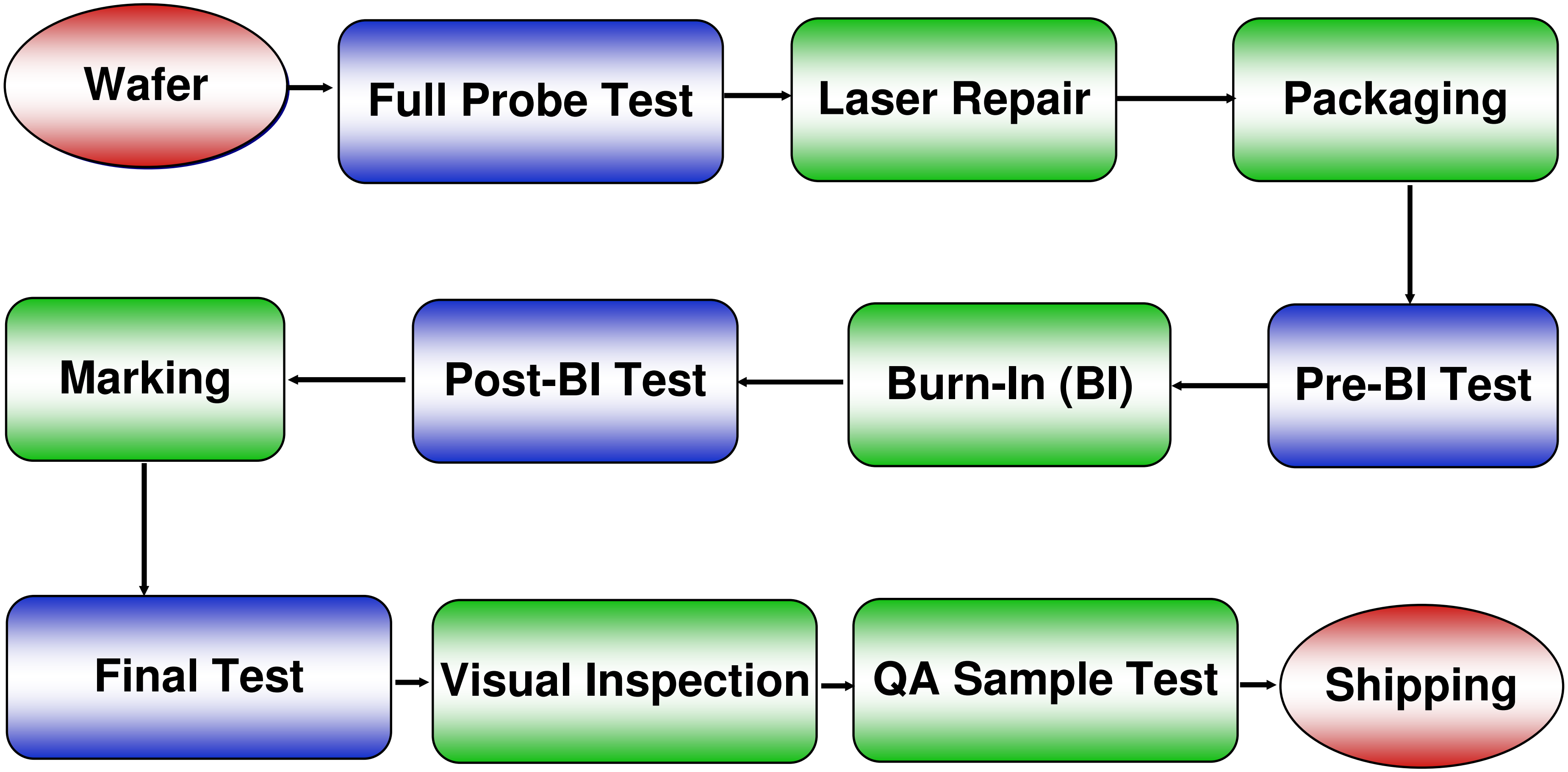
Chapter 8

Memory Testing and Built-In Self-Test

What is this chapter about?

- ☐ Basic concepts of memory testing and BIST
- ☐ Memory fault models and test algorithms
- ☐ Memory fault simulation and test algorithm generation
  - RAMSES: fault simulator
  - TAGS: test algorithm generator
- ☐ Memory BIST
  - BRAINS: BIST generator

Typical RAM Production Flow



Wafer

Full Probe Test

Laser Repair

Packaging

Pre-BI Test

Burn-In (BI)

Post-BI Test

Marking

Final Test

Visual Inspection

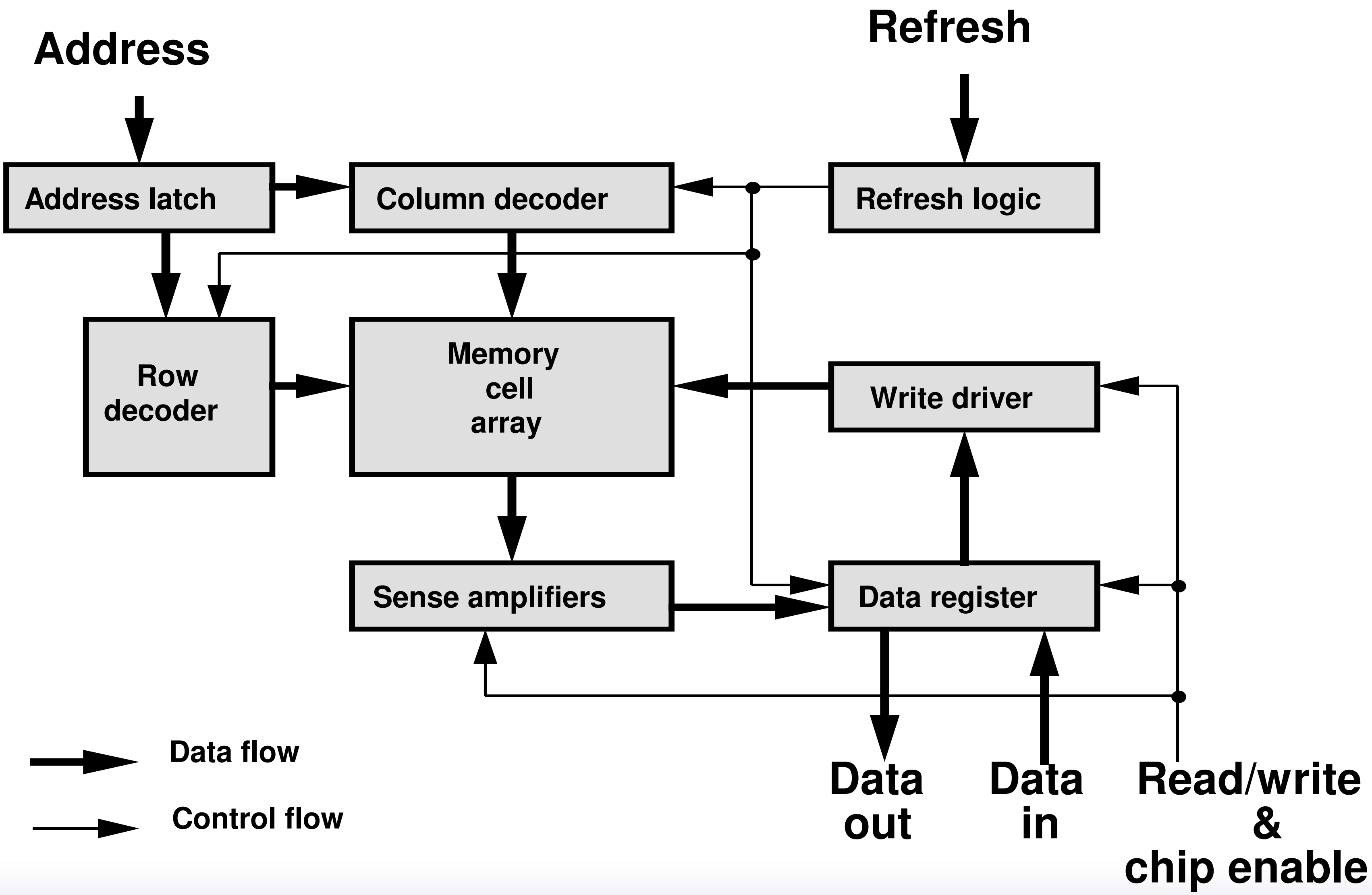
QA Sample Test

Shipping

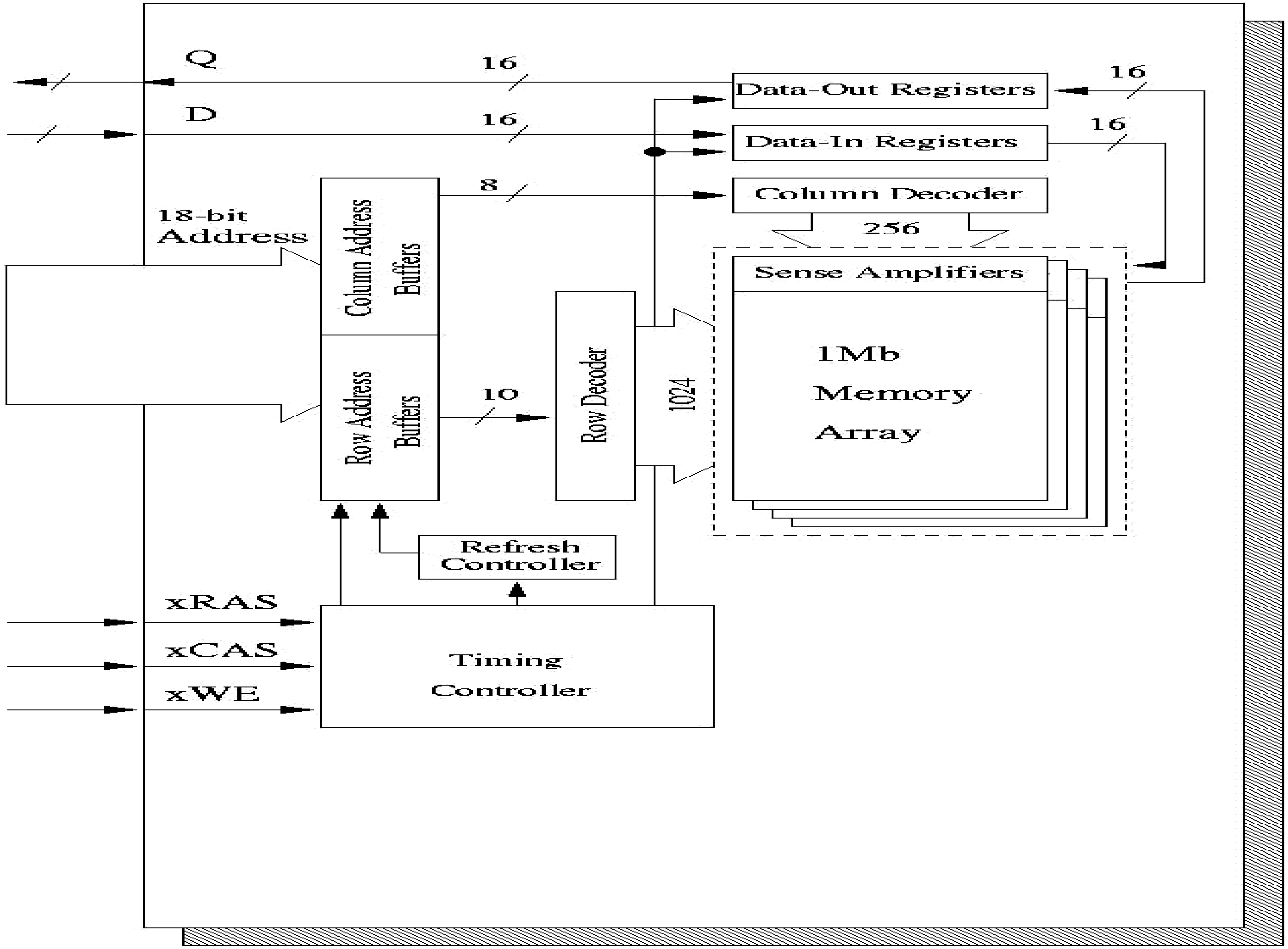
Off-Line Testing of RAM

- Parametric Test: DC & AC
- Reliability Screening
  - Long-cycle testing
  - Burn-in: static & dynamic BI
- Functional Test
  - Device characterization
    - Failure analysis
  - Fault modeling
    - Simple but effective (accurate & realistic?)
  - Test algorithm generation
    - Small number of test patterns (data backgrounds)
    - High fault coverage
    - Short test time

DRAM Functional Model



DRAM Functional Model Example



Functional Fault Models

- Classical fault models are not sufficient to represent all important failure modes in RAM.
- Sequential ATPG is not possible for RAM.
- Functional fault models are commonly used for memories:
  - They define functional behavior of faulty memories.
- New fault models are being proposed to cover new defects and failures in modern memories:
  - New process technologies
  - New devices

Static RAM Fault Models: SAF/TF

- Stuck-At Fault (SAF)
  - Cell (line) SA0 or SA1
    - A stuck-at fault (SAF) occurs when the value of a cell or line is always 0 (a stuck-at-0 fault) or always 1 (a stuck-at-1 fault).
    - A test that detects all SAFs guarantees that from each cell, a 0 and a 1 must be read.
- Transition Fault (TF)
  - Cell fails to transit from 0 to 1 or 1 to 0 in specified time period.
    - A cell has a transition fault (TF) if it fails to transit from 0 to 1 (a $<\uparrow/0>$ TF) or from 1 to 0 (a $<\downarrow/1>$ TF).

Static RAM Fault Models: AF

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

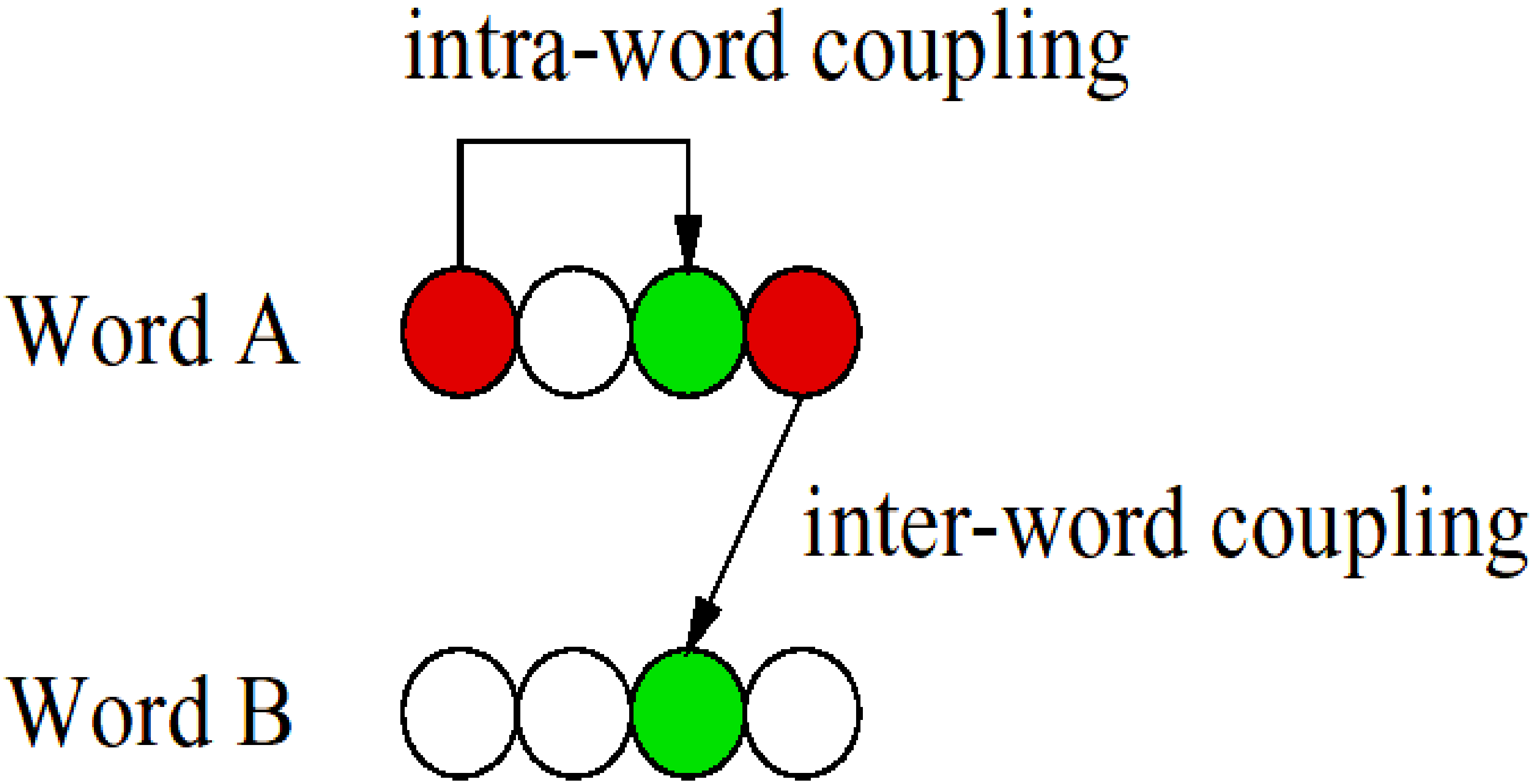
Static RAM Fault Models: SOF

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

RAM Fault Models: CF

- Coupling Fault (CF)
  - A coupling fault (CF) between two cells occurs when the logic value of a cell is influenced by the content of, or operation on, another cell.
  - State Coupling Fault (CFst)
    - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state.
  - Inversion Coupling Fault (CFin)
    - Transition in coupling cell complements (inverts) coupled cell.
  - Idempotent Coupling Fault (CFid)
    - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0.

Intra-Word & Inter-Word CFs



RAM Fault Models: DF

- Disturb Fault (DF)
  - Victim cell forced to 0 or 1 if we (successively) read or write aggressor cell (may be the same cell):
    - Hammer test
  - Read Disturb Fault (RDF)
    - There is a read disturb fault (RDF) if the cell value will flip when being read (successively).

RAM Fault Models: DRF

- Data Retention Fault (DRF)
  - DRAM
    - Refresh Fault
    - Leakage Fault
  - SRAM
    - Leakage Fault
      - Static Data Losses---defective pull-up

Test Time Complexity (100MHz)

Size	N	10N	NlogN	$N^{1.5}$	N^2
1M	0.01s	0.1s	0.2s	11s	3h
16M	0.16s	1.6s	3.9s	11m	33d
64M	0.66s	6.6s	17s	1.5h	1.43y
256M	2.62s	26s	1.23m	12h	23y
1G	10.5s	1.8m	5.3m	4d	366y
4G	42s	7m	22.4m	32d	57c
16G	2.8m	28m	1.6h	255d	915c

RAM Test Algorithm

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

March Test Notation

- ⇑: address sequence is in the ascending order
- ⇓: address changes in the descending order
- ⇕: address sequence is either ⇑ or ⇓
- r: the Read operation
  - Reading an expected 0 from a cell (r0); reading an expected 1 from a cell (r1)
- w: the Write operation
  - Writing a 0 into a cell (w0); writing a 1 into a cell (w1)
- Example (MATS+): $\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$

Classical Test Algorithms: MSCAN

- Zero-One Algorithm [Breuer & Friedman 1976]
  - Also known as MSCAN
  - SAF is detected if the address decoder is correct (not all AFs are covered):
    - Theorem: A test detects all AFs if it contains the march elements ⇑(ra,…,wb) and ⇓(rb,…,wa), and the memory is initialized to the proper value before each march element
  - Solid background (pattern)
  - Complexity is 4N

$$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(w1);\Updownarrow(r1)\}$$

Classical Test Algorithms: Checkerboard

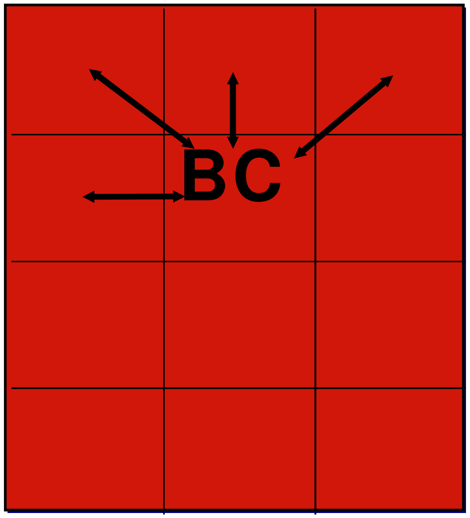
- Checkerboard Algorithm
  - Zero-one algorithm with checkerboard pattern
  - Complexity is 4N
  - Must create true physical checkerboard, not logical checkerboard
  - For SAF, DRF, shorts between cells, and half of the TFs
    - Not good for AFs, and some CFs cannot be detected

1	0	1
0	1	0
1	0	1

Classical Test Algorithms: GALPAT

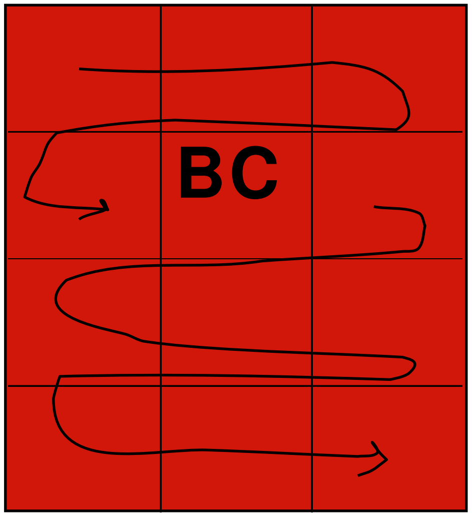
- Galloping Pattern (GALPAT)
  - Complexity is 4N**2—only for characterization
  - A strong test for most faults: all AFs, TFs, CFs, and SAFs are detected and located

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



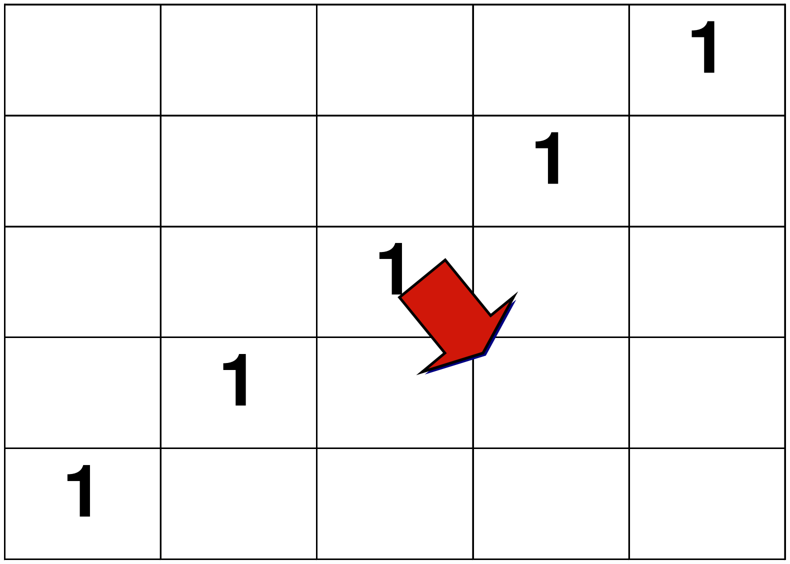
Classical Test Algorithms: WALPAT

- Walking Pattern (WALPAT)
  - Similar to GALPAT, except that BC is read only after all others are read.
  - Complexity is 2N**2.



