논문 2013-50-2-14

IEEE 1500 표준 기반의 효율적인 프로그램 가능한 메모리 BIST

(IEEE std. 1500 based an Efficient Programmable Memory BIST)

박 영 규*, 최 인 혁*, 강 성 호**

(Youngkyu Park, Inhyuk Choi, and Sungho Kang)

요 약

Systems-On-Chips(SoC)에서 내장 메모리가 차지하는 비중은 비약적으로 증가하여 전체 트랜지스터 수의 80%~90%를 차지하고 있어, SoC에서 내장된 메모리에 대한 테스트 중요성이 증가하고 있다. 본 논문은 다양한 테스트 알고리즘을 지원하는 IEEE 1500 래퍼 기반의 프로그램 가능한 메모리 내장 자체 테스트(PMBIST) 구조를 제안한다. 제안하는 PMBIST는 March 알고리즘 및 Walking, Galloping과 같은 non-March 알고리즘을 지원하여 높은 flexibility, programmability 및 고장 검출률을 보장한다. PMBIST는 최적화된 프로그램 명령어와 작은 프로그램 메모리에 의해 최적의 하드웨어 오버헤드를 가진다. 또한 제안된 고장 정보 처리 기술은 수리와 고장 진단을 위해 2개의 진단 방법을 효과적으로 지원하여 메모리의 수율 향상을 보장한다.

Abstract

As the weight of embedded memory within Systems-On-Chips(SoC) rapidly increases to 80-90% of the number of total transistors, the importance of testing embedded memory in SoC increases. This paper proposes IEEE std. 1500 wrapper based Programmable Memory Built-In Self-Test(PMBIST) architecture which can support various kinds of test algorithm. The proposed PMBIST guarantees high flexibility, programmability and fault coverage using not only March algorithms but also non-March algorithms such as Walking and Galloping. The PMBIST has an optimal hardware overhead by an optimum program instruction set and a smaller program memory. Furthermore, the proposed fault information processing scheme guarantees improvement of the memory yield by effectively supporting three types of the diagnostic methods for repair and diagnosis.

Keywords : Memory BIST, IEEE std. 1500, Test Algorithm, Diagnostic

Ⅰ. 서 론

반도체 공정 기술과 설계 기술이 발달함에 따라 많은 수의 Intellectual Property(IP) 코어들이 System on Chip(SoC)화 되고 있다. 전체 SoC의 복잡도 및 트랜지스터는 Moor의 법칙에 따라 기하 급수적으로 증가하고 있고, 내장 메모리가 차지하는 비중이 급속이 증가하여 전체 트랜지스터 수의 80%~90%를 차지해 SoC에서 내장 메모리에 대한 테스트 중요성이 점점 증가하고 있다. 내장 메모리의 비중이 증가하는 반면에 다양한 크기의 메모리들이 사용되어 테스트에 많이 시간이 필요하게 되었다. 따라서 Automatic Test Equipment(ATE)

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
※ 본 연구는 지식경제부 및 정보통신산업진흥원의 IT융합 고급인력과정 지원사업의 연구결과로 수행되었음(NIPA-2013-H0401-13-1006)
접수일자: 2012년11월8일, 수정완료일: 2013년1월20일

를 이용하여 테스트하는 방법은 많은 테스트 시간이 필요하며, at-speed 테스트가 불가능하다.

현재 내장 메모리의 테스트는 고가의 외부 테스트 장비를 사용하지 않고, 디바이스 별로 자체적인 테스트를 수행하여 전체 시스템의 테스트 복잡도를 크게 줄여 빠른 시간에 테스트를 수행할 수 있는 내장 자체 테스트 (BIST: Built-In Self-Test) 기법을 많이 사용하고 있다[1][2]. 또한 메모리 BIST 기법은 테스트를 위하여 수많은 포트가 필요하지 않으며, 메모리의 동작 속도로 테스트가 가능하여 at-speed 테스트가 가능하다[3]. 그러나 메모리 BIST 기법은 제한적인 테스트 알고리즘만을 지원하여, flexibility가 낮고 고장 검출률이 제한적이라는 단점을 가진다.

따라서 메모리 BIST의 단점을 보완한 프로그램 가능한 내장 자체 테스트 (PMBIST: Programmable Memory Built-In Self-Test) 기법이 제안되었다[4]. PMBIST는 크게 micro-code를 이용한 방식과 Finite State Machine(FSM)을 이용한 방식으로 나누어진다. FSM을 이용한 방식은 기존 메모리 BIST에 비해 다양한 알고리즘을 지원할 수 있지만, March 기반의 알고리즘만 지원이 가능하여 flexibility와 고장 검출률이 제한적이다. Micro-code를 이용한 방식은 알고리즘을 구현하기 위하여 명령어를 사용하며, 명령어 구조에 따라 다양한 테스트 알고리즘을 보다 쉽게 지원할 수 있다는 장점을 가진다. 하지만 non-March 알고리즘과 같은 복잡한 알고리즘을 구현하기 위해서는 회로의 복잡도가 커져 하드웨어 오버헤드가 증가한다는 단점을 가진다.

