논문 2008-45SD-2-11

내장 메모리를 위한 프로그램 가능한 자체 테스트와 플래시 메모리를 이용한 자가 복구 기술

(Programmable Memory BIST and BISR Using Flash Memory for Embedded Memory)

홍 원 기*, 최 정 대*, 심 은 성*, 장 훈**

(Won-Gi Hong, Jung-Dai Choi, Eun-Sung Shim, and Hoon Chang)

요 약

메모리 기술이 발달함에 따라 메모리의 집적도가 증가하게 되었고, 이러한 변화는 구성요소들의 크기를 작아지게 만들고, 고장의 감응성이 증가하게 하였다. 그리고 고장은 더욱 복잡하게 되었다. 또한, 칩 하나에 포함되어있는 저장 요소가 늘어남에 따라 테스트 시간도 증가하게 되었다. 본 논문에서 제안하는 테스트 구조는 내장 테스트를 사용하여 외부 테스트 환경 없이 테스트가 가능하다. 제안하는 내장 테스트 구조는 여러 알고리즘을 적용 가능하므로 높은 효율성을 가진다. 또한 고장 난 메모리를 여분의 메모리로 재배치함으로써 메모리 수율 향상과 사용자에게 메모리를 투명하게 사용할 수 있도록 제공할 수 있다. 본 논문에서는 고장 난 메모리 부분을 여분의 행과 열 메모리로 효율적인 재배치가 가능한 복구 기술을 포함한다. 재배치 정보는 고장 난 메모리를 매번 테스트 해야만 얻을 수 있다. 매번 테스트를 통해 재배치 정보를 얻는 것은 시간적 문제가 발생한다. 이것을 막기 위해 한번 테스트해서 얻은 재배치 정보를 플래시 메모리에 저장해 해결할 수 있다. 본 논문에서는 플래시 메모리를 이용해 재배치 정보의 활용도를 높인다.

Abstract

The density of Memory has been increased by great challenge for memory technology, so elements of memory become smaller than before and the sensitivity to faults increases. As a result of these changes, memory testing becomes more complex. The number of storage elements is increased per chip, and the cost of test becomes more remarkable as the cost per transistor drops. Proposed design doesn't need to control from outside environment, because it integrates into memory. The proposed scheme supports the various memory testing algorithms. Consequently, the proposed one is more efficient in terms of test cost and test data to be applied. Moreover, we proposed a reallocation algorithm for faulty memory parts. It has an efficient reallocation scheme with row and column redundant memory. Previous reallocation information is obtained from faulty memory every each tests. However proposed scheme avoids to this problem. because onetime test result from reallocation information can save to flash memory. In this paper, a reallocation scheme has been increased efficiency because of using flash memory.

Keywords : Embedded Memory, BIST, Programmable BIST, BISR

I. 서 론

최근 급속한 시스템 온 칩(SOC) 기술의 발달로 대용량의 내장 메모리를 통합할 수 있게 되었다[1~2]. 그림 1에서 ITRS 2000년도 로드맵을 보면 SOC를 구성하는 많은 요소 중에서 내장 메모리가 차지하는 비중은 점차 증가하여 2002년에는 50%를 넘어선 것을 알 수 있다. 이는 SOC 칩을 테스트하는 과정에서의 시간과 비용을 줄이기 위해서는 내장 메모리를 테스트하는 과정이 가장 큰 영향을 미친다는 것을 의미한다.

내장 자체 테스트 (BIST : Built-In Self Test)는 기

* 학생회원, ** 정회원, 숭실대학교 컴퓨터학과
(Department of Computer, Soongsil University)
※ 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (No. R01-2006-000-11038-0)
접수일자: 2007년10월4일, 수정완료일: 2008년1월7일

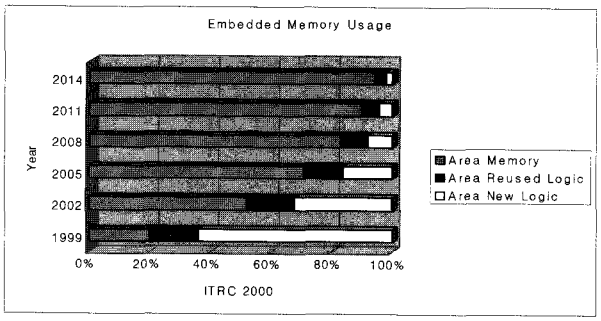


그림 1. 내장 메모리 사용 비율
Fig. 1. Embedded memory usage.

존의 테스트 문제점들을 해결하는 방법을 제시한다[3~6]. 그러나 이러한 테스트 구조는 이미 정해진 하나의 테스트 알고리즘만을 적용 가능하다. 반면에, 다양한 테스트 알고리즘을 지원하는 프로그램 가능한 메모리 내장 자체 테스트(PMBIST : Programmable Memory BIST)는 테스트가 필요한 시점의 환경에 적합한 테스트 알고리즘을 선택 가능하므로, 높은 효율성을 가지고 있다. 많은 프로그램 가능한 메모리 내장 자체 테스트가 개발되었다[7].

본 논문에서는 매크로 코드에 의해 동작하는 FSM 방식의 내장 자체 테스트 구조를 제안한다. 이 구조에서는 알고리즘을 선택하기 위한 간단한 코드만을 ATE가 내장 자체 테스트 실행 명령과 함께 입력하여 주면 제안하는 내장 자체 테스트가 자동으로 선택된 알고리즘에 맞는 테스트 명령을 만들어 테스트를 수행하게 된다.

내장 메모리 및 SoC의 수율을 개선시키기 위하여 사용되는 고장복구(BISR : Built-In Self-Repair)는 BIST와 BISD(Built-In Self-Diagnostics), BIRA(Built-In Redundancy Analysis) 등과 같은 여러 가지 메커니즘을 수반하고 있다. 일반적인 BIST의 구조는 테스트 회로를 테스트하기 위해 패턴을 자동적으로 생성해 테스트 회로에 입력시킨 후, 테스트 패턴과 테스트 후 결과값이 사용자가 원하는 결과 값과의 동일 여부로 회로의 고장 유무를 판단하는 구조를 가지고 있다.

본 논문에서는 BIST를 통해 고장 난 부분을 찾아낸 후, 내장 메모리의 고장 난 부분을 여분의 행과 열 메모리로 재배치 할 수 있도록 알고리즘을 제안하였다. 내장 메모리에 고장이 났을 경우 여분의 메모리 행이나 열만을 이용해 재배치하는 경우 보다 여분의 메모리에 행과 열 모두를 이용해 고장 난 부분을 행과 열로 재배치하는 것이 보다 효율적이다[8~10]. 하지만 행과 열을 이용해 여분의 메모리를 재배치하는 알고리즘은 NP-hard 문제이기 때문에 매우 어렵다[11].

이제까지 여분의 메모리 재배치를 위해 테스트 장비를 이용한 여분의 메모리 분석 알고리즘들이 제안되어져 왔다[12~13]. 하지만 BISR의 구조는 회로에 내장되어야 하는데 제안되어졌던 알고리즘들은 많은 비용을 소비하여 내장 회로 내에 적용할 수 없다.

여분의 메모리를 구현하는 방법은 Hard Redundancy (Spare)[14]와 Soft Redundancy(Spare)[15]로 나누어지는데 본 논문에서는 두 가지 방법을 혼합한 방식을 취하여 재배치 알고리즘을 통해 생성된 정보를 플래시 메모리에 저장해 반영구적으로 고장 난 위치의 정보를 치환하는 방법을 사용하였다. 이렇게 재배치 정보를 플래시 메모리에 저장해 사용하면 매번 BIST를 수행하지 않아도 되는 시간적 절약에 대한 장점이 있는 반면, 재배치 정보를 플래시에 담고 난 후 메모리 셀이 고장이 났을 경우 고장을 검출할 수 없는 문제점이 발생하게 된다. 이것은 사용자의 경험에 비추어 BIST 수행 여부를 판단하면 될 것이다.

