Testing of Random Access Memories

Jin-Fu Li
Advanced Reliable Systems (ARES) Lab.
Department of Electrical Engineering
National Central University
Jhongli, Taiwan
Outline

- Introduction
- Fault Models and Test Algorithms
  - Fault models
  - Test Algorithms
- Memory BIST/BISD Design
  - BIST Design
  - BISD Design
Introduction

- Modern system-on-chip (SOC) designs typically consist of hundreds of memories
  - Memories usually dominate the chip area
- Furthermore, memories are designed with the aggressive design rules such that they are prone to defects
- Thus the memory yield heavily impacts the SOC yield
  - Increasing memory yield can significantly increase the SOC yield
- Yield-enhancement techniques for memories
  - Diagnosis & repair
Yield of an SOC

\[ Y_S = Y_M \times Y_L \]

Improve the yields of memories can drastically increase the yields of SOCs

For example, UltraSparc chip yield

Source: R. Rajsuman, IEEE D&T, 2001
Yield Learning Curve

Yield

Diagnosis/repair

Early phase

Intermediate phase

Mature phase

Time

Repair
Testing and Repair of RAMs in SOCs

16-core SPARC (Oracle)

DFT features:
1. Scan test + test compression
2. Programmable memory built-in self-test (MBIST) + repair
3. SerDes internal and external look-back tests

Niagara2 (Sun)

DFT features:
1. 32 Scans + ATPG
2. BIST for arrays
3. ….

POWER6 (IBM)

DFT features:
1. Logic BIST
2. BIST for arrays
3. BISR for arrays
4. …
Fault Models and Test Algorithms
RAM Architecture

- n-bit address
- row decoder
- row decoder
- row decoder
- row decoder
- column decoder
- 2^{m+k} bits
- 2^{n-k} words
- m-bit data I/Os
- column mux, sense amp, write buffers
Bitline Structure
Functional RAM Model

address latch \rightarrow \text{column decoder} \rightarrow \text{memory cell array} \rightarrow \text{write driver} \rightarrow \text{data register} \rightarrow \text{sense amplifier} \rightarrow \text{data out}

address \leftarrow \text{address latch} \leftarrow \text{row decoder} \rightarrow \text{row decoder} \rightarrow \text{read/write & enable} \rightarrow \text{data in}
Reduced Functional RAM Model

- The address latch, the row, and the column-decoder are combined to the address decoder
  - They all concern addressing the right cell or word
- The write driver, the sense amplifier, and the data register are combined to the read/write logic
  - They all concern the transport of data from and to the memory cell array
Stuck-At Fault & Transition Fault

- **Stuck-at fault (SAF)**
  - **Definition:** The logic value of a stuck-at (SA) cell or line is always 0 or 1. It is always in state 0 or in state 1 and cannot be changed to the opposite state.
  - **Detection requirement:** From each cell or line, a 0 or 1 must be read.

- **Transition fault (TF)**
  - **Definition:** A cell that fails to undergo a 0 to 1 transition when it is written is said to contain an **up transition fault**. A **down transition fault** indicates that a cell fails to undergo a 1 to 0 transition.
  - **Detection requirement:** Each cell should undergo up and down transitions and be read after each transition before undergoing further transitions.
State Diagram for SAF & TF

State diagram of a good cell

SA0 fault

SA1 fault

TFu
Address Decoder Fault

- Address Decoder Fault (AF)
  - An address decoder fault (AF) is a functional fault in the address decoder that results in one of four kinds of abnormal behavior:
    - Given a certain address, no cell will be accessed
    - A certain cell is never accessed by any address
    - Given a certain address, multiple cells are accessed
    - A certain cell can be accessed by multiple addresses
BF & SOF

- Bridging Fault (BF)
  - A bridging fault (BF) occurs when there is a short between two cells
  - AND-type BF
  - OR-type BF

- Stuck-Open Fault (SOF)
  - A stuck-open fault (SOF) occurs when the cell cannot be accessed due to, e.g., a broken word line
  - A read to this cell will produce the previously read value
Coupling Fault

- **Coupling Fault (CF)**
  - A coupling fault (CF) between two cells occurs when the logic value of a cell is influenced by the content of, or operation on, another cell

- **State Coupling Fault (CFst)**
  - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state

- **Inversion Coupling Fault (CFin)**
  - Transition in coupling cell complements (inverts) coupled cell

- **Idempotent Coupling Fault (CFid)**
  - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0
State Diagram for CFs

State diagram of two good cells

State diagram of an CFin<u;i>

State diagram of an CFid<u;1>
Summary of CFs

- Note that all definitions talk about single-way faults, that is, the presence of a CF from cell $i$ to cell $j$ does not imply the presence of a CF from cell $j$ to cell $i$.