또한 SoC의 내장된 코어에 대한 접근 방법 및 테스트 절차, 테스트 패턴의 입력 및 관측 방법에 대한 테스트 인터페이스로 IEEE 1500 표준과 시스템 수준에서 테스트 설계 기술로 IEEE 1149.1 표준이 있다[5~6]. IEEE 1500과 IEEE 1149.1을 사용하여 SoC 내부 테스트를 위하여 효율적인 제어가 가능하다.

본 논문에서는 내장된 메모리를 효율적으로 테스트하기 위해 다양한 알고리즘의 적용이 가능하고 다양한 고장 정보를 제공하는 IEEE 1500 표준 기반의 Programmable Memory BIST (PMBIST) 구조를 제안한다. IEEE 1500 표준을 사용하여 PMBIST를 제어하고, 다수의 메모리를 하나의 PMBIST로 테스트가 가능한 구조이다. 그리고 March 기반의 모든 알고리즘과 Galloping, Walking 등의 non-March 알고리즘 및 data retention 테스트를 지원하여 flexibility와 고장 검출률이 높고, 내장 메모리의 높은 신뢰성을 확보할 수 있다. 또한 repair와 고장 진단을 위한 고장 정보를 제공하여 높은 수율을 확보할 수 있다. 그리고 다양한 테스트 패턴을 생성하기 위한 최적의 프로그램 명령어 구조 및 IEEE 1500 표준 등을 통하여 최소의 하드웨어 오버헤드를 가진다.

Ⅱ. 제안하는 Programmable Memory BIST

본 논문은 기존에 제안하였던 BIST[7]를 바탕으로 다양한 테스트 알고리즘 지원이 가능한 micro code 방식의 프로그램 가능한 BIST (PMBIST) 구조를 제안한다. 제안하는 PMBIST는 외부로부터 IEEE 1500 표준을 이용하여 명령어로 프로그램된 알고리즘을 입력받아 내부의 프로그램 메모리에 저장하고, 이 프로그램 명령어를 사용하여 알고리즘을 구현한다. 그리고 IEEE 1500 래퍼를 이용하여 다수의 내장된 메모리를 하나의 PMBIST로 테스트가 가능하다. 또한 March 기반의 모든 알고리즘과 Galloping, Walking 등의 non-March 알고리즘 및 data retention 테스트를 효과적으로 지원하기 위한 프로그램 명령어 구조를 제안한다. March 기반의 테스트 알고리즘으로 검출할 수 있는 고장이 제한적이기 때문에 높은 신뢰성을 확보하기 위해서는 non-March 알고리즘 및 data retention 테스트 등을 지원할 수 있어야 한다. 제안하는 프로그램 명령어는 9 bits의 사이즈로 구성되며, March 및 non-March 테스트 알고리즘 등을 최소의 bit으로 구현할 수 있다.

제안하는 PMBIST는 repair 및 고장 진단을 위해 다양한 고장 정보를 제공하는 2개 모드를 지원하는 고장 정보 처리기를 제안한다. 고장 정보 처리기는 메모리 수리를 위한 고장 정보(FDR: fault data for repair) 모드와 메모리 고장 진단을 위한 고장 정보(FDD: fault data for diagnosis) 모드를 지원한다. FDR은 메모리를 테스트하여 고장이 검출되는 메모리 셀의 주소 정보를 제공한다. FDD는 고장 진단을 위해 자세한 고장 정보를 제공하는 모드이고, 고장 셀의 주소와 고장을 검출한 패턴의 정보를 제공한다. 고장 정보 처리기의 고장 정보는 IEEE 1500 표준을 이용하여 외부로 효과적으로 내보낸다.

제안하는 PMBIST는 모든 March 기반 알고리즘,

non-March 알고리즘 및 data retention 테스트 지원하여 높은 flexibility와 고장 검출률을 확보하였다. 제안된 프로그램 명령어 구조는 다양한 테스트 패턴을 최소의 bit으로 효과적으로 생성이 가능하여 하드웨어 오버헤드를 최소화하였다. 그리고 IEEE 1500 래퍼 사용하여 알고리즘을 구현한 프로그램 명령어를 효율적으로 입력하고, 메모리 테스트를 통해 얻은 고장 정보를 외부로 내보낸다. 또한 IEEE 1149.1 TAP을 사용하여 IEEE 1500 래퍼와 PMBIST를 효과적으로 제어한다.

