스캔 환경에서 간접 유추 알고리즘을 이용한 경로 지연 고장 검사 입력 생성기

김 원 기[†] · 김 명 균[††] · 강 성 호[†††]

요 약

회로가 복잡해지고, 고속화되면서 회로의 동작에 대한 검사 뿐 아니라, 회로가 원하는 시간 내에 동작함을 보장하는 지연 검사의 중요성이 점점 커지고 있다. 본 논문에서는 주사환경을 사용하는 순차회로에서의 경로 지연 고장을 위한 테스트 패턴 생성 과정을 효율적으로 수행할 수 있도록 빠른 시간에 간접 유추를 수행할 수 있는 알고리즘을 제안한다 구조적으로 발생 가능한 정적 학습 과정은 테스트 패턴 생성 과정 중의 선행 처리 단계에서 각각의 게이트에 정적 학습이 발생할 수 있는 경우를 분석하여 그 정보를 각각의 게이트에 대해 저장하고 있다가 알고리즘을 이용한 테스트 패턴 생성 과정 중 조건에 만족하는 경우에 유추될 수 있는 값을 바로 할당하게 된다. 본 논문에서는 이를 지연고장 검출에 맞도록 수정하여 이용하였다.

회로 내에 몇몇 주입력에서 나온 신호선을 모두 포괄하는 분할지점이 존재하면, 이 지점을 지나는 경로들 중에 그 이전, 혹은 이후의 경로가 동일한 경로들은 분할지점에 의해 분할된 입력의 부분들이 같은 입력값을 필요로 함을 예상할 수 있다. 본 논문에서는 경로 지연 고장 검출에서 유용하게 사용될 수 있는 이러한 회로분할을 사용하여 보다 효율적으로 테스트 입력을 생성하였다.

마지막으로, 이 두 가지 알고리즘을 적용한 효율적인 경로 지연 고장 테스트 입력 생성기를 개발하였으며, 알고리즘의 효용성을 실험을 통하여 입증하였다.

Delay Fault Test Pattern Generator Using Indirect Implication Algorithms in Scan Environment

Won-Gi Kim[†] · Myoung-Gyun Kim[††] · Sungho-Kang[†††]

ABSTRACT

The more complex and large digital circuits become, the more important delay test becomes which guarantees that circuits operate in time. In this paper, the proposed algorithm is developed, which enable the fast indirect implication for efficient test pattern generation in sequential circuits of standard scan environment.

Static learning algorithm enables application of a new implication value using contrapositive proposition. The static learning procedure found structurally, analyzes the gate structure in the preprocessing phase and store the information of learning occurrence so that it can be used in the test pattern generation procedure if it satisfies the implication condition.

※ 이 논문은 1997년 학술진흥재단의 학술연구조성비에 의하여 지원되었습니다.
† 준 회 원 : 삼성전기
†† 준 회 원 : 연세대학교 대학원 전기 · 컴퓨터공학과
††† 정 회 원 : 연세대학교 전기공학과 교수
논문접수 : 1998년 12월 3일, 심사완료 : 1999년 4월 13일

If there exists a signal line which include all paths from some particular primary inputs, it is a partitioning point. If paths passing that point have the same partial path from primary input to the signal or from the signal to primary output, they will need the same primary input values which separated by the partitioning point. In this paper test pattern generation can be more effective by using this partitioning technique.

Finally, an efficient delay fault test pattern generator using indirect implication is developed and the effectiveness of these algorithms is demonstrated by experiments.

1. 서 론

오늘날 VLSI 칩 집적도 증가에 따라 테스트의 중요성이 강조되고 있으며, 우수한 반도체 칩 개발의 필요성이 급속히 증가하고 있어, 회로의 기능적 특성뿐만 아니라 신호 전달 지연 시간의 검증이 고속 동작을 위한 회로 설계에서 우선적으로 고려해야 하는 사항이 되었다. 양질의 칩을 얻기 위하여 기존의 고착 고장에 대한 테스트 외에도 지연고장 테스트(delay fault test)가 수행되고 있다. 지연고장 테스트[1]의 목적은 생산된 회로가 주어진 동작 클럭레이트(functional clock rate)에서 정상적으로 구동되는지를 확인하는 것이다. 지연 고장 테스트는 신호의 변화가 회로를 통하여 주어진 시간 내에 통과하느냐를 결정하기 때문에 동적 논리(dynamic logic)를 다루는 경우를 제외하고는 적어도 2개의 벡터를 필요로 한다. 따라서 지연고장 테스트는 설계된 칩 사용자가 요구하는 동작 주파수의 범위에서 작동하는지를 검사할 수 있다.

앞서 말한 대로, 지연 고장 테스트는 적어도 두 벡터를 필요로 하며, 일반적으로 첫 번째 벡터를 초기값, 두 번째를 최종값이라고 한다. 지연고장 테스트를 수행하기 위한 가장 극단적인 방법(brute force method)은 모든 가능한 입력 쌍을 적용시켜 보는 것이다. 한 번의 테스트를 위해 두 입력 벡터를 필요로 하므로, 만약 n개의 입력이 존재한다면 적용해야 할 모든 가능한 입력쌍은 개가 된다. 즉 이 방법을 쓸 경우 입력의 수가 증가함에 따라 이를 해결하기 위해서는 입력 쌍의 수가 지수 함수적으로 증가하므로 실시간 내에 테스트하기가 어렵다. 이를 해결하기 위해서는 고장 모델을 사용해야 하고 모델링된 고장을 모두 포괄할 수 있는 방법을 개발해야 한다. 이러한 관점에서 볼 때 지연고장(delay fault)은 회로 내 신호의 진행 지연이 모델링된(또는 허용된) 지연 이상으로 증가하게 되는 제조과정에서 생긴 결함을 모델링한 것으로 정의할 수 있다. 또 지연 고장 테스트는 지연고장을 검출하거나 위치를 찾아내기 위해 공급되어야 할 입력을 결정하는 과정이다. 회로의 지연을 모델링할 때는 소자의 입력과 출력에 관성(inertial delay)이 있다고 가정한다. 그러나 실제 지연 모델은 일정 범위 내에서 규정되는 상승 지연과 하강 지연, 그리고 관성지연으로 모델링된다. 고착 고장과는 달리 지연 고장은 시스템의 정상 상태에서의 논리동작에는 영향을 미치지 않지만 전체 시스템의 성능을 저하시킨다. 회로내의 모든 경로가 주어진 시간 명세보다 신호를 일찍 전달한다면 지연고장은 없게 된다.

