Memory Test

- Memory market and memory complexity
- Notation
- Faults and failures
- MATS+ March Test
- Memory fault models
- March test algorithms
- Inductive fault analysis
- Summary

Test Time in Seconds (Memory Size n Bits)

<table>
<thead>
<tr>
<th>Size</th>
<th>Number of Test Algorithm Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n×log₂n</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>

Notation

- 0 – A cell is in logical state 0
- 1 – A cell is in logical state 1
- X – A cell is in logical state X
- A – A memory address
- ABF – AND Bridging Fault
- AF – Address Decoder Fault
- B – Memory # bits in a word
- BF – Bridging Fault
- C – A Memory Cell
- CF – Coupling Fault
- CFdyn – Dynamic Coupling Fault
- CFid – Idempotent Coupling Fault
- CFin – Inversion Coupling Fault
- coupling cell – cell whose change causes another cell to change
- coupled cell – cell forced to change by a coupling cell
- DRF – RAM Data Retention Fault
- K – Size of a neighborhood
- M – memory cells, words, or address set
- n – # of Memory bits
- N – Number of address bits: a = 2ⁿ
- NPSF – Neighborhood Pattern Sensitive Fault
March Test Notation

- \( r \) -- Read a memory location
- \( w \) -- Write a memory location
- \( r_0 \) -- Read a 0 from a memory location
- \( r_1 \) -- Read a 1 from a memory location
- \( w_0 \) -- Write a 0 to a memory location
- \( w_1 \) -- Write a 1 to a memory location
- \( \uparrow \) -- Write a 1 to a cell containing 0
- \( \downarrow \) -- Write a 0 to a cell containing 1
- \( \uparrow \) -- Complement the cell contents
- \( \downarrow \) -- Increasing memory addressing
- \( \uparrow \) -- Decreasing memory addressing
- \( \uparrow \) -- Either increasing or decreasing

More March Test Notation

- \( \forall \) -- Any write operation
- \( < \ldots > \) -- Denotes a particular fault, ...
- \( \langle \, / \, F \, \rangle \) -- \( i \) is the fault sensitizing condition, \( F \) is the faulty cell value
- \( \langle i_1, \ldots, i_n-1 ; \, \text{In} / \, F \, \rangle \) -- Denotes a fault covering \( n \) cells
  - \( i_1, \ldots, i_n \) are fault sensitization conditions in cells \( 1 \) through \( n - 1 \) for cell \( n \n \)
  - \( \text{In} \) gives sensitization condition for cell \( n \)
  - If \( \text{In} \) is empty, write \( \text{In} / \, F \) as \( F \)

MATS+ March Test

\( M_0 \): (March element \( \langle \, (w_0) \, \rangle \))
  for \( \text{cell} := 0 \) to \( n - 1 \) (or any other order) do
  write 0 to \( A[\text{cell}] \);

\( M_1 \): (March element \( \langle r_0, w_1 \rangle \))
  for \( \text{cell} := 0 \) to \( n - 1 \) do
  read \( A[\text{cell}] \); \{ Expected value = 0 \}
  write 1 to \( A[\text{cell}] \);

\( M_2 \): (March element \( \forall (r_1, w_0) \))
  for \( \text{cell} := n - 1 \) down to 0 do
  read \( A[\text{cell}] \); \{ Expected value = 1 \}
  write 0 to \( A[\text{cell}] \);

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

Functional Model
**Simplified Functional Model**

- Address
  - Address Decoder
  - Memory Cell Array
  - Read/Write Logic
  - Data

**Subset Functional Faults**

<table>
<thead>
<tr>
<th>Functional fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>a - Cell stuck</td>
</tr>
<tr>
<td>b - Driver stuck</td>
</tr>
<tr>
<td>c - Read/write line stuck</td>
</tr>
<tr>
<td>d - Chip-select line stuck</td>
</tr>
<tr>
<td>e - Data line stuck</td>
</tr>
<tr>
<td>f - Open circuit in data line</td>
</tr>
<tr>
<td>g - Short circuit between data lines</td>
</tr>
<tr>
<td>h - Crosstalk between data lines</td>
</tr>
</tbody>
</table>

**Subset Functional Faults (Continued)**

<table>
<thead>
<tr>
<th>Functional fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>i - Address line stuck</td>
</tr>
<tr>
<td>j - Open circuit in address line</td>
</tr>
<tr>
<td>k - Shorts between address lines</td>
</tr>
<tr>
<td>l - Open circuit in decoder</td>
</tr>
<tr>
<td>m - Wrong address access</td>
</tr>
<tr>
<td>n - Multiple simultaneous address access</td>
</tr>
<tr>
<td>o - Cell can be set to 0 but not to 1 (or vice versa)</td>
</tr>
<tr>
<td>p - Pattern sensitive cell interaction</td>
</tr>
</tbody>
</table>

