병렬 테스트 방법을 적용한 고집적 SRAM을 위한 내장된 자체 테스트 기법

(Built-In Self Test for High Density SRAMs Using Parallel Test Methodology)

康 容 碩 , 李 種 哲 , 姜 成 昊

(YongSeok Kang, Jong-Cheol Lee, and Sungho Kang)

社團法人 大韓電子工學會

論文98-35C-8-2

병렬 테스트 방법을 적용한 고집적 SRAM을 위한 내장된 자체 테스트 기법

(Built-In Self Test for High Density SRAMs Using Parallel Test Methodology)

康 容 碩 *, 李 種 哲 *, 姜 成 昊 *

(Yong-Seok Kang, Jong-Cheol Lee, and Sungho Kang)

요 약

SRAM의 집적도가 향상되어 대용량화함에 따라 하나의 셀만을 읽고 쓰는 방식을 사용한 기존의 SRAM 테스트 알고리듬은 많은 테스트 시간을 필요로 하고 이에 따라 테스트 비용도 크게 상승하게 되었다. 본 논문에서는 고집적 SRAM을 효과적으로 테스트하기 위해 동시에 여러 개의 셀을 접근할 수 있는 병렬 테스팅 기법을 제안하고 제안된 방법을 내장된 자체 테스트 기법(Built-In Self Test; BIST)으로 구현하였다. 전체 메모리 셀 어레이를 몇 개의 기본 march 블록으로 나누어 동시에 각 march 블록을 접근할 수 있도록 하드웨어를 구성하여 march 알고리듬이 병렬로 수행될 수 있도록 하였다. 새로 제시된 병렬 테스트 알고리듬은 기존 테스트 알고리듬에 비해 테스트 시간을 획기적으로 줄일 수 있으며 작은 하드웨어 오버헤드로 BIST로 구현이 가능하다. 본 논문의 결과는 적용된 march 알고리듬이 갖는 고장 검출률과 같은 검출률을 낮은 동작 복잡도로 얻어질 수 있음을 보여준다. 제시된 새로운 병렬 테스트 알고리듬은 $\sqrt{k}\times\sqrt{k}$ 크기의 기본 march 블록으로 구성된 $\sqrt{n}\times\sqrt{n}$ 크기의 SRAM을 $O(5\times\sqrt{k}\times(\sqrt{k}+\sqrt{n}))$의 복잡도로 테스트할 수 있다.

Abstract

To handle the density increase of SRAMs, a new parallel testing methodology based on built-in self test(BIST) is developed, which allows to access multiple cells simultaneously. The main idea is that a march algorithm is performed concurrently in each basic marching block which makes up whole memory cell array. The new parallel access method is very efficient in speed and requires a very tiny hardware overhead for BIST circuitry. Results show that the fault coverage of the applied march algorithm can be achieved with a lower complexity order. This new parallel testing algorithm tests an $\sqrt{n}\times\sqrt{n}$ SRAM which consists of $\sqrt{k}\times\sqrt{k}$ basic marching blocks in $O(5\times\sqrt{k}\times(\sqrt{k}+\sqrt{n}))$ test sequence.

I. 서 론

집적회로 공정 기술의 발달과 대용량 메모리에 대한 수요증가로 메모리 회로의 집적도는 빠르게 증가하고 있다. 대부분의 단일 칩 메모리는 기술적인 제약조건을 만족하는 다이(die) 크기를 유지하면서 많은 메모리 용량을 얻기 위해서 1마이크론 이하의 공정기술을 사용한다. 이러한 회로 크기의 소형화는 보다 정밀한 제조 공정을 요구하고 이에 따른 제조 비용 및 테스트 비용이 급격히 상승한다. 하지만 칩의 고집적화는 칩

* 正會員, 延世大學校 電氣工學科
(Yonsei University Dept. of Electrical Eng.)

※ 본 연구는 삼성전자의 연구비 지원에 의한 결과임
接受日字:1997年8月29日, 수정완료일:1998年8月1日

제조과정 중의 치명적인 결함에 아주 취약하여 결국 수율을 낮추게 된다. 따라서 대용량 메모리 칩을 위한 테스팅 기법의 개발은 아주 중요한 의미를 갖는다. 반도체 메모리 소자를 위한 테스트 비용과 테스트 시간은 집적도의 증가에 따라 현격하게 증가하고 있다.

초기의 메모리 테스트 알고리듬들에서는 한 개의 셀의 값을 변화시킨 후에 전체 메모리를 읽던가 혹은 값이 변한 한 개의 셀만을 읽은 동작을 기본으로 구성된다. March 알고리듬은 가장 널리 이용되는 메모리 테스트 알고리듬 중 하나이다. 이 알고리듬에서, 하나의 셀에 대하여 동작 시퀀스를 수행한 후 다음 셀에서 같은 시퀀스가 계속된다. 이러한 동작 시퀀스를 march 요소(march element)라고 부른다. 하나의 march 요소는 한 개의 셀에 1 또는 0 쓰기 동작과 한 개의 셀에서 기대되는 값 1 또는 0을 읽는 동작들의 일련된 시퀀스로 구성된다. 해당 march 요소의 모든 동작이 한 개의 셀에 대하여 수행된 후에 다음 셀에 같은 march 요소가 적용된다. 여러 종류의 march 요소를 순서를 정하여 시퀀스를 만들어 각각의 셀에 가하는 것이 march 알고리듬이다. 지금까지 march 알고리듬에 관한 많은 연구[1,2,3,4]가 발표되었으며 연역적 고장 분석법(Inductive fault analysis)과 물리적 결함 분석 방법을 사용하여 SRAM에서의 기능적 고장(functional fault)의 발생 가능성이 증명되었다[2]. 각각의 기능적 고장 모델에 따라 적절한 march 요소가 제안되었고 march 테스트의 효과도 증명되었다[5]. 그리고 이러한 march 알고리듬에 기반을 둔 메모리 BIST(Built-In Self Test)에 관한 연구도 많이 발표되었다[6,7,8,9,10]. 하지만, n 비트 메모리에 대하여 각각의 march 알고리듬은 $O(n)$의 복잡도(complexity)를 갖고 아주 많은 테스트 시간을 필요로 한다. 결과적으로 기존의 march 알고리듬은 대용량 메모리를 위해서는 더 이상 충분한 방법이 되지 못한다.

많은 테스트 시간을 필요로 하는 고집적 메모리 테스트 문제를 극복하기 위한 방법으로 몇가지 병렬 테스팅 기법이 제안되었다[11,12,13]. 이중 한 연구에서는 여러 개의 셀을 병렬로 접근하기 위해 열 어드레스 디코더를 수정한 방법을 사용하였다[13]. 그러나 이 방법은 분리된 읽기/쓰기 데이터 선을 갖는 아키텍처에서만 적용 가능하다. 메모리의 패턴 감응 고장(pattern sensitive faults; PSFs)을 병렬로 테스트하기 위해 열 어드레스 디코더와 병렬 비교기를 사용한 테스트 기법[11]도 제안되었다. 이 테스트 방법은 $O(\sqrt{n})$의 복잡도를 갖는다. 이것은 모든 제1형(type-1) 이웃 패턴 감응 고장(neighborhood pattern sensitive faults; NPSFs)과 대칭적인(symmetric) 제2형 이웃 패턴 감응 고장을 검출할 수 있다. 이 알고리듬은 SRAM의 고장 모델 중에서 대부분을 차지하는 결합 고장(coupling faults; CFs), 어드레스 고장(address faults; AFs)등 중에서 아주 작은 부분만을 검출할 수 있고 BIST 아키텍처가 오직 이웃 패턴 감응 고장의 검출만을 위해 개발되어 다른 고장의 검출을 위한 알고리듬을 적용하기 힘들다. 더구나 이웃 패턴 감응 고장은 그 발생 원인이 커패시턴스 결합 효과에 의해 발생하므로 DRAM 테스트에서 중요성을 갖지만 SRAM에서는 고려하지 않는다[2,5].

