표준 스캔 설계 환경을 위한 경로 지연 고장 검사입력 생성기

(A Path-Delay Fault Test Generator for Standard Scan Design Environments)

康 容 碩 *, 姜 成 昊 *

(Yong Seok Kang and Sungho Kang)

社團法人 大韓電子工學會

論文96-33A-9-22

표준 스캔 설계 환경을 위한 경로 지연 고장 검사입력 생성기

(A Path-Delay Fault Test Generator for Standard Scan Design Environments)

康 容 碩 *, 姜 成 昊 *

(Yong Seok Kang and Sungho Kang)

요 약

회로의 정확한 동작을 위해서는 회로의 모든 경로에 대한 전달 지연 시간이 주어진 클럭 주기보다 적아야 한다. 제조된 칩이 이러한 조건을 만족하는 가를 확인하기 위해 정상적인 클럭 주기로 동작하는 회로에 검사입력을 가하여 회로를 검사한다. 본 논문에서는 회로의 지연 결함을 효율적으로 나타낼 수 있는 경로 지연 고장 모델을 이용하여 표준 스캔 설계 환경에서 동작하는 검사입력 생성기를 제안하였다. 표준 스캔 환경에서 어려운 문제인 경성 검사입력의 생성의 보장과 검사입력 생성의 효율성 향상을 위하여 20개의 논리값을 사용한 새로운 알고리즘을 개발하였다. 결과에서는 다양한 종류의 검사입력의 효과적 생성과 메모리 사용이 효율적인 동작을 보여준다.

Abstract

Correct operation of a logic circuit requires propagation delays of all paths in the circuit to be smaller than the intended clock interval. Tests can be used to insure that path delays in manufactured circuits meet the specifications at the normal clocking rate. This paper describes a test generator for standard scan environments using path-delay fault model to handle delay defects effectively. A new algorithm based on 20-valued logic is developed to guarantee robustness of tests and increase the efficiency of test generation. Results show the effective test generation of various types and memory-efficient operation.

I. 서 론

논리 회로가 정확히 동작하는 것을 보장하기 위해서는 회로의 기능적 특성(functional characteristics)의 검증뿐만 아니라 회로를 통해 신호의 전달 지연(propagation delay) 시간이 특정한 클럭 사이클 안에 정상적으로 동작하는 가를 검증하여야 한다. 한편 더욱 더 성능이 우수한 VLSI 칩의 개발의 필요성은 꾸준히 증가하고 있으며 고속 동작을 위한 회로 설계에서 최악의 경우의 동작 시간과 지연 시간을 고려하기보다는 확률적인 동작 시간 및 지연 시간 분석을 고려하는 것이 기본이 되는 추세이다. 따라서 지연 고장 검사는 큰 중요성을 갖게 된다. 지연 고장 검사는 주어진 클럭 주기 안에 회로가 정확하게 동작하는가를 검사하는 것으로, 신호의 전이(transition)가 주어진 시간 안에 회로를 통해 전달되는가를 판별해야하므로 적어도 두 개의 검사입력을 필요로 한다.

지연 고장 모델은 크게 나누어서 게이트(gate) 지연 고장 모델[1]과 경로 지연 고장 모델[2]로 나눌 수 있다. 게이트 지연 고장 모델은 지연 고장이 게이트의 입력이나 출력에서 발생하여 특정 시간 범위를 벗어나게 된다고 모델링한 것으로 이 모델로는 각 게이트에서 발생하는 작은 지연이 누적되어 일어나는 지연 고장을 검사할 수 없다. 반면에 경로 지연고장 모델은 검사되

* 正會員, 延世大學校 電氣工學科
(Dept. of Electrical Eng., Yonsei University)

接受日字: 1996年2月17日, 수정완료일: 1996年8月26日

고 있는 전체 경로상에 누적된 작은 지연으로 인해서 회로가 주어진 동작 주파수 안에 정상 동작을 할 수 없는 가능성을 가정한다. 게이트 지연 고장 모델은 고착 고장과 비슷하게 고장의 수를 게이트의 수에 따라 정량적으로 나타낼 수 있지만 경로 지연 고장 모델은 회로가 커질수록 경로의 수가 기하급수적으로 늘어나 모든 경로에 대한 검사입력을 생성하는 것이 불가능하다. 이러한 단점에도 불구하고 게이트 지연 고장 모델이 국소화된(localized) 지연고장만을 검사할 수 있는데 반해 경로 지연고장 모델은 국소화된 지연뿐만 아니라 분포된(distributed) 지연고장도 검사할 수 있는 장점을 가져 본 논문에서는 경로 지연고장 모델을 사용한다.

순차회로에서 고착고장을 검사하기 위해 플립플롭을 직렬로 연결하고 각 플립플롭의 초기값을 외부에서 입력할 수 있게 하여 제어가능성(controllability)과 탐지가능성(observability)을 높인 스캔 플립플롭을 사용한 설계를 많이 사용하고 있다. 스캔(scan) 설계를 사용하여 고착 고장(stuck-at fault)을 검사하는데 많은 이득을 얻게 되어 많은 회로에서 쓰이고 있으나 경로 지연 고장 검사에는 큰 도움이 되지 못했다. 따라서 스캔 환경을 위한 경로 지연 검사용 검사입력 생성기의 개발은 큰 중요성을 갖는다.

표준 스캔 환경에서 동작하는 경로 지연 고장을 위한 자동 검사입력 생성기의 개발은 마이크로프로세서를 비롯한 많은 종류의 칩을 설계할 때 클럭 주파수에 근접하는 경로상의 미묘하지만 해로울 수 있는 공정의 변화를 관찰할 수 있는 기법을 제공할 수 있고 가장 긴 경로를 검사하여 칩을 동작 속도에 따라 분류할 수 있는 등 많은 필요성을 갖는다.

