2000年度 大韓電子工學會 秋季綜合學術大會 論文集 第23卷 第2號 2000/11

고장 시뮬레이션 가속 알고리즘에 관한 연구

장 현 익 , 김 규 철
단국대학교 대학원 컴퓨터공학과
전화 : 02-709-2591

A Study on the Acceleration Algorithm of Fault Simulation

Hyeon-ik Jang , Kyu-chull Kim
Dept. of Computer Science of Graduate School of Dankook University
E-mail : crow1999@dankook.ac.kr

Abstract

Akers Algorithm has more useful information and technique of undetectable faults for fault simulation. To verify of this algorithm, we have constructed A1, A2 and A3 simulator and surveyed simulation time.

I. 서 론

테스팅이란 회로의 고장 유무를 확인하는 작업이다. 테스팅을 완벽히 함으로써 회로의 신뢰도를 높이고 오동작에서 오는 피해를 최대한 줄일 필요가 있다.

최근 설계 기술의 발달과 자동화, 그리고 공정 기술의 발달로 회로의 집적도가 크게 향상되고 있다. 동시에 초집적 회로의 제조 가격에서 테스팅이 차지하는 비중이 크게 상승함에 따라 출고 제품의 신뢰도 향상뿐만 아니라 제품의 가격 인하 및 이익 증대 측면에서도 테스팅이 매우 중요한 문제로 부각되고 있다.

테스트 기술에서 기본이 되는 고장 시뮬레이션은 회로 내에 가상적으로 고장을 주입하고 고장에 의한 회로의 반응으로부터 주어진 테스트 패턴으로 회로의 고장을 검출할 수 있는지 여부를 판정하는 실험이다[3].

기존의 방식을 이용하여 고장 시뮬레이션을 실행할 경우, 검출되지 않은 모든 고장에 대하여 시뮬레이션을 수행하기 때문에 많은 시간이 소비된다.

이러한 문제점을 해결하기 위해 본 논문에서는 직접 시뮬레이션 해야하는 고장의 수를 줄일 수 있는 알고리즘(Akers Algorithm)[1]과 결합하여 고장 시뮬레이션을 수행하였다.

II. 고장 시뮬레이션

고장 시뮬레이션은 무고장 회로(good circuit)와 고장이 주입된 회로(faulty circuit)에 동일한 테스트 패턴을 인가하여 그 회로의 주출력단에서의 반응을 비교하여 반응의 차이가 존재하면 인가된 테스트 패턴으로 가정된 고장이 검출됨을 알아내기 위한 과정이다.[3]

본 논문에서는 동시 고장 시뮬레이션 방식을 수행한다. 동시 고장 시뮬레이션은 무고장 회로와 논리 값의 차이가 있는 고장 회로에 대하여만 시뮬레이션을 수행한다. 각 고장을 별개의 이벤트 - 동적 데이타 구조(dynamic data structure)[2.5] - 로 처리하므로 고장 전파가 차단된 고장에 대하여는 이벤트를 새로 만들지 않으므로 더 이상의 시뮬레이션을 수행하지 않아 불필요한 연산을 줄일 수 있다.

Ⅲ. 고장 시뮬레이션 가속 알고리즘

Akers Algorithm은 주어진 테스트 벡터에 대하여 검출할 수 없는 고장(undetectable fault)들의 집합을 미리 결정하기 위해 제안된 알고리즘이며 기본 목적은 직접 시뮬레이션 해야하는 고장의 수를 줄여 고장 시뮬레이션의 시간을 감소시키는 것이다.

이 알고리즘은 먼저 정상 회로에 대한 논리 시뮬레이션을 수행하여 각 회로의 논리 값을 결정하고 출력에서 입력까지의 모든 게이트를 처리하는 진행을 갖는다. 그렇게 함으로써 주어진 벡터에 의해 검출할 수 없는 고장들의 유용한 정보를 드러낸다.

고장을 검출할 수 없는 선은 'X'표시를 하고, 그 반대의 선은 '*'를 표시하여 S^x와 S^*의 집합을 유도해 낸다. S^*는 회로의 모든 주출력 값들을 완벽하게 결정하는 신호선들의 집합이다. 또한 S^x 선들 상의 단일 고착 고장은 검출 할 수 없으며 따라서 고장 시뮬레이션 과정에서 제외시킬 수 있다.

