고장검사 적용시의 버스충돌에 관한 연구

김 규 철†

요 약

고장 시뮬레이터는 생성된 검사패턴의 품질을 평가하기 위하여 사용된다. 지금까지 발표된 대부분의 고장 시뮬레이터는 버스구조를 갖는 회로를 취급하지 않고 있다. 버스구조를 갖는 회로는 검사패턴에 따라 버스충돌을 일으킬 수 있다. 이 논문에서는 버스구조를 갖는 회로에 검사패턴을 적용할 때 발생 가능한 모든 버스충돌을 분석하여 여러 유형으로 분류하고 버스충돌을 확인하는 효율적인 방법을 제안하였다. 제안된 버스충돌 확인 방법을 사용한 고장 시뮬레이터는 주어진 검사패턴의 품질을 평가함과 동시에 버스충돌을 일으키는 검사패턴의 사용을 경고하여 버스충돌에 의한 버스구동기의 파괴를 막을 수 있다. 이러한 기능을 검사패턴 생성기와 연계하여 사용하면 버스충돌을 일으키지 않는 검사패턴을 생성할 수도 있다.

A Study on Bus Conflicts When Applying Test Patterns

Kyu chull Kim†

ABSTRACT

Fault simulators are used to evaluate the quality of a test pattern generated. So far, most fault simulators did not handle bus conflicts properly. We analyzed all possible bus conflicts when test patterns are applied to a circuit with bus structure and categorized bus conflicts into various types. Also, we proposed an efficient method to identify various types of bus conflicts. The fault simulator which employs the proposed method can evaluate the quality of test patterns generated and also can avoid destruction of bus drivers due to bus conflicts by warning the use of test patterns which cause bus conflicts. The proposed method can also be incorporated into a test pattern generator so that it can generate conflict-free test patterns.

1. 서 론

고장 시뮬레이션(fault simulation)은 집적회로 설계 과정의 중요한 부분을 차지하며 생성된 검사패턴(test pattern)의 품질을 평가하기 위하여 사용된다. 또한 검사패턴 생성기(test pattern generator)와 연계하여 검사패턴 생성 시간을 단축시키고 동시에 검사패턴의 길이를 줄이는 데에도 사용된다. 집적회로의 게이트 밀도가 높아짐에 따라 고장 시뮬레이션 시간도 N^2에 비례하여 증가한다. 여기서 N은 집적회로 내의 게이트 수이다. 그러므로 게이트 밀도가 높은 복잡한 회로에 대하여 정확한 시뮬레이션을 할 수 있는 고속의 고장 시뮬레이터가 필요하다.

문헌에 발표된 기본적인 고장 시뮬레이션 알고리즘은 병렬적 고장 시뮬레이션 (parallel fault simulat-ion)[1], 연역적 고장 시뮬레이션 (deductive fault simulation)[2], 동시적 고장 시뮬레이션(concurrent fault simulation)[3] 등이 있다. 이들 고장 시뮬레이션 알고리즘은 조합회로(combinational circuit)에 대하여 뿐만 아니라 동기식 순차회로(synchronous seq-uential circuit)

※ 본 연구는 단국대학교 대학연구비 지원에 의하여 수행되었음.
† 정 회 원 : 단국대학교 전자컴퓨터공학부 교수
논문접수 : 1997년 10월 21일, 심사완료 : 1998년 6월 29일

에도 확장되어 적용할 수 있다. 그러나 동기식 순차회로에 대한 연역적 고장 시뮬레이션은 미지값(unknown value)을 다른 고장 시뮬레이션처럼 쉽게 처리하지 못하는 단점이 있어 거의 사용되지 않는다. 최근의 동기식 순차회로에 대한 고장 시뮬레이터는 대부분 병렬적 고장 시뮬레이션이나 동시적 고장 시뮬레이션에 바탕을 둔 것이다. 예로, PROOFS[4]와 HOPE[5]는 병렬적 고장 시뮬레이션을 확장한 것이며 HYSIM[6]은 동시적 고장 시뮬레이션을 확장한 것이다. 그밖에 동기식 순차회로를 위한 고장 시뮬레이터로 DSIM[7], PARIS[8] 등이 있다. 이외에도 고속고장시뮬레이션(FFS: fast fault simulation) 알고리즘[9]-[12]은 팬아웃무존영역(fanout free region), 도미네이터(dominator), 독립브랜치(independent branch) 같은 회로의 특성을 사용하여 직접 수행해야 할 고장시뮬레이션의 양을 줄인다.

버스구조는 여러 종류의 디지털회로에 사용되고 있다. 그러나 대부분의 고장 시뮬레이터[1]-[12]는 버스구조를 가지고 있는 회로를 다루고 있지 않다. 동기식 순차회로에 대한 고장 시뮬레이션에는 논리값 0, 1 외에도 초기화되지 않은 메모리 소자(memory element)를 취급하기 위하여 미지값 X가 필요하다. 그러나 회로가 버스구조를 포함하고 있으면 버스에 연결된 버스구동기(bus driver)가 활성화(enabled)되지 않았을 때의 고저항(high impedance) 값을 적절히 처리하여야 하므로 고저항 상태를 나타내는 새로운 논리값 Z가 포함되어야 한다. PROOFS는 0, 1, X, Z 의 사용을 제안하고 있으나 비트 인코딩(bit encoding) 및 논리값 계산이 정확하지 않을 뿐만 아니라 Z 값을 사용한 예를 보이지 않았다. [13], [14]에서는 3상태 소자(tristate device)와 버스 그리고 양방향성 소자(bidirectional device)를 취급하였는데 검사패턴 자동생성기와의 이진값 호환을 위하여 불필요하게 미지값과 결정불가능값을 구분하여 사용하였고, 버스 상태표를 적용한 다음 올바른 버스값을 결정하기 위하여 다시 버스 패치(bus patch)를 하였다.

