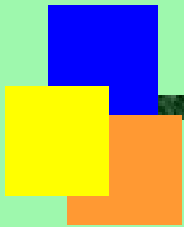


國立清華大學
電機工程學系
102 學年度第一學期

EE-6250
超大型積體電路測試
VLSI Testing



授課教師：黃錫瑜

2013 年 Fall Semester

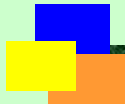
清華大學電機系

超大型積體電路測試

講義

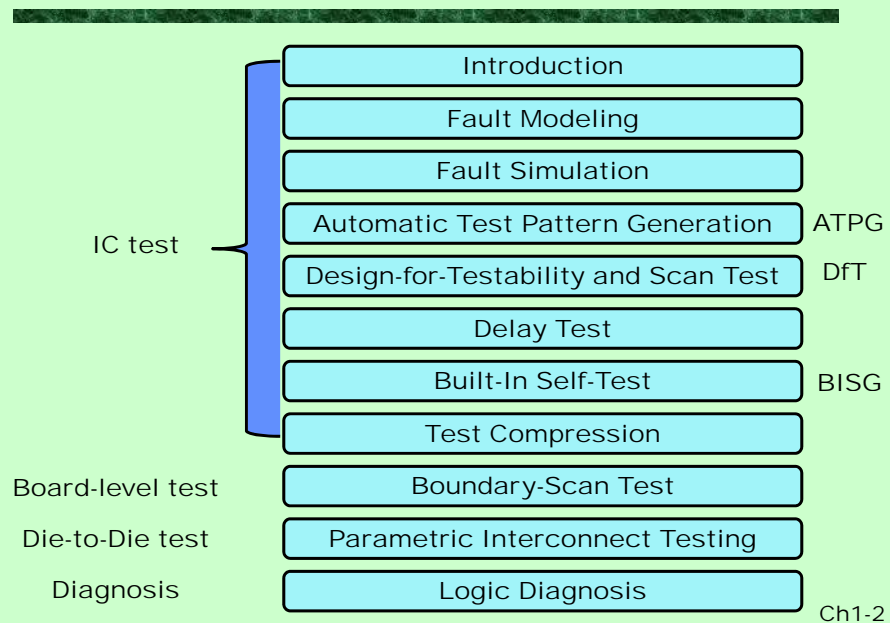
國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 1
Introduction

Course Flow

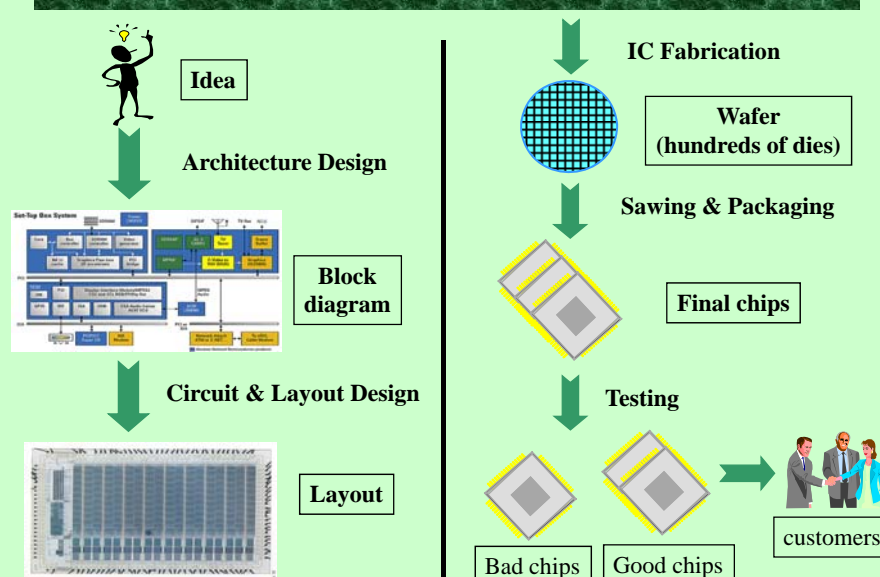


What You can Benefit from this Course?

- **Values of Acquired Knowledge**
 - Making ICs more testable
 - Making ICs/Boards/Systems more debuggable
 - Making ICs Faster Time-to-Market
 - Making ICs Faster Time-to-Volume
- **Academic Training**
 - Testing is a rich field as you will know.
 - Testing is a good topic for MS/Ph.D. theses.
- **Career Development**
 - IC 設計公司 (讓晶片的可測試性更高)
 - 半導體廠 (故障診斷, 良率追蹤與分析與改善)
 - 測試產業 (量產測試之規劃與執行)
 - 電子系統廠 (系統故障診斷, 可靠度分析與改善)

Ch1-3

Chip Design & Manufacturing Flow



Ch1-4

Design Verification, Testing and Diagnosis

- Design Verification:
 - Ascertain the design perform its specified behavior
- Testing:
 - Exercise the system and analyze the response to ascertain whether it behaves correctly **after manufacturing**
- Diagnosis:
 - To locate the **cause(s)** of misbehavior after the incorrect behavior is detected

Ch1-5

Manufacturing Defects

- Material Defects
 - bulk defects (cracks, crystal imperfections)
 - surface impurities
- Processing Faults
 - missing contact windows
 - parasitic transistors
 - oxide breakdown
- Time-Dependent Failures
 - dielectric breakdown
 - electro-migration
- Packaging Failures
 - contact degradation
 - seal leaks

Ch1-6

Faults, Errors and Failures

- Fault:
 - A **physical defect** within a circuit or a system
 - May or may not cause a system failure
- Error:
 - Manifestation of a fault that results in **incorrect circuit (system) outputs or states**
 - Caused by faults
- Failure:
 - Deviation of a circuit or system from its specified **behavior**
 - Fails to do what it should do
 - Caused by an error
- Fault ---> Error ---> Failure

Ch1-7

Reliability Test

- Temperature Related
 - Hi-Temperature Life Test
 - Low-Temperature Life Test
 - Temperature-Cycling Test
- Humidity Test
- Salt Mist Test
- UV (Ultra-Violet) Test
- ESD Test
 - ESD stands for Electro-Static Discharge
- Whole Mechanical Test

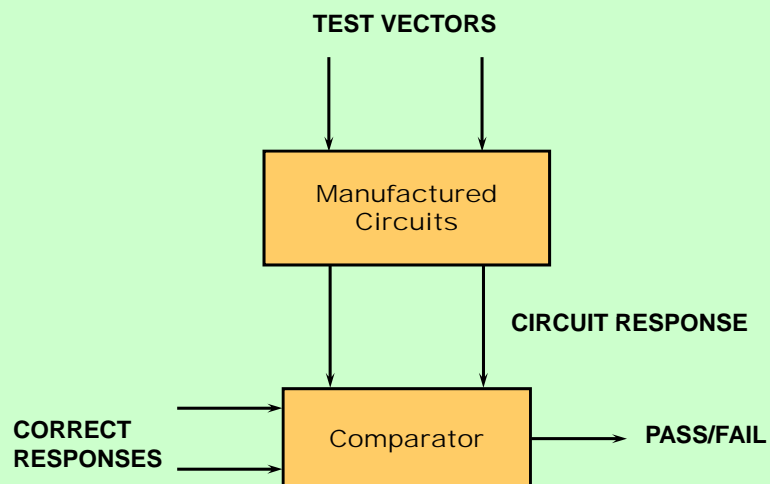
Ch1-8

Detailed Reliability Test Items

- **Temperature Related**
 - Operation: 0°C/120hr ~ 70°C/120hr (商規)
 - Operation: -40°C/120hr ~ 85°C/120hr (工規)
 - Storage: -40°C/200hr ~ 85°C/500hr
 - Junction Temperature: Max. 95°C
- **Humidity Test**
 - Operation: 25°C/95% humidity (商規)
 - Operation: 40°C/95% humidity (工規)
 - Storage: 85°C/95% humidity
- **Salt Mist Test**
 - Salt Water Spray
- **UV Test**
 - UV (254nm), 15Ws/cm²
 - X-ray exposure, 0.1Gy/1hr
- **ESD Test**
 - For example, For Contact Pads, ±4KV, Human Body Mode
- **Whole Mechanical Test**
 - Vibration (15G, 10 to 2KHz), Impact, Torque, Bending, Drop test

Ch1-9

Scenario of Manufacturing Test



Ch1-10

Courses on Agilent 93000 at CIC



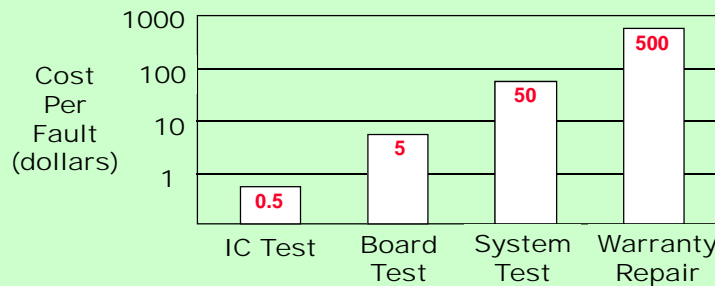
Sample Information: (What to expect from that kind of course)

上課地點	新竹CIC - 訓練教室B (新竹市科學園區路一號26號八樓)
課程說明	本課程將介紹如何利用Agilent 93000 SoC Tester來進行數位晶片的測試，課程內容包含loadboard使用方式、測試程式開發、量測結果分析。
課程大綱	(1) Agilent 93000 SoC Tester 介紹 (2) Loadboard 使用方法 (3) Test Pattern 轉換 (4) Test flow 建立 (5) Result Analysis

Ch1-11

Purpose of Testing

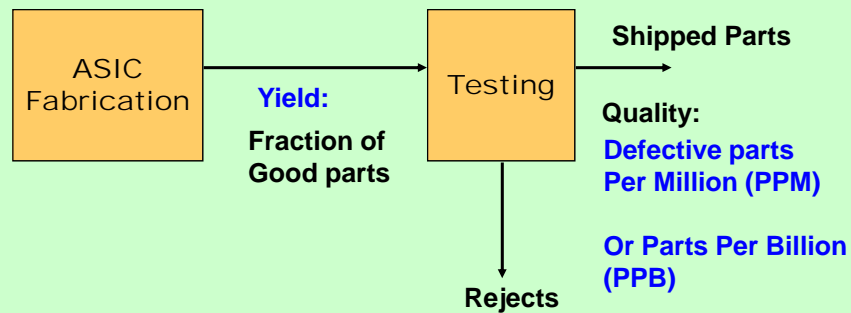
- Verify Manufacturing of Circuit
 - Improve System Reliability
 - Diminish System Cost
- Cost of repair
 - goes up by an order of magnitude each step away from the fab. line



B. Davis, "The Economics of Automatic Testing" McGraw-Hill 1982

Ch1-12

Testing and Quality



Quality of shipped part is a function of **yield Y** and the **test (fault) coverage T**.

Ch1-13

Fault Coverage

- Fault Coverage T
 - Is the measure of the **ability of a set of tests** to **detect** a given class of faults that may occur on the device under test (DUT)

$$T = \frac{\text{No. of detected faults}}{\text{No. of all possible faults}}$$

Ch1-14

Defect Level

- Defect Level

- Is the fraction of the shipped parts that are defective (單位 ppm or ppb)

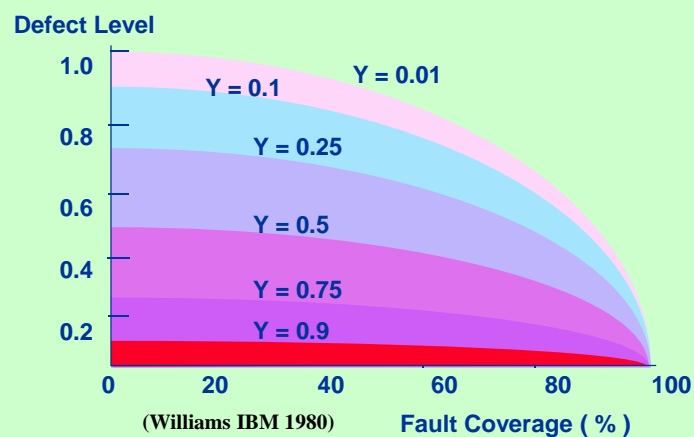
$$DL = 1 - Y^{(1-T)}$$

Y: yield

T: fault coverage

Ch1-15

Defect Level v.s. Fault Coverage



High fault coverage → Low defect level

Ch1-16

DPM v.s. Yield and Coverage

Yield	Fault Coverage	Defective PPM
50%	90%	67,000
75%	90%	28,000
90%	90%	10,000
95%	90%	5,000
99%	90%	1,000
90%	90%	10,000
90%	95%	5,000
90%	99%	1,000
90%	99.9%	100

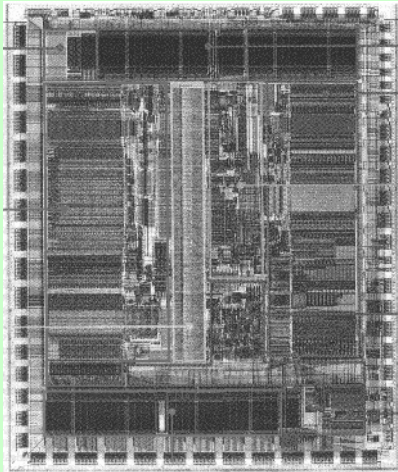
Ch1-17

Why Testing Is Difficult ?

- Test application time could explode for exhaustive testing of VLSI
 - For a combinational circuit with 50 inputs, we need $2^{50} = 1.126 \times 10^{15}$ test patterns.
 - Assume one test per 10^{-7} sec, it takes 1.125×10^8 sec = 3.57yrs. to test such a circuit.
 - Test generation for **sequential circuits** are even more difficult due to the lack of **controllability** and **observability** at flip-flops (latches)
- Functional testing
 - may **NOT** be able to detect the physical faults

Ch1-18

DEC Alpha Chip (1994)

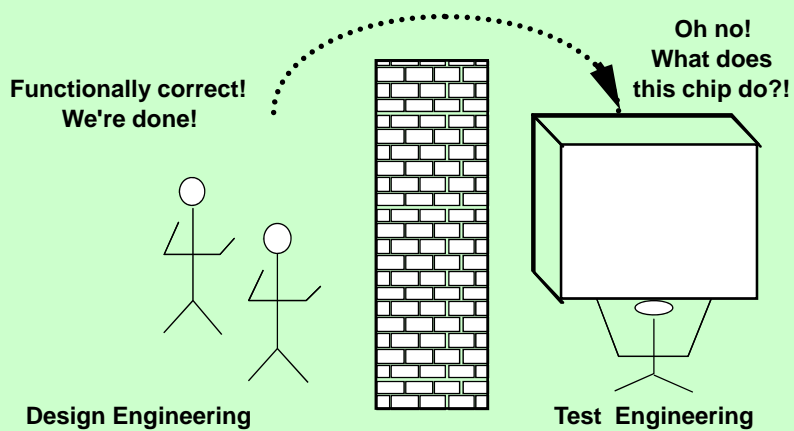


- 64-bit RISC
- 200 MHz
- 400 MIPS
- 200 Mflops
- 16.8 x 13.9 mm² die
- 0.68 million transistors
- 431-pin package
- 3.3 V
- 30 W power consumption

Ch1-19

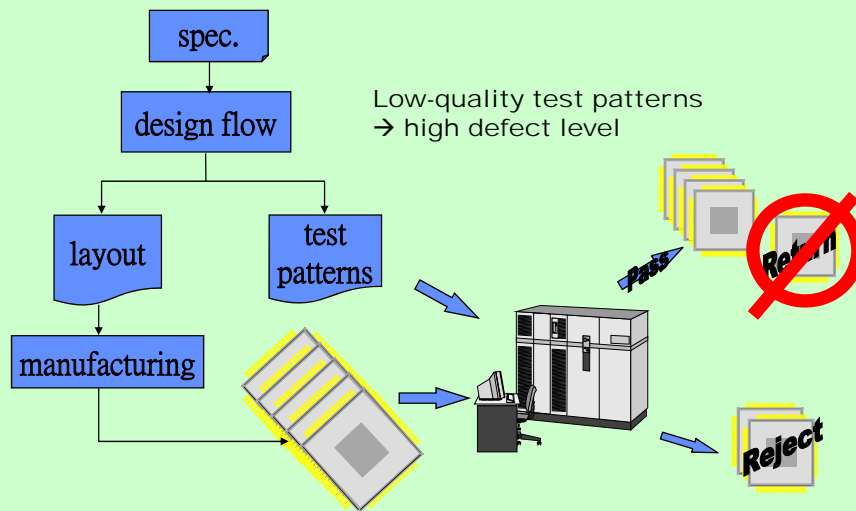
The Infamous Design/Test Wall

**30 years of experience proves that
test after design does not work!**



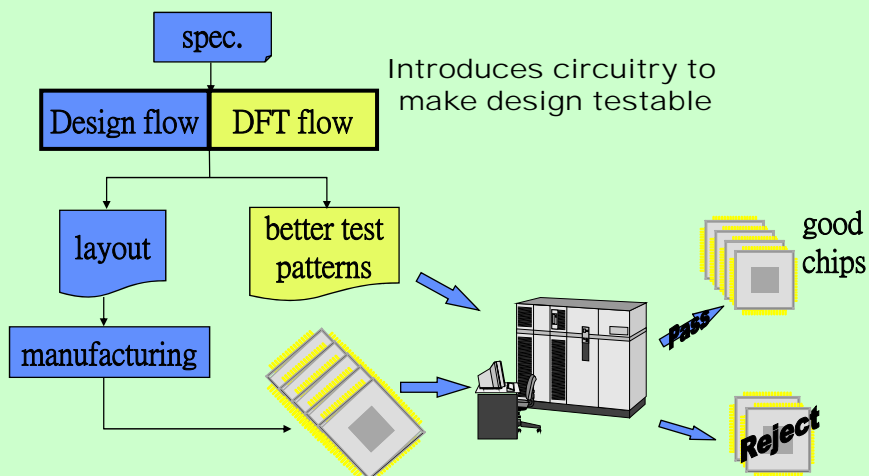
Ch1-20

Old Design & Test Flow



Ch1-21

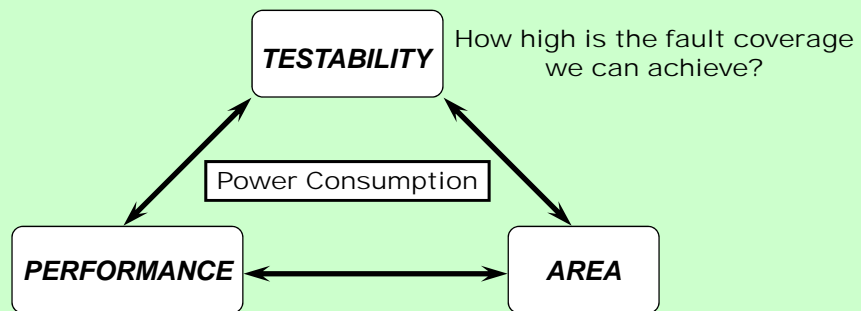
New Design and Test Flow



Ch1-22

New Design Mission

- Design circuit to optimally satisfy their design constraints in terms of area, performance and testability.



Ch1-23

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 2
Fault Modeling

Functional v.s. Structural Testing

- **I/O functional tests inadequate for manufacturing**
- **Exhaustive testing is prohibitively expensive**

Question: How to Generate Compact yet High-Quality Test Vectors?

Why Fault Model ?

- **Fault model identifies target faults**
 - Model faults most likely to occur
- **Fault model limits the scope of test generation**
 - Create tests only for the modeled faults
- **Fault model makes effectiveness measurable by experiments**
 - **Fault coverage can be computed** for specific test patterns to reflect its effectiveness
- **Fault model makes analysis possible**
 - Associate specific defects with specific test patterns

Scientific Study: Hypothesis (Assumption) → Evaluation → Refinement

Ch2-3

Fault Modeling

- **Fault Modeling**
 - Model the effects of physical defects on the logic function and timing
- **Physical Defects**
 - Silicon Defects
 - Photolithographic Defects
 - Mask Contamination
 - Process Variation
 - Defective Oxides

Ch2-4

Common Fault Types Used To Guide Test Generation

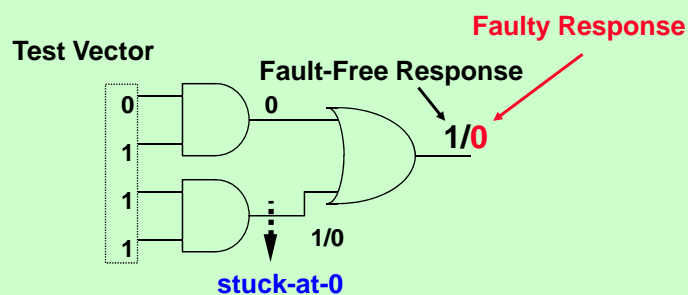
- Stuck-at Faults
- Bridging Faults
- Open Faults
- Transistor Stuck-On Faults
- Delay Faults
- IDDQ Faults (**Q**ui^{escent} current at VDD pin)
- Memory Faults

IDDQ Testing: canary in the coalmine, alarming of un-modeled defects

金絲雀

Ch2-5

Single Stuck-At Fault



Assumptions:

- Only One line is faulty
- Faulty line permanently set to 0 or 1
- Fault can be at an input or output of a gate

Ch2-6

Multiple Stuck-At Faults

- **Several stuck-at faults occur at the same time**
 - Mostly used in logic diagnosis
- **For a circuit with k lines**
 - there are $2k$ single stuck-at faults
 - there are $3^k - 1$ multiple stuck-at faults
 - A line could be **stuck-at-0**, **stuck-at-1**, or **fault-free**
 - One out of 3^k resulting circuits is fault-free

Ch2-7

Why Single Stuck-At Fault Model?

- **Complexity is greatly reduced**
 - Many different physical defects may be modeled by the same logical single stuck-at fault
- **Stuck-at fault is technology independent**
 - Can be applied to TTL, ECL, CMOS, BiCMOS etc.
- **Design style independent**
 - Gate array, standard cell, custom VLSI
- **Detection capability of un-modeled defects**
 - Empirically, many defects accidentally detected by test derived based on single stuck-at fault
- **Cover a large percentage of multiple stuck-at faults**

Single SA model survives well (due to its **simplicity** and **effectiveness**)

Ch2-8

Multiple Faults

- **Multiple stuck-fault coverage by single-fault tests of combinational circuit:**

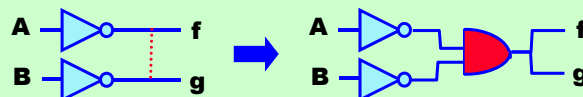
- 4-bit ALU (Hughes & McCluskey, **ITC-84**)
All double and most triple-faults covered.
- Large circuits (Jacob & Biswas, **ITC-87**)
Almost 100% multiple faults covered for circuits with 3 or more outputs.

Ch2-9

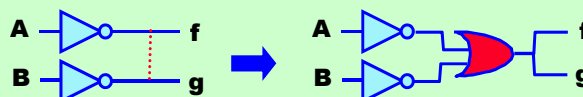
Bridging Faults

- **Two or more normally distinct points (lines) are shorted together erroneously**

- Logic effect depends on technology
- **Wired-AND for TTL**



- **Wired-OR for ECL**



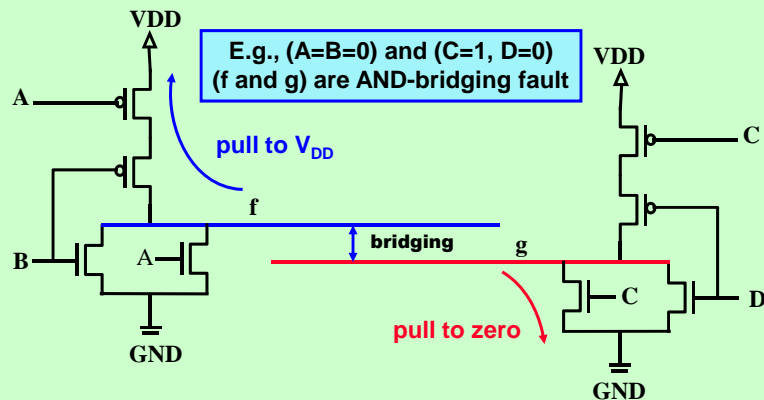
- **CMOS ?**

Ch2-10

Bridging Faults For CMOS Logic

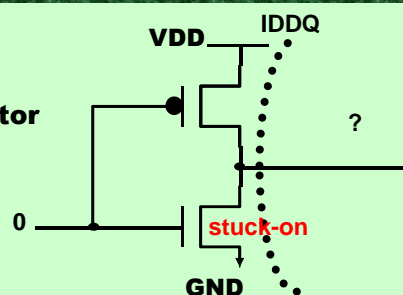
- **The result**

- could be AND-bridging or OR-bridging
- depends on the inputs



CMOS Transistor Stuck-On

Example:
N-type transistor
is always ON



- **Transistor Stuck-On**

- May cause **ambiguous logic level**
- Depends on the relative impedances of the pull-up and pull-down networks

- **When Input Is Low**

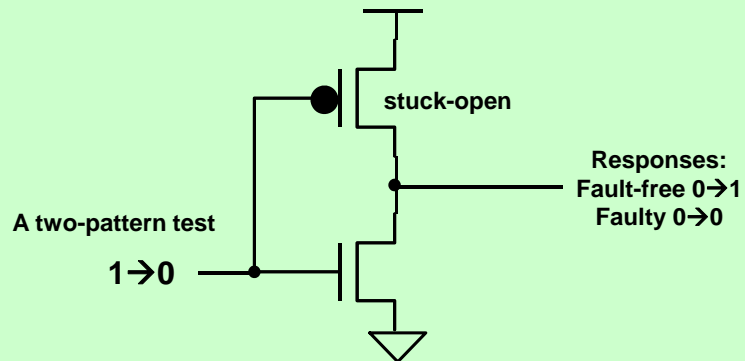
- Both P and N transistors are conducting, causing increased quiescent current, could be detected by $IDDQ$ test

Ch2-12

CMOS Transistor Stuck-Open (I)

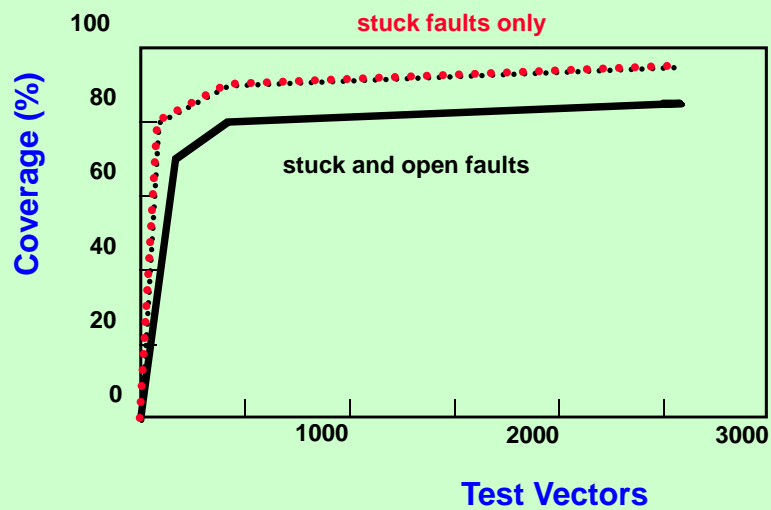
- **Transistor stuck-open**

- May cause the output to be **floating**
- The fault exhibits **sequential behavior**
- Need **two-pattern test** (to set it to a known value first)



Ch2-13

Fault Coverage in a CMOS Chip



Ch2-14

Summary of Stuck-Open Faults

- **First Report:**
 - Wadsack, Bell System Technology, J., 1978
- **Recent Results**
 - Woodhall et. al, **ITC-87** (1-micron CMOS chips)
 - 4552 chips passed the test
 - 1255 chips (27.57%) failed tests for stuck-at faults
 - 44 chips (0.97%) failed tests for stuck-open faults
 - **4 chips with stuck-open faults passed tests for stuck-at faults**
- **Conclusion**
 - **Stuck-at faults** are about 20 times more frequent than stuck-open faults
 - About 91% of chips with stuck-open faults may also have stuck-at faults
 - Faulty chips **escaping** tests for **stuck-at faults = 0.121%**

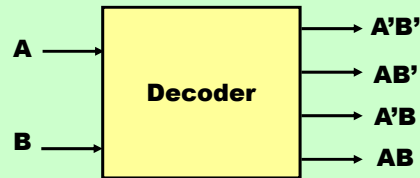
Ch2-15

Functional Faults

- **Fault effects modeled at a higher level than logic for functional modules, such as**
 - Decoder
 - Multiplexers
 - Adders
 - Counters
 - ROMs

Ch2-16

Functional Faults of Decoders



- **$f(L_i/L_k)$:** One active output, but wrong one
 - Instead of input line L_i , L_k is selected
- **$f(L_i/L_{i+k})$:** More than one active outputs
 - In addition to line L_i , L_k is also selected
- **$f(L_i/0)$:** No active output
 - None of the lines is selected

Ch2-17

Memory Faults

- **Parametric Faults**
 - Any fault that causes the response to deviate from its fault-free nominal value by some amount
 - Ex. A cell with parametric delay fault (with for example 93% more than normal)
 - Due to all kinds of factors like PVT variation
- **Functional Faults**
 - Stuck Faults in Address Register, Data Register, and Address Decoder
 - Cell Stuck Faults
 - Adjacent Cell Coupling Faults
 - Pattern-Sensitive Faults

Ch2-18

Memory Faults

- **Pattern-sensitive faults: the presence of a faulty signal depends on the signal values of the neighboring cells**

- Mostly in DRAMs

0	0	0
0	d	b
0	a	0

$a=b=0 \rightarrow d=0$

$a=b=1 \rightarrow d=1$

- **Adjacent cell coupling faults**
 - Pattern sensitivity between a pair of cells

Ch2-19

Memory Testing

- **Test could be time-consuming**
 - The length of the test sequence for memory testing could be **prohibitively long**
- **Example:**
 - A pattern sensitive test is **$5n^2$** long for an n-bit RAM
 - Testing a 1-M bit chip at 10ns pattern would take 14 hours
 - For a 64-M bit chip, it would take 6 years

Ch2-20

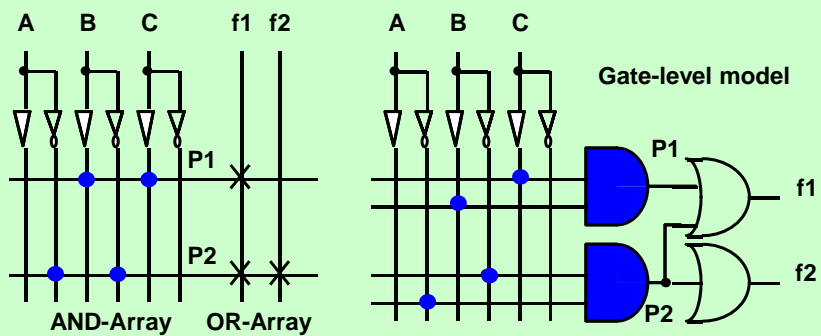
PLA Faults

- **Stuck-at Faults**
- **Cross-point Faults**
 - Extra/Missing Transistors
- **Bridging Faults**
- **Break Faults**

Ch2-21

Stuck-at Faults in PLA

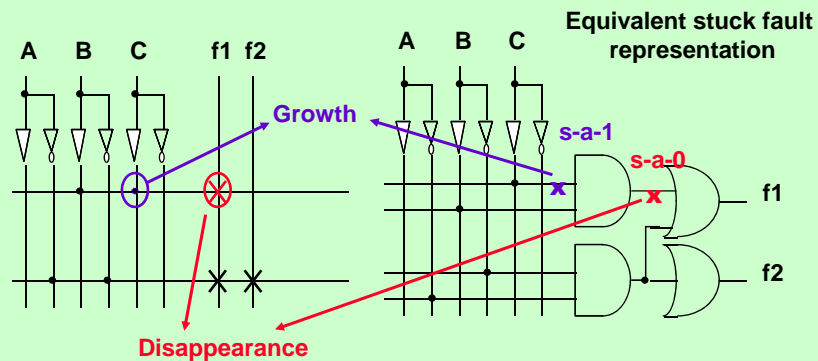
- **s-a-0 & s-a-1 faults**
 - on inputs, input inverters, product lines, and outputs are easy to simulate in its gate-level model



Ch2-22

Missing Cross-Point Faults in PLA

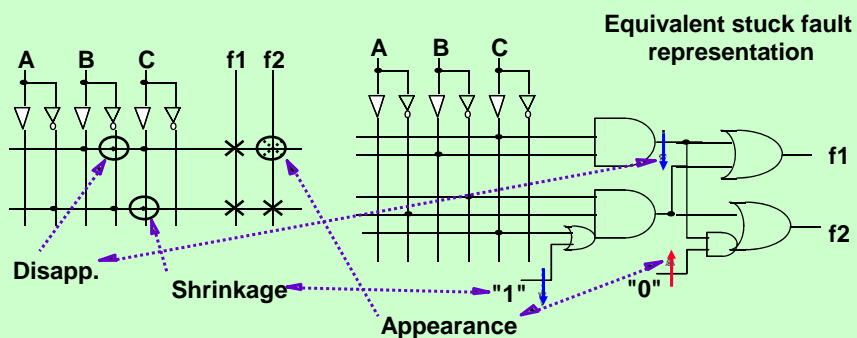
- **Missing Crosspoint in AND-array**
 - Growth Fault
- **Missing Crosspoint in OR-array**
 - Disappearance fault



Ch2-23

Extra Cross-Point Faults in PLA

- **Extra cross-point in AND-array**
 - Shrinkage or disappearance fault
- **Extra cross-point in OR-array**
 - Appearance fault



Ch2-24

Summary of PLA Faults

- **Cross-Point Faults**

- 80 ~ 85% covered by stuck-fault tests
- **Layout-dependence** in folded PLA

- **Bridging Faults**

- 99% covered by stuck-fault tests
- **Layout-dependence** in all PLAs
- (Ref: Agrawal & Johnson, **ICCD-86**)

Ch2-25

Delay Testing

- **Chip with Timing Defects**

- may pass the DC stuck-fault testing, but **fail when operated at the system speed**
- For example, a chip may pass the test under 10 MHz operation, but fail under 100 MHz

- **Delay Fault Models**

- Gate-Delay Fault
- Path-Delay Fault

Ch2-26

Downloaded from <http://ajph.org/> on November 10, 2014

- \bar{x} is slow to rise when channel resistance R_1 is

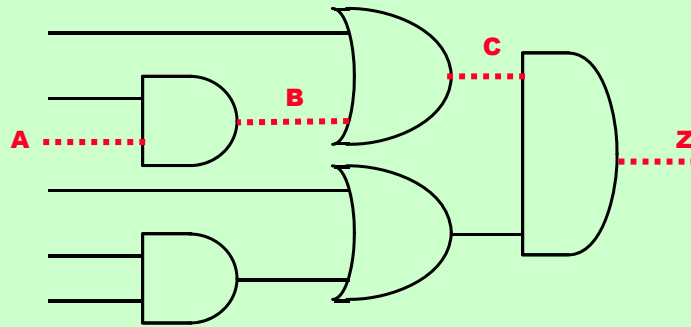
- VDD VDD

Downloaded from <http://ajph.org/> on November 10, 2014

- May not detect those delay faults that result

Path-Delay Fault

- **Associated with a Path (e.g., A-B-C-Z)**
 - Whose delay exceeds the clock interval
- **More complicated than gate-delay fault**
 - Because the number of paths grows exponentially



Ch2-29

Fault Detection

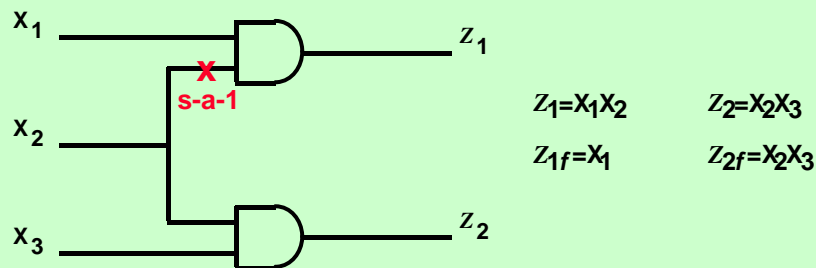
- **Fault Activation**
- **Fault Propagation**

Definition Of Fault Detection

- A test (vector) t detects a fault f iff

- t detects $f \Leftrightarrow z(t) \neq z_f(t)$

- Example



The test $(x_1, x_2, x_3) = (100)$ detects f because $z_1(100)=0$ while $z_{1f}(100)=1$

Ch2-31

Fault Detection Requirement

- A test t that detects a fault f

- (1) **Activate** f (or generate a fault effect at the site of the fault)
 - (2) **Propagate** the fault effect to a primary output w

- Sensitized Line:

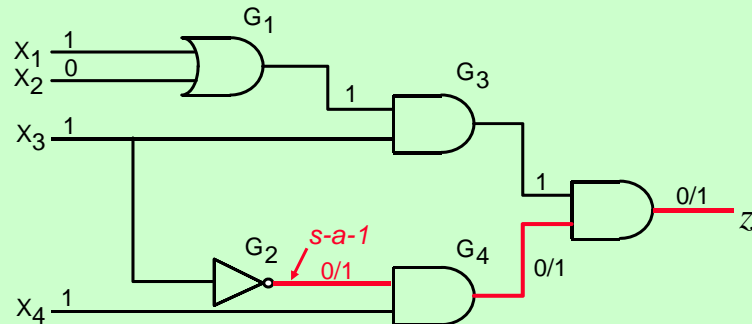
- A line whose faulty value is different from its fault-free one is said to be sensitized by the test in the faulty circuit

- Sensitized Path:

- A path composed of sensitized lines is called a sensitized path

Ch2-32

Fault Sensitization



$z(1011)=0$ $z_f(1011)=1$
 1011 detects the fault f (G_2 stuck-at 1)
 v/v_f : v = signal value in the fault free circuit
 v_f = signal value in the faulty circuit

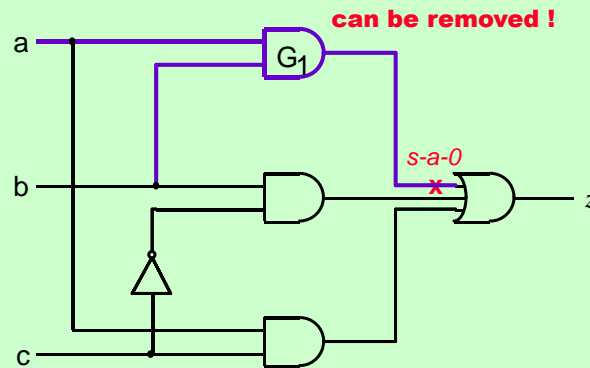
Ch2-33

Detectability

- **A fault f is said to be detectable**
 - if there exists a test t that detects f ;
 otherwise,
 f is an **undetectable fault**
- **For an undetectable fault f**
 - No test can **simultaneously** activate f and
 create a sensitized path to a primary output

Ch2-34

Undetectable Fault



- **G₁ output stuck-at-0 fault is undetectable**
 - Undetectable faults do not change the function of the circuit
 - The **related circuit can be deleted** to simplify the circuit

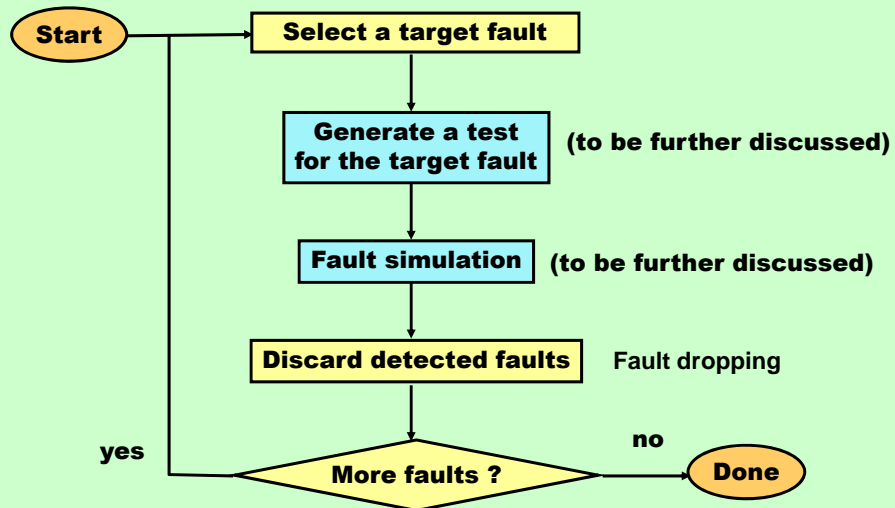
Ch2-35

Test Set

- **Complete detection test set:**
 - A set of tests that detect any detectable faults in a class of faults
- **The quality of a test set**
 - is measured by fault coverage
- **Fault coverage:**
 - Fraction of faults that are detected by a test set
- **The fault coverage**
 - can be determined by fault simulation
 - >95% is typically required for single stuck-at fault model
 - >99.9% in IBM

Ch2-36

Typical Test Generation Flow



Ch2-37

Fault Collapsing

- Fault Equivalence
- Fault Dominance
- Checkpoint Theorem

Fault Equivalence

- **Distinguishing test**

- A test t distinguishes faults α and β if

$$Z_{\alpha}(t) \oplus Z_{\beta}(t) = 1$$

- **Equivalent Faults**

- Two faults, α & β are said to be equivalent in a circuit, iff the function under α is equal to the function under β for **any input combination** (sequence) of the circuit.
- No test can distinguish between α and β

Ch2-39

Fault Equivalence

- **AND gate:**

- all $s-a-0$ faults are equivalent

- **OR gate:**

- all $s-a-1$ faults are equivalent

- **NAND gate:**

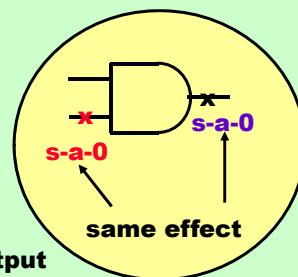
- all the input $s-a-0$ faults and the output $s-a-1$ faults are equivalent

- **NOR gate:**

- all input $s-a-1$ faults and the output $s-a-0$ faults are equivalent

- **Inverter:**

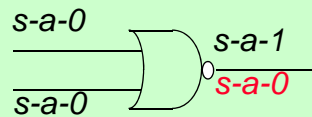
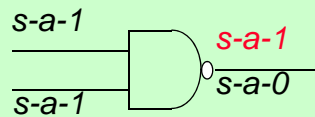
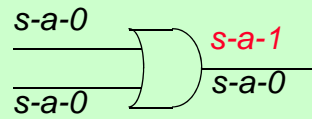
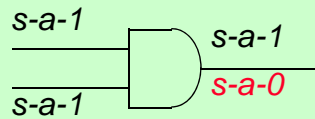
- input $s-a-1$ and output $s-a-0$ are equivalent
- input $s-a-0$ and output $s-a-1$ are equivalent



Ch2-40

Equivalence Fault Collapsing

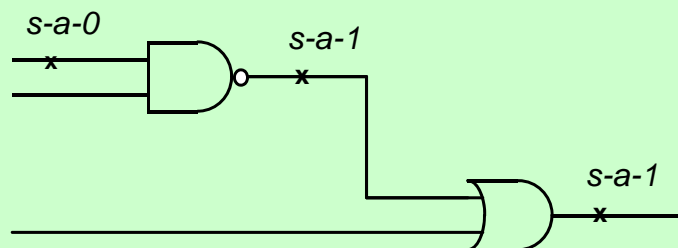
- $n+2$ instead of $2(n+1)$ faults need to be considered for n -input gates



Ch2-41

Equivalent Fault Group

- In a combinational circuit
 - Many faults may form an equivalent group
 - These equivalent faults can be found by **sweeping the circuit from the primary outputs to the primary inputs**



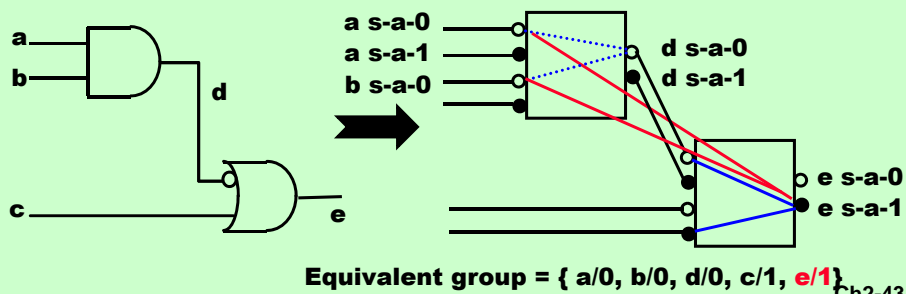
Three faults shown are equivalent !

Ch2-42

Finding Equivalent Group

- **Construct a Graph**

- Sweeping the netlist from PO's to PI's
- When a fault α is equivalent to a fault β , then an edge is connected between them
- **Transitive Rule:**
 - When α connects β and β connects γ , then α connects γ

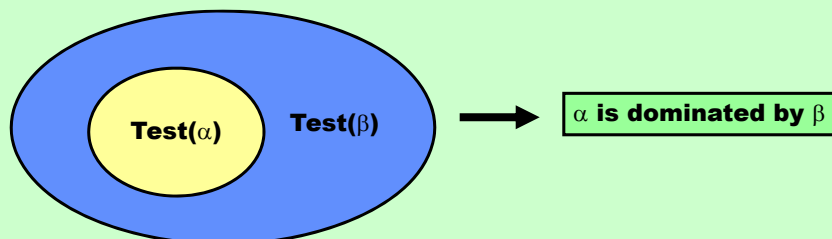


Ch2-43

Fault Dominance

- **Dominance Relation**

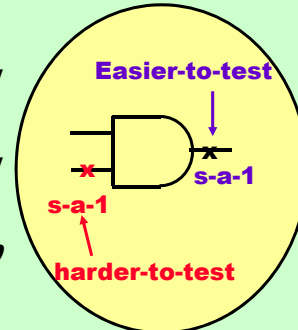
- A fault β is said to **dominate** another fault α in a circuit, iff every test (sequence) for α is also a test (sequence) for β .
- I.e., **test-set(β) > test-set(α)**
- No need to consider fault β for fault detection



Ch2-44

Fault Dominance

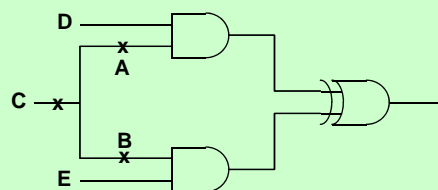
- **AND gate:**
 - Output *s-a-1* dominates any input *s-a-1*
- **NAND gate:**
 - Output *s-a-0* dominates any input *s-a-1*
- **OR gate:**
 - Output *s-a-0* dominates any input *s-a-0*
- **NOR gate:**
 - Output *s-a-1* dominates any input *s-a-0*
- **Dominance fault collapsing:**
 - The reduction of the set of faults to be analyzed based on dominance relation



Ch2-45

Stem v.s. Branch Faults

C: stem of a multiple fanout
A & B: branches



- **Detect A sa1:**

$$z(t) \oplus z_f(t) = (CD \oplus CE) \oplus (D \oplus CE) = D \oplus CD = 1$$

$$\Rightarrow (C=0, D=1)$$

- **Detect C sa1:**

$$z(t) \oplus z_f(t) = (CD \oplus CE) \oplus (D \oplus E) = 1$$

$$\Rightarrow (C=0, D=1) \text{ or } (C=0, E=1)$$

- **Hence, C sa1 dominates A sa1**

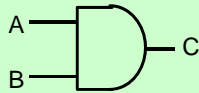
- **Similarly**

- C sa1 dominates B sa1
- C sa0 dominates A sa0
- C sa0 dominates B sa0

- **In general, there might be no equivalence or dominance relations between stem and branch faults**

Ch2-46

Analysis of a Single Gate



AB	C	A	B	C	A	B	C
		sa1	sa1	sa1	sa0	sa0	sa0
00	0			1			
01	0	1		1			
10	0		1	1			
11	1				0	0	0

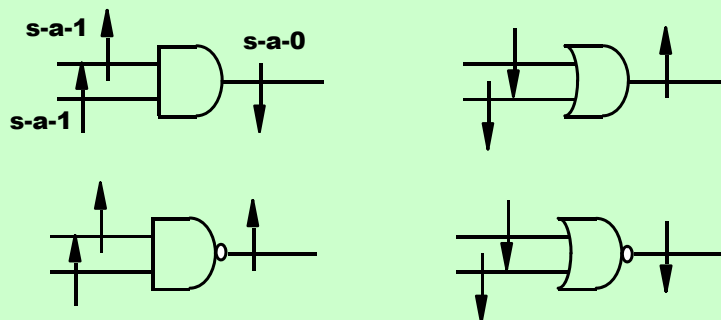
Negligible fault

- **Fault Equivalence Class**
 - (A s-a-0, B s-a-0, C s-a-0)
- **Fault Dominance Relations**
 - (C s-a-1 > A s-a-1) and (C s-a-1 > B s-a-1)
- **Faults that can be ignored:**
 - A s-a-0, B s-a-0, and C s-a-1

Ch2-47

Fault Collapsing

- **Equivalence + Dominance**
 - For each n -input gate, we only need to consider $n+1$ faults during test generation



Ch2-48

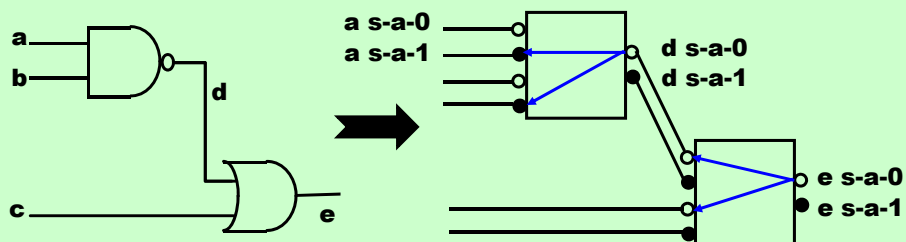
Dominance Graph

- **Rule**

- When fault α dominates fault β , then an arrow is pointing from α to β

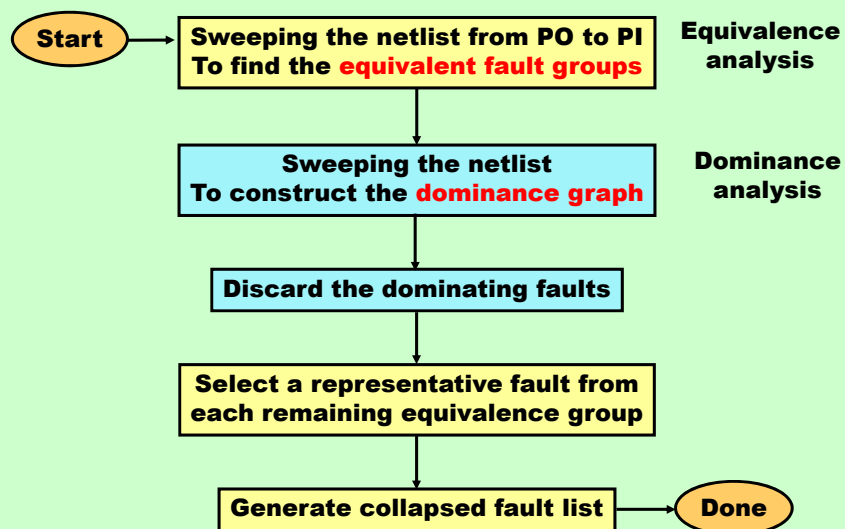
- **Application**

- Find out the **transitive dominance relations** among faults



Ch2-49

Fault Collapsing Flow



Ch2-50

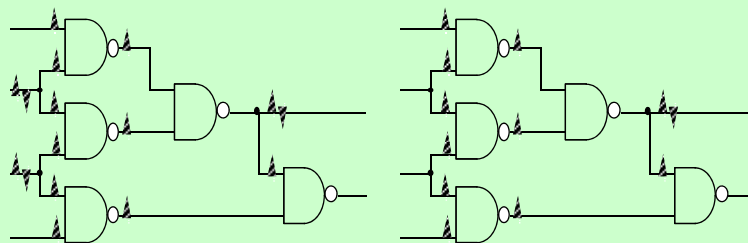
Prime Fault

- α is a prime fault if every fault that is dominated by α is also equivalent to α

Ch2-51

Why Fault Collapsing ?

- Memory and CPU-time saving
- Ease testing generation and fault simulation



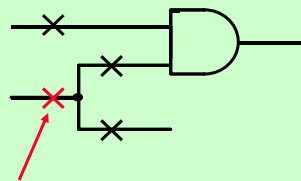
* 30 total faults → 12 prime faults

Ch2-52

Checkpoint Theorem

- Checkpoints for test generation

- A test set detects every fault on the **primary inputs** and **fanout branches** is complete
- I.e., this test set detects all other faults too
- Therefore, **primary inputs** and **fanout branches** form a *sufficient* set of checkpoints in test generation
- In fanout-free combinational circuits, primary inputs are the sole checkpoints



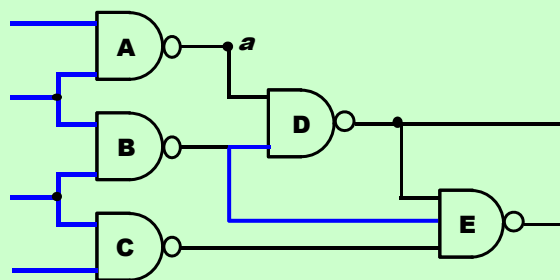
Stem is not a checkpoint !

Ch2-53

Why Inputs + Branches Are Enough ?

- Example

- Checkpoints are marked in blue
- Sweeping the circuit **from PI to PO** to examine every gate, e.g., based on an order of (A->B->C->D->E)
- For each gate, **output faults are detected if every input fault is detected**

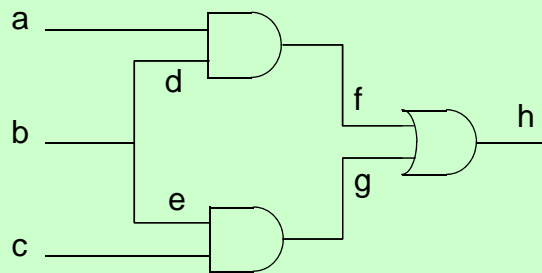


Ch2-54

Fault Collapsing + Checkpoint

- **Example:**

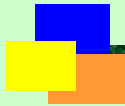
- 10 checkpoint faults
- $a \text{ s-a-0} \Leftrightarrow d \text{ s-a-0}$, $c \text{ s-a-0} \Leftrightarrow e \text{ s-a-0}$
 $b \text{ s-a-0} > d \text{ s-a-0}$, $b \text{ s-a-1} > d \text{ s-a-1}$
- 6 tests are enough



Ch2-55

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 3
Fault Simulation

Outline

- ➔ • Fault Simulation for Comb. Ckt
 - Basic of Logic Simulation
 - Parallel Fault Simulation
 - Deductive Fault Simulation
 - Concurrent Fault Simulation
- Approximation Approach
- Techniques for Sequential Circuits

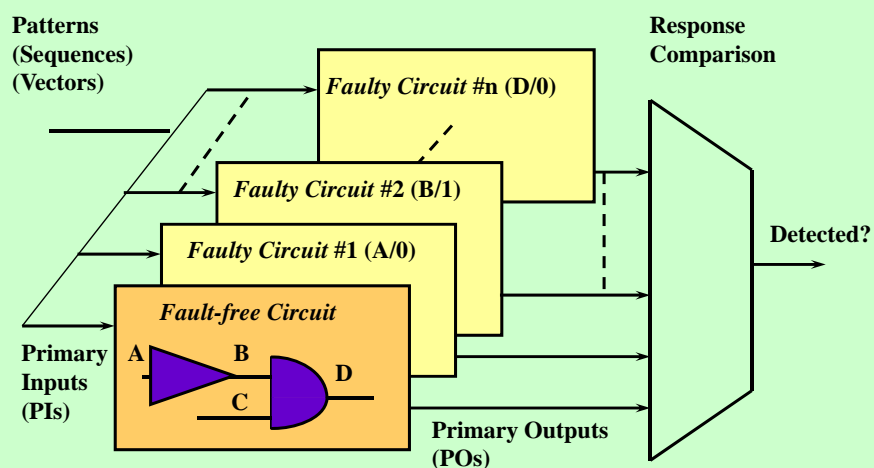
Note: Comb. Ckt: Combinational Circuits

Why Fault Simulation ?

- To evaluate the quality of a test set
 - I.e., to compute its fault coverage
- Part of an ATPG program
 - A vector usually detects multiple faults
 - Fault simulation is used to compute the faults **accidentally** detected by a particular vector
- To construct fault-dictionary
 - For post-testing diagnosis
- To Evaluate the fault coverage of a functional patterns

Ch3-3

Conceptual Fault Simulation



Logic simulation on both good (fault-free) and faulty circuits

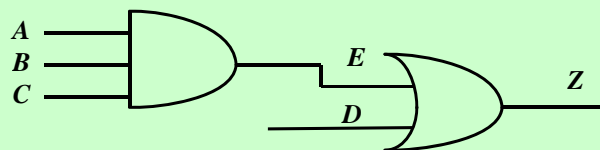
Ch3-4

Some Basics for Logic Simulation

- For fault simulation purpose,
 - mostly the **gate delay is assumed to be zero** unless the delay faults are considered. Our main concern is the **functional faults**
- The logic values
 - can be either two (0, 1) or **three values (0, 1, X)**
- Two simulation mechanisms:
 - Oblivious **compiled-code**:
 - circuit is translated into a program and **all** gates are executed for each pattern. (may have **redundant computation**)
 - Interpretive **event-driven**:
 - Simulating a vector is viewed as a sequence of **value-change events** propagating from the PI's to the PO's
 - Only those logic gates affected by the events are **re-evaluated**

Ch3-5

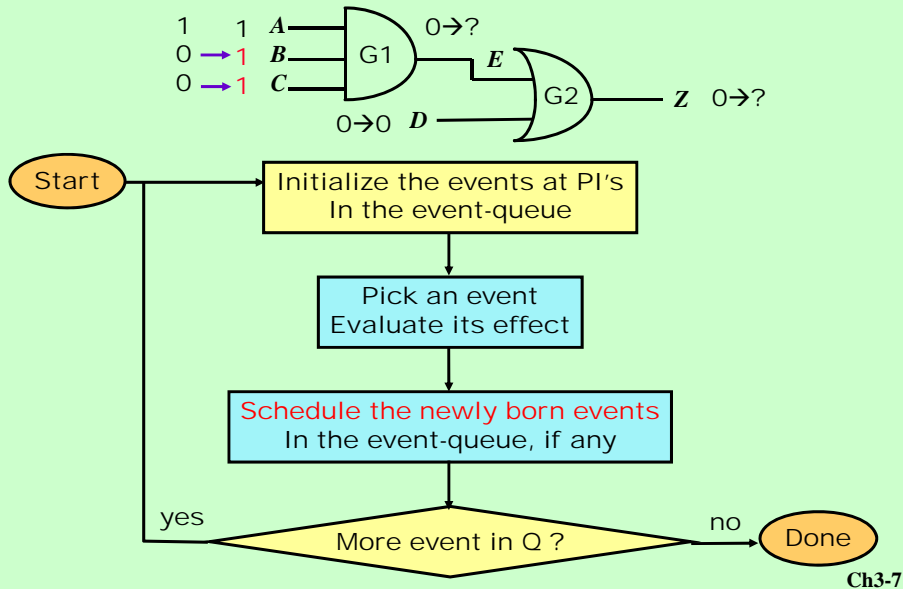
Compiled-Code Simulation



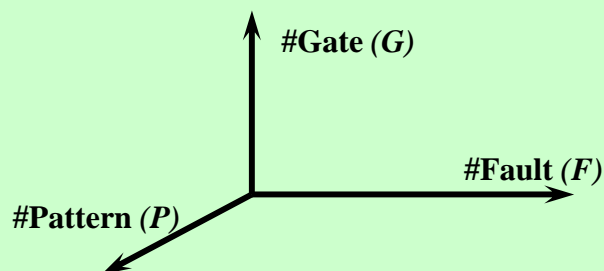
- Compiled code
 - LOAD *A* /* load accumulator with value of *A* */
 - AND *B* /* calculate *A and B* */
 - AND *C* /* calculate *E = AB and C* */
 - OR *D* /* calculate *Z = E or D* */
 - STORE *Z* /* store result of *Z* */

Ch3-6

Event-Driven Simulation



Complexity of Fault Simulation

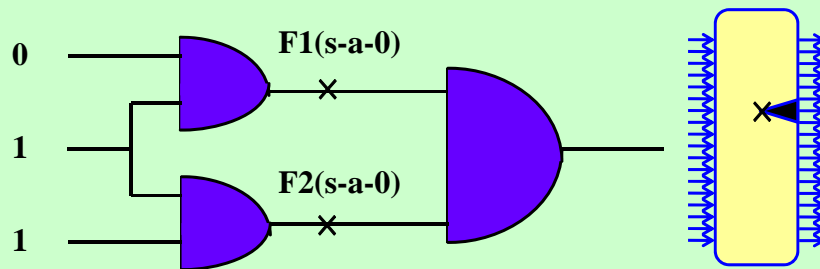


- Complexity $\sim F \cdot P \cdot G \sim O(G^3)$, where G is the no. of gates
- The complexity is higher than logic simulation by a factor of F , while usually is much lower than ATPG
- The complexity can be greatly reduced using
 - **Fault dropping** and other advanced techniques

Ch3-8

Characteristics of Fault Simulation

- Fault activity with respect to fault-free circuit
 - is often **sparse** both in **time** and in **space**.
- For example
 - F1 is not activated by the given pattern, while F2 affects only the lower part of this circuit.



