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

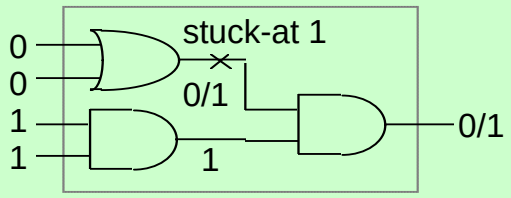
- **To accelerate the ATPG**
  - Random patterns are often generated first to detect easy-to-detect faults, then a deterministic TG is performed to generate tests for the remaining faults

Combinational ATPG

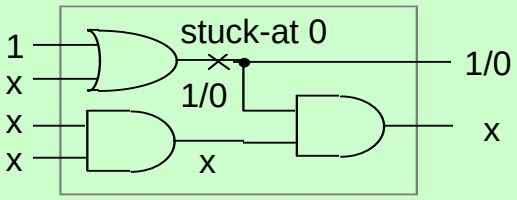
- **Test Generation (TG) Methods**
  - Based on Truth Table
  - Based on Boolean Equation
  - Based on Structural Analysis
- **Milestone Structural ATPG Algorithms**
  - D-algorithm [Roth 1967]
  - 9-Valued D-algorithm [Cha 1978]
  - PODEM [Goel 1981]
  - FAN [Fujiwara 1983]

A Test Pattern

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

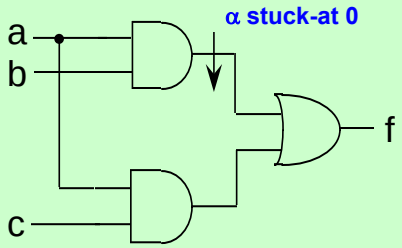


A Partially Specified Test Pattern (certain PI's could be undefined)



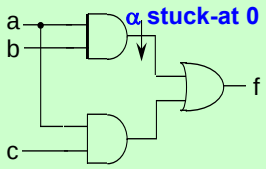
Test Generation Methods
(From Truth Table)

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



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

Test Generation Methods
(Using Boolean Equation)



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

$T\alpha$ = the set of all tests for fault α

$= ON\_set(f \oplus f\alpha)$

$= ON\_set(f) * OFF\_set(f\alpha) + OFF\_set(f) * ON\_set(f\alpha)$

$= \{(a,b,c) \mid (ab+ac)(ac)' + (ab+ac)'(ac) = 1\}$ ← 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

Boolean Difference

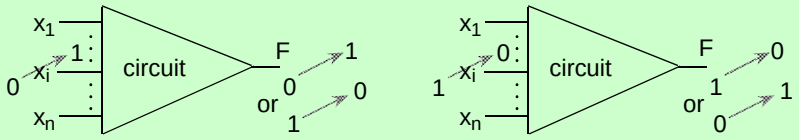
- **Physical Meaning of Boolean Difference**
  - For a logic function $F(X)=F(x_1, \ldots, x_i, \ldots, x_n)$, find all the input combinations that make a value-change at x_i also cause a value-change at F.
- **Logic Operation of Boolean Difference**
  - The Boolean difference of F(X) w.r.t. input x_i is

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

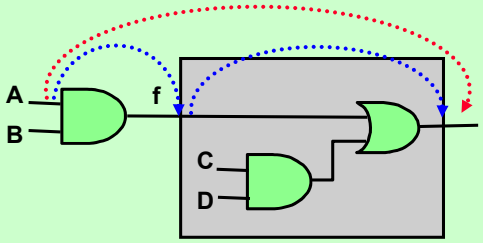
Where

$$F_i(0) = F(x_1, \ldots, 0, \ldots, x_n)$$
$$F_i(1) = F(x_1, \ldots, 1, \ldots, x_n)$$

- **Illustrations of Boolean Difference**



Chain Rule



G(f(A, B), {C, D})
{A,B} and {C,D} have no variables in common

$f = AB$
$G = f + CD$

$\Rightarrow$

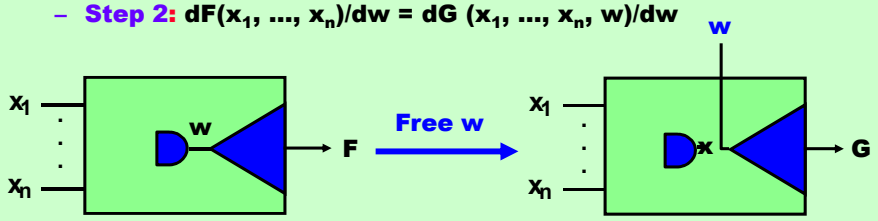
$dG/df = (C' + D')$
$df/dA = B$

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

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

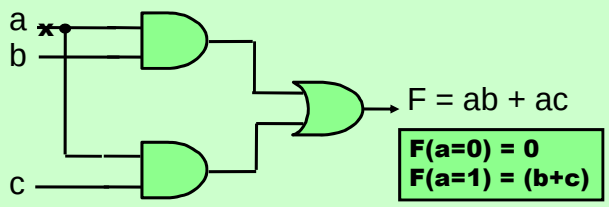
Boolean Difference (con't)

- **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: $dF(x_1, \ldots, x_n)/dw = dG(x_1, \ldots, x_n, w)/dw$



Test Gen. By Boolean Difference

Case 1: Faults are present at PIs.



➡ **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) | $a' \cdot (b+c)=1$} = {(01x), (0x1)}.
Test-set for *a* s-a-0 = {(a,b,c) | $a \cdot (b+c)=1$} = {(11x), (1x1)}.

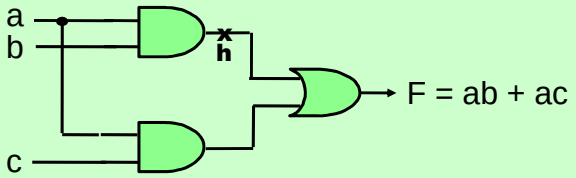
No need to compute The faulty function !!

Fault activation requirement

Fault sensitization requirement

Test Generation By Boolean Difference (con't)

Case 2: Faults are present at internal lines.



G(i.e., F with h floating) = h + ac
dG/dh = G(h=0) ⊕ G(h=1) = (ac ⊕ 1) = (a'+c')

Test-set for *h* s-a-1 is

{ (a,b,c)| h' • (a'+c')=1 } = { (a,b,c)| (a'+b') • (a'+c')=1 } = { (0xx), (x00) }.

Test-set for *h* s-a-0 is

{(a,b,c)| h • (a'+c')=1} = {(110)}.