버스구조를 가지고 있는 회로에는 여러 버스구동기가 버스에 연결되어 사용된다. 회로를 구현한 기술이 바이폴라(bipolar) 기술인가 MOS 기술인가에 따라 다른 형태의 버스구동기가 사용되는데 이 논문에서는 3상태 버스구동기가 사용되는 CMOS 회로만을 취급하였다.

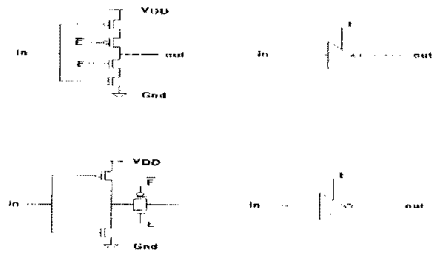
고장검출을 위하여 회로에 검사패턴을 적용할 때에 검사패턴에 따라 동일한 버스에 연결된 버스구동기가 동시에 여러 개 활성화되는 수도 있다. 이러한 경우 구동기의 출력값이 일치하지 않으면 활성화된 버스구동기를 통하여 VDD에서 Gnd로 전류 경로가 형성되어 과전류가 흐르게 된 버스구동기가 파괴될 수 있다. 활성화된 구동기의 출력값이 일치한 경우 버스구동기의 파괴는 없더라도 IDDQ [15]의 증가에 의하여 고장 마스킹(fault masking)을 일으킬 수 있다. 이 논문에서는 이러한 버스충돌(bus conflict)을 분석하여 여러 유형으로 분류하고 버스충돌을 확인하는 방법을 제안하였다. 제안된 방법을 사용한 고장 시뮬레이터는 시뮬레이션 도중 버스충돌이 발생하면 사용자에게 버스충돌의 발생을 경고하여 사용자가 검사패턴을 수정할 수 있도록 하여준다. 또는 사용자의 선택에 따라 버스충돌의 발생 후에도 고장 시뮬레이션을 정확하게 계속할 수 있다. 현재까지 공개된 ISCAS 85 [16], ISCAS 89 [17] 벤치마크 회로는 버스구조를 갖지 않은 것이어서 제안된 고장시뮬레이터를 직접 실험할 수 없었다. 앞으로 버스 구조를 갖는 벤치마크 회로가 공개되면 이를 사용하여 여러 가지 문제점 및 개선점을 찾아 보완하고, 검사패턴 자동생성기와 연계하여 버스충돌이 없는 검사패턴을 생성할 수 있는 검사 패턴 자동 생성기에 대하여 연구가 가능할 것이다.

이 논문의 구성은 다음과 같다. 2절에서는 버스에 연결되는 3상태 버스구동기의 특성과 논리값 계산에 관하여 논의하였다. 3절에서는 버스충돌시에 일어나는 버스구동기의 파괴에 대하여 설명하였으며 여러 형태의 버스충돌을 분석하고 분류하였다. 4절에서는 버스충돌을 확인하는 효율적인 알고리즘을 제시하였다. 5절에서는 버스충돌을 일으키지 않는 검사패턴 생성기의 가능성에 대하여 논의하였으며 6절에는 결론이 나와 있다.

2. 3상태 버스구동기

그림 1은 버스에 연결되는 두 형태의 3상태 버스구동기를 보이고 있다. 이 두 형태의 버스구동기는 전파지연(propagation delay)을 제외한 다른 기능은 동일하다. 이 3상태 버스구동기의 동작은 다음과 같다. 버스구동기가 활성화되면 (E=1일 때), 출력에는 입력이 반전(complemented)되어 나타난다. 다시 말하면 활성화

되었을 때는 NOT 게이트처럼 동작한다. 이 버스구동기가 비활성으로 되면 (E=0일 때) 출력은 고저항값 Z를 갖는다. 표 1은 그림 1에 보인 3상태 버스구동기의 진리표를 보이고 있다. 활성화되었을 때 입력값이 출력으로 버퍼처럼 전달되고, 비활성일 때는 고저항값을 갖는 버스구동기에 대해서도 출력값이 반대인 점을 제외하고는 모든 결과가 동일하게 적용된다.



(그림 1) 두 형태의 3상태 버스구동기
(Fig. 1) 2 types of tristate bus driver

〈표 1〉 3상태 버스구동기의 진리표
〈Table 1〉 Truth table of tristate bus driver

E	input	output
0	0	Z
0	1	Z
1	0	1
1	1	0

〈표 2〉 3값 입력에 대한 3상태 버스구동기의 진리표
〈Table 2〉 Ternary valued truth table of tristate bus driver

E	input	output
0	0	Z
0	1	Z
0	X	Z
1	0	1
1	1	0
1	X	X
X	0	1Z
X	1	0Z
X	X	XZ

동기식 순차회로에 대한 고장 시뮬레이션에는 논리값으로 0, 1 외에도 X가 더 필요하다. 표 2는 3값 (0, 1, X) 입력에 대한 3상태 버스구동기의 진리표를 보이고 있다. 표 1과 마찬가지로, E = 0이면 출력은 Z가 되고, E = 1이면 출력에는 입력이 반전된 값이 나타나며, E = X이면 출력은 입력이 0인 경우 1 또는 Z를 갖고, 입력이 1인 경우 0 또는 Z를 갖고, 입력이 X인 경우는 X 또는 Z를 갖는다. 이를 각각 1Z, 0Z, XZ 라고 표시한다.

