCR$_{1E}$
15	$f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11} = 0/1$	LCC$_4$	LCR$_{0F}$/ LCR$_{1F}$

TABLE IV
STATISTICS OF LCRs IN MCNC BENCHMARK FUNCTIONS

Class	Skew Type	Total LCR	Ratio
LCC$_1$	Nonskew	30,610	8.1%
	Skew	11,005	2.9%
LCC$_2$	Nonskew	114,305	30.3%
	Skew	8,652	2.3%
LCC$_3$	Nonskew	85,927	22.8%
	Skew	2,063	0.6%
LCC$_4$	Nonskew	124,681	33.0%
	Skew	540	0.1%

Definition 2: A linear cofactor class (LCC) contains all the LCRs that have the same sum of g_0, g_1, g_2, g_3. We use LCC$_n$ to denote the linear cofactor class, where $n = g_0 + g_1 + g_2 + g_3$.

For example, both $\mathrm{LCR}_{\langle 0,0,1,1,0 \rangle}$ and $\mathrm{LCR}_{\langle 1,1,0,0,1 \rangle}$ belong to LCC$_2$, even though they have different skew type. $\mathrm{LCR}_{\langle 0,1,0,0,0 \rangle}$ and $\mathrm{LCR}_{\langle 0,1,1,1,0 \rangle}$ belong to LCC$_1$ and LCC$_3$, respectively.

Table III summarizes all the cofactor relationships discussed so far. We adopt the skew terminology when the EXOR of the cofactors is 1.

C. Statistics for LCRs in MCNC Benchmarks

Classical and single-variable symmetries are common in Boolean functions. It is interesting to see how common the other LCRs are to decide if it is worthwhile to detect these relationships in Boolean functions.

Table IV gives the total number and ratio of nonskew and skew LCRs in each of the linear cofactor classes, for all MCNC multilevel combinational benchmark functions. The complete table can be found in [45], which shows that LCRs exist in all MCNC benchmarks. More than 90% of the LCRs are nonskew and among all LCRs, 32.6% are classical and single-variable symmetries.

Existing electronic-design-automation algorithms often make use of classical symmetries. It is possible that other

LCRs can be used to improve these algorithms. We will touch upon some potential applications in Section V.

IV. Fast Computation of LCRs

While many algorithms have been proposed to detect classical symmetries efficiently [20]–[23], the only method to detect LCRs had been the naïve method, which computes the four cofactors for each variable pair in the Boolean function and checks if the cofactors satisfy the relationships defined in Table III. This method is straightforward, yet very inefficient for large circuits. In [21], an efficient method to compute classical symmetries was proposed, based on a theorem in [23]. Theorem 1 extends the scope of [23] to all LCRs. This allowed us to develop, based on the method proposed in [21], a fast algorithm for the computation of all LCRs.

Theorem 1: Let $f(x_1, \ldots, x_n)$ be a Boolean function and let the variables x_i, x_j, and x_k be distinct variables in the support of f. x_i and x_j have LCR$_g$ in f iff they have LCR$_g$ in both cofactors of f with respect to x_k.

Proof: x_i and x_j have LCR$_g$ in function f iff: $\forall x_1, \ldots, x_n[g_4 = g_3 f_{11} \oplus g_2 f_{10} \oplus g_1 f_{01} \oplus g_0 f_{00}]$. This equation holds true for all values of x_k, where $1 \leq k \leq n$, $k \neq i$ and $k \neq j$. That is

$$\forall x_1, \ldots, x_{k-1} x_{k+1}, \ldots, x_n[g_4 = g_3 f_{11} \oplus g_2 f_{10} \oplus g_1 f_{01} \oplus g_0 f_{00}][x_k \leftarrow 0]$$

and

$$\forall x_1, \ldots, x_{k-1} x_{k+1}, \ldots, x_n[g_4 = g_3 f_{11} \oplus g_2 f_{10} \oplus g_1 f_{01} \oplus g_0 f_{00}][x_k \leftarrow 1].$$

Taking $[x_k \leftarrow 0]$ and $[x_k \leftarrow 1]$ into the equations, we have

$$\forall x_1, \ldots, x_n[g_4 = g_3 f_{11}[x_k \leftarrow 0] \oplus g_2 f_{10}[x_k \leftarrow 0] \oplus g_1 f_{01}[x_k \leftarrow 0] \oplus g_0 f_{00}[x_k \leftarrow 0]]$$

and

$$\forall x_1, \ldots, x_n[g_4 = g_3 f_{11}[x_k \leftarrow 1] \oplus g_2 f_{10}[x_k \leftarrow 1] \oplus g_1 f_{01}[x_k \leftarrow 1] \oplus g_0 f_{00}[x_k \leftarrow 1]].$$

Therefore, x_i and x_j have the same LCR$_g$ in both cofactors of f with respect to x_k. The reverse implication holds because x_k must be 0 or 1. ■

A. Computational Core

Theorem 1 states that we can compute the LCRs of a function if we know the LCRs of the function's cofactors with respect to a variable. Therefore, we can recursively solve the subproblems and derive the final solution from the partial solutions. There are two recursive procedures in the proposed algorithm: ComputeLCR and LCRVars. ComputeLCR detects variable pairs with LCRs in f_x and $f_{x'}$, where x is the top variable in the BDD. LCRVars detect LCRs between variable x and the other variables in the support of f.

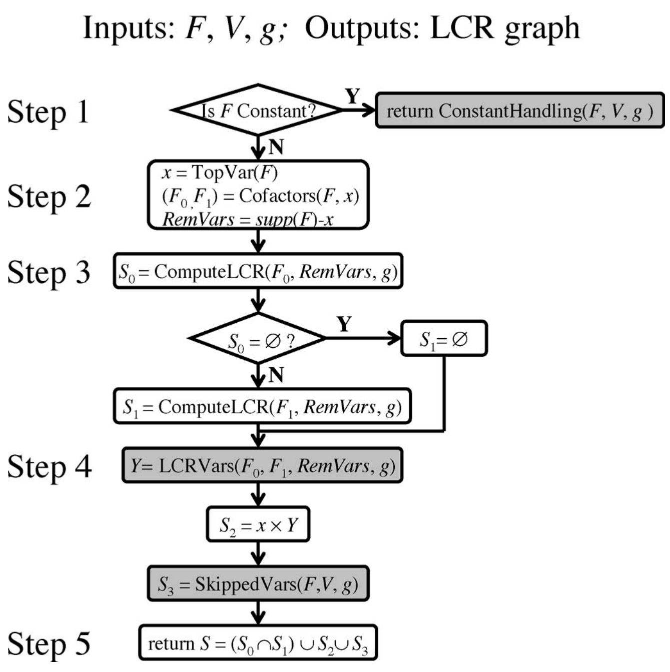

Inputs: F, V, g; Outputs: LCR graph
Step 1
Is F Constant?
Y
return ConstantHandling(F, V, g)
N
Step 2
x = TopVar(F)
(F0, F1) = Cofactors(F, x)
RemVars = supp(F)-x
Step 3
S0 = ComputeLCR(F0, RemVars, g)
S0 = ∅ ?
Y
S1 = ∅
N
S1 = ComputeLCR(F1, RemVars, g)
Step 4
Y= LCRVars(F0, F1, RemVars, g)
S2 = x × Y
S3 = SkippedVars(F,V, g)
Step 5
return S = (S0∩S1) ∪ S2∪ S3

Fig. 9. Flowchart of the top-level recursion: ComputeLCR.

