論文93-30A-1-8

메모리에서 PSF 검출을 위한 알고리듬 및 BIST 설계

(PSF detection algorithm and BIST design in memory)

李 仲 鎬*, 趙 相 福*

(Joong Ho Lee and Sang Bock Cho)

要 約

본 논문에서는 RAM에서의 functional 고장인 PSF를 검출할수 있는 "알고리듬 마"를 제안한다. 이 알고리듬은 PSF의 형태를 한정시켜서 제한된 범위의 PSF(restricted PSF or neighborhood PSF)를 검출하는 것으로써, "알고리듬 마"는 SNPSF, PNPSF 및 일부의 ANPSF를 검출하며, 고전적인 고장인 stuck-at 고장 및 천이(transition)고장도 검출한다. 이 알고리듬의 시간 복잡도는 1536×P로써 P는 메모리블럭의 분할갯수를 나타낸다. 또한 "알고리듬 마"의 BIST scheme을 제안하였다.

Abstract

We propose "algorithm MA" which can detect PSF that is the functional fault in RAM. This algorithm based on the restricted PSF(or neighborhood PSF) and can detect not only conventional stuck-at and transition faults but also SNPSF, PNPSF and partially ANPSF. The time complexity of "algorithm MA" has 1536×P(P = no. of partition). We propose total BIST(built-in self test) scheme which implement this algorithm in memory chip.

Ⅰ. 서 론

메모리 집적도 증가로 인한 더 많은 종류의 고장형태들이 나타나고 있으며 또한 고장검출이 더욱 어려워지고 있다. RAM의 Functional 고장은 Stuck-at 고장, 천이고장, 결합고장 및 PSF(pattern sensitive fault)등이 있다. PSF는 메모리셀 내의 누설전류나 전자기적 영향으로 인접한 셀에 내용을 변화시키는 고장을 말하는데 메모리 집적도 증가에 따라 고장발생 확률이 더욱 증가하는 고장 모델로써 본 논문에서는 이고장검출 모델에 중점을 둔다. 결합고장이나 PSF의 검출은 어려우며 그 범위 또한 방대하다. 더우기 메모리 집적도가 매우 증가함에 따라 결합고장이나 PSF의 발생 가능성은 더욱 증가하고 있으므로, 이들의 고장검출이 필수적이다. 이러한 고장은 각각 그 종류가 많고 복잡하며 완전히 이 고장들을 검출한다는 것도 불가능하다.[1,2]

메모리 집적도는 매 2-4년 사이에 4배의 증가율을 보이고 있는 현재, 대규모의 메모리 셀을 테스트하기 위한 경비 및 테스트 시간의 선형적 증가를 초래하고 있다. 이러한 테스트 시간 및 경비를 절감하기 위해 테스트 회로를 내장한 BIST(built-in self test) 방식이 적용되고 있고 또한 필수적이라 할 수 있겠다.[3,4,5,12,13] 한 예로 GALPAT사에서는 n-bit RAM에 대해 $4n^2 + 4n$의 테스트 복잡도를 가지는 방식으로 4M-bit RAM을 테스트할 경우 (200 nsec cycle time)에 162일이나 소요되었다.[6] 또한 GALPAT

*正會員, 蔚山大學校 電子工學科
(Dept. of Elec. Eng., Ulsan Univ.)

(※본 논문은 1991년도 교육부 학술 연구조성비에 의하여 연구되었음.)

사에서 사용한 테스트 방법은 테스트 engineer의 경험에 의한 테스트 방법을 사용했기 때문에 대용량의 메모리에 적용하기에는 부적당한 방법들이었다.[6] 최근 line mode 테스트, micro programmable ROM을 이용한 테스트, ECC(error checking and correcting)등 여러 가지 테스트 방식들이 제안되고 있다. 그러나 위의 방식들 모두가 테스트 속도, 고장 검출 범위, 부가 회로 면적의 세가지 조건을 모두 만족시키지는 못한다.

본 논문에서는 제한된 PSF를 검출할 수 있는 알고리듬을 제안한다. 제안한 "알고리듬 마" 의 PNPSF, ANPSF 및 SNPSF 검출정도 및 기존의 고전적인 고장인 stuck-at 및 천이 고장 검출정도에 대해서 논한다. 또한 제안한 알고리듬을 BIST 기법으로 설계하여 1.5㎛ CMOS 기술로 제작하였다.

Ⅱ. 고장모델

본 논문에서는 고장을 검출하는 것에 주 관점을 두며 고장의 위치는 고려하지 않는다. 이러한 이유로 RAM의 메모리 셀외의 주변회로를 functional 테스트에 대한 모델 설정에 있어서 크게 세부분으로 단순화 시킬수 있다. 즉 어드레스 디코더, 메모리셀 어레이, read/write 회로에 대한 모델이 functional 테스트시에 고려되어진다.[7] 또한 대부분의 read /write 회로 및 어드레스 디코더에서 발생하는 고장들은 메모리셀 어레이에서의 고장들로 매핑(mapping)되어 나타난다.[8,9] 따라서 본 논문에서는 메모리셀을 일정한 형태의 블럭으로 나누어(tiling), 즉 제한된 영역내에서 PSF를 아래와 같이 설정한다.

