An Improved AVPG Algorithm for SoC Design Verification Using Port Order Fault Model

Chun-Yao Wang, Shing-Wu Tung, and Jing-Yang Jou
*Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
{wcyao,swtung,jyjou}@eda.ee.nctu.edu.tw

Abstract

Embedded cores are being increasingly used in the design of large System-on-a-Chip (SoC). Because of the high complexity of SoC, the design verification is a challenge for system integrator. To reduce the verification complexity, the port order fault (POF) model proposed in [1] has been used for verifying core-based designs and the corresponding verification pattern generation have been developed [2] [3]. Here we present an automorphic technique to improve the efficiency of the automatic verification pattern generation (AVPG) proposed in [3] for SoC design verification based on POF model. On average, the size of pattern sets obtained on the ISCAS-85 and MCNC benchmarks are 45% smaller and the run time decreases 16% as compared with the results of AVPG.

1. Introduction

Spurred by process technology leading to the availability of more than 1 million gates per chip, and more stringent requirements upon time-to-market and performance constraints, system level integration and platform-based design [4] are evolving as a new paradigm in system designs. A multitude of components that are needed to implement the required functionality make it hard for a company to design and manufacture an entire system in time and within reasonable cost. Hence, design reuse and reusable building blocks (cores) trading shall now be considered a necessity. However, present design methodologies are not enough to deal with cores which come from different design groups and are mixed and matched to create a new system design. In particular, verifying whether a design satisfies all requirements is one of the most difficult task.

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

The focus of core-based design verification should be on how the cores communicate with each other [5]. By creating the testbenches at a high level, a connectivity-based design fault model, **p**ort **o**rder **f**ault (**POF**), proposed in [1] is used for reducing the time on core-based design verification [2] [3].

In [2], Tung et al. had proposed a verification pattern generation algorithm based on POF model. The algorithm generates the verification pattern set only for detecting the simple POF (SPOF) (two ports misplaced at a time).

In [3], we had proposed an **a**utomatic **v**erification **p**attern **g**eneration (AVPG) based on POF model. The AVPG considers **all possible misplacements** among the ports of the cores rather than two ports misplacements. However, the approach of determining the **u**ndetected **p**ort sequences (UPSs) in the AVPG is deficient. **It does not eliminate all detected port sequences from the fault set and generates redundant patterns for some detected port sequences.**

In this paper, we present an automorphic technique, the **superset of all automorphisms** (**SAA**) technique, to calculate the remaining UPSs during the AVPG. This technique accelerates the AVPG and reduces the size of verification pattern set.

2. Preliminary

The POF model assumes a faulty cell has at least two I/O ports misplaced. It also assumes the components are fault free and only the interconnection among the components could be faulty. There are three types of POFs [1].

Definition 1: The type I POF is at least an output misplaced with an input. The type II POF is at least two inputs misplaced. The type III POF is at least two outputs misplaced.

Example 2.1: A fault free 4-bit adder is shown in Fig. 1(a). The function of the adder is $\{Cout, S(3:0)\} = A(3:0) + B(3:0) + Cin$. An example of the type I POF is shown in Fig. 1(b). Input $B0$ is misplaced with output $S0$. Fig. 1(c)

1081-7735/01 $10.00 © 2001 IEEE

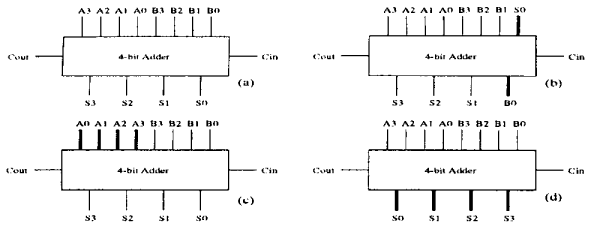

Figure 1. POF model

shows an example of the type II POF. Input $A(3:0)$ are misplaced. Fig. 1(d) shows an example of the type III POF. Output $S(3:0)$ are misplaced.

