논문 2007-44SD-12-13

Sign Bit을 사용한 고효율의 메모리 자체 수리 회로 구조

(The Efficient Memory BISR Architecture using Sign Bits)

강 일 권*, 강 성 호**

(Ilkwon Kang and Sungho Kang)

요 약

메모리 설계 기술과 제조 공정의 발전에 따라, 고집적 메모리의 생산이 본격화 되었다. 이러한 메모리의 고집적화는 복잡하고 정밀한 설계와 제조 공정을 필요로 하기 때문에, 메모리 내에 더 많은 고장을 존재할 가능성을 낳았다. 이에 따라 메모리에서 발생하는 여러 고장을 분석하고, 메모리를 수리하여 공정상의 문제를 수정하기 위해, BISR(Built-In Self-Repair) 회로의 중요성이 부각되고 있다. 본 논문에서는 주어진 예비 메모리를 효율적으로 사용하여 고장이 발생한 메모리를 효과적으로 수리할 수 있는 메모리 내장형 자체 수리 회로의 구조와 그 방법론에 대해서 소개하고자 한다. 제안하는 자체 수리 회로는 sign bit 이라는 추가적인 저장 장치를 이용하여 메모리 수리를 수행한다. 이는 기존에 비해 좀 더 향상된 성능을 가지고 있다.

Abstract

With the development of the memory design and process technology, the production of high-density memory has become a large scale industry. Since these memories require complicated designs and accurate manufacturing processes, it is possible to exist more defects. Therefore, in order to analyze the defects, repair them and fix the problems in the manufacturing process, memory repair using BISR(Built-In Self-Repair) circuit is recently focused. This paper presents an efficient memory BISR architecture that uses spare memories effectively. The proposed BISR architecture utilizes the additional storage space named 'sign bit' for the repair of memories. This shows the better performance compared with the previous works.

Keywords: memory repair, memory BISR, BIRA, self-repair

Ⅰ. 서 론

현대의 반도체 산업에서 SoC(System on Chip)에 대한 중요성이 점점 더 커짐에 따라서 SoC에 사용되는 내장형 메모리(Embedded Memory)의 용량도 점차적으로 증대되고 있다. 이러한 경향은 앞으로도 계속되어 오는 2014년이면 전체 SoC의 면적에서 메모리가 차지하는 면적이 94%에 이를 것으로 전망된다[1]. 그 결과, 최근에 이르러서는 메모리의 수율에 의해 전체 SoC의 수율이 결정되는 상황에 이르렀다. 즉, 메모리 모듈의 수율을 합리적인 수준으로 유지하는 것은 전체 SoC를 위해서 필수적인 요소가 된 것이다. 이를 위해서 고장난 chip을 발견하는 것뿐만 아니라 고장 진단 및 repair 알고리즘을 사용하여 이를 repair하는 것 또한 SoC 제작에 있어서 매우 중요한 과정이 되었다.

SoC에 들어가는 내장형 메모리를 위한 테스트는 메모리 모듈 자체가 하나의 chip을 이루는 단품 메모리를 위한 테스트와는 차이가 있다. 내장형 메모리는 I/O pin이 외부로 노출되어 있지 않기 때문에, 이를 위한 테스트 및 수리는 아예 chip 내부에서 자체적으로 수행하는 것이 유리하다. 외부 장비를 이용한 전통적인 메모리 수리 과정은 몇 가지 한계를 지니고 있다. 메모리의 테스트 비용을 크게 증가시키고 failure bitmap을 외부로 전송할 때, 제한된 I/O pin을 사용함으로 인해 고속의 repair에 한계가 있다. 이러한 한계점들을 극복하기 위하여 BISR이 제안되었다. 이를 위해 특별히 설치된 회

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
접수일자: 2007년8월7일, 수정완료일: 2007년11월15일

로를 BIST(Built-In Self-Test) 및 BISR(Built-In Self-Repair)이라고 칭한다. BISR은 BIST의 개념에서 조금 더 발전한 개념이라고 볼 수 있다. 하드웨어 그 스스로가 자체적으로 테스트를 수행할 수 있도록 회로를 구성한 것이 BIST라면, BISR은 하드웨어 그 스스로가 고장이 발생한 부분을 수리까지 할 수 있도록 하는 것이다. 예비 메모리를 사용하여 고장이 발생한 메모리 셀의 repair를 수행할 수 있고, 아울러 내장형 메모리의 수율을 향상시킬 수 있다. 이와 같은 산업 현장의 기술 흐름에 따라, SoC 환경에서 내장형 메모리의 수율을 향상시키기 위한 효율적인 BIST 및 BISR 구조에 대한 연구가 꾸준히 요구되고 있고 또한 활발하게 진행되고 있다.

최근까지 BISR에 관한 다수의 연구 결과가 발표되었다. [2]에서는 *most-repair-rule*을 제안하여 효율적인 BISR을 위한 기본 원칙을 제시하였다.

