논문 2005-42SD-2-7

고장 모델 기반 메모리 BIST 회로 생성 시스템 설계

(Memory BIST Circuit Generator System Design Based on Fault Model)

이 정 민*, 심 은 성*, 장 훈**

(Jeong-Min Lee, Eun-Sung Shim, and Hoon Chang)

요 약

본 논문에서는 사용자로부터 테스트하고자 하는 고장 모델을 입력받아 적절한 march 테스트 알고리즘을 만들고 BIST 회로를 생성해 주는 Memory BIST Circuit Creation System(MBCCS) 을 제안하고 있다. 기존의 툴들은 널리 사용되고 있는 알고리즘에 국한되어 메모리의 사양이 변할 경우 거기에 맞는 BIST 회로를 다시 생성해주는 번거로움이 있었다. 하지만 본 논문에서 제안한 툴에서는 다양해진 메모리 구조에 적합한 메모리 BIST 회로를 사용자 요구에 맞는 알고리즘을 적용해서 자동적으로 생성하게 하였고, 임의적으로 선택된 고장 모델에 대한 알고리즘을 제안된 규칙에 따라 최적화함으로 해서 효율성을 높였다. 또한 다양한 크기의 폭을 갖는 주소와 데이터를 지원하며 IEEE 1149.1 회로와의 인터페이스도 고려하였다.

Abstract

In this paper, we propose a memory BIST Circuit Creation System which creates BIST circuit based on user defined fault model and generates the optimized march test algorithm. Traditional tools have some limit that regenerates BIST circuit after changing the memory type or test algorithm. However, this proposed creation system can automatically generate memory BIST circuit which is suitable in the various memory type and apply algorithm which is required by user. And it gets more efficient through optimizing algorithms for fault models which is selected randomly according to proposed rule. In addition, it support various address width and data and consider interface of IEEE 1149.1 circuit.

Keywords: BIST, Memory BIST, Embedded Memory, SRAM, Test

I. 서 론

최근 들어 하나의 칩에 대한 회로의 집적도는 시스템의 고성능화, 고기능화 및 소형화 요구와 함께 설계, 공정 기술의 발달에 힘입어 급속하게 증가되고 있다. 이에 따라, 제한된 면적 안에 더 많은 트랜지스터를 집적시킬 수 있게 되었으며, 칩의 기능을 더욱 향상시키기 위해 예전에는 칩 외부에 배치하였던 메모리 같은 모듈들도 이제는 칩 안에 내장되는 추세이다.

이와 같이 복잡해진 칩의 테스트는 갈수록 어려운 문제가 되어가고 있으며, 특히 내장된 메모리의 테스트는 가장 어려운 부분으로 여겨지고 있다. 메모리가 내장되기 전의 칩 테스트는 테스트 용이화 설계 기법(Design for Testability)과 경계 주사(Boundary Scan)기법을 사용하여 어느 정도의 오버헤드를 감소하는 수준에서 만족할 만한 테스트 결과를 얻을 수 있었다[1-2].

IEEE 1149.1은 VLSI급의 칩들이 다수 포함되어 있는 기판 수준의 테스트를 효율적으로 하기 위하여 제안되었으며[3-4], 현재 대부분의 반도체 회사들이 이를 칩의 구현에 실제 적용하고 있다[5-7]. IEEE 1149.1을 적용할 경우 기판상의 칩 상호간 연결 상태를 쉽게 테스트할 수 있으며, 기판상의 칩들에 적용된 테스트 관련 회로의 표준화된 인터페이스를 제공하고, 테스트 관련 입출력 단자 수를 줄일 수 있다. 또한 칩의 디버깅을 시스템 수준에서 할 수 있다는 장점을 가지고 있다. 메모리 BIST 역시 IEEE 1149.1을 이용하여 제어된다[8-9]. IEEE 1149.1을 통하여 메모리 BIST를 동작시키고, 메모리 테스트가 종료된 후 테스트 결과를 칩 외부로 출력한다. 이러한 BIST는 부수적인 면적의 증가 등과 같은 오버헤드를 갖게 되지만, 각 모듈별로 자체적인 테스트가

* 학생회원, ** 정회원 숭실대학교 컴퓨터학과
(Department of Computing, Soongsil University)
※ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어 졌음
접수일자: 2004년8월4일, 수정완료일 2005년2월1일

수행되므로 전체 시스템의 테스트에 있어서 테스트의 복잡도가 크게 줄어들고, 고가의 외부 테스트 장비를 사용하지 않고도 빠른 시간에 테스트를 수행할 수 있다는 장점 때문에 최근 그 사용이 확산되고 있는 추세다.

이렇듯 여러 종류의 메모리가 생산되고 거기에 따라 새로운 알고리즘이 도입되면서 메모리 코어별로 메모리 BIST 모듈을 설계하는 것은 매우 어렵게 되었다. 따라서 이 과정을 자동화하기 위한 툴이 필수적이다. 현재 상용화 툴은 소수의 보편적인 메모리 테스트 알고리즘 중에서만 선택을 해야 하고 사용자가 원하는 새로운 알고리즘의 구현이 불가능하다.