**Reduced Functional Faults**

<table>
<thead>
<tr>
<th>Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAF</td>
</tr>
<tr>
<td>TF</td>
</tr>
<tr>
<td>CF</td>
</tr>
<tr>
<td>NPSF</td>
</tr>
</tbody>
</table>

**Stuck-at Faults**

- Condition: For each cell, must read a 0 and a 1.
- $\forall a \forall b$ ($\forall a \forall b$)

(a) State diagram of a good cell

(b) SA0 fault

(c) SA1 fault

**Transition Faults**

- Cell fails to make $0 \rightarrow 1$ or $1 \rightarrow 0$ transition
- Condition: Each cell must undergo an $\uparrow$ transition and a $\downarrow$ transition, and be read after such, before undergoing any further transitions.
- $\uparrow a \downarrow a$ 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$_{dyn}$) – Read or write on $j$ forces $i$ to 0 or 1.

**Inversion Coupling Faults (CF$_{in}$)**

- $i \rightarrow j$ in cell $i$ inverts contents of cell $j$.
- Condition: For all cells that are coupled, each should be read after a series of possible CF$_{in}$s may have occurred, and the # of coupled cell transitions must be odd (to prevent the CF$_{in}$s from masking each other).

$\langle \uparrow i; \downarrow j \rangle$ and $\langle \downarrow i; \uparrow j \rangle$

**Good Machine State Transition Diagram**

**Inversion Coupling Faults (CF$_{in}$)**

- $i \rightarrow j$ in cell $i$ inverts contents of cell $j$.
- Condition: For all cells that are coupled, each should be read after a series of possible CF$_{in}$s may have occurred, and the # of coupled cell transitions must be odd (to prevent the CF$_{in}$s from masking each other).

$\langle \uparrow i; \downarrow j \rangle$ and $\langle \downarrow i; \uparrow j \rangle$

**Idempotent Coupling Faults (CF$_{id}$)**

- $i \rightarrow j$ transition in $j$ sets cell $i$ to 0 or 1.
- Condition: For all coupled faults, each should be read after a series of possible CF$_{id}$s may have happened, such that the sensitize CF$_{id}$s do not mask each other.
- Asymmetric: coupled cell only does $\uparrow$ or $\downarrow$.
- Symmetric: coupled cell does both due to fault.

$\langle \uparrow i; \downarrow j \rangle, \langle \downarrow i; \uparrow j \rangle, \langle \downarrow j; \uparrow i \rangle, \langle \uparrow j; \downarrow i \rangle$

**CF$_{id}$ Example**

(b) State diagram of an $\langle \uparrow i; \downarrow j \rangle$ CF$_{id}$.

(c) State diagram of an $\langle \uparrow i; \downarrow j \rangle$ CF$_{id}$. 


**Dynamic Coupling Faults (CF\text{dyn})**

- 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>`, and `<r1 | w1 ; 1>`
- Denotes “OR” of two operations
- More general than CF\text{id}, because a CF\text{dyn} 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 y 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>`

**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
- Decoders have CMOS stuck-open faults

**Address Decoder Faults**

<table>
<thead>
<tr>
<th>A_y</th>
<th>C_x</th>
<th>A_y</th>
<th>C_x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault 1</td>
<td>Fault 2</td>
<td>Fault 2</td>
<td>Fault 4</td>
</tr>
<tr>
<td>No Cell Accessed for A_y</td>
<td>No Address to Access cell C_x</td>
<td>Multiple Cells Accessed with A_y</td>
<td>Multiple Addresses for Cell C_x</td>
</tr>
</tbody>
</table>

**Fault Hierarchy**

Diagram showing fault hierarchy with various types of faults and their relationships.
**Functional RAM Testing with March Tests**

- March Tests can detect AFs — NPSF Tests Cannot
- Conditions for AF detection:
  - Need \( (r_x, w_x) \)
  - Need \( (r_x, w_x) \)
- In the following March tests, addressing orders can be interchanged

**Irredundant March Tests**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATS</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1) } \end{array} )</td>
</tr>
<tr>
<td>MATS+</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0) } \end{array} )</td>
</tr>
<tr>
<td>MATS++</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_0, r_1) } \end{array} )</td>
</tr>
<tr>
<td>MARCH X</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0); \downarrow (r_0) } \end{array} )</td>
</tr>
<tr>
<td>MARCH C–</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0); \downarrow (r_0, r_1) } \end{array} )</td>
</tr>
<tr>
<td>MARCH A</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0); \downarrow (r_0, r_1) } \end{array} )</td>
</tr>
<tr>
<td>MARCH Y</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1, r_1); \downarrow (r_1, w_0, r_0); \downarrow (r_0, w_1) } \end{array} )</td>
</tr>
<tr>
<td>MARCH B</td>
<td>( \begin{array}{l} { \uparrow (w_0); \downarrow (r_0, w_1, r_1); \downarrow (r_1, w_0, r_0); \downarrow (r_0, w_1, r_0) } \end{array} )</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>

**MATS+ Example**

**Cell (2, 1) SA0 Fault**

(a) Good machine after M0.
(b) Good machine after M1.
(c) Good machine after M2.
(d) Bad machine after M0.
(e) Bad machine after M1.
(f) Bad machine after M2.

