Memory Testing

- Introduction
- Memory Architecture & Fault Models
- Test Algorithms
- DC / AC / Dynamic Tests
- Built-in Self Testing Schemes
- Built-in Self Repair Schemes

Memory Market Share in 1999

- DRAM: 8×10^{17}
- Flash: 6×10^{16}
- ROM: 2×10^{16}
- SRAM: 9×10^{15}

DRAM Price per Bit

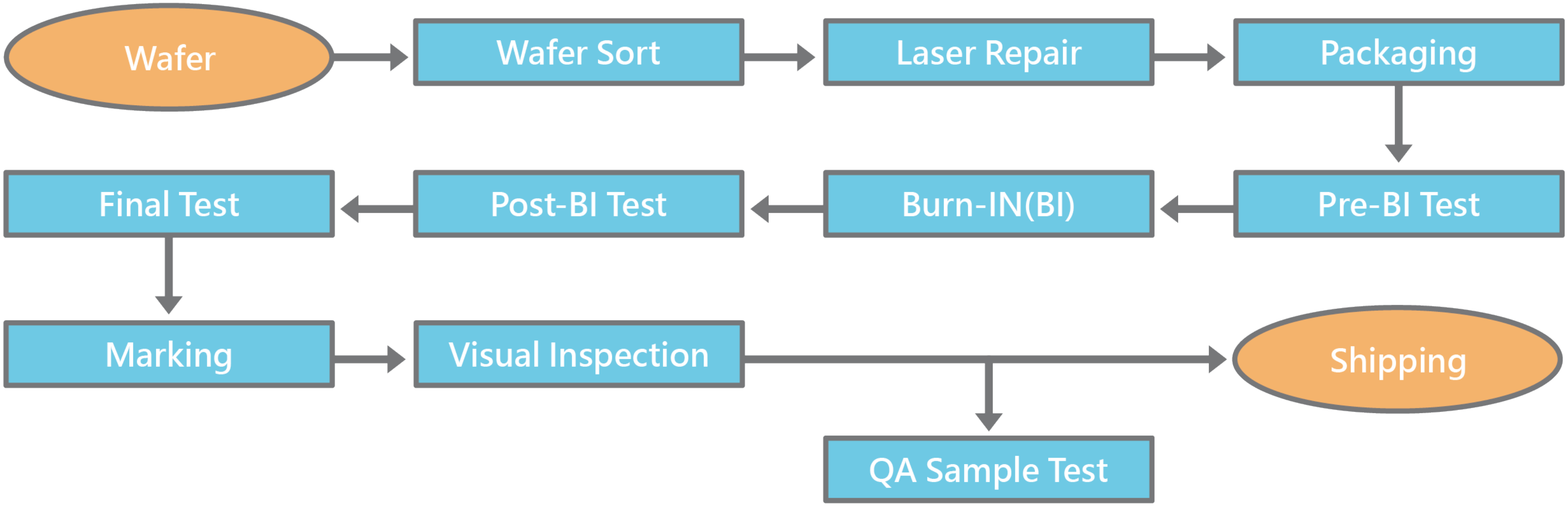
1991: US$ 400 / Mega bits

1995: US$ 3.75 / Mega bits

1999: US$ 0.1~0.3 / Mega bits

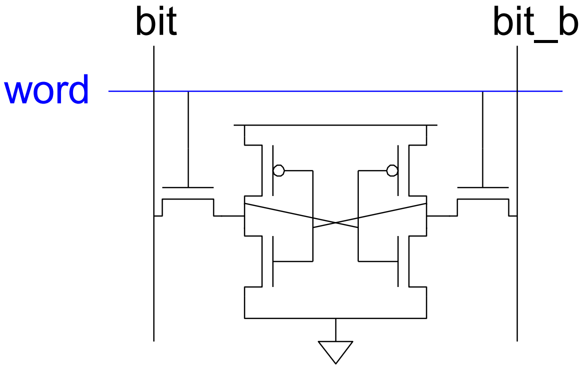
2000: US$ 0.06~0.15 / Mega bits

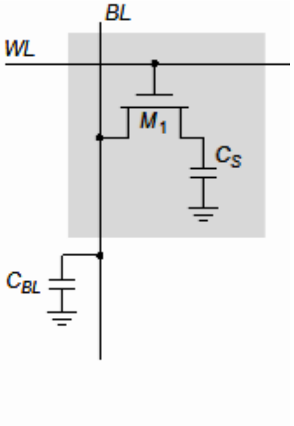
Typical Memory Test Flow

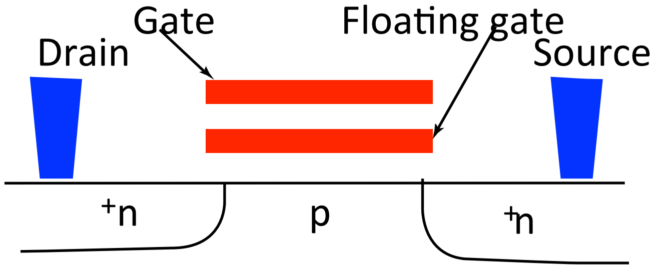


Types of Memory

Type	Area	Speed	Retention	Application	Test Method
SRAM	Largest 6T	Fastest < 1 ns	As long as power	Embedded SRAM cache, registers	BIST
DRAM	Medium 1T + 1C	Medium ～ 10 ns	< sec.	Embedded DRAM	BIST / ATE
				On-board memory	ATE
Flash	Smallest 1T	Slowest ～ 100 µs	years	SSD, USB drive	ATE





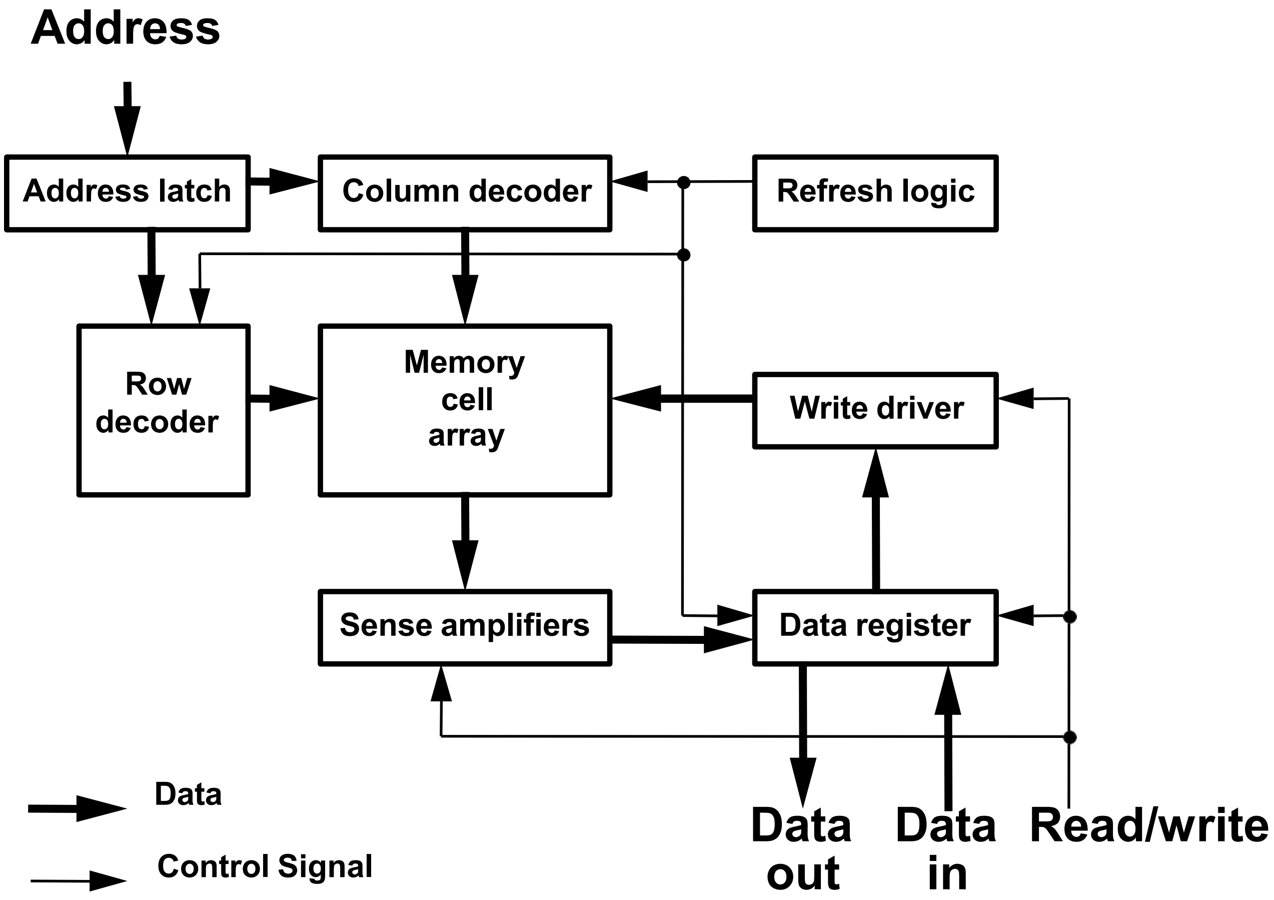


Test Time as a Function of Memory Size

Cycle time: 10 ns

Size n	Testing time (in seconds)			
	$64n$	$n\ log_2n$	$n^{3/2}$	n^2
16k	0.01	0.0023	0.021	2.7
64k	0.04	0.01	0.168	42
256k	0.17	0.047	1.34	11.4 Mins
1M	0.67	0.21	10.7	183 Mins
4M	2.68	0.92	85.9	49.2 Hrs
16M	10.8	4.03	11.4 Mins	36.5 Days
64M	43.2	16.2	91.6 Mins	584 Days

Architecture of a DRAM Chip



Fault Models

1.	SAF	Stuck-At Fault
2.	TF	Transition Fault
3.	CF	Coupling Fault
4.	NPSF	Neighborhood Pattern Sensitive Fault
5.	AF	Address decoding fault

Stuck-At Fault

- The logic value of a cell or a line is always 0 or 1.

Transition Fault

- A cell or a line that fails to undergo a 0 → 1 or a 1→0 transition.

Coupling Fault

- A write operation to one cell changes the content of a second cell.

Coupling Fault

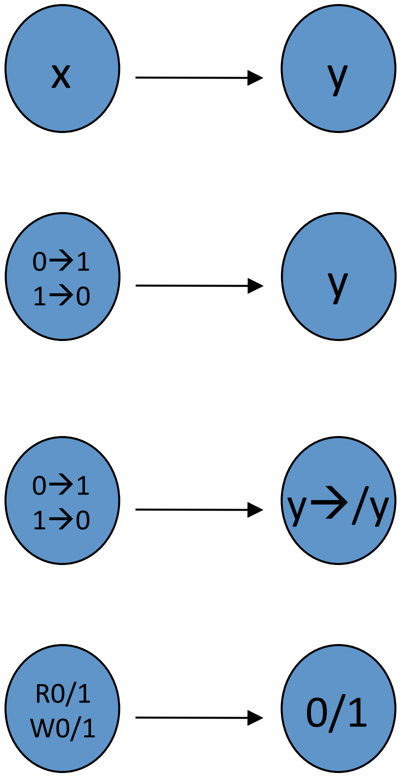
- Coupling Fault (CF):

  * State coupling fault (CFst)

  * Indempotent coupling fault (CFid)

  * Inversion coupling fault (CFin)

  * Dynamic coupling fault (CFdyn)



Neighborhood Pattern Sensitive Fault

- **The content of a cell, or the ability to change its content, is influenced by the contents of some other cells in the memory.**

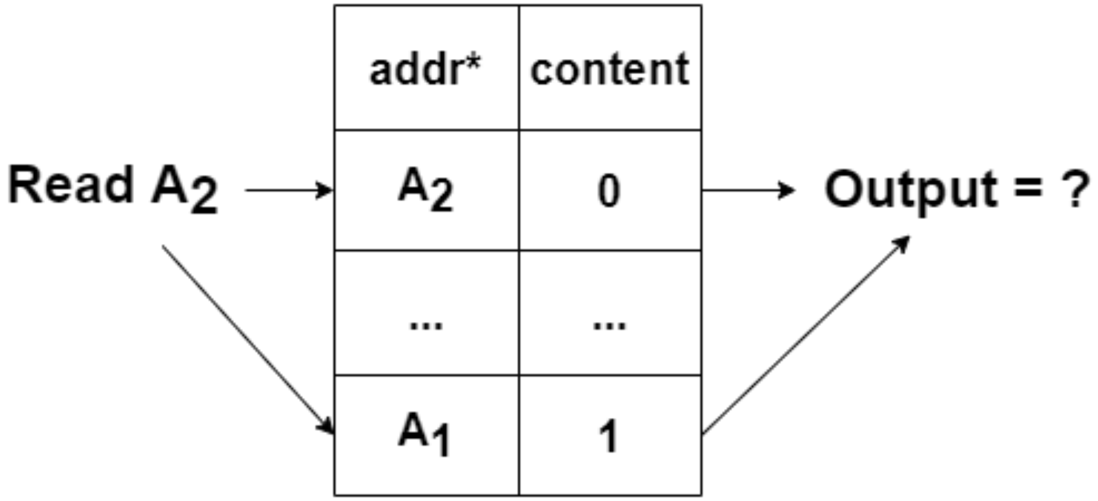
Address Decoder Fault (AF)

- **Any fault that affects address decoder:**

1. With a certain address, no cell will be accessed.
2. A certain cell is never accessed.
3. With a certain address, multiple cells are accessed simultaneously.
4. A certain cell can be accessed by multiple addresses.

Multiple cells Addressed Simultaneously

AF between A_1 and A_2

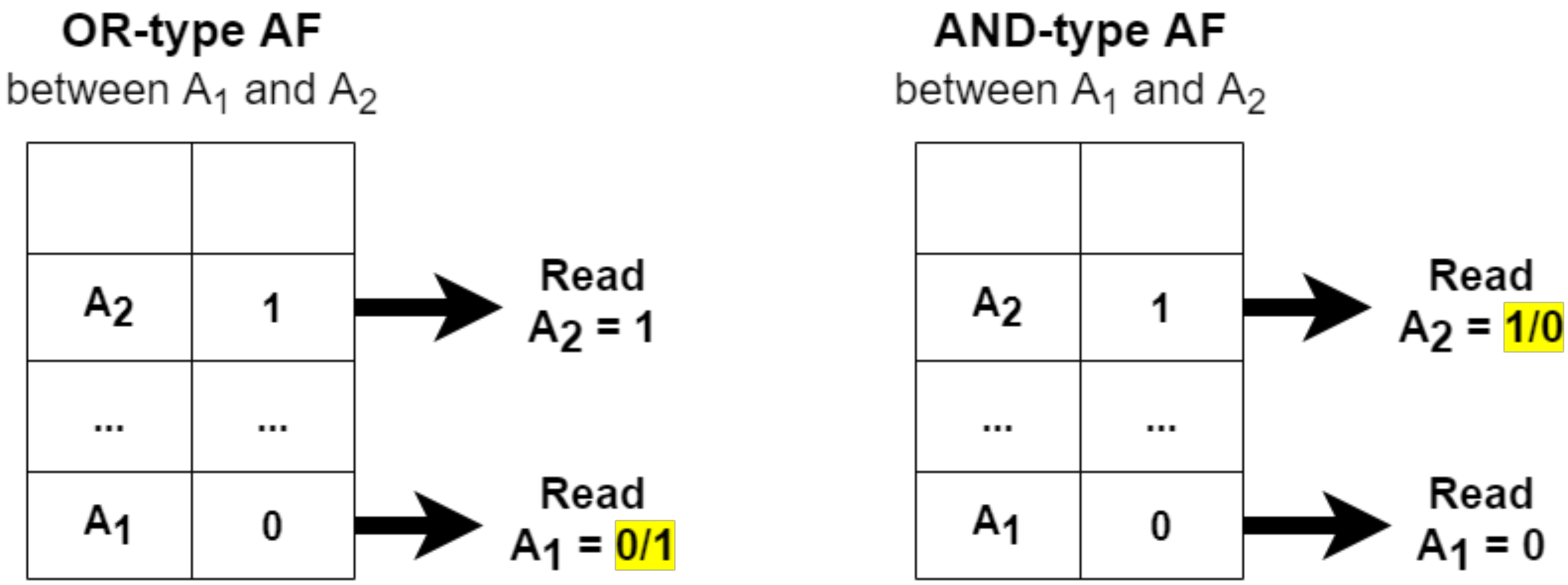


addr*	content
A_2	0
...	...
A_1	1

Read A_2

Output = ?

Multiple cells Addressed Simultaneously



OR-type AF

between A_1 and A_2

A_2	1
...	...
A_1	0

Read $A_2 = 1$

Read $A_1 = 0/1$

AND-type AF

between A_1 and A_2

A_2	1
...	...
A_1	0

Read $A_2 = 1/0$

Read $A_1 = 0$

NPSF

n	n	n
n	b	n
n	n	n

b: base cell
n: neighbor cells

ANPSF:

Active Neighborhood
Pattern Sensitive Fault

n changes

⇨ b changes

Ex:

n: 0 → 1
b: 1 → 0

PNPSF:

Passive Neighborhood
Pattern Sensitive Fault

Contain n patterns

⇨ b cannot change

Ex:

n: 00000000
b: 0 or 1

SNPSF:

Static Neighborhood
Pattern Sensitive Fault

Contain n patterns

⇨ b is forced to a certain value

Ex:

n: 11111111
b: 1

Memory Chip Test Algorithms

- Traditional tests
- Tests for stuck-at, transition and coupling faults
- Tests for neighborhood pattern sensitive faults

Traditional Tests

Algorithm	Test length	Test Time Order
- Zero-One	$4n$	$O(n)$
- Checkerboard	$4n$	$O(n)$
- GALPAT	$2(n+2n^2)$	$O(n^2)$
- Walking 1/0	$2(3n+n^2)$	$O(n^2)$
- Sliding Diagonal	$6n+2n\cdot\sqrt{n}$	$O(n\cdot\sqrt{n})$
- Butterfly	$2[3n+5n(n/2-1)]$	$O(n\cdot\log_2 n)$

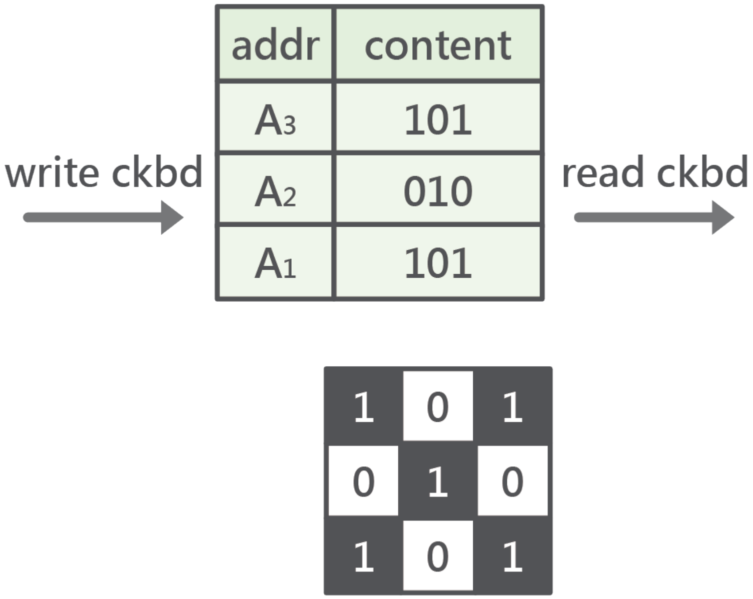
- n is the number of bits of the memory array.

