論文97-34C-8-5

빠른 무해 인식에 의한 효율적인 테스트 패턴 생성

(An Efficient Test Pattern Generation Based on the Fast Redundancy Identification)

韓 相 潤 *, 姜 成 昊 **

(Sang Yun Han and Sung Ho Kang)

요 약

효율적인 테스트 패턴 생성을 수행하기 위해 빠른 무해 인식이 필요하다. 무해 고장은 목적들간의 의존과 고장 전파를 방해하는 재합류 분기점들에 의해 발생한다. 본 논문에서는 조합 회로에 존재하는 무해 고장을 검출하는데 유용한 구조 동일성 알고리듬과 유사 지배자 알고리듬을 제안한다. 구조 동일성 알고리듬은 BDD를 이용하여 고장 검출을 위해 반드시 필요한 목적들이 입력들로부터 동시에 만족될 수 없음을 보인다. 유사 지배자 알고리듬은 주어진 분기점까지의 모든 경로를 고려하여 고장 전파가 가능한지를 결정한다. 벤치 마크 회로에 대한 실험 결과는 알고리듬들의 효율성과 실용성을 보여준다.

Abstract

The fast redundancy identification is required to perform an efficient test pattern generation. Due to the reconvergent fanouts which make the dependency among objectives and the fault propagation blocking, there may exist redundant faults in the circuit. This paper presents the isomorphism identification and the pseudo dominator algorithms which are useful to identify redundant faults in combinational circuits. The isomorphism identification algorithm determines whether mandatory objectives required for fault detection cannot be simultaneously satisfied from primary input assignments or not using binary decision diagram. The pseudo dominator algorithm determines whether fault propagation is possible or not by considering all paths at a given fanout node. Several experiments using ISCAS 85 benchmark circuits demonstrate the efficiency and practicability of the algorithms

I. 서 론

VLSI 기술이 급속도로 발전함에 따라 회로의 집적도와 복잡도는 대단히 증가하였으며, 또한 더욱 고성능의 칩을 요구하게 되었다. 이에 따라 테스트는 점점 어려워지고 있으며, 테스트에 드는 비용 또한 크게 증가하였다. 테스트란 회로내의 고장(fault)으로 인한 불량 반도체 소자를 검출해내는 것으로 많은 테스트를 할수록 양질의 칩임을 입증할 수 있으나 그만큼 검사 비용의 증가를 가져온다. 따라서 제품의 가격과 품질을 최적화 시킬 수 있는 효율적인 테스트 패턴 생성이 중요하게 되었다. 고장은 회로내의 물리적인 결함을 모델링한 것으로 고착 고장(stuck-at fault), 지연 고장(delay fault), 합선 고장(bridging fault)등 많은 고장 모델이 존재한다. 가장 많은 연구 대상이 되어온 고착 고장은 회로내의 논리값이 0 (stuck-at-0) 또는 1(stuck-at-1)로 영구히 정해져서 발생하는 오동작에 대한 고장 모델이다. 회로내의 고장들은 테스트 패턴

* 正會員, LG 半導體 MP Center
(Multimedia Processor Center LG Semicon Co. Ltd.)
** 正會員, 延世大學校 電氣工學科
(Dept. of Electrical Engineering, Yonsei university)
接受日字:1996年10月15日, 수정완료일:1997年7月25日

생성 과정을 거치면서 검출 고장(detected fault), 무해 고장(redundant fault), 중단 고장(aborted fault) 등으로 분류된다. 고장은 시도 철회(backtrack)를 사용하여 주어진 시도 철회 상한 내에서 고장을 검출하지 못하거나 무해 고장임을 입증하지 못하면 그 고장은 중단 고장이 된다. 시도 철회의 상한을 증가시킨다면 중단 고장은 검출 고장이나 무해 고장이 되지만 실행 시간이 그만큼 증가하게 된다. 구해진 테스트 패턴들은 고장 검출율(fault coverage)을 사용하여 효율성을 나타낸다.

테스트 패턴 생성에 관한 연구는[1] [2] 단일 활성화(unique sensitization)[3] [4], 정적/동적/순환 학습(static/dynamic/recursive learning)[5] [6] [7]과 같은 조사 영역(search space)[8]을 줄이고자하는 것과 다중 후방 추적(multiple backtrace)[3], 조절 용이도(controllability)[9], 관측 용이도(observability)[10] 측정과 같은 발전적 지도법(heuristic)에 관한 것들이 있다. 조사 영역을 줄이고자 하는 것은 회로내에 이미 정해진 값들을 이용하여 새로운 값을 할당하는 간접적인 유추(implication)라 할 수 있으며 이 과정들은 직접적으로 테스트 패턴 생성 시에 영향을 끼치게 된다. 그러나 회로의 크기가 커짐에 따라 불필요한 유추 과정이 선행 처리 단계에서 많은 시간을 사용하면서 테스트 패턴 생성에 큰 도움을 주지 못하는 경우가 발생한다. 발전적 지도법은 테스트 패턴 생성 시 선택(decision) 과정을 효율적으로 수행하기 위한 연구로 조사 영역을 직접적으로 줄이는 것은 아니다.

