論文96-33A-3-19

CMOS VLSI의 IDDQ 테스팅을 위한 ATPG 구현

(Implementation of ATPG for IDDQ Testing in CMOS VLSI)

金 强 哲 *, 柳 珍 守 **, 韓 哲 鵬 ***

(Kang Chul Kim, Jin Soo Ryu, and Seok Bung Han)

요 약

VLSI의 집적도가 증가함에 따라 설계와 제조과정에서 기존의 논리 테스트 방법으로는 검출하기 어려운 고장들이 발생하고 있다. 최근에는 이러한 고장을 검출하기 위한 IDDQ 테스팅 방법의 중요성이 증대되고 있다. 본 논문에서는 CMOS를 사용하는 디지털 조합회로에 대하여 IDDQ 테스팅 방식에 적용될 수 있는 자동 테스트 패턴 생성기(Gyeongsang Automatic Test Pattern Generator: G-ATPG)를 구현하였다. G-ATPG는 프리미티브 게이트(primitive gate)와 XOR 게이트의 내부에서 발생할 수 있는 stuck-at, 트랜지스터 내부 브리징, GOS 고장을 스위치 레벨에서 프리미티브 고장패턴(primitive fault pattern)으로 모델링하였다. 그리고 게이트 내부의 고장을 활성화시키는 입력 조합을 구성하고, 전체 회로를 게이트 레벨에서 시뮬레이션하였다. 따라서 전체 회로를 스위치 레벨에서 모델링한 것과 같은 효과를 나타내어 기억장치의 사용을 최소로 하면서 컴퓨터 사용시간을 단축할 수 있었다. IDDQ 테스팅 방식은 고장 게이트 내에서 전류 경로만을 형성하면 전원에서 공급되는 전류를 관찰하여 고장의 검출이 가능하므로 고장이 발생한 게이트를 활성화 시키는데 필요한 게이트 만을 논리 시뮬레이션이 가능하도록 하는 고장활성 게이트 개념을 도입하였다. 그리고 하나의 테스트 패턴이 생성되면 모든 게이트에 대하여 프리미티브 고장패턴을 그레이딩(grading)하여 테스트 패턴 생성 시간을 단축하였다. 벤치마크 회로에 대한 G-ATPG의 시뮬레이션 결과에서 기존의 자동 테스트 패턴 생성기보다 컴퓨터 사용시간과 고장 검출율이 향상되었음을 확인하였다.

Abstract

As the density of VLSI increases, the conventional logic testing is not sufficient to completely detect the new faults generated in design and fabrication processing. Recently, IDDQ testing becomes very attractive since it can overcome the limitations of logic testing. In this paper, G-ATPG(Gyeongsang Automatic Test Pattern Generator) is designed which is able to be adapted to IDDQ testing for combinational CMOS VLSI. In G-ATPG, stuck-at, transistor stuck-on, GOS(gate oxide short) or bridging faults which can occur within primitive gate or XOR is modelled to primitive fault patterns and the concept of a fault-sensitizing gate is used to simulate only gates that need to sensitize the faulty gate because IDDQ test does not require the process of fault propagation. Primitive fault patterns are graded to reduce CPU time for the gates in a circuit whenever a test pattern is generated. The simulation results in bench mark circuits show that CPU time and fault coverage are enhanced more than the conventional ATPG using IDDQ test.

* 正會員, 晉州産業大學校 電子計算學科
(Chinju National University)
** 準會員, *** 正會員, 慶尙大學校 電子工學科, 自動化 및 컴퓨터 應用技術硏究所 硏究員
(Gyeongsang National University)
接受日字: 1995年7月1日, 수정완료일: 1996年2月13日

Ⅰ. 서 론

반도체 공정 및 회로 설계기술의 발전으로 IC의 집적도가 증가함에 따라 설계와 제조 과정에서 많은 물리적인 결함들이 발생하고 있다. 이러한 결함들은 기존

의 논리 테스팅 방식의 주된 고장 모델이었던 stuck-at 형태의 고장 외에 브리징 고장, 개방 고장, GOS 고장 등으로 대부분 논리 고장은 발생하지 않으면서 신호지연이 발생하고, 시간이 지남에 따라서 그 상태가 더욱 악화되어 현장에서 사용하는 도중에 논리 고장을 일으켜 시스템 신뢰도를 현저하게 감소시키는 새로운 형태의 고장들이다.[1-4]

지난 30 여 년 동안, 고장을 회로의 한 라인에 0 또는 1의 논리 값이 고정된 것으로 가정하여 전압 측정에 기반을 둔 전압 테스팅 또는 논리 테스팅 방식이 주류를 이루어 왔다.[5][6] 그러나 브리징 고장 같은 물리적인 결함과 신뢰도에 영향을 미치는 GOS 고장에는 논리 테스팅에 의해서 만족할 만한 고장 검출율을 얻지 못하였다.[7] 이와 같은 고장들은 VDD와 GND 사이에 전류 경로를 형성하게 되어 평형상태(steady state)에서 큰 고장전류를 흐르게 한다. 최근에 회로 내의 고장 유무에 따라 크게 변화하는 평형상태의 전류 값(IDDQ : quiescent current)을 비교하여 다양한 형태의 고장을 용이하게 검출할 수 있는 IDDQ 테스팅 방법이 연구되기 시작하였다.[8]

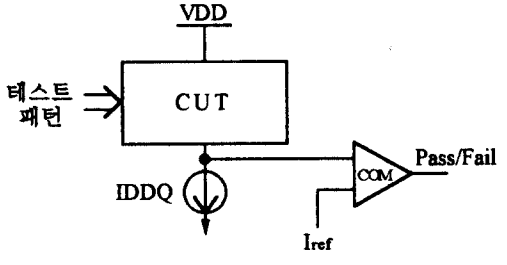

그림 1. IDDQ 테스팅 방법
Fig. 1. IDDQ testing method.

