論文2001-38SD-4-6

CMOS VLSI의 효율적인 IDDQ 테스트 생성을 위한 패턴 생성기의 구현

(Implementation of Pattern Generator for Efficient IDDQ Test Generation in CMOS VLSI)

裵 晟 桓 *, 金 觀 雄 **, 田 炳 實 **

(Sung Hwan Bae, Kwan Woong Kim, and Byoung Sil Chon)

요 약

IDDQ 테스트는 CMOS VLSI 회로에서 발생 가능한 여러 종류의 물리적 결함을 효율적으로 검출 할 수 있는 테스트 방식이다. 본 논문에서는 CMOS에서 발생 빈도가 가장 높은 합선고장을 효과적으로 검출할 수 있는 IDDQ 테스트 알고리즘을 이용하여 패턴 생성기를 개발하였다. 고려한 합선고장 모델은 회로의 레이아웃 정보에 의존하지 않으며, 내부노드 혹은 외부노드에 한정시킨 합선고장이 아닌 테스트 대상회로의 모든 노드에서 발생 가능한 단락이다. 구현된 테스트 패턴 생성기는 O(n2)의 복잡도를 갖는 합선고장과 전압 테스트 방식에 비해 상대적으로 느린 IDDQ 테스트를 위해서 새롭게 제안한 이웃 조사 알고리즘과 고장 collapsing 알고리즘을 이용하여, 빠른 고장 시뮬레이션 시간과 높은 고장 검출율을 유지하면서 적은 수의 테스트 패턴 생성이 가능하다. ISCAS 벤치마크 회로의 모의실험을 통하여 기존의 다른 방식보다 우수한 성능을 보였다.

Abstract

IDDQ Testing is a very effective testing method to detect many kinds of physical defects occurred in CMOS VLSI circuits. In this paper, we consider the most commonly occurring bridging faults in current CMOS technologies and develop pattern generator for IDDQ testing using efficient IDDQ test algorithms. The complete set of bridging faults between every pair of all nodes(internal and external nodes) within circuit under test is assumed as target fault model. The merit of considering the complete bridging fault set is that layout information is not necessary. Implemented test pattern generator uses a new neighbor searching algorithm and fault collapsing schemes to achieve fast run time, high fault coverage, and compact test sets. Experimental results for ISCAS benchmark circuits demonstrate higher efficiency than those of previous methods.

* 正會員, 漢麗大學校 멀티미디어情報通信工學科
(Dept. of Multimedia, Information and Telecommunication Eng., Hanlyo Univ.)
** 正會員, 全北大學校 電子工學科
(Dept. of Electronic Eng., Chonbuk Nat'l Univ.).
接受日字:2000年7月20日, 수정완료일:2001年3月26日

I. 서 론

최근에는 산업 및 인간 생활에 직·간접적으로 영향을 주고 있는 VLSI 회로에 대한 높은 품질 및 신뢰성에 강한 요구가 있으며 이러한 경향은 매우 높은 레벨의 테스트가 필요하게 되었다[1]. 합선고장(Bridging Fault)은 두선 혹은 그 이상의 선이 서로 의도하지 않

게 연결되는 고장형태이며 CMOS VLSI 회로에서 발생 빈도가 가장 높은 고장이다. 또한, 단일 고착고장으로는 모델링되지 못하는 많은 결함을 포함하고 있으며, 실제 CMOS 결함의 40~50% 정도가 합선고장을 이용하여 효과적으로 모델링 될 수 있다. 따라서 CMOS에서 발생 빈도가 가장 높은 합선고장을 효과적으로 검출할 수 있는 테스트 기법이 필요하다[2].

IDDQ테스트(quiescent power supply current)는 CMOS에서 발생 가능한 여러 종류의 물리적 결함을 효율적으로 검출 할 수 있는 테스트 방식이다[1,3]. CMOS 게이트는 nMOS 풀다운 블록과 pMOS 풀업 블록으로 이루어져 있다. 고장이 없는 회로에서는 임의의 입력상태에 대하여 출력단이 VDD 또는 GND 중 한 노드만으로 연결되어 VDD와 GND 사이에 전류의 도통 경로가 형성되지 않는다. 따라서 이러한 정적 상태(steady state) 동안에는 수 nA정도의 무시할 만한 누설 전류만이 흐르게 된다. 그러나, 게이트 옥사이드 단락, 합선 결함, 기생 트랜지스터 누설, 누설 PN 결합, 개방 결함, 그리고 전송 게이트의 개방 등과 같은 물리적 결함은 CMOS로 이루어진 회로의 정적 상태 전류를 증가시켜 많은 전력손실을 발생시키며 회로의 신뢰성을 감소시킨다.

IDDQ 테스트를 통하여 두 선 l_1과 l_2 사이에 발생한 합선고장을 검출하는 경우, l_1은 "1" 논리 값을 그리고 l_2는 "0" 논리 값을 갖도록 (혹은 그 반대로)해 준다. 따라서 두 선이 합선된 경우 VDD와 GND 사이의 전류경로가 형성되어 회로의 정적상태에서도 많은 양의 전류가 흐르게 되어 가정한 고장을 검출할 수 있다[4]. 합선고장은 테스트 회로의 노드수가 n인 경우, 발생 가능한 총 고장의 수는 $O(n^2)$의 복잡도를 가지게 되어 회로의 집적도가 증가함에 따라 고려할 고장의 수가 매우 많게 된다. Jee와 Ferguson은 전체 합선고장 수를 줄이기 위해서 IFA(Inductive Fault Analysis) 방식을 사용하여 CMOS 회로의 결함을 해석하였다[5]. IFA 방식은 회로의 고장을 레이아웃 레벨에서 회로레벨로 변환시키는 추출과정이 필요하므로 테스트 회로의 레이아웃에 관한 자세한 정보가 요구된다. 그러나 회로의 레이아웃 정보는 항상 유용하지 않으며, 정보가 유용해도 집적도가 큰 회로의 경우에는 레이아웃 정보를 해석하여 고장을 추출하는 과정은 많은 시간과 메모리의 사용이 필요하다[6~7].

