상태 정보 학습을 이용한 새로운 순차회로 ATPG 기법

정회원 이 재 훈*, 송 오 영*

New Test Generation for Sequential Circuits Based on State Information Learning

Jaehoon Lee*, Ohyoung Song* *Regular Members*

요 약

조합형 회로에 대한 테스트 패턴 생성의 문제는 거의 만족할 만한 수준에 도달한데 반해 순차형 회로에 대한 테스트 패턴 생성은 여전히 많은 연구를 필요로 하고 있다. 본 연구에서는 효율적인 검사 패턴 생성을 위하여 검사 패턴 생성 과정에서 탐색되어지는 상태 공간 정보의 효율적으로 저장하고, 그렇게 저장된 상태 공간 정보를 이용하여 효율적으로 검사패턴을 생성하는 알고리즘을 제안한다. 그리고 제안된 알고리즘과 기존의 결정적 검사 패턴 생성 알고리즘을 실험을 통하여 비교함으로써 제안된 알고리즘의 효율성을 검증한다.

ABSTRACT

While research of ATPG(automatic test pattern generation) for combinational circuits almost reaches a satisfiable level, one for sequential circuits still requires more research. In this paper, we propose new algorithm for sequential ATPG based on state information learning. By efficiently storing the information of the state searched during the process of test pattern generation and using the state information that has been already stored, test pattern generation becomes more efficient in time, fault coverage, and the number of test patterns. Through some experiments with ISCAS '89 benchmark circuits, the efficiency of the proposed method is shown.

I. 서 론

반도체 제조 및 설계 기술의 발달로 단일 칩에 집적되는 논리 게이트의 수는 백만 단위를 넘어선 지 오래다. 칩의 집적도가 증가함에 따라 신뢰성을 보장하기 위한 테스트는 점점 더 어려워지고 있다. 이는 고집적된 논리 회로를 상대적으로 매우 적은 수의 입출력 핀으로 테스트해내야 하기 때문이다.

회로는 메모리의 유무에 따라 조합회로와 순차회로로 나뉜다. 이 중 조합회로의 테스트패턴 생성 문제는 지난 이십여 년간의 연구[8,5,3,9,7]를 통해 만족할 만한 수준에 도달한 것으로 평가되고 있다. 그러나 순차회로의 패턴생성 연구는 꾸준히 진행되어 왔음에도 불구하고 여전히 만족할 결과를 보이지 못하는 것으로 평가되고 있다.

순차회로가 조합회로에 비해 테스트 패턴 생성이 어려운 이유는 다음과 같다.

· 첫째, 테스트패턴 생성을 위해 탐색해야 할 검색 공간의 크기가 같은 크기의 조합회로에 비해 매우 크다. p개의 입력을 갖는 조합회로의 경우 검색 공간이 2p인데 반해 p개의 입력과 n개의 플립플롭을 갖는 순차회로의 검색 공간은 2p+9n 이다[9].

* 중앙대학교 전자전기공학부(song@jupiter.cie.cau.ac.kr)
논문번호 : 99511-1230, 접수일자 : 1999년 12월 30일
※ 본 연구는 한국과학재단 특정기초 연구지원사업(과제번호: 95-0100-2103-3)에 의하여 수행되었습니다.
※ 본 연구는 중앙대학교의 연구기자재 구입 지원 프로그램의 도움을 받아 수행되었습니다.

· 둘째, 순차회로의 피드백 루프로 인하여 테스트 패턴 생성 시 고장효과가 계속적으로 그 루프를 통하여 순환할 수가 있다.

· 셋째, 고장이 매 시간 프레임 상에 존재하므로 다중고장(multiple fault)에 대한 처리를 해야한다.

이러한 순차회로에 대한 테스트의 어려움으로 인하여 스캔설계 기법과 같은 테스트용이도를 고려한 설계[11]가 일반화되고 있다. 그럼에도 여전히 순차회로에 대한 ATPG가 중요한 문제로 인식되고 활발히 연구되는 이유는 다음과 같다.

· 첫째, 스캔설계회로에 비해 설계의 부담을 줄일 수 있다. 스캔의 경우 스캔체인 구성을 위하여 추가의 로직을 필요로 함으로 필연적으로 회로의 면적 증가와 성능 저하라는 문제점을 초래한다.

· 둘째, 테스트 시간을 줄일 수 있다. 순차형 패턴을 이용한 테스트는 정상동작 속도에서 테스트가 가능한데 반해, 완전스캔의 경우 매 패턴을 인가할 때마다 스캔체인의 길이에 해당하는 클럭시간을 요구한다.

· 셋째, 고장검출율을 높일 수 있다. 정상동작 속도의 테스트가 가능하므로 지연고장(delay fault)들이 추가로 검출될 수 있다.

