The D-Algorithm

- An Algorithmic Approach which Generates a TV for a Given Fault if One Exists
- Multiple Path Sensitization
- 5-Valued Logic { 0, 1, X, D, $\overline{D}$}
- **D**: *A Line is Assigned a* **D** *value if it has a value* ***1*** *in the* ***Fault-Free*** *Circuit but has a 0 Value in the Faulty Circuit.*
  (**D** ≈ S_@_0)
- $\overline{\mathbf{D}}$: *A Line is Assigned a* $\overline{\mathbf{D}}$ *value if it has a value* ***0*** *in the* ***Fault-Free*** *Circuit but has a 1 Value in the Faulty Circuit.*
  ($\overline{\mathbf{D}}$ ≈ S_@_1)
- D / $\overline{D}$ Follow Rules of Boolean Algebra

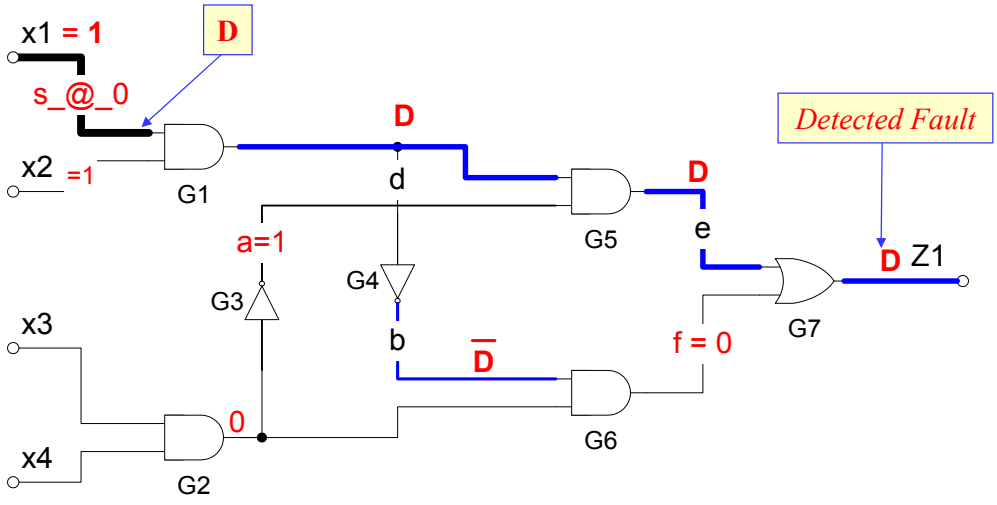
The D-Algorithm

$D + D = D$	$\overline{D} + \overline{D} = \overline{D}$
$D \cdot D = D$	$\overline{D} \cdot \overline{D} = \overline{D}$
$\overline{D} + D = 1$	$D \cdot \overline{D} = 0$
$D \cdot 1 = D$	$\overline{D} \cdot 1 = \overline{D}$
$D \cdot 0 = 0$	$\overline{D} \cdot 0 = 0$
$D + 1 = 1$	$\overline{D} + 1 = 1$
$D + 0 = D$	$\overline{D} + 0 = \overline{D}$