테스트 비용을 현저하게 줄이기 위해서는 기존의 메모리 아키텍처를 수정하는 것이 필요하다. 이를 통하여 많은 셀들을 동시에 접근할 수 있으며 병렬로 테스트할 수 있다. 본 논문에서는 병렬 테스팅 기법을 사용할 수 있는 새로운 SRAM BIST 아키텍처와 대용량 SRAM을 테스트하기 위한 $O(5\times\sqrt{k}\times(\sqrt{k}+\sqrt{n}))$의 복잡도를 갖는 march 알고리듬에 기반을 둔 새로운 테스트 알고리듬을 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.

고장 모델은 다양한 방법으로 만들어질 수 있다. 논리적 고장(logical faults)의 모델링을 하기 위해서는 메모리 회로의 물리적인 설계형태와 레이아웃 상의 결함들을 조사해야한다. SRAM 칩의 기능적(functional) 모델은 어드레스 래치(latch), 행 디코더, 열 디코더, 메모리 셀 어레이, 감지 증폭기(sense amplifier), 쓰기 증폭기 그리고 데이터 레지스터 등 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[3]. 물리적 스팟(spot) 결함을 기반으로 한 SRAM의 고장 모델들은 이전의 연구[2]에서 발표되었다. 이것에 의하면 SRAM은 고장 모델링을 위해 앞서 말한 세 개의 기본적인 기능 블록으로 나눌 수 있다. 이 장에서는 1비트 단위(1 bit-oriented) SRAM의 기능적 블록들에 대한 고장 모델을 설명한다.

1. 메모리 셀 어레이에서의 고장 모델

많은 다양한 고장들이 메모리 셀 어레이에서 발생할 수 있다[14,15]. 이러한 셀 어레이에서의 고장 모델들은 금속성(metalization) 단락이나 커패시티브 결합(capacitive coupling)과 같이 물리적 결함에 근거한다. 이러한 고장들은 오직 한 개의 셀만을 포함하는 고장과 한 개의 셀 혹은 한 무리의 셀이 다른 셀에 영향을 미치는 고장으로 분류할 수 있다. 셀 어레이에서의 고장을 아래에 나타내었다.

- 고착 고장(SAF) : 한 개의 메모리 셀이 항상 논리값 0(SA0)이나 논리값 1(SA1)에 영구적으로 고착된 고장을 말한다.
- 전이 고장(TF) : 한 개의 메모리 셀이 0에서 1로의 전이나 1에서 0으로의 전이를 할 수 없는 고장을 말한다.
- 개방고장(SOF) : 한 개의 메모리 셀이 워드선의 개방과 같은 결함에 의해 접근되지 못하는 고장을 말한다.
- 상태 결합 고장(SCF) : 결합원셀(coupling cell) j가 어떤 특정한 상태일 경우 피결합셀(coupled cell) i의 값이 하나의 논리 상태로 고정되는 고장으로 셀 j의 값이 0인 경우 셀 i의 값이 1로 고정되는 것을 <0;0/1>로 나타내면 다음과 같은 네 가지 종류의 상태 결합 고장이 있을 수 있다. <0;0/1>, <0;1/0>, <1;0/1>, <1;1/0>
- 동행 결합 고장(CFid) : 결합원셀 j의 전이 쓰기 동작에 의해 피결합셀 i의 값이 특정한 값으로 고정되는 고장으로 셀 j의 0에서 1로의 전이 쓰기 동작이 셀 j의 값을 0으로 고정시키는 것을 <↑;0>로 나타내면 다음과 같이 네 가지 종류의 동행 결합 고장이 있다. <↑;0>, <↑;1>, <↓;0>, <↓;1>
- 반전 결합 고장(CFin) : 결합원셀 j의 전이 쓰기 동작이 피결합셀 i의 내용을 반전시키는 고장으로 결합원셀 j의 0에서 1로의 전이 쓰기에 의해 셀 i의 값이 반전되는 것을 <↑;↕>로 나타내면 다음과 같이 두 가지 종류의 반전 결합 고장이 존재한다.<↑;↕>, <↓;↕>

2. 어드레스 디코더에서의 고장 모델

어드레스 디코더 고장(AF)은 다음과 같이 4가지 종류의 고장으로 분류할 수 있다.

고장 1. 특정한 어드레스로 어떤 셀도 접근할 수 없는 고장
고장 2. 특정한 셀을 접근할 수 있는 어드레스가 없는 고장
고장 3. 특정한 어드레스로 여러 개의 셀들이 동시에 접근되는 고장
고장 4. 여러 개의 어드레스가 하나의 셀로 접근되는 고장이 있다.

어드레스 수와 같은 수의 메모리 셀이 존재하기 때문에 이와 같은 네 가지 종류의 고장은 오직 하나만이 존재할 수 없고 두 가지 종류의 고장이 조합을 이루게 된다. 만약 고장 1이 발생하면 고장 2나 고장 3이 발생하게 된다. 고장 2가 발생한 경우에는 최소한 고장 1이나 고장 4가 함께 발생하고 고장 3이 발생하면 고장 1이나 고장 4가 발생하고 고장 4가 발생하면 고장 3이나 고장 4가 발생한다. 즉, 고장 1과 고장 2, 고장 1과 고장 3, 고장 2와 고장 4, 그리고 고장 3과 고장 4의 조합이 존재하여 어드레스 고장이 발생하게 된다.

3. 읽기/쓰기 회로에서의 고장

감지 증폭기나 쓰기 구동 회로의 선이 0이나 1로 고착될 수 있다. 하지만 어떤 경우라도 이러한 고장은 메모리 셀 어레이에서의 고착 고장과 같이 간주될 수 있다[3,7,16]. 또한 데이터의 입출력 선이 단락되거나 커패시티브 결합이 발생할 수도 있다. 이러한 고장 역시 메모리 셀 어레이에서의 결합 고장으로 나타낼 수 있다. 따라서 별도의 읽기/쓰기 회로를 위한 고장 모델 없이 셀 어레이에서의 고장 모델만으로도 충분하다.

Ⅲ. 병렬 테스트를 위한 회로 수정

초창기의 기능적 메모리 테스팅 방법은 임기 응변적

인 방법을 취하였다. 이렇게 임기 응변적인 방법에 기초한 테스팅 알고리듬으로는 marching 1's/0's와 GALPAT 등이 있다. 이러한 알고리듬의 복잡도는 각각 $O(n)$과 $O(n^2)$이다. 비록 march 1's가 선형적인 복잡도를 갖지만 메모리에 존재하는 모든 기능적 고장을 검출할 수 없다. 높은 고장 검출율을 갖는 테스트 알고리듬도 대용량 메모리 시스템을 위해서는 받아들이기 힘든 많은 테스트 시간을 필요로 한다. 고장 검출율을 증가시키면서 동시에 복잡도를 $O(n)$로 낮춘 향상된 메모리 테스트 알고리듬[2,3,16,17]이 여러 가지 존재한다. 하지만 최근의 수 메가비트의 메모리를 테스트하기 위해 이러한 알고리듬을 사용하면 여전히 많은 테스트 시간을 필요로 한다.

테스트 시간은 동시에 여러 개의 셀을 테스트하거나 여러 개의 칩을 테스트함으로써 줄일 수 있다. 메모리 테스팅의 복잡도와 비용을 줄이기 위해 제안된 방법 중의 하나인 BIST를 사용함으로써 여러 개의 칩을 동시에 테스트할 수 있다. 메모리의 여러 개의 셀을 동시에 테스트하기 위해서는 어드레스 디코더의 수정이 필요하다. 수정된 디코더를 사용하여 디코더에 의해 선택된 모든 셀에 하나의 논리값을 병렬로 쓸 수 있다. 만약 읽기 동작에서 여러 개의 워드선을 통해 여러 개의 셀을 선택하면 같은 비트선을 공유하는 선택된 셀들이 저장하고 있는 데이터가 섞여 정확히 셀의 데이터를 읽는 것이 불가능하다. 따라서 읽기 모드에서는 오직 하나의 워드선과 여러 개의 비트선을 선택하여야 하고 선택된 워드선과 비트선에 의해 접근되는 셀들의 데이터를 병렬로 읽는다. 병렬 비교기를 사용하여 읽기 동작에서 읽혀야 하는 비교값을 저장하고 단지 모든 선택된 셀들이 같은 논리값을 갖고 있는가만 결정하면 된다. 왜냐하면 만약 어떤 고장에 의해 잘못된 쓰기 동작이 행해지거나 어떤 셀의 값이 변하면 비교기의 입력들이 일치하는 값을 가질 수 없기 때문이다. 비교기는 이러한 비정상적인 입력값을 검출하여 고장의 발생을 알리는 역할을 한다.

