고집적 메모리를 위한 효율적인 고장 진단 알고리듬

An Efficient Diagnosis Algorithm for High Density Memory

朴 漢 源* · 姜 成 昊**
(Han-Won Park · Sung-Ho Kang)

社團法人 大韓電氣學會
THE KOREAN INSTITUTE OF ELECTRICAL ENGINEERS

고집적 메모리를 위한 효율적인 고장 진단 알고리듬

論 文
50D-4-5

An Efficient Diagnosis Algorithm for High Density Memory

朴 漢 源* · 姜 成 昊**
(Han-Won Park · Sung-Ho Kang)

Abstract - As the high density memory is widely used in the various applications, the need for the reproduction of memory is increased. In this paper we propose an efficient fault diagnosis algorithm of linear order *O(n)* that enables the reproduction of memory. The new algorithm can distinguish various fault models and identify all the cells related to the faults. In addition, a new BIST architecture for fault diagnosis is developed. Using the new algorithm, fault diagnosis can be performed efficiently. And the performance evaluation with previous approaches proves the efficiency of the new algorithm.

Key Words : fault model, memory test, fault diagnosis, BIST

1. 서 론

메모리의용량이 급속도로 증가함에 따라 결함의 빈도 또한 높아지고 있으며 메모리 생산에 있어 중요한 요소 중 하나인 수율(yield)도 낮아지는 문제가 발생한다. 이 문제를 해결하고 수율을 높이기 위해 메모리를 수리(repair)하는 것이 중요한 비중을 차지하며 이는 다음과 같은 두 가지 단계로 나뉘어 진다. 우선 메모리에서의 모든 고장을 검출하고 그 위치를 결정하는 진단(diagnosis)과 검출된 고장을 위해 여분을 배치하는 단계가 필요한 두 단계이다. 그러므로 메모리의 수리를 위해서는 메모리 테스토와 더불어 검출된 고장의 종류와 위치를 명확하게 결정할 수 있는 진단 과정에 대한 연구가 진행되어야 한다. 하지만 지금까지의 연구는 대부분 메모리를 테스트하기 위한 방법들을 다루어 왔다[2, 3, 7, 8, 9]. 따라서 본 논문에서는 메모리에서의 효율적인 고장 진단 알고리듬을 제안한다.

지금까지 메모리 진단에 대한 연구들은 다음과 같은 범위에서 이루어져 왔다. 우선 [1]에서는 메모리에서의 결합 고장(coupling fault : CF)과 고착 고장(stuck at fault : SAF)의 위치를 확인하기 위한 진단 과정이 제안되었으나 다양한 고장 모델을 고려하지 못하여 실제적으로 메모리에서 고장이 발생할 경우 이를 진단하기가 어렵다. [4]에서는 메모리에서 고장을 진단하기 위한 마치 테스트(march test)가 제안되었으나 새로운 고장 모델에 근거해 고장을 진단하는 방법이 사용되었기 때문에 메모리 테스트에서 일반적으로 사용되는 기능적 고장 모델에 대한 분석과 진단이 이루어지지 않는다. [5]에서는 메모리에서 진단 과정을 수행할 수 있는 내장된 자체 테스트(Built-in self test : BIST)를 위한 구조가 제안되었다. 하지만 이 경우 진단 과정에서 고려하는 고장 모델들에 대한 분석이 이루어지지 않고 있으며 BIST에서 스캔(scan)을 이용하여 단지 고장이 검출되는 셀의 위치만을 확인할 수 있다. 또 [12]에서는 메모리 테스트 알고리듬에서 각각의 고장 모델에 따라 알고리듬에서의 고장 검출 여부에 대한 결과에 대한 표를 작성하고 이를 실제의 테스트 결과와 비교하여 메모리의 셀에서 고장이 존재할 경우 고장의 종류를 분석한다. 이 경우 다양한 종류의 고장들에 대한 분석이 가능하지만 결합 고장의 경우 피결합셀(coupled cell)의 위치만을 확인하고 결합원셀(coupling cell)의 위치를 확인하지 못하므로 고려하는 고장 모델에 관련된 모든 셀을 진단하지 못한다. [13]에서는 제안된 마치 테스트를 초기에 사용하여 단일 셀 고장(single cell fault)과 다중 셀 고장(multiple cell fault)를 구별하고 결합원셀을 찾기 위한 패턴을 사용하여 결합원셀의 확인이 가능하다. 하지만 결합원셀을 찾기 위한 패턴을 가하는 어드레스 시퀀스가 고장 모델에 관계없이 일정하므로 비효율적인 단점이 있으며 초기 마치 테스트가 반드시 이루어져야 한다.

따라서 본 논문에서는 메모리에서 일반적으로 사용되는 고장 모델들에 근거하여 테스트에서 고장이 검출될 경우 그 고장의 종류와 고장과 관련된 모든 셀의 위치를 확인할 수 있는 진단 방법을 제안한다. 고려하는 고장 모델의 모든 고장의 종류와 위치를 확인할 수 있는 진단 알고리듬을 개발하여 단순히 테스트에서 고장이 검출되는 셀 뿐 아니라 고장의 원인이 되는 셀도 확인할 수 있게 하여 고장에 관련된 모든 셀의 위치를 확인할 수 있도록 한다. 또 고장 모델에 따라 적절한 어드레스 시퀀스를 사용함으로서 진단 과정을 효율적인 시간내에 수행할 수 있다. 또 이러한 진단 알고리듬뿐 아니라 테스트 알고리듬의 수행 결과를 이용한 진단 방법을 제

* 準 會 員 : 延世大 工大 電氣電子工學科 碩士課程
** 正 會 員 : 延世大 工大 電氣電子工學科 副教授 · 工博
接受日字 : 2001年 3月 2日
最終完了 : 2001年 4月 2日

시하여 테스트를 고려한 진단을 가능하게 한다. 그리고 진단 과정의 수행을 위해 새로운 BIST 구조를 제안한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 진단 과정에서 고려하는 고장 모델들에 대해 설명한다. 3장에서는 고려하는 고장 모델들의 진단을 위한 패턴들을 제시한다. 4장에서는 메모리에서의 테스트 과정이나 결과들에 대한 정보가 없는 경우와 있는 각각의 경우들에 대한 진단 알고리듬을 제시한다. 5장에서는 march C-알고리듬을 예로 들어 제안된 진단 방법을 설명한다. 6장에서는 메모리에서의 고장의 진단 과정의 수행을 위한 새로운 BIST 구조를 제안하며 7장에서는 기존의 제안된 메모리 진단 방법들과 본 논문에서 제안된 진단 방법들을 비교한 성능 평가 비교 및 결과를 보여준다. 그리고 8장은 결론으로써, 본 논문을 통한 연구 내용을 요약한다.

2. 고장 모델

효율적인 진단 과정을 위해서는 고려해야 할 고장 모델을 결정해야 한다. 여기에서는 [6]에서 제시된 일반적으로 사용되는 기능적 고장 모델에 근거해 다음과 같은 고장 모델들을 고려하기로 한다.