Checkerboard

- Detects all SAF and half TF
- Does not detect all AF and CF
- Time complexity is 4N

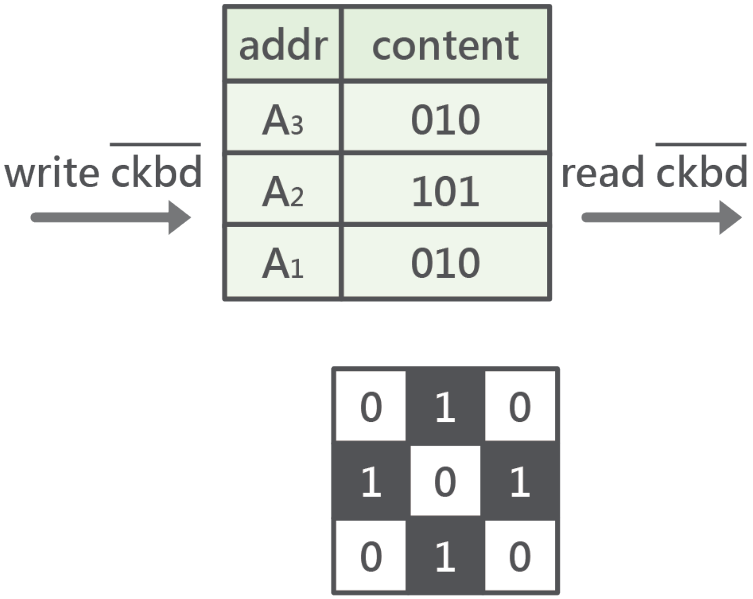
<u>Checkerboard Algorithm</u>

1. Write ckbd pattern to all cells
2. Read ckbd pattern from all cells
3. Write $\overline{\text{ckbd}}$ pattern to all cells
4. Read $\overline{\text{ckbd}}$ pattern from all cells



addr	content
A_3	101
A_2	010
A_1	101

1	0	1
0	1	0
1	0	1



addr	content
A_3	010
A_2	101
A_1	010

0	1	0
1	0	1
0	1	0

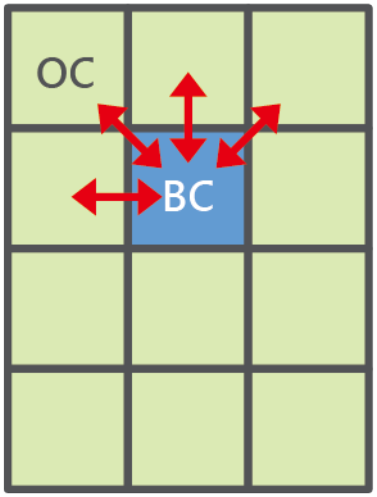
Galloping (GALPAT)

- Detects all SAF and half TF
- Detects CF and AF
- Time complexity is $4N^2$

→ Too Long

GALPAT Algorithm

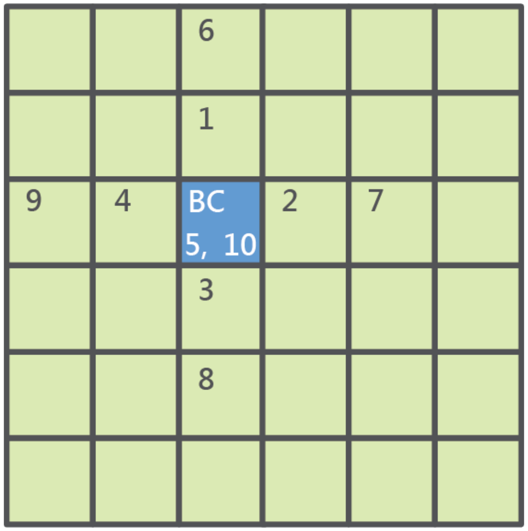
```
1. Write background 0 pattern to all cells
2. For BC=0 to N-1
      Complement BC;
       For OC=0 to N-1, OC!=BC;
            Read BC; Read OC;
        Complement BC;
3. Write background 1 to all cells;
4. Repeat Step 2;
```



BC: Base Cell
OC: Other Cell

Butterfly Algorithm

- Detects SAF and TF
- Does not detect all CF, AF
- Complexity is 10NlogN
  - 5 reads for each dist
  - dist doubles each loop
  - Repeat in line 4



<u>BUTTERFLY Algorithm</u> // given MAXDIST < 0.5 col / row size

```
1. Write background 0;
2. For BaseCell = 0 to N - 1
       Complement BC;  dist = 1;
       While dist <= MAXDIST
           Read cell @ dist north from BC;
           Read cell @ dist east from BC;
           Read cell @ dist south from BC;
           Read cell @ dist west from BC;
           Read BC;    dist = dist * 2;
       Complement BC;
3. Write background 1;
4. Repeat Step 2;
```

March Algorithms

Algorithm March X

Step1: **write** 0 with up addressing order;

Step2: **read** 0 and **write** 1 with up addressing order;

Step3: **read** 1 and **write** 0 with down addressing order;

Step4: **read** 0 with down addressing order.

Notation of March Algorithms

⇑ : address 0 to address n-1

⇓ : address n-1 to address 0

⇕ : either way

w0 : write 0

w1 : write 1

r0 : read a cell whose value should be 0

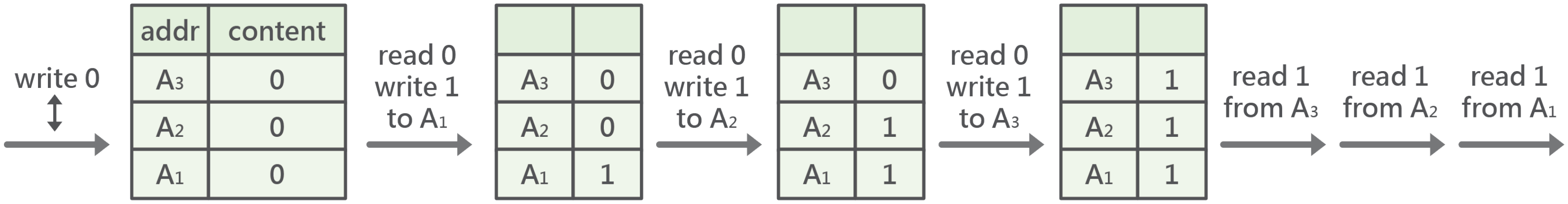
r1 : read a cell whose value should be 1

MATS

- **Modified Algorithm Test Sequence (MATS)**
- **3 march elements:**
  { ⇳(w0); ⇳(r0, w1); ⇳(r1) }
- **Detects all SAF, half TF**
- **Complexity 4N**

MATS+

1. Write zero to all cells
   In ascending or descending address order
2. Read zero and then write one
   In ascending or descending address order
3. Read one in ascending or descending address order



addr	content
A_3	0
A_2	0
A_1	0

A_3	0
A_2	0
A_1	1

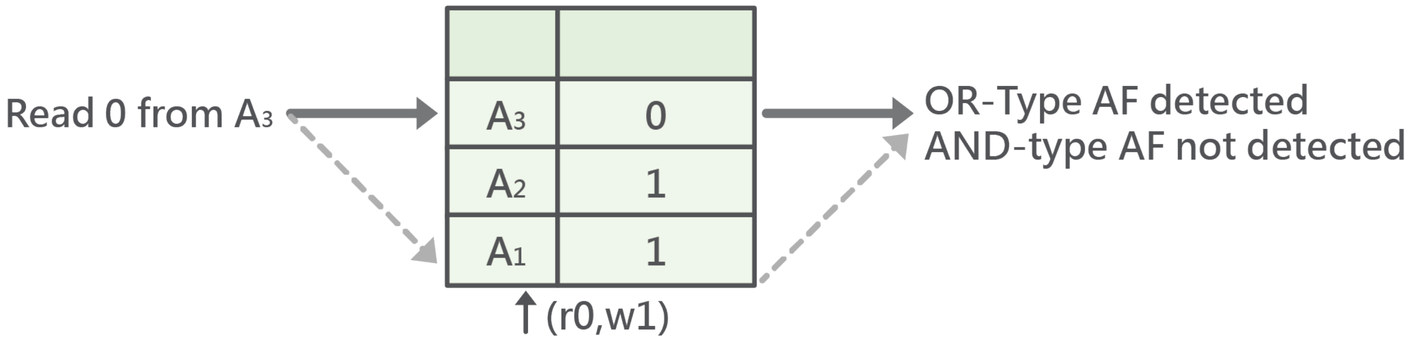
A_3	0
A_2	1
A_1	1

A_3	1
A_2	1
A_1	1

MATS

- **Can MATS algorithm detect AF?**

  -- { ⇳(w0); ⇳(r0, w1); ⇳(r1) }
- **Example OR-type AF between A_3 and A_1**



A_3	0
A_2	1
A_1	1

- **How to fix it? *Reverse MATS*. Detects AND-type but not OR-type**

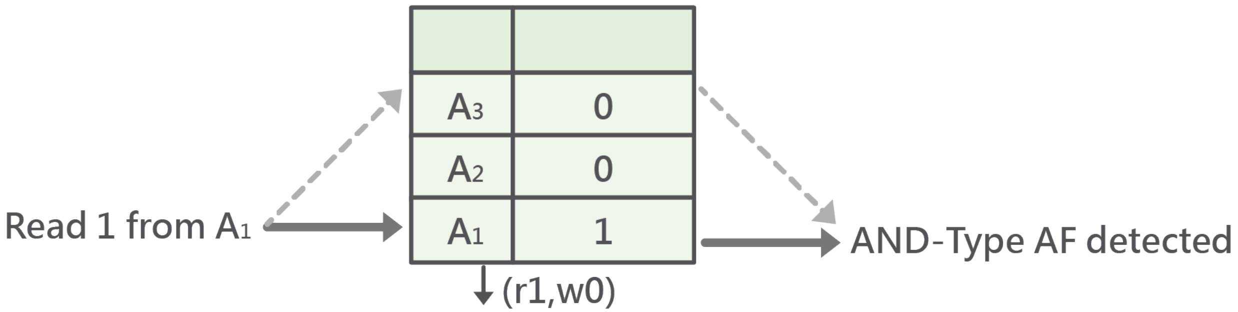
  → { ⇳(w1); ⇳(r1, w0); ⇳(r0) }

	SAF	AF	TF	CF	complexity
MATS	D	1/2	1/2	*	4N

D=all detected
1/2=half detected
* =not detected

MATS+

- **OR-type AF detection MATS**
  -- { ⇳(w0); ⇳(r0, w1); ⇳(r1) }
- **AND-type AF detection MATS**
  -- { ⇳(w1); ⇳(r1, w0); ⇳(r0) }



A3	0
A2	0
A1	1

- **MATS+ combines OR-type and AND-type MATS**
  { ⇳(w0); ⇑(r0, w1); ⇓(r1,w0) }

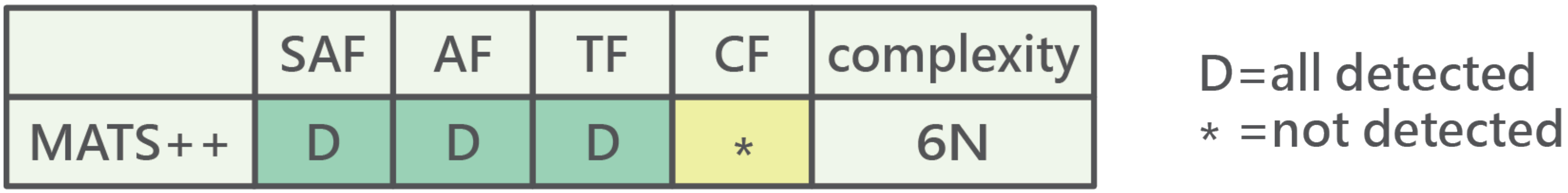
	SAF	AF	TF	CF	complexity
MATS+	D	D	1/2	*	5N

D=all detected
1/2=half detected
* =not detected

MATS++

- **Original MATS+:**
  { ⇕(w0); ⇑(r0, w1); ⇓(r1,w0) }

- **MATS++ algorithm:**
  { ⇕(w0); ⇑(r0, w1); ⇓(r1,w0,r0) }

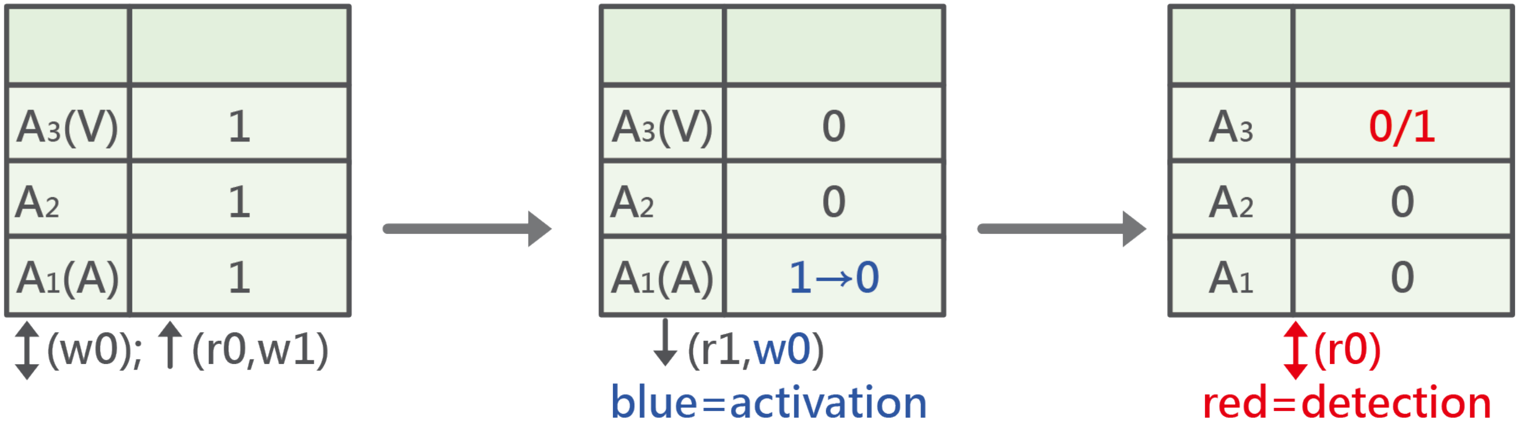


	SAF	AF	TF	CF	complexity
MATS++	D	D	D	*	6N

D=all detected
* =not detected

March X

- MATS++ algorithm { ⇕(w0); ⇑(r0, w1); ⇓(r1, w0, r0) }
- March X algorithm { ⇕(w0); ⇑(r0, w1); ⇓(r1, w0); ⇕(r0) }
- Detects AF, SAF, TF, CF_{in}
- Example:
    - CF_{in} <↓; ∀ / ↕> between A_1(A) and A_3(V)



A_3(V)	1
A_2	1
A_1(A)	1

↕(w0); ↑(r0,w1)

A_3(V)	0
A_2	0
A_1(A)	1→0

↓(r1,w0)

blue=activation

A_3	0/1
A_2	0
A_1	0

↕(r0)

red=detection

March X (cont’d)

- March X algorithm { ⇕(w0); ⇑(r0, w1); ⇓(r1, w0); ⇕(r0) }
- All four cases CF_{in} are detected, still 6N but better than MATS++

V	0/1
A_2	1
A	0→1

↑ (r0,w1)

A	0→1
A_2	0
V	1/0

↑(r0,w1);↓ (r1,w0)

A	1→0
A_2	0
V	1/0

↓ (r1,w0)

V	0/1
A_2	0
A	1→0

↓(r1,w0);↕(r0)

blue=activation
red=detection

	SAF	AF	TF	CF	complexity
March x	D	D	D	CF_{in}	6N

Some March Algorithms

MATS : ⇕ **(w0);** ⇕ **(r0,w1);** ⇕ **(r1)**

MATS+: ⇕ **(w0);** ⇑ **(r0,w1);** ⇓ **(r1,w0)**

Marching 1/0 : ⇕ **(w0);** ⇑ **(r0,w1,r1);** ⇓ **(r1,w0,r0);**

⇕ **(w1);** ⇑ **(r1,w0,r0);** ⇓ **(r0, w1, r1);**

MATS++ : ⇕ **(w0);** ⇑ **(r0,w1);** ⇓ **(r1,w0,r0);**

MARCH X : ⇕ **(w0);** ⇑ **(r0,w1);** ⇓ **(r1,w0);** ⇕ **(r0)**

MARCH C : ⇕ **(w0);** ⇑ **(r0,w1);** ⇑ **(r1,w0);** ⇕ **(r0);**

⇓ **(r0,w1);** ⇓ **(r1,w0);** ⇕ **(r0);**

Some March Algorithms (Cont.)

MARCH A : $\Updownarrow$ (w0); $\Uparrow$ (r0,w1,w0,w1); $\Uparrow$ (r1,w0,w1);
$\Downarrow$ (r1,w0,w1,w0); $\Downarrow$ (r0,w1,w0);

MARCH Y : $\Updownarrow$ (w0); $\Uparrow$ (r0,w1,r1); $\Downarrow$ (r1,w0,r0); $\Updownarrow$ (r0)

MARCH B : $\Updownarrow$ (w0); $\Uparrow$ (r0,w1,r1,w0,r0,w1); $\Uparrow$ (r1,w0,w1);
$\Downarrow$ (r1,w0,w1,w0); $\Downarrow$ (r0,w1,w0)

Tests for Stuck-At, Transition and Coupling Faults

March alg.	Test len.	Fault coverage
MATS	4n	Some AFs, SAFs
MATS+	5n	AFs, SAFs
Marching 1/0	14n	AFs, SAFs, TFs
MATS++	6n	AFs, SAFs, TFs
March X	6n	AFs, SAFs, TFs, Some CFs
March C-	10n	AFs, SAFs, TFs, Some CFs
March A	15n	AFs, SAFs, TFs, Some CFs
March Y	8n	AFs, SAFs, TFs, Some CFs
March B	17n	AFs, SAFs, TFs, Some CFs

March C

- March C = two March X combined in opposite address order
  → { ⇕(w0); ⇑(r0, w1); ⇑(r1, w0); ⇕(r0); ⇓(r0, w1); ⇓(r1, w0); ⇕(r0) }
- Detects AF, SAF, TF, & all CF
- March C detects all eight cases of CF$_{id}$
  → A_1 is aggressor and A_3 is victim
  → The other four cases (A_1 is V, A_3 is A) are symmetric

A_3	0/1
A_2	0
A_1	0→1

↑(r0,w1)

V	1/0
A_2	1
A	0→1

↑(r0,w1);↑(r1,w0)

V	1/0
A_2	1
A	1→0

↑(r1,w0)

V	0/1
A_2	0
A	1→0

↑(r1,w0);↕(r0)

DC Parametric Testing

- Contains:

1. Open / Short test.
2. Power consumption test.
3. Leakage test.
4. Threshold test.
5. Output drive current test.
6. Output short current test.

AC Parametric Testing

- Output signal: - the rise & fall times.
- Relationship between input signals:
  - the setup & hold times.
- Relationship between input and output signals:
  - the delay & access times.
- Successive relationship between input and output signals:
  - the speed test.

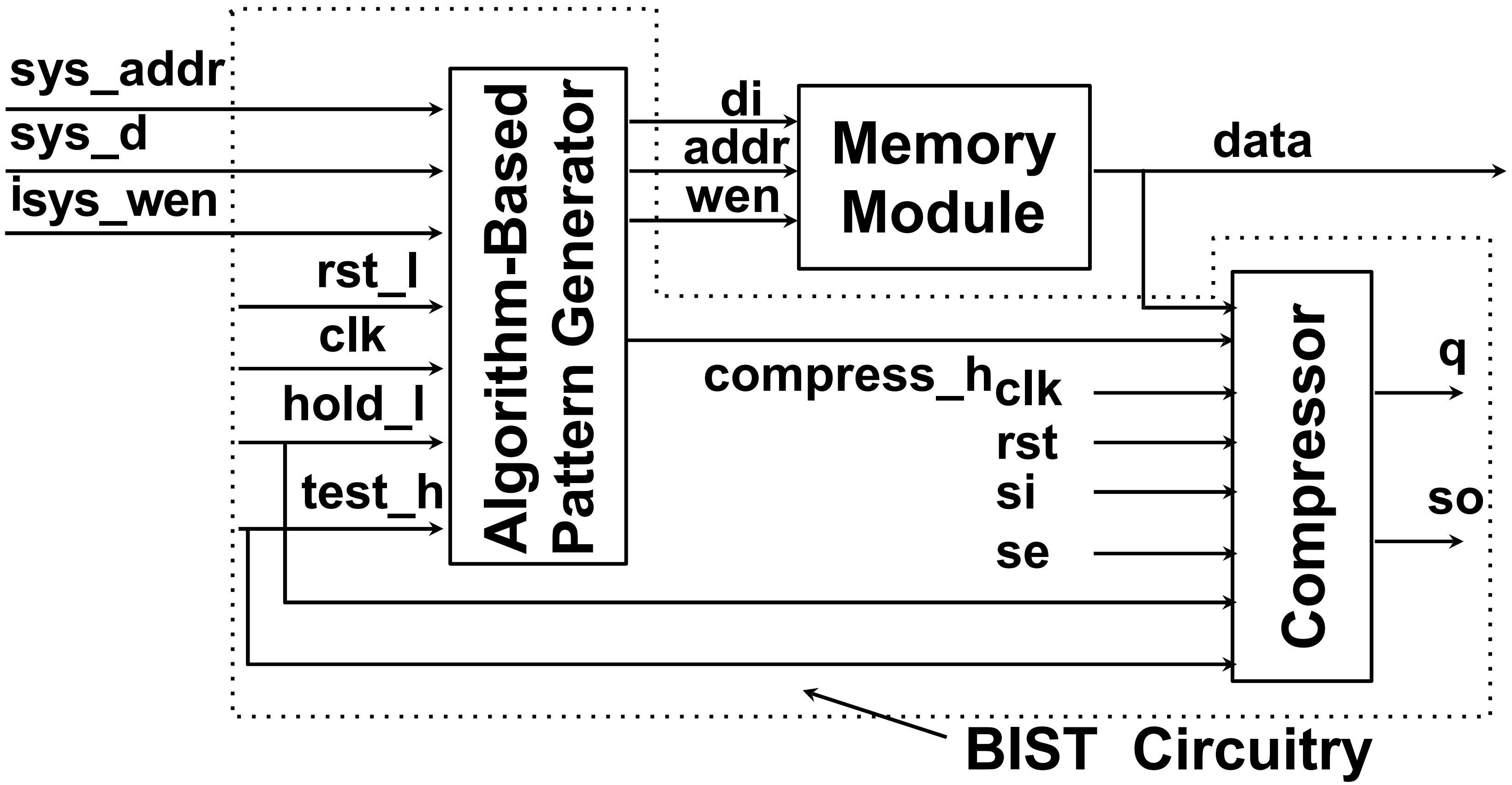
Dynamic Faults

1. Recovery faults:
   - Sense amplifier recovery
   - Write recovery.
2. Retention faults:
   - Sleeping sickness
   - Refresh line stuck-at
   - Static data loss.
3. Bit-line precharge voltage imbalance faults.

