SoC 내장 메모리를 위한 ARM 프로세서 기반의 프로그래머블 BIST

(ARM Processor-based programmable BIST for Embedded Memory in SoC)

이 민 호 † 홍 원 기 † 송 좌 희 † 장 훈 ††
(Min-ho Lee) (Won-gi Hong) (Jwa-hee Song) (Hoon Chang)

요 약 메모리 기술이 발달함에 따라 메모리의 집적도가 증가하게 되었고, 그에 따라 구성요소들의 크기가 작아지게 되고, 고장의 감응성이 증가하게 되어, 테스트는 더욱 복잡하게 된다. 또한, 칩 하나에 포함되어 있는 저장요소가 늘어남에 따라 테스트 시간도 증가하게 된다. SoC 기술의 발달로 대용량의 내장 메모리를 통합할 수 있게 되었지만, 테스트 과정은 복잡하게 되어 외부 테스트 환경에서는 내장 메모리를 테스트하기 어렵게 되었다.

본 논문은 ARM 프로세서 기반의 SoC 환경에서의 임베디드 메모리를 테스트할 수 있는 프로그램 가능한 메모리 내장 자체 테스트를 제안한다.

키워드 : BIST, Embedded Memory, Programmable BIST, ARM, AMBA

Abstract The density of Memory has been increased by great challenge for memory technology; therefore, elements cf 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.

We present a ARM processor-programmable built-in self-test (BIST) scheme suitable for embedded memory testing in the SoC environment. The proposed BIST circuit can be programmed vis an on-chip microprocessor.

Key words : BIST, Embedded Memory, Programmable BIST, ARM, AMBA

1. 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, 더욱 복잡하게 되었기 때문이다. 두 번째는 칩 하나에 포함되어 있는 저장요소가 늘어남에 따라 테스트 시간도 증가하게 되었기 때문이다. 그래서 트랜지스터 하나를 생산하는 비용보다 테스트하는데 소비되는 비용이 더 크게 증가 하였다.

최근 급속한 SoC(System On-Chip)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 임베디드 메모리

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 학생회원 : 숭실대학교 컴퓨터학과
mhlee@watt.ssu.ac.kr
wghong@watt.ssu.ac.kr
jhsong@watt.ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학부 교수
hoon@ssu.ac.kr

논문접수 : 2007년 11월 27일
심사완료 : 2008년 3월 10일

정보과학회논문지: 시스템 및 이론 제35권 제6호(2008.6)

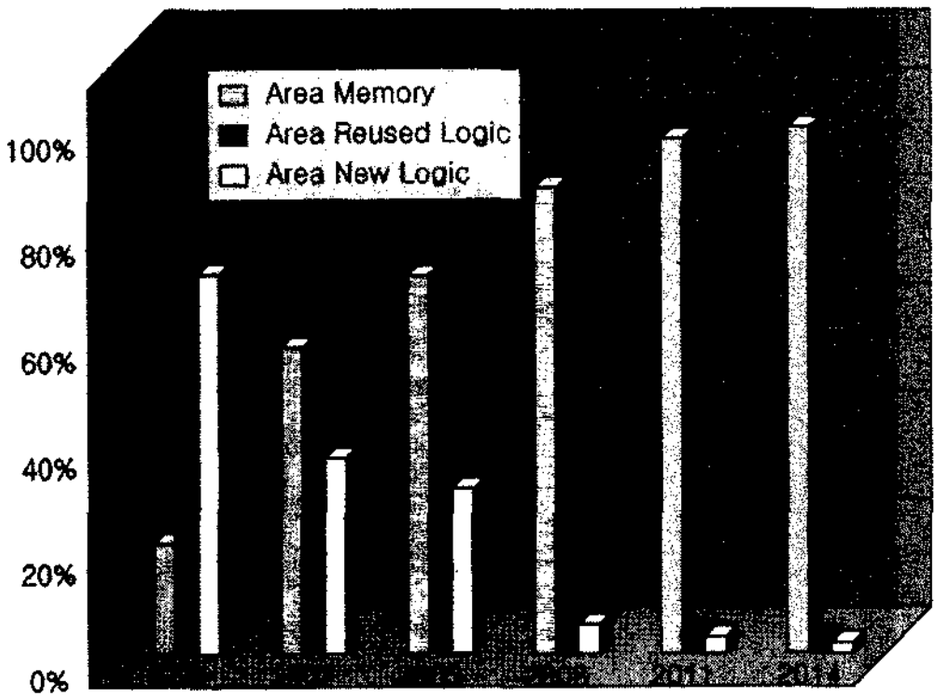


그림 1 Embedded Memory Usage

가 점점 많은 영역을 차지해 가고 있다는 것을 알 수 있다. 이는 SoC 칩을 테스트하는 과정에서의 시간과 비용을 줄이기 위해서는 내장 메모리를 테스트하는 과정이 가장 큰 영향을 미친다는 것을 뜻한다.

내장 메모리의 용량은 커져가는 반면에, 크기는 점점 줄어듦에 따라 테스트 과정은 복잡하게 되어 많은 시간과 복잡한 알고리즘이 필요하게 되었다. 따라서 ATE (Automatic Test Equipment)를 이용하여 외부에서 메모리를 접근하여 테스트하는 방법은 매우 오랜 테스트 시간을 필요로 하고, At-speed 테스트가 불가능하게 되었다.

내장 자체 테스트(BIST)는 기존의 테스트 문제점들을 해결하는 방법을 제시한다. 메모리 내장 자체 테스트(MBIST)는 테스트 하에 있는 칩을 위한 메모리 테스트 알고리즘을 구현한다[3-6]. 메모리 내장 자체 테스트는 SoC 안에 내장되어 있으므로, 외부로부터 많은 수의 포트를 필요로 하지 않는다. 그러나 이러한 테스트 구조는 이미 정해진 하나의 테스트 알고리즘만을 적용가능하다. 하나의 알고리즘만 사용하여 테스트 하는 방법은 다음과 같은 이유에서 이러한 방법은 적합하지 않다. 첫 번째, 이유로 메모리 장치는 메모리 생산하는 과정을 반복하는 동안 필요한 테스트 알고리즘이 변하게 된다. 생산 초기에는 비록 복잡하고 시간이 오래 걸리지만, 다양한 고장을 찾아내는 알고리즘이 필요하지만, 생산이 거듭될수록 생산 공정이 안정화되고 수율이 높고 빠른 알고리즘으로 더 간단히 충분한 테스트가 가능하다. 두 번째, 이유는 SoC는 다양한 종류와 크기의 메모리를 갖기 때문에 각 메모리의 종류와 크기에 따른 적합한 테스트 알고리즘이 필요하다. 메모리 생산 공정의 수율과 메모리 블록에 따른 적합한 알고리즘들을 모두 지원하기 위해서는 알고리즘마다 각각 구현회로를 설계해서 내장하여야 하므로 오버헤드가 증가하게 된다.

