# 결함 주입 기반 고장 검출을 위한 프로그램 가능한 BIST 구조

## Programmable BIST Architecture to find faults based on defect injection

이창욱, 홍원기, 장훈
숭실대학교
culee@watt.ssu.ac.kr

## Abstract

The density of memory has been increased by great challenge for memory technology. Therefore, elements of memory become smaller than before and the sensitivity to faults increases. As a result of these changes, memory testing becomes more complex. Recent development in system-on-chip (SOC) technology makes it possible to incorporate large embedded memories into a chip. However, it also complicates the test process, since usually the embedded memories cannot be controlled from the external environment. Proposed design doesn't need controls from outside environment, In general, there are a variety of memory modules in SOC, and it is not possible to test all of them with a single algorithm. Thus, the proposed scheme supports he various memory testing process. Moreover, it is able to detect faults based on defect injection.

## I. 서론

1. 고장 모델링

새로운 메모리 기술이 개발될 때마다 메모리의 집적도가 증가하게 되었다. 그에 따라 고장의 종류가 다양해지고, 테스트에 필요한 시간이 증가하게 되어, 테스트에 사용되는 비용도 함께 증가되어왔다[1],[2]. 테스트 비용과 시간을 줄이기 위해서는 정확한 고장 모델링이 필요하다. 테스트의 고장 검출률은 테스트에 사용되는 고장 모델링에 의해 결정되어 진다. 모델링을 단순하게 하면 속도는 빠르지만, 다양한 고장들을 정의할 수 없게 되어 고장 검출률이 낮아지게 된다. 그러므로 고장 모델링이 실제 메모리에서의 고장을 정확히 반영하고 있어야만 높은 고장 검출률을 얻을 수 있다.

1980년대에 많은 메모리 고장 모델이 소개 되었다. Address Decoder Faults, Stuck-At Faults가 모델링 되었지만, 이것들은 매우 추상적이었기에 실제 메모리 고장을 완벽하게 반영할 수 없었다. 그 후, 실제 디자인에서의 실제 고장을 반영하기 위해서 물리적 레이아웃 수준의 실험 결과를 바탕으로 State-Coupling Fault, Data-Retention Fault 고장 모델이 성립되었다. 하지만 메모리의 크기가 커짐에 따라 기존의 기능 고장 모델로는 모든 고장을 표현하는 것이 불가능해졌다.

메모리의 집적도가 커지고 더 작은 공정이 사용됨에 따라 새로운 고장이 발생하게 되었고, 새로운 고장에 대한 모델링이 필요하게 되었다. 1990년대 후반, 새롭게 제안된 고장 모델은 결함 주입 기반의 회로 시뮬레이션을 통하여 모델링 되었다. 새롭게 소개된 고장 모델은 Read Destructive Coupling fault, Write Disturb Fault, Transition Coupling Fault, Incorrect Read fault 등이다[3]. 새로운 고장이 모델링 되었기 때문에, 새롭게 추가된 고장을 검출 할 수 있는 새로운 알고리즘의 필요성 또한 대두 되었다.

2. 고장 검출 알고리즘

| | Zero-One | MATS+ | Marching 1/0 | March X | March C- | March A | March B | March U | March LR | March SS |
|---|---|---|---|---|---|---|---|---|---|---|
| SF | △ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| TF | ✖ | △ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| WDF | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ |
| RDF | ✖ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| DRDF | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ |
| IRF | ✖ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| CFst | ✖ | △ | △ | △ | ○ | △ | △ | ○ | ○ | ○ |
| CFdsrx | ✖ | △ | △ | △ | ○ | △ | △ | ○ | ○ | ○ |
| CFdsxwx | ✖ | △ | △ | △ | ○ | ○ | ○ | ○ | ○ | ○ |
| CFdsxw'x | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ |
| CFtr | ✖ | △ | △ | △ | ○ | △ | △ | ○ | ○ | ○ |
| CFwd | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ |
| CFrd | ✖ | △ | △ | △ | ○ | △ | △ | ○ | ○ | ○ |
| CFdrd | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ |
| CFir | ✖ | △ | △ | △ | ○ | △ | △ | ○ | ○ | ○ |

