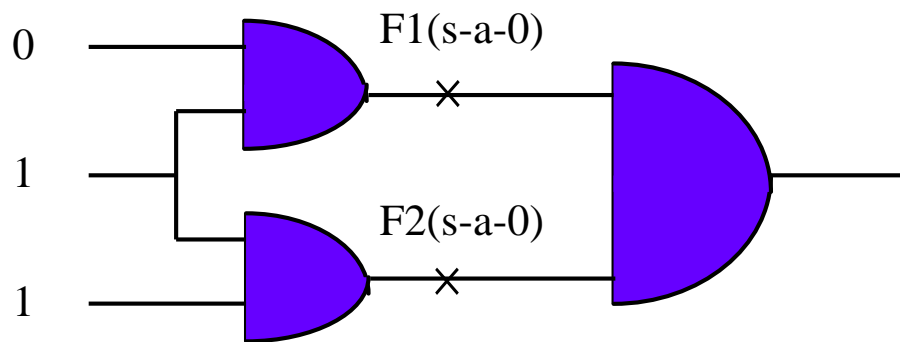


Characteristics of Fault Simulation

- ❑ 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

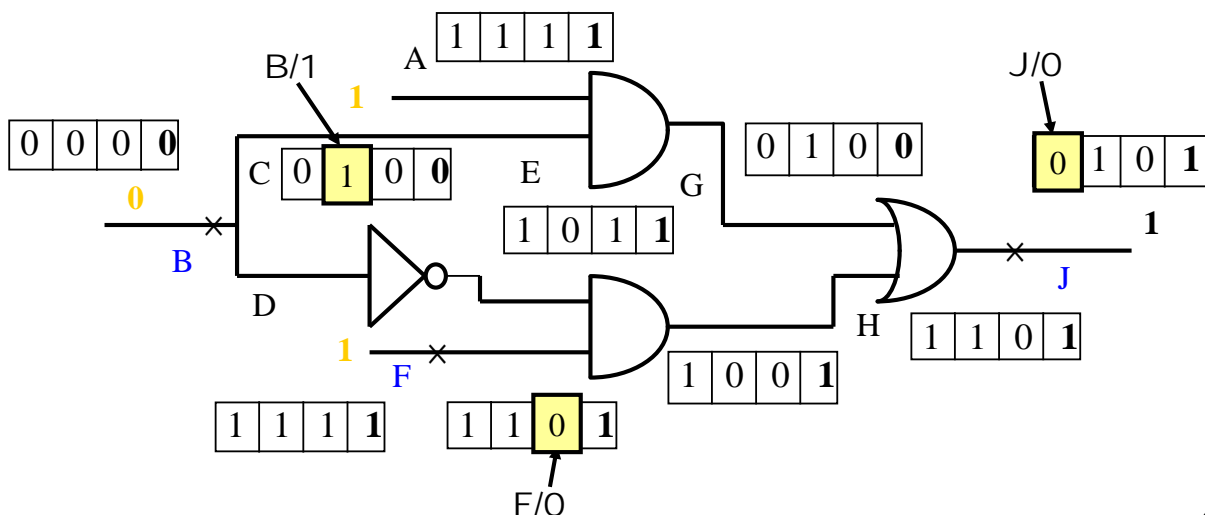
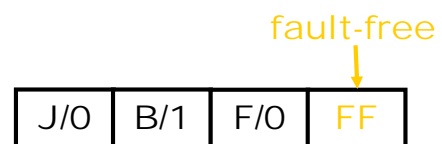
- An event or a value change of a single faulty or fault-free circuit leads to the computation of an entire word
- The fault-free logic simulation is repeated for each pass

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

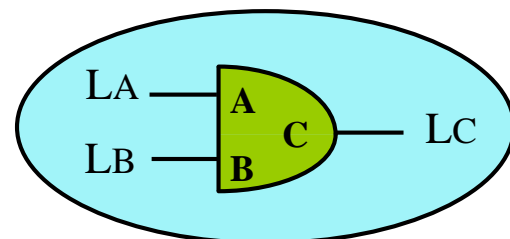
Deductive Fault Simulation

- 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

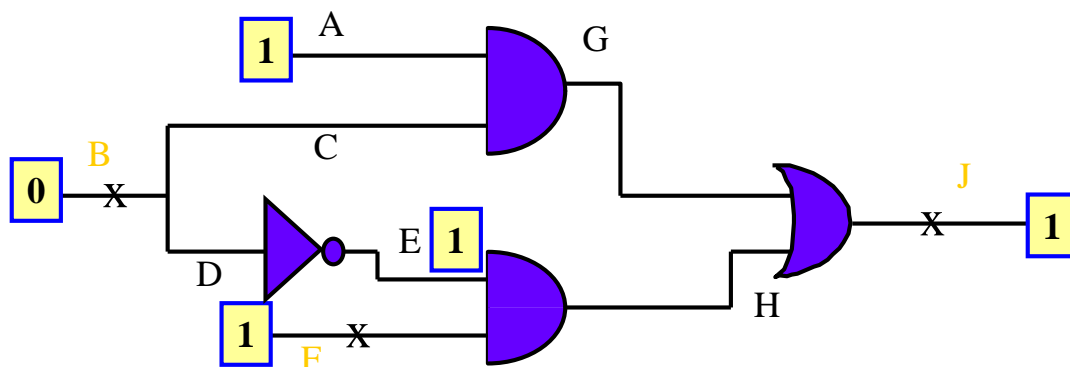
	a	b	z	Output fault list
AND	0	0	0	$\{L_a \cap L_b\} \cup z_1$
	0	1	0	$\{L_a - L_b\} \cup z_1$
	1	0	0	$\{L_b - L_a\} \cup z_1$
	1	1	1	$\{L_a \cup L_b\} \cup z_0$
OR	0	0	0	$\{L_a \cup L_b\} \cup z_1$
	0	1	1	$\{L_b - L_a\} \cup z_0$
	1	0	1	$\{L_a - L_b\} \cup z_0$
	1	1	1	$\{L_a \cap L_b\} \cup z_0$
NOT	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:

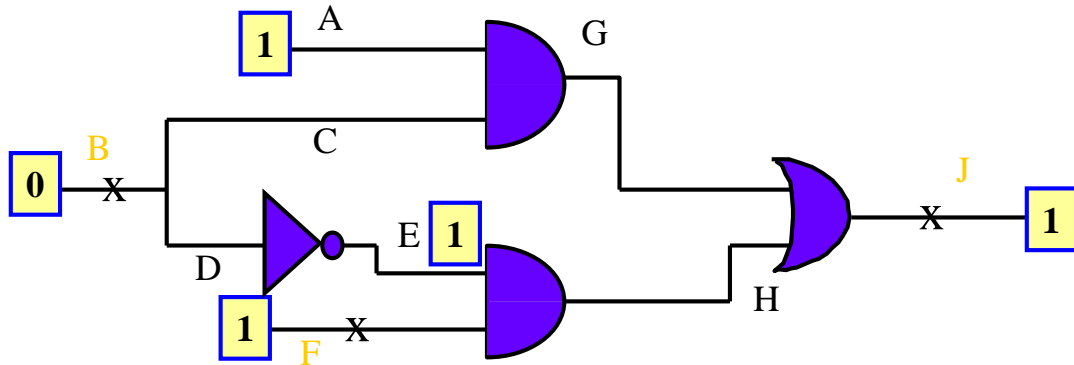
$LB = \{B/1\}$, $LF = \{F/0\}$, $LA = \emptyset$, $LC=LD = \{B/1\}$

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:

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

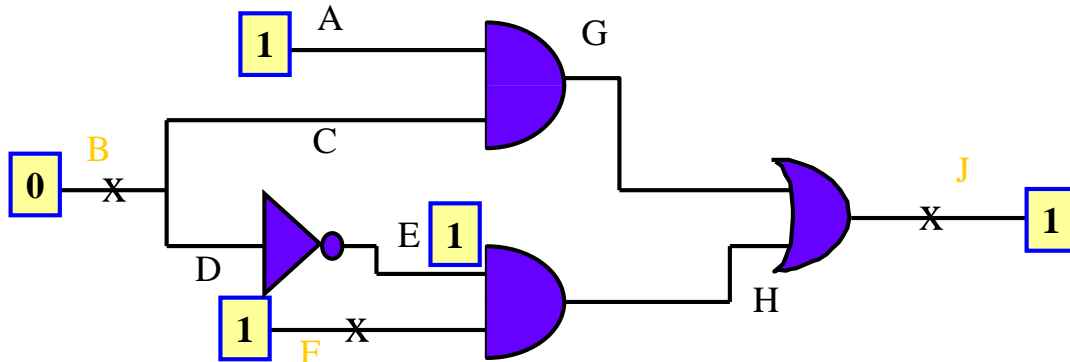
$LE = (LD) \cup E/0 = \{B/1, E/0\}$

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:

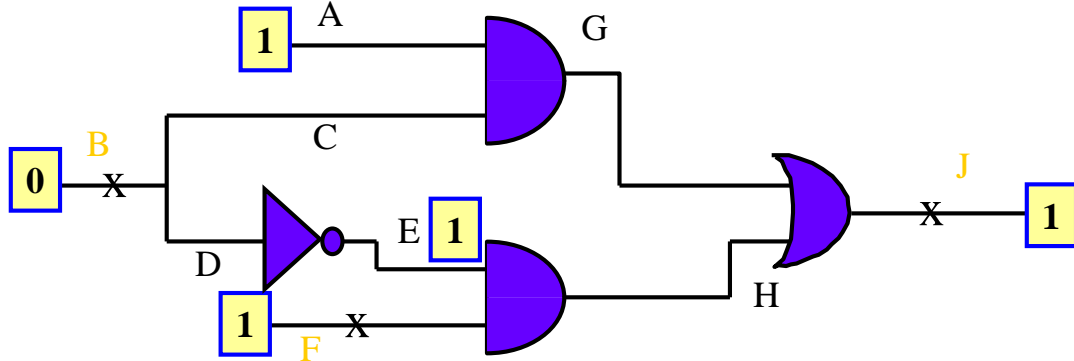
$LH = (LE \cup LF) \cup LH = \{B/1, E/0, F/0, H/0\}$

66

Deductive Fault Simulation

Example (4/4)

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



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

Final fault list at PO J:

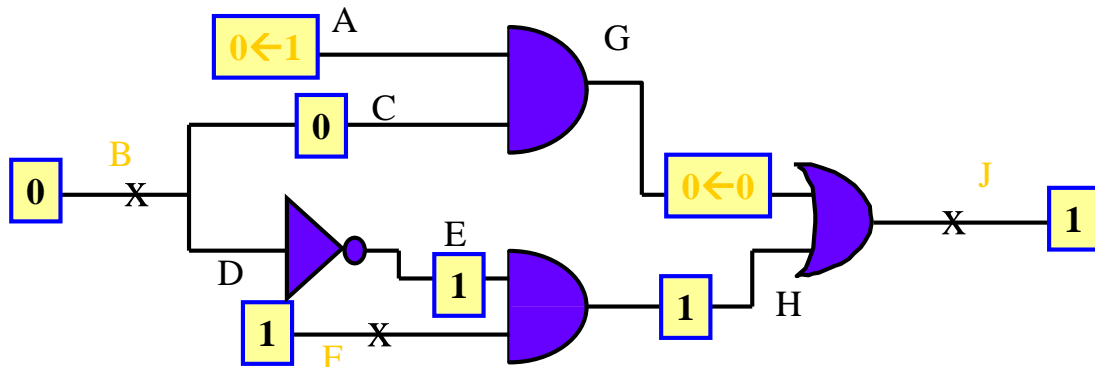
$LJ = (LH - LG) \cup LJ = \{\text{E/0}, F/0, J/0\}$

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

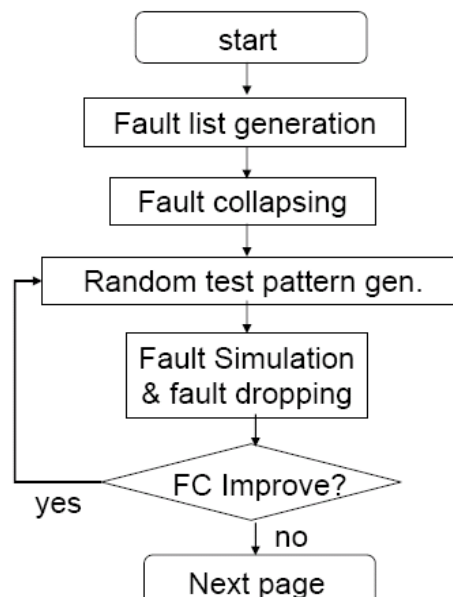
□ PODEM

□ Design for Testability

69

Typical ATPG Flow

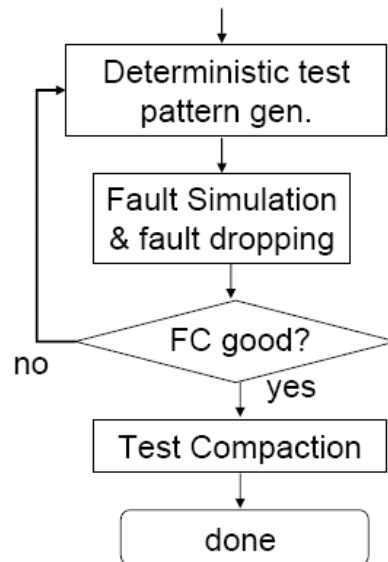
□ 1st phase: random test pattern generation



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

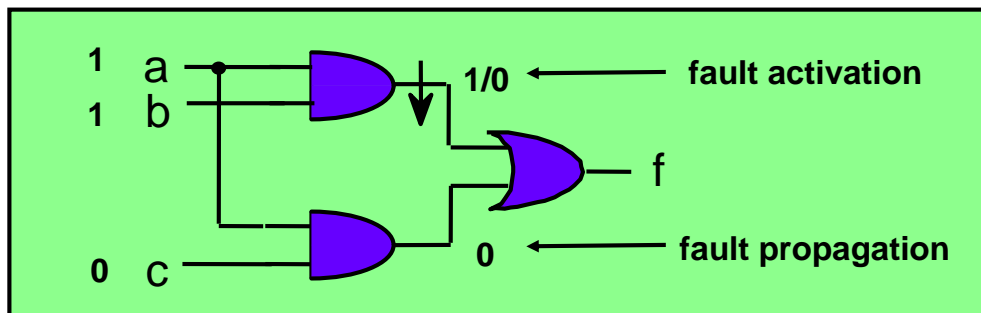
Input vectors (1,1,0) and (1,1,-) are fully and partially specified test patterns of fault α , respectively.

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)



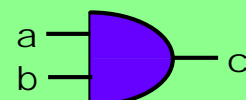
73

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

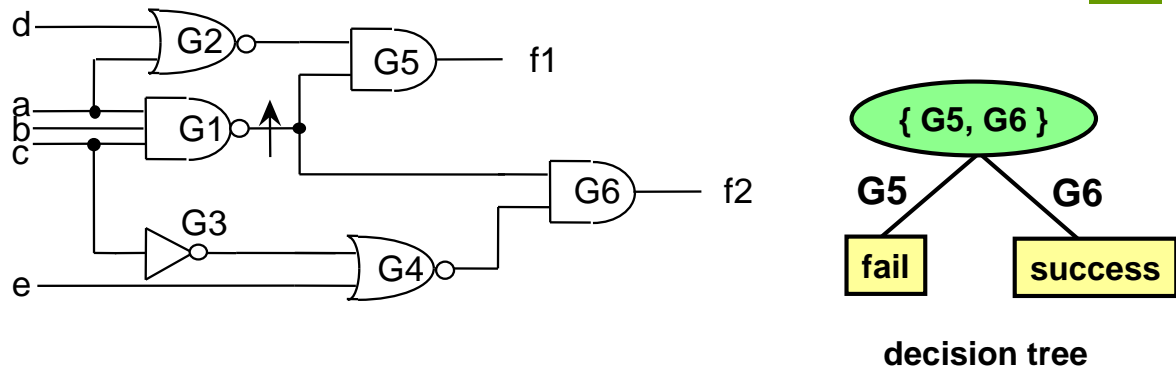
To justify $c=1 \rightarrow a=1$ and $b=1$ (implication)
To justify $c=0 \rightarrow a=0$ or $b=0$ (decision)



