論文99-36C-12-6

효율적인 캐쉬 테스트 알고리듬 및 BIST 구조

(An Effective Cache Test Algorithm and BIST Architecture)

金 弘 植 *, 尹 度 鉉 *, 姜 成 昊 *

(Hong-Sik Kim, Do-Hyun Yoon, and Singho Kang)

요 약

급속한 프로세서 성능 향상에 따라 메인 메모리와의 속도차이를 극복하기 위해서 캐쉬메모리의 사용이 일반화되었다. 일반적으로 내장된 캐쉬 블록의 메모리는 그 크기가 작기 때문에 테스트 관점에서 테스트 시간보다는 고장 검출률이 중요하다. 따라서 본 논문에서는 다양한 고장 모델을 테스트할 수 있는 테스트 알고리듬과 상대적으로 적은 오버헤드를 갖는 새로운 BIST(Built-In Self Test) 구조를 제안하였다. 새로운 동시 테스트 BIST 구조에서는 캐쉬 제어 블록의 비교기를 태그 메모리 결과분석기로 사용한다. 이를 위한 비교기의 선행 테스트를 위해 변형된 주사사슬을 사용하여 테스트 클록을 감소하였다. 및 개의 경계주사 명령어를 추가하여 내부 테스트 회로들을 제어할 수 있다. 새로운 메모리 테스트 알고리듬은 12N의 복잡도를 갖고 SAFs, AFs, TFs linked with CFs, CFins, CFids, SCFs, CFdyns 및 DRFs의 고장을 테스트할 수 있으며, 새로운 BIST 구조는 합성결과 기존의 동시 테스트 방법보다 약 11%의 오버헤드 감소가 가능하였다.

Abstract

As the performance of processors improves, cache memories are used to overcome the difference of speed between processors and main memories. Generally cache memories are embedded and small sizes, fault coverage is a more important factor than test time in testing point of view. A new test algorithm and a new BIST architecture are developed to detect various fault models with a relatively small overhead. The new concurrent BIST architecture uses the comparator of cache management blocks as response analyzers for tag memories. A modified scan-chain is used for pre-testing of comparators which can reduce test clock cycles. In addition several boundary scan instructions are provided to control the internal test circuitries. The results show that the new algorithm can detect SAFs, AFs, TFs linked with CFs, CFins, CFids, SCFs, CFdyns and DRFs models with O(12N), where N is the memory size and the new BIST architecture has lower overhead than traditional architecture by about 11%.

* 正會員, 延世大學校 電氣 및 컴퓨터 工學科

(Department of Electrical and Computer Engineering, Yonsei University)

※ 이 논문은 1998년 한국 학술 진흥 재단의 학술연구비에 의하여 지원되었습니다.

接受日字:1999年8月28日, 수정완료일:1999年11月10日

I. 서 론

컴퓨터 시스템은 프로세서와 메인 메모리의 처리 속도의 불일치로 인해 불편을 겪어왔다. 이러한 속도의 불일치는 메인프레임이나 워크스테이션 등의 고성능 시스템에서 매우 중요한 문제가 되었다. 그 결과 이러한 고속 동작을 요하는 시스템은 SRAM 캐쉬(cache)를 사용하게 되었다. SRAM 캐쉬는 프로세서와 DRAM

사이에서 비싸지만 고속의 동작을 할 수 있는 적은 용량의 메모리로 프로세서와 메인 메모리 간 처리 속도의 차이를 최소화하였다. 그런데 캐쉬 메모리의 테스트는 순수한 메모리의 테스트와는 다른 양상을 갖고 있다. 캐쉬 메모리를 구성하고 있는 SRAM과 캐쉬 제어 회로 및 태그 RAM을 모두 테스트해야 하는 어려움을 갖고 있다. 게다가 프로세서 내에 내장되어 있는 캐쉬를 테스트하기 위해서는 프로세서와 캐쉬를 분리시키는 작업이 필요하게 된다. 이렇게 복잡한 구조를 갖고 있는 캐쉬를 테스트하기 위해서는 설계 단계에서부터 테스트를 고려하는 DFT(design for testability) 기법이 사용되어야 한다. DFT 기법은 테스트 대상인 회로의 특성에 맞게 여러 가지 방법을 사용할 수 있으며 캐쉬의 경우 프로세서에 내장되어 사용되기 때문에 DFT 기법 중 하나의 내장된 자체 테스트 기법(BIST : Built-In Self Test)을 사용하여 효율적으로 테스트 할 수 있다. 내장된 자체 테스트 기법은 테스트 대상인 회로에 테스트하는 회로를 내장시키는 방법으로 테스트와 관련된 비용 및 시간을 크게 절감시킬 수 있으나 첨가된 회로 때문에 결국 전체 칩의 면적이 커지게 되므로 가장 작은 양의 회로를 첨가하여 쉽게 테스트를 할 수 있도록 하는 것이 중요하다.

일반적으로 캐쉬의 테스트는 분리된 영역의 테스트가 아니라 프로세서 테스트의 영역 내에 포함되어 이루어진다. 따라서 주로 내장 메모리에 대한 연구가 캐쉬 테스트를 위한 기반 연구로 진행되어 왔다. 일반적으로 캐쉬 내의 논리 회로들은 나머지 논리회로의 테스트의 연속선상에서 테스트되는 것이 바람직하고 거의 대부분의 마이크로 프로세서들의 테스트 방법이 그러하다[1-3]. 내장 메모리는 성능 향상에 도움이 되지만 반면에 테스트 용이도를 감소시키는 단점이 있다. 프로세스 기술이 향상함에 따라 보다 많은 회로가 칩 내에 내장이 가능하게 되었고, 따라서 외부로부터 내장 메모리를 접근하는 것이 더욱 어려워 졌다. 따라서 제안되는 내장 메모리 테스트 방법이 BIST 기법이다[4-6].

메모리 BIST는 기본적으로 메모리 어레이에 미리 정해진 테스트 패턴을 가하기 위해 설계된 FSM 로직을 갖는다. 이러한 FSM 블록은 특정한 알고리듬을 기반으로 하여 설계되기 때문에 다른 종류의 테스트 패턴을 가할 수는 없다. 테스트 패턴은 일반적으로 data-in 레지스터의 '0' 또는 '1' 값을 테스트 어드레스 집합 내에서 읽기/쓰기 동작을 반복하면서 메모리 어레이에 가해진다. 메모리 어레이로부터의 출력은 이러한 data-in 레지스터의 값과 비교되어 pass/fail을 출력한다. 이러한 일련의 동작을 수행하는 BIST 블록은 이밖에도 테스트를 위해 칩을 초기화하거나 특정 상태로 세팅하는 등의 동작을 수행하기도 한다[7,8].

