논문 2007-44SD-12-10

내장 메모리를 위한 프로그램 가능한 자체 테스트

(Programmable Memory BIST for Embedded Memory)

홍 원 기*, 장 훈**

(Won-Gi Hong 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. 그리고 SOC 기술의 발달로 대용량의 내장 메모리를 통합할 수 있게 되었지만, 테스트 과정이 복잡하게 되어 외부 테스트 환경에서는 내장 메모리를 테스트하기 어렵게 되었다. 본 논문에서 제안하는 테스트 구조는 내장 테스트를 사용하여 외부 테스트 환경 없이 테스트가 가능하다. 제안하는 내장 테스트 구조는 다양한 알고리즘을 적용 가능하므로, 생산 공정의 수율 변화에 따른 알고리즘 변화에 적용이 가능하다. 그리고 메모리에 내장되어 테스트하므로, At-Speed 테스트가 가능하다. 즉, 다양한 알고리즘과 여러 형태의 메모리 블록을 테스트 가능하기 때문에 높은 효율성을 가진다.

Abstract

The density of Memory has been increased by great challenge for memory technology. Therefore, elements of memory become more smaller than before and the sensitivity to faults increases. As a result of these changes, memory testing becomes more complex. In addition, as the number of storage elements per chip increases, the test cost becomes more remarkable as the cost per transistor drops. 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, because it integrates into memory. 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 the various memory testing process. Moreover, it is able to At-Speed test in a memory module. consequently, the proposed is more efficient in terms of test cost and test data to be applied.

Keywords: Memory, BIST, Programmable BIST, Embedded Memory

I. 서 론

메모리의 집적도가 증가함에 따라 메모리 테스트에 필요한 비용도 함께 증가하게 된다. 그 이유는 첫 번째, 각각의 구성요소들의 크기가 작아짐에 따라 고장의 감응성이 증가하게 되고, 고장은 더욱 복잡하게 되어 모델링이 어려워졌기 때문이다. 두 번째 이유는, 칩 하나에 포함되어있는 저장 요소가 늘어남에 따라 테스트 시간도 증가하게 되었기 때문이다. 그리하여 트랜지스터 하나를 생산하는 비용보다 테스트하는데 소비되는 비용이 더 크게 되었다.

최근 급속한 시스템 온 칩(SOC) 기술의 발달로 대용량의 내장 메모리를 통합할 수 있게 되었다[1~2]. ITRS 2000년도 로드맵을 보면 SOC에서 내장 메모리의 비율이 어떻게 변화하였는지를 알 수 있다. 그림 1을 보면 SOC를 구성하는 많은 요소 중에서 내장 메모리가 차지하는 비중은 1999년 20%에서 2002년에는 50%를 넘어선 것을 알 수 있다. 이는 SOC 칩을 테스트하는 과정에서의 시간과 비용을 줄이기 위해서는 내장 메모리를

* 학생회원, ** 정회원, 숭실대학교 컴퓨터학과
(Department of Computer, Soongsil University)

※ 본 연구보고서는 정보통신부 출연금으로 MIC/ IITA/ ETRI, SoC산업진흥센터에서 수행한 IT SoC 핵심설계인력양성사업의 연구결과입니다.
접수일자: 2007년7월19일, 수정완료일: 2007년11월13일

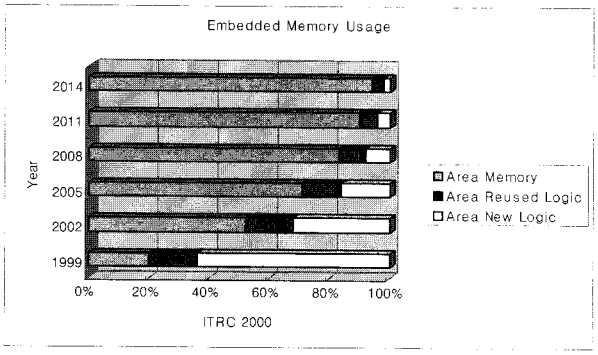


그림 1. 내장 메모리 사용 비율
Fig. 1. Embedded memory usage.

