VI. Concluding Remarks

Currently, two main research directions can be identified in low-power testing. The first research direction considers power dissipation during test as a minimization objective. The second direction considers power dissipation as a design constraint, while test application time becomes the minimization objective. This paper presented a link between the two separate research directions, using a new power profile manipulation approach based on the following components: test vector reordering, test sequence expansion, two local peak power approximation model, and test sequence rotation. Test vector reordering is used to lower and reshape the test power profiles. Test sequence expansion further lowers the power profiles of shorter tests which do not affect the total test application time. Then, the proposed two local peak power approximation model translates the power profile it into a simple, reliable, and accurate test power representation, which can be exploited by test sequence rotation in order to increase test concurrency under a power constraint. Since the proposed power profile manipulation is orthogonal to the test scheduling policy and the test set values, the distinctive feature of the proposed solution is that it can be *equally* included in, and consequently leverage the performance of, *any* existing power constrained test scheduling algorithm.

References

[1] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self test—Part 2: Applications," *IEEE Design Test Computers*, vol. 10, pp. 69–77, June 1993.

[2] K. Chakrabarty, "Design of system-on-a-chip test access architectures under place-and-route and power constraints," in *Proc. IEEE/ACM Design Automation Conf. (DAC)*, 2000, pp. 432–437.

[3] R. M. Chou, K. K. Saluja, and V. D. Agrawal, "Scheduling tests for VLSI systems under power constraints," *IEEE Trans. VLSI Syst.*, vol. 5, pp. 175–184, June 1997.

[4] F. Corno, M. Rebaudengo, M. S. Reorda, and M. Violante, "Optimal vector selection for low power BIST," in *IEEE Int. Symp. Defect Fault Tolerance in VLSI Syst.*, 1999, pp. 219–226.

[5] G. L. Craig, C. R. Kime, and K. K. Saluja, "Test scheduling and control for VLSI built-in self-test," *IEEE Trans. Comput.*, vol. 37, pp. 1099–1109, Sept. 1988.

[6] V. Dabholkar, S. Chakravarty, I. Pomeranz, and S. M. Reddy, "Techniques for minimizing power dissipation in scan and combinational circuits during test application," *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, vol. 17, pp. 1325–1333, Dec. 1998.

[7] P. Flores, J. Costa, H. Neto, J. Monteiro, and J. Marques-Silva, "Assignment and reordering of incompletely specified pattern sequences targeting minimum power dissipation," in *12th Int. Conf. VLSI Design*, 1999, pp. 37–41.

[8] P. Girard, "Low power testing of VLSI circuits: Problems and solutions," in *First Int. Symp. Quality Electronic Design (ISQED)*, 2000, pp. 173–180.

[9] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac, "Reduction of power consumption during test application by test vector ordering," *IEE Electron. Lett.*, vol. 33, no. 21, pp. 1752–1754, 1997.

[10] V. Iyengar and K. Chakrabarty, "Precedence-based, preemptive, and power-constrained test scheduling for system-on-a-chip," in *VLSI Test Symp. (VTS)*, 2001, pp. 368–374.

[11] E. Larsson and Z. Peng, "An integrated system-on-a-chip test framework," in *Design Automation Test Conf. Europe (DATE)*, 2001, pp. 138–144.

[12] V. Muresan, X. Wang, V. Muresan, and M. Vladutiu, "A comparison of classical scheduling approaches in power-constrained block-test scheduling," in *Proc. IEEE Int. Test Conf. (ITC 2000)*, 2000, pp. 882–891.

[13] N. Nicolici. (2000, Oct.) Power minimization techniques for testing low power VLSI Circuits. Ph.D. dissertation, Univ. Southampton, U.K.. [Online]. Available: http://www.bib.ecs.soton.ac.uk/records/4937/.

[14] N. Nicolici and B. M. Al-Hashimi, "Power conscious test synthesis and scheduling for BIST RTL data paths," in *Proc. IEEE Int. Test Conf. (ITC 2000)*, 2000, pp. 662–671.

[15] N. Nicolici, B. M. Al-Hashimi, and A. C. Williams, "Minimization of power dissipation during test application in full scan sequential circuits using primary input freezing," *IEE Proc.—Computers and Digital Techniques*, vol. 147, no. 5, pp. 313–322, Sept. 2000.

[16] M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Trans. Design Automation Electron. Systems (TODAES)*, vol. 1, no. 1, pp. 3–56, Jan. 1996.

[17] C. P. Ravikumar, G. Chandra, and A. Verma, "Simultaneous module selection and scheduling for power-constrained testing of core based systems," in *13th Int. Conf. VLSI Design*, 2000, pp. 462–467.

[18] K. Roy and S. Prasad, *Low-Power CMOS VLSI Circuit Design*. New York: Wiley, 2000.

[19] R. R. Oruganti, R. Sankaralingam, and N. A. Touba, "Static compaction to control scan vector power dissipation," in *VLSI Test Symp.*, 2000, pp. 35–40.

[20] H. Schwab. (1997) LP solve. [Online]. Available: http://elib.zib.de/pub/Packages/mathprog/linprog/lp-solve.

[21] "Rethink fault models for submicron-ic test," *Test Measurement World*, Oct. 2001.

[22] S. Wang and S. K. Gupta, "ATPG for heat dissipation minimization during test application," *IEEE Trans. Comput.*, vol. 47, pp. 256–262, Feb. 1998.

[23] Y. Zorian, "A distributed BIST control scheme for complex VLSI devices," in *Proc. 11th IEEE VLSI Test Symp.*, 1993, pp. 4–9.

[24] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded core-based system chips," *Computer*, vol. 32, no. 6, pp. 52–60, June 1999.

An Automorphic Approach to Verification Pattern Generation for SoC Design Verification Using Port-Order Fault Model

Chun-Yao Wang, Shing-Wu Tung, and Jing-Yang Jou

***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 integrators. To reduce the verification complexity, the port-order fault (POF) model was proposed. It has been used for verifying core-based designs and the corresponding verification pattern generation has been developed. Here, the authors present an automorphic technique to improve the efficiency of the automatic verification pattern generation (AVPG) for SoC design verification based on the 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 previous results of AVPG.**

***Index Terms*—Automatic verification pattern generation (AVPG), automorphism, characteristic vector (CV), port-order fault (POF), SoC, superset of all automorphism (SAA), verification.**

I. 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 [1] are evolving as a new paradigm

Manuscript received July 11, 2001; revised December 31, 2001 and April 29, 2002. This work was supported in part by the R.O.C. National Science Council under Grant NSC89-2215-E-009-009. This paper was recommended by Associate Editor T. Cheng.

