Graph Automorphism-based Algorithm for Determining Symmetric Inputs

Chen-Ling Chou*, Chun-Yao Wang, Geeng-Wei Lee*, and Jing-Yang Jou*
* *Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.*
{nicy, gwlee, jyjou}@eda.ee.nctu.edu.tw
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
wcyao@cs.nthu.edu.tw

Abstract

We propose a graph automorphism-based algorithm for computing maximal sets of symmetric inputs of circuits. It can be used to identity nonsymmetric inputs in a circuit and enhance the efficiency of input matching, library binding, as well as logic verification problems. We conduct the experiments on some benchmarks. The experimental results demonstrate that our approach distinguishes more nonsymmetric inputs than that of previous work.

1. Introduction

Two inputs x_i an x_j of a logic function f(X) are said to be symmetric, if exchanging x_i and x_j does not change f, i.e., $f(x_1..., x_i,..., x_j,...x_n) = f(x_1,..., x_j,..., x_i,..., x_n)$[12]. The problem of finding maximal symmetric input sets is discussed in [11-13]. The definition of this symmetry translates into the following requirements for the cofactors of function $f_{x_i\overline{x_j}} = f_{\overline{x_i}x_j}$ with respect to any two inputs x_i and x_j .

Symmetric input sets are important in problems of design verification and diagnosis [3]-[5][7]-[10] and of technology mapping [2][6][14][17].

For these applications, our goal in this paper is to efficiently compute maximal symmetric input sets of a given function, i.e., focus on nonskew nonequivalence symmetry only. In parallel to our work on this problem [13][15][16], methods using BDD's [1] are investigated in [13][15]. However, these BDD-based algorithms are not applicable to the designs that described in behavior level or RT-level, e.g., soft Intellectual Property, or that do not have compact BDD representation. Thus, simulation-based approach [16] was proposed to compute the maximal sets of symmetric inputs. [16] establishes two steps to accomplish the input symmetric identification. The first step uses heuristic to identify inputs that do not belong to the same symmetric input set. Although the signature-based heuristic used in [2][14][17] can be used for this purpose, they were applied to circuits having compact BDD representation. The second step uses test generation techniques to further identify the inputs that were not distinguished by the heuristic. The efficiency of [16]-like approach for computing the maximal sets of symmetric inputs in circuits without compact BDD representation depends on the "ability" of heuristics. Good heuristics can distinguish more nonsymmetric inputs prior to entering computation intensive test generation step. Thus, this paper focuses on the first step of [16]-like approach. We propose a graph automorphism-based heuristic to distinguish nonsymmetric inputs as many as possible. The experimental results show that our heuristic can distinguish much more nonsymmetric inputs than [16] for most benchmarks circuits, where we only apply the heuristic while [16] applies both heuristic and test generation technique.

This work was supported in part by ROC National Science Council under Grant NSC89-2215-E-009-009

2. Symmetric-ASymmetric Inputs (SASIs) representation

SASIs represent the maximal symmetric inputs sets. For an N-input circuit, we assume that all inputs are symmetric initially, and the corresponding SASIs representation is (1 2 3 ... N). If we claim that input *i* is asymmetric to the other inputs by our methods, the input *i* is isolated from original group and can be expressed as (*i*)(1 2 ... *i*-1 *i*+1 ... N). By the SASIs representation, if any two inputs are not placed in the same group, then they are nonsymmetric inputs. Otherwise they are "possibly" symmetric.

3. Graph automorphism-based ACSP

The proposed heuristic for finding maximal symmetric input sets is named Automatic Computing Symmetric Procedure (ACSP). Its flow chart is shown in Fig. 1 which is similar to the concept proposed in [16]. We observe that the stages which profoundly influence the efficiency of ACSP are Pattern_Generation and SASIs_Calculation.

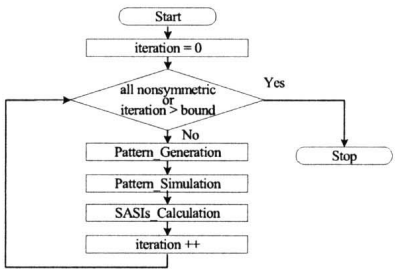

Figure 1. The flowchart of the ACSP.

3.1. Pattern_Generation

Definition 3.1 : For an N-input combinational circuit, the set that consists of all patterns with m 1s and (N-m) 0s is denoted as θ_m^N ,where m∈ [0, 1, 2, . . .,N -1, N].

