Overview

- Motivation and introduction
- Functional model of a memory
- A simple minded test and its limitations
- Fault models
- March tests and their capabilities
- Neighborhood tests
- Summary

Memory Cells Per Chip

<table>
<thead>
<tr>
<th>Size</th>
<th>Number of Test Algorithm Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
</tr>
<tr>
<td>1 Mb</td>
<td>0.06</td>
</tr>
<tr>
<td>4 Mb</td>
<td>0.25</td>
</tr>
<tr>
<td>16 Mb</td>
<td>1.01</td>
</tr>
<tr>
<td>64 Mb</td>
<td>4.03</td>
</tr>
<tr>
<td>256 Mb</td>
<td>16.11</td>
</tr>
<tr>
<td>1 Gb</td>
<td>64.43</td>
</tr>
<tr>
<td>2 Gb</td>
<td>128.9</td>
</tr>
</tbody>
</table>

Test Time in Seconds (Memory Size n Bits)

- Memory cycle time = 60 ns

Memory Test Levels
A simple minded test
for cell := 0 to n - 1 (or any other order) do
write 0 to A [cell];
read A [cell]; { Expected value = 0}
write 1 to A [cell];
read A [cell]; { Expected value = 1 }
end for;

What does this test achieve?
What kind of faults does it detect and it fault coverage?

<table>
<thead>
<tr>
<th>Fault</th>
<th>Functional fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAF</td>
<td>Cell stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>Driver stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>Read/write line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>Chip-select line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>Data line stuck</td>
</tr>
<tr>
<td>SAF</td>
<td>Open circuit in data line</td>
</tr>
<tr>
<td>CF</td>
<td>Short circuit between data lines</td>
</tr>
<tr>
<td>CF</td>
<td>Crosstalk between data lines</td>
</tr>
<tr>
<td>AF</td>
<td>Address line stuck</td>
</tr>
<tr>
<td>AF</td>
<td>Open circuit in address line</td>
</tr>
<tr>
<td>AF</td>
<td>Shorts between address lines</td>
</tr>
<tr>
<td>AF</td>
<td>Open circuit in decoder</td>
</tr>
<tr>
<td>AF</td>
<td>Wrong address access</td>
</tr>
<tr>
<td>AF</td>
<td>Multiple simultaneous address access</td>
</tr>
<tr>
<td>TF</td>
<td>Cell can be set to 0 (1) but not to 1 (0)</td>
</tr>
<tr>
<td>NPSF</td>
<td>Pattern sensitive cell interaction</td>
</tr>
</tbody>
</table>

Stuck-at Faults
- **Condition**: For each cell, must read a 0 and a 1.
- **Diagram**:
  - (a) State diagram of a good cell.
  - (b) SA0 fault.
  - (c) SA1 fault.

Transition Faults
- **Condition**: Each cell must undergo a transition and a transition, and be read after such, before undergoing any further transitions.
- **Diagram**: < |0> transition fault
Coupling Faults

- **Coupling Fault (CF):** Transition in bit \( j \) causes unwanted change in bit \( i \)
- **2-Coupling Fault:** Involves 2 cells, special case of \( k \)-Coupling Fault
  - Must restrict \( k \) cells to make practical
- **Inversion and Idempotent CFs:** Special cases of 2-Coupling Faults
- **Bridging and State Coupling Faults:** Involve any # of cells, caused by logic level
- **Dynamic Coupling Fault (CF\(_{\text{dyn}}\):** Read or write on \( j \) forces \( i \) to 0 or 1

March Test Notation

- \( r0 \) -- Read a 0 from a memory location
- \( r1 \) -- Read a 1 from a memory location
- \( w0 \) -- Write a 0 to a memory location
- \( w1 \) -- Write a 1 to a memory location

- \( r \) -- Write a 1 to a cell containing 0
- \( w \) -- Write a 0 to a cell containing 1

March Test Notation (Continued)

- \( \downarrow \) -- Complement the cell contents
- \( \uparrow \) -- Increasing memory addressing
- \( \downarrow \) -- Decreasing memory addressing
- \( \uparrow \) -- Either increasing or decreasing

