논문 2006-43SD-8-5

수정된 의사 무작위 패턴을 이용한 효율적인 로직 내장 자체 테스트에 관한 연구

(A Study on Logic Built-In Self-Test Using Modified Pseudo-random Patterns)

이 정 민*, 장 훈**

(Jeong-Min Lee and Hoon Chang)

요 약

내장 자체 테스트 과정에서 의사 무작위 패턴 생성기에 의해 만들어진 패턴들은 효율적인 고장 검출을 제공하지 못한다. 쓸모없는 패턴들은 테스트 시간을 줄이기 위해 제거하거나 수정을 통해 유용한 패턴으로 바꾸어야한다. 본 논문에서는 LFSR에서 생성하는 의사 무작위 패턴을 수정하고 추가적인 유효 비트 플래그를 사용하여 테스트 길이를 개선하고 높은 고장 검출률을 높이는 방법을 제안하고 있다. 또한 쓸모없는 패턴을 제거하거나 유용한 패턴으로 변경하기위해 reseeding 방법과 수정 비트 플래그 모두 사용한다. 패턴을 수정할 때는 테스트 길이를 줄일 수 있도록 비트의 변화가 가장 적은 수를 선택한다. 본 논문에서는 단일 고착 고장만을 고려하였으며 결정 패턴을 사용하는 seed를 통해 100%의 고장 검출률을 얻을 수 있다.

Abstract

During Built-In Self-Test(BIST), The set of patterns generated by a pseudo-random pattern generator may not provide sufficiently high fault coverage and many patterns were undetected fault. In order to reduce the test time, we can remove useless patterns or change from them to useful patterns. In this paper, we reseed modify the pseudo-random and use an additional bit flag to improve test length and achieve high fault coverage. the fat that a random tset set contains useless patterns, so we present a technique, including both reseeding and bit modifying to remove useless patterns or change from them to useful patterns, and when the patterns change, we choose number of different less bit, leading to very short test length. the technique we present is applicable for single-stuck-at faults. the seeds we use are deterministic so 100% faults coverage can be achieve.

Keywords : LBIST, BIST, reseeding, LFSR

I. 서 론

최근 급속히 발전한 설계와 패키지 기술로 인해 외부 테스트는 더욱 어려워 졌고, 이에 대한 VLSI 테스팅의 유망한 해결책으로 자체 고장 테스트(BIST: Built_In Self Test)가 대두되고 있다. BIST는 칩 안에 구현되어 있는 테스트 로직을 이용하여 시스템의 고장 요소들을 찾아내는 테스트 기술이다. 기존의 ATE(automatic test equipment)에 비해 테스트 패턴을 저장하고 적용하고 결과값을 분석하는데 더 적은 비용을 소요한다. BIST 기술의 주요 구성 요소는 테스트 패턴 생성기(TPG: test pattern generator)와 응답 압축기(response compactor), 신호 분석기이다. 테스트 패턴 생성기에서 패턴을 순차적으로 CUT에 적용하고, 그 결과는 응답 압축기에 의해 신호로 압축되며, 압축된 신호는 고장 없는 참조 값과 비교한다.

LFSR(Linear Feedback Shift Register)과 cellular automata와 같은 의사 무작위 패턴 생성기는 BIST에 사용하는 패턴을 만들기 위해 사용한다. 이러한 의사

* 학생회원, ** 정회원 숭실대학교 컴퓨터학과
(Department of Computing, Soongsil University)
※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어 졌음
접수일자: 2006년5월15일, 수정완료일: 2006년8월2일

무작위 패턴을 이용한 테스트는 회로를 구성하는 비용을 감소 할 수 있고, 쉽게 구현이 가능하기 때문에 많은 기술자들이 연구하고 사용하였다. 하지만 테스트 과정에 몇 가지 어려움이 있으며 고장 검출률이 80% 내외로서 요구를 충족시키지 못하는 단점이 있다. 또한 많은 디지털 회로들은 의사 무작위 패턴 테스팅의 검출률을 제한하는 random-pattern-resistant 고장들을 가지고 있다[1]. 이러한 고장 검출률을 향상시키기 위해 BIST를 이용한 몇몇 기술들이 제안 되었다. 이러한 기술들은 아래와 같이 분류할 수 있다. (1) 테스트 포인트를 추가하여 테스트 받는 회로(CUT: circuit under test)를 수정[1][2]하거나 CUT를 재설계하여 고장 검출률을 향상 시키는 것[3][4], (2)추가 회로를 이용하여 Weighted pseudo-random 패턴을 만들어 r.t.r 고장 검출률을 증가시키는 방법[5], (3)두 가지 단계를 모두 사용하여 테스트하는 Mixed-mode로써 첫 번째 단계는 pseudo-random 패턴을 적용하고, 두 번째 단계에서는 검출하지 못한 고장을 위해 결정적 테스트 패턴을 적용한다[6][7]. 결정적 테스트 패턴은 테스터에 저장되거나 온칩 롬, 또는 reseedng 회로와 같은 하드웨어로 구현한다[7][8].

