論文2002-39SD-11-6

이웃 패턴 감응 고장을 위한 효과적인 메모리 테스트 알고리듬

(An Effective Memory Test Algorithm for Detecting NPSFs)

徐 一 碩 *, 康 容 碩 **, 姜 成 昊 ***

(Ilseok Suh, Yong-Seok Kang, and Sungho Kang)

요 약

반도체 기술의 발달로 인하여 메모리가 고집적화 됨에 따라 테스트의 복잡도와 시간도 같이 늘어나게 되었다. 실제로 널리 쓰이는 메모리 테스트 방법인 March 알고리듬은 DRAM에서 발생되는 고장을 검출하기 위해 고안된 것이다. 그러나 DRAM의 집적도가 증가함으로 반드시 고려해야 하는 이웃 패턴 감응 고장을 기존의 March 알고리듬으로는 테스트할 수 없고 DRAM의 이웃 패턴 감응 고장을 테스트하기 위한 기존 알고리듬들은 메모리 셀의 개수를 n이라고 할 때 $O(n^2)$의 복잡도를 갖기 때문에 테스트 시간을 많이 소요하게 된다. 본 논문에서는 메모리 테스트에 많이 쓰이는 March 알고리듬을 확장하여 메모리의 이웃 패턴 감응 고장 검출율을 효과적으로 높일 수 있는 알고리듬을 제안하였다.

Abstract

Since memory technology has been developed fast, test complexity and test time have been increased simultaneously. In practice, March algorithms are used widely for detecting various faults. However, March algorithms cannot detect NPSFs(Neighborhood Pattern Sensitive Faults) which must be considered for DRAMs. This paper proposes an effective algorithm for high fault coverage by modifying the conventional March algorithms.

Key Words : NPSF, Memory test, March algorithm

Ⅰ. 서 론

반도체 메모리에 있어서의 지속적인 발전은 메모리 칩의 저장용량을 꾸준히 증가시켜왔다. 메모리 발전과정을 보면 대체로 3년마다 용량이 4배로 증가해 왔다. 이러한 반도체 제조기술의 발달은 더욱 작고 복잡한 회로의 구현을 가능하게 하여 단위면적당 메모리셀(cell)의 수를 증가시켜 64K DRAM에서 64M DRAM까지 세대마다 평균 약 60%이상으로 셀의 크기가 축소되었다. 이러한 회로 크기의 소형화는 보다 정밀한 제조공정을 요구하고 이에 따른 제조 비용 및 테스트 비용이 급격히 상승한다. 또한 메모리의 집적도가 증가함에 따라 메모리 셀들은 더욱 가깝게 위치하게 되고 이로 말미암아 이웃 셀의 영향에 더욱 민감하게 작용하여 주소(address)나 데이터선의 잡신호에 의한 방해를 더욱 잘 받게 되었다.[1] 이러한 주위셀의 영향으로 발생한 고장을 패턴 감응고장(PSF: Pattern Sensitive Fault)이라 하고 인접한 셀의 영향만을 고려한 것을 이

* 正會員, 三星電子시스템 LSI事業部
(Samsung Electronics System LSI Division)
** 正會員, LG電子시스템 IC事業部
(LG Electronics System IC Division)
*** 正會員, 延世大學校 電氣電子工學科
(Dept. of Electrical & Electronic Engineering Yonsei Univ.)

接受日字:2001年6月21日, 수정완료일:2002年10月24日

웃 패턴 감응 고장(NPSF: Neighborhood Pattern Sensitive Fault)이라 한다.[2] 이웃 패턴 감응 고장은 DRAM에서 고려해야 하는 중요한 고장 모델이고 메모리가 고집적화 됨에 따라 SRAM에서도 역시 중요한 고장 모델이 되고 있다.[3] 기존의 메모리 테스트 알고리듬들은 한 개의 셀의 값을 변화시킨 후에 그 셀을 읽던가 혹은 값을 변화시킨 후에 전체 메모리를 읽는 동작을 기본으로 구성된다. 이러한 알고리듬으로 0-1 알고리듬이나 체커보드 알고리듬, GALPAT 알고리듬, March 알고리듬을 예로 들 수 있다. 그 중 March 알고리듬은 가장 널리 이용되는 메모리 테스트 알고리듬이다. 기존의 메모리 테스트 알고리듬은 이웃 패턴 감응 고장이 고려되지 않았기 때문에 이웃 패턴 감응 고장을 검출하기 위한 다른 알고리듬이 필요하게 되었고 이웃 패턴 감응 고장을 검출할 수 있는 알고리듬이 많이 제안되었다.[4~6]

