고집적 회로에 대한 고속 경로지연 고장 시뮬레이터

임 용 태† · 강 용 석† · 강 성 호††

요 약

스캔 환경에 바탕을 둔 대부분의 경로 지연고장 시뮬레이터들은 개선된 스캔 플립플롭을 사용하며 일반적인 논리 게이트를 대상으로만 동작한다. 본 연구에서는 새로운 논리값을 사용한 새로운 경로 지연고장 시뮬레이션 알고리듬을 고안하여 이의 적용범위를 CMOS 소자를 포함하는 대규모 집적회로로 확장하였다. 제안된 알고리듬에 기초하여 표준 스캔 환경 하에서 동작하는 고속 지연고장 시뮬레이터를 개발하였다. 실험결과는 새 시뮬레이터가 효율적이며 정확함을 보여준다.

A High Speed Path Delay Fault Simulator for VLSI

Yong Tae Yim† · Yong Seok Kang† · Sung Ho Kang††

ABSTRACT

Most of the available path delay fault simulators for scan environments rely on the use of enhanced scan flip-flops and exclusively consider circuits composed of only discrete gates. In this research, a new path delay fault simulation algorithm using new logic values is devised to enlarge the scope to the VLSI circuits which consist of CMOS elements. Based on the proposed algorithm, a high speed path delay fault simulator for standard scan environments is developed. The experimental results show that the new simulator is efficient and accurate.

1. 서 론

양질의 고집적회로에 대한 요구가 점차로 증대해짐과 동시에 최악의 시간 조건에 대한 분석보다는 통계적인 시간 조건에 대한 분석이 고속회로 설계의 바탕이 됨에 따라 지연고장 테스트가 매우 중요한 과제로 등장하고 있다. 지연고장 테스트를 통해 검사하고자 하는 회로가 동작 주파수의 범위에서 작동하는지를 검사할 수 있다. 따라서 지연고장 테스트는 적어도 두 벡터를 필요로 한다. 일반적으로 첫 번째 벡터를 초기값, 두 번째 벡터를 최종값이라고 한다.

지연고장을 검사하기 위해 지연고장 모델이 사용되어 왔다. 지연고장 모델에는 게이트 지연고장 모델(gate delay fault model)[1, 2]과 경로 지연고장 모델(path delay fault model)[3-5]의 두 종류가 있다. 게이트 지연고장 모델은 회로의 전체 지연이 특정 게이트들에 집중되어 있다는 가정을 바탕으로 한다. 따라서 회로의 지연을 고착고장에서와 같이 정량화할 수 있으며 빠른 시간 내에 검사할 수 있다. 그러나 최근의 지연고장 테스트에서 나타나듯 회로 내의 모든 게이트들이 명세(specification)에서 규정한 시간 이내에서 동작하지만 회로의 최종 출력에서는 지연고장이 발생되는 경우가 있다. 이러한 지연은 회로 내의 특정

※이 논문은 1995년도 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음.

† 준 회 원:연세대학교 전기공학과
†† 정 회 원:연세대학교 전기공학과

논문접수:1996년 7월 31일, 심사완료:1996년 11월 18일

경로 상에 게이트들의 작은 지연들이 경로를 따라 누적되어 나타난 결과로서 기존의 게이트 지연고장 모델로는 다룰 수가 없다. 이를 해결하기 위해 경로 지연고장 모델이 제시되었다. 경로 지연고장 모델은 게이트 지연고장 모델과는 달리 회로 내의 경로를 대상으로 지연고장을 검사하게 된다. 그 결과 특정 게이트들에 의한 지연고장 뿐만 아니라 경로 상에 누적된 지연고장도 검사할 수 있다. 하지만 일반적으로 경로의 수는 게이트의 수보다 많으므로 경우에 따라서는 무수히 많은 경로를 검사해야 한다는 결점이 있다. 따라서 일반적으로 임계경로를 추출하여 검사하게 된다. 경로 지연고장 모델이 게이트 지연고장 모델보다 검사에 더 많은 시간을 필요로 하지만 집중된 지연고장 뿐만 아니라 누적된 지연고장도 다룰 수 있으므로 본 논문에서도 경로 지연고장 모델을 이용하였다.

일반적으로 회로의 설계자가 제공한 벡터가 실제로 그 경로를 테스트할 수 있는지를 조사할 필요가 있다. 또 어떤 임계경로나 회로의 기능을 확인할 필요가 있다. 그리고 생성된 입력 벡터를 ROB(robust:경성)[6-8] 테스트와 NR(non robust:연성)[6-8] 테스트로 분류할 필요가 있다. 이러한 목적을 위해 경로 지연고장 시뮬레이터의 개발이 필요하다. 특히 순차회로의 고착고장을 검사하기 위해 사용하는 스캔환경을 지연고장에 적용할 수 있다면 매우 효율적인 시뮬레이션이 가능할 것이다. 따라서 표준 스캔환경(standard scan environment) 하에서 동작하는 지연고장 시뮬레이터는 지연고장 검사 과정에 통합되는 경향이다. 특히 최근의 집적 회로 설계에서는 전력과 칩 면적의 최소화를 위해 VLSI 칩의 주요부분에 CMOS 기술을 바탕으로 한 주문형 설계(custom design)가 사용된다는 점을 생각할 때, 이러한 시뮬레이터가 스위치 레벨로 구성된 CMOS 회로에서 동작할 수 있다면 보다 광범위한 회로들에 대해서 정확한 시뮬레이션이 가능할 것이다. 그러나 현재까지 경로 지연고장 테스트 생성기에 대한 연구는 많이 이루어져 있지만 지연고장 시뮬레이션에 대한 연구는 별로 이루어져 있지 않다. 경로 지연고장 시뮬레이션을 다루고 있는 [7, 8]에서는 NR 테스트에 대한 경로 지연고장 시뮬레이션을 보다 빨리 수행하기 위해 [3]에서 사용된 6개의 논리값을 4개의 논리값으로 줄이는 시도를 하였다. 그러나 이들은 모두 게이트 레벨의 조합회로에 대한 경로 지연고장 시뮬레이션만을 다루었다. 이들이 사용한 논리값만으로는 스위치 레벨에 대한 정확한 지연고장 시뮬레이션을 수행할 수 없다. 또한 조합회로만을 다루므로 표준 스캔환경 하에서도 동작할 수 없다. 따라서 본 논문에서는 새로운 논리값을 도입하여 지연고장 시뮬레이션의 범위를 CMOS 논리가 사용된 스위치 레벨의 회로로 넓히는 시도와 함께 표준 스캔환경에서 동작하는 경로 지연고장 시뮬레이터를 제안한다.