본 논문에서는 무해 고장 인식[11]을 위해 무해 고장 발생 원인에 근거하여 새로운 알고리듬을 제안한다. 첫째, 초기 목적(initial objective)들간의 상호 의존에 의한 무해 고장을 위해 구조 동일성(isomorphism identification) 알고리듬을 둘째, 고장 전파가 존재하지 않는 무해 고장에 대해 유사 지배자(pseudo dominator) 알고리듬을 제안한다. 이와 같은 알고리듬들을 기존의 테스트 패턴 생성기에 적용함으로써 중단 고장을 빠르게 무해 고장으로 인식하여 테스트 패턴 생성 과정을 효율적으로 수행한다. 본 논문의 테스트 패턴 생성기는 스위칭 전략[12]을 사용하여 고장 영향(fault excitation)과 고장 전파(fault propagation)를 만족시키는 순서를 바꿔서 테스트 패턴 생성 과정을 각각 수행하며, 이때 순환 학습[7]과 시도 철회의 상한, 그리고 제안된 무해 고장 알고리듬을 효율적으로 적용하여 최적의 테스트 패턴 생성기를 구성하게 된다. 다음 장에서 무해 고장의 분류를 정의하고, 3장과 4장에서는 2장에서 분류한 무해 고장을 위한 인식 알고리듬을, 5장에서는 본 논문의 전체 테스트 패턴 생성기의 구성을 제시한다. 6장과 7장에서는 각각 벤치 마크 회로[13]를 사용한 실험 결과와 앞으로의 연구 과제에 대해 논의하고자 한다.

Ⅱ. 무해 고장의 분류

무해 고장은 그 고장에 대한 테스트 패턴이 존재하지 않는 것으로 그림 1에 간단한 무해 고장의 예가 있다. B의 입력에 존재하는 고착 고장 1은 고장 삽입(fault insertion)을 위해 0을 할당하여야 한다. 그리고 B를 통해 고장을 전파시키기 위해 A의 출력에 1을 할당하여야 하지만 고장 삽입을 위해 이미 A의 출력은 0으로 정해져 있으므로 고장 전파를 할 수 없게 된다. 이와 같은 고장을 무해 고장이라 한다.

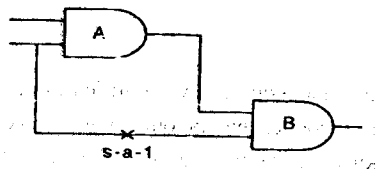

그림 1. 무해 고장의 예
Fig. 1. Redundant fault example.

무해 고장은 회로내의 재합류 분기점(reconvergent fanout)에 의해 구조적으로 발생하는데, 무해 고장의 발생 원인에 따라 다음과 같이 두 가지 형태로 분류하였다.

정의 1 : 고장을 검출하는데 필요한 목적을 만족시키는 입력이 서로 다른 목적들에 의존함 (dependency)으로 인하여 발생하는 무해 고장이다. 이와 같이 목적들간의 상호 의존에 의해 발생되는 무해를 **OCD (Object Correlation Dependency)**라 정의한다.

이의 예를 그림 2에 나타내었는데 그림 2에서 고착 고장 0의 고장 영향 유발을 위해 A와 B중 하나는 반드시 0이 되어야 한다. 그럴 경우 고장을 전파시키기 위해 다른 한 입력은 1이 되어야 한다. 그러나 A와 B가 같은 입력에 의해 서로 의존함으로 다른 값을 할당할 수 없게 된다. 따라서 이 고장을 검출할 테스트 패

던은 존재하지 않는다.

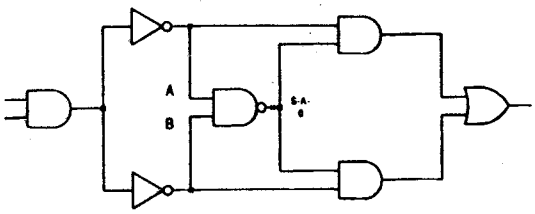

그림 2. OCD
Fig. 2. OCD.

정의 2 : 고장을 출력까지 가능한 모든 경로에 따라 전파시킬 때 각각의 경로에 대해 필요한 논리 값 할당들이 재합류 분기에 의해 모든 경로에서 상충을 일으킴으로써 발생하는 무해 고장이 있다. 이와 같이 고장 전파가 불가능함에 인한 무해를 **PFR (Propagation Failure Redundancy)**이라 정의한다.

이의 예는 그림 3의 회로로서 고장을 진행시키기 위해서는 A와 B 모두에 1의 할당이 필요하나, 고장 삽입을 위해서는 A, B중 적어도 하나에는 0을 할당해야 하므로 그림의 고착 고장에 대한 테스트 패턴은 존재하지 않게 된다.

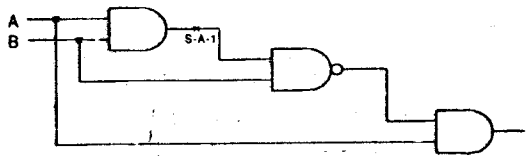

그림 3. PFR
Fig. 3. PFR.

위와 같이 정의된 두 종류의 무해 고장들의 빠른 인식을 위하여 구조 동일성 알고리듬과 유사 지배자 알고리듬을 제안한다.

Ⅲ. 무해고장 인식을 위한 새 알고리듬

1. 구조 동일성 알고리듬

구조 동일성 알고리듬은 BDD[14]를 사용하여 OCD에 의한 무해 고장들 중 고장 검출을 위해 반드시 필요한 두개의 목적들이 서로 의존함에 따라 입력들로부터 동시에 만족될 수 없음을 보임으로써 무해 고장을 인식해 낸다. 이 알고리듬의 설명을 위해 다음을 정의한다.

정의 3 : 고장 검출을 위해 반드시 필요한 할당이 이루진 후, 주어진 목적에 연결되는 주입력을 '**부가 주입력**'이라 정의한다.

정의 4 : 두 목적 Oi와 Oj에 연결된 부가 주입력들을 각각 Gi, Gj라 할 때 Gi와 Gj가 서로 같으면 두 목적 Oi와 Oj는 서로 '의존' 관계를 갖는다고 정의한다.