이웃 패턴 감응 고장을 검출하는 알고리듬의 대표적인 방법은 참이웃 셀에 해밀토니언 시퀀스나 오일리언 시퀀스를 가하고 기준 셀을 읽는 타일링 방법이나 2-그룹 방법이 있다. 1형 타일링 방법은 메모리 셀을 중첩되지 않는 이웃들의 그룹으로 분리하여 테스트를 수행하는 방법이고 2-그룹 방법은 메모리 셀을 기준셀들 그룹과 참이웃셀 그룹으로 두 개의 체커보드 형태의 타일링 그룹으로 분리하여 테스트를 수행한다. 이 방법들은 고장 검출율은 매우 높으나 알고리듬의 복잡도가 크고 또한 이웃 패턴감응 고장을 제외한 다른 고장 테스트도 병행해야 하므로 매우 긴 테스트 시간을 요한다. 테스트 비용과 시간을 줄이기 위한 방법으로 한 개의 알고리듬으로 이웃 패턴 고장을 포함한 모든 고장을 검출하는 테스트 방법들이 제안되었는데 대표적으로 의사임의 테스트와 널리 사용되는 March 알고리듬을 이웃 패턴 감응 고장을 검출할 수 있도록 확장하여 테스트 하는 방법 등이 있다. 의사임의 테스트는 메모리 셀의 주소나 셀에 쓰는 값(0,1) 등을 의사임의 패턴 생성기를 이용하여 테스트한다.[7] 그러나 기존의 알고리듬을 그대로 사용하면서 이웃 패턴 감응 고장을 추가로 고려[8]하기 때문에 이웃 패턴 감응 고장을 위한 다양한 패턴을 생성하지 못하므로 낮은 검출율을 가진다. March 알고리듬을 확장하는 방법은 워드단위(word-oriented) 메모리에서 워드 단위의 테스트 패턴을 이웃 패턴 감응 고장을 검출할 수 있도록 변형하는 방법[9]과 기존 March 알고리즘을 확장하는 방법[10]이 있다. 그러나 이 방법 역시 테스트 시간은 감소하나 이웃 패턴 감응 고장 검출율이 낮다. [11]에서는 기본 알고리즘($12n$)에 백그라운드 데이터(8가지)를 가하며 베이스셀을 읽는 알고리듬이 제안되었다. 그러나 이웃 패턴 감응 고장의 특성으로 인해 높은 고장 검출율을 얻기 위해서 테스트 복잡도가 증가하였다.

따라서 본 논문에서는 이웃 패턴 감응 고장 검출율을 효과적으로 높이면서 테스트 시간을 줄이는 March 알고리듬의 확장 방법을 제안한다.

Ⅱ. 고장 모델

고장 모델이란 예상되는 결함들에 의한 전체 혹은 특정 부분의 동작 형태를 충분히 표현할 수 있도록 가정된 고장들의 집합을 말한다. 테스트 시간의 효율적 활용을 목적으로 하는 경우 가장 발생할 확률이 높은 결함에 대해 높은 검출율을 갖는 테스트 입력을 생성하기 위해서 고장 모델들이 사용된다. 메모리의 기능적 모델은 어드레스 래치(latch), 행 디코더, 열 디코더, 메모리 셀 어레이, 감지 증폭기(sense amplifier), 쓰기 증폭기등 많은 블록들로 구성되어 진다. 비록 이들 각 블록들이 각각 결함을 가질 수 있지만, 각 블록에서의 고장들이 같은 고장 동작을 나타낼 수 있다. 이와 같은 이유로 인해 고장 모델링을 목적으로 하는 경우, 메모리의 기능적 모델은 축소된 기능적 블록들로 간소화될 수 있다. 이 축소된 모델은 어드레스 디코더, 메모리 셀 어레이, 그리고 읽기/쓰기 블록으로 나타낼 수 있다.

1. 고착 고장(Stuck-at fault) ; SAF

고착 고장은 하나 또는 그 이상의 메모리 셀 또는 선의 항상 0 또는 1로 고착된 상태로 그 값이 변하지 않는 고장으로 각각을 0-고착 고장(SA0 고장)과 1-고착 고장(SA1 고장)라고 한다.

2. 천이 고장(Transition fault) ; TF

천이 고장은 고착 고장의 특수한 경우라 할 수 있다. 이는 어떤 하나의 셀 또는 선의 천이 동작이 이루어지지 않는 고장으로서 0→1 천이가 이루어지지 않는 경우 상승 천이 고장이라고 하고 이와 유사하게 1→0 천이를 할 수 없는 고장을 하강 천이 고장이라고 한다.