테스트하는 과정이 가장 큰 영향을 미친다는 것을 의미한다. 내장 메모리의 용량은 커져가는 반면에, 크기는 점점 줄어듦에 따라 테스트 과정은 복잡한 알고리즘과 많은 시간이 필요하게 되었다. 따라서 Automatic Test Equipment(ATE)를 이용하여 외부에서 메모리를 접근하여 테스트하는 방법은 매우 오랜 테스트 시간을 필요로 하고, AT-Speed 테스트가 불가능하게 되었다.

내장 자체 테스트 (Built-in Self Test)는 기존의 테스트 문제점들을 해결하는 방법을 제시한다. 메모리 내장 자체 테스트(Memory Built-in Self Test)는 테스트하에 있는 칩을 위한 메모리 테스트 알고리즘을 구현한다[3~6]. 메모리 내장 자체 테스트는 메모리에 내장 되어 있으므로, 외부로부터 많은 수의 포트를 필요로 하지 않고, 메모리의 동작 속도로 테스트가 가능하므로, At-Speed 테스트가 가능하다.

그러나 이러한 테스트 구조는 이미 테스트 회로 설계 시 정해진 하나의 테스트 알고리즘만을 적용 가능하다. 하나의 알고리즘만 사용하여 테스트 하는 방법은 다음과 같은 이유에서 적합하지 않다.

첫 번째 이유로 메모리의 생산 과정이 반복되는 동안 메모리 장치에 필요한 테스트 알고리즘이 변하게 된다. 생산 초기에는 비록 복잡하고 시간이 오래 걸리더라도, 다양한 고장을 찾아내는 알고리즘이 필요하지만, 생산이 거듭될수록 생산 공정이 안정화되어 수율이 높아지므로 더 간단하고 빠른 알고리즘으로 테스트가 가능하다. 두 번째로 SOC는 다양한 형식의 메모리 블록을 포함한다. 각 메모리 형식에 따라 그에 적합한 테스트 알고리즘들이 필요하다. 메모리 생산 공정의 수율과 메모리 블록에 따른 적합한 알고리즘들을 모두 지원하려면 모든 알고리즘을 구현한 회로를 내장하여야 하므로 오버헤드가 증가하게 된다.

반면에, 다양한 테스트 알고리즘을 지원하는 프로그램 가능한 메모리 내장 자체 테스트(Programmable Memory Built-in Self Test)는 테스트가 필요한 시점의 환경에 적합한 테스트 알고리즘을 선택 가능하므로, 높은 효율성을 가지고 있다. 프로그램 가능한 메모리 내장 자체 테스트는 마이크로 코드를 이용한 방법과 유한상태머신(FSM)을 이용한 접근방법으로 나누어진다[7].

일반적으로 마이크로 코드를 이용한 방법이 다양한 알고리즘의 적용이 쉽기 때문에 많이 사용된다. 마이크로 코드를 이용한 테스트에서는 알고리즘의 각 단계를 가리키는 명령어들이 정의되어있다. 알고리즘의 각 단계는 미리 정의된 명령어들에 의해서 구성된다. 즉, 테스트 알고리즘의 각 단계마다 명령어가 필요하게 되고, 매번 외부에서 내장 자체 테스트로 명령어를 전송해 주어야 한다. 이 방법은 자유롭게 테스트에 사용할 알고리즘을 선택할 수 있는 장점이 있지만, 내장 자체 테스트 회로의 복잡도가 커지고, 오버헤드가 증가하게 된다. 그리하여, SOC 안에 존재하는 내장 메모리라면, ATE는 테스트를 수행하기 위한 많은 명령어를 전송해야하므로, 외부의 명령어를 입력 받을 수 있는 연결 핀이 필요하고, 테스트 시간 또한 증가하게 된다[7].

본 논문에서는 매크로 코드에 의해 동작하는 FSM방식의 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. 제안하는 구조에서는 두 가지 FSM이 존재 한다. 첫 번째는 알고리즘을 선택하여 March Elements를 만들어주고 만들어진 March Elements를 테스트 제어 모듈에 적절한 시간에 맞추어 보내어 주는 FSM과 메모리에 접근할 때 사용하는 FSM으로 구성되어 진다.