1. 어드레스 디코더 회로의 수정

수정된 어드레스 디코더는 그림 1의 점선으로 표시된 것과 같이 두 개의 입력 비트열을 갖는다. 하나는 일반적인 어드레스 비트열(A_2)이고 다른 하나는 마스크 어드레스 비트열($\overline{M_2}$)이다. 정상 모드에서는 마스크 어드레스의 모든 비트는 논리값 0을 갖는다. 테스트 모드에서는 마스크 어드레스의 최상위 비트가 메모리 어레이를 같은 크기의 두 개의 어레이로 나누어 접근할 수 있도록 한다. 즉, 좌우로 셀 어레이를 나누던지 상하로 셀 어레이를 나눈다. 그리고 최상위 비트의 값에 따라 나누어진 두 개의 셀 어레이를 모두 선택하거나 둘 중에 하나만을 선택할 수 있다. 두 번째 최상위 비트는 나누어진 두 개의 셀 어레이를 다시 두 개로 나누고 선택할 수 있다. 이러한 방식으로 마스크 어드레스는 하나에서 여러 개의 셀을 동시에 접근할 수 있도록 한다. 여러 개의 셀을 선택할 수 있도록 하기 위해서는 마스크 어드레스의 한 비트가 논리값 1을 갖는 경우 일반 어드레스의 같은 위치의 비트를 마스크하여 해당 어드레스 비트가 논리값 1을 갖는 경우와 0을 갖는 경우가 모두 선택될 수 있도록 마스크 어드레스 디코더를 구현하여야 한다.

그림 1에서는 세 개의 일반 어드레스 선을 갖는 어드레스 디코더에서 일반 어드레스 선중 최상위 비트를 마스킹하기 위한 마스크 어드레스 디코더를 나타내었다. 이를 위해 한 비트의 마스크 어드레스 입력과 어드레스 마스킹을 위해 회로를 추가하는 방법에 대하여 나타내었다. M으로 나타낸 것이 마스크 어드레스이고 A로 나타낸 것이 일반 어드레스 입력이다. $\overline{O_2}$와 O_2는 반전된 어드레스 입력과 반전되지 않은 어드레스를 입력으로 사용하는 기존의 어드레스 디코더의 입력이다. M_2의 값이 1인 경우 두 출력 $\overline{O_2}$와 O_2는 모두 논리값 1을 갖는다. 반면에 M_2의 값이 0인 경우에는 두 출력 $\overline{O_2}$와 O_2의 값은 각각 일반 어드레스 비트 A_2의 반전된 값과 반전되지 않은 값과 같게 되어 마스킹되지 않는다. 이러한 어드레스 디코더 회로의 수정은 기존의 디코더에 약간의 회로를 추가하는 것으로 쉽게 구현할 수 있다. 사용하는 마스크 어드레스의 비트수가 m이라고 하면 $6m$개의 트랜지스터가 추가적으로 필요하다. 그림 1에 나타낸 마스크 어드레스 디코더는 일반 어드레스 입력이 3개이므로 모두 8개의 셀을 접근할 수 있다.

그림 1의 동작을 그림 2에 나타내었다. 그림 2에서 볼 수 있듯이 마스크 어드레스 M_2의 값이 1이 되면 일반 어드레스 A_2의 값에 상관없이 두 어드레스 입력 A_1과 A_0만으로 선택하게 되어 8개의 셀을 둘로 나눈 후 같은 위치에 있는 두 개의 셀을 동시에 접근할 수 있다.

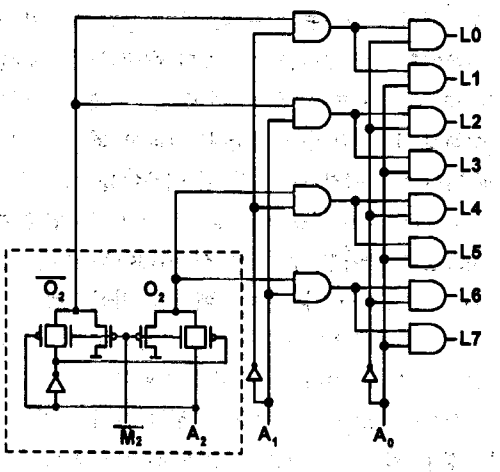


그림 1. 마스크 어드레스 디코더의 예
Fig. 1. Example of Mask Address Decoder.

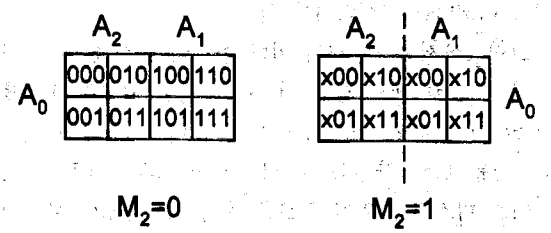


그림 2. 그림 1에 나타난 마스크 어드레스의 동작
Fig. 2. Operation of mask address decoder in Fig 1.

2. 병렬 비교기

병렬 비교기는 그림 3에 나타내었다. 병렬 비교기는 선택된 셀들의 출력값들이 같은 값을 갖고 있는가를 비교하는 역할을 한다. 만약 선택된 여러 개의 셀들 중에서 어떤 비트선의 값이 다른 비트선의 값과 다른 경우 비교기의 출력은 0이 되고 고장이 있음을 나타낸다. 정상 동작 모드에서는 비교기가 감지 증폭기와 분리되어 메모리 동작의 성능을 저하하지 않도록 한다. 테스트 모드에서는 감지 증폭기를 병렬 비교기와 연결하는데 그림 3에서와 같이 WD와 RD를 사용하여 감지 증폭기의 반전되지 않은 출력과 반전된 출력을 각각 병렬 비교기에 연결한다. 두 신호 WD와 RD는 데이터 생성기(DG)에서 발생시키게 되는데 생성된 데이터에 따라 감지 증폭기의 출력이 1인 경우에는 WD는 0이 되고 RD는 1이 되어 반전된 값이 접지와 연결된 NMOS의 게이트 입력으로 사용되어 게이트의 입력이 한 개라도 1인 경우에는 미리 충전된 고장 표시선(Error Flag)과 접지사이에 전류 경로가 형성되어 표시선의 출력이 0이 된다. 감지 증폭기의 출력이 0인 경우에는 WD는 1이 되고 RD는 0이 되어 역시 마찬가지로 고장을 검출할 수 있다. 테스트 모드에서는 WD와 RD는 같은 값을 가질 수 없으며 정상 동작 모드에서는 두 신호 모두 논리값 0을 갖게 되어 감지증폭기와 병렬 비교기를 전기적으로 분리할 수 있다.

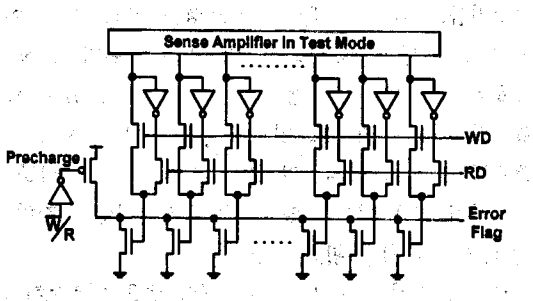


그림 3. 병렬 비교기
Fig. 3. Parallel Comparator.