본 논문의 구성은 서론에 이어 Ⅱ장 제안하는 프로그램 가능한 메모리 내장 자체 테스트의 구조를 언급하고, Ⅲ장 플래시 메모리를 이용한 고장 메모리 재배치 기술을 소개한다. 마지막으로, Ⅳ장에서는 제안하는 내장 메모리 자체 테스트 및 복구 구조를 검증하고, Ⅴ장 결론으로 본 논문을 마친다.

Ⅱ. 제안하는 프로그램 가능한 메모리 내장 테스트 구조

1. 선택 가능한 테스트 알고리즘

제안하는 프로그램 가능한 메모리 내장 자체 테스트에서는 표 1에서와 같이 총 7개의 알고리즘을 지원한다. 메모리 생산 공정 초기에는 많은 고장이 발생하게 되므로, 현재까지 모델링 된 모든 고장을 검출해 낼 수 있는 March SS 알고리즘[16~17]이 포함되어있고, 생산이 거듭될수록 공정은 안정화 되므로, 단순하고 빠른 MATS+ 와 같은 알고리즘을 지원한다.

표 1에서 번호는 외부 ATE 장비에서 내장 자체 테스트에서 사용할 알고리즘의 코드이다. Selection은 3비트로 되어있으며 7개의 알고리즘을 지원한다. 코드는 March elements 마다 5비트로 되어 있으며, 메모리에 적용할 March Elements를 가리킨다.

코드의 최상위 비트는 적용할 March Element의 주소 증감을 나타낸다. 주소를 증가시키면서 테스트를 하

표 1. 알고리즘 선택표
Table 1. Algorithm selection table.

번호	알고리즘	코드
001	MATS+	10001 10010 00011
010	March X	10001 10010 00011 10100
011	March C-	10001 10010 10011 00010 00011 10100
100	March B	10001 10101 10110 00111 01000
101	March U	10001 11001 10010 01010 00011
110	March LR	10001 00010 11010 10011 11001 10100
111	March SS	10001 11011 11100 01101 01110 10100

표 2. March Elements 선택표
Table 2. March Elements selection table.

March Element	코드 [3:0]	동작
M1	0001	w0
M2	0010	r0,w1
M3	0011	r1,w0
M4	0100	r0
M5	0101	r0,w1,r1,w0,r0,w1
M6	0110	r1,w0,w1
M7	0111	r1,w0,w1,w0
M8	1000	r0,w1,w0
M9	1001	r0,w1,r1,w0
M10	1010	r1,w0,r0,w1
M11	1011	r0,r0,w0,r0,w1
M12	1100	r1,r1,w1,r1,w0
M13	1101	r0,r0,w0,r0,w1
M14	1110	r1,r1,w1,r1,w0

여야 하거나, 주소의 증감이 자유로운 March Element의 경우는 '1'을 부여하고, 반대로 주소를 감소시키면서 적용해야 하는 경우에는 '0'을 할당하였다. 코드의 나머지 4비트는 전체 알고리즘의 March Element들을 정리하여 주소의 증감 여부를 제외한 읽기/쓰기 동작이 동일한 것끼리 묶어 같은 번호를 부여하였다.

즉, 모든 알고리즘의 총 March Element수는 35개이지만, 동일한 March Element에는 같은 번호를 부여하였기에, 1에서 14까지의 번호가 부여된 March elements로 모든 알고리즘을 표현할 수 있게 되었다. 이러한 취합 과정을 거쳐 정리된 March Elements는 표 2와 같다.

2. 제안하는 프로그램 가능한 내장 자체 테스트의 구조

제안하는 프로그램 가능한 내장 자체 테스트는 그림 2와 같이 외부로부터 알고리즘을 선택 받고, 알고리즘의 March elements 순서에 맞게 March Elements를 전달하는 알고리즘 생성 모듈(Algorithm Generator), March Element를 메모리에 적용시키는 테스트 제어 모듈(Test Controller)로 구성되어 있다.

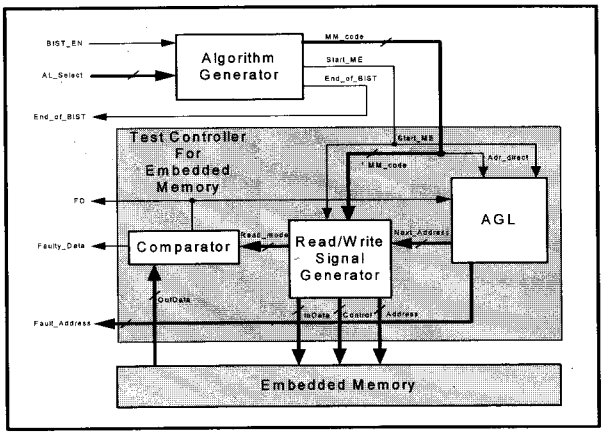


그림 2. 제안하는 프로그램 가능한 내장 자체 테스트의 구조
Fig. 2. Proposed programmable memory BIST.

테스트 제어 모듈은 주소를 발생 시키는 주소 생성 회로(Address Generator Logic, AGL), 내장 메모리에 읽기/쓰기 동작에 적합한 신호를 발생시키는 읽기/쓰기 신호 생성 모듈(Read/Write Signal Generator), 마지막으로 메모리에서 읽어 온 값을 정상 동작일 경우의 값과 비교하는 비교 모듈(Comparator)로 구성되어 진다.

제안하는 프로그램 가능한 메모리 내장 자체 테스트의 전체 흐름은 다음과 같다. 가장 먼저 ATE에서 테스트 시작과 함께 알고리즘을 선택하여 주면, 알고리즘 생성 모듈에서는 주어진 알고리즘의 March Elements를 준비한다. 준비된 March Element는 테스트 제어 모듈의 상태에 따라 전달되게 된다. 테스트 제어 모듈에서는 전달 받은 March Element를 메모리의 읽기/쓰기 동작으로 바꾸고, 테스트할 주소를 발생시킨다.

가. 알고리즘 생성 모듈

알고리즘 생성 모듈은 외부 ATE 장치로부터 신호를 받아 내장 자체 테스트를 실행시키는 모듈이다. 이 모듈은 외부 ATE 장치로부터 내장 자체 테스트를 실행하라는 BIST_EN 신호와 알고리즘을 선택하는 3비트 AL_Select 신호, 클럭 Clk를 입력 받고, 테스트 제어 모듈이 다음 March Element를 요청하는 신호인

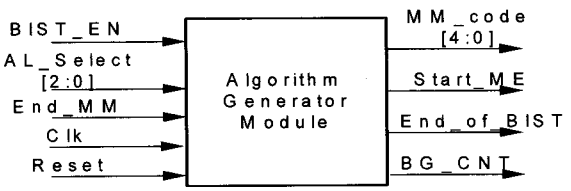


그림 3. 알고리즘 생성 모듈의 블록 다이어그램
Fig. 3. Block diagram of algorithm generation module.

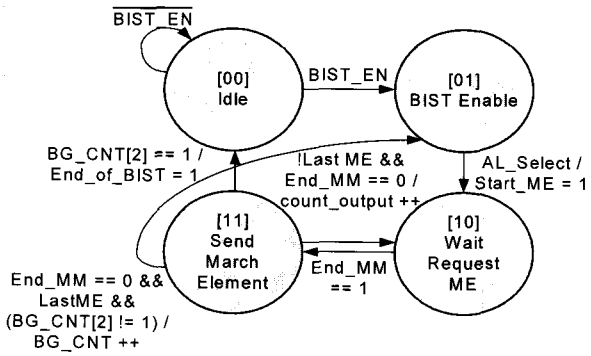


그림 4. 알고리즘 생성 모듈의 FSM
Fig. 4. FSM of algorithm generation module.