본 논문의 구성은 서론에 이어 2장 기존의 고장 모델링과 메모리 내장 자체 테스트 알고리즘을 설명하고, 3장에서는 제안하는 프로그램 가능한 메모리 내장 자체 테스트의 구조를 언급한다. 마지막으로, 4장에서는 제안하는 내장 메모리 자체 테스트 구조를 검증하고, 5장 결론으로 본 논문을 마친다.

Ⅱ. 기존 연구

1. 고장 모델링

새로운 메모리 기술이 개발될 때마다 메모리의 집적도가 증가하게 되었다. 그에 따라 고장의 종류가 다양

해지고, 테스트에 필요한 시간이 증가하게 되어, 테스트에 사용되는 비용도 함께 증가되어왔다. 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 회로 시뮬레이션을 통하여 모델링 되었다. 새롭게 소개된 고장 모델은 Read Destructive Coupling fault, Write Disturb Fault, Transition Coupling Fault, Incorrect Read fault 등 이다[8]. 새로운 고장이 모델링되었기 때문에, 새롭게 추가된 고장을 검출 할 수 있는 새로운 알고리즘의 필요성 또한 대두 되었다.

2. March 테스트 알고리즘

메모리 셀이 정상적으로 동작하는지 확인하기 위해서는 특정한 순서로 읽기 동작과 쓰기 동작을 수행하여야 한다. 읽기 동작과 쓰기 동작의 횟수와 순서는 찾아내고자 하는 고장의 종류에 따라 달라진다. 최근 내장 메모리의 고장을 검출하는데 가장 많이 사용되는 알고리즘이 March 테스트기반 알고리즘이다. March 테스트 알고리즘은 March Elements로 구성되어 있다. 그리고 March Elements는 메모리 셀에 적용되는 값과 읽기, 쓰기 동작으로 구성되어 있다. 메모리의 크기가 N이라 할 때, March Element는 주소 0에서 N-1까지 증가 하면서, 또는 N-1에서 0까지 감소하면서 각 메모리 셀에 주어진 하나 이상의 읽기, 쓰기 동작을 수행하며 고장을 검출해낸다. March Elements를 구성하는 요소는 표 1과 같다.

표 1. March elements 구성 요소
Table 1. Summary of notations.

기호	동작
r	읽기 동작
w	쓰기 동작
	주소 증가
	주소 감소
	주소 증감 모두 가능

표 2. March elements를 사용한 다양한 내장 메모리 테스트 알고리즘
Table 2. Various embedded memory test algorithms using march elements.

Algorithm	March Elements
Zero-One	{(w0); (r0); (w1); (r1);}
MATS+	{(w0); (r0,w1); (r1,w0)}
Marching 1/0	{(w0); (r0,w1,r1); (r1,w0,r0); (w1); (r1,w0r0); (r0,w1,r1);}
March X	{(w0); (r0,w1); (r1,w0); (r0)}
March C-	{(w0); (r0,w1); (r1,w0); (r0,w1); (r1,w0); (r0)}
March A	{(w0); (r0,w1,w0,w1); (r1,w0,w1); (r1,w0,w1,w0); (r0,w1,w0)}
March B	{(w0); (r0,w1,r1,w0,r0,w1); (r1,w0,w1); (r1,w0,w1,w0); (r0,w1,w0)}
March U	{(w0); (r0,w1,r1,w0); (r0,w1); (r1,w0,r0,w1); (r1,w0)}
March LR	{(w0); (r0,w1); (r1,w0,r0,w1); (r1,w0); (r0,w1,r1,w0); (r0)}
March SS	{(w0); (r0,r0,w0,r0,w1); (r1,r1,w1,r1,w0); (r0,r0,w0,r0,w1); (r1,r1,w1,r1,w0); (r0)}

예를 들어 (w0,r0) 은 메모리 셀에 '0'을 쓰는 동작을 하고 읽어온 값이 '0'인지 확인한 후, 주소가 증가하는 방향으로 다음 주소에 동일한 읽기, 쓰기 동작을 반복 수행하라는 March Element이다. (r1) 은 읽어온 값을 '1'과 비교한 후, 주소 증감 방향에 관계없이 일정한 방향으로 읽기 동작을 수행하라는 의미이다.

