Yield and Reliability-Enhancement Techniques for Random Access Memories

Jin-Fu Li

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

- Background
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions
Outline

- Introduction
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions
Embedded Memory—Quality

- During manufacture
  - Yield
  - Exponential yield model
    \[ Y = e^{-\sqrt{AD}} \], where \( A \) and \( D \) denote the area and defect density, respectively

- After manufacture
  - Reliability

- During use
  - Soft error rate
An Explosion in Embedded Memories

- Hundreds of memory cores in a complex chip is common
- Memory cores usually represent a significant portion of the chip area

Dual-core AMD Opteron™ processor
Memory Repair

- Repair is one popular technique for memory yield improvement
- Memory repair consists of three basic steps
  - Test
  - Redundancy analysis
  - Repair delivery
Conventional Memory Repair Flow

Requirements:
1. Memory tester
2. Laser repair equipment

Disadvantages:
1. Time consuming
2. Expensive
Memory BISR Flow

**Required Circuit**

- **BIST**
  - Built-In Self-Test
  - Test

- **BISD**
  - Built-In Self-Diagnosis
  - Fault Location

- **BIRA**
  - Built-In Redundancy-Analyzer
  - Redundancy allocation

  - Reconfiguration
  - Swap Defective Cells
Typical Memory BISR Architecture

- Normal I/Os
- BIRA
- BIST
- Test Collar & Reconfiguration mechanism
- RAM
- Redundancy
Typical Memory BIST Architecture
Redundancy Scheme

Three typical redundancy schemes

Spare rows

Spare columns

Spare rows and Spare columns
Spare Column & Spare IO

Advanced Reliable Systems (ARES) Lab., EE. NCU
Jin-Fu Li
Reconfiguration Scheme

32 columns

Spare column

... Sense Amplifier

32 columns

Spare column

... Sense Amplifier

Decoder

10-bit data

Programming Module (Flash)

[M. Yarmaoka, et al., JSSC, 2002]
Types of Reconfiguration Schemes

- Three kinds of reconfiguration techniques
  - Soft reconfiguration
    - By programming FFs to store repair information
  - Firm reconfiguration
    - By programming non-volatile memories to store repair information
  - Hard (permanent) reconfiguration
    - Laser-blown or electrically-blown polysilicon or diffusion fuses
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| **Soft** | 1. Multi-time repair  
2. Low design overhead | 1. Some latent defects cannot be repaired  
2. Long repair setup time |
| **Firm** | 1. Multi-time repair  
2. Short repair setup time | 1. High-voltage programming circuit is required |
| **Hard** | 1. Short repair setup time | 1. One-time repair  
2. Specific technology is required |
Memory BISR Techniques

- Dedicated BISR scheme
  - A RAM has a self-contained BISR circuit
- Shared BISR scheme
  - Multiple RAMs share a BISR circuit
  - E.g., processor-based BISR scheme and IP-based BISR scheme
- BISR classification according to the capability of redundancy analysis
  - BISR with redundancy analysis capability
  - BISR without redundancy analysis capability
RAM BISR Using Redundant Words

Address, Data Input, Control

BIST

Mux

Fuse Box

Redundancy Logic

RAM

Mux

[ V. Schober, et. al, ITC01]
Redundancy Wrapper Logic

- The redundancy logic consists of two basic components
  - Spare memory words
  - Logic to program the address decoding
- The address comparison is done in the redundancy logic
  - The address is compared to the addresses that are stored in the redundancy word lines
- An overflow bit identifies that there are more failing addresses than possible repair cells
- The programming of the faulty addresses is done during the memory BIST or from the fuse box during memory setup
An Array of Redundant Word Lines

MBIST
- Address
- Write Data

F
- Address
- Expected Data

Fail
- Fail Address
- RAM Data

TDI
- FA Address Data
- FA Address Data
- FA Address Data

FO
- Overflow

TDO
- Data out

Address, Data Input, Control

RAM

Word Redundancy

[V. Schober, et. al, ITC01]
Applications of Redundancy Logic

- Faulty addresses can be streamed out after test completion. Then the fuse box is blown accordingly in the last step of the test.
  - This is called here hard repair
  - This is normally done at wafer level test
- Furthermore, the application can be started immediately after the memory BIST passes.
  - This is called here soft repair
Redundancy Word Line

