VLSI Test Technology and Reliability (ET4076)

Lecture 7

Memory Testing

Said Hamdioui

Computer Engineering Lab
Delft University of Technology
2009-2010

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)
Date: 2009-2010

Delft University of Technology

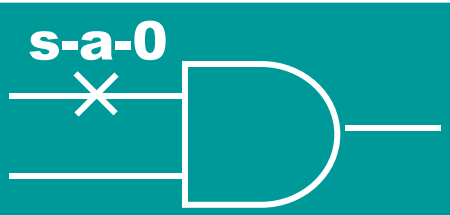
Motivation

- Combinational logic contains no flip flops (no memory)

- Space of possible tests is
  - $O(2^{no\_pi})$, no_pi = #primary inputs
- If no_pi = 64 & circuit runs @ 1GHz
  How long does a full test take??
  → full test takes **585 years!**

- How can we deal with this problem??

Stuck-at faults are used to model defects and limit test space



Motivation

- Sequential logic contains some flip flops (limited memory)
- Space of possible tests is
  - $O(2^{no\_pi} \times 2^{no\_ff})$

    no_pi = #primary inputs

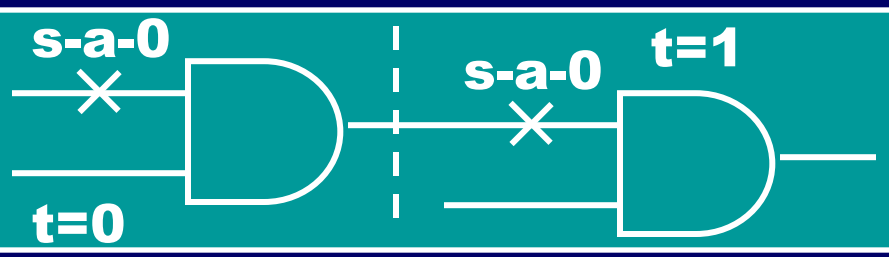
    no_ff = #flip flops
- If no_pi = 64, no_ff = 16 @ 1GHz

  How long should we test??

  ➔ test takes about 50 million years!

- How can we deal with this problem??

Stuck-at faults &
time-frame expansion



Motivation

- Memory contains millions of states
- Testing memory is EXTREMELY difficult
- Exhaustive testing takes more than the age of the UNIVERSE!!
- Special fault models are needed

Learning aims

- Describe the problem of memory testing, its objectives and its importance
- Describe different fault types
- Define the different memory fault models
- Classify the memory test algorithms
- Analyze the fault coverage of memory algorithms
- Develop memory algorithms for given fault models
- Describe the major challenges for memory testing

Contents

1. Basics of memory devices
2. Semiconductor memory architecture
3. Functional fault models
4. Memory test development

   (to detect the memory faults we defined)
5. Well-known memory tests

VLSI Test Technology and Reliability (ET4076)

Memory Testing:
1. Basics of memory devices

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)

Delft University of Technology

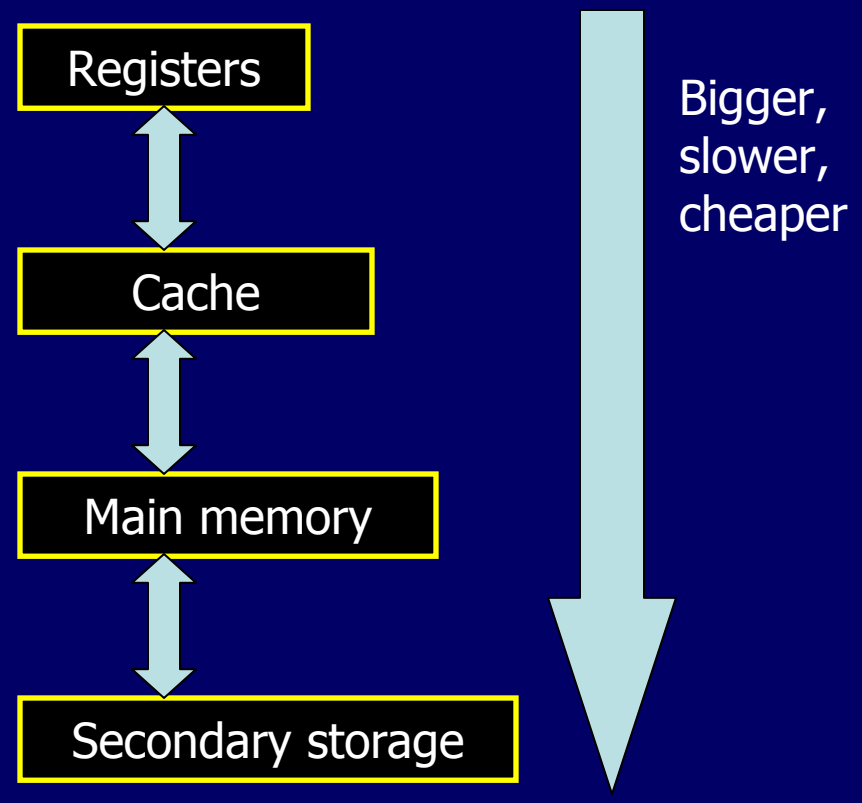
1. Basics of memory devices

Topics:

- History of memory
- History of DRAM
- History of SRAM
- Importance of memory testing
- Test objectives and requirements

1. Basics: history of memory

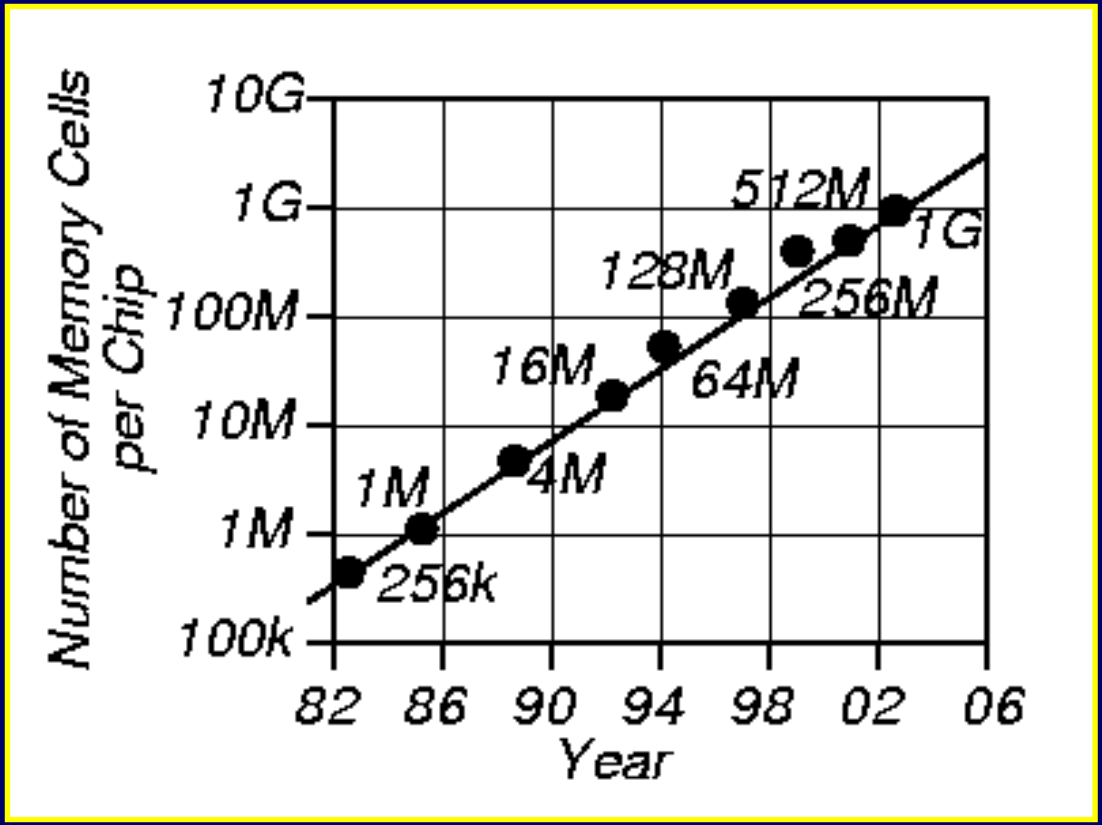
- Memory is important for data processing
- More memory needed for faster CPUs
- Hierarchy of memory used:
  - Registers: Flip flops
  - Cache: SRAM
  - Main: DRAM
  - Secondary: Hard disk



1. Basics: history of DRAM

DRAMs are

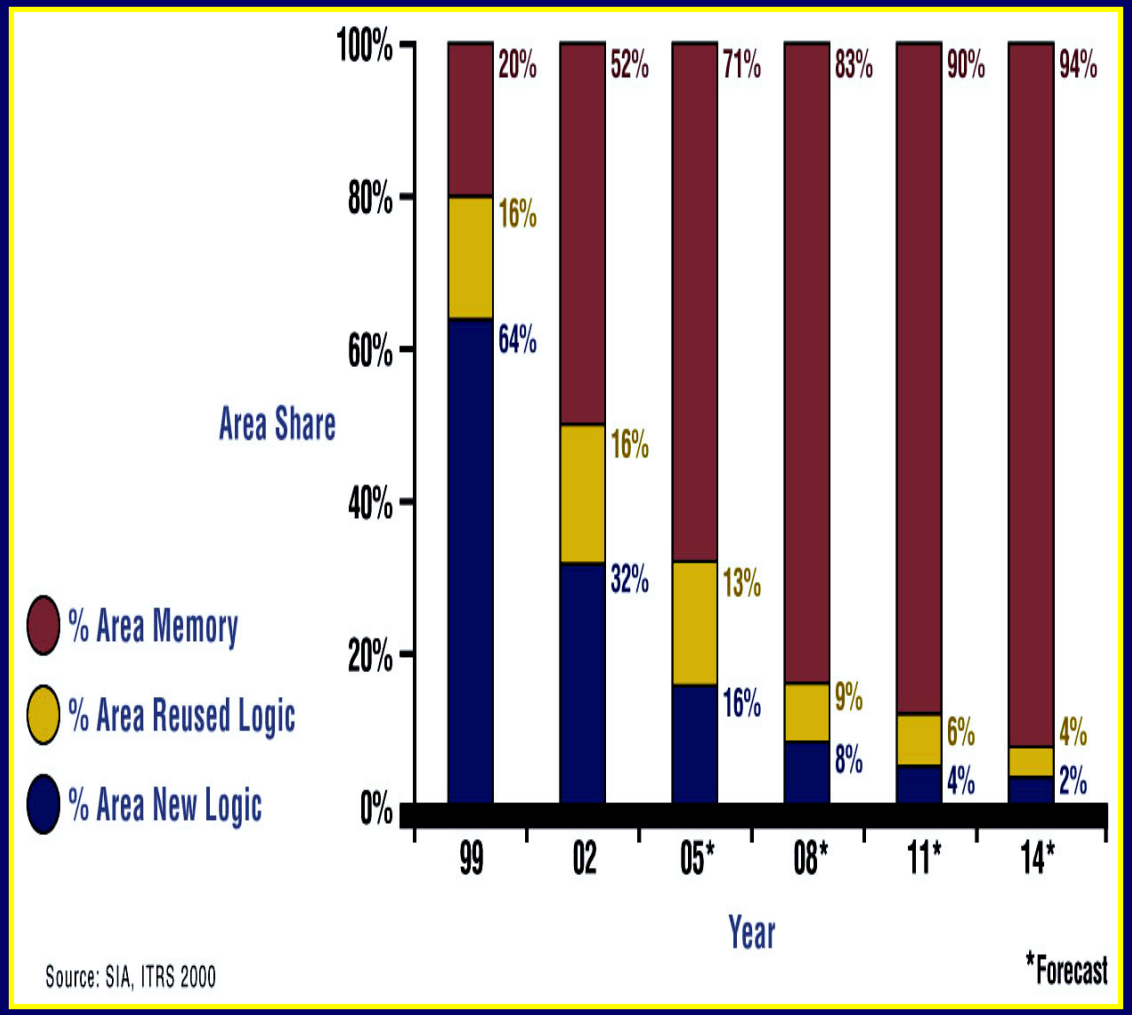
- Widely used
- Standalone chip
- Most sensitive VLSI chip
- Increase exponentially in size
  - 32 GB+ in 2020!