표 2는 March elements를 기반으로 하는 다양한 테스트 알고리즘을 보여준다. 각 알고리즘은 각기 다른 March elements로 구성되어져 있기 때문에, 알고리즘에 따라 검출 가능한 고장 모델들이 다르다. 표 3은 각 March 테스트 기반 알고리즘마다의 고장 검출률을 보여준다. ○는 모두 검출 가능하다는 의미이고, △는 부분적으로 검출 가능, ✖는 검출이 전혀 불가능하다는 의미이다.

회로 시뮬레이션을 통하여 새롭게 모델링된 Read Destructive fault, Write Disturb Fault, Transition Coupling Fault, Read destructive fault 등의 실질적인

표 3. 다양한 March 알고리즘의 고장 검출률
Table 3. Fault coverage of each march algorithm.

	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	×	△	△	△	○	△	△	○	○	○

메모리 고장을 가장 잘 반영하는 알고리즘은 표 3에서 보는 바와 같이 March SS 이다[9].

III. 제안하는 프로그램 가능한 내장 자체 테스트

1. 선택 가능한 테스트 알고리즘

제안하는 프로그램 가능한 메모리 내장 자체 테스트에서는 표 4에서와 같이 총 7개의 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.

표 4에서의 Selection은 외부 ATE 장비에서 내장 자체 테스트를 실행하며 실행할 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 하여야 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. 예를 들어, 모든

표 4. 알고리즘 선택표
Table 4. Algorithm selection table.

번호	알고리즘	March Elements 코드
001	MATS+ (3n)	{ (w0); (r0,w1); (r1,w0)}
		10001 10010 00011
010	March X (4n)	{ (w0); (r0,w1); (r1,w0); (r0)}
		10001 10010 00011 10100
011	March C- (6n)	{ (w0); (r0,w1); (r1,w0); (r0,w1); (r1,w0); (r0)}
		10001 10010 10011 00010 00011 10100
100	March B (5n)	{ (w0); (r0,w1,r1,w0,r0w1); (r1,w0,w1); (r1,w0,w1,w0); (r0,w1,w0)}
		10001 10101 10110 00111 01000
101	March U (5n)	{ (w0); (r0,w1,r1,w0); (r0,w1); (r1,w0,r0,w1); (r1,w0)}
		10001 11001 10010 01010 00011
110	March LR (6n)	{ (w0); (r0,w1); (r1,w0,r0,w1); (r1,w0); (r0,w1,r1,w0); (r0)}
		10001 00010 11010 10011 11001 10100
111	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)}
		10001 11011 11100 01101 01110 10100

알고리즘은 (w0) 로 시작하게 되어있다. 그러므로 (w0) 는 주소의 증감이 자유롭기 때문에 최상위 비트를 '1'로 할당하고, (w0)를 의미하는 0001 March Element 번호를 부여하였다.

즉, 모든 알고리즘의 총 March Element수는 35개 이지만, 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 5와 같다.

2. 제안하는 프로그램 가능한 내장 자체 테스트의 구조

제안하는 프로그램 가능한 내장 자체 테스트는 그림 2와 같은 구조로 되어 있다. 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.

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 값

표 5. March Elements 선택표
Table 5. 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

과 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 읽기/쓰기 동작으로 바꾸고, 테스트할 주소를 발생시킨다. 그리고 테스트할 메모리에 맞는 신호를 발생 시킨다.