It has been proven that the type II POFs **dominate** the other two types of POFs [1]. Hence, in this paper, the AVPG focuses on the **type II POFs** solely.

Definition 2:A port sequence is an input port numbers permutation that indicates the relative positions among these input ports. The **fault free port sequence** is a port sequence that none of the input ports were misplaced.

3. Integration verification

Fig. 2 depicts a generic verification scheme for the core-based system chip. Since these cores, BLK1 $\sim$ 5, are pre-verified, the verification efforts during the integration phase should focus on the interconnection among them. To verify these interconnection, designers apply patterns T to primary inputs (PIs) of the design, then compare the responses R to the expected results in primary outputs (POs). If R are inconsistent with the expected ones, some interconnection are misplaced. The generation of T depends on the functionalities of BLK1 $\sim$ 5. As the complexity of cores increase or more cores are involved in the integration, T become harder to generate.

To conquer this problem, we exploit the technique of design for testability (DFT) to conduct verification. The solution is **IEEE P1500 standard for embedded core test** (SECT) [6]. IEEE P1500 SECT is a standard under developed that aims at improving ease of reuse and facilitating interoperability with respect to the test of core-based chips. The most important component in this standard is the **P1500**

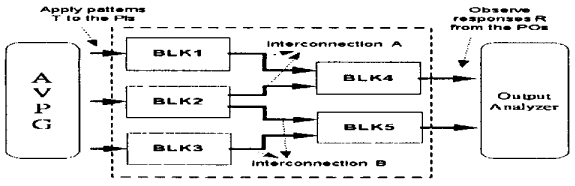

Figure 2. Generic verification scheme

wrapper. It is a thin shell around the core that provides the switching capability between the core and its various access mechanisms. P1500 establishes the mechanism that the test patterns of any circuits under test (CUT) given by core providers can be applied to PIs of the system chip (source) and propagated to POs of the system chip (sink) via user defined test access mechanisms (TAMs).

For the cores to be integrated into a system, they are sorted in topological order from PIs to POs. The entire system is integrated block-wise and follows the topological order. As a block is added into the system, the verification patterns for the block are generated for verification.

We exploit P1500 wrappers and user defined TAMs to propagate the verification patterns from PIs to the wrappers in the predecessors of the core under verification (CUV) and to propagate responses of the CUV to POs. The P1500 wrapper has proposed with a few pre-defined operations , such as core-internal test, core-external test, bypass, isolation and normal modes.

To verify the interconnection among the CUV and its predecessors, the CUV is set in normal mode which allows the CUV functioning in its normal system operation. The predecessors **connected to the CUV directly** are set in external test mode which allow verifying the interconnected wiring between cores via the **ordinary input/output ports** in the core wrappers. The other predecessors of the CUV are all set in bypass mode which allow the stimuli being bypassed through these predecessors to the CUV.

For example, assume the topological order of the embedded cores are BLK1 $\sim$ 5 as shown in Fig. 2. In the beginning, the BLK1 $\sim$ 3 are added into the system. Since these blocks do not have any predecessors, it is not necessary to conduct the POF verification. As the BLK4 is added into the system, the BLK1 $\sim$ 2 are the predecessors that directly connected to it. To verify the interconnection A among these blocks, the BLK4 is set in normal mode, and the BLK1 $\sim$ 2 are set in external test mode to propagate the POF stimuli from PIs through the wrappers (of BLK1 and BLK2) to the inputs of the BLK4 as shown in Fig. 3(a). If there are any misplacements in the interconnection A, the inconsistent results will be observed in the output analyzer. Similarly, as the BLK5 is added into the system, it is set in normal mode. The BLK2 $\sim$ 3 are set in external test mode as shown in Fig. 3(b).

