

Detecting Support-Reducing Bound Sets using Two-Cofactor Symmetries¹

Jin S. Zhang
Department of ECE
Portland State University
Portland, OR 97201
jinsong@ece.pdx.edu

Malgorzata Chrzanowska-Jeske
Department of ECE
Portland State University
Portland, OR 97201
jeske@ece.pdx.edu

Alan Mishchenko
Department of EECS
UC Berkeley
Berkeley, CA 94708
alanmi@eecs.berkeley.edu

Jerry R. Burch
Advanced Technology Group
Synopsys Inc.
Hillsboro, OR 97124
jrb@synopsys.com

Abstract - Detecting support-reducing bound sets is an important step in Boolean decomposition. It affects both the quality and the runtime of several applications in technology mapping and re-synthesis. This paper presents an efficient heuristic method for detecting support-reducing bound sets using two-cofactor symmetries. Experiments on the MCNC and ITC benchmarks show an average 40x speedup over the published exhaustive method for bound set construction.

1. INTRODUCTION

Constructive synthesis [1-4] is a technique combining Boolean decomposition and technology mapping during logic synthesis so as to achieve better quality of synthesized circuits. Kravets et al [3] used symmetry as the functional property to restrict the decomposition functions to a small library of pre-characterized symmetric primitives, while achieving support-reduction in the composition function. In their notion of symmetry, functions stay invariant under swapping of pairs or groups of variables. These are first- and higher-order classical symmetries.

This work was later extended by [1]. One of the extensions concerns the exhaustive computation of 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. 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 paper, we propose an improvement to the decomposition-mapping flow of [1] by making use of functional properties. A key observation is there is a tight relationship between support-reducing bound sets and the two-cofactor symmetries between the variables in these bound sets. We propose heuristics to identify the variables with two-cofactor symmetries as potential support-reducing bound set candidates. Experimental results show that the proposed heuristics can detect over 80% of all support-reducing bound sets, while achieving a 40x runtime reduction compared to the exhaustive method.

The remainder of this paper is organized as follows. We introduce background information in Section 2. Section 3 gives a theoretical motivation for the proposed heuristics.

Experimental results are listed in Section 4. Finally, Section 5 concludes the paper.

2. BACKGROUND

In this paper, we use completely specified Boolean functions and Boolean variables.

The *support* of f , $supp(f)$, is the set of variables X that f depends on. The support size is denoted by $|X|$.

A *cofactor* of a function $f(\dots, x_i, \dots, x_j, \dots)$ with respect to (w.r.t.) variables x_i and x_j , is the function derived from f by substituting x_i and x_j with specific values. For example, the cofactor of f w.r.t. $x_i = 0$ and $x_j = 0$, is the function $f[x_i \leftarrow 0, x_j \leftarrow 0]$, which is denoted by f_{00} .

2.1 Support-reducing Decomposition

For a function f and a subset of its support X_1 , the set of distinct cofactors, $q_1(X)$, $q_2(X)$, ..., $q_\mu(X)$, of f w.r.t. X_1 is derived by substituting all assignments of X_1 into $f(X)$ and eliminating duplicated functions. The number of distinct cofactors, μ , is the *column multiplicity* of f w.r.t. X_1 .

Given a partition of X into two disjoint subsets, X_1 and X_2 , the Ashenurst-Curtis decomposition of $f(X)$ is:

$$f(X) = h(g_1(X_1), g_2(X_1), \dots, g_k(X_1), X_2).$$

Subsets X_1 and X_2 are called the *bound set* and the *free set*, respectively. Functions $g_i(X_1)$, $1 \leq i \leq k$, are the decomposition functions. Function $h(G, X_2)$ is the composition function. The 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 $n = |X_1|$ [5,6].

In this paper, we are only interested in finding support-reducing bound sets involving three, four and five variables because they are the ones most often used in functional decomposition and technology mapping. Bound sets with more variables can be similarly derived using the proposed heuristics. Figure 1 shows a schematic of the Ashenurst-Curtis decomposition for function f with a 4-to-2 support-reducing decomposition function g .