본 논문에서는 LFSR과 reseeding 회로만을 이용한 기존 연구에 추가적으로 유효 비트 플래그를 사용하여 scan 체인에서 사용하는 캡쳐 모드에서의 쓸모없는 클록의 낭비를 줄이는 방법을 제안한다. 더불어 수정 비트 플래그를 이용하여 LFSR을 통해 나온 패턴을 수정하여 쓸모없는 패턴을 유용한 패턴으로 만들어준다. 이러한 수정비트 플래그를 이용하여 패턴의 수를 줄일 수 있고 전체 테스트 길이를 효과적으로 감소시킬 수 있다. 제안하는 유효 비트 플래그와 수정비트 플래그를 위하여 추가적인 하드웨어 오버헤드가 발생하지만 전체 회로를 기준으로 5% 내외에 불과하다. 하드웨어 오버헤드가 증가하는 반면 테스트 시간을 줄일 수 있을 뿐만 아니라 기존의 방법들에 비해 높은 고장 검출률을 얻을 수 있다.

II. 기존 연구

1. 전통적인 scan 기반의 LBIST

그림 1은 CUT(Circuit under test)에 단일 스캔 체인을 삽입한다고 가정하였을 때 전통적인 스캔 기반의 BIST에서의 reseeding 접근 방법을 보여주고 있다. 조합회로와 스캔 체인 길이와 같은 개수인 n개의 메모리

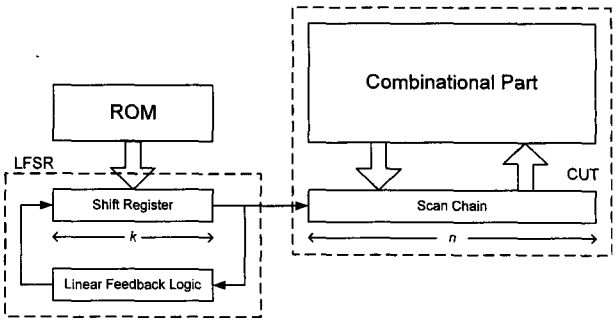
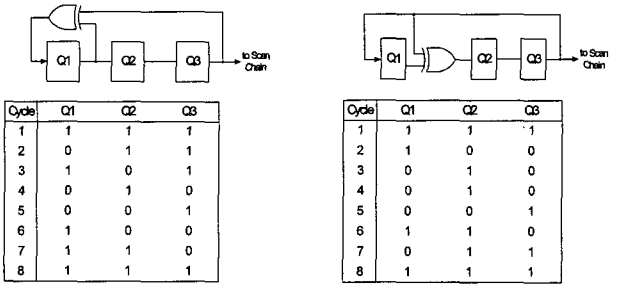


그림 1. 전통적인 스캔 기반의 BIST 구조
Fig. 1. Conventional BIST structure based on scan.

요소(flip-flop)로 구성된 순차회로를 고려한다. TPG는 k개의 플립플롭으로 만든 LFSR과 seed를 저장하고 있는 ROM으로 구성한다. 이때 LFSR의 플립플롭 개수 k는 스캔 체인의 플립플롭의 개수보다 작아야 한다. 전통적인 LBIST에서 LFSR은 이미 정의된 테스트 벡터들에서부터 시작하여 새롭게 생성하는 seed를 주기적으로 로드하여 스캔 체인으로 밀어 넣는다.

2. LFSR을 이용한 패턴 생성

그림 APG(Autonomous Pattern Generator)는 클록이 들어오는 동안 외부입력이 없는 시프트 레지스터와 추가적인 되먹임 연결선으로 구성된 독립 유한 상태머신이다. APG는 XOR 게이트를 사용하여 되먹임 회로를 구성할 때 선형(Linear)이라고 말한다. LFSR은 이러한 APG의 대표적인 방법으로서 외부 되먹임 방식과 내부 되먹임 방식이 있다. 두 방식은 XOR 회로가 외부에 위치하는지 내부에 위치하느냐에 따라 구분 짓는다. LFSR의 기본적인 동작원리는 플립플롭에서 나오는 값들을 XOR 게이트를 통해 되먹임해서 패턴을 생성하는 것이다. 생성된 패턴은 모든 비트 값이 0인 것을 제외한 모든 경우의 수가 발생한다. 따라서 플립플롭의 개수가 k라고 했을 때 생성되는 패턴의 수는 2k-1이다.



Cycle	Q1	Q2	Q3
1	1	1	1
2	0	1	1
3	1	0	1
4	0	1	0
5	0	0	1
6	1	0	0
7	1	1	0
8	1	1	1

Cycle	Q1	Q2	Q3
1	1	1	1
2	1	0	0
3	0	1	0
4	0	1	0
5	0	0	1
6	1	1	0
7	0	1	1
8	1	1	1

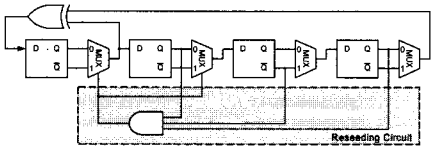
(a) External Feedback LFSR (b) Internal Feedback LFSR

그림 2. x3 + x1 + 1 방정식에 대한 LFSR
Fig. 2. LFSR of x3 + x1 + 1 equation.