지금까지 제안된 BISR의 연구 결과를 살펴보면, 보통 메모리의 수율은 5%에서 20%가 향상되었고, 이를 통해 전체 chip의 수율은 2%에서 10%정도가 향상된 것으로 나타났다.[3] [4]에서 제안된 구조는 특유의 융통성으로 인하여 매우 다양한 형태의 고장 그룹을 수리할 수 있지만, 높은 area overhead를 가지는 단점을 가지고 있다. [5]에서 제안된 spare row와 column의 조합을 기초로 하는 방법론은 각개격파 전략(divide and conquer strategy)을 사용하였으나 실제로 하드웨어를 만들기 위해서는 기술적인 문제가 발생하였다. 이는 좀 더 복잡한 spare 구조로의 확장성을 이루는 데에 장애요소가 되었다.

[6]에서 제안된 CRESTA(Comprehensive Real Time Exhaustive Search Test and Analysis, CRESTA)는 주어진 redundancy 조건에서 수리가 가능한 chip을 100%의 신뢰도로 찾아 낼 수 있고 solution이 존재한다면 수리 방법까지도 제시해준다. 그러나 CRESTA는 spare row나 column의 수가 많아질 경우에 복잡도가 크게 증가하고, test에 걸리는 시간 또한 급격하게 증가하는 단점이 있다. [7]은 2-D redundancy를 갖는 RAM을 위한 BISR 방법을 제시하였다. 여기서는 예비 메모리를 효과적으로 활용하여 성공적인 repair를 수행할 수 있었으나 물리적인 repair가 아닌 논리적인 방식의 repair를 채택하고 있기 때문에, BIST에 따른 remapping의 결과를 저장할 저장 매체가 비휘발성 저장매체가 아니라면 전원을 켤 때마다 다시 BIST를 거쳐서 고장 주소를 remapping하는 단점을 가지고 있다.

Ⅱ. 효과적인 BISR을 위한 새로운 방법론

1. sign bit의 설치

본 논문에서 제안하는 BISR 방법론에서 가장 두드러진 특징은 sign bit의 설치이다. 메모리 블록의 row 주소와 column 주소에 각각 별도의 sign bit을 추가하여 각 주소에 발생한 고장의 개수를 표시하는 역할을 하도록 하고 이를 이용하여 효율적인 repair가 이루어지도록 한다. sign bit을 설치한 BISR의 대략적인 구조도를 나타내면 그림 1과 같다. 그림 1에서 메모리 블록의 위쪽에 나타나 있는 column address sign bit과 메모리 블록의 왼쪽에 나타나 있는 row address sign bit을 확인할 수 있다.

row와 column address sign bit은 각각 row와 column의 크기만큼의 길이로 이루어져있다. 각각의 sign bit은 하나씩의 row와 column 주소에 서로 일대일로 대응된다.

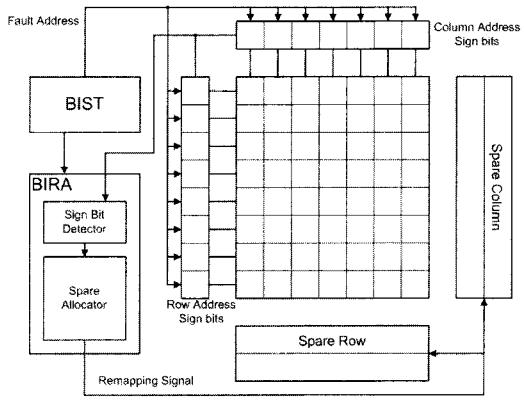

그림 1. 제안하는 BISR 방법론의 구조도
Fig. 1. Diagram of the proposed BISR method.

2. sign bit의 적용 및 동작 예시

가. sign bit을 이용한 고장의 기록

그림 2는 sign bit에 고장의 개수를 기록한 예시이다. 그림에서 X로 표시한 부분은 고장이 검출된 셀의 위치이고, row와 column 각각의 sign bit에 표시된 숫자는 해당하는 주소에서 위치한 고장의 개수를 의미한다. 예를 들어 column address sign bit에 표시된 1, 1, 1, 3에서 3의 의미는 3이 표시된 bit에서 바로 아래의 column에 총 3개의 고장이 존재한다는 의미이다. 마찬가지로 row address sign bit의 1, 2, 2, 1에서 각각의 숫자가 의미하는 것은 1이 표시된 bit의 바로 오른쪽 row에서 1개의 고장이 검출되었다는 뜻이고, 2가 표시된 bit의

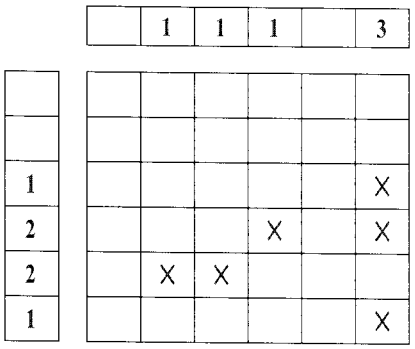

그림 2. sign bit의 사용 예
Fig. 2. The example of the sign bit.

오른쪽 row에서는 2개의 고장이 검출되었다는 뜻이다.