IDDQ 테스팅은 그림 1.과 같이 CUT(circuit under test)에 테스트 패턴이 인가되어 고장이 활성화되면, VDD에서 공급되는 IDDQ 값을 측정하고 기준전류(I_{ref})와 비교하여 즉시 고장을 관찰할 수 있으므로 고장을 주출력까지 전파하는 과정을 제거할 수 있는 장점을 가지고 있다. 그리고 IDDQ 테스트는 논리 테스트 보다 적은 수의 테스트 패턴을 사용하여 stuck-at, 브리징 고장 뿐만 아니라 게이트 누설전류와 같이 신뢰도에 관계되는 고장도 검출할 수 있다.[9][10] 그러나 기존의 테스트 장비를 사용할 수 없고, 속도가 느리며, 기준 전류의 설정이 어려운 단점이 있다.[11]

테스트 생성의 목적은 고장이 없는 회로와 고장이 있는 회로의 출력을 서로 다르게 나타낼 수 있는 입력 패턴을 구하는 것으로 공간탐색 방식에 따라 D, PODEM 및 FAN 알고리즘으로 구분된다.[12] D 알고리즘은 가능한 모든 경로를 통하여 고장효과를 전달하는 것으로 테스트 패턴이 존재하는 경우라면 테스트 패턴을 얻을 수 있다. 그러나 하나의 결정(decision)을 내린 후 이에 대한 확인 과정을 거치지 않고 다음 결정을 내리기 때문에 충돌의 발생가능성이 많아 백트랙(backtrack)의 수가 증가될 수 있다. PODEM은 탐색공간을 주입력에 한정하고 여러 개의 결정을 동시에 내리지 않아 충돌의 발생 가능성이 적어지며, XOR 게이트 등을 포함하는 고장 수정 및 변환 회로에서 D 알고리즘 보다 월등히 좋은 성능을 발휘한다. FAN은 PODEM을 변형한 것으로 팬아웃(fan-out)에 중점을 두고, 테스트 생성 시간을 줄이기 위하여 다중 백트랙을 사용하고 있다.

IDDQ 테스팅을 위하여 전류 모니터(current monitoring)[13]에 관한 연구뿐만 아니라, 테스트 패턴 생성 알고리즘에 관한 연구도 많이 수행되고 있다. 기존의 논리 테스팅에서 얻어진 테스트 패턴을 IDDQ 테스팅 방식에서도 사용할 수도 있지만[14], 이 경우에는 고장이 게이트 레벨에서 모델링되어 게이트 내부에서 발생한 고장을 완벽하게 검출할 수 없다. 따라서 IDDQ 테스팅을 위해서는 스위치 레벨에서 고장을 모델링하여야 높은 고장 검출율을 얻을 수 있다.[15][16] 그러나 스위치 레벨에서 고장이 모델링되면 처리하여야 할 데이타가 많기 때문에 큰 기억장치가 필요하고, 컴퓨터 사용시간이 길어지게 되므로 컴퓨터 사용시간을 단축할 수 있고, 적은 양의 기억장치를 필요로 하는 새로운 고장 모델이 필요하다. 그리고 IDDQ 테스팅의 느린 속도를 보상하기 위하여 테스트 패턴의 수를 줄일 수 있으며, 높은 고장 검출율을 얻을 수 있는 새로운 알고리즘의 개발이 필요하다.[17]

본 논문에서는 CMOS를 사용하는 디지털 조합회로에 대하여 IDDQ 테스팅 방식에 적용될 수 있고, PODEM에 기초을 둔 자동 테스트 패턴 생성기(G-ATPG)를 구현하였다. G-ATPG는 프리미티브 게이트(primitive gate)와 XOR 게이트의 내부에서 발생할 수 있는 stuck-at, 트랜지스터 내부 브리징, GOS 고장을 활성화시킬 수 있는 입력조합을 스위치

레벨에서 구하여 프리미티브 고장패턴(primitive fault pattern)으로 모델링하였다. IDDQ 테스팅 방식은 기존의 논리 테스트와는 달리 고장 게이트 내에서 전류 경로만을 형성하면 전원에서 공급되는 전류를 관찰하여 고장의 검출이 가능하므로 고장이 발생한 게이트를 활성화시키는데 필요한 게이트만을 시뮬레이션이 가능하도록 하는 고장활성 게이트 개념을 제안하여 기억장치의 사용을 최소로 하면서 컴퓨터 사용시간을 단축할 수 있도록 하였다. 그리고 하나의 테스트 패턴이 생성되면 모든 게이트에 대하여 프리미티브 고장패턴을 그레이딩(grading)하여 테스트 패턴 생성 시간을 단축할 수 있는 알고리즘을 제안하였다.

본 논문의 구성은 Ⅱ장에서 IDDQ 테스팅을 위한 고장 모델과 프리미티브 고장패턴을 생성하는 방식을 기술하고, Ⅲ장에서는 고장활성 게이트를 계산하는 알고리즘과 논리 시뮬레이션 및 고장패턴 그레이딩(fault pattern grading)에 관하여 설명한다. Ⅳ장에서는 G-ATPG의 구성과 테스트 패턴 생성 알고리즘을 설명한다. Ⅴ장에서는 벤치마크 회로들에 대한 실험 결과를 보여주고, Ⅵ장에서 결론을 기술한다.

Ⅱ. IDDQ 테스팅을 위한 고장모델

CMOS VLSI 회로에는 설계의 결함이나 공정 변수의 변화 등이 원인이 되어 stuck-at 고장뿐만 아니라, GOS 및 트랜지스터 내부 브리징 고장 등이 나타나며, 이들은 VDD와 GND 사이에 전류 경로가 형성되므로 노드 사이의 브리징 고장으로 나타낼 수 있다. IDDQ 테스팅에서는 모든 회로를 스위치 레벨에서 모델링하는 것이 가장 이상적이지만, 이 경우에 처리되는 데이터가 너무 방대하여 많은 기억장치와 컴퓨터 사용시간을 필요로 한다. 본 논문에서는 고장이 발생한 게이트만을 스위치 레벨에서 고려하고, 나머지 게이트들에 대해서는 게이트 레벨에서 시뮬레이션 하여 테스트 패턴을 생성하므로 전체 회로를 스위치 레벨에서 모델링한 것과 같은 효과를 나타낼 수 있도록 하였다. 그리고 정적 CMOS로 구성되는 NAND, AND, NOR, OR, INV(inverter), BUF(buffer) 등의 프리미티브 게이트와 통과 게이트(transfer gate)로 구성되는 XOR 게이트의 단일 게이트 내에서 발생할 수 있는 stuck-at, 브리징, GOS, 트랜지스터 내부 브리징 고장을 대상으로 스위치 레벨에서 고장을 모델링하였다.

