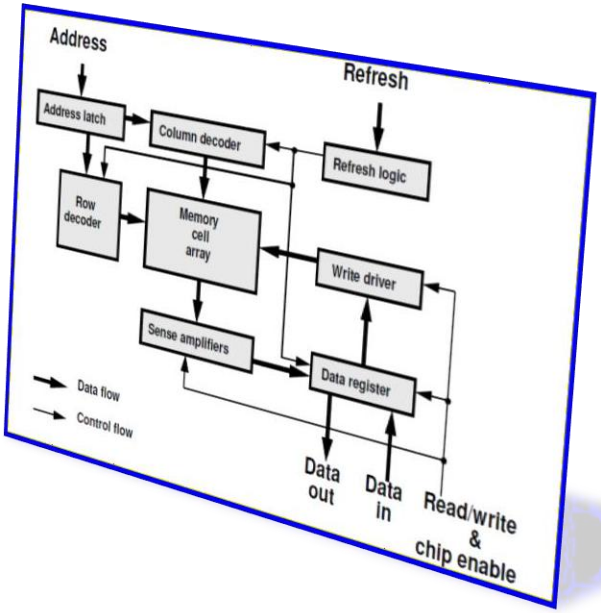
Memory Testing

Why Am I Learning This?

- **Testing memory is important**
  - **Because many memories are needed in modern designs**
- **Memory testing is very different from logic testing**

Outline

- **Introduction**
- **Memory Fault Models**
- **Memory Test Algorithms**
- **Memory Fault Simulation (*not in exam)**
- **Memory Test Generation (*not in exam)**
- **Memory BIST (*not in exam)**

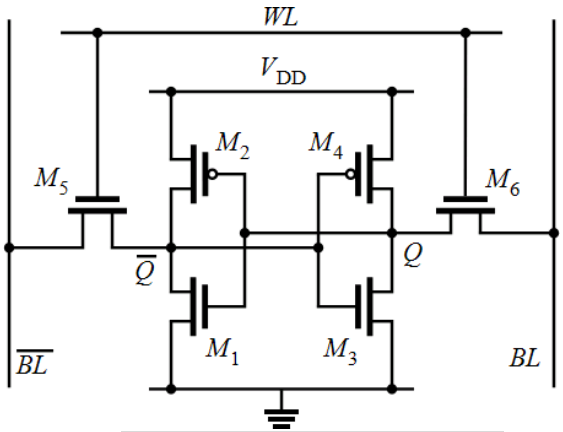


Memory Testing

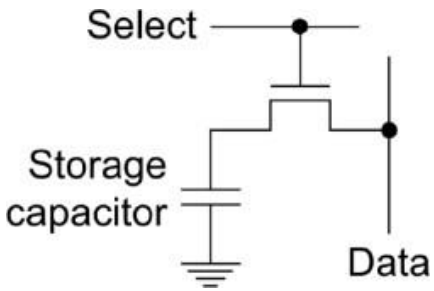
- Testing of memory is crucial for quality. Research started 1960's
  - Memory accounts for ~30% of semiconductor market (2019)
- Popular memory test items
  - Off-chip tests
    - DC parametric test: *e.g.* leakage current, output voltage level
    - AC parametric test: *e.g.* rise time, fall time
    - Functional test: *e.g.* march test (see 16.3)
    - Retention test: measure retention time of DRAM
    - Reliability test (Burn-in)
  - On-chip tests
    - *Error Detection and Correction* (*EDAC*): on-line testing
    - Built-in Self Test (BIST)
    - *Built-in Self Diagnosis & Repair* (*BISDR*)
- BIST important for *System-on-chip (SOC)* with *embedded memories*

Types of Memories

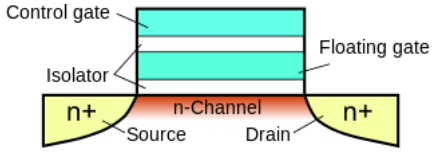
type	area	speed	retention	application	test method
SRAM	largest 6T	fastest <1ns	as long as power	embedded SRAM cache, registers	BIST
DRAM	medium 1T+1C	medium ~10ns	< sec.	embedded DRAM	BIST/ ATE
				on-board memory	ATE
Flash	smallest 1T	slowest ~100μs	years	SSD, USB drive	ATE



SRAM cell [wikipedia]

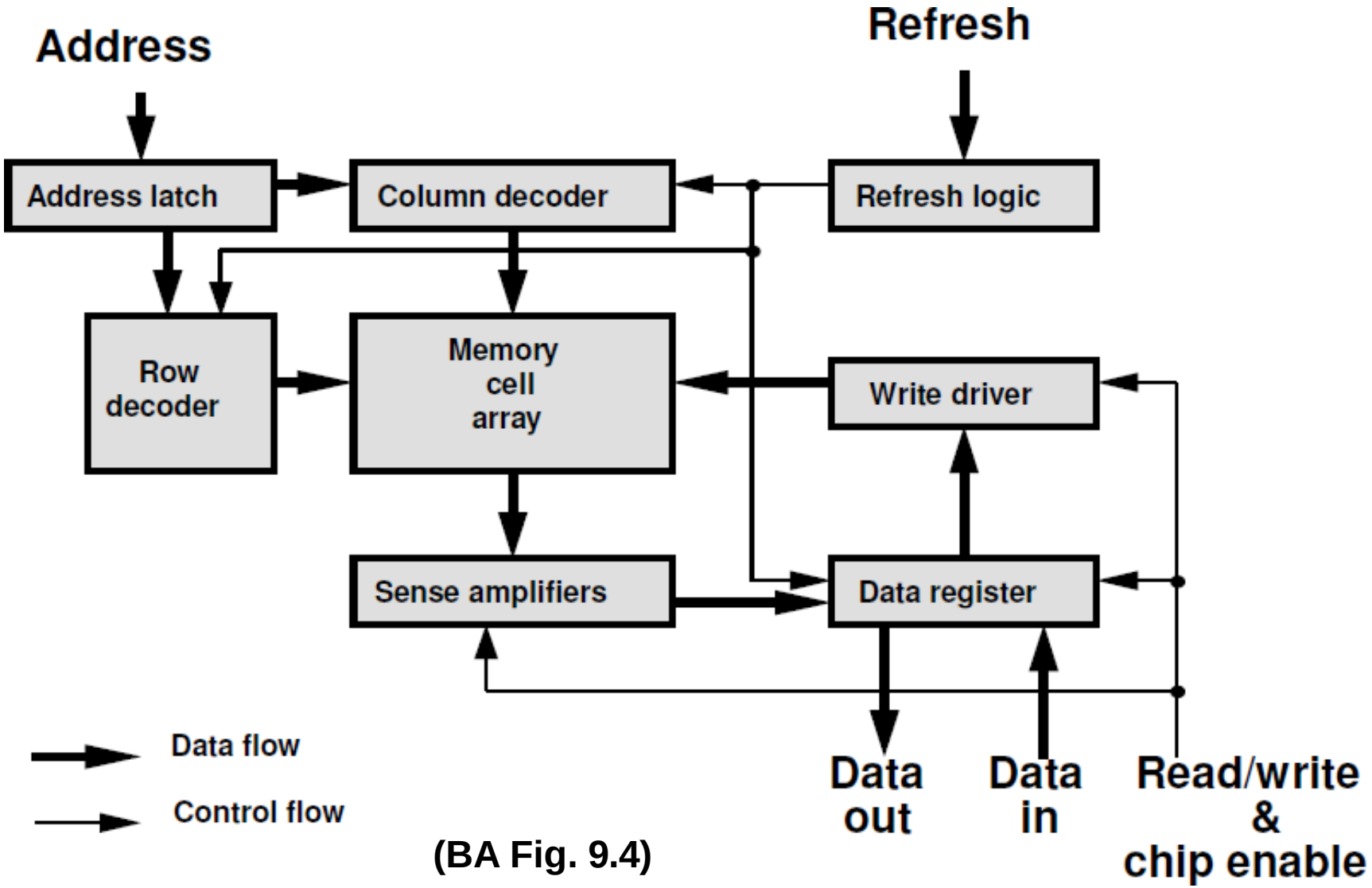


DRAM cell



Floating Gate Transistor [wikipedia]

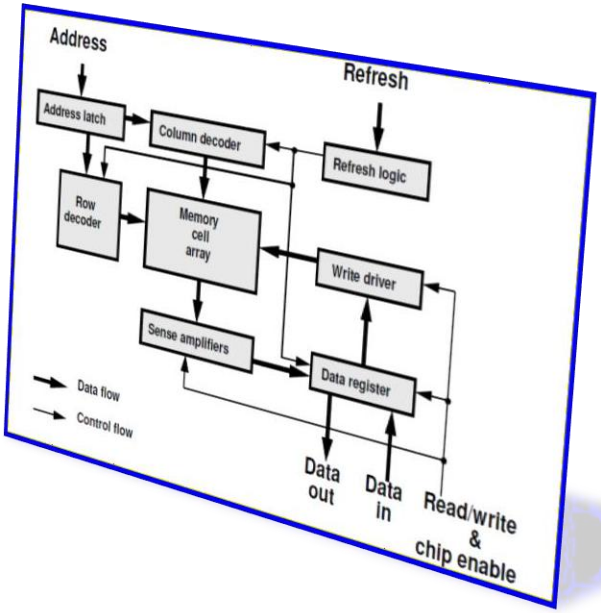
DRAM Functional Model



(BA Fig. 9.4)

Outline

- Introduction
- Memory Fault Models (focus on RAM)
  - Static faults
    - Single cell fault, Double cell fault, Address-decoder fault
  - Dynamic faults
    - Recovery fault, Retention fault
- Memory Test Algorithms
- Memory Fault Simulation (*not in exam)
- Memory Test Generation (*not in exam)
- Memory BIST (*not in exam)



RAM Fault Models

- *Functional testing* is commonly used for memories
  - Scan testing for logic is not applicable to memory. (Why? FFT)
- *Functional fault models* are behavior model for faulty memories
  - based on real defects, not imagination
- Popular RAM functional fault models
  1. Static fault models: faulty behavior does NOT change with time
     - single cell, double cell, address decoder
  2. Dynamic fault models: faulty behavior changes with time
     - recovery, data retention

- NOTE: many other RAM fault models (neighborhood pattern sensitive faults …)

Different Memories Need Different Fault Models

RAM Fault Models (1) – single cell

- *Stuck-At Fault* (SAF)
  - a cell is always 0, SA0
  - a cell is always 1, SA1
- *Stuck-Open Fault* (SOF)
  - a cell cannot be accessed due to broken wire
- *Transition Fault* (TF)
  - a cell fails to
    - Rise from 0 to 1 <↑ / 0>
    - Fall from 1 to 0 <↓ / 1>

NOTATION: <*S*/*F*>: a fault in a cell [van de Goor 91]

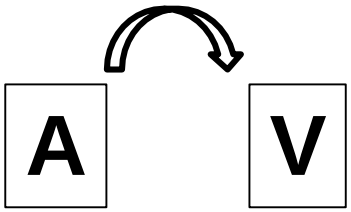
- *S* is value or operation activating fault , $S \in \{0, 1, \uparrow, \downarrow, \updownarrow, \forall\}$
  - ↑ is rising; ↓ is falling; ↕ is either ↑ or ↓; ∀ means any condition
- *F* is faulty value of cell , $F \in \{0, 1, \updownarrow\}$
  - ↕ is complement

RAM Fault Models (2) – double cell

- *Coupling Fault* (CF): *Victim* cell is affected by *Aggressor* cell
  - *1.State Coupling Fault* (CF_{st})
    - if aggressor cell is in given state, victim cell is forced to 0 or 1
    - 4 types: <0; 0/1> or <0; 1/0> or <1; 1/0> or <1; 0/1>
  - *2.Inversion Coupling Fault* (CF_{in})
    - if aggressor cell rise/fall, victim cell is complemented
    - 2 types: $<\uparrow ; \forall/\updownarrow >$ or $< \downarrow ; \forall/\updownarrow >$
  - *3.Idempotent Coupling Fault* (CF_{id})
    - if aggressor cell rise/fall, victim cell is forced to 0 or 1
    - 4 types: $< \uparrow$; 0/1> or $< \uparrow$; 1/0> or $< \downarrow$; 0/1 > or $< \downarrow$; 1/0 >

NOTATION: $<S_1; S_2/F>$ faults in 2 cells

- S_1 is value or operation activating fault in aggressor
- S_2 is value or operation activating fault in victim
- F is faulty value of victim



CF Examples

CF_{st}<0,0/1>

addr	content*
A	0
V	0

⇩ do nothing

addr	content
A	0
V	0/1

⇩ read V

detected

*typically many bits in one address but only one bit here for illustration

CF_{id}<↑; 0/1>

A	0
V	0

⇩ write 1 to A

A	1
V	0/1

⇩ read V

detected

A	0
V	1

⇩ write 1 to A

A	1
V	1

⇩ read V

not detected

CF_{in}<↑; ∀ /↕ >

A	0
V	0

⇩ write 1 to A

A	1
V	0/1

⇩ read V

detected

A	0
V	1

⇩ write 1 to A

A	1
V	1/0

⇩ read V

detected

QUIZ

Q: Consider CF_{in}<↓, ↕ >. Fill in values for two cases. Can we detect faults in both cases?

