AN AVPG FOR SOC DESIGN VERIFICATION WITH PORT ORDER FAULT MODEL

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

Department of Electronics Engineering,
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
{wcyao,swtung,jyjou}@eda.ee.nctu.edu.tw

ABSTRACT

Embedded cores are being increasingly used in the design of large system-on-a-chip (SoC). Because the high complexity of SoC, the design verification is a challenge for system integrator. To reduce the verification complexity, the port order fault (POF) model has been used for verifying the core-based design [1]. In this paper, we present a verification scheme and an automatic verification pattern generation (AVPG) system based on POF model.

1. INTRODUCTION

Spurred by process technology leading to the availability of more than 1 million gates per chip, and more stringent requirements upon time-to-market and performance constraints, system level integration and platform-based design [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 re-use and Intellectual Property(IP) trading shall now be considered a necessity. However, present design methodologies are not enough to deal with IPs 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 a difficult task.

The focus of core-based design verification should be on how the cores communicate with each other [3]. By creating the test-benches at a high level, a connectivity-based design fault (port order fault, **POF**) model proposed in [4] is used for reducing the time on core-based design verification [1]. It assumes the IPs are pre-verified and verifies the interconnections among them only. The POF model has been used in library coherence [4] and reduces the time on library verification.

In [1], however, it simplifies the POF model to the simple POF(SPOF) model(only two ports misplaced at a time) and co-operates with the stuck at fault(SAF) automatic test pattern generation(ATPG) to generate the verification set. In our AVPG, we consider all possible misplacements among the ports rather than SPOF and only use the simulation information of the circuit to generate the verification set. In this paper, in addition to AVPG, we also present a verification scheme to ease the design verification. We have integrated the POF AVPG into SIS [5] and conduct the experiments on ISCAS-85 and MCNC benchmarks. Experimental results show that the AVPG can efficiently generate verification set with high POF coverage. The association of the proposed verification scheme and verification set can detect POFs occurred among the blocks and provides a sufficient high level of confidence on the correctness of the core-based design.

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

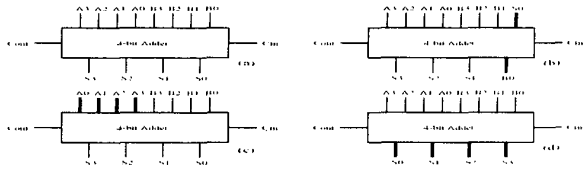

Figure 1: POF model

2. PRELIMINARY

Definition 1: The type I POF is at least one 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 [4].

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

It has been proven that the type II POF **dominates** the type I and III POFs [4]. In this paper, we consider the **type II POF**.

Example 2.2: Given a 4-input circuit, we number the inputs from 1 to 4. Any input permutation is a **port sequence** of the circuit. The 1234 denotes the **fault free port sequence** and 2134 denotes port 1 and port 2 are misplaced. The number of the POFs is 4!-1.

3. INTEGRATION VERIFICATION

3.1. Design Verification of System Chip

Fig. 2(a) depicts a generic verification scheme for core-based system chip. Since these cores, BLK1, BLK2 and BLK3, are pre-verified, the verification efforts during integration phase should focus on the interconnections among the cores. To verify the interconnections from BLK1 and BLK2 to BLK3, designers apply patterns T to the primary inputs(PIs) of the integrated design, then compare the responses R to the expected results in the primary outputs(POs). If the responses R are inconsistent with the expected one, some interconnections are misplaced. The generation of patterns T depends on the functionalities of BLK1, BLK2 and BLK3.