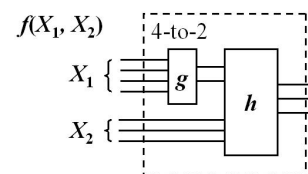


Fig 1. Illustration of X_1 as a 4-to-2 support-reducing bound set

¹ This work was supported in part by the NSF grant CCR-9988402

We assume that the reader is familiar with the basic terminology of Binary Decision Diagrams (BDDs) [7]. When BDDs are used to represent a Boolean function $f(X)$, the cofactors of f w.r.t. X_1 are the nodes in the BDD of f that have incoming edges from the nodes located above the cut separating X_1 from X_2 , assuming the variables in X_1 are above those in X_2 in the variable order [8,9].

Example 1. Figure 2 shows the upper part of the BDD for some function $f(X)$ and the decomposition pattern for bound set $X_1 = \{x, y, z\}$. Since there are four unique cofactors c_0, c_1, c_2 and c_3 , $k = \lceil \log_2 4 \rceil = 2$, and $n = |X_1| = 3$, it follows that X_1 is a 3-to-2 support-reducing bound set. The decomposition pattern is derived by labeling each equivalence class of cofactors with a unique number starting from zero. To assure that the representation of decomposition patterns is canonical, we require that the integer derived by putting together all the numbers in the natural order of cofactors is minimum. The canonical representation of the decomposition pattern in Figure 2 is “00123333”.

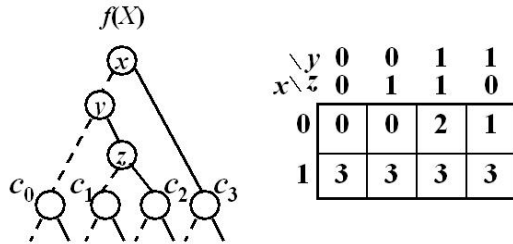


Fig 2. Decomposition pattern for bound set $\{x, y, z\}$

2.2 Two-cofactor Symmetries

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

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 *non-skew non-equivalent symmetry*, denoted by $x_i \text{NE} x_j$. *Non-skew equivalent symmetry*, denoted by $x_i \text{E} x_j$, 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* exists when two cofactors are complemented rather than equivalent to each other. For example, a Boolean function with *skew equivalent symmetry* is such that $f_{00} \oplus f_{11} = 1$. Taken together, all of the above symmetries are called *classical symmetries* [10].

For any two variables in a Boolean function, there are six possible cofactor pairs. *Single variable symmetries* [10] are defined when the function has equivalent (non-skew symmetry) or complement (skew symmetry) relationships of the other four cofactor pairs, not included in classical symmetries. For example, $f_{00} = f_{01}$ is a non-skew relationship that means the negative cofactor of f w.r.t variable x_i does not depend on x_j .

Both classical and single variable symmetries involve equivalent or complement relationships of two cofactors. We

use the term *two-cofactor symmetry* to include all classical and single variable symmetry. Table 1 summarizes the functional invariance, cofactor relationships and symbols of all two-cofactor symmetries.

Table 1. Two-cofactor Symmetries

Invariant of the function (non-skew/skew)	Cofactor relationship	Symbol
$f(\dots x_i, \dots x_j, \dots) \oplus f(\dots x_j, \dots x_i, \dots) = 0/1$	$f_{10} \oplus f_{01} = 0/1$	T_1 (NE)
$f(\dots x_i, \dots x_j, \dots) \oplus f(\dots x_j', \dots x_i', \dots) = 0/1$	$f_{00} \oplus f_{11} = 0/1$	T_2 (E)
$f(\dots 0, \dots x_j, \dots) \oplus f(\dots 0, \dots x_j', \dots) = 0/1$	$f_{00} \oplus f_{01} = 0/1$	T_3
$f(\dots 1, \dots x_j, \dots) \oplus f(\dots 1, \dots x_j', \dots) = 0/1$	$f_{10} \oplus f_{11} = 0/1$	T_4
$f(\dots x_i, \dots 0, \dots) \oplus f(\dots x_i', \dots 0, \dots) = 0/1$	$f_{00} \oplus f_{10} = 0/1$	T_5
$f(\dots x_i, \dots 1, \dots) \oplus f(\dots x_i', \dots 1, \dots) = 0/1$	$f_{01} \oplus f_{11} = 0/1$	T_6

Classical symmetries are very common and have many applications, such as logic synthesis [11], Boolean matching [12] and BDD minimization [13]. Single variable symmetries, on the other hand, are rarely used. Table 2 gives the number of non-skew (NS) and skew (S) classical and single variable symmetries in a randomly selected group of MCNC benchmark functions. It is clear that there are many more non-skew single variable symmetries than non-skew classical symmetries. It is therefore, more beneficial to also consider single variable symmetries rather than just classical symmetries as in [3] when looking for support-reducing bound sets.

