# DIGITAL SYSTEM TESTING
# COE -545

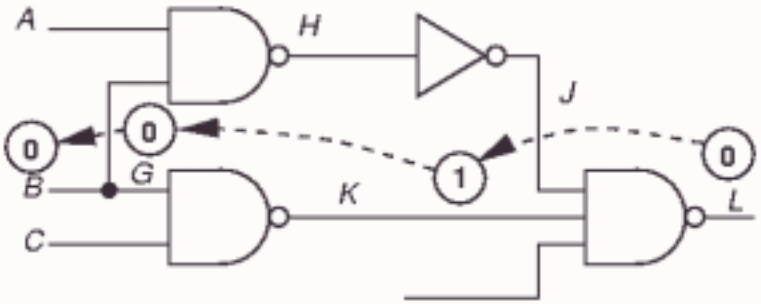
**Lecture - 10**

## FAN -- *FANout-Oriented ATPG Algorithm* (Fujiwara and Shimono -1983)

- **FAN Is a Podem-based ATPG Algorithm That Adds Several Speed-up Heuristic Strategies**
- **New concepts:**
  - **Immediate Implication (*Uniquely-Implied signals* → *Both Forward & Backward* )**
  - ***Unique sensitization***
  - **Stop Backtrace at *Headlines***
  - ***Multiple Backtrace* (*Satisfy Multiple Objectives* )**

## Immediate Impication of Unique Signals

### PODEM Fails to Determine Unique Signals

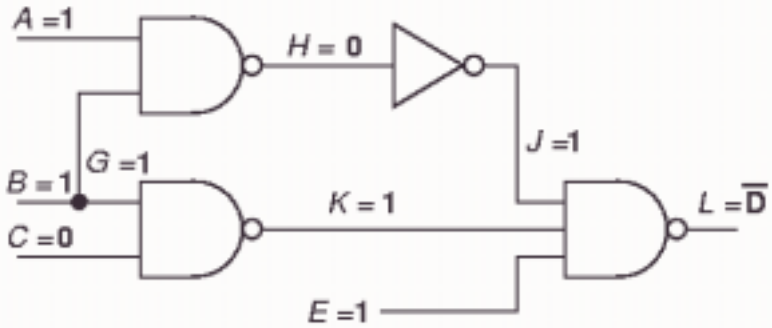



- **Backtracing operation fails to set all 3 inputs of gate *L* to 1**
  - **(L, 0) → (K,1) → (B, 0) → IMPLY L=1 → Failure →PI-Remake B=1**
  - **Causes unnecessary search**



## Immediate Impication of Unique Signals

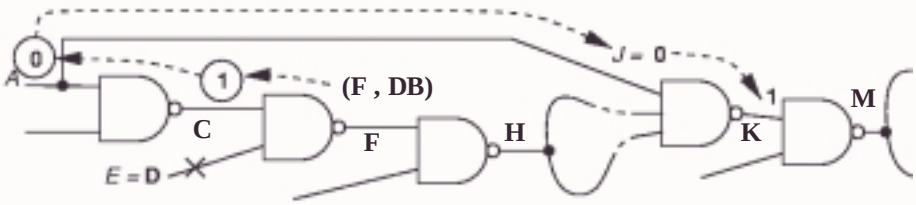
### FAN -- Early Determination of Unique Signals




- **Determine all unique signals *Implied* (*Forward & Backward*) by current decisions immediately**
- **Objective, L=DB → Assign L=0 →**
  **Imply J=K=E=1, A=B=1, C=0**
  - **Avoids unnecessary search**

## Unique Sensitization

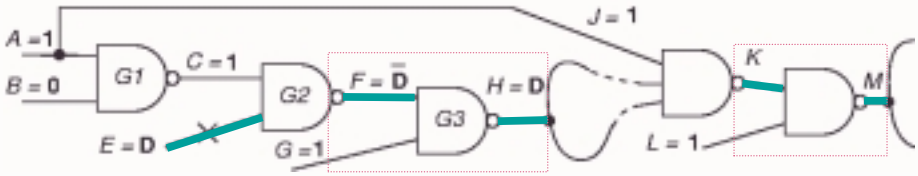
### PODEM Makes Unwise Signal Assignments