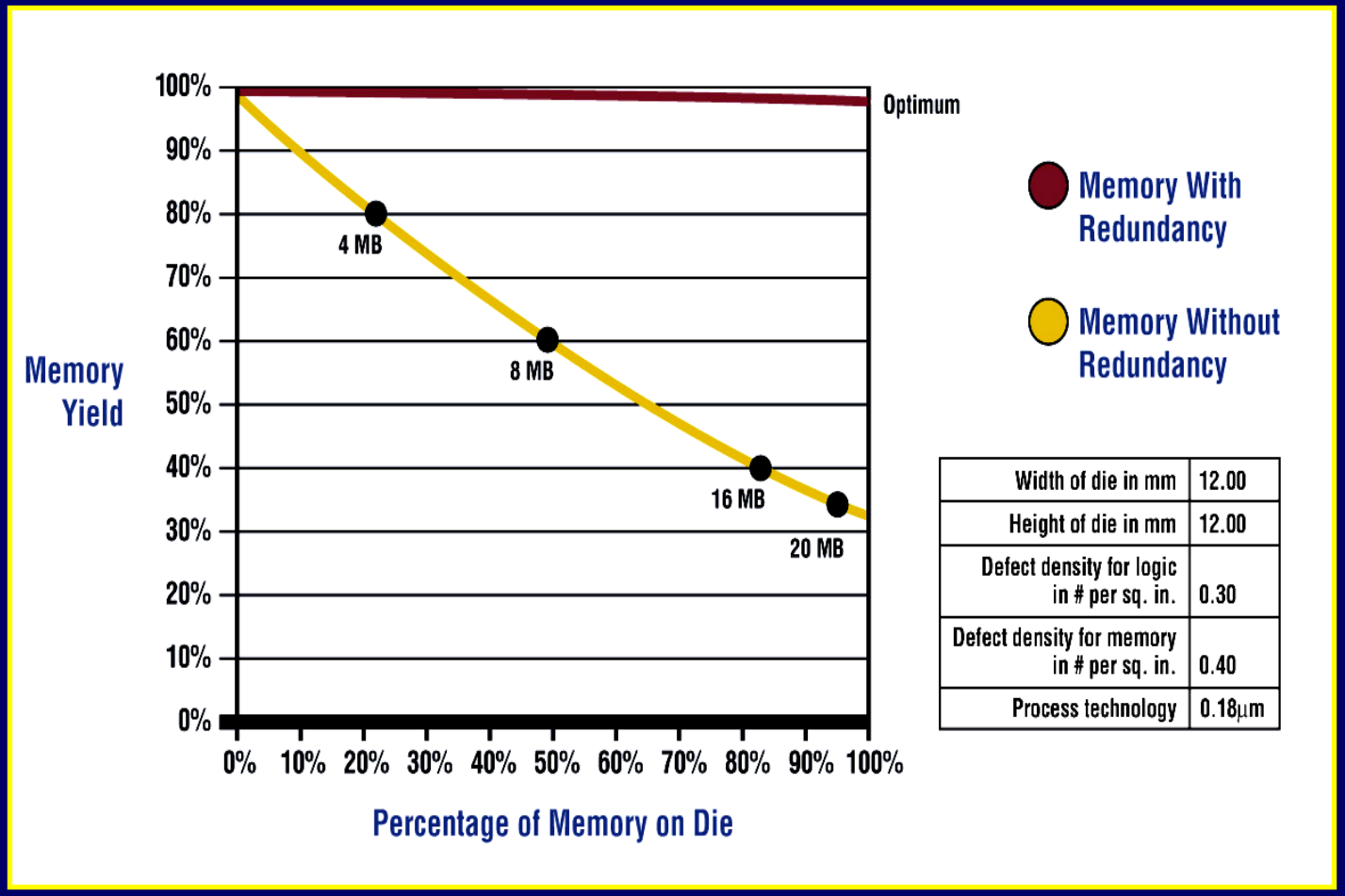
1. Basics: history of SRAM

SRAMs are

- Widely used
- Embedded with logic
- More sensitive than logic
- Increase in size
- Dominate chip area
  - 94% in 2014!



1. Basics: importance of memory testing



Width of die in mm	12.00
Height of die in mm	12.00
Defect density for logic in # per sq. in.	0.30
Defect density for memory in # per sq. in.	0.40
Process technology	0.18μm

- Yield decreases dramatically with increasing in memory size
- Redundancy dramatically improves memory yield

1. Basics: test objectives & requirements

- Requirements to detect faults

High fault coverage

Tests required!

High quality

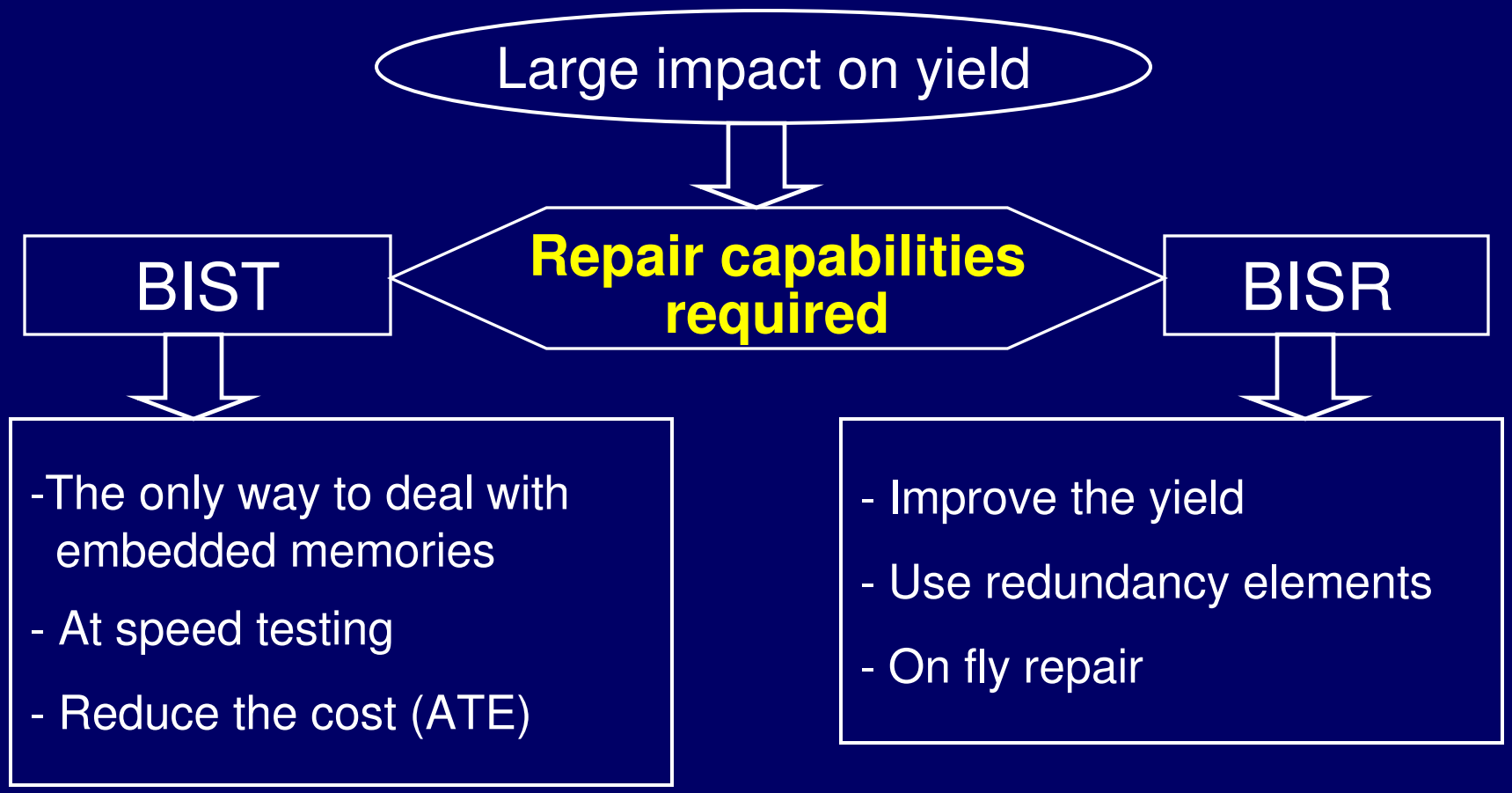
- Realistic fault models
- Memory-specific
- Appropriate stresses

Low cost

T=10 ns	16 Kbits	32 Kbits	64 Kbits
O(n) [10n]	**1.64 ms**	**3.28 ms**	**6.56 ms**
$O(n^2)$ [$10n^2$]	26.84 **s**	107.37 s	7.16 min
$O(n^3)$ [$10n^3$]	122.2 **h**	977.3 h	?????

