論文2000-37SD-6-10

내장된 자체 테스트에서 경로 지연 고장 테스트를 위한 새로운 가중치 계산 알고리듬

(New Weight Generation Algorithm for Path Delay Fault Test Using BIST)

許 潤 *, 姜 成 昊 *

(Yun Hur and Sungho Kang)

요 약

경로 지연 고장의 테스트 패턴은 두 개의 패턴을 가진 쌍패턴으로 이루어져 있다. 따라서 가중 무작위 패턴 생성 방법을 이용하여 지연 고장 테스트를 하기 위해서는 기존의 고착 고장을 위한 방법과는 다른 새로운 가중치 생성 방법이 적용되어야 한다. 결정론적 테스트 패턴을 이용하여 가중치를 계산할 때는 테스트 패턴의 집합을 패턴간의 해밍 거리가 너무 크지 않도록 분할하여 주는 것이 일반적이나 지연 고장 테스트에 있어서는 이 분할 방법이 너무 많은 가중치 집합을 생성하게 될 수도 있을 뿐만 아니라 부정확한 가중치를 계산하게 될 수도 있다. 따라서 본 논문에서는 결정론적 테스트 패턴의 분할 없이 가중치를 계산하여 고장 시뮬레이션을 생성하는 실험을 해 보았다. ISCAS 89 벤치마크 회로에 대한 실험 결과는 본 논문에서 제시한 경로 지연 고장을 위한 가중치 생성 방법의 효율성을 보여준다.

Abstract

The test patterns for path delay faults consist of two patterns. So in order to test the delay faults, a new weight generation algorithm that is different from the weight generation algorithm for stuck-at faults must be applied. When deterministic test patterns for weight calculation are used, the deterministic test patterns must be divided into several subsets, so that Hamming distances between patterns are not too long. But this method makes the number of weight sets too large in delay testing, and may generate inaccurate weights. In this paper, we perform fault simulation without pattern partition. Experimental results for ISCAS 89 benchmark circuits prove the effectiveness of the new weight generation algorithm proposed in this paper.

I 서 론

최근 VLSI 기술의 발달로 회로의 속도는 빨라지고 크기는 점점 감소함에 따라 테스트 방법은 갈수록 복잡해지고 테스트에 드는 비용은 점점 늘어나고 있다. 전통적인 테스트 방법은 회로의 고착 고장을 검출하는 것이다. 그러나 고착 고장을 위한 테스트 패턴은 회로의 기능적인 면만을 테스트하므로 보통 회로의 지연 고장으로 모델링되는 타이밍상의 오류를 감지할 수 없다는 단점이 있다. 지연 고장 모델은 점 결함(spot defect)에서 뿐만 아니라 VLSI의 제조 공정상의 파라미터의 변동에 기인하는 타이밍 고장을 검출하기 위해서 사용된다. 지연 고장 테스트의 목적은 주어진 회로의 모든 경로에 대해서 주입력에서 주출력까지의 지연이

* 正會員, 延世大學校 電氣·컴퓨터工學科
(Yonsei Univ. Dept. of Electric and Computer Eng.)

※ 이 논문은 1998년도 연세대학교 학술 연구비의 지원에 의하여 이루어진 것임.

接受日字:1999年10月2日, 수정완료일:2000年5月30日

시스템 클럭의 주기보다 짧음을 보장하기 위함이다. 시간적인 측면을 고려하여 회로에 테스트 패턴을 가하기 위해서는 회로의 동작 주파수와 같은 속도로 동작하는 자동 테스트 장치를 필요로 하게 된다. 그러나 자동 테스트 장치의 최고 속도는 오늘날의 고속 VLSI 회로의 높은 동작 주파수를 따라가지 못하고 있으며, 더군다나 고속의 자동 테스트 장치는 매우 고가이기 때문에 이것을 사용하는 것이 테스트 비용의 측면에서 항상 효율적인 것만은 아니다. 따라서 고속의 회로를 테스트하는데 있어서 회로 자체 내에서 테스트를 수행하여 회로 테스트 시간을 줄이고 효율적인 테스트 수행을 가능하게 하는 내장된 자체 테스트 방식에 대한 연구가 활발하다.

지연 고장을 검출하기 위해서는 한 쌍의 테스트 패턴을 필요로 한다. 첫 번째 패턴은 테스트하려는 회로의 경로를 원하는 값으로 초기화하고, 두 번째 벡터가 이 경로상에서 천이를 일으켜 이것이 정해진 시간 내에 주출력까지 전파되는지를 검증하면 된다. 지연 고장의 고장 모델로는 크게 게이트 지연 고장과 경로 지연 고장으로 나뉘어진다. 경로 지연 고장 모델을 대상으로 하는 지연 테스트 방식에서는 활성화된 경로상에 존재하는 게이트 지연 고장들을 포함하므로 보다 높은 신뢰성을 갖는 테스트 방식이지만 회로 크기가 증가하면서 고려해야 할 경로의 수가 기하급수적으로 증가하기 때문에 적은 시간 내에 높은 고장 검출율에 도달할 수 있는 효율적인 테스트 생성 방법의 선택이 가장 중요한 문제가 된다.

지연 고장 검출을 위한 내장된 자체 테스트 생성기로 가장 많이 쓰이는 것은 LFSR(Linear Feedback Shift Register)이나 CA(Cellular Automata), MISR(Multiple Input Signature Register), 또는 카운터이다. 이 때 전수 패턴 테스트를 하게 되면 테스트하려는 회로의 주입력의 수를 n이라고 하면 가능한 모든 쌍패턴의 조합의 수는 $2^n(2^n-1)$이 되는데, 이 수는 n이 증가함에 따라 기하급수적으로 증가하기 때문에 테스트 길이를 줄이기 위한 여러 가지 방법들이 연구되어졌다. 쌍패턴은 첫 번째 패턴과 두 번째 패턴 사이에 서로 다른 비트 수가 여러 개인 MIC(Multiple Input Change)와 단 한 개 뿐인 SIC(Single Input Change)로 나뉘어진다.

[1]과 [2]에서는 상수 병렬 입력 벡터(constant parallel input vector)를 가지는 MISR을 사용하는 방법을 제안하였다. 먼저 검출 가능한 모든 경로에 대해 결정론적 테스트 패턴을 구한 후 이 패턴들을 생성할 수 있는 최소수의 상수 MISR 입력 벡터의 수를 계산하는 방식이다. [3]에서는 LFSR를 이용한 RPG(Random Pattern Generator)와 SSR(Scan Shift Register)를 사용하여 적은 하드웨어 오버헤드만으로 의사 무작위 SIC 패턴 생성기를 구성하는 방법에 대해 설명하고 있다. 위에서 설명한 여러가지 전수 패턴 테스트 및 의사 무작위 패턴 생성기는 대체적으로 하드웨어 오버헤드가 적다는 장점이 있으나 지연 고장 중에는 무작위 패턴만으로는 잘 검출되지 않는 무작위 패턴 저항 고장(random pattern resistant fault)들이 존재하는 경우가 많고 이런 고장들을 검출하기 위해서는 매우 긴 테스트 길이가 필요하기 때문에 테스트 시간의 측면에서 실제적으로 적용하기에는 비효율적이라는 단점을 갖는다.