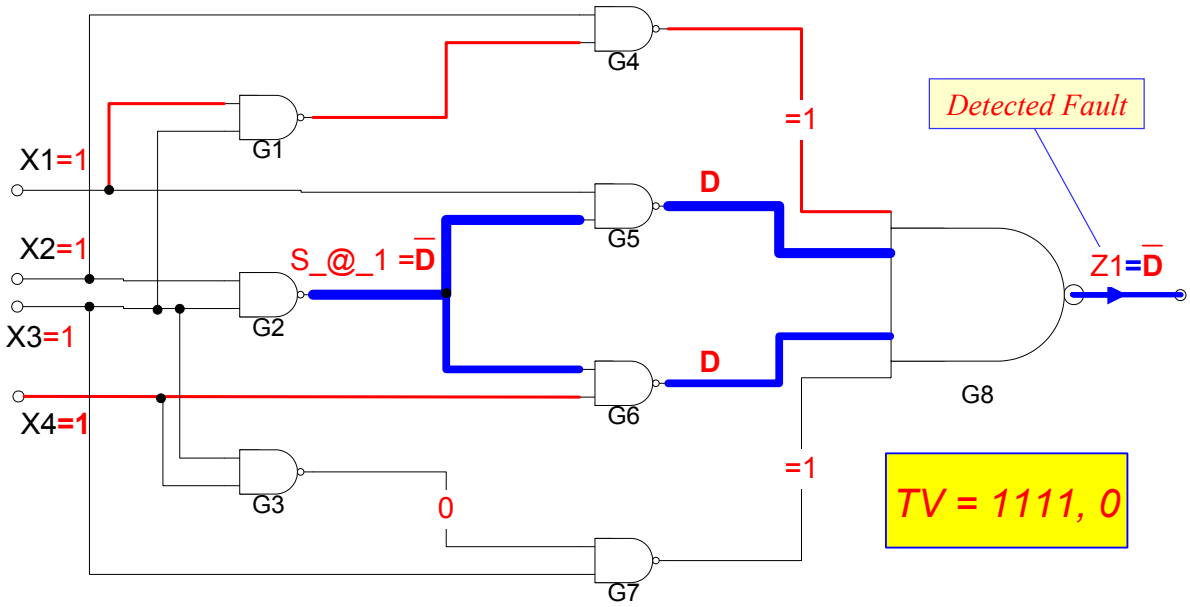
•	0	1	D	$\overline{D}$	x
0	0	0	0	0	0
1	0	1	D	$\overline{D}$	x
D	0	D	D	0	x
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$	x
x	0	X	x	x	x

AND Operation of the 5-Valued D-Calculus

Example - Single Path Sensitization



Example - Multiple Path Sensitization



Definitions

1. Singular Covers (SC) of Some Function F
   (*Primitive Cubes of F*) : *Minimal Set of Logic Signal Assignments Showing Essential Prime Implicants*

   = Prime Implicants of $\overline{F}$ ($\alpha 1$) &&
   Prime Implicants of $\overline{F}$ ($\alpha 0$)

Examples Singular Covers of 2-Input NAND Gate

A	B	F
0	X	1
X	0	1
1	1	0

PIs of F ($\alpha 1$)

Pis of $\overline{F}$ ($\alpha 0$)

Definitions-- Singular Covers (SC)

(*Primitive Cubes*)

Example

A	B	C	F
X	X	1	1
0	0	X	1
X	1	0	0
1	X	0	0

PIs of F ($\alpha 1$)

PIs of $\overline{F}$ ($\alpha 0$)

SCs of F

BC

A	00	01	11	10
0	1	1	1	
1		1	1	

K-Map of F

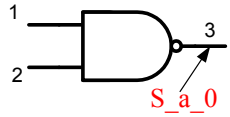
Definitions

2. Primitive D-Cubes of a Fault (PDCF)

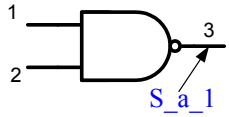
= Prime Implicants of the *Faulty* Function which Produce $D / \overline{D}$ Output

{I/P. Stimuli Required to Activate Faulty Condition at a Gate/Module Output}

Examples PDCF of 2-Input NAND Gate S_a_0 && S_a_1



1	2	3
0	X	D
X	0	D



1	2	3
1	1	$\overline{D}$

Definitions -- *Primitive D-Cubes of a Fault (PDCF)*

Computing PDCF of a General Module (Function)

1. Obtain the SCs of the Fault-Free Function ($\alpha 1$, $\alpha 0$)
2. Obtain the SCs of the Faulty Function ($\beta 1$, $\beta 0$)
3. PDCF for this module Result from Intersecting $\alpha 1$ with $\beta 0$ and $\alpha 0$ with $\beta 1$.