Ⅳ. 테스트 알고리듬

만약 전체 메모리 셀의 개수가 n개이고 각각 열 어드레스의 개수가 $\sqrt{n}$, 행 어드레스가 $\sqrt{n}$개로 구성된 메모리가 있다면 이는 $\sqrt{n}\times\sqrt{n}$의 2차원 구조의 셀 어레이 형태를 취할 것이다. 이러한 형태의 셀 어레이는 작은 여러 개의 블록의 집합으로도 생각할 수 있다. 한 개의 블록이 $0<k<n$의 조건을 만족하는 $\sqrt{k}\times\sqrt{k}$로 구성된다면 마스크 어드레스 디코더를 사용하여 각 블록의 같은 위치에 있는 셀들을 동시에 접근하여 테스트할 수 있다. 따라서 전체 메모리를 테스트하는데 소요되는 시간은 한 개의 블록을 테스트하는데 필요한 시간으로 줄어들 수 있다. 이러한 블록을 "march 블록"이라 정의하여 사용하자. march 블록의 크기는 고장 검출율, 테스트 시간, 그리고 하드웨어 오버헤드 등의 요소에 따라 결정된다.

그림 4에 나타낸 March C- 알고리듬이 본 논문에서 제안된 병렬 BIST 방법에 적용되었다. March C-는 $10\times n$의 동작수를 필요로 하고 모든 고착고장, 어드레스 고장, 전이 고장, 그리고 모든 상태 결합 고장, 동행 결합 고장, 반전 결합 고장을 검출할 수 있다. 개방 고장의 경우 감지 증폭기의 구성 방식에 따라 March C- 알고리듬으로 검출 가능할 수도 있고 불가능할 수도 있다. 만약 감지 증폭기가 순차회로로 구성된 경우에는 March C-로는 검출이 불가능하고 알

고리듬을 수정하여야 한다. 그렇지 않은 경우 모든 개방 고장의 검출이 가능하다[2,5,14].

$\Uparrow(w0);\ \Uparrow(r0, w1);\ \Uparrow(r1, w0);\ \Downarrow(r0, w1);\ \Downarrow(r1, w0);\ \Downarrow(r0)$
$M_1 \qquad M_2 \qquad M_3 \qquad M_4 \qquad M_5 \qquad M_6$

그림 4. March C- 알고리듬
Fig. 4. March C- Algorithm.

March C- 알고리듬은 2장에서 제시한 모든 고장 모델을 검출할 수 있지만 수 메가비트의 메모리를 테스트하기 위해서는 많은 테스트 시간을 필요로 한다. 하지만 병렬 테스트 방법으로 테스트 시간은 대폭 감소시킬 수 있다.

다음에 나오는 표시들은 전체 메모리 셀 어레이가 $\sqrt{n}\times\sqrt{n}$로 구성되고 march 블록이 $\sqrt{k}\times\sqrt{k}$로 구성된 경우에 고장 검출율과 테스트 시간을 계산하기 위해서 사용된다.

- C : 열 어드레스 선의 수 : $\frac{1}{2}\log_2 n$
- R : 행 어도레스 선의 수 : $\frac{1}{2}\log_2 n$
- BC : march 블록의 열 어드레스 선의 수 : $\frac{1}{2}\log_2 k$
- BR : march 블록의 행 어드레스 선의 수 : $\frac{1}{2}\log_2 k$
- MC : 열 마스크 어드레스 선의 수 : $\frac{1}{2}\log_2\left(\frac{n}{k}\right)$
- MR : 행 마스크 어드레스 선의 수 : $\frac{1}{2}\log_2\left(\frac{n}{k}\right)$

테스트 모드에서 쓰기 동작만으로 구성된 march 요소(element) M_1을 수행하기 위해선 모든 마스크 어드레스를 1로 하고 일반 어드레스를 증가시키면 각 march 블록의 같은 위치에 있는 $\frac{n}{k}$개의 셀들에 0을 동시에 쓸 수 있다. 따라서 march 요소 M_1을 위해 필요로 하는 동작수는 k개의 셀로 구성된 march 블록에 있는 셀들의 수와 같다. march 요소 M_2, M_3, M_4와 M_5의 쓰기 동작은 M_1과 마찬가지로 같은 방법에 의해 쉽게 계산될 수 있다. 따라서 전체 쓰기 동작수는 $5k$가 된다. 읽기 동작은 보다 많은 동작수를 필요로 한다. 본 논문에서 제시한 테스트 방법에 같은 워드선을 공유하는 셀들만을 동시에 읽을 수 있다. 각 march 블록의 같은 위치에 있고 같은 워드선을 공유하는 셀들을 읽을 때 행 마스크 어드레스는 모두 0의 값을 가져야 한다. 이렇게 함으로써 일반 행 어드레스는 마스크가 되지 않게 되어 읽기 동작수는 쓰기 동작수의 $\sqrt{\frac{n}{k}}$배가 된다. 결과적으로 march 요소 M_2, M_3, M_4, M_5와 M_6의 총 읽기 동작수는 $5k\times\sqrt{\frac{n}{k}}$가 되어 $\sqrt{n}\times\sqrt{n}$의 셀 어레이가 $\sqrt{k}\times\sqrt{k}$ 크기의 march 블록으로 구성된 SRAM의 테스트를 위해 필요요한 총 동작수는 $5k+5k\times\sqrt{\frac{n}{k}}=5\sqrt{k}\times(\sqrt{k}+\sqrt{n})$이 되어 복잡도는 march 블록의 크기를 나타내는 변수 k의 값에 따라 변하게 된다.

표 1에 64K 비트의 march 블록 64개로 구성된 4M 비트의 SRAM을 테스트하기 위한 어드레스 시퀀스를 나타내었다. 표 1에 나타낸 이진 숫자는 어드레스 선의 값을 나타낸다. 4M 비트의 SRAM은 셀의 개수 n이 2^{22}개이므로 총 열 어드레스 선의 수 C와 행 어드레스 선의 수 R는 각각 11비트가 되고 march 블록의 어드레스 선의 수인 BC와 BR는 각 8 비트가 된다. 따라서 마스크 어드레스 MC와 MR는 각각 3 비트씩 사용된다. 이 6 비트의 마스크 어드레스를 이용하여 64개의 march 블록을 1에서 64개까지 선택할 수 있게 된다. 표 1에서 MR와 MC가 모두 '111'인 경우는 쓰기 동작으로 일반 어드레스 C와 R의 최상위 3비트는 마스크되므로 무상관(don't care)을 나타내는 'X'로 나타내었다. MR이 '000'이고 R의 최상위 3 비트가 마스크되지 않은 경우는 읽기 동작을 나타낸다. 읽기/쓰기 동작의 구별은 $R/\overline{W}$로 나타내었다. 표에서 볼 수 있듯이 일반 어드레스의 하위 8 비트씩을 나타내는 march 블록 어드레스 BC와 BR는 1씩 증가하거나 1씩 감소하고 쓰기 동작에서는 마스크되는 일반 행 어드레스의 상위 3 비트도 읽기 동작에서는 1씩 증가하거나 1씩 감소하여 이들은 계수기를 사용하여 쉽게 구현할 수 있다. 더구나 마스크 어드레스는 모두 1을 갖던가 0을 갖게 되면 되므로 구현하기 용이하다.

변수 k는 고장 검출율에도 영향을 준다. 고착 고장, 전이 고장, 개방 고장의 경우에는 100% 테스트할 수 있지만 어드레스 고장과 결합 고장의 경우에는 기존의 *March C-* 알고리듬에 비해 고장 검출율이 약간 하락하게 된다. 이러한 문제가 발생하는 이유는 각 march 블록의 같은 위치에 있는 셀들 사이에 결합 고장이 있던지 어드레스 고장이 있는 경우가 발생하기 때문이다. 따라서 n개의 셀들 중에 $\frac{n}{k}$개의 셀들 사이의 결합 고장이나 어드레스 고장을 검출할 수 없으

므로 기존의 March C- 알고리듬의 결합 고장과 어드레스 고장 검출율을 100%로 할 때 제안한 알고리듬의 결합 고장과 어드레스 고장에 대한 고장 검출율은 $100\times(1-\frac{1}{k})\%$로 감소한다.