This mechanism allows AVPG solely focusing on the functionality of the added block when generating the verification pattern set and reduce the complexity of POF verification. Furthermore, by using the P1500 test structure for POF verification, no more hardware overhead is introduced in the chip implementation. The mechanism reuses the hardware overhead incurred in the testing phase.

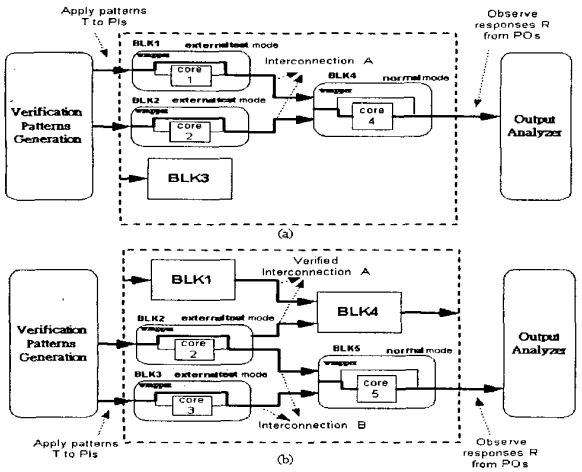

Figure 3. POF verification when integrating the BLK4 and BLK5

4. The SAA technique in the AVPG

The AVPG [3] reads the combinational core and generates heuristic patterns. The patterns simulation results determine the valid verification patterns and the remaining **undetected port sequences (UPSs)** so that more verification patterns can be generated accordingly. When the fault coverage (F_C) reaches 100% or the iterations are over the bound, the AVPG will be terminated. The detailed descriptions of these procedures, pattern generation in particular, in the AVPG can be found in [3]. Here only some important ideas are presented.

The remaining UPSs in each iteration guide the further pattern generation. **If they are not precise enough, some of the further verification patterns could be redundant and the processing time to reach the desired fault coverage will increase.** In [3], the **characteristic vector (CV)** approach of determining the remaining UPSs had encountered this weakness. Thus, the **superset of all automorphisms (SAA)** technique is proposed to improve this procedure.

4.1. UPS representation

For the POF-based AVPG, the fault list is not enumerated explicitly, this is because the total number of POFs in an n-input core is $(n!-1)$. This number grows rapidly when n increases, for instance, as n=69, $n!-1 \approx 1.7\times10^{98}$. Instead, an **implicit** representation is used to indicate the remaining UPSs during the verification pattern generation.

Example 4.1: Given an 8-input core, the input ports are numbered from 1 to 8. The UPSs (12345678) represent the UPSs that caused by all possible misplacements of the port numbers 1~8. The UPSs (125)(4)(3678) indicate the UPSs that caused by all possible misplacements of the port numbers 1, 2 and 5 and/or all possible misplacements of the port numbers 3,6,7 and 8. The number of the undetected POFs is $3!\times1!\times4!-1$. The UPS (1)(2)(3)(4)(5)(6)(7)(8) represents that 8!-1 POFs are all detected and the remaining UPS is empty. If the UPSs are induced from (12345678) to (1)(2)(3)(4)(5)(6)(7)(8), all POFs are detected.

4.2. The comparison of the CV approach and the SAA approach

Definition 3: Given a set of patterns S with the same length, we count the number of digits 1 in the same bit position to form a vector with the same length. This vector is called the **characteristic vector(CV)** of S and is denoted as CV_S.

Definition 4: Given two pattern sets S and T, if the patterns in the S and T are all identical, we said S = T; otherwise S $\neq$ T. If the corresponding bits in the CV_S and CV_T are all the same, we said CV_S = CV_T; otherwise CV_S $\neq$ CV_T.

Lemma 1. *One pattern set has only one CV.*

Lemma 2. *Given two pattern sets S and T, if CV_S $\neq$ CV_T, then S $\neq$ T.*

In [3], the pattern generation stage generates all patterns with the same number of digits 1s and digits 0s, then the patterns with the same outputs are grouped into the same set. Thereafter, the CV of each pattern set is calculated and the valid verification patterns and the remaining UPSs are determined by the CV.

