論文 97-22-7-2

내장된 메모리를 위한 향상된 March 테스트 알고리듬의 설계 및 구현

正會員 박 강 민*, 장 훈**, 양 승 민**

Design and Implementation of Improved March Test Algorithm for Embedded Memories

Gang-Min Park*, Hoon Chang**, Seung-Min Yang** *Regular Members*

※본 연구는 서울대학교 반도체공동연구소의 교육부 반도체분야 학술연구조성비(과제번호 ISRC(95-E-2012))에 의해 수행되었습니다.

요 약

본 논문에서는 내장된 메모리를 위한 효율적인 March 테스트 알고리듬과 BIST 구조를 제안하였다. 제안된 테스트 알고리듬은 고장 모델에 따른 고착 고장, 천이 고장, 결합 고장을 완전히 검출할 수 있다. 또한, 제안된 알고리듬은 기존의 March 테스트에서는 검출할 수 없었던 이웃패턴 감응 고장도 워드 단위의 메모리 테스트에 사용되는 배경 데이터를 이용하여 검출할 수 있다.

ABSTRACT

In this work, an efficient test algorithm and BIST architecture for embedded memories are presented. The proposed test algorithm can fully detect stuck-at fault, transition fault, coupling fault. Moreover, the proposed test algorithm can detect neighborhood pattern sensitive fault which could not be detected in previous march test algorithms. The proposed test algorithm performs testing for neighborhood pattern sensitive fault using background data which has been used word-oriented memory testing.

I. 서 론

고집적화된 칩의 테스트에 있어서 가장 어려운 부분중에 하나로 여겨지는 것은 내장된 메모리(embedded memory)의 테스트이다. 내장된 메모리의 테스트는 기존의 조합 회로에 대한 고착 고장(stuck-at-fault) 모델만으로는 테스트하기가 어려우며, 입·출력 신호

*숭실대학교 대학원 전자계산학과
**숭실대학교 컴퓨터학부
論文番號:97068-0221
接受日字:1997年 2月 21日

를 칩의 외부에서 직접적으로 제어하기가 어렵기 때문에 기존 방식으로는 효율적인 테스트가 어렵다[1, 2]. 이러한 문제들을 해결하기 위해 가장 널리 사용되는 방법은 내장된 자체검사 기법(BIST:Built-In Self Test)을 사용하는 것이다[3].

본 논문에서는 내장된 메모리의 테스트에 널리 사용되고 있는 기존의 March 알고리듬에서 테스트가 가능한 고장뿐만 아니라 이웃패턴 감응 고장(neighborhood pattern sensitive fault)까지 테스트할 수 있는 테스트 알고리듬을 제안하였다.

Ⅱ. 메모리의 고장 모델

메모리 테스트의 목적은 주로 고장의 유형이나 위치를 파악하기 보다는, 단순히 고장의 발생유무를 파악하는 것이다. 그러므로 메모리 테스트에서는 먼저 메모리의 구조를 기능 모델(functional model)로 단순화시킨다[6]. 테스트를 위한 메모리의 기능적 모델은 그림 1과 같다.

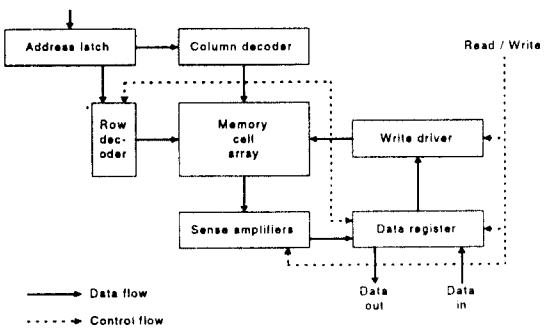


그림 1. 메모리의 기능적 모델

이러한 기능적 모델에서 제안되는 고장들(functional faults)은 크게 다음의 4가지로 분류된다[4].

2.1 고착, 천이, 결합 고장 모델

가. 고착 고장 모델

메모리 셀의 논리값이 0이나 1로 고정되어 그 논리값이 변하지 않는 고장이다.