[V. Schober, et. al, ITC01]
One-Bit Fuse Box

- One-bit fuse box contains a fuse bit and a scan flip flop for controlling and observing the fuse data
- Test_Update=0: the chain of inverters is closed (The value is latched)
- Test_Update=1: it is possible to set the internal node from TDO
- The ports TDI and TDO are activated at scan mode

![Diagram of One-Bit Fuse Box]

Reset cycle to read out the fuse information

[V. Schober, et. al, ITC01]
The fuse box can be connected to a scan register to stream in and out data during test and redundancy configuration.
Parallel Access of the Fuse Information

[V. Schober, et. al, ITC01]
Serial Access of the Fuse Information

Advanced Reliable Systems (ARES) Lab., EE. NCU

[ V. Schober, et. al, ITC01]
Test Flow to Activate the Redundancy

Initialization of the BIST
Load faulty addresses

Increment address
Access memory

Test finished?
No
Fail?
Yes
Free register?
No
Hard repair
Yes
Write expected data
Write address
Write Fail flag

Fuse to be blown?
No
Soft repair
Yes
Stream out faulty addresses

Unrepairable
Redundancy Analysis

- A repairable memory with 1D redundancy
  - Redundancy allocation is straightforward
- A repairable memory with 2D redundancy
  - Redundancy analysis (redundancy allocation) is needed

Redundancy analysis problem

- Choose the minimum number of spare rows and columns that cover all the faulty cells
BIRA Algorithm – CRESTA

- Comprehensive Real-time Exhaustive Search Test and Analysis
- Assume that a memory has 2 spare rows (Rs) & 2 spare columns (Cs), then all possible repair solutions
  - R-R-C-C (Solution 1)
  - R-C-R-C (Solution 2)
  - R-C-C-R (Solution 3)
  - C-R-R-C (Solution 4)
  - C-R-C-R (Solution 5)
  - C-C-R-R (Solution 6)

[ T. Kawagoe, et. al, ITC00]
CRESTA Flow Chart

Start

Test

Fail ?

S1    S2    S3    ...    S6

Finish ?

Result Output

End

[T. Kawagoe, et al, ITC00]
Heuristic BIRA Algorithms

- Most of heuristic BIRA algorithms need a local bitmap for storing the information of faulty cells detected by the BIST circuit
- An example of $4 \times 5$ local bitmap

<table>
<thead>
<tr>
<th>Address</th>
<th>C1</th>
<th>C1</th>
<th>C1</th>
<th>C1</th>
<th>C1</th>
<th>CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
A BIRA Flow for Performing Heuristic RAs

START

BIST

Fail?

No

Yes

Local Bitmap

Full?

Update Bitmap

No

Yes

Redundancy Allocation
Redundancy Allocation Rules

- Typical redundancy analysis algorithms
  - Two-phase redundancy allocation procedure: *must-repair phase* and *final-repair phase*

- Must-repair phase
  - Row-must repair (column-must repair): a repair solution forced by a failure pattern with \( >S_C \) \((>S_R)\) defective cells in a single row (column), where \( S_C \) and \( S_R \) denote the number of available spare columns and spare rows

- Final-repair phase
  - Heuristic algorithms are usually used, e.g., repair-most rule
Shared BISR Techniques

- A complex SOC usually has many RAMs with different sizes
- Each repairable RAM has a dedicated BISR circuit
  - Area cost is high
- If a BISR circuit can be shared by multiple RAMs, then the area cost of the BISR circuit can drastically be reduced
- Shared BISR techniques
  - Reconfigurable BISR or IP-based BISR technique
NCU/FTC BISR Scheme

- Reconfigurable BISR scheme for multiple RAMs

[ T. W. Tseng, et. al, ITC06 ]
Repair Process

**Test & Repair**
- BIST
- BIRA
- Load Repair Signatures into the Fuse Group
  - Pre-Fuse Testing
  - Program Fuse

**Normal Operation**
- Power-On
- Repair Signature Setup
- Normal Access
Test and Repair Mode

RAM 1
Wrapper

RAM 2
Wrapper

\ldots

RAM N-1
Wrapper

BIST
ReBIRA
ReBISR

RSO
Shift_en
Register
Fuse

Fuse Group

Advanced Reliable Systems (ARES) Lab., EE. NCU
Jin-Fu Li
Normal Mode

Advanced Reliable Systems (ARES) Lab., EE. NCU
Jin-Fu Li
NCU/FTC BISR Scheme

Reconfigurable BIRA architecture

[T. W. Tseng, et. al, ITC06]
Evaluation of Repair Efficiency

➢ Repair rate
  ➢ The ratio of the number of defective memories to the number of repaired memories