Ⅱ. 이론적 배경

동작 관점에서 경로 지연고장 검사의 최종 목표는 주어진 회로에서 모든 경로의 전달 지연 시간이 시스템의 클럭 주기보다 작다는 것을 보장하는 것이다. 동적(dynamic) 논리 회로를 제외하고 각각의 지연 고장을 검사하기 위해서는 고착고장에서 사용한 단패턴(single pattern) 검사와 달리 쌍패턴(two pattern) 검사를 사용해야 한다는 것은 잘 알려진 사실이다[2,3]. 쌍패턴 검사에서 첫 번째 시간 프레임의 입력 벡터를 초기화 벡터라 하며 이를 사용하여 회로를 초기화시킨 후에 두 번째로 전달 벡터를 입력하여 검사를 행하는 경로를 통해 신호의 전이가 전달되어 회로의 최종 출력단(primary output)이나 스캔이 가능한 출력 래치(latch)의 입력에 나타나도록 하여 시스템 클럭 안에 회로가 동작하는 가를 판별하게 된다.

경로 지연고장을 검사하기 위해 생성된 검사입력은 여러 종류가 있다. HFR(hazard-free robust) 검사입력은 경로상의 신호가 동적 해저드(dynamic hazard)를 갖지 않고, 또한 회로의 나머지 부분의 지연과는 무관하게 경로 지연고장를 검출할 수 있음을 보장하는 쌍패턴 검사입력(two pattern test)을 말한다. 경성(robust) 검사입력은 특정한 경로의 지연 고장을 다른 회로의 지연과 무관하게 검출할 수 있다는 것만 보장하는 쌍패턴 검사입력이다[3]. 경성(ROB) 검사입력의 조건을 만족시키지 못하면 연성(non-robust) 검사입력[3]이라고 하는데 WNR(weak non-robust) 검사입력은 만약 경로상의 전이가 각각의 회로 소자에 도달하기 전에 경로외(off-path)의 모든 입력들이 최종값들로 안정된다는 조건을 만족하면 지연고장을 검출할 수 있음을 보장하는 쌍패턴 검사입력을 말한다[4]. SNR(strong non-robust) 검사입력은 경로외의 입력들이 정적 해저드(static hazard)를 갖지 않으면 경성 검사입력이 되고 그렇지 않으면 WNR가 되는 검사입력을 말한다[4]. 토글(toggle) 검사입력은 경로상에 있는 소자의 출력에 입력의 전이가 나타나게 하는 과정만을 거쳐 최종 출력까지 전이를 전달시키는 검사입력을 말한다.

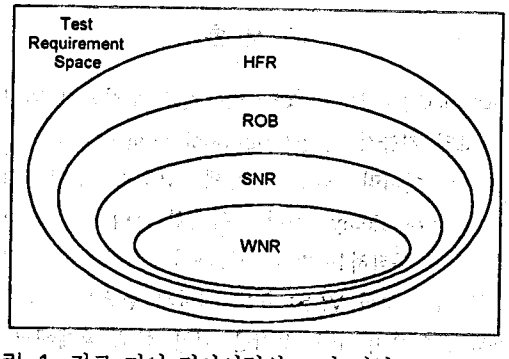

그림 1. 경로 지연 검사입력의 조건 집합
Fig. 1. Requirement Set for Path-Delay Tests.

HFR, ROB, SNR, WNR 4가지 종류의 검사입력의 정의에 의해 부과된 조건들은 그림 1과 같이 계층적으로 나타낼 수 있다. 이 계층적 조건은 각 검사입력의

검출 능력과 더불어 경로 지연고장을 위한 검사입력 생성에 특별한 접근법을 제공한다. 이 접근법은 그림 1에 나타낸 검사입력 조건 공간을 최대한 점유하기 위하여 가장 많은 조건을 만족하는 검사입력의 생성을 우선적으로 시행한다.

검사 대상 칩의 고장 진단이 목적이 아닌 고장 경로의 검출을 목적으로 하는 검사입력 생성시에, HNR 검사입력은 생성하지 않고 각 경로에 대해 ROB 검사를 먼저 시도한다. 만약 ROB 검사입력을 생성하지 못하면 해당 경로에 대해 SNR 검사를 행한다. 또다시 생성에 실패하면 WNR 검사입력의 생성을 시도한다. 하지만 지연 고장의 진단을 목적으로 할 때는 HFR 검사입력의 생성을 가장 먼저 시도한다. 회로상에 존재하는 모든 경로에 대한 검사입력의 생성은 실현 불가능하므로 상용 타이밍 분석 장치(timing analyzer)를 사용하여 회로에서 가장 긴 경로를 찾아 검사입력을 생성한다. 고장경로의 검출을 목적으로 할 경우 HFR는 필요없으나 많은 종류의 검사입력을 생성하기 위해 본 연구에서는 HFR, ROB, SNR, WNR 등 4가지 검사입력을 사용한다.

두 개의 시간 프레임을 동시에 표현하는 논리값을 사용하여 쌍패턴 검사를 다룰 수 있는 몇 가지 방법이 있다. 스캔 설계 환경하에서 동작하는 대부분의 검사입력 생성기는 특별한 스캔 플립플롭을 사용하여 지연고장 검사를 위해 필요로 하는 연속적인 클럭 사이클의 검사 입력값들을 스캔하는 확장된 스캔(enhanced scan) 설계 환경을 위해서 고안되었다. 이 방법은 그림 2에 나타낸 것과 같이 래치를 한 개 더 추가하여 두 번째 입력 벡터가 스캔되는 동안 첫 번째 벡터를 래치가 유지할 수 있게 하여 두 개의 벡터를 스캔 플립플롭을 통하여 외부에서 입력 받을 수 있게 하였다. 이러한 방법은 확대된 스캔 소자를 사용해야 하므로 회로의 면적을 크게 하고 검사 시간이 오래 걸리는 단점을 갖게 되어 실제 회로 설계에서는 사용되지 않는다. 또한 표준 스캔 환경에서 스캔 이동(scan shifting)을 사용하는 방법도 있다[1]. 이 방법은 직렬로 연결된 스캔 플립플롭(scan chain)에 첫 번째 시간 프레임의 검사입력값을 스캔한 후 플립플롭에 클럭을 가해 회로를 초기화시키고 두 번째 시간 프레임의 검사입력값은 플립플롭에 저장되어 있는 값들을 스캔 체인(scan chain)을 통해 이동시켜 사용하는 방법이다. 이러한 방법을 사용하면 이동된 입력값의 상호연관성으로 인해 검사가능한 경로를 검사가 불가능하다고 처리할 수 있다. 따라서 이 두가지 방법은 본 연구에서 사용하지 않는다.