순수한 순차 회로에서는 조합 회로에서 보다 입력을 생성하기가 매우 까다롭다. 조합 회로에서와는 달리 순차 회로에서의 출력은 입력뿐 아니라 그 회로의 초기상태에 의해서도 영향을 받는다. 또한, 순차 회로는 각 기억소자의 초기상태를 파악하기가 쉽지 않고, 첫 번째 입력에 의해 상태가 어떻게 변하는지 추적하기가 힘들기 때문에 지연 고장을 테스트하기 위한 입력을 생성하기가 어렵다. 순차 회로에서 고착 고장을 검사하기 위해 플립플롭을 직렬로 연결하여 각 플립플롭의 초기값을 외부에서 입력할 수 있게 함으로써 조절 용이도(controllability)와 관측 용이도(observability)를 높인 스캔 플립플롭을 사용하여 효율적으로 테스트 패턴을 생성해 낼 수 있으나, 경로 지연 고장은 테스트를 수행하기 위해 연속적으로 두 패턴을 필요로 하므로 스캔 플립플롭의 사용이 테스트 패턴 생성 과정을 크게 향상시키지는 못한다.

고착 고장을 위한 테스트 패턴 생성에 관한 연구는 정적(static)/동적(dynamic)/순환 학습(recursive learning)과 같은 조사 영역(search space)을 줄이고자 하는 것과 다중 후방 추적(multiple backtrack), 조절 용이도, 관측 용이도 측정과 같은 발전적 지도법(heuristic)에 관한 것들이 있다. 조사 영역을 줄이고자 하는 것은 회로 내에 이미 정해진 값들을 이용하여 새로운 값

을 할당하는 간접적인 유추(implication)라 할 수 있으며, 이 과정들은 테스트 패턴 생성시 영향을 끼치게 된다.

경로 지연 고장을 검사하기 위해 생성할 수 있는 검사 입력의 종류에는 여러 가지가 있으며, 경로 지연 고장 테스트 패턴 생성 과정은 각각의 종류에 대해 경로 지연 고장 영향을 전파시킬 수 있는 경로외(off-path)입력들을 구하면 된다. 따라서 경로 지연 고장 테스트 패턴 생성기는 고착고장과는 달리 고장의 영향을 주출력까지 전파시키는 과정이 필요 없으며, 주어진 경로에 대한 초기 목적(initial objective)을 만족시키는 과정만을 수행하면 된다. 이것은 목적들을 주입력 방향으로 만족시킬 수 있는 간접 유추 알고리즘들이 고착 고장에 비해 경로 지연 고장에서 더 효율적으로 적용될 수 있음을 나타낸다.

본 논문에서는 스캔 환경을 사용하는 순차회로에서의 경로 지연 고장을 위한 테스트 패턴 생성 과정을 효율적으로 수행할 수 있도록 빠른 시간에 간접 유추를 수행할 수 있는 알고리즘을 제안한다. 이 알고리즘은 다양한 논리값으로 구성되는 경로 지연 고장 테스트 패턴 생성기에도 동일한 선행 처리 시간을 사용하여 정적 학습에 필요한 회로의 구조를 검출해 낼 수 있다. 이러한 방법들을 사용하여 스캔 플립플롭에서도 시간 프레임을 역으로 추적하여 표준 스캔 환경의 회로에서도 효율적으로 적용될 수 있다.

2. 이론적 배경

지연 고장 모델은 크게 게이트(gate) 지연 고장 모델[2]과 경로(path) 지연 고장 모델로 나눌 수 있다. 게이트 지연 고장 모델은 신호의 전달 지연이 게이트의 입력이나 출력에서 발생하여 신호 전달 시간이 특정 시간 범위를 벗어나게 된다고 모델링한 것이다. 이에 반해 경로 지연 고장 모델은 검사되고 있는 회로내의 전체 경로 상에서 누적된 작은 지연들로 인해서 회로가 주어진 동작 주파수 안에 정상 동작을 할 수 없다고 가정한다. 게이트 지연 고장 모델은 고착 고장과 비슷하게 고장의 수를 게이트의 수와 입출력의 수에 따라 정량적으로 나타낼 수 있지만 경로 지연 고장 모델은 회로가 커지고 복잡해질수록 경로의 수가 기하급수적으로 늘어나 모든 경로에 대한 검사 입력을 생성하기가 불가능하다. 이러한 단점에도 불구하고 국소화된 지연 고장만을 검사할 수 있는 게이트 지연 고장 모델에 비해 국소화된 지연 고장뿐만 아니라 회로에 널리 분포하는 지연 고장도 검사할 수 있는 장점을 가진 경로 지연 고장 모델을 본 논문에서 사용하였다.

경로 지연 고장에 대한 테스트는 크게 무해저드 경성(hazard free robust : HFR) 테스트[4], 경성(robust : ROB) 테스트, 연성(strong non robust : SNR) 테스트[5]의 세 종류로 나눌 수 있다.