나. sign bit의 사용 시의 repair 알고리즘

그림 3은 row 및 column address sign bit을 적용했을 때의 repair 동작 흐름을 효과적으로 설명하기 위하여 임의의 고장을 가정한 메모리 블록이다. 메모리 repair를 위한 예비 메모리로는 각각 2개씩의 spare row와 spare column을 가정하였다. 제안하는 BISR 방법론으로 repair를 수행할 때는 sign bit에 표시된 내용만을 참고하여 repair를 수행한다. 우선 row와 column address sign bit에서 최대값을 찾는데, 이는 고장의 개수가 많은 것을 먼저 repair하여 spare row와 spare column의 효용성을 높이기 위함이다. 예를 들어 하나의 spare column이 있다고 할 때, sign bit의 값이 1인 주소를 repair하면 하나의 고장만 repair되지만, sign bit의 값이 3인 주소를 repair 하면 한 번에 3개의 고장을 repair 할 수 있으므로 동일한 예비 메모리를 훨씬 더 효율적으로 사용할 수 있다.

그림 3의 예에서는 column address sign bit의 최대

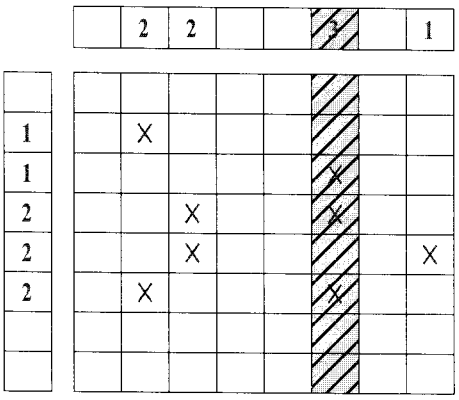

그림 3. BIST가 끝난 직후의 메모리 블록 및 sign bit에 저장되어 있는 값
Fig. 3. Memory cell array and values of the sign bit after the process of BIST.

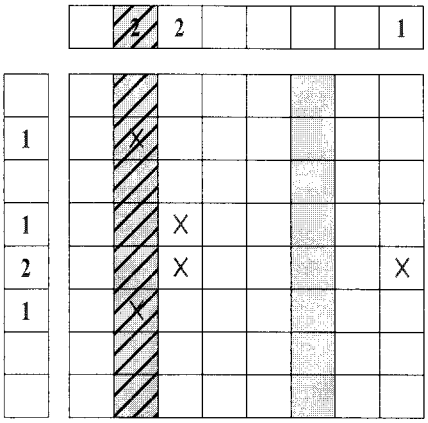

그림 4. spare column을 사용한 repair 직후의 메모리 블록과 sign bit
Fig. 4. Memory cell array and the sign bit after the repair using a spare column.

값이 3이고 row address sign bit의 최대값이 2이므로 우선 spare column을 사용한 repair를 먼저 실행하도록 한다. 한 번의 repair로 가장 많은 고장을 한꺼번에 처리할 수 있기 때문에 예비 메모리를 가장 효율적으로 사용할 수 있다. spare column을 통한 repair가 끝나고 나면, 남아있는 고장을 토대로 sign bit에 저장된 data를 update한다.

그림 4는 spare column을 통한 repair와 동시에 sign bit의 update까지 완료한 상황을 나타낸 그림이다. 그림 3의 column address sign bit에서 3으로 표시되었던 sign bit의 아래에 있는 column의 고장이 모두 repair된 것을 확인할 수 있다. 또한 row address sign bit의 내용이 새로운 값으로 update 된 것도 확인할 수 있다.

앞의 repair 과정에서 spare column을 하나 사용하였기 때문에 이제 2개의 spare row와 1개의 spare column이 남았다. 다음 repair를 위해 row와 column address sign bit의 최대값을 비교한다. row address sign bit의 최대값은 2이고 column address sign bit의 최대값 또한 2이다. 이처럼 row address sign bit의 최대값과 column address의 최대값이 같을 경우에는 spare row와 column 중에서 어떤 것을 먼저 사용할지 우선순위를 정해야 하는데 제안하는 알고리즘에서는 column을 이용한 repair에 우선권을 주는 것으로 하였다. 그러므로 그림 4의 경우에서는 column address sign bit의 숫자가 2인 곳의 repair를 먼저 진행하게 된다. 이 때, 2의 값을 가지는 column address sign bit이 두 개가 존재하여 또다시 문제가 발생하게 된다. 제안하는 알고리즘에서는 이럴 경우 column 주소의 값이 작은 sign bit에 기록된 주소에 우선권을 주게 된다. 즉,

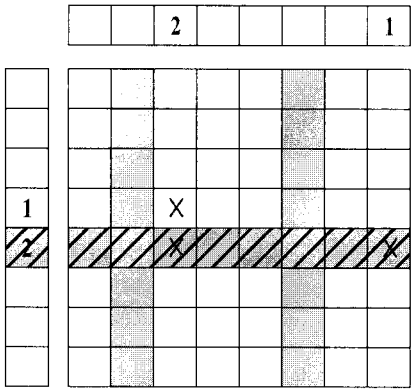

그림 5. spare column을 사용한 두 번째 repair 직후의 메모리 블록과 sign bit

Fig. 5. Memory cell array and the sign bit after the second repair using a spare column.

