SYMMETRY DETECTION IN BOOLEAN FUNCTIONS USING SAT SOLVERS

Dan Steffy[*] *Stacey Balamucki*[†] *Debatosh Debnath*[‡]

ABSTRACT

This paper presents a method for detecting symmetric variables in a Boolean function which is given as a multilevel netlist of gates. A function $f(x_1, \ldots, x_n)$ is symmetric with respect to x_i and x_j if and only if $f_{x_i\bar{x}_j} = f_{\bar{x}_i x_j}$. The proposed method is based on determining such functional equivalence for every pair of variables by using a Boolean satisfiability (*aka* SAT) solver. The technique is promising for functions whose two-level representations are difficult to generate or manipulate. Experimental results for benchmark functions by using a preliminary implementation of the method are encouraging.

1. INTRODUCTION

Detecting symmetric variables in an arbitrarily large Boolean function—especially when the function is represented as a multilevel netlist of gates—is a nontrivial task [1, 2]. If the presence of symmetric variables is known, many problems in logic synthesis can be solved more efficiently through the use of more specific algorithms that can take advantage of such information. Symmetry properties have applications in circuit design [3], Boolean matching [4], and building binary decision diagrams (BDDs) [5]. An overview of the different symmetry detection algorithms can be found in [1].

In this paper, we reduce the symmetry detection problem to a Boolean satisfiability problem [6]. Recently many efficient systems have been developed to solve the Boolean satisfiability [7, 8]. The SAT solvers are based on algorithms such as the Davis Putnam backtrack search, conflict driven learning, and other heuristic algorithms [7, 8]. By reducing the problem of symmetry detection to Boolean satisfiability we hope to take advantage of these well developed methods. In our implementation we use zChaff [7], which is one of the top SAT solvers. The usefulness of the proposed method is shown through experimental results. We also outlined some strategies that can be used to improve the applicability of the method.

The remainder of the paper is organized as follows: Section 2 introduces the terminology. Section 3 discusses the symmetry detection method. Section 4 reports the experimental results. Section 5 outlines how the proposed method can be improved. Section 6 presents conclusions.

2. DEFINITIONS AND TERMINOLOGY

This section defines the basic terminology that we use throughout the paper.

Definition 1 *A Boolean function is said to be **totally symmetric** if it remains invariant under any permutation of its variables.*

Definition 2 *A Boolean function is said to be **partially symmetric** with respect to a subset of its variables if it remains invariant under any permutation of those variables.*

Definition 3 *A **conjunctive normal form (CNF)** of a Boolean function is its representation as a conjunction of sum terms, where a sum term is a disjunction of complemented and/or uncomplemented variables. A sum term in a CNF is referred to as a **clause**.*

3. SYMMETRY DETECTION METHOD

By using Shannon expansion theorem we can prove that a Boolean function $f(x_1, \ldots, x_n)$ is symmetric with respect to x_i and x_j if and only if $f_{x_i\bar{x}_j} = f_{\bar{x}_i x_j}$, where $f_{x_i\bar{x}_j} = f|_{x_i=1,x_j=0}$ and $f_{\bar{x}_i x_j} = f|_{x_i=0,x_j=1}$ [2, 9]. To find the largest sets of symmetric variables we have to check functional equivalence of $f_{x_i\bar{x}_j}$ and $f_{\bar{x}_i x_j}$ for every pair of variables of f. However, checking functional equivalence is a computationally difficult problem. The problem often becomes more challenging if the functions are represented as multilevel netlists of small elements such as gates. Usually BDD-based methods are used to determine functional equivalence;

[*]Dept. of Mathematics and Statistics, Oakland Univ., Rochester, MI 48309, U.S.A. (e-mail: desteffy@oakland.edu).

[†]Dept. of Comp. Sci. and Eng., Lake Superior State Univ., Sault Ste. Marie, MI 49783, U.S.A. (e-mail: sbalamuc@student.lssu.edu).

[‡]Dept. of Comp. Sci. and Eng., Oakland Univ., Rochester, MI 48309, U.S.A. (e-mail: debnath@oakland.edu).