0-7803-6685-9/01/$10.00©2001 IEEE

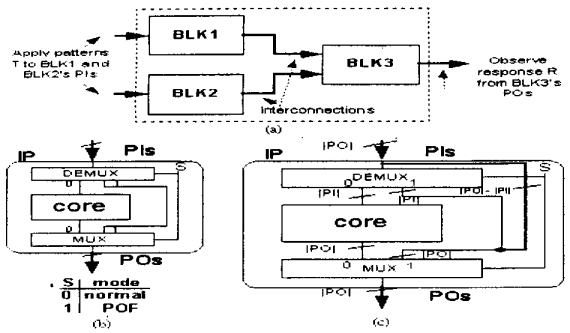

Figure 2: Generic Verification Scheme

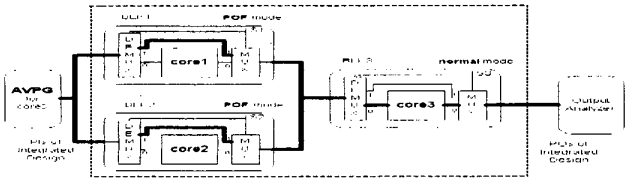

Figure 3: System Design Verification

As more cores are involved in the integration verification, the patterns T become harder to generate. In section 3.2, we propose a verification scheme that allows us to ignore the functionalities of BLK1 and BLK2 when verifying the interconnections.

3.2. Verification Scheme

Fig. 2(b) shows the proposed verification scheme of a core. An **IP** consists of a pre-designed core, a de-multiplexer(DEMUX) and a multiplexer(MUX). The DEMUX and MUX are both controlled by S. When S is set to 0, the core is operated in **normal** mode. The signals coming from PIs go through the core and propagate to POs. When S is set to 1, the core is operated in **POF** mode. The signals bypass the core and propagate to POs.

In Fig. 3, we show the operation of verifying the interconnections among BLK1, BLK2 and BLK3. We set the BLK1 and BLK2 to POF mode and BLK3 to normal mode, the generated POF verification set no longer depends on the functionalities of BLK1 and BLK2. Because the scheme allows the input verification set to bypass the core1 and core2 and to go into the core3. Hence, we only need to consider the functionality of BLK3 when generating the verification set.

If the number of PIs(|PI|) of the BLK3 is m, from the configuration shown in Fig. 3, we know the sum of the number of POs(|PO|) of the BLK1 and BLK2 must be equal to m. However, the individual |PI| of the BLK1 and BLK2 could be great than, equal or less than the individual |PO|. The scheme must have enough bus width to propagate the verification set from the AVPG to the PIs of the BLK3 through the BLK1 and BLK2. Therefore, for the core whose |PO| great than |PI|, we add a bus with (|PO| - |PI|) bits from the PIs of each block to its MUX, thus the AVPG would have enough bus width to propagate verification set. The modified verification scheme is shown in Fig. 2(c). The bold line is the additional bus.

4. OUR AVPG SYSTEM FOR POF

The AVPG reads the **combinational** circuit and generates the **heuristic verification patterns** iteratively. It terminates when the fault coverage reaches 100% or the iterations over the bound. The main algorithm in our AVPG is the **heuristic verification pattern generation** algorithm.

4.1. The RUPS Representation

Typically, SAF ATPG build fault list **explicitly** first, then generate random and deterministic test patterns. For our POF AVPG, however, we generate heuristic patterns instead of them. Furthermore, we **do not enumerate** fault list explicitly, this is because the number of POFs in an N-input circuit is (N!-1). This number grows rapidly when N increases, for instance, as N=69, $N!-1 \approx 1.7 \times 10^{98}$. Hence, we use a proper representation to indicate the remaining undetected port sequences(**RUPSs**) during the algorithm.

Example 4.1: (12345678) represents the RUPSs caused by the all possible misplacements among port 1 ~ port 8. The number of RUPSs is 8!. (125)(4)(3678) indicates the RUPSs that caused by all possible misplacements among the port 1, 2 and 5 or among the port 3,6,7 and 8. The number of the RUPSs is $3! \times 1! \times 4!$. Please note that the port number 4 is the only one element in its group. This means that the port sequences whose port number 4 in the wrong position have been detected. (1)(2)(3)(4)(5)(6)(7)(8) represents that all 8!-1 POFs are detected. Each group has only one element, therefore, no misplacement could be occurred in each group. The number of the RUPS is $(1!)^8 = 1$ and it is the fault free port sequence. When we induce the RUPSs from (12345678) to (1)(2)(3)(4)(5)(6)(7)(8), we can claim all POFs are detected.