Theorem 3.1: For an N-input circuit, the pattern set $\in \theta_{m_1}^{n_1} X \theta_{m2}^{n2} X \ldots X \theta_{m_k}^{n_k}$ with $n_1 + n_2 + \ldots + n_k = N$, $0 \le m_p \le n_p$ where p = 1,2,…,k and satisfies at least one group with different logic assignments can be the generated pattern sets in the SASIs $(a_1 a_2 \cdots a_{n_1})(b_1 b_2 \cdots b_{n_2}) \cdots (r_1 r_2 \cdots r_{n_k})$ and each of them is capable of distinguishing more nonsymmetric inputs.

For example, given a 5-input circuit, assume the SASIs becomes (123)(45) after θ_1^5 , θ_4^5 simulations. In this case, the further generated pattern sets come from $\theta_{m_1}^3 \times \theta_{m_2}^2$ where $m_1 \in$ [0, 1, 2, 3] and $m_2 \in$ [0, 1, 2]. Therefore, the possible generated pattern sets are listed in Table I. Note that through Theorem 3.1, the real generated pattern sets are $A_5 \sim A_8$. Table II shows the selected generated patterns of θ_2^5 and their corresponding outputs. The patterns in A_5 can be separated into three sets according to their outputs. Assume handling the set {01010} first, we can know that inputs 1 and 2 are nonsymmetric and inputs 2 and 3 are nonsymmetric, either. For the same reason, we continue handling other sets if it really can distinguish the inputs. Notice that assume these generated pattern sets have p different groups and certainly can be separated into p sets. After arbitrary (p-1) sets are processed, we ensure that the last one cannot distinguish any nonsymmetric inputs. Thus, only gray patterns in Table II are used to be the valid generated pattern sets to figure out the updated SASIs which is discussed in SASIs_Calculation using the graph automorphism algorithm in Section 3.2.

3.2. SASIs_Calculation

An undirected, weighted graph G(V,E) is constructed in this step, which corresponds to the set S1 with |S1| patterns, P_1 to $P_{|S1|}$. $P_i[j]$ in S1 denotes the j^{th} bit in P_i where i = 1 ~ |S1| and j = 1 ~N. The vertex V*k* in G corresponds to the k^{th} input variable in S1. For all patterns P_1 to $P_{|S1|}$ in S1, when $P_i[k] = P_i[k'] = 1$, an edge V*k*V*k'* is added into G and W(V*k*V*k'*)=1. The problem of distinguishing the nonsymmetric inputs in SASIs by S1 patterns is now transformed to finding all automorphisms of G.

Definition 3.2 : DV of a graph is a vector that contains of each vertex's degree, that is, DV[i] = degree of i^{th} vertex.

Definition 3.3 : The partial vector of vertex V_i is the i^{th} row of Adj(G). Besides, if vertex i is not in a single element group (SEG) in the SASI representation, it is called automorphism message of i, denoted as AM_i.

Now, there are four steps in finding automorphism of G, Aut(G), where G is an undirected graph with N vertices.

Step 1(Disjoint Graph - DG): If the graph is composed by t disconnected subgraphs with different number of vertices, the SASI representation is divided into t groups.

Step 2(Degree Vector - DV): Calculate the DV of the graph G. Then group the vertices with the same degrees into one group.

The updated SASIs can be obtained from the intersection of automorphisms derived from Step 1 and Step 2 now.

Step 3(Repeated Automorphism - RA): Grouping the partial vector of each SEG vertex and intersect this grouping result with the updated automorphism representation. If the updated automorphism representation has newly generated SEG, then repeat Step 3, otherwise go to Step 4.

Step 4(Automorphism Message - AM): Keeping AM_i if vertex i has different neighbors in its group.

We demonstrate the SASIs_Calculation stage using the following example in Fig. 2. The four steps of SASI_Calculation are shown in Fig. 2(d). Notes that the partial vector of the first group and the 4^{th} group is useless because the neighbors of each group are the same. Thus, the updated SASI remains unchanged. This iteration of SASI_Calculation is now finished.

TABLE I
All possible pattern sets.

