논문 2007-44SD-12-11

Crosstalk 고장 점검을 위한 효과적인 연결선 테스트 패턴 생성 알고리즘에 관한 연구

(An Efficient Interconnect Test Pattern Generation Algorithm for Crosstalk Faults)

한 주 희*, 송 재 훈*, 이 현 빈*, 김 진 규*, 박 성 주**

(Juhee Han, Jaehoon Song, Hyunbean Yi, Jinkyu Kim, and Sungju Park)

요 약

고성능의 칩을 설계함에 있어 연결선 사이의 크로스토크 고장은 무시할 수 없는 요인이 되었다. 본 논문에서는 칩 및 보드 레벨에서의 연결선 테스트를 위한 효과적인 테스트 패턴 알고리즘을 제시한다. 크로스토크 고장 점검율이 100%인 기존 6n 알고리즘을 분석하고 실질적으로 크로스토크 영향을 주는 net를 고려하여 보다 적은 패턴으로 동일한 고장 점검율을 얻는 새로운 알고리즘을 제안한다.

Abstract

The effect of crosstalk errors is most significant in high-performance circuits. This paper presents effective test patterns for SoC and Board level interconnects considering actual effective aggressors. Initially '6n' algorithm, where 'n' is the total number of interconnect nets, is analyzed to detect and diagnose 100% crosstalk faults. Then, more efficient algorithm is proposed reducing the number of test patterns significantly while maintaining complete crosstalk fault coverage

Keywords : Crosstalk, Fault Model, Interconnect Test

I. 서 론

초고밀도 반도체 직접 기술의 발전은 시스템 설계 시 미리 설계된 여러 개의 검증된 코아들을 이용하여 하나의 시스템 칩으로 구현할 수 있는 SoC (System-on-a-chip) 기술을 가능하게 하였다. SoC 기술은 작은 면적과 저전력을 통해 보다 많은 기능을 통합하여 수행할 수 있는 장점을 가진다. 시스템이 복잡해지고 내부 코아의 수가 증가함에 따라 SoC의 신뢰성은 코아 내부뿐 아니라 코아와 코아 사이의 연결선들의 안전성에도 크게 영향을 받게 되었다. 또한 나노 공정의 발달로 인해 연결선상에서 일어나는 정적 및 동적 고장의 종류가 다양해지면서 SoC 테스트 설계 기술에 대한 요구가 더욱 증대되고 있다. 특히 고속의 신호 전송 시 발생되는 연결선 사이의 크로스토크 현상은 정적 고장과 더불어 큰 이슈가 되고 있으며[1~4], 제품의 수율을 높이면서 비용은 최소화하기 위해서는 최소한의 테스트 패턴으로 효과적인 테스트를 할 수 있는 기술을 필요로 한다.

연결선에서 발생되는 정적고장은 고착고장(stuck-at fault), 개방고장(open-net fault), 단락고장(shorted-net fault)이 있다. 고착고장은 연결선이 0 또는 1로 고정되는 현상이고, 개방고장은 연결선 일부가 개방되는 현상이다. 단락고장은 두 개 이상의 연결선이 서로 영향을

* 학생회원, 한양대학교 컴퓨터공학과
(Department of Computer Science & Engineering, Hanyang University)
** 정회원, 한양대학교 전자컴퓨터공학부
(Department of Computer Science & Engineering, Hanyang University)
※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11038-0 (2007)) 지원으로 수행되었음.
접수일자: 2007년7월16일, 수정완료일: 2007년11월26일

미쳐 wired-AND/wired-OR/Dominant 고장이 발생하는 현상이다. 이러한 정적고장을 테스트하기 위해 연결선 테스트 알고리즘이 연구되어 왔으며, 이런 알고리즘은 고장에 대한 검출(detection) 능력뿐만 아니라 고장의 위치진단(diagnosis) 능력까지 필요로 한다. 이러한 연결선 테스트를 수행하기 위해 제안된 log(n+2) 알고리즘은 카운팅 시퀀스를 이용하여 쉽게 고장 검출을 할 수 있다[5]. 2log(n) 알고리즘은 고장검출 뿐 아니라 부분적인 고장진단이 가능하도록 하였고[6], 2(n) 알고리즘은 각 테스트 벡터마다 ID를 부여함으로써 완벽한 고장 검출 및 진단을 할 수 있는 방법을 제시했으며[7], 패턴의 개수를 줄인 n+1 알고리즘까지 제안되었다[8].