4.2. The Heuristic Verification Pattern Generation

In this section, we describe the heuristic verification pattern generation algorithm in our AVPG. Before the illustration of the algorithm, we state an important theorem.

Theorem 1: *For an N-input circuit, assume the POF caused by the misplacement of input ports x_i and x_j and denoted as $POF(x_i,x_j)$ can be detected by a pattern V with m bits 1s and (N-m) bits 0s, then this pattern can actually detect m! × (N-m)! POFs in total. This characteristic is called the* **domination property** *of a POF pattern.*

Example 4.2: For a 4-input circuit, given a pattern 1000 which detects the POF(1,2) and the output of 1000 is A. Because the **10**00 detects POF(1,2), the output of the **01**00 must not equal A. The additional misplacements occurred among the 0 bits themselves in 0100 make the pattern, 0100, remain the same. Thus, this pattern 1000 actually detect 1! × 3! POFs and they are {2134, 2143, 3124, 3142, 4123, 4132}=x1xx (x means any other port numbers).

The best way to explain our algorithm is to discuss it with an example. Given an 8-input combinational circuit, the initial RUPSs are (12345678). The number of RUPSs is 8!=40320. In Fig. 4(a), we generate "one 0 patterns" and simulate the outputs. We use symbolic output representation and group the port numbers into the same group if their outputs are the same, This grouping result (1)(2345678) are the updated RUPSs. We explain clearly why the simulation results of these "one 0 patterns" can induce the RUPSs from (12345678) to (1)(2345678). If we apply the pattern 01111111 into the circuit and assume the circuit is fault free, the port sequence is 12345678 and the output is A0 as shown