본 논문에서는 기존의 메모리 고장 모델과 그러한 고장을 검출하기 위해 어떠한 알고리즘들이 필요한지를 알아보고, 그러한 각각의 알고리즘들을 적용할 수 있는 BIST 회로를 자동으로 생성해 주는 툴을 제안하였으며 실제로 툴을 이용해 Verilog 코드를 생성하고 이를 이용하여 내장된 메모리를 테스트함으로써 각 알고리즘의 효율성을 검증하였다. 또한 제안된 툴을 통해 구현한 내부 알고리즘에 의해 생성되어 나온 테스트 알고리즘이 기존의 알고리즘들 같이 최적화가 되는지 확인하였다.

Ⅱ. 고장 모델링과 테스트 알고리즘

메모리는 다른 반도체 소자와 마찬가지로 복잡한 제조 공정을 거쳐 만들어지므로 다양한 결함들이 발생할 수 있다. 이러한 결함들은 대부분 이상 현상을 유발하여 메모리의 오동작을 일으키게 된다. 이러한 결함들은 이웃하는 셀 간의 단락, 워드선이나 비트선의 연결 또는 개방, 게이트 옥사이드 단락 등의 다양한 형태로 나타날 수 있다. 가장 대표적인 고장으로 주소 디코더 고장(Address decoder fault), 고착 고장(Stuck at fault), 천이 고장(Transition fault), 결합 고장(Coupling fault) 등이 있다.

1. 고장 모델과 march 테스트

March 테스트는 March 요소(march element)로 구성된다. March 요소는 모든 셀에 부가되는 여러 번의 쓰기와 읽기 동작이다. 메모리의 크기가 N이라 할 때, March 요소는 0번지에서 N-1번지까지 증가하면서, 또는 N-1번지에서 0번지로 감소하면서 각 메모리 셀에 대한 유한개의 읽기 동작 또는 쓰기 동작을 수행한다. March 테스트는 유한 개수의 March 요소로 구성된다. March 요소에서 사용된 기호 ⇓, ⇑, w0, w1, r0, r1의 정의는 다음과 같다.

⇓ : 메모리의 주소를 높은 주소에서 낮은 주소로 감소
⇑ : 메모리의 주소를 낮은 주소에서 높은 주소로 증가
w0 : 메모리 셀에 논리값 0 쓰기
w1 : 메모리 셀에 논리값 1 쓰기
r0 : 메모리 셀에서 논리값 0 읽기
r1 : 메모리 셀에서 논리값 1 읽기

예를 들어 ⇓(r0,w1) 동작은 메모리 주소를 감소시키면서 현재 주소에 해당하는 셀에 논리값 0을 읽고 논리값 1을 쓰는 동작이다. 각각의 고장을 검출하기 위해 사용되는 알고리즘을 요소를 이용해 나타내면 다음과 같다.

가. 주소 디코더 고장 검출

주소 디코더 고장 모델의 경우, 특정 주소로 메모리의 어떤 셀도 접근할 수 없는 주소 디코더 고장은 메모리의 모든 셀에 0(1)을 쓰고 0(1)을 읽어보고, 다시 1(0)을 쓰고 1(0)을 읽어봄으로써 검출된다. 이 과정을 march 요소로 나타내면 ⇓(w0); ⇓(r0, w1); ⇓(r1, w0); 으로 표현 가능하다. 그리고 한 주소가 두 개의 메모리 셀을 접근하는 고장과 서로 다른 주소로 하나의 메모리 셀을 접근하는 고장은 주소가 감소하는 방향으로 1을 읽고 0을 쓴 후, 다시 주소가 증가하는 방향으로 0을 읽고 1을 씀으로써 검출된다. 이 과정은 ⇓(r1, w0); ⇑(r0, w1); 으로 표현 할 수 있다.

나. 고착 고장 검출

고착 고장은 메모리 셀이 고착되어서 그 셀의 논리값이 항상 0 또는 1로 머물러 있는 상태를 의미한다. 예를 들어 그림 1의 (a)와 같이, 가운데에 있는 메모리 셀에 1을 쓰도록 명령을 했지만 (b)의 그림과 같이 아무런 변화가 없이 0으로 고착되어 있는 경우 0으로 고착되었다고 한다. 따라서 고착-1 고장은 0을 쓴 후 0을 읽어봄으로써 검출되고, 고착-0 고장은 1을 쓰고 1을 읽어봄으로써 검출된다. 이 과정은 ⇓(r0, w1); ⇓(r1, w0); 단계에서 모두 완료된다.

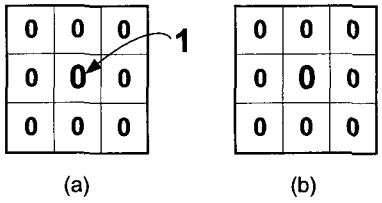

그림 1. 고착-0 고장의 예
Fig. 1. Example of struck at 0 fault.

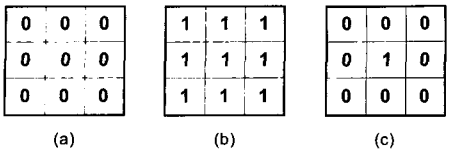

그림 2. 하향 천이 고장의 예
Fig 2. Example of transition fault.