ANS:

addr	content
A	1
V	1

⇩ write 0 to A

addr	content
A	0
V	?

⇩ read V=?

addr	content
V	1
A	1

⇩ write 0 to V

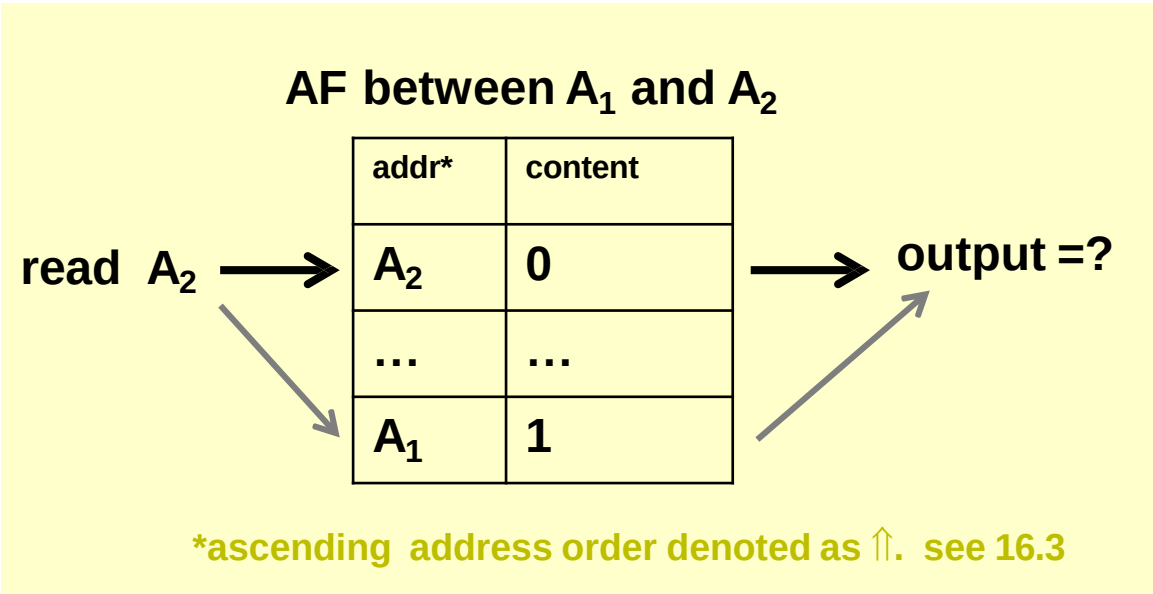
addr	content
V	0
A	?

⇩ read A=?

Address Order Matters

RAM Fault Models (3) – AF

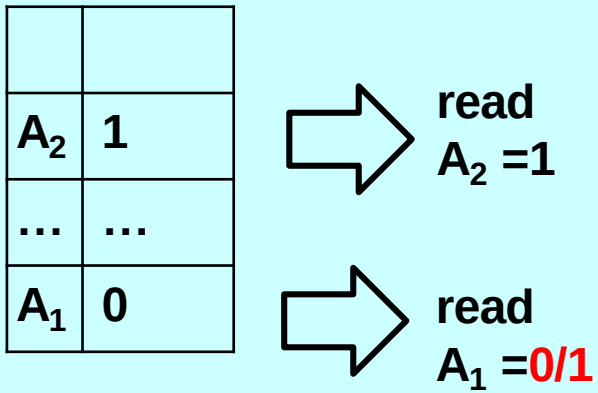
- *Address-Decoder Fault* (AF) **Four faulty behavior:**
  1. **Given a certain address, no cell will be accessed**
  2. **A certain cell is never accessed by any address**
  3. **A certain cell can be accessed by multiple addresses**
  4. ☆ **Given a certain address, multiple cells are accessed**
     - **AND-type**
     - **OR-type**



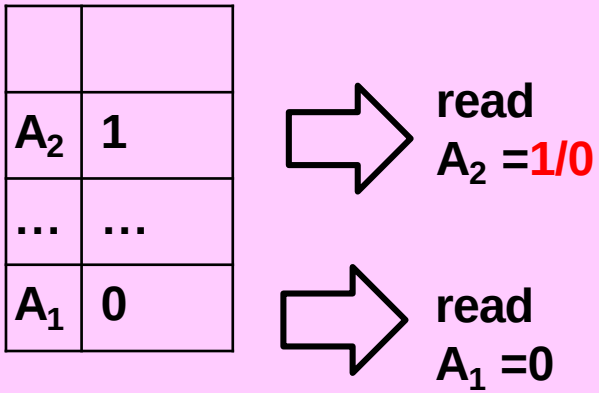
Only Consider #4 AF

AF Examples

OR-type AF
between A_1 and A_2

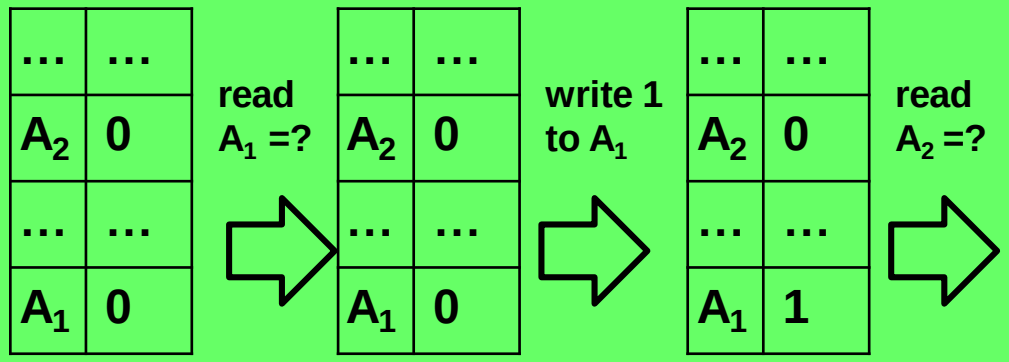


AND-type AF
between A_1 and A_2

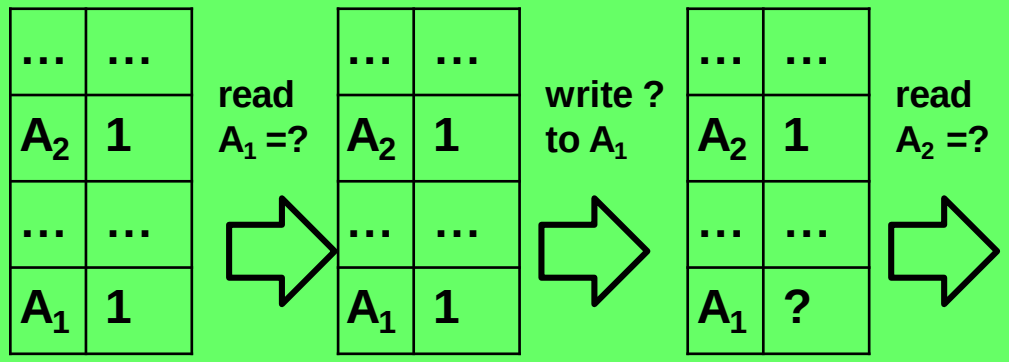


QUIZ

Q1: Given OR-type AF between A_1 and A_2. Fault detected?



Q2: Given AND-type AF. Find a test to detect fault.



AND/OR AF Need Opposite Test Data

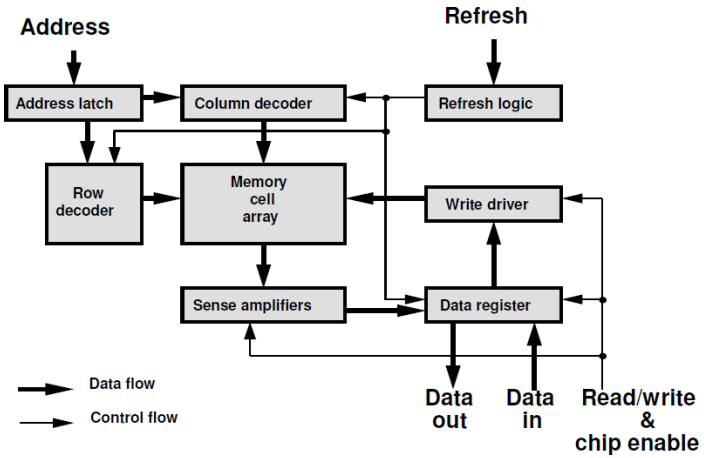
RAM Fault Models (4) – dynamic faults

- ***Data Retention Fault (DRF):*** **data changed after a certain time**
  - **DRAM**
    1. **Charge leakage loose data in capacitor**
    2. **Refresh logic fails to refresh correctly**
  - **SRAM**
    - **Defective pull-up device inducing excessive leakage current which changes the state of cell**
- ***Sense amplifier recovery fault***
  - **Sense amp. saturated after reading/writing a long string of 0 or 1**
- ***Write recovery fault***
  - **A write followed by a read/write at a different location results in reading or writing at the same location due to slow address decoder**

Time Consuming to Test Dynamic Faults

Summary

- Memory test important. Good diagnosis/repair can improve yield
- Popular RAM functional fault models
  - Static fault models
    - Single cell: Stuck-at (SAF), Stuck-open (SOF), Transition (TF)
    - Double cell coupling faults: CF_{in}, CF_{id}, CF_{st}
    - Address decoder fault (AF): AND-type OR-type
  - Dynamic fault models
    - Data retention faults (DRF). Recovery faults
- Fault model must be realistic
  - Different memories need different fault models



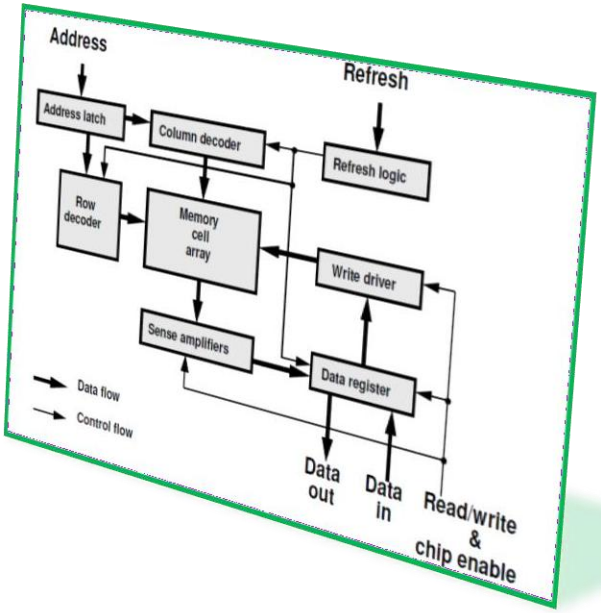
FFT

- **Q1: Scan testing for logic is not applicable to memory.**
  - **Why no scan?**

- **Q2: AF has four faulty behavior.**
  - **We only consider #4. The others are easy to test, why?**
    1. **Given a certain address, no cell will be accessed**
    2. **A certain cell is never accessed by any address**
    3. **A certain cell can be accessed by multiple addresses**
    4. ☆ **Given a certain address, multiple cells are accessed**

Memory Testing

- Introduction
- Memory Fault Model
- **Memory Test Algorithms**
  - **Classical algorithms**
  - March algorithms
- Memory Fault Simulation
- Memory Test Generation
- Memory BIST



Memory Test Algorithms

- A *Test algorithm* is a finite sequence of *test elements*
- A *test element* contains a number of
  1. *Memory operations*
     - Read or Write
  2. *Data pattern* (aka. *data background*)
     - Zero or One
  3. *Address sequence*
     - *Ascending* or *Descending*
- *Test (time) Complexity* of test algorithm is expressed in terms of *N*
  - *N* = memory size = number of memory cells
  - Higher complexity means longer test time

Mem. Test Alg. is Different from ATPG Alg.