1. 프로그램 명령어 구조

제안하는 프로그램 명령어는 9 bits의 사이즈로 March 및 non-March 테스트 알고리즘을 효과적으로 구현할 수 있는 최적의 구조를 가진다. 또한 data retention 테스트도 지원한다. 그리고 복잡한 패턴을 효과적으로 구현하기 위하여 branch 레지스터를 이용하여 멀티 루프를 지원하고, Reverse Data Rerun Branch (RDRB) 옵션을 사용하여 최소의 bit으로 알고리즘을 구현한다. 따라서 하드웨어 오버헤드를 최소화할 수 있다. 그림 1은 프로그램 명령어 구조를 보여주며, 각 bit을 간단히 살펴보면 다음과 같다.

Inst[8:7]은 명령어 제어 부분으로 프로그램 명령어의 동작 상태를 지장한다. Increment는 현재의 명령어를 실행한 후 다음 명령어를 실행하도록 한다. Branch는 branch 레지스터에 지정된 명령어로 점프하여 실행한다. RDRB는 branch 레지스터에 지정된 명령어로 점프를 하여 반전된 데이터 값으로 테스트 프로그램을 다시 실행하도록 한다. Pause는 data retention 고장을 검출하기 위해 외부로부터 제어되는 시간 동안 명령어를 붙잡고 대기하는 명령이다. 명령어에서 Branch와 RDRB는 branch 레지스터 값을 사용하여 점프를 한다. Inst[6]은 주소 증/감 제어 부분으로 March 알고리즘 및 non-March 알고리즘의 각 sequence의 주소 증/감 방향을 지정한다. Inst[5]는 백그라운드 데이터 제어 부분으로 백그라운드 데이터를 반전할 것인지, 반전하지 않을 것인지를 지정한다. Inst[4]는 메모리 동작 제어 부분으로 March 및 non-March 알고리즘 sequence 내의 데이터를 읽기/쓰기 동작을 지정한다. Inst[3]은 카운터 제어 부분으로 주소 생성을 위해 A 카운터와 B 카운터 중에 카운팅할 카운터를 선택한다. Inst[2:0]는 명령어 옵션 제어 부분으로 프로그램 명령어의 옵션을 지정한다. '#A', '#B' 옵션은 카운터를 증가시키지 않고 그대로 유지하라는 옵션이고, '+A', '+B' 옵션은 Inst[3]의 카운터 제어와 동시에 다른 카운터의 값을 증가시키는 옵션이다. 그리고 'A←B', 'B←A' 옵션은 카운터의 값을 이용하는 옵션이다.

프로그램 명령어 구조는 March 및 non-March 알고리즘을 구현하는데 최소의 명령어를 사용한다. 테스트 알고리즘에서 데이터 값을 반전하여 동일한 읽기와 쓰기 동작들이 연속적으로 사용되는 경우가 많다. 이런 경우에 제안하는 프로그램 명령어는 RDRB 옵션을 사용한 명령어로 구현할 수 있다. 예를 들어, ⇑(r1, w0, r0), ⇑(r0, w1, r1)와 같이 6개의 element로 구성된 알고리즘의 경우를 보면, 기존에 제안된 명령어들은 최소 8개의 명령어를 사용하여 구현한다. 하지만 제안하는

그림 2. Galloping 알고리즘의 프로그램 명령어 구현
Fig. 2. Program instruction implementation of Galloping algorithm.

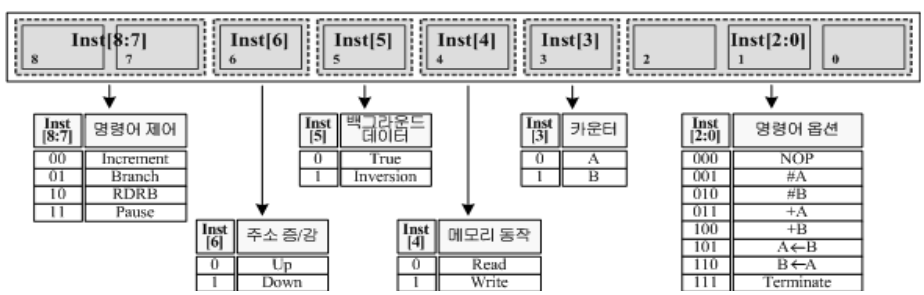


그림 1. 프로그램 명령어 구조
Fig. 1. Program instruction architecture.