나. 천이 고장 모델

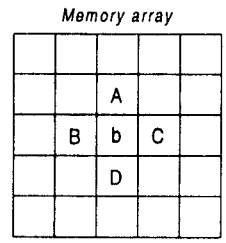
메모리 셀의 논리값이 0에서 1(상향 천이), 또는 1에서 0(하향 천이)으로의 천이(transition)가 되지 않는 고장이다.

다. 결합 고장 모델

특정 메모리 셀에서의 논리값 변화로, 이 셀과 연관된 다른 메모리 셀의 값이 변하는 고장이다.

2.2 이웃패턴 감응 고장모델

이웃패턴 감응 고장 모델에서의 셀 배치는 그림 2와 같다. 이 모델에서는 영향을 받는 셀 b를 기준 셀(base cell), 영향을 주는 주변 셀 A, B, C, D 및 b를 이웃 셀(neighborhood cell), 이웃 셀에서 기준 셀을 뺀 셀들을 참이웃 셀(deleted neighborhood cell)이라 한다.



b : base cell
b + A, B, C, D : neighborhood cells
A, B, C, D : deleted neighborhood cells

그림 2. 이웃패턴 감응 고장 모델의 셀 배치

이웃패턴 감응 고장은 다음의 세 종류로 나눌 수 있다[1, 5, 7].

- 정적(static) 이웃패턴 감응 고장
- 수동(passive) 이웃패턴 감응 고장
- 능동(active) 이웃패턴 감응 고장

Ⅲ. 기존의 메모리 테스트 알고리듬

이 장에서는 메모리 고장 모델의 분류에 따른 기존의 테스트 알고리즘에 대해 설명한다.

3.1 고착, 천이 및 결합 고장 테스트알고리듬

비트 단위의 메모리에서 이들 고장을 검출하는데 널리 사용되는 테스트 알고리듬은 9N(N은 전체 메모리 셀의 갯수)의 시간복잡도를 가지는 9N March 테스트 알고리듬으로, 그림 3에서 이를 보여준다[4, 6, 8].

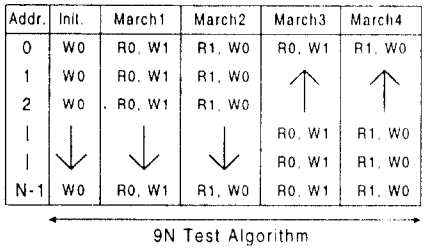


그림 3. 9N March 테스트 알고리즘

이 알고리듬에서 기호의 정의는 다음과 같다.

- W0 : = 메모리 셀에 0을 쓴다.
- W1 : = 메모리 셀에 1을 쓴다.
- R0 : = 메모리 셀에서 0을 읽는다.
- R1 : = 메모리 셀에서 1을 읽는다.

3.2 워드단위의 메모리 테스트

워드단위의 메모리는 워드 단위로 읽기와 쓰기가 일어나게 되므로, 하나의 워드 내에서 발생될 수 있는 fault masking 문제를 고려해야만 한다. 이를 위해 배경 데이터(background data)라고 불리는 비트 패턴들이 필요하다[1, 2]. 워드의 크기를 8비트라고 가정할 때 본 논문에서 사용될 배경 데이터는 그림 4와 같다.

	W(0)	W(1)
P0	00000000	11111111
P1	01010101	10101010
P2	10101010	01010101
P3	00110011	11001100
P4	11001100	00110011
P5	00001111	11110000
P6	11110000	00001111
P7	11111111	00000000

그림 4. 배경 데이터

워드 단위로 확장된 9N March 테스트 알고리듬을 수행하기 위해서는 그림 3에서 정의된 W0, R0, W1, R1 대신에 W(0), W(1), R(0), R(1)이 사용된다.

- W(0) : = 배경 데이터를 쓴다.
- W(1) : = 역전된(inverted) 배경 데이터를 쓴다.
- R(0) : = 배경 데이터를 읽는다.
- R(1) : = 역전된 배경 데이터를 읽는다.