내장된 메모리의 성능 저하를 최소화하기 위해 BIST가 적용된 캐쉬 메모리 주위와 주사 가능한 마지막 주사 집합들 사이에 논리 회로 블록을 설계하기도 한다[1]. 이 논리 회로를 테스트하기 위해서는 메모리 코어 주위에 바이패스 경로를 삽입하거나, 주사만 가능한 래치를 추가하는 DFT 기법을 적용한다[9]. 또는 메모리의 테스트를 수행한 뒤 메모리에 테스트 패턴을 저장하여 이것을 이용하여 내장된 메모리의 주변 회로를 테스트하는 방법이 연구되기도 하였다[10]. 이 때 메모리에 특정 값을 저장하기 위해서 메모리 BIST 회로를 사용하는 데, 이것은 주변회로의 테스트가 아직 이루어지지 않은 상태이기 때문에 주변로직을 가로질러 패턴을 가하는 것의 안전성이 보장되지 않기 때문이다. 이를 위해서는 BIST 회로의 어드레스 생성부 및 데이터 생성부를 외부에서 제어할 수 있도록 설계해야 한다.

BIST 기법 외에 내장 메모리의 입출력 단을 외부로 인터페이스 하는 방법이 연구되기도 하였다. Super SPARC의 경우 SRMTST라는 입력 핀을 갖는데, 이 핀에 의해 칩이 SRAM 테스트 모드로 진입하게 된다. 이 모드에서 칩은 마치 8 비트 워드의 메모리처럼 동작하게 된다. 따라서 일반적인 메모리 테스터를 사용하여 내장 메모리를 테스트할 수 있다[11]. 그러나 이러한 방식은 내장 메모리 수와 크기가 커지면 입출력단자 제한 때문에 한계가 있다. UltraSPARC은 메모리 BIST 기법을 사용하여 내장 메모리를 테스트한다[12].

특정 알고리듬에 따라 일단 설계된 BIST 회로는 다른 종류의 테스트 패턴은 가할 수 없다. 따라서 고안된 BIST 기법은 프로그램이 가능한 BIST 기법이다[13-14]. 프로그램이 가능한 BIST는 마이크로 프로세서와 유사하게 동작한다. 메모리 BIST 프로그램이 보통의 마이크로 코드 메모리 어레이에 주사되고 각각의 명령어가 디코드 되고 수행되어 메모리 BIST 마이크로프로세서에 의해 어레이에 가해진다. 다양한 테스트 알고리듬을 하나의 BIST 블록에 의해 적용할 수 있다는 장점이 있다.

일반적으로 내장 메모리들은 각각 별개의 테스트 대상으로 간주되어 BIST 회로를 사용하여 단계적으로 테스트한다. 그러나 내장 메모리 테스트 시간 및 하드웨

어 오버헤드를 줄이기 위해 내장 메모리를 하나의 BIST 블록에 의해 테스트하는 동시 테스트 방법이 연구되었다. 내장된 메모리 블록들은 적용되는 테스트 알고리듬이 동일하기 때문에 하나의 메모리 BIST 상태 머신에 의해 완전 테스트가 가능하다. 이러한 원리는 각각의 메모리에 별도의 BIST 회로를 부착하는 것보다 하드웨어 오버헤드가 크게 줄어든다[10,15]. 동시 테스트를 위해 내장된 메모리들은 공통된 테스트 데이터/어드레스, 비교 데이터 라인을 갖고, 각각의 메모리들은 비교기를 따로 내장한다. 또한 write-enable(WE) 신호를 따로 갖고 load-result (LR) 신호를 갖는다[10]. WE 신호를 사용하여 내장 메모리들에 테스트 데이터를 쓰고 LR 신호를 사용하여 개별적으로 비교 데이터와 결과분석을 수행한다.

본 논문에서는 전체 테스트 시간과 BIST 하드웨어 오버헤드를 줄이기 위한 동시 테스트 기법을 개발하였다. 기존의 방법과는 다르게 캐쉬 내부의 비교기를 이용하여 결과 분석을 시도하는 구조를 이용한다. 이러한 방법은 이러한 BIST 구조 구현을 위해서는 비교기의 선행 테스트가 필수적인데 이것은 경계주사를 통해 이루어진다. 경계 주사를 이용하여 비교기를 테스트하기 위해서는 추가 명령어가 필요하고 또한 주사사슬의 변경 및 멀티플렉싱 회로의 사용이 필요하다.

II. 비교기 테스트 방법

일반적인 조합 회로들은 LFSR에 의해 제공되는 의사 무작위 테스트에 의해 효율적으로 테스트가 가능하다[16]. 그러나 대부분의 비교기들은 무작위 패턴 저항고장(random pattern resistant faults)을 갖고 있어 LFSR로 테스트하기는 매우 어렵다. 따라서 결정 테스트 패턴을 가해서 테스트 해야한다. 그런데 일반적으로 프로세서의 로직 부분을 테스트하기 위해 무작위 패턴을 사용한다. 따라서 비교기와 나머지 논리 회로들은 따로 분리하여 테스트한다. 이것의 제어는 주사기법을 통해 이뤄진다.

표 1은 w 비트 비교기를 테스트하기 위한 테스트 패턴이다. 이와 같은 테스트는 경계주사를 통해 이루어진다. w 비트 비교기를 일반 주사 사슬을 사용해서 테스트한다면, 총 2(w+1) 개의 패턴이 필요하다. 한 패턴당 쉬프트를 위한 2w 클록과 패턴을 가하기 위한 1 클록이 필요하고 마지막 결과를 주사해서 빼내기 의해 1 클록이 필요하다. 따라서 전체 테스트 클록은 $2(w+1) \times (2w+1) + 1 = 4w^2 + 6w + 3$ 클록이 필요하다. 그림 1은 본 논문에서 제안하는 변형된 경계주사 구조이다. 기본적으로 중요한 전제는 원하는 경계주사 사슬만을 Shift-DR 상태로 제어가능 해야한다는 것이다. 이것은 표 1의 테스트 패턴을 보면 알 수 있다. 테스트 패턴들은 비교기의 한쪽 입력(예를 들면 비교기 입력 0)만을 쉬프트 시켜주면 되기 때문에 경계주사 사슬을 그림처럼 멀티플렉서를 사용하여 변형시킨 뒤 추가 경계주사

표 1. w 비트 비교기를 위한 테스트 패턴
Table 1. Test pattern set for w bit comparator.

비교기 입력 0	비교기 입력 1	Valid	Hit/Miss
0000……0	0000……0	0	1
0000……0	0000……0	1	0
0000……0	1000……0	0	0
0000……0	0100……0	0	0
0000……0	0010……0	0	0
0000……0	……	0	0
0000……0	0000……1	0	0
1111……1	1111……1	0	1
1111……1	0111……1	0	0
1111……1	1011……1	0	0
1111……1	1101……1	0	0
1111……1	……	0	0
1111……1	1111……0	0	0

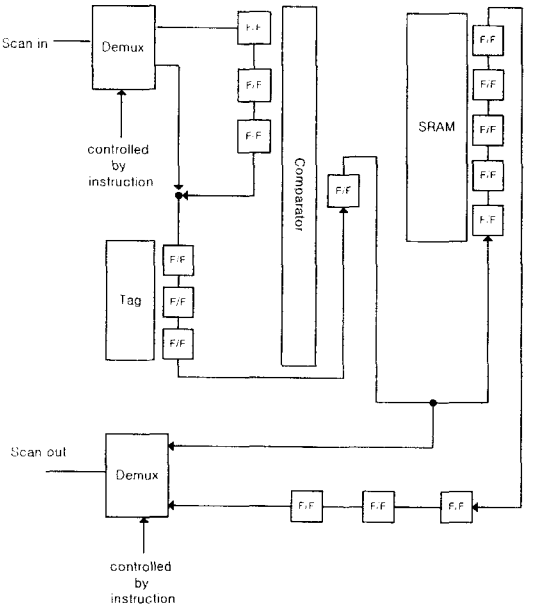

그림 1. 멀티플렉싱 회로의 사용
Fig. 1. Usage of multiplexing elements.