We use an LCR graph to represent the computed LCRs. The vertices of the LCR graph correspond to variables in the support of the function. The edges connect variable pairs with an LCR. The graph operations union ($\cup$) and intersection ($\cap$) are defined as the set union and intersection on the sets of edges. The Cartesian product ($\times$) of variable x by a set of variables Y results in a graph composed of edges connecting the vertex of variable x with the vertices of variables in Y. In the software implementation, the LCR graph is represented by zero-suppressed binary DDs (ZDD) [24], which give a canonical representation of the LCR information.

1) Top Level Recursion—ComputeLCR: Fig. 9 shows the flowchart of ComputeLCR. It takes as inputs: function F, a set of variables V, and integer g. F is the function whose LCRs are being computed. V is initialized to be the support of F at the top level, but could become a super set of F's support in the subsequent recursive calls. g is an integer indicating the LCR to be computed, as given in Table III. The program returns the LCR graph representing variable pairs with the particular LCR. It is worth noting that only variables that function F depends on participate in this computation. In the case of multioutput functions, the LCRs for each output are computed separately using the true support of the output function.

The recursive steps are the same for all type of LCRs. However, the handling of Step 1 and the S_3 computation in Step 4 is different (indicated by shaded boxes in the flowchart) due to the difference in the cofactor relationships. The following paragraphs explain each step in the flowchart. A brief discussion of classical symmetries is included here for completeness; the details can be found in [21].

Step 1) If F is a constant, then $F = F_{11} = F_{10} = F_{01} = F_{00}$. Checking if any variable pairs in V have LCRs is equivalent to checking if g satisfies the following equation: $g_4 = (g_3 \oplus g_2 \oplus g_1 \oplus g_0) \bullet F$.

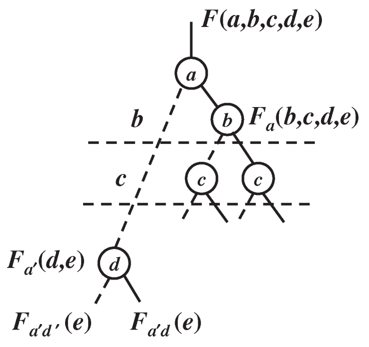


Fig. 10. Illustrations of the skipped variables.

If F is constant 0, then each pair of variables in V has all 15 types of nonskew LCRs.

If F is constant 1, then each pair of variables in V has nonskew LCC_2, nonskew LCC_4, skew LCC_1, and skew LCC_3 LCRs.

When F is a constant, the program either returns a complete LCR graph, or an empty set, depending on the value of the function and the LCR to be computed.

Step 2) Compute the support of F and the cofactors of F with respect to the top variable x of the BDD. This results in two different Boolean functions F_0 and F_1.

Step 3) The recursive calls are performed in this step. First, ComputeLCR is called with F_0 and $\mathrm{supp}(F) - x$. Next, the same procedure is repeated for the positive cofactor. If there is no LCR in the negative cofactor F_0, there is no need for the second recursive call, according to Theorem 1. As a result, the computation is very efficient in this case.

Step 4) The first part of the solution comes from the intersection of partial solutions S_0 and S_1, because, according to Theorem 1, only the LCRs of both cofactors are LCRs of the function.

The set Y contains those variables y such that variables x (chosen in Step 2) and y have LCR_g. This set is computed using another recursive procedure LCRVars, which will be discussed in Section IV-B-2.

The set S_3 contains LCRs involving variables that are in V but not in the support of F. This is relevant when a cofactor of the function does not depend on all the variables in the initial support of F less x. Fig. 10 is a fragment of the BDD illustrating this scenario. The negative cofactor of F with respect to variable a only depends on variables d and e. Therefore, in the recursive procedure, variables b and c are skipped over. As a result, we need to add the LCRs involving the "skipped" variables. It is easy to see that between the skipped variables b and c, we have $F_{a'b'c'} = F_{a'b'c} = F_{a'bc'} = F_{a'bc}$, and between the skipped variable b and the variable d in the support of $F_{a'}$, we have $F_{a'b'd'} = F_{a'bd'}$ and $F_{a'b'd} = F_{a'bd}$.

a) LCRs among the skipped variables. In this case: $F_{11} = F_{10} = F_{01} = F_{00}$. Checking if any skipped variable pairs have LCRs is equivalent to checking if g satisfies the following equation: $g_4 = (g_3 \oplus g_2 \oplus g_1 \oplus g_0) \bullet F_{00}$. All nonskew LCC_2 and nonskew LCC_4 LCRs satisfy this equation.

b) LCRs between the skipped variables and the variables in the support of F. In this case: $F_{10} = F_{00}$ and $F_{11} = F_{01}$, where the cofactors are with respect to one skipped variable and one variable in the support of F. Checking if they have LCRs is equivalent to checking if g satisfies the following equation: $g_4 = (g_3 \oplus g_1) \bullet F_{01} \oplus (g_2 \oplus g_0) \bullet F_{00}$. Different LCRs exist depending on the values and the relationships of F_{00} and F_{01}. For example, if F_{01} is complement to F_{00}, then $g_4 = (g_3 \oplus g_1) \oplus (g_3 \oplus g_2 \oplus g_1 \oplus g_0) \bullet F_{00}$. $\langle 1,1,1,0,0\rangle$ and $\langle 0,1,0,1,0\rangle$ are two of the LCRs that satisfy this equation. Table V gives the complete set of LCRs between the skipped variables and the variables in the support of F.

TABLE V
LCRs BETWEEN SKIPPED VARIABLES AND THE VARIABLES IN THE SUPPORT OF F IN COMPUTELCR

Case	Corresponding LCRs
$F_{00} = 0$	All LCR_g, where $g = \langle 0, 0, \text{-}, 0, \text{-}\rangle$ and $\langle 0, 1, \text{-}, 1, \text{-}\rangle$.
$F_{00} = 1$	All LCR_g, where $g = \langle 0, 0, 1, 0, 1\rangle, \langle 0, 1, 0, 1, 0\rangle, \langle 0, 1, 1, 1, 1\rangle, \langle 1, 0, 0, 0, 1\rangle, \langle 1, 0, 1, 0, 0\rangle, \langle 1, 1, 0, 1, 1\rangle, \langle 1, 1, 1, 1, 0\rangle$.
$F_{01} = 0$	All LCR_g, where $g = \langle 0, \text{-}, 0, \text{-}, 0\rangle$ and $\langle 0, \text{-}, 1, \text{-}, 1\rangle$.
$F_{01} = 1$	All LCR_g, where $g = \langle 0, 1, 0, 1, 0\rangle, \langle 0, 0, 1, 0, 1\rangle, \langle 0, 1, 1, 1, 1\rangle, \langle 1, 0, 0, 1, 0\rangle, \langle 1, 1, 0, 0, 0\rangle, \langle 1, 0, 1, 1, 1\rangle, \langle 1, 1, 1, 0, 1\rangle$.
$F_{01} = F_{00}$	All nonskew LCC_2 and nonskew LCC_4 LCRs.
$F_{01} = F_{00}'$	All LCR_g, where $g = \langle 0, 0, 1, 0, 1\rangle, \langle 0, 1, 0, 1, 0\rangle, \langle 0, 1, 1, 1, 1\rangle, \langle 1, 1, 0, 0, 1\rangle, \langle 1, 1, 1, 0, 0\rangle, \langle 1, 0, 1, 1, 0\rangle, \langle 1, 0, 0, 1, 1\rangle$.