정의 5 : 두 목적 Oi와 Oj가 의존 관계일 때 부가 주입력들이 동일하므로 Oi와 Oj로부터 주입력까지의 각각의 경로들은 회로 내부의 분기점에서 반드시 만나게 되는데 이 분기점을 '**의사 입력**'이라 정의한다.

정의 6 : 주어진 목적을 주출력으로 하고 의사 입력을 주입력으로 하는 회로를 '**내부 회로**'라 정의한다.

OCD에 의한 무해 고장은 초기 목적들이 서로 의존 관계를 형성하고 동일한 내부 회로를 가질 수 있다. 이와 같은 구조는 한 목적에 대해서는 만족시키는 입력을 구할 수 있으나 그 입력들로 인하여 다른 목적에서 상충을 발생시킨다. 시도철회 방법은 상충이 일어난 경우 기존의 할당된 입력을 변화시켜서 상충을 없애고 다른 입력을 할당하며 상충이 발생할 때마다 이와 같은 과정을 반복하므로 두 목적들에 대해 계속된 상충을 일으켜 시도철회 수만 증가하면서 무해 고장으로 인식할 수 없게 된다. 이와 같이 두 목적의 의존 관계에 의해 발생하는 무해 고장 인식을 위해 다음과 같이 정리하였다.

정리 1 : 고장을 검출하기 위해 서로 다른 논리값을 필요로 하는 게이트를 A, B라 하자. 필요한 할당이 이루어진 후 A와 B가 의존 관계를 갖고 입력으로부터 A와 B까지의 내부 회로가 같은 경우 그 고장은 무해 고장이다.

증명 : 고장 검출을 위해 A = 1(0), B = 0(1)을 만족시켜야 한다고 가정하자. A와 B의 입력이 서로 같다면 A의 값을 만족시키기 위한 입력의 변화는 동시에 B의 값을 변화시킨다. 그러므로 A와 B를 출력으로 하는 내부 회로가 같다면 A와 B의 값을 하나씩은 만족시킬 수 있어도 모든 입력들이 서로 의존하므로 A와 B를 위해 필요한 값을 동시에 만족시킬 수 없다. 따라서 위의 정리 1을 만족시키는 고장은 무해 고장이 된다. ¶

예를 들어 회로의 구성이 그림 4와 같은 경우 구조 2와 구조 3은 의존 관계를 갖게 되며 a, b, c가 두 내부 회로를 위한 의사 입력이 된다. 이 그림에서 구조

4에 존재하는 고장들 중 노드 1과 노드 2에서 서로 다른 값을 목적으로 갖게 될 때 무해 고장이 발생하는 경우는 구조 2와 구조 3의 내부 회로가 동일하여 노드 1과 노드 2에 항상 같은 값이 출력될 때 발생한다. 그러므로 구조 2와 구조 3의 내부 회로가 동일함을 입증한다면 구조 4에 존재하는 고장들 중 노드 1과 노드 2에 서로 다른 값을 필요로 하는 모든 고장들을 무해 고장으로 바로 인식할 수 있다. 이와 같이 두 부분의 구조가 같음을 입증하는 방법은 작은 회로의 경우 모든 할당을 고려할 수 있으므로 큰 회로에서 더욱 효과적이다.

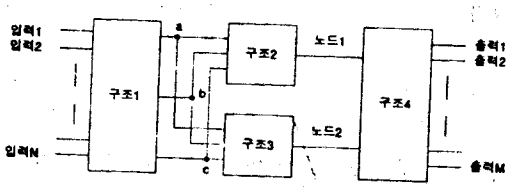

그림 4. OCD 발생 구조
Fig. 4. OCD occurrence structure.

본 연구에서는 두 내부 회로가 서로 같음을 입증하기 위하여 BDD를 사용한다. BDD의 사용은 실행 시간의 증가와 큰 메모리를 요구하므로 구조 동일성 알고리듬의 적용을 줄이기 위해 단계(level)[15]를 고려하였다. 따라서 두 목적의 단계나 입력들이 같은 경우에만 BDD를 만든다. 단계가 다른 경우에도 기능이 같은 경우가 존재하지만 이 경우를 전부 고려하면 너무 경우의 수가 많아지는데 비해 실제 이로 인한 무해 고장의 수는 적어서 무해를 찾는 시간이 너무 오래 걸리게 되어 무해고장 검출이 비효율적이 될 수 있다. 두목적의 선택은 적정 범위 내에서 가능한 조합을 시도하여 선택된다. 비교할 두 목적이 의존 관계를 가지려면 위의 두 조건을 반드시 만족해야 하기 때문이다. 그림 5에 OCD 인식을 위한 구조 동일성 알고리듬이 있다.

그림 6은 제안된 알고리듬을 적용하여 무해 고장을 쉽게 검출할 수 있는 예가 있다. 노드 A의 고착 고장 0은 고장 삽입을 위하여 노드 1과 노드 2중 적어도 하나에는 0을 할당하여 노드 A의 값을 1로 만들어야 한다. 그럴 경우 0이 할당되지 않은 다른 입력은 고장 영향을 전파하기 위하여 1을 할당하여야 한다. 따라서 노드 1과 노드 2는 서로 다른 값을 목적으로 갖는다. 그러나 전체 회로가 그림과 같고 구조 2와 구조 3이 기능적으로 동일할 때 노드 1과 노드 2는 같은 값만을 갖게 되므로 두 목적을 동시에 만족시킬 수 있는 테스트 패턴이 존재하지 않으므로 무해 고장이 된다. 이와 같은 고장은 시도 철회를 하기 전에 이 알고리듬을 적용할 경우 빠르게 무해 고장임을 인식할 수 가 있다. 마찬가지로 그림의 괄호 속에 표시된 다른 고장들도 이 알고리듬을 사용하여 쉽게 무해 고장임을 알 수 있다.

```
OCD(obj1, obj2)
{
  if( obj1->level != obj2->level )   /* 단계가 다르면 중단 */
                                     return(END);
  compare_pi(obj1, obj2);
  if(obj1->no_pi != obj2->no_pi)     /* 입력의 수가 다르면 중단 */
                                     return(END);
  compare_bdd(obj1, obj2);
  if(obj1 != obj2)                   /* 구조가 다르면 중단 */
    return(END);
  else
        return(REDUNDANT);
}
```

그림 5. 구조 동일성 알고리듬
Fig. 5. Isomorphism identification algorithm.

