논문 2005-42SD-7-6

Crosstalk과 정적 고장을 고려한 효과적인 연결선 테스트 알고리즘 및 BIST 구현

(Efficient Interconnect Test Patterns and BIST Implementation for Crosstalk and Static Faults)

민 병 우*, 이 현 빈*, 송 재 훈*, 박 성 주**

(Pyoungwoo Min, Hyunbean Yi, Jaehoon Song, and Sungju Park)

요 약

본 논문은 보드 또는 SoC 상에서 코아와 코아 사이의 연결선 고장 점검을 위한 효과적인 테스트 패턴 알고리즘과 테스트 패턴 생성기를 소개한다. 연결선 고장 모델 분석을 통해 crosstalk과 정적인 고장을 100% 점검할 수 있는 6n 패턴 알고리즘을 소개한다. 보다 적은 4n+1 개의 패턴으로 100%에 가까운 고장 점검율을 얻으면서 crosstalk 뿐 아니라 정적고장의 검출 및 진단도 가능한 알고리즘을 제안하고, 효과적인 BIST구현 기술에 대하여 소개한다.

Abstract

This paper presents effective test patterns and their BIST implementations for SoC and Board interconnects. Initially '6n' algorithm, where 'n' is the total number of interconnect nets, is introduced to completely detect and diagnose both static and crosstalk faults. Then, more economic '4n+1' algorithm is described to perfectly capture the crosstalk faults for the interconnect nets separated within a certain distance. It will be shown that both algorithms can be easily implemented as interconnect BIST hardwares with small area penalty than conventional LFSR.

Keywords: interconnect testing, crosstalk faults, test pattern generator, BIST, SoC, static faults

I. Introduction

System chips are increasingly designed by embedding reusable cores such as processors, memories, and peripheral interfaces. Today's boards become tomorrow's IC's, whereby today's IC's become tomorrow's cores. It is crucial to test various kinds of dynamic as well as static defects on SoCs and boards. Scan design techniques have emerged to overcome some of the difficulties of producing test data for different fault models and levels of packaging (e.g. chips, boards, modules). In-circuit tests and functional tests have been the primary method for testing chips on a board and interconnections of chips on a board. In-circuit test based on the bed-of-nails probing technique makes it possible to test each chip and the interconnections among chips. However it requires the automatic test equipment (ATE) to probe each chip pin and the increasing use of surface mounting techniques make it difficult to perform in-circuit test. Although functional test does not need extensive access to chip

* 학생회원, 한양대학교 컴퓨터공학과
(Department of Computer Science & Engineering, Hanyang University.)
** 정회원, 한양대학교 전자컴퓨터공학부
(Department of Electronical Engineering Computer Science, Hanyang University.)
※ This work was support by grant no. R01-2003-000-101-50-0 from Korea Science & Engineering Foundation.
접수일자: 2005년3월15일, 수정완료일: 2005년5월7일

pins on the board, it can not be guaranteed to derive high coverage test patterns. Also it may require a detailed description of each chip which often is not available from the vendor. Boundary scan is a design for testability technique aiming to improve the board level testability by embedding a dedicated boundary scan register or making use of the part of the scan register in each chip. IEEE 1149.1 boundary scan design which uses an explicit test protocol has been a widely adopted industry standard. IEEE P1500 pursues a standard for testing embedded cores by providing a flexible hardware interface between an embedded core and its environment so that the predefined test patterns can be efficiently delivered to and from the embedded core. It becomes highly important to capture critical timing defects as well as conventional static faults on the interconnect lines among SoCs on a board and cores on an SoC. When boundary scan design techniques are adopted, the chip to chip interconnection test generation and application of test patterns is greatly simplified. Various test generation algorithms have been developed for shorted nets faults. The log(n+2) approach (where 'n' is the total number of nets), the 2*log(n) counting and complement counting algorithm, 2*n marching 0 and 1 patterns, and completely diagnosable patterns have been proposed to attack the static interconnect faults[1-4]. In addition to the static faults, crosstalk faults generated through high speed signal transmission become significant challenges for SoC interconnect testing[5-10]. A fast and accurate technique to evaluate the effectiveness of the test sets to detect crosstalk defects was developed[5]. Although linear feedback shift registers (LFSR) were extensively adopted to generate a few random patterns, the fault coverage was not satisfiable. Several deterministic and pseudorandom test pattern generators of embedding necessary patterns for crosstalk faults have been proposed[7,8], however the lengthy application time and area penalty become a big burden upon the growth of total number of nets, and the true necessity of those core patterns was not confirmed.

