논문 2007-44SD-2-14

내장 메모리 자가 복구를 위한 여분의 메모리 분석 알고리즘

(Built-In Redundancy Analysis Algorithm for Embedded Memory Built-In Self Repair with 2-D Redundancy)

심 은 성*, 장 훈**

(Eun-Sung Shim and Hoon Chang)

요 약

최근 VLSI 회로 직접도가 급속도로 증가함에 따라 하나의 시스템 칩에 고밀도와 고용량의 내장 메모리가 구현되고 있다. 고장난 메모리를 여분의 메모리로 재배치함으로써 메모리 수율 향상과 사용자에게 메모리를 투명하게 사용할 수 있도록 제공할 수 있다. 본 논문에서는 고장난 메모리 부분을 여분의 행과 열 메모리 사용으로 고장난 메모리를 고장이 없는 메모리처럼 사용할 수 있도록 여분의 메모리 재배치 알고리즘을 제안하고자 한다.

Abstract

With the advance of VLSI technology, the capacity and density of memories is rapidly growing. In this paper we proposed reallocation algorithm. All faulty cell of embedded memory is reallocated into the row and column spare memory. This work implements reallocation algorithm and BISR to verify its design.

Keywords : BIRA, BISR, Embedded Memory

I. 서 론

최근 VLSI 회로 직접도가 급속도로 증가하고 있다. 이에 걸맞게 하나의 시스템 칩에 고밀도와 고용량의 내장 메모리가 구현되고 있다. SIA(Semiconductor Industry Association)의 기술보고서에서는 2014년에 내장 메모리가 차지하는 면적의 비중이 전체 SoC (System-On-Chip)의 94%에 이를 것으로 전망하고 있다[1].

내장 메모리에 대한 수율은 전체 SoC의 수율을 결정하는 요인이 될 것이며 SoC 테스트의 가장 중요한 부분이라고 할 수 있다. 내장 메모리를 효율적으로 테스트하기 위해서는 메모리 BIST(Built-In Self-Test) 로직이 필수적이며, 또한 구현 기술의 우수성과 효율성도 고려되어져야 한다[2 3].

내장 메모리의 비중이 이같이 높아짐에 따라 테스트의 효율성 뿐만 아니라 내장 메모리를 포함하고 있는 SoC의 수율 자체도 중요한 문제로 부각되고 있다. 따라서 고장복구(BISR : Built-In Self-Repair)가 가능한 메모리를 사용함으로써 내장 메모리 및 SoC의 수율을 크게 개선시키는 노력이 이루어지고 있으며 그 사용이 널리 확대되고 있다. 그림 1은 여분의 메모리를 사용 했을 때와 그렇지 않을 때의 수율을 비교한 그래프이다. 그림 1에서 알 수 있듯이 여분의 메모리를 사용하면 그렇지 않을 때보다 메모리 크기가 커질 수록 전체 메모리 수율은 현저히 높은 수율을 기대할 수 있다[4].

BISR은 BIST와 BISD(Built-In Self-Diagnostics), BIRA(Built-In Redundancy Analysis) 등과 같은 여러 가지 메커니즘을 수반하고 있다. 본 논문에서는 여분의 행과 열 메모리(row and column redundant memory)를 통한 내장 메모리의 고장난 부분을 여분의 메모리로 재

* 학생회원, ** 정회원 숭실대학교 컴퓨터학과
(Department of Computing, Soongsil University)
※ 본 연구보고서는 정보통신부의 출연금 등으로 수행한 정보통신연구개발사업의 연구결과입니다.
접수일자: 2006년11월21일, 수정완료일: 2007년1월16일

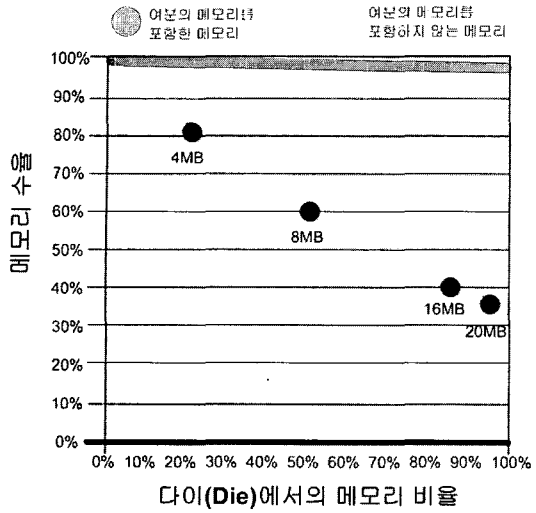


그림 1. 메모리 크기별 수율
Fig. 1. Memory sizes versus yield.