- **Initial Objective: (F, DB) → (C, 1) → A = 0 → Initial Objective Met, But Test is Not Possible ( K=1 )**
- **Fault propagation Is not Possible→ Faulty Signals Must Propagate along Path Segments K & M Which are Blocked By the assignment J=0**



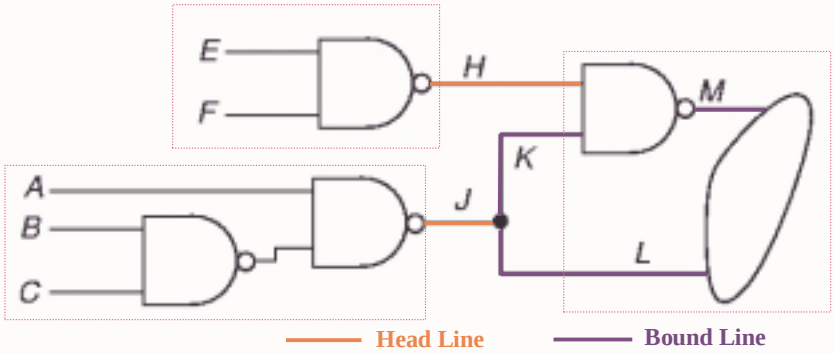
## Unique Sensitization of FAN with No Search




Path over which fault is uniquely sensitized

- Every Path From G2 Passes Through **F-H** and **K-M**
- Might As Well Go Ahead and Sensitize It. If the Resulting Internal Objectives Cannot Be Met, We Find Out Early.
- FAN Immediately Sets Necessary Signals to Propagate Fault/Sensitize Path (C=1, G=1, J=1, L=1)

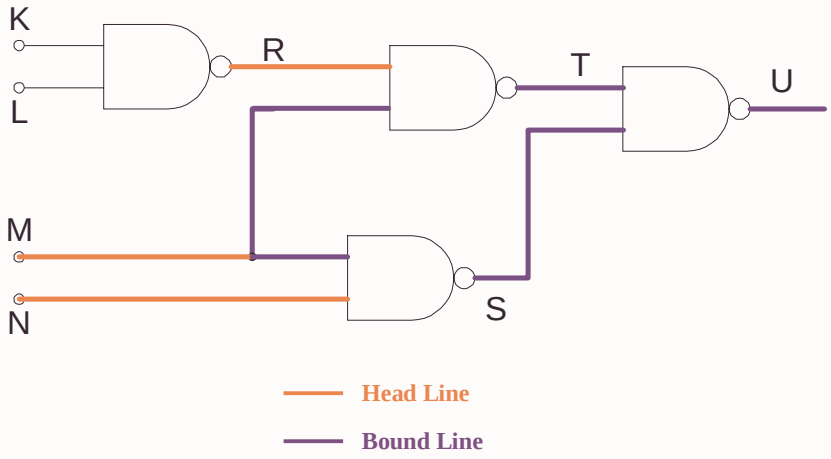
## Headlines



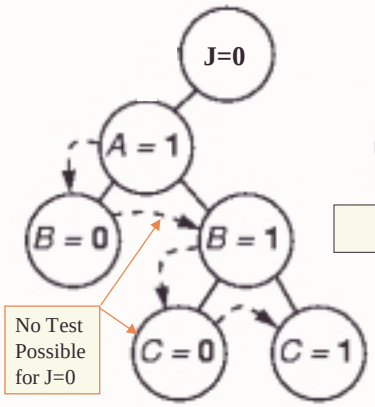

- Bound Line: Line Fed (Directly or Indirectly) by Fanout Stem
- Free Line: Line that is Not Bound
- Head Line: A Free Line Directly feeding a Bound Line
  - Output line of a *fanout-free subcircuit*, and
  - Its successor Gates are fed by signals from regions containing fanout.
- Head Lines *H* and *J* separate circuit into 3 parts, for which test generation can be done independently



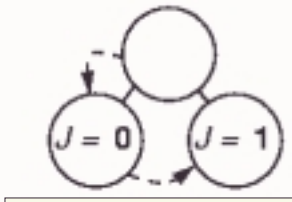
## Example



## Stop Backtrace at Head Lines




PODEM decision tree *for* (J,0)