반면에, 다양한 테스트 알고리즘을 지원하는 프로그램 가능한 메모리 내장 자체 테스트(Programmable Memory Built-in Self Test)는 테스트 환경(크기, 종류 등)에 적합한 테스트 알고리즘을 선택 가능하므로, 높은 효율성을 가지고 있다. 많은 프로그램 가능한 메모리 내장 자체 테스트가 개발되었다[7]. 프로그램 가능한 메모리 내장 자체 테스트는 크게 두 가지 접근 방법으로 나뉜다. 첫 번째는 마이크로 코드를 이용한 방법이고 다른 하나는 유한 상태머신(FSM)을 이용한 접근방법이다. 일반적으로 마이크로 코드를 이용한 방법이 다양한 알고리즘의 적용이 쉽기 때문에 많이 사용된다. 마이크로 코드를 이용하여 테스트 알고리즘을 적용하기 위해서는 LDA, STA, LDX, STA 등의 명령어들의 반복으로 테스트를 한다. 하지만 이 방법은 명령어의 주소 지정방식과 각 명령어의 처리시간에 따라 많은 영향을 받는다. FSM방식은 속도가 빠르지만 알고리즘을 선택하기 위한 코드를 ATE를 이용하여 입력해야 한다.

본 논문에서는 매크로 코드에 의해 동작하는 FSM방식과 ARM 마이크로 코드 방식의 내장 자체 테스트 구조를 제안한다. 이 구조는 ARM 마이크로 코드를 이용하여 테스트 Elements와 메모리의 크기 등을 설정해 주고 FSM 방식의 BIST로 테스트를 수행하게 된다. 제안하는 구조에서는 8개의 레지스터와 세 가지 로직회로가 존재한다. 8개의 레지스터는 ARM 코어에서 보내오는 테스트를 위한 설정들을 저장한다. 로직회로는 Elements에 따른 메모리 주소를 생성해 주는 Address Generator Logic(AGL)과, 메모리에서 읽어온 값과 정상 동작일 경우의 값을 비교하는 Data Comparator Logic, 메모리의 읽기/쓰기 동작에 적합한 신호와 AGL 회로를 제어하는 Test Controller Logic로 구성되어 진다.

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

2. 기존 연구

2.1 고장 모델링

테스트 비용과 시간을 줄이기 위해서는 정확한 고장 모델링이 필요하다. 테스트의 고장 검출률은 테스트에 사용되는 고장 모델링에 의해 결정되어 진다. 1980년대에 많은 메모리 고장 모델이 소개 되었다. Address Decoder Faults, Stuck-At Faults가 모델링 되었지만, 고장을 완벽에게 반영할 수는 없었다. 그 후, 실제 디자인에서의 실제 고장을 반영하기 위해서 물리적 레이아웃

수준의 실험 결과를 바탕으로 State-Coupling Fault, Data-Retention Fault 고장 모델이 성립되었다. 메모리의 집적도가 커지면서 새로운 고장모델링이 필요하게 되었다. 새롭게 소개된 고장 모델은 Read Destructive Coupling fault, Write Disturb Fault, Transition Coupling Fault, Incorrect Read Fault 등이다[8]. 새로운 고장이 모델링되었기 때문에, 새롭게 추가된 고장을 검출 할 수 있는 알고리즘의 필요하게 되었다.

2.2 March 테스트 알고리즘

메모리를 테스트하기 위해서는 특정한 순서로 읽기와 쓰기 동작을 수행하여야 한다. 읽기 동작과 쓰기 동작의 횟수와 순서는 찾아내고자 하는 고장의 종류에 따라 달라진다. 가장 많이 사용하는 알고리즘은 March 테스트이다. March 테스트 알고리즘은 여러 가지의 March Elements로 구성되어 있다. March Elements의 구성요소는 메모리 셀에 적용되는 값, 읽기/쓰기 동작, 주소의 가감 방향으로 구성되어 있다. 표 1은 March Element의 구성요소를 보여준다. 예를 들어, ⇑(w0,r0)은 메모리 셀에 0을 쓰는 동작을 하고 0을 읽어 온 후, 주소가 증가하는 방향으로 다음 주소에 동일한 읽기/쓰기 동작을 반복 수행하라는 March Element이다. ⇳(r1)은 1을 읽어온 후, 주소 증감 방향에 관계없이 일정한 방향으로 읽기 동작의 수행을 의미한다.

표 2는 March Elements를 사용한 다양한 테스트 알고리즘을 보여준다. 각 알고리즘은 각기 다른 March elements로 구성되어져 있기 때문에, 알고리즘에 따라 검출 가능한 고장 모델들이 다르고, 고장 검출율도 다르다. 고장 검출율에 영향을 미치는 것은 알고리즘뿐만 아니라 고장 검출을 위한 배경데이타의 패턴에 의해서도

표 1 March Elements 구성 요소

기 호	동 작
r	읽기 동작
w	쓰기 동작
⇑	주소 증가
⇓	주소 감소
⇳	주소 증감 모두 가능

표 4 테스트 알고리즘의 검출률(%) : MATS+(a); March X(b); March C-(c)

Fault	P_1	P_2	P_3	$P_{2,3}$	$P_{1,2,3}$	P_{all}
SAF	100.0	100.0	100.0	100.0	100.0	100.0
SOF	100.0	100.0	100.0	100.0	100.0	100.0
TF	100.0	100.0	100.0	100.0	100.0	100.0
AF	99.7	99.9	99.9	100.0	100.0	100.0
CFin	100.0	100.0	100.0	100.0	100.0	100.0
CFid	37.5	37.5	37.5	62.6	75.9	89.1
CFst	50.0	50.0	50.0	75.0	87.5	100.0
(a)						
SAF	100.0	100.0	100.0	100.0	100.0	100.0
SOF	0.8	0.8	0.8	0.8	0.8	0.8
TF	100.0	100.0	100.0	100.0	100.0	100.0
AF	99.7	99.9	99.9	100.0	100.0	100.0
CFin	100.0	100.0	100.0	100.0	100.0	100.0
CFid	50.0	50.0	50.0	78.1	90.7	100.0
CFst	62.5	62.5	62.5	84.4	93.0	100.0
(b)						
SAF	100.0	100.0	100.0	100.0	100.0	100.0
SOF	0.8	0.8	0.8	0.8	0.8	0.8
TF	100.0	100.0	100.0	100.0	100.0	100.0
AF	99.7	99.9	99.9	100.0	100.0	100.0
CFin	100.0	100.0	100.0	100.0	100.0	100.0
CFid	99.9	99.9	99.9	99.59	100.0	100.0
CFst	99.9	99.9	99.9	99.59	100.0	100.0
(c)						

영향을 받는다. 표 3은 4-bit words의 배경데이타를 보여준다[7]. 4-bit의 배경데이타는 8-bit, 16-bit, 32- bit로 확장이 가능하다. 예를 들어서, 8-bit words의 P2의 배경데이타는 '01010101'로 확장될 수 있다.