➢ A simulator was implemented to simulate the repair rate [R.-F. Huang, et. al, IEEE D&T, 2005 (accepted)]

➢ Simulation setup
  ➢ Simulated memory size: 4096x128
  ➢ Simulated memory samples: 500
  ➢ Poisson defect distribution is assumed
  ➢ Original yield is about 60%
Repair Rate

Case 1: 100% single-cell faults

![Graph showing repair rate for different (R,C) values. Red bars represent RCFA, black bars represent Opt.]
Case 2: 50% single-cell faults, 20% faulty rows, 20% faulty columns, and 10% column twin-bit faults
ReBISR Implementations

- FTC 0.13\(\mu m\) standard cell library is used
- Three cases are simulated

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td>64x2x8</td>
<td>64x2x16</td>
<td>64x2x32</td>
</tr>
<tr>
<td>Core 1</td>
<td>128x4x16</td>
<td>128x4x32</td>
<td>128x2x64</td>
</tr>
<tr>
<td>Core 2</td>
<td>256x8x32</td>
<td>256x8x64</td>
<td>256x4x128</td>
</tr>
<tr>
<td>Core 3</td>
<td>512x16x64</td>
<td>512x8x128</td>
<td>512x4x256</td>
</tr>
</tbody>
</table>
Simulation Results

- **Delay and area overhead**

<table>
<thead>
<tr>
<th>ReBIRA Parameter</th>
<th>Memory Area (um²)</th>
<th>ReBIRA Area (um²)</th>
<th>Ratio (%)</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x16x64</td>
<td>1496258.4</td>
<td>18766</td>
<td>1.25</td>
<td>2.5</td>
</tr>
<tr>
<td>512x8x128</td>
<td>1497561.6</td>
<td>20303</td>
<td>1.36</td>
<td>2.5</td>
</tr>
<tr>
<td>512x4x256</td>
<td>1528848</td>
<td>23255</td>
<td>1.52</td>
<td>2.5</td>
</tr>
</tbody>
</table>

- **BIRA time overhead w.r.t. a 14N March test with solid data background**

<table>
<thead>
<tr>
<th>ReBIRA Parameter</th>
<th>Repair Rate (%)</th>
<th>ReBIRA Cycles</th>
<th>BIST Cycles</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x16x64</td>
<td>83.6</td>
<td>29952</td>
<td>47939584</td>
<td>0.06</td>
</tr>
<tr>
<td>512x8x128</td>
<td>82.3</td>
<td>30698</td>
<td>23605658</td>
<td>0.13</td>
</tr>
<tr>
<td>512x4x256</td>
<td>83.8</td>
<td>30404</td>
<td>12013568</td>
<td>0.25</td>
</tr>
</tbody>
</table>
NCU/FTC BISR Scheme

- Layout view for an experimental case
Infrastructure IP

What is Infrastructure IP

Unlike the functional IP cores used in SOCs, the infrastructure IP cores do not add to the main functionality of the chip. Rather, they are intended to ensure the manufacturability of the SOC and to achieve lifetime reliability.

Examples of such infrastructure IPs

- Process monitoring IP
- Test & repair IP
- Diagnosis IP
- Timing measurement IP
- Fault tolerance IP
Infrastructure IP – STAR

STAR IIP

[Advanced Reliable Systems (ARES) Lab., EE. NCU Jin-Fu Li]
Infrastructure IP – STAR

- The infrastructure IP is comprised of a number of hardware components, including
  - A STAR processor, a fuse box, and intelligent wrappers (IWs)

- The STAR Processor
  - Performs all appropriate test & repair coordination of a STAR memory
  - It is programmed by a set of instructions to control the operation of the internal modules

- The Intelligent Wrapper
  - Address counters, registers, data comparators and multiplexers
Infrastructure IP – ProTaR