무작위 패턴 저항 고장을 쉽게 검출하려면 입력 패턴의 각 비트별로 1이 생성되는 확률에 대해 가중치를 주는 가중 무작위 테스트 패턴을 사용하여야 한다. 이러한 가중 무작위 테스트 패턴 생성 방법은 주로 고착 고장(stuck-at fault)을 검출하기 위한 방법들이 연구되어졌다. 가중치를 계산하는 방법은 크게 회로의 구조 분석을 통한 방법과, 결정론적 테스트 패턴을 먼저 구하고 이를 바탕으로 가중치를 구하는 방법과 이 두 가지 방법을 혼용하여 쓰는 방법으로 나눌 수 있다. 회로의 구조를 분석하는 방법은 ATPG(Automatic Test Pattern Generator)의 과정에서 사용할 수 있다는 장점이 있지만 높은 고장 검출율을 얻지 못하는 경우가 많고, 결정론적 테스트 패턴을 이용하는 방법은 ATPG 과정에서는 이용할 수 없지만 높은 고장 검출율을 얻을 수 있다는 장점이 있다. 전자의 방법으로 [4, 5]에서는 비제어값(noncontrolling logic value)의 전파 확률을 고려하여 가중치를 구하였고 [6]에서는 마코프 체인(Marcov chain)을 사용하여 각 게이트들의 천이 발생 확률을 구하여 평균하는 방식으로 가중치를 구하였다. [7, 8, 9]과 같은 논문에서는 결정론적 테스트 패턴을 구하고 압축하여 가중치를 구하는 방법을 사용하였다. 이와 같이 고착 고장에 대해서는 여러 가지 가중 무작위 테스트 생성을 위한 가중치 계산 방법들이 연구되었으나 지연 고장에 대해서는 발표된 논문이 별로 많지 않은 실정이다. [10]에서는 초기화 패턴으로 제어값

(controlling logic value)과 팬인(fan-in)수를 고려하여 주출력에서 주입력까지의 전파 확률을 계산하였으나 적절한 패턴 수 내에서 얻을 수 있는 고장 검출율의 한계를 극복하지는 못하였고 더군다나 대부분의 회로에서는 단일 가중치 집합으로는 높은 고장 검출율을 얻을 수가 없다. [11]의 논문에서는 결정론적 테스트 패턴을 먼저 구하여 이것을 이용하여 첫 번째 패턴을 생성하고 이 패턴의 모든 비트들을 한번씩 천이시켜, 주입력 수가 n개라고 하면 하나의 첫 번째 패턴에 대해서 천이가 일어나는 비트의 위치가 서로 다른 n개의 SIC 쌍패턴이 만들어지게 된다. 이 논문의 결과는 앞에서 설명한 방법이 작은 ISCAS 89 회로에 대해서 성공적으로 적용되고 있음을 보여준다. 그러나 이 방법은 경로 지연 고장을 하나도 검출할 수 없는 패턴에 대해서도 그 패턴을 초기화 패턴으로 갖는 n개의 쌍패턴이 만들어져야 한다. 따라서 어떤 경로에 대한 지연 고장도 검출할 수 없는 초기화 패턴이 만들어지게 되면 역시 어떤 경로도 테스트할 수 없는 n개의 쌍패턴이 만들어지게 된다. 이런 방법은 주입력의 수가 작을 때는 별 문제가 되지 않을 수도 있지만 주입력의 수가 많아지게 되면 효율이 떨어지게 된다. 가중 무작위 패턴 방식의 성질상 갈수록 새로운 경로에 대한 지연 고장을 검출할 확률이 떨어지게 되어 새로운 하나의 경로에 대한 지연 고장을 검출하기 위해서도 많은 쌍패턴이 필요하게 되는데 이 때도 하나의 초기화 패턴에 대해서 주입력수만큼의 쌍패턴이 만들어지게 되면 주입력의 수가 많이질수록 낭비되는 패턴의 수가 늘어나게 된다. 또한 일정한 수의 쌍패턴을 회로에 가한 후에 새로운 가중치를 계산해 주는 방식을 사용하기 때문에 테스트 길이가 길어지면 길어질수록 필요한 가중치 집합의 수가 늘어나게 된다는 단점을 가진다. 경로 지연 고장 테스트를 하는데 있어서 적은 패턴 수로 높은 고장 검출율을 얻기 위해서는 우수한 초기화 패턴이 만들어져야 될 뿐만 아니라 우수한 두 번째 패턴을 만들 수 있도록 해야만 한다.

이 논문에서는 결정론적 테스트 패턴을 이용하여 초기화 패턴에 대한 가중치와 두 번째 패턴에 대한 가중치를 따로 구하여 테스트 패턴을 생성함으로써 효율적으로 경성 경로 지연 고장을 검출할 수 있는 가중 무작위 테스트 패턴 생성 방법에 대하여 연구하였다.

Ⅱ. 경로 지연 고장 테스팅

1. 경로 지연 고장의 종류

경로 지연 고장에 대한 테스트에는 크게 무해저드 경성 (hazard free robust: HFR) 테스트, 경성(robust: ROB) 테스트, 연성(*non robust*: NR) 테스트의 세 종류로 나눌 수 있다.

무해저드 경성 테스트는 경로에의 신호들에 동적 해저드(dynamic hazard)가 없고 회로의 다른 부분에서의 지연과 무관하게 경로 상의 지연 고장을 검출할 수 있는 두 패턴의 테스트이다. 무해저드 경성 테스트는 지연 고장 진단에 적합하고 시험중인 경로가 초과의 지연을 가지면 테스트가 실패하고 또 테스트가 실패하면 시험중인 경로가 초과의 지연을 갖는다. 경성 테스트는 회로의 다른 부분에서의 지연과 무관하게 경로 상의 지연고장을 검출할 수 있는 두 패턴의 테스트로 경로에 특정 천이를 유발시켜야 하고 경로 상의 모든 신호들은 특정 천이가 도달하기 전까지 최종값을 알 수 없어야 한다. 경성 테스트는 시험중인 경로가 초과지연을 가지면 실패한다. 하지만 이는 또 다른 이유로도 실패할 수 있다. 즉 회로내의 다른 경로가 초과 지연을 갖는 경우이다.