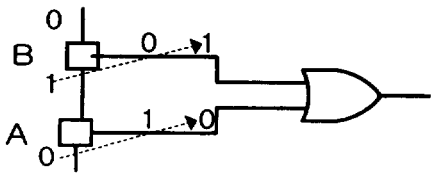
무해저드 경성 테스트는 경로상의 신호들에 동적 해저드(dynamic hazard)가 없고 회로의 다른 부분에서의 지연과 무관하게 경로상의 지연 고장을 검출할 수 있는 두 패턴의 테스트이다. 경성 테스트는 지연 고장 진단에 적합하고 시험 중인 경로가 초과의 지연을 가지면 테스트가 실패하고 또 테스트가 실패하면 시험 중인 경로가 초과 지연을 갖는다. 경성 테스트는 회로의 다른 부분에서의 지연과 무관하게 경로상의 지연 고장을 검출할 수 있는 두 패턴의 테스트로 경로에 특정 전이를 유발시켜야 하고 경로상의 모든 신호들은 특정 전이가 도달하기 전까지 최종값을 알 수 없어야 한다. 경성 테스트는 시험 중인 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 검출만이라면 경성 테스트를 구하는 것이 무해저드 경성 테스트를 구하는 것보다 유리하다. 이는 경성 테스트를 구하는 것이 더 쉽기 때문이다. 경성 테스트의 요건을 모두 갖추지 못한 테스트를 연성 테스트라고 부른다. 이는 2개로 구분될 수 있다. 먼저 경성 테스트를 생성하는 접근법으로부터 나온 연성 테스트를 강연성(strong non-robust) 테스트(SNR), 또는 근사 경성(approximate robust) 테스트라고 부른다. 강연성 테스트(근사 경성 테스트)는 모든 요건이 경성 테스트와 비슷하지만 경성 테스트는 경로외 입력에 정적 해저드(static hazard)가 없어야 하는 경우에도, 강연성 테스트에서는 정적 해저드가 있어도 무방한 두 패턴의 테스트이다. Schulz[5]는 두 번째 사이클의 경우에 경성 테스트와 요구되어지는 값이 같고 첫 번째 사이클의 경우 요구되어지는 것이 경로의 시작부분의 초기값으로만 결정되는 방법으로 연성 테스트를 구하였는데 이를 약연성(weak non-robust) 테스트(WNR)이라고 부른다.[6] 약연성 테스트는 경로상의 전이가 도달하기 전에 회로의

다른 부분이 최종값으로 안정된다고 가정할 수 있다면 경로상의 지연 고장을 검출할 수 있는 두 패턴의 테스트이다.[7] 약연성 테스트를 이용할 경우는 경로를 제외한 회로의 나머지 부분에서는 지연 고장이 없어야 경로상의 지연 고장을 검출할 수 있다.

이러한 테스트의 정의에 따라 요건들을 비교하면, 경성 테스트의 요건 집합이 강연성 테스트의 요건 집합을 포함한다. 이는 경성 테스트를 구하기 위한 요건이 더 많다는 것이며, 따라서 더 구하기 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.

실제로 고착 고장을 위한 테스트에서는 테스트를 돕기 위해 스캔(scan)을 이용하여 순차 회로를 조합 회로로 바꾼다. 지연 고장 테스트에서는 스캔을 쓰는 회로에 대해서도 두 개의 사이클을 가진 순차 회로로 여겨진다. 이를 기존의 조합 회로에 대한 기법을 사용하려면 2개의 벡터를 동시에 저장할 수 있는 확장된 스캔(enhanced scan)을 사용하여야 한다. 그러나 이를 사용하면 너무 많은 면적을 차지하기 때문에 거의 사용하지 않는다. 따라서 표준 스캔을 이용한 회로에 대해서 연구가 활발해지고 있다. 표준 스캔 회로의 경우에는 스캔 이동(scan shifting)기법을 이용하거나 기능적 지정(functional justification)을 이용하는 방법이 있다.

스캔 이동 기법[1]을 사용하는 경우에는 테스트 가능한 많은 경로에 대한 테스트를 생성하지 못할 수 있다. 간단한 예는 (그림 1)과 같다.



(그림 1) 스캔 이동으로 테스트 생성이 불가능한 예

만일 스캔 체인(scan chain)이 A-B로 되어 있고 A의 하강 천이를 고려할 경우 이의 테스트를 위해 B는 해저드 없는 0을 가져야 한다. 하지만 스캔 이동에 의해 B에는 두 번째 시간 프레임에 1을 가지게 된다. 따라서 이 고장을 검출하는 테스트를 구할 수 없다.

스캔 이동의 문제점을 해결하고 정확한 지연 검사를 위해 첫 벡터를 읽어서 이를 사용하고 두 번째 벡터는 회로를 통해 회로의 기능에 따라 정해지며, 두 시간 프레임 동안 테스트에 필요한 요건을 만족시켜야 한다. 이를 기능적 지정이라고 부르며, 이를 이용해야 한다.

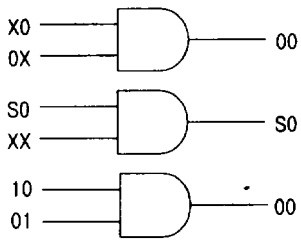
조합 회로의 경우 경로 지연 고장 검사 입력의 생성을 위해 5개의 논리값 {S0, S1, X0, X1, X}를 사용한다. S는 두 시간 프레임동안 안정함을 의미한다. 그러나 표준 스캔 환경에서 첫 번째 벡터는 스캔 플립플롭에 의해 입력할 수 있으나 두 번째 시간 프레임은 앞에서 설명한 함수의 지정에 위해 결정되므로 5개의 논리값만으로 불충분하게 된다. 예를 들어, 플립플롭의 출력에 X0을 지정(justification)하기 위해서는 플립플롭의 입력에 0x라는 값이 필요하다. 이와 같이 표준 스캔 환경의 회로에서 경로 지연 고장 검사를 수행하기 위해 논리값의 확장이 필요하다. 완벽한 경로 지연 고장 검사를 위한 논리값은 51개로 구성되어야 한다는 사실이 [8]에서 보여졌다. 본 논문에서는 간결성과 완벽한 논리값에 의해 제공되는 시도철회(backtrack)동안의 상충을 피할 수 있는 방법을 적절히 고려하고 유추표의 엄청난 복잡도를 피하기 위해 29개의 논리값을 사용한다.[9] 먼저 {0, 1, X} 외에 Y를 추가한다. Y는 2진 논리 소자(binary logic level)에서는 고임피던스(i.e. Z)를 나타내며, 게이트 수준에서는 명시할 수 없는(unspecifiable unknown)값을 나타낸다. 이것은 X가 명시되지 않은(unspecified unknown)값을 나타내는 것과 구별된다. 함수적 지정과 고임피던스를 다루기 위해 적어도 19개의 논리값이 요구된다. ((그림 2)에서 00부터 SZ) 또한 앞서 정의한 논리값을 보충하기 위해 9개의 논리값을 추가한다. ((그림 2)에서 00T부터 XTB) T는 값이 안정적임을 보장하지 못 한다는 것을 나타낸다.