따라서 레이아웃에 관한 정보에 의존하지 않고, 일반적으로 적용 가능한 합선고장 모델링 기법이 필요하다. 이 경우 일반적인 합선고장 모델링은 두 가지에 문제점이 존재하게 된다. 첫째, 합선고장은 $O(n^2)$의 복잡도를 가지게 되어 테스트 패턴 생성기(test pattern generator)를 이용하여 고장을 검출하기 위해서는 전체 고장을 표현하는 기법과 무해고장(redundant fault)을 효율적으로 찾아내는 알고리즘이 필요하다. 둘째, IDDQ 테스트 방식은 높은 고장 및 결함 검출률을 갖지만 상대적으로 전압 테스트 방식에 비해 느린 테스트 시간을 가진다. 따라서 높은 고장을 유지하면서 가능한 적은 수의 테스트 패턴을 얻어야 한다. 결함이 발생하는 위치에 따라 다음과 같은 모델링이 가능하다[4]. 금속층에 발생 가능한 결함을 고려할 경우에는 외부 합선고장(external bridging fault)으로 모델링이 가능하며[7, 8], 게이트 내부의 트랜지스터들의 결함을 고려할 경우는 내부 합선고장(internal bridging fault)으로 모델링이 가능하다[9, 10]. 또한, 스위치 레벨의 모든 노드들 사이에 발생 가능한 결함은 전체 합선고장(all bridging fault)을 이용하여 모델링 할 수 있다[4, 6].

본 논문에서는 회로의 레이아웃 정보에 의존하지 않고 내부노드 혹은 외부노드에 한정시킨 합선고장이 아닌 테스트 대상회로의 모든 노드에서 발생 가능한 단락을 고려하였다. $O(n^2)$의 복잡도를 가지는 합선고장 모델과 전압 테스트 방식에 비해 상대적으로 느린 IDDQ 테스트를 위해서 고장 시뮬레이션 수행시간이 짧고, 높은 고장 검출률을 유지하면서 적은 수의 테스트 패턴의 생성이 가능하도록 이웃 조사 알고리즘과 고장 collapsing 알고리즘을 제안한다. 본 논문은 2장에서 고장 모델과 제안된 고장 시뮬레이션 알고리즘에 관해서 논의하고, 3장에서는 개선된 고장 collapsing 기술을 제안한다. 또한, 4장에서는 구현된 테스트 패턴 생성기에 대한 모의 실험결과를 검토하고 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 고장 모델 및 고장 시뮬레이션 알고리즘

1. 고장 모델

가정한 고장모델은 회로를 구성하는 모든 게이트들의 외부와 내부 노드에서 발생 가능한 단락을 고려한 전체 합선고장이다[4, 6]. 그림 1에는 2-입력 NAND 게

이트의 내부 노드 d와 인버터의 출력단 k사이에 발생한 합선고장의 예를 보인다.

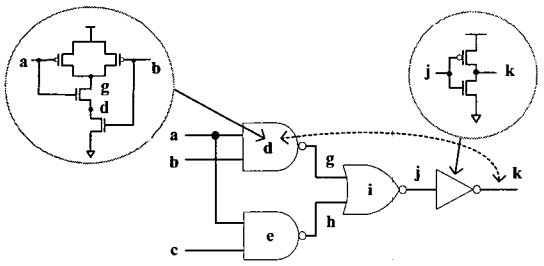


그림 1. 내부 노드와 외부 노드 사이의 전체 합선고장 예
Fig. 1. Illustration of all bridging fault between internal and external nodes.

그림 1의 예에서와 같이 전체 합선고장 모델은 테스트 대상 회로를 게이트 레벨이 아닌 스위치 레벨로 해석해야한다. 따라서, 발생 가능한 합선고장의 수가 기존의 외부노드 사이에서 발생하는 합선고장모델[7, 8]이나 내부노드 사이에서 발생하는 합선고장모델[9, 10]에 비하여 매우 크기 때문에, 효과적으로 무해고장을 검출하는 고장 collapsing 알고리즘과 빠른 고장 시뮬레이션 알고리즘이 필요하다. 테스트 회로에서 합선고장이 발생한 경우, IDDQ 테스트는 합선이 발생한 두 선 l_1과 l_2 사이에 l_1은 "1" 논리 값을 그리고 l_2는 "0" 논리 값을 갖도록 (혹은 그 반대로)해 준다면 회로의 정적 상태에서도 VDD와 GND 사이에 전류의 경로가 생성되어 회로의 정적상태에서도 많은 양의 전류가 흐르게 되어 가정한 고장을 검출할 수 있다. 전체 합선고장 모델에서 노드 x, y 사이에 발생한 합선고장 검출은 다음 두 가지의 조건을 만족해야 한다[4].