End_MM 신호를 입력받는다. 그림 3에서와 같이 출력으로는 March Element와 주소 증감 여부가 정해지는 5비트 MM_code와 알고리즘의 시작을 알리는 Start_AL, 모든 테스트가 종료 된 후 발생하는 End_of_BIST, 백그라운드 데이터를 바꿔주는 BG_CNT 신호가 있다.

알고리즘 생성 모듈은 FSM을 기반으로 동작한다. 해당 FSM은 그림 4와 같이 외부로부터 아무런 입력이 존재하지 않을 시에는 Idle[00] 상태에서 외부 입력을 기다리게 된다. BIST_EN 신호가 인가되면 BIST_Enable 상태로 전환되고, 외부로부터 AL_Select 신호를 입력 받으면 해당 알고리즘을 준비하고 알고리즘의 March element가 발생되었음을 알리는 Start_ME 신호를 발생하게 된다. 그 후 테스트 제어 모듈로부터 March Element 요구 신호인 End_MM을 반복적으로 받을 때 마다 March Element를 MM_code에 인가한다. 마지막 March Element를 보내고 나서 백그라운드 데이터를 바꾸어 주는 BG_CNT를 하나 증가시켜 주고 바뀐 백그라운드 데이터를 적용시키기 위하여 알고리즘을 다시 실행한다. 이러한 과정을 반복하여 백그라운드 데이터를 모두 적용하게 되면 End_of_BIST 신호를 발생시키고 내장 자체 테스트가 끝났음을 알린다.

나. 테스트 제어 모듈

테스트 제어 모듈은 3가지 모듈로 구성되어 진다. 3가지 모듈은 주소를 발생 시키는 주소 생성 회로(Address Generator Logic, AGL), 내장 메모리에 읽기/쓰기 동작에 적합한 신호를 발생시키는 읽기/쓰기 신호 생성 모듈(Read/Write Signal Generator), 마지막으로 메모리에서 읽어 온 값을 정상 동작일 경우의 값과 비교하는 비교 모듈(Comparator)이다.

테스트 제어 모듈의 세부 모듈들은 그림 5와 같이 동작한다. 알고리즘 생성 모듈로부터 받은 MM_code를 최상위 비트는 주소 생성 모듈로 보내지고 나머지 비트는 읽기/쓰기 신호 생성 모듈로 보내어 진다.

주소 생성 모듈은 입력 받은 MM_code의 최상위 비트가 '1'일 경우에는 주소를 증가시키는 방향으로 발생시키고, 0일 때에는 감소하는 방향으로 주소를 만들어 메모리로 보내게 된다.

읽기/쓰기 신호 생성 모듈은 MM_code의 나머지를 가지고 해당 March Element의 읽기/쓰기 동작을 순서에 맞게 발생 시킨다. 예를 들어, M3 (r1,w0) March Element의 경우, 코드는 0011이다. 가장 먼저 상태[000]에서 코드 0011이 입력되면 상태[100]으로 이동하게 된다. 상태[100]은 읽어온 '1'값을 비교(r1)하는 과정을 수행하는 상태이다. 그 이후, 상태[001]로 이동하게 된다. 이 상태는 메모리에 '0'값을 쓰는(w0) 과정을 수행하는 상태이다. 다음으로 다시 상태[000]으로 돌아오게 되고 주소 생성 모듈에 End_OP 신호를 보내 주소를 증가 또는 감소시키게 된다.

각 r0/r1/w0/w1 상태에서는 내장 메모리에 접근하는 신호를 발생시킨다. 생성되는 신호는 다음과 같다. Row Address 접근을 알리는 _RAS 신호, Column Address 어드레스 접근을 알리는 _CAS 신호 , 메모리에 저장을 알리는 _WE 신호, 메모리에서 해당 주소 값을 읽어 오는 _OE 신호, Row와 Column 주소를 전달하는 Address, 그리고 마지막으로 메모리에 읽고, 쓰게 되는 값 OutData, InData이다.

비교기는 메모리에서 읽어오는 값 OutData를 알고리즘에서 예상하는 값과 비교하여 값이 상이할 경우 고장이 발생한 경우이기에 고장이 발생한 곳의 주소와 고장 데이터 값을 출력하고, Fault_detected 신호를 내보내게 된다.

Ⅲ. 제안하는 플래시 메모리를 이용한 고장 메모리 복구 기술

1. 재배치 알고리즘

본 논문에서 제안하는 재배치 알고리즘은 메인 메모리에 고장이 발생 하여 BIST의 테스트 제어 모듈이 FD 신호를 인가하면, 테스트 제어 모듈로부터 Fault_address 신호와 Faulty_data 신호를 받아 저장하게 된다.

본 알고리즘은 메인 메모리의 고장 난 위치와 고장 난 주소를 이용하여 행을 기준으로 가장 고장이 많은 하나의 행, 또는 열을 선택하여 여분의 행 메모리로 재

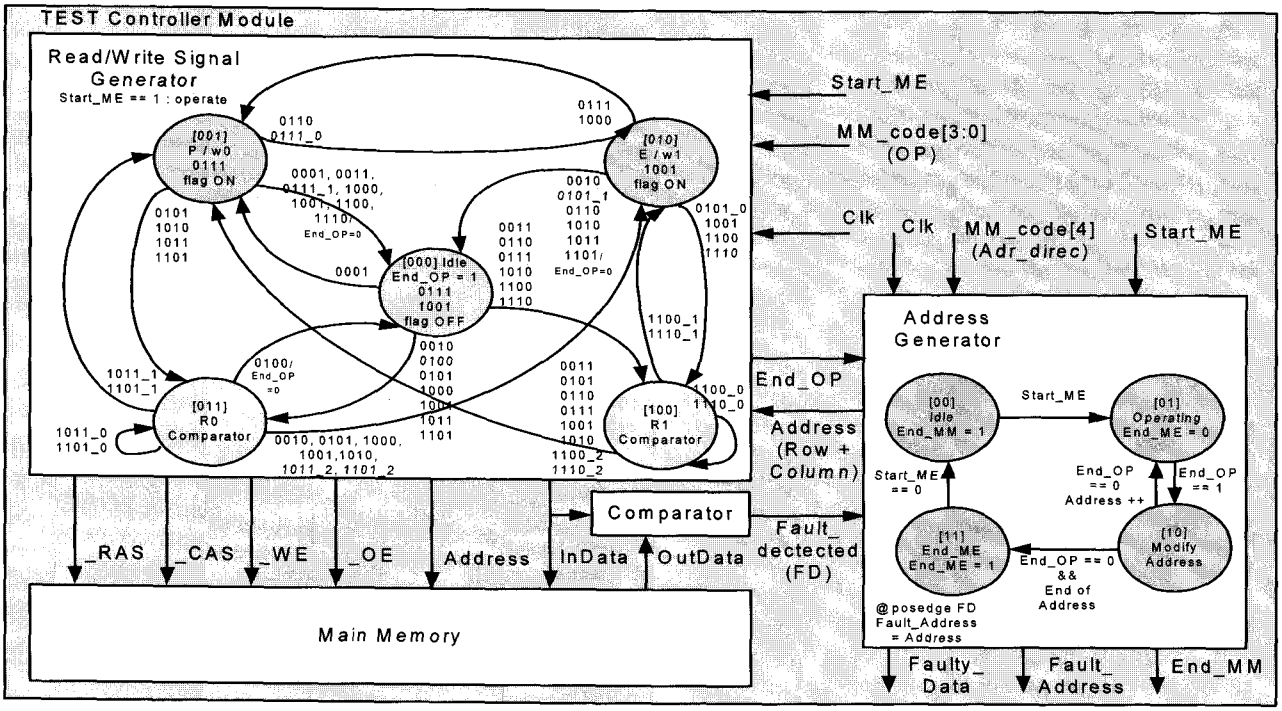


그림 5. 테스트 제어 모듈의 FSM
Fig. 5. FSM of test controller module.