LFSR을 사용할 경우 추가적인 하드웨어 오버헤드가 작고 쉽게 구성이 가능하다는 장점이 있다. 하지만 테스트 받는 회로가 커지고 플립플롭의 개수가 증가함에 따라 테스트 시간이 기하급수적으로 증가 하고 고장 검출률이 80%대에 그치는 단점이 발생한다. 이러한 단점을 보완하고 고장 검출률과 테스트 시간을 개선하는 방법들이 소개되었다. 직접적으로 테스트 포인터를 삽입[1][2]하거나 reseeding 회로를 이용하여 CUT (Circuit under test)를 수정하는 방법, 고장 검출 가능성을 개선하기 위해 가중치 의사 결정적 테스트 패턴을 이용하는 방법, 그리고 이러한 두 방법을 혼용하여 사용하는 혼합 모드 테스팅이 대표적이다.

3. reseeding 회로를 이용한 LBIST 구조

그림 3은 LFSR의 구조와 그 외부에 위치하는 reseeding 회로를 보여주고 있다. 기존의 LFSR과 다르게 플립플롭의 출력부분에 멀티플렉서를 이용하여 출력값 또는 그 역의 값을 선택할 수 있도록 구성되어 있다. reseeding 회로의 출력 값은 LFSR의 어떤 스테이지에서 역수를 취하기 위해 멀티플렉서의 선택라인을 활성화 한다. 여기에서 6번째 사이클 이후에 reseeding을 하기 원한다고 가정하자. reseeding 회로는 c6에서의 출력값을 이용하여 AND 게이트의 입력으로 사용한다. AND 게이트의 출력값은 다시 다음 상태값을 결정하기 위해 멀티플렉서의 선택선으로 들어가게 된다. 멀티플렉서의 선택선은 c6의 상태값과 원하는 seed 바로 전 상태값을 이용하여 계산한다. 이렇게 하는 이유는 원하는 seed 값을 다음 클록에 로드하기 위해서다. c6의 값이 0101이고 원하는 seed의 값인 c12의 바로 앞의 상태인 c11의 값이 1001이므로 두 값을 XOR하게 되면 1100 값을 얻을 수 있다. 1100 값이 바로 멀티플렉서의 선택신호 값이 된다.그림 3의 표에서 S1S2S3S4는 각 멀티



Cycle	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$
0	1 0 0 0	0 0 0 0
1	1 1 0 0	0 0 0 0
2	1 1 1 0	0 0 0 0
3	1 1 1 1	0 0 0 0
4	0 1 1 1	0 0 0 0
5	1 0 1 1	0 0 0 0
7	1 0 1 0	d d d d

Cycle	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$
8	1 1 0 1	d d d d
9	0 1 1 0	d d d d
10	0 0 1 1	d d d d
11	1 0 0 1	0 0 0 0
13	0 0 1 0	0 0 0 0
14	0 0 0 1	0 0 0 0
15	0 0 0 0	d d d d

Reseeding 지점 = Cycle 6 = 0101
Seed = 0100 = Cycle 12
Mux 선택선 = (C6) XOR (C11)
= 0101 XOR 1001
= 1100
=> Q1 과 Q2의 선택선이 활성화

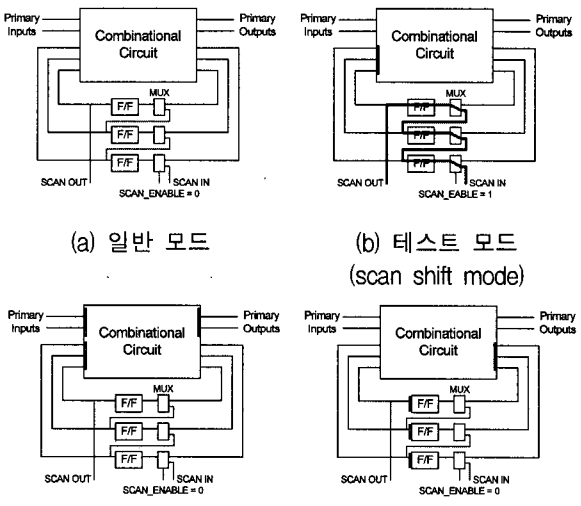
그림 3. reseeding 회로 예제
Fig. 3. Example of reseeding circuit.

플렉서의 선택선 값을 나타낸다. Reseeding 지점 인 c6의 이전까지는 선택선 값이 모두 0000이 되고 c6에서 1100으로 바뀌는 것을 확인 할 수 있다. 원하는 seed 값의 바로 앞 상태인 c11 이전과 c6 사이의 선택선 값은 어떠한 값이 들어와도 상관하지 않는다. 의미 있는 부분은 오로지 reseeding 지점과 원하는 seed가 위치하는 지점의 선택선 값뿐이다.

III. 추가적인 유효 비트 플래그

1. 스캔 테스트 동작 과정

그림 4는 APG에서 생성되어 나온 패턴을 이용하여 스캔 테스트를 하는 동작 과정을 보여주고 있다. (a)는 일반 모드로서 SCAN_ENABLE 값이 0으로 설정되어 있어서 일반적인 순차회로로서 동작한다. (b)에서 SCAN_ENABLE 값이 1로 설정되면서 테스트 모드로 진입하게 되고 SCAN IN을 통해 패턴이 삽입된다. 모든 플립플롭에 테스트 벡터가 삽입이 되고 나면 (c)에서와 같이 Primary Input과 테스트 벡터를 적용하여 조합회로를 동작하여 Primary Output과 플립플롭으로 들어갈 값을 출력한다. (d)는 캡쳐 모드로서 조합회로를 통해 나와 플립플롭으로 들어가는 값들을 플립플롭의 출력으로 보내 주는 과정으로서 추가적인 한 클록이 소요가 된다. 이 모든 과정이 끝나면 SCAN_ ENABLE을 1로 설정하고 플립플롭의 값을 SCAN OUT으로 내보내 준다. 여기서 SCAN OUT을 통해 나온 값을 확인하여 고장 유무를 판단 할 수 있다. 이러한 스캔 테스트



(a) 일반 모드
(b) 테스트 모드 (scan shift mode)
(c) PI과 테스트 벡터 적용
(d) capture mode

그림 4. 스캔 테스트 동작 과정
Fig. 4. Working process of scan test.