74

Structural Test Generation

D-Algorithm: Fault Propagation



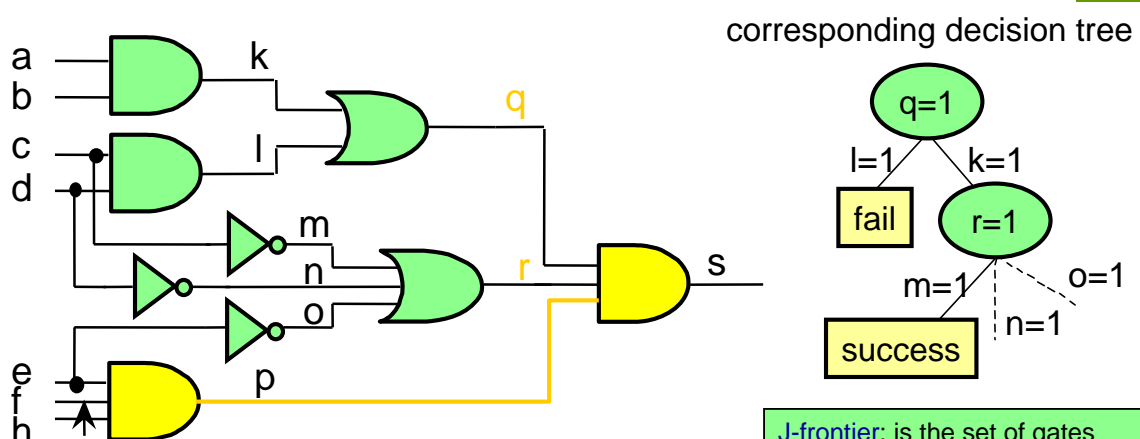
- **Fault activation**
 - $G1=0 \rightarrow \{a=1, b=1, c=1\} \rightarrow \{G3=0\}$
- **Fault propagation:** through G5 or G6
- **Decision through G5:**
 - $G2=1 \rightarrow \{d=0, a=0\} \rightarrow$ inconsistency at a \rightarrow backtrack !!
- **Decision through G6:**
 - $\rightarrow G4=1 \rightarrow e=0 \rightarrow$ 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 }.

75

Structural Test Generation

D-Algorithm: Line Justification



- **FA** \rightarrow set h to 0
- **FP** $\rightarrow e=1, f=1 (\rightarrow o=0)$; **FP** $\rightarrow q=1, r=1$
- **To justify** $q=1 \rightarrow l=1$ or $k=1$
- **Decision:** $l=1 \rightarrow c=1, d=1 \rightarrow m=0, n=0 \rightarrow r=0 \rightarrow$ inconsistency at r \rightarrow backtrack !
- **Decision:** $k=1 \rightarrow a=1, b=1$
- **To justify** $r=1 \rightarrow m=1$ or $n=1 (\rightarrow c=0$ or $d=0) \rightarrow$ 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.