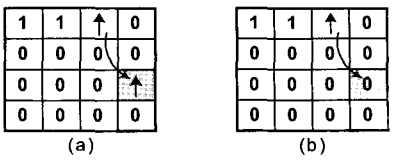

그림 3. 결합 고장의 예
Fig 3. Example of coupling fault.

다. 천이 고장 검출

천이 고장은 고착 고장의 특수한 경우로서 하나의 셀이 0에서 1로 상향 천이를 하지 못할 때 상향 천이 고장, 1에서 0으로의 하향 천이를 하지 못할 때 하향 천이 고장이라 한다. 상향 천이 고장은 0을 읽고 1을 쓴 후, 다시 1을 읽음으로써 검출된다. 마찬가지로 하향 천이 고장은 1을 읽고 0을 쓴 후, 다시 0을 읽음으로써 검출된다. 그림 2는 하향 천이 고장을 자세하게 설명한 그림이다. 그림의 (a)에서와 같이 메모리의 모든 셀이 0을 가진 상태에서 (b)에서처럼 모든 메모리에 1을 쓴다. 그 후 모든 셀에 1이 제대로 써졌는지 확인한 후 다시 모든 셀에 0을 썼을 경우, (c)의 그림과 같이 가운데에 있는 셀은 0으로 하향천이가 이루어지지 않았음을 알 수 있다. 이 과정은 ⇓(r1 ,w0);⇑(r0, w1)단계로 완료된다.

라. 결합 고장 검출

결합 고장이란 2개 또는 그 이상의 셀 사이에서 셀들의 비정상적인 연결로 인해 발생하는 고장이다. 그림 3은 결합 고장의 대표적인 예를 보여주고 있다.

그림 3의 (a)에서처럼 하나의 셀에서 천이 쓰기 동작을 할 때 결합된 두 번째 셀이 원래 내용과 반대로 바뀌는 경우를 반전 결합 고장(Inversion Coupling Fault: CFin)이라고 한다. 그림의 (b)와 같이 하나의 셀에서 천이 쓰기 동작이 일어 날 때 결합된 두 번째 셀이 0 또는 1로 고정되는 경우를 동행 결합 고장(Idempotent coupling Fault: CFid)이라고 한다.이 과정은 영향을 발생하는 셀의 주소가 영향을 받는 셀의 주소보다 하위에 있는 경우와 상위에 있는 경우로 구분하여 설명한다. 먼저 영향을 받는 셀의 위치가 상위에 있는 경우를 보면 영향을 발생하는 셀의 주소에서 ⇓(r0, w1) 또는 ⇑(r0, w1)의 w1 동작 즉, 상향 천이를 발생하면 결함으로 역시 다른 상위 주소 셀의 내용을 0 또는 1로 수정하는 경우가 발생한다. 만일 영향을 받는 셀의 내용이 0으로 수정된 경우는 ⇑(r1, w0)에 의해 검출된다. 또 하향 천이에 의해 영향을 받는 상위 주소의 셀 내용이 0과 1인 경우로 구분된다. 셀 내용이 0으로 수정되는 경우 즉, ⇑(r1, w0)의 w0 동작에 의해 발생된 CF는 다시 (r1, w0)에서 검출된다. 셀 내용이 1로 수정되는 경우 ⇑(r1, w0);의 w0에 의해 발생된 CF는 ⇑(r0) 에 의해 검출된다.

표 1. 결함에 따른 march 알고리즘
Table 1. March algorithm for fault.

March 알고리즘	검출 가능한 고장
MATS(4n)	SAF
MATS+(5n)	SAF, ADF
MATS++(6n)	SAF, TF, ADF
MarchX(6n)	SAF, TF, ADF, CFin
March C-(10n)	SAF, TF, ADF, CFin, CFid

2. March 테스트 알고리즘

March 테스트 알고리즘은 앞에서 언급한 고장들을 검출하는 테스트 알고리즘으로 행진 하듯이 주소 순서에 따라 오름차순 또는 내림차순으로 메모리에 대한 읽기와 쓰기 동작을 수행하는 방법이다. March 테스트 알고리즘은 검출 할 수 있는 결함에 따라 다음 표와 같이 분류할 수 있다.

앞에서 언급한 고장 별 march 요소를 단순히 나열해서 조합을 할 경우 표1과 같은 최적화된 알고리즘을 얻어 낼 수 없다. 가능한 적은 시간 복잡도를 사용하고 요구하는 고장을 검출 할 수 있는 알고리즘을 찾는 것이 중요하다. 다음 장에서는 논문에서 제안한 MBCCS의 구성에 대해 알아보고, 그것을 통해 생성된 알고리즘들이 표1에서의 알고리즘과 같이 최적화 되면서 요구되는 고장을 모두 검출 해 낼 수 있는지 살펴보도록 하겠다.

Ⅲ. 알고리즘을 적용하기 위한 MBCCS 구성

1. MBCCS의 구성

MBCCS의 설정파일 부분에서는 크게 세 부류로 구분된다. 첫 번째로 메모리 모델을 구분하는 부분으로서 단일포트 메모리, 다중포트 메모리 중 테스트할 메모리를 선택하며, 선택한 메모리에 입력, 출력 핀의 이름과 핀의 크기를 정의하고 메모리의 읽기, 쓰기 시의 타이

밍을 기술해 준다. 두 번째로 MBCCS는 여러 개의 같은 사이즈의 메모리 BIST IP가 지원되므로, 테스트 하고자 하는 메모리 모델 개수를 입력한다. 마지막으로 적용할 알고리즘과 배경데이터, 컨트롤 신호등이 입력되면 모든 설정내용을 전략적으로 적용시켜 Verilog HDL로 기술된 BIST IP를 생성해 낸다.