명령어를 이용해서 제어함으로써 테스트 패턴을 가하는 시간을 크게 단축시킬 수 있다. 본 논문에서 제안하는 멀티플렉싱 기법을 사용하면 전체 사슬에 0 또는 1을 주사하기 위한 2개의 패턴만 2w의 쉬프트가 필요하고 나머지 패턴들은 w 번의 쉬프트가 필요하다. 따라서 $2\times(2w + 1) + 2w\times(w+1) + 1 = 2w^2 + 4w + 2$ 클록이면 충분하다. 따라서 테스트 시간을 상당히 감소시킬 수 있다.

Ⅲ. 제안된 메모리 테스트 알고리듬

캐쉬 메모리는 일반 메모리에 비해 작은 크기를 갖는다. 메모리 테스트 시간은 메모리 크기의 함수이기 때문에 캐쉬의 테스트 시간은 그다지 크지 않다. 따라서 캐쉬의 테스트에서 테스트 알고리듬에서 중요한 것은 고장 검출률이라 할 수 있다. 즉, 가능한 다양한 고장 모델을 고려하여 신뢰성 있는 테스트를 수행하는 것이 중요하다고 할 수 있다. 그림 2는 제안된 알고리듬이다.

$$\underset{M_0}{\uparrow(w0)};\ \underset{M_1}{\uparrow(r0;w1;r1)};\ \underset{M_2}{\uparrow(r1;w0;)};\ \underset{M_3}{\downarrow(r0;w1)};\ \underset{M_4}{\downarrow(r1;w0;r0)};\ \underset{M_5}{(no\ operation);\ \downarrow(r0)}$$

그림 2. 새로운 알고리듬
Fig. 2. New algorithm.

메모리 크기를 N이라 했을 때, 새로운 알고리듬은 12N의 복잡도를 갖으며, AFs, SAFs, TFs, CFins, CFids, CFdyns, TFs linked with CFids 및 데이터 보존 고장을 추가로 테스트할 수 있다.

1. 새로운 알고리듬으로 테스트 가능한 고장

새로운 알고리듬은 조건 $\uparrow(r\,x,\cdots,w\,\bar{x});\ \uparrow(r\,\bar{x},\cdots,w\,x)$을 M_1, M_4와 M_2, M_3에서 만족하기 때문에 모든 AFs를 테스트할 수 있다. 그리고 모든 셀들이 0, 1 상태에 있기 때문에 SAFs를 테스트할 수 있다. 또한 나머지 고장 모델들은 다음과 같이 테스트 가능하다.

1) 동행 결합고장(CFids)

동행결합 고장에 대한 테스트는 결합셀이 피결합 셀보다 어드레스가 낮은 경우와 반대의 두 가지 경우로 나누어 생각한다. 피결합 셀은 C_i로 표기하고 결합셀은 C_j로 표기한다. 여기서 i와 j는 각각 어드레스를 의미한다.

1. C_i가 자신보다 낮은 어드레스의 셀에 의해 결합되었고 C_j가 그러한 셀들 중 가장 어드레스가 높은 셀이라고 하자. CFids의 네 가지 경우에 대해서 증명이 이루어져야 한다.

(a) C_i가 C_j에 <↑; 0> 결합되었다고 하면, 그 고장은 M3과 M4에서 검출이 가능하다.

표 2. <↑; 0> 동행 결합 고장 검출 과정
Table 2. Detection process of <↑; 0> CFid.

C_j에의 M_3 동작	C_j	C_i	C_i의 고장
r0	0	1	-
w1	1	0	<↑; 0> CFid
C_j에의 M_3 동작	Cj	Ci	-
r1	1	0	-

M_3에서 C_j에 1을 쓸 때 <↑; 0> 결합고장에 의해 C_i는 0을 갖게 된다. 그리고 M_4에서 C_i에 대한 읽기 동작으로 1 대신 0이 읽히기 때문에 고장을 알 수 있다.

(b) C_i가 C_j에 <↑; 1> 결합되었다고 하면 M_1에서 검출이 가능하다.

표 3. <↑; 1> 동행 결합 고장 검출 과정
Table 3. Detection process of <↑; 1> CFid.

C_j에의 M_1 동작	C_j	C_i	C_i의 고장
r0	0	0	-
w1	1	1	<↑; 1> CFid
C_j에의 M_1 동작	C_j	C_i	-
r0	1	1	-

우선 C_j에서 0이 읽혀지고 나서 1이 쓰여진다. 이 때 <↑; 1> 결합고장에 의해 C_i에 1이 써지게 된다. M_1의 C_i에 대한 읽기 동작에서 0 대신에 1일 읽히기 때문에 고장을 알 수 있다.

(c) C_i가 C_j에 <↓; 0> 결합되었다면 위와 유사한 방법으로 M_2에서 고장을 검출함을 알 수 있다.

(d) C_i가 C_j에 <↓; 1> 결합되었다면 위와 유사한 방법으로 M_4와 M_5에 의해 고장을 검출함을 알 수 있다.

2. C_i가 자신보다 높은 어드레스의 셀에 의해 결합되었고 C_j가 그러한 셀들 중 가장 낮은 어드레스를 갖는다고 하자. 그 테스트 과정은 1과 유사한데, M_1을 M_3으로, M_2를 M_4로, M_3을 M_1로, M_4를 M_2로 그리고 M_5

를 M_3으로 바꾸어 주기만 하면 된다.

2) 반전 결합고장(CFins)

본 알고리듬은 March C-의 변형된 형태를 갖는다. 따라서 반전 결합고장을 테스트할 수 있지만 연계된 경우의 반전 결합고장(Linked CFins)은 테스트할 수 없다. 연계되지 않은 경우의 CFins는 다음과 같은 방식으로 테스트 가능하다. CFins도 결합고장의 일종이기 때문에 CFids와 같이 두 가지 경우로 나누어 생각한다.

1. C_i가 자신보다 낮은 어드레스의 셀에 의해 결합되었고 그 셀들 중 가장 높은 어드레스를 갖는 셀을 C_j라고 하자.

(a) C_i가 C_j에 <↑; ↕> 결합되었다고 하면 M_1 또는 M_3과 M_4에 의해 테스트가 가능하다.

표 4. <↑; ↕> 반전 결합고장 검출 과정
Table 4. Detection process of <↑; ↕> CFin.