Classical Test Algorithms: Sliding

- Sliding (Galloping) Row/Column/Diagonal
  - Based on GALPAT, but instead of shifting a 1 through the memory, a complete diagonal of 1s is shifted:
    - The whole memory is read after each shift
  - Detects all faults as GALPAT, except for some CFs
  - Complexity is 4N**1.5.

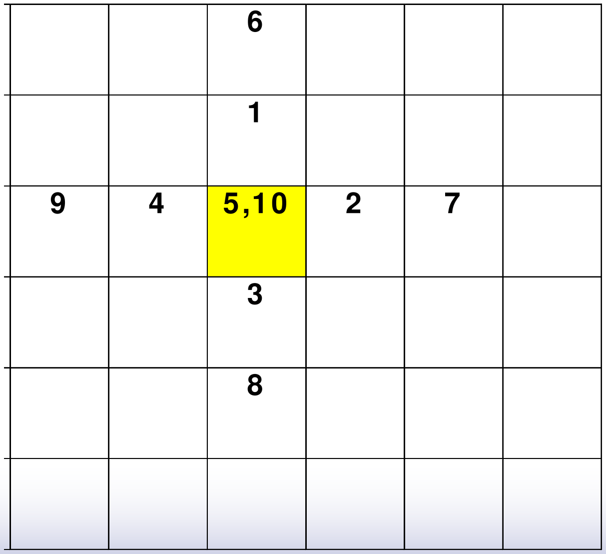


				1
			1	
		1		
	1			
1				

Classical Test Algorithms: Butterfly

- ❑ Butterfly Algorithm
  - Complexity is 5NlogN
  - All SAFs and some AFs are detected

```
1. Write background 0;
2. For BC = 0 to N-1
   { Complement BC; dist = 1;
     While dist <= mdist     /* mdist < 0.5 col/row length */
       {  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 *= 2; }
     Complement BC; }
3. Write background 1;  repeat Step 2;
```



Classical Test Algorithms: MOVI

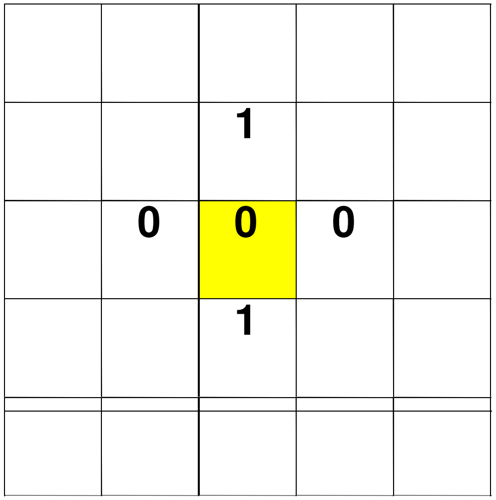
- Moving Inversion (MOVI) Algorithm
  - For functional and AC parametric test
    - Functional (13N): for AF, SAF, TF, and most CF

      $\{\Downarrow(w0);\Uparrow(r0,w1,r1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1);\Downarrow(r1,w0,r0)\}$
    - Parametric (12NlogN): for Read access time
      - 2 successive Reads @ 2 different addresses with different data for all 2-address sequences differing in 1 bit
      - Repeat T2~T5 for each address bit
      - GALPAT---all 2-address sequences

Classical Test Algorithms: SD

- ❑ Surround Disturb Algorithm
  - Examine how the cells in a row are affected when complementary data are written into adjacent cells of neighboring rows.
  - Designed on the premise that DRAM cells are most susceptible to interference from their nearest neighbors (eliminates global sensitivity checks).

```
1. For each cell[p,q]   /* row p and column q */
   { Write 0 in cell[p,q-1];
     Write 0 in cell[p,q];
     Write 0 in cell[p,q+1];
     Write 1 in cell[p-1,q];
     Read 0 from cell[p,q+1];
     Write 1 in cell[p+1,q];
     Read 0 from cell[p,q-1];
     Read 0 from cell[p,q]; }
2. Repeat Step 1 with complementary data;
```



Simple March Tests

- Zero-One (MSCAN)
- Modified Algorithmic Test Sequence (MATS)
  - OR-type address decoder fault

$$\{\Updownarrow(w0);\Updownarrow(r0,w1);\Updownarrow(r1)\}$$

  - AND-type address decoder fault

$$\{\Updownarrow(w1);\Updownarrow(r1,w0);\Updownarrow(r0)\}$$

- MATS+
  - For both OR- & AND-type AFs and SAFs
  - The suggested test for unlinked SAFs

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$$

March Tests: Marching-1/0

- Marching-1/0
  - Marching-1: begins by writing a background of 0s, then read and write back complement values (and read again to verify) for all cells (from cell 0 to n-1, and then from cell n-1 to 0), in 7N time
  - Marching-0: follows exactly the same pattern, with the data reversed
  - For AF, SAF, and TF (but only part of the CFs)
  - It is a *complete test*, i.e., all faults that should be detected are covered
  - It however is a *redundant test*, because only the first three march elements are necessary

$$\{\Uparrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);$$
$$\Uparrow(w1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1)\}$$

March Tests: MATS++

- ❑ MATS++
  - Also for AF, SAF, and TF
  - Optimized marching-1/0 scheme—complete and irredundant
  - Similar to MATS+, but allow for the coverage of TFs
  - The suggested test for unlinked SAFs & TFs

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)\}$$

March Tests: March X/C

- March X
  - Called March X because the test has been used without being published
  - For AF, SAF, TF, & CFin

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

- March C
  - For AF, SAF, TF, & all CFs, but semi-optimal (redundant)

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);$$

$$\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

March Tests: March C-

- March C-
  - Remove the redundancy in March C
  - Also for AF, SAF, TF, & all CFs
  - Optimal (irredundant)

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

- Extended March C-
  - Covers SOF in addition to the above faults

$$\{\Updownarrow(w0);\Uparrow(r0,w1,r1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

Fault Detection Summary

Name	Faults detected	Algorithm
MATS++	SAF/AF	$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)$
March X	AF/SAF/TF/CFin	$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$
March Y	AF/SAF/TF/CFin	$\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)$
March C–	SAF/AF/TF/CF	$\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$

Comparison of March Tests

	MATS++	March X	March Y	March C-
SAF	∨	∨	∨	∨
TF	∨	∨	∨	∨
AF	∨	∨	∨	∨
SOF	∨		∨	
CFin		∨	∨	∨
CFid				∨
CFst				∨

Word-Oriented Memory

- A word-oriented memory has Read/Write operations that access the memory cell array by a word instead of a bit.
- Word-oriented memories can be tested by applying a bit-oriented test algorithm repeatedly with a set of different data backgrounds:
  - The repeating procedure multiplies the testing time

Testing Word-Oriented RAM

- Background bit is replaced by background word
  - MATS++: $\{\Updownarrow(wa);\Uparrow(ra,wb);\Downarrow(rb,wa,ra)\}$
- Conventional method is to use logm+1 different backgrounds for m-bit words
  - Called *standard backgrounds*
  - m=8: 00000000, 01010101, 00110011, and 00001111
  - Apply the test algorithm logm+1=4 times, so complexity is 4*6N/8=3N

Note: *b* is the complement of *a*

Cocktail-March Algorithms

- Motivation:
  - Repeating the same algorithm for all logm+1 backgrounds is redundant so far as intra-word coupling faults are concerned
  - Different algorithms target different faults.
- Approaches:
  1. Use multiple backgrounds in a single algorithm run
  2. Merge and forge different algorithms and backgrounds into a single algorithm

- Good for word-oriented memories

Ref: Wu *et al.*, IEEE TCAD, 04/02

March-CW

- Algorithm:
  - March C- for solid background (0000)
  - Then a 5N March for each of other standard backgrounds (0101, 0011):

$$\{\Updownarrow(wa, wb, rb, wa, ra)\}$$

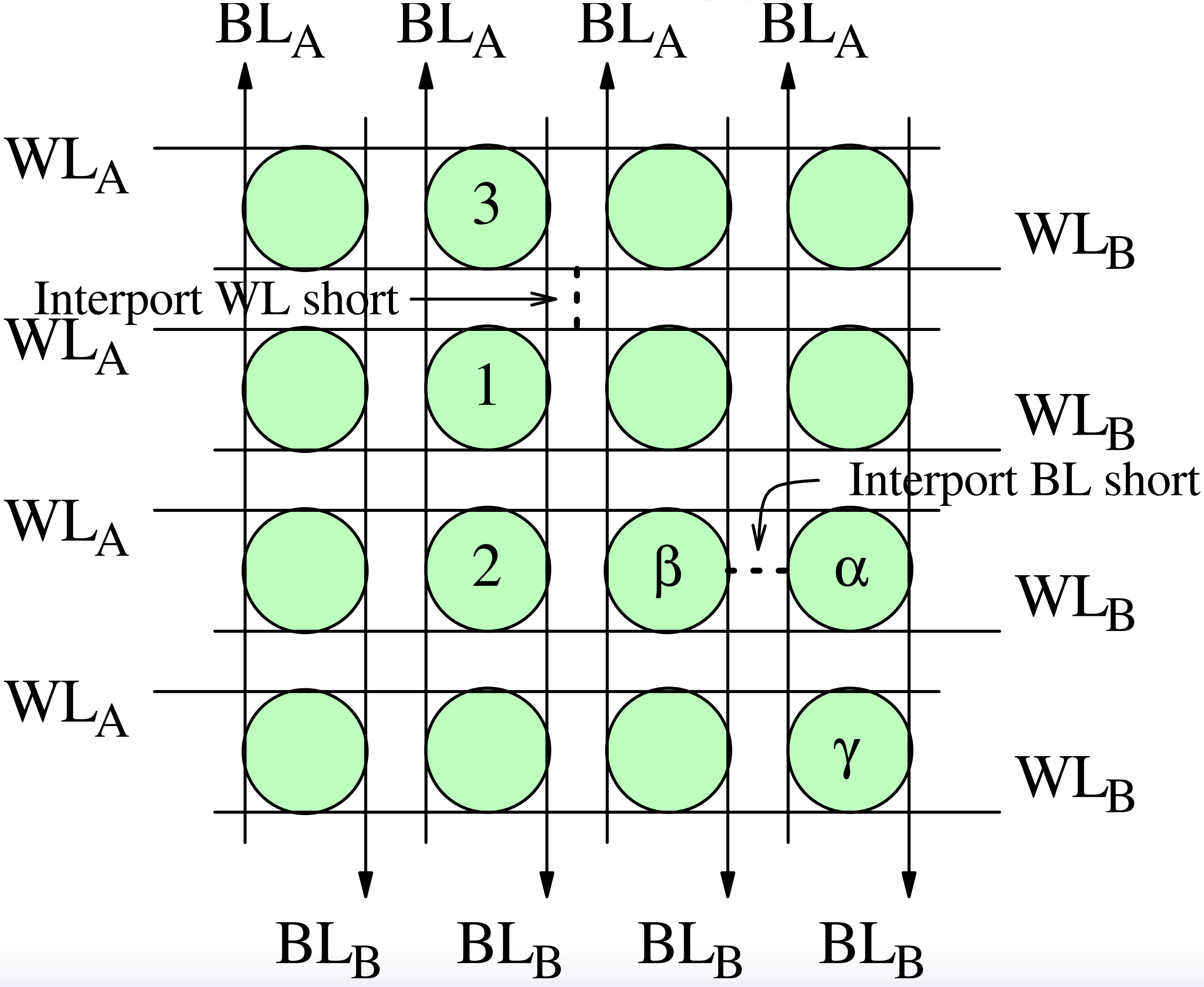
- Results:
  - Complexity is (10+5logm)N, where m is word length and N is word count
  - Test time is reduced by 39% if m=4, as compared with extended March C-
  - Improvement increases as m increases

Ref: Wu et al., IEEE TCAD, 04/02

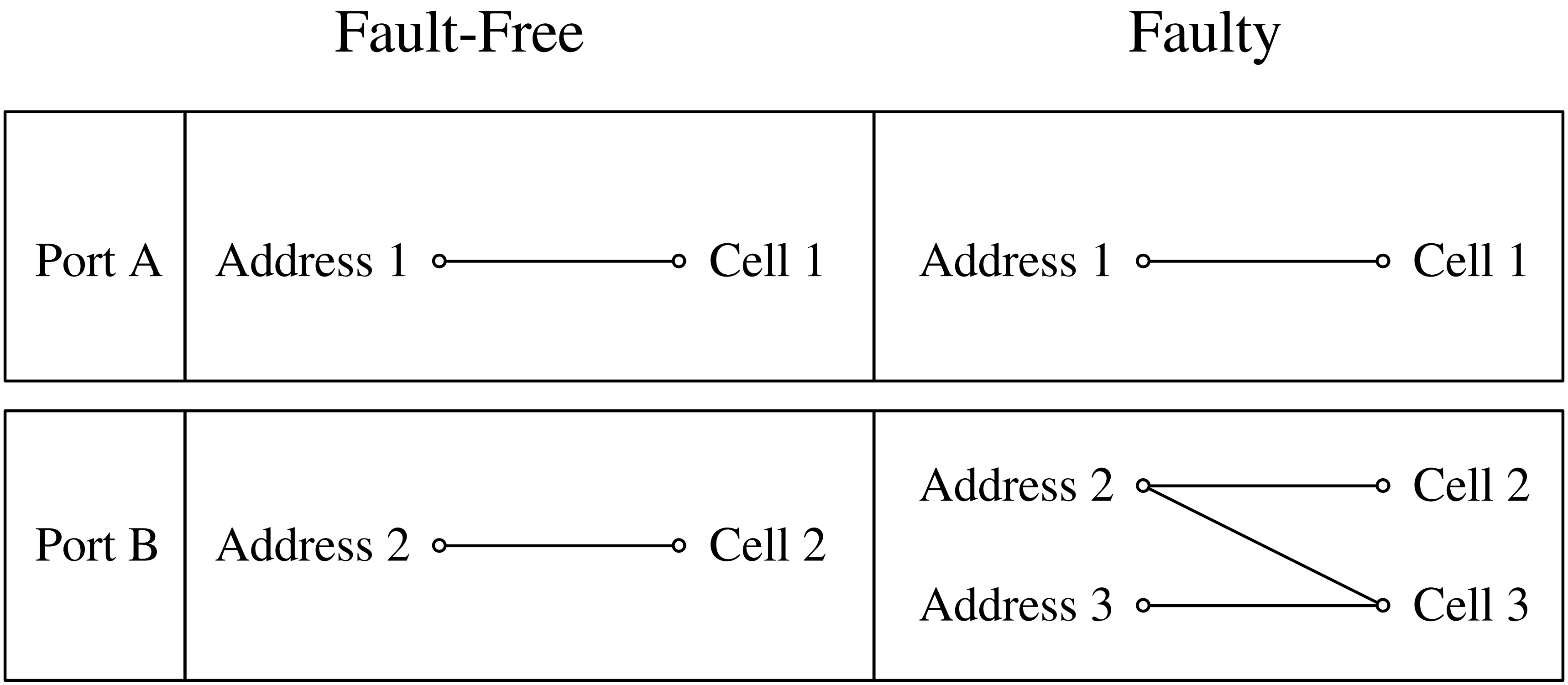
Multi-Port Memory Fault Models

- Cell Faults:
  - Single cell faults: SAF, TF, RDF
  - Two-cell coupling faults
    - Inversion coupling fault (CFin)
    - State coupling fault (CFst)
    - Idempotent coupling fault (CFid)
- Port Faults:
  - Stuck-open fault (SOF)
  - Address decoder fault (AF)
  - Multi-port fault (MPF)

2-Port RAM Topology



Inter-Port Word-Line Short



Fault-Free

Faulty

Port A

Address 1 — Cell 1

Address 1 — Cell 1

Port B

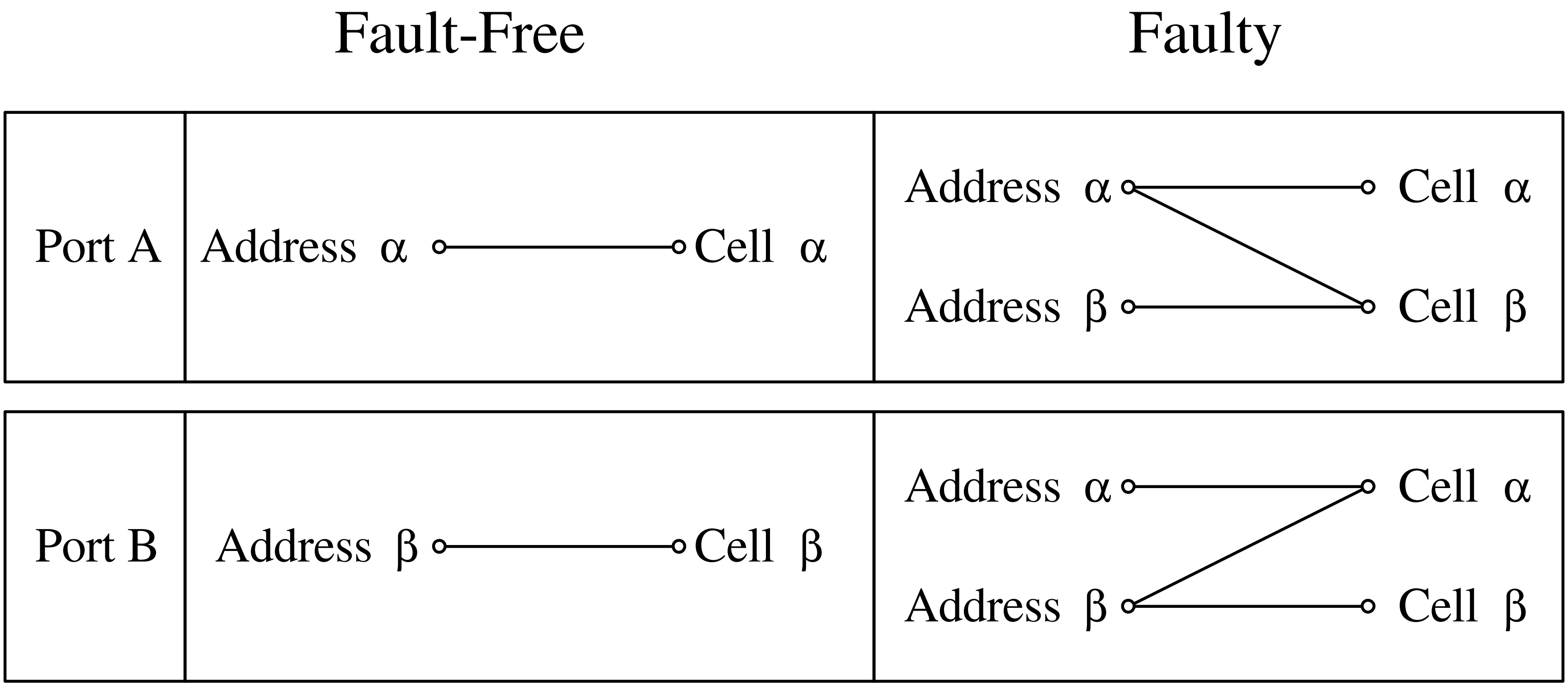
Address 2 — Cell 2

Address 2 — Cell 2

Address 3 — Cell 3

* Functional test complexity: $O(N^3)$

Inter-Port Bit-Line Short



	Fault-Free	Faulty
Port A	Address α Cell α	Address α Cell α Address β Cell β
Port B	Address β Cell β	Address α Cell α Address β Cell β

* Functional test complexity: $O(N^2)$

Why Memory Fault Simulation?