프로그램 명령어를 사용하면, ⇑(r1, w0, r0)은 3개의 명령어로 구현을 하고, ⇑(r0, w1, r1)은 RDRB 옵션을 사용한 1개의 명령어로 구현한다. 따라서 기존의 명령어와 비교하여 2개의 명령어를 적게 사용한 총 4개의 명령어로 구현이 가능하다. 14개 element로 구성된 March C+ 알고리즘은 2개의 RDBranch 옵션을 사용하여 10개의 명령어로 구현된다. 따라서 제안하는 알고리즘 명령어는 최소의 명령어로 테스트 알고리즘을 구현할 수 있다. 그림 2는 non-March 알고리즘 중에 Galloping 알고리즘을 제안한 프로그램 명령어 7개를 사용하여 구현한 예제이다.

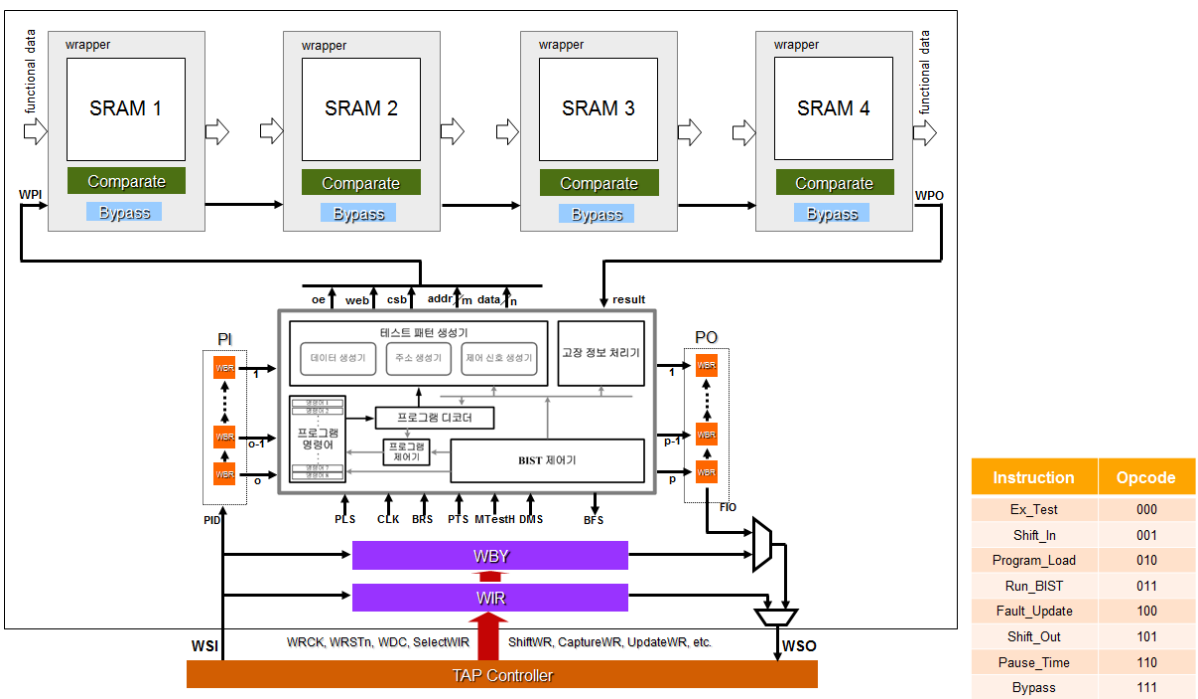
2. PMBIST 구조

제안하는 PMBIST는 알고리즘을 구현한 프로그램 명령어를 외부로부터 입력받아 패턴을 생성하는 구조이다. 외부로부터 입력받은 프로그램은 프로그램 메모리에 저장하며, 프로그램 메모리의 명령어를 사용하여 패턴을 생성한다.

그림 2는 제안하는 IEEE 1500 기반의 PMBIST 구조이다. 그림 2를 살펴보면, 크게 내장된 메모리를 테스트하는 PMBIST와 PMBIST를 제어하기 위한 IEEE 1500 wrapper로 구성된다. PMBIST는 TAP controller에서 Wrapper Instruction Register (WIR)의 명령어를 사용하여 제어한다.