1. 프리미티브 게이트의 고장 모델

하나의 게이트 내에 발생할 수 있는 브리징, GOS 및 stuck-at 등의 고장을 고려하여 이들을 활성화시킬 수 있는 게이트의 입력 패턴을 프리미티브 고장패턴이라 한다. CMOS 프리미티브 게이트 내에서 개방 고장을 제외한 모든 고장은 stuck-at 테스트 세트에 의해서 100% 검출이 가능하다는 것이 알려져 있으며, N 입력 프리미티브 게이트에 대해서는 (N+1) 개의 고장패턴이 필요하다.[11][17]

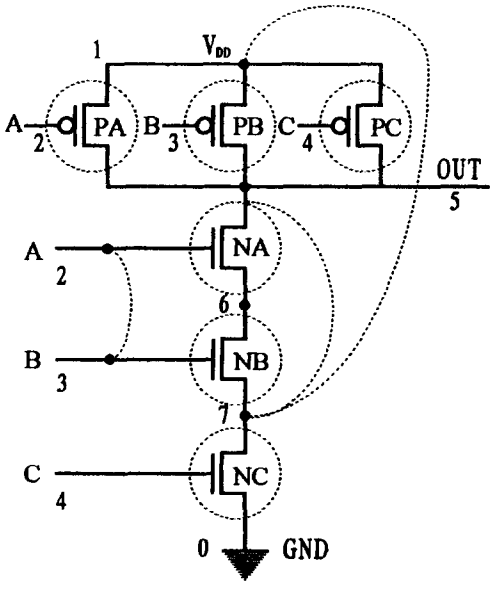

그림 2. 3 입력 NAND 게이트 내에 존재하는 브리징 고장
Fig. 2. Bridging faults in 3-input NAND gate.

그림 2는 3 입력 NAND 게이트를 나타낸 것으로 점선은 게이트 내부에서 발생할 수 있는 브리징 고장의 일부를 나타낸 것이다. 3 입력 NAND 게이트 내부에서는 8 개의 노드가 존재하여 모두 28개의 브리징이 발생할 수 있으며, 각각의 브리징에 대하여 그 고장을 활성화시킬 수 있는 즉, VDD와 GND 사이에 전류 경로를 만들 수 있는 게이트의 입력 패턴을 구할 수 있다. VDD와 GND 사이의 브리징 고장은 입력 패턴을 인가하지 않아도 검출이 가능하다. 모든 입력 패턴을 간략화 시키면 4 개의 입력 패턴(011, 101, 110, 111)이 된다. 같은 방법으로 프리미티브 게이트 AND(NAND), OR(NOR)와 INV(BUF)에 대하여 입력 패턴을 구하면 표 1과 같다.

표 1. 프리미티브 게이트의 고장패턴
Table 1. Fault pattern of primitive gates.

AND(NAND)				OR(NOR)				INV(BUF)
2 입력		3 입력		2 입력		3 입력		입력
A	B	A	B C	A	B	A	B C	A
0	1	0	1 1	0	0	0	0 0	0
1	0	1	0 1	0	1	0	0 1	1
1	1	1	1 0	1	0	0	1 0	
		1	1 1			1	0 0	

표 1의 입력 조합들은 각 입출력선에 0 또는 1을 모두 인가할 수 있으므로 각 입출력선의 stuck-at-0/1 고장도 테스트할 수 있음을 알 수 있다. 그리고 하나의 신호선에 대해 이웃하는 신호선이 서로 반대되는 논리값을 가지게 되어, 두 신호선의 단락 유무에 따라 전류 경로가 형성될 수 있으므로 그림 2의 각 입출력선 사이의 브리징 고장도 테스트할 수 있다. 또한 GOS 고장은 트랜지스터 내부 브리징 고장으로 모델링될 수 있으므로 모두 검출이 가능하다.

2. XOR 게이트의 고장 모델

XOR 게이트는 정적 CMOS, 통과 게이트 또는 프리미티브 게이트의 조합으로 구성될 수 있다. 정적 CMOS를 이용하는 XOR 게이트는 프리미티브 게이트와는 달리 N 입력의 경우에 2^N 개의 고장패턴을 사용하여야 모든 브리징 고장의 검출이 가능하며, 프리미티브 게이트의 조합으로 구성된 XOR 게이트의 고장패턴은 표 1에 있는 고장패턴을 이용할 수 있다.

표 2. XOR 게이트의 고장패턴
Table 2. Fault pattern of XOR gates.

2 입력		3 입력		
A	B	F1	C	F2
1	1	0	1	1
1	0	1	0	1
0	1	1	1	0

본 논문에서는 XOR 게이트를 IC의 설계에 가장 많이 사용되는 통과 게이트로 구성되는 복합 게이트(complex gate)로 가정하여 고장패턴을 구한다. 그림 3과 같이 XOR 게이트에는 6 개의 노드가 존재하므로 15 개의 노드 쌍이 존재하여 각각에 대한 전류 경로를 구하여 개방 고장을 제외한 모든 고장을 검출할 수 있는 고장패턴을 구하면 표 2와 같다.

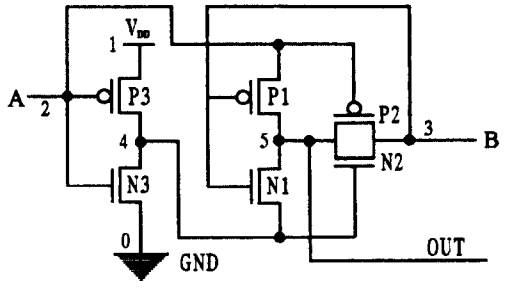

그림 3. XOR 게이트 회로
Fig. 3. Circuit of XOR gate.

표 2에서 AB의 입력 중 01의 경우는 0X로부터 유도되며, 이는 게이트의 입력이 하나 증가될 때 인가할 입력 조합에 일련의 규칙성을 부여할 수 있다.

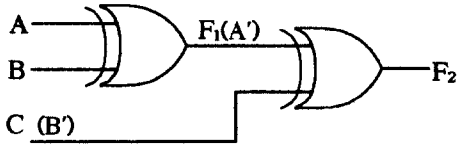

그림 4. 3-입력 XOR 게이트
Fig. 4. 3-input XOR gate.