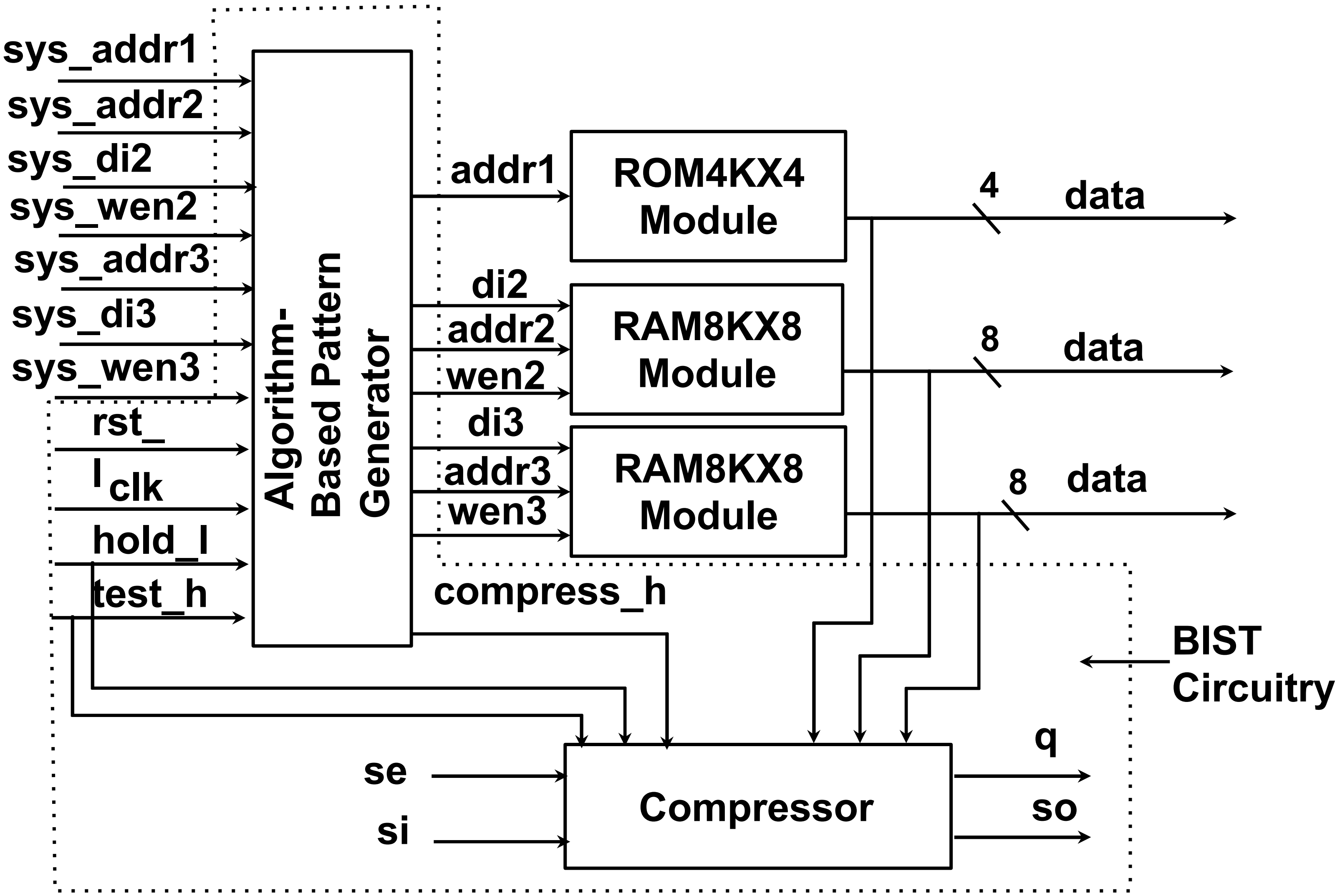
BIST: Pros & Cons

- Advantages:
  1. Minimal use of testers.
  2. Can be used for embedded RAMs.
- Disadvantages:
  1. Silicon area overhead.
  2. Speed; slow access time.
  3. Extra pins or multiplexing pins.
  4. Testability of the test hardware itself.
  5. A high fault coverage is a challenge.

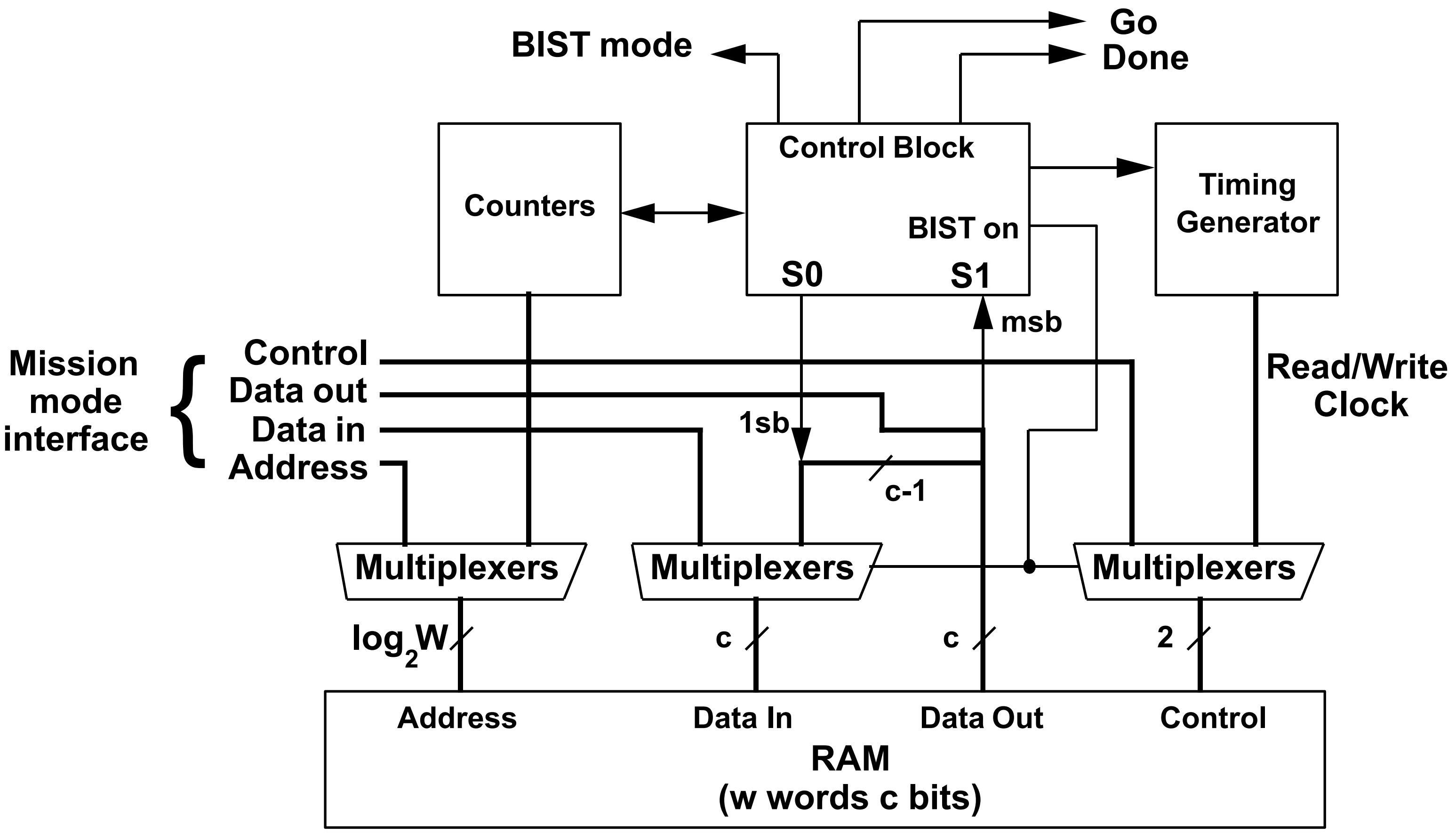
Typical Memory BIST Architecture Using Mentor's Architecture



Multiple Memory BIST Architecture



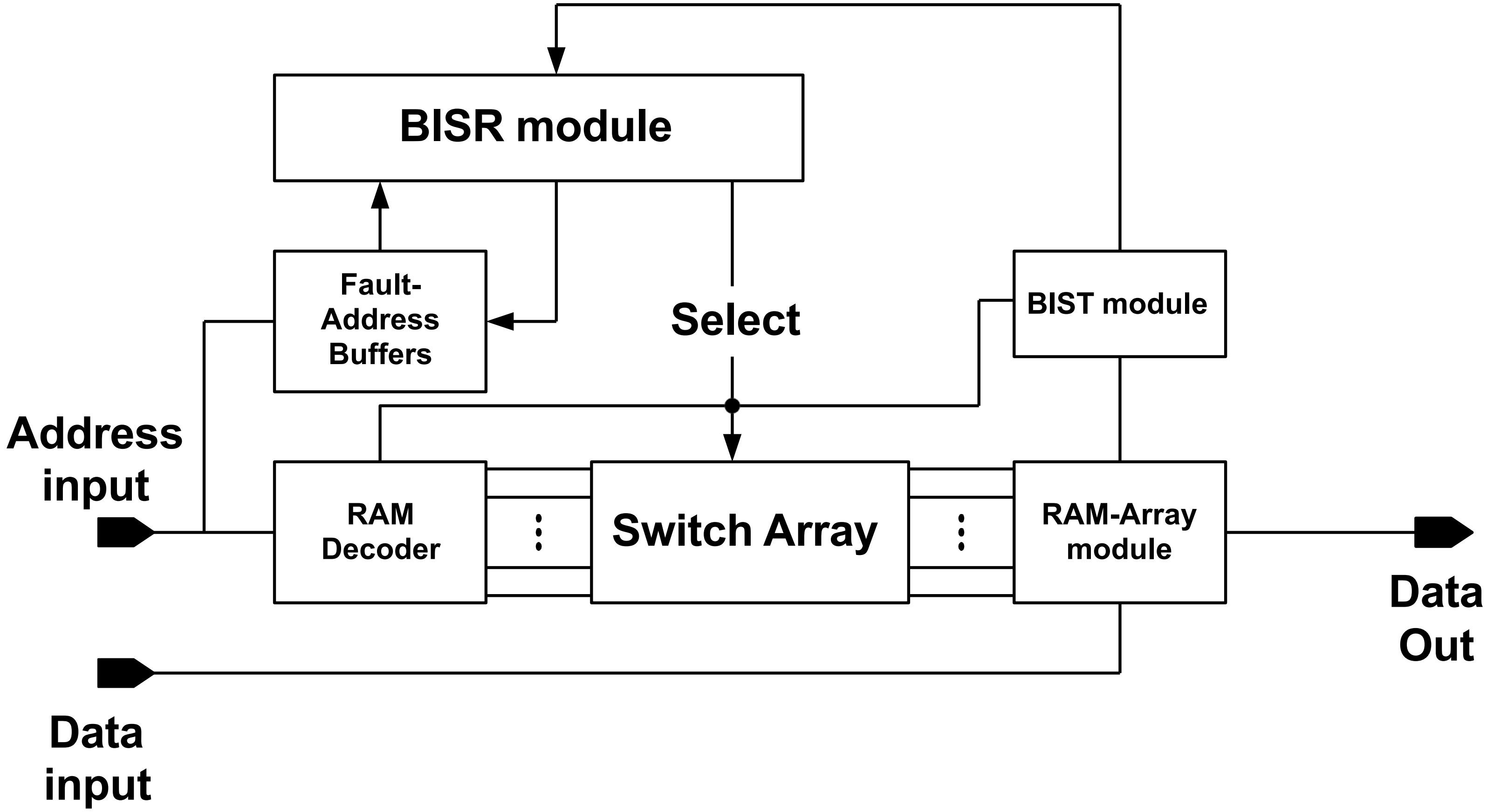
Serial Testing of Embedded RAM



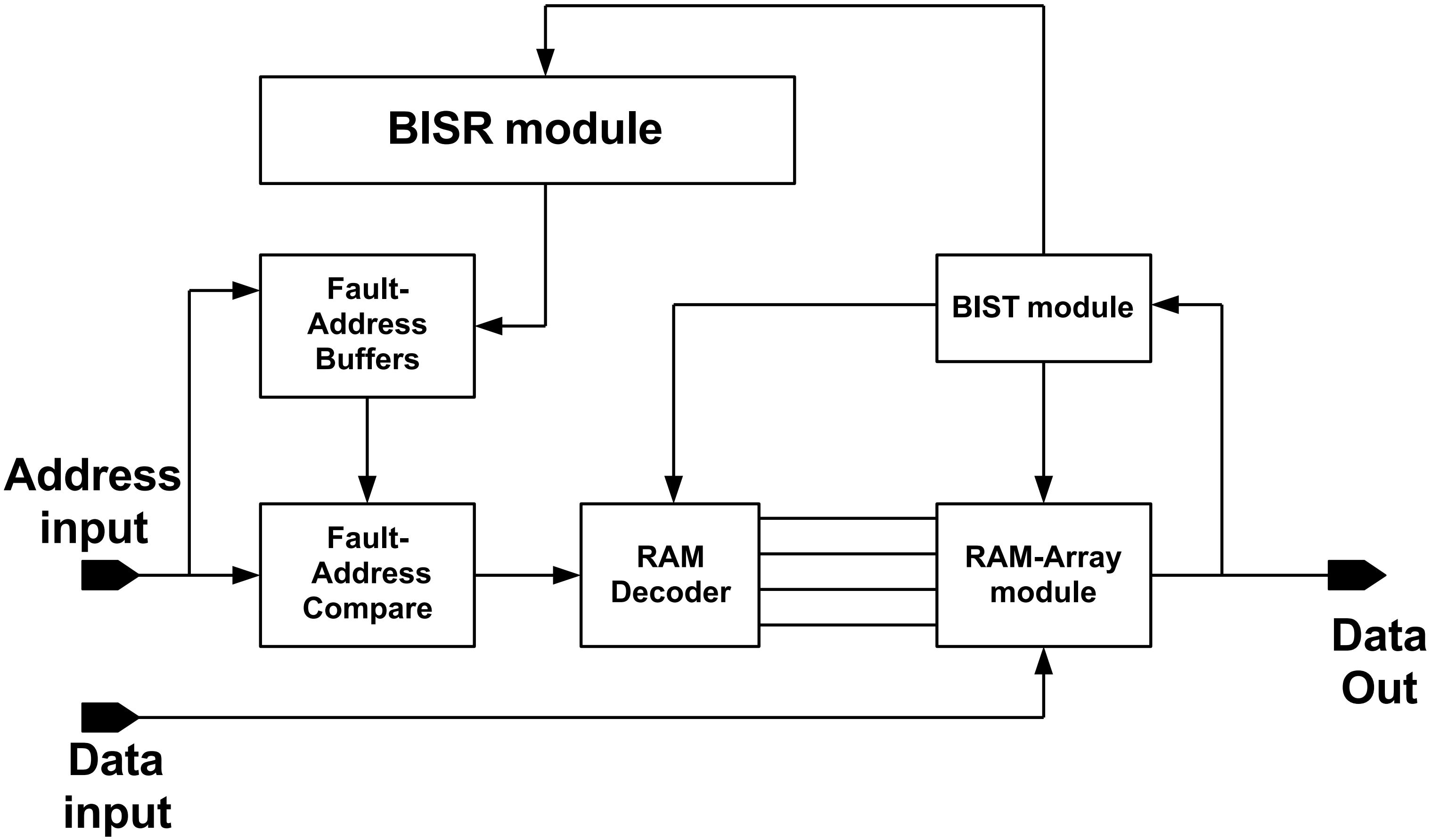
Built-in Self-Repair

- BIST can only identify faulty chip.
- Laser cut may be infeasible in some cases, e.g., field testing.
- Two types:
  - Use fault-array comparator
    - Repair by cell
    - Repair by column (or row)
  - Use switch array

BISR Using Switch Array



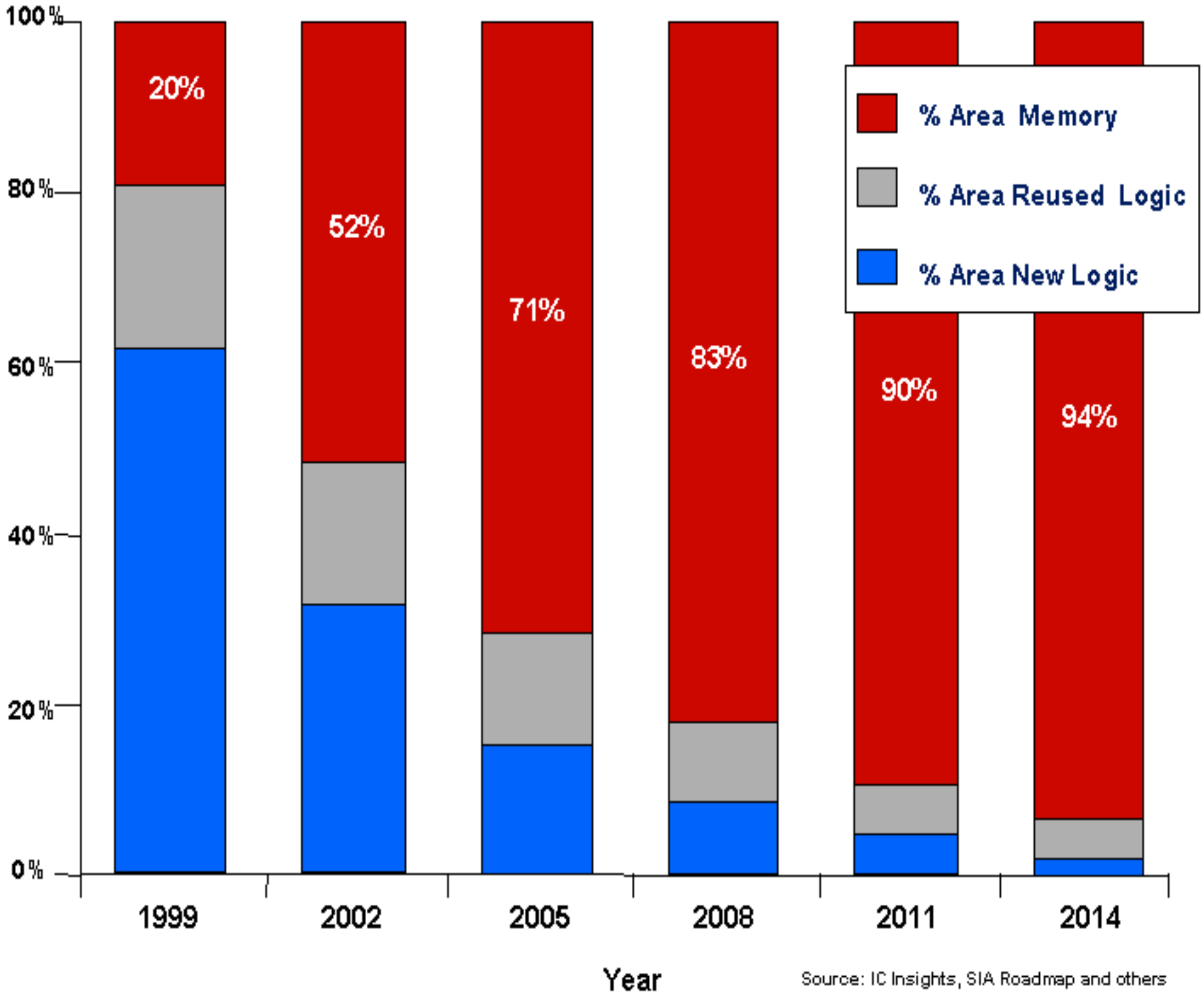
BISR via Fault-Address Comparison



Testing and Diagnostic Methods for Multiple Embedded Memory

- Introduction
- Parallel BIST Method
- Parallel BISD Method
- Parallel Transparent BIST Method
- Parallel Transparent BISD Method

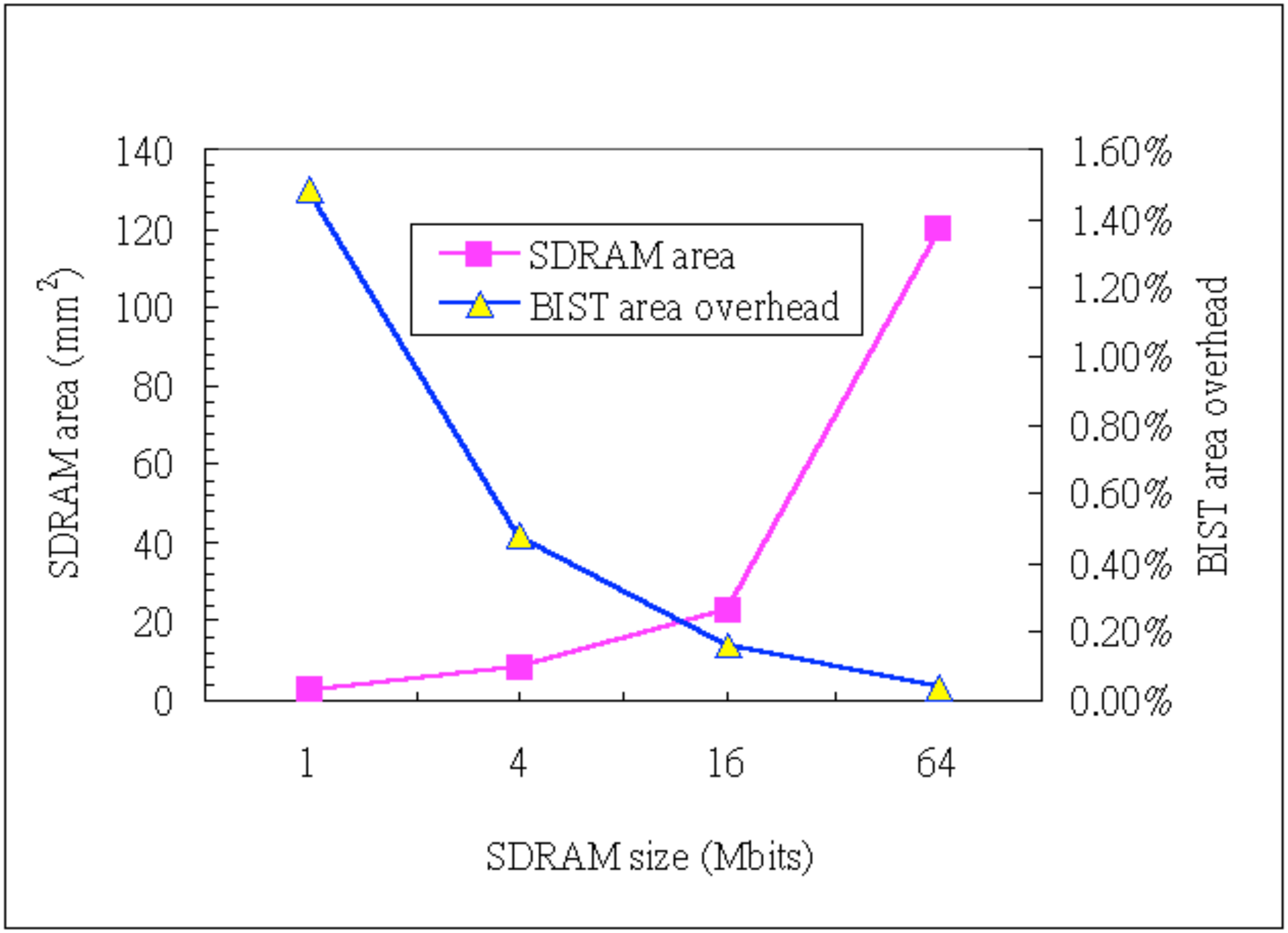
Introduction



Source: IC Insights, SIA Roadmap and others

Introduction

Area Overhead



Introduction

- Difficulties of testing memory buffers
  - Many buffers condensed in a single chip (ex. SoC)
  - Buffer size vs. BIST controller size
  - Buffers are spatially distributed on entire chip
  - Most buffers are deeply embedded inside chip