배치 할 수 있도록 알고리즘을 제안하였다. 내장 메모리의 행 혹은 열에 고장이 났을 경우 여분의 메모리 행이나 열만을 이용해 고장난 부분을 여분의 메모리 행 혹은 열로 재배치하는 경우 보다 여분의 메모리에 행과 열 모두를 이용해 고장난 부분을 행과 열로 재배치하는 것이 하나로만 재배치하는 것 보다는 좀 더 효율적이다[5-7]. 하지만 행과 열을 이용해 여분의 메모리를 재배치하는 알고리즘은 NP-hard 문제이기 때문에 매우 어렵다[8].

여분의 메모리 재배치를 위해 테스트 장비(ATE : Automatic Test Equipment)를 이용한 여분의 메모리 분석 알고리즘들이 제안되어져 왔다[9-10]. BISR의 구조는 회로에 내장되어야 하는데 제안되어졌던 알고리즘들은 많은 비용이 소비되며 내장 회로 내에 적용할 수 없다. 또한 [11]에서는 여분의 메모리 워드를 이용해 BISR 구조를 제안하였다. 이 방법은 여분의 행 메모리만을 이용해 여분의 메모리 분석이 요구 되지 않는다. [12]에서 제안한 알고리즘은 최적의 해결 방법을 찾기 위해 소모되는 시간이 너무 오래 걸리며 복잡한 계산이 수행된다.

BIST를 통해 고장난 부분을 고장난 주소와 고장난 데이터의 위치를 확인하고 그에 대한 정보를 이용해 행과 열을 가지는 여분의 메모리(row and column redundant memory)에 내장 메모리의 고장난 부분을 여분의 행과 열 메모리 재배치하기 위해 제안한 논문이다. 여분의 메모리로 고장난 데이터 정보를 재배치하기 위해 BIST 작업은 우선되어 졌다고 가정한다.

본 논문의 구성은 다음과 같다.Ⅱ에서 제안하는 재배치 알고리즘을 살펴보고, Ⅲ에서는 재배치 정보를 이용한 자가 복구 구조를 설명한다. Ⅳ에서 실험결과를 알아본 후 Ⅴ결론을 맺는다.

Ⅱ. 제안하는 메모리 재배치 알고리즘

1. 재배치 알고리즘

메인 메모리가 고장이 나면, 고장난 부분을 여분의 (Row, Column) 메모리 부분으로 재배치 메인 메모리의 고장난 위치와 그에 주소를 재배치 처리 공간이 적으면, 고장난 위치에 대한 재배치가 가능하더라도 고장난 정보를 담을 수 없어 처리 오류가 발생한다. 만약 저장 공간이 커지게 되면 오버헤드의 문제가 생기게 됨으로 고장난 위치와 주소의 정보를 저장하기 위한 공간은 여분의 행 메모리 개수와 여분의 열 메모리의 개수의 합으로 한다. 메인 메모리의 고장이 여분의 행 메모리 개수와 여분의 열 메모리 개수의 합 이상이 되면 재배치 공간이 충분치 않아 재배치 실패가 된다.

메인 메모리의 고장난 위치와 고장난 주소를 저장하고 있는 위치에서 행을 기준으로 가장 고장이 많은 하나의 행을 선택한다. 열을 기준으로 해도 알고리즘의 결과는 같게 나온다. 가장 고장이 많은 행은 여분의 행 메모리로 재배치한다. 가장 고장이 많은 행의 고장난 셀의 개수가 같게 나오게 되면 처음 발생한 행을 선정한다. 다음 고장이 있는 주소의 첫 번째 고장 셀에서 그 주소의 행에 고장난 셀 개수와 열에 고장난 셀 고장 개수를 비교한다. 고장의 개수가 행이 많다면 여분의 행 메모리로 재배치하고 열의 개수가 많다면 열의 셀 위치를 여분의 열 메모리로 재배치한다.

행에 고장난 셀의 개수와 열에 고장난 고장난 고장 개수를 비교하기 전에 크게 두 가지를 먼저 계산한다. 첫 번째로 가장 고장이 많은 행의 고장 셀 위치와 열의 위치 비교를 하기 위한 고장의 열 위치가 겹치게 되는지를 확인하다. 가장 고장이 많은 행은 무조건 여분의 행 메모리로 재배치하기 때문에 그것과 고장의 위치가 겹치게 되는 열의 셀 위치에 있는 고장을 여분의 열 메모리로 재배치할 경우 그 고장은 여분의 행 메모리와 여분의 열 메모리 두 곳으로 재배치 되게 된다. 이렇게 되면 여분의 행 메모리나 여분의 열 메모리 두 곳 중 한곳에서 데이터가 업데이트 되었을 때 나머지 곳도 같이 업데이트를 해주어야 하는 복잡한 상황이 발생하게 된다. 따라서 가장 고장이 많은 행의 고장 위치와 고장난 셀이 겹치게 되면 무조건 여분의 행 메모리로 재배

치한다.

두 번째는 같은 주소에 고장난 셀이 2개 이상일 경우 하나의 셀 고장이 여분의 열 메모리로 재배치 되었다면 고장난 셀의 공유를 막기 위해 같은 주소의 모든 셀들은 여분의 열 메모리로 재배치 되어야 한다. 또한 고장난 셀의 열 위치가 다른 주소에 의해 여분의 열 메모리로 재배치 되었다면 역시 고장의 공유를 막기 위해 여분의 열 메모리로 재배치 되어야 한다.

이와 같은 재배치 알고리즘은 아래의 그림 2에서와 같이 의사 코드로 나타내었다. BIST를 통해 메인 메모리의 고장난 위치와 고장난 위치를 저장한 후 고장난 위치를 여분의 메모리로 재배치하기 위한 알고리즘이 시작된다. 고장난 정보들을 통해 가장 고장이 많은 셀을 가지고 있는 셀의 개수의 행을 선정한다(Search_TheMostFaultyRowCell()). 고장이 가장 많은 셀을 포함하고 있는 주소(행)는 여분의 행 메모리로 재배치(Reallocation_RowSpareMemory())한다. 가장 고장이 많은 셀을 포함하는 주소가 아니면 가장 고장난 셀의 개수가 많은 행과 열의 셀 고장 위치가 겹치(TheMostFaultyRowCell_Overlap)는지 판단한다. 만약 겹치면 여분의 행 메모리에 재배치한다.

가장 고장이 많은 셀과 겹치지 않는다면, 같은 주소에 있는 다른 고장 셀이 이전에 여분의 열 메모리로 재배치(Already_ReallocationColumnSpareMemory_atLocation) 되었거나 고장 위치가 다른 주소에 의해 여분의 열 메모리로 재배치(Already_ReallocationColumnSpareMemory_atFaultyAddress) 되었다면 여분의 열 메모리로 재배치한다.

고장난 셀이 위의 세 가지 경우가 아니라면 고장난 셀의 행의 개수(TheNumberOfRowFaultyCell)와 열의 개수(TheNumberOfColumnFaultyCell)의 개수를 비교해 열의 고장난 셀 개수가 크면 여분의 열 메모리에 재배치하고, 그렇지 않으면 여분의 행 메모리에 재배치한다. 재배치에 대한 간다한 예를 그림 3에서 보이고 있다. 이 예는 5bit 주소와 행 여분의 메모리, 열 여분의 메모리 각각 4개씩 존재할 때, 재배치 방법을 설명하고 있다.

Pseudo Code of The BISA Algorithm

```
01 : Search_TheMostFaultyRowCell();
02 : IF (TheMostFaultyRowCell) THEN
03 :     Reallocation_RowSpareMemory();
04 : ELSE
05 :     IF (TheMostFaultyRowCell_Overlap) THEN
06 :         Reallocation_RowSpareMemory();
07 :     IF (Already_ReallocationColumnSpareMemory_atLocation ||
08 :         Already_ReallocationColumnSpareMemory_atFaultyAddress) THEN
09 :         Reallocation_ColumnSpareMemory();
10 :     IF (TheNumberOfRowFaultyCell < TheNumberOfColumnFaultyCell) THEN
11 :         Reallocation_ColumnSpareMemory();
12 :     ELSE
13 :         Reallocation_RowSpareMemory();
```

그림 2. 재배치 의사 코드

Fig. 2. Pseudo Code of Reallocation Algorithm.

2. 재배치 알고리즘의 예

그림 3은 BIST에서 고장을 확인하는 즉시 Reallocation Logic 안의 Save Logic에 고장난 주소와 고장난 위치 정보가 저장된다. 고장난 정보 중에 가장 고장이 많은 셀을 포함하고 있는 행의 주소(고장난 주소)를 찾아 행 여분의 메모리에 재배치한다. 그림 3의 예제에서는 고장난 주소 10100을 여분의 행 메모리(1000)에 재배치한다. Save Logic 안에서 다음 고장난 정보가 있는 주소(11101)에서 첫 번째 고장 셀의 열(02)의 고장난 셀 개수와 행의 고장 개수를 비교한다. 하지만, 주소(1101) 열의 셀 위치 05와 06, 07, 21번 위치에 고장이 있다. 이 위치는 가장 고장이 많은 주소(10100)의 열 위치와 겹치는 고장의 위치이다. 따라서 주소(11101)는 여분의 행 메모리(0100)에 재배치한다.

다음 주소(00010)의 열 위치(09)에서는 가장 고장이 많은 행의 열 고장난 셀의 위치와 겹치는 부분이 없고, 같은 주소에서 이전에 여분의 열 메모리로 재배치 한적이 없어 열 위치의 고장난 셀 개수 3개와 행의 고장난 셀 개수 2개를 비교한다. 열의 고장난 셀 개수가 많음으로 열의 09번은 여분의 열 메모리(1000) 부분으로 재배치한다. 같은 주소(00010)의 다음 고장난 셀의 열 위치 25번은 이전 위치 09번에서 열 여분의 메모리로 재배치 되었기 때문에 열의 위치 25번도 열 여분의 메모

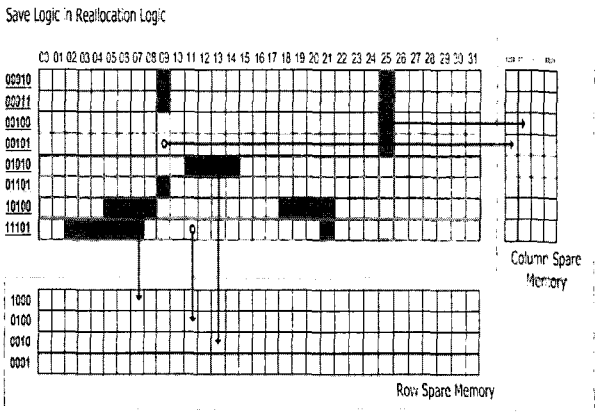


그림 3. 재배치 알고리즘 예

Fig. 3. Example of Reallocation Algorithm.

Row / Column Flag	Spare Memory Address	Faulty Data	Faulty Address
0	0001	-	10100
0	0010	-	11101
1	1100	00000000_01000000_00000000_01000000	00010
1	1100	00000000_01000000_00000000_01000000	00011
1	0100	00000000_00000000_00000000_01000000	00100
1	0100	00000000_00000000_00000000_01000000	00101
0	0100	-	01010
1	1000	00000000_01000000_00000000_00000000	01101

그림 4. 재배치 데이터
Fig. 4. Reallocation Data.

리(0100)로 재배치된다.

다음 주소 고장 위치 00011과 01101의 고장난 셀 위치는 열의 09번이다. 09번은 이전 주소(00010)에서 여분의 열 메모리로 재배치 되었기 때문에 주소 00011과 01101의 09번도 여분의 열 메모리(1000)로 재배치한다. 고장 주소 00011의 열 고장난 셀 위치 25번째 역시 여분의 열 메모리(0100)로 재배치한다.

고장난 주소 00100과 00101도 주소 00010에서 열 고장난 셀 위치 25번이 여분의 열 메모리(0100)로 재배치되어 주소 00100과 00101의 열 고장난 셀 위치 25번이 여분의 열 메모리(0100)로 재배치한다. 다음 주소 01010의 열 고장난 셀 위치 11번은 행과 열의 고장난 셀 개수가 각각 4개와 1개이므로 여분의 행 메모리(0010)로 재배치한다.

재배치된 데이터 정보들은 Reallocation Logic에서 그림 4에서와 같이 저장된다. 재배치된 최종 정보는 메인 메모리에 고장난 위치(Faulty Data)와 메인 메모리에 고장난 주소(Faulty Address), 여분의 메모리로 재배치 될 주소(Spare Memory Address), 여분의 행 메모리 또는 여분의 열 메모리로 재배치될지에 대한 Flag(Row/Column Flag) 총 4개의 정보가 재배치 정보로 Reallocation Logic으로부터 출력된다.

Row / Column Flag 정보는 '0'이면 여분의 행 메모리로 재배치되고, '1'이면 여분의 열 메모리로 재배치된다. 여분의 메모리 주소(Spare Memory Address)는 여분의 메모리로 재배치될 수 있는 개수 만큼 비트수를 갖는다. 여분의 행 메모리로 재배치되는 경우는 여분의 행 메모리 전체를 재배치하기 때문에 고장난 위치를 확인할 필요가 없어 '-'로 표시하며, 여분의 열 메모리의 경우는 고장난 위치의 셀을 재배치하기 때문에 고장난 메인 메모리의 위치를 파악해야 한다. 따라서 고장난 셀의 위치는 '1'로 고장이 없는 셀의 위치는 '0'로 나타낸다. 마지막으로 메인 메모리의 고장난 주소의 위치는 Faulty Address에 표현한다.

여분의 메모리의 주소 표현은 2^n으로 나타내지 않고, 여분의 메모리 개수를 주소로 나타낸다. 예를 들어 여분의 열 메모리 개수가 4개라면 일반적으로는 2^2(00, 01, 10, 11)로 나타낸다. 하지만 제안하는 구조에서는 접근 가능한 부분을 1000, 0100, 0010, 0001로 나타낸다. 일반적인 방식으로는 2비트로 모든 것을 나타낼 수 있지만, 제안하는 구조는 4비트가 필요하다. 이렇게 표현한 이유는 동시에 고장난 위치가 2(00, 01)곳일 경우 일반적인 방식으로는 2비트씩 2번 전송이 필요하다. 제안하는 방식은 주소를 OR 연산 후 4비트(1100)를 1번만 전송하면 된다. 또한 여분의 메모리 주소를 저장하고 있는 총 비트수에서 일반적인 방식은 8비트가 소비되면서 여분의 메모리에 접근하기 위한 중복된 데이터 값들이 저장되어지지만, 제안하는 주소 체계는 4비트면 중복 저장을 피할 수 있다.

III. 재배치 정보를 이용한 자가 복구 구조

1. 재배치 고장복구 구조

Reallocation Logic을 통해 고장난 메모리의 셀 위치와 주소를 여분의 메모리로 재배치 할 수 있는 정보로 가공한다. 가공된 최종 정보는 메인 메모리에 고장난 위치와 메인 메모리에 고장난 주소, 여분의 메모리로 재배치 될 주소, 여분의 행 메모리 또는 여분의 열 메모리로 재배치될지에 대한 Flag 총 4개의 정보가 재배치 정보로 Reallocation Logic으로부터 출력된다. 출력된 재배치 정보를 이용해 사용자가 고장난 주소로 데이터를 메인 메모리에 입력, 출력 했을 때 고장이 없는 것처럼 사용할 수 있게 데이터를 메인 메모리에 고장 위치로 입력, 출력 시키지 않고, 여분의 메모리를 이용해 고장이 없는 메인 메모리를 사용할 수 있게 한다.

그림 5는 BISR의 전체 구조도이다. 그림 5에서 나타

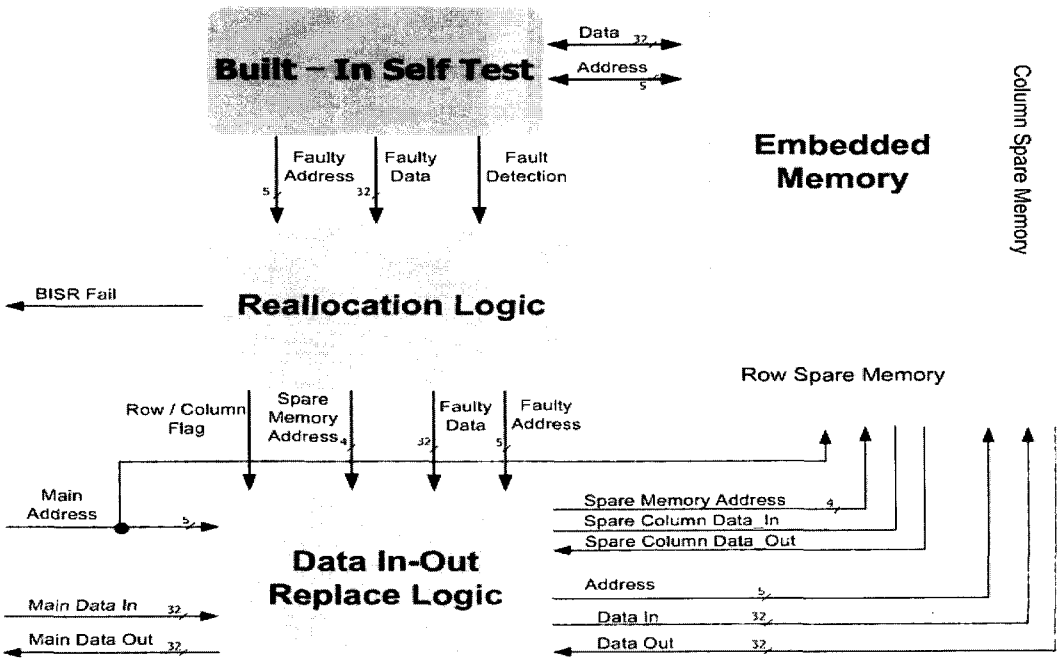


그림 5. BIST와 Reallocation Logic, Data In-Out Replace Logic의 전체구조도
Fig. 5. Block Diagram of BIST, Reallocation Logic and Data In-Out Replace Logic.

내는 비트 수들은 메인 메모리의 주소 5비트와 데이터 32비트로 이루어져 있으며, 여분의 행 메모리와 여분의 열 메모리를 각각 4개 즉, 4비트로 표현한 구조도이다. BIST를 통해 메인 메모리의 고장난 위치와 고장난 주소를 Fault Detection 신호와 함께 Reallocation Logic에 보내지게 된다.

Reallocation Logic을 통해 재배치 정보를 받은 Data In-Out Replace Logic은 메인 메모리의 고장난 위치에 접근하는 데이터를 정상적인 데이터로 교체 해주는 일을 수행한다. Data In-Out Replace Logic에는 Fault Address Comparator와 Data Replace Logic, Spare Memory Address Reallocation Logic이 존재한다.

Fault Address Comparator는 고장난 주소와 사용자가 메인 메모리에 접근 하고자 하는 메인 메모리와의 비교를 통해 주소에 해당하는 데이터 고장의 유무를 판별한다. 만약 데이터 고장이 없는 주소라면 Data In-Out Logic을 거치지 않고 메인 메모리에 접근한다.

Data Replace Logic은 재배치 데이터가 여분의 행 메모리 또는 여분의 열 메모리로 재배치되는 지를 구분(Row/Column Flag)한다. 고장난 데이터가 여분의 행 메모리로 재배치 될 경우는 데이터 전체(32비트)가 여분의 행 메모리로 재배치되기 때문에 Data Replace Logic을 거쳐 여분의 행 메모리로 재배치되거나 여분의 행 메모리로부터 치환되어 출력된다. 만약 여분의 열 메모리로 재배치되면 Reallocation Logic의 고장난 위치(Faulty Data)를 이용해 사용자가 입력한 데이터가 메인 메모리의 고장난 위치에 접근하는 데이터 위치를 파악해 메인 메모리의 고장난 위치에 접근하는 데이터를 추출하여 Spare Memory Address Reallocation Logic으로 보내는 역할을 한다.

Spare Memory Address Reallocation Logic은 여분의 행 메모리나 여분의 열 메모리의 주소를 선택해 준다. Data Replace Logic에서 입력받은 재배치할 데이터를 열 여분의 메모리에 재배치하는 역할을 한다.

Ⅳ. 실험결과

본 논문에서 제안하는 재배치 알고리즘과 BISR 구조에 대한 설계 검증은 VerilogHDL로 기술하여 구현하였다. 구현에 대한 검증은 Xilinx사의 Xilinx Foundation에서 제공하는 Simulator를 사용하여 RTL 검증을 하였다. 주소 8비트와 데이터 32비트를 갖는 메인 메모리와 여분의 행 메모리 6개, 여분의 열 메모리 10개를 갖는 여분의 메모리를 사용하였다.

BIST를 통해 메인 메모리 고장 1 (주소: A8h, 고장 위치: 0000002Bh)과 고장 2(주소:AAh, 고장 위치:

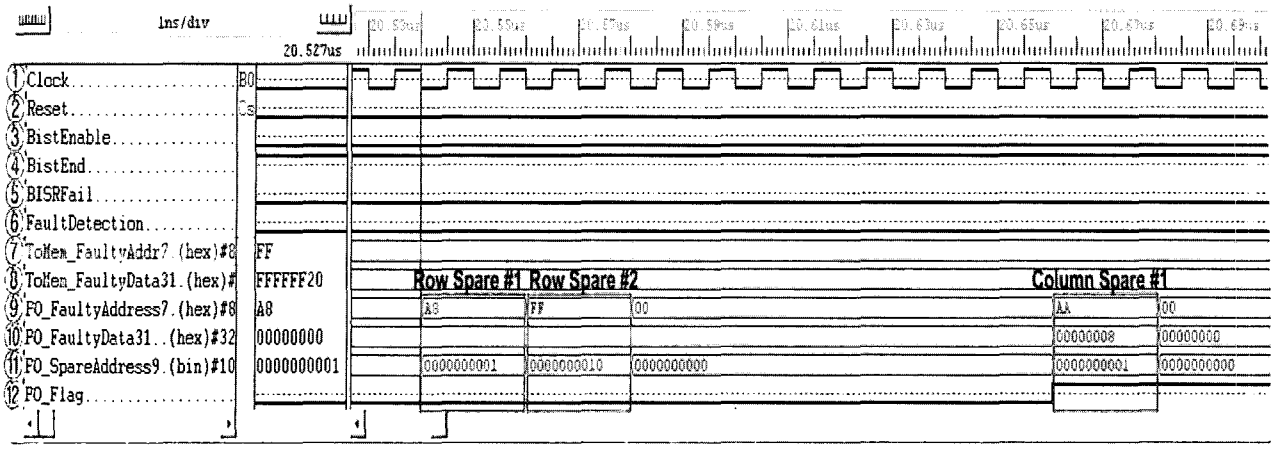


그림 6. 재배치 회로의 결과 파형
Fig. 6. Reallocation Wave Form.

00000001h), 고장 3(주소:FF, 고장 위치:FFFFFF20h)을 인식하게 되고 그 고장에 대해 제안하는 재배치 알고리즘을 통해 그림 6에서 처럼 여분의 메모리로 재배치 할 수 있도록 재배치 정보를 출력하게 된다.

그림 6에서는 총 12개의 신호가 있다. ①과 ②는 각각 Clock, Reset 신호이다. BIST 동작시에는 BistEnable (③) 신호가 '1'이 된다. BIST가 동작할 때 고장을 감지하게 되면, FaultDection(⑥) 신호를 통해 Reallocation Logic에 '1'을 인가한다. FaultDection(⑥) 신호를 인가한 후 메인 메모리의 고장난 주소(⑦:ToMem_FaultyAddr)와 위치(⑧:ToMem_FaultyData)를 함께 Reallocation Logic에 보낸다. BIST 동작이 끝나면 ③ 신호인 BistEnable 신호가 '0'이 되면서 BistEnd(④) 신호는 '1'이 된다.

Reallocation Logic을 통해 재배치가 끝나게 되면 고장난 주소(⑨:PO_FaultyAddress)와 고장난 위치(⑩:PO_FaultyData), 여분의 메모리 재배치 주소(⑪:PO_Spare Address), 여분의 행 메모리로 재배치되었는지 여분의 열 메모리로 재배치되었는지에 대한 구분 신호(⑫:PO_Flag)를 출력하게 된다. 고장 1과 고장 2, 고장 3은 그림 6에서와 같이 여분의 행 메모리로 두 곳(Row Spare #1, Row Spare #2), 여분의 열 메모리로 한 곳(Column Spare #1) 재배치된 것을 확인 할 수 있다. 만약 여분의 메모리 공간이 부족해 재배치가 불가능할 경우는 ⑤ 신호인 BISRFail 신호가 '1'로 활성화 된다.

그림 7은 메인 메모리의 크기 별로 임의의 고장을 삽입한 후 본 논문에서 제안하는 재배치 알고리즘을 사용해 재배치에 필요한 여분의 메모리 개수를 나타낸 그림이다. 그림에서 확인할 수 있듯이 여분의 행 메모리가 여분의 열 메모리 보다 더 많이 필요하다. 그 이유는 재배치 알고리즘을 행 기준으로 동작했기 때문이다. 만약 열 기준으로 동작했을 경우는 여분의 열 메모리가 더 필요할 것이다.

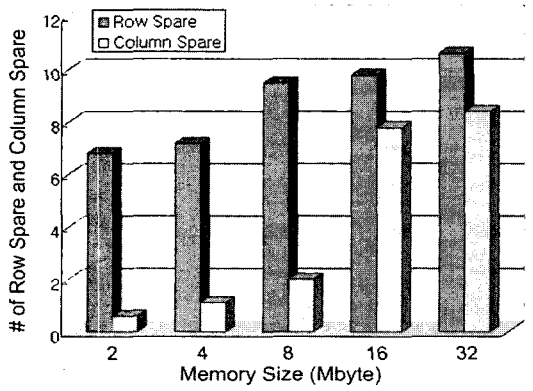


그림 7. 메모리 크기에 따른 여분의 메모리 개수
Fig. 7. The Number of Spare Memory for Memory Size.