1. Basics: test objectives & requirements

- Requirements to increase yield



VLSI Test Technology and Reliability (ET4076)

Memory Testing:
2. Semiconductor memory architecture

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)

Delft University of Technology

2. Semiconductor memory architecture

Topics:

- Memory models
- Behavioral model
- Functional model
- Electrical model
- Layout model (rarely reported)

2. Architecture: memory models

Increasing level of abstraction

Behavioral

Functional

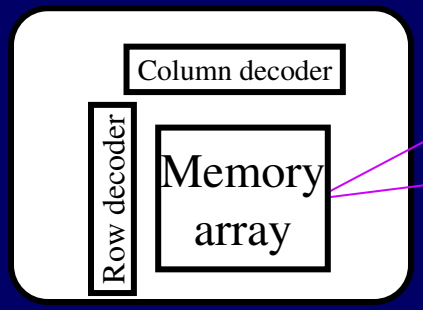
Logical

Electrical

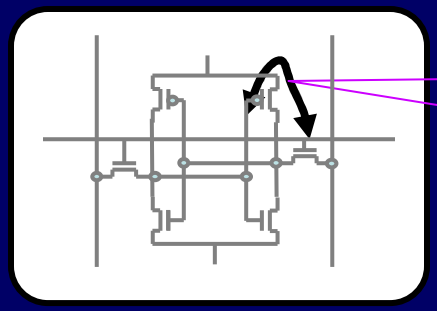
Layout

If (cond) then read (x)

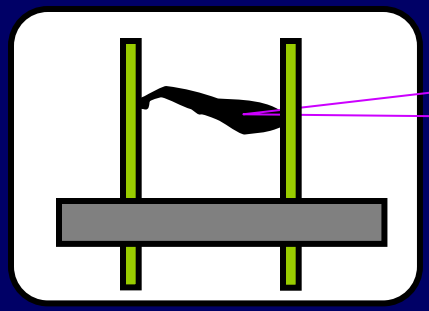
Behavioral fault



Functional fault; <1; 0/1/->



Electrical fault



Defect

2. Architecture: behavioral model

- General memory Model
  - Nothing known about the internal structure
  - Data sheet describing functionality provided

  - B: the word width
  - Data-in and Data-out are usually combined
    => Bi-directional data lines

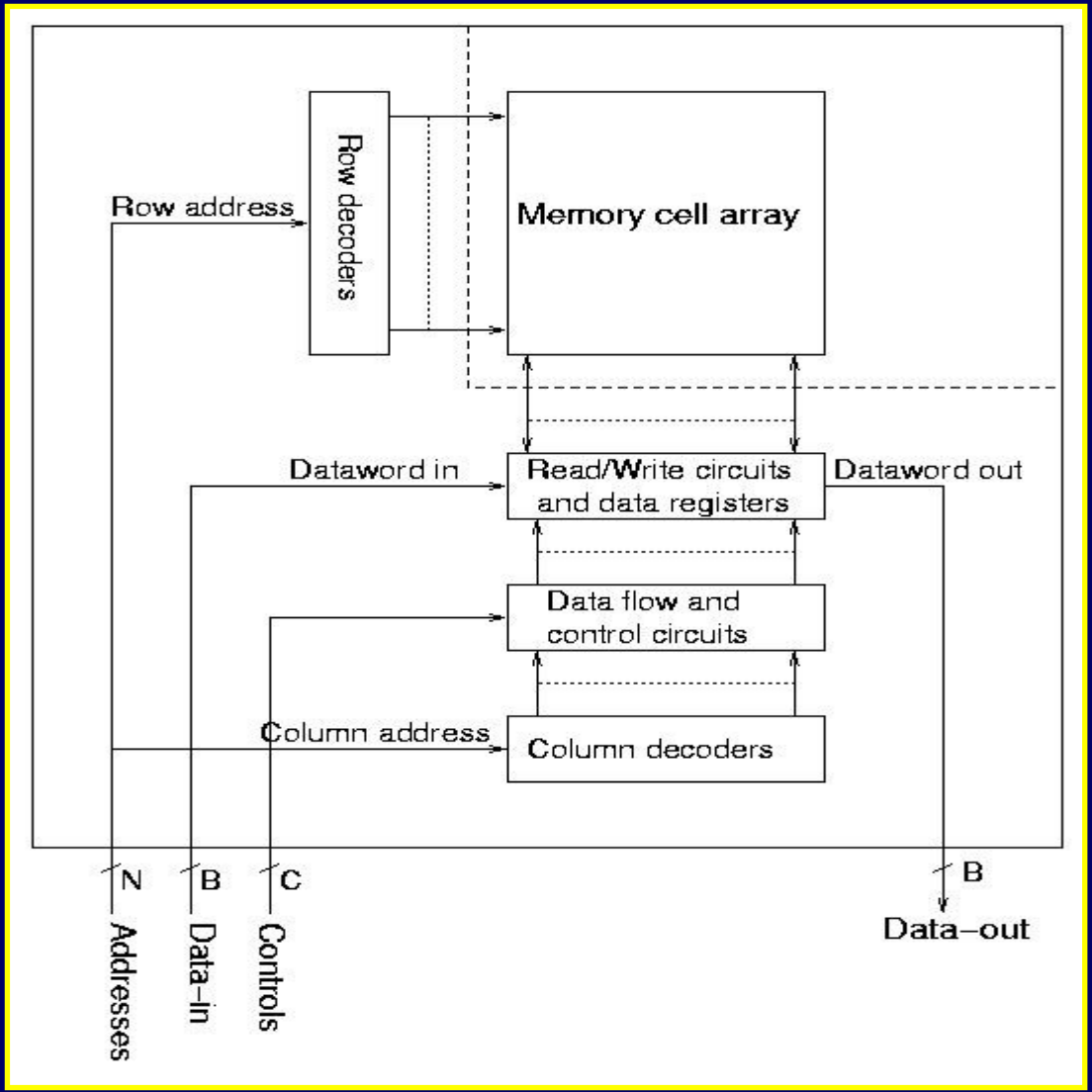
2. Architecture: functional model

- Functions of the system
- functional blocks (with behavior model)
- Memory cell array consist of *n* cells

$n = R \times C$

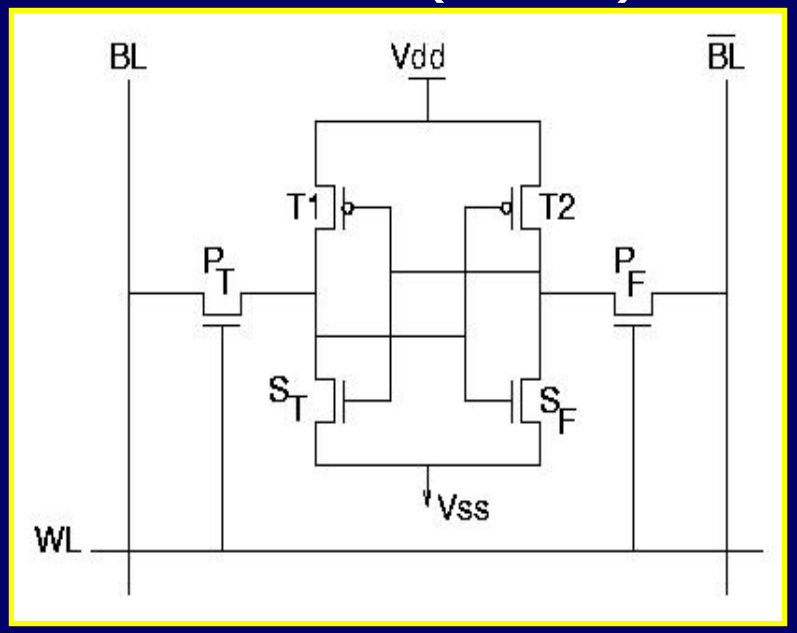
R: Rows (Word lines)

C: Columns (Bit lines)



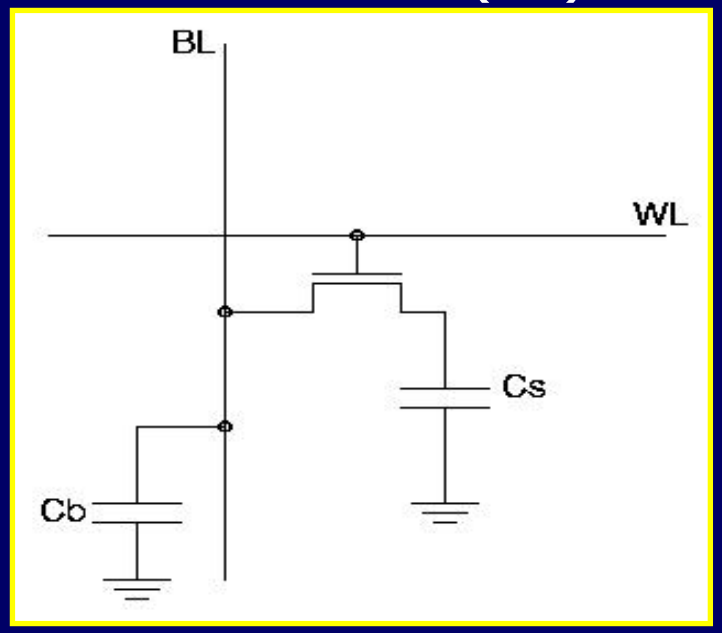
2. Architecture: electrical model (array)

SRAM cell (CMOS)



Six Transistors per bit
Low density
Fast access time (e.g., 2ns)
High prices

DRAM cell (1T)



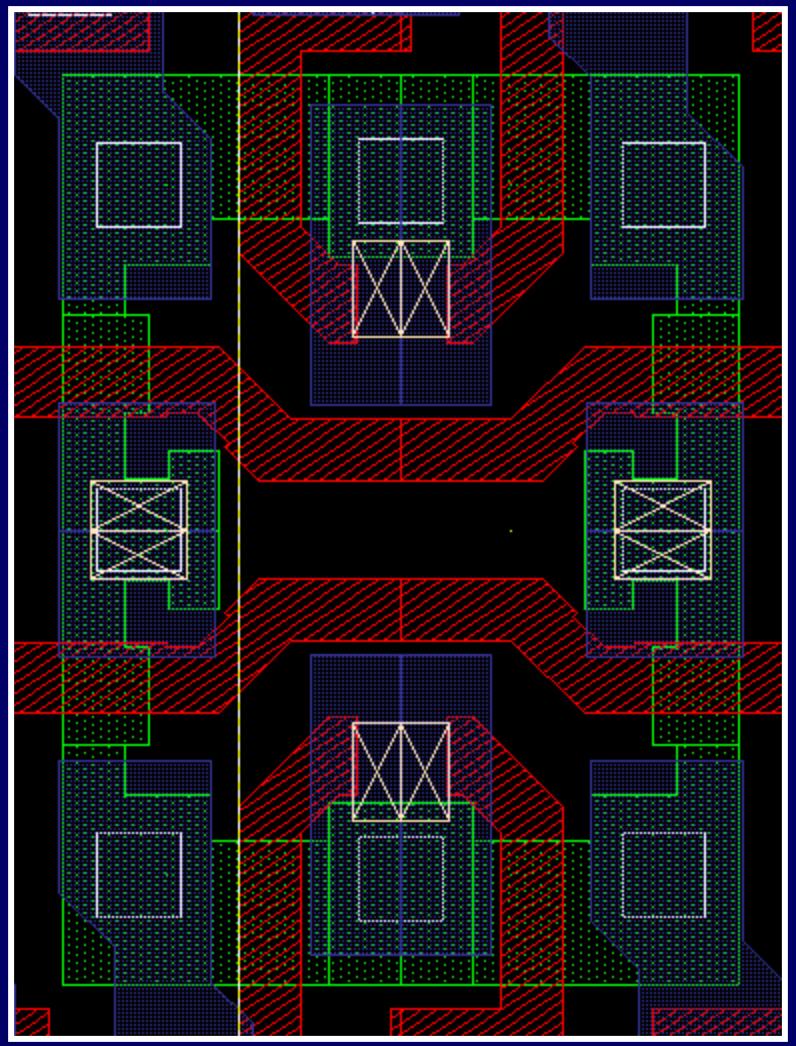
Due to leakage, refresh required
High density
Low access time (e.g., 20 ns)
Low prices

2. Architecture: layout model (array)

- Physical implementation of the system
- Diffusion regions, metal layers, contacts, etc.