Table 2. Two-cofactor Symmetry Statistics

Name	Classical		Single Variable	
	NS	S	NS	S
9symml	36	0	0	0
alu2	7	1	15	6
alu4	11	1	31	6
apex6	408	16	2540	29
b1	5	3	0	4
b9	81	8	464	9
c8	114	16	343	25
cc	44	3	166	6
f51m	5	9	4	67
frg1	7	3	101	4
i1	103	14	268	18
pm1	85	1	255	2
k2	559	0	4191	0
rot	540	93	5216	99
t481	8	0	16	0
term1	70	2	510	0
too_large	20	0	482	0
x1	300	6	1696	7
z4ml	22	5	0	52
Total	2425	181	16298	334

3. DETECTING SUPPORT-REDUCING BOUND SETS

The presence of non-skew two-cofactor symmetries in a Boolean function results in functionally equivalent cofactors, thereby reducing the number of cofactors. It is intuitive that the variables with non-skew two-cofactor symmetries, when grouped with other variables, may lead to support-reducing bound sets.

Example 2. In Figure 3(a), variables x and y have symmetry T_1 , which means $f_{01} = f_{10}$ (the equivalence is indicated with the shaded nodes). As a result, there are 3 distinct cofactors w.r.t. x and y , i.e. column multiplicity $\mu = 3$. Variables x and y are not support-reducing because $k = \lceil \log_2 3 \rceil = 2$ and $n = |\{x, y\}| = 2$. In Figure 3(b), on the other hand, variables x and y have two symmetries, T_1 and T_3 . This reduces the number of distinct cofactors to 2, which leads to a 2-to-1 support-reducing decomposition with $k = \lceil \log_2 2 \rceil = 1$.

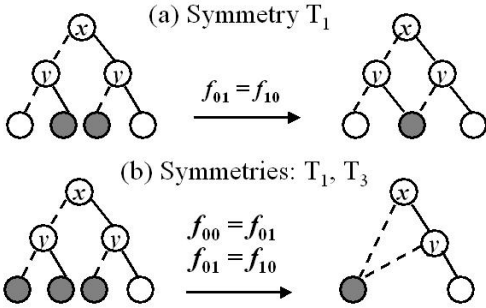


Fig 3. Two-cofactor symmetries and support-reducing bound sets

Example 2 shows that the existence of one two-cofactor symmetry is not a sufficient condition for the bound set to be support-reducing. However, the existence of multiple symmetries can be used as a heuristic to quickly identify support-reducing bound set candidates, thereby reducing the search space and runtime.

Another way of looking at the relationship between support-reducing bound sets and non-skew two-cofactor symmetries is to study how common these symmetries are in the existing support-reducing bound sets. To this end, we generated 3-to-1, 3-to-2, 4-to-1, 4-to-2 and 4-to-3 decomposition patterns using several arbitrarily chosen MCNC benchmarks: frg2, des, alu4, and rot, as shown in Table 3.

Table 3. Percentage of decomposition patterns with symmetries

Decomposition pattern	Total Patterns	Unique patterns	Patterns with symmetries	Percentage
3-to-1	4400	60	60	100%
3-to-2	2328	352	308	87.5%
4-to-1	3947	204	204	100%
4-to-2	2170	778	699	89.8%
4-to-3	217	107	67	62.6%

The total number of these patterns is given in column 2 of Table 3. There is a lot of duplications among these patterns. The number of unique patterns is given in Column 3. For each

of these patterns, we check whether the variables have non-skew two-cofactor symmetries. Column 4 gives the number of patterns which contain symmetries; Column 5 gives the percentage of the unique decomposition patterns with symmetries.

Table 3 demonstrates that the majority of the decomposition patterns in the selected benchmark functions contains non-skew two-cofactor symmetries, and therefore they could potentially be identified using these symmetries. The percentage in Table 3 is derived based on unique decomposition patterns. However, in actual circuits, typical decomposition patterns appear more than once, as witnessed by Table 3. Therefore, the heuristics based on symmetries can detect a higher percentage of support-reducing bound sets than shown in Table 3. This conjecture is confirmed by the experimental results.

Example 3. Figures 4 and 5 show two 3-to-2 decomposition patterns, respectively. P_1 has no non-skew two-cofactor symmetry, while P_2 has two-cofactor symmetries T_1 and T_3 . Using the same encoding of cofactors, the decomposition functions g_1 and g_2 for P_2 are noticeably simpler than those of P_1 .