2. 테스트 유형과 입력 제한 조건

일반적으로 여러 유형의 경로 지연고장 테스트가 있다. 경로입력의 신호가 출력에 동적 해저드를 유발하지 않고 회로의 나머지 부분의 지연에 관계없이 경로상의 지연을 검출할 수 있는 두 패턴의 테스트를 HFR (hazard free robust:무해저드 경성) 테스트라고 한다[9]. 회로의 나머지 부분에서의 지연에 관계없이 경로상의 지연을 검출할 수 있는 두 패턴의 테스트를 ROB 테스트라고 한다[6-8]. ROB 테스트의 조건을 만족하지 못할 때 이를 NR 테스트라고 하는데 이는 다시 SNR(strong non robust:강연성) 테스트[10]와 WNR (weak non robust:약연성) 테스트[10]의 두 종류로 나눌 수 있다. 경로 상의 천이(transition)가 경로 상에 있는 각각의 소자들에 도달하기 전에 모든 경로외 입력들이 최종값으로 안정된다면 경로 지연고장을 검출할 수 있는 두 패턴의 테스트를 WNR 테스트라고 한다. SNR 테스트는 경로외 입력에 정적 해저드가 없다면 ROB 테스트가 되고, 그렇지 않다면 WNR 테스트가 된다.

네 종류의 테스트에 대한 정의에 따르면 이 네 종류의 테스트는 계층적인 구조를 가짐을 알 수 있다. 즉, 한 경로에 대해 HFR 테스트가 존재한다면 이 경로에 대해 ROB 테스트는 항상 존재한다. 마찬가지로 한 경로에 대해 ROB 테스트가 존재한다면 이 경로에 대해 SNR 테스트는 항상 존재한다. SNR 테스트와 WNR 테스트에 대해서도 같은 관계가 성립한다. 이러한 관계는 경로 지연고장 검사에 우선 순위를 부여한다. 즉, 경로 지연고장을 검사할 때는 가장 먼저 HFR 테스트에 대한 검사를 수행하게 된다. 만약 HFR 테스트가 존재하지 않는다면 ROB 테스트에 대

한 검사를 수행한다. 보다 하위의 테스트에 대해서도 같은 방식의 순위가 적용된다.

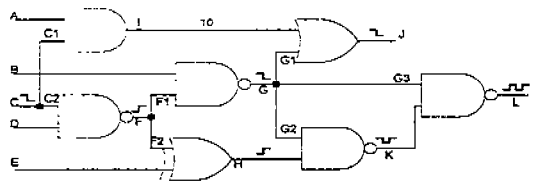
AND(NAND), OR(NOR), XOR(XNOR) 게이트에 적용되는 각각의 테스트에 대한 경로외 입력의 제한조건을 〈표 1〉에, 스위치 레벨 소자인 NMOS와 PMOS에 적용되는 각각의 테스트에 대한 경로외 입력의 제한조건을 〈표 2〉에 나타내었다.

〈표 1〉 경로외 입력의 제한조건(게이트 레벨 소자)
〈Table 1〉 Off-path input constraints(gate level elements)

소자 유형	경로입력천이	경로외 입력 조건			
		HFR	ROB	SNR	WNR
AND (NAND)	상승천이	S1	X1	X1	X1
	하강천이	S1	S1	11	X1
OR (NOR)	상승천이	S0	S0	00	X0
	하강천이	S0	X0	X0	X0
XOR (XNOR)	상승천이-상승천이	S0	S0	00	X0
	상승천이-하강천이	S1	S1	11	X1
	하강천이-상승천이	S1	S1	11	X1
	하강천이-하강천이	S0	S0	00	X0

〈표 2〉 경로외 입력의 제한조건(스위치 레벨 소자)
〈Table 2〉 Off-path input constraints(switch level elements)

소자 유형	경로입력	천이유형	경로외 입력 조건			
			HFR	ROB	SNR	WNR
NMOS	게이트	상승-상승	S1	X1	X1	X1
		상승-하강	S0	X0	X0	X0
		하강-상승	S0	S0	00	X0
		하강-하강	S1	S1	11	X1
	데이터 (드레인)	모든 천이	S1	S1	11	X1
PMOS	게이트	상승-상승	S0	S0	00	X0
		상승-하강	S1	S1	11	X1
		하강-상승	S1	X1	X1	X1
		하강-하강	S0	X0	X0	X0
	데이터 (드레인)	모든 천이	S0	S0	00	X0



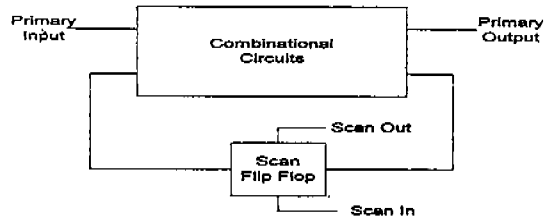
(그림 1) ROB 테스트와 NR 테스트
(Fig. 1) ROB tests and NR tests

〈표 1〉과 〈표 2〉에서 S1은 두 시간 프레임 동안 해저드 없는 논리값 1을, S0는 두 시간 프레임 동안 해저드 없는 0을 유지함을 의미한다. 11은 두 시간 프레임 동안 1의 값을 갖지만 해저드가 있을 수도 있음을 의미한다. 마찬가지로 00은 두 시간 프레임 동안 0의 값을 갖지만 역시 해저드가 있을 수도 있음을 의미한다. X1과 X0는 최종값이 각각 1과 0이지만 초기값은 X임을 의미한다. 예를 들어, (그림 1)의 회로에서 입력 패턴이 (A, B, C, D, E)=(10, 11, 10, 11, 00)로 주어진다면 C→C2→F→F1→G→G3→L로 이루어지는 경로에 대해 주어진 입력 패턴이 이 경로에 대해 ROB 테스트가 될 수 있는지를 살펴보자. 만약 H를 통과하는 경로에 지연이 있다면 K에서는 정적 해저드가 발생하게 된다. 이러한 해저드는 출력 L에서 해저드를 유발시키고 결과적으로 경로입력의 지연을 관찰할 수 없게 만든다. 왜냐하면, 경로입력 G3의 하강천이가 도달하기도 전에 L에서 최종값이 출력되므로 경로입력에 지연이 있는데도 불구하고 없는 것으로 판단하게 하는 오류를 유발시킬 수도 있기 때문이다. 따라서 ROB 테스트가 되기 위해서는 H가 S1을 가져야만 한다. 하지만 경로입력 G3이 상승천이를 가진다면 경로외 입력 H에 지연이 있더라도 K에서는 해저드가 발생하지 않는다. 반면에, C→C2→F→F1→G→G1→J로 이루어지는 경로를 고려해 보면 경로입력이 하강천이를 가지므로 경로외 입력 I에 지연이 있더라도 출력 J에서는 해저드가 발생하지 않는다. 따라서 경로외 입력의 지연에 관계없이 경로입력의 지연을 검사할 수 있다. 하지만, 경로입력 G1이 상승천이를 가지고 경로외 입력 I에 지연이 있다면 출력 J에서 해저드가 발생하므로 경로입력의 지연을 정확하게 검사할 수 없다. 유사한 방법으로 다른 게이트 유형에 대해서도 〈표 1〉과 같은 결과를 유도해 낼

수 있다.