· 넷째, 모든 플립플롭이 스캔으로 처리될 수 없다. 어떤 플립플롭은 임계경로(critical path)에 있어 스캔체인에 연결 될 수 없는 경우가 있다.

순차회로의 자동 테스트패턴 생성에 관해서는 크게 세 가지 방향으로 많은 연구들이 진행되어 왔다:

(1) 결정적 테스트패턴 생성(deterministic test pattern generation) [2,4,6,12,13,14,15,16]

(2) 시뮬레이션에 기반을 둔 테스트패턴 생성(simulation-based test pattern generation) [17,18,19,20, 21,22]

(3) (1)과 (2)의 혼합 방식 [23,24,25]

본 논문에서는 결정적 테스트패턴 생성의 효율성을 더욱 향상시키기 위하여 상태정보 학습을 이용하는 기법을 제시한다. 순차회로는 시간후레임(time frame)별로 조합회로 셀을 반복시키는 "반복 배열 모델(iterative array model)"[11]로 모델링된다. 순차회로의 결정적 테스트패턴 생성은 일반적으로 다음의 세 가지 단계로 구성된다.

고장효과 생성 및 전파: 고장지점(fault origin)에서 최초로 고장효과(fault effect)를 생성시키고 고장의 효과를 주요출력(primary output: PO)이나 의사출력(pseudo-primary output: PPO)으로 보내는 단계이다. 이 과정에서는 의사입력을 주요입력(primary input: PI)으로 의사출력을 주요출력으로 간주하고 조합회로 테스트패턴 생성 방법을 그대로 이용한다.

상태정당화(state justification): 조합회로와 달리 순차회로는 메모리소자의 값에 따라 회로의 상태가 변화한다. 따라서 고장효과 생성 및 전파 과정(fault effect propagation)에서 구한, 의사입력(pseudo-primary input: PPI)과 주요입력에 관한 입력패턴이 특정한 상태를 필요로 하는 경우가 있다. 이때 그 특정상태를 리셋상태나 무지상태(unknown state)로부터 유도해 내는 입력시퀀스를 구하는 단계이다.

상태차별화(state differentiation): 어떤 고장의 경우는 고장의 효과가 주요출력으로 전파되지 못하고 의사출력으로만 전파되는 경우가 있다. 이때 의사출력에 도착한 고장의 효과를 주요출력으로 전파시키는 단계이다.

순차회로를 위한 결정적 테스트패턴 생성 알고리즘의 연구는 주로 다음 세 가지 방향으로 진행되어 왔다. 첫 번째는 가장 고전적인 방법으로 순차회로의 상태천이 도표를 이용한 방법이다. 이 방법은 모든 상태에 대한 천이관계를 메모리 상에 표현하고서 이를 이용하여 상태공간을 탐색하는 방법이다. 이러한 형태의 발전된 방법으로 묵시적 적기계탐색(IPMT: implicit product machine traversal)법[2]이 있다. 이 방법은 적기계 상에서 리셋상태로부터 도달가능 상태 쌍을 묵시적으로 열거하여, 불일치 상태 쌍을 폭 우선 방식(breadth first search) 혹은 폭 우선/깊이 우선 방식(depth first search)으로 탐색하는 테스트 생성법이다. 이 방법은 검출 가능한 고장에 대해서는 반드시 테스트생성이 가능하다는 의미에서 완전한 알고리즘이다. 하지만 회로의 규모가 커지게 되면 다음상태 열거 처리에 현실적인 허용량을 초과하는 방대한 기억용량을 초과하게 됨에 따라 높은 고장검출율을 얻을 수 없게 되는 문제가

있다.

두 번째로는 D-알고리즘을 확장한 방법이다. 이러한 방법으로는 HITEC[6]이 있는데, 이 방법은 순차 회로를 동적으로 확장시켜 조합회로로 모형화함으로써 패턴생성을 시도하는 방법이다. 이 방법은 순차 회로 패턴생성에 그 동안 개발된 우수한 조합회로 패턴생성 알고리즘을 이용할 수 있는 장점이 있지만, 하나의 패턴생성에 고려해야 하는 시간의 폭이 늘어날 경우 막대한 양의 메모리를 요구하게 되는 단점이 있다.

세 번째는 상태천이 정보를 부분적으로 저장하는 방법이다[4]. 이 방법은 ON/OFF 집합의 개념을 기반으로 결정적 테스트패턴 알고리즘을 구성하였다. 이 방법은 막대한 기억 용량을 요구하는 각각의 상태천이 정보를 그대로 메모리에 저장하는 대신, 회로 내의 각각의 메모리소자의 값을 '0'으로 만드는 주요입력과 의사입력의 조합(OFF 집합)이나 '1'로 만드는 조합(ON 집합)을 이용하는 것이다. 상태공간의 탐색은 큐브 형태로 표현되는 ON/OFF 집합을 교차 연산함으로 이루어진다. 이 방법은 메모리 요구량이 작으면서도 효과적으로 상태공간 탐색을 가능케 하는 장점이 있다.