- **ProTaR** [C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]
  - Processor for Test and Repair of RAMs
- The infrastructure IP is comprised of a number of hardware components, including
  - A ProTaR processor
  - A wrapper
- **Features**
  - Parallel test and diagnosis
  - Serial repair
  - Support multiple redundancy analysis algorithms
Architecture of the Proposed IIP

[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]
Multiple Redundancy Analysis Algorithms Support

- In the IIP, the ProTaR has one global BIRA module and each wrapper has one local BIRA module.
- The local BIRA module performs the must-repair phase of a redundancy analysis algorithm.
- Then, the global BIRA module performs the final-repair phase of the redundancy analysis algorithm.
Global/Local Bitmaps and RA Instructions

- **Local bitmap**
  - CAR
    - R0: 1 0 0 0
    - R1: 0 1 0 1
    - R2: 0 1 0 0
    - R3: 0 0 1 0

- **Global bitmap**
  - CID
    - r0: 1 0 0 0
    - r1: 0 1 0 1
    - r2: 0 1 0 0
    - r3: 0 0 1 0

<table>
<thead>
<tr>
<th>RA algorithm</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local repair-most (LRM) alg.</td>
<td>LRM</td>
</tr>
<tr>
<td>Essential spare pivoting (ESP) alg.</td>
<td>{FHFR, ROW_FIRST, COL_FIRST}</td>
</tr>
<tr>
<td>Row first alg.</td>
<td>{ROW_FIRST, COL_FIRST}</td>
</tr>
<tr>
<td>Column first alg.</td>
<td>{COL_FIRST, ROW_FIRST}</td>
</tr>
</tbody>
</table>

[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Advanced Reliable Systems (ARES) Lab., EE. NCU

Jin-Fu Li
Block Diagram of the ProTaR

Advanced Reliable Systems (ARES) Lab., EE. NCU

Jin-Fu Li

[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]
Block Diagram of the Wrapper

[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Advanced Reliable Systems (ARES) Lab., EE. NCU

Jin-Fu Li
Area Cost of the Wrapper

- Area overhead of the Wrapper is defined as the ratio of the area of the wrapper to the area of the corresponding memory.

- Experimental results for an 8Kx64-bit memory:

<table>
<thead>
<tr>
<th>Redundancy Configuration</th>
<th>Wrapper Area</th>
<th>Area Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>2R2C</td>
<td>6739 gates</td>
<td>2.3%</td>
</tr>
<tr>
<td>2R3C</td>
<td>7342 gates</td>
<td>2.5%</td>
</tr>
<tr>
<td>3R3C</td>
<td>8317 gates</td>
<td>2.8%</td>
</tr>
</tbody>
</table>

- Area cost of Wrappers for different memory sizes:

<table>
<thead>
<tr>
<th>Memory Configuration</th>
<th>Wrapper Area</th>
<th>Area Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>8K x 16</td>
<td>3944 gates</td>
<td>4.6%</td>
</tr>
<tr>
<td>4K x 32</td>
<td>4825 gates</td>
<td>5.8%</td>
</tr>
<tr>
<td>2K x 64</td>
<td>6501 gates</td>
<td>7.1%</td>
</tr>
</tbody>
</table>
Area Cost of the IIP

- An IIP for four memories is implemented
  - The size of Mem0, Mem1, Mem2, and Mem3 are 8Kx64, 8Kx64, 4Kx14, and 2Kx32, respectively
  - The redundancy configurations of the Mem0, Mem1, Mem2, and Mem3 are (3x3), (2x2), (2x2), and (2x2), respectively
- The area of the four memories is 6798472um²
- The area of all the redundancies is about 896060um²
- The area of the IIP is only about 309893um²
- Thus, the area overhead of the IIP is only about 4.56%
Layout View of the IIP

- Layout view of the proposed IIP for four RAMs

[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]
Outline

- Introduction
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions
Reliability-Enhancement Techniques

- Fault-tolerant techniques are widely used to improve the reliability of systems
- All fault-tolerant techniques require redundancy
  - Redundancy is simply the addition of information, resources, or time beyond what is needed for normal system operation
- Types of redundancy
  - Hardware redundancy
  - Software redundancy
  - Information redundancy
  - Time redundancy
Memory Reliability-Enhancement Techniques