게이트 레벨의 회로에서와는 달리 스위치 레벨의 회로에서는 트랜지스터의 특성 상 각각의 트랜지스터에 대한 입력의 천이가 정확한 상승천이나 하강천이의 형태로 전달되지 않는다. 예를 들면, PMOS의 드레인 입력이 전원(VDD)이고 게이트 입력이 상승천이일 때 출력은 첫 번째 시간 프레임에서 1의 값을 가지고 두 번째 시간 프레임에서는 Z의 값을 가진다. 여기서 Z는 하이 임피던스를 의미한다. 따라서 두 시간 프레임 동안 1Z의 값을 가진다. NMOS 트랜지스터에 대해서도 마찬가지로 드레인 입력이 접지(GROUND)이고 게이트의 입력이 상승천이일 때 두 시간프레임 동안 출력은 Z0이 된다. 그러나 이러한 출력 신호들은 모두 wire에서 정확한 천이의 파형을 형성하게 된다. 이러한 이유로 인해 스위치 레벨에서는 ROB 테스트에 대한 명확한 한계를 정하기가 어려운 실정이다. 본 논문에서는 스위치 레벨에 대해 〈표 2〉와 같은 경로외 입력의 제한조건을 사용하였다. 예를 들어 NMOS의 게이트 입력이 경로입력이고 하강천이일 경우, 출력에 해저드가 없기 위해서는 데이터(드레인)입력에 해저드가 없어야 한다. 그 이유는 본 논문에서는 Z를 일종의 천이로 고려했기 때문이다. 따라서 ROB 테스트가 되기 위해서는 데이터 입력에 해저드가 없어야 한다. 다른 천이유형과 다른 테스트 유형에 대해서도 비슷한 관계를 적용시킬 수 있다. 스위치 레벨에서는 일반적으로 사용되는 네 가지 논리값(0, 1, X, Z)만으로는 실제로 ROB 테스트가 존재하는 경로에 대해서 검출이 불가능할 수도 있다. 따라서 본 논문에서는 다음절에서 이러한 문제를 해결하기 위해서 새로운 논리값을 도입하였다.

스캔환경 하에서 고장 시뮬레이터를 개발할 때는 최종값을 다룰 수 있는 효율적인 방법을 고려해야 한다. 한 가지 방법은 두 시간 프레임 동안 제어가 가능한 개선된 스캔 플립플롭[11]을 사용하는 것이고 또 다른 방법은 표준 스캔환경 하에서 스캔 이동(scan shifting)[12-15]을 이용하거나 기능적 지정(functional justification)[16]을 이용하는 것이다. 첫 번째 방법은 칩 면적과 테스트 시간의 증가를 초래하기 때문에 바람직한 방법이 될 수 없다. 두 번째 방법 역시 이동되는 입력의 값에 따라 테스트 가능한 경로에 대해서도 테스트 불가능으로 판정될 수 있는 위험이 있기 때문에 바람직한 방법이 될 수 없다. 따라서 본 논문에서는 위의 두 가지 방법의 사용은 제외되었다. 본 논문에서 채택된 방법은 표준 스캔환경 하에서 기능적 지정을 이용하는 것으로 초기값은 플립플롭을 통해 제공되고 최종값은 회로의 동작에 의해 피드백되는 값으로 제공된다. 이를 (그림 2)에 나타내었다. 테스트 모드일 때 스캔플립플롭의 스캔입력(그림의 scan in)을 통해 가해진 입력이 초기값으로 사용된다. 최종값은 조합회로를 통해 피드백되는 값이 사용된다.



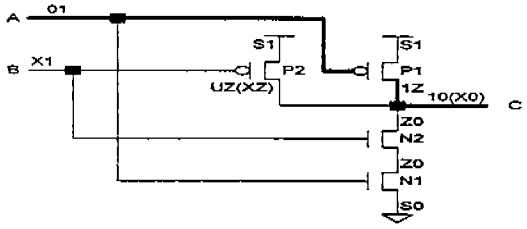
(그림 2) 표준 스캔에 대한 하드웨어 모델
(Fig. 2) Hardware model for standard scan

3. 논리값

앞 절에서도 언급했듯이 일반적으로 지연고장의 테스트에 사용되는 논리값은 0, 1, X, Z의 네 가지 논리값이다. 여기서 Z는 하이 임피던스를 나타낸다. 하지만 이 네 가지 논리값만으로 지연고장에 대한 테스트를 수행한다면 트랜지스터의 게이트 입력이 X일 경우 출력도 X가 되어 실제로 테스트가 존재함에도 불구하고 찾아내지 못 할 수도 있다. 따라서 이러한 경우를 다룰 수 있는 새로운 논리값이 필요하게 된다. 본 논문에서는 이를 위해 두 종류의 새로운 논리값 U와 D를 제시하였다. 만약 NMOS 트랜지스터의 게이트 입력이 X이고 드레인 입력이 0이라면 출력은 Z 또는 0이 된다. 이를 논리값 D로 나타내었다. 또 NMOS 트랜지스터의 게이트 입력이 X이고 드레인 입력이 1이면 출력은 Z 또는 1이 된다. 이를 논리값 U로 나타내었다. 논리값 U와 D의 wire에서의 연산을 〈표 3〉에, 새로운 논리값의 사용 예를 (그림 3)의 회로를 통해 나타내었다. 〈표 3〉에서 2 입력 wire의 입력이 각각 U와 0일 경우 wire의 출력은 X가 된다. 나머지 입력들에 대해서도 표를 통해 알 수 있다.

〈표 3〉 2 입력 wire에서의 U와 D의 계산
〈Table 3〉 Evaluation of U and D for 2 input wire

논리값	0	1	X	Z	U	D
0	0	X	X	0	X	0
1	X	1	X	1	1	X
X	X	X	X	X	X	X
Z	0	1	X	Z	U	D
U	X	1	X	U	U	X
D	0	X	X	D	X	D



(그림 3) 새로운 논리값의 예
(Fig. 3) Example of new logic values

그림에서 굵은 선으로 표시된 부분이 테스트 경로에 해당된다. 그림에서 만약 네 개의 논리값만 사용된다면 첫 번째 시간 프레임에서 P2의 출력이 X가 되어 C에서의 출력이 X가 될 것이다(괄호 안의 논리값 참조). 따라서 그림에서 제공된 테스트는 실제로 ROB 테스트임에도 불구하고 스위치 회로에 대해서는 ROB 테스트가 될 수 없다. 하지만 본 논문에서 제시하는 새로운 논리값을 적용하면, P2에서의 출력이 U가 되어 C에서는 논리값 1을 갖게 된다. 따라서 스위치 회로에 대해서도 역시 ROB 테스트가 됨을 알 수 있다. 예에서 알 수 있듯이 새로운 논리값을 사용하면 스위치 회로에 대해서도 보다 정확히 지연고장을 테스트할 수 있다.