3.3 이웃패턴 감응 고장 검출을 위한 테스트

가. 정적 이웃패턴 감응 고장

메모리 배열의 기준 셀에 대해서 이웃패턴 감응 고장을 테스트하고자 할 때에는, 5개의 이웃 셀들(b, A, B, C, D)의 모든 패턴들의 조합인 32개의 패턴들을 생성해야만 하고 이것은 그림 5에 나타나 있다. 여기서는 b가 0인 경우(16개의 패턴)만을 표시한다.

b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

그림 5. 정적 이웃패턴 감응 고장 테스트 패턴

나. 수동 이웃패턴 감응 고장

수동 이웃패턴 감응 고장의 테스트 또한 32개의 테스트 패턴들을 읽을 수 있어야만 하고 이것은 그림 6에 나타나 있다. 여기서는 b가 상승 천이('↑')인 경우(16개의 패턴)만을 표시한다.

b	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

그림 6. 수동 이웃패턴 감응 고장 테스트 패턴

다. 능동 이웃 패턴 감응 고장

능동 이웃 패턴 감응 고장의 테스트도 그림 7의 비

트 패턴들이 각각 쓰여지고 읽혀져야만 한다. 여기서는 b가 0이고 A에서 상승천이가 일어나는 경우(16개의 패턴)만을 표시한다.

b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	↑	↑	↑	↑	↑	↑	↑	↑	↓	↓	↓	↓	↓	↓	↓	↓
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

그림 7. 능동 이웃 패턴 감응 고장 테스트 패턴

Ⅳ. 개선된 메모리 테스트 알고리듬

제안된 알고리듬은 기존의 9N March 알고리듬을 크게 두 단계에 걸쳐서 개선시킨다. 첫 번째는 워드 단위의 March 테스트에서 사용되는 배경 데이터를 이용하는 방법이고, 다음은 하나의 워드에 대한 읽기와 쓰기를 분할하는 방법이다.

4.1 자리이동을 이용한 테스트 알고리듬

본 논문에서 제안된 알고리듬은 메모리에 배경 데이터를 쓰는 과정에서 하나의 배경 데이터만으로 전체 메모리를 채우는 것이 아니라, 그림 8에 나온 순서대로 8개의 배경 데이터를 메모리 셀 배열 전체에 순차적으로 반복해서 써주게 된다.

	W(0)	W(1)
P0	00000000	11111111
P2	10101010	01010101
P4	11001100	00110011
P6	11110000	00001111
P1	01010101	10101010
P3	00110011	11001100
P5	00001111	11110000
P7	11111111	00000000

그림 8. 배경 데이터 적용 순서의 변화

본 논문에서 제안된 알고리듬이 기존 방식과 가장 크게 다른 점은 모든 배경 데이터를 동시에 사용하며, 자리이동(shift)을 통해 가능한 모든 테스트 패턴을 생성한다는 점이다. 자리이동을 통한 테스트 패턴의 생성과정은 다음과 같다. 즉, 초기에 메모리에 쓰여져 있던 배경 데이터를 한 워드만큼 하향이동시키고 8번째 워드의 내용은 제일 처음의 워드 주소로 이동된다. 이와 같은 과정을 모두 8번(배경 데이터의 수) 반복해준다.

본 알고리듬에서는 또한 각각의 하향이동에 대해 배경 데이터 패턴의 비트수(8번)만큼의 우측이동을 수행하여 테스트 패턴을 생성한다. 이 경우에도 한 워드의 마지막 비트 논리값은 그 워드의 첫번째 비트로 이동된다.

자리이동을 이용한 알고리듬에서는 두 가지의 이동과정이 모두 조합되어서 일어나므로 결론적으로 8비트 단위의 메모리에서는 배경 데이터들이 64번(8×8; 하향이동×우측이동)의 이동이 이루어진다. 각각의 이동이 이루어질 때마다 메모리에 쓰여져 있는 배경 데이터들에 대해 March 알고리듬이 적용되고, 64번의 이동이 모두 이루어지면 전체 테스트 과정을 종료한다.

그림 9는 그림 8의 배경 데이터를 테이블 형태로 표현한 것으로 메모리에 쓰여질 비트 패턴과 동일하다. 메모리의 임의의 한 셀은 64번의 자리이동 과정을 통하여 전체 메모리 배열상의 64개의 논리값을 모두 가질 수 있게 된다.

