여분의 메모리를 이용한 SRAM 재사용 설계 및 검증

준회원 심 은 성*, 정회원 장 훈**

SRAM Reuse Design and Verification by Redundancy Memory

Eun sung Shim*, Hoon Chang** *Regular Members*

요 약

본 논문에서는 내장된 메모리의 자체 테스트를 통한 메모리 고장 유무 확인과 더불어 메인 메모리의 고장난 부분을 여분의 메모리로 재배치하여 사용자로 하여금 고장난 메모리를 정상적인 메모리처럼 사용할 수 있도록 BISR(Build-In Self Repair) 설계 및 구현을 하였다. 메인 메모리를 블록 단위로 나누어 고장난 셀의 블록 전체를 재배치하는 방법을 사용하였으며, BISR은 BIST(Build-In Self Test) 모듈과 BIRU(Build-In Remapping Unit) 모듈로 구성된다. 실험결과를 통해 고장난 메모리를 여분의 메모리로 대체하여 사용자가 메모리를 사용함에 있어서 투명하게 제공하는 것을 확인 할 수 있다.

Key Words : BISR, BIST, SRAM, SOC, Embedded Memory

ABSTRACT

In this paper, built-in self-repair(BISR) is proposed for semiconductor memories. BISR is consisted of BIST (Buit-in self-test) and BIRU(Built-In Remapping Uint). BIST circuits are required not only to detect the presence of faults but also to specify their locations for repair. The memory rows are virtually divided into row blocks and reconfiguration is performed at the row block level instead of the traditional row level. According to the experimental result, we can verify algorithm for replacement of faulty cell.

Ⅰ. 서 론

반도체 공정기술이 발달하고 VLSI가 고성능화됨에 따라 점점 더 많은 수의 코어들이 SOC(System on Chip)화 되고 있다. 그 중 내장 메모리는 전체 SOC 트랜지스터 수의 80~90%를 차지하고 있으며 SOC 개발시 내장 메모리의 테스트가 새로운 문제로 대두되고 있다[1]. 이로 인해 여느 때보다 SOC 내에서 메모리 테스트의 중요성은 높아지고 메모리의 고장의 유무만을 판단하는 것을 떠나 고가의 메모리을 여분의 메모리로 재사용함으로써 메모리 수율을 증가 시킬 수 있다[2-5]. 메모리에 자가 고장 복구(BISR : Built-In Self Repair) 회로를 추가함으로써 고장난 메모리를 다시 사용할 수 있다. BISR의 구성은 내장 메모리의 고장 유무를 판단하기 위해 가장 널리 사용되는 방법인 메모리 BIST(Built-In Self Test) 기법과 고장난 메인 메모리의 셀을 여분의 메모리에 적절히 배치 할 수 있도록 하는 자가 재배치(BIRU : Built-In Remapping Unit) 회로가 필요하다. 마지막으로 BIST와 BIRU의 설정값을 가지고 정상 적인 메모리 동작에서 BISR의 회로를 이용하면 메인 메모리의 고장난 셀의 부분을 여분의 메모리로 재배치하는 방법을 사용해 전반적인 SOC 수율 증가를 위한 IPs 구축개발을 가져오게 된다[6-7].

여분의 메모리를 구현하는 방법은 Hard Redundancy

* 숭실대학교 대학원 컴퓨터학과 석사과정 (esshim@ssu.ac.kr), ** 숭실대학교 컴퓨터학과 컴퓨터구조 연구실 (hoon@ssu.ac.kr)
논문번호 : KICS2005-01-053, 접수일자 : 2005년 1월 30일
※본 연구는 시스템 집적 반도체 기반 기술 개발 사업을 통해 지원된 개발 결과물입니다.

[8]와 Soft Redundancy[9]로 나누어진다. Hard Redundancy는 재배치할 메모리 원소의 정보의 저장을 퓨즈로써 구현하는 것으로 퓨즈에는 레이저 퓨즈, 전기적 퓨즈, EPROM 메모리 등이 있다. 레이저 퓨즈, 전기적 퓨즈, EPROM 등은 여분의 메모리가 메인 메모리의 값들을 저장할 수 있도록 하드웨어적으로 레이저나 전기적으로 절단해 칩으로 만드는 것이다. 이렇게 칩으로 만들어진 여분의 메모리는 하나의 메모리에 대해서 반영구적으로 여분의 메모리를 사용할 수 있지만, 외부의 다른 테스트 장비가 필요하며 다른 메모리에 대한 여분의 메모리 사용이 용이하지 못하다.

반면 Soft Redundancy 방법은 칩에 전원이 인가되면 여분의 메모리와 고장이 있는 메인 메모리 사이의 재배치 계산을 하기 위해 BIST 및 BIRU 모듈을 실행해야 하는 번거로움이 존재한다. 그러나 외부의 테스트 장비를 사용하지 않아도 되고, 여분의 메모리를 융통적으로 모든 메인 메모리에 적용시킬 수 있다는 장점을 가지고 있다. 본 논문에서는 Soft Redundancy의 방법으로 설계 및 구현한다.