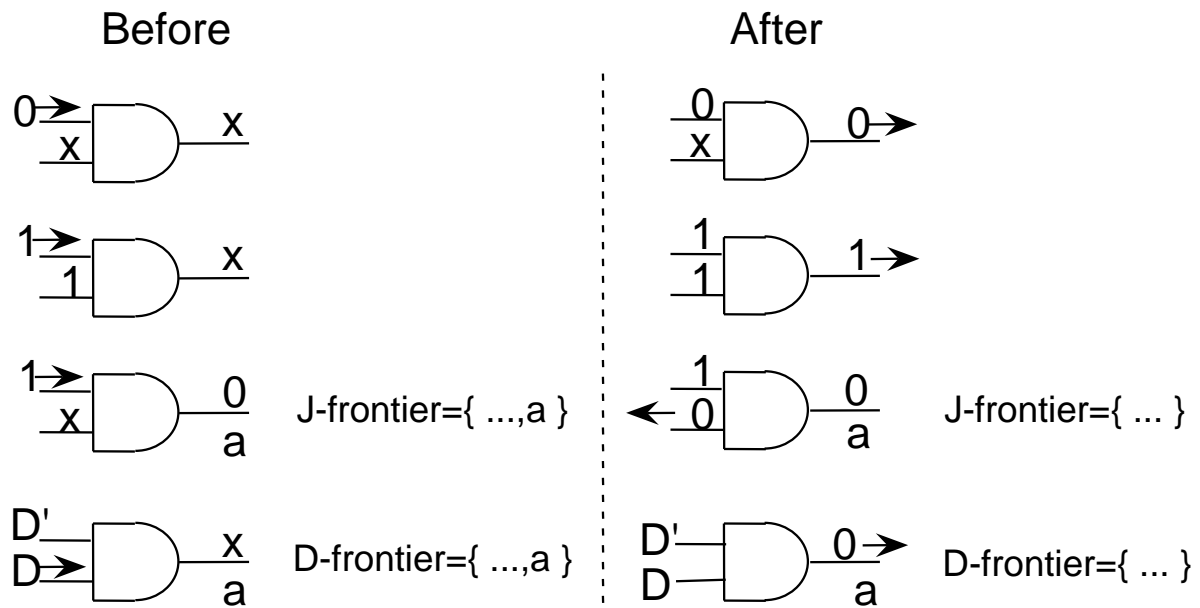
77

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**

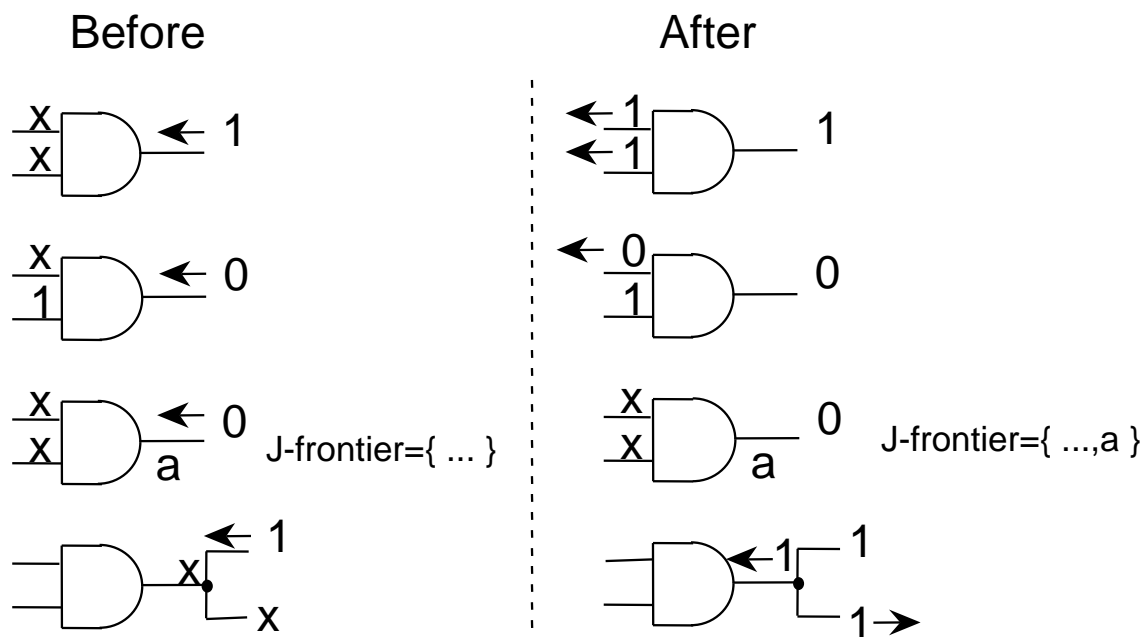
78

Forward Implication



79

Backward Implication

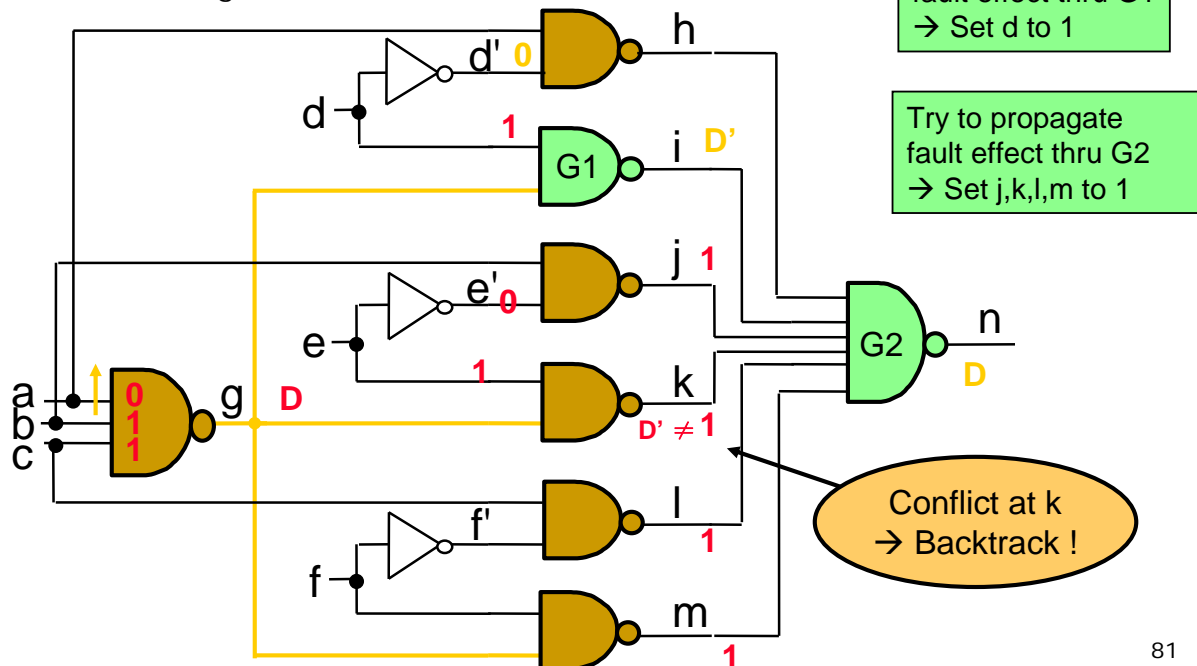


80

D-Algorithm (1/4)

Example

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

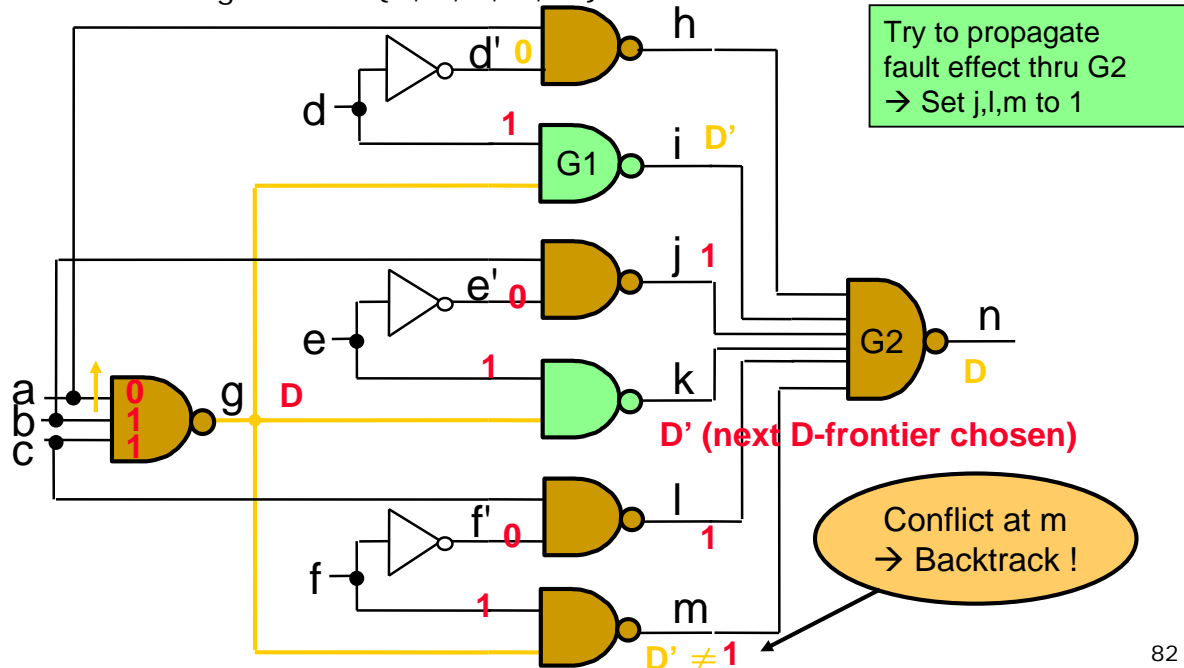


81

D-Algorithm (2/4)

Example

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

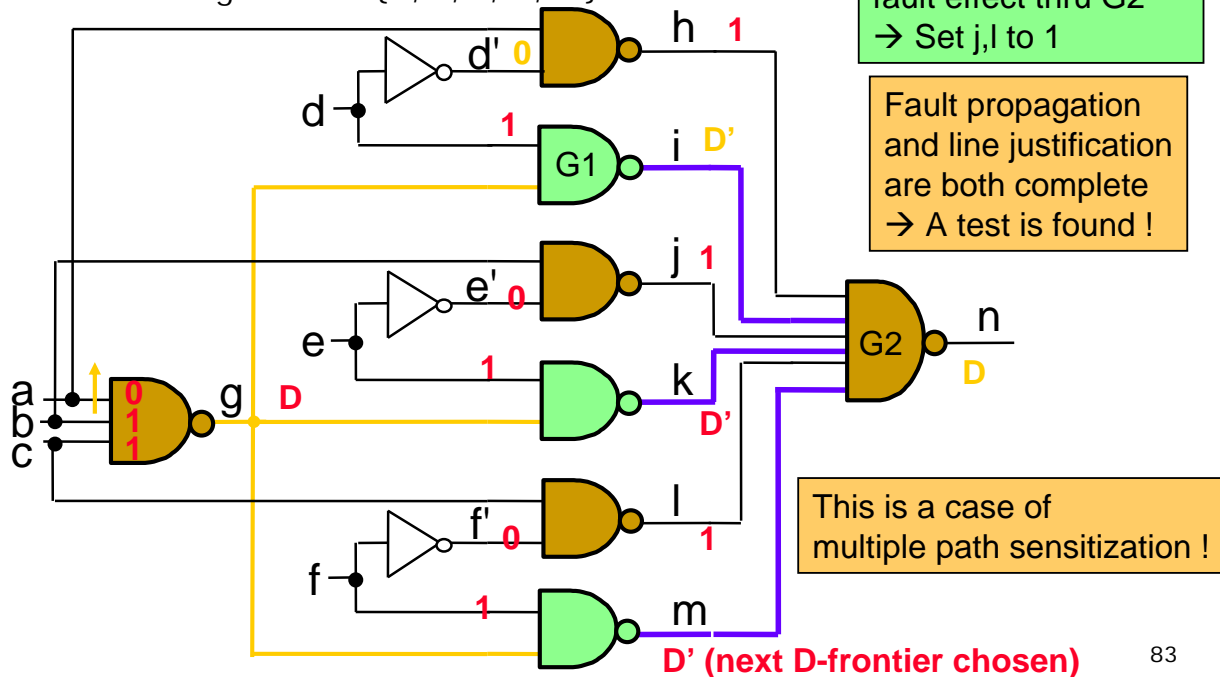


82

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

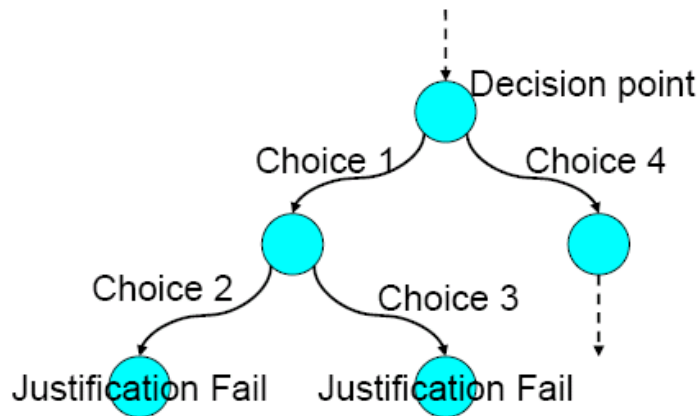
e=1	k=D' e'=0 j=1	Propagate via k
l=1 m=1	n=D f'=0 f=1 m=D'	Propagate via n Contradiction
f=1	m=D' f'=0 l=1 n=D	Propagate via m