- Hardware redundancy
  - Built-in self-repair technique
- Error correction code
  - Use information redundancy to protect stored data from soft error
- Periodic transparent testing
  - Periodically apply tests to detect hard faults manifested by latent faults
Typical Error-Correction-Code Scheme

![Diagram of error-correction-code scheme]

- Check Bit Generator
- RAM
- Syndrome Generator
- Data redundancy
- Data
- Corrector
- Data Output
- Data Input
Hamming Error-Correction Code

- The Hamming single error-correction code uses $c$ parity check bits to protect $k$ bits of information. The relationship between the values of $c$ and $k$ is
  \[ 2^c \geq c + k + 1 \]

- Suppose that there are four information bits $(d_3, d_2, d_1, d_0)$ and, as a result, three parity check bits $(c_1, c_2, c_3)$. The bits are partitioned into groups as $(d_3, d_1, d_0, c_1)$, $(d_3, d_2, d_0, c_2)$, and $(d_3, d_2, d_1, c_3)$. Each check bit is specified to set the parity of its respective group, i.e.,
  \[
  c_1 = d_3 + d_1 + d_0 \\
  c_2 = d_3 + d_2 + d_0 \\
  c_3 = d_3 + d_2 + d_1
  \]
What is Transparent Test?

- Transparent testing
  - Leave the original content of the circuit under test unchanged after the testing is completed if no faults are presented

- Features
  - Ensure the reliability of stored data throughout its life time
  - Provide better fault coverage than non-transparent testing for unmodeled faults

- Limitation
  - Must be performed while systems are idle
Principle of Transparent Testing

1. Read (CONTENT), take signature S(CONTENT)
2. Read (CONTENT), Write (CONTENT. XOR. TP)=NEW_CONTENT
3. Read (NEW_CONTENT), take new signature S(NEW_CONTENT)
4. Write (NEW_CONTENT. XOR. TP)

NEW_CONTENT. XOR. TP=CONTENT. XOR. TP. XOR. TP=CONTENT

S(NEW_CONTENT)=S(CONTENT. XOR. TP)=S(CONTENT). XOR. S(TP)
Issues of Transparent Testing

- Test interrupts
  - In comparison with manufacturing testing, one special issue of transparent testing is that the transparent testing process may be interrupted.

- Aliasing
  - If a transparent built-in self-test scheme is considered, the signature generation typically is done by a MISR.

- Fault location
  - If a fault is detected, it is very difficult to locate the fault.
A Typical Transparent March Test

- A typical transparent March test consists of two-phase tests
  - Signature-prediction test
  - Transparent March test

- Types of transparent test schemes
  - Transparent March tests
  - Symmetric transparent March tests
  - Combination of Transparent March tests and ECCs
Notation

- In a test algorithm
  - $D$ denotes the initial content of a cell or a word for bit-oriented or word-oriented memories
  - $D_a$ is data of the bit-wise XOR operation on $D$ and $a$
  - $⇑(⇓)$ represents the ascending (descending) address sequence
  - $⇑$ denotes either ascending or descending address sequence
  - $wX$ denotes a write $X$ operation
  - $rX$ denotes a read operation with expect data $X$
A Typical Transformation Method

Bit-oriented march test

- Remove the initialization march element. Add a Read operation at the beginning of the first march element.

- Replace \( r_0 \) or \( r_1 \) with \( r_{Da} \) or \( r_{Da'} \).
- Replace \( w_0 \) or \( w_1 \) with \( w_{Da} \) or \( w_{Da'} \).

- If the content of memory cell is the inverse of initial data after the last Write operation, insert two additional Read and Write operations in the end of the march test.

Transparent bit-oriented march test

- Remove all Write operations

Signature prediction algorithm

[M. Nicolaidis, IEEE TC, 1996]
Consider the March C- test:

{↓ (w0); ↑ (r0, w1); ↑ (r1, w0); ↓ (r1, w0); ↓ (r0, w1); ↓ (r0)}

After Step 1 transformation:

{↑ (r0, w1); ↑ (r1, w0); ↓ (r1, w0); ↓ (r0, w1); ↓ (r0)}

After Step 2 transformation:

{↑ (rD_a, wD_a^-); ↑ (rD^-_a, wD_a); ↓ (rD_a, wD_a); ↓ (rD^-_a, wD_a); ↓ (rD_a)}

The content of memory cell after the last operation is the same as the initial state. Step 3 is omitted.