본 논문에서 제안한 경로 지연고장에 대한 검사입력 생성은 표준 스캔 환경하에서 수행된다. 즉, 순수 조합회로를 위해 고안된 기존의 검사입력 생성 방법은 적용될 수 없다. 따라서 각 쌍패턴 지연고장 검사입력의 초기화 벡터는 스캔 플립플롭을 통해 입력할 수 있으나 두 번째 벡터는 회로의 함수(function)에 의해 결정되므로 검사입력의 생성을 위해 필요한 것은 두 클럭 사이클동안 검사입력의 조건을 추적하면 된다. 이러한 접근 방법을 함수적 지정(functional justification)이라고 한다. 이 방법을 사용하기 위한 몇몇 방법이 소개되었지만 너무 단순화시켜 문제에 접근하여 ROB 또는 HFR를 보장할 수 없는 결과를 초래하였다. III과 IV절에서 경성 검사입력을 보장할 수 있는 새로운 논리값과 알고리즘을 제시하였다.

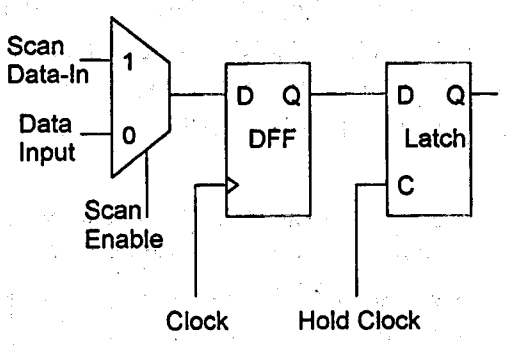

그림 2. 확대된 스캔 소자
Fig. 2. Augmented Scan Element.

III. 논리값

조합회로의 경우 경로 지연고장 검사입력의 생성을 위해 5개의 논리값(logic value), {S0, S1, X0, X1, X}, 을 사용했다. 여기서 S는 두 시간 프레임동안 안정(stable)함을 의미한다. 그러나 이 값들로 완전 스캔 환경에서 함수적 지정 방법을 사용하기 위해서는 불충분하다. 함수적 지정 방법은 단지 두 개의 시간 프레임으로 제한돼 있다는 것을 제외하면 순차회로의 검사입력 생성을 위해 사용하는 시간 프레임 확장기술과 비슷하다. 따라서 플립플롭의 출력에 X0을 지정(justification)하기 위해서는 플립플롭의 입력에 0X라는 값이 필요하다. 이와 비슷한 이유로 함수적 지정을

위해서는 최소한 11개의 논리값(표 1에서 00부터 S1까지)이 필요하다.

표 1. 20가지 논리값
Table 1. 20-Valued Logic.

00	11	01	10	X0	X1	0X	1X	XX
S0	S1							
00T	11T			X0T	X1T	0XT	1XT	
XT0	XT1							XTB

앞서 정의한 논리값의 의미를 세분화하기 위해 9개의 논리값(표 1에서 00T부터 XTB까지)을 추가로 사용한다. 이들 논리값에서 T는 "전이 가능"을 나타낸다. 이 논리값들은 앞서 소개한 5개의 논리값으로 지정할 수 없는 값을 명확(specification)하게 할 수 있고, 따라서 이를 사용하면 불완전하게 명기된 논리값을 갖는 노드(node)에서도 검사입력을 생성시 지정 과정에서의 성공적이지 못한 검색으로 인한 시간 소비를 줄일 수 있게 해준다. 예를 들어 OR 게이트의 모든 입력이 XT1면 이 값들은 출력값으로 XT1을 유추(implication)한다. 이것으로 OR 게이트의 출력값을 정확히 결정할 순 없지만 이 게이트의 출력이 S1로 지정될 수 없다는 알 수 있다. 표 1의 첫 줄은 {0, 1, X}로 만들 수 있는 모든 조합을 나타낸 것으로 앞의 글자가 초기화 시간 프래임의 값을 나타내고 두 번째는 최종 시간 프래임을 나타낸다. 아직 이 논리값들로는 신호의 해저드의 유무를 나타낼 수 없다. {S0, S1}은 두 시간 프래임 모두에서 각각 {0, 1}을 나타내는 것인데 신호에 해저드가 없음을 보장한다. {00T, 11T, X0T, X1T, 0XT, 1XT}는 각각 {00, 11, X0, X1, 0X, 1X}와 비슷하나 신호에 해저드가 없다고 보장할 수 없다는 의미를 추가한다.

두 개의 시간 프래임에서 모두 0값을 갖는 논리값이 3개, {00, S0, 00T}, 가 있다. 이들 사이의 차이점은 신호에 정적 해저드의 유무에 관한 정보와 관련되어 있다. 예를 그림 3에 나타내었다. 그림 3의 게이트 B에서 명확하지 않은(unspecified) 입력은 게이트의 출력값에 정적 해저드의 유무를 결정할 수 없다. 게이트 A의 출력값은 정적 해저드를 갖지 않는다고 보장됨을 나타낸다. 게이트 E에서는 게이트의 출력값에 해저드가 없음을 보장할 수 없음을 나타낸다. 따라서 00은 S0과 00T의 합집합이고 마찬가지로 11은 S1과 11T의 합집합이다.

표 2. NOR 게이트에 대한 논리 연산표
Table 2. Implication Table for an NOR Gate.