배치한다. 가장 고장이 많은 행이 다수가 되면 처음 발생한 행을 선정한다. 다음 고장이 있는 주소의 첫 번째 고장 셀에서 그 주소의 행에 고장 난 셀 개수와 열에 고장난 셀 고장 개수를 비교한다. 고장의 개수가 행이 많다면 셀 위치를 여분의 행 메모리로 재배치하고 열의 개수가 많다면 여분의 열 메모리로 재배치한다.

행과 열에 고장이 발생한 셀의 개수를 비교하기 전에 두 가지를 먼저 계산한다. 첫 번째로 고장의 열 위치가 겹치게 되는지를 확인하여 우선적으로 여분의 행 메모리로 재배치한다. 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 재배치되어 이에 따른 문제를 발생하게 된다.

두 번째는 같은 주소에 고장 난 셀이 2개 이상일 때와 해당 열의 위치가 다른 주소에 의해 재배치되었다면 고장 난 셀의 공유를 막기 위해 같은 주소의 모든 셀들은 여분의 열 메모리로 재배치되어야 한다.

2. 재배치 알고리즘의 예

BIST는 고장을 확인하는 즉시 Reallocation Logic 안의 Save Logic에 고장 난 주소와 고장 난 위치 정보가

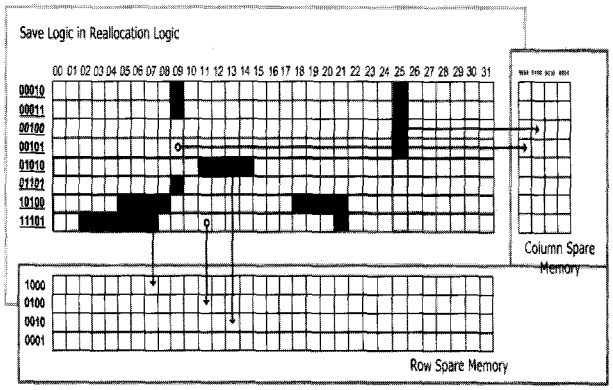


그림 6. 재배치 알고리즘 예
Fig. 6. Example of Reallocation Algorithm.

저장된다. 그림 6을 살펴보면 장난 정보 중에 가장 고장이 많은 셀을 포함하고 있는 행의 주소 10100을 행 여분의 메모리 1000에에 재배치한다. Save Logic 안에서 다음 고장 난 정보가 있는 주소(11101)에서 첫 번째 고장 셀의 열(02)의 고장 난 셀 개수와 행의 고장 개수를 비교한다. 하지만, 주소(1101) 열의 고장이 난 위치 05와 06, 07, 21번 위치는 가장 고장이 많은 주소(10100)의 열 위치와 겹치는 고장의 위치이다. 따라서 주소(11101)는 여분의 행 메모리(0100)에 재배치한다.

다음으로 주소(00010)의 열 위치(09)에서는 가장 고장이 많은 행의 열 고장 난 셀의 위치와 겹치는 부분이

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 7. 재배치 데이터
Fig. 7. Reallocation Datas.

없고, 같은 주소에서 여분의 열 메모리로 재배치 한 적이 없어 열 위치의 고장 난 셀 개수 3개와 행의 고장 난 셀 개수 2개를 비교한다. 열의 고장 난 셀 개수가 많음으로 열의 09번은 여분의 열 메모리(1000) 부분으로 재배치한다. 같은 주소(00010)의 다음 고장 난 셀의 열 위치 25번은 앞서와 같이 열 여분의 메모리(0100)로 재배치된다.

다음 주소 고장 위치 00011과 01101의 고장 난 셀 위치는 열의 09번이다. 09번은 이전 주소(00010)에서 여분의 열 메모리로 재배치되었기 때문에 주소 00011과 01101의 09번도 여분의 열 메모리(1000)로 재배치한다. 다음 주소 01010의 열 고장 난 셀 위치 11번은 행과 열의 고장 난 셀 개수가 각각 4개와 1개이므로 여분의 행 메모리(0010)로 재배치한다.

재배치된 데이터 정보들은 Reallocation Logic에서 그림 7에서와 같이 저장된다. 재배치된 최종 정보는 메인 메모리에 고장 난 위치(Faulty Data)와 메인 메모리에 고장 난 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 개 수 만큼 비트수를 갖는다. 여분의 행 메모리로 재배치되는 경우는 여분의 행 메모리 전체를 재배치하기 때문에 고장 난 위치를 확인할 필요가 없어 '-'로 표시하며, 여분의 열 메모리의 경우는 고장 난 위치의 셀을 재배치하기 때문에 고장 난 셀의 위치는 '1'로 고장이 없는 셀의 위치는 '0'로 나타낸다. 마지막으로 메인 메모리의 고장 난 주소의 위치는 Faulty Address에 표현한다.

여분의 메모리의 주소 표현은 여분의 메모리 개수를 주소로 나타낸다. 예를 들어 여분의 열 메모리 개수가 4개라면 일반적으로는 2비트로(00, 01, 10, 11) 나타낸다. 하지만 제안하는 구조에서는 접근 가능한 부분을 4비트로(1000, 0100, 0010, 0001) 나타낸다. 이렇게 표현한 이유는 동시에 고장 난 위치가 2(00, 01)곳일 경우 일반적인 방식으로는 2비트씩 2번 전송이 필요하지만 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.

기존의 재배치 정보는 휘발성 메모리에 저장되어 있어 전원이 끊어지게 되면 재배치 정보를 다시 얻어야만 한다. 따라서 매번 메모리를 수행할 때마다 메모리 자체 테스트를 통해 재배치 정보를 가지고 메모리 자체 복원을 수행해야만 한다. 본 논문에서는 재배치 정보를 비휘발성 메모리인 플래시 메모리에 저장한고 이 저장된 재배치 정보를 반영구적으로 사용할 수 있다.

3. 재배치 정보를 이용한 자가 복구 구조

가. 재배치 고장복구 구조

본 논문에서는 Reallocation Logic을 통해 고장 난 메모리의 셀 위치와 주소를 여분의 메모리로 재배치 할 수 있는 정보로 가공한다. 가공된 최종 정보는 그림 7에서 살펴보았던 4개의 정보가 재배치 정보로 Reallocation Logic으로부터 출력된다. 출력된 재배치 정보를 이용해 사용자가 고장 난 주소의 데이터를 사용시 고장이 없는 것 처럼 사용할 수 있게 데이터를 고장 위치로 입력, 출력 시키지 않고, 여분의 메모리를 이용해 고장이 없는 메인 메모리를 사용할 수 있게 한다.

그림 8은 BISR의 전체 구조도이다. 그림 8에서 나타내는 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 고장 난 위치와 고장 난 주소를 Fault Detection 신호와 함께 Reallocation Logic에 보내지게 된다.

나. 재배치 고장복구 회로 모듈의 특징

Reallocation Logic을 통해 재배치 정보를 받은 Data In-Out Replace Logic은 메인 메모리의 고장 난 위치에 접근하는 데이터를 정상적인 데이터로 교체 해주는 일을 수행한다. 그림 9는 Data In-Out Logic의 구조도를 나타낸다.