Thus, the transparent March C- test is as follows:

{↑ (rD_a, wD_a^-); ↑ (rD^-_a, wD_a); ↓ (rD_a, wD_a); ↓ (rD^-_a, wD_a); ↓ (rD_a)}

Remove the Write operations. The signature prediction algorithm is as follows:

{↑ (rD_a); ↑ (rD^-_a); ↓ (rD_a); ↓ (rD^-_a); ↓ (rD_a)}
Word-Oriented Transparent Tests

- Word-oriented transparent test can be obtained by applying the transformation rules to all the bits of each word [Nicolaids, ITC92].

- E.g., a word-oriented March C- for 4-bit words

  - T1:
    \[
    \{(w0000) ; \uparrow (r0000, w1111) ; \uparrow (r1111, w0000) ; \downarrow (r1111, w0000) ; \}
    \[
    \{(r0000, w1111) ; \downarrow (r0000) \} 
    \]
  
  - T2:
    \[
    \{(w0101) ; \uparrow (r0101, w0101) ; \uparrow (r1010, w0101) ; \downarrow (r0101, w1010) ; \}
    \[
    \{(r1010, w0101) ; \downarrow (r0101) \} 
    \]
  
  - T3:
    \[
    \{(w0011) ; \uparrow (r0011, w1100) ; \uparrow (r1100, w0011) ; \downarrow (r0011, w1100) ; \}
    \[
    \{(r1100, w0011) ; \downarrow (r0011) \} 
    \]

- Thus, the transparent word-oriented March C-

  - T1': \{(\uparrow (rD_{a0}, wD_{a0}) ; \uparrow (rD_{a1}, wD_{a1}) ; \downarrow (rD_{a0}, wD_{a0}) ; \downarrow (rD_{a1}, wD_{a1}) ; \}\}

  - T2': \{(\uparrow (rD_{a0}, wD_{a0}) ; \uparrow (rD_{a1}, wD_{a1}) ; \downarrow (rD_{a1}, wD_{a1}) ; \uparrow (rD_{a1}, wD_{a2}) \}\}

  - T3': \{(\uparrow (rD_{a2}, wD_{a2}) ; \uparrow (rD_{a2}, wD_{a2}) ; \downarrow (rD_{a2}, wD_{a2}) ; \uparrow (rD_{a2}, wD_{a0}) \}\}
Problem

- Transparent tests are usually applied in the idle state of systems or components
- Reducing the test time is very important
  - Avoiding the interrupt of testing
- However, conventional transparent word-oriented march tests are directly obtained
  - By executing the corresponding bit-oriented march test on each bit of word
- Thus, conventional transformation does not generate a time-efficiency word-oriented march test
Efficient Word-Oriented Transparent Tests

Bit-oriented march test

Replace the 0 or 1 of a bit-oriented march test with all-0 or all-1 data. Obtain a march test called SBMarch.

If the first operation of SBMarch is a Write operation, add a Read operation in the beginning of SBMarch.

TSMarch

Replace all \( r_0 \) or \( r_1 \) with \( r_{D_{a0}} \) or \( r_{\overline{D_{a0}}} \)
Replace \( w_0 \) or \( w_1 \) with \( w_{D_{a0}} \) or \( w_{\overline{D_{a0}}} \)

The last Write operation of TSMarch is \( w_{D_{a0}} \)

ATMarch\(=\) \{\\( (r_{D_{a0}}, TO_1, TO_2, \ldots, TO_{\lfloor \log_2 B \rfloor}, w_{D_{a0}}) \)\}
\( TO_i = (w_{D_{a_i}}, w_{D_{a_i}}, r_{D_{a_i}}, w_{D_{a_i}}, r_{D_{a_i}}) \)

The last Write operation of TSMarch is \( w_{D_{a0}} \)

ATMarch\(=\) \{\\( (r_{D_{a0}}, TO_1, TO_2, \ldots, TO_{\lfloor \log_2 B \rfloor}, w_{D_{a0}}) \)\}
\( TO_i = (w_{D_{a_i}}, w_{D_{a_i}}, r_{D_{a_i}}, w_{D_{a_i}}, r_{D_{a_i}}) \)