경성 테스트의 요건을 갖추지 못한 테스트를 연성 테스트라고 부른다. 이는 2개로 구분이 될 수 있다. 먼저 경성 테스트에 가까운 연성 테스트를 강연성(strong non-robust) 테스트(SNR) 또는 근사 경성(approximate robust) 테스트라 부른다. 강연성 테스트(근사 경성 테스트)는 모든 요건이 경성 테스트와 같지만 경성 테스트 중에서 경로외 입력에 정적 해저드(static hazard)가 없어야 하는 경우에도 강연성 테스트에서는 정적 해저드가 있어도 무방한 두 패턴의 테스트이다. 두 번째 사이클의 경우에 경성 테스트와 요구되어지는 값이 같고 첫 번째 사이클의 경우 요구되어지는 것이 경로의 시작 부분의 초기값으로만 결정되는 방법으로 연성 테스트를 구하였는데 이를 약연성(weak non-robust)테스트(WNR)이라고 부른다. 약연성 테스트는 경로 상의 천이가 도달하기 전에 회로의 다른 부분이 최종값으로 안정된다고 가정할 수 있다면 경로 상의 지연 고장을 검출할 수 있는 두 패턴의 테스트이다. 약연성 테스트를 이용할 경우는 경로를 제외한 회로의 나머지 부분

에서는 지연 고장이 없어야 경로 상의 지연 고장을 검출할 수 있다.

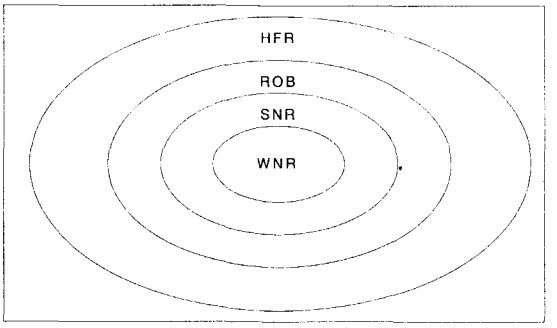


그림 1. 테스트의 정의에 따른 요건
Fig. 1. The conditons with test definition.

이 4가지 종류의 테스트의 정의에 따른 요건들을 비교하면 앞의 그림 1과 같다. 위의 그림에서 보면 경성 테스트의 요건 집합이 강연성 테스트의 요건 집합을 포함한다. 이는 경성 테스트를 구하기 위한 요건이 더 많다는 것이며 따라서 더 구하기가 어렵다. 따라서 경성 테스트는 강연성 테스트보다는 고질의 테스트이며 어떤 한 경로에 따라 경성 테스트가 있으면 강연성 테스트도 존재한다. 그러나 강연성 테스트가 있다고 해서 반드시 경성 테스트가 존재하는 것은 아니다. 이러한 관계는 강연성 테스트와 약연성 테스트에 대해서도 마찬가지이다. 각 유형의 테스트에 대해 경로외 입력이 가져야 하는 입력 제한 조건을 AND(NAND) 와 OR(NOR) 게이트를 예로 표 1에 나타내었다. 표 1에서 S_1과 S_0는 각각 두 시간 프레임동안 해저드 없는 논리값 1과 0을 가짐을 나타내고 11과 00은 두 시간 프레임 동안 각각 논리값 1과 0을 가지지만 해저드가 없음을 보장할 수 없는 경우를 나타낸다. X_1과 X_0는 각각 첫 번째 시간 프레임에서는 논리값 X를, 두 번째 시간 프레임에서는 각각 논리값 1과 0을 가짐을 나타낸다.

표 1.각 유형의 테스트에 대한 경로외 입력의 제한 조건
Table 1. The restriction conditions of off-path input to test type.

소자 유형	경로입력 천이	경로외 입력 조건			
		HFR	ROB	SNR	WNR
AND (NAND)	상승 천이	S1	X1	X1	X1
	하강 천이	S1	S1	11	X1
OR (NOR)	상승 천이	S0	S0	00	X0
	하강 천이	S0	X0	X0	X0

앞에서 설명한 바와 같이 경성 테스트는 시험중인 경로가 초과 지연을 가지면 실패한다. 하지만 이는 또 다른 이유로도 실패할 수 있다. 즉 회로내의 다른 경로가 초과 지연을 갖는 경우이다. 따라서 회로에 대한 테스트의 목적이 진단이 필요한 경우가 아니고 단지 고장 검출 만이라면 테스트 패턴을 구하는 방법이 훨씬 더 쉬운 경성 테스트를 구하는 것이 무해저드 경성 테스트를 구하는 것보다 유리하다.

2. 논리값

표 2에서는 표준 주사 환경에서 필요한 11가지 논리값들을 보여주고 있다. 각 논리값의 앞의 값은 첫 번째 시간 프레임의 초기화 벡터를 나타내고 두 번째 값은 두 번째 시간 프레임의 값을 나타낸다. 조합회로의 경우 경로 지연고장 검사입력의 생성을 위해 5개의 논리값(logic value), {S0, S1, X0, X1, X}을 사용한다. 여기서 S는 두 시간 프레임 동안 해저드가 없이 안정(stable)함을 의미하고 X는 무상관(don't care)을 의미한다. 그러나 이 논리값들로는 표준 주사 환경에서 기능적 정당화(functional justification) 방법을 사용하기 위해서는 불충분하다. 본 논문에서 적용한 기능적 정당화 방법은 단지 시간 프레임이 두 개로 제한돼 있다는 것을 제외하면 순차회로의 검사입력 생성을 위해 사용하는 시간 프레임 확장기술과 비슷하다. 따라서 플립플롭의 출력에 X1을 정당화하기 위해서는 플립플롭의 입력에 1X라는 값을 필요로 한다. 이와 같은 이유로 플립플롭에서 기능적 정당화 방법을 사용하기 위해서는 위에서 말한 5개의 논리값에 6개를 추가하여 최소한 표 2와 같은 11개의 논리값을 필요로 하게 된다.

표 2. 11가지 논리값
Table 2. 11 logic values.

