전수받은 값을 이용한 조합회로에 대한 검사 패턴 발생

송 상 훈†

요 약

본 논문은 효과적인 검사 패턴 발생 방법을 제안한다. 기존의 검사 패턴 발생 방법들은 고장 F_{i+1}에 대한 검사 패턴 발생을 고장 F_1, F_2, ..., F_i들에 대한 검사 패턴 발생시 행한 계산과는 독립적으로 행하게 된다. 제안된 방법에서는 고장 F_i에 대한 검사벡터를 전수받아 고장 F_{i+1}에 대한 검사벡터를 발생한다. 전수받은 값을 점차로 바꾸어 나가면서 새로운 검사벡터가 발생된다. 전수받은 값은 부분적으로 고장 신호를 활성화하고 이 고장 신호를 전파시키기도 한다. 보통 이들은 다음 탐색과정에서 결정단계의 수와 회귀의 수를 감소 시킨다. 잘 알려진 벤치마크 회로에 대한 실험 결과는 낮은 회귀한계에서 매우 효과적임을 보여주고, 다른 알고리즘과 병합시키면 임의의 회귀한계에서도 매우 효과적임을 보여준다.

Test Pattern Generation for Combinational Circuits using Inherited Values

Sang Hoon Song†

ABSTRACT

This paper proposes an efficient method for test pattern generation. Current test pattern generation systems generate a test vector for fault F_{i+1} independently of the computation previously done for faults F_1, F_2, ..., F_i. The proposed algorithm generates a test vector for fault F_{i+1} by inheriting the test vector for fault F_i. A new test vector is generated from inherited values by gradually changing the inherited values. The inherited values may partially activate a fault and propagate the fault signal. Normally, this reduces the number of decision steps and backtracks in the second search. Experimental results for well-known benchmark circuits show that the proposed algorithm is very efficient with small backtrack limit; in combination with other algorithms, it is very efficient for arbitrary backtrack limits.

1. 서 론

VLSI 기술이 급속도로 발전함에 따라, 집적회로의 집적도가 급격하게 증가해 왔다. 급격히 증가된 집적회로의 집적도에 따른 회로의 고장 검사도(testability)가 떨어짐으로써, 검사 패턴 발생(test pattern generation)이 더욱 어려워 졌다. 검사 패턴 발생이란 디지탈 시스템을 검사하기 위해 필요한 입력들의 집합을 찾는 것이다. 이런 입력들의 집합을 검사 집합이라 하고, 이 검사 집합의 질은 집합의 크기와 고장 검출율(fault coverage)로 평가할 수 있다. 고장 검출율은 검사 집합에 의해 검출되는 고장들의 수와 검사 대상인 디지

※본 논문은 1995년도 대양학술 연구비 지원에 의해 작성됨.
† 정 회 원:세종대학교 전산학과 조교수
논문접수:1996년 11월 20일, 심사완료:1997년 1월 20일

탈 시스템내의 총 고장 수와의 비율을 말한다.

일반적으로 단일 고장 모델(single fault model)에 대한 회귀 탐색 방법(backtrack search method)을 사용하고 있는 검사 패턴 발생 알고리즘들은 한 고장에 대한 검사 패턴을 주어진 회귀 한계내에서 찾지 못하면 포기하고 다음 고장에 대한 검사 패턴을 찾는 작업으로 넘어 간다. 검사 패턴을 찾기가 어려워 포기하게 되는 고장들이 많아지면 결국 고장 검출율은 떨어지게 된다. 회귀 한계를 높여서 고장 검출율을 조금 높일 수 있으나, 여전히 포기하게 되는 고장들에 그 만큼 더 많은 시간을 소비하게 되어 효과적이 아니다.

본 연구에서는 아주 낮은 회귀 한계에서 효과적으로 검사를 발생하면서 다른 검사 집합을 찾을 수 있는 새로운 방법 NTPG(new test pattern generation)를 제안하고, 이를 기존의 방법과 병합시킴으로써 전체 고장 검출율을 높이는 것을 보여준다. 기존의 방법들은 주어진 고장에 대한 검사 패턴 발생시 회로의 주입력(primary input)들의 상태를 X 값(undefined state)에서 시작하여 회귀 탐색 과정을 통하여 이들의 값을 정해 나간다[1, 2, 3, 4]. 성공적으로 검사 패턴이 발생된 경우에 탐색 과정에서 정해진 주입력들의 값에 따라서 회로내의 노드(gate)들이 0 또는 1의 상태를 갖게 된다. 대부분의 이들 값은 주어진 고장에 대한 고장 신호를 활성화하고, 이 고장 신호를 출력쪽으로 전달하는 역할을 하게 된다. 이 상태에서 이들의 값을 새로운 결정(decision)에 의하여 변경할 수 있는 값으로 만든다. 주입력의 값을 백트레이스(backtrace) 과정을 거쳐 정해지는 결정에 따라 변경 가능한 값들을 하나씩 변경해 나가면 다른 고장에 대한 검사를 찾아낼 수가 있을 것이다. 기존의 방법에 비하여 고장에 따라서는 더 많은 계산이 필요한 경우도 발생할 수 있지만, 기존의 방법으로 높은 회귀 한계에서도 포기되는 고장들에 대한 검사를 쉽게 찾아낼 수도 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 연구 배경과 기본 개념을 설명하고, 3 장에서는 전수받은 값을 이용하는 검사 발생과 이를 기존의 방법과 병합시키는 것에 대하여 설명한다. 4 장에서는 벤치마크 회로에 대한 실험 결과를 분석하고 5 장에서 본 논문의 결론을 맺는다.