- 어드레스 디코더 고장 (Address decoder fault : AF)
  메모리에서 여러 기능별 블럭 중 어드레스 디코더 부분에서의 고장을 말하며 이는 보통 메모리 테스트에서 메모리 셀 부분에서의 고장으로 매핑(mapping)된다.
- 고착 고장 (Stuck at fault : SAF)
  하나 또는 그 이상의 메모리 셀이나 선이 고착되어서 값이 0 혹은 1로 항상 고정되어 있는 상태의 고장을 말한다. 따라서 이러한 고장을 검출하려면 메모리 셀에서 0과 1의 값을 읽는 동작을 수행하여야 한다.
- 천이 고장 (Transition fault : TF)
  고착 고장의 특수한 경우라고 할 수 있는데 셀이 0에서 1로 혹은 1에서 0으로의 천이 쓰기 동작을 수행하지 못하고 실패하는 고장을 말한다. 그러므로 이 고장을 검출하기 위해서는 0에서 1로의 천이 쓰기 동작과 1에서 0으로의 천이 쓰기 동작을 수행한 후 각각의 값들을 읽는 동작을 수행해야 한다.
- 결합 고장 (Coupling fault : CF)
  두 개 또는 그 이상의 셀들 사이에서의 비정상적인 연결로 인해 발생하는 고장으로 한 셀에서의 동작이 다른 셀에 영향을 끼쳐 오동작을 발생시키는 고장을 말한다. 따라서 이 고장은 고착 고장이나 천이 고장과 달리 테스트에서 고장이 검출되는 셀 외에도 고장과 관련된 셀이 존재하게 된다. 따라서 진단 과정에서는 고장이 검출 되는 셀뿐만 아니라 고장과 관련된 셀 모두를 확인하는 과정이 필요하다.
- 이웃 패턴 감응 고장 (NPSF)
  기준셀 주위의 이웃한 여러 셀들의 특정 패턴이나 천이 쓰기 동작이 기준셀에 영향을 끼쳐 오동작을 일으키는 형태의 고장을 말한다. 이러한 고장의 경우 여러 가지로 세분화될 수 있는데 각각 이웃셀들의 특정 패턴에서의 고착 고장, 천이 고장, 결합 고장과 유사한 고장의 특성을 갖는다.

이러한 고장 모델에 포함되는 모든 고장의 경우 진단 과정에서는 고장의 종류에 따른 구별이 가능해야 하며 고장과 관련된 모든 셀들의 위치를 명확하게 결정할 수 있도록 하는 진단 패턴들을 결정해야 한다.

3. 메모리 진단을 위한 패턴

이 장에서는 본 논문에서 고려하는 메모리의 모든 고장들에 대해 진단 과정 수행을 위해 필요한 패턴들을 분석한다. 또한 고장과 관련된 셀을 찾기 위한 패턴들도 분석한다. 메모리 테스트에서 검출된 셀의 위치를 (i, j)라고 가정한다. 우선 이 고장셀이 고착 고장이나 천이 고장처럼 하나의 셀과 관련해서 일어나는 단일 셀 고장인지 결합 고장이나 NPSF처럼 여러 개의 셀들이 관련되어 일어나는 다중 셀 고장인지를 진단하는 과정부터 수행하여야 한다. 그리고 만약 검출된 고장이 다중 셀 고장이라면 이 고장에 대해 적절한 패턴을 가해 주어 단지 두 셀들간에 결합 관계가 존재하여 발생하는 결합 고장과 같은 형태의 고장인지 단지 두 셀들 간에 존재하는 결합 고장이 아니라 검출된 셀 주위의 이웃한 여러 개의 셀들과 관련된 NPSF 형태의 고장인지를 규명해야 한다. 그리고 진단 과정의 모든 단계에서 NPSF 고장이 존재하여 진단 과정을 방해할 가능성이 있기 때문에 NPSF 고장인지에 대한 테스트로서 기준셀의 이웃셀들의 값을 바꾸어 준 후 같은 패턴을 가해주는 과정을 추가하여 수행한다. 이러한 일련의 과정들과 각각의 고장의 검출을 위한 패턴들은 다음과 같이 나타낼 수 있다.

3.1 고착 고장, 천이 고장 : PTN-ST

$$W_{i,j}(0),\ R_{i,j}(0),\ W_{i,j}(1),\ R_{i,j}(1)\ W_{i,j}(0),\ R_{i,j}(0)$$

여기서 $W_{i,j}$는 셀 (i, j)에 쓰기 동작을 수행하는 것을 의미하고 $R_{i,j}$는 셀 (i, j)의 값을 읽는 동작을 의미한다. 고장셀 (i, j)에 위와 같이 0과 1을 쓰고 읽는 동작을 수행한다. 이러한 동작들을 수행함으로서 고장이 결합 고장인지 고착 고장나 천이 고장과 같은 단일셀 고장인지를 알 수 있다. 우선 위의 동작들을 수행할 때 고장이 검출되지 않는다면 고착 고장나 천이 고장과 같은 단일셀 고장이 아니므로 결합 고장이나 NPSF와 같은 다중 셀 고장에 대한 진단 과정이 필요하다. 이와는 달리 만일 여기서 $R_{i,j}(0)$이나 $R_{i,j}(1)$이 실패하여 고장이 검출된다면 고착 고장, 천이 고장과 같은 단일 셀 고장으로 진단된다. 하지만 이 경우 고장이 발생한 순간 셀 (i, j)의 이웃셀들의 특정 패턴들 때문에 위의 동작들이 실패하는 passive, static NPSF일 수도 있으므로 진단 과정에서 착오가 있을 수 있다. 따라서 셀 (i, j)의 이웃셀들의 값을 바꾸어 준 후 위의 동작들을 다시 수행함으로서 고착 고장, 천이 고장과 같은 단일셀 고장인지 NPSF인지가 결정되므로 진단 과정은 완료된다.

3.2 결합 고장 : PTN-C

3.1의 과정에서 고장이 검출되지 않은 경우들에 대해 위와 같은 패턴들을 가해 주는 진단 과정을 수행한다. 이 과정에서는 메모리에서 존재하는 고장이 결합 고장일 경우 고장의

원인이 되는 결합원셀의 위치를 확인한다.

$W_{i,j}(0)$ ↕ ($W_n(0)$, $W_n(1)$, $R_{i,j}(0)$, $W_n(0)$, $R_{i,j}(0)$)

$W_{i,j}(1)$ ↕ ($W_n(0)$, $W_n(1)$, $R_{i,j}(1)$, $W_n(0)$, $R_{i,j}(1)$)

여기서 Wi,j는 셀 (i, j)에 쓰기 동작을 수행하는 것을 의미하며 Wn은 셀 (i, j)를 제외한 모든 셀들에 쓰기 동작을 수행하는 것을 의미한다. 우선 셀 (i, j)에 0이나 1의 값을 쓰고 다른 셀들에 0을 쓴 후 다시 1을 쓰는 동작을 수행한 후 셀 (i, j)의 값을 읽어서 확인해 본다. 그리고 다시 0을 쓰고 셀 (i, j)의 값을 읽어서 확인한다. 이러한 동작을 어드레스를 변화시키면서 반복적으로 수행하는 중에 고장이 검출되지 않는다면 결합 고장이 아니므로 NPSF의 검출에 대한 진단을 수행하여야 한다. 만일 위의 동작을 수행하는 중 고장이 검출된다면 피결합셀 뿐 아니라 결합원셀의 위치도 확인된다고 생각할 수 있다. 하지만 이 경우 결합원셀이 기준셀의 이웃셀중에 하나일 경우에는 고장이 발생한 순간 셀 (i, j)의 이웃셀들이 특정한 값을 가진 상태에서만 위의 동작이 실패하는 것일 가능성이 있으므로 결합 고장이 아니라 active NPSF일 수도 있다. 따라서 3.1의 경우와 마찬가지로 이웃셀들의 패턴을 바꾸어 준 후 위의 패턴들을 다시 수행함으로서 결합 고장인지 active NPSF인지를 결정하고 진단 과정을 완료하게 된다.

3.3 NPSF : PTN-N

3.1과 3.2의 과정을 통해 검출되지 않는 고장일 경우 아래와 같이 좀 더 특수한 형태의 고장인 NPSF에 대한 진단 과정을 수행하게 된다. 여기서 테스트에서 고장이 검출된 셀 (i, j)는 기준셀이 되고 상하 좌우의 인접한 셀들이 이웃셀이 된다. 이러한 NPSF의 검출을 위해 필요한 패턴들은 다음과 같다.