가. 메모리 모델 이름 정의

keyword는 model이며, model_name이 필요하다. 1번 줄에서 메모리 RAM1을 기술한 부분을 볼 수 있다.

나. 메모리 핀 정의

메모리의 핀 정보에 관한 핀 타입 정의 keyword는 address, data_in, data_out, write_en, ram_en, reset 이다. address keyword는 메모리 주소의 이름과 사이즈를 정의한다. data_in은 메모리의 입력 데이터 버스이고 data_out는 메모리의 출력 데이터 버스이다. <name>과 <bit_width>는 각각 address, data_in, data_out의 핀 이름과 사이즈를 정의한다. write_en, read_en, ram_en은 메모리의 컨트롤 신호를 정의한다. <pin>은 각 컨트롤 신호의 이름을 정의한다. 그리고 <assert_state>는 컨트롤 신호의 액티브 상태를 정의하는 것으로써 high 또는 low로 표시한다. 예를 들어 주소의 이름과 사이즈는 ADDR, 3-bit이고, 데이터의 입출력 포트의 이름과 사이즈는 각각 DIN, DOUT, 4-bit이다. 그리고 각 컨트롤 신호의 이름과 <assert_state> 상태는 각각 WEN (low), REN(low), RAMEN(low) 이다. 메모리 핀 정의를 포함한 정의는 1~6 줄에서 볼 수 있다.

```
1:  model RAM1( //메모리 이름 기술
2:      address ADDR 4        //4비트 주소 포트
3:      data_in DIN 8         //8비트 입력 포트
4:      data_out DOUT 8       //8비트 아웃 포트
5:      write_en WEN high     //active high
6:      read_en REN high      //active high
7:
8:      read_write_port (
9:              read_cycle( //읽기 사이클 기술
10:             change ADDR
11:             wait
12:             assert REN
13:             expect DOUT
14:             wait
15:             wait
16:     )                     //읽기 사이클 정의 끝
17:     write_cycle( //쓰기 사이클 기술
18:             change ADDR
19:             change DIN
20:             wait
21:             assert WEN
22:             wait
23:             wait
24:     )       //쓰기 사이클 정의 끝
25: )           //읽기_쓰기 포트 정의 끝
26: )           //모델 정의 끝
```

다. 읽기/ 쓰기 포트와 동작을 정의

포트의 타입은 read_port, write_port, read_write_port중 하나이다. 포트의 타입 안에는 사이클 타입을 정의한다. 사이클 타입은 read_cycle과 write_cycle로 구성된다. 그리고 각 사이클 타입 안에는 읽기/쓰기 동작이 기술된다. 포트의 정의는 아래와 같다.

```
port_type (
    cycle_type (
        <operation expression>
    )
)
```

<operation expression>는 메모리의 읽기/쓰기 동작을 기술한다. 읽기/쓰기 동작을 기술하기 위하여 사용되는 keyword에는 change, assert, wait, except가 있다. change keyword는 address나 data의 값 변화를 표시한다. assert는 컨트롤 신호를 활성 상태로 변경할 때 사용된다. assert를 설정하고 한 클럭이 지나면 자동으로 비활성화 상태로 되돌아간다. 따라서 두 클럭 이상을 활성화 상태로 유지하려면 매 클럭마다 assert를 설정하여야 한다. wait는 한 클럭 시간 흐름을 나타낸다. change, assert는 값을 설정할 뿐 시간 흐름을 설정하지는 않는다. except는 메모리의 읽기 동작 시 유효한 데이터가 출력되는 시점을 나타낸다. 8~25 줄의 정의를 읽기 쓰기 동작 파형으로 나타내면 그림 6과 같이 나타낼 수 있다.

라. 메모리 등록 단계

메모리 모델을 구분하는 단계가 끝이 나면 메모리 모델 등록 (Memory Config) 에서는 생성한 메모리 모델을 종류와 개수를 등록할 수 있다. 앞서 정의된 메모리

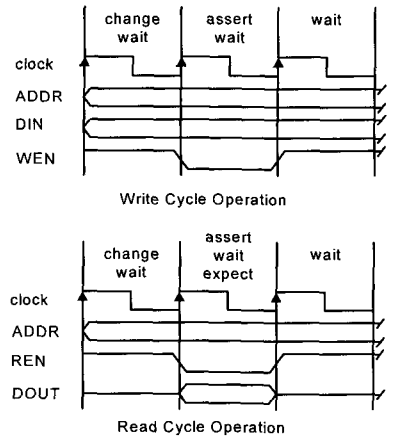

그림 4. 읽기/쓰기 동작 파형
Fig. 4. read/write waveform.

모델의 종류들이 같으면 그 개수와 상관없이 BIST IP 하나를 생성한다. 반면에 메모리 개수에는 상관없이 다른 종류의 모델이 있다면 메모리 모델의 개수만큼 BIST IP를 생성해 준다.