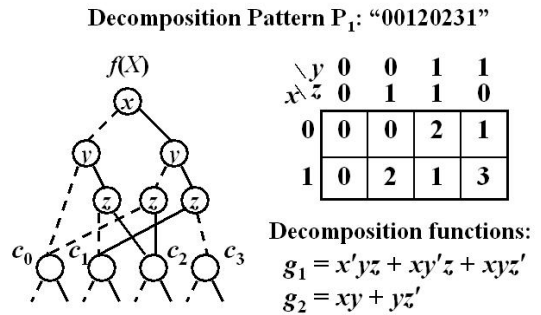


Fig 4. Complex decomposition functions when w/o symmetry

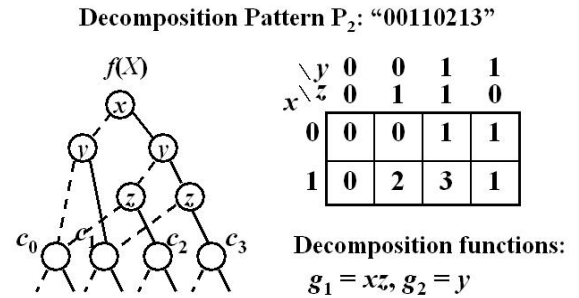


Fig 5. Simpler decomposition functions with symmetries

Example 3 demonstrates that the decomposition patterns without symmetries can be more complex. This is important because even though they are detected using a more expensive, exhaustive method, it is unlikely that they will lead to a decomposition that is better than the one afforded by simpler patterns. It appears reasonable to consider only support-reducing bound sets with simple decomposition patterns, because they can be quickly detected using heuristic filters based on non-skew two-cofactor symmetries.

The following theorems lay the foundation of the heuristics used in this paper.

Theorem 1 [14]. If variables x and y in a Boolean function f have two types of two-cofactor symmetries (*double symmetries*), then variables x and y form a 2-to-1 support-reducing bound set.

Theorem 2. If X_1 is an n -to- k support-reducing bound set, then adding another variable to X_1 will form an $(n+1)$ -to- m support-reducing bound set, where $m \leq k + 1$.

Theorem 2 is straightforward. Figure 6 illustrates the case for a 2-to-1 support-reducing bound set $\{x, y\}$. Adding another variable z to this bound set always results in a 3-to-2 bound set. If the variable pairs $\{x, z\}$ or $\{y, z\}$ have symmetries, then the bound set $\{x, y, z\}$ can potentially be 3-to-1 support-reducing.

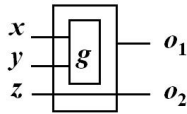


Fig 6. Illustration of Theorem 2

Heuristic 1. According to Theorem 1 and 2, we compute the set of variable pairs having double symmetries. 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.

Example 4. Function f has 5 variables in its support $\{a, b, c, d, e\}$. If variables a and b have double symmetries, then $\{a, b\}$ are 2-to-1 support-reducing. According to theorem 2, $\{a, b, c\}$, $\{a, b, d\}$, $\{a, b, e\}$, $\{a, b, c, d\}$, $\{a, b, c, e\}$, $\{a, b, d, e\}$ are also support-reducing.

There is an alternative efficient method to compute n -to-1 support-reducing bound sets by using BDD-based disjoint support decomposition [15]. However, this method is much more complicated, compared to the algorithm used in this paper to compute the two-cofactor symmetries.

Example 5. Figure 7 is the upper part of the BDD for function $f(X)$ with a symmetric variable set $\{x, y, z\}$. Variable pairs $\{x, y\}$, $\{x, z\}$, and $\{y, z\}$ all have classical symmetry T_1 . It is obvious that $\{x, y, z\}$ is a 3-to-2 support-reducing bound set. However, each variable pair $\{x, y\}$, $\{x, z\}$, and $\{y, z\}$ is not 2-to-1 support-reducing.

Example 5 shows that in a 3-to-2 support-reducing bound set, it is possible that any two variables do not have double symmetries and therefore are not support-reducing. However there are multiple symmetries among different variable pairs. As a result, the whole set is support-reducing.