3.3.1 Active NPSF

$W_{i,j}(D)$

APPLY(ANPSF) : (표 1 참조)

$R_{i,j}(D)$

여기서 D는 데이터 0 과 1을 의미한다. 또 APPLY(ANPSF)는 표 1과 같은 NPSF의 검출을 위한 패턴들을 이웃셀에 적용하는 것을 의미한다. 따라서 D가 0인 경우와 1인 경우 모두에 대해서 패턴들을 인가하고 진단 과정을 수행한다. active NPSF는 이웃셀들이 특정 값을 가진 상태에서 이웃셀 중 하나의 셀에서 천이 쓰기 동작이 일어날 경우 기준셀의 값이 바뀌게 되는 고장이다. 따라서 이 경우에는 기준셀에 0과 1의 값을 쓰고 이웃셀들에서 하나의 천이 동작을 포함한 모든 패턴들에 대해 각각 기준셀에 대해 0과 1을 읽는 동작을 수행하여 검출 가능하다. 이러한 NPSF의 경우에 필요한 패턴들은 다음 표1과 같다. 여기서 u는 0에서 1로의 쓰기 동작을 의미하고 d는 1에서 0으로의 쓰기 동작을 의미한다.

3.3.2 Passive NPSF

APPLY(PNPSF) : (표 2참조)

$W_{i,j}(0)$, $R_{i,j}(0)$, $W_{i,j}(1)$, $R_{i,j}(1)$ $W_{i,j}(0)$, $R_{i,j}(0)$

Passive NPSF 고장은 이웃셀들이 특정값을 가지고 있을 때 기준셀에서 0에서 1로의 쓰기 동작이나 1에서 0으로의 쓰기 동작을 제대로 수행하지 못하는 형태의 고장을 말한다. 따라서 passive NPSF 고장의 경우는 active NPSF의 경우와 달리 이웃셀들의 모든 천이 쓰기 동작을 고려할 필요없이 가능한 모든 상태들에 대해 각각 위의 동작을 기준셀에 수행하고 읽기 동작을 수행하여 값을 확인하면 된다. 이러한 passive NPSF의 경우 필요한 패턴들은 표 2과 같다.

표 1. Active NPSF검출을 위해 필요한 패턴들
Table 1. Patterns for active NPSF

(i,j)	00000000000000001111111111111111
(i-1,j)	uuuuuuuuddddddddduuuuuuuuddddddddd
(i,j+1)	00001111000011110000111100001111
(i+1,j)	00110011001100110011001100110011
(i,j-1)	01010101010101010101010101010101
(i,j)	00000000000000001111111111111111
(i-1,j)	00001111000011110000111100001111
(i,j+1)	uuuuuuuuddddddddduuuuuuuuddddddddd
(i+1,j)	00110011001100110011001100110011
(i,j-1)	01010101010101010101010101010101
(i,j)	00000000000000001111111111111111
(i-1,j)	00001111000011110000111100001111
(i,j+1)	00110011001100110011001100110011
(i+1,j)	uuuuuuuuddddddddduuuuuuuuddddddddd
(i,j-1)	01010101010101010101010101010101
(i,j)	00000000000000001111111111111111
(i-1,j)	00001111000011110000111100001111
(i,j+1)	00110011001100110011001100110011
(i+1,j)	01010101010101010101010101010101
(i,j-1)	uuuuuuuuddddddddduuuuuuuuddddddddd

표 2. Passive NPSF 검출을 위해 필요한 패턴들
Table 2. Patterns for passive NPSF

(i,j)	uuuuuuuuuuuuuuuuddddddddddddddddd
(i-1,j)	00000000111111110000000011111111
(i,j+1)	00001111000011110000111100001111
(i+1,j)	00110011001100110011001100110011
(i,j-1)	01010101010101010101010101010101

3.3.3 Static NPSF

$W_{i,j}(D)$

APPLY(SNPSF) : (표 3참조)

$R_{i,j}(D)$

여기서 D는 데이터 0과 1을 의미하며 0인 경우와 1인 경우에 대해서 표3과 같은 패턴들을 이용하여 진단 과정을 수행한다. Static NPSF은 이웃셀들이 특정 패턴을 가지고 있을 때 기준셀의 값이 0혹은 1의 일정한 상태를 갖게 되는 형태의 고장을 말한다. 따라서 이 경우에는 기준셀에 0과 1의 값을 쓰고 이웃셀들의 모든 패턴들에 대해 각각 기준셀에 대해

0과 1을 읽는 동작을 수행하여 검출 가능하다. static NPSF의 경우 필요한 패턴들은 표 3과 같다.

표 3. Static NPSF 검출을 위해 필요한 패턴들
Table 3. Patterns for static NPSF

(i,j)	00000000000000001111111111111111
(i-1,j)	00000000111111110000000011111111
(i,j+1)	00001111000011110000111100001111
(i+1,j)	00110011001100110011001100110011
(i,j-1)	01010101010101010101010101010101

3.4 어드레스 디코더 고장 : PTN-A

앞의 3.1에서 3.3까지의 과정을 통해서 고장의 종류와 위치가 진단되지 않는다면 어드레스 디코더 고장일 가능성이 있고 이를 확인하기 위해 어드레스 디코더 고장에 대한 테스트를 수행한다. 이 고장은 메모리에서 어드레스 디코더 부분에서 관련된 고장으로 대부분의 경우 메모리 셀에서의 고장으로 매핑(mapping)된다. [6]에서는 다음과 같은 조건을 만족하는 테스트를 수행할 때 어드레스 디코더 고장을 검출할 수 있음을 보여준다.

$\Uparrow(R(D), \cdots, W(\bar{D}))$
$\Downarrow(R(\bar{D}), \cdots, W(D))$

앞의 진단 과정들에서는 위와 같은 패턴들을 모두 인가하지 않았기 때문에 모든 어드레스 디코더 고장을 검출할 수는 없다. 그리고 모든 어드레스 디코더 고장을 진단하기는 어려우므로 앞에서 말한 모든 고장들에 대한 진단 패턴으로도 검출이 되지 않을 경우에 한해 위에서 말한 조건을 만족하는 가장 짧은 길이의 MATS+ 알고리듬을 이용하여 어드레스 디코더 고장에 대한 진단을 수행한다.

3.5 워드 본위 메모리(word oriented memory)에서의 진단

지금까지의 테스트와 진단 패턴들은 비트 본위 메모리(bit oriented memory)의 측면에서 설명되었다. 그런데 하나의 비트(bit)가 아닌 워드(word)단위로 읽고 쓰는 동작을 수행하는 워드 본위 메모리의 경우로 확장하여 보면 앞에서 고려한 고장외에도 새롭게 고려해야할 고장이 존재한다. 서로 다른 워드의 셀들 사이에서 발생하는 결합 고장은 앞에서 설명했던 패턴들로 검출이 가능하다. 하지만 하나의 워드내의 셀들간에 발생하는 결합 고장의 경우에는 비트 본위 메모리에 근거한 일반적인 알고리듬으로만은 존재하는 고장을 검출할 수 없다. 따라서 이를 위해 하나의 워드내에서 발생할 수 있는 결합 고장을 검출하기 위한 추가의 패턴이 필요하다는 문제가 발생하게 된다. 이와 관련해 워드 본위 메모리에서 하나의 워드내에서 발생할 수 있는 고장들을 검출할 수 있는 패턴들이 제안되었다[10][11]. 이를 통해 서로 다른 워드의 셀들간에 발생할 수 있는 결합 고장에 대한 비트 본위 테스트에 하나의 워드내에서 발생하는 고장들에 대한 패턴을 추가로 결합함으로서 고장들을 효율적으로 검출할 수 있게 된다. 예를 들어 4비트 워드인 경우의 워드 본위 메모리에서 하나의 워드내에서 발생할 수 있는 결합 고장을 검출하기 위해 추가로 필요한 패턴은 표 4와 같다.