00	01	0Y	0X	10	11	1Y	1X
Y0	Y1	YY	YX	X0	X1	XY	XX
S0	S1	SZ					
00T	0XT	X0T	XT0	11T	1XT	X1T	XT1	XTB
SB							

(그림 2) 29개 논리값

SB는 단지 미지 논리 블록(custom logic block)의 정당화(justification)에서만 사용되며, 유추 과정에서는

적용되지 않는다. (그림 2)의 처음 두 줄은 {0, 1, X, Y}로 만들 수 있는 모든 조합을 나타낸 것이며, 이 논리값으로는 신호들에 대해 해저드의 유무를 알 수 없다. {S0, S1, SZ}은 두 시간 프레임 모두에서 각각 {0, 1, Z}를 나타내는 것이며, 신호에 해저드가 없음을 보장한다. {00T, 0XT, X0T, 11T, 1XT, X1T}는 각각 {00, 0X, X0, 11, 1X, X1}과 비슷하나 신호에 해저드가 반드시 존재함을 나타낸다. 예를 들어, {00, 00T, S0}는 두 시간 프레임에서 모두 00을 나타내며, {00}과 {S0}의 합을 나타낸다. (그림 3)에 00에 대한 논리값 할당의 예가 있다.

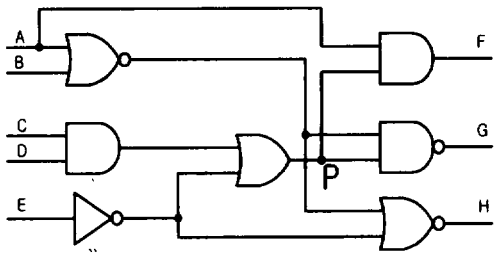


(그림 3) 다양한 논리값 0

{XT0, XT1}은 두 시간 프레임 모두 X를 나타내지만, {S0, S1}이 될 수 없음을 보장하며, {XTB}도 마찬가지로 두 시간 프레임 모두 X를 나타내지만 {S0}과 {S1}이 될 수 없음을 나타낸다. {SB}는 {S0}과 {S1}의 결합이다.

3. 회로 분할 알고리즘

이 절에서는 스캔 플립플롭을 사용하는 순차 회로에서 지연 고장의 검출을 위한 테스트 생성을 효율적으로 수행할 수 있는 회로 분할 알고리즘을 제안한다. 다음과 같은 회로를 보자.



(그림 4) 회로 분할의 예

위의 회로에서 점 P를 기점으로 입력 C, D, E와 나머지 입력 A, B는 분할된다. 즉, P를 지나는 모든 경로는 C, D 또는 E에서부터 P까지의 경로와 입력과 P에서 출력 F나 G까지의 경로와 입력이 어떤 경우라도 서로 모순되지 않을 것이 보장된다. 이와 같은 경우 경로가 분할되었다고 하고, 점 P를 지나는 모든 경로는 이와 같은 속성을 가진다. 따라서 경로 C→P→G에 대한 지연 고장 테스트 패턴의 C→P→F에 대한 지연 고장 테스트 패턴의 C, D, E 입력과 동일하게 된다. 다음의 표를 보자.

〈표 1〉 분할 지점을 지나는 경로에 대한 테스트 입력

경로	테스트 패턴 입력
C→P→F	<S1,XX,01,S1,S1>
C→P→G	<S0,S0,01,S1,S1>
D→P→F	<S1,XX,S1,01,S1>
D→P→G	<S0,S0,S1,01,S1>
E→P→F	<S1,XX,S0,XX,01>
E→P→G	<S0,S0,S0,XX,01>

위 표의 첫 번째 줄과 두 번째 줄을 비교해 보자. 두 경로는 모두 회로의 분할 지점인 P를 지나고 P 이전의 경로가 동일하다. 따라서 P 이전의 경로의 입력을 결정하는 입력 C, D, E의 값은 모두 <01, S1, S1>로 동일하다. 마찬가지로 첫 번째 줄과 세 번째 줄을 비교해 보면, P 이후의 경로가 동일하기 때문에, 그 후의 경로의 입력을 결정하는 입력 A, B의 값이 동일함을 확인할 수 있다. 이처럼 회로 내에 P와 같은 분할 지점이 존재하면, 이 P를 지나는 경로들 중에 P 이전, 혹은 이후의 경로가 동일한 경로들은 분할지점에 의해 분할된 입력의 부분들이 같은 입력값을 필요로 함을 예상할 수 있다. 여기서는 경로 지연 고장 검출에서 유용하게 사용될 수 있는 이러한 회로 분할을 사용하여 보다 효율적인 테스트 입력을 생성하고자 한다. 이를 위한 구체적인 알고리즘은 다음과 같다.