Thus PDCF = $\{ \alpha 1 \cap \beta 0 \ , \ \alpha 0 \cap \beta 1 \ \}$

Definitions-- -- *Primitive D-Cubes of a Fault (PDCF)* -- Example

Example

A	B	C	F
X	X	1	1
0	0	X	1
X	1	0	0
1	X	0	0

PIs of F ($\alpha 1$)

PIs of $\bar{F}$ ($\alpha 0$)

A	B	C	F
X	X	1	1
0	1	X	1
X	0	0	0
1	X	0	0

PIs of F_α ($\beta 1$)

PIs of $\bar{F}_\alpha$ ($\beta 0$)

PDCF

A	B	C	F
0	0	0	D
0	1	0	$\bar{D}$

$\alpha 1 \cap \beta 0$

$\alpha 0 \cap \beta 1$

Definitions

3. Propagation D-Cube of a Gate/Module (PDC)

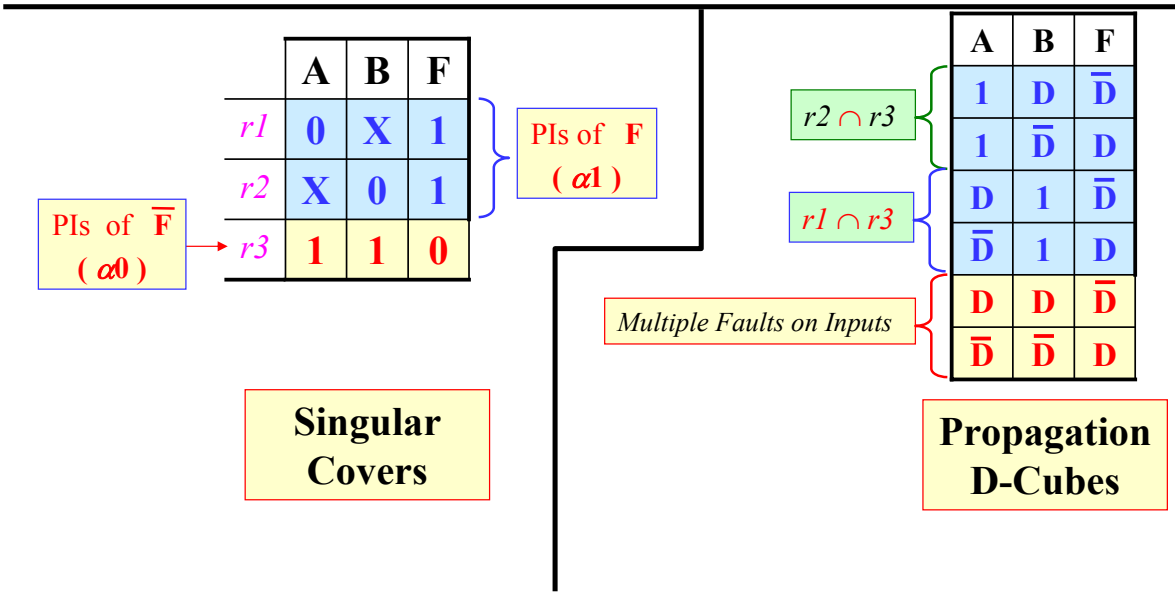
   = Prime Implicants of the Function which Allow a D/$\bar{D}$ Values on Inputs to Propagate to the Output

- Let { $\alpha 0$, $\alpha 1$ } be the Singular Covers of the Function F, Where;
  - $\alpha 1$ = the Prime Implicants of F
  - $\alpha 0$ = the Prime Implicants of $\bar{F}$

$$PDC = \{ \alpha 1 \cap \alpha 0 \}$$

Definitions-- *Propagation D-Cube (PDC)*

Example PDCF of 2-Input NAND Gate



Intersection of D-Cubes

- Let A = $(a_1, a_2, a_3, \ldots, a_n)$, and
- Let B = $(b_1, b_2, b_3, \ldots, b_n)$ be 2 D-Cubes, where

$$a_i, b_i \in \{0, 1, x, D, \overline{D}\}$$

- ***The D-intersection*: of A and B, denoted $A \cap B$ is given by $(a_i \cap b_i \mid a_i \in A, b_i \in B)$, where :**

1. $x \cap a_i = a_i$
2. If ($a_i \neq x$ and $b_i \neq x$) Then

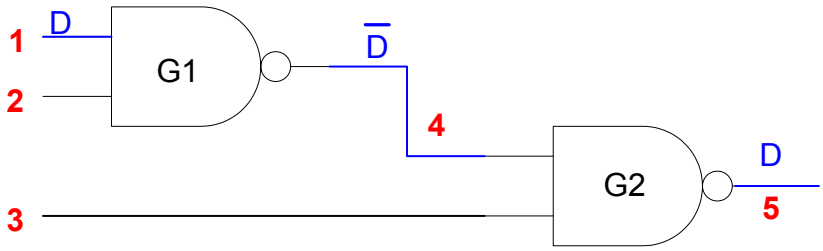
$$a_i \cap b_i = \begin{cases} a_i & IF \quad a_i = b_i \\ \Phi & IF \quad a_i \neq b_i \end{cases}$$