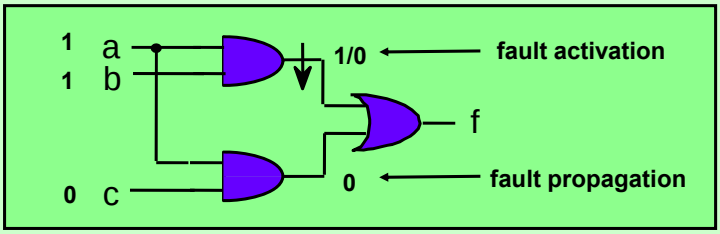
For fault activation **For fault sensitization**

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]**

Test Generation Method (From Circuit Structure)

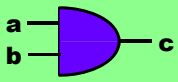
- **Two basic goals**
  - (1) Fault activation (FA)
  - (2) Fault propagation (FP)
  - Both of which requires Line Justification (LJ), I.e., finding input combinations that force certain signals to their desired values
- **Notations:**
  - 1/0 is denoted as D, meaning that good-value is 1 while faulty value is 0
  - Similarly, 0/1 is denoted D'
  - Both D and D' are called fault effects (FE)



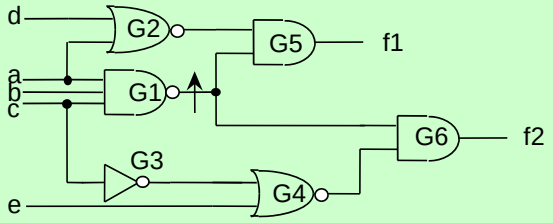
Common Concepts for Structural TG

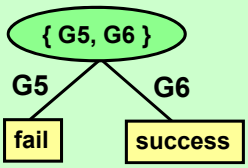
- **Fault activation**
  - Setting the faulty signal to either 0 or 1 is a Line Justification problem
- **Fault propagation**
  - (1) select a path to a PO → decisions
  - (2) Once the path is selected → a set of line justification (LJ) problems are to be solved
- **Line Justification**
  - Involves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

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



Ex: Decision on Fault Propagation





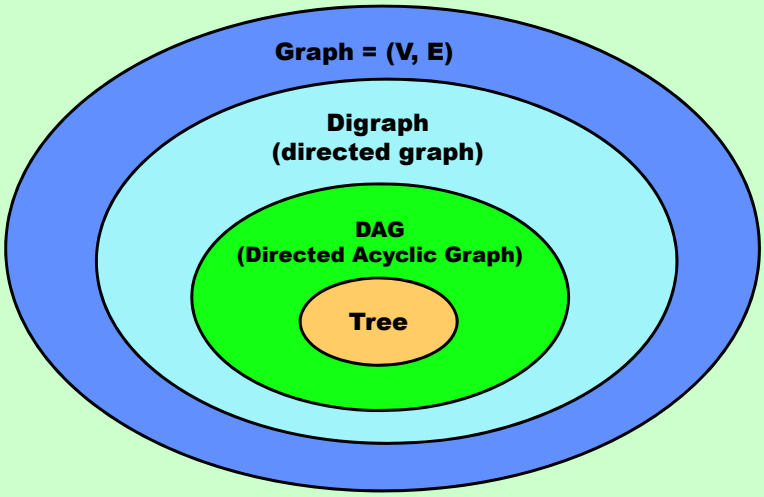
decision tree

- **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)**

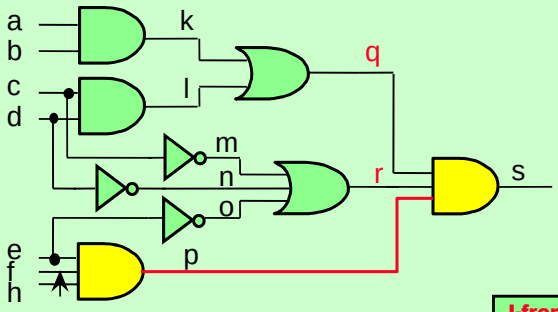
D-frontiers: are the gates whose output value is x, while one or more Inputs are D or D'. For example, initially, the D-frontier is { G5, G6 }.

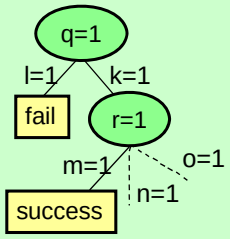
Various Graphs

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



Ex: Decisions On Line Justification





The corresponding decision tree

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}

- **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)

Branch-and-Bound Search

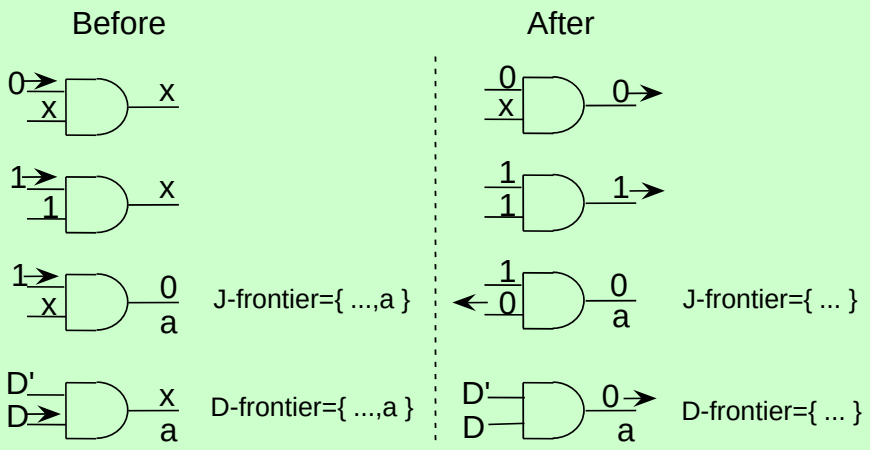
- **Test Generation**
  - Is a branch-and-bound search
  - Every decision point is a **branching** point
  - If a set of decisions lead to a **conflict** (or **bound**), a **backtrack** is taken to explore other decisions
  - **A test is found** when
    - (1) fault effect is propagated to a PO
    -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