- Suppose that a transition or state in cell $j$ can induce a coupling fault in cell $i$. cell $i$ is then said to be coupled cell (or victim); cell $j$ is called the coupling cell (or aggressor).

- A test that has to detect and locate all coupling faults should satisfy
  - For all coupled cells, each cell should be read after a series of possible coupling faults may have occurred.
CFs in Word-Oriented RAMs

- CFs in bit-oriented RAMs

- CFs in word-oriented RAMs
  - Inter-word CFs & intra-word CFs
Neighborhood Pattern Sensitive Fault

- Pattern-Sensitive Fault (PSF)
  - The PSF is a general (multi-cell) coupling fault, which causes the content of a memory cell, or the ability to change the content, to be influenced by certain patterns of other cells in the memory
  - In general, the number of aggressor and victim cells may be 4, 5, 9, etc.
- The target PSF is the Neighborhood PSF (NPSF)
  - The aggressor cells are the neighborhood of the victim cell
  - 5-cell neighborhood
  - 9-cell neighborhood
5-Cell & 9-Cell NPSFs

- Neighborhood Pattern Sensitive Fault (NPSF)
  - Type-1 neighborhood
    - Base cell
    - Deleted neighborhood
  - Type-2 neighborhood
Types of NPSFs

- Three types of NPSFs
  - Active NPSF (ANPSF)
  - Passive NPSF (PNPSF)
  - Static NPSF (SNPSF)

- ANPSF
  - The base cell changes due to a change in the pattern of the deleted neighborhood
  - An ANPSF test has this necessary condition
    - Each base cell must be read in state 0 and state 1, for all possible deleted neighborhood pattern changes
Types of NPSFs

- **PNPSF**
  - A specific neighborhood pattern prevents the base cell from changing
  - The necessary condition to detect and locate a PNPSF
    - Each base cell must be written and read in state 0 and in state 1, for all deleted neighborhood pattern permutations

- **SNPSF**
  - The base cell is forced into a particular state when the deleted neighborhood contains a particular pattern
  - The necessary condition of test is
    - Each base cell must be read in state 0 and in state 1, for all deleted neighborhood pattern permutations
Disturb Fault

- Disturb Fault (DF)
  - Victim cell forced to 0 or 1 if we (successively) read or write aggressor cell (may be the same cell)
  - Hammer test

- Read Disturb Fault (RDF)
  - There is a read disturb fault (RDF) if the cell value will flip when being read (successively)
Data Retention Fault

- Data Retention Fault (DRF)
  - DRAM
    - Refresh Fault
      - Refresh-Line Stuck-At Fault
    - Leakage Fault
      - Sleeping Sickness---loose data in less than specified hold time (typically tens of ms)
  - SRAM
    - Leakage Fault
      - Static Data Losses---defective pull-up
    - Checkerboard pattern triggers max leakage
RAM Test Algorithms

- A test algorithm (or simply test) is a finite sequence of test elements
  - A test element contains a number of memory operations (access commands)
    - Data pattern (background) specified for the Read and Write operation
    - Address (sequence) specified for the Read and Write operations
- A march test algorithm is a finite sequence of march elements
  - A march element is specified by an address order and a finite number of Read/Write operations
March Test Notation

- \(\uparrow\): address sequence is in the ascending order
- \(\downarrow\): address changes in the descending order
- \(\uparrow\downarrow\): address sequence is either \(\uparrow\) or \(\downarrow\)
- \(r\): the Read operation
  - Reading an expected 0 from a cell (\(r0\)); reading an expected 1 from a cell (\(r1\))
- \(w\): the Write operation
  - Writing a 0 into a cell (\(w0\)); writing a 1 into a cell (\(w1\))
- Example (MATS+): \(\uparrow\downarrow(w0); \uparrow(r0,w1); \downarrow(r1,w0)\)
Classical Test Algorithms: Checkerboard