제한된 PSF(restricted PSF, or neighborhood PSF)를 본 논문에서는 NPSF라 표기하겠다. NPSF 를 검출하기 위해 테스트하는 셀을 기본셀(base cell)이라 하고, 기본셀을 포함한 이웃셀을 neigh- borhood cell이라 하며 기본셀이 제외된 이웃셀을 deleted neighborhood cell이라고 한다. NPSF 의 고장모델은 아래와 같다.[10,11]

1. Active NPSF (ANPSF) or Dynamic NPSF(DNPSF)

이웃셀의 내용변화로 인해 기본셀의 내용이 변화되는 고장을 말한다. 이러한 고장을 검출하기 위해서는 deleted neighborhood cell의 내용을 변화시키는 모든 가능한 pattern에서 각 셀을 0 과 1 상태에서 읽어야 한다.

2. Passive NPSF (PNPSF)

이웃셀의 어떤 특정한 패턴으로 인해 기본셀의 내용을 변화시켰음에도 불구하고, 기본셀의 내 용이 변화하지 않는 고장을 말한다. 이러한 고장을 검출하기 위해서는 deleted neighborhood cell의 모든 pattern 에 대해 각 셀의 0 과 1 의 상태에서 쓰고 읽어야 한다.

3. Static NPSF (SNPSF)

이웃셀의 어떤 특정한 패턴으로 인해 기본셀의 내용이 어떤 특정한 값으로 되는 고장을 말한다. 이러한 고장을 검출하기 위해서는 deleted neighborhood cell의 모든 pattern에 대해 각셀 을 0과 1 상태에서 각각 읽어야 한다.

이러한 고장이외에 본 논문에서는 다음과 같은 고장을 정의한다.

[정의1] 어떤 특정한 테스트 패턴을 write 후 read 했을때 error 가 발생하지 않고 일정한 시간 (t) 만큼 경과 후 그 테스트 패턴의 영향으로 고장이 나타나는 것을 시간경과 고장(time delay fault :TDF)이라 한다.

정의 1과 같은 고장은 메모리 셀내의 전자기적 영향으로 인한 누설전류와 같은 형태로 나타난다. 위의 3가지 NPSF모델 및 정의 1과 같은 모델이 실제 메모리 셀에서 발생되는 경우를 살펴보면 다음과 같다.

(가) 주변셀의 고정된 pattern에 대해
- ㉠기본셀의 변화는 SNPSF 에 해당
- ㉡기본셀의 쓰기 동작이 수행안되는 경우는 PNPSF에 해당

(나) 주변셀의 내용이 변화할 때
- ㉢기본셀의 변화는 ANPSF(DNPSF)에 해당
- ㉣기본셀의 쓰기 동작이 수행안되는 경우는 ㉡에 해당한다. 이 경우는 주변셀이 변화하고 난 후에 기본셀에 쓰기 동작을 수행하는 결과이므로 결국 주변셀의 고정된 pattern 에 대한 ㉡의 고장과 같다.

(다) 기본셀의 내용 변화에 의한 주변셀의 변화
이 경우 기본셀이 주변셀이 되어 다른 기본셀에 영향을 미치는 것과 같으므로 ㉢, ㉣와 같은 고장 형태로 매핑할 수 있다.

(라) 시간 경과에 따른 고장
특정셀을 테스트하기 위해 특정셀을 read 할때는 고장이 검출 되지않고 즉, 고장이 발생하지 않고 일정 시간이 지난 후에 고장이 발생하는 경우 (메모리셀의 누설 전류로 인해 발생 가능함) 이러한 고장은 당장 검출은 불가능하나, 다음 pattern의 테스트시 검출 가능하고, 특히 고장 강도를 높혀서 테스트하면 더욱 효과적이다.

위에서 고려한 고장 이외에 read시에 발생하는 고장을 고려할 수 있다. 그 예를 들어 테스트시 에 어떤 특정한 하나의 셀만 1 이고, 나머지 모든 셀이 0 이라 가정한다면, 이때 분명히 고장이 발생 되었음에도 불구하고 sense amp 를 통해서 비트 라인으로 출력되는 동안 라인간의 기생 커패시턴스에 의해 1→0으로 변하는 고장 형태는 본 논문에서는 고려하지 않았다.

Ⅲ. 고장검출 및 알고리듬

Ⅱ장에서와 같이 고장모델들을 검출하기 위해 전체 메모리 셀을 일정한 형태로 tiling하여 전체 tile에 대해 똑같은 테스트 패턴을 알고리듬에 의해 가하게 된다. 전체 tile은 그림1과 같으며 효과적으로 테스트를 수행하기 위해 Set A 와 Set B로 tiling 한다. 이때 그림에서 보다시피 2번으로 label 된 셀은 기본셀(base cell)이 되며 1,3,4번으로 label된 셀도 주변셀(neighborhood cell)이 된다.