연결선 상에서의 동적고장 중 대표적인 크로스토크는 인접한 연결선들에서의 로직 값 천이에 따라 glitch, delay, speed up 고장 등이 발생되는 현상을 말한다. 이러한 크로스토크 고장을 적은 테스트 패턴을 이용하여 검출하도록 하는 6n 알고리즘이 제안되었다[3].

본 논문은 대상 공정 파라미터를 이용하여 연결선 간의 coupling capacitance 값을 추출하고 실질적으로 크로스토크 영향을 주는 net을 고려[10]하여 100% 혹은 그에 가까운 크로스토크 고장 점검율을 얻으면서 패턴의 개수를 최소화 하는 효과적인 테스트 패턴 알고리즘을 제안한다.

구성은 본론에서 크로스토크 고장의 종류 및 고장 검출 패턴, 제안된 알고리즘에 대해 알아보고 III장 결론의 순서로 되어 있다.

II. 본 론

1. 크로스토크 고장 모델

크로스토크 고장 모델은 크게 glitch, delay, speed up 동적 고장모델로 구분할 수 있다[10]. 그림 1.은 세분화된 크로스토크 고장모델을 나타내고 있다. victim은 크로스토크 고장이 발생하는 net으로, aggressor는 victim에 크로스토크를 유발시키는 net으로 정의한다. glitch 크로스토크는 positive glitch, negative glitch로 구분 되는데 victim net에서는 일정한 값 (0 또는 1)을 유지하는 반면 주변의 모든 aggressor들이 동일한 방향으로 천이를 일으키는 경우, victim에 의도 하지 않은 glitch가 발생하는 현상이다. Delay 크로스토크는 victim과 aggressor들의 천이가 반대 방향으로 일어나게 되는 경우에 victim의 천이가 느려지는 현상이며 speed up 크로스토크는 victim과 aggressor의 천이가 모두 동일한 방향으로 이루어지는 경우에 victim에서의 천이가 너무 빨리 일어나 hold time 등의 문제를 가져오는 현상이다. delay, speed up 역시 각각 rising, falling의 두 경우에 대해서 고장모델링을 한다.

크로스토크를 효과적으로 검출하기 위해서는 다양한 유형의 aggressor들의 천이에 대하여 victim에 필요한 값을 인가하여야한다[3~4].

2. 크로스토크 테스트를 위한 테스트 패턴

크로스토크 테스트를 위해 다음과 같이 정의하기로 한다.

1) Net: 칩의 연결선 (interconnect wire)을 말한다.

2) 테스트 패턴(Test Pattern): 테스트 패턴은 각 연결선마다 할당하기 위해 필요한 논리 값의 집합을 말한다. 예로 표1. (a)의 첫 번째 테스트 패턴은 첫 번째 열로서 '00000...0' 이다.

3) 테스트 벡터(Test Vector): 각 net에 해당하는 테스트 패턴의 요소로 구성된 스트림 집합을 말한다. 예로 표1. (a)의 net1에 해당하는 테스트 벡터는 첫 번째 행으로서 '0001...01' 이다.

4) 크로스토크 고장 점검 패턴: 하나의 크로스토크

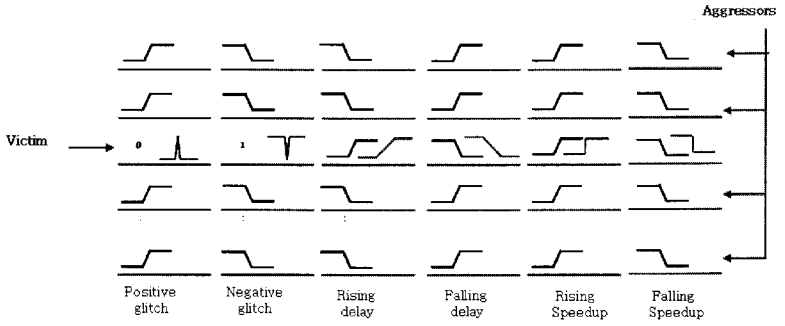


그림 1. 크로스토크 고장 모델

Fig. 1. Crosstalk faults model.

고장을 점검하기 위해서는 두 개의 테스트 패턴이 필요하다. 첫 번째 테스트 패턴은 헤드(head), 두 번째 테스트 패턴은 테일(tail)이라 한다. 검출 가능한 크로스토크 고장에 따른 테스트 패턴을 pi(positive glitch), ni(negative glitch), ri(rising delay), fi(falling delay), sri(rising speedup), sfi(falling speedup)로 정의한다. 여기서 i는 i번째 net을 나타낸다. 예로 표 1. (a)에서 p1은 첫 번째 net의 positive glitch를 점검하기 위한 테스트 패턴으로서 헤드 '000000...0'와 테일 '011111...1'로 구성된다.

