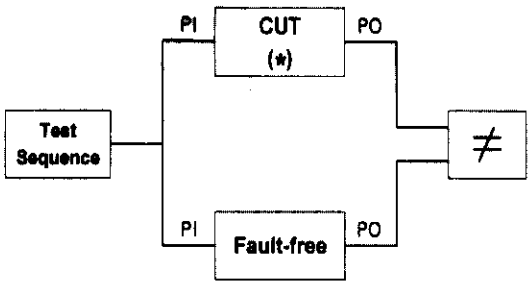
특 집

순차회로의 테스트생성 기술

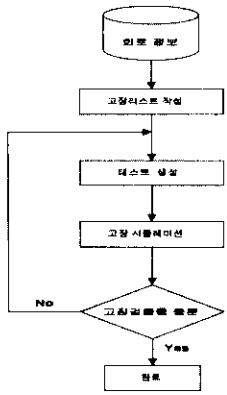
최 호 용
충북대학교 전기전자공학부

I. 서 론

반도체기술의 발달에 따라 집적회로의 규모는 급격히 커지고, 복잡도 또한 급격히 증가하게 되었다. 따라서 이들 회로의 신뢰성 확보를 위한 테스트기술의 중요성은 점점 더 증가하고 있다. 일반적으로 집적회로의 테스트는 〈그림 1〉과 같이 대상회로(circuit under test:CUT)의 외부입력단자(PI)에 테스트계열을 인가하고 외부출력단자(PO)에서의 출력을 관측하여, 무고장(fault-free)회로의 예상출력과 비교함으로써 수행된다. 테스트에 필요한 테스트계열을 생성하는 것을 테스트생성이라고 한다. 테스트생성은 1) 회로정보(회로의 종류 및 결선에 관한 기술), 2) 회로에 가정되는 고장들의 집합을 입력으로 하여 고장들을 외부출력에서 검출할 수 있는 입력벡터 계열을 구하는 것이다. 대상회로의 종류에는 크게 조합회로(combinational circuits)와 순차회로(sequential circuits)로 구별되고, 회로에 가정되는 고장에 관해서는 여러 모델이 가정되나, 실제로 일어나는 고장의 대부분은 고착고장(stuck-at fault)에 귀착되기 때문에 고착



〈그림 1〉 테스트생성



〈그림 2〉 테스트생성 흐름도

고장만을 고려하는 경우가 대부분이다.

일반적인 테스트생성의 흐름은 〈그림 2〉와 같다. 회로정보가 주어지면 등가고장을 탐색해 대표고장을 정해 고장리스트를 작성한다. 고장리스트 중에 있는 고장에 대해 이를 검출하는 테스트를 생성한다. 생성된 테스트의 평가를 위해 고장시뮬레이션(fault simulation)을 수행한다. 고장시뮬레이션은 고장회로와 무고장회로에 대해 테스트를 인가했을 때의 회로동작의 시뮬레이션을 수행하고 주어진 테스트에 의해 어느 고장이 검출되는가를 조사한다. 이때 부수적인 고장에 대해서도 검출이 가능하여 검출된 고장을 고장리스트에서 고장제거(fault drop)를 한다. 최초로 가정된 고장중 검출된 고장 비율인 고장검출율(fault coverage)이 불충분한 경우, 미검출 고장에 대해 테스트생성과 고장시뮬레이션을 반복 수행한다.

조합회로의 테스트생성에 있어서는 FAN, TOPS, SOCRATES 등[1-5]의 효율적인 테스트생성 알고리듬이 제안되어, 대부분의 고장에 대해 현실적인

시간으로 테스트생성이 가능하여 실용적인 테스트 생성시스템이 구축되어 있다. 그러나, 순차회로의 테스트생성에 있어서는 최근 활발히 연구가 수행되고 있지만 실용적인 관점에서는 아직도 어려운 문제로 남아 있다. 현재 ISCAS'89 순차회로 벤치마크회로[6]에 대해서도 현실적인 시간안에 충분한 고장검출율을 얻지 못하는 실정이다.

순차회로의 테스트생성이 어려운 이유는

1) 거대한 탐색공간:동일 규모의 조합회로에 비해 거대한 탐색공간을 갖는다. 조합회로에 대해서는 탐색공간이 2^p(p는 PI수)인데 비해 순차회로에 대해서는 시간대당 2^p+9^n(n은 플립플롭수)의 탐색공간을 갖는다[12].

2) 회로 내부상태의 설정 및 관측이 어렵다.

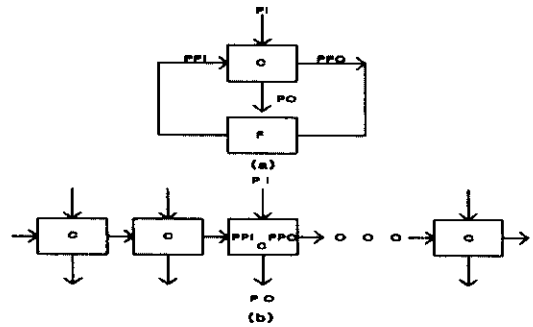
3) 다중고장 취급:주어진 순차회로에서의 고장이 단일고장이라 하더라도 시간전개한 반복어레이(iterative array) 조합회로에는 이를 다중고장으로 취급하여야 한다.