그림 2의 PMBIST의 구성을 살펴보면, 알고리즘을 효과적으로 구현하기 위해 입력받은 프로그램을 저장하는 프로그램 메모리와 테스트 패턴을 생성하기 위한 프로그램 메모리를 제어하는 프로그램 제어기가 있다. 또한 프로그램 메모리의 프로그램 명령어를 해독하는 프로그램 디코더와 디코더의 신호를 받아 메모리 제어신호, 데이터 및 주소 패턴을 생성하는 테스트 패턴 생성기가 있다. 테스트 패턴 생성기는 제어신호 생성기, 데이터 생성기 및 주소 생성기로 구성된다. 그리고 테스트 결과 값을 비교하여 고장의 유무를 판별하는 비교기와 비교기의 결과를 사용하여 repair와 고장 진단을 위한 고장 정보를 생성하는 고장 정보 처리기가 있다. 마지막으로 외부로부터 테스트 알고리즘을 입력받는 프로그램 모드와 테스트 패턴을 생성하는 테스트 모드 및 고장 정보 생성 모드의 PMBIST 동작을 제어하는 BIST 제어기로 구성된다.

PMBIST는 프로그램 메모리에 프로그램 명령어를 최대 9개를 저장할 수 있다. 알고리즘을 구현하기 위해서는 다수의 명령어를 사용하지만, 패턴을 생성하는데 한번에 최대 9개 이상의 명령어를 사용하지 않기 때문



Instruction	Opcode
Ex_Test	000
Shift_In	001
Program_Load	010
Run_BIST	011
Fault_Update	100
Shift_Out	101
Pause_Time	110
Bypass	111

그림 3. IEEE 1500 기반 PMBIST 구조
Fig. 3. IEEE 1500 based the PMBIST architecture.

이다. March C+ 알고리즘(14N)을 보면, March C+는 6개의 march element로 구성된다. 6개의 march element를 각각 구현할 때, 3개 이상의 명령어가 필요한 march element는 없다. non-March 알고리즘도 한번에 8개 이상의 명령어를 사용하지 않는다. 따라서 프로그램 명령어의 사이즈를 최적화하여 하드웨어 오버헤드를 최소화하였다. 그리고 테스트 패턴 생성기에서 주소 생성기는 복잡한 주소 생성을 위해서 2개의 카운터(A, B 카운터)를 사용하여 이중 루프를 지원한다.

그림 2에서 PMBIST의 입출력 포트와 신호들을 간단히 살펴보면, CLK, BRS, MTestH, Pause Time Select(PTS), Program Loading Signal (PLS), Program Instruction Download(PID), Diagnostic information Mode Select(DMS), Fault Information Out(FIO), BIST Finish Signal(BFS) 등이 있다. PLS와 PID는 알고리즘을 구현한 프로그램 명령어를 입력받는 신호와 포트이다. PLS는 외부로부터 프로그램 명령어를 입력받아 명령어 메모리에 저장하는 모드를 지정하는 입력 신호이고, PID는 IEEE 1500의 PI를 통해 프로그램 명령어를 PMBIST로 입력받는 포트이다. MTestH는 PMBIST가 테스트 패턴을 생성하여 메모리를 테스트하는 테스트 모드를 시작하라는 입력 신호이다. Wrapper Parallel Input(WPI)로 테스트 패턴을 인가하여 테스트를 수행하며, Wrapper Parallel Output(WPO)를 통해 테스트 결과 값을 가져온다. PTS는 data retention 고장을 검출하기 위해 외부로부터 pause 시간을 입력 받는 포트이다. 프로그램 명령어에서 Pause 명령이 실행되고, 외부로부터 PTS 신호가 들어오는 동안에는 명령어 실행이 정지되고 PTS 신호가 끝나면 다음 명령어를 실행한다. 그리고 DMS와 FIO는 테스트 결과를 사용하여 고장 정보를 생성하는 모드를 선택하고 출력하는 신호와 포트이다. DMS 입력에 의해 고장 정보의 종류를 선택하고, FIO는 PMBIST의 테스트 결과인 고장 정보를 IEEE 1500의 PO를 통해 외부로 출력한다. BFS은 PMBIST의 동작이 끝났음을 알려주는 출력 신호이다.

그림 4는 고장 정보 처리기를 보여준다. 고장 정보 처리기는 비교기의 테스트 결과와 테스트 패턴 생성기의 주소 생성기 그리고 프로그램 제어기의 정보를 사용하여 고장 정보를 생성한다. 고장 정보 처리기는 메모리 수리를 위한 고장 정보(FDR: fault data for repair)와 메모리 고장 진단을 위한 고장 정보(FDD: fault data for diagnosis)로 2개 타입의 모드를 지원한다. FDR은 redundancy analysis를 위해 메모리의 고장 주소 정보를 제공하는 모드이다. FDD는 고장 진단을 위하여 고장 데이터 정보와 고장 주소 및 고장을 검출한 패턴의 정보를 제공하는 모드이다. 그림 5는 FDR과 FDD의 고장 정보의 포맷을 보여준다.