XOR 게이트는 하나의 입력이 증가될 때, 현재 XOR 게이트의 출력이 다음 게이트의 입력으로 사용되기 때문에 게이트의 전체적인 형태는 바뀌지 않는다. 예를 들어, 그림 4.와 같이 XOR 게이트의 입력 수가 늘어 날 때, 앞단 게이트의 출력(F1)이 현재 게이트의 첫 번째 입력(A')으로 다시 사용되고, 추가되는 다른 입력(C)을 두 번째 입력(B')으로 사용하여 전체 3-입력(ABC)의 XOR 게이트를 만든다. 이 때, 출력 F1의 순서와 입력 A의 순서가 같아야 다음 게이트의 입력으로 재사용 될 수 있다. 그러나 F1과 A의 순서가 달라도 입력 C의 조합을 101로 인가하므로 F1을 다음 게이트의 입력으로 재사용 가능하게 하였다. 즉 AB 입력 조합과 F1C의 입력 조합의 순서가 다를 뿐이며 인가하는 전체적인 입력 조합은 같게 된다. 그러므로 입력수가 늘어나도 같은 조합을 반복하여 사용할 수 있다.

위와 같이 프리미티브 게이트는 N 입력에 대하여

(N+1) 개의 고장패턴이 필요하고, XOR 게이트는 입력 수에 관계없이 3 개의 고장패턴을 인가하면 게이트 내에 존재할 수 있는 stuck-at, GOS, 브리징 고장을 검출할 수 있다.

Ⅲ. 고장활성 게이트 계산과 고장패턴 그레이딩

IDDQ 테스팅 기법은 고장을 활성화시키기만 하면 전원에서 흐르는 전류를 측정하여 고장의 유무를 판별할 수 있으므로 논리 테스팅 기법과는 달리 주출력까지 고장을 전파할 필요가 없다. 따라서 고장 게이트에 관련된 게이트만을 고려하면 테스트 패턴 생성 시간을 단축시킬 수 있다.

테스트 패턴 생성 과정에서 고장 게이트의 프리미티브 고장패턴에 대하여 주입력 값을 결정하게 되는데, 현재까지의 주입력 상태에서 최근에 할당된 주입력 값이 테스트 패턴으로 타당한지의 여부를 확인하기 위해서 논리 시뮬레이션을 수행한다. 논리 시뮬레이션의 진행은 주입력에서 전방향으로 고장 게이트까지만 진행하는데 이를 위해서 고장활성 게이트들을 참조하게 된다. 여기서 고장활성 게이트는 고장 게이트에서 주입력까지의 신호 전달 경로에 포함되는 모든 게이트들을 의미한다. 고장 게이트가 선택이 되면, 이 게이트에 대한 모든 고장활성 게이트들의 플래그(flag)를 설정하여 등록시킨다. 고장활성 게이트 중에서 특히 고장 게이트의 입력 단에 직접 연결되어 있는 게이트들을 필수 고장활성 게이트라 하며, 함의조작(implication) 과정에서 한 게이트의 출력값에 대하여 특정 입력선에 값이 할당되어야 할 경우 이를 오브젝티브(objective)라 한다. enrol_act_gate()와 log sim(pi obj) 함수는 고장활성 게이트 계산과 논리 시뮬레이션에 대한 알고리즘이다.

```
enrol_act_gate( )
/* 필수 고장활성 게이트는 abs_gate의 linked list로 등록
   되어 있다. */
{
    회로 내의 모든 게이트의 flag = OFF; /* flag의 초
        기화 */
    tmp_gate = abs_gate; /* 하나의 필수 고장활성 게
        이트에 대하여 */
    while (tmp_gate 가 NULL이 아님) {
        tmp_gate의 flag = ON; /* 고장활성 게이트
            로 등록 */
        push(tmp_gate2); /* tmp_gate의 입력에
            연결되어 있는 게이트들 중 주입력이 아니
            고 flag가 off인 게이트(tmp_gate2)를
            stack에 저장한다. */
        stack에 저장되어 있는 게이트들을 하나씩
            pop( )하여 고장활성 게이트로 등록한다;
        tmp_gate = abs_gate->next; /* 다음 필수
            고장활성 게이트로 이동 */
    }
}

log_sim(pi obj)
/* 주입력에 대한 오브젝티브 pi_obj를 인수로 받는다. */
{
    push(tmp_gate); /* 스택에 pi_obj에 연결된 모든
        게이트를 저장한다. */
    while (스택에 처리할 오브젝티브가 존재) {
        a_gate = pop (스택); /* 스택에서 한 게이
            트를 꺼내어 a_gate에 둔다. */
        a_gate의 출력값을 결정한다;
        if (a_gate 출력값과 할당값 != X &&
            a_gate의 출력값 != a_gate의 할당
            값) return STOP;
        if (a_gate에 연결된 다음단 게이트 형태 !=
            주출력 &&
            a_gate에 연결된 다음단 게이트명 !=
                고장 게이트명 &&
            a_gate에 연결된 다음단 게이트 == 고
                장활성 게이트)
            push(a_gate에 연결된 다음단 게이트);
    }
    return CONTINUE;
}
```

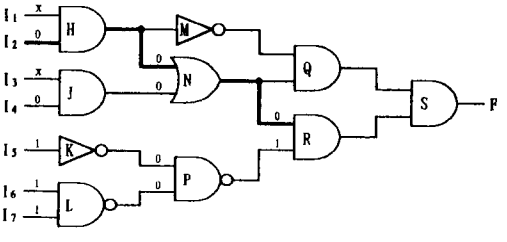

그림 5. 논리 시뮬레이션과 고장패턴 그레이딩의 예제 회로

Fig. 5. An example circuit of logic simulation and fault pattern grading.