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

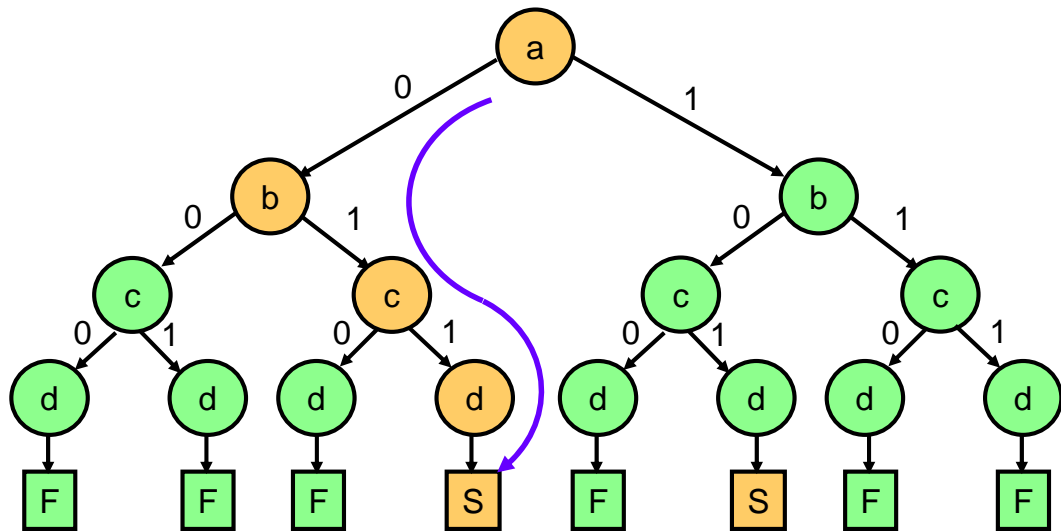
- The test generation is done through a sequence of **direct assignments at PI's**
- Decision points are at PIs, thus the number of **backtracking** might be **fewer**