메모리의 임의의 셀이 양방향의 이동과정을 통해 그림 9의 비트 패턴 테이블에 나오는 모든 주소상의 논리값을 한 번씩 갖게 된다는 사실을 이용하여 이웃 패턴 감응 고장을 다음과 같이 검출할 수 있다. 본 논

y \ x	0	1	2	3	4	5	6	7
7	0	0	0	0	0	0	0	0
6	1	0	1	0	1	0	1	0
5	1	1	0	0	1	1	0	0
4	1	1	1	1	0	0	0	0
3	0	1	0	1	0	1	0	1
2	0	0	1	1	0	0	1	1
1	0	0	0	0	1	1	1	1
0	1	1	1	1	1	1	1	1

그림 9. 비트 패턴 테이블

문에서 제안한 알고리즘이 고착 고장, 천이 고장, 결합 고장에 대한 테스트가 가능하다는 것은 쉽게 증명되므로 설명을 생략한다.

가. 정적 이웃패턴 감응 고장의 검출

정적 이웃패턴 감응 고장을 테스트하기 위해서는 메모리의 모든 셀에 대해 그림 5의 모든 이웃패턴을 생성할 수 있어야만 한다. 이 때 한 워드 내에서 마지막 비트가 우측이동에 의해 첫 번째 비트의 위치로 이동하게 되므로 실제적으로 이 두 개의 비트는 이어진 것과 같고, 하향이동 또한 8번째 워드가 첫 번째 워드와 연결되어 있는 것과 같다는 점을 고려해야만 한다. 따라서 그림 5의 비트 패턴을 찾을 때 비트 패턴 테이블은 그림 10과 같이 만들어져야만 한다. 그림 5의 비트 패턴을 찾을 때는 기준셀(그림 2의 b 셀)이 항상 그림 10에서 안쪽의 사각형 내에 위치해야만 한다.

	00000000		
※ 0	00000000	0	
0	10101010	1	
0	11001100	1	
0	11110000	1	
1	01010101	0	
1	00110011	0	
1	00001111	0	
1	11111111	1	
	00000000		

(a)

	00000000	
1	11111111	1
※ 0	10101010	1
0	11001100	1
0	11110000	1
1	01010101	0
1	00110011	0
1	00001111	0
1	11111111	1
	00000000	

(b)

그림 10. 오름차순에서의 R(0), W(1) 실행 예

그림에서 '※' 기호는 그 줄에 해당하는 워드가 읽혀지는 것을 뜻하며, 음영으로 표시된 비트는 그 비트를 기준 셀로 하여서 테스트 패턴을 검출해낼 수 있다는 것을 의미한다. (a)에서 첫 번째 워드를 읽음으로서 테스트 패턴 "00000", "00001"이 생성되고, (b)에서 첫 번째 워드에 쓰기 동작을 한 후 두 번째 워드를 읽음으로서 테스트 패턴 "01111", "01110", "11001", "11000"이 생성된다(단, 여기서 비트 패턴의 위치순서는 "bABCD"이다). 이런 방식으로 R(0), W(1) 과정을 통해 그림 5의 32개의 테스트 패턴들이 모두 생성되며, 자리이동을 통하여 실제 메모리의 각 셀이 32개의 모든 이웃패턴 조합을 가지게 되므로 모든 정적 이웃패턴 감응 고장이 완벽하게 검출됨을 알 수 있다.

나. 수동 이웃패턴 감응 고장의 검출

자리이동 방식의 패턴 생성 알고리듬에서는 수동 이웃패턴 감응 고장은 테스트할 수 없다.

다. 능동 이웃패턴 감응 고장의 검출

자리이동을 이용한 테스트 기법에서는 그림 7의 테스트 패턴 중 B 셀과 C 셀에서 천이가 발생하는 테스트 패턴은 생성할 수 없다. 그러나 A 셀과 D 셀에서 천이가 발생하는 테스트 패턴은 생성이 가능하다. 그림 11은 W(1)과 R(0)을 이용하여 참이웃 셀 A에 천이가 발생하는 과정을 보여주고 있다.