과정에 착안하여 본 논문에서는 SCAN_ENABLE을 직접 제어하는 추가 유효 비트 플래그를 사용하여 유용한 패턴과 그렇지 못한 패턴을 구분하여 테스트 유무를 결정하여 캡쳐 모드에서 불필요하게 소요되는 클록을 최소화 한다.

2. 추가적인 유효 비트 플래그

기존의 reseeding 회로를 이용한 LBIST 구조는 패턴의 수를 줄이는 데 매우 효과적이다. 하지만 패턴 생성의 관점에서만 고려된 생각으로서 실제 스캔 체인에 패턴을 삽입하고 테스트 하는 과정에서 소요되는 클록에 대해 고려하지 않고 있다. 앞 절에서 본 것처럼 패턴이 생성되어 스캔에 삽입한 뒤 테스트 결과를 확인하기 위해 캡쳐 모드에서 추가적인 한 클록이 소요된다. 스캔 체인에 삽입되어있는 테스트 패턴이 유용한 패턴인지 그렇지 않은 패턴인지 모를 경우 모든 패턴에 대해 캡쳐 동작을 수행하게 될 것이고 그에 따라 쓸모없는 클록을 낭비하게 된다. 이러한 클록의 낭비를 방지하기 위해 본 논문에서는 패턴의 유용성에 대한 추가적인 플래그 세트를 사용하여 단점을 보완하려고 한다.

가. 추가적인 유효 비트 플래그

그림 5는 추가적인 유효 플래그를 사용한 예를 보여주고 있다. 4개의 플립플롭으로 이루어진 LFSR과 앞장에서 본 reseeding 회로를 기본으로 유용한 패턴이 사이클 0, 2, 12에서 나타난다고 가정하도록 한다. 사이클 0의 패턴이 4비트 스캔 체인에 삽입되는 시점은 그림에서 보는 바와 같이 사이클 4이다. 따라서 4개의 사이클 동안 스캔 체인에 들어 있는 패턴은 의미가 없다. 스캔 체인에 의미 있는 패턴이 들어 있다는 표시로 유효 플래그가 1로 설정이 되어 있으며 그 값을 참조하여 SCAN_ENABLE 값을 0으로 설정해 준다. 캡쳐된 결과 값을 확인 하고 다시 SCAN_ENABLE 값을 1로 설정해 주고 스캔 체인으로 패턴을 삽입하게 된다. 두 번째 유효한 패턴으로 사이클 2의 1110이 나타난다. 이 패턴은 이전의 유효한 패턴에서 한 사이클이 지난다음 나타나는 것을 알 수 있다. 기존의 방식대로라면 유효한 패턴 사이의 한 사이클에서도 SCAN_ENABLE 값을 0으로 설정하여 캡쳐 모드를 가동할 것이다. 하지만 본 논문에서 제안하는 유효 플래그를 이용하여 직접적으로 SCAN_ ENABLE 값을 제어하게 되면 결과값을 캡쳐하기 위해 불필요하게 들어가는 사이클의 낭비를 막을 수 있다. 기존의 reseeding 방법을 이용하여 여러

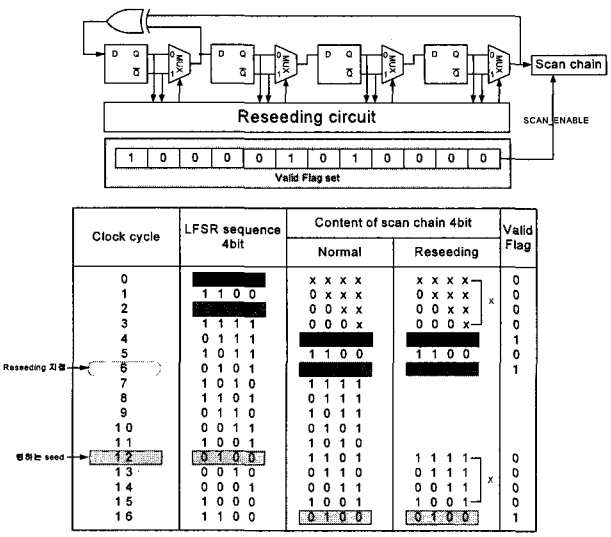
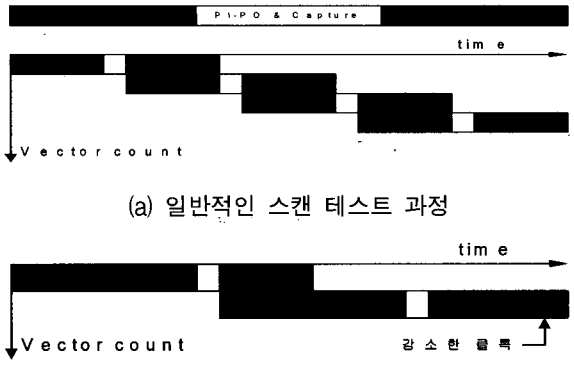


그림 5. 추가적인 유효 플래그 예제
Fig. 5. Example of additional valid flag.