NOR	00	11	01	10	X0	X1	0X	1X	XX	S0	S1	00T	11T	X0T	X1T	0XT	1XT	XT0	XT1	XTB
00	11	00	10	01	X1	X0	1X	0X	XX	11	S0	11T	00T	X1T	X0T	1XT	0XT	XT1	XT0	XTB
11	00	00	00	00	00	00	00	00	00	00	S0	00	00	00	00	00	00	00	00	00
01	10	00	10	00T	X0T	X0	10	00	X0	10	S0	10	00T	X0T	X1T	10	00T	X0	X0T	X0T
10	01	00	00T	01	01	00	0XT	0X	0X	01	S0	01	00T	01	00T	0XT	0XT	0X	0XT	0XT
X0	X1	00	X0T	01	X1	X0	XT0	0X	XX	X1	S0	X1T	00T	X1T	X0T	XTB	0XT	XT1	XT0	XTB
X1	X0	00	X0	00	X0	X0	X0	00	X0	X0	S0	X0	00	X0	X0	X0	00	X0	X0	X0
0X	1X	00	10	0XT	XT0	X0	1X	0X	XX	1X	S0	1XT	00T	XTB	X0T	1XT	0XT	XT1	XT0	XTB
1X	0X	00	00	0X	0X	00	0X	0X	0X	0X	S0	0X	00	0X	00	0X	0X	0X	0X	0X
XX	XX	00	X0	0X	XX	X0	XX	0X	XX	XX	S0	XT1	00	XT1	X0	XT1	0X	XT1	XX	XT1
S0	11	00	10	01	X1	X0	1X	0X	XX	S1	S0	11T	00T	X1T	X1T	1XT	0XT	XT1	XT0	XTB
S1	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0
00T	11T	00	10	01	X1T	X0	1XT	0X	XT1	11T	S0	11T	00T	X1T	X0T	1XT	0XT	XT1	XTB	XTB
11T	00T	00	00T	00T	00T	00	00T	00	00	00T	S0	00T	00T	00T	00T	00T	00T	00	00T	00T
X0T	X1T	00	X0T	01	X1T	X0	XTB	0X	XT1	X1T	S0	X1T	00T	X1T	X0T	XTB	0XT	XT1	XTB	XTB
X1T	X0T	00	X1T	00T	X0T	X0	X0T	00	X0	X1T	S0	X0T	00T	X0T	X0T	X0T	00T	X0	X0T	X0T
0XT	1XT	00	10	0XT	XTB	X0	1XT	0X	XT1	1XT	S0	1XT	00T	XTB	X0T	1XT	0XT	XT1	XTB	XTB
1XT	0XT	00	00T	0XT	0XT	00	0XT	0X	0X	0XT	S0	0XT	00T	0XT	00T	0XT	0XT	0X	0XT	0XT
XT0	XT1	00	X0	0X	XT1	X0	XT1	0X	XT1	XT1	S0	XT1	00	XT1	X0	XT1	0X	XT1	XT1	XT1
XT1	XT0	00	X0T	0XT	XT0	X0	XT0	0X	XX	XT0	S0	XTB	00T	XTB	X0T	XTB	0XT	XT1	XT0	XTB
XTB	XTB	00	X0T	0XT	XTB	X0	XTB	0X	XT1	XTB	S0	XTB	00T	XTB	X0T	XTB	0XT	XT1	XTB	XTB

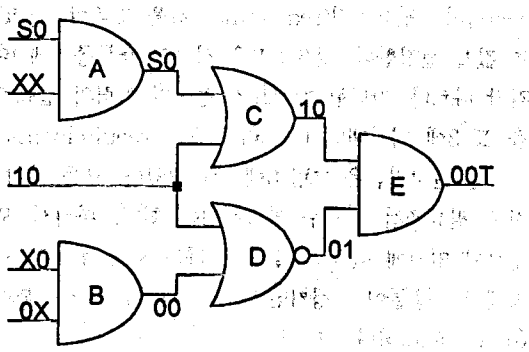

그림 3. 0값을 갖는 논리값의 차이
Fig. 3. Different Logic Values For Logic 0.

논리값 XT0은 두 개의 시간 프래임동안 명확하지 않은 논리값인 X를 갖지만 절대 S0은 될 수 없다는 것을 표시한다. 마찬가지로 XT1은 두 시간 프래임의 값이 각각 X이지만 절대 S1은 될 수 없음을 나타낸다. 논리값 XTB 역시 두 개의 시간 프래임동안 명확하지 않은 논리값 X를 갖지만 절대 S0나 S1이 될 수 없음을 나타낸다. 즉 XTB는 XT0와 XT1의 합집합이다.

20개의 논리값이 복잡해 보일 수 있으나 이들로도 함수적 지정을 사용한 경로 지연 검사의 문제를 위한 논리값으론 완벽한 것이 못 된다. 이러한 문제를 해결하기 위한 완벽한 논리값 세트(set)가 51개로 구성된다는 것은 [5]에서 보여졌다. 본 논문에서는 간결성과 완벽한 논리값 세트에 의해 제공되는 역추적(backtracking) 동안의 충돌(conflict)을 피할 수 있는 방법을 적절히 고려하고 유추표(implication table)의 엄청난 복잡도를 피하기 위해 51개의 논리값을 사용하지 않고 20개의 논리값을 사용하였다.

지금까지 소개한 20개 논리값의 논리 연산은 각 연산에 대해 연산표를 만들어 사용한다. 한가지 예로 NOR 게이트를 위한 연산표를 표 2에 나타내었다. 다른 논리 연산도 이와 같이 표를 만들어 사용한다.

Ⅳ. 알고리즘

경로 지연고장 검사는 쌍패턴 검사이다. 제안된 검사입력 생성기는 각 회로를 두 개의 시간 프래임에 걸쳐 조합회로로 가정한다. 스캔 플립플롭의 처음 시간 프래임에서의 값은 제어가능(controllable)하지만 두 번째 프래임의 값은 회로의 함수에 의해 결정된다. 기존의 순차회로를 위한 검사입력 생성 방법은 마지막 시간 프래임에서 시작하여 시간을 역으로 거슬러 올라갔다[6]. 본 연구에서도 이 역시간 처리(reverse-time processing) 방법을 사용한다. 이 방법을 사용할 때, 각각 하나의 시간 프래임에서 검사입력 생성 과정을 행하면 회로 소자가 두 개의 시간 프래임에서 같은 값을 갖더라도 해저드가 없다는 것을 보장할 수 없기 때문에 생성된 검사입력이 경성 검사입력임을 보장할 수 없다[6]. 따라서 본 연구에서는 경성 검사입력 보장을 위해 앞서 소개한 많은 논리값을 사용한다.