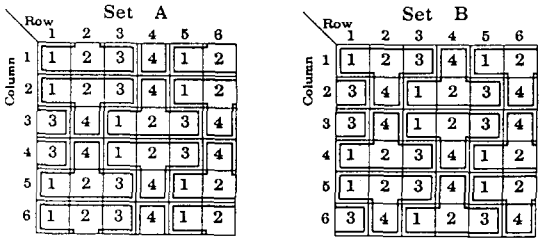


그림 1. tiling 된 6×6 메모리셀 어레이
Fig. 1. Tiling of 6×6 memory cell array.

또한 하나의 tile에 인가될 테스트 패턴을 시퀀스 발생기에 의해서 4×24개의 패턴을 발생시킨다. 이와 같은 패턴을 전체 tile에 대해 똑같이 적용시킨다. Set A,B를 실행함으로서 실제적으로 발생되는 패턴은 128가지로 나타난다. 시퀀스 발생기에서 발생한 패턴은 Eulerian path에 의해 발생된 test pattern 이다.

1. 알고리듬 마

본 알고리듬은 그림 1에서와 같이 1개의 기본셀과 3개의 이웃셀을 가지도록 전체 tile를 구성하였다. 이러한 tiling된 각 셀에 테스트 패턴을 알고리듬에 의하여 인가하게 된다. 이에 대한 알고리듬 은 아래와 같다.

〈〈알고리듬 마〉〉

㉠ 메모리 블럭의 첫번째 partition 을 선택

㉡ 첫번째 테스트 패턴을 발생하여 각 셀에 0을 write(초기화)

㉢ tile 내부의 label을 1로 둔다.

㉣ 지정한 label의 셀값을 전체 tile에 대해 동시에 읽는다.

㉤ 지정한 label의 셀에 테스트 패턴을 동시에 write한다.

㉥ write한 셀을 다시 읽는다.

㉦ tile내의 4개의 label에 대해 ㉣-㉥까지를 완료했으면, 다음 테스트 패턴을 발생시키고 ㉢로 간다. 그렇지 않으면 label번지를 1 증가시키고 ㉣-㉥를 반복수행.

㉧ 128개의 테스트 패턴을 모두 발생했으면 테스트를 종료한다. 그렇지 않으면 ㉢-㉧를 반복수행한다.

㉨ 다음 메모리 partition을 선택하고 ㉡로 간다. 모든 메모리 partition에 대해 ㉡-㉧을 모두 수행 했으면 테스트를 종료한다.

1)고장검출 범위

(가) SNPSF

기본셀을 먼저 read 해 봄으로써 주변셀의 고정된 pattern 에 대해 기본셀이 변화된 고장을 검출 할 수 있으며, 기본셀에 테스트 패턴을 write 후 read 해 봄으로써 기본셀에 쓰기동작이 수행 안되는 고장을 검출할 수 있다. 모든 tiling 번지의 셀에 대해 위와 같은 방법으로 SNPSF 를 검출할 수 있다.

(나) PNPSF

(가)의 방식으로 PNPSF 또한 검출할 수 있다.

(다) DNPSF 혹은 ANPSF

기본셀에 write 하기전에 read 함으로써 주변셀의 바로 직전 테스트 패턴의 write 수행으로 인해 기본셀이 변화된 고장을 검출할 수 있다. 이와 같은 방법으로 모든 기본셀에 대해 똑같은 방식으로 고장을 검출할 수 있다.

라) 시간 경과에 따른 고장

"알고리듬 마"에서 첫번째 테스트 패턴이 0000 이었고 두번째 테스트 패턴이 0001 이었다면, 실제로는 제일 하위 비트만 0→1 로 변한 결과이지만, 첫번째 테스트 패턴으로 테스트시 최하위 비트 에서 최상위 비트까지 모두 read-write-read 를 수행하고, 다음 두번째 테스트 패턴 또한 최하위에서 최상위 비트까지 모두 read-write-read 를 수행하므로 변하지 않은 세 비트에서 시간 경과에 따른 고장이 발생했다 해도 모

두 검출 가능하다. 또한, 변한 최하위 비트는 다음번째 테스트 패턴의 테스트 시에 모두 검출 가능하다.