< 조 건 >

i) T(x) ≠ T(y), ii) T(x) ≠ f 그리고 T(y) ≠ f
(T(k)) : 테스트 벡터 T를 입력 단에 인가할 때 노드 k의 값, f : 부동값(floating value))

예를 들어 그림 1의 회로에 테스트 벡터 T = "000"을 입력 단에 인가하면, 단락이 발생한 두 노드 {d, k}는 T(d) = f, T(k) = 1이 되어 고장 검출이 불가능하다. 그러나 테스트 벡터 T = "110"의 경우에는 T(d) = 0, T(k) = 1의 값을 가지게 되어 고장을 검출할 수 있다. 일반적으로 부동값 f의 존재로 인하여 고장 검출이 어려운 합선고장을 PDF(Potentially Detected Faults)라고 한다. 전체 합선고장 모델에서는 많은 PDF가 존재하여, 내부 합선고장 모델이나 외부 합선고장 모델과는 다른 표현기법을 이용한 고장 시뮬레이션 알고리즘이 필요하다. 기존의 트리구조 알고리즘은 각 테스트 벡터의 고장 검출률을 계산하기 위해서는 다음과 같은 정의를 이용한다[4].

- NP(Node partitions)
  : 임의의 테스트 벡터에 항상 같은 값을 가지는 노드 그룹, 기호 : { }
- IP(Indistinguishable pairs)
  : 임의의 테스트 벡터에 PDF를 포함하여 항상 같은 값을 가지게 되는 NP 그룹쌍, 기호 : < >

기본적인 IDDQ 테스트 원리를 응용하여, 임의의 테스트 패턴에 같은 값을 같는 노드별로 등가그룹을 할당하게 되면 다음의 정의와 식을 이용하여 전체 합선고장 모델에서 고장 검출률의 계산이 가능하다.

- 전체 합선고장 수 (A) = ${}_nC_2 = n(n-1)/2$ (1)
(n : 테스트 대상회로를 구성하는 게이트의 내부와 외부의 총 노드 수)
- 검출하지 못한 합선고장 수 (B) = (C) + (D)
  · NP의 고장 수 (C) = $\sum_i \frac{|E_i| * (|E_i|-1)}{2}$
  ($|E_i|$: i번째 등가그룹의 노드 수) (2)
  · IP의 고장 수 (D) = $\sum_{i,j} |E_i| * |E_j|$ (3)
(임의의 테스트 벡터에 PDF를 포함하여 항상 같은 값을 가지게 되는 NP 그룹쌍)
- 고장 검출률(%)= $\frac{(A)-(B)}{(A)} \times 100$ (4)

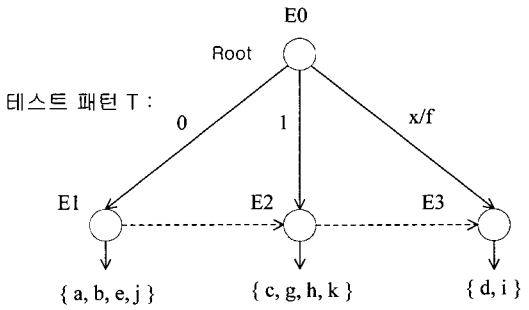


그림 2. 등가그룹의 추출
Fig. 2. Extracting equivalent groups.

그림 2는 임의의 테스트 패턴 T = "001"을 그림 1의 테스트 회로의 입력노드에 인가하여 등가노드 그룹을 생성시키는 예를 보인다.

- 전체 NP = { E1, E2, E3 } = { { a, b, e, j }, { c, g, h, k }, { d, i } }
- 전체 IP = { < E1, E3 >, < E2, E3 > }
  = { < { a, b, e, j }, { d, i } >, < { c, g, h, k }, { d, i } > }

수식 (1)~(4)을 이용하여 테스트 패턴 T의 고장 검출률을 계산할 수 있다.

- 전체 합선고장의 수 (A) = nC2 = n(n - 1)/2 = (10×9)/2 = 45
- 검출하지 못한 합선 고장 수 (B) = (C) + (D) = 29
  - NP의 고장 수 (C) = E1(E1 - 1)/2 + E2(E2 - 1)/2 + E3(E3 - 1)/2 = 6 + 6 + 1 = 13
  - IP의 고장 수 (D) = E1×E3 + E2×E3 = 8 + 8 = 16
- 고장 검출률 = $\frac{(A)-(B)}{(A)} \times 100$ = (45 - 29)/45 × 100 = 35.56%

트리구조 알고리즘의 특성을 설명하기 위해서 그림 3에 연속적인 3개의 테스트 패턴을 적용 한 후의 트리 구조의 성장을 보였다. 테스트 패턴 T3을 입력노드에 인가한 후의 트리 구조 알고리즘은 NP와 IP를 계산하기 위해 먼저 평행선에 있는 주변노드를 탐색하게 된다. NP E8은 IP를 구하기 위해서 E9~E12의 NP그룹과 비교를 시작한다. NP E8은 NP E11, E12와는 T3 테스트 패턴으로 구별이 불가능하여 위 경로로 탐색방향을 바꾼다. T2 테스트 패턴에서는 NP E11과 구별이 가능하지만, 아직도 NP E12와는 구별이 불가능하여 한 단계 위로 탐색과정을 바꾼다. T1 테스트 패턴에서도 NP E12와는 구별이 불가능하게 된다. 따라서 모든 전체 경로를 탐색하여도 E12와는 구별이 불가능하게 됨으로 < E8, E12 >는 IP그룹에 추가된다. 위와 같은 과정을 NP E9 ~ E12에 반복 수행하면 그림 3의 트리 구조의 경우에 고장 검출률은 다음과 같다.

- 전체 NP = { E8, E9, E10 } = { { a, e }, { c, d }, { f, g } }
- 전체 IP = { < E8, E12 >, < E11, E12 > } = { < { a, e }, { h } >, < { b }, { h } > }

수식 (1) ~ (4)을 이용하여 고장 검출률을 계산할 수 있다.