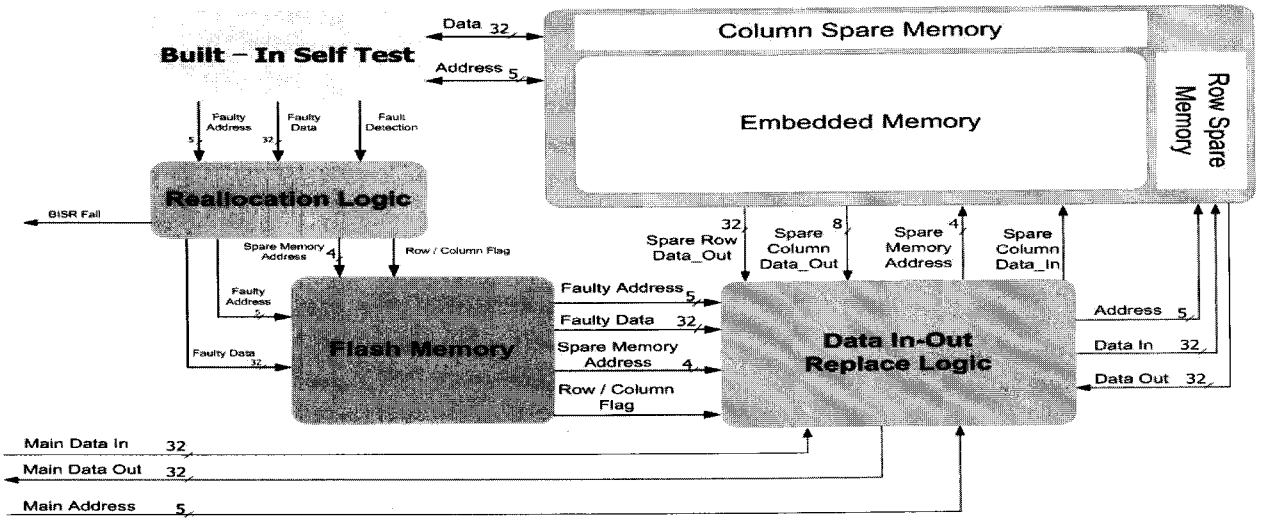


그림 8. BIST와 Reallocation Logic, Data In-Out Replace Logic의 전체구조도
Fig. 8. Block Diagram of BIST, Reallocation Logic, Data In-Out Replace Logic.

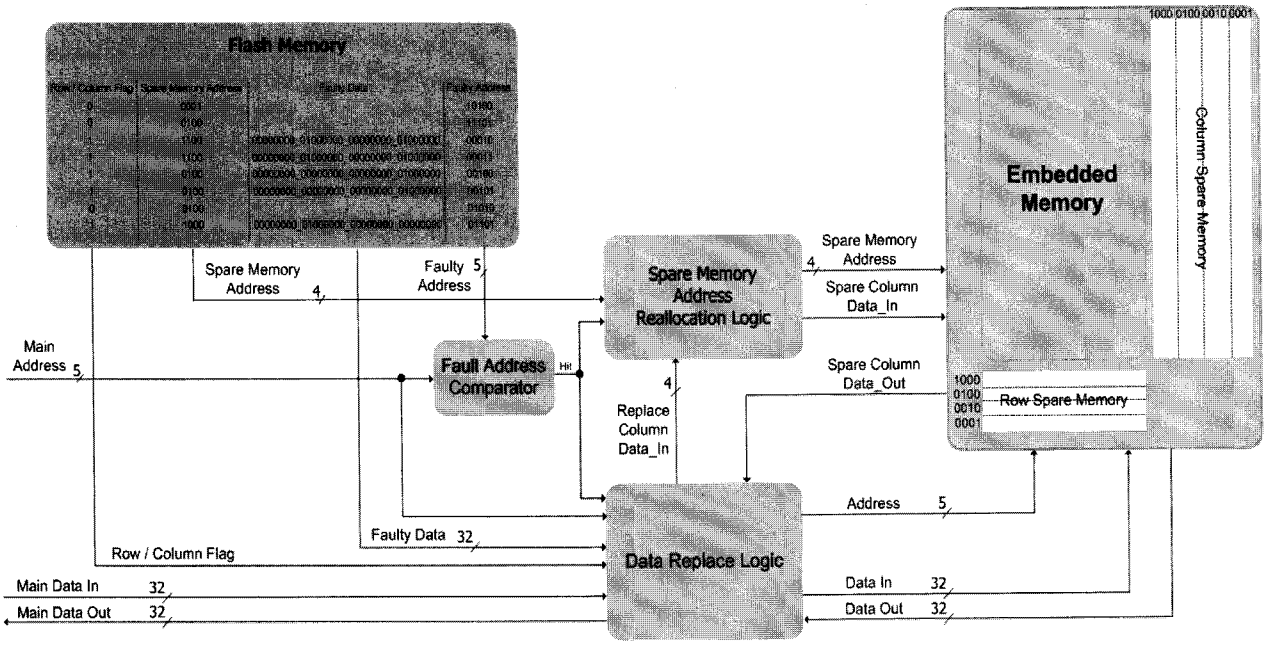


그림 9. Data In-Out Logic 구조도
Fig. 9. Block Diagram of Data In-Out Logic.

(1) Fault Address Comparator

Fault Address Comparator는 고장 난 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.

(2) Data Replace Logic

Data Replace Logic은 재배치 데이터가 여분의 행 메모리 또는 여분의 열 메모리로 재배치되는 지를 구분(Row/Column Flag)한다. 고장 난 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 고장 난 위치(Faulty Data)를 이용해 사용자가 입력한 데이터가 메인 메모리의 고장 난 위치에 접근하는 데이터 위치를 파악해 메인 메모리의 고장 난 위치에 접근하는 데이터를 추출하여 Spare Memory

Address Reallocation Logic으로 보내는 역할을 한다.

(3) Spare Memory Address Reallocation Logic

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

다. 여분의 메모리(Spare Memory) 읽기, 쓰기 동작

여기서는 각각 여분의 행 메모리의 쓰기 동작과 여분의 열 메모리의 읽기 동작을 예를 들어 살펴보도록 한다.

(1) 여분의 행 메모리 쓰기 동작

메인 메모리에 5비트 주소 '11101'에 32비트 데이터 'FF000000h'를 쓰려고 한다. 메인 메모리의 주소 '11101'은 여분의 메모리로 재배치 되었는지 Fault Address Comparator에서 비교 확인한다.

재배치된 주소임을 확인하게 되면 Fault Address Comparator로부터 hit 신호가 활성화 되어 Data Replace Logic으로 입력된다. 그 때, Reallocation Logic으로부터 Row/Column Flag 신호 '0'은 Data Replace Logic으로 입력되고, Spare Memory Address 값 '0100'이 Spare Memory Address Reallocation Logic로 입력된다. Row/Column Flag 신호 '0'과 Spare Memory Address Reallocation Logic에 입력된 값 '0100'에 따라서 여분의 행 메모리의 '0100' 위치에 입력된 데이터 값 'FF000000h'이 저장된다.

(2) 여분의 열 메모리 읽기 동작

메인 메모리에 5비트 주소 '00011'에 데이터를 읽으려 한다. 메인 메모리의 주소 '00011'은 여분의 메모리로 재배치되었는지 Fault Address Comparator에서 비교 확인한다.

재배치된 주소임을 확인하게 되면 Fault Address Comparator로부터 hit 신호가 활성화 되어 Data Replace Logic으로 입력된다. 그 때, Reallocation Logic으로부터 Row/Column Flag 신호 '1'과 Faulty Data가 Data Replace Logic으로 입력되고, Spare Memory Address 값 '1100'이 Spare Memory Address Reallocation Logic로 입력된다. Row/Column Flag 신호 '1'이면, 여분의 열 메모리로 재배치되어진 것이다.