IEEE 1500 래퍼는 크게 래퍼 명령 레지스터(WIR: Wrapper Instruction Register), 래퍼 바이패스 레지스터(WBY: Wrapper Bypass Register) 및 래퍼 경계 레지스터(WBR: Wrapper Boundary Register)로 구성된다. 그리고 IEEE 1149.1의 TAP 제어기와 같은 TAP 제어기를 사용하여 다양한 테스트 명령어를 정의해 IEEE 1500 기반 PMBIST를 제어한다.

IEEE 1500 기반 PMBIST는 알고리즘을 구현한 프로그램 명령어를 래퍼 직렬 입력(WSI: Wrapper Serial Input)과 WBR의 직렬 및 병렬 도메인을 사용하여 PMBIST에 효과적으로 입력하고, BIST를 동작을 제어하여 내장된 메모리의 테스트를 수행하게 한다. 그리고 래퍼 직렬 출력(WSO: Wrapper Serial Output)과 WBR의 직렬 및 병렬 도메인을 사용하여 고장 정보를 외부로 출력한다. 그리고 TAP 제어기에서 WIR의 명령어를 사용하여 PMBIST를 제어한다.

WIR의 Shift_In과 Program_Load 명령을 사용하여 WSI으로 WBR에 알고리즘을 구현한 프로그램을 업데

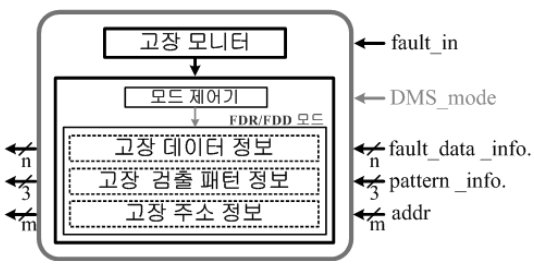


그림 4. 고장 정보 처리기
Fig. 4. The fault information processing module.

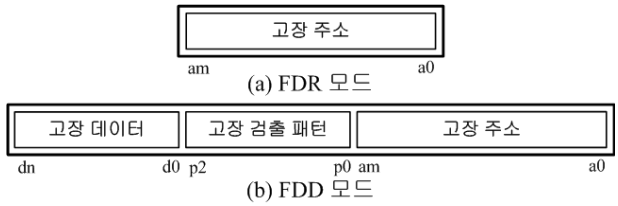


그림 5. 고장 정보 포멧
Fig. 5. The fault information format.

이트하고, PMBIST로 parallel하게 입력을 한다. Run_BIST 명령으로 PMBIST를 실행하여 내장 메모리를 테스트한다. 그리고 PMBIST의 테스트 수행이 끝나면 Fault_Update와 Shift_Out 명령을 사용하여 PMBIST로부터 WBR에 고장 정보를 업데이트하여 WSO으로 출력한다. Pause_Time는 data retention 고장을 검출할 때 사용하는 명령이고, Bypass는 다수의 메모리를 테스트할 때 사용하는 명령이다.

Ⅲ. 검증 및 성능평가

본 논문에서 제안하는 PMBIST 구조 검증을 위하여 단일 포트 메모리(16.3K×16)를 사용하여 검증을 하였다. 7 bits 열 주소와 7 bits 행 주소를 가지며, 16 bits 데이터 워드를 가지는 단일 포트 SRAM을 사용하였다. PMBIST의 검증을 위해 March C-(10N), March SS (22N) 알고리즘[8]과 non-March 알고리즘인 Galloping 알고리즘을 Mentor Graphics의 Modelsim을 이용하여 functional 시뮬레이션을 통하여 동작 검증하였다. 표 1은 제안한 9 bits의 프로그램 명령어를 사용하여 알고리즘을 구현하는데 필요한 명령어 개수와 명령어 bit를 보여준다. 제안하는 알고리즘 명령어는 최소의 명령어로 테스트 알고리즘을 구현할 수 있는 최적의 구조이다. 따라서 March C-와 March SS 알고리즘은 8개와 14개의 명령어를 사용하여 72 bits과 126bits의 명령어로 구현된다. 또한 Galloping 알고리즘은 7개의 명령어로 구현이 가능하여 최소의 bit을 사용하여 구현된다.

표 1. 알고리즘별 명령어 개수 및 bit
Table 1. Instruction number and bit sizes for the algorithms.