Transparent word-oriented march test= TSMarch+ATMarch

Source: J.-F. Li, IEEE TCAD, 2007 (accepted)
Example

- Consider a bit-oriented March U [15]
  - \{\overline{0}, (w0); \overline{1}, (r0, w1, r1, w0); \overline{1}, (r0, w1); \overline{1}, (r1, w0, r0, w1); \overline{1}, (r1, w0)\}

- Then, the solid March U (SBMarch U) is as follows
  - \{\overline{0}, (w\overline{0}); \overline{1}, (r\overline{0}, w\overline{1}, r\overline{1}, w\overline{0}); \overline{1}, (r\overline{0}, w\overline{1}); \overline{1}, (r\overline{1}, w\overline{0}, r\overline{0}, w\overline{1}); \overline{1}, (r\overline{1}, w\overline{0})\}
  - where \(\overline{0}\) and \(\overline{1}\) denote all-0 and all-1 data

- According to the transformation rules described above, the transparent SBMarch U (TSMarch U) is
  - \{\overline{1}, (rD_{a0}, wD_{a\overline{0}}, rD_{a\overline{0}}, wD_{a0}); \overline{1}, (rD_{a0}, wD_{a\overline{0}}); \overline{1}, (rD_{a\overline{0}}, wD_{a0}, rD_{a0}, wD_{a\overline{0}}); \overline{1}, (rD_{a\overline{0}}, wD_{a0})\}
  - where \(a0\) denotes all-0 data

- The last operation of TSMarch U is \(wD_{a0}\)
  - \(ATMarch = \overline{1}, (rD_{a0}, wD_{a1}, wD_{a\overline{1}}, rD_{a1}, wD_{a1}, rD_{a1}, wD_{a2}, wD_{a\overline{2}}, rD_{a2}, wD_{a2}, rD_{a2}, wD_{a\overline{0}}, wD_{a\overline{0}), rD_{a3}, wD_{a3}, rD_{a3}, wD_{a3}, rD_{a3}, wD_{a0}})\)
Symmetric Transparent Tests

- Feature
  - The symmetric transparent test method takes advantage of the symmetric characteristic of a signature analyzer to eliminate the signature prediction phase

- Symmetric characteristic of a signature analyzer
  - Let $\text{sig}(z, S, h)=u$ denote a serial signature analyzer which has an initial state $S$, a feedback polynomial $h$, a data string for analysis $z$, and the corresponding signature $u$. Then we can obtain $\text{sig}(z^*, u^*, h^*)=S^*$, where $z^*$, $u^*$, $h^*$, and $S^*$ denote the reverse of $z$, $u$, $h$, and $S$, respectively [V. N. Yarmolik and S. Hellebrand, DATE99]
An Example

- A 2n-bit data string \( Z=(x_{2n-1}x_{2n-2}...x_nx_{n-1}...x_1x_0) \) is called a symmetric data string if
  
  - \( x_{n-1}=x_n, x_{n-2}=x_{n+1}, \ldots, x_1=x_{2n-2}, x_0=x_{2n-1} \)
  
  - or \( x_{n-1}=x_n, x_{n-2}=x_{n+1}, \ldots, x_1=x_{2n-2}, x_0=x_{2n-1} \)

- Consider a symmetric data string \( Z=(zz^*) \).
  
  Assume that a reconfigurable signature analyzer \( \text{sig}(-,0,h) \) is used to analyze the symmetric data string \( Z \)
  
  - Step 1: \( z \) is analyzed and \( \text{sig}(z,0,h)=u \)
  
  - Step 2: analyzer is configured as \( \text{sig}(-,u^*,h^*) \)
  
  - Step 3: \( z^* \) is analyzed and \( \text{sig}(z^*,u^*,h^*)=0^*=0 \)
Symmetric Transparent March Tests

- A transparent March test is a symmetric transparent March test if the read data of the Read operations of the transparent March test is a symmetric data string $Z$.