2. 연구 배경

조합회로에 대한 검사 패턴 발생 방법들이 많이 연구되어 왔는데[1, 2, 3, 4, 5, 6, 7], 일반적으로 경로 감응(path sensitizing) 방법을 사용한다[1, 2, 3, 4]. 이들은 고장 신호를 대상 고장(target fault) 지점에서 생성되게 하여 이 신호를 출력쪽으로 전파되도록 주입력의 논리값을 회귀 탐색 과정을 통하여 결정해 나간다. 일반적인 디지탈 논리회로에서 총 주입력의 수가 n이면 탐색 공간의 크기는 2^n이 된다. 탐색 공간을 체계적으로 탐색하는 과정에서 회귀의 수를 줄이기 위해 많은 heuristics들이 제안되어 왔다. 대표적인 알고리즘으로 PODEM[1], FAN[2], 그리고 SOCRATES[3] 들이 있다.

경로 감응 방법에서 대부분의 계산 시간이 백트레이스와 임플리케이션(implication) 과정에 쓰인다. 백트레이스 과정은 대상 고장 지점에 고장 신호를 생성되게 하거나, 이 신호를 출력쪽으로 전파되도록 하기 위한 주입력의 논리값을 선택하는게 목적이다. 한 고장 신호를 출력쪽으로 전파되도록 하기 위해서는 이 고장 신호가 출력 쪽으로 전파되다가 멈춘 노드의 X 값을 갖는 다른 입력선들을 비제어값(non-controlling value)으로 만들어야 한다. 이를 초기 목적(initial objective)이라 하고, 이 선들을 초기 목적선(initial objective line)이라 한다. 백트레이스 과정은 이런 초기 목적을 이루기 위해 필요한 정보를 초기 목적선으로부터 주입력쪽으로 전달한다. 임플리케이션 과정은 백트레이스 과정에서 선택된 주입력 노드(또는 회로내의 노드)와 그 노드에 지정된 값의 영향으로 결정되는 회로내의 다른 모든 노드들의 값을 정하는 것을 뜻한다. 임플리케이션 과정은 전향 임플리케이션(forward implication)과 후향 임플리케이션(backward implication)이 있는데, 전향 임플리케이션에서는 노드들의 값이 결정되어 전파되는 방향이 출력쪽이고, 후향 임플리케이션은 입력 방향으로 전파된다.

FAN 알고리즘은 PODEM에서 없는 다중 백트레이스(multiple backtrace)와 유일 감응(unique sensitization) 등의 보다 개선된 임플리케이션 방법을 제안하였다. SOCRATES는 FAN보다 더욱 백트레이스와 임플리케이션 과정을 개선하였다. 이 방법들은 전체 고장 집합에서 하나의 대상 고장을 선택하여 검사를 발생하고, 발생된 검사에 의해 검출될 수 있는 또 다른 고장들을 고장 집합에서 찾기 위해 고장 시뮬레이션을 하게 된다. 고장 시뮬레이션에 의해 많은 고장

들이 고장 집합에서 삭제되고 나머지 고장들에서 또 다른 대상 고장을 선택하여 검사를 발생하는 과정이 반복된다. 이런 검사 발생 과정에서 선택되는 대상 고장들의 연속을 F1, F2, ..., Fm이라 하자.

검사 큐브(cube) T_i^j는 고장 F_i에 대한 검사 발생 과정에서 j 번째 결정 후의 모든 입력선에 할당된 상태를 나타낸다. T_i^0와 T_i^f는 각각 초기 입력 상태와 검사 발생이 완료된 후의 마지막 입력 상태를 나타낸다. 하나의 검사 큐브는 n-투플(tuple)$(x_1, x_2, ..., x_n)$로 나타낼 수 있는데, x_i는 0, 1, 또는 X 값을 갖는다. 고장 F_i에 대한 회귀 탐색 방법을 사용하는 검사 발생 과정은 초기 검사 큐브 T_i^0에서 시작하여 최종 검사 큐브 T_i^f를 형성해 나가는 과정이라 볼 수 있다. 기존의 검사 발생 알고리즘들은 고장 F_i에 대한 검사 발생을 회로내의 모든 노드들에 X 값을 할당하여 시작하게 된다. 즉, 초기 검사 큐브 T_i^0는 (X, X, ..., X)가 된다. 고장 F_i에 대한 검사 발생이 종료되면 검사 큐브 T_i^f에 많은 원소가 0 또는 1 값을 갖게 된다. 이를 검사 T_i, 또는 T_i에 할당된 값이라 하자.

NTPG에서는 기존의 방법과 다르게 초기 검사 큐브 T_{i+1}^0를 T_i^f 값으로 시작하여 고장 F_{i+1}에 대한 검사 발생을 시작한다. 회귀 한계를 낮게 두고서 검사 T_i로부터 전수 받은 값에서 시작하여 점차로 새로운 검사 T_{i+1}를 생성해 나간다. 앞 검사에서 전수 받은 논리 값들은 새로운 결정에 의해 회귀 없이 변경할 수 있지만, 새로운 결정에 의해서만 제어되는 논리 값들은 회귀 없이 변경할 수가 없다. 따라서 전수 받은 논리 값과 새로운 결정에 의해 제어되는 논리 값들을 구분할 수 있는 논리 시스템이 필요하게 되는데 이는 3.2 절에서 설명한다.