- 전체 합선고장의 수 (A) = $_nC_2$ = n(n - 1)/2 = (8×7)/2 = 28
- 검출하지 못한 합선 고장 수 (B) = (C) + (D) = 6
  - NP의 고장 수 (C) = E8(E8 - 1)/2 + E9(E9 - 1)/2 + E10(E10 - 1)/2 = 1 + 1 + 1 = 3
  - IP의 고장 수 (D) = E8×E12 + E11×E12 = 2 + 1 = 3
- 고장 검출률 = $\frac{(A)-(B)}{(A)} \times 100$ = (28 - 6)/28 × 100 = 78.57%

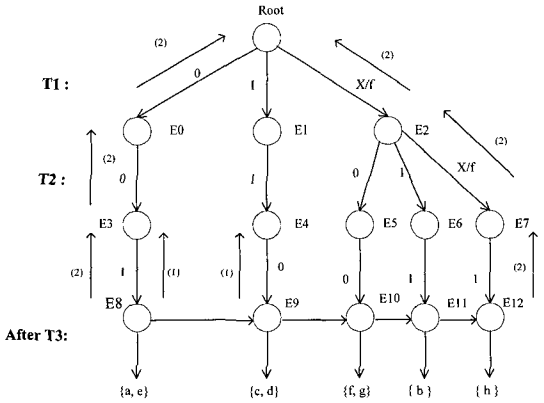


그림 3. 트리 구조를 이용 NP와 IP 추출 알고리즘
Fig. 3. Extracting algorithm for NPs and IPs from tree structure.

위의 예를 통해서 전체 합선고장 모델에서 트리 구조 알고리즘을 이용하여 고장 시뮬레이션을 수행하기에는 너무 많은 NP와 IP의 비교 과정을 필요로 한다. 따라서 VLSI 회로의 경우에는 트리 구조의 단계가 진행될 수록 엄청난 양의 고장 시뮬레이션 시간과 메모리의 사용을 요구하게 된다. 본 논문에서는 고장 시뮬레이션 시간과 메모리의 사용을 효율적으로 줄이기 위한 이웃조사 알고리즘 방식을 제안한다.

2. 제안된 고장 시뮬레이션 알고리즘

제안된 알고리즘은 그림 4에 보인 바와 같이 각 그룹은 그룹의 번호, 그룹의 현재 값, 그룹에 속해 있는 노드, 이웃 그룹과의 관계를 나타내는 4개의 정보로 구성된다.

그림 5에 임의의 테스트 패턴이 초기 노드에 인가될 경우 그룹 #1의 분해 과정을 보인다. 첫 번째 테스트 패턴이 입력 노드에 인가되면, 초기의 각 노드는 "0",

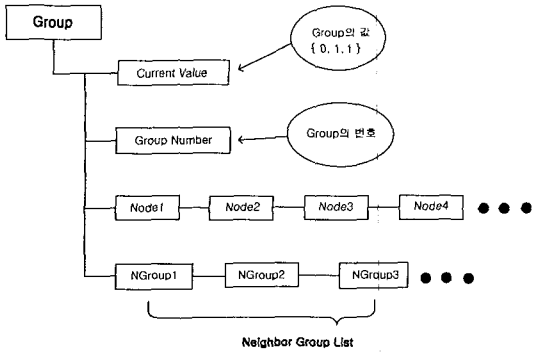


그림 4. 이웃 조사 알고리즘의 그룹 구조
Fig. 4. Group structure of neighbor searching algorithm.

"1", "f" 중에 하나의 값을 가지게 되어 각각 그룹 #1, #2, #3으로 분해된다. 그룹 #1의 현재 값은 "f"로 할당되고 이웃 노드와의 관계를 조사하기 위해서 그림 5에서와 같이 그룹 #2, #3에 자신의 그룹 번호를 기록하고 자신의 이웃 그룹 리스트에는 #2와 #3을 기록한다.

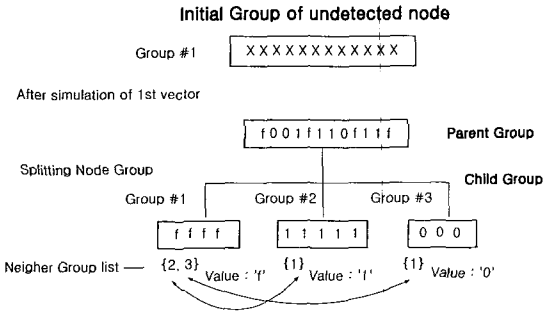


그림 5. 이웃 조사 알고리즘의 노드 분할 과정(1)
Fig. 5. Node split processing(1) of neighbor searching algorithm.

두 번째 테스트 패턴이 입력노드에 인가되면 이웃조사 알고리즘은 그림 6과 같은 작업을 수행하게 된다. 그룹 #1, #2, #3의 각 노드는 새로운 입력 값에 따라서 "f", "1", "0"중에 하나의 값의 가지게 된다. 먼저 그룹 #1은 "f", "1", "0"에 따라서 새로운 그룹 #1, #4, #5가 되고, 자신의 이웃노드 리스트에 기록되어 있는 그룹 #2와 #3을 찾아가 새로운 정보를 기록한다. 따라서 그룹 #2와 #3의 이웃노드 리스트는 {1, 4, 5}로 바뀌게 된다. 다음에는 그룹 #1 #4, #5간의 분해 과정을 하게 된다. 이 단계를 거친 후 그룹 #1 #4, #5의 이웃노드 리스트는 각각 {2, 3, 4, 5}, {2, 3, 1}, {2, 3, 1 }로 새롭게 갱신된다. 이러한 노드 분할의 과정은 그룹 #2, #3에 대해서 차례로 진행되어 앞의 과정을 반복 처리하면 다음의 입력 테스트 패턴을 받게 된다.