본 연구에서는 상태정보 학습을 이용하는 테스트패턴 생성 알고리즘의 효율성을 보이기 위해서 [4]에서 제시된 알고리즘에 상태정보 학습 방법을 구현하였다. 본 논문에서 제시되는 상태정보 학습 방법은 다른 결정적 테스트패턴 생성 방식에서도 적용될 수 있다. 자동 테스트패턴 생성(ATPG)기는 결정적 테스트패턴 생성(DTPG: deterministic test pattern generation)기와 고장시뮬레이터로 구성된다. 따라서 효율적으로 패턴을 생성하기 위해서는 두 가지 알고리즘 모두 효율적이어야 한다. 본 연구에서는 고장 시뮬레이션 알고리즘은 [10]에 제시된 방법을 기반으로 하였다.

Ⅱ 상태정보 저장 및 학습 기법

효율적인 패턴생성의 문제는 패턴생성 과정에서 얻어지는 수많은 정보들을 얼마나 효율적으로 이용하느냐에 달려있다. 이를 위하여 우리는 그림 1에서 보듯이 상태천이 정보를 나무구조(tree)의 형태로 저장하는 방법을 제안한다. 여기에서 한 노드는 어떤 시간후레임(t)의 주요입력 벡터와 다음상태(next state), 즉 의사출력 값으로 구성되는 무고장 상태천이(state transition) 정보, 및 고장 상태천이 정보를 저장한다. 이와 같은 정보들은 상태정당화 과정에서 얻어 지거나, 혹은 고장시뮬레이션 과정에서 얻어지는 상태천이 정보들이다. 고장 상태천이 정보는 고장시뮬레이션 과정의 결과로 얻어 지며 각 노드의 고장 상태정보 리스트(faulty state information list)에 저장된다.

그림 1은 주요입력이 4개이고 의사입력과 의사출력이 각각 3개인 회로에 대한 것이다. 노드 n2의 경우를 보자. 노드 n2는 시간후레임(t)과 관련되어 있다고 하자. 이때 노드 n1은 시간후레임(t-1)과 관련된다. 노드 n2는 시간후레임(t)의 주요입력 벡터가 v2 (0110)이고 의사출력 벡터가 s2 (110)인 것을 보여 준다. 시간후레임(t)의 의사입력 벡터는 노드 n1의 PPO vector부분에 s1 (010)으로 저장되어 있다. 노드 n3와 노드 n4는 시간후레임(t+1)과 관련된다. 시간후레임(t+1)에 한 개이상의 노드가 관련될 수 있음을 주목하여야 한다. 의사입력 벡터가 s2 (110)인 시간후레임(t+1)에서 주요입력 벡터 v3 (0011)가 인가되어 무고장시뮬레이션을 수행한 결과 의사출력 벡터가 s3 (111)가 되었다고 가정하자. 이 의사출력 벡터 s3는 노드 n3의 PPO vector부분에 저장된다. 만약 다른 주요입력 벡터 v4 (1010)가 노드 n2와 관련된 시간후레임(t)에 인가되어 의사출력 벡터 s4 (000)를 무고장시뮬레이션 결과로 얻었을 경우에는 새로운 노드 n4를 만들어서 s4를 저장한다. 다음은 고장 fault 2를 포함하고 있는 고장회로를 고려하자. 노드 n4와 관련되어 있는 시간후레임(t+1)에서 주요입력 벡터 v4가 인가되었을 때, 고장회로와 무고장회로에서 첫 번째 의사출력의 값이 서로 다르다고 가정하자. 예컨대 무고장회로에서는 0이나 고장회로에서는 1의 값을 가질 경우, 의사출력번호와 고장값 쌍 (PPI 1, 0)을 노드 n4의 고장 상태정보 리스트에 저장한다. 구체적인 저장 방법은 그림 1에 자세히 보여진다. 이와 같은 방식으로 고장효과가 시간후레임(t+1)의 의사출력으로 전달되는 모든 고장들은 고장 상태천이 정보 리스트에 저장된다. 고장목록(fault list)에 있는 각 고장들은 모든 노드의 고장 상태정보 리스트에 저장되어 있는 해당 고장들과 관련시킨다. 만약 어떤 고장이 검출될 경우에 모든 노드의 고장 상태정보 리스트 상에 저장되어 있는 해당 고장들의 고장 상태정보는 고장목록을 참조함으로써 효율적으로 제거할 수 있다.