이 문제를 해결하기 위하여 스캔패스(scan path) 설계방식 등[7,8]의 테스트용이화설계(design for testability)를 이용하여 순차회로를 조합회로의 테스트생성 문제로 귀착시키는 방법이 있다. 그러나 테스트용이화를 위한 하드웨어의 증가, 회로성능의 저하, 테스트계열의 증대 등의 문제가 있어 여전히 순차회로에 대해 직접 검사입력을 생성하는 테스트 생성기술 대한 요망이 강한 실정이다.

본고에서는 주요 순차회로에 대한 테스트생성 방법을 소개한다. 순차회로의 테스트생성법을 크게 topological 테스트생성법, simulation-based 테스트생성법, symbolic 테스트생성법의 세 부분으로 나누어 각 부분의 주요 방법을 소개하고, 또한 장단점도 기술한다.

II. 순차회로의 테스트생성

동기식 순차회로는 〈그림 3(a)〉와 같이 논리 게이트로 구성되는 조합논리회로부 C와 플립플롭으로 구성되는 동기식 플립플롭부 F로 모델링된다. 순차회로의 테스트계열을 생성하기 위해 되먹임루프(feedback loop)를 자르고 시간적으로 전개하면 〈그림 3(b)〉와 같은 반복 어레이(iterative array) 조합논리회로가 된다. 한 시간대의 조합논리회로 C(이하 '셀'이라 함)는 원래 순차회로의 조합논리회로 C와 동일하고, 위 입력과 아래 출력은 주입력(primary inputs:PI)과 주출력(primary outputs:PO)을 나타내고 각 셀의 왼쪽 입력과 오른쪽 출력 의사 주입력(pseudo PI:PPI), 의사 주출력(pseudo PO:PPO)을 나타낸다. 한 시간대(time frame)의 출력 PPO는 다음시간대의 입력 PPI로 연결된다.



〈그림 3〉 동기식 순차회로의 반복어레이 회로
(a) 순차회로 모델, (b) 반복어레이 모델

일반적으로 순차회로의 테스트생성은 다음 세 단계로 나누어진다.

1) 고장활성화(fault activation)벡터 생성단계: 한 시간대에서 PI나 PPI에 값을 할당해 고장위치에 고장이 있는 경우와 고장이 없는 경우에 대한 신호값의 차이를 발생시켜 고장이 활성화(activation)되도록 하고, 이를 그 시간대의 PO나 PPO에 전달시키는 고장활성화벡터를 구하는 단계(PPI의 값을 활성화상태라 함)

2) 고장영향전파(fault effect propagation)계열 생성단계:1)에서 PPO에 전달된 고장영향이 회로외부에서 관측될 수 있게 하는, 즉 PO에까지 전파시키는 고장영향계열을 구하는 단계

3) 활성화상태 정당화(activation state justification)계열 생성단계:초기상태(미지상태나 특정상태)에서 1)의 활성화상태 PPI를 정당화시키기

위한 정당화계열(justification sequence)을 구하는 단계

1)-3)단계에서 얻어진 상태정당화계열, 고장활성화벡터, 활성화상태 정당화계열의 연결이 주어진 고장을 검출하기 위한 테스트계열이 된다.

각 단계는 시간에 따른 입력계열의 결정형태에 따라 순방시간처리(forward time processing: FTP)방법과 역방시간처리(reverse time processing:RTP)로 나누어진다. FTP에서는 입력 순서열이 시간의 순방향으로 결정되고, RTP에서는 시간의 역방향으로 결정된다. 테스트생성법은 1) FTP, 2) RTP, 3) FTP와 RTP의 조합을 이용한 방법으로 나누어진다. 대표적인 예로 1)은 simulation-base에 의한 테스트생성법[21-23], 2)는 BACK 알고리즘[12], 3)은 HITEC[14] 등이 있다.

지금까지 제안된 순차회로의 테스트생성 알고리즘은 크게 topological 방법, symbolic 방법, simulation-based 방법으로 나누어진다. topological 방법은 조합회로에서 이용한 경로활성화법을 순차회로에 확장 적용한 방법으로 조합회로에서 사용한 휴리스틱(heuristics)을 그대로 적용할 수 있으나, 회로 규모가 크면 탐색공간이 넓어져 많은 테스트생성 시간이 필요하다. symbolic 방법은 Boolean 함수 등으로 표현한 회로의 구조정보를 이용하는 방법으로, 상태수가 제한된 경우 가장 좋은 해를 얻을 수 있으나, 큰 회로에 대해서는 테스트가 불가하다. simulation-based 방법은 고장시뮬레이터 등에 의해 얻어진 결과를 사용하는 방법으로, 대규모 회로에도 적용가능하나, 고장검출율이 나쁘고 테스트불가 고장을 판별할 수 없다.

III. Topological 테스트생성법

반복어레이 모델을 사용하여 조합회로에서 제안된 분지한정(branch and bound)법을 순차회로에 확장시킨 방법이다[9-15]. 이 방법의 유효성은 탐색하는 데 얼마나 효율적인 휴리스틱을 사용하느냐에 크게 의존한다. 이 방법은 대규모 회로에 대해서는 탐색공간이 너무 커 효율적인 테스트생성이 어렵다. 본 장에서는 대표적인 topological 테스트생성법인 HITEC[14]과 BACK 알고리듬[12]을 소개한다.