3. $A \cap B = \Phi$ "Empty Cube" IF $a_i \cap b_i = \Phi$ For Any i

D- Intersection

- D-Intersection is Used to Generate Sensitized Paths (**D-Drive**)

Example PDC of 2-Input NAND Gate →



A	B	F
1	D	D̄
1	D̄	D
D	1	D̄
D̄	1	D
D	D	D̄
D̄	D̄	D

1	2	3	4	5
D	1	*x*	D̄	*x*

∩

1	2	3	4	5
x	*x*	1	D̄	D

=

1	2	3	4	5
D	1	1	D̄	D

D- Algorithm

1. Initialize Test Cube to all *x*'s (*x, x, x, … x*)
2. Fault Provoking → Select a Primitive D-Cube of the Fault (PDCF) → Usually a *Choice Step*
3. Path Sensitization From the Faulty Line To a **PO**
   - ➢ Successive X-ion of Test Cube with the Propagation D-Cubes of Successor Gates (*D-Frontier*) Till a **PO** Gets a **D** / **D̄** Value (*Choice Step*) → This Step is known as *D-Drive*
   - ➢ This Step Requires *2 Major Procedures*

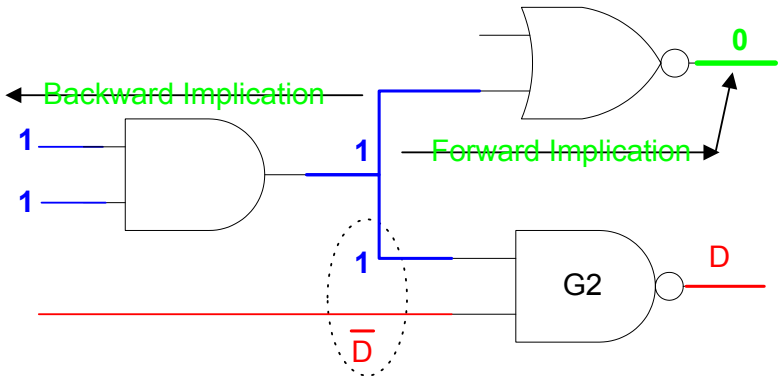
   a. Implication (Forward and Backward), and

   b. Line Justification (***Consistency Checks***)→ May Lead to *Backtracks* if Inconsistencies are Detected.

Implication

- Assignments Made Due to Choices, e.g.a Propagation D-Cube, Usually Uniquely Imply Other Signal Values {Forward & Backward}

Example



Chosen Propagation Cube

D-Frontier

- The D-Frontier Consists of **ALL** Gates whose Current O/P Value is "***x***" but have one or more **D/$\overline{\mathbf{D}}$** on their inputs
- One of the D-Frontier Gates is Usually Chosen to Propagate The Fault → "D-Drive"

Line Justification

Is a Backward *Implication Step* Where The Gate I/P Values are Selected *To Justify* the Specified Gate Output. This Step is Repeated Till the Relevant **PIs** are Defined.

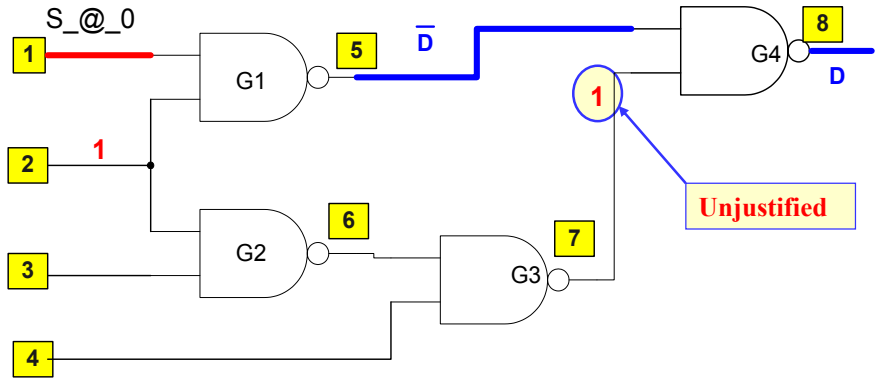
- *Line Justification* is Performed after D/$\overline{\text{D}}$ Appear at Some **PO**.