그림 1에서 보이는 것처럼 학습에 의해서 효율적으로 저장된 상태정보를 이용하여 상태정당화 시퀀스를 효율적으로 구할 수 있는 알고리즘은 다음과

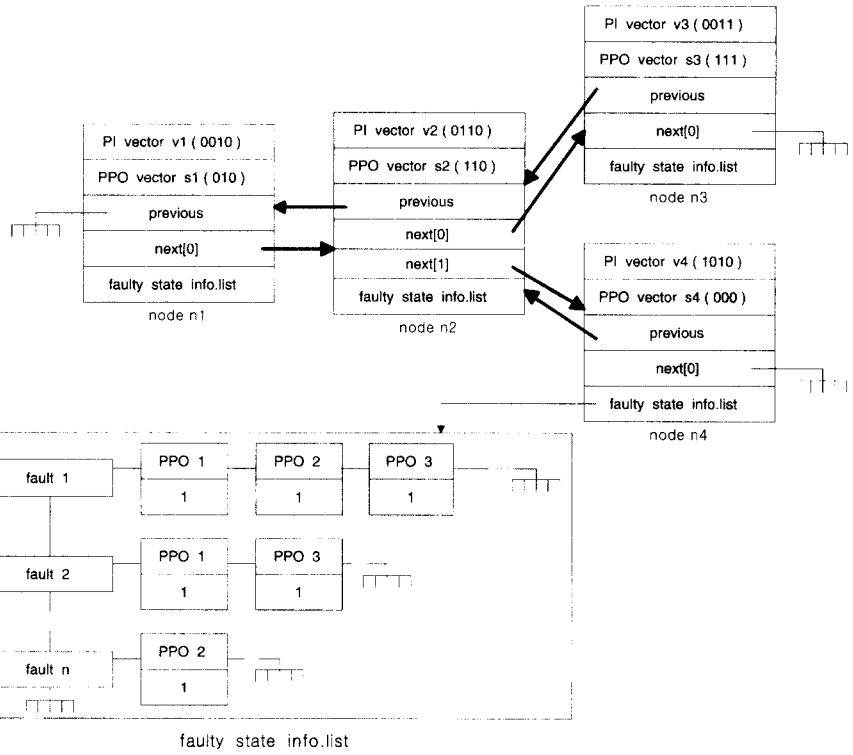


그림 1. 패턴과 상태천이 정보를 동시에 저장하는 자료구조

같다. 어떤 특정한 고장의 고장효과 생성 및 전파 과정에서 그 고장의 효과를 주요출력 혹은 의사출력으로 전파시키는 주요입력 벡터 v와 의사입력 벡터 w가 구해졌다고 가정하자. 의사입력 벡터 w에 대하여 상태정당화를 수행하기 전에, 상태천이 정보의 노드들에 대해 넓이 우선 탐색(breadth first search)을 수행한다. 탐색은 각 노드의 의사출력 벡터 s와 w를 비교하여 s가 w의 부분 상태일 경우, 최초 노드로부터 s를 포함하는 노드까지에 포함되어 있는 주요입력 벡터들을 시간후레임 순서대로 구성한 벡터 시퀀스를 w에 대한 상태정당화 시퀀스로 사용한다. 다음에는 그 고장에 대해서 상태차별화를 수행한다. 상태정당화를 수행하여 상태정당화 시퀀스를 구하는 것은 매우 많은 계산시간을 필요로 하지만 학습된 상태정보를 이용하여 상태정당화 시퀀스를 구하는 것은 매우 효율적이다. 실제 여러 회로에 대한 실험에서 대부분의 상태정당화 시퀀스는 학습된 상태정보를 이용하여 찾을 수 있음을 알 수 있었다.

상태정당화 단계에서 주요입력 값이 부분적으로만 확정돼있어 다음상태들 또한 부분적으로 확정될 경우, 탐색된 상태천이 정보들을 추가의 처리 없이 그대로 이용한다면 정보의 가치가 떨어진다. 상태 A를 "110X", B를 "1101", C를 "11XX"라고 가정해 보자. 그리고 상태 A와 C의 관계처럼 하나의 상태가 다른 상태의 부분 상태가 될 때, "상태 A는 상태 B보다 작다", 역으로 "상태 B는 상태 A보다 크다"라고 하자. 이때 A의 상태정당화 과정에서 이용된 상태천이 정보는 상태 A보다 큰 C의 상태정당화 과정에서 그대로 이용될 수 있다. 그러나 상태 A보다 작은 상태인 B의 정당화에는 이용될 수 없다. 그러나 상태 B의 상태정당화에 이용되었던 상태 천이 정보들은 상태 A와 C의 상태정당화에 모두 이용될 수 있다. 즉, 상태정당화에 유효하게 이용될 수 있는 정보는 보다 작은 상태의 상태정당화 정보인 것이다. 우리는 패턴의 저장 시 입력에 확정되지 않은 값들에 임의의 값을 배정함으로써 상태정당화 단계에서 만들어진 상태천이 정보보다 작은 상태들의 변화 정보를 만들어 낼 것이다.