Step 5) The final result returned by ComputeLCR is $S = (S_0 \cap S_1) \cup S_2 \cup S_3$.

2) Inner Recursion—LCRVars: Procedure LCRVars takes as inputs: F_0 and F_1, the two cofactors of F with respect to variable x, the set of candidate variables Y, and integer g. It returns the subset of Y such that x has LCRs with the variables in this subset. Fig. 11 is the flowchart for this procedure (hereafter, functions G and H are used to represent the cofactors F_0 and F_1).

For each LCR, the program differs in Step 1, and R_2 computation in Steps 4 and 5, as indicated in the shaded boxes in the flowchart.

Step 1) Three scenarios are handled in this case: 1) G and H are both constant functions. 2) H is a constant function, but G is not. 3) G is a constant function, but H is not.

a) G and H are both constant functions. In this case, $G = F_{01} = F_{00}$ and $H = F_{11} = F_{10}$. Checking if variable x has any LCRs with the variables in Y is equivalent to checking if g satisfies the following equation:

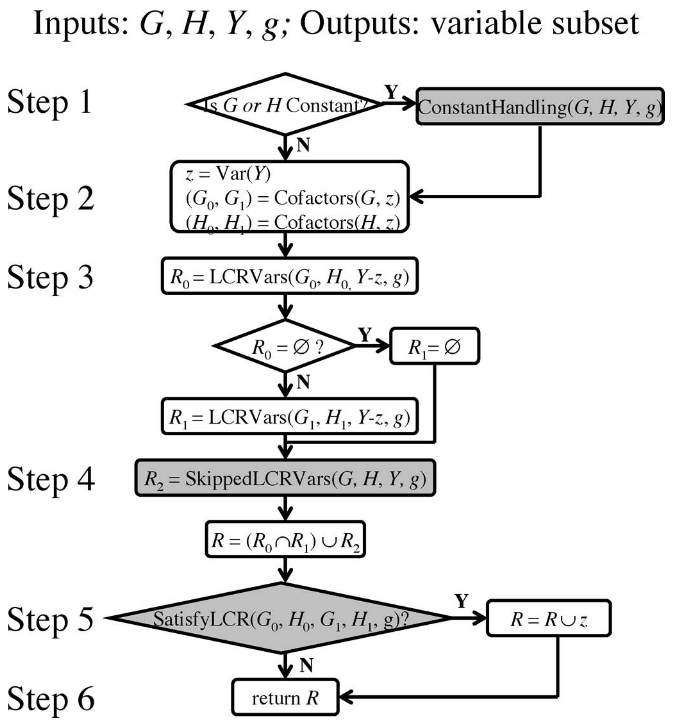

Inputs: G, H, Y, g; Outputs: variable subset
Step 1
Is G or H Constant?
Y
ConstantHandling(G, H, Y, g)
N
Step 2
z = Var(Y)
(G0, G1) = Cofactors(G, z)
(H0, H1) = Cofactors(H, z)
Step 3
R0 = LCRVars(G0, H0, Y-z, g)
R0 = ∅ ?
Y
R1 = ∅
N
R1 = LCRVars(G1, H1, Y-z, g)
Step 4
R2 = SkippedLCRVars(G, H, Y, g)
R = (R0 ∩ R1) ∪ R2
Step 5
SatisfyLCR(G0, H0, G1, H1, g)?
Y
R = R ∪ z
N
Step 6
return R

Fig. 11. Flowchart of the inner recursion: LCRVars.

TABLE VI
LCRs WHEN H AND G ARE CONSTANT

H/G	Corresponding LCRs
0/0	All LCR_g: where $g = \langle 0, \text{-}, \text{-}, \text{-}, \text{-} \rangle$
0/1	All LCR_g, where $g = \langle 0, \text{-}, \text{-}, 0, 0 \rangle, \langle 0, \text{-}, \text{-}, 1, 1 \rangle, \langle 1, \text{-}, \text{-}, 0, 1 \rangle, \langle 1, \text{-}, \text{-}, 1, 0 \rangle$
1/0	All LCR_g, where $g = \langle 0, 0, 0, \text{-}, \text{-} \rangle, \langle 0, 1, 1, \text{-}, \text{-} \rangle, \langle 1, 0, 1, \text{-}, \text{-} \rangle, \langle 1, 1, 0, \text{-}, \text{-} \rangle$
1/1	All skew LCC_1 and LCC_3 LCRs, nonskew LCC_2 and LCC_4 LCRs

$g_4 = (g_3 \oplus g_2) \bullet H \oplus (g_1 \oplus g_0) \bullet G$. The existence of LCRs depends on the values of G and H. For example, if $H = 1$ and $G = 0$, we have $g_4 = (g_3 \oplus g_2)$. The following g satisfy this equation: $\langle 0, 0, 0, -, - \rangle$, $\langle 0, 1, 1, -, - \rangle$, $\langle 1, 0, 1, -, - \rangle$, and $\langle 1, 1, 0, -, - \rangle$. Therefore, there are 16 different LCRs under this value combination. Table VI gives all four combinations of H and G with the corresponding LCRs.

b) H is a constant function, but G is not. If H is constant 0, we have $g_4 = (g_3 \oplus g_2) \bullet 0 \oplus (g_1 \bullet F_{01} \oplus g_0 \bullet F_{00})$. There are three values of g that satisfy this equation for arbitrary F_{01} and F_{00} : $\langle 0, 0, 1, 0, 0 \rangle$, $\langle 0, 1, 0, 0, 0 \rangle$, and $\langle 0, 1, 1, 0, 0 \rangle$. If F_{01} is constant 0, then there are three additional LCRs: $\langle 0, 0, 1, 1, 0 \rangle$, $\langle 0, 1, 0, 1, 0 \rangle$, and $\langle 0, 1, 1, 1, 0 \rangle$. If F_{01} is constant 1, then the additional LCRs are $\langle 1, 0, 1, 1, 0 \rangle$, $\langle 1, 1, 0, 1, 0 \rangle$, and $\langle 1, 1, 1, 1, 0 \rangle$. The cases where F_{00} is constant or H is constant 1 can be derived similarly.

c) G is a constant function, but H is not. The LCRs in this case are analogous to the above cases.

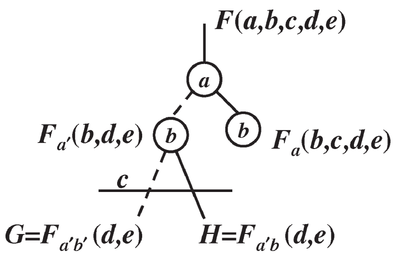


Fig. 12. Illustration of skipped variables in LCRVars.