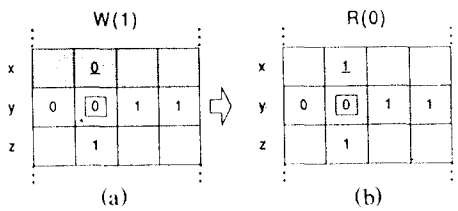


그림 11. 능동 이웃패턴 감응 고장의 검출

그림 11의 (a)에서 테스트 진행순서가 오름차순이고 주소 x에 W(1) 동작이 발생된다고 가정하면, y에서의 R(0) 동작은 그림 (b)와 같게 되므로 주소 y의 두번째 셀을 기준 셀로 할 때 기준셀의 논리값이 0이고 참이웃셀들의 값이 "↑011"인 테스트 패턴을 생성할 수 있다. 또한, 테스트 진행순서가 내림차순일 때는 참이웃 셀 D의 천이가 발생하는 테스트 패턴의 생성이 가능하다.

4.2 확장된 테스트 알고리듬

자리이동을 이용한 알고리듬에서 검출이 불가능했던 고장을 테스트하기 위해 확장된 알고리듬에서는 하나의 워드에 대해 세 단계의 쓰기 동작을 수행하는 방법을 사용하였다. 그림 12는 하나의 워드에 대한 세 단계의 쓰기 동작을 보여준다.

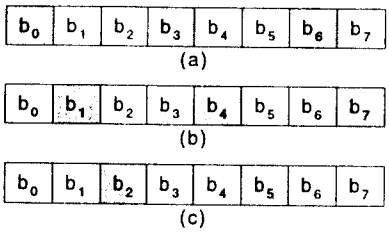


그림 12. 세 단계의 쓰기 동작

그림 12에서 음영으로 처리된 비트는 그 셀의 논리값을 반전시키는 것을 의미한다. 그림 (a)는 이 워드의 현재 논리값에서 b0, b3, b6 비트만을 반전시킨 논리값을 다시 워드에 쓰는 동작을 가리킨다. 그림 (b)와 (c)는 반전되는 셀의 위치만 다를 뿐 (a)와 같은 동작을 수행한다. 확장된 테스트 알고리듬을 포함하는 전체 개선된 테스트 알고리듬은 그림 13과 같다.

확장된 테스트 알고리듬은 자리이동을 이용한 테스트 알고리듬에 다음의 과정(March 5)을 추가시킨 것과 같다.

$W_1, R_1, W_1, W_2, R_2, W_2, W_3, R_3, W(1)$

위에서 새롭게 사용된 기호의 의미는 다음과 같다.

- $W_1 := b_0, b_3, b_6$ 셀을 반전시킨다.
- $W_2 := b_1, b_4, b_7$ 셀을 반전시킨다.
- $W_3 := b_2, b_5$ 셀을 반전시킨다.
- $R_1 :=$ W_1에 의한 배경 데이터를 읽는다.
- $R_2 :=$ W_2에 의한 배경 데이터를 읽는다.
- $R_3 :=$ W_3에 의한 배경 데이터를 읽는다.

확장된 테스트 알고리듬은 March 1 단계의 R(0), W(1)에서 R(0)가 W_1, R_1, W_1, W_2, R_2, W_2, W_3, R_3로 분리된 것과 같다. 세 단계 쓰기 동작(W_1, W_2, W_3)에서는 동시에 천이가 일어나는 셀들이 항상 두 셀만큼의 간격을 가지므로, 천이가 일어나는 셀이 기준셀로 간주되건 참이웃셀로 간주되건, 하나의 테스트 패턴을 구성하는 5개의 이웃 셀(b, A, B, C, D)에서 천이가 일어나는 셀은 항상 하나이다. 또한 확장된 알고리듬에서 반복되는 쓰기(세 번째의 W_1과 여섯 번째의 W_2)는 부분적으로 이루어진 천이를 무효화시켜 다시 초기상태에서 다음 단계의 천이가 일어날 수 있도록 해준다. 마지막으로 W(1)을 수행하므로서 March 1 단계와 같이 그림 5의 정적 이웃패턴 감응 고장에 대한 테스트 패턴이 확장된 알고리듬에서도 생성될 수 있도록 해준다.

가. 수동 이웃패턴 감응 고장의 검출

예를 들어 그림 12의 (b)에서 b_1을 천이가 발생한 기준 셀로 가정한다면, b_0와 b_2는 참이웃셀 B와 C로 볼 수 있다. 이 때, 참이웃 셀 B와 C에 해당하는 셀에서는 천이가 발생하지 않으며, 그 워드의 모든 셀이 기준 셀이 되어 수동 이웃패턴 감응 고장을 검출하는 테스트 패턴의 생성이 가능하다.