표 1. March C-의 병렬 테스트를 위한 어드레스 시퀀스

Table 1. Address Sequences for Parallel March C-.

march 요소	M_1	M_2와 M_3	M_4와 M_5	M_6
어드레스 시퀀스 및 읽기/쓰기 동작	R=XXX00000000 C=XXX00000000 MR=111 MC=111 R/$\overline{W}$=0 R=XXX00000000 C=XXX00000001 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=XXX00000000 C=XXX11111111 MR=111 MC=111 R/$\overline{W}$=0 R=XXX00000001 C=XXX11111111 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=XXX11111111 C=XXX11111111 MR=111 MC=111 R/$\overline{W}$=0	R=00000000000 C=XXX00000000 MR=000 MC=111 R/$\overline{W}$=1 R=00100000000 C=XXX00000000 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=11100000000 C=XXX00000000 MR=000 MC=111 R/$\overline{W}$=1 R=XXX00000000 C=XXX00000000 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=00000000000 C=XXX00000001 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=11100000000 C=XXX00000001 MR=000 MC=111 R/$\overline{W}$=1 R=XXX00000000 C=XXX00000001 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=11111111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 R=XXX11111111 C=XXX11111111 MR=111 MC=111 R/$\overline{W}$=0	R=11111111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 R=11011111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=00011111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 R=XXX11111111 C=XXX11111111 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=11111111111 C=XXX11111110 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=00011111111 C=XXX11111110 MR=000 MC=111 R/$\overline{W}$=1 R=XXX11111111 C=XXX11111110 MR=111 MC=111 R/$\overline{W}$=0 ⋮ R=00000000000 C=XXX00000000 MR=000 MC=111 R/$\overline{W}$=1 R=XXX00000000 C=XXX00000000 MR=111 MC=111 R/$\overline{W}$=0	R=11111111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 R=11011111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=00011111111 C=XXX11111111 MR=000 MC=111 R/$\overline{W}$=1 R=11111111111 C=XXX11111110 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=00011111111 C=XXX11111110 MR=000 MC=111 R/$\overline{W}$=1 ⋮ R=00000000000 C=XXX00000000 MR=000 MC=111 R/$\overline{W}$=1

하지만 k의 값이 충분히 크면 결합 고장과 어드레스 고장에 대한 고장 검출율은 약 99.9%가 된다. 예를 들어 64K 비트의 march 블록들로 구성된 4M 비트의 SRAM의 경우 결합 고장과 어드레스 고장에 대한 고장 검출율은 99.998%가 되어 아주 미미하다. 더구나 결합 고장과 어드레스 고장의 발생 빈도가 고착 고장, 전이 고장 그리고 개방 고장이 발생하는 전체 빈도에 비해 낮기 때문에 감소되는 고장 검출율은 더욱 낮아진다[14].

V. BIST 아키텍처

March C-를 병렬로 수행하기 위해 구현된 BIST 아키텍처는 표 1에 나타낸 어드레스 시퀀스를 생성하고 그에 알맞은 읽거나 쓰기 위한 데이터를 생성해야 한다. March C- 알고리듬은 BIST로 구현하기 알맞은 알고리듬이지만 *MC*와 *MR*로 표시한 마스크 어드레스와 제어 신호를 생성하는 것은 쉽지 않은 문제이다. 일반 어드레스선 *C*와 *R*중에서 총 $\log_2 k$ 비트인 march 블록 어드레스 *BC*와 *BR*는 그림 5에 나타낸 동기식 가감 계수기를 사용하여 생성하고 테스트 모드(TM)상의 읽기 동작에서 1씩 변하는 상위 $\frac{1}{2}\log_2\left(\frac{n}{k}\right)$ 비트의 행의 어드레스는 그림 6에 나타낸 계수기를 사용하여 생성한다. 마스크되는 열 어드레스의 경우 테스트 모드에서는 읽기/쓰기 동작 모두에서 무상관 상태이므로 별도로 생성할 필요가 없다. 그림 5와 6에서 C_D와 C_U로 끝나는 신호는 계수기가 1씩 감소하는 경우와 증가하는 경우에 각각 사용하는 클럭 신호이다.

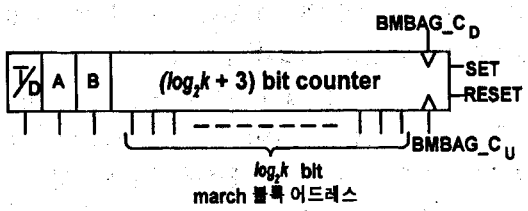


그림 5. march 블록 어드레스 생성기(*BMBAG*)

Fig. 5. Basic March Block Address Generator.

이러한 접근 방법으로 읽기 동작과 쓰기 동작수의 차이로 인해 하나의 계수기를 사용하는 경우 복잡한 제어 회로를 필요로 하는 문제를 해결할 수 있다. 그림 5와 6에서 $\overline{T}$/D, A, B 그리고 C와 같이 필요

로 하는 어드레스 비트 외에 추가된 비트들은 제어 신호를 생성하기 위해서 사용된다. 이들 신호와 생성된 클럭 신호만으로 메모리의 크기에 상관없이 제어 신호를 생성할 수 있다. 이들 신호는 수행해야 할 march 요소, 현재 동작이 읽기인가 쓰기인가의 구별, 다음 접근할 어드레스의 값이 증가된 값인가 감소된 값인가를 표시한다. 마스크 어드레스 생성기(*MAG*)는 테스트 모드에서 상위 $\frac{1}{2}\log_2\left(\frac{n}{k}\right)$ 비트의 모든 열 어드레스를 마스크하기 위해서 마스크 열 어드레스(*MC*)는 항상 모두 1값을 갖는다. 하지만 마스크 행 어드레스 (*MR*)는 테스트 모드이더라도 읽기 동작(RD)에서는 상위 $\frac{1}{2}\log_2\left(\frac{n}{k}\right)$ 비트의 행 어드레스를 마스크하지 않고 쓰기 동작(WD)에서만 마스킹한다. 데이터 생성기(*DG*)는 예상되는 읽기 데이터(RD)와 써야할 데이터(WD)는 현재 진행중인 march 요소에 의해 결정되기 때문에 각 march 요소가 다음 요소로 바뀔 때 변화하는 B 신호가 데이터 생성기를 제어하는 신호로 사용된다.

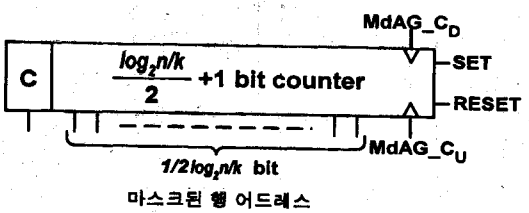


그림 6. 마스크되는 행 어드레스 생성기(*MdAG*)
Fig. 6. Masked Row Address Generator.

BIST 회로의 제어 신호를 생성하기 위한 기본 신호들은 표 2에 나타낸 것과 같은 동작을 수행한다. 표 2에서 Up과 Dn은 각각 어드레스가 현재 증가하고 있는가 혹은 감소하고 있는가를 의미하고 RM과 WM은 읽기/쓰기 동작을 구별하기 위하여 사용된다. 표 1의 시퀀스에서 알 수 있듯이 어드레스가 증가하는 방향으로 읽는 동작이 2개고 쓰는 동작은 3개이다. 또한 어드레스가 감소하는 방향으로 읽는 동작은 총 3개이고 쓰기 동작은 2개이다.

제안된 BIST 아키텍처의 각 march 요소들의 동작을 살펴보자. 64개의 64K 비트 march 블록으로 구성된 4M 비트 SRAM을 표 1에 나타낸 순서로 테스트하는 과정은 다음과 같다. 표 2에서는 각 제어신호의 생성 방법과 BIST의 동작을 나타낸다. TM이 1로 인가되자마자 모든 BIST 회로들은 리셋된다. 그 후, M_1 march 요소가 병렬로 수행된다. BMBAG_C_U 신호가 인가되어 march 블록 어드레스 생성기는 $\log_2 k$ 비트의 어드레스를 생성하고 모든 마스크 어드레스 비트 *MC*와 *MR*는 1의 값을 갖는다.