Ch3-9

Fault Simulation Techniques

- Serial Fault Simulation
 - trivial single-fault single-pattern
- Parallel Fault Simulation
- Deductive Fault Simulation
- Concurrent Fault Simulation

Ch3-10

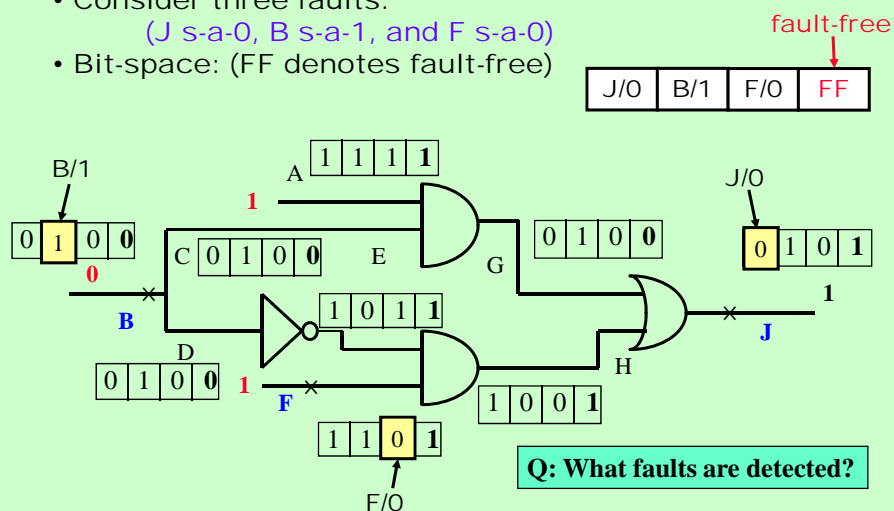
Parallel Fault Simulation

- Simulate multiple circuits at a time:
 - The inherent parallel operation of **computer words** to simulate faulty circuits in parallel with fault-free circuit
 - **The number of faulty circuits**, or faults, can be processed simultaneously is limited by the word length, e.g., **32** circuits for a 32-bit computer
- Extra Cost:
 - An **event**, a value-change of a single fault or fault-free circuit leads to the computation of the entire word
 - The **fault-free logic simulation** is repeated for each pass

Ch3-11

Example: Parallel Fault Simulation

- Consider three faults: (J s-a-0, B s-a-1, and F s-a-0)
- Bit-space: (FF denotes fault-free)



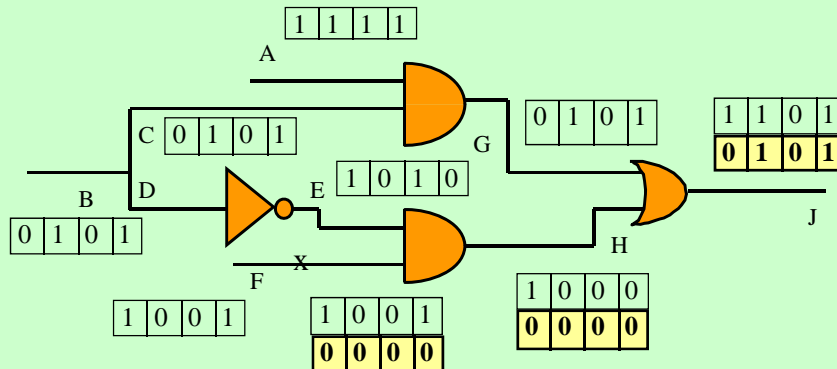
Ch3-12

Example: Parallel-Pattern Simulation

- Consider one fault F/0 and four patterns: P3,P2,P1,P0

Bit-Space:

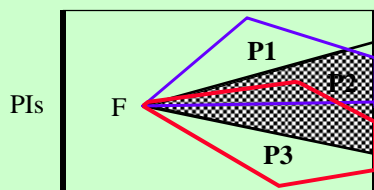
P3	P2	P1	P0
----	----	----	----



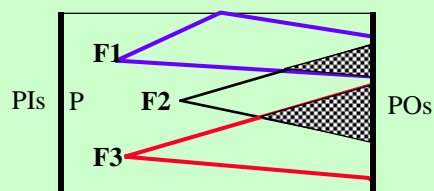
Ch3-13

Parallel-Pattern v.s. Parallel-Fault

Parallel-pattern



Parallel-fault



- P1, P2, P3 are patterns events
- F1, F2, F3 are faults
- Complexity**
 - Is proportional to the **events** that need to be processed
 - The **value-change events** (upper figure) seems to be fewer than the **fault-events** (lower figure)
 - Hence, **parallel-pattern** seems to be more efficient than **parallel-fault** methods

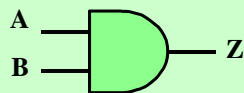
Ch3-14

Deductive Fault Simulation

- Simulate all faulty circuits in one pass
 - For each pattern, sweep the circuit from PI's to PO's.
 - During the process, a list of faults is associated with each line
 - The list contains faults that would produce a **fault effect** on this line
 - The **union fault list at every PO** contains the detected faults by the simulated input vector
- Major operation: fault list propagation
 - Related to the gate types and values
 - The size of the list may grow dynamically, leading to a potential **memory explosion problem**

Ch3-15

Controlling Value of a Logic Gate



Whenever there is a '0' in the inputs, Z will be '0'
→ Controlling value for NAND gate is '0'
→ Non-Controlling value is '1'

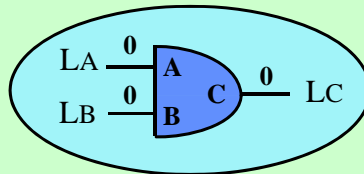
Gate Type	Controlling Value	Non-Controlling Value
AND	'0'	'1'
OR	'1'	'0'
NAND	'0'	'1'
NOR	'1'	'0'

Ch3-16

Example: Fault List Propagation

Fault-free simulation results: $\{A=0, B=0, C=0\}$
Q: What is the detected fault list at line C?

- (Reasoning) To create a fault effect at line C, we need $\{A=1, B=1\}$
 → which means that we need a fault effect at A as well as B
 → It can be achieved in faulty circuits $LA \cdot LB$
 → Also C/1 is a new fault to be included in the fault list of C



LA, LB, LC are fault list propagated to their respective lines

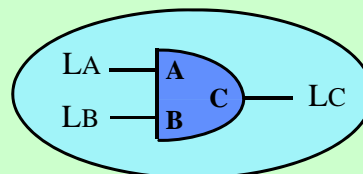
\overline{LA} is the set of all faults not in LA

Ch3-17

Example: Fault List Propagation

LA, LB, LC are detected fault list at their respective lines

Consider a two-input AND-gate:



Non-controlling case: **Case 1:** $A=1, B=1, C=1$ at fault-free,
 $LC = LA + LB + \{C/0\}$

Controlling cases: **Case 2:** $A=1, B=0, C=0$ at fault-free,
 $LC = \overline{LA} \cdot LB + \{C/1\}$

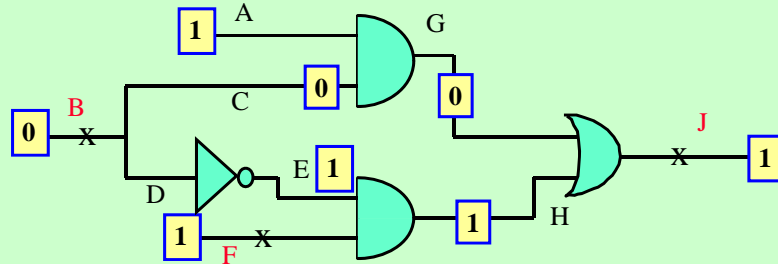
Case 3: $A=0, B=0, C=0$ at fault-free,
 $LC = LA \cdot LB + \{C/1\}$

\overline{LA} is the set of all faults not in LA

Ch3-18

Example: Deductive Simulation (1)

- Consider 3 faults: B/1, F/0, and J/0



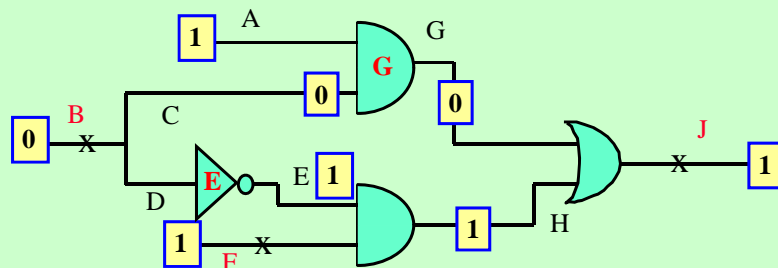
Fault List at PI's:

$$\mathbf{LB = \{B/1\}, \quad LF = \{F/0\}, \quad LA = \phi, \quad LC=LD = \{B/1\}}$$

Ch3-19

Example: Deductive Simulation (2)

- Consider 3 faults: B/1, F/0, and J/0



Fault Lists at G and E:

$$\mathbf{LB = \{B/1\}, \quad LF = \{F/0\}, \quad LA = \phi, \quad LC=LD = \{B/1\},}$$

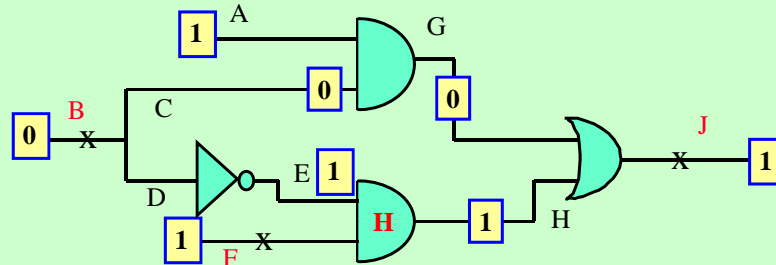
$$\mathbf{LG = (\overline{LA} * LC) = \{B/1\}}$$

$$\mathbf{LE = (LD) = \{B/1\}}$$

Ch3-20

Example: Deductive Simulation (3)

- Consider 3 faults: B/1, F/0, and J/0



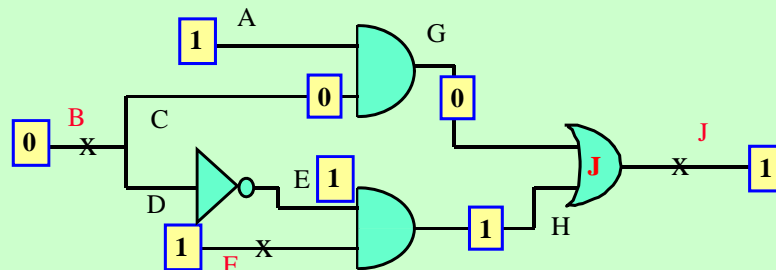
Computed Fault List at H:

$$\begin{aligned} \mathbf{LB} &= \{\mathbf{B/1}\}, \quad \mathbf{LF} = \{\mathbf{F/0}\}, \quad \mathbf{LC=LD} = \{\mathbf{B/1}\}, \\ \mathbf{LG} &= \{\mathbf{B/1}\}, \quad \mathbf{LE} = \{\mathbf{B/1}\} \\ \mathbf{LH} &= (\mathbf{LE} + \mathbf{LF}) = \{\mathbf{B/1, F/0}\} \end{aligned}$$

Ch3-21

Example: Deductive Simulation (4)

- Consider 3 faults: B/1, F/0, and J/0



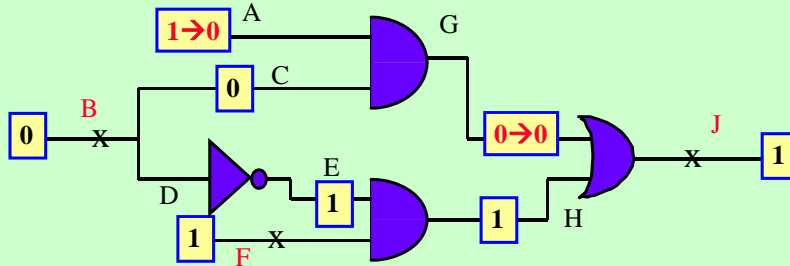
Final Fault List at the output J:

$$\begin{aligned} \mathbf{LB} &= \{\mathbf{B/1}\}, \quad \mathbf{LF} = \{\mathbf{F/0}\}, \quad \mathbf{LC=LD} = \{\mathbf{B/1}\}, \\ \mathbf{LG} &= \{\mathbf{B/1}\}, \quad \mathbf{LE} = \{\mathbf{B/1}\} \\ \mathbf{LH} &= \{\mathbf{B/1, F/0}\}, \\ \mathbf{LJ} &= (\overline{\mathbf{LG}} \cdot \mathbf{LH}) \{\mathbf{F/0, J/0}\} \end{aligned}$$

Ch3-22

Example: Even-Driven Deductive Fault Simulation

- When A changes from 1 to 0



Event-driven operation:

$LB = \{B/1\}$, $LF = \{F/0\}$, $LA = \phi$

$LC=LD = \{B/1\}$, $LG = \phi$,

$LE = \{B/1\}$, $LH = \{B/1, F/0\}$, $LJ = \{B/1, F/0, J/0\}$

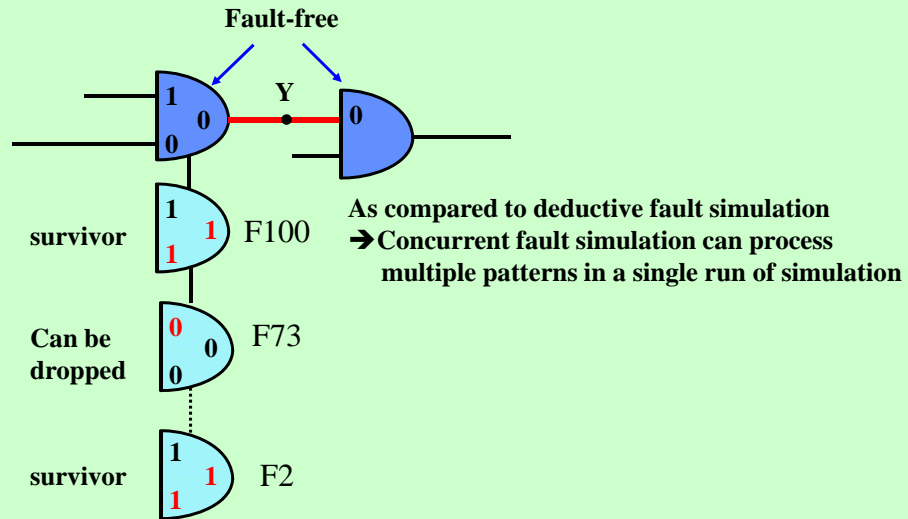
Ch3-23

Concurrent Fault Simulation

- Simulate all faulty circuits in one pass:
 - Each gate retains a list of fault copies, each of which stores the status of a fault exhibiting difference from fault-free values
- Simulation mechanism
 - is similar to the conceptual fault simulation except that only the dynamical difference w.r.t. fault-free circuit is retained.
- Theoretically,
 - all faults in a circuit can be processed in one pass
- Practically,
 - memory explosion problem may restrict the number of faults that can be processed in each pass

Ch3-24

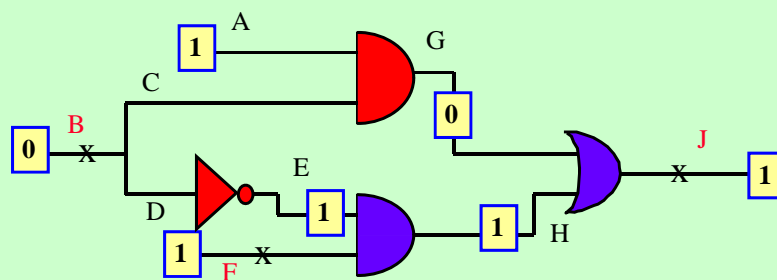
Concurrent Fault Simulation



Ch3-25

Example: Concurrent Simulation (1)

- Consider 3 faults: B/1, F/0, and J/0



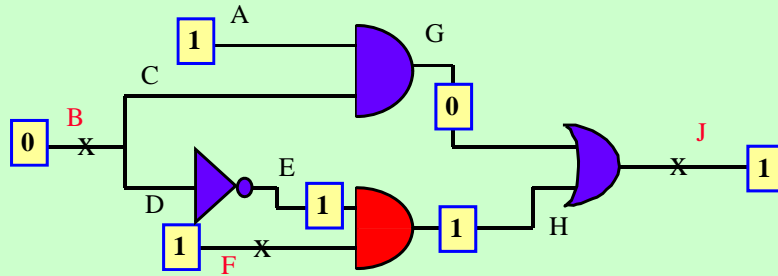
$LG = \{10_0, B/1:11_1\}$ $LE = \{0_1, B/1:1_0\}$

Fault Free A fault B/1

Ch3-26

Example: Concurrent Simulation (2)

- Consider 3 faults: B/1, F/0, and J/0

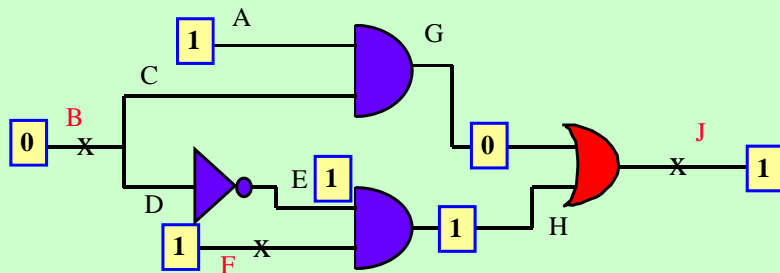


LG = {10_0, B/1:11_1} LE = {0_1, B/1:1_0}
 LH = {11_1, B/1:01_0, F/0:10_0}

Ch3-27

Example: Concurrent Simulation (3)

- Consider 3 faults: B/1, F/0, and J/0



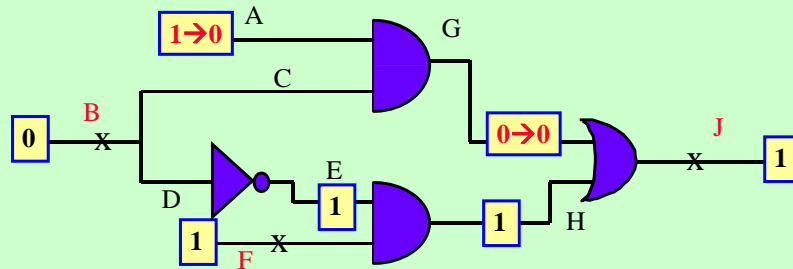
LG = {10_0, B/1:11_1} LE = {0_1, B/1:1_0}
 LH = {11_1, B/1:01_0, F/0:10_0}
 LJ = {01_1, B/1:10_1, F/0:00_0, J/0:01_0}

dropped

Ch3-28

Example: Concurrent Simulation (4)

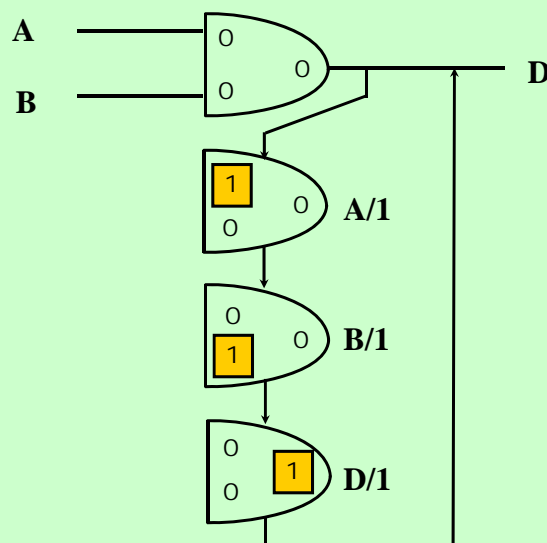
- When A changes from 1 to 0



$LG = \{00_0, B/1:01_0\}$ $LE = \{0_1, B/1:1_0\}$
 $LH = \{11_1, B/1:01_0, F/0:10_0\}$
 $LJ = \{01_1, B/1:00_0, F/0:00_0, J/0:01_0\}$

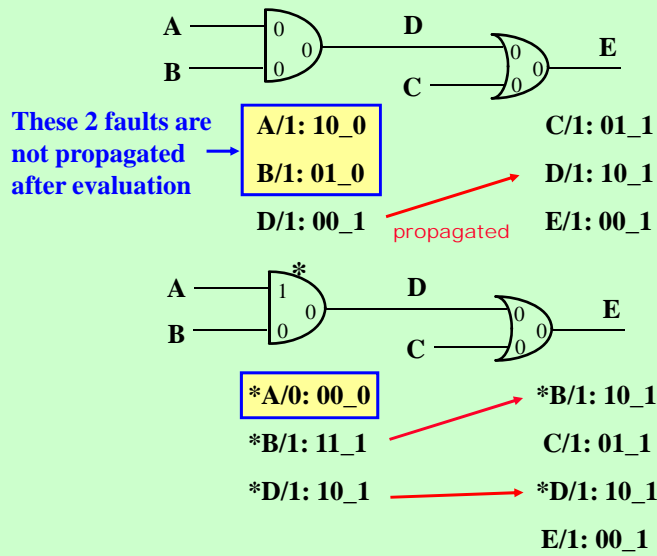
Ch3-29

Fault List Including New Borns



Ch3-30

Fault List Propagation



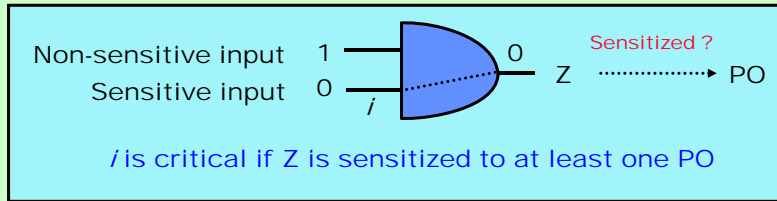
Ch3-31

Outline

- Fault Simulation for Comb. Circuits
- ➔ • Approximation Approach
 - Critical Path Tracing
 - Probabilistic Approach
- Techniques for Sequential Circuits

Ch3-32

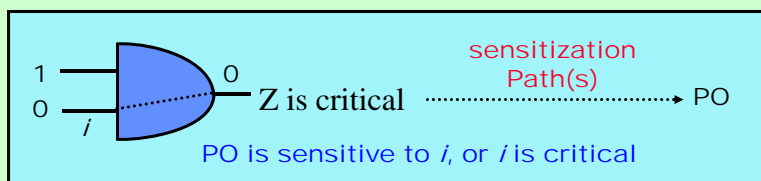
Sensitive Input and Critical Path



- Sensitive Input of a gate:
 - A gate input i is *sensitive* if complementing the value of i changes the value of the gate output
- Critical line
 - Assume that the fault-free value of w is v in response to t
 - A line w is *critical* w.r.t. a pattern t iff t detects the fault w stuck-at \bar{v}
- Critical paths
 - Paths consisting of critical lines only

Ch3-33

Basics of Critical Path Tracing



- A gate input i is critical w.r.t. a pattern t if
 - (1) the gate *output is critical* and
 - (2) i is a *sensitive input* to t
 - Use recursion to prove that i is also critical
- In a fanout-free circuit
 - the criticality of a line can be determined by *backward traversal* to the sensitive gate's inputs from PO's, *in linear time*

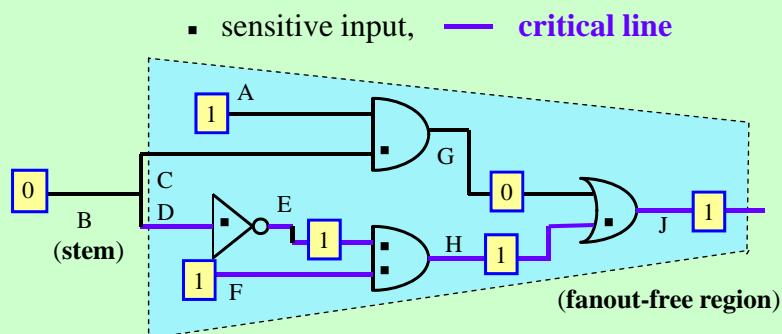
Ch3-34

Analysis of Critical Path Tracing

- Three-step Procedure:
 - Step 1: Fault-free simulation
 - Step 2: Mark the **sensitive inputs** of each gate
 - Step 3: Identification of the **critical lines** by backward critical path tracing)
- Complexity is $O(G)$
 - Where G is the gate count
 - for fanout-free circuits --- very rare in practice
- Application
 - Applied to **fanout-free regions**, while stem faults are still simulated by parallel-pattern fault simulator.

Ch3-35

Example of Critical Path Tracing



Detected faults in the fanout-free region:

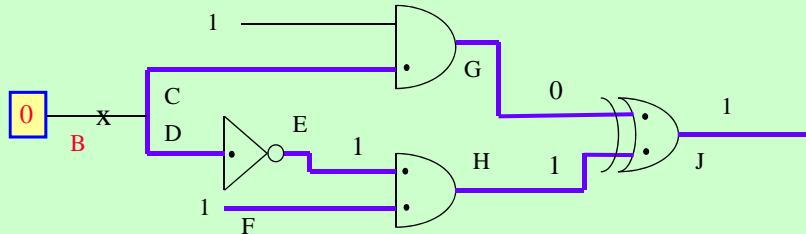
{J/0, H/0, F/0, E/0, D/1}

Question: is B stuck-at-1 detected ?

Ch3-36

Anomaly of Critical Path Tracing

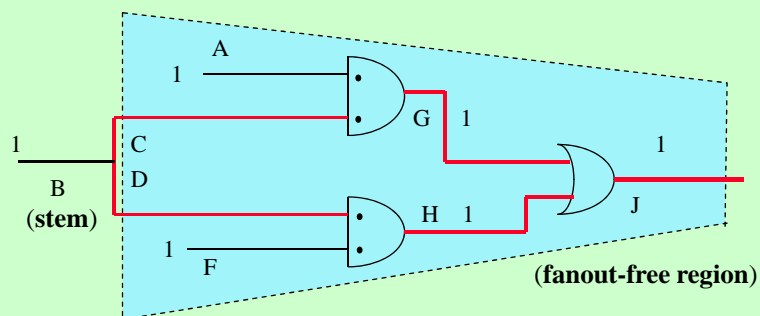
- **Stem criticality** is hard to infer from branches.
E.g. is B/1 detectable by the given pattern?



- It turns out that **B/1 is not detectable** even though both C and D are critical, because their effects cancel out each other at gate J, (i.e., **fault masking problem**)
- There is also a so-called **multiple path sensitization problem**.

Ch3-37

Multiple Path Sensitization



Both C and D are not critical, yet **B is critical** and B/0 can be **detected at J by multiple path sensitization**.

Ch3-38

Parallel and Distributed Simulation

- To share the fault simulation effort
 - by a number of processors either **tightly** connected as in parallel computation or **loosely** connected as in distributed computation.
- The speed-up
 - with respect to the processor number depends on the **degree of duplicated computation**, and the **communication overhead** among processors.
- The distributed simulation
 - on a cluster of **networked workstations** is especially appealing.

Ch3-39

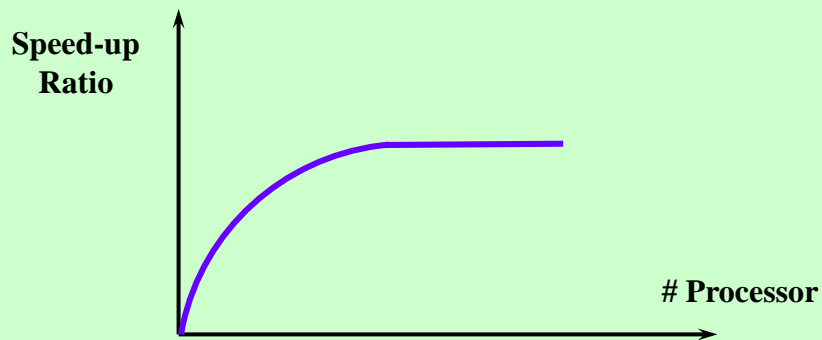
Distributed Simulation Techniques

- Fault Partition
 - **Distributes faults** among many processors.
 - Works relatively well for both combinational and sequential circuits.
- Pattern Partition
 - **Distributes patterns** among processors.
 - no duplicated logic simulation
 - Works well for combinational circuits.
- Circuit Partition
 - Difficult to achieve synchronization without incurring excessive communication overhead.

Ch3-40

Distributed Fault Simulation

- Typical Speed-up versus No. of Processors



- Diminished increase of speed-up ratio with more processors

Ch3-41

Fault Grading

- Approximate fault coverage
 - Can be obtained in much shorter computational time than regular fault simulation.
 - Not suitable for high fault-coverage requirement.
- Typical fault grading methods:
 - Toggle test, e.g. DATAS
 - Detection probability computation, e.g. STAFAN
 - Fault sampling
 - estimate from a selected subset of total faults
 - Test set sampling
 - estimate from a subset of complete test sequence

Ch3-42

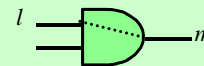
STAFAN

- Compute fault detection probability from logic simulation.
 - d_l = detection probability of s-a-0 on l = $C1(l)O(l)$
 - d_l = detection probability of s-a-1 on l = $C0(l)O(l)$

$$C0(l) = \frac{0\text{-count}}{n}, \quad C1(l) = \frac{1\text{-count}}{n}$$

$$S(l) = \frac{\text{sensitization-count}}{n}$$

$$O(l) = S(l)O(m)$$



- **m** is the immediate successor of **l**
- **observability** can be computed backwards from POs

Ch3-43

STAFAN (cont.)

$$d_f^n = 1 - (1 - d_f)^n \quad n \text{ is the no. of vectors}$$

$$\text{Statistical Fault Coverage} = \frac{\sum_{\Phi} d_f^n}{|\Phi|}$$

the summation of each fault's detection probability

Φ is the set of faults of interest

- More sophisticated than toggle test with same computation complexity

Ch3-44

Outline

- Fault Simulation for Comb. Circuits
- Approximation Approach
 - Toggle Counting
 - Critical Path Tracing
 - Probabilistic Approach
- ➔ • Techniques for Sequential Circuits

Ch3-45

Fault Grading for Functional Input Sequence

Inputs:

- (1) A test application program
- (2) A sequential design

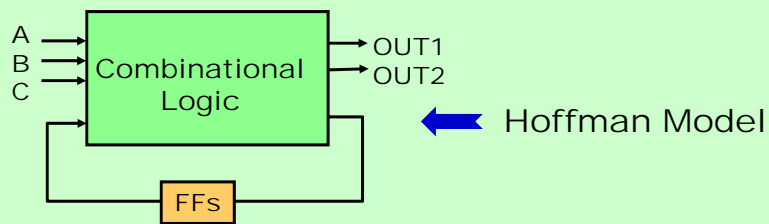
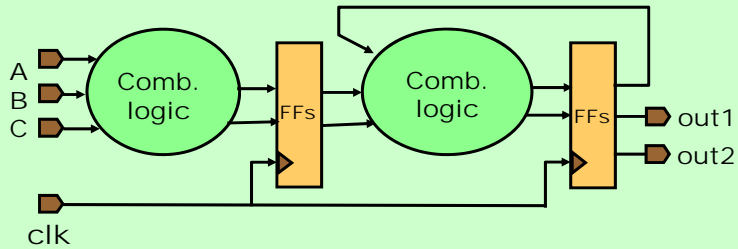
Output: The fault coverage

Application: High-Performance CPU Designs

Major challenge: often too time-consuming

Sequential Design Model

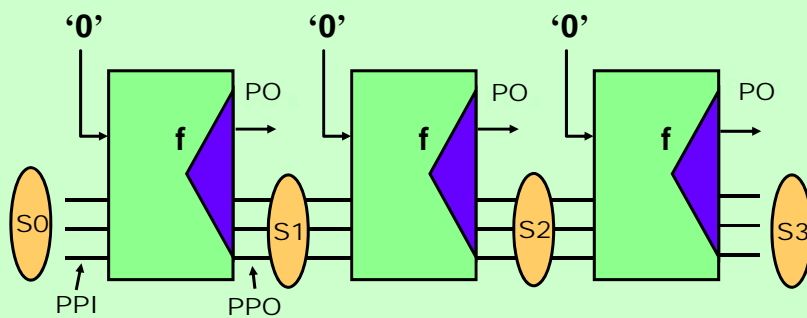
Sequential Circuits



Ch3-47

Time-Frame-Expansion Model

Ex: Input Sequence ('0', '0', '0')
State Sequence ($S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3$)



Notations: PPI: pseudo primary inputs (i.e., outputs of flip-flops)
PPO: pseudo primary outputs (i.e., inputs of flip-flops)

A single fault becomes multiple faults in the time-frame-expansion model

Ch3-48

Hypertrophic Faults

- A hypertrophic fault
 - Is a fault that diverges from the fault-free circuit with a large number of Xs, which usually is a stuck-at fault occurring at a control line and thus prevents the circuit initialization
- A small number of hypertrophic faults
 - account for a large percentage of fault events and CPU time
- These faults are sometimes dropped
 - as potentially detected faults to reduce simulation time. However, the resultant fault coverage then becomes approximate

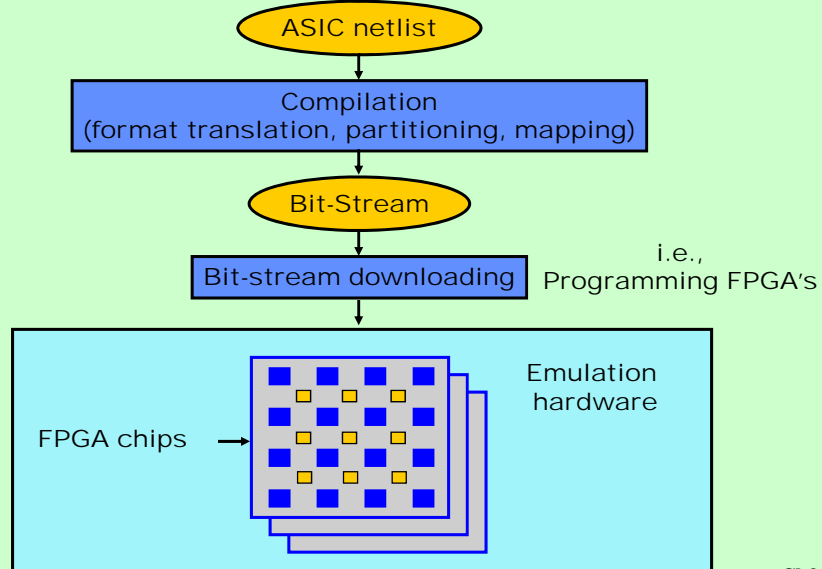
A potentially detected fault is a fault detected only when the circuit is powered on in certain states, not every state.

Ch3-49

Fault Emulation

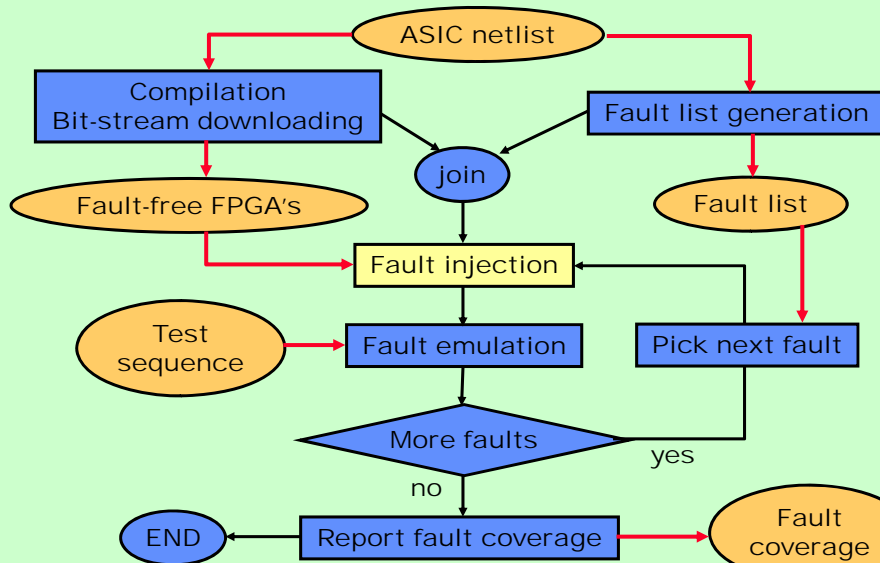
We can utilize FPGA to speed up the sequential fault grading

FPGA-Based Emulation Process



Ch3-51

Serial Fault Emulation by FPGA's



Ch3-52

Fault Injection Should Be Efficient !

- Fault Injection

- Is to convert a **fault-free FPGA** implementation to a **faulty one**
- If not efficient, could become the new **bottleneck**

- (1) Static Fault Injection

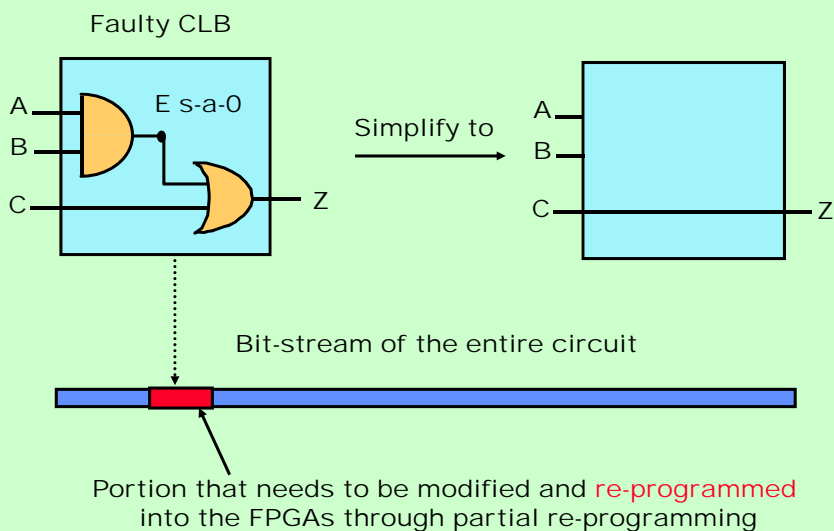
- Directly changes the **configuration** of the fault-free implementation to a faulty one

- (2) Dynamic Fault Injection

- Do not change the configuration directly
- Fault inject is injected through the **control of some hardware** originally built-in to the netlist

Ch3-53

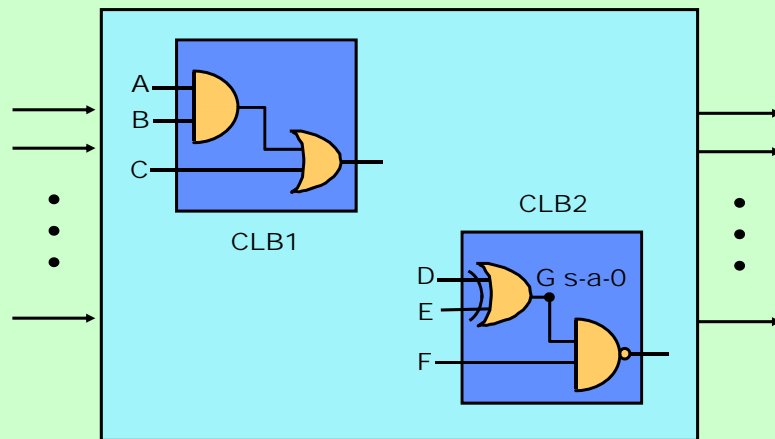
Static Fault Injection



Ch3-54

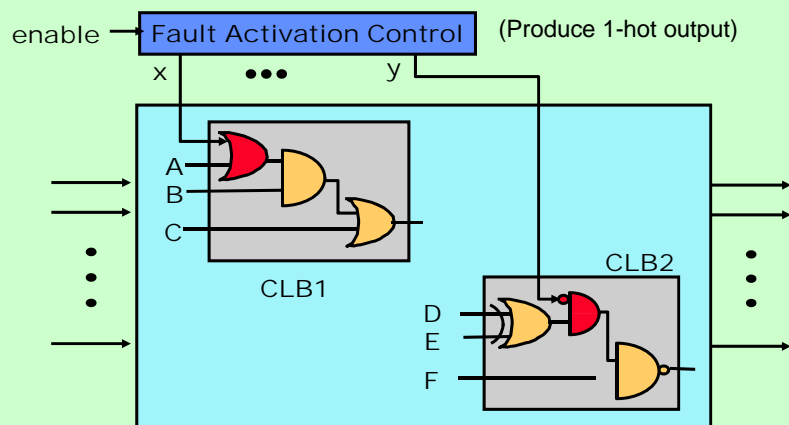
Example: FPGA-implementation

Two faults are being considered:
A stuck-at 1
G stuck-at-0



Ch3-55

Dynamic Fault Injection (I)

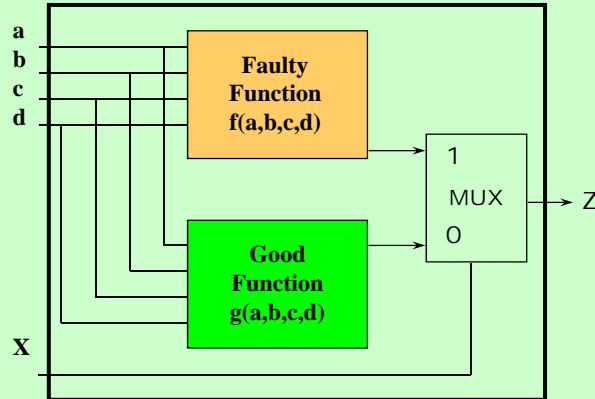


($x=1, y=0$) → The above netlist behaves like A s-a-1 faulty circuit
($x=0, y=1$) → The above netlist behaves like G s-a-0 faulty circuit

Ch3-56

Dynamic Fault Injection (II)

- (1) Conservatively map only 4-input function to a CLB, which is originally assumed to be capable of realizing 5-input function.
- (2) Extra input, i.e., x , is reserved for the control of dynamic fault injection.

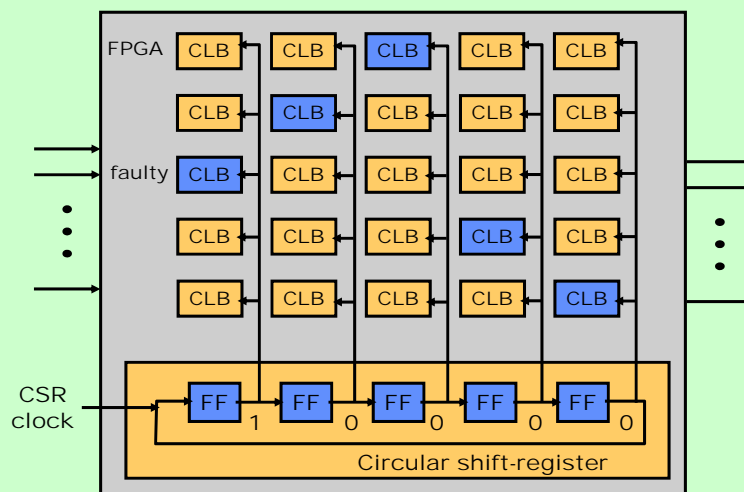


A Configurable Logic Block (CLB)
with a dynamic fault injected (activated with $x=1$)

Ch3-57

Overview of Dynamic Fault Injection (II)

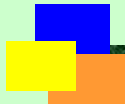
In the following configuration:
5 faults are injected (one for each column), but only 1 is activated



Ch3-58

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 4
Automatic Test Pattern Generation

General ATPG Flow

• **ATPG (Automatic Test Pattern Generation)**

- Generate a set of vectors for a set of target faults

• **Basic flow**

Initialize the vector set to **NULL**

Repeat

Generate **a new test vector**

Evaluate fault coverage for the test vector

If the test vector is acceptable, then add it to the vector set

Until required fault coverage is obtained

• **To accelerate the ATPG**

- **Random patterns** are often generated first to detect easy-to-detect faults, then a deterministic TG is performed to generate tests for the remaining faults

Combinational ATPG

- **Test Generation (TG) Methods**

- Based on **Truth Table**
- Based on **Boolean Equation**
- Based on **Structural Analysis**

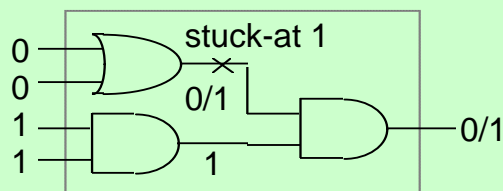
- **Milestone Structural ATPG Algorithms**

- D-algorithm [Roth 1967]
- 9-Valued D-algorithm [Cha 1978]
- PODEM [Goel 1981]
- FAN [Fujiwara 1983]

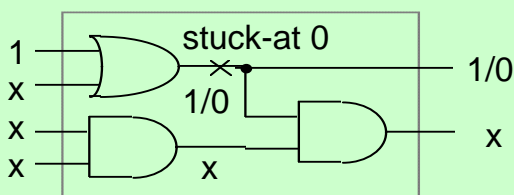
ch4-3

A Test Pattern

A Fully Specified Test Pattern
(every PI is either 0 or 1)



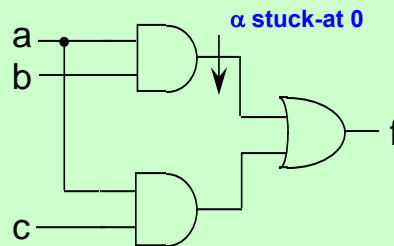
A Partially Specified Test Pattern
(certain PI's could be **undefined**)



ch4-4

Test Generation Methods (From Truth Table)

Ex: How to generate tests
for the stuck-at 0 fault
(fault α)?



abc	f	f_α
000	0	0
001	0	0
010	0	0
011	0	0
100	0	0
101	1	1
110	1	0
111	1	1

ch4-5

Test Generation Methods (Using Boolean Equation)

$$f = ab + ac, f_\alpha = ac$$

T_α = the set of all tests for fault α

$$= \text{ON_set}(f \oplus f_\alpha)$$

$$= \text{ON_set}(f) * \text{OFF_set}(f_\alpha) + \text{OFF_set}(f) * \text{ON_set}(f_\alpha)$$

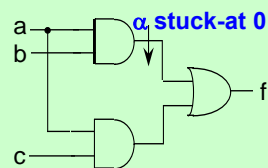
$$= \{(a,b,c) \mid (ab+ac)(ac)' + (ab+ac)'(ac) = 1\} \leftarrow \text{Boolean equation}$$

$$= \{(a,b,c) \mid abc' = 1\}$$

$$= \{(110)\}.$$

High complexity !!

Since it needs to compute the faulty function for each fault.



* **ON_set(f)**: All input combinations to which f evaluates to 1.
OFF_set(f): All input combinations to which f evaluates to 0.
 Note: a function is characterized by its **ON_SET**

ch4-6

Boolean Difference

- **Physical Meaning of Boolean Difference**

- For a logic function $F(X)=F(x_1, \dots, x_i, \dots, x_n)$, find **all the input combinations** that make a value-change at x_i also cause a value-change at F .

- **Logic Operation of Boolean Difference**

- The Boolean difference of $F(X)$ w.r.t. input x_i is

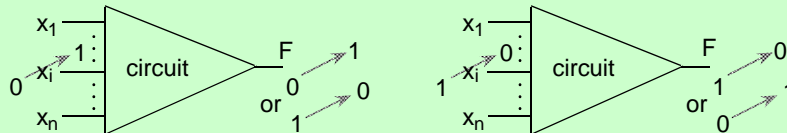
$$dF(x)/dx_i = F_i(0) \oplus F_i(1) = F_i(0) \cdot F_i(1)' + F_i(0)' \cdot F_i(1)$$

Where

$$F_i(0) = F(x_1, \dots, 0, \dots, x_n)$$

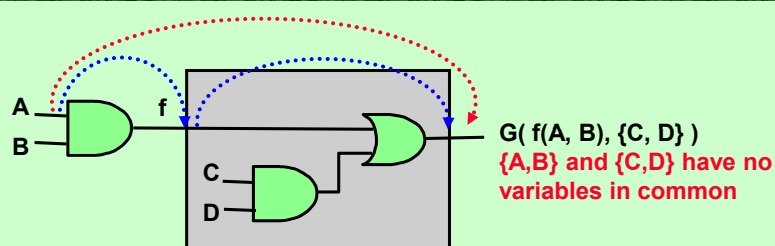
$$F_i(1) = F(x_1, \dots, 1, \dots, x_n)$$

- **Illustrations of Boolean Difference**



ch4-7

Chain Rule



$$\begin{aligned} f &= AB \\ G &= f + CD \end{aligned} \quad \Rightarrow \quad \begin{aligned} dG/df &= (C' + D') \\ df/dA &= B \end{aligned}$$

$$\Rightarrow dG/dA = (dG/df) \cdot (df/dA) = (C' + D') \cdot B$$

An Input vector v sensitizes a fault effect from **A to G**
 iff v sensitizes the effect from **A to f** and from **f to G**

ch4-8

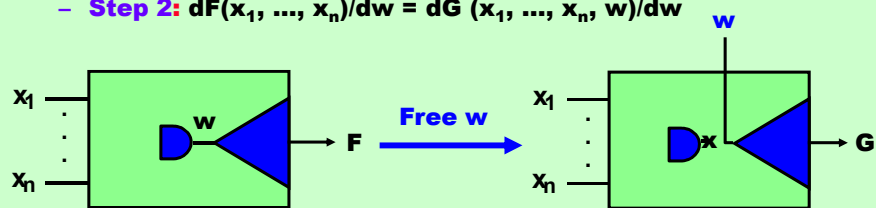
Boolean Difference (con't)

• Boolean Difference

- With respect to an **internal signal**, w , Boolean difference represents the set of **input combinations that sensitize** a fault effect from w to the primary output F

• Calculation

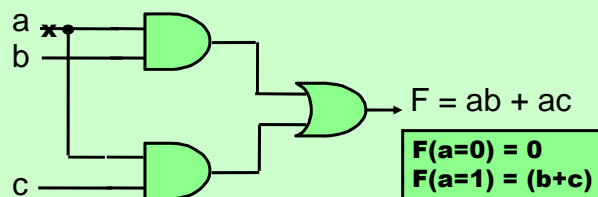
- **Step 1:** convert the function F into a new one G that takes the **signal w as an extra primary input**
- **Step 2:** $dF(x_1, \dots, x_n)/dw = dG(x_1, \dots, x_n, w)/dw$



ch4-9

Test Gen. By Boolean Difference

Case 1: Faults are present at Pls.



➔ **Fault Sensitization Requirement:**
 $dF/da = F(a=0) \oplus F(a=1) = 0 \oplus (b+c) = (b+c)$

Test-set for a s-a-1 = $\{(a,b,c) \mid \underline{a' \cdot (b+c)=1}\} = \{(01x), (0x1)\}$.

Test-set for a s-a-0 = $\{(a,b,c) \mid \underline{a \cdot (b+c)=1}\} = \{(11x), (1x1)\}$.

**No need to compute
The faulty function !!**

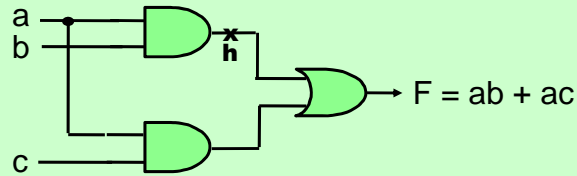
Fault activation
requirement

Fault sensitization
requirement

ch4-10

Test Generation By Boolean Difference (con't)

Case 2: Faults are present at internal lines.



$$G(\text{i.e., } F \text{ with } h \text{ floating}) = h + ac$$

$$dG/dh = G(h=0) \oplus G(h=1) = (ac \oplus 1) = (a' + c')$$

Test-set for h s-a-1 is

$$\{(a,b,c) \mid h' \cdot (a' + c') = 1\} = \{(a,b,c) \mid (a' + b') \cdot (a' + c') = 1\} = \{(0xx), (x00)\}.$$

Test-set for h s-a-0 is

$$\{(a,b,c) \mid \underline{h} \cdot \underline{(a' + c')} = 1\} = \{(110)\}.$$

For fault activation For fault sensitization

ch4-11

Outline

• Test Generation (TG) Methods

- Based on Truth Table
- Based on Boolean Equation
- Based on Structural Analysis
- ➡ - D-algorithm [Roth 1967]
- 9-Valued D-algorithm [Cha 1978]
- PODEM [Goel 1981]
- FAN [Fujiwara 1983]

ch4-12

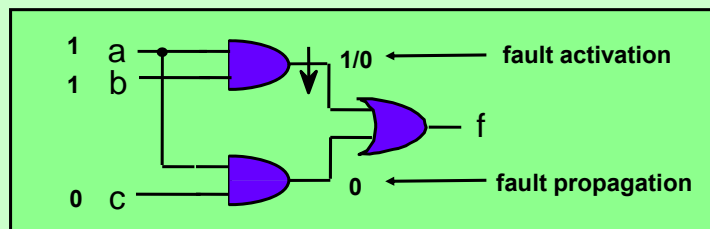
Test Generation Method (From Circuit Structure)

- **Two basic goals**

- (1) **Fault activation (FA)**
- (2) **Fault propagation (FP)**
- Both of which requires **Line Justification (LJ)**, i.e., finding input combinations that force certain signals to their desired values

- **Notations:**

- **1/0 is denoted as D**, meaning that good-value is 1 while faulty value is 0
- Similarly, **0/1 is denoted as D'**
- Both D and D' are called **fault effects (FE)**



ch4-13

Common Concepts for Structural TG

- **Fault activation**


- Setting the faulty signal to either 0 or 1 is a **Line Justification** problem

- **Fault propagation**

- (1) select a path to a PO → **decisions**
- (2) Once the path is selected → a set of line justification (LJ) problems are to be solved

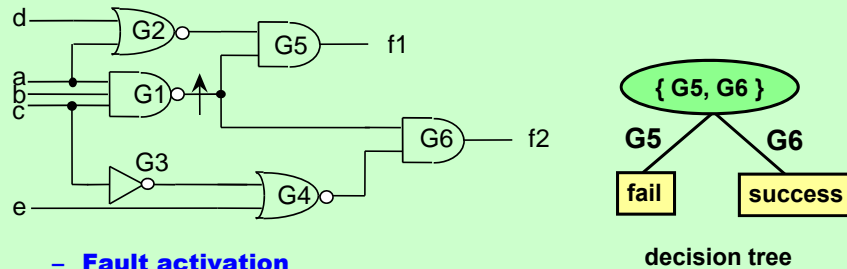
- **Line Justification**

- Involves **decisions** or **implications**
- Incorrect decisions: need **backtracking**

To justify $c=1 \rightarrow a=1$ and $b=1$ (implication) a —  — c
 To justify $c=0 \rightarrow a=0$ **or** $b=0$ (decision) b —

ch4-14

Ex: Decision on Fault Propagation



- Fault activation

- $G1=0 \rightarrow \{a=1, b=1, c=1\} \rightarrow \{G3=0\}$

- Fault propagation: through G5 or G6

- Decision through G5:

- $G2=1 \rightarrow \{d=0, a=0\} \rightarrow$ inconsistency at a \rightarrow backtrack !!