- Fault coverage evaluation can be done efficiently, especially when the number of fault models is large.
- In addition to bit-oriented memories, word-oriented memories can be simulated easily even with multiple backgrounds.
- Test algorithm design and optimization can be done in a much easier way.
- Detection of a test algorithm on unexpected faults can be discovered.
- Fault dictionary can be constructed for easy diagnosis.

Sequential Memory Fault Simulation

- Complexity is N**3 for 2-cell CF

```
For each fault          /* N**2 for 2-cell CF */
    Inject fault;
    For each test element     /* N for March */
        {
            Apply test element;
            Report error output;
        }
```

Parallel Fault Simulation

- ❑ RAMSES [Wu, Huang, & Wu, DFT99 & IEEE TCAD 4/02]
  - Each fault model has a fault descriptor

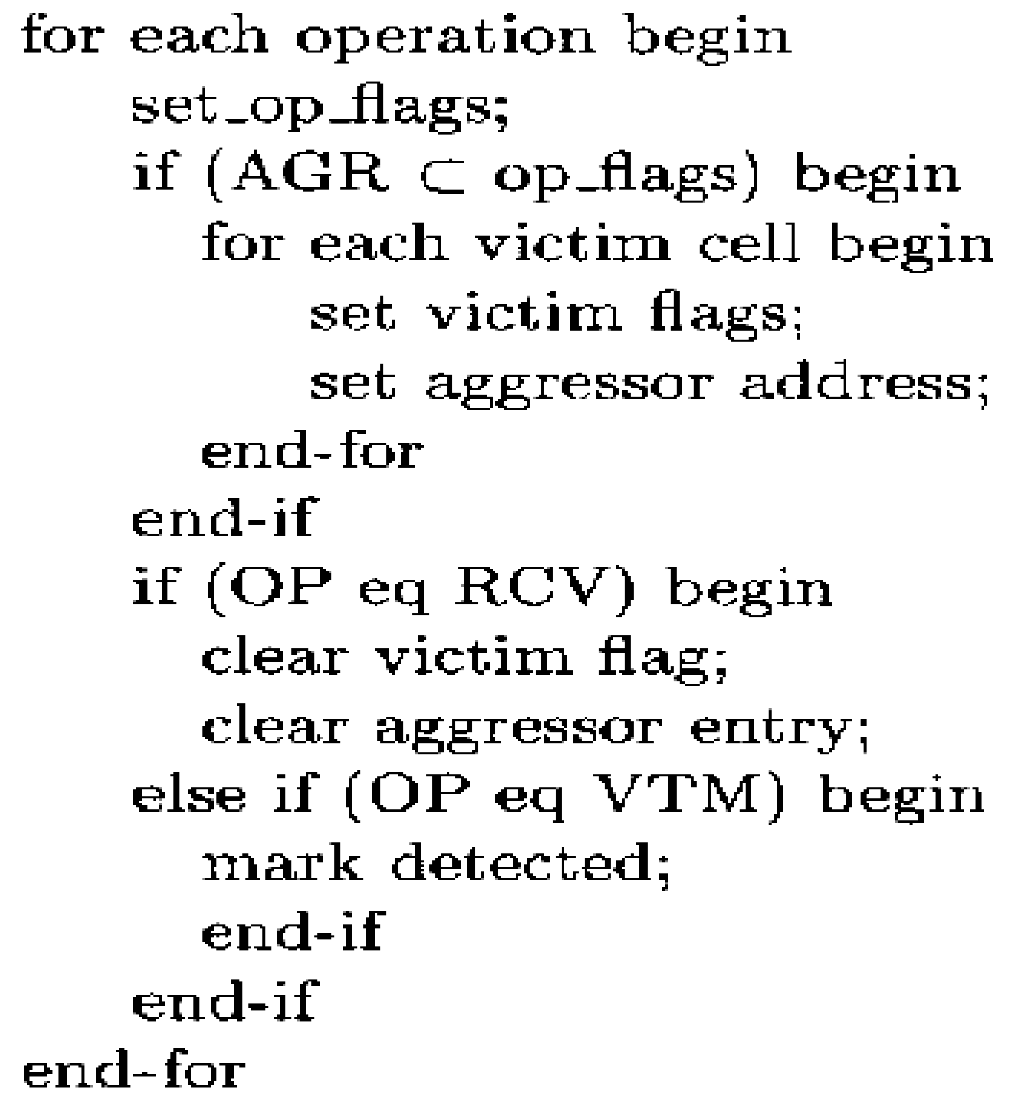
```
# S/1
AGR := w0
SPT := @          /* Single-cell fault */
VTM := r0
RCV := w1

# CFst <0;s/1>
AGR := v0
SPT := *          /* All other cells are suspects */
VTM := r0
RCV := w1
```

RAMSES

- Complexity is N**2

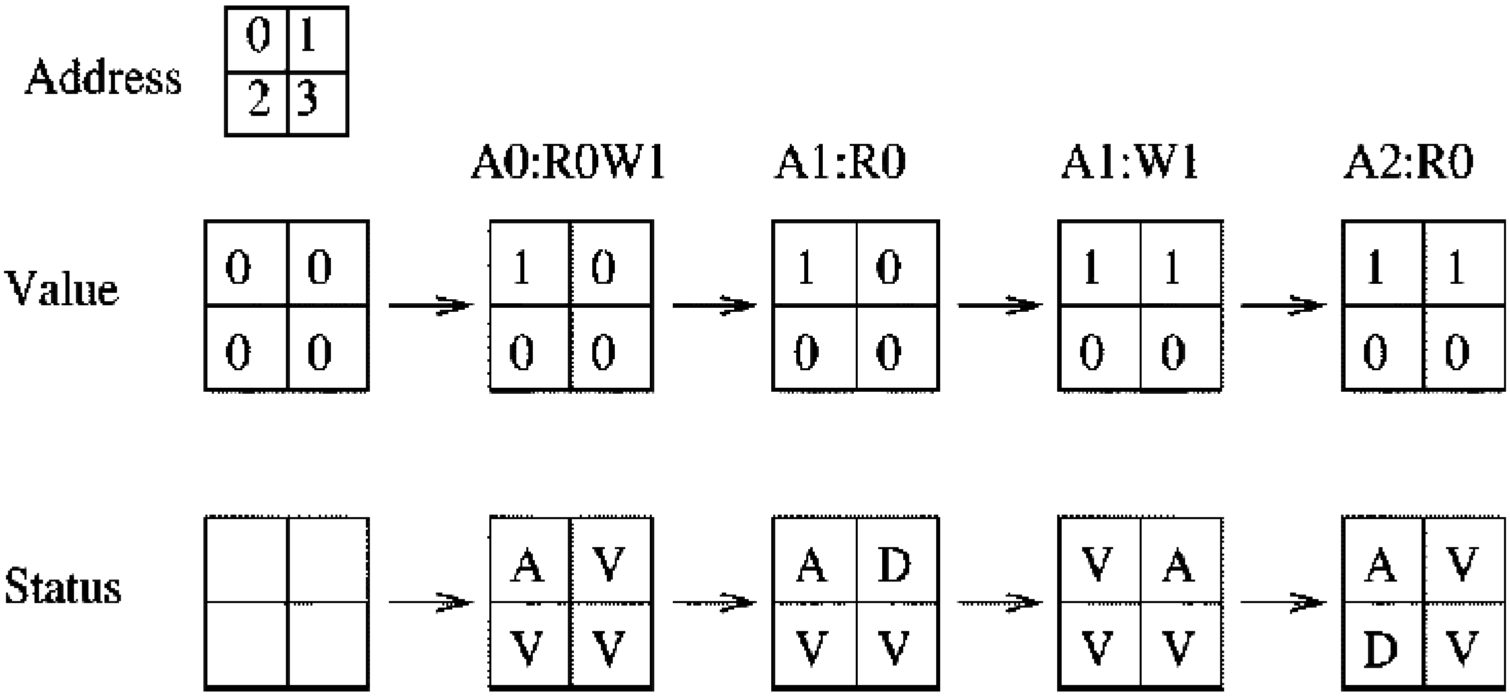
```
For each test operation
   {
     If op is AGR then mark victim cells;
     If op is RCV then release victim cells;
     If op is VTM then report error;
   }
```

RAMSES Algorithm



```
for each operation begin
    set_op_flags;
    if (AGR ⊂ op_flags) begin
      for each victim cell begin
          set victim flags;
          set aggressor address;
      end-for
    end-if
    if (OP eq RCV) begin
      clear victim flag;
      clear aggressor entry;
    else if (OP eq VTM) begin
      mark detected;
      end-if
    end-if
end-for
```

RAMSES Example for CFin< ↑; ↕>



Address	
0	1
2	3

		A0:R0W1	A1:R0	A1:W1	A2:R0
Value	0 0 0 0	1 0 0 0	1 0 0 0	1 1 0 0	1 1 0 0
Status		A V V V	A D V V	V A V V	A V D V

Coverage of March Tests

	MATS++	March X	March Y	March C-
SAF	1	1	1	1
TF	1	1	1	1
AF	1	1	1	1
SOF	1	.002	1	.002
CFin	.75	1	1	1
CFid	.375	.5	.5	1
CFst	.5	.625	.625	1

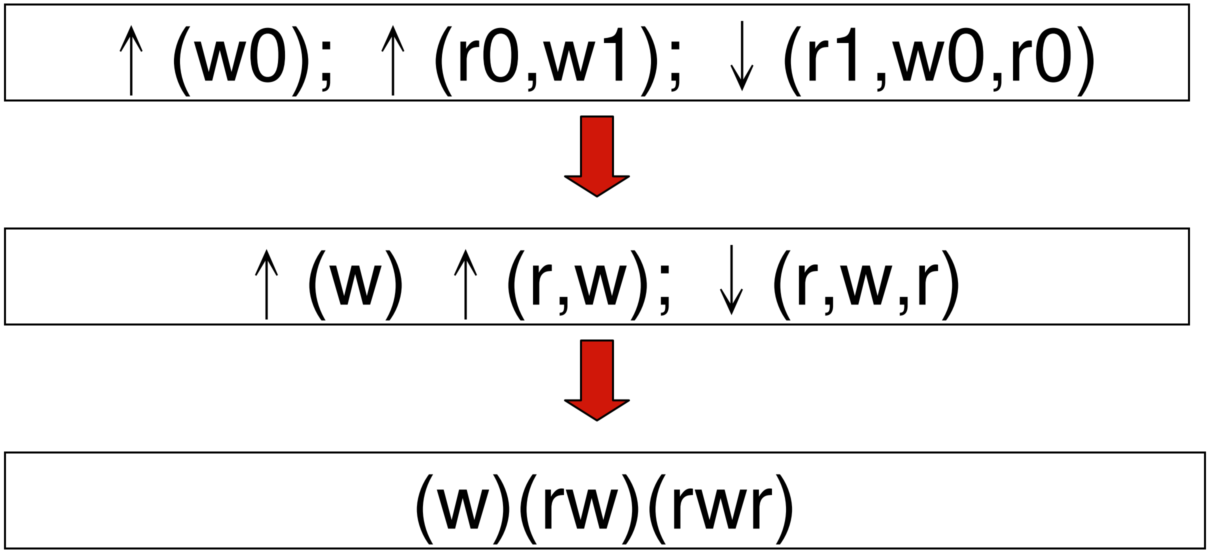
- ☞ Extended March C- has 100% coverage of SOF

Test Algorithm Generation Goals

- Given a set of target fault models, generate a test with 100% fault coverage
- Given a set of target fault models and a test length constraint, generate a test with the highest fault coverage
- Priority setting for fault models
  - Test length/test time can be reduced
- Diagnostic test generation
  - Need longer test to distinguish faults

Test Algorithm Generation by Simulation (TAGS)

- March template abstraction:



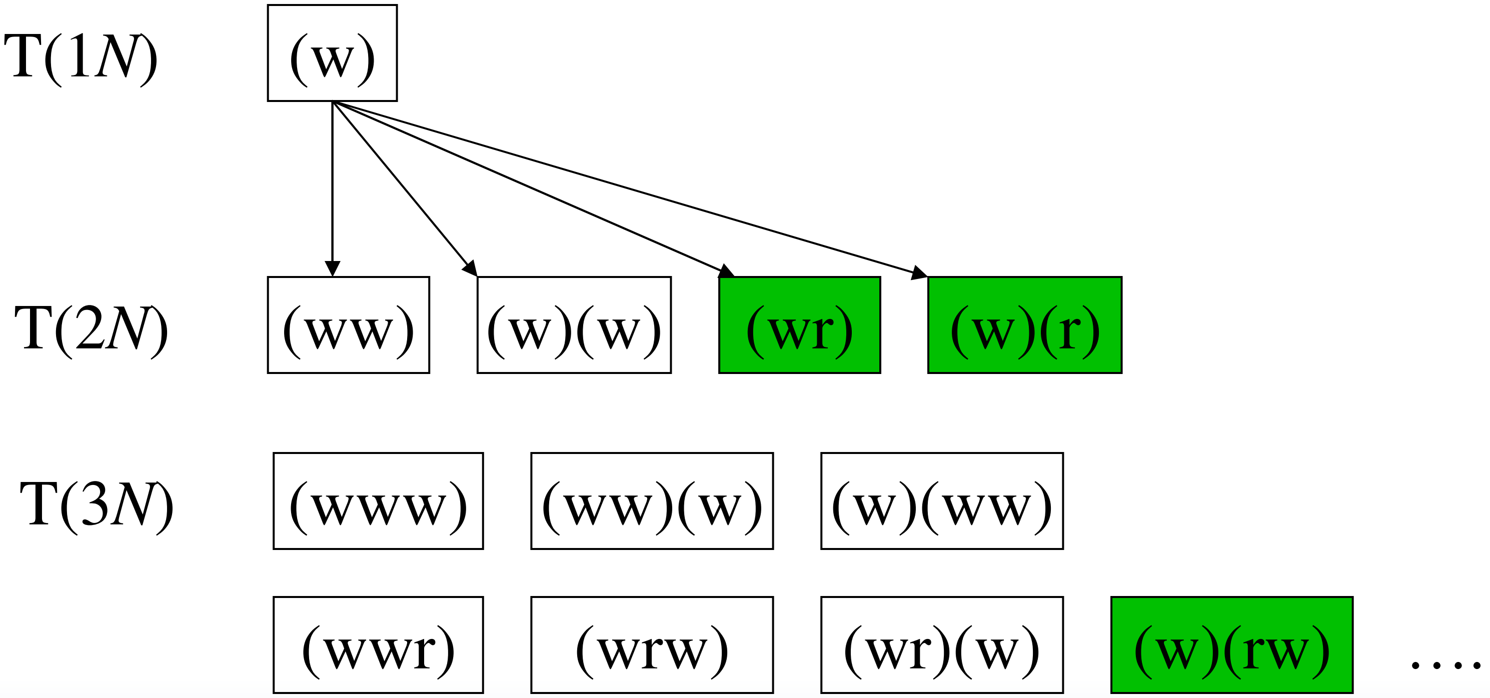
$\uparrow(w0);\ \uparrow(r0,w1);\ \downarrow(r1,w0,r0)$

$\uparrow(w)\ \ \uparrow(r,w);\ \downarrow(r,w,r)$

(w)(rw)(rwr)

Template Set

- Exhaustive generation: complexity is very high, e.g., 6.7 million templates when $N = 9$
- Heuristics should be developed to select useful templates



TAGS Procedure

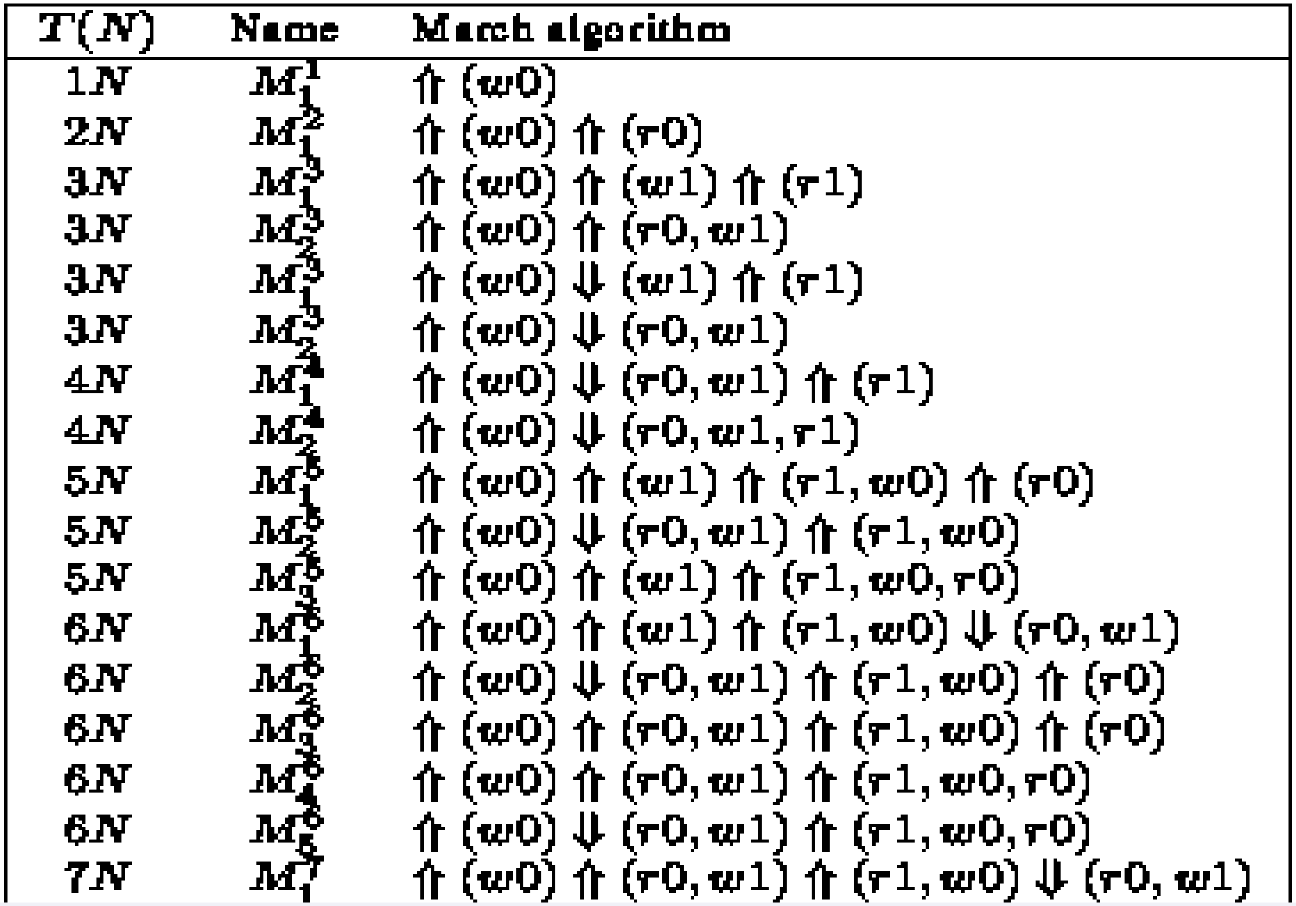
1. Initialize test length as 1N, T(1N) = {(w)};
2. Increase test length by 1N: apply generation options;
3. Apply filter options;
4. Assign address orders and data backgrounds;
5. Fault simulation using RAMSES;
6. Drop ineffective tests;
7. Repeat 2-6 using the new template set until constraints met;

Template Generation/Filtering

- Generation heuristics:
  - (r) insertion
  - (…r), (r…) expansion
  - (w) insertion
  - (…w), (w…) expansion
- Filtering heuristics:
  - Consecutive read: (…rr…)
  - Repeated read: (r)(r)
  - Tailing single write: …(w)