개의 패턴을 건너뛰었을 때 추가적인 유효 비트 플래그의 효율이 더 높게 나타난다.사이클 6에서 12로 패턴들을 건너뛴다고 했을 때, 사이클 12에서부터 15까지 4개의 사이클 동안 의미 없는 패턴들이 스캔에 삽입된다. 이것은 처음 사이클이 시작하면서 스캔에 패턴을 입력할 때 소요되는 사이클의 수와 같다. 따라서 reseeding을 많이 하면 할수록 유효 비트 플래그에 대한 필요성이 더욱 강조되는 것이다. 만약 기존의 방식대로 의미 없는 패턴이 스캔에 삽입되어 있을 때도 캡쳐 동작을 수행 한다면 사이클의 낭비가 그만큼 많아진다. 따라서 본 논문에서 제안하는 유효 비트 플래그를 사용하면 reseeding을 통해 패턴을 줄일 뿐만 아니라 줄어든 패턴만큼 소요되는 사이클의 수를 감소할 수 있다.

나. 테스트 처리과정 비교

그림 6은 일반적인 스캔 테스트 과정과 제안하는 추가적인 유효 비트 플래그를 사용할 때를 비교해 놓았다. (a)는 일반적인 스캔일 때 소요 되는 클록을 나타내고 있다. 세부분으로 나누어 처음 부분은 테스트 패턴이 스캔으로 들어가는 클록을 나타내고, 다음 부분은 primary input과 primary output을 이용하여 결과값을 얻기 위한 캡쳐에 사용하는 클록을 나타낸다. 마지막 부분은 스캔 아웃에 쓰이는 클록이다. 이전 패턴이 스캔 아웃이 되면서 그 다음 패턴이 스캔으로 들어오게 되므로 그림에서 보는 것과 같이 겹쳐서 나타낼 수 있다. 항상 캡쳐를 하기 위해 한 클록이 필요한 것을 알 수 있다. 하지만 추가적인 유효 비트 플래그를 사용할



(a) 일반적인 스캔 테스트 과정

(b) 추가적인 유효 비트 플래그를 사용한 테스트 과정

그림 6. 테스트 처리 과정 비교
Fig. 6. Comparison working process of scan test.

경우 (b)에서 보는 것과 같이 유용한 패턴이 아닐 경우 캡쳐모드를 거치지 않도하여 클록의 낭비를 줄여준다. reseeding 하는 횟수가 늘어나고 테스트 패턴의 개수가 늘어나게 되면 더 많은 클록의 감소를 기대할 수 있다.

IV. 수정 비트 플래그

앞 장에서 보았던 reseeding 회로를 이용한 방법은 LFSR에서 생성된 패턴들 중 유용한 패턴이 멀리 떨어져 있을 때 많이 사용한다. 하지만 유용한 패턴이 하나의 패턴이나 두개의 패턴 간격으로 나온다고 했을 때 reseeding 회로를 이용하는 것은 회로의 복잡도와 면적 오버헤드를 기하급수적으로 상승시키는 요인이 된다. 본 논문에서는 이러한 문제점을 해결하고 패턴의 수를 줄이기 위해 수정 비트 플래그 방법을 제안하고 있다.

1. 수정 비트 플래그 구조

그림 7은 제안하고 있는 수정 비트 플래그를 포함하는 LFSR 구조를 보여 주고 있다. 회로를 위한 테스터에 메모리를 사용하여 수정할 비트를 저장한다. 수정 비트 플래그는 일정한 패턴이 LFSR에 저장 되었을 때

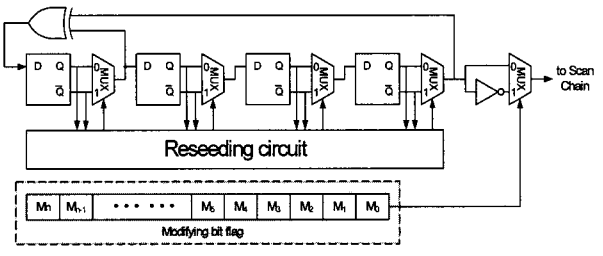


그림 7. reseeding 회로와 같이 구성한 수정비트 구조
Fig. 7. Structure of modify bit with reseeding circuit.

참조를 시작하여 패턴의 수정이 끝나면 일반적인 LFSR 구조로 돌아간다. 기존의 Reseeding 회로에 두개의 입력값과 하나의 선택신호를 가지는 멀티플렉서를 추가하여 구성한다. 멀티플렉서의 두개의 입력값은 LFSR을 통해 나오는 패턴의 일반값과 그 값을 역으로 취한 값이며 선택신호는 수정 비트 플래그의 값을 참조한다. 수정 비트가 1로 설정이 되면 LFSR의 값을 수정하여 스캔으로 밀어 넣어 준다.