제안된 이웃노드 조사 알고리즘은 기존의 트리 구조 알고리즘에 비해서 다음과 같은 특징을 가진다. 트리 구조 알고리즘에서는 테스트 패턴이 인가된 후 현재 입력 테스트 벡터에 대해서 위 경로를 포함하는 모든 노드에 대한 비교 과정을 거치게 되며 VLSI 회로의 경우에는 트리 구조의 단계가 진행될수록 더욱 많은 비교과정이 필요하게 되어 많은 양의 고장 시뮬레이션 시간과 메모리의 사용을 요구하게 된다. 그러나 이웃조사 알고리즘은 현재 상태에서 이웃 노드와의 관계를 비교하여, 자신의 그룹노드에 속하는 노드의 수가 1개인 경우와 자신의 이웃 노드 리스트가 비어있는 경우에 전체 노드에서 제거할 수 있는 장점이 있다. 이는 VLSI 회로와 같이 발생 가능한 전체 합선고장의 수가 큰 경우에 위 경로로의 조사과정이 필요없고 또한, 다른 노드와 구별이 된 노드를 중간에서 제거할 수 있는 장점을 가지므로 빠른 시간에 고장 시뮬레이션이 가능하다.

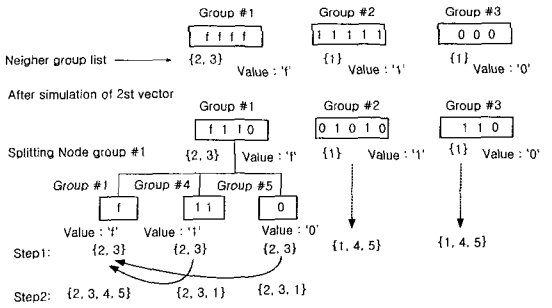


그림 6. 이웃 조사 알고리즘의 노드 분할 과정(2)
Fig. 6. Node split processing(2) of neighbor searching algorithm.

Ⅲ. 제안된 고장 collapsing 알고리즘

두 개의 합선된 노드가 항상 같은 논리 값을 갖는 경우, 두 단락된 노드에 "1"과 "0"값의 적용이 불가능하게 된다. 이러한 합선고장은 모든 테스트 패턴으로도 검출이 불가능하여 무해고장으로 부르며, 무해고장을 효과적으로 검출하는 기술을 고장 collapsing이라 부른다[4, 6]. 따라서 효율적인 고장 collapsing 알고리즘은 IDDQ 테스트로 검출하지 못하는 노드를 삭제함으로써,

고장 시뮬레이션과 테스트 패턴 생성에 요구되는 시간을 줄여준다.

본 논문에서 고려한 전체 합선고장의 고장 collapsing 방법은, 기존의 논문에서 제안한 고장 collapsing 알고리즘[4, 6]을 기본으로 하고 다음과 같은 기법이 새로이 추가된 고장 collapsing 알고리즘이다. 버퍼, AND, OR 게이트의 경우는 NAND, NOR, 인버터 등의 게이트로 분해하는 방식을 사용한다. 실제로 버퍼, AND, OR 게이트는 분해가 불가능하지만, 모든 노드를 고려하는 전체 합선고장을 시뮬레이션하는 경우에 스위치레벨의 분석이 필요하기 때문에 그림 7의 예와 같이 AND 게이트는 NAND와 인버터의 구조로 분석 할 수 있다. 기존의 고장 collapsing 방식에서는 AND 게이트 G0과 인버터 G1은 서로 값이 달라 collapsing이 불가능하다. 그러나 AND 게이트를 NAND 게이트와 인버터로 분해했을 경우에 NAND 게이트 -G0와 인버터 G1은 등가노드가 되어 고장 collapsing이 가능해진다.

논문에서 새롭게 제안한 고장 collapsing 알고리즘과 기존의 알고리즘과의 비교 예를 그림 8에서 보인다. 그림 8(b)는 제안된 알고리즘을 이용할 경우 3개의 추가적인 등가노드를 검출하는 것을 보여준다. 따라서 제안된 방식이 더 많은 수의 등가노드를 효율적으로 검출함을 알 수 있다. 제안된 고장 collapsing 방법을 이용하여 ISCAS '85 벤치마크 회로에 적용한 결과를 표 1에 보인다. 제안된 알고리즘이 효과적으로 많은 등가노드를 검출 할 수 있음을 알 수 있다.

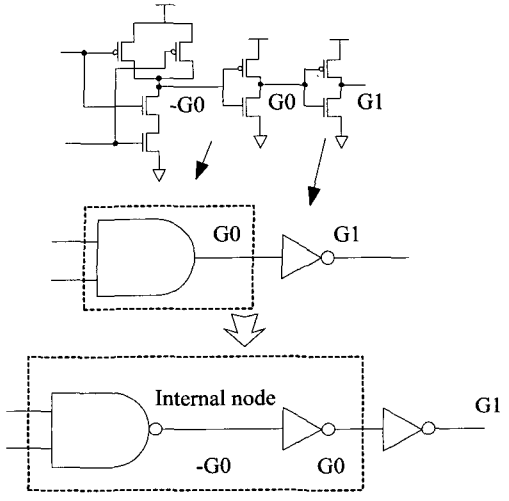
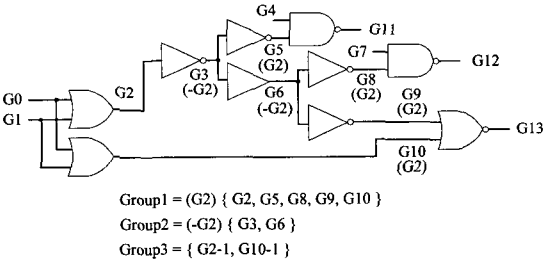


그림 7. 제안된 전체 노드의 고장 collapsing
Fig. 7. Proposed fault collapsing for all nodes.