For example, assume the original UPSs are (1234567), all $\binom{7}{3}$ "three 1s patterns" are generated and simulated (assume all $\binom{7}{1}$ "one 1 patterns" and $\binom{7}{2}$ "two 1s patterns" have been simulated and known ineffectual in reducing UPSs), and assume these patterns can be grouped into two sets S1 and S2 only, where **patterns in the same set have the same output**. Fig. 4(a) shows one of the pattern sets, S1, and its CV, CV_S1[1:7]=1121223.

Lemma 3. *Assume a pattern P has m 1s and (n-m) 0s. If it turns to P' after any port misplacements π, the pattern P' still has m 1s and (n-m) 0s.*

Theorem 1: *Assume a pattern set S1 consists of all patterns with the same output and each pattern in the S1 has the same number of digits 1s and 0s. If S1 turns to another pattern set S1' after the port misplacements π and the CV_S1 $\neq$ CV_S1', then the port misplacements π will be detected by S1 patterns.*

According to Theorem 1, the port misplacements that change CV_S1 will be detected by S1 patterns. Consequently, **the port misplacements which cannot change**

the CV_S1 are regarded as the remaining UPSs. Hence, the AVPG figures out the CV_S1[1:7] and groups the different numbers in the CV_S1[1:7] into different subgroups. In this example shown in Fig. 4(a), the CV_S1[1:7] is 1121223, it groups the three 1s in one subgroup, three 2s in another subgroup and one 3 in the third subgroup. The grouped results can be represented as (111)(222)(3) and its corresponding UPSs are (124)(356)(7). The port misplacements occurred in one subgroup keep the CV being the same and are regarded as the remaining UPSs. Thus, when S1 is added into the verification pattern set, the remaining UPSs become (124)(356)(7). The AVPG generates further verification patterns according to this remaining UPSs in the next iteration and so on [3].

However, the remaining UPSs derived by using the CV approach are not the real remaining UPSs in some situation. In this example, the real remaining UPSs are (14)(2)(3)(56)(7) after the **exhaustive examination**. Therefore, the **automorphic approach** is proposed to reach the real remaining UPSs **more closely**.

Definition 5: A graph G with n vertices and m edges consists of a vertex set V(G)={V1,$\cdots$, Vn} and an edge set E(G)={E1, $\cdots$, Em}. Each edge consists of two vertices called its endpoints. UV is an edge with endpoints U and V. A graph is undirected if there is no "direction" on the edges. A graph is weighted if there are positive integer weights on the edges. The weight of the edge UV is denoted as W(UV).

Definition 6: An **isomorphism** from graph G to H is a bijection f:V(G) $\rightarrow$ V(H) such that UV$\in$E(G) if and only if f(U)f(V)$\in$E(H). An **automorphism** of graph G is a permutation of V(G) that preserves adjacency.

To solve the problem of calculating the remaining UPSs of the pattern set S1 with n bits, an undirected, weighted graph G(V,E) is constructed, which corresponds to the set S1 with |S1| patterns, P_1 to $P_{|S1|}$. $P_i[j]$ in the S1 denotes the j^{th} bit in P_i where i=1 $\sim$ $|S1|$ and j=1 $\sim$ n. The vertex Vk in G corresponds to the k^{th} input variable/port in the S1. For all patterns P_1 to $P_{|S1|}$ in the S1, when $P_i[k]$=$P_i[k']$=1, an edge VkVk' is added into G and W(VkVk')=1 where (k, k') are all $\binom{n}{2}$ bit pairs. If the edge VkVk' has existed in G, W(VkVk') is increased one.

For the P_1[1:7] in the S1 shown in Fig. 4(a), 1010001, since $P_1[1]$=$P_1[3]$=$P_1[7]$=1, edges V1V3, V1V7 and V3V7 are added into G, respectively, and so on. The constructed graph G is shown in Fig. 4(b). Its corresponding adjacency matrix, Adj(G), is shown in Fig. 4(c).