- Decision through G6:

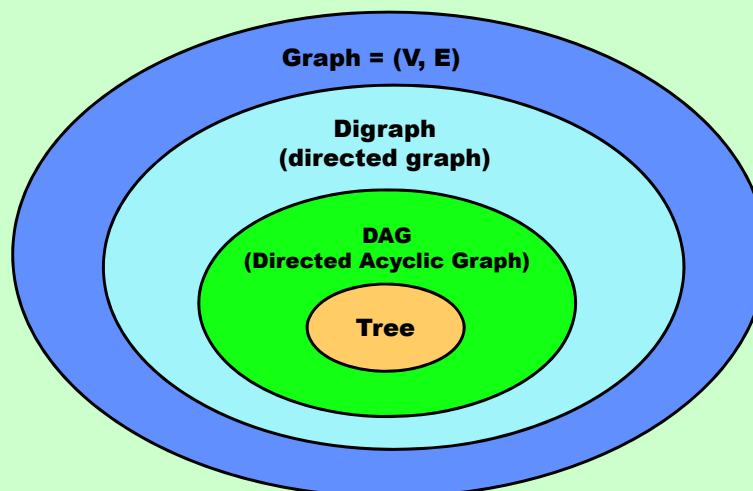
- $\rightarrow G4=1 \rightarrow e=0 \rightarrow$ done !! The resulting test is (111x0)

D-frontiers: are the gates whose output value is x, while one or more inputs are D or D'. For example, initially, the D-frontier is { G5, G6 }.

ch4-15

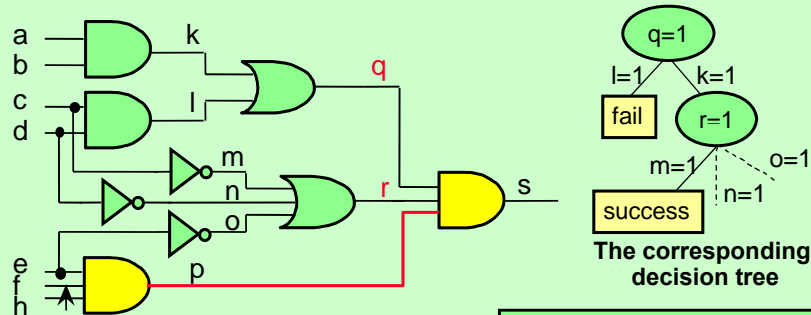
Various Graphs

A **Combinational Circuit**: is usually modeled as a **DAG**, but not **tree**



ch4-16

Ex: Decisions On Line Justification



- FA → set h to 0
- FP → e=1, f=1 (→o=0) ; FP → q=1, r=1
- To justify q=1 → l=1 or k=1 Decision point
- Decision: l=1 → c=1, d=1 → m=0, n=0 → r=0 → inconsistency at r → backtrack !
- Decision: k=1 → a=1, b=1
- To justify r=1 → m=1 or n=1 (→c=0 or d=0) → Done ! (J-frontier is ϕ)

ch4-17

Branch-and-Bound Search

• Test Generation

- Is a branch-and-bound search
- Every decision point is a **branching** point
- If a set of decisions lead to a **conflict** (or **bound**), a **backtrack** is taken to explore other decisions
- A test is found when
 - (1) fault effect is propagated to a PO
 - (2) all internal lines are justified
- No test is found after all possible decisions are tried
→ Then, target fault is **undetectable**
- Since the search is **exhaustive**, it will find a test if one exists

For a **combinational** circuit, an **undetectable** fault is also a **redundant** fault → Can be used to simplify circuit.

ch4-18

Implications

• Implications

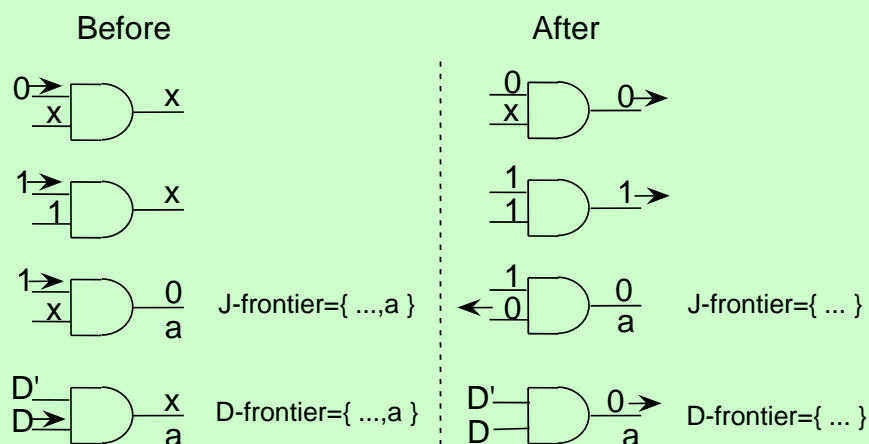
- Computation of the values that can be uniquely determined
 - **Local implication:** propagation of values from one line to its immediate successors or predecessors
 - **Global implication:** the propagation involving a larger area of the circuit and re-convergent fanout

• Maximum Implication Principle

- Perform as many implications as possible
- It helps to either reduce the number of problems that need decisions or to **reach an inconsistency sooner**

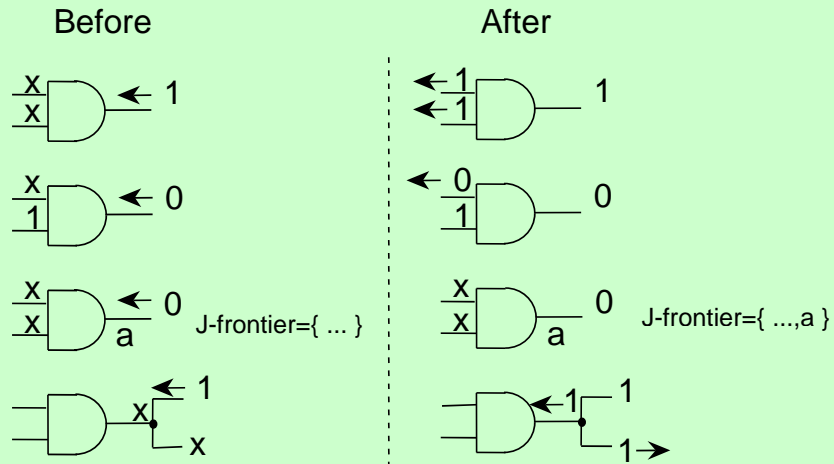
ch4-19

Local Implications (Forward)



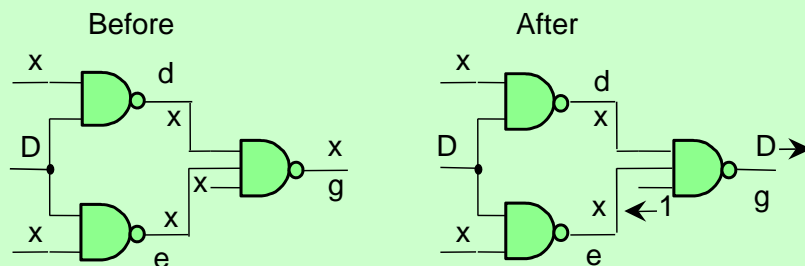
ch4-20

Local Implications (Backward)



ch4-21

Global Implications



• Unique D-Drive Implication

- Suppose D-frontier (or D-drive) is $\{d, e\}$, $\rightarrow g$ is a **dominator** for both d and e , hence a **unique D-drive** is at g

g is called a dominator of d :
because every path from d to an **PO** passes through g

ch4-22

Learning for Global Implication

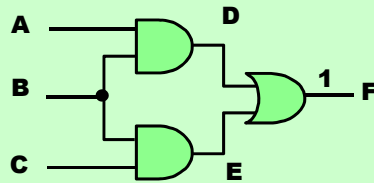
• Static Learning

$$A \rightarrow B \Rightarrow \sim B \rightarrow \sim A$$

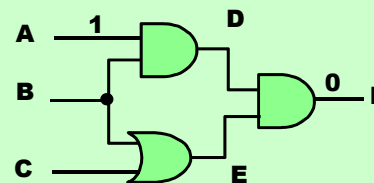
- Global implication derived by **contraposition law**
- Learn static (i.e., **input independent**) signal implications

• Dynamic Learning

- **Contraposition law** + other **signal values**
- Is input pattern dependent



F=1 implies B=1
Because $B=0 \rightarrow F=0$
(Static Learning)

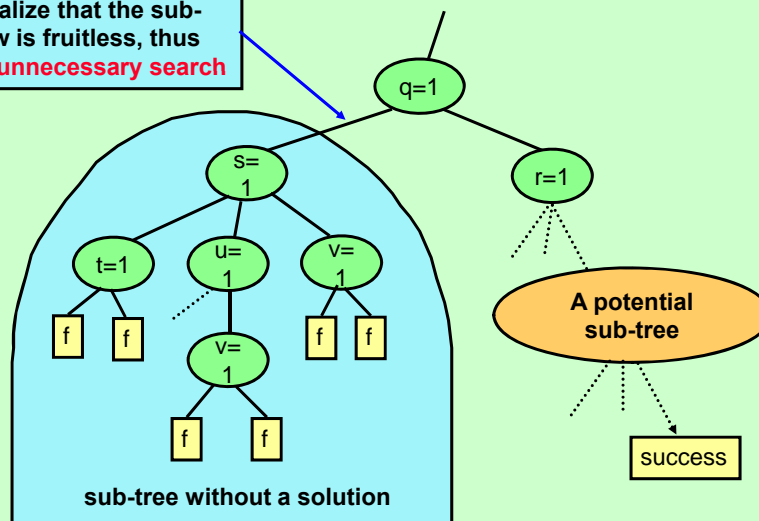


F=0 implies B=0 When A=1
Because $\{B=1, A=1\} \rightarrow F=1$
(Dynamic Learning)

ch4-23

Early Detection of Inconsistency

Aggressive implication may help to realize that the sub-tree below is fruitless, thus avoiding **unnecessary search**

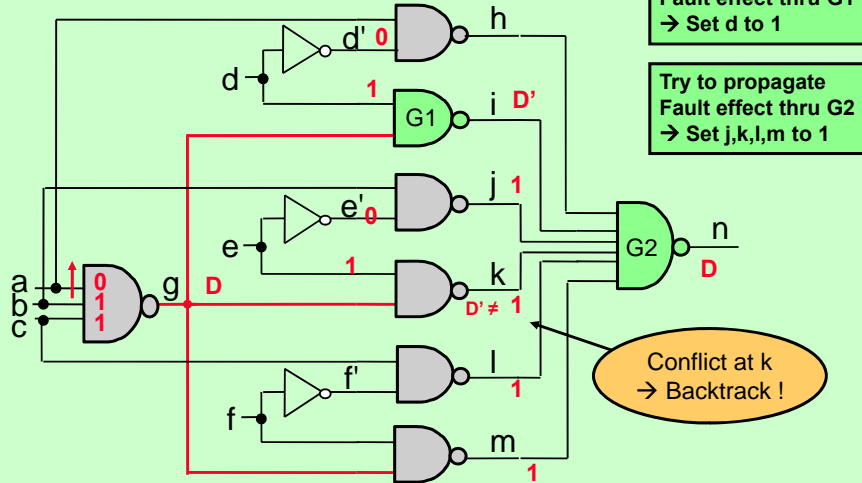


ch4-24

Ex: D-Algorithm (1/3)

• Five logic values

– { 0, 1, x, D, D' }

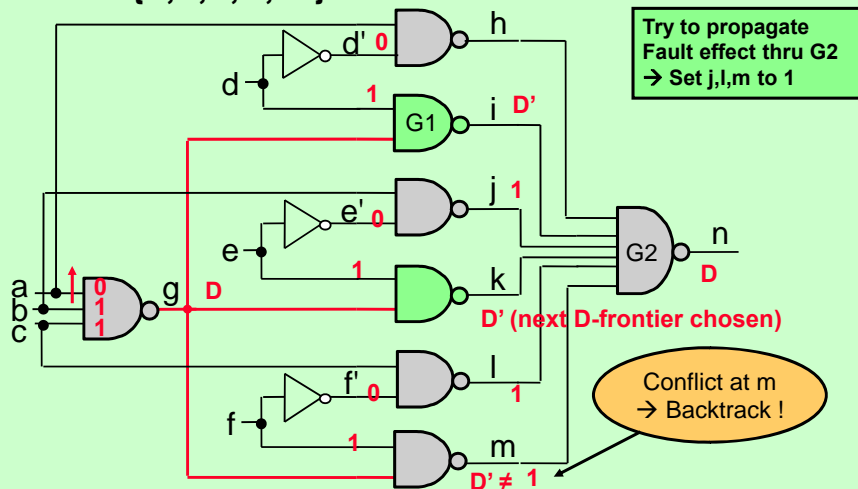


ch4-25

Ex: D-Algorithm (2/3)

• Five logic values

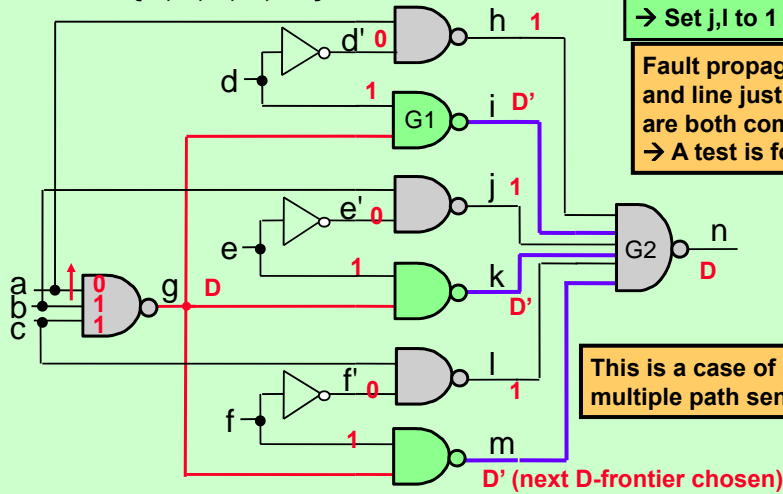
– { 0, 1, x, D, D' }



ch4-26

Ex: D-Algorithm (3/3)

- Five logic values**
 - { 0, 1, x, D, D' }



Try to propagate
Fault effect thru G2
→ Set j,l to 1

Fault propagation
and line justification
are both complete
→ A test is found !

This is a case of
multiple path sensitization !

ch4-27

D-Algorithm: Value Computation

Decision	Implication	Comments
	a=0 h=1 b=1 c=1 g=D	Active the fault Unique D-drive
d=1	i=D' d'=0	Propagate via i
j=1 k=1 l=1 m=1	n=D e'=0 e=1 k=D'	Propagate via n Contradiction

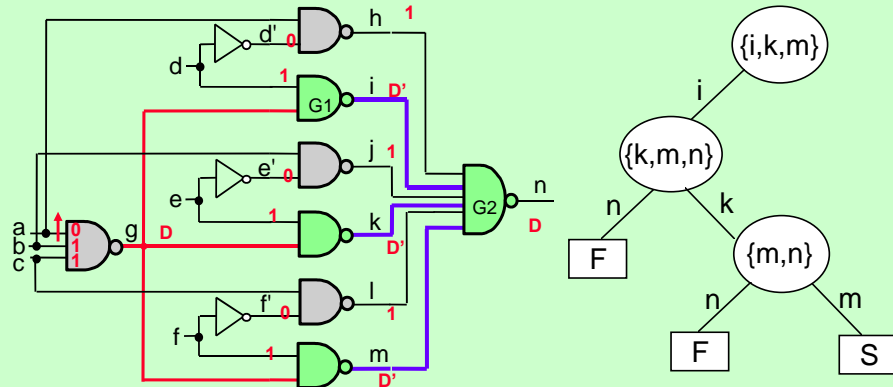
e=1	k=D' e'=0 j=1	Propagate via k
l=1 m=1	n=D f'=0 f=1 m=D'	Propagate via n Contradiction
f=1	m=D' f'=0 l=1 n=D	Propagate via m

ch4-28

Decision Tree on D-Frontier

- The decision tree below

- Node → D-frontier
- Branch → Decision Taken
- A **Depth-First-Search (DFS)** strategy is often used



ch4-29

9-Value D-Algorithm

- Logic values (fault-free / faulty)

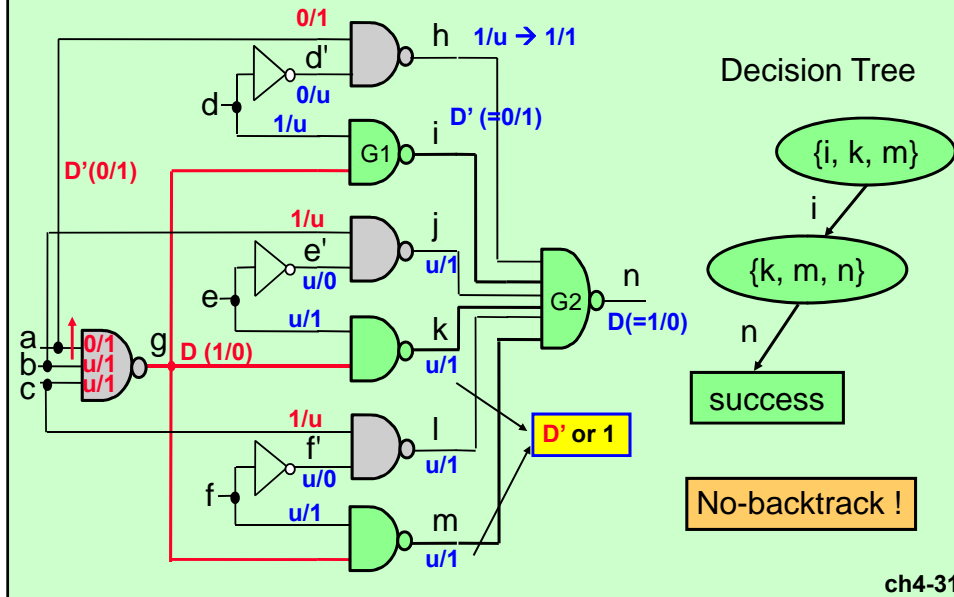
- $\{0/0, 0/1, 0/u, 1/0, 1/1, 1/u, u/0, u/1, u/u\}$,
- where $0/u=\{0,D'\}$, $1/u=\{D,1\}$, $u/0=\{0,D\}$, $u/1=\{D',1\}$, $u/u=\{0,1,D,D'\}$.

- Advantage:

- Automatically considers **multiple-path sensitization**, thus reducing the amount of search in D-algorithm
- The speed-up is **NOT** very significant in practice because **most faults are detected through single-path sensitization**

ch4-30

Example: 9-Value D-Algorithm



Final Step of 9-Value D-Algorithm

- **To derive the test vector**
 - **A = (0/1) → 0 (take the fault-free one)**
 - **B = (1/u) → 1**
 - **C = (1/u) → 1**
 - **D = (u/1) → 1**
 - **E = (u/1) → 1**
 - **F = (u/1) → 1**
- **The final vector**
 - **(A,B,C,D,E,F) = (0, 1, 1, 1, 1, 1)**

ch4-32

Outline

• Test Generation (TG) Methods

- Based on Truth Table
- Based on Boolean Equation
- Based on Structural Analysis
- D-algorithm [Roth 1967]
- 9-Valued D-algorithm [Cha 1978]
- ➔ – **PODEM** [Goel 1981]
- FAN [Fujiwara 1983]

ch4-33

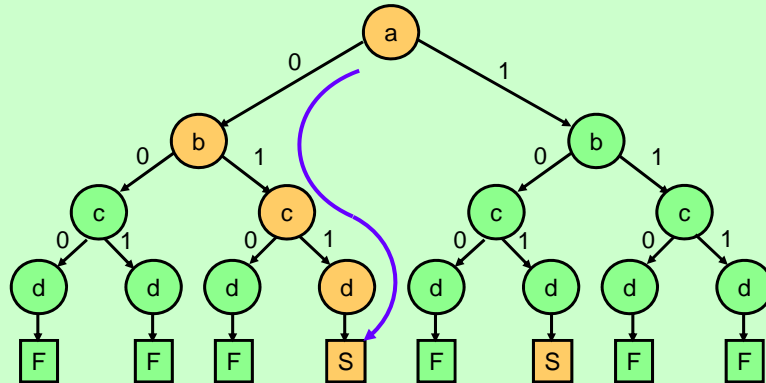
PODEM: Path-Oriented DEcision Making

- **Fault Activation (FA) and Propagation (FP)**
 - lead to sets of Line Justification (LJ) problems. The LJ problems can be solved via value assignments.
- **In D-algorithm**
 - TG is done through **indirect signal assignment** for FA, FP, and LJ, that eventually maps into **assignments at PI's**
 - The **decision points** are at **internal lines**
 - The worst-case number of **backtracks is exponential** in terms of the number of decision points (e.g., at least 2^k for k decision nodes)
- **In PODEM**
 - The test generation is done through a sequence of **direct assignments at PI's**
 - Decision points are at PIs, thus the number of **backtracking** might be **fewer**

ch4-34

Search Space of PODEM

- **Complete Search Space**
 - A binary tree with 2^n leaf nodes, where n is the number of PI's
- **Fast Test Generation**
 - Need to find a path leading to a **SUCCESS** terminal **quickly**



ch4-35

Objective() and Backtrace()

- **PODEM**
 - Also aims at establishing a sensitization path based on **fault activation and propagation** like D-algorithm
 - Instead of **justifying** the signal values required for sensitizing the selected path, **objectives** are setup to guide the **decision process at PI's**
- **Objective**
 - is a **signal-value pair** (w, v_w)
- **Backtrace**
 - **Backtrace** maps a desired objective into a **PI assignment** that is likely to contribute to the achievement of the objective
 - Is a process that traverses the circuit back from the objective signal to PI's
 - The result is a **PI signal-value pair** (x, v_x)
 - **No signal value is actually assigned during backtrace !**

往輸入端追蹤

ch4-36

Objective Routine

- **Objective Routine Involves**

- The selection of a **D-frontier**, **G**
- The selection of an **unspecified input gate** of **G**

```
Objective() {  
  /* The target fault is w s-a-v */  
  /* Let variable obj be a signal-value pair */  
  if (the value of w is x)    obj = ( w, v' ); ← fault activation  
  else {  
    select a gate (G) from the D-frontier; ← fault propagation  
    select an input (j) of G with value x;  
    c = controlling value of G;  
    obj = (j, c');  
  }  
  return (obj);  
}
```

ch4-37

後追蹤 Backtrace Routine

- **Backtrace Routine**

- Involves finding an **all-x path from objective site to a PI**, i.e., every signal in this path has value **x**

```
Backtrace(w, vw) {  
  /* Maps objective into a PI assignment */  
  G = w; /* objective node */  
  v = vw; /* objective value */  
  while (G is a gate output) { /* not reached PI yet */  
    inv = inversion of G;  
    select an input (j) of G with value x;  
    G = j; /* new objective node */  
    v = v ⊕ inv; /* new objective value */  
  }  
  /* G is a PI */ return (G, v);  
}
```

ch4-38

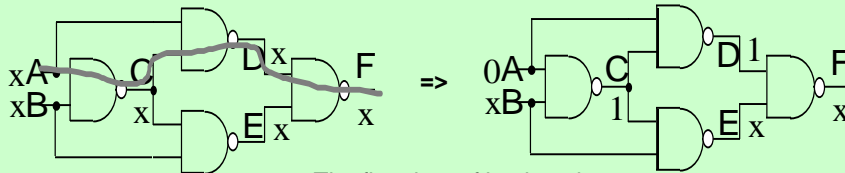
Example: Backtrace

Objective to achieve: (F, 1)

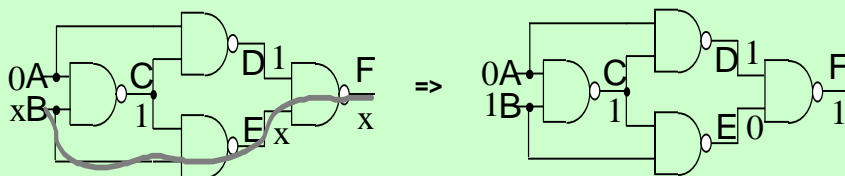
PI assignments:

(1) A = 0 → fail

(2) B = 1 → succeed



The first time of backtracing



The second time of backtracing

ch4-39

PI Assignment in PODEM

Assume that: PI's: { a, b, c, d }

Current Assignments: { a=0 }

Decision: b=0 → objective fails

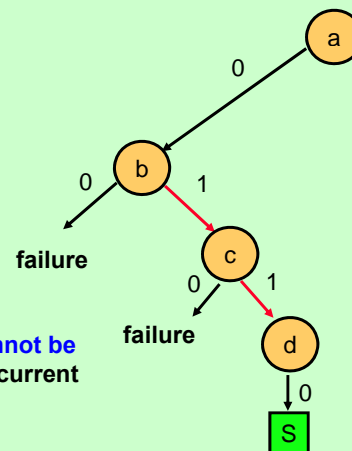
Reverse decision: b=1

Decision: c=0 → objective fails

Reverse decision: c=1

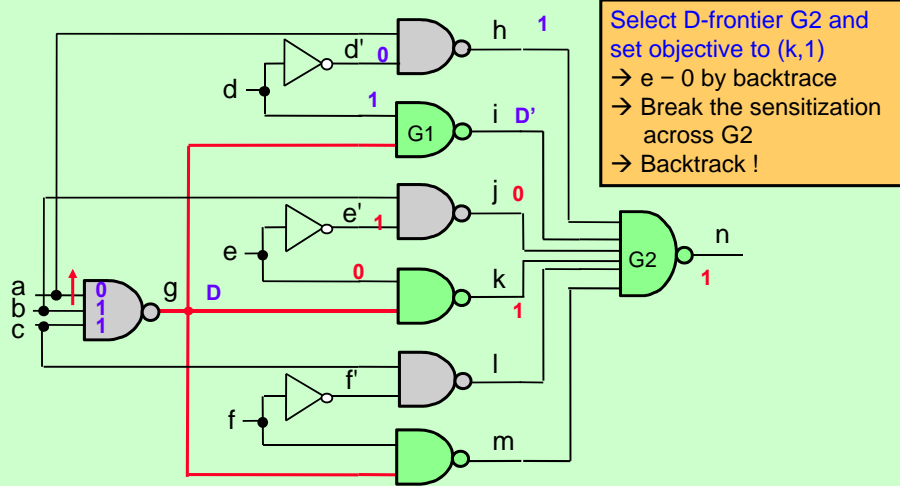
Decision: d=0

Failure means fault effect cannot be propagated to any PO under current PI assignments



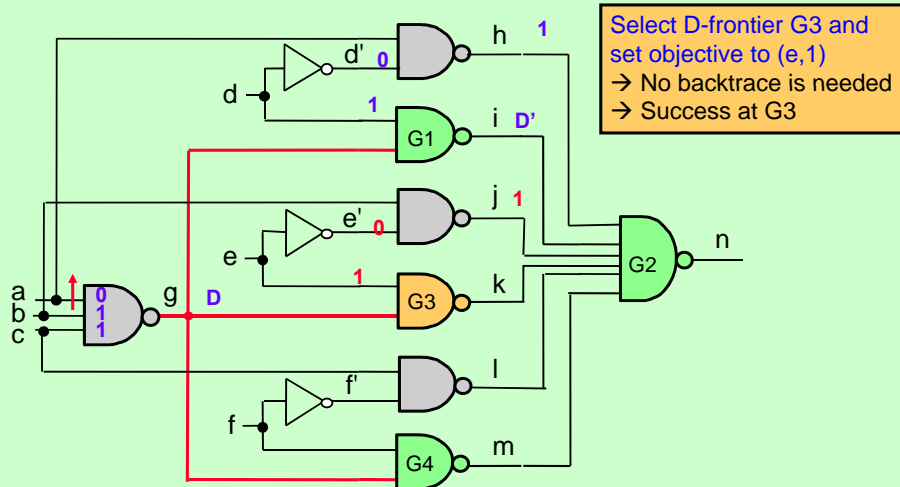
ch4-40

Example: PODEM (1/3)



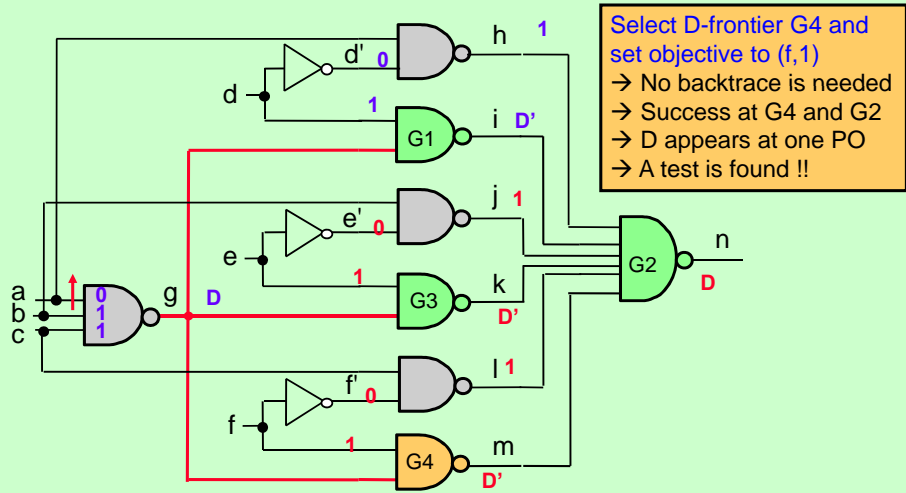
ch4-41

Example: PODEM (2/3)



ch4-42

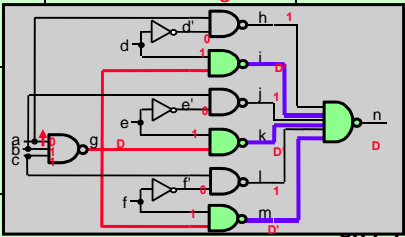
Example: PODEM (3/3)



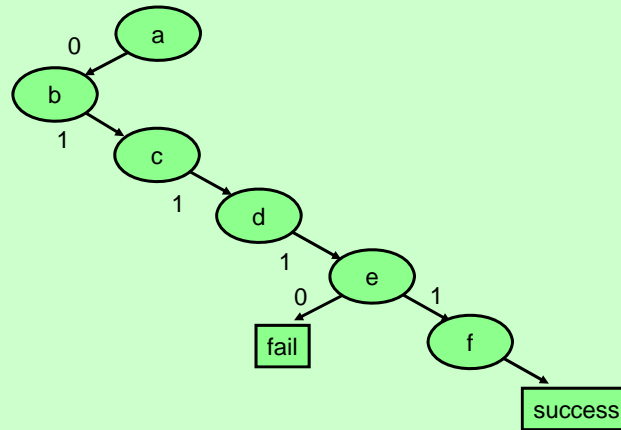
ch4-43

PODEM: Value Computation

Objective	PI assignment	Implications	D-frontier	Comments
a=0	a=0	h=1	g	
b=1	b=1		g	
c=1	c=1	g=D	i,k,m	
d=1	d=1	d'=0		
		i=D'	k,m,n	
k=1	e=0	e'=1 j=0 k=1 n=1	m	Assignments need to be reversed during backtracking
	e=1	e'=0 j=1 k=D'	m,n	no solutions ! → backtrack reverse PI assignment
l=1	f=1	f'=0 l=1 m=D' n=D		



Decision Tree in PODEM



- **Decision node:** the PI selected through backtrace for value assignment
- **Branch:** the value assignment to the selected PI

ch4-45

Terminating Conditions

• D-algorithm

- **Success:**
 - (1) Fault effect at an output (D-frontier may not be empty)
 - (2) J-frontier is empty
- **Failure:**
 - (1) D-frontier is empty (all possible paths are false)
 - (2) J-frontier is **not** empty

• PODEM

- **Success:**
 - Fault effect seen at an output
- **Failure:**
 - Every **PI assignment** leads to failure, in which **D-frontier is empty** while fault has been activated

ch4-46

PODEM: Recursive Algorithm

PODEM () /* using depth-first-search */

begin

If(error at PO) **return**(SUCCESS);

If(test not possible) **return**(FAILURE);

(k, v_k) = **Objective**(); /* choose a line to be justified */

(j, v_j) = **Backtrace**(k, v_k); /* choose the PI to be assigned */

Implied (j, v_j); /* make a decision */

If (**PODEM**()==SUCCESS) **return** (SUCCESS);

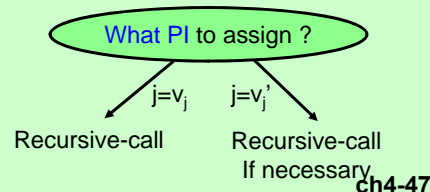
Implied (j, v_j'); /* reverse decision */

If (**PODEM**()==SUCCESS) **return**(SUCCESS);

Implied (j, x);

Return (FAILURE);

end



ch4-47

Overview of PODEM

• PODEM

- examines all possible input patterns **implicitly** but **exhaustively** (branch-and-bound) for finding a test
- It is **complete** like D-algorithm (I.e., will find one if a test exists)

• Other Key Features

- **No J-frontier**, since there are no values that require justification
- **No consistency check**, as conflicts can never occur
- **No backward implication**, because values are propagated only forward
- **Backtracking** is implicitly done by simulation rather than by an explicit and **time-consuming save/restore process**
- Experimental results show that PODEM is **generally faster** than the D-algorithm

ch4-48

The Selection Strategy in PODEM

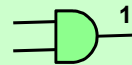
• In Objective() and Backtrace()

- Selections are done **arbitrarily** in original PODEM
- The algorithm will be more efficient if certain guidance used in the selections of **objective node** and **backtrace path**

• Selection Principle

- **Principle 1:** Among **several unsolved problems**

- → Attack the **hardest** one
- Ex: to justify a '1' at an AND-gate output



- **Principle 2:** Among **several solutions** for solving a problem

- → Try the **easiest** one
- Ex: to justify a '1' at OR-gate output



ch4-49

Controllability As Guidance

• Controllability of a signal w

- **CY1(w):** the **probability** that line w has value 1.
- **CY0(w):** the **probability** that line w has value 0.

- **Example:**

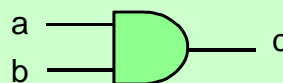
- $f = ab$
- Assume $CY1(a)=CY0(a)=CY1(b)=CY0(b)=0.5$
- $CY1(f)=CY1(a) \times CY1(b)=0.25$,
- $CY0(f)=CY0(a)+CY0(b)-CY0(a) \times CY0(b)=0.75$

• Example of Smart Backtracing

- Objective (c, 1) → choose path $c \rightarrow a$ for backtracing
- Objective (c, 0) → choose path $c \rightarrow b$ for backtracing

CY1(a) = 0.33
CY0(a) = 0.67

CY1(b) = 0.5
CY0(b) = 0.5



ch4-50

Testability Analysis

- **Applications**

- To give an early warning about the testing problems that lie ahead
- To provide **guidance in ATPG**

- **Complexity**

- Should be simpler than ATPG and fault simulation, i.e., need to be **linear** or almost linear in terms of circuit size

- **Topology analysis**

- Only the **structure** of the circuit is analyzed
- No test vectors are involved
- Only approximate, **reconvergent fanouts cause inaccuracy**

ch4-51

SCOAP

(Sandia Controllability/Observability Analysis Program)

- **Computes six numbers for each node N**

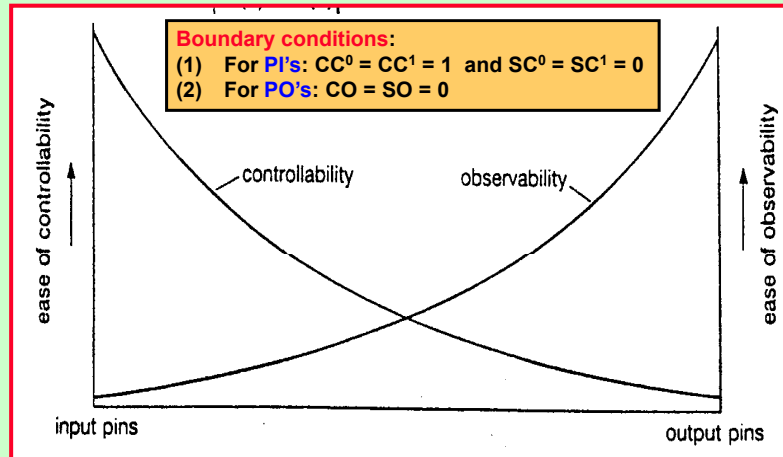
- **CC⁰(N) and CC¹(N)**
 - Combinational 0 and 1 controllability of a node N
- **SC⁰(N) and SC¹(N)**
 - Sequential 0 and 1 controllability of a node N
- **CO(N)**
 - Combinational observability
- **SO(N)**
 - Sequential observability

值越大→代表越困難

ch4-52

General Characteristic of Controllability and Observability

Controllability calculation: sweeping the circuit from PI to PO
Observability calculation: sweeping the circuit from PO to PI



ch4-53

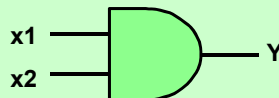
Controllability Measures

– $CC^0(N)$ and $CC^1(N)$

- The **number of combinational nodes** that must be assigned values to justify a 0 or 1 at node N

– $SC^0(N)$ and $SC^1(N)$

- The **number of sequential nodes** that must be assigned values to justify a 0 or 1 at node N



$$\begin{aligned} CC^0(Y) &= \min [CC^0(x1), CC^0(x2)] + 1 \\ CC^1(Y) &= CC^1(x1) + CC^1(x2) + 1 \\ SC^0(Y) &= \min [SC^0(x1), SC^0(x2)] \\ SC^1(Y) &= SC^1(x1) + SC^1(x2) \end{aligned}$$

ch4-54

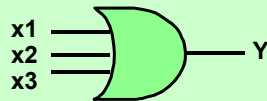
Controllability Measure (con't)

– $CC^0(N)$ and $CC^1(N)$

- The **number of combinational nodes** that must be assigned values to justify a 0 or 1 at node N

– $SC^0(N)$ and $SC^1(N)$

- The **number of sequential nodes** that must be assigned values to justify a 0 or 1 at node N



$$\begin{aligned}
 CC^0(Y) &= CC^0(x1) + CC^0(x2) + CC^0(x3) + 1 \\
 CC^1(Y) &= \min [CC^1(x1), CC^1(x2), CC^1(x3)] + 1 \\
 SC^0(Y) &= SC^0(x1) + SC^0(x2) + SC^0(x3) \\
 SC^1(Y) &= \min [SC^1(x1), SC^1(x2), SC^1(x3)]
 \end{aligned}$$

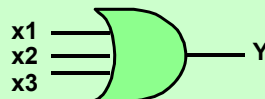
ch4-55

Observability Measure

– $CO(N)$ and $SO(N)$

- The **observability of a node N** is a function of the **output observability** and of the cost of holding all **other inputs at non-controlling values**

Example: X1 observable: (Y observable) + (side-inputs 配合)



$$\begin{aligned}
 CO(x1) &= CO(Y) + CC^0(x2) + CC^0(x3) + 1 \\
 SO(x1) &= SO(Y) + SC^0(x2) + SC^0(x3)
 \end{aligned}$$

ch4-56

PODEM: Example 2 (1/3)

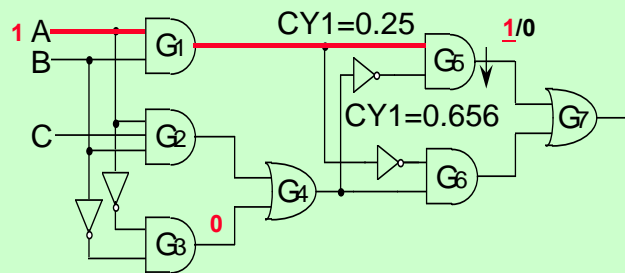
Initial objective=(G5,1).

G5 is an AND gate → Choose the hardest-1

→ Current objective=(G1,1).

G1 is an AND gate → Choose the hardest-1

→ Arbitrarily, Current objective=(A,1). A is a PI → Implication → G3=0.



ch4-57

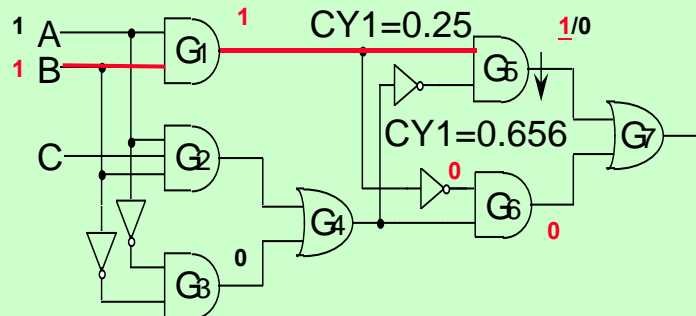
PODEM: Example 2 (2/3)

The initial objective satisfied? No! → Current objective=(G5,1).

G5 is an AND gate → Choose the hardest-1 → Current objective=(G1,1).

G1 is an AND gate → Choose the hardest-1

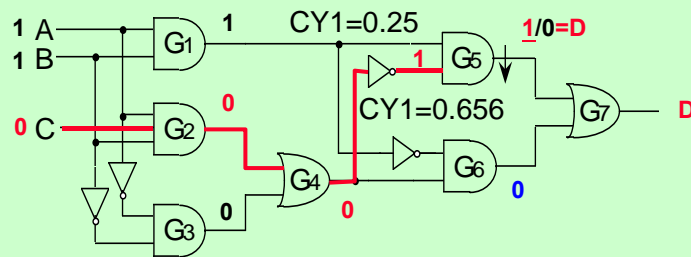
→ Arbitrarily, Current objective=(B,1). B is a PI → Implication → G1=1, G6=0.



ch4-58

PODEM: Example 2 (3/3)

The initial objective satisfied? No! → **Current objective=(G5,1).**
 The value of G1 is known → **Current objective=(G4,0).**
 The value of G3 is known → **Current objective=(G2,0).**
 A, B is known → **Current objective=(C,0).**
 C is a PI → Implication → G2=0, G4=0, G5=D, G7=D.

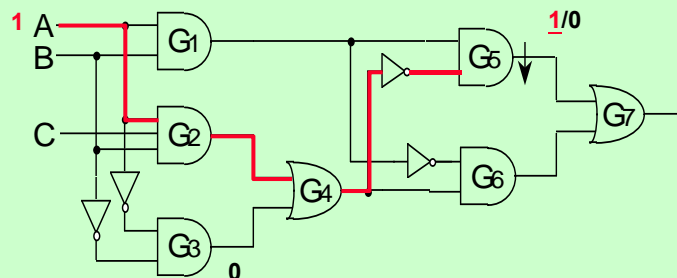


No backtracking !!

ch4-59

If The Backtracing Is Not Guided (1/3)

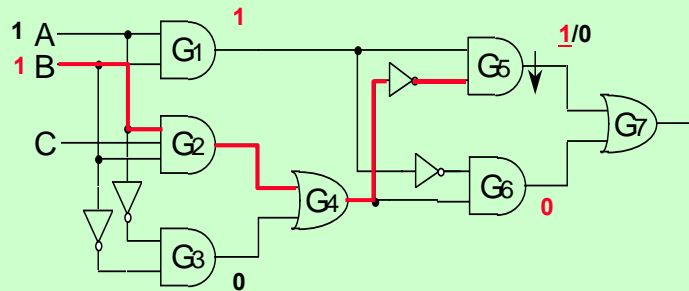
Initial objective=(G5,1).
 Choose path G5-G4-G2-A → **A=0.**
 Implication for A=0 → G1=0, G5=0 → **Backtracking to A=1.**
 Implication for A=1 → G3=0.



ch4-60

If The Backtracing Is Not Guided (2/3)

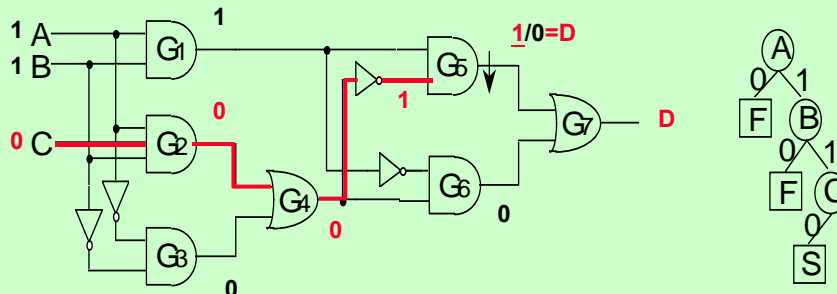
The initial objective satisfied? No! → **Current objective=(G5,1).**
 Choose path G5-G4-G2-B → **B=0.**
 Implication for B=0 → G1=0, G5=0 → **Backtracking to B=1.**
 Implication for B=1 → G1=1, G6=0.



ch4-61

If The Backtracing Is Not Guided (3/3)

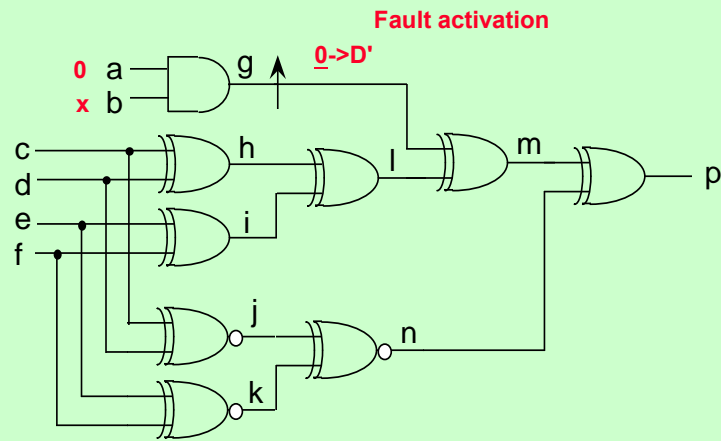
The initial objective satisfied? No! → **Current objective=(G5,1).**
 Choose path G5-G4-G2-C → **C=0.**
 Implication for C=0 → G2=0, G4=0, G5=D, G7=D.



Two times of backtracking !!

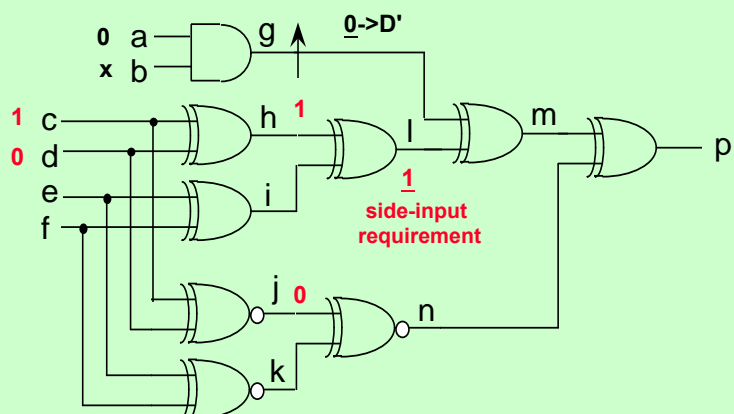
ch4-62

ECAT Circuit: PODEM (1/3)



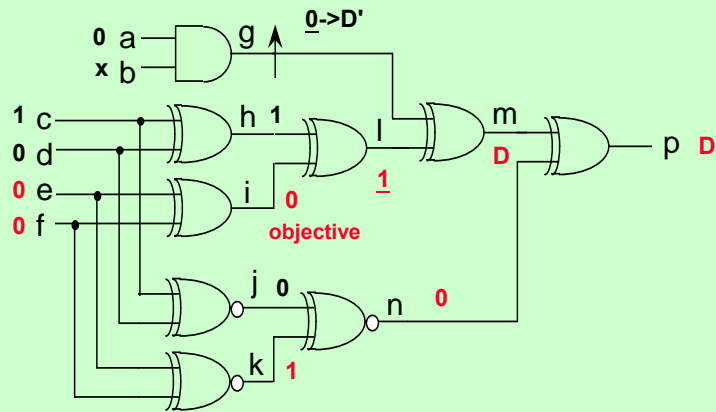
ch4-63

ECAT Circuit: PODEM (2/3)



ch4-64

ECAT Circuit: PODEM (3/3)



No backtracking !!

ch4-65

Outline

• Test Generation (TG) Methods

- Based on Truth Table
- Based on Boolean Equation
- Based on Structural Analysis
- D-algorithm [Roth 1967]
- 9-Valued D-algorithm [Cha 1978]
- PODEM [Goel 1981]
- ➡ - FAN [Fujiwara 1983]

ch4-66

FAN (Fanout Oriented) Algorithm

- **FAN**

- Introduces two major extensions to PODEM's backtracing algorithm

- **1st extension**

- Rather than stopping at PI's, backtracing in FAN may **stop at an internal lines**

- **2nd extension**

- FAN uses **multiple backtrace** procedure, which attempts to satisfy a set of objectives simultaneously

ch4-67

Headlines and Bound Lines

- **Bound line**

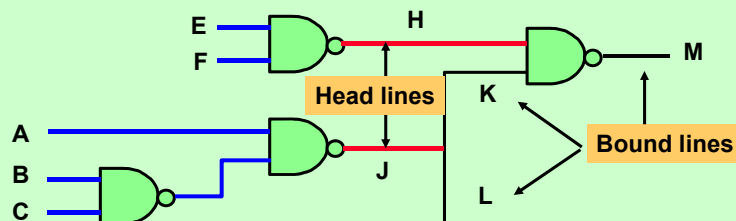
- A line reachable from at least one **stem**

- **Free line**

- A line that is NOT bound line

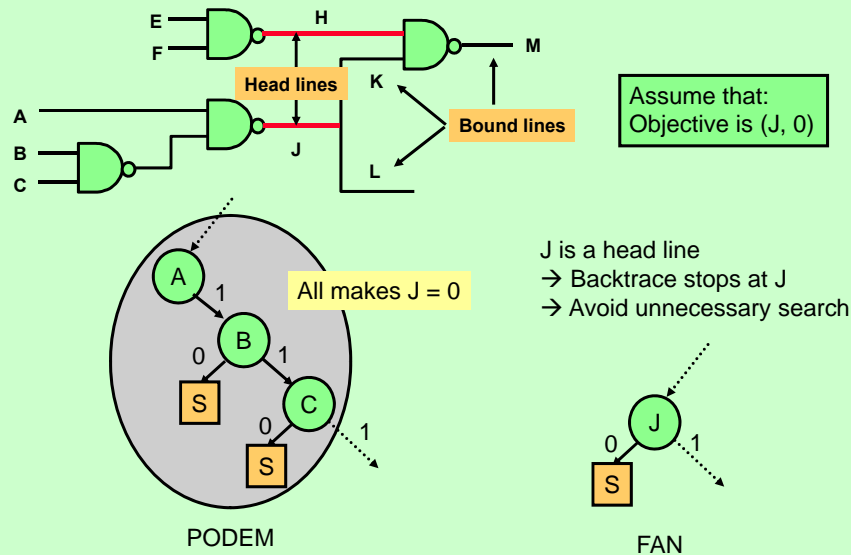
- **Head line**

- A free line that directly feeds a bound line



ch4-68

Decision Tree (PODEM v.s. FAN)



ch4-69

Why Stops at Head Lines ?

- **Head lines are mutually independent**
 - Hence, for each given **value combination at head lines**, there always exists an **input combination** to realize it.
- **FAN has two-steps**
 - **Step 1: PODEM using headlines as pseudo-PI's**
 - **Step 2: Generate real input pattern to realize the value combination at head lines.**

ch4-70

Why Multiple Backtrace ?

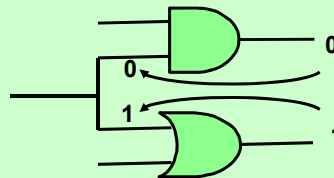
• Drawback of Single Backtrace

- A PI assignment satisfying one objective → may preclude achieving another one, and this leads to backtracking

• Multiple Backtrace

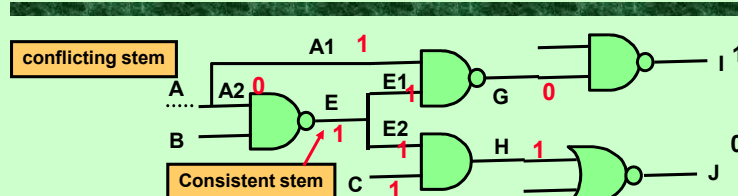
- Starts from a **set of objectives** (Current_objectives)
- Maps these multiple objectives into a **head-line assignment** $k=v_k$ that is likely to
 - Contribute to the **achievement of a subset** of the objectives
 - Or show that some subset of the original objectives **cannot be simultaneously achieved**

Multiple objectives
May have conflicting
Requirements at a stem



ch4-71

Example: Multiple Backtrace



Current_objectives	Processed entry	Stem_objectives	Head_objectives
(I,1), (J,0)	(I,1)		
(J,0), (G,0)	(J,0)		
(G,0), (H,1)	(G,0)		
(H,1), (A1,1), (E1,1)	(H,1)		
(A1,1), (E1,1), (E2,1), (C,1)	(A1,1)	A	
(E1,1), (E2,1), (C,1)	(E1,1)	A,E	
(E2,1), (C,1)	(E2,1)	A,E	
(C,1)	(C,1)	A,E	C
Empty → restart from (E,1)		A	C
(E,1)	(E,1)	A	C
(A2,0)	(A2,0)	A	C
empty		A	C

ch4-72

Multiple Backtrace Algorithm

```

Mbacktrace (Current_objectives) {
  while (Current_objectives  $\neq \emptyset$ ) {
    remove one entry (k, vk) from Current_objectives;
    switch (type of entry) {
      1. HEAD_LINE: add (k, vk) to Head_objectives;
      2. FANOUT_BRANCH:
          j = stem(k);
          increment no. of requests at j for vk; /* count 0s and 1s */
          add j to Stem_objectives;
      3. OTHERS:
          inv = inversion of k; c = controlling value of k;
          select an input (j) of k with value x;
          if ((vk  $\oplus$  inv) == c) add(j, c) to Current_objectives;
          else { for every input (j) of k with value x
                  add(j, c') to Current_objectives; }
    }
  }
  } TO BE CONTINUED ...

```

ch4-73

Multiple Backtrace (con't)

```

Mbacktrace (Current_objectives) {
  while (Current_objectives  $\neq \emptyset$ ) {body in previous page}
  if(Stem_objectives $\neq \emptyset$ ) {
    remove the highest-level stem (k) from Stem_Objectives;
    vk = most requested value of k;
    /* recursive call here */
    add (k, vk) to Current_objectives;
    return (Mbacktrace(Current_objectives);
  }
  else { remove one objective (k, vk) from Head_objectives;
        return (k, vk)
  }
}

```

ch4-74

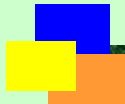
References

- [1] Sellers et al., "Analyzing errors with the Boolean difference", IEEE Trans. Computers, pp. 676-683, 1968.
- [2] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", IBM Journal of Research and Development, pp. 278-291, July, 1966.
- [2'] J. P. Roth et al., "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits", IEEE Trans. Electronic Computers, pp. 567-579, Oct. 1967.
- [3] C. W. Cha et al., "9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits", IEEE TC, pp. 193-200, March, 1978.
- [4] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. Computers, pp. 215-222, March, 1981.
- [5] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms", IEEE TC, pp. 1137-1144, Dec. 1983.
- [6] M. H. Schulz et al., "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System", IEEE Trans. on CAD, pp. 126-137, 1988.
- [6'] M. H. Schulz and E. Auth, "Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification", IEEE Trans CAD, pp. 811-816, 1989.

ch4-75

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 5
Design For Testability
& Scan Test

Outline

- 
- Introduction
 - Why DFT?
 - What is DFT?
 - Ad-Hoc Approaches
 - Full Scan
 - Partial Scan

Why DFT ?

- Direct Testing is Way Too Difficult !
 - Large number of FFs
 - Embedded memory blocks
 - Embedded analog blocks
- Design For Testability is inevitable
 - Like death and tax

ch5-3

Design For Testability

- Definition
 - Design For Testability (DFT) refers to those **design techniques** that make test generation and testing cost-effective
- DFT Methods
 - **Ad-hoc methods**
 - **Scan**, full and partial
 - **Built-In Self-Test** (BIST)
 - **Boundary scan**
- Cost of DFT
 - **Pin count, area, performance, design-time, test-time**

ch5-4

Why DFT Isn't Universally Used Previously?