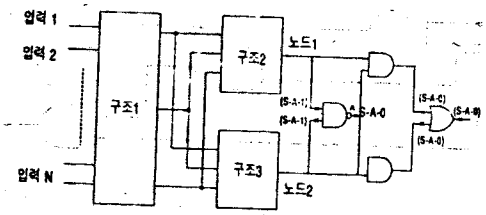

그림 6. BDD를 이용한 OCD 인식
Fig. 6. OCD identification using BDD.

2. 유사 지배자 알고리듬

회로 내부에 삽입된 고장의 영향을 PO까지 전파시키기 위해서 활성화된 경로(sensitized path)의 부입력(side input)들은 부입력들의 경로가 활성화 할 수 있느냐에 따라 다음과 같이 논리값이 할당되어야 한다.

표 1. 고장 전파를 위해 부입력에 할당할 수 있는 조건
Table 1. Condition of side input assignment for fault propagation.

부입력	가능한 논리값
비활성화 경로	비제어값
활성화 경로	비제어값, 고장 영향과 동일한 1/0 또는 0/1

이와 같은 특성은 고장이 존재하는 노드에서 고장의 진행은 다음 분기점 노드까지는 반드시 그 경로를 따라 진행하므로 그 경로에 속하는 게이트들에 대한 부입력값을 할당할 수가 있게 된다. 따라서 고장 영향 전파는 처음 고장이 고려된 후 처음 분기점까지 부입력을 할당하고 분기점 노드에서 어떤 경로를 따라 진행할지를 결정하여 그 경로에 대해 값을 할당하고 상충 발생시 시도 철회를 반복 수행하여 그 고장의 테스트 패턴을 구하거나 무해 고장임을 입증하게 된다. 그러나 회로가 매우 복잡하고 재합류 분기점의 영향에 의해 고장 전파가 불가능한 고장들의 경우에 위와 같은 임의 경로를 반복 할당하는 과정들은 매우 큰 시도 철회를 유발하며 그 고장이 무해함을 인식하는데 많은 시간이 걸리거나 혹은 중단 고장으로 인식하게 된다.

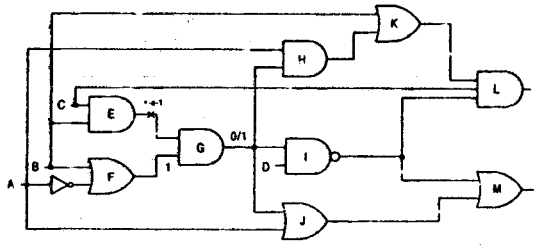

그림 7. PFR 예
Fig. 7. PFR example.

그림 7은 고장을 전파시킬 수 있는 경로가 많이 있음에도 불구하고 재합류 분기점으로 인하여 모든 가능한 경로에 대해서 고장을 진행시킬 수 없음을 보이는 예이다. 이 그림의 회로가 큰 회로의 일부라 가정할 때 G에 위치한 고장 영향이 PO로 진행할 수 있는 경로는 표 2에 나타난 바와 같이 4개가 있다. 표에서 고장 전파 게이트는 각 경로에 대해 거치는 게이트이고, 부입력 할당은 부입력에 고장 전파를 위해 가능한 논리값을 할당한 후의 시뮬레이션 결과를 나타낸다. 표의 결과는 가능한 모든 경로에 대해 상충을 일으키므로 그림의 E에 위치한 고장은 무해 고장임을 알 수 있다. 본 연구에서는 고장의 경로가 다음 분기점에 도달할 때까지는 일정하게 결정되는 특성을 이용하여 분기점에서 한 개의 경로를 임의로 선택하여 다음 분기점 노드까지 그 경로를 통해 고장이 전파되기 위해 필요한 모든 부입력을 한꺼번에 할당하여 그 경로에 대한 고장 전파가 가능한지를 조사한다. 이와 같은 과정을 분기점에서 가능한 모든 경로에 대해 반복 적용할 때 모든 경로들에 대해 상충이 발생한다면 그 고장을 PO까지 전파시키는 경로가 존재하지 않는 것이므로 무해 고장으로 인식할 수 있다. 부입력 할당을 순환적으로 진행시키기 위해 다음의 용어 정의와 정리를 제안한다.

표 2. 부입력 할당을 통한 PFR 인식
Table 2. PFR identification using side input assignment

	고장 전파 게이트	부입력 할당
경로 1	H, K, L	A = 1 B = 0, G = 1 상충 C = 1 I = 1 또는 0/1
경로 2	I, L	D = 1 C = 1, B = 0, H = 1, A = 1, G = 1 상충 K = 1 또는 1/0
경로 3	I, M	D = 1 J = 0 또는 1/0, A = 0, G = 0 상충
경로 4	J, M	A = 0 I = 0 또는 0/1, D = 1, G = 1 상충

정의 7

· 순환 수준(depth) : 고장 영향의 첫 번째 분기점부터 몇 번째 분기점인가를 정의한다.