본 논문의 구성은 다음과 같다. Ⅱ에서 기존의 연구를 살펴보고, Ⅲ에서는 제안하는 BISR의 개념을 설명한다. Ⅳ에서 BISR의 설계구성을 알아보고, Ⅴ에서 실험결과를 알아본 후 Ⅵ에서 결론을 맺는다.

Ⅱ. 기존 연구

SOC 개발시 내장 메모리를 테스트하기 위해 가장 널리 사용되는 방법은 메모리 BIST이다[10-11]. 메모리 BIST 기법은 칩의 내부에 테스트 회로를 내장하여 자체적으로 테스트를 수행하는 기법으로써 부수적인 면적의 증가와 같은 오버헤드를 갖게 되지만, 각 모듈별로 자체적인 테스트가 수행되므로 전체 시스템의 테스트에 있어서 테스트의 복잡도가 크게 줄어들고, 고가의 외부 테스트 장비를 사용하지 않고도 빠른 시간에 테스트를 수행할 수 있다는 장점 때문에 내장된 메모리의 테스트를 위한 BIST 기법의 사용이 보편화 되어 있다. 더욱이 내장된 메모리의 크기가 점차 커져감에 따라 내장된 자체 테스트 회로의 단점인 면적 오버헤드가 상대적으로 크게 감소하게 되므로 그 장점이 더욱 부각되고 있다. 본 논문에서는 메모리 BIST 구조와 메모리 BIRU를 통해 고장을 파악하고 여분의 메모리로 대체할 수 있도록 BISR 구조를 제안한다. 여분의 메모리를 사용해 고장난 메모리를 재배치하기 위한 방법으로 그림 1와 같은 개념을 사용했었다[12-13].

[12-13]에서의 방법은 메인 메모리와 여분의 메모리를 블록 단위로 나눈 뒤에 메인 메모리의 블록 안의 셀이 고장이 일어났을 경우 고장난 셀이 포함된 블록 전체를 여분의 메모리의 블록으로 교체 하는 방법을 사용한다. 여분의 메모리로 대채 할 때 메인 메모리와 여분의 메모리의 같은 열의 블록으로 재배치한다. 만약에 다른 주소들의 같은 열에 블록들로 재배치 될 경우 여분의 메모리가 모두 채워지게 되면 다른 행에 여분의 메모리가 있어도 더 이상 재배치가 이루어 지지 않는 문제가 있다.

이 방법은 원천적인 메모리 설계시에는 유용할지 모르나 메모리가 만들어진 후에는 제안한 방법으로는 테스트하기는 어렵다. 즉, 메모리의 입력, 출력 핀만 가지고 테스트하기가 곤란하다. 본 논문에서는 이러한 문제를 해결하고자 BISR을 설계 및 구현을 제안한다.

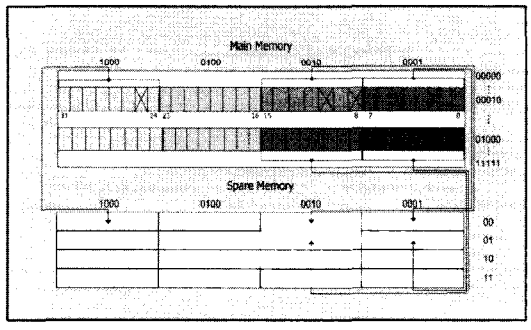

그림 1. [12-13]에서의 블록 구조의 기본개념

Ⅲ. 자가 고장 복구 개념

메인 메모리의 특정 주소의 하나의 셀 또는 여러 개의 셀이 고장 나면 그 셀에 포함하는 블록을 전부 여분의 메모리의 블록으로 재배치한다. 본 논문은 그림 2에서와 같은 블록 재배치 개념을 이용해 여분의 메모리를 사용한다. 본 논문에서 제안한 BIST와 BIRU, BISR은 5bit 주소와 1워드(32bit)를 갖는 메인 메모리와 2bit 주소와 1워드(32bit)를 갖는 여분의 메모리를 1워드(32bit)에 4개의 블록(8bit)으로 나누어 설명한다. 그림 3은 BIST와 BIRU, BISR, 여분의 메모리(Spare Memory), 메인 메모리(Main Memory)의 관계를 보인다.

BIST 회로에서 메인 메모리에 테스트 주소(Test Address)가 가르키는 지점에 테스트 데이터(Test Data)를 인가한 후 Test Data와 메인 메모리에서

읽어온 데이터(Reference Data)를 비교해 메인 메모리의 고장 유무를 판단한다. 만약 고장이 나면 고장난 셀을 포함하고 있는 블록의 개수를 Faulty Group Count를 통해 BIRU에 전달한다.