3. 결합 고장(Coupling fault) ; CF

결합 고장은 2개 또는 그 이상의 셀 사이에서 셀들의 비정상적인 연결로 인해 발생하는 고장으로 관련된

셀의 수가 k일 때 k 결합 고장이라 한다. 이때, 영향을 주는 셀을 결합셀이라 하고 영향을 받는 셀을 피결합셀이라고 한다. 이 고장을 세분화하면 다음과 같다. 반전 결합 고장(Inversion coupling fault; CFin) 모델은 하나의 셀에서 전이 쓰기 동작을 할 때 결합된 두 번째 셀이 원래 내용과 반대로 바뀌는 고장이고 상태 결합 고장(State coupling fault; CFst)은 두 개의 셀 중 한 셀 j가 특정 값 y를 갖고 있을 때 다른 셀 i가 어떠한 값 x를 갖게 되는 것이다. 동행 결합 고장(Idempotent coupling fault; CFid)은 하나의 셀에서 전이 쓰기 동작을 할 때 결합된 두 번째 셀의 내용이 강제로 0 또는 1로 바뀌는 형태의 고장이다. 두 개 이상의 결합 고장이 똑같은 피결합셀에 존재하거나 다른 고장이 연계된 경우를 연계 결합 고장(Linked coupling fault; Linked CF)이라고 있다.

4. 패턴 감응 고장(Pattern sensitive fault) ; PSF

패턴 감응 고장(PSF) 특정한 셀의 값이나 전이가 메모리의 다른 모든 셀이 갖는 값들에 의해 영향을 받을 때 발생하게 된다. 그러나 모든 메모리 셀의 패턴을 고려하는 것은 비현실적이므로 이웃 패턴 감응 고장(NPSF) 모델을 사용한다. 이 축소 모델은 패턴 감응 고장이 물리적인 이웃(neighborhood)에서 발생하는 누설 전류에 의해서 발생한다는 가정에 의해서 타당성을 갖게 된다.[1] <그림 1>에 나타낸 것과 같이 이웃 셀은 검사 대상이 되는 기준셀(base cell)과 기준셀을 제외한 나머지 부분인 참이웃셀(deleted neighborhood)로 구성되어 있다. 참이웃셀의 수에 따라 1형과 2형으로 분류되고 고장 발생 원인에 따라 아래와 같이 3가지 종류로 나눌 수 있다.

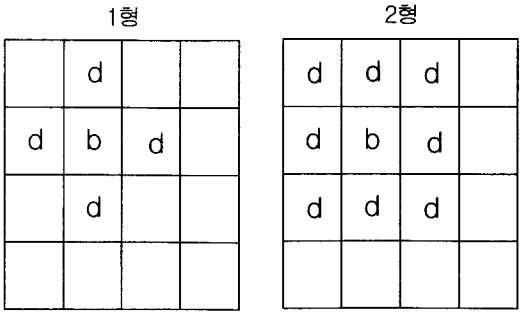


그림 1. 이웃 패턴 감응 고장의 2가지 분류
Fig. 1. Two types of NPSFs.

가. 정적 이웃 패턴 감응 고장(SNPSF) : 기준셀이 주위 셀의 특정한 패턴에 의해 0이나 1의 값으로 고정되는 고장

나. 수동 이웃 패턴 감응 고장(PNPSF) : 제외된 이웃의 특정한 패턴이 기준셀의 내용이 바뀌어지는 것을 방해하는 고장

다. 능동 이웃 패턴 감응 고장(ANPSF) : 제외된 이웃의 한 셀에서 발생하는 전이 동작에 의하여 기준셀의 값이 다른 값으로 바뀌어지는 고장

Ⅲ. 새로운 이웃 패턴 감응 고장 테스트 알고리듬

이웃 패턴 감응 고장을 고려하지 않은 기존 메모리 테스트에 가장 널리 쓰이는 방법은 March 알고리듬이다. <그림 2>는 March 알고리듬으로 메모리를 테스트하는 과정을 보여준다. 모든 메모리 셀을 0으로 초기화하고 순차적으로 어드레스를 증가시키면서 셀에 1을 쓰는 동작이며 셀에 로직 값을 쓰고 바로 그 셀을 읽는 방식과 모든 셀에 로직 값을 쓴 후에 어드레스를 변화시키면서 읽는 방식이 있다. <그림 2>의 March 알고리듬은 셀에 로직 값을 쓰고 바로 그 셀을 읽는 방식이라고 가정한다.

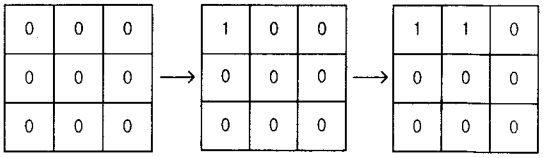


그림 2. March 알고리듬 진행 과정(1)
Fig. 2. Procedure of March algorithm(1).

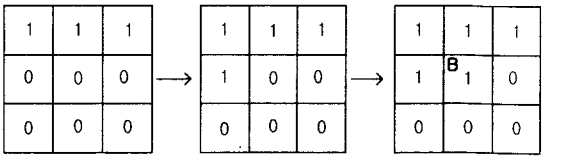


그림 3. March 알고리듬의 진행 과정(2)
Fig. 3. Procedure of March algorithm(2).