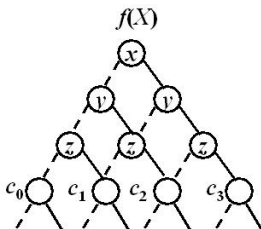


Fig 7. Support-reducing decomposition exists for variable pairs w/o double symmetries

Heuristic 2. We compute the sets of three variables that contain one symmetry in at least two of the variable pairs.

This set is potentially 3-to-2 support-reducing. The four- and five-variable bound sets are formed by iteratively adding another variable from the support of the function to the existing set.

Heuristic 2 does not guarantee that all the returned bound sets are support-reducing because there are cases when the existence of symmetries in two variable pairs does not result in a support-reducing bound set, as established in the following example.

Example 6. In Figure 8, variables x and y have symmetry T_3 ; variables y and z have symmetry T_5 (the BDD is not reduced to make the existence of these symmetries more obvious). However, the bound set composed of x, y and z has five unique cofactors and is, therefore, not support-reducing

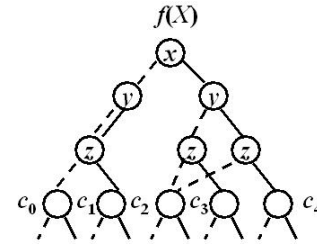


Fig 8. Heuristic 2 could return non support-reducing bound sets

The two heuristics, although simple, are quite powerful in finding the majority of support-reducing bound sets. The runtime is on average 40x faster, compared with the exhaustive method, because symmetry computation based on the algorithm in [16] is very fast.

4. EXPERIMENTAL RESULTS

The program was written in C using the CUDD package [17] and the EXTRA library of DD procedures [18]. We compare our results with the exhaustive bound set computation method [1], which is also included in the EXTRA library. The programs were run on a 750MHz Pentium III PC running Red Hat Linux 7.3.

The experiments were conducted on a set of MCNC and ITC'99 benchmarks. We perform partial collapsing of these circuits using the *eliminate* command in MVSIS. Each created node had no more than 20 fanins. This was done because:

- 1) It is useful to compare the exhaustive and the heuristic methods on functions whose support size does not exceed a given value. The results of the comparison will also be valid for larger functions because the gain of the heuristic method over the exhaustive one increases with the support size.
- 2) Functions with 10 to 15 inputs are typically used in re-synthesis because when the functions have more than 15 inputs, the decomposition is too slow, and when the functions are smaller than 10, re-synthesis is too local to be efficient.

In our experiments, we used the Boolean functions of logic nodes derived by selective collapsing of several multi-level benchmarks, so that the results are representative of a wide variety of circuits.

The MCNC benchmark *term1* is an extreme example demonstrating the variation in the effectiveness of the method. Table 4 gives the detailed results computed for this circuit. Five collapsed nodes were selected with input sizes from 10 to 14. Results for three-, four-, and five-variable bound sets are reported separately.

The following notation is used in Table 4. Column “T” shows the total number of support-reducing bound sets of a given size. Column “F” represents all the candidate bound sets found by the two heuristics. Column “G” is the number of “good” bound sets, that is, true support-reducing bound sets among the candidates. Column “G/T” shows the ratio of “good” bound sets to the total number of support-reducing bound sets, while column “G/F” shows the ratio of good bound sets to the total number of candidates.

For node 4 in Table 4, the heuristics work very well in detecting all the support-reducing bound sets. However, for node 2, only 27% of the three-variable support-reducing bound sets are detected. Two reasons contribute to this low detection rate: 1) there are fewer two-cofactor symmetries in this node; 2) Most of the candidates found by Heuristic 2 turned out not to be support-reducing. On average for benchmark *term1*, the proposed heuristics detect over 80% of all support-reducing bound sets; and out of all the candidates detected by the heuristics, around 80% of them are truly support-reducing. Benchmark *term1* is an extreme case because most other benchmarks don’t exhibit noticeable variation in the effectiveness of the heuristics applied to the collapsed nodes.

The heuristic method is on average 37x faster than the exhaustive method presented in [1] for *term1*. Column “E Time” represents the time it takes for the exhaustive method to detect all the bound sets. Column “H Time” is the runtime of the proposed heuristic method. The runtimes are in seconds; they contain only bound set detection time. The time it takes to read the benchmark file, construct the BDDs, and perform synthesis is not included. The reported numbers for both cases are the results of running the programs 100 times in order to get reliable measurements. “Gain” is the ratio between the two numbers, representing the performance gain of the proposed method.