수동 이웃패턴 감응 고장은 천이가 기준셀에서만 일어나고 기준셀에서의 상향 천이와 하향 천이 각각에 대해 모든 참이웃 패턴의 조합이 생성된다면 정적 이웃패턴 감응 고장을 검출했던 배경 데이터로 수동 이웃패턴 감응 고장도 검출할 수 있다. 세 단계의 쓰기 동작에서 천이가 기준 셀에서만 일어난다는 사실을 알 수 있으며, 확장된 테스트 알고리듬에서는 매

Addr.	Init.	March 1	March 2	March 3	March 4	March 5
0	W(0)	R(0),W(1)	R(1),W(0)	R(0),W(1)	R(1),W(0)	$W_1,R_1,W_1,W_2,R_2,W_2,W_3,R_3,W(1)$
1	W(0)	R(0),W(1)	R(1),W(0)	↑	↑	$W_1,R_1,W_1,W_2,R_2,W_2,W_3,R_3,W(1)$
2	W(0)	R(0),W(1)	R(1),W(0)	↑	↑	$W_1,R_1,W_1,W_2,R_2,W_2,W_3,R_3,W(1)$
I	↓	↓	↓	R(0),W(1)	R(1),W(0)	↓
I	↓	↓	↓	R(0),W(1)	R(1),W(0)	↓
N-1	W(0)	R(0),W(1)	R(1),W(0)	R(0),W(1)	R(1),W(0)	$W_1,R_1,W_1,W_2,R_2,W_2,W_3,R_3,W(1)$

자리이동을 이용한 9N 테스트 알고리듬 — 확장된 테스트 알고리듬

그림 13. 개선된 테스트 알고리듬

단계의 쓰기가 항상 기준셀의 논리값을 반전시키므로 그림 5의 정적 이웃패턴 감응 고장 검출을 위한 테스트 패턴에서 기준 셀의 논리값이 0인 경우는 하향 천이로, 논리값이 1인 경우는 상향 천이로 볼 수 있다. 따라서 정적 이웃패턴 감응 고장을 검출했던 배경 데이터로 확장된 테스트 알고리즘에서는 수동 이웃패턴 감응 고장도 완벽하게 검출할 수 있게 된다.

나. 능동 이웃패턴 감응 고장의 검출

능동 이웃패턴 감응 고장은 천이가 참이웃 셀 B나 C에서만 일어나고 B 셀과 C 셀에서의 상향 천이와 하향 천이 각각에 대해 모든 참이웃 패턴의 조합이 생성된다면 정적 이웃패턴 감응 고장을 검출했던 배경 데이터로 능동 이웃패턴 감응 고장도 검출할 수 있다.

먼저 세 단계의 쓰기 동작에서 천이는 B 셀이나 C 셀중 하나에서만 일어나며, b_0부터 b_7까지의 각각의 셀을 기준 셀로 가정하여 B 셀과 C 셀의 천이를 세 단계의 쓰기 동작을 통하여 모두 생성시킬 수 있다. 또한 확장된 테스트 알고리듬에서는 매 단계의 쓰기가 항상 B 셀이나 C 셀의 논리값을 반전시키므로 그림 5의 정적 이웃패턴 감응 고장 검출을 위한 테스트 패턴에서 B 셀이나 C 셀의 논리값이 0인 경우는 하향 천이로, 논리값이 1인 경우는 상향 천이로 볼 수 있다. 따라서 정적 이웃패턴 감응 고장을 검출했던 배경 데이터로 능동 이웃패턴 감응 고장도 완벽하게 검출할 수 있게 된다.

V. 제안된 BIST 회로의 설계

제안된 메모리 테스트를 위한 알고리듬은 VHDL을 이용하여 구현하였다.

그림 14는 구현된 BIST 회로의 전체 블록 다이어그램을 보여주고 있다.