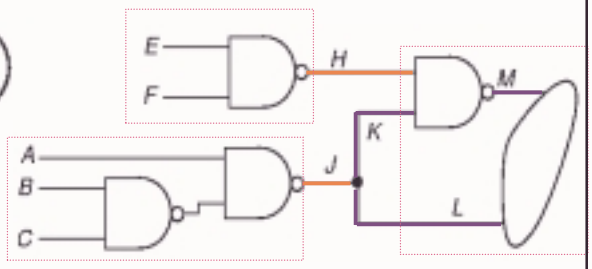



FAN decision tree

**Example**

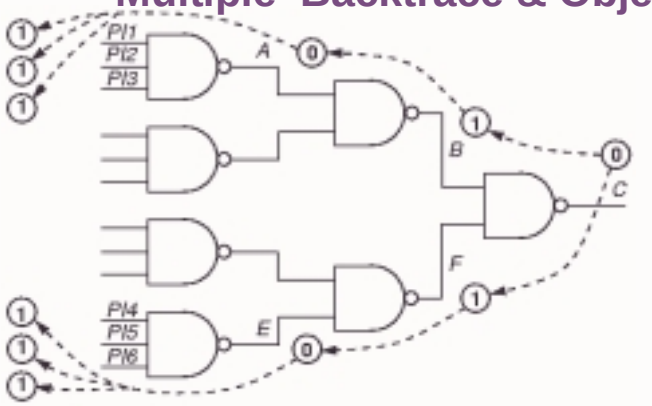
**Current Objective *(J, 0)***

**Leads to Inconsistency**






## Multiple Backtrace & Objectives




**FAN – Breadth-First passes – 1 time**

**PODEM – Depth-First passes – 6 times**

- At **fanout points**, FAN compares the incoming number of 0 and 1 requests and assigns the larger value
- Objectives become *triples*: (l, v) → (l , $n_0$(l), $n_1$(l) )

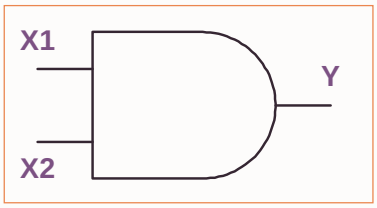
## Multiple Backtrace & Objectives

- Multiple Backtrace:
  - Starts with More than One Objective
  - Works in *Breadth-First* Manner from Initial Objectives
  - Works Backwords Towards *Head Lines*

**AND Gate**

X1 → *Easier* To Control Input

X2 → *Harder* To Control Input




- Starting From Output Objective (Y, $n_0(Y)$, $n_1(Y)$ ), the Input Objectives are Obtained as follows :
  - $n_0(x1) = n_0(Y)$ && $n_1(x1) = n_1(Y)$
  - $n_0(x2) = 0$ && $n_1(x2) = n_1(Y)$

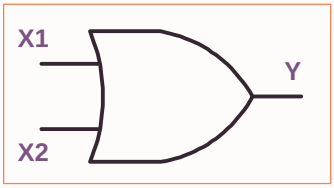


## Multiple Backtrace & Objectives

**OR Gate**

X1 → *Easier* To Control Input

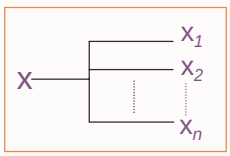
X2 → *Harder* To Control Input



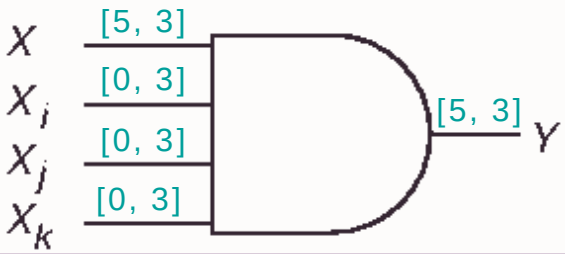

- Starting From Output Objective (Y, $n_0(Y)$, $n_1(Y)$ ), the Input Objectives are Obtained as follows :
  - $n_0(x1) = n_0(Y)$ && $n_1(x1) = n_1(Y)$
  - $n_0(x2) = n_0(Y)$ && $n_1(x2) = 0$