표 1.은 6가지 종류의 크로스토크 고장 점검을 위한 테스트 패턴을 보여준다. 표 1. (a)는 각 net의 positive glitch를 점검하기 위한 테스트 패턴이다. p1은 첫 번째 net을, p2는 두 번째 net을, pi는 i번째 net의 positive glitch를 점검하기 위한 것으로 헤드와 테일로 구성된다. victim에는 벡터 '00'이 입력되고 aggressor에는 벡터 '01'이 입력된다. 임의의 i에 대한 pi의 i번째 net에 해당하는 테스트 벡터는 victim으로, 나머지 net에 해당하는 테스트 벡터는 aggressor로 작용하며 ni, ri, fi, sri, sfi에도 동일하게 적용된다.

n개의 net에 대해 상기 6종류의 크로스토크 고장을 점검하기 위해 필요한 테스트 패턴의 개수는 다음과 같다.

크로스토크 고장 점검을 위해서는 각 테스트 벡터들의 천이를 통해서 victim과 aggressor를 만들 수 있다. 예로 표 1. (a)의 positive glitch를 점검하기 위한 테스트 패턴 개수는 p1, p2, ... , pn 마다 헤드/테일로 구성된 2개의 테스트 패턴이 필요함으로 총 2n개 이다. 동일한 방식이 negative glitch, rising delay, falling delay에도 적용되어 4가지 크로스토크 고장 점검을 위해 필요한 테스트 패턴 개수는 8n이 된다. rising speed up과 falling speed up의 경우는 동일한 패턴을 각 net의 고장 점검에 사용하기 때문에 6가지 크로스토크 고장을 점검하기 위해 필요한 테스트 패턴의 수는 8n+2가 된다.

표 1. 크로스토크 고장 점검을 위한 패턴
Table 1. Test patterns for crosstalk faults.

(a) positive glitch

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	0	0	0	1	0	1
net 2	0	1	0	0	0	1
net 3	0	1	0	1	0	1
net 4	0	1	0	1	0	1
net 5	0	1	0	1	0	1
net 6	0	1	0	1	0	1
net i	0	1	0	1	0	0
	p1		p2		pi	

(b) negative glitch

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	1	1	1	0	1	0
net 2	1	0	1	1	1	0
net 3	1	0	1	0	1	0
net 4	1	0	1	0	1	0
net 5	1	0	1	0	1	0
net 6	1	0	1	0	1	0
net i	1	0	1	0	1	1
	n1		n2		ni	

(c) rising delay

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	0	1	1	0	1	0
net 2	1	0	0	1	1	0
net 3	1	0	1	0	1	0
net 4	1	0	1	0	1	0
net 5	1	0	1	0	1	0
net 6	1	0	1	0	1	0
net i	1	0	1	0	0	1
	r1		r2		ri	

(d) falling delay

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	1	0	0	1	0	1
net 2	0	1	1	0	0	1
net 3	0	1	0	1	0	1
net 4	0	1	0	1	0	1
net 5	0	1	0	1	0	1
net 6	0	1	0	1	0	1
net i	0	1	0	1	1	0
	f1		f2		fi	

(e) rising speed up

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	0	1	0	1	0	1
net 2	0	1	0	1	0	1
net 3	0	1	0	1	0	1
net 4	0	1	0	1	0	1
net 5	0	1	0	1	0	1
net 6	0	1	0	1	0	1
net i	0	1	0	1	0	1
	sr1		sr2		sri	

(f) falling speed up

	Input Vector					
nets	head	tail	head	tail	head	tail
net 1	1	0	1	0	1	0
net 2	1	0	1	0	1	0
net 3	1	0	1	0	1	0
net 4	1	0	1	0	1	0
net 5	1	0	1	0	1	0
net 6	1	0	1	0	1	0
net i	1	0	1	0	1	0
	sf1		sf2		sfi	