- Most detailed description of any system
- Most complex level to analyze

- Usually confidential

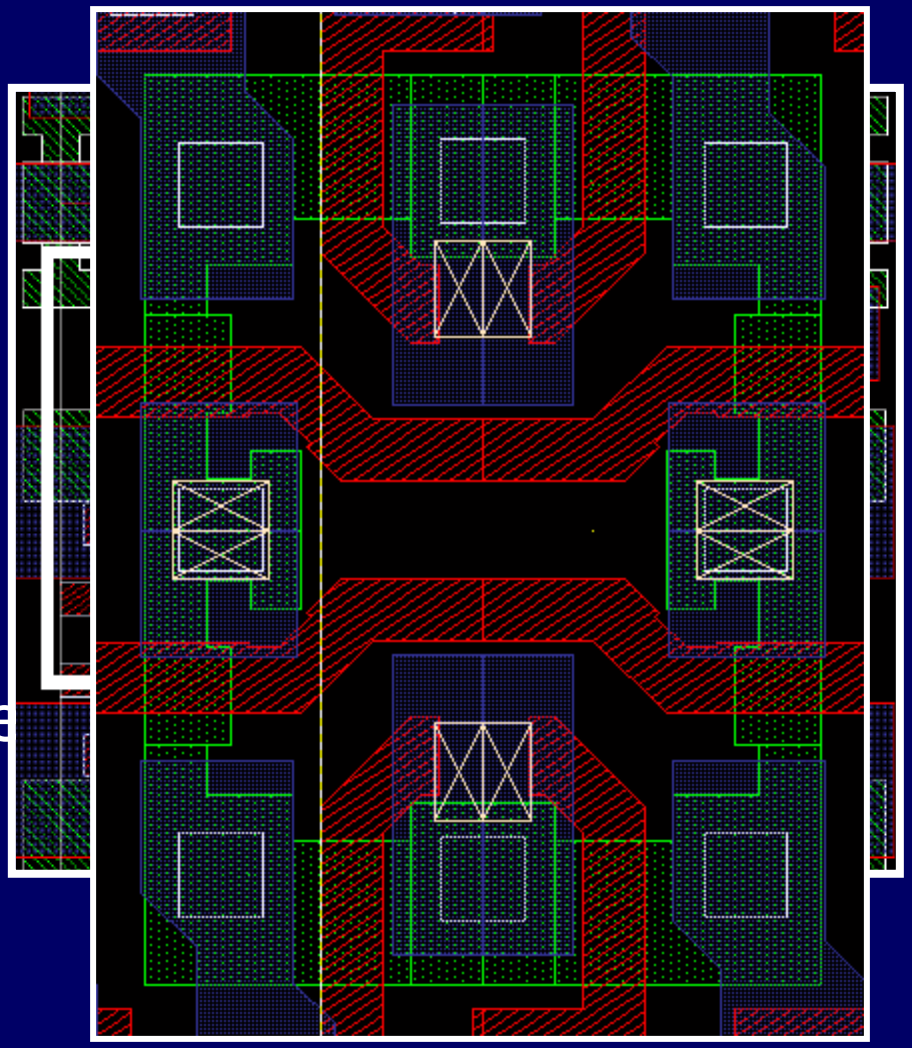


2. Architecture: layout model (array)

- Physical implementation of the system
- Diffusion regions, metal layers, contacts, etc.

- Most detailed description of any system
- Most complex level to analyze

- Usually confidential



VLSI Test Technology and Reliability (ET4076)

Memory Testing:
3. Functional fault models

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)

Delft University of Technology

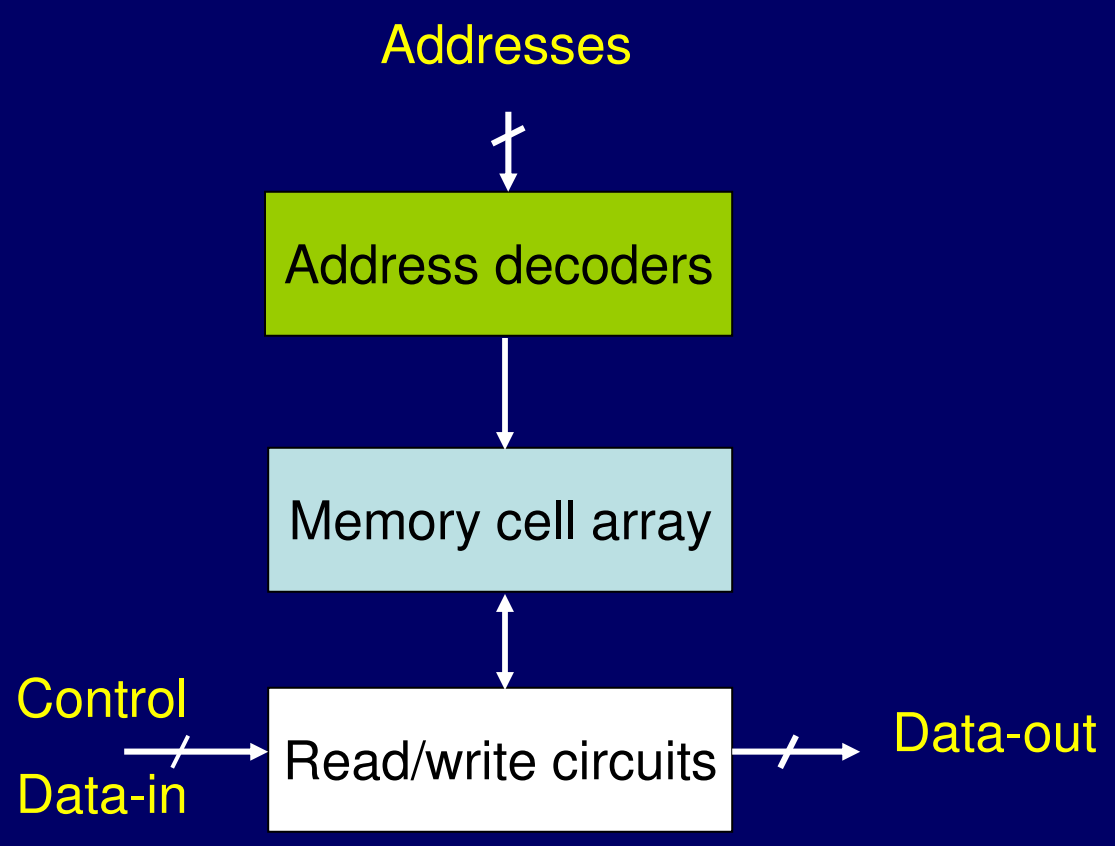
3. Functional fault models

Topics:

- Reduced memory model
- Memory cell array faults
  - Fault primitive concept
  - Classification
  - Definition of fault models
  - Neighborhood pattern sensitive faults
- Address decoder faults
- Read/Write logic faults

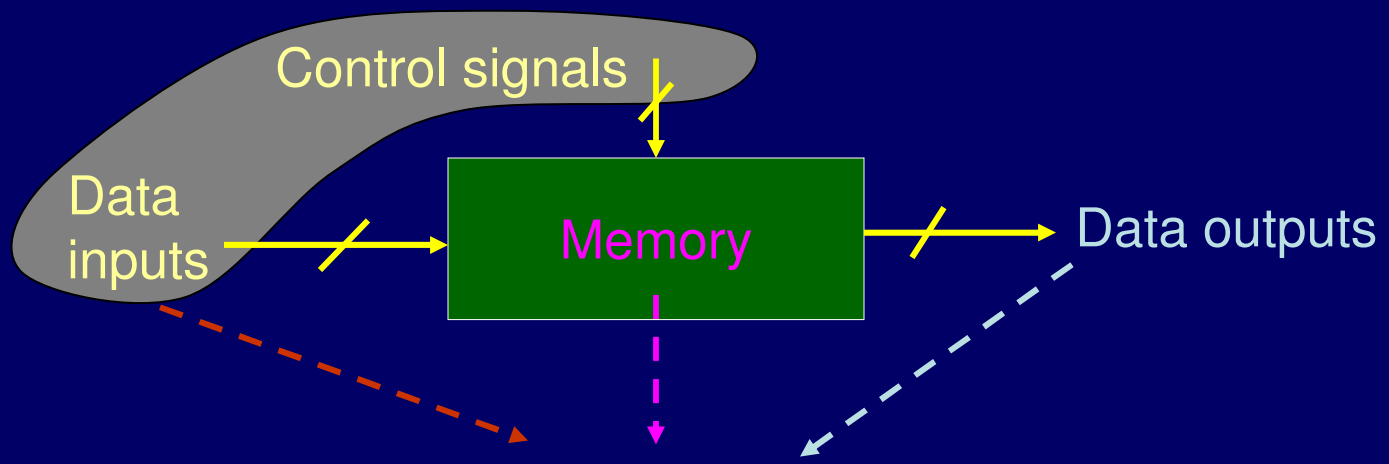
3. Faults: reduced memory model

- Memory faults can be divided into three types/classes



3. Faults: array (fault primitives and fault models)

Fault primitive concept (FP)



Notation: **<S / F / R > = < 0r0 / 1 / 1 >**

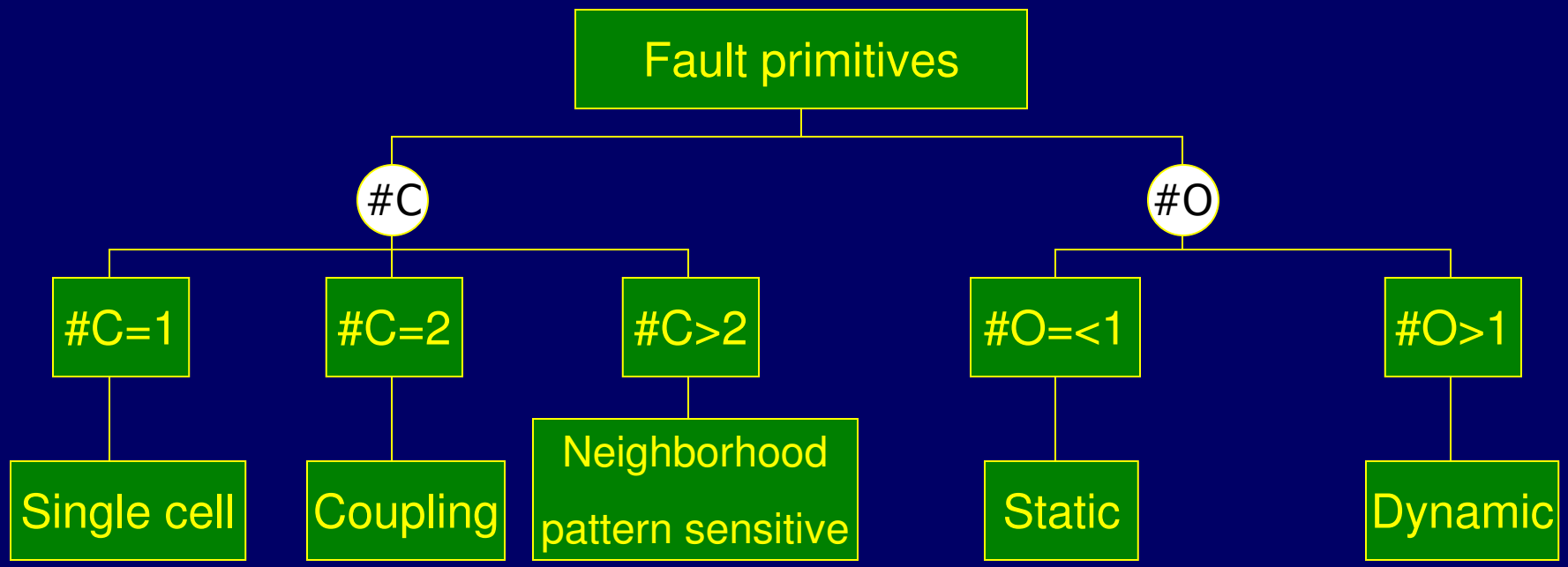
- **S**: **sensitizing sequence**; e.g. a read 0 operation **(0r0)**
- **F**: **Fault effect**; e.g. cell flips from 0 to 1 **(1)**
- **R**: **Read value**; e.g. an incorrect value **1** is read **(1)**