**Fanout Node**

- $n_0(x) = \sum n_0(x_i)$ && $n_1(x) = \sum n_1(x_i)$



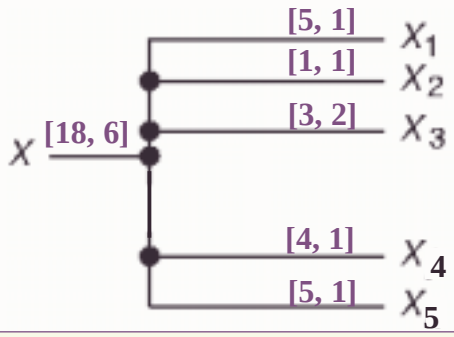
## AND Gate Vote Propagation




- **AND Gate**
  - **Easiest-to-control Input –**
    - **# 0's = OUTPUT # 0's**
    - **# 1's = OUTPUT # 1's**
  - **All other inputs --**
    - **# 0's = 0**
    - **# 1's = OUTPUT # 1's**

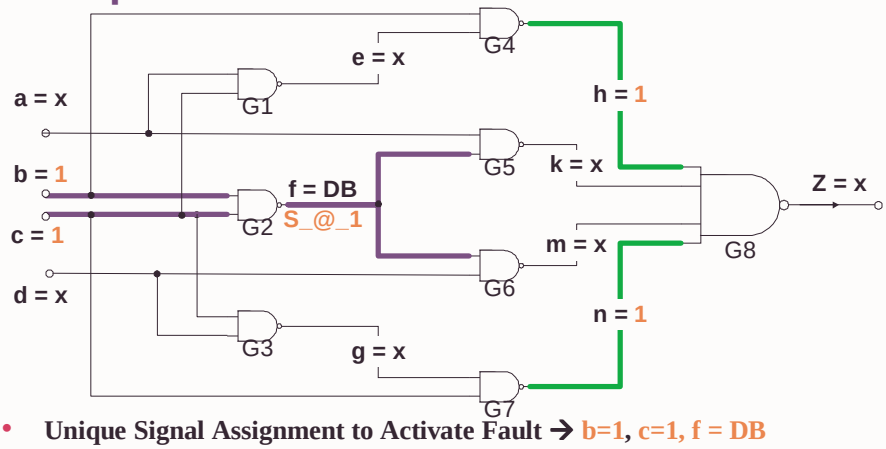


## Multiple Backtrace Fanout Stem Voting




- **Fanout Stem --**
  - **# 0's = Σ Branch # 0's,**
  - **# 1's = Σ Branch # 1's**

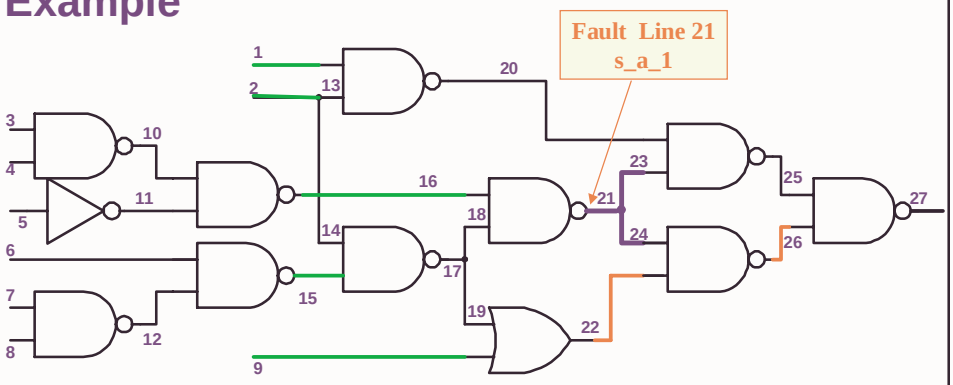
## Example




- Unique Signal Assignment to Activate Fault → b=1, c=1, f = DB
- Unique Path Sensitization → h=1 & n=1 → Justify By Initial Multiple Objective (h, 0, 1) & (n, 0, 1) → Multiple Backtrace → a= 1, d= 1
- Forward Implicate Results in Z = DB