Memory Test (Time) Complexity

Size	N	$10N$	$N \lg N$	$N^{1.5}$	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	59c
16G	2.8m	28m	1.6h	261d	936c

N = number of memory cells; 100MHz test speed

Linear Complexity Feasible for Production Tests

QUIZ

Q: Which one is correct about memory testing?

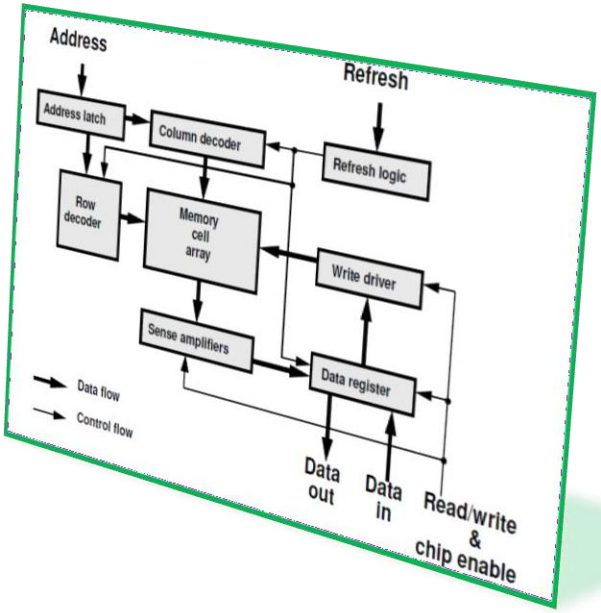
A) Higher complexity means longer CPU time

B) Test algorithm means a finite sequence of test elements, which contain: memory operation, data pattern, and address sequence

C) (*N* log *N*) Complexity is acceptable for large memory

Memory Testing

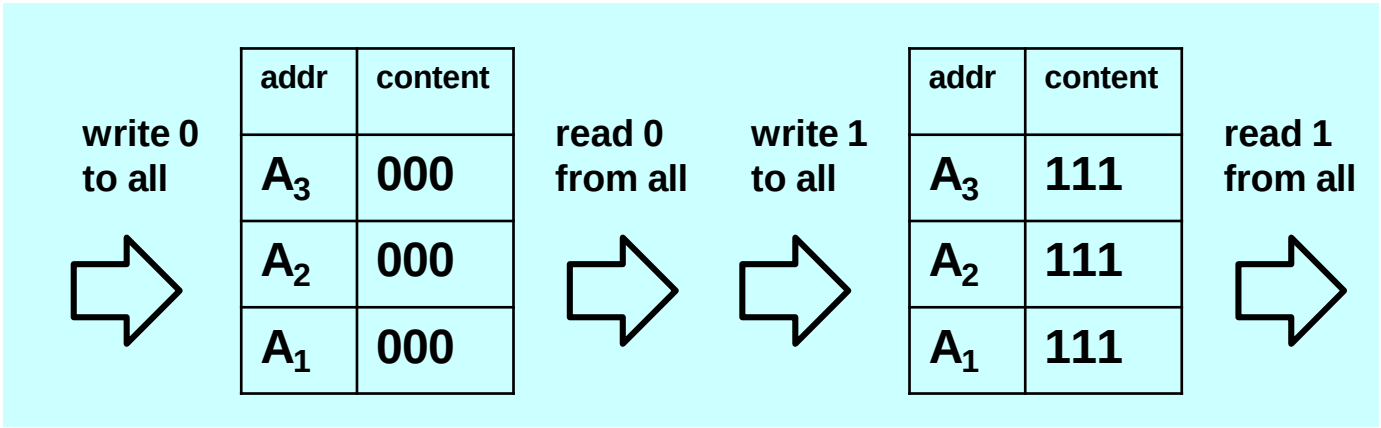
- Introduction
- Memory Fault Model
- **Memory Test Algorithms**
  - **Classical algorithms**
    - **MSCAN**
    - **Checkerboard**
    - **GALPAT**
    - **Butterfly**
  - March algorithms
- Memory Fault Simulation
- Memory Test Generation
- Memory BIST



MSCAN

- aka. *zero-one algorithm*
- Detects all SAF
- Detects <↑/0> TF, not <↓/1>
- Does NOT detect all AF, CF
- Complexity is *4N*
  - 4 operations each cell

MSCAN
1. Write zero to every cell
2. Read zero from every cell
3. Write one to every cell
4. Read one from every cell



Checkerboard

- Detects all SAF and half TF
- Does NOT detect all AF, CF
- Complexity is *4N*

Same as MSCAN but Detects *Bridging Fault*

CHECKERBOARD
1. Write chbd pattern to all cells
2. Read ckbd pattern from all cells
3. Write ckbd' pattern to all cells
4. Read ckbd' pattern from all cells

ckbd means 01 alternating
ckbd' is complement of ckbd

write ckbd

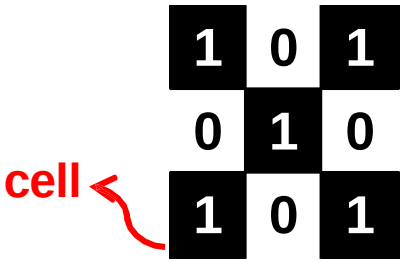
addr	content
A_3	101
A_2	010
A_1	101

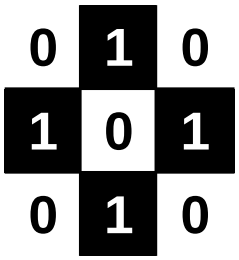
read ckbd

write ckbd'

addr	content
A_3	010
A_2	101
A_1	010

read ckbd'

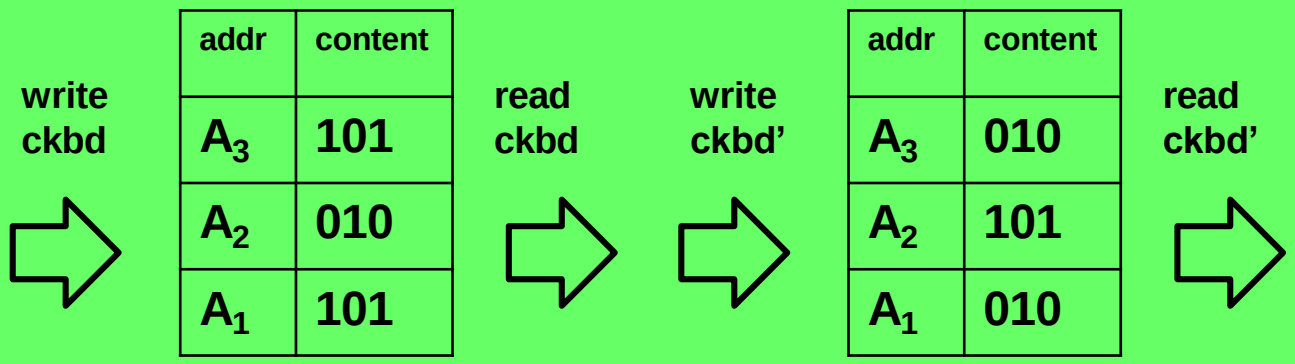




QUIZ

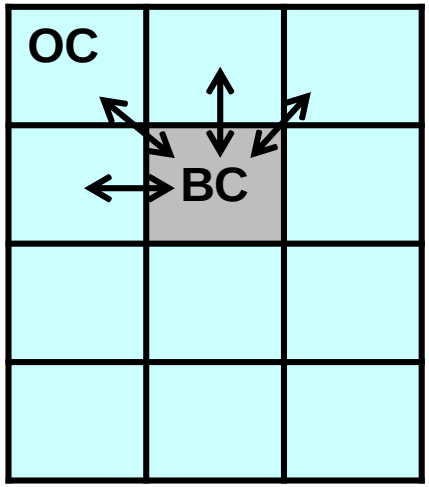
Q1: Can checkerboard detect OR-type AF between A_1 and A_2?
ANS:

Q2: Can checkerboard detect OR-type AF between A_1 and A_3?
ANS:



Galloping Test (GALPAT)

- Aka. *Ping-pong test*
- Detects all SAF, TF
- Detects CF, and AF
- Complexity is $4N^2$
  - Line 2: $2N^2$
  - Line 4: $2N^2$



memory cell array

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

Too Long! Only for Characterization

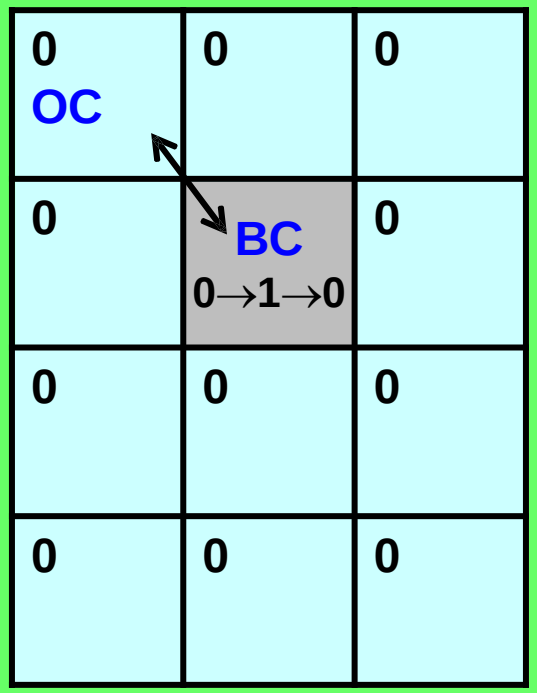
QUIZ

Suppose BC is aggressor and OC is victim.

Q1: Can we detect CF_{st} <1; 0/1>?

Q2: Can we detect CF_{id} <↑; 0/1> ?

Q3: Can we detect CF_{in} <↑ ; ∀/↕ >?

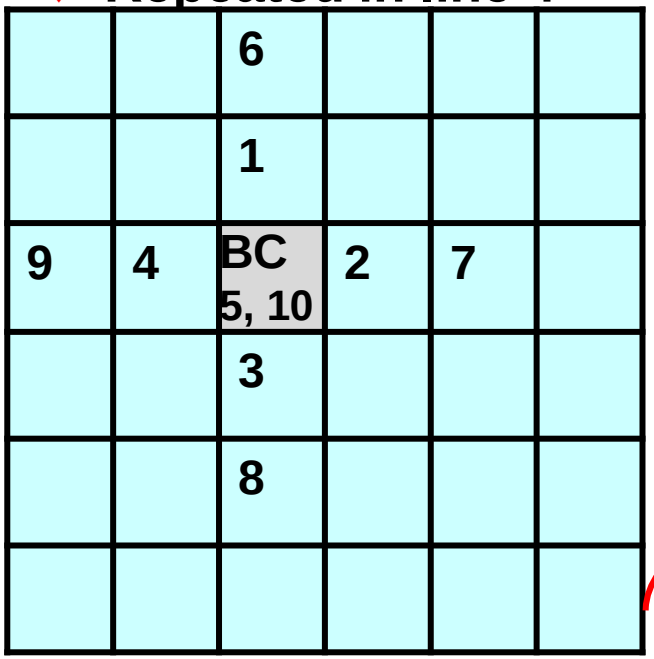


GALPAT

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

Butterfly Algorithm

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



memory cell array

```
BUTTERFLY      // 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;
```

Summary

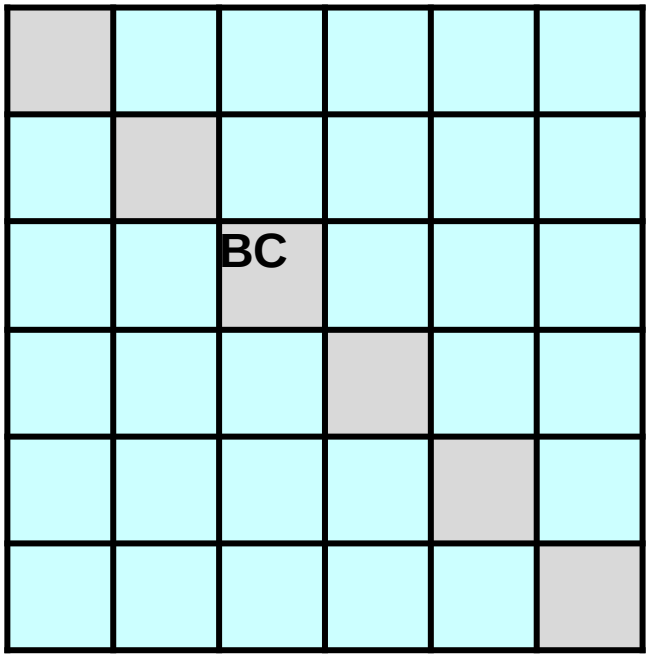
- Test algorithm is a finite sequence of test elements
  - which are: Memory operation, Data pattern, Address sequence
  - Test complexity means test time (not CPU time)
- Four classical test algorithms
  - MSCAN and Checkerboard's fault coverage NOT good
  - GALPAT and Butterfly are too slow
- Need linear-time test algorithms with good fault coverage
  - *March test algorithms* (see 16.3)

	SAF	AF	TF	CF	Complexity
MSCAN	D	-	-	-	$4N$
Checkerboard	D	-	-	-	$4N$
GALPAT	D	D	D	D	$4N^2$
Butterfly	D	-	D	-	$10\ N \log N$

D: all detected;
- : not all detected

FFT

- Q: In GALPAT, choosing all cells as base cell is slow. Can we choose only cells on the diagonal line as BC? What is complexity?
- Q: Why do we choose cells in the same column and row of BC in Butterfly test?