검사 T_i에 할당된 값들을 사용하면 고장 F_{i+1}에 대한 검사 발생을 할 때 많은 결정 단계를 뛰어 넘는 경우가 발생한다. 예를 들어, PODEM에서 고장 F_i와 고장 F_{i+1}에 대한 검사 T_i=(1, 0, X, 0, 1, X)와 T_{i+1}=(1, 1, 0, 0, 1, X)를 발생한다고 하자. PODEM에서는 내부 노드에서 결정과 후향 임플리케이션이 일어나지 않으므로, 적어도 한 입력선에 논리값을 할당하기 위해서는 백트레이스 결과에 의한 결정 단계가 필요하다. 따라서 고장 F_{i+1}에 대한 검사 T_{i+1} 발생시 적어도 5 번의 결정이 발생할 것이다. 회귀가 발생하게 되면 더욱 많은 결정 단계가 필요하게 된다. 하지만 고장 F_{i+1}에 대한 검사 T_{i+1} 발생을 T_i^f를 전수받아 시작하면, 백트레이스 과정에 의해 고장 신호를 발생시키거나 이 신호를 출력 쪽으로 전파하기 위해 변경이 필요한 입력선들을 찾게된다. 간단히 설명하기 위해 백트레이스 경로가 전수받은 값에 의해 영향을 받지 않는다고 가정하면, 입력 x_2=1과 x_3=0에 대한 2 번의 결정 단계만 필요할 것이다. 이렇게 하여 발생된 검사는 (1^*, 1, 0, 0^*, 1^*, X)이 될 것이고, *-표시된 값들은 전수된 값들을 뜻한다. 그러나, 전수 받은 상태에서의 초기 목적과 백트레이스 과정에서 거치는 경로가 다르고, 전수 받은 것들 중에서 이용이 가능한 값도 변경될 수 있으므로 이렇게 하여 생성된 검사는 T_{i+1}와 달라질 수 있다.

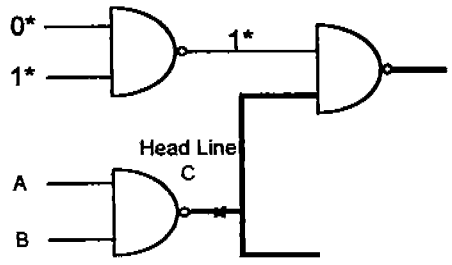
3. 전수받은 검사의 사용

3.1 전수받은 검사 패턴을 이용하는 효과

일반적으로 연속된 두 검사간에 비트의 변화는 검사들에서 유효한 비트들의 평균 길이에 비하여 상당히 작다[16]. 즉, 고장 F_{i+1}에 대한 검사 T_{i+1}는 많은 신호 선들이 T_i와 같은 값을 갖는다는 것을 뜻한다. 극단적인 예로, 한 입력선의 s-a-1(stuck at 1) 고장에 대한 검사 T_{i+1}는 같은 입력선의 s-a-0(stuck at 0) 고장에 대한 검사 T_i로부터 그 입력선의 값만 반전하면 된다. 이 두 고장은 서로 독립된 고장(independent faults)들로서 결코 하나의 검사에 의해 검출되지 않는다[8]. 그리고 헤드라인(head line)[2]을 통과해야 하는 고장들에 대한 검사 발생시, 헤드라인 이후의 신호들은 공통적으로 사용할 수 있다. 이들 신호는 단지 헤드라인을 통과한 고장 신호를 출력까지 전달시키는 역할만을 하기 때문이다. 예를 들면 (그림 1)에서 A 또는 B 노드의 고장 신호는 헤드라인 C를 반드시 통과해야 한다. 이 경우에 고장신호를 발생하여 헤드라인까지 전파하는 신호들과 헤드라인 이후에 고장신호를 출력까지 전파하는 신호들은 서로 독립적으로 설정되므로, 이들 헤드라인 이후부터 출력까지 전파하기 위한 신호들은 A 또는 B 노드의 고장에 대한 검사에 공통적으로 사용된다.

전수받은 검사를 효과적으로 사용하면 이미 많은 시간을 사용하여 계산한 결과들을 다음 검사 발생에 이용함으로써 전체 성능을 높일 수 있다. 하지만 전

수 받은 값들을 전부 이용하기도 힘들고, 어떤 신호가 유용한지 아니면 변경되어야 되는지를 찾기는 검사 발생만큼 어려운 일이다. 전수받은 신호들중에는 고장 신호를 생성하거나, 고장 신호를 전파하는 역할을 도울 수도 있고, 일부는 방해할 수도 있다. 방해되는 신호를 변경시키는 것을 초기 목적(initial objective)으로 하여 백트레이스를 시작하여서 초기 목적을 만족시킬 만한 입력선과 그 값을 찾아낼 수 있다. 회귀 탐색 과정에 의해 이와 같이 방해되는 신호로 여겨지는 것들을 하나씩 변경시키게 되는데, 변경되어야 할 입력선의 수가 많지 않은 경우는 쉽게 검사를 발생할 수가 있을 것이다. 하지만 전수 받은 신호를 하나씩 변경해 나가면, 전수 받은 신호들중에 고장 신호를 전파하는 역할을 하던 것이 논리 값이 변하여 이들도 변경해야 되는 경우가 발생할 수 있다. 이런 과정에서 새로 변경된 값들에 의해서만 신호값이 결정되는 노드들과 전수받은 값에 의해 영향을 받는 노드들을 구분할 수 있어야 한다. 새로운 결정에 의해 정해진 값들을 고정된 값이라 하는데 이들을 변경하는 것은 회귀에 의해서만 이루어진다. 전수받은 값에 의해 영향을 받는 값을 가변 값이라 하는데 이들은 회귀 없이 새로운 결정에 의해 변경이 가능하다.