Introduction

- Difficulties of diagnosing memory buffers

  - A very limited access interface (serial interface)
  - Many faults existing together - SAF, TF, CFst ,CFdyn, CFid and CFin
  - Lack of the diagnostic information due to response compression
  - Testing and locating faults in a parallel manner

Introduction

- Low area overhead

  - adopts a scan chain and a single BIST/BISD controller

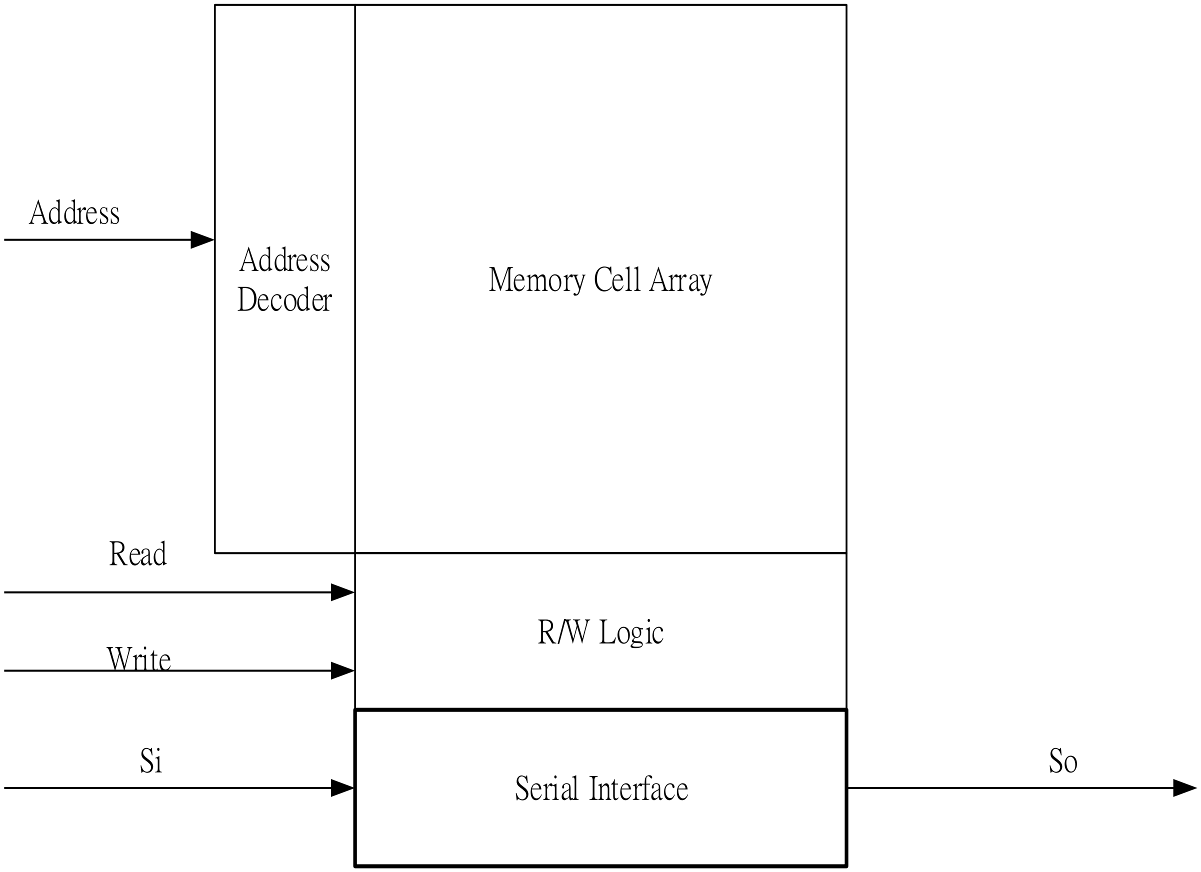
- Small test and diagnosis time

  - test and diagnosis in a parallel way

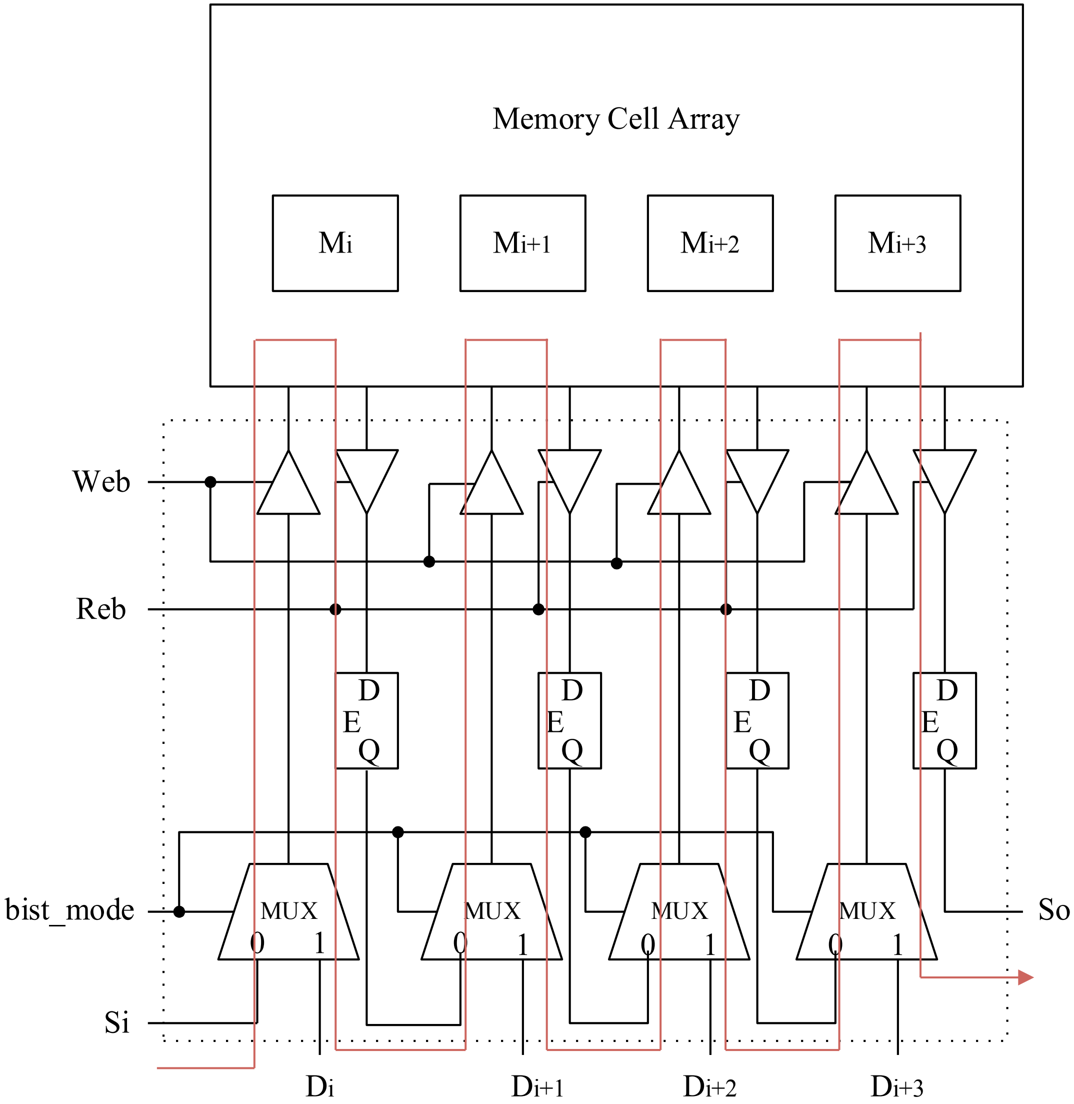
- High fault coverage

  - the "*redundant*" operations do not mask fault detection

Serial Interface



Serial Interface



SMarch Alogorithm

$$M1: \Uparrow (r\,x\,w0)^c(r0w0)^c$$

$$M2: \Uparrow (r0w1)^c(r1w1)^c$$

$$M3: \Uparrow (r1w0)^c(r0w0)^c$$

$$M4: \Downarrow (r0w1)^c(r1w1)^c$$

$$M5: \Downarrow (r1w0)^c(r0w0)^c$$

$$M6: \Downarrow (r0w1)^c(r1w1)^c$$

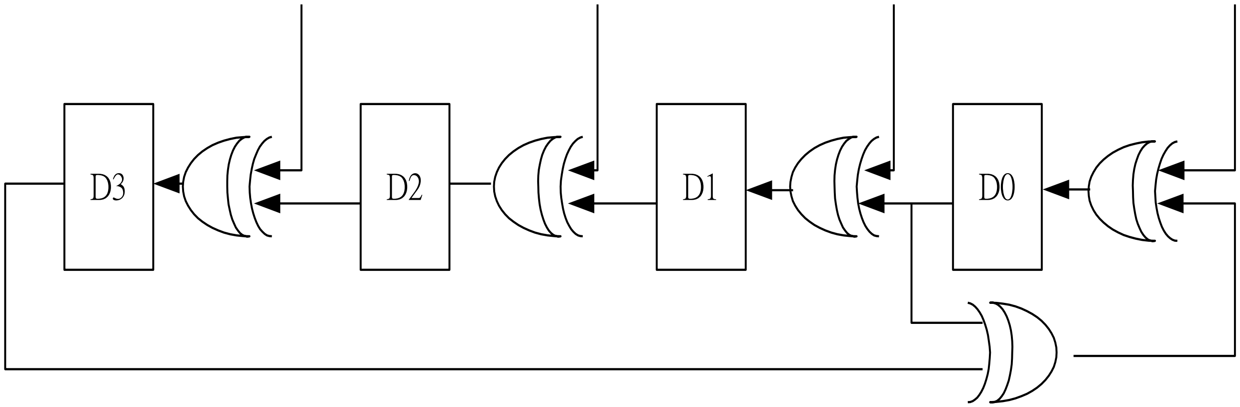
SMarch Alogorithm

Time	Operation	Si	Word contents				So
0		0	1	1	1	1	0
1	R1	0	1	1	1	1	1
2	W0	0	0	1	1	1	1
3	R1	0	0	1	1	1	1
4	W0	0	0	0	1	1	1
5	R1	0	0	0	1	1	1
6	W0	0	0	0	0	1	1
7	R1	0	0	0	0	1	1
8	W0	0	0	0	0	0	1

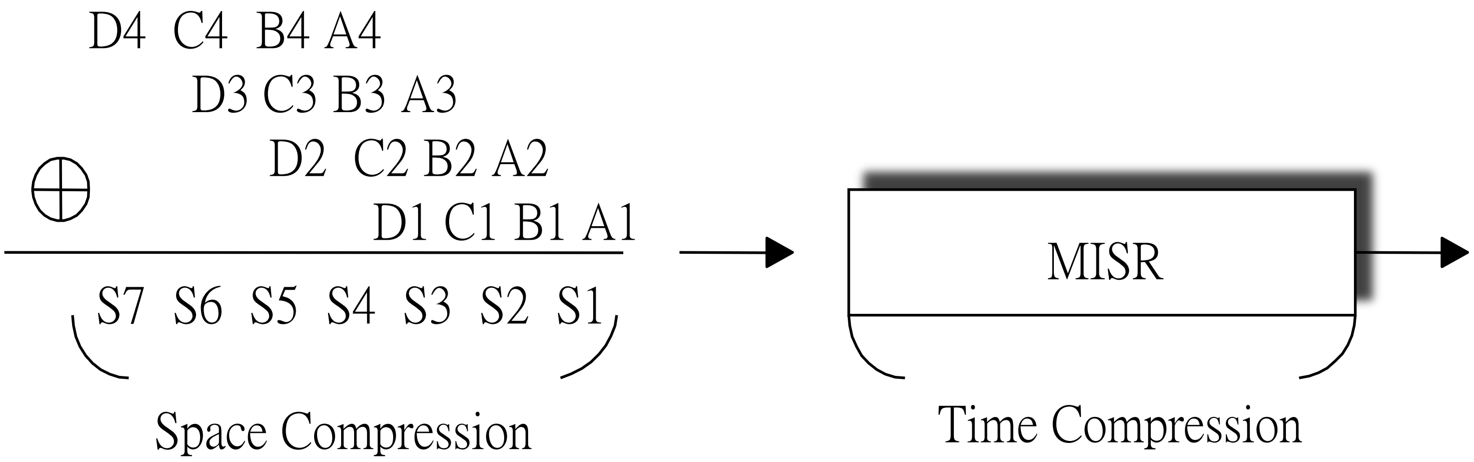
$(R1W0)^4$

MISR

- BIST Response Compress Scheme
  - MISR (Multiple Input Signature Register)



MISR



D4 C4 B4 A4

D3 C3 B3 A3

D2 C2 B2 A2

⊕ D1 C1 B1 A1

S7 S6 S5 S4 S3 S2 S1

Space Compression

MISR

Time Compression

Fault Model

- Stuck-at fault (SAF): The logic value of memory is always stuck-at 1 (SA1) or 0 (SA0)

- Transition fault (TF): A cell fails to undergo a 0 → 1 (SA0) or 1 →0 (SA1)

Fault Model

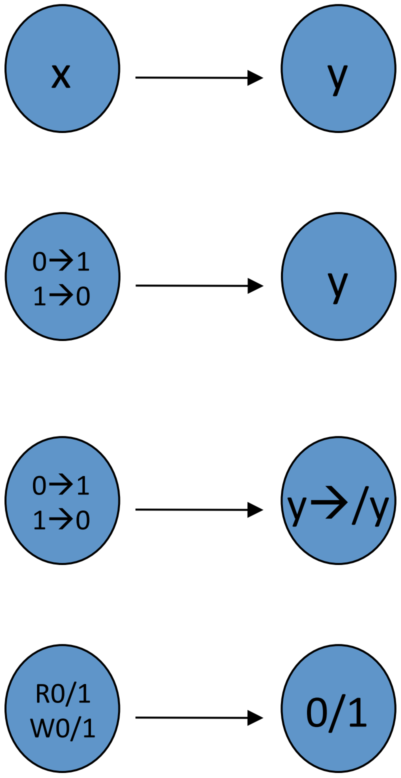
- Coupling Fault (CF):

* State coupling fault (CFst)

* Indempotent coupling fault (CFid)

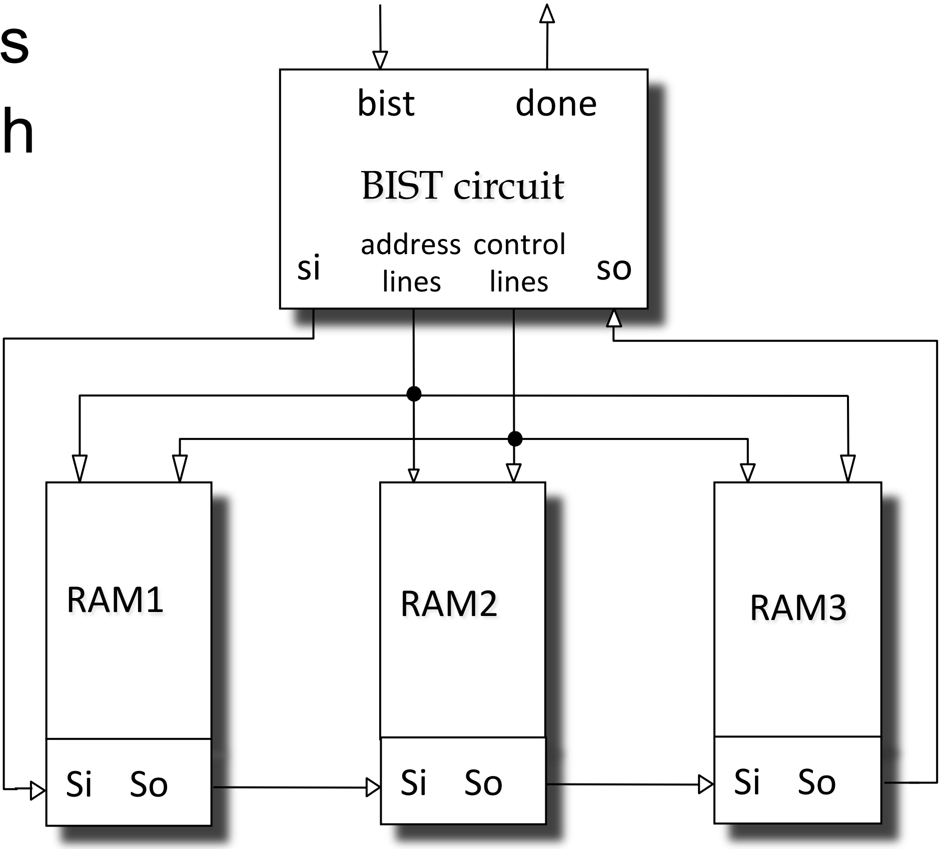
* Inversion coupling fault (CFin)

* Dynamic coupling fault (CFdyn)



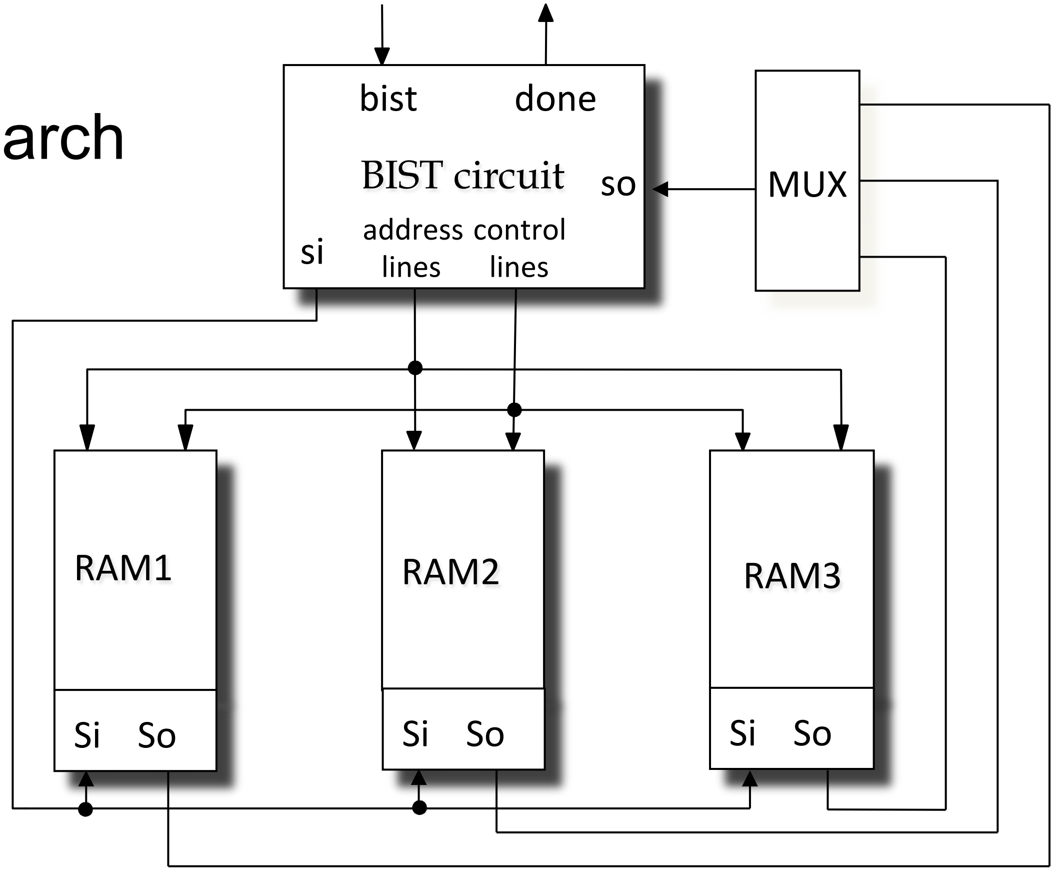
Daisy Chain Method

- Daisy chain method
  - All RAMs must have the same number of words
  - The test time is determined by the total word width of all memory arrays
  - Adopts SMarch



Time Multiplex Method

- Time Multiplex method
  - Test one RAM at a time
  - Additional reconfigurable counter required to deal with different array sizes
  - Adopts SMarch