TABLE VII
LCRs BETWEEN SKIPPED VARIABLES AND THE VARIABLES IN THE SUPPORT OF F IN LCRVARS

Case	Corresponding LCRs
$G = 0$	All LCR_g, where $g = \langle 0, 0, 0, \text{-}, \text{-} \rangle$ and $\langle 0, 1, 1, \text{-}, \text{-} \rangle$.
$G = 1$	All LCR_g, where $g = \langle 1, 0, 0, 0, 1 \rangle, \langle 1, 0, 0, 1, 0 \rangle, \langle 0, 0, 0, 1, 1 \rangle, \langle 0, 1, 1, 0, 0 \rangle, \langle 1, 1, 1, 0, 1 \rangle, \langle 1, 1, 1, 1, 0 \rangle, \langle 0, 1, 1, 1, 1 \rangle$.
$H = 0$	All LCR_g, where $g = \langle 0, \text{-}, \text{-}, 0, 0 \rangle$ and $\langle 0, \text{-}, \text{-}, 1, 1 \rangle$.
$H = 1$	All LCR_g, where $g = \langle 1, 0, 1, 0, 0 \rangle, \langle 1, 1, 0, 0, 0 \rangle, \langle 0, 1, 1, 0, 0 \rangle, \langle 0, 0, 0, 1, 1 \rangle, \langle 1, 0, 1, 1, 1 \rangle, \langle 1, 1, 0, 1, 1 \rangle, \langle 0, 1, 1, 1, 1 \rangle$.
$H = G$	All nonskew LCC_2 and LCC_4 LCRs.
$H = G'$	All LCR_g, where $g = \langle 1, 0, 1, 0, 1 \rangle, \langle 1, 1, 0, 1, 0 \rangle, \langle 0, 1, 1, 1, 1 \rangle, \langle 1, 1, 0, 0, 1 \rangle, \langle 0, 1, 1, 0, 0 \rangle, \langle 1, 0, 1, 1, 0 \rangle, \langle 0, 0, 0, 1, 1 \rangle$.

The variable subset Y is returned when both G and H are constants. In the last two scenarios, the program continues.

Step 2) A variable z in Y is selected and the functions are cofactored with respect to this variable.

Step 3) Based on Theorem 1, a variable belongs to the solution iff it belongs to the solutions of both cofactors. If one of the subproblems returns an empty set, there is no need to solve the other one.

Step 4) Fig. 12 is a fragment of a BDD illustrating the skipped variable situation. We want to check whether variable b has LCR with any variables in the support of $F_{a'}$. Since variable c is skipped over, we need to detect LCRs between variables b and c. It is easy to see that $F_{a'b'c'} = F_{a'b'c}$ and $F_{a'bc'} = F_{a'bc}$. For skipped variables, $G = F_{01} = F_{00}$ and $H = F_{11} = F_{10}$, to check if variable x has any LCRs to the skipped variables is equivalent to checking if there are any values of g that satisfy the following equation: $g_4 = (g_3 \oplus g_2) \bullet H \oplus (g_1 \oplus g_0) \bullet G$. This again depends on the values and relationships of G and H, as shown in Table VII. The intermediate result R is $(R_0 \cap R_1) \cup R_2$.

Step 5) This step checks to see if variable x has an LCR to variable z, according to Table III. If x has LCR to z, then z is added to the resulting set R.

The worst case complexity of the algorithm is cubic in the number of the BDD nodes, because the complexity of the procedure ComputeLCR is linear, while each call to LCRVars inside ComputeLCR has the worst case quadratic complexity.

TABLE VIII
BENCHMARK RESULTS FOR cm151a

LCR	Num. of LCRs	New (sec)	Naïve (sec)	Performance gain
LCR_{01}	0	0.23	2.19	10
LCR_{02}	11	0.74	2.19	3
LCR_{04}	0	0.16	2.15	13
LCR_{08}	11	0.71	2.15	3
LCR_{06}	0	0.05	2.21	44
LCR_{09}	0	0.1	2.19	21
LCR_{03}	0	0.05	2.20	44
LCR_{0C}	0	0.03	2.14	71
LCR_{05}	24	0.55	2.17	4
LCR_{0A}	46	0.65	2.13	3
LCR_{07}	0	0.2	2.13	11
LCR_{0B}	0	0.1	2.11	21
LCR_{0D}	0	0.09	2.13	24
LCR_{0E}	0	0.12	2.14	18
LCR_{0F}	56	0.63	2.06	3

TABLE IX
SYMMETRY AND PERFORMANCE DATA ON MCNC BENCHMARK

Benchmark statistics			Total LCRs	Performance		
Name	In/Out	|BDD|		Fast	Naïve	Gain
alu4	14/8	1182	129	0.01	0.18	18
dalu	75/16	1407	11249	0.15	3.33	22
des	256/245	4291	15241	0.17	1.49	9
frg2	143/139	2089	19212	0.07	1.13	16
k2	45/45	1381	10104	0.14	1.63	12
pair	173/137	5487	22577	0.16	8.86	55
rot	135/107	6119	7988	0.19	20.45	108
too_large	38/3	815	738	0.02	0.88	44
C1355	41/32	29586	256	0.29	170.98	590
C1908	33/25	9519	3362	0.23	29.62	129
C2670	233/140	4488	15669	1.40	86.43	62
C3540	50/22	24504	6243	1.81	72.00	40
C432	36/7	1226	212	0.01	2.57	257
C499	41/21	28177	256	0.28	155.34	555
C5315	178/123	2903	51327	0.93	17.66	19
C7552	207/108	8439	53725	0.91	160.27	176

However, for the benchmark functions, the observed runtime is close to linear in the number of the BDD nodes.

B. Implementation of the Algorithm

The proposed algorithm was implemented in C with the CU Decision Diagram package [25] and the EXTRA library of DD procedures [26].

BDDs are used to represent Boolean functions, while ZDDs are used to represent variable sets and LCR graphs. First, the shared BDD of a multioutput benchmark function is constructed. This BDD is not modified during the computation that follows. No additional BDD nodes are built, which makes the implementation very fast. Similar to other algorithms implemented using BDDs, partial results of computation are cached to prevent multiple calls with the same arguments. The calls to the caching procedures are omitted in the above discussion for simplicity. The cache lookups are performed before Step 2 and the cache insertions before returning the result in both ComputeLCR and LCRVars.

ZDDs provide a canonical representation of the LCR graph. The ZDDs are usually small, compared to the BDDs of the benchmark functions, because the LCR graphs are usually sparse. Even when a function has an LCR between each pair of variables, the ZDD representation is still compact. In this case, the LCR graph is a clique, whose ZDD representation is quadratic in the number of variables.

Although the program does not generate new BDD nodes, a small number of ZDD nodes are created to manipulate the LCR graph and the variables sets in the recursive procedures. However, experiments show that the increase in the number of ZDD nodes is still negligible, compared to the size of the shared BDDs of the original functions.

C. Experimental Results

The program was run on a 750-MHz Pentium III PC under Red Hat Linux 7.3. We only compare our results with the naïve method because it is the only other method known to compute all LCRs.

In Table VIII, the MCNC benchmark function cm151a (12 inputs, 1 output) is used to demonstrate the efficiency of the program when there is no LCR in the circuit. All 15 nonskew LCRs are computed and reported individually using the naïve method and the algorithm proposed in the paper. The runtimes reported are accumulated over 100 runs to capture reliable data. They include only LCR computation time and do not include the time to read the benchmark file and construct the BDDs. No variable pairs in cm151a have LCR_{01}. Our algorithm is 10$\times$ faster than the naïve method in this case. For LCR_{02}, which contains 11 variable pairs, the speedup is 3$\times$. Similar results can also be observed for other linear cofactor classes.