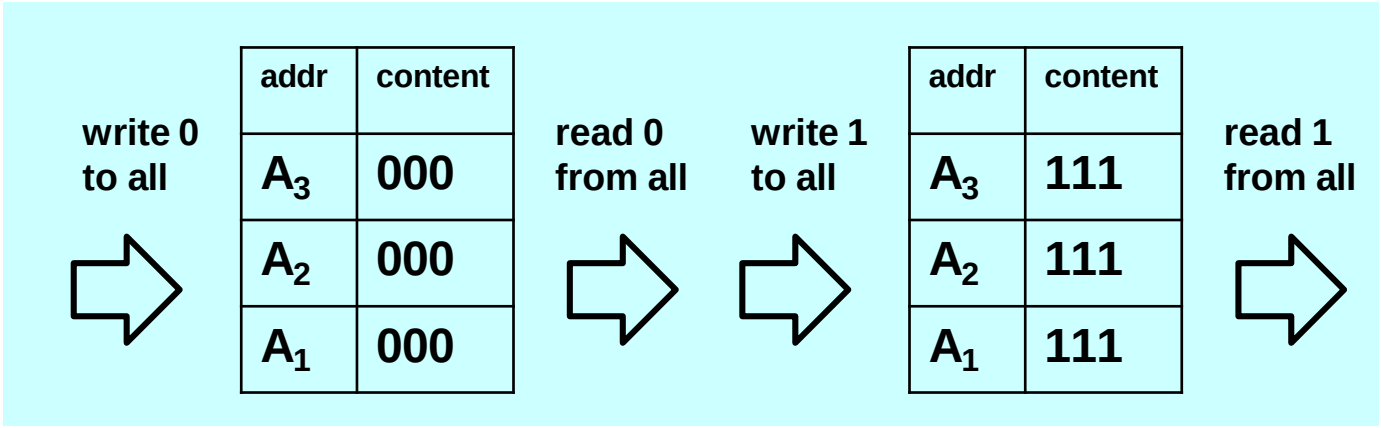
GALPAT-DIAGONAL

1. Write background 0 to all cells;
2. For BaseCell = 0 to N-1
   // BC must be on diagonal line
   Complement BC;
   For OtherCell = 0 to N-1, OC != BC;
   Read BC; Read OC;
   Complement BC;
3. Write background 1 to all cells;
4. Repeat Step 2;

MSCAN [Breuer & Friedman 1976]

- aka. *zero-one algorithm*
- Detects all SAF
- Detects <↑/0> TF, not <↓/1>
- Does NOT detect all AF, CF
- Complexity is *4N*

MSCAN
1. Write zero to every cell
2. Read zero from every cell
3. Write one to every cell
4. Read one from every cell

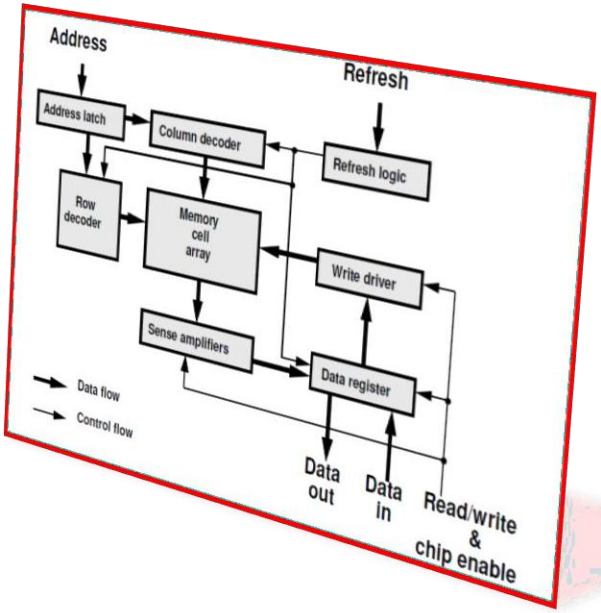


	SAF	AF	TF	CF	Complexity
MSCAN	D	-	1/2	-	4N

D: all detected
- : not all detected

Memory Testing

- Introduction
- Memory Fault Model
- **Memory Test Algorithms**
  - Classical algorithms
  - **March algorithms**
    - **MATS (1979), MATS+(1983), MATS++(1991)**
    - **March X**
    - **March C (1982), March C-(1991)**
- Memory Fault Simulation (*not in exam)
- Memory Test Generation (*not in exam)
- Memory BIST (*not in exam)



March Test Algorithms

- *March test algorithm* has a sequence of *march elements*
- Each *march element* is specified by
  1. *Operations* and data :
     - Reading an expected 0 (r0); reading an expected 1 (r1)
     - Writing 0 to a cell (w0); writing 1 to a cell (w1)
  2. *Address sequence* :
     - ⇑: address changes in ascending order
     - ⇓ : address changes in descending order
     - □: address sequence either ⇑ or ⇓
- Example: {□ (*w*0); ⇑(*r*0,*w*1); ⇓ (*r*1)}
  - has 3 march elements, separated by ;
  - complexity = 4*N*

March Tests are Linear Time

QUIZ

Q: {□ w0); ⇑(r0, w1);⇑(r1, w0); □(r0); ⇓(r0, w1); ⇓(r1, w0); □(r0)}

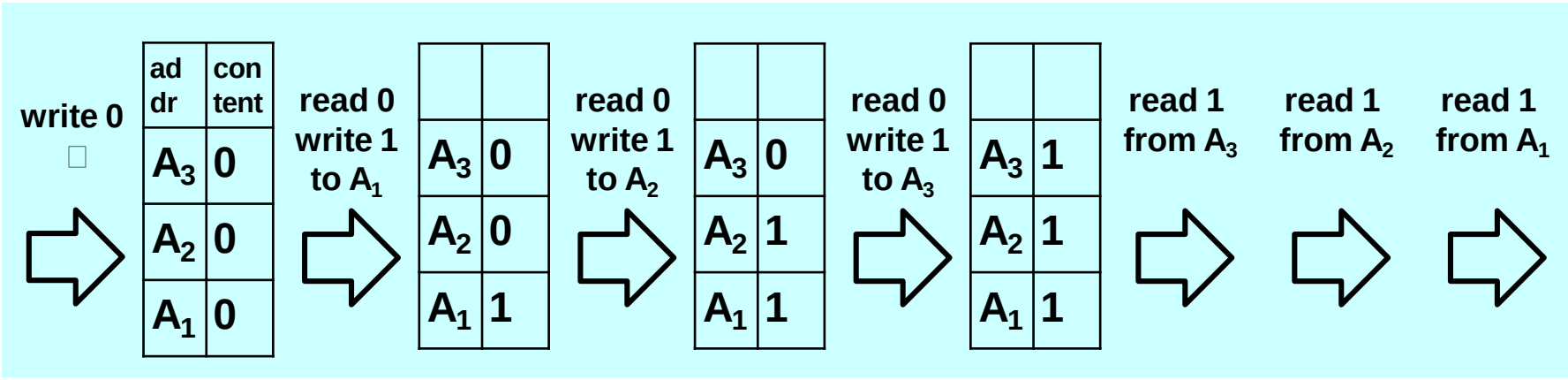
How many march elements? Time complexity=?

ANS:

MATS [Nair 1979]

- *Modified Algorithmic Test Sequence* (MATS)
- 3 march elements
  - {□ (*w0*); ⇑(*r0,w1*); ⇓ (*r1*)}
- Detects all SAF, half TF
- Complexity *4N*

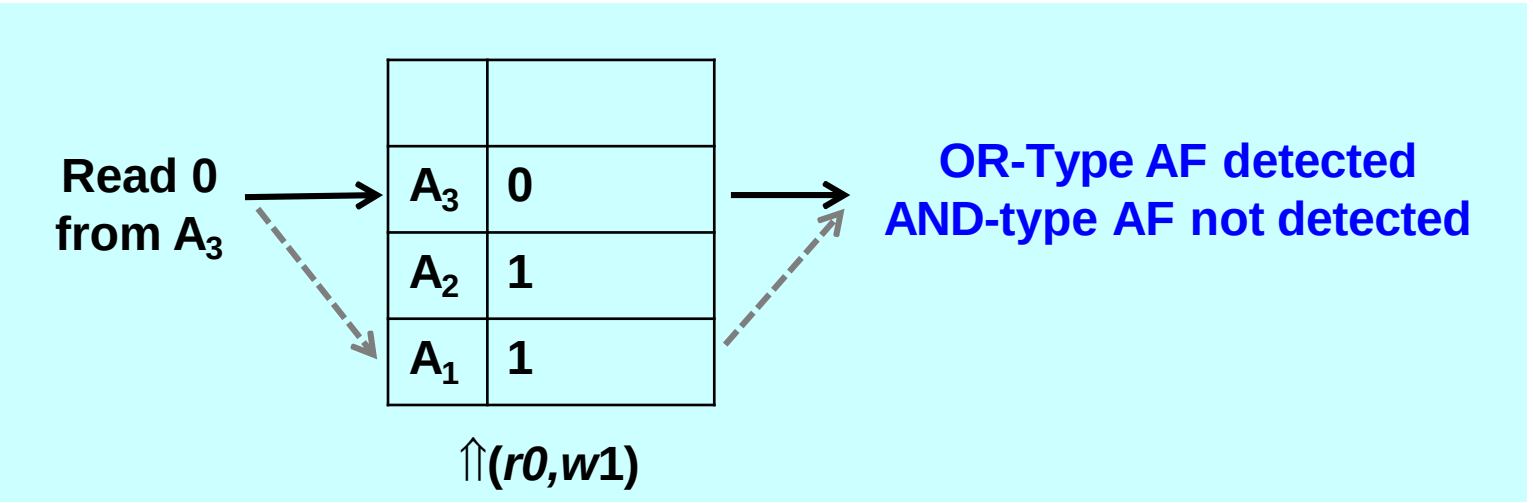
MATS
1. **Write zero** to all cells
   in ascending or descending address order
2. **Read zero** and then **write one**
   in ascending address order
3. **Read one** in descending address order



How about AF?