Ⅳ. BIST 설계

앞에서 제안한 "알고리듬 마"를 효과적으로 수행하기 위한 전체 BIST scheme블럭 다이어그램을 그림 2 에 나타내었다. 이 scheme 은 기존의 RAM에 부가적으로 어드레스 생성기(address genera-tor logic: AGL), 테스트 패턴 발생기(test pattern generator: TPG)와 병렬 writer(parallel writer: PW) 그리고 에러 검출단(error detector)이 필요하게 된다. 테스트 신호가 인가되면 클럭 발생기(그림 2의 CLK) 에 의해서 열 선택기(column address selector:그림 2의 CS)와 행 선택기(row address selector: 그림 2의 RS)를 제어한다. 동시에 테스트 패턴을 발생하는 기본 클럭을 발생하여 테스트패턴 발생부(TPG)의 클럭으로 인가하게 된다. 동시에 TPG에서는 64개의 테스트 패턴 을 순차적으로 발생하여 각 패턴을 연속적으로 선택하며 data I/O 단을 거쳐서 메모리 셀에 write 동작을 수행하게 된다. 행 선택기는 병렬 테스트를 수행하기에 적합하도록 BIST Scheme 과 연결되어 각 비트라인을 병렬로 선택하게 된다. 열 선택기는 멀티플랙서 2개로 구성되며, fan-in, fan-out 을 고려하여 각 sub-array 에 첨가된다. BIST scheme은 Valid Logic Workbench 상에서 로직 시뮬 레이션을 완성하였다.

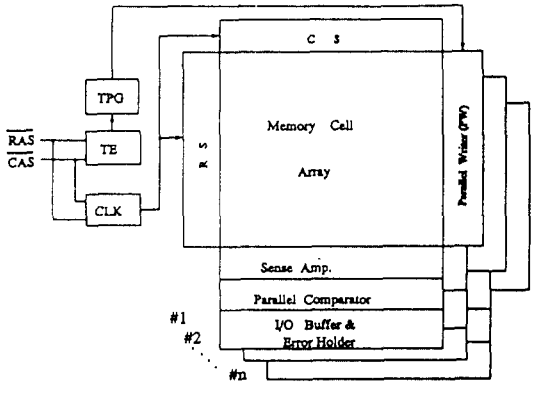


그림 2. BIST scheme의 전체 블럭 다이어그램
Fig. 2. The block diagram of total BIST scheme.

1. 테스트 모드

본 논문에서는 메모리 테스트 모드로 들어가기 위해 부가적인 테스트 핀을 따로두지 않고 기존의 입력핀을 이용하는 방법을 적용했다. 이때 $\overline{RAS}$와 $\overline{CAS}$의 입력핀을 사용하였다. 그림3에 test enable (TE) 회로와 타이밍다이어그램을 나타내었다. 정상메모리 동작시에는 $\overline{RAS}$ 신호가 $\overline{CAS}$ 신호 보다 먼저 low로 떨어진다. 그러나 테스트 모드로 들어가려면 $\overline{CAS}$ 신호가 $\overline{RAS}$ 신호보다 먼저 low로 떨어지게 한다.

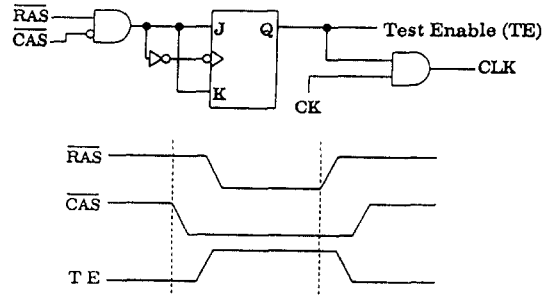


그림 3. 테스트 enable 회로및 타이밍 다이어그램
Fig. 3. The test enable circuit & timing diagram.

2. 테스트 패턴 발생기 (TPG: Test Pattern Generator)

TPG는 Eulerian path에 의해 유출된 64개의 테스트 패턴을 발생시키는 시퀀스 발생기(SG) 와 시퀀스 콘트롤러(SC)로 구성된다.

1) 시퀀스 발생기(SG: Sequence Generator)

시퀀스 발생기 및 로직 시뮬레이션 결과를 그림 4에 나타내었다. 시퀀스 발생기의 구성은 상단의 4개의 플립플롭과 2개의 AND 게이트로 이루어진 동기식 이진카운터로 구성되어있다. 이 카운터의 출력이 아래단의 조합회로를 거치면서 4비트로 구성된 64개의 테스트 패턴이 PGO⟨1⟩-PGO⟨4⟩ 단을 통해서 출력된다. 그림4에 이에대한 회로및 로직시뮬레이션 결과를 나타내었다.이 회로에 대한 4비트의 출력 함수는 아래와 같다.

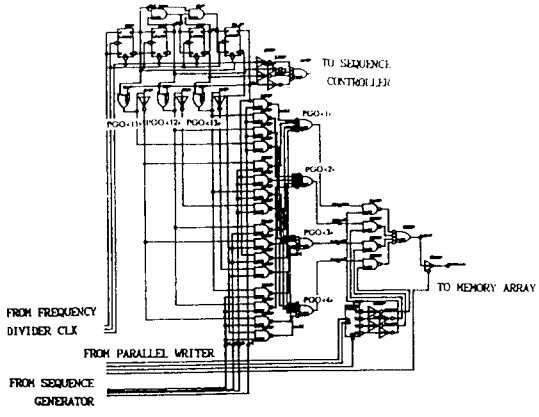
PGO⟨1⟩ = Q(JK3)*OUT3 + PGO⟨10⟩*OUT2 + PGO⟨11⟩*OUT1 + PGO⟨12⟩*OUT0