The problem of calculating the remaining UPSs in the S1 is now transformed to finding automorphisms of G. The effectiveness of this problem transformation is that **the position relation of digits 1s in each pattern in the S1 are transformed to the connectivity relation in G. Finding port misplacements that maintain S1 to be invariant (calculating UPSs) is equivalent to finding automorphisms of G.**

Fig. 4(d) shows an implication chart that is used for identifying all automorphisms of G. Each grid(Vi,Vj) of the implication chart is filled with the conditions to be satisfied such that the exchange of Vi and Vj is an automorphism. Comparing the columns Ci and Cj in the Adj(G) can examine the relationship between Vi and Vj.

There are three steps in completing the implication chart:

Step 1: If the Ci and Cj are identical, an "O" is filled in the grid(Vi,Vj); otherwise go to step 2.

Example 4.2: Fig. 4(e) shows that the 1^{st} and 4^{th} columns of Fig. 4(c) are identical, an "O" is filled in the grid(V1,V4).

Step 2: We focus on **the vertices that have different degrees connected to Vi and Vj only**. If these vertices exist a perfect matching, M(Vx-Vy), that every matched vertex pair, Vx-Vy, has opposite degrees connected to Vi and Vj, respectively, then this matching is filled in the grid(Vi,Vj); otherwise an "X" is filled in the grid(Vi,Vj).

Example 4.3: Fig. 4(f) shows an example that only V5 and V6 are considered when comparing C5 and C6 in the Adj(G). (V5, V6) is a perfect matching that has opposite degrees, therefore, M(V5-V6) is filled in the grid(V5,V6). Fig. 4(g) shows another example that only V3, V5, V6 and V7 are considered. These vertices make two perfect matchings, M(V3-V5, V6-V7) or M(V3-V6, V5-V7). Therefore, these two matchings are filled in the grid(V1,V2). Each perfect matching is the sufficient condition to justify the automorphism of V1 and V2.

Example 4.4: Fig. 4(h) shows an instance that the rounded vertices cannot make a perfect matching, therefore, an "X" is filled in the grid(V1,V3).

The step 1 and step 2 are conducted iteratively until all grids(Vi,Vj) are filled with an "O", an "X" or matchings. Then the **complete implication chart** is obtained, which is shown in Fig. 4(i). At this time, the step 3 is processed.

Step 3: The grid(Vi,Vj) filled with matchings, M(Vx-Vy), has to be examine again. We have to **examine the grid(Vx,Vy) before justifying the grid(Vi,Vj)**. If the grid(Vx,Vy) has been filled with an "X", we mark the grid(Vi,Vj) with an "X", too; otherwise the grid(Vi,Vj) remains unchanged.

Example 4.5: For the grid(V1,V2), it is filled with two matchings, M(V3-V5, V6-V7) and M(V3-V6, V5-V7). Therefore, we examine the grid(V3,V5), grid(V6,V7), grid(V3,V6), and grid(V5,V7) before justifying the grid(V1,V2). From Fig. 4(i), these grids are all filled with "X". Thus, we mark the grid(V1,V2) with "X", too. For the grid(V5,V6), it is filled with a matching, M(V5-V6), this matching is compatible to the grid(V5,V6), therefore, we leave it unchanged in the grid(V5,V6).