표 4는 위에서 제시한 테스트 알고리즘과 배경데이타를 사용하였을 때의 고장 검출율을 보여준다[7].

3. 제안하는 프로그램 가능한 내장 자체 테스트

3.1 SoC 구성과 내장 자체 테스트

표 2 March Elements를 사용한 다양한 알고리즘

Algorithm	March Element
Zero-One	{⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)}
MATS+	{⇑(w0); ⇑(r0,w1); ⇓(r1,w0)}
March X	{⇳(w0); ⇑(r0,w1); ⇓(r1,w0); ⇳(r0) }
March C-	{⇕(w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0), ⇳(r0)}

표 3 4-bit words 배경 데이타

Patterns	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Data	0000	0101	0011	0110	0001	0010	0100	1000

SoC의 구성은 microcontroller core, embedded SRAM, I/O buffer로 되어 있다. microcontroller core의 구성은 CPU, DMAC, embedded SRAM controller, external memory controller, peripheral circuit interface로 구성되었다. 제안하는 내장 자체 테스트는 AMBA Bus에 위치한다. 그림 2는 SoC의 일반적인 구성과 SoC 안에서의 제안하는 내장 자체 테스트의 위치를 보여준다.

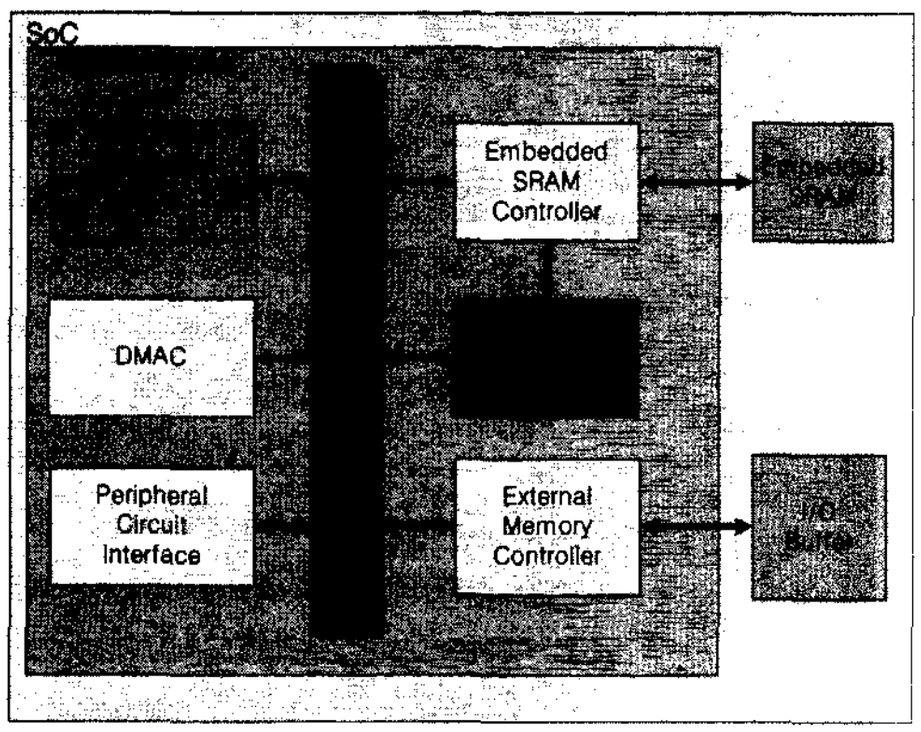


그림 2 SoC 구성과 BIST 위치

3.2 선택 가능한 테스트 알고리즘

제안하는 프로그램 가능한 메모리 내장 자체 테스트에서는 표 2에서와 같이 4개의 알고리즘을 지원한다. 메모리 생산 공정 초기에는 많은 고장이 발생하게 되므로, March C- 알고리즘을 지원하고, 생산이 거듭될수록 공정은 안정화 되므로, 단순하고 빠른 MATS+와 같은 알고리즘을 지원한다. 알고리즘의 설정은 R_{ME}를 이용하여 설정한다. 그림 3은 March Element Register의 구조를 보여준다.

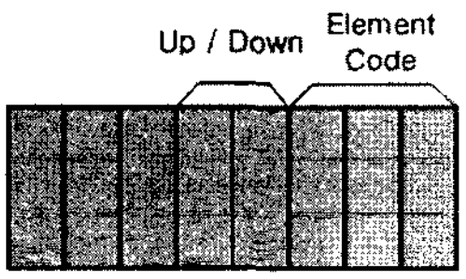


그림 3 March Element Register 구조

표 5 테스트 주소 설정표

Address_direction	Code[4:3]	Operation
Init	00	00000000
UP	01	⇑
DOWN	10	⇓

R_{ME}의 March Element의 선택은 하위 3비트로 되어 있다. 3번 비트와 4번 비트는 적용할 March Element의 주소 증감을 선택한다. 표 5는 테스트 주소의 방향을 설정하기 위한 R_{ME}의 3bit와 4bit의 설정표이다.

주소를 증가시키면서 테스트를 하여야 하거나, 주소의 증감이 자유로운 March Element의 경우는 '01'로 설정하고, 반대로 주소를 감소시키면서 적용해야 하는 경우에는 '10'을 설정하고, March Element가 끝나고 주소를 초기화하기 위한 경우에는 '00'을 할당하였다.

표 6은 March Element를 테스트 알고리즘에 맞게 설정하기 위한 설정표이다. R_{ME}의 하위 3비트는 전체 알고리즘의 March Element들의 정리하여 주소의 증감 여부를 제외한 읽기/쓰기 동작이 동일한 것들을 묶어 같은 번호를 부여하였다.

표 7은 테스트 알고리즘들을 주소 설정표와 읽기/쓰기 설정표를 이용하여 알고리즘들을 코드화한 것이다.

하나의 예를 들어 설명하면, 모든 알고리즘은 ⇳(w0)로 시작하게 되어 있다. 그러므로, 주소의 증감이 자유롭기 때문에 주소 설정비트는 '01'로 할당하고, (w0)를 의미하는 '001'을 할당하였다. 즉 모든 알고리즘의 총 March Element 수는 17개이지만, 동일한 March Element에는 같은 번호를 부여하였기에, 코드로 변환된 6

표 6 읽기/쓰기 설정표

March element	Code[2:0]	Operation Memory
M1	001	w0
M2	010	r0
M3	011	w1
M4	100	r1
M5	101	r0w1
M6	110	r1w0

표 7 알고리즘 코드 변환표

Algorithm	March Element
Zero-One	{⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)}
	01001 01010 01011 01100
MATS+	{⇑(w0); ⇑(r0,w1); ⇓(r1,w0)}
	01001 01101 10110
March X	{⇳(w0); ⇑(r0,w1); ⇓(r1,w0); ⇳(r0) }
	01001 01101 10110 01010
March C-	{⇳(w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0), ⇳(r0)}
	01001 01101 01110 10101 10110 01010