PGO⟨2⟩ = PGO⟨12⟩*OUT3 + Q(OUT3)*OUT2 + PGO⟨10⟩*OUT1 + PGO⟨11⟩*OUT0

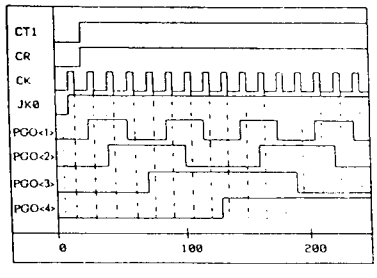
PGO⟨3⟩ = PGO⟨11⟩*OUT3 + PGO⟨12⟩*OUT2 + Q(JK3)*OUT1 + PGO⟨10⟩*OUT0

PGO⟨4⟩ = PGO⟨10⟩*OUT3 + PGO⟨11⟩*OUT2 + PGO⟨12⟩*OUT1 + Q(JK3)*OUT0

초기에 clear 신호가 0→1로 set 되고, 모든 플립플롭이 클럭(CLK)에 의해 동기된 신호가 출력된다. 이 출력을 조합 회로에서 조합하여 테스트 패턴을 #1 ⇒#64까지 발생하는데, 이때 시퀀스 콘트롤러로부터 출력된 OUT3 -OUT0 신호에 의해 각 테스트 패턴의 발생이 제어되어 진다. OUT3-OUT0 신호는 그림 5에 표시하였다. 이렇게 발생된 테스트 패턴은 data I/O 단을 거쳐서 순차적으로 write 되어 지기 전에 입력받는다. 2-to-4 디코더를 거쳐나온 신호에 의해 각 셀의 tiling 번지에 쓰여져야할 테스트 패턴 값이 선택되어진다.



(a)



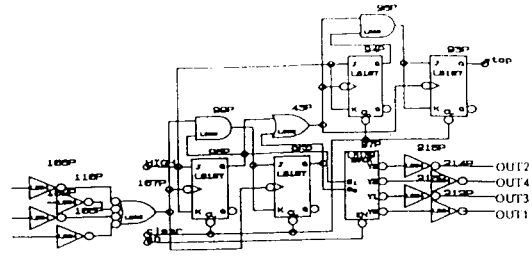
(b)

그림 4. 시퀀스 발생기 및 로직시뮬레이션 결과
(a) 시퀀스 발생기
(b) 로직시뮬레이션 결과

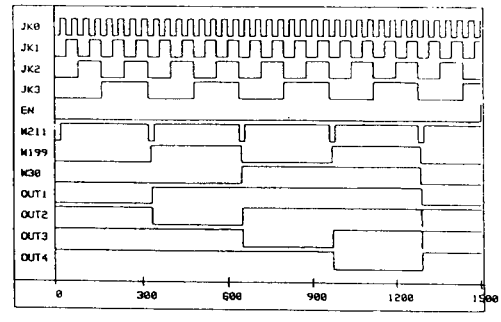
Fig. 4. sequence generator and logic simulation result
(a) sequence generator
(b) logic simulation result.

2) 시퀀스 콘트롤러(SC:Sequence Controller)

시퀀스 콘트롤러는 parallel write scheme을 구성하는 분주기의 출력을 클럭으로 하단의 두개의 플립플럽과 이들을 동기시키기 위한 AND 게이트, OUT3-OUT0신호를 제어하는 디코더와 그리고 제안한 알고리듬에서 2개의 set A, B에 필요로하는 테스트 패턴이 모두 발생되면 STOP 신호가 출력되는 회로로 구성된다. SG 에서 OUT3-OUT0 신호를 발생하여 PG의 테스트 패턴을 제어하도록 인가되어진다. 즉 OUT3 는 테스트 패턴 TP1-TP16, OUT2는 TP17-TP32, OUT1은 TP33- TP48 그리고 OUT0는 TP49-TP64를 제어하여 다음 단으로 테스트 패턴을 전송시킨다. 그림 5에 SG 회로 및 로직 시뮬레이션 결과를 나타내었다.



(a)



(b)

그림 5. 시퀀스 발생기 및 로직 시뮬레이션 결과
(a) 시퀀스 콘트롤러
(b) 시퀀스 콘트롤러의 로직 시뮬레이션 결과

Fig. 5. Sequence generator and logic simulation result
(a) Sequence controller
(b) Logic simulation results of sequence controller.

3. 병렬 writer (Parallel Writer:PW)

PW는 클럭 발생기, 열 선택기 및 행 선택기로 구성된다. 여기서 클럭 발생기는 테스트가 enable 된 후 테스트 패턴을 발생하기 위한 클럭이 발생되며, 이에 따라 한개의 테스트 패턴 TP# 가 발생된다. 이

때 클럭 발생기는 8분주 카운터로 구성된다. 또한 클럭 발생기로부터 행 선택기와 열 선택기를 동시에 동작하게 하여, PG 로부터 발생되어 전송된 테스트 패턴을 tiling 된 번지의 셀에 인가한다.