- A **Fault model (FM)** is a **non-empty** set of FPs
  - E.g., Read Destructive Fault (RDF): {<0r0/ 1 / 1 >, <1r1/ 0 / 0 >}

3. Faults: array (classification of fault primitives)

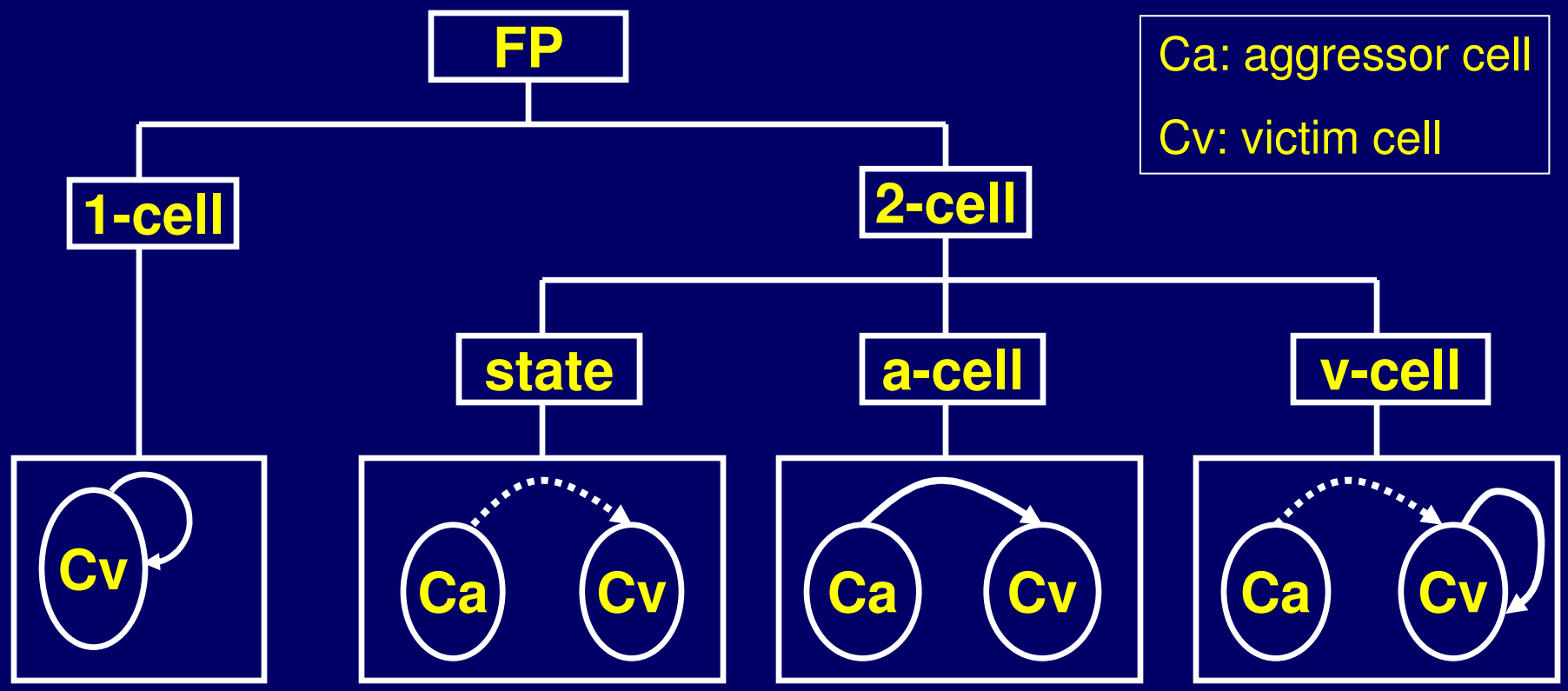
Fault Primitive: FP = <S/F/R>

- Depending on #Cells (C) in **S**
  - Single-cell, two-cell, and neighborhood pattern sensitive faults
- Depending on #Operations (O) in **S**
  - Static faults versus dynamic faults



3. Faults: array (classification of fault primitives)

- Static faults can be classified as:
  - Single-cell (1-cell)
  - Two-cells (2-cell) (i.e., coupling faults)



3. Faults: array (definition of fault models)

- Single-cell FPs: <S/F/R>

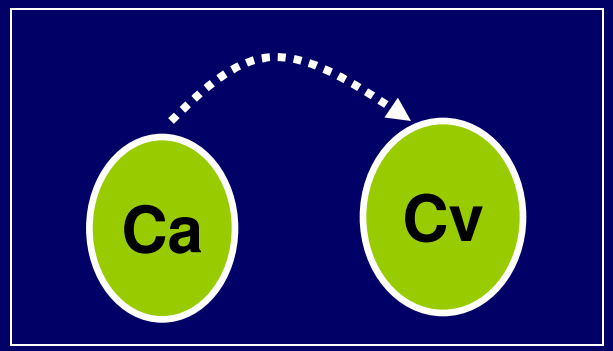
S∈{0,1,0w0,1w1,0w1,1w0, 0r0, 1r1}
F∈{0,1}
R∈{0,1,-}

- Compile the 12 FPs into 6 FMs
  1. **State Fault {1,2} (SF)/(SAF)**
  2. **Write Disturb Fault {3,4} (WDF)**
  3. **Transition Fault {5,6} (TF)**
  4. **Incorrect Read Fault {7, 10} (IRF)**
  5. **Read Destructive Fault {9, 12} (RDF)**
  6. **Deceptive Read Destructive Fault {8,11} (DRDF)**

#	S	F	R	<S/F/R>
1	0	1	-	<0/1/->
2	1	0	-	<1/0/->
3	0w0	1	-	<0w0/1/->
4	1w1	0	-	<1w1/0/->
5	0w1	0	-	<0w1/0/->
6	1w0	1	-	<1w0/1/->
7	0r0	0	1	<0r0/0/1>
8	0r0	1	0	<r0r/1/0>
9	0r0	1	1	<0r0/1/1>
10	1r1	1	0	<1r1/1/0>
11	1r1	0	1	<1r1/0/1>
12	1r1	0	0	<1r1/0/0>

3. Faults: array (definition of fault models)

- Two-cell FPs: state



<S/F/R>= <Sa; Sv/F/R>

Sa∈{0,1}, Sv∈{0,1}

F∈{0,1}

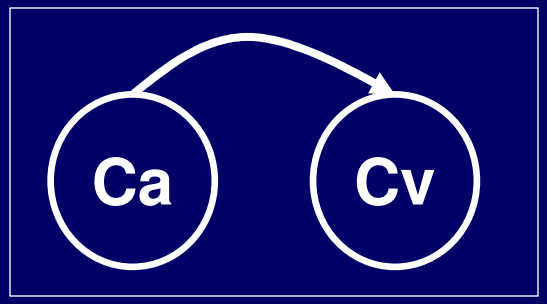
R∈{0,1,-}

#	Sa	Sv	F	R	<Sa; Sv/F/R>
1	0	0	1	-	<0; 0/1/->
2	0	1	0	-	<0; 1/0/->
3	1	0	1	-	<1; 0/1/->
4	1	1	0	-	<1; 1/0/->

Compile the 4 FPs into 1 FM:

State Coupling Fault (CFst)

3. Faults: array (definition of fault models)



Two-cell FPs: a-cell accessed

- Sa∈{0w0, 1w1, 0w1, 1w0, 0r0, 1r1}
- Sv∈{0,1}; F∈{0,1}; R∈{-}
- x*= NOT x; x∈{0,1}
- r= read; w= write

Compile the 12 FPs into 1 FM:

Disturb Coupling Fault (CFds)

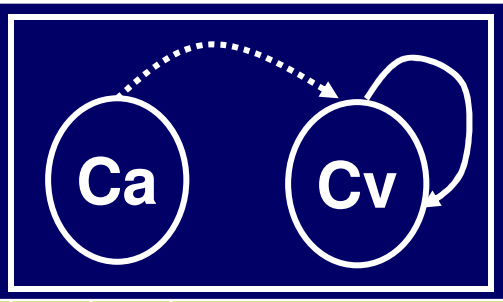
CFds divided into 3 types:

- $CFds_t$ {1, 2} (transition write)
- $CFds_n$ {3, 4} (non-trans. write)
- $CFds_r$ {5,6} (read)

#	Sa	Sv	F	R	<Sa; Sv/F/R>
1	xwx*	0	1	-	<xwx*; 0/1/->
2	xwx*	1	0	-	<xwx*; 1/0/->
3	xwx	0	1	-	<xwx; 0/1/->
4	xwx	1	0	-	<xwx; 1/0/->
5	xrx	0	1	-	<xrx; 0/1/->
6	xrx	1	0	-	<xrx; 1/0/->

3. Faults: array (definition of fault models)

Two-cell FPs: 1PF2**v** (v-cell accessed)



Sa∈{0,1}

Sv∈{0w0,1w1,0w1,1w0, 0r0, 1r1}

F∈{0,1}

R∈{0,1,-}

x∈{0,1}

#	Sa	Sv	F	R	<S/F/R>
1	x	0w0	1	-	<x; 0w0/1/->
2	x	1w1	0	-	<x; 1w1/0/->
3	x	0w1	0	-	<x; 0w1/0/->
4	x	1w0	1	-	<x; 1w0/1/->
5	x	0r0	0	1	<x; 0r0/0/1>
6	x	0r0	1	0	<x; r0r/1/0>
7	x	0r0	1	1	<x; 0r0/1/1>
8	x	1r1	1	0	<x; 1r1/1/0>
9	x	1r1	0	1	<x; 1r1/0/1>
10	x	1r1	0	0	<x; 1r1/0/0>

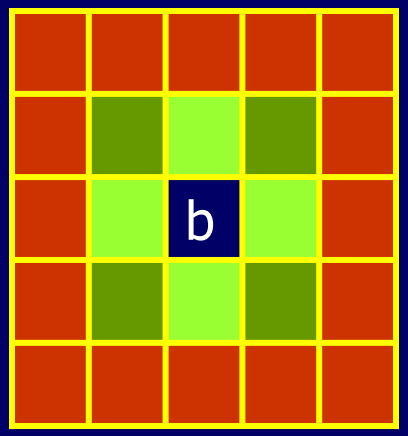
Compile the 24 FPs into 5 FMs:

CF= Coupling Fault

1. **Write Disturb CF {1,2} (CFwd)**
2. **Transition CF {3,4} (CFtr)**
3. **Incorrect Read CF {5,8 } (CFir)**
4. **Read Destructive CF {7,10}(CFrd)**
5. **Deceptive CFrd {6,9} (CFdr)**