· 최대 순환 수준(depth_max) : 고려하고자 하는 가장 큰 순환 수준을 정의한다.

· ${}^{d}P = \{ P_1, P_2, P_3 - - - \}$: 순환 수준 d에서 고장 전파가 가능한 모든 경로의 수를 정의한다.

· ${}^{d}P^{k} = \{ f_1, f_2, f_3 - - - \}$: 순환 수준 d의 k번째 경로에 대해 고장 전파를 일으키는 모든 게이 트를 정의한다.

· ${}^{d}C^{k} = \{ G_1 = V_1 , G_2 = V_2 - - - \}$: 순환 수준 d의 k번째 경로에 대해 고장 전파를 가능하게 하는 모든 부입력들의 논리 값 할당을 정의한다.

정리 2 : 고장 영향의 분기점으로부터 최대 순환 수준까지의 경로의 수를 k라 할 때, ${}^{depth\_max}C^{1}$, ${}^{depth\_max}C^{2}$ - - - ${}^{depth_max}C^{k}$가 모두 상충을 나타낼 경우 그 고장은 무해 고장이다.

증명 : 고장 전파를 위한 경로를 a라 할 때, 그 경로에 대해 ${}^{depth_max}P^{a}$가 결정된다. 따라서 이 경로를 통해 고장 전파가 이루어지기 위해서는 모든 ${}^{depth_max}P^{a}$를 만족시켜야 하며, ${}^{depth_max}C^{a}$를 수행할 때 상충이 발생할 경우 경로 a를 통한 고장 전파는 이루어질 수

없다. 이와 같은 과정을 반복적으로 수행하여 최대 순환 수준까지의 모든 $^{depth_max}P$가 상충이 발생 할 때 고장 전파를 위한 경로가 존재하지 않으므로 무해 고장이 된다.

위와 같은 분기점까지의 가능한 경로들에 대해서 부입력들의 논리값을 순환적으로 할당하여 무해 고장을 검출해내는 알고리듬을 유사 지배자 알고리듬이라 한다. 그림 8에 유사 지배자 알고리듬이 나타나 있다. 유사 지배자 알고리듬은 순환 수준이 주어진 최대 순환 수준에 도달 될 때까지 순환적으로 수행되며, 순환 수준은 분기점 도달 시에만 증가하게 된다. 최대 순환 수준 이전에 상충이 발생한다면, 그 경로에 대해 고장 전파가 이루어 질 수 없음을 의미한다. 어떤 한 경로에 대해서 최대 순환 수준까지 할당시 상충이 발생하지 않으면 유사 지배자 알고리듬은 중단된다.

```
pseudo_dom_process(gate_pointer, depth)
{
  if(depth == depth_max)
    return(depth_max);          /* depth_max는 0보다 크거나
                                   같은 양의 정수*/
  flag = assign(gate_pointer);  /* 고장 전파를 위한 비제어값 할
                                   당 */
  if(conflict)
    return(pseudo_redundant);   /* pseudo_redundant = -1 */
  if(fanout > 1)
  {
    depth++;
    for all fanout
      flag = pseudo_dom_process(gate_pointer->no_fanout,
             depth);
  }
  else
      flag = pseudo_dom_process(gate_pointer->no_fanout,
             depth);
  if(fanout > 1)
    depth--;
  return(flag);                 /* flag은 depth 값을 갖게 됨 */
}
```

그림 8. 유사 지배자 알고리듬
Fig. 8. Pseudo dominator algorithm.

그림 9에 최대 순환 수준 2를 사용하여 유사 지배자 알고리듬을 실행하여 검출되는 무해 고장에 대한 예가 있다. 표 3에서 최대 순환 수준이 1일 때는 3개의 경로가 존재하나 상충이 발생하지 않으며 최대 순환 수준을 2로 할 경우 모든 6개의 경로에 대해 상충이 발생하여 무해 고장임을 알 수 있다.

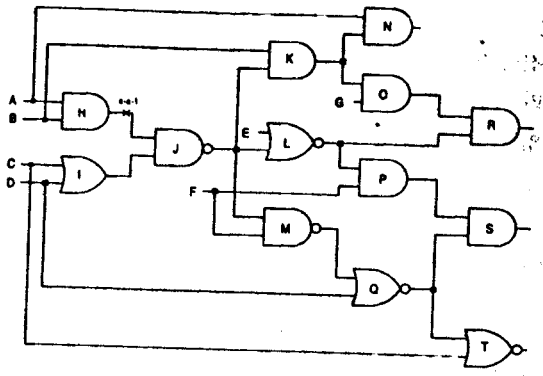

그림 9. 유사 지배자 알고리듬을 이용한 PFR 인식
Fig. 9. PFR identification using pseudo dominator algorithm.

표 3. 유사 지배자 알고리듬을 이용한 PFR 인식 과정
Table 3. Procedure of finding PFR using pseudo dominator algorithm