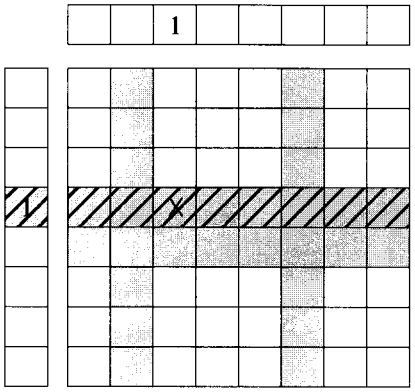

그림 6. spare row를 사용한 첫 번째 repair 직후의 메모리 블록과 sign bit

Fig. 6. Memory cell array and the sign bit after the first repair using a spare row.

column 주소를 왼쪽부터 0, 1, 2, …, 7이라고 하였을 때, column 주소가 1인 곳과 2인 곳 중에서 column 주소가 1인 곳의 column에 우선권을 준다는 뜻이다. 그러므로 그림 4에서는 spare column을 사용하여 왼쪽에서 두 번째 column의 repair를 먼저 수행하게 된다. 그 결과, 그림 5와 같은 결과를 얻을 수 있다.

그림 5는 두 번째의 spare column을 사용하여 repair를 수행한 후의 메모리 블록과 sign bit의 상태를 나타낸 그림이다. 그림에서 column address sign bit의 최대값은 2이고 row address sign bit의 최대값 또한 2이다. sign bit의 최대값이 서로 같기 때문에 column repair 우선 원칙에 의해서 spare column을 이용한 repair가 먼저 이루어져야 하지만 앞의 두 단계의 repair 과정을 통해서 이미 2개의 spare column을 모두 써버렸기 때문에 이제는 spare row를 통한 repair를 수행하여야 한다. 그러므로 row address sign bit의 숫자가 2인 row를 repair하게 된다.

그림 6은 첫 번째의 spare row를 사용하여 고장을 repair하고 난 이후의 메모리 블록과 sign bit의 모습을 나타낸 것이다.

이제 마지막 하나의 고장을 spare row를 통해 repair하면 된다. BIST 과정을 통해서 검출된 모든 고장이 주어진 예비 메모리를 통해서 repair 되었기 때문에, 각각의 sign bit에 표시된 값은 모두 사라지게 된다. 이로써 모든 고장이 완전히 수리되었음을 확인할 수 있다.

다. sign bit의 사용 시의 repair 원칙

위에서 볼 수 있듯이 알고리즘 적용을 위하여 몇 가지 원칙을 정하고 메모리 repair를 수행하여야 한다. 주요한 몇 가지 항목을 정리하여 나타내면 아래와 같다.

① BIST 과정을 통해서 고장을 검출하고, 고장이 발견되는 대로 sign bit을 update하여 발견된 고장의 개수를 기록하도록 한다.

② BIST 과정이 모두 종료되면 BISR 모듈은 repair를 시작한다.

③ row address sign bit와 column address sign bit의 최대값을 서로 비교하여 더 큰 값을 갖는 주소의 row나 column부터 repair를 수행한다.

④ ③의 과정이 끝나면 repair된 주소의 고장을 제외한 상태로 sign bit을 update 한다. sign bit에 대한 update가 끝나면 다시 ③의 과정을 반복한다.

⑤ row address sign bit의 최대값과 column address sign bit의 최대값이 서로 같다면 이때는 spare column을 통한 repair를 우선 수행한다.

⑥ 사용가능한 spare row가 남아있지 않다면 남아있는 spare column을 통해서 repair를 수행한다. 혹은 사용가능한 spare column이 남아있지 않다면 남아있는 spare row를 통해서 고장을 repair한다.

⑦ 어떤 row address sign bit나 column address sign bit에서 최대값과 동일한 값이 여러 개 존재한다면 주소 값이 작은 것을 우선 repair 한다.

⑧ 모든 sign bit의 값이 없어지면, 검출된 모든 고장이 repair되었다는 의미이므로 메모리 repair를 성공한 것이고, 아직 sign bit의 값이 남아있는데 더 이상 사용할 수 있는 spare row나 column이 남아있지 않으면, 메모리 repair를 실패한 것이다.

III. 제안된 BISR 방법론의 실험적 검증

1. 실험 환경

제안하는 알고리즘을 검증하기 위해서 C++를 이용하여 실험 환경을 구성하였다. 이를 통해서 sign bit을 이용한 BISR의 전반적인 흐름에 대한 검증과 동시에 repair rate를 계산해 낼 수 있다. 또한 다양한 메모리 크기와 고장 개수 조건, 고장 주소 조건과 함께 여러 조건 하에서 BISR 알고리즘 검증을 수행할 수 있다. 이번 장을 통해서는 제안하는 BISR 알고리즘의 검증을 위한 실험과 그 흐름에 대해서 설명하도록 하겠다.