The authors are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: wcyao@eda.ee.nctu.edu.tw; swtung@eda.ee.nctu.edu.tw; jyjou@eda.ee.nctu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2002.802266.

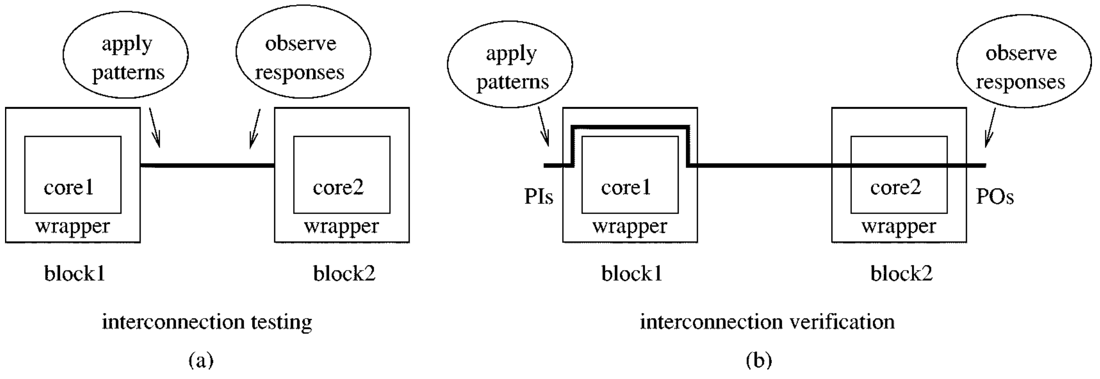

Fig. 1. The schemes of interconnection testing and interconnection verification.

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 are becoming popular in the system-on-a-chip (SoC) era. 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 tasks.

Usage of cores divides the IC design community into two groups: core providers and system integrators. In traditional system-on-board (SoB) design, the components that go from provider to system integrator are ICs, which are designed, verified, manufactured, and tested. The system integrator verifies the design by using these components as fault-free building blocks. SoB verification is limited to detecting faults in the interconnection among the components. Similarly, in SoC design, the components are cores. The system integrator verifies the design by using the cores as design error-free building blocks. The focus of this core-based design verification should be on how the cores communicate with each other [2]. However, before the interface verification, the interconnection between the cores in an SoC have to be verified first. This is because the SoC integrator has to connect a large number of ports (hundreds or even thousands of ports) in an SoC. The likelihood of interconnection misplacements between the cores is high. Furthermore, the correct interconnection between the cores is the minimum requirement to verify the interface protocols. In other words, if the interconnections between the cores are misplaced, the process of the verification on the interface between the cores will be in vain. Thus, the interconnection verification can be conducted as the first step to the interface verification between the cores in a SoC.

Most previous work in testing interconnection focused on the development of deterministic tests for interconnection between chips at the board level [3], [4]. The main purpose is to test if the interconnection are connected properly (neither short nor open). In the interconnection testing phase, the basic assumption for a system under test is that the system design is correct, and the faults are due to manufacturing defects on interconnection among components. For the core-based SoC design verification, however, the system is not fully verified yet and the most of system design errors are due to the incorrect interconnection among predesigned cores. The incorrect interconnections are normally introduced by the misinterpretation of port description of IP cores, and this misinterpretation is usually caused by some factors, such as ambiguous or cryptic port names, Big Endian or Little Endian byte order of address bus, etc. Therefore, the extension of these board level testing methods is inadequate for connectivity-based design verification. Fig. 1(a) and (b) shows the schemes to demonstrate the processes of interconnection testing and interconnection verification, respectively. In the interconnection testing, the test engineers focus on the success of implementation of interconnected wires between block1 and block2. The testing patterns and corresponding responses are applied and observed at the ends of the interconnects to check whether the interconnects are manufactured correctly. On the other hand, in the interconnection verification, the system integrators verify whether the interconnections between block1 and block2 are located in the correct ports. They apply the verification patterns to primary inputs (PIs) of the integrated design, then observe the corresponding responses in primary outputs (POs) of the integrated design, and match them against the specification instead.

By creating the testbenches at a high level, a connectivity-based design fault model, port-order fault (POF), proposed in [5], is used for reducing the time on core-based design verification [6], [7]. In [6], we proposed an automatic verification pattern generation (AVPG) based on the POF model. The AVPG generates a pattern set to detect all possible misplacements among the ports of the cores. However, the approach of determining the undetected port 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 sometimes.

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.

The remainder of this paper is organized as follows: the POF model and some terminology are introduced in Section II. Section III describes the mechanism of conducting POF verification and the verification environment which exploits the IEEE P1500 standard for embedded core test (SECT) [8], [9]. The AVPG flow and the SAA technique are presented in Section IV. Section V presents the experimental results. Section VI concludes the paper.

II. Preliminary

The POF model belongs to the group of pin-faults models [10], which assumes that a faulty cell has at least two I/O ports misplaced. It also assumes that the components are fault free and only the interconnections among the components could be faulty. There are three types of POFs [5].

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.

It has been proven that the type II POFs dominate the other two types of POFs [7]. Hence, in this paper, the AVPG focuses on the type II POFs solely.

Definition 2: A port sequence is an input port number permutation that indicates the relative positions among these input ports.