기능적 정당화를 위한 논리값	안정
00 X0	S0
11 X1	
01 0X	S1
10 1X	
XX	

Ⅲ. 가중 무작위 경로 지연 고장 테스팅

그림 2는 가중 무작위 경로 지연 고장 테스트의 알고리듬을 보여주고 있다. 가장 처음으로 수행하는 과정은 determine_path()로 회로의 경로를 결정하는 일이다. 경로 지연 고장 테스트는 경로에 흩어져 있는 작은 지연들이 모여서 발생하는 지연 고장까지도 검출할 수 있다는 점에서 소자에 집중되어 있는 지연 고장밖에 검출하지 못하는 게이트 지연 고장 테스트에 비해서 더 신뢰성있는 테스트 방법이기는 하나, 회로의 경로의 수는 회로의 크기가 커짐에 따라 기하급수적으로 증가하기 때문에 이들 모든 경로를 다 테스트한다는 것은 불가능한 일이다. 따라서 이번 실험에서는 회로의 경로의 수를 5000개로 제한하였다. 만일 회로 전체 경로의 수가 5000개 보다 작은 경우에는 모든 경로를 고려하고, 회로 전체 경로의 수가 5000개 이상인 경우에는 무작위로 선택된 5000개의 경로만 고려하였다.

회로의 경로가 정의되었으면 deterministic_test_generation()에서 이들 경로에 대하여 테스트 패턴을 생성한다. 그리고 이들 테스트 패턴을 이용하여 calculate_weights()에서 가중 무작위 경로 지연 고장 테스트를 위한 가중치를 생성한다. 본 논문에서 제시하는 경로 지연 고장 테스트를 위한 가중치의 생성 방법은 고착 고장 테스트를 위한 가중치의 생성 방법과 거의 유사하다. 다른 점이 있다면 고착 고장 테스트의 경우 하나의 고장에 대한 테스트 패턴이 단 하나의 패턴으로 이루어져 있지만, 경로 지연 고장 테스트의 경우 하나의 경로에 대한 테스트 패턴이 초기화 패턴과 두 번째 패턴이라는 두 개의 패턴으로 이루어진 쌍패턴이

```
weighted_random_pattern_generation()
{
    not_detect = 0
    determine_paths()                          //테스트할 경로를 결정한다.
    deterministic_test_generation()            //결정론적인 테스트 패턴을 구한다.
    calculate_weights()                        //가중치를 계산한다.
    round_weights()                            //가중치를 라운딩한다.
    while (fault_coverage < 100)
    {
        weighted_random_fault_simulation()     //가중 무작위 고장 시뮬레이션을 수행한다.
        if (new_detected_fault = 0)            //새로 검출된 고장이 없으면 not_detect를
            not_detect++                         1 증가시키고 새로 검출된 고장이 있으면
        else                                     not_detect를 0으로 초기화시킨다.
            not_detect = 0
        if ((not_detect×32) = k)               //k개의 연속적인 가중 무작위 패턴이
        {                                        새로운 고장을 검출하지 못했을 때는
            deterministic_test_generation()      아직 검출되지 않은 새로 가중치를 계산한
            calculate_weights()                  다.
            round_weights()
        }
    }
}
```

그림 2. 가중 무작위 경로 지연 고장 테스트 알고리듬
Fig. 2. Weighted random path delay fault test algorithm.

기 때문에 각각의 주입력에 대해서 초기화 패턴에 대한 가중치와 두 번째 패턴에 대한 가중치를 따로 구하여야 한다. 따라서 회로의 주입력의 수가 n개라고 한다면 각각 n개의 가중치들로 이루어진 가중치 집합이 2개가 생성되게 된다. 가중치의 생성 방법은 다음과 같다.

주입력의 수가 n개인 회로에서 , (t12, t22), …, (t_{1n}, t_{2n})을 결정론적 테스트 패턴 집합이라고 하자. 여기서 (t_{1j}, t_{2j})은 결정론적 테스트 패턴 집합의 j번째 쌍패턴이고 t_{1j}과 t_{2j}은 각각 j번째 쌍패턴의 초기화 패턴과 두 번째 패턴을 의미한다. 그리고 $t_{1j}[i]$를 j번째 결정론적 테스트 패턴의 초기화 패턴 t_{1j}의 i번째 비트라고 하고 $t_{2j}[i]$를 j번째 결정론적 테스트 패턴의 두 번째 패턴 t_{2j}의 i번째 비트라고 하자. 또한 편의를 위하여 초기화 패턴들로만 이루어진 집합 T_1과 두 번째 패턴들로만 이루어진 집합 T_2를 정의하겠다.

$$T1 = \{t_{11}, t_{12}, \cdots, t_{1n}\}$$
$$T_2 = \{t_{21}, t_{22}, \cdots, t_{2n}\}$$

W_1과 W_2는 각각 i번째 비트의 초기화 패턴에 대한 가중치 w_{1i}와 번째 비트의 두 번째 패턴에 대한 가중치 w_{2i}로 이루어진 가중치 집합을 의미한다. 이 때 초기화 패턴의 i번째 비트의 가중치 w_{1i}는 다음 식에 의해 구할 수 있다.

$$w_{1i} = \frac{|\{t_{1j} \in T_1 \mid t_{1j}[i] = 1\}|}{|\{t_{1j} \in T_1 \mid t_{1j}[i] \neq X\}|} \quad (1)$$

같은 방법에 의해서 두 번째 패턴의 i번째 비트의 가중치 w_{2i}는 다음 식으로 표현될 수 있다.

$$w_{2i} = \frac{|\{t_{2j} \in T_2 \mid t_{2j}[i] = 1\}|}{|\{t_{2j} \in T_2 \mid t_{2j}[i] \neq X\}|} \quad (2)$$

이 때 하나의 결정론적 테스트 패턴 (t_{1j}, t_{2j}) 샘플링 확률 P_j는 다음 식 (3)에 의해 계산할 수 있다.

$$P_j = \prod_{i=1, t_{1j}[i] \neq X}^{n} \{(w_{1i} \times t_{1j}[i]) + (1 - w_{1i}) \times (1 - t_{1j}[i])\} \times \prod_{i=1, t_{2j}[i] \neq X}^{n} \{(w_{2i} \times t_{2j}[i]) + (1 - w_{2i}) \times (1 - t_{2j}[i])\} \quad (3)$$

결정론적 테스트 패턴을 이용해서 가중 무작위 패턴을 생성하는 방법들은 가중치 집합의 수에 따라 두 가지로 분류될 수 있다. 하나는 단일 가중치 집합에 기초한 방법이고 또 다른 하나는 다중 가중치 집합에 기초한 방법이다. 단일 가중치를 이용하게 되면 다중 가중치를 사용하는 경우에 비해서 하드웨어 오버헤드를 크게 줄일 수 있다는 장점이 있다. 그러나 대부분의 회로에 있어서 단일 가중치로는 합리적인 수의 패턴 내에서 높은 고장 검출율을 얻을 수 없는 경우가 많은데 이것은 서로간의 해밍 거리가 너무 먼 결정론적 테스트 패턴들을 한데 묶어 가중치를 계산할 경우 효율적인 가중치의 생성을 방해하기 때문이다. 따라서 결정론적 테스트 패턴 집합을 이용하여 가중치를 계산할 때는 먼저 결정론적 테스트 패턴 집합에 포함되어 있는 패턴들을 적당한 해밍 거리에 따라 여러 개의 부분 집합으로 분할한 뒤, 이 부분 집합에 포함되어 있는 패턴들로만 가중치를 구하는 방법이 일반적이다. 그러나 이러한 패턴 분할 방법을 경로 지연 고장 테스트에서 사용하게 되면 큰 문제가 발생하게 되는데 이것은 경로 지연 고장 테스트를 하기 위해서는 두 개의 테스트 패턴을 필요로 한다는 것과 두 개의 패턴 사이에는 밀접한 연관 관계가 있다는 것 때문이다. 더 자세히 설명하면 두 번째 패턴은 아무 때나 한 번 생성되기만 하면 되는 것이 아니고 어떤 특정한 초기화 패턴이 원하는 경로의 출력을 특정한 값으로 초기화시켜 주었을 때 나와야 한다는 것을 말한다.