가. 메모리 블록과 고장 및 예비 메모리의 설정

실험에 쓰일 메모리는 2차원 배열을 통해서 작성하였다. 미리 row와 column의 크기를 지정해주고 이에 따라서 배열의 크기를 정해서 메모리를 구현한다. 실험에서는 1024X1024의 메모리 블록을 가정하였다. 메모리 블록의 생성이 끝난 후에는 고장을 삽입한다. 고장을 삽입할 주소는 전체 메모리에 무작위로 고장을 뿌리는 랜덤 주소와 특정 주소 값을 기준으로 정규분포로 고장을 뿌린 가우시안 주소를 사용하였다. 실험에서는 고장의 개수가 10개인 것을 기본 조건으로 삼고, spare row와 spare column을 각각 2개와 4개씩 설치하였다. 이 조건들을 실험의 제한 조건에 따라 바꿔가면서 제안하는 방법론의 특징을 도출하였다. 이 같은 가정을 토대로 각각 1000번씩의 실험을 수행하였다.

나. 메모리 BIST 동작의 구현

메모리의 제일 처음 주소부터 순차적으로 검증한다. 메모리 블록을 순차적으로 살펴봐서 고장이 나올 때마다 해당하는 sign bit 값을 1씩 증가시킨다. BIST 과정이 종료된 이후에는 전체 메모리 블록의 고장 상황이 각각의 row address sign bit와 column address sign bit에 저장된다. BIST 과정이 종료되고 나면 row와 column address sign bit을 순서대로 살펴봐서 미리 정해놓은 sign bit의 한계 값(이 실험에서는 3)보다 큰 것들을 sign bit의 한계 값으로 대체하도록 한다. 실제로 sign bit을 구현할 때는 저장 공간의 한계로 인하여 무한대의 고장을 저장하는 것이 불가능하기 때문이다.

다. 메모리 BISR core : repair 알고리즘 동작

그림 7은 repair 동작을 나타낸 pseudo 알고리즘이다. 이 알고리즘은 spare row나 spare column이 하나라도

```
/* 남아있는 spare memory가 있을 경우 동작 */
While ( # of row spare != 0 or # of column spare != 0 )
  /* row address sign bit의 최대값을 구하는 과정 */
  For ( row index 값 <= memory cell array의 row size )
    If ( sign bit [row index] > Maximum of row sign bit )
      Maximum of row sign bit = sign bit [row index] ;
      Row address to be repaired = row index ;
  /* column address sign bit의 최대값을 구하는 과정 */
  For ( column index 값 <= memory cell array의 column size )
    If ( sign bit [column index] > Maximum of column sign bit )
      Maximum of column sign bit = sign bit [column index] ;
      Column address to be repaired = column index ;
  /* Row 와 column address sign bit의 maximum 값이 모두 0인 경우 */
    Break ;

/* spare row와 spare column이 모두 남아있을 경우의 repair 동작 */
If ( # of row spare != 0 and # of column spare != 0 )
  /* Row repair process */
  If ( row sign bit의 최대값 > column sign bit의 최대값 )
    For ( column index 값 <= 메모리 블록의 column size )
      If ( ( row address to be repaired, column index ) == 1 )
        Column address sign bit [column index] -- ;
        ( row address to be repaired, column index ) = 0 ;
    Row address sign bit [Row address to be repaired] = 0 ;
    # of spare row -- ;
  /* Column repair process */
  Else if ( row sign bit의 최대값 <= column sign bit의 최대값 )
    For ( row index 값 <= 메모리 블록의 row size )
      If ( ( row index, column address to be repaired ) == 1 )
        Row address sign bit [row index] -- ;
        ( row index, column address to be repaired ) = 0 ;
    Column address sign bit [Column address to be repaired] = 0 ;
    # of spare column -- ;
  /* spare row에만 여분이 있을 경우의 repair 동작 */
  Else if ( # of row spare != 0 and # of column spare == 0 )
    For ( column index 값 <= 메모리 블록의 column size )
      If ( ( row address to be repaired, column index ) == 1 )
        Column address sign bit [column index] -- ;
        ( row address to be repaired, column index ) = 0 ;
    Row address sign bit [Row address to be repaired] = 0 ;
    # of spare row -- ;
 /* spare column에만 여분이 있을 경우의 repair 동작 */
  Else if ( # of row spare == 0 and # of column spare != 0 )
    For ( row index 값 <= 메모리 블록의 row size )
      If ( ( row index, column address to be repaired ) == 1 )
        Row address sign bit [row index] -- ;
        ( row index, column address to be repaired ) = 0 ;
    Column address sign bit [Column address to be repaired] = 0 ;
    # of spare column -- ;

/* While 문 종료 */

/* BISR 결과 판정 process */
If (Maximum of row sign bit = 0 and Maximum of column sign bit = 0)
  BISR success ;
Else
  BISR failed ;
```

그림 7. repair 과정의 pseudo 알고리즘
Fig. 7. The pseudo algorithm of the repair process.