그림 5의 회로에서 R 게이트가 대상 고장 게이트이면 필수 고장활성 게이트는 N과 P 게이트가 되며, 고장활성 게이트는 H, J, K, L, N, P의 게이트들이 된

다. 여기서 고장 게이트 R에 대해 01의 프리미티브 고장패턴으로 주입력 I2의 값을 0으로 결정하였다면, 이에 대해 논리 시뮬레이션을 수행하게 된다. 이 때, I2의 신호는 H 게이트로 통과된 후 팬아웃을 만나게 된다. 여기서 M 게이트는 고장활성 게이트로 등록되어 있지 않으므로 N 게이트로만 통과된다. 다음에 만나는 팬아웃(fanout)에서도 마찬가지로 Q 게이트로는 통과되지 않고 R의 고장 게이트를 만나 I2에 대한 논리 시뮬레이션은 여기에서 중단된다. 이와 같이 고장활성 게이트를 사용하여 고장 게이트까지의 시뮬레이션을 수행하므로, 고장 신호를 주출력까지 전파시키는 기존의 논리 테스팅 방법보다 테스트 패턴 생성 시간을 감소시킬 수 있다. 그리고 고장 게이트의 프리미티브 고장패턴 만을 활성화시키면 고장의 검출이 가능하므로 고장 신호를 주출력까지 전파하는 과정에서 발생할 수 있는 충돌이 제거되어 고장 검출율을 향상시킬 수 있다.

고장 게이트의 프리미티브 고장패턴에 대하여 주입력 값과 모든 고장활성 게이트들의 입력이 정해졌을 때 고장활성 게이트에 정해진 입력 조합에 대해서 프리미티브 고장패턴에 해당하는 입력 조합들을 제거할 수 있다. 이를 고장패턴 그레이딩이라 한다. 즉, 제거되는 입력 조합은 해당 게이트의 프리미티브 고장패턴에 속하고, 현재의 주입력 값으로 만족되므로 그 고장패턴에 대해서 테스트 패턴을 생성할 필요가 없다. 고장패턴 그레이딩은 모든 오브젝티브를 만족하는 주입력이 결정되어 있고, 이 주입력들을 테스트 패턴으로 저장하기 직전의 단계에서 수행된다. 이는 모든 고장활성 게이트들을 대상으로 결정된 입력 조합 중에서 X(don't care)가 존재하지 않고, 0/1로 값이 모두 정해진 것에 대해 해당 고장패턴 플래그(flag)를 세팅(setting)시킨다. 플래그는 게이트 구조체 내에 자동 테스트 패턴 생성기에서 고려된 최대 게이트 입력 수만큼 크기의 배열로 정의되어 있다. 해당 패턴 플래그를 세팅시켜 놓으면, 프리미티브 고장패턴을 선택할 때 해당 패턴 플래그를 조사하여 그 고장패턴을 선택할 것인지 아닌지를 결정하게 된다. fault_grading() 함수는 고장패턴 그레이딩 알고리즘이다.

```
fault_grading( )
/* 고장 게이트에 대한 고장활성 게이트들의 입력값이 정해져
    있다 */
{
    a_gate = 0 레벨의 첫 게이트;
    while (a_gate) {
        if (a_gate 형태 == 주입력 || a_gate 형태
            == 주출력 || a_gate != 고장활성 게이
            트 || a_gate의 입력 == X) {
            a_gate = 레벨 정보에 의한 a_gate의 다음
                게이트;
            continue;
        }
            a_gate의 입력 조합을 읽어 들인다;
            프리미티브 고장패턴 테이블에서 a_gate의 입
                력 조합과 일치하는 곳의 패턴 flag를 ON
                시킨다;
            a_gate = 레벨 정보에 의한 a_gate의 다음
                게이트;
        }
}
```

그림 5의 회로에서 모든 게이트에 고장을 주입시킨 상태이고, 현재 R이 대상 고장 게이트이며, 01의 프리미티브 고장패턴에 대해서 테스트 패턴이 X0X0111로 생성되어 모든 고장활성 게이트들의 입력이 결정된 단계이다. 이 때, 고장패턴 그레이딩을 할 수 있는 곳은 게이트 K, L, N에 대해 각각 1, 11, 00의 고장패턴들을 그레이딩시킬 수 있다. 여기서, 게이트 P의 입력 조합(00)은 프리미티브 고장패턴이 아니기 때문에 고장패턴 그레이딩이 되지 않았고, 게이트 H(X0)와 J(X0)의 입력들은 X가 포함되어 있어서 역시 고장패턴 그레이딩이 되지 않았다. 고장패턴 그레이딩을 수행하여 각 게이트의 프리미티브 고장패턴을 줄이므로 전체 테스트 패턴 생성 시간을 줄일 수 있고, 생성되는 테스트 패턴의 수를 감소시킬 수 있다.

Ⅳ. G-ATPG의 설계 및 구현

1. G-ATPG의 구성

본 논문에서 제안한 G-ATPG는 그림 6과 같이 전처리 과정, 테스트 패턴 생성 과정과 후처리 과정으로 나누어진다. CUT의 회로 기술에 대한 데이터 화일을 입력으로 받아들이고, 모델링에서 얻어진 각 게이트의 입력 수에 대한 고장패턴 테이블은 배열로서 주어진다. 게이트의 입력 수는 최대 9개까지를 고려하였으며, 필요에 따라 고장패턴 테이블을 늘려 더 많은 입력도 고려할 수 있다. 고장 게이트의 테스트 패턴 생성 동안에 백트랙 발생 회수 또는 컴퓨터 사용시간을 수행조건으로 받아들여 최종적으로 생성된 결과를 테스트 패턴

화일에 저장한다. 테스트 패턴 생성 과정에서 제한된 백트랙 발생 회수 이내에 테스트 패턴을 생성하지 못하면 해당 고장패턴에 대한 테스트 패턴 생성을 포기하고, 실패 고장 집합 화일에 기록된다. 주어진 회로에 대한 컴퓨터 사용시간, 고장 검출율, 테스트 패턴 수 등의 모든 정보는 실행 정보화일에 기록되고, 테스트 생성과정에서 발생된 중복 고장은 중복고장 집합화일에 저장된다. 하나의 테스트 생성 과정 중에서 그레이딩을 거친 데이타는 고장패턴 집합화일에 저장된다.

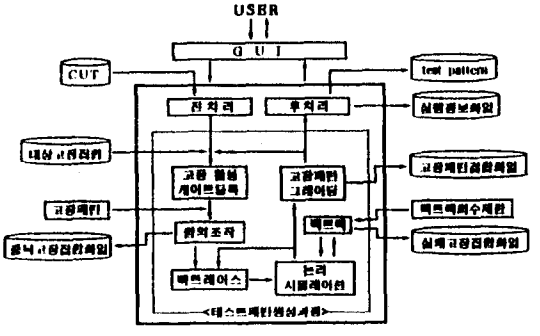

그림 6. G-ATPG의 구성도
Fig. 6. Overview of G-ATPG.