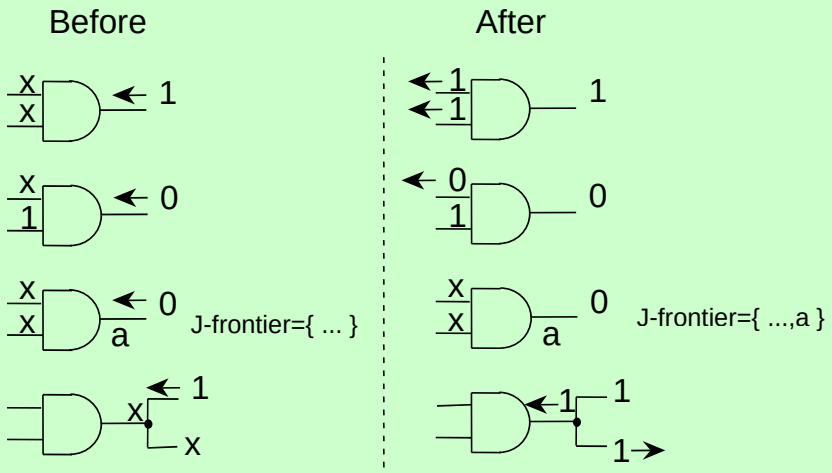
Implications

- **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**

Local Implications (Forward)

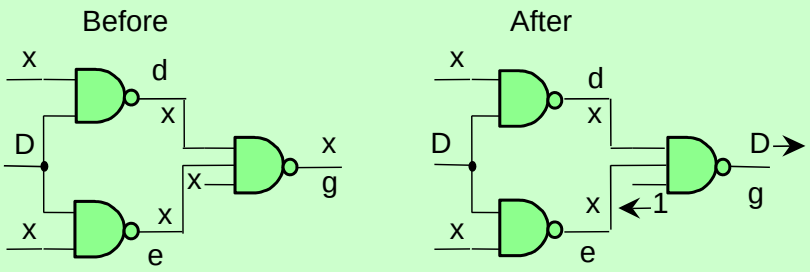


Local Implications (Backward)



ch4-21

Global Implications



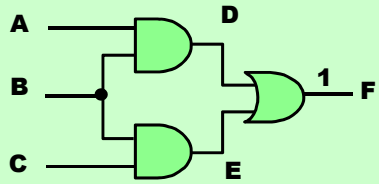
- **Unique D-Drive Implication**
  - **Suppose D-frontier (or D-drive) is {d, e}, → g is a dominator for both d and e, hence a unique D-drive is at g**

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

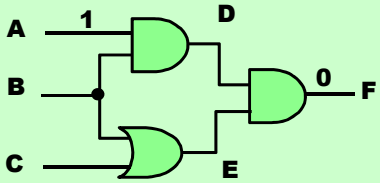
ch4-22

Learning for Global Implication

- **Static Learning** $\boxed{A \rightarrow B \Rightarrow \sim B \rightarrow \sim A}$
  - 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 → F=0
(Static Learning)

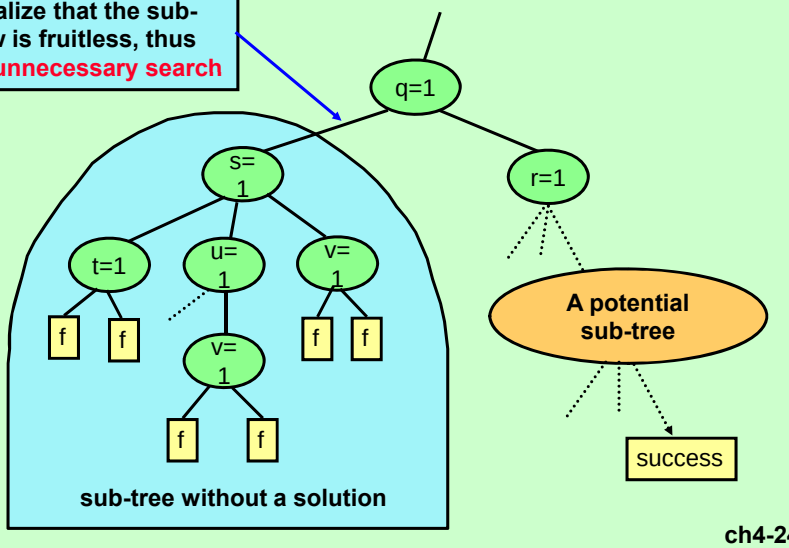


F=0 implies B=0 When A=1
Because {B=1, A=1} → F=1
(Dynamic Learning)

ch4-23

Early Detection of Inconsistency

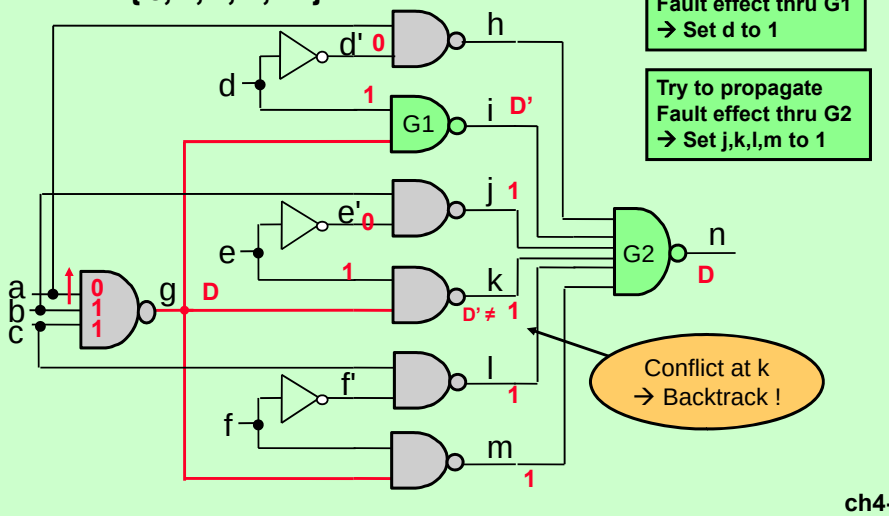
Aggressive implication may help to realize that the sub-tree below is fruitless, thus avoiding unnecessary search



ch4-24

Ex: D-Algorithm (1/3)