본 논문에서 제안된 지연고장 시뮬레이터는 병렬패턴 고장 시뮬레이션 알고리듬(parallel pattern fault simulation algorithm)[7, 8]을 사용하기 때문에 논리값을 인코딩하여야 한다. 본 논문에서 사용된 논리값의 인코딩을 〈표 4〉에 나타내었다. 여기서 $zv_i[j]$는 UD를 구별하기 위해, $bv_i[j]$는 부울값을 구별하기 위해, 그리고 $xv_i[j]$는 XZ값을 구별하기 위해 사용되었다. 특히 표에 나타낸 인코딩은 벌렬연산의 처리과정에서 발생하는 연산의 수를 최소화하도록 고안되었다. 표에서 i는 입력, j는 시간 프레임을 나타낸다.

〈표 4〉 논리값 인코딩(6 논리값)
〈Table 4〉 Logic value encoding(6 logic values)

논리값	$zv_i[j]$	$bv_i[j]$	$xv_i[j]$
0	1	0	0
1	0	1	0
X	1	1	1
Z	0	0	1
D	1	0	1
U	0	1	1

일반적인 논리 게이트의 연산에는 U와 D가 사용되지 않으므로 네 개의 논리값만 사용하면 된다. 이에 대한 인코딩을 〈표 5〉에 나타내었다. 〈표 5〉에 사용된 비트 구분자는 〈표 4〉에서와 비슷한 의미를 가진다. 즉, $bv_i[j]$는 부울값의 구분을 위해, $xv_i[j]$는 XZ값의 구분을 위해 사용되었다. i는 입력, j는 시간 프레임을 나타낸다. 하지만 알고리듬의 복잡도를 줄이기 위해 모든 계산 결과는 여섯 개의 논리값으로 인코딩되어 메모리에 저장된다.

〈표 5〉 논리값 인코딩(4 논리값)
〈Table 5〉 Logic value encoding(4 logic values)

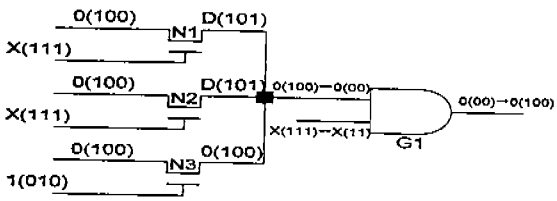
논리값	$bv_i[j]$	$xv_i[j]$
0	0	0
1	1	0
X	1	1
Z	0	1

〈표 5〉에 사용된 인코딩 역시 병렬연산의 처리과정에서 발생할 수 있는 연산의 수를 최소화하도록 고안되었다. 게이트 레벨과 스위치 레벨이 혼합된 회로를 테스트할 경우에는 네 개의 논리값과 여섯 개의 논리값 사이의 인코딩 변환이 필요하다. 〈표 4〉와 〈표 5〉를 비교해 보면, 여섯 개의 논리값에서 네 개의 논리값으로의 인코딩 변환은 매우 간단함을 알 수 있다.

단순히 $zv_i[j]$를 제거해 버리면 된다. 네 개의 논리값을 여섯 개의 논리값 인코딩으로 변환하는 것은 (그림 4)의 루틴을 통해 얻을 수 있다. (그림 4)에서 out은 출력, t는 시간 프레임을 나타내고, C 언어와 유사하게 ~은 bitwise not, ^은 bitwise xor를 나타낸다.

$$zv_{out}[t] = \sim(bv_{out}[t] \hat{} xv_{out}[t])$$

(그림 4) 논리값의 변환
(Fig. 4) Logic value conversion



(그림 5) 논리값의 인코딩 변환의 예
(Fig. 5) Example of encoding conversion of logic value

논리값의 인코딩 변환 예를 (그림 5)의 회로를 통해 살펴보자. (그림 5)에서 괄호 안의 값은 각 논리값에 대하여 인코딩된 비트값을 나타낸다. 예를 들어 N1의 입력은 각각 0과 X이므로 메모리에 저장된 인코딩 비트값은 각각 100과 111이다(괄호 안의 인코딩 비트값 참조). 이 입력으로 연산을 수행하면 결과는 D이므로 메모리에 저장되는 인코딩 비트값은 101이 된다. N1, N2, N3와 wire에 대한 연산이 끝나고 나면 AND 게이트에 대한 연산을 수행한다. 이 때 AND 게이트의 입력은 각각 0과 X이므로 메모리에 저장된 인코딩 비트값은 각각 100과 111이다. 그러나 게이트 소자인 AND 게이트에 대한 연산을 수행하기 위해서는 네 개의 논리값만으로도 충분하므로 이 값들은 각각 최상위 비트를 제외한 00과 11이 된다(괄호 안의 인코딩 비트값 참조). AND 게이트에 대한 연산이 끝나면 결과는 0이므로 인코딩 비트값은 00이 된다. 이 값을 메모리에 저장하기 위해서는 3비트의 인코딩 비트값으로 변환시켜야 하므로 (그림 4)에서 제시된 루틴을 통해 $zv_{out}[t]$를 계산해야 한다. 계산 결과 $zv_{out}[t]$ =1이므로 인코딩 비트값 100를 얻을 수 있고 이 값이 다음 소자의 연산을 위해 메모리에 저장된다. 다른 논리값들에 대해서도 같은 과정이 적용될 수 있다. 물론 스위칭 소자들에 대해서는 이러한 과정이 필요없다.

논리값을 표현할 때 가장 어려운 점이 정적 해저드를 갖지 않는 안정된 값을 다루는 경우이다. 본 논문에서는 이를 해결하기 위해 신호의 안정도를 나타내는 값으로 sv를 사용하였다. sv가 1일 때 그 신호에는 두 시간 프레임 동안 해저드가 없음을 나타내고, sv가 0일 때는 그 신호에 정적 해저드가 존재함을 의미한다. 예를 들어 신호의 초기값과 최종값이 모두 0이고 sv가 1이라면, 그 신호는 S0임을 의미한다. 만약 sv가 0이라면 그 신호는 00임을 의미한다. 1과 Z도 같은 방식으로 설명된다. 안정도값이 나타내는 의미를 〈표 6〉에 나타내었다.