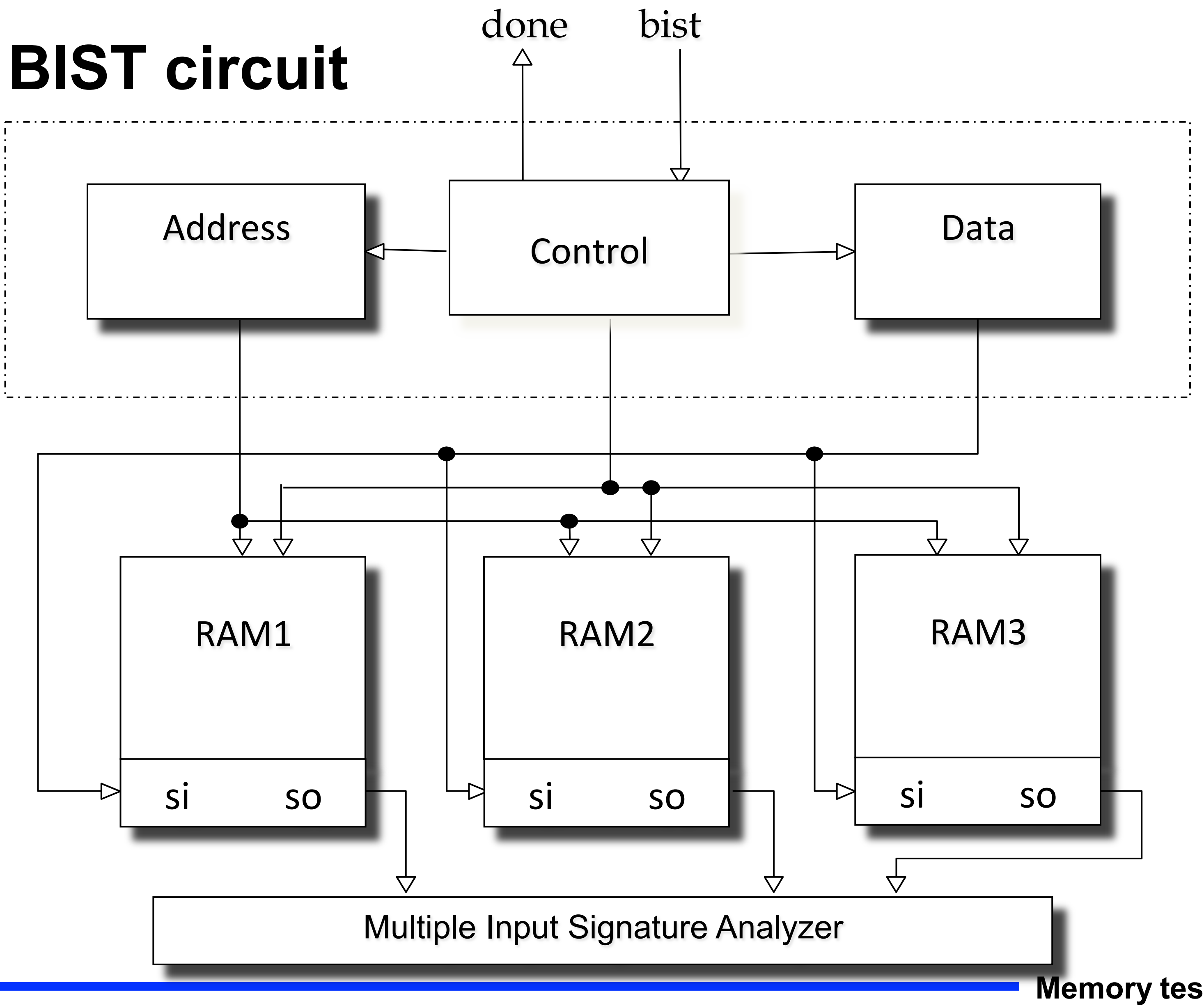
Parallel BIST Method

An Off-Line Solution

Parallel BIST Method

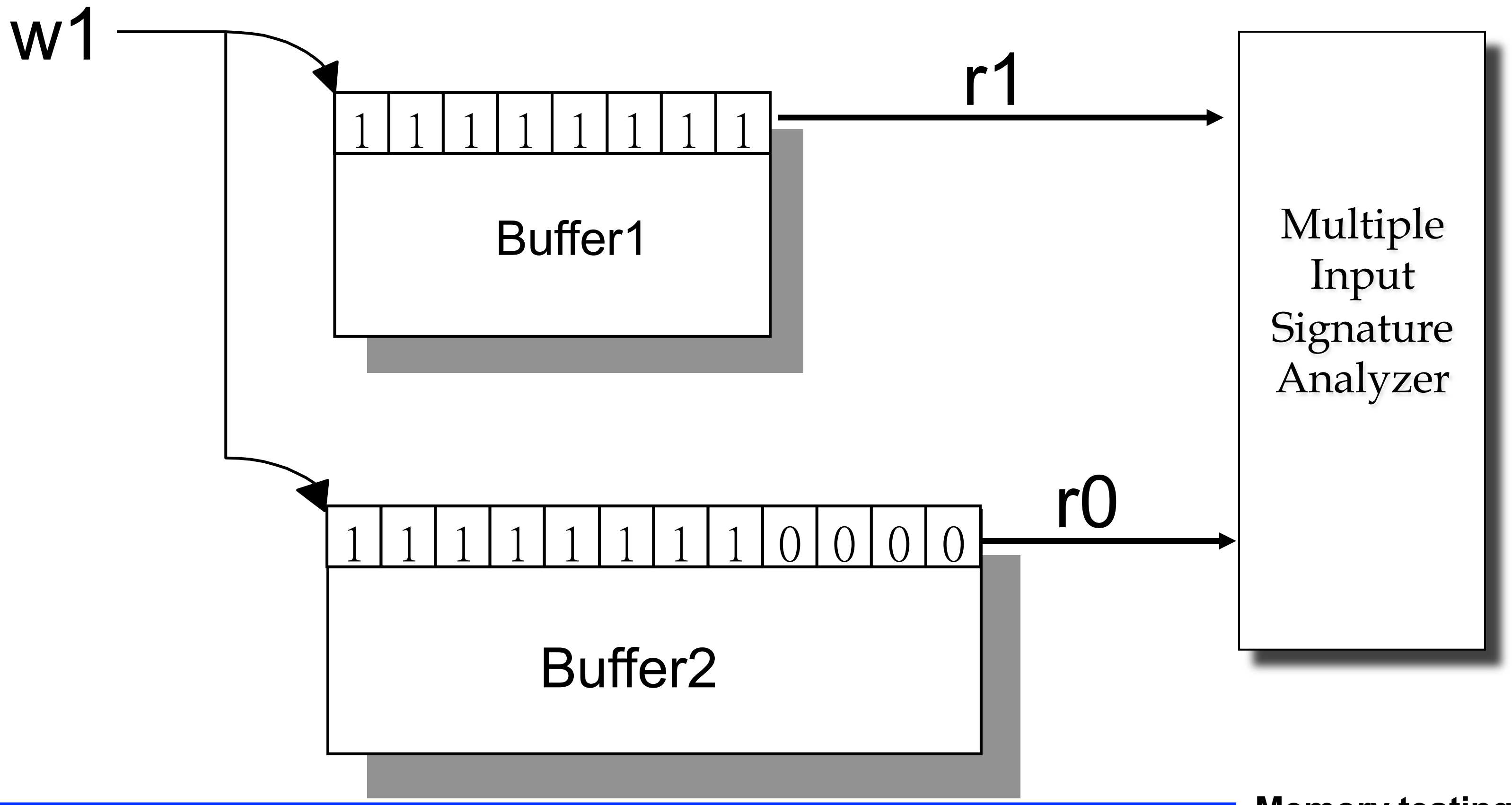
- Basic idea
  - Adopt the maximum word width of all buffers
  - Adopt the maximum word length of all buffers
- Large-size buffer dominates the test process
- Small-size buffer might receive extra patterns

Parallel BIST Method



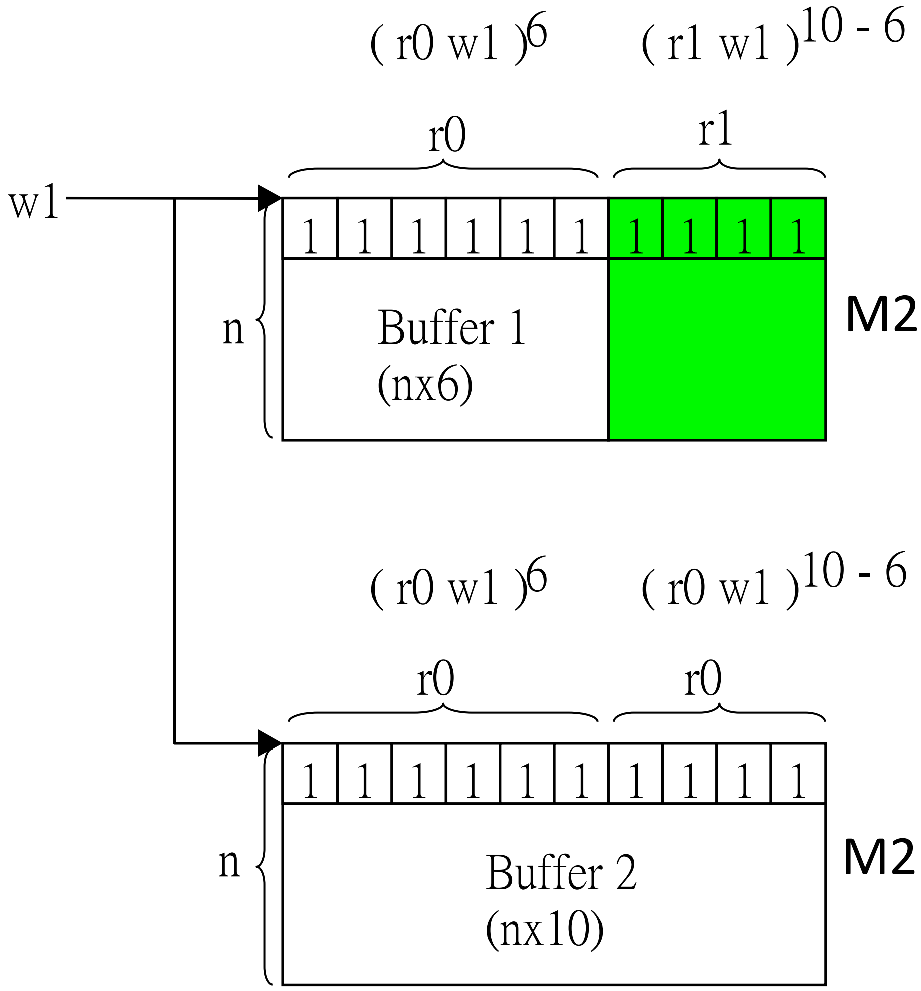
Parallel BIST Method

- Different length problem
- Redundant operations
- SMarch can not deal with *r1* and *r0* simultaneously



Parallel BIST Method

Horizontal Redundant March Operations:

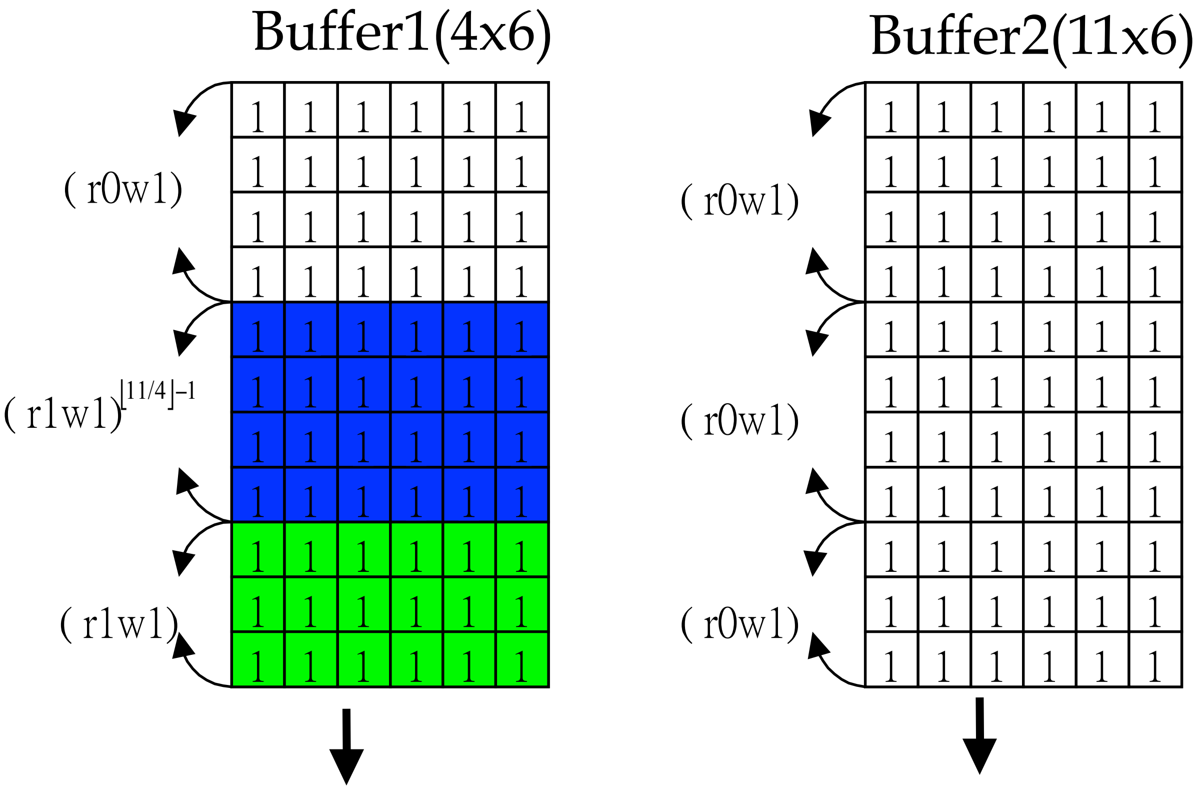


M2 : $\Uparrow_0^{n-1}(\,r0w1\,)^6(\,r1w1\,)^4(\,r1w1\,)^6(\,r1w1\,)^4$

M2: $\Uparrow_0^{n-1}(\,r0w1\,)^{10}(\,r1w1\,)^0(\,r1w1\,)^{10}(\,r1w1\,)^0$

Parallel BIST Method

Vertical Redundant March Operations:



M2: $\Uparrow_0^3 \ (r0w1)^6 (r1w1)^0 (r1w1)^6 (r1w1)^0$

M2: $\Uparrow_0^{10} (r0w1)^6 (r1w1)^0 (r1w1)^6 (r1w1)^0$

$(\Uparrow_0^3 (r1w1)^{12})^{\lfloor 11/4 \rfloor -1}$

$\Uparrow_0^{(11 \bmod 4)-1} (r1w1)^{12}$

Horizontal

Vertical

Parallel BIST Method

RSMarch (Redundant SMarch) Algorithm:

M1 : $\Uparrow_{0}^{n'-1}(\text{rxw0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}$

$(\Uparrow_{0}^{n'-1}(\text{r0w0})^{2c})^{\lfloor n/n' \rfloor -1}$

$\Uparrow_{0}^{(n \bmod n')-1}(\text{r0w0})^{2c}$

M2 : $\Uparrow_{0}^{n'-1}(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}$

$(\Uparrow_{0}^{n'-1}(\text{r1w1})^{2c})^{\lfloor n/n' \rfloor -1}$

$\Uparrow_{0}^{(n \bmod n')-1}(\text{r1w1})^{2c}$

M3 : $\Uparrow_{0}^{n'-1}(\text{r1w0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}$

$(\Uparrow_{0}^{n'-1}(\text{r0w0})^{2c})^{\lfloor n/n' \rfloor -1}$

$\Uparrow_{0}^{(n \bmod n')-1}(\text{r0w0})^{2c}$

M4 : $\Downarrow_{0}^{n'-1}(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}$

$(\Downarrow_{0}^{n'-1}(\text{r1w1})^{2c})^{\lfloor n/n' \rfloor -1}$

$\Downarrow_{0}^{(n \bmod n')-1}(\text{r1w1})^{2c}$

M5 : $\Downarrow_{0}^{n'-1}(\text{r1w0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}$

$(\Downarrow_{0}^{n'-1}(\text{r0w0})^{2c})^{\lfloor n/n' \rfloor -1}$

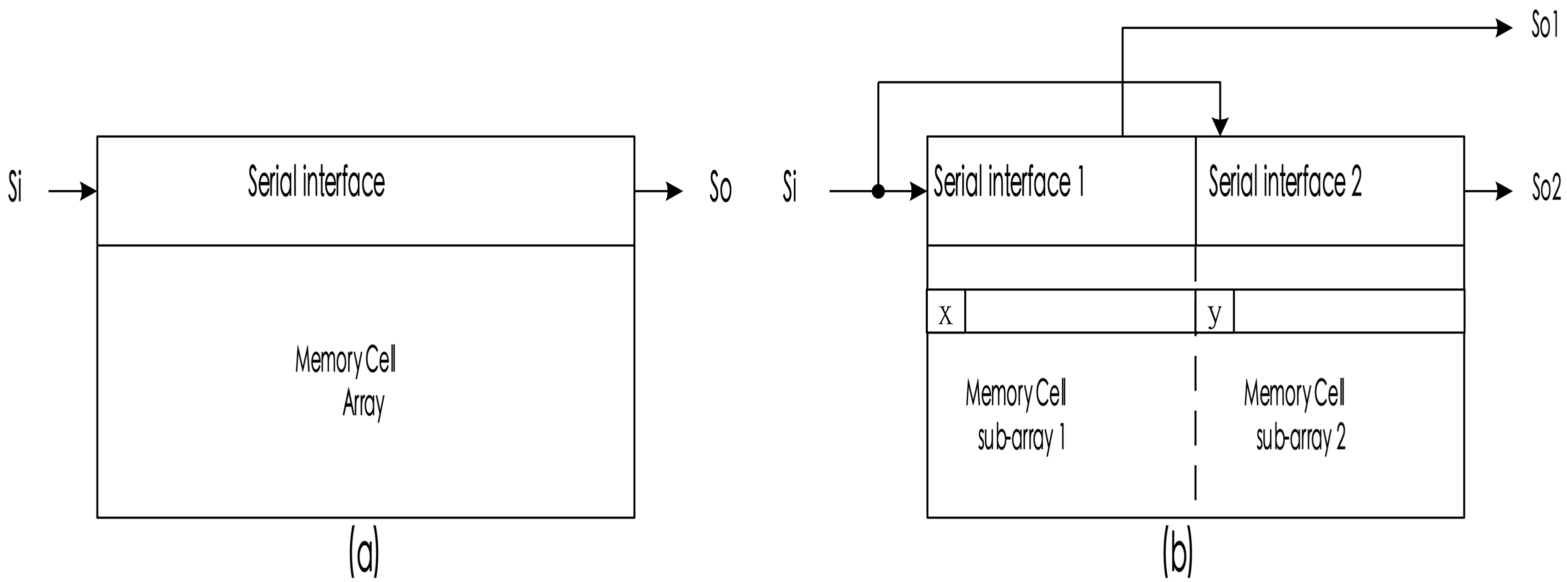
$\Downarrow_{0}^{(n \bmod n')-1}(\text{r0w0})^{2c}$

M6 : $\Downarrow_{0}^{n'-1}(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}$

$(\Downarrow_{0}^{n'-1}(\text{r1w1})^{2c})^{\lfloor n/n' \rfloor -1}$

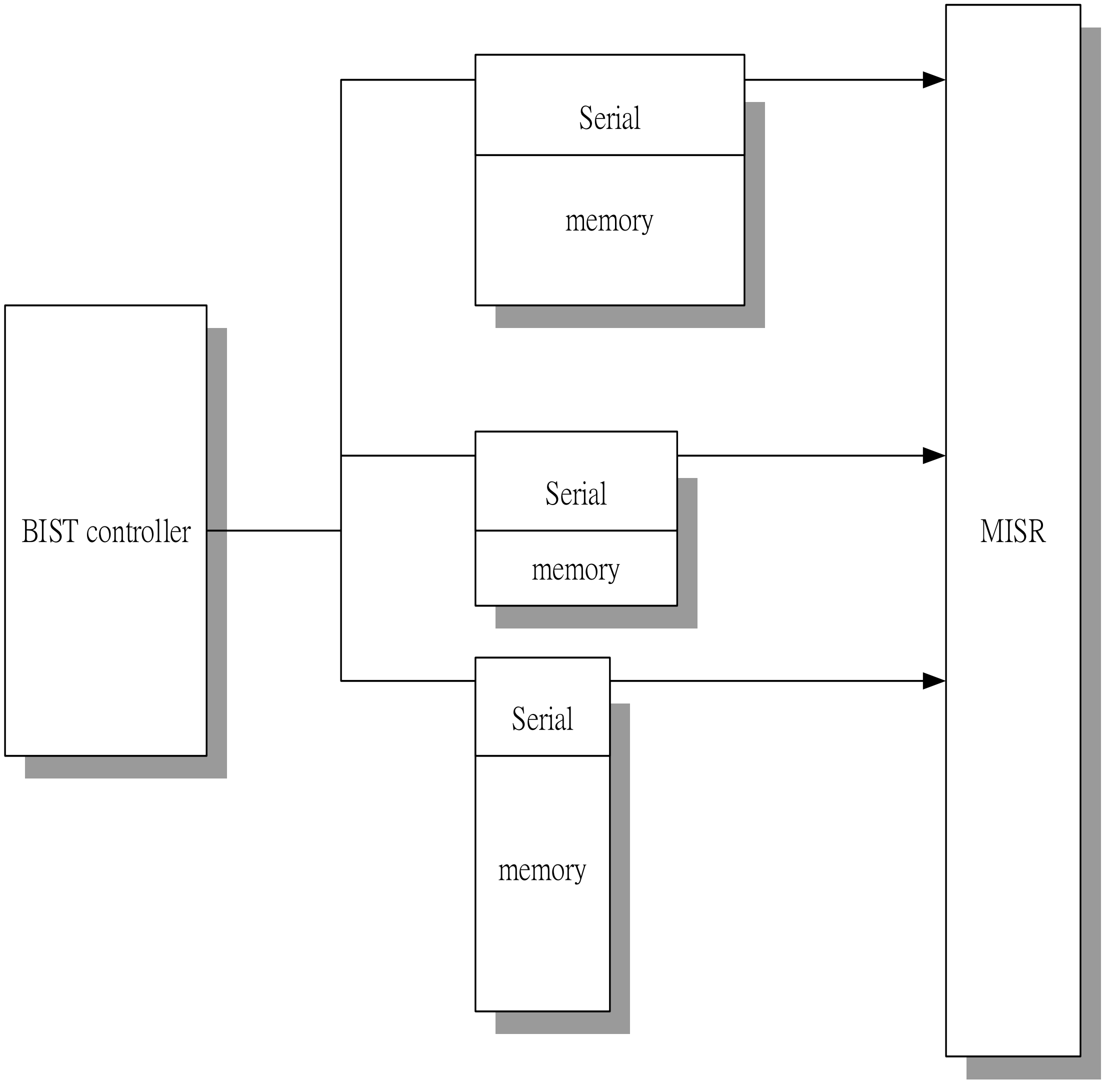
$\Downarrow_{0}^{(n \bmod n')-1}(\text{r1w1})^{2c}$

