On Automatic-Verification Pattern Generation for SoC With Port-Order Fault Model

Chun-Yao Wang, *Student Member, IEEE*, Shing-Wu Tung, *Member, IEEE*, and Jing-Yang Jou, *Senior Member, IEEE*

***Abstract*—Embedded cores are being increasingly used in the designs 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 has been used for verifying core-based designs (Tung and Jou, 1998) . In this paper, we present an automatic-verification pattern generation (AVPG) for SoC design verification based on the POF model and perform experiments on combinational and sequential benchmarks. Experimental results show that our AVPG can efficiently generate verification patterns with high POF coverage.**

***Index Terms*—Automatic-verification pattern generation (AVPG), design verification, IEEE P1500, port-order fault (POF), SoC, undetected port sequence (UPS).**

I. Introduction

SPURRED by process technology leading to the availability of more than one-million gates per chip and more stringent requirements upon time-to-market and performance constraints, system-level integration and platform-based design [2] 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 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.

Verification is a process used to demonstrate the functional correctness of a design. Testing is a process that verifies whether the design was manufactured correctly. Fig. 1 shows the reconvergent paths model for both verification and testing [3]. During testing, the finished silicon is reconciled with the netlist that was submitted for manufacturing. Therefore, when a design is claimed to be fully tested, i.e., 100% fault coverage, under a fault model, such as stuck-at fault (SAF) model, that means it is manufactured correctly. However, we still cannot guarantee that the chip satisfies the design specification if we do not verify it properly before manufacturing. The chip may be a manufactured correctly but designed incorrectly chip. Thus, designers spend about 70% of their efforts on the verification. But design verification is still on the critical path of the design flow [3].

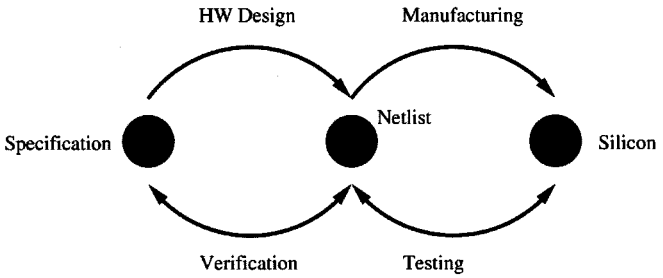


Fig. 1. Reconvergent paths model for both verification and testing.

Usage of cores divides the integrated circuit (IC) design community into two groups: core providers and system integrators. In traditional system-on-board (SoB) design, the components that go from providers to system integrators 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. Based on this assumption, SoC verification could be focused on detecting the misplacements of the interconnection among the cores as the first step. This higher level of abstraction decreases the complexity of design verification on a system chip and reduces the time on design verification of the entire system.

Most previous work in testing interconnection focused on the development of deterministic tests for interconnection between chips at the board level [4], [5]. The extension of these board-level testing methods to core-based design verification is inappropriate. In the interconnection testing phase, only system-level interconnection is tested. The basic assumption for a system under test is based on the system design being correct and the faults are due to manufacturing defects on interconnection among components. For core-based SoC design verification, the system is not fully verified yet and the most of system design errors are due to incorrect designs of interconnection among predesigned cores. The methods proposed to test interconnection do not guarantee that the interconnection among cores is correct.

The focus of core-based design verification should be on how the cores communicate with each other [6]. By creating the testbenches at a higher level, a connectivity-based design fault model, port-order fault (POF), proposed in [7] is used for reducing the time on core-based design verification [1]. In [1],

Manuscript received May 23, 2001; revised November 1, 2001. 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 J. H. Kukula.

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

Publisher Item Identifier S 0278-0070(02)02471-5.

0278-0070/02$17.00 © 2002 IEEE

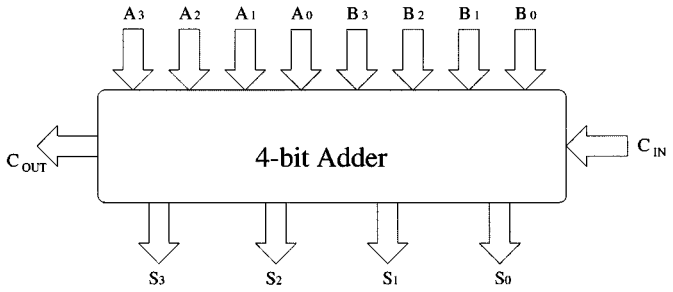


Fig. 2. A fault-free 4-bit adder.

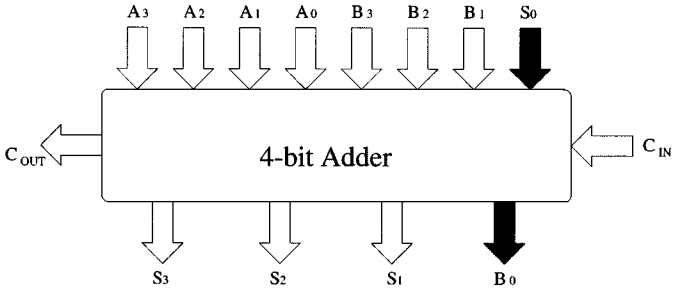


Fig. 3. A 4-bit adder with the type-I POF.

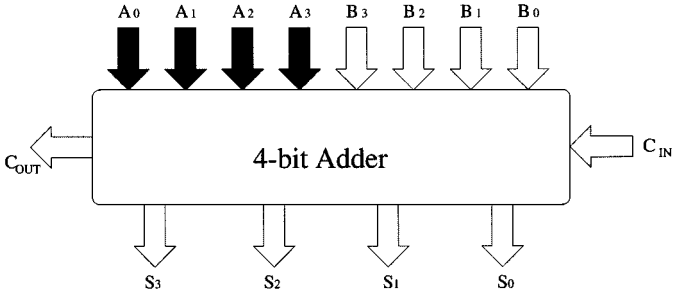


Fig. 4. A 4-bit adder with the type-II POF.

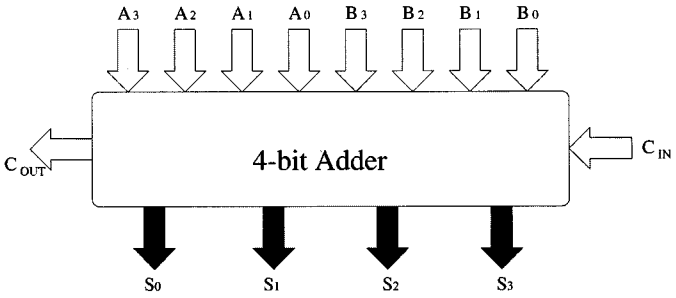


Fig. 5. A 4-bit adder with the type-III POF.

Tung *et al*. proposed a verification pattern generation algorithm based on the POF model. The algorithm cooperates with the SAF automatic test pattern generation to generate the verification pattern set for detecting the simple POF (SPOF) (two ports misplaced at a time). This approach needs the circuit netlist to generate the verification patterns, thus, it loses the capability of integrating different levels of embedded cores (soft cores, firm cores and hard cores) into a system. For example, this algorithm cannot generate verification patterns for soft cores. Furthermore, the simplified SPOF model is not enough to deal with all possible misplacements occurred in a real design during the cores integration phase. In this automatic-verification pattern generation (AVPG), however, all possible misplacements among the ports of cores are considered rather than two ports misplacements. Furthermore, it only uses the simulation information of the core rather than the circuit netlist to generate the verification pattern set. This loosened requirement allows the different levels of cores being integrated together for the POF verification.

The POF AVPG is integrated into the SIS [8] environment. Experiments are conducted on combinational and sequential benchmarks, such as ISCAS-85, ISCAS-89, and MCNC benchmarks. Experimental results show that the AVPG can efficiently generate verification pattern sets with high POF coverage in the proposed verification environment which exploits the IEEE P1500 Standard for embedded core test (SECT) [9].

The remainder of this paper is organized as follows. The POF model and some terminologies are introduced in Section II. Section III describes the mechanism of conducting POF verification. The POF AVPG is presented in Section IV. Section V presents 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 input–output (I/O) ports misplaced. It also assumes that the components are fault free and only the interconnection among the components could be faulty. There are three types of POFs [7].

Definition 1: The type-I POF is at least one input x_k misplaced with one output y_j, i.e., the input ports of faulty core $X' = \{x_1, \ldots, x_i, \ldots, y_j, \ldots, x_n | 1 \leq i \leq n, i \neq k\}$ and output ports of faulty core $Y' = \{y_1, \ldots, y_i, \ldots, x_k, \ldots, y_m | 1 \leq i \leq m, i \neq j\}$.

Example 1: A fault-free 4-bit adder is shown in Fig. 2. The functionality of the adder is $\{C_{OUT}, S(3:0)\} = A(3:0) + B(3:0) + C_{IN}$. An example of the type-I POF is shown in Fig. 3. Input port B_0 is misplaced with output port S_0.

Definition 2: The type-II POF is at least two input ports misplaced, i.e., the input ports of the fault-free core $X = \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n | i \neq j\}$, but the input ports of the faulty core $X' = \{x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n | i \neq j\}$.