Address Decoder Faults (ADFs)

- **Address decoding error assumptions:**
  - Decoder does not become sequential
  - Same behavior during both read & write
- **Multiple ADFs must be tested for**

MATS+ March Test

**M0:** {March element \( \downarrow (w0) \)}
- for cell := 0 to \( n - 1 \) (or any other order)
- write 0 to \( A_{[cell]} \)

**M1:** {March element \( \uparrow (r0, w1) \)}
- for cell := 0 to \( n - 1 \)
- read \( A_{[cell]} \); {Expected value = 0}
- write 1 to \( A_{[cell]} \)

**M2:** {March element \( \downarrow (r1, w0) \)}
- for cell := \( n - 1 \) down to 0
- read \( A_{[cell]} \); {Expected value = 1}
- write 0 to \( A_{[cell]} \)

Theorem 9.2

- A March test satisfying conditions 1 & 2 detects all address decoder faults.
- ... Means any # of read or write operations
- Before condition 1, must have \( w_x \) element
  - \( x \) can be 0 or 1, but must be consistent in test

<table>
<thead>
<tr>
<th>Condition</th>
<th>March element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( (r_x, \ldots, w_x) )</td>
</tr>
<tr>
<td>2</td>
<td>( (r_x, \ldots, w_x) )</td>
</tr>
</tbody>
</table>
**Proof Illustration**

Combinations that must be tested

![Diagram](image)

**Conditions for proof**

- Fault A
- Fault B
- Fault C
- Fault D

**Necessity Proof**

- Removing \( r_x \) from Condition 1 prevents A or B fault detection when \( x \) read
- Removing \( r_x \) from Condition 2 prevents A or B fault detection when \( x \) read
- Removing \( r_x \) or \( w_x \) from Condition 1 misses fault \( D_2 \)
- Removing \( r_x \) or \( w_x \) from condition 2 misses fault \( D_3 \)
- Removing both writes misses faults C and D1

**Sufficiency Proof**

- **Faults A and B**: Detected by SAF test
- **Fault C**: Initialize memory to \( h \) (or \( \bar{h} \)). Subsequent March element that reads \( h \) and writes \( \bar{h} \) detects Fault C.
  - Marching writes \( h \) to \( A_i \). Detection: read \( A_i \)
- **Fault D**: Memory returns random result when multiple cells read simultaneously. Generate fault by writing \( A_j \). Detection: read \( A_j \) (or marches)

**Irredundant March Tests**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATS</td>
<td>{ (w0); (r0, w1); (r1) }</td>
</tr>
<tr>
<td>MATS+</td>
<td>{ (w0); (r0, w1); (r1, w0) }</td>
</tr>
<tr>
<td>MATS++</td>
<td>{ (w0); (r0, w1); (r1, w0, r0) }</td>
</tr>
<tr>
<td>MARCH X</td>
<td>{ (w0); (r0, w1); (r1, w0); (r0) }</td>
</tr>
<tr>
<td>MARCH C—</td>
<td>{ (w0); (r0, w1, w0, w1); (r1, w0, w1); (r0, w1, w0) }</td>
</tr>
<tr>
<td>MARCH A</td>
<td>{ (w0); (r0, w1, w0, w1); (r1, w0, w1); (r0, w1, w0) }</td>
</tr>
<tr>
<td>MARCH Y</td>
<td>{ (w0); (r0, w1, r1, w0, r0); (r0) }</td>
</tr>
<tr>
<td>MARCH B</td>
<td>{ (w0); (r0, w1, r1, w0, r0); (r0) }</td>
</tr>
</tbody>
</table>

**Irredundant March Test Summary**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SAF</th>
<th>AF</th>
<th>TF</th>
<th>CF</th>
<th>CF</th>
<th>CF</th>
<th>SCF</th>
<th>Linked</th>
<th>in</th>
<th>id</th>
<th>dyn</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATS</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MATS++</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MARCH X</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MARCH C—</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MARCH A</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Some</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MARCH Y</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Some</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>MARCH B</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Some</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