Split Mode Test



xy:00, 11, 01,10

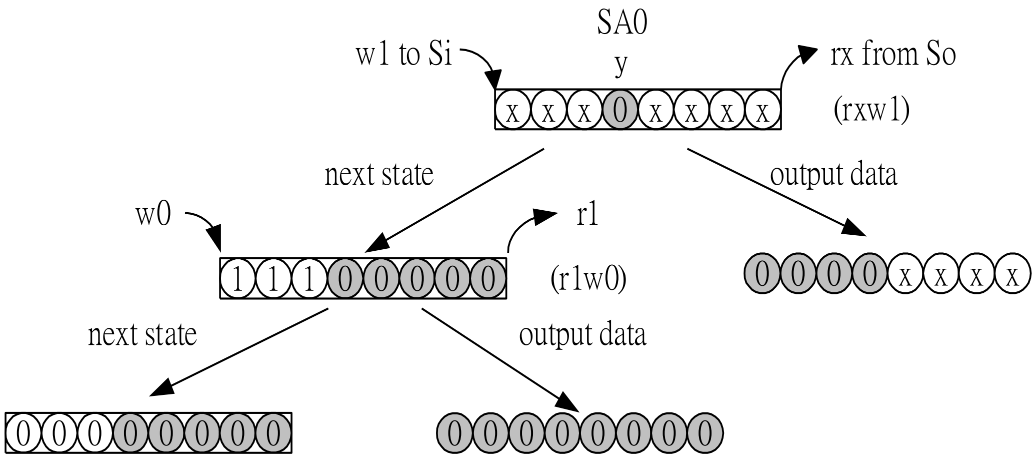
Parallel BIST Method



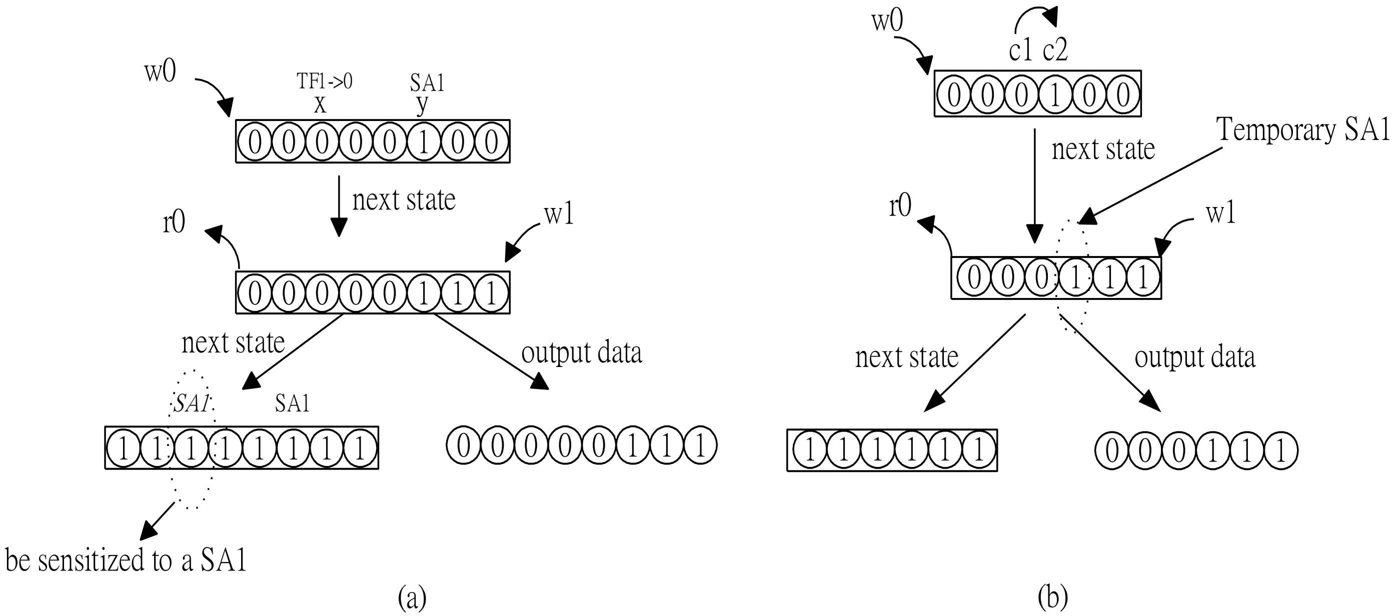
Parallel BISD Method

An Off-Line Solution

A serial fault masking effect



(1) The serial fault masking effects for SAF

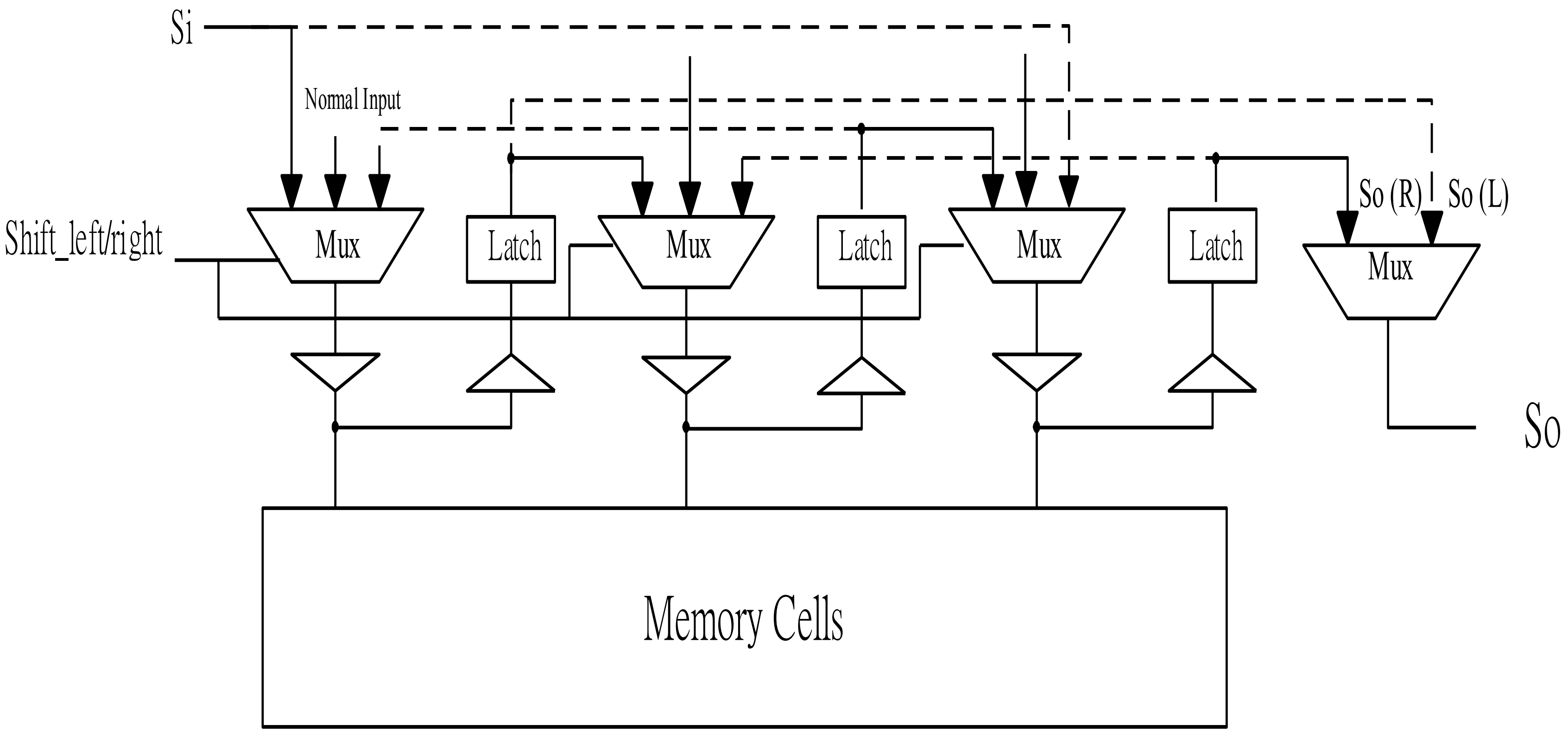


(a)

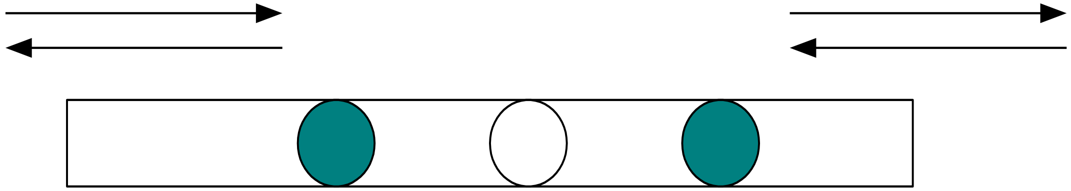
(b)

(2) The serial fault masking effects by TF and CFst faults.

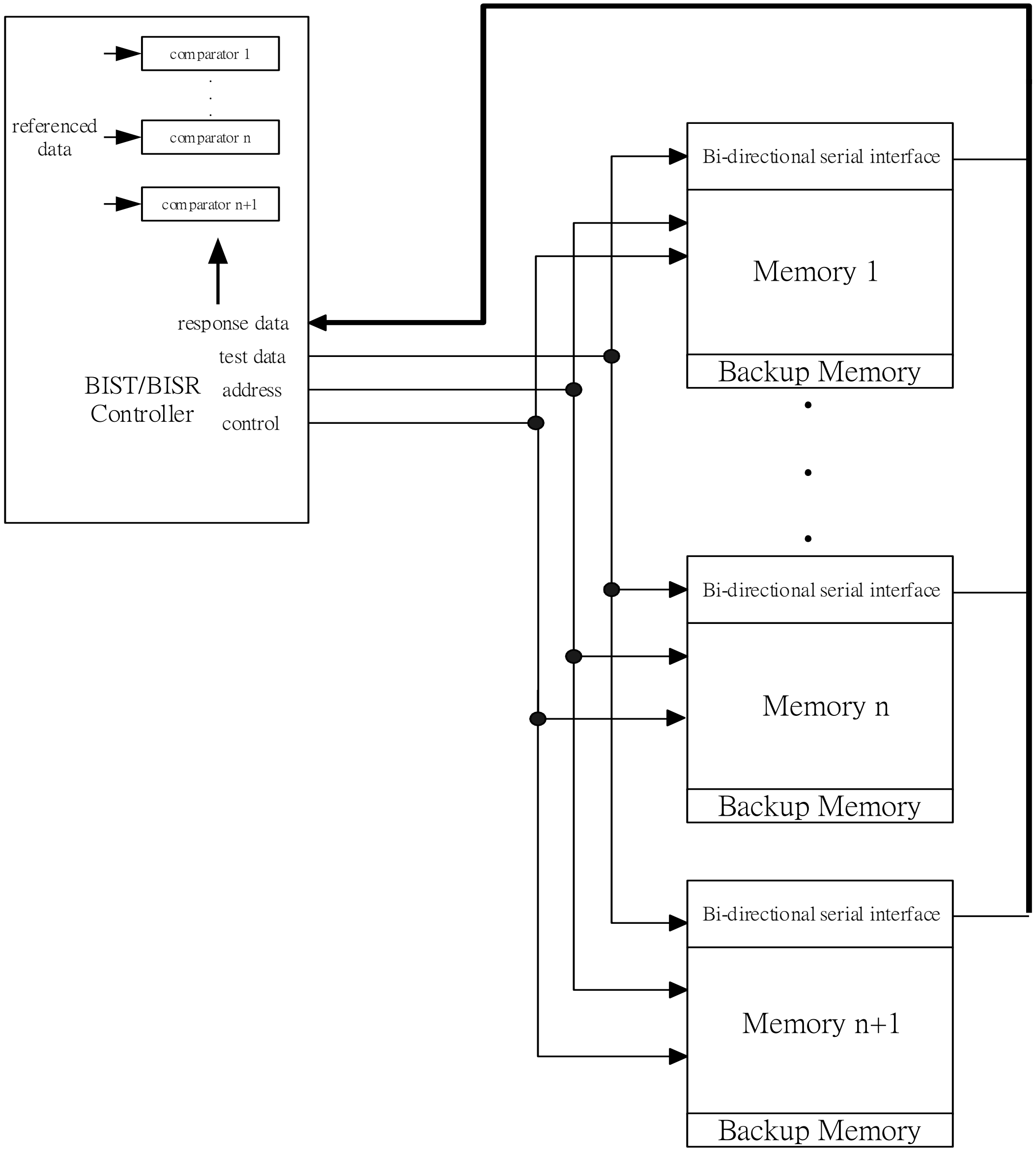
Bi-directional Serial Interface



Diagnosing the faulty cells from the outermost side.



A Parallel Diagnosis Scheme



DiagRSMarch Algorithm

$$M1: \Uparrow_0^{n'-1}\left(\begin{array}{l}
[(\text{rxw0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}]_R \\
[(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}]_L \\
[(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}]_R \\
[(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}]_L \\
[(\text{r11w01})^{c'}(\text{r01w01})^{c-c'}(\text{r01w01})^{c'}(\text{r01w01})^{c-c'}]_R \\
[(\text{r01w10})^{c'}(\text{r10w10})^{c-c'}(\text{r10w10})^{c'}(\text{r10w10})^{c-c'}]_L \\
[(\text{r10w10})^{c'}(\text{r10w10})^{c-c'}(\text{r10w10})^{c'}(\text{r10w10})^{c-c'}]_R \\
[(\text{r10w01})^{c'}(\text{r01w01})^{c-c'}(\text{r01w01})^{c'}(\text{r01w01})^{c-c'}]_L \\
[(\text{r01w00})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}]_R
\end{array}\right)_{\text{repeat}} \quad \text{M1-1} \sim \text{M1-9}$$

$$\{\Uparrow_0^{n'-1}\left(\begin{array}{l}
[(\text{rxw0})^{c'}(\text{r0w0})^{2c-c'}]_R[(\text{r0w0})^{2c}]_L[(\text{r0w1})^{c'}(\text{r1w1})^{2c-c'}]_R[(\text{r1w1})^{2c}]_L[(\text{r11w01})^{c'}(\text{r01w01})^{2c-c'}]_R \\
[(\text{r01w10})^{c'}(\text{r10w10})^{2c-c'}]_L[(\text{r10w10})^{2c}]_R[(\text{r10w01})^{c'}(\text{r01w01})^{2c-c'}]_L[(\text{r01w00})^{c'}(\text{r0w0})^{2c-c'}]_R
\end{array}\right)\text{repeat}\}^{\lfloor n/n'\rfloor-1}$$

$$\Uparrow_0^{(n \bmod n')-1}\left(\begin{array}{l}
[(\text{rxw0})^{c'}(\text{r0w0})^{2c-c'}]_R[(\text{r0w0})^{2c}]_L[(\text{r0w1})^{c'}(\text{r1w1})^{2c-c'}]_R[(\text{r1w1})^{2c}]_L[(\text{r11w01})^{c'}(\text{r01w01})^{2c-c'}]_R \\
[(\text{r01w10})^{c'}(\text{r10w10})^{2c-c'}]_L[(\text{r10w10})^{2c}]_R[(\text{r10w01})^{c'}(\text{r01w01})^{2c-c'}]_L[(\text{r01w00})^{c'}(\text{r0w0})^{2c-c'}]_R
\end{array}\right)\text{repeat}$$

$$M2: \Uparrow_0^{n'-1}(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}$$

$$\{\Uparrow_0^{n'-1}[(\text{r1w1})^{2c}]\}^{\lfloor n/n'\rfloor-1}$$

$$\Uparrow_0^{(n \bmod n')-1}(\text{r1w1})^{2c}$$

$$M3: \Uparrow_0^{n'-1}(\text{r1w0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}$$

$$\{\Uparrow_0^{n'-1}[(\text{r0w0})^{2c}]\}^{\lfloor n/n'\rfloor-1}$$

$$\Uparrow_0^{(n \bmod n')-1}(\text{r0w0})^{2c}$$

$$M4: \Downarrow_0^{n'-1}(\text{r0w1})^{c'}(\text{r1w1})^{c-c'}(\text{r1w1})^{c'}(\text{r1w1})^{c-c'}$$

$$\{\Downarrow_0^{n'-1}[(\text{r1w1})^{2c}]\}^{\lfloor n/n'\rfloor-1}$$

$$\Downarrow_0^{(n \bmod n')-1}(\text{r1w1})^{2c}$$

$$M5: \Downarrow_0^{n'-1}(\text{r1w0})^{c'}(\text{r0w0})^{c-c'}(\text{r0w0})^{c'}(\text{r0w0})^{c-c'}$$

$$\{\Downarrow_0^{n'-1}[(\text{r0w0})^{2c}]\}^{\lfloor n/n'\rfloor-1}$$

$$\Downarrow_0^{(n \bmod n')-1}(\text{r0w0})^{2c}$$

$$M6: \Downarrow_0^{n'-1}(\text{r0wx})^{c'}(\text{rxwx})^{c-c'}(\text{rxwx})^{c'}(\text{rxwx})^{c-c'}$$