우리는 이번 실험에서 먼저 결정론적 테스트 패턴을 구하고 이 결정론적 테스트 패턴들의 초기화 패턴들만의 집합 T_1만으로 하나의 가중치 집합 W_2을 계산하고 결정론적 테스트 패턴들의 두 번째 패턴들만의 집합 T_2만으로 또 하나의 가중치 집합 W_2을 계산하려고 한다. 만일 W_1과 W_2를 계산할 때, 단순히 초기화 패턴의 집합과 두 번째 패턴의 집합을 각각 독립적으로 패턴간의 해밍 거리에 따라 분할한 후에 가중치를 구하게 되면 고장을 검출할 수 없는 엉뚱한 가중치가 생성될 수도 있다. 다음의 예를 생각해 보자.

만일 T가

T = (X11101, X11111), (X11001, X0000), (X10010, X00001)

라고 한다면 초기화 패턴들의 집합 T_1과 두 번째 패턴들의 집합 T_2는 각각

T_1 = {X11101, X11001, X10010}
T_2 = {X11111, X10000, X00001}

가 될 것이다. 여기서 T_1과 T_2 각각을 해밍 거리 3을 기준으로 분할해 보자. 이 말은 패턴간의 해밍 거리가 3보다 작거나 같은 경우에는 같은 부분 집합에 속해질 수 있지만 해밍 거리가 3보다 클 경우에는 서로 다른 부분 집합에 포함시켜야 된다는 것을 의미한다.

두 패턴 p_1과 p_2간의 해밍 거리 $\delta(p_1, p_2)$는 다음의 식 (4)에 의해 계산되어질 수 있다.

$$\delta(p_1, p_2) = \{(p_1 = 0 \wedge p_2 = 1) \vee (p_1 = 1 \wedge p_2 = 0)\} \quad (4)$$

먼저 T_1의 첫 번째 패턴 t_{11} = X11101을 기준으로 하여 T_1을 분할해 보자. T_1의 두 번째 패턴 t_{12} = X11001는 t_{11}과의 해밍 거리 $\delta(t_{11}, t_{12})$ = 1이다. 따라서 t_{12}는 t_{11}과 마찬가지로 같은 부분 집합 T_{11}내에 포함되어질 수 있다. T_1의 세 번째 패턴 t_{13} = X10010은 t_{11}과의 해밍 거리가 4이므로 t_{11}과는 다른 새로운 부분 집합 T_{12}에 포함되어져야 한다. 이번에는 두 번째 패턴들의 집합 T_2를 t_{21} = X11111을 기준으로 하여 같은 방식으로 분할하여 보자. 그러면 t_{22}는 t_{21}과의 해밍 거리가 4이므로 t_{11}과 같은 집합에 속할 수 없고 새로운 부분 집합 T_{22}에 속하게 된다. 마지막으로 t_{23}은 t_{11}과의 해밍 거리가 4이므로 T_{21}에는 속할 수 없고 t_{22}와의 해밍 거리는 1이므로 T_{22}에 속하게 된다. 분할된 결과는 다음과 같을 것이다.

T_{11} = X11101, X11001, T_{12} = X10010
T_{21} = X11111, T_{22} = X10000, X00001

이번에는 분할된 패턴들을 이용하여 가중치를 계산해 보자. 초기화 패턴에 대한 첫 번째 가중치는 집합 내에 패턴의 수가 가장 많은 T_{11}을 이용하여 계산하고 두 번째 패턴에 대한 첫 번째 가중치도 역시 패턴의 수가 가장 많은 T_{22}를 이용하여 계산하게 된다. 그러면 초기화 패턴의 i번째 비트의 가중치 $w1_i$와 두 번째 패턴의 i번째 비트의 가중치 $w2_i$는 각각 표 3과 같다.

표 3. 계산된 가중치
Table 3. The calculated weights.

w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}
0.5	1	1	0.5	0	1
w_{21}	w_{22}	w_{23}	w_{24}	w_{25}	w_{26}
0.5	0.5	0	0	0	0.5

이 가중치를 사용하여 가중 무작위 패턴을 생성한다고 했을 때 첫 번째 결정론적 테스트 패턴 (t_{11}, t_{21})이 샘플링될 확률을 계산해 보면 0이 나와서 (t_{11}, t_{21})이라는 패턴은 절대로 샘플링 될 수가 없다. 이와 같이 잘못된 가중치가 계산된 이유는 초기화 패턴과 두 번째 패턴간의 상관 관계를 무시한 채 패턴을 분할했기 때문이다. t_{21}은 t_{11}이 초기화 패턴인 상태에서 두 번째 패턴으로 생성될 때만 어떤 경로에 대한 고장을 검출할 수 있다. 따라서 경로 지연 고장 검출을 위한 가중치 생성 과정에서 패턴을 분할할 때는 먼저 초기화 패턴을 적당한 해밍 거리에 따라서 분할하되, 두 번째 패턴들은 같이 쌍패턴을 이루는 초기화 패턴과 같은 방식으로 패턴의 집합을 나눠줘야 한다. 위의 예에서 초기화 패턴들은 첫 번째 초기화 패턴 t_{11}과 두 번째 초기화 패턴 t_{12}가 하나의 부분 집합을 이루고, 세 번째 초기화 패턴 t_{13}이 또 하나의 부분 집합을 이루도록 분할하였다. 따라서 두 번째 패턴들도 우선은 t_{21}과 t_{22}를 하나의 부분 집합으로 나누고 t_{23}를 또 하나의 부분 집합으로 분할하여야 한다. 따라서 T_1과 T_2의 부분 집합은 다음과 같이 될 것이다.