Example 2: An example of the type-II POF is shown in Fig. 4. Input ports $A(3:0)$ are misplaced with input ports $A(0:3)$. The order of the input ports $A(3:0)$ is reversed.

Definition 3: The type-III POF is at least two output ports misplaced, i.e., the output ports of the fault-free cell $Y = \{y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m | i \neq j\}$, but the output ports of the faulty cell $Y' = \{y_1, \ldots, y_j, \ldots, y_i, \ldots, y_m | i \neq j\}$.

Example 3: An example of the type-III POF is shown in Fig. 5. Output ports $S(3:0)$ are misplaced with output ports $S(0:3)$. The order of the output ports $S(3:0)$ is reversed.

Because the misplaced output port in the type-I POF has no driving source, it has invariant value (typical high impedance) that could be detected by applying any verification patterns. Therefore, the verification patterns for the type-II POF can also detect the type-I POF.

In Fig. 6, it shows a generic system chip which contains six cores, BLK1 $\sim$ BLK6. The verification on type-III POF of

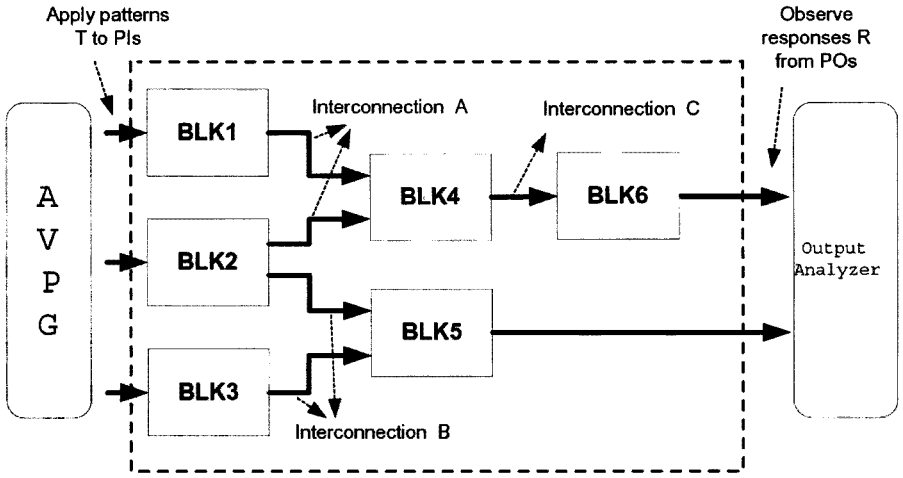


Fig. 6. Generic verification scheme.

BLK1 $\sim$ BLK4 is equivalent to the verification on type-II POF of BLK4 $\sim$ BLK6. Therefore, if the type-II POF of all cores in the system are verified, the type-III POF are also verified simultaneously. Hence, in this paper, the AVPG generates the verification patterns for detecting the type-II POF solely.

Definition 4: A port sequence is an input port numbers permutation that indicates the relative positions among these input ports.

Definition 5: The fault-free port sequence is a port sequence that none of the input ports is misplaced. For an N-input core, the input ports are numbered from 1 to N. The number of the input port numbers permutation 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 the core with some particular POFs and are called faulty port sequences. In this paper, the POFs and the faulty port sequences are used exchangeably.

III. Integration Verification

Using core-based design methodology could reduce the time-to-market for system chips. However, the verification efforts of system chip are still proportional to the design complexity. Both simulation and verification technique cannot reduce the total verification time effectively if those predesigned and preverified blocks are to be verified exhaustively during the integration phase. Therefore, in this section, we introduce the IEEE P1500, which is a standard under development and is used for embedded core testing, to reduce the complexity of design verification. Besides, we will explain how to use the POF verification pattern set for verifying integrated SoC designs.

Fig. 6 depicts a generic verification scheme for the core-based system chip. Since these cores, BLK1 $\sim$ BLK6, are preverified, the verification efforts during the integration phase should focus on the interconnection among the cores. To verify the interconnection among the BLK1 $\sim$ BLK6, designers apply the patterns T to primary inputs (PIs) of the integrated design, then compare the responses R to the expected results in primary outputs (POs). If the responses R are inconsistent with the expected ones, some interconnection are misplaced. The generation of the patterns T depends on the functionalities of BLK1 $\sim$ BLK6. As the complexity of cores increase or more cores are involved in the SoC integration, the patterns T become harder to generate.

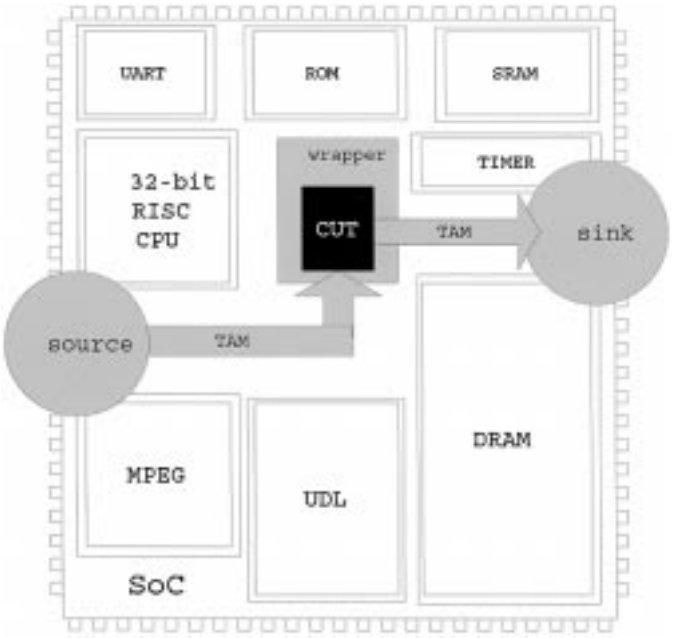


Fig. 7. Generic test access architecture for embedded cores.

To conquer this problem, we exploit the technique of design for testability to conduct verification. The solution is the IEEE P1500 SECT [9], [11]. The 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 IEEE P1500 working group has suggested a module-level boundary-scan structure which is very similar to the IEEE 1149.1 (JTAG) [12] structure. The structure, called "wrapper", is a thin shell around the core that allows intercore and intracore test functions to be carried out via a test access mechanism (TAM). The TAM itself is user defined and is not specified in the draft standard [13]. Fig. 7 depicts a generic access architecture for testing embedded core schematically [11]. Fig. 8 shows a typical configuration of how the chip-level connection of the cores might be connected in one serial TAM and one parallel

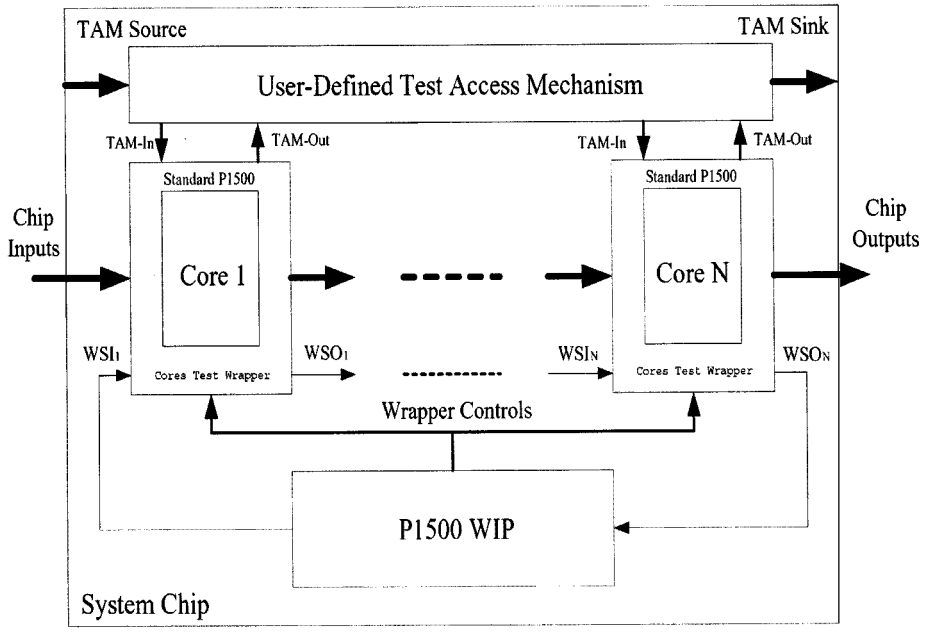


Fig. 8. A typical configuration of the cores integration using the IEEE P1500 architecture.

TAM per core [14]. The IEEE P1500 establishes the mechanism that the test patterns of any circuits under test 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 TAMs.

Most of the POFs occur because of human introduced errors in the SoC design process. These human errors are normally introduced by misinterpretation of the design specification of interconnection. Usually, the IP cores are integrated by system integrators and the test structure among the cores is constructed by test engineers. That is to say, the cores which are connected for normal operation is done by an individual in the core integration phase and the cores which are connected for test mode is done by another individual in the test insertion phase. These two mode connections are independent. Therefore, the likelihood of making the same mistake is very small.