표 2. 각 march 요소에 대한 BIST 동작
Table 2. BIST Operation in Each March Element.

march 요소	$\overline{T}$/D A B C	활성 신호선
M_1	0 0 0 0	U_n_WM1
M_2	0 0 1 0 0 0 1 1	U_n_RM1 U_n_WM2, *MdAG* 리셋
M_3	0 1 0 0 0 1 0 1	U_n_RM2 U_n_WM3, *MdAG* 리셋
	0 1 1 0	*MdAG*와 *BMBAG* 리셋
M_4	1 1 1 1 1 1 1 0	D_n_RM3 D_n_WM2, *MdAG* 셋
M_5	1 1 0 1 1 1 0 0	D_n_RM2 D_n_WM1, MdAG 셋
M_6	1 0 1 1 1 0 1 0	D_n_RM1 *MdAG* 셋, D_n
	1 0 0 0	테스트 종료

동시에 데이터 생성기는 WD 신호에 0을 인가하고 R/$\overline{W}$ 신호는 0이 되어 $\overline{CS}$ 신호가 0일 때 0을 병렬로 쓰는 동작을 수행한다. 한번의 쓰기 동작으로 64개의 셀에 0을 동시에 쓰게 된다. 이러한 과정은 march 블록 어드레스 생성기의 *B* 신호가 0의 값을 갖고 있는 동안 계속된다. *B* 신호가 1이 되면 M_2 march 요소가 적용되기 시작한다. M_2의 동작에서는 처음의 읽기 동작을 위하여 클럭 BMBAG_C_U를 무효화(disable) 시키고 MdAG_C_U 클럭 신호가 인가되어 마스크된 행 어드레스 생성기가 증가되는 방향으로 계수를 시작한다. 각 march 블록의 같은 위치에 있는 64개의 셀을 같은 워드선을 공유하는 셀들만 병렬로 읽게 되므로 모두 8번의 읽기 동작을 필요로 한다. 읽기 동작에서는 $\overline{CS}$ 신호가 0이 되기 전에 R/$\overline{W}$ 신호가 먼저 1이 되어야 한다. WD와 RD가 각각 1과 0이 되어 그림 3의 병렬 비교기를 통해 읽은 셀들이 모두 0의 값을 갖고 있는가를 확인한다. 만약 한 개의 셀이더라도 1의 값을 갖는 경우에는 그림 3의 Error Flag가 0이 되어 고장이 있음을 알 수 있다. 8번의 읽기 동작이 끝나면 C가 1이 되고 한 번의 쓰

기 동작이 수행된다. 그 후, march 블록 어드레스 생성기는 어드레스를 하나 증가시킨다. 이렇게 하나의 어드레스에 대하여 8번의 읽기 동작과 1번의 쓰기 동작은 march 블록 어드레스 생성기의 A 신호가 1이 될 때까지 계속된다. march 요소 M_3의 동작은 데이터 값의 변화 외에 M_2의 동작과 거의 같다. M_3를 위한 동작이 완전히 끝난 후 march 블록 어드레스 생성기와 마스크되는 행 어드레스 생성기는 March C- 알고리듬의 어드레스 시퀀스를 만족시키기 위해서 어드레스가 감소하는 방향으로 계수를 한다. 따라서 계수기의 값을 모두 1로 하는 셋(set) 신호가 필요하다. 표 2에서 입력인 $\overline{I}$/D A B C 의 값이 0110의 비트열에 해당하는 출력값은 두 개의 계수기, march 블록 어드레스 생성기와 마스크되는 행 어드레스 생성기, 의 값을 모두 1로 하는 셋 신호로 사용된다. march 요소 M_4와 M_5는 어드레스의 계수 방향만 다를 뿐 M_2와 M_3와 같다. march 요소 M_6에서는 오직 읽기 동작만이 수행된다. 따라서 표 2에 나타낸 것과 같이 쓰기 신호를 8번 읽은 후 쓰기 동작을 하는 것을 막기 위해 D_n 신호를 생성하여 사용한다. 따라서 한 번의 필요없는 읽기 동작이 추가되나 하드웨어를 절약할 수 있다. 또한 읽기 동작은 고장을 발생시키지 않는다는 고장 모델의 가정에 의해 고장 검출율에도 영향을 주지 않는다. 두 개의 계수기의 클럭 허용(enable) 신호, 칩 선택신호($\overline{CS}$), 그리고 읽기/쓰기 신호(R/$\overline{W}$)는 표 2의 신호들을 사용하여 쉽게 생성할 수 있다. 이러한 클럭 허용 신호들로 BIST 아키텍처 안의 계수기에 정확한 클럭 신호를 인가할 수 있다.

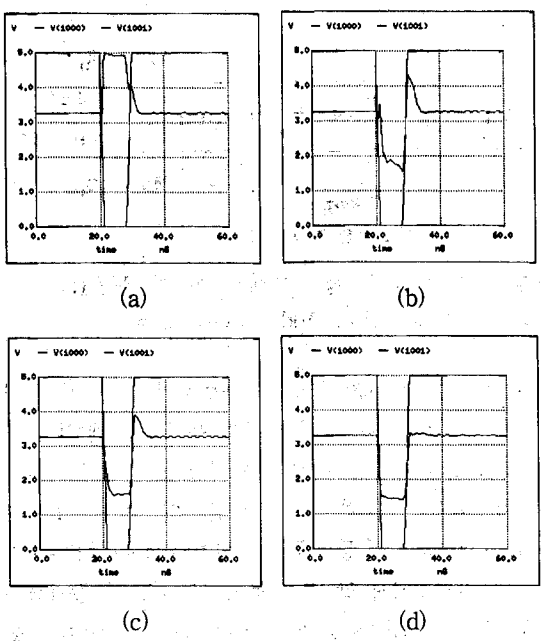
VI. 결과

앞에서 설명한 병렬 테스트 방법을 사용하면 테스트 시간은 현격하게 줄어들 수 있다. 표 3은 다른 크기의 march 블록에 대하여 기존의 March C- 알고리듬과 병렬 테스트 알고리듬의 각 블록 크기별 동작수를 비교한 것이다. 이러한 비교에서 세 가지 중요한 특징을 알 수 있다. 첫째, 제안한 병렬 테스트 알고리듬이 기존의 March C- 알고리듬에 비해 훨씬 빠르다. 둘째, 메모리의 크기가 4배씩 증가하면 제안된 아키텍처를 사용할 경우의 동작수는 2배씩 증가한다. 예를 들어 4M 비트의 SRAM이 64K 비트의 march 블록으로 구성된 경우 2.929M번의 동작을 필요로 한다. 이는 기존의 March C-이 필요로 하는 동작수의 약 6.983%에 지나지 않는다. 또 하나의 중요한 특징은 March C- 알고리듬. 이외의 다른 알고리듬도 BIST 회로를 조금만 수정하여 테스트의 복잡도를 낮추기 위해 병렬 테스트 방법을 적용할 수 있다는 것이다.