MATS (2)

- **Can MATS algorithm detect AF?**
  - ***{⇕(w0); ⇑(r0,w1); ⇓(r1)}***
- **Example: OR-type AF between A_3 and A_1**



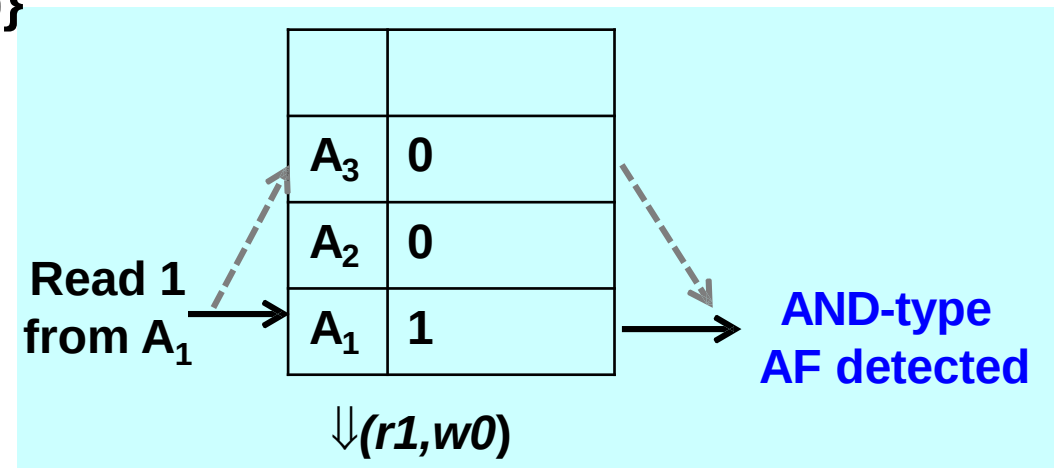
- **How to fix it? Reverse MATS. Detects AND-type but not OR-type**
  - ***{⇕(w1); ⇑(r1,w0); ⇓(r0)}***

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

D=all detected
½=half detected
- = not detected

MATS+ [Abdir 1983]

- OR-type MATS
  - {□ (*w0*); □(*r0*,*w1*); □(*r1*)}
- AND-type MATS
  - {□ (*w1*); □(*r1*,*w0*); □(*r0*)}
- **MATS+ combines both AND-type OR-type MATS**
  - **{□ (*w0*); ⇑(*r0*,*w1*); ⇓(*r1*,*w0*)}**
- **Detects all SAF and AF**
- **Detect half TF**
- **Complexity 5*N***



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

MATS++ [Goor 1991]

- Original MATS+ {□ (*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0)}
- MATS++ Algorithm {□ (*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0)}
- Detects all SAF, AF and TF
  - Similar to MATS+, but detects all TF
- Complexity 6*N*

	SAF	AF	TF	CF	Complexity
MATS++	D	D	D	-	6*N*

How about CF?

QUIZ

Q: Can MATS++ {□ (*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0)} detect CF_{in} <↓; ∀/ □> ?

A1 is aggressor, A3 is victim

ANS:

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

□(*w*0); ⇑(*r*0,*w*1);

A_3 (V)	0
A_2	0
A_1 (A)	1

⇓(r1,w0, r0)

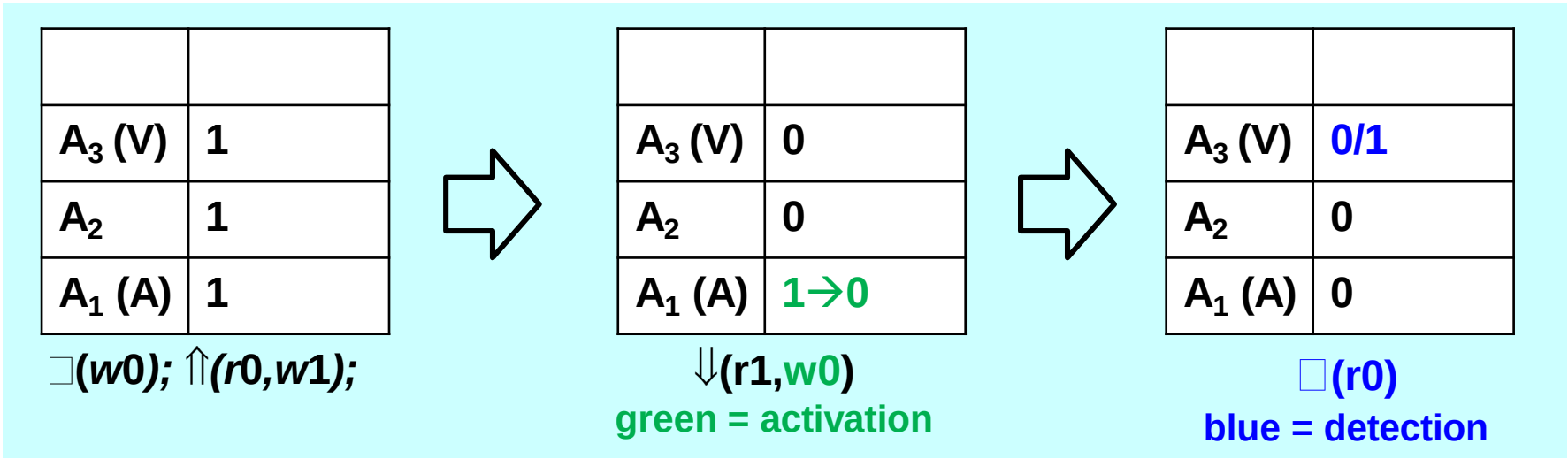
A_3 (V)	?
A_2	?
A_1 (A)	?

NOT detected

March X

*Called *March X* because it's not published

- 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)



March X (cont'd)

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

<↑ ; ∀/ □>

V	0/1
A_2	0
A	0→1

⇑(r0,w1)

<↑ ; ∀/ □>

A	0→1
A_2	1
V	1/0

⇑(r0,w1); ⇓(r1,w0)

< ↓; ∀/ □>

A	1→0
A_2	1
V	1/0

⇓(r1,w0)

<↓; ∀/ □>

V	0/1
A_2	0
A	1→0

⇓(r1,w0);□ (r0)

green = activation
blue = detection

	SAF	AF	TF	CF	Complexity
March X	D	D	D	CF_{in}	6*N*

March C [Marinescu 1982]

- 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

<↑ ; 0/1 >	
V	0/1
A_2	0
A	0→1

⇑(r0,w1)

<↑ ; 1/0 >	
V	1/0
A_2	1
A	0→1

⇓(r0,w1) ; ⇓(r1, w0)

< ↓; 1/0 >	
V	1/0
A_2	1
A	1→0

⇑(r1,w0)

<↓; 0/1 >	
V	0/1
A_2	0
A	1→0

⇓(r1,w0) ; □(r0)

QUIZ

- Q: Can March C detects CF_{st} ?
  - {□ w0); ⇑(r0, w1); ⇑(r1, w0); □(r0); ⇓(r0, w1); ⇓(r1, w0); □(r0)}
  - consider A_1 = Aggressor and A_3 = Victim

ANS:

<1 ; 0/1 >

V	?
A_2	?
A	?

⇑(r0,w1)

<1 ; 1/0 >

V	?
A_2	?
A	?

⇓(r0,w1) ; ⇓(r1, w0)

< 0; 1/0 >

V	?
A_2	?
A	?

⇑(r1,w0)

<0; 0/1 >

V	?
A_2	?
A	?

⇑(r1,w0) ; □(r0)

March C Detects All CF

March C- [Goor 1991]

- March C
  - {□ w0); ⇑(r0, w1);⇑(r1, w0); □(r0); ⇓(r0, w1); ⇓(r1, w0); □(r0)}
- March C- remove redundancy in March C
  - {□ (w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0); □(r0)}

- March C- detects AF, SAF, TF, & all CF
- Complexity 10*N*
  - Shortest test that detect all four faults

	SAF	AF	TF	CF	Complexity
March C	D	D	D	D	11 *N*
March C-	D	D	D	D	10 *N*

Summary

- March Tests contains march elements
  - Operations (R/W)
  - Address order
- March tests
  - Linear time
  - Good FC
- March C-
  - 10 *N*
  - Detects 4 faults

	SAF	AF	TF	CF	Complexity
MSCAN	D	-	-	-	$4N$
checkerboard	D	-	-	-	$4N$
GALPAT	D	D	D	D	$4N^2$
BUTTERFLY	D	-	D	-	$5N \log N$
MATS	D	-	-	-	$4N$
MATS+	D	D	-	-	$5N$
MATS++	D	D	D	-	$6N$
March X	D	D	D	-	$6N$
March C	D	D	D	D	$11N$
March C-	D	D	D	D	$10N$

March Tests Very Useful in Practice

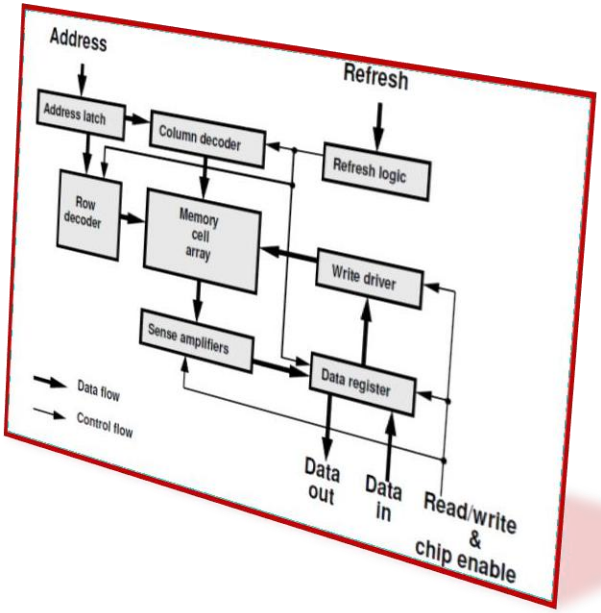
FFT

- Q: Even with linear algorithm, 28 minutes is still too long.
  - how to reduce it?

Size	N	$10N$	$N \lg N$	$N^{1.5}$	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	59c
16G	2.8m	28m	1.6h	261d	936c

Outline

- Introduction
- Memory Fault Model
- Memory Test Algorithms
- Memory Fault Simulation (*not in exam)
- Memory Test Generation (*not in exam)
- Memory BIST (*not in exam)



Memory Fault Simulation

- What is *memory fault simulation*?
  - Given test algorithm, memory architecture
    - find fault coverage of different fault models

- Why fault simulation?
  - Evaluate fault coverage efficiently,
    - especially when many fault models
  - Help test algorithm design and optimization
  - Fault dictionary can be constructed for easy diagnosis

Sequential Memory Fault Simulation

```
For each fault    /* N² for 2-cell CF */
Inject fault;
For each test element    /* N for March */
            {
              Apply test element;
              Report error output
            }
```

- Complexity is N^3 for 2-cell CF
  - This is very slow

Parallel Fault Simulation [Wu 02]

- **Random Access Memory Simulator for Error Screening (RAMSES)**
- **Consider all faults at a time**
- **Complexity N^2**

```
# 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
```

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;

}

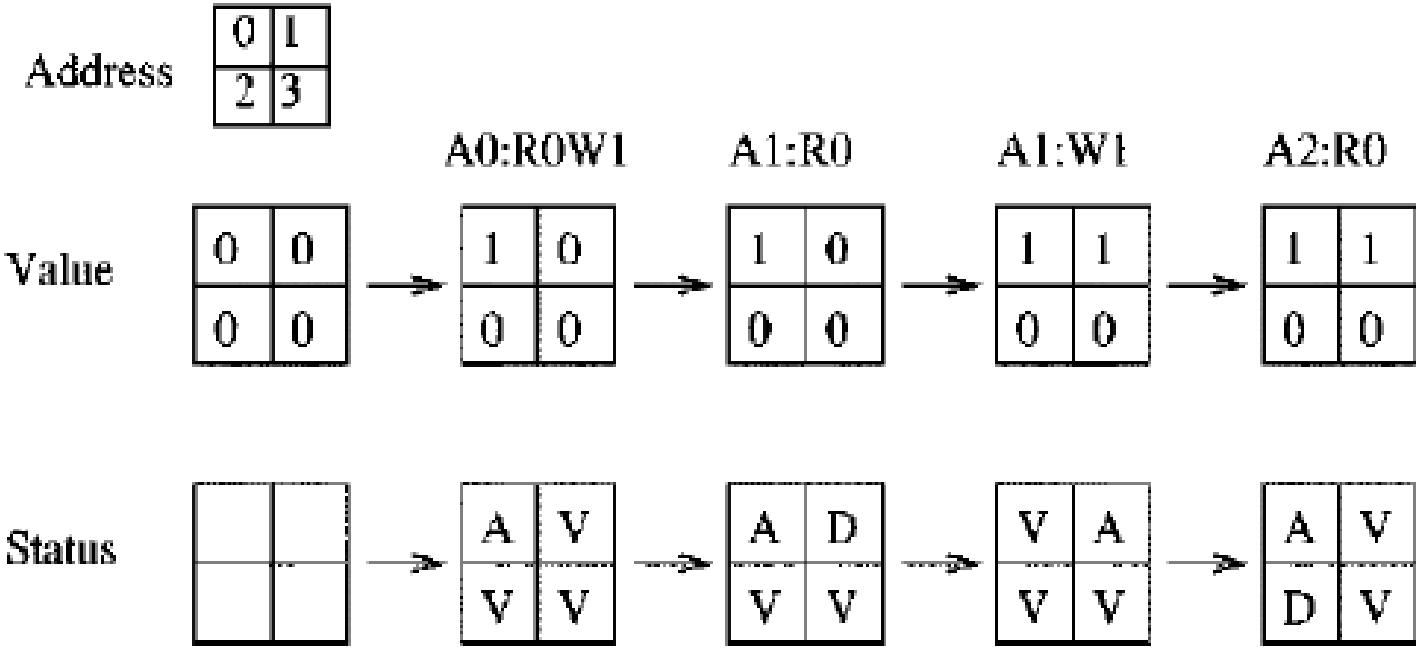
AGR=aggressor
SPT=suspects
VTM=victim
RCV=recover
@=this cell
***=all cells**