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

We exploit the IEEE P1500 wrappers and user defined TAMs to propagate the verification patterns from PIs to the wrappers in the predecessor of the core under verification (CUV) and to propagate responses of the CUV to POs. The IEEE 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 CUV and its predecessors, the CUV is set in normal mode which allows the CUV to function 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 I/O ports in the core wrappers. The other predecessors of the CUV are all set in bypass mode which allow the stimuli being bypassed through cores to the CUV.

For example, assume the BLK1 $\sim$ BLK6 have to be integrated into a system as shown in Fig. 6. In the beginning, the BLK1 $\sim$ BLK3 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 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. 9. 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. 10. And so forth, as the BLK6 is added into the system, it is set in normal mode. The BLK4 is the predecessor that is directly connected to the BLK6. Hence, 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 is shown in Fig. 11.

This verification mechanism allows us solely focusing on the functionality of the added block when generating the verification pattern set and reduce the complexity of POF verification. Note that for verifying the interconnection of an added core, the core is exercised via the normal operation path. This is because system integrators possibly have misunderstanding about the correct interconnection among cores. Only the consistency of simulation results and expected results can guarantee the correctness of integration.

By using the IEEE P1500 test structure for the POF verification, we do not introduce any more hardware overhead in the

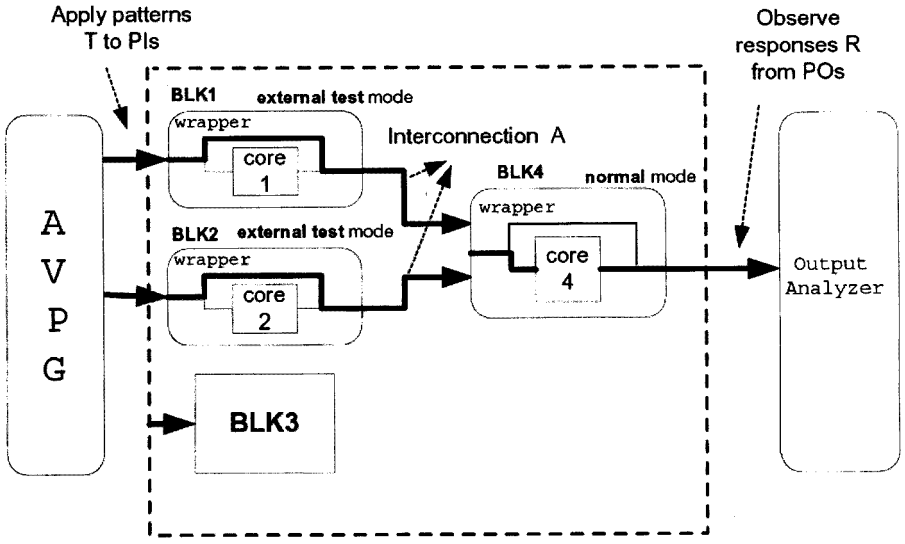


Fig. 9. The POF verification when integrating the BLK4.

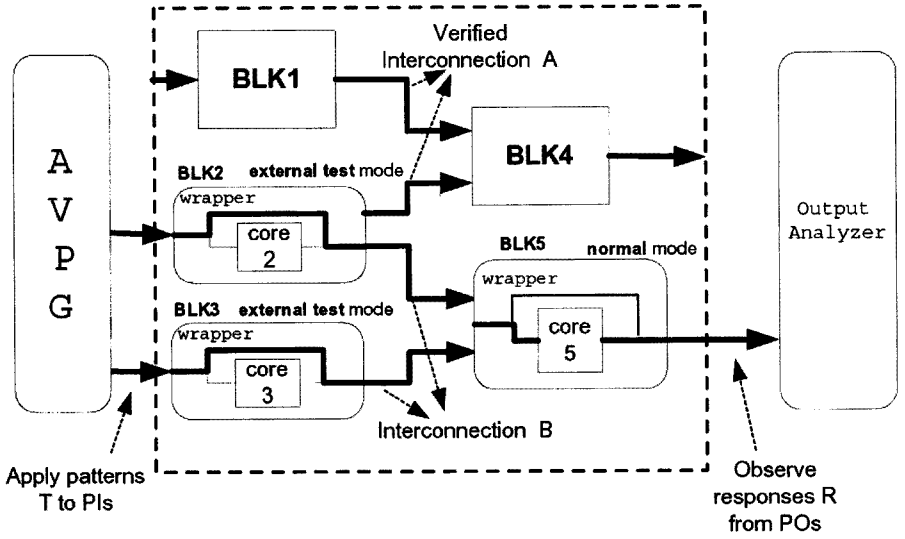


Fig. 10. The POF verification when integrating the BLK5.

chip implementation. In fact, we reuse the hardware overhead incurred in the testing phase.

IV. The POF-Based AVPG

This section describes the AVPG that is shown in the left part of Fig. 11.

A. The Combinational AVPG

1) UPS Representation: Typically, the automatic pattern generator for functional errors, such as transition fault [15] or manufacturing faults, such as SAF [16], builds fault list explicitly first to explore how many faults have to be detected with specific patterns, then generates random patterns and deterministic patterns. For the POF-based AVPG, however, the fault list cannot be 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 $N = 20$, $N!-1 \approx 2.4\times 10^{18}$, as $N = 69$, $N!-1 \approx 1.7\times 10^{98}$. Instead, an implicit representation is used to indicate the remaining undetected port sequences (UPSs) during the verification pattern generation. When the remaining UPS becomes empty, all POFs are detected. In the pattern generation stage, the heuristic patterns are generated instead of random patterns and deterministic patterns. In other words, the approach is to "search" proper patterns for the POF verification in a systematic way.

In the beginning, Example 4 demonstrates the implicit UPSs representation.

Example 4: Given an 8-input core, the input ports are numbered from 1 to 8. The UPS's representation (12 345 678) represents the UPSs that caused by all possible misplacements among the port numbers in the same group, i.e., Port 1 to Port 8. The number of undetected POFs is $8! - 1$ and the one in the $8! - 1$ accounts for the fault-free port sequence. The UPS's representation (125)(4)(3678) indicates the UPSs that 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

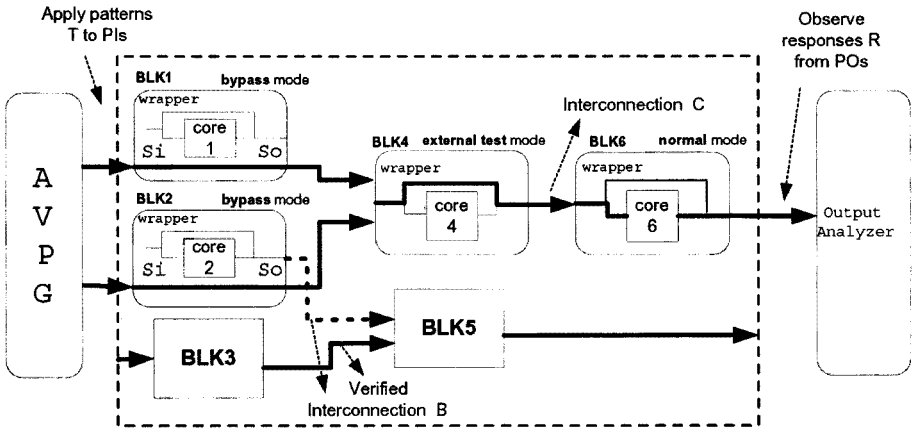


Fig. 11. The POF verification when integrating the BLK6.

POFs is $3! \times 1! \times 4! - 1$. Note that the port number 4 is the only one element in the second group. It means that the port sequences whose port number 4 in the wrong position are not represented by this UPS's representation. The order of the groups in the UPS's representation is irrelevant, neither is the order of the numbers in each UPS's group. For example, the UPSs (125)(4)(3678) can also be expressed as (4)(215)(8763). The UPSs (12)(3)(4)(5)(6)(78) contain four port sequences and they are 12345678, 21345678, 12 345 687, and 21 345 687. However, the UPS's representation of the port sequences 21 345 678 and 12 345 687 is also (12)(3)(4)(5)(6)(78). This is because we always use one UPS's representation to express all UPSs. The UPS representation (1)(2)(3)(4)(5)(6)(7)(8) has eight groups and each group has only one element, therefore, no misplacement could be occurred in each group. The number of the undetected POFs is $1! \times 1! \times 1! \times 1! \times 1! \times 1! \times 1! \times 1! - 1 = 0$. Hence, (1)(2)(3)(4)(5)(6)(7)(8) represents $8! - 1$ POFs are all detected. If the UPS's representation is induced from (12 345 678) to (1)(2)(3)(4)(5)(6)(7)(8), all POFs are detected.

2) The Verification Pattern Generation: This section describes the verification pattern generation algorithm, which is the foundation of our AVPG.

Definition 6: For an N-input combinational core, the exhaustive pattern set is defined as Φ^N. The size of Φ^N is the number of patterns in Φ^N and is denoted as $|\Phi^N|$ and $|\Phi^N|$ equals 2^N.