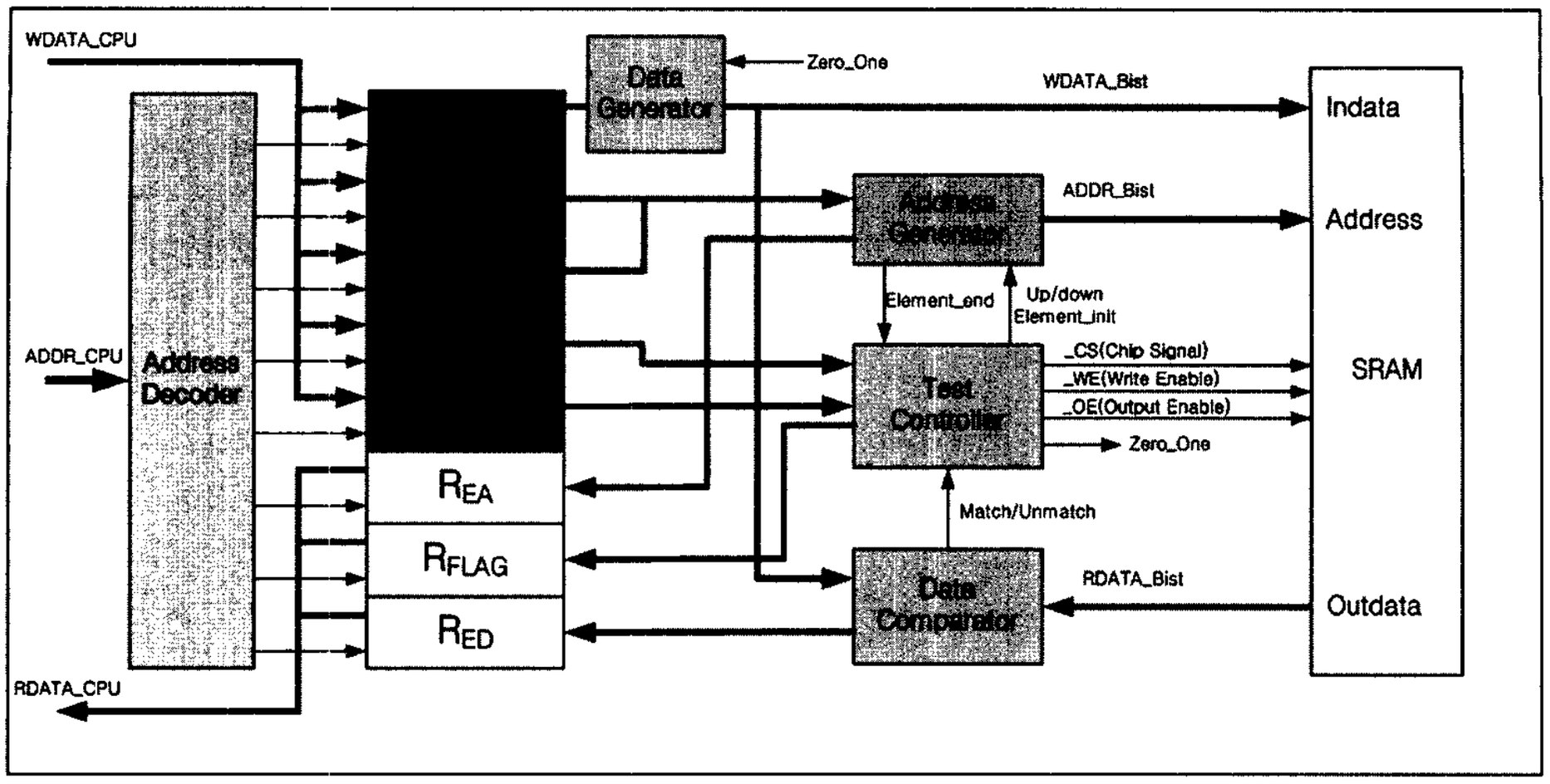


그림 4 제안하는 프로그램 가능한 내장 자체 테스트 구조

개로 제안한 모든 알고리즘을 표현할 수 있게 되었다.

3.3 제안하는 프로그램 가능한 내장 자체 테스트의 구조

제안하는 프로그램 가능한 내장 자체 테스트는 그림 4와 같은 구조로 되어 있다. ARM 코어에서 마이크로 코드를 이용해서 설정값을 원하는 레지스터에 넣기 위한 주소 해독기(Address Decoder) 회로, 원하는 테스트 알고리즘 설정과 고장시 고장주소와 데이타를 저장하기 위한 8개의 레지스터(R_{BG}, R_{AL}, R_{AH}, R_{ME}, R_{IR}, R_{EA}, R_{FLAG}, R_{ED}), 테스트 알고리즘에 따른 주소를 발생시키는 주소 생성 회로(Address Generator Logic, AGL), 내장 자체 테스트의 모든 모듈을 제어하는 테스트 제어 회로(Test Controller Logic), 마지막으로 메모리에서 읽어온 값을 정상 동작일 경우의 값과 비교하는 비교기 회로(Data Comparator Logic)로 구성되어 진다.

프로그램 가능한 메모리 내장 자체 테스트의 전체 흐름은 다음과 같다. ARM 코어의 마이크로 코드를 이용하여 설정 레지스터에 설정값을 넣어주고, 내장 자체 테스트를 동작 시키면 테스트 제어 모듈에서 March Elements를 분석하여서 주소 방향을 설정하여 주소 생성 회로로 보내고, 읽기/쓰기 동작의 신호를 발생시킨다.

각 회로에 대해 자세히 살펴보면 다음과 같다.

3.3.1 설정 레지스터

설정 레지스터는 테스트 할 메모리의 크기와 배경데이타, 적용 알고리즘 등을 마이크로 코드로 설정하기 위해 사용한다. 고장시에는 고장주소와 고장주소의 데이타 값을 저장한다. 표 8은 각각 레지스터의 역할을 보여 준다. R_{BG}는 테스트에 사용할 배경데이타를 설정하는 레지스터이고, R_{AL}와 R_{AH}는 테스트 할 메모리의 최하위 주소와 최상위 주소를 설정하는 레지스터이다. 그리고 이 두 개의 레지스터를 이용하여 테스트 할 메모리의 영역을 설정한다. R_{ME}는 적용할 알고리즘의 March Elements를 설정하는 레지스터이고 R_{IR}는 내장 자체 테스트의 동작 상태를 설정하는 레지스터이다. 또, R_{FLAG}는 내장 자체 테스트에서 현재 실행중인 Element의 실행 상태와 고장 검출 상태를 표시하며 R_{ED}는 고장이 검출되었을 때 고장난 주소를 저장하는 레지스터이다. 마지막으로 R_{EA}는 고장이 검출되었을 때 고장난 주소의 데이타를 저장하는 레지스터이다.