2. 전처리 과정

테스트 패턴 생성 알고리즘을 적용하기 전에 회로 구조로부터 필요한 정보를 분석하여 저장해 두어야 한다. 이 과정을 전처리 과정이라 하며, CUT의 입력, 회로의 레벨화, 제어도 계산 등이 수행된다.

G-ATPG는 CUT의 회로 구조에 대한 입력 데이터로서, 회로 기술언어를 입력으로 받는다. 테스트할 회로가 결정되면 회로에 대한 내부 자료구조를 구성하고, 테스트 패턴 생성 과정에서 필수적인 고장 게이트의 순차처리와 제어도 계산에 필요한 정보를 제공하기 위하여 회로의 레벨화가 수행된다. 그리고 테스트 패턴을 효과적으로 생성하기 위하여 SCOAP, CAMELOT, COP 및 FD의 테스트 용이도를 휴리스틱 조건으로 사용하며[11,18], IDDQ 테스팅 방식을 사용하여 고장 신호를 주출력까지 전파시킬 필요가 없으므로 제어도만 사용한다. 제어도를 계산하기 위해서는 주입력을 각 테스트 용이도의 특성에 따라 알맞은 값으로 초기화시키고, 회로의 레벨에 따라서 각 게이트에 대한 테스트 제어도를 계산한다. 이러한 테스트 제어도는 모든 회로에 대하여 항상 정확한 정보를 제공하여 주는 것이 아니라, 특정 회로에 대해서만 다른 알고리즘에 비하여 좋은 결과를 얻게 된다.

3. 테스트 패턴 생성 과정

전처리 과정이 끝나면 고장 게이트를 선택하고 백트랙 제한 회수를 입력받은 후에 CUT에 적용할 휴리스틱 조건을 선택한다. 그리고 고장 게이트에 대한 프리미티브 고장패턴을 선택하여 고장활성 게이트와 필수 고장활성 게이트의 등록, 함의조작, 백트레이스(backtrace), 논리 시뮬레이션 및 백트랙, 고장패턴 그레이딩 등을 수행한다.

고장 리스트에서 처리할 고장이 존재하면 가능한 많은 고장패턴 그레이딩을 수행하기 위해서 주출력에서 가까운 하나의 고장 게이트를 선택한다. 고장 게이트의 각 입력에 연결되어 있는 바로 앞 단의 게이트들을 필수 고장활성 게이트로 저장시키고, 이를 포함한 앞단 게이트들은 고장활성 게이트로 등록시킨다. 고장 게이트의 형(type)과 입력 수에 따라 프리미티브 고장패턴 테이블에서 프리미티브 고장패턴을 선택하고, 필수 고장활성 게이트의 출력에는 선택된 패턴에 따라 정해지는 값을 할당하며, 이를 고장패턴 설정이라 한다. 필수 고장활성 게이트에 할당된 값으로부터 함의조작을 수행하여 테스트 패턴이 존재하기 위해서 최소한 만족되어야 하는 신호선의 값을 미리 결정한다. 함의조작 시에 충돌이 발생하면 현재의 프리미티브 고장패턴을 만족시킬 수 없는 것이므로 현재 선택된 고장패턴에 대해서는 테스트 패턴이 존재하지 않는다.

게이트의 출력 값이 정해졌지만 입력 값이 미정인 신호선이 존재할 수 있으므로 함의조작만으로는 모든 게이트들의 입력 값을 유일하게 결정할 수 없다. 함의조작에 의하여 발생되는 오브젝티브들은 반드시 입력이 결정되어야 하므로 오브젝티브로부터 주입력 쪽으로 탐색공간에서 제어도를 참조하면서 경로를 찾아 하나의 주입력을 결정하게 되는데, 이 과정을 백트레이스라 한다. 백트레이스는 제어도에 의하여 경로를 선택하여 잘못된 판단을 내릴 수도 있으므로 하나의 주입력이 결정되면 논리 시뮬레이션을 수행하여 충돌 여부를 관찰한다. 논리 시뮬레이션 과정에서 먼저 값이 정해진 신호선과 충돌이 발생하면 백트랙을 수행하여 현재의 주입력 값들 중에서 최근에 결정된 값을 바꾸어 다시 논리 시뮬레이션을 수행한다. 이 과정에서 모든 가능한 주입력의 입력 조합을 시도하여도 오브젝티브를 만족

시킬 수 없다면 현 고장패턴에 대해서는 테스트 패턴이 존재하지 않는다. 오브젝티브를 모두 만족시키는 주입력 값들이 결정되었으면 고장활성 게이트들에 대한 프리미티브 고장패턴을 그레이딩한다.

현재의 주입력 값들을 테스트 패턴으로 하여 임시 화일에 저장한 후, 다음 프리미티브 고장패턴을 선택하여 수행한다. 대상 고장 게이트에 대한 각각의 프리미티브 고장패턴에 대해 모두 테스트 패턴이 생성되었으면, 레벨화 정보를 참조하여 시도되지 않은 고장 게이트를 순차적으로 선택하여 위의 과정을 반복 수행한다.

4. 후처리 과정

모든 고장 게이트들에 대해 테스트 패턴을 생성한 후에 테스트 패턴의 수를 줄이는 과정을 테스트 패턴 컴팩션(test pattern compaction)이라 한다. 임시 화일에 저장되어 있는 생성된 모든 테스트 패턴들 중에서 첫 번째 테스트 패턴을 선택하여 나머지 테스트 패턴들에 대해 컴팩션될 수 있는 패턴들을 골라 하나의 테스트 패턴으로 컴팩션시킨 후 테스트 패턴 화일에 저장시킨다. 그리고 컴팩션 대상 패턴들을 임시 화일에서 지우고, 이를 반복 수행함으로써 테스트 패턴 컴팩션이 수행된다.

V. 실험 및 결과

G-ATPG는 C 언어로 구현되었으며, 알고리즘의 성능 평가를 위해 Pentium-66의 Linux상에서 ISCAS 85 벤치마크 회로를 실험하였다. 이 때, 회로 기술 언어에 대한 파서(parser)는 UNIX 상에서 지원되는 LEX(scanner)와 YACC(Yet Another Compiler-Compiler) 유틸리티를 사용하였다.