Definition 7: The set that consists of all patterns with m ones and $(N - m)$ zeroes is denoted as Θ_m^N, where $m \in [0, 1, 2, \ldots, N-1, N]$. The size of Θ_m^N is the number of patterns in Θ_m^N and is denoted as $|\Theta_m^N|$ and $|\Theta_m^N|$ equals $\binom{N}{m}$ where $\binom{N}{m} = N!/(m!(N-m)!)$.

Example 5: For a 4-input core, Φ^4 is the exhaustive pattern set with 4 bits. $|\Phi^4| = 2^4 = 16$. $\Theta_0^4 = \{0000\}$, $|\Theta_0^4| = \binom{4}{0} = 1$. $\Theta_1^4 = \{1000, 0100, 0010, 0001\}$, $|\Theta_1^4| = \binom{4}{1} = 4$. $\Theta_2^4 = \{1100, 1010, 1001, 0110, 0101, 0011\}$, $|\Theta_2^4| = \binom{4}{2} = 6$, $\Theta_3^4 = \{1110, 1101, 1011, 0111\}$, $|\Theta_3^4| = \binom{4}{3} = 4$. $\Theta_4^4 = \{1111\}$, and $|\Theta_4^4| = \binom{4}{4} = 1$.

For an N-input core, Φ^N can be used for verifying the functionality of the core completely. However, $|\Phi^N|$ equals 2^N and this number grows rapidly when N increases. Hence, functional verification using Φ^N as verification pattern set is impractical. We have to use another strategy to generate proper patterns for verification.

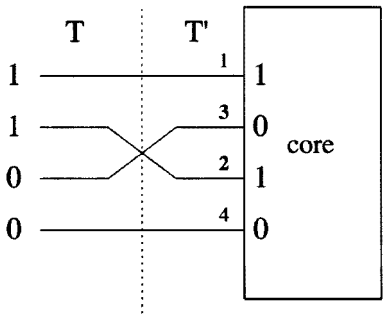
The following paragraphs are going to introduce the three steps in the pattern generation algorithm: fault activation, fault propagation, and fault domination. They are operated in sequence and iteratively and are described in detail in the subsequent sections.

a) Fault Activation: Fault activation is the most important procedure in the algorithm. If the fault effect is not activated, it surely cannot be propagated out. To activate a POF, the logic assignments of the corresponding input ports cannot be all the same. For example, to activate the faulty port sequence 1243, the assignments of Port 3 and Port 4 have to be different, either Port 3 is assigned zero, Port 4 is assigned one or vice versa. All $N! - 1$ POFs have to be activated during the verification pattern generation. The following theorem states the completeness of the POF activation.

Theorem 1: Θ_m^N *where* $m \in [1, 2, \ldots, N-2, N-1]$ *can activate all* $(N! - 1)$ *POFs.*

Proof: Given a pattern $T \in \Theta_m^N$, there are m ones and $(N - m)$ zeroes in T. After any port misplacements, the pattern T turns to T' where T' must also have m ones and $(N - m)$ zeroes ($T' \in \Theta_m^N$). Because Θ_m^N contains all patterns with m ones and $(N-m)$ zeroes, for each POF, there must exist a pair of patterns $(T, T') \in \Theta_m^N$ that corresponds to the original pattern T and the activated pattern T'. Thus, $(N! - 1)$ POFs are all activated by Θ_m^N for $m \in [1, 2, \ldots, N-2, N-1]$. Q.E.D.

Example 6: Fig. 12 shows an example to illustrate that Θ_2^4 can activate all $(4! - 1)$ POFs. In Fig. 12(a), the pattern T, 1100, is the original pattern. After the misplacement of Port 2 and Port 3, the real pattern applied into the core is T', 1010. The pattern T is different with T', hence, the faulty port sequence 1324 is activated. In Fig. 12(b), the patterns in the column T represent the original patterns and the column T' represents the real patterns applied into the core after the port misplacements shown in the second column. In the second row, the pattern pair (1100, 1010) can activate faulty port sequences 1324, 1423, 2314, and 2413. In the third row, the pattern pair (1100, 1001) can activate faulty port sequences 1342, 1432, 2341, and 2431 and so on. $(4! - 1)$ POFs are all activated by pattern pairs $(T, T') \in \Theta_2^4$. To activate a faulty port sequence, the pattern pair (T, T') is not unique. For example, to activate the faulty port sequence 1243, the only requirement is that the assignments of Port 3 and Port 4 have to be different, therefore, both pattern pair (1010, 1001)



(a)

T	activated faulty port sequences	T'
1100	1324, 1423 2314, 2413	1010
1100	1342, 1432 2341, 2431	1001
1100	3124, 4123 3214, 4213	0110
1100	3142, 4132 3241, 4231	0101
1100	3412, 4312 3421, 4321	0011
1010	2134	0110
1010	2143	0101
1010 (0110)	1243	1001 (0101)

(b)

Fig. 12. The $4!-1$ POFs are all activated by patterns in Θ_2^4.

and (0110, 0101) can activate this faulty port sequence, that is shown in the last row of Fig. 12(b).

Corollary 1: Θ_0^N and Θ_N^N cannot be the verification patterns.

According to Theorem 1 and Corollary 1, we can arbitrary apply Θ_m^N for $m \in [1,2,\ldots,N-2,N-1]$ to the inputs of the core to activate $(N!-1)$ POFs. However, note that

$$\left|\Theta_m^N\right| \le \left|\Theta_{m+1}^N\right|, \quad \text{for } m \in \left[1,2,\ldots,\left\lfloor\frac{(N-1)}{2}\right\rfloor\right] \tag{1}$$

$$\left|\Theta_m^N\right| \le \left|\Theta_{m-1}^N\right|, \quad \text{for } m \in \left[\left\lceil\frac{(N+1)}{2}\right\rceil,\ldots,N-1\right]. \tag{2}$$

Equations (1) and (2) show that the $|\Theta_m^N|$ is smaller when m is closer to the end points of interval $[1,2,\ldots,N-1]$. Therefore, to minimize the number of simulation patterns, we select m from one up to $\lfloor N/2 \rfloor$ or from $N-1$ down to $\lfloor N/2 \rfloor$.

b) Fault Propagation: The simulation results of the applied patterns are observed to determine which activated POFs are propagated to POs. The fault effects are propagated to POs if there exists different responses among these input patterns.

c) Fault Domination: If a POF caused by the misplacement of input ports x_i and x_j and denoted as POF(x_i, y_j) is detected by a pattern T, we can figure out what the other POFs are detected by T simultaneously. This characteristic of a POF pattern can reduce the size of the verification pattern set and is stated in Theorem 2. At this stage, the remaining UPSs are also calculated so that more verification patterns can be generated accordingly.

Theorem 2: For an N-input core, assume the POF(x_i, x_j) can be detected by a verification pattern $T \in \Theta_m^N$, then T can actually detect $m! \times (N-m)!$ POFs in total. This characteristic is called the domination property of a POF pattern.

Proof: Suppose the output of the verification pattern T is A. Because T detect the POF(x_i, x_j), the logic value assignments of x_i and x_j in T must be different ($x_i = 0$, $x_j = 1$ or vice versa). The exchange of x_i and x_j reform a new pattern T' and the output of T' is B where $B \neq A$. This is because T is a verification pattern for the POF(x_i, x_j), the outputs of T and T' must be different. If the zeroes in T' are misplaced with themselves, or the ones in T' are misplaced with themselves, the representation of the pattern is still T' and the output remains B. These additional misplacements combined with the POF(x_i, x_j) are all detected by T (output value $B \neq A$) and the total number of these POFs are $m! \times (N-m)!$. Therefore, the verification pattern T can actually detect $m! \times (N-m)!$ POFs. Q.E.D.

Example 7: For a 5-input core, assume a verification pattern 11 000 detects the POF(2,3) and the output of the pattern 11 000 is A. Because the verification pattern 11 000 detects the POF(2,3), the output of the verification pattern 10 100 must not be A (assume it is B). The additional misplacements among the zeroes or between the ones in 10 100 make the pattern 10 100 intact and the output is still B. Therefore, these additional misplacements combined with POF(2,3) are all detected by 11 000 and the amount of them are $2! \times 3!$.

By using the property of the POF activation and domination and the implicit UPS's representation to handle the POF fault list, the process of the POF-based combinational AVPG algorithm is proposed.

The best way to demonstrate the algorithm is to discuss it with an example. Given an 8-input core, according to the UPS's representation, the initial UPSs are (12 345 678). The number of UPSs is $8! - 1 = 40\,320 - 1$. The simulation results of Θ_1^8 are shown in Fig. 13 and are represented in symbolic output representation. The simulation results depend on the functionality of the core. For the first pattern, 10000000, the first bit is one and the other bits are zeroes, the simulation output of 10 000 000 is represented as $A0$. For the second pattern, 01000000, the output is $B0$ ($B0 \neq A0$) and so on. If the first pattern 10 000 000 is applied into the core and assume the interconnection is fault free, the port sequence is 12 345 678 and the output is $A0$ as shown in Fig. 14(a). However, if the Port 1 and Port 2 are misplaced with each other, the port sequence becomes 21 345 678. When the same pattern 10 000 000 is applied, the real pattern assigned into the core is 01 000 000 and the output becomes

initial UPSs=(12345678)
1 0 0 0 0 0 0 0 -> A0
0 1 0 0 0 0 0 0 -> B0
0 0 1 0 0 0 0 0 -> B0
0 0 0 1 0 0 0 0 -> B0
0 0 0 0 1 0 0 0 -> B0
0 0 0 0 0 1 0 0 -> B0
0 0 0 0 0 0 1 0 -> B0
0 0 0 0 0 0 0 1 -> B0
updated UPSs=(1)(2345678)
verification pattern set={ 10000000 }

Fig. 13. The simulation results of Θ_1^8 patterns.