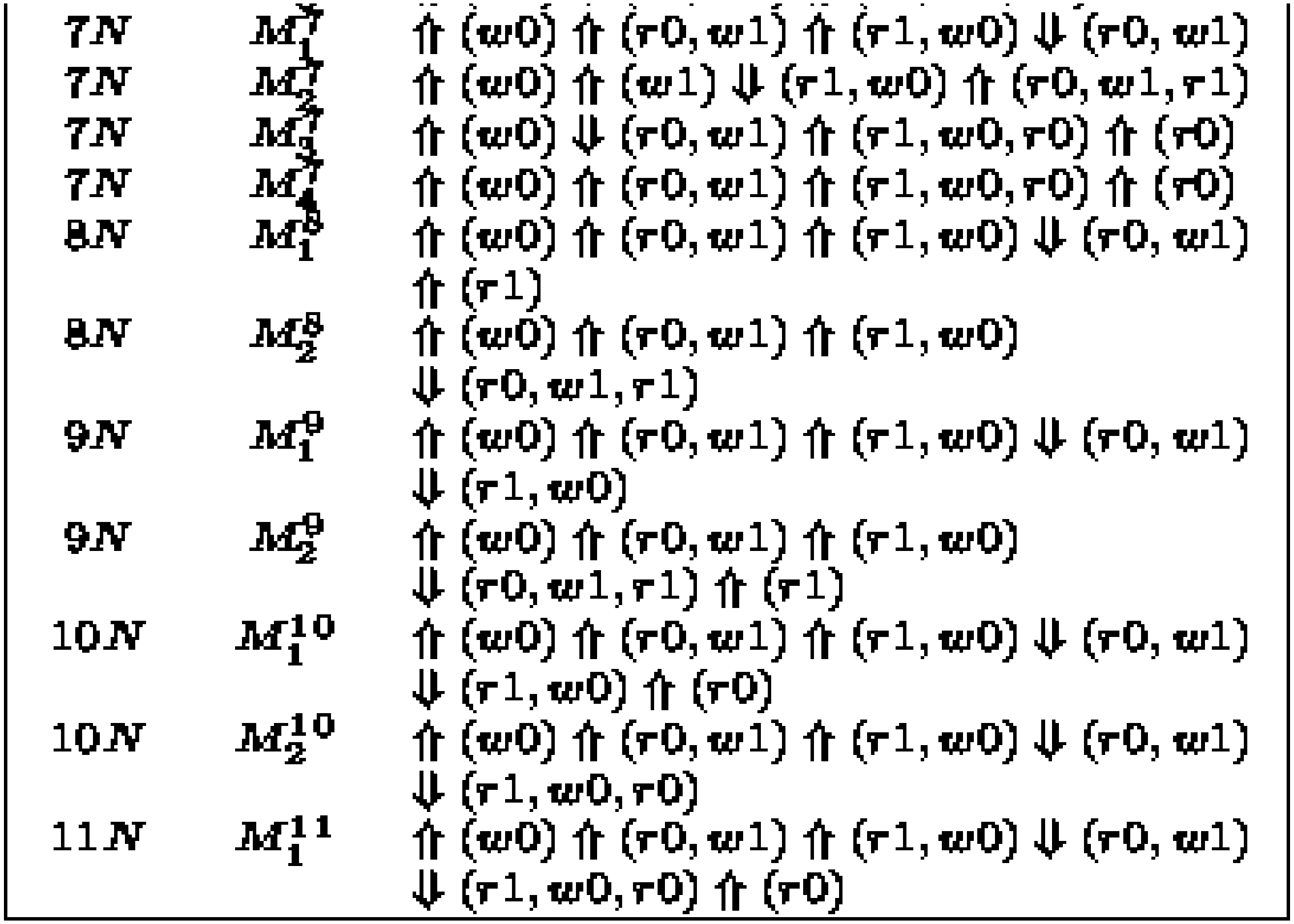
TAGS Example (1/2)

- Target fault models (SAF, TF, AF, SOF, Cfin, Cfid, CFst), time constraints ∞:



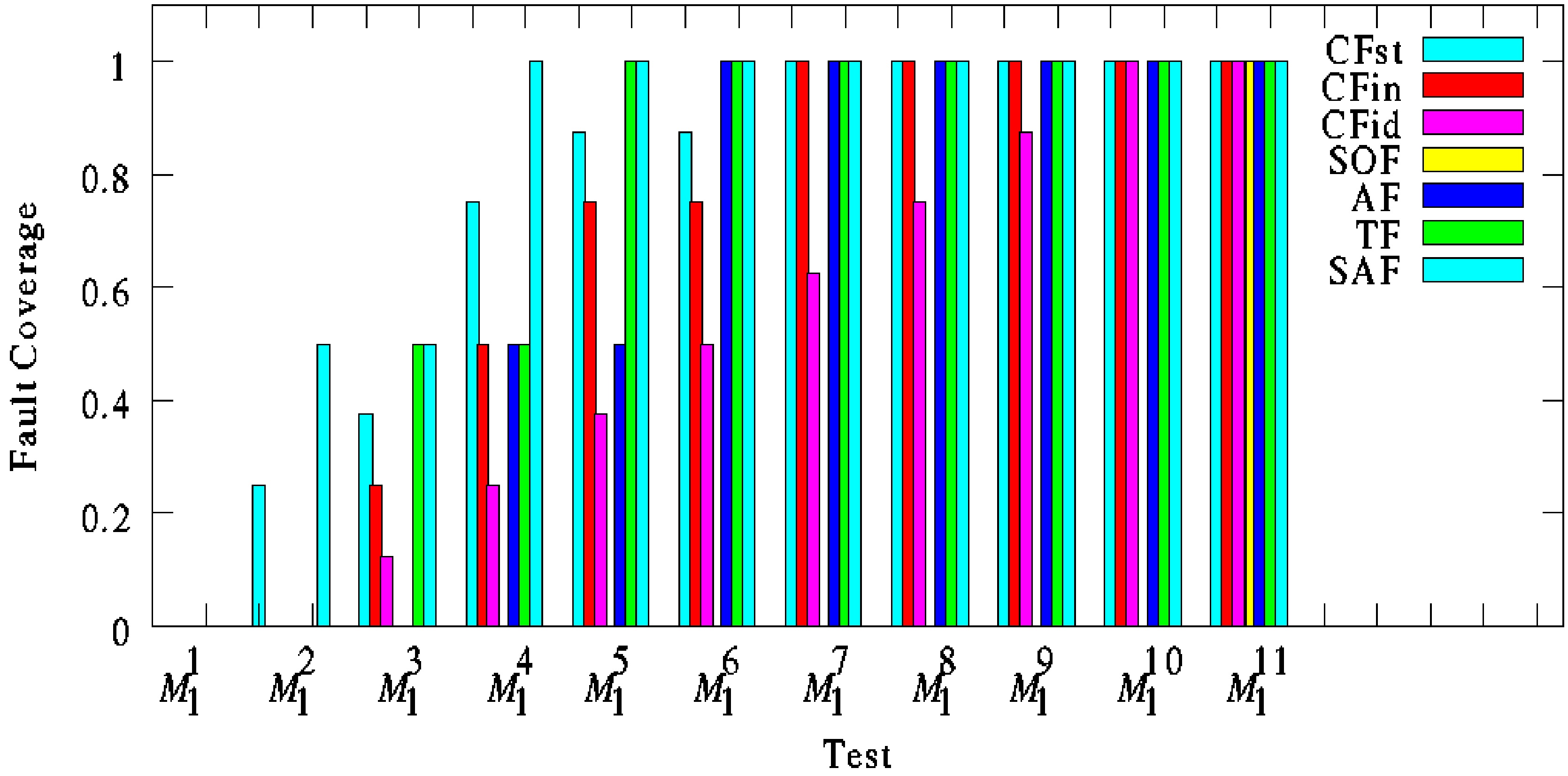
$T(N)$	Name	March algorithm
$1N$	M_1^1	$\Uparrow(w0)$
$2N$	M_1^2	$\Uparrow(w0)\ \Uparrow(r0)$
$3N$	M_1^3	$\Uparrow(w0)\ \Uparrow(w1)\ \Uparrow(r1)$
$3N$	M_2^3	$\Uparrow(w0)\ \Uparrow(r0,w1)$
$3N$	M_1^3	$\Uparrow(w0)\ \Downarrow(w1)\ \Uparrow(r1)$
$3N$	M_2^3	$\Uparrow(w0)\ \Downarrow(r0,w1)$
$4N$	M_1^4	$\Uparrow(w0)\ \Downarrow(r0,w1)\ \Uparrow(r1)$
$4N$	M_2^4	$\Uparrow(w0)\ \Downarrow(r0,w1,r1)$
$5N$	M_1^5	$\Uparrow(w0)\ \Uparrow(w1)\ \Uparrow(r1,w0)\ \Uparrow(r0)$
$5N$	M_2^5	$\Uparrow(w0)\ \Downarrow(r0,w1)\ \Uparrow(r1,w0)$
$5N$	M_3^5	$\Uparrow(w0)\ \Uparrow(w1)\ \Uparrow(r1,w0,r0)$
$6N$	M_1^6	$\Uparrow(w0)\ \Uparrow(w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$
$6N$	M_2^6	$\Uparrow(w0)\ \Downarrow(r0,w1)\ \Uparrow(r1,w0)\ \Uparrow(r0)$
$6N$	M_3^6	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Uparrow(r0)$
$6N$	M_4^6	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0,r0)$
$6N$	M_5^6	$\Uparrow(w0)\ \Downarrow(r0,w1)\ \Uparrow(r1,w0,r0)$
$7N$	M_1^7	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$

TAGS Example (2/2)

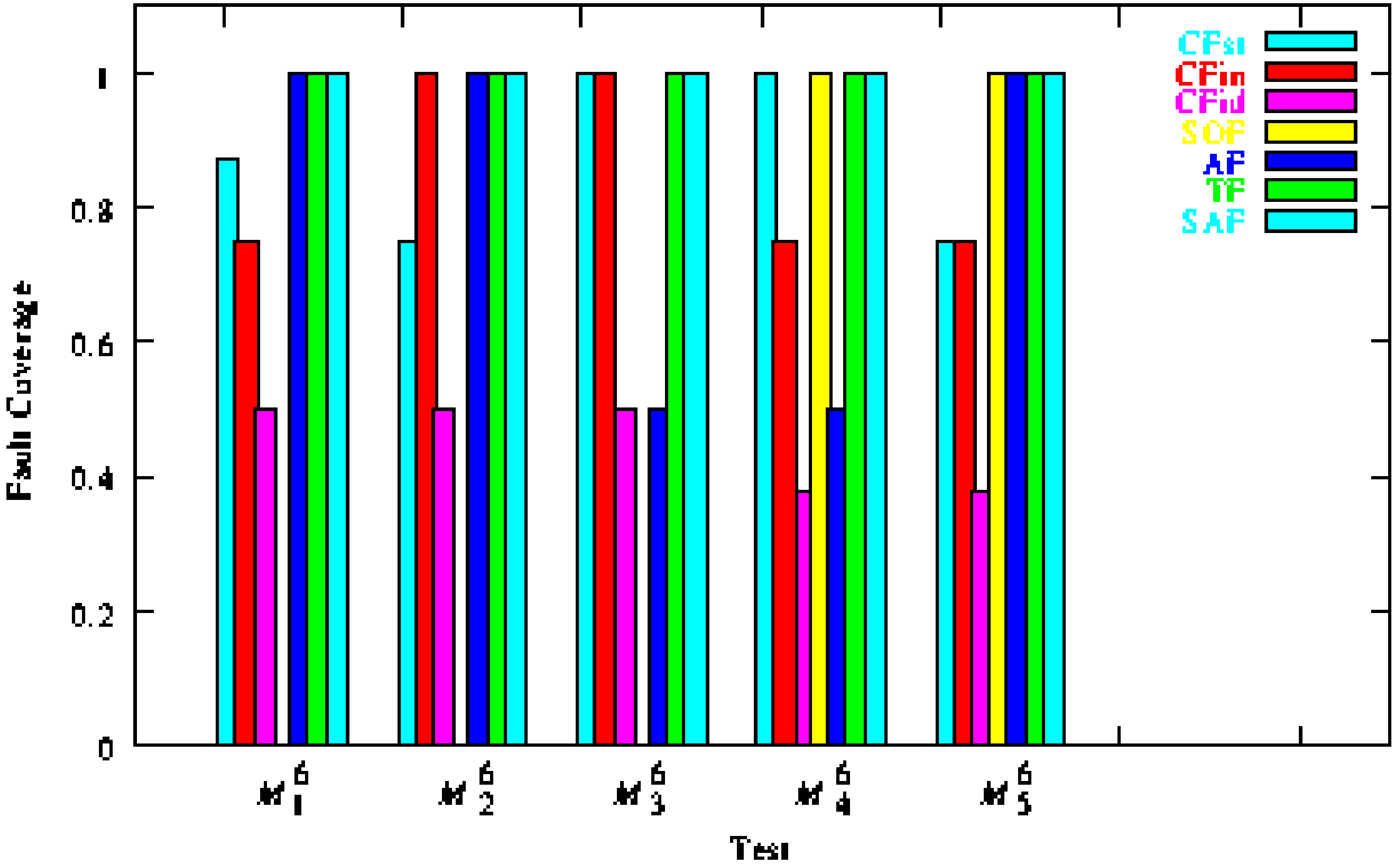


$7N$	M_1^7	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$
$7N$	M_2^7	$\Uparrow(w0)\ \Uparrow(w1)\ \Downarrow(r1,w0)\ \Uparrow(r0,w1,r1)$
$7N$	M_3^7	$\Uparrow(w0)\ \Downarrow(r0,w1)\ \Uparrow(r1,w0,r0)\ \Uparrow(r0)$
$7N$	M_4^7	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0,r0)\ \Uparrow(r0)$
$8N$	M_1^8	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$ $\Uparrow(r1)$
$8N$	M_2^8	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)$ $\Downarrow(r0,w1,r1)$
$9N$	M_1^9	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$ $\Downarrow(r1,w0)$
$9N$	M_2^9	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)$ $\Downarrow(r0,w1,r1)\ \Uparrow(r1)$
$10N$	M_1^{10}	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$ $\Downarrow(r1,w0)\ \Uparrow(r0)$
$10N$	M_2^{10}	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$ $\Downarrow(r1,w0,r0)$
$11N$	M_1^{11}	$\Uparrow(w0)\ \Uparrow(r0,w1)\ \Uparrow(r1,w0)\ \Downarrow(r0,w1)$ $\Downarrow(r1,w0,r0)\ \Uparrow(r0)$

RAMSES Simulation Results

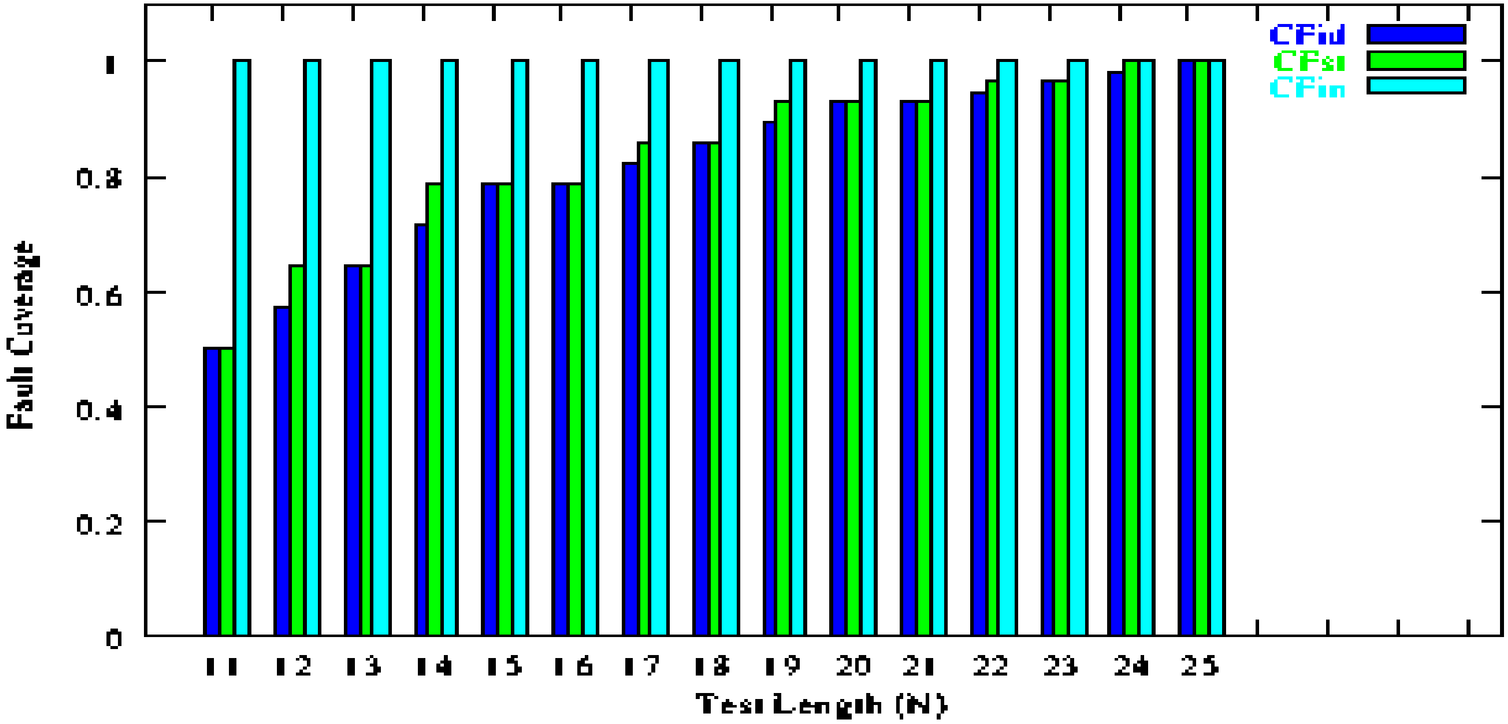


FC Spectrum for 6N Tests



Word-Oriented TAGS

1. Construct bit-oriented test algorithms
2. Generate initial Cocktail-March: Assign each data background to the test in Step 1—a cascade of multiple March algorithms
3. Optimize the Cocktail-March (!P_1) /* non-solid backgrounds */
4. Optimize the Cocktail-March (P_1) /* solid background */



3. Cocktail March Optimization (!P_1)

For each non-solid data background P (P != P_1)

a) Generate a new Cocktail–March test by replacing the March algorithm having P as its background with a shorter one from the set of algorithms generated in Step 1.

b) Run RAMSES for the new Cocktail–March.

c) Repeat 3(a) and 3(b) until the FC drops and cannot be recovered by any other test algorithm of the same length.

d) Store the test algorithm candidates used in the previous step.

4. Cocktail March Optimization (P_1)

a) Generate a new Cocktail–March test by replacing the March algorithm having P_1 as its background with a shorter one from the test set generated in Step 1. Repeat with every test candidate for other backgrounds.

b) Run RAMSES for the new Cocktail–March.

c) Repeat 4(a) and 4(b) for all candidate test algorithms from 3(d) until the FC drops and cannot be recovered by any other test algorithm of the same length or by selecting other candidates.