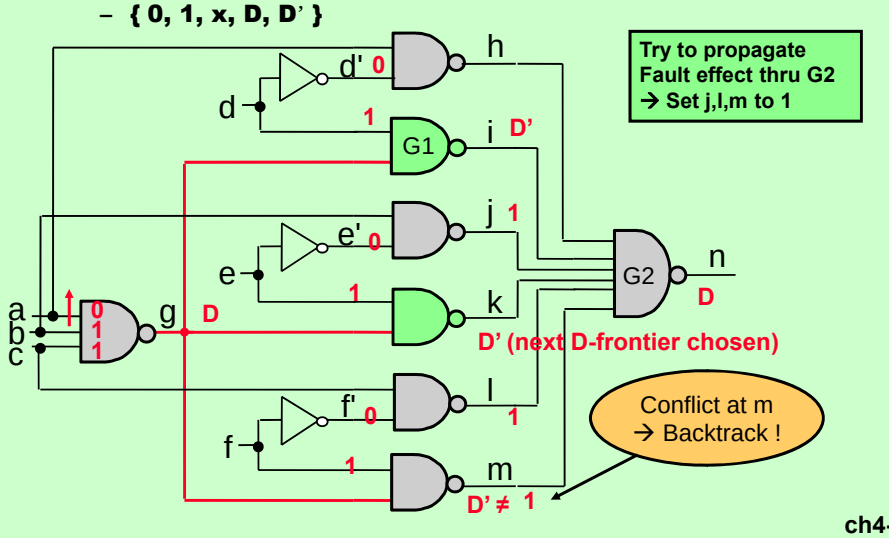
- **Five logic values**
  - **{ 0, 1, x, D, D' }**



ch4-25

Ex: D-Algorithm (2/3)

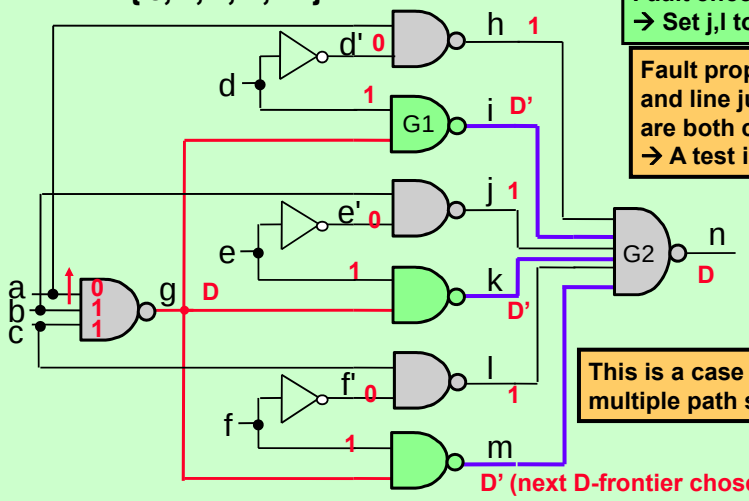
- **Five logic values**
  - **{ 0, 1, x, D, D' }**



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 !

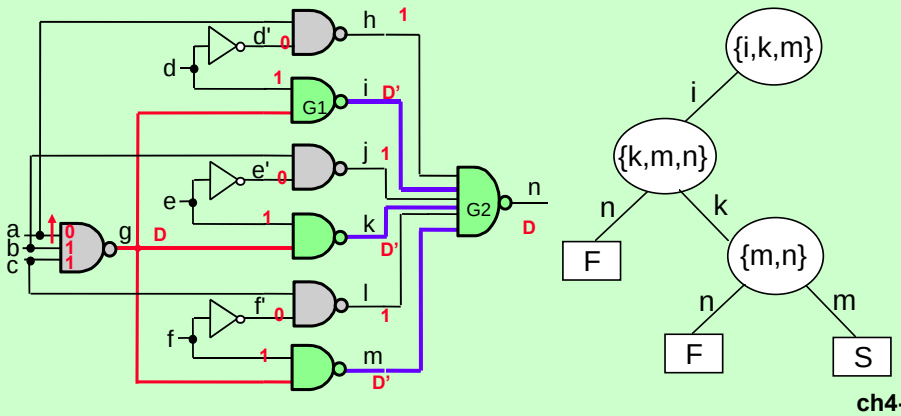
D-Algorithm: Value Computation

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

Decision Tree on D-Frontier

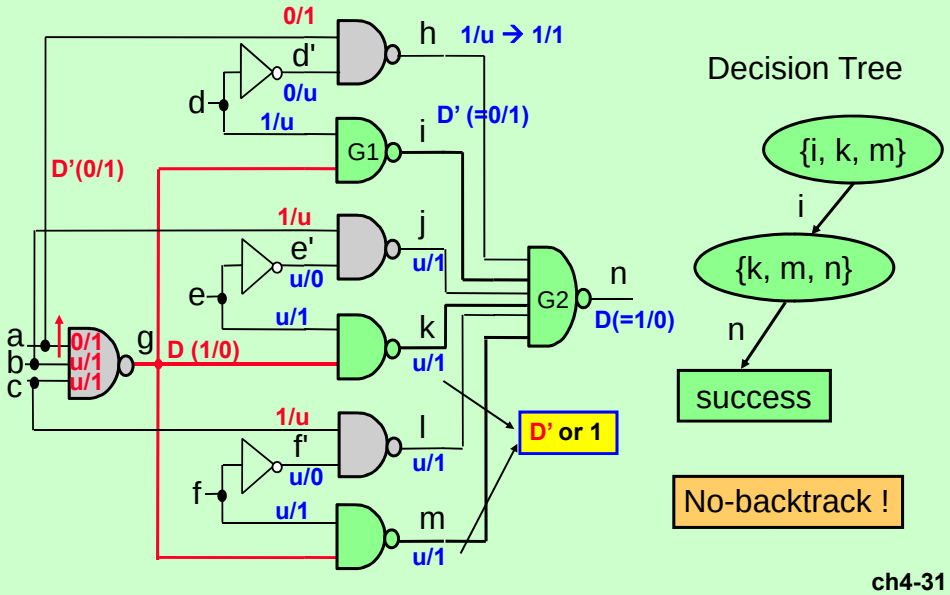
- **The decision tree below**
  - **Node → D-frontier**
  - **Branch → Decision Taken**
  - **A Depth-First-Search (DFS) strategy is often used**



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**

Example: 9-Value D-Algorithm



ch4-31

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

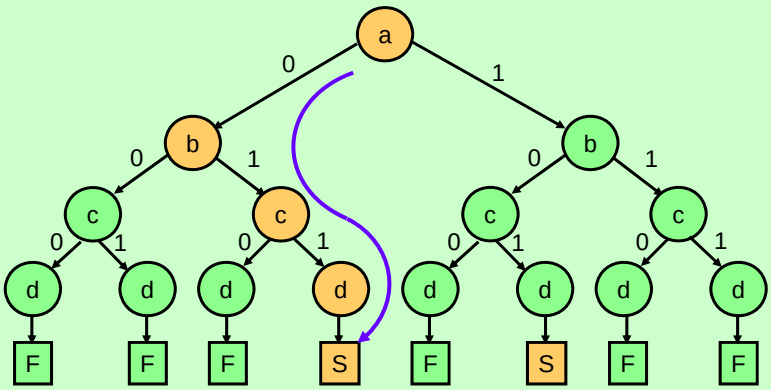
- **Test Generation (TG) Methods**
  - Based on Truth Table
  - Based on Boolean Equation
  - Based on Structural Analysis
  - D-algorithm [Roth 1967]
  - 9-Valued D-algorithm [Cha 1978]
  - PODEM [Goel 1981]
  - FAN [Fujiwara 1983]