2. 수정 비트 플래그 알고리즘

수정 비트 플래그를 생성하기 위해서는 고장 시뮬레이션을 통해 결정된 패턴(deterministic pattern)을 파악하고, LFSR에서 생성되는 패턴들과 비교한다. reseeding을 하지 않는 범위 내에서 수정 비트 플래그를 사용했을 경우 효율성이 가장 높은 패턴을 찾는다. LFSR이 어떤 시작값을 가지고 있을 때 이 패턴이 나오는지 확인한다. 패턴이 수정이 되었을 때 다음 패턴에 미치는 영향도 고려해야 하며 reseeding과 중복되는 것을 확인해야 한다. 본 논문에서 제안하는 수정 비트 생성 알고리즘의 의사 코드는 다음과 같다.

```
제안하는 수정 비트 생성 알고리즘
1:  n = 스캔 체인 길이
2:  k = LFSR 플립플롭 개수
3:  lp = LFSR에 의해 만들어지는 패턴 집합
4:  dp = deterministic patterns
5:  p = 각 알고리즘에 적용되는 패턴
6:  m = 매핑 로직
7:  Fault simulation
8:  Read deterministic pattern
9:  LFSR_pattern_generator(n, k)
10: while (no more reseeding)
11:  compare_pattern(lp, dp)
12:  reseeding_algorithm_genetator(n, k, pi, pj)
13:  apply_reseeding_algorithm(pi, pj)
14:  endwhile
15:  while (not reduce fault coverage)
16:  detect_useless_pattern(lp)
17:  modify_bit(lp, b1 ... bi)
18:  redundant_check(pi)
19:  mapping_logic_generator(pi, mi)
20: endwhile
```

3. 수정 비트 플래그 예제

그림 8은 수정 비트 플래그를 이용하여 스캔에 삽입되는 패턴을 바꾸는 과정을 보여주고 있다. LFSR의 시작상태에 따라 스캔에 삽입되는 패턴은 그림에서 보는 바와 같다. 다섯 개의 시작 상태 중에서 1001로 시작하여 스캔에 삽입이 될 경우 그 패턴은 쓸모없는 패턴으

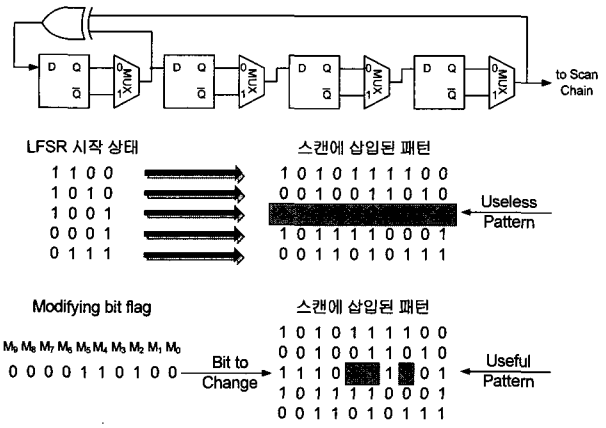


그림 8. 수정 비트 예제
Fig. 8. Example of modify bit.

로 처리되어 클록의 낭비를 가지고 온다. 이러한 단점을 보완하기 위해 기존의 LFSR과 reseeding 구조를 변경하지 않고 추가적으로 수정 비트 플래그를 사용함으로서 패턴의 수를 줄일 수 있다. 그림에서 보는 것과 같이 스캔에 패턴이 입력되기 전에 수정 비트 플래그를 참조하여 1로 설정된 플래그를 만날 경우 LFSR에서 나오는 기존의 패턴 값의 역수를 스캔에 삽입한다. 1001로 시작하여 기존의 방법대로 LFSR을 통해 패턴을 삽입할 경우 1110001001이 스캔에 삽입된다. 이 패턴은 고장 시뮬레이션의 결과와 비교했을 때 쓸모없는 패턴으로 판단된 것이다. 이 패턴을 재사용하기 위해 수정 비트로 0000110100을 사용한다. 그림8의 아래 그림에서 보는 것과 같이 기존의 패턴은 1110111101로 변경되며 유용한 패턴으로서 테스트에 사용이 가능해 진다. 이렇게 수정 비트 플래그를 사용하게 되면 기존의 reseeding 만을 사용하는 방법 보다 더 적은 수의 패턴을 사용하여 높은 고장 검출률을 얻을 수 있다.

V. 제안하는 LBIST 구조

그림 9에서 본 논문에서 제안하고 있는 수정 비트 플래그와 유효 비트 플래그를 사용한 LBSIT 구조를 보여주고 있다. 테스트 받는 회로를 중심으로 LBIST Controller가 구성되어 있는 것을 확인 할 수 있다.LBIST Controller는 크게 Reseeding 회로를 포함하는 LFSR과 제어 신호 생성기(Control signal Generator), 수정 비트 플래그(Modifying bit flag), 유효 비트 플래그(Valid flag set)로 구성한다. 제어 신호 생성기는 LBIST Controller 전체를 제어하는 신호를 생성해 주는 모듈로서 유효 비트 플래그의 값과 수정 비트

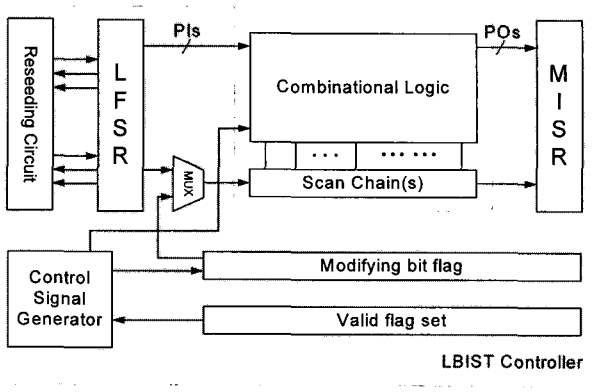


그림 9. 제안하는 시스템 관점에서의 LBIST 구조
Fig. 9. The system view of proposed LBIST structure.