Cocktail March Example (m=8)

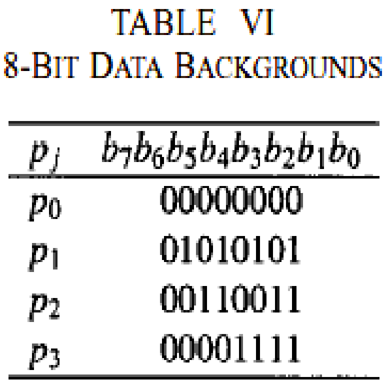


TABLE VI
8-BIT DATA BACKGROUNDS

p_j	$b_7b_6b_5b_4b_3b_2b_1b_0$
p_0	00000000
p_1	01010101
p_2	00110011
p_3	00001111

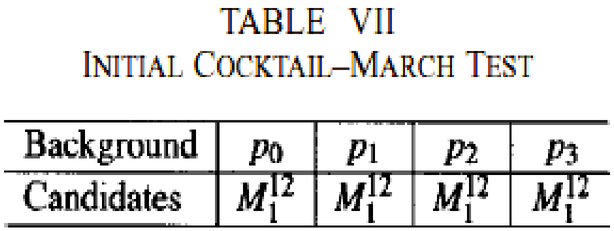


TABLE VII
INITIAL COCKTAIL–MARCH TEST

Background	p_0	p_1	p_2	p_3
Candidates	M_1^{12}	M_1^{12}	M_1^{12}	M_1^{12}

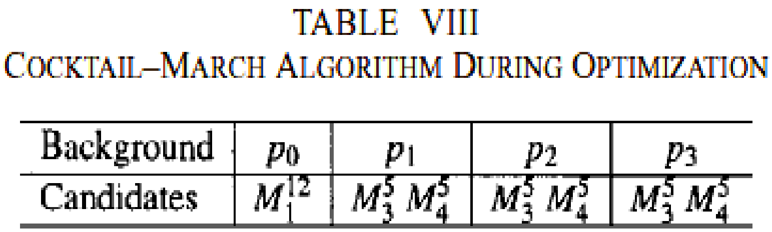


TABLE VIII
COCKTAIL–MARCH ALGORITHM DURING OPTIMIZATION

Background	p_0	p_1	p_2	p_3
Candidates	M_1^{12}	M_3^5 M_4^5	M_3^5 M_4^5	M_3^5 M_4^5

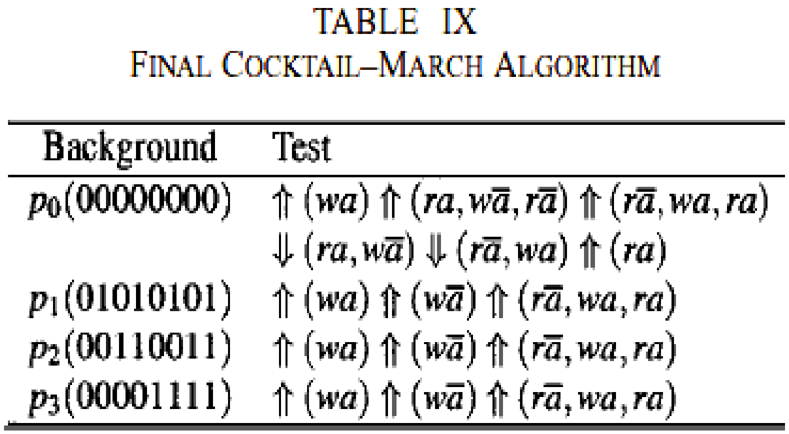


TABLE IX
FINAL COCKTAIL–MARCH ALGORITHM

Background	Test
$p_0(00000000)$	$\Uparrow(wa)\Uparrow(ra,w\bar{a},r\bar{a})\Uparrow(r\bar{a},wa,ra)$ $\Downarrow(ra,w\bar{a})\Downarrow(r\bar{a},wa)\Uparrow(ra)$
$p_1(01010101)$	$\Uparrow(wa)\Uparrow(w\bar{a})\Uparrow(r\bar{a},wa,ra)$
$p_2(00110011)$	$\Uparrow(wa)\Uparrow(w\bar{a})\Uparrow(r\bar{a},wa,ra)$
$p_3(00001111)$	$\Uparrow(wa)\Uparrow(w\bar{a})\Uparrow(r\bar{a},wa,ra)$

Ref: Wu *et al.*, IEEE TCAD, 4/02

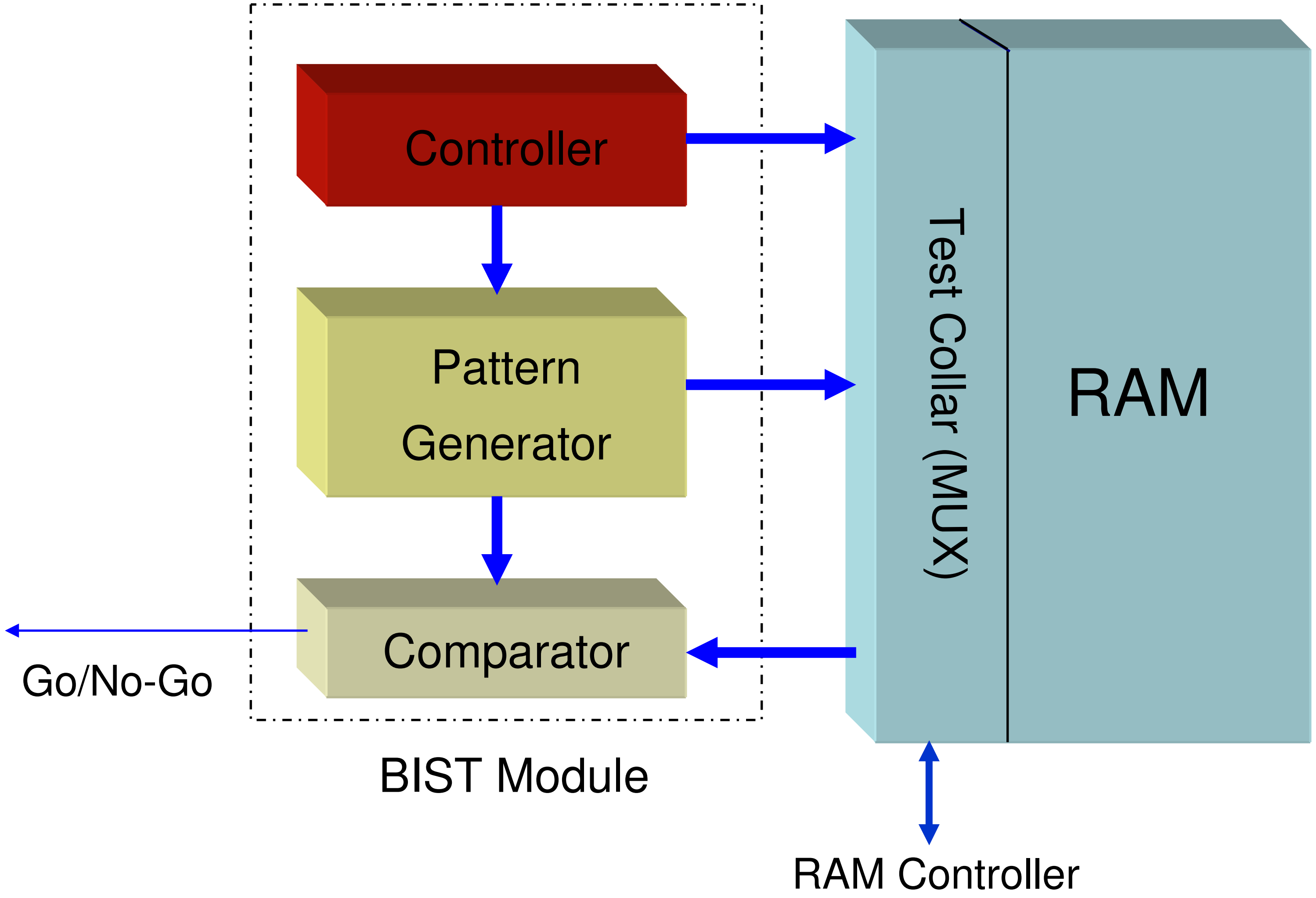
What Can BIST do?

- What are the functional faults to be covered?
  - Static and dynamic
  - Operation modes
- What are the defects to be covered?
  - Opens, shorts, timing parameters, voltages, currents, etc.
- Can it support fault location and redundancy repair?
- Can it support BI?
- Can it support on-chip redundancy analysis and repair?
- Does it allow characterization test as well as mass production test?
- Can it really replace ATE (and laser repair machine)?
  - Programmability, speed, timing accuracy, threshold range, parallelism, etc.

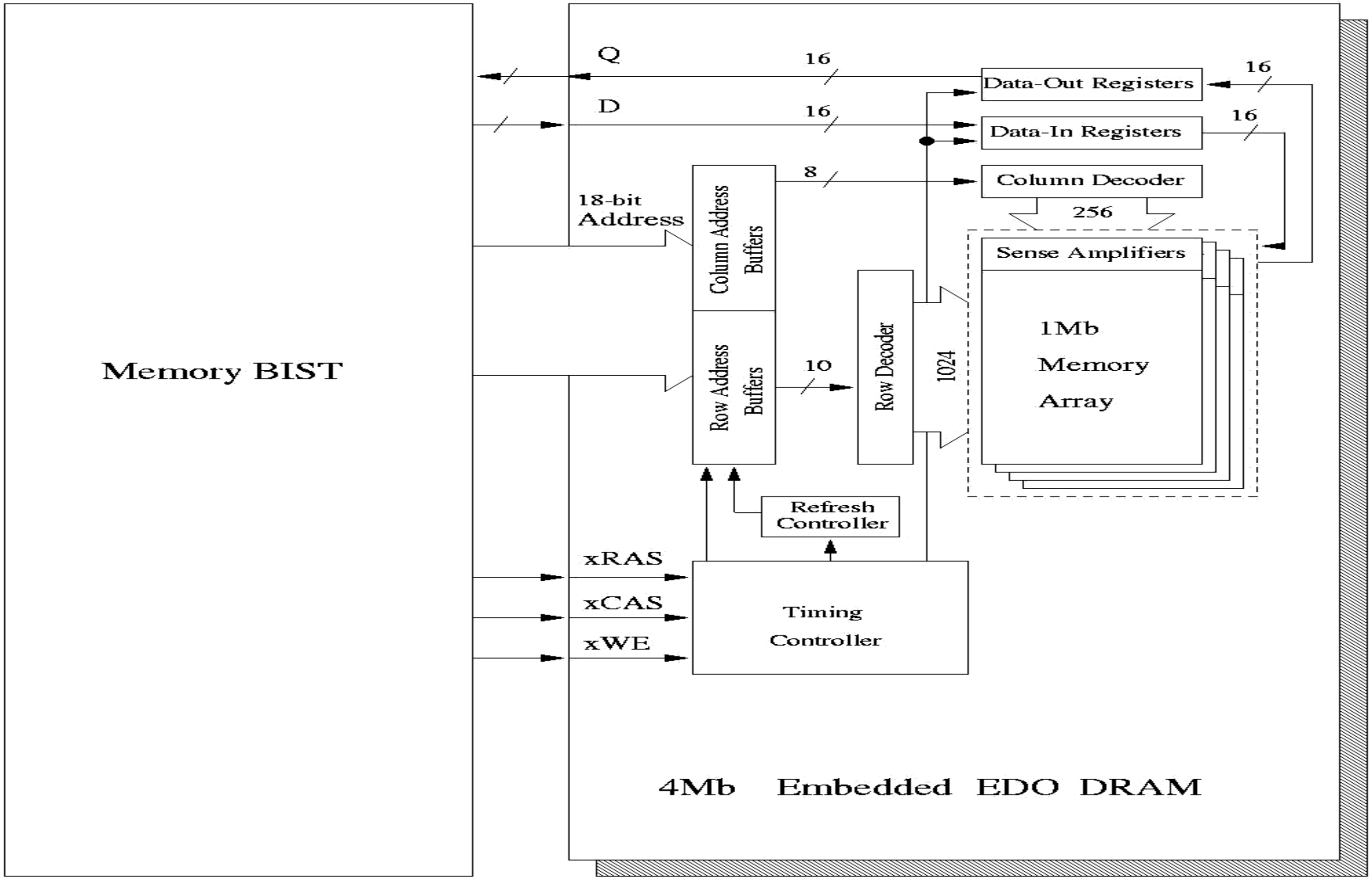
Typical RAM BIST Approaches

- Methodology
  - Processor-based BIST
    - Programmable
  - Hardwired BIST
    - Fast
    - Compact
  - Hybrid
- Interface
  - Serial (scan, 1149.1)
  - Parallel (embedded controller; hierarchical)
- Patterns (address sequence)
  - March & March-like
  - Pseudorandom
  - Others

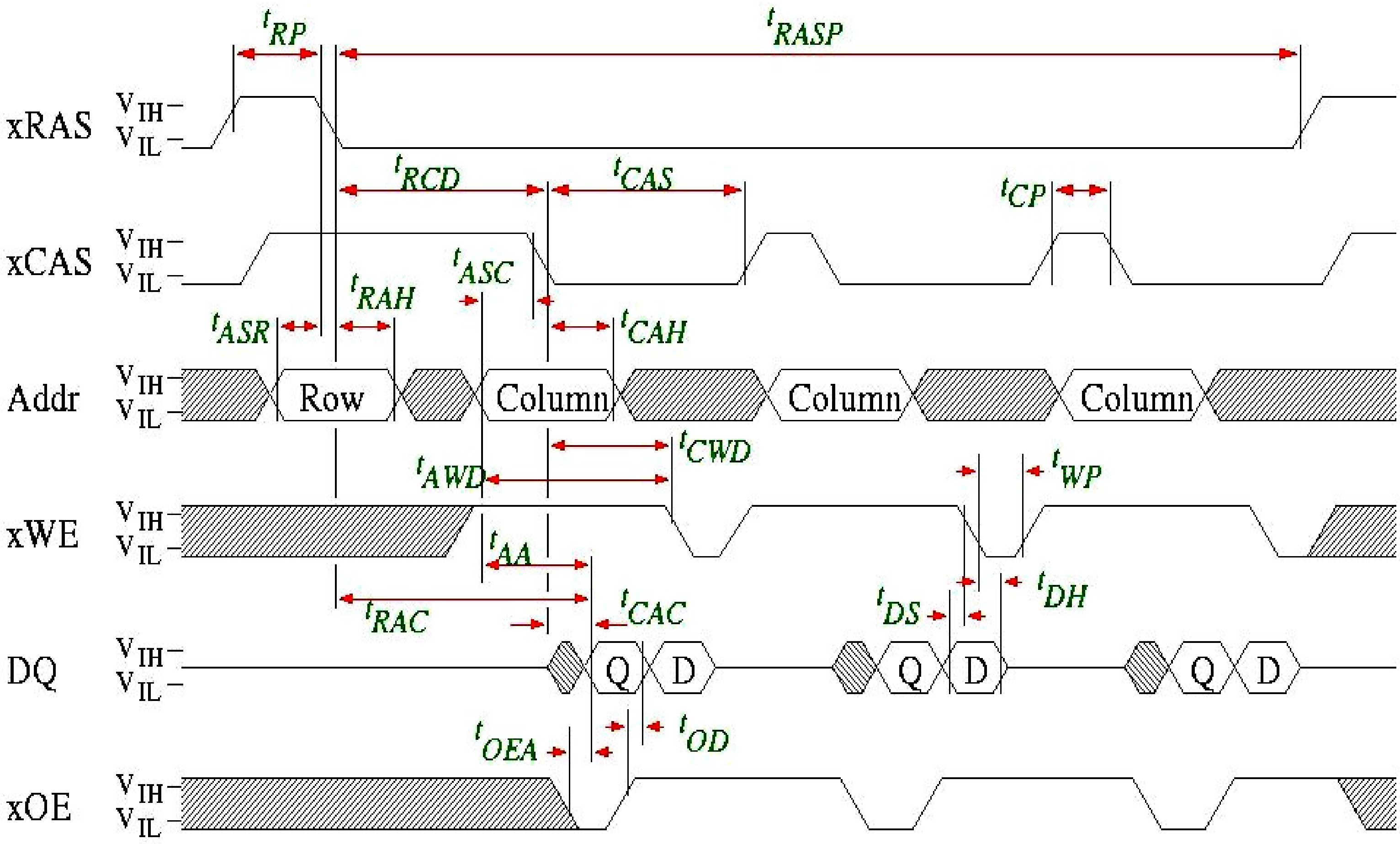
Typical RAM BIST Architecture



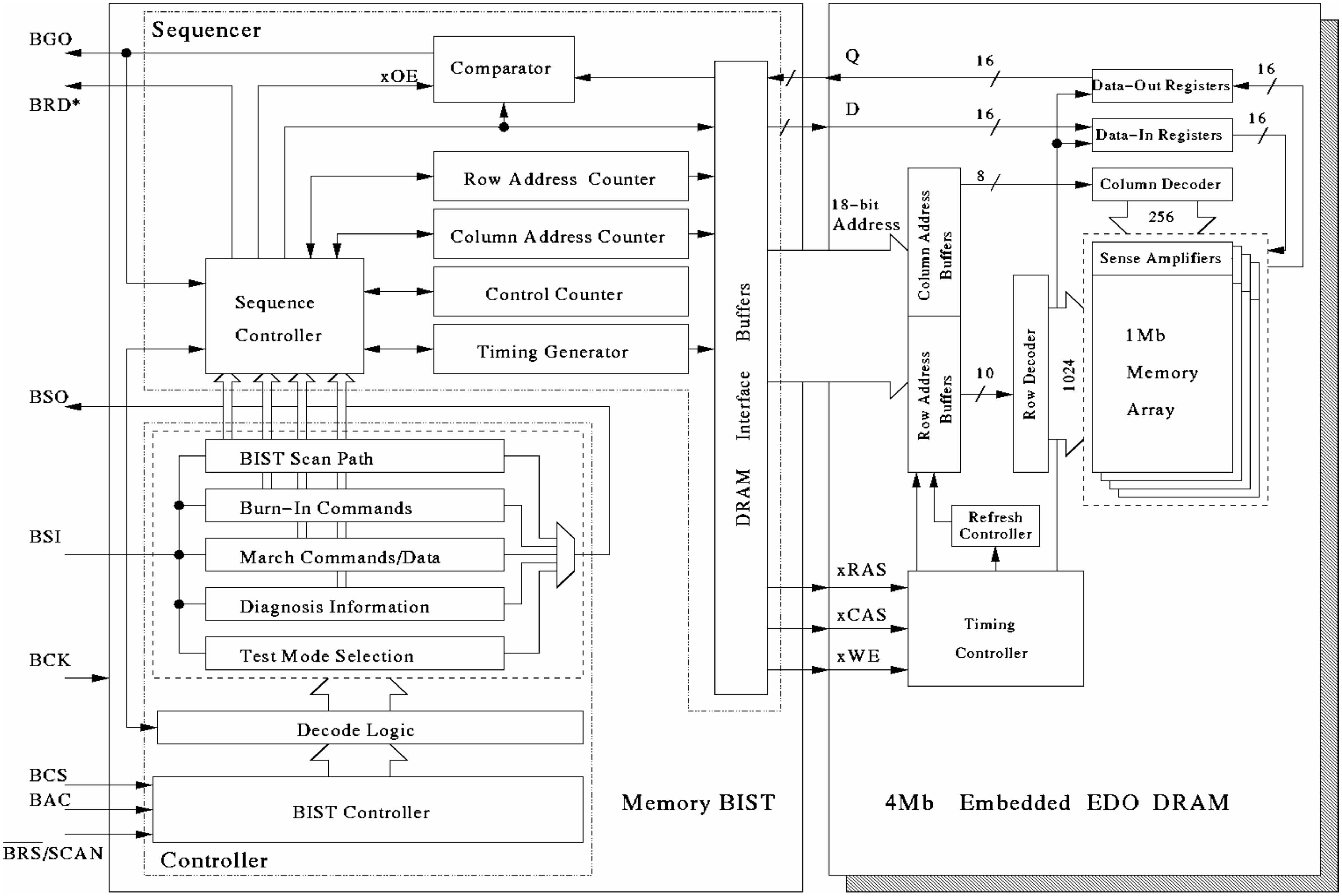
EDO DRAM BIST Example



DRAM Page-Mode Read-Write Cycle



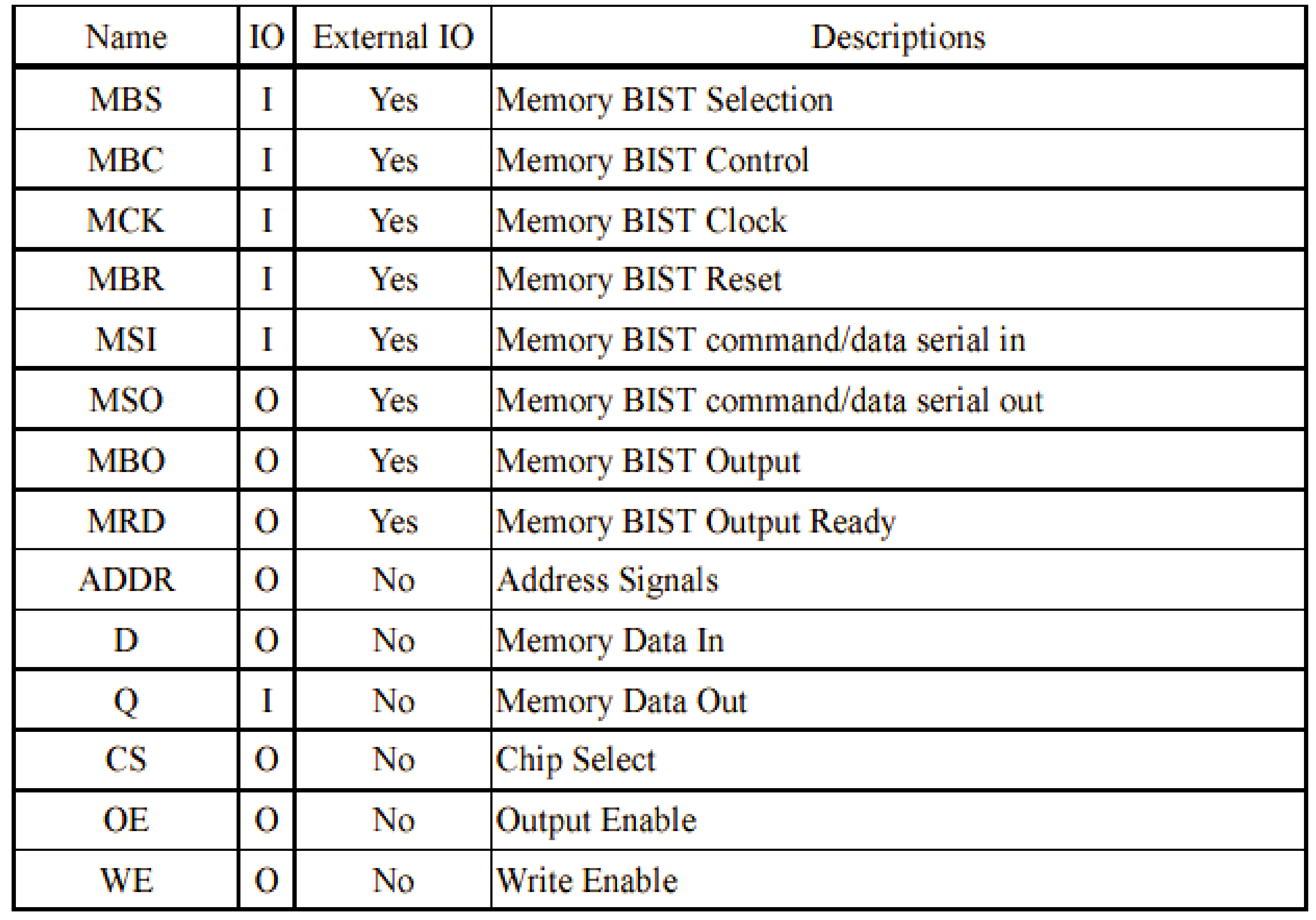
BIST Architecture



BIST External I/O

- MBS (Memory BIST Selection): controller test collar (normal/test mode selection)
- MBC (Memory BIST Control): Controller input
- MCK (Memory BIST Clock)
- MBR (Memory BIST Reset)
- MSI (Memory BIST Scan In): for test commands and scan test inputs
- MSO (Memory BIST Scan Out): for diagnostic data and scan test outputs
- MBO (Memory BIST Output): error indicator
- MRD (Memory BIST Output Ready): BIST completion flag