RAMSES Algorithm

```
For each fault opeartion
  set op_flags;
  if (ARG ⊂ op_flags)  {
     for each victim cell {
        set victim flags;
        set aggressor address;
     }
  }
  if (OP eq RCV)  {
     clear victim flag;
     clear aggressor entry;
  }
  else if (OP eq VTM) {
      mark detected;
  }
}
```

RAMESE Example

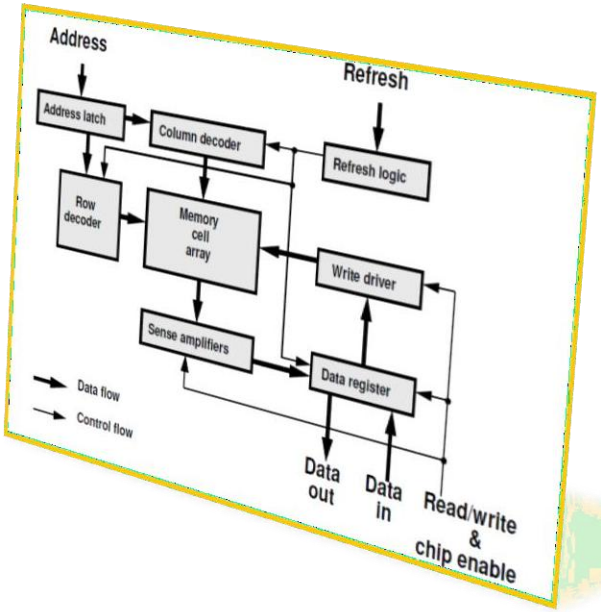
- CF_{in} <↑ / ↕ >
- March element ⇑(r0, w1)



D = detect; A = aggressor; V=victim

Outline

- Introduction
- Memory Fault Model
- Memory Test Algorithms
- Memory Fault Simulation
- Memory Test Generation (* not in exam)
- Memory BIST



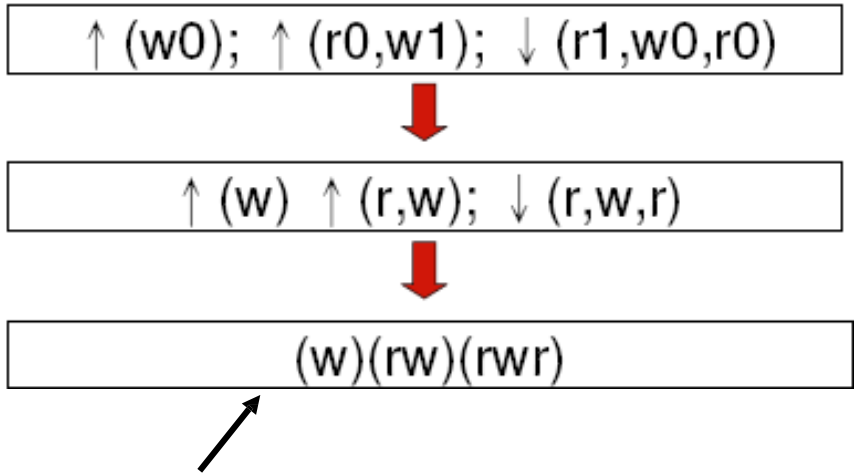
Memory Test Generation

- What is memory test algorithm generation
  - 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

TAGS [Wu 00]

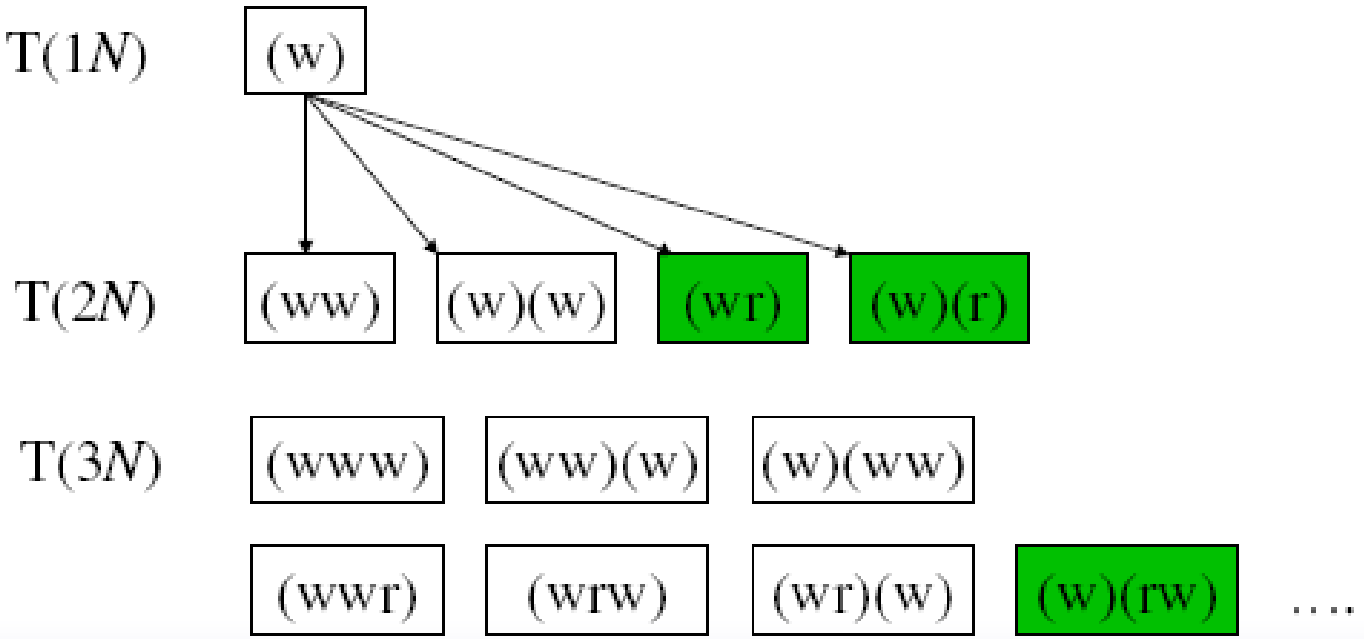
- **Test Algorithm Generation by Simulation (TAGS)**

- March template abstraction:



Template Set

- ***Template set***
  - **T(*cN*) = a set of march templates of *c* read or write**
- **Exhaustive generation of template set is very expensive**
  - **e.g., 6.7 million templates when *N* = 9**
- **Heuristics should be developed to select useful templates**



TAGS Procedure

- 1. Initialize test length as 1*N*, T(1*N*) = {(w)};
- 2. Increase test length by 1*N*: apply generation options;
- 3. Apply *filtering* heuristics;
- 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 and 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)**

3*N* March Test

- 3*N* March Test generated by TAGS after step 3
  - WWW Table 8.7

	Test
1	⇑ (w0) ⇑(w1, r1)
2	⇑(w0) ⇓(w1, r1)
3	⇑(w0) ⇑(w0, r0)
4	⇑(w0) ⇓(w0, r0)
5	⇑(w0) ⇑(r0, w1)
6	⇑(w0) ⇓(r0, w1)
7	⇑(w0) ⇑(r0) ⇑(r0)
8	⇑(w0) ⇑(w1) ⇑(r1)

- 3N March test selected by RAMSES
  - WWW Table 8.8

	Test
3	⇑(w0) ⇑(w0, r0)
5	⇑(w0) ⇑(r0, w1)
8	⇑(w0) ⇑(w1) ⇑(r1)

TAGS Examples (1/2)

- **WWW Table 8.9**

$T(N)$	Name	March algorithm
$1N$	$MATS$	$\Uparrow (w0)$
$2N$	$MATS+$	$\Uparrow (w0) \Uparrow (r0)$
$3N$	$MATS++$	$\Uparrow (w0) \Uparrow (w1) \Uparrow (r1)$
$3N$	$MarchX$	$\Uparrow (w0) \Uparrow (r0, w1)$
$3N$	$MarchY$	$\Uparrow (w0) \Downarrow (w1) \Uparrow (r1)$
$3N$	$MarchC-$	$\Uparrow (w0) \Downarrow (r0, w1)$
$4N$	$PMOVI$	$\Uparrow (w0) \Downarrow (r0, w1) \Uparrow (r1)$
$4N$	$MarchA$	$\Uparrow (w0) \Downarrow (r0, w1, r1)$
$5N$	$MarchB$	$\Uparrow (w0) \Uparrow (w1) \Uparrow (r1, w0) \Uparrow (r0)$
$5N$	$MarchC$	$\Uparrow (w0) \Downarrow (r0, w1) \Uparrow (r1, w0)$
$5N$	$MarchU$	$\Uparrow (w0) \Uparrow (w1) \Uparrow (r1, w0, r0)$
$6N$	$MarchLR$	$\Uparrow (w0) \Uparrow (w1) \Uparrow (r1, w0) \Downarrow (r0, w1)$
$6N$	$MarchC+$	$\Uparrow (w0) \Downarrow (r0, w1) \Uparrow (r1, w0) \Uparrow (r0)$
$6N$	$MarchA+$	$\Uparrow (w0) \Uparrow (r0, w1) \Uparrow (r1, w0) \Uparrow (r0)$
$6N$	$MarchU+$	$\Uparrow (w0) \Uparrow (r0, w1) \Uparrow (r1, w0, r0)$
$6N$	$MarchY+$	$\Uparrow (w0) \Downarrow (r0, w1) \Uparrow (r1, w0, r0)$
$7N$	$MarchA++$	$\Uparrow (w0) \Uparrow (r0, w1) \Uparrow (r1, w0) \Downarrow (r0, w1)$