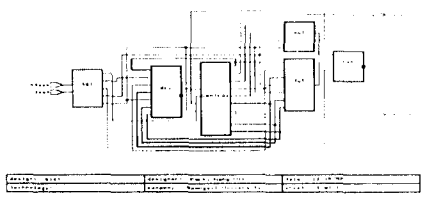


그림 14. 구현된 BIST 회로의 전체 블록 다이어그램

구현된 BIST 회로의 기능과 구조를 각각의 모듈별로 살펴보면 다음과 같다.

- 제어 회로(control logic)는 테스트 진행 과정중에 BIST 회로의 각 모듈의 동작을 제어하는 회로이다.
- 데이터 생성 회로(data generation logic)는 테스트 알고리듬의 진행순서에 따라 테스트될 메모리에 배경 데이터 패턴을 공급해주는 회로이다.
- 주소 생성 회로(address generation logic)는 테스트될 워드를 지정하는데 사용되는 주소를 생성하는 회로이다.
- 데이터 비교 회로(data comparison logic)는 메모리에서 읽어들인 테스트 결과값과 예상되는 테스트 결과값을 비교하여 메모리의 고장여부를 판단하는 회로이다.
- 테스트 클럭 생성 회로(test clock generation logic)는 BIST 회로의 외부에서 클럭 신호를 받아들여 BIST 회로에서 각각의 모듈별로 사용되는 클럭 신호를 생성해주는 회로이다.

VI. 실험 결과 및 고찰

6.1 제안된 알고리듬의 시간복잡도

이웃패턴 감응 고장을 테스팅하는데 사용되는 기존의 알고리듬들 중에서 대표적인 것들의 시간 복잡도를 살펴보면 다음의 그림 15와 같다[1].

Types of test	APNPSF	ANPSF	PNPSF	SNPSF
Locate tiling	194 N	162 N	66 N	(39 + 1/5)N
Locate 2-group	(195 + 1/2)N	(163 + 1/2)N	(67 + 1/2)N	(40 + 7/8)N
Detect 2-group	165 N	(99 + 1/2)N		(36 + 1/8)N

그림 15. 기존 NPSF 테스트 알고리듬의 시간복잡도

위의 그림에서 나온 기존의 알고리듬들 중에서 본 논문에서 제안한 알고리듬과 비교할 수 있는 것은 165 N의 시간 복잡도를 가지는 APNPSF의 Detect 2-group 기법이다.

알고리듬의 시간 복잡도라는 문제에 있어서도 알 수 있듯이, 본 논문에서 제안하는 알고리듬은 동일한

고장을 테스트하는 기존의 알고리듬과 비교할 때 더 짧은 시간(144 N)에 테스트를 수행한다는 것을 알 수 있다.

6.2 BIST 회로의 면적 오버헤드 문제

테스트를 위하여 부가되는 BIST 회로는 면적의 오버헤드가 적을수록 좋다.

그러나 BIST 회로의 면적 오버헤드 문제에 있어서 또 하나 고려해야 할 사항은 메모리에 대한 BIST 회로의 상대적인 크기 문제이다. 테스트해야 할 메모리의 크기가 커짐에 따른, 제안된 알고리듬을 구현한 BIST 회로의 상대적인 크기가 그림 16에 나타나 있다.

	Number of Instance	Total gate equivalence	Total width	Area ratio
BIST for 2K byte memory	836	1834.2	14358.4	1
BIST for 4K byte memory	845	1866.7	14611.2	1.018
BIST for 8K byte memory	860	1882.2	14768.0	1.026

그림 16. 메모리 크기에 따른 BIST 회로의 상대적인 크기

그림에서 알 수 있듯이 2K, 4K 및 8K 바이트 메모리에 대해서, 구현된 BIST 회로의 상대적인 크기는 각각 1, 1.018 그리고 1.026이므로 매우 작은 상대적 크기를 가진다고 할 수 있다.

6.3 구현된 BIST 회로의 시뮬레이션

제안된 알고리듬을 구현한 BIST 회로는 VHDL을 이용하여 구현하였다. 사용된 메모리는 Synopsys, Inc.의 DesignWare Component Library의 메모리를 사용하여 설계하였다[9]. Sparc 20 워크스테이션에서 사용된 VHDL 컴파일러는 Synopsys, Inc.의 VHDL Compiler Version 3.4a이고, VHDLdbx와 Design Analyzer Version 3.4a를 이용하여 회로의 동작 시뮬레이션과 합성(synthesis)을 수행하였다[10, 11, 12].