표 1. 정적고장 점검을 위한 2n 테스트 패턴 알고리즘
Table 1. 2n (n=6) March Test Patters for Static Interconnect Faults.

nets	Input vector											
net1	0	1	1	1	1	1	1	0	0	0	0	0
net2	1	0	1	1	1	1	0	1	0	0	0	0
net3	1	1	0	1	1	1	0	0	1	0	0	0
net4	1	1	1	0	1	1	0	0	0	1	0	0
net5	1	1	1	1	0	1	0	0	0	0	1	0
net6	1	1	1	1	1	0	0	0	0	0	0	1

In this paper, an efficient algorithm to generate interconnect test patterns is introduced along with simple BIST implementation. This paper is organized as follows. After describing some definitions and fault models in section II, 6n test patterns are introduced in section III. Our 4n+1 algorithm is precisely investigated in section IV, and its BIST implementation is depicted in section V followed by conclusions.

II. Definitions and Fault Models

We consider the following classes of dynamic as well as static fault models for interconnect nets on SoCs or boards.

Static faults include conventional *S-at-1* and *S-at-0*, *S-open* modeling any open net fault as either a pull-up or pull-down circuit, and Shorted Nets Faults of AND, OR, OPEN, DOMINATOR types.

Dynamic faults include *crosstalk* and *intermittent* type defects. Crosstalk faults can be classified into positive glitch, negative glitch, falling delay, and rising delay as

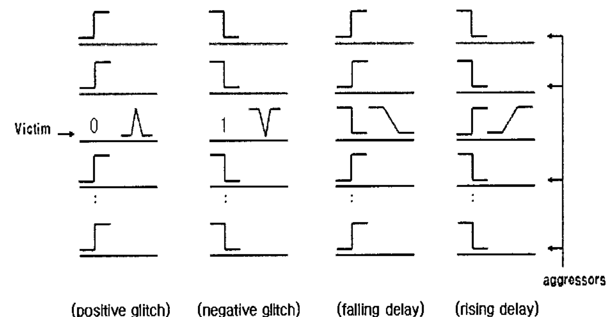


그림 1. Crosstalk 고장 모델
Fig. 1. Crosstalk fault model.

shown in Figure 1. The *victim* is the net on which the crosstalk fault occurs, and the *aggressor* is the net which triggers the crosstalk fault on the victim net. To effectively detect the crosstalk faults, proper values must be applied to the victim net with regard to different types of aggressor transitions[7,8].

The following definitions are used throughout the rest of this paper.

Definition 1: A *net* is defined as the interconnect wiring between output pins and input pins of chips or card. The *source* of a net is defined as chip output pins or card input pins which can drive signals on the net. Likewise *sink* of a net is defined as chip input pins or card output pins.

Definition 2: A *test pattern* is a set of logic values which need to be assigned to each boundary scan cell and card pin. In table 2, the 1st test pattern is '00000..0' which is the first column of the input vector.

Definition 3: A *test vector* is a sequence of test pattern applied to each net. In table 2, the test vector for the net1 is '0001..01' which is the net1 row of the input vector.

Definition 4: An *input vector* is defined to describe the stream of input patterns of which each element belongs to a different test pattern. An *output vector* is similarly used to illustrate the stream of output patterns corresponding to an input vector. In general, the fault free output vector is supposed to be the same as the input vector in a boundary scan designed board.

Definition 5: Two test patterns containing transitions are needed to detect a crosstalk fault of which the 1st pattern is called as a *head* and the 2nd pattern as a *tail*. According to the different types of crosstalk faults, set of test patterns is defined as pi (positive glitch), ni (negative glitch), fi (falling delay), and ri (rising delay), where 'i' indicates the victim net i. For example, in Table 2 p1 is a set of two test patterns to test positive glitch of net1, where '000000...0' is a head pattern and '011111...1' is a tail pattern.

Definition 6: Since some aggressors are physically located far away from the victim net, the crosstalk effect to the victim net can be negligible. Hence we will call this aggressor an *ineffective aggressor*.