**March Test Complexity**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATS</td>
<td>4n</td>
</tr>
<tr>
<td>MATS+</td>
<td>5n</td>
</tr>
<tr>
<td>MATS++</td>
<td>6n</td>
</tr>
<tr>
<td>MARCH X</td>
<td>6n</td>
</tr>
<tr>
<td>MARCH C—</td>
<td>10n</td>
</tr>
<tr>
<td>MARCH A</td>
<td>15n</td>
</tr>
<tr>
<td>MARCH Y</td>
<td>8n</td>
</tr>
<tr>
<td>MARCH B</td>
<td>17n</td>
</tr>
</tbody>
</table>
Neighborhood Pattern Sensitive Coupling Faults

Notation
- ANPSF -- Active Neighborhood Pattern Sensitive Fault
- APNPSF – Active and Passive Neighborhood PSF
- Neighborhood -- Immediate cluster of cells whose pattern makes base cell fail
- NPSF – Neighborhood Pattern Sensitive Fault
- PNPSF -- Passive Neighborhood PSF
- SNPSF -- Static Neighborhood Pattern Sensitive Fault

Type 1 Active NPSF
- **Active**: Base cell changes when one deleted neighborhood cell transitions
- **Condition for detection & location**: Each base cell must be read in state 0 and state 1, for all possible deleted neighborhood pattern changes.

Type 2 Active NPSF
- **Used when diagonal couplings are significant, and do not necessarily cause horizontal/vertical coupling**

Passive NPSF
- **Passive**: A certain neighborhood pattern prevents the base cell from changing
- **Condition for detection and location**: Each base cell must be written and read in state 0 and in state 1, for all deleted neighborhood pattern changes.
Static NPSF

- Static: Base cell forced into a particular state when deleted neighborhood contains particular pattern.
- Differs from active -- need not have a transition to sensitive NPSF
- Condition for detection and location: Apply all 0 and 1 combinations to k-cell neighborhood, and verify that each base cell was written.

Type 1 Tiling Neighborhoods

- Write changes \( k \) different neighborhoods
- Tiling Method: Cover all memory with non-overlapping neighborhoods

Two Group Method

- Only for Type-1 neighborhoods
- Use checkerboard pattern, cell is simultaneously a base cell in group 1, and a deleted neighborhood cell in 2

RAM Tests for Layout-Related Faults

Inductive Fault Analysis:
1. Generate defect sizes, location, layers based on fabrication line model
2. Place defects on layout model
3. Extract defective cell schematic & electrical parameters
4. Evaluate cell testing

Memory Testing Summary

- Multiple fault models are essential
- Combination of tests is essential:
  - March – SRAM and DRAM
  - NPSF – DRAM
  - DC Parametric – Both
  - AC Parametric – Both
- Inductive Fault Analysis is now required

Summary

- Functional and fault model of memory
  - Many fault models
- March tests and their capabilities
  - Variety of tests
- Neighborhood pattern sensitive tests
  - Variety of fault models and tests
Density and Defect Trends

- 1970 -- DRAM Invention (Intel) 1024 bits
- 1993 -- 1st 256 MBit DRAM papers
- 1997 -- 1st 256 MBit DRAM samples
  - \( 1 \text{ bit} \rightarrow 120 \times 10^{-6} \text{ bit} \)
- Kilburn -- Ferranti Atlas computer (Manchester U.) -- Invented Virtual Memory
  - 1997 -- Cache DRAM -- SRAM cache + DRAM now on 1 chip

Faults

- **System** -- Mixed electronic, electromechanical, chemical, and photonic system (MEMS technology)
- **Failure** -- Incorrect or interrupted system behavior
- **Error** -- Manifestation of fault in system
- **Fault** -- Physical difference between good & bad system behavior

Fault Types

- **Fault types:**
  - Permanent -- System is broken and stays broken the same way indefinitely
  - Transient -- Fault temporarily affects the system behavior, and then the system reverts to the good machine -- time dependency, caused by environmental condition
  - Intermittent -- Sometimes causes a failure, sometimes does not