〈표 6〉 안정도값
〈Table 6〉 Stable value

시간프레임 1		시간프레임 2		sv	의미하는 논리값
인코딩	논리값	인코딩	논리값		
100	0	100	0	1	S0
100	0	100	0	0	00
010	1	010	1	1	S1
010	1	010	1	0	11
001	Z	001	Z	1	SZ
001	Z	001	Z	0	ZZ

4. 알고리듬

본 논문에서 제안한 시뮬레이터에 사용된 지연고장 시뮬레이션의 알고리듬을 (그림 6)에 나타내었다. 그림에서 알 수 있듯이 먼저 회로 내의 모든 소자들에 대해 병렬연산을 수행한다. 이 때 연산을 행할 소자가 MOS와 같은 스위칭 소자라면 여섯 개의 논리값으로, 일반적인 게이트 소자라면 네 개의 논리값으로 연산을 수행한다. 각 소자에 대한 병렬연산이 끝

```
path_delay_fault_simulation()
{
    for (two time frames)
    {
        read_input_vectors(32 input vectors)
        for (all elements)
        {
            if (element is gate level element)
                parallel_evaluate(element, 4 values)
            else
                parallel_evaluate(element, 6 values)
        }
    }
    for (all paths)
    {
        if (HFR detection is not performed)
        {
            fault_detection(HFR)
            if (HFR detection failed and
               ROB detection is not performed)
            {
                fault_detection(ROB)
                if (ROB detection failed and
                   SNR detection is not performed)
                {
                    fault_detection(SNR)
                    if (SNR detection failed and
                       WNR detection is not performed)
                        fault_detection(WNR)
                }
            }
        }
    }
}
```

(그림 6) 경로 지연고장 시뮬레이션 알고리듬
(Fig. 6) Path delay fault simulation algorithm

나면, 연산의 결과를 이용하여 모든 경로에 대해 고장 검출이 수행된다. 고장 검출은 가장 상위레벨의 테스트 유형에 대해 먼저 수행되고 이것이 실패할 경우에만 보다 하위 레벨의 테스트 유형에 대해 수행된다. 즉 HFR 테스트에 대한 고장검출의 과정이 가장 먼저 수행되고 검출에 실패할 경우에만 ROB 테스트에 대한 고장 검출이 수행된다. 보다 하위레벨의 테스트 유형에 대해서도 같은 과정이 적용된다. 〈표 7〉은 각 논리값의 입력에 대한 NMOS의 출력을 나타낸다. 표에서 괄호 안의 수는 각 논리값에 대해 인코딩된 비트값을 나타낸다. 이를 바탕으로 각 비트에 대한 Karnaugh 도표를 작성할 수 있다. 〈표 8〉은 *zv* 비트값에 대한 Karnaugh 도표를 나타낸다. 각 인코딩 비트에 대한 Karnaugh 도표를 이용하여 각 비트의 연산식을 구할 수 있다. 구해진 모든 인코딩 비트에 대한 연산식을 (그림 7)에 나타내었다. (그림 8)은 2입력 AND 게이트에 대한 연산식을 나타낸다. 다른 소자들에 대해서도 유사한 방식으로 유도할 수 있다. 사용된 모든 연산식에서 *out*은 출력을, $t(=0, 1)$는 시간 프레임을, *gate*는 트랜지스터의 게이트 입력을, *data*는 트랜지스터의 드레인 입력을 나타낸다. 또한 *on-path*는 경로 입력을, *off-path*는 경로외 입력을 나타낸다. 사용된 연산기호는 C 언어와 유사하게 ~은 bitwise not을, ^는 bitwise xor를, &는 bitwise and를, |는 bitwise or를 나타낸다. 0과 1은 각각 첫 번째 시간 프레임과 두 번째 시간 프레임을 나타낸다.

병렬 연산은 두 시간 프레임 동안 진행되는데 첫 번째 시간 프레임의 연산이 끝나면 회로의 입력이 갱신되고 플립플롭의 입력이 출력으로 전달된다. 특히 두 번째 시간 프레임의 연산을 수행하기에 앞서 신호의 안정도를 구하기 위해 안정도값이 계산된다. 예를 들어 NMOS의 경우, 출력에 해저드가 없기 위해서는

〈표 7〉 NMOS의 특성
〈Table 7〉 Property of NMOS

data(drain) \ *gate*	0(100)	1(010)	X(111)
0(100)	Z(001)	0(100)	D(101)
1(010)	Z(001)	1(010)	U(011)
X(111)	Z(001)	X(111)	X(111)
Z(001)	Z(001)	Z(001)	Z(001)
D(101)	Z(001)	D(101)	D(101)
U(011)	Z(001)	U(011)	U(011)

〈표 8〉 $zv_{out}[t]$ 값에 대한 Kamaugh 도표
〈Table 8〉 Kamaugh map of $zv_{out}[t]$ value

zv_{data}[t] \ zv_{gate}[t]	1	0	1
1	0	1	1
0	0	0	0
1	0	1	1
0	0	0	0
1	0	1	1
0	0	0	0

게이트 입력이 S0이거나, 게이트 입력이 S1이고 데이터(드레인)입력에 해저드가 없어야 한다. 게이트 입력이 S0인 경우는 입력의 초기값과 최종값이 모두 0(100)이고 *sv*가 1인 경우이므로 $zv_{gate}[0]\&\sim bv_{gate}[0]\&\sim xv_{gate}[0]\&\sim zv_{gate}[1]\&\sim bv_{gate}[1]\&\sim xv_{gate}[1]\&\sim sv_{gate}$와 같은 식으로 나타낼 수 있다. 두 번째 경우에도 같은 방법으로 구하면 (그림 9)와 같은 안정도값 연산식이 구해진다. 다른 소자들에 대해서도 유사한 방법으로 구할 수 있다. 두 시간 프레임 동안의 연산이 모두 끝나면 그 결과를 고장 검출의 과정에서 이용하기 위해 메모리에 저장한다.