플래그 값을 참조 하기 위한 내장된 매핑 알고리즘에 의해 동작한다. 테스트가 시작되면 Reseeding 회로를 포함하는 LFSR에서 패턴을 생성하여 스캔 체인으로 값을 밀어 넣어 준다. 이 때 쓸모없는 패턴이 나올 경우 앞장에서 설명한 것과 같이 수정 비트 플래그를 참조하여 패턴을 수정 한 후 스캔 체인으로 보내준다. 여기서 수정 비트 플래그의 값은 LFSR과 스캔 체인 사이에 있는 멀티플렉서의 선택선으로 사용한다. 일련의 스캔 테스트 과정 동안 제어 신호 생성기(Control Signal Generator)는 유효 비트 플래그를 참조하고 있으며 그 값에 따라 Scan_Enable 값을 결정한다. 유용한 패턴이 스캔 체인에 들어가 있을 경우 패턴이 모두 삽입된 시점에서 Scan_Enable를 0으로 설정하고 그 값을 확인한다. 반대의 경우 패턴에 대한 결과를 확인하지 않고 계속해서 스캔 체인으로 패턴을 밀어 넣어준다. 본 장에서 회로를 테스트하기 위한 전체적인 구조를 살펴보았다. 다음 장에서는 본 논문에서 제안하는 유효 비트 플래그와 수정비트 플래그를 사용하였을 경우 시뮬레이션 결과를 알아보도록 하겠다.

VI. 실험결과

본 장에서는 본 논문에서 제안하는 유효 비트 플래그와 수정 비트 플래그를 이용한 고장 테스트의 우수성을 입증하기 위해 ISCAS89 벤치마크 회로를 대상으로 실험 결과를 기술한다. 결정적 테스트 패턴을 생성하는 ATPG와 고장 시뮬레이션을 위하여 시놉시스의 tetraMax를 사용하였으며 하드웨어 오버헤드를 측정하기 위하여 design analyzer를 이용하였다. 제안하는 방법을 이용한 패턴을 생성하기 위해 리눅스 레드헷 9에서 gcc를 이용하였다. 표 1은 실험에 사용한 ISCAS 89

표 1. ISCAS 89 벤치마크 회로
Table 1. ISCAS 89 benchmark circuit.

Benchmark Circuit	PIs	POs	Scan Chain size	LFSR size	Faults
s1423	17	5	74	50	2892
s1488	8	19	6	6	2532
s1494	8	19	6	6	2582
s5378	35	49	179	61	6468
s9234	36	39	211	80	8612
s13207	62	152	638	45	16550
s15850	77	150	534	150	18446
s35932	35	320	1728	60	43370
s38417	28	106	1636	200	52164
s38584	38	304	1426	200	62168

표 2. test length와 고장 검출률 비교
Table 2. Comparison of test length and fault coverage.

Benchmark Circuit	test length			고장검출률(%)		
	랜덤 패턴	[8]	proposed	랜덤 패턴	[8]	proposed
s1423	10000	8160	7496	99.27	100	100
s1488	10000	4096	3654	100	100	100
s1494	10000	4096	3686	99.96	100	100
s5378	30000	27648	24883	99.81	100	100
s9234	80000	76800	69888	95.94	100	100
s13207	60000	60416	53166	99.41	100	100
s15850	50000	45056	41000	95.71	100	100
s35932	10000	5120	4710	100	100	100
s38417	90000	89088	78397	97	100	100
s38584	80000	76800	68352	99.61	100	100

벤치마크 회로에 대한 기본적인 정보를 보여주고 있다.

표 2는 ISCAS89 벤치마크 회로를 대상으로 실험한 결과로써 test length 와 고장 검출률을 보여주고 있다. 랜덤 패턴을 이용한 방법과 [8]에서 reseeding만을 이용하였을 때 test length를 비교했을 때 제안하는 방법이 10% 내외로 줄어든 것을 확인 할 수 있다. 고장 검출률의 경우 랜덤 패턴이 test length 가 더 크지만 100%의 고장 검출률을 얻지 못하지만, 본 논문에서 제안하는 방법의 경우 짧은 test length를 이용하여 모든 회로에서 100%의 고장 검출률을 얻을 수 있다. 수정 비트를 이용하여 reseeding 하고 난 후 일정한 수의 비트만을 수정해 줌으로써 test length를 줄이면서 높은 고장 검출률을 얻을 수 있다.