그림 1에 보여지는 것과 같은 형태로 패턴과 상태천이 정보를 모두 저장하게 되면 여러 가지 이점이 존재하게 된다.

첫 번째로는 상태정당화 과정에서 이용될 수 있는 정보의 양이 훨씬 더 늘어나게 된다. 기존에는 상태정당화 단계에서 이용되었던 정보만을 재 사용하였는데, 본 연구에서 제시된 자료구조를 이용하므로써 상태차별화 단계에 이용된 주요입력 벡터 시퀀스에 의해 천이하는 상태정보나 무작위 패턴에 대한 고장시뮬레이션에서 얻어진 상태천이 정보들까지 모두 저장한다.

두 번째로는 고장시뮬레이션에서 주요출력으로 검출되지 않고 상태변수(의사출력)에만 고장효과를 전파한 고장들의 정보를 재 사용함으로써, 패턴생성 시간을 줄이고 고장검출율을 높일 수 있다. 무고장 회로와 다른 고장 상태정보를 가지고 있는 고장들은 이미 고장의 효과가 상태변수에 도착한 것이므로, 고장효과 생성 및 전파 단계와 상태정당화 단계 없이 상태차별화 단계만을 시도함으로써 패턴을 구할 수 있다. 일반적으로 ATPG에서는 각 고장에 대한 테스트패턴 생성 시간에 제한을 둔다. 그 시간 제한 내에 어떤 고장에 대한 테스트패턴 생성이 불가능할 경우에는 패턴 생성을 포기하게 된다. 본 논문에서 제안한 상태천이 정보의 효율적인 학습을 이용하지 않을 경우, 상태정당화에 많은 계산을 하게 되어 패턴 생성이 중지되는 고장들이 있을 수 있다. 이런 고장들의 고장효과가 이미 상태변수에 도착한 정보를 본 논문에서 제안한 기법에 의해 학습할 수 있게 한다면, 고장효과 생성 및 전파 단계와 상태정당화 단계를 생략하고 상태차별화 단계만을 시도하여 패턴 생성을 할 수 있다. 결과적으로 이 기법은 제한시간의 제약으로 인한 패턴 생성 포기의 가능성을 줄여서 고장검출율을 높일 수 있게 한다.

세 번째로는 고장시뮬레이션 시간을 큰 폭으로 줄일 수 있다. 사전연구에 따르면 상태정당화 과정에서 많은 부분이 이전의 상태정당화에 이용되었던 입력벡터 시퀀스를 재 사용하는 것으로 나타났다. 어떤 회로의 경우에는 생성된 패턴의 90%가 이미 이전에 생성된 패턴과 동일함을 관찰할 수 있었다. 기존의 기법들에서는 매 패턴이 발생될 때마다 이에 대한 고장시뮬레이션을 진행하였다. 하지만 실상 패턴의 일부분이 재 사용된다면 중복된 부분에 대한 고장시뮬레이션을 줄일 수 있다. 그림 2는 pattern1이 재 사용되어 pattern2와 pattern3이 만들어진 것을 보여 준다. pattern1은 3개의 순차적인 입력 벡터들인 v1, v2, v3로 구성되어 있다. pattern2는 pattern1의 고장시뮬레이션 과정에서 입력벡터 v2를 입력시켰을 때 고장의 효과가 현재 시간후레임의 의사출력에 도착한 고장에 대하여 상태차별화만을 시도하여 만들어진 패턴이다. pattern2에 대한 고장시뮬레이션에서 v1, v2 부분은 이미 고장시뮬레이션이 수행되어졌으므로 또 다시 고장 시뮬레이션을 수행할 필요가 없이 v2와 관련된 노드에 저장되어진 무고장회로의 상태와 고장회로의 상태를 그대로 이용해서 v4에 대한 고장 시뮬레이션만을 수행함으로써 pattern2에 대한 고장시뮬레이션을 수행할 수 있다.

네 번째로는 패턴을 나무구조의 형태로 저장하므로 추가의 비용 없이 어느 정도의 패턴축약 효과를 볼 수 있다. 그림 2에서 보듯이 pattern1 전체는 pattern3의 일부분으로 재 사용되었다. 입력 벡터 v5가 나무구조에 저장될 때 v5와 관련된 노드에 저장한 후, 입력 벡터 v3를 저장하고 있는 노드의 자노드로서 관련 지워 둔다. 이런 저장 방식을 이용하면 패턴목록에서 pattern1은 자연히 제거되어 pattern3에 축약되는 효과를 얻게 된다.