표 1. 전체 노드의 고장 collapsing 결과
Table 1. Fault collapsing results of all nodes.

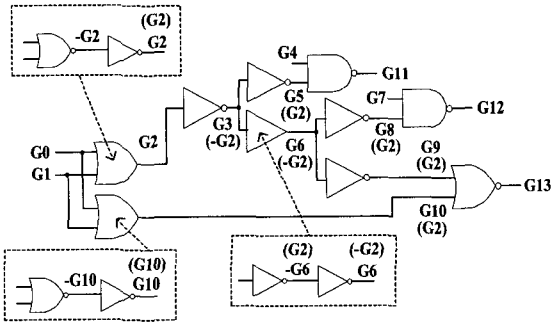
회 로	총 합선고장 수	Thadikaran[4]	Terry Lee*[6]	제안한 방법
c432	108,345	106,030	74,305	75,305
c499	526,851	494,515	271,953	167,331
c880	436,645	-	323,610	374,545
c1355	675,703	603,351	593,505	639,015
c1908	1,540,890	832,695	634,501	459,361
c2670	3,904,615	2,907,666	2,168,403	1,842,240
c3540	6,402,831	4,235,505	3,777,126	3,081,403
c5315	15,100,260	11,681,361	10,490,490	8,994,289
c6288	12,941,328	12,860,056	12,622,800	12,477,510
c7552	27,162,135	19,615,716	17,508,403	13,873,278

*Terry Lee 경우는 전체노드에 입력노드는 포함하지 않음



Group1 = (G2) { G2, G5, G8, G9, G10 }
Group2 = (-G2) { G3, G6 }
Group3 = { G2-1, G10-1 }

(a)



Group1 = (G2) { G2, G5, -G6, G8, G9, G10 }
Group2 = (-G2) { -G2, -G10, G3, G6, G13 }
Group3 = { G2-1, G10-1 }

(b)

그림 8. 전체 노드의 고장 collapsing 방법의 비교
Fig. 8. Comparison of two fault collapsing methods for all nodes.

Ⅳ. 테스트 패턴 생성기의 모의 실험 및 검토

개발한 테스트 패턴 생성기는 PC의 Windows기반에서 Visual Basic 소프트웨어를 이용하여 사용자의 편의를 위해 GUI 방식으로 구현하였다. 조합회로로 구성되어 있는 ISCAS '85 벤치마크 회로에 대해 제안된 알고리즘을 이용하여 구현된 테스트 패턴 생성기의 성능 평가를 표 2에 보인다. 효율성 평가를 위해서 고장 검출률, 생성 테스트 패턴의 개수, 테스트 패턴 생성 시간을 기존의 방식과 비교하였다. 구현된 테스트 패턴생성기는 입력 탐색방식을 적용하여 회로의 모든 소자가 아닌 입력단자의 값이 변하는 소자만 연산하는 기법을 적용하였고, 고장 시뮬레이션을 위해 입력단에 인가된 테스트 패턴은 무작위로 선택된 임의의 패턴을 한 비트만 역 변환하여 연속적으로 적용함으로, 테스트 대상 회로의 입력노드에 인가되는 테스트 패턴이 단자 1-비트 변화가 발생하여 값이 변하는 소자의 수를 급격히 줄여 빠른 시뮬레이션 수행이 가능하다.

표 2. ISCAS '85 벤치마크 회로에 대한 시뮬레이션 결과
Table 2. Simulation results for ISCAS '85 benchmark circuits.

회 로	테스트 패턴 수	검출하지 못한 합선고장	고장 검출률 (%)	CPU time (second)
c432	22	212	99.72	11
c499	34	292	99.83	51
c880	28	128	99.97	65
c1355	67	216	99.97	188
c1908	55	54	99.99	43
c2670	30	341	99.98	777
c3540	66	298	99.99	628
c5315	40	753	99.99	2,263
c6288	32	831	99.99	799
c7552	54	2610	99.98	6,331

구현한 테스트 패턴 생성기의 고장 검출률을 기존에 발표된 방식과 비교하여 표 3에 보였다. c1355를 제외한 모든 ISCAS '85 벤치마크 회로에서 기존의 구현한 방식보다 높은 고장 검출률을 보이고 있다. 표 3과 4를 통하여, 제안한 알고리즘을 이용한 테스트 패턴생성기는 기존의 제안한 방식에 비해서 상당히 감소한 개수의 테스트 패턴을 생성하여 높은 고장 검출률을 얻을 수 있음을 확인 할 수 있다.

표 3. 고장 검출률 비교(%)
Table 3. Comparison of fault coverage(%).

회 로	제안한 방법	Thadikaran[4]	Terry Lee[6]
c432	99.72	98.88	99.69
c499	99.83	99.72	99.80
c880	99.97	-	99.89
c1355	99.97	99.98	99.92
c1908	99.99	99.72	99.79
c2670	99.98	99.82	99.80
c3540	99.99	99.86	99.90
c5315	99.99	99.92	99.93
c6288	99.99	99.97	99.98
c7552	99.98	99.89	99.74

표 4. 생성된 테스트 패턴 개수의 비교
Table 4. Comparison of number of generated test patterns.