남아있어야 동작한다. 일단 예비 메모리가 있어서 repair 동작을 시작하게 되면 우선 row와 column 각 sign bit의 최대값을 구하는 과정을 먼저 거치게 된다. spare row와 column이 모두 남아있을 때는 row와 column address sign bit의 최대값을 서로 비교하여 최대값이 큰 쪽의 repair를 수행하게 된다.

row repair process를 진행할 때에는 sign bit이 최대값인 row 주소를 spare row를 사용해서 repair한다. 또한 동시에, repair될 row 주소의 메모리 블록에 고장이 있다면 해당하는 column 주소의 sign bit을 '1'씩 감소시키고 repair되는 row address sign bit의 값을 '0'으로

고쳐준다. 또한 spare row를 하나 사용하였으니 남아있는 spare row의 개수를 '1'만큼 감소시킨다. column repair process를 진행할 때는 위의 내용과 상반된 내용을 수행하면 된다.

spare row/column에만 여분이 있을 때에는 각각 위의 row repair process/column repair process와 동일한 과정을 진행하면 BISR의 repair 과정을 마칠 수 있다.

repair process는 사용가능한 예비 메모리가 있을 경우에만 반복적으로 수행된다. repair를 종료하는 시점은 sign bit의 최대값이 모두 '0'으로 되어 메모리 블록 내에 고장이 없을 때이다. repair process가 종료된 이후에 sign bit의 값을 조사하면 BISR의 성공 여부를 판가름할 수 있다. sign bit의 값이 모두 '0'이면 메모리의 고장이 모두 repair 되었다고 볼 수 있다.

마. 대조군

repair rate의 개선 효과를 잘 드러내기 위하여 몇 가지 대조군을 설정하였다. 본 논문의 실험에서 설정한 대조군은 다음의 두 가지이다.

ROW priority : 메모리의 고장 개수 상황과는 관계없이 row repair를 먼저 수행하는 경우

COLUMN priority : 메모리의 고장 개수 상황과는 관계없이 column repair를 먼저 수행하는 경우

row priority는 row spare와 column spare가 모두 여유가 있을 때는 무조건 row repair를 먼저 수행하고, spare row의 여유분이 없을 때 spare column을 이용한 repair를 수행하는 실험이다. 또한 column priority는 column repair를 먼저 수행하고, spare column의 여유분이 없을 때에만 spare row를 이용한 repair를 수행하는 실험으로 이를 두 번째 대조군으로 하였다.

2. 실험 결과

이번 장에서는 제안하는 BISR에서 sign bit의 설치로 인한 repair rate의 상승이, 어떤 경우에 있어서 가장 효과적인지 알아보는 실험을 진행하겠다.

가. 균일 고장 분포 조건

다음은 고장을 주소에 따라 균일하게 흩뿌리고 실험한 결과이다. 표 1은 균일 고장 분포 조건에서 고장의 개수에 따른 repair rate를 나타낸 표이다. 고장의 개수가 6개 이하일 때는 항상 repair rate가 100.0%로 나오는데 이는 예비 메모리의 개수가 6개이기 때문이다. 즉, 고장의 개수가 6개가 될 때까지는 하나의 고장에 하나의 spare를 사용하더라도 모든 고장에 대해서 repair가 가능하므로 이러한 결과가 나타난다. 그러나 고장이 7개가 되면 repair rate가 급격히 낮아지는 것을 볼 수 있는데, 이는 고장이 전체 메모리에 걸쳐서 매우 고르게 분포하고 있으므로 row 주소나 column 주소가 겹치는 고장이 드물기 때문이다.

표 1. 고장의 개수에 따른 repair rate (단위 : %)
Table 1. Repair rates with different fault numbers

고장의 개수	ROW priority	COLUMN priority	Proposed BISR
0	100.0	100.0	100.0
3	100.0	100.0	100.0
6	100.0	100.0	100.0
7	2.1	1.8	2.9

표 2. 고장 영역에 일정한 제한을 뒀을 때의 repair rate (단위 : %)
Table 2. Repair rates with constraints about the fault region.

ROW fault region	COLUMN fault region	ROW priority	COLUMN priority	Proposed BISR
R	C	0.0	0.0	0.0
R/16	C	0.0	0.0	0.0
R	C/16	0.1	0.2	0.3
R/32	C	0.5	0.0	0.5
R	C/32	0.2	1.4	1.5
R/64	C	3.8	0.1	3.9
R	C/64	1.8	12.1	12.3
R/64	C/16	8.0	0.9	8.9
R/16	C/64	5.0	13.8	15.9
R/64	C/32	13.2	3.8	16.4
R/32	C/64	10.1	14.7	21.6
R/128	C	33.7	0.5	34.0
R	C/128	19.2	75.6	75.6
R/128	C/4	37.6	1.2	38.8
R/4	C/128	21.6	75.8	76.2
R/128	C/16	36.8	3.6	40.4
R/16	C/128	30.1	72.8	75.1
R/128	C/64	52.1	22.2	62.6
R/64	R/128	54.7	75.2	83.7
R/256	C	100.0	9.8	100.0
R	C/256	100.0	100.0	100.0
R/256	C/16	100.0	10.9	100.0
R/16	C/256	100.0	100.0	100.0