1) 행 선택기 (Row Address Selector)

PW 의 둘째단 플립플롭의 출력으로부터 제어되는 멀티플렉서를 통해 tiling 된 셀을 순차적으로 선택하게 된다. 그림 6에 tiling 된 것은 SEL 입력이 1 상태일 때, 즉 set A에 대한 동작 수행 과정을 나타낸 것이다. set B 를 선택하기 위해서는 SEL 입력을 0 상태로 인가하면 된다. 회로에 포함된 각 버퍼들은 타이밍 delay 및 fan-out 을 조절하려는 목적으로 삽입되었다.

2) 열 선택기 (Column Address Selector)

PW 의 둘째단과 네째단 플립플롭의 출력을 디코딩한 출력과 세째단 플립플롭의 출력을 2-to-4 디코더의 입력으로 인가한다. 이 인가된 신호를 2-to-4 디코더를 거쳐 tiling 된 셀을 선택하기 위한 열을 선택한다.

3) 병렬 쓰기 동작

SEL 이 1 일때, 열 선택기(Column Address Selector)의 2-To-4 decoder 에 의해 첫 번째 열 이 선택되어지고, 행 선택기(Row Address Selector)에 의해 첫 번째, 두 번째 행이 동시에 선택 되어진다. 이 때 set A 의 1 번으로 lable 된 전체 메모리 셀에 한 비트의 테스트 패턴이 인가된다. 연속적으로 2-To-4 decoder에 의해 세 번째 열이 선택되어지고 세 번째, 네 번째 행에 해당되는 모든 셀에 테스트 동작이 수행되어진다. 이때 선택되어진 두개의 셀에는 패턴 발생기로부터 발생된 테스트 패턴이 data I/O 단을 거쳐서 순차적으로 각각의 tiling 된 셀에 인가된다. 위와 같은 동작에 의해 64개의 테스트 패턴이 set A 로 tiling 된 메모리셀들에 인가되어 테스트 동작이 완료된다. SEL=0으로 하여 set B 에 대해 똑 같은 동작을 수행한다. Parallel Write scheme 및 로직 시뮬레이션 결과를 그림 6에 나타내었다. 로직시뮬레이션 결과 행과 열을 선택하는 과정에서 전체의 지연 시간이 완전히 일치하지 않아서 아주 미세한 펄스가 튀는 것을 볼 수 있지만, 이 펄스의 최대폭이 7ns에 불과하며 메모리 read/write에 영향을 주지 않을 것으로 생각된다.

앞에서 살펴본 각 scheme 이외에 오류 검출기, 오류 신호 유지기가 있다. 본장에서 설계한 전체 BIST scheme 은 결국 하나의 테스트 패턴에 대해 "알고리듬 마"에서와 같이 2번의 read 와 1번의 write 동작을 수행하게 되는데 이때 본 알고리듬의 테스트 시간을 측정할 수 있는 최소 길이의 클럭 이 총 16 cycle 이 필요하다.

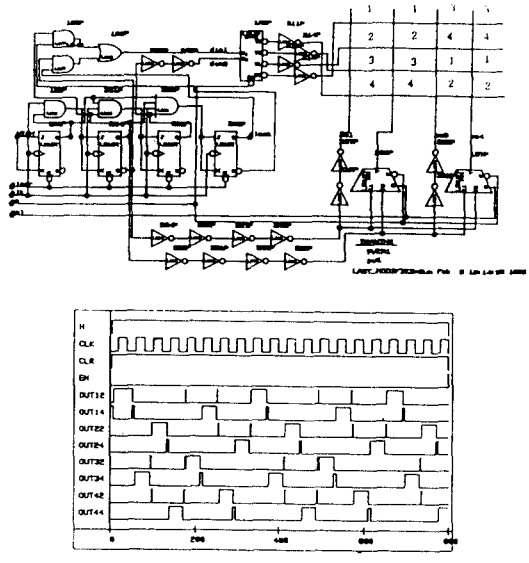


그림 6. 병렬 Write Scheme 및 로직 시뮬레이션 결과

Fig. 6. Parallel writer scheme and logic simulation result.

Ⅴ. 알고리듬의 검증 및 제작

1. 테스트 시간의 분석 및 고장검출 범위

제안한 알고리듬에 의해 소요되는 테스트 시간은 아래와 같다.

"알고리듬 마"의 시간 복잡도

= 테스트 패턴수 × read/write 수 × tile내의 label수 × 메모리 partition 수

= 128 × 3 × 4 × P = 1536 × P

표 1. 각 알고리듬의 고장 검출 범위

Table 1. Fault coverage of each algorithm.