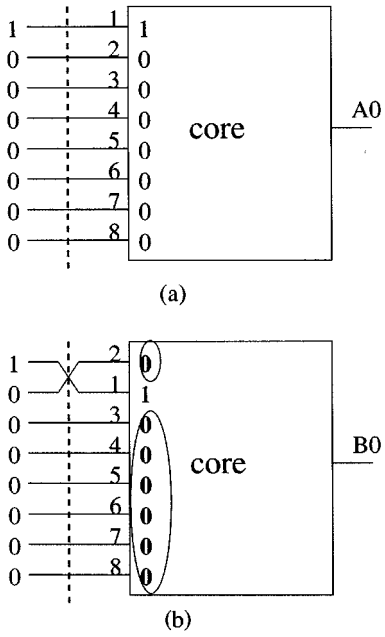


Fig. 14. Different outputs caused by the misplacement of Port 1 and Port 2.

$B0$ as shown in Fig. 14(b). Because the fault-free output $A0$ and faulty output $B0$ are different, the faulty port sequence 21 345 678 can be detected by the pattern 10 000 000. Furthermore, according to Theorem 2, the Port $2 \sim 8$ are all zeroes, arbitrary port misplacements occurred among the ports $2 \sim 8$ after the POF(1,2) will evaluate to the same output ($B0$) and all of them are detected by the pattern 10000000, too. This is shown in Fig. 14(b). Thus, the pattern 10 000 000 can detect {x1xxxxxx} faulty port sequences where x means any other port numbers. For another situation, if the Port 1 and Port 3 are misplaced with each other, the port sequence becomes 32 145 678. When the same pattern 10 000 000 is applied, the real pattern assigned into the core is 00 100 000 and the output becomes $B0$ again according to the simulation outputs of Θ_1^8 shown in Fig. 13. Thus, 10 000 000 can detect this port sequence 32 145 678 and dominate port sequences {xx1xxxxx}. For the other ports, such as ports 4–8, when Port 1 is misplaced with them, the results are similar. Thus, the faulty port sequences that are detected by 10 000 000 are {x1xxxxxx, xx1xxxxx, xxx1xxxx, xxxx1xxx, xxxxx1xx, xxxxxx1x, xxxxxxx1}. The faulty port sequences that are detected by 10 000 000 have been figured out, therefore, the faulty port sequences that cannot be detected by 10 000 000 are decided as well. These UPSs are {1xxxxxxx} which can be represented as (1)(2 345 678) after the pattern 10 000 000 is added into the verification pattern set. These results are shown in Fig. 13.

When the second pattern 01 000 000 in Fig. 13 is applied into the core, since the first pattern 10 000 000 in Fig. 13 is the only one pattern which has different output with that of the second pattern, the second pattern only detects the POF(1,2) and dominates the port sequences {2xxxxxxx}. However, the port sequences {2xxxxxxx} have been detected by verification pattern 10 000 000 in the previous discussion (the updated UPSs are (1)(2 345 678). The second pattern does not detect any new faulty port sequences, therefore, it is not chosen as the verification pattern. Similarly, for the remaining patterns in Θ_1^8, they have no contribution in reducing the size of UPSs, either. Thus, 10 000 000 is the only verification pattern added into the verification pattern set in this iteration.

In Θ_1^8, the ith bit is different with the other bits on the ith pattern. We use the port number i to represent the ith pattern and group the port number i according to the pattern outputs. In this example, the output of the first pattern is different with that of the other patterns, therefore, the grouping result of these port numbers is (1)(2 345 678). This grouping result is the same with the updated UPSs obtained from our detailed discussion above. Hence, in the following discussion with applying $\Theta_1^{N'}$ and $\Theta_{N'-1}^{N'}$ (only one bit is different with the other bits) to the input ports, the updated UPSs can be determined directly from the input port number grouping according to the simulation outputs of these patterns.

Definition 8: A single-element group (SEG) is a group that contains only one port number in the UPS's representation. A multiple-element group (MEG) is a group that contains more than one port numbers in the UPS's representation.

The group (1) in the updated UPSs (1)(2 345 678) is an SEG and the group (2 345 678) in the updated UPSs (1)(2 345 678) is a MEG. The physical meaning of the SEG is that the remaining UPSs are all irrelevant to the port in the SEG and the further pattern generation does not have to activate any POFs related to the port in the SEG. Therefore, when we search Θ_7^8 for additional verification patterns, we find that the pattern 01 111 111 cannot activate any remaining POFs in the updated UPSs (1)(2 345 678). This is because the logic assignments in the ports $2 \sim 8$ of pattern 01 111 111 are all the same. Therefore, we exclude it from Θ_7^8 to minimize the number of simulation patterns. The other patterns in Θ_7^8 and their simulation outputs are shown in Fig. 15. We put an X in the output of the pattern 01 111 111 to indicate the exclusion of this pattern from Θ_7^8 simulations. The remaining patterns in Θ_7^8 are grouped into three groups {10 111 111}, {11 011 111}, and {11101111, 11110111, 11111011, 11111101, 11 111 110} according to their outputs and the corresponding input port numbers grouping is (2)(3)(45678). These groups are sorted by size in ascending order. To add additional patterns into the verification pattern set, we always choose the group with the smallest size if it indeed can detect new faulty port sequences. In this example, when the first group {10 111 111} is added into the verification pattern set, the updated UPSs become

UPSs=(1)(2345678)

0 1 1 1 1 1 1 1 -> X
1 0 1 1 1 1 1 1 ->A1
1 1 0 1 1 1 1 1 ->B1
1 1 1 0 1 1 1 1 ->C1
1 1 1 1 0 1 1 1 ->C1
1 1 1 1 1 0 1 1 ->C1
1 1 1 1 1 1 0 1 ->C1
1 1 1 1 1 1 1 0 ->C1

updated UPSs=(1)(2)(3)(45678)
verification pattern set = {10000000, 10111111, 11011111}

Fig. 15. The simulation results of Θ_7^8 patterns.

(1)(2)(3)(345 678). When the second group {11 011 111} is included into the verification pattern set, the updated UPSs become (1)(2)(3)(45678). These results come from the input port numbers grouping discussed above directly. For the third group, it has no contribution in reducing the size of UPSs further. Hence, it is not added into the verification pattern set. Consequently, in this iteration, the pattern {10 111 111} and {11 011 111} are added into the verification pattern set and the updated UPSs become (1)(2)(3)(45678). The size of the UPSs currently is reduced to $1! \times 1! \times 1! \times 5! - 1 = 120 - 1$.

Hence, the search for the verification pattern of the UPSs (1)(2)(3)(45678) is continued. The UPSs have four groups and are numbered from $G1$ to $G4$, i.e., $G1$ is (1), $G2$ is (2), $G3$ is (3), and $G4$ is (45678). $G1 \sim G3$ are SEGs and $G4$ is a MEG. Note that if the UPSs (1)(2)(3)(45678) can be reduced to (1)(2)(3)(4)(5)(6)(7)(8), the remaining POFs are all detected. The remaining POFs are only related to the ports in the MEGs and the further pattern generation is focused on the activation of these undetected POFs in the MEGs solely.

Then Θ_2^8 are applied into the core. The patterns in Θ_2^8 have two ones and six zeroes. These two ones in each pattern can be placed in the SEGs, MEGs or both. The SEG groups and the MEG group are placed into two sides, respectively, and all combinations of Θ_2^8 patterns are listed in Fig. 16. In Fig. 16, the pattern set P1 does not activate any undetected POFs. Therefore, we have no need to simulate these patterns. In the pattern sets P2, P3, and P4, the MEG side assignments are Θ_1^5 and all remaining POFs are activated according to Theorem 1. For the pattern set P5, the MEG side assignments are Θ_2^5 and also activate all remaining POFs. The SEG side assignments in each pattern set influence on the propagation of the activated fault effects. The propagation of fault effects is determined by the simulation outputs of these pattern sets. When the outputs of a pattern set are different, the fault effects are propagated out and the remaining UPSs could be further reduced.