표 3. 메모리 크기와 march 블록 크기에 따른 동작 횟수

Table 3. Number of Operations for Memory Size and March Block Size.

march 블록 크기	메모리 크기에 따른 동작횟수				
	4M 비트	16M 비트	64M 비트	256M 비트	1G 비트
16K 비트	1.393M	2.703M	5.325M	10.568M	21.053M
64K 비트	2.929M	5.571M	10.813M	21.299M	42.271M
256K 비트	6.554M	11.796M	22.282M	43.254M	85.197M
1M 비트	15.729M	26.214M	47.186M	89.129M	173.015M
기존의 March C-	41.943M	167.772M	671.089M	2.684G	10.737G



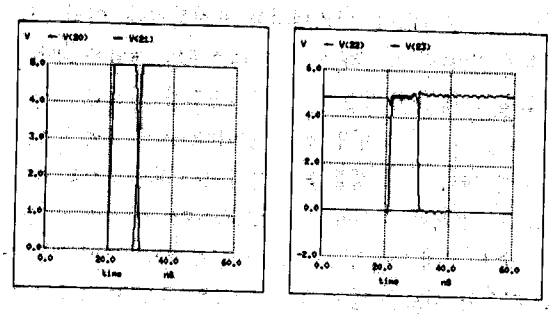
(a) (b) (c) (d)

그림 7. 병렬 비교기의 시뮬레이션 결과
(a) 고장이 없는 경우 (b) 하나의 입력이 0인 경우 (c) 세 개의 입력이 0인 경우 (d) 아홉 개의 입력이 0인 경우

Fig. 7. Simulation Results of Parallel Comparator.
(a) Fault free (b) One bit in logic 0 (c) Three bits in logic 0 (d) Nine bits in logic 0

그림 3의 병렬 비교기를 SPICE3을 이용해 시뮬레

이션을 수행한 것을 그림 7에 나타내었다. 그림에서 0ns에서 5V의 값을 갖는 신호는 반전된 precharge 신호이다. prechare가 끝나고 매치 레지스터부터 읽혀진 값이 비교기로 입력되었을 때 고장이 없는 경우에는 Error Flag 신호선의 값이 5V로 상승하는 것을 알 수 있다(a). 그러나 고장이 발생하여 비교기로 입력된 값이 모두 동일하지 않을 경우에는 그림의 (b)와 같이 Error Flag의 값이 1.5V 수준으로 하강하게 됨을 알 수 있다. 실험에서 볼 수 있듯이 비교기로 입력되는 신호중 고장인 셀의 수가 많을수록 더욱 깨끗하고 분명한 신호를 얻을 수 있었으나 한 개의 값이 0인 경우에도 충분히 그 결과를 알 수 있는 신호가 발생되었다. 따라서 개발된 비교기를 사용하면 단일 고장뿐 아니라 다중 고장도 검출이 가능함을 알 수 있다. 또한 보다 효율적으로 precharge 하기 위해 precharge 신호가 들어오는 곳의 PMOS의 채널은 그 폭을 다른 트랜지스터의 채널보다 더 넓게 설정해 주었다. 상호 비교기의 precharge 입력은 쓰기 동작을 나타내는 신호를 이용하여 precharge 시켜주고 읽기 신호가 발생하면 precharge를 멈추고 비교기의 입력 값들을 비교하면 된다. 비교기를 위한 제어신호는 데이터 생성기에서 발생된 신호를 그대로 이용할 수 있으며 게이트로 설계했을 때보다 오버헤드를 많이 줄일 수 있는 장점이 있다.



(a) (b)

그림 8. 마스크 어드레스 디코더의 시뮬레이션 결과
(a) 마스크 어드레스 디코더의 입력 (b) 수정된 디코더의 출력

Fig. 8. Simulation Results for Mask Address Decoder.
(a) Input for Mask Address decoder (b) Output of Modified decoder

그림 8은 그림 1에서 보여주는 마스크 어드레스 디코더 회로 중에서 점선으로 표시된 부분의 회로를 SPICE3을 사용하여 검증한 결과이다. 그림 8의 (a)는 마스크 어드레스 디코더의 입력(그림 1의 M_2와 A_2)을 보여주고 있다. 20ns에서 30ns 까지의 부분에서 1이된 신호가 마스크 어드레스(그림 1의 M_2)이며 30ns에서 1로 전이되는 신호가 일반 어드레스(그림 1의 A_2)의 값이다. 이와 같은 입력이 주어졌을 때 결과는 그림 8의 (b)와 같이 나타났다. 그림 (b)의 0ns에서 1인 신호는 Om 신호이다. 0ns에서 20ns사이에서 마스크 어드레스는 0을 갖고 보통 어드레스도 0을 갖는다. 이때는 마스크 어드레스 디코더는 보통의 디코더와 같이 동작하게 되고 수정된 디코더 두 개의 출력중 하나만 1의 값을 갖게 된다. 일반 어드레스의 입력값이 0이므로 이때에는 수정된 회로의 Om 신호가 1이된다. 20ns에서 30ns에서는 마스크 어드레스가 1이되도록 하였다. 이때에는 BIST 회로가 테스트 모드에 있는 경우로 마스크 어드레스 디코더의 수정부의 출력이 모두 1이 되어야 한다. 그림 8의 (b)로부터 설계된 회로가 정상적으로 동작하고 있음을 알 수 있다. 30ns 이후에는 다시 마스크 어드레스가 0이 되고 보통 어드레스는 1을 유지하고 있다. 이 경우에는 수정된 회로의 Om+1만이 1이됨을 실험 결과로부터 알 수 있다.

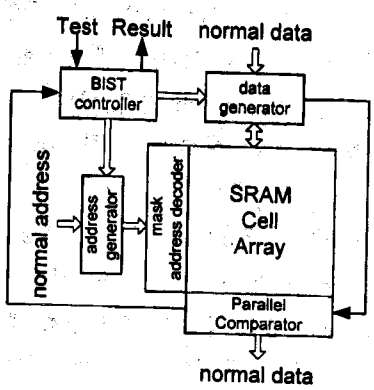


그림 9. 전체 SRAM BIST 블럭도
Fig. 9. SRAM BIST Block Diagram.

그림 9는 전체 SRAM BIST의 아키텍쳐를 나타내고 이러한 아키텍쳐를 VHDL을 이용하여 Synopsys를 사용한 고장 검출 시뮬레이션을 그림 10에 나타내었다. 편의를 위해서 행과 열 어드레스를 각 4비트로 하고 행과 열의 마스크 어드레스를 각 2비트로 하여 시뮬레이션을 수행하였다. 그림 10의 파형에서 왼쪽

부분에서 첫 번째 March 요소가 끝이 나고 그림의 오른쪽 부분에서 두 번째 March 요소가 시작되는 것을 확인할 수 있다. 현재의 동작 상황에 따라서 보통 어드레스와 마스크 어드레스가 정확하게 출력되고 있음을 확인할 수 있다.

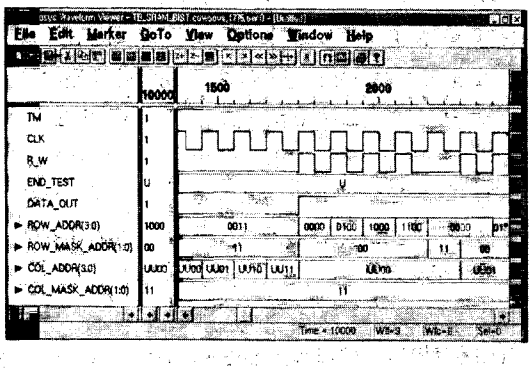


그림 10. BIST 시뮬레이션 결과
Fig. 10. BIST Simulation Result.