3.1 경로 지연 고장의 회로 분할 분석 과정

회로 내의 어떤 게이트에서 회로가 분할되는지 판정하는 데는 몇몇 기준이 있다. 이러한 기준은 다음과 같이 세 가지로 생각할 수 있다. 첫째로 그 게이트에 들어오는 모든 입력이 주입력에서부터 한 번도 갈라져 나온 적이 없이 그 게이트까지 도착했으면, 이러한 게

이트는 회로를 분할하는 지점이 된다. 둘째로 주입력에서 시작해서 중간에 갈라져 나온 적이 있더라도, 갈라져 나온 모든 신호선이 다시 그 게이트의 입력으로 모인다면, 이러한 게이트는 회로를 분할하는 지점이 된다. 셋째로 갈라져 나온 신호선이 그 게이트로 전부 모이지 않는다고 하더라도, 게이트를 통과한 신호선들과 게이트 이전에 갈라져 나온 신호선들이 서로 만나지 않는다면 그 게이트로 회로를 분할하는 지점이 된다. 위의 세 조건 중 하나만 만족하면 그 게이트에 영향을 미치는 주입력과 그 외의 주입력은 그 게이트를 지나는 경로의 경로외 입력에 대해 서로 모순될 우려가 없으므로, 그 게이트를 중심으로 경로를 나누어 생각할 수 있게 한다. 이러한 조건을 확인하기 위해 다음과 같은 과정을 거친다.

```
for( i = 0 ; i < Num_gate ; i++ )
{
    forward_simulate(i) ;
            // check lines from gate i to POs //
    backward_simulate(i) ;
            // check lines from PIs to gate i //
    flag=forward_simulate_fanout() ;
            //check reconvergent fan-out //
    if(flag)
        flag = 0;
    lse
        add_partition(i) ; // save partition point //
    }
```

(그림 5) 회로 분할 지점 찾기

처음의 forward_simulate(i)에서는 해당 게이트에서 주출력까지 거쳐가는 신호선들을 모두 표시한다. backward_simulate(i)에서는 해당 게이트에서 주입력까지 거꾸러 올라가며, 신호선들이 통과하는 게이트를 표시한다. backward_simulate(i)에서 표시된 게이트 중에서 신호선이 갈라져서 나가는 것들은 스택에 저장되었다가, forward_simulate_fanout()에서 갈라져 나온 신호선이 다시 해당 게이트로 수렴하는지 확인하고(두 번째 조건), 수렴하지 않는다면 이 신호가 처음 forward_simulate(i)에서 표시한 게이트들을 지나는지(세 번째 조건) 확인한다. 만약 갈라져 나오는 신호선이 없으면(첫 번째 조건) 스택에 저장된 것이 없어 forward_simulate_fanout()을 거치지 않고 다음으로 넘어간다. 이런 과정을 거쳐 세 가지 조건중 하나라도 만족하는 게이트는 flag가 '0'이 되어 분할 지점으로서 저장되게 된다.

3.2 경로 지연 고장의 회로 분할 적용

앞의 회로 분할 분석과정에서 저장된 회로의 분할 지점은 테스트 입력 과정에서 고려된다. 테스트하려는 경로가 회로의 분할 지점을 지나면, 일단 이를 처리하는 부분으로 가게되고, 분할 지점을 지나지 않는 경로는 일반적인 테스트 생성 방법에 따라 처리된다. 분할 지점을 지나는 경로는 테스트 입력 생성 전과 후의 두 번에 걸쳐 처리 루틴을 거치게 된다.

```
pre_part_process(the_path)
{
    flag_before = before_search(the_path) ;
                // search saved info. 1 //
    flag_after = after_search(the_path) ;
                // search saved info. 2 //
    if(flag_before && flag_after)
    {
                both_found(the_path) ;
                return() ;
    }
    if(flag_before)
    {
                before_found(the_path) ;
                after_not_found(the_path) ;
    }
    else if(flag_after)
    {
                before_not_found(the_path) ;
                after_found(the_path) ;
    }
    else
    {
                before_not_found(the_path) ;
                after_not_found(the_path) ;
    }
    find_one_path(the_path) ;
    return(0) ;
}
```

(그림 6) 테스트 생성 이전의 회로 분할 처리

before_search(the_path)는 분할 지점이전의 경로에 해당하는 주입력값들이 저장되어 있는지 확인한다. 분할 지점이전의 경로가 같다면, 이에 해당하는 경로외 입력값을 만들어주는 주입력값들도 동일할 것이기 때문에 이미 구해진 주입력값들을 그대로 사용할 수 있다. 마찬가지로 after_search(the_path)도 분할 지점이

후의 경로에 해당하는 주입력값들이 앞선 경로들의 테스트 입력 생성결과로 이미 만들어져 있는지 확인한다. 두 검색 결과, 두 가지 입력값들이 모두 존재하면, 이 경로에 대해서는 따로 입력을 구하지 않고, 이미 구해진 입력값들을 단순히 합쳐줌으로써 테스트 입력 생성이 가능하다. both_found(the_path)루틴에서 이러한 처리를 해주고, find_one_path()를 거치지 않고 다음 경로에 대한 테스트 생성으로 넘어가게 된다. 분할 지점의 전이나 후의 한쪽에서만 이미 만들어진 입력이 발견된 경우, 만들어진 부분의 경로는 생략하고 나머지 부분만 find_one_path(the_path)로 보내어 나머지 테스트 입력을 생성하게 된다. 이러한 처리를 해주는 부분이 각각 before_found(the_path)와 after_found(the_path)이다. 한편 검색 결과, 이 경로가 처음 나타난 경로일 경우, 결과를 저장하여 다음에 같은 경로 부분을 갖은 경로가 이용할 수 있도록 색인을 만들고 저장장소를 확보하는 일은 before_not_found(the_path)와 after_not_found(the_path)에서 처리해 준다.

테스트가 생성된 후에는 위에서 만들어준 저장장소에 생성된 주입력값 중 해당하는 값을 저장한다. 이를 처리하기 위한 루틴은 find_one_test()내에 위치한다. 이 안에는 또한 부분적인 경로에 대해서만 생성한 입력값을 저장된 입력값과 합쳐서 완전한 테스트 입력을 만들어 주는 처리 루틴도 포함된다.