Line Justification-Example

- An Unjustified Line is a Defined Gate Output which is not Implied By The Gate Inputs

J-Frontier

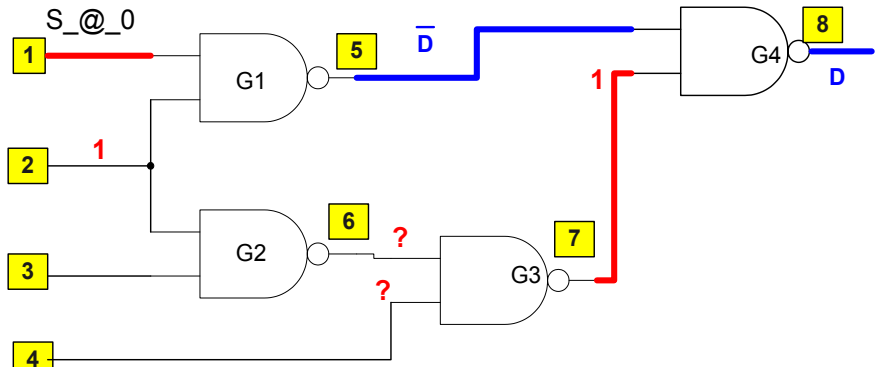
Is the Set of ALL Gates whose Output Lines are *Unjustified*



Line Justification

Example (*Line Justification*) → Line 1 S_@_0

$t^0 = PDFC^1$



1	2	3	4	5	6	7	8
D	1			$\overline{D}$			
D	1			$\overline{D}$		1	D

D-Drive

J-Frontier = {**G3**}"*Line 7 Unjustified*"

Either Line 4 = 0, OR

Line 6=0

	1	2	3	4	5	6	7	8	
$t^2=t^1 \cap Sc^3$	D	1	*x*	*x*	$\overline{D}$	0	1	D	J_F={G_2}
$t^3=t^2 \cap Sc^2$	D	1	1	*x*	$\overline{D}$	0	1	D	J_F={Φ}

NOTES on D-ALGORITHM

1. Exhaustive Search of All Possible Choices
2. Guaranteed to Find a Solution (TV) *IF One Exists*.
3. Stops As Soon As A Solution Is Found
4. Being Exhaustive, the Worst Case Complexity is Exponential in the # of Gates
5. The Best Case Behavior Occurs When a Solution IS Generated *WITHOUT ANY BACKTRACKING*:
   - Always Correct Decisions Are Made
   - Solution Is Obtained Only Through Forward & Backward Implication.