1. HITEC

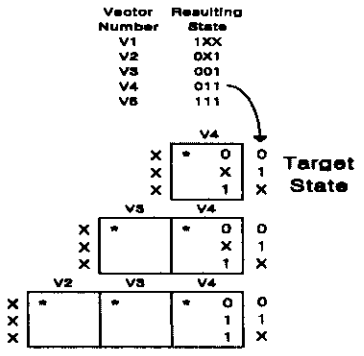
HITEC은 조합회로에서 사용한 효율적인 탐색법을 접목하고, 상태정보를 이용하여 상태정당화를 효율적으로 수행하고, 고장시뮬레이션을 통해 고장전파에 관해 얻어진 정보 등을 이용한 테스트생성법으로 가장 대표적인 topological 테스트 방법이다. HITEC에서는 고장활성화와 고장영향 전파과정은 FTP로, 상태정당화는 RTP로 하여 FTP와 RTP의 조합으로 수행한다.

HITEC에서는 FAN[3], TOPS[4], SOCRATES[5] 등에서 제안된 지배노드(dominator)개념을 활용한다. 지배노드는 고장영향전파경로 상에서 고장영향이 반드시 통과되어야 할 논리소자이다. 그 지배노드 게이트에 고장을 출력으로 전파하기 위해서는 게이트입력에 무영향논리값(noncontrolling value; AND, NAND 게이트에 대해서는 1, OR, NOR 게이트에 대해서는 0)을 할당시켜야 한다. 이와 같은 강제값할당(mandatary assignment)은 탐색공간을 줄일 수 있다. 그러나 D-경계(frontier) 노드(입력에 D 출력은 X인 게이트)가 여러개 있는 경우 각 D-경계노드들의 지배노드를 동시에 고려하면 지배노드수가 줄어들게 되어 탐색 공간이 커진다. 따라서, HITEC에서는 고장영향전파과정을 수행할 때, 항상 D-경계를 하나의 목표(target) D-경계만으로 제한시켜 강제값 할당 수를 증가시킴으로써 탐색공간을 줄인다. 나머지 D-경계는 스택에 저장되어 시도철회(backtrack)가 있을 때에 사용한다. 한 D-경계가 목표 D-경계로 선정되면 지배노드와 전후방영향분석(forward and backward implication)을 통해 모든 강제값할당을 수행한다. 할당된 값들에 의한 목표(objective)들은 PODEM의 후방추적(backtrace)에 의해 정당화된다.

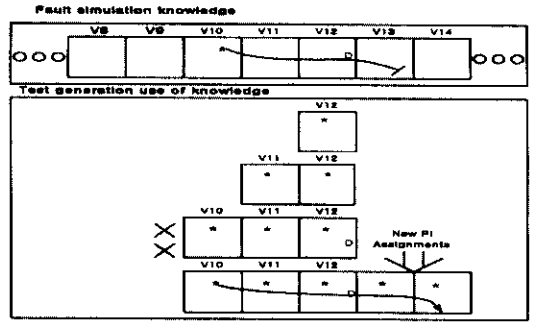
HITEC에서는 효율적인 상태정당화를 위해 [28, 29] 등에서 사용한 방법과 비슷하게 이미 생성된

테스트벡터에 의한 무고장회로의 상태정보를 이용한다. 즉, 정당화할 현재상태(활성화상태)가 상태정보 안에 있으면, 무고장회로에서 이 상태에 이르게 한 벡터는 고장회로에서도 그 상태에 도달하게 할 가능성이 많다는 사실을 이용한다. 상태정보를 이용하여 목표상태에 도달시킨 마지막 벡터로부터 한번에 하나씩 벡터를 역순으로 인가하면서 고장회로에서도 유효한지를 확인한다. 만약 활성화상태에 도달하면 테스트가 얻어되고, 만약 상충이 발생되거나 정당화해야 할 상태가 상태정보에 없다면 PODEM을 이용하여 단일시간대만 역시간처리를 수행해 새롭게 정당화할 상태를 찾는다. 그리고 다시 상태정보를 이용하여 정당화를 계속한다. 〈그림 4〉는 상태정당화의 한 예를 보여준다. 상단표는 무고장회로에서의 전에 생성된 벡터와 대응상태를 나타낸다. 지금 01x를 활성화상태라고 하면 이 상태는 상태정보 011을 포함하므로 벡터 V4를 인가한다. 이는 활성화상태 01x를 만족하지 않으므로 벡터 V3을 다시 추가 인가한다. 아직도 만족하지 못하므로 1 프레임을 증가하고, V2를 인가한다. 여기서 초기상태를 만족하므로 이 계열 V2, V3, V4는 정당화계열이 된다.