BIST I/O Summary

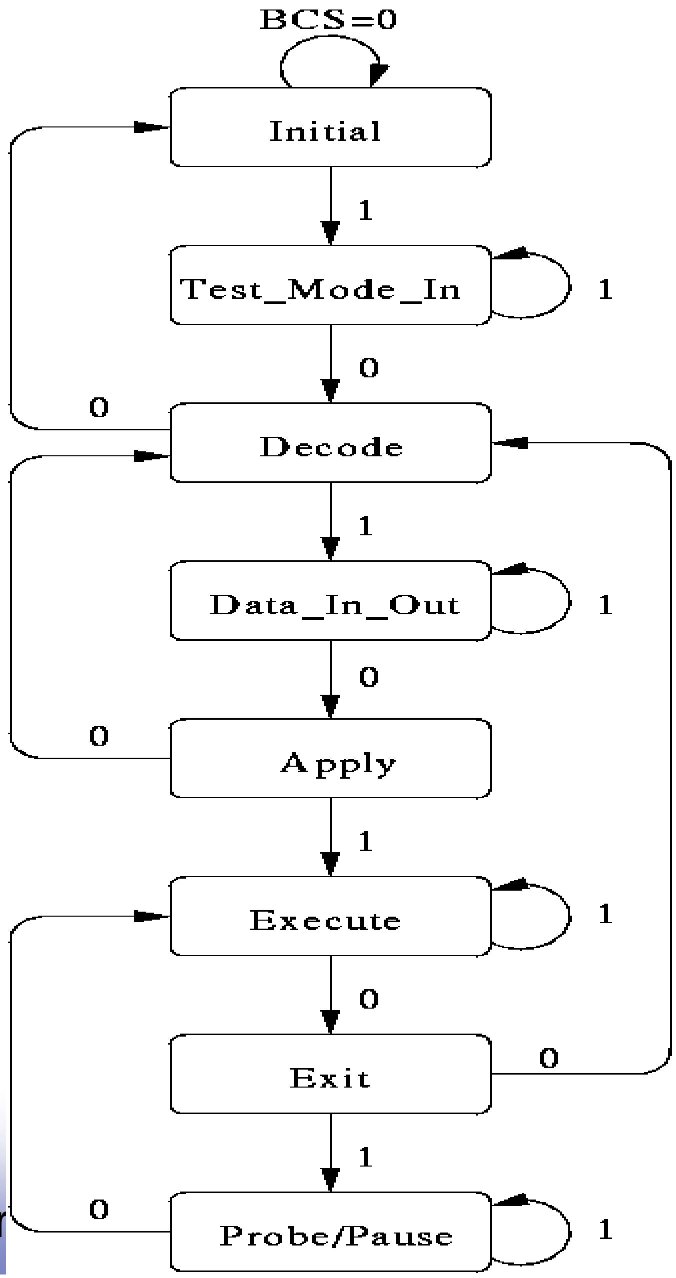


Name	IO	External IO	Descriptions
MBS	I	Yes	Memory BIST Selection
MBC	I	Yes	Memory BIST Control
MCK	I	Yes	Memory BIST Clock
MBR	I	Yes	Memory BIST Reset
MSI	I	Yes	Memory BIST command/data serial in
MSO	O	Yes	Memory BIST command/data serial out
MBO	O	Yes	Memory BIST Output
MRD	O	Yes	Memory BIST Output Ready
ADDR	O	No	Address Signals
D	O	No	Memory Data In
Q	I	No	Memory Data Out
CS	O	No	Chip Select
OE	O	No	Output Enable
WE	O	No	Write Enable

Controller and Sequencer

- ❑ Controller
    - Microprogram
    - Hardwired
    - Shared CPU core
    - IEEE 1149.1 TAP
    - PLD
- ❑ Sequencer (Pattern Generator)
    - Counter
    - LFSR
    - LUT
    - PLD

Controller



Initial/reset state: all BIST outputs retain safe values.

Test mode selection.

Command decoding.

Data scan: shift in test inputs and shift out results.

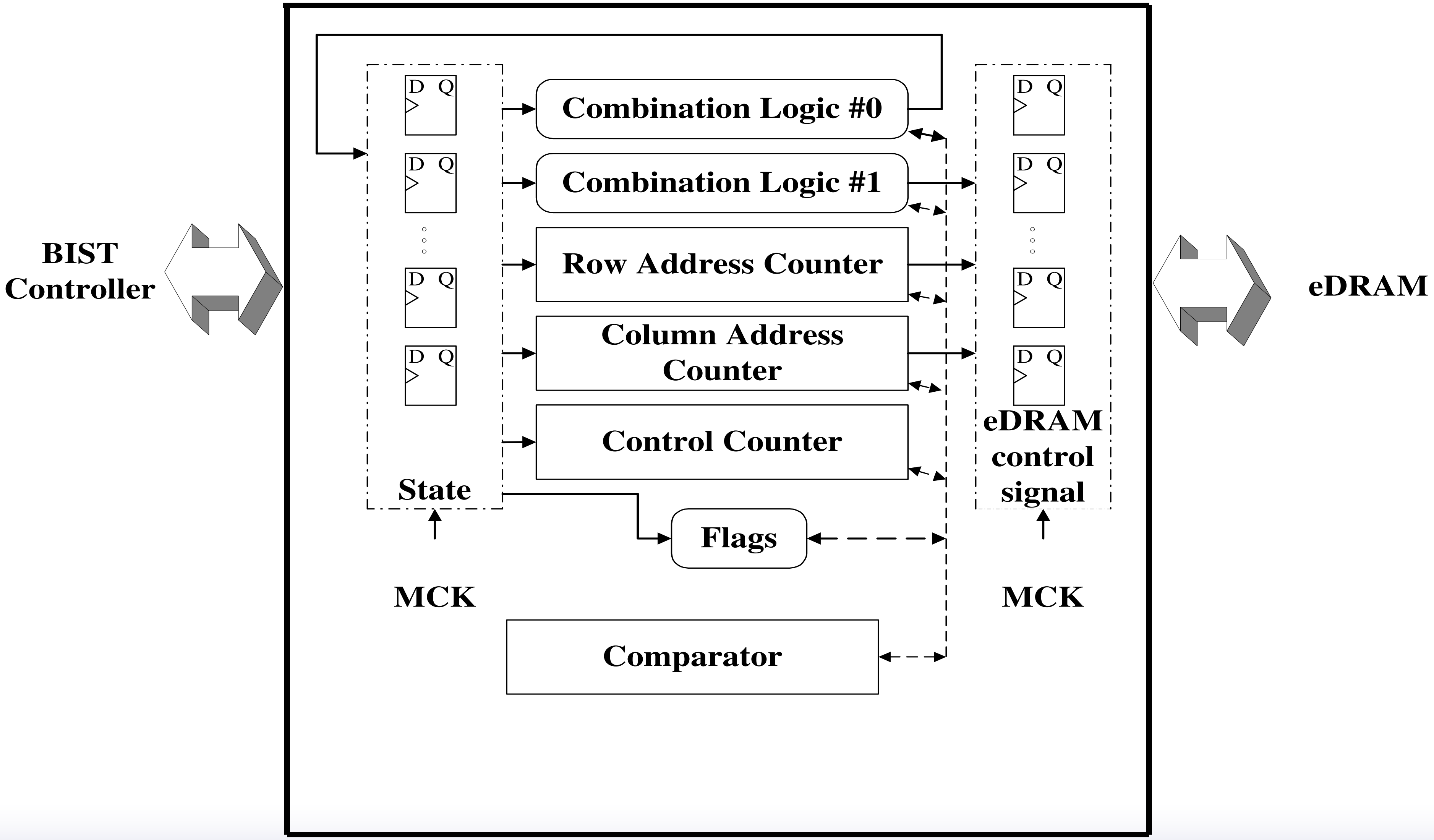
Scan test application and BIST activation

Memory function test, BI, AC test, etc.

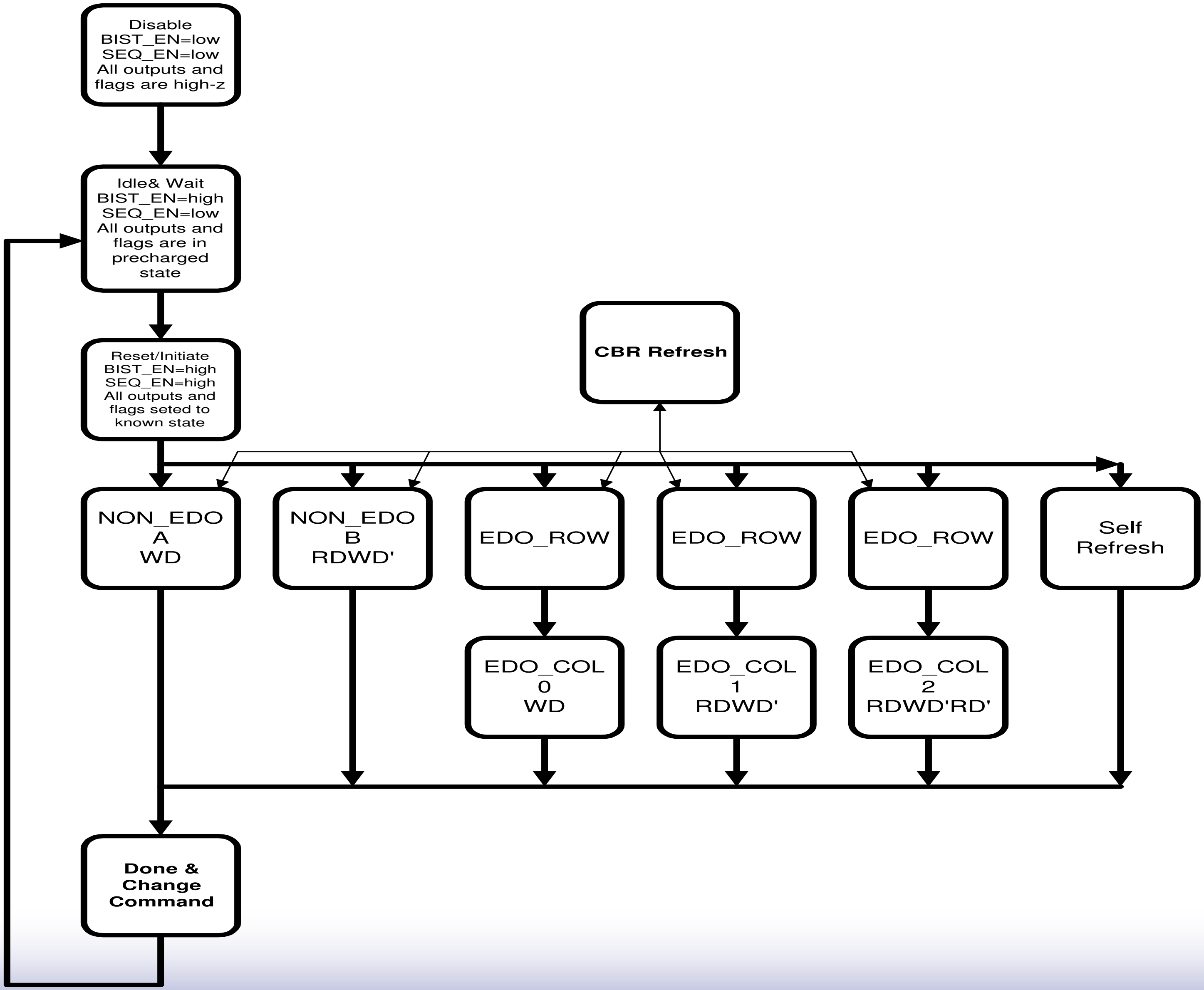
Pause for observation, or exit the execution phase.

Shifting out results, or pause for retention test.

Sequencer


BIST
Controller
D Q
Combination Logic #0
Combination Logic #1
Row Address Counter
Column Address
Counter
Control Counter
State
Flags
MCK
Comparator
eDRAM
control
signal
MCK
eDRAM

Sequencer States



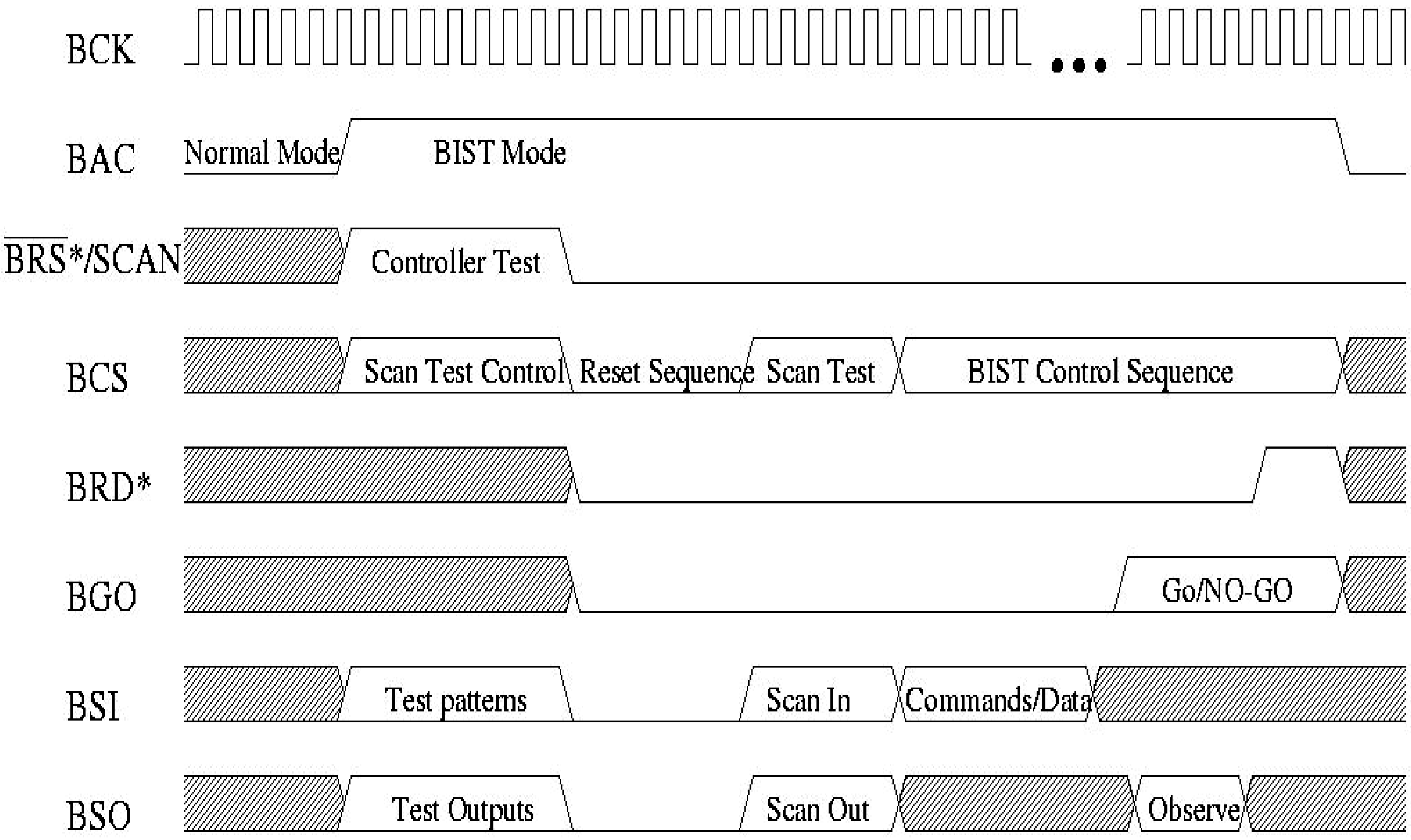
BIST Test Modes

1. Scan-Test Mode
2. RAM-BIST Mode
   1. Functional faults
   2. Timing faults (setup/hold times, rise/fall times, etc.)
   3. Data retention faults
3. RAM-Diagnosis Mode
4. RAM-BI Mode

BIST Controller Commands

Bit 4	Bit 3	Bit 2, Bit 1, Bit 0		
Addressing order	Data type	Operations		
1: ⇑ (increasing)	1: d = DB	000:	EOT	(End of test)
0: ⇓ (decreasing)	0: d = ~DB	001:	Rd	(READ Cycle)
		010:	Wd	(Early WRITE Cycle)
		011:	RdW~d	(READ-WRITE) Cycle
				EDO-PAGE-MODE
		100:	Wd	(Early WRITE Cycle
		101:	RdW~d	(READ-WRITE) Cycle
		110:	RdW~dR~d	(READ Early WRITE Cycle)
		111:	Refresh	

BIST Control Sequence



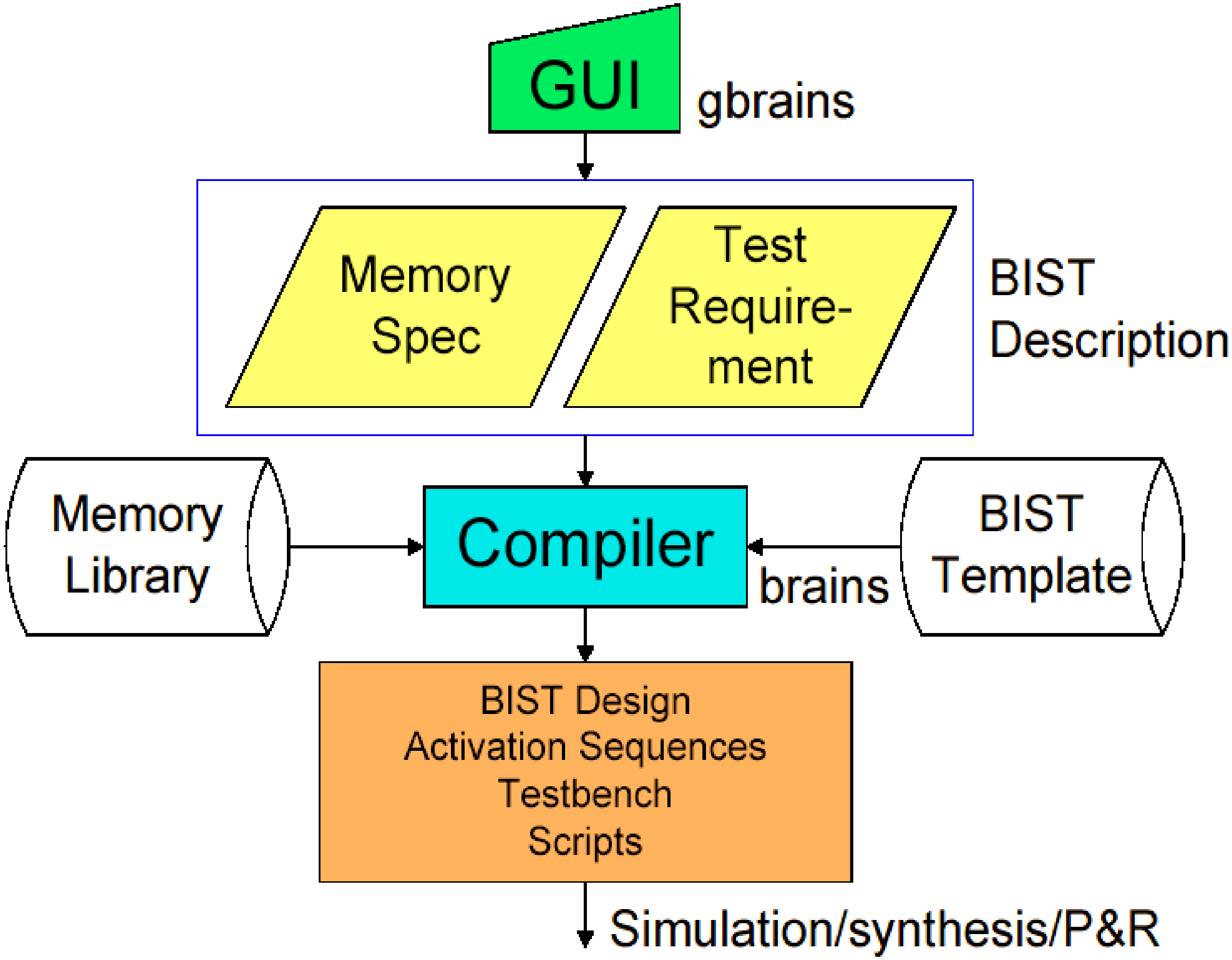
RAM BIST Compiler

- Use of RAM cores is increasing
  - SRAM, DRAM, flash RAM
  - Multiple cores
- RAM BIST compiler is the trend
- BRAINS (BIST for RAM in Seconds)
  - Proposed BIST Architecture
  - Memory Modeling
  - Command Sequence Generation
  - Configuration of the Proposed BIST