3. 6n 알고리즘

6n 알고리즘은 높은 고장 점검율은 유지하며 테스트 패턴의 수를 줄이기 위해 제안된 알고리즘이다. 임의의 i 번째 net의 크로스토크 고장 점검을 위해 필요한 테스트 패턴 pi, ni, ri, fi는 각각 헤드와 테일로 구성되어 있다. 하나의 고장모델에 대한 테스트 패턴의 헤드/테일은 다른 고장모델의 테일/헤드가 될 수 있기 때문에 헤드-테일 관계의 중첩성을 이용하면 테스트 패턴의 수를 줄일 수 있다. 예를 들어, 표 1. (a)에서 p1의 테일

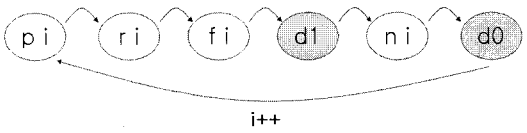


그림 2. 크로스토크 고장 점검을 위한 6n 패턴
Fig. 2. 6n test patterns for crosstalk faults.

표 2. net 1 점검을 위해 필요한 테스트 패턴 (6n 알고리즘)
Table 2. Test patterns for crosstalk faults on net 1. (6n Algorithm)

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
	p1	r1	f1	n1		

'011111...1'은 표 1. (c)에서 r1의 헤드 '011111...1'과 같다. 또한 r1의 테일 '100000...0'은 f1의 헤드와 같게 됨으로 3개의 테스트 패턴을 통해서 r1, f1 또는 f1, r1 순으로 고장 점검을 할 수 있다.

net의 개수가 n이고 임의의 net i에 대하여 pi, ni, ri, fi 사이의 관계는 다음과 같다.

pi의 테일은 ri의 헤드와 같고, ri의 테일은 fi의 헤드와 동일하다. 이러한 관계를 이용하여 6n 알고리즘은 그림 2.와 같이 제안 되었다. d1과 d0는 각각 전체 패턴이 1 혹은 0으로 구성된 경우를 의미한다. 표 2.는 net 1에 대한 pi, ni, ri, fi 크로스토크 고장 점검을 위한 패턴 예 이다. 하나의 net에 대한 고장 점검을 위해 필요한 테스트 패턴의 수가 6개 이므로 n개의 net에 대해 필요한 테스트 패턴의 개수는 6n이 됨을 알 수 있다[3].

6n 알고리즘은 적은 테스트 패턴으로 크로스토크 고장을 검출할 수 있는 알고리즘이나 하나의 패턴으로 하나의 고장만을 검사하기 때문에 패턴의 사용이 비효율적이다. 그리고 speed up 고장의 경우는 고려하지 못한 문제점을 가지고 있다.

4. 제안된 테스트 패턴 알고리즘

크로스토크 효과는 인접한 연결선 사이의 coupling capacitance에 의해 생기게 되며 두 선 사이의 거리가 멀어질수록 coupling capacitance 값은 작아지게 된다. 하나의 victim에 대한 aggressor들을 victim과의 거리에 따라 level 1, level 2, level 3으로 나누었다. level이 커질수록 victim에 미치게 되는 영향은 작아지며 그 영향이 무시해도 좋을 정도로 작아지게 될 경우는 ineffective aggressor로 한다. 그림 3.에서는 level 4 이상의 aggressor는 ineffective aggressor로 간주 되었다. 이 수치는 실험적인 결과에 근거한 것이며 위 실험은

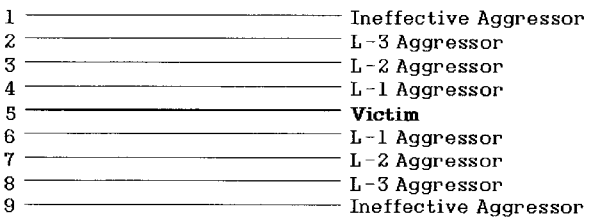


그림 3. Level에 따른 Aggressor 분류
Fig. 3. Aggressor classification according to level.

표 4. speed up 고장 검출
Fig. 4. Test patterns for speed up crosstalk faults.

nets	Input Vector 1							
net 1	0	1	0	0	1	0	1	1
net 2	0	1	0	1	0	1	1	0
net 3	0	1	0	1	0	1	1	0
net 4	0	1	0	1	0	1	1	0
net 5	0	1	0	0	1	0	1	1
net 6	0	1	0	1	0	1	1	0
net 7	0	1	0	1	0	1	1	0
net 8	0	1	0	1	0	1	1	0
	sri sfi							

TSMC 0.18 um 공정 파라미터로 이루어 진 것이다[10].