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

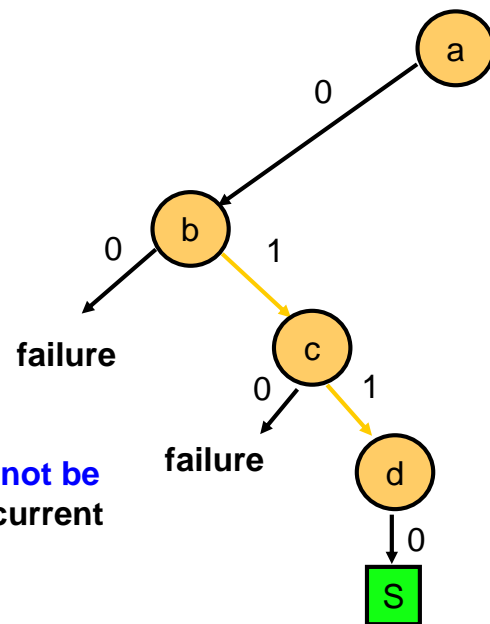
```
Backtrace(w, v_w) {  
    /* Maps objective into a PI assignment */  
    G = w; /* objective node */  
    v = v_w; /* objective value */  
    while (G is a gate output) { /* not reached PI yet */  
        inv = inversion of G;  
        select an input (j) of G with value x;  
        G = j;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

 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);

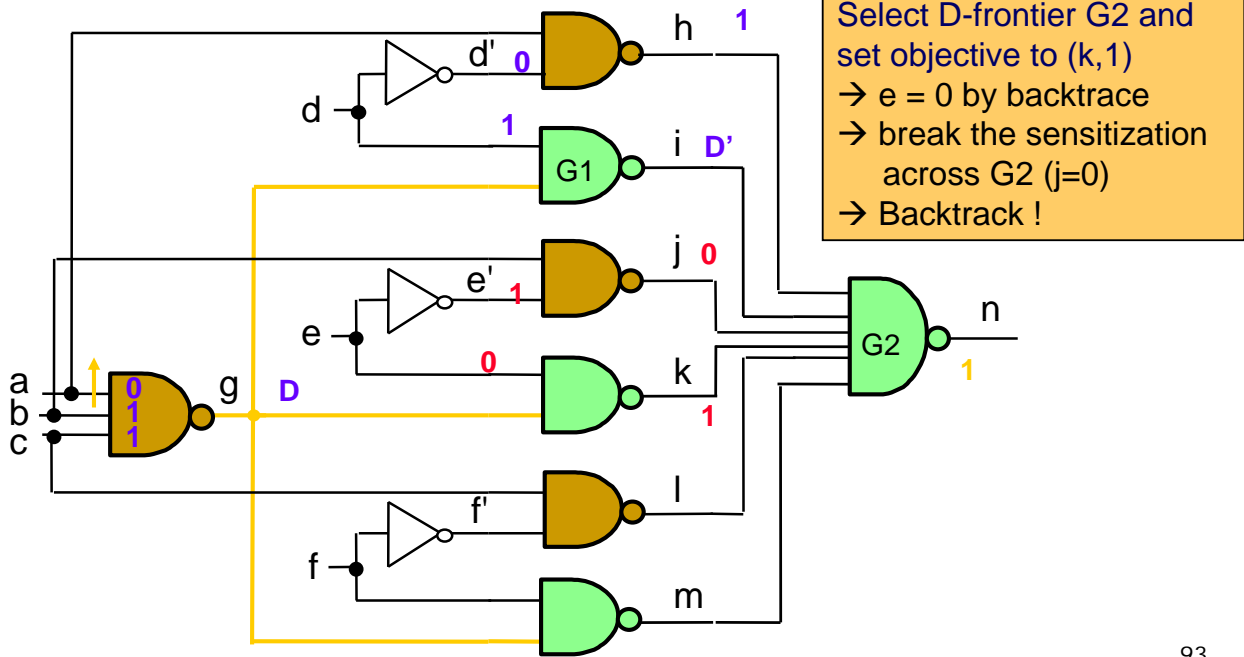
 Return (FAILURE);

end

92

PODEM Algorithm (1/4)

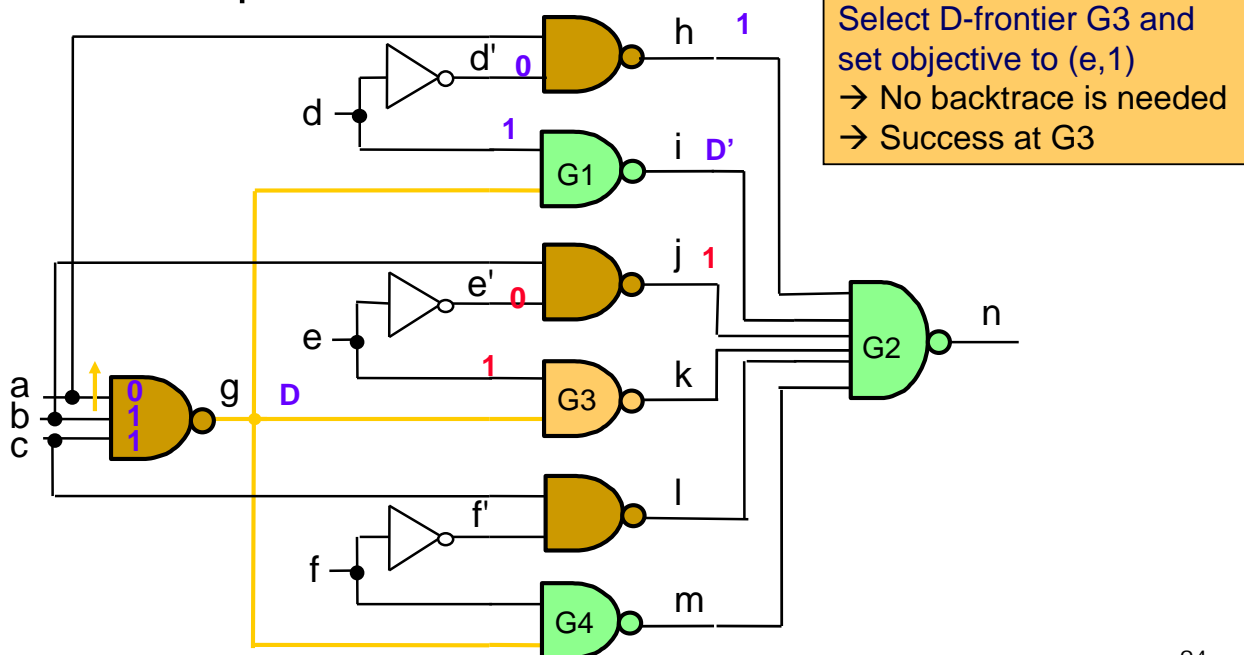
Example



93

PODEM Algorithm (2/4)

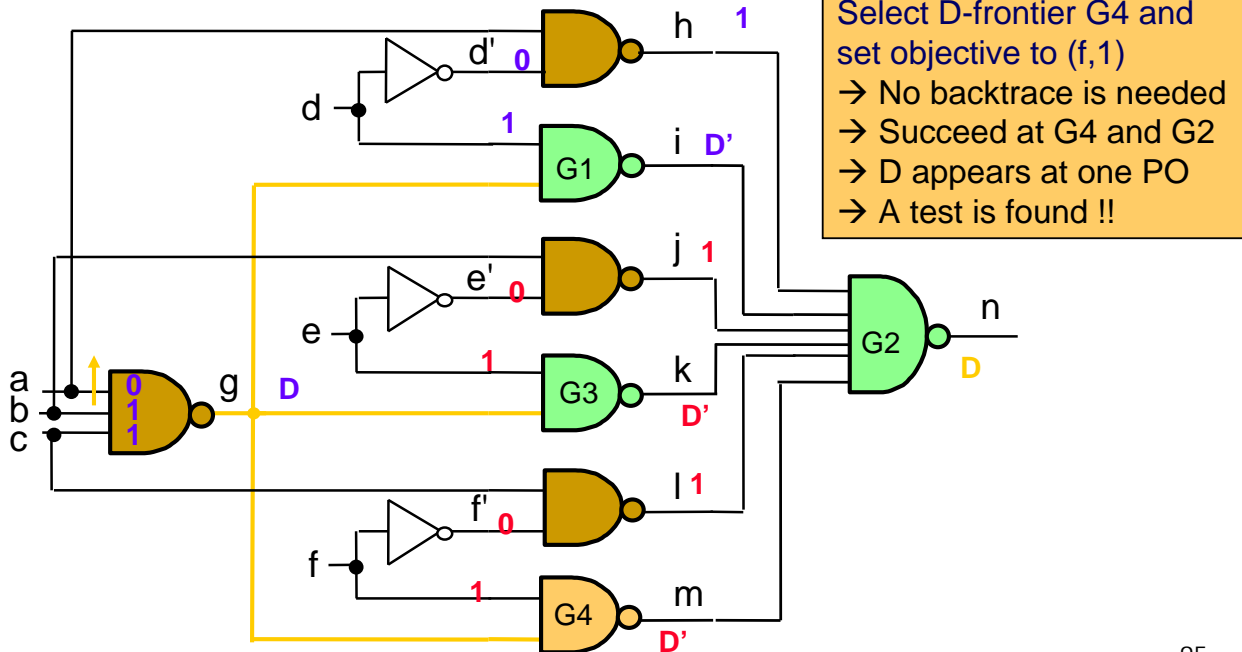
Example



94

PODEM Algorithm (3/4)

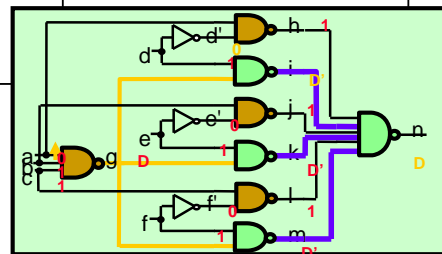
Example



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	m	no solutions! → backtrack flip PI assignment
	e=1	e'=0 j=1 k=D'	m,n	
l=1	f=1	f'=0 l=1 m=D' n=D		

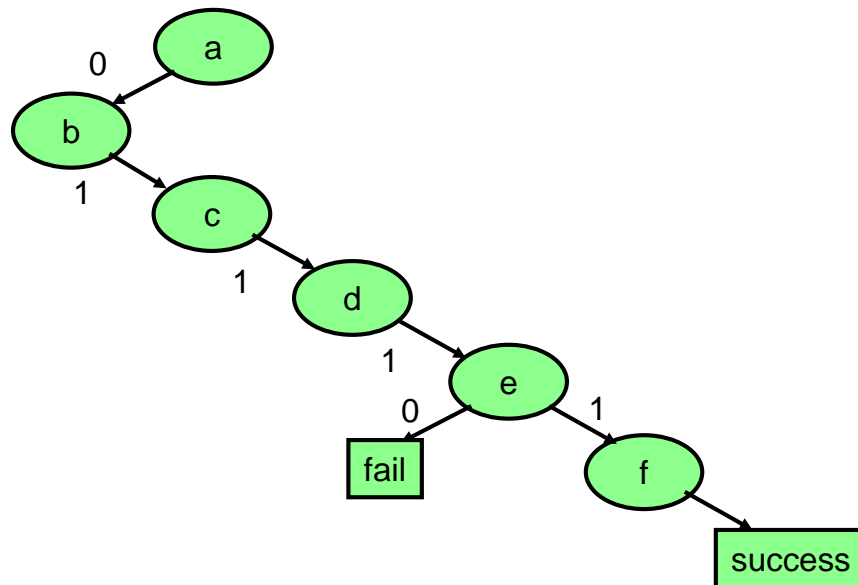