회 로	제안한 방법	Thadikaran[4]	Terry Lee[6]
c432	22	32	65
c499	34	170	216
c880	28	-	84
c1355	67	130	196
c1908	55	190	349
c2670	30	64	162
c3540	66	146	260
c5315	40	93	208
c6288	32	56	51
c7552	54	226	880

이러한 결과는 이웃 조사 알고리즘을 이용하여, 가속화된 고장 시뮬레이션을 통해 많은 테스트 패턴의 적용이 가능하였고, 또한 새롭게 제안한 고장 collapsing 알고리즘을 이용하여 무해고장 수를 효과적으로 감소시킨 결과이다.

테스트 패턴을 생성하는데 소요되는 시간을 표 5에 보인다. Thadikaran의 경우는 SUN SPARC-2 워크스테이션을 사용하였고, Terry Lee는 SUN SPARC 5/70

표 5. 테스트 패턴 생성 시간의 비교(초)
Table 5. Comparison of test pattern generation time(sec.).

회로	Thadikaran[4] (SUN SPARC-2)	Terry Lee[6] (SUN SPARC 5/70)	제안한 방법 (Pentium-pro 400MHz)
c432	24	85	11
c499	300	582	51
c880	-	230	65
c1355	1,320	597	188
c1908	4,320	1,710	43
c2670	2,940	793	777
c3540	7,800	3,320	628
c5315	7,248	3,040	2,263
c6288	2,580	800	799
c7552	121,320	21,500	6,331

표 6. 10, 20, 30번째 패턴에서의 고장 검출률 비교(%)
Table 6. Fault coverage comparison of 10th, 20th and 30th test pattern(%).

회 로	테스트 패턴 수		고장 검출률							
			10개 패턴		20개 패턴		30개 패턴		전 체	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
c432	22	65	99.49	97.64	99.69	99.40	-	99.61	99.72	99.69
c499	34	216	97.82	93.89	99.27	97.66	99.79	98.60	99.83	99.79
c880	28	84	99.62	98.01	99.95	99.62	-	99.79	99.97	99.89
c1355	67	196	96.63	94.98	98.45	97.76	99.34	98.72	99.97	99.92
c1908	55	349	99.07	93.59	99.79	97.36	99.95	98.48	99.99	99.79
c2670	30	162	99.44	96.96	99.96	99.31	99.98	99.63	99.99	99.80
c3540	66	260	98.22	96.45	99.80	99.18	99.95	99.63	99.99	99.90
c5315	40	208	99.42	96.99	99.97	99.50	99.99	99.81	99.99	99.93
c6288	32	51	99.58	99.62	99.99	99.97	99.99	99.98	99.99	99.98
c7552	54	880	99.35	96.01	99.94	98.42	99.97	98.90	99.98	99.74

(1) 제안한 방법 (2) Terry Lee[6]

을, 본 논문에서는 Pentium-pro 400MHz를 사용하여 서로 사용한 하드웨어 플랫폼이 다르기 때문에 정확한 성능 비교는 어렵지만, ISCAS '85 벤치마크 회로를 통한 시뮬레이션 결과, 모든 회로에서 빠른 시간에 테스트 패턴을 생성이 가능함을 확인할 수 있다(c7552에서도 6,331초). 표 3, 4, 5의 성능 비교를 통해 본 논문에서 구현한 테스트 패턴 생성기의 효율성이 기존의 다른 방식들에 비해 증대되었음을 알 수 있다. 생성된 테스트 패턴을 IDDQ 테스트에 적용할 경우, 가장 적은 수의 테스트 패턴으로 높은 고장 검출률을 요구한다. 생성된 테스트 패턴 10, 20, 30개의 단계별 고장 검출률의 증가 상태를 표 6에 비교하였다. 그림 9를 통해서 구현된 테스트 패턴 생성기의 빠른 고장 검출률을 확인할 수 있다.

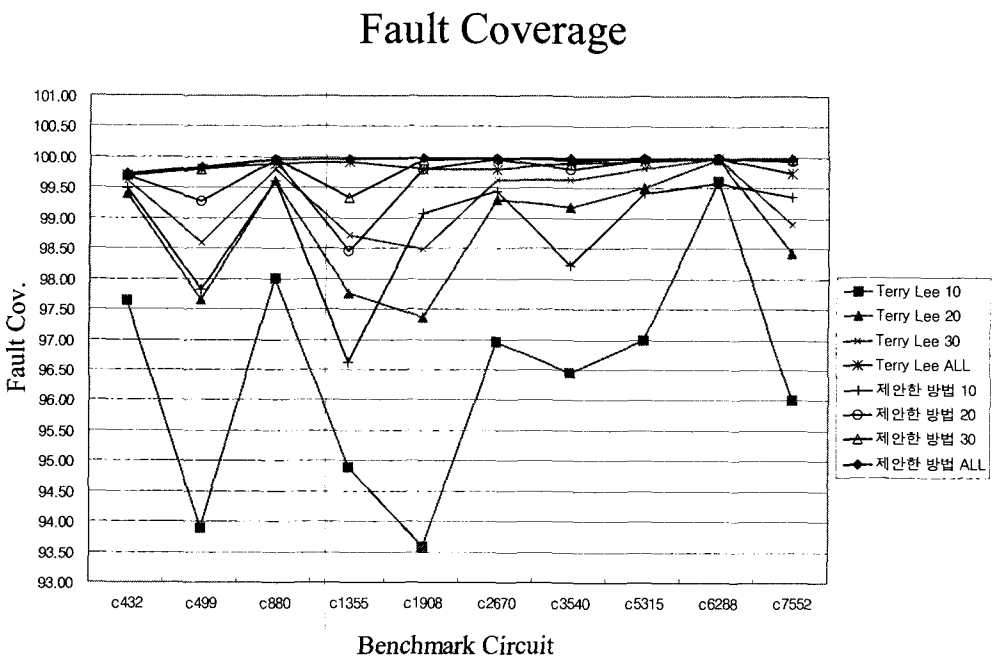


그림 9. 고장 검출률 비교
Fig. 9. Comparison of fault coverage.