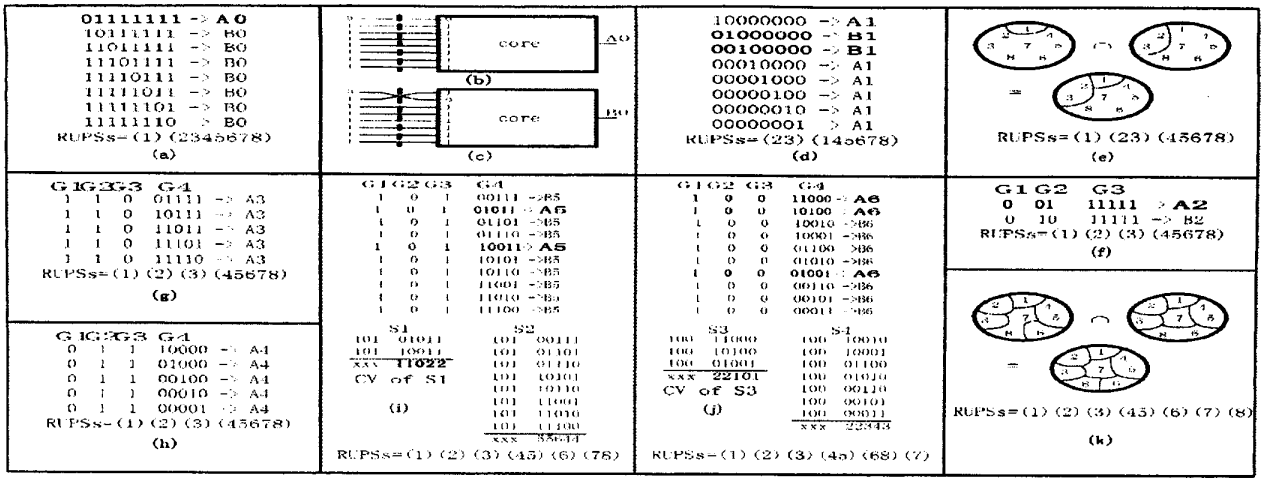

Figure 4: An Example of POF Verification Patterns Generation

in Fig. 4(b). However, if the port 1 and port 2 are misplaced with each other, the port sequence becomes **21**345678. When we apply the pattern 01111111 into it, the real pattern assigned into it is 10111111, this makes the output become $B0$ as shown in Fig. 4(c). Because the fault free output $A0$ and faulty output $B0$ are different, we claim this pattern 01111111 can detect the port sequence **21**345678. Furthermore, according to Theorem 1, the port $2 \sim 8$ are all 1s, arbitrary POFs occurred among the port $2 \sim 8$ after the POF(1,2) would evaluate to the same output($B0$). Thus, the pattern 01111111 can detect x1xxxxxx port sequences. For another situation, if the port 1 and 3 are misplaced with each other, the port sequence becomes **32**145678. When we apply the same pattern 01111111 into it, the real pattern assigned into it is 11011111, this makes the output become $B0$ again. Thus, 01111111 can detect this port sequence **32**145678 and **dominate** port sequences xx1xxxxx. For the other ports, such as port 4, 5, 6, 7 and 8, when port 1 is misplaced with them, they have similar results. Thus, the port sequences can be detected by 01111111 are {x1xxxxxx, xx1xxxxx, xxx1xxxx, xxxx1xxx, xxxxx1xx, xxxxxx1x, xxxxxxx1}. We have figured out the port sequences that are detected by 01111111, we can decide which port sequences that **cannot be detected** by 01111111 as well. These port sequences are {1xxxxxxx} and can be represented as (1)(2345678) in our RUPSs representation. Hence, the updated RUPSs are (1)(2345678) after we choose 01111111 as verification pattern.

We have generated the "one 0 patterns" and discussed the effect of them. We find that the "one 1 patterns" have the same property and can reduce the RUPSs further. Fig. 4(d) shows the "one 1 patterns" and the corresponding outputs. We choose {01000000, 00100000} as verification patterns and the corresponding RUPSs are (23)(145678). The overall RUPSs when we select {01111111, 01000000, 00100000} as verification set are (1)(23)(45678). This result comes from the intersection of the RUPSs (1)(2345678) and (23)(145678) as shown in Fig. 4(e). Why do we intersect the two RUPSs to get the updated RUPSs? This is because there exists some RUPSs that are in (1)(2345678) but not in (23)(145678). For instance, port sequence 13456782 $\in$ (1)(2345678) and cannot be detected by {01111111}, however, it $\notin$ (23)(145678) and can be detected by {01000000, 00100000}. Therefore, when we choose {01111111, 01000000, 00100000} as the verification set, the real RUPSs are the intersection of (1)(2345678) and (23)(145678). The number of the RUPSs is reduced to $1! \times 2! \times 5!$=240.

Thereafter, we continue to find the verification pattern for the RUPSs (1)(23)(45678). The RUPSs have three groups and we number them from $G1$ to $G3$, i.e., $G1$ is (1), $G2$ is (23) and $G3$ is (45678). We have known that if we can make the RUPSs (1)(23)(45678) become (1)(2)(3)(4)(5)(6)(7)(8), the remaining 240 port sequences are detected. Our strategy is to attack one group at one iteration. The group Gi with $|Gi| \geq 2$ is called possible target group, which can be chosen as target group arbitrary at any one iteration. Here we choose $G2$ as the target group first. For the inputs in the $G1$ or $G3$, we heuristically assign values to them and let the assigned values be the same if they are in the same group. For the inputs in the $G2$, we assign "one 0 patterns" to them. Fig. 4(f) shows such assignments and the corresponding outputs. Since the outputs, $A2$ and $B2$, are different, we can choose any one of them, for instance, 00111111, as verification pattern, then the updated RUPSs are (1)(2)(3)(45678). Why do we let the assigned values in the same group be the same? This is because we want to **fully exploit** the **domination property** of a pattern to reduce the size of RUPSs rapidly. We explain this idea. We rewrite the RUPSs (1)(23)(45678) as {(1)**23**(45678), (1)**32**(45678)}. According to the outputs of these two patterns shown in Fig. 4(f), we know that we can detect POF(2,3) obviously when we choose either one pattern as verification pattern. Furthermore, the same assignments in the $G3$ make the POFs occurred among the ports in the $G3$ be detected simultaneously. Therefore, the RUPSs (1)**32**(45678) are detected completely and the updated RUPSs are (1)**23**(45678), we present it as (1)(2)(3)(45678).