Failure Mechanisms

- **Permanent faults:**
  - Missing/Added Electrical Connection
  - Broken Component (IC mask defect or silicon-to-metal connection)
  - Burnt-out Chip Wire
  - Corroded connection between chip & package
  - Chip logic error (Pentium division bug)

Failure Mechanisms (Continued)

- **Transient Faults:**
  - Cosmic Ray
  - An \( \alpha \) particle (ionized Helium atom)
  - Air pollution (causes wire short/open)
  - Humidity (temporary short)
  - Temperature (temporary logic error)
  - Pressure (temporary wire open/short)
  - Vibration (temporary wire open)
  - Power Supply Fluctuation (logic error)
  - Electromagnetic Interference (coupling)
  - Static Electrical Discharge (change state)
  - Ground Loop (misinterpreted logic value)
Failure Mechanisms (Continued)

• Intermittent Faults:
  – Loose Connections
  – Aging Components (changed logic delays)
  – Hazards and Races in critical timing paths (bad design)
  – Resistor, Capacitor, Inductor variances (timing faults)
  – Physical Irregularities (narrow wire -- high resistance)
  – Electrical Noise (memory state changes)

Physical Failure Mechanisms

• Corrosion
• Electromigration
• Bonding Deterioration -- Au package wires interdiffuse with Al chip pads
• Ionic Contamination -- Na⁺ diffuses through package and into FET gate oxide
• Alloying -- Al migrates from metal layers into Si substrate
• Radiation and Cosmic Rays -- 8 MeV, collides with Si lattice, generates n-p pairs, causes soft memory error

Fault Modeling

• Behavioral (black-box) Model -- State machine modeling all memory content combinations -- Intractable
• Functional (gray-box) Model -- Used
• Logic Gate Model -- Not used Inadequately models transistors & capacitors
• Electrical Model -- Very expensive
• Geometrical Model -- Layout Model -- Used with Inductive Fault Analysis

Reduced Functional Model (van de Goor)

• n Memory bits, B bits/word, n/B addresses
• Access happens when Address Latch contents change
• Low-order address bits operate column decoder, high-order operate row decoder
• read -- Precharge bit lines, then activate row
• write -- Keep driving bit lines during evaluation
• Refresh -- Read all bits in 1 row and simultaneously refresh them

Inversion Coupling Faults (CFin)

• \( | \) or \( | \) in cell\( j \) inverts contents of cell\( i \)

• Condition: For all cells that are coupled, each should be read after a series of possible CFins may have occurred, and the # of coupled cell transitions must be odd (to prevent the CFins from masking each other).

Good Machine State Transition Diagram
CFin State Transition Diagram

Idempotent Coupling Faults (CFid)

- \( | or | \) transition in \( j \) sets cell \( i \) to 0 or 1
- **Condition**: For all coupled faults, each should be read after a series of possible CFids may have happened, such that the sensitized CFids do not mask each other.
- **Asymmetric**: coupled cell only does \( | or | \)
- **Symmetric**: coupled cell does both due to fault
- \( < 0; 0>, < 0; 1>, < 1; 0>, < 1; 1> \)

CFid Example

Dynamic Coupling Faults (CFdyn)

- Read or write in cell of 1 word forces cell in different word to 0 or 1
- \( < r0 | w0 ; 0>, < r0 | w0 ; 1>, < r1 | w1 ; 0>, < r1 | w1 ; 1> \)
  - \( | \) Denotes “OR” of two operations
- More general than CFid, because a CFdyn can be sensitized by any read or write operation

Bridging Faults

- Short circuit between 2+ cells or lines
- 0 or 1 state of coupling cell, rather than coupling cell transition, causes coupled cell change
- Bidirectional fault -- \( i \) affects \( j \), \( j \) affects \( i \)
- **AND Bridging Faults (ABF)**: \( < 0,0 / 0,0>, <0,1 / 0,0>, <1,0 / 0,0>, <1,1 / 1,1> \)
- **OR Bridging Faults (OBF)**: \( < 0,0 / 0,0>, <0,1 / 1,1>, <1,0 / 1,1>, <1,1 / 1,1> \)