3.3.2 주소 해독기(Address Decoder)

주소 해독기는 메모리 테스트를 위해서 ARM 코어의

표 8 설정 레지스터의 역할과 설정 레지스터와 특정주소의 매핑

Register	Function	Address
R_{BG}	store background data	FFFFFFF0
R_{AL}	store lowest address	FFFFFFF1
R_{AH}	store highest address	FFFFFFF2
R_{ME}	store current March element	FFFFFFF3
R_{IR}	instruction register BIST circuit	FFFFFFF4
R_{EA}	address of defective memory cell	FFFFFFF5
R_{FLAG}	status register of BIST circuit	FFFFFFF6
R_{ED}	erroneous response of defective memory cell	FFFFFFF7

마이크로 코드를 사용하여 설정 레지스터를 설정하여야 하는데, 이때 설정 레지스터에 접근하기 위한 특정한 주소를 매핑을 하도록 한다. 표 8은 레지스터와 특정주소와의 매핑을 보여준다.

3.3.3 주소 생성 회로(Address Generator Logic)

주소 생성 회로는 입력 받은 ME_code(March Element code)의 주소 설정 부분에 따라 주소를 증가하고 감소시키는 역할을 하는 Address Counter부분과 테스트 할 메모리 블록의 테스트가 모두 끝났는지를 비교하는 Address Comparator로 구성되어 있다. 그림 5는 주소 생성 회로의 블록도이다. Address Counter 부분은 테스트 제어 회로에서 보내주는 주소의 초기화(00), 증가(01), 감소(10)의 신호에 따라서 초기화일 때는 메모리의 주소를 0번지로 초기화하고, 증가일 때는 최하위 메모리 번지부터 증가를 시키고, 감소일 때는 최상위 메모리 번지부터 감소를 시킨다. Address Comparator 부분은 테스트 제어 회로에서 보내주는 주소의 초기화, 증가, 감소에 따라서 메모리는 테스트 할 블록의 마지막 주소를 설정한다. 초기화일 때는 0번지로 초기화하고, 증가일 때는 최상위 주소를 넣고, 감소일 때는 최하위 주소를 넣어준다. 테스트를 실행하면서 생성되는 주소를 마지막 주소와 비교하여서 같아지면 하나의 March Element가 끝난 것이므로, Element_End신호를 테스트 제어 회로에 보내 준다.

3.3.4 테스트 제어 회로(Test Controller Logic)

그림 6은 테스트 제어 회로의 블록도이다. 테스트 제어 회로는 IR Analyzer, FLAG Generator, ME Analyzer로 구성되어 있다. IR Analyzer는 RIR 레지스터의 값을 분석해서 내장 자체 테스트를 동작시키고 Flag Generator는 데이타 비교기에서 보내오는 Match/Unmatch 신호와 주소 생성 회로에서 보내오는 Element end 신호를 RFLAG 레지스터에 설정한다. 그리고 ME