마. 테스트 알고리즘 적용 (MBIST Configuration)

User Define Mode에서는 폴트별로 AF, SAF, TF, CFin, CFids를 단일 혹은 조합가능하게 선택 할 수 있다. 고장별로 선택을 하게 되면 구현한 프로그램 내부에서는 최적화된 March 시퀀스를 생성해서 실행하게 된다. 그리고 Select March Algorithm은 기존의 March Algorithm을 선택해서 그대로 적용 가능하다. 뿐만 아니라 일반적으로 제공해주는 알고리즘이 아닌 사용자가 원하는 알고리즘을 메모리 모델에 상관없이 적용시킬 수 있다. 이 단계에서는 알고리즘 적용뿐 아니라 실제 BIST IP에서 사용하게 될 Clock 신호와 Reset 신호와 같은 제어 신호의 이름을 정의한다.

2. MBCCS의 메모리 알고리즘 최적화

March 테스트는 빠르고, 단순하며 규칙적인 구조를 가지고 있다. March 테스트의 내부 동작 각각의 구성을 원소로 구별하며 이 원소들의 개수가 March 테스트의 복잡도로 계산된다. 원소들은 메모리의 각 셀에서 읽기, 쓰기 동작을 순차적으로 메모리 주소의 증가, 감소하면서 동작을 수행한다. 기존에 선보인 March 알고리즘이 아닌 사용자가 원하는 고장 검출을 위한 알고리즘을 생성하려면 시간과 복잡도면에서 볼 때 상당히 많은 시간이 소비 될 것이며 높은 복잡도를 가질 것이고, 이렇게 만들어진 March 알고리즘이 정확한 알고리즘이라고 보장하기 어려울 것이다. 본 논문에서 사용하는 메모리 최적화 March 알고리즘은 다수의 메모리 고장을 검출 할 수 있으며, 고장 모델로는 대표적인 메모리 고장 모델인 고착 고장(Stuck-at Fault), 천이 고장(Transition Fault), 두 셀 간의 고장(Coupling Fault), 주소 디코더 고장(Address Decoder Fault)들을 사용하였다. 기존에 제시된 March 알고리즘 뿐만 아니라 사용자가 원하는 고장 알고리즘의 최적화와 사용자가 원하는 알고리즘을 직접 기술하여 적용시킬 수 있다. 최적화 알고리즘은 그림 5의 순서도로 진행된다.

중복되는 동작들을 최적화 규칙에 따라 삭제하는 것은 아래의 표1에서 나오는 규칙을 따른다. 규칙1은 메모리 초기화를 위해서 쓰기 동작이 선행 되어야함을 나타낸다. 규칙2에서는 메모리 테스트 알고리즘의 마지막 동작은 올바른 값의 유무를 판단하기 위해 읽기 동작이어야 한다는 것이다. 규칙3과 4에서는 중복된 동작에 대한 삭제 원칙을 설명한다.

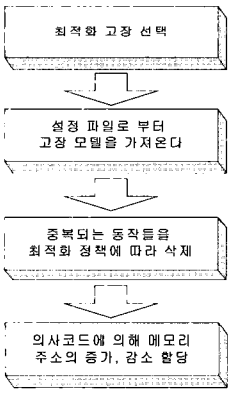

그림 5. 테스트 알고리즘 최적화 순서도
Fig. 5. Flowchart for optimizing test algorithm.

표 2. 최적화에 따른 삭제 규칙
Table 2. Elimination rule for optimization.

삭제 규칙	내 용
규칙 1	메모리 초기 동작은 w0, w1이어야 함
규칙 2	메모리 마지막 동작은 r0, r1이어야 함
규칙 3	원소 중복에 의해 삭제 대상인 동작의 근접한 동작의 삭제 후 영향을 고려해 삭제 함
규칙 4	삭제 후 근접한 동작은 두 개 이상의 동작 연결이 가능

Ⅳ. 생성된 메모리 BIST 회로의 구조

MBCCS에 의해 생성된 메모리 BIST IP는 IP 전체를 제어하는 제어 모듈(CONTROL Module)과 테스트 패턴을 인가하기 위한 주소를 생성 하는 주소 생성 모듈(AGL Module), 테스트 패턴을 만들어 내는 데이터 생성 모듈(DGL Module), 마지막으로 테스트 패턴을 인가한 후 고장 유무를 판단하기 위해 메모리로부터 데이터를 받아 비교하는 데이터 비교 모듈(DCL Module)로

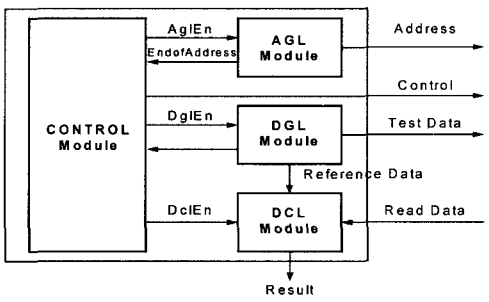

그림 6. 단일 포트 메모리 BIST IP 구조
Fig. 6. BIST IP structure for single port memory.