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

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



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

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);
}
```

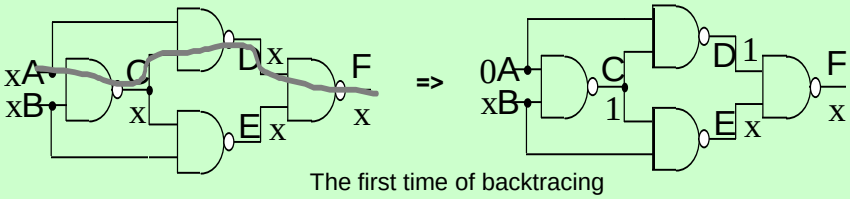

後追蹤 Backtrace Routine

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

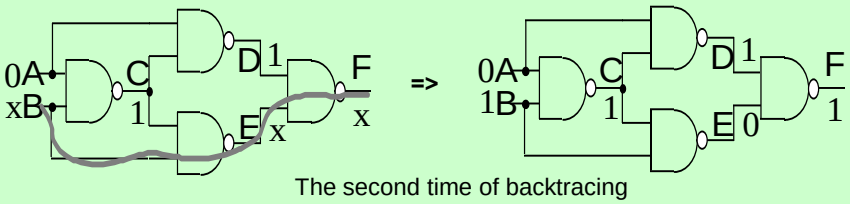
```
Backtrace(w, v_w) {
   /* Maps objective into a PI assignment */
   G = w;  /* objective node */
   v = v_w; /* objective value */
   while (G is a gate output) { /* not reached PI yet */
        inv = inversion of G;
        select an input (j) of G with value x;
        G = j;          /* new objective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

Example: Backtrace

Objective to achieve: (F, 1)
PI assignments:
(1) A = 0 → fail
(2) B = 1 → succeed



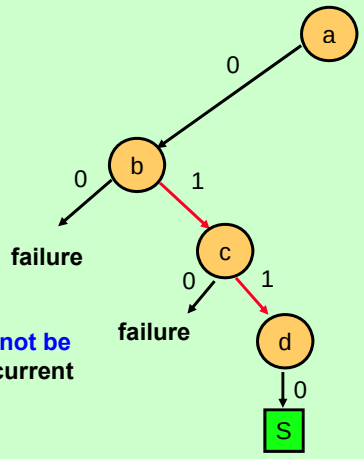
The first time of backtracing



The second time of backtracing

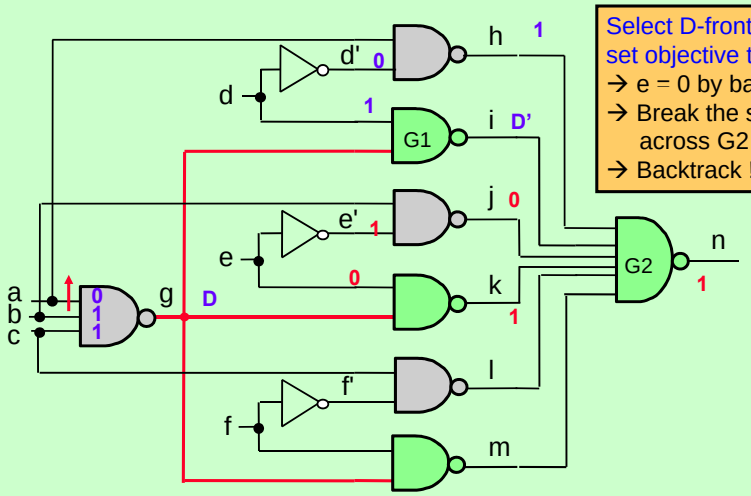
PI Assignment in PODEM

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

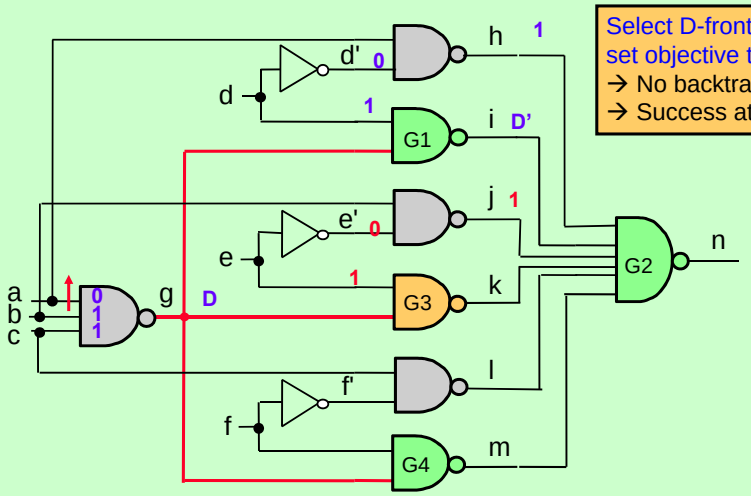
Example: PODEM (1/3)



Select D-frontier G2 and set objective to (k,1)
→ e = 0 by backtrace
→ Break the sensitization across G2
→ Backtrack !

ch4-41

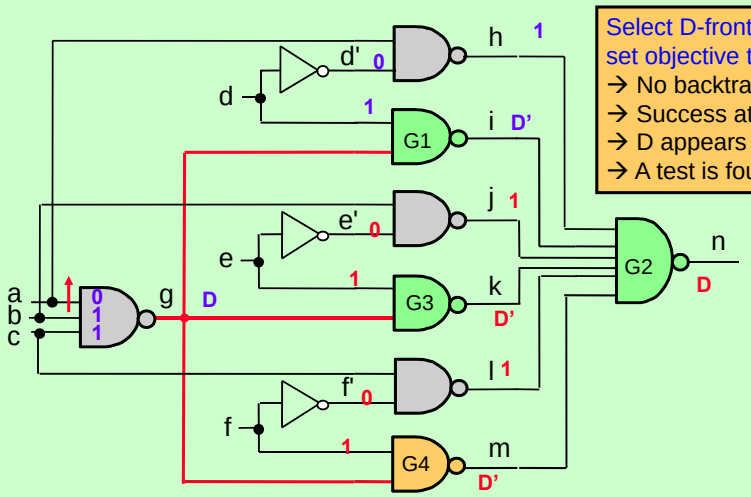
Example: PODEM (2/3)



Select D-frontier G3 and set objective to (e,1)
→ No backtrace is needed
→ Success at G3

ch4-42

Example: PODEM (3/3)

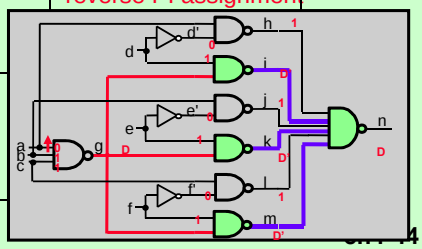


Select D-frontier G4 and set objective to (f,1)
→ No backtrace is needed
→ Success at G4 and G2
→ D appears at one PO
→ A test is found !!