몇몇 검사입력 생성[7]과 검증 기술은 순차회로의 가능한 상태 전이(state transition)를 열거하기 위해 BDDs(binary decision diagrams)를 사용하여 접근하였다. 이러한 접근은 회로의 모든 순차적 특성에 관한 정보를 제공하고 검사입력의 유무를 판별하여 검사입력이 있을 경우 검사입력을 찾을 수 있다. 하지만 이 방법을 이용하여 높은 효율을 얻기 위해서는 많은 메모리를 필요로 하는 단점을 갖고 있다. 사실 큰 회로에서 BDDs를 구성하는 것은 불가능하다. 본 연구에서는 큰 회로를 효과적으로 다루기 위해 메모리 사용을 최소한으로 하는 역시간 처리를 기반으로 한다. 상태 전이는 단지 순차회로 소자의 출력과 입력사이에서 논리값의 전달로 이루어진다. 상태 유지 소자(state-holder)는 상태 공간 검색(state space search)에서 각 소자가 팬인(fanin) 소자의 함수로서 표현되는 다른 소자들과 마찬가지로 다뤄진다. 지정 과정에서 두 번째 시간 프래임에서 플립플롭을 만나면 플립플롭의 두 번째 시간 프래임의 값은 첫 번째 시간 프래임의 플립플롭 입력값에 복사된다. 첫 번째 시간 프래임에서 플립플롭을 만나면 지정 과정은 정지된다. 상태 전이의 예를 그림 4에 C, D로 나타내었다.

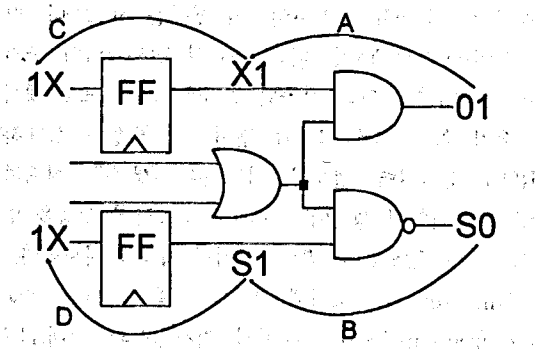

그림 4. 상태 전이와 유추의 예
Fig. 4. Example of State Transition and Implication.

```
test_path()
{
  for (i=0;i<Num_paths;i++) {
    path = read_path(i);
    /* find a hazard-free robust test */
    flag = find_test(path, HFR);
    if (flag != TEST_GENERATED) {
      /* find a robust test */
      flag = find_test(path, ROB);
      if (flag != TEST_GENERATED) {
        /* find a strong non-robust test */
        flag = find_one_test(path, SNR);
        if (flag != TEST_GENERATED) {
          /* find a weak non-robust test */
          flag = find_one_test(path, WNR);
        }
      }
    }
  }
}
find_test(path, test_type)
{
  /* put requirements according to test types */
  flag = set_requirements(path, test_type);
  if (flag==UNTESTABLE || flag==ABORTED)
    return(flag);
  /* do implication */
  flag = implication();
  if (flag==UNTESTABLE || flag==ABORTED)
    return(flag);
  /* justify the values */
  flag = justification();
  if (flag==UNTESTABLE || flag==ABORTED)
    return(flag);
  else
    /* test is generated */
    mark_success(path);
}
```

그림 5. 경로 지연고장 검사입력 생성 알고리즘
Fig. 5. Path Delay Test Generation Algorithm.

경로 지연고장 검사입력 생성의 주 알고리즘을 그림 5에 나타내었다. 검사 과정은 가능한 한 그림 1에서 나타낸 조건 공간을 최대한 점유하는 과정을 취한다. 고장 경로를 검출하기 위한 접근 방법에서 우선 특정한 경로에 대한 HFR 검사입력의 생성을 시도한다. 만약 검사입력이 존재하면 다른 종류의 검사입력의 조건은 이 검사입력에 의해 충족된 조건에 포함되므로 생성할 필요가 없다. 만약 HFR 검사입력 생성에 실패하면 ROB 검사입력 생성이 시도된다. 생성을 실패하면 SNR 검사입력 생성이 수행되고 성공하면 생성을 멈추고 그렇지 않으면 WNR 검사입력 생성을 시도한다.

"find_test"를 고려해 보자. 처음 단계인 "set_requirements"에서는 주어진 검사입력의 형태(HFR, ROB, SNR, WNR)에 따라, 검사되고 있는 경로상의 모든 소자의 경로외 입력(off-path input)의 값이 검사입력의 필요조건(requirements)을 만족할 수 있도록 값을 설정한다. 필요조건은 검사입력의 종류에 따라 각각 다르다. 간단한 게이트의 경로외 입력의 필요조건을 표 3에 나타내었다. 그리고 "set_requirements"의 상세한 알고리즘은 그림 6에 나타내었다. 모든 값의 설정은 필요조건을 모두 만족시켜야 한다. 따라서 모든 경로외 입력에 필요조건을 만족하는 값을 설정하는 과정에서 충돌이 생기면 그 경로는 검사불가능(untestable)하다.

표 3. 경로외 입력의 필요조건
Table 3. Requirements on Side Inputs.

소자 종류	경로상의 전이	경로외 입력 제약조건			
		HFR	ROB	SNR	WNR
AND/NAND	Rising	S1	X1	X1	X1
	Falling	S1	S1	11	X1
OR/NOR	Rising	S0	S0	00	X0
	Falling	S0	X0	X0	X0

```
set_requirements(path_list, test_type)
{
  /* start from head of the path */
  path = path_list->path_head;
  /* get an element on the path */
  while (element != NULL) {
    flag = is_conflict(
            element->real_val, element->just_val);
    if (flag==CONFLICT)
      return UNTESTABLE;
    for (i=0;i<element->fanin_num;i++) {
      fanin = element->fanin_list[i];
      /* do not consider on_path fanin */
      if (fanin->on_path==NO) {
        /* put the requirement value
           according to the test type */
        fanin->just_val =
          get_requirements(element,fanin,test_mode);
        /* if putting required value is impossible */
        flag =
            is_conflict(fanin->val, fanin->just_val);
        if (flag==CONFLICT)
          return UNTESTABLE;
      }
    }
    /* get next element on path */
    path = path->next;
    element = path->element;
  }
  return SUCCESS;
}
```

그림 6. 필요조건 설정 알고리즘
Fig. 6. Requirements Setting Algorithm.