Definition 3: The fault-free port sequence is a port sequence that none of the input ports were misplaced. For an n-input core, its input

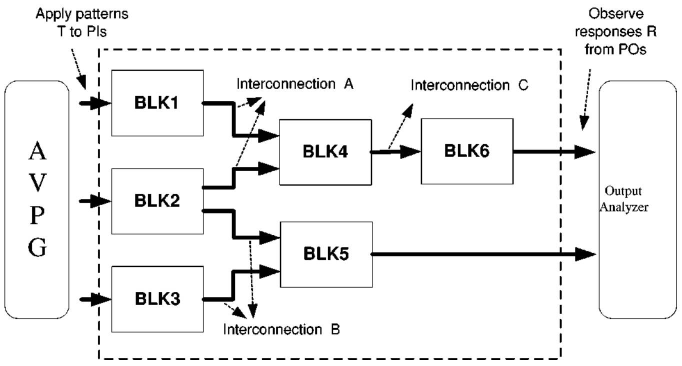

Fig. 2. Generic verification scheme.

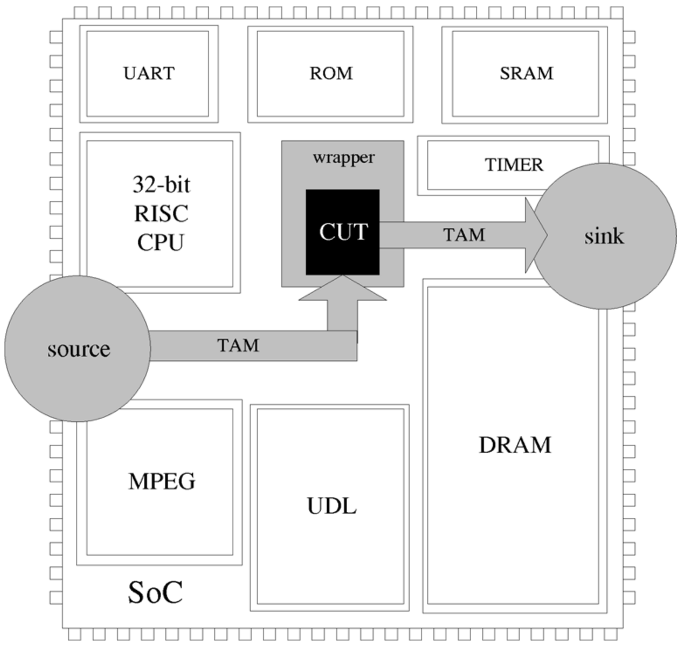

Fig. 3. Generic test access architecture for embedded cores.

variables are numbered from 1 to n. The number of the input variable permutations is $n!$ and these $n!$ permutations represent the $n!$ port sequences of the core. Except the fault-free port sequence, the remaining $(n!-1)$ port sequences represent those corresponding cores with particular POFs and are called faulty port sequences. In this paper, the POFs and the faulty port sequences are used exchangeably.

III. Integration Verification

Fig. 2 depicts a generic verification scheme for the core-based system chip. These cores are lined up sequentially for better illustration. In fact, the interconnection between the cores could be unidirectional or bidirectional according to the required bus structure. During interconnection verification, the cycles in the interconnection are broken up, hence, the interconnected wires between the cores are verified. A typical SoC configuration is as seen in Fig. 3.

Since BLK1~BLK6 in Fig. 2 are preverified, the verification efforts during the integration phase should focus on the interconnection among the cores. To verify the interconnection among the BLK1~BLK6, designers apply the pattern T to PIs of the integrated design, then compare the responses R to the expected results in Ps. If the responses R are inconsistent with the expected ones, some interconnection are misplaced. The generation of the pattern T depends on the functionalities of BLK1~BLK6. As the complexity of cores increase or more cores are involved in the SoC integrated design, the pattern T becomes 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) [8], [9]. IEEE P1500 SECT is a standard under development 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 wrapper. It is a thin shell around the core that provides the switching capability between the core and its various access mechanisms. Fig. 3 depicts a generic access architecture for testing embedded core schematically [11]. The IEEE P1500 SECT 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). This characteristic allows the interconnection verification patterns being propagated to the internal of SoC to verify the interconnection.

The glue logic in the interface between cores is absorbed into the wrapper of cores and is assumed to be preverified. Since the outer boundary of the wrapper is also standardized in the IEEE P1500 standard, this approach can still handle the glue logic in the interface between the cores. The mismatch of the number of primary inputs to the CUT and the number of pins available to the SoC can be handled as well by one mandatory serial interface in the wrapper, which is also standardized in the P1500.

A straightforward core integration methodology is used and the system is integrated block-wise. As a block is added into the system, the verification patterns for the added block are generated from the AVPG and are applied to the integrated system for the interconnection verification.

We exploit P1500 wrappers and user-defined TAMs to propagate the verification patterns from PIs to the wrappers in the predecessor of the added block and to propagate responses of the added block to POs. The P1500 wrapper was proposed with a few predefined operations, such as core-internal test, core-external test, bypass, isolation, and normal modes.

In order to verify the interconnection among the added block and its predecessors, the added block is set in normal mode which allows the added block functioning in its normal system operation. The predecessors connected to the added block 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 added block are all set in bypass mode which allow the stimuli being bypassed through these predecessors to the added block. The bypass mode propagates the stimuli via a single S_i and S_o port directly without involving I/O cells in the wrapper and therefore shortens the pattern propagation time during verification.

For example, assume the BLK1~BLK6 have to be integrated into a system as shown in Fig. 2. In the beginning, the BLK1~BLK3 are added into the system, respectively, and they are set in normal mode. The interconnection verification between them and the AVPG are verified by the verification patterns. As the BLK4 is added into the system, the BLK1 and BLK2 are the predecessors that are directly connected to it. In order to verify the interconnection A among these blocks, the BLK4 is set in normal mode, and the BLK1 and BLK2 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. 4. Hence, the verification patterns can easily go through the system from PIs to POs and verify the interconnection 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 and BLK3 are set in external test mode as shown in Fig. 5. When the patterns are applied into

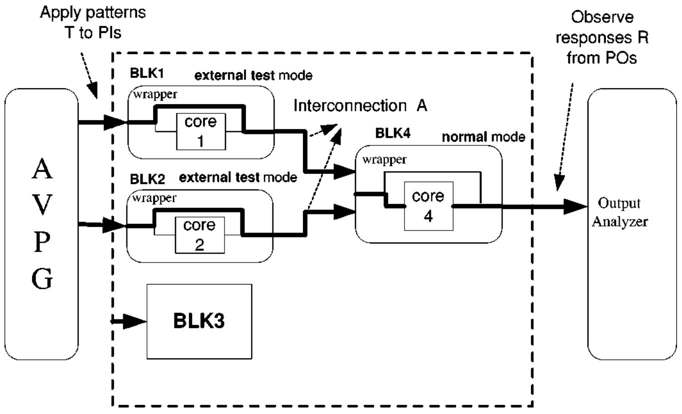

Fig. 4. POF verification when integrating the BLK4.