☐ Checkerboard Algorithm
  ■ Zero-one algorithm with checkerboard pattern
  ■ Complexity is 4N
  ■ Must create true physical checkerboard, not logical checkerboard
  ■ For SAF, DRF, shorts between cells, and half of the TFs
  ☐ Not good for AFs, and some CFs cannot be detected

```
1 0 1
0 1 0
1 0 1
```
Classical Test Algorithms: GALPAT

- Galloping Pattern (GALPAT)
  - Complexity is $4N^2$—only for characterization
  - A strong test for most faults: all AFs, TFs, CFs, and SAFs are detected and located

- Pseudo code of GALPAT

1. Write background 0;
2. For BC = 0 to N-1
   { Complement BC;
     For OC = 0 to N-1, OC != BC;
     { Read BC; Read OC; }
     Complement BC; }
3. Write background 1;
4. Repeat Step 2;
March Test

- An example of march test \{\uparrow (w1); \downarrow (r1, w0)\}

![Diagram showing the march test process]

Initial state

- Addressing cell 0
  - \(w_1\) transition:
    - \(1\ X\)
    - \(X\ X\)

- Addressing cell 1
  - Transition:
    - \(1\ 1\)
    - \(X\ X\)

- Addressing cell 2
  - Transition:
    - \(1\ 1\)
    - \(1\ X\)

- Addressing cell 3
  - Transition:
    - \(1\ 1\)
    - \(1\ 1\)

- Addressing cell 3
  - Transition:
    - \(1\ 1\)
    - \(1\ 0\)

- Addressing cell 2
  - Transition:
    - \(1\ 0\)
    - \(0\ 0\)

- Addressing cell 1
  - Transition:
    - \(0\ 0\)
    - \(0\ 0\)

- Addressing cell 0
  - Transition:
    - \(0\ 0\)
    - \(0\ 0\)
Detection of SAFs and TFs

- MATS+: \{\uparrow (w0); \downarrow (r0, w1); \downarrow (r1, w0)\}
- MATS+ detection of SA0 fault

```
0 0 0
0 0 0
0 0 0
```

Good memory after M0

```
1 1 1
1 1 1
1 1 1
```

Good memory after M1

```
0 0 0
0 0 0
0 0 0
```

Good memory after M2

```
0 0 0
0 0 0
0 0 0
```

Bad memory after M0

```
1 1 1
0 1 1
1 1 1
```

Bad memory after M1

```
0 0 0
0 0 0
0 0 0
```

Bad memory after M2

- MATS+ detection of TFu & TFd can be proved in the same way
Tests for Detecting SAFs & TFs

- Conditions for detecting SAFs & TFs
  - SAFs & TFs can be detected by a march test which contains the following two march elements (or single march element containing both elements)
  - (... \(w_0, r_0,...\)) to detect SA1 faults and TFd
  - (... \(w_1, r_1,...\)) to detect SA0 faults and TFu
Detection of CFs

- March C - : \{\text{l} (w0); \text{u} (r0, w1); \text{u} (r1, w0); \text{d} (r0, w1); \text{d} (r1, w0); \text{r} (r0)\}

- Detection of CFs

<table>
<thead>
<tr>
<th>M1 is executed</th>
<th>1 0 0</th>
<th>1 1 0</th>
<th>1 1 1</th>
<th>1 1 1</th>
<th>...</th>
<th>1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td></td>
<td>1 1 1</td>
</tr>
<tr>
<td>Cell 0 is</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Cell 8 is addressed</td>
<td></td>
</tr>
<tr>
<td>addressed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M3 is executed</th>
<th>0 0 0</th>
<th>0 0 0</th>
<th>0 0 0</th>
<th>0 0 0</th>
<th>...</th>
<th>1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 1</td>
<td>0 0 0</td>
<td>0 1 1</td>
<td>1 1 1</td>
<td></td>
<td>1 1 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Cell 0 is addressed</td>
<td></td>
</tr>
</tbody>
</table>

Cell 0 is addressed
Cell 1 is addressed
Cell 2 is addressed
Cell 3 is addressed
Cell 8 is addressed
Cell 7 is addressed
Cell 6 is addressed
Cell 5 is addressed

Tests for Detecting CFs

- Conditions for detecting CFs
  - A march test which contains one of the two pairs of march elements of Case A & Case B can detect simple CFs (CFin, CFst, CFid)