표 1 다양한 알고리즘의 고장 검출률

다양한 고장이 모델링 됨에 따라 많은 고장 검출 알고리즘들이 개발되었다. 단순히 SF 고장만을 검출 할 수 있는 Zero-One 알고리즘에서부터 March elements를 기반으로 새롭게 모델링 된 Read Destructive Coupling fault, Write Disturb Fault, Transition Coupling Fault, Incorrect Read fault 등을 검출할 수 있는 March SS 까지 개발되었다.

표 1에서 보는 바와 같이 각 알고리즘이 검출 할 수 있는 고장의 종류는 다르다. 이 중에서 메모리 수율 또는 메모리 생산 과정에서의 변화에 맞춰 즉시 알고리즘을 선택하여 설계하는 것은 테스트 비용과 시간 측면에서 많은 낭비를 발생시킨다. 그리하여, 본 논문에서는 테스트 시간과 비용을 최적화 할 수 있는 알고리즘들을 선택하여 최저의 오버헤드를 가지는 복수개의 알고리즘을 선택 가능한 BIST 구조를 제안한다.

## II. 본론

1. 선택 가능한 테스트 알고리즘

제안하는 프로그램 가능한 메모리 내장 자체 테스트에서는 표 2에서와 같이 총 3개의 알고리즘을 지원한다. 메모리 생산 공정 초기에는 많은 고장이 발생하게 되므로, 현재까지 모델링 된 모든 고장을 검출해 낼 수 있는 March SS 알고리즘이 포함되어있고, 생산이 거듭될수록 공정은 안정화 되므로, 단순하고 빠른 MATS+와 같은 알고리즘을 지원할 수 있게 하였다.

| 번호 | 알고리즘 | March Elements |
|---|---|---|
| | | 코드 |
| 01 | MATS+ (3n) | {↕(w0);↑(r0,w1);↓(r1,w0)} |
| | | 1000 1001 0010 |
| 10 | March C- (6n) | {↕(w0);↑(r0,w1);↑(r1,w0);↓(r0,w1);↓(r1,w0);↕(r0)} |
| | | 1000 1001 1010 0001 0010 1011 |
| 11 | March SS (6n) | {↕(w0);↑(r0,r0,w0,r0,w1);↑(r1,r1,w1,r1,w0);↓(r0,r0,w0,r0,w1);↓(r1,r1,w1,r1,w0);↕(r0)} |
| | | 1000 1100 1101 0100 0101 1011 |

표 2 알고리즘 선택표

표 2에서의 번호는 외부 ATE 장비에서 내장 자체 테스트를 실행하며 실행할 알고리즘의 코드이다. 코드는 March Elements 마다 4비트로 되어 있으며, 메모리에 적용할 March Elements를 가리킨다. 코드의 최상위 비트는 적용할 March Elements의 주소 증감을 나타낸다. 주소를 증가시키면서 테스트를 해야 하거나, 주소의 증감이 자유로운 March Elements의 경우는 '1'을 부여하고, 반대로 주소를 감소시키면서 적용해야 하는 경우에는 '0'을 할당하였다. 코드의 나머지 3비트는 전체 알고리즘의 March Elements를 정리하여 주소의 증감 여부를 제외한 읽기/쓰기 동작이 동일한 것끼리 묶어 같은 번호를 부여하였다.

2. March Elements

본 논문에서 사용되는 알고리즘들의 공통적인 March Elements를 모아 동일한 코드를 부여함으로써, 알고리즘을 저장하는데 필요한 공간을 최적화 하였다. 정리된 코드는 표 3과 같다.