3. Faults: array (definition of fault models)

- Pattern sensitive faults, PSF, (k-coupling faults)
  - Conditional single-cell fault
    - k-1 cell in a certain state
  - Conditional two-cell fault
    - k-2 cells in a certain state
- Neighborhood PSF (NPSF)
  - PSF very complicated
  - Restrictions: k cells are neighbors (5 or 9)
- Types of NPSFs
  - Passive NPSF: conditional Transition Fault (TF)
  - Static NPSF: conditional State Coupling Fault (CFst)
  - Disturb NPSF: conditional Disturb coupling Fault (CFds)



b Base cell

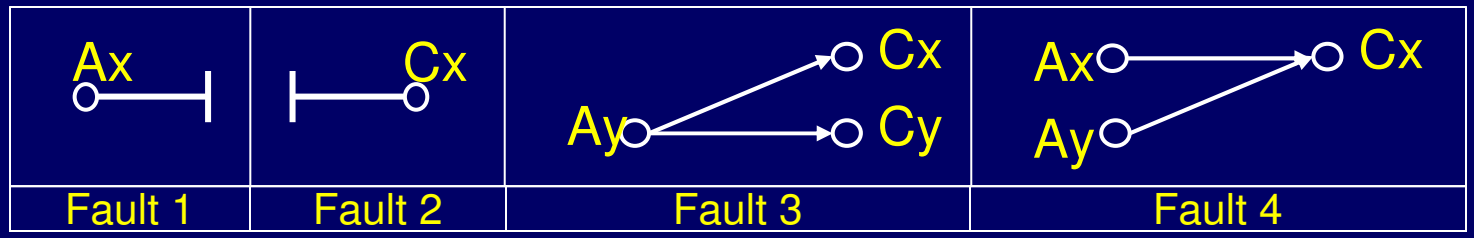
5 neighbors

9 neighbors

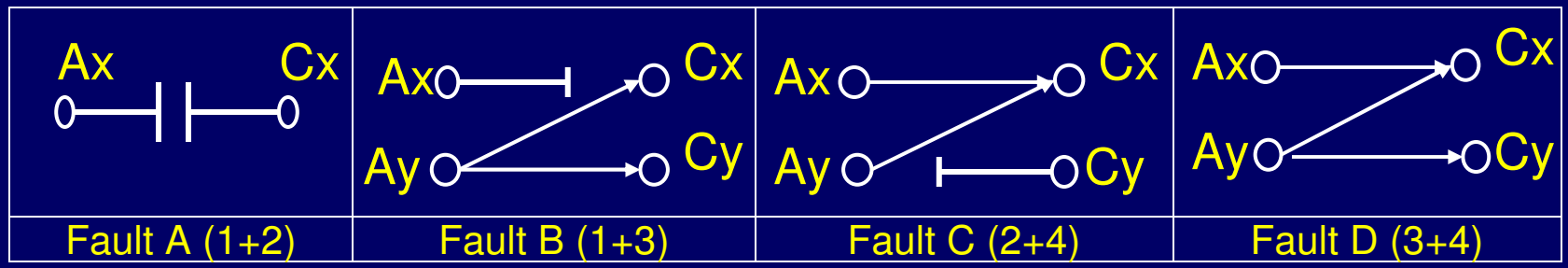
K-coupling

3. Faults: address decoder (functional faults)

- 1-1 correspondence: Address <----> Cell
- Possible faults:
  1. An address does not access a certain cell
  2. A cell is not accessed with a certain address
  3. With a certain address, multiple cells are accessed
  4. A certain cell can be accessed with multiple addresses

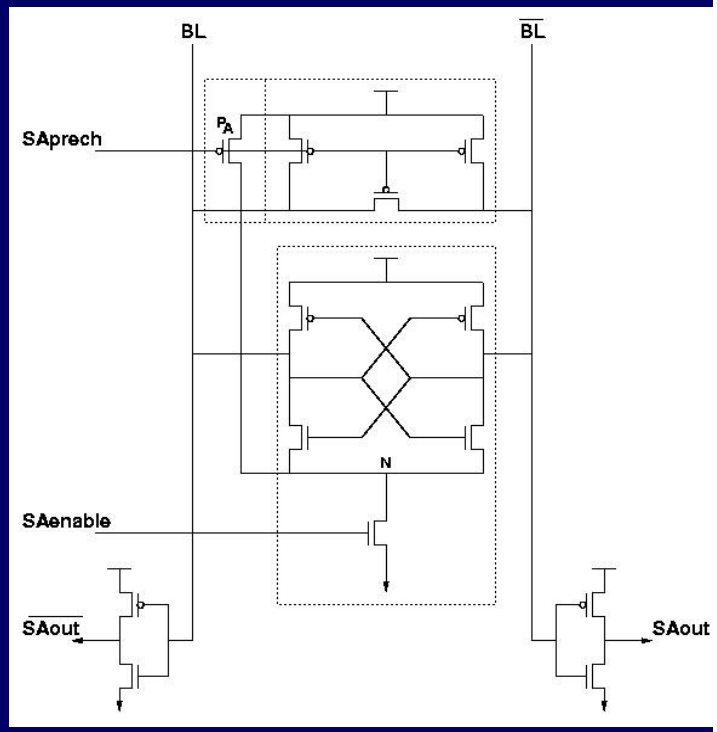


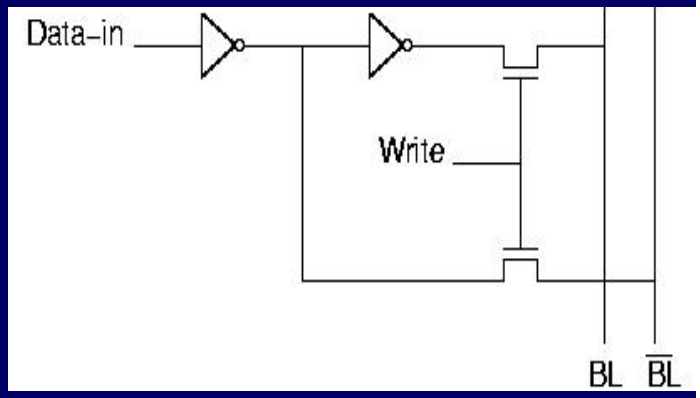
- Fault combinations



3. Faults: Read/Write logic (functional faults)

- Consist of faults in
  - Sense amplifiers, Write drivers
  - Pre-charge circuits, Multiplexes
  - Etc
- They have never been analyzed
- Research map them into memory cell array faults
  - Assuming a defect to be
    - Complete open
    - A short / bridges with very low resistance value
- Good coverage realized in the past (old technologies >0.1um)

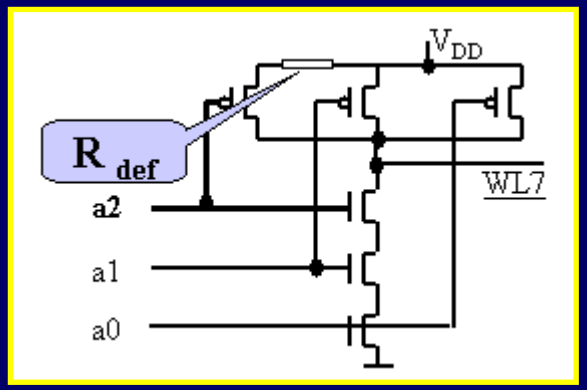


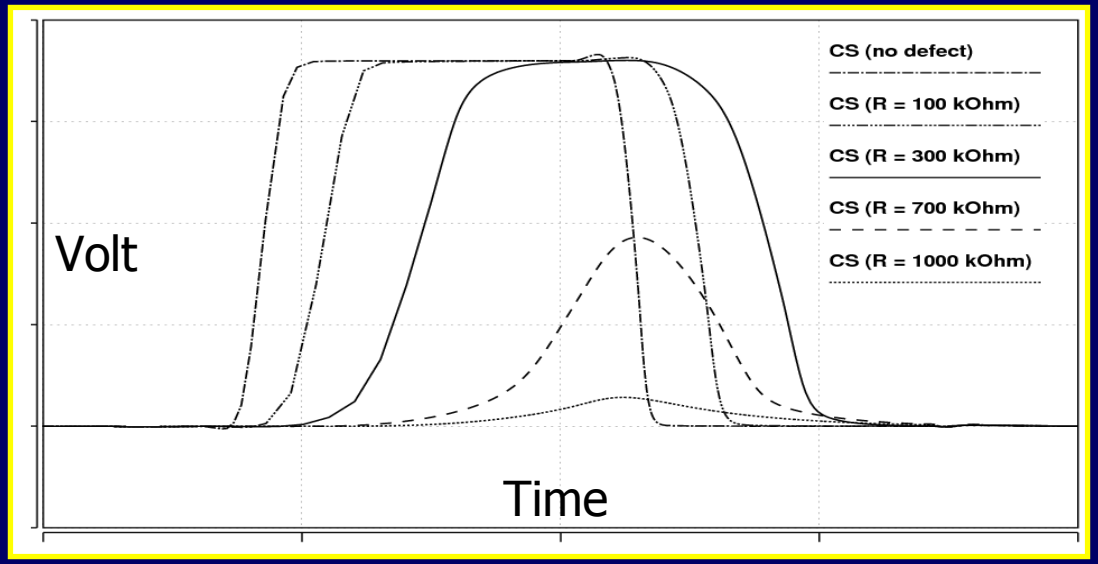


3. Faults: address decoder (delay/dynamic faults)

- Delay faults
  - Word lines and bit lines have an increasing load and high capacitance
  - Open defects are becoming dominant in new technologies

  ⇒ Delay faults are becoming important





3. Faults: Read/Write logic (delay/dynamic faults)

- Consist of e.g.,:
  - Slow Sense Amplifier Fault (SSAF)
  - Slow Write Driver Fault (SWDF)
  - Slow PRecharge circuit Fault (SPRF)

  ⇒Speed related faults

- Cannot be mapped nto memory cell array faults
  - Require specific ***operation sequences!***
  - Requires specific ***addressing direction!***

VLSI Test Technology and Reliability (ET4076)

Memory Testing:
4. Memory test development

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)

Delft University of Technology

Contents ...

Topics:

1. Test notation

   How we can describe a memory test

2. Detection conditions

   Fault model → test requirements

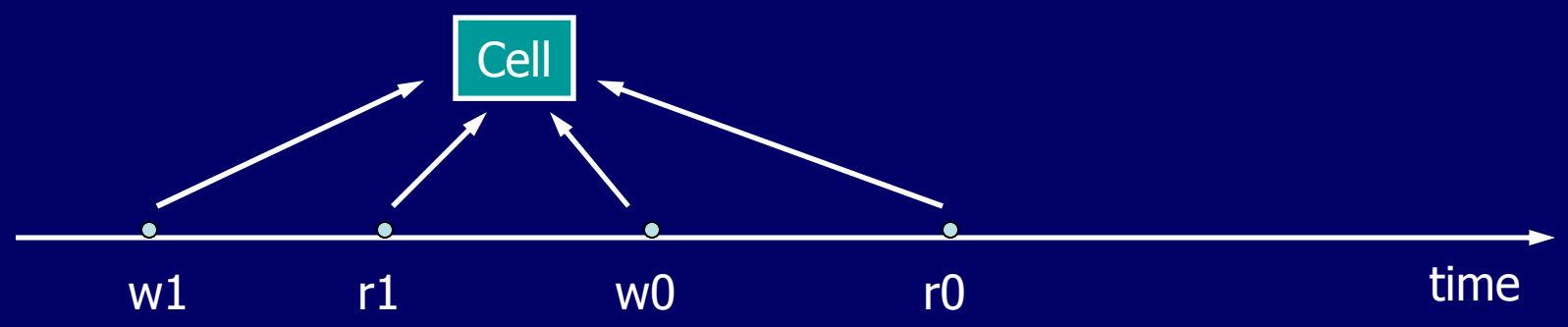
1. Test notation… needed info

How can we test a memory??