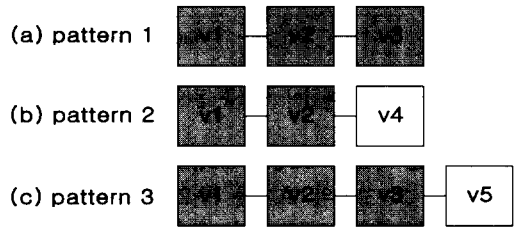


그림 2. 상태 천이 정보를 재 사용해서 생성된 패턴의 예

Ⅲ. 실 험

본 논문의 자동 테스트패턴 생성 알고리즘은 C++언어를 이용하여 SUN Solaris/Ultrasparc 167MHz CPU/256MB 워크스테이션에서 구현되어졌다. 새롭게 제안된 알고리즘의 성능 측정을 위하여 제안된 기법과 기존의 기법[4]을 동일한 워크스테이션에서 수행시켜서 그 결과를 비교하였다. 그리고 테스트패턴 생성을 위한 회로로는 자동 테스트패턴 생성 알고리즘의 성능 측정에 널리 이용되는 ISCAS '89 성능 분석용 회로[1]와 기존의 기법[4]이 발표될 때 성능 분석용으로 이용되었던 회로를 이용되었다. ISCAS '89 성능 분석용 회로에는 플립플롭의 초기 값이 지정되어 있지 않으므로 모든 플립플롭이 '0'인 상태를 초기 상태로 가정했다.

표 1에는 실험에 이용된 회로들의 구조적 특징을

표 1. 실험에 이용된 회로의 구조 분석

circuit	# PIs	# POs	# FFs	# GATEs	# Faults
s386	7	7	6	159	384
s444	3	6	21	181	474
s510	19	7	6	211	564
s820	18	19	5	289	850
s953	16	23	29	395	1079
s1196	14	14	18	529	1242
s1488	8	19	6	653	1486
cse	7	7	4	192	519
sse	7	7	6	130	368
sand	11	9	6	555	1425
planet	7	19	6	606	1482
scf	27	54	8	959	2456
sbc	40	56	28	1011	1985
key	62	48	56	1342	8520
pewd	117	101	112	1873	3483
stage	113	64	64	2700	7227
dsip	228	197	224	3654	6781

분석하여 정리하였다. 표 2에는 각 회로에 대해서 총 고장의 수(total), 우리의 기법을 수행한 결과로 검출된 고장의 수(detected), 검출 불가능 고장의 수(untestable), 검출 포기 고장의 수(aborted), 기존 기법의 고장검출율(old), 우리 기법의 고장검출율(new)을 보여 준다. 여기에서 고장검출율 TFC는 검출 가능한 고장에 대한 검출율로 다음과 같이 구해진다.

$$TFC = \frac{\text{검출된고장의수}}{\text{전체고장의수} - \text{검출불가능한고장의수}}$$

기존의 기법의 경우 고장검출율이 100%가 아닌 모든 회로에서, 우리 기법의 고장검출율이 기존 기법의 고장검출율보다 증가함을 알 수 있다. 이것은 여러 가지 형태로 의사출력에 도달한 고장에 대하여 저장된 고장 상태정보를 이용하여 상태차별화를 시도함으로 얻어진 것이다.

표 2. 기존의 기법과 고장검출율의 비교

circuit name	# of faults				TFC(%)	
	total	detected	untestable	abort	old	new
s386	384	314	70	0	100.0	100.0
s444	474	425	32	17	86.8	96.2
s510	564	564	0	0	99.8	100.0
s820	850	815	35	0	99.9	100.0
s953	1079	1069	10	0	100.0	100.0
s1196	1242	1212	0	30	92.5	97.6
s1488	1486	1446	40	0	100.0	100.0
cse	519	517	2	0	100.0	100.0
sse	368	298	70	0	100.0	100.0
sand	1425	1322	95	8	96.5	99.4
planet	1482	1431	51	0	100.0	100.0
scf	2456	2321	123	12	99.0	99.5
sbc	1985	1922	56	7	92.3	99.6
key	8520	8072	448	0	99.9	100.0
pewd	3483	3483	0	0	100.0	100.0
stage	7227	6668	435	124	94.2	97.5
dsip	6781	6780	1	0	100.0	100.0

표 4에는 기존의 기법에 의해 생성된 패턴의 수, 새로운 기법에 의해 만들어진 패턴의 수, 실제로 고장시뮬레이션이 수행된 패턴의 수, 상태정보를 저장하는 데 쓰여진 메모리의 크기를 정리하였다. 표 4에서 보듯이 대부분의 경우 패턴의 수가 훨씬 작아짐을 볼 수 있고, 고장 시뮬레이션이 수행된 패턴의 수 또한 매우 작다. 그리고 상태정보를 저장하기 위해 소요된 메모리도 매우 작음을 볼 수 있다.