The larger the circuit, the greater is the speedup of calculating LCRs. Table IX gives the total LCRs and runtime information over a number of large MCNC benchmarks. The first three columns show the benchmark name, number of inputs and outputs, and the number of the BDD nodes after reading and reordering by the sifting algorithm [25]. The next column gives the total number of nonskew LCRs. Average runtime for both the fast and naïve methods are given in the two columns under "Performance." Each runtime is an average of 15 runs computing 15 different nonskew LCRs. Column "Gain" records the performance speedup between the fast and naïve algorithms. Table IX demonstrates a significant speedup of the fast algorithm over the naïve method for all benchmarks.

V. APPLICATIONS OF LCRS

Symmetry is an important characteristic of Boolean functions. Many applications in electronic design automation exploit classical symmetries to achieve better results and/or improve performance. It is worth noting that theoretical foundations for classical symmetries were formulated long before practical applications were identified and developed. LCRs, excluding classical symmetries and single-variable symmetries, are relatively new. In this section, we touch upon some natural applications of LCRs. We hope that the theoretical foundations of LCRs will encourage additional potential applications.

TABLE X
EFFECTIVENESS OF USING LCRS AS SIGNATURES

NPN Equivalence Class	Total	LCR	Classical
2-variable	4	3	3
3-variable	14	12	8
4-variable	222	172	20
5-variable	616126	4037	43

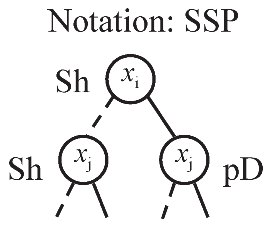


Fig. 13. Illustration of PKFDD expansion notation.

A. Boolean Matching

Boolean matching is a technique used to determine if a subfunction can be implemented with a specific cell from a given technology library. Two n-input Boolean functions match if one of them can be transformed into another by one or more of the following transformations: input permutation (P1), input negation (P2), and output negation (P3). Functions are P-equivalent under P1, NP-equivalent under P1 and P2, and NPN-equivalent under all three transformations.

Classical symmetries have been used as signatures for Boolean matching [5], [6], [27], [28]. We will show that using LCRs as signatures increase the distinguishing power significantly, compared to that of classical symmetries. The following theorems justify using LCRs as signatures for Boolean matching.

Theorem 2: P1 and P2 transformations do not alter the total number of LCRs in each linear cofactor class.

Theorem 3: P3 transformation changes the skew type for the LCRs in LCC_1 and LCC_3, and preserves the skew type for LCRs in LCC_2 and LCC_4.

Theorems 2 and 3 state that the number of LCRs in each linear cofactor class are preserved under NPN transformations, and therefore, can be used as signatures in Boolean matching. Table X compares the effectiveness of using all LCRs versus classical symmetries as signatures. The signature using LCRs includes eight integers, representing the total number of nonskew and skew LCRs in LCC_1, nonskew classical symmetries in LCC_2, skew classical symmetries in LCC_2, nonskew single-variable symmetries in LCC_2, skew single-variable symmetries in LCC_2, nonskew and skew LCRs in LCC_3, nonskew LCRs in LCC_4, and skew LCRs in LCC_4. Because both classical and single-variable symmetries are closed under NPN transformations, we can separate them into different components of the signatures. Since output negation changes the skew type for LCC_1 and LCC_3, we use the sum of skew and nonskew LCRs in those cases. The signature using classical symmetries includes two integers, representing the total number of nonskew classical symmetries and the total number of skew classical symmetries. The column "Total" contains the total number of NPN equivalence classes for two, three, and four variables. The column "LCR" lists the number of distinct LCR signatures in these NPN equivalence classes, whereas the column "Classical" lists the number of distinct classical signatures.

The signatures fail to distinguish two two-variable NPN equivalence classes $f = a'$ and $f = 1$ because the proposed computation algorithm only detects LCRs for variables in the support of the function. For these two functions, there is no variable pair in the support.

The above analysis shows that the distinguishing power of LCRs increases dramatically compared to that of classical symmetries, as the number of variables in the NPN equivalence class increases. With the efficient computation algorithm proposed in this paper, LCRs can easily be incorporated as filters to quickly prune unnecessary tautology checks. Other techniques [5], [6], [27], [28] can also be used in combination with LCRs to facilitate the task of Boolean matching.

B. PKFDD Minimization

The size of DDs is crucial in many computer-aided design (CAD) applications. For BDDs, the variable ordering is the only parameter available to reduce its size. Many algorithms have been proposed for BDD minimization [29], [30]. Several algorithms exploit classical symmetries to create smaller BDDs [8], [22], [31].

PKFDD is the most general bit-level DD for switching functions, since each node can be decomposed using Shannon, positive Davio, or negative Davio expansions. Canonicity is sacrificed in PKFDDs to achieve smaller DDs.

Exact PKFDD minimization was discussed in [32]. Because both the variable order and the decomposition type affect the size of the DD, it is hard to find a variable order and decomposition type at each node that result in the smallest PKFDD. Heuristic methods have been proposed in PKFDD minimization [33]–[35].

LCRs can be used to assist PKFDD minimization. While only classical and single-variable symmetries help in reducing the size of BDDs, all LCRs can reduce the size of PKFDDs. Detecting these relationships can help decide both the ordering of the variables and the decomposition type to be applied in order to reduce the size of the DD.

We use the following notation to represent the decomposition combination. The combination is denoted using three letters, with S representing Shannon expansion, P representing positive Davio expansion, and N representing negative Davio expansion. The order indicates the expansion type of the parent node, the left child, and the right child. For example, Fig. 13 illustrates the expansion for function f with notation "SSP" as decomposition types.

In Table XI, we list all the nonskew and skew LCRs and the corresponding expansion combinations that have shared nodes due to these LCRs (skew LCRs result in complemented edges in the DDs). In total, there are 27 different expansion combinations. Symbol "X" in the cell states that the LCR in the

TABLE XI
LCR TYPES WITH CORRESPONDING EXPANSIONS

Exp. Types	**Linear cofactor relationships**														
	LCR_{-1}	LCR_{-2}	LCR_{-4}	LCR_{-8}	LCR_{-6}	LCR_{-9}	LCR_{-3}	LCR_{-C}	LCR_{-5}	LCR_{-A}	LCR_{-7}	LCR_{-B}	LCR_{-D}	LCR_{-E}	LCR_{-F}
SSS					X	X	X	X	X	X					
SPS		X				X		X	X		X	X			
SNS	X				X			X		X	X	X			
SSP				X	X		X		X				X	X	
SSN			X			X	X			X			X	X	
SPP		X		X					X			X	X		X
SNN	X		X							X	X			X	X
SPN		X	X			X					X		X		X
SNP	X			X	X							X		X	X
PSS			X	X			X				X	X			X
PPS		X	X		X	X					X				X
PNS	X			X	X	X						X			X
PSP			X				X			X		X	X	X	
PSN				X			X		X		X		X	X	
PPP		X	X		X			X		X				X	
PNN	X			X		X		X	X				X		
PPN		X				X		X	X		X			X	
PNP	X				X			X		X		X	X		
NSS	X	X						X					X	X	X
NPS	X							X		X	X	X	X		
NNS		X						X	X		X	X		X	
NSP	X			X	X	X								X	X
NSN		X	X		X	X							X		X
NPP	X			X		X	X			X	X				
NNN		X	X		X		X		X			X			
NPN			X			X	X			X		X	X		
NNP				X	X		X		X		X			X	
SUM	10	10	10	10	12	12	10	10	10	10	12	12	12	12	10

column results in a shared node for the expansion combination in the corresponding row. Table XI shows that, for each pair of variables with an LCR, there are many expansion combinations that could be assigned to create shared nodes. If the diagrams are mapped directly to circuit implementation, it is beneficial to choose the combination that uses more Shannon expansions to reduce the cost. Also, for each expansion combination, there are six different LCRs that reduce the size of PKFDD. Table IV shows that LCRs are abundant in benchmark functions. It is, therefore, promising to attempt PKFDD reduction taking advantage of these functional properties.