III. Analysis of Test Patterns for Crosstalk faults

Test patterns to detect 4 different types of crosstalk faults are described in Table 2. Table 2(a) shows the test patterns to detect positive glitch defect on a net. Each pi for net i consists of head and tail patterns. If p1 test patterns of Table 2(a) are applied, net1 with '00' test vector is considered to be a victim net and the rest nets with '01' test vector are considered as aggressor nets. Similarly ith net with regard to pi, ni, fi, and ri test patterns becomes victim net and the other nets become aggressor nets.

How many test patterns are required to capture 4 different types of crosstalk faults for n number of nets? Starting from very loose upper bound, eventually we will derive very tight upper bound to minimize test application time.

In order to test crosstalk faults, head and tail two patterns are required for each pi, ni, fi, and ri, and test patterns must provide appropriate transitions to victim and aggressor nets depending on fault types. Since two test patterns are needed for 4 different types of crosstalk faults, total number of 8n test patterns are easily derived, and they become a very loose upper bound.

Let us derive the lower bound. Since 4 different crosstalk faults can occur on each net, 4n number of faults exist for n number of nets. Suppose consecutive test patterns can generate the appropriate transitions for 4n faults, 4n number of test patterns plus one more for the last net, that is 4n+1 number of test patterns, are able to detect the whole crosstalk faults. In summary, lower and upper bounds

표 2. 각각의 crosstalk 고장 점검을 위한 패턴
Table 2. Test Patterns for 4 Different Crosstalk Faults.

nets	Input vector					
net1	0	0	0	1	0	1
net2	0	1	0	0	0	1
net3	0	1	0	1	0	1
net4	0	1	0	1	0	1
net5	0	1	0	1	0	1
net6	0	1	0	1	0	1
net i	0	1	0	1	0	0
	p1		p2		pi	

(a) positive glitch

nets	Input vector					
net1	1	1	1	0	1	0
net2	1	0	1	1	1	0
net3	1	0	1	0	1	0
net4	1	0	1	0	1	0
net5	1	0	1	0	1	0
net6	1	0	1	0	1	0
net i	1	0	1	0	1	1
	n1		n2		ni	

(b) negative glitch

nets	Input vector					
net1	1	0	0	1	0	1
net2	0	1	1	0	0	1
net3	0	1	0	1	0	1
net4	0	1	0	1	0	1
net5	0	1	0	1	0	1
net6	0	1	0	1	0	1
net i	0	1	0	1	1	0
	f1		f2		fi	

(c) falling delay

nets	Input vector					
net1	0	1	1	0	1	0
net2	1	0	0	1	1	0
net3	1	0	1	0	1	0
net4	1	0	1	0	1	0
net5	1	0	1	0	1	0
net6	1	0	1	0	1	0
net i	1	0	1	0	0	1
	r1		r2		ri	

(d) rising delay

of the number of test patterns can be summarized as in Equation 1.

$$4n+1 <= \text{number of test patterns} <= 8n \qquad (1)$$

Instead of a loose upper bound, the focus is given to derive a very tight upper bound. Since each test pattern for pi, ni, fi, and ri crosstalk faults includes a head and a tail, respectively, by arranging the patterns as a chain probably the total number can be significantly reduced. As seen in Table 2 (c) test patterns to detect the falling delay f1 for net1 consist of a head '100000...0' and a tail '011111...1', and in Table 2 (d) it can be noted that the tail '011111...1' becomes the head of rising delay r1. Similarly, the tail pattern for r1 can be reused as the head pattern for f1, thus 3 test patterns can sufficiently test r1 then f1 or vice versa. It can be observed that there exist certain chain relations among head and tail patterns for pi, ni, fi, and ri. The relation for fi and ri can be graphically described as Figure 2(a). If the pi is added, Figure 2(b) relation can be obtainable among pi, ri, and fi faults, where pi can only transit to ri, and similarly Figure 2(c) relation for ni, fi, and

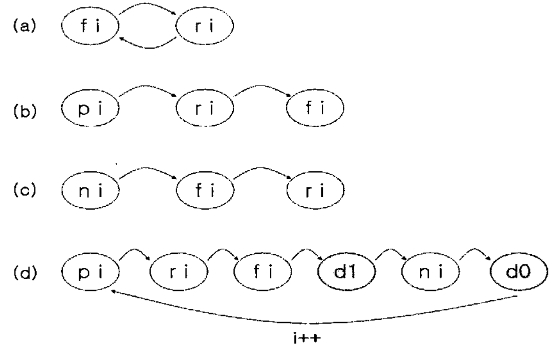


그림 2. crosstalk 고장 점검을 위한 패턴 변환 그래프
Fig. 2. Test pattern transition graph for crosstalk faults.