나뉜다. 그림 6에서는 단일 포트 메모리 BIST 구조를 나타내고 있다.

1. 제어 모듈 (CONTROL Module)

제어 모듈은 테스트 진행 과정 중에 메모리 BIST IP의 각 모듈의 동작을 제어하는 회로이다. 이 모듈은 전체적인 테스트 시작과 종료 시점을 판단하며, 각각의 모듈에 적절한 신호를 인가하여 테스트가 원활하게 돌아갈 수 있도록 총체적인 역할을 담당한다. 제어 모듈은 설정파일에 입력된 알고리즘과 배경데이터는 각각의 상태머신이 존재하며, 알고리즘과 배경데이터를 상태머신으로 분석해 메모리의 주소가 끝날 때까지 알고리즘은 단계적으로 반복하여 알고리즘 부분의 각 단계에서 적절한 배경 데이터를 인가할 수 있도록 데이터 생성 모듈에 DGLEnable 신호를 주고, 테스트 데이터가 메모리의 정확한 주소에 읽기, 쓰기를 할 수 있도록 주소를 생성해주는 주소 생성 모듈에 AGLEnable 신호를 만들어 준다. 마지막으로 고장의 유무를 판단하는 데이터 비교 모듈에 DCLEnable 신호를 인가하여 고장을 판단할 수 있는 구조로 동작한다.

2. 주소 생성 모듈 (AGL Module)

주소 생성 모듈은 테스트 모드 시에 제어 모듈로부터 생성되는 신호를 받아 테스트 데이터 값을 정확한 위치에 읽고, 쓰기 할 수 있도록 메모리의 0번지 주소에서부터 마지막 주소까지 순차적으로 증가, 감소 할 수 있는 카운터를 만들었으며 증가 상태에서 감소상태로 변할 때 혹은 감소 상태에서 증가 상태로 변할 때 주소의 보수를 취해 원하는 주소를 생성할 수 있도록 하였다. 또한 여러 개의 메모리가 사용될 경우 현재 테스트가 진행 중인 메모리의 주소의 크기에 맞도록 마지막 메모리 주소를 조정할 수 있게 구현하였다.

3. 데이터 생성 모듈 (DGL Module)

데이터 생성 모듈은 제어 모듈에서 발생하는 DGLEnable 신호에 의해 테스트 패턴을 만들어 내는 모듈이다. 현재 테스트가 진행 중인 메모리를 위한 데이터나 배경데이터를 생성해 메모리에 전달한다.

4. 데이터 비교 모듈 (DCL Module)

데이터 비교 모듈은 메모리에서 읽혀진 값과 데이터 비교 모듈에서 생성된 데이터 값을 읽어 비교 수행하는 모듈이다. 메모리에서 읽혀진 값과 데이터 비교 모듈에서 생성된 값이 각종 연산을 통해 결과를 쉬프트 연산에 의해 IEEE 1149.1 또는 IEEE P1500을 통해 외부로 최종 테스트 결과 값을 출력한다.

V. 제안된 메모리 자체 테스트 회로의 검증 및 성능평가

본 논문에서 개발한 MBCCS를 검증하기 위해 다양한 구성 형태와 알고리즘을 사용하여 생성된 BIST IP를 Xilinx사의 Xilinx Foundation에서 제공하는 simulator를 사용하여 RTL 검증을 하였으며, GTL 검증은 Xilinx사의 Spartan XCS40PQ240 FPGA를 사용하여 합성 후 실제 동작을 검증하였다. 또한 고장 모델별로 취사선택하여 최적화 테스트 알고리즘을 생성한다. 이 최적화 테스트 알고리즘을 생성하는 프로그램은 Windows OS환경 하에서 구현하였으며 소프트웨어 개발 툴은 그래픽 사용자 인터페이스의 작성을 위하여 Microsoft 사의 Visual C++ Compiler를 사용하였다.

그림 7은 고장 모델 ADF(Address Decoder Fault), TF(Transition Fault), SAF(Stuck-at Fault)을 선택하여 March 알고리즘을 생성하여 준다. 알고리즘 d(w0)u(r0,w1)u(r1,w0)d(r0)이 생성되기까지의 CPU시간은 0.359초가 걸린다. 생성되어 나온 알고리즘의 요소들은 다음 표 4와 같이 설명한다. 프로그램에서 구현한 내부 알고리즘에 의해 생성되어 나온 최적화 테스트 알고리즘 d(w0)u(r0,w1)u(r1,w0) d(r0)는 표 3에서 보는 것과 같이 기존의 MATS++와 같음으로써 최적화가 되었음을 알 수 있다.

그림 8은 모든 경우의 고장 모델 ADF(Address Decoder Fault), TF(Transition Fault), SAF(Stuck-at Fault), CFin(Inversion Coupling Fault), CFids (Idempotent Coupling Fault)을 선택하여 최적화 테스트 알고리즘을 생성하는 화면을 보여준다. 알고리즘

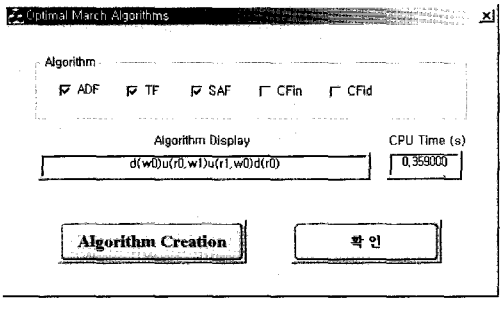

그림 7. ADF, TF, SAF 고장 모델 선택 알고리즘 생성
Fig. 7. generation of algorithm for ADF, TF and SAF.