(그림 1) 전수 받은 값의 이용
(Fig. 1) Use of inherited signals

3.2 전수받은 검사 벡터를 사용한 계산 방법

한 노드의 입력들의 값이 고정된 값과 가변의 값들이 섞여 있을 경우에 노드의 값은 다음과 같이 결정된다. 한 노드의 입력들 중에서 고정된 제어 값을 갖는 입력은 모든 다른 입력들의 값과 관계없이 그 노드의 값을 결정하고 고정된 값을 갖게한다. 입력들 중에서 고정된 제어 값을 갖는 것이 없으면 가변 값을 갖는 입력이 노드의 값을 제어한다. 즉, 가변 값을 갖는 입력들을 변경하여 노드의 값을 새로운 값으로 만들 수 있다. 하나의 고장에 대한 검사 발생이 끝난 후에 다음 검사 발생을 하기 위해 모든 고정된 값을 X 값으로 초기화하지 않고 똑같은 논리값을 갖지만 변경이 가능한 가변 값으로 바꾸어 놓는다.

고정된 값은 기존의 방식과 똑같이 나타내고, 가변의 값들은 이들을 구분하기 위해 *를 붙여서 표시한다. 즉, 총 9 개의 논리 값, X, 0, 1, D, $\overline{D}$, 0*, 1*, D*, $\overline{D}$*, 이 필요하게 된다. 〈표 1〉은 가변의 값과 고정된 값이 혼합되어 있을 때의 AND 게이트(gate)에 대한 계산법을 나타낸다. 〈표 1〉에서 보면 고정된 값과 가변의 값이 혼합된 경우에서 고정된 제어값(AND 게이트의 경우에 0 값)이 있는 경우에는 노드의 값이 고정된 값을 갖게 되고, 고정된 제어값이 없는 경우에는 가변의 값을 갖게 된다. 즉, 임의의 가변의 값을 갖는 입력을 변경시켜서 노드의 값을 변경시킬 수 있다.

〈표 1〉 AND 게이트에 대한 논리 계산
〈Table 1〉 Logic calculus for 2-input AND gate

∩	X	0	1	D	$\overline{D}$	0*	1*	D*	$\overline{D}$*
X	X	0	X	X	X	0*	X	X	X
0	0	0	0	0	0	0	0	0	0
1	X	0	1	$\overline{D}$	0*	1*	D*	$\overline{D}$*	
D	X	0	D	D	0	0*	D*	D*	0*
$\overline{D}$	X	0	$\overline{D}$	0	0*	$\overline{D}$*	0*	$\overline{D}$*	
0*	0*	0	0*	0*	0*	0*	0*	0*	0*
1*	X	0	1*	D*	$\overline{D}$*	0*	1*	D*	$\overline{D}$*
D*	X	0	D*	D*	0*	0*	D*	D*	0*
$\overline{D}$*	X	0	$\overline{D}$*	0*	$\overline{D}$*	0*	$\overline{D}$*	0*	$\overline{D}$*

게이트들의 평균 입력수가 작고 3 가지 이상의 값을 갖는 경우에 상태 천이표 방법을 사용하면 효과적이다. 논리값 계산은 상태 천이표와 게이트의 각 입력 값에 의해 제어되는 유한 상태 기계(finite state machine)에 의해 계산된다. 유한 상태 기계의 최종 상태가 논리 값을 나타내게 되는데, 논리 값 계산과 더불어 필요한 D-프론티어 상태도 얻을 수 있다. (그림 2)는 상태 천이표를 사용한 논리 계산법을 보여준다.

```
s := s1 ; { initial state for a gate type }
for i:= 1 to net_list[id]->num_input
        input_value := get_input_value(id,i);
        s := STT[s][input_value];
end for
```

(그림 2) 상태 천이표를 사용한 논리 계산
(Fig. 2) Logic evaluation using state transition table

3.3 백트레이스 과정에서의 목적 전달

백트레이스 과정은 초기 목적을 이루기 위해 필요한 정보를 초기 목적 선으로 부터 주입력 쪽으로 전달한다. 한 게이트의 목적은 목적 내용과 그 게이트의 유형에 따라 새로운 목적이 그 게이트의 입력선들에 전달된다. 하나의 목적은 (m, n_0, n_1)으로 정의 되는데, m은 노드를 뜻하고 n_0와 n_1는 각각 노드 m의 값이 0과 1이 되기를 원하는 노드들의 수를 뜻한다고 볼 수 있다. 초기 목적선에 대한 목적은 0 또는 1로 만들려는 것이므로 n_0와 n_1 중 하나만 1이 되고 다른 것은 0이 된다. 그러나 이런 정보가 주입력 쪽으로 전달되는 과정에서 한 개 이상의 노드들을 구동시키는 팬아웃 줄기(fanout stem)의 각 가지(branch)로 부터 전달되는 서로 다른 목적에 의하여 팬아웃 줄기에서의 목적은 0이 아닌 n_0와 n_1 값을 갖을 수 있다.

목적의 전달은 가변의 값을 갖는 노드들에 의해 영향을 받는다. 초기 목적선이 n_0와 n_1 중 하나만 1이 되고 다른 것은 0이므로, 이들이 팬아웃 줄기까지 전달되는 과정의 목적들도 n_0와 n_1 중 하나만 1이 되고 다른 것은 0이 된다. 이런 경우에 목적에 포함된 n_0 또는 n_1은 초기 목적선의 목적을 위해서 필요한 그 노드의 0 또는 1의 값을 나타낸다. 이미 0^* (또는 1^*)의 가변의 값을 갖는 노드에서 같은 0 (또는 1)의 값이 필요한 경우는 그 가변 값이 유지되기를 기대하면서 목적 전달을 멈춘다. 그 외에는 기존의 방법과 똑같은 방법으로 목적을 전달한다. 따라서 0^* 또는 1^*의 값을 갖는 팬아웃 줄기는 각 팬아웃 가지로 부터 서로 상반되는 목적이 전달되지 않는다. 그러나 D^* 또는 $\overline{D}^*$