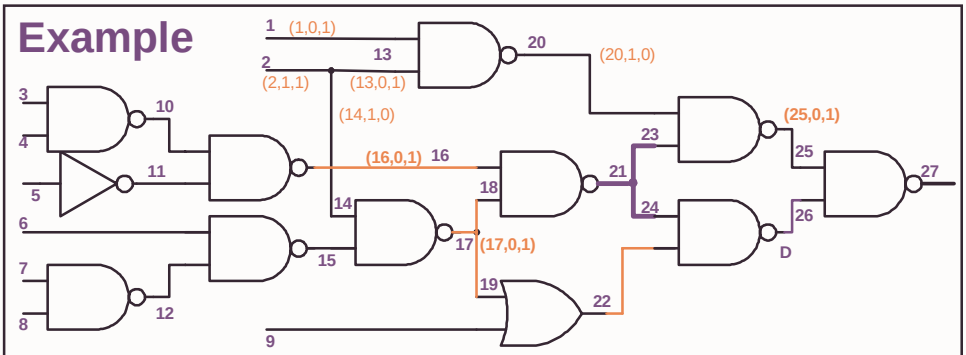


## Example




- Head Lines: {1, 2, 15, 16, 9}
- Fault Activation Unique Assignment → 16=1 && 17, 18, 19=1
- Unique Implication → 22=1 → 26 = D
- Fault Propagation D-Frontier = {G25, G27} → G27 (Higher Observability)
- Multiple Backtrace: Initial Objectives {(25,0,1) ,(16,0,1) , (17, 0, 1)}

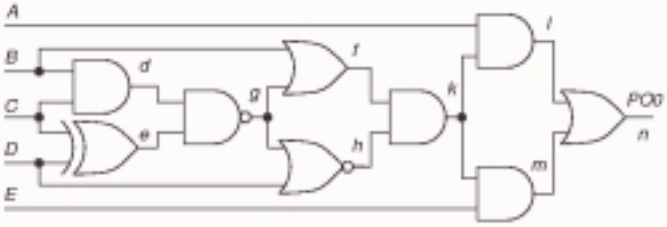
## Example




- **Head Lines:** **{1, 2, 15, 16, 9}**
- **Initial Objectives {(25,0,1) ,(16,0,1) , (17, 0, 1)} → Mapped Into Set of Headline Objectives {(16, 0, 1) , (2, 1, 1) , (1, 0, 1) }**
- **Stem 2 Has Contradictory Requirement (** $n_0$ , $n_1 \neq 0$) → Since $n_0 = n_1$ → Pick Line 2 = 0 (Choice)
- Forward Implication → **27 = DB → Solution Reached**
- **Justify Head Line 16 to 1 → Line 11 = 0 → Line 5=1**

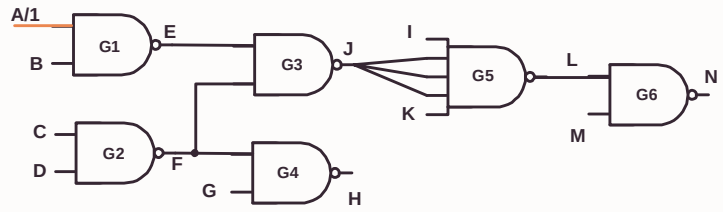


## TOPS – Dominators Kirkland and Mercer (1987)




- **Improves Immediate Assignments over FAN Using Dominators**
- ***Dominator* of *g* – all paths from *g* to PO must pass through the dominator**
  - ***Absolute*** -- ***k*** *dominates* ***B*** *(All Paths from B to ALL POs Pass Through k)*
  - ***Relative* – Dominates only the paths to a** *given* **PO**
  - **If Dominator of the fault becomes 0 or 1, backtrack**

## TOPS – Dominators Mandatory Assignment




- **Dominators: E, J, L, N**
- **All *OFF-Path* Signals of Dominators Should be Assigned Non-Controlling Values**
- ***B=1, F=1, I=1, K=1, M=1***



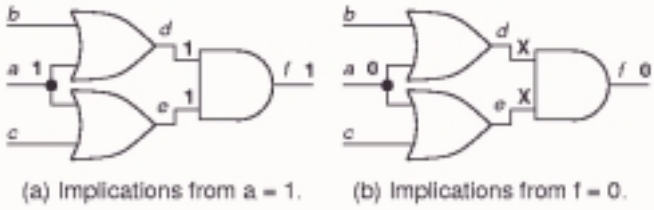
## SOCRATES- Structure-Oriented Cost-Reducing Automatic TESt pattern generation [Schultz et al. 1988]