HITEC에서는 고장시뮬레이터에서 얻은 정보를 이용하여 무해고장(redundant) 혹은 테스트곤란한 고장을 효율적으로 테스트할 수 있다. 고장시뮬레이션의 수행에서 어느 한 PPO에 어떤 고장들이 어느 시간대에 전파되었나 하는 정보를 얻을 수 있다. PPO에 전파된 고장에 대해 전파된 시간대로부터 역으로 벡터를 인가하면서 그 고장이 활성



〈그림 4〉 무고장회로 상태정보 이용 예



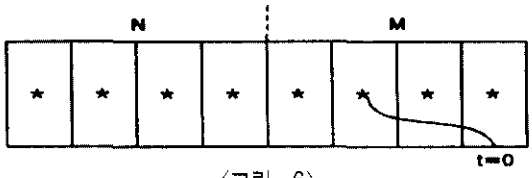
〈그림 5〉 고장시뮬레이션의 정보 이용 예

화되고 PPO에 전파될 때까지 계속 수행한다. 다음에 PPO에 도달된 고장영향을 순방향처리로 PO로 전파시킨다. 예로 〈그림 5〉를 보자. 고장시뮬레이터로부터 목표고장이 시간대 12에서 한 PPO에 전파되었다는 정보를 갖고 있다고 하자. 벡터 V12를 인가하고 PPO에 D가 전파되었는지 확인한다. 어느 PPO에도 D가 전파되지 않았음으로 V11을 인가하고, 다시 V10을 인가하면 한 PPO에 D가 전파된다. 따라서 벡터 V10, V11, V12는 정당화된 상태에서 목표고장을 활성화 하고 이를 PPO로 고장을 전파한 것이 된다. 이와 같이 PPO에 전파된 고장영향을 새로운 PI를 인가하여 FTP로 PO로 전파한다. 이 방법은 테스트하기 어려운 고장을 테스트하는 데 효율적이다.

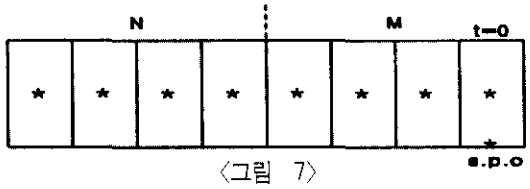
2. BACK 알고리듬

BACK 알고리듬[12]은 역방시간처리(RTP)만으로 테스트생성을 하는 EBT(extended backtrace)법 등[9-11]을 개량한 생성법이다.

EBT법은 고장을 검출하기 위해서는 적어도 하나의 외부출력에 활성화치(D 혹은 D)가 나타나야 한다는 사실에 기초하고, 고장점으로부터 PO에 이르는 하나의 활성화 경로를 미리 선택하고, 이 PO로부터 시작하여 선택된 활성화 경로를 따라, 회로적으로 또한 시간적으로 RTP를 수행하여 정당화한다(그림 6). 테스트 생성 동안에는 회로 상태의 저장을 위해 오직 현시간대(current time frame)와 전시간대(previous time frame)의 두 시간대만 이용하므로 메모리 문제가 해결된다. 그러나, EBT법은 topology 정보만을 이용하여 활성화 경



〈그림 6〉



〈그림 7〉

로를 잘못 선택하게 되는 경우가 많을 수 있고, 또한 탐색 활성화경로가 너무 많은 문제점이 있다.

BACK 알고리듬은 활성화 경로를 미리 선택하는 EBT법과는 달리 활성화된 주출력(sensitized primary output : spo)을 미리 선택하고, 이 값을 RTP로 정당화하여 테스트 생성을 한다(그림 7). BACK 알고리듬의 전체 과정은 오직 RTP로만 행하므로 메모리 문제를 해결할 수 있고, 또한 외부출력의 수도 많지 않기 때문에 BACK 알고리듬은 모든 가능한 외부출력에 대해 시도할 수 있다.

BACK 알고리듬에서는 활성화 외부출력을 선택하는 것이 중요한데, 이 활성화 외부출력을 정확히 선택하기 위하여 구동 용이도(drivability)를 이용한다. 한 신호선의 구동 용이도는 고장점으로부터 그 신호선까지 고장의 영향을 구동하기 어려운 정도를 나타낸다. 이 구동 용이도에 따라 PO와 활성화 값을 선택한다. 그리고 이 PO로부터 고장위치로 역방 처리로 정당화한다. 현시간대에 필요한 모든 값을 정당화하는 데 상태노드값의 정당화가 필요하면 시간대를 증가시켜 반복으로 정당화를 수행한다.

신호치 정당화 과정을 가속화하기 위하여 BACK 알고리듬은 SPLIT 값 모델[16]을 사용한다. SPLIT 값 모델은 9치 모델의 변형된 형태로, 9치를 무고장회로와 고장회로의 각각 3치(0, 1, X)를 갖는 2세트로 분할한다. 무고장회로와 고장회로의 상태를 독립적으로 저장하고 독립적으로 정당화함으로써, 각 회로의 정당화는 5치 모델과 같이 적은 탐색 공간으로 수행 가능하다. 9치 모델의 경우에는 무고장회로와 고장회로의 값을 동시에 저장하고 동시에 정당화하여야 하므로, 어느 한 회로에 충돌이 발생하면, 두 회로 모두 시도철회를 해야 하므로 충돌이 발생하지 않은 다른 회로까지 쓸모 없이 고려하게 되어 낭비를 초래한다. 그러나, SPLIT 모델에서는 무고장회로와 고장회로를 서로 독립된 회로로 가정하여 각각 정당화하므로, 두 회로를 동시에 정당화하는 것보다 탐색공간이 적어 단순하게 된다. 또한 SPLIT 모델은 실제로 무고장회로와 고장회로는 독립적이 아니고, 고장점에 의해 영향을 받는 회로 부분에서만 다르고 거의 동일한 회로 상태를 갖는 것에 고려하여, 무고장회로와 고장회로 사이의 관계(relation)를 UNKNOWN, DIFFERENCE, EQUIVALENCE로 정의하고, 이 관계 정보를 이용하여 탐색공간을 더욱 줄일 수 있도록 하였다.