4. 정적 학습 알고리즘

이와 같이 29개의 논리값으로 구성된 경로 지연 고장 검사 테스트 입력 생성 과정에서 쉽고 효율적으로 적용될 수 있는 정적 학습 알고리즘을 제안하고자 한다. SOCRATES에서 제안된 알고리즘은 대우 관계를 이용하여 선택(decision) 과정에서 새로운 유추값의 적용을 가능하게 한다. 정적 학습 과정은 대우 관계를 이용하여 구해진다.

구조적으로 발생 가능한 정적 학습 과정은 테스트 패턴 생성 과정 중의 선행 처리 단계에서 각각의 게이트에 정적 학습이 발생할 수 있는 경우를 분석하여 그 정보를 각각의 게이트에 대해 저장하고 있다가 알고리즘을 이용한 테스트 패턴 생성 과정 중 조건에 만족하는 경우에 유추될 수 있는 값을 바로 할당하게 된다. 고착 고장에서 가능한 정적 학습은 두 시간프레임을 고려한다는 점을 제외하고는 똑같은 형태로 경로 지연 고장에서도 이용할 수 있다. 그러나 고착 고장에서 사용하는 논리값은 0, 1, X 세 값이므로 선행 처리 단계에서 많은 시간과 메모리를 사용하지 않고 회로 구조를 분석 할당할 수 있으나, 경로 지연 고장은 효율적인 테스트 패턴 생성 과정을 위해 많은 논리값을 필요로 하므로 각각의 논리값에 대해 선행 처리 단계에서 하나씩 모두 분석한다는 것은 대단히 큰 시간과 메모리를 요구하게 되므로 효율적이지 못하게 된다. 따라서 본 연구에서는 고착 고장에서와 같은 오버헤드를 가지고 많은 논리값을 갖은 경로 지연 고장에서 정적 학습 구조를 분석하도록 하기 위해서 다음과 같은 알고리즘을 제안한다.

4.1 경로 지연 고장의 정적 학습 분석 과정

경로 지연 고장에서 사용하는 논리값이 매우 많다고 하더라도 이것은 두 시간 프레임을 사용함으로 인해 발생한 것이며, 동일한 시간 프레임만 고려한다면 기본적으로는 0, 1, X만으로 연산 과정이 이루어진다. 그러므로 경로 지연 고장을 위한 정적 학습 알고리즘은 먼저 시간 프레임을 나누어서 정적 학습 구조를 분석한 후 테스트 패턴 생성 과정에서 29개의 논리값에 알맞도록 값을 적용하도록 한다. 따라서 선행 처리 단계에서는 각 노드들에 알맞은 0X, 1X, ZX를 할당한 후에 시뮬레이션을 실행한다. 그 후에 각 노드들의 값을 분석하여 정적 학습이 발생한 노드를 해당 게이트 데이터 스트럭쳐에 저장하게 된다. 정적 학습의 분석 알고리즘은 다음 (그림 7)에 있다.

```
learn()
{
      for every signal ;
      {
         assign_value (i, logic_value)
           // logic_value = 0X or 1X or ZX //
         implication ()
         analyze_result (i)
      }
}
```

(그림 7) 정적 학습 알고리즘

assign_value는 AND, 또는 NOR인 경우 0X를 NAND, 또는 OR인 경우는 1X를 XOR, 또는 XNOR는 0X, 1X 둘 다를 할당한다. 시뮬레이션 후에 구한 값들

은 29개 논리값이므로 이것들을 (그림 8)과 같은 식으로 3개 논리값으로 변환시켜준다.

```
inverter29to3value (value)
{
  if(value==LV_0X || value==LV_01 || value==LV_0Y ||
    value==LV_00 || value==LV_0XT || value==LV_S0 ||
    value==LV_00T)
    logic_value = LOGIC_0 ;
  if(value==LV_1X || value==LV_11 || value==LV_1Y ||
    value==LV_10 || value==LV_S1 || value==LV_1XT ||
    value==LV_11T
    logic_value = LOGIC_1 ;
  else logic_value = X ;
  return(logic_value) ;
}
```

(그림 8) 첫 번째 시간 프레임에 의한 논리값 분류

변환된 3개 논리값을 가지고 분석하여 정적 학습의 관계를 얻어낸다. 분석 알고리즘은 (그림 9)에 있다. (그림 9)에서 AND, 또는 NOR에서 출력이 1인 경우, NAND, 또는 OR에서 출력이 0인 경우, XOR 또는 XNOR의 경우 0, 1 모두에 대해서 정적 학습의 관계를 유도할 수 있으며, 저장 시에는 유추된 반대의 값이 할당되어 쓰이게 되므로 반대값이 저장된다.

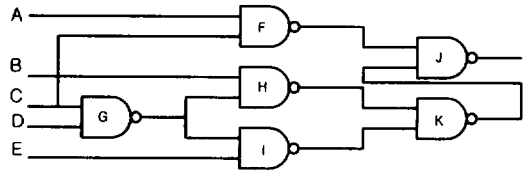
```
analyze_result ( i )
{
   vi = value of signal i
   for every signal j with value of j ≠ X and
       with j ≠ i
   {
     vj = value of signal j
     g = gate with output signal j
     tg = type of gate g
     if ( vj = 1 and ( tg = AND or tg = NOR ) or
        vj = 0 and ( tg = OR or tg = NAND ) or
        tg = XOR or tg = XNOR )
     {
        if ( check_path ( j, i ) = 0 ) then
        {
            save_learn ( j = ~vj → i = ~vi )
        }
     }
   }
}
```

(그림 9) 분석 알고리즘

4.2 경로 지연 고장의 정적 학습 적용

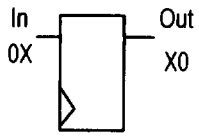
선행 처리 단계에서 구해진 정보들은 고착 고장에 대한 값들이 된다. 따라서 테스트 패턴 생성 과정 중에 논리값 할당을 필요로 할 때에는 세 개의 논리값들을 29 논리값에 알맞도록 변환시킨 후에 할당해 주어야 한다. 테스트 패턴 생성 과정 중 초기 목적값을 만족시키는 과정은 시간 프레임 별로 각각 진행되므로 정적 학습 값을 진행되는 각각의 시간 프레임에 맞도록 논리값을 분리하여 할당하게 하여야 한다.