96

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

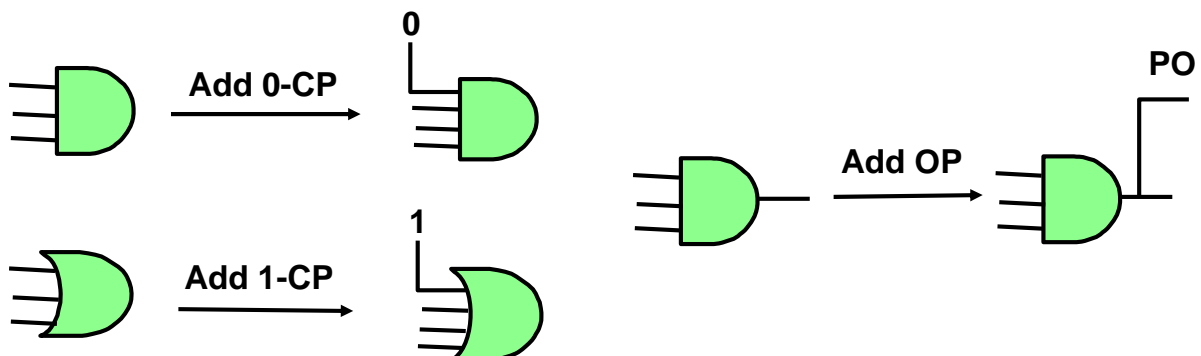
□ Employ test points to enhance **controllability** and **observability**

- CP: Control Points

- Primary inputs used to enhance controllability

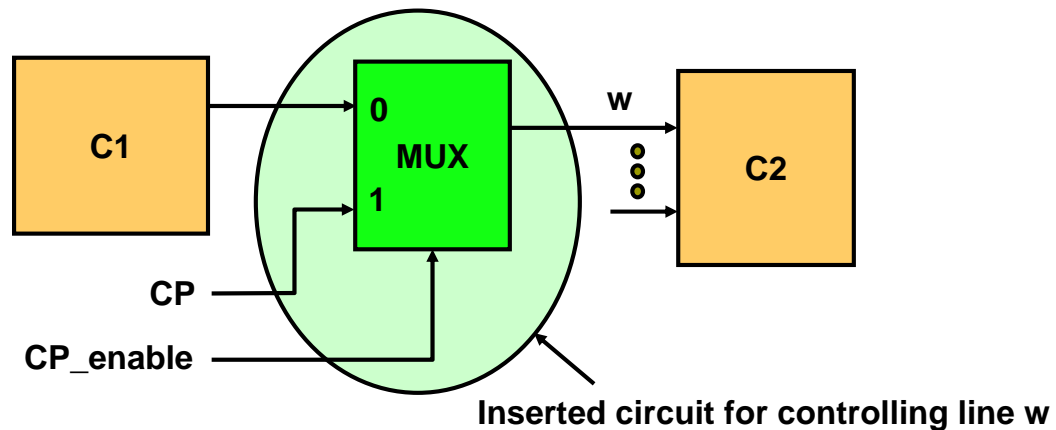
- OP: Observability Points

- 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

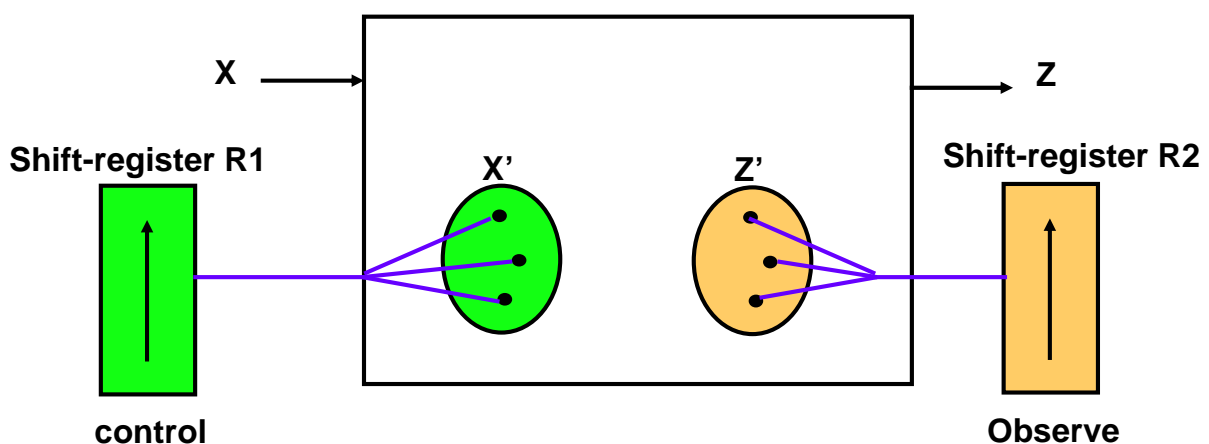
- Stem lines with **more fanouts**
- Global feedback paths
- Redundant signal lines
- Output of logic devices having many inputs
 - MUX, XOR trees
- **Output from state devices**
- Address, control and data buses

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

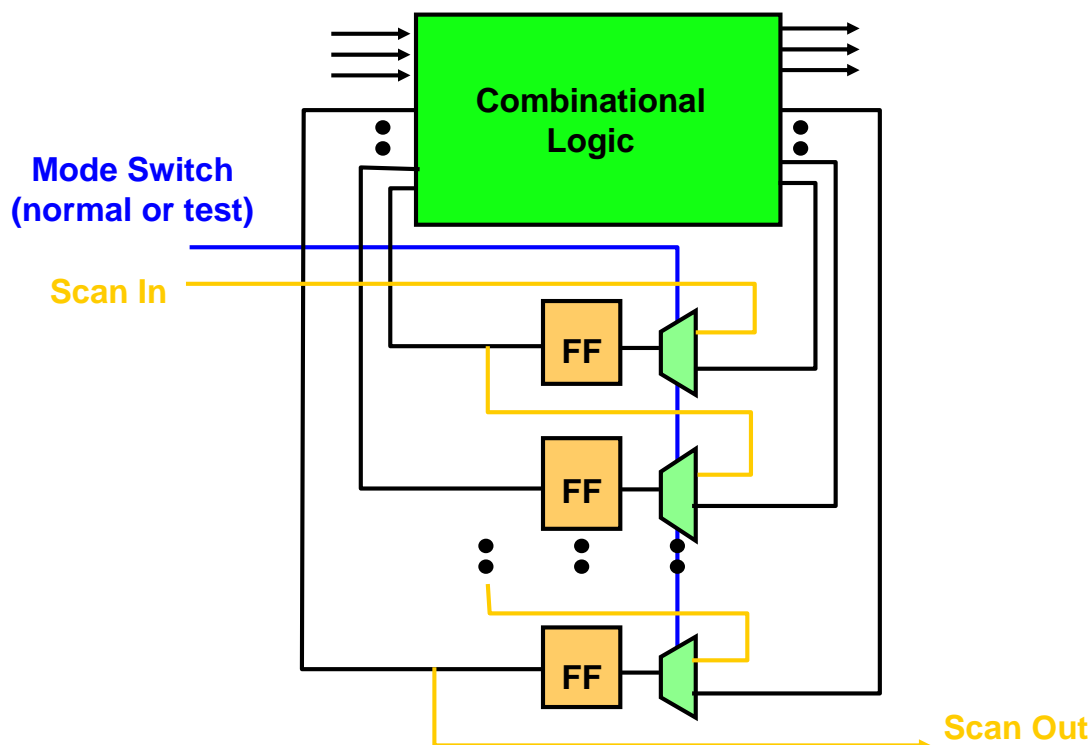
- Add **test mode** control signal(s) to circuit
- Connect **flip-flops to form shift registers** in test mode
- Make inputs/outputs of the flip-flops in the shift register controllable and observable

□ Types

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