Table 1. Translating Gate Functions into CNFs [8, 10].

Gate	Gate Function	CNF
OR	$v = \sum_{i=1}^{j} u_i$	$\left[\prod_{i=1}^{j}(\bar{u}_i + v)\right] \cdot \left(\sum_{i=1}^{j} u_i + \bar{v}\right)$
AND	$v = \prod_{i=1}^{j} u_i$	$\left[\prod_{i=1}^{j}(u_i + \bar{v})\right] \cdot \left(\sum_{i=1}^{j} \bar{u}_i + v\right)$
NOT	$v = \bar{u}$	$(u + v) \cdot (\bar{u} + \bar{v})$

however, for many instances such methods require huge memory resources and computation time.

In this paper, our strategy is to use SAT solvers for determining functional equivalence. A Boolean formula is said to be *satisfiable* if it evaluates to 1 for at least a combination of its inputs; otherwise it is *unsatisfiable*. SAT solvers can be used to determine Boolean satisfiability and unsatisfiability. The formula $f_{x_i\bar{x}_j} \oplus f_{\bar{x}_i x_j}$ is unsatisfiable if and only if $f_{x_i\bar{x}_j}$ and $f_{\bar{x}_i x_j}$ are functionally equivalent.

We consider multilevel netlist of given functions in Berkeley logic interchange format (BLIF). Since CNF is used as the input for the majority of SAT solvers, we need to transform the netlist in BLIF into CNF. In BLIF a Boolean function is represented as a set of interconnected logic blocks, where each block represents a two-level AND-OR network. We convert each of the logic blocks into CNF one by one; for ease of conversion, we introduce many temporary variables. Table 1 shows how basic *gate functions* can be transformed into CNFs, and the following example shows how we translate a functional equivalence checking problem into a CNF, which is used as the input for the SAT solver.

Example 1 Let

$$f = x_1\bar{x}_2 + x_1x_3 + x_2x_3\bar{x}_4, \text{ and} \quad (1)$$
$$g = x_1\bar{x}_2 + x_4 \quad (2)$$

be given functions. We want to verify whether f and g are functionally equivalent. From (1), by introducing three temporary variables t_1, t_2, and t_3, we generate the following four gate functions:

$$f = t_1 + t_2 + t_3, \quad (3)$$
$$t_1 = x_1\bar{x}_2, \quad (4)$$
$$t_2 = x_1x_3, \text{ and} \quad (5)$$
$$t_3 = x_2x_3\bar{x}_4. \quad (6)$$

Let the CNFs for the gate functions in (3), (4), (5), and (6) be y_1, y_2, y_3, and y_4, respectively. By using Table 1, we have

$$y_1 = (\bar{t}_1 + f)(\bar{t}_2 + f)(\bar{t}_3 + f)(t_1 + t_2 + t_3 + \bar{f}),$$
$$y_2 = (x_1 + \bar{t}_1)(\bar{x}_2 + \bar{t}_1)(\bar{x}_1 + x_2 + t_1),$$
$$y_3 = (x_1 + \bar{t}_2)(x_3 + \bar{t}_2)(\bar{x}_1 + \bar{x}_3 + t_2), \text{ and}$$
$$y_4 = (x_2 + \bar{t}_3)(x_3 + \bar{t}_3)(\bar{x}_4 + \bar{t}_3)(\bar{x}_2 + \bar{x}_3 + x_4 + t_3).$$

Similarly, gate functions for (2) are

$$g = t_4 + x_4, \text{ and} \quad (7)$$
$$t_4 = x_1\bar{x}_2, \quad (8)$$

where t_4 is another temporary variable. Let the CNFs for the gate functions in (7) and (8) be y_5 and y_6, respectively. By using Table 1, we have

$$y_5 = (\bar{t}_4 + g)(\bar{x}_4 + g)(t_4 + x_4 + \bar{g}), \text{ and}$$
$$y_6 = (x_1 + \bar{t}_4)(\bar{x}_2 + \bar{t}_4)(\bar{x}_1 + x_2 + t_4).$$

Since $f = g$ if $f \oplus g$ is unsatisfiable, a CNF for checking the equivalence of f and g be $y_1y_2y_3y_4y_5y_6(f + \bar{g})(\bar{f} + g)$. ∎

Symmetry between variables in a Boolean function is a transitive relation, i.e., if x_i and x_j are symmetric and x_j and x_k are symmetric, then x_i and x_k are also symmetric. When determining all the largest sets of symmetric variables in a function, this property can greatly reduce the number of pairs of variables checked for symmetry.