C_j에의 M_3 동작	C_j	C_i	C_i의 고장
r0	0	1	-
w1	1	0	<↑; ↕> CFin
C_j에의 M_4 동작	C_j	Ci	-
r1	1	1	-

M_3의 C_j에 대한 전이동작은 CFin에 의해 C_i의 값을 반전시키게 된다. 그리고 M_4의 C_j에 대한 읽기 동작에서 1 대신에 0을 읽게 됨으로 고장을 알 수 있다.

(b) C_i가 C_j에 <↓; ↕> 결합되었다면 위와 유사한 방법으로 M_2 또는 M_4와 M_5에 의해 검출할 수 있음을 알 수 있다.

2. C_i가 자신보다 높은 어드레스를 갖는 C_j와 반전 결합되었을 경우도 테스트 가능하며 그 과정은 위와 유사하다.

3) 동적 결합고장(CFdyn)

동적 결합고장은 다음과 같이 4가지 경우로 나뉜다. <r0|w0;0>, <r0|w0;1>, <r1|w1;0> 그리고 <r1|w1;1>이다. 여기서 '|'는 결합셀에 가해질 수 있는 읽기 동작과 쓰기 동작의 OR 동작을 의미한다. 표 5는 새 알고리듬의 M_1에서 M_4 단계까지의 C_i, C_j 셀에 대한 동작 및 상태 변화를 표시한다. 여기서 C_i의 상태가 x_i이고 C_j의 상태가 y_j라면 그 상태를 Sx_iy_j로 표시한다. 표를 보면 위의 4 가지 경우의 CFdyns 조건을 모두 만족시키는 것을 확인할 수 있다.

4) 상태 결합고장(SCFs)

모든 SCFs는 Sx_iy_j가 {(00), (01), (10), (11)}의 4가지 경우를 만족하면 검출 가능하다[4]. 제안된 알고리듬에 의한 메모리 셀들의 상태를 표시한 표 5에 의하면 두 셀은 가능한 모든 상태에 다다름을 알 수 있다. 따라서 모든 SCFs가 테스트가 가능하다.

표 5. March 요소 중의 셀 상태 변화도
Table 5. Cell state transition during March elements.

March 요소	동작 전 상태	동작	동작 후 상태
M0	-	w0 into i	-
	-	w0 into j	S00
M1	S00	r0 from i	S00
	S00	w1 into i	S10
	S10	r1 from i	S10
	S10	r0 from j	S10
	S10	w1 into j	S11
	S11	r1 from j	S11
M2	S11	r1 from i	S11
	S11	w0 into i	S01
	S01	r1 from j	S01
	S01	w0 into j	S00
M3	S00	r0 from j	S00
	S00	w1 into j	S01
	S01	r0 from i	S01
	S01	w1 into i	S11
M4	S11	r1 from j	S11
	S11	w0 into j	S10
	S10	r0 from i	S10
	S10	r1 from i	S10
	S10	w0 into I	S00
	S00	r0 from j	S00

5) 결합고장과 연계된 전이 고장(TFs linked with CFs)

상승 전이 고장은 M_1에서 검출 가능하다. 왜냐하면 고장이 w_1에 의해 자극되고 r_1에 의해 검출되기 때문이다. 그리고 쓰기와 읽기 동작 사이에 다른 셀에 대한 동작이 없기 때문에 CF에 의해 고장 마스크 되는 경우는 없다. 마찬가지로 하강 전이 고장은 M_4에서 검출 가능하고 위와 같은 이유로 고장이 마스크가 되지 않

는다.

6) 데이터 보존 고장(DRF)

M_4와 M_5 사이에 적당한 no operation 간격을 두어 M_4에서 저장한 값이 일정 시간 후에도 보존되는지를 확인함으로써 DRF의 테스트가 가능하다. 여기서 no operation 상태는 BIST 회로에서 생성하도록 설계하였다.

2. 제안된 알고리듬의 효율성

표 6은 크기 N의 메모리에 대한 여러 가지 March 알고리듬들의 복잡도 및 테스트 가능한 고장 모델들을 설명한다[19]. 이와 비슷한 복잡도를 갖는 알고리듬들로는 March Y, March C-, March 1/0, March A 등이 있는데, 제안된 알고리듬은 이들 알고리듬에 비해 많은 고장 모델을 테스트 할 수 있다.

표 6. March 알고리듬과의 비교
Table 6. Comparison with March algorithms.

March 알고리듬	복잡도	테스트 가능한 고장 모델
Marching 1/0	14N	AFs, SAFs, TFs
MATS++	6N	AFs, SAFs, TFs
March X	6N	AFs, SAFs, TFs, CFins
March C-	10N	AFs, SAFs, TFs, CFins, CFids, CFdyn, SCFs
March A	15N	AFs, SAFs, TFs, CFins, CFdyn, SCFs, Linked CFids
March Y	8N	AFs, SAFs, TFs linked with CFins
March B	17N	AFs, SAFs, TFs, CFins, CFdyn, SCFs, Linked CFids, TFs linked with CFids
새로운 알고리듬	12N	AFs, SAFs, TFs, CFins, CFids, CFdyn, SCFs, DRFs, TFs linked with CFs

Ⅳ. 메모리 BIST구조 및 테스트 입출력

1. 메모리 BIST구조

메모리 BIST 회로는 패턴 생성기, 테스트 상태 제어기로 구성된다. 태그 메모리와 데이터 메모리는 같은 크기의 어드레스를 갖기 때문에 테스트 어드레스를 공유한다. 그림 3은 2 way set associative mapped 캐쉬에 대한 전체 메모리 BIST 구조도이다. 데이터 메모리는 2개의 블록으로 구성되기 때문에 결과분석기를 양 블록의 값을 비교하는 방식으로 설계하였다. 여기서 데이터 메모리 한 블록은 m×2n 의 크기의 메모리로, 태그 메모리 한 블록은 w×2n의 크기의 메모리로 구성된다. 블록을 동시에 테스트하기 위해 결과분석기를 BIST회로 내에 두지 않고 메모리 블록 내부에 내장시켜서 버스의 사용을 하지 않도록 했다. 즉 데이터 메모리 출력값을 상호 비교하여 에러를 판별한다. 메모리 상호 비교기법은 결과 분석 단계를 단순화시킴으로써 회로를 단순화시킬 수 있다[20].

테스트 어드레스는 테스트 어드레스/데이터 생성기에서 생성한다. 어드레스 생성기의 레지스터들은 주사사슬을 구성해서 에러가 발생하는 순간의 어드레스를 저장하고 있다. 쉬프트 신호가 들어오면 출력해준다. 이렇게 주사된 어드레스는 redundancy를 이용한 고장 교정에 사용된다. 어드레스 생성은 March 상태에 따라 증가/감소를 반복하여 이루어진다. 그리고 테스트 어드레스/데이터 생성기는 테스트 상태 제어기에 의해 제어되어 적절한 테스트 데이터를 메모리에 가한다. 이 때 태그 메모리는 데이터 메모리 테스트 데이터의 하위 w 비트를 테스트 데이터로 필요로 한다. 테스트 상태 제어기는 적절한 March 상태를 결정해서 각각의 테스트 생성기에 보내준다. 또한 테스트 상태 제어기는 태그 메모리로부터 온 hit/miss 신호가 1이거나 데이터 메모리에 에러가 있다고 분석하게 되면 바로 어드레스 생성기와 데이터 생성기의 동작을 중지하고 에러 신호를 출력한다.