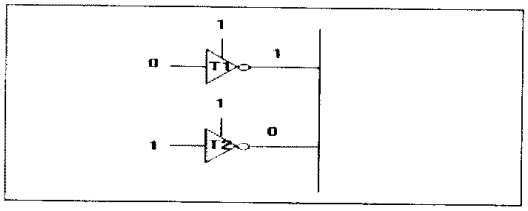
3. 버스충돌

버스는 3상태 버스구동기에 의해서 구동된다. 버스에 연결된 버스구동기는 상호 배타적으로 활성화되어야 한다. 동시에 둘 이상의 버스구동기가 활성화되면 버스충돌이 발생하게 된다. 고장검출을 위한 검사패턴은 정상적인 동작을 위한 패턴이 아니고 고장을 여기시켜서 그 영향을 외부 출력으로 전파시키기 위하여 생성된 패턴이므로 버스충돌을 일으킬 수 있다.

버스충돌에는 버스충돌을 일으킨 버스구동기가 모두 동일한 값을 갖는 경우와 그렇지 않는 경우가 있다. 활성화된 두 개 이상의 버스구동기가 서로 다른 논리값으로 버스를 구동하면, 버스구동기를 통한 VDD에서 Gnd로의 직접적인 전류 경로가 형성된다. 이 경로를 통하여 한계 이상의 전류가 흐르면 경로 상에 있는 버스구동기의 MOS 소자들이 파괴된다. 활성화된 두 개 이상의 버스구동기가 같은 값으로 버스를 구동하는 경우에는 MOS 소자의 파괴는 일어나지 않지만, IDDQ를 증가시켜 고장마스킹(fault masking)을 일으킬 수 있으므로 IDDQ 검사[13]를 위해서는 이러한 버스충돌을 피하여야 한다.

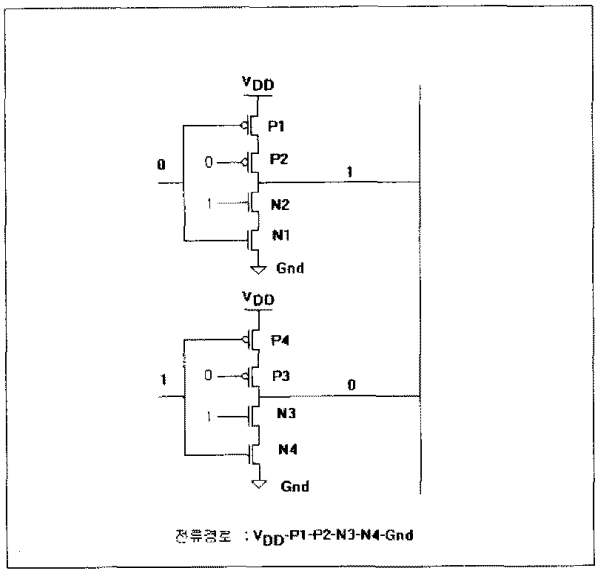
3.1 논리적 버스충돌

한 버스가 두 개의 버스구동기에 의하여 동시에 "0"과 "1"로 구동되면 논리적 버스충돌이 발생되었다고 한다. 그림 2가 이러한 경우를 보이고 있다. 그림에서 두 개의 버스구동기가 모두 활성화되어 위의 버스구동기는 버스를 "1"로 구동하고 아래의 버스구동기는 버스를 "0"으로 구동하고 있다.



(그림 2) 논리적 버스충돌
(Fig. 2) Logical bus conflict

그림 3은 논리적 버스충돌이 일어난 경우의 회로를 보이고 있다. 그림에서 두 개의 버스구동기가 모두 활성화되어 있으므로 입력이 0인 위의 버스구동기는 두 개의 PMOS 소자 P1과 P2를 통하여 VDD를 버스에 연결하고, 입력이 1인 아래의 버스구동기는 두 개의 NMOS 소자 N3과 N4를 통하여 Gnd를 버스에 연결한다. 따라서 P1, P2, N3, N4는 VDD에서 Gnd로의 직접적인 전류 경로를 형성하여 한계 이상의 전류가 흐르면 P1, P2, N3, N4 소자가 파괴될 수 있다.