병렬연산이 수행된 후에는 모든 유형의 고장에 대해 고장 검출의 과정이 수행된다. 사용된 고장 검출의 알고리듬을 (그림 10)에 나타내었다. 고장 검출의 과정에서는 각 유형의 테스트에 대해 경로의 시작점을 제외하고는 경로 상의 입력에 대해서는 특정 천이가 일어나는지를 조사할 필요가 없다. 왜냐하면, 경로의 입력의 값이 경로 입력에서 특정 천이가 일어나도록 설정되기 때문이다. 따라서 고장 검출 과정에서 수행되는 루틴은 각 테스트의 유형에 대해 원하는 천이가 일어나도록 경로의 입력이 설정되었는지 만을 조사하면 된다.

$$
\begin{aligned}
temp &= zv_{gate}[t] \ \hat{}\ \sim bv_{gate}[t] \ \hat{}\ \sim xv_{gate}[t] \\
zv_{out}[t] &= zv_{data}[t] \ \&\ \sim temp \\
bv_{out}[t] &= bv_{data}[t] \ \&\ \sim temp \\
xv_{out}[t] &= xv_{data}[t] \mid temp \mid zv_{gate}[t] \mid \sim bv_{gate}[t] \mid xv_{gate}[t]
\end{aligned}
$$

(그림 7) 병렬 연산(NMOS)
(Fig. 7) Parallel evaluation(NMOS)

$$
\begin{aligned}
xv_{out}[t] = \ & (bv_{off-path}[t] \ \&\ xv_{on-path}[t]) \mid \\
& (bv_{on-path}[t] \ \&\ xv_{off-path}[t]) \mid \\
& (xv_{on-path}[t] \ \&\ xv_{off-path}[t]) \\
bv_{out}[t] &= xv_{out}[t] \mid (bv_{on-path}[t] \ \&\ bv_{off-path}[t])
\end{aligned}
$$

(그림 8) 병렬 연산(2입력 AND 게이트)
(Fig. 8) Parallel evaluation(2 input AND gate)

$$
\begin{aligned}
sv_{out} = \ & (zv_{gate}[0] \ \&\ \sim bv_{gate}[0] \ \&\ \sim xv_{gate}[0] \ \& \\
& zv_{gate}[1] \ \&\ \sim bv_{gate}[1] \ \&\ xv_{gate}[1] \ \&\ sv_{gate}) \mid \\
& (\sim zv_{gate}[0] \ \&\ bv_{gate}[0] \ \&\ \sim xv_{gate}[0] \ \& \\
& \sim zv_{gate}[1] \ \&\ bv_{gate}[1] \ \&\ \sim xv_{gate}[1] \ \&\ sv_{gate} \ \&\ sv_{data})
\end{aligned}
$$

(그림 9) 안정도값 계산(NMOS)
(Fig. 9) Stable value evaluation(NMOS)

본 논문에서 사용된 경로외 입력을 조사하는 루틴을 NMOS(PMOS)와 AND(OR) 게이트에 대해 각각 〈표 9〉와 〈표 10〉에 나타내었다. 예를 들어 AND 게이트일 경우 경로입력이 상승천이라면 ROB 테스트에 대한 경로외 입력의 조건은 X1이므로(〈표 1〉 참

```
fault_detection()
{
    if (the head of the path has desired transition)
        for (all elements on the path except head of the path)
        {
            if (each element has desired off path condition
                according to test types)
                return TESTED
            else
                return UNTESTED
        }
}
```

(그림 10) 고장 검출 알고리듬
(Fig. 10) Fault detection algorithm

〈표 9〉 NMOS와 PMOS에 대한 경로외 입력의 제한조건(ROB 테스트)
〈Table 9〉 Off-path input constraints for NMOS and PMOS (ROB test)

유형	경로 입력	천이	경로외 입력
NMOS	게이트	상승-상승	$\sim zv_{data}[1]$ & $bv_{data}[1]$ & $\sim xv_{data}[1]$
		하강-하강	$\sim zv_{data}[0]$ & $bv_{data}[0]$ & $\sim xv_{data}[0]$ & sv_{data}
		상승-하강	$zv_{data}[1]$ & $\sim bv_{data}[1]$ & $\sim xv_{data}[1]$
		하강-상승	$zv_{data}[0]$ & $\sim bv_{data}[0]$ & $\sim xv_{data}[0]$ & sv_{data}
	데이터 (드레인)	모든 천이에 대해	$\sim zv_{gate}[0]$ & $bv_{gate}[0]$ & $\sim xv_{gate}[0]$ & $\sim zv_{gate}[1]$ & $bv_{gate}[1]$ & $\sim xv_{gate}[1]$ & sv_{gate}
PMOS	게이트	상승-상승	$zv_{data}[0]$ & $\sim bv_{data}[0]$ & $\sim xv_{data}[0]$ & sv_{data}
		하강-하강	$zv_{data}[1]$ & $\sim bv_{data}[1]$ & $\sim xv_{data}[1]$
		상승-하강	$\sim zv_{data}[0]$ & $bv_{data}[0]$ & $\sim xv_{data}[0]$ & sv_{data}
		하강-상승	$\sim zv_{data}[1]$ & $bv_{data}[1]$ & $\sim xv_{data}[1]$
	데이터 (드레인)	모든 천이에 대해	$zv_{gate}[0]$ & $\sim bv_{gate}[0]$ & $\sim xv_{gate}[0]$ & $zv_{gate}[1]$ & $\sim bv_{gate}[1]$ & $\sim xv_{gate}[1]$ & sv_{gate}

〈표 10〉 AND와 OR 게이트에 대한 경로외 입력의 제한조건 (ROB 테스트)
〈Table 10〉 Off-path input constraints for AND and OR gates(ROB test)

유형	천이	경로외 입력
AND	상승-상승	$bv_{off-path}[1]$ & $\sim xv_{off-path}[1]$
	하강-하강	$bv_{off-path}[0]$ & $\sim xv_{off-path}[0]$ & $bv_{off-path}[1]$ & $\sim xv_{off-path}[1]$ & $sv_{off-path}$
OR	상승-상승	$\sim bv_{off-path}[0]$ & $\sim xv_{off-path}[0]$ & $\sim bv_{off-path}[1]$ & $\sim xv_{off-path}[1]$ & $sv_{off-path}$
	하강-하강	$\sim bv_{off-path}[1]$ & $\sim xv_{off-path}[1]$

조) 경로외 입력의 최종값이 1이면 된다. 즉, $bv_{off\text{-}path}[1]\&\sim xv_{off\text{-}path}[1]$의 값이 1이 되어야 한다. 경로입력이 하강천이일 때도 같은 방법으로 경로외 입력을 조사할 수 있다. NMOS와 PMOS에 대해서도 〈표 2〉를 참조하면 〈표 9〉와 같은 루틴을 얻을 수 있다. 유사한 방법으로 다른 소자들과 다른 유형의 테스트에 대해서도 경로외 입력을 조사하는 루틴을 구할 수 있다.

각 유형의 테스트들은 서로 계층적인 구조를 가지므로 가장 제한적인 유형에 대해 테스트가 존재한다면 다른 유형의 테스트에 대한 검사를 수행할 필요가 없다. 따라서 고장 검출의 과정에서 가장 먼저 조사되는 테스트의 유형은 HFR 테스트가 되도록 하였다. 만약 HFR 테스트에 검사가 실패한다면 ROB 테스트에 대한 검사를 수행하도록 하였다. 마찬가지로 ROB 테스트에 대한 검사가 실패하면 SNR 테스트에 대한 검사를, 마지막으로 WNR 테스트에 대한 검사를 수행하게 하였다.