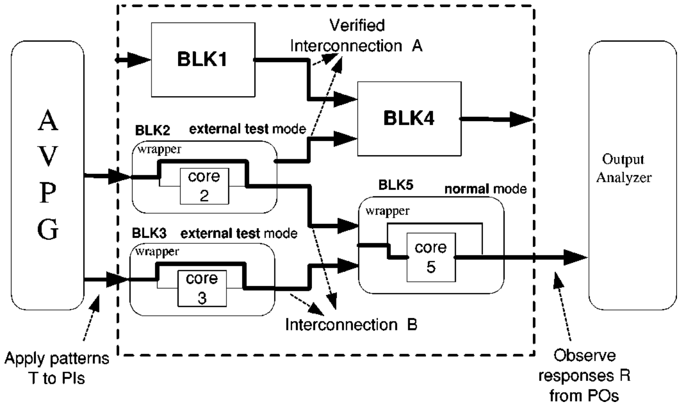

Fig. 5. POF verification when integrating the BLK5.

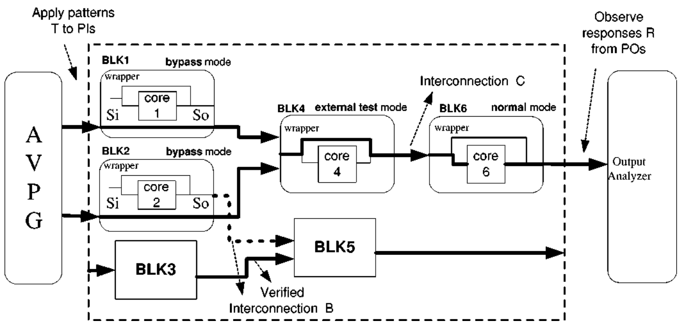

Fig. 6. POF verification when integrating the BLK6.

the system from PIs, the interconnection B is verified, and so forth; as the BLK6 is added into the system, it is set in normal mode. The BLK4 is the predecessor that directly connected to the BLK6, therefore, it is set in external test mode. The BLK1 and BLK2 are the other predecessors of the BLK6, they are set in bypass mode. This operation is shown in Fig. 6 and the interconnection C is verified.

This verification mechanism allows AVPG to focus solely on the functionality of the added block when generating the verification pattern set and reduce the complexity of POF verification. Please note that for verifying the interconnection of an added core, this core is exercised via the normal operation path. This is because only the consistency of simulation results and expected results can guarantee the correctness of integrated design. 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.

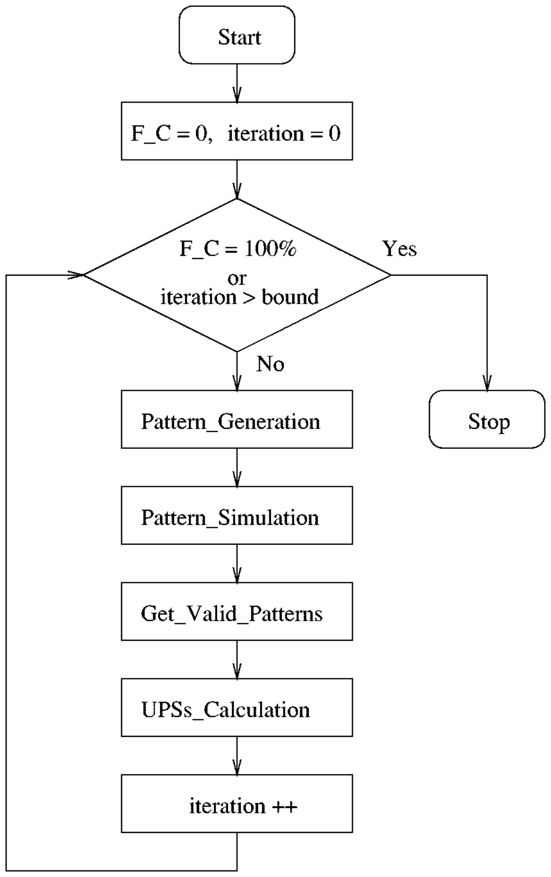

Fig. 7. The flowchart of the POF-based AVPG.

IV. The SAA Technique in the AVPG

The AVPG flow proposed in [6] is shown in Fig. 7. It 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 stages, the pattern generation stage in particular, in the AVPG can be found in [6]. Here, only the AVPG flow and some important ideas are presented.

The UPS's calculation (UPSs_Calculation) procedure shown in Fig. 7 determines what the remaining UPSs are and guides the further pattern generation. If the results of UPS's calculation 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 [6], the characteristic vector (CV) approach of determining the remaining UPSs encountered this weakness. Thus, the superset of all automorphisms (SAA) technique is proposed to improve the UPSs_Calculation procedure so that the AVPG will be more efficient.

A. Undetected Port Sequence (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 $n = 69$, $n! - 1 \approx 1.7 \times 10^{98}$. Instead, an implicit representation is used to indicate the remaining UPSs during the verification pattern generation.

In the beginning, Example 1 demonstrates the implicit UPS representation.

Example 1: Given an eight-input core, the input ports are numbered from 1 to 8. The UPSs (12 345 678) represent the UPS that are caused by all possible misplacements of the port numbers in the same group, i.e., port 1∼port 8. The UPSs (125)(4)(3678) indicate the UPSs that are caused by all possible misplacements among the port numbers 1, 2, and 5 and/or all possible misplacements among the port numbers 3, 6, 7, and 8. The number of the undetected POFs is $3! \times 1! \times 4! - 1$. 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 (12 345 678) to (1)(2)(3)(4)(5)(6)(7)(8), all POFs are detected.

B. The Comparison of the CV Approach and the SAA Approach

In this section, we will describe the CV approach, which exploits the similar technique proposed in [12], and the SAA approach in determining the remaining UPSs and compare their results.