(그림 3) 논리적 버스충돌에 의하여 형성된 전류 경로
(Fig. 3) Current path formed by logical bus conflict

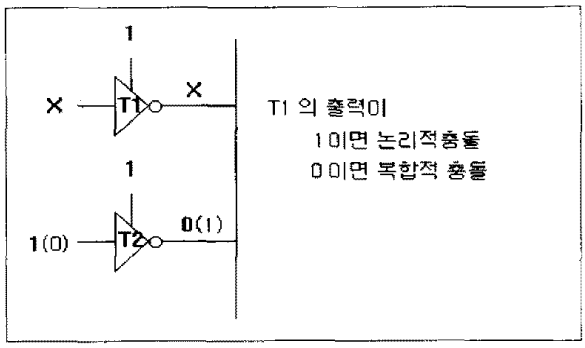
3.2 IDDQ 버스충돌

두 개 이상의 버스구동기가 모두 동일한 논리값으로 버스를 구동하는 경우의 버스충돌을 IDDQ 버스충돌이라고 한다. 즉, 그림 3에서 버스구동기의 입력이 모두 0이거나 1이면, 버스구동기의 출력은 1이거나 0이 되어 버스를 구동하게 된다. 이 경우 버스의 값은 1 또는 0으로 결정되고 논리적 버스충돌의 경우처럼 VDD에서 Gnd로의 직접적인 전류 경로는 형성되지 않으므로 활성화된 버스구동기의 MOS 소자 파괴는 일어나지 않는다. 그러나, IDDQ 버스충돌은 IDDQ의 증가를 초래하여 IDDQ 검사에서 고장마스킹을 일으킬 수 있다. IDDQ 검사에 사용되는 검사 패턴은 IDDQ 충돌을 일으켜서는 안되므로 생성된 검사 패턴이 IDDQ 충돌을 일으키는지 여부를 고장 시뮬레이터를 사용하여 확인하여야 한다.

3.3 복합적 버스충돌

순차회로를 시뮬레이션 또는 고장 시뮬레이션할 때에는 초기화되지 않은 메모리 소자를 취급하기 위해 X값이 사용된다. 시뮬레이션에서의 X값은 실제회로에서는 "0"이나 "1" 값을 갖는다. 즉, 시뮬레이션할 때, 실제 회로에서 "0"이 될지 "1"이 될지 알 수 없음을 나타낸다. 이런 이유로 X를 미지값(unknown value)이라고 한다.

그림 4의 회로에서는 버스구동기 T1과 T2가 모두 활성화되어 있는데 T1은 버스를 X로 구동하고 T2는 버스를 0으로 구동한다. 실제회로에서 X가 0인 경우에는 IDDQ 버스충돌이 일어나고 1인 경우에는 논리적 버스충돌이 일어난다. 이러한 버스충돌을 복합적 버스충돌이라고 한다. T2의 출력이 1인 경우에도 T1의 출력이 X이면 복합적 버스충돌이 발생한다.



(그림 4) 복합적 버스충돌
(Fig. 4) Complex bus conflict

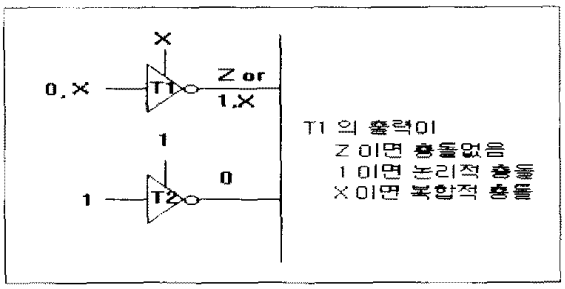
3.4 잠재적 버스충돌

잠재적 버스충돌이란 충돌이 일어나지 않을 수도 있고 충돌이 일어날 수도 있는 경우를 말한다. 충돌이 일어나는 경우에는 앞에서 분류한 충돌 형태중 하나의 버스충돌 형태를 갖는다.

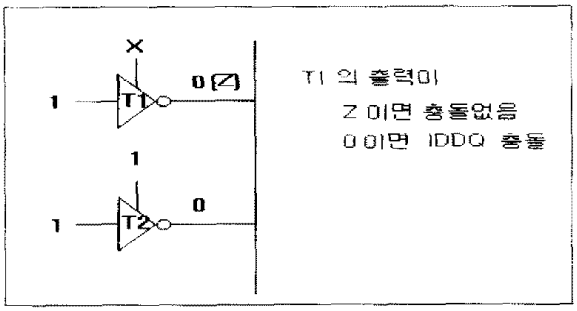
(1) 한 버스구동기의 활성화 입력이 X인 경우

앞에서는 두 개의 버스구동기가 모두 활성화되었음에 반하여 그림 5에서는 3상태 버스구동기 T2는 활성화되었으나 T1의 활성화 입력이 X이어서 T1이 활성화될 수도 있고 활성화되지 않을 수도 있다. T1의 활성화 입력에 있는 X가 실제회로에서 0이라면 버스충돌이 일어나지 않겠지만 1이라면 버스충돌이 발생하게 되므로 잠재적 버스충돌이 된다. 버스충돌이 발생한 경우, T1과 T2의 출력값이 동일하면 IDDQ 버스충돌,

상이하면 논리적 버스충돌, 그리고 어느 하나라도 X이면 복합적 버스충돌이 된다. 즉, 삼태식 버스충돌의 경우 버스충돌이 발생하면 IDDQ, 논리적, 복합 버스충돌이 모두 가능하다.



(그림 5) 잠재적 버스충돌 (논리적/복합적 버스충돌)
(Fig. 5) Potential bus conflict (logical/complex bus conflict)



(그림 6) 잠재적 버스충돌 (IDDQ 버스충돌)
(Fig. 6) Potential bus conflict (IDDQ bus conflict)

그림 5는 논리적/복합적 버스충돌이 가능한 잠재적 버스충돌의 경우를 보이고 있고, 그림 6은 IDDQ 버스충돌을 일으킬 수 있는 잠재적 버스충돌을 보이고 있다. 표 3은 T1의 활성화 입력이 X이고 T2의 활성화 입력이 1인 모든 경우의 버스충돌을 정리한 것이다.