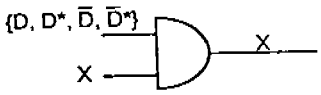
```
Propagate_Objective( l, n0, n1)
begin
        if (l.val=0*) and (n1=0) then return; { compare objective value
        if (l.val=1*) and (n0=0) then return;    with current value      }
        if (Is_fanout_stem(l) then
            l.n0 = l.n0 + n0;
            l.n1 = l.n1 + n1;
            if Not_fan_object(l) then Put_fan_object(l);
            return;
        end if
        if inversion_parity(l) then
            pn0=n1; pn1=n0;
        else
            pn0=n0; pn1=n1;
        end if;
        s=easiest_input(l) ; { select the easiest one to control }
        Put_current_object(s,pn0,pn1);
        if controlling_val = 0 then
            pn0 = 0;
        else
            pn1 = 0;
        end if
        if (pn0 != 0) or (pn1 != 0) then
            for all input i of l with not fixed value and i!=s do
                Put_current_object(i,pn0,pn1);
        if controlling_val = 0 then
            end for
        end if
end;
```

(그림 3) 백트레이스 과정
(Fig. 3) Backtrace procedure

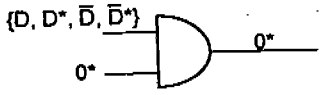
값을 갖는 팬아웃 줄기는 각 팬아웃 가지로 부터 서로 상반되는 목적이 전달될 수도 있다. (그림 3)은 목적을 전달하는 백트레이스 과정을 나타낸다.

3.4 D-프론티어와 비인정선

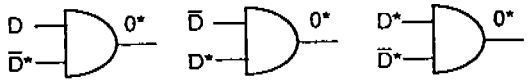
전수받은 값들을 이용하는 경우에는 D-프론티어에 속하는 게이트들이 다양해진다. D-프론티어들은 입력선중에 적어도 하나가 고정 또는 가변의 고장 신호를 갖고 있고 노드의 출력 값이 X 또는 가변의 0* 또는 1*을 갖는 경우이다. (그림 4(a))는 출력이 X 값을 갖는 경우를 나타낸다. 이 경우에 X 값을 갖는 입력선을 비제어값으로 정해주면 고장 신호를 전파시킬 수 있다. (그림 4(b))는 가변의 제어값을 갖는 입력선에 의해 고장 신호가 전파되지 못하는 경우이다. 가변의 제어값을 갖는 입력선을 비제어값을 갖게 해주면 된다. (그림 4(c))는 서로 상반되는 가변 고장 신호에 의해 상호 고장 신호를 상쇄시키는 경우이다. 가변 고장 신호중의 하나를 비제어값으로 정해주면 된다.



(a) X 값을 갖는 경우



(b) 제어 값을 갖는 경우



(c) 상호 상쇄

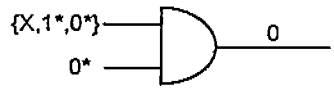
(그림 4) D-프론티어 종류
(Fig. 4) Types of D-frontier gates

전수받은 가변 값들은 회귀없이 새로운 결정에 의해서 변경이 가능하므로 D-프론티어의 한 고장 신호를 출력으로 전파시키기 위하여서 비제어 값, 또는 전파되는 고장 신호와 똑같은 값을 갖게 하면 된다. 즉, 전파시키려는 고장 신호와 반전된 신호값을 갖거나 가변의 제어 값을 갖는 입력선을 초기 목적선에 두게 된다. 가변의 비제어 값을 갖는 입력선들은 변할지 모르므로 고정시켜 두는게 좋지만 이를 위한 작업량이 크기 때문에 가변의 값 상태로 두고 사용한다.

FAN에서 많은 비인정(Unjustified)선들이 유일 감응, 고장 삽입, 팬아웃 줄기에서의 결정들에 의하여 발생된다. 고장 삽입과 유일 감응에 의해 회로 내의 신호선들이 입력선에 의하지 않고 결정되게 된다. 이렇게 회로 내부에서 정해진 신호를 정당화(justification)하기 위하여 이들을 위한 입력선들의 값을 정해야 한다. 후향 임플리케이션(backward implication)은 입력선들이 유일하게 결정되는 경우에 이들의 값을 정하게 되는데, 유일하게 결정짓지 못하는 경우는 비인정선으로 남게된다. 예를 들어 AND 게이트의 경우에 출력이 1의 값을 갖고 있다면 모든 입력은 비제어값인 1의 값으로 유일하게 결정된다. 하지만 AND 게이트의 출력이 0이 되는 경우는 어느 입력선 하나만 제어값인 0을 갖고 있으면 된다. 이런 경우에 어느 입력선으로 유일하게 정할 수 없기 때문에 이 게이트의 입력선은 어느 하나도 정할 수 없고 이 게이트의 출력선을 비인정선에 포함시켜서 초기 목적선(initial objectives)에 포함시켜 탐색공간에서 찾게 한다.

전수받은 값을 사용하는 경우의 후향 임플리케이션에서 유일하게 결정되는 가변의 비제어값을 갖는 입력선들은 고정된 비제어값으로 바꾸어 놓는 것이 좋다. 이렇게 함으로써 신호선의 선택할 수 있는 경우의 수를 줄임으로써 회귀 상태를 일찍 발견하여 필요없는 계산 시간을 줄일 수 있다. 유일하게 결정짓지 못하고 비인정선에 속한 것들은 전부 고정된 값을 갖고 있는데, 이들의 입력선중에 하나를 고정된 제어값으로 만들면 된다. 이 경우에 두가지 비인정선이 있게 되는데, P-비인정선과 N-비인정선이다. P-비인정선은 (그림 5(a))와 같이 적어도 하나의 입력선이 가변의 제어값을 갖는 경우이다. N-비인정선은 (그림 5(b))와 같이 입력선 중에 가변의 제어값이 없는 경우이다. 초기 목적선에는 P-비인정선은 포함되지 않는다. 즉, 전수받은 값을 이용하여 P-비인정선에 대한 목적을 만족시키기 때문이다. 따라서 N-비인정선만 초기 목적선에 포함된다. 테스트 발생 도중에 비인정선에 속한 한 선은 N-비인정선과 P-비인정선으로 여러번 상태가 변경될 수도 있다. 테스트 벡터가 성공적으로 끝나는 조건은 고장 신호가 주 출력선으로 전

파되고 N-비인정선이 없게되는 것이다.