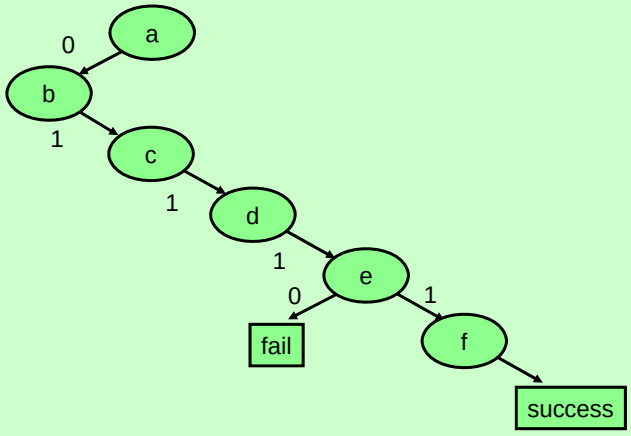
PODEM: Value Computation

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

Assignments need to be reversed during backtracking



Decision Tree in PODEM



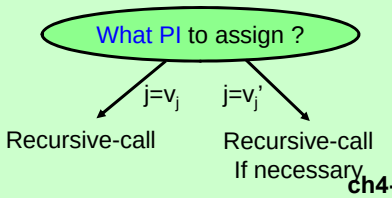
- Decision node: the PI selected through backtrace for value assignment
- Branch: the value assignment to the selected PI