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 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 상태에서 외부 입력을 기다리게 된다. BIST_EN 신호가 인가되면 BIST_Enable

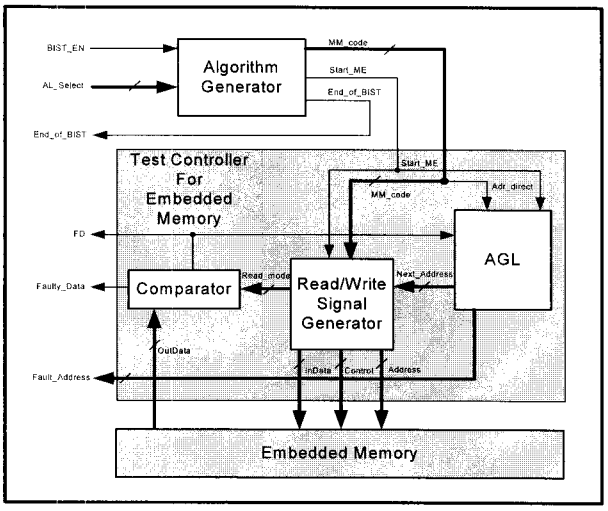


그림 2. 제안하는 프로그램 가능한 내장 자체 테스트의 구조
Fig. 2. Proposed programmable memory BIST

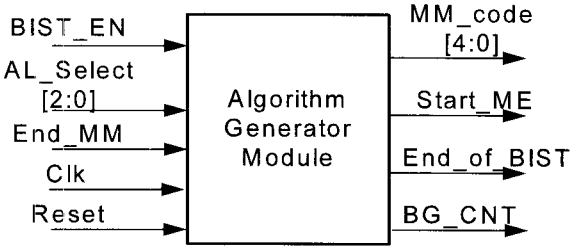


그림 3. 알고리즘 생성 모듈의 블록 다이어그램
Fig. 3. Block diagram of algorithm generation module.

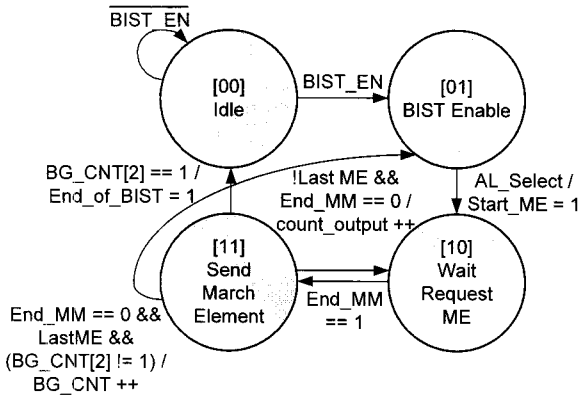


그림 4. 알고리즘 생성 모듈의 FSM
Fig. 4. FSM of algorithm generation module.

상태로 전환되고, 외부로부터 AL_Select 신호를 입력받으면 해당 알고리즘을 준비하고 알고리즘의 March element가 발생되었음을 알리는 Start_ME 신호를 인가하게 된다. 알고리즘 생성 모듈은 End_MM 신호를 받을 때마다 순서대로 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_BIST를 인가하여 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 발생시킨다. 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/w1상태에서는 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