(a) P-비인정선

(b) N-비인정선

(그림 5) 비인정선 종류
(Fig. 5) Types of unjustified lines

3.5 병합한 검사 패턴 발생 방법

전수받은 검사를 이용하면 회귀 한계가 커지면서 성능이 떨어질 수가 있다. 임플리케이션 과정에서 신호들이 쉽게 출력 쪽으로 전파되기 때문에 기존의 방법에 비해 결정 횟수가 많아지면 훨씬 많은 논리 계산이 필요하게 된다. 가변의 고장 신호는 회귀 없이 변경이 되므로 D-프론티어도 항상 출력쪽으로 진행하지 않고, 입력 쪽으로 후퇴하는 경우도 발생할 수 있다. 이런 경우가 자주 발생하면 전체 성능은 떨어지게 될 것이다. 물론 검사 패턴이 존재하는 고장들에 대해서는 회귀 한계를 충분히 높게 두면 항상 검사 패턴을 발생시킬 수 있다. 회귀 한계를 낮게하여 전수받은 검사를 이용하여 검사 패턴을 발생하다가 높은 회귀 한계가 필요한 경우들은 기존의 방법들을 사용하는 다음 단계로 넘긴다. 성공적으로 검사가 발생된 고장들의 일부는 기존의 방법으로는 높은 회귀 한계에서도 검사를 찾지 못하는 것 들일 수가 있다. 이들이 전체 성능에 미치는 영향은 크다. 즉 대부분의 고장들은 낮은 회귀 한계에서 기존의 방법에 비해 쉽게 검사가 발생되고, 이들 중 일부는 기존의 방법으로 검사가 찾기 어려운 것들도 포함되게 된다. 낮은 회귀 한계에서 한번 시도해서 포기한 것들은 순수히 overhead가 되는데 무시할 수가 있을 것이다.

병합한 경우의 단계 1에서 회귀 한계를 10으로 두고 전수받은 값을 이용하여 검사 패턴을 발생하고, 고장 시뮬레이션을 하여 발생된 검사에 의하여 검출되는 다른 고장들을 찾아 고장 집합에서 삭제한다. 그리고 다음 고장에 대한 검사 패턴을 발생하기 위하여 현재의 논리 값들을 *가 붙은 값으로 변경하여 다음 검사 패턴 발생에 사용한다. 회귀 한계 10 안에서 검사 패턴을 찾지 못하는 고장들은 다음 단계로 넘긴다. 단계 2에서는 회귀 한계를 1000으로 늘이고 한 고장에 대한 검사 패턴 발생이 성공적으로 끝나면 고장 시뮬레이션 후에 논리 값들을 X 값으로 초기화 한다. 즉, 전수 받지 않으면 알고리즘은 FAN과 같이 동작하게 된다. 이런 방법으로 임의의 경로 감응 알고리즘을 변경하여 2 단계로 만드는 것은 어렵지 않을 것이다. 메모리 요구사항은 기본적으로 같은 구조의 알고리즘을 사용하여 자료구조가 비슷하게 요구되므로 큰 변동이 없고, 단지 3.2 절에서 언급한 논리 계산법을 위한 표의 크기가 커지는데 무시할 정도이다.

4. 실험 결과

백트레이스 과정에서 이용되는 각 노드들의 검사도(testability)들은 SCOAP[9]의 방식을 이용하였다. 각 고장에 대하여 발생된 검사 패턴들에서 사용하지 않는 입력들이 많이 나타나게 된다. 이들을 이용하여 동시에 검출할 수 있는 더 많은 고장들을 찾게 되면 전체 검사 집합의 크기가 줄어들 것이다. 이를 검사 압축(test compaction)이라 하는데 검사 패턴 발생이 끝난 후에 검사 패턴들에 대한 대수학적 조작에 의하여 압축하거나 또는 하나의 고장에 대한 검사 패턴 발생이 끝난 후에 사용하지 입력들을 이용하여 압축하는 방법이 있다[11]. 본 실험에서는 검사 패턴 발생 알고리즘의 성능만을 비교하기 위하여 압축과정은 생략하였고, 고장 시뮬레이션은 동시 고장 시뮬레이션(concurrent fault simulation)[12] 방법을 이용하였다.

전수받은 검사를 이용한 검사 발생 방법을 NTPG라 하고, NTPG와 FAN 알고리즘을 병합시킨 것을 CTPG라한다. NTPG와 CTPG를 Solbourne 5/701 workstation에서 구현하여 FAN과 서로 비교하였다. ISCAS'85 벤치마크 회로에 대하여 회귀 한계를 10으로 하여 NTPG와 FAN 알고리즘을 비교한 결과를 〈표 2〉에 보였다. 〈표 2〉에서 faults는 ISACAS'85에서 주어진 고장의 수이고 test는 발생된 검사 패턴의 수인데 압축 과정을 생략하였기 때문에 그 수가 크다. abrted와 cvrage는 각각 포기된 고장들의 수와 고장 검출율을

나타낸다.