**MATS+**

\( \{ M_0: \uparrow (w_0); M_1: \downarrow (r_0, w_1); M_2: \downarrow (r_1, w_0) \} \)

**Cell (2, 1) SA1 Fault**

(a) Good machine after M0.
(b) Good machine after M1.
(c) Good machine after M2.
(d) Bad machine after M0.
(e) Bad machine after M1.
(f) Bad machine after M2.

**MATS+**

\( \{ M_0: \uparrow (w_0); M_1: \downarrow (r_0, w_1); M_2: \downarrow (r_1, w_0) \} \)

**MATS+ Example**

**Cell (2, 1) SA1 Fault**

(a) Good machine after M0.
(b) Good machine after M1.
(c) Good machine after M2.
(d) Bad machine after M0.
(e) Bad machine after M1.
(f) Bad machine after M2.

**MATS+**

\( \{ M_0: \uparrow (w_0); M_1: \downarrow (r_0, w_1); M_2: \downarrow (r_1, w_0) \} \)

**MATS+ Example**

**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 #

(a) Good machine after M0.
(b) Good machine after M1.
(c) Good machine after M2.
(d) Bad machine after M0.
(e) Bad machine after M1.
(f) Bad machine after M2.

**MATS+**

\( \{ M_0: \uparrow (w_0); M_1: \downarrow (r_0, w_1); M_2: \downarrow (r_1, w_0) \} \)
Pattern Sensitive and Electrical Memory Test

- Notation
- Neighborhood pattern sensitive fault algorithms
- Cache DRAM and ROM tests
- Memory Electrical Parametric Tests
- Summary

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
- PMPSF -- Passive Neighborhood PSF
- SNPSF -- Static Neighborhood Pattern Sensitive Fault

Neighborhood Pattern Sensitive Coupling Faults
- Cell’s ability to change influenced by all other memory cell contents, which may be a 0/1 pattern or a transition pattern.
- Most general k-Coupling Fault
- Base cell -- cell under test
- Deleted neighborhood -- neighborhood without the base cell
- Neighborhood is single position around base cell
- Testing assumes read operations are fault free

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.
  - \( C_{ij} < d_2, d_1, d_0, d_4 \uparrow b > \)
  - \( C_{ij} < 0, 1, 1; 0 > \) and \( C_{ij} < 0, 1, 1 > \)

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.
  - \( \uparrow / 0 ( \downarrow / 1 ) \) – Base cell fault effect indicating that base cannot change
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 SNPSF
- Condition for detection and location: Apply all 0 and 1 combinations to k-cell neighborhood, and verify that each base cell was written.
- $C_{ij} < 0, 1, 0, 1$ and $C_{ij} < 0, 1, 0, 1$

Type 1 Tiling Neighborhoods

- Write changes $k$ different neighborhoods
- Tiling Method: Cover all memory with non-overlapping neighborhoods

NPSF Testing Algorithm Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Fault Location?</th>
<th>Fault Coverage</th>
<th>Fault Coverage</th>
<th>Operation Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDANPSF1G</td>
<td>No</td>
<td>L</td>
<td>D</td>
<td>163.5 $n$</td>
</tr>
<tr>
<td>TLANPNSF1G</td>
<td>Yes</td>
<td>L, L</td>
<td>L</td>
<td>195.5 $n$</td>
</tr>
<tr>
<td>TLANPNSF2T</td>
<td>Yes</td>
<td>L, L</td>
<td>L</td>
<td>512 $n$</td>
</tr>
<tr>
<td>TLANPNSF1T</td>
<td>Yes</td>
<td>L, L</td>
<td>L</td>
<td>194 $n$</td>
</tr>
<tr>
<td>TLSNPSF1G</td>
<td>Yes</td>
<td>L</td>
<td>D</td>
<td>43.5 $n$</td>
</tr>
<tr>
<td>TLSNPSF1T</td>
<td>Yes</td>
<td>L</td>
<td>D</td>
<td>39.2 $n$</td>
</tr>
<tr>
<td>TLSNPSF2T</td>
<td>Yes</td>
<td>L</td>
<td>D</td>
<td>569.78 $n$</td>
</tr>
<tr>
<td>TDSNPSF1G</td>
<td>No</td>
<td>L</td>
<td>D</td>
<td>36.125 $n$</td>
</tr>
</tbody>
</table>

Fault Hierarchy