4. EXPERIMENTAL RESULTS

In order to evaluate the suitability of using SAT solvers in symmetry detection, we implemented the proposed method in C++ and conducted experiments by using a set of benchmark functions on an Intel Pentium 4 processor (2.8-GHz) running Linux. For each function the method generates a set of CNFs and writes them to files; the zChaff SAT solver [7] is then used to read the CNFs and to determine whether each of them is satisfiable.

Table 2 presents experimental results for the benchmark functions. We first list the benchmarks and their number of inputs and outputs. We then list the number of outputs that we found to be totally symmetric as well as partially symmetric. For brevity we list the sizes of the symmetric variable sets for only the output with index 0. If multiple symmetric variable sets of the same size appear, we list the number of sets of that size in the parenthesis. We next include some statistics about what information was sent to the zChaff. We list

Table 2. Experimental Results.

Data	Inputs	Outputs	Total	Partial	Sizes	Clauses	Variables	Time
9symml	9	1	1	0	9	1004	336	0.50
c8	28	18	0	18	25	1380	456	3.87
cc	21	20	0	20	2,18	513	232	0.97
cht	47	36	0	36	43	1261	408	3.83
cm150a	21	1	0	0	none	394	170	6.36
cm151a	12	2	0	0	none	191	90	1.51
cm152a	11	1	0	0	none	88	42	1.05
cm162a	14	5	0	5	2(2),6	339	148	1.25
cm163a	16	5	0	5	2,10	307	138	0.94
cm85a	11	3	0	1	none	340	156	1.35
cmb	16	4	0	4	4,12	283	126	0.56
comp	32	3	0	1	none	940	396	22.60
count	35	16	0	16	2,30	835	370	3.05
cu	14	11	0	11	2,10	449	190	0.90
f51m	8	8	0	6	none	872	218	1.11
frg1	28	3	0	3	2(2),3	1949	302	11.90
ldd	9	19	0	18	2(2),3	793	292	0.88
mux	21	1	0	0	none	458	148	6.88
my_adder	33	17	0	17	2(15),3	1202	424	12.56
parity	16	1	1	0	16	265	124	0.92
pcler8	27	17	0	17	2,9,16	434	198	1.73
pcle	19	9	0	9	2,8,9	304	134	1.10
pm1	16	13	0	13	3,13	472	212	0.77
sct	19	15	0	15	16	965	352	1.95
tcon	17	16	0	16	16	260	132	0.52
unreg	36	16	0	16	30	652	298	1.68
x2	10	7	0	7	2,7	311	126	0.70
z4ml	7	4	0	4	2(2),3	644	158	0.43

the average number of clauses and the average number of variables sent to the SAT solver each time it was called. This gives some rough estimate of the complexity of the problem zChaff was called to solve. In addition, we list the computation time in seconds; the average time per output is listed for multiple-output functions.

Table 3 compares computation time of our method with that of another method developed by Tsai and Marek-Sadowska [1], which is based on the representation of functions in the generalized Reed-Muller (GRM) forms. The GRM-based method [1]—which was run on a DEC5000 workstation—outperforms our method on these instances and is able to solve some instances that our method is unable to solve; it is somewhat expected because of the exploratory nature of our initial implementation, which has significant room for improvement as outlined in the next section.

5. STRATEGIES FOR IMPROVEMENTS

The experimentation with the initial implementation of our symmetry detection method provided the following insight about how it can be improved.