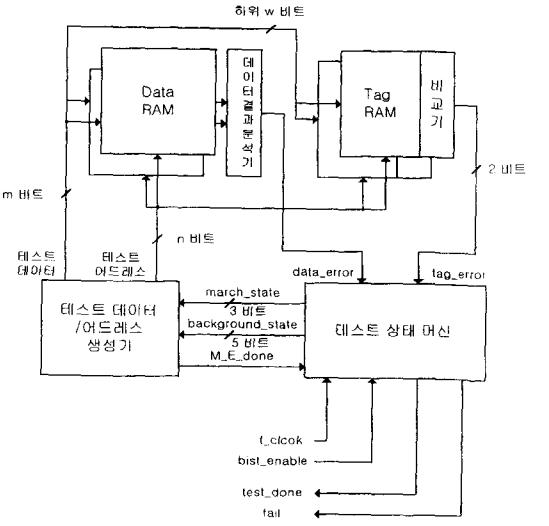

그림 3. BIST 구조도
Fig. 3. BIST architecture.

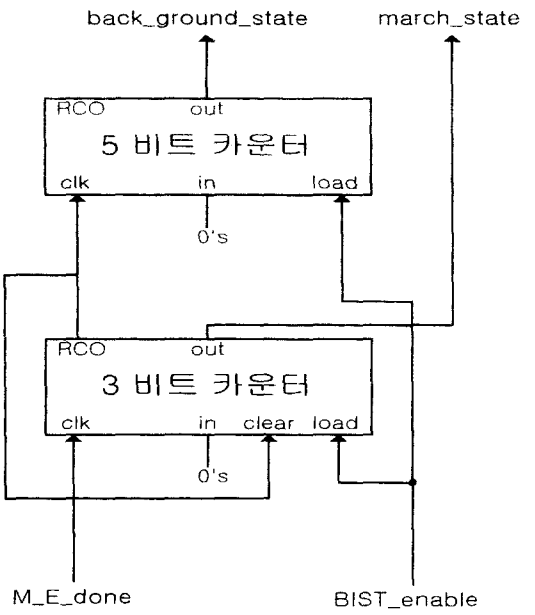

그림 4. March 상태 머신의 개념도
Fig. 4. Block diagram of March state machine.

그림 4는 테스트 상태 머신의 개념도이다. 비트 카운터는 March element의 상태를 알려주기 위한 신호를 생성하고 5비트 카운터는 background 데이터의 상태를 알려주는 신호를 생성하기 위한 회로들이다. BIST_enable 신호가 입력되면 3 비트 카운터와 5 비트는 작동을 시작한다. March 상태는 테스트 어드레스/데이터 생성기가 하나의 March element를 완료하면 카운트되어 다음 March 상태를 가리켜야한다. 따라서 M_E_done 신호를 받아 카운트된다.

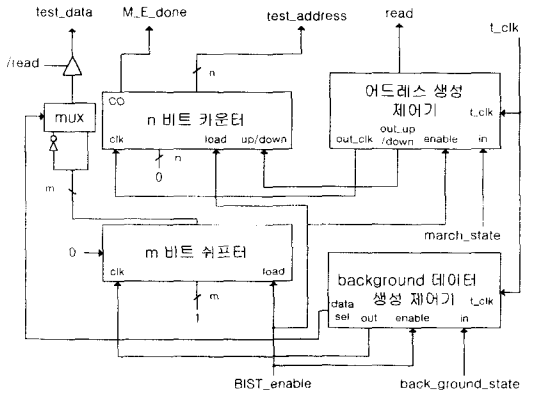

그림 5. 테스트 어드레스/데이터 생성기 개념도
Fig. 5. Block diagram of test address/data generator.

그림 5는 테스트 데이터/어드레스 생성기의 개념도이다. 테스트 데이터는 32개의 background 데이터로 구성된다. background 데이터 생성 제어기는 background_state 신호로부터 m 비트 쉬프터를 제어하여 적절한 상태에서 적절한 테스트 데이터를 생성한다. 테스트 어드레스는 March element에 따라 증가/감소한다. 따라서 어드레스 생성제어기에서 march_state 신호에 따라 n 비트 카운터의 증감의 조절과 카운팅이 이루어진다. 또한 어드레스 생성 제어기에서 읽기/쓰기 신호인 read 신호를 발생시킨다.

2. 전체 테스트 입출력

본 논문의 가장 중요한 특징은 메모리 동시 테스트이다. 이를 위해 태그 메모리의 결과 분석기를 기존의 비교기를 사용하여 구현한다. 이를 위해 비교기의 테스트가 전제가 되는데, 이는 앞 절에서 설명했듯이 결정 테스트 패턴을 사용해야한다. 결정 테스트 패턴을 비교기 부분에만 가하기 위해 경계 주사의 구조가 그림 6와 같이 설계되어야 하고 앞 절에서 설명한 테스트 시퀀스가 가해지도록 mux, demux 및 스위치를 경계 주사 명령어에 의해 제어해야 한다. 이를 위해 기존의 경계 주사 명령어 외에 약간의 추가 명령어가 있어야 한다. 전체적으로 멀티플렉싱 회로는 3가지 동작을 필요로 한다. 따라서 2 비트의 제어 비트를 필요로 한다. IEEE 1149.1에 의하면 BYPASS 명령어는 모든 비트가 1이 되어야 한다. 따라서 2 비트의 제어 비트를 '11'을 제외한 값으로 설정한다. 표 7은 멀티플렉싱 회로의 동작 및 추가 명령어의 인코딩을 설명한다. 일반 주사 동작에서는 일반 논리 회로의 테스트를 위한 주사사슬의 연결을 해준다. 이 때는 보통의 연결이 이루어진다. 캐쉬의 비교기를 테스트하기 위해서는 두 단계가 필요하다. 비교기의 모든 입력단에 1(또는 0)을 채우는 것과 한쪽 입력에 0(또는 1)을 이동시키면서 테스트하는 것이다. 이를 위해 멀티플렉싱 회로는 표와 같이 두 단계로 나뉘어 동작한다. 즉 모든 입력에 0/1을 주사하기 위한 조합과 walking on 1/0을 하기 위한 동작이다. 이에 관한 구체적인 멀티플렉싱 회로의 동작은 표 7과 같다.

앞서 설명하였듯이 멀티플렉싱 회로를 구동하기 위해서는 IEEE 1149.1 표준의 강제/선택 명령어 집합 이외에 그림 7와 같은 추가 명령어가 필요하다. BYPASS 명령어는 모든 비트가 '1'로 인코딩 되어야하고, 일반 주사명령어 및 0/1 주사 명령어와 walking on 0/1 명령어는 표 7에 따라 인코딩 된다. 나머지 SAMPLE/

PRELOAD 및 RUN_BIST 명령어의 인코딩이 그림 7에 나타나 있다. 이와 같은 명령어 집합을 이용하여 캐쉬 블록을 테스트한다. 그림 8의 테스트 플로우는 전체 테스트를 위한 경계주사 상태 흐름을 나타낸다.