하나의 알고리즘이 선택 되면 BIST Controller에서는 해당 알고리즘의 March Elements 코드를 발생시킴으로 메모리에 바로 알고리즘을 적용 시킬 수 있다.

| March elements | 코드 [2:0] | 동작 |
|---|---|---|
| M1 | 000 | w0 |
| M2 | 001 | r0,w1 |
| M3 | 010 | r1,w0 |
| M4 | 011 | r0 |
| M5 | 100 | r0,r0,w0,r0,w1 |
| M6 | 101 | r1,r1,w1,r1,w0 |

표 3 March Elements 선택표

3. 제안하는 BIST 구조

검출하고자 하는 고장에 맞춰 선택적인 알고리즘을 선택하기 위한 BIST 구조는 그림 1과 같다. 제안하는 BIST는 선택된 알고리즘의 March Elements 코드를 발생시키는 Algorithm Generator와 해당 코드를 적용시키는 Test Controller로 구성되어 있다.

제안하는 BIST구조는 외부로부터 알고리즘을 선택하는 신호를 받아 Algorithm Generator에 부여하면 Algorithm Generator는 알고리즘에 따라 March Elements 코드를 알고리즘 순서에 따라 Test Controller로 보낸다. Test Controller에서는 Algorithm Generator로부터 받은 코드를 분석하여 최상위 비트 4번째 비트를 보고 주소의 증가 또는 감소를 결정한다. 이 결정에 따라 내부의 AGL은 주소를 발생시키게 되고, 코드의 나머지 3비트에 따라 Read/Write Signal Generator는 메모리에 읽기 또는 쓰기 신호를 메모리 타이밍에 맞춰 발생시켜 테스트를 수행하게 된다.

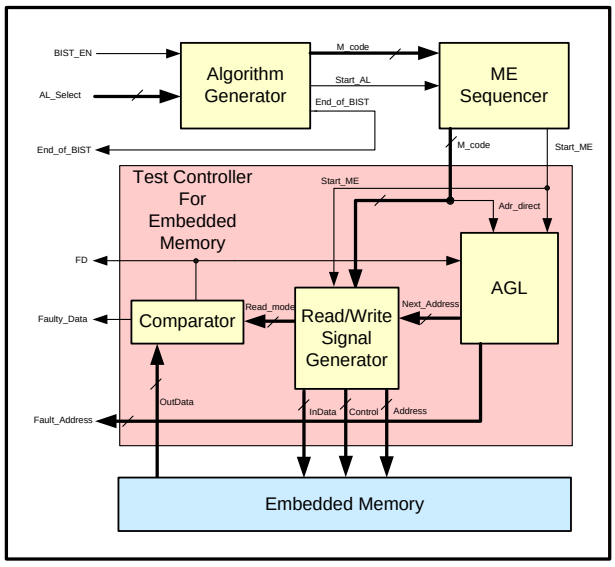


그림 1 제안하는 프로그램 가능한 내장 자체 테스트의 구조

본 논문에서 제안한 내장 자체 테스트 설계에 대한 구현은 VerilogHDL로 기술하여 구현하였다. 구현에 대한 검증은 Xilinx사의 ISE 8.1i에서 제공하는 시뮬레이터를 사용하여 RTL 검증을 하였다.

4. 실험 결과

그림 2는 메모리 내장 자체 테스트가 정상적인 메모리에서 동작하면서 발생하는 파형의 일부이다. 그림 2의 (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 신호이고, (5)는 테스트에 사용할 알고리즘을 지정해주는 외부에서 입력 받는 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 나타내고, 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 신호이다.