In the pattern set P2, P3, since the outputs of the patterns in each set are all the same, therefore, the activated POFs cannot be propagated to POs and the UPSs remain (1)(2)(3)(45678). In the pattern set P4, the MEG assignments are Θ_1^5 and the outputs are grouped into two groups. According to the previous discussion, the remaining UPSs can be determined directly by the input port numbers grouping. The grouping result is (45)(678), thus, the updated UPSs become (1)(2)(3)(45)(678) when the patterns

UPSs=(1)(2)(3)(45678)

pattern set	SEG side (1)(2)(3)	MEG side (45678)	outputs	description
P1	1 1 0	00000	X	Two 1s are placed in the SEG side, **no POF activation.**
	1 0 1	00000		
	0 1 1	00000		
P2	1 0 0	10000	A2	One 1 is placed in the SEG side, and another 1 is placed in the MEG side. For the P2 and P3, fault effects are not propagated to POs. **For the P4, fault effects are propagated to POs.**
	1 0 0	01000	A2	
	1 0 0	00100	A2	
	1 0 0	00010	A2	
	1 0 0	00001	A2	
P3	0 1 0	10000	A3	
	0 1 0	01000	A3	
	0 1 0	00100	A3	
	0 1 0	00010	A3	
	0 1 0	00001	A3	
P4	**0 0 1**	**10000**	**A4**	
	0 0 1	**01000**	**A4**	
	0 0 1	00100	B4	
	0 0 1	00010	B4	
	0 0 1	00001	B4	
P5	0 0 0	11000	A5	Two 1s are placed in the MEG side, POFs are activated but not propagated.
	0 0 0	10100	A5	
	0 0 0	10010	A5	
	0 0 0	10001	A5	
	0 0 0	01100	A5	
	0 0 0	01010	A5	
	0 0 0	01001	A5	
	0 0 0	00110	A5	
	0 0 0	00101	A5	
	0 0 0	00011	A5	

updated UPSs=(1)(2)(3)(45)(678)
verification pattern set = {10000000, 10111111, 11011111, **00110000, 00101000**}

Fig. 16. The simulation results of Θ_2^8 patterns.

{00110000, 00 101 000} are added into the verification pattern set. Because the outputs of patterns in the P5 are all the same, P5 is invalid for the POF verification.

In the next iteration, Θ_6^8 are applied into the core and the simulation outputs are shown in Fig. 17. The only pattern set that activates and propagates the remaining POFs is P10. According to the outputs in the P10, the patterns are grouped into two sets S1 and S2 shown in Fig. 17. Since the assignments in the MEG side of P10 are not $\Theta_1^{N'}$ and $\Theta_{N'-1}^{N'}$, the remaining UPS cannot be determined by input port numbers grouping directly. Here, we introduce the characteristic vector (CV) grouping instead.

Definition 9: Given a set of patterns S, 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 CV of S and is denoted as CV_S.

Definition 10: Given two pattern sets S and S', if the patterns in the S and S' are all identical, we said $S = S'$, otherwise $S \neq S'$. If the corresponding bits in the CV_S and CV_S' are all the same, we said $CV\_S = CV\_S'$, otherwise $CV\_S \neq CV\_S'$.

The $CV\_S1[1:8]$ is 22 211 022 and the $CV\_S2[1:8]$ is 88855644, as shown in Fig. 17. Since the updated UPSs are (1)(2)(3)(45)(678), the POFs related to ports in the SEG side are all detected, thus, we only consider the $CV\_S1[4:8]$ and $CV\_S2[4:8]$ when analyzing the updated UPSs.

Lemma 1: One pattern set has only one CV.

UPSs=(1)(2)(3)(45)(678)

pattern set	SEG side (1)(2)(3)	MEG side (45)(678)	outputs	description
P6	0 0 1 0 1 0 1 0 0	11 111 11 111 11 111		Five 1s are placed in the MEG side, **no POF activation.**
P7	0 1 1 0 1 1 0 1 1 0 1 1 0 1 1	01 111 10 111 11 011 11 101 11 110	A7 A7 A7 A7 A7	Two 1s are placed in the SEG side, and four 1s are placed in the MEG side. POFs are activated but not propagated.
P8	1 0 1 1 0 1 1 0 1 1 0 1 1 0 1	01 111 10 111 11 011 11 101 11 110	A8 A8 A8 A8 A8	
P9	1 1 0 1 1 0 1 1 0 1 1 0 1 1 0	01 111 10 111 11 011 11 101 11 110	A9 A9 A9 A9 A9	
P10	1 1 1 **1 1 1** 1 1 1 1 1 1 **1 1 1** 1 1 1 1 1 1 1 1 1 1 1 1	00 111 **01 011** 01 101 01 110 **10 011** 10 101 10 110 11 001 11 010 11 100	A10 **B10** A10 A10 **B10** A10 A10 A10 A10 A10	Three 1s are placed in the SEG and MEG sides, respectively. POFs are activated and propagated to POs.

S1
111 01011
111 10011
222 **11022**
CV_S1[1:8]

S2
111 00111
111 01101
111 01110
111 10101
111 10110
111 11001
111 11010
111 11100
888 **55644**
CV_S2[1:8]

updated UPSs=(1)(2)(3)(45)(6)(78)
verification pattern set={10000000, 10111111, 11011111,
00110000, 00101000, **11101011**, **11110011** }

Fig. 17. The simulation results of Θ_6^8 patterns.

Lemma 2: Given two pattern sets S and S', if $CV\_S \neq CV\_S'$, then $S \neq S'$.

Theorem 3: A pattern set S turns to another pattern set S' after the port misplacements π. If $CV\_S \neq CV\_S'$, then the port misplacements π will be detected by S patterns.

Proof: Because $CV\_S \neq CV\_S'$, according to Lemma 2, $S \neq S'$. Furthermore, because $|S| = |S'|$, S' must exist a pattern $T \notin S$. Since S consists of all patterns with the same outputs, the outputs of T must be different with that of patterns in the S. Thus, the port misplacements π will cause the outputs of S and S' different and will be detected by S patterns. Q.E.D.

According to Theorem 3, if $S1$ in Fig. 17 is chosen as the verification patterns, the port misplacements that change $CV\_S1[4 : 8]$ will be detected. Consequently, the port misplacements which cannot change the $CV\_S1[4 : 8]$ are regarded as the remaining UPSs. The $CV\_S1[4 : 8] = 11\,022$ is further grouped into subgroups, (11)(0)(22) and it corresponds to (45)(6)(78) of the UPSs. The port misplacements occurred in a subgroup will keep the CV remaining the same. Thus, when S1 is added into the verification pattern set, the updated UPSs become (1)(2)(3)(45)(6)(78). It is rewritten as (1)(2)(3)(6)(45)(78) for convenience. For the S2, since the corresponding UPSs of the grouping result of $CV\_S2[4 : 8]$, (45)(6)(78), is the same as that of $CV\_S1[4 : 8]$, it is not added into the verification pattern set.

UPSs=(1)(2)(3)(6)(45)(78)

pattern set	SEG side (1)(2)(3)(6)	MEG side (45)(78)	outputs	description
P11	1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1	00 00 00 00 00 00 00 00		Three 1s are placed in the SEG side, **no POF activation.**
P12	**1 1 0 0** 1 1 0 0 1 1 0 0 1 1 0 0	**10 00** 01 00 00 10 00 01	**A12** B12 B12 B12	(45)(78) ∩ (4)(578) =(4)(5)(78) (1)(2)(3)(6)+(4)(5)(78) **=(1)(2)(3)(4)(5)(6)(78)**
P13	1 0 1 0 **1 0 1 0** 1 0 1 0 1 0 1 0	10 00 **01 00** 00 10 00 01	A13 **B13** A13 A13	(4)(5)(78) ∩ (5)(478) =(4)(5)(78) (1)(2)(3)(6)+(4)(5)(78) =(1)(2)(3)(4)(5)(6)(78)
P14	1 0 0 1 1 0 0 1 1 0 0 1 **1 0 0 1**	10 00 01 00 00 10 **00 01**	A14 A14 A14 **B14**	(4)(5)(78) ∩ (457)(8) =(4)(5)(7)(8) (1)(2)(3)(6)+(4)(5)(7)(8) **=(1)(2)(3)(4)(5)(6)(7)(8)**

"+" presents the concatenation operation

updated UPSs=(1)(2)(3)(4)(5)(6)(7)(8)
verification pattern set = {10000000, 10111111, 11011111, 00110000,
00101000, 11101011, 11110011, **11001000**, **10010001** }

Fig. 18. The simulation results of Θ_3^8 patterns.