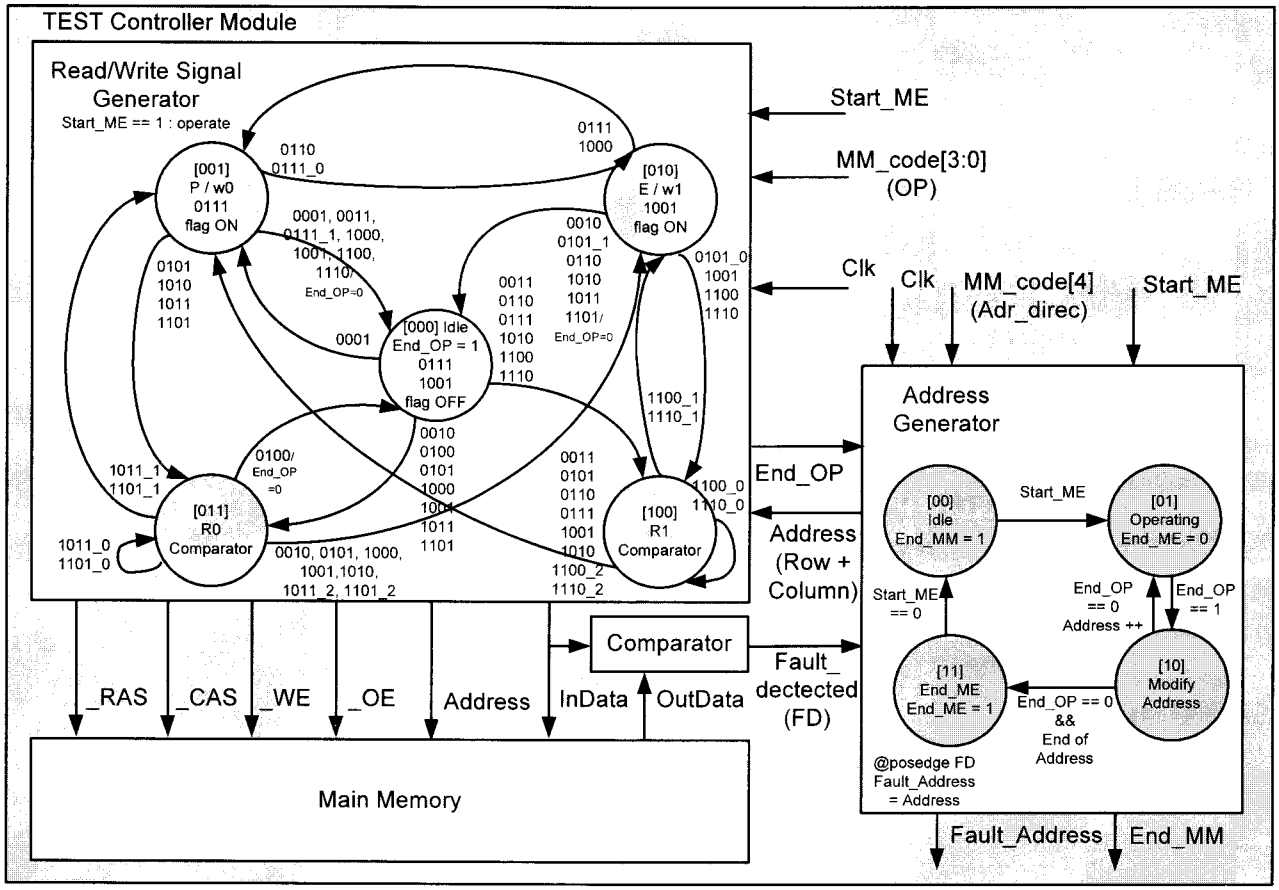


그림 5. 테스트 제어 모듈의 FSM
Fig. 5. FSM of test controller module.

Address, 그리고 마지막으로 메모리에 읽고, 쓰게 되는 값 OutData, InData이다.

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

IV. 실험 결과

1. 프로그램 가능한 내장 자체 테스트 구조 검증

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.