Table 5 gives the detailed result for a larger circuit, *b14_opt_C*, a subset of the Vipor processor from the ITC ’99 benchmark set. In this case, the chosen nodes were selected to be of larger size: 15 to 20 inputs, in which case the heuristic method shows greater performance gain than that for *term1*.

Table 6 is the average results on a few MCNC and ITC benchmarks. Five nodes of each benchmark are used with input size ranging from 10 to 15. Column “G/F” and “G/T” represent the same ratios as in Tables 4 and 5. Column “H1/G” is the ratio of the number of bound sets found by heuristic 1 to the total number of bound sets found by both heuristics.

As shown in Table 6, about 75% of the support-reducing bound sets are detected using Heuristic 1. This is good because all bound sets detected using Heuristic 1 are true support-reducing. We can also see that the percentage of

support-reducing bound sets detected is inversely proportional to the performance gain. *dalu* has the worst “G/F” and “G/T” ratios among these benchmarks, but it has the highest performance gain of 147. On the other hand, the heuristics can detect all the support-reducing bound sets for *k2*, but the performance gain is only 18. Therefore, there is a tradeoff between the number of support-reducing bound sets detected vs. the performance improvement over the exhaustive method. Even though other heuristics based on symmetries can be employed to further improve the “G/F” and “G/T” ratios, we believe that achieving over 80% on G/F and G/T ratios and 40x performance gain is a good middle ground.

5. CONCLUSIONS

This paper describes a heuristic method to compute support-reducing bound sets for completely specified Boolean functions. The heuristics make use of the two-cofactor symmetries between variable pairs. These symmetries can be efficiently computed from the BDD representation of the functions. The new method is much faster than the exhaustive method [1], 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 cannot be detected by the proposed method. The constructive decomposition, which constitutes an important step in technology mapping and re-synthesis, can be performed more efficiently using the proposed method.

REFERENCES

- [1] A. Mishchenko, X. Wang, and T. Kam. “A new enhanced constructive decomposition and mapping algorithm”. *Proc. DAC ’03*, pp. 143-147, June 2003.
- [2] V. N. Kravets. “Constructive multi-level synthesis by way of functional properties”, *PhD Thesis*, University of Michigan, 2001.
- [3] V. N. Kravets and K. A. Sakallah. “Constructive library-aware synthesis using symmetries”, *Proc. DATE ’00*, pp. 208-216, 2000.
- [4] V. N. Kravets and K. A. Sakallah. “Re-synthesis of multi-level circuits under tight constraints using symbolic optimization”, *Proc. ICCAD ’02*, pp. 687-693, 2002.
- [5] R. L. Ashenurst. “The decomposition of switching functions”, *Computational Lab*, Harvard University, Vol. 29, pp. 74-116, 1959.
- [6] A. Curtis. “New approach to the design of switching circuits”, *Van Nostrand*, Princeton, NJ, 1962.
- [7] R. E. Bryant. “Graph-based algorithms for Boolean function manipulation”, *IEEE Trans. Comp.*, Vol. C-35, No. 8, pp. 677-692, August 1986.
- [8] Y. T. Lai, M. Pedram and S. B. K. Vrudhula. “BDD-based decomposition of logic functions with applications to FPGA synthesis”, *Proc. DAC ’93*, pp. 642-647, 1993.
- [9] T. Sasao. “FPGA design by generalized functional decomposition”, In T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic publishers, 1993.
- [10] C. R. Edward and S. L. Hurst. “A digital synthesis procedure under function symmetries and mapping methods”, *IEEE Trans. On Computers*, vol. C-27, No.11, pp. 985-997, November 1978.

[11] B.-G. Kim and D. L. Dietmeyer. "Multilevel logic synthesis of symmetric switching functions", *IEEE Trans. CAD*, 10(4), pp.436-446, April 1991.

[12] Y.-T. Lai, S. Sastry, and M. Pedram. "Boolean matching using binary decision diagrams with applications to logic synthesis and verification", *Proc. ICCAD*, pp. 452-458, October 1992.

[13] Ch. Scholl, D. Möller, P. Molitor, and R. Drechsler. "BDD minimization using symmetries", *IEEE Trans. CAD*, 18(2) pp. 81-100, February 1999.

[14] M. Chrzanowska-Jeske, W. Wang, J. Xia, and M. Jeske. "Disjunctive decomposition of switching functions using symmetry information," *Proc IEEE SBCCI '00*, pp. 67, September 2000.