표 3. net 1에 대한 6n 테스트 패턴 알고리즘
Table 3. 6 Test Patterns for Crosstalk Faults on net 1.

nets	Input vector					
net1	0	0	1	0	1	1
net2	0	1	0	1	1	0
net3	0	1	0	1	1	0
net4	0	1	0	1	1	0
:	:	:	:	:	:	:
net i	0	1	0	1	1	0
	d0	p1	r1	f1	d1	n1

ri can be derived. In the sequel, only three different types of crosstalk faults can be detected by consecutively applying the tail and head test patterns. How can the remaining one fault be detected by augmenting Figure 2(b) or (c) with a minimal number of directed edges? In Figure 2(b) the fault ni(negative glitch on the net i) not detected can be captured by adding d1 vector with only 1 values like '111111...1' and d0 vector with only 0 values like '00000...0' as shown in Figure 2(d).

The last d0 vector in Figure 2(d) aimed to test net i is reused as a head pattern to capture positive glitch of net (i+1). Therefore, the whole test patterns for 4 different crosstalk faults on a net i can be derived with the chaining graph of Figure 2(d), and it is seen that 6 patterns are needed. For net (i+1) also 6 patterns are easily achievable. The 6 test patterns for net 1 is listed in Table 3.

Now it is shown that the upper bound of the test patterns for crosstalk faults becomes 6n instead of 8n as described in Equation 2.

$$4n+1 <= \text{number of test patterns} <= 6n \qquad (2)$$

Is 6n the tight upper bound? Because some nets are located too far from the others. There may not exist any crosstalk effect, and this analysis has been keenly performed in [10]. A reduced 4n+1 algorithm considering the distance among nets is investigated in the following section.

Ⅳ. Efficient 4n+1 Algorithm for Realistic Crosstalk Faults

The crosstalk effects of aggressor toward victim nets are different depending upon the distances among them. The closer the aggressor is to the victim, the stronger is the crosstalk effect. The aggressor nets giving effects to a victim net are classified into affecting lines just neighbored and supporting lines indirectly effective as shown in Figure 3. The level of supporting line is determined upon the distance from the victim net, and it has been analyzed that the crosstalk effect of supporting lines with levels of more than 3 would become negligible[10]. The neighboring information of the interconnect nets are obtainable from the net list file generated through physical layout. Now, our focus is given to drive a minimal number of test patterns whose number is less than 6n, and which at least provide necessary values to the effective aggressor nets.

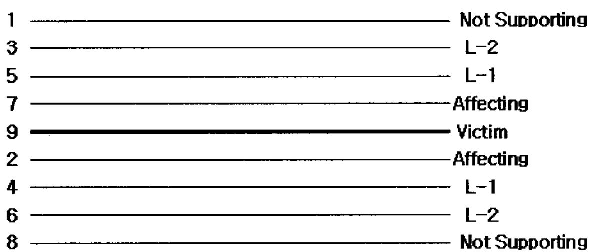


그림 3. Affecting line 과 supproting line
Fig. 3. Affecting and supporting lines.

As mentioned previously the lower bound of the number of crosstalk test patterns is 4n+1, and the crucial difference between 6n and 4n+1 was to insert d0 and d1 thus consecutive testing of pi, ri, fi, and ni became possible. If only the affecting and supporting lines with a level less than 3 are explicitly targeted to provide proper crosstalk test patterns, without inserting d0 and d1 patterns 4n+1 can be augmented to generate consecutive test patterns for 4 different faults. The following properties are extracted from the head and tail patterns for crosstalk faults.

① Tail patterns for pi and fi are the same as d1 except for only one value.

② Tail patterns for ri and ni are the same as d0 except for only one value.