IV. Simulation-based 테스트생성법

대규모 회로에 대해 topological 테스트생성 방법을 사용하면 후방추적(backtrace)시 거대한 탐색공간으로 테스트생성이 매우 어렵다. 따라서 전방처리로만 테스트생성이 가능하다면 후방추적의 복잡함을 피할 수 있어 테스트생성이 보다 용이해질 수 있다.

simulation-based 테스트생성법은 고장시뮬레이션을 이용하여 FTP만을 수행함으로써 대규모 회로에 대해서도 테스트생성을 할 수 있다. 코스트값이나 경험적인 규칙을 적용하여 무작위로 입력벡터를 생성하고, 생성된 벡터로 고장시뮬레이션을 하여 고장검출을 한다. 공간탐색의 어려움 없이 많은 고장을 손쉽게 검출할 수 있다. 그러나 효율적인 입력벡터를 구하지 않으면 낮은 고장검출율이 얻고, 특히 테스트할 수 없는 고장을 규명할 수 없는 문제가 있다.

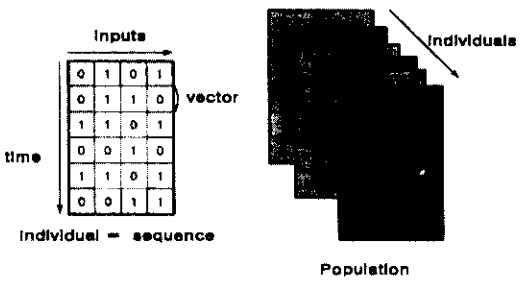
Seshu[17] 이후 많은 simulation-based 테스트생성법이 제안되어 왔다[18-26]. 고장시뮬레이션을 사

용하여 무작위 벡터집합을 평가하고 각 시간대에 최선의 벡터를 선택하거나[18], 가중치 무작위 패턴 생성기를 이용하는 방법[19]들도 제안되었다. Cheng 등이 개발한 CONTEST[20]에서는, 동적 제어용이도(controllability)와 관측용이도(observability)를 기초로 하여 현재 벡터에 한 비트를 변경하여 테스트벡터를 얻고, 동시 고장시뮬레이션을 이용하여 얻어진 테스트를 평가하였다. 이와 같은 테스트생성법들은 topological 테스트생성법보다 테스트생성시간은 비교적 짧지만, 생성된 테스트 집합이 매우 길고 테스트곤란 고장에 대해서는 자주 검출이 안된다.

최근에는 유전알고리듬(genetic algorithm:GA)[27]을 이용한 Simulation-based 테스트생성법이 다수 제안되었다[21-26]. 이 방법은 대규모회로에도 비교적 고장검출율도 높으면서 고속으로 테스트생성이 가능한 테스트생성법으로 알려져 있다. 본 장에서는 유전알고리즘에 관해 설명을 하고, 이를 이용한 테스트생성법인 GATTO 알고리즘[23]도 병행하여 기술한다.

유전알고리즘(GA)은 자연에서 생명체 특성을 개선하는 방식을 원용한 진화 알고리즘으로, 탐색이나 최적문제의 해를 구하는데 이용되고 있다. GA에서는 한 집단(population)이 개체(individual)들로 이루어지고 각 개체는 유전자(gene)들의 열(string)로 주어진다(그림 8).

각 개체는 평가함수(evaluation function)에 의해 주어지는 fitness 값을 갖는데 이 값은 그 개체가 최적해에 가까운 정도를 나타낸다. 부모집단에서 적응력 있는 개체는 살아남고, 그렇지 못한 개체는 없어지는 대신에 새로운 개체를 생성 삽입하여 자손 세대의 집단을 형성한다. 이 과정은 무작위나 특정 방식으로 형성된 초기집단으로부터 시작하고, 교차(crossover), 돌연변이(mutation) 등의 진화처리를 이용하여 집단의 fitness 값의 개선이 없을 때까지, 혹은 최대 세대(generation)수에 도달할 때까지 계속된다.



〈그림 8〉 집단과 개체

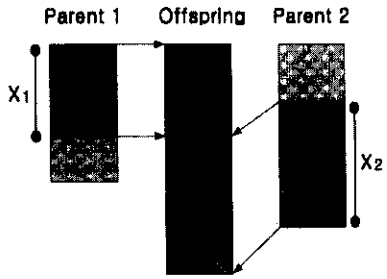
1) 교차(crossover)는 기존에 가장 낮은 fitness 값을 가진 개체 대신 집단에 삽입할 새 개체를, 선택된 두 개체의 열을 결합해서 얻는 방법이다. 각 개체의 선택 확률은 개체의 fitness 값에 비례하게 한다. 따라서 새 개체는 교체대상 개체보다 더 큰 fitness 값을 가진다. 따라서 이 과정은 최적해를 얻는 쪽으로 진행된다.

2) 돌연변이(mutation)에서는 선택된 개체의 유전자(gene)가 무작위로 바뀐다. 이것은 아직 탐색되지 않은 영역으로 들어가도록 진행된다.