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

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 */
    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
```



ch4-47

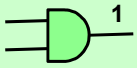
Overview of PODEM

- **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

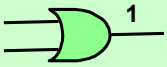
ch4-48

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

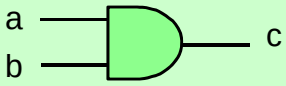
- **Controllability of a signal w**
  - CY1(w): the probability that line w has value 1.
  - CY0(w): the probability that line w has value 0.
  - Example:
    - f = ab
    - Assume CY1(a)=CY0(a)=CY1(b)=CY0(b)=0.5

      → CY1(f)=CY1(a)xCY1(b)=0.25,

      → CY0(f)=CY0(a)+CY0(b)-CY0(a)xCY0(b)=0.75
- **Example of Smart Backtracing**
  - Objective (c, 1) → choose path c→a for backtracing
  - Objective (c, 0) → choose path c→a 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)

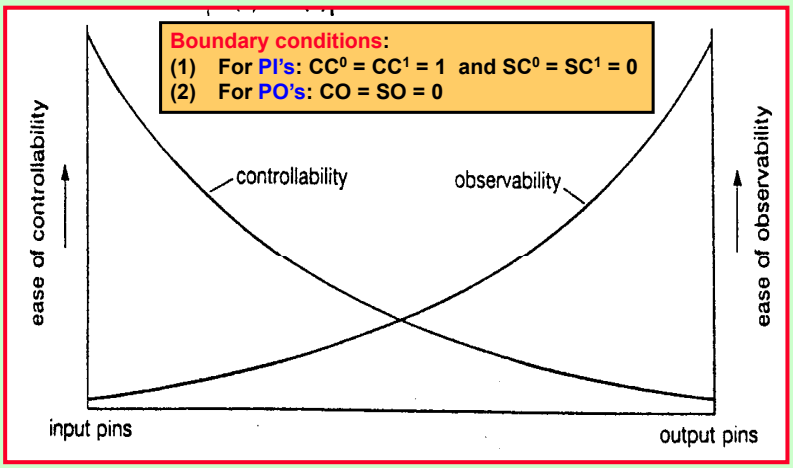
- **Computes six numbers for each node N**
  - $CC^0(N)$ and $CC^1(N)$
    - Combinational 0 and 1 controllability of a node N
  - $SC^0(N)$ and $SC^1(N)$
    - Sequential 0 and 1 controllability of a node N
  - CO(N)
    - Combinational observability
  - SO(N)
    - Sequential observability

值越大➔代表越困難

ch4-52

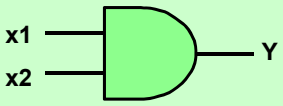
General Characteristic of Controllability and Observability

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



Controllability Measures

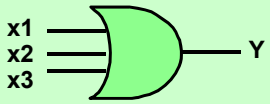
- $CC^0(N)$ and $CC^1(N)$
  - The number of combinational nodes that must be assigned values to justify a 0 or 1 at node N
- $SC^0(N)$ and $SC^1(N)$
  - The number of sequential nodes that must be assigned values to justify a 0 or 1 at node N



$$CC^0(Y) = \min [CC^0(x1), CC^0(x2)] + 1$$
$$CC^1(Y) = CC^1(x1) + CC^1(x2) + 1$$
$$SC^0(Y) = \min [SC^0(x1), SC^0(x2)]$$
$$SC^1(Y) = SC^1(x1) + SC^1(x2)$$

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

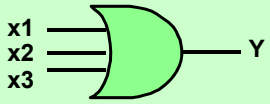


$$CC^0(Y) = CC^0(x1) + CC^0(x2) + CC^0(x3) + 1$$
$$CC^1(Y) = \min [\, CC^1(x1), CC^1(x2), CC^1(x3) \,] + 1$$
$$SC^0(Y) = SC^0(x1) + SC^0(x2) + SC^0(x3)$$
$$SC^1(Y) = \min [\, SC^1(x1) , SC^1(x2) , SC^1(x3) \,]$$

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

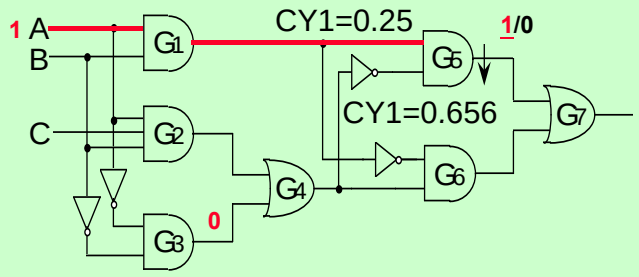
Example: X1 observable: (Y observable) + (side-inputs 配合)



$$CO(x1) = CO(Y) + CC^0(x2) + CC^0(x3) + 1$$
$$SO(x1) = SO(Y) + SC^0(x2) + SC^0(x3)$$

PODEM: Example 2 (1/3)

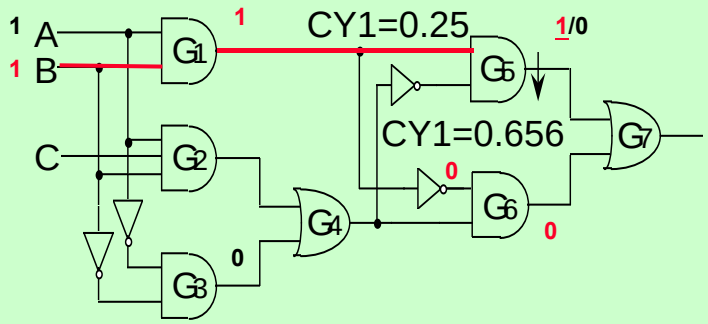
Initial objective=(G5,1).
G5 is an AND gate → Choose the hardest-1
→ Current objective=(G1,1).
G1 is an AND gate → Choose the hardest-1
→ Arbitrarily, Current objective=(A,1). A is a PI → Implication → G3=0.



ch4-57

PODEM: Example 2 (2/3)

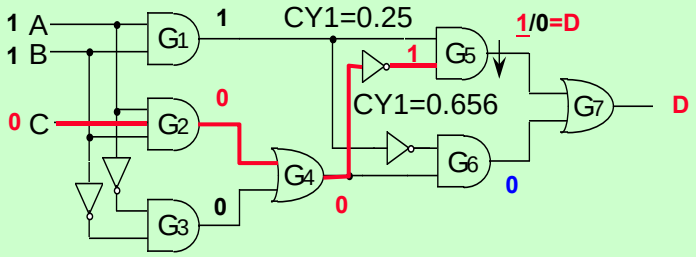
The initial objective satisfied? No! → Current objective=(G5,1).
G5 is an AND gate → Choose the hardest-1 → Current objective=(G1,1).
G1 is an AND gate → Choose the hardest-1
→ Arbitrarily, Current objective=(B,1). B is a PI → Implication → G1=1, G6=0.



ch4-58

PODEM: Example 2 (3/3)

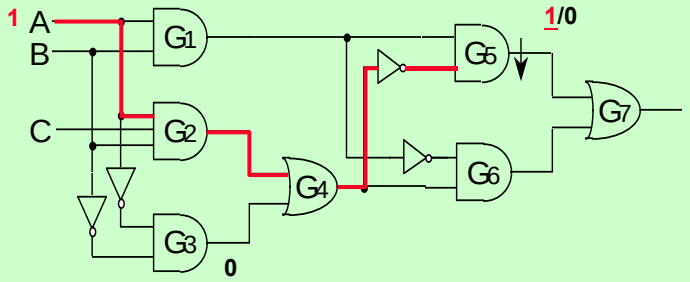
The initial objective satisfied? No! → Current objective=(G5,1).
The value of G1 is known → Current objective=(G4,0).
The value of G3 is known → Current objective=(G2,0).
A, B is known → Current objective=(C,0).
C is a PI → Implication → G2=0, G4=0, G5=D, G7=D.



No backtracking !!