본 논문에서 제안하는 방법을 통해 test length를 줄이는 것이 가능하고 또한 높은 고장 검출률을 얻을 수 있는 것을 확인 하였다. 하지만 이러한 회로들을 구현하기 위해 추가적으로 하드웨어 오버헤드가 생기게 된다. 표 3에서는 이러한 하드웨어 오버헤드를 벤치마크 회로 대한 비율로 나타내고 있다. s1423 회로를 제외하고 모든 회로에서 5% 이내의 하드웨어 오버헤드를 가지는 것을 확인 할 수 있다. 또한 표 3에서는 Ⅲ장에서 제안하고 있는 추가적인 유효비트 플래그를 사용하였을 경우 테스트 시간에 대한 기존의 방법들과 비교한 것을 보여주고 있다. 랜덤 패턴의 경우 패턴의 삽입이 불규칙적으로 발생하기 때문에 테스트 시간이 매우 크게 나타나고 있다. reseeding 만을 이용한 [8]의 테스트 시간도 랜덤 패턴에 비해 크게 줄었지만 불필요한 클록의 소요로 회로가 커짐에 따라 테스트 시간도 매우 큰 폭으로 증가한다. 반면 본 논문에서 제안하는 추가적인 유효 비트 플래그를 사용할 경우 불필요한 캡쳐 클록을 삭제함으로서 테스트 시간이 크게 감소하는 것을 확인 할 수 있다.

표 3. 하드웨어 오버헤드 및 테스트 시간 비교
Table 3. Comparison of hardware overhead and test time.

Benchmark Circuit	area	하드웨어 오버헤드	test time		
			랜덤 패턴	[8]	proposed
s1423	4,531	10%	0.37	0.18	0.03
s1488	3,555	3.5%	1.29	0.21	0.04
s1494	3,563	3.5%	1.12	0.20	0.04
s5378	14,376	5.8%	3.11	0.49	0.22
s9234	25,840	4.7%	17.20	0.90	0.31
s13207	44,255	3.0%	22.94	1.37	0.76
s15850	48,494	4.7%	22.94	2.30	0.88
s35932	106,198	3.0%	0.54	0.67	0.36
s38417	120,180	3.4%	125.62	9.09	4.85
s38584	115,855	3.5%	107.00	6.26	3.57

Ⅳ. 결 론

내장형 자체 테스트 기법은 점점 더 거대해지고 있는 회로를 적은 비용과 동작 속도로 테스트하기 위한 효과적인 방법으로 주목받고 있는 방법이다. BIST의 효율성은 테스트 시간과 테스트하드웨어, 고장검출률로 나타낼 수 있다. 많은 BIST 구조가 의사 무작위 테스트, 재초기화 방법, 가중치 의사 무작위 테스트, 결정 테스트 방법 등에 관해 이러한 parameter들의 절충점을 찾기 위해 연구되어 왔다.

본 논문은 의사 무작위 테스트와 결정 테스트를 위해 효율적인 구조와 알고리즘을 제시하였다. SCAN_ENABLE을 직접 제어하는 추가 유효 비트 플래그를 사용하여 유용한 패턴과 그렇지 못한 패턴을 구분하여 테스트 유무를 결정하여 캡쳐 모드에서 불필요하게 소요되는 클록을 최소화 하였다. 또한 수정 비트 플래그 알고리즘을 통해 쓸모없는 패턴을 수정하여 패턴의 수를 줄임으로서 효율적인 테스트 결과를 얻을 수 있었다.

실험은 ISCAS 89 벤치마크 회로를 사용하였으며, 고장 검출률과 테스트 길이, 추가적인 하드웨어 오버헤드, 테스트 시간에 대해 확인하였다. 테스트 길이를 줄이면서 기존의 연구와 같은 높은 고장 검출률을 얻을 수 있었으며, 불필요한 클록 사용을 없앰으로서 테스트 시간을 줄일 수 있었다. 또한 회로가 클수록 하드웨어 오버헤드의 비율이 줄어드는 것을 확인 하였다.

참 고 문 헌

[1] Eichelberger, E. B., and E. Lindbloom, "Random Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," IBM Jounal of Research and Development, Vol. 27, No.3, pp. 265-272, May. 1983.

[2] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. of VLSI Test Symposium, pp. 2-8, 1996.

[3] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. of VLSI Test Symposium, pp. 2-8, 1996.

[4] Chiang, C.-H., and S.K. Gupta, "Random Pattern Testable Logic Synthesis," Proc. of International Conference on Computer-Aided Design (ICCAD), pp. 125-128, 1994.

[5] Eichelberger, E.B., and E. Lindbloom, F. Motica, and J. Waicukauski, "Weighted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 89.

[6] Hellebrand, S., B. Reeb, S. Tarnick, and H.-J.Wunderlich, "Pattern Generation for a Deterministic BIST Scheme," Proc. of International Conference on Computer-Aided Design(ICCSAD), pp.88-94, 1995.

[7] Touba, N. A. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.

[8] Al-Yamani A., and E.J. McCluskey, "Built-In Reseeding for Serial BIST," VLSI Test Symposium, Apr., 03.

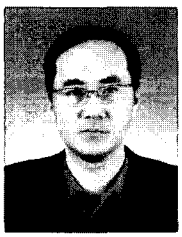
저 자 소 개

이 정 민(학생회원)

2004년 숭실대학교 컴퓨터학부 학사 졸업.

2006년 숭실대학교 컴퓨터학과 석사 졸업.

<주관심분야 : VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD>



장 훈(정회원)

1987년 서울대학교 공대 전자공학과 학사 졸업.

1989년 서울대학교 공대 전자공학과 석사 졸업.

1993년 University of Texas at Austin 졸업.

1991년 IBM Inc. Senior Member of Technical Staff.

1993년 Motorola Inc. Senior Member of Technical Staff.

1994년~현재 숭실대학교 컴퓨터학부 부교수.

<주관심분야 : 통신, 컴퓨터, 신호처리, 반도체>