- We need:
  - Operations (reads and writes)
  - Data-in (what to write)
  - Data-out (what to read)
  - Addresses (which cells)

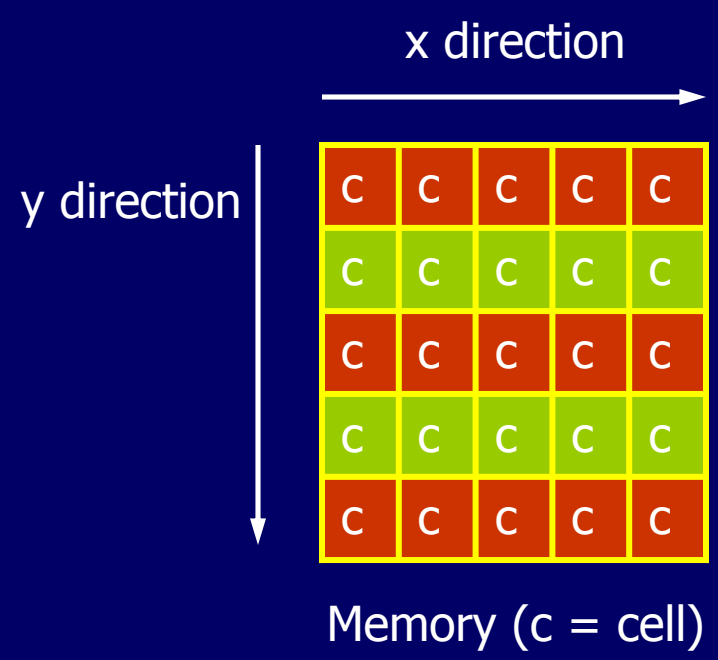
1. Test notation… operations and data

- Operations and data
  - wx = write data x
  - rx = read data (expected value x)
- Example of a sequence
  - (w1, r1, w0, r0)
  - Write and read a cell with 1
  - Then write and read a 0



1. Test notation… addresses

- Addressing:
  - Up addressing: ⇑ (i.e., from i=0 to N)
  - Down addressing: ⇓ (i.e., from i=N to 0)
  - Irrelevant: ⇕
- Two dimensional arrays
  - x-addressing: ⇑**x**
  - y-addressing: ⇑**y**
- Example:
  - ⇑(w1, r1, w0, r0)
  - Perform sequence going up!
  - ***MARCH ELEMENT***



1. Test notation… march tests

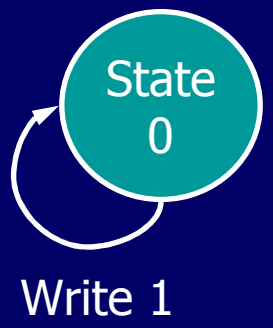
- A **march test** = sequence of **march elements**
- Example: {⇕(w0); ⇑(r0, w1); ⇓(r1)}
  - Up or down, write 0 in all cells
  - Going up, read 0, then write 1
  - Going down, read 1

c	c	c	c	c
c	c	c	c	c
c	c	c	c	c
c	c	c	c	c
c	c	c	c	c

Memory (c = cell)

2. Detection conditions… what is it?

- A **detection condition** = incomplete **march test**
- Detects a specific set of faults!
- Example:
  - Transition Fault 1 (TF1) = <0w1/0/->
  - To detect, start with 0, write 1, then read it!
  - Detection condition
    - **⇕(…0, w1, …, r1, …)**



2. Detection conditions… single-cell faults

- Single-cell FPs: <S/F/R>

S∈{0,1,0w0,1w1,0w1,1w0, 0r0, 1r1}

F∈{0,1}

R∈{0,1,-}

- Compile the 12 FPs into 6 FMs
  1. **State Fault {1,2} (SF)/(SAF)**
  2. **Write Disturb Fault {3,4} (WDF)**
  3. **Transition Fault {5,6} (TF)**
  4. **Incorrect Read Fault {7, 10} (IRF)**
  5. **Read Destructive Fault {9, 12} (RDF)**
  6. **Deceptive Read Destructive Fault {8,11} (DRDF)**

#	S	F	R	<S/F/R>
1	0	1	-	<0/1/->
2	1	0	-	<1/0/->
3	0w0	1	-	<0w0/1/->
4	1w1	0	-	<1w1/0/->
5	0w1	0	-	<0w1/0/->
6	1w0	1	-	<1w0/1/->
7	0r0	0	1	<0r0/0/1>
8	0r0	1	0	<0r0/1/0>
9	0r0	1	1	<0r0/1/1>
10	1r1	1	0	<1r1/1/0>
11	1r1	0	1	<1r1/0/1>
12	1r1	0	0	<1r1/0/0>

2. Detection conditions… single-cell faults

- State Faults (SF) = {<0/1/->, <1/0/->}
- Incorrect Read Fault (IRF) = {<0r0/0/1>, <1r1/1/0>}
- Read Destructive Fault (RDF)= {<0r0/1/1>, <1r1/0/0>}

  Condition: Read 0 and read 1 from each cell

- Deceptive RDF (DRDF)= {<0r0/1/0>, <1r1/0/1>}

  Apply a read to sensitize the fault, followed with a read to detect it

- Transition Fault (TF)= {<0w1/0/->, <1w0/1/->}

  *Apply **a transition write** followed with **a read** operation*

- Write Destructive Fault (WDF) = {<0w0/1/->, <1w1/0/->}

  *Apply **a non-transition write** followed with **a read***

2. Detection conditions… single-cell faults

Question: consider the test **Scan= {⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)}**

Which faults (fault models) are not detected at all?

a. SF b. IRF c. RDF
d. DRDF e. TF f. WDF

Which faults are partially detected?

a. SF b. IRF c. RDF
d. DRDF e. TF f. WDF

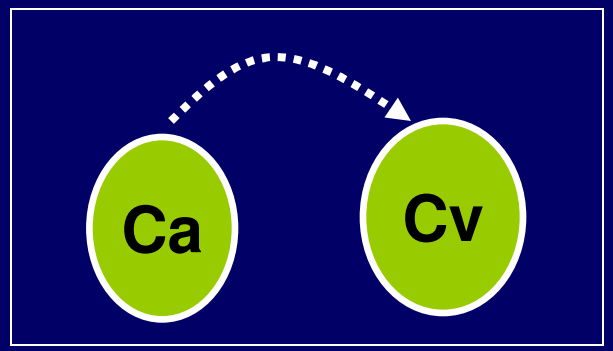
How many single-cell faults does Scan detect?

a. 12/12 b. 11/12 c. 10/12
d. 9/12 e. 8/12 f. 7/12

2. Detection conditions… two-cell faults

- Two-cell FPs: state

<S/F/R>= <Sa; Sv/F/R>
Sa∈{0,1}, Sv∈{0,1}
F∈{0,1}
R∈{0,1,-}



Compile the 4 FPs into 1 FM:
State Coupling Fault (CFst)

#	Sa	Sv	F	R	<Sa; Sv/F/R>
1	0	0	1	-	<0; 0/1/->
2	0	1	0	-	<0; 1/0/->
3	1	0	1	-	<1; 0/1/->
4	1	1	0	-	<1; 1/0/->

2. Detection conditions… two-cell faults

1. State Coupling Faults CFst= {<0; 0/1/->, <0; 1/0/->, <1; 0/1/->, <1; 1/0/->}
   - *Sensitization*: the four states of any two cells are reached
   - *Detection*: a read operation is applied to each cell within each state
   - Example:{⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)}; *consider cells Ci and Cj where i<j*

M0 M1 M2 M3

#	Mi	State	Oper.	State	Effect
1	M0	- -	w0 to Ci	0 -	-
2	M0	0 -	w0 to Cj	00	<0; 0/1/->i,j and <0; 0/1/->j,i *sensitized*
3	M1	00	r0 to Ci	00	<0; 0/1/->j,i *detected*
4	M1	00	r0 to Cj	00	<0; 0/1/->i,j *detected*
5	M2	00	w1 to Ci	10	<0; 1/0/->j,i and <1; 0/1/->i,j *sensitized*
6	M2	10	w1 to Cj	11	<1; 1/0/->i,j and <1; 1/0/->j,i *sensitized*
7	M3	11	r1 to Ci	11	<1; 1/0/->j,i *detected*
8	M3	11	r1 to Cj	11	<1; 1/0/->i,j *detected*

Fault Coverage FC=4/8 sub-Fault Primitives

2. Detection conditions… two-cell faults

Two-cell FPs: 1PF2**v** (v-cell accessed)

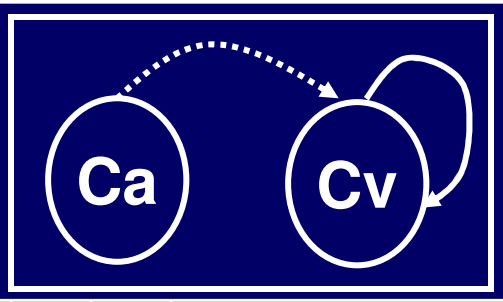
Sa∈{0,1}

Sv∈{0w0,1w1,0w1,1w0, 0r0, 1r1}

F∈{0,1}

R∈{0,1,-}

x∈{0,1}



#	Sa	Sv	F	R	<S/F/R>
1	x	0w0	1	-	<x; 0w0/1/->
2	x	1w1	0	-	<x; 1w1/0/->
3	x	0w1	0	-	<x; 0w1/0/->
4	x	1w0	1	-	<x; 1w0/1/->
5	x	0r0	0	1	<x; 0r0/0/1>
6	x	0r0	1	0	<x; r0r/1/0>
7	x	0r0	1	1	<x; 0r0/1/1>
8	x	1r1	1	0	<x; 1r1/1/0>
9	x	1r1	0	1	<x; 1r1/0/1>
10	x	1r1	0	0	<x; 1r1/0/0>

Compile the 24 FPs into 5 FMs:

CF= Coupling Fault

1. **Write Disturb CF {1,2} (CFwd)**
2. **Transition CF {3,4} (CFtr)**
3. **Incorrect Read CF {5,8 } (CFir)**
4. **Read Destructive CF {7,10}(CFrd)**
5. **Deceptive CFrd {6,9} (CFdr)**

2. Detection conditions… two-cell faults

2. Incorrect Read Coupling Faults

- CFir= {<0; 0r0/0/1>, <0; 1r1/1/0>, <1; 0r0/0/1>, <1; 1r1/1/0>}
- *Sensitization/ Detection:* the four states of any two cells are reached, and a read operation is applied to each cell within each state.
- Example: {⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)} ; *consider cells Ci and Cj where i<j*

#	Mi	State	Oper.	State	Effect
1	M0	- -	w0 to Ci	0 -	-
2	M0	0 -	w0 to Cj	00	-
3	M1	00	r0 to Ci	00	<0; 0r0/0/1>j,i *sensitized & detected*
4	M1	00	r0 to Cj	00	<0; 0r0/0/1>i,j *sensitized & detected*
5	M2	00	w1 to Ci	10	-
6	M2	10	w1 to Cj	11	-
7	M3	11	r1 to Ci	11	<1; 1r1/1/0>j,i *sensitized & detected*
8	M3	11	r1 to Cj	11	<1; 1r1/1/0>i,j *sensitized &detected*