- A major limiting factor of our method is the size of the problem we send to zChaff, where the problem size is determined by the number of clauses and the number of variables on which the clauses depend. Since each of the given BLIF files represents a network in terms of small elements, it is necessary to introduce many temporary variables when directly translating BLIF into CNF. Table 2 shows that even for functions with only about 10 variables the CNF contains hundreds of variables. In order to alleviate the problem, our strategy is to first collapse the given multilevel BLIF specification into a set of larger two-level nodes and then translate them into CNF.

Table 3. Comparison with GRM-Based Method [1].

Data	Inputs	Outputs	SAT Time	GRM Time
cm82a	5	3	0.19	0.044
f51m	8	8	1.11	0.113
z4ml	7	4	0.43	0.113

- A complete integration of our program with zChaff would greatly reduce the total run time, because it would eliminate the time required to write and then read a huge number of CNF files.
- As it is outlined in Example 1, to translate $f_{x_i\bar{x}_j} \oplus f_{\bar{x}_i x_j}$ into CNF, our implementation first generates CNFs for both $f_{x_i\bar{x}_j}$ and $f_{\bar{x}_i x_j}$ and then combines them by using CNF for exclusive-OR. Since CNFs for both $f_{x_i\bar{x}_j}$ and $f_{\bar{x}_i x_j}$ are derived from the same multilevel netlist, they often have many common clauses that are independent of x_i and x_j. Although only one copy of such common clauses is sufficient, our present implementation generates two copies of them—one copy with the CNF for $f_{x_i\bar{x}_j}$ and the other copy with the CNF for $f_{\bar{x}_i x_j}$. The use of only one copy of the common clauses would result in considerably fewer clauses as well as temporary variables in the CNF for $f_{x_i\bar{x}_j} \oplus f_{\bar{x}_i x_j}$. We are presently working on to generate such common clauses only once. It should be noted that the SAT solvers are unable to detect such common clauses because $f_{x_i\bar{x}_j}$ and $f_{\bar{x}_i x_j}$ introduce different sets of temporary variables in their respective CNFs.
- It is often possible to quickly determine sets of variables which are not symmetric by using functional properties that are easy to compute [1, 2]. This can be used to initially eliminate the possibility of many symmetries, thereby improving the speed of any symmetry detection algorithm.

6. CONCLUSIONS AND COMMENTS

One of the strengths of SAT solvers is their ability in determining whether multilevel representations of two Boolean functions are functionally equivalent; this equivalence checking is done without generating any two-level representations, which are difficult to generate and manipulate. Moreover, many symmetry detection methods require structural and functional properties; however, SAT solvers are able to work without such information, which makes our symmetry detection method more general.

The primary objective of the present implementation of the proposed method is to determine the feasibility and practicality of using SAT solvers for symmetry detection. Our experimental results demonstrate that the method can easily handle problems with moderate size. Currently we are working on to incorporate the improvement strategies outlined in the previous section. We believe the new implementation would be able to handle significantly more complex problems and would require shorter computation time than the present implementation.

7. ACKNOWLEDGMENTS

This work was supported in part by the REU Program of the NSF. Additional support was provided by the Ford Motor Company and DaimlerChrysler Corporation. We thank Dr. L. Zhang for helping us with their zChaff software, Mr. B. Clark for assistance in programming, and Professors I. K. Sethi, F. Mili, and J. Li for their support and encouragement.

8. REFERENCES

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

[2] S. R. Das and C. L. Sheng, "On detecting total or partial symmetry of switching functions," *IEEE Trans. Comput.*, vol. 20, no. 3, pp. 352–355, Mar. 1971.

[3] B.-G. Kim and D. L. Dietmeyer, "Multilevel logic synthesis of symmetric switching functions," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 436–446, Apr. 1991.

[4] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," in *Proc. IEEE European Conference on Design Automation*, Feb. 1993, pp. 86–92.

[5] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1994, pp. 628–631.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[7] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. IEEE/ACM Design Automation Conference*, June 2001, pp. 530–535.

[8] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *Proc. IEEE/ACM Design Automation Conference*, June 2000, pp. 675–680.

[9] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[10] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, Jan. 1992.