Further Pattern Set	SASIs = (123)(45)	Explanation
A_1	(m_1=0,m_2=0)	$\in \theta_0^5$
A_2	(m_1=0,m_2=1)	$\in \theta_1^5$
A_3	(m_1=0,m_2=2)	$\in \theta_2^5$
A_4	(m_1=1,m_2=0)	$\in \theta_1^5$
A_5	**(m_1=1,m_2=1)**	$\in \theta_2^5$
A_6	**(m_1=1,m_2=2)**	$\in \theta_3^5$
A_7	**(m_1=2,m_2=0)**	$\in \theta_2^5$
A_8	**(m_1=2,m_2=1)**	$\in \theta_3^5$
A_9	(m_1=2,m_2=2)	$\in \theta_4^5$
A_{10}	(m_1=3,m_2=0)	$\in \theta_2^5$
A_{11}	(m_1=3,m_2=1)	$\in \theta_4^5$
A_{12}	(m_1=3,m_2=2)	$\in \theta_5^5$

TABLE II
θ_2^5 pattern sets.

Generated Pattern Set	SASIs=(123)(45)	Outputs
A5	10010	a1
	01010	a2
	00110	a1
	10001	a3
	01001	a3
	00101	a3
A7	11000	b1
	10100	b2
	01100	b2

4. Experimental results

The GA-based algorithm is implemented in Programming Language Interface (PLI) environment. Experiments are conducted over a set of ISCAS-85 and some MCNC benchmarks. The benchmarks are described in Verilog HDL format. The first four columns show the parameters of each benchmark. The remaining columns show the sets of inputs that cannot be distinguished. In fact, these input sets are possibly symmetric inputs sets. They are expressed by pairs (*size*, number of sets), where *size* is the size of an input set and the following is the number of sets of that size.

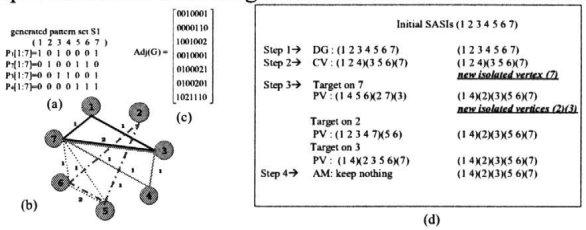

Figure 2. SASIs_Calculation with GA technique.

The iteration bound is set to 100. The CPU time is measured in second on a SUN Sparc II workstation. The algorithm will be terminated automatically if iterations are over the bound or all inputs are nonsymmetric, and SASIs are returned. According to Table III, we can find that in previous approach, c499, c1355, and c1908 still have potentially symmetric input sets, but our approach can distinguish each input as a nonsymmetric input. Table IV also shows the results on some MCNC benchmarks. Most nonsymmetric inputs are distinguished efficiently as usual for most benchmarks. These results demonstrate that our approach acts as a good filter to identify nonsymmetric inputs as many as possible in a circuit such that significant efforts can be saved for other succeeding applications.

5. Conclusions

We propose a pattern generation algorithm to generate pattern sets and a GA-based technique to improve the SASIs_Calculation stage in ACSP. The combination of these two algorithms efficiently distinguishes more nonsymmetric inputs and therefore accelerates the computation of maximal sets of symmetric inputs.

Table III
Comparisons on experimental results.

Circuit	Parameters			Previous approach in [16]		GA approach	
	|PI|	|PO|	Lits.	(size, number of sets)	Time (s)	(size, number of sets)	Time (s)
c17	5	2	12	(1,5)	0.04	(1,5)	0.23
c880	60	26	703	(1,54)(2,3)	2.86	(1,54)(2,3)	1.74
c1355	41	32	1032	(1,32)(9,1)	3.16	(1,41)	0.91
c1908	33	25	1497	(1,22)(5,1)(6,1)	1.81	(1,33)	0.93
c432	36	7	382	(1,36)	0.24	(1,36)	0.41
c499	41	32	616	(1,32)(9,1)	1.07	(1,41)	0.72
c3540	50	22	2934	(1,50)	19.52	(1,50)	10.70
c5315	178	123	4369	(1,178)	33.95	(1,178)	33.42
c2670	233	140	2043	(1,221)(2,2)(8,1)	59.95	(1,223)(2,2)(6,1)	42.55
c7552	207	108	6098	(1,166)(2,6)(3,1)(5,2)(4,4)	5514	(1,183)(2,8)(3,1)(5,1)	191.33
c6288	32	32	4800	(1,32)	6.53	(2,16)	2.84

Table IV
Results of some MCNC benchmarks.