〈표 3〉 그림 5, 6의 회로에 대한 잠재적 버스충돌
〈Table 3〉 Potential bus conflicts for a circuit in Fig. 5 and Fig. 6

T1 출력	T2 출력	버스충돌
0Z	0	충돌없음, IDDQ 충돌
0Z	1	충돌없음, 논리적 충돌
0Z	X	충돌없음, 복합적 충돌
1Z	0	충돌없음, 논리적 충돌
1Z	1	충돌없음, IDDQ 충돌
1Z	X	충돌없음, 복합적 충돌
XZ	0	충돌없음, 복합적 충돌
XZ	1	충돌없음, 복합적 충돌
XZ	X	충돌없음, 복합적 충돌

(2) 두 버스구동기의 활성화 입력이 X 인 경우

〈표 4〉 두 버스구동기의 활성화 입력이 모두 X인 경우의 잠재적 버스충돌
〈Table 4〉 Potential bus conflicts when enable inputs of both drivers are X

T1의 출력	T2의 출력	버스충돌
0Z	0Z	충돌없음, IDDQ 충돌
0Z	1Z	충돌없음, 논리적 충돌
0Z	XZ	충돌없음, 복합적 충돌
1Z	0Z	충돌없음, 논리적 충돌
1Z	1Z	충돌없음, IDDQ 충돌
1Z	XZ	충돌없음, 복합적 충돌
XZ	0Z	충돌없음, 복합적 충돌
XZ	1Z	충돌없음, 복합적 충돌
XZ	XZ	충돌없음, 복합적 충돌

(고장) 시뮬레이션할 때 두 개의 활성화 입력이 모두 X인 경우에는 표 4와 같이 9 가지 경우가 발생한다.

(1), (2)의 경우를 정리한 표 3, 4를 살펴보면 활성화 입력이 X인 경우에는 표 2의 동작표를 사용하여 버스구동기의 출력값을 0Z, 1Z, XZ 로 결정하면, 이 값들을 조사하여 쉽게 버스충돌의 형태를 결정할 수 있음을 알 수 있다.

삼태적 버스충돌의 경우, 버스충돌이 없을 수도 있지만 버스충돌이 발생하면 버스구동기의 파괴를 유발시키거나 IDDQ 고장마스킹을 일으킬 수 있으므로, 생성된 검사 패턴이 잠재적 버스충돌을 일으킬 수 있는지의 여부를 고장 시뮬레이션을 통하여 확인하여야 한다.

4. 버스충돌을 찾는 알고리즘

3절에서는 버스충돌의 여러 가지 경우에 대하여 살펴보았다. 이 절에서는 이러한 버스충돌을 확인하는 알고리즘을 설명한다. 이 알고리즘은 버스에 연결된 3상태 버스구동기의 값을 조사함으로써 버스충돌을 조사하게 된다. 그러므로, 이 알고리즘을 적용하기 전에 버스에 연결된 모든 버스구동기의 값이 정확하게 계산되어야 한다. 3상태 버스구동기의 값 계산은 2절에서 설명되었다. 3상태 버스구동기가 가질 수 있는 값은 모두 7 개로서, E = 1일 때의 0, 1, X와 E = 0일 때의 Z, 그리고 E = X일 때의 0Z, 1Z, XZ이다.

표 5는 7값중 하나를 갖는 두 버스구동기가 버스에

연결되었을 때의 버스의 상태를 나타내고 있다. 이 표의 각 항목은 다음과 같이 구분된다

비충돌값 : 0, 1, X, Z 등과 같이 표에서 단독으로 나타나며 버스충돌을 일으키지 않는다.

충돌값 : (0,0), (0,1), (1,0), (X,0), (0,X), (X,1), (1,X), (1,1) 등과 같이 괄호 안에 쌍으로 나타난다. 괄호 안의 두 값이 (0,0)이나 (1,1)처럼 동일한 이진값(0 또는 1)이면 IDDQ 충돌을 나타내고, (0,1)이나 (1,0)처럼 서로 다른 이진값을 가지면 논리적 충돌을 나타낸다. 그리고 (X,0), (1,X), (X,X) 등과 같이 X 값을 포함하고 있으면 복합적 버스충돌을 나타낸다. 괄호로 표시된 충돌값은 위 첨자 I, L, C를 사용하여 IDDQ, 논리적, 복합적 버스충돌로 구분하였다.

잠재적 버스충돌 : 표 안의 항목이 비충돌값과 충돌값을 동시에 가지고 있으면 이는 잠재적 충돌에 해당된다. 예로, {0, $(0,0)^I$}은 비충돌값 0과 IDDQ 충돌값 (0,0)I이 가능한 잠재적 버스충돌이다.

버스구동기에 의한 버스값의 결정은 표 5를 사용하여 할 수 있다. 다음은 확인하고자 하는 버스충돌의 유형에 따른 표 5의 적용을 설명하고 있다.

4.1 모든 버스충돌을 허용하지 않는 경우

표 6은 모든 버스충돌을 허용하지 않는 경우의 버스값 계산을 위하여 표 5를 변형한 것이다.