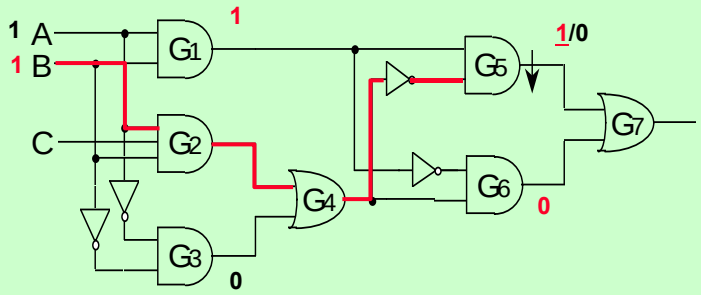
If The Backtracing Is Not Guided (1/3)

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.



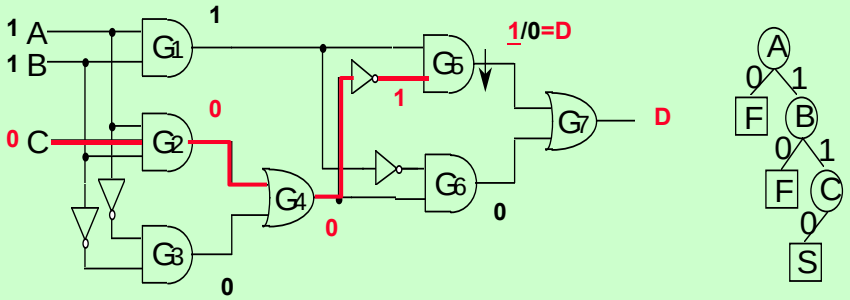
If The Backtracing Is Not Guided (2/3)

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.



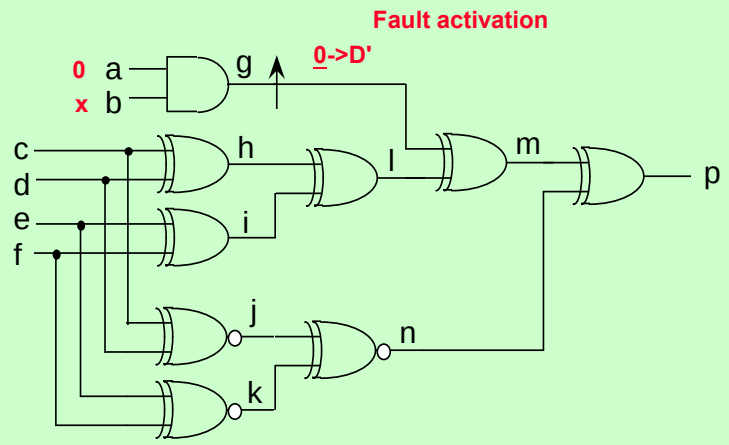
If The Backtracing Is Not Guided (3/3)

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.

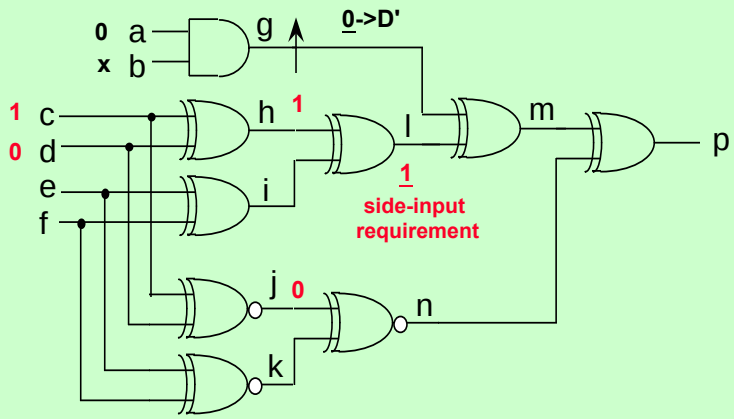


Two times of backtracking !!

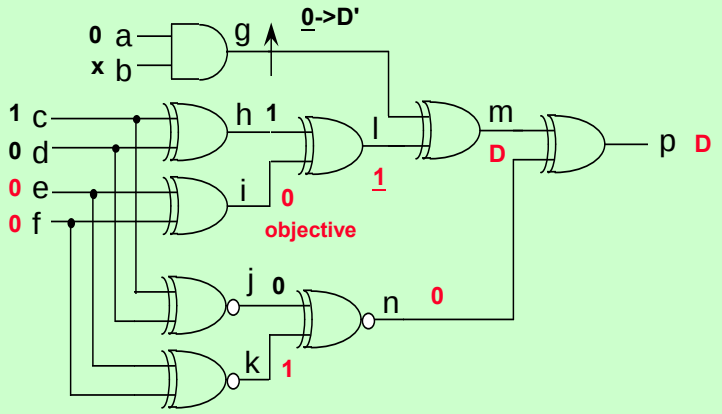
ECAT Circuit: PODEM (1/3)



ECAT Circuit: PODEM (2/3)



ECAT Circuit: PODEM (3/3)



No backtracking !!

Outline

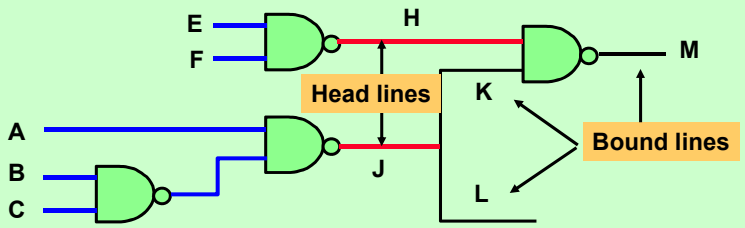
- **Test Generation (TG) Methods**
  - Based on Truth Table
  - Based on Boolean Equation
  - Based on Structural Analysis
  - D-algorithm [Roth 1967]
  - 9-Valued D-algorithm [Cha 1978]
  - PODEM [Goel 1981]
  - FAN [Fujiwara 1983]

FAN (Fanout Oriented) Algorithm

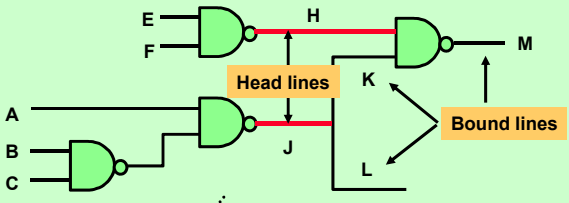
- **FAN**
  - Introduces two major extensions to PODEM's backtracing algorithm
- **1st extension**
  - Rather than stopping at PI's, backtracing in FAN may stop at an internal lines
- **2nd extension**
  - FAN uses multiple backtrace procedure, which attempts to satisfy a set of objectives simultaneously

Headlines and Bound Lines

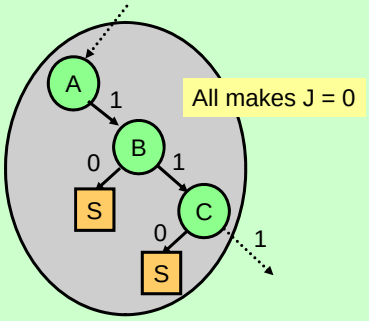
- **Bound line**
  - A line reachable from at least one stem
- **Free line**
  - A line that is NOT bound line
- **Head line**
  - A free line that directly feeds a bound line



Decision Tree (PODEM v.s. FAN)

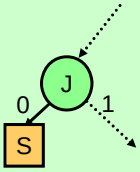


Assume that:
Objective is (J, 0)



PODEM

J is a head line
→ Backtrace stops at J
→ Avoid unnecessary search



FAN

ch4-69

Why Stops at Head Lines ?

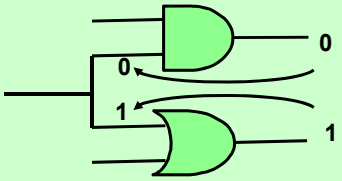
- **Head lines are mutually independent**
  - **Hence, for each given value combination at head lines, there always exists an input combination to realize it.**
- **FAN has two-steps**
  - **Step 1: PODEM using headlines as pseudo-PI's**
  - **Step 2: Generate real input pattern to realize the value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

ch4-70

Why Multiple Backtrace ?

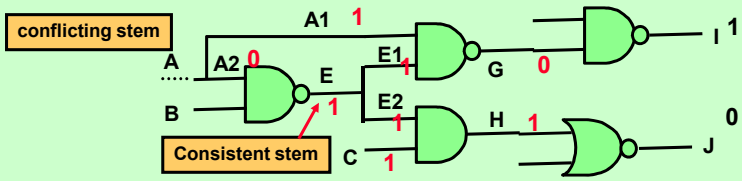
- **Drawback of Single Backtrace**
  - A PI assignment satisfying one objective →may preclude achieving another one, and this leads to backtracking
- **Multiple Backtrace**
  - Starts from a set of objectives (Current_objectives)
  - Maps these multiple objectives into a head-line assignment $k=v_k$ 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 ≠ϕ ) {
        remove one entry (k, v_k) from Current_objectives;
        switch (type of entry) {
        1. HEAD_LINE:   add (k, v_k) to Head_objectives;
        2. FANOUT_BRANCH:
                j = stem(k);
                increment no. of requests at j for v_k; /* 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 ((v_k ⊕ 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 ≠ϕ ) {body in previous page}
    if(Stem_objectives≠ϕ ) {
        remove the highest-level stem (k) from Stem_Objectives;
        v_k = most requested value of k;
        /* recursive call here */
        add (k, v_k) to Current_objectives;
        return (Mbacktrace(Current_objectives);
    }
    else {      remove one objective (k, v_k) from Head_objectives;
                return (k, v_k)
    }
}
```

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.