$$\{\Downarrow_0^{n'-1}[(\text{rxwx})^{2c}]\}^{\lfloor n/n'\rfloor-1}$$

$$\Downarrow_0^{(n \bmod n')-1}(\text{rxwx})^{2c}$$

/* L : shift left, R: shift right */

Parallel Transparent BIST Method

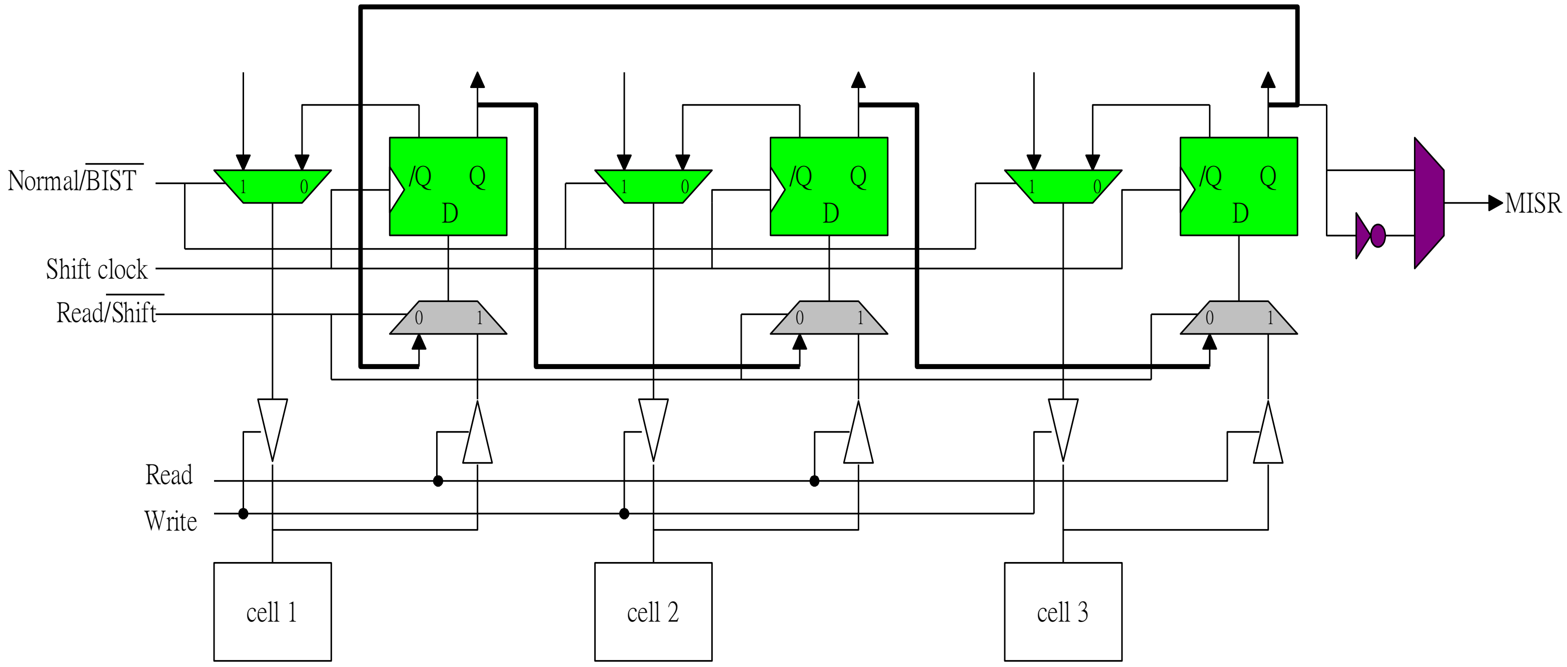
An On-Line Solution

Transparent Test Interface

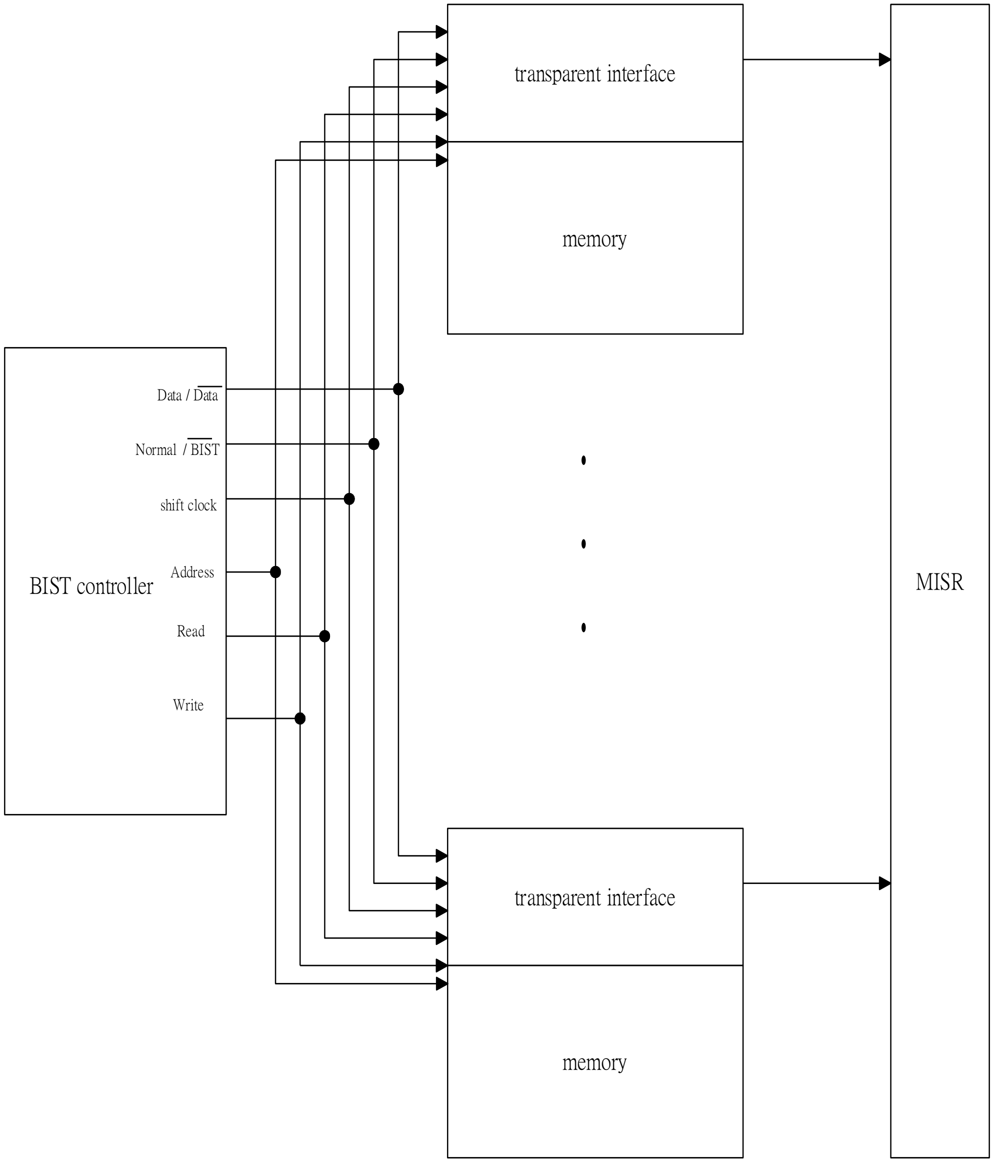
circular chain: 🟦

test pattern generation: 🟩

signature predictable: 🟪


Normal/$\overline{\text{BIST}}$
Shift clock
Read/$\overline{\text{Shift}}$
Read
Write
/Q
Q
D
1
0
cell 1
cell 2
cell 3
MISR

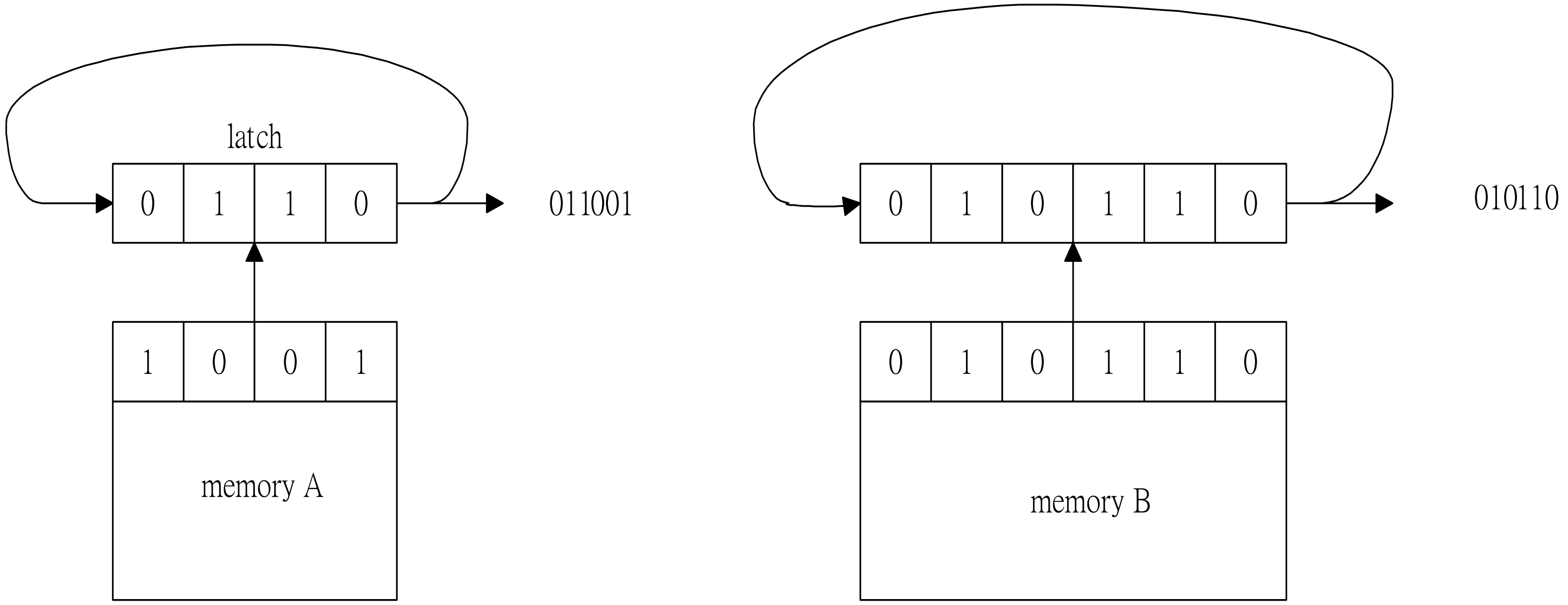
Transparent BIST Scheme



The Horizontal TRSMarch

$[r(1001)s(1001)^{4}s(1001)^{6-4}]_{circle}$

$[r(010110)s(010110)^{6}]_{circle}$



latch

0 1 1 0

011001

1 0 0 1

memory A

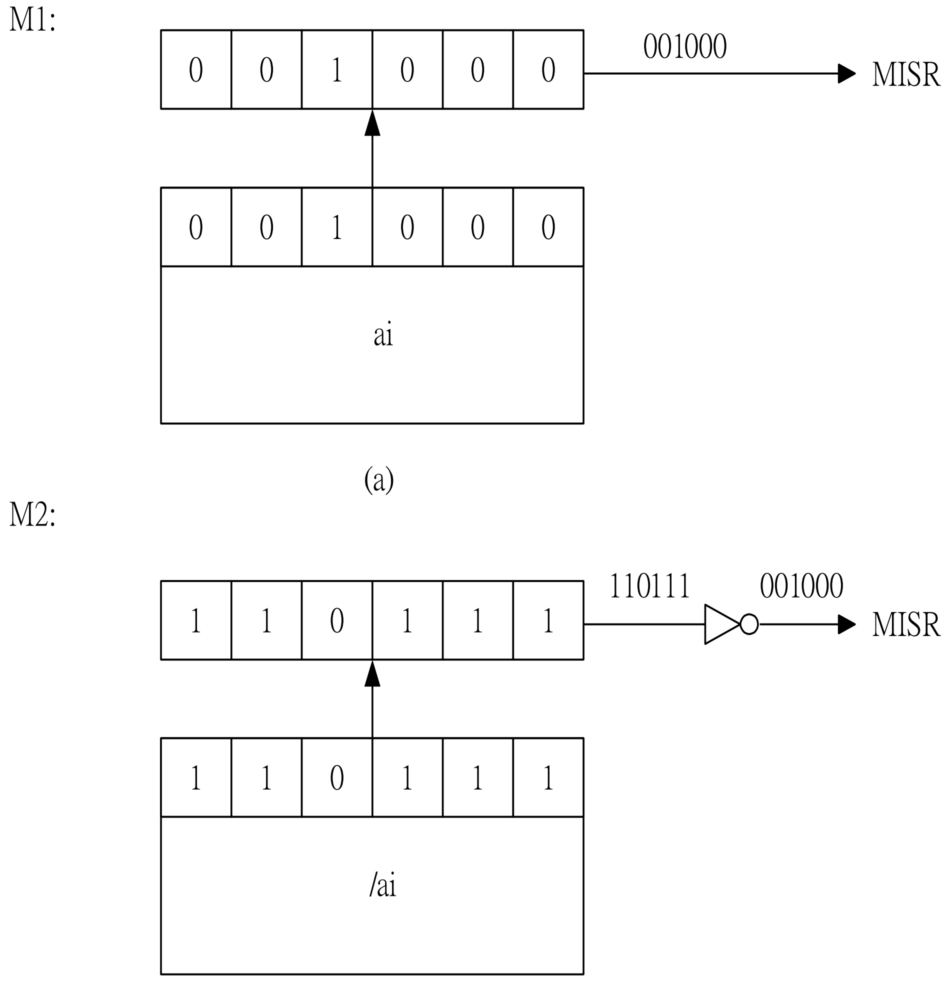
0 1 0 1 1 0

010110

0 1 0 1 1 0

memory B

The Identification



(a)

(b)

The TRSMarch

$$M1 : \Uparrow_0^{n'-1} \overbrace{r(ai)[s(ai)^{c} s(ai)^{c\text{-}c'}]_{\text{circle}}}^{\text{Check tested result}} \overbrace{[r(ai)w(/ai)]}^{\text{Exercitation}}$$

$$M2 : \Uparrow_0^{n'-1} r(/ai)[s(/ai)^{c} s(/ai)^{c\text{-}c'}]_{\text{circle}} \ [r(/ai)w(ai)]$$

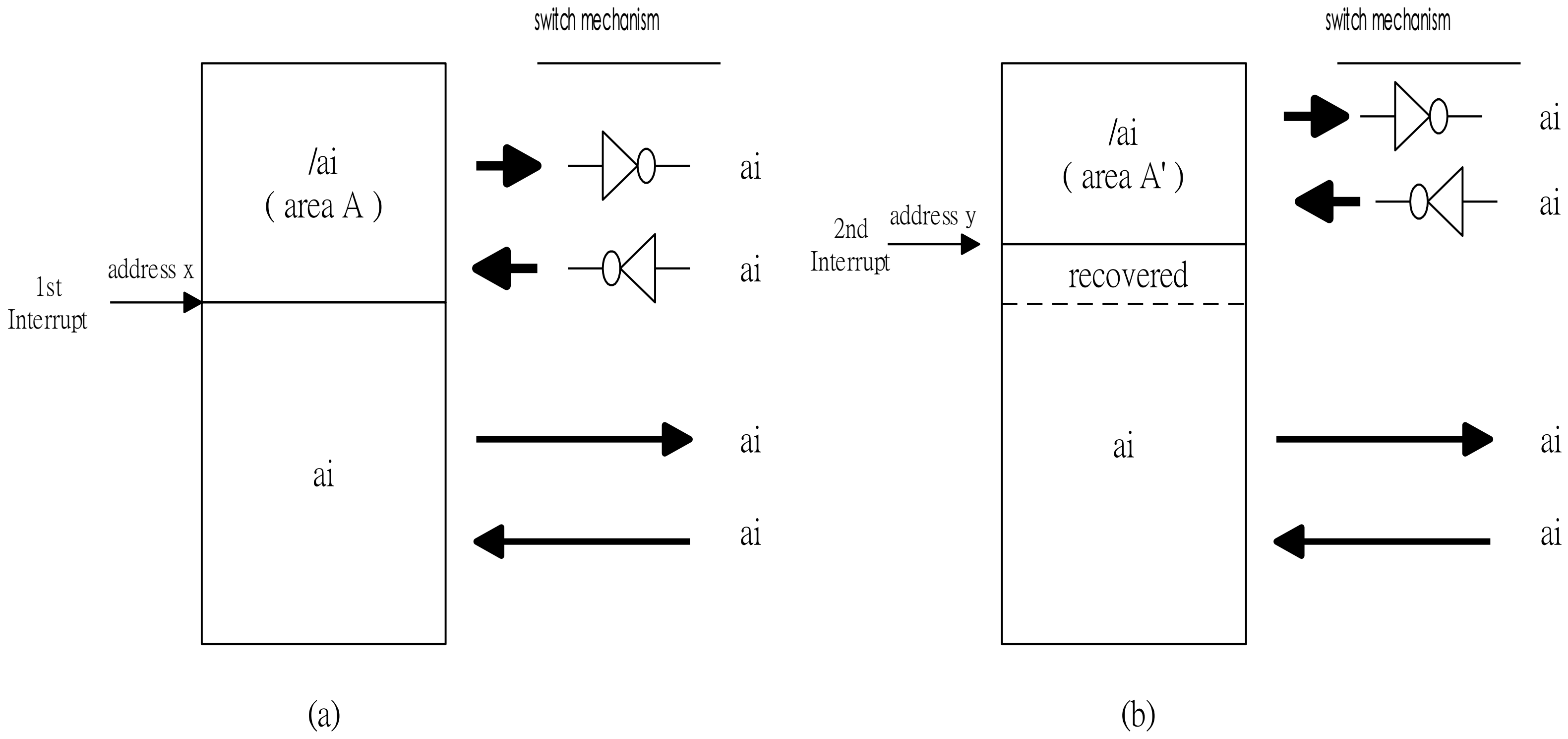
$$M3 : \Uparrow_0^{n'-1} r(ai)[s(ai)^{c} s(ai)^{c\text{-}c'}]_{\text{circle}}$$

$$M4 : \Downarrow_0^{n'-1} r(ai)[s(ai)^{c} s(ai)^{c\text{-}c'}]_{\text{circle}} \ [r(ai)w(/ai)]$$

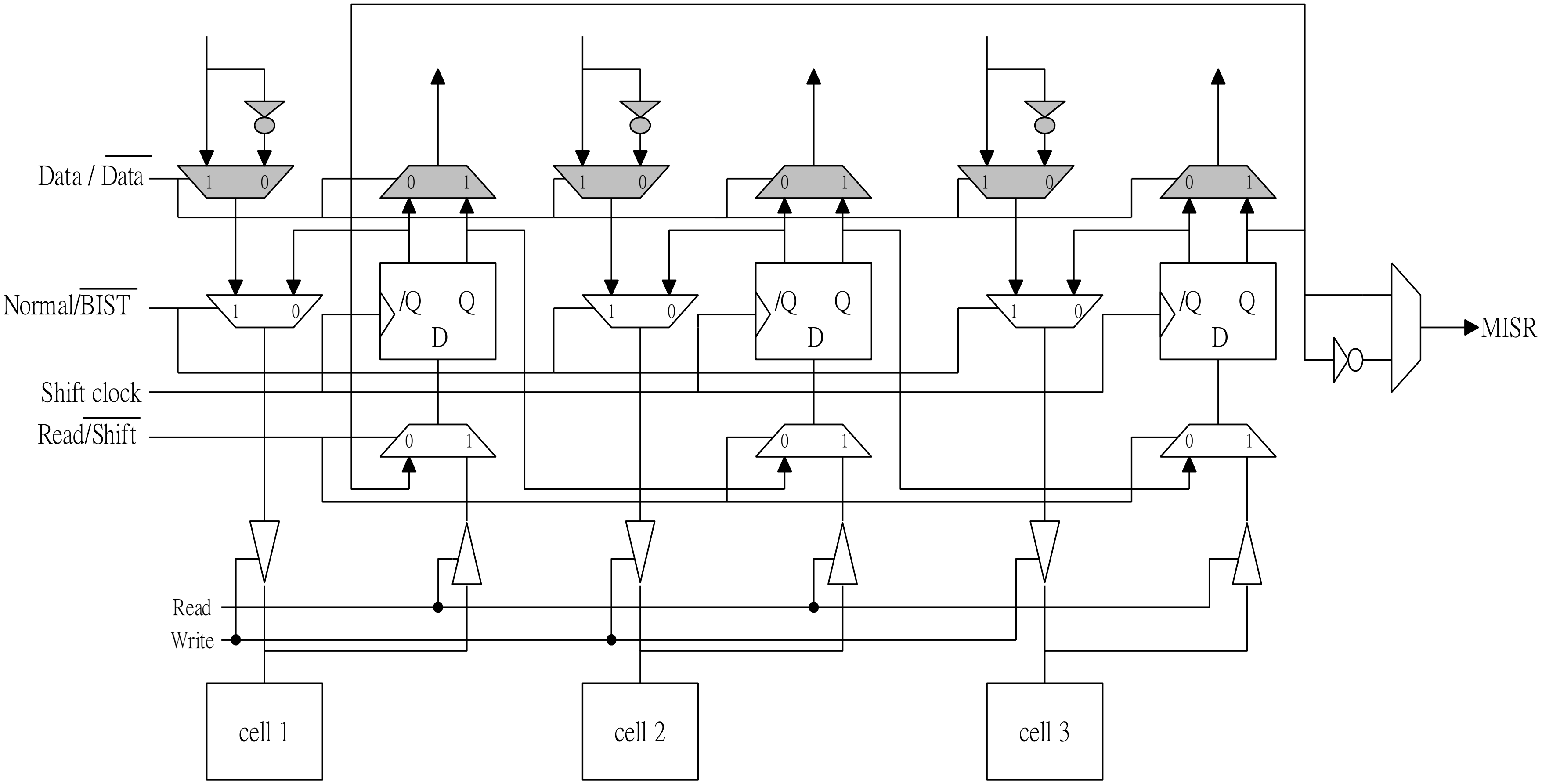
$$M5 : \Downarrow_0^{n'-1} r(/ai)[s(/ai)^{c} s(/ai)^{c\text{-}c'}]_{\text{circl e}} [r(/ai)w(ai)]$$

$$M6 : \Downarrow_0^{n'-1} r(ai)[s(ai)^{c} s(ai)^{c\text{-}c'}]_{\text{circle}}$$

An Interrupt Scheme



An Interrupt Interface


Data / $\overline{\text{Data}}$
Normal/$\overline{\text{BIST}}$
Shift clock
Read/$\overline{\text{Shift}}$
Read
Write
/Q
Q
D
1
0
MISR
cell 1
cell 2
cell 3

Fault Coverage Analysis

Theorem 15: For a CFst in the same word, the detection probability equals to $1-(3/4)^{n}$ while performing the TRSMarch algorithm n times.

Ex. If n=10, then $1-(3/4)^{10} = 0.944$ (i.e., 94.4%).