	최대 순환 수준 1	최대 순환 수준 2
J = 1/0	$^1P^1$ = { P_1, P_2, P_3 } 1. $^1P^1$ = { K } ⇒ $^1C^1$ = { B = 1 } 고장 전파 가능 2. $^1P^2$ = { L } ⇒ $^1C^2$ = { E = 0 } 고장 전파 가능 3. $^1P^3$ = { M, Q } ⇒ $^1C^3$ = { F = 1, D = 0 } 고장 전파 가능	$^2P^1$ = { P_1, P_2, P_3, P_4, P_5, P_6 } 1. $^2P^1$ = { K, N } ⇒ $^2C^1$ = { B = 1, A = 1 } 상충 2. $^2P^2$ = { K, O, R } ⇒ $^2C^2$ = { B = 1, G = 1, L = 1 또는 1/0 } 상충 3. $^2P^3$ = { L, R } ⇒ $^2C^3$ = { E = 0, O = 1 또는 0/1} 상충 4. $^2P^4$ = { L, P, S } ⇒ $^2C^4$ = { E = 0, F = 1, Q = 1 또는 0/1} 상충 5. $^2P^5$ = { M, Q, S } ⇒ $^2C^5$ = { F = 1, D = 0, P = 1 또는 1/0} 상충 6. $^2P^6$ = { M, Q, T } ⇒ $^2C^6$ = { F = 1, D = 0, C = 0 } 상충 모든 경로에 대해 상충 : 무해 고장

3. 테스트 패턴 생성기 구성

본 논문의 테스트 패턴 생성기는 임의 패턴 테스트 생성, 우선 영향(excitation-first) 테스트 패턴 생성, 우선 전파(propagation-first)테스트 패턴 생성등 세 부분으로 구성되었다. 먼저 선행처리 단계에서 회로를 분석하고 테스트 패턴 생성을 용이하게 하는데 필요한 정보를 구한다. 이와 같은 회로 분석과 정보를 토대로 임의 패턴 고장 시뮬레이션을 통해 고장들에 대한 테스트 패턴을 생성해 낸다. 고장 시뮬레이션[12] [16]은 병렬 패턴 단일 고장 전파(parallel pattern single

fault propagation)[15]를 사용하여 임의의 32개 패턴에 대해 동시에 시뮬레이션을 진행한다. 주어진 조건을 넘어서면 임의 패턴 고장 시뮬레이션에 의한 테스트 패턴 생성 과정을 중단하고 남아 있는 고장들에 대해 우선 영향 테스트 패턴 생성을 수행한다. 우선 영향 테스트 생성은 각각의 고장들에 대하여 고장을 삽입하고 고장 삽입에 반드시 필요한 목적들을 만족시키는 입력들을 구한다. 그리고 나서 고장을 전파시키기 위해 임의의 경로를 따라 고장 영향을 출력 단까지 보내는 과정을 모두 만족시킴으로써 삽입한 고장에 대한 테스트 패턴을 구하게 된다.

```
CATPG( )
{
  perform preprocessing
  for random patterns
    {
      perform random pattern fault simulation
        if(not detected new fault for consecutively 4 random fault
          simulation)
          break;
    }
  for all faults left
    {
      apply pseudo dominator algorithm for PFR identification
      perform excitation-first test generation
    }
  for all faults left
    {
      while( objective )
        {
          apply isomorphism identification algorithm for OCD
          identification perform recursive learning
        }
      perform propagation-first test generation
    }
  for all test patterns
    {
      perform fault simulation
      remove already detected faults
      store effective test patterns
    }
}
```

그림 10. 테스트 패턴 생성기 알고리듬
Fig. 10. Automatic test pattern generation algorithm.

본 논문에서 제안한 무해 고장 검출 알고리듬 중 유사 지배자 알고리듬은 우선 영향 테스트 패턴 생성시 적용된다. 마지막으로 우선 전파 테스트 패턴 생성을 수행하는데 이 때 남아 있는 고장들은 검출해 내기 어려운 고장들이므로 제안된 구조 동일성 알고리듬과 간접 유추 과정을 위한 순환 학습[7]을 적용하였다. 순환 학습의 사용은 간접 유추 과정을 반드시 필요한 경우에만 적용하기 위하여 선행처리단계에서 정적 학습을 분석하여 모든 경우에 할당하지 않고 어려운 고장에만 수행하고자 하기 위해서이다. 이는 구조적으로 발생하는 정적 학습이 모든 회로에 대해 항상 효율적인 것은 아니므로, 이를 필요로 하는 검출하기 어려운 고장들에 대해서만 선택적으로 적용함으로써 테스트 패턴 생성시간을 줄이고자 함이다. 이와 같은 무해 고장 알고리듬의 적용은 유사 지배자 알고리듬의 경우 고장 영향 유발과 고장 전파의 순서에 상관없이 선행처리 역할과 같으므로 우선 영향 테스트 패턴 생성시에 사용하였으며, 구조 동일성 알고리듬은 비교할 회로가 커짐에 따라 실행 시간이 크게 증가하므로 검출하기 어려운 최소의 고장들에만 적용하기 위해 우선 전파 테스트 패턴 생성에 적용하였다. 그림 10에 전체 테스트 패턴 생성기를 위한 알고리듬이 있다.

Ⅳ. 실험 결과 및 고찰

본 논문에서 제안한 무해 고장 인식 알고리듬을 적용한 테스트 패턴 생성기를 CATPG라 한다. CATPG는 C 언어를 사용하여 구현하였으며 32 Mbyte의 메모리를 갖는 SPARC-Ⅱ 워크스테이션에서 실험하였다. CATPG의 성능을 비교하기 위해 ISCAS 85 벤치마크 회로에 대해 시도 철회의 수를 100으로 제한하여 테스트 패턴 생성을 수행하였다.

표 4에서는 3개의 알고리듬의 실행결과를 비교하였는데 FAN[3] 알고리듬에 근거한 F는 head line과 다중 후방 추적을 포함하며 SOCRATES[5]에 사용된 알고리듬에 근거한 S는 임의 테스트 패턴 생성과 정적 학습, 고장 우세성[5][6] 알고리듬을 포함하며 C는 CATPG를 나타낸다. 표 4에서 #R, #A, FC는 각각 무해 고장, 중단 고장, 고장 검출율을, CPU는 전체 테스트 패턴 생성시간을 나타낸다. 한다. S와 CATPG에서 임의 테스트 패턴 생성은 연속에서 4회 동안 한 개의 고장도 검출하지 못하면 중단하게 된다. CATPG에서 정적 학습 대신 순환 학습[7]을 사용한 이유는 구조적으로 학습을 찾기 위해 SOCRATES에서 제안한 정적 학습의 선행 처리 단계에서 드는 시간과 불필요한 유추 과정에 의한 시간을 줄이고 어려운 고장들에 대해서만 학습을 이용하기 위해서이다.