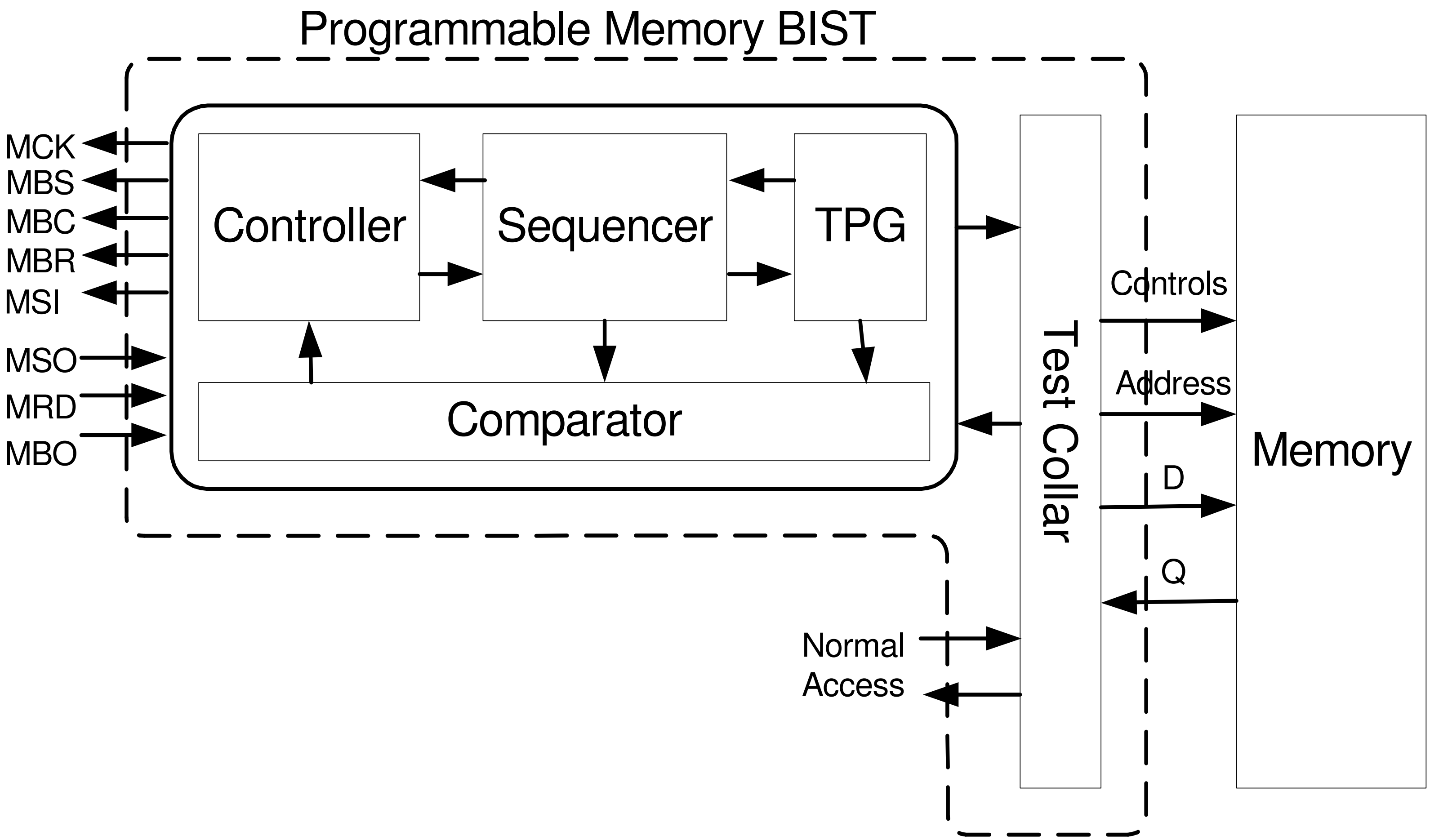
BRAINS Output Specification

- Synthesizable BIST design
  - At-speed testing
  - Programmable March algorithms
  - Optional diagnosis support
    - BISD
- Activation sequence
- Test bench
- Synthesis script

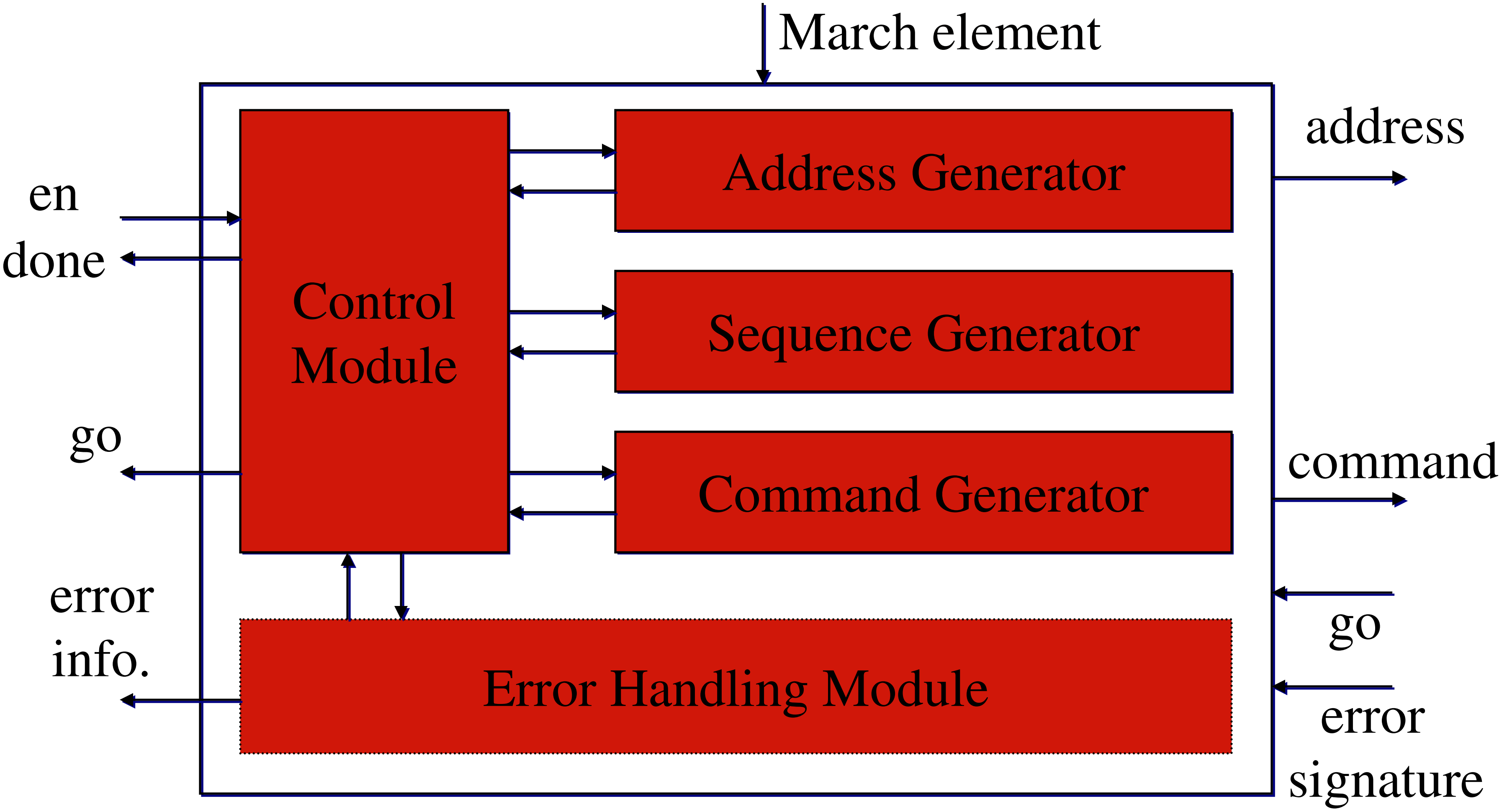
BRAINS Inputs and Outputs



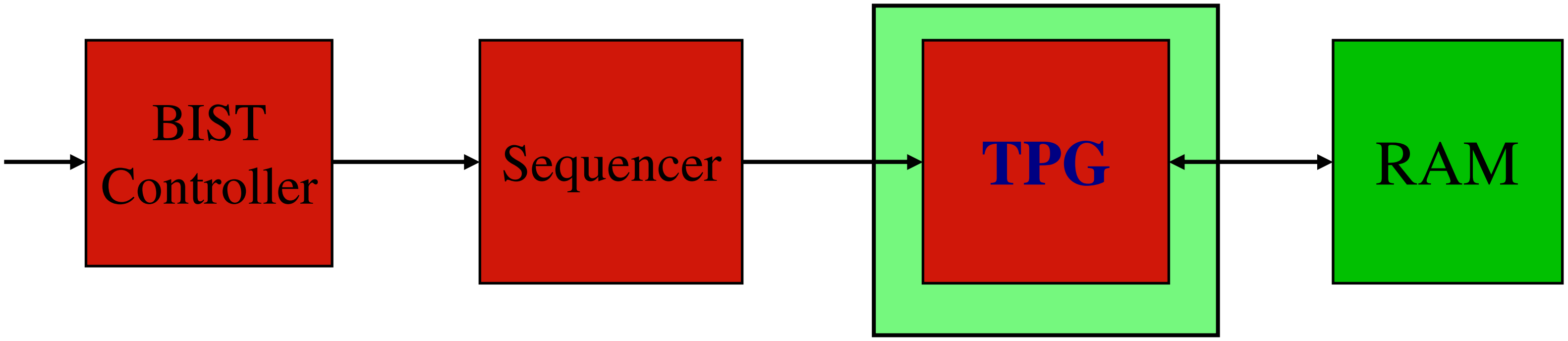
General RAM BIST Architecture



Sequencer Block Diagram

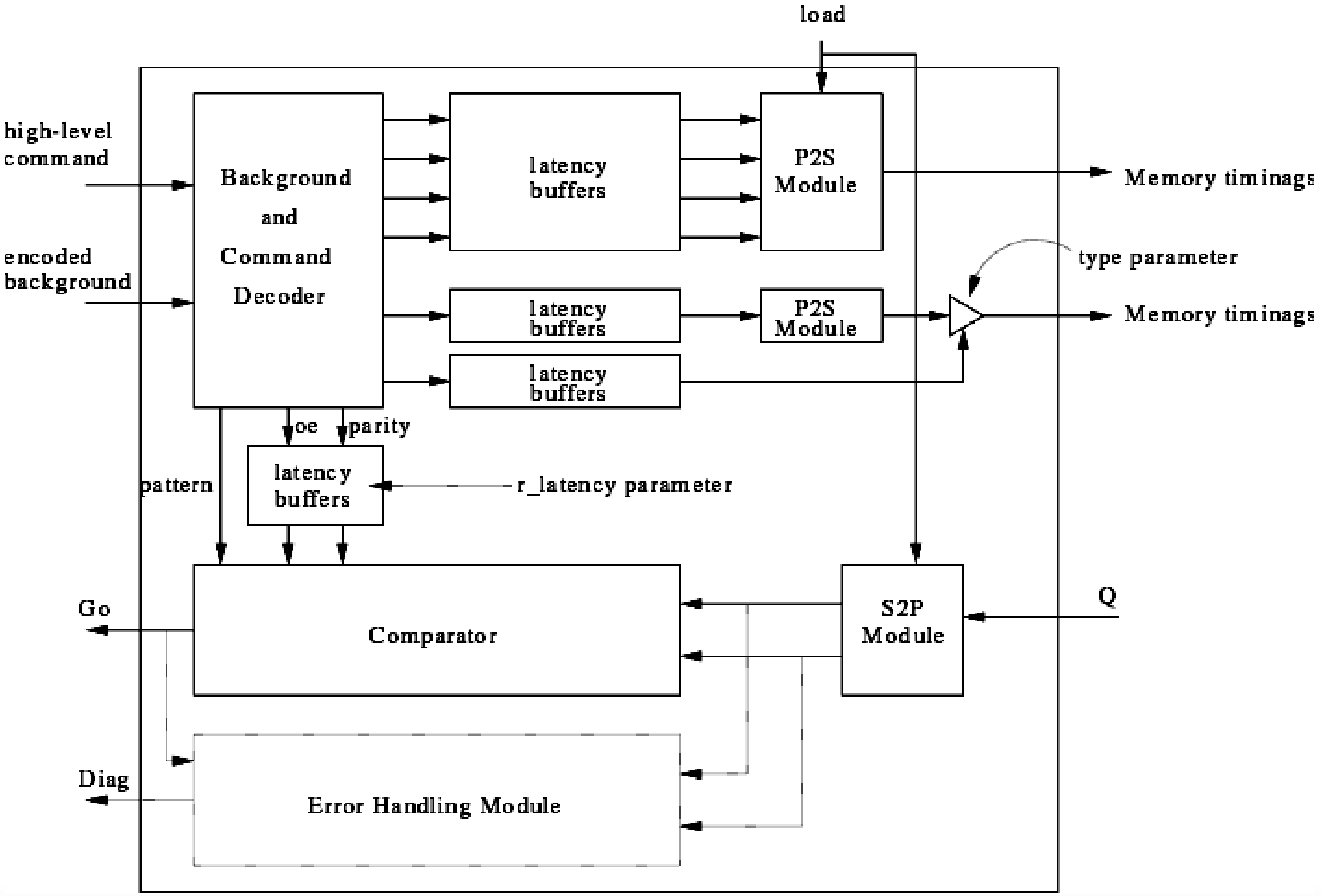


Function of the TPG



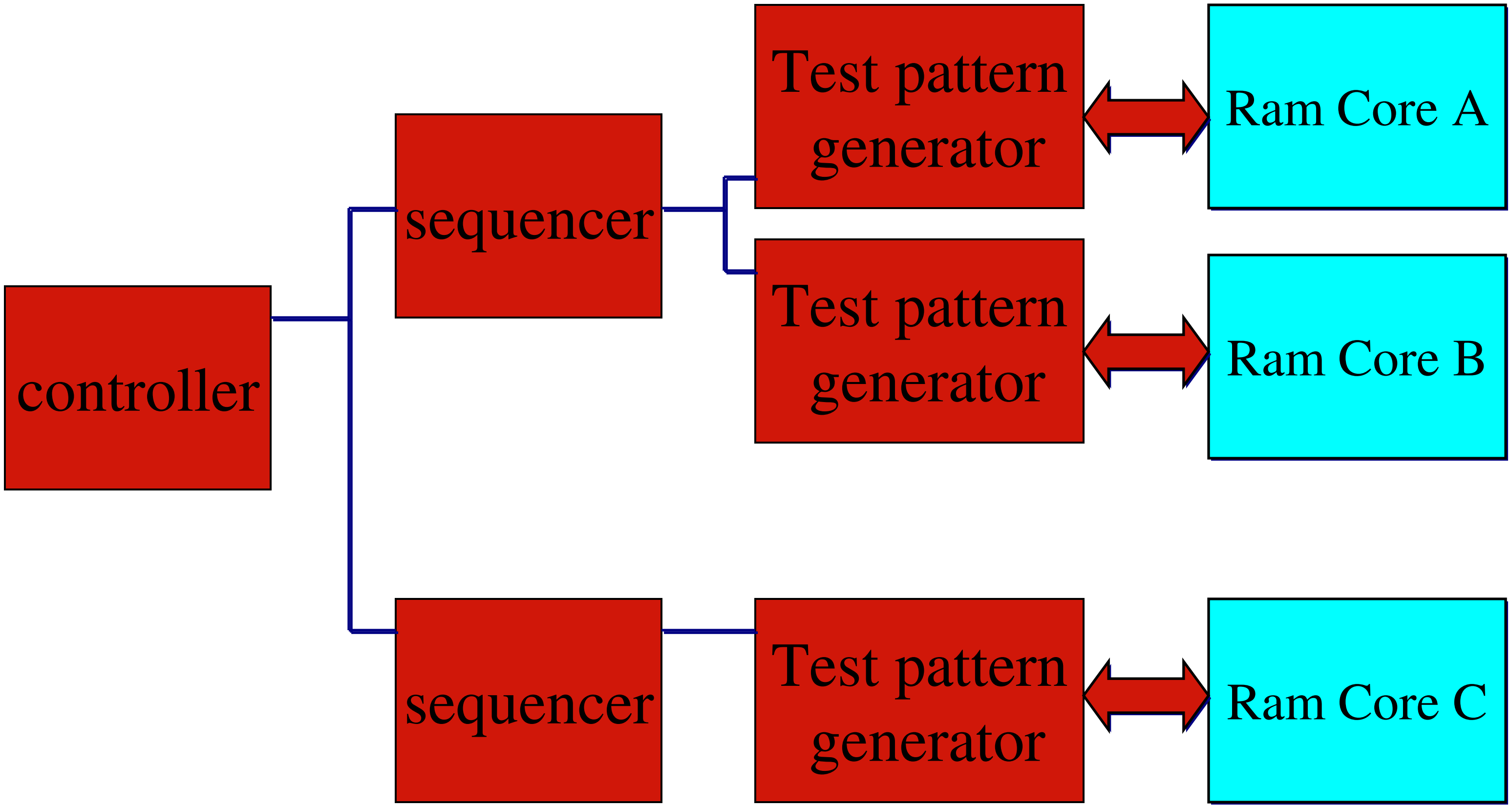
- The test pattern generator (TPG) translates high-level memory commands to memory input signals.
- Four parameters to model a memory's I/Os:
  - Type: input, output, and in/out
  - Width
  - Latency: number of clock cycles the TPG generates the physical signal after it receives a command from the sequencer
  - Packet_length: number of different signal values packed within a single clock cycle