T_{11} = {X11101, X11001} , T12 = {X10010}
T_{21} = {X11111, X10000} , T22 = {X00001}

그런데 두 번째 패턴을 단순히 첫 번째 패턴과 같은 방식으로 분할하게 되면 두 번째 패턴의 각 부분 집합들간의 해밍 거리는 전혀 고려가 되지 않았다. 따라서 여기서 좀 더 효율적인 가중치를 구하고자 한다면 분할된 두 번째 패턴의 각 부분 집합들을 적당한 해밍 거리를 기준으로 하여 다시 한 번 분할해 주어야 한다. 그런데 두 번째 패턴들의 집합을 먼저 초기화 패턴에 따라서 분할하고 다시 두 번째 패턴들간의 해밍 거리에 따라서 나누게 되면 분할되는 부분 집합의 수가 너무 많아지게 된다. 또한 패턴을 분할하는 기준이 되는 최적의 해밍 거리를 찾기 위해서는 초기화 패턴뿐만 아니라 두 번째 패턴의 해밍 거리도 바꿔가며 고장 시뮬레이션을 해 보아야 하기 때문에 주입력의 수가 큰

회로에 대해서 가중치를 구하기 위해서는 너무 많은 수의 고장 시뮬레이션을 수행하여야만 한다. 따라서 본 논문에서는 패턴의 분할 없이 전체 결정론적인 테스트 패턴을 이용하여 가중치를 계산하고 이 가중치를 이용하여 가중 무작위 테스트 패턴을 생성하여 고장 시뮬레이션을 수행한 뒤, 아직 검출되지 않은 고장들에 대해서 다시 결정론적인 테스트 패턴을 생성하여 새로운 가중치를 계산하는 방법을 제안한다.

위에서 구한 가중치를 생성하기 위한 회로를 하드웨어로 구현하게 되면 하드웨어 오버헤드가 대단히 커지게 된다. 따라서 실제의 경우에 있어서 가중치 무작위 패턴 테스트를 위한 가중치 회로를 설계하기 위해서는 가중치의 라운딩이 필수적이다. round_weights()에서는 가중치를 {0, 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, 1}의 9개의 단계로 라운딩한다. 라운딩 방법은 실제의 가중치 값을 위의 9개의 라운딩된 가중치 중에서 가장 가까운 값을 취하는 방식인데 가중치로 0과 1을 택할 때는 주의하여야 한다. 그 이유는 만일 라운딩하지 않은 가중치의 값이 1/32라고 할 때 이 가중치를 라운딩하게 되면 0이 되어서 가중 무작위 패턴 중에서 그 비트에는 1이 나오지 않게 된다. 그러나 라운딩 되지 않은 가중치의 값이 작기는 하나 정확히 0이 아니라는 것은 대략 32개의 패턴중에 한번쯤은 1이 나와야 한다는 것을 뜻한다. 따라서 그 비트의 가중치가 0으로 라운딩되게 되면 그 비트에 1을 필요로 하는 경로는 테스트를 할 수가 없다는 이야기가 된다. 따라서 가중치를 라운딩할 때 라운딩하지 않은 가중치의 값이 아무리 작다고 할지라도 정확히 0인 경우에만 0으로 라운딩을 해 주고 그 외의 경우에는 1/8로 라운딩을 해 주어야 한다. 같은 이유로 라운딩하지 않은 가중치가 정확히 1일 때만 1로 라운딩해 주어야 한다.

라운딩의 과정이 끝나게 되면 weighted_random_fault_simulation()에서는 고장 검출율이 100%에 도달할 때까지 계산된 W_1을 이용하여 가중 무작위 쌍패턴의 초기화 패턴을 생성하고 W_2를 이용하여 가중 무작위 패턴의 두 번째 패턴을 생성한다. 이 가중 무작위 쌍패턴은 한 번에 32개 단위로 생성되어 병렬적으로 동시에 고장 시뮬레이션 되어진다. 32개의 가중 무작위 패턴의 고장 시뮬레이션이 끝날 때마다 고장 검출율을 계산하여 고장 검출율이 100%에 도달하였으면 고장 시뮬레이션 알고리듬을 끝마치게 되고 아직 100%에 도달하지 못 하였을 때에는 다시 32개의 가중 무작위 패턴을 생성하여 고장 시뮬레이션을 계속한다. 만일 이 32개의 가중 무작위 테스트 패턴이 새로운 경로에 대한 고장을 검출하지 못하게 되면 not_detect를 1만큼 증가시키게 되고, 새로 검출된 고장이 있으면 not_detect를 0으로 초기화시킨다. 이 과정을 계속하다가 not_detect가 k/32와 같아지게 되면, 즉 다시 말해서 연속적인 k개의 패턴이 새로운 고장을 검출하지 못할 때에는 가중 무작위 패턴의 생성을 중단하고 아직 검출되지 않은 경로들에 대한 결정론적 테스트 패턴을 다시 생성하여 새로운 가중치를 계산하고 라운딩하여 이 새로운 가중치로 가중 무작위 고장 시뮬레이션을 계속하게 된다. 위의 과정은 고장 검출율이 100%에 도달하게 될 때까지 계속한다.

Ⅳ. 실험 결과

본 알고리즘은 C 언어로 구현하였으며 회로는 표준 ISCAS 89 벤치마크 회로를 사용하였다. 벤치마크 회로에서 경로는 회로 전체의 경로가 5000개가 안 되는 경우에는 회로의 전체 경로에 대하여 실험을 하였고, 회로 전체의 경로가 5000개 이상일 때는 무작위로 5000개의 경로를 추출하여 실험을 하였다. 본 실험에서는 경성 지연 고장을 테스트하는 방법을 목표로 하였는데 경성 지연 고장 테스트는 앞에서 설명한 바와 같이 테스트 패턴을 구하는 과정이 연성 지연 고장의 경우보다 어렵기는 하나 연성 지연 고장의 모든 조건들을 포함하는 테스트 방법이기 때문에 연성 지연 고장 테스트에 비해 더 고품질의 테스트 방법이기 때문이다. 모든 회로에 초기화 패턴을 가할 때 플립플롭에는 주사 방식을 사용하여 패턴을 가하고* 두 번째 패턴은 회로 내에서 기능적 정당화(functional justification)를 통해서 구해진다. 일반적인 주사 방식의 회로에서 플립플롭의 두 번째 패턴까지 제어하기 위해서는 두 번째 패턴도 주사 사슬을 통해서 시프팅을 해 줘야 하기 때문에 테스트 시간이 너무 오래 걸리게 된다. 이를 막기 위해서는 개선된 주사 방식을 사용하여야 하는데 이를 위해서 사용되는 플립플롭의 수 때문에 하드웨어 오버헤드가 너무 커지기 때문에 실제적으로는 거의 사용되지 않는다. 따라서 일반적인 주사 방식은 개선된 주사 방식에 비해 회로의 제어가능성이 떨어지게 되지만 실제

적으로 많이 쓰이는 방식이기 때문에 이 실험 결과는 의미를 갖게 된다.