(그림 10)에서 정적 학습이 적용 가능한 예가 있다. 저장된 정적 학습 정보는 두 가지 경우에 사용되는데 첫째, 지연 고장 검출 테스트 입력 생성 과정 중 경로상의 값들과 반드시 만족되어야 하는 경로외 입력에 대한 값들로 인한 필수적인 값일 때와 둘째는 값들을 할당하는 과정에서 값들이 변화해서 계속 변화될 수 있는 선택적인 값인 경우의 두 가지 경우가 있다. (그림 10)의 예에서 A, F, J의 경로에서 주입력의 상승 전이를 고려할 때 경로외 입력 조건에 따라 K에 S1이 할당되어야 할 경우에 바로 G에 S1을 할당할 수가 있다.



(그림 10) 정적 학습이 적용되는 지연 고장 테스트의 예

앞에서 보인 정적 학습 방법은 고착 고장에서 고려한 것과 동일한 형태로 발생할 수 있는 구조를 시간 프레임이 같은 것들에 대한 논리값을 할당하는 경우이다. 그러나 경로 지연 고장 모델에서는 두 개의 시간 프레임이 사용되므로 스캔 플립플롭을 통해서 서로 다른 시간 프레임에서도 정적 학습 구조를 검출할 수가 있다. (그림 11)에서처럼 스캔 플립플롭의 출력단 쪽에서 두 번째 시간 프레임의 논리값은 스캔 플립플롭의 입력단의 첫 번째 시간 프레임의 논리값을 유추할 수



(그림 11) 스캔 플립플롭의 입출력 할당

있게 된다. 스캔 플립플롭에 의한 정적 학습을 위한 방법도 위의 방법과 마찬가지로 0X, 1X, ZX 세 논리값의 할당을 고려하지만 29개의 논리값들은 두 번째 시간 프레임을 고려하여 분류하게 된다.

테스트 패턴 생성 과정에서의 적용은 스캔 플립플롭을 통해서 시간을 역추적하는 것이므로 두 번째 시간 프레임을 고려하여 첫 번째 시간 프레임만으로 적용될 것이다.

5. 실험 결과

구현된 지연 고장 검사 입력 생성기의 성능을 실험해 보았다. 지연 고장의 검사 입력 생성기는 29개 논리값을 사용한 기본적인 생성기에 정적 학습 알고리즘과 회로 분할 알고리즘을 적용한 것을 사용하여 실험하였다. 지연 고장 검사 입력 생성기에 대한 실험 결과는 기존의 패턴 자동 생성기 중 우수한 것으로 알려진 DSTEED 알고리즘을 구현한 지연 고장 검사 입력 생성기와 비교하였다.

실험은 ISCAS 85와 89 벤치마크 회로를 사용하여 실시하였다. 테스트 입력을 생성하는 경로는 세그먼트가 3개 이상인 것만으로 한정하여 생성하였으며, 경로의 수가 1000개를 넘은 회로에 대해서는 무작위로 1000개만을 선택하였다. 벤치마크 회로 중 정적 학습이 적용된 회로는 조합 회로 중 C880, C1908, C2670, C3540, C5315, C7552의 여섯 개, 순차 회로 중 S713, S9234의 두 개였다. 또한 회로 분할이 사용된 벤치마크 회로는 C880, C2670, C7552의 세 개였으며, 순차 회로 중에는 사용된 경우가 없었다. 실험 결과는 다음 쪽의 <표 2>와 같다.

사용된 테스트 입력 자동 생성기는 C 코드를 사용하여 개발된 경로 지연 테스트 입력 자동 생성기에 정적 학습 알고리즘과 회로 분할을 적용하여 개조한 후, SUN Ultra Sparc machine Workstation에서 컴파일하여 만들었으며, 실행은 동종의 시스템에서 실시하였다. 그 결과를 같은 환경에서 실행한 DSTEED 테스트 입력 자동 생성기의 결과와 비교한 것이다. DSTEED는 연성 테스트만을 수행하므로, 공정한 비교를 위해 간접 유추를 사용한 테스트 입력 생성기에서도 연성 테스트 입력만을 생성하였다. 구축된 간접 유추를 사용하는 테스트 입력 생성기에서 경성 테스트 입력을 생성한 결과는 <표 3>에 따로 나타내었다.

결과를 보면 정적 학습과 회로 분할을 사용함으로써 Backtrack의 횟수를 줄이고, 시간을 절약하는 효과가 있음을 알 수 있다. 대부분의 회로에서 1/10 정도로 테스트 시간이 크게 줄어들었으며, 특히 C880에서는 테스트 시간이 1% 이내로 줄어드는 획기적인 성능 향상을 보였다.