For the next iteration, we apply the same procedure. We choose $G4$, (45678), as target group, the assigned patterns which $G4$ is assigned in "one 0 patterns" are shown in Fig. 4(g). Since the outputs are all the same($A3$), the RUPSs remain the same. Again, another set of patterns which $G4$ is assigned in "one 1 patterns" are shown in Fig. 4(h). Because the outputs are still the same($A4$), the RUPSs cannot be reduced further in this iteration.

We generate $\binom{N}{1}$ "one 0 patterns" and $\binom{N}{1}$ "one 1 patterns"

and use them as part of the input patterns to attack the target group Gi with $|Gi|$=N. However, when the outputs of the assigned patterns are all the same, we will extend our strategy to generate $\binom{N}{2}$ "two 0s patterns" and $\binom{N}{2}$ "two 1s patterns". This strategy is based on the following fact. We rewrite the combination formula, $\binom{N}{m}=\frac{N!}{m!(N-m)!}$, as N!=$\binom{N}{m}$×m!(N-m)!. This identity means that the N! port sequences can be distributed over $\binom{N}{m}$ patterns and each pattern covers m!×(N-m)! port sequences disjointly. Therefore, we can further extend to $\binom{N}{m}$ "m 0s/1s patterns" for attacking a target group Gi with $|Gi|$=N if necessary, where m≤ $\lfloor N/2 \rfloor$.

Fig. 4(i) shows the generated $\binom{5}{2}$ "two 0s patterns" in the $G4$ and the assigned values in the $G1$~$G3$ to attack the target group $G4$. We group the patterns into two sets $S1$ and $S2$ according to their outputs and focus on the assignments in the $G4$. Here we define the **characteristic vector** of a set of patterns first for better illustration in the succeeding discussion.

Definition 2: Given a set of patterns with the same length, we calculate the number of 1 digit in the same bit position to form a vector with the same length. This vector is called the **characteristic vector(CV)** of the given set.

The CV of $S1$ and $S2$ are 11022 and 55644, respectively, and are shown in Fig. 4(i). We examine the patterns in the $S1$. The 5! port sequences are distributed over the $\binom{5}{2}$ patterns. If we choose $S1$ as verification patterns, the port misplacements that transform at least one pattern in the $S1$ into $S2$ are all detected according to the different outputs in $S1$ and $S2$. The RUPSs are exactly the misplacements that **cannot achieve this transformation**. In other words, assume we say the $S1$ patterns will transform to $S1'$ after some port misplacements where $|S1|=|S1'|$. If $S1' \cup S1 = S1$, these port misplacements are the RUPSs. We find that a property will be held if $S1' \cup S1 = S1$. The property is the CVs of $S1'$ and $S1$ are identical. Hence, we figure out the CV of $S1$ and regard the port misplacements that make the CV of $S1'$ be identical to that of $S1$ as the RUPSs. The CV of $S1$ is 11022, the port misplacements of (45) or (78) keep the CV remaining the same. Thus, when we add $S1$ into our verification set, the updated RUPSs become (1)(2)(3)(45)(6)(78). Fig. 4(j) shows the generated patterns that assign "two 1s patterns" to $G4$ and heuristic assignments to the other groups. The CV of $S3$ is 22101, therefore, the RUPSs are (1)(2)(3)(45)(68)(7) when we include $S3$ into the verification set. When we include both $S1$ and $S3$ into the verification set, the updated RUPSs are (1)(2)(3)(45)(6)(7)(8) as shown in Fig. 4(k) which come from the intersection of (1)(2)(3)(45)(6)(78) and (1)(2)(3)(45)(68)(7). Thus, the size of RUPSs is reduced to $(1!)^7 \times 2!=2$.