Ⅶ. 결 론

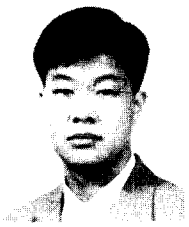
본 논문에서는 고착 고장, 천이 고장, 결합 고장 뿐만 아니라, 기존의 March 테스트로는 검출할 수 없었던 패턴 감응 고장 또한 효과적으로 검출할 수 있는 메모리 테스트 알고리듬을 제안하였다. 또한 제안된 알고리듬을 수행하는 BIST 회로를 구현하여 실제 적용에서의 그 효용성을 검증하였다.

향후 연구과제로는 BIST 회로의 오버헤드 최적화, 다수의 메모리(data memory, program memory, cache memory 등)를 갖는 시스템에의 적용 등이며 실제 상용 시스템에의 적용을 통한 테스트 용이도 및 효용성 입증에 관한 연구를 수행할 예정이다.

참 고 문 헌

1. A. J. Goor, Testing Semiconductor Memories, John Wiley & Sons Ltd., 1991.
2. M. Abramovici, M. A. Breuer and A. D. Friedman, Digital systems testing and testable design, Computer Science Press, 1990.
3. 김대우, 배성기, 이창기, 이상진, 전병실, "메모리 테스트를 위한 BIST 기술," 전자공학회지 제 22권 제 12호, 1995년 12월.
4. R. Dekker, F. Beenker, L. Thijseen, "Fault Modeling and Test Algorithm Development for Static Random Access Memories," International Test Conference, 1988.
5. J. Savir and W. H. McAnney, S. R. Vecchio, "Testing for Coupled Cells in Random-Access Memories," International Test Conference, 1989.
6. R. Dekker, F. Beenker, L. Thijseen, "A Realistic Self-Test Machine for Static Random Access Memories," International Test Conference, 1988.
7. 이화준, 소병세, "차세대 반도체 메모리의 테스트 기술," 전자공학회지 제 22권 제 12호, 1995년 12월.
8. H. D. Oberle and P. Muhmenthaler, "Test Pattern Development and Evaluation for DRAMs with Fault Simulator RAMSIM," International Test Conference, 1991.
9. DesignWare Component Databook, Version 3.4a, Synopsys Inc. 1996.
10. (V)HDL Compiler Reference Manual, Version 3.4a, Synopsys Inc. 1996.

11. Design Analyzer Reference, Version 3.4a, Synopsys Inc. 1996.
12. P. Kurup and T. Abbasi, Logic Synthesis Using Synopsys, Kluwer Academic Publishers, 1995.



박 강 민(Gang-Min Park) 정회원
1995년 : 숭실대학교 전자계산학과 졸업(B.S.)
1995년~현재 : 숭실대학교 전자계산학과 석사과정 재학중
※주관심분야 : 컴퓨터 시스템, 알고리즘, VLSI 설계, 테스팅등임

장 훈(Hoon Chang) 정회원
1987년 : 서울대학교 전자공학과 졸업(B.S.)
1989년 : 서울대학교 전자공학과 졸업(M.S.)
1993년 : University of Texas at Austin 박사학위 취득
1991년 : IBM Inc.
1993년 : Motorola Inc. Senior Member of Technical Staff.
1994년~현재 : 숭실대학교 컴퓨터학부 조교수
※주관심분야 : 컴퓨터 시스템, 알고리즘, VLSI 설계, 테스팅등임

양 승 민(Seung-Min Yang)정회원
1978년 : 서울대학교 전자공학과 졸업(B.S.)
1978년~1981년 : 삼성전자(주) 엔지니어.
1981년~1983년 : University of South Florida MS in Computer Science.
1983년~1986년 : University of South Florida Ph.D in Computer Science.
1996년~1987년 : University of South Florida 조교수
1987년~1992년 : University of Texas of Arlington 조교수.
1993년~현재 : 숭실대학교 컴퓨터학부 부교수
※주관심분야 : 결함허용 실시간 시스템, 분산처리 시스템, 소프트웨어 공학