다음 과정으로 출력값으로 입력값을 찾는 유추

(implication) 과정을 실행한다. 이 과정에서는 모든 논리값을 사용할 필요가 없다. 소자의 출력이 안정한 값을 갖으면 플립플롭을 제외한 다른 소자의 입력값으로 안정한 값을 유추할 수 있다. 따라서 출력이 안정한 값을 갖는 경우에는 3개의 논리값(S0, S1, X)만 고려된다. 출력에 이외의 값들을 갖는 경우에는 두 개의 시간 프래임을 동시에 표현하는 논리값 1개를 하나의 시간 프래임만을 표현하는 2개의 값으로 분리하여 각각의 시간 프래임에 대해 한 번씩, 즉 두 번의 역추적 과정을 거쳐 유추된 값을 두 개의 시간 프래임을 동시에 표현하는 논리값으로 합치는 방법을 사용하게 되어 3개의 논리값(0, 1, X)만으로 다룰 수 있다. 필요조건값으로 XT1과 같은 안정이 불가능함을 나타내는 논리값을 사용할 수 없으므로 이와 같은 방법을 사용할 수 있다. 그림 4에서 A, B로 역추적 과정의 예를 나타내었다. 이 역방향 유추 과정은 더 이상의 유추가 불가능할 때까지 계속된다. 이 과정에서 만약 이미 설정된 값과 유추된 값과의 충돌이 발생하면 이 경로는 검사가 불가능한 경로이다. 왜냐하면 이 유추된 값은 해당 검사입력 종류가 필요로 하는 모든 조건을 만족하는 값이기 때문이다. 또한 이 과정동안 주입력이나 플립플롭의 출력이 X 이외의 다른 값으로 설정되면 유추된 값을 사용하여 출력값을 정하는 시뮬레이션을 실행한다.

유추 과정 후에 회로의 입력들과 상태 변수들을 통한 상태 공간 검색으로 남아 있는 소자들의 지정 과정이 행해진다. 이 상태 공간 검색은 PODEM과 유사한 조합회로 검사입력 생성 방법을 사용한다. 이는 설정할 수 있는 값을 선택하는 것이므로 충돌이 발생하면 모든 가능한 선택을 고려하여야 한다. 더 이상의 선택이 없으면 해당 경로는 검사불가능하다. 검색 공간을 줄이기 위해 안정된 값을 먼저 고려한다. 그 후 두 번째 시간 프래임을 고려하고 마지막으로 첫 번째 시간 프래임을 고려한다. 안정된 값을 위한 지정 알고리즘은 그림 7에 나타내었다.

안정된 값을 필요로 하는 소자를 기록하고 있는 목록에 소자가 없을 때까지 지정 과정을 계속한다. 이 과정을 마치면 해당 경로는 검사불가능하다고 판명되던지 검색이 중단(abort)된다. 소자의 출력값이 필요로 하는 값과 일치하면 이 소자는 목록에서 제거되고 다음 소자를 고려한다. 만약 값이 다르면 필요로 하는 값을 생성하기 위해 주입력이나 플립플롭까지 역추적(backtrack)한다. 주입력이나 플립플롭에 적당한 값을 할당한 후 순방향으로 시뮬레이션을 수행한다. 이 과정에서 충돌이 없으면 다음 소자를 고려한다. 만약 충돌이 있으면 "backup"에서 충돌을 해결하기 위해 또 다른 값을 할당한다. 이 과정은 PODEM에서의 역추적 과정과 유사하다. 더 이상의 선택적인 할당 값이 없으면 해당 경로는 검사불가능하다고 결정된다.

```
stable_justification(){
  /* get an element from the stable value list */
  element = get_element_from_list(Stable_list);
  while (element!=NULL) {
/* when the required value is the same as the
 current value, the element is removed from the
list */
    if (element->real_val==element->just_val)
      remove_from_list(Stable_list, element);
    else {
      for (i=0;i<element->fanin_num;i++) {
        fanin = element->fanin_list[i];
        /* put value that make just_val */
        fanin->just_val =
              get_just_val(element, fanin, Stable);
        if (fanin==PI || fanin==PPI) {
          flag = simulation(fanin);
          if (flag==CONFLICT)
            flag = backup();
            if (flag==CONFLICT)
              return CONFLICT;
            else if (flag==ABORTED)
              return ABORTED;
        }
        else
        /* store the fanin for further justification */
          put_element_to_list(Stable_list, fanin);
      }
    }
    element = get_element_from_list(Stable_list);
  }
  return SUCCESS;
}
```

그림 7. 안정된 값의 지정 알고리즘
Fig. 7. Stable Value Justification Algorithm.

20개의 논리값을 사용하여 충돌을 미리 예상할 수 있어 계산 시간을 줄일 수 있다. 그림 8에 한가지 예를 제시하였다. D의 출력값을 S1로 만드는 것이 목적이라고 가정하자. 만약 확장된 20개의 논리값을 사용하지 않았다면 A의 출력은 XX로 B의 출력은 X1로 표시될 것이다.(이러한 경우를 괄호 안에 나타내었다.) 이 경우 D의 출력을 S1로 만들기 위해 B와 C의 출력 중 하나를 S1로 만들어야 하는데 이 때 어떤 것을 선택할 지를 결정할 수 없다.

만약 B의 출력을 S1로 하기로 선택하여 계속 역추적을 하면 A의 출력을 S1로 할 것이고 이 값은 이미 할당된 A의 입력에 의해 A의 출력이 S1이 될 수 없

음을 알고 처음으로 돌아가 B대신 C를 선택할 것이다. 하지만 본 논문에서 사용한 논리값을 사용하면 A는 XTB가 되고 B는 XT1이 되어 D를 S1로 만들기 위한 유일한 선택은 C가 되어 C의 입력 0X를 S0으로 만들므로서 불필요한 충돌을 피하여 시간을 절약한다.