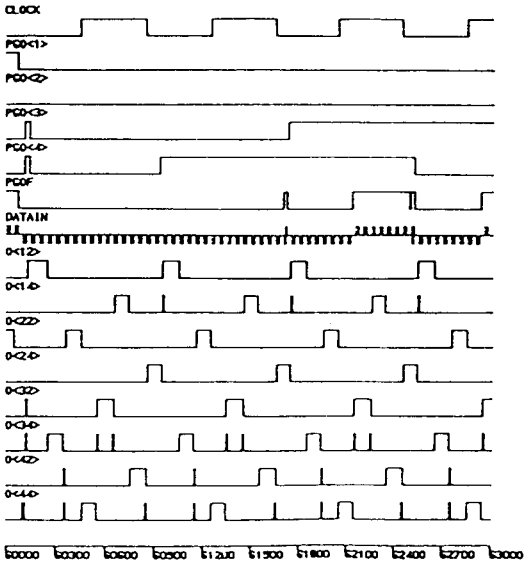
	SNPSF	PNPSF	ANPSF	이웃셀수	시간복잡도
P.MAZUMDER & J.K. PATEL	O	O		4	321n for n-bit RAM
알고리듬 마	O	O	일부검출	3	1536×P

"알고리듬 마"는 하나의 tile에 대해 read/write를

총 3번 행하게 된다. 그리고 P는 병렬 테스트가 가능한 최대의 block 을 고려했을때 전체 메모리에 대한 블럭의 갯수가 된다.

위에서 살펴본 알고리듬의 시간 복잡도에 read/write 시에 소요되는 최소의 시간을 고려하면 실제적인 테스트 시간이 된다. 테스트시 리프레쉬(refresh) 시간은 고려하지 않았다.

표 1.은 본 알고리듬의 고장검출 범위를 기존의 방식 [14] 과 비교하여 나타내고 있다.



Clock : PG의 클럭
PG0⟨1⟩⇒PG0⟨4⟩: 테스트 패턴
DATAIN : data I/O단을 통해서 입력되는 테스트 패턴
0⟨12⟩⇒0⟨44⟩ : 가상적인 메모리셀

그림 7. 전체 BIST scheme에 대한 로직 시뮬레이션 결과
Fig. 7. Logic simulation results of total BIST scheme.

3. 제 작

본 "알고리듬 마"에 대한 전체 layout은 BIST scheme 만을 고려하여 설계완성하였고, 실제 메모리 적용시의 routing 문제를 고려하지 않았다. 그림 8은 layout한 회로에 대한 전체 블럭배치도를 나타낸 것으로써 US_TPGCNT, US_TPGSIG, US TPGDEC는 테스트 패턴 발생부에 해당하며, US CROWSEL 등은 병렬 writer(PW)에 해당한다. 본 BIST scheme의 칩 제작에서 실제 메모리와는 결합하지 않고 BIST scheme 자체만을 layout 하였으므로, 그림 8에서 보다시피 총 5개의 입력핀과 15개의 출력핀이 소요되었다. 그러나 설계한 BIST scheme은 부가적인 테스트 입력핀을 필요로 하지 않고, 그림 3에서 보다시피 기존의 입력핀을 그대로 사용하기 때문에 실제 메모리칩에 내장 시 간편하게 테스트 enable 모드로 들어갈 수 있다.

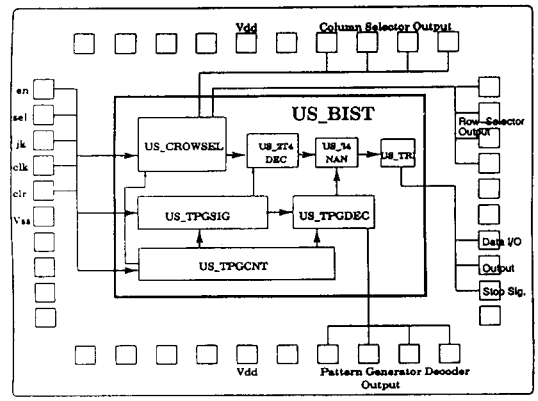


그림 8. BIST scheme의 전체 layout블럭 배치도
Fig. 8. Total layout block placement diagram of BIST scheme.

Ⅵ. 결 론

본 논문에서는 메모리 테스트 속도를 개선시킨 더욱 효과적인 방법에 대해 연구하였다. 메모리의 전체 셀을 3개의 이웃셀을 가지도록 tiling 하여, 제한된 범위의 PSF를 검출할 수 있는 "알고리듬 마"를 제안하였다. 이 알고리듬은 기존의 stuck-at 고장, 천이 고장 뿐만 아니라 SNPSF, PNPSF 및 일부의 ANPSF를 검출할 수 있으며 시간 복잡도는 1536P를 가진다. 이때 P는 메모리의 전체 블럭을 적당히 분할한 갯수이며, 이것은 테스트시 병렬동작을 원활히 수행할 수 있도록 하기위한 것이다.