After checking all grids(Vi,Vj) which are filled with matchings, the **refined implication chart** is obtained as shown in Fig. 4(j). In the refined implication chart, if

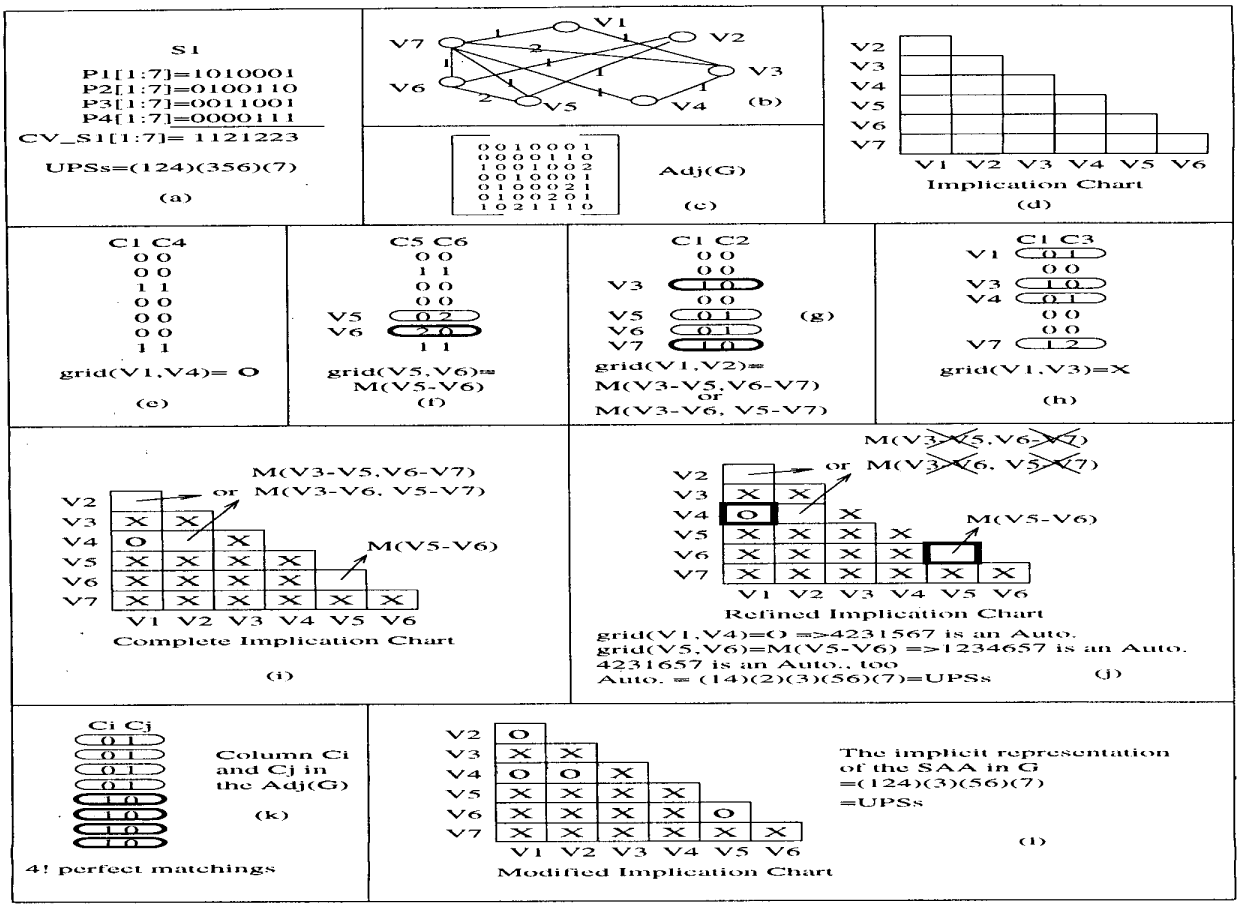

Figure 4. The demonstration of the automorphic technique in calculating the remaining UPSs

the grid(Vi,Vj) is filled with an "O" , the exchange of Vi and Vj is an automorphism. If the grid(Vi,Vj) is filled with a matching, M(Vx-Vy), the combination of exchange of (Vi,Vj) and (Vx,Vy) is an automorphism. All combinations of different automorphisms are automorphisms, too. In Fig. 4(j), the grid(V1,V4) and grid(V5,V6) are **not filled with "X"**, thus all automorphisms of this example are 1234567, **4231**567, 1234**65**7, and **4231657**. They can also be expressed as (14)(2)(3)(56)(7) in our implicit UPSs representation and it is identical to the real remaining UPSs in the previous discussion.