표 2는 동일한 크기의 메모리에 대해서 전체 영역에 고장을 뿌리지 않고 제한된 영역에 고장을 집중시키고 BISR을 수행했을 때의 결과를 정리한 것이다. 표 2에서 '(R or C)/N'로 표시한 것의 의미는 전체 1024개의 row 혹은 column 주소 중에서 '1/N'에 해당하는 영역에 대해서만 고장이 삽입되게끔 실험을 수행하였다는 의미이다. 이와 같이 실험을 수행하면 고장이 좀 더 집중적으

로 분포하게 된다. 예를 들어 'R/4'는 전체 row의 1/4의 영역에 고장이 분포한다는 뜻이다. 'R/4'의 경우보다 'C/64'의 경우에 고장이 삽입될 수 있는 영역이 더 좁기 때문에 고장이 좀 더 조밀하게 분포한다. 이렇게 해서 실험을 수행하면 같은 row 주소나 column 주소에 고장이 겹칠 확률도 커지게 되므로 제안하는 방법론의 효과가 잘 드러날 수 있을 것이다. 표 1과는 달리, 표 2에서는 동일한 조건에서도 전체적인 repair rate가 높게 형성되어, 고장이 조밀하게 분포할수록 repair rate가 높은 것을 확인할 수 있다. 어떤 경우에 대해서도 제안하는 BISR 방법을 통한 repair rate가 가장 높게 형성되는 것을 확인할 수 있는데, 이는 고장이 조밀하게 분포한다면 sign bit을 이용하여 고장이 많은 주소의 row나 column부터 repair함으로써 예비 메모리를 사용하는 것이 가장 repair 효율이 높기 때문이다.

나. 가우시안 고장 분포 조건

다음은 고장을 주소의 중앙값을 평균으로 하는 정규 분포의 형태로 흩뿌리고 실험한 결과이다. 모든 실험에서 기본적인 표준 편차는 3으로 설정하고 실험하였다. 표 3은 고장 개수에 따라 변화하는 repair rate를 표시한 것이다. 표를 통해서 볼 수 있듯이 고장의 개수가 증가하면 repair rate는 감소한다. 표 1의 실험과 마찬가지로 고장이 6개이면 repair rate는 100%임을 볼 수 있다. 표 1에 비해서 전반적인 repair rate가 높게 형성되고 있는 것은 표준 편차를 3으로 설정하여 고장을 상대적으로 조밀하게 분포시켰기 때문이다. 또한 row priority보다 column priority의 결과가 조금 더 낮은 이유는 row spare가 2개로 column spare보다 작은 상태에서 무조건 column을 이용한 repair를 먼저 수행하기 때문에 spare column 뿐만 아니라 spare row의 효율도 함께 떨어졌기 때문이다.

표 3. 고장 개수에 따른 repair rate (단위 : %)
Table 3. Repair rates depended on the number of faults.

고장의 개수	ROW priority	COLUMN priority	Proposed BISR
6	100.0	100.0	100.0
7	96.6	91.4	98.7
8	89.0	74.3	94.5
9	70.7	57.0	84.1
10	49.2	43.4	66.7
11	29.3	33.9	49.0
12	19.6	17.1	26.8
13	10.9	10.8	16.4
14	7.7	7.8	11.4
15	3.4	2.9	4.1
20	0.0	0.0	0.1

표 4는 고장을 뿌릴 때의 편차를 조정함으로써 고장의 밀도를 조정함으로써 이에 따른 repair rate를 관찰하였다. 그림 8은 표 4의 결과를 그래프로 나타낸 것이다. 제안하는 BISR 알고리즘이 대조군의 알고리즘보다 더 높은 repair rate를 가지는 것을 볼 수 있다. 이 실험을 통해 고장이 조밀하게 분포할수록 repair rate가 우

표 4. 표준 편차에 따른 repair rate (단위 : %)
Table 4. Repair rates depended on the standard deviation.

Standard Deviation	ROW priority	COLUMN priority	Proposed BISR
2.0	77.1	69.5	85.1
2.5	66.2	55.2	79.7
3.0	53.0	43.6	66.8
3.5	40.0	33.1	55.3
4.0	31.4	23.3	41.1
4.5	26.2	17.7	35.3
5.0	19.5	14.7	28.2
5.5	15.2	10.5	21.3
6.0	11.1	8.0	17.9
6.5	9.8	7.4	14.1

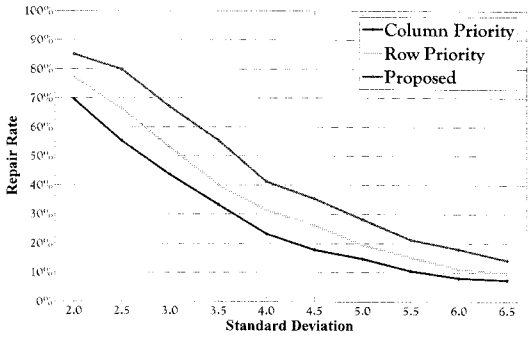

그림 8. 표준 편차에 따른 repair rate (단위 : %)
Fig. 8. Repair rates depended on the standard deviation.