표 3은 백트랙 제한 회수를 100으로 하여, 벤치마크 회로에 대하여 실험한 결과를 보여 주고 있다. 실패 고장패턴 수와 중복 고장패턴 수는 백트랙 제한 회수 내에 검출되지 않은 고장패턴 수와 중복 회로 내에서 만족될 수 없는 고장패턴 수를 각각 나타낸다. 순수 고장 검출율(pure fault coverage)은 전체 고장패턴 수에 대한 검출된 고장패턴 수의 백분율을 나타내고, 고장 검출율은 전체 고장 패턴 수에 대한 검출된 고장 패턴 수와 중복 고장 패턴 수를 합한 것의 백분율을 나타낸다. 평균 시간은 하나의 고장패턴 당 소비되는 컴퓨터 사용시간이며, 평균 백트랙 수는 테스트 패턴 생성 동안에 발생한 총 백트랙 수에 대한 총 고장패턴으로 계산되어 하나의 고장패턴 당 발생한 백트랙 회수를 나타낸다. G-ATPG는 C2670과 C3540을 제외하고 100%의 고장 검출율을 얻었다.

표 3. G-ATPG의 실행 결과 (백트랙 제한 회수 = 100)

Table 3. Experimental results of G-ATPG (Backtrack Limit = 100)

구분 / 회로	총고장 패턴 수	테스트 패턴 수	실패 고장 패턴 수	중복 고장 패턴 수	순수 고장 검출율	고장 검출율	평균 시간 (ms)	평균 백트랙 수
c432	496	30	0	0	100.00	100.00	1.61	0.01
c499	610	99	0	0	100.00	100.00	11.96	47.20
c880	1112	30	0	0	100.00	100.00	2.24	0.04
c1355	1610	97	0	0	100.00	100.00	3.85	0.35
c1908	2377	168	0	0	100.00	100.00	3.53	0.15
c2670	3268	43	11	17	99.14	99.66	5.75	1.44
c3540	4605	106	32	17	98.94	99.31	7.55	3.06
c5315	6693	81	0	2	99.97	100.00	6.97	0.43
c6288	7216	126	0	35	99.51	100.00	7.92	0.63
c7552	9656	136	0	20	99.79	100.00	7.53	0.26

표 4.는 IDDQ 테스팅을 위한 기존의 ATPG와 G-ATPG의 순수 고장 검출율을 나타낸 것이다. [7]에서는 stuck-at, stuck-on과 브리징 고장 모델을 대상으로 하나의 고장 게이트에 대한 고장 입력 조합을 만족시키기 위하여 주입력에서 임의적으로 테스트 패턴을 인가하였을 때의 random test set과 주입력에서 stuck-at test set을 인가하여 IDDQ 테스팅 기법을 사용한 고장 시뮬레이터로 시뮬레이션한 고장 검출율을 각각 보여주고 있다.

[8]은 IDDQ 테스팅을 하기 위한 모듈화되고, 계층적 접근 방법을 사용하고 있다. 가장 낮은 계층에서는 스위치 레벨 모델을 사용하여 레이아웃(layout)으로부터 브리징 고장이 발생할 가능성이 있는 고장을 찾아내어 백트랙 제한 회수를 20으로 하여 테스트 패턴을 생성한 결과를 보여준다. [9]에서는 stuck-on과 브리징 고장은 IDDQ 테스팅 기법을 사용하여 시뮬레이션한 고장 검출율이며, stuck-at 고장은 논리 테

스팅 기법을 사용한 고장 검출율을 나타내고 있다. 표 4.에서와 같이 G-ATPG는 기존의 ATPG 보다 순수 고장 검출율이 높다는 것을 알 수 있다.

표 4. 기존의 ATPG와 G-ATPG의 순수 고장 검출율
Table 4. Fault coverages of conventional ATPGs and G-ATPG.

회로 \ 구분	G-ATPG	[7] random test set	[7] stuck-at test set	[8]	[9] stuck-on	[9] stuck-at	[9] bridging
c432	100.00	96.10	96.10	99.16	-	-	-
c499	100.00	94.43	91.75	97.53	-	-	-
c880	100.00	99.98	100.00	99.97	92.45	-	98.39
c1355	100.00	99.61	99.56	95.21	91.16	98.45	91.03
c1908	100.00	98.63	98.63	99.80	89.79	98.56	89.40
c2670	99.14	98.33	98.39	98.80	95.75	95.82	99.22
c3540	98.94	98.72	98.72	99.45	95.04	91.10	98.19
c5315	99.97	99.20	99.20	99.62	-	-	
c6288	99.51	99.80	99.80	96.02		-	-
c7552	99.79	98.26	98.88	99.59	95.99	96.70	98.50

표 5. 컴퓨터 사용시간(sec)
Table 5. CPU time(sec).

	G-ATPG	[8]	[9]
사용 컴퓨터	Pentium 66	DEC3100	SUN Sparc 1
C432	0.80	9.7	
C499	7.30	19.0	
C880	2.49	7.9	34.24
C1355	6.20	51.6	73.86
C1908	8.39	22.0	158.52
C2670	18.20	38.8	192.71
C3540	34.77	78.7	
C5315	46.65	78.6	
C6288	57.15	251.7	
C7552	72.71	130.5	1154.70

각각의 논문에서 사용된 컴퓨터와 기억장치가 서로 다르므로 컴퓨터 사용시간에 대한 직접적인 비교는 어렵지만, 표 5는 스위치 레벨에서 고장을 모델링하고, IDDQ 테스팅 기법을 사용한 기존의 ATPG와 본 논문과의 컴퓨터 사용시간을 비교한 것이다. 컴퓨터 사용시간은 한 회로의 모든 고장을 검출하는데 필요한 시간을 나타낸 것으로, 평균 시간에 총 고장 패턴 수를 곱하여 계산하였다. [9]는 stuck-on 고장 모델에 대한 결과만을 나타낸 것이다. G-ATPG는 고장 게이트만을 스위치 레벨에서 모델링하고 나머지 게이트는 게이트 레벨에서 논리 시뮬레이션을 수행하였다. 그리고 고장활성 게이트의 도입으로 고장 게이트에 영향을 미치는 게이트만이 논리 시뮬레이션에 사용되고, 하나의 테스트 패턴이 생성되면 프리미티브 고장패턴이 그레이딩되어 컴퓨터 사용시간이 기존의 ATPG 보다 줄어들었음을 알 수 있다.