State Coupling Faults

- **Coupling cell / line \( j \) is in a given state that forces coupled cell / line \( i \) into state \( x \)**
- \( < 0;0>, < 0;1>, < 1;0>, < 1;1> \)

(b) Diagram of a state coupling fault (SCF) <\( 1; 1 \)>
Fault Modeling Example 1

Fault Modeling Example 2

Multiple Fault Models

• Coupling Faults: In real manufacturing, any # can occur simultaneously
• Linkage: A fault influences behavior of another
• Example March test that fails:
  \[ \{ (w0); (r0, w1); (w0, w1); (r1) \} \]
  \[ \{ x; 1x \} \]
  \[ \{ x; 0x \} \]
  \( (a) \) Unlinked faults
  \( (b) \) Linked faults

Fault Hierarchy

Tests for Linked AFs

- Cases 1, 2, 3 & 5 -- Unlinked
- Cases 4 & 6 -- Linked

DRAM/SRAM Fault Modeling

<table>
<thead>
<tr>
<th>DRAM or SRAM Faults</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shorts &amp; opens in memory cell array</td>
<td>SAF, SCF</td>
</tr>
<tr>
<td>Shorts &amp; opens in address decoder</td>
<td>AF</td>
</tr>
<tr>
<td>Access time failures in address decoder</td>
<td>Functional</td>
</tr>
<tr>
<td>Coupling capacitances between cells</td>
<td>CF</td>
</tr>
<tr>
<td>Bit line shorted to word line</td>
<td>IDDQ</td>
</tr>
<tr>
<td>Transistor gate shorted to channel</td>
<td>IDDQ</td>
</tr>
<tr>
<td>Transistor stuck-open fault</td>
<td>SOF</td>
</tr>
<tr>
<td>Pattern-sensitivity fault</td>
<td>PSF</td>
</tr>
<tr>
<td>Diode-connected transistor 2 cell short</td>
<td></td>
</tr>
<tr>
<td>Open transistor drain</td>
<td></td>
</tr>
<tr>
<td>Gate oxide short</td>
<td></td>
</tr>
<tr>
<td>Bridging fault</td>
<td></td>
</tr>
</tbody>
</table>
SRAM Only Fault Modeling

<table>
<thead>
<tr>
<th>Faults found only in SRAM</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-circuited pull-up device</td>
<td>DRF</td>
</tr>
<tr>
<td>Excessive bit line coupling capacitance</td>
<td>CF</td>
</tr>
</tbody>
</table>

DRAM Only Fault Modeling

<table>
<thead>
<tr>
<th>Faults only in DRAM</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data retention fault (sleeping sickness)</td>
<td>DRF</td>
</tr>
<tr>
<td>Refresh line stuck-at fault</td>
<td>SAF</td>
</tr>
<tr>
<td>Bit-line voltage imbalance fault</td>
<td>PSF</td>
</tr>
<tr>
<td>Coupling between word and bit line</td>
<td>CF</td>
</tr>
<tr>
<td>Single-ended bit-line voltage shift</td>
<td>PSF</td>
</tr>
<tr>
<td>Precharge and decoder clock overlap</td>
<td>AF</td>
</tr>
</tbody>
</table>

Functional RAM Testing with March Tests

- March Tests can detect AFs -- NPSF Tests Cannot
- Conditions for AF detection:
  - Need (r, x, w, x)
  - Need (r, x, w, x)

MATS+ Example

Cell (2,1) SA0 Fault

MATS+:

\[
\{ M0: \downarrow (w0); M1: \downarrow (r0, w1); M2: \downarrow (r1, w0) \}
\]

Cell (2,1) SA1 Fault

MATS+:

\[
\{ M0: \downarrow (w0); M1: \downarrow (r0, w1); M2: \downarrow (r1, w0) \}
\]

Multiple AF Type C

- Cell (2,1) is not addressable
- Address (2,1) maps into (3,1) & vice versa
- Can’t write (2,1), read (2,1) gives random #