'X'로 표시된 선들 $L_1,L_2,,...,L_i$의 집합 $\{S_i^x\}$와 '*'로 표시된 선들의 집합 $\{S_i^*\}$을 살펴보면, 각각의 S_i^x와 S_i^*에 대해 다음과 같은 등식이 성립함을 알 수 있다.

$$S^x = \bigcup_i S_i^x \qquad S^* = \bigcap_i S_i^*$$

그러나 이것으로 검출할 수 없는 모든 고장들을 포함하는 것은 아니다. 팬아웃 스템(fanout stem)의 고착 고장은 검출할 수 없으나 그 브랜치(branch)들 중의 하나는 고착 고장을 검출할 수 있다고 가정하자. 만약 S^x에 스템이 위치하고 S^*에 브랜치가 있다면 S^x의 스템에서 S^*의 브랜치에 이르는 명백한 활성화 경로가 존재한다. 이것은 S^x에서 S^*의 선에 이르기까지 활성화 경로가 없어야 한다는 조건에 반대되는 것이다. 이와 같이 상기의 알고리즘으로 검출할 수 없는 모든 고장들을 정의 할 수 없다.

또한 만약 선택 값이 하나 이상일 경우에는 임의의 입력을 선택하는 것으로써 모든 게이트에 대한 처리과정을 수행한다. 이것은 팬아웃 스템의 단일 고착 고장이 게이트의 입력 두개 이상을 연속적으로 변화시킬 수 있기 때문이다. 그러나 이러한 비결정적 선택(non-deterministic choice)때문에 각각의 실행에서 만들어지는 다양한 선택은 적지 않은 시간의 낭비를 가져온다.

그래서 Akers Algorithm은 제어 벡터들을 고려하는 것으로 알고리즘을 확장하였다. 이 방법은 알고리즘을 실행하기 전에 각각의 게이트에 나타나는 입력 벡터들이 제어 벡터인지 아닌지를 결정하여 제어 벡터를 갖는 게이트에 대한 연산을 피한다.

제어 벡터를 확인하는 방법은, 우선 주입력에 패리티(A와 A')를 지닌 심볼을 전파하고 이것이 각 게이트의 입력에 도달하게 되면 그 집합을 수집하여 제어 벡터를 식별하는 과정을 거친다.

전처리로 계산된 (A,A'D'E'), (A,E'), (A',B,E'), (A,B,C')의 문자 집합을 갖는 4입력 NAND 게이트를 고려하자. 먼저 처음과 네 번째에 나타나며 짝수 패리티의 경로를 갖는 문자 A를 살펴보면, _11_의 형태에서는 어떠한 입력도

제어 벡터가 될 수 없다는 것을 알 수 있다. A'에 의해 __1의 모든 벡터들도 제어 벡터에서 제외된다. B는 11__을 제거한다. 등등. 모든 조건들을 고려하면 0000, 1000, 0100, 0010 그리고 1010의 제어 벡터가 존재한다.

입력이 제어 벡터이고 출력이 '*'로 표시된 '0'의 값을 갖는 u, v, w의 3입력 NAND게이트를 고려하자. 확장하기 전의 Akers Algorithm이 세 가지 입력선 중 임의의 하나를 선택해서 '*'표시를 하고 나머지 두개는 'X'표시를 하는 반면, 제어 벡터를 고려하여 확장된 Akers Algorithm은 세 개의 입력선 모두에 'X'표시를 한다는 것에 주목하자.

이와 같이 Akers Algorithm은 검출할 수 없는 고장들의 집합들을 빨리 식별하는 기술을 제공한다.