〈표 2〉 간접 유추를 이용한 경로 지연 고장 약연성 테스트 입력 생성과 DSTEED를 이용한 테스트 입력 생성 결과 비교

벤치회로	경로수	정적 학습의 사용 횟수	회로 분할의 사용 횟수	테스트 입력 자동 생성기	고장검출율 (%)	Backtrack 횟수	테스트 입력 생성 시간
C880	234	389	6	DSTEED	61.54	477517	312.70
				간접 유추	81.62	212	1.54
C1908	608	555	0	DSTEED	9.50	920000	1133.50
				간접 유추	24.34	21021	28.67
C2670	1000	118	6	DSTEED	43.40	4654037	7141.20
				간접 유추	43.40	1258187	837.19
C3540	782	534	0	DSTEED	53.60	1132455	2912.00
				간접 유추	51.79	638650	975.54
C5315	1000	252	0	DSTEED	97.40	24420	143.80
				간접 유추	97.40	7696	22.10
C7552	1000	1070	6	DSTEED	80.00	1158363	5215.20
				간접 유추	91.20	328597	355.50
S713	456	177	0	DSTEED	13.80	1462	5.10
				간접 유추	13.82	1132	3.42
S9234	1000	1977	0	DSTEED	56.60	84156	3232.20
				간접 유추	64.20	378138	514.11

〈표 3〉 간접 유추를 이용한 경로 지연 고장 경성 테스트 입력 생성 결과

벤치 회로	경로수	정적 학습의 사용 횟수	회로 분할의 사용 횟수	고장 검출율 (%)	Backtrack의 횟수	테스트 입력 생성 시간
C880	234	264	4	63.25	56	1.47
C1908	608	214	0	16.28	1114015	1063.55
C2670	1000	184	4	31.00	23793	20.71
C3540	782	1166	0	41.18	1016937	1359.27
C5315	1000	709	0	93.20	92983	89.66
C7552	1000	1070	6	91.20	328597	355.55
S713	456	67	0	5.04	171	1.91
S9234	1000	2823	0	46.70	146618	192.92

6. 결 론

이 논문에서는 경로 지연 고장 검출을 위한 테스트 입력 생성을 보다 효율적으로 수행하기 위한 두 가지 유추 알고리즘으로서 정적 학습과 회로 분할 기법을 제안하였다. 이 두 가지 방법은 테스트 입력 생성시 이미 가지고 있는 데이터를 이용하여 중복되는 작업을 한 번만 수행하도록 함으로써 테스트 입력 생성에 소요되는 시간을 줄이고자 하는 것이다. 이를 위한 선행 처리 과정이 필요하고, 실제 입력 생성시에 이를 사용하는 부분도 첨가하였다.

선행 처리 과정에서 소요되는 시간은 테스트 입력 생성 시간에 비해 매우 작으므로 시간의 오버헤드가 문제되지는 않는다. 위의 두 가지 간접 유추 알고리즘에서 가장 큰 문제가 되는 것은 실제로 적용될 수 있는 회로가 몇몇 회로로 한정되고 있다는 것이다. 실험 결과에서 알 수 있듯이 전체 벤치마크 회로 중, 실제 간접 유추 알고리즘이 사용되는 회로는 3~8개에 지나지 않는다. 또한 간접 유추 알고리즘이 적용되는 회로에서도 실제 사용되는 횟수가 많지 않다. 그러나 스캔 환경에 적합한 논리값과 간접 유추를 동시에 사용하여 테스트 입력의 생성 시간이 크게 향상되었음을 확인할 수 있다.

주목할 만한 점은 벤치마크 회로 중, 비교적 크기가 크고 회로가 복잡하여 테스트 생성이 어렵거나 시간이 많이 걸리는 C1908, C2670, C7552 등의 회로의 경우에는 간접 유추 알고리즘이 많이 사용된다는 점이다. 회로가 복잡해지더라도 선행 처리 과정에서 걸리는 시간은 게이트 수에 비례하여 증가하는 정도이고, 복잡한 회로일수록 간접 유추 알고리즘의 사용 가능성이 높아지며, 실제 사용하게 되면 그 효과가 크게 나타나게 된다. 따라서 실제의 크고 복잡한 회로에서 정적 학습과 회로 분할 기법이라는 간접 유추 알고리즘을 사용하면 벤치마크 회로를 사용한 실험에서 보다 더욱 향상된 결과를 얻을 수 있을 것으로 예상된다.

참 고 문 헌

[1] V. Iyengar, B. Rosen, and I. Spillinger, "Delay Test Generation Algebra and Algorithms," Proc. of International Test Conference, pp.867-876, 1988.

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

[3] S. Reddy, C. Lin and Patil, "An Automatic Test Pattern Generation for the Detection of Path Delay Fault," Proc. ICCAD, pp.284-287, 1987.

[4] C. Lin and S. Reddy, "On Delay Fault Testing in Logic Circuits," Trans. on Computer, Step. 1987.

[5] M. Schulz et al., "Advanced Automatic Test Pattern Generation Technique for Path Delay Fault," Proc. of FTCS, 1989.

[6] B. Underwood, S. Kang and O. Law, "A Path-Delay Test Generator for Large VLSI Circuits," Proc. of ICVC, 1993.

[7] S. Kang, B. Underwood and W. Law, "Path Delay Fault Simulation for a Standard Scan Design Methodology," Proc. ICCD, pp.359-362, 1994.

[8] K. Fuchs, H. Wittmann and K. Antreich, "Fast Test Pattern Generation for All Path Delay Faults Considering Various Test Case," Proc. of

European Test Conference, pp.89-98, 1993.
[9] Bill Underwood, Wai-On Law, Sungho Kang, Haluk Konuk "Fastpath : A Path-Delay Test Generator for Standard Scan Design." ITC, pp. 154-163, 1994.

김 원 기

e-mail : wgkim@dopey.yonsei.ac.kr
1997년 2월 연세대학교 전기공학과 졸업(학사)
1997년 3월~1999년 2월 연세대학교 전기공학과 대학원 졸업(공학석사)
1999년 2월 현재 삼성전기 근무
관심분야 : VLSI CAD, 테스팅, VLSI 설계

김 명 균

e-mail : gundam2000@sleepy.yonsei.ac.kr
1999년 2월 연세대학교 전기공학과 졸업(학사)
1999년 3월~현재 연세대학교 전기·컴퓨터 공학과 대학원 석사과정(석사 1학기)
관심분야 : VLSI CAD, 테스팅, VLSI 설계

강 성 호

e-mail : shkang@bubble.yonsei.ac.kr
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 : VLSI CAD, 테스팅, 설계검증, VLSI 설계