이 표에서는 7 값 0, 1, X, Z, 0Z, 1Z, XZ를 제외한

〈표 6〉 버스 상태표 (모든 버스충돌 불허)
〈Table 6〉 Bus status table (any bus conflicts not allowed)

	0	1	X	Z	0Z	1Z	XZ
0	I0	L	C	0	PI0	PL	PC
1	L	I1	C	1	PL	PI1	PC
X	C	C	C	X	PC	PC	PC
Z	0	1	X	Z	0Z	1Z	XZ
0Z	PI0	PL	PC	0Z	PI0Z	PL	PC
1Z	PL	PI1	PC	1Z	PL	PI1Z	PC
XZ	PC	PC	PC	XZ	PC	PC	PC

모든 항목은 버스충돌을 나타내며 각 항목의 부호는 다음과 같은 의미를 갖는다.

I0, I1 : IDDQ 버스충돌이며, 버스값은 각각 0과 1이다.
L : 논리적 버스충돌을 나타낸다.
C : 복합적 버스충돌을 나타낸다.
PI0, PI1, PI0Z, PI1Z, PL, PC : 잠재적 버스충돌을 나타낸다. PI0과 PI1은 버스충돌 여부에 상관없이 버스값이 각각 0과 1임을 나타낸다. PI0Z와 PI1Z는 각각 버스충돌 여부에 상관없이 버스값이 0Z와 1Z가 됨을 나타낸다. PL과 PC는 버스충돌 발생시 각각 논리적 버스충돌과 복합적 버스충돌임을 나타낸다.

버스구동기가 3개 이상일 경우에는 버스충돌이 나타나거나 모든 버스구동기 값이 조사될 때까지 표 6을 반복 적용한다. 그 다음 버스값이 0Z, 1Z, XZ는 모두 버스값을 X로 변환하고 시뮬레이션이나 고장 시뮬레이션을 계속한다.

〈표 5〉 버스구동기의 값에 따른 버스상태
〈Table 5〉 Bus status for the combinations of bus driver output values

	0	1	X	Z	0Z	1Z	XZ
0	$(0,0)^I$	$(0,1)^L$	$(0,X)^C$	0	0, $(0,0)^I$	0, $(0,1)^L$	0, $(0,X)^C$
1	$(1,0)^L$	$(1,1)^I$	$(1,X)^C$	1	1, $(1,0)^L$	1, $(1,1)^I$	1, $(1,X)^C$
X	$(X,0)^C$	$(X,1)^C$	$(X,X)^C$	X	X, $(X,0)^C$	X, $(X,1)^C$	X, $(X,X)^C$
Z	0	1	X	Z	0Z	1Z	XZ
0Z	0,$(0,0)^I$	1,$(0,1)^L$	X,$(0,X)^C$	0Z	0, Z,$(0,0)^I$	0,1,Z,$(0,1)^L$	0,X,Z,$(0,X)^I$
1Z	0,$(1,0)^L$	1,$(1,1)^I$	X,$(1,X)^C$	1Z	0,1,Z,$(1,0)^L$	1,Z,$(1,1)^I$	1,X,Z,$(1,X)^C$
XZ	0,$(X,0)^C$	1,$(X,1)^C$	X,$(X,X)^C$	XZ	0,X,Z,$(X,0)^C$	1,X,Z,$(X,1)^C$	X,Z, $(X,X)^C$

4.2 IDDQ 버스충돌만 허용하는 경우

〈표 7〉 버스 상태표 (IDDQ 버스충돌 허용)
〈Table 7〉 Bus status table
(Only IDDQ bus conflict allowed)

	0	1	X	Z	0Z	1Z	XZ
0	0	L	C	0	0	PL	PC
1	L	0	C	1	PL	1	PC
X	C	C	C	X	PC	PC	PC
Z	0	1	X	Z	0Z	1Z	XZ
0Z	0	PL	PC	0Z	0Z	PL	PC
1Z	PL	1	PC	1Z	PL	1Z	PC
XZ	PC	PC	PC	XZ	PC	PC	PC

IDDQ 검사를 수행하지 않는 경우에는 IDDQ 버스충돌을 허용할 수 있다. IDDQ 버스충돌을 허용하는 경우, I0, I1 IDDQ 버스충돌은 버스값을 각각 0과 1로 만들고, PI0, PI1 잠재적 버스충돌은 버스값을 각각 0과 1로 만든다. 그리고 PI1Z는 버스값을 1Z로, PI0Z는 버스값을 0Z로 만든다. 표 7은 표 6을 이렇게 수정한 버스 상태를 나타낸다.

4.1의 경우와 마찬가지로 3 개 이상의 버스구동기가 버스에 연결되어 있는 경우는 IDDQ 버스충돌 이외의 버스충돌이 나타나거나 모든 버스구동기의 값이 조사될 때까지 표 7을 반복 적용하고 최종 버스값이 0Z, 1Z, XZ인 경우에는 버스값을 모두 X로 수정한 다음 시뮬레이션이나 고장 시뮬레이션을 계속한다.

4.3 모든 버스충돌을 허용하는 경우

경우에 따라서는 버스충돌에도 불구하고 버스구동기가 파괴되지 않는 수도 있다. 이런 경우, 사용자의 목적에 따라 버스충돌이 일어나더라도 고장 검사를 계속 수행할 수 있다. 이를 위해서는 버스충돌이 일어나더라도 논리 시뮬레이션이나 고장 시뮬레이션을 계속 수행하여야 한다. IDDQ 버스충돌을 제외한 다른 버스충돌이 일어난 경우에는 버스값을 결정할 수 없으므로 버스값을 미지값 X로 수정하고 시뮬레이션이나 고장 시뮬레이션을 계속한다. 표 8은 이렇게 수정한 버스 상태표를 보이고 있다.