- Short-sighted **view** of management
- **Time-to-market** pressure
- Life-cycle **cost ignored** by development management/contractors/buyers
- **Area/functionality/performance** myths
- Lack of **knowledge** by design engineers
- Testing is **someone else's problem**
- **Lack of tools** to support DFT until recently

We don't have to worry about this management barrier any more
→ **Most design teams now have DfT people**

ch5-5

Important Factors

- **Controllability**
 - Measure the ease of controlling a line
- **Observability**
 - Measure the ease of observing a line at PO
- **Predictability**
 - Measure the ease of predicting output values
- **DFT deals with ways of improving**
 - Controllability
 - Observability
 - Predictability

ch5-6

Outline

- Introduction
- ➡ • Ad-Hoc Approaches
 - Test Points
 - Design Rules
- Full Scan
- Partial Scan

ch5-7

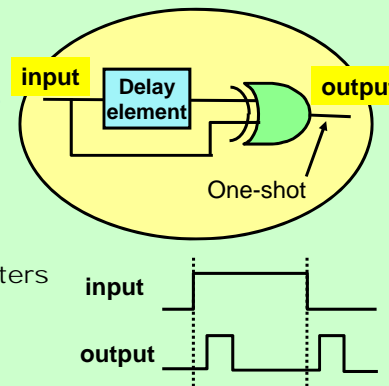
Ad-Hoc Design For Testability

- Design Guidelines
 - Avoid redundancy
 - Avoid asynchronous logic
 - Avoid clock gating (e.g., ripple counter)
 - Avoid large fan-in
 - Consider tester requirements (tri-stating, etc.)
- Disadvantages
 - High fault coverage not guaranteed
 - Manual test generation
 - Design iterations required

ch5-8

Some Ad-Hoc DFT Techniques

- Test Points
- Initialization
- Monostable multivibrators
 - One-shot circuit
- Oscillators and clocks
- Counters / Shift-Registers
 - Add control points to long counters
- Partition large circuits
- Logical redundancy
- Break global feedback paths



ch5-9

On-Line Self-Test & Fault Tolerance By Redundancy

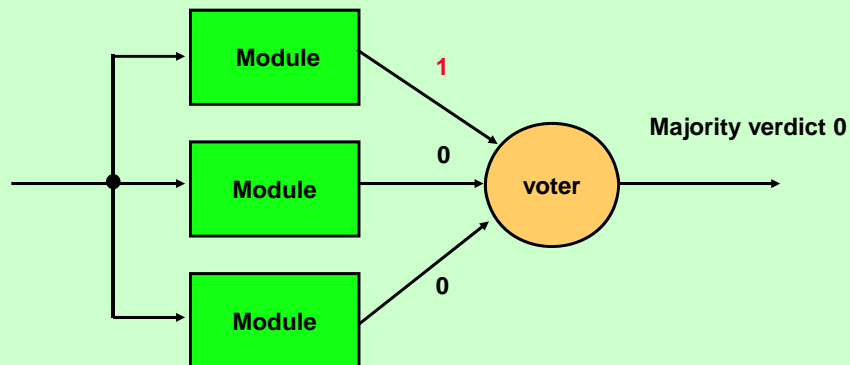
- Information Redundancy
 - Outputs = (information-bits) + (check-bits)
 - Information bits are the original normal outputs
 - Check bits always maintains a specific pre-defined logical or mathematical relationship with the corresponding information bits
 - Any time, if the information-bits and check-bits violate the pre-defined relationship, then it indicates an error
- Hardware Redundancy
 - Use extra hardware (e.g., duplicate or triplicate the system) so that the fault within one module will be masked (I.e., the faulty effect never observed at the final output)

ch5-10

Module Level Redundancy

- Triple Module Redundancy (TMR)

- majority voting on three identical modules' outputs help mask out faults that occur in a single module



ch5-11

Test Point Insertion

- Employ test points to enhance

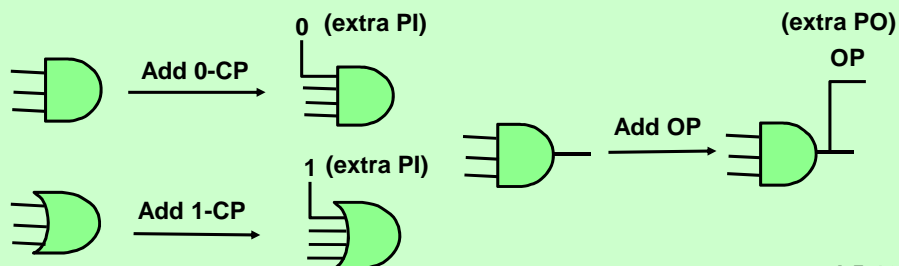
- Controllability
- Observability

- CP: Control Points

- Primary inputs used to enhance controllability

- OP: Observability Points

- Primary outputs used to enhance observability



ch5-12

0/1 Injection Circuitry

- Normal operation

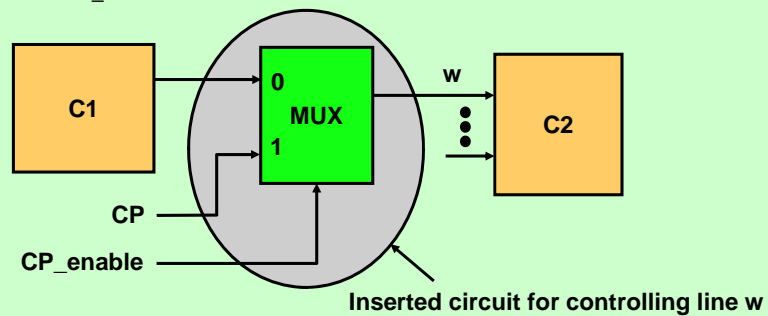
When CP_enable = 0

- Inject 0

– Set CP_enable = 1 and CP = 0

- Inject 1

– Set CP_enable = 1 and CP = 1

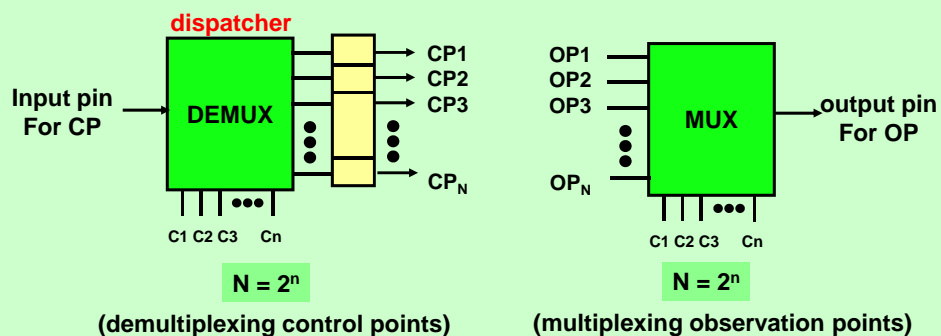


ch5-13

Single I/O Port for Multiple Test Points

- Constraints of using test points

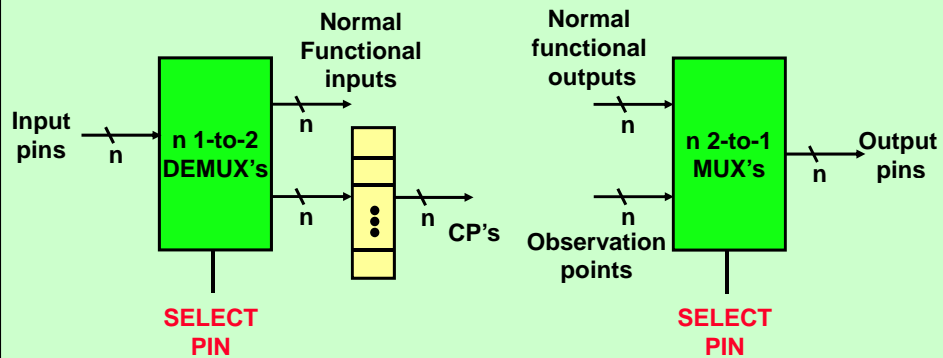
- A large demand on I/O pins
- This constraint can be somewhat relieved by using MUX & DEMUX at the cost of increasing the test time



ch5-14

Sharing Between Test Points & Normal I/O

- Advantage: Even fewer I/O pins for Test Points
- Overhead: Extra MUX delay for normal I/O



ch5-15

Control Point Selection

- Impact
 - The **controllability** of the fanout-cone of the added point is improved
- Common selections
 - Control, address, and data buses
 - Enable / Hold inputs
 - Enable and read/write inputs to memory
 - **Clock** and **set/clear** signals of flip-flops
 - Data select inputs to multiplexers and demultiplexers

ch5-16

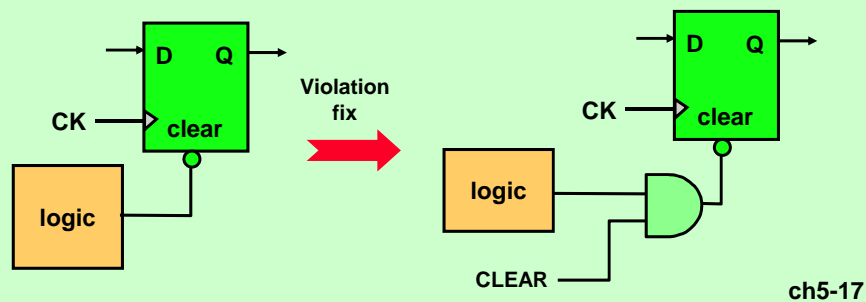
Example: Use CP to Fix DFT Rule Violation

- DFT rule violations

- The **set/clear** signal of a flip-flop is generated by other logic, instead of directly controlled by an input pin
- **Gated clock** signals

- Violation Fix

- Add a control point to the **set/clear signal** or clock signals

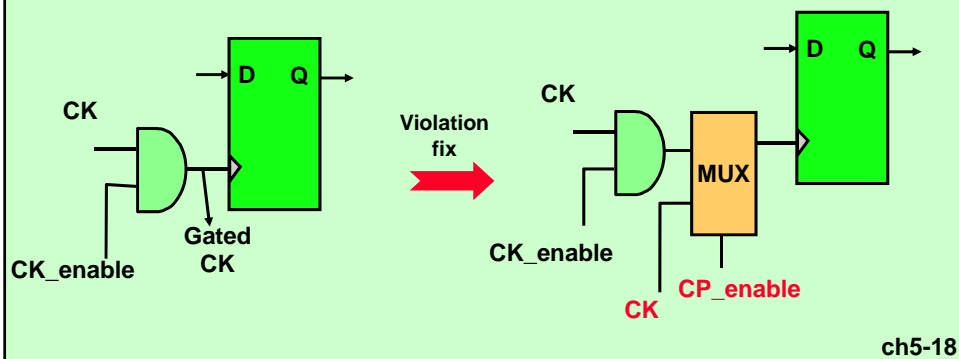


Example: Fixing Gated Clock

- Gated Clocks

- **Advantage**: power dissipation of a logic design can thus be reduced
- **Drawback**: the design's testability is also reduced

- Testability Fix



Example: Fixing Tri-State Bus Contention

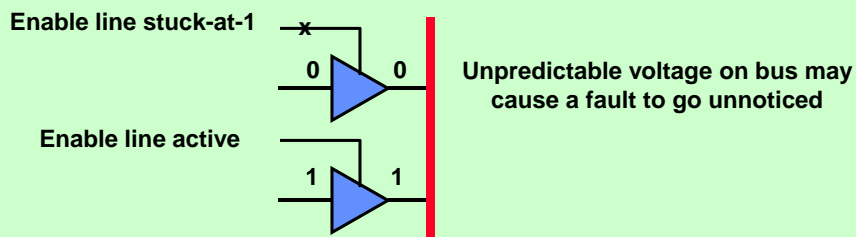
- **Bus Contention**

- A stuck-at-fault at the **tri-state enable line** may cause bus contention – **multiple active drivers** are connected to the bus simultaneously

- **Fix**

- Add CPs to **turn off tri-state devices** during testing

(A Bus Contention Scenario in the presence of a fault)

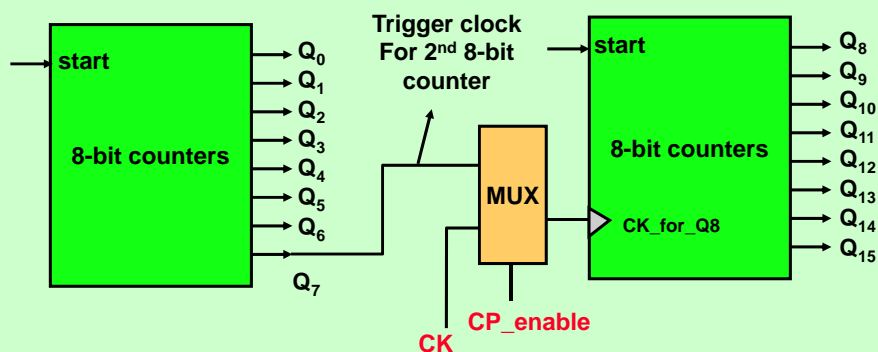


ch5-19

Example: Partitioning Counters

- **Consider a 16-bit ripple-counter**

- Could take up to $2^{16} = 65536$ cycles to test
- After being partitioned into **two 8-bit counters** below, it can be tested with just $2 \times 2^8 = 512$ cycles



ch5-20

Observation Point Selection

- Impact

- The **observability** of the fanin-cone (or transitive fanins) of the added OP is improved

- Common choice

- Stem lines having a **large number of fanouts**
- Global feedback paths
- Redundant signal lines
- Output of logic devices having many inputs
 - MUX, XOR trees
- **Output from state devices**
- Address, control and data buses
(常為電路區塊間之介面訊號)

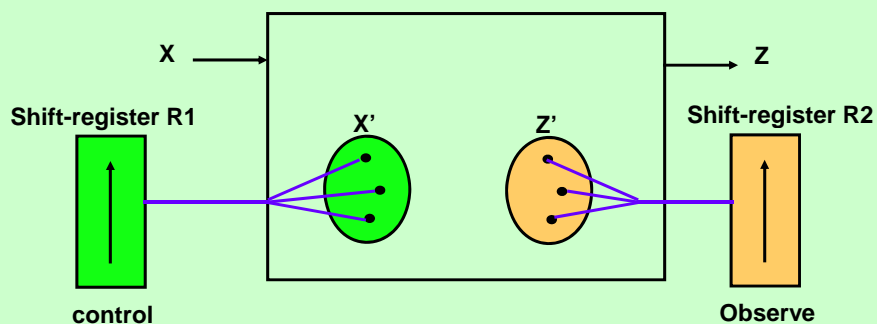
ch5-21

Problems of CP & OP

- Large number of I/O pins

- Add MUXes to reduce the number of I/O pins
- Serially shift CP values by shift-registers

- Larger test time



ch5-22

Outline

- Introduction
- Ad-Hoc Approaches
- ➡ • Full Scan
 - The Concept
 - Scan Cell Design
 - Random Access Scan
- Partial Scan

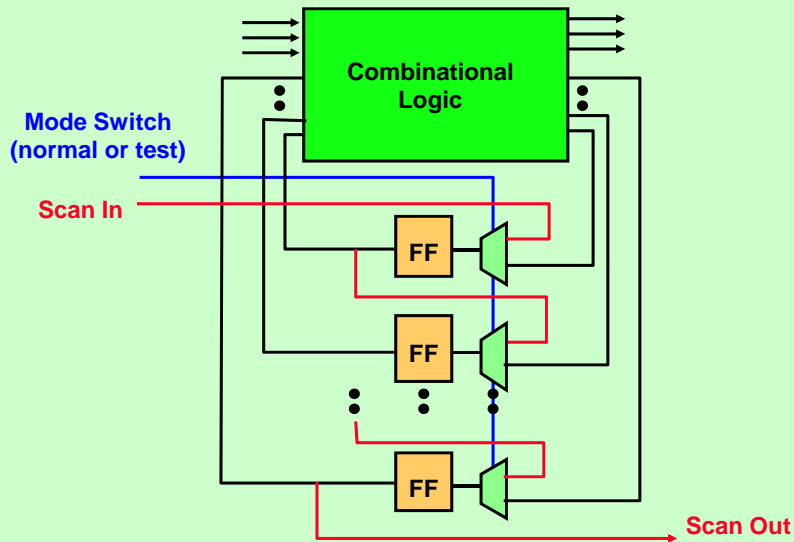
ch5-23

What Is Scan ?

- Objective
 - To provide controllability and observability at **internal state variables** for testing
- Method
 - Add **test mode** control signal(s) to circuit
 - Connect **flip-flops to form shift registers** in test mode
 - Make inputs/outputs of the flip-flops in the shift register controllable and observable
- Types
 - Internal scan
 - **Full scan, Partial scan, Random access**
 - Boundary scan

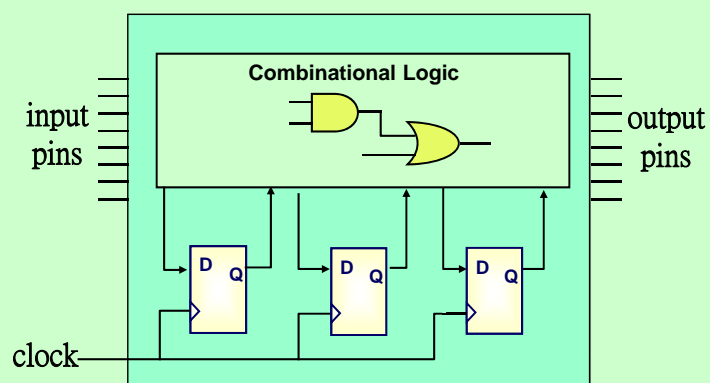
ch5-24

The Scan Concept



ch5-25

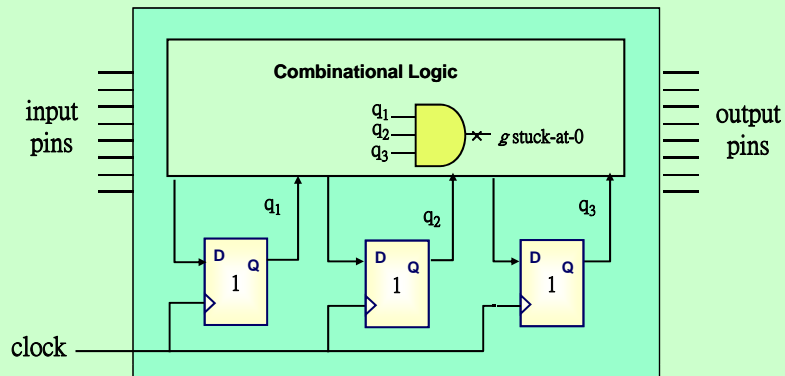
A Logic Design Before Scan Insertion



Sequential ATPG is extremely difficult:
due to the lack of controllability and observability at flip-flops.

ch5-26

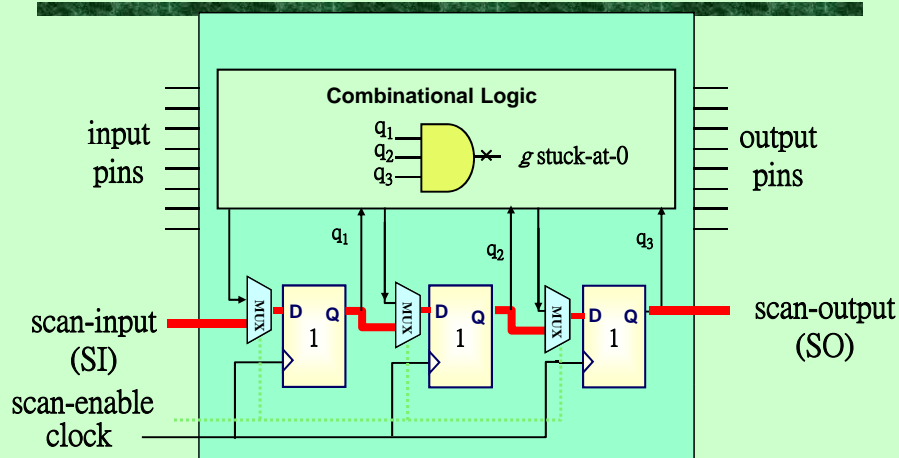
Example: A 3-stage Counter



It takes 8 clock cycles to set the flip-flops to be (1, 1, 1),
for detecting the g stuck-at-0 fault
(2^{20} clock cycles for a 20-stage counter !)

ch5-27

A Logic Design After Scan Insertion



Scan Chain provides an **easy access** to flip-flops
→ Pattern Generation is much easier !!

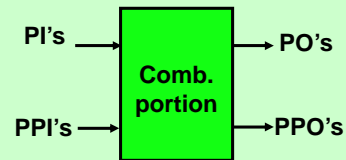
Note: Scan Enable (SE), not shown here, controls every MUX.

ch5-28

Procedure of Applying Test Patterns

• Notation

- Test vectors $\mathbf{T} = \langle t_i^I, t_i^F \rangle i=1, 2, \dots$
- Output Response $\mathbf{R} = \langle r_i^O, r_i^F \rangle i=1, 2, \dots$



• Test Application

- (1) $i = 1$;
- (2) Scan-in t_1^F /* scan-in the **first state vector** for PPI's */
- (3) Apply t_i^I /* apply **current input vector** at PI's */
- (4) Observe r_i^O /* observe **current output response** at PO's */
- (5) **Capture PPOs to FFs as r_i^F** /* capture the response at PPO's to FFs */
 - (Set to '**Normal Mode**' by raising SE to '1' for one clock cycle)
- (6) Scan-out r_i^F while scanning-in t_{i+1}^F /* **overlap scan-in and scan-out** */
- (7) $i = i+1$; Goto step (3)

ch5-29

Testing Scan Chain ?

• Common practice

- Scan chain is often **first tested** before testing the core logic by a so-called **flush test** - which pumps random vectors in and out of the scan chain

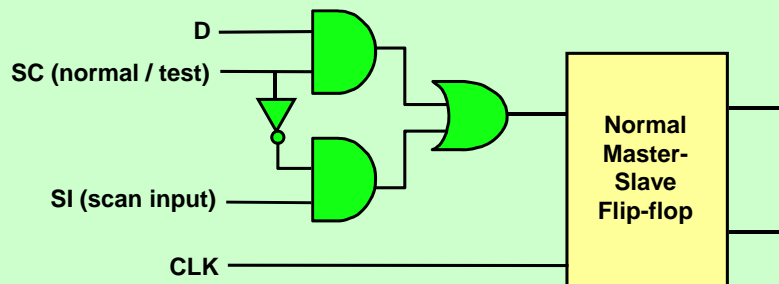
• Procedure (flush test of scan chain)

- (1) $i = 0$;
- (2) Scan-in 1st random vector to flip-flops
- (3) Scan-out (i)th random vector while scanning-in (i+1)th vector for flip-flops.
 - The (i)th **scan-out vector** should be identical to (i)th **vector scanned in** earlier, otherwise scan-chain is **mal-functioning**
- (4) If necessary $i = i+1$, goto step (3)

ch5-30

MUX-Scan Flip-Flop

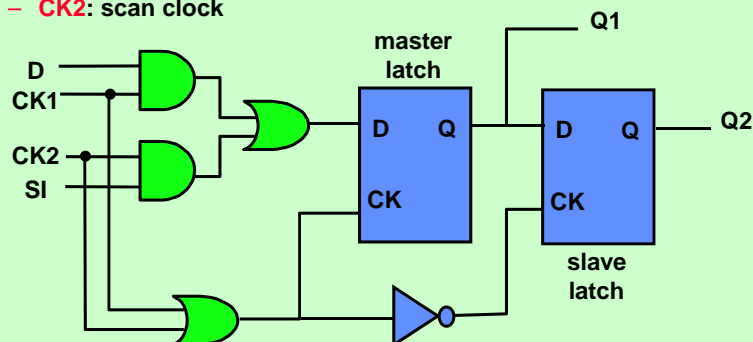
- Only **D-type master-slave flip-flops** are used
- All flip-flop clocks controlled from primary inputs
 - No **gated clock** allowed
- Clocks must not feed data inputs of flip-flops
- Most popularly supported in standard cell libraries



ch5-31

Two-Port Dual-Clock Scan FF

- **Separate normal clock from the clock used for scanning**
 - **D**: normal input data
 - **CK1**: normal clock
 - **SI**: scan input
 - **CK2**: scan clock

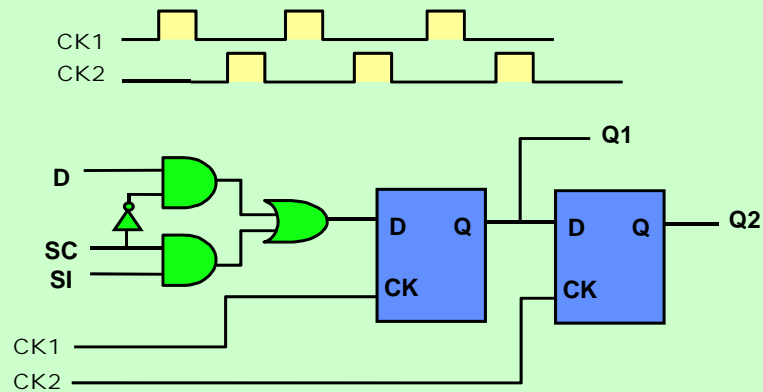


ch5-32

Race-Free Scan FF

- Use two-phase clocking

- CK1 and CK2 are **two-phase non-overlapping** clocks which insure race-free operation



ch5-33

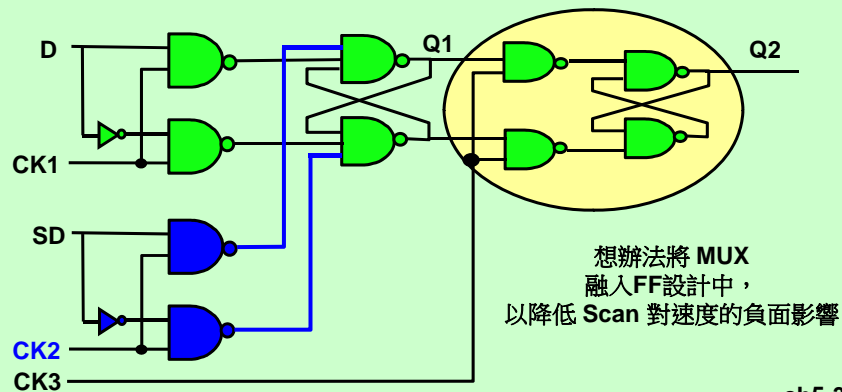
LSSD flip-flop (1977 IBM)

- LSSD: Level Sensitive Scan Design

- Less performance degradation than MUX-scan FF

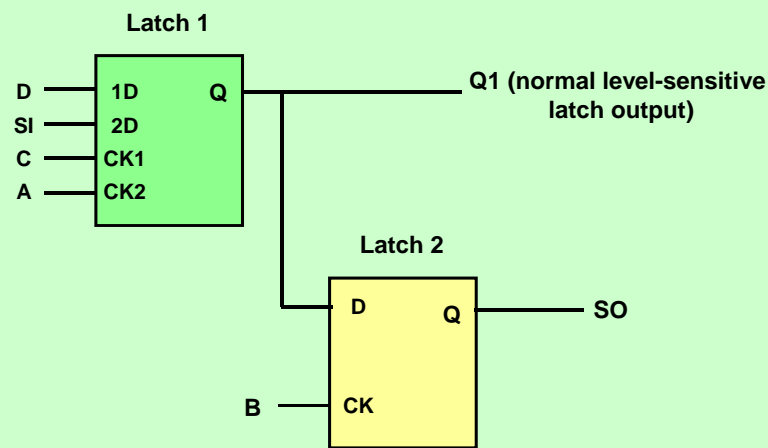
- Clocking

- Normal operation: non-overlapping CK1=1 → CK3=1
- Scan operation: non-overlapping CK2=1 → CK3=1



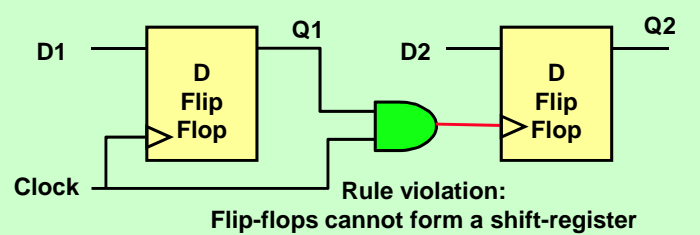
ch5-34

Symbol of LSSD FF

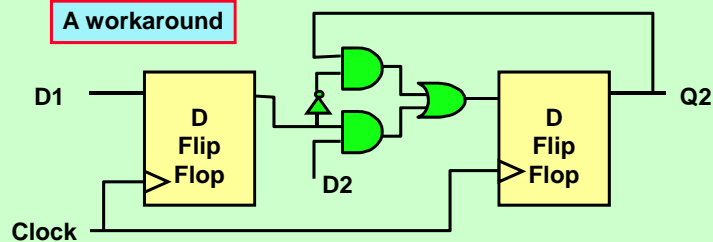


ch5-35

Scan Rule Violation Example



A workaround



All FFs are triggered by the same clock edge
Set and reset signals are not controlled by any internal signals

ch5-36

Some Problems With Full Scan

Major Commercial Test Tool Companies

Synopsys
Mentor-Graphics
SynTest (華騰科技)
Cadence

- Problems
 - Area overhead
 - Possible performance degradation
 - High test application time
 - Power dissipation
- Features of Commercial Tools
 - Scan-rule violation check (e.g., DFT rule check)
 - Scan insertion (convert a FF to its scan version)
 - ATPG (both combinational and sequential)
 - Scan chain reordering after layout

ch5-37

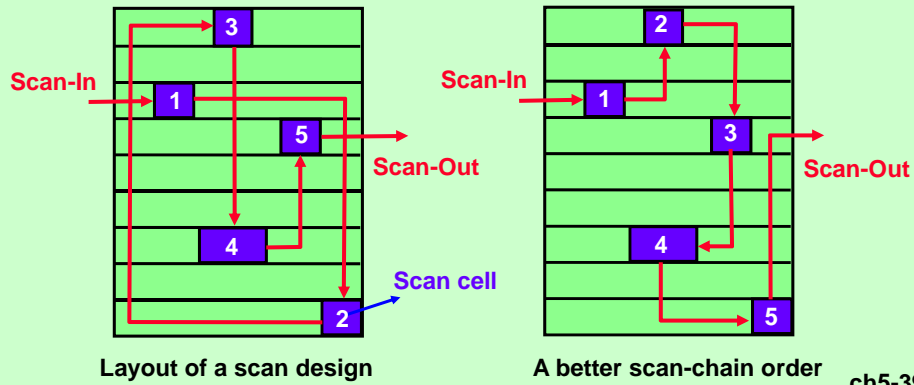
Performance Overhead

- The increase of delay along the normal data paths include:
 - Extra gate delay due to the multiplexer
 - Extra delay due to the capacitive loading of the scan-wiring at each flip-flop's output
- Timing-driven partial scan
 - Try to avoid scan flip-flops that belong to the timing critical paths
 - The flip-flop selection algorithm for partial scan can take this into consideration to reduce the timing impact of scan to the design

ch5-38

Scan-Chain Reordering

- Scan-chain order is often decided at gate-level **without knowing the physical locations of the cells**
- Scan-chain consumes a lot of **routing resources**, and could be minimized by **re-ordering the flip-flops** in the chain after layout is known



Overhead of Scan Design

- Number of CMOS gates = 2000
- Fraction of flip-flops = 0.478

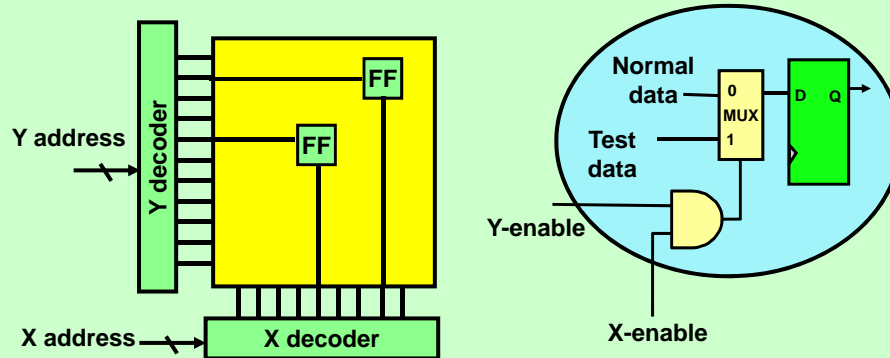
Scan implementation	Predicted overhead	Actual area overhead	Normalized operating frequency
None	0	0	1.0
Hierarchical	14.05%	16.93%	0.87
Optimized	14.05%	11.9%	0.91

ch5-40

Random Access Scan

- Comparison with Scan-Chain

- More **flexible** – any FF can be accessed in constant time
- **Test time** could be reduced
- **More hardware and routing overhead**



ch5-41

Outline

- Introduction
- Ad-Hoc Approaches
- Full Scan
- ➡ • Partial Scan

ch5-42

Partial Scan

- Basic idea

- Select a subset of flip-flops for scan
- Lower overhead (area and speed)
- Relaxed design rules

- Cycle-breaking technique

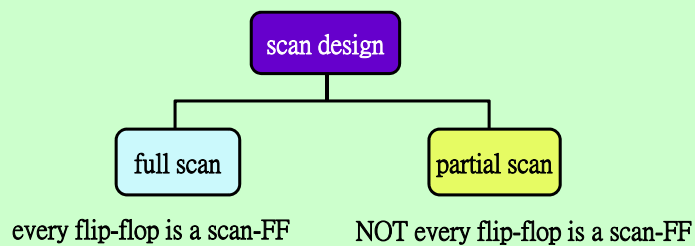
- Cheng & Agrawal, IEEE Trans. On Computers, April 1990
- Select scan flip-flops to simplify sequential ATPG
- Overhead is about 25% off than full scan

- Timing-driven partial scan

- Jou & Cheng, ICCAD, Nov. 1991
- Allow optimization of area, timing, and testability simultaneously

ch5-43

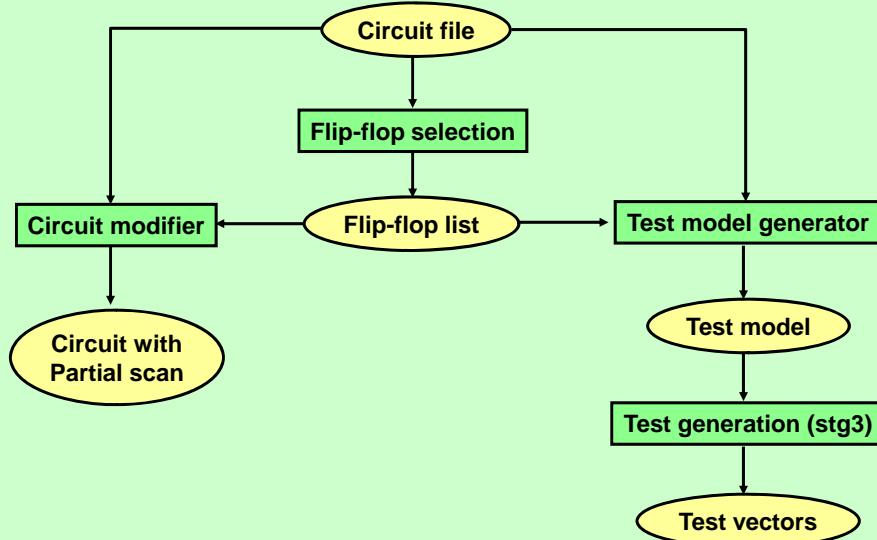
Full Scan vs. Partial Scan



test time	longer	shorter
hardware overhead	more	less
fault coverage	~100%	unpredictable
ease-of-use	easier	harder

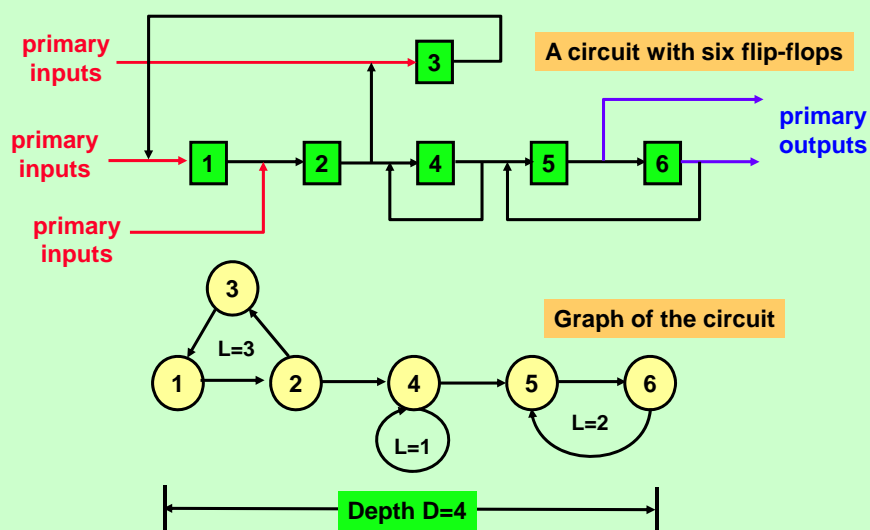
ch5-44

A Partial-Scan DfT Flow



ch5-45

Directed Graph Of A Synchronous Sequential Circuit



ch5-46

Partial Scan For Cycle-Free Structure

- Select minimal set of flip-flops
 - To eliminate some or **all cycles**
- Self-loops of unit length
 - **Are not broken** to reduce scan overhead
 - The number of self-loops in real design can be quite large
- Limit the length of
 - **Consecutive self-loop paths**
 - Long consecutive self-loop paths in large circuits may pose problems to sequential ATPG

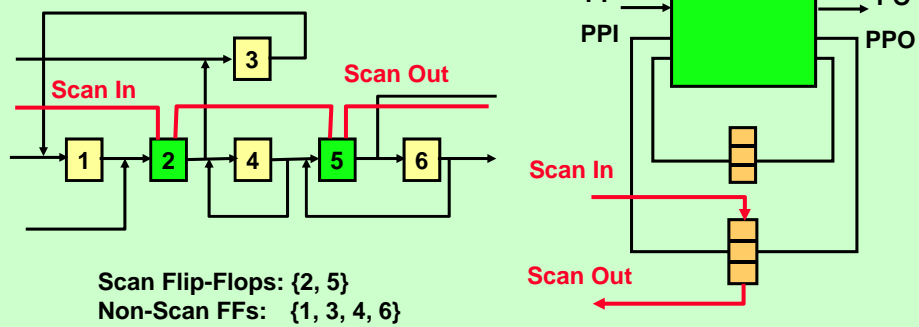
ch5-47

Test Generation for Partial Scan Circuits

- Separate scan clock is used
- Scan flip-flops are removed
 - And their input and output signals are added to the PO/PI lists
- A sequential circuit test generator
 - is used for test generation
- The vector sequences
 - Are then converted into **scan sequences**
 - Each vector is preceded by a **scan-in sequence** to set the states of scanned flip-flops
 - A **scan-out sequence** is added at the end of applying each vector

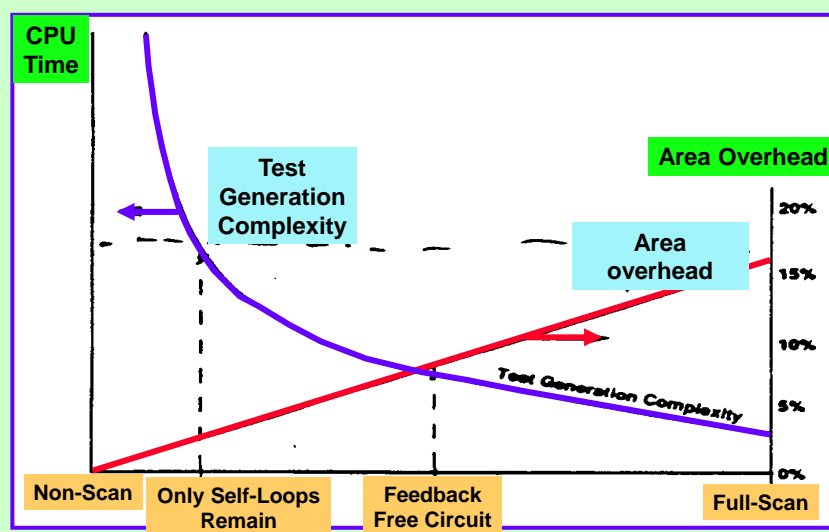
ch5-48

Partial Scan Design



ch5-49

Trade-Off of Area Overhead v.s. Test Generation Effort



ch5-50

Summary of Partial-Scan

- Partial Scan
 - Allows the trade-off between test generation effort and hardware overhead to be automatically explored
- Breaking Cycles
 - Dramatically simplifies the sequential ATPG
- Limiting the Length of Self-Loop Paths
 - Is crucial in reducing test generation effort for large circuits
- Performance Degradation
 - Can be minimized by using timing analysis data for flip-flop selection

ch5-51

Chapter 6

Delay Testing

Acknowledgements:
Mainly based on the lecture notes of
“VLSI Test Principles and Architectures”

ch6-1

Introduction of Delay Testing

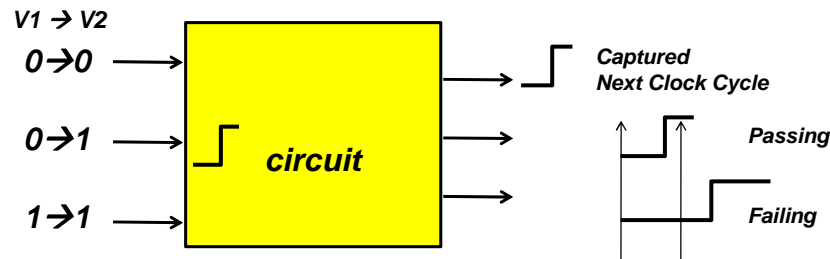
- ❑ **Delay Faulty:**
 - Fault that cause delay across a circuit to violate certain timing constraint
- ❑ **Delay Fault Models:**
 - **Path delay fault**
 - Too much delay along a path
 - **Transition fault** (or Gate delay fault)
 - Too much delay across a particular gate

ch6-2

Basic Delay Testing

□ Delay Test Pattern:

- A **two-pattern test**: <v1, v2>
- v1 is an initialization vector
- v2 causes the fault to be detected



Challenge: The **launch time** and **capture time** are just away by a high-speed clock cycle time

ch6-3

Applications of Delay Tests

□ Launch-off shifting (LOS)

- Aka (also known as) **skewed-load**
- v1 is arbitrary, v2 is derived by a 1-bit shift of v1

□ Launch-off capture (LOC)

- Aka **broadside** or **double-capture**
- v1 is arbitrary, v2 is derived from v1 through the circuit function

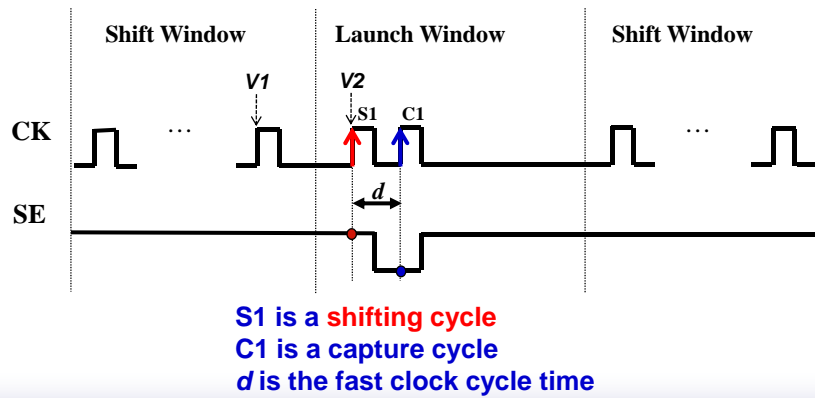
ch6-4

Timing Sequence of Launch-off-Shifting

PROS: Easier Test Generation to achieve a Higher Fault Coverage

CONS: Hard to produce the Scan-Enable signal 'SE'

(Note: 'SE' has to go LOW between S1 and C1)



ch6-5

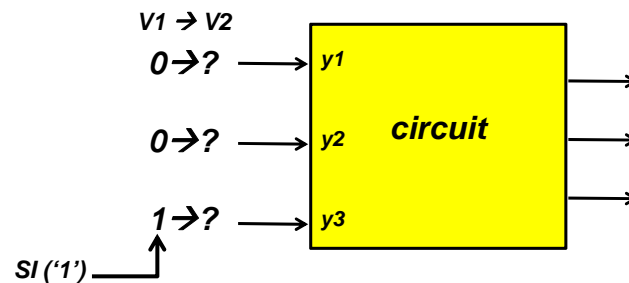
Example of LOS

Question:

v1 is {y1='0', y2='0', y3='1'}

What is vector v2 if using LOS?

Assuming scan chain order y3→y2→y1

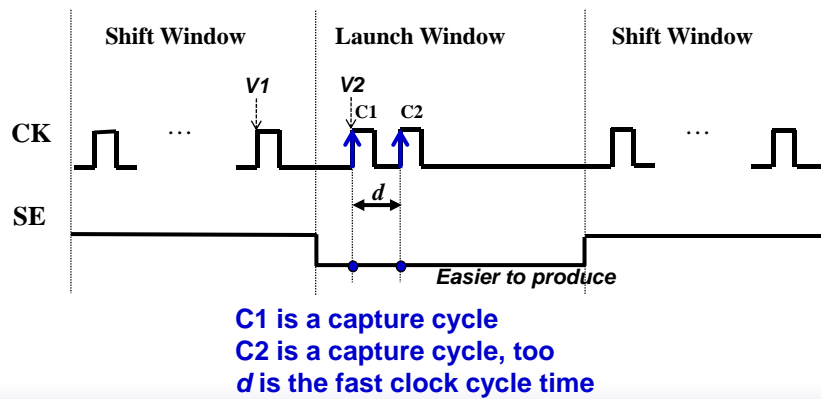


ch6-6

Timing Sequence of Launch-off-Capture

PROS: Scan-Enable signal 'SE' to easy to produce

CONS: Fault Coverage is Lower than LOS



ch6-7

Transition Fault Model

- **Assumption:**
 - a large/gross delay is present at a circuit node
- **Path independence:**
 - Irrespective of which path the effect is propagated, the gross delay defect will **be arriving late at an observable point**
- **De-facto standard in Industry**
 - Simple and the number of faults is linear to circuit size
 - Also needs 2 vectors to test a fault
- **Formulation of transition-fault test generation:**
 - Node **x slow-to-rise (x-STR)** can be modeled simply as **two stuck-at faults**
 - (1) First time-frame: x/1 needs to be excited
 - (2) Second time-frame: x/0 needs to be excited and propagated

ch6-8

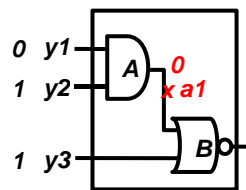
Ex: LOS Pattern Generation

- Target fault:
- A slow-to-rise

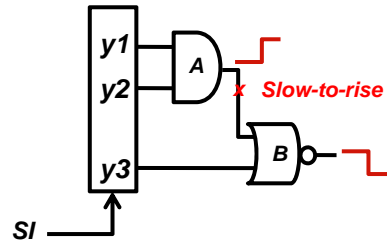
Test Requirement:

1st time frame: initialize a1 to '0'

2nd time frame: detect a2 s-a-0 fault



1st Time Frame



2nd Time Frame

Final 1st Pattern: (y1, y2, y3, SE) = (0, 1, 1, 0) → Shifted to become 2nd Pattern

ch6-9

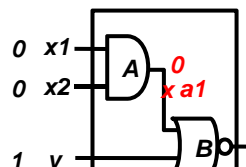
Ex: LOC Pattern Generation

- Target fault:
- A slow-to-rise

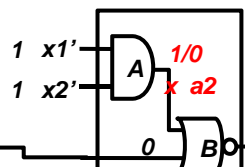
Test Requirement:

1st time frame: initialize a1 to '0'

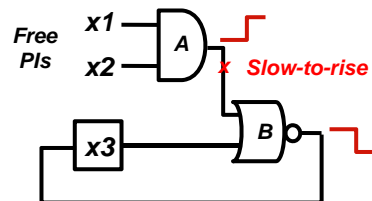
2nd time frame: detect a2 s-a-0 fault



1st Time Frame



2nd Time Frame



ch6-10

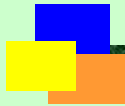
Summary

- ❑ More and more ICs require **delay testing** (or called timing testing, performance testing), to ensure that an IC can perform up to its target speed.
- ❑ Better understand what **LOS, LOC** means, since It's industrial practice.
- ❑ Some IC, e.g., CPU, needs to go through **speed binning process**, to determine the “quality bin” of each IC and its sell price.
- ❑ Delay test is still a tough issue and still evolving. Rigorous delay testing also aims to detect “**small defects**” so as to **reduce the test escape of latent defects** that might hurt an IC's reliability in its field.

ch6-11

國立清華大學電機系

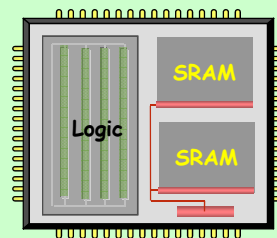
EE-6250
超大型積體電路測試
VLSI Testing



Chapter 7
Built-In Self-Test

Design-for-Testability

- Design activities for generating a set of test patterns with a high fault coverage.
- Methodology
 - Logic
 - Automatic Test Pattern Generation (ATPG)
 - Scan Insertion (to ease the ATPG process)
 - Built-In Self-Test
 - Memory (SRAM, DRAM, ...)
 - Built-In Self-Test



ch7-2

Outline

- ➡ • Basics
 - Test Pattern Generation
 - Response Analyzers
 - BIST Examples
 - Memory BIST

ch7-3

Definition & Advantages of BIST

- Built-In Self-Test (BIST) is a design-for-testability (DFT) technique in which testing (test generation , test application) is accomplished through built-in hardware features.

– [V.D. Agrawal, C.R. Kime, and K.K. Saluja]

- ➡ Can lead to significant test time reduction
Especially attractive for embedded cores

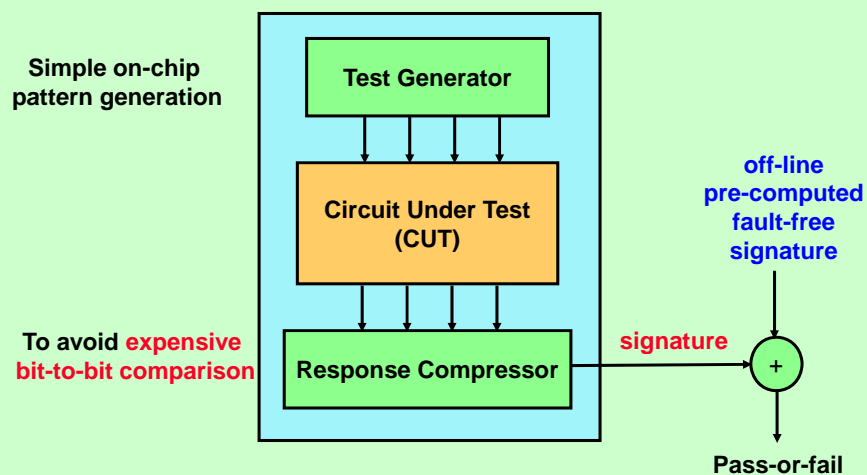
ch7-4

Good Things About BIST

- At-Speed Testing
 - catching timing defects
- Fast
 - reduce the testing time and testing costs
 - a major advantage over scan
- Board-level or system-level testing
 - can be conducted easily in field

ch7-5

General Organization of BIST



ch7-6

Why Compression ?

- Motivation

- Bit-to-bit comparison is infeasible for BIST

- Signature analysis

- Compress a very long **output sequence** into a **single signature**
- Compare the compressed word with the **pre-stored golden signature** to determine the correctness of the circuit

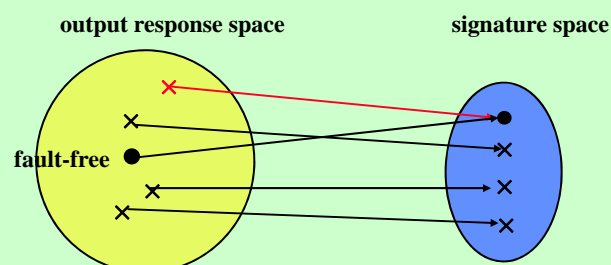
- Problems

- Many output sequences may have the same signature after the compression leading to the aliasing problem
- **Poor diagnosis resolution** after compression

ch7-7

Aliasing Effect in Response Compression

- **Aliasing** - the probability that a faulty response is mapped to the same signature as the fault-free circuit (魚目混珠) 錯變成對的機率



Response compression is a mapping
from the output response space to the signature space
In this example, aliasing prob. = $1 / 4 = 25\%$

ch7-8

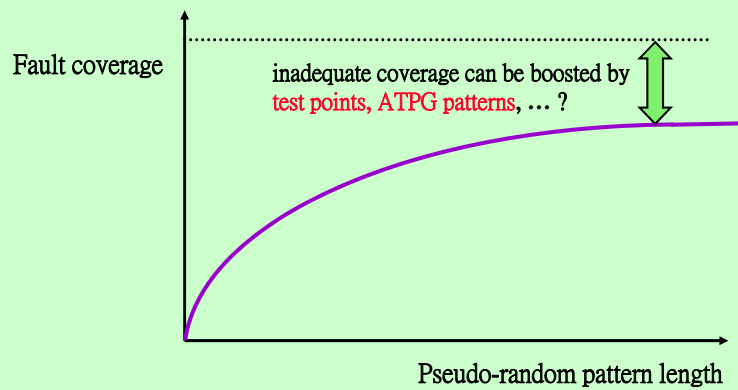
BIST Issues

- Area Overhead
- Performance Degradation
- Fault Coverage
 - Most on-chip generated patterns may not achieve a very high fault coverage
- Diagnosability
 - The chip is even harder to diagnose due to response compression

ch7-9

Random Pattern Resistant Faults

- An RPRF cannot be detected by random patterns
- is a major cause of low fault coverage in BIST

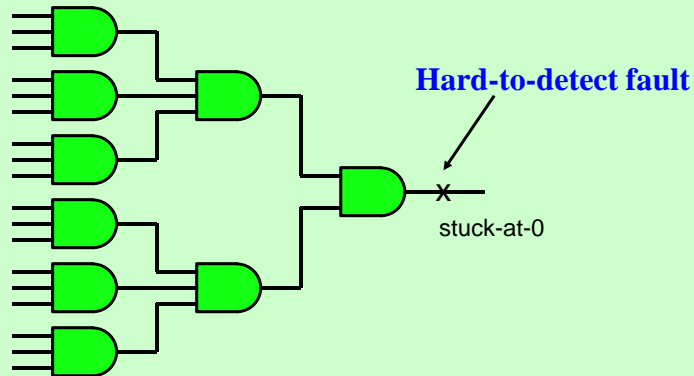


ch7-10

Example: Hard-To-Detect Fault

- Hard-to-detect faults

- Faults that are not covered by random testing
- E.g., an output signal of an 18-input AND gate



ch7-11

Reality of Logic BIST

- BIST is NOT a replacement for scan
 - it is built on top of full-scan
- BIST does NOT result in fewer patterns
 - it usually uses many more patterns than ATPG patterns
- BIST does NOT remove the need for testers
 - tester still required to
 - initiate test
 - read response
 - apply ATPG patterns to other part of IC

ch7-12

BIST Techniques

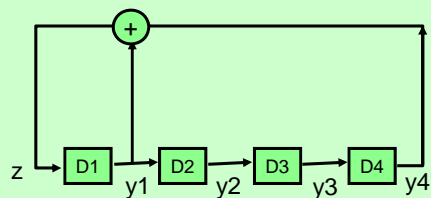
- Stored-Vector Based
 - Micro-instruction support
 - Stored in ROM
- Hardware-Based Pattern Generators
 - Counters
 - Linear Feedback Shift Registers
 - Cellular Automata

ch7-13

Linear Feedback Shift Register (LFSR)

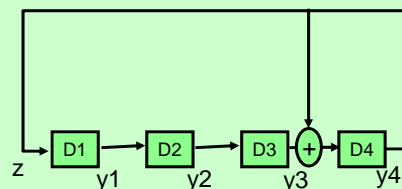
- Flip-Flop: one cycle delay
- XOR gate: modulo-2 addition
- Connection: modulo-2 multiplication

Type 1: Out-Tap



$$z = y4 + y1 = D^4(z) + D(z)$$

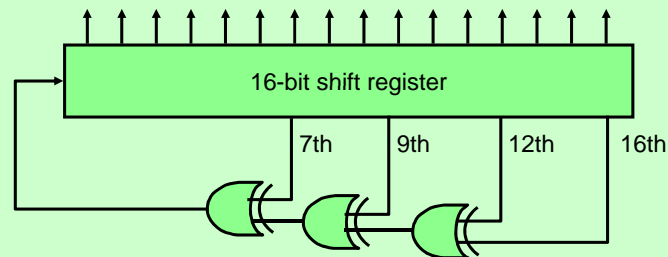
Type 2: In-Tap



$$\begin{aligned} z &= y4 = D(y3 + y4) = D(D^3(z) + z) \\ &= D^4(z) + D(z) \end{aligned}$$

ch7-14

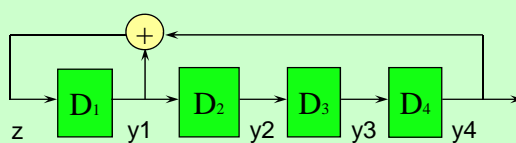
LFSR – Example



This **sixteen-stage LFSR** will **autonomously** generate a maximum length of $2^{16}-1 = 65,535$ state before the sequence repeats.
The **seed** (i.e., initial state of the LFSR) should not be all-0 state.
All 0-state is called a **forbidden seed**.

ch7-15

LFSR Example



$$\begin{bmatrix} y1(t+1) \\ y2(t+1) \\ y3(t+1) \\ y4(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y1(t) \\ y2(t) \\ y3(t) \\ y4(t) \end{bmatrix}$$

Characteristic polynomial

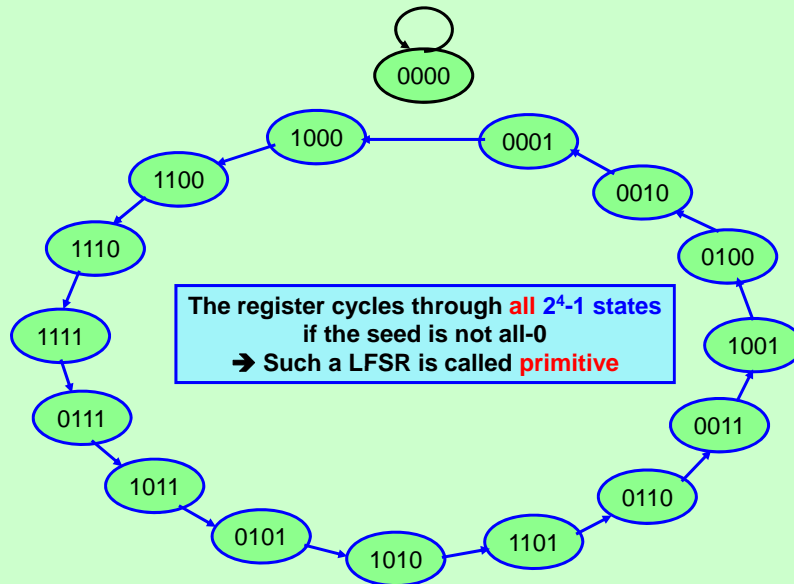
$$g(x) = x^4 + x^1 + 1$$

repeating \rightarrow 1 0 0 0

D_4	D_3	D_2	D_1
1	0	0	0
0	0	0	1
0	0	1	1
0	1	1	1
1	1	1	1
1	1	1	0
1	1	0	1
1	0	1	0
0	1	0	1
1	0	1	1
0	1	1	0
1	1	0	0
1	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

ch7-16

Ex: Primitive LFSR – State Diagram



ch7-17

Primitive Polynomials (Up to Degree 100)

Note: "24 4 3 1 0" means $p(x) = x^{24} + x^4 + x^3 + x^1 + x^0$

<i>n</i>	<i>Exponents</i>	<i>n</i>	<i>Exponents</i>	<i>n</i>	<i>Exponents</i>	<i>n</i>	<i>Exponents</i>
1	0	26	8 7 1 0	51	16 15 1 0	76	36 35 1 0
2	1 0	27	8 7 1 0	52	3 0	77	31 30 1 0
3	1 0	28	3 0	53	16 15 1 0	78	20 19 1 0
4	1 0	29	2 0	54	37 36 1 0	79	9 0
5	2 0	30	16 15 1 0	55	24 0	80	38 37 1 0
6	1 0	31	3 0	56	22 21 1 0	81	4 0
7	1 0	32	28 27 1 0	57	7 0	82	38 35 3 0
8	6 5 1 0	33	13 0	58	19 0	83	46 45 1 0
9	4 0	34	15 14 1 0	59	22 21 1 0	84	13 0
10	3 0	35	2 0	60	1 0	85	28 27 1 0
11	2 0	36	11 0	61	16 15 1 0	86	13 12 1 0
12	7 4 3 0	37	12 10 2 0	62	57 56 1 0	87	13 0
13	4 3 1 0	38	6 5 1 0	63	1 0	88	72 71 1 0
14	12 11 1 0	39	4 0	64	4 3 1 0	89	38 0
15	1 0	40	21 10 2 0	65	18 0	90	19 18 1 0
16	5 3 2 0	41	3 0	66	10 9 1 0	91	84 83 1 0
17	3 0	42	23 22 1 0	67	10 9 1 0	92	13 12 1 0
18	7 0	43	6 5 1 0	68	9 0	93	2 0
19	6 5 1 0	44	27 26 1 0	69	29 27 2 0	94	21 0
20	3 0	45	4 3 1 0	70	16 15 1 0	95	11 0
21	2 0	46	21 20 1 0	71	6 0	96	49 47 2 0
22	1 0	47	5 0	72	53 47 6 0	97	6 0
23	5 0	48	28 27 1 0	73	25 0	98	11 0
24	4 3 1 0	49	9 0	74	16 15 1 0	99	47 45 2 0
25	3 0	50	27 26 1 0	75	11 10 1 0	100	37 0

ch7-18

Galois Field GF(2)

- Operation

- Modulo-2 addition, subtraction, multiplication, and division of binary data

- Properties

- Modulo-2 addition and subtraction are identical
- $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$
- $0-0=0$, $0-1=1$, $1-0=1$, $1-1=0$

$(x^3 + x^2 + x + 1) \times (x^2 + x + 1)$ is given by:

$$\begin{array}{r}
 x^3 + x^2 + x + 1 \\
 \quad x^2 + x + 1 \\
 \hline
 x^3 + x^2 + x + 1 \\
 x^4 + x^3 + x^2 + x \\
 \hline
 x^5 + x^4 + x^3 + x^2 \\
 x^5 + 0 + x^3 + x^2 + 0 + 1 \\
 \hline
 \end{array}$$

Bit-stream multiplication

$$\begin{array}{r}
 x^3 + x^2 + x + 1 \\
 x^2 + x + 1 \overline{) x^5 + 0 + x^3 + x^2 + 0 + 1} \\
 \underline{x^5 + x^4 + x^3} \\
 x^4 + 0 + x^2 \\
 \underline{x^4 + x^3 + x^2} \\
 x^3 + 0 + 0 \\
 \underline{x^3 + x^2 + x} \\
 x^2 + x + 1 \\
 \underline{x^2 + x + 1} \\
 0
 \end{array}$$

Bit-stream division

9

Why LFSR ?

- Simple and regular structure
 - D-flip-flops and XOR gates
- Compatible with scan DFT design
- Capable of exhaustive and/or pseudo exhaustive testing
 - If the LFSR is properly configured
- Low aliasing probability
 - The **fault coverage lost** due to the response compression is **less** than other compression schemes

LFSR – Definitions

- Maximum-length sequence
 - A sequence generated by an **n-stage LFSR** is called a maximum-length sequence if it has a period of **$2^n - 1$**
 - A maximum-length sequence is called **m-sequence**
- Primitive polynomial
 - The **characteristic polynomial** associated with a maximum-length sequence is called a **primitive polynomial**
- Irreducible polynomial
 - A polynomial is **irreducible** if it cannot be factorized into two (or more) parts, i.e., it is not **divisible** by any polynomial other than 1 and itself.