  - Case A
    1. \(\uparrow (rx, \ldots, wx) \uparrow (r\overline{x}, \ldots, wx)\)
    2. \(\downarrow (rx, \ldots, wx) \downarrow (rx, \ldots, wx)\)

  - Case B
    1. \(\uparrow (r\overline{x}, \ldots, wx) \uparrow (rx, \ldots, wx)\)
    2. \(\downarrow (r\overline{x}, \ldots, wx) \downarrow (rx, \ldots, wx)\)

- A.1 (A.2) will sensitize the CFs, and it will detect the fault, when the value of the fault effect is \(x' (x)\), by the \(rx\) (\(rx'\)) operation of the first (second) march element when the coupled cell has a higher (lower) address than the coupling cell
Detection of DRFs

- Data retention faults (DRFs)
  - DRF has two subtypes
    - A stored ‘1’ will become a ‘0’ after a time $T$
    - A stored ‘0’ will become a ‘1’ after a time $T$

- Conditions for detecting DRFs
  - Any march test can be extended to detect DRFs
  - The detection of each of the two DRF subtypes requires that a memory cell be written into the corresponding logic states

- If we are interested in detecting simple DRFs only
  - The delay elements can be placed between any two pairs of march elements, e.g., $\uparrow (rx,\cdots,wx)$ ; $\downarrow (rx,\cdots,wx)$
Tests for Detecting AFs

☐ Conditions for detecting AFs

<table>
<thead>
<tr>
<th>Condition</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\uparrow (r_x, \ldots, w_x)$</td>
</tr>
<tr>
<td>2</td>
<td>$\downarrow (r_x, \ldots, w_x)$</td>
</tr>
</tbody>
</table>

☐ Condition 1

- Read the value $x$ from cell 0, then write $x'$ to cell 0, ..., read the value $x$ from cell n-1, then write $x'$ to cell n-1

☐ Condition 2

- Read the value $x'$ from cell n-1, then write $x$ to cell n-1, ..., read the value $x'$ from cell 0, then write $x$ to cell 0
Tests for NPSFs

- Type 1 tiling neighborhood
  - The figure shows that a cell-2 as base cell
  - The deleted neighborhood of all base cells-2 is formed by a cell-0, a cell-1, a cell-3, and a cell-4
# Tests for NPSFs

- **Type 2 tiling neighborhood**
  - Similar to type 1 NPSFs tiling method

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Tests for NPSFs

- Two-group method for type 1 neighborhood
  - Based on the duality of cells: a cell is a base cell in one group while it is a deleted neighborhood cell in the other group

```
A  2  B  2  A  2  B  2  2  C  2  D  2  C  2  D
B  2  A  2  B  2  A  2  2  D  2  C  2  D  2  C
A  2  B  2  A  2  B  2  2  C  2  D  2  C  2  D
B  2  A  2  B  2  A  2  2  D  2  C  2  D  2  C
```

- This method can not extend to test type 2 NPSFs because it depends on the duality concept

```
1  A  1  B  1  A  1  B  C  1  D  1  C  1  D  1
1  B  1  A  1  B  1  A  D  1  C  1  D  1  C  1
1  A  1  B  1  A  1  B  C  1  D  1  C  1  D  1
1  B  1  B  1  A  1  A  D  1  C  1  D  1  C  1
```
March Tests for Word-Oriented RAMs

- Fault models for word-oriented memories (WOMs)
  - Only the class of memory cell array faults for bit-oriented memories (BOMs) has to be extended in order to cover WOMs

- The fault models for WOMs can be classified into two classes
  - Single-cell faults
    - SAFs, TFs, data retention faults (DRFs), etc.
  - Faults between memory cells
    - CFs

- Two classes of faults between memory cells for WOMs needed to be considered
Converting BOM Test to WOM Test

- Any given BOM march test can be converted to a WOM march test
  - With additional tests to cover intra-word faults
- A WOM march test is a concatenation of two march tests
  - \{Inter-word march test, intra-word march test\}
- The inter-word march test can directly be obtained from the BOM march test
  - Replace the bit-operation “r0”, “w0”, “r1”, and “w1” with the word-operation “rD”, “wD”, “rD’”, and “wD’”, where D is called data background
Converting BOM Test to WOM Test