Result: FC=4/8 sub-Fault Primitives

2. Detection conditions… two-cell faults

3. Read Destructive Coupling Faults

- CFrd= {<0; 0r0/1/1>, <0; 1r1/0/0>, <1; 0r0/1/1>, <1; 1r1/0/0>}
- *Sensitization/ Detection:* the four states of any two cells are reached, and a read operation is applied to each cell within each state.
- Example: {⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)} ; *consider cells Ci and Cj where i<j*

#	Mi	State	Oper.	State	Effect
1	M0	- -	w0 to Ci	0 -	-
2	M0	0 -	w0 to Cj	00	-
3	M1	00	r0 to Ci	00	<0; 0r0/0/1>j,i *sensitized & detected*
4	M1	00	r0 to Cj	00	<0; 0r0/0/1>i,j *sensitized & detected*
5	M2	00	w1 to Ci	10	-
6	M2	10	w1 to Cj	11	-
7	M3	11	r1 to Ci	11	<1; 1r1/1/0>j,i *sensitized & detected*
8	M3	11	r1 to Cj	11	<1; 1r1/1/0>i,j *sensitized &detected*

Result: FC=4/8 sub-Fault Primitives

2. Detection conditions… two-cell faults

4. Deceptive Read Destructive Coupling Faults

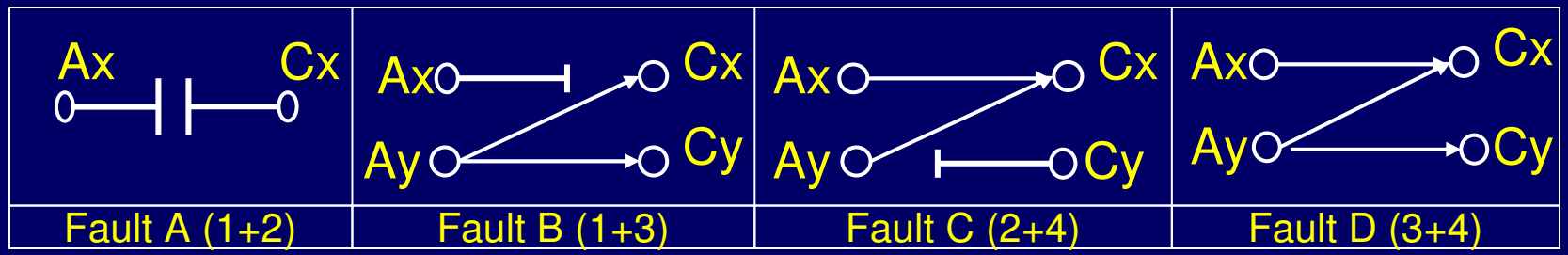
- CFdr= {<0; 0r0/1/0>, <0; 1r1/0/1>, <1; 0r0/1/0>, <1; 1r1/0/1>}
- *Sensitization:* the four states of any two cells are reached, and a read operation is applied to each cell within each state.
- *Detection*: ***another read*** is applied to each cell within each state
- Example: {⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)} ; *consider cells Ci and Cj where i<j*

#	Mi	State	Oper.	State	Effect
1	M0	- -	w0 to Ci	0 -	-
2	M0	0 -	w0 to Cj	00	-
3	M1	00	r0 to Ci	00	<0; 0r0/1/0>j,i *sensitized*
4	M1	00	r0 to Cj	00	<0; 0r0/1/0>i,j *sensitized*
5	M2	00	w1 to Ci	10	-
6	M2	10	w1 to Cj	11	-
7	M3	11	r1 to Ci	11	<1; 1r1/0/1>j,i *sensitized*
8	M3	11	r1 to Cj	11	<1; 1r1/0/1>i,j *sensitized*

Result: FC=0/8 sub-Fault Primitives

2. Detection conditions… AFs

- Address decoder faults:



Detection condition

A test detects AFs iff it contains the following two march elements; *x=0* or *x=1*:

- $\Uparrow$ (r*x*, …, w*x**, [r*x**]^h)
- $\Downarrow$ (r*x**, …, w*x* , [r*x*]^h)

h for hammer; *h*≥ *1*

Question: what is the optimal test detecting all AFs?

- $\Uparrow$ (r*x*, …, w*x**, [r*x**]^h)
- $\Downarrow$ (r*x**, …, w*x* , [r*x*]^h)

1. {$\Uparrow$(r1,w1,r1); $\Downarrow$(r1,w0,r0) }?
2. {$\Uparrow$(r0,w1,r1); $\Downarrow$(r1,w0,r0) }?
3. {$\Uparrow$(w0); $\Uparrow$(r0,w1,r1); $\Downarrow$(r1,w0,r0) }?

VLSI Test Technology and Reliability (ET4076)

Memory Testing: 5. Well-known memory tests

Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Section: Computer Engineering Laboratory (CE)

Delft University of Technology

5. Memory tests… Ad-hoc tests

- Most know Ad-hoc test:
  - Scan= Zero-One Test
  - Checkerboard
  - GalPat
  - Walking 1/0
- Typically until 1980's
- Absence of formal fault models
- Not acceptable due to
  - Low fault coverage (Scan/ Checkerboard)
  - High cost (GalPat/ Walking 1/0)

5. Memory tests… Ad-hoc tests

Scan= Zero-One Test

1. *Write 0 in all cells (starting at cell 0)*
2. *Read all cells (starting at cell 0)*
3. *Write 1 in all cells (starting at cell 0)*
4. *Read all cells (starting at cell 0)*

- In short notation: {⇑(w0); ⇑(r0); ⇑(w1); ⇑(r1)}
- Test length: 4n => O(n): ***linear time complexity***
- Low Fault coverage:
  - 7/12 single-cell faults (1PF1s)
  - 18/72 two-cell faults (1PF2s)
  - No address decoder faults (AFs)

5. Memory tests… Ad-hoc tests

- **Checkerboard**

  1. Divide the memory cell into two groups, say cells-1 and cells-2

  2. Write 1 in cells-1 and 0 in cells-2

  3. Read all cells

  3. Write 0 in all cells-1 and 1 in cells-2.

  4. Read all cells

- Test length: 4n => O(n)

- Short Notation: $\{\Uparrow(w1_i, w0_{i+1}); \Uparrow(r1_i, r0_{i+1}); \Uparrow(w0_i, w1_{i+1}); \Uparrow(r0_i, r1_{i+1})\}$

  where $0 \leq i < n-2$

- Low fault coverage (similar to Scan)

1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

Memory array

5. Memory tests… Ad-hoc tests

Galpat, Walking 1/0

```
for d=0 to d=1
{       for i=0 to n-1
                A[i]=d;
        for base-cell=0 to n-1
        {   A[base-cell]=d*;
            Perform Read-Action;
            A[base-cell]=d;
        }
}
```

Read-Action ***Galpat***

```
for cell=0 to n-1 (base-cell excluded)
{     if A[cell]≠d
            then output ("ERROR")
      if A[base-cell]≠d*
            then output ("ERROR")
}
```

Read-Action ***Walking 1/0***

```
for cell=0 to n-1 (base-cell excluded)
{   if A[cell]≠d
      then output ("ERROR")
}
if A[base-cell]≠d*
    then output ("ERROR")
```

➔ **Time complexity: O(n^2)**

Low fault coverage

Galpat: 10/12 1PFs, 44/72 1PF2s, no AFs

Walking 1/0: 8/12 1PFs, 40/72 1PF2s, no AFs

5. Memory tests… Ad-hoc vs march

- Ad-hoc tests:
  - Low fault coverage, and/or
  - High time complexity
    - Industrially not acceptable for serious test purposes
  - Based on speculations not on fault models

- March tests:
  + Based on fault models
  + Linear time complexity
  + Acceptable fault coverage
  - With newer technologies
    - New defects
    - New fault models required
    - New advanced tests

5. Memory tests… march tests

List of some well known march tests

- MATS+ [Nair, 79]: {⇕(w0); ⇑(r0,w1); ⇓(r1,w0)}
- MATS++ [Breuer, 76}: {⇕(w0); ⇑(r0,w1); ⇓(r1,w0,r0)}
- PMOVI [de Jonge, 76]
  {⇓(w0); ⇑(r0,w1,r1); ⇑(r1,w0,r0); ⇓(r0,w1,r1); ⇓(r1,w0,r0)}
- March B [Suk, 81]:
  {⇕(w0); ⇑(r0,w1,r1,w0,r0,w1); ⇑(r1,w0,w1); ⇓(r1,w0,w1,w0); ⇓(r0,w1,w0)}
- March C- [van de Goor, 91]:
  {⇕(w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0); ⇕(r0)}

5. Memory tests… march tests

- Advanced tests: fault primitive-based
  - High fault coverage
  - Cover realistic faults
  - Linear with the size of the memory
- An example of advanced tests based on fault primitives is as follows
  - March MSS [Hamdioui 02, Harutunvan, 05] (18n): *test for all static simple faults*

March MSS={

⇕(w0);	M0
⇑(r0,r0,w1,w1);	M1
⇑(r1,r1,w0,w0);	M2
⇓(r0,r0,w1,w1);	M3
⇓(r1,r1,w0,w0);	M4
⇕(r0)}	M5

5. Memory tests… march tests

Fault Model	MATS+ (5n)	March C- (10n)	PMOVI (13n)	March SR (14n)	March G (23n)	Ham. (49n)	Galpat $O(n^2)$	March MSS (22n)
SAF	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
TF	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
WDF	0/2	0/2	0/2	0/2	0/2	2/2	0/2	2/2
IRF/ RDF	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
DRDF	0/2	0/2	2/2	2/2	0/2	2/2	0/2	2/2
CFst	4/8	8/8	8/8	8/8	6/8	8/8	8/8	8/8
CFds [r*x*]	3/8	8/8	8/8	8/8	7/8	8/8	8/8	8/8
CFds [*x*w*x**]	3/8	8/8	7/8	8/8	8/8	7/8	4/8	8/8
CFds [*x*w*x*]	0/8	0/8	0/8	0/8	0/8	7/8	0/8	8/8
CFtr	2/8	8/8	8/8	8/8	4/8	8/8	4/8	8/8
CFwd	0/8	0/8	0/8	0/8	0/8	8/8	0/8	8/8
CFrd/ CFir	4/8	8/8	8/8	8/8	4/8	8/8	8/8	8/8
CFdr	0/8	0/8	6/8	4/8	0/8	6/8	0/8	8/8
Total	29/84	56/84	63/84	62/84	57/84	80/84	54/84	84/84

Trends in memory testing

- Dynamic faults/ address decoder delay faults
- Intra-word faults
- Faults in the periphery circuits (speed related faults)
- Built-in-Self-Testing (BIST) and repair (BISR)
- On-line testing/Built-in-self-correction (BISC)/soft faults/errors
- Reliability in memories
- Testing ROMs (FLASH memories)

Summary

- Memory testing is very important
  - Large share of SoC area, large impact on quality, yield and reliability
- Memory Faults classified into three types
  - Memory cell array faults
    - Single cell versus coupling faults
  - Address decoder faults
  - Peripheral circuits faults
- Static faults versus dynamic faults
- Three classes of memory test algorithms
  - Ad-hoc tests
  - March tests
  - Fault-primitive based tests
- Major challenges
  - Speed related faults
  - Repair, diagnosis, …