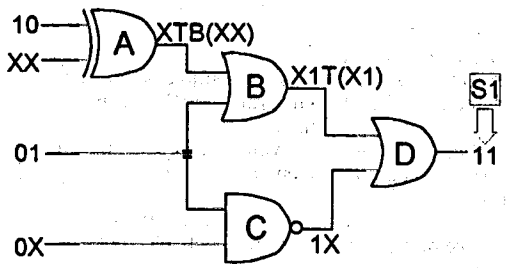

그림 8. 신속한 충돌 검출의 예
Fig. 8. Example of Early Conflict Detection.

두 번째 시간 프레임 지정 과정에서는 두 번째 시간 프레임을 나타내는 값만 필요하므로 검색 공간을 줄이기 위해 두 개의 시간 프레임을 동시에 나타내는 논리값에서 {0, 1, X}만을 추출하여 지정 과정을 수행한다. 이 후, 할당된 새로운 값은 첫 번째 시간 프레임의 값과 합쳐진다. 이 알고리즘은 새로 할당된 값이 두 번째 시간 프레임의 값이고 충돌을 해결하기 위한 더 이상의 남아있는 할당 값이 없을 경우 안정된 값의 지정 과정으로 되돌아가는 것을 제외하면 안정된 값의 지정 알고리즘과 유사하다.

첫 번째 시간 프레임 지정 알고리즘도 {0, 1, X}를 추출하여 사용한다. 이 알고리즘은 새로 할당된 값이 첫 번째 시간 프레임의 값이고 충돌을 해결하기 위한 할당 값의 선택이 남아 있지 않은 경우 두 번째 시간 프레임 지정 과정을 되돌아가는 것을 제외하면 두 번째 시간 프레임 지정 알고리즘과 유사하다. 목록에 남아 있는 소자가 없을 때 검사입력의 모든 조건이 충족되고 해당경로의 검사입력이 생성된다.

알고리즘을 예시하기 위하여 그림 9에 간단한 예를 들었다. 이 예에서는 진한 선으로 표시된 경로, 즉 주 입력 B의 하향 전이(falling transition)가 {D, I, K}를 거쳐 최종 출력에 이르는 경로에 대한 경성 검사입력을 생성하는 알고리즘을 보여준다. 우선 해당 경로에 상향 전이(rising transition)일 경우 01을 하향일 경우 10값을 할당(B=10, D=01, I=01, K=10)한다. 이제 알고리즘의 첫 단계로 경성 검사입력의 필요조건을 만족시키는 경로외 입력값을 설정한다. 표 2를 참조하면 이 값을 알 수 있다. 그림 9에서는 *가 있는 값들이 경성 검사입력을 위한 경로외 입력값들(C=S1, F2=X1, J=S0)이다.

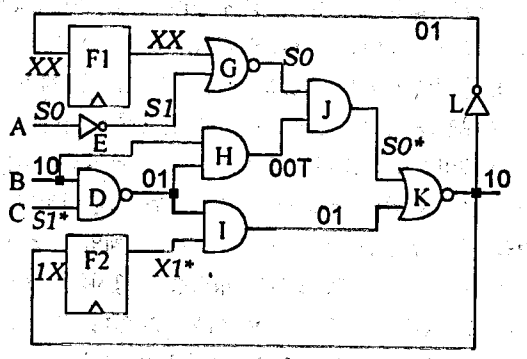

그림 9. 경성 검사입력 생성 과정 예
Fig. 9. Example of Generating A Robust Test.

다음 과정으로 유추 과정을 수행한 후 주 입력이나 플립플롭에 설정된 값을 가지고 시뮬레이션을 행하게 되는데 이 과정에서 {H=00T, L=01}이 정해진다. 이제 지정 과정을 수행한다. J를 S0으로 만들기 위한 입력값으로 G를 S0으로 한다. 왜냐하면 이미 H는 S0이 될 수 없음을 나타내므로 선택할 필요가 없다. G를 S0으로 만들기 위해 F1=S1 또는 E=S1 중 하나를 선택하여야 하는데 F1의 출력은 이미 설정된 값에 의해 S1이 될 수 없으므로 E=S1, A=S0이 되고 F1은 XX가 된다. 이 과정에서 충돌이 발생하지 않았으므로 이 경로에 대한 경성 검사입력은 다음과 같다. 첫 번째 시간 프레임의 값은 {A B C F1 F2 = 0 1 1 X X}이고 두 번째 시간 프레임의 값은 {A B C = 0 0 1}이 된다.

V. 실험 결과

본 알고리즘은 C 언어로 구현하였으며 회로는 표준 ISCAS'89 벤치마크 회로(benchmark circuits)를 사용하였고 각 회로에서 가장 긴 경로 1000개를 상용 타이밍 분석기로 추출하여 사용하였다. 프로그램은 32 메가 바이트의 메모리 용량을 갖는 SUN Sparc 2에서 실행하여 실험 결과를 얻었다. 검사입력 생성 결과는 표 4에 나타내었다. 표 4에서는 4가지 종류의 각 검사입력에 대하여 검사입력이 생성된 경로, 검사불가능한 경로 그리고 생성이 포기된 경로의 수를 표시하였고 검사입력 생성을 위한 CPU 동작시간과 사용한 메모리의 양을 표시하였다. 많은 종류의 검사입력을 얻기 위

해 HFR, ROB, SNR, WNR 4가지 종류의 검사입력을 구하였으며 만약 어떤 경로에 대하여 보다 제약조건이 많은 검사입력이 생성되면 이 경로는 더 이상 고려하지 않았다.

다시말해, 보다 제약조건이 많은 검사입력의 생성이 불가능한 경로나 포기된 경로에 대해서, 시행된 검사입력보다 적은 제약조건을 갖는 검사입력의 생성을 시도했다. 따라서 표 3에서 보듯이 각 회로의 ROB에 표시된 모든 경로수의 합은 보다 제약조건이 많은 HFR의 검사불가능한 경로수와 포기된 경로수이 합과 같게 된다. SNR과 WNR도 마찬가지이다. 실험 과정에서 각 검사입력 종류에 대하여 한 경로당 역추적의 수를 10000번으로 제한하였다. 생성된 모든 검사입력은 경로 지연고장 검사용 시뮬레이터[4]를 이용하여 정확함을 검증하였다.