NOTES on D-ALGORITHM

6. THUS, the # of Backtracks Determines the Complexity of the Algorithm (Should be Minimized).

7. An Upper Limit is Placed on the # of Backtracks Beyond Which a Fault Is Declared *UNTESTABLE*!

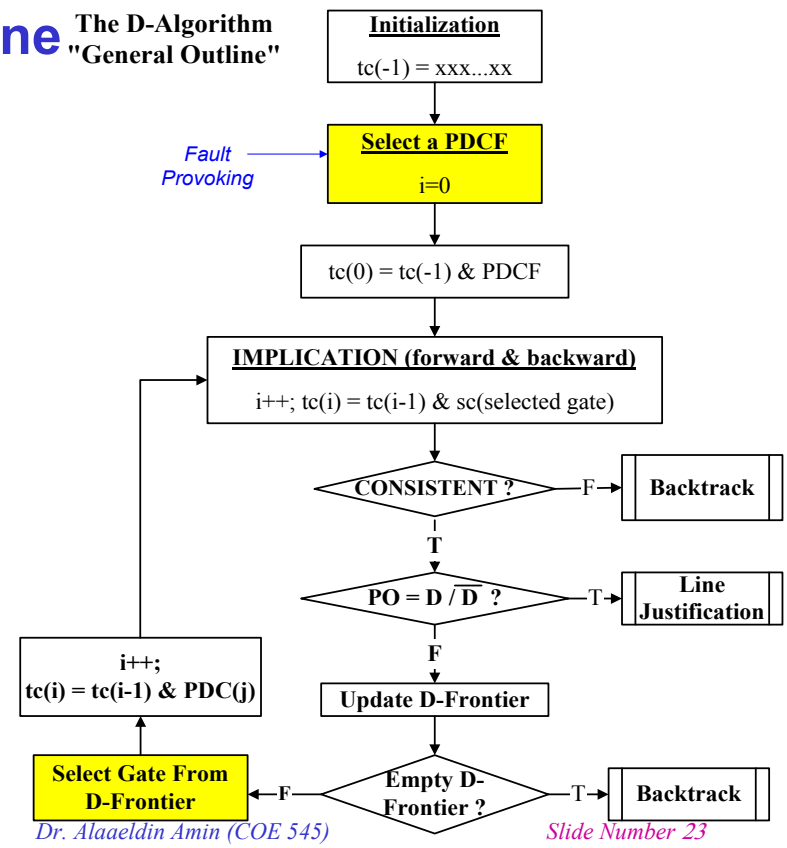
8. To Minimize the # of Backtracks, It is Advisable to Discover Inconsistencies as Early as Possible Through *Forward & Backward Implications for Each Change in the Test Cube*

OutLine of D-Algorithm

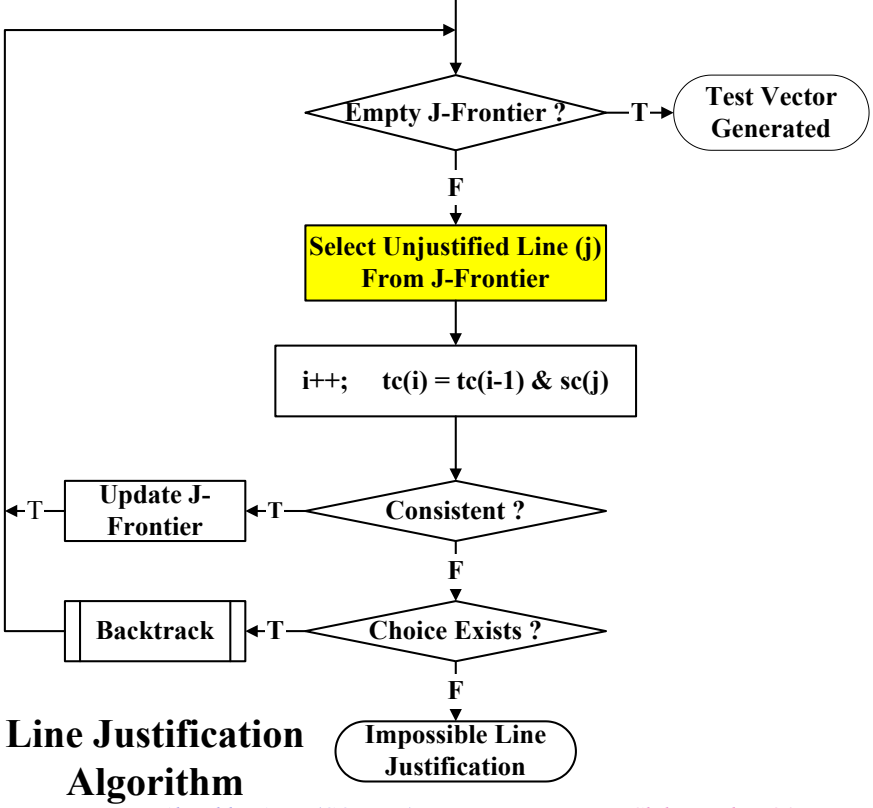
1. Model Fault with appropriate *Primitive D-cube of Fault* (PDCF)
2. Select *Propagation D-cubes* to Propagate fault effect to a circuit output (*D-drive* procedure)
3. Select *singular cover* cubes to justify internal circuit signals (*Line Justification / Consistency* procedure)