전체적으로 NTPG는 포기되는 고장들의 수가 FAN에 비해서 작다. 즉 회귀 한계를 10으로 아주 낮게 했어도 많은 고장들이 검출됨을 보여준다. 실행시간은 unix의 gprof 유틸리티를 사용하여 측정하였고, 고장 시뮬레이션은 검사 패턴 발생 방법에 관계없이 똑같은 방법이 사용되므로 제안된 검사 패턴 발생의 효과만을 비교하기 위하여 이에 소비된 시간은 포함시키지 않았다. NTPG는 FAN에 비하여 전체 실행시간이 40%에서 60%의 시간정도가 감소됨을 알 수 있다. 선택되는 고장의 순서도 영향을 미치리라 고려 되는데 경계 고장(boundary fault)[4]들 중에서 임의의 순서대로 선택한 것과 경계 고장들 중에서 위치적으로 가까운 것들을 먼저 하는 것과의 실험 결과는 별 차이가 없었다. 그리고 전수받은 값을 이용하여 쉽게 검사 패턴을 찾을 수 있는 대상 고장을 찾는 것도 쉽지가 않다.

〈표 2〉 NTPG와 FAN의 비교(회귀 한계 : 10)
〈Table 2〉 Comparison of NTPG and FAN(backtack limit : 10)

회로 이름	Faults	NTPG				FAN			
		test	abrt	cvrage	time (sec)	test	abrt	cvrage	time (sec)
c1355	1574	175	56	96.44	10.2	182	501	68.17	49.6
c1908	1879	297	4	99.79	4.7	319	21	98.88	12.9
c2670	2747	750	53	98.07	9.5	778	103	96.25	15.7
c3540	3428	745	20	99.42	26.3	901	20	99.42	38.3
c5315	5350	1385	7	99.87	10.0	1376	121	97.74	30.8
c6288	7744	367	22	99.72	14.9	120	146	98.11	39.3
c7552	7550	1098	87	98.85	49.9	998	306	95.95	114.4

NTPG의 회귀 한계를 1000으로 증가하였을 때의 결과를 〈표 3〉에 보였다. 성능이 좋아 지는 경우도 있지만 회로 c2670과 c3540의 경우에 성능이 나빠짐을 알 수 있다. c2670의 경우에 포기된 고장수가 적음에도 불구하고 시간이 더 걸린 이유는 한 신호선의 변경에 대한 영향이 쉽게 출력 쪽으로 전파되기 때문인 것으로 추정된다. 그리고 c3540의 경우에는 포기된 고장이 많은 이유는 가변의 고장 신호를 갖고 있는 노드들도 D-프론티어로 간주하기 때문에 고장신호가 항상 출력쪽으로 전파되는 것이 아니고 입력 쪽으로 후퇴하는 경우도 많이 발생하는 것으로 추정된다. 이런 현상을 방지하기 위하여 회귀 한계를 낮게 두어 대부분의 고장에 대한 검사 패턴 발생을 효율적으로 하고, 회기가 자주 발생할 것으로 추정되는 것들은 일찍 포기하여 FAN으로 처리한다.

〈표 3〉 NTPG 와 FAN 의 비교(회귀 한계 : 1000)
〈Table 3〉 Comparison of NTPG and FAN(backtack limit : 1000)

회로 이름	Faults	NTPG				FAN			
		test	abrt	cvrage	time (sec)	test	abrt	cvrage	time (sec)
c1355	1574	176	9	99.43	47.3	164	328	79.16	858.9
c1908	1879	301	2	99.89	14.6	325	4	99.79	42.8
c2670	2747	750	37	98.65	233.5	799	39	98.58	211.0
c3540	3428	753	13	99.62	183.2	894	0	100.00	41.7
c5315	5350	1389	1	99.98	21.1	1441	2	99.96	53.2
c6288	7744	387	2	99.97	50.7	156	37	99.52	429.1
c7552	7550	1089	70	99.07	484.6	1094	114	98.49	1069.0

회귀 한계를 1000으로하여 병합한 알고리즘인 CTPG와 FAN 알고리즘을 비교한 결과를 〈표 4〉에 보였다. CTPG의 경우는 먼저 회귀 한계를 10으로 하여 전수받은 값을 이용하여 검사를 발생하고 나서 포기되는 고장들에 대해서 회귀 한계를 1000으로 변경하여 다시 FAN 알고리즘에 의해 처리하였다. 〈표 4〉의 고장 검출율을 보면 고장 수에 비해 포기된 고장의 수가 너무 적어서 큰 차이가 없는 것처럼 보인다. 그러나, 전체 성능은 이들 포기된 고장에 소비된 시간에 영향을 받기 때문에 검출율을 조금 향상 시켜도 전체 성능은 많이 향상된다. c5315의 경우에 2 개의 고장이 회귀 한계 1000에서도 포기되었지만 이들은 NTPG에 의하여 회귀 한계 10에서 발생된 검사 패턴에 의해 검출됨을 알 수 있다. c6288과 c7552의 경우는 훨씬 더 많은 고장들이 이런 경우에 해당됨을 알 수 있다. 검사 발생에 소비된 시간은 회로에 따라 성능 향상 편차가 크지만 대략 50% 정도로 시간이 줄어드는 것을 볼 수 있다. 전수받은 검사를 이용하면 FAN 알

고리즘에서 실패하는 고장들의 일부에 대한 검사를 낮은 회귀 한계에서 성공적으로 찾아낸다는 것을 알 수 있다.