- An ATPG "System"—not just a test generation algorithm
- Main Features
  - "*Learning-based*" ATPG
  - *More complex Implications* (= Learning)
  - Seeding with random tests
  - Fast combinational fault simulation built-in
  - More complex unique sensitization
  - Allows higher-level primitives

## SOCRATES Learning

**Contrapositive Law**: **(P ⇒ Q) ≡ (!Q ⇒ !P)**

**Example**




- **IF** $(a = 1) \Rightarrow (f = 1)$, **THEN** $!(f = 1) \Rightarrow !(a = 1)$

**Accordingly,**

*IF* $(a = 1) \Rightarrow (f = 1)$ we **Learn** that $(f = 0) \Rightarrow (a = 0)$

- **Set Each Signal First to 0, and Then to 1**
- **Discover *implications***



## Fault-Independent Test Generation
## Critical Path Tracing

**Objective:** Generate TVs that Detect SSL Faults along Some Critical Path → The Longer the Path The Better.

**Advantages**:

- "Fault-independent" ATPG Derives TVs Which Detect Many Faults, *not just targeting a specific fault.*
- New Tests are Easily Generated by Modifying Existing Ones → Avoids Duplicated Effort Inherent in Fault-Oriented ATPG

**Def:** A Line *x* has a *Critical Value v* in some Test Vector *T*, IFF *T* Detects the Fault *x*/*v*.

**Def:** A Gate Input is *Critical* in some Test Vector *T*, IF Complementing its Value Changes the Value of the Gate Output.

## Fault-Independent Test Generation
## Critical Path Tracing

**Algorithm:**

1. • Select a PO and assign it a (critical) value $v \in \{0, 1\}$
2. • Recursively Justify $v$ by assigning *Critical* values to Gate Inputs
3. Repeat Step 2 for Critical Value $\overline{v}$
4. Repeat the Procedure for other Pos.

wherever possible

**Notes:**

- Values of PO's are Always Critical
- To Justify a Critical Value, *Critical Cubes* , rather than *Primitive Cubes*, are Used.



## Fault-Independent Test Generation
## Critical Path Tracing

**Example:** Line Justification for an AND-Gate

| A | B | C | Z |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 0 | *x* | *x* | 0 |
| *x* | 0 | *x* | 0 |
| *x* | *x* | 0 | 0 |

**Line Justification By Primitive Cubes (Singular Covers)**

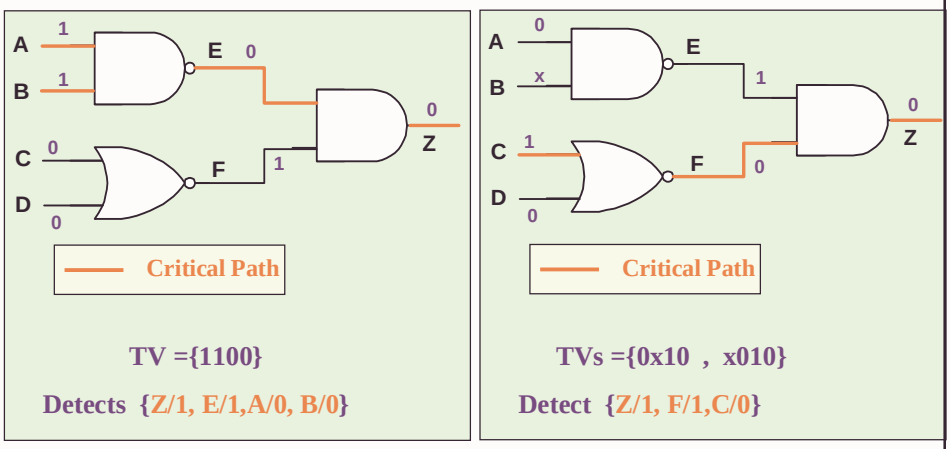
Critical Values

| A | B | C | Z |
|---|---|---|---|
| **1** | **1** | **1** | 1 |
| **0** | 1 | 1 | 0 |
| 1 | **0** | 1 | 0 |
| 1 | 1 | **0** | 0 |