그림 7은 버스값을 계산하고 버스충돌을 확인하는 알고리즘을 나타내고 있다. 변수 bus_val을 Z로 초기화한 다음, 버스충돌 허용 정도에 따라 표 6, 7, 8중 하나를 사용하여 각 버스구동기의 driver_val과 bus_val

〈표 8〉 버스 상태표 (모든 버스충돌 허용)
〈Table 8〉 Bus status table (All bus conflicts allowed)

	0	1	X	Z	0Z	1Z	XZ
0	0	X	X	0	0	X	X
1	X	0	X	1	X	1	X
X	X	X	X	X	X	X	X
Z	0	1	X	Z	X	X	X
0Z	0	X	X	X	X	X	X
1Z	X	1	X	X	X	X	X
XZ	X	X	X	X	X	X	X

에 대한 새로운 bus_val을 구한다. 만일 bus_val이 0, 1, X, Z, 0Z, 1Z, XZ 중 하나의 값을 가지면 버스충돌이 없는 경우이므로 for 루프를 사용하여 버스 상태표를 계속 적용한다. 만일 7 값중 하나를 갖지 못하면 버스충돌이 일어난 것이며 이 때의 버스충돌의 형태는 각 버스 상태표에 의해 확인된다. 버스충돌이 일어난 경우에는 시뮬레이션을 중단한다. 버스충돌이 없이 for 루프가 완료되면 버스값을 조사하여 만일 0Z, 1Z, XZ의 값을 갖는다면, 실제의 회로는 이런 값을 가질 수 없으므로 이 값들을 모두 X로 변환한 다음 시뮬레이션이나 고장 시뮬레이션을 수행한다.

```
bus_val := Z;
for each bus driver
   bus_val := Table(bus_val, driver_val)
   if bus_val not in { 0, 1, X, Z, 0Z, 1Z, XZ }
      BusConflictFound;
   endif
endfor
if bus_val is in {0Z, 1Z, XZ}
   bus_val := X;
endif
```

(그림 7) 버스충돌을 확인하는 알고리즘
(Fig. 7) Algorithm for identifying bus conflicts

모든 버스충돌이 허용된 경우에는 버스충돌이 검사되지 않으며, IDDQ 버스충돌만 허용된 경우에는 IDDQ 버스충돌은 검사되지 않고 버스값은 자동적으로 결정된다. 그러므로, 허용되지 않은 버스충돌의 경우에만 시뮬레이션을 중단한다.

5. 고장 시뮬레이션에서의 고장 전파와 검사 패턴 생성

5.1 3상태 버스구동기에서의 고장 전파

어떤 고장에 대한 3상태 버스구동기의 입력과 제어 입력의 값이 결정되면, 표 2를 사용하여 그 고장에 대한 출력값을 구한다. 즉, 활성화 입력이 1인 경우 입력에 도달한 고장리스트를 모두 출력으로 전파한 다음, 3상태의 구동기의 값에 따라 지역고장(local fault)을 삽입한다. 즉, 3상태 구동기의 값이 0이면 1-고착고장(stuck-at-1 fault)을 삽입하고, 1이면 0-고착고장(stuck-at-0 fault)을 삽입한다. 고장 시뮬레이션을 단순하게 하기 위하여 버스구동기의 제어입력을 통한 고장 전파는 하지 않는다.

5.2 버스에서의 고장 전파

버스구동기에 도달한 각 고장값에 대하여 버스의 상태표 표 6, 7, 8을 사용하여 버스값을 구하면, 그 고장이 버스충돌을 일으키는지의 여부를 알 수 있다. 버스충돌을 일으키는 경우, 그 고장에 대한 버스값이 결정되지 못하므로 그 고장의 전파는 멈추게 된다. 또한 정상적인 버스값이 X인 경우에도 버스를 통한 고장전파는 중단된다. 충돌이 없는 경우, 그 고장 값이 정상값과 다른 값을 가지게 되면 그 고장은 버스를 통하여 입력에서 출력으로 전파된다.

5.3 버스충돌이 없는 검사패턴의 생성

주어진 검사패턴에 대한 고장 시뮬레이션 도중 버스충돌 가능성에 대한 경고가 있으면 버스구동기의 파괴나 IDDQ 고장 마스킹을 방지하기 위하여 그 검사패턴을 고장 검사에 사용하여서는 안된다. 따라서 버스충돌이 확인된 패턴은 버스충돌이 일어나지 않도록 수정되어야 한다. 제안된 버스충돌 확인 방법을 검사패턴 생성기에 도입하면 검사 패턴 생성과정중에 버스충돌이 확인하여 버스충돌을 피할 수 있도록 검사패턴을 수정생성할 수 있다. 즉, 제안된 버스충돌 확인 방법은 고장 검사패턴 생성기와 연계하여 사용하면 버스충돌이 없는 고장 검사패턴을 생성할 수 있다.