③ The tail for ri can be reused as the head for pj, only if net i behaves as an ineffective aggressor for the victim net j in that the

표 4. net = 9일때 4n+1 테스트 패턴 예
Table 4. 4n+1 Test Patterns for n=9 Nets.

nets	Input vector																																			
net1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
net2	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0
net3	0	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0
net4	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0
net5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
net6	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0
net7	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0
net8	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0
net9	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0
Test	p1	r1	p6	r6	p2	r2	p7	r7	p3	r3	p8	r8	p4	r4	p9	r9	p5	r5	f5	n1	f1	n6	f6	n2	f2	n7	f7	n3	f3	n8	f8	n4	f4	n9	f9	n5	

consecutive two patterns, and pj for this case is defined as $p_{j,i}$.

④ The tail for fi can be reused as the head for nj, only if net i behaves as an ineffective aggressor for the victim net j in that the consecutive two patterns, and nj in this case is defined as $n_{j,i}$.

By considering the above properties, the consecutive test patterns for crosstalk faults are derived as in Equations (3) and (4).

$$pa \rightarrow ra \rightarrow pb,a \rightarrow rb \rightarrow pc,b \rightarrow rc \rightarrow \ldots \rightarrow pv,e \rightarrow rv \quad (3)$$

$$na \rightarrow fa \rightarrow nb,a \rightarrow fb \rightarrow nc,b \rightarrow fc \rightarrow \ldots \rightarrow nv,e \rightarrow fv \quad (4)$$

In $p_{j,i,}$, in order for net i to become ineffective for the victim net j they must be physically separated by at least 3 levels[10]. Therefore, in Equations (3) and (4) nets i, j, and k must be separated by more than 3 levels for $p_{i,j}$ and $n_{i,k}$. The separation condition among ineffective aggressor and victim nets is satisfied by arranging patterns as in Equation (5).

$$p_i \rightarrow r_i \rightarrow p_{\lceil n/2 \rceil +i,i} \rightarrow r_{\lceil n/2 \rceil +i} \rightarrow p_{i+1,\lceil n/2 \rceil +i} \rightarrow r_{i+1} \rightarrow p_{\lceil n/2 \rceil +i+1,i+1} \rightarrow r_{\lceil n/2 \rceil +i+1} \rightarrow \ldots$$
$$n_i \rightarrow f_i \rightarrow n_{\lceil n/2 \rceil +i,i} \rightarrow f_{\lceil n/2 \rceil +i} \rightarrow n_{i+1,\lceil n/2 \rceil +i} \rightarrow f_{i+1} \rightarrow n_{\lceil n/2 \rceil +i+1,i+1[} \rightarrow f_{\lceil n/2 \rceil +i+1} \rightarrow \ldots \quad (5)$$

Observation: In Equation (5) the minimum distance among victim and ineffective aggressor nets regarding certain crosstalk test patterns is kept as ($\lceil n/2 \rceil +i$) - (i+1) = $\lceil n/2 \rceil -1$. If the total number of nets (n) is more than 8, the distance becomes greater than 3, thus 4n+1 patterns instead of 6n can completely capture crosstalk faults on the supporting lines[10].

For example, 4n+1 test patterns for 9 interconnect nets are listed in Table 4, where the victim and ineffective aggressor nets are shown as shaded and the minimum distance among them is observed as 4. It is not difficult to figure out that 4n+1 patterns embed 2n march patterns known to test all the static interconnect faults. Therefore, 4n+1 patterns are sufficient to test crosstalk faults as well as all the static interconnect faults. The crosstalk and static fault coverages for 6n, pseudo random, and 4n+1 test patterns are listed in Table 5 where LI-BIST is especially weak in testing static interconnect faults.