본 논문은 6n 알고리즘에 level 4 이상의 aggressor는 ineffective aggressor가 된다는 실험 결과를 적용하여 첫 번째 victim과 level 4이상 떨어져 있는 ineffective aggressor를 새로운 victim으로 설정하는 알고리즘을 제안하였다. 즉, 하나의 패턴으로 다중 victim에 대한 크로스토크 고장 검출이 가능한 알고리즘이다.

표 3.은 net의 개수가 8개인 경우에 제안된 알고리즘을 적용한 테스트 패턴을 보여준다. net 사이의 거리가 4 이상 되어 서로간의 천이가 크로스토크 고장 발생에 거의 영향을 미치지 않게 되는 net들을 동시에 테스트 하였다. 입력 벡터 1은 net 1과 net 5에서의 고장 검출

표 3. 8개의 net에 적용된 제안된 알고리즘
Table 3. 6n test patterns for n = 8 nets

nets	Input Vector 1						Input Vector 2						Input Vector 3						Input Vector 4					
net 1	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0	0	1	0	1	1	0	0	1	0	1	1	0
net 2	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0	0	1	0	1	1	0
net 3	0	1	0	1	1	0	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0
net 4	0	1	0	1	1	0	0	1	0	1	1	0	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**
net 5	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0	0	1	0	1	1	0	0	1	0	1	1	0
net 6	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0	0	1	0	1	1	0
net 7	0	1	0	1	1	0	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**	0	1	0	1	1	0
net 8	0	1	0	1	1	0	0	1	0	1	1	0	0	1	0	1	1	0	**0**	**0**	**1**	**0**	**1**	**1**
	p1,p5 r1,r5 f1,f5 n1,n5						p2,p6 r2,r6 f2,f6 n2,n6						p3,p7 r3,r7 f3,f7 n3,n7						p4,p8 r4,r8 f4,f8 n4,n8					

을 동시에 할 수 있도록 만들어 졌으며 입력벡터 2는 net 2와 net 6, 입력벡터 3은 net 3과 net 7 그리고 입력벡터 4는 net 4와 net 8에서의 고장 점검을 동시에 할 수 있도록 만들어 졌다. 하나의 패턴으로 하나의 net에 대한 고장 검출을 하는 것이 아니라 동시에 여러 net에서의 고장 검출이 가능하기 때문에 전체 net을 검사할 때 필요하게 되는 패턴의 수는 현저하게 감소한다.

사용되는 패턴의 수는 다음과 같다.

4i+1 번째 다중 victim의 고장 점검을 위한 패턴의 수: 6개
4i+2 번째 다중 victim의 고장 점검을 위한 패턴의 수: 6개
4i+3 번째 다중 victim의 고장 점검을 위한 패턴의 수: 6개
4i+4 번째 다중 victim의 고장 점검을 위한 패턴의 수: 6개
사용된 전체 패턴의 수 : 6 X 4 = 24개

i=0, 1, 2, ... (k-1), k= $\lceil$ *# of net / 4* $\rceil$
d : multiple victim 간의 거리

n개의 net에 대하여 크로스토크 고장 점검을 위하여 필요한 패턴의 수는 24개 이다. 따라서 패턴의 개수가 연결선의 개수에 의존적인 기존 알고리즘 [3][11]에 비해 효율적인 테스트가 가능하다.

또한 본 논문은 rising speedup, falling speedup 고장에 대해서도 고려한다. rising speedup, falling speedup 고장 검출을 위해서 표 4.와 같이 테스트 패턴 시작 부분에 d0, d1을 추가 시켰다. 모든 net이 rising speedup, falling speedup의 victim이 되며 rising/falling speedup 고장은 순차적으로 검출 된다.

최종적으로 speed up 고장 검출까지 고려한 테스트 패턴의 수는 26개 이다. 따라서 제안된 알고리즘은 각 net에 대하여 최소한의 패턴으로 positive glitch, negative glitch, rising delay, falling delay, rising speed up, falling speed up 고장을 검출할 수 있는 가장 효율적인 알고리즘이 될 수 있다.

III. 결 론

본 논문에서는 연결선 고장 점검 시 크로스토크와 같은 동적 고장점검을 목적으로 최소한의 패턴으로 positive glitch, negative glitch, rising delay, falling delay, rising speed up, falling speed up 고장점검을 할 수 있는 테스트 패턴 생성 알고리즘을 제안하였다. 제안한 알고리즘은 서로 간의 크로스토크 영향이 거의 없는 연결선을 동시에 테스트 하도록 한 다중 victim 테스트 패턴 알고리즘으로 연결선의 수와 관계없이 26개의 패턴으로 모든 크로스토크 고장을 검출 할 수 있다.