C. Regular Layout

Layout regularity is desirable because it offers predictability in circuit area and delay. It also significantly simplifies routing, reduces gate output load and improves testability. Regularity of layout is especially important when circuits are mapped into programmable or field programmable logic devices and gate arrays, since the majority of these devices have a large portion of their routing resources available as local and neighbor-to neighbor connections.

One approach to achieve layout regularity is to transform a Boolean function into a specialized type of DD that can be mapped directly into regular circuits composed of Shannon and Davio gates. Boolean functions are transformed into pseudosymmetric BDD (PSBDD) through joint-vertex operations in [36]–[39]. This operation reintroduces the same variables at multiple levels, thereby increasing the number of levels comparing to that of an ROBDD. The benefit is a regular symmetric array structure, which is usually triangular in shape. This technique was generalized to create PSKFDD [40]–[42], which offers greater flexibility and increases the solution space. However, the process of generating PSKFDD also creates repetition of some variables, resulting in circuits that are larger in the number of gates, though regular.

Detecting and utilizing LCRs in Boolean functions can reduce the number of levels added to PSKFDDs as a result of repeating variables. First, we detect LCRs between all pairs of variables in the circuit. Then, we find the longest sequence of variables such that each pair of adjacent variables has an LCR. Next, the expansions are assigned to each variable on the path in such a way that the corresponding reduction type (as shown in Table XI) allows for merging of sufficient nodes so that the planar layout is created on these levels [14]. This guarantees that, for the variables included in the longest path, they will not be repeated in the DDs. For functions with no LCRs at all, or if the longest path does not include all variables, the rest of the diagram is constructed using a heuristic algorithm, which could potentially introduce repetition of variables.

We compared results generated using LCRs with earlier works on PSBDDs and PSKFDDs to achieve a regular layout. Fig. 14 shows that the LCR approach results in fewer logic levels in PSKFDDs comparing with that in [42]. It is worth pointing out that while the resulting PSKFDDs contain fewer levels than that of the corresponding PSBDDs, the logical complexity of each node is larger. Fig. 15 shows the level

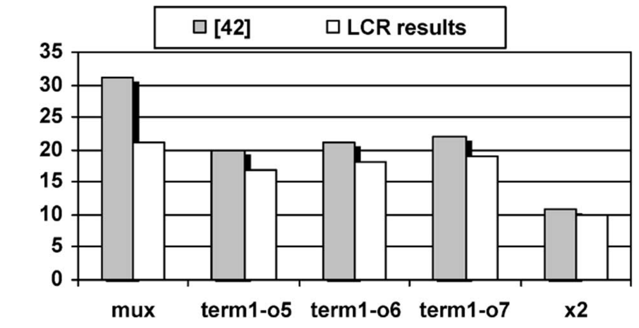


Fig. 14. Comparison with [42] on the number of levels in the DD.

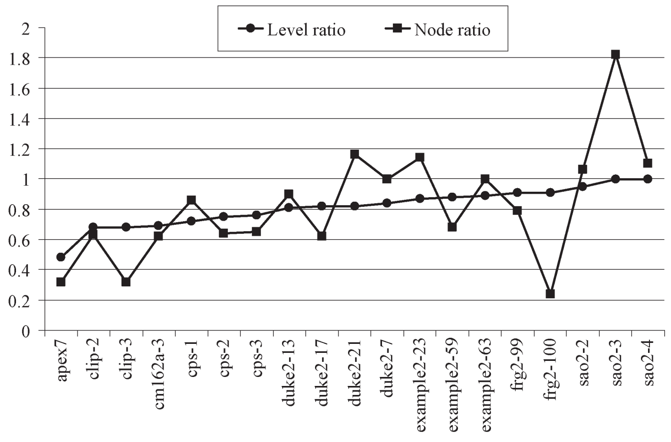


Fig. 15. Level and node ratio: LCR results normalized against [36].

and node ratios of the LCR approach to that of [36]. The LCR approach consistently produces fewer levels on all 19 benchmarks, even though the number of nodes may be more in a few cases. These comparisons show the potential of using LCRs in achieving aregular layout with less area.

D. Detecting Support-Reducing Bound Sets

Detecting support-reducing bound sets is an important step in Boolean decomposition. It affects both the runtime and the quality of results of several applications in technology mapping and resynthesis. Mishchenko *et al.* [7] proposed an efficient method to compute all support-reducing bound sets—that is, all groups of variables that can lead to the decomposition resulting in the reduction of a function's support [46]. This method is quite efficient because it does not explicitly enumerate through all bound sets. Instead, it creates all bound sets implicitly and uses a cache to avoid repeated computations. Detailed profiling of the decomposition system has shown that the exhaustive bound-set detection is the most time-consuming task in the flow. In this section, we will show that LCRs can be used as heuristics to detect most of the support-reducing bound sets of three, four, and five variables, while achieving great performance improvement over the exhaustive method [15].

A bound set X_1 leads to an n-to-k support-reducing decomposition if k satisfies $\lceil \log_2 \mu \rceil \leq k < n$, where $n = |X_1|$ and μ is the number of distinct cofactors for variables in X_1 [43], [44]. The presence of nonskew LCC_2 LCRs in a Boolean function results in shared nodes in the BDD for the function,

TABLE XII
AVERAGE G/T AND G/F RATIOS AND PERFORMANCE GAIN

Name	3-var		4-var		5-var		Gain
	G/T (%)	G/F (%)	G/T (%)	G/F (%)	G/T (%)	G/F (%)	
9symml	73	88	66	95	67	99	36
alu4	87	99	76	99	65	89	61
apex6	96	84	94	92	98	95	32
b15	99	97	98	99	98	100	38
b17	92	91	94	96	95	99	41
b20	96	97	90	99	91	99	41
c8	80	99	89	100	94	100	27
C2670	98	61	95	80	97	95	34
C3540	93	94	91	96	91	99	35
C5315	98	73	99	81	100	93	29
C7552	97	85	97	89	98	96	34
dalu	60	84	61	95	65	100	147
frg2	94	97	89	97	84	99	55
i10	98	85	96	94	98	99	34
k2	100	100	100	100	100	100	18
pair	100	60	100	71	100	84	40
rot	80	81	83	87	88	96	37
Ave	90	87	89	92	90	97	43

thereby reducing the number of cofactors. Therefore, we can use nonskew LCC_2 LCRs as heuristics to identify potential support-reducing bound-set candidates.

Heuristic 1) We compute the set of variable pairs with two nonskew LCC_2 LCRs. These sets are 2-to-1 support-reducing bound sets. We then iteratively add another variable from the support of the function to form three-, four-, and five-variable support-reducing bound sets.