표 4. ISCAS 85 회로의 F, S, C 실행 결과
Table 4. Results for F, S, and C using ISCAS 85 benchmark circuits.

회로	고장수	#R			#A			FC(%)			CPU(sec)		
		F	S	C	F	S	C	F	S	C	F	S	C
c432	524	1	2	4	3	2	0	99.24	99.24	99.24	29	2	2
c499	758	8	8	8	78	0	0	88.65	98.94	98.94	126	2	1
c880	942	0	0	0	0	0	0	100	100	100	44	4	3
c1355	1574	8	8	8	184	0	0	87.80	99.49	99.49	686	12	6
c1908	1979	7	9	9	29	0	0	98.08	99.52	99.52	465	18	13
c2670	2747	86	98	117	41	19	0	95.37	95.49	95.49	536	52	28
c3540	3428	137	133	137	12	4	0	95.65	96.00	96.00	787	75	47
c5315	5350	59	59	59	7	1	0	98.76	98.87	98.90	760	38	27
c6288	7744	34	34	34	1352	0	0	82.10	99.56	99.56	15724	65	46
c7552	7550	71	73	131	153	58	0	97.33	98.36	98.36	4120	179	91

실험 결과, CATPG는 모든 중단 고장을 무해 고장으로 인식하였으며 실행 시간도 F 나 S에 비하여 많은 향상을 가져왔다. 비교한 세 테스트 패턴 생성기들에 대해 테스트 패턴 생성시간은 중단 고장의 수와 회로 크기에 비례하였다. 그러나 c6288은 F에서 실행 시간과 중단 고장의 수가 S나 CATPG에 비해 매우 크게 나왔는데, 그 이유는 이 회로가 F에 구현되지 않은 임의 패턴 생성에서 많은 고장들을 검출해 냈기 때문이다. 본 논문에서 제안된 알고리듬들에 의해 CATPG는 S에서 중단 고장으로 인식된 어려운 고장들에 대해 빠르게 무해 고장으로 인식했음을 보여준다. 표 5는 큰 몇몇 회로에 대해 테스트 패턴 생성 과정 중에 발생된 전체 시도 철회의 수를 S와 CATPG에 대해 비교하였다. c2670과 c7552의 경우 시도 철회의 수가 급격히 감소함을 알 수 있다. 이것은 표 3에서 S의 c2670과 c7552에 나타난 많은 중단 고장들이 시도 철회 없이 본 논문에서 제안된 알고리듬들에 의해 무해 고장으로 빠르게 인식되었음을 보여준다. c3540의 경우는 CATPG에서 적은 시간과 더 많은 무해 고장을 인식했음에도 불구하고 시도 철회의 수가 약간 증가하였는데, 그 이유는 S에서 정적 학습을 통해 우선 영향 테스트 패턴 생성에서 검출된 몇몇의 고장들이 CATPG에서는 우선 영향 테스트 패턴 생성에서 검출되지 않고 우선 전파 테스트 패턴 생성에서 검출되어 우선 영향 테스트 패턴 생성 중에 시도 철회의 수를 증가시켰기 때문이다.

표 6은 본 논문에서 제안한 구조 동일성 알고리듬과 유사 지배자 알고리듬을 적용하여 무해 고장으로 인식된 고장들에 대해 회로별로 나타내었다. 특히 유사 지배자 알고리듬의 경우는 선행처리 단계에서 실행되는 것과 같은 역할을 하여 우선 영향 테스트 패턴 생성 이전에 수행되므로 시도 철회를 줄이는데 크게 기여한다. 따라서 표 5에서처럼 c3540이나 c5315에 비해 c2670과 c7552에서 시도 철회의 수가 급격히 감소함을 알 수 있다.

표 5. S와 CATPG의 시도철회 비교
Table 5. Backtrack comparison of S and CATPG

이름	시도철회 수	
	S	C
c2670	4365	1149
c3540	801	1002
c5315	251	63
c7552	12142	342

표 6. 제안된 알고리듬에 의해 검출된 무해 고장의 수
Table 6. Number of identified redundant faults using the proposed algorithms.

이름	검출된 무해 고장 수	
	OCD	PFR
c2670	11	20
c3540	4	0
c5315	0	1
c7552	0	64

표 7. 내부 회로의 크기와 최대 반복 수준
Table 7. Maximum subcircuit size and maximum recursion depth

이름	OCD 인식을 위한 최대 내부 회로 크기 (게이트수)	PFR 인식을 위한 최대 순환 수준(depth_max)
c2670	118	1
c3540	5	-
c5315	-	1
c7552	4	3

※ '-' 는 알고리듬이 적용되지 않음을 나타낸다.

표 7의 최대 내부 회로 크기는 구조 동일성 알고리듬 적용시 각각의 회로에서 사용된 내부 회로들 중 게

이트 수가 가장 큰 것을 나타낸 것이며, 최대 반복 수준은 모든 PFR 무해 인식을 위해 각각의 게이트에서 필요한 반복 수준을 말한다. 표의 결과는 c2670이 게이트수가 118개로 가장 크며 c7552의 최대 반복 수준이 3을 나타내므로 두 알고리듬의 적용이 테스트 패턴 생성시에 많은 영향을 끼치지 않음을 알 수 있다.