We apply the same procedure to reduce the RUPSs iteratively. However, if we cannot reduce the RUPSs further under the iteration bound, we terminate the AVPG and return the verification set and fault coverage(F.C.). The fault coverage is defined as $1-\frac{no.of undetected POFs}{no.of all POFs}$. In this example, the verification set is {01111111, 01000000, 00100000, 00111111, 10101011, 10110011, 10011000, 10010100, 10001001}, which is shown in bold in Fig. 4. The undetected POF is 12**3**5**4**678 and the fault coverage is $1-\frac{1}{8!-1}$=99.998%.

5. EXPERIMENTAL RESULTS

The POF AVPG described above has been integrated into SIS [5] environment. Experiments are conducted over a set of ISCAS-85 and MCNC benchmarks. Table 1 summaries the experimental results of our AVPG. We set the iteration bound to 100. The CPU time is measured on Ultra Sparc II workstation. According to Table 1, we find that the F.C. of more than half benchmarks achieve 100% and the processing time are acceptable. Furthermore, the size of the verification sets are very small as compared with the number of POFs. For example, the number of POFs in c5315 is 178!-1, but the size of the verification set is only 371 for reaching 100% F.C.. For the other benchmarks, the F.C. also reach 99.999999% high. The experimental results illustrate that our AVPG is not a complete algorithm, it is an effective one though. If we want to guarantee the completeness of AVPG, the exhaustive assignments in all input ports are indispensable, and this make the AVPG be time-consuming. Hence, we set the iteration number to bound the processing time and also provide the very high F.C..

bench	parameters			our AVPG		
	|PI|	|PO|	lits.	pats.	F.C.	time(sec.)
c17	5	2	12	5	1	< 1
c880	60	26	703	243	0.99999999	70
c1355	41	32	1032	64	1	13.4
c1908	33	25	1497	51	1	42
c432	36	7	372	38	1	4.6
c499	41	32	616	33	1	8.3
c3540	50	22	2934	145	1	727
c5315	178	123	4360	371	1	931
c2670	233	140	2043	521	0.99999999	721
c7552	207	108	6098	1627	0.99999999	1826
c6288	32	32	4800	30	0.99999999	175
des	256	245	7412	428	1	159
alu4	14	8	1278	22	1	2.8
apex6	135	99	904	234	0.99999999	406
i9	88	63	1453	139	1	24.7
i8	133	81	4626	266	1	415
i7	199	67	1311	292	1	103
i6	138	67	1037	165	1	77
i5	133	66	556	155	1	63
duke2	22	29	1746	74	1	83.4
rot	135	107	1424	524	0.99999999	246
x1	51	35	2141	275	0.99999999	34.1
x3	135	99	1816	249	0.99999999	171
x4	94	71	1040	352	0.99999999	69
pair	173	137	2667	217	1	443

Table 1: Experimental Results

6. CONCLUSIONS

In this paper, we have presented an AVPG system based on POF model. To reduce the time on functional verification in core-based design methodology, we adopt the connectivity-based **POF** model to elevate the abstraction level of the design verification. In our AVPG, we solve the N! POF problem systematically with using the **domination property** of a valid pattern and the implicit RUPSs representation. We also propose a verification scheme for verifying the interconnections among the cores. The association of the verification scheme and verification set can detect POFs occurred among the integrated cores and provide a sufficient high level of confidence on the correctness of the core-based design.

7. REFERENCES

[1] Shing-Wu Tung and Jing- Yang Jou, "Verification pattern generation for core-based design using port order fault model.", The 7th Asian Test Symposium, Dec. 1998, pp.402-407.

[2] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly et al., "Surviving the SOC revolution - a guide to platform-based design," Kluwer Academic Publishers, 1999.

[3] J. Rowson and A. Sangiovanni-Vincentelli, "Interface-based design," Proceedings of the Design Automation Conference, 1997, pp.178-183.

[4] Shing-Wu Tung and Jing-Yang Jou, "Library coherence checking using port order fault model," The fourth Asia Pacific Conference on HDL, Aug. 1997, pp.83-90.

[5] E. Sentovich et al., "SIS: a system for sequential circuit synthesis," *Memorandum UCB M93/41, UC, Berkeley*,