참 고 문 헌

[1] H. Zhou and D. F. Wang, "Global Routing with Crosstalk Constaints", in Proc. IEEE Design Automation Conf., 1988, pp. 374-377.

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

[3] X. Bai, S. Dey and J. Rajski, "Self-test methodology for at-speed test of crosstalk in chip interconnects," in Proc. ACM/IEEE Design Automation Conf., 2000, pp. 619 - 624.

[4] K. Sekar and S. Dey, "LI-BIST: a low-cost self-test scheme for SoC logic cores and interconnects," in Proc. VLSI Test Symp., 2002, pp. 417 - 422.

[5] A. Hassan, J. Rajski, and V.K. Agrawal, "Testing and diagnosis of interconnects using boundary scan architecture," in Proc. IEEE Int. Test Conf., 1988, pp.126-137.

[6] W. T. Cheng, J. L. Lewandowski, and E. Wu, "Optimal diagnostic methods for wiring interconnects," IEEE Trans. on Computer-Aided Design, vol. 11, No. 9, pp. 1161-1166, Sept. 1992.

[7] N. Jarwala and C. W. Yau, "A new framework for analyzing test generation and diagnosis algorithms for wiring interconnects," in Proc. IEEE Int. Test Conf., 1989, pp. 63 - 70.

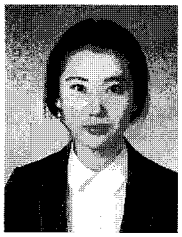
[8] Sungju Park, "A new complete diagnosis patterns for wiring interconnects," in Proc. IEEE Design Automation Conf. 1996, pp. 203 - 208.

[9] R. Pendurkar, A. Chatterjee, and Y. Zorian, "Switching activity generation with automated BIST synthesis for performance testing of interconnects," IEEE Trans. on Computer-Aided Design, vol. 11, No. 9, pp. 1143 - 1158, Sept. 2001.

[10] W. Sirisaengtaksin and S. K. Gupta "Enhanced crosstalk fault model and methodology to generate tests for arbitrary inter-core interconnect topology," in Proc. IEEE Asian Test Symp., 2002, pp. 163 - 169.

[11] P. Min et. al, "Efficient Interconnect Test Patterns for Crosstalk and Static Faults", IEEE Trans. on Computer-Aided Design, vol. 25, No. 11, pp. 2605 - 2608, Nov. 2006.

저 자 소 개



한 주 희(학생회원)

2005년 한양대학교 컴퓨터공학과 학사 졸업.

2006년~현재 한양대학교 컴퓨터 공학과 석사 재학중.

<주관심분야 : VLSI 설계, SoC DFT, 테스팅 Crosstalk>

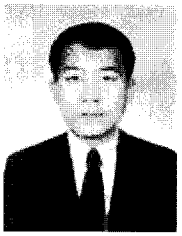
이 현 빈(학생회원)

2001년 한양대학교 전자컴퓨터 공학과 학사 졸업.

2003년 한양대학교 컴퓨터공학과 석사 졸업.

2007년 한양대학교 컴퓨터공학과 박사 졸업.

<주관심분야 : SoC 테스팅, ASIC 설계, 네트워크 시스템 설계>



박 성 주(정회원)

1983년 한양대학교 전자공학과 학사 졸업.

1983년~1986년 금성사 소프트 웨어개발 연구원.

1992년 Univ. of Massachusetts 전기 및 컴퓨터공학과 박사 졸업.

1992년~1994년 IBM Microelectronics 연구스텝.

1994년~현재 한양대학교 전자컴퓨터공학부 정교수

<주관심분야 : 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC 설계, 고속 신호처리 시스템 설계, 그래프 이론>

송 재 훈(학생회원)

2000년 한양대학교 전자컴퓨터 공 학과 학사 졸업.

2002년 한양대학교 컴퓨터공학과 석사 졸업.

2003년 서울대학교 SoC 설계 센터 연구원

2004년~현재 한양대학교 컴퓨터 공학과 박사 재학중

<주관심분야 : SoC 테스팅, DFT, 테스트 패턴 압축>

김 진 규 (학생회원)

2006년 한양대학교 컴퓨터 공학과 학사 졸업.

2006년-현재 한양대학교 컴퓨터 공학과 석사 과정 중.

<주관심분야: 반도체, 테스팅, ASIC 설계, CAD/VLSI>