**Line Justification By Critical Cubes (Singular Covers)**

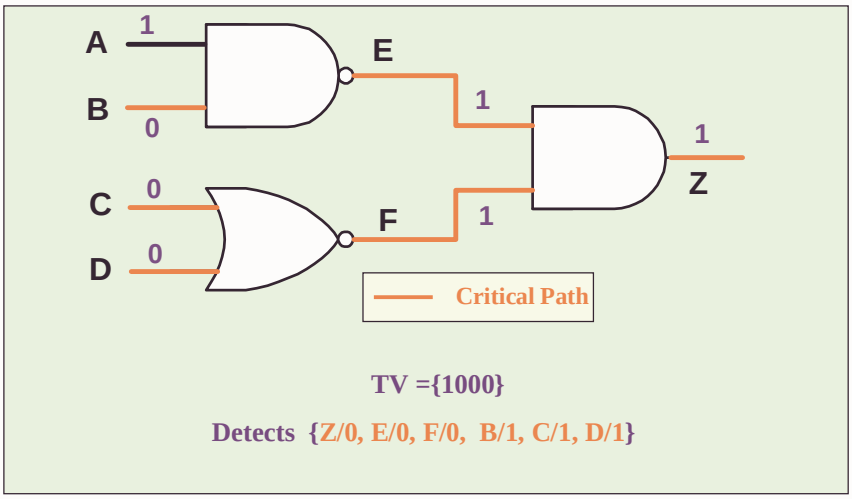
## Fault-Independent Test Generation
## Critical Path Tracing

**Example:** Fanout-Free Circuits






## Fault-Independent Test Generation
## Critical Path Tracing



## Fault-Independent Test Generation
## Critical Path Tracing

**Problems with Reconvergent Fanout Circuits:**

1. Conflicts
   - Line Justification Uses *Critical (not primitive) Cubes* → More Values Specified → Leads to Conflicts
   - If Line Justification Using CC's Fails → Use Primitive Cubes till the Conflict is Overcome
2. self- masking
   - A *Fanout Branch* which is *Critical* → Doesn't Necessarily Imply that the *Stem* is Also *Critical.* This is Solved as Follwos:
     - *Simulate the Effect of the Stem Fault*
     - *Rely on the Seldomness of Self-Masking and Continue TG Process by Assuming that the Stem Fault is Detected*

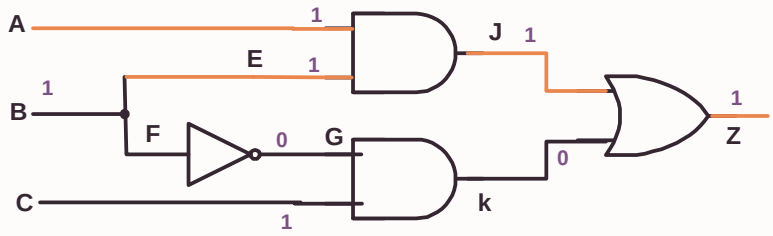


## Fault-Independent Test Generation
## Critical Path Tracing

**Problems with Reconvergent Fanout Circuits:**

3. May Fail to Detect Faults Detectable Only by multiple- path sensitization (*Since Critical Cubes are Used not SC*) _
4. *overlap among PO cones* (*Same Line Justification Problem May Be Encountered More than Once*)

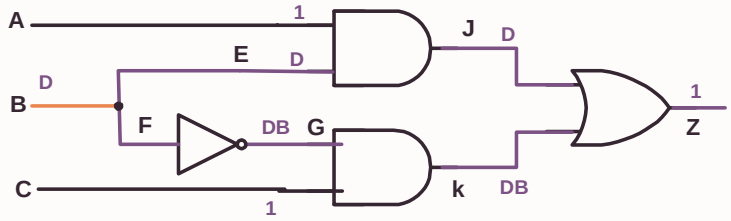
## CPT- Example




- **TV = {111}, Detects SSL Faults: Z/0, J/0, A/0, E/0**
- **The Fanout Stem B Cannot Be Ascertained as Critical w/out Further Analysis (Since Reconverging Paths May Mask Fault at Stem)**



## CPT- Example




**For the Fault B/0**

- **TV = {111} → Z = 1 → Fault Undetectable even though E/0 is Detectable → *Self-Masking***
- **To Determine whether a Fanout Stem is Critical a Simulation Step May be needed To the First Capture Line**