Definition 4: 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 [12]. This vector is called the characteristic vector (CV) of S and is denoted as CV_S.

Definition 5: Given two pattern sets S and T, if $S \subseteq T$ and $S \supseteq T$, 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 ≠ CV_T.

Lemma 1: One pattern set only has one CV.

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

In [6], the pattern generation stage generates all patterns with the same number of 1s and 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 (1 234 567), 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. 8 shows one of the pattern sets, S1, and its CV, $\mathrm{CV\_S1}[1:7] = 1\,121\,223$.

Lemma 3: Assume a pattern P has m 1s and $(n - m)$ 0s. If it turns to P′ after any port misplacements π, then the pattern P′ also has m 1s and $(n - m)$ 0s and is a permutation of P by the port misplacements π.

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 ≠ CV_S1′, then the port misplacements π will be detected by S1 patterns.

Proof: Because CV_S1 ≠ CV_S1′, according to Lemma 2, we assure S1 ≠ S1′. Furthermore, because |S1| = |S1'|, and each pattern in the S1′ is a permutation of the corresponding pattern in the S1, S1′ must contain a pattern $P \notin S1$. According to Lemma 3, the pattern P has the same number of digits 1s and 0s as S1 patterns and because S1 consists of all patterns with the same output, the output of P must be different from that of patterns in the S1. Thus, when applying S1 into the design with the port misplacements π, the real patterns assigned into the design are S1′. The difference between the real outputs and the expected outputs makes the fault effect of the misplacements π propagate and be observed in POs. Q.E.D.

According to Theorem 1, the port misplacements that change CV_S1 will be detected by S1 patterns. Consequently, the port misplacements which cannot change CV_S1 are regarded as the remaining UPSs. Hence, the AVPG figures out CV_S1[1 : 7] and groups the different numbers in CV_S1[1 : 7] into different subgroups. In the example shown in Fig. 8, 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

S1

P_1[1:7]= 1010001
P_2[1:7]= 0100110
P_3[1:7]= 0011001
P_4[1:7]= 0000111

CV_S1[1:7]= 1121223

UPSs=(124)(356)(7)

Fig. 8. The pattern set S1 and the CV_SI.

(111)(222)(3) and its corresponding UPSs are (124)(356)(7). The port misplacements occurred in one subgroup keep the CV 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) as shown in Fig. 8.

The AVPG generates further verification patterns according to this remaining UPS (124)(356)(7) in the next iteration. Here, we describe how to generate additional verification patterns from the UPSs (124)(356)(7) briefly. The UPSs have three groups and we number them from G1 to G3, i.e., G1 is (124), G2 is (356) and G3 is (7). We have known that if we can reduce the UPSs from (124)(356)(7) to (1)(2)(3)(4)(5)(6)(7), the remaining $3! \times 3! \times 1!$-1 faulty port sequences are detected. Our strategy is to attack one group at one iteration. The group Gi with $|\mathrm{Gi}| \geq 2$ is called a possible target group, which can be chosen as the target group in arbitrary order at any iteration. For instance, we choose G1 as the target group first. For the input assignments in G1, we assign $\binom{3}{1}$ "one 1 patterns" to them. For the input assignments in G2 or G3, we heuristically assign values to them and let the assigned values be the same if they are in the same group. The same value assignments in G2 and G3 groups accelerate the calculation of the updated UPSs [6]. Fig. 9 shows such assignments and the corresponding outputs. Since the outputs, A1 and B1, are different, these three patterns are grouped into two sets, S3 and S4, according to the outputs. Then we choose the smaller set, S3, as the verification pattern. From the previous explanation, the grouping result of CV_S3[G1] is the same as the corresponding UPSs on G1 group. Thus, the updated UPSs become (1)(24)(356)(7) and the size of the UPSs is reduced to $1! \times 2! \times 3! \times 1! - 1$. However, if such input assignments cannot differentiate the outputs of these patterns, other assignments are chosen. A more detailed description of the AVPG can be found in [6].

The pattern generation process in the AVPG depends on the remaining UPSs. However, the remaining UPSs derived by using the CV approach are not the real remaining UPSs in some situations. In this example, the real remaining UPSs are (14)(2)(3)(56)(7) after the exhaustive examination when only S1 is added into the verification pattern set [the UPSs derived by the CV approach are (124)(356)(7)]. Actually, the CV approach gets pessimistic results sometimes. Therefore, the automorphic approach is proposed to reach the real remaining UPSs more closely.

Definition 6: A graph G with n vertices and m edges consists of a vertex set $V(G) = \{V1, \ldots, Vn\}$ and an edge set $E(G) = \{E1, \ldots, Em\}$. 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 7: An automorphism of graph G is a permutation of V(G) that preserves adjacency.

Definition 8: A matching in a graph G is a set of pairwise disjoint edges. The vertices belonging to the edges of a matching are saturated by the matching; the others are unsaturated. If a matching saturates every vertex of G, then it is a perfect matching.

G1 124	G2 356	G3 7	outputs
100	**111**	**0**	**A1**
010	111	0	B1
001	111	0	B1

target group G1

	G1 124	G2 356	G3 7
S3	**100**	**111**	**0**

CV_S3[G1]=**100**

	G1 124	G2 356	G3 7
S4	010	111	0
	001	111	0

UPSs[G1]=(1)(24)

UPSs=(1)(24)(356)(7)

Fig. 9. The demonstration of generating additional patterns from the UPSs (124)(356)(7).

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 S1 denotes the jth bit in P_i where $i = 1 \sim |S1|$ and $j = 1 \sim n$. The vertex Vk in G corresponds to the kth input variable/port in S1. For all patterns P_1 to $P_{|S1|}$ in 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.

We use the same example discussed previously to illustrate this transformation. For $P_1[1:7]$ in S1 shown in Fig. 10(a) again, 1010001, since $P_1[1] = P_1[3] = P_1[7] = 1$, edges V1V3, V1V7, and V3V7 are added into G, respectively. For $P_2[1:7]$, 0100110, edges V2V5, V2V6, and V5V6 are added into G, respectively, and so on. The constructed graph G is an undirected weighted graph and is shown in Fig. 10(b). Its corresponding adjacency matrix, Adj(G), is shown in Fig. 10(c). In Adj(G), if there is no edge between Vk and Vk' in G, Adj(G)[k][k'] = Adj(G)[k'][k] = 0; otherwise Adj(G)[k][k'] = Adj(G)[k'][k] = W(VkVk').