<그림 3>은 <그림 2>의 March 알고리듬이 계속 진행되는 과정을 나타내었다. 여기서 B셀에 1을 쓰고 그 셀을 읽는 다면 상승 천이 고장과 0-고착 고장을 검출할 수 있다. 만약 이웃 패턴 감응 고장을 고려한다면 1개의 수동 이웃 패턴 감응 고장과 1개의 정적 이웃 패

턴 감응 고장을 검출할 수 있다. 그러나 수동 이웃 패턴 감응 고장과 정적 이웃 패턴 감응 고장의 수가 각각 한 셀당 $2k$(k: 이웃 셀 수)개인 점을 고려하면 극히 낮은 검출율을 가진다.

따라서 기존의 March 알고리듬은 이웃 패턴 감응 고장을 검출할 수 없으므로 이웃 패턴 감응 고장을 검출할 수 있도록 March 알고리듬을 확장하는 방법을 사용한다. 이 방법은 기존의 March 알고리듬보다 테스트 시간은 증가하나 추가적인 면적 오버헤드 없이 이웃 패턴 감응 고장을 고려할 수 있는 장점이 있다. 예를 들어 Yarmolik[9]의 March PS도 March 알고리듬을 확장하는 방법 중의 하나이며 알고리듬은 <그림 4>와 같다. 이 방법은 테스트 시간을 많이 감소시킬 수 있으나 능동 이웃 패턴 감응 고장이 고려되지 않았고 또한 나머지 이웃 패턴 감응 고장 검출율도 그리 높지 않다. March PS 알고리듬은 메모리 셀에 로직 값을 쓰고 그 셀을 읽으므로 참이웃셀의 천이로 야기되는 능동 이웃 패턴 감응 고장은 검출하지 못한다.

↕(w0); ↑(r0,w1,r1,w0,r0,w1); ↑(r1,w0,r0,w1,r1); ↑(r1,w0,r0,w1,r1,w0), ↑(r0,w1,r1,w0,r0)

그림 4. March PS(23 n) 알고리듬
Fig. 4. March PS(23 n) Algorithm.

총 이웃 패턴 감응 고장 개수 중에 능동 이웃 패턴 감응 고장의 개수가 가장 많은 부분을 차지하므로 이 고장을 고려하는 것은 중요하다. 따라서 본 논문은 March PS와는 다른 방법으로 March 알고리듬을 이웃 패턴 감응 고장 검출율을 능동 이웃 패턴 감응 고장 검출율을 효과적으로 높일 수 있는 알고리듬을 다음과 같이 제안한다. March 알고리듬에 쓰기 동작에 이웃셀을 읽는 동작을 추가하는 것이다. 이 알고리듬의 자세한 동작을 <그림 5>에 나타내었다. <그림 5(a)>는 March 알고리듬의 진행과정이 도식적으로 표시되었고 B는 기준셀을, N, W, S, E는 참이웃셀을 나타낸다. <그림 5(b)>에서 March 알고리듬의 동작이 B셀에 1을 쓰는 동작을 하고 그 셀을 읽는다면 한 개의 수동 이웃 패턴 감응 고장과 한 개의 정적 이웃 감응 패턴을 검출할 수 있다. 또한 B셀에 1을 쓰는 동작은 N을 기준 셀로 고려할 때 N셀에 대한 능동 이웃 패턴 감응 고장을 유발한다. 즉 <그림 5(c)>와 같이 B셀에 1을 쓰고 N셀을 읽으면 N셀의 한 개의 능동 이웃 패턴 감응 고장과 한 개의 정적 이웃 패턴 감응 고장이 검출할 수 있다. 이것은 나머지 참이웃셀인 S, W, E셀에서도 <그림 5(d), (e), (f)>와 같이 N셀의 경우와 같은 결과를 얻을 수 있다.

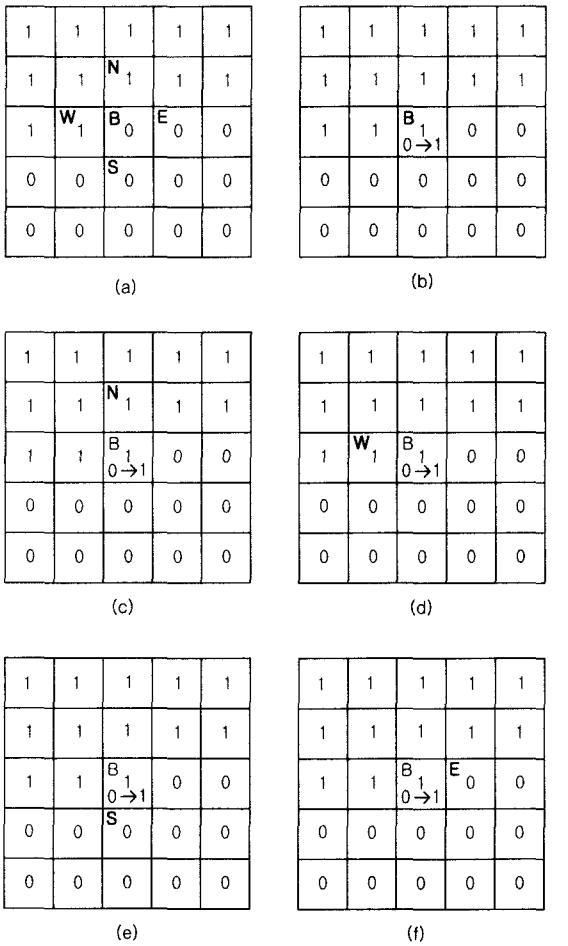


그림 5. 제안된 이웃 패턴 감응 고장 알고리듬
Fig. 5. Proposed Algorithm for NPSFs.