Reallocation Logic에서 입력받은 여분의 열 메모리의 주소 '1100'을 '1000'과 '0100'으로 나눈다. 여분의 열 메모리 주소 '1000'과 '0100'의 값 '1'과 '0'을 Spare Memory Address Reallocation Logic에 입력 값으로 넣는다. 그 때에 메인 메모리의 주소 '00011'에 해당하는 데이터 값을 Data Replace Logic에 입력한다. Data Replace Logic에서는 입력된 고장 난 셀의 위치(Faulty Data:00000000010000000000000001000000)값을 토대로 고장난 셀 위치 7번과 23번에 데이터 값을 '0'과 '1'로 치환하여 최종 데이터 출력으로 내보낸다.

Ⅳ. 실험 결과

1. 프로그램 가능한 내장 자체 테스트 기술 검증

본 논문에서 제안한 내장 자체 테스트 설계에 대한 구현은 VerilogHDL로 기술하여 구현하였다. 구현에 대한 검증은 Xilinx사의 Xilinx Foundation에서 제공하는 시뮬레이터를 사용하여 RTL 검증을 하였다.

그림 10은 메모리 내장 자체 테스트가 정상적인 메모리에서 동작하면서 발생하는 파형의 일부이다. 그림 10의 (1), (2), (3)는 각각 Clock, BistEnable, Reset 신호이다. BistEnable 신호가 '1'이 되면 BIST 동작이 시작된다. 신호 (4)는 내장 자체 테스트가 끝났음을 알리는 신호이다. (5)는 테스트에 사용할 알고리즘을 지정해주는 외부에서 입력받는 신호이다. (6)는 백그라운드 데이터를 선택하여 주는 신호로 내장 자체 테스트 구조 내에서 하나의 백그라운드 데이터로 모든 주소 테스트가 끝나면 자동으로 다음 백그라운드 데이터로 새롭게 테스트를 시작한다. (7)는 현재 테스트 중인 메모리의 주소를 나타낸다. 신호(8), (9)은 메모리에 쓰는 데이터 값과 메모리로부터 읽어 들이는 데이터 값을 표현한다. 신호 (10), (11), (12)은 고장 검출에 사용된다. 현재 테스트 중인 주소의 셀에 고장이 존재할 경우, 고장이 발생하였다는 FD 신호 (10)에 '1'을 인가하고, 고장이 발생한 셀의 주소 (11)와 고장이 발생한 데이터를 외부 (12)로 출력시킨다. 신호 (13), (14), (15), (16)는 테스트를 제어하는 신호로서 동작 순서는 다음과 같다.

테스트 인가 신호 (1)가 들어오게 되면 선택된 알고리즘 (5)에 따라 March Element Code (13)와 March Element를 실행하라는 신호 (15)를 발생 시킨다. 이에 따라, 테스트 제어 모듈은 해당 March Element에 따른 읽기/쓰기 동작을 하나의 주소에 수행하게 되고 하나의 주소에 대한 동작이 끝나면 읽기/쓰기 동작이 끝났음을 알리는 신호 (16)에 '0'을 인가한다. 신호 (16)가 '0'으

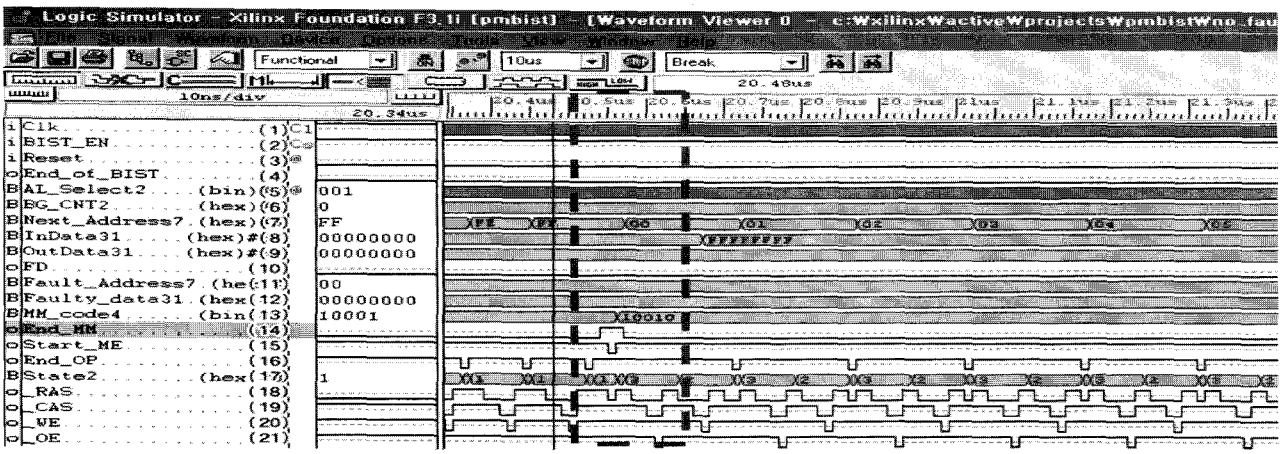


그림 10. 정상동작 메모리 테스트 결과 파형
Fig. 10. Simulated waveform of normal memory test.

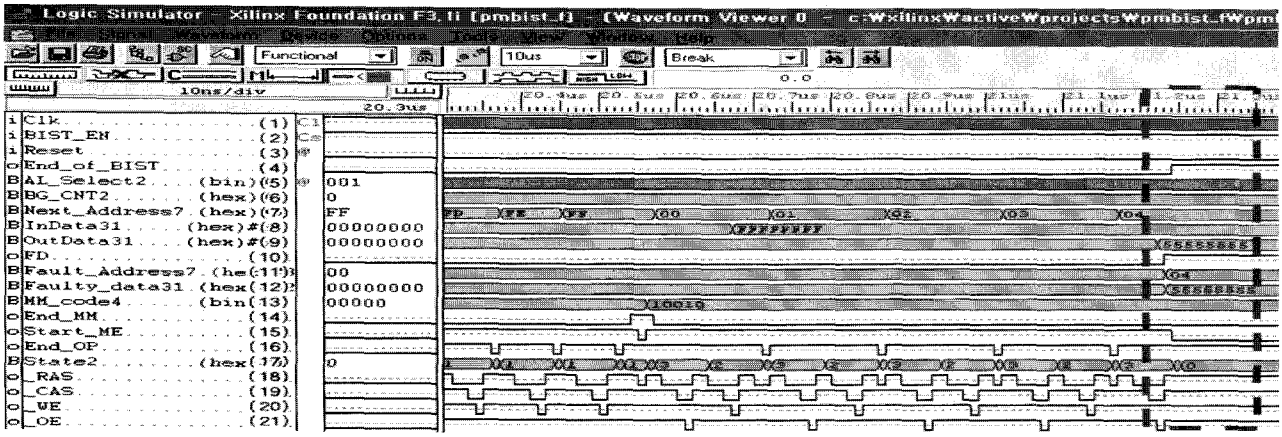


그림 11. 고장 메모리 테스트 결과 파형
Fig. 11. Simulated waveform of faulted memory test.

로 떨어지게 되면 주소 생성 모듈에서는 주소를 March Element의 다섯 번째 키트에 따라 주소 증감을 결정하여 다음 주소 (7)를 발생 시킨다. 나머지 신호 (17), (18), (19), (20), (21)은 현재 동작 상태를 알려주는 신호와 메모리에 Row 주소를 할당하는 신호와 Column 주소를 할당하는 신호, Write Enable, Out Enable 신호이다. 각 신호는 메모리 제어 신호의 동작 순서에 맞게 발생 된다.