Circuit	|PI|	|PO|	Lits.	(size, number of sets)	Time(s)
9symml	9	1	277	(9,1)	0.95
b1	3	4	17	(1,1)(2,1)	0.24
b9	41	21	236	(1,31)(2,5)	3.88
cm138a	6	8	35	(1,4)(2,1)	0.29
cm162a	14	5	58	(1,12)(2,1)	0.37
cm163a	16	5	53	(12,1)(4,1)	0.36
cm82a	5	3	26	(1,3)(2,1)	0.39
cmb	16	4	62	(4,2)(8,1)	0.98
count	35	16	174	(1,33)(2,1)	0.58
frg1	28	3	130	(1,26)(2,1)	0.41
lal	26	19	223	(1,16)(2,5)	1.87
pm1	16	13	85	(1,9)(3,1)(4,1)	0.31
term1	34	10	625	(1,32)(2,1)	0.48
x1	51	35	2141	(1,49)(2,1)	20.97
x2	10	7	71	(1,8)(2,1)	0.49
x3	135	99	1816	(1,133)(2,1)	9.37
x4	94	71	1040	(1,92)(2,1)	5.14
z4ml	7	4	77	(1,2)(2,2)(3,1)	0.77
alu4	14	8	1278	(1,14)	0.51
apex6	135	99	904	(1,135)	12.88
des	256	245	7412	(1,256)	5.47
i5	133	66	556	(1,133)	10.65
i6	138	67	1037	(1,138)	10.12
i7	199	67	1311	(1,199)	12.77
i8	133	81	4626	(1,133)	13.65
i9	88	63	1453	(1,88)	1.12
pair	173	137	2667	(1,173)	21.38
rot	135	107	1424	(1,115)(2,5)(3,2)(4,1)	17.4

6. References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computer*, Aug. 1986, pp. 677-691.

[2] D. I. Cheng and M. Marek-Sadowska, "Verifying equivalence of functions with unknown input correspondence," in *Proceedings of European Design Automation Conference (EDAC)*, 1993, pp.81-85.

[3] P. Y. Chung and I. N. Hajj, "ACCORD: Automatic catching and correction of logic design errors in combinational circuits," in *Proceedings of International Test Conference*, 1992, pp. 742-751.

[4] S. Devadas, H. K. T. Ma, and A. R. Newton, "On the verification of sequential machinesat differing levels of abstraction," *IEEE Transactions on Computer-Aided Design*, June 1988, pp. 713-722.

[5] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham, "Probabilistic design verification," in *Proceedings of International Conference Computer-Aided Design*, Nov. 1991, pp. 468-471.

[6] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in *Proceedings of Design Automation Conference*, 1987, pp. 341-347.

[7] S. Y. Kuo, "Locating logic design errors via test generation and don't-care propagation," in *Proceedings of European Design Automation Conference (EDAC)*, Sep. 1992, pp. 466-471.

[8] H. T. Liaw, J. H. Tsaih, and C. S. Line, "Efficient automatic diagnosis of digital circuits," in *Proceedings of International Conference Computer-Aided Design*, Nov. 1990, pp. 464-467.

[9] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," in *Proceedings of International Conference Computer-Aided Design*, Nov. 1989, pp. 30-33.

[10] F. Maruyama and M. Fujita, "Hardware verification," *IEEE Computer*, Feb. 1985, pp. 22-32.

[11] E. J. McCluskey, "Detection of group invariance or total symmetry of a Boolean function," *Bell System Tech, J.*, Nov. 1956, pp. 1445-1453.

[12] E. J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, 1986.

[13] Alan Mishchenko, "Fast computation of symmetries in Boolean functions" *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, No. 11, Nov. 2003, pp. 1588-1593.

[14] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," in *Proceedings of European Design Automation Conference (EDAC)*, 1993, pp. 86-92.

[15] D. Moller, J. Mohnke, and M. Weber, "Detection of symmetry of Boolean functions represented by ROBDDs," in *Proceedings of International Conference Computer-Aided Design*, Nov. 1993, pp. 608-684.

[16] I. Pomeranz and S.M. Reddy, "On determining symmetries in inputs of logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 13, No. 11, Nov. 1994, pp. 1428-1434.

[17] U. Schlichtmann, F. Brglez., and M. Hermann, "Characterization of Boolean functions for rapid matching in EPGA technology mapping," in *Proceedings of Design Automation Conference*, June 1992, pp. 347-379.