가중 무작위 지연 고장 테스트 방법에 대하여 연구되어 있는 논문으로는 [10]와 [11]과 같은 것들이 있다. 그러나 우리가 회로들의 전체 경로 중에서 5000개를 선택하여 고장 시뮬레이션을 수행한데 반해서 [10]에서는 회로들의 모든 경로에 대하여 10000개의 패턴을 가했을 때 검출되는 고장의 수를 결과로 싣고 있고, [11]에서는 ISCAS 89회로에 개선된 주사 방식을 사용하여 순차 회로를 조합 회로로 변환한 회로에 대한 결과이기 때문에 본 논문의 결과와는 회로의 경로수와 테스트 가능한 경로의 수가 다르게 된다. 따라서 이번 실험 결과에는 비교할만한 다른 실험 결과를 첨부하지 못하였다.

먼저 가중치를 라운딩하지 않았을 때의 결과가 표 4와 표 5에 나타나 있다. 표 4는 k값이 4800일 때의 결과이고 표 5는 k값이 9600일 때의 결과를 나타낸다. 보통 k값이 작게 되면 높은 고장 검출율을 얻는데 필요한 가중치 집합의 수는 늘어나게 되지만 반면에 필요한 패턴의 수는 작게 되고, 반대로 k값이 커지게 되면 높은 고장 검출율을 얻는데 필요한 가중치 집합의 수는 적어지나 필요한 패턴의 수는 늘어나게 된다.

표 4에서 두 번째 열은 회로의 주입력의 수를, 세 번째 열은 회로의 플립플롭의 수를 나타내고 있고 네 번째 열과 다섯 번째 열은 각각 회로의 전체 경로 수와 이중에서 결정론적인 테스트 패턴의 생성이 가능한 경로의 수를 나타내고 있다. 그 다음 행부터는 새로운 가중치가 계산되기 전까지 회로에 가한 패턴의 수와 이때까지의 고장 검출율를 나타낸다. 표에서 볼 수 있듯이 모든 회로에 대해서 1개에서 3개의 가중치만으로 100%의 고장 검출율을 얻을 수 있었다.

표 5는 k 값이 9600일 때의 결과이다. 예상했던 대로 100%의 고장 검출율을 얻는데 필요한 가중치 집합의 수는 몇몇 회로의 경우에서는 줄어들었고 대부분의 회로에서 필요한 패턴의 수는 증가하였다.

다음 두 개의 표에서는 가중치를 9개의 단계로 라운딩했을 때의 결과를 보여준다.

먼저 표 6은 k값이 4800일 때의 결과이다. 예상했던 바와 같이 몇몇 회로에서는 라운딩하지 않았을 때와 비교해서 필요한 가중치 집합의 수와 패턴의 수가 늘

표 4. 라운딩을 하지 않았을 때의 고장 시뮬레이션 결과 (k = 4800)
Table 4. The fault simulation results when the weights aren't rounded (k = 4800).

Circuits	Primary Input	Flip-Flop	Number of Paths	Testable Paths	1st		2nd		3rd	
					Fault Coverage	# of Patterns	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns
s208	11	8	290	80	0.963	22752	1.000	22784		
s298	3	14	462	116	1.000	2624				
s344	9	15	710	253	1.000	7360				
s349	9	15	730	253	1.000	7360				
s382	3	21	800	154	1.000	992				
s386	7	6	414	145	1.000	4800				
s400	3	21	896	150	1.000	2112				
s420	19	16	738	208	0.918	14848	0.955	22848	1.000	22880
s444	3	21	1070	148	0.993	5760	1.000	5792		
s510	19	6	738	164	1.000	4608				
s526	3	21	820	141	0.986	6848	1.000	6880		
s641	35	19	3488	1027	0.979	28576	0.999	35392	1.000	35424
s713	35	19	5000	210	0.990	13056	1.000	13152		
s820	18	5	984	341	0.985	20000	1.000	20352		
s832	18	5	1012	342	0.982	20000	1.000	20768		
s953	16	29	2312		0.964	34656	0.994	51680	0.998	58112
s1488	8	6	1924	470	0.979	17632	0.994	23072	1.000	23584
s1494	8	6	1952	473	0.981	17632	0.994	23520	1.000	24160
s13207	31	669	5000	635	0.987	9344	1.000	10816		
s15850	14	597	5000	1868	0.991	17952	1.000	19744		
s35932	35	1728	5000	3711	1.000	9088				
s38417	28	1636	5000	1636	0.990	40064	1.000	44992		
s38584	12	1452	5000	62	1.000	96				

어났으나 s536, s713, s1488과 s1494와 같은 회로에서는 도리어 필요한 가중치 집합의 수와 패턴의 수가 줄어들었다. 이는 가중 무작위 패턴의 무작위성 (randomness) 때문이므로 그리 큰 의미를 갖지는 않는다. 마지막으로 표 7은 k값이 9600일 때의 결과를 나타내고 있다. 라운딩을 했을 때와 마찬가지로 몇몇 회로에서 필요한 가중치 집합의 수가 줄어들었고 100%의 고장 검출율을 얻는데 필요한 테스트 패턴의 수는 늘어났다.

위에서 보여주는 결과는 표 6와 표 7에서 보여주는 결과는 가중치를 라운딩하더라 하더라도 s420을 제외한 모든 회로에서 3개 이내의 가중치 집합으로 모든 테스트 가능한 경로에 대해서 100%의 고장 검출율을 얻을 수 있음을 보여준다.

V. 결 론

본 논문에서는 경로 지연 고장 가중 무작위 테스트를 위한 새로운 가중치 생성 알고리듬을 제시하였다. 먼저 주어진 회로에 대하여 결정론적 테스트 패턴을 구한 뒤 초기화 패턴과 두 번째 패턴을 따로 분리하여 두 개의 가중치 집합을 생성한 후, 이 두 가중치 집합을 이용하여 초기화 패턴과 두 번째 패턴을 독립적으로 생성하는 방식을 사용하였다. 이 알고리듬을 사용하여 ISCAS 89 회로에 대하여 고장 시뮬레이션을 수행한 결과는 본 논문에서 제시한 방법이 효율적으로 적용되고 있음을 보여준다. 앞으로 더욱 연구가 필요한 방향은 경로 지연 고장 테스트를 위하여 결정론적 테스트 패턴으로 가중치를 구할 때 입력쌍들간의 충돌을 줄여서 좀 더 효율적인 가중치가 계산될 수 있도록 패턴을 분할하는 방법이다.