그림 10의 테스트 시간 20.55us를 보면 테스트 제어 모듈에서 모든 주소에서 이전 March element의 수행을 마치고 End_MM 신호 (14)를 보내게 되면 알고리즘 생성 모듈에서는 End_MM 신호 (14)를 받고나서 Start_ME 신호 (15)에 '0'을 인가하여 새로운 March element code (10010)를 준비한다. 다음 차례의 March element가 준비되면 MM_code (13)에 해당 코드를 할당 한 후, 다시 Start_ME 신호 (15)에 '1'을 인하가여 새로운 March element가 준비되었음을 알린다. 테스트 제어 모듈에서는 Start_ME 신호 (15)가 인가되면 End_MM 신호 (14)를 '0'으로 내린 후, 새로 받은 MM_code (13)에 따라 테스트를 수행한다.

그림 11은 고장이 존재하는 메모리에 대한 테스트 결과 파형이다. 해당 메모리에는 주소 4에 16진수 '55555555h'로 고착되는 고장이 존재한다. 그림 7을 보면 '10010' March Element (13)를 수행하는 것을 볼 수 있다. 해당 March Element는 'r0/w1'으로 메모리에서 읽어온 값을 '0'과 비교하고 메모리에 '1'을 쓰는 코드이다. 그리하여 메모리의 고장이 없는 주소에서는 '0'을 읽고, '1'을 쓰는 정상 동작을 하는 것을 시간 20.55us와 21.15us사이에서 즉, 주소 0에서 3사이에서 확인 할 수 있고, 쓰기 동작에서는 메모리에 쓰는 데이터 값 (8)이 'FFFFFFFFh'로 인가되어 있음을 확인 할 수 있다.

시뮬레이션 시간 1.2us를 보면, 테스트할 주소가 4가

되고, Row 주소 (18), Column 주소(19) 가 모두 인가되고, Out Enable 신호 (21)가 '0'으로 떨어졌을 때, 해당 주소의 메모리 값을 읽어 온다. 메모리 주소 4에는 미리 '55555555h' 고착 고장을 만들어 놓았으므로 '00000000h'을 읽어야 하지만, 고장 값 '55555555h'를 읽어 오게 되고, 고장이 발생하였음을 알게 된다. 그리하여 테스트 제어 모듈에서는 고장 발생 신호 (10)에 '1'을 인가하고, 고장이 발생한 곳의 주소 (11)와 데이터 값을 신호 (12)에 인가한다. 그리고 고장이 발생하였음으로 더 이상 테스트를 진행 할 필요가 없으므로, End_of_BIST 신호 (4)를 발생시키고 모든 테스트를 중단하게 된다.

2. 플래시를 이용한 고장 메모리 복구 기술 검증

고장 메모리 복구 기술에 대한 검증은 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:00000001h), 고장 3(주소:FF, 고장 위치:FFFFFF20h)을 인식하게 되고 그 고장에 대해 제안하는 재배치 알고리즘을 통해 그림 12에서 처럼 여분의 메모리로 재배치 할 수 있도록 재배치 정보를 출력하게 된다.

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

Reallocation Logic을 통해 재배치가 끝나게 되면 고장난 주소 (9)와 고장 난 위치 (10), 여분의 메모리 재배치 주소 (11), 여분의 행 메모리로 재배치되었는지 여분의 열 메모리로 재배치되었는지에 대한 구분 신호 (12)를 출력하게 된다. 고장 1과 고장 2, 고장 3은 그림 5에서와 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 BISRFail 신호 (5)가 '1'로 활성화 된다.

Xilinx Foundation 3.1i 을 사용하여 제안하는 메모리 테스트 구조를 구현하여 보았다. 테스트 할 메모리의 크기를 1KByte, 16KByte, 256KByte로 정하였고, 각 메모리는 32bit 워드를 가지며 28, 212, 216 크기의 주소를 가진다. 각 메모리 크기에 따른 테스트 회로의 오버헤드를 그림 13에서 비율로 표시하였다.

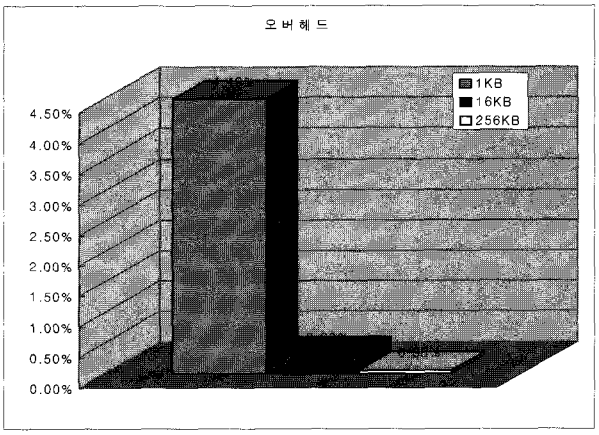


그림 13. 제안하는 메모리 테스트의 메모리 크기별 오버헤드

Fig. 13. Overhead of proposed memory test for each memory size.

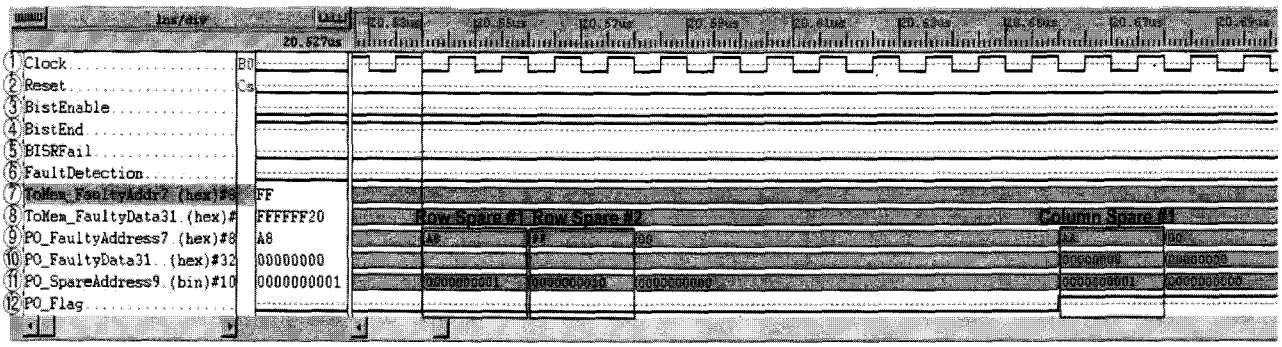


그림 12. 재배치 회로의 결과 파형

Fig. 12. Simulated waveform of Reallocation Logic.

표 3. Xilinx로 구현한 결과 값
Table 3. Result of implementation from Xilinx.

알고리즘	t_{min}(ns)	f_{max}(MHz)
MATS+	9.7	103
March X	8.7	114
March C-	9.5	105
March A	10.2	98
March B	11.8	84
Zero One	9.8	101
기존 PMBIST	17.4	57
제안하는 PMBIST	11.955	83.647

그림 13을 보면 메모리의 크기가 1KByte인 메모리를 테스트하는 메모리 테스트 회로의 오버헤드는 4.49%이고, 16KByte 메모리를 테스트하는 회로의 오버헤드는 0.30%, 256KByte 메모리의 경우 0.05%의 오버헤드를 가졌다. 메모리는 용량이 커짐에 따른 면적 증가율이 해당 메모리를 테스트하는 회로의 크기 증가율보다 매우 크게 증가 하였다. 즉, 메모리 면적은 메모리 크기에 거의 비례하게 증가하였지만, 제안하는 테스트 회로는 비슷한 수준의 크기를 유지하는 것을 알 수 있다.

표 3은 Xilinx Foundation을 이용하여 구현 후의 타이밍 정보이다. 상위 6개의 알고리즘은 기존 존재하는 하나의 알고리즘만을 지원하는 테스트 구조이고 7번째 알고리즘은 기존 프로그램 가능한 메모리 내장 자체 테스트 구조이다[18]. 그리고 마지막은 제안하는 프로그램 가능한 메모리 내장 자체 테스트 구조의 결과 값이다.