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

Fig. 10(d) shows an implication chart that is used for identifying all automorphisms of G. The vertices in G are listed in the X axial and Y axial of the implication chart. Each pair of vertices has a grid entry. Therefore, the total number of grid entries in the implication chart is $\binom{7}{2}$. Each grid entry (Vi, Vj) is filled with the conditions to be satisfied such that the exchange of Vi and Vj is an automorphism. Column Ci in Adj(G) represents the connectivity relation of Vi to the other vertices. Therefore, comparing columns Ci and Cj in Adj(G) can examine the relationship between Vi and Vj.

There are three steps in completing the implication chart. Step 1: If Ci and Cj are identical, this means the connectivity of Vi and Vj to the other vertices in G are identical, then an "O" is filled in the grid entry (Vi,Vj); otherwise, go to step 2.

Example 2: Fig. 10(e) shows that the first and fourth columns of Fig. 10(c) are identical, therefore, an "O" is filled in the grid entry (V1, V4). This "O" means that the exchange of V1 and V4 preserves the adjacency and is an automorphism.

Step 2: If Ci and Cj are not identical, we focus on the vertices that have different degrees connected to Vi and Vj only. If these vertices exhibit a perfect matching, M(Vx–Vy), such that every matched vertex pair, Vx–Vy, has opposite degrees connected to Vi and Vj, respectively, then this matching is filled in the grid entry (Vi, Vj); otherwise an "X" is filled in the grid entry (Vi, Vj).

Example 3: Fig. 10(f) shows an example that only V5 and V6 (rounded) are considered when comparing C5 and C6 in Adj(G). (V5, V6) is a perfect matching that has opposite degrees, therefore, M(V5–V6) is filled in the grid entry (V5, V6). Fig. 10(g) shows another example that only V3, V5, V6, and V7 are considered (they have different degrees connected to V1 and V2). 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 entry (V1,

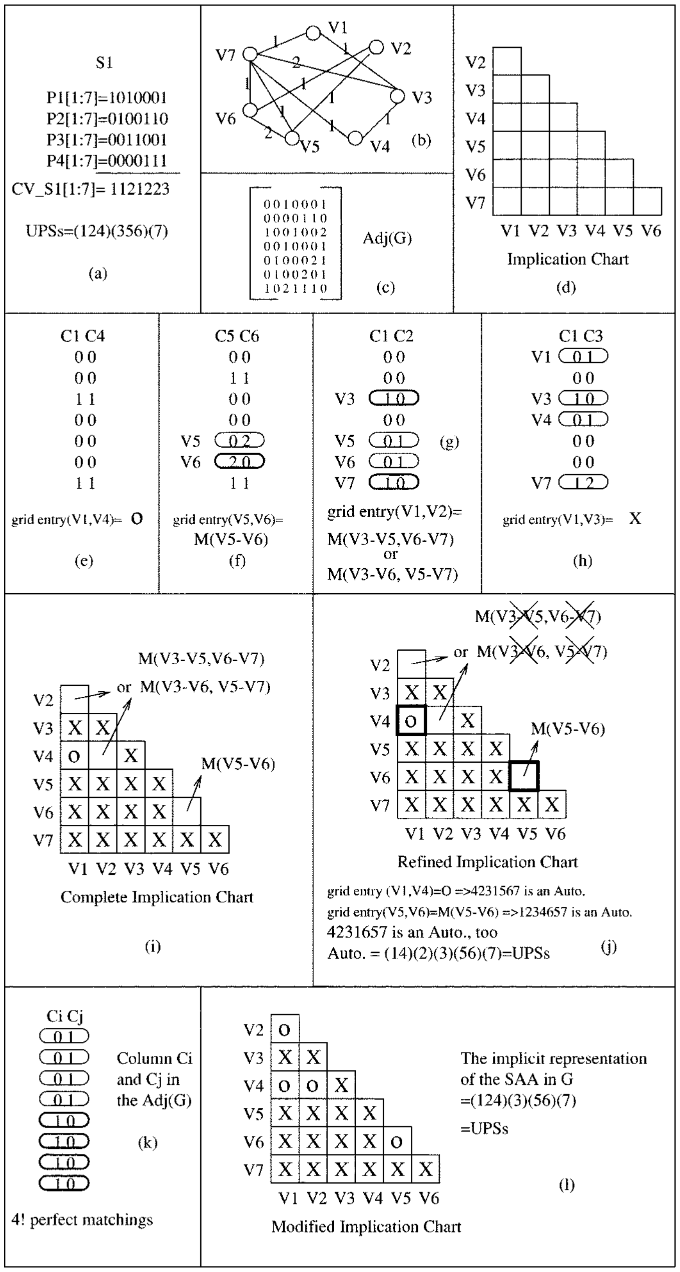
S1
P1[1:7]=1010001
P2[1:7]=0100110
P3[1:7]=0011001
P4[1:7]=0000111
CV_S1[1:7]= 1121223
UPSs=(124)(356)(7)
(a)
V1 V2 V3 V4 V5 V6 V7
(b)
Adj(G)
(c)
Implication Chart
(d)
grid entry(V1,V4)= O
(e)
grid entry(V5,V6)= M(V5-V6)
(f)
grid entry(V1,V2)= M(V3-V5,V6-V7) or M(V3-V6, V5-V7)
(g)
grid entry(V1,V3)= X
(h)
Complete Implication Chart
(i)
Refined Implication Chart
grid entry (V1,V4)=O =>4231567 is an Auto.
grid entry(V5,V6)=M(V5-V6) =>1234657 is an Auto.
4231657 is an Auto., too
Auto. = (14)(2)(3)(56)(7)=UPSs
(j)
Column Ci and Cj in the Adj(G)
(k)
4! perfect matchings
The implicit representation of the SAA in G =(124)(3)(56)(7) =UPSs
Modified Implication Chart
(l)

Fig. 10. The demonstration of the automorphic technique and the SAA approach in calculating the remaining UPSs.