알고리즘	명령어 개수	명령어 bit
March C- (10N)	8	72 bits(8 × 9 bits)
March SS (22N)	14	126 bits(14 × 9 bits)
Galloping	7	63 bits(7 × 9 bits)

또한 Synopsys의 Design Compiler를 사용해 PMBIST 구조를 합성하여 하드웨어 오버헤드를 검증하였다. 합성에는 TSMC 0.13μm 공정 라이브러리를 사용하였다. PMBIST를 합성한 결과, 2-input nand gate를 기준으로 하드웨어 오버헤드는 5,083 gates이고, 최대 동작 속도는 약 300MHz이다.

표 2는 기존의 PMBIST와 제안하는 PMBIST 구조를 비교한 것이다. 기존의 PMBIST와 제안하는 PMBIST 구조를 지원하는 테스트 알고리즘, 명령어 사이즈, March C- 알고리즘과 Galloping 알고리즘을 구현하는데 필요한 명령어 bit 그리고 하드웨어 오버헤드 등으로 비교하였다. 또한 repair 및 고장 진단을 위한 고장 정보를 지원하는지 비교하였다.

[9]는 IEEE 1500을 사용하는 PMBIST이다. 이 구조는 11개의 processor instruction set을 4bits의 명령어 코드로 지정하여 알고리즘을 구현하며, 백그라운드 데이터를 지정하는데 사용하는 명령, 주소를 지정하는 명령 및 루프를 위해 특정한 주소로 점프를 지정하는 명령 등으로 구성된다. 그리고 March 알고리즘들만 지원

표 2. PMBIST 성능 비교
Table 2. Performance comparison of the PMBIST.

	[9]	[10]	[11]	PMBIST
March based 테스트 알고리즘	Y	Y	Y	Y
Non-March 테스트 알고리즘	N	Y	Y	Y
Data Retention Test	N	N	N	Y
Flexibility	Low	Medium	Medium-High	High
Programmability	Medium	Medium-High	Medium-High	High
Multi-loop	N	Y	Y	Y
고장 정보 처리 모듈	N	Y	N	Y
고장 정보 분석 효율	-	Medium	-	High
명령어 사이즈 (bits)	4	8	9	9
March C- 알고리즘의 명령어 bit (bits)	128	152	90	72
Galloping 알고리즘의 명령어 bit (bits)	-	-	108	63
하드웨어 오버헤드 (gates)	7.9K	13.6K	6.4K	5.1K

할 수 있어 제한적인 flexibility를 가진다.

[10]와 [11]은 micro-code 방식의 PMBIST로 모든 March 알고리즘과 non-March 알고리즘의 일부를 지원하며, multi-loop를 지원하여 복잡한 패턴을 생성한다. 하지만 data retention 테스트는 지원하지 못한다. [10]는 March C- 알고리즘을 구현하는데 152 bits의 많은 명령어가 필요하며, 하드웨어 오버헤드가 매우 크다는 단점을 가진다. 그리고 고장 정보를 제공하는 1.2K gates의 진단 로직을 포함하고 있지만, 고장 메모리의 셀 주소와 fail-map 데이터만을 고장 정보로 제공하여 정확한 고장 분석이 불가능하다. [11]은 March C- 알고리즘을 구현하는데 90 bits의 명령어가 필요하지만, 고장 정보를 지원하는 고장 진단 로직이 없고 고장의 유무만을 제공한다.

제안하는 IEEE 1500 기반 PMBIST는 다양한 테스트 알고리즘을 지원하여 높은 고장 검출률을 가지며, RDRB 옵션 등으로 최적화된 프로그램 명령어를 사용하여 최소의 bit으로 알고리즘을 구현할 수 있다. March C- 알고리즘은 8개의 프로그램 명령어로 구현이 가능하여 총 72 bits (8×9bits)가 필요하다. 이것은 기존의 구조들보다 18~80bits 적은 명령어로 구현된다. PMBIST는 repair와 고장 진단을 위해 FDR과 FDD 모드를 통해 다양한 고장 정보를 지원하는 효율적인 고장 정보 처리기를 가지고 있다. 고장 정보 처리기는 다양한 고장 정보를 IEEE 1500의 인터페이스를 사용해 외부의 ATE에 효과적으로 제공하여 높은 메모리 수율을 보장한다. IEEE 1500 래퍼 사용하여 알고리즘을 구현한 프로그램 명령어를 효율적으로 입력하고, 메모리 테스트를 통해 얻은 고장 정보를 외부로 내보낸다. 또한 하드웨어 오버헤드도 기존에 제안된 구조들보다 1.3K~8.5K gates의 차이를 보인다. 따라서 제안하는 PMBIST는 최적의 프로그램 명령어로 다양한 테스트 알고리즘을 지원하여 programmability와 flexibility가 높고, 최소의 하드웨어 오버헤드를 가진다.