〈표 4〉 CTPG 와 FAN 의 비교(회귀 한계 : 1000)
〈Table 4〉 Comparison of CTPG and FAN(backtack limit : 1000)

회로 이름	Faults	CTPG				FAN			
		test	abrt	cvrage	time (sec)	test	abrt	cvrage	time (sec)
c1355	1574	178	46	97.08	117.2	164	328	79.16	858.9
c1908	1879	298	3	99.84	14.5	325	4	99.79	42.8
c2670	2747	757	31	98.87	156.4	799	39	98.58	211.0
c3540	3428	759	0	100.00	27.8	894	0	100.00	41.7
c5315	5350	1392	0	100.00	10.1	1441	2	99.96	53.2
c6288	7744	380	7	99.91	67.4	156	37	99.52	429.1
c7552	7550	1113	59	99.22	451.9	1094	114	98.49	1069.0

5. 결 론

본 연구에서는 아주 낮은 회귀 한계에서 전수받은 값을 사용하여 효과적으로 검사를 발생하는 방법을 제안하였다. 기존의 방법들은 주어진 고장에 대한 검사 패턴 발생시 회로의 주입력들의 상태를 X 값에서 시작하여 회귀 탐색 과정을 통하여 이들의 값을 정해 나가지만, 제안된 방법은 바로 전 고장에 대한 검사 패턴에서 전수받은 값을 점차로 바꾸어 나가면서 새로운 검사벡터가 발생이 된다. 전수받은 값은 부분적으로 고장 신호를 활성화하고 이 고장 신호를 전파시키기도 한다. 보통 이들은 다음 탐색과정에서 결정단계의 수와 회귀의 수를 감소 시킨다. 이 방법에서는 회로내의 모든 노드들이 값을 가지고 있기 때문에 한 노드의 값이 변하면 쉽게 출력쪽으로 전파되게 된다. 따라서 회귀 탐색 과정에서 하나의 결정에 대한 계산량이 많아져서 회귀 한계가 높아지면 성능이 떨어질 수가 있다. 기존의 방법에 비하여 고장에 따라서는 더 많은 계산이 필요한 경우도 발생할 수 있지만, 기존의 방법으로 높은 회귀 한계에서도 포기되는 고장들에 대한 검사를 쉽게 찾아낼 수도 있다. 낮은 회귀 한계에서의 실험결과는 전체 실행시간이 40%에서 60% 정도 감소한다. 제안된 방법을 기존의 방법과 병합시킴으로써 임의의 회귀 한계에서도 전체 고장 검출율을 효과적으로 높일 수 있는데, 회로에 따라 성능 향상 편차가 크지만 대략 50% 정도로 시간이 줄어드는 것을 보여주었다.

참 고 문 헌

[1] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. Computers, Vol C-30, No. 3, pp. 215-222, Mar. 1981.

[2] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms", IEEE Trans. Computers, Vol C-32, No. 12, pp. 1137-1144, Dec. 1983.

[3] M.H. Schulz, E. Trischler, and T.M. Sarfert, "SOCRATES:a highly efficient automatic test pattern generation system", IEEE Int. Test Conference, pp. 1016-1026, 1987.

[4] M. Abramovici, J.J. Kulikowski, P.R. Menon, and D.T. Miller, "SMART and FAST:Test Generation for VLSI scan design circuits", IEEE Design & Test of Computers, pp. 43-54, Aug. 1986.

[5] V. Agrawal and K. Cheng, "A directed search method for test generation usings concurrent simulator", IEEE Trans. Computers, Vol. 8, pp. 131-138, Feb. 1989.

[6] K. Cheng and V. Agrawal, "A simulation-based directed search method for test generation", Proc. Int. Conf. Computer Design(ICCD-87), pp. 48-51, Oct. 1987.

[7] M. Cutler, S.Y.H. Su and M. Wang, "Test Generation by Critical Backtracing with Time Reducing Heuristics", IEEE Int. Test Conf., pp. 1035-1042, 1987.

[8] S. Akers and B. Krishnamurthy, "On the role of independent fault set in the generation of minimal test sets", IEEE Int. Test Conference, pp. 1100-1107, 1987.

[9] L.H. GoldStein and Everlyn L. Thigpen, "SCOAP:

Sandia Controllability/Observability Analysis Program", Proc. of 17th Design Automation Conference, pp. 190-196, June 1980.

[10] M. Abramovici, P.R. Menon, and D.T. Miller, "Critical Path Tracing: An Alternative to fault Simulation", IEEE Design & Test of Computers, Vol. 1, No. 1, pp. 83-93, Aug. 1984.

[11] P. Goel and B.C. Rosales, "Test generation and dynamic compaction of test sets", Proc. Test Conf., pp. 189-192, Oct. 1979.

[12] E.G. Ulrich and T.G. Baker, "Concurrent Simulation of Nearly Identical Digital Networks", Computer, Vol. 7, No. 4, pp. 39-44, April 1974.

[13] O.H. Ibarra and S. Sahni, "Plynomially Complete Fault Detection Problem", IEEE Trans. Computers, Vol C-24, No. 3, pp. 242-249, Mar. 1975.

[14] D.B. Armstrong, "A Deductive Method of Simulating faults in Logic Circuits", IEEE Trans. Computers, Vol C-21, No. 9, pp. 464-471, May 1972.

[15] M. Abramovici, P.R. Menon, and D.T. Miller, "Checkpoint faults are not sufficient target fault for test generation", IEEE Trans. Computers, Vol C-35, No. 8, pp. 769-771, Aug. 1986.

[16] S. Song and L. Kinney, "Incremental Test Pattern Generation", 11th IEEE VLSI Test Symposium, pp. 244-250, April 1993.

송 상 훈

1977년 연세대학교 전자공학과 학사

1979년 한국과학기술원 전산학과 석사

1992년 University of Minnesota, 전산학과 박사

1992년~현재 세종대학교 전산학과 조교수

관심분야: Fault tolerant computing, VLSI testing, 컴퓨터 구조, 병렬처리