V2). Each perfect matching is the sufficient condition to justify the automorphism of V1 and V2. For example, if both matching pairs V3–V5 and V6–V7 in M(V3–V5, V6–V7) are automorphic, then the exchange of V1 and V2 is an automorphism.

Example 4: Fig. 10(h) shows an instance that the rounded vertices cannot make a perfect matching, therefore, an "X" is filled in the grid entry (V1, V3). This "X" means that the exchange of V1 and V3 is not an automorphism.

The step 1 and step 2 are conducted iteratively until all grid entries (Vi, Vj) are filled with an "O," an "X" or matchings. Then the complete implication chart is obtained, which is shown in Fig. 10(i). At this time, the step 3 is processed.

Step 3: The grid entry (Vi, Vj) that is filled with matchings, M(Vx–Vy), has to be examined again. Because each matching in the grid entry (Vi, Vj) is the sufficient condition to justify the grid entry (Vi, Vj), we have to examine the grid entry (Vx, Vy) before justifying the grid entry (Vi, Vj). If the grid entry (Vx, Vy) has been filled with an "X," we mark the grid entry (Vi, Vj) with an "X," too; otherwise, the grid entry (Vi, Vj) remains unchanged.

Example 5: For the grid entry (V1, V2) and grid entry (V2, V4), they are filled with two matchings, M(V3–V5, V6–V7) and M(V3–V6, V5–V7). Therefore, we examine the grid entry (V3, V5), grid entry (V6, V7), grid entry (V3, V6), and grid entry (V5, V7) before justifying the grid entry (V1, V2) and grid entry (V2, V4). From Fig. 10(i), the grid entry (V3, V5), grid entry (V6, V7), grid entry (V3, V6), and grid entry (V5, V7) are all filled with "X." Thus, we mark the grid entry (V1, V2) and grid entry (V2, V4) with "X," too. For the grid entry (V5, V6), it is filled with a matching, M(V5–V6). This matching is compatible to the grid entry (V5, V6), therefore, we leave it unchanged in the grid entry (V5, V6).

After checking all grid entries (Vi, Vj) which are filled with matchings, the refined implication chart is obtained as shown in Fig. 10(j). In the refined implication chart, if the grid entry (Vi, Vj) is filled with an "O," the exchange of Vi and Vj is an automorphism. If the grid entry (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. 10(j), the grid entry (V1, V4) and grid entry (V5, V6) are not filled with "X," thus all automorphisms of this example are as follows:

1) identity permutation, 1 234 567;
2) exchange of (V1, V4) which corresponds to the vertex permutation, 4 231 567;
3) exchange of (V5, V6) which corresponds to the vertex permutation, 1 234 657;
4) combination of exchanges of (V1, V4) and (V5, V6), which corresponds to the vertex permutation, 4 231 657.

The automorphisms of G 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 [13], [14]. Therefore, in the proposed graph automorphism algorithm, when we compare Ci and Cj of Adj(G) of an n-vertex graph, there may exist $n/2!$ perfect matchings of vertices that satisfy the opposite degrees requirement. In the worst case as shown in Fig. 10(k) with an example of $n = 8$, it takes 4! operations before justifying the grid entry (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. The SAA contains all automorphisms and some nonautomorphisms. In Fig. 10(l), if an "O" is filled in the grid entry (Vi, Vj) directly instead of a matching in which this matching should be filled originally in Fig. 10(i), then the implication chart is obtained as shown in Fig. 10(l) and it is called the modified implication chart. This process avoids examining all matchings in the grid entry (Vi, Vj) of the complete implication chart in completing the refined implication chart, and therefore 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. 10(l), the implicit representation of the SAA in G is (124)(3)(56)(7) and is regarded as the remaining UPSs when the pattern set S1 is included into the verification pattern set.

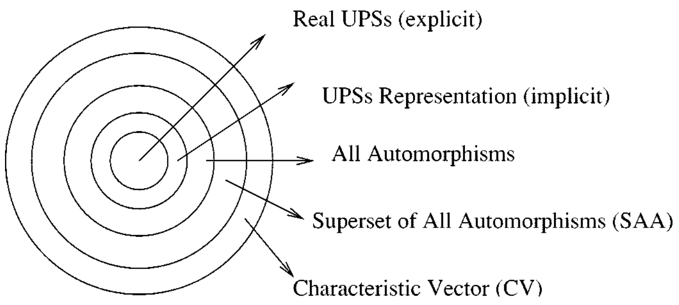
Real UPSs (explicit)
UPSs Representation (implicit)
All Automorphisms
Superset of All Automorphisms (SAA)
Characteristic Vector (CV)

Fig. 11. Hierarchical relation among the different approaches.

In this example, the remaining UPSs obtained from the CV approach are (124)(356)(7). However, it can be further reduced to (124)(3)(56)(7) by the SAA approach 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. The intuition of this fact is that the port misplacements that maintain the pattern set S1 invariant are the real UPSs. The automorphic approach is a way to find these port misplacements, whereas the CV approach finds port misplacements that just maintain the CV invariant.

Fig. 11 shows the hierarchical relation among the different approaches. It has five levels from the center to the boundary. The first level represents the set of real remaining UPSs explicitly. The second level is the implicit UPS representation of the first level. The third level represents the UPSs that are obtained by the automorphism approach. The SAA approach and CV approach are shown in the fourth and fifth 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.