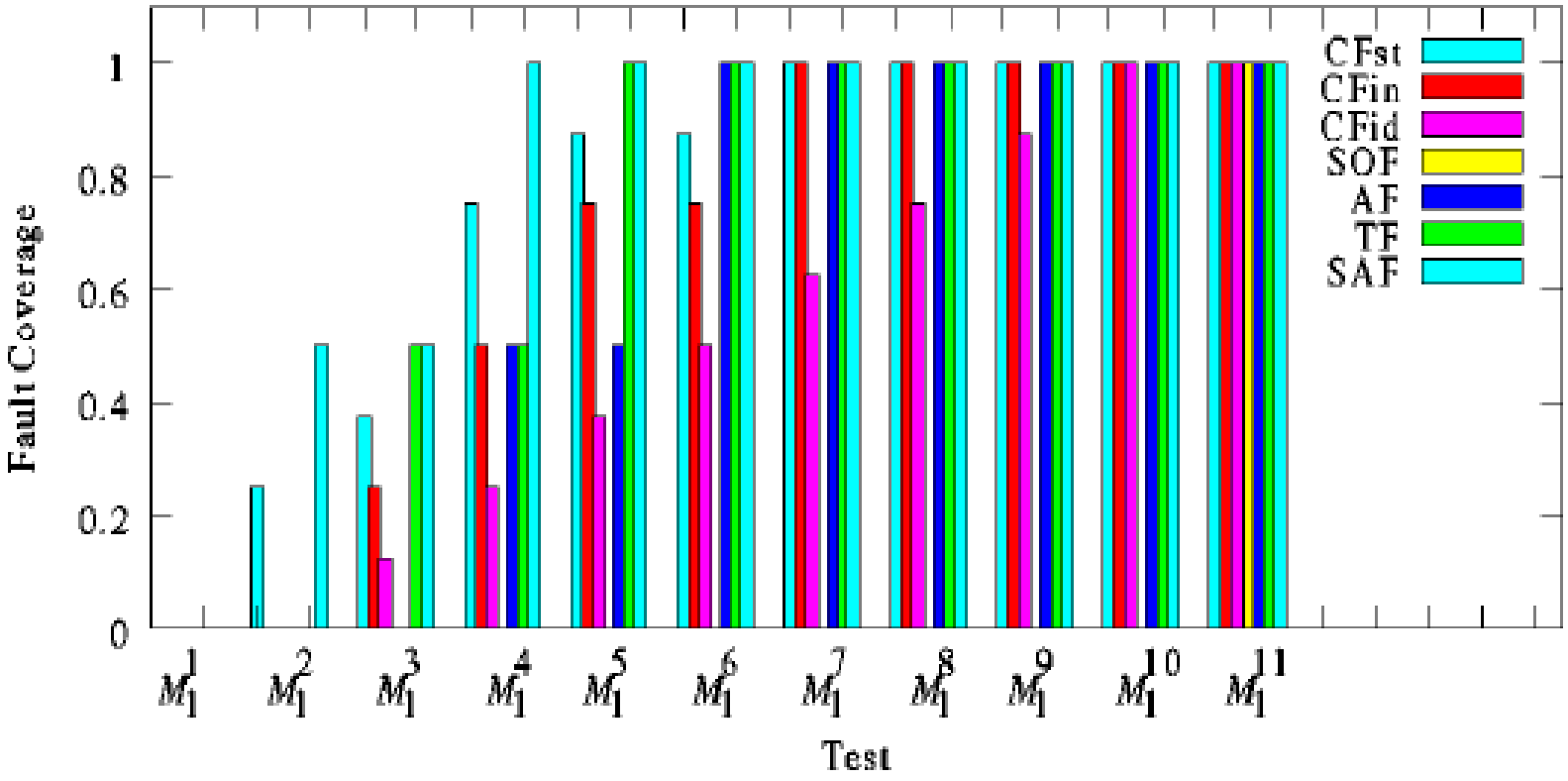
TAGS Examples (2/2)

- **WWW Table 8.9 cont'd**

$7N$	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}	$\Uparrow (w0)\ \Uparrow (r0, w1)\ \Uparrow (r1, w0)\ \Downarrow (r0, w1)$ $\Downarrow (r1, w0, r0)\ \Uparrow (r0)$

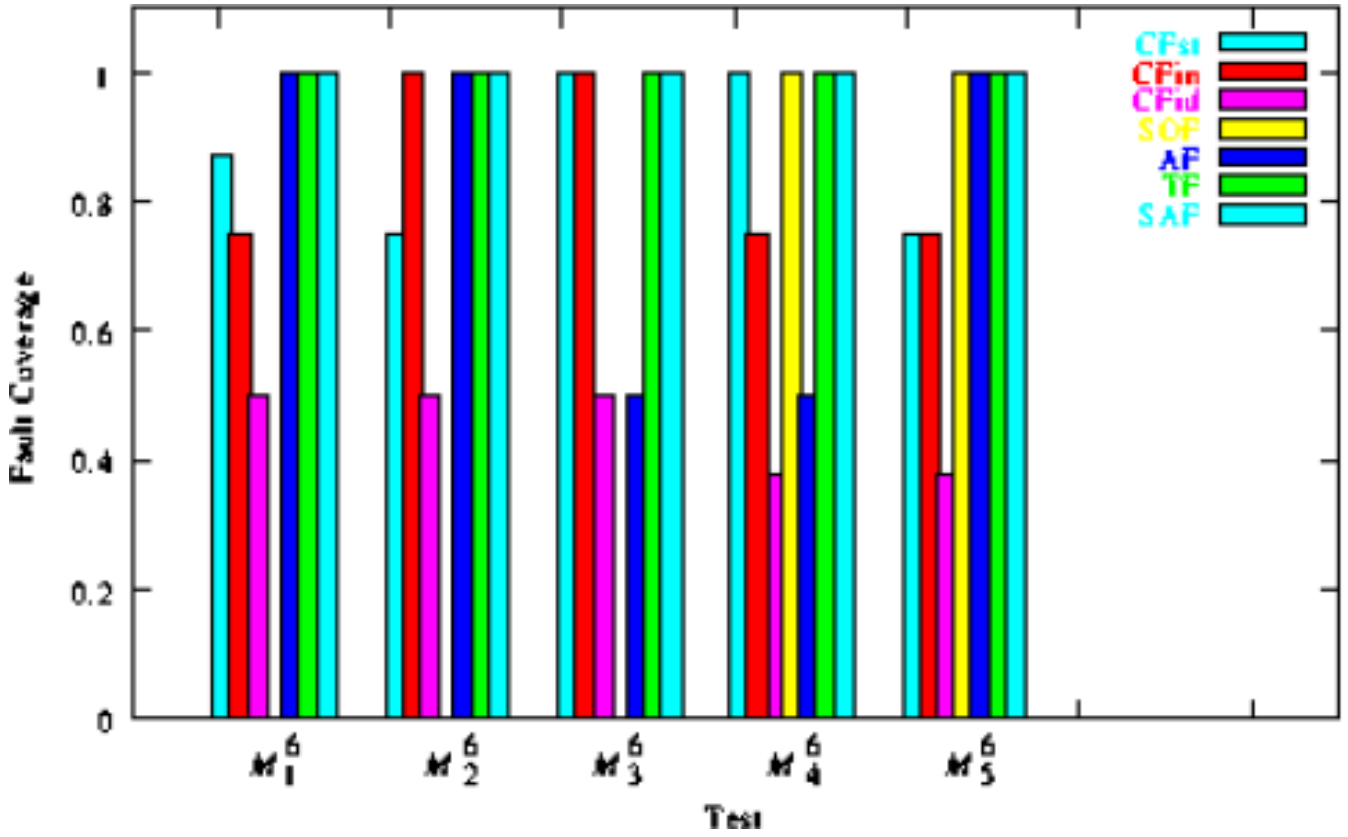
RAMSES Simulation Results

- **WWW Figure 8.13**



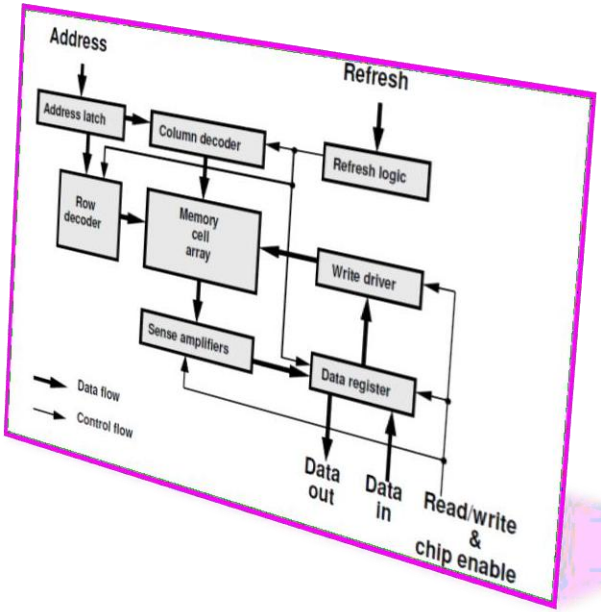
RAMSES Simulation Results

- **WWW Figure 8.14**
  - **Five different 6*N* tests**



Outline

- Introduction
- Memory Fault Model
- Memory Test Algorithms
- Memory Fault Simulation* (not in exam)
- Memory Test Generation* (not in exam)
- Memory BIST* (not in exam)

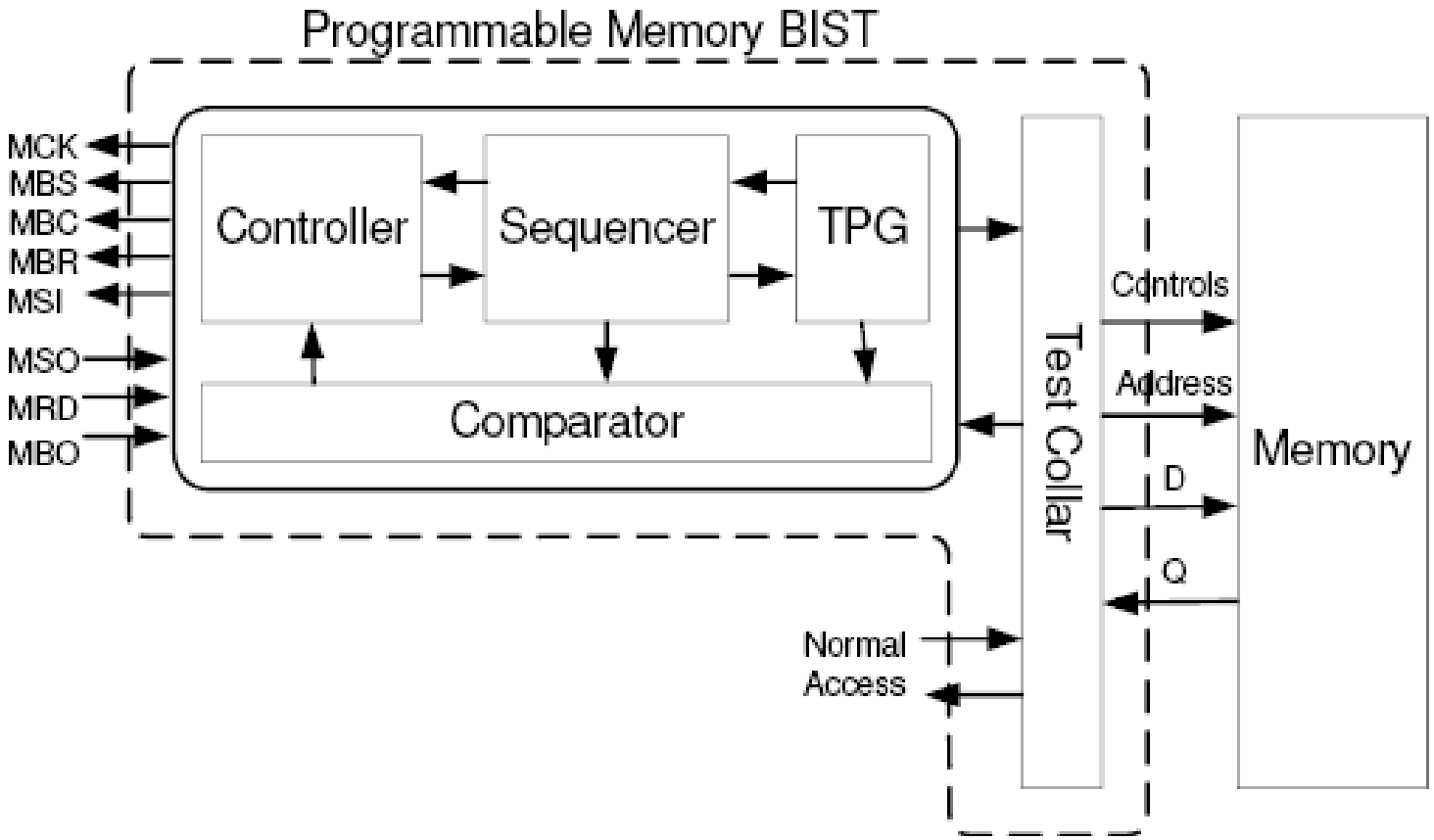


Typical RAM BIST Approaches

- **Methodology (Architecture)**
  - ***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**

General RAM BIST Architectures

- ***Controller*: overall BIST flow**
- ***Sequencer*: address, data and timing sequence**