- Put Signal Assignments in *Test Cube*
- Regrettably, Cubes Are Selected Very *Arbitrarily* by D-ALG

Algorithm OutLine



Algorithm OutLine



Line Justification Algorithm

D-Algorithm – Top Level

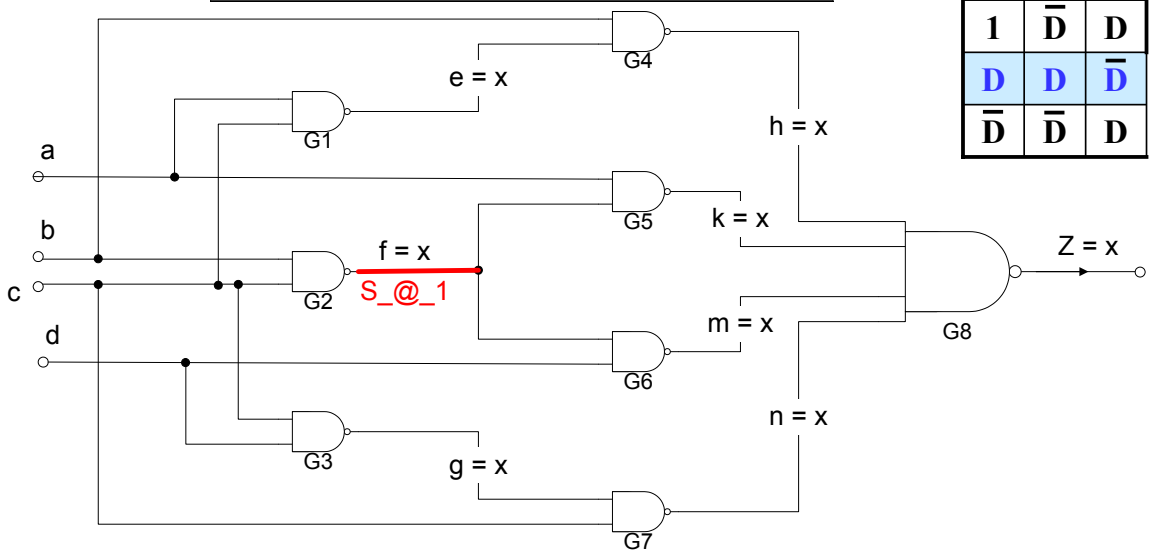
1. Levelize the Circuit (Gates are Defined in Levels)
   - Primary Inputs Are Assigned a Zero Level
   - Level of a Gate = Max(Levels of Gate Input Signals) + 1
2. Number All Circuit Lines in Increasing Level Order From **PIs** to **POs**;
3. Initialize the ***Test Cube*** (t^{-1}) to All *x*'s;
4. Select a Primitive D-cube of the Fault (**PDCF**) and Generate the new ***Test Cube*** (t^{0}); -- *Choice*
   - Construct the initial *D-Frontier*;
5. ***D-drive*** **();** -- *Generate a Chain of PDC from Fault to a PO*
6. ***Consistency*** **();** -- *Justify Unjustified Gate Outputs*
7. **Return ();**

PDC of 2 I/P NAND

A	B	F
D	1	$\bar{D}$
$\bar{D}$	1	D
1	D	$\bar{D}$
1	$\bar{D}$	D
D	D	$\bar{D}$
$\bar{D}$	$\bar{D}$	D

Example Initialize Test Cube tc(-1)