표 5. 라운딩을 하지 않았을 때의 고장 시뮬레이션 결과 (k = 9600)
Table 5. The fault simulation results when the weights aren't rounded (k = 9600).

Circuits	1st		2nd		3rd	
	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns
s208	0.963	27552	1.000	27584		
s298	1.000	2624				
s344	1.000	7360				
s349	1.000	7360				
s382	1.000	992				
s386	1.000	4800				
s400	1.000	2112				
s420	0.918	19648	0.995	30272	1.000	30304
s444	1.000	7360				
s510	1.000	4608				
s526	1.000	10464				
s641	0.979	33376	0.998	45792	1.000	45824
s713	1.000	14624				
s820	0.988	32768	1.000	33728		
s832	0.994	46720	1.000	46752		
s953	0.971	50592	0.998	90688	1.000	90720
s1488	0.996	35968	1.000	36000		
s1494	0.996	35968	1.000	36000		
s13207	0.987	14114	1.000	17504		
s15850	0.991	17952	1.000	19744		
s35932	1.000	9088				
s38417	0.991	64448	1.000	68064		
s38584	1.000	96				

표 6. 라운딩을 했을 때의 고장 시뮬레이션 결과 (k = 4800)
Table 6. The fault simulation results when the weights are rounded (k = 4800).

Circuits	1st		2nd		3rd	
	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns
s208	0.775	12384	0.988	26816	1.000	26848
s298	1.000	2624				
s344	1.000	13216				
s349	1.000	13216				
s382	1.000	2752				
s386	1.000	11648				
s400	1.000	2752				
s420	0.846	30368	0.875	45056	0.952	57312
s444	0.993	6976	1.000	7008		
s510	0.982	7936	1.000	7968		
s526	1.000	2240				
s641	0.967	28960	0.993	39392	1.000	39456
s713	1.000	13152				
s820	0.962	15360	0.997	21184	1.000	21216
s832	0.959	17024	0.997	23424	1.000	23456
s953	0.970	46144	0.996	66368	1.000	67424
s1488	0.983	17632	1.000	23712		
s1494	0.985	15776	1.000	17760		
s13207	0.987	9344	1.000	10816		
s15850	0.996	34112	1.000	35840		
s35932	1.000	6432				
s38417	0.991	24704	1.000	25216		
s38584	1.000	96				

표 7. 라운딩을 했을 때의 고장 시뮬레이션 (k = 9600)
Table 7. The fault simulation results when the weights are rounded (k = 9600).

Circuits	1st		2nd		3rd	
	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns	Fault Coverage	# of Patterns
s208	0.950	32000	0.988	41824	1.000	41856
s298	1.000	2624				
s344	1.000	13216				
s349	1.000	13216				
s382	1.000	2752				
s386	1.000	11648				
s400	1.000	2752				
s420	0.880	63264	0.913	88864	0.981	102464
s444	1.000	7328				
s510	1.000	10592				
s526	1.000	2240				
s641	0.967	33760	0.995	63168	1.000	63200
s713	1.000	13152				
s820	0.988	46720	1.000	55424		
s832	0.988	46720	1.000	49216		
s953	0.979	84672	0.994	94624	1.000	95488
s1488	0.996	34496	1.000	34528		
s1494	0.985	20576	1.000	22912		
s13207	0.987	14144	1.000	19552		
s15850	0.997	46112	1.000	46624		
s35932	1.000	6432				
s38417	0.993	56160	1.000	57056		
s38584	1.000	96				

참 고 문 헌

[1] A. Vuksicand and K. Fuchs, "A New BIST Approach for Delay Fault Testing", *Proc. of VLSI Test Symposium*, pp. 284-288, 1994.

[2] B. Wurth and K. Fuchs, "A BIST Approach to Dekay Fault Testing with Reduced Test Length", *Proc. of Euro. Design & Tet Conf.*, pp. 418-423, March 1994.

[3] P. Girard, C. Landrault, V. Moreda and S. Pravossoudovitch, "BIST Test Pattern Generation for Delay Testing", *Electronic Letters*, pp. 1429-1431, 1997.

[4] J. A. Waicukauski and E. Lindbloom, "A Method for Generating Weighted Random Test Patterns" *IBM, J. Res. Dev.*, 33(2), pp 149-161, March 1989.

[5] J. A. Waicukauski and E. Lindbloom, "Fault Detection Effectiveness of Weighted Random Patterns", *Proc. Int. Test Conf.*, pp. 236-244, 1988.

[6] P. H. Bardell, W. H. McAnney and J. Savir, "BUILT-IN TEST for VLSI : Pseudo-random Techniques", *Wiley Interscience*, 1987.

[7] R. Kapur, S. Patil, T. J. Snethen and T. W. Wiliams, "Design of Efficient eighted Random

Pattern Generation System", *Proc. Int. Test Conf.*, pp. 491-500, 1994.

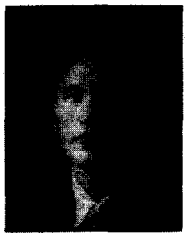
[8] I. Pomeranz and S. M. Reddy, "3-weight Pseude Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuit", *IEEE Trans. Computer-Aided Design*, 12(7), pp. 1050-1058, July 1993.

[9] M. Bershteyn, "Calculation of Multiple Sets of Weights for Weighted Random Testing", *Proc. Int. Test Conf.*, pp. 1031-1040, 1993.

[10] 허용민 & 임인칠, "조합 논리 회로의 경로 지연 고장 검출을 위한 가중화 임의 패턴 테스트 기법", *전자공학회논문집*, 제32권 A편 12호, pp. 229-240, 1995

[11] W. Wang and S. K. Gupta, "Weighted Random Robust Path Delay Testing of Synthesized Multilevel Circuits", *Proc. of VLSI Test Symposium(VTS)*, pp. 291-297, 1994.

저 자 소 개



許 潤(正會員)

1999년 연세대학교 전기공학과 졸업. 1999년~현재 연세대학교 전기·컴퓨터공학과 석사과정. 관심분야는 BIST, 지연 고장 테스팅, 메모리 테스팅

姜 成 昊(正會員)

1986년 서울대 제어계측공학과 졸업. The University of Texas at Austin, Electrical and Computer Engineering (M.S. 1988). The University of Texas at Austin, Electrical and Computer Engineering (Ph.D. 1992). 1989~1992 : Schlumberger Inc. Research Scientist. 1992~1994 : Motorola Inc. Research Scientist. 1994~1999 : 연세대학교 공과대학 기계전자공학부 조교수. 1999~현재 : 연세대학교 공과대학 기계전자공학부 부교수. 관심분야는 VLSI, 테스팅, DFT, CAD