그림 6은 메모리 내장 자체 테스트가 정상적인 메모

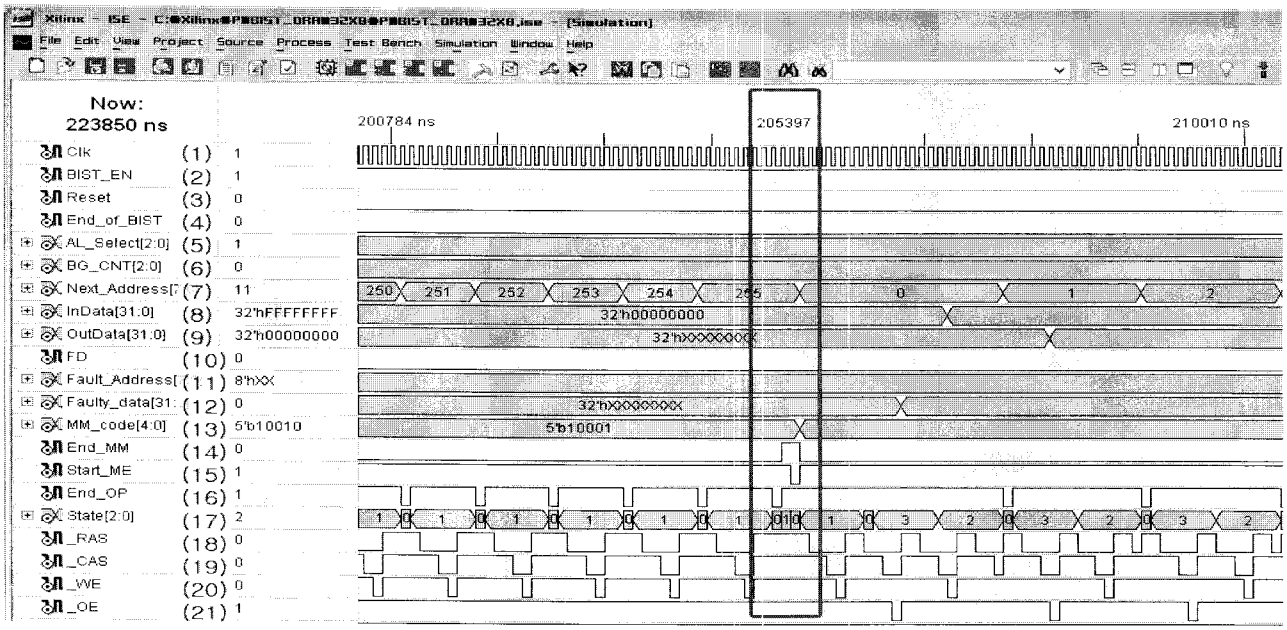


그림 6. 정상동작 메모리 테스트 결과 파형

Fig. 6. Simulated waveform of normal memory test.

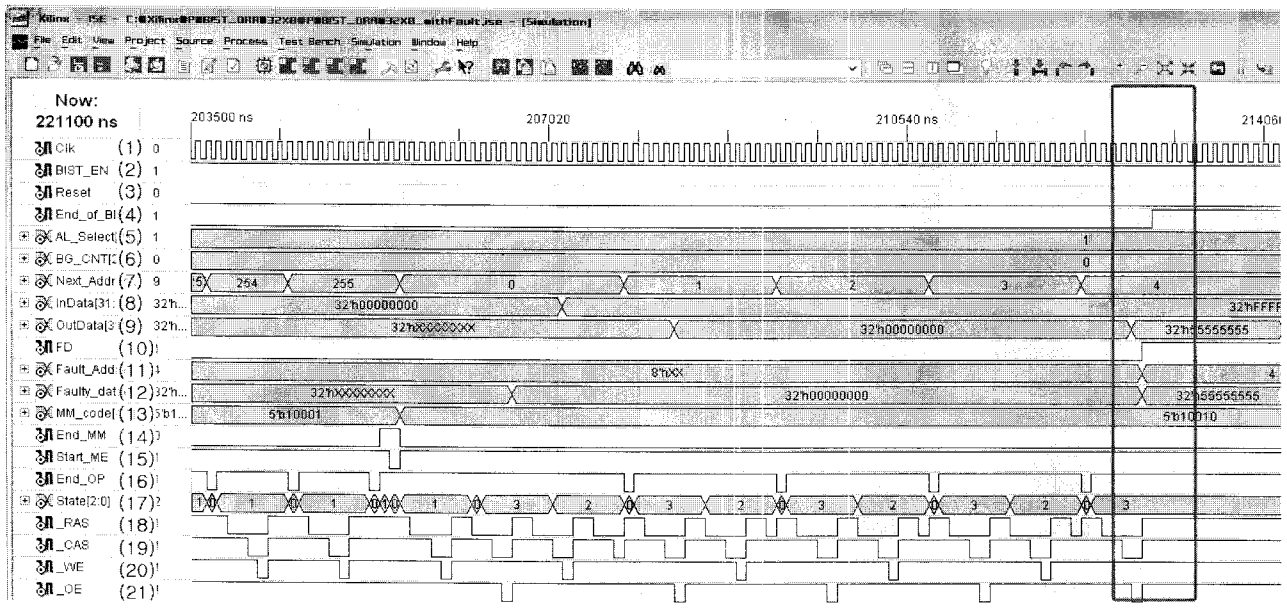


그림 7. 고장 메모리 테스트 결과 파형

Fig. 7. Simulated waveform of faulted memory test.

리에서 동작하면서 발생하는 파형의 일부이다. 그림 6의 (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 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 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 신호로서 동작 순서는 다음과 같다.