Fig. 18 shows the succeeding procedure that applies Θ_3^8 into the core. In the P12, the result of the input port numbers grouping is (4)(578) when the pattern 11 001 000 is added into the verification pattern set. It intersects with the UPSs in the MEG side, (45)(78), to get the updated UPSs (4)(5)(78). Then it concatenates with the UPSs in the SEG side to form the updated UPSs (1)(2)(3)(4)(5)(6)(78). This is shown in the description column in Fig. 18. In the P13, although the output of the pattern, 10100100, is different from that of the other patterns, it is not added into the verification pattern set. This is because the UPSs are not further reduced. In the P14, the pattern 10 010 001 reduces the UPSs from (1)(2)(3)(4)(5)(6)(78) to (1)(2)(3)(4)(5)(6)(7)(8) and the algorithm is terminated at this step. Because the generation of these pattern sets are from the smallest size to the largest size, other pattern sets with larger size in Θ_3^8 are not generated and not shown in Fig. 18. The quality of the verification pattern set is determined by the measurement of fault coverage. The fault coverage is defined as

$$\text{fault coverage} = 1 - (\#\_\text{of\_undetected\_POFs}/\#\_\text{of\_all\_POFs}) \quad (3)$$

therefore, in this example, the fault coverage is $1 - 0/(8! - 1) = 100\%$ and the verification pattern set is

```
Algorithm: Combinational Automatic Verification Pattern Generation (Combinational AVPG)
Input: CUV(N inputs), UPSs
Output: verification_pattern_set, fault_coverage (F_C), UPSs
{
01  F_C ← 0;
02  verification_pattern_set ← ∅;
    For (m=1;m<N/2;m++)
    {
        For (i=0;i<2;i++) /* two iterations for Θ_m^N and Θ_{N-m}^N, respectively */
        {
            if (i == 1)
03              m=N-m;
04          Θ_m^N ← pattern_generation(UPSs,m); /* generate Θ_m^N that activate remaining undetected POFs */
05          outputs ← simulation(Θ_m^N, CUV); /* simulate the generated patterns Θ_m^N */
06          {valid_patterns, updated_UPSs} ← outputs_analysis(outputs, UPSs); /* determine the valid patterns and the updated UPSs */
            if (updated_UPSs ≠ UPSs)
            {
07              F_C ← F_C_calculation(updated_UPSs); /* calculate the fault coverage */
08              verification_pattern_set ← verification_pattern_set ∪ valid_patterns; /* add valid patterns into the verification pattern set*/
09              UPSs ← updated_UPSs; /* update the UPSs for the next iteration */
                if (F_C == 1) break;
            }
        }
        if (F_C == 1) break;
    }
10  return(verification_pattern_set, F_C, UPSs);
}
```

Fig. 19. The pseudocode of the combinational AVPG.

```
Algorithm: Sequential Automatic Verification Pattern Generation (Sequential AVPG)
Input: CUV(N inputs, Q states), UPSs
Output: verification_pattern_set, fault_coverage (F_C), UPSs
{
01  S ← {S_1, S_2, ···, S_Q};
02  verification_pattern_set ← ∅;
03  i ← 1;
    while (F_C < 1 and S≠ ∅)
    {
04      S_i ← set_state_value(); /* set S_i state from S to the core */
05      S ← S - {S_i};
06      {verification_pattern_set(S_i), F_C, UPSs} ← Combinational_AVPG(CUV, UPSs); /* resue the combinational AVPG algorithm */
07      verification_pattern_set ← verification_pattern_set ∪ verification_pattern_set(S_i);
08      i++;
    }
09  return(verification_pattern_set, F_C, UPSs);
}
```

Fig. 20. The pseudocode of the sequential AVPG.

{10000000, 10111111, 11011111, 00110000, 00101000, 11101011, 11110011, 11001000, 10 010 001}.

Since the AVPG will generate all Θ_m^N, for $m = 1, 2, \ldots, N-1$ if necessary and simulate the outputs for searching the verification patterns, it is a complete algorithm, i.e., given enough time, the verification pattern set for 100% fault coverage will be obtained. The pseudocode of this combinational AVPG is shown in Fig. 19. In line 4, 5, it generates patterns and simulates the outputs. The effectiveness of the simulated patterns is determined in line 6. In line 8, it adds the valid patterns into the verification pattern set. At the end of the algorithm, the verification pattern set, fault coverage, and UPSs are returned.

B. The Sequential AVPG

The development of the sequential AVPG is based on the same assumption as the combinational AVPG is, i.e., the CUV is preverified and fault free. The fault occurs only at the interconnection between the cores. Because the core is surrounded by the IEEE P1500 wrapper, by taking the advantages of the scan chains in the wrapper, the CUV can be set in arbitrary *state_values*. Hence, a sequential core can be seen as a combinational one and the verification pattern generation algorithm used in the combinational AVPG is applicable to the sequential AVPG. The only difference is that the sequential core has to be set a *state_value* before being evaluated. Different *state_values*

```
Algorithm: Heuristic Combinational AVPG
Input: CUV(N inputs), UPSs, bound
Output: verification_pattern_set, fault_coverage (F_C), UPSs
{
01  F_C ← 0;
02  verification_pattern_set ← ∅;
03  iteration ← 0; /* clear the iteration counter */
    while (F_C ≠ 1 and iteration ≤ bound)
    {
        For (m=1;m<N/2;m++)
        {
            For (i=0;i<2;i++)
            {
                if (i == 1)
04                  m=N-m;
05              P∈Θ_m^N ← patterns_generation(UPSs,m); /* only generate a pattern set P to simulate the outputs */
06              outputs ← simulation(P, CUV);
07              {valid_patterns, updated_UPSs} ← outputs_analysis(outputs, UPSs);
                if (updated_UPSs == UPSs)
                {
08                  iteration++;
09                  if (iteration > bound) break; /* terminate the algorithm when the number of iterations is over the bound*/
                }
                else
                {
10                  F_C ← F_C_calculation(updated_UPSs);
11                  verification_pattern_set ← verification_pattern_set ∪ valid_patterns;
12                  UPSs ← updated_UPSs;
13                  if (F_C == 1) break;
                }
            } /* end of For (i=0;i¡2;i++) */
14          if (F_C == 1 or iteration > bound) break;
        } /* end of For (m=1;m¡N/2;m++) */
    } /* end of while (F_C ≠ 1 and iteration ≤ bound) */
15  return(verification_pattern_set, F_C, UPSs);
}
```

Fig. 21. The pseudocode of the heuristic combinational AVPG.

affect whether the fault effects could be propagated to POs or not. The algorithm sets the sequential core to every possible state until the fault coverage is 100%. The pseudocode of the sequential AVPG is shown in Fig. 20. In line 4, it sets S_i state to the sequential core before simulation. Then it reuses the combinational AVPG algorithm shown in Fig. 19. This process is listed in line 6. The *state_value* has to be attached to the verification patterns obtained by the reused combinational AVPG. Thus, in line 6, it uses verification_pattern_set(S_i) to represent the verification pattern set obtained in S_i state.

C. The Heuristic AVPG

Definition 12: An untestable POF is a POF which cannot be detected by Φ^N.

The untestable POF is harmless for the integration, therefore, they should be regarded as detected POF in computing fault coverage.

The complete AVPG algorithm may not be practical in generating the verification pattern set for large designs. The possibility of existing untestable POFs in the CUV makes the algorithm very time-consuming. Therefore, a heuristic AVPG algorithm is proposed to trade-off between the fault coverage and the execution time. Here, we only address the heuristic combinational AVPG. This is because the combinational AVPG is the basis of the sequential AVPG.

We review Fig. 16, which shows the simulation outputs of applying Θ_2^8 in the complete combinational AVPG. In this figure, the pattern sets P2 $\sim$ P5 are generated and simulated. However, according to Theorem 1, any one of them can activate all remaining POFs and has the possibility to reduce the size of the remaining UPSs. Hence, the heuristic approach is to generate

TABLE I
EXPERIMENTAL RESULTS OF THE HEURISTIC COMBINATIONAL AVPG

	parameters				combinational AVPG		
bench	|PI|	|PO|	lits.	#_of_POFs	pats.	F_C(%)	time(sec.)
*c*17	5	2	12	5!-1	5	100	< 1
*c*880	60	26	703	60!-1	255	99.999999	75
*c*1355	41	32	1032	41!-1	64	100	13.4
*c*1908	33	25	1497	33!-1	51	100	42
*c*432	36	7	372	36!-1	38	100	4.6
*c*499	41	32	616	41!-1	33	100	8.3
*c*3540	50	22	2934	50!-1	158	100	921
*c*5315	178	123	4369	178!-1	371	100	931
*c*2670	233	140	2043	233!-1	547	99.999999	729
*c*7552	207	108	6098	207!-1	1427	99.999999	1811
*c*6288	32	32	4800	32!-1	30	99.999999	175
des	256	245	7412	256!-1	428	100	159
*alu*4	14	8	1278	14!-1	22	100	2.8
*apex*6	135	99	904	135!-1	234	99.999999	406
*i*9	88	63	1453	88!-1	139	100	24.7
*i*8	133	81	4626	133!-1	266	100	415
*i*7	199	67	1311	199!-1	292	100	103
*i*6	138	67	1037	138!-1	174	100	84
*i*5	133	66	556	133!-1	155	100	63
*duke*2	22	29	1746	22!-1	74	100	83.4
rot	135	107	1424	135!-1	529	99.999999	204
*x*1	51	35	2141	51!-1	275	99.999999	34.1
*x*3	135	99	1816	135!-1	249	99.999999	171
*x*4	94	71	1040	94!-1	398	99.999999	69
pair	173	137	2667	173!-1	217	100	443