V. 결 론

고집적화에 따른 CMOS VLSI 회로에서 발생빈도가 가장 높은 합선고장을 효과적으로 검출하기 위한 IDDQ 테스트를 위한 패턴 생성기를 구현하였다. 본 논문에서 고려한 전체 합선고장 모델은 내부노드 혹은 외부노드에 한정시킨 합선고장이 아닌 테스트 대상회로의 모든 노드에서 발생 가능한 단락을 고려하였다. 구현된 테스트 패턴 생성기는 PC의 Windows 기반에서 Visual Basic 소프트웨어를 이용하여 사용자의 편의를 위해 GUI 방식으로 구현하였으며, ISCAS '85 벤치마크 회로를 이용한 시뮬레이션 결과로 고장 검출률, 생성된 테스트 패턴의 개수, 테스트 패턴 생성 시간에서 기존의 방식에 비해 효율성이 증가되었음을 확인하였다(표 3, 표 4, 표 5 참조). 이러한 결과는 제안된 이웃 조사 알고리즘을 이용하여 빠른 고장 시뮬레이션을 통해 많은 테스트 패턴의 적용이 가능하였고, 또한 새롭게 제안한 고장 collapsing 알고리즘을 이용하여 전체 고장 수를 효과적으로 감소시킨 결과이다.

따라서 구현된 IDDQ용 테스트 패턴 생성기를 이용하여 높은 고장 검출률을 갖는 적은 수의 테스트 패턴을 생성할 경우, 상대적으로 느린 전류 테스트의 단점을 보완할 수 있으며, 기존의 전압 테스트 방식에 비해서 VLSI CMOS 회로에 대한 높은 레벨의 신뢰성 있는 테스트가 가능하다. 현재 제안된 알고리즘을 순차회로에 적용시키는 연구가 진행중이며, 이와 병행하여 생성된 테스트 패턴과 내장형 전류 감지기(Built-in Self Sensor)를 이용하여 내장형 자체 테스트(Built-in Self Test) 기법에 응용시키는 연구가 진행되어야 할 것이다.

참 고 문 헌

[1] R. Rajsuman, *IDDQ Testing for CMOS VLSI*, *Artech* House, 1994.

[2] J. A. Abraham, "Challenges in fault detection," *International Symposium on Fault-Tolerant Computing*, pp. 96~114, 1995.

[3] 전병실 외, "기능테스트와 IDDQ 테스트를 위한 자체 점검 BIST 회로의 설계," 서울대학교 반도체공동연구소 연구보고서, 1998

[4] P. J. Thadikaran, "Evaluation, selection and generation of IDDQ tests," PHD. Thesis,

Department of Computer Science, State University of New York, 1996.

[5] A. Jee and F. J. Ferguson, "Carafe: An inductive fault analysis tool for CMOS VLSI circuits," *Proc. IEEE Int'l Conf.*, pp. 73~82, 1993.

[6] T. Lee, I. N. Hajj, E. M. Rudnick, J. H. Patel, "Genetic-algorithm based test generation for current testing of bridging faults in CMOS VLSI circuits," *IEEE VLSI Test Symposium*, pp. 456~462, 1996.

[7] T. Shinogi and T. Hayashi, "An iterative improvement method for generating compact tests for IDDQ testing of bridging faults," *IEICE Trans. INF & SYST.*, vol. E81-D. no. 7, July 1998.

[8] S. Chakravarty and P. J. Thadikaran, "Simulation and generation of IDDQ tests for bridging faults in combinational circuits," *IEEE Trans. Computers*, vol. 45, no. 10, pp. 1131~1140, Oct. 1996.

[9] 전병실 외, "합선고장을 위한 IDDQ 테스트 패턴 발생기의 구현," 한국통신학회논문지, vol. 24, no. 12-A, pp. 2008~2014, 1999

[10] U. Mahlstedt, et al., "Test generation for IDDQ testing and leakage fault detection in CMOS circuits," *European DAC*, pp. 486~491, Sep. 1992.

저 자 소 개

裵 晟 桓(正會員)

1993년 2월 : 전북대학교 전자공학과 졸업(공학사), 1995년 2월 : 전북대학교 대학원 전자공학과 졸업(공학석사), 2000년 2월 : 전북대학교 대학원 전자공학과 졸업(공학박사), 현재 한려대학교 멀티미디어정보통신공학과 전임강사. 관심분야 : VLSI 설계, ASIC 테스팅

金 觀 雄(學生會員)

1996년 2월 : 전북대학교 전자공학과 졸업(공학사), 1998년 2월 : 전북대학교 대학원 전자공학과 졸업(공학석사), 1998년 3월 - 현재 : 전북대학교 대학원 전자공학과 박사과정. 관심분야 : ATM 교환기술, 트래픽제어, 라우팅

田 炳 實(正會員)

1967년 2월 : 전북대학교 전기공학과 졸업(공학사), 1969년 2월 : 전북대학교 대학원 전자공학과 졸업(공학석사), 1974년 2월 : 전북대학교 대학원 전자공학과 졸업(공학박사), 1979년 ~ 1980년 : 미국 University Nortre Dame 전기공학과 객원교수, 1971년 - 현재 : 전북대학교 공과대학 전자정보공학부 교수, 관심분야 : ATM, 트래픽제어, Design for testability