테스트 인가 신호 (2)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)이 '0'으로 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 6의 테스트 시간 205397ns를 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 7은 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 210540ns 전후에서 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 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 진행할 필요가 없으므로, End_of_BIST 신호 (4)를 발생시키고 모든 테스트를 중단하게 된다.

Xilinx ISE 8.1i를 사용하여 제안하는 메모리 테스트 구조를 구현하여 보았다. 테스트 할 메모리의 크기를 1KByte, 64KByte, 256KByte로 정하였고, 각 메모리는

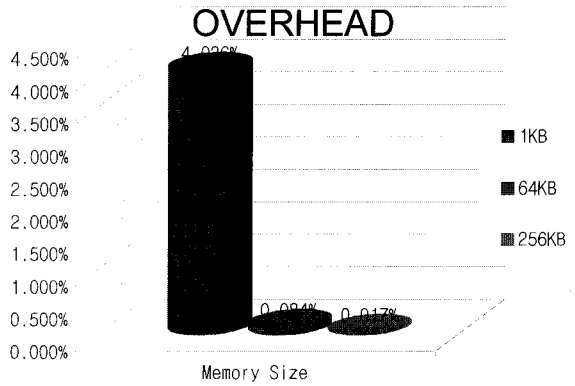


그림 8. 제안하는 메모리 테스트의 메모리 크기별 오버헤드

Fig. 8. Overhead of proposed memory test for each memory size.

표 6. Xilinx ISE 8.1i로 구현한 결과 값
Table 6. 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

32bit 워드를 가지며 8, 14, 16 bit 크기의 주소를 가진다. 각 메모리 크기에 따른 테스트 회로의 오버헤드를 그림 8에서 비율로 표시하였다.

그림 8을 보면 메모리의 크기가 1KByte인 메모리를 테스트하는 메모리 테스트 회로의 오버헤드는 4.036%이고, 64KByte 메모리를 테스트하는 회로의 오버헤드는 0.084%, 256KByte 메모리의 경우 0.017%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 6은 Xilinx ISE 8.1i를 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[10]. 그리고 마지막은 제안하는 프로그램 가능한 메모리 내장 자체 테스트 구조의 결과 값이다.

제안하는 메모리 테스트는 기존의 프로그램 가능한 내장 자체 테스트[10]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 6과 같이 제안하는 테스트 회로는 기존의 테스트 보다 더 많은 고장을 검출하지만, 그 시간은 증가하지 않고 오히려 줄어든 것을 알 수 있다. 지원하는 알고리즘이 많음에 따라 테스트 회로의 크기가 증가하였지만, 회로의 면적 증가 오버헤드는 그림 8에서 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.

메모리가 하나의 칩 안에 내장됨에 따라 외부에서 내장된 메모리에 대한 접근성이 떨어져 내장된 자체 테스트가 필요하게 되었다. 그리고 하나의 알고리즘을 사용하는 테스트는 메모리 생산 과정의 반복함으로써 얻어지는 수율 변화에 따른 알고리즘 변경이 불가능 하여 알고리즘 변경이 필요할 때마다 새롭게 설계를 하여야 했다. 그리하여 메모리 테스트를 수행하는 과정에서 시간과 비용이 증가하게 되었다. 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 전체 메모리 크기와 비교하였을 경우, 매우 작은 면적 오버헤드를 가지고 있다. 또한 기존의 내장 자체 테스트[10]보다 지원 가능한 알고리즘의 개수가 더 많고, 더 빠른 속도로 동작이 가능하다. 결과적으로 제안하는 테스트 구조를 사용하면 테스트 시간과 비용을 줄일 수 있을 것으로 예상한다.

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.

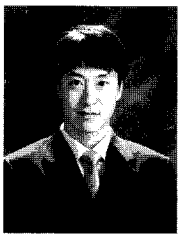
[2] V. G. Mikitjuk, V. N. Yarmolik and A. J. van de Goor, "RAM testing algorithms for detecting multiple linked faults," *in Proc. European Design and Test Conf.*, pp. 435-439, 1996.

[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] 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.
[9] 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
[10] 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>

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>