An Eight TRSMarch

$M1 : \Uparrow_0^{n'-1} r(ai)[s(ai)^c s(ai)^{c-c'}]_{circle}\ [r(ai)w(/ai)]$

$M2 : \Uparrow_0^{n'-1} r(/ai)[s(/ai)^c s(/ai)^{c-c'}]_{circle}\ [r(/ai)w(ai)]$

$M3 : \Uparrow_0^{n'-1} r(ai)[s(ai)^c s(ai)^{c-c'}]_{circle}$

$M4 : \Downarrow_0^{n'-1} r(ai)[s(ai)^c s(ai)^{c-c'}]_{circle}\ [r(ai)w(/ai)]$

$M5 : \Downarrow_0^{n'-1} r(/ai)[s(/ai)^c s(/ai)^{c-c'}]_{circl\ e}\ [r(/ai)w(ai)]$

$M6 : \Downarrow_0^{n'-1} r(ai)[s(ai)^c s(ai)^{c-c'}]_{circle}\ [r(ai)w(ei)]$

$M7 : \Downarrow_0^{n'-1} r(ei)[s(ei)^c s(ei)^{c-c'}]_{circle}\ [r(ei)w(/ai)]$

$M8 : \Downarrow_0^{n'-1} r(/ai)[s(/ai)^c s(ai)^{c-c'}]_{circle}\ [r(/ai)w(ai)]$

note: the format of ai is QQ, /ai is /Q/Q and ei is Q/Q

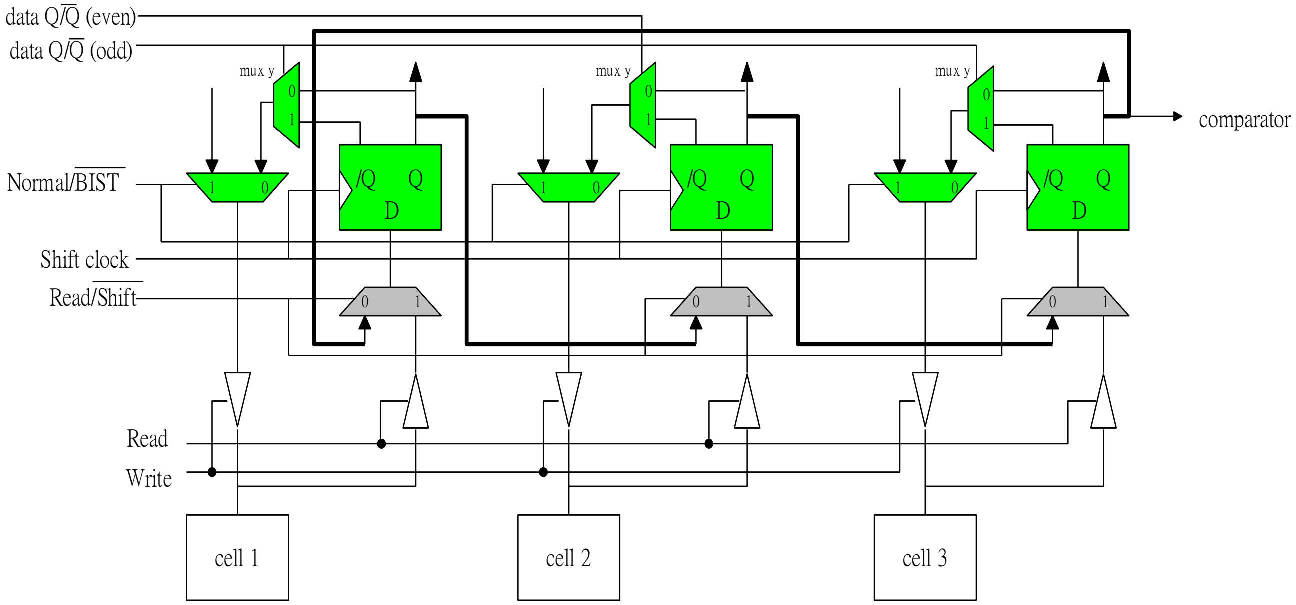
Parallel Transparent BISD Method

An On-Line Solution

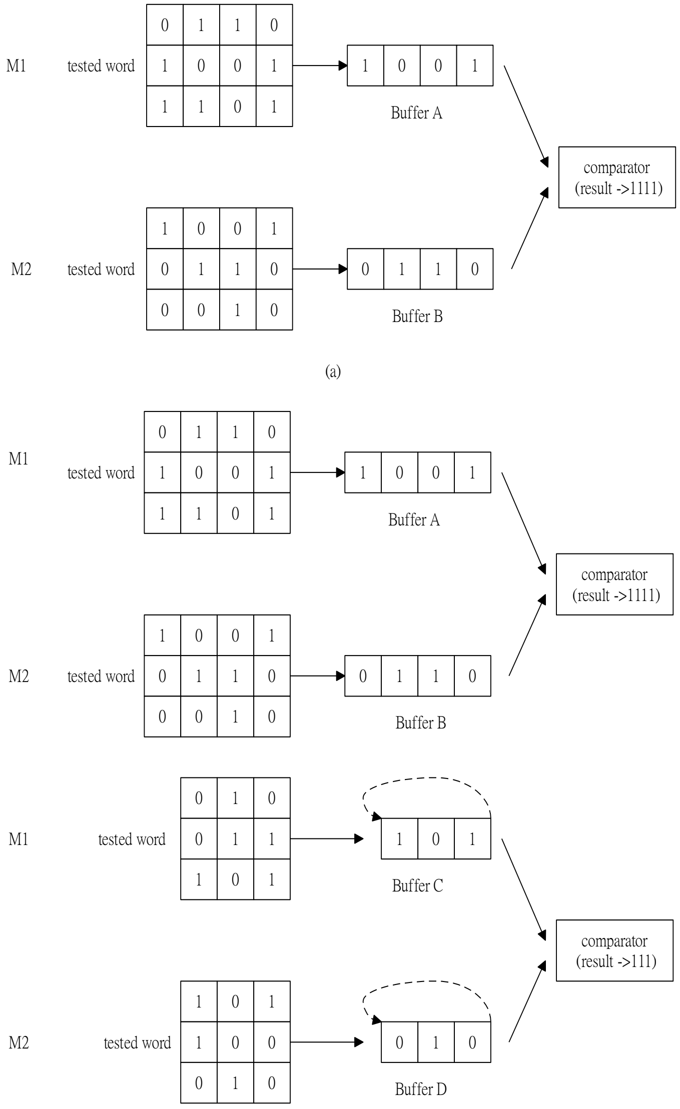
A Transparent Diagnostic Interface

circular chain: 🟦

test pattern generation: 🟩



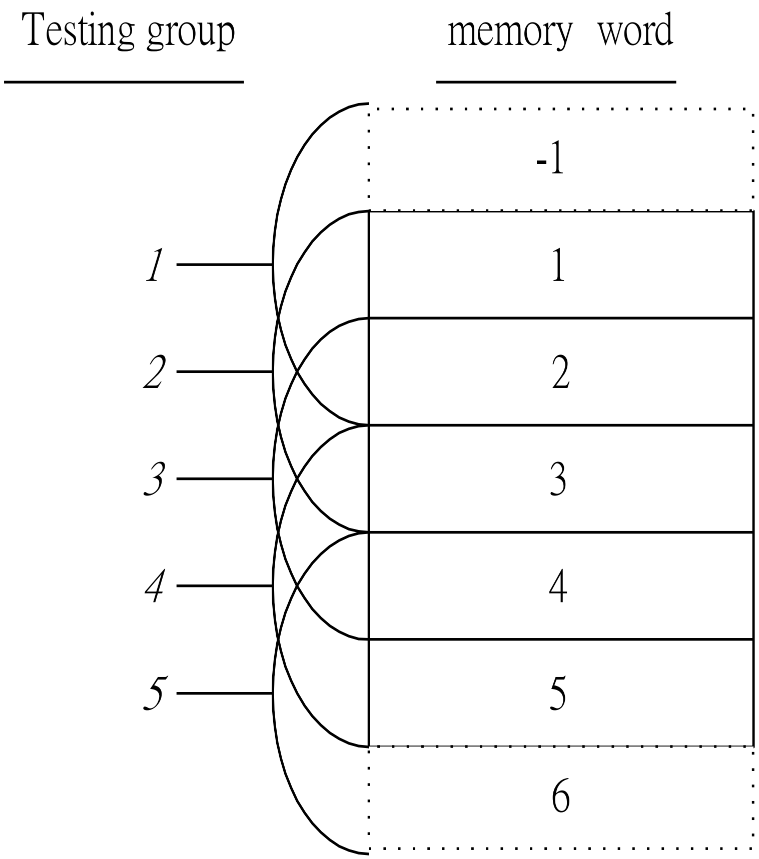
A Circular Comparing Scheme



(a)

(b)

A Window Based Method



A Seven March TDiagRSMarch

$$\mathrm{M1}: \quad \Uparrow_{j}^{j+2}\{r(ai)[s(ai)^{c}s(ai)^{c\text{-}c'}]_{circle}\}_{j+1}[r(ai)w(/ai)]$$

$$\mathrm{M2}: \quad \Uparrow_{j}^{j+2}\{r(/ai)[s(/ai)^{c}s(/ai)^{c\text{-}c'}]_{circle}\}_{j+1}[r(/ai)w(ai)]$$

$$\mathrm{M3}: \quad \Downarrow_{j}^{j+2}\{r(ai)[s(ai)^{c}s(ai)^{c\text{-}c'}]_{circle}\}_{j+1}[r(ai)w(/ai)]$$

$$\mathrm{M4}: \quad \Downarrow_{j}^{j+2}\{r(/ai)[s(/ai)^{c}s(/ai)^{c\text{-}c'}]_{circle}\}_{j+1}[r(/ai)w(ai)]$$

$$\mathrm{M5}: \quad \Uparrow_{j}^{j+2}\{r(ai)[s(ai)^{c}s(ai)^{c\text{-}c'}]_{circle}\}_{j+1}[r(ai)w(di)]$$

$$\mathrm{M6}: \quad \Uparrow_{j}^{j+2}\{r(di)[s(di)^{c}s(di)^{c\text{-}c'}]_{circle}\}_{j+1}[r(di)w(/di)]$$

$$\mathrm{M7}: \quad \Downarrow_{j}^{j+2}\{r(/di)[s(/di)^{c}s(/di)^{c\text{-}c'}]_{circle}\}_{j+1}[r(/di)w(ai)]$$

A Five March TDiagRSMarch

$$\mathrm{M1:}\quad \Uparrow_{j}^{j+2} \{ \mathrm{r(ai\ )[\,s(ai\ )^{c}s(ai\ )^{c\text{-}c'}]_{circle}} \}_{\mathrm{j+1}} [r(\mathrm{ai\ })w(/\mathrm{ai\ })]$$

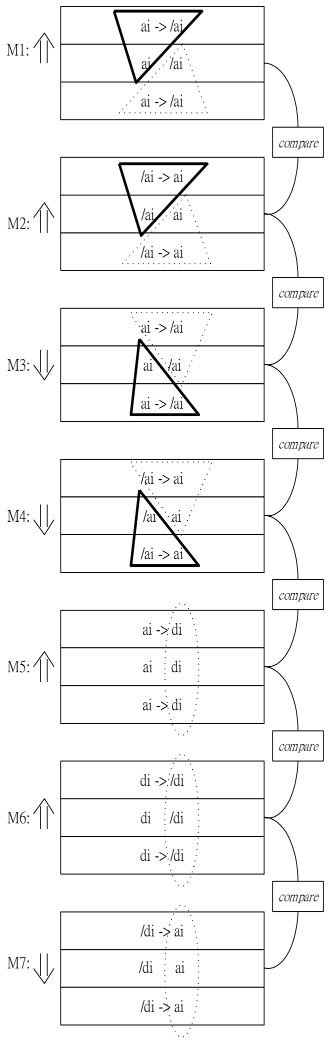
$$\mathrm{M2:}\quad \Uparrow_{j}^{j+2} \{ \mathrm{r(/ai\ )[s(/ai\ )^{c}s(/ai\ )^{c\text{-}c'}]_{circle}} \}_{\mathrm{j+1}} [r(/\mathrm{ai\ })w(\mathrm{ai\ })]$$

$$\mathrm{M3:}\quad \Downarrow_{j}^{j+2} \{ \mathrm{r(ai\ )[\,s(ai\ )^{c}s(ai\ )^{c\text{-}c'}]_{circle}} \}_{\mathrm{j+1}} [r(\mathrm{ai\ })w(/\mathrm{ai\ })]$$

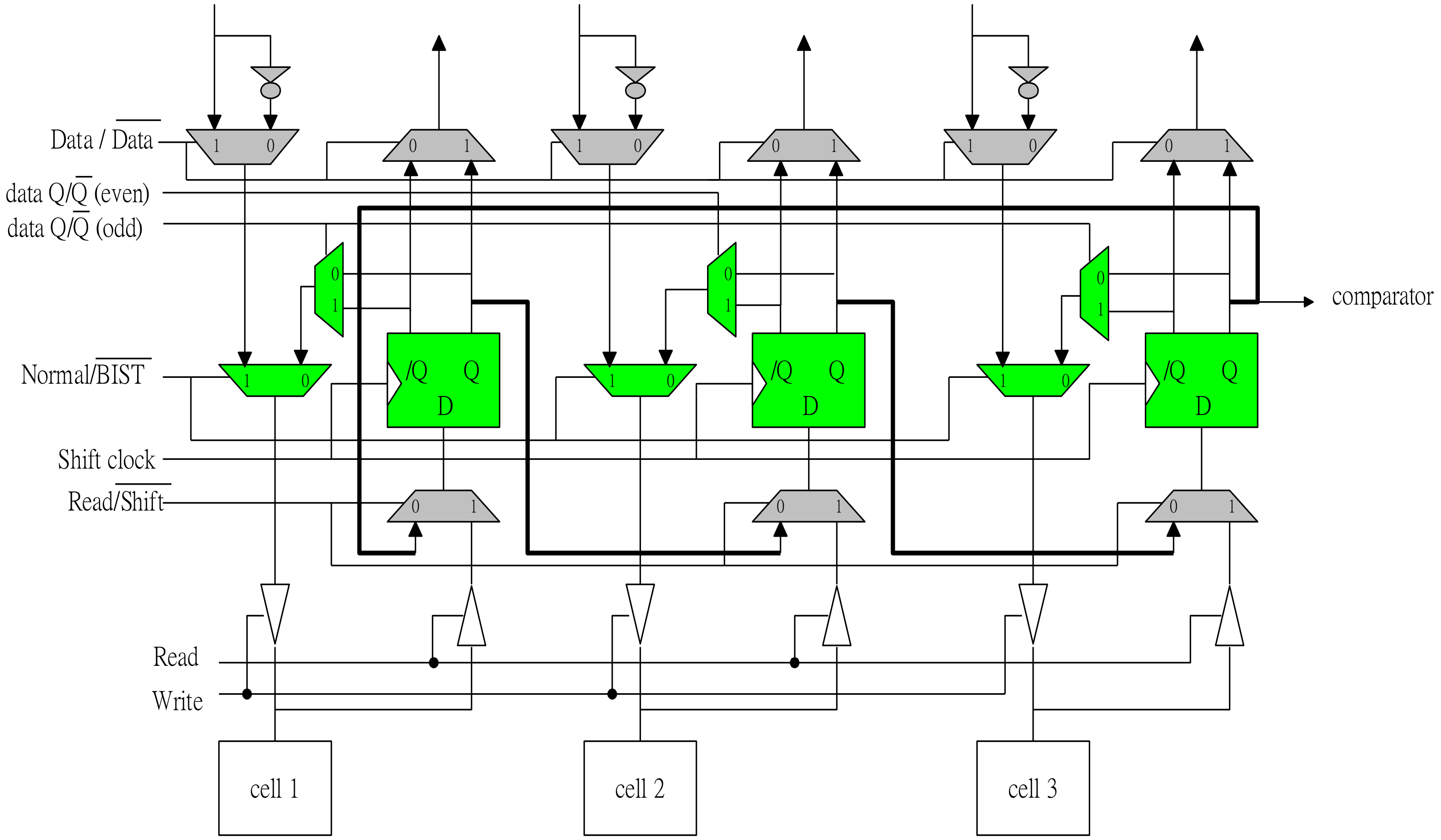
$$\mathrm{M4:}\quad \Downarrow_{j}^{j+2} \{ \mathrm{r(/ai\ )[s(/ai\ )^{c}s(/ai\ )^{c\text{-}c'}]_{circle}} \}_{\mathrm{j+1}} [r(/\mathrm{ai\ })w(\mathrm{ai\ })]$$

$$\mathrm{M5:}\quad \Uparrow_{j}^{j+2} \{ \mathrm{r(ai\ )[\,s(ai\ )^{c}s(ai\ )^{c\text{-}c'}]_{circle}} \}_{\mathrm{j+1}}$$

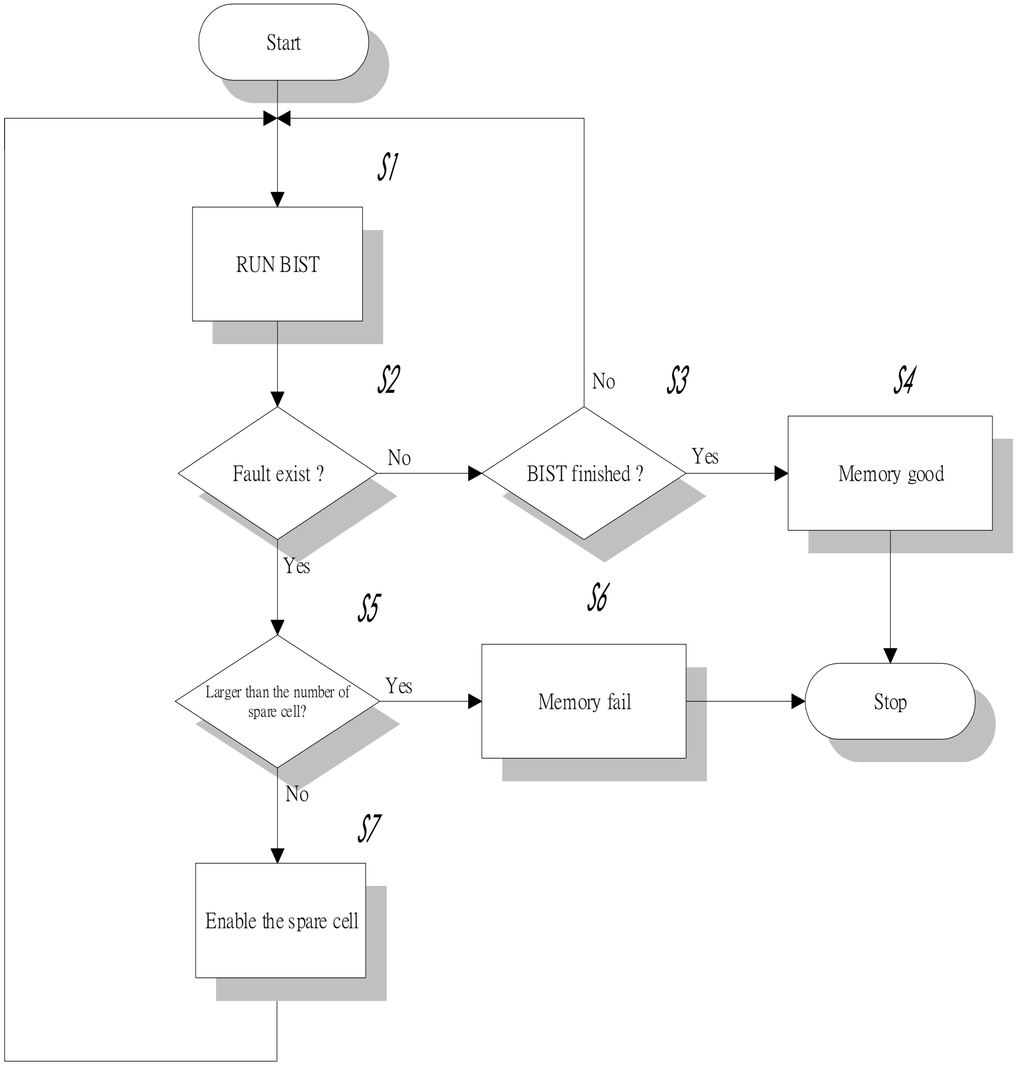
An Identify Scheme



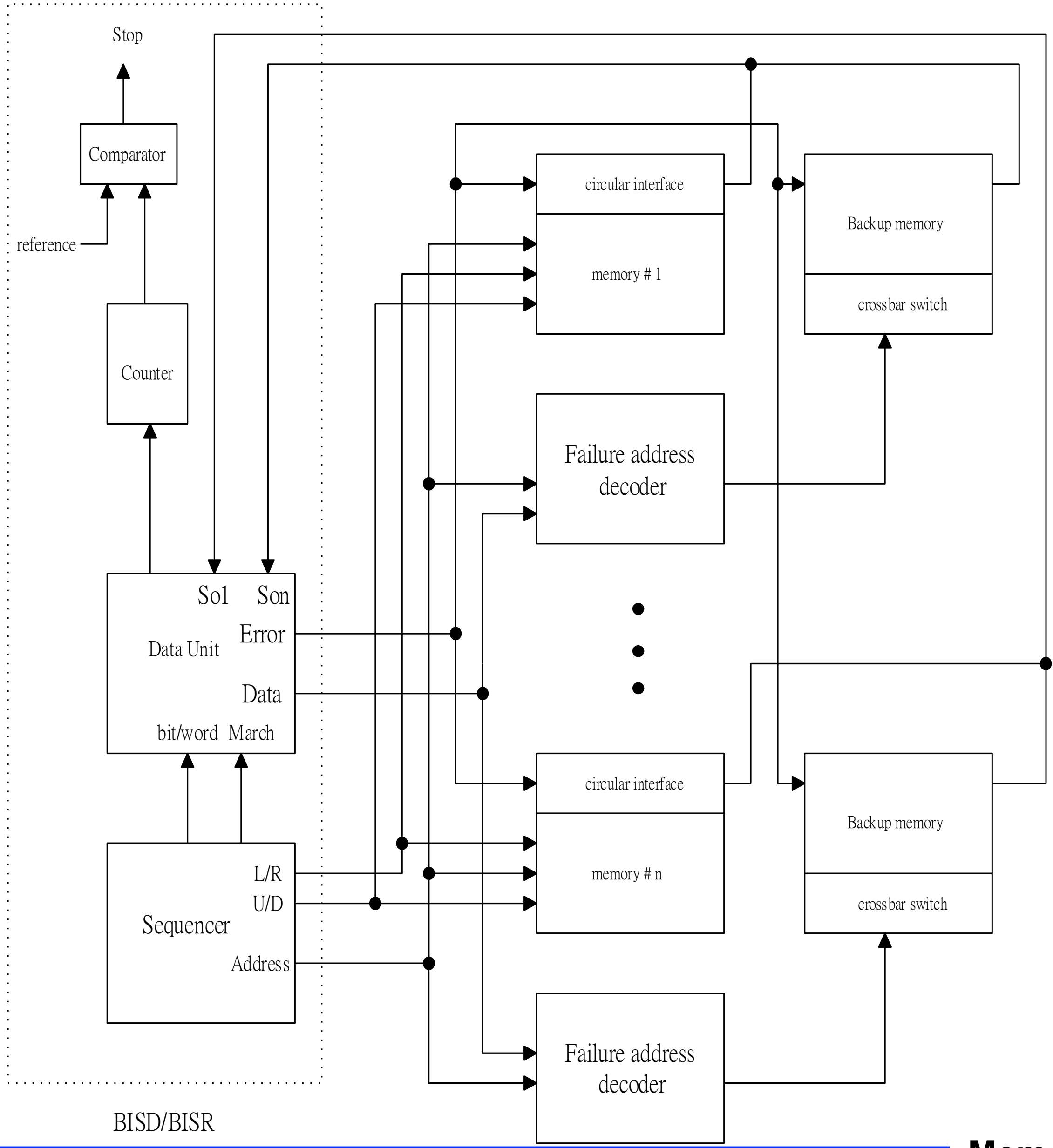
An Interrupt Interface



A Self-Repair Flow Chart



A BIST/BISD Scheme


Stop
Comparator
reference
Counter
Sol
Son
Data Unit
Error
Data
bit/word
March
L/R
U/D
Sequencer
Address
BISD/BISR
circular interface
memory # 1
Backup memory
crossbar switch
Failure address decoder
circular interface
memory # n
Backup memory
crossbar switch
Failure address decoder