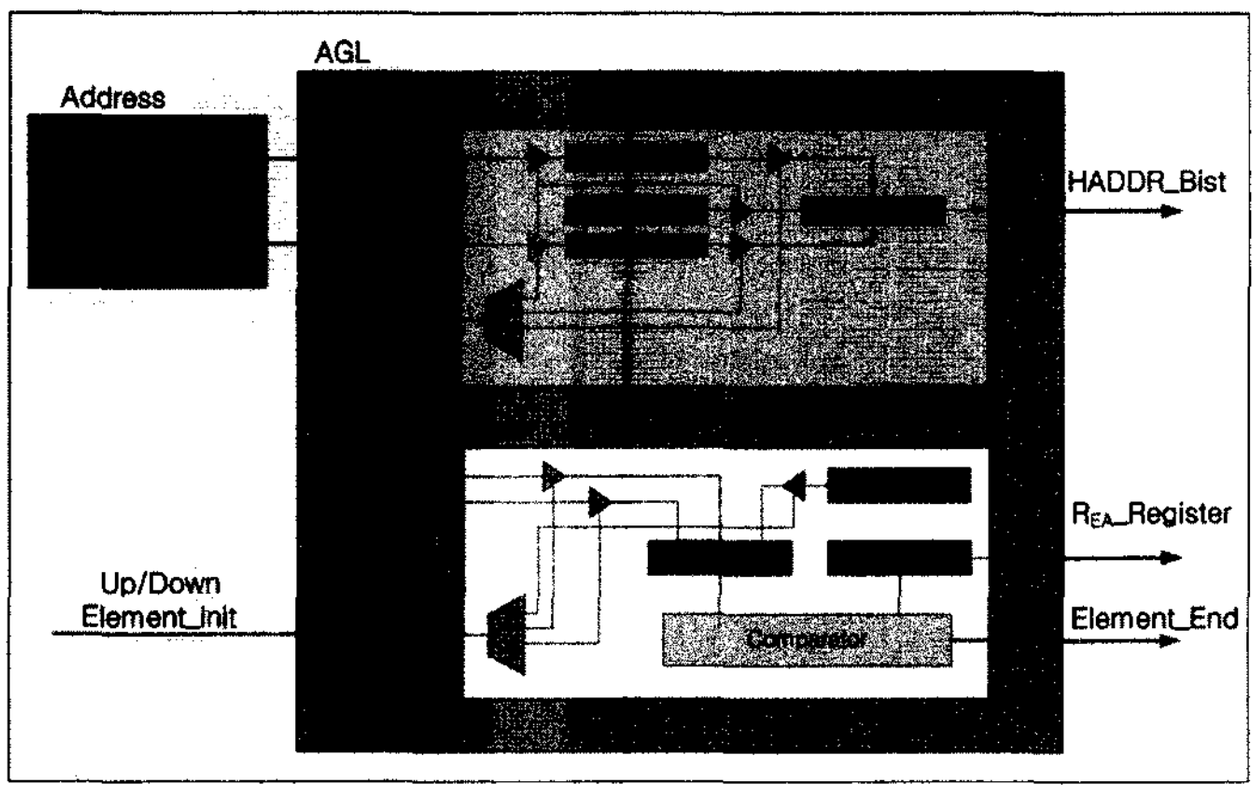


그림 5 주소 생성 회로 구조

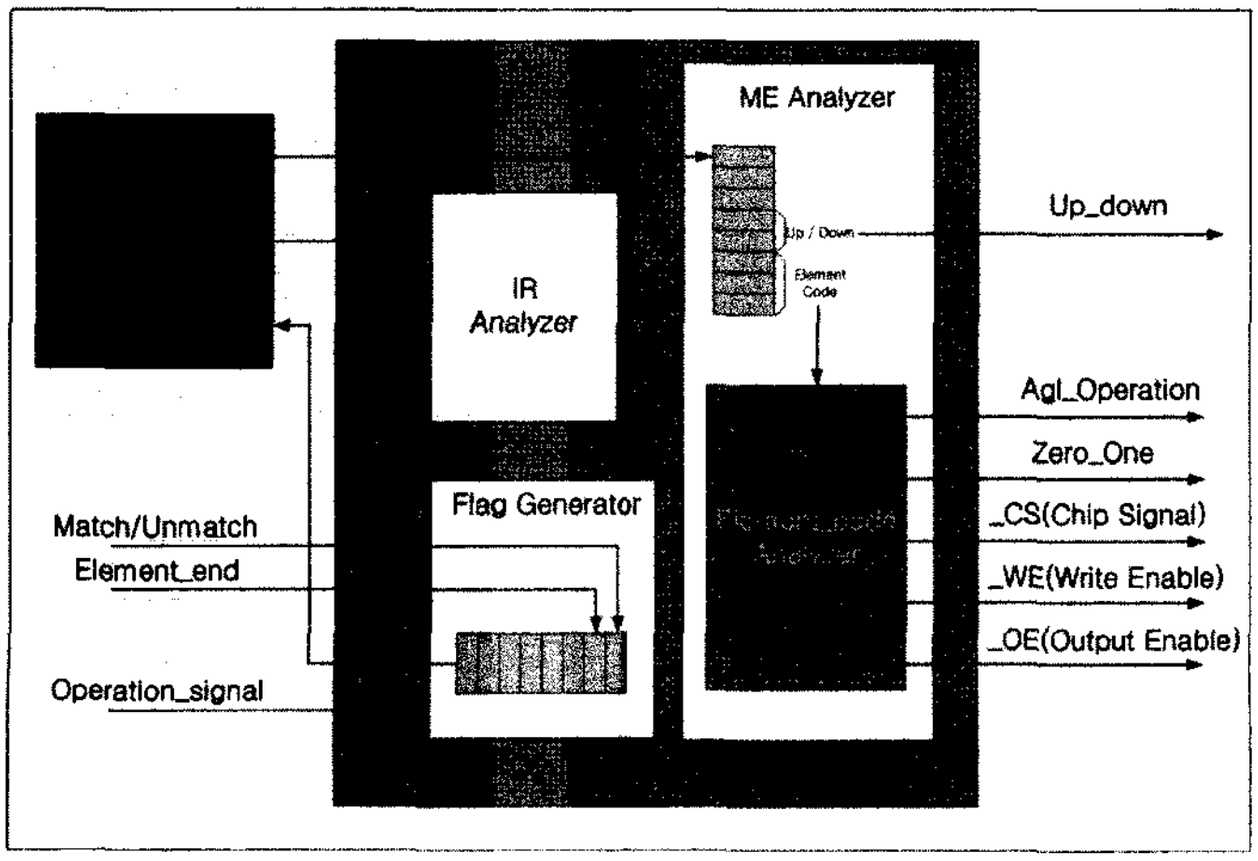


그림 6 테스트 제어 회로 구조

Analyzer는 RME 레지스터의 값을 분석해서 주소 생성 회로에 보내주고, 메모리에 읽기와 쓰기 신호를 보낸다.

4. 실험결과

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 7에서는 총 8개의 신호가 있다. (1)은 Clock 신호이다. (2)는 설정 레지스터에 접근하기 위한 맵핑 주소 값이다. (3)은 설정할 데이타 값이다. (4), (5), (6), (7), (8)은 설정 레지스터의 데이타 값이다. 주소와 데이타가 입력되는 것을 보여준다. ARM 코어의 명령어를 이용하여 주소 'hFFFFFFF0'에 배경 데이타 값 'h5555 5555'를 넣어준다. 주소 범위는 주소 'hFFFFFFF1'에 'h00000000'를 주소 'hFFFFFFF2'에 'h00000003'을 넣어서 테스트 범위를 설정한다. 다음은 주소 'hFFFFF FF3'에 'h00000009'를 넣어서 March element는 "⇑(w0)"로 설정하였다. 다음에 마지막으로 'hFFFFFFF4'에 'h000000001'을 넣어서 BIST를 동작시킨다.

그림 8의 (9), (10), (11)은 설정 레지스터의 데이타 값이다. (12), (13), (14)는 메모리의 주소와 입력데이타, 출력데이타의 값이다. (15), (16), (17)메모리의 제어 신호이다. (18)은 March element_end 신호이다. (19)는 테스트의 고장시 발생하는 신호이다. ①블럭을 보면 메모리의 주소를 증가시키는 것을 볼 수 있다. 그리고 (13)에 메모리 입력데이타를 보면 'h55555555'의 값이 인가되는 것을 볼 수 있다. 이와 같이 설정한 대로 "⇑(w0)"의 작업을 실행한다. (18)을 보면 설정한 테스트 범위인 3번까지의 테스트가 끝났으므로 신호가 발생하는 것을 볼 수 있다. 이렇게 element_end 신호가 발생하면 (10)번의 플래그 레지스터를 갱신한다. ARM 코어는 이 플래그 레지스터를 체크해서 하나의 element가 끝난 것을 감지하면 다음의 element를 실행한다. ②블럭을 보면 "⇑(w0)"의 작업이 끝난 후 "⇑(r0)"를 설정하는 작업을 보여 준다. 우선 'hFFFFFFF4'에 '0'을 넣어서 작업을 중단하고 새로운 element를 설정한 후에 다시 Bist를 동작시키는 작업이다. ③블럭을 보면 "⇑(r0)"의 작업을 실행한다. (14)의 메모리 출력데이타를 보면 첫 번째 element에서 쓰기 작업을 했던 값이 정상적으로 나오는 것을 볼 수 있다. 정상 동작시에는 메모리의 쓰기와 읽기가 제대로 동작을 해서 (19)번의 신호는 변화가 없다.

그림 9에서는 메모리의 2번지 주소에 고착고장을 넣어서 테스트한 결과이다. ④블럭에서 나타나듯이 2번 주소의 값을 읽으면 'h11111111'의 값이 나와서 ⑤블럭에서 고장 신호가 발생한다. 고장 신호가 발생하면 (9)에 고장난 주소, (10)에 플래그의 상태, (11)에 고장 주소의 데이타 값이 출력되는 것을 확인할 수 있다.