The graph automorphism problem is a well-known and well-studied problem. However, it is not known to be either in P or NP-complete [8] [9]. Therefore, in the proposed graph automorphism algorithm, when we compare the Ci and Cj of the Adj(G) of an n-vertex graph, there may exist $\frac{n}{2}!$ perfect matchings of vertices that satisfy the opposite degrees requirement. In the worst case as shown in Fig. 4(k) with an example of n=8, it takes 4! operations before justifying the grid(Vi,Vj). This number is factorial to n and grows fast when n increases.

To conquer this dilemma, we confine the problem to finding the **superset of all automorphisms** (**SAA**) of G instead of all automorphisms of G. In Fig. 4(l), if an "O" is filled in the grid(Vi,Vj) directly instead of a matching in Fig. 4(i), then the **modified implication chart** is obtained as shown in Fig. 4(l). The modified implication chart can be constructed in **polynomial** time. Thereafter, the same method is applied to acquire the results of automorphisms and they are the SAA of G. In Fig. 4(l), the implicit representation of the SAA in G is (124)(3)(56)(7) and is regarded as the remaining UPSs.

In this example, the remaining UPSs obtained from the CV approach are (124)(356)(7), from SAA approach are (124)(3)(56)(7), and the real remaining UPSs are (14)(2)(3)(56)(7). These results demonstrate that the SAA approach gets more precise remaining UPSs than the CV approach does. Fig. 5 shows the hierarchical relation among

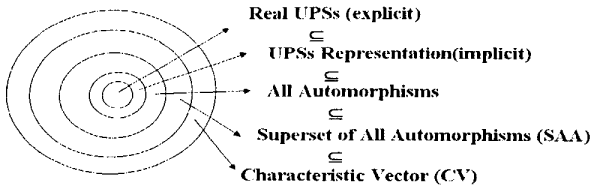

Figure 5. Hierarchical relation among the different approaches

the different approaches. The 1^{st} level represents the set of real remaining UPSs **explicitly**. The 2^{nd} level is the **implicit** UPSs representation of the 1^{st} level. The 3^{rd} level represents the UPSs that obtained by the **automorphism approach**. The **SAA approach** and **CV approach** are shown in the 4^{th} and 5^{th} levels, respectively. The UPSs in the inner levels are the subset of the outer levels. Hence, the approaches closer to the inner levels are more precise. Nevertheless, this figure only indicates the relative relation among them. In an extreme situation, these five levels could be overlapped completely.

Table 1. Experimental results

bench	parameters			CV approach			SAA approach		
	|PI|	|PO|	lits.	pats.	F_C	time(s.)	pats.	F_C	time(s.)
c17	5	2	12	5	100	< 1	5	100	< 1
c880	60	26	703	243	99.99	70	130	99.99	73
c1355	41	32	1032	64	100	13.4	51	100	25
c1908	33	25	1497	51	100	42	45	100	30
c432	36	7	372	38	100	4.6	35	100	2
c499	41	32	616	33	100	8.3	40	100	7
c3540	50	22	2934	145	100	727	89	100	301
c5315	178	123	4369	371	100	931	222	100	530
c2670	233	140	2043	521	99.99	721	351	99.99	666
c7552	207	108	6098	1627	99.99	1826	448	99.99	2604
c6288	32	32	4800	30	99.99	175	30	99.99	170
des	256	245	7412	428	100	159	255	100	91
alu4	14	8	1278	22	100	2.8	17	100	1.6
apex6	135	99	904	234	99.99	406	187	100	100
i9	88	63	1453	139	100	24.7	107	100	13
i8	133	81	4626	266	100	415	204	100	486
i7	199	67	1311	292	100	103	240	100	100
i6	138	67	1037	165	100	77	138	100	25
i5	133	66	556	155	100	63	133	100	20
duke2	22	29	1746	74	100	83.4	21	100	20
rot	135	107	1424	524	99.99	246	247	99.99	74
x1	51	35	2141	275	99.99	34.1	75	99.99	24
x3	135	99	1816	249	99.99	171	165	99.99	155
x4	94	71	1040	352	99.99	69	141	99.99	76
pair	173	137	2667	217	100	443	186	100	102
total	-	-	-	6520	-	6815	3562	-	5697
ratio	-	-	-	1	-	1	**0.546**	-	**0.836**