Heuristic 2) We compute the sets of three variables that contain one nonskew LCC_2 LCR in at least two of the variable pairs. These are potentially 3-to-2 support-reducing sets. The four- and five-variable bound sets are formed by iteratively adding another variable from the support of the function to the existing set.

The experiments were conducted on a set of MCNC and International Test Conference (ITC)'99 benchmarks. The following notation is used in Table XII. Column "G/T" shows the ratio of true support-reducing bound sets found by the heuristics to the total number of support-reducing bound sets of a given size. Column "G/F" is the ratio of true support-reducing bound sets to the total number of candidates found by the heuristics. These two ratios characterize the efficiency of the heuristics from two different points of view. Column "Gain" represents the runtime improvement of the proposed method compared to the exhaustive method.

As shown in Table XII, the heuristics can detect about 90% of all support-reducing bound sets. The percentage of support-reducing bound sets detected is inversely proportional to the performance gain. Benchmark dalu has the worst "G/F" and "G/T" ratios among these benchmarks, but it has the highest performance gain of a factor of 147. On the other hand, the heuristics can detect all the support-reducing bound sets for $k2$, but the performance gain is only a factor of 18. Therefore, there is a tradeoff between the number of support-reducing bound sets detected versus the performance improvement over the exhaustive method. Even though other heuristics can be employed to further improve the "G/F" and "G/T" ratios, we believe that achieving around 90% on G/F and G/T ratios and 40× performance gain is a good middle ground.

Experimental results show that the heuristic method is much faster than the exhaustive method [7], yet it finds most of the support-reducing bound sets of three, four, and five variables. The detected support-reducing bound sets typically result in simpler decomposition functions, compared to those that are not detected by the proposed method. As a result, the constructive decomposition, which constitutes an important step in technology mapping and resynthesis, can be performed more efficiently.

VI. Conclusion

This paper presents LCRs for pairs of variables in a Boolean function. This notion subsumes classical and single-variable symmetries. Experiments on MCNC benchmarks show that these relationships are common in Boolean functions.

An efficient algorithm is proposed to detect LCRs. The algorithm is characterized as follows.

1) It works on the shared BDD of multioutput functions and computes the LCR information for each output.
2) It exploits the compactness and canonicity of the ZDD representation to store the LCR information computed for a node in the shared BDD.
3) It computes all 30 types of LCRs.
4) It is particularly fast when applied to Boolean functions with no LCRs.

The experimental results show that the overall performance of the algorithm is significantly better than the naïve method.

The proposed efficient LCR detection enables the use of LCRs in CAD applications. Several such applications of LCRs are discussed in this paper: Boolean matching, DD minimization, synthesis of regular layout-friendly circuits, and detection of support-reducing bound sets in Boolean functions. We expect other applications of LCRs to emerge in very large scale integration (VLSI) IC design flow.

Acknowledgment

The authors would like to thank Prof. R. Brayton for proposing the term "linear cofactor relationship" (LCR). The authors also acknowledge anonymous reviewers for their comments, which played an important role in shaping and clarifying the paper.

References

[1] C. R. Edward and S. L. Hurst, "A digital synthesis procedure under function symmetries and mapping methods," *IEEE Trans. Comput.*, vol. C-27, no. 11, pp. 985–997, Nov. 1978.

[2] B.-G. Kim and D. L. Dietmeyer, "Multilevel logic synthesis of symmetric switching functions," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 4, pp. 436–446, Apr. 1991.

[3] M. Chrzanowska-Jeske, W. Wang, J. Xia, and M. Jeske, "Disjunctive decomposition of switching functions using symmetry information," in *Proc. IEEE Int. Symp. Integrated Circuits and System Design (SBCCI)*, Manaus, Brazil, Sep. 2000, p. 67.

[4] V. N. Kravets, "Constructive multi-level synthesis by way of functional properties," Ph.D. dissertation, Comput. Sci. Eng., Univ. Michigan, Ann Arbor, 2001.

[5] Y.-T. Lai, S. Sastry, and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Oct. 1992, pp. 452–458.

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

[7] A. Mishchenko, X. Wang, and T. Kam, "A new enhanced constructive decomposition and mapping algorithm," in *Proc. Design Automation Conf.*, Anaheim, CA, Jun. 2003, pp. 143–147.

[8] C. Scholl, D. Möller, P. Molitor, and R. Drechsler, "BDD minimization using symmetries," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 2, pp. 81–100, Feb. 1999.

[9] V. N. Kravets and K. A. Sakallah, "Generalized symmetries in Boolean functions," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2000, pp. 526–532.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–692, Aug. 1986.

[11] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," in *Proc. Eur. Design Automation Conf. (EDAC)*, Brussels, Belgium, 1992, pp. 43–47.

[12] U. Kebschull and W. Rosenstiel, "Efficient graph-based computation and manipulation of functional decision diagrams," in *Proc. Eur. Design Automation Conf. (EDAC)*, Paris, France, 1993, pp. 278–282.

[13] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams," in *Proc. Design Automation Conf.*, San Diego, CA, Jun. 1994, pp. 415–419.

[14] M. Chrzanowska-Jeske, A. Mischenko, J. S. Zhang, and M. Perkowski, "Logic synthesis for layout regularity using decision diagrams," in *Proc. Int. Workshop Logic Synthesis*, Temecula, CA, Jun. 2004, pp. 149–154.

[15] J. S. Zhang, M. Chrzanowska-Jeske, A. Mischenko, and J. R. Burch, "Detecting support-reducing bound sets using 2-cofactor symmetries," in *Proc. Asia South Pacific Design Automation Conf.*, Shanghai, China, Jan. 2005, pp. 266–271.

[16] M. Davio, J.-P. Deschamps, and A. Thayse, *Discrete and Switching Functions*. New York: McGraw-Hill, 1978.

[17] M. A. Perkowski, "The generalized orthonormal expansion of functions with multiple-valued inputs and some of its applications," in *Proc. Int. Symp. Multi-Valued Logic*, Sendai, Japan, 1992, pp. 442–450.

[18] M. Chrzanowska-Jeske, "Generalized symmetric and generalized pseudo-symmetric functions," in *Proc. Int. Conf. Electronics, Circuits and Systems*, Pafos, Cyprus, Sep. 1999, pp. 343–346.

[19] ——, "Generalized symmetric variables," in *Proc. Int. Conf. Electronics, Circuits, and Systems*, St. Julians, Malta, Sep. 2001, pp. 1147–1151.

[20] D. Möller, J. Mohnke, and M. Weber, "Detection of symmetry of Boolean functions represented by ROBDDs," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1993, pp. 680–684.

[21] A. Mishchenko, "Fast computation of symmetries in Boolean functions," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 11, pp. 1588–1593, Nov. 2003.

[22] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1994, pp. 628–631.

[23] C.-C. Tsai and M. Marek-Sadowaska, "Generalized Reed–Muller forms as a tool to detect symmetries," *IEEE Trans. Comput.*, vol. 45, no. 1, pp. 33–40, Jan. 1996.

[24] S. Minato, "Zero-suppressed BDDs for set manipulation in combinational problems," in *Proc. Design Automation Conf.*, Dallas, TX, 1993, pp. 272–277.