제안하는 메모리 테스트는 기존의 프로그램 가능한 내장 자체 테스트[18]보다 더 많은 알고리즘을 지원한다. 그리하여 Read Destructive Coupling fault, Write Disturb Fault, Transition Coupling Fault, Incorrect Read fault 와 같은 가장 최근에 모델링된 고장도 검출이 가능하며, 단순한 MATS+ 알고리즘을 이용하면 SF만을 검출하지만, 가장 빠른 속도로 테스트 할 수 있다. 또한 표 3과 같이 제안하는 테스트 회로는 기존의 테스트 보다 더 많은 고장을 검출하지만, 그 시간은 증가하지 않고 오히려 좀 더 줄어든 것을 알 수 있다. 지원하는 알고리즘이 많음에 따라 테스트 회로의 크기가 증가하였지만, 회로의 면적 증가 오버헤드는 그림 13에서 보는 바와 같이 메모리 회로의 크기에 비하여 매우 작은 것을 알 수 있다.

V. 결 론

내장 메모리 기술 발달에 따라 많은 시스템에 메모리가 내장되게 되었다. 이에 따라 내장되어 있는 메모리에 대한 테스트의 정확성과 비용, 시간이 중요한 문제로 부각되었다. 따라서 본 논문에서는 적은 오버헤드를 가지고, 빠른 속도로 동작하며, 다양한 알고리즘을 지원하는 프로그램 가능한 메모리 내장 자체 테스트를 개발하였다.

제안하는 프로그램 가능한 메모리 내장 자체 테스트는 메모리에 내장 되어 테스트를 실행하므로, 외부 테스트 환경의 제약 없이 테스트 가능하고, 여러 개의 알고리즘을 지원하여 생산과정의 필요성에 따라 다양한 알고리즘을 설계 변경 없이 적용할 수 있게 되었다.

본 논문에서 제안하는 프로그램 가능한 내장 자체 테스트는 기존의 내장 자체 테스트[18]와 비교하여 지원 가능한 알고리즘의 개수가 더 많고, 더 빠른 속도로 동작이 가능하다.

또한, 제안하는 테스트 구조에서 검출해 낸 고장이 발생한 데이터와 주소 값을 사용하여 고장이 발생한 내장 메모리를 사용하는 사용자가 여분의 메모리를 이용해 고장이 없는 내장 메모리처럼 사용할 수 있도록 고장난 내장 메모리의 부분을 여분의 메모리로 재배치하였다. 재배치 기법은 여분의 메모리를 효율적으로 다루기 위해 행과 열로 나누는 알고리즘을 제안하였다.

행과 열로 재배치하는 방법으로는 고장이 발생한 하나의 셀을 포함하고 있는 행의 고장 개수와 열의 고장 개수를 비교해 고장이 발생한 셀의 개수가 많은 쪽 여분의 행 혹은 열 메모리에 재배치를 한다. 만약 가장 고장이 많은 셀의 위치와 고장 난 셀이 겹치게 되면 여분의 행 메모리로 재배치를 하고(행을 기준으로 했을 때), 이전에 열로 재배치가 되었으면 여분의 열 메모리로 재배치를 하는 방법을 사용하였다.

기존의 재배치 정보는 휘발성 메모리에 저장되어 있어 전원이 끊어지게 되면 재배치 정보를 다시 얻어야만 한다. 전원이 끊어진 후 고장 난 메모리를 사용할 때에는 매번 메모리 자체 테스트를 통해 재배치 정보를 가지고 메모리 자체 복원을 수행해야만 한다.

본 논문에서는 플래시 메모리를 사용해 재배치 정보를 비휘발성 메모리인 플래시 메모리에 저장한다. 이 저장된 재배치 정보를 이용해 전원이 끊어져도 고장 난 메모리를 사용할 때에는 다시 메모리 자체 테스트를 수행하지 않고 재배치 정보를 플래시 메모리로부터 읽어와 고장 난 메모리의 고장 위치를 파악해 자체 복원할 수 있다. 제안한 재배치 알고리즘과 구조를 사용하여 BISR 회로에 적용시켰을 경우 메모리 고장에 대해 정

상 동작시키며, 여분의 메모리로 재사용하므로서 메모리의 수율을 증가 시킬 수 있을 것이다.

참 고 문 헌

[1] A. J. van de Goor and A. Offerman, "Towards a uniform notation for memory tests," *in Proc. European Design and Test Conf.*, pp. 420-427, 1996.

[2] V. G. Mikitjuk, V. N. Yarmolik and A. J. van de Goor, "RAM testing algorithms for detecting multiple linked faults," *in Proc. European Design and Test Conf.*, pp. 435-439, 1996.

[3] P. H. Bardell and W. H. McAnney, "Built-in test for RAMs," *IEEE Design & Test of Computers,* Vol. 5, No. 4, pp. 29-36, Aug 1988.

[4] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A tutorial on built-in self-test. I. Principles," *IEEE Design & Test of Computers,* Vol. 10, No. 1, pp. 73-82, Mar 1993.

[5] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A tutorial on built-in self-test. 2. Principles," *IEEE Design & Test of Computers,* Vol. 10, No. 2, pp. 69-77, Mar 1993.

[6] S. Park, K. Lee, C. Im, N. Kwak, K. Kim and Y. Choi, "Designing built-in self-test circuits for embedded memories test," *in Proc. AP-ASIC 2000, 2nd IEEE Asia Pacific Conf.*, pp. 315-318, Aug 2000.

[7] K. Zarrineh and S. J. Upadhyaya, "On programmable memory built-in self test architectures," *in Proc. IEEE Design, Automation and Test in Europe Conf.*, pp. 708-713, Mar 1999.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] V. Schober, S. Paul, and O. Picot, "Memory Built-In Self-Repair using redundant words", *Proceedings of International Test Conference*, 2001.

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

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

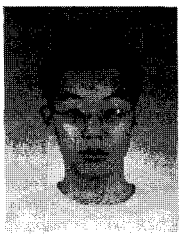
[17] S. Hamdioui, A. J. van de Goor and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," *Memory Technology, Design and Testing, 2002. (MTDT 2002). in Proc. The 2002 IEEE International Workshop*, pp. 95-100, July 2002.

[18] P. C. Tsai, S. J. Wang and F. M. Chang, "FSM-Based Programmable Memory BIST with Macro Command," *in Proc. The 2005 IEEE International Workshop on Memory Technology, Design, and Testing*, pp. 72-75, Aug 2005.

저 자 소 개

홍 원 기(학생회원)

2005년 숭실대학교 컴퓨터학부 학사 졸업.
2007년 숭실대학교 대학원 컴퓨터학과 석사 졸업.
2007년~현재 숭실대학교 대학원 컴퓨터학과 박사 과정.
<주관심분야 : 메모리 테스트, VLSI 설계 및 테스트, 컴퓨터구조, 임베디드 시스템>

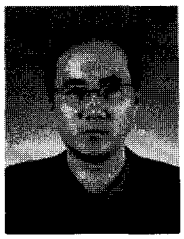


심 은 성(학생회원)

2003년 한서대학교 컴퓨터 정보학과 학사 졸업.
2005년 숭실대학교 대학원 컴퓨터학과 석사 졸업.
2007년~현재 숭실대학교 대학원 컴퓨터학과 박사 과정.
<주관심분야 : 메모리 테스트, VLSI 설계 및 테스트, 컴퓨터구조, 임베디드 시스템>

최 정 대(학생회원)

2007년 숭실대학교 컴퓨터학부 학사 졸업.
2008년~현재 숭실대학교 대학원 컴퓨터학과 석사 과정.
<주관심분야 : 컴퓨터구조, 메모리 테스트, VLSI 설계 및 테스트, 임베디드 시스템>



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 : 컴퓨터구조, 메모리 테스트, VLSI 설계 및 테스트, 임베디드 시스템>