5. Experimental results

The AVPG that uses the SAA technique to calculate the remaining UPSs has been integrated into SIS [7] environment. Experiments are conducted over a set of ISCAS-85 and MCNC benchmarks. These benchmarks are in BLIF format. Table 1 summaries the experimental results of the AVPG using CV approach [3] and SAA approach to calculate the remaining UPSs, respectively. The first five columns show the parameters of each benchmark. The remaining columns show the number of verification patterns (pats.), fault coverage (F_C) and CPU time (time) measured in second. The fault coverage is defined as 1-$\frac{\#\_of\_undetected\_POFs}{\#\_of\_all\_POFs}$. The iteration bound is set to 100. The CPU time is measured on Ultra Sparc II workstation. The algorithm will be terminated automatically if iterations are over the bound or the fault coverage reaches 100%, and the verification pattern set and the fault coverage are returned. According to Table 1, the size of the pattern set obtained by SAA approach is 45% smaller than that obtained by CV approach on average. Furthermore, the run time also decreases 16% as compared with the previous work. These results are shown in the last row "ratio" in Table 1 and demonstrate that the SAA approach outperforms CV approach in the AVPG.

6. Conclusions

In this paper, we have presented the graph automorphism technique to improve the results of remaining UPSs in each iteration of AVPG. However, due to the high complexity of the graph automorphism technique, we modify this technique to a linear time approach, the SAA approach. The SAA approach gets more precise remaining UPSs, and therefore accelerates the AVPG and generates more efficient verification pattern set for verifying core-based designs.

References

[1] S.-W. Tung, and J.-Y. Jou, "A logic fault model for library coherence checking," *Journal of Information Science and Engineering*, pp.567-586, Sep. 1998.

[2] S.-W. Tung, and J.-Y. Jou, "Verification pattern generation for core-based design using port order fault model," *in Proc. Asian Test Symposium*, pp.402-407, Dec. 1998.

[3] C.-Y. Wang, S.-W. Tung, and J.- Y. Jou, "An AVPG for SoC design verification with port order fault model,", *in Proc. IEEE International Symposium on Circuits And Systems 2001*, pp.V259-V262, May. 2001.

[4] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, "Surviving the SOC revolution - a guide to platform-based design," Norwell, Massachusetts, Kluwer Academic Publishers, 1999.

[5] J. A. Rowson, and A. Sangiovanni-Vincentelli, "Interface-based design," *in Proc. Design Automation Conference*, pp.178-183, Jun. 1997.

[6] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, "Towards a standard for embedded core test:An example," *in Proc. IEEE International Test Conference*, pp.616-627, Sep. 1999.

[7] E. M. Sentovich, K. T. Singh, C. Moon, A. Sangiovanni-Vincentelli et al., "Sequential circuit design using synthesis and optimization," *in Proc. IEEE International Conference on Computer Design*, pp.328-333, Oct. 1992.

[8] M. Agrawal, and V. Arvind, "A note on decision versus search for graph automorphism," *Eleventh Annual IEEE Conference on Computational Complexity*, pp.272-277, 1996.

[9] R. Chang, W. Gasarch, and J. Toran, "On finding the number of graph automorphism," *IEEE Structure in Complexity Theory Conference*, pp.288-298, 1995.