V. 결 론

본 논문에서는 고장난 내장 메모리를 사용하는 사용자가 여분의 메모리를 이용해 고장이 없는 내장 메모리처럼 사용할 수 있도록 고장난 내장 메모리의 부분을 여분의 메모리로 재배치하였다. 여분의 메모리를 효율적으로 다루기 위해 행과 열로 나누어 재배치 알고리즘을 제안하였다.

행과 열로 재배치하는 방법으로는 고장난 하나의 셀을 포함하고 있는 행의 고장 개수와 열의 고장 개수를 비교해 고장난 셀의 개수가 많은 쪽 여분의 행 혹은 열 메모리에 재배치를 한다. 만약 가장 고장이 많은 셀의

위치와 고장난 셀이 겹치게 되면 여분의 행 메모리로 재배치를 하고(행을 기준으로 했을 때), 이전에 열로 재배치가 되었으면 여분의 열 메모리로 재배치를 하는 방법을 사용하였다.

본 논문에서 제안한 재배치 알고리즘을 사용하여 BISR 회로에 적용시켰을 경우 메모리 고장에 대해 정상 동작시키며, 메모리의 물리적 구조에 독립적인 회로로 존재한다. 고가의 메모리를 여분의 메모리로 재사용함으로서 메모리의 수율을 증가 시킬 수 있을 것이다.

참 고 문 헌

[1] Allan A., Edenfeld D., Joyner W.H. Jr., Kahng A.B., Rodgers M., and Zorian, Y., "2001 technology roadmap for semiconductors," *Computer*, vol. 35, pp. 42-53, January 2002.

[2] Jone W.B., Huang D.C., Wu S.C., and Lee K.J., "An efficient BIST method for distributed small buffers," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, pp. 512-515, August 2002.

[3] Hamdioui S., and van de Goor A.J., "Thorough testing of any multiport memory with linear tests," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 217-231, February 2002.

[4] Hamdioui, S., Gaydadjiev, G., and van de Goor, A.J., "The state-of-art and future trends in testing embedded memories," *Records of the 2004 International Workshop, Memory Technology, Design and Testing*, pp. 54-59, 9-10 August 2004.

[5] Horiguchi M., Etoh J., Aoki M., Itoh K., and Matsumoto T., "A flexible redundancy technique for high-density DRAMs," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 12-17, January 1991.

[6] Ilyoung Kim, Zorian Y., Komoriya G., Pham H., Higgins F.P., and Lewandowski J.L., "Built in self repair for embedded high density SRAM," *Proceedings of International Test Conference*, pp. 1112-1119, 18-23 October 1998.

[7] Heon Cheol Kim, Dong Soon Yi, Jin Young Park, and Chang Hyun Cho, "A BISR (built-in self-repair) circuit for embedded memory with multiple redundancies," *International Conference on VLSI and CAD*, pp. 602-605, 26-27 October 1999.

[8] Sy Yen Kuo, and Fuchs W.K., "Efficient Spare Allocation in Reconfigurable Arrays" *Conference on Design Automation*, pp. 385-390, 29-2 June 1986.

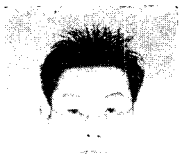
[9] Chin Long Wey, and Lombardi F., "On the Repair of Redundant RAM's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, pp. 222-231, March 1987.

[10] Wei Kang Huang, Yi Nan Shen, and Lombardi F., "New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, pp. 323-328, March 1990.

[11] Schober V., Paul S., and Picot O., "Memory built-in self-repair using redundant words," *Proceedings of International Test Conference*, pp. 995-1001, 30 October-1 November 2001.

[12] Shyue Kung Lu, Yu Chen Tsai, Hsu C. H., Kuo Hua Wang, and Cheng Wen Wu, "Efficient built-in redundancy analysis for embedded memories with 2-D redundancy," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 14, pp. 34-42, January 2006.

저 자 소 개



심 은 성(학생회원)

2003년 한서대학교 컴퓨터정보학과 학사 졸업

2005년 8월 숭실대학교 대학원 컴퓨터학과 석사 졸업

2005년 9월 ~현재 숭실대학교 대학원 컴퓨터학과 박사 과정

<주관심분야 : VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD>

장 훈(정회원)

1987년 서울대학교 공대 전자공학과 학사 졸업.

1989년 서울대학교 공대 전자공학과 석사 졸업.

1993년 University of Texas at Austin 졸업.

1991년 IBM Inc. Senior Member of Technical Staff.

1993년 Motorola Inc. Senior Member of Technical Staff.

1994년~현재 숭실대학교 컴퓨터학부 부교수.

<주관심분야: 통신, 컴퓨터, 신호처리, 반도체>