또한 본 알고리듬을 BIST scheme으로 설계하였다. 설계한 BIST scheme 을 1.5㎛ CMOS 설계규칙에 의해 ASIC 기법으로 layout하였다. 이때 전체 크기는 폭이 2359.00㎛, 높이가 631.50㎛ 으로써 custom design 할경우 더욱 면적을 축소하여 layout할 수 있을 것으로 기대된다. 또한 본 논문에서 제안한 BIST scheme은 외부의 부가적인 테스트 입력핀을 사용하지 않고 기존의 핀을 사용하였기 때문에 테스트 모드로 들어가기 위한 입력핀에 대한 문제점을 제거하였다. 앞에서 제시한 layout 면적은 본

논문의 연구범위상 실제 메모리셀과 그의 부가회로를 제외한 테스트회로 자체만의 면적으로써, 제시한 테스트회로를 메모리칩에 내장시 차지하는 면적은 산출하지 않았으나 실제 routing 문제를 고려하여 수% 이내의 부가면적을 요구할 것으로 예상된다.

參考文獻

[1] J.H.Lee, W.C.Jung, J.M.Yeo and S.B. Cho "BIST scheme design for PSF detection of DRAM proceedings of JTC-CSCC'92," July 27-28,1992,pp. 190-193

[2] J.M.Yeo, J.H.Lee and S.B.Cho "Algorithms for Detection of Coupling Faults in Semi- conductor Memory proceedings of HICEC'92," Aug 14-15, 1992.

[3] S.Fuji, M.Ogihara, M.Shimizu, et al., "A 45-ns 16-Mbit DRAM with triple- well structure, *J. Solid-State Circuits*," vol. SC-24, pp. 1170-1175, Oct. 1989.

[4] T.Mano, T.Matsumura, J.Yamada, et al., "Circuit Technologies for 16Mb DRAMs," ISSCC Digest of Technical Papers, pp. 22-23, 1987.

[5] K.Shimohigashi, K.Kimura, et al., "A 65ns CMOS DRAM with a twisted Driveline Sense Amplifier," ISSCC Digest of Technical Papers, pp. 18-19, 1987.

[6] K.Arimoto, K.Fujishima, et al.,"A 60ns 3.3V 16Mb DRAM," ISSCC Digest of Techni- cal Papers, pp. 244-245, 1989.

[7] A.J.Van De Goor and C.A.Verruijt, "An Overview of Deterministic Functional RAM chip Testing," ACM computing Surveys, vol.22, no.1, March 1990, pp. 5-33.

[8] Nair.R., Thatte, S.M., and Abraham, J.A., "Efficient Algorithms for Testing Semi- conductor Random-Access Memories," *IEEE Trans. Comput*, C-27, Jun, pp.572-576, 1978

[9] Thatte, S.M., and Abraham, J.A. "Testing of Semiconductor Random-Access Memories," In the proceedings of the 7th Annual Internatonal Conference on Fault-Tolerant Computing. *IEEE computer Society*, pp.81-87, 1977.

[10] Suck, D.S., and Reddy, S.M., "Test procedures for a class of Pattern-sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans. Comput*, C-29, June, pp.419-429, 1980.

[11] Saluja, K.K., and Kinoshita, K., "Test pattern Generation for API Faults in RAM," *IEEE Trans. Comput*, C-34, Mar 1985, PP. 284-287.

[12] T. Takeshima et al., "A 55-ns 16-Mb DRAM with Built-in Self-Test Functional Using Microprogram ROM," *IEEE J. Solid-State Circuits*, vol.25, no.4, pp.903-911, Aug, 1990.

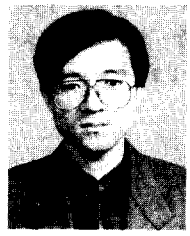
[13] M. Franklin, K.K.Saluja, and K. Kinoshita "Built-in Self-Test Alborithm for Row/Col-umn Pattern Sensitive Faults in RAMS," *IEEE J.Solid-State Circuits*, vol.25, no.2,pp.514-524, Apr.1990

[14] P.Maazumder, and J.K.patel "parallel Testing for Pattern-Sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans*. on Comp. vol.38, no.3, Mar. 1989.

著 者 紹 介

趙 相 福(正會員)

1955年 6月 10日生. 1979年 2月 한양대학교 전자공학과 졸업 학사. 1981年 2月 한양대학교 대학원 전자공학과 졸업 석사. 1985年 2月 한양대학교 대학원 전자공학과 박사. 1981年 2月~1986年 2月 광운대학교, 한양대학교 강사. 1986年 2月~현재 울산대학교 전자 및 전산기공학과 부교수. 주관심분야는 VLSI/ULSI테스트 및 설계, 초고집적 RAM의 Test 및 Testable Design, ASIC Design, VLSI CAD tool 개발 등임.



李 仲 鎬(正會員)

1965年 1月 23日生. 1988年 2月 울산대학교 전자및 전산기공학과 졸업 학사. 1990年 2月 대학원 전자및 전산기공학과 졸업 석사. 1990年 3月~현재 울산대학교 대학원 전자 및 전산기공학과 박사과정 재학 중. 주 관심 분야는 RAM의 Testable design 및 회로 설계, VLSI의 sistem 설계, ASIC Design 등임.