5. 결 과

본 논문에서 제안된 지연고장 시뮬레이션의 알고리듬은 약 1,4000 줄 정도의 C로 구현되었다. 시뮬레이션에 사용된 회로는 표준 ISCAS 89 회로[17]를 바탕으로 하였는데 스위치 레벨의 회로 테스트를 위해 일정 비율의 논리게이트들을 CMOS로 구성된 스위치 레벨 회로로 변형하였다. 실험에는 10%, 30%, 50%의 변형비율을 갖는 회로들이 사용되었다. 10%의 회로에 대해서는 1000개의 임의경로를, 30% 회로에 대해서는 4000개의 임의경로를, 50%의 회로에 대해서는 8000개의 임의경로를 선택하여 실험하였다. 각 회

로의 변형비율에 따라 경로의 수에 차이를 둔 것은 스위치 레벨의 회로가 많아질수록 회로 내의 경로수도 많아지므로 보다 정확한 결과의 비교를 위해서는 회로의 변형비율이 높아질수록 선택되어야 하는 경로도 많아져야 하기 때문이다. 입력 벡터들은 10000개를 임의로 생성하여 사용하였다. 실험은 Intel Pentium 120MHz의 CPU, 32 Mbytes의 주 메모리, 110 Mbytes의 수왑공간을 가진 IBM PC에서 수행되었다. 〈표 11〉, 〈표 12〉, 〈표 13〉에 실험 결과를 나타내었다. 각 유형의 테스트에 대해 검출된 경로의 수, 고장검출율(FC:Fault Coverage), 검출이 안된 경로의 수(UP), 전체 고장검출율(TFC), 소비된 시간(CPU)등을 표에 나타내었다. 각 테스트 유형에 따른 고장검출율은 상위 유형의 고장검출율을 제외한 것이다. 예를 들면, SNR 테스트의 고장검출율은 ROB 테스트의 고장검출율을 제외한 것이다. 하지만 ROB 테스트는 모두 SNR 테스트로도 사용될 수 있으므로 이를 SNR 테스트의 고장검출율에 포함시켜서 계산할 수도 있다. HFR 테스트를 통해 회로의 특정 경로의 지연을 분석할 수 있다. 하지만 본 실험에서는 지연의 분석에는 중점을 두지 않았기 때문에 HFR 테스트에 대한 결과는 생략하였다. 각 실험결과들을 비교해 보면, 일반적으로 회로 내에 스위치 레벨의 소자가 많아질수록 검출된 ROB 테스트의 고장검출율이 낮아짐을 알 수 있다. 그 이유는 트랜지스터에 대한 ROB 테스트의 제한조건이 그 특성 상 일반적인 게이트에 비해 더 엄격해 질 수밖에 없기 때문이다. 즉 트랜지스터의 드레인이 경로 입력이 되면, ROB 테스트가 되기 위해서는 경로외 입력에 해저드가 없어야 한다(〈표 2〉 참조). 예를 들어, (그림 3)의 CMOS 회로에서 주어진 테스트 벡터로 경로 A→N1→N2→C를 테스트할 경우 게이트 레벨의 NAND 게이트에 대해서는 주어진 테스트가 ROB 테스트가 되지만 (그림 3)과 같은 CMOS 스위치 레벨 회로에서는 주어진 테스트는 WNR 테스트가 된다. B에 인가되는 테스트 벡터가 11일 경우도 마찬가지로 게이트 레벨의 회로에서는 ROB 테스트가 되지만 (그림 3)과 같은 CMOS 스위치 레벨의 회로에서는 SNR 테스트가 된다. 이 예에서 알 수 있듯이 게이트 레벨에서는 주어진 입력 벡터가 ROB 테스트가 되지만 같은 경로의 스위치 레벨 회로에 대해서는 SNR 테스트나 WNR 테스트가 될 경우도 있다. 또한 사용된 입력 벡터를 임의로 선택하였기 때문에 실제로 검출할 수 있는 경로에 대해서도 검출할 수 없는 결과를 낳은 것으로 여겨진다. 하지만 전체 고장검출율은 대체로 일정한 범위 내에서 유지됨을 알 수 있다. 회로에 따라 전체 고장검출율에 오차가 생기는 이유는 경로와 입력 벡터를 임의로 선택하였기 때문이다. 예를 들면 s1238의 경우 50% 회로에서 고장검출율이 많이 감소하였는데 이는 실험에 선택된 경로들이 비효율적인 임의의 벡터 입력으로는 지연고장의 검출이 힘든 경로들을 많이 포함하고 있기 때문인 것으로 생각된다. 하지만 전체 경로에 대해 검사입력 생성기에 의해 생성된 양질의 벡터를 사용한다면 오차는 더 줄어들 것으로 생각된다. 제안된 시뮬레이터는 병렬고장시뮬레이션 알고리듬을 채택하여 큰 회로에 대해서도 실시간 내에 처리할 수 있을 만큼 시뮬레이션 시간을 단축하였고, 각 회로 소자들의 연산수의 최소화를 유도하는 효율적인 인코딩을 통하여 각 소자들에 대한 연산 시간을 대폭 단축하였다. 또한 게이트 레벨과 스위치 레벨 사이의 논리값 인코딩 변환으로 게이트 레벨 회로에서 불필요한 연산의 과정을 수행하지 않도록 하여 전체 회로에 대한 연산 시간을 단축하였다. 현재까지는 표준 스캔환경의 스위치 레벨이 포함된 회로에서 동작하는 시뮬레이터에 대한 연구 보고가 나와있지 않으므로 정확한 비교는 할 수 없지만 기존의 게이트 레벨 실험 결과와 비교해 보면, 스위치 레벨 회로에 의한 복잡도의 증가를 고려할 때 제안된 시뮬레이터가 고속으로 동작함을 알 수 있다. 따라서 CMOS 회로로 구성된 주문형 설계가 많이 사용되는 최근의 회로 설계 경향에 비추어 볼 때 제안된 시뮬레이터의 이용가치는 상당히 높다고 할 수 있을 것이다.