본 결과의 CPU 동작시간과 메모리 사용을 Dsteed[6]와 비교한 것을 표 5에 나타내었다. Dsteed는 처음으로 알려진 표준 스캔 설계를 위한 경로 지연 고장 검사입력 생성기이다. Dsteed는 동작할 때 한가지 종류의 검사입력만을 생성하고, 또한 경성 검사입력을 생성하지 못하기 때문에 본 연구에서 개발한 프로그램은 단지 SNR만 생성하도록 동작시켰다. 표 5에서 볼 수 있듯이 큰 회로에서 Dsteed는 많은 메모리를 사용하게 되고 또한 회로에 포함된 플립플롭의 수에 비례하여 급격한 메모리 사용의 증가를 보여준다. 하지만 본 연구에서 제시한 알고리즘은 메모리 사용이 플립플롭의 개수와 무관하며 회로의 크기와의 상관관계가 적고 검사입력의 생성이 빠른 시간에 이루어짐을 알 수 있다

표 5. Dsteed와의 비교
Table 5. Comparison with Dsteed.

회로	플립플롭 개수	본 논문		Dsteed	
		동작 시간 [초]	메모리 사용 [Mbyte]	동작 시간 [초]	메모리 사용 [Mbyte]
s713	19	8.20	1.54	133.6	1.45
s1196	18	277.22	0.93	70.4	0.96
s1238	18	291.71	0.89	60.2	0.97
s1423	74	901.73	1.40	24150.5	8.44
s1488	6	305.40	0.84	240.9	0.70
s1494	6	312.01	0.81	240.2	0.68
s9234	228	40.44	3.45	8.9	12.30
s13207	669	48.82	5.07	★	★
s15850	597	56.06	5.87	★	★
s38417	1636	318.16	13.19	★	★

★ 약 25Mbyte 이상의 메모리를 사용한 후 메모리 부족으로 동작 중단

VI. 결 론

본 논문에서는 20개의 논리값에 기반을 둔 함수적 지정 방법을 사용한 새로운 알고리즘을 개발하여 완전스캔 설계 환경에서 동작하는 경로 지연 고장 검사입력 생성기를 제안하였다. 이것은 스캔 설계를 이용한

표 4. 검사입력 생성 결과
Table 4. Test Generation Results.

회로	HFR			ROB			SNR			WNR			CPU 동작 시간 [초]	메모리 사용 [Mbyte]
	D	U	A	D	U	A	D	U	A	D	U	A		
s713	0	1000	0	12	988	0	2	986	0	108	878	0	30.09	1.52
s1196	135	863	2	83	780	2	17	763	2	39	726	0	604.37	0.91
s1238	54	946	0	97	847	2	12	834	3	28	809	0	617.43	0.92
s1423	5	989	6	0	993	2	0	984	11	0	955	40	3770.85	1.39
s1488	169	831	0	179	652	0	96	556	0	375	181	0	1181.18	0.83
s1494	167	833	0	178	655	0	94	561	0	371	190	0	1172.82	0.84
s9234	0	1000	0	0	1000	0	0	1000	0	0	1000	0	138.57	3.45
s13207	0	1000	0	0	1000	0	0	1000	0	0	1000	0	157.41	5.03
s15850	0	1000	0	0	1000	0	0	1000	0	0	1000	0	187.45	5.92
s38417	124	876	0	61	732	83	10	804	1	345	459	1	4137.57	14.27

D: 검사입력이 생성된 경로수, U: 검사불가능한 경로수, A: 포기된 경로수

여러 회로나 칩의 성능 검사와 향상을 위해 중요한 역할을 할 수 있으며 기존의 연구와 비교할 때 메모리와 CPU 사용에서 현격한 향상이 있었음을 실험 결과를 통해 알 수 있다. 본 알고리즘을 기반으로 하여 논리 게이트만으로 이루어진 회로뿐만 아니라 CMOS 수준으로 표현된 부분을 포함한 표준 스캔 회로에도 동작하는 검사입력 생성기에 관한 연구를 수행하고 있다.

참 고 문 헌

[1] J. Waicukauski, E. Lindbloom, B. Rosen and V. Iyengar, "Transition Fault Simulation", IEEE Design and Test, pp. 32-38, April 1987.

[2] S. Reddy, C. Lin and S. Patil, "An Automatic Test Pattern Generation for the Detection of Path Delay Faults", Proc. ICCAD, pp.284-287, 1987.

[3] C. Lin and S. Reddy, "On Delay Fault Testing in Logic Circuits", IEEE Trans. On CAD, Vol. 6, No. 5, pp. 694-703, Sept. 1987.

[4] S. Kang, B. Underwood and W. Law, "Path Delay Fault Simulation for a Standard Scan Design Methodology", Proc. ICCD, pp. 359-362, 1994.

[5] K. Fuchs, H. Wittmann and K. Antreich, "Fast Test Pattern Generation for All Path Delay Faults Considering Various Test Classes", Proc. ETC, pp. 89-98, 1993.

[6] K-T. Cheng, S. Devadas and K. Keutzer, "Delay-Fault Test Generation and Synthesis for Testability Under A Standard Scan Design Methodology", IEEE Trans. CAD, pp. 1217-1231, Aug. 1993.

[7] H. Cho, S. Jeong, F. Somenzi and C. Pixley, "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation", Journal of Electronic Testing:Theory and Applications, pp. 19-31, 1993.

저 자 소 개

姜 容 碩(正會員)

1995년 2월: 연세대학교 전기공학과 졸업(공학사). 현재: 연세대학교 대학원 전기공학과 석사과정. 주관심 분야 : 테스팅, DFT와 VLSI & CAD 등임.

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(공학 석사). 1992년 5월: The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학 박사). 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. 주관심분야 : VLSI CAD, 테스팅, 테스트를 고려한 설계, VLSI설계.