[15] V. Bertacco and M. Damiani. "Disjunctive decomposition of logic functions", *Proc. ICCAD '97*, pp. 78-82, 1997.

[16] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch. "Fast computation of generalized symmetries in Boolean functions", *Proc. IWLS '04*, pp. 424-430, June 2004.

[17] F. Somenzi. *CUDD Package, Release 2.3.1*. <http://vlis.Colorado.EDU/~fabio/CUDD/cuddIntro.html>

[18] A. Mishchenko. *EXTRA Library of DD procedures*. <http://www.ee.pdx.edu/~alanmi/research/extra.htm>

Table 4. Experimental results on MCNC benchmark "term1"

Node	Ins	3-var					4-var					5-var					E Time	H Time	Gain
		T	F	G	G/T	G/F	T	F	G	G/T	G/F	T	F	G	G/T	G/F			
1	10	36	32	30	83%	94%	159	130	126	79%	97%	244	228	228	93%	100%	4.92	0.17	29
2	11	25	37	10	27%	40%	182	180	126	69%	70%	437	376	376	86%	100%	7.72	0.19	41
3	12	38	37	31	82%	84%	269	230	200	74%	87%	719	588	576	80%	98%	11.36	0.26	44
4	13	86	86	86	100%	100%	395	395	395	100%	100%	1015	1015	1015	100%	100%	15.26	0.44	35
5	14	84	174	84	100%	48%	441	761	441	100%	58%	1330	1810	1330	100%	73%	24.79	0.63	39
Ave.					81%	71%				85%	82%				94%	92%			37

Table 5. Experimental results on ITC99 benchmark "b14_opt_C"

Node	Ins	3-var					4-var					5-var					E Time	H Time	Gain
		T	F	G	G/T	G/F	T	F	G	G/T	G/F	T	F	G	G/T	G/F			
1	15	117	122	112	96%	92%	798	786	751	94%	96%	2645	2523	2495	94%	99%	6.91	0.01	691
2	16	33	31	30	91%	97%	272	220	213	78%	97%	1385	943	925	67%	98%	2.71	0.01	271
3	17	156	155	148	95%	96%	1386	1254	1217	88%	97%	5370	4804	4750	89%	99%	1.96	0.02	98
4	18	72	70	69	96%	99%	621	557	545	88%	98%	3219	2666	2609	81%	98%	4.36	0.01	436
5	19	102	102	102	100%	100%	805	801	801	99%	100%	3973	3855	3855	97%	100%	9.38	0.02	469
6	20	123	121	116	94%	96%	1149	1092	1029	90%	94%	6468	5876	5548	86%	94%	8.41	0.02	420
Ave.					95%	96%				90%	97%				86%	98%			398

Table 6. Average G/T and G/F ratios and performance gain

Name	3-var			4-var			5-var			Gain
	H1/G	G/T	G/F	H1/G	G/T	G/F	H1/G	G/T	G/F	
9symml	68%	73%	88%	56%	66%	95%	52%	67%	99%	36
alu4	81%	87%	99%	73%	76%	99%	77%	65%	89%	61
apex6	87%	96%	84%	84%	94%	92%	84%	98%	95%	32
b15	83%	99%	97%	82%	98%	99%	85%	98%	100%	38
b17	90%	92%	91%	85%	94%	96%	85%	95%	99%	41
b20	81%	96%	97%	73%	90%	99%	75%	91%	99%	41
c8	82%	80%	99%	70%	89%	100%	86%	94%	100%	27
C2670	78%	98%	61%	75%	95%	80%	76%	97%	95%	34
C3540	38%	93%	94%	32%	91%	96%	36%	91%	99%	35
C5315	70%	98%	73%	63%	99%	81%	67%	100%	93%	29
C7552	84%	97%	85%	79%	97%	89%	80%	98%	96%	34
dal	77%	60%	84%	60%	61%	95%	54%	65%	100%	147
frg2	87%	94%	97%	81%	89%	97%	78%	84%	99%	55
i10	78%	98%	85%	77%	96%	94%	81%	98%	99%	34
k2	100%	100%	100%	100%	100%	100%	100%	100%	100%	18
pair	100%	100%	60%	100%	100%	71%	100%	100%	84%	40
rot	63%	80%	81%	55%	83%	87%	55%	88%	96%	37
Average	79%	90%	87%	73%	89%	92%	75%	90%	97%	43