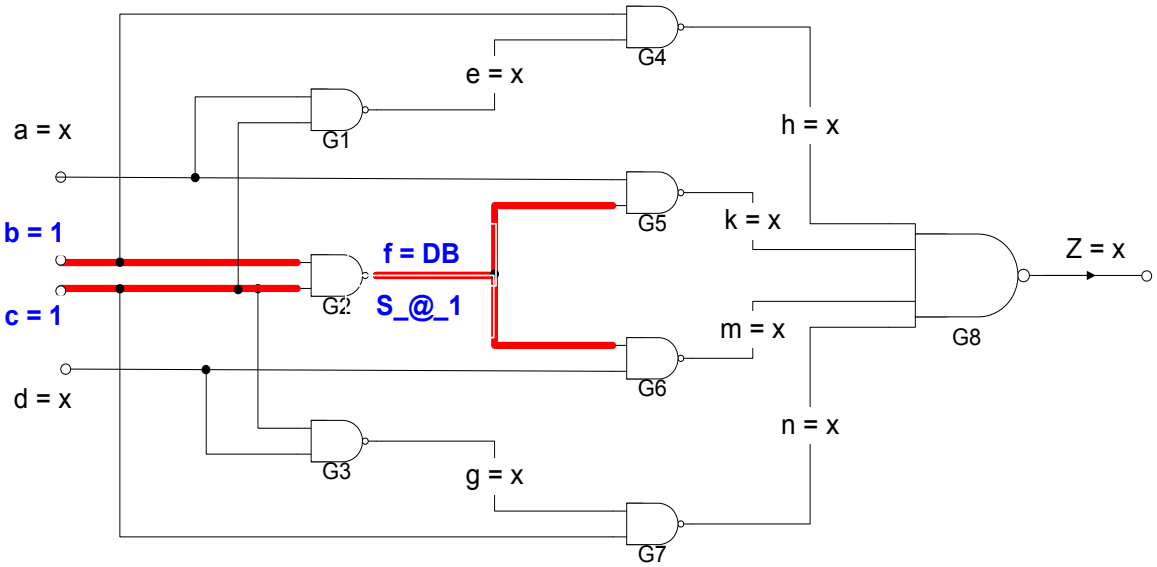
a	b	c	d	e	f	g	h	k	m	n	z
x	x	x	x	x	x	x	x	x	x	x	x



Example

Excite (Provoke) the Fault
(Select a PDCF)

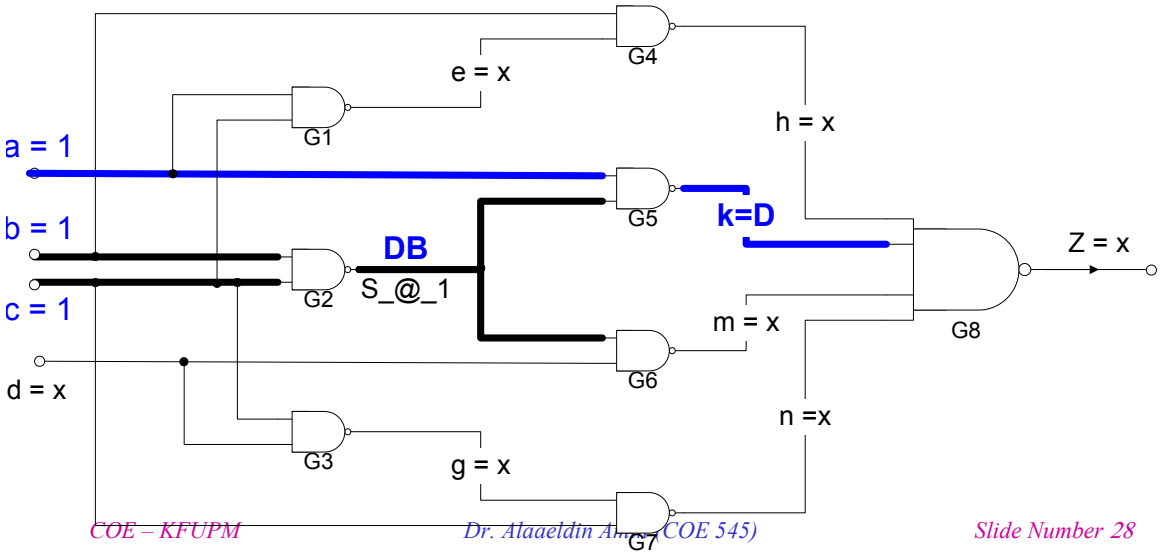
	a	b	c	d	e	f	g	h	k	m	n	z
tc(-1)	x	x	x	x	x	x	x	x	x	x	x	x
PDCF(G2)		1	1			DB						
tc(0) = tc(-1) & PDCF	x	1	1	x	x	DB	x	x	x	x	x	x
D - Frontier	{G5 , G6 }											
J - Frontier	{ }											



Example
D-Drive

Propagate Fault (D-Drive)
(Select D-Frontier Gate ε{G5,G6}
"CHOICE"