따라서 4비트 워드 메모리의 경우 고장에 대한 테스트와 진단을 수행하기 위해서는 표 4의 패턴들을 사용한 과정이 추가로 이루어져야 한다. 하지만 실제로는 표4에 나타낸 하나의 워드내의 셀들간의 결합 고장을 위한 패턴 중 3개의 패턴들은 비트 본위 테스트에서 사용되었으므로 추가로 수행되지 않아도 된다. 이러한 워드내의 결합 고장을 위한 패턴을 추가로 수행하기 위한 과정을 일반화하면 b비트 워드 메모리의 경우 $6*n/b*\log_2 b$ 만큼의 동작을 필요로 한다.

표 4. 4 비트 워드 본위 메모리에 필요한 패턴
Table 4. Patterns for a 4 bit word oriented memory

#	워드 패턴
0	0000
1	1111
2	0000
3	0101
4	1010
5	0101
6	0011
7	1100
8	0011

4. 메모리 진단 알고리듬

이 장에서는 앞에서 설명한 진단 패턴에 근거해서 메모리에서 고장이 발생할 경우 진단 알고리듬을 제시한다. 메모리에서의 진단은 테스트 방법에 대한 정보가 있는 경우와 정보가 없는 경우로 나뉘어질 수 있다. 우선 테스트 방법에 대한 정보가 없는 경우는 테스트 알고리듬을 수행하는 과정이나 그에 따른 결과들에 대해 알지 못하는 경우를 말하며 이 경우 단지 고장이 검출된 셀만을 파악한 경우를 말한다. 이러한 경우에는 테스트에서 고장이 검출된 셀을 대상으로 3장에서 설명된 진단 패턴들을 적절한 시퀀스로 가해 주어서 검출되는 고장의 종류를 분류하고 고장과 관련된 모든 셀들을 확인할 수 있어야 한다. 다음으로 테스트 방법에 대한 정보가 있는 경우는 테스트 알고리듬을 수행하는 과정이나 각 셀에서의 결과값에 대한 정보를 얻을 수 있는 경우를 말한다. 이러한 경우에는 메모리를 테스트하는 알고리듬에 따른 분석이 필요하다. 또 알고리듬 수행 시 각 셀에서의 결과를 진단 과정에 이용함으로서 고장의 예측 가능성을 증대시킨다. 따라서 테스트를 고려한 진단을 가능하게 된다. 이러한 테스트 알고리듬을 이용한 진단 방법은 뒤에서 설명하도록 하고 우선 진단 과정 이전에 수행되는 테스트 방법이나 결과를 이용하지 못하는 경우의 일반적인 진단 알고리듬을 제안한다.

그림 1에서는 테스트에서 고장이 검출된 셀에 대해 고려하는 모든 고장 모델들을 포함하는 진단 과정을 나타낸다. 그림 1의 경우 NPSF도 포함되므로 일반적인 DRAM의 경우에 해당되는 진단 알고리듬이라 할 수 있다. 하지만 SRAM의 경우에는 NPSF를 고려하지 않으므로 NPSF에 대한 진단 과정을 제외한다면 SRAM의 경우의 진단 알고리듬은 그림 1의 진단 과정이 매우 간략하게 수정된 형태와 같다.

그림 1의 메모리 진단 과정에서 PTN-ST, PTN-C, PTN-N, PTN-A 과정은 앞의 3장에서 설명한 고착 고장/천이 고장, 결합 고장, NPSF, 어드레스 디코더 고장에 대한 각각의 진단 패턴을 의미한다. 그리고 PTN-ST/N과 PTN-C/N

는 고장이 검출된 셀 (i, j)의 이웃셀들의 패턴을 바꾸어 준 후 동일한 동작을 수행하는 과정을 의미한다. 이러한 과정을 통하여 PTN-ST나 PTN-C의 과정을 통해 각각 고착 고장, 천이 고장이나 결합 고장을 NPSF와 착오를 일으키지 않게 된다. 진단 과정을 차례대로 설명하면 다음과 같다.

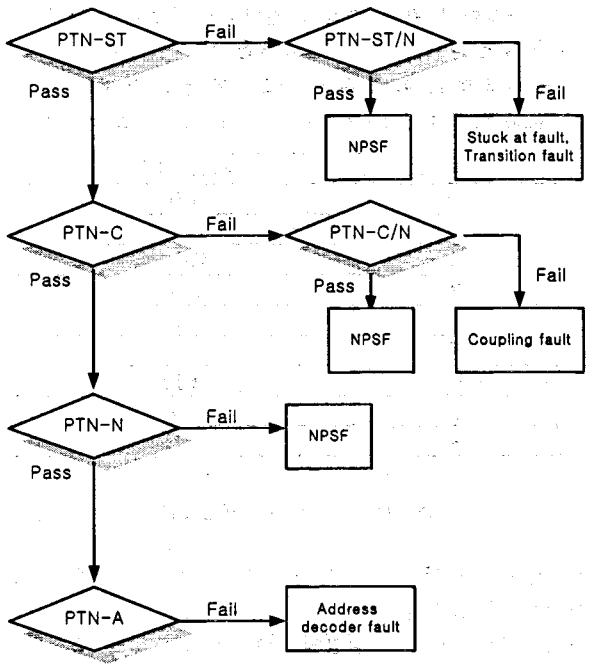

그림 1. 메모리 고장 진단 알고리듬
Fig 1. Memory fault diagnosis algorithm

우선 PTN-ST의 패턴은 단일 셀 고장인지 여부를 판단하기 위해 사용된다. PTN-ST의 과정을 통과할 경우 단일 셀 고장이 아닌 다중 셀 고장으로 예측되어 PTN-C의 과정을 수행하게 되며 실패할 경우에는 단일 셀 고장로 예상되지만 NPSF일 가능성이 존재하므로 위에서 설명한 것처럼 이웃셀의 패턴을 바꾸어 주는 PTN-ST/N을 통해 고착이나 천이 고장인지 NPSF인지 진단을 완료하게 된다. 다음으로 PTN-ST을 통과한 경우에 대해 PTN-C의 패턴을 인가하게 되는데 이를 통해 결합 고장을 발생하게 하는 결합원셀을 찾을 수 있다. PTN-C의 과정에서 실패하는 경우 역시 결합 고장외에 NPSF일 가능성이 있으므로 이웃셀 의 패턴을 바꾸어 주는 PTN-C/N를 통해 결합 고장인지 NPSF인지를 진단하는 과정을 완료하게 된다. 그리고 PTN-C의 과정을 통과할 경우에는 NPSF에 대한 테스트인 PTN-N의 과정을 수행하게 된다. 여기서 실패할 경우에는 NPSF로 진단 과정이 완료되며 통과할 경우에는 어드레스 디코더 고장에 대한 테스트인 PTN-A의 과정 을 수행하게 된다. 진단 과정의 마지막 과정인 PTN-A에서 실패할 경우는 어드레스 디코더 고장으로 진단이 완료된다. 따라서 이러한 과정들을 통해 메모리에서 고장이 발생할 경우 고장의 종류와 고장이 검출되는 셀 뿐만 아니라 고장의 원인이 되는 셀을 포함해 고장과 관련된 모든 셀을 진단하게 된다.

제안된 메모리 진단 과정을 요약하면 다음과 같다. 우선 단일셀 고장의 여부를 판단하기 위해 사용되는 PTN-ST는 6만큼의 동작수를 필요로 하게 된다. 그리고 결합 고장의 경우 결합원 셀을 찾기 위해 사용되는 PTN-C는 $2+10n$의 동작을 필요하게 된다. 또 NPSF의 검출을 위해서는 active NPSF의 경우 258, passive NPSF의 경우 112, static NPSF의 경우 66의 동작이 필요하므로 PTN-N은 436의 동작수를 필요하게 된다. 마지막으로 어드레스 디코더 고장을 검출하기 위한 PTN-A는 $5n$의 동작이 필요하다. 따라서 전체 진단 과정의 길이는 $15n+444$과 같다. 그리고 워드 본위 메모리의 경우 이와 더불어 하나의 워드내의 셀들간의 결합 고장의 검출을 위해서는 $6*n/b*\log_2 b$의 동작이 추가로 수행되어야 한다. 여기서 b는 하나의 워드당 비트수를 의미한다. 따라서 워드 본위 메모리의 경우 전체 진단 알고리듬의 최대 수행 시간은 $(15+6*\log_2 b)*n/b+444$와 같다.

앞에서는 메모리 테스트의 알고리듬 등의 과정들에 대한 정보를 이용하지 않는 일반적인 메모리의 진단 알고리듬을 제시하였다. 다음으로 테스트 알고리듬의 수행 결과를 이용한 진단 방법에 대해 설명하도록 한다.

앞에서 제시한 진단 알고리듬의 경우는 메모리 테스트에 의해 고장이 검출되는 셀을 대상으로 진단 패턴을 가하여서 발생한 고장의 종류와 그 고장에 관련된 셀들의 위치를 확인하는 동작을 수행하는 것이다. 이 경우 진단 과정 이전의 메모리 테스트 알고리듬은 단지 고장을 검출하는 목적으로 사용되며 진단을 위한 알고리듬은 이와는 별개로 검출된 셀에 적절한 패턴을 가하는 과정을 통해 진단 과정을 수행하게 되는 것이다. 하지만 메모리 테스트를 위한 알고리듬의 각각의 읽기 동작에서 고장이 없을 경우와 고장이 존재하는 경우 또 그 고장이 어떠한 종류의 고장 모델인지에 따라서 알고리듬에서 읽기 동작을 수행하였을 때 나타나는 결과가 달라질 수 있다. 따라서 고려하는 각각의 고장 모델에 따라 테스트 알고리듬의 읽기 동작에서 나타나는 결과들이 분석된다면 테스트 알고리듬을 메모리에 가했을 때 나타나는 결과를 관찰하는 것으로서 메모리에서의 고장에 대한 예측의 가능성이 높아지므로 진단 과정 또한 일반적인 진단 과정과 다른 형태로 변화될 수 있다. 이러한 경우에는 읽기 동작에서 고장이 검출되어 테스트 알고리듬이 중지되는 시점의 마치 요소(march element)에서 최초로 발생하여 검출될 수 있는 고장에 대한 분석이 필요하다.

여기에서는 메모리의 테스트 알고리듬으로 임의의 마치 요소를 가지는 알고리듬이 사용된 경우 고장이 발생했을 때 진단하는 방법에 대해 알아본다. 우선 이를 위해서는 진단 과정에서 고려하는 각각의 고장 모델들과 이를 검출하는 마치 요소들에 대한 분석이 필요하다. 그리고 테스트에서 고장이 검출된 부분의 마치 요소와 비교해서 진단 과정을 수행해야 한다. 진단 과정에서 고려하는 각각의 고장 모델을 테스트 시 검출할 수 있는 패턴은 다음과 같이 표 5와 같이 나타낼 수 있다. 표 5에는 진단 과정에서 고려하는 고장들과 이를 테스트 알고리듬에서 검출하는 테스트 패턴들을 나타낸다. 그러므로 마치 형태의 테스트 알고리듬을 통해 고장을 검출할 경우 고장을 검출한 읽기 동작이 포함되는 마치 요소들 중 표 5에 포함되는 것들에 따른 고장으로 예측될 수 있고 그에 따른 진단 과정을 우선적으로 수행한다. 표 5에 나타난 고장들 중 결합 고장의 경우의 a와 v는 각각 결합원셀과 피

결합셀을 나타내며 a<v는 결합원셀의 어드레스가 피결합셀의 어드레스보다 하위인 경우를 말하며 a>v는 결합원셀의 어드레스가 피결합셀의 어드레스보다 상위인 경우를 말한다. 예를 들어 ↑:0(a<v)와 ↑:0(a>v)는 동작의 형태는 같은 고장이지만 결합원셀과 피결합셀의 어드레스의 관계에 따라 나뉘어지는 것이고 이를 검출하는 마치 요소와 진단 패턴도 달라지게 되는 것이다.

표 5. 각각의 고장 모델을 검출하는 마치 요소들
Table 5. March elements detecting each fault model

고장 모델		마치 요소
SA0		⇕(R1)
SA1		⇕(R0)
TF↑		⇕(R0,W1)⇕(R1)
TF↓		⇕(R1,W0)⇕(R0)
결합 고장	↑:0(a<v)	⇓(R0,W1)⇕(R1)
	↑:0(a>v)	⇑(R0,W1)⇕(R1)
	↓:0(a<v)	⇑(R1,W0)
	↓:0(a>v)	⇓(R1,W0)
	↑:1(a<v)	⇑(R0,W1)
	↑:1(a>v)	⇓(R0,W1)
	↓:1(a<v)	⇓(R1,W0)⇕(R0)
	↓:1(a>v)	⇑(R1,W0)⇕(R0)

진단 과정을 수행하기 위해 각 고장들에 대한 변형된 진단을 위한 패턴들은 다음 표 6과 같이 나타낼 수 있다. 따라서 마치 알고리듬을 이용한 테스트를 수행하는 중 고장이 검출되면 고장을 검출하는 마치 요소에 따라 표 5과 같은 고장 모델들에 대한 진단 과정이 필요하며 이에 따라 표 6과 같이 각각의 고장 모델들에 대해 진단 패턴을 사용하여야 한다.

따라서 테스트 과정이나 결과에 대한 정보를 이용한 고장 진단 과정은 그림 1의 진단 알고리듬을 수행함에 있어 각각의 일반적인 진단 패턴들 대신 표 5와 표 6에 의해 예측되는 고장 모델들에 따른 변형된 진단 패턴들을 사용해서 좀 더 간략화될 수 있으며 진단 과정에 소요되는 시간의 측면에서 볼 때 효율적인 진단 과정을 수행할 수 있게 한다.

표 6. 고장 모델들에 따른 변형된 진단 패턴들
Table 6. Modified diagnosis patterns for each fault model

고장 모델		변형된 진단 패턴
SA0		$W_{i,i}(1)$, $R_{i,i}(1)$
SA1		$W_{i,i}(0)$, $R_{i,i}(0)$
TF↑		$W_{i,i}(0)$, $R_{i,i}(0)$, $W_{i,i}(1)$, $R_{i,i}(1)$
TF↓		$W_{i,i}(1)$, $R_{i,i}(1)$, $W_{i,i}(0)$, $R_{i,i}(0)$
결합 고장	↑:0(a<v)	$W_{i,i}(1)\uparrow(W_n(0), W_n(1), R_{i,i}(1))$
	↑:0(a>v)	$W_{i,i}(1)\downarrow(W_n(0), W_n(1), R_{i,i}(1))$
	↓:0(a<v)	$W_{i,i}(1)\uparrow(W_n(1), W_n(0), R_{i,i}(1))$
	↓:0(a>v)	$W_{i,i}(1)\downarrow(W_n(1), W_n(0), R_{i,i}(1))$
	↑:1(a<v)	$W_{i,i}(0)\uparrow(W_n(0), W_n(1), R_{i,i}(0))$
	↑:1(a>v)	$W_{i,i}(0)\downarrow(W_n(0), W_n(1), R_{i,i}(0))$
	↓:1(a<v)	$W_{i,i}(0)\uparrow(W_n(1), W_n(0), R_{i,i}(0))$
	↓:1(a>v)	$W_{i,i}(0)\downarrow(W_n(1), W_n(0), R_{i,i}(0))$

5. March C- 알고리듬의 경우 진단 과정의 예

4장에서는 테스트에 대한 정보가 없는 경우의 진단 알고리듬을 제시하고 테스트 알고리듬을 고려한 진단 방법에 대해서도 알아보았다. 이 장에서는 메모리 테스트에서 널리 사용되는 march C- 알고리듬을 예로 들어 마치 알고리듬이 테스트에 사용되었을 경우 진단 방법에 대해 알아본다. march C- 알고리듬에서 고장이 없을 경우와 고장이 존재하는 경우 또 그 고장이 어떠한 종류의 고장 모델인지에 따라서 알고리듬에서 읽기 동작을 수행하였을 때 나타나는 결과는 다음 표 7와 같다. 표 7의 고장들 중 결합 고장에서 표기되는 a는 결합원셀을 나타내고 v는 피결합셀을 나타낸다. 또 표에서 짙은 색으로 표시된 경우가 고장이 발생하여 테스트 알고리듬을 수행 중에 기대값과 다른 값이 읽게 되는 경우를 나타낸다. 예를 들어 테스트 중에 M1의 R0에서 고장이 발생한다면 표 5로부터 SA1이나 ↑:1 고장임을 알 수 있다. 하지만 테스트에서 M0, M1, M2를 통과하고 M3에서 고장이 발생하여 중지된다면 하강 천이 고장혹은↑:1이나 ↓:1의 고장으로 예측된다. 물론 M3에서도 SA1을 검출할 수 있지만 이러한 고장이 만약 존재한다면 이전의 마치 요소 M0에서 검출될 것이므로 M3에서 최초로 검출되는 고장은 SA1이 아닌 하강 천이 고장이나 결합 고장으로 분석이 가능하다. 이와 유사하게 표 5와 표 7로부터 march C- 알고리듬에 대해 각각의 마치 요소에서 최초로 검출되는 고장들은 다음의 표 8의 결과와 같이 분석할 수 있다.

표 7. March C- 알고리듬에서 고장에 따른 각각의 마치 요소에서의 읽기 결과
Table 7. Read result of march element for each fault in case of March C- algorithm

	⇕(W0) M0	⇑(R0,W1) M1	⇑(R1,W0) M2	⇓(R0,W1) M3	⇓(R1,W0) M4	⇕(R0) M5
Defect -free		0	1	0	1	0
SA0		0	0	0	0	0
SA1		1	1	1	1	1
TF↑		0	0	0	0	0
TF↓		0	1	1	1	1
↑:1(a<v)		1	1	0	1	0
↑:1(a>v)		0	1	1	1	0
↓:1(a<v)		0	1	0	1	1
↓:1(a>v)		0	1	1	1	0
↑:0(a<v)		0	1	0	0	0
↑:0(a>v)		0	0	0	1	0
↓:0(a<v)		0	0	0	1	0
↓:0(a>v)		0	1	0	0	0

표 8. 각각의 마치 요소에서 최초로 검출되는 고장들
Table 8. The faults detected first in each march element

마치 요소	고장 모델
M1	SA1, ↑:1(a<v)
M2	SA0, TF↑, ↑:0(a>v), ↓:0(a<v)
M3	TF↓, ↑:1(a>v), ↓:1(a>v)
M4	↑:0(a<v), ↓:0(a>v)
M5	↓:1(a<v)

예를 들어 테스트 중에 M4에서 고장이 검출될 경우 이 고장은 ↑:0(a<v), ↓:0(a>v) 인데 이러한 고장이 존재하는 경우에는 고장에 관련된 두 셀 모두를 찾아내야 하므로 결합원셀의 위치를 확인하기 위해 진단 패턴을 가해 주어야 한다.

또 표 5와 표 7로부터 마치 요소를 분석해 보면 결합 고장의 경우에는 각각의 경우에 따라 결합원셀의 어드레스와 피결합셀의 어드레스 중 어느 것이 상위인지를 알 수 있다. 이를 이용하여 진단 과정 중 결합원셀의 위치를 확인하기 위해 패턴을 가할 때 어드레스를 하위에서 증가시킬 것인지 상위에서부터 감소시키며 패턴을 가할지를 효율적으로 결정할 수 있다. 그러므로 이를 통해 세분화된 고장 모델에 상관없이 일정한 어드레스 시퀀스로 패턴을 가할 때에 비해 결합원셀의 위치를 진단하는데 걸리는 시간을 줄일 수 있다. 예를 들어 M4의 경우에는 ↑:0 혹은 ↓:0 고장이 검출되는데 ↑:0의 경우는 결합원셀의 어드레스가 피결합셀의 어드레스 보다 하위 어드레스인 경우인 ↑:0(a<v)이고 ↓:0의 경우는 결합원셀의 어드레스가 피결합셀의 어드레스 보다 상위 어드레스인 경우인 ↓:0(a>v)이다. 그러므로 테스트 중에 M4에서 고장이 검출되어 진단 과정을 수행하는 과정에서 결합 고장에 대한 패턴을 가할 때 ↑:0의 경우에는 셀 (i, j)보다 하위 어드레스에 대해서만 수행하면 되고 ↓:0의 경우에는 셀 (i, j)보다 상위 어드레스 셀들에 대해서만 패턴을 가해 주면 된다. 따라서 $W_{i,j}(1)$ ↑($W_n(0)$, $W_n(1)$, $R_{i,j}(1)$) ↓($W_n(0)$, $R_{i,j}(1)$)을 통해 ↑:0과 ↓:0 고장이 존재할 경우 결합원셀을 진단할 수 있다.

그러므로 결합원셀과 피결합셀의 어드레스를 고려하여 메모리 테스트 알고리듬이 고장이 발생하는 셀을 검출하는 순간부터 진단 과정은 각각의 마치 요소에서 다음의 표 9와 같은 패턴을 사용하여 수행된다. 표 9에 나타난 마치 요소에 따른 변형된 진단 패턴은 3장의 진단 패턴들을 각각의 마치 요소에서 검출되는 고장에 따라 세분화한 것이다. 따라서 테스트에서 고장이 검출될 경우 이에 따른 패턴들을 가하며 진단 과정을 수행하여야 한다. 이를 바탕으로 march C-를 이용한 테스트의 M4에서 고장이 검출될 경우 수행되는 진단 과정을 예를 들어 나타내면 그림 2와 같다.

표 9. 마치 요소에 따른 간략화된 진단 패턴
Table 9. Simplified diagnosis patterns for each march element

마치 요소	검출되는 고장	변형된 진단 패턴
M1	SA1	$W_{i,j}(0)$, $R_{i,j}(0)$
	↑:1(a<v)	$W_{i,j}(0)$ ↑($W_n(0)$, $W_n(1)$, $R_{i,j}(0)$)
M2	SA0, TF↑,	$W_{i,j}(0)$, $R_{i,j}(0)$, $W_{i,j}(1)$, $R_{i,j}(1)$
	↑:0(a>v), ↓:0(a<v)	$W_{i,j}(1)$ ↓($W_n(0)$, $W_n(1)$, $R_{i,i}(1)$) ↑($W_n(0)$, $R_{i,i}(1)$)
M3	TF↓	$W_{i,j}(1)$, $R_{i,j}(1)$, $W_{i,j}(0)$, $R_{i,j}(0)$
	↑:1(a>v), ↓:1(a>v)	$W_{i,j}(0)$ ↓($W_n(0)$, $W_n(1)$, $R_{i,j}(0)$, $W_n(0)$, $R_{i,i}(0)$)
M4	↑:0(a<v), ↓:0(a>v)	$W_{i,j}(1)$ ↑($W_n(0)$, $W_n(1)$, $R_{i,i}(1)$) ↓($W_n(0)$, $R_{i,i}(1)$)
M5	↓:1(a<v)	$W_{i,j}(0)$ ↑($W_n(1)$, $W_n(0)$, $R_{i,j}(0)$)

테스트 중에 M4에서 고장이 검출될 경우 고착 고장이나 천이 고장과 같은 단일 셀 고장이 아니므로 그림 2는 4장에서 설명한 그림 1에서의 단일 셀 고장에 대한 PTN-ST의 과정은 생략하게 됨을 알 수 있다. 그리고 그림 2의 간략화된 진단 과정의 예에서 결합 고장에 대한 PTN-C의 과정에서 사용하는 패턴은 3장에서 설명한 $W_{i,j}(1)$↕($W_n(0)$, $W_n(1)$, $R_{i,j}(1)$, $W_n(0)$, $R_{i,j}(1)$)와 $W_{i,j}(0)$↕($W_n(0)$, $W_n(1)$, $R_{i,j}(0)$, $W_n(0)$, $R_{i,j}(0)$)이 아닌 $W_{i,j}(1)$↑($W_n(0)$, $W_n(1)$, $R_{i,j}(1)$)↓($W_n(0)$, $R_{i,j}(1)$)와 같이 변화하게 된다. 따라서 테스트 중에 M4에서 고장이 검출될 경우에는 최종적으로 그림 2와 같은 진단 과정을 수행하게 된다. 그러므로 일반적인 테스트 알고리듬에서 고장이 검출되는 마치 요소에 따라서 변형된 진단 패턴을 알 수 있으며, 이에 따라 PTN-ST, PTN-C, PTN-N과 같은 과정들도 변형되어 수행되며 결과적으로 그림 1의 진단 알고리듬 또한 테스트의 결과를 고려하여 변형되어 수행된다.

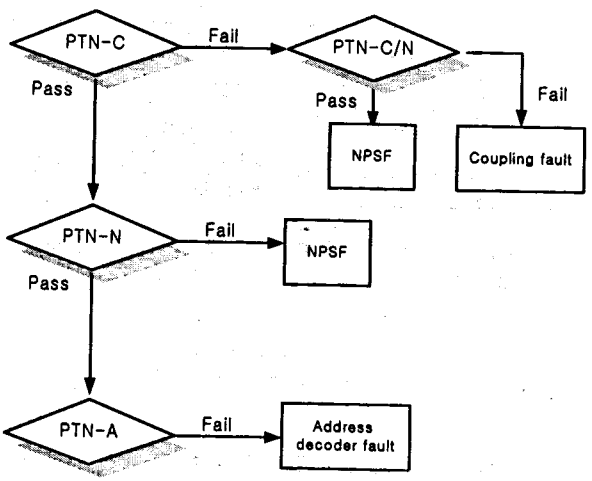

그림 2. 간략화된 진단 과정의 예
: M4에서 고장이 검출된 경우
Fig 2. Example of the simplified diagnosis procedure
: In case of fault detection in M4

6. 진단 과정의 BIST 구현

앞에서는 메모리에서 효율적인 고장 진단을 위한 알고리듬을 제시하였고 메모리 테스트 알고리듬의 수행 결과를 이용하여 진단 알고리듬을 목적에 맞게 보다 간략하게 수정하여 적용하는 과정에 대하여 설명하였다. 본 장에서는 메모리에서의 진단 과정을 내장된 메모리를 테스트하기 위해 널리 사용되는 BIST로 구현할 수 있는 방법을 설명하고 이를 위한 BIST 구조를 제안한다. 진단 과정을 BIST로 구현하기 위해서는 우선 테스트 과정을 통해 고장이 검출될 경우 진단 과정을 수행하여 고장이 발생한 셀을 확인할 수 있는 BIST 구조가 필요하다. 이를 위해 제안된 BIST 구조는 그림 3과 같으며 짙은 색으로 표시된 부분이 메모리에서의 고장 진단을 위해 스캔 경로를 형성하게 되는 부분을 의미한다.

제안된 BIST 구조에서는 테스트 도중에 고장이 검출되는 순간 테스트는 중지되고 스캔 경로가 활성화되어 고장 셀 어드레스 등의 정보를 스캔 아웃함으로써 메모리에서의 고장을 진단한다. 또한 일반적인 하나의 스캔 경로가 아닌 두 개의 스캔 경로를 사용함으로서 효율적으로 진단을 수행할 수 있

도록 한다. Scan out(A)를 통해서는 고장이 검출되는 경우의 어드레스가 출력되므로 고장셀의 위치를 확인할 수 있고 또 다른 스캔 경로에서의 scan out(B)를 통해서는 고장이 있을 경우의 메모리의 데이터가 출력되고 셀의 고착 상태를 알 수 있다. 여기서 제안된 BIST 구조는 [5]에서 제안된 구조와는 다르게 두 개의 스캔 경로를 사용하고 병렬적으로 고장 진단을 수행할 수 있으므로 어드레스와 데이터값이 각각의 경로를 통해 출력되므로 이에 소요되는 시간을 감소시켜서 보다 효율적인 진단을 가능하게 한다.

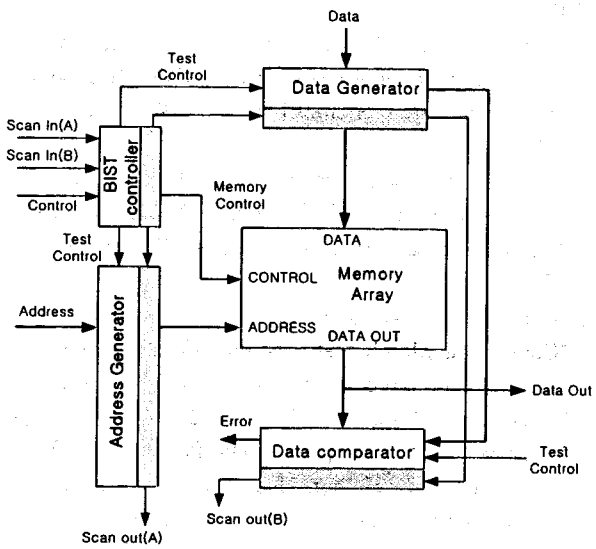

그림 3. 메모리 진단을 위한 BIST 아키텍쳐

Fig 3. BIST architecture for memory diagnosis

7. 성능 평가 비교 및 결과

본 논문에서 제안된 진단 알고리듬을 기존의 제안된 진단에 관한 방법들과 비교하면 표 11과 같다. [12]의 방법은 메모리의 진단을 위한 별도의 패턴을 사용하지 않고 테스트 알고리듬의 수행 결과 분석을 통해 고장의 종류를 분석한다. 메모리 테스트 알고리듬에서 각각의 고장 모델이 존재할 경우에 따른 각각의 마치 요소에서의 고장 검출 여부에 대한 결과를 표를 통해 작성하고 이를 테스트 결과와 비교하여 메모리의 셀에서 고장이 존재할 경우 각 셀에서의 고장의 종류를 분석한다. 이러한 방법의 경우 테스트 알고리듬이 검출하는 다양한 종류의 고장들의 종류를 진단할 수 있지만 고장이 검출되는 셀만을 확인할 뿐 고장과 관련된 다른 셀들을 찾아내지 못한다는 단점이 있다. 또 [13]에서 제안된 방법은 제안된 초기 마치 테스트를 수행하여 단일 셀 고장인지 다중 셀 고장인지를 구별하고 결합원셀을 찾기 위한 패턴을 사용한다. 따라서 피결합셀외에 결합원셀을 찾아낼 수 있다. 하지만 이러한 방법의 경우에는 진단 과정의 수행을 위해 반드시 초기 마치 테스트를 수행하는 과정이 포함되어야 하며 NPSF에 대한 진단이 이루어지지 않는다는 단점이 있다. 그리고 본 논문에서 제안된 방법에서는 대부분의 고장 모델의 종류 분석 가능하고 고장이 검출되는 셀 뿐만 아니라 고장의 원인이 될 수 있는 셀과 같은 고장과 관련된 모든 셀의 확인이 가능하다. 또 일반적인 진단 과정 뿐 아니라 메모리 테스트의 결과 분석을 고려한 진단 패턴의 변형 및 진단 방법을 제안하였으며 이러한 경우의 진단 과정에서는 세분화된 고장 모델에 따른 효율적인 어드레스 시퀀스를 사용하였다. 본 논문에서 제안된 전체 진단 과정의 길이는 $15n+444$와 같다. 그리고 워드 본위 메모리의 경우 이와 더불어 하나의 워드내의 셀들 간의 결합 고장의 검출을 위해서는 $6*n/b*\log_2 b$만큼의 동작이 추가로 수행되어야 한다. 따라서 워드 본위 메모리의 경우 전체 진단 과정의 길이는 $(15+6*\log_2 b)*n/b+444$와 같다.

표 10. 다양한 진단 방법들의 비교

Table 10. Comparison of the various diagnosis methods

방법들	[12]	[13]	제안된 방법
고장 모델	고착 고장, 천이 고장, 결합 고장	고착 고장, 천이 고장, 어드레스 디코더 고장, 결합 고장	고착 고장, 천이 고장, 어드레스 디코더 고장, 결합 고장, NPSF
어드레스 시퀀스	세분화된 어드레스 시퀀스	일정한 시퀀스 사용	세분화된 어드레스 시퀀스
진단 범위	고장 검출 셀	고장 관련 셀 포함	고장 관련 셀 포함
제한 요소	고장 관련 셀을 포함하지 못함	초기 마치 테스트의 수행 NPSF를 진단하지 못함	진단 소요 시간 증가

8. 결 론

본 논문에서는 고집적 메모리에서의 효율적인 고장 진단을 위한 알고리듬을 제시하였다. 우선 메모리에서 가장 일반적으로 사용되는 고장 모델들을 고려하고 이에 근거하여 각각의 고장들에 대한 진단을 위한 패턴들을 분석하였다. 또 이를 바탕으로 테스트에서 고장이 검출될 경우 고장의 종류를 구별하는 것을 가능하게 하는 진단 방법을 제시하고 단지 테스트에서 고장이 검출되는 셀 뿐만 아니라 고장의 원인이 될 수 있는 셀들을 포함하여 고장과 관련된 모든 셀들을 확인할 수 있게 하였다. 또한 테스트에 대한 정보가 없는 일반적인 경우의 진단 알고리듬 뿐 아니라 메모리 테스트의 각 셀에서의 수행 결과를 이용하여 적합한 진단 패턴을 결정하고 이에 따라 효율적인 진단 과정을 수행하는 방법을 제시함으로서 메모리 테스트를 고려한 진단 과정을 가능하게 하였다. 그리고 테스트에서 고장이 검출된 셀 뿐 만 아니라 고장과 관련된 셀들의 위치를 찾기 위한 진단 패턴을 일정한 어드레스 시퀀스를 사용하는 것이 아니라 고장이 검출된 셀과 고장과 관련된 다른 셀의 어드레스의 관계에 의한 세분화된 고장의 종류에 따라 동작수를 줄이는 것이 가능한 어드레스 시퀀스를 결정하여 사용함으로서 진단 과정에 소요되는 시간을 효율적으로 줄일 수 있게 하였다. 또한 내장된 메모리에서의 효율적인 고장 진단을 위한 새로운 BIST 구조를 개발하였다. 본 논문에서 제안된 메모리에서의 고장 진단 알고리듬을 통

해 메모리에서 발생하는 거의 모든 종류의 고장들에 대한 진단이 가능하고 이를 통해 고장이 검출된 메모리의 효율적인 재생산을 통해 수율을 향상시키는데 도움이 될 뿐 아니라 고장의 발생 특성을 분석하는 것을 가능하게 한다.

참 고 문 헌

[1] M. F. Chang, W. K. Fuchs, J. H. Patel, "Diagnosis and repair of memory with coupling faults," computers, IEEE Transactions on, volume.38, No.4, April 1989, Page(s): 493-500.

[2] Hayes, J. P., "Pseudo-Boolean Logic Circuits", IEEE Trans. Computers, vol. C-35, no. 7, pp. 602-612, July 1988.

[2] J. Otterstedt, D. Niggemeyer, T. W. Williams, "Detection of CMOS address decoder open faults with March and pseudo random memory tests," Test Conference, 1998. Proceedings., International, 1998, Page(s): 53 -62

[3] Sying-Jyan Wang, Chen-Jung Wei, "Efficient built-in self-test algorithm for memory," Asian Test Symposium, 2000. (ATS 2000). Proceedings of the Ninth, 2000 Page(s): 66 -70

[4] Lin Shen, B. F. Cockburn, "An optimal march test for locating faults in DRAMs," Memory Testing, 1993., Records of the 1993 IEEE International Workshop on, 1993, Page(s): 61-66.

[5] Chin Tsung Mo, Chung Len Lee, Wen Ching Wu, "A self-diagnostic BIST memory design scheme," Memory Technology, Design and Testing, 1994., Records of the IEEE International Workshop on, 1994, Page(s): 7-9.

[6] A. J. van de Goor, Testing Semiconductor Memories: Theory and practice, J. Wiley & Sons, 1991.

[7] Chih-Tsun Huang, Jing-Reng Huang, Cheng-Wen Wu, "A programmable built-in self-test core for embedded memories," Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific, 2000, Page(s): 11 -12

[8] M. Azimane, A. L. Ruiz, "New short and efficient algorithm for testing random-access memories," Electronics, Circuits and Systems, 1998 IEEE International Conference on , Volume: 1, 1998, Page(s): 541 - 544

[9] V. Kim, T. Chen, "Assessing defect coverage of memory testing algorithms," VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on, 1999, Page(s): 340 -341

[10] A. J. van de Goor, I. B. S. Tlili, "March tests for word-oriented memories," Design, Automation and Test in Europe, 1998., Proceedings, 1998, Page(s): 501 -508

[11] A. J. van de Goor, I. B. S. Tlili, S. Hamdioui, "Converting March tests for bit-oriented memories into tests for word-oriented memories," Memory Technology, Design and Testing, 1998. Proceedings. International Workshop on, 1998, Page(s): 46-52

[12] C. F. Wu, C. T. Huang, "Error catch and analysis for semiconductor memories using march tests," Computer-Aided Design, 2000. ICCAD 2000. Digest of Technical Papers. 2000 IEEE/ACM International Conference on, 2000, Page(s) : 468-471

[13] T. J. Bergfeld, D. Niggemeyer, E. M. Rudnick, "Diagnostic testing of embedded memories using BIST", Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings, 2000, Page(s): 305-309

저 자 소 개

박 한 원 (朴 漢 源)

1976년 4월 18일 생. 1999년 연대 전기공학과 졸업. 1999년~현재 연대 전기전자공학과 석사과정

E-mail : hwpark@dopey.yonsei.ac.kr

강 성 호 (姜 成 昊)

1963년 4월 13일 생. 1986년 2월 서울대 공대 제어계측공학과 졸업. 1988년 5월 The University of Texas at Austin 전기 및 컴퓨터공학과 졸업 (석사). 1992년 5월 The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(공박). 미국 Schlumberger연구원. Motorola 선임연구원. 현재 연대 공과대학 전기전자공학과 부교수.

Tel : 02-2123-2775, Fax : 02-313-8053

E-mail : shkang@yonsei.ac.kr