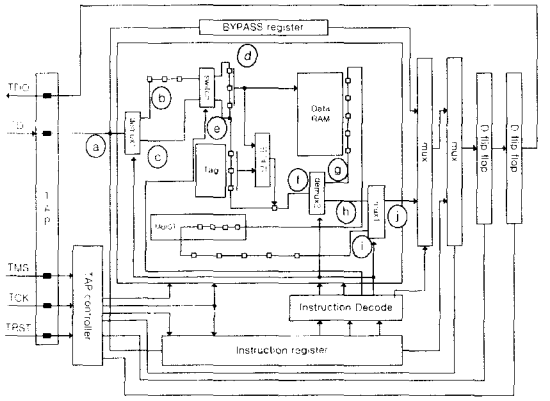

그림 6. 경계주사 구조도
Fig. 6. Boundary scan architecture.

표 7. 멀티플렉싱 회로의 동작
Table 7. Operations of the multiplexing circuits.

		멀티플렉싱 회로 동작		명령어 인코딩		
일반 주사 동작		demux	ⓐ→ⓑ	n-2 비트	0	01
		switch	ⓑ→ⓓ			
		demux	ⓕ→ⓗ	n-1 비트	1	
		mux	ⓗ→ⓙ			
비교기 테스트	0/1 주사	demux	ⓐ→ⓒ	n-2 비트	0	00
		switch	ⓒ→ⓓ			
		demux	ⓕ→ⓗ	n-1 비트	0	
		mux	ⓗ→ⓙ			
	walking on 1/0	demux	ⓐ→ⓒ	n-2 비트	1	10
		switch	ⓒ→ⓔ			
		demux	ⓕ→ⓗ	n-1 비트	0	
		mux	ⓗ→ⓙ			

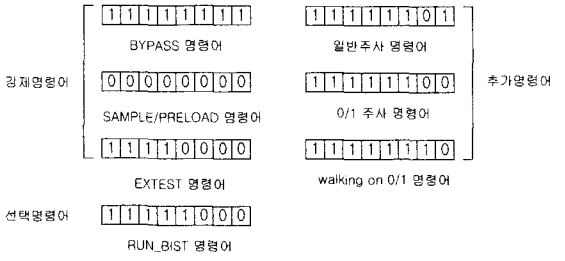

그림 7. 캐쉬 테스트를 위한 명령어 집합
Fig. 7. Instruction set to test cache.

전체 테스트의 시작은 일반 로직 회로를 테스트하기 위한 일반 주사 명령어의 로드와 디코드로 시작한다. 이 단계에서 일반로직은 메모리 BIST 회로를 포함한다. 메모리 BIST 회로 및 모든 로직 회로들은 주사사슬을 적용하여 의사 무작위한 방법으로 테스트를 한다. 이 때 비교기 부분의 테스트도 함께 이루어진다. 그러나 앞 절에서 언급하였듯이 비교기는 그 자체가 무작위 저항 고장을 갖기 때문에 완전 테스트가 불가능하다. 그렇다고 테스트 시간을 줄일 목적으로 비교기 부분을 제외하고 무작위 테스트를 수행하는 것은 하드웨어 복잡도를 증가시키는 단점이 있다. 따라서 본 논문에서는 비교기를 포함하여 무작위 테스트를 하고 비교기를 나중에 다시 테스트하는 방법을 제시한다. 일반 로직의 테스트가 완료되면 메모리 테스트를 수행해야 한다.

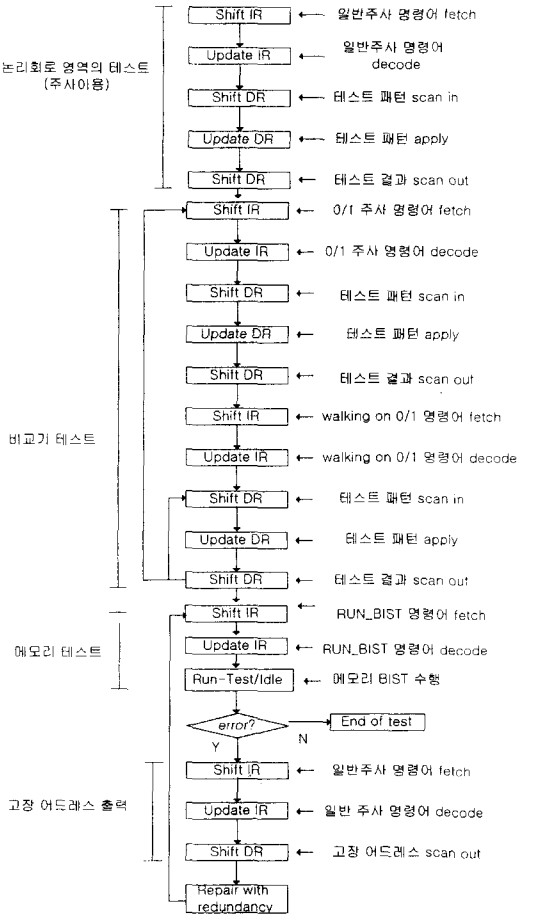

그림 8. 전체 테스트 플로우
Fig. 8. The total test flow.

일반 주사 명령어를 로드/디코드 하면 회로는 내부의 전체 주사사슬을 연결한다. Shift-DR, Update-DR, Shift-DR을 하여 의사 무작위 테스트 패턴으로 전체 로직 회로를 테스트하고 결과를 출력한다. 이 과정을 정해진 패턴 수만큼 반복한다. 모든 테스트 패턴이 가해지고 회로에 고장이 없음이 확인되면 비교기 테스트 단계로 넘어간다.

본 논문에서 제시하는 메모리 테스트의 가장 큰 특징은 동시 테스트이고 이것은 비교기의 선행 테스트가 전제되어야 한다. 따라서 비교기 테스트를 수행한 뒤 메모리 테스트를 수행한다. 비교기의 테스트는 우선 비교기 양단의 입력을 0으로 채우는 작업을 수행해야한다. 다음으로 한쪽 입력에 1을 이동시키면서 테스트를 수행한다. 이를 위해 walking on 1/0 명령어를 로드, 디코드하고 데이터 레지스터들을 Shift-DR, Update-DR한다. Shift-DR, Update-DR 동작은 w 비트 비교기의 경우, w회 반복한다. 다음으로 위의 과정을 입력 패턴을 반전하고 한번 반복하면 비교기에 대한 테스트가 완료된다. 비교기가 고장이 없음이 보장되면 메모리 테스트에 들어간다.