그림 2는 고장이 존재하는 메모리에 대한 테스트 결과 파형이다. 해당 메모리에는 주소 4에 16진수 '55555555h' 로 고착되는 고장이 존재한다. 그림 2를 보면 '1001' March Elements (13)를 수행하는 것을 볼 수 있다. 해당 March Elements는 '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 210540ns전후에서 즉, 주소 0에서 3사이에서 확인 할 수 있고, 쓰기 동작에서는 메모리에 쓰는 데이터 값 (8)이 'FFFFFFFFh' 로 인가되어 있음을 확인 할 수

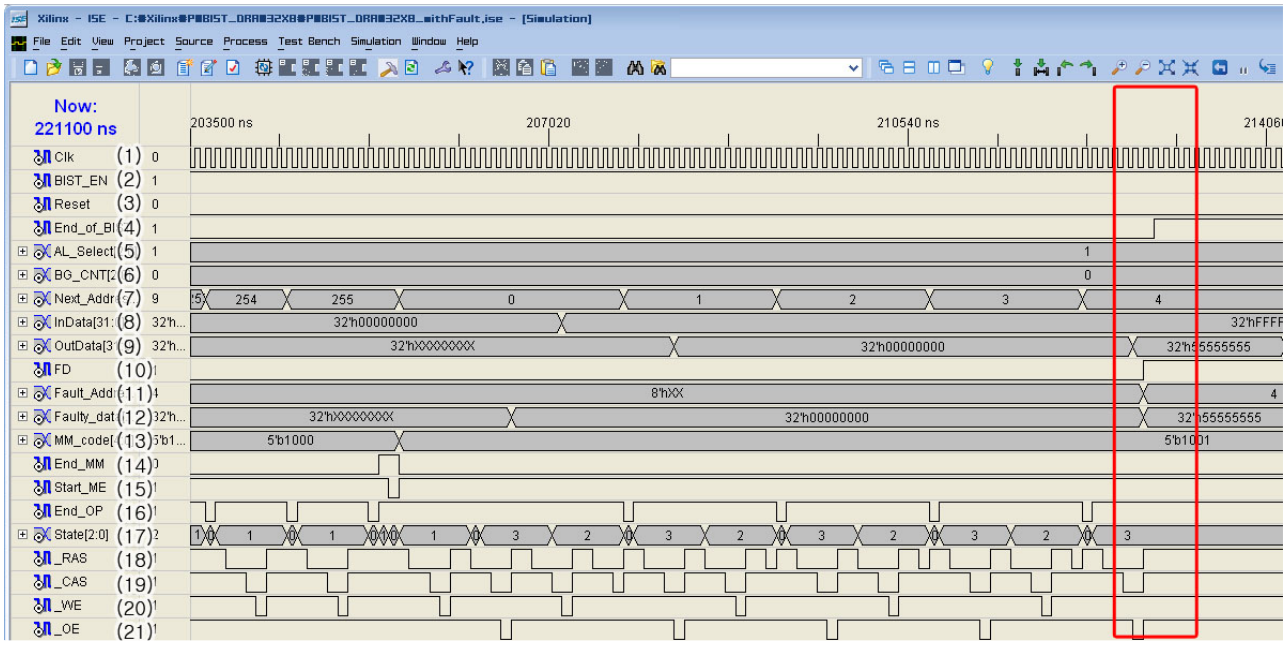


그림 2 고장 발생 메모리 테스트 결과 파형

있다.

시뮬레이션 시간 210540ns 이후의 빨간 네모 부분을 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.

## III. 결론

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 테스트 구조를 개발하였다. 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, 빠른 속도로 테스트 할 수 있다. 제안하는 프로그램 가능한 메모리 내장 자체 테스트는 메모리에 내장 되어 테스트를 실행하므로, 외부 테스트 환경의 제약 없이 테스트 가능하고, 복수 개의 알고리즘을 지원하여 생산과정의 필요성에 따라 다양한 알고리즘을 설계 변경 없이 적용할 수 있게 되었다. 결과적으로 제안하는 테스트 구조를 사용하면 테스트 시간과 비용을 줄일 수 있을 것으로 예상한다.

## 참고문헌

[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] 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.