새로운 알고리듬을 사용하여 검출할 수 있는 이웃 패턴 감응 고장을 정리하면 다음과 같다. 여기에서 괄호표기는 (B, N, W, S, E) 순이다.

<그림 5(b)> : B셀을 읽기 동작을 한 경우
PNPSF(01100→11100), SNPSF(11100)

<그림 5(c)> : N셀을 읽기 동작을 한 경우
ANPSF(11110→11111), SNPSF(11111)

<그림 5(d)> : W셀을 읽기 동작을 한 경우
ANPSF(11100→11110), SNPSF(11110)

<그림 5(e)> : S셀을 읽기 동작을 한 경우

ANPSF(01000→01100), SNPSF(01100)

<그림 5(f)> : E셀을 읽기 동작을 한 경우

ANPSF(00000→01000), SNPSF(01000)

제안된 알고리듬은 모든 March 알고리듬에 적용 가능하다. 예를 들어 <그림 6>은 March C-의 알고리듬이다. 이 알고리듬을 제안된 알고리듬으로 적용한 것을 <그림 7>에 나타내었다.

↕(w0), ↑(r0,w1), ↑(r1,w0), ↓(r0,w1), ↓(r1,w0), ↕(r0)

그림 6. March C-(10 n) 알고리듬
Fig. 6. March C-(10 n) Algorithm

↕(w0), ↑(r0,**M1**), ↑(r1,**M0**), ↓(r0,**M1**), ↓(r1,**M0**), ↕(r0)

그림 7. 제안된 March CE-(30 n) 알고리듬
Fig. 7. Proposed March CE-(30 n) Algorithm.

그림 7에 표기된 M은 쓰기 동작과 이웃 셀을 읽는 동작을 하나로 표현한 것이다. 즉 M0은 메모리 셀에 0을 쓰고 이웃 셀을 읽는 동작을 한다. 본 논문에서는 1형 이웃 패턴 감응 고장을 고려했으므로 1n의 쓰기동작마다 5 n의 읽기 동작이 추가된다. 따라서 March C-의 알고리듬은 10 n에서 30 n으로 늘어난다.

본 논문에서 제안하는 새로운 이웃 패턴 감응 고장 테스트 알고리듬은 March 알고리듬의 쓰기 동작마다 5번의 읽기 동작을 추가하여 March 알고리듬으로 검출할 수 없던 이웃 패턴 감응 고장을 검출하게 하는 것이다. 제안된 알고리듬은 March C-알고리듬 이외에 다른 모든 March 알고리듬에 적용 가능하다. 그림 8은 다른 March 알고리듬에 새로운 이웃 패턴 감응 고장 알고리듬을 적용한 것을 나타내었다.

<그림 9(b)>와 같이 March A와 March B와 같이 쓰기 동작이 많은 경우 알고리듬의 모든 쓰기 동작에 이웃 패턴 감응 고장을 고려하는 것은 검출한 고장을 반복하여 검출하는 불필요한 동작을 하므로 테스트 시간이 길어진다. 따라서 테스트 시간을 줄이기 위해서 <그림 9>와 같이 이웃 패턴 감응 고장을 고려한 M 동작을 최소화하였다.

Ⅳ. 결 과

본 논문에서 제안한 알고리듬은 메모리 시뮬레이터를 구현하여 검증하였다. 메모리 시뮬레이터는 Visual

March X	↕(w0), ↑(r0,w1), ↑(r1,W0), ↕(r0)
March XE	↕(w0), ↑(r0,**M1**), ↑(r1,**M0**), ↕(r0)
March Y	↕(w0), ↑(r0,w1,r1), ↓(r1,w0,r0), ↕(r0)
March YE	↕(w0), ↑(r0,**M1**,r1), ↓(r1,**M0**,r0), ↕(r0)
March A	↕(w0), ↑(r0,w1,w0,w1), ↑(r1,w0,w1) ↓(r1,w0,w1,w0), ↓(r0,w1,r0)
March AE	↕(w0), ↑(r0,**M1**,**M0**,**M1**), ↑(r1,**M0**,**M1**) ↓(r1,**M0**,**M1**,**M0**), ↓(r0,**M1**,r0)
March B	↕(w0), ↑(r0,w1,r1,w0,r0,w1), ↑(r1,w0,w1) ↓(r1,w0,w1,w0), ↓(r0,w1,r0)
March BE	↕(w0), ↑(r0,**M1**,r1,**M0**,r0,**M1**), ↑(r1,**M0**,**M1**) ↓(r1,**M0**,**M1**,**M0**), ↓(r0,**M1**,r0)

그림 8. 확장된 March 알고리듬
Fig. 8. Extended March Algorithms.

(a) ↕(w0) ↑(r0,w1,w0,w1) ↑(r1,w0,w1) ↓(r1,w0,w1,w0) ↓(r0,w1,w0)
(b) ↕(w0) ↑(r0,M1,M0,M1) ↑(r1,M0,M1) ↓(r1,M0,M1,M0) ↓(r0,M1,M0)
(c) ↕(w0) ↑(r0,**M1**,w0,w1) ↑(r1,**M0**,w1) ↓(r1,**M0**,w1,w0) ↓(r0,**M1**,w0)

그림 9. March A알고리듬에 대한 제안된 알고리듬
Fig. 9. Proposed Algorithm for March A algorithm.

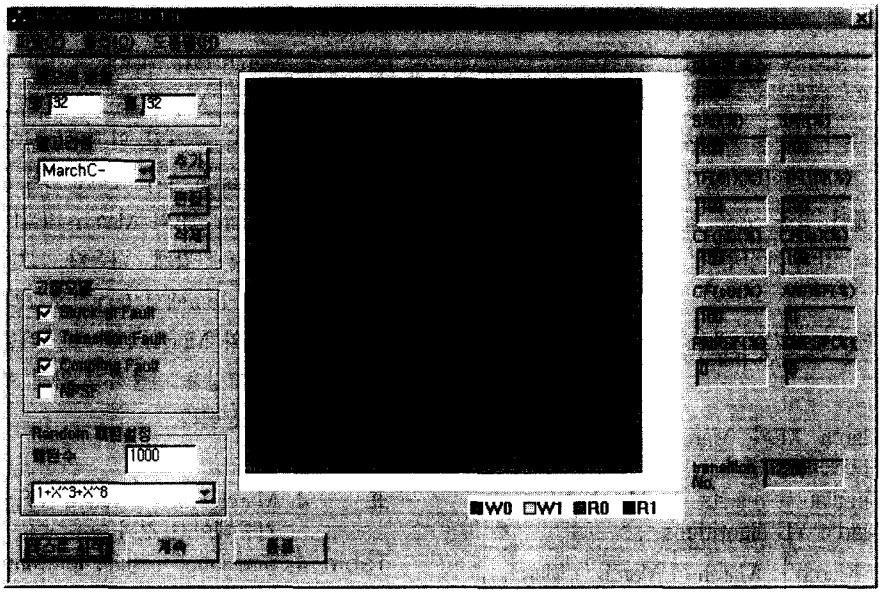


그림 10. 메모리 시뮬레이터
Fig. 10. Memory Simulator.

C++로 작성되었고 <그림 10>에 보이는 것과 같이 시뮬레이터는 메모리의 테스트 과정 즉 읽고 쓰는 동작을 그래픽으로 구현하여 사용자가 테스트 과정을 볼 수 있도록 하였으며 기존의 March 알고리듬 이외에 사용자가 임의의 알고리듬을 구성 가능하게 하였다. 메모리 시뮬레이터의 고장 검출 원리는 고장을 가상으로 삽입하여 검출하는 것이 아니라 가해진 테스트 알고리듬으로 고장 검출율을 구할 수 있도록 하였다. 예를 들어, w0, r0이라는 테스트 패턴이 가해진다면 1-고착 고장을 검출할 수 있다.

<표 1>은 기존의 이웃 패턴 감응 고장을 검출하는 알고리듬을 사용할 때 필요한 셀 당 동작 수를 나타내었다. 타일링 방법이나 2 그룹 방법은 능동, 수동 이웃 패턴 감응 고장을 같이 검출하기 위해서는 매우 긴 테스트 시간을 요한다는 것을 알 수 있다.

본 논문에서 제안한 방법은 모든 March 알고리듬에 적용 가능하다. <표 2>는 March X와 March Y알고리듬에 대한 시뮬레이션 결과를 표시하였다. March XE는 본 논문에서 제안한 확장된 알고리듬을 March X에 적용시킨 알고리듬이다. 제안된 알고리듬을 적용한 명칭은 알고리듬의 뒤에 E를 적어 구별하였다. March Y는 읽기 동작이 March X보다 $2n$ 추가되었기 때문에 March Y의 PNPSF 검출율이 March X의 PNPSF 검출율 보다 3% 증가된다. 그러나 본 논문에서 제안한 알고리듬을 적용하면 <표 2>에 보이는 것과 같이 March XE와 March YE의 검출율이 같다. 이것은 확장된 알고리듬은 쓰기 동작을 하고 바로 기준셀과 이웃셀을 읽으므로 한 요소에 M 동작 후의 읽기 동작은 검

표 1. 이웃 패턴 감응 고장 테스트를 위한 셀 당 동작 수(1형)
Table 1. Number of operations per cell for NPSFs(1 type).

테스트 방식	능동, 수동 이웃 패턴감응 고장	능동 이웃 패턴 감응고장	수동 이웃 패턴 감응고장	정적 이웃 패턴 감응고장
타일링 방법	$194\frac{1}{2}n$	$162\frac{1}{2}n$	$66\frac{1}{2}n$	$39\frac{1}{5}n$
위치검출 2-그룹 방법	$195n$	$163n$	$67n$	$40\frac{7}{8}n$
검출 2-그룹 방법	$165n$	$99n$	-	$36\frac{1}{8}n$

출율에 영향을 끼칠 수기 때문이다. 이웃 패턴 감응 고장의 종류는 ANPSF(능동이웃 패턴감응 고장), PNPSF(수동이웃 패턴감응 고장), SNPSF(정적이웃 패턴감응 고장)로 3가지이다. 따라서 총 이웃 패턴 감응 고장 검출율은 각 고장마다 고장의 개수가 틀리기 때문에 총 이웃 패턴 감응 고장 검출율을 NPSF 검출율이라 하고 다음과 같이 계산한다.

$$NPSF\ 검출율 = \frac{(ANPSF \times 검출율) + (PNPSF \times 검출율) + (SNPSF \times 검출율)}{총\ 이웃패턴\ 감응고장\ 개수}$$

표 2. March XE와 March YE의 알고리듬 시뮬레이션 결과

Table 2. Simulation results for March XE and March YE algorithms.

	March X(6 n)	March XE(16 n)	March Y(8 n)	March YE(18 n)
ANPSF	0%	5%	0%	5%
PNPSF	3%	6%	6%	6%
SNPSF	9%	18%	9%	18%
NPSF 검출율	2%	6%	2.5%	6%

<표 3>은 March C- 알고리듬에 대한 시뮬레이션 결과이다. 또한 March 알고리듬을 확장하는 다른 방법인 March PS[9]도 같이 비교하였다. March CE- 알고리듬은 March C- 알고리듬보다 3배의 복잡도를 가지지만 NPSF 검출율은 약 5배가 높아진 것을 알 수 있다. 또한 총 이웃 패턴 감응 고장 개수 중에 능동 이웃 패턴 감응 고장 개수가 가장 많은 부분을 차지하므로 March PS와 비교를 해도 NPSF 검출율은 본 논문이 제안한 알고리듬이 더 높음을 알 수 있다.

표 3. March CE- 알고리듬 시뮬레이션 결과

Table 3. Simulation results for March CE- algorithm.

	March C-(10 n)	March PS(23 n)[9]	March CE-(30 n)
ANPSF	0%	0%	12%
PNPSF	6%	25%	12%
SNPSF	15%	25%	31%
NPSF 검출율	3.3%	8.3%	15.2%

<표 4>는 March B와 March A 알고리듬에 대한 시뮬레이션 결과이다. March B는 March A 알고리듬에 읽기 동작이 2 n이 추가된 알고리듬이다. 따라서 <표 2>의 결과처럼 March A와 March B의 PNPSF만 검출율이 다르고 March AE와 March BE의 이웃 패턴 감응 고장 검출율은 같게 나온다. 그러나 복잡도가 March A의 경우 15 n에서 65 n으로 크게 증가하였다. 이것은 March A와 March B의 알고리듬은 선이 고장과 연계된 결합 고장을 검출하기 위해 쓰기 동작이 많이 추가되었기 때문이다.

표 4. March AE와 March BE 알고리듬의 시뮬레이션 결과

Table 4. Simulation results for March AE and March BE algorithms.

	March A(15 n)	March AE(65 n)	March B(17 n)	March BE(65 n)
ANPSF	0%	17%	0%	17%
PNPSF	3%	18%	6%	18%
SNPSF	12%	43%	12%	43%
NPSF 검출율	2%	21.5%	2.5%	21.5%

<그림 9>에서 최소화한 알고리듬의 시뮬레이션 결과를 <표 5>에 나타내었다. <표 5>에 보이듯이 March AE에서 (b)의 경우 테스트 길이는 65 n이었지만 (c)의 경우 35 n으로 복잡도는 거의 절반으로 감소되면서 NPSF 검출율은 약 2% 정도로 거의 차이가 없었다.

표 5. 최적화한 March AE, March BE 시뮬레이션 결과

Table 5. Simulation results for optimized March AE and March BE algorithms.

	March A(15 n)	March AE(35 n)	March B(17 n)	March BE(37 n)
ANPSF	0%	16%	0%	16%
PNPSF	3%	12%	6%	12%
SNPSF	12%	43%	12%	43%
NPSF 검출율	2%	19.8%	2.5%	19.8%

본 논문에서 제안한 방법은 모든 March 알고리듬에

적용 가능하다. <표 6>은 위 알고리듬에 대해 복잡도와 NPSF 검출율을 비교하였다. <표 6>의 결과에서 알 수 있듯이 March A 알고리듬에 본 논문에서 제안한 방법을 적용시키는 것이 가장 효율적으로 이웃 패턴 감응 고장을 검출할 수 있다는 것을 알 수 있다.

표 6. 제안한 알고리듬의 NPSF 검출율
Table 6. Fault coverage of proposed algorithms for NPSFs.

제안한 알고리듬	복잡도	검출 가능 고장	NPSF 검출율
March XE	16 n	AFs, SAFs, TFs, CFins	6%
March YE	18 n	AFs, SAFs, CFins, TFs, linked CFins	6%
March CE-	30 n	AFs, SAFs, TFs, CFins, CFids	11.9%
March AE	35 n	AFs, SAFs, TFs, CFins, linked CFids	19.8%
March BE	37 n	AFs, SAFs, CFins, linked CFids, TFs linked with CFids	19.8%

V. 결 론

메모리의 고집적화로 인하여 이웃 패턴 감응 고장 검출의 중요성이 부각되면서 테스트 시간과 고장 검출율간의 상호 절충(trade-off)을 해결하기 위한 많은 연구들이 발표되었다. 본 논문에서는 March 알고리듬의 확장으로 이웃 패턴 고장 검출율을 효과적으로 높일 수 있는 방법을 제안하였다. 제안된 방법은 약간의 테스트 시간의 증가로 고집적 DRAM에서 필히 고려해야 되는 이웃 패턴 감응 고장을 검출 할 수 있다.

참고문헌

[1] A. J. van de Goor, "Testing Semiconductor Memories," John Wiley and Sons, Chichester, UK, 1991.

[2] D. S. Suk, S. M. Reddy, "Test Procedures for a Class of Pattern-Sensitive Faults in Semiconductor RAMs," *IEEE Trans. on Computers*, pp. 419~429, June, 1980.

[3] A. J. van de Goor, I. B. S. Tlili, "Disturb Neighborhood Pattern Sensitive Fault," *Proc. of VLSI Test Symposium*, pp. 37-45, 1997.

[4] Rochit Rajsuman, "Algorithms to Test PSF And Coupling Faults in Random Access Memories," *Proc. of IEEE International Workshop*, pp. 49-54, 1993.

[5] Manoj Franklin, Kewal K. Saluja, "An Algorithm to Test RAMs for Physical Neighborhood Pattern Sensitive Faults," *Proc. of International Test Conference*, pp. 26-30, Oct, 1991.

[6] D. Kang, S. Cho, "An Efficient Built-In Self-Test Algorithm for Neighborhood Pattern Sensitive Faults in High-Density Memories," *Proc. of Korea-Russia Int'l Symp. on Science and Tech*, Vol. 2, pp. 218-223, 2000.

[7] M. S. Abadir and H. K. Reghbati, "Functional testing of semicondictor radnom access memories," *Computing Surveys*, vol. 15, no. 3, pp. 175-198, Sept. 1983.

[8] Rene David and Antoine Fuentes, "Fault Diagnosis of RAM's from Random Testing Experiments," *IEEE Trans. on Computers*, Vol. 39, pp. 220-229, Feb. 1990.

[9] V. Yarmolik, Klimets, S. Yu. Demidenko, "March PS(23N) test for DRAM pattern-sensitive faults," *Proc. of Asian Test Symposium*, pp. 354 -357, 1998.

[10] G. Park and H. Chang, "An Extended March Test Algorithm for Embedded Memories," *Proc. of Asian Test Symposium*, pp. 404-409, 1997.

[11] K. L. Cheng, M. F. Tsai, C. W. Wu, "Efficient neighborhood pattern-sensitive fault test algorithms for semiconductor memories," Proc. of *VLSI Test Symposium*, pp. 225-230, 2001.

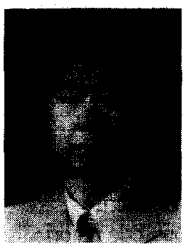
저 자 소 개

徐 一 碩(正會員)

1995년 2월 연세대학교 전기공학과 졸업. 2002년 2월 연세대학교 전기전자공학과 졸업 (석사). 현재 삼성전자 시스템 LSI 사업부 SOC 연구소

康 容 碩(正會員)

1995년 2월 연세대학교 전기공학과 졸업. 1997년 8월 연세대학교 전기공학과 졸업(석사). 2002년 2월 연세대학교 전기전자공학과 졸업(박사). 현재 LG 전자 시스템 IC 사업부 System IC R&D 센터



姜 成 昊(正會員)

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 Schlumberger 연구원. Motorola 선임 연구원. 현재 연대 공과대학 전기전자공학과 부교수