표 5. 예비 메모리의 개수에 따른 repair rate (단위 : %)
Table 5. Repair rates depended on the number of spare memory.

The number of Spare Row	The number of Spare Column	ROW priority	COLUMN priority	Proposed BISR
2	2	1.6	1.3	1.8
2	3	14.6	12.3	20.3
3	2	15.7	17.0	23.5
3	3	50.7	52.3	68.6
2	4	50.7	42.0	66.2
4	2	43.7	52.9	61.5
3	4	87.1	77.6	93.0
4	3	77.1	85.5	92.9
4	4	92.5	93.3	98.7

수함을 확인할 수 있다. 또한 편차가 증가하여 고장이 넓게 분포함에 따라 repair rate가 감소하는 정도는 대체로 유사함을 확인할 수 있다.

예비 메모리의 개수 변화에 따라 repair rate가 변화하는 흐름을 분석하기 위하여 표 5는 spare row와 spare column의 개수에 따른 repair rate를 나타낸다. 표 5의 결과를 통해 전체 예비 메모리의 개수가 많아짐에 따라 repair rate 또한 향상됨을 알 수 있다. 또한 spare row나 column 중에서 어느 하나의 개수가 더 작을 경우에는 작은 쪽에 priority를 준 것이 repair rate가 더 높게 나타나는데 이는 그만큼 예비 메모리의 효율이 더 높아졌기 때문으로 풀이할 수 있다.

Ⅳ. 결 론

본 논문에서 제안한 BISR에서는 예비 메모리를 최대한 효율적으로 사용하기 위해 sign bit이라는 별도의 공간에 해당 주소에서 발생하는 고장의 개수를 기록하도록 하였다. 그리하여 sign bit에 기록된 고장 개수 정보를 토대로 고효율의 메모리 repair를 수행할 수 있도록 하였다. 제안된 BISR 방법론은 제한된 spare를 이용하여 고효율의 repair rate를 달성할 수 있는 방법을 제시하였다는 점에서 큰 의미가 있다. 고장이 발생한 메모리의 상황에 따라서, 하나의 예비 메모리를 통해 더 많은 수의 고장을 repair할 수 있는 쪽을 먼저 repair 함으로써 BISR의 효율을 개선한 것이다.

위의 결과를 분석해 볼 때, 제안하는 BISR 방법은 고장이 메모리의 전체 영역에 걸쳐서 널리 퍼져서 분포할 때보다, 고장이 특정 address의 row나 column에 집중적으로 분포할 때 더욱 효과적임을 알 수 있다. 또한 비교되는 기타 방법들보다 우수한 repair rate를 가지고 있음을 확인할 수 있다.

참 고 문 헌

[1] S. Hamdioui, G. Gaydadjiev and Ad J. van de Goor, "The State-of-art and Future Trends in Testing Embedded Memories", Record of the 2004 International Workshop on Memory Technology, Design and Testing, pp. 54-59, August 2004

[2] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories", *Electron*, pp. 175-179, Jan. 1984

[3] R. Rajsuman, "Design and test of large embedded memories : An overview", IEEE Design Test Comput., Vol. 18, No. 3, pp. 16-27, May 2001

[4] T. Chen and G. Sunada, "Design Self-Testing and Self-Repairing Structure for High Hierarchical Ultra-Large Capacity Memory Chips", IEEE Trans. On VLSI, Vol. 1, No. 2, pp. 86-95, 1995

[5] D. K. Bhavsar, "An Algorithm for Row-Column Self-Repair of RAMs and Its Implementation in the Alpha 21264", In Proc. of ITC, pp. 311-317. 1999

[6] T. Kawagoe, et. al, "A Built-in Self-Repair Analyzer (CRESTA) for embedded DRAMs", In Proc. of ITC, pp. 567-573, 2000

[7] J. F. Li, J. C. Yeh, R. F. Huang and C. H. Wu, "A Built-In Self-Repair Design for RAMs With 2-D Redundancy", IEEE Trans. On VLSI, Vol. 13, No. 6, pp. 742-745, June 2005

저 자 소 개

강 일 권(학생회원)
2006년 연세대학교 전기전자 공학과 학사 졸업.
2007년 현재 연세대학교 전기전자 공학과 석사 과정.
<주관심분야 : memory test, SoC 설계, DFT>

강 성 호(평생회원)
1986년 서울대학교 제어계측 공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터 공학과 박사 졸업.
1992년 미국 Schlumberger Inc. 연구원.
1994년 Motorola Inc. 선임연구원.
2007년 현재 연세대학교 전기전자공학과 교수.
<주관심분야 : SoC 설계 및 응용, DFT, SoC 테스트>