표 3에는 패턴생성 시간을 기존의 기법과 실험적으로 비교하였다. 전체적으로 기존의 기법에 비해 매우 빠른 시간 내에 패턴을 생성해내고 있다. s1196의 경우 cpu 수행 시간이 더 걸리긴 하였으나 이는 고장시뮬레이션에서 얻어진 고장의 정보를 이용하여 상태차별화를 기존의 기법보다 더 많이 시도하였기 때문이다. 그러나 이러한 시도는 기존의 기법보다 더 많은 고장을 검출할 수 있게 하였음을 표 2에서 고장검출율이 약 5%이상 증가한 것으로부터 알 수 있다. 표 3으로부터 DTPG의 수행시간이 s444 및 s1196을 제외한 모든 회로에서 감소함을 알 수 있고, 모든 회로에서 고장시뮬레이션의 수행시간이 감소함을 알 수 있다.

표 3. 기존의 기법과 패턴 생성 시간 비교

circuit name	old			new		
	cpu time(sec.)			cpu time(sec.)		
	DTPG	FSim	total	DTPG	FSim	total
s386	0.6	0.6	1.2	0.41	0.05	0.50
s444	83.1	3.3	86.4	95.8	0.4	96.26
s510	0.7	1.9	2.6	0.32	0.08	0.45
s820	4.1	8.6	12.7	3.3	0.3	3.68
s953	1.5	7.2	8.7	0.6	0.3	1.00
s1196	593	12	605	1425.4	0.6	1428.09
s1488	4	28	32	2.3	0.6	3.05
cse	0.3	1.3	1.6	0.17	0.13	0.32
sse	0.6	0.6	1.2	0.35	0.05	0.46
sand	15	20	35	12.9	0.7	13.82
planet	4	34	37	2.4	0.5	3.04
scf	60	148	208	41.9	2.0	44.37
sbc	265	42	307	113.9	0.9	116.00
key	290	7929	8221	150.1	17.8	169.50
pewd	21	422	444	2.9	0.6	4.96
stage	1598	409	2007	232.9	1.6	248.56
dsip	119	3351	3470	15.3	1.7	22.65

표 4. 생성된 패턴의 수, 실제 시뮬레이션 된 패턴의 수, 상태 정보를 저장에 이용된 메모리 요구량

circuit name	pattern length		simulated patterns	mem. for state info.
	old	new		
s386	285	248	86	9 K
s444	1550	1095	324	566 K
s510	1073	672	109	10 K
s820	1112	892	252	16 K
s953	1111	808	221	18 K
s1196	428	368	223	27 K
s1488	1781	1147	213	15 K
cse	430	423	145	12 K
sse	395	255	81	9 K
sand	754	906	308	19 K
planet	1862	1003	193	14 K
scf	2789	1653	419	30 K
sbc	762	793	330	23 K
key	8867	2483	935	58 K
pewd	2257	236	110	25 K
stage	463	168	145	6 K
dsip	4512	400	152	51 K

Ⅳ. 결 론

본 논문에서는 효과적인 순차회로용 자동 테스트 패턴 생성 알고리즘을 제시하였다. ATPG 과정에서 얻어지는 상태천이 정보와 고장들의 상태정보들을 효율적으로 저장하는 자료구조를 제안하고 이를 효율적으로 이용하는 알고리즘을 제안하였다. 그리고 제안된 기법의 효율성을 실험을 통하여 검증하기 위하여 기존의 기법[4]를 기반으로 구현하고 제안된 기법과 기존의 기법을 비교하였다. 실험 결과 대부분의 경우, 상태 정보를 이용함으로써 패턴생성 시간이 현저하게 줄어듦을 볼 수 있었고, 또한 여러 가지 형태로 의사출력에 전파되어 고장 상태정보로 저장된 고장들에 대해 단지 상태차별화만을 시도함으로써 고장검출율이 증가함을 볼 수 있었다. 또한 테스트패턴의 수가 줄어들어 어느 정도의 테스트패턴 축약 효과가 있었음을 확인할 수 있었다.

참 고 문 헌

[1] Brglez, F., Bryan, D., and Kozminski, K.," Combinational Profiles of Sequential Benchmark Circuits," Proc. IEEE Int'l. Symp. Cir.& Sys., pp. 1929--1934, May 1989.

[2] H. Cho, G.D. Hachtel, and F. Somenzi, "Fast sequential ATPG based on implicit state enumeration." in Proc. Init. Test Conf., pp. 67-74, Oct. 1991.

[3] Hideo Fujiwara, Takeshi Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Transactions on Computers, Volume C-32, Number 12, pp. 1137-1144, December 1983.

[4] A. Ghosh et al., "Test Generation and Verification for Highly Sequential Circuits," IEEE Trans. on CAD, Vol. 10, No. 5, pp652-668, May 1991.

[5] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, Vol. C-30, No. 3, pp215-222, March 1981.

[6] Thomas M. Niermann, "Techniques for Sequential Circuit Automatic Test Generation," Technical Report, UILU-ENG-91-2214 (CRHC -91-8), University of Illinois at Urbana-Champaign, March 1991.