V. Experimental Results

The AVPG that uses the SAA technique to calculate the remaining UPSs has been integrated into an SIS [15] environment, which is developed by the University of California, Berkeley. Experiments are conducted over a set of ISCAS-85 and MCNC benchmarks. The benchmarks are in Berkeley Logic Interchange Format BLIF) format which is a netlist level design description. However, only the simulation information of these benchmarks is needed to conduct the experiments and therefore arbitrary levels of design description can be used for generating verification pattern set. Table I summaries the experimental results of the AVPG using the CV [6] and SAA approaches to calculate the remaining UPSs, respectively. The first five columns show the parameters of each benchmark, including name, |PI|, |PO|, the number of literals (lits.) and the number of POFs. The |PI| represents the number of the inputs and the size of the POF's set is |PI|!-1. The |PO| represents the number of the outputs and influences on the probability of fault effects propagation. The number of literals indicates the complexity of a benchmark. The remaining columns show the number of verification patterns (pats.), fault coverage (F_C), and CPU time (time). The fault coverage is defined as $1 - (\#\_of\_undetected\_POFs/\#\_of\_all\_POFs)$. The iteration bound is set to 100. The CPU time is measured on an Ultra Sparc

TABLE I
EXPERIMENTAL RESULTS

	parameters				CV approach			SAA approach		
bench	|PI|	|PO|	lits.	#_of_POFs	pats.	F_C(%)	time(sec.)	pats.	F_C(%)	time(sec.)
$c17$	5	2	12	5!-1	5	100	< 1	5	100	< 1
$c880$	60	26	703	60!-1	243	99.999999	70	130	99.999999	73
$c1355$	41	32	1032	41!-1	64	100	13.4	51	100	25
$c1908$	33	25	1497	33!-1	51	100	42	45	100	30
$c432$	36	7	372	36!-1	38	100	4.6	35	100	2
$c499$	41	32	616	41!-1	33	100	8.3	40	100	7
$c3540$	50	22	2934	50!-1	145	100	727	89	100	301
$c5315$	178	123	4369	178!-1	371	100	931	222	100	530
$c2670$	233	140	2043	233!-1	521	99.999999	721	351	99.999999	666
$c7552$	207	108	6098	207!-1	1627	99.999999	1826	448	99.999999	2604
$c6288$	32	32	4800	32!-1	30	99.999999	175	30	99.999999	170
des	256	245	7412	256!-1	428	100	159	255	100	91
$alu4$	14	8	1278	14!-1	22	100	2.8	17	100	1.6
$apex6$	135	99	904	135!-1	234	99.999999	406	187	100	100
$i9$	88	63	1453	88!-1	139	100	24.7	107	100	13
$i8$	133	81	4626	133!-1	266	100	415	204	100	486
$i7$	199	67	1311	199!-1	292	100	103	240	100	100
$i6$	138	67	1037	138!-1	165	100	77	138	100	25
$i5$	133	66	556	133!-1	155	100	63	133	100	20
$duke2$	22	29	1746	22!-1	74	100	83.4	21	100	20
rot	135	107	1424	135!-1	524	99.999999	246	247	99.999999	74
$x1$	51	35	2141	51!-1	275	99.999999	34.1	75	99.999999	24
$x3$	135	99	1816	135!-1	249	99.999999	171	165	99.999999	155
$x4$	94	71	1040	94!-1	352	99.999999	69	141	99.999999	76
$pair$	173	137	2667	173!-1	217	100	443	186	100	102
$total$	-	-	-	-	6520	-	6815	3562	-	5697
$ratio$	-	-	-	-	1	-	1	0.546	-	0.836

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. For example, in $c5315$ benchmark, the number of verification patterns is decreased from 371 to 222 and the processing time is reduced from 931 to 530 as well. This is because the more precise remaining UPSs in the AVPG avoid generating redundant verification patterns and reduce the simulation time of further pattern generation. However, the processing time of the SAA approach is not always smaller than that of CV approach. In $c7552$ and $i8$, for example, the processing time is larger in SAA approach. Actually, the overall processing time consists of the time in every stage of the AVPG and it depends on the remaining UPSs calculated in every iteration. If the difference of the remaining UPS's obtained by these two approaches is less, the processing time of SAA approach could be larger due to the higher computation complexity; otherwise, the SAA approach could be faster. According to Table I, the size of the pattern set obtained by SAA approach is 45% smaller than that obtained by the 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 I and demonstrate that the SAA approach outperforms CV approach in the AVPG.

VI. CONCLUSION

In the SoC era, the embedded cores are mixed and integrated to create a system chip. The verification of the core-based system design should be focused on how the cores communicate with each other. However, before the interface verification, the interconnections between the cores in an SoC have to be verified first. System integrators integrate those cores manually and have the possibility of incorrect integration due to the misplaced I/O ports. Therefore, we adopt the connectivity-based POF model to raise the abstraction level of the design verification and to reduce the time on functional verification in core-based design methodology.

In this paper, we have presented the graph automorphism technique to improve the UPSs_Calculation procedure proposed in [6]. 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 a more efficient verification pattern set for verifying core-based designs.

REFERENCES

[1] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution—A Guide to Platform-Based Design*. Norwell, MA: Kluwer, 1999.

[2] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface-based design," in *Proc. Design Automation Conf.*, June 1997, pp. 178–183.

[3] P. Goel and M. T. McMahon, "Electronic chip-in-place test," in *Proc. IEEE Int. Test Conf.*, Oct. 1982, pp. 83–90.

[4] A. Hassan, V. K. Agarwal, B. Nadeau-Dostie, and J. Rajski, "BIST of PCB interconnects using boundary-scan architecture," *IEEE Trans. Computer-Aided Design*, pp. 1278–1288, Oct. 1992.

[5] S.-W. Tung and J.-Y. Jou, "A logic fault model for library coherence checking," *J. Inform. Sci. Eng.*, pp. 567–586, Sept. 1998.

[6] C.-Y. Wang, S.-W. Tung, and J.-Y. Jou, "An AVPG for SoC design verification with port order fault model," in *Proc. IEEE Int. Symp. Circuits Syst. 2001*, May, pp. V259–V262.

[7] ——, "On automatic-verification pattern generation for SoC with port-order fault model," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 466–479, Apr. 2002.

[8] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, "Toward a standard for embedded core test: An example," in *Proc. IEEE Int. Test Conf.*, Sep. 1999, pp. 616–627.

[9] [Online]. Available: http://grouper.ieee.org/groups/1500.

[10] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Digital systems testing and testable design," *Computer Sci.*, p. 95, 1990.

[11] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," in *Proc. IEEE Int. Test Conf.*, Oct. 1998, pp. 130–143.

[12] D. I. Cheng and M. Marek-Sadowska, "Verifying equivalence of functions with unknown input correspondence," in *Proc. Euro. Conf. Design Automation Euro. Event ASIC Design*, Feb. 1993, pp. 81–85.

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

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

[15] E. M. Sentovich, K. T. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1992, pp. 328–333.