- The intra-word faults can be detected by a single march element with different operations and data backgrounds
  - E.g., intra CFst can be covered by \((wd_1, rd_1, ..., wd_n, rd_n)\) with various data backgrounds (DBs)
  - Note that the DBs can be applied in any order

- The above intra-word test can be modified as follows, without any impact on the fault coverage
  - Extra Read operations can be added
  - The single march element can be divided into any number of march elements, and for each march element the addressing order can be chosen freely
Cocktail March Tests for WOM

- If you have a bit-oriented march test, then you can obtain a compact WOM test with
  - Replace the bit-operation “r0”, “w0”, “r1”, and “w1” with all-0 and all-1 data backgrounds
  - Concatenate a march element \((wd_1, rd_1, \ldots, wd_n, rd_n)\) for \(d=\{0101..01, 0011..11, \ldots\}\)

- For example, the March C- can be extended as follows to test a memory with 4-bit words

\[
\{(\downarrow (w0000) ; \uparrow (r0000, w1111) ; \uparrow (r1111, w0000) ; \\
\downarrow (r0000, w1111) ; \downarrow (r1111, w0000) ; \uparrow (r0000) ; \\
\uparrow (w0101, w1010, r1010, w0101, r0101) ; \\
\uparrow (w0011, w1100, r1100, w0011, r0011)\}
\]
Memory BIST/BISD Designs
General BIST Architecture
An Example of ROM BIST

![Diagram of ROM BIST System]

- **Counter**
- **ROM**
- **MISR**
- **Controller**
- **Status**
- **Go/No-Go**
Typical RAM BIST Architecture

- Test Controller
- Test Pattern Generator
- Comparator

Normal I/Os

Go/No-Go
FSM-Based BIST

- An example of the state diagram of controller

\[\text{S}_0\] -> NOT last address?
\[\text{S}_1\] -> NOT last address?
\[\text{S}_2\] -> NOT first address?
\[\text{S}_3\] -> NOT first address?
\[\text{S}_4\] -> NOT first address?
\[\text{S}_5\] -> NOT first address?
\[\text{S}_6\] -> NOT first address?
\[\text{S}_7\] -> End
Programmable RAM BIST

- An example of the programmable RAM BIST
FSM State Diagram of Controller

- Idle
- BRS=1
- Shift_cmd
- BSC=1
- Get_cmd
- BSC=0
- Apply
- ENA=1
- DONE=0
- DONE=1

Jin-Fu Li
ARES Lab. EE, NCU
Programmability

- The programmability can be achieved by using test command
- The test command format
  - U/D: ascending/descending address sequence
  - OP: test operations
    - For example, wa, rawa’, rawa’ra, warawa’ra’, etc.
  - Data backgrounds
- The width of each field affects the programmability of the BIST design
  - For example, if 4 bits are used for OP, then only 16 possible test operations can be generated
FSM State Diagram of TPG

Idle

Init

Ifetch

Exec

Dfetch

Compare

No-Go

DONE/GO

ENA=0

ENA=1

Null=1

Null=0

Error=0

Error=1
Shared Memory BIST Architecture
Multiple RAM Groups

- RAM 1
- RAM 2
- RAM k
- RAM N-1

- TPG
- Test Collar
- WSI
- WSO
- WSC
- 1500 Wrapper
- CTR
Multiple RAM Groups

TDI → TAP Controller → Instruction Register → Decoder → Data Register → CTR 

TCK → TAP Controller → Instruction Register → Decoder → Data Register → CTR

TMS → TAP Controller → Instruction Register → Decoder → Data Register → CTR

TRST → TAP Controller → Instruction Register → Decoder → Data Register → CTR

TDO

RAM 1: + Test Collar
RAM 2: + Test Collar
RAM k: + Test Collar
RAM N-1: + Test Collar

TPG
RAM BISD

- Embedded memory test and diagnosis is an important issue in SOC development
- BIST is a cost-effective solution, even is the best solution, for embedded memories
  - Low cost
  - At-speed testing
  - Low pin count overhead
- A BISD design for embedded memories includes
  - BIST
  - Diagnostic data receiver (DDR)
RAM BISD Architecture

BISD in Diagnosis Mode

- In diagnosis mode it can run user-specified march algorithm for test/diagnosis
- EOP format:

<table>
<thead>
<tr>
<th>Addr</th>
<th>Session</th>
<th>Syndrome</th>
</tr>
</thead>
</table>

- A sample of timing diagram is as follows