VI. 결 론

본 논문에서는 IDDQ 테스팅에서 게이트 내에 발생이 가능한 브리징, GOS, stuck-on 및 stuck-at 등의 모든 고장을 검출할 수 있는 자동 테스트 패턴 생성기인 G-ATPG를 구현하였다. G-ATPG는 XOR 게이트를 포함하여 프리미티브 게이트를 대상으로 게이트 내의 고장을 활성화시키는 입력 조합을 먼저 구성하고, 이를 전체 회로에 대해서 논리 시뮬레이션을 수행하므로 전체 회로를 스위치 레벨에서 모델링한 것과 같은 효과를 갖게 하였다. 그리고 각 프리미티브 고장패턴에 대해서 고장활성 게이트들에 정해지는 고장패턴을 미리 그레이딩하여 적은 수의 테스트 패턴을 생성하도록 하였으며, 고장활성 게이트를 참조하면서 고장 게이트까지만 논리 시뮬레이션을 수행하므로써 컴퓨터 사용시간을 단축시키고 고장 검출율을 향상시켰다. 본 논문에서 구현된 G-ATPG는 벤치마크 회로에 대한 실험을 통하여 기존의 테스트 패턴 생성기보다 고장 검출율이 향상되었고, 테스트 생성시간이 감소한 것을 확인하였다.

본 논문에서는 프리미티브 게이트와 정적 CMOS로 구성된 XOR 게이트를 대상으로 모델링하였으나, 앞으로는 복합 게이트, 동적 CMOS 등으로 확장하기 위한 데이타 베이스가 구축되어야 할 것이다. 그리고 게이트 사이(inter-gate)의 모든 노드에 대하여 브리징 고장을 검출할 수 있는 알고리즘과 IDDQ 테스팅에서 검출하기 어려운 stuck-open 고장에 대한 효과적인 모델

링에 대한 연구가 수행될 예정이다.

참 고 문 헌

[1] Thomas M. Storey and Wojciech Maly, "CMOS Bridging Fault Detection," Proc. Int. Test Conf., pp.842-851, Sept, 1990.

[2] V. H. Champac, A. Rubio and J. Figueras, "Electrical Model of the Floating Gate Defect in CMOS IC's: Implications on IDDQ Testing," IEEE Trans. on Computer-Aided Design, Vol.13, No.3, pp. 359-369, Mar.1994.

[3] Marek Syrzycki, "Modeling of Gate Oxide Shorts in MOS Transistors," IEEE Trans. Computer-Aided Design, Vol. 8, No. 3, pp.193-202, Mar. 1989.

[4] 한석붕, 김영일, 이건기, 이문수, "전류 테스팅 기법을 사용한 CMOS IC의 고장분석", 전자공학회 하계 종합 학술 대회 논문집, 제16권, 제1호, pp.484-488, 1993년 7월

[5] F. J. Ferguson and J. P. Shen, "Extraction and Simulation of Realistic CMOS Faults Using Inductive Fault Analysis," Proc. Int. Test Conf., pp. 475-484, 1988.

[6] 김윤홍, "VLSI회로의 고장 검출을 위한 테스트 패턴 생성에 관한 연구," 한양대 박사 학위 논문, 1992년 6월

[7] U. Mahlstedt, M. Heinitz, and J. Alt. "Test Generation for IDDQ Testing and Leakage Fault Detection in CMOS Circuits," Proc. EURODAC 92, 1992.

[8] S.Wayne Bollinger and Scott F.Midkiff, "Test Generation for IDDQ Testing of Bridging Faults in CMOS Circuits," IEEE Trans. Comput., vol.13, pp. 1413-1418, Nov. 1994

[9] Kuen-Jong Lee, "SWiTEST: A Switch Level Test Generation System for CMOS Combinational Circuits," IEEE Trans. Comput., vol.13, pp.625-636, May 1994.

[10] 김강철, 한석붕, "CMOS 회로의 전류 테스팅을 위한 내장형 전류감지기 설계," 대한전자공학회 논문지, 제 32권 B편 11호, pp. 70-80, 1995년 12월

[11] Rochit Rajsuman, Iddq Testing for CMOS VLSI, Artech House, 1995

[12] Rochit Rajsuman, Digital Hardware Testing: Transistor-Level Fault Modeling and Testing, Artech House, 1992.

[13] K. Baker, A. Bratt, A. Richarddon and A. Welbers, "Development of a Class 1 QTAG Monitor," ITC, pp.213-222, 1994.

[14] P. C. Maxwell, R. C. Aitken, V. Johansen, and I. Chiang, "The Effectiveness of IDDQ, Functional and Scan Tests: How Much Fault Coverages Do We Need?," Proc. Int. Test Conf., pp.168-177, 1992.

[15] W. Maly and P. Nigh, "Built-in Current Testing--Feasibility Study," Proc. Int. Conf. Computer-Aided Design, pp.340-343, 1988.

[16] M. Keating and D. Meyer, "A New Approach to Dynamic Idd Testing," Proc. Int. Test Conf., pp.316-321, 1987.

[17] 류진수, 이효상, 송근호, 김강철, 한석붕, 김윤홍, "CMOS VLSI의 전류 테스팅을 위한 자동 테스트 패턴 생성기의 구현," 전자공학회 하계 종합학술대회 논문집, 제18권, 제1호, pp635-638, 1995

[18] Hideo Fujiwara, Logic Testing and Design for Testability, MIT press, 1985.

저 자 소 개

金 强 哲(正會員)

1958년 12월 13일생. 1981년 2월 서강대학교 전자공학과 졸업(공학사). 1983년 2월 서강대학교 대학원 전자공학과 졸업(공학석사). 1992년 3월 ~ 경상대학교 대학원 전자공학과 박사과정. 1983년 3월 한국전자통신 연구소 반도체 연구단. 1989년 7월 삼성종합기술원 정보시스템 연구소. 1990년 3월 진주농림전문대학 전자계산학과. 1993년 7월 ~ 현재 진주산업대학교 전자계산학과 조교수

韓 晳 鵬(正會員) 第 32卷 B編 第 11號 參照
현재 경상대학교 전자공학과 부교수

柳 珍 守(準會員)

1967년 2월 2일 생. 1990년 2월 경상대학교 전자공학과 졸업(공학사). 1995년 8월 경상대학교 대학원 전자공학과 졸업(공학석사).