6. 결 론

대부분의 지연고장 시뮬레이터가 알고리듬 상의 어려움으로 인해 개선된 스캔 환경 하에서 게이트 레벨에 대한 검사를 수행하기 때문에 표준 스캔 환경 하에서 CMOS 회로가 사용된 VLSI 회로에서 동작할 수 있는 시뮬레이터를 개발하였다. 제안된 시뮬레이터는 테스트 생성기에서 생성된 테스트 벡터의 고장검출율을 계산하기 위해 사용할 수도 있고, 주어진

〈표 11〉 고장 시뮬레이션의 결과(10%)
〈Table 11〉 Fault simulation results(10%)

회로	ROB	FC(%)	SNR	FC(%)	WNR	FC(%)	UP	TFC(%)	CPU(sec.)
s838	14	1.40	312	31.20	86	8.60	588	41.20	137.91
s953	232	23.20	43	4.30	301	30.10	424	57.60	136.42
s1196	156	15.60	160	16.00	326	32.60	358	64.20	201.57
s1238	142	14.20	137	13.70	371	37.10	350	65.00	175.58
s1423	6	0.60	10	1.00	21	2.10	963	3.70	271.04
s1488	444	44.40	36	3.60	342	34.20	178	82.20	222.23
s1494	352	35.20	90	9.00	391	39.10	207	79.30	210.50
s5378	230	23.00	138	13.80	485	48.50	147	85.30	1066.50

〈표 12〉 고장 시뮬레이션의 결과(30%)
〈Table 12〉 Fault simulation results(30%)

회로	ROB	FC(%)	SNR	FC(%)	WNR	FC(%)	UP	TFC(%)	CPU(sec.)
s838	410	10.25	396	9.90	1032	25.80	2162	45.95	765.17
s953	665	16.38	47	1.18	1171	29.27	2127	46.83	741.87
s1196	499	12.47	502	12.55	1602	40.05	1397	65.08	1097.81
s1238	362	9.05	473	11.82	1015	25.38	2150	46.25	1168.14
s1423	6	0.15	29	0.72	89	2.23	3876	3.10	1520.86
s1488	1655	41.38	63	1.57	1752	43.80	530	86.75	1425.41
s1494	1599	39.98	91	2.27	1588	39.70	722	81.95	1581.75
s5378	126	3.15	168	4.20	3682	92.05	24	99.40	5186.17

〈표 13〉 고장 시뮬레이션의 결과(50%)
〈Table 13〉 Fault simulation results(50%)

회로	ROB	FC(%)	SNR	FC(%)	WNR	FC(%)	UP	TFC(%)	CPU(sec.)
s838	610	7.62	590	7.38	1661	20.76	5139	35.76	1677.76
s953	578	7.22	74	0.93	2418	30.23	4930	38.38	2189.08
s1196	447	5.59	624	7.80	3462	43.27	3467	56.66	2935.01
s1238	31	0.39	12	0.15	206	2.58	7751	3.11	3713.80
s1423	6	0.07	29	0.36	143	1.79	7822	2.22	4043.41
s1488	3489	43.61	131	1.64	2790	34.88	1590	80.82	4089.42
s1494	1968	24.60	204	2.55	3692	46.15	2136	73.30	4345.02
s5378	134	1.68	244	3.05	7598	94.97	24	99.70	13608.90

테스트를 경성 테스트와 연성 테스트로 분류하기 위해 사용할 수 있다. 특히 본 시뮬레이터는 회로 내에 스위치 레벨 회로가 포함되어 있어도 동작할 수 있으므로 주문형 설계가 사용된 회로에서 효과적으로 이용될 수 있을 것으로 여겨진다.

참 고 문 헌

[1] Z. Barzilai and B. Rosen, "Comparison of AC Self-Testing Procedures," Proc. of International Test Conference, pp. 89-94, 1983.

[2] J. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition Fault Simulation," IEEE Design and Test, pp. 32-38, April 1987.

[3] Gordon L. Smith, "Model for Delay Faults Based Upon Paths," Proc. of International Test Conference, pp. 342-349, 1985.

[4] S. Reddy, C. Lin, and S. Patil, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults," Proc. of International Conference on Computer Design, pp. 284-287, 1987.

[5] M. Schulz, K. Fuchs, and F. Fink, "Advanced Automatic Test Pattern Generation Techniques for Path Delay Faults," Proc. of Fault Tolerent Computing Symp., pp. 44-51, 1989.

[6] C. Lin and S. Reddy, "On Delay Fault Testing in Logic Circuits," IEEE Trans. on CAD, pp. 694-703, Sep. 1987.

[7] M. Schulz, F. Fink, and K. Fuchs, "Parallel Pattern Fault Simulation of Path Delay Faults," Proc. of Design Automatic Conference, pp. 357-363, 1989.

[8] F. Fink, K. Fuchs, and M. Schulz, "Robust and Non robust Path Delay Fault Simulation by Paralle Processing of Patterns," IEEE Trans. on Computers, pp. 1527-1536, Dec. 1992.

[9] J. Savir and W. McAnney, "Random Pattern Testability of Delay Faults," Proc. of International Test Conference, pp. 263-273, 1986.

[10] B. Underwood, S. Kang, and W. Law, "A Path-Delay Test Generator for Large VLSI Circuits," Proc. of International Conference on VLSI and CAD, pp. 368-371, 1993.

[11] B. Dervisoglu and G. E. Stong, "Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement," Proc. of International Test Conference, pp. 365-374, 1991.

[12] K. Cheng, S. Devadas, and K. Keutzer, "A Partial Enhanced-Scan Approach to Robust Delay-Fault Generation for Sequential Circuits," Proc. of International Test Conference, pp. 403-410, 1991.

[13] J. Savir, "Skewed-Load Transition Test: Part I, Calculus," Proc. of International Test Conference, pp. 705-713, 1992.

[14] S. Patil and J. Savir, "Skewed-Load Transition Test: Part II, Coverage," Proc. of International Test Conference, pp. 714-722, 1992.

[15] J. Savir and R. Berry, "At-Speed Test Is Not Necessarity and AC Test," Proc. of International Test Conference, pp. 722-728, 1991.

[16] K. Cheng, "Robust Delay-Fault Test Generation and Synthesis for Testability Under Standard Scan Design Automatic Conference," pp. 80-86, 1991.

[17] F. Brglez et. al., "Combinational profiles of sequential benchmark circuits," Proc. of International Symp. Circuits and Systems, pp. 1929-1934, 1989.

강 성 호

1986년 2월 서울대학교 제어계측공학과 졸업(학사)

1988년 5월 The University of Texas Austin 전기 및 컴퓨터공학과(공학석사)

1992년 5월 The University of Texas Austin 전기 및 컴퓨터공학과(공학박사)

1989년 11월~1992년 8월 Schlumberger Inc. Research Scientist

1992년 9월~1992년 10월 The Univ. of Texas at Austin Post Doctoral Fellow

1992년 8월~1994년 6월 Motorola Inc. Senior Staff

Engineer
1994년 9월~현재 연세대학교 전기공학과 조교수
관심분야:CAD 및 VLSI, 테스팅, 시뮬레이션, 검증

임 용 태
1995년 2월 연세대학교 전기공학과 졸업(학사)
1995년 9월~현재 연세대학교 전기공학과 대학원 석사과정(3학기)
관심분야:CAD 및 VLSI, 테스팅

강 용 석
1995년 2월 연세대학교 전기공학과 졸업(학사)
1995년 9월~현재 연세대학교 전기공학과 대학원 석사과정(3학기)
관심분야:CAD 및 VLSI, 테스팅