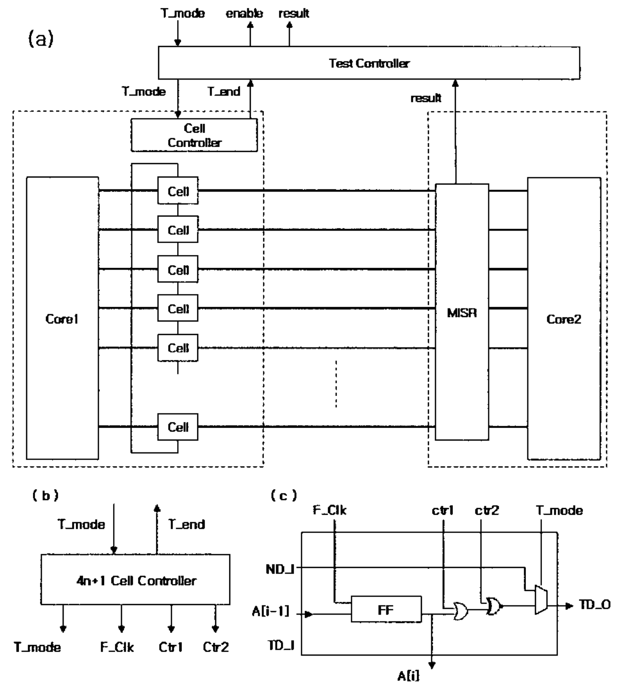


그림 4. (a) 연결선 고장점검을 위한 BIST구조
(b) 4n+1 알고리즘을 위한 셀 컨트롤러
(c) 4n+1 테스트 패턴 생성 셀

Fig. 4. (a) BIST architecture for interconnect test.
(b) Cell controller for 4n+1 algorithm.
(c) Primary cell to generate 4n+1 test patterns.

표 5. 각 테스트 알고리즘에 따른 고장점검율 및 고장점검 내용

Table 5. Crosstalk and Static Fault Coverages for Different Test Algorithms.

Test algorithm	types	crosstalk	static
6n[7]	deterministic	100%	100%
LI-BIST[8]	random	99.7%	91.36%
4n+1	deterministic	100%	100%

V. BIST Implementation of 4n+1 Algorithm

Because 4n+1 algorithm consists of repetition of 4 different simple states, the BIST implementation incurs a slightly more area penalty compared with conventional random LFSR BIST[4]. If all the n nets

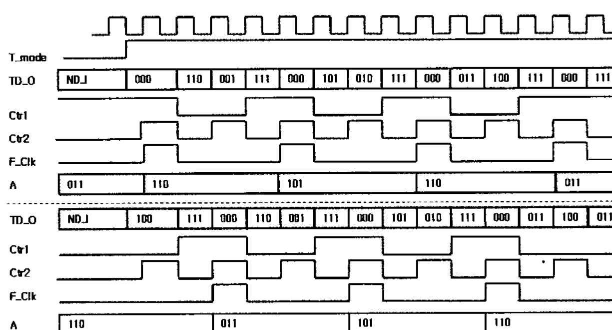


그림 5. 4n+1 테스트 패턴 생성을 위한 신호
Fig. 5. Simulation results in generating 4n+1 test patterns

표 6. 각 테스트 알고리즘에 대한 BIST 오버헤드 비교
Table 6. BIST Area Overheads for Different Interconnect Test Algorithms.

total nets	16	100
6n cell	520	2240
LI-BIST[8]	About 80% of 6n cell design	
4n+1 cell	505	2188

Dimension : gate count

are looked as 2 separated sets, a set from net 1 to net ⌈n/2⌉, and the other set from net ⌈n/2⌉+1 to net n, simple states of either d0, shift, NOT, and d1 or d1, shift, NOT, and d0 are repeated in 4n+1 algorithm. The primary cell for the generation of 4n+1 test patterns is shown in Figure 4, and by controlling the ctr1, ctr2, F_Clk, and A[i] signals through the cell controller as shown in Figure 5. 4n+1 BIST logic can be simply implemented. Since the operations of 6n algorithm is very regular as 4n+1, the area penalty for both algorithms are similar. Table 6 describes the area overhead of random (LI-BIST), 6n, and 4n+1 algorithms and Synopsys synthesis tool was extensively used for the analysis. In both cases for 16 and 100 nets, the random BIST resulted in about a 20% less area penalty than 6n, and 4n required 90% overhead of 6n algorithm.

Ⅳ. Conclusions

In this paper an efficient 4n+1 test algorithm, shorter than 6n algorithm known to be necessary and sufficient patterns, was proposed to capture crosstalk faults on interconnects separated within effective distance[10]. 4n+1 algorithm is a minimal number of test patterns to capture crosstalk as well as static faults on SoC or board interconnects, and the area penalty incurred in BIST implementation is comparable to simple random techniques.