- For example, consider the March test MATS$+$
  - $\{\uparrow (w0); \uparrow (r0, w1); \downarrow (r1, w0)\}$

- It can be transformed to a transparent March test
  - $\{\uparrow (rD_{a_0}, wD_{a_0}); \downarrow (rD_{a_0}, wD_{a_0})\}$

- The read data can be expressed as $Z=(z, z^{*c})$.
Limitations

- Symmetric transparent March tests have two major limitations
  - Fault masking effect
  - Test interrupts cause the symmetric characteristic to be invalid

- Consider a 4-bit memory with initial content \((d_0d_1d_2d_3)=(0100)\). Assume that the memory has an idempotent coupling fault in which the aggressor and victim are at \(d_0\) and \(d_1\). Also, the value of the victim is forced to 0 while the aggressor has a 0 to 1 transition
Fault Masking Effect

- Assume that the symmetric transparent MATS+ is used to test a 4-bit memory with a CFid
- Transparent MATS+: \{\uparrow (rD_{a_0}, wD_{a_0}); \downarrow (rD_{a_0}, wD_{a_0})\}

<table>
<thead>
<tr>
<th>$rD_{a_0}$</th>
<th>$d_0$</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0100]</td>
<td>[1000]</td>
<td>[1000]</td>
<td>[1110]</td>
<td></td>
</tr>
<tr>
<td>$wD_{a_0}$</td>
<td>[1000]</td>
<td>[1100]</td>
<td>[1111]</td>
<td></td>
</tr>
<tr>
<td>$d_0$ is addressed</td>
<td>$d_1$ is addressed</td>
<td>$d_3$ is addressed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$rD_{a_0}$</th>
<th>$d_0$</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1111]</td>
<td>[0111]</td>
<td>[0011]</td>
<td>[0001]</td>
<td></td>
</tr>
<tr>
<td>$wD_{a_0}$</td>
<td>[0111]</td>
<td>[0011]</td>
<td>[0000]</td>
<td></td>
</tr>
<tr>
<td>$d_0$ is addressed</td>
<td>$d_1$ is addressed</td>
<td>$d_3$ is addressed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Read data (0000) and Read data (1111)
Read: Read the data D at DO; Check if Code_Gen(D)=C
Write: Write the data D’. XOR. TP

Source: J.-F. Li, IEEE TCAD, 2007 (accepted)
Features

➢ No signature prediction phase is needed. This shortens the testing time such that the probability of an interruption is reduced.

➢ Really restoring the original content of the memory under test is achieved if the number of faulty bits of a word is less than the correction capability of the applied ECC.

➢ It can locate the faulty bit of the faulty word by the checking response. The fault location capability is also related to the correction capability of the applied ECC.
Consider a 3x4-bit memory with Hamming ECC. Also, the transparent March MATS+ is used to test the memory.

Transparent March MATS+: \{\uparrow (rD_{a_0}, wD_{a_0}); \downarrow (rD_{a_0}, wD_{a_0})\}

Assume that \(d_2\) of the first word has a stuck-at-0 fault.

\[
\begin{array}{cccc|ccc}
\text{W}_0 & \text{W}_1 & \text{W}_2 & d_3 & d_2 & d_1 & d_0 & h_2 & h_1 & h_0 \\
1001 & 1110 & 1010 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1110 & 1010 & 0110 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1010 & 0101 & 0100 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc|ccc}
\text{W}_0 & \text{W}_1 & \text{W}_2 & d_3 & d_2 & d_1 & d_0 & h_2 & h_1 & h_0 \\
1001 & 1110 & 1010 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0001 & 0101 & 1000 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0101 & 0110 & 0100 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc|ccc}
\text{W}_0 & \text{W}_1 & \text{W}_2 & d_3 & d_2 & d_1 & d_0 & h_2 & h_1 & h_0 \\
1001 & 1110 & 1010 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1110 & 1010 & 0110 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1010 & 0101 & 0100 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{align*}
h_0 &= d_3 + d_1 + d_0 \\
h_1 &= d_3 + d_2 + d_0 \\
h_2 &= d_3 + d_2 + d_1
\end{align*}
\]
Conclusions

- Embedded memories represent more and more area of system-on-chip (SOC) designs
  - The BISR technique is essential for memories in SOCs
- With the advent CMOS technology, enhancing the reliability of an integrated circuit becomes one major challenge
  - Effective and efficient reliability-enhancement techniques must be developed
References