각 march 블록에 따라 추가된 트랜지스터의 수와 고장 검출율을 표 4에 나타내었다. 병렬 비교기는 전체 메모리 셀이 n개인 경우 $3\times\frac{1}{2}\log_2 n+3$개의 트랜지스터를 필요로 한다. 수정된 어드레스 디코더에 추가된 트랜지스터의 수는 전체 n개의 셀을 갖는 메모리가 k개의 셀의 march 블록으로 구성된 경우 $6\times\log_2\left(\frac{n}{k}\right)$개다. BIST 아키텍처에 필요한 트랜지스터의 수를 살펴보면, march 블록 어드레스 생성기와 마스크된 행 어드레스 생성기는 각각 $36(\log_2 k+3)$개와 $36\left(\frac{1}{2}\log_2\frac{n}{k}+1\right)$개의 트랜지스터를 필요로 한다. 마스크 어드레스 생성기와 데이터 생성기는 각각 $4\times\log_2\left(\frac{n}{k}\right)+12$개와 8개의 트랜지스터를 필요로 한다. 그리고 기본 신호 생성기, 제어 신호 생성기 그리고 클럭 생성기는 각각 38, 50, 36개의 트랜지스터로 쉽게 구현할 수 있다. 전체 BIST아키텍처에서 필요로 하는 트랜지스터의 수는 따라서 $14\times\log_2 k+22\times\log_2 n+288$개가 된다. 병렬 비교기와 디코더에 필요한 트랜지스터를 추가하면 전체 소요되는 트랜지스터의 수는 $8\times\log_2 k+29.5\log_2 n+291$개가 된다. 표 4에 계산된 결과를 보면 알 수 있듯이 메모리의 크기가 4배씩 증가하여도 march 블록의 크기가 일정한 경우 BIST 아키텍처에 필요한 하드웨어는 59개의 트랜지스터를 추가하기만 하면 된다. 예를 들어 march 블록의 크기가 64K 비트인 경우 4M 비트의 메모리를 테스트하기 위해서는 1068개의 트랜지스터가 추가로 필요하고 각 march 블록의 같은 위치에 있는 셀들 사이의 결합 고장을 테스트할 수 없어 약 0.001525% 가량 고장 검출율이 감소한다. 같은 march 블록의 크기로 64M 비트의 메모리를 테스트하기 위해서는 1186개의 추가적인 트랜지스터가 필요하다. 따라서 본 아키텍처는 대용량 SRAM을 테스트하는데 아주 효과적이며 march 블록 크기와 반비례하는 고장 검출율의 감소치는 아주 작아 무시할 수 있는 정도이다.

표 4. 하드웨어 오버헤드 및 고장 검출율 감소치
Table 4. Hardware Overhead and Fault Coverage Loss.

march 블록	추가된 트랜지스터 개수					고장 검출율 감소치
	4M 비트	16M 비트	64M 비트	256M 비트	1G 비트	
16K 비트	1052	1111	1170	1229	1288	0.006104%
64K 비트	1068	1127	1186	1245	1304	0.001526%
256K 비트	1084	1143	1202	1261	1320	0.000381%
1M 비트	1100	1159	1218	1277	1336	0.000095%

Ⅶ. 결 론

본 논문에서는 SRAM 테스팅 시간을 줄일 수 있는 효과적인 방법을 제안하였다. 그리고 대용량 메모리를 위한 새로운 BIST 아키텍처를 제시하였다. 여러 개의 셀들을 병렬로 접근하는 자체 테스트의 개념은 대용량 단일칩의 테스트 시간과 비용을 절감하기 위한 실용성을 갖는 효과적인 방법이 될 것이다. BIST 회로를 사용하여 여러 개의 셀들을 동시에 접근할 수 있으며 각 march 블록에서 march 알고리듬이 동시에 수행될 수 있다. 결과에서 알 수 있듯이 March C- 알고리듬을 적용한 새로운 병렬 접근 방법은 속도 면에서 아주 효과적이고 작은 하드웨어 오버헤드를 가지고 높은 고장 검출율을 얻을 수 있다. 또한 제안된 방법은 다른 march 알고리듬을 적용하는 경우에도 효과적이다. 새로운 병렬 테스트 방법은 $\sqrt{k}\times\sqrt{k}$의 march 블록으로 구성된 $\sqrt{n}\times\sqrt{n}$ 크기의 SRAM을 $O(5\times\sqrt{k}\times(\sqrt{k}+\sqrt{n}))$의 복잡도로 테스트할 수 있다.

참 고 문 헌

[1] J. P. Hayes, "Testing Memories for

Single Cell Pattern Sensitive Faults," *IEEE Trans. on Computers*, March 1980, pp. 245-254.

[2] R. Dekker, F. Beenker and L. Thijssen, "A Realistic Fault Model and Test Algorithm for Static Random Access Memories," *IEEE Trans. on CAD*, June 1990, pp. 567-572.

[3] R. Nair, S. M. Thatte and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories," *IEEE Trans. on Computers*, June 1978, pp. 572-576.

[4] M. Sachdev and M. Verstaranelen, "Development of a Fault Model and Test Algorithms for Embedded DRAMs," *IEEE Proc. of ITC*, 1993, pp. 815-824.

[5] A. J. Goor, "Using March Tests to Test SRAMs," *IEEE Design and Test of Computers*, March 1993, pp. 8-14.

[6] K. K. Saluja, S. H. Sng and Kinlshita, "Built-In self Testing RAM: A Practical Alternative," *IEEE Design and test of Computers*, Feb. 1987, pp. 42-51.

[7] S. Jain and C. Stroud, "Built-in Self Testing of Embedded Memories," *IEEE Design and Test of Computers*, Oct. 1986, pp. 27-37.

[8] P. Camurati, P. Prinetto, M. reorda, S. Barbagallo and D. Medina, "Industrial BIST of Embedded RAMs," *IEEE Design and Test of Computers*, Fall 1995, pp. 86-95.

[9] J. van Sas, G. van Wauwe, E. Huyskens and D. Rabaey, "BIST for Embedded Static RAMs with Coverage Calculation," *IEEE Proc. of ITC*, pp. 339-348.

[10] M. Franklin and K. Saluja, "Embedded RAM Testing," *IEEE Int. Workshop on Memory Technology, Design and Testing*, 1995, pp. 29-33.

[11] P. Mazumder and J, K, Patel, "Parallel Testing for Pattern-Sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans. on Computers*, March 1989, pp. 394-408.

[12] C. Elm and D. Tavangarian, "Associative Search Based Test Algorithm for Test Acceleration in FAST-RAMs," *IEEE Int. Workshop on Memory Testing*, 1993, pp. 38-43.

[13] Y. Morooka, S. Mori, H. Miyamoto and M. Yamada, "An Address Maskable Parallel Testing for Ultra High Density DRAMs," *IEEE Proc. of ITC*, 1991, pp. 556-563.

[14] A. J. Goor, *Testing Semiconductor Memories; Theory and Practice*. Singapore: John Wiley and sons, 1995.

[15] P. Mazumder and K. Chakraborty, *Testing and Testable Design of High-Density Random-Access Memories*. USA: Kluwer Academic Publishers, 1996.

[16] C. A. Papachriston and N. B. Sahgal, "An Improved Method for Detecting Functional Faults in Semiconductor Random Access Memories," *IEEE Trans. on Computers*, Feb. 1985, pp. 110-116.

[17] D. S. Suk and S. M. Reddy, "Test Procedure for a Class of Pattern-Sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans. on Computers*, June 1980, pp. 419-426.

[18] P. Camurati, P. Prinetto, M. reorda, S. Barbagallo and D. Medina, "Industrial BIST of Embedded RAMs," *IEEE Design and Test of Computers*, Fall 1995, pp. 86-95.

저 자 소 개

康 容 碩(正會員) 第 33卷 A編 第 9號 參照
1995년 2월 연세대학교 전기공학과 졸업(공학사). 1997년 8월 연세대학교 전기공학과 졸업(공학석사). 현재 연세대학교 전기공학과 박사 과정 재학 중. 주관심분야는 메모리 테스팅, Design for Testability, ATPG, VLSI & CAD 등임

姜 成 昊(正會員) 第 33卷 A編 第 9號 參照
1986년 2월 서울대학교 제어계측공학과 졸업(공학사). 1988년 5월 The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학석사). 1992년 5월 The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학박사). 1992년 11월 ~ 1992년 8월 미국 Schlumberger Inc. Research Scientist. 1992년 9월 ~ 1992년 10월 미국 The Univ. of Texas at Austin, Post Doctoral Fellow. 1992년 8월 ~ 1994년 6월 미국 Motorola Inc., Senior Staff Engineer. 1994년 9월 ~ 현재 연세대학교 기계전자공학부 조교수. 주관심분야는 테스팅, DFT, VLSI & CAD, Design Verification 등임

李 種 哲(正會員)
1996년 2월 연세대학교 전기공학과 졸업(공학사). 1998년 2월 연세대학교 전기공학과 졸업(공학석사). 현재 삼성전자 SRAM 사업부. 주관심분야는 테스팅, VLSI & CAD 등임