Xilinx사의 ISE 8.1i를 사용하여 제안하는 메모리 테

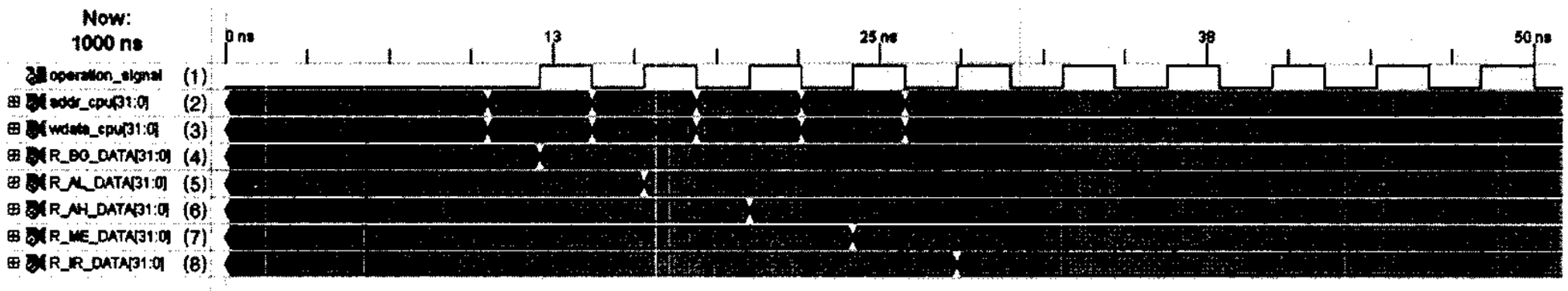


그림 7 테스트를 위한 설정 레지스터 데이타 입력

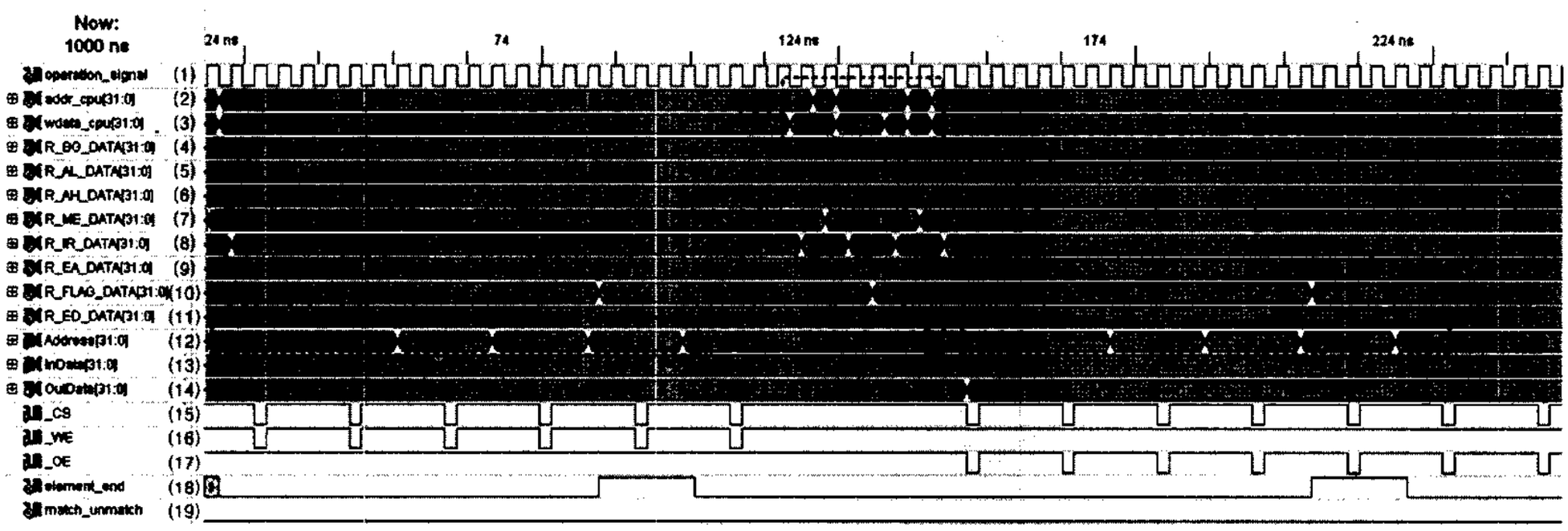


그림 8 정상 동작 메모리 테스트 결과 파형

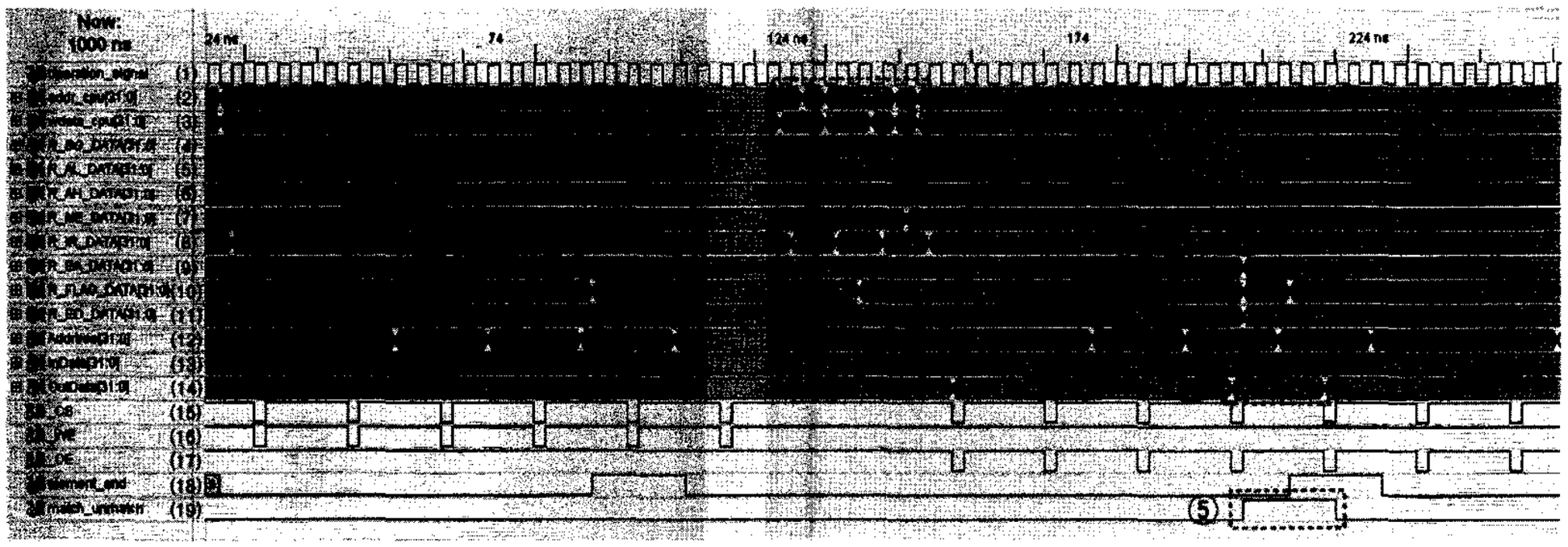


그림 9 고장 메모리 테스트 결과 파형

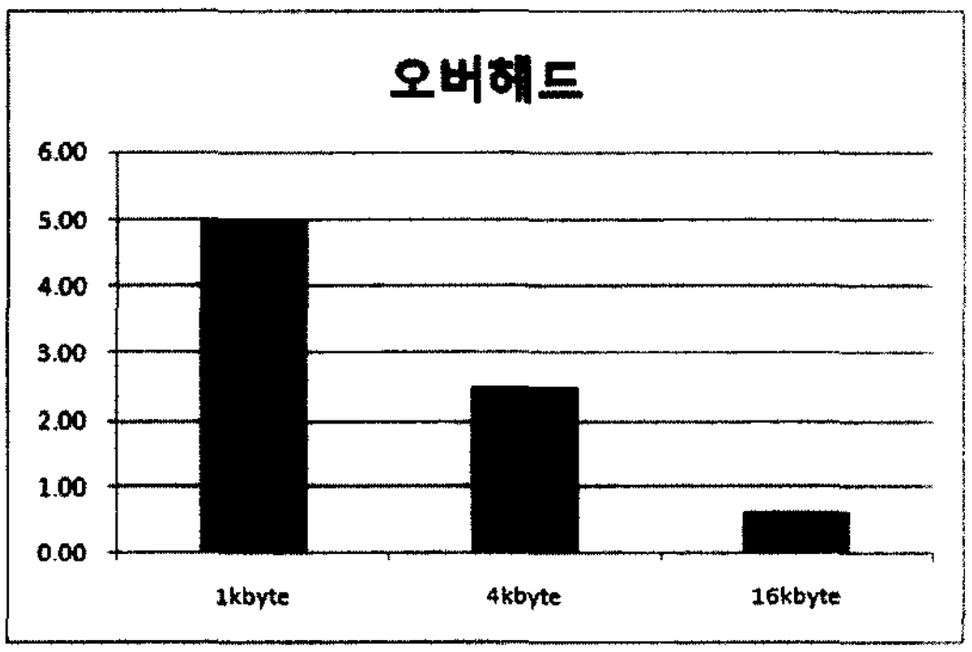


그림 10 제안하는 메모리 테스트의 메모리 크기별 오버헤드

스트의 구조를 구현하여 보았다. 테스트할 메모리의 크기를 1KByte, 4KBytes, 16KBytes로 정하였고, 각 메모리는 32bit 워드를 가지며 2^8, 2^{10}, 2^{12} 크기의 주소를 가진다. 그림 9를 보면 메모리의 크기가 1KByte인 메모리를 테스트하는 메모리 테스트 회로의 오버헤드는 5.01%이고, 4KByte 메모리를 테스트하는 회로의 오버헤드는 2.51%이고, 16KByte 메모리를 테스트하는 회로의 오버헤드는 0.63%의 오버헤드를 가졌다. 메모리는 용량이 커짐에 따른 면적 증가율이 해당 메모리를 테스트하는 회로의 크기 증가율보다 매우 크게 증가하였다. 즉, 메모리 면적은 메모리 크기에 거의 비례하게 증가하였지만, 제안하는 테스트 회로는 비슷한 수준의 크기를 유지하는 것을 알 수 있다. 끝으로 구현한 프로그램 가능한 메모리 내장 자체 테스트의 최대 동작 주파수는 71.441MHz이다.

5. 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.

메모리가 하나의 칩 안에 내장됨에 따라 외부에서 내장된 메모리에 대한 접근성이 떨어져 내장된 자체 테스트가 필요하게 되었고, 하나의 알고리즘을 사용하는 테스트는 메모리 생산 과정을 반복함으로써 얻어지는 수율 변화에 따른 알고리즘 변경이 불가능 하여 알고리즘 변경이 필요할 때마다 새롭게 설계를 하여야 했다. 그리하여 메모리 테스트를 수행하는 과정에서 시간과 비용을 줄일 수 있는 프로그램 가능한 메모리 내장 자체 테스트가 중요하게 되어서, 본 논문에서는 적은 오버헤드를 가지고, 빠른 속도로 동작하며, 다양한 알고리즘을 지원하는 프로그램 가능한 메모리 내장 자체 테스트를 개발하였다.

기존의 프로그램 가능한 메모리 내장 자체 테스트는 외부의 ATE장비를 통해서 알고리즘을 설정하여야 한다. 그렇지만, 제안하는 프로그램 가능한 메모리 내장 자체 테스트는 ARM 프로세서의 명령어를 이용하여 알고리즘을 설정하기 때문에 외부 테스트 환경의 제약 없이 테스트 가능하고, 여러 개의 알고리즘을 지원하여 생산과정의 필요성에 따라 다양한 알고리즘을 설계 변경 없이 쉽게 적용할 수 있게 되었다. 또한, 테스트 컨트롤러가 ARM 프로세서의 도움을 받아 동작함으로써 컨트롤러의 크기를 줄일 수 있어 회로의 면적을 줄일 수 있었다.