TABLE II
EXPERIMENTAL RESULTS OF THE HEURISTIC SEQUENTIAL AVPG

	parameters					sequential AVPG		
bench	|PI|	|PO|	lits.	FFs	#_of_POFs	pats.	F_C(%)	time(sec.)
$s1196$	14	14	1009	18	14!-1	24	99.999999	16
$s1238$	14	14	1041	18	14!-1	16	99.999999	27
$s13207$	31	121	11241	669	31!-1	64	99.999999	1165
$s1423$	17	5	1164	74	17!-1	107	100	48
$s1488$	8	19	1387	6	8!-1	10	100	6
$s1494$	8	19	1393	6	8!-1	14	100	5
$s15850$	14	87	13659	597	14!-1	11	99.999999	256
$s208$	11	2	166	8	11!-1	25	100	6
$s27$	4	1	18	3	4!-1	3	100	<1
$s344$	9	11	269	15	9!-1	10	100	2
$s349$	9	11	273	15	9!-1	10	100	2
$s35932$	35	320	28269	1728	35!-1	549	99.999999	3113
$s420$	19	2	336	16	19!-1	294	99.999999	16
$s510$	19	7	424	6	19!-1	170	100	16
$s5378$	35	49	4212	164	35!-1	393	99.999999	363
$s641$	35	23	539	19	35!-1	86	99.999999	23
$s713$	35	23	591	19	35!-1	70	99.999999	20
$s820$	18	19	757	5	18!-1	117	99.999999	43
$s832$	18	19	767	5	18!-1	23	100	9
$s838$	35	2	670	32	35!-1	1845	99.999999	279
$s9234$	19	22	7971	228	19!-1	23	100	109
$s953$	16	23	743	29	16!-1	95	100	16
scf	27	56	1865	7	27!-1	417	100	84
$minmax10$	13	10	735	30	13!-1	51	100	24
$minmax12$	15	12	909	36	15!-1	71	100	21
$minmax32$	35	32	3089	96	35!-1	706	99.999999	154
$minmax5$	8	5	335	15	8!-1	10	100	2
$mult32$	32	64	14745	32	32!-1	31	100	20
sbc	40	56	1586	28	40!-1	182	100	91
tlc	3	5	423	10	3!-1	3	100	<1

arbitrary one of them to simulate the outputs in one iteration instead of all pattern sets. For example, it can only generate and simulate P2 or P3, etc. No matter what the result of the simulation of generated pattern set is, it proceeds to the next iteration. On the other hand, the heuristic AVPG generates the verification patterns iteratively, therefore, it sets an iteration counter to bound the processing time.

The heuristic combinational AVPG algorithm is shown in Fig. 21. In line 5, it generates a pattern set P instead of all Θ_m^N heuristically where P is a subset of Θ_m^N. The pattern set P satisfies the requirement of activating all remaining undetected POFs. The pattern set P3, P8, or P12 listed in Figs. 16–18 are all instances of P. The check points in the algorithm bound the AVPG to be executed in the acceptable run time. These codes in lines 3, 8, 9, and 14 are all highlighted in Fig. 21.

V. EXPERIMENTAL RESULTS

Rather than simulating the entire SoC, the complexity of SoC design verification can be significantly alleviated by using preverified IP cores and concentrating on verifying the integration of the cores in the SoC. The interconnection verification among each single block during integration is the first step of the core-based SoC design verification. Furthermore, the generation of the verification patterns for verifying the interconnection among the cores only depends on the functionality of the added core. Therefore, the experiment only reports the results of single block examples.

The heuristic POF AVPG algorithm described above has been integrated into the SIS [8] environment, which is developed by the University of California at Berkeley. Experiments are conducted over a set of ISCAS-85 and MCNC combinational benchmarks for the heuristic combinational AVPG and a set of ISCAS-89 sequential benchmarks for the heuristic sequential AVPG. Note that 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 are used to conduct the experiments and therefore, arbitrary level of design description can be used for generating verification patterns. Table I summaries the experimental results of the heuristic combinational AVPG. The first five columns show the parameters of each benchmark, including name, $|\text{PI}|$, $|\text{PO}|$, the number of literals (lits.), and the number of POFs. The $|\text{PI}|$ represents the number of inputs and the size of the POFs set is $|\text{PI}|! - 1$. The $|\text{PO}|$ represents the number of outputs and influences on the probability of fault effect 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 iteration bound was set to 100. The CPU time is measured on an Ultra Sparc II workstation. The algorithm will be terminated automatically if the iteration counter is over the bound or the fault coverage reaches 100% and the verification pattern set, fault coverage, and updated UPSs are returned. According to Table I, the fault coverage of more than half benchmarks achieve 100% and the processing time is acceptable. Furthermore, the size of the verification pattern sets are very small as compared with the $(N! - 1)$ POFs. For example, the number of POFs in c5315 is $(178! - 1)$, but the size of the verification pattern set is only 371

for 100% fault coverage. For the other benchmarks, the fault coverage also reach 99.999 999% high.

The results demonstrate that the heuristic combinational AVPG is very efficient to generate high-quality verification pattern set. Note that the undetected POFs in a benchmark could be untestable. If we omit these untestable POFs from the fault set, the fault coverage of the pattern set would be even higher.

Table II summaries the experimental results of the heuristic sequential AVPG. The fifth column lists the number of flip-flops in a benchmark. Table II also demonstrates that all sequential benchmarks achieve high fault coverage with acceptable run time similar to the results indicated in the heuristic combinational AVPG.

VI. Conclusion

In the SoC era, the embedded cores are mixed and integrated to create a system chip. System designers integrate those cores manually and have the possibility of incorrect integration due to the misplaced I/O ports. Furthermore, without the knowledge of the internal structures of the embedded cores, system designers have difficult time to locate the position of having erroneous interconnection. Therefore, we adopt the connectivity-based POF model and use the proposed verification mechanism to integrate the system blockwise. This raised abstraction level of the design verification reduce the time on functional verification in core-based design methodology.

The POF-based AVPG algorithm generates the verification pattern set for $(N! - 1)$ POFs systematically by using the property of the POF activation and domination and the implicit UPS's representation. We exploit the IEEE P1500 wrapper structure as the integration verification mechanism. The POF verification provides a sufficient high level of confidence on verifying the correctness of the core-based system design.

References

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

[2] 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 Academic, 1999.

[3] J. Bergeron, *Writing Testbenches-Functional Verification of HDL Model*. Norwell, MA: Kluwer Academic, 2000.

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

[5] A. Hassan, V. K. Agarwal, B. Nadeau-Dostie, and J. Rajski, "BIST of PCB interconnects using boundary-scan architecture," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1278–1288, Oct. 1992.

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

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

[8] 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.

[9] 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.*, Sept. 1999, pp. 616–627.

[10] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. New York: Computer Science Press, 1990, p. 95.

[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] S. Runyon, "Testing big chips becomes an internal affair," *IEEE Spectrum Mag.*, vol. 36, pp. 49–55, Apr. 1999.

[13] R. G. B. Bennetts. IEEE P1500 Embedded Core Test Standard. [Online]. Available: http://www.semiconductorfabtech.com/dft/tutorial/p1500.PDF

[14] M. Ricchetti. Overview of proposed ieee scalable architecture for testing embedded cores. presented at IEEE P1500 SECT Meeting During DAC'01. [Online]. Available: http://www.grouper.ieee.org/groups/1500/dac01/ctag-dac01.pdf.

[15] K.-T. Cheng and J.-Y. Jou, "A functional fault model for sequential machines," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1065–1073, Sept. 1992.

[16] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 126–137, Jan. 1988.

Chun-Yao Wang (S'00) was born on Oct 21, 1973 in Taiwan, ROC. He graduated from the Department of Electronics Engineering, National Taipei Institute of Technology, Taiwan, ROC, in 1994. He is currently pursuing the Ph.D. degree at the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC.

His research interests include testing, design verification, and logic synthesis.

Shing-Wu Tung (M'92) received the B.S. degree from the Department of Electrical Engineering, National Cheng Kung University, and the M.S. degree from the Department of Computer Science and Information Engineering at National Chiao Tung University, Hsinchu, Taiwan, ROC, in 1987 and 1989, respectively. He is working toward the Ph.D. degree at the Department of Electronics Engineering, National Chiao Tung University.

He is a senior manager at Prolific Technology Inc. He has worked at Industrial Technology Research Institute. His current research activities focus on design verification of platform-based SoC design. His other areas of interest include computer architecture, CPU design, and VLSI system verification.

Jing-Yang Jou (S'82–M'83–SM'02) received the B.S. degree from the Department of Electrical Engineering at National Taiwan University, Taiwan, ROC, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, in 1979, 1983, and 1985, respectively.

He is currently a full Professor and Chairman of Electronics Engineering Department at National Chiao Tung University, Hsinchu, Taiwan, ROC. Before joining National Chiao Tung University, he was with GTE Laboratories and AT&T Bell Laboratories. His research interests include behavioral, logic and physical synthesis, design verification, and CAD for low power. He has published more than 100 journal and conference papers.

Dr. Jou is a member of Tau Beta Pi and is the recipient of the distinguished paper award of the IEEE International Conference on Computer-Aided Design, 1990. He served as the Technical Program Chair of the Asia-Pacific Conference on Hardware Description Languages (APCHDL'97).