

Introduction to Electronic Design Automation

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University



Spring 2012

1

Testing

Slides are by Courtesy of Prof. S.-Y. Huang and C.-M. Li

2

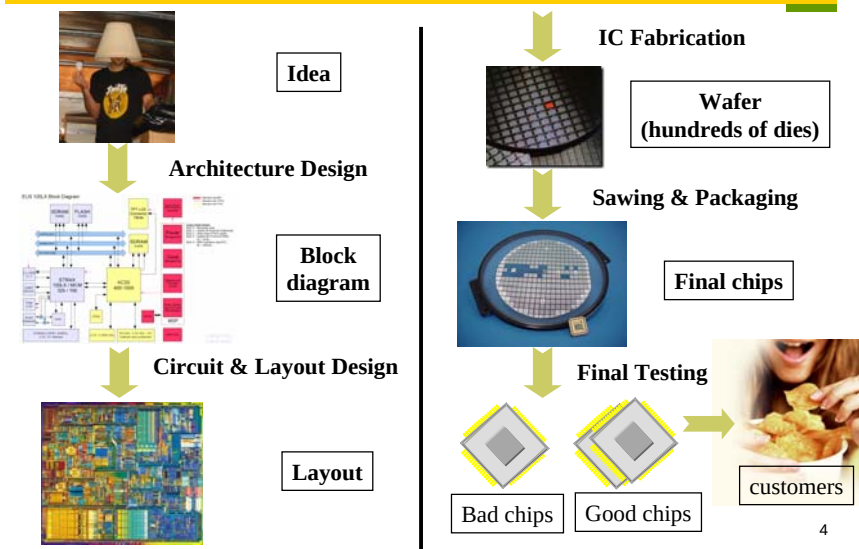
Testing

■ Recap

- Design verification
 - Is what I specified really what I wanted?
 - Property checking
- Implementation verification
 - Is what I implemented really what I specified?
 - Equivalence checking
- Manufacture verification
 - Is what I manufactured really what I implemented?
 - Testing; post manufacture verification
 - Quality control
 - Distinguish between good and bad chips

3

Design Flow



4

Manufacturing Defects

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

5

Faults, Errors and Failures

- Faults
 - A physical defect within a circuit or a system
 - May or may not cause a system failure
- Errors
 - Manifestation of a fault that results in incorrect circuit (system) outputs or states
 - Caused by faults
- Failures
 - Deviation of a circuit or system from its specified behavior
 - Fail to do what is supposed to do
 - Caused by errors
- Faults cause errors; errors cause failures

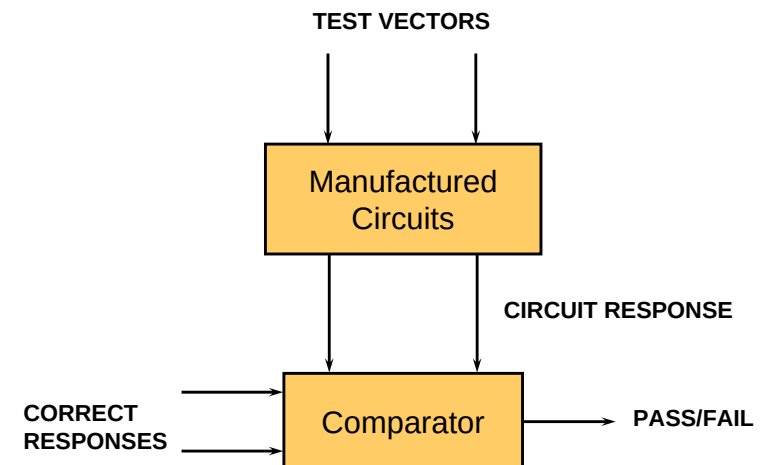
6

Testing and Diagnosis

- Testing
 - Exercise a system and analyze the response to ensure whether it behaves correctly **after manufacturing**
- Diagnosis
 - Locate the **causes** of misbehavior after the incorrectness is detected

7

Scenario of Manufacturing Test



8

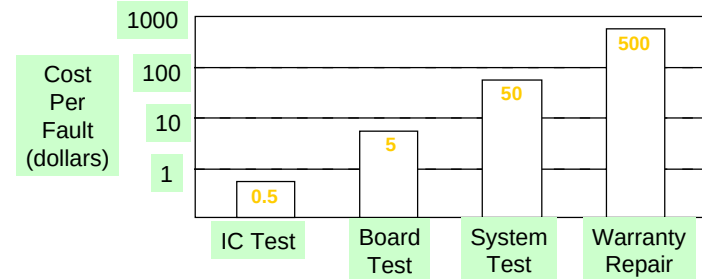
Test Systems



9

Purpose of Testing

- Verify manufactured circuits
 - Improve system reliability
 - Reduce repair costs
 - Repair cost goes up by an order of magnitude each step away from the fab. line

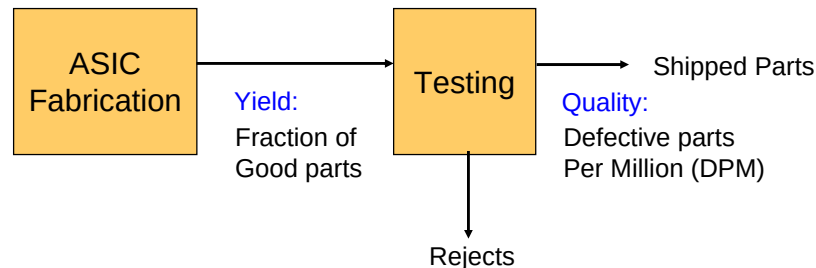


B. Davis, "The Economics of Automatic Testing" McGraw-Hill 1982

10

Testing and Quality

- Quality of shipped part can be expressed as a function of the yield Y and test (fault) coverage T.



11

Fault Coverage

- Fault coverage T
 - Measure of the ability of a test set to detect a given set of faults that may occur on the Design Under Test (DUT)

$$T = \frac{\text{\# detected faults}}{\text{\# all possible faults}}$$

12

Defect Level

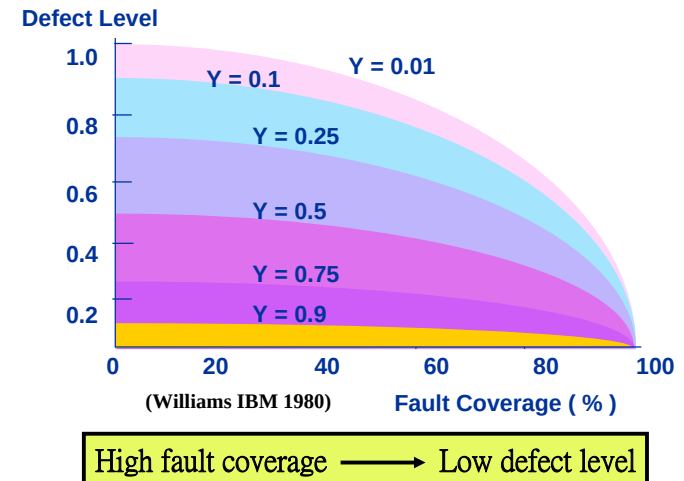
- A defect level is the fraction of the shipped parts that are defective

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

Y: yield
T: fault coverage

13

Defect Level vs. Fault Coverage



14

DPM vs. Yield and Coverage

Yield	Fault Coverage	DPM
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

15

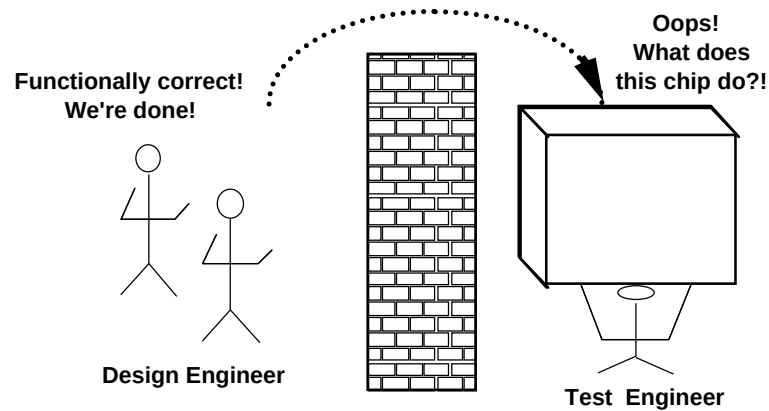
Why Testing Is Difficult ?

- Test time explodes exponentially in exhaustive testing of VLSI
 - For a combinational circuit with 50 inputs, need $2^{50} = 1.126 \times 10^{15}$ test patterns.
 - Assume one test per 10^{-7} sec, it takes 1.125×10^8 sec = 3.57years.
 - 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

16

The Infamous Design/Test Wall

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



17

Outline

- ❑ Fault Modeling
- ❑ Fault Simulation
- ❑ Automatic Test Pattern Generation
- ❑ Design for Testability

18

Functional vs. Structural Testing

- ❑ I/O functional testing is inadequate for manufacturing
 - Need fault models
- ❑ Exhaustive testing is daunting
 - Need abstraction and smart algorithms
 - Structural testing is more effective

19

Why Fault Model ?

- ❑ Fault model identifies target faults
 - Model faults that are most likely to occur
- ❑ Fault model limits the scope of test generation
 - Create tests only for the modeled faults
- ❑ Fault model makes testing effective
 - Fault coverage can be computed for specific test patterns to measure its effectiveness
- ❑ Fault model makes analysis possible
 - Associate specific defects with specific test patterns

20

Fault Modeling vs. Physical Defects

□ 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

21

Fault Modeling vs. Physical Defects (cont'd)

□ Electrical effects

- Shorts (bridging faults)
- Opens
- Transistor stuck-on/open
- Resistive shorts/opens
- Change in threshold voltages

□ Logical effects

- Logical stuck-at-0/1
- Slower transition (delay faults)
- AND-bridging, OR-bridging

22

Typical Fault Types

□ Stuck-at faults

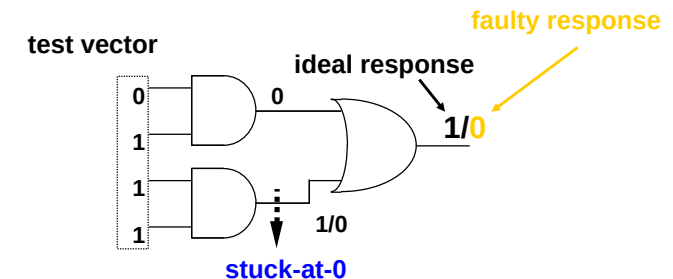
- Bridging faults
- Transistor stuck-on/open faults
- Delay faults
- IDDQ faults
- State transition faults (for FSM)
- Memory faults
- PLA faults

23

Single Stuck-At Fault

□ Assumptions:

- Only one wire is faulty
- Fault can be at an input or output of a gate
- Faulty wire permanently sticks at 0 or 1



24

Multiple Stuck-At Faults

- Several stuck-at faults occur at the same time
 - Common in high density circuits
- 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

25

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 design
- Detection capability of un-modeled defects
 - Empirically, many un-modeled defects can also be detected accidentally under the single stuck-at fault model
- Cover a large percentage of multiple stuck-at faults

26

Why Logical Fault Modeling ?

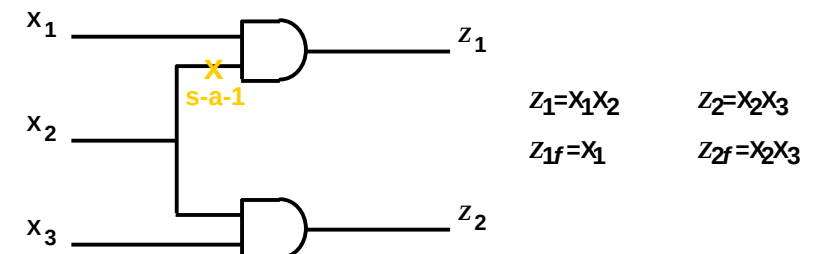
- Fault analysis on logic rather than physical problem
 - Complexity is reduced
- Technology independent
 - Same fault model is applicable to many technologies
 - Testing and diagnosis methods remain valid despite changes in technology
- Wide applications
 - The derived tests may be used for physical faults whose effect on circuit behavior is not completely understood or too complex to be analyzed
- Popularity
 - Stuck-at fault is the most popular logical fault model

27

Definition of Fault Detection

- A test (vector) t detects a fault f iff t detects f (i.e. $z(t) \neq z_f(t)$)

Example



Test $(x_1, x_2, x_3) = (100)$ detects f because $z_1(100) = 0$ and $z_{1f}(100) = 1$

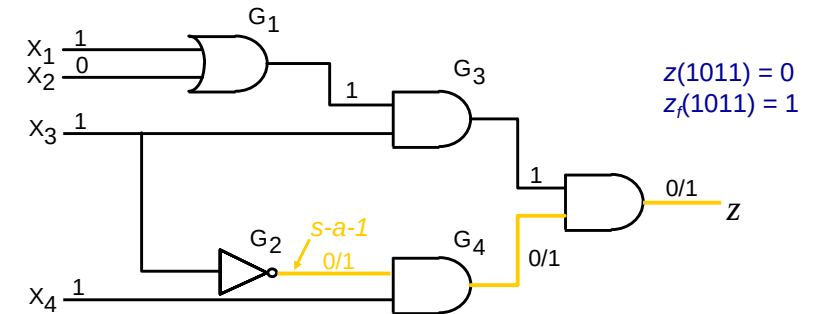
28

Fault Detection Requirement

- A test t that detects a fault f
 - **activates** f (or generate a fault effect) by creating different v and v_f values at the site of the fault
 - **propagates** the error to a primary output z by making all the wires along at least one path between the fault site and z have different v and v_f values
- Sensitized wire
 - A wire whose value in response to the test changes in the presence of the fault f is said to be **sensitized by the test** in the faulty circuit
- Sensitized path
 - A path composed of sensitized wires is called a **sensitized path**

29

Fault Sensitization



30

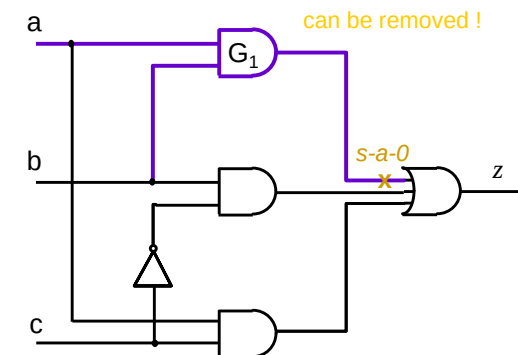
Detectability

- 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 some primary output

31

Undetectable Fault

- The stuck-at-0 fault at G_1 output is undetectable
 - Undetectable faults do not change the function of the circuit
 - The related circuit can be deleted to simplify the circuit



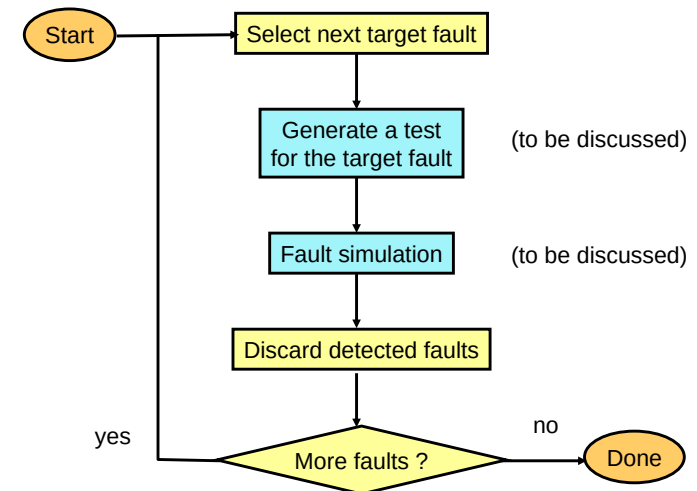
32

Test Set

- Complete detection test set
 - A set of tests that detects any detectable fault in a designated set of faults
- Quality of a test set
 - is measured by fault coverage
- Fault coverage
 - Fraction of the faults detected by a test set
 - can be determined by **fault simulation**
 - >95% is typically required under the single stuck-at fault model
 - >99.9% required in the ICs manufactured by IBM

33

Typical Test Generation Flow



34

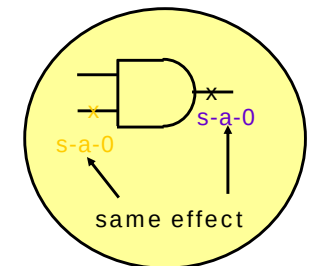
Fault Equivalence

- Distinguishing test
 - A test t distinguishes faults α and β if $z_\alpha(t) \neq z_\beta(t)$ for some PO function z
- Equivalent faults
 - Two faults α and β are said to be equivalent in a circuit iff the function under α is equal to the function under β for every input assignment (sequence) of the circuit.
 - That is, no test can distinguish α and β , i.e., **$\text{test-set}(\alpha) = \text{test-set}(\beta)$**

35

Fault Equivalence

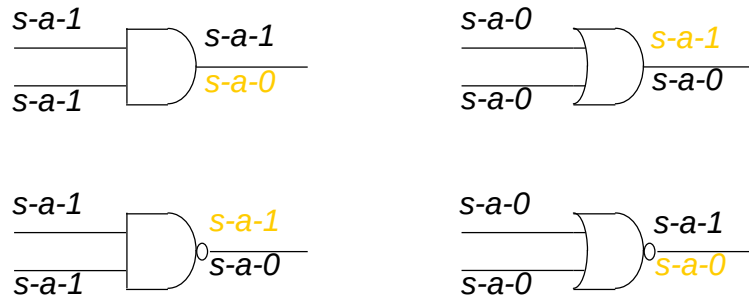
- AND gate:
 - all s-a-0 faults are equivalent
- OR gate:
 - all s-a-1 faults are equivalent
- NAND gate:
 - all the input s-a-0 faults and the output s-a-1 faults are equivalent
- NOR gate:
 - all input s-a-1 faults and the output s-a-0 faults are equivalent
- Inverter:
 - input s-a-1 and output s-a-0 are equivalent
 - input s-a-0 and output s-a-1 are equivalent



36

Equivalence Fault Collapsing

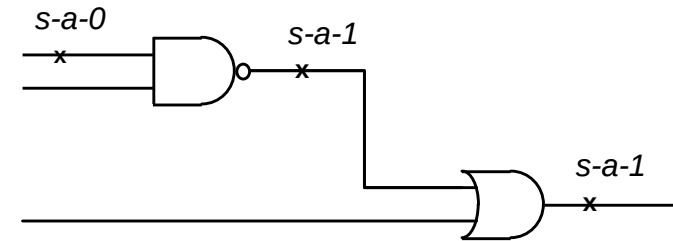
- $n+2$, instead of $2(n+1)$, single stuck-at faults need to be considered for n -input AND (or OR) gates



37

Equivalent Fault Group

- In a combinational circuit
 - Many faults may form an equivalence group
 - These equivalent faults can be found in a reversed topological order from POs to PIs

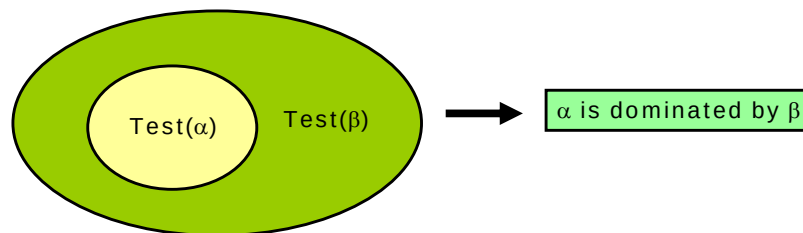


Three faults shown are equivalent !

38

Fault Dominance

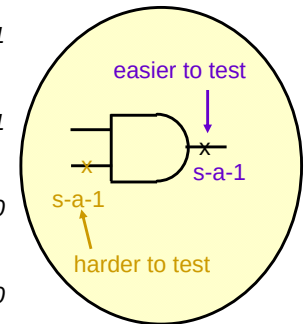
- Dominance relation
 - A fault β is said to *dominate* another fault α in an irredundant circuit iff every test (sequence) for α is also a test (sequence) for β , i.e., $\text{test-set}(\alpha) \subseteq \text{test-set}(\beta)$
 - No need to consider fault β for fault detection



39

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
 - Reducing the set of faults to be analyzed based on the dominance relation



40

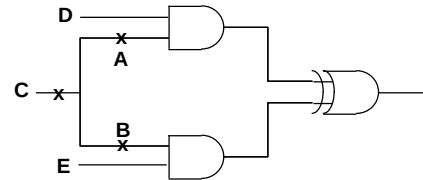
Stem vs. Branch Faults

- Detect A s-a-1:
 $z(t) \oplus z_f(t) = (CD \oplus CE) \oplus (D \oplus CE)$
 $= D \oplus CD \Rightarrow (C=0, D=1)$

- Detect C s-a-1:
 $z(t) \oplus z_f(t) = (CD \oplus CE) \oplus (D \oplus E)$
 $\Rightarrow (C=0, D=1, E=0) \text{ or } (C=0, D=0, E=1)$

- Hence, C s-a-1 does not dominate A s-a-1

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

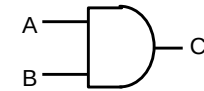


C: **stem** of a multiple fanout
A, B: **branches**

41

Analysis of a Single Gate

- 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

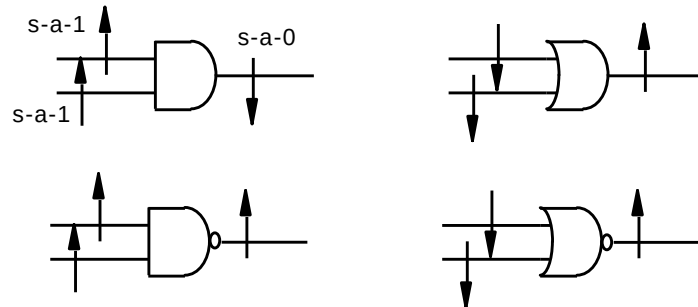


AB	C	A	B	C	A	B	C
		sa1	sa1	sa1	sa0	sa0	sa0
00	0			1			
01	0	1		1			
10	0		1	1			
11	1				0	0	0

42

Fault Collapsing

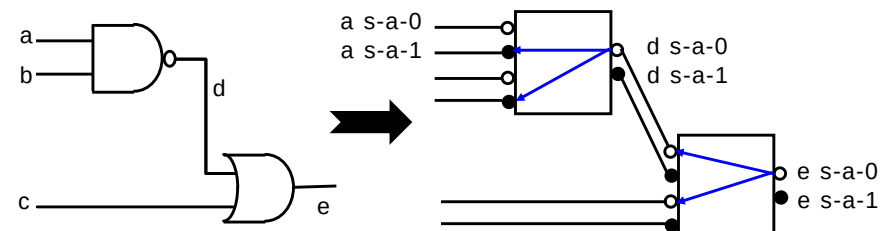
- Collapse faults by fault equivalence and dominance
 - For an n -input gate, we only need to consider $n+1$ faults in test generation



43

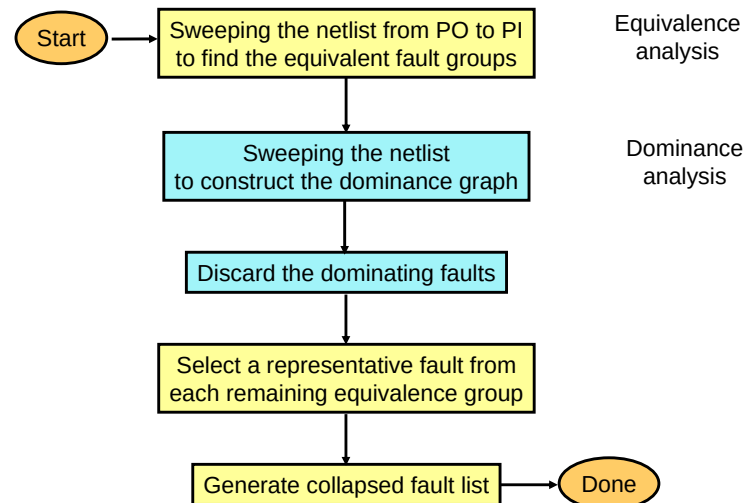
Dominance Graph

- 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



44

Fault Collapsing Flow



45

Prime Fault

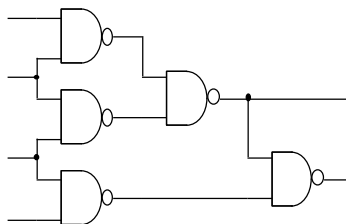
- α is a **prime fault** if every fault that is dominated by α is also equivalent to α
- **Representative Set of Prime Fault (RSPF)**
 - A set that consists of exactly one prime fault from each equivalence class of prime faults
 - True minimal RSPF is difficult to find

46

Why Fault Collapsing ?

- Save memory and CPU time
- Ease testing generation and fault simulation

Exercise

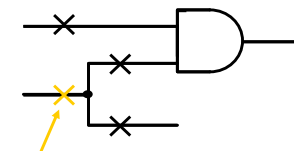


* 30 total faults → 12 prime faults

47

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 (i.e., every gate has only one fanout), primary inputs are the checkpoints



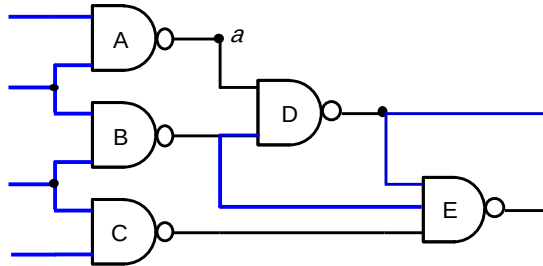
Stem is not a checkpoint !

48

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

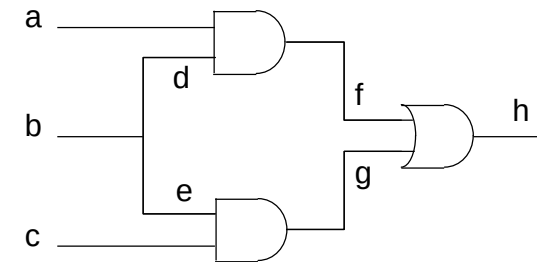


49

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 faults are enough



50

Outline

- Fault Modeling
- Fault Simulation
- Automatic Test Pattern Generation
- Design for Testability

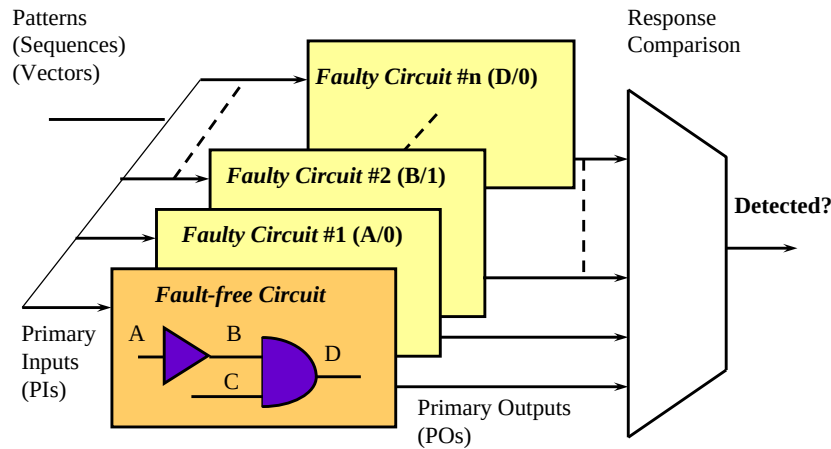
51

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 that are accidentally detected by a particular vector
- To construct fault-dictionary
 - For post-testing diagnosis

52

Conceptual Fault Simulation



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

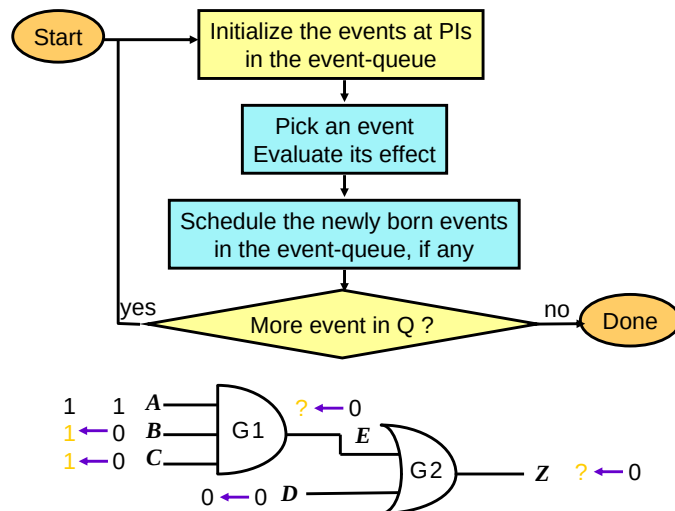
53

Some Basics for Logic Simulation

- In fault simulation, our main concern is functional faults; gate delays are assumed to be **zero** unless delay faults are considered
- Logic values can be either $\{0, 1\}$ (for two-value simulation) or $\{0, 1, X\}$ (for three-value simulation)
- Two simulation mechanisms:
 - **Compiled-code valuation:**
 - A circuit is translated into a program and **all** gates are executed for each pattern (may have redundant computation)
 - **Event-driven valuation:**
 - Simulating a vector is viewed as a sequence of **value-change events** propagating from PIs to POs
 - Only those logic gates affected by the events are **re-evaluated**

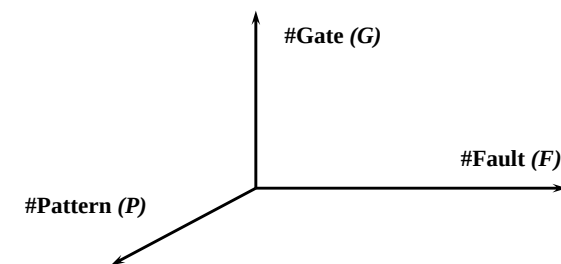
54

Event-Driven Simulation



55

Complexity of Fault Simulation

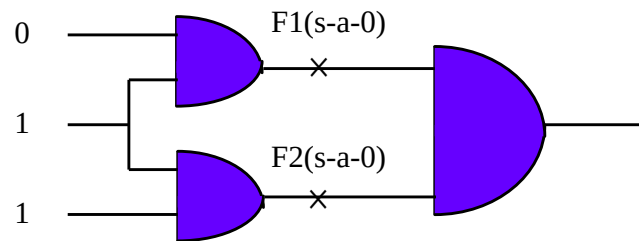


- Complexity $\sim F \cdot P \cdot G \sim O(G^3)$
- The complexity is higher than logic simulation by a factor of F , while it is usually much lower than ATPG
- The complexity can be greatly reduced using
 - fault collapsing and other advanced techniques

56

Characteristics of Fault Simulation

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



57

Fault Simulation Techniques

- ❑ Parallel Fault Simulation
- ❑ Deductive Fault Simulation

58

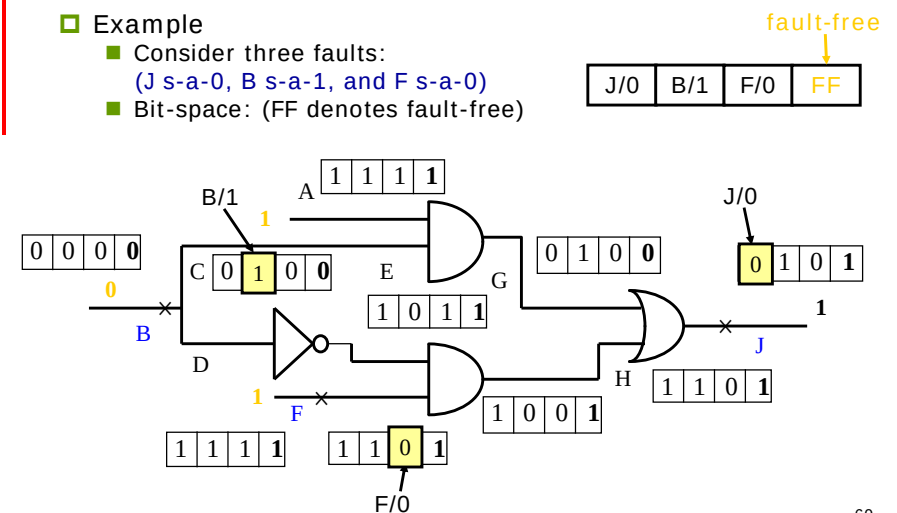
Parallel Fault Simulation

- ❑ Simulate multiple circuits simultaneously
 - 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
- ❑ Complication
 - An event or a value change of a single faulty or fault-free circuit leads to the computation of an entire word
 - The fault-free logic simulation is repeated for each pass

59

Parallel Fault Simulation

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



60

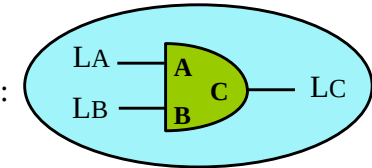
Deductive Fault Simulation

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

61

Illustration of Fault List Propagation

Consider a two-input AND-gate:



Non-controlling case:

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

Controlling cases:

Case 2: $A=1, B=0, C=0$ at fault-free,
 $LC = (\overline{LA} \cap LB) \cup \{C/1\}$

Case 3: $A=0, B=0, C=0$ at fault-free,
 $LC = (LA \cap LB) \cup \{C/1\}$

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

62

Rule of Fault List Propagation

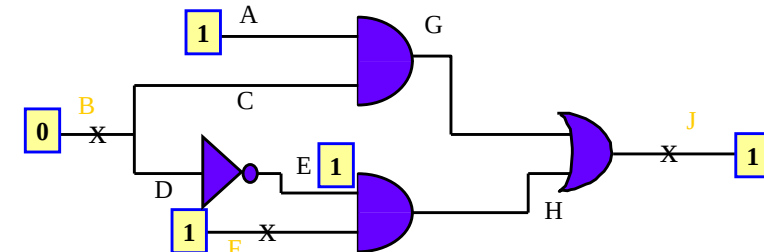
	<i>a</i>	<i>b</i>	<i>z</i>	Output fault list
AND	0	0	0	$\{L_a \cap L_b\} \cup Z_1$
	0	1	0	$\{L_a - L_b\} \cup Z_1$
	1	0	0	$\{L_b - L_a\} \cup Z_1$
	1	1	1	$\{L_a \cup L_b\} \cup Z_0$
OR	0	0	0	$\{L_a \cup L_b\} \cup Z_1$
	0	1	1	$\{L_b - L_a\} \cup Z_0$
	1	0	1	$\{L_a - L_b\} \cup Z_0$
	1	1	1	$\{L_a \cap L_b\} \cup Z_0$
NOT	0		1	$L_a \cup Z_0$
	1		0	$L_a \cup Z_1$

63

Deductive Fault Simulation

□ Example (1/4)

- Consider 3 faults: $B/1$, $F/0$, and $J/0$ under $(A,B,F) = (1,0,1)$



Fault list at PIs:

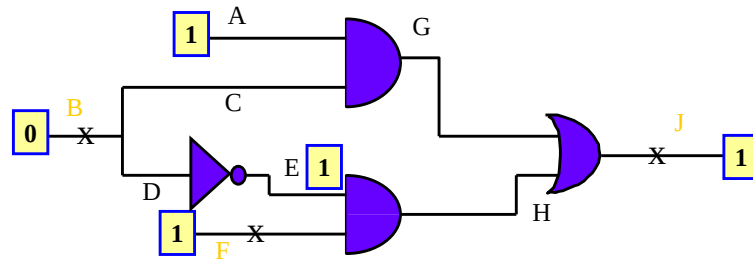
$LB = \{B/1\}$, $LF = \{F/0\}$, $LA = \emptyset$, $LC=LD = \{B/1\}$

64

Deductive Fault Simulation

Example (2/4)

- Consider 3 faults: B/1, F/0, and J/0 under (A,B,F) = (1,0,1)



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

Fault lists at G and E:

$LG = (LA \cap LC) \cup G/1 = \{B/1, G/1\}$

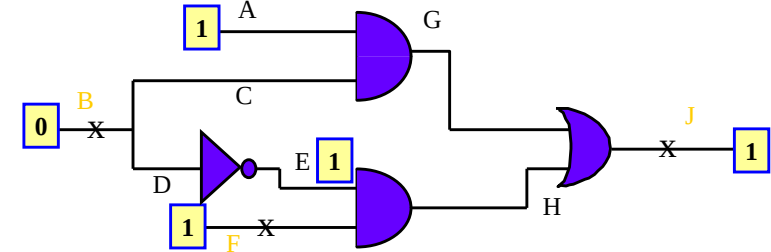
$LE = (LD) \cup E/0 = \{B/1, E/0\}$

65

Deductive Fault Simulation

Example (3/4)

- Consider 3 faults: B/1, F/0, and J/0 under (A,B,F) = (1,0,1)



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

$LG = \{B/1, G/1\}$, $LE = \{B/1, E/0\}$

Fault list at H:

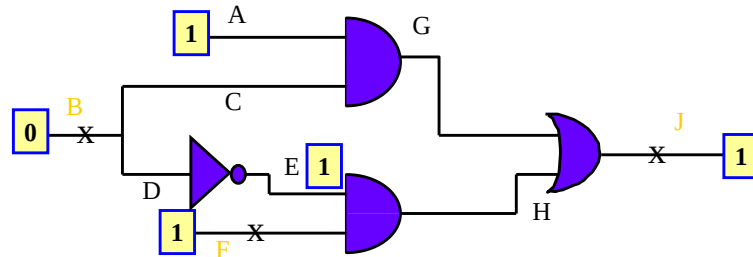
$LH = (LE \cup LF) \cup H/0 = \{B/1, E/0, F/0, H/0\}$

66

Deductive Fault Simulation

Example (4/4)

- Consider 3 faults: B/1, F/0, and J/0 under (A,B,F) = (1,0,1)



$LB = \{B/1\}$, $LF = \{F/0\}$, $LA = \emptyset$, $LC = LD = \{B/1\}$, $LG = \{B/1, G/1\}$, $LE = \{B/1, E/0\}$, $LH = \{B/1, E/0, F/0, H/0\}$

Final fault list at PO J:

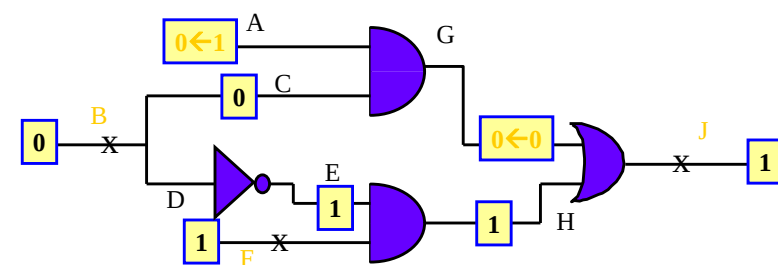
$LJ = (LH - LG) \cup J/0 = \{E/0, F/0, J/0\}$

67

Deductive Fault Simulation

Example (cont'd)

- Consider 3 faults: B/1, F/0, and J/0 under (A,B,F) = (0,0,1)



Event driven updates:

$LB = \{B/1\}$, $LF = \{F/0\}$, $LA = \emptyset$, $LC = LD = LE = \{B/1\}$,

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

68

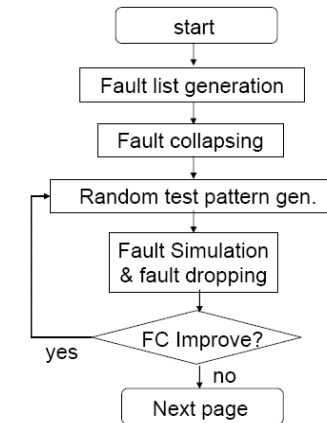
Outline

- Fault Modeling
- Fault Simulation
- Automatic Test Pattern Generation (ATPG)
 - Functional approach
 - Boolean difference
 - Structural approach
 - D-algorithm
 - PODEM
- Design for Testability

69

Typical ATPG Flow

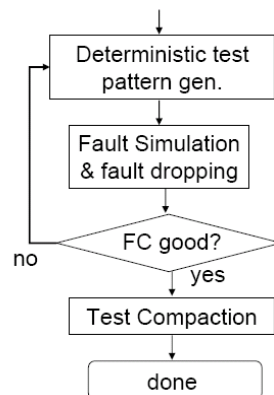
- 1st phase: random test pattern generation



70

Typical ATPG Flow (cont'd)

- 2nd phase: deterministic test pattern generation



71

Test Pattern Generation

- The test set T of a fault α with respect to some PO z can be computed by

$$T(x) = z(x) \oplus z_{\alpha}(x)$$

- A test pattern can be fully specified or partially specified depending on whether the values of PIs are all assigned
 - Example

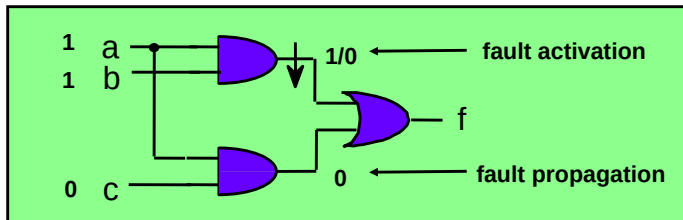
abc	z	z_{α}
000	0	0
001	0	0
010	0	0
011	0	0
100	0	0
101	1	1
110	1	0
111	1	0

Input vectors (1,1,0) and (1,1,-) are fully and partially specified test patterns of fault α , respectively.

72

Structural Test Generation D-Algorithm

- Test generation 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)

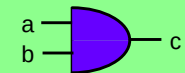


73

Structural Test Generation D-Algorithm

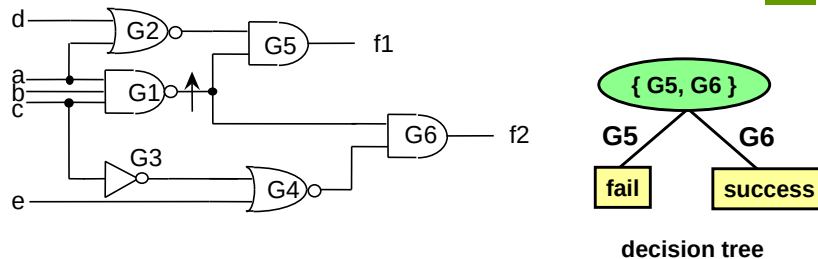
- Fault activation
 - Setting the faulty signal to either 0 or 1 is a Line Justification problem
- Fault propagation
 1. select a path to a PO → decisions
 2. once the path is selected → a set of line justification (LJ) problems are to be solved
- Line justification
 - Involves decisions or implications
 - Incorrect decisions: need backtracking

To justify c=1 → a=1 and b=1 (implication)
To justify c=0 → a=0 or b=0 (decision)



74

Structural Test Generation D-Algorithm: Fault Propagation

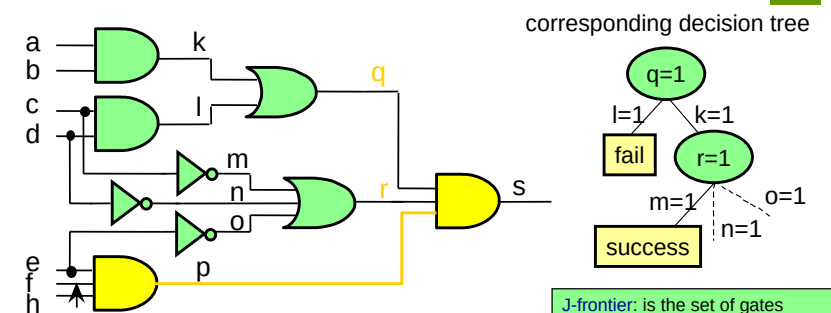


- Fault activation
 - G1=0 → { a=1, b=1, c=1 } → { G3=0 }
- Fault propagation: through G5 or G6
- Decision through G5:
 - G2=1 → { d=0, a=0 } → inconsistency at a → backtrack !!
- Decision through G6:
 - → G4=1 → e=0 → done !! The resulting test is (111x0)

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 }.

75

Structural Test Generation D-Algorithm: 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: 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)

J-frontier: is the set of gates whose output value is known (i.e., 0 or 1), but is not implied by its input values.
Ex: initially, J-frontier is {q=1, r=1}

76

Test Generation

- A branch-and-bound search
- Every decision point is a **branching** point
- If a set of decisions lead to a **conflict**, a **backtrack** is taken to explore other decisions
- A **test is found** when
 1. fault effect is propagated to a PO, and
 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.

77

Implication

□ Implication

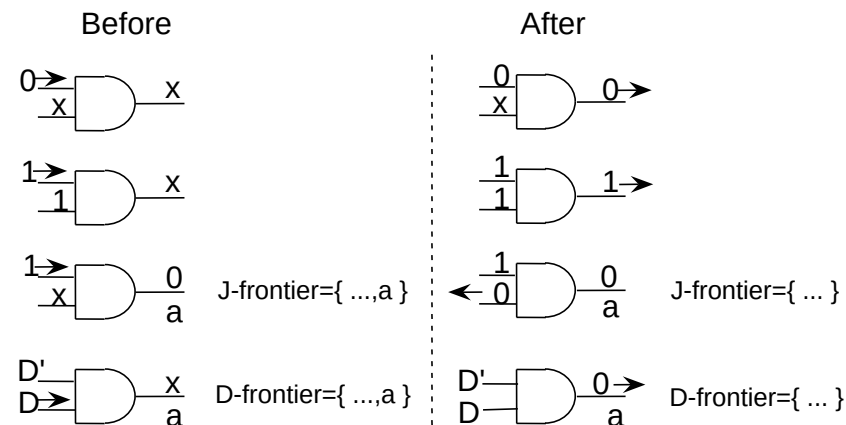
- Compute the values that can be uniquely determined
 - **Local implication**: propagation of values from one line to its immediate successors or predecessors
 - **Global implication**: the propagation involving a larger area of the circuit and re-convergent fanout

□ 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**

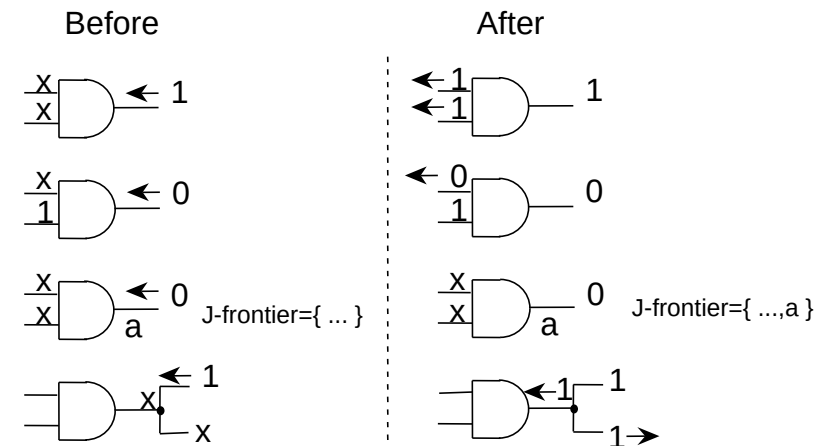
78

Forward Implication



79

Backward Implication

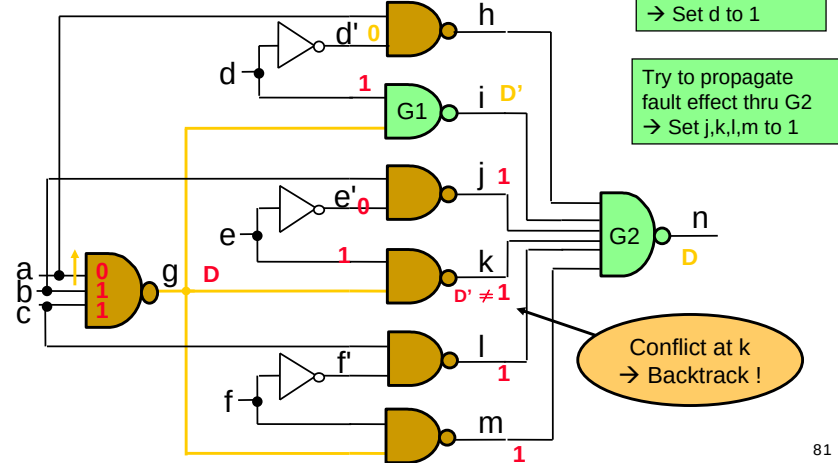


80

D-Algorithm (1/4)

Example

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

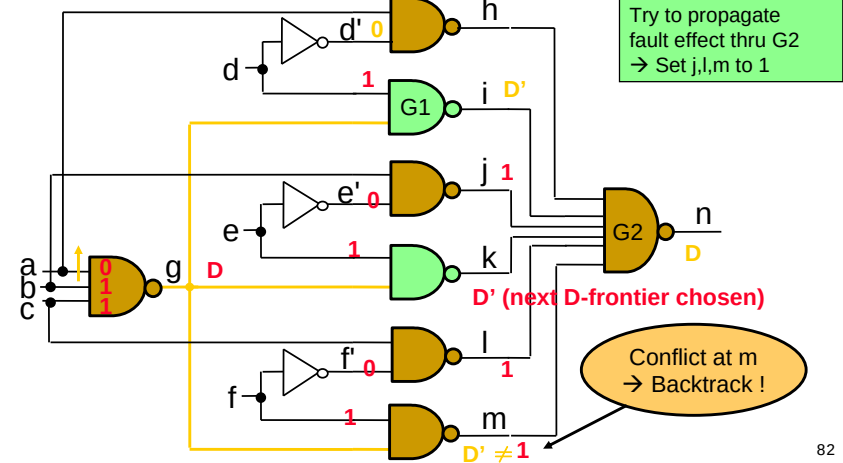


81

D-Algorithm (2/4)

Example

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

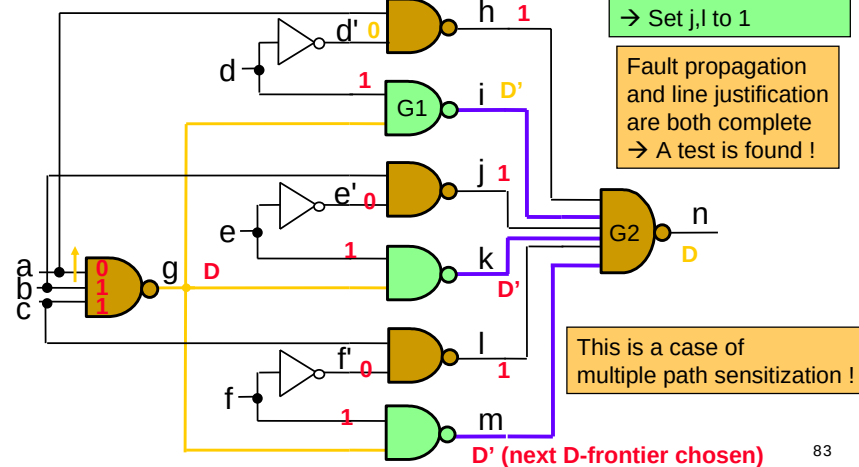


82

D-Algorithm (3/4)

Example

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



83

D-Algorithm (4/4)

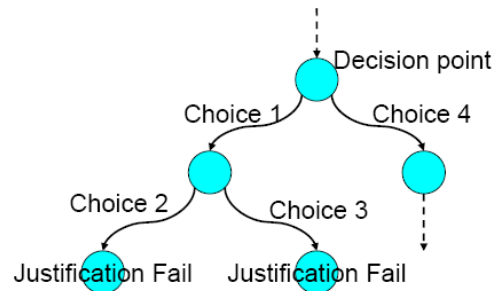
Decision	Implication	Comments
	a=0 h=1 b=1 c=1 g=D	Active the fault Unique D-drive
d=1	i=D' d'=0	Propagate via i
j=1 k=1 l=1 m=1	n=D e'=0 e=1 k=D'	Propagate via n Contradiction

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

84

Decision Tree on D-Frontier

- The decision tree
 - Node → D-frontier
 - Branch → decision taken
 - A **Depth-First-Search** (DFS) strategy is often used



85

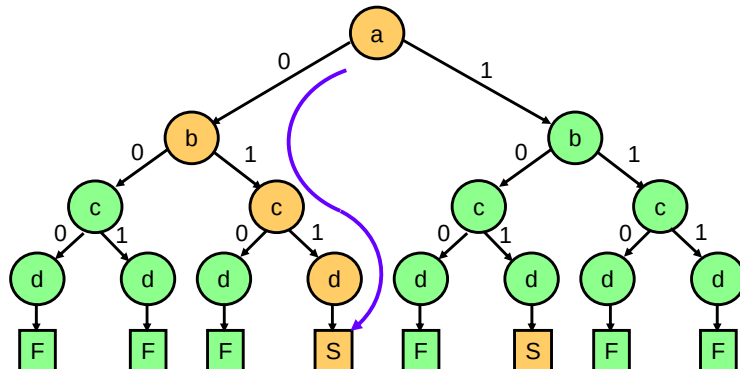
PODEM Algorithm

- 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**

86

PODEM Algorithm Search Space of PODEM

- Complete search space
 - A binary tree with 2^n leaf nodes, where n is the number of PIs
- Fast test generation
 - Need to find a path leading to a SUCCESS terminal quickly



87

PODEM Algorithm 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 PIs**
- 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 PIs
 - The result is a **PI signal-value pair** (x, v_x)
 - **No signal value is actually assigned during backtrace (toward PI) !**

88

PODEM Algorithm Objective

Objective routine involves

- selection of a **D-frontier**, G
- 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' );
  else {
    select a gate (G) from the D-frontier;
    select an input (j) of G with value x;
    c = controlling value of G;
    obj = (j, c');
  }
  return (obj);
}
```

← fault activation

← fault propagation

89

PODEM Algorithm Backtrace

Backtrace routine involves

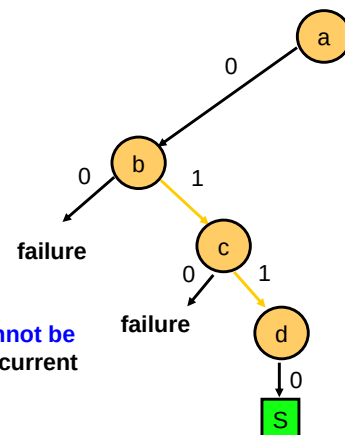
- finding an all-x path from objective site to a PI, i.e., every signal in this path has value x

```
Backtrace(w, v_w) {
  /* Maps objective into a PI assignment */
  G = w; /* objective node */
  v = v_w; /* 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);
}
```

90

PODEM Algorithm PI Assignment

PIs: { 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

91

PODEM Algorithm

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

begin

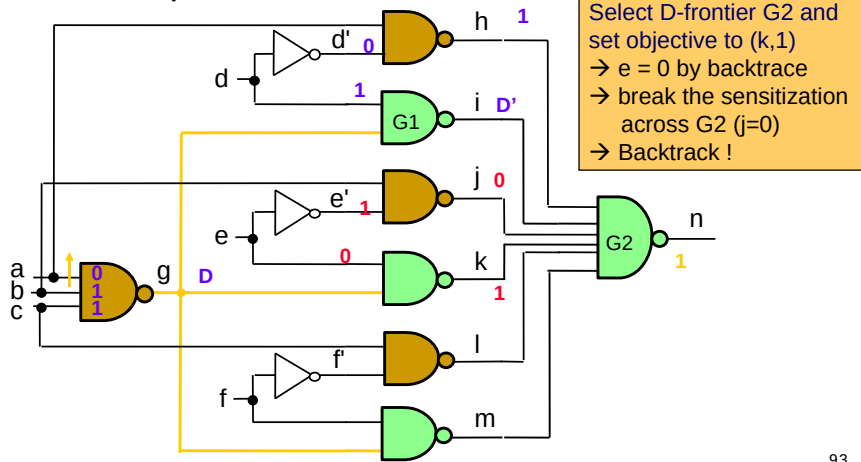
```
If(error at PO) return(SUCCESS);
If(test not possible) return(FAILURE);
(k, v_k) = Objective(); /* choose a line to be justified */
(j, v_j) = Backtrace(k, v_k); /* choose the PI to be assigned */
Imply (j, v_j); /* make a decision */
If ( PODEM() == SUCCESS ) return (SUCCESS);
Imply (j, v_j'); /* reverse decision */
If ( PODEM() == SUCCESS ) return(SUCCESS);
Imply (j, x);
Return (FAILURE);
```

end

92

PODEM Algorithm (1/4)

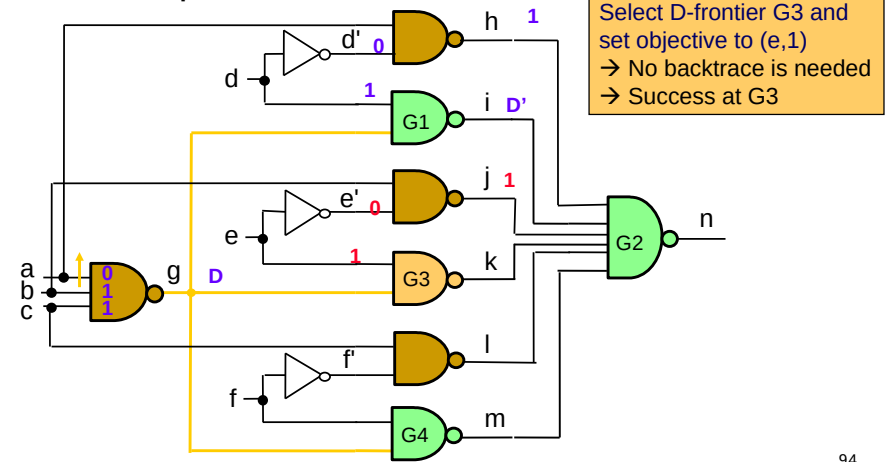
Example



93

PODEM Algorithm (2/4)

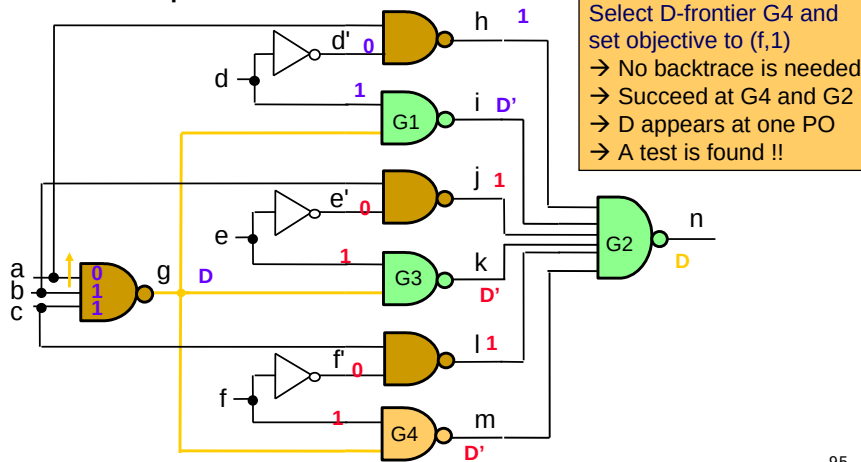
Example



94

PODEM Algorithm (3/4)

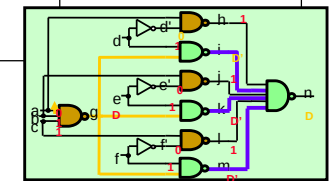
Example



95

PODEM Algorithm (4/4)

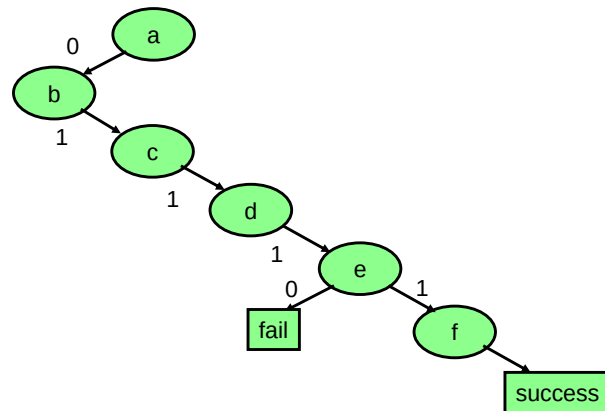
Objective	PI assignment	Implications	D-frontier	Comments
a=0	a=0	h=1	g	
b=1	b=1		g	
c=1	c=1	g=D	i,k,m	
d=1	d=1	d'=0		
		i=D'	k,m,n	
k=1	e=0	e'=1 j=0 k=1 n=1		Assignments need to be reversed during backtracking
	e=1	e'=0 j=1 k=D'	m	no solutions! → backtrack flip PI assignment
l=1	f=1	f'=0 l=1 m=D'	m,n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



96

PODEM Algorithm Decision Tree

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



97

Termination 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**

98

PODEM Overview

- **PODEM**
 - examines all possible input patterns **implicitly** but **exhaustively** (branch-and-bound) for finding a test
 - **complete** like D-algorithm (i.e., will find a test if 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**
 - Experiments show that PODEM is **generally faster** than D-algorithm

99

Outline

- **Fault Modeling**
- **Fault Simulation**
- **Automatic Test Pattern Generation**
- **Design for Testability**

100

Why DFT ?

- ❑ Direct testing is way too difficult !
 - Large number of FFs
 - Embedded memory blocks
 - Embedded analog blocks

101

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, full and partial scan, built-in self-test (BIST), boundary scan
- ❑ Cost of DFT
 - Pin count, area, performance, design-time, test-time, etc.

102

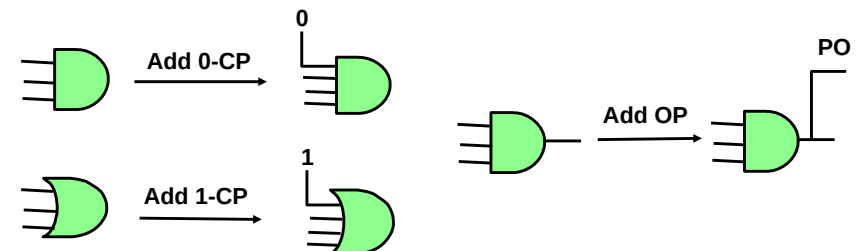
Important Factors

- ❑ Controllability
 - Measure the ease of controlling a line
- ❑ Observability
 - Measure the ease of observing a line at PO
- ❑ DFT deals with ways of improving
 - Controllability and observability

103

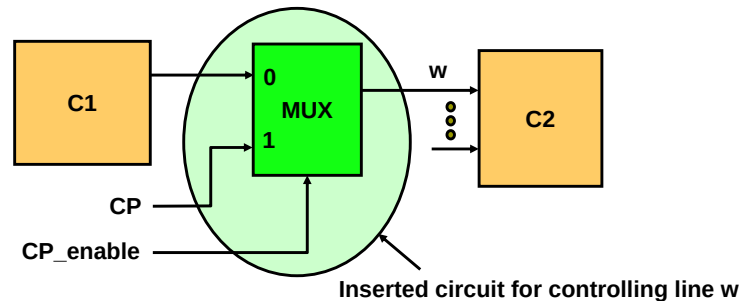
Test Point Insertion

- ❑ Employ test points to enhance **controllability** and **observability**
 - CP: Control Points
 - ❑ Primary inputs used to enhance controllability
 - OP: Observability Points
 - ❑ Primary outputs used to enhance observability



104

Control Point Insertion



- Normal operation:
When CP_enable = 0
- Inject 0:
Set CP_enable = 1 and CP = 0
- Inject 1:
Set CP_enable = 1 and CP = 1

105

Control Point Selection

□ Goal

- **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 preset/clear signals of flip-flops
- Data select inputs to multiplexers and demultiplexers

106

Observation Point Selection

□ Goal

- **Observability** of the transitive fanins of the added point is improved

□ Common choice

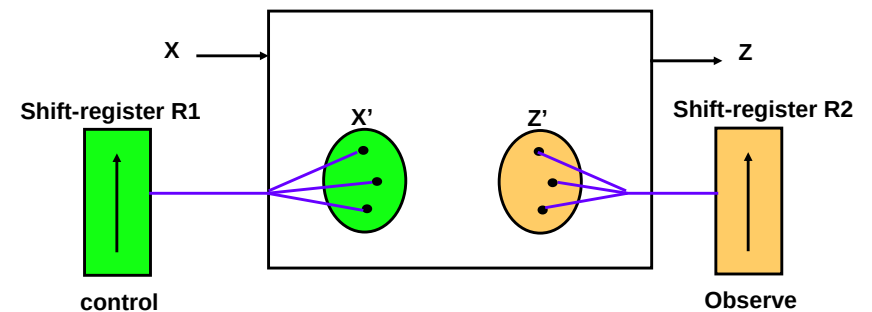
- Stem lines with **more fanouts**
- Global feedback paths
- Redundant signal lines
- Output of logic devices having many inputs
 - MUX, XOR trees
- **Output from state devices**
- Address, control and data buses

107

Problems with Test Point Insertion

□ Large number of I/O pins

- Can be resolved by adding MUXs to reduce the number of I/O pins, or by adding shift-registers to impose CP values



108

What Is Scan ?

Objective

- To provide controllability and observability at **internal state variables** for testing

Method

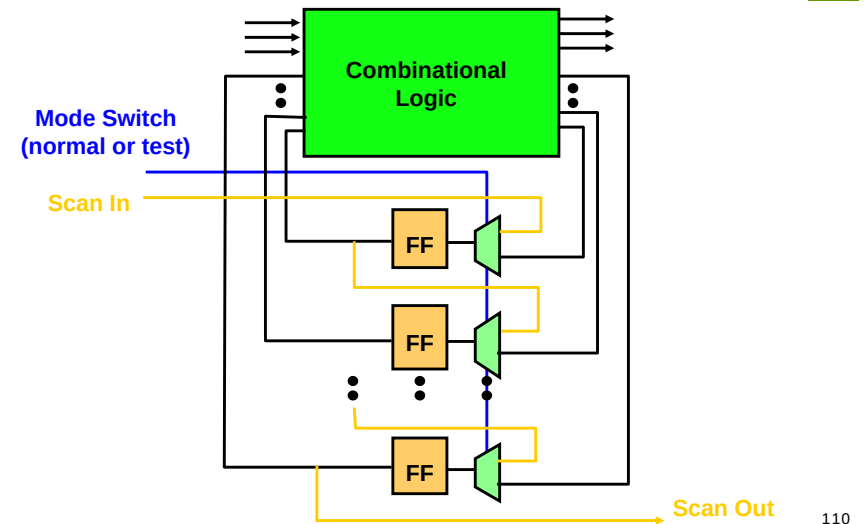
- Add **test mode** control signal(s) to circuit
- Connect **flip-flops to form shift registers** in test mode
- Make inputs/outputs of the flip-flops in the shift register controllable and observable

Types

- Internal scan
 - Full scan, partial scan, random access
- Boundary scan

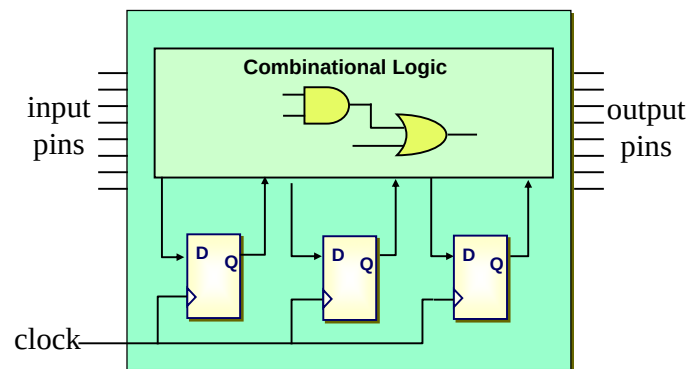
109

Scan Concept



110

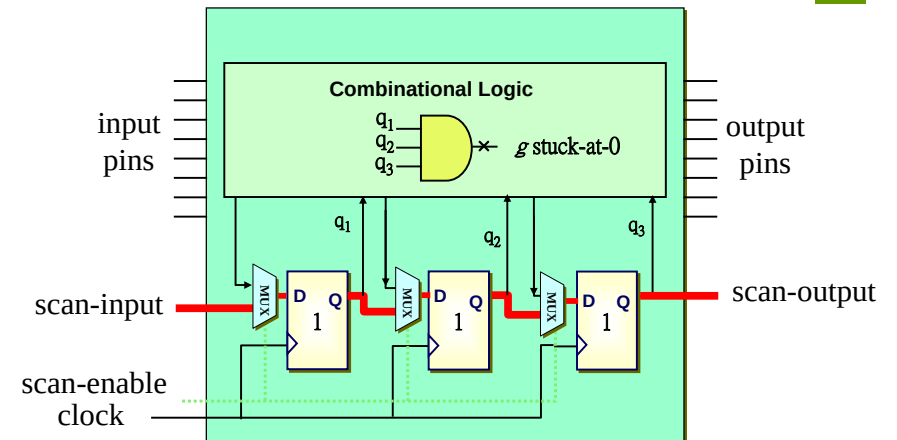
Logic Design before Scan Insertion



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

111

Logic Design after Scan Insertion



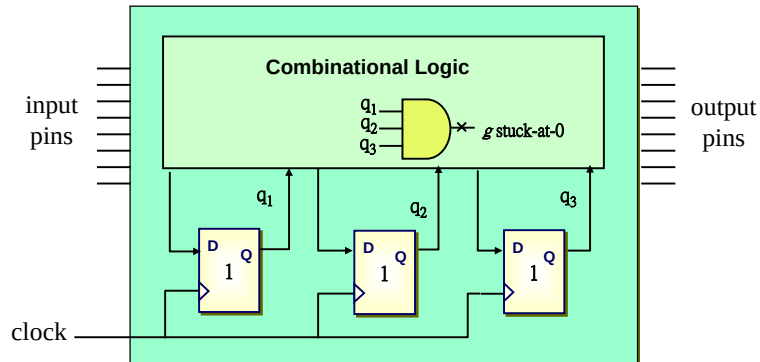
Scan Chain provides an easy access to flip-flops
→ Pattern generation is much easier !!

112

Scan Insertion

Example

- 3-stage counter



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

113

Overhead of Scan Design

Case study

- #CMOS gates = 2000
- Fraction of flip-flops = 0.478
- Fraction of normal routing = 0.471

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

114

Full Scan Problems

Problems

- Area overhead
- Possible performance degradation
- High test application time
- Power dissipation

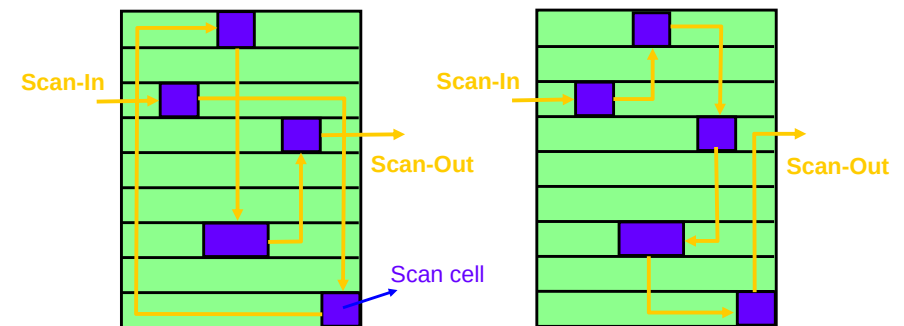
Features of commercial tools

- 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

115

Scan-Chain Reordering

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



Layout of a cell-based design

A better scan-chain order

116

Partial Scan

□ Basic idea

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

□ Cycle-breaking technique

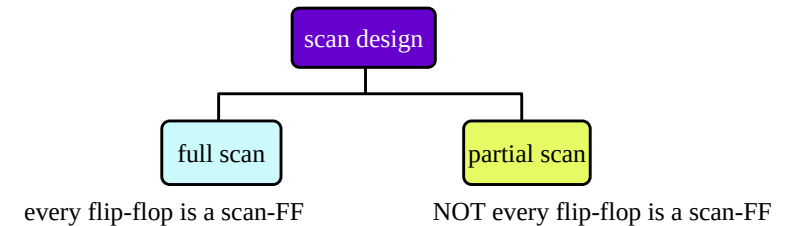
- **Cheng & Agrawal**, IEEE Trans. On Computers, April 1990
- Select scan flip-flops to **simplify sequential ATPG**
- Overhead is about **25% off** than full scan

□ Timing-driven partial scan

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

117

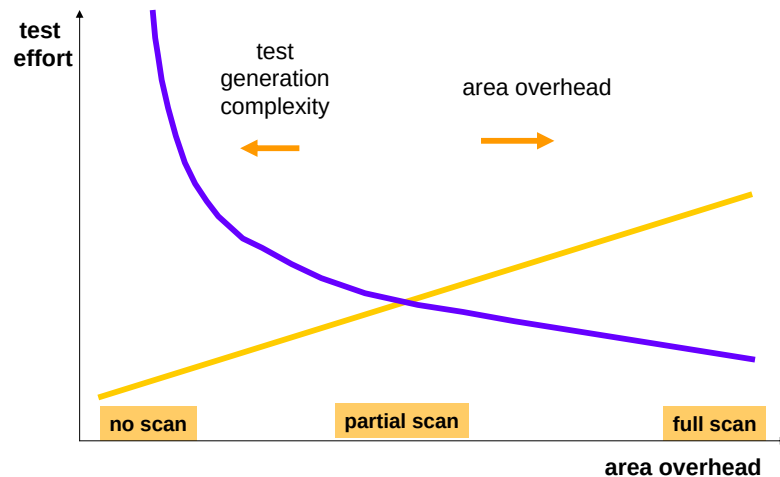
Full Scan vs. Partial Scan



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

118

Area Overhead vs. Test Effort



119

Conclusions

□ Testing

- Conducted after manufacturing
- Must be considered during the design process

□ Major fault models

- Stuck-at, bridging, stuck-open, delay fault, ...

□ Major tools needed

- Design-for-Testability
 - By scan chain insertion or built-in self-test
- Fault simulation
- ATPG

□ Other Applications in CAD

- ATPG is a way of Boolean reasoning and is applicable to many logic-domain CAD problems

120