GA를 테스트생성에 적용할 때 집단은 테스트집합으로, 개체는 고장을 검출하는 하나의 테스트계열로, 열은 한 입력벡터에 대응된다. 테스트생성에서는 주어진 한 테스트계열에 대한 해의 근접 정도를 나타내는 효율적인 평가함수를 정하는 것이 중요하다. GATTO[23]에서는 평가함수로 1) 고장영향이 전파된 가중화 게이트 수(가중치는 관측용이도에 의해 정해짐) 2) 고장영향이 전파된 가중화 FF수를 파라메터로 이용한다. 이는 평가함수의 값이 클수록 즉 FF이나 게이트에 전파된 고장이 많을수록 PO에 전파되어 고장검출이 될 가능성이 많을 것으로 평가하기 때문이다.

GATTO[23]는 3단계를 통해 테스트생성을 한다. 단계 1에서는 미검출 고장으로부터 목표 고장을 선택하고, 단계 2에서는 목표고장에 대한 테스트계열을 생성하고, 단계 3에서는 테스트계열에 대한 고장시뮬레이션과 고장제거(fault drop)를 수행한다. 단계 1과 단계 2를 좀더 구체적으로 설명하면 다음과 같다.

1) 단계 1—무작위 탐색을 이용하여 목표고장을 선택한다:먼저 L개의 벡터를 가진 M개의 계열을 무작위로 생성한 집단에 대해 고장시뮬레이션을



〈그림 9〉 교차

한다. 적어도 하나 이상의 고장을 활성화한다면 (평가함수의 fitness 값이 일정 이상이 됨) 평가함수의 값이 최대인 고장을 목표고장으로 선택한다. 여기서 검출된 고장은 고장제거 된다.

2) 단계 2-GA를 이용하여 목표고장에 대한 테스트계열을 생성한다: 단계 1에서 얻은 집단으로부터 시작한다. 각 계열에 대해 fitness 값이 주어진다. 새 집단이 진화로부터 생성된다. 즉 fitness 값이 큰 계열은 남고 나머지는 교차와 돌연변이의 진화 작업에 의해 교체된다. 교차는 현재 집단에서 2개의 부모개체(parents)를 선택하고 무작위로 각 개체가 가질 부분계열길이를 생성하여, 각 개체의 부분계열을 결합해서 자손개체(offspring)를 얻는다(그림 9). 부모개체의 선택은 개체의 fitness 값에 비례하여 선택된다. 한편 돌연변이는 하나의 새로이 생성된 테스트계열을 선택해 그 중 1비트의 값을 반전시킨다.

이와 같이 새로이 생성된 집단에 대해 고장시뮬레이션을 하고 단계 1~3을 반복하여 테스트생성을 한다.

이밖에도 GA를 이용한 테스트생성이 많이 제안되어 있다. CRIS[21]는 활동도에 의한 회로 내의 균형정도를 평가함수로 하고 논리시뮬레이션을 사용하여 계산하는 방법으로, 테스트생성 시간은 짧으나 고장검출율이 낮다. GATEST[22]는 평가함수에 고장시뮬레이터에 의해 얻어진 결과를 이용하여 한번에 한 벡터만 생성하는 방법으로 콤팩트하나 테스트생성 시간이 길다. 또한 테스트불가고장에 대해 식별 가능한 topological 방법과 고속 생성 가능한 GA를 이용한 방법을 결합한 효율 좋은 테스트생성법도 제안되어 있다[24-26].

V. Symbolic 테스트생성법

symbolic 테스트생성은 회로의 구조정보를 부울함수로 표현하거나 혹은 상태 천이 그래프(state transition graph: STG) 등으로 표현하여 고장전파와 상태정당화를 효율적으로 수행하는 테스트생성법이다[28-31]. 이는 고장검출율이 아주 높고 테스트생성시간은 짧으나 FF수가 수십 개 정도의 규모의 순차회로에 밖에 적용이 안되는 단점을 갖고 있다.

본 장에서는 STEED[29]와 PMT법[30,31]에 관해 기술한다.

1. STEED

STEED[29]는 회로의 구조 정보로서 조합회로부에 대한 ON/OFF 집합을 이용하고, 리셋 상태를 가정한 순차회로에 대한 테스트생성 알고리듬이다. STEED에서는 조합회로부의 테스트생성과정, 무고장회로에서의 상태정당화(state justification)과정, 무고장회로에서의 상태구별화(state differentiation) 과정의 세 부분으로 나누어 테스트생성을 한다.

1) 먼저 고장이 주어지면 조합회로에서의 고장으로 간주하고 조합회로 테스트생성 알고리듬을 이용하여 테스트생성을 한다. 이 과정에서는 무해고장이 아닌 한에 있어서 PO나 PPO에 고장의 영향이 전파되고, 입력벡터와 활성화상태가 생성된다.