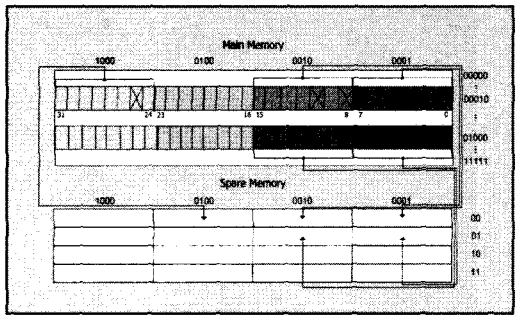

그림 2. 블록 재배치의 기본 개념

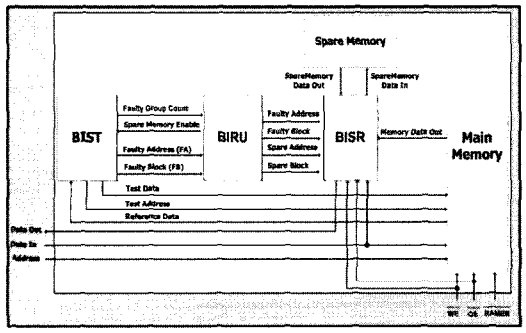

그림 3. BIST와 BIRU, BISR, 여분의 메모리, 메인 메모리의 구조

BIRU에서 여분의 메모리의 상태를 파악한 후 재배치할 수 있으면, Spare Memory Enable 신호를 통해 액티브 하이 값을 BIST에 전달한다. 이 신호를 받은 BIST 회로는 메인 메모리에 고장난 주소(FA)와 고장난 셀을 포함한(FB)정보를 BIRU에 준다. BIST 동작을 모두 마치게 되면 BIRU는 메인 메모리의 고장에 대한 정보(FA, FB, SA, SB)를 BISR에 넘겨주게 된다.

Ⅳ. 자가 고장복구 설계

4.1 자가 고장 테스트

일반적으로 메모리나 논리회로를 테스트하기 위해서는 메모리 테스트 장비나 논리 테스트 장비를 사용한 후 메모리 테스트 장비를 사용하는 것은 현실적이지 못하다. 따라서, 내장형 메모리에 대한 자체 테스트의 필요가 절실하다. BIST는 외부로부터의 제어 신호나 테스트 데이터를 공급받지 않고 칩 자체에서 자발적으로 테스트 수행이 가능 하도록 해준다. 그림 4는 본 논문에서 제안하는 메모리 BIST의 구성을 보인다.

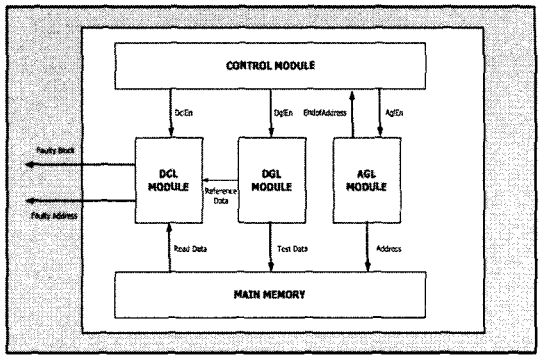

그림 4. 메모리 BIST 회로의 구조

4.1.1 제어 모듈(CONTROL Module)

제어 모듈은 테스트 진행 과정 중에 메모리 BIST의 각 모듈의 동작을 제어하는 회로로써 전체적인 테스트 시작과 종료 시점을 판단한다. 메모리 테스트하기 위해 필요한 알고리즘과 배경 데이터에 대한 상태머신이 존재하며, 상태머신은 알고리즘과 배경 데이터를 분석해 메모리 주소 적용된 알고리즘을 통해 적절한 배경 데이터를 인가한다. 각각의 모듈에 적절한 신호를 인가해 전체적인 테스트가 원활히 동작할 수 있도록 총체적인 역할을 담당한다.

4.1.2 주소 생성 모듈(AGL Module)

주소 생성모듈은 제어 모듈로부터 신호를 받아 테스트 데이터 값을 정확한 주소 위치에 읽고, 쓰기 할 수 있도록 메모리 주소를 순차적으로 증가, 감소시킨다. 그림 5는 주소 생성 모듈의 구조를 보여주고 있으며 EndOfAddress는 CONTROL 모듈의 입력으로 사용된다.

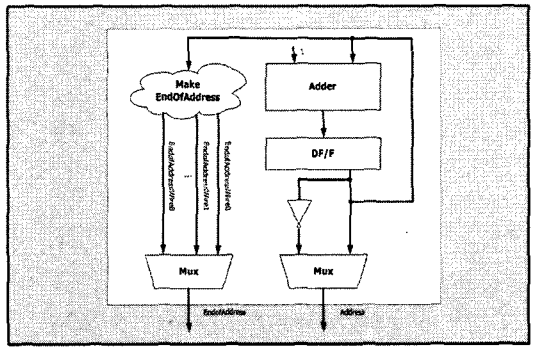

그림 5. 주소생성 모듈의 구조

이 신호는 주소의 마지막까지 진행되었음을 제어 모듈에 알려준다. EndOfAddressWire0, EndOf AddressWire1, EndOfAddressWire1, ... 신호는 현재 테스트가 진행되고 있는 메인 메모리의 주소를 마지막 주소를 생성할 때 발생되는 신호이다.