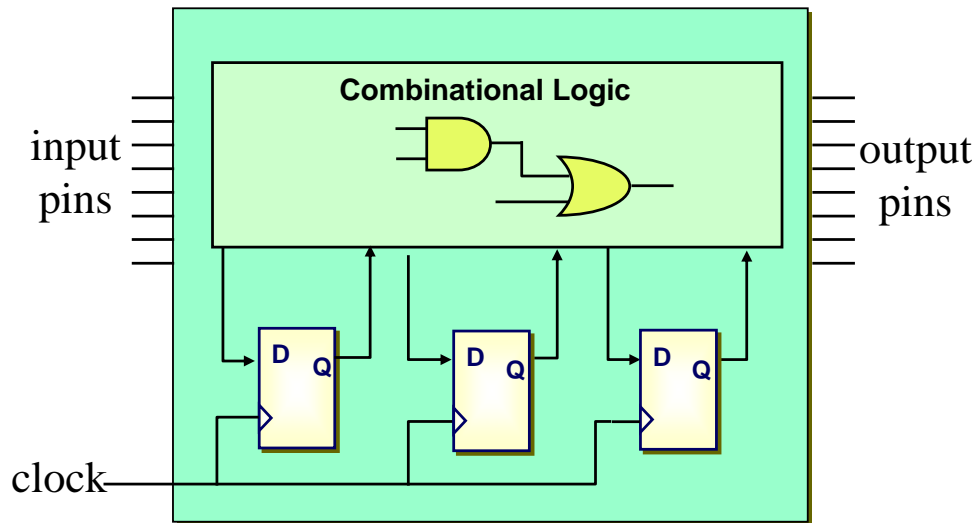
109

Scan Concept



110

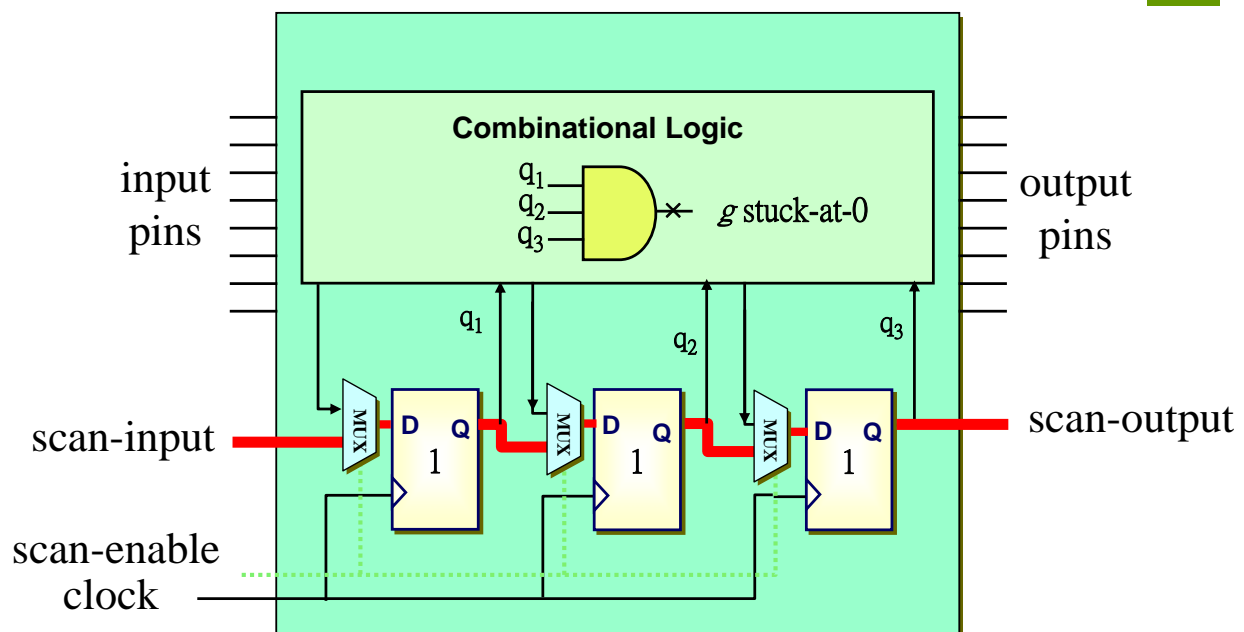
Logic Design before Scan Insertion



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

111

Logic Design after Scan Insertion



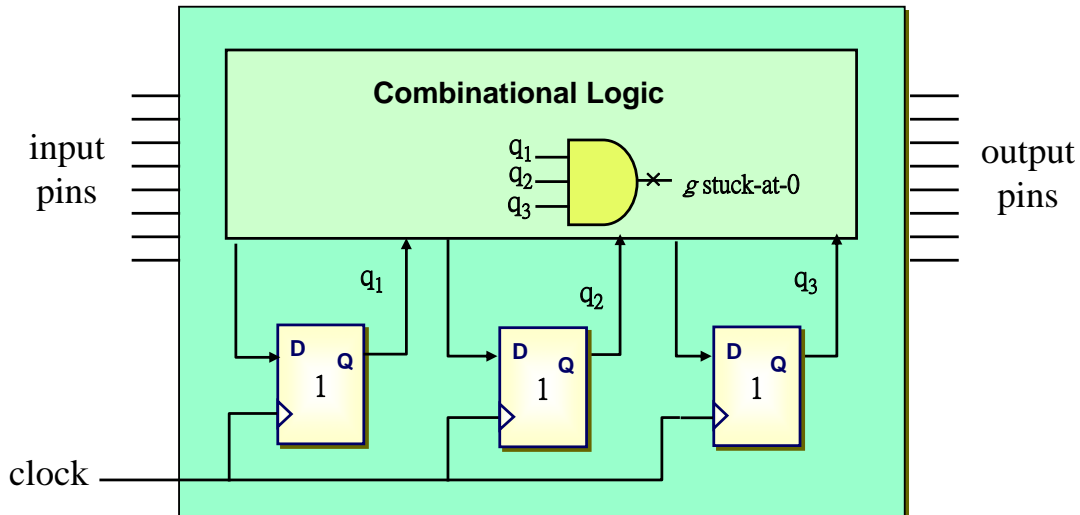
Scan Chain provides an easy access to flip-flops
→ Pattern generation is much easier !!

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

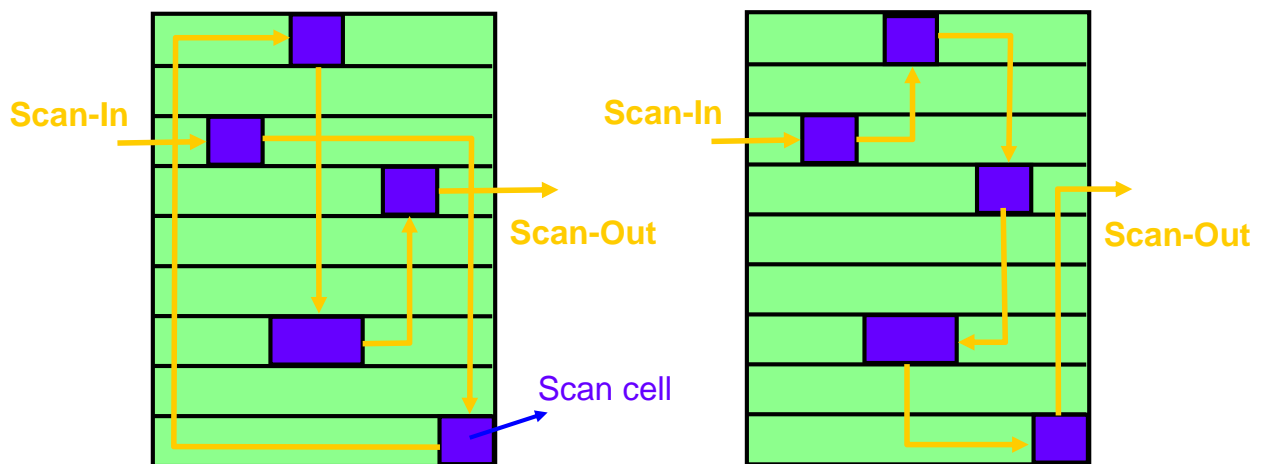
□ Features of commercial tools

- 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

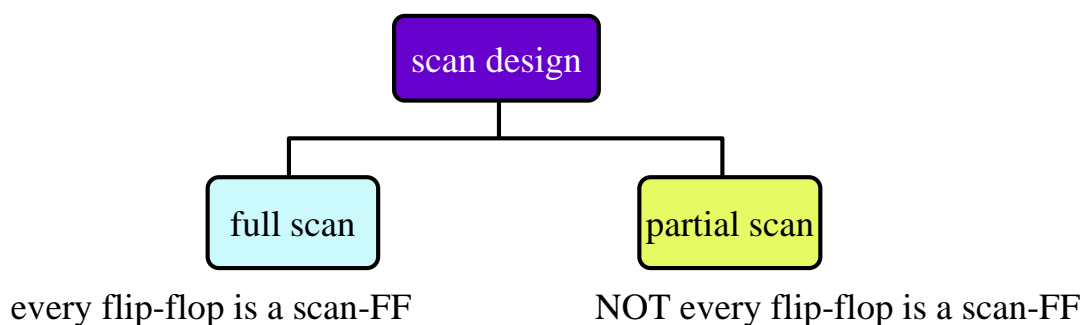
- **Cheng & Agrawal**, IEEE Trans. On Computers, April 1990
- Select scan flip-flops to **simplify sequential ATPG**
- Overhead is about **25% off** than full scan

□ Timing-driven partial scan

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

117

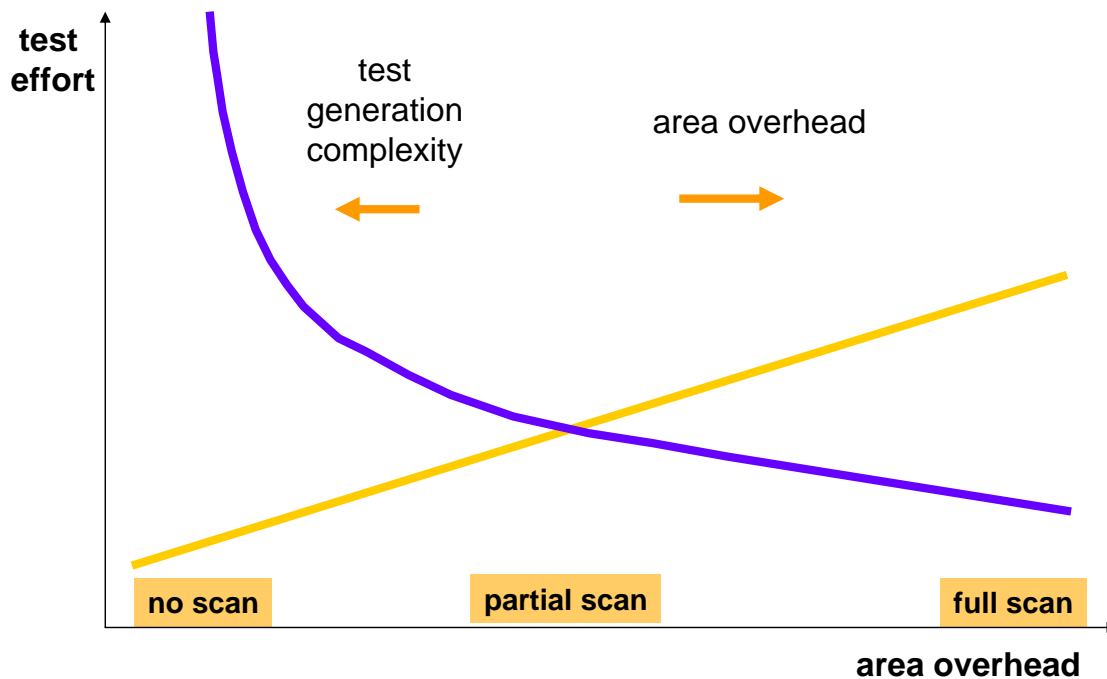
Full Scan vs. Partial Scan



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