Ⅳ. 시뮬레이션 실행 결과 및 결론

4.1 시뮬레이션 실행 결과

Circuit	Number of Faults	Number of Runs (Random Vector)	FS+CPT (A1)	FS+A (A2)	FS+A+CPT (A3)
	fault coverage	FS run time	(mean time -> 1/100 sec)		
C432	524	100	0.038 증가	0.088 감소	0.092 감소
	0.78%	0.856			
C499	758	100	0.044 증가	0.049 증가	0.028 증가
	0.96%	1.141			
C880	942	50	0.062 증가	0.154 감소	0.166 감소
	0.67	1.352			
C1908	1879	50	0.091 증가	0.209 감소	0.228 감소
	0.65%	2.014			
C3540	3328	50	0.164 증가	0.301 감소	0.347 감소
	0.71%	3.227			

A : Akers Algorithm
FS : 동시 고장 시뮬레이션(Concurrent Fault Simulation)
CPT : 임계 경로 추적 기법(Critical path tracing)

표 1. A1, A2, A3 시뮬레이터의 수행시간 비교

본 논문에서는 동시 고장 시뮬레이션에 임계 경로 추적(critical path tracing)기법과 Akers Algorithm을 적용하여 고장 시뮬레이션의 수행 시간을 비교하였다.

표 1은 ISCAS85[4] 회로에 세 가지 고장 시뮬레이터 A1, A2, A3을 적용한 결과를 보여준다.

A1 고장 시뮬레이터는 동시 고장 시뮬레이션과 임계 경로 추적 기법을 결합하였다. 그 결과 동시 고장 시뮬레이션만을 수행한 결과 보다 더 많은 시간이 걸렸다. 이것은 임계 경로 추적 기법이 팬아웃과 둘 이상의 제어 벡터에서 진행이 멈추어 많은 고장을 검출하지 못하는 단점 때문이다.

A2 고장 시뮬레이터는 동시 고장 시뮬레이션과 Akers Algorithm을 결합하였다. 임계 경로 추적 기법의 단점인 팬아웃의 문제점을 해결하였다. 검출할 수 없는 고장들의 목록을 제거한 나머지 고장들에 대해서만 시뮬레이션을 수행하여 시간의 단축을 가져왔다.

A3 고장 시뮬레이터는 동시 고장 시뮬레이션과 임계 경로 추적 방식 그리고 Akers Algorithm을 모두 결합하여 수행하였다.

임계 경로 추적 기법과 Akers Algorithm의 수행으로 검출할 수 있는 고장들과 검출할 수 없는 고장들의 목록을 제외한 나머지 고장들에 대해서만 고장 시뮬레이션을 수행하는 이 기법은 보다 적은 시뮬레이션 시간을 보인다.

검증 결과 A3 시뮬레이터가 가장 빠른 처리 속도를 보였다. 또한 Akers Algorithm에 지적한 것처럼, C499와 같은 XOR 게이트들이 지배적인 회로 또는 XOR 게이트와 동등하게 구성된 회로에서는 수행이 효율적이지 못하다는 것도 검증하였다.

이것은 이전 방식의 고장 시뮬레이션 보다 A2 혹은 A3 시뮬레이터로 구현한 방식이 더 효율적이라는 것을 증명한다.

4.2 결론

Akers Algorithm은 논리 시뮬레이션을 사용하여 주어진 테스트 벡터에 의해 검출되지 않는 다수의 고장을 식별한다. 이 방식은 고장 시뮬레이션을 실행할 때 각 회로에 주입되는 고장의 수와

실행 시간을 단축시켰다.

또한 본 논문에서는 Akers Algorithm에 바탕을 둔 A3 시뮬레이터의 구현으로 고장 시뮬레이션의 시간이 더욱 감소 된 것을 증명했다.

추후 XOR 게이트에 대한 문제 해결로 고장 검출 율의 향상을 기대한다.

참 고 문 헌

[1] Sheldon B. Akers, Balakrishnan Krishnamurthy, Sungju Park, and Ashok Swaminathan, "Why is less information from logic simulation more useful in fault simulation?",

[2] Abramovici, Breuer, Friedman, "Digital systems testing and testable design", IEEE Press, 1990.

[3] 홍성제, 박은세, 강성호, 최호용, 장훈 공저, "테스팅 및 테스팅을 고려한 설계", 홍릉과학출판사, pp. 21-141, 1998.

[4] Brglez, F., and Fujiwara, H., "A Neutral Netlist of 10 Combinational Benchmark Circuit and Target Translator in FORTRAN," Special Session on Recent Algorithms for Gate-Level ATPG with Fault Simulation and Their Performance Assessment, Intl. Symp. on Circuits and Systems, June 1985.