Architecture of the TPG



Multiple RAM Cores

- Controller and sequencer can be shared



controller

sequencer

Test pattern generator

Ram Core A

Test pattern generator

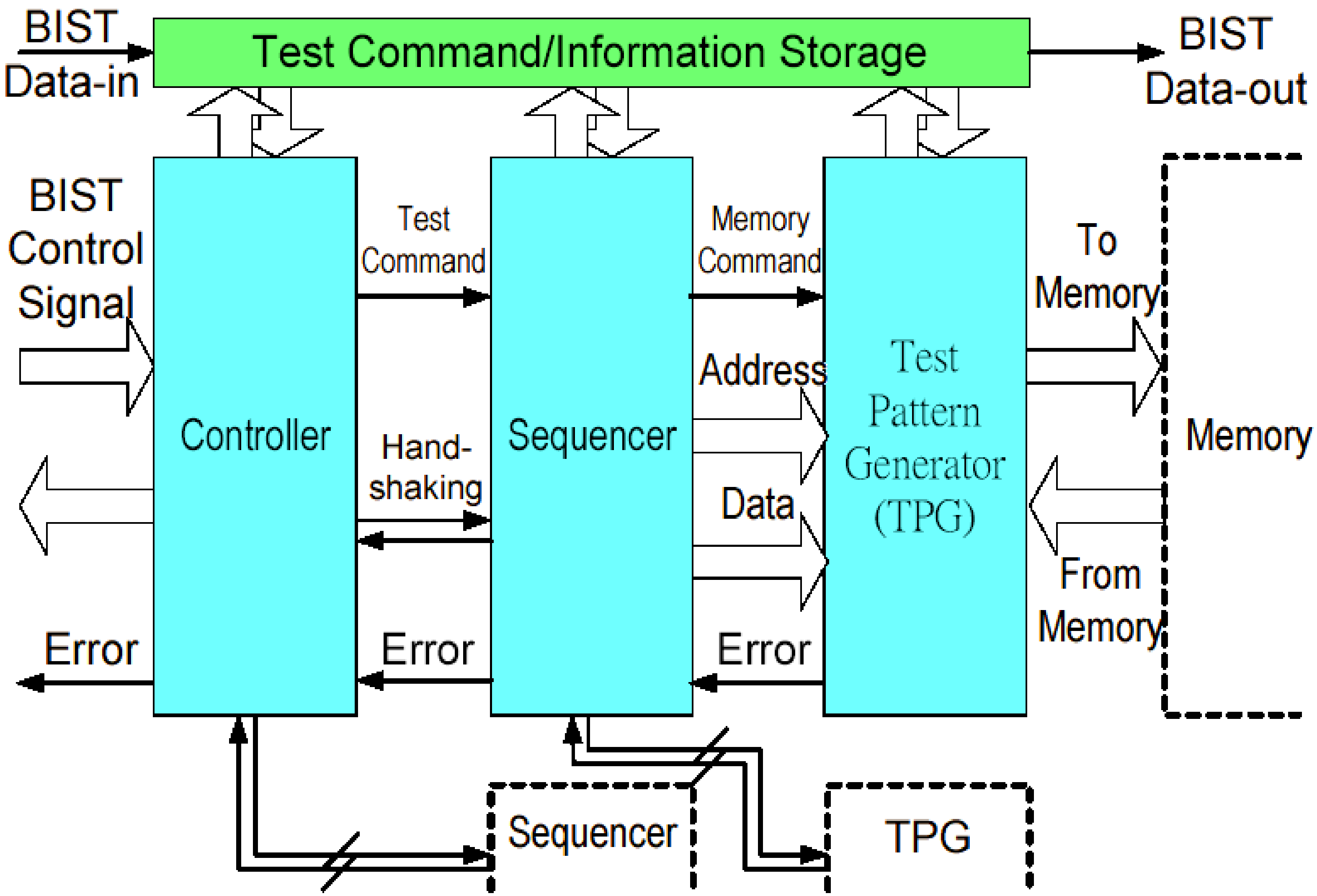
Ram Core B

sequencer

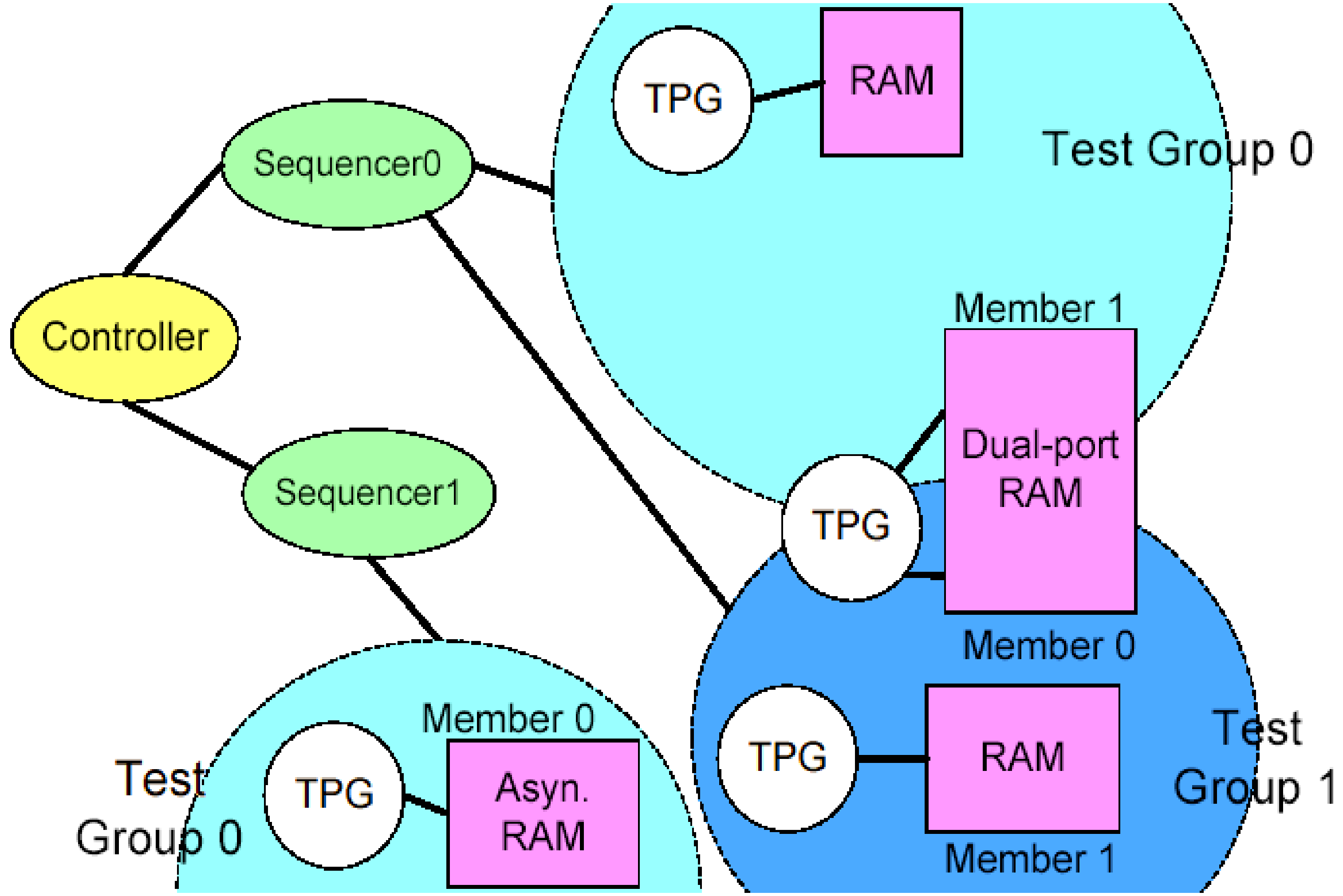
Test pattern generator

Ram Core C

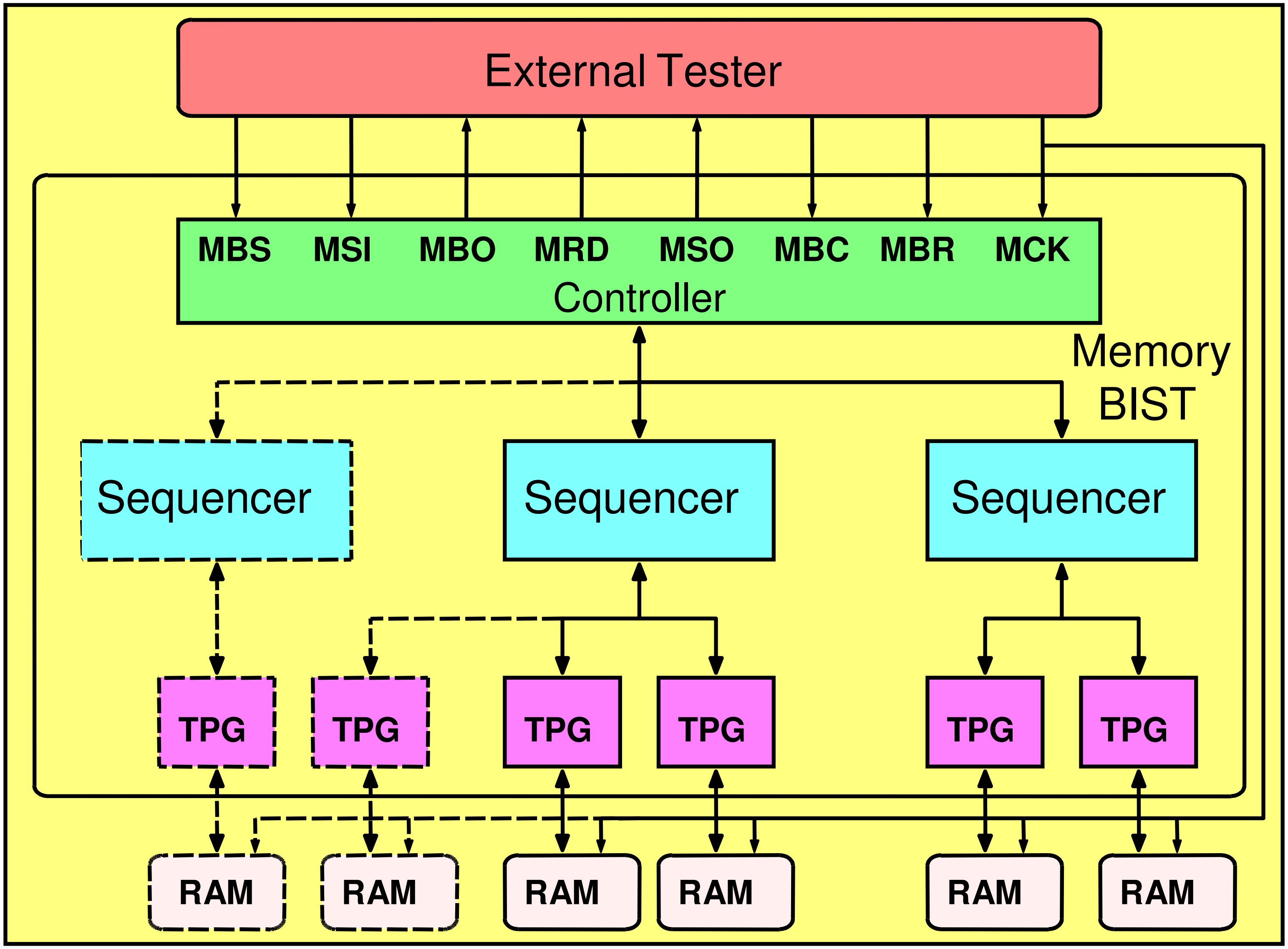
Sharing Controller & Sequencer



Grouping and Scheduling

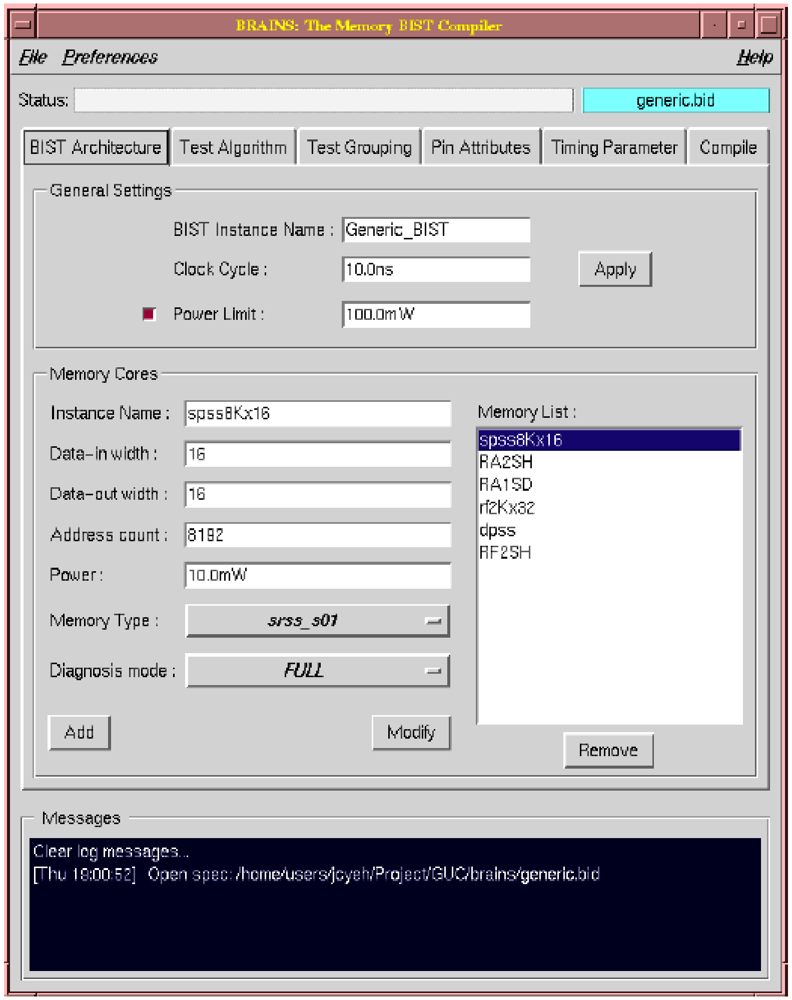


BRAINS BIST Architecture



Source: ATS'01

BRAINS GUI



Status: generic.bid

BIST Instance Name : Generic_BIST

Clock Cycle : 10.0ns

☒ Power Limit : 100.0mW

Instance Name : spss8Kx16

Data-in width : 16

Data-out width : 16

Address count : 8192

Power : 10.0mW

Memory Type : srss_s01

Diagnosis mode : FULL

Memory List :
- spss8Kx16
- RA2SH
- RA1SD
- rf2Kx32
- dpss
- RF2SH

```
Clear log messages...
[Thu 18:00:52]   Open spec: /home/users/jcyeh/Project/GUC/brains/generic.bid
```

BRAINS Release 2

Laboratory for Reliable Computing (LaRC)
Department of Electrical Engineering
National Tsing Hua University

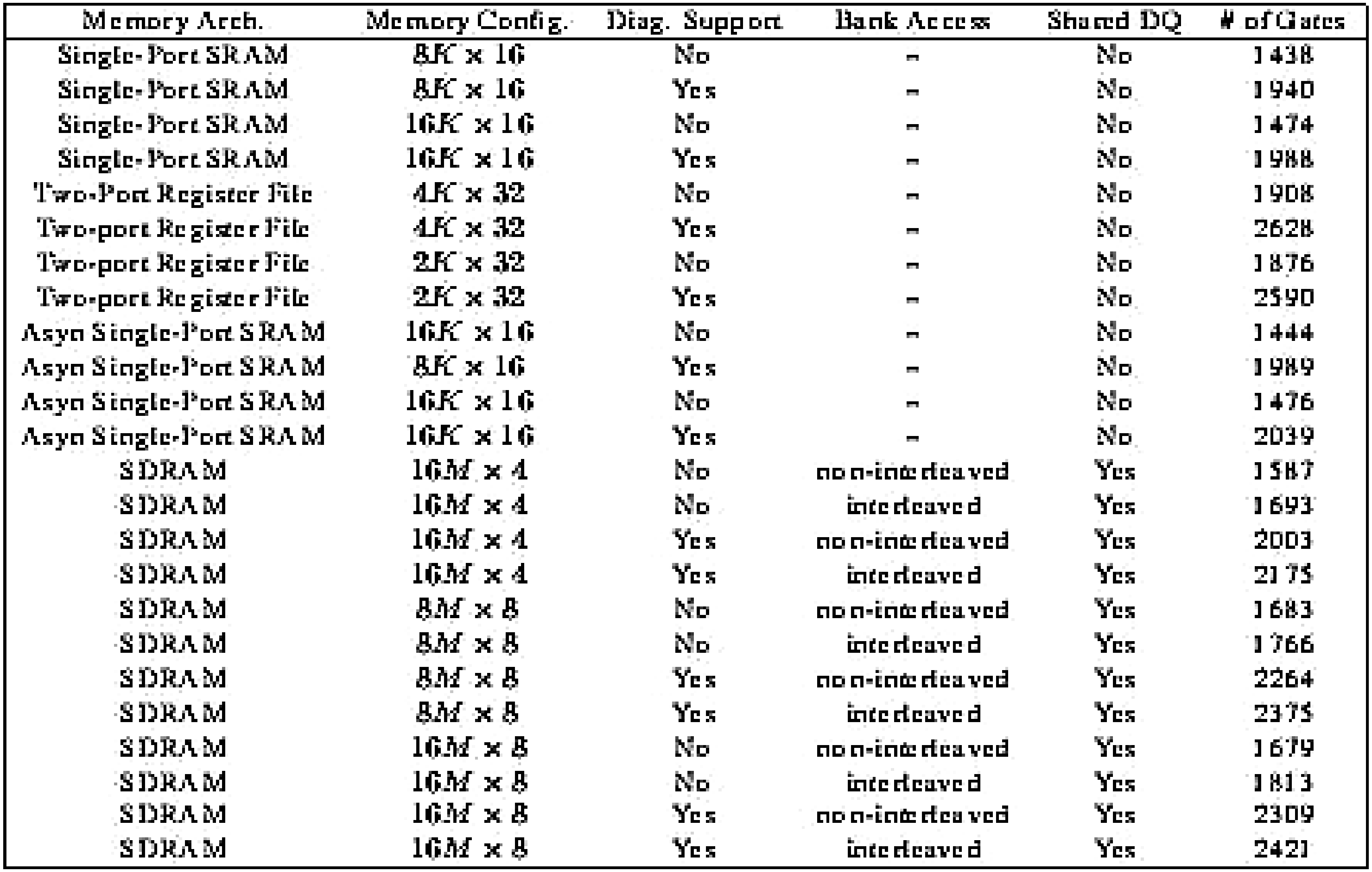
Authors:
Huang, Chih-Tsun
Cheng, Kuo-Liang
Hsueh, Chia-Ming
Yeh, Jen-Chieh
Huang, Rei-Fu

Contact Author:
Chih-Tsun Huang (cthuang@larc.ee.nthu.edu.tw)

Supported Memories

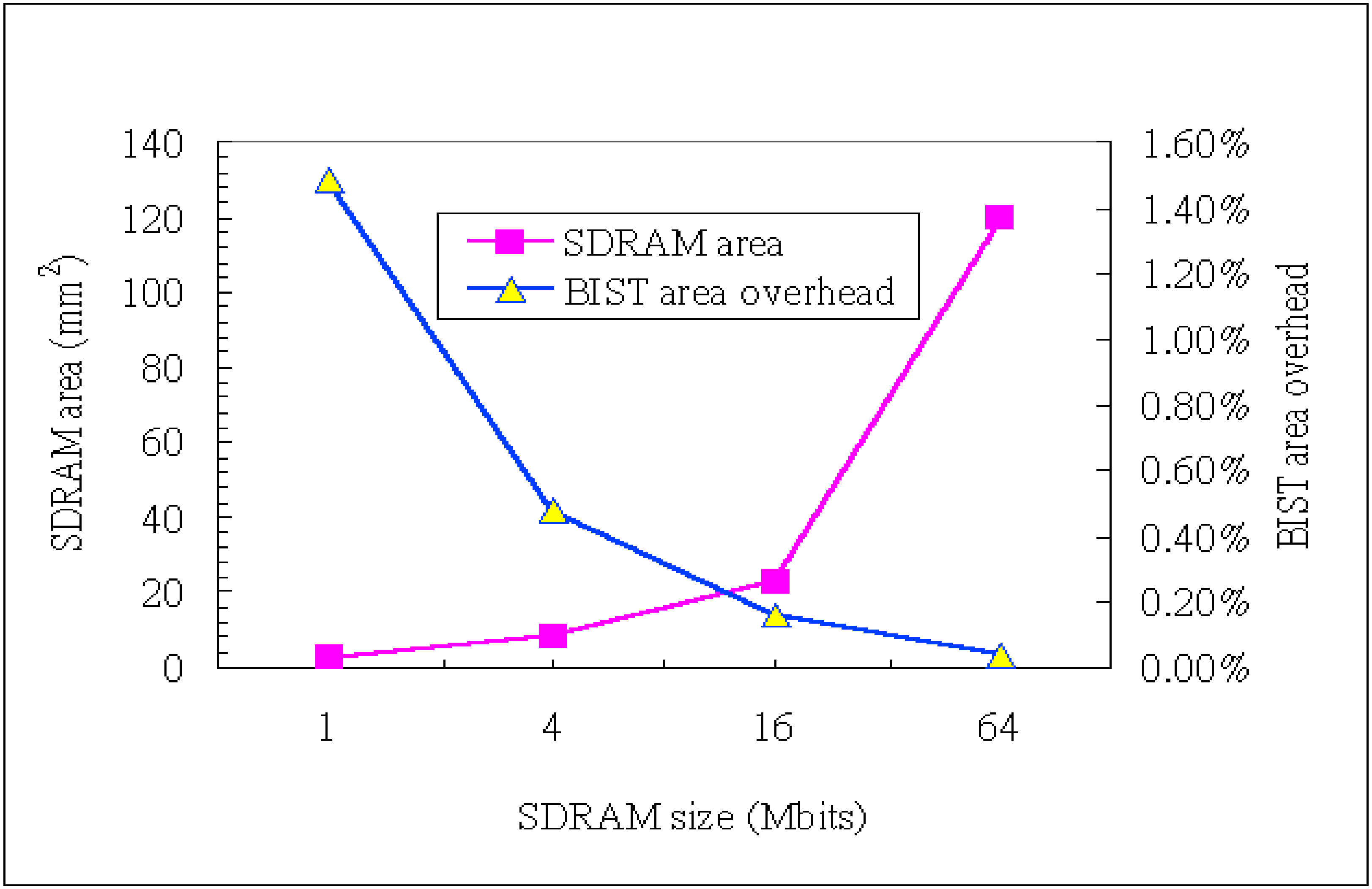
- ☐ The Built-In Memory List
  - DRAM
    - EDO DRAM
    - SDRAM
    - DDR SDRAM
  - SRAM
    - Single-Port Synchronous SRAM
    - Single-Port Asynchronous SRAM
    - Two-Port Synchronous Register File
    - Dual-Port Synchronous SRAM
    - Micron ZBT SRAM
- ☐ BRAINS can support new memory architectures easily

Examples



Memory Arch.	Memory Config.	Diag. Support	Bank Access	Shared DQ	# of Gates
Single-Port SRAM	8K × 16	No	–	No	1438
Single-Port SRAM	8K × 16	Yes	–	No	1940
Single-Port SRAM	16K × 16	No	–	No	1474
Single-Port SRAM	16K × 16	Yes	–	No	1988
Two-Port Register File	4K × 32	No	–	No	1908
Two-port Register File	4K × 32	Yes	–	No	2628
Two-port Register File	2K × 32	No	–	No	1876
Two-port Register File	2K × 32	Yes	–	No	2590
Asyn Single-Port SRAM	16K × 16	No	–	No	1444
Asyn Single-Port SRAM	8K × 16	Yes	–	No	1989
Asyn Single-Port SRAM	16K × 16	No	–	No	1476
Asyn Single-Port SRAM	16K × 16	Yes	–	No	2039
SDRAM	16M × 4	No	non-interleaved	Yes	1587
SDRAM	16M × 4	No	interleaved	Yes	1693
SDRAM	16M × 4	Yes	non-interleaved	Yes	2003
SDRAM	16M × 4	Yes	interleaved	Yes	2175
SDRAM	8M × 8	No	non-interleaved	Yes	1683
SDRAM	8M × 8	No	interleaved	Yes	1766
SDRAM	8M × 8	Yes	non-interleaved	Yes	2264
SDRAM	8M × 8	Yes	interleaved	Yes	2375
SDRAM	16M × 8	No	non-interleaved	Yes	1679
SDRAM	16M × 8	No	interleaved	Yes	1813
SDRAM	16M × 8	Yes	non-interleaved	Yes	2309
SDRAM	16M × 8	Yes	interleaved	Yes	2421

Area Overhead



Concluding Remarks

- ❑ BIST is considered the best solution for testing embedded memories:
  - Low cost
  - Effective and efficient
- ❑ Further improvement can be expected to extend the scope of RAM BIST:
  - Timing/delay faults and disturb faults
  - BISD and BISR
  - CAM BIST and flash BIST
  - BIST/BISD/BISR compiler
  - Wafer-level BI and test
    - Known good die