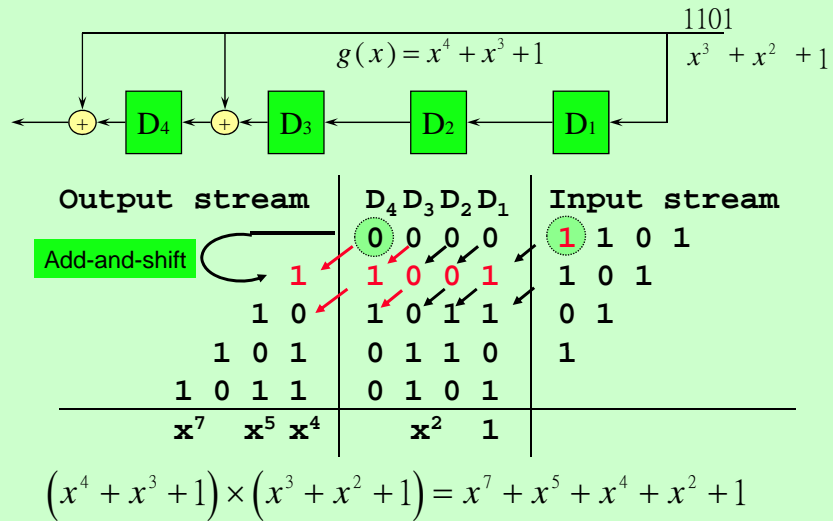
ch7-21

LFSR – Properties

- No. of 1s and 0s
 - The number of 1s in an *m*-sequence differs from the number of 0s by only one
- Pseudo-random sequence
 - The sequence generated by an LFSR is called a **pseudo-random sequence**
- The correlation
 - Between any two output bits is very close to zero
- Consecutive run of 1s and 0s
 - An *m*-sequence produces an equal number of runs of 1s and 0s.
 - In every *m*-sequence, one **half the runs have length 1**, **one fourth have length 2**, one eighth have length 3, and so forth

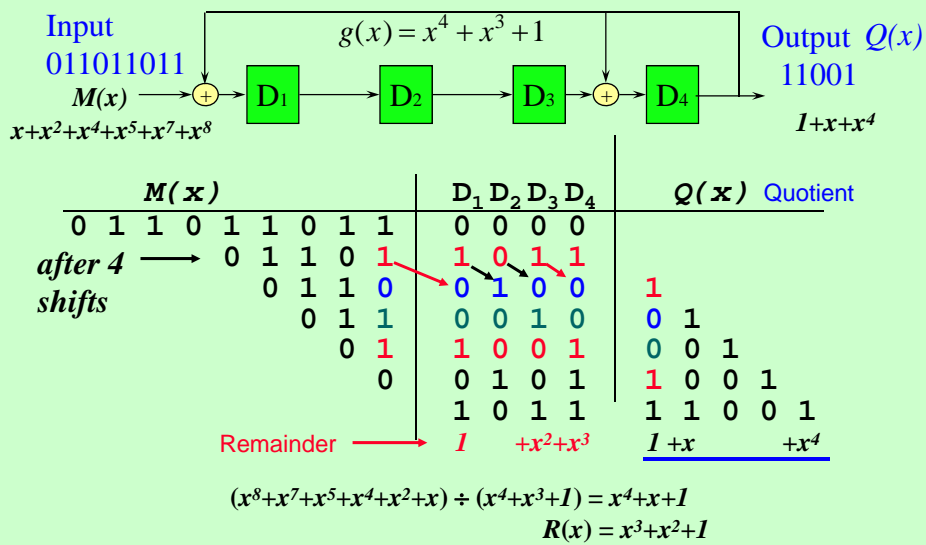
ch7-22

LFSR – Polynomial Multiplication



ch7-23

LFSR – Polynomial Division (Example)



ch7-24

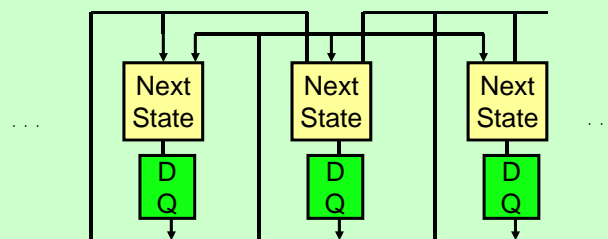
LFSR – Summary

- LFSRs have two types
 - In-tap and Out-tap
- LFSRs
 - Can be used to implement polynomial multiplication and division in GF(2)
- As polynomial multiplier
 - LFSRs are capable of generating pseudo random vectors
- As polynomial divisors
 - LFSRs are capable of compressing test response

ch7-25

Cellular Automaton (CA)

- An one-dimensional array of cells
- Each cell contains a storage device and next state logic
- Next state is a function of current state of the cell and its neighboring cells



Three-cell neighbor

ch7-26

Cellular Automata – Name

- Name of CA functions

– Is determined by its truth table

State	A0	A1	A2	A3	A4	A5	A6	A7	Next State K-Map F _{CA}			
C _{i+1}	0	0	0	0	1	1	1	1	A ₀	A ₂	A ₄	A ₆
C _i	0	0	1	1	0	0	1	1	A ₁	A ₃	A ₅	A ₇
C _{i-1}	0	1	0	1	0	1	0	1				

$$Name = \sum_{i=0}^7 A_i 2^i \text{ (defined by Wolfram)}$$

Example: $F_{CA} = C_{i-1} \oplus C_i$

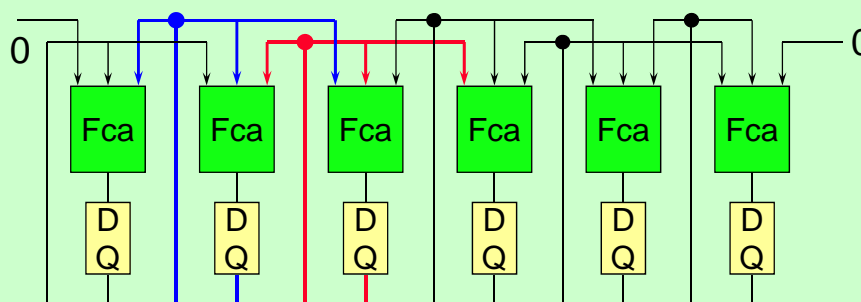
$C_i C_{i-1}$		00	01	11	10
C_{i+1}	0	0	1	0	1
	1	0	1	0	1

$$Name = 64 + 32 + 4 + 2 = 102$$

ch7-27

Cellular Automata – Hardware

CA with Null Boundary Condition

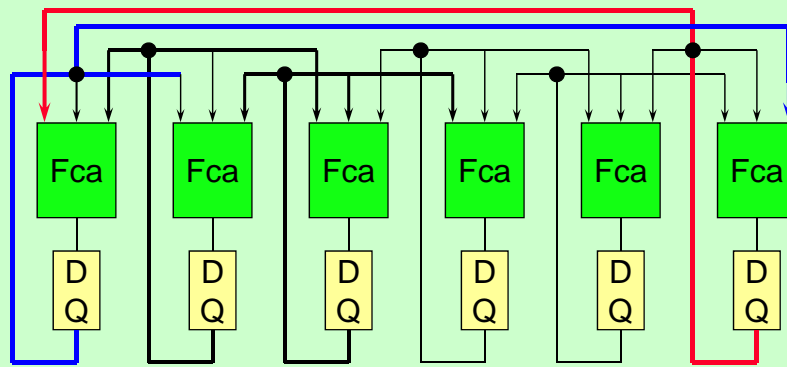


Standard – All the CAs are of the same type
Hybrid – The CAs are of different type

ch7-28

Cellular Automata – Hardware

CA with cyclic Boundary Condition



ch7-29

Outline

- Basics
- ➡ • Test Pattern Generation
 - How to generate patterns on chip using minimum hardware, while achieving high fault coverage
- Response Analyzers
- BIST Examples
- Memory BIST

ch7-30

On-Chip Pattern Generation

PG Hardware	Pattern Generated
<ul style="list-style-type: none"> • Stored Patterns • Counter Based • LFSR Based • Cellular Automata 	<ul style="list-style-type: none"> • Deterministic • Pseudo-Exhaustive • Pseudo-Random • Pseudo-Random

Pseudo Random Patterns: Random patterns with a specific sequence defined by a **seed**

ch7-31

Counter Based Pattern Generation

- Generates regular test sequences
 - Such as **walking sequence** and counting sequence for memory **interconnect** testing

cycle	Walking Sequence	Counting Sequence
1	1 0 0 0 0 0 0 0	0 0 0
2	0 1 0 0 0 0 0 0	0 0 1
3	0 0 1 0 0 0 0 0	0 1 0
4	0 0 0 1 0 0 0 0	0 1 1
5	0 0 0 0 1 0 0 0	1 0 0
6	0 0 0 0 0 1 0 0	1 0 1
7	0 0 0 0 0 0 1 0	1 1 0
8	0 0 0 0 0 0 0 1	1 1 1

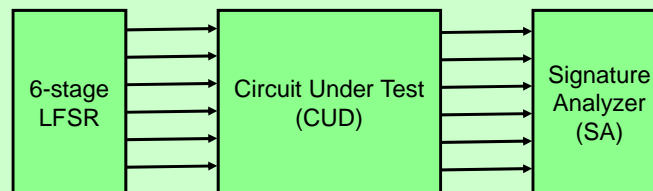
line id 1 2 3 4 5 6 7 8

← coupling between interconnects can be tested by walking sequence

ch7-32

On-Chip Exhaustive Testing

- Exhaustive testing
 - Apply all possible input combinations to CUD
 - A **complete functional testing**
 - 100% coverage on all possible faults
- Limitation
 - Only applicable for circuits with medium number of inputs



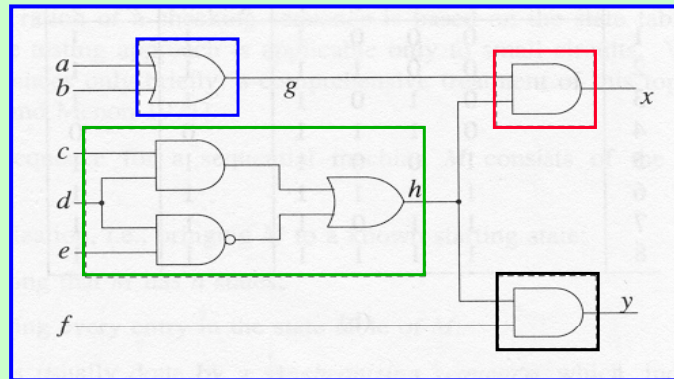
ch7-33

Pseudo Exhaustive Testing (PET)

- Apply **all possible input combinations** to every **partitioned sub-circuits**
- **100% fault coverage** on single faults and **multiple faults** within the sub-circuits
- **Test time** is determined by the number of sub-circuits and the number of inputs to the sub-circuit
- **Partitioning** is a difficult task

ch7-34

Example for Pseudo-Exhaustive Testing



10 vectors are enough to **pseudo-exhaustively** test this circuit,
Compared to $2^6=64$ vectors for naive exhaustive testing

ch7-35

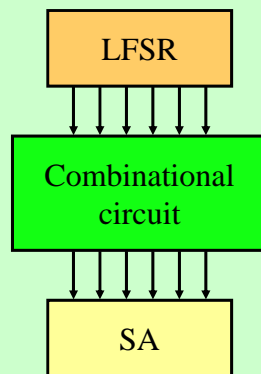
LFSR-Based Pattern Generation

- Apply random test sequence generated by **LFSR/CA**
- Simplest to design and implement
- **Lowest** in hardware overhead
- **Fault coverage**
 - Is a function of the **test length** and the **random testability of the circuits**
 - Certain circuits are more **resistant** to random patterns than others

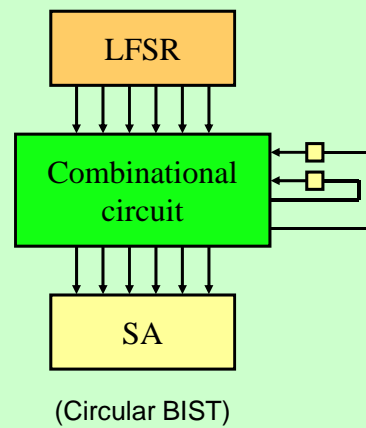
ch7-36

Pseudo Random Testing Hardware

Combinational

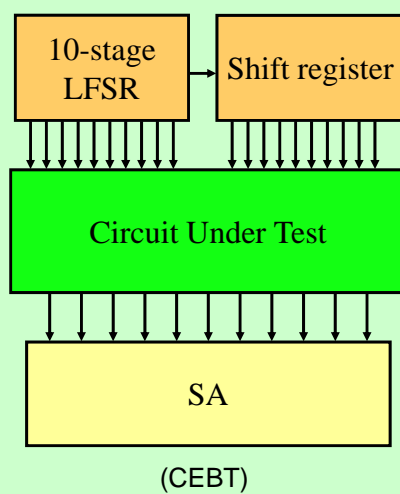


Sequential

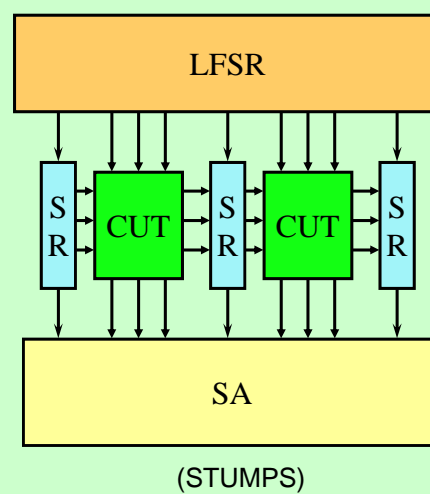


ch7-37

BIST – Pseudo Random Testing Hardware



test-per-clock configuration



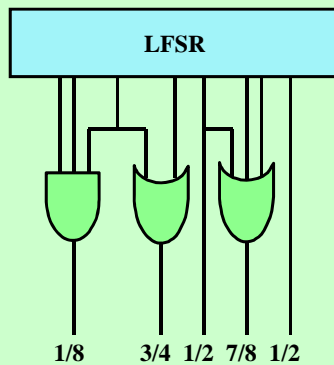
test-per-scan configuration

ch7-38

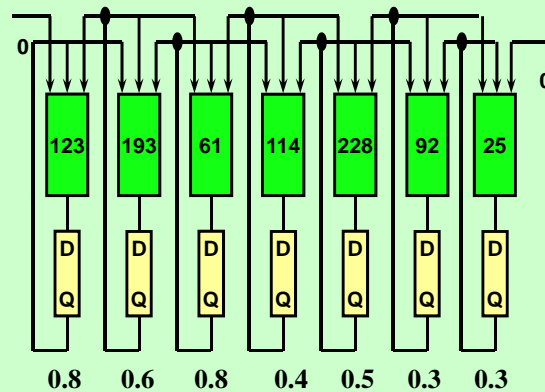
Weighted Pseudo Random Testing

It was observed that **weighted random patterns** could achieve **higher fault coverage** in most cases !

LFSR Based



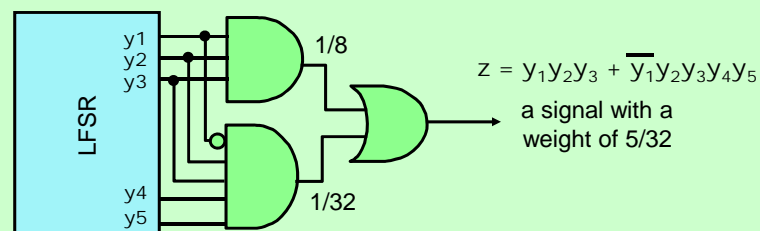
Weighted Cellular Automaton



ch7-39

Signal of An Arbitrary Weight

- To implement a signal
 - with a **signal-1 probability (weight)** of $5/32$
- Procedure
 - Decompose** into a sum of basic weights
 $5/32 = 4/32 + 1/32 = 1/8 + 1/32$
 - Use AND and OR gates to realize the weight



ch7-40

Outline

- Basics
- Test Pattern Generation
- ➡ • Response Analyzers
 - How to compress the output response without losing too much accuracy
- BIST Examples
- Memory BIST

ch7-41

Types of Response Compression

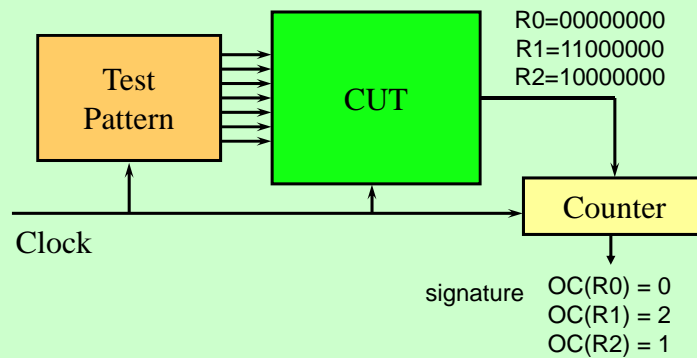
- Ones-counting compression
- Transition-counting compression
- Signature Analysis

ch7-42

Ones-Counting Signature

- Procedure

- Apply the predetermined patterns
- Count the number of ones in the output sequence



ch7-43

Zero-Aliasing Test Set for Ones-Counting

- Notations

- T0: set of test vectors whose fault-free response is 0
- T1: set of test vectors whose fault-free response is 1

- Theorem

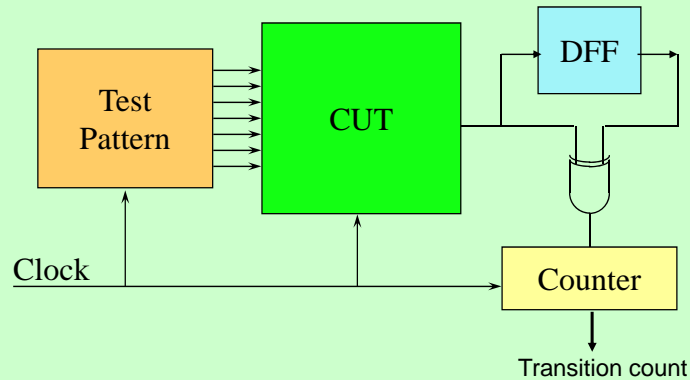
- The following new test set does **NOT** suffer from fault masking using ones count testing
- $T = \{T0, (|T0|+1) \text{ copies of every pattern in } T1\}$
- Note that the **fault masking** only occurs when a **fault** is detected by the same number of patterns in T0 and T1; the above new test set avoid this condition

ch7-44

Transition-Counting Signature

- Procedure

- Apply predetermined patterns
- Count the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions



ch7-45

Aliasing of Transition-Counting

- Consider a sub-sequence of bits

$(\dots r_{j-1} \textcolor{red}{r_j} r_{j+1} \dots)$

If r_{j-1} is not equal to r_{j+1} , then an error occurring at $\textcolor{red}{r_j}$ will not be detected by transition counting.

- Example

1. $(0, \textcolor{red}{1}, 1) \rightarrow (0, \textcolor{red}{0}, 1)$
2. $(0, \textcolor{red}{0}, 1) \rightarrow (0, \textcolor{red}{1}, 1)$
3. $(1, \textcolor{red}{1}, 0) \rightarrow (1, \textcolor{red}{0}, 0)$
4. $(1, \textcolor{red}{0}, 0) \rightarrow (1, \textcolor{red}{1}, 0)$

ch7-46

Aliasing of Transition Counting

- Aliasing Probability

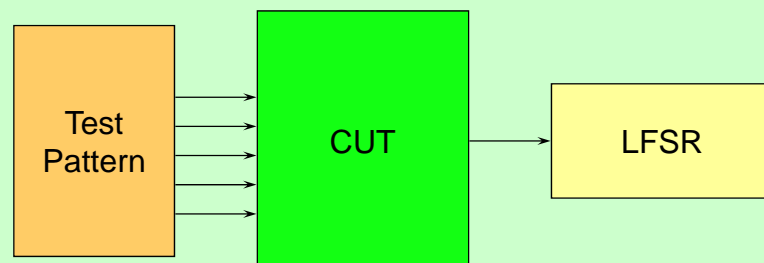
- Notations
 - m : the test length
 - r : the number of transitions
- Highest when $r=m/2$
- No aliasing when $r=0$ or $r=m$
- For combinational circuits, **permutation** of the input sequence results in a different signature
- One can **reorder** the test sequence to **minimize the aliasing probability**

ch7-47

Signature Analysis by LFSR

- Procedure

- Apply predetermined patterns
- **Divide** the output sequence by LFSR



ch7-48

Example: Aliasing Probability

- Assume that

- Output number to be compressed has $m=4$ bits
- The **compression** is done by dividing **output number** by a **divisor of 2^n-1** , (e.g., the divisor is $2^2-1 = 3$ when $n=2$)
- The **remainder** is taken as the signature

- Possible signatures

output = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

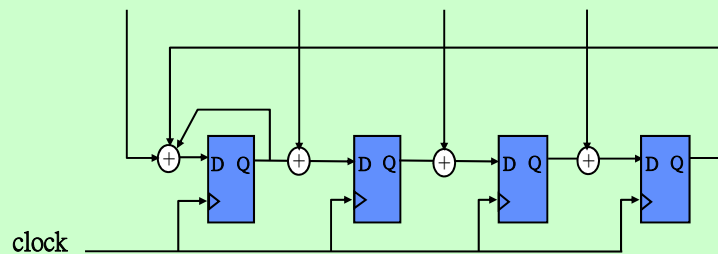
remainder = 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0

aliasing prob. when signature is 0 = $(2^m/(2^n-1)) / 2^m$
 $= 1/(2^n-1) \sim 2^{-n}$

ch7-49

Multiple Input Shift Register (MISR) (Temporal Compression)

- A MISR compacts responses from multiple circuit outputs into a signature



Aliasing probability of m stage = 2^{-m}

ch7-50

Outline

- Basics
- Test Pattern Generation
- Response Analyzers
-  • BIST Examples
- Memory BIST

ch7-51

Key Elements in a BIST Scheme

- Test pattern generator (TPG)
- Output response analyzer (ORA)
 - Also called Signature Analyzer (SA)
- The circuit under test (CUT)
- A distribution system (DIST)
 - which transmits data from TPG's to CUT's and from CUT's to ORA's
 - e.g., wires, buses, multiplexers, and scan paths
- A BIST controller
 - for controlling the BIST circuitry during self-test
 - could be off-chip

ch7-52

HP Focus Chip (Stored Pattern)

- Chip Summary

- 450,000 NMOS devices, 300,000 Nodes
- 24MHz clocks, 300K-bit on-chip ROM
- Used in HP9000-500 Computer

- BIST Micro-program

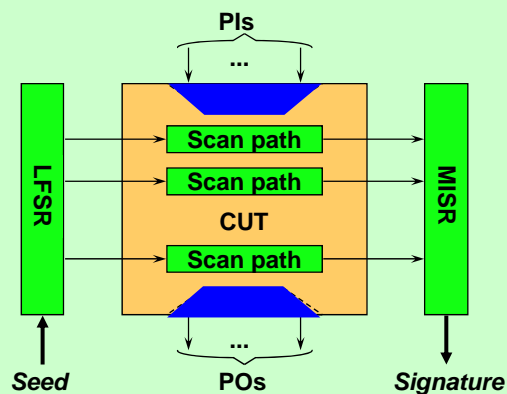
- Use **microinstructions** dedicated for testing
- **100K-bit** BIST micro-program in CPU **ROM**
- Executes **20 million** clock cycles
- Greater than 95% stuck-at coverage
- A power-up test used in **wafer test**, **system test**, **field test**

ch7-53

Logic BIST Example

- Features

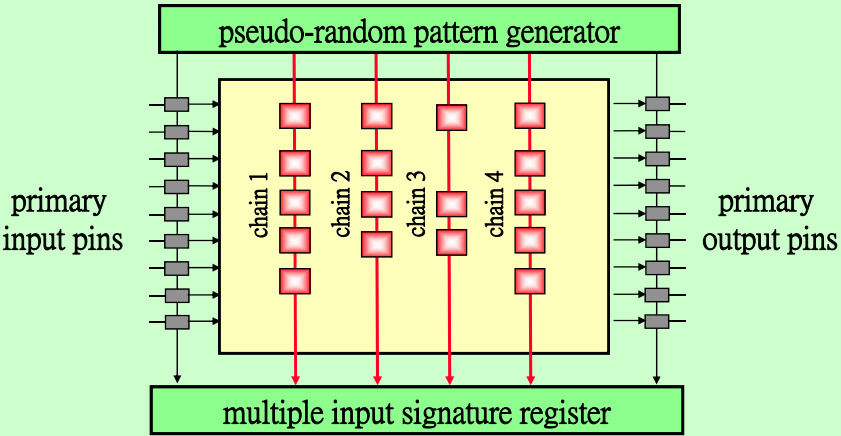
- [Bardell 1982, 84]
- Self-Test using LFSR and Parallel MISR
- **Multiple scan chains** to reduce test time



ch7-54

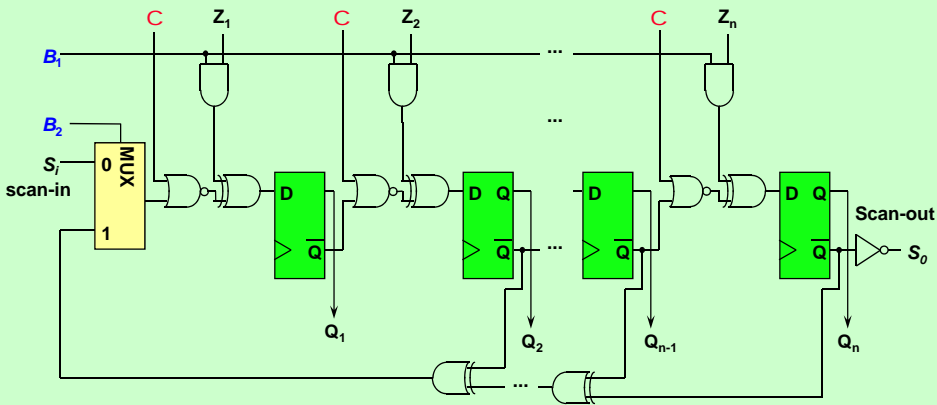
Scan-Based Logic BIST Architecture

called **STUMPS** architecture by Mentor Graphics



ch7-55

Built-In Logic Block Observation (BILBO)



B_1	B_2	operation mode	C
0	0	shift register	0
0	1	LFSR pattern generation	0
1	1	MISR response compressor	0
1	0	parallel load (normal operation)	1

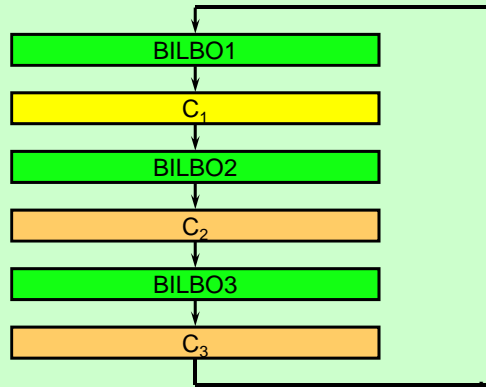
ch7-56

Example: BILBO-Based BIST

- Test procedure

- each logic block C1, C2, C3 are tested in a **serial manner**
- BIST controller needs to **configure each BILBO** registers properly during self-testing

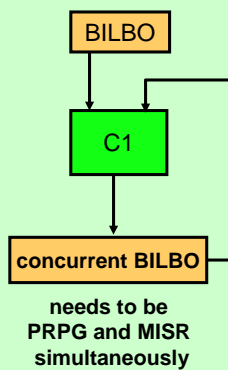
when testing C1
BILBO1 is a PRPG
BILBO2 is a MISR



ch7-57

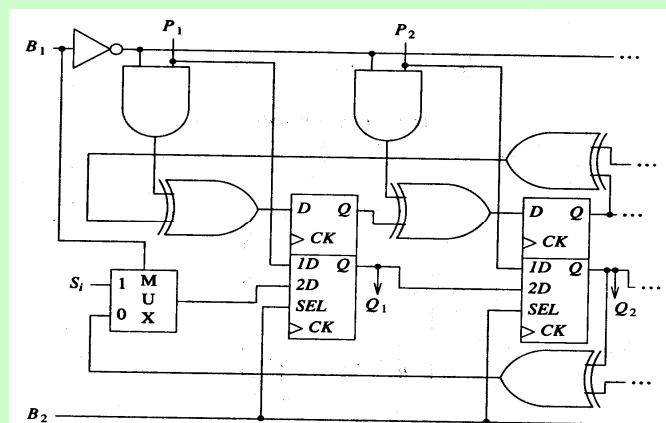
Concurrent BILBO

Logic with self-loop



B_1	B_2	Operation mode
—	0	Normal
1	1	Scan
0	1	PRPG/MISR

top-row of D-FFs → MISR
bottom-row of D-FFs → PRPG



ch7-58

Outline

- Basics
- Test Pattern Generation
- Response Analyzers
- BIST Examples
- ➡ • Memory BIST

ch7-59

The Density Issues

- Historical π -Rule
 - The number of bits per chip has **quadrupled** roughly every 3.1 (or π) years
- Density Induced Faults
 - The cells are **closer** together
 - More sensitive to influences of **neighbors**
 - More vulnerable to **noise** on the address and data lines

ch7-60

Test Time May Get Too Long !

- For today's memory chips
 - Test time becomes a big issue !
 - We can afford nothing but linear test algorithm
- Example
 - assume that the clock cycle time is 100 ns

Algorithm complexity Capacity n	Testing time (in seconds)			
	$64n$	$n \cdot \log_2 n$	$3n^{3/2}$	$2n^2$
16k	0.1	0.023	0.63	54
64k	0.4	0.1	5.03	14 Mins
256k	1.7	0.47	40.3	3.8 Hrs
1M	6.7	2.1	5.4 Mins	61 Hrs
4M	26.8	9.2	43 Mins	41 Days
16M	1.8 Mins	40.3	5.7 Hrs	2 Years

ch7-61

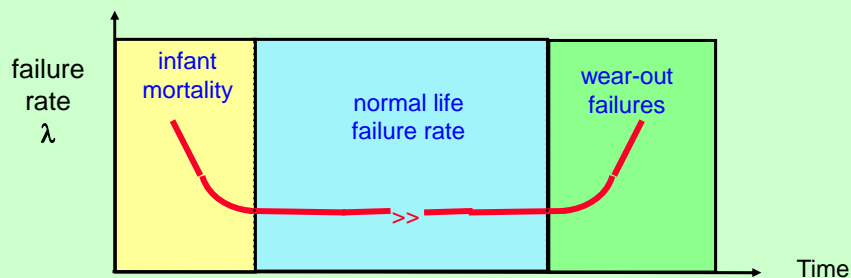
IC Failure Rate Versus Time

Def: failure rate

The no. of failures per unit time as a fraction of total population

IC's failure rate is like a bathtub curve with three stages:

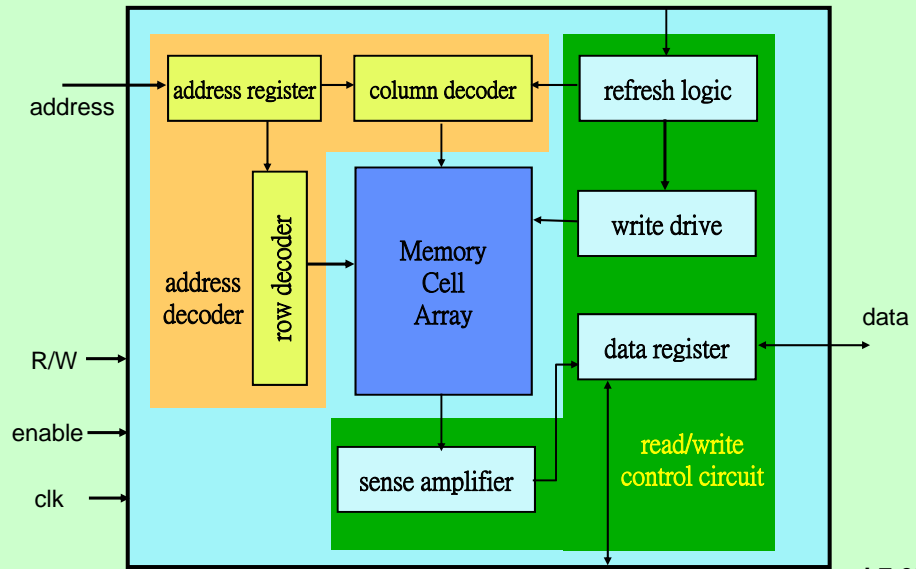
1. Infant mortality stage: typically a few weeks
2. Normal life failure stage: up to 25 years or so
3. Wear-out stage



Short period of accelerated stress test prior to shipment
 → To eliminate the infant mortality

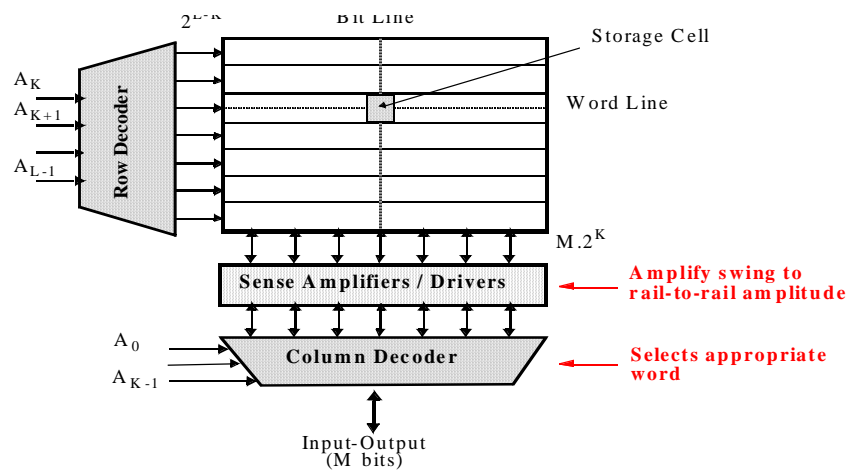
ch7-62

Memory Model



Memory Array

Problem: ASPECT RATIO or HEIGHT >> WIDTH



ch7-64

Fault Models

- Stuck-At Faults (SAF)
 - cell, data line, address line, etc.
- Open Faults (SAF)
 - open in data line or in address line
- Transition Faults (TF)
 - Cell can be set to 0, but not to 1
- Address Faults (AF)
 - faults on decoders
- Coupling Faults (CF)
 - **short** or **cross-talk** between data (or address) lines
 - A cell is affected by **one** of its neighboring cells
- Neighborhood Pattern Sensitive Fault (NPSF)
 - A cell is affected by when its **neighbors form a pattern**

1	0	1
0	1	0
1	0	1

cell is affected

ch7-65

Example Faults

- SAF : Cell stuck
- SAF : Driver stuck
- SAF : Read/write line stuck
- SAF : Chip-select line stuck
- SAF : Data line stuck
- SAF : Open in data line
- CF : Short between data lines
- CF : Cross-talk between data lines
- AF : Address line stuck
- AF : Open in address line
- AF : Open decoder
- AF : Shorts between address lines
- AF : Wrong access
- AF : Multiple access
- TF : Cell can be set to 0 but not to 1 (or vice-versa)
- NPSF : Pattern sensitive interaction between cells

Fault Models

ch7-66

Simple Test Algorithms

- Test Algorithm

- is an abstract description of a sequence of test patterns.

- Commonly Used Algorithms

- Background patterns

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

- Checkerboard patterns

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

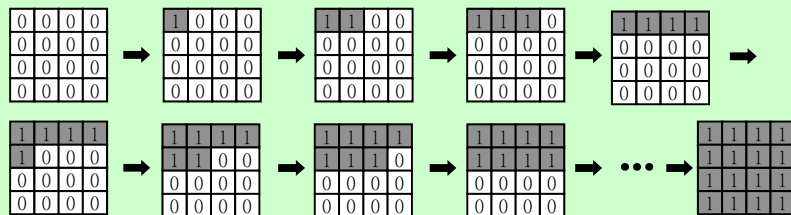
1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

- March Patterns

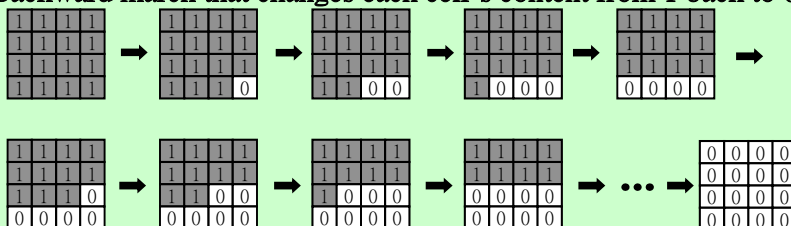
ch7-67

A March Algorithm

(Forward march that changes each cell's content from 0 to 1)

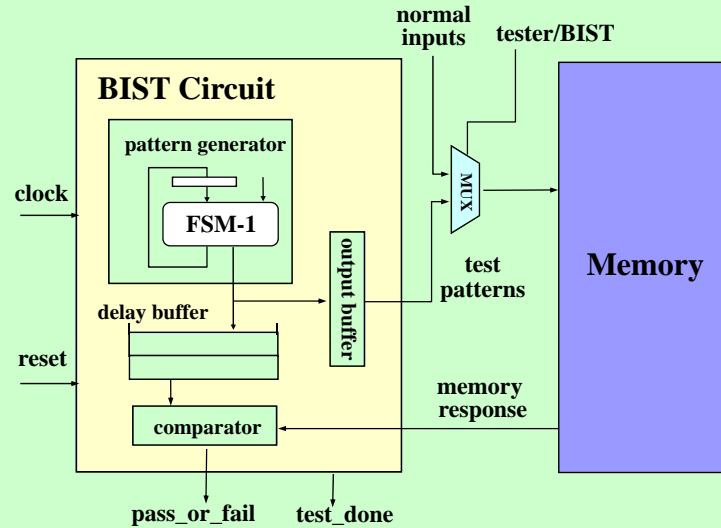


(Backward march that changes each cell's content from 1 back to 0)



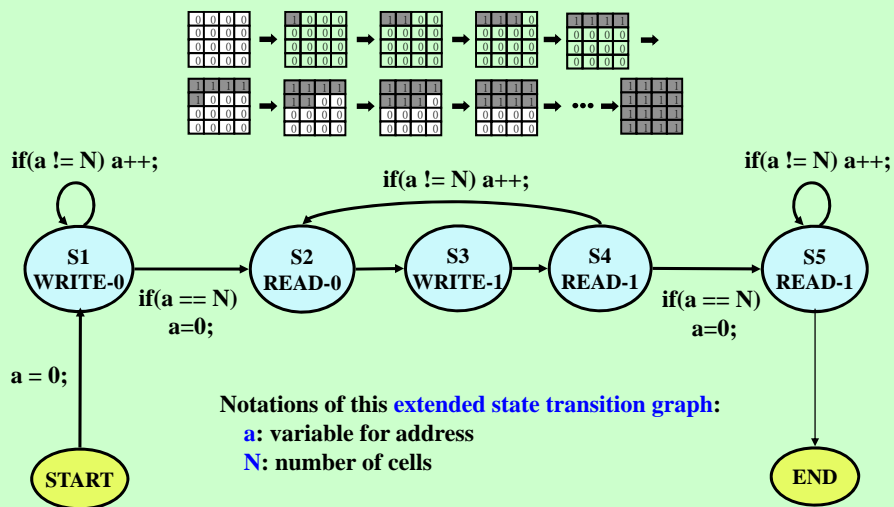
ch7-68

Example: A Memory BIST



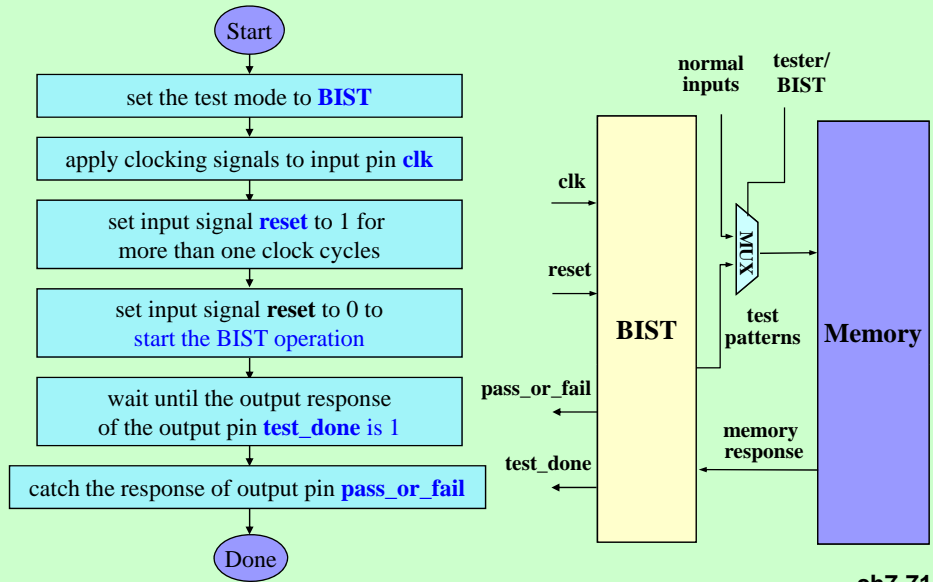
ch7-69

Finite State Machine for March Alg.

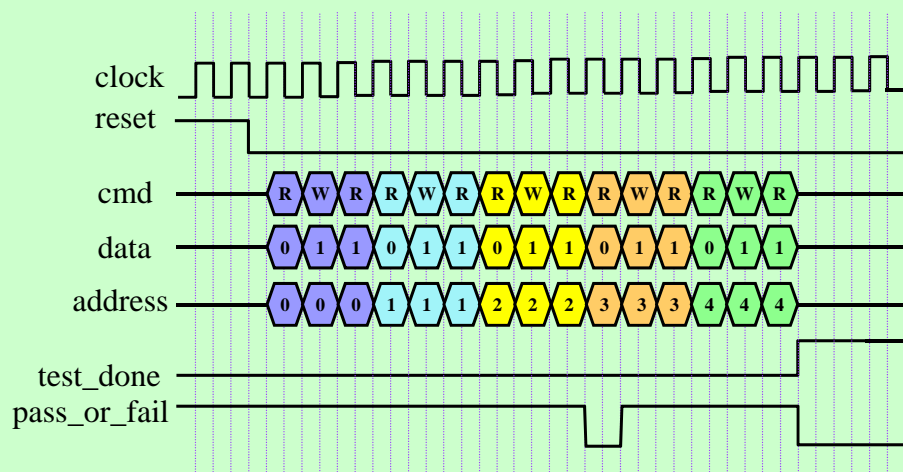


ch7-70

Testing Procedure of BISTed Memory

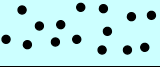





A Waveform Example



ch7-72

Quality Measures of BIST

BIST-vs.-Tester Profile		Tester	
		pass	fail
B I S T	pass	(I) 	(III)  漏網之魚
	fail	(II)  誤殺者	(IV) 

To minimize region (II) and (III):

1. False Negative Ratio: (II) / #chips e.g., (1/20) = 5%
2. False Positive Ratio: (III) / #chips e.g., (2/20) = 10%

ch7-73

Chapter 8

Test Compression

Acknowledgements:

**Mainly based on the lecture notes of
Chapter 6, “VLSI Test Principles and Architectures”**

ch8-1

What is this Chapter about?

- ❑ **Introduce the basic concepts of test data compression**
- ❑ **Focus on stimulus compression and response compaction techniques**
- ❑ **Present and discuss commercial tools on test compression**

ch8-2

Test Compression

- ❑ Introduction
- ❑ Test Stimulus Compression
- ❑ Test Response Compaction
- ❑ Industry Practices
- ❑ Concluding Remarks

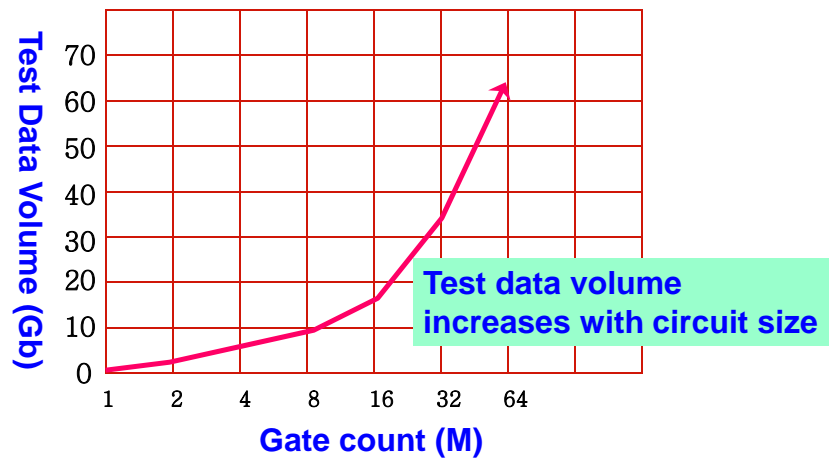
ch8-3

Introduction

- ❑ **Why do we need test compression?**
 - Test data volume
 - Test time
 - Test pins
- ❑ **Why can we compress test data?**
 - Test vectors have a lot of “don’t care” (X’s)

ch8-4

Test Data Volume v.s. Gate Count



(Source: Blyler, *Wireless System Design*, 2001)

ch8-5

Test Compression Categories

❑ Test Stimulus Compression

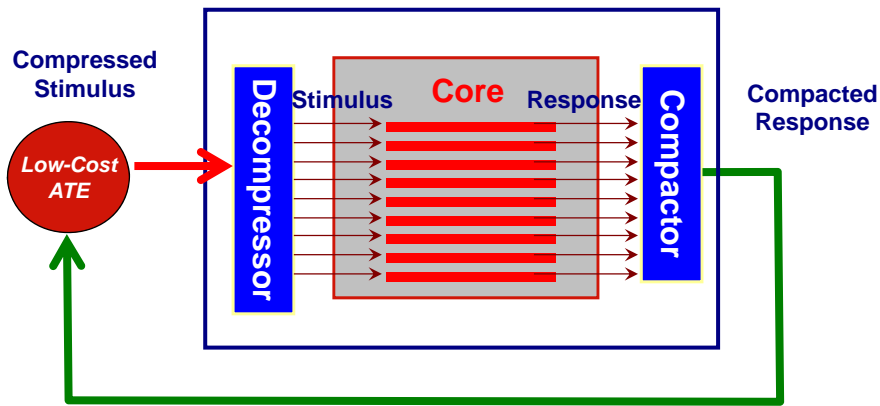
- (1) Code-based schemes
- (2) Linear-decompression-based schemes
- (3) Broadcast-scan-based schemes

❑ Test Response Compaction

- Space compaction
- Time compaction
- Mixed time and space compaction

ch8-6

Architecture for Test Compression



ch8-7

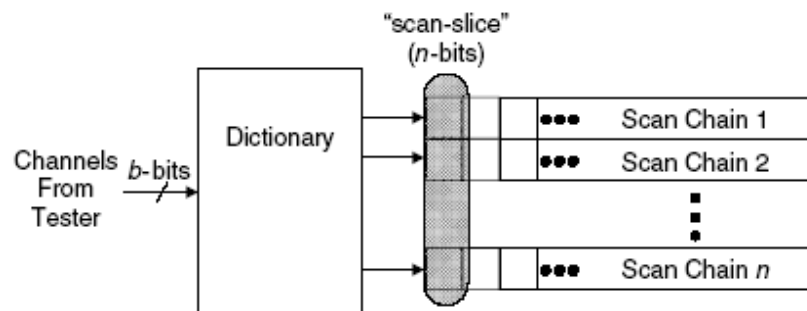
Test Stimulus Compression

- ➡ ☐ **Code-based schemes**
 - Dictionary code (fixed-to-fixed)
 - Huffman code (fixed-to-variable)
 - Run-length code (variable-to-fixed)
 - Golomb code (variable-to-variable)
- ☐ **Linear-decompression-based schemes**
- ☐ **Broadcast-scan-based schemes**

ch8-8

Dictionary Code

□ Dictionary code (fixed-to-fixed)



A test vector is considered as a **two-dimensional image**
In multiple scan chains (e.g., n scan chains as shown)

ch8-9

Huffman Code

□ Huffman code (fixed-to-variable)

Symbol	Frequency	Pattern	Huffman Code	Selective Code
S_0	22	0010	10	10
S_1	13	0100	00	110
S_2	7	0110	110	111
S_3	5	0111	010	00111
S_4	3	0000	0110	00000
S_5	2	1000	0111	01000
S_6	2	0101	11100	00101
S_7	1	1011	111010	01011
S_8	1	1100	111011	01100
S_9	1	0001	111100	00001
S_{10}	1	1101	111101	01101
S_{11}	1	1111	111110	01111
S_{12}	1	0011	111111	00011
S_{13}	0	1110	—	—
S_{14}	0	1010	—	—
S_{15}	0	1001	—	—

Each is code from ATE

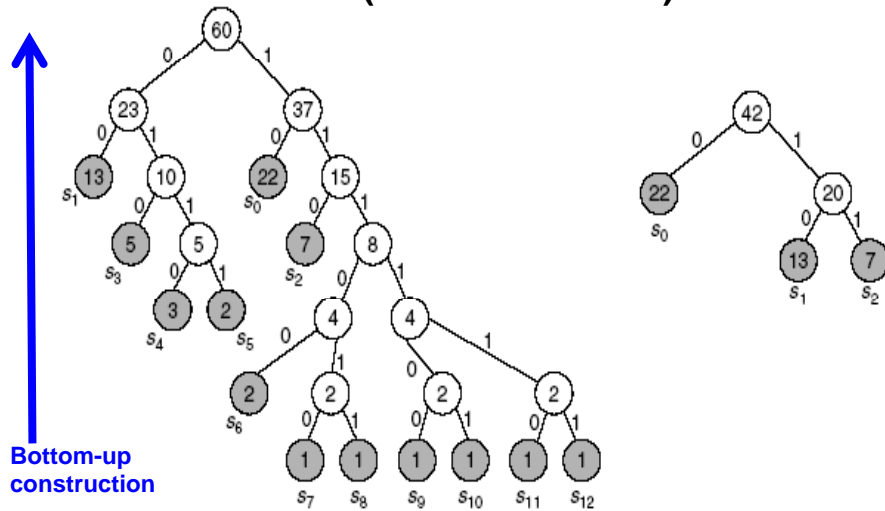
A test vector is partitioned into a number of 4-bit patterns

ch8-10

Huffman Tree

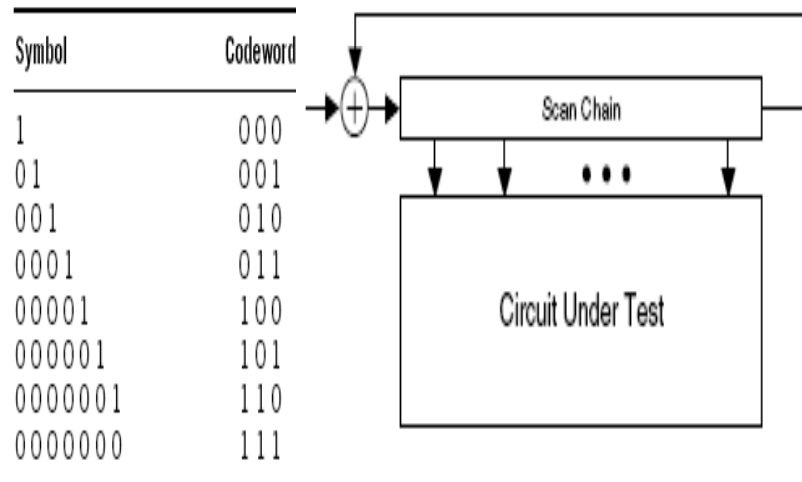
(More Frequent Symbol, Shorter Code)

□ Huffman code (fixed-to-variable)



Run-Length Code

□ Run-length code (variable-to-fixed)



Golomb Code

□ Golomb code (variable-to-variable)

Group	Run-Length	Group Prefix	Tail	Codeword
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
...

ch8-13

Example of Golomb Code

□ Golomb code (variable-to-variable)

$T_D = 001\ 00001\ 0001\ 00001\ 00001\ 0000\ 01\ 001\ 00000001\ 00\ 01$
 $\underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}}$
 $l_1=2\ l_2=4\ l_3=3\ l_4=4\ l_5=4\ l_6=5\ l_7=2\ l_8=7\ l_9=3$

Using Golomb code shown in Table 6.4

$T_E = 010\ 1000\ 011\ 1000\ 1000\ 1001\ 010\ 1011\ 011$

The length of T_D is 43 bits

The length of T_E is 32 bits

ch8-14

Test Stimulus Compression

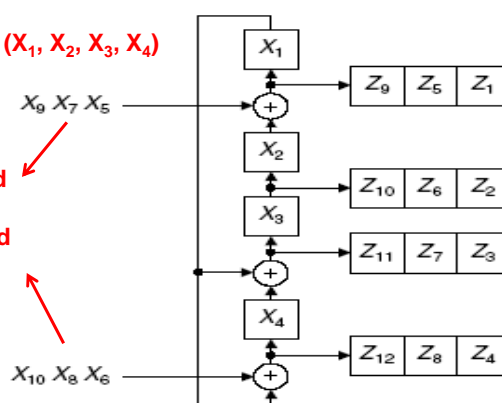
- ❑ Code-based schemes
- ➡ ❑ Linear-decompression-based schemes
- ❑ Broadcast-scan-based schemes

ch8-15

Linear-Decompression-Based Schemes

Seed of LFSR: (X_1, X_2, X_3, X_4)

Compressed
Test vector
To be applied
From ATE



$$\begin{aligned} Z_9 &= X_1 \oplus X_4 \oplus X_9 \\ Z_{10} &= X_1 \oplus X_2 \oplus X_5 \oplus X_6 \\ Z_{11} &= X_2 \oplus X_3 \oplus X_5 \oplus X_7 \oplus X_8 \\ Z_{12} &= X_3 \oplus X_7 \oplus X_{10} \end{aligned}$$

$$\begin{aligned} Z_5 &= X_3 \oplus X_7 \\ Z_6 &= X_1 \oplus X_4 \\ Z_7 &= X_1 \oplus X_2 \oplus X_5 \oplus X_6 \\ Z_8 &= X_2 \oplus X_5 \oplus X_8 \end{aligned}$$

$$\begin{aligned} Z_1 &= X_2 \oplus X_5 \\ Z_2 &= X_3 \\ Z_3 &= X_1 \oplus X_4 \\ Z_4 &= X_1 \oplus X_6 \end{aligned}$$

ch8-16

Matrix Form (Linear-Decompression-Based Schemes)

$$\begin{array}{c} \text{Decompressor} \\ \text{Matrix (?)} \end{array}
 \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}
 \begin{array}{c} \text{Compressed Test Vector + Seed (?)} \\ \uparrow \\ \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \end{bmatrix} \end{array}
 =
 \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \\ Z_7 \\ Z_8 \\ Z_9 \\ Z_{10} \\ Z_{11} \\ Z_{12} \end{bmatrix}
 \begin{array}{c} \text{Original} \\ \text{Test} \\ \text{Vector} \\ \text{(Given)} \end{array}$$

ch8-17

Solving Linear Decompressor & Its Seed

$$\begin{array}{c} Z = 1-011-000000 \\ \text{(Z is Test Vector)} \end{array}
 \begin{array}{c} X = 0111000001 \end{array}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}
 \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}
 \xrightarrow{\text{Gaussian Elimination}}
 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

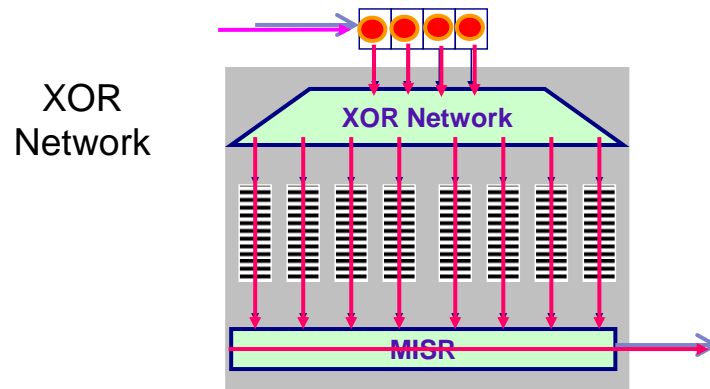
(Z Vector with only care bits) Pivot elements indicated in circles

$$\begin{array}{c} Z = 1-0-1-000000 \\ \text{(Z is Test Vector)} \end{array}
 \begin{array}{c} X = \text{No Solution} \end{array}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
 \xrightarrow{\text{Gaussian Elimination}}
 \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

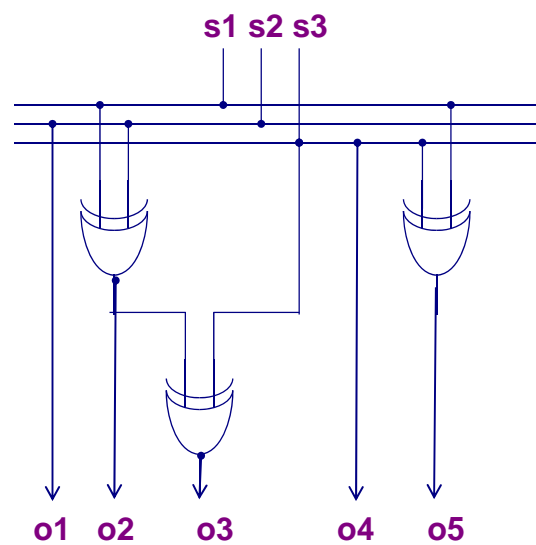
ch8-18

Hardware for Linear-Decompressor



ch8-19

XOR Network: a 3-to-5 Example



ch8-20

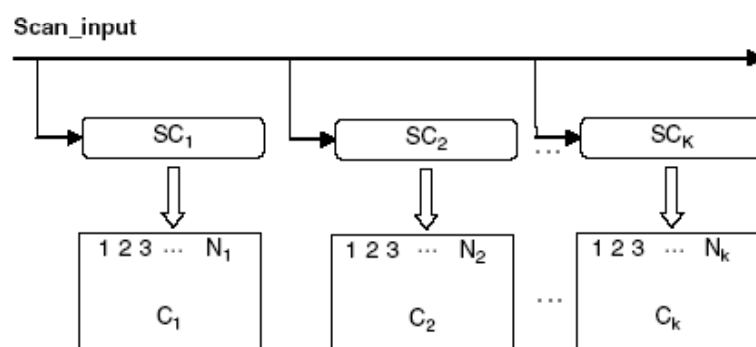
Test Stimulus Compression

- ❑ Code-based schemes
- ❑ Linear-decompression-based schemes
- ➡ ❑ Broadcast-scan-based schemes

ch8-21

Basic Concept: Broadcast-Scan

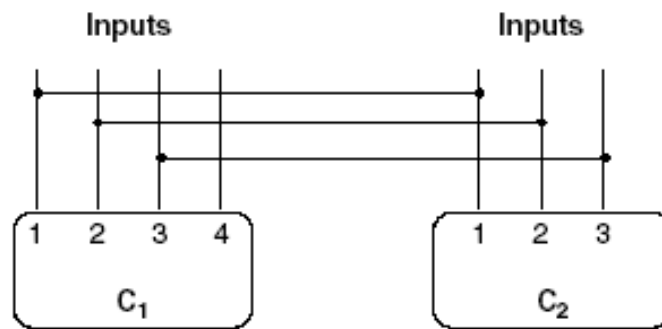
$\{SC_1, SC_2, \dots, SC_k\}$ shares the same test patterns applied by ATE



ch8-22

ATPG Supporting Broadcast-Scan

- Force ATPG tool to generate patterns for broadcast scan (by binding certain PI's together)



ch8-23

Reconfigurable Broadcast Scan

- **Reconfigurable broadcast scan**
 - **Static reconfiguration**
 - The reconfiguration can only be done when a new pattern is to be applied
 - **Dynamic reconfiguration**
 - The configuration can be changed while scanning in a pattern

ch8-24

Broadcast-Scan Based Scheme

- First configuration is: 1-→{2,3,6}, 2-→{7}, 3-→{5,8}, 4-→{1,4}
- Second configuration is: 1-→{1,6}, 2-→{2,4}, 3-→{3,5,7,8}

Scan Chain 1	1	X	1	X	X	X	0	0	X	X
Scan Chain 2	X	X	0	X	1	0	X	1	X	1
Scan Chain 3	X	X	X	X	1	1	1	X	X	1
Scan Chain 4	1	1	X	X	0	0	0	X	0	1
Scan Chain 5	0	X	1	X	X	X	X	X	X	X
Scan Chain 6	X	0	X	1	X	0	X	0	0	X
Scan Chain 7	0	X	0	X	X	1	1	X	X	X
Scan Chain 8	X	X	1	X	X	X	X	1	X	X

First Partition

Second Partition

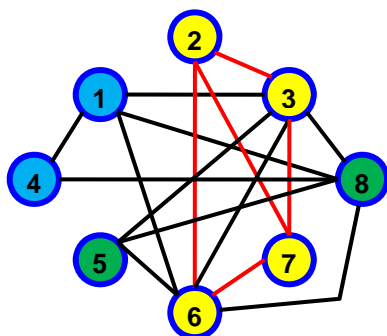
ch8-25

Compatibility Graph – Finding Cliques

Original Test Pattern

Scan Chain 1	1	X	1	X	X
Scan Chain 2	X	X	0	X	1
Scan Chain 3	X	X	X	X	1
Scan Chain 4	1	1	X	X	0
Scan Chain 5	0	X	1	X	X
Scan Chain 6	X	0	X	1	X
Scan Chain 7	0	X	0	X	X
Scan Chain 8	X	X	1	X	X

First Partition



Cliques (fully connected sub-graphs):

(1){SC2, SC3, SC6, SC7} → (000X1)

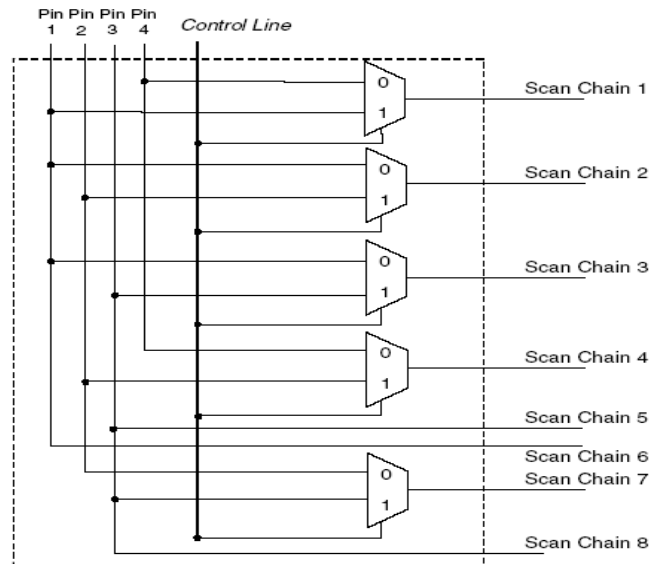
(2){SC5, SC8} → (0X1XX)

(3){SC1, SC4} → (111X0)

→ Overall 8 sub-patterns down to 3

ch8-26

Broadcast-Scan Based Scheme



ch8-27

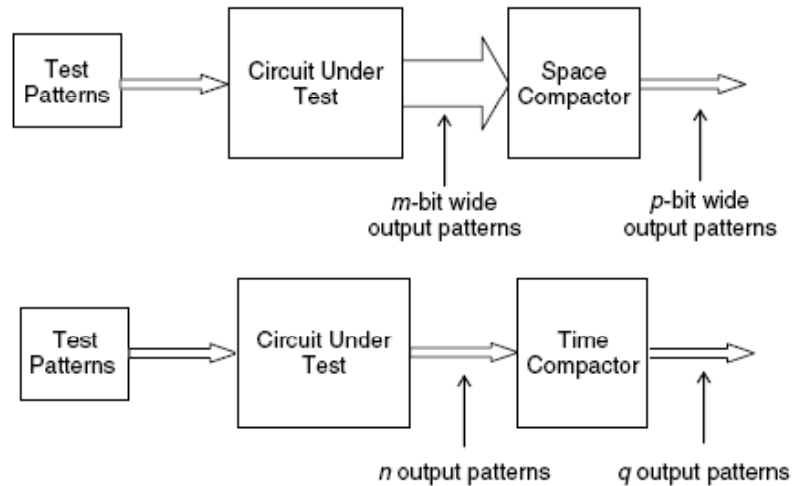
Test Response Compaction (or Called Output Compaction)

- ☐ Space compaction
- ☐ Time compaction
- ☐ Mixed time and space compaction

Unlike lossless input stimulus compression,
Output compaction is often lossy, leading to aliasing...

ch8-28

Test Response Compaction



ch8-29

Space (Output) Compaction

- **Space (output) compaction**
 - Zero-aliasing output compaction
 - X-compactor
 - X-blocking & X-masking techniques
 - X-impact-aware ATPG

ch8-30

Zero-Aliasing Output Compaction

Theorem 6.1

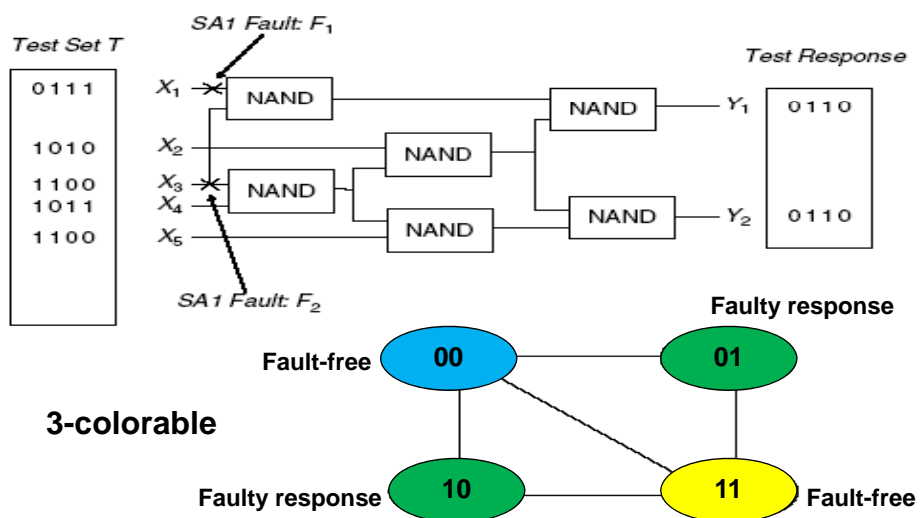
For any test set T , for a circuit that implements function C , there exists a zero-aliasing output space compactor for C with q outputs where $q = \lceil \log_2(|T| + 1) \rceil$.

Theorem 6.2

Let G be a response graph. If G is 2^q colorable, then there exists a q -output zero-aliasing space compactor for the circuit C .

ch8-31

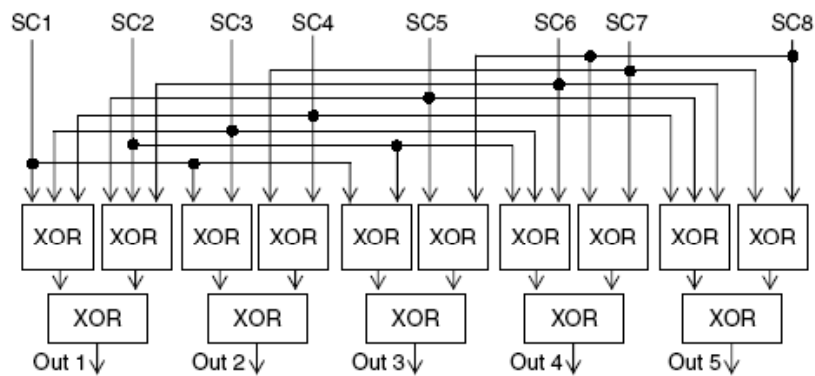
Example: Response Graph



ch8-32

Architecture of X-Compactor

□ X-compactor with 8 inputs and 5 outputs



ch8-33

X-compact Matrix

	Out1	Out2	Out3	Out4	Out5	
$M =$	1	1	1	0	0	SC1 → SC1 drives {Out1, Out2, Out3}
	1	0	1	1	0	SC2
	1	1	0	1	0	SC3
	1	1	0	0	1	SC4
	1	0	1	0	1	SC5
	1	0	0	1	1	SC6
	0	1	0	1	1	SC7
	0	0	1	1	1	SC8

Matrix Form:

$$SC_{1 \times 8} \cdot M_{8 \times 5} = \begin{bmatrix} Out1 \\ Out2 \\ Out3 \\ Out4 \\ Out5 \end{bmatrix} = Out_{5 \times 1}$$

$SC = [SC1 \ SC2 \ SC3 \ SC4 \ SC5 \ SC6 \ SC7 \ SC8]$

ch8-34

X-Blocking or Masking Techniques

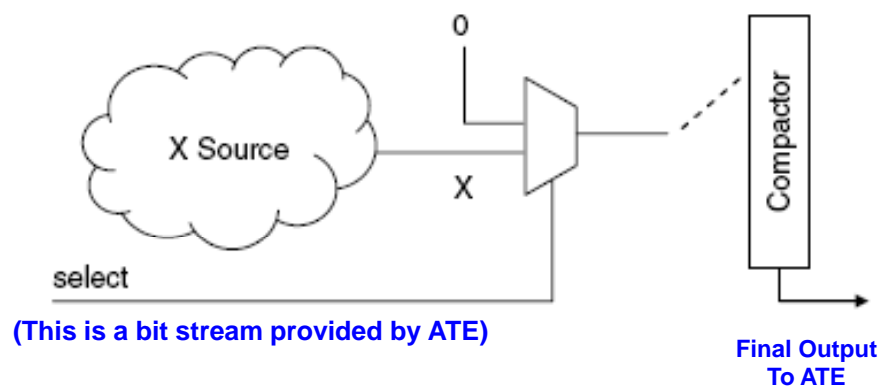
□ X-blocking (or X-bounding, X-avoiding)

- X's can be blocked before reaching the response compactor
- To ensure that no X's will be observed
- May still have fault coverage loss
- Add area overhead and may impact delay

ch8-35

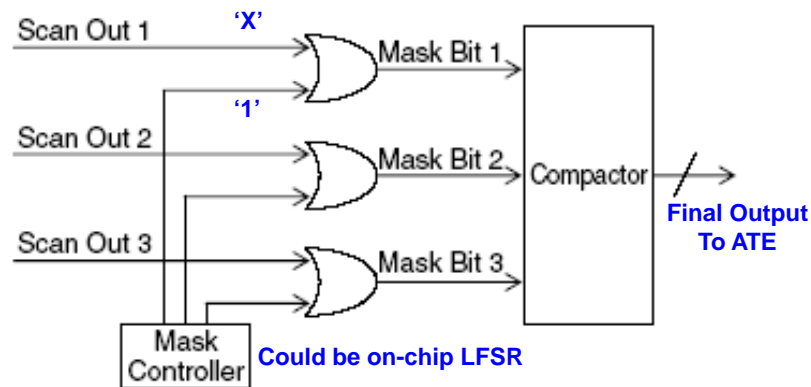
X-Blocking by Selection

- Illustration of the X-blocking scheme



ch8-36

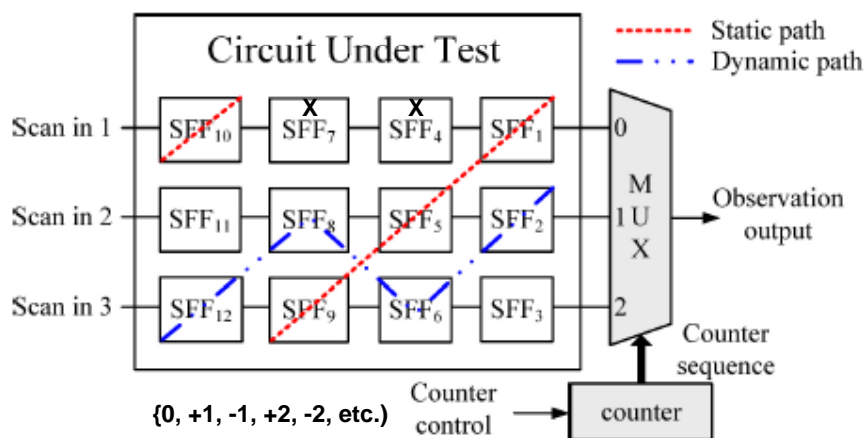
X-Masking by Masking Logic



When there is an X in a scan chain output, a controlling value, i.e., '1' in this example, is issued to mask it out

ch8-37

X-Tolerance by Counter-Based Output Selection



Dynamic path means counter operation can be changed at any scan cycle

ch8-38

X-Impact-Aware ATPG

□ Concept

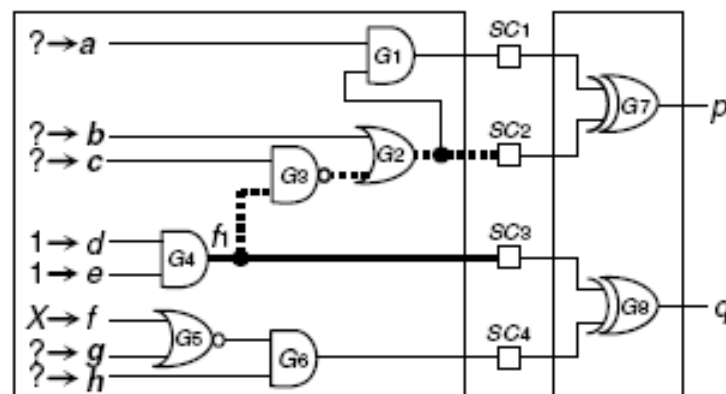
- Simply use ATPG to algorithmically handle the impact of residual X's on the space compactor
- Without adding any extra circuitry

ch8-39

Example: Handling X in ATPG

Path (G5→G6→SC4→G8→q) might be contaminated by 'X' at f

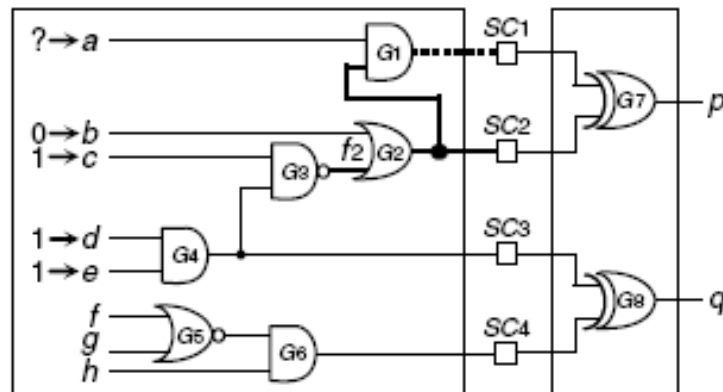
- (1) Propagate the fault effect through (f1→G3→G2→SC2→G7→p) → b=0, c=1
- (2) Kill the X by assigning g to '1' → SC4=0 → q is observable



ch8-40

Output-Compactor-Aware ATPG

- $f_2/1$ fault could be masked as propagated to p
- Block aliasing by assigning a to '0'



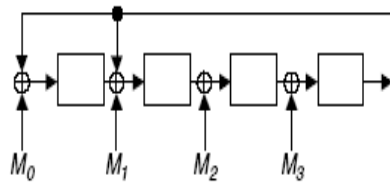
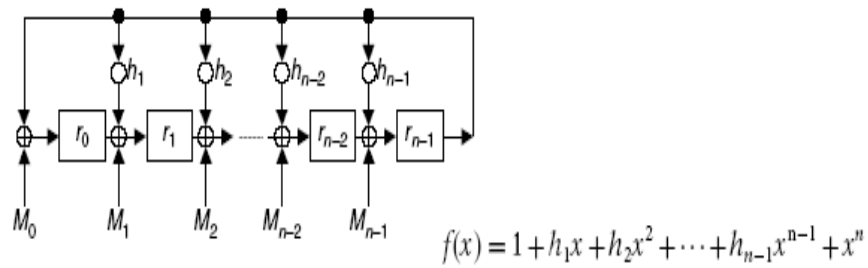
ch8-41

Time Compaction

- Time compaction
 - A time compactor uses sequential logic to compact test responses
 - MISR is most widely adopted
 - n -stage MISR can be described by specifying a *characteristic polynomial* of degree n

ch8-42

Multiple-Input Signature Register

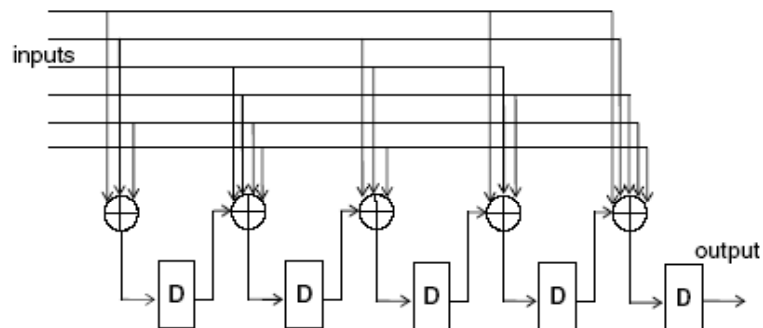


M_0	1 0 0 1 0
M_1	0 1 0 1 0
M_2	1 1 0 0 0
M_3	1 0 0 1 1
M	1 0 0 1 1 0 1 1

ch8-43

Mixed Time & Space Compaction

□ Mixed time and space compaction



ch8-44

Industry Practices

- ❑ OPMISR+
- ❑ Embedded Deterministic Test
- ❑ Virtual Scan and UltraScan
- ❑ Adaptive Scan
- ❑ ETCompression

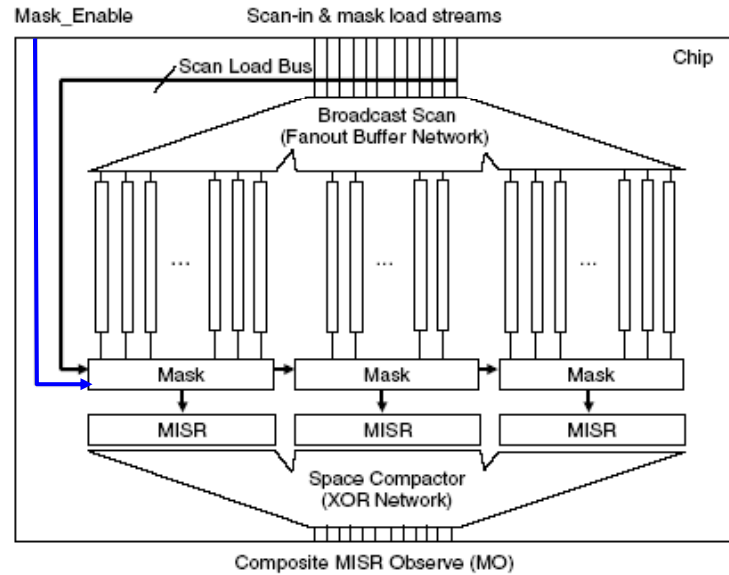
ch8-45

Industry Solutions Categories

- ❑ **Linear-decompression-based schemes**
 - Two steps
 - ETCompression, LogicVision
 - TestKompress, Mentor Graphics
 - SOCBIST, Synopsys
- ❑ **Broadcast-scan-based schemes**
 - Single step
 - SPMISR+, Cadence
 - VirtualScan and UltraScan, SynTest
 - DFT MAX, Synopsys

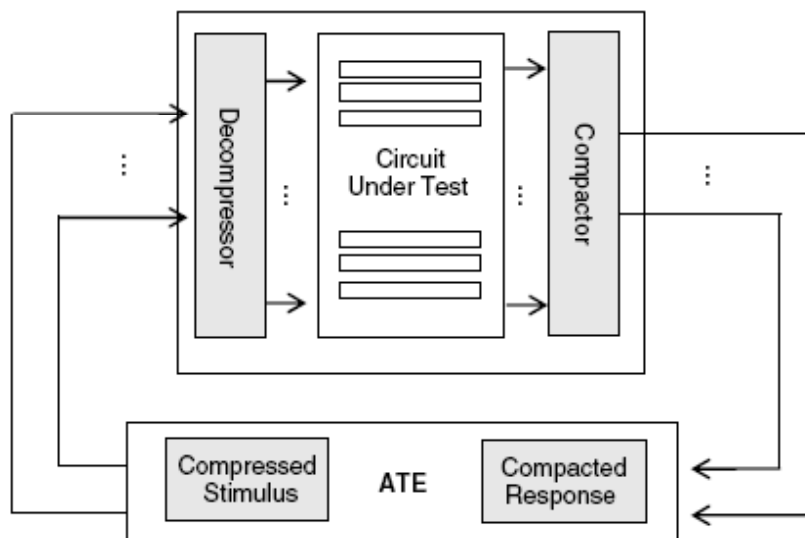
ch8-46

General Scan Architecture for OPMISR+



ch8-47

EDT (TestKompression) Architecture



ch8-48

Concluding Remarks

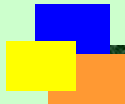
■ **Test compression is**

- **An effective method for reducing test data volume and test application time with relatively small cost**
- **An effective test structure for embedded hard cores**
- **Easy to implement and capable of producing high-quality tests**
- **Successful as part of standard design flow**

ch8-49

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



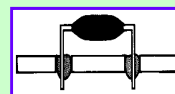
Chapter 9
Boundary Scan

Objectives

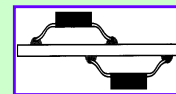
• Standards for board level testing

• Used for

- Chips
- Chip interconnections
- Modules
- Modules interconnections
- Subsystems
- Systems
- Multi-chip modules
 - Die-to-board integration →

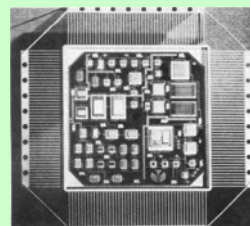


Through-hole
mounting



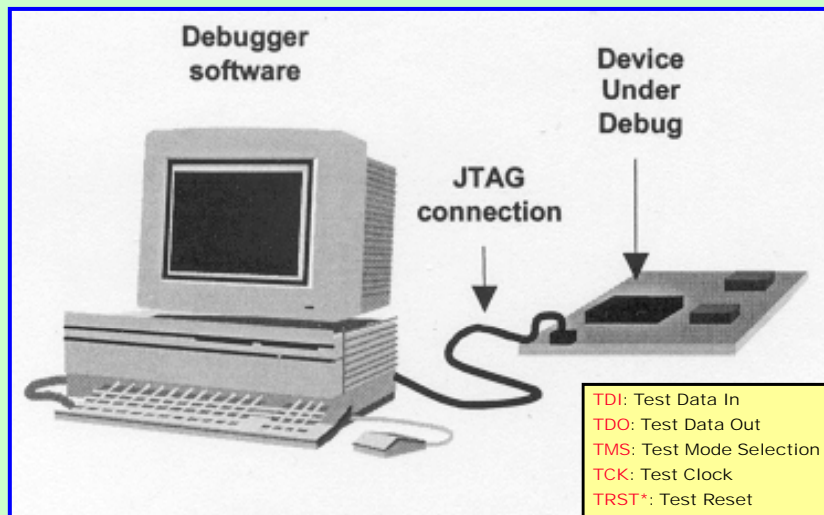
Surface
mount

53 ICs + 40 discrete devices



ch9-2

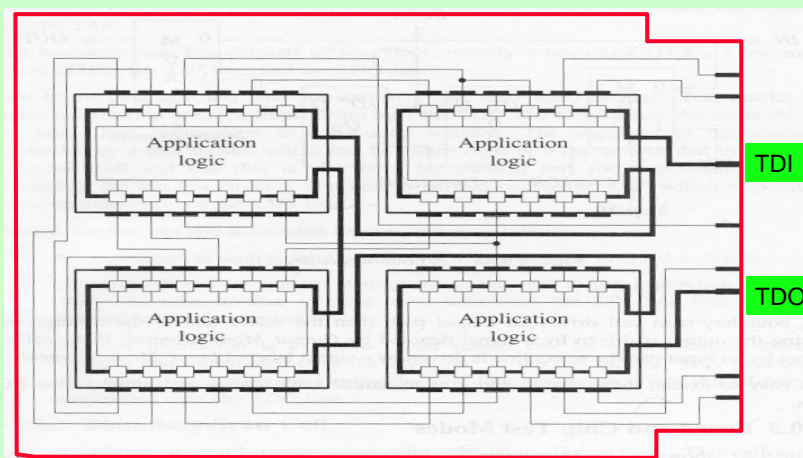
Board Testing Setup



ch9-3

A Printed Circuit Board with Boundary Scan

Boundary scan use **4 or 5 wire bus** to provide **accessibility** to the I/O pins of selected on-board IC → thereby facilitating **board-level testing**



ch9-4

History

- **1985**
 - Joint **European** Test Action Group (JETAG, Philips)
- **1986**
 - VHSIC Element-Test & Maintenance (ETM) bus standard (IBM et al.)
 - VHSIC Test & Maintenance TM Bus Structure (IBM et al.)
- **1988**
 - **Joint Test Action Group** (JTAG) proposed Boundary Scan Standard
- **1990**
 - Boundary Scan approved as IEEE Std. 1149.1-1990
 - **Boundary Scan Description Language** (BSDL) proposed by HP
- **1993**
 - 1149.1a –1993 approved to replace 1149.1-1990
- **1994**
 - 1149.1b BSDL approved
- **1995**
 - 1149.5 approved

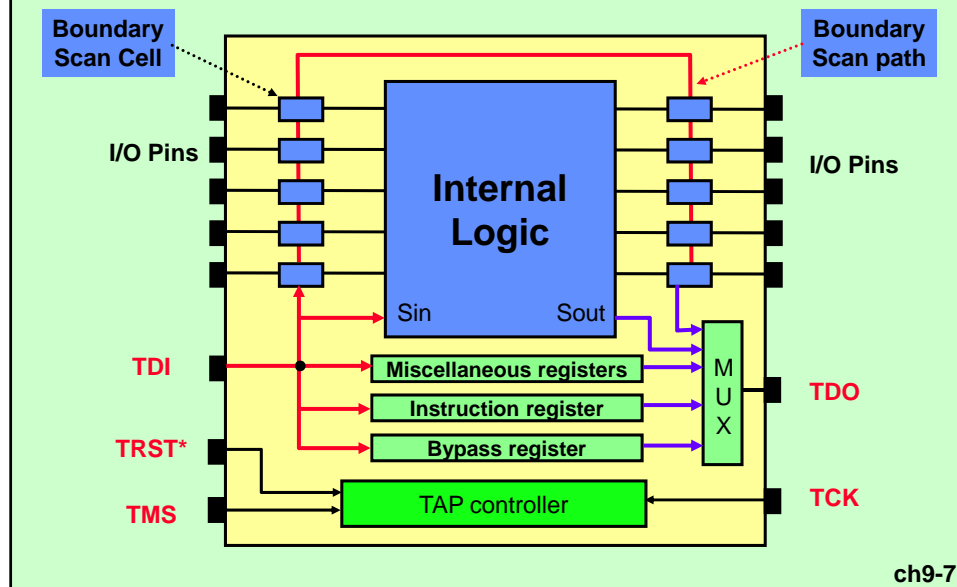
ch9-5

Overview of P1149 Family

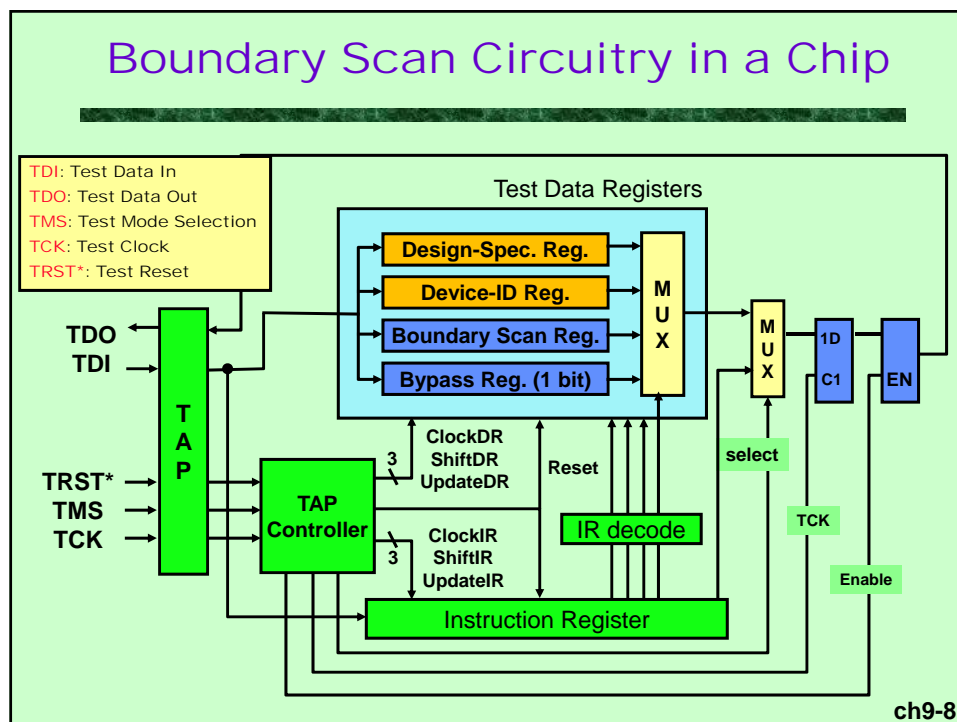
Number	Title	Status
1149.1	Testing of digital chips and Interconnections between Chips	Std. 1149.1-1990 Std. 1149.1a-1993 Std. 1149.1b-1994 (BSDL)
1149.2	Extended Digital Serial Interface	Near Completion
1149.3	Direct Access Testability Interface	Discontinue
1149.4	Mixed-Signal Test Bus	Started Nov. 1991
1149.5	Standard Module Test and Maintenance (MTM) Bus Protocol	Std. 1149.5-1995
1149	Unification	Not yet started

ch9-6

Basic Chip Architecture of 1149.1



Boundary Scan Circuitry in a Chip

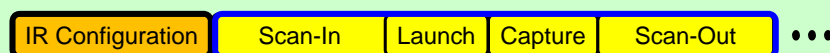


Hardware Components of 1149.1

- TAP (Test Access Port)
 - TMS, TCK, TDI, TDO, TRST* (optional)
- TAP Controller
 - A finite state machine with 16 states
 - Input: TCK, TMS
 - Output: 9 or 10 signals including ClockDR, UpdateDR, shiftDR, ClockIR, UpdateIR, ShiftIR, Select, Enable, TCK, and the optional TRST*
- IR (Instruction Register)
- TDR (Test Data Register)
 - Mandatory: boundary scan register and bypass register
 - Optional: device-ID register, design-specific registers, etc.

ch9-9

Bus Protocol



Serially send instruction over TDI into **instruction register**

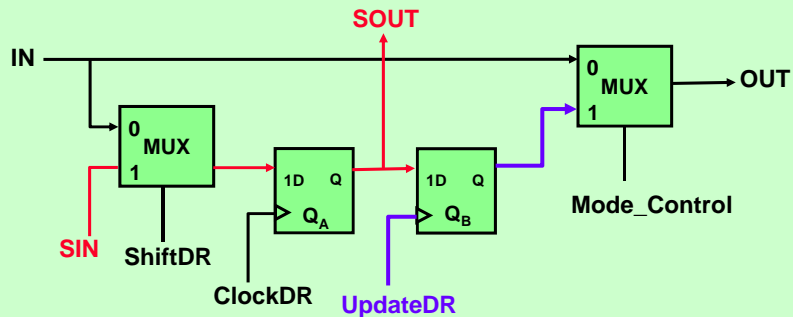
Test circuitry is configured
To respond to instruction
(Scan in data through TDI)

Execute test instruction

Shift out test results through TDO
New test data on TDI can be shifted in simultaneously

ch9-10

A Typical Boundary Scan Cell

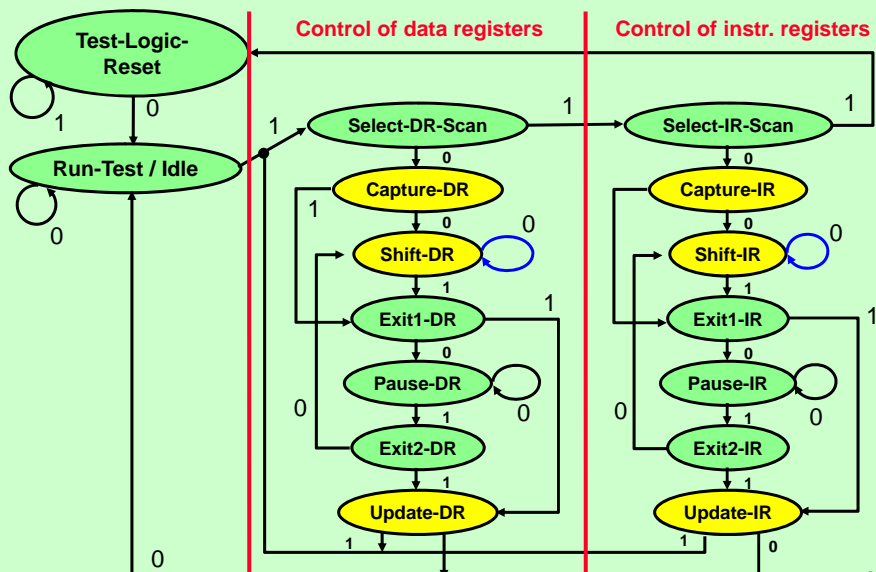


• Operation Modes

- Normal: Mode_control=0; IN→OUT
- Scan: ShiftDR=1, ClockDR; TDI→...→SIN→SOUT→...→TDO
- 捕 - Capture: ShiftDR=0, ClockDR; IN→Q_A, OUT driven by IN or Q_B
- 投 - Update: Mode_Control=1, UpdateDR; Q_B→OUT

ch9-11

State Diagram of TAP Controller



ch9-12

States of TAP Controller

- Test-Logic-Reset: normal mode
- Run-Test/Idle: wait for internal test such as BIST
- Select-DR-Scan: initiate a data-scan sequence
- Capture-DR: load test data in parallel
- Shift-DR: load test data in series
- Exit1-DR: Finish phase-1 shifting of data
- Pause-DR: Temporarily hold the scan operation
(allow the bus master to reload data)
- Exit2-DR: finish phase-2 shifting of data
- Update-DR: parallel load from associated shift registers

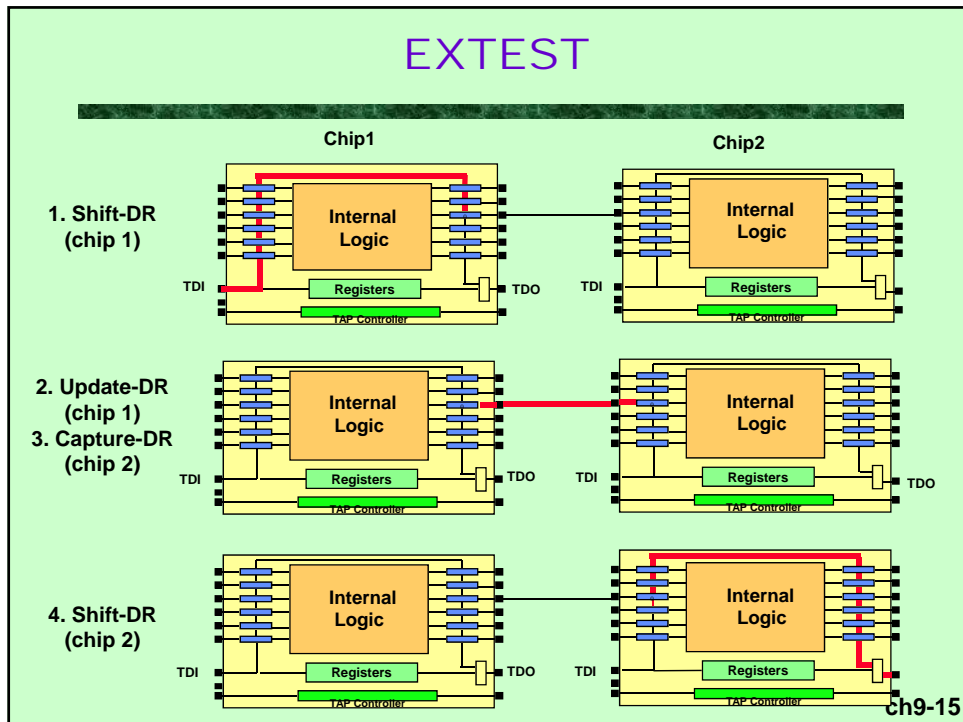
ch9-13

Instruction Set

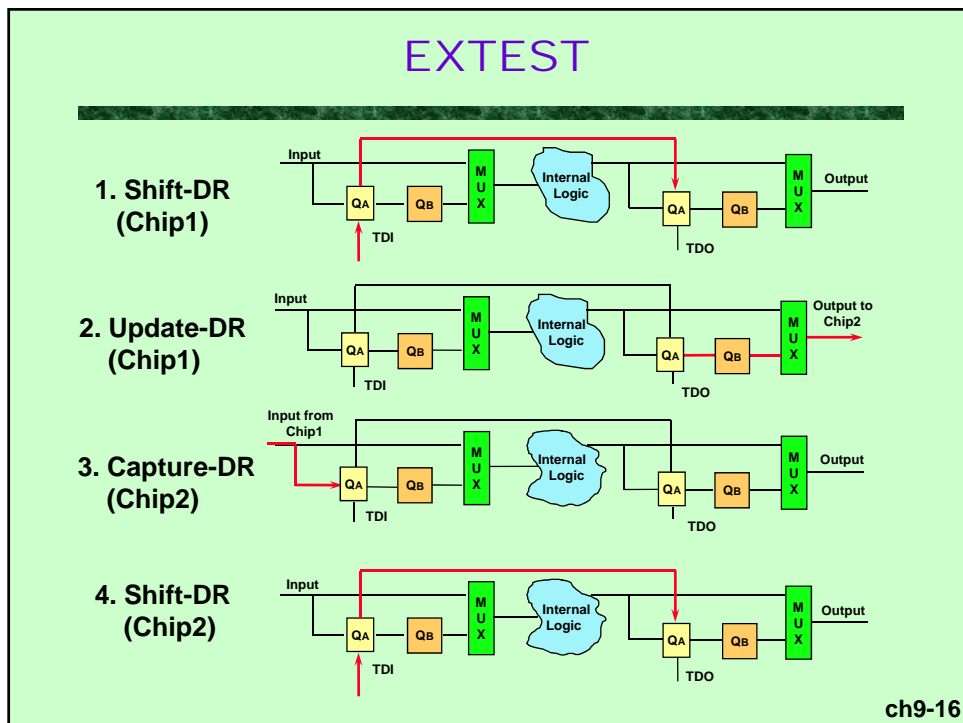
- EXTEST
 - Test Interconnection between chips and board
- SAMPLE/PRELOAD
 - Sample and shift out data or shift data only
- BYPASS
 - Bypass data through a chip
- Optional
 - Intest, RunBist, CLAMP, Idcode, usercode, High-Z, etc.

ch9-14

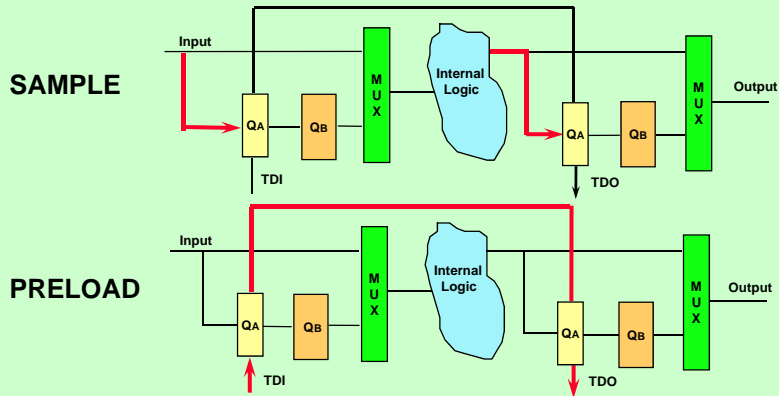
EXTEST



EXTEST



SAMPLE/PRELOAD

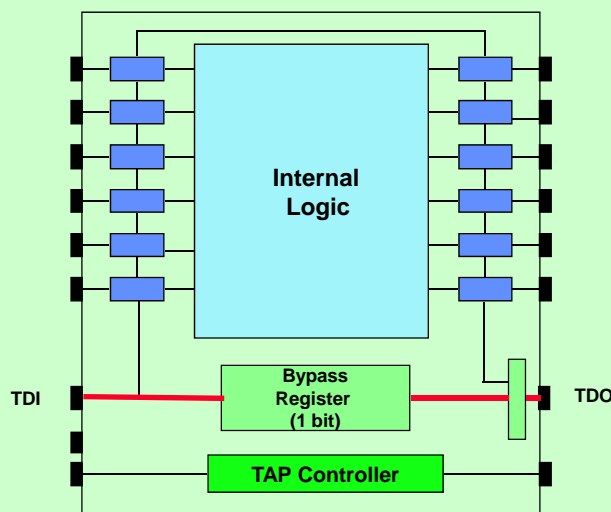


Sample/Preload is one instruction that allows

1. Sample and shift (out) or
2. Shift (in) only

ch9-17

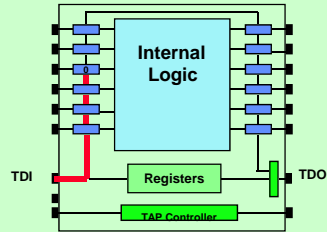
BYPASS



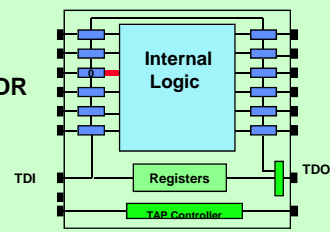
ch9-18

INTEST

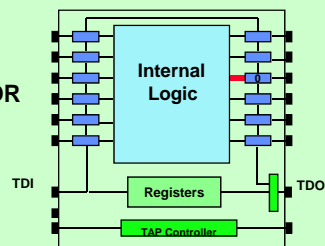
1. Shift-DR



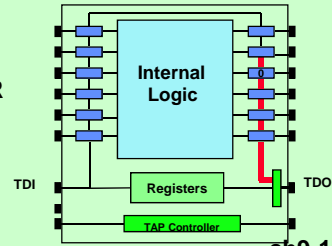
2. Update-DR



3. Capture-DR



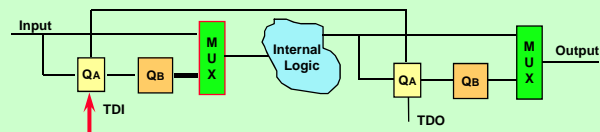
4. Shift-DR



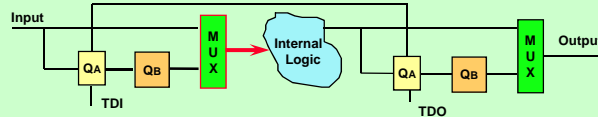
ch9-19

INTEST

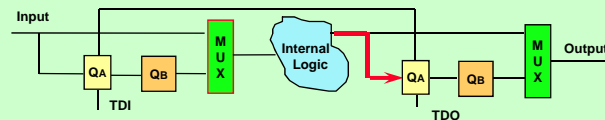
1. Shift-DR



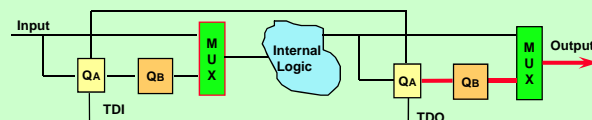
2. Update-DR



3. Capture-DR

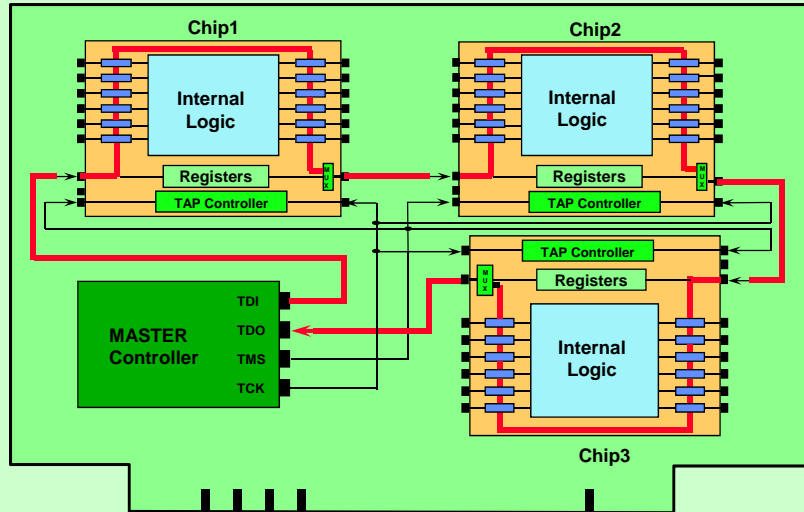


4. Shift-DR



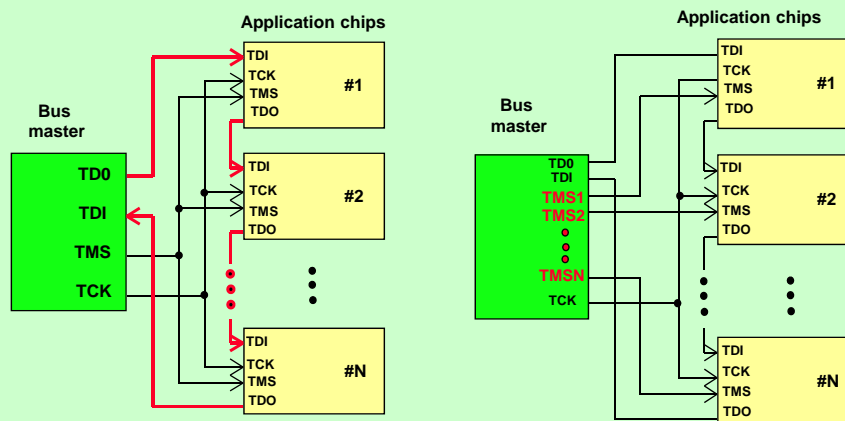
ch9-20

A Printed Circuit Board With 1149.1 (Ring configuration, test controller on board)



ch9-21

Test Bus Configuration



Ring configuration

Star configuration

ch9-22



國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 10
High-Speed Interconnect Testing

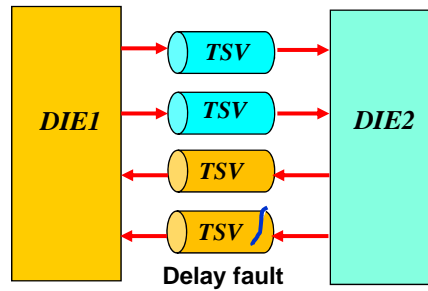
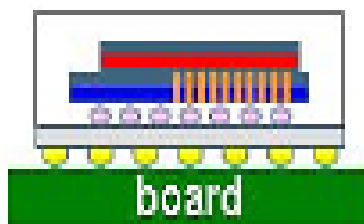
Outline

- 
-
- 
- ◆ **Introduction**
 - ◆ **Problem, Objective, Review, and Motivation**
 - ◆ **Pulse-Vanishing Test (PV-Test)**
 - ◆ **VOT-Based Oscillation Test**

Testing Interconnects in 3D IC

Problem Addressed:

To develop a **low-cost** method to **test the delay fault** associated with the TSV (Through Silicon Via)

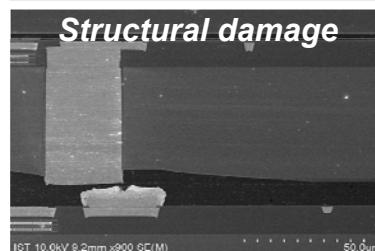
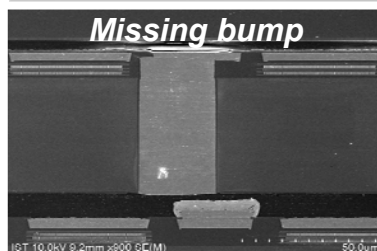
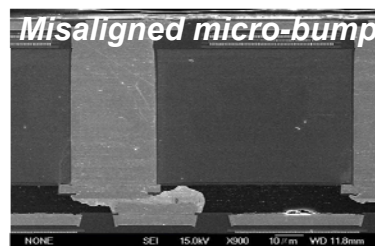
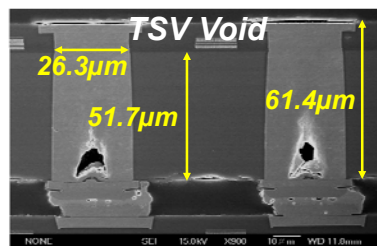


3D-IC using TSVs

3

SEM Photos of TSV Defects (0.18um Through Silicon Stacking at ITRI)

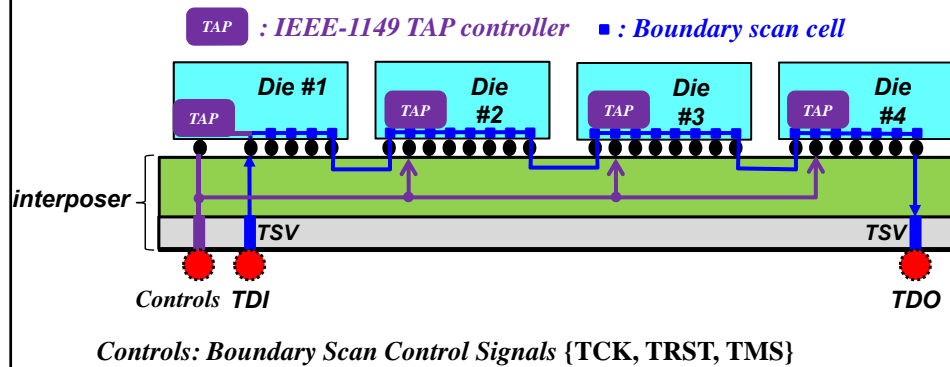
A **partially faulty TSV** may not operate as fast as we expect
(and it could deteriorate over time...)



4

Testing Interconnects in 2.5-D IC

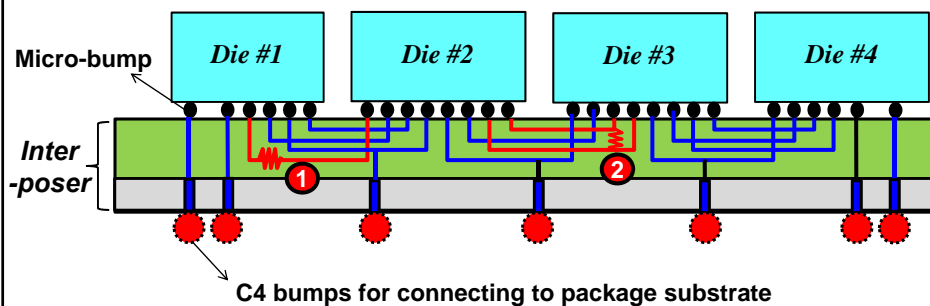
- ◆ For each die, interposer wires are like **Pseudo-I/Os**
- ◆ Boundary scan cells needed for (1) **Die Test**, and (2) **Interconnect Test**



5

Parametric Faults in High-Speed Die-to-Die Interposer Wires

- (1) **Resistive Open Fault** in an interposer wire ①
- (2) **Resistive Bridging Fault** between two interposer wires ②



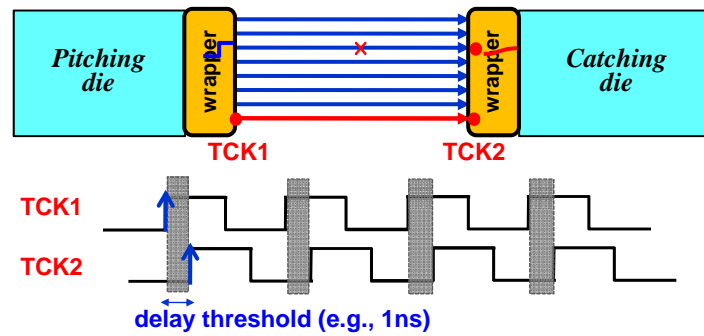
6

Objective and Challenge

Objective: To detect parametric faults (e.g., <1ns delay fault)

→ May need to maintain a **pitcher-catcher timing relationship across dies**
(This type of cross-die clock synchronization may not be easy)

→ There are so other choices...



Note: test clocks TCK1 and TCK2 are low-speed test clocks (e.g., 10MHz)

7

Outline

◆ Introduction



◆ Pulse-Vanishing Test (PV-Test)

- At-speed testing for high-speed interconnects

◆ VOT-Based Oscillation Test

8

Downloaded from <http://ajphaphysocpharm.sagepub.com/> at 11:01 11 November 2014



9

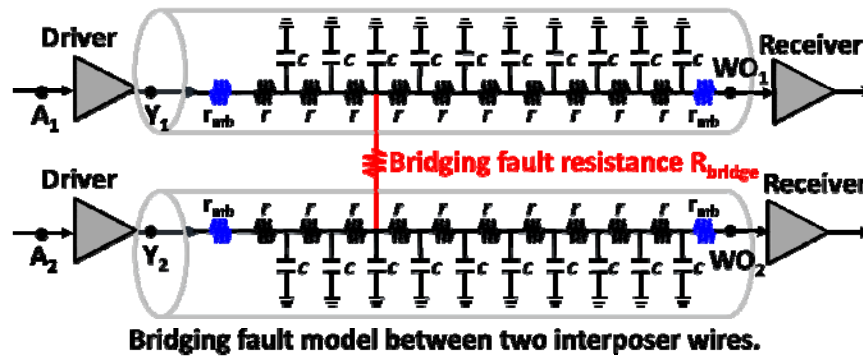
Downloaded from <http://ajph.org/> on November 10, 2014



(b) **Faulty** model of an interconnect with a **resistive open fault**.

10

Resistive Bridging Fault Model



11

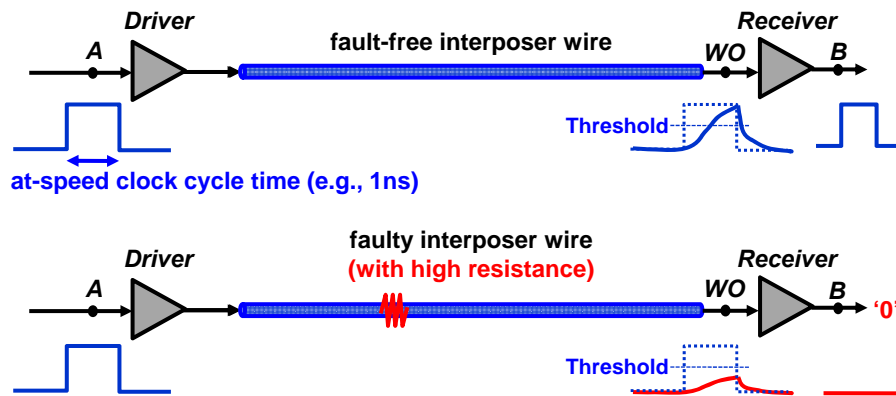
Pulse-Vanishing Test (PV-Test)

Pulse-Vanishing Test:

(1) **Test Stimulus:** A short-duration pulse (0-1-0)

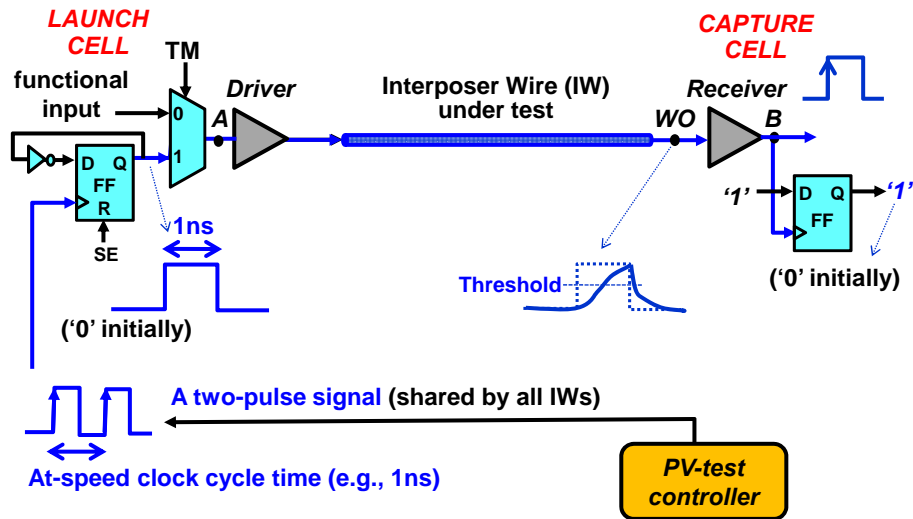
(2) **Fault Detection Criterion:**

If the **pulse vanishes** at the receiver's output, then there is a **delay fault**



12

Primitive DfT Circuit (for Pulse-Vanishing Test)

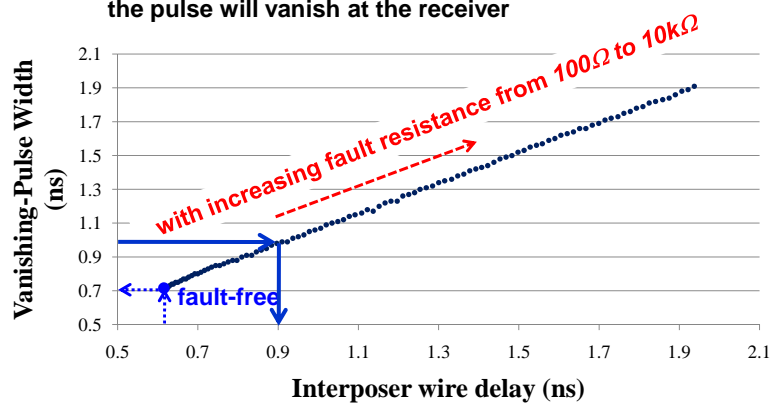


13

Vanishing Pulse Width (for 1000 μm Long Interposer Wire)

Def: **Vanishing Pulse-Width (VPW)**

The pulse-width of the applied test pulse above which the pulse will vanish at the receiver



Comment: A larger **test pulse width** implies larger **delay test threshold**

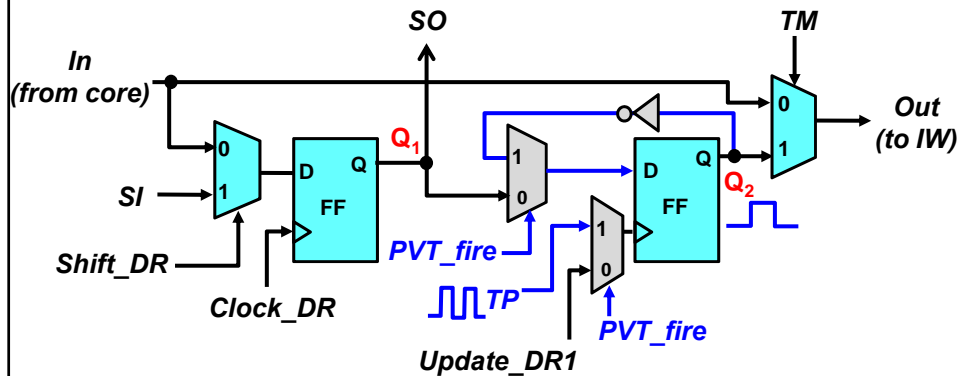
14

Boundary-Scan Compatible *Launch Cell*

When 'PVT_fire' is '1':

(1) 2nd FF behaves like a **toggle-type FF**

(2) Two-Pulse signal 'TP' is applied to the clock port of 2nd FF



Note: Q2 needs to be **initialized to '0'** before a test session

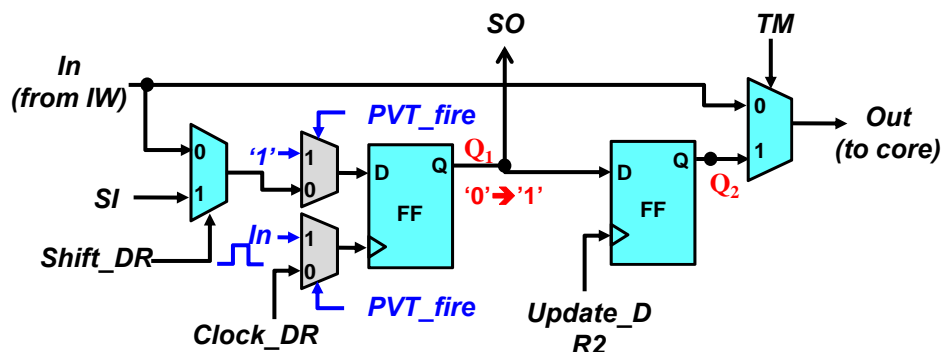
15

Boundary-Scan Compatible *Capture Cell*

When 'PVT_fire' is '1':

(1) 1st FF is set to '1' if receiving a clock pulse, otherwise stays '0'

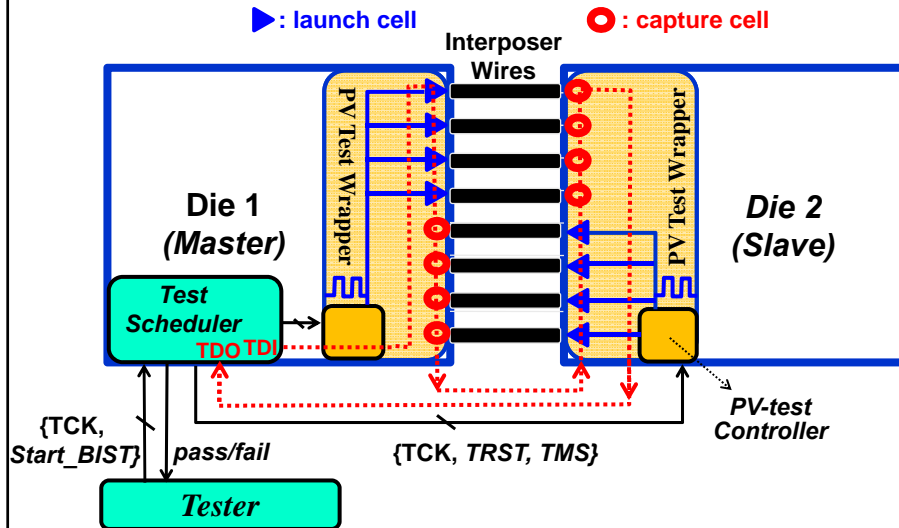
(2) Input signal 'IN' is applied to the clock port of 1st FF



Note: Q1 needs to be **initialized to '0'** before a test session
The **final test result is stored at Q1**

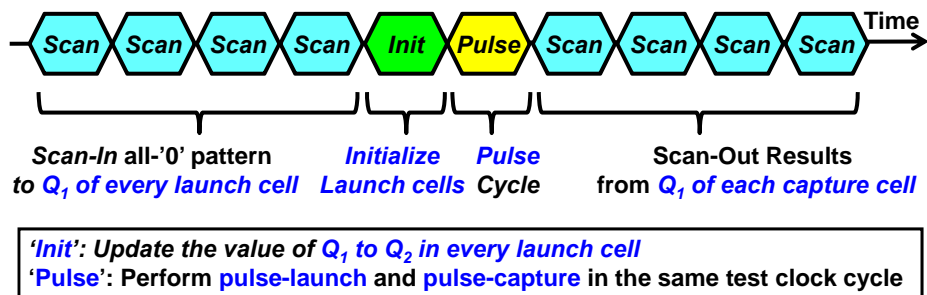
16

Built-In Self-Test Architecture



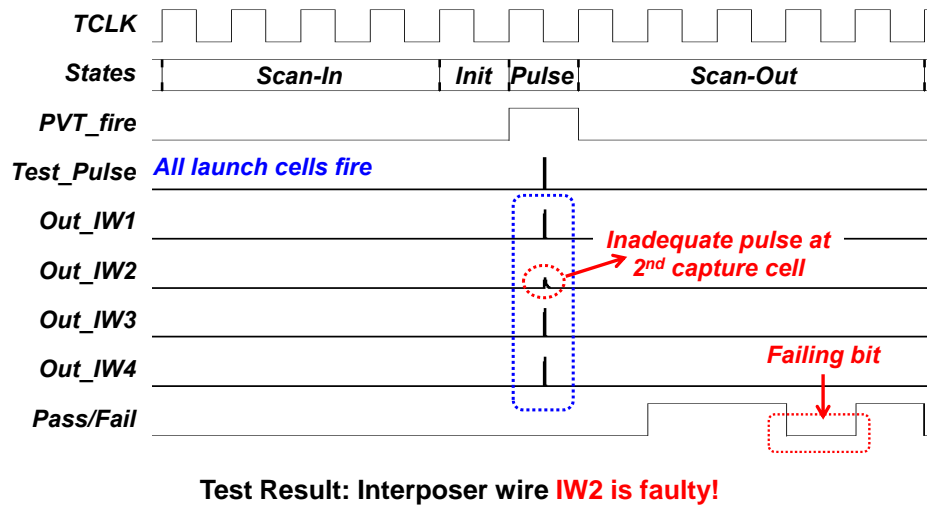
17

PV-Test Procedure (Scan-In, Init, Pulse, Scan-Out)



18

Simulation Waveforms of a PV-Test



19

Test Time

- ◆ A PV-test session using 10MHz test clock is about
 - 0.82 ms for 1024 interposer wires
 - 26.21 ms for 32K (32,768) interposer wires

20

Area Overhead

Estimation is based on a 90nm CMOS process

Area overhead		
Type	Cell Name	Layout Area (μm^2)
Basic Cells	INVERTER	2.82
	2-input NAND Cell	2.94
	MUX Cell	8.47
	FF Cell	17.64
Basic Macros	Boundary Scan Cell	52.22
	Launch Cell	92.56
	Capture Cell	69.16
	PV-test controller	670.3
Overhead Percentage Over 1149.1	55.55% for 1024 interposer wires	
	54.9% for 32,768 interposer wires	

21

Summary of PV-Test

The **interposer** needs to be **tested alone and thoroughly**.
And also, when a 2.5-D IC fails,

We know **if the interposer should be responsible**.

◆ Advantages of Pulse-Vanishing Test

- Simple fault detection scheme (**No post-processing**)
- Delay Test **without die-to-die high-speed clock synchronization**
- **Boundary-Scan-Like** Test Architecture (55.55% overhead)
- **On-the-spot Diagnosis** (good for future self-repair)

22

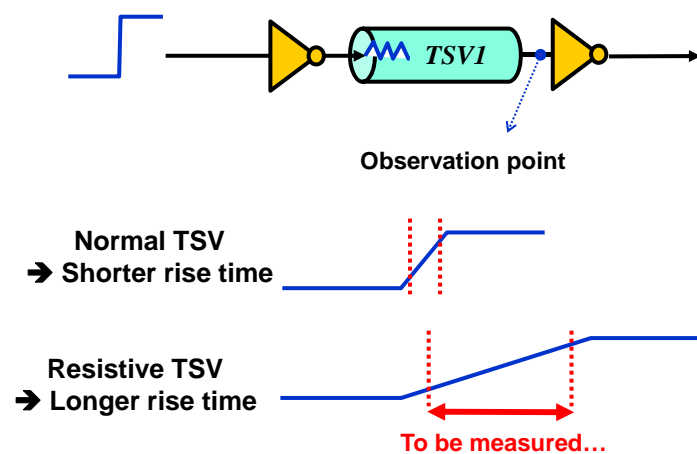
Outline

- ◆ Introduction
- ◆ Pulse-Vanishing Test (PV-Test)
- ➔ ◆ VOT-Based Oscillation Test
 - Characterization-based parametric fault testing

23

Concept 1: *It's a matter of transition time measurement!*

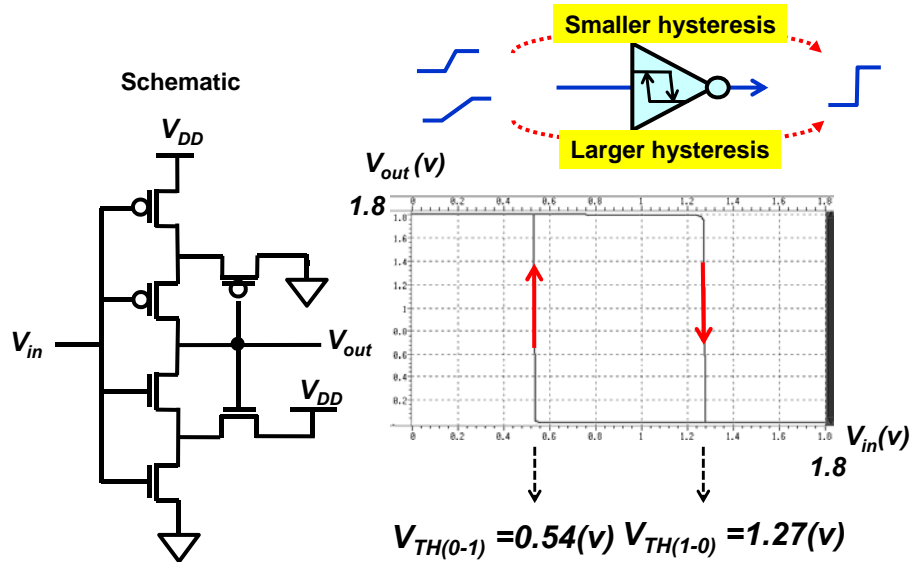
A TSV with delay fault → Longer Rise/Fall Time



24

Concept 2: Use Schmitt-Trigger Inverter

- *Hysteresis proportional to the input Transition time*



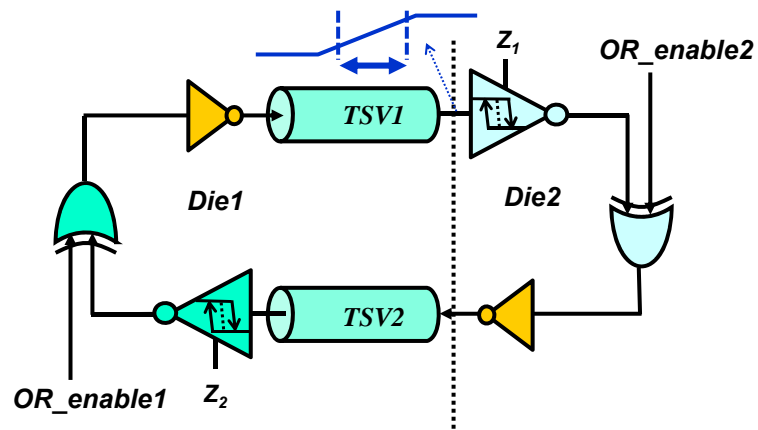
25

Architecture of VOT Scheme (Per TSV Pair)

(VOT: Variable Output Threshold)

Use **Variable-Threshold Output Inverter** for each TSV:

- (1) Control signal $Z = 0 \rightarrow$ **Normal Inverter**
- (2) Control signal $Z = 1 \rightarrow$ **Schmitt-Trigger Inverter (WITH HYSTERESIS)**



26

Brief Summary of our Idea

TSV Delay → Transition Time

Transition Time → Oscillation Period Change

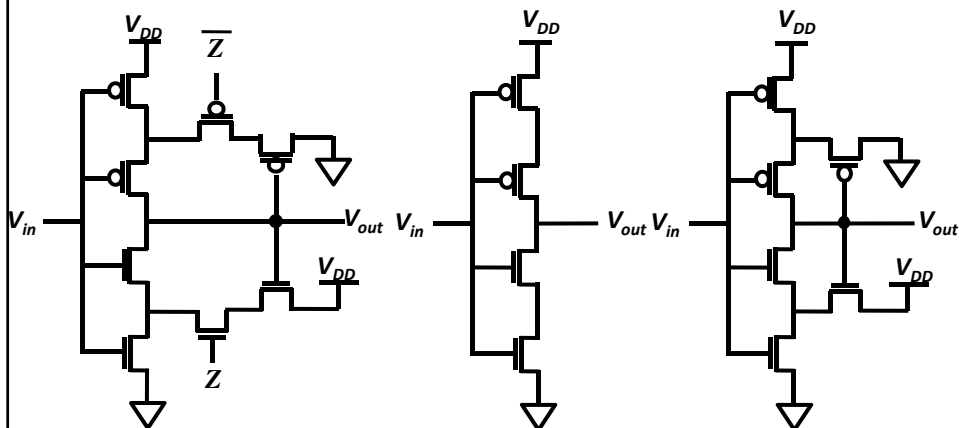
(from normal to Schmitt-Trigger)

(Easily Measurable)

27

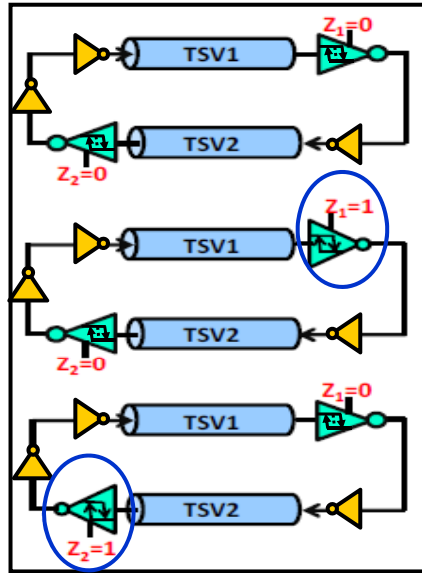
Schematic of a VOT Inverter

(a) overall schematic (b) normal inverter ($Z=0$) (c) ST inverter ($Z=1$)



28

Three Oscillation Periods in VOT-Analysis



(1) Normal mode:
Oscillation period = T_{REF}

(2) TSV1-in-ST mode
Oscillation period = T_{ST1}
TSV1 delay $\sim \Delta T_1 (T_{ST1} - T_{REF})$

(3) TSV2-in-ST mode
Oscillation period = T_{ST2}
TSV2 delay $\sim \Delta T_2 (T_{ST2} - T_{REF})$

29

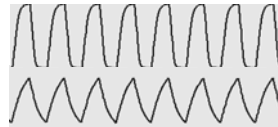
Example: Predict the Delay of Each TSV

$R_{TSV1} = 10 \text{ } (\Omega)$ $C_{TSV1} = 400 \text{ (fF)}$ $R_{TSV2} = 1 \text{ (k}\Omega)$ $C_{TSV2} = 800 \text{ (fF)}$

Waveforms under the normal configuration

$T_{REF} = 4.42 \text{ ns}$

endpoint of TSV1



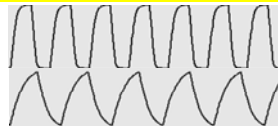
endpoint of TSV2



Smaller
Transition times
Larger
Transition times

Waveforms under the Schmitt-Trigger configuration

endpoint of TSV1



endpoint of TSV2



$T_{ST1} = 5.05 \text{ ns}$

$T_{ST2} = 6.49 \text{ ns}$

Normal Configuration: $T_{REF} = 4.42 \text{ ns}$

TSV1-in-ST Configuration: $T_{ST1} = 5.05 \text{ ns}$ (smaller increase from T_{REF})

TSV2-in-ST Configuration: $T_{ST2} = 6.49 \text{ ns}$ (larger increase from T_{REF})

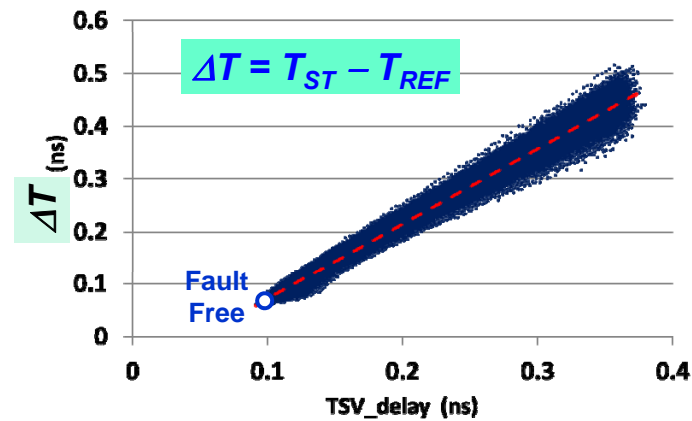
$\Delta T_{ST1} = 5.05 - 4.42 = 0.63 \text{ ns}$

$\Delta T_{ST2} = 6.49 - 4.42 = 2.07 \text{ ns}$

30

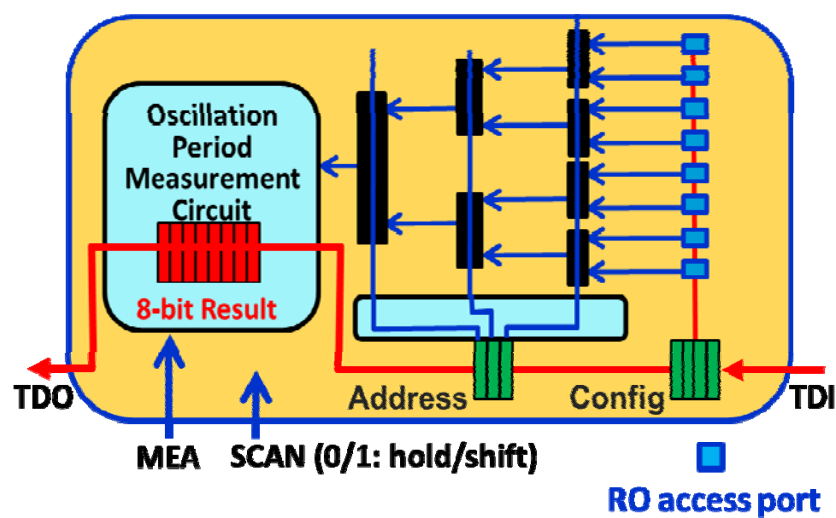
Ex: Correlation between TSV Delay and ΔT

- ◆ Fault Population: Resistive Open Faults
- ◆ An outlier in measurable ΔT is an outlier in TSV delay



31

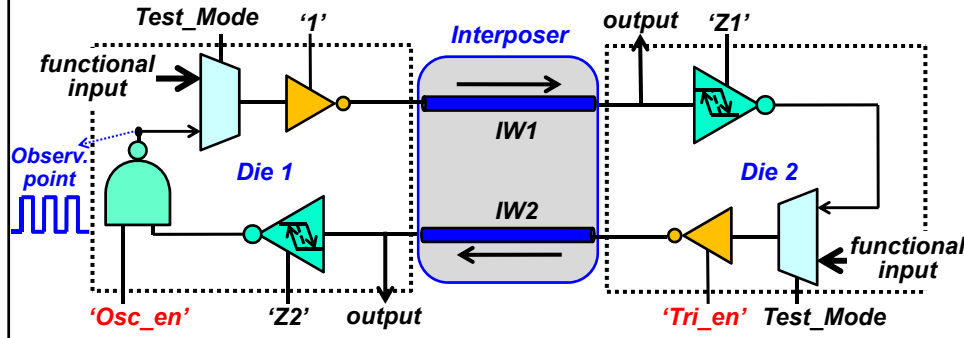
RO, MUX Tree, and Measurement Circuits



32

Ring Oscillator (RO) (for One Pair of Interposer Wires)

Two extra Control Signals (to support **bridging fault detection**):
 (1) '**Osc_en**': enabling signal for oscillation
 (2) '**Tri_en**': tri-state enabling signal for the driver of IW2



33

Three Test Strategies

Principles:

- (1) All ROs oscillate concurrently to detect "**resistive open faults**"
- (2) One RO oscillates at a time to detect "**inter-RO resistive bridging faults**"
- (3) No RO oscillates to detect "**intra-RO resistive bridging faults**"

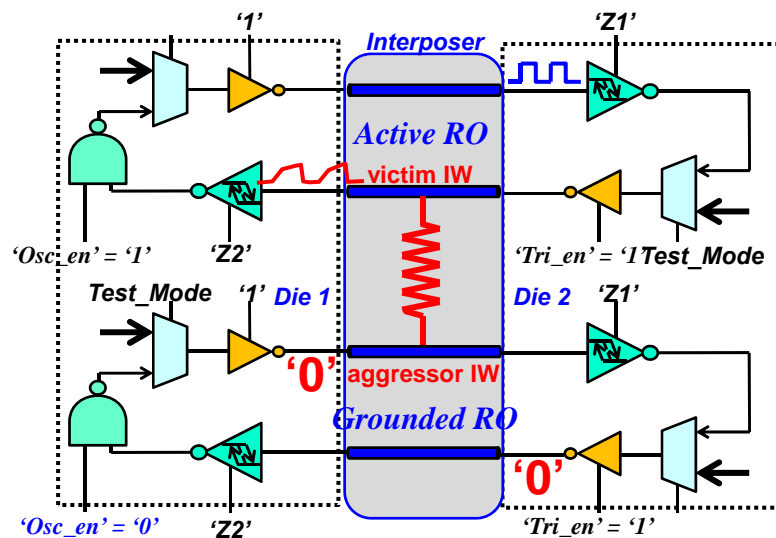
	Test Strategy	RO Settings	Test Actions
Test OPEN	AO-strategy (All Oscillation)	Every RO is Active	Measure $\{T_{REF}, T_{ST1}, T_{ST2}\}$ of every RO in sequence
Test Inter-RO BRIDGING	OO-strategy (One Oscillation)	Target RO is Active (One RO at a time)	Measure $\{T_{REF}, T_{ST1}, T_{ST2}\}$ of the target RO
		The others are Grounded	NA
Test Intra-RO BRIDGING	NO-strategy (No Oscillation)	Every RO is Half-Floating	Measure $\{T_{REF}\}$ of every RO

34



35

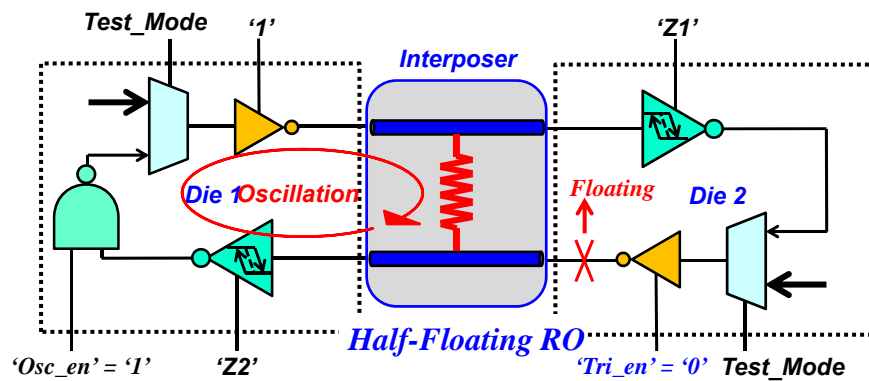
Figure 1



36

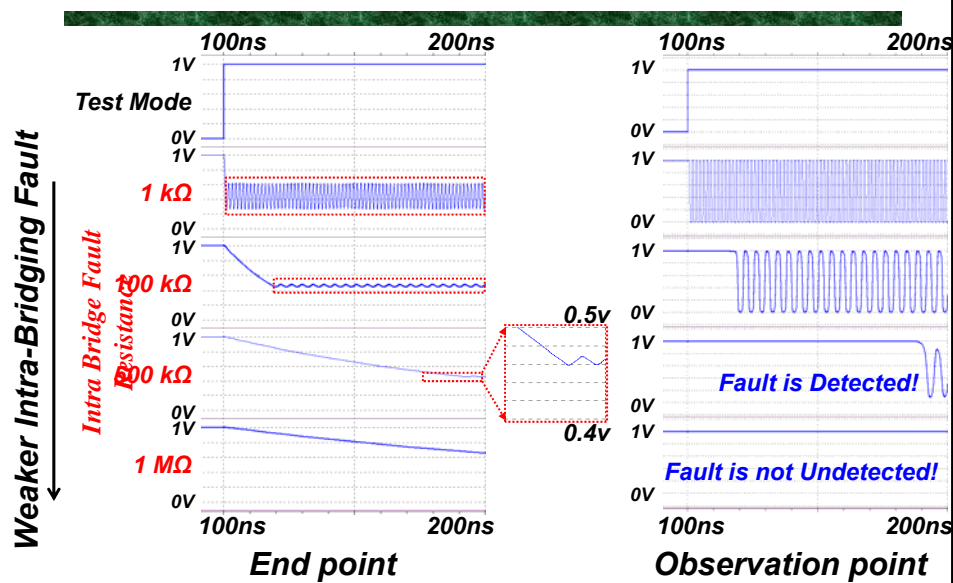
NO-strategy (No-Oscillation) (to detect an intro-RO bridging faults)

The existence of an intro-RO bridging fault will cause a half-floating RO to oscillate abnormally.



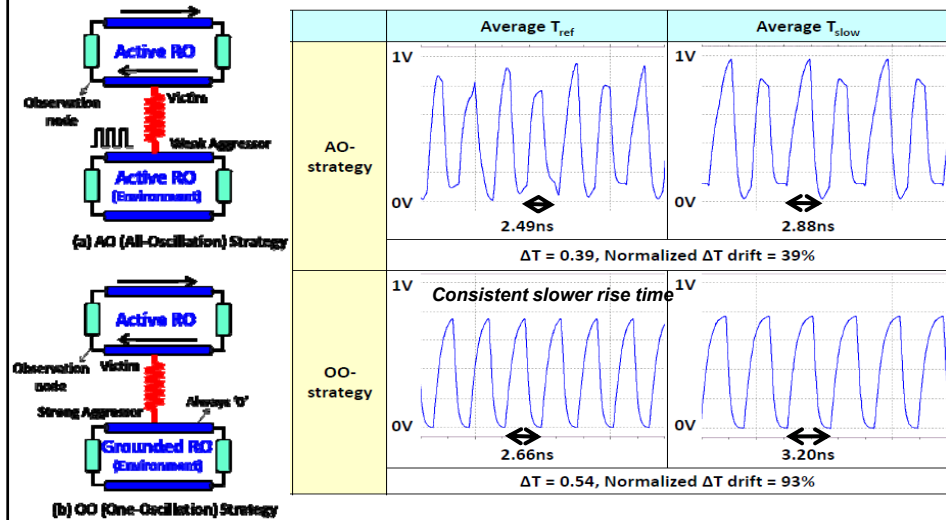
37

Waveform of 'End Point' and 'Observation point' in NO-strategy



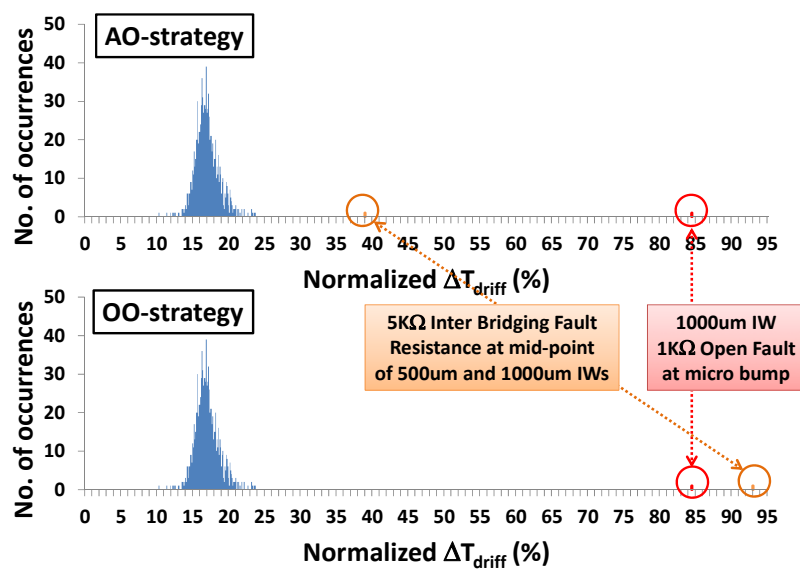
38

Example for an Inter-Bridging Fault



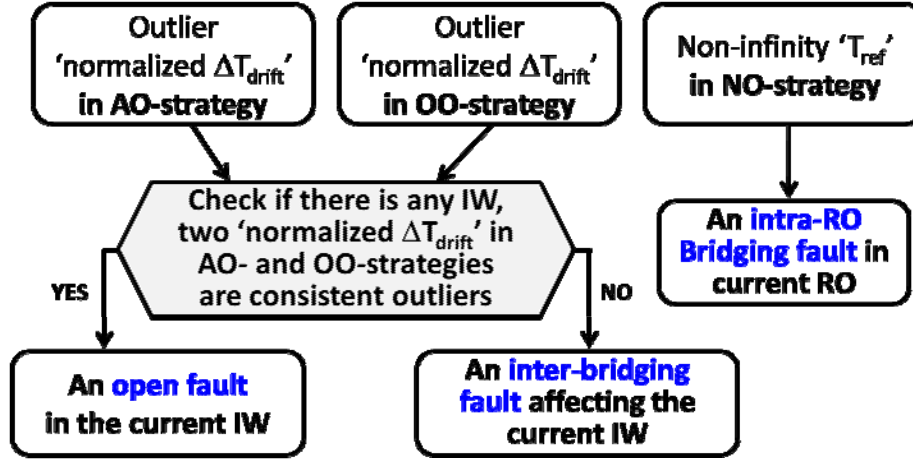
39

Example for an Inter-Bridging Fault



40

Fault Type Classification



41

Normalized ΔT_{drift} for Outlier Analysis

For each IW w_i , we have **two versions of ΔT** :

$$\Delta T_{sim}(w_i) = T_{ST_sim}(w_i) - T_{REF_sim}(w_i)$$

$$\Delta T_{measure}(w_i) = T_{ST_measure}(w_i) - T_{REF_measure}(w_i)$$

$\Delta T_{drift}(w_i)$ represents the **drifting amount** of a measurement version of ΔT away from its simulation version:

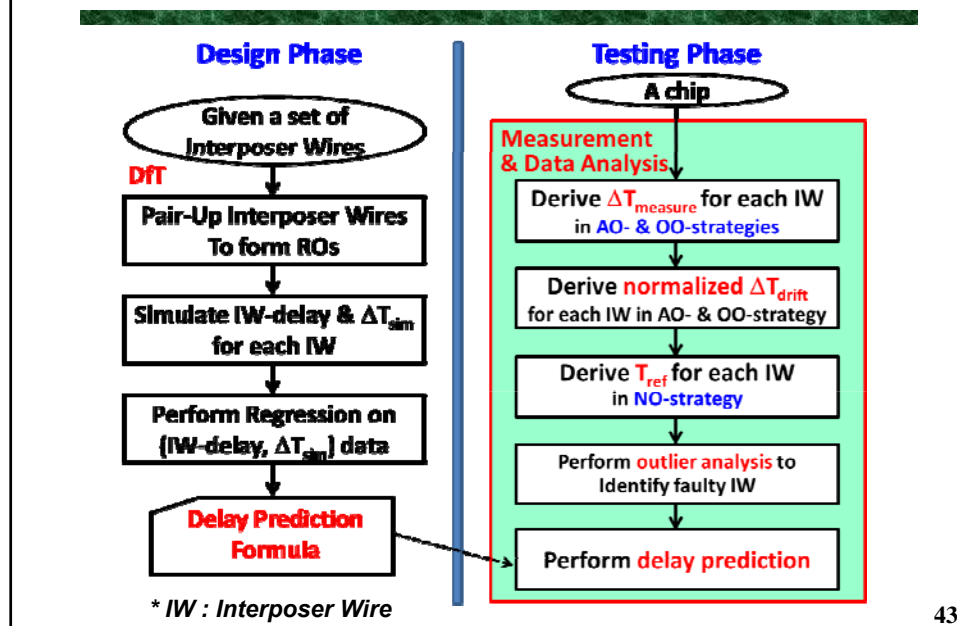
$$\Delta T_{drift}(w_i) = \Delta T_{measure}(w_i) - \Delta T_{sim}(w_i)$$

To take into account of the wire-length diversity, we further **normalize it**:

$$(\text{Normalized } \Delta T_{drift}) = \left(\frac{\Delta T_{measure} - \Delta T_{sim}}{\Delta T_{sim}} \right) \cdot 100\%$$

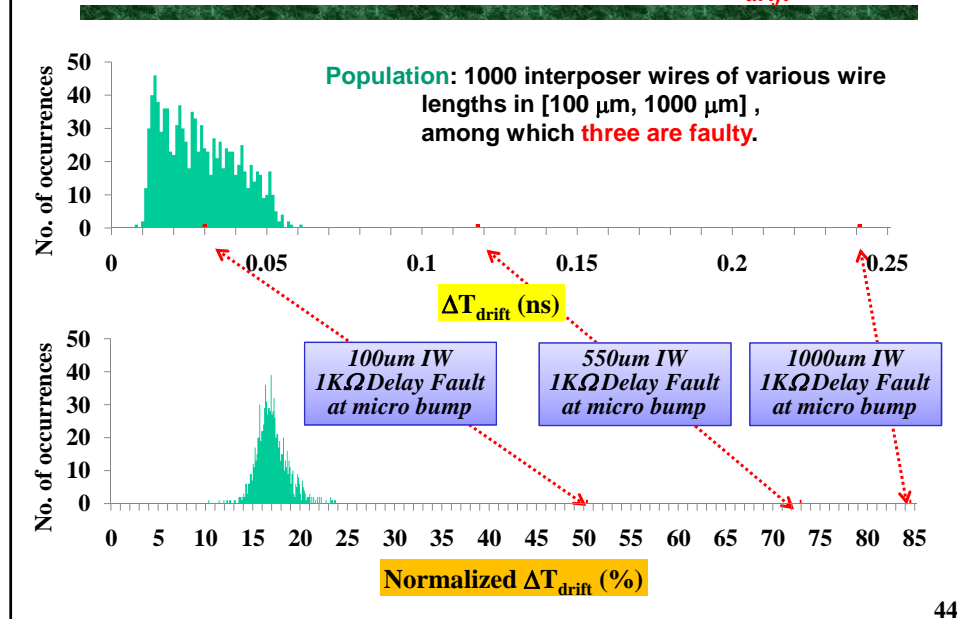
42

Testing and Characterization Flow



43

Fault Detection (Finding Outliers in Normalized ΔT_{drift})



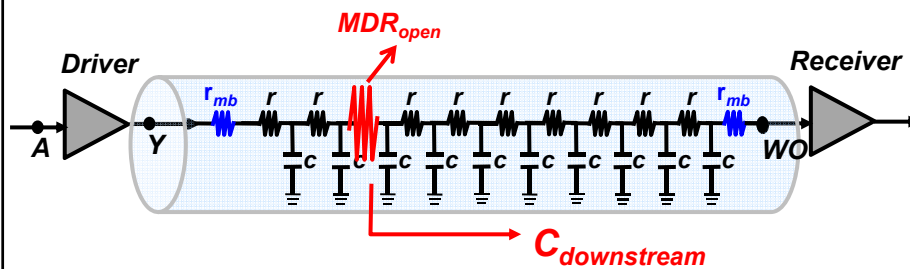
44

Fault Detection Capability (For Resistive Open Faults)

MDR_{open} : Minimum Detectable Open Fault Resistance $245\ \Omega$

This metric refers to the open fault resistance value beyond which the proposed test method can detect the fault successfully based on the **outlier analysis using 3σ rule**.

Detectable Extra-RC: $(MDR_{open}) * (C_{downstream})$ $50.7\ ps$



45

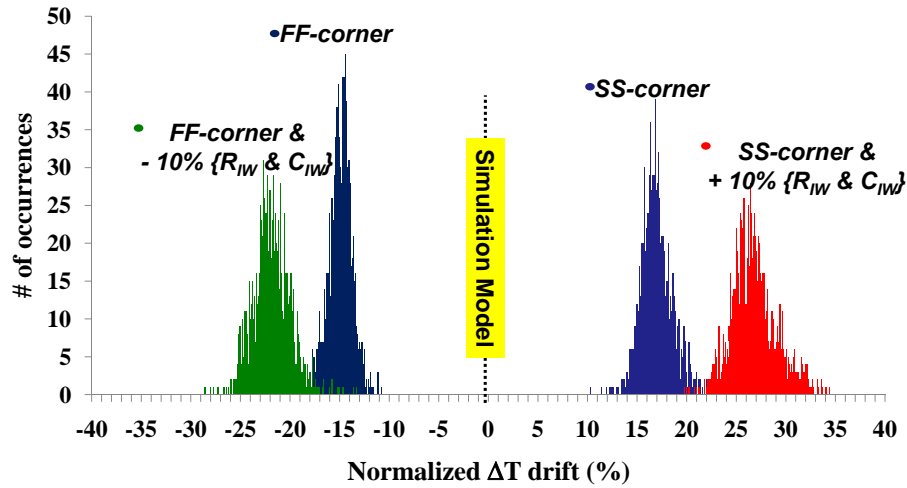
Resistive Open Fault Detection Capability

A resistive open fault occurring at the **micro-bump of the driver side** of a **1000um** long interposer wire.

Pseudo Chip Conditions	MDR_{open} (Min. Detectable Open Fault Resistance)	Detectable Extra-RC
#1 (FF & -10% RC)	$245\ \Omega$	$50.7\ ps$
#2 (FF)	$76\ \Omega$	$17.5\ ps$
#3 (SS)	$113\ \Omega$	$26.0\ ps$
#4 (SS & +10% RC)	$78\ \Omega$	$19.7\ ps$
Average	$145\ \Omega$	$31.4\ ps$

46

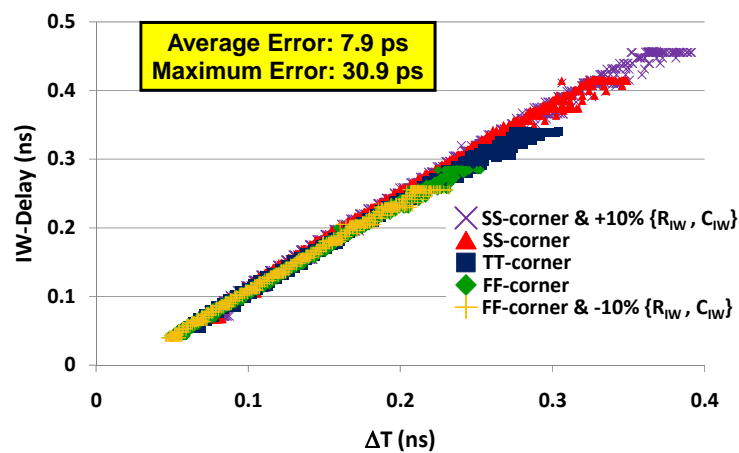
Process Drift from Simulation Model



47

Fault-Free IW-Delays vs. ΔT for Various Pseudo Chip Conditions

Implication: Regression mode derived by TT corner is applicable under process variations.



48

Summary of VOT-Based Oscillation Test

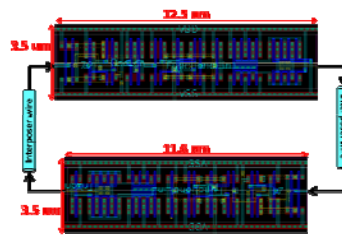
The **interposer** needs to be **tested alone and thoroughly**.
And also, when a 2.5-D IC fails,
We know **if the interposer should be responsible**.

Test Time

11 ms for 1024 wires
413 ms for 32K wires

Using 10MHz TCLK

Layout of an RO



~55%
Over boundary
scan test

49

Conclusion

Criterion	PV-test	VOT-based oscillation test
Basic Concept	Check if pulse will vanish	Measure ΔT
Fault Detection Scheme	Test threshold based	Outlier analysis
Area overhead	55.5% over IEEE-1149.1	55.7% over IEEE-1149.1
Test time	<u>0.82</u> ms for 1024 wires <u>26.21</u> ms for 32K wires	<u>4.7</u> ms for 1024 wires <u>177</u> ms for 32K wires
Other benefits	No post-processing On-the-spot diagnosis Easier self-repair	Delay characterization Process tracking

Outlier analysis: A **measurement sample** that significantly deviates away from the entire population indicates a **fault**

50

國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing



Chapter 11
Logic Diagnosis

Outline

- ➡ ☐ **Introduction**
- ☐ **Combinational Logic Diagnosis**
- ☐ **Scan Chain Diagnosis**
- ☐ **Logic BIST Diagnosis**
- ☐ **Conclusion**

Ch11-2

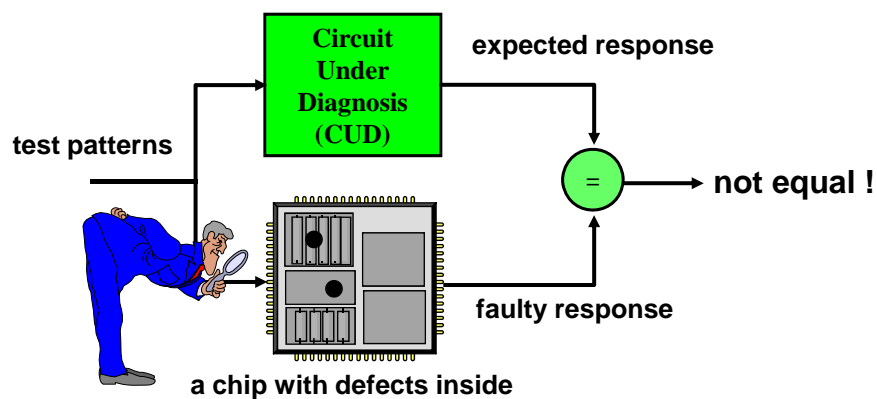
What would you do when chips fail?

- ❑ **Is it due to design bugs?**
 - If most chip fails with the same syndrome when running an application
- ❑ **Is it due to parametric yield loss?**
 - Timing-related failure?
 - Insufficient silicon speed?
 - Noise-induced failure?
 - supply noise, cross-talk, leakage, etc.?
 - Lack of manufacturability?
 - inappropriate layout?
- ❑ **Is it due to random defects?**
 - Via misalignment, Via/Contact void, Mouse bite,
 - Unintentional short/open wires, etc.

Ch11-3

Problem: Fault Diagnosis

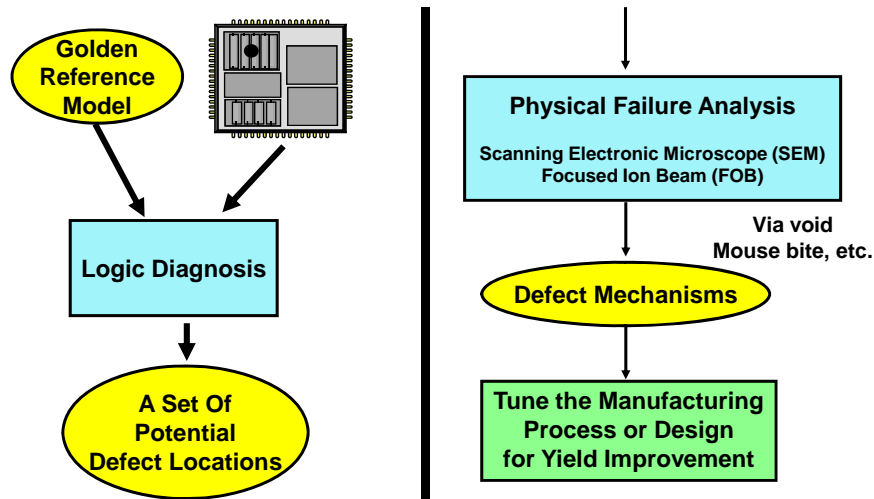
This chapter focuses more on diagnosis of defects or faults, not design bugs



Question: Where are the fault locations ?

Ch11-4

Diagnosis For Yield Improvement



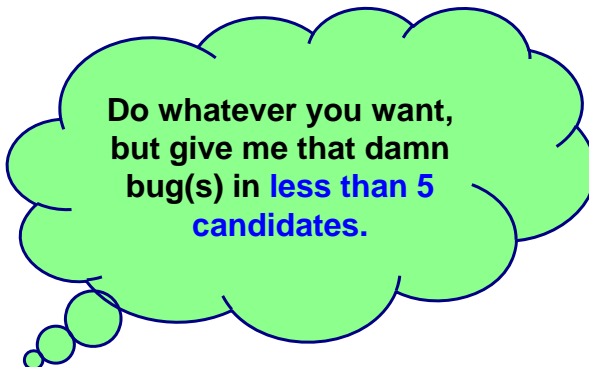
Ch11-5

Quality Metrics of Diagnosis

- ❑ **Success rate**
 - The percentage of hitting at least one defect in the physical failure analysis
 - This is the ultimate goal of failure analysis
- ❑ **Diagnostic resolution**
 - Total number of fault candidates reported by a tool
 - The perfect diagnostic resolution is 1
 - Though perfect resolution does not necessarily imply high hit rate
- ❑ **First-hit index**
 - Used for a tool that reports a ranked list of candidates
 - Refers to the index of the first candidate in the ranked list that turns out to be a true defect site
 - Smaller first-hit index indicates higher accuracy
- ❑ **Top-10 hit**
 - Used when there are multiple defects in the failing chip
 - The number of true defects in the top 10 candidates

Ch11-6

Challenge

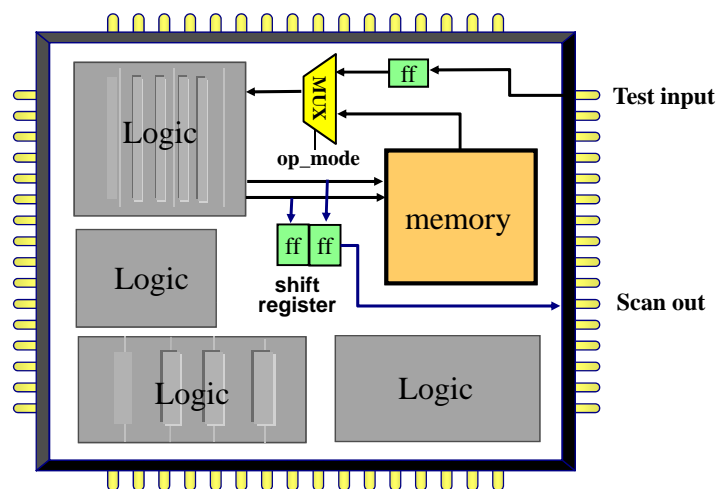


failure analysis people
under time-to-market pressure

Ch11-7

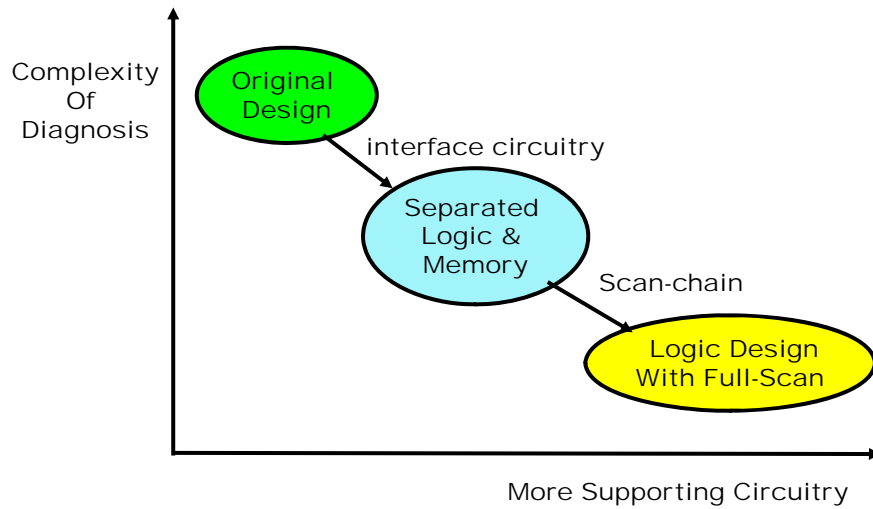
Supporting Circuitry

Supporting Circuitry:
Makes Logic's inputs controllable and outputs observable



Ch11-8

Design For Diagnosis



Ch11-9

Possible Assumptions Used in Diagnosis

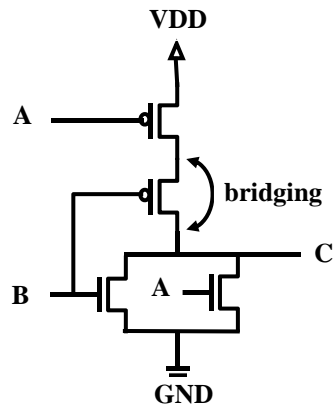
- ❑ **Stuck-At Fault Model Assumption**
 - The defect behaves like a stuck-at fault
- ❑ **Single Fault Assumption**
 - Only one fault affecting any faulty output
- ❑ **Logical Fault Assumption**
 - A fault manifests itself as a logical error
- ❑ **Full-Scan Assumption**
 - The chip under diagnosis has to be full-scanned

*Note: A diagnosis approach **less dependent** on the fault assumptions is more capable of dealing with practical situations.*

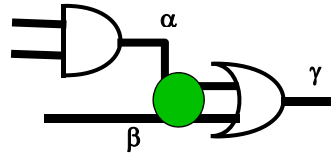
Ch11-10

Examples of Faults

Node Fault



Short Fault (Bridging)



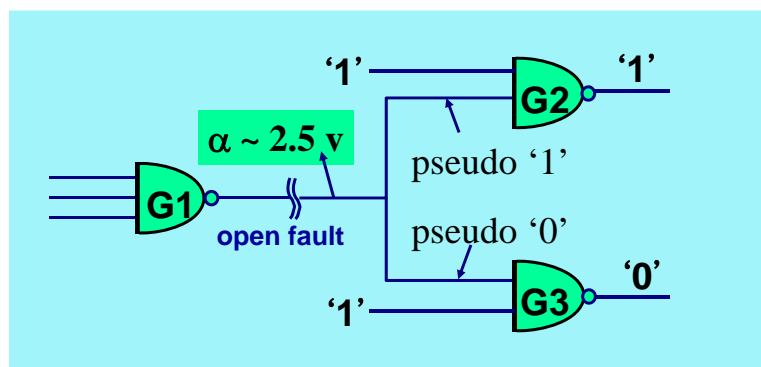
Most diagnosis algorithms perform at the gate level, trying to identify the troubling signals or cells

Ch11-11

Byzantine Open Fault

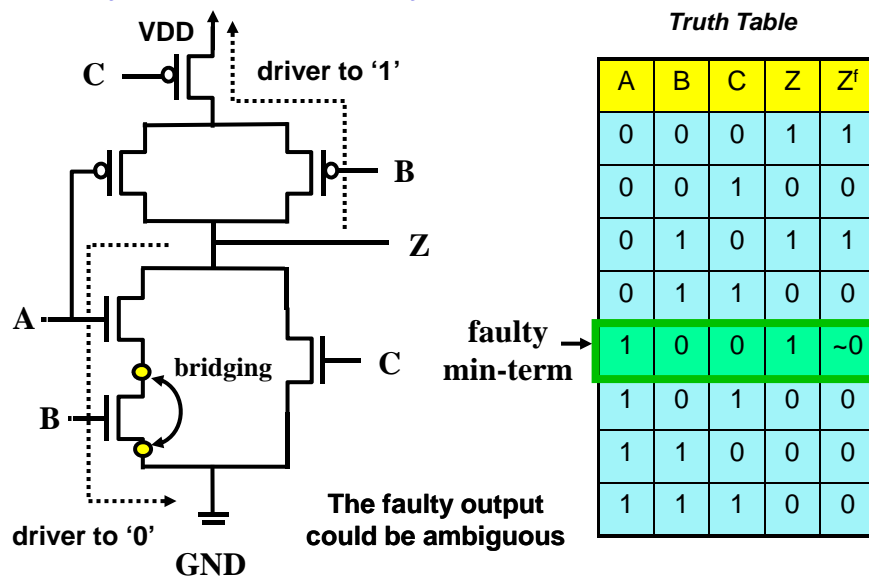
Definition of Byzantine Fault:

- A fault that causes an ambiguous voltage level



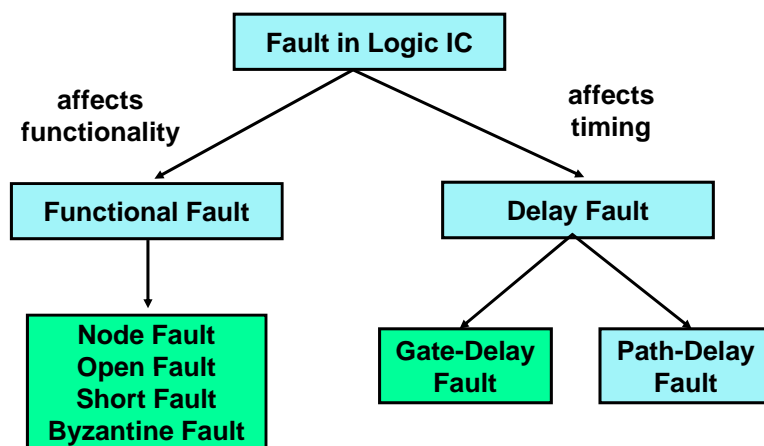
Ch11-12

A Byzantine Node Type



Ch11-13

Fault Classification



Ch11-14

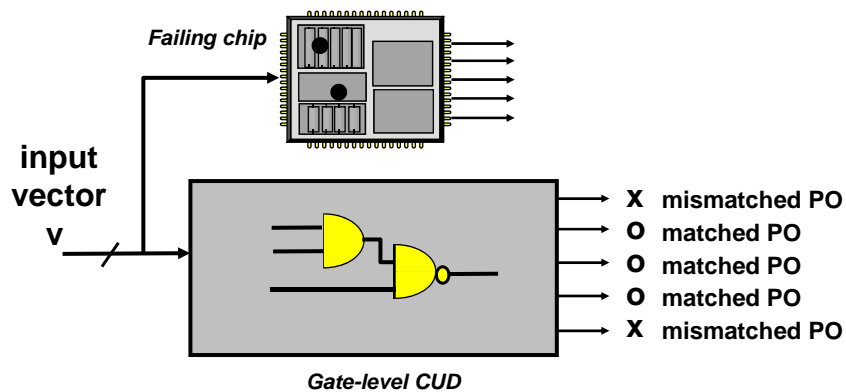
Outline

- Introduction
- ➔ □ **Combinational Logic Diagnosis**
 - **Cause-Effect Analysis**
 - Effect-Cause Analysis
 - Chip-Level Strategy
 - Diagnostic Test Pattern Generation
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

Ch11-15

Terminology

- **Device Under Diagnosis (DUD): The Failing Chip**
- **Circuit Under Diagnosis (CUD): The Circuit Model**
- **Failing Input Vector: Causes Mismatches**



Ch11-16

Cause-Effect Analysis

- ❑ **Fault dictionary (pre-analysis of all causes)**
 - Records test response of every fault under the applied test set
 - Built by intensive fault simulation process
- ❑ **A chip is diagnosed (effect matching)**
 - By matching up the failing syndromes observed at the tester with the pre-stored fault dictionary

Ch11-17

Fault Dictionary Example

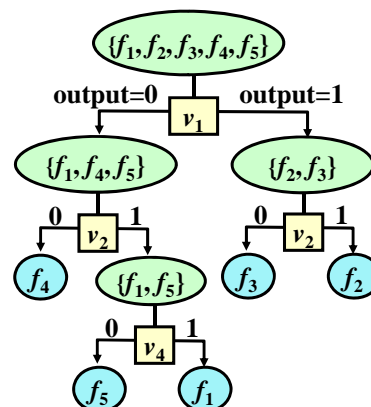


(a) Circuit under diagnosis

Circuits	Test vectors in terms of (a, b, c)				
	v_1	v_2	v_3	v_4	v_5
fault-free	0	0	0	0	1
f_1	0	1	1	1	1
f_2	1	1	1	0	1
f_3	1	0	0	1	1
f_4	0	0	1	0	0
f_5	0	1	1	0	1

(b) Full-response dictionary

A diagnosis session:
traverse from a path from root to a leaf



(c) Diagnostic tree

Ch11-18

Fault Dictionary Reduction – P&R

(a) Full-response table

Fault	Output Response (z_1, z_2)			
	t_1	t_2	t_3	t_4
f_1	10	10	11	10
f_2	00	00	11	00
f_3	00	00	00	00
f_4	01	00	00	01
f_5	01	00	01	01
f_6	01	00	01	01
f_7	10	00	10	00
f_8	11	11	11	11

Fault	Pass (0) or Fail (1)			
	t_1	t_2	t_3	t_4
f_1	1	1	0	1
f_2	1	0	0	1
f_3	1	0	1	1
f_4	1	0	1	0
f_5	1	0	1	0
f_6	1	0	1	0
f_7	1	0	1	1
f_8	0	1	0	1

(b) Pass-fail dictionary

(c) P&R compression dictionary

Fault ID	Pass-fail + Extra outputs			
	t_1	t_2	t_3	t_4
f_1	1 1	1	0 1	1
f_2	1 0	0	0 1	1
f_3	1 0	0	1 0	1
f_4	1 0	0	1 0	0
f_5	1 0	0	1 1	0
f_6	1 0	0	1 1	0
f_7	1 1	0	1 0	1
f_8	0 1	1	0 1	1

Response of z_1

Response of z_2

Ch11-19

Detection Fault Dictionary

(a) Full-response table

Fault ID	Output Response (z_1, z_2)			
	t_1	t_2	t_3	t_4
f_1	10	10	11	10
f_2	00	00	11	00
f_3	00	00	00	00
f_4	01	00	00	01
f_5	01	00	01	01
f_6	01	00	01	01
f_7	10	00	10	00
f_8	11	11	11	11

failing output vectors

(c) Detection dictionary

Fault ID	Detection information (Test ID : Output Vector)
f_1	$t_1:10 \ t_2:10 \ t_4:10;$
f_2	$t_1:00 \ t_4:00;$
f_3	$t_1:00 \ t_3:00 \ t_4:00;$
f_4	$t_1:01 \ t_3:00;$
f_5	$t_1:01 \ t_3:01;$
f_6	$t_1:01 \ t_3:01;$
f_7	$t_1:10 \ t_3:10 \ t_4:00;$
f_8	$t_2:10 \ t_4:11;$

Fault ID	Pass (1) or Fail (0)			
	t_1	t_2	t_3	t_4
f_1	1	1	0	1
f_2	1	0	0	1
f_3	1	0	1	1
f_4	1	0	1	0
f_5	1	0	1	0
f_6	1	0	1	0
f_7	1	0	1	1
f_8	0	1	0	1

(b) Pass-fail dictionary

Ch11-20

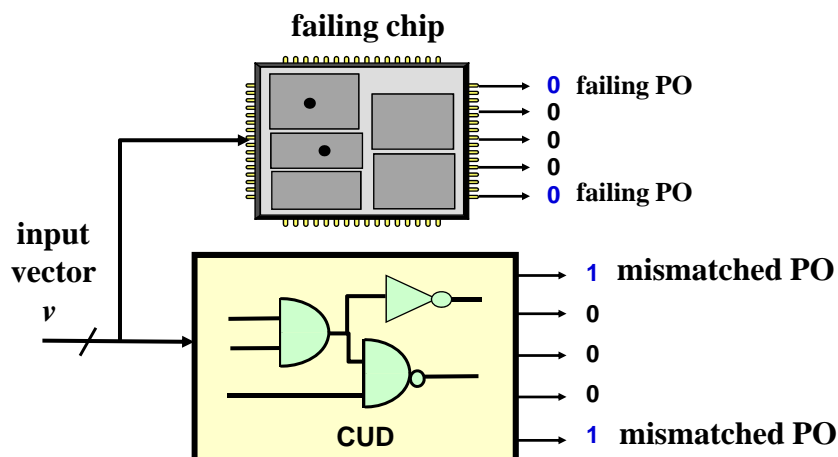
Outline

- Introduction
- Combinational Logic Diagnosis
 - Cause-Effect Analysis
 - ➡ ▪ **Effect-Cause Analysis**
 - Chip-Level Strategy
 - Diagnostic Test Pattern Generation
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

Ch11-21

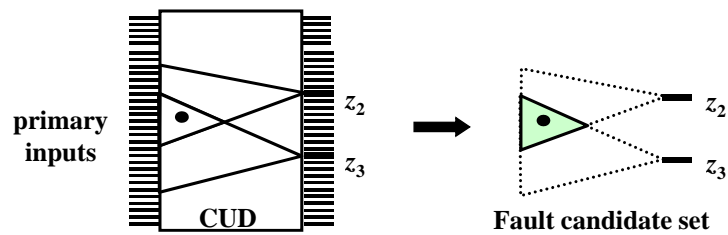
Terminology: Mismatched Output

*Effect-cause analysis does not build fault dictionary
It predicts fault locations by analyzing CUD from mismatch PO's*

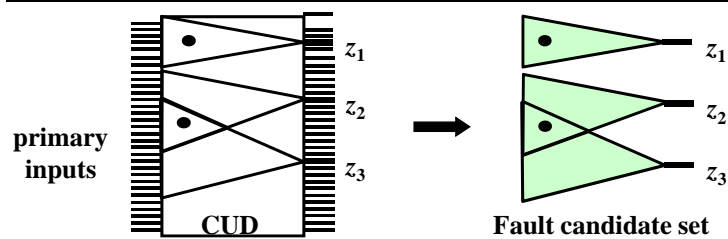


Ch11-22

Structural Pruning – Intersection or Union?



(a) Cone intersection.



(b) Cone union when there are multiple faults.

Ch11-23

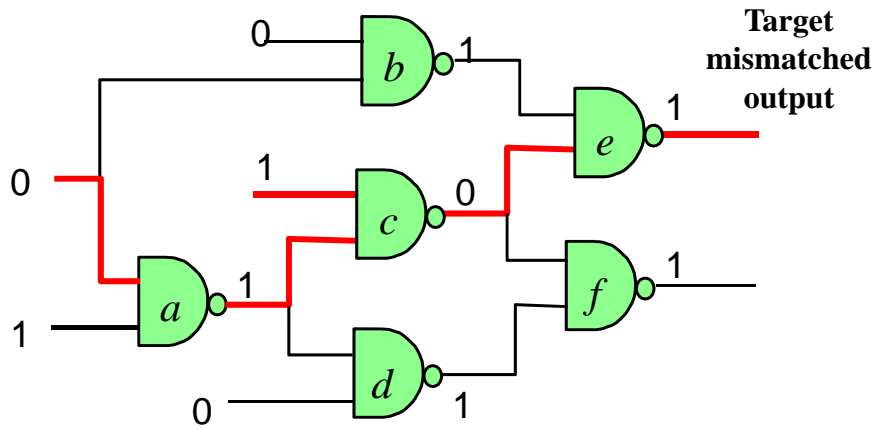
Backtrace Algorithm

- ❑ **Trace back from each mismatched PO**
 - To find out suspicious faulty locations
- ❑ **Functional Pruning**
 - During the traceback, some signals can be disqualified from the fault candidate set based on their signal values.
- ❑ **Rules**
 - (1) At a controlling case (i.e., 0 for a NAND gate): Its fanin signals with non-controlling values (i.e., 1) are excluded from the candidate set.
 - (2) At a non-controlling case (i.e., 1 for a NAND gate): Every fanin signal remains in the candidate set.

Ch11-24

Backtrace Example

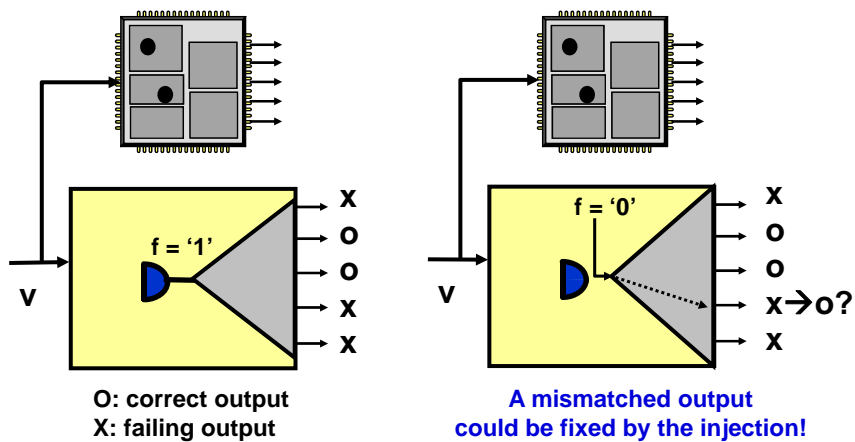
All suspicious fault locations are marked in red.



Ch11-25

Terminology – Injection

An injection at a signal f flips its current value which could create value-change events downstream.

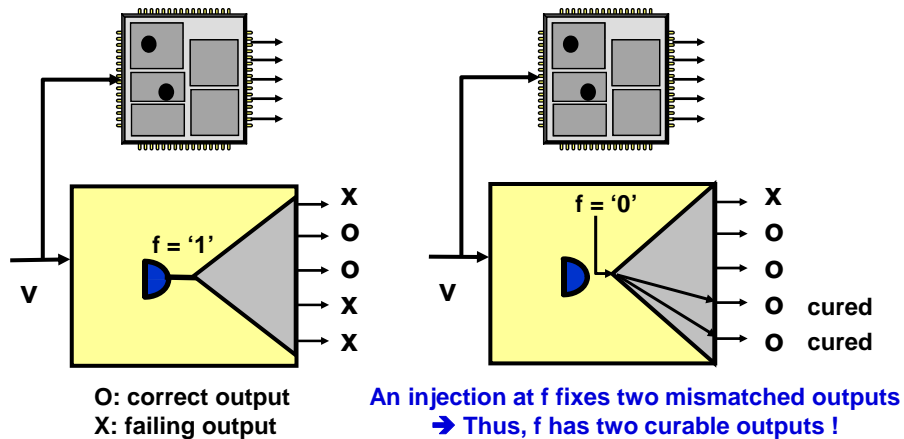


Ch11-26

Terminology – Curable Output

□ Diagnosis Criterion

- A signal is more suspicious if it has more curable outputs



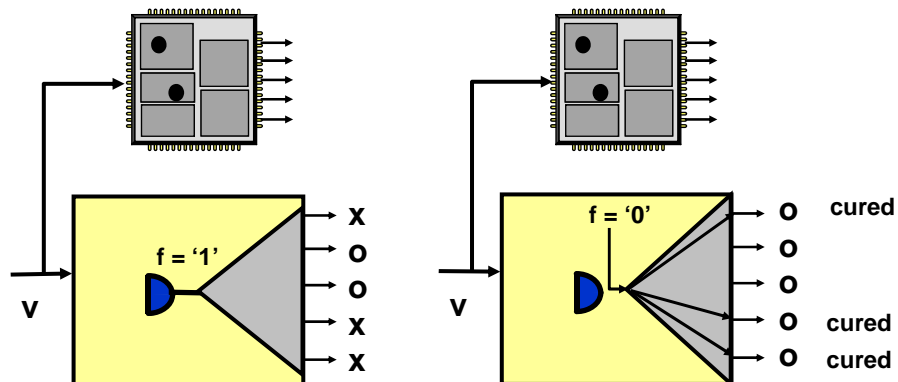
Ch11-27

Terminology – Curable Vectors

v is a curable vector by f

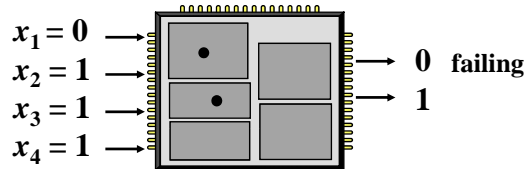
- because an injection at f exists such that
it cures all mismatches without creating new one

Curable vector is a stronger diagnosis indicator than curable output !

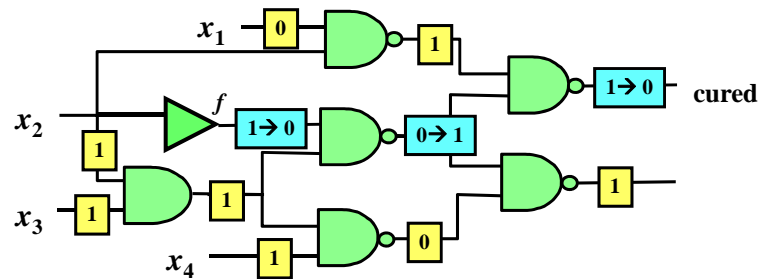


Ch11-28

Example of Curable Vector



(a) Failing Chip



(b) Circuit Under Diagnosis

Ch11-29

Why Curable Vector ?

Information theory

- A less probable event contains more information
- Curable output is an easy-to-satisfy criterion, high aliasing
- Curable vector is a hard-to-satisfy criterion, low aliasing

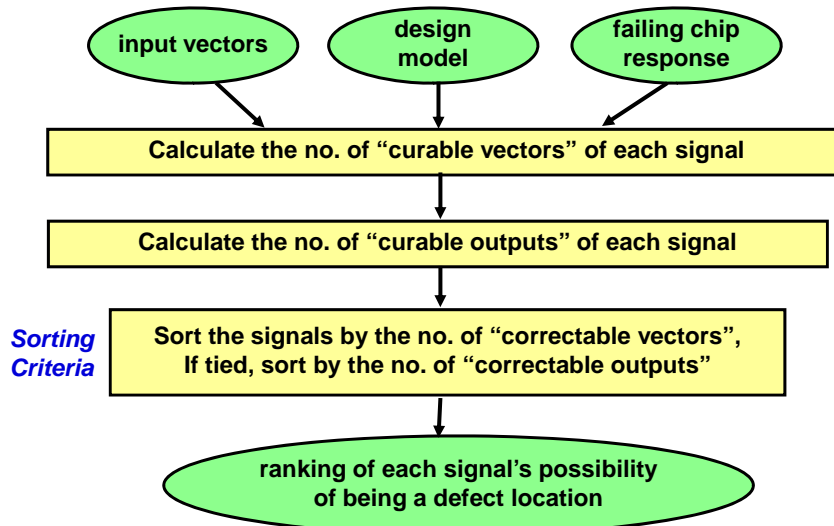
Not all failing input vectors are equal !

Niche input vector

- Is an failing input vector that activates only one fault
- Likely to be a curable vector of certain signals
- Few, but tells more about the real fault locations

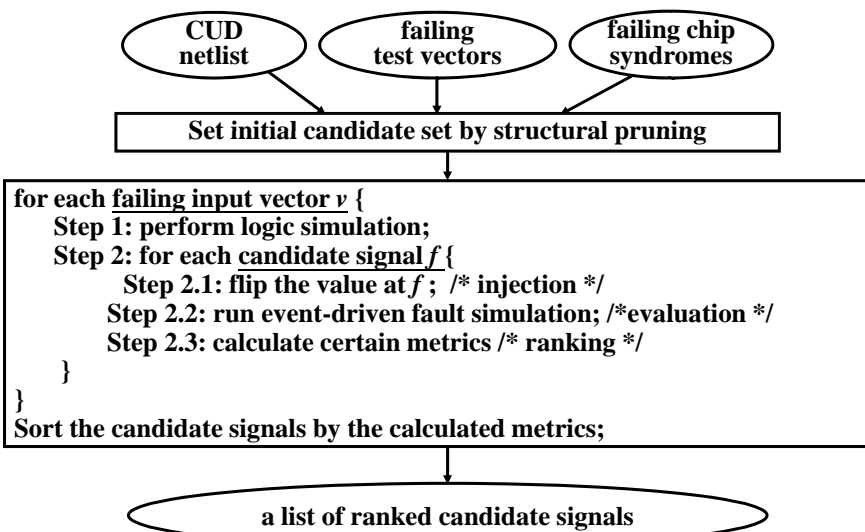
Ch11-30

Inject-and-Evaluate Paradigm



Ch11-31

Detailed Computation – Inject-and-Evaluate Paradigm

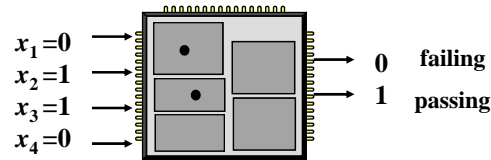


Ch11-32

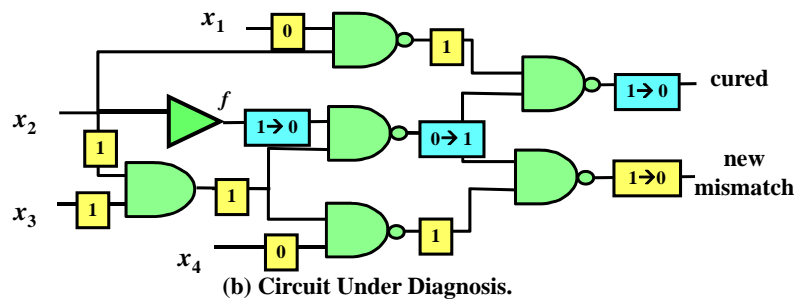
Reward-and-Penalty Heuristic

Rank1: curable vector count

*Rank2 = (curable output count – 0.5 * new mismatched output count)*



(a) Failing Chip.

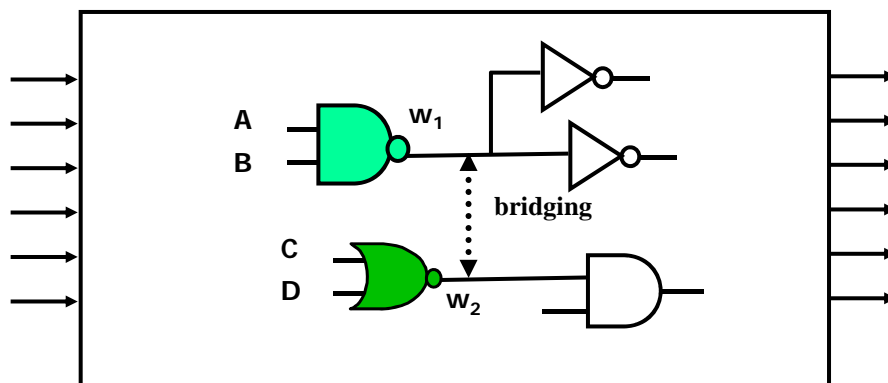


(b) Circuit Under Diagnosis.

Ch11-33

Targeting Bridging Faults

Even in a realistic bridging fault, there is only one victim at any time. This victim will expose his location by owning some curable vectors.

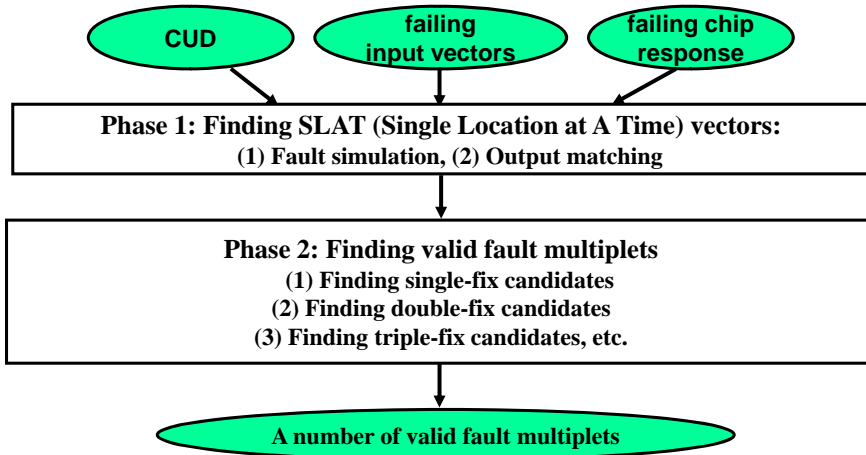


Ch11-34

SLAT Paradigm

Ref: *SLAT* (Single Location At a Time) paradigm [Bartenstein 2001]

Note: A SLAT vector is a curable vector



Ch11-35

Example: SLAT Paradigm

Failing Input Vectors	Signals in the CUD						
	f_1	f_2	f_3	f_4	f_5	f_6	f_7
v_1	*				*		
v_2	*	*	*				
v_3			*	*			*
v_4					*	*	
v_5		*			*		
v_6		*			*		
v_7	*		*				
v_8			*				*
v_9			*		*		
v_{10}					*		*

A mark * means the corresponding vector is a SLAT vector of the corresponding signal. $(f_3 \text{ and } f_5)$ is a valid fault multiplet

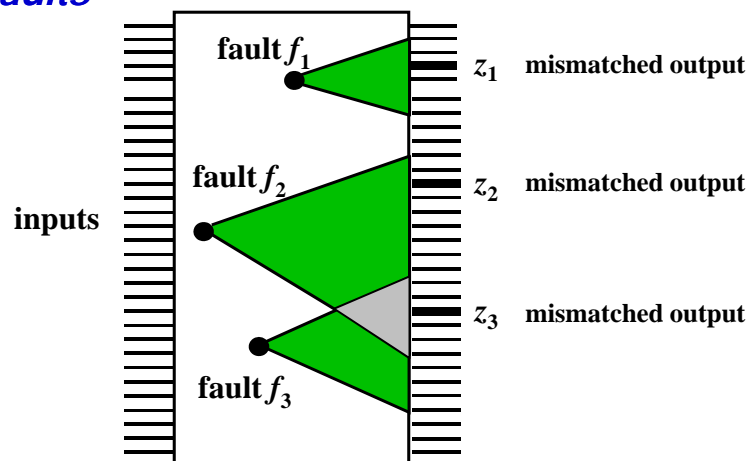
Ch11-36

Outline

- Introduction
- Combinational Logic Diagnosis
 - Cause-Effect Analysis
 - Effect-Cause Analysis
 - ➡ ▪ **Chip-Level Strategy**
 - **Diagnostic Test Pattern Generation**
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

Ch11-37

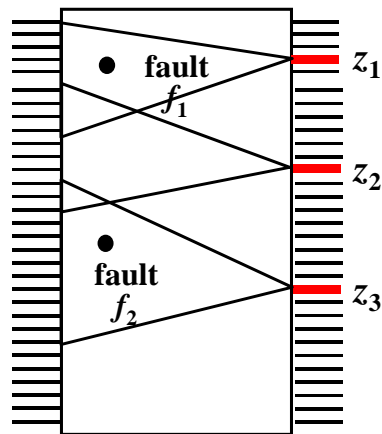
Structurally Dependent and Independent Faults



Fault f_1 is an independent fault.
Faults f_2 and f_3 are dependent faults.

Ch11-38

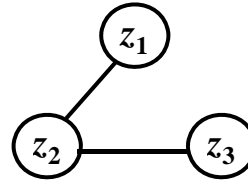
Dependency Graph



*Direct divide-and-Conquer
does not work well !*



dependency graph



one connected component

Two **independent** faults, f_1 and f_2 , lead to one diagnosis block.

Ch11-39

Main Strategy: Detach-Divide-and-then-Conquer

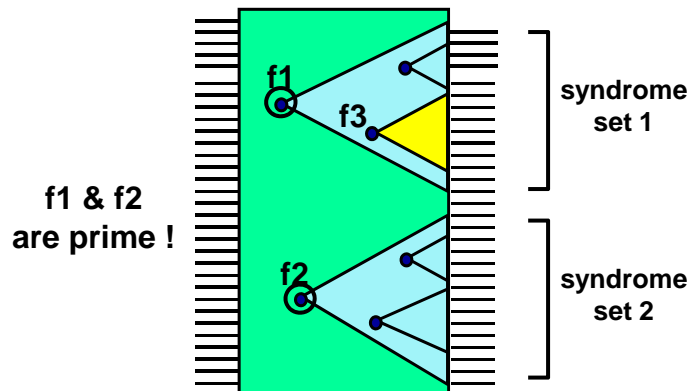
- ❑ **Phase 1: Isolate Independent Faults**
 - Search for prime candidates
 - Use word-level information
- ❑ **Phase 2: Locate Dependent Faults As Well**
 - Perform partitioning
 - Aim at finding one fault in each block

Ch11-40

Prime Candidates

A signal f is a prime candidate if

- (1) All failing input vectors are partially curable by f
- (2) Curable-Output-Set(f) is not covered by any other's

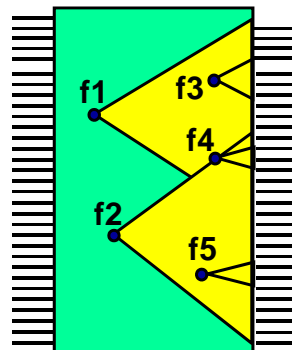


Ch11-41

Fake Prime Candidates

- Structurally Independent Faults
 - are often prime candidates
- Fake Prime Candidates
 - are prime candidates that are NOT really faults - aliasing

Example: Dependent Double Faults $f1$ & $f2$
May create fake prime candidates $\{f1, f2, f3\}$.



Ch11-42

Word-Level Registers and Outputs

*Signals in a design are often defined in words.
This property can be used to differentiate fake prime candidates from the real ones.*

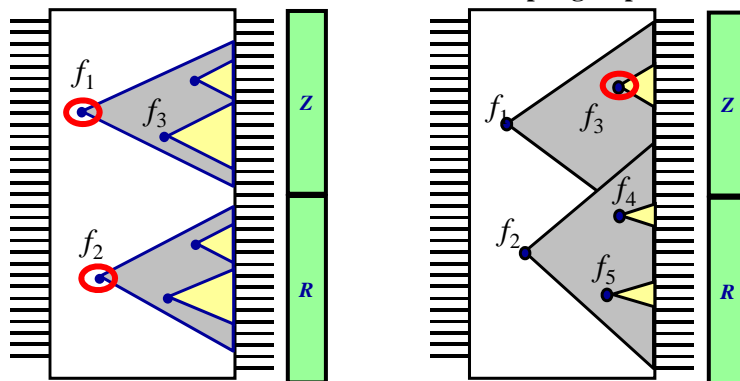
Word-Level Output: O1
Word-Level Registers: R1, R2, State

```
module  design( O1, ...)
  output[31:0]  O1;
  reg[31:0]     R1, R2;
  reg[5:0]      State
  ...
endmodule
```

Ch11-43

Word-Level Prime Candidates

Note: Z and R are two word-level output groups.



Original prime candidates: $\{f_1, f_2\}$
Word-level prime candidates $\{f_1, f_2\}$

Assumed original prime candidates: $\{f_3, f_4, f_5\}$
 $\{f_4, f_5\}$ will be identified as fake
→ Final Word-level prime candidates $\{f_3\}$

Ch11-44

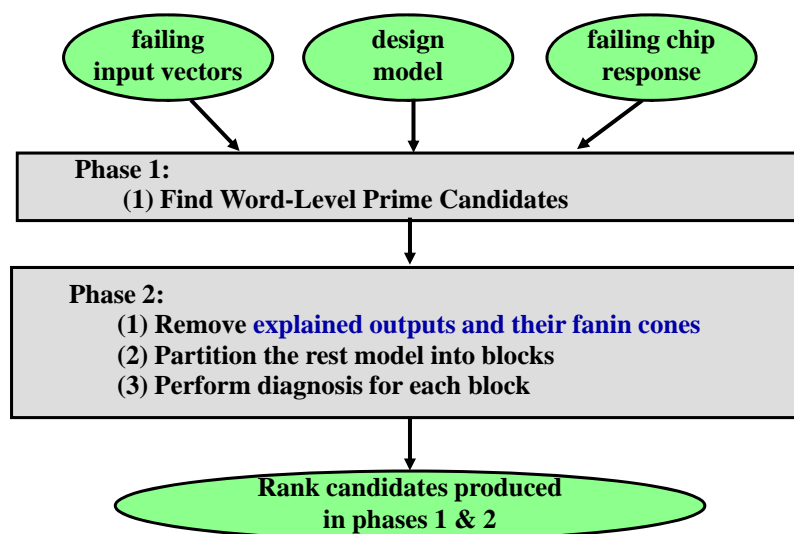
Efficiency of Using Word-Level Info.

- Without word-level Information
 - 2.4 real faults out of 72.3 candidates
- With word-level Information
 - 1.23 real faults out of 3.65 candidates

# of candidates	Original	After Filtering	Filtering Ratio
Prime Candidates	2.375	1.23	48.2 %
Fake Prime Candidates	69.96	2.42	96.5 %

Ch11-45

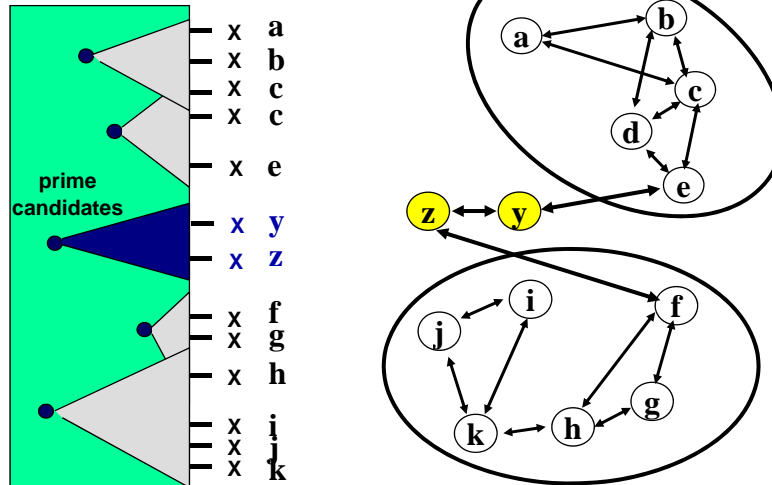
Overall Flow



Ch11-46

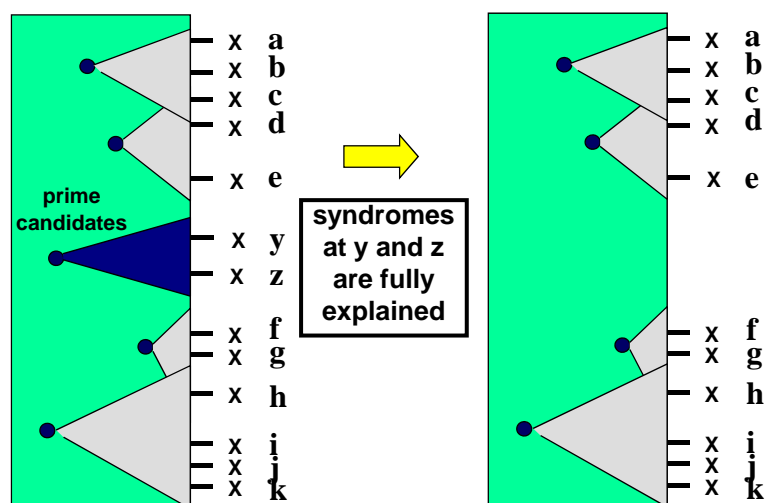
Grouping Using Dependency Graph

An example with five faults
One of them is identified as the prime candidate



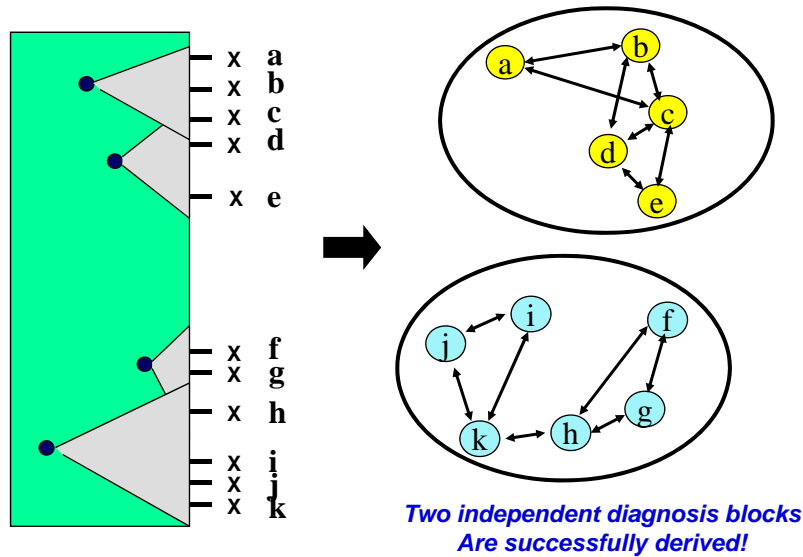
Ch11-47

Removed Explained Faulty Outputs



Ch11-48

Grouping Example



Ch11-49

Summary

□ Strategy

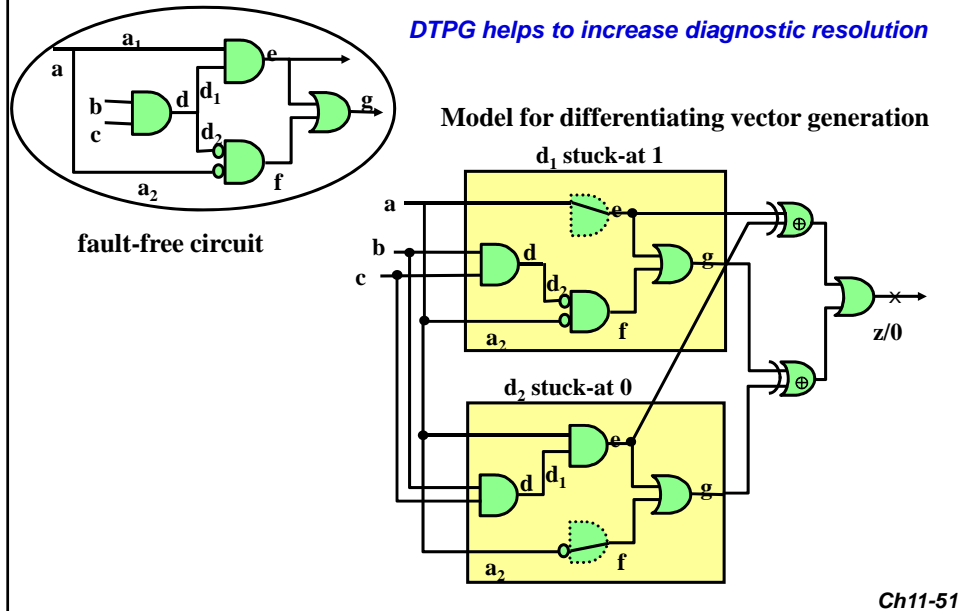
- (1) Search For Word-Level Prime Candidates
- (2) Identify Independent Faults First
- (3) Locate Dependent Faults As Well

□ Effectiveness

- identify 2.98 faults in 5 signal inspections
- find 3.8 faults in 10 signal inspections

Ch11-50

Diagnostic Test Pattern Generation

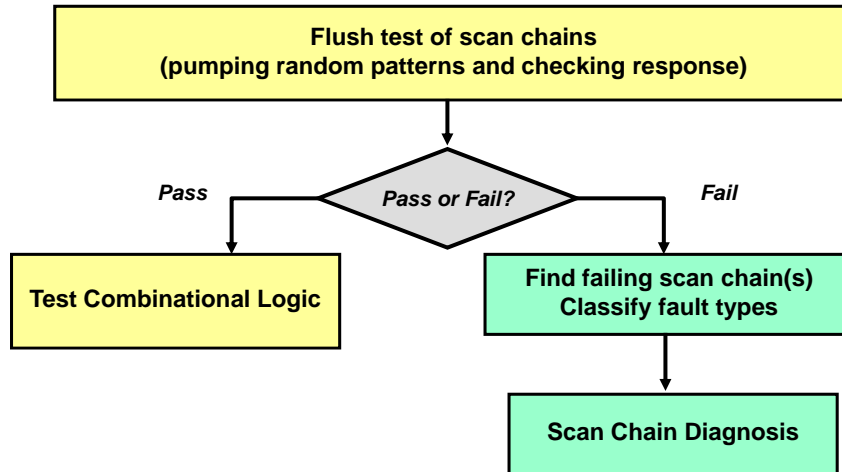


Outline

- Introduction
- Combinational Logic Diagnosis
- ▣ **Scan Chain Diagnosis**
 - Preliminaries
 - Hardware-Assisted Method
 - Signal-Profiling Based Method
- Logic BIST Diagnosis
- Conclusion

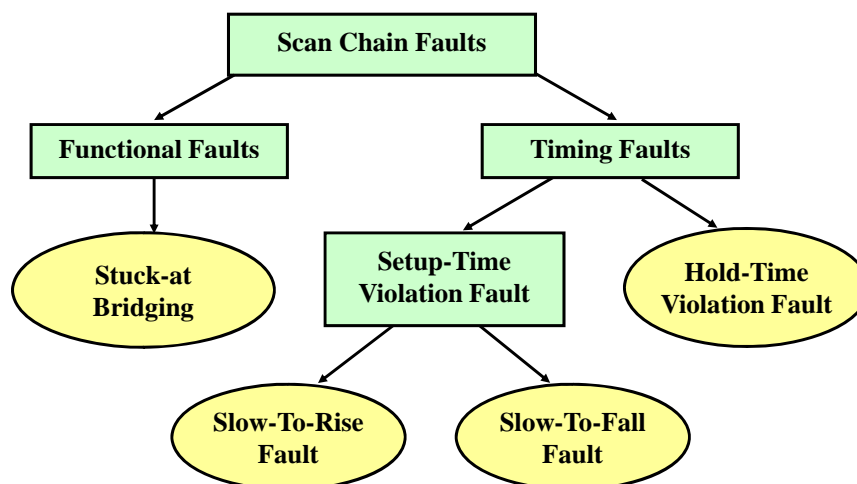
Ch11-52

Scan Test and Diagnosis



Ch11-53

Commonly Used Fault Types in Scan Chains

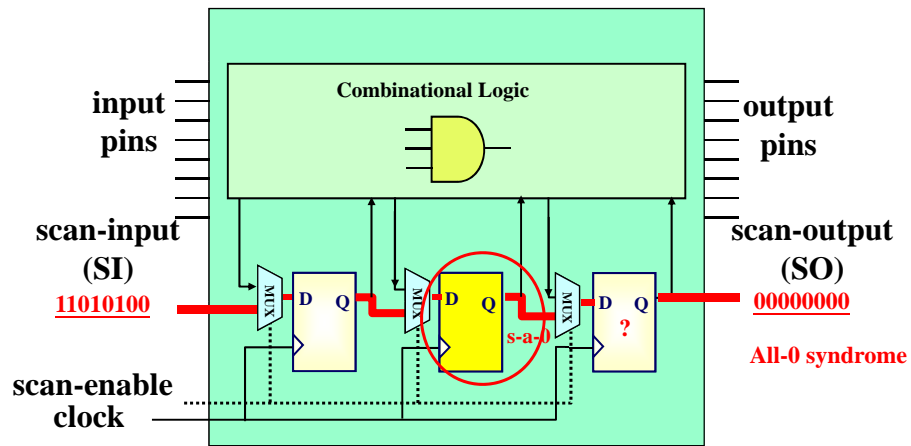


Each fault could be permanent or intermittent.

Ch11-54

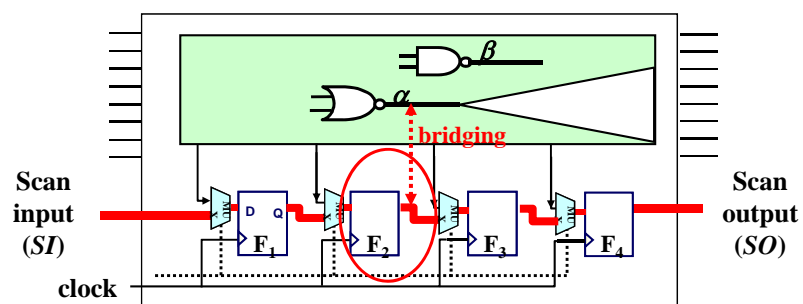
A Stuck-At Fault In the Chain

Effect: A **killer** of the scan-test sequence

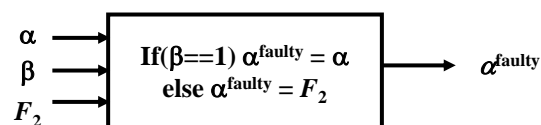


Ch11-55

A Realistic Bridging Fault Model



(a) Bridging between a flip-flop and a logic cell.

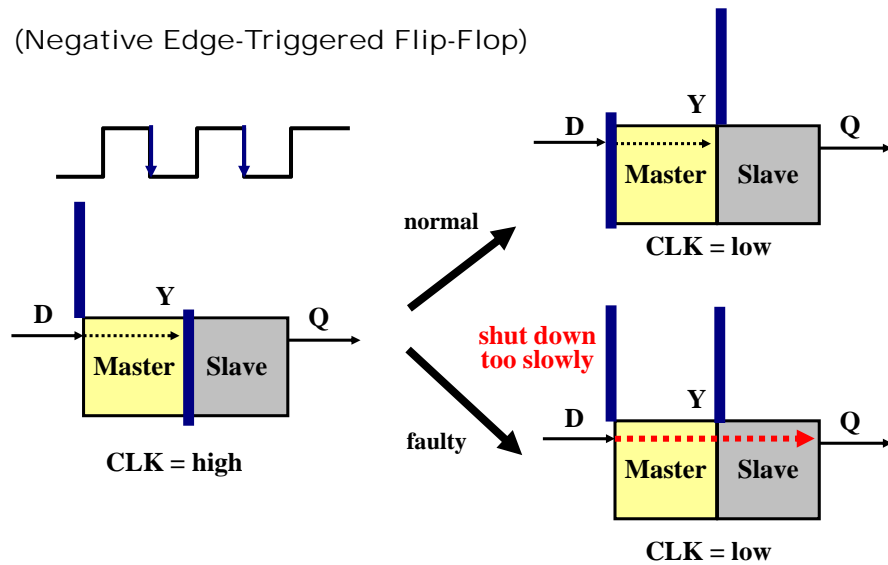


(b) Our bridging fault model.

Ch11-56

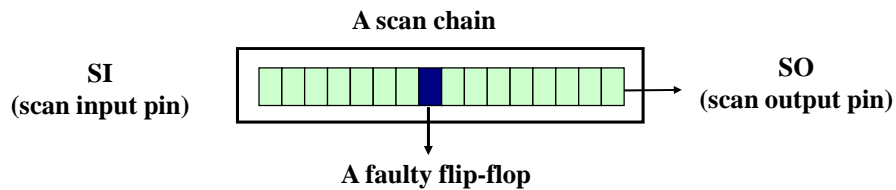
Potential Hold-Time Fault?

(Negative Edge-Triggered Flip-Flop)



Ch11-57

Example: Faulty Syndrome of a Scan Chain



Fault Type	Scan-In Pattern	Observed Syndrome
Stuck-at-0	1100110011001100	<u>0000000000000000</u>
Stuck-at-1	1100110011001100	<u>1111111111111111</u>
Slow-to-Rise	1100110011001100	1 <u>000</u> 1 <u>000</u> 1 <u>000</u> 1 <u>000</u>
Slow-to-Fall	1100110011001100	1101110111011100

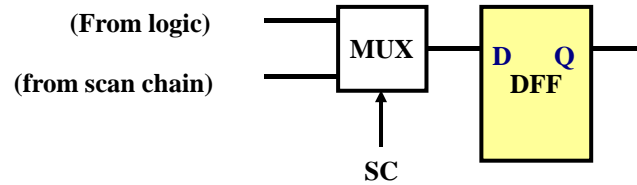
The rightmost bit goes into the scan first

The rightmost bit gets out of the scan first

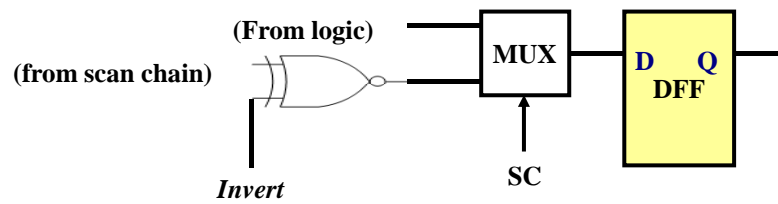
A underlined bit in the observed image is failing.

Ch11-58

Augmentation of a Flip-Flop for Easy Diagnosis



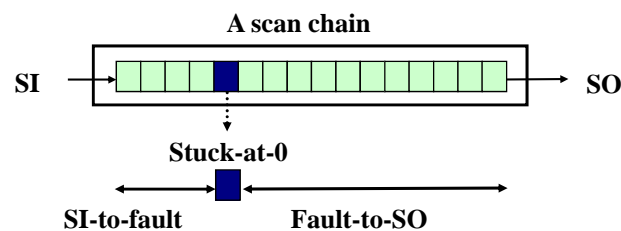
(a) A normal scan flip-flop.



(b) A modified scan flip-flop for easy inversion.

Ch11-59

Fault Location via Inversion Operation

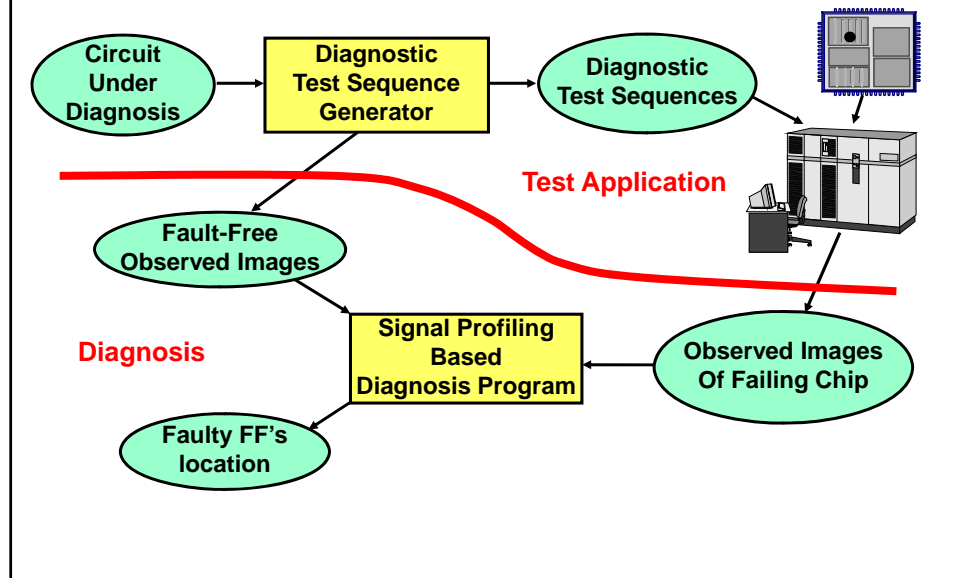


- (1) Original bitstream pattern = (111111111111111)
- (2) After scan-in: snapshot image = (111100000000000)
- (3) After inversion: snapshot image = (000001111111111)
- (4) After scan-out: observed image = (000001111111111)

The fault location is at the edge between 0's and 1's

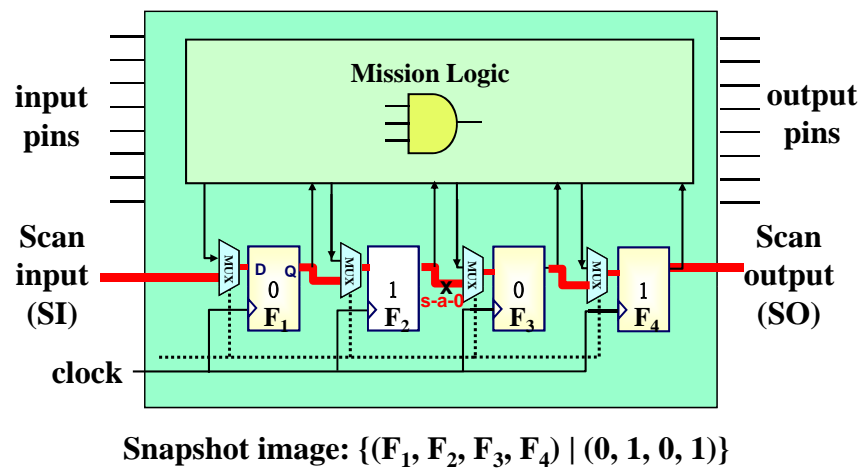
Ch11-60

Scan Chain Diagnosis Flow



Definition: Snapshot Image

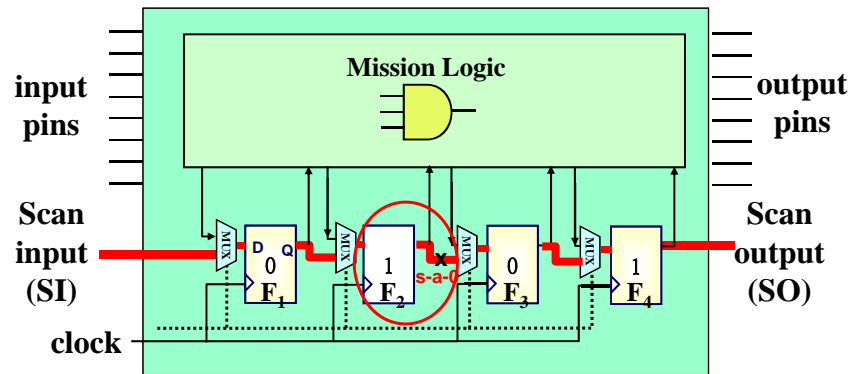
Def: A snapshot image is the combination of flip-flop values at certain time instance



Ch11-62

Definition: Observed Image

Def: An observed image is the scanned-out version of a snapshot image.

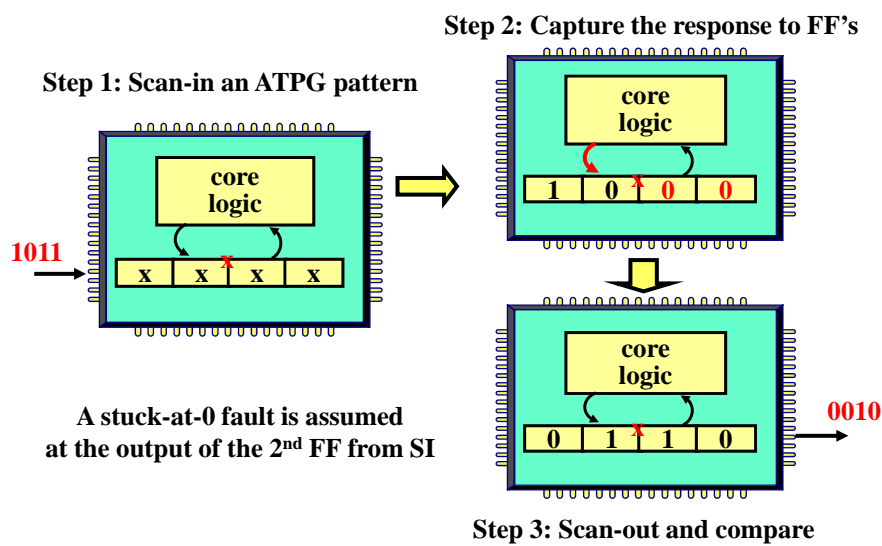


Snapshot image: $\{(F_1, F_2, F_3, F_4) \mid (0, 1, 0, 1)\}$

Observed image: $\{(F_1, F_2, F_3, F_4) \mid (0, 0, 0, 1)\}$

Ch11-63

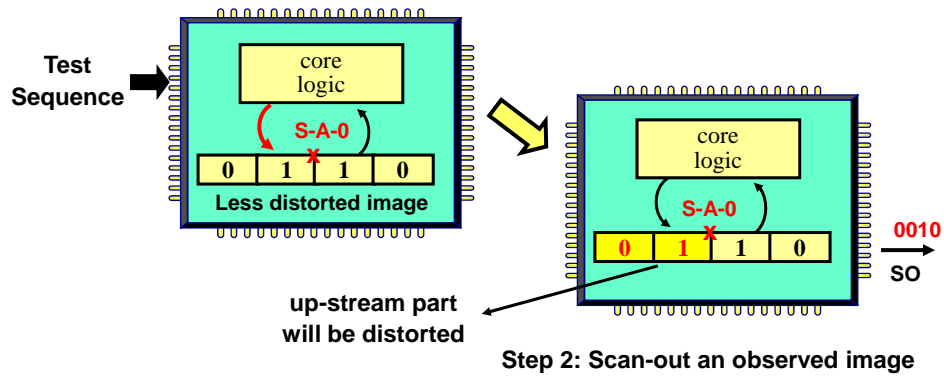
Modified Inject-and-Evaluate Paradigm



Ch11-64

Test Application: Run-and-Scan

Step 1: Apply a test sequence from PI's
→ Setting up a snapshot image at FF's

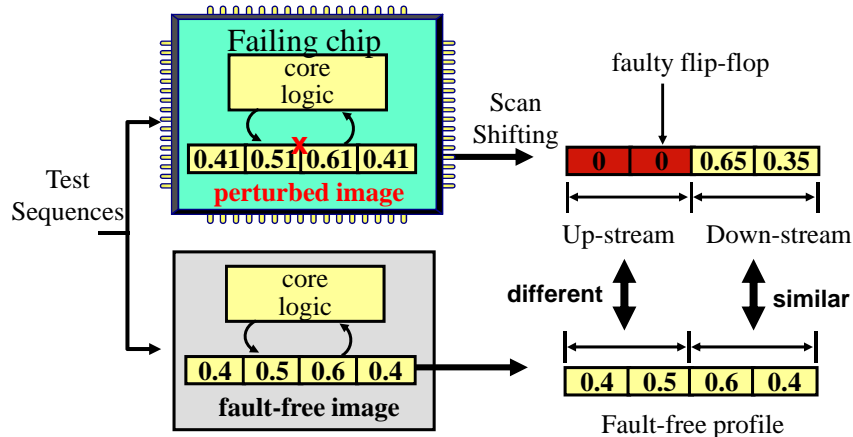


The fault location is embedded in the observed image

Ch11-65

Signal Profiling

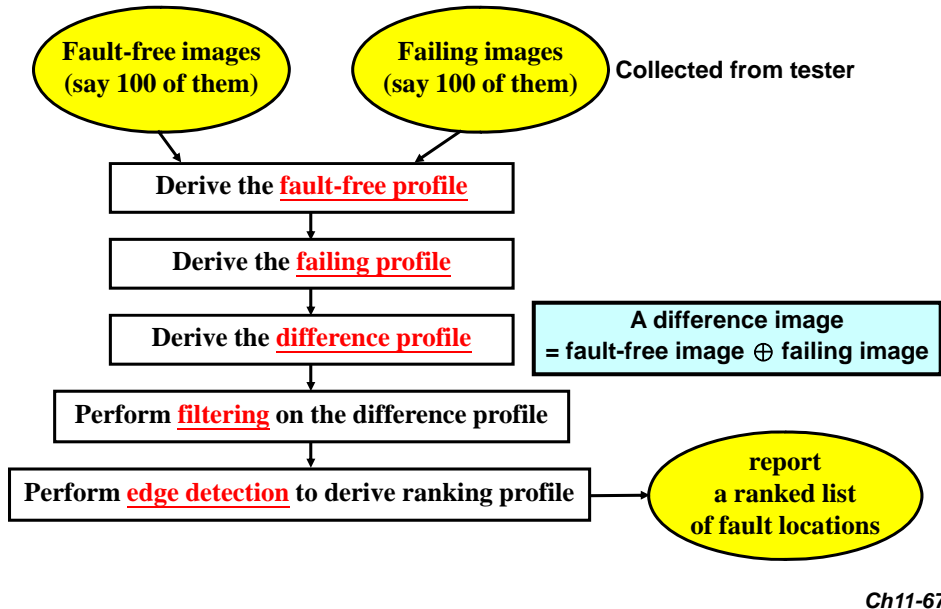
A **profile** is the **distribution of certain statistics of the flip-flops**.



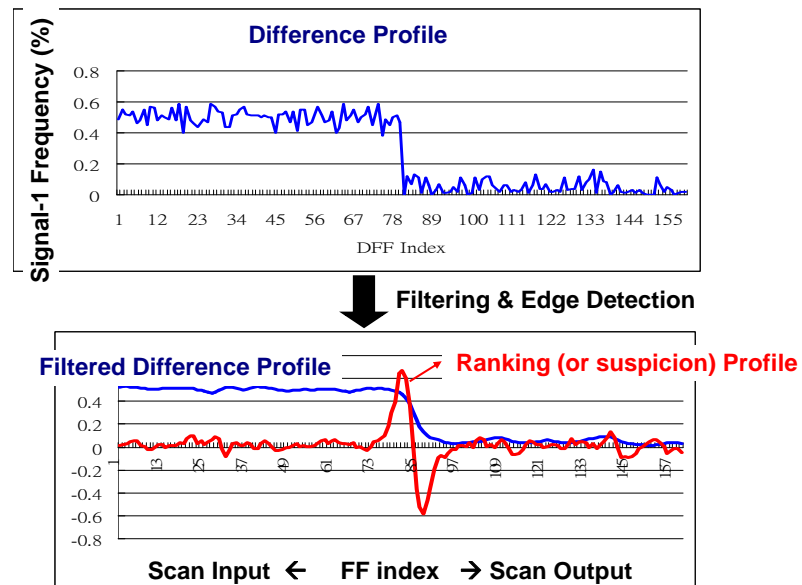
Comparing failing profile with the fault-free profile
→ Could reveal the fault location

Ch11-66

Profile Analysis



Example: Filtering & Edge Detection



Computation of Average-Sum Filtering

- (Average-sum filtering) Assume that the difference profile is given and denoted as $D[i]$, where i is the index of a flip-flop. We use the following formula to compute a smoothed difference profile, $SD[i]$:

$$SD[i] = 0.2 * (D[i-2] + D[i-1] + D[i] + D[i+1] + D[i+2])$$

Ch11-69

Computation of Edge Detection

- The true location of the faulty flip-flop is likely to be the *left-boundary of the transition region in the difference profile*. To detect this boundary, we can use a simply *edge detection formula* defined below.
- (Edge detection) On the smoothed difference profile $SD[i]$, the following formula can be used to compute the faulty frequency of each flip-flop as a suspicious profile.

$$suspicion[i] = [-1, -1, -1, 1, 1, 1] \cdot \begin{bmatrix} |SD[i] - SD[i-3]| \\ |SD[i] - SD[i-2]| \\ |SD[i] - SD[i-1]| \\ |SD[i] - SD[i+1]| \\ |SD[i] - SD[i+2]| \\ |SD[i] - SD[i+3]| \end{bmatrix}$$

Ch11-70

Summary of Scan Chain Diagnosis

- ❑ **Hardware Assisted**
 - Extra logic on the scan chain
 - Good for stuck-at fault
- ❑ **Fault Simulation Based**
 - To find a faulty circuit matching the syndromes [Kundu 1993] [Cheney 2000] [Stanley 2000]
 - Tightening heuristic → upper & lower bound [Guo 2001][Y. Huang 2005]
 - Use single-excitation pattern for better resolution [Li 2005]
- ❑ **Profiling-Based Method**
 - Locate the fault directly from the difference profiles obtained by run-and-scan test
 - Applicable to bridging faults
 - Use signal processing techniques such as filtering and edge detection

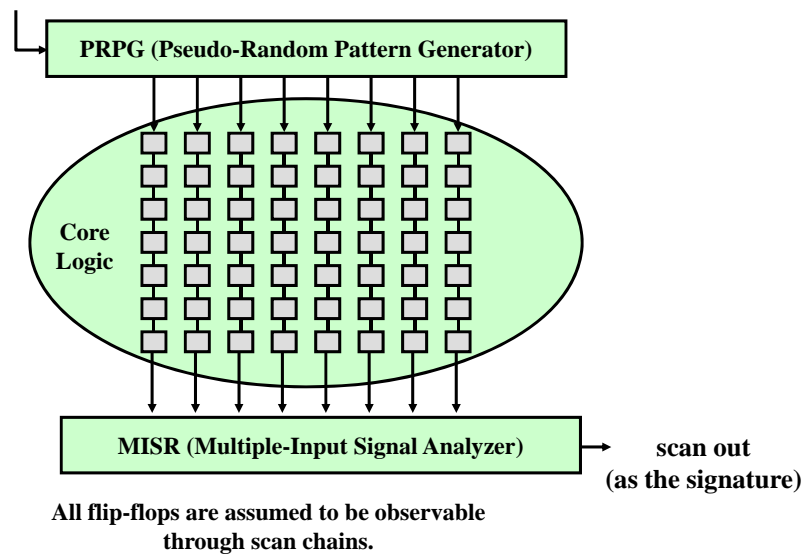
Ch11-71

Outline

- ❑ **Introduction**
- ❑ **Combinational Logic Diagnosis**
- ❑ **Scan Chain Diagnosis**
- ➡ ❑ **Logic BIST Diagnosis**
 - **Overview**
 - **Interval-Based Method**
 - **Masking-Based Method**
- ❑ **Conclusion**

Ch11-72

A Logic BIST Architecture



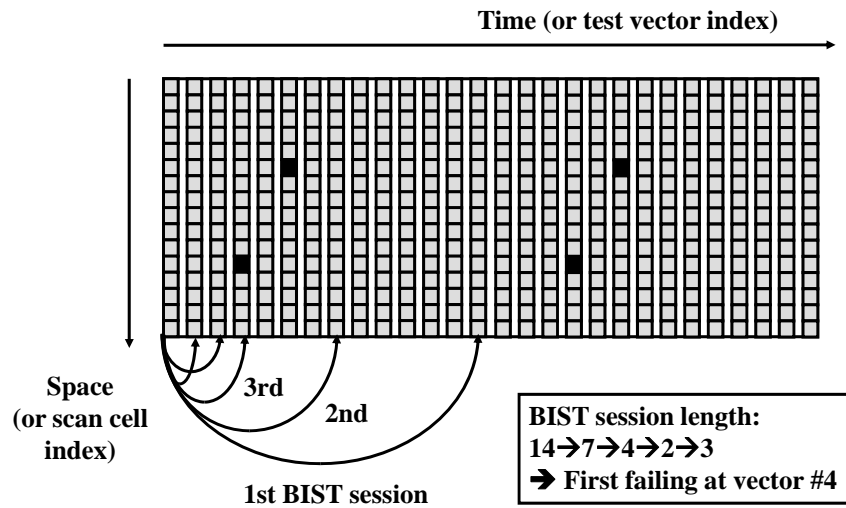
Ch11-73

Diagnosis for BISTed Logic

- **Diagnosis in a BIST environment requires**
 - determining from compacted output responses which test vectors have produced a faulty response (*time information*)
 - determining from compacted output responses which scan cells have captured errors (*space information*)
- **The true fault location inside the logic**
 - Can then be inferred from the above space and time information using previously discussed combinational logic diagnosis

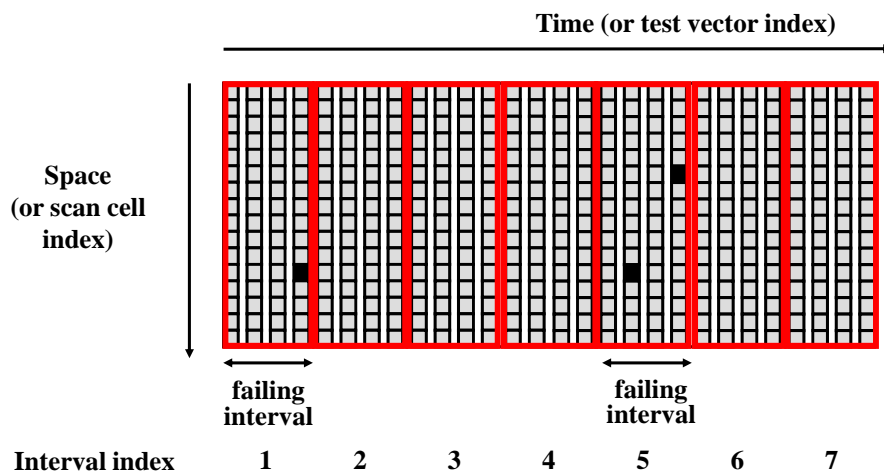
Ch11-74

Binary Search To Locate 1st Failing Vector



Ch11-75

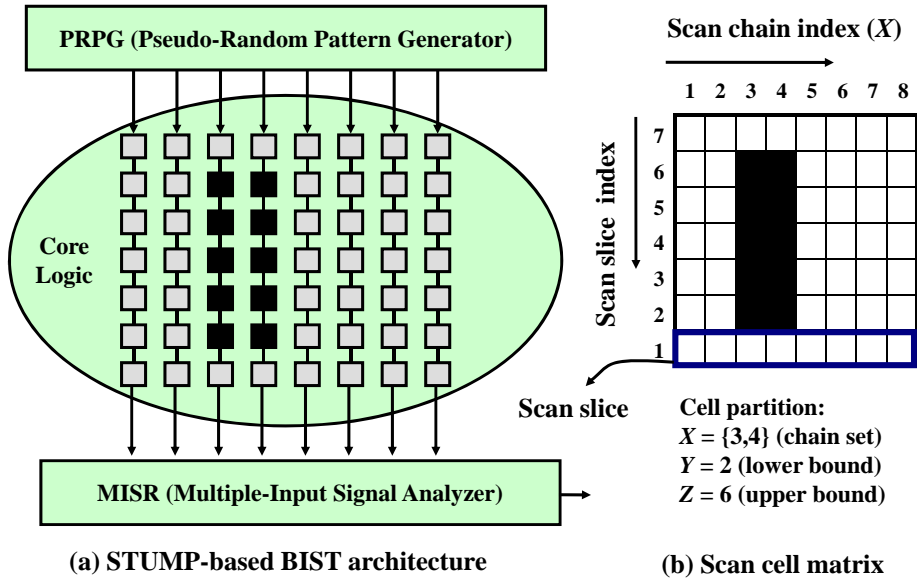
Interval Unloading-Based Diagnosis



A signature is scanned out to the tester
for comparison at the end of each interval

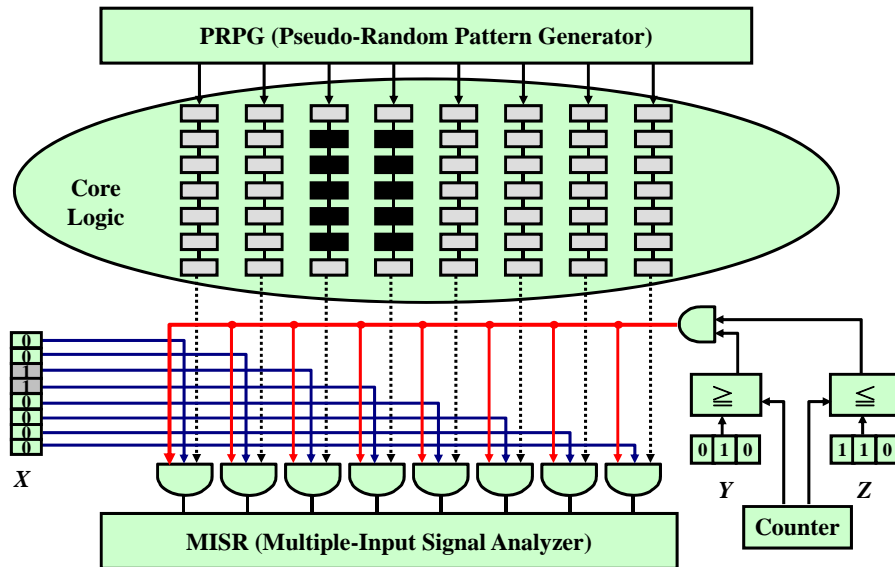
Ch11-76

Deterministic Masking-Based Diagnosis



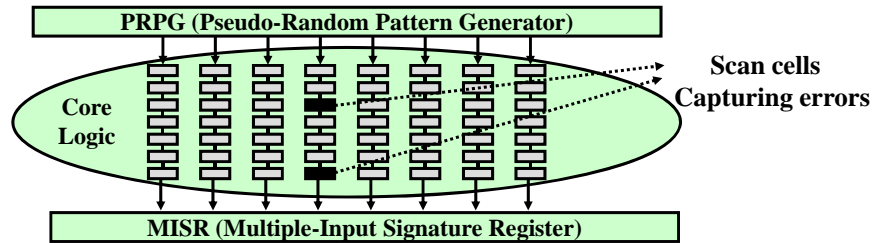
Ch11-77

Circuitry to Support Deterministic Masking

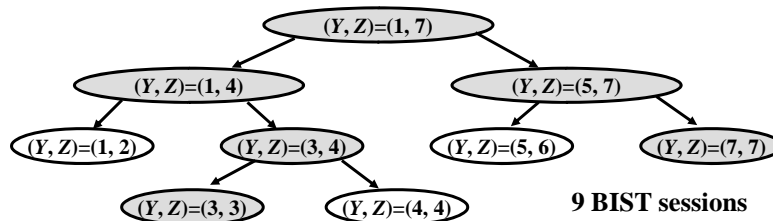


Ch11-78

A Search for Scan Cells Capturing Errors



(a) Scan cells capturing errors in the fourth scan chain



(b) The search tree

Ch11-79

Conclusions

- ❑ **Logic diagnosis for combinational logic**
 - Has been mature
 - Good for not just stuck-at faults, but also bridging faults
- ❑ **Scan chain diagnosis**
 - Making good progress ...
 - Fault-simulation-based, or signal-profiling based
- ❑ **Diagnosis of scan-based logic BIST**
 - Hardware support is often required
 - Interval-unloading, or masking-based
- ❑ **Future challenges**
 - Performance (speed) debug
 - Diagnosis for logic with on-chip test compression and decompression
 - Diagnosis for parametric yield loss due to nanometer effects

Ch11-80