References

[1] A. Hassan and J. Rajski and V.K. Agrawal, "Testing and Diagnosis of Interconnects using Boundary Scan Architecture" Proceedings International Test Conference, pp.126-137. 1988.

[2] W. T. Cheng, J. L. Lewandowski and E. Wu, " Optimal Diagnostic Methods for Wiring Interconnects" IEEE Transactions on Computer-Aided Design, Vol 11, No. 9, pp. 1161-1166, Sept. 1992.

[3] N. Jarwala and C.W. Yau, "A new framework for analyzing test generation and diagnosis algorithms for wiring interconnects" Test Conference, Proceedings, International, pp. 63 - 70, Aug. 1989.

[4] Yongjoon Kim; Hyun-don Kim; Sungho Kang, "A new maximal diagnosis algorithm for interconnect", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Volume 12, Issue 5, pp. 532 - 537, May 2004.

[5] Yi Zhao; Dey, S., "Fault-coverage analysis techniques of interconnects crosstalk in chip", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , Volume: 22, pp. 770 - 782, June, 2003.

[6] M. Cuviello, S.Dey, X.Bai, and Y.Zhao, "Fault modeling and simulation for crosstalk in system-on-chip interconnects", In Proc. Int. Conf. Computer-Aided Design, pages 297-303, Nov. 1999.

[7] Xiaoliang Bai and S. Dey and J. Rajski, "Self-test methodology for at-speed test of crosstalk in chip interconnects," Design Automation Conference, Proceedings 2000. 37th, pp. 619 - 624, June 5-9, 2000.

[8] K. Sekar and S. Dey, "LI-BIST: a low-cost self-test scheme for SoC logic cores and interconnects," VLSI Test Symposium, (VTS 2002). Proceedings 20th IEEE , pp. 417 - 422, 28 April-2 May 2002.

[9] R. Pendurkar and A. Chatterjee and Y. Zorian, "Switching activity generation with automated BIST synthesis for performance testing of interconnects," Computer-Aided Design of

Integrated Circuits and Systems, IEEE Transactions on, Volume 20, pp. 1143 - 1158, Sept. 2001.

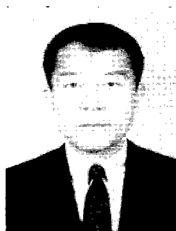
[10] Sirisaengtaksin and Sandeep K. Gupta "Enhanced crosstalk fault model and methodology to generate tests for arbitrary inter-core interconnect topology" Test Symposium, 2002. (ATS '02). Proceedings of the 11th Asian, 18-20 pp. 163 - 169, Nov. 2002.

Authors

민 병 우(학생회원)
2003년 한양대학교 전자컴퓨터공학과 학사 졸업.
2005년 한양대학교 컴퓨터공학과 석사 졸업.
2005년~현재 삼성전자 메모리사업부 플래쉬솔루션 개발팀
<주관심분야: SoC 테스팅, 메모리BIST, ASIC>

송 재 훈(학생회원)
2000년 한양대학교 전자컴퓨터공학과 학사 졸업.
2002년 한양대학교 컴퓨터공학과 석사 졸업.
2003년 서울대학교 SoC 설계센터 연구원.
2004년~현재 한양대학교 컴퓨터 공학과 박사 과정.
<주관심분야: SoC 테스팅, DFT, 테스트 패턴 압축>

이 현 빈(학생회원)
2001년 한양대학교 전자컴퓨터공학과 학사 졸업.
2002년 한양대학교 컴퓨터공학과 석사 졸업.
2003년~현재 한양대학교 컴퓨터공학과 박사 과정.
<주관심분야: SoC 테스팅, ASIC 설계, 네트워크 시스템 설계>



박 성 주 / 교신저자(정회원)
1983년 한양대학교 전자공학과 학사.
1983년~1986년 금성사 소프트웨어 개발.
1992년 Univ. of Massachusetts 전기 및 컴퓨터공학과 박사졸업.
1992년~1994년 IBM Microelectronics 연구스텝,
1992년~1994년 한양대학교 전자컴퓨터공학부 정교수.
<주관심분야: 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC설계, 고속 신호처리 시스템 설계, 그래프이론 등>