Ⅳ. 결 론

최근에 SoC 환경이 급속히 늘어 가면서 상당 부분의 비중을 차지하고 있는 내장된 메모리의 테스트에 대한 많은 연구가 진행되고 있다. IEEE 1500 기반 PMBIST는 다양한 테스트 알고리즘을 지원하여 높은 고장 검출률을 가지며, 프로그램 명령어를 이용하여 사용자가 사용자에 의해 정의되는 알고리즘의 프로그램이 가능해 높은 flexibility와 programmability를 가진다. 또한 고장 정보 처리기는 repair와 고장 진단을 위한 다양한 고장 정보를 효과적으로 제공하여 높은 메모리 수율을 보장한다. 또한 IEEE 1500 래퍼를 사용하여 프로그램 명령어의 입력과 고장 정보의 출력을 효율적으로 수행할 수 있다. 그리고 PMBIST의 프로그램 메모리를 최소화하고, 최적의 프로그램 명령어로 최소의 하드웨어 오버헤드를 가지는 효과적인 구조이다.

참 고 문 헌

[1] A. van de Goor, C. Jung, S. Hamdioui, and H. Kukner, “Generic, Orthogonal and Low-cost March Element based Memory BIST,” Proceeding of IEEE ITC, pp. 1-10, 2011.

[2] W. L. Wang, K. J. Lee, and J. F. Wang, “An on-chip march pattern generator for testing embedded memory cores,” IEEE Transactions on Very Large Scale Integration Systems, vol 9, Issue 5, pp. 730-735, 2001.

[3] Yamasaki, I. Suzuki, A. Kobayashi, K. Horie, Y. Kobayashi, H. Aoki, H. Hayashi, K. Tada, K. Tsutsumida, and K. Higeta, “External memory BIST for system-in-package,” Proceeding of International Test Conference, pp. 1145-1154, Nov. 2005.

[4] A. W. Hakmi, H. J. Wunderlich, C. G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel, and L. Souef, “Programmable deterministic Built-In Self-Test,” Proceeding of IEEE International Test Conference, pp. 1-9, Oct. 2007.

[5] IEEE Computer Society, “IEEE Standard Test Access Port and Boundary-Scan Architecture,” IEEE Standards Board, Feb. 1990.

[6] IEEE Computer Society, “IEEE Standard Testability Method for Embedded Core-based Integrated Circuits,” IEEE Standards Board, Aug. 2005.

[7] Y. Park, Y. Lee, I. Choi, and S. Kang, “IEEE std. 1500 based Programmable Memory Built-In Self-Test(BIST) for Embedded Memory in SoC,” Proceeding of Korea Test Conference, pp. C-3, Jun. 2012.

[8] S. Hamdioui, A. J. Van de Goor, and M. Rodgers, “March SS: a test for all static simple RAM

faults," Proceedings of IEEE International Workshop on Memory Technology, Design and Testing, pp. 95-100, Jul. 2002.
[9] D. Appello, P. Bernardi, A. Fudoli, M. Rebaudengo, M.S. Reorda, V. Tancorre, and M. Violante, "Exploiting Programmable BIST for the Diagnosis of Embedded Memory Cores," Proceeding of IEEE International Test Conference, pp. 379-385, Oct. 2003.
[10] X. Du, N. Mukherjee, C. Hill, W-T. Cheng, and S. Reddy, "A Field Programmable Memory BIST Architecture Supporting Algorithms with Multiple Nested Loops," Proceeding of IEEE Asian Test Symposium, pp. 287-292, Nov. 2006.
[11] Y. Park, J. Park, T. Han, and S. Kang, "An Effective Programmable Memory BIST for Embedded Memory," IEICE Transactions on Information and Systems, vol. E92-D, no. 12, pp. 2508-2511, Dec. 2009.

저 자 소 개

박 영 규(학생회원)
2004년 호서대학교 전기공학과 학사 졸업.
2007년 연세대학교 전기전자 공학과 석사 졸업.
2007년 현재 연세대학교 전기전자 공학과 박사 과정.
<주관심분야 : Memory test, BIST, DFT>

최 인 혁(학생회원)
2009년 연세대학교 전기공학과 학사 졸업.
2009년 현재 연세대학교 전기전자 공학과 석박사 통합 과정.
<주관심분야 : SoC 설계, DFT, DFD>

강 성 호(평생회원)
1986년 서울대학교 제어계측공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터공학과 박사 졸업.
1992년 미국 Schlumberger Inc. 연구원.
1994년 Motorola Inc. 선임연구원.
2007년 현재 연세대학교 전기전자공학과 교수
<주관심분야 : SoC 설계, SoC 테스트>