표 3. 실험 결과
Table 3. Experimental Results.

고장 리스트					생성된 March 테스트와 시간 복잡도		CPU 시간(s)	기존의 March 테스트
SAF	TF	ADF	CFin	CFid				
O					u(w1)d(r1,w0)d(r0)	4n	0.326	MATS(4n)
O		O			u(w1)u(r1,w0)d(r0,w1)	5n	0.334	MATS+(5n)
O	O	O			u(w0)u(r0,w1)d(r1,w0)u(r0)	6n	0.359	MATS++(6n)
O	O	O	O		u(w0)d(w1)d(r1,w0)u(r0w1)	6n	0.362	MarchX(6n)
O	O	O	O	O	u(w0)u(r0,w1)u(r1,w0)d(r0,w1)d(r1,w0)d(r0)	10n	0.375	MarchC-(10n)
			O		u(w0)u(r0,w1w0)d(r0)	5n	0.341	Not Found

표 4. 생성된 알고리즘 요소에 대한 설명
Table 4. Specification for generated algorithm element.

알고리즘 요소	동작 설명
d(w0)	메모리 상위에서 하위주소로 진행하면서 셀에 '0' 값을 써준다.
u(r0,w1)	메모리 하위에서 상위주소로 진행하면서 셀에 '0'을 읽고 '1'을 쓴다.
u(r1,w0)	메모리 하위에서 상위주소로 진행하면서 셀에 '1'값을 읽고'0'값을 쓴다.
d(r0)	메모리 상위에서 하위주소로 진행하면서 셀에 '0' 값을 읽는다.

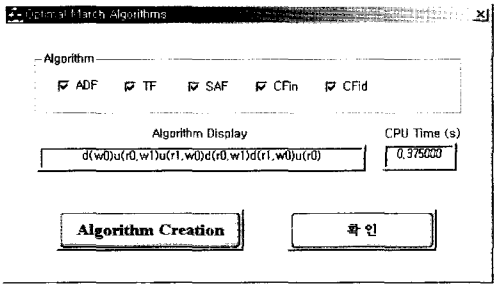

그림 8. ADF, TF, SAF, CFin, CFids 고장 모델 선택 알고리즘 생성
Fig. 8. generation of algorithm for ADF, TF, SAF, CFin and CFids.

d(w0) u(r0,w1)u(r1,w0)d(r0,w1)d(r1,w0)u(r0) 이 만들어 지기까지의 CPU 시간은 0.375초가 소요된다.

프로그램에서 구현한 내부 알고리즘에 의해 생성되어 나온 최적화 테스트 알고리즘 d(w0)u(r0,w1)u(r1,w0)d(r0,w1)d(r1,w0)u(r0) 또한 표3에서 보는 것과 같이 기존의 March C- 알고리즘과 같음으로써 최적화가 되었음을 알 수 있다. 사용자가 검출하고자 하는 고장 모델을 선택한 후 본 논문에서 제안한 툴을 실행하여 생성된 march 테스트 알고리즘의 시간 복잡도에 대한 실험 결과를 표3에서 확인 할 수 있다. 제안된 툴에서 생성된 알고리즘은 기존의 march 테스트와 같은 march 요소와 시간 복잡도를 갖는 것을 볼 수 있다. 실험 결과에서도 볼 수 있듯이 ADF, TF, SAF, CFin, CFids 모두 검출 가능한 march 테스트 알고리즘을 선택한 경우 10n으로 모든 고장을 검출할 뿐만 아니라 시간 복잡도

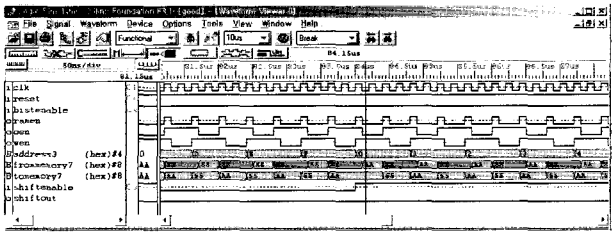

그림 9. 단일 포트 메모리 정상 파형
Fig. 9. normal waveform for single port memory.

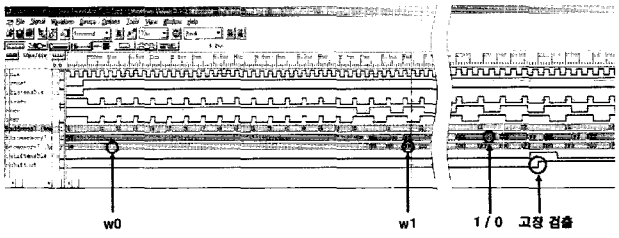

그림 10. 단일 포트 메모리 고장 검출 파형
Fig. 10. waveform for single port memory detected fault.