[25] F. Somenzi. *CUDD Package, Release 2.3.1*. [Online]. Available: http://vlsi.Colorado.EDU/~fabio/CUDD/cuddIntro.html

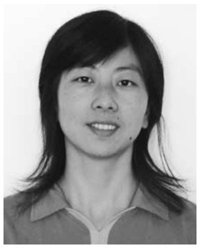
[26] A. Mishchenko. *EXTRA Library of DD Procedures*. [Online]. Available: http://www.ee.pdx.edu/~alanmi/research/extra.htm

[27] C.-C. Tsai and M. Marek-Sadowska, "Boolean matching using generalized Reed–Muller form," in *Proc. Design Automation Conf.*, San Diego, CA, Jun. 1994, pp. 339–344.

[28] H. Savoj, M. J. Silva, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Boolean matching in logic synthesis," in *Proc. Eur. Design Automation Conf.*, Hamburg, Germany, Feb. 1992, pp. 168–174.

[29] R. Drechsler, N. Drechsler, and W. Günther, "Fast exact minimization of BDDs," in *Proc. Design Automation Conf.*, San Francisco, CA, Jun. 1998, pp. 200–205.

[30] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, 1991, pp. 472–475.
[31] D. Möller, P. Molitor, and R. Drechsler, "Symmetry based variable ordering for ROBDDs," in *Proc. IFIP Workshop Logic and Architecture Synthesis*, Grenoble, France, 1994, pp. 47–53.
[32] T. Sasao, *Logic Synthesis and Optimization*. Norwell, MA: Kluwer, 1993.
[33] R. Drechsler and B. Becker, "Dynamic minimization of OKFDDs," in *Proc. Int. Conf. Computer Design*, Austin, TX, Oct. 1995, pp. 602–607.
[34] R. Drechsler, B. Becker, and N. Göckel, "Minimization of OKFDDs by genetic algorithms," in *Proc. Int. Symp. Soft Computing*, Reading, MA, 1996, pp. B:263–B:528.
[35] P. Lindgren, R. Drechsler, and B. Becker, "Improved minimization methods of pseudo Kronecker expressions," in *Proc. Int. Symp. Circuit and Systems*, Monterey, CA, 1998, pp. VI:187–VI:190.
[36] M. Chrzanowska-Jeske, Y. Xu, and M. Perkowski, "Logic synthesis for a regular layout," *VLSI Des.—An International Journal of Custom-Chip Design, Simulation and Testing*, vol. 10, no. 1, pp. 35–55, 1999.
[37] W. Wang and M. Chrzanowska-Jeske, "Generating linear arrays using symmetry chain," in *Proc. Int. Workshop Logic Synthesis*, Lake Tahoe, CA, Jun. 1999, pp. 115–119.
[38] M. Chrzanowska-Jeske and Z. Wang, "Mapping of symmetric and partially symmetric functions to CA-type FPGAs," in *Proc. IEEE Midwest Symp. Circuits and Systems*, Rio de Janeiro, Brazil, 1995, pp. 290–293.
[39] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. I. Long, "Wave steering in YADDs: A novel non-iterative synthesis and layout technique," in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 446–471.
[40] M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice diagrams using Reed–Muller logic," in *Proc. Int. Workshop Applications Reed–Muller Expansions*, Oxford, U.K., 1997, pp. 85–102.
[41] M. Chrzanowska-Jeske and J. Zhou, "AND/EXOR regular function representation," in *Proc. IEEE Midwest Symp. Circuits and Systems*, Sacramento, CA, 1997, pp. 1034–1037.
[42] P. Lindgren, R. Drechsler, and B. Becker, "Synthesis of pseudo-Kronecker lattice diagrams," in *Proc. Int. Workshop Applications Reed–Muller Expansions*, Victoria, Canada, 1999, pp. 197–204.
[43] R. L. Ashenhurst, "The decomposition of switching functions," in *Computational Lab*, vol. 29. Cambridge, MA: Harvard Univ., 1959, pp. 74–116.
[44] A. Curtis, *New Approach to the Design of Switching Circuits*. Princeton, NJ: Van Nostrand, 1962.
[45] [Online]. Available: http://www.ece.pdx.edu/~alanmi/research/lcr/index.htm
[46] V. N. Kravets and K. A. Sakallah, "Constructive library-aware synthesis using symmetries," in *Proc. Design, Automation and Test Europe (DATE)*, Paris, France, 2000, pp. 208–216.



Jin S. Zhang (M'93) received the B.S. degree in electrical engineering and the B.A. degree in English for science and technology from Tianjin University, Tianjin, China, in 1991, the M.S.E.E. degree from Tianjin University in 1994, and the M.S. degree in electrical and computer engineering from Portland State University, Portland, OR, in 2001. She is currently working toward the Ph.D. degree at Portland State University.

From 1995 to 2002, she has worked with Lattice Semiconductor Corporation, Cadence Design Systems, and Real Intent, Inc., on logic and layout verification.

Malgorzata Chrzanowska-Jeske (S'86–M'86–SM'98) received the M.S. degree in electrical engineering from Politechnika Warszawska (the Technical University of Warsaw), Warsaw, Poland, in 1972, and the Ph.D. degree in electrical engineering from Auburn University, Auburn, AL, in 1988.

She has served on the faculty of the Technical University of Warsaw, and as a Design-Automation Specialist at the Research and Production Center of Semiconductor Devices, Warsaw. Since 1989, she has been with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR, where she is currently a Professor and the Department Chair. Her research interests include vertically integrated computer-aided design (CAD) for very large scale integration (VLSI) IC and Mixed Signal System On Chip (MS-SOC), three-dimensional (3-D) chip architectures, field-programmable gate array (FPGA) synthesis and architecture, design for manufacturability and testability in deep submicrometer, and nano/bio electronics. She has published more than 100 technical papers and serves as a panelist for the National Science Foundation (NSF) and as a reviewer for National Research Council Canada (NRC) and many international journals and conferences.

Dr. Chrzanowska-Jeske has served on the Technical, Steering, and Organizing Committees of many international conferences, and was a Technical Chair of the 2002 International Conference on Electronics, Circuits and Systems. In 2004, she was a Guest Editor of the *International Journal on Analog Integrated Circuits and Signal Processing*. She was a recipient of the 1990 Best Paper Award from the Alabama Section of IEEE for a paper on the simulation of a bipolar transistor at low temperature published in the IEEE TRANSACTIONS ON ELECTRON DEVICES. She is a member of the VLSI Systems and Applications Technical Committee of the IEEE Circuits and Systems Society, a member of the ACM, and of Eta Kappa Nu.

Alan Mishchenko (M'99) graduated from Moscow Institute of Physics and Technology, Moscow, Russia, in 1993, and received the Ph.D. degree in computer science from Glushkov Institute of Cybernetics, Kiev, Ukraine, in 1997.

He has been a Research Scientist in the U.S. since 1998. His work has been funded by several grants from Intel Corporation. Currently, he is affiliated with the University of California at Berkeley. His research interests are in developing computationally efficient methods for logic synthesis and verification.

Jerry R. Burch (S'92–M'92) received the B.S. and M.S. degrees in computer science from the California Institute of Technology, Pasadena, in 1984 and 1985, respectively, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1992.

From 1992 to 1994, he was a Postdoctoral Scholar in the Computer Science Department at Stanford University. From 1994 to 2002, he was with Cadence Berkeley Labs. He is currently with the Advanced Technology Group of Synopsys, Inc., in Hillsboro, OR.