2) 생성된 활성화상태에서 리셋 상태에 이르는 정당화순서열(justification sequence)을 무고장회로 상에서 RTP로 구한다. 각 시간대마다 정당화해야 할 활성화상태의 값을 ON/OFF 집합에 대한 일련의 큐브 교차(cube intersection)를 통해 얻고 지고, 테스트 벡터와 현재 상태를 생성한다. 이를 리셋 상태까지 RTP를 수행하여 하나의 무고장회로의 정당화순서열을 구한다. 이 정당화순서열은 고장회로에서도 활성화 상태가 정당화 되는지를 고장시뮬레이션을 통해 확인한다. 만약 활성화상태가 정당화되면, 그 고장에 대한 정당화순서열이

되고, 그렇지 않으면, 리셋에서 처음으로 PPO에 고장이 전파되는 순서열로 이용한다.

3) 마지막으로, 활성화상태가 정당화되었으면 고장의 영향이 전파된 PPO에 외부출력으로 전파하는 구별화순서열(differentiation sequence)을 FTP로 구한다. PPO의 무고장값과 고장값으로 이루어진 무고장-고장 상태쌍(fault-free faulty state pair)에 대해 무고장회로에서의 외부출력의 ON-OFF 집합 표현을 이용하여 일련의 큐브 교차 조작을 통해 얻는다. 테스트는 정당화순서열, 구별화순서열을 접합(concatenation)하여 고장시뮬레이션을 통해 다시 목표 고장의 테스트를 검증한다.

2. PMT법

STEED 등[28, 29]에서는 순차회로의 구조정보인 ON/OFF 집합 등을 명시적(explicit)으로 표현함으로써 표현량이 방대하게 되는 문제점을 갖고 있다. 그러나 근래에 제안된 BDD(binary decision diagram)[32, 33]와 비명시적 상태 열거(implicit state enumeration)법[34]을 이용하면 상태(state) 등의 회로구조 정보를 효율적으로 표현, 활용할 수 있다.

PMT(product machine traversal : 적기계탐색)법은 <그림 1>과 같은 무고장회로와 고장(CUT에 고장을 가정)회로로 구성된 적기계를 고려하고, 적기계의 상태천이그래프 상에서, 리셋 상태로부터 시작하여 적기계의 출력을 1(무고장회로와 고장회로의 출력이 서로 다를 때)로 주는 상태와 입력벡터를 탐색한다. 만약 이와 같은 상태와 입력이 존재하면 그 상태에서 리셋 상태로의 경로를 후방추적(backtrace)으로 얻음으로써 테스트순서열을 생성한다. 그렇지 않으면 그 고장은 순차적인 무해고장으로 판별한다.

이 과정을 명시적(explicit)으로 단순 처리하면 계산량이 방대하게 된다. PMT법에서는 BDD의 데이터 구조를 이용한 비명시적 상태 열거(implicit state enumeration)법[34]을 도입한다. 비명시적 상태 열거는 상태 집합을 특성 함수(characteristic function : 어느 한 집합이 전체 집합의 부분집합이 되면 1이 되는 함수)라 하는 논리함수로 표시하고, 상계산(현 상태 집합에서 도달 가능한 다음 상태 집합의 계산) 및 해의 존재 여부 등을 논리함수로 처리한다. 이 방법으로 방대한 상태천이그래프를 효율적으로 표현하고, 연산을 고속으로 수행할 수 있다.

PMT법은 적기계의 상태천이그래프상의 모든 상태쌍에 대해 탐색하기 때문에, 무해 고장이 아닌 고장에 대하여, 충분한 시간과 기억량을 가지면 반드시 테스트 생성이 가능한 완전 알고리듬이다. 따라서 STEED와 같이 고장 테스트를 위하여 별도의 고장 시뮬레이션을 할 필요가 없다. 또한 적기계에서 폭우선으로 탐색되기 때문에 생성된 테스트는 주어진 고장에 대한 최단 순서열이 되고, 특히 테스트 곤란한 고장이나 무해 고장에 대해서도 빠른 시간 내에 처리할 수 있어 유용하다. 그러나 FF수가 수십 개 정도의 규모의 순차회로에 밖에 적용이 안되는 단점을 갖고 있다.

VI. 결 론

본고에서는 순차회로의 테스트생성법을 크게 topological 테스트생성법, simulation-based 테스트생성법, symbolic 테스트생성법의 세 부분으로 분류하여 소개하였다.

topological 테스트생성법은 손쉽게 구현가능하나 테스트생성시간을 많이 요하고, symbolic 테스트생성법은 고장검출율이 높으나 큰 회로에 대해서는 테스트생성이 어렵고, simulation-based 테스트생성법은 대규모 회로에도 적용가능하나, 고장검출율이 나쁘다.

이와 같이 다수의 순차회로의 테스트생성법이 제안되고 있지만 실용적인 관점에서는 아직도 어려운 문제로 남아 있다. 따라서 회로분할 방법의 적용 혹은 병렬처리 등으로 대상회로를 보다 더 대규모화하고 테스트생성시간을 줄일 수 있는 순차회로의 테스트생성법에 관한 연구가 필요하다. 또한 순차회로의 테스트생성법 단독보다는 부분스캔(partial scan) 설계방식 등의 테스트용이화설

계(design for testability)와 결합한 보다 더 효율적인 테스팅 방법의 연구도 필요하다.

참 고 문 헌

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," IBM Journal of Research and Development, vol. 10, no. 4, pp. 278-291, July 1966.

[2] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. on Computers., vol. C-30, no. 3, pp. 215-222, Mar. 1981.

[3] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," IEEE Trans. on Computers., vol. C-30, no. 12, pp. 1137-1144, Dec. 1983.