6. 결 론

버스구조를 가지고 있는 회로에 검사패턴을 적용할 경우 일어날 수 있는 여러 형태의 버스충돌에 관하여 살펴보았다. 버스구조를 가지고 있는 회로는 버스구동기의 활성화 입력값과 데이터 입력값에 따라 다양한 형태의 버스충돌이 일어날 수 있는데, 버스충돌을 IDDQ 버스충돌과 논리적 버스충돌, 복합적 버스충돌로 구분하였다. 또한 충돌 가능성에 따라 잠재적 버스충돌을 구분하였다. 잠재적 버스충돌에서도 버스충돌이 발생하면 IDDQ, 논리적, 복합적 버스충돌이 되므로 잠재적 버스충돌도 피해야 된다. 버스충돌을 확인하는 간단하고 효과적인 방법을 제시하였다. 제안된 방법을 사용하여 구현된 고장 시뮬레이터는 사용자의 선택에 따라 모든 버스충돌을 허용하지 않을 수도 있고, IDDQ 버스충돌만 허용할 수 있고, 모든 버스충돌을 허용할 수도 있다. 또한, 버스충돌이 없는 고장 패턴 생성기의 가능성에 대하여도 고찰하였다.

지금까지 발표된 고장 시뮬레이터는 4값 (0, 1, X, Z) 시뮬레이션을 정확하게 하지 못하거나 사용한 예를 보이지 않았으며, 버스구조를 가지고 있는 회로 및 버스충돌을 다루는 고장 시뮬레이터는 많지 않았다.

현재 공개된 벤치마크회로가 3상태 버스구동기와 버스 구조를 갖지 않은 것이어서, 제안된 고장 시뮬레이터를 직접 실험할 수는 없었으나 버스구조를 갖는 벤치마크회로가 공개되면, 이를 사용하여 제안된 고장 시뮬레이터를 개선하고, 검사패턴 생성기와 연계하여 버스충돌이 없는 검사패턴을 생성할 수 있는 자동 검사패턴 생성기에 대한 연구가 가능할 것이다.

참 고 문 헌

[1] S. Seshu, "On an Improved Diagnosis Program," IEEE Trans. on Electronic Computers, Vol.EC-12, No.2, pp.230-238, June, 1977.

[2] D. B. Armstrong, "A Deductive Method of Simulating Faults in Logic Circuits," IEEE Trans. on Computer-Aided Design, Vol.C-21, No.5, pp.464-471, May,1972.

[3] E. G. Ulrich and T. G. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," IEEE Design & Test of Computers, Vol.1, No.3, pp.66-75, August, 1984.

[4] W. T. Cheng and J. H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simu-

lator, IEEE Design Automation Conference, pp.535-540, June 1990.

[5] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," IEEE Design Automation Conference, pp.336-340, June 1992.

[6] K. Kim and K. K. Saluja, "HYSIM: Hybrid Fault Simulation for Synchronous Sequential Circuits," VLSI Design, International Journal of Custom Chip Design, Vol.4, No.3, pp.181-197, July 1996.

[7] W. Cheng and M. L. Yu, "Differential Fault Simulation for Sequential Circuits," Journal of Electronic Testing: Theory and Applications, pp.7-13, 1990.

[8] N. Gouders and R. Kaibel, "PARIS: A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits," IEEE Int. Conf. on Computer Aided Design, pp.542-545, November 1991.

[9] K. J. Antreich, M. H. Schulz, "Fast Fault Simulation in Combinational Circuits", IEEE Int. Conf. on CAD, ICCAD-86, pp.330-333, Nov. 1986.

[10] K. J. Antreich, M. H. Schulz, "Accelerated Fault Simulation and Fault Grading in Combinational Circuits", IEEE Trans. on CAD, Vol.CAD-6, pp.704-712, 1987.

[11] M. H. Schulz, D. Pellkofer, "A Three-Valued Fast Fault Simulator For Scan-Based VLSI-Logic", Proc. 1st European Test Conf., pp.41-48, April 1989.

[12] B. Underwood, J. Ferguson, "The Parallel Test-Detect Fault Simulation Algorithm", Proc. International Test Conf., pp.712-717, 1989.

[13] M. H. Konijnenburg, J. Th. van der Linden, and A. J. van der Goor, "Test Pattern Generators with Restrictors", Proc. Int. Test Conference, pp.598-605, Nov. 1993.

[14] J. Th. van der Linden, M. H. Konijnenburg, and A. J. van der Goor, "Test Generation and Three-state Elements, Buses, and Bidirectionals", Proc. 12th IEEE VLSI Test Symp., pp.114-121, April, 1994.

[15] R. K. Gulati, W. Mao and D. K. Goel, "Detection of Untestable Faults Using IDDQ Testing", Proc. International Test Conf., pp.770-777, 1992.

[16] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in FORTRAN", Proc. Int. Symp. on Circuits And Systems, Special session, June 1985.

[17] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. Int. Symp. on Circuits And Systems, pp.1929-1934, May 1989.

김 규 철

1978년 서울대학교 자연과학대학 물리학과(학사),

1980년 서울대학교 물리학과 대학원(이학석사),

1986년 미국 위스콘신 대학(전기 컴퓨터공학 석사),

1992년 미국 위스콘신 대학(전기 컴퓨터공학 박사),

1992년 미국 위스콘신 대학 연구원,

1993년 삼성전자 마이크로본부 선임연구원,

1993년~현재 단국대학교 전자컴퓨터공학부 근무(조교수). 연구관심분야는 VLSI 설계, VLSI 검사, 고장 시뮬레이션, 검사패턴 자동생성, Built-In Self Test, Design for Testability, 지연고장 검사