메모리 테스트는 앞서 언급하였듯이 메모리 BIST 회로가 전담한다. 메모리 BIST 회로를 구동하기 의해 RUN_BIST 명령어를 명령어 레지스터에 Shift-IR 하고 Update-IR 한다. 그러면 명령어 디코더가 메모리 BIST에 Enable 신호를 가하고 경계주사 TAP 제어기는 idle 상태에 들어간다. 메모리 BIST가 수행되는 동안 에러가 발생하면 그 에러 어드레스를 출력해야하는데, 이를 위해 데이터 레지스터를 일반주사 모드에 놓아야 한다. 즉 일반주사 명령어를 로드/업데이트하고 데이터 레지스터를 Shift-DR 해서 고장 어드레스를 외부로 출력한다. 만약 메모리가 고장이 없음이 확인되면 모든 테스트가 완료된다.

V. BIST 합성 결과

본 논문에서 제안한 메모리 BIST 구조는 VHDL로 설계하여 Synopsys로 합성하였다. 표 8은 회로의 오버헤드에 관한 표이다. BIST 회로 각각의 하위 블록에 관한 오버헤드 및 전체 캐쉬의 트랜지스터 수를 100으로 했을 때의 상대적인 오버헤드가 표시되어 있다. 여러 가지 크기의 캐쉬에 대한 BIST 회로들의 오버헤드가 계산되었다. 여기서 BIST 회로의 트랜지스터 수는 합성된 BIST 회로의 Synopsys 셀면적을 근거로 계산하였고, 캐쉬는 6 트랜지스터 SRAM 셀을 기준으로 해서 메모리 트랜지스터 수를 산출하였다. 메모리 BIST 회로는 테스트 대상 메모리의 크기에 상관없이 거의 일정한 하드웨어 오버헤드를 갖는 특징이 있다. 따라서 메모리 크기가 증가할수록 그 오버헤드는 줄어듦을 알 수 있다. 일반적으로 캐쉬 테스트에서 크기가 작은 것은 기능적 테스트에 의존하고 크기가 큰 메모리는 BIST를 사용한다. 표 8의 결과를 보면 메모리 크기가 증가함에 따라 BIST 회로의 상대적인 오버헤드는 감소함을 알 수 있다. 32K 이상의 메모리에 대해서는 0.5% 이하의 오버헤드를 보인다.

표 8. BIST 회로의 오버헤드
Table 8. BIST circuit overhead.

오버헤드 / 캐쉬 크기	BIST 회로 트랜지스터 수(%)				캐쉬 트랜지스터 수(%)	테스트 시간
	테스트 데이터/어드레스 생성기	테스트 상태 머신	결과 분석기	전체 BIST		
4K 바이트	3.43	1.27	0.20	4.90	100	15.85ms
8K 바이트	1.70	0.60	0.10	2.40	100	31.69ms
32K 바이트	0.35	0.13	0.02	0.50	100	127ms
128K 바이트	0.101	0.035	0.004	0.14	100	507ms
512K 바이트	0.022	0.007	0.001	0.03	100	2.03s
1M 바이트	0.014	0.005	0.001	0.02	100	4.06s

SRAM의 동작 속도를 80ns로 가정하였을 때, 메모리 테스트 시간은 메모리 크기의 함수이기 때문에 메모리 크기가 증가함에 따라 일정하게 증가함을 알 수 있다. 본 논문에서 제안하는 12N March 알고리듬 및 BIST 구조는 이 만큼의 테스트 시간으로 AFs, SAFs, CFins, CFdyns, CFids, SCF, DRFs 및 TF linked with CFs의 다양한 고장을 검출 할 수 있다. 또한 메모리 BIST의 삽입은 하드웨어 오버헤드 이외에 성능에도 영향을 미친다. 일반적으로 메모리 BIST의 경우 일반 I/O와 메모리 BIST를 멀티플렉싱하기 위한 mux가 필요하다. 메모리의 정상동작에 영향을 미칠 수 있는 BIST 회로는 이 mux 뿐이다. 따라서 이 부분에서 성능저하가 발생할 수 있다. 즉, BIST 출력부의 mux와 캐쉬 회로 내부의 주사 레지스터 및 mux에서 지연만큼의 성능저하가 발생한다.

표 9는 기존의 BIST 회로와의 비교 결과이다. 기존의 BIST구조는 MISR을 사용한 Ternullo의 방식으로 합성하였다[10]. 테스트 알고리듬으로는 본 논문에서 제안하는 12N 알고리듬을 사용하였다. 그리고 4K에서 1M까지의 다양한 메모리 사이즈에 대해서 메모리 BIST 회로를 합성하였다. 본 논문의 BIST 오버헤드와 Ternullo 방식의 오버헤드를 비교하였다. 여기서 오버헤드는 Synopsys 셀면적을 사용하였다. 그 결과 본 논문에서 제안하는 방법이 모든 경우에 대해서 약 11% 정도의 오버헤드가 감소되는 것을 알 수 있다. 기존의 방법은 모든 메모리들에 결과분석을 위해 MISR을 따로 부착해야 하는 데서 오버헤드 추가가 발생한다. 그리고 MISR을 사용하였을 경우 태그 메모리와 데이터 메모리가 각각 서로 다른 압축치를 갖기 때문에 동시 결과분석이 불가능하다. 따라서 결과 분석 단계가 2단계로 나뉘어서 이루어지는데, 이를 위한 오버헤드가 증가한다. 본 논문의 방식은 태그 메모리는 기존의 비교기를 사용하기 때문에 비교기 테스트를 위한 멀티플렉서가 추가되었고, 데이터 메모리 블록의 경우는 병렬 테스트 기법을 사용하여 하드웨어 복잡도를 감소하였다. 그리고 결과분석을 동시에 할 수 있기 때문에 이에 따른 하드웨어가 줄어든다

표 9. 다른 방식의 메모리 동시 테스트 BIST와의 오버헤드 비교

Table 9. Comparison of overheads between the new BIST and other concurrent BIST.

BIST 방식 / 캐쉬 크기	Ternullo's BIST	New BIST	% 감소율
4 K 바이트	3541	3162	11.99
8 K 바이트	3547	3176	11.68
32 K 바이트	3539	3171	11.61
128 K 바이트	3542	3181	11.35
512 K 바이트	3536	3172	11.48
1 M 바이트	3540	3175	11.50