본 논문에서 제안하는 프로그램 가능한 내장 자체 테스트는 전체 메모리 크기와 비교하였을 경우, 매우 작은 면적 오버헤드를 가지고 있다. 결과적으로 제안하는 테스트 구조를 사용하면 테스트 시간과 비용을 줄일 수 있을 것으로 예상한다.

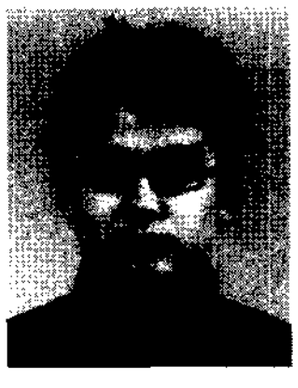
참 고 문 헌

[1] A. J. van de Goor and A.Offerman, "Towards a uniform notation for memory tests," *in Proc. Eu-*

ropean Design and Test Conf., 1996, pp. 420-427.
[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.*, 1996, pp. 435-439.
[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. 2. Principles," *IEEE Design & Test of Computers*, Vol.10, No.2, 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 & Teat of Computers*, Vol.10, No.2, pp. 69-77, March. 1993.
[6] S. Park, K. Lee, C. Im, N. Kwak, K. Kim, 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.
[7] K. Zarrin도, 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.

이 민 호
2005년 호원대학교 전자공학과 학사 졸업. 2008년 숭실대학교 대학원 컴퓨터학과 석사 졸업. 2008년~현재 (주)서림테크놀로지 연구소 연구원. 관심분야는 VLSI 설계 및 테스트, 컴퓨터구조, 임베디드 시스템

홍 원 기
2005년 숭실대학교 컴퓨터학부 학사 졸업. 2007년 숭실대학교 대학원 컴퓨터학과 석사 졸업. 2007년~현재 숭실대학교 대학원 컴퓨터학과 박사 과정. 관심분야는 메모리 테스트, VLSI 설계 및 테스트, 컴퓨터구조, 임베디드 시스템



송 좌 희
2007년 숭실대학교 컴퓨터학부 학사 졸업. 2007년~현재 숭실대학교 대학원 컴퓨터학과 석사 과정. 관심분야는 메모리 테스트, VLSI 설계 및 테스트, 컴퓨터구조, 임베디드 시스템

장 훈
1987년 서울대학교 공대 전자공학과 학사 졸업. 1989년 서울대학교 공대 전자공학과 석사 졸업. 1993년 University of Texas at Austin 졸업. 1994년~2007년 숭실대학교 컴퓨터학부 부교수. 2008년~현재 숭실대학교 컴퓨터학부 정교수
관심분야는 컴퓨터구조, 메모리 테스트, VLSI 설계 및 테스트, 임베디드 시스템