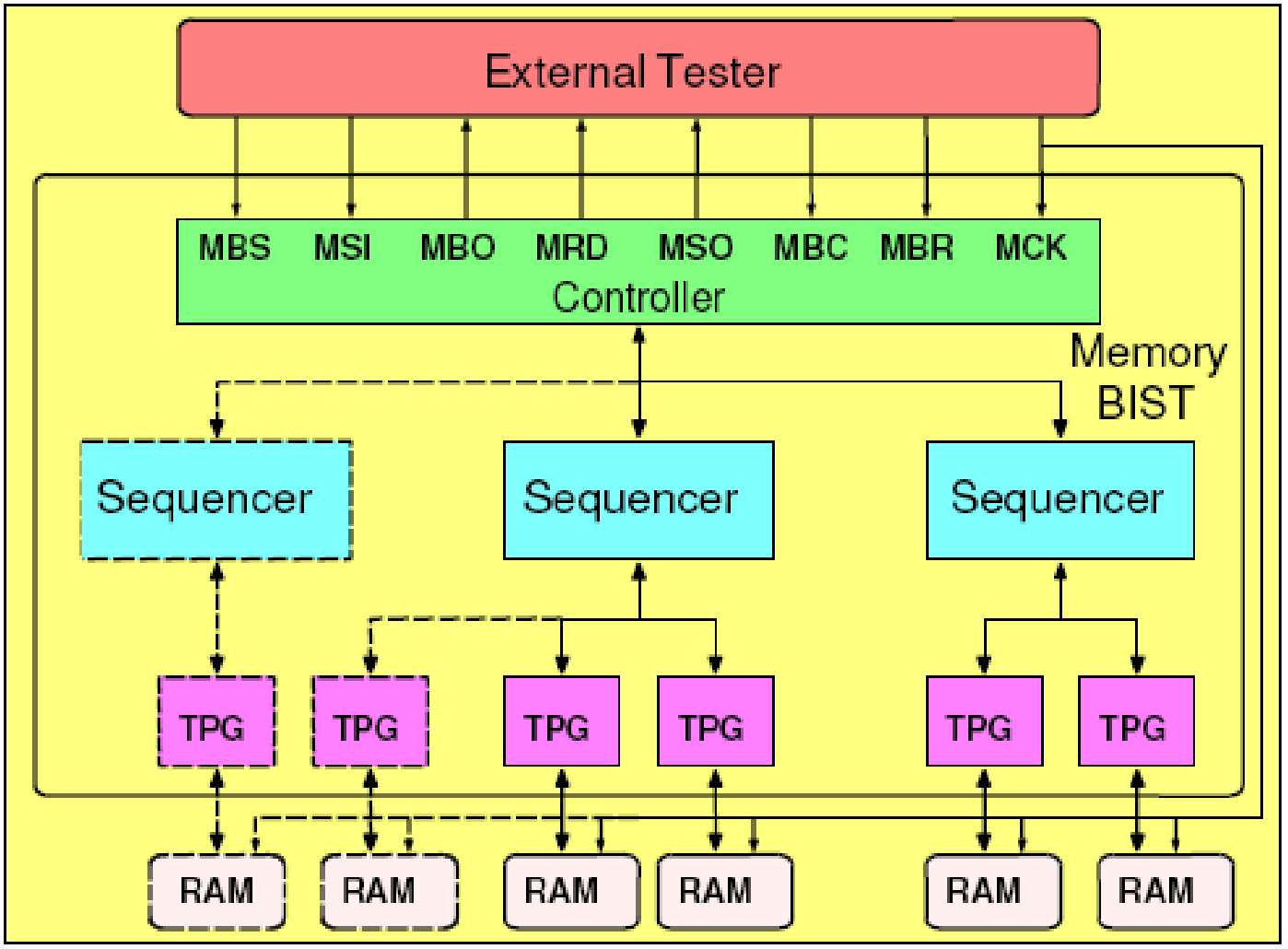


BIST I/O Pins

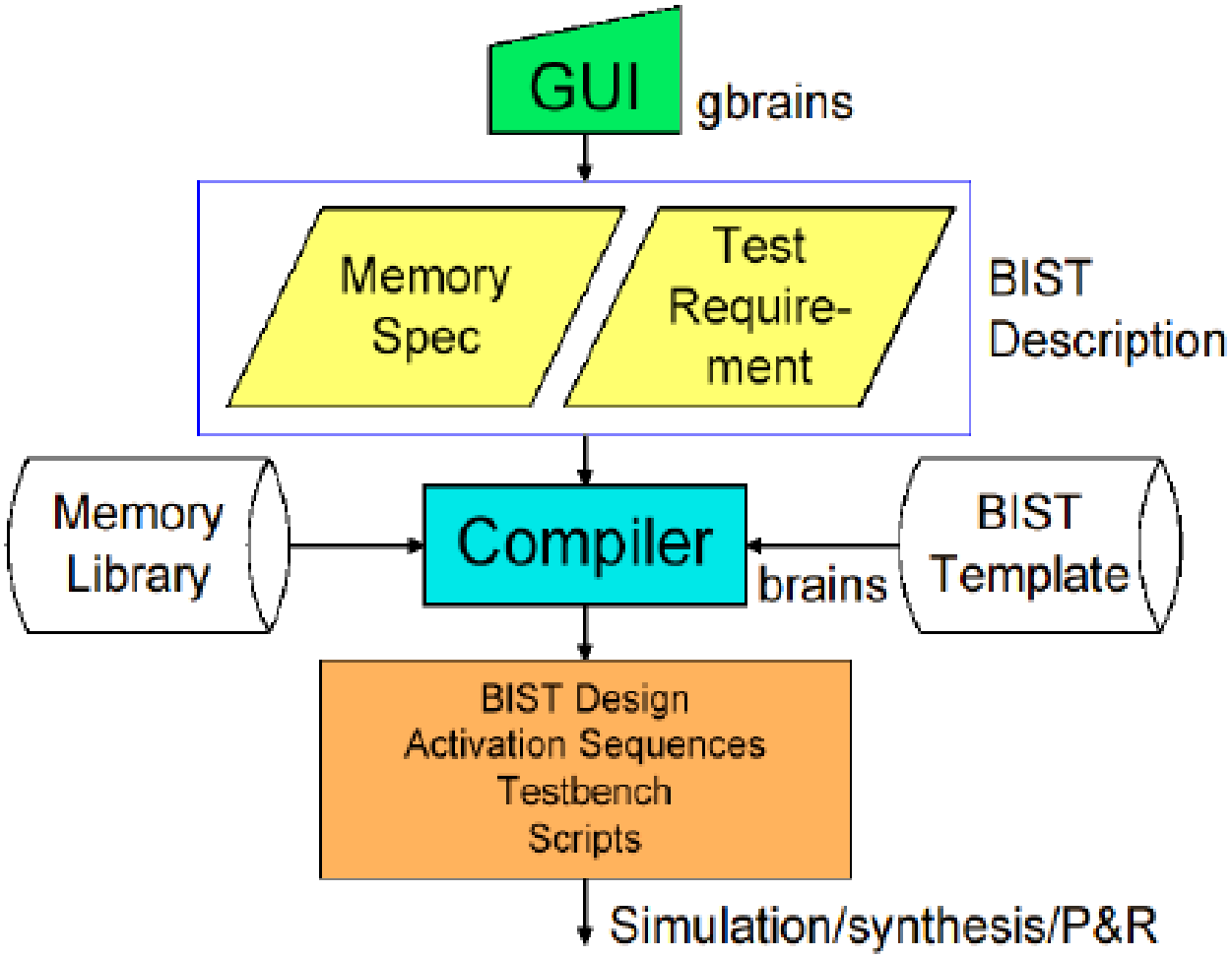
Name	IO	External IO	Descriptions
MBS	I	Yes	Memory BIST Selection
MBC	I	Yes	Memory BIST Control
MCK	I	Yes	Memory BIST Clock
MBR	I	Yes	Memory BIST Reset
MSI	I	Yes	Memory BIST command/data serial in
MSO	O	Yes	Memory BIST command/data serial out
MBO	O	Yes	Memory BIST Output
MRD	O	Yes	Memory BIST Output Ready
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

Multiple RAM Cores

- **Controller and sequencer can be shared**



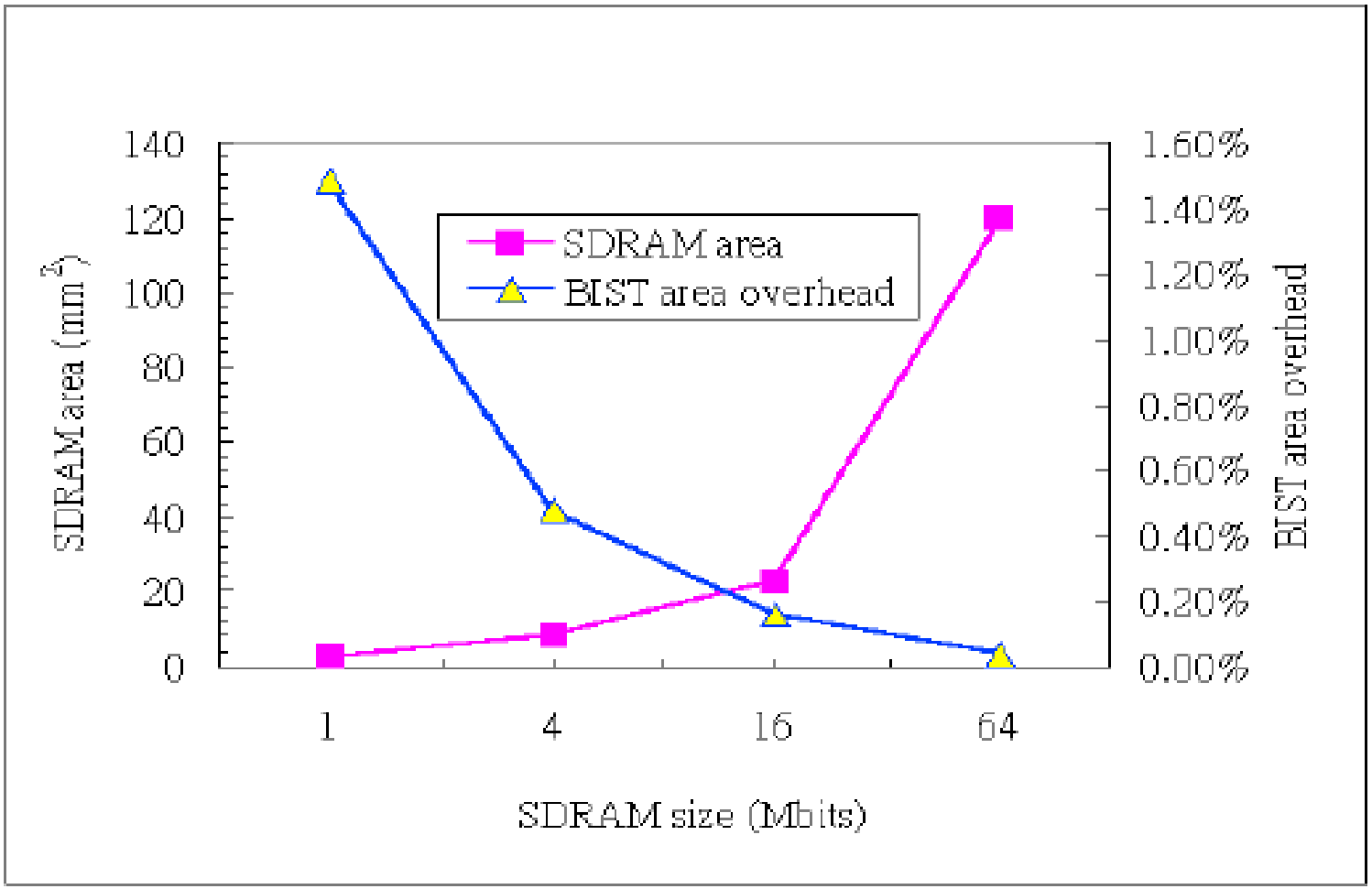
BRAINS Inputs and Outputs [Huang 99]



BRAINS Outputs

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

Area Overhead



Summary

- Memory testing are important
- Different memory requires different fault models
- March C is shortest march that detect all 4 fault models: 10N
- RAM BIST are needed for embedded memories
  - EDA tools available