VI. 결 론

일반적으로 메모리 테스트 시 중요한 문제는 테스트 시간과 고장 검출률을 들 수 있다. 그리고 내장된 메모리의 경우 테스트 접근이 어려운데, 이 문제를 BIST를 사용하여 해결하려면 오버헤드의 문제가 또한 부각된다. 그런데 캐쉬 테스트를 고려할 때는 기본적으로 캐쉬의 용량이 그다지 크지 않기 때문에 어느 정도까지는 테스트 시간보다는 고장 검출률과 BIST 하드웨어 오버헤드가 중요하다고 할 수 있다. 메모리 고장 검출률은 고려하는 고장 모델이 무엇인가의 문제와 연결되는데, 본 논문에서는 12N의 복잡도로 SAFs, AFs, TFs linked with CFs, CFins, CFids, SCFs, CFdyns 및 DRFs를 검출하는 새로운 March 알고리듬을 제안하였다. 이것은 적절한 수준의 복잡도로 다양한 고장 모델을 테스트할 수 있는 알고리듬이라고 할 수 있다. 그리고 BIST 하드웨어 오버헤드의 중복성을 피하고 테스트 시간을 단축하기 위해 태그 메모리와 데이터 메모리를 동시에 접근, 테스트하는 방법을 제시하였다. 기존의 동시 테스트 구조가 같은 하드웨어의 중복성 문제를 해결하기 위해 본 논문에서는 캐쉬에 내장된 비교기를 태그 메모리의 결과분석기로 사용하였다. 이를 위해 비교기의 테스트가 선행되어야 하는데 이것을 IEEE 1149.1 표준안의 경계주사 구조 및 명령어를 사용하여 구현하였다. 테스트 클록을 상당히 줄일 수 있도록 주사사슬을 변형하였다. 비교기의 테스트 시간이 일반주사 기법을 이용한 것은 $4w^2 + 6w + 3$의 클록이 필요한데 비해, 멀티플렉서를 이용한 것이 $2w^2 + 4w + 2$의 클록으로 상당히 감소함을 알 수 있다. 전체 메모리 BIST는 기존 동시 테스트 메모리 BIST에 비해 11% 정도의 오버헤드 감소를 확인했다. 결론으로 본 논문에서 제안하는 알고리듬, 메모리 구조 BIST 구조 및 전체 테스트 방법은 하드웨어 중복성을 피할 수 있고, 내장 메모리에 대한 테스트 접근성을 향상시키며, 경계주사 기법을 통해 전체적인 관점의 캐쉬 테스트를 가능하게 해준다. 또한 테스트 회로에 대한 테스트 및 오버헤드의 문제를 줄일 수 있는 방법이라고 할 수 있다.

참 고 문 헌

[1] C. Pyron, J. Praudo, and J. Golab, "Test Strategy for the PowerPC 750 Microprocessor," *IEEE Design & Test of Computers*, pp. 90-97, 1998.

[2] D. K. Bhavsar, D. R. Akeson, M. K. Gowan,

and D. B. Jackson, "Testability Access of the High Speed Test Features in the Alpha 21264 Microprocessor," *Proc. of International Test Conference*, pp. 487-499, 1998.

[3] M. P. Kusco, B. J. Robbins, T. J. Snethen, P. Song, T. G. Foote, and W. V. Huott, "Microprocessor Test and Test Tool Methodology for the 500 MHz IBM S/390 G5 Chip," *Proc. of International Test Conference*, pp. 717-725, 1998.

[4] R. Dekker, "Fault Modeling and Test Algorithm Development for Static Random Access Memories," *Proc. of IEEE International Test Conference*, pp. 343-352, Washington D. C., 1988.

[5] H. Bonges, and R. D. Adams, "A 576k 3.5 ns Access BiCMOS ECL Static RAM with Array Built-in Self-Test," *IEEE Journal of Solid State Circuits*, Vol. 27, No. 4, 1992, pp. 649-656.

[6] R. D. Adams, G. S. Koch, J. Connor, and L. Ternullo, "A 370-MHz Memory Built-in Self Test Machine," *Proc. of the European Design and Test Conference*, 1995, pp. 139-141.

[7] P. H. Bardell and W. H. McAnney, "Self-Test of Random Access Memories," *Proceedings of International Test Conference*, 1985, pp. 352-355.

[8] W. V. Huott T. J. Koprowski, B. J. Robbins, M. P. Kusko, S. V. Pateras, D. E. Hoffman, T. G. McNamara, and T. J. Snethen, "Advanced Microprocessor Test Strategy and Methodology," *IBM Journal of Research and Development*, 1997.

[9] B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Technique," *Proc. of International Test Conference*, October 1979, pp. 37-41.

[10] L. Ternullo Jr., R. D. Adams, J. Connor, G. S. Koch, "Deterministic Self-Test of A High-Speed Embedded Memory And Logic Processor Subsystem," *Proc. of International Test Conference*, pp. 33-44, 1995.

[11] R. Patel and K. Yarlagadda, "Testability Feature of The SuperSPARCTM Microprocessor," *Proc. of International Test Conference*, pp. 773-781, October, 1993.

[12] M. Levitt, S. Nori, S. Narayanan, G. Grewal, L. Youngs, A. Jones, G. Billus and S. Paramanandam, "Testability, Debuggability, and Manufacturability Features of the Ultra SPARC-I Microprocessor," *Proc. of International Test Conference*, pp 157-166, 1995.

[13] P. Shephard III, W. Huott, P. R. Turgeon, R. W. Berry, Jr., P. Patel, G. Yasar, J. Hanley, and F. J. Cox, "Programmable Built-In Self Test Method and Controller for Arrays," U. S. Patent 5,633,877, May, 1996.

[14] W. Huott, T. J. Slegel, T. Lo, and P. Patel, "Programmable Computer System Element with Built-In Self Test Method and Apparatus for Repair During Power-On," U. S. Patent 5,659,551, May 1966.

[15] J. H. Dreibelbis, E. L. Hedberg, and J. G. Petrovick, Jr., "Built-in Self-Test for Integrated Circuits," US. Patent umber 5,173,906.

[16] P. H. Bardell, W. McAnney, and J. Savior, "Built-in Test for VLSI: Pseudo-Random Techniques," Chapter 7, *J. Willey and Sons*, New York, 1987.

[17] R. Dekker, F. Beenker, and L. Tijssen, "A Realistic Fault Models and Test Algorithms for Static Random Access Memories," *IEEE Trans. on Computer-Aided Design*, Vol. 9, No. 6 1990.

[18] Yong Seok Kang, Jong Cheol Lee, and Sungho Kang "An Efficient Built-In Self Test for High Density SRAMs," *Proc. of International Conference on VLSI and CAD*, pp. 52-54, October, 1997.

[19] D. Suk and S. Reddy, ", A March Test for Functional Faults in Semiconductor Random-Access Memories," *IEEE Trans. on Computers*, Vol. C-30, pp. 982-985, Dec. 1981.

[20] Myung-Hoon Yang, Jong Cheol Lee, and

Sungho Kang, "A New BIST for Diagnosis and Transparent Testing in High Density SRAMs," *Proc. of International Technical Conference on CSCC*, pp. 1317-1320, July 1998.

저 자 소 개

金 弘 植(正會員)
1997년 2월 연세대학교 전기공학과 졸업. 1999년 8월 현재 동 대학원 전기 및 컴퓨터 공학과 석사졸업. 1999년 8월~현재 동 대학원 전기 및 컴퓨터 공학과 박사과정

姜 成 昊(正會員)
1986년 2월 서울대학교 제어계측공학과 졸업(학사). 1992년 5월 The University of Texas at Austin(공박). 1989년 11월~1992년 8월 Schlumberger Inc. Research Scientist. 1992년 9월~1992년 10월 The Univ. of Texas at Austin Post Doctorial Fellow. 1992년 8월 1994년 6월 Motorola Inc. Senior Staff Engineer. 1994년 9월~현재 연세대학교 전기공학과 부교수

尹 度 鉉(正會員)
1998년 2월 연세대학교 전기공학과 졸업. 1999년 3월~현재 동 대학원 전기 및 컴퓨터 공학과 석사과정