[7] W. Kunz and D. K. Pradhan, "Accelerated Dynamic learning for Test Pattern Generation in Combinational Circuits," IEEE Trans. on CAD, Vol. 12, No. 5, pp684-694, May 1993.

[8] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Res. Dev., Vol. 10, No. 4, pp278-291 ,July 1966.

[9] Michael H. Schulz, Erwin Trischler, Thomas M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Transactions on Computer-Aided Design, Vol. 7, No. 1, January 1988.

[10] T.M. Niermann, Wu-Tung Cheng, J.H. Patel, "PROOFS: a fast, memory-efficient sequential circuit fault simulator," IEEE Tr. on Computer-Aided Design, Vol. 11, pp. 198 -207, Feb. 1992.

[11] Miron Abramovici, Melvin A. Breuer, Author D. Friedman, "Digital Systems Testing and Testable Design," IEEE Press, 1990.

[12] R. Marlett, "An Efficient Test Generation System for Sequential Circuits," 23rd DAC, June 1986, pp. 250-256.

[13] M. Schulz and E. Auth, "ESSENTIAL: An

Efficient Self_Learning Test Pattern Generation Algorithm For Sequential Circuits," 1989 ITC, pp. 28-37.

[14] W. Cheng and T. Chakraborty, "Gentest-An Automatic Test Generation System for Sequential Circuits,: IEEE Computer, pp. 43-49, April 1989.

[15] T. Niermann and J. Patel, "HITEC: A Test Generation Package for Sequential Circuit,: European Conf. on Design Automation 1991, pp. 214-218.

[16] T. Kelsey, K. Saluja, and S. Lee, "An Efficient Algorithm for Sequential Circuit Test generation,: IEEE Trans. on computer, vol. 42, pp. 1361-1371, November 1993.

[17] D. G. Saab, et. al., "CRIS: A Test Cultivation Program for Sequential VLSI Circuits,: Intl. Conf. on CAD, Nov. 1992, pp. 216-219

[18] P. Prinetto, M. Rebaudengo, and M. S. Reorda, : An Automatic Test Pattern Generator for Large Sequential Circuits based on Generation Algorithm: , ITC, 1994, pp. 240-249

[19] E. M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of Simple Genetic Algorithms to Sequential Circuit Test Generation," Proc. European Design Test Conference, 1994, pp. 40-45

[20] E. M. Rudnick, J. H. Patel, G. S. Greenstein and T. M. Niermann, " Sequential Circuit Test Generation in a Genetic Algorithm Framework," Proc. DAC, June 1994, pp. 698-704

[21] I. Pomeranz and S. M. Reddy, "LOCSTEP: A Logic Simulation Based Test Generation Procedure," 25th Fault-tolerant Computing Symp., pp. 110-119, June 1995.

[22] M. S. Hsioa, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal," European Design and Test Conference, pp. 22-28, 1996.

[23] E. M. Rudnick and J. H. Patel, "Combining Deterministic and Genetic Approaches for Sequential Circuit Test Generation ," Proc. 32nd Design Autom. Conf., June 1995, pp. 183-188

[24] X. Lin, I. Pomeranz and S. M. Reddy, "MIX: A Test Generation System for Synchronous Sequential Circuits," Proc. 11th Intl. Conf. on VLSI Design, pp. 456-463, January 1998.

[25] X. Lin, I. Pomeranz and S. M. Reddy, "Techniques for Improving the Efficiency of Sequential Circuit Test Generation," Intl. Conf. on CAD, pp.147-151, November 1999.

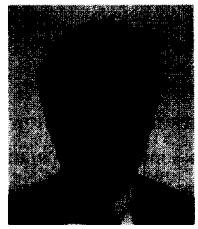
이 재 훈(Jaehoon Lee) 정회원

1996년 2월 : 중앙대학교 제어계측공학과 학사

1999년 2월 : 중앙대학교 제어계측공학과 석사

<주관심 분야> VLSI시스템설계 및 테스트, Computer Network 시스템설계

송 오 영(Ohyoung Song) 정회원



1980년 2월 : 서울대학교 전기공학과 학사

1982년 2월 : 한국과학기술원 전기 및 전자공학과 석사

1992년 2월 : University of Massachusetts at Amherst, 전기 및 컴퓨터공학과 박사

1982년 3월~1985년 5월 : 국방부 기술연구 사무관

1991년 9월~1992년 10월 : Intergraph Corp. Electronics 수석연구원

1992년 1월~1993년 11월 : IBM Microelectronics 수석연구원

1994년 1월~1994년 8월 : 삼성전자 LSI사업부 수석연구원

1994년 9월~현재 : 중앙대학교 전자전기공학부 부교수

<주관심 분야> VLSI시스템설계 및 테스트