[4] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG," in ACM/IEEE Proc. Design Automation Conf., pp. 502-508, June 1987.

[5] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," IEEE Trans. on Computer-Aided Design, vol. CAD-7, no. 1, pp. 126-137, Jan. 1988.

[6] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in Proc. Int. Symposium Circuits and Systems, pp. 1929-1934, June 1989.

[7] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," in ACM/IEEE Proc. Design Automation Conf., pp. 462-468, June 1977.

[8] M. Abramovici, M. A. Breuer, and A. D. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, 1990.

[9] R. Marlett, "EBT: A comprehensive generation technique for highly sequential circuits," in Proc. 15th ACM/IEEE Design Automation Conf., pp. 335-339, June 1978.

[10] S. Mallela and S. Wu, "A sequential circuit test generation system," in Proc. IEEE Int. Test Conf., pp. 57-61, Nov. 1985.

[11] R. Marlett, "An effective test generation system for sequential circuits," in Proc. 23rd ACM/IEEE Design Automation Conf., pp. 250-256, June 1986.

[12] W.-T. Cheng, "The BACK algorithm for sequential test generation," in Proc. Int. Conf. Computer Design, pp. 66-69, Oct. 1988.

[13] G. R. Putzolu and J. P. Roth, "A heuristic algorithm for the testing of asynchronous circuits," IEEE Trans. Computers, vol. C-20 no. 6, pp. 639-647, June 1971.

[14] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in Proc. European Conf. Design Automation, pp. 214-218, Feb. 1991.

[15] D. H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," in Proc. Int. Conf. Computer-Aided Design, pp. 446-449, Nov. 1991.

[16] W. T. Cheng, "Split circuit model for test generation," in Proc. 25th ACM/IEEE Design Automation Conf., pp. 96-101, June 1988.

[17] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," IRE Trans. Electron. Computers, vol. 11, no. 8, pp. 459-465, Aug. 1962.

[18] M. A. Breuer, "A random and an algorithmic technique for fault detection

test generation for sequential circuits," IEEE Trans. Computers, vol. C-20, no. 11, pp. 1364-1370, Nov. 1971.

[19] H. D. Schnurman, E. Lindbloom, and R. G. Carpenter, "The weighted random test-pattern generator," IEEE Trans. Computer, vol. C-24, no. 7, pp. 695-700, July 1975.

[20] V. D. Agrawal, K.-T. Cheng, and P. Agrawal, "CONTEST: A concurrent test generator for sequential circuits," in Proc. Design Automation Conf., pp. 84-89, June 1988.

[21] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," in Proc. Int. Conf. Computer-Aided Design, pp. 216-219, Nov. 1992.

[22] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework," in Proc. Design Automation Conf., pp. 698-704, June 1994.

[23] P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "An automatic test pattern generator for large sequential circuits based on genetic algorithms," in Proc. Int. Test Conf., pp. 240-249, Oct. 1994.

[24] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Iterative[simulation-based genetics+deterministic techniques]= complete ATPG," in Proc. Int. Conf. Computer-Aided Design, pp. 40-43, Nov. 1994.

[25] E. M. Rudnick, J. H. Patel, "Combining deterministic and genetic approaches for sequential circuit test generation," in Proc. Design Automation Conf., pp. 183-188, June 1995.

[26] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Alternating strategies for sequential circuit ATPG," in Proc. European Design and Test Conf., pp. 368-374, Mar. 1996.

[27] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[28] H-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test generation for sequential circuits," IEEE Trans. Computer-Aided Design, vol. CAD-7, no. 7, pp. 1081-1093, Oct. 1988.

[29] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation and verification for highly sequential circuits," IEEE Trans. Computer-Aided Design, vol. CAD-10, no. 5, pp. 652-667, May 1991.

[30] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration," IEEE Trans. Computer-Aided Design, vol. CAD-12, no. 7, pp. 935-945, July 1993.

[31] H. Choi, T. Kohara, N. Ishiura, I. Shirakawa, and A. Motohara, "Test generation for sequential circuits based on Boolean function manipulation," IEICE Trans., vol. J76-A, no. 6, pp. 835-843, June 1993 (in Japanese).

[32] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. on Computers, vol. C-35, no. 8, pp. 677-691, Aug. 1986.

[33] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," in Proc. 27th ACM /IEEE Design Automation Conf., pp. 52-57, June 1990.

[34] H. J. Touati, H. Savoj, B. Lin, R. K.

Brayton, and A. Sangiovanni-Vincentelli, "Implicit state enumeration of finite state machines using BDD's," in Proc. IEEE Int. Conf., pp. 40-45, June 1990.

저 자 소 개

崔 湜 鎔

1957년 2월 1일생, 1980년 2월 서울대학교 공과대학 전자공학과 공학사, 1982년 2월 한국과학기술원 전기및전자공학과 공학석사, 1994년 3월 오사카대학 전자공학과 공학박사, 1982년 3월~1985년 8월 삼성반도체(주) 연구소 연구원, 주임연구원, 1985년 9월~1996년 8월 부경대학교 전자공학과 전임강사, 조교수, 부교수, 1996년 9월~현재 충북대학교 부교수, 〈주관심 분야: VLSI 테스팅, VLSI설계〉