V. 결 론

본 논문에서는 새로운 무해 고장 인식을 위한 알고리듬을 사용하여 회로 내에 존재하는 모든 무해 고장을 인식하였으며 이를 적용하여 효율적인 테스트 패턴 생성기를 구성하였다. OCD 인식 을 위한 구조 동일성 알고리듬은 초기 목적들이 서로 의존하여 두 목적을 동시에 만족시킬 수 있는 값이 입력에 존재하지 않음을 입증하여 그 고장이 무해함을 인식하는 것이다. 이를 위해 초기 목적 게이트를 출력으로 하고 두 목적에 공통으로 연결된 분기점을 입력으로 하는 내부 회로를 구성하여 BDD를 적용함으로써 두 회로가 기능적으로 같음을 증명하였다. PFR을 위한 유사 지배자 알고리듬은 고장 전파시 분기점이 없는 경우는 계속해서 그 경로를 따라 진행됨을 이용하여 분기점에 도달했을 경우 각각의 경로에 대해 다음 경로까지 할당해 봄으로써 주어진 분기점에서 모든 경로들에 대해 상충이 발생한 경우를 찾아내어 무해 고장임을 입증한다.

새로운 무해 고장 인식 알고리듬의 제안에도 불구하고 모든 무해 고장을 시도 철회 없이 주어진 초기 목적만 가지고 검출해내지는 못한다. 유사 지배자 알고리듬은 PFR에 의한 모든 무해 고장을 검출할 수 있지만, 구조 동일성 알고리듬은 OCD에 의한 무해 고장 중 두 목적 사이의 의존 관계와 정적인 구조에 대해서만 고려하였다. 따라서 초기 목적에 따라 동적으로 발생하는 무해 고장의 인식과 여러 목적 사이에 발생하는 상호 의존에 관한 알고리듬의 연구가 더욱 필요할 것이다. 이와 같은 연구를 통해 모든 무해 고장을 시도 철회 없이 초기 목적만 가지고 검출해 냄으로써 고성능의 테스트 패턴 생성기를 구현할 수 있을 것이다.

감사의 글

※ 이 논문은 1994년도 학술 진흥 재단의 대학 부설 연구소 연구 과제 연구비에 의해 수행되었으며 이에 감사드립니다.

참 고 문 헌

[1] J. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", IBM Journal, July, 1966 pp. 278~291.

[2] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Transaction on Computers, vol. C-30, March 1981, pp. 215~222.

[3] H, Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms", IEEE Transactions on Computers, vol. C-32, no. 12, Dec, 1983, pp. 1137~1144.

[4] J. Silva and K. Sakallah, "Dynamic Search-Space Pruning Techniques in Path Sensitization", Proc. of Design Automation Conference, 1994, pp. 705~711.

[5] M. Schulz, E. Trischler and T. Sarfet, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System", Proc. of International Test Conference, 1987, pp. 1016~1026.

[6] M. Schulz and E. Auth, "Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification", IEEE Transactions on CAD, vol. 8, no. 7, July 1989, pp. 811~816.

[7] W. Kunz et al., "Recursive Learning : An attractive alternative to the decision tree for test generation in digital circuits", Proc. of International Test Conference, 1992. pp. 816~825.

[8] M. Termoto, "A Method for Reducing the Search Space in Test Pattern Generation" Proc. of International Test Conference, 1993, pp. 429~435.

[9] L. Goldstein, "Controllability/Observability Analysis of Digital Circuits", IEEE Transactions on Circuits and Systems, vol. CAS-26, Sept. 1979, pp. 685~695.

[10] L. Goldstein and E. Thigpen, "SCOAP: Sandia Controllability/Observability An-

alysis Program", Proc. of Design Automation Conference, pp. 190~196, June 1980.

[11] M. Abramovici et al., "One-pass Redundancy Identification and Removal", Proc. of International Test Conference, 1992, pp. 807~815.

[12] J. Waicukauski, P. Shupe, D. Giramma and A. Matin, "ATPG for Ultra-Large Structured Designs", Proc. of International Test Conference, 1990, pp. 44~51.

[13] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", Proc. of International Symposium on Circuits and Systems, 1885, pp. 695~698.

[14] K. Brace. R. Rudell and R. Bryant, "Efficient Implementation of a BDD Package", Proc. of IEEE Design Automation Conference, 1990, pp. 40~45.

[15] M. Abramovici, M. Breuer, A. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, New York, 1990, p28, μ.156, pp. 343~346.

[16] G. Indrajo and L. Wang, "Fault Simulation Rates Effectiveness of Test Patterns", EDN, 1986, pp. 181~188.

저 자 소 개

姜 成 昊(正會員)

1986년 2월 서울대학교 제어계측공학과 졸업(공학사). 1988년 5월 The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학석사). 1992년 5월 The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학박사). 1989년 11월 ~ 1992년 8월 미국 Schlumberger Inc. Research Scientist. 1992년 9월 ~ 1992년 10월 미국 The Univ. of Texas at Austin Post Doctoral Fellow. 1992년 8월 ~ 1994년 6월 미국 Motorola Inc. Senior Staff Engineer. 1994년 9월 ~ 현재 연세대학교 전기공학과 조교수. 주관심분야는 VLSI CAD, 테스팅. 테스트를 고려한 설계, VLSI설계

韓 相 潤(正會員)

1995년 2월 연세대학교 전기공학과(학사). 1997년 2월 연세대학교 전기공학과(석사). 1997년 3월 ~ 현재 LG 반도체 연구원. 주관심분야는 VLSI & CAD, VLSI Testing 등