또한 다른 알고리즘과 많은 차이를 보이지 않고 있다. 따라서 비슷한 시간 복잡도를 가지면서 모든 고장 모델을 검출 가능한 march C-를 사용하는 것이 더 효율적이라는 것을 확인 할 수 있다. 그림 9는 MBCCS를 이용해 단일 포트 메모리 BIST IP를 생성하여 Xilinx Foundation에서 제공하는 simulator를 통해 검증한 시뮬레이션 파형 이며 고장이 없을 때를 보여주고 있다.

단일 포트를 위한 테스트는 March C- 알고리즘을 사용하였다. 만약 메모리에 고장이 있었다면 shiftenable 신호에 액티브 하이 값을 인가한 시점 이후 shiftout 신호에 액티브 하이 신호가 나오도록 설계되었다. 그림 10는 그림 9과 같은 조건에서 메모리에 고장이 발생했을 경우의 시뮬레이션 파형을 보여주고 있다.

그림 10은 천이고장이 발생한 메모리의 파형으로 메모리에 16진수 값 00h를 쓰기한 후, 1h번지에서 FFh값 쓰기 동작이 후에 1h번지의 값을 읽을 경우, 그림 11과 같이 FFh 값을 읽어 와야 한다. 그러나 고장의 영향으로 00h값을 읽어와 천이 고장이 발생하였음을 알 수 있다. shiftenable 신호를 액티브 하이로 인가하여 고장 발

생을 shiftout 신호를 통해 알 수 있다.

VI. 결 론

최근에 SoC 환경이 급속히 늘어 가면서 상당 부분의 비중을 차지하고 있는 내장된 메모리에 대한 테스트에 대한 많은 연구가 진행되고 있다. 하지만 다양한 메모리가 생산되고 거기에 따른 새로운 결함들과 그것들을 해결하기 위한 새로운 알고리즘들이 도입 되면서 기존의 개별적인 수동적 메모리 BIST IP 모듈 설계 방법은 한계들 드러내고 있다.

본 논문에서는 기존의 메모리를 테스트를 위한 메모리 BIST IP의 개발을 손쉽게 구현 할 수 있도록 범용적인 MBCCS(Memory BIST Circuit Creation System)를 통해 이러한 한계점을 극복하였다. 기존의 알고리즘 뿐만 아니라 사용자가 정의한 알고리즘도 적용이 가능하며 새롭게 생성되는 알고리즘은 제안된 규칙에 의해 최적화 된다. 다중 메모리의 경우 BIST 회로 공유를 통한 SOC 내의 면적 오버헤드를 최적화하였다. 제안된 툴을 이용하여 메모리 테스트를 위한 회로를 자동으로 생성함으로써 메모리 BIST IP의 설계에 소요되는 많은 시간과 노력을 줄일 수 있다. 특히 다양한 종류의 메모리와 사용자 요구에 따른 BIST 회로의 재구성을 위한 시간과 노력을 줄일 수 있다.

참 고 문 헌

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, Digital system testing and testable design, "Computer Science Press" 1990.

[2] Test Technology Standards Committee, IEEE Standard Test Access Port and Boundary Scan Architecture, "IEEE Computer Society Press", 1990.

[3] IEEE Stndard 1149.1-1990, "IEEE Standards Test Access Port and boundary-scan Architecture," IEEE Standards Board, New York, 1990.

[4] Test Technology Standards Committee, "IEEE Standard Test Access Port and Boundary-Scan Architecture," IEEE Computer Society Press, 1993.

[5] Parulkar, I., Ziaja, T., Pendurkar, R., D'Souza, A. and Majumdar, A., "A scalable, low cost design -for-test architecture for UltraSPARC/spl trade/ chip multi-processors," International Test Confe -rence,. Vol. 7, no. 10, pp. 726-735, Oct 2002.

[6] Braden, J., Lin, Q. and Smith, B., "Use of BIST in Sun FireTM servers," International Test Conference,. pp. 1017-1022, 2001.

[7] R. Raina, R. Bailey, D. Belete, V. Khosa, R. Molyneaux, J. Prado, A. Razdan, "DFT Advances in Motorola's Next-Generation 74xx PowerPC TM Microprocessor," In Proc. IEEE International Test Conference,. pp. 131-140, 2000.

[8] Daehan Youn, Ohyoung Song and Hoon Chang, Design-for-testability of the FLOVA," In Proceedings of the Second IEEE Asia Pacific Conference, pp. 28-30, 2000.

[9] Appello, D., Fudoli, A., Tancorre, V., Corno, F., Rebaudengo, M. and Sonza Reorda, M., "A BIST-based solution for the diagnosis of embedded memories adopting image processing techniques," In Proc. International On-Line Testing Workshop, Vol. 8, no. 10, 2002pp. 206-210, July 2002.

저 자 소 개

이 정 민(학생회원)
2004년 숭실대학교 컴퓨터학부 학사 졸업.
2004년 3월~현재 숭실대학교 대학원 컴퓨터학과 석사과정
<주관심분야 : VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD>

심 은 성(학생회원)
2003년 한서대학교 컴퓨터정보학과 학사졸업.
2003년 3월~현재 숭실대학교 대학원 컴퓨터학과 석사과정
<주관심분야 : VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD>

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 : VLSI 설계 및 테스트, 컴퓨터구조, VLSI CAD>