4.1.3 데이터 생성 모듈(DGL Module)

데이터 생성 모듈은 제어 모듈로부터 신호를 받아 테스트 패턴을 만들어 테스트가 진행 중인 메모리를 위한 데이터나 배경 데이터를 생성해 메모리에 전달한다. 테스트 모드시에 제어 모듈에서 생성되는 DglEn이라는 신호에 의해 테스트 패턴을 생성하는 모듈이다.

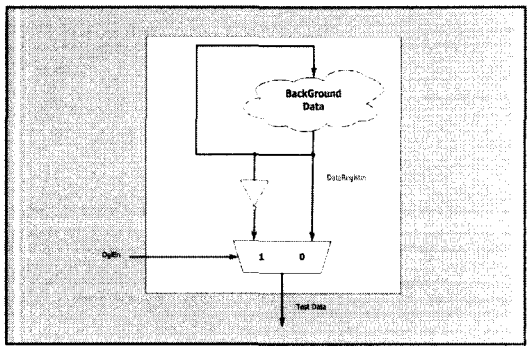

그림 6. 데이터 생성 모듈의 구조

그림 6의 Test Data는 실제 메모리에 들어가는 테스트 패턴으로 DglEn이 '0'인 경우는 Data Register 값 그대로 인가되며, DglEn이 '1'일 경우 DataResiger 값을 보수를 취해 Test Data로 배경 데이터 값을 출력한다.

4.1.4 데이터 비교 모듈(DCL Module)

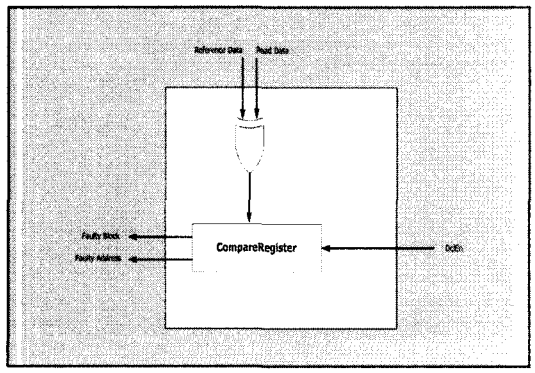

그림 7. 데이터 비교 모듈의 구조

데이터 비교 모듈은 메모리에서 읽어온 값(Read Data)과 데이터 생성 모듈에서 생성된 값(Reference Data)을 읽어 비교를 수행하는 모듈이다. DclEn 신호가 활성화 되었을 경우 Read Data와 Reference Data를 XOR후의 값이 같지 않을 경우 메인 메모리의 주소 값(Faulty Address)과 블록의 값(Faulty Block)을 BIRU 모듈에 보내준다.

4.2 여분의 메모리 재배치

특정 주소에 메인 메모리의 고장난 셀의 블록을 여분의 메모리의 블록에 재배치하여 메인 메모리의 데이터를 읽거나 쓰기를 할 때 여분의 메모리로 읽기, 쓰기를 자유롭게 할 수 있도록 BIRU에서는 메인 메모리의 고장난 블록의 개수와 여분의 메모리 블록의 개수를 판단해 적절히 재배치한다. 메인 메모리의 고장난 블록을 여분의 메모리로 재배치하기 전에 여분의 메모리 셀의 고장 유무를 파악해야 한다.

그림 8에서와 같이 BIRU는 Process 부분과 ARU (Address Remapping Unit) 두 부분으로 나눈다. BIST의 DCL 모듈에서 메인 메모리의 고장이 확인되면, 고장난 셀이 포함하고 있는 블록의 개수를 BIRU의 Process에 보낸다. Process에서는 여분의 메모리에 첫 번째 주소의 사용 가능한 블록의 개수를 ARU에서 받아 DCL 모듈에서의 고장난 블록의 개수와 여분의 메모리에서 사용할 수 있는 블록의 개수를 비교한다. Process에서 여분의 메모리 주소 값이 ARU에 인가되면 그 주소에 있는 블록이 이전에 사용했는지(Valid)와 여분의 메모리 블록 자체에 고장이 있는 셀이 존재 하는지(Fail)를 판단 후 최종적으로 사용 할 수 있는 여분의 메모리 주소의 개수를 Process에 보내진다.

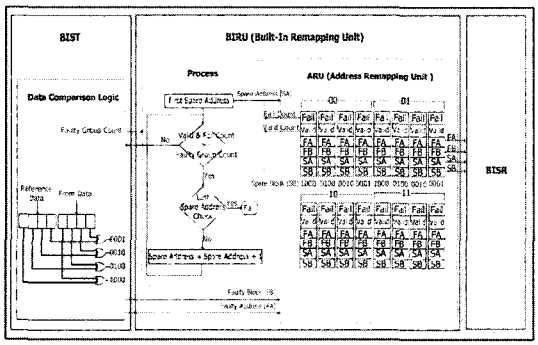

그림 8. BIRU 흐름도

Process에서는 고장난 블록의 개수(Faulty Group Count)와 사용 가능한 여분의 메모리(Valid & Fail Count)를 비교 한 후 고장난 블록 개수가 여분의

메모리 블록의 개수 보다 더 작다면 메인 메모리의 고장난 블록의 값(FB : Faulty Block),(블록의 순서에 따라, '1000', '0100', '0010', '0001'), 메인 메모리의 주소 값(FA : Faulty Address), 여분의 메모리 주소 값(SA : Spare Address)을 ARU 임시 레지스터에 저장을 하고, 그때 여분의 메모리 블록의 값(SB : Spare Block)도 같이 저장을 한다. 만약 고장난 블록의 개수가 더 크면 그 때의 여분의 메모리 주소가 마지막 여분의 메모리 주소인가를 확인하고 마지막 주소가 아니면, 여분의 메모리 주소를 다음 주소로 증가 시킨 후 다시 비교를 한다. 여분의 메모리 주소가 마지막이었다면, 여분의 메모리의 크기보다 메인 메모리에서 고장난 셀의 블록이 수가 더 많아 재배치 할 수 없는 상태를 의미하며 BISR를 실행할 필요가 없게된다.

4.3 자가 고장 복구

BIST와 BIRU의 동작이 모두 끝마치고 난 후 BIRU 안의 모듈 ARU 회로의 설정값을 가지고 고장난 메인 메모리의 데이터값을 여분의 메모리로 대신 사용하게 된다. 메인 메모리의 제어신호가 RAMEN (RamEnable) = '0', WE(WriteEnable='0', OE(Output Enable) = '1'을 가지면 메인 메모리에 데이터를 쓰고, RAMEN(RamEnable) = '0', WE(WriteEnable) = '1', OE(Output Enable) = '0'의 값들을 가지면 데이터를 메인 메모리로부터 읽는다.

메인 메모리의 셀 고장에 의해 여분의 메모리를 사용하려면 여분의 메모리도 메인 메모리가 읽기나 쓰기 동작시에 같은 읽기, 쓰기 동작을 해야 한다. 본 논문에서는 /WE의 신호를 가지고 메인 메모리와 여분의 메모리가 동시에 읽기와 쓰기를 할 수 있도록 구분하였다(/OE를 가지고 구분해도 상관없음).

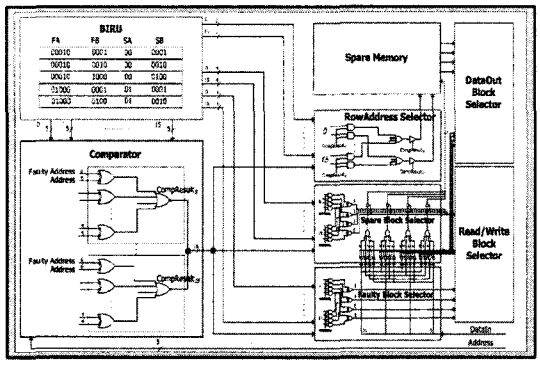

그림 9. Comparator와 Faulty Block Selector, Spare Block Selector의 구조

4.3.1 비교기와 여분의 메모리 행 주소 선택기

메인 메모리에 데이터를 쓸 때, 주소를 선택하고 그 선택한 주소에 데이터를 쓴다. 이 때 선택한 주소에 고장 유무가 있었는지를 그림 9에 나타난 회로를 통해 알 수 있다. Comparator 회로는 BIRU로부터 각각의 블록의 FA값들과 메인 메모리에 접근한 주소를 병렬로 비교한다. FA값과 주소의 값을 각각 XOR 한 후 OR 게이트로 묶는다. 만약 주소가 일치하면 '0'값을 출력하고, 같지 않으면 '1'을 출력한다. 메인 메모리에 '00010' 주소를 가지고 데이터를 읽거나 쓰기를 하면 Comparator 회로에서 '1111111111111000' 값을 출력한다.

이 출력값은 RowAddress Selector 회로와 Spare Block Selector 회로, Faulty Block Selector 회로에 입력값을 가지며, 만약 고장이 없는 메인 메모리의 워드에 접근 할 때에는 BISR의 회로를 동작하지 않고 출력할 수 있도록 MUX의 선택선의 입력 값으로 인가 된다. RowAddress Selector회로는 BIRU에서의 Row Address 값을 각각의 블록 단위로 입력 받아 Comparator의 출력 값을 RowAddress Selector 회로의 삼상 버퍼의 선택 선으로 인가되며 OR 게이트를 거쳐 '00' 값을 출력하게 된다. 이 값은 여분의 메모리에 주소값으로 출력된다.

4.3.2 고장난 데이터 블록 선택기와 여분의 블록 선택기

그림 9의 Faulty Block Selector 회로에서는 선택선에 의해 삼상 버퍼의 출력으로 나올 수 있는 값은 '1000', '0010', '0001'이다. 이 값을 OR 게이트로 통과 시킨 후 그 값 '1011'를 Read/Write Block Selector와 DataOut Block Selector의 입력으로 인가시킨다. 메인 메모리에 입력되는 데이터들을 블록의 개수로 분할해 메인 메모리로 입력되는 데이터 블록 하나가 여분의 메모리로 입력되는 경우의 수 만큼 입력하며 Read/Write Block Selector의 출력값에 의해 여분의 메모리 입력 블록을 결정한다.

Spare Block Selector 회로는 선택선에 의해 삼상 버퍼의 출력으로 나올 수 있는 값은 '0100', '0010', '0001'이다. 이 값들을 OR 한 후의 값 '0111'을 Read/Write Block Selector 회로의 입력 값으로 인가한다.

4.3.3 읽기/쓰기 블록 선택기

그림 10의 Read/Write Block Selector회로는 Faulty Block Selector회로와 Spare Block Selector

회로에서의 OR한 값을 MUX 회로에 입력으로 받는다. 이 MUX의 선택선은 메인 메모리의 제어 신호인 /WE 값으로 메인 메모리의 읽기 동작을 할 때는 '1' 값을 가지며, 쓰기 동작 때는 반대의 값인 '0'을 갖는다.

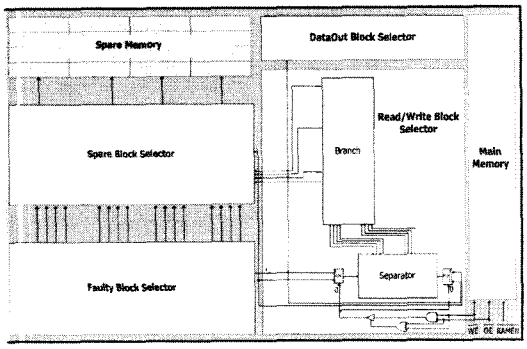

그림 10. Read/Write Block Selector의 구조

MUX의 선택선이 /WE값에 의해 결정되면 각각의 MUX 출력 값이 Separator 회로의 입력으로 들어간다. Separator 회로를 통해 출력된 값들은 Branch 회로를 통과해 Spare Block Selector 회로로 입력된다. 입력된 값들은 삼상 버퍼의 선택선으로 인가되며 메인 메모리에서 고장난 블록을 여분의 메모리에 쓰기 동작을 하게된다.

4.3.4 데이터 출력 선택기

메인 메모리의 고장난 위치를 읽을 경우에는 여분의 메모리에 쓰기를 했던 블록의 값과 메인 메모리의 값을 치환한다. 여분의 메모리의 워드 값을 블록 단위로 그림 11의 회로에서 입력으로 출력한다. DataOut Block Selector 회로에 입력된 여분의 메모리 워드 값들이 메인 메모리에서 출력된 워드 값과 치환될 위치 블록을 선택하기 위해 Read/Write Block Selector의 출력 값과 Faulty Block Selector 회로 출력 값을 가지고 DataOut Block Selector 회로의 삼상 버퍼와 MUX의 선택선으로 인가된다.

Read/Write Block Selector에서 나온 값('0000', '0100', '0100', '0010', '0001')이 각각의 삼상 버퍼의 선택선으로 입력되면 여분의 메모리로부터 출력되는 데이터 값들을 선택선에의해 위치가 구분된다. 고장난 블록을 가지고 있는 메인 메모리로부터 출력되는 데이터 값들은 여분의 메모리로부터 나와 삼상 버퍼와 OR게이트를 지나온 데이터 값들로 치환되기 위해서는 Faulty Block Selector에서 나온 '1011'의 값을 선택선으로 가져 MUX를 통과해 치환된 데이터 값은 아무런 고장 없이 메인 메모리의 출력 값으로 나가게 된다. 만약 고장이 없는 주소를 읽었을 경우 Comparator에서 '1111111111111111' 값이 나오고 그 값은 DataOut에 걸려있는 MUX의 선택선을 가져 BISR의 회로를 거치지 않고 Bypass 된다.

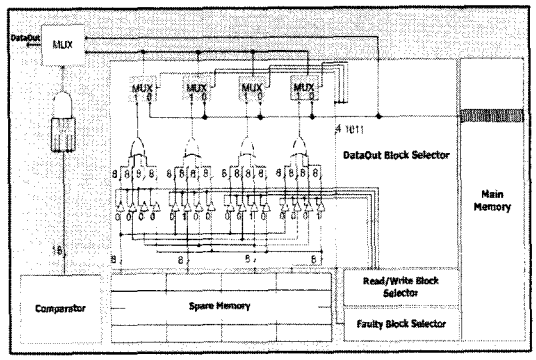

그림 11. DataOut Block Selector의구조

V. 실험결과

본 논문에서 제안한 BISR 설계에 대한 검증은 VerilogHDL로 기술하여 구현하였다. 구현에 대한 검증은 Xilinx사의 Xilinx Foundation에서 제공하는 simulator를 사용하여 RTL 검증을 하였다. 아래의 그림 12는 메모리 BIST가 동작하면서 고장난 셀을 BIRU를 통해 여분의 메모리로 재배치 되기 위한 정보를 출력해 주는 파형의 일부이다.

그림 12의 (1), (2), (3)은 각각 Clock, Reset, BistEnable 신호이다. BistEnable 신호가 '1'이 되면 BIST 동작이 시작된다. (4), (5), (6) 신호들은 RamEnable, OutputEnable, WriteEnable이며, (7)번 신호는 메모리의 주소를 나타낸다.

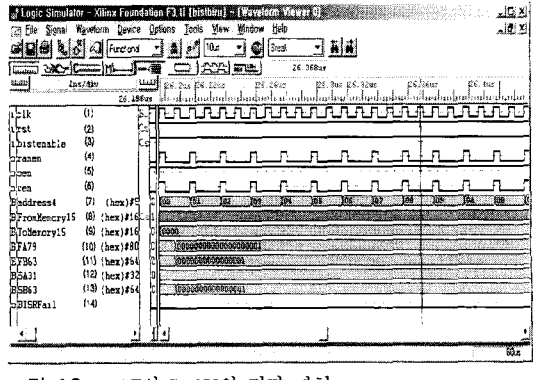

그림 12. BIST와 BIRU의 결과 파형

(8), (9) 신호들은 메모리에 테스트 데이터를 입력하는 신호와 메모리에서 테스트 데이터를 읽어오는 신호들이다. 만약 메모리에서 읽어온 데이터가 메모리에 쓰기를 한 데이터와 일치 하지 않는다면 그 때의 여분의 메모리의 공간 상태를 파악해 여분의 메모리로 데이터를 재배치할 수 있도록 여분의 메모리의 블록정보를 담고 있는 신호들이 (10), (11), (12), (13)번 신호들이다. 여분의 메모리 블록에 더 이상 데이터를 입력시키지 못할 때에는 (14)번 BISRFail 신호가 '1'이 된다. 그림 12의 결과 값 FA, FB, SA, SB을 가지고 BISR 회로를 실행시켰을 때 여분의 메모리에 읽고, 쓰기가 가능함을 그림 13에서 확인 할 수 있다.

그림 13의 (1), (2) 신호는 OutputEnable, Write Enable 신호이며, (3)은 Address 값이다. Address 신호가 16진수(h) '01h'의 위치에 메모리를 쓰려고 할때 그림 13에서 메인 메모리의 고장난 블록의 위치가 파악되어 메인 메모리에 쓰려고 하는 데이터 값(4) '01234567h'이 여분의 메모리에 첫 번째(5), 두 번째(6), 세 번째(7) 블록에 '67h', '45h', '01h' 값이 저장되며 네 번째 블록(8)에는 값이 존재하지 않게된다. 메인 메모리가 '01h'주소에 값을 읽을때는 메인 메모리에서 출력된 값 '11230121h'값이 여분의 메모리에 값과 치환이 되어 최종 출력 값(9)은 '01234567h'를 출력하게 된다.

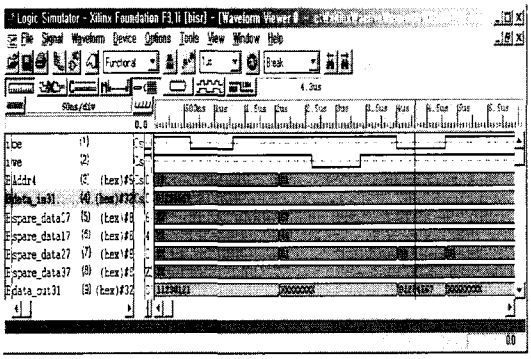

그림 13. BISR의 결과 파형

그림 14는 메인 메모리의 크기와 여분의 메모리, 블록의 개수를 다양하게 적용시켜 BISR의 전체 회로를 NAND 게이트로 환산해 나타낸 것이다. 메인 메모리와 여분의 메모리, 블록의 개수가 증가 할 수록 게이트 수는 증가된다. 하지만, 많은 수의 고장난 블록을 재배치 할 수 있어 메모리 수율을 향상을 볼 수 있다.

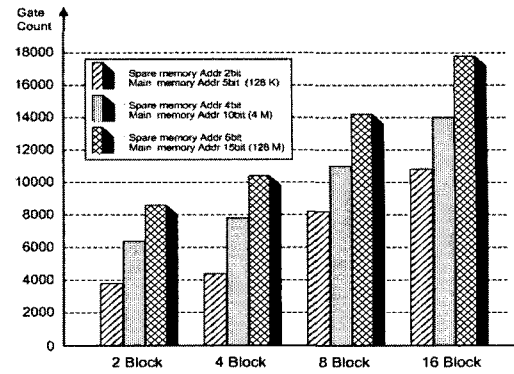

그림 14. BISR의 오버헤드

Ⅵ. 결 론

본 논문에서는 내장된 SRAM의 특정 주소를 블록으로 나눈 뒤 BIST를 이용해 고장의 유무를 판단한다. 고장이 발생 한 셀이 포함한 블록을 여분의 메모리로 대치하여 사용자에게 투명성을 제공하는 BISR 구조를 개발 하였다. BISR에 필요한 메인 메모리의 자가 테스트, 여분의 메모리에 메인 메모리의 블록을 재배치하기 위한 BIRU 회로 설계를 하였다.

BISR은 특정 주소가 고장난 블록을 가지고 있는지를 판단하는 Comparator 회로, 고장난 데이터를 여분의 메모리로 재배치하기 위한 Faulty Block Selector 회로, 여분의 메모리에 메인 메모리의 고장난 블록을 적절한 위치에 재배치하기 위한 Spare Block Selector 회로, 여분의 메모리의 주소를 선택하기 위한 Row Address Selector 회로, 메인 메모리의 읽기동작과 쓰기동작을 판단해서 데이터 값들의 위치를 결정해 주는 Read/Write Block Selector 회로이며, 이들 회로는 쓰기동작시에 필요하다. 읽기 동작시에는 쓰기 동작때 필요한 회로들과 메인 메모리와 여분의 메모리의 치환을 담당하는 Data-Out Block Selector 회로가 필요하다. 본 논문에서 제안한 BISR 회로는 메모리 고장 시에 SOC를 정상 동작시키며, 메모리의 물리적 구조에 독립적인 회로로 존재한다. 고가의 메모리를 여분의 메모리로 재사용 함으로서 메모리의 수율을 증가 시킬수 있을 것이다.

참 고 문 헌

[1] R. Rajsuman, "Design and Test of Large Embedded Memories: An Overview", *IEEE*

Design & Test of Computers, 2001.

[2] S. E. Schuster, "Multiple word/bit line redundancy for semiconductor memories", *IEEE Journal of Solid-State Circuits*, 1978.

[3] M. Horguchi, J. Etoh, M. Masakazu, K. Itoh, and T. Matumoto, "A flexible redundancy technique for high-density DRAM's", *IEEE Journal of Solid-State Circuits*, 1991.

[4] T. Yamagata, H. Sato, K. Fujita, Y. Nishimura, and K. Anami, "A distributed globally replaceable redundancy scheme for sub-half-micro ULSI memories and beyond", *IEEE Journal of Solid-State Circuits*, 1996.

[5] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins, and J. L. Lweandowski, "Built in self repair for embedded hish density SRAM", *in Proc. Int. Test Conf.(ITC)*, 1998.

[6] Y. Zorian, "Embedded memory test & repair: infrastructure IP for SOC yield", *in Proc. Int. Test Conf. (ITC)*, 2002.

[7] Y.Zorian, "Embedded infrastructure IP for SOC yield improvement", in Proc. *IEEE/ ACM Design Automation Conf. (DAC)*, 2002.

[8] V. Schober, S. Paul, and O. Picot, "Memory Built-In Self-Repair using redundant words", *in Proc. Int. Test Conf. (ITC)*, 2001.

[9] R. Dean Adams, "High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test", *Kluwer Academic Publishers*, 2003.

10] Benso, A., Di Carlo, S., Di Natale, G., Prinetto, P. and Lobetti Bodoni, M. "A programmable BIST architecture for clusters of multiple-port SRAMs," *in Proc. Int. Test Conf. (ITC)*, 2000.

[11] A. Bommireddy, J. Khare, S. Shaikh and S. Su. "*Test and debug of networking SoCs a case study*", *Montreal*, 2000.

[12] S. K. Lu, and C. H. Hsu, "Built-In Self-Repair for Divided Word Line Memory", *IEEE Int. Symposium, Circuits and Systems*, 2001.

[13] S. K. Lu, and S. C. Huang, "Built-in Self-Test and Repair(BISTR) Techniques for Embedded RAMs", *IEEE Design and Testing, Workshop on Memory Technology*, 2004.

심 은 성(Eun sung Shim) 준회원

2003년 2월 한서대학교 컴퓨터 정보학과 학사졸업

2003년 3월~현재 숭실대학교 대학원 컴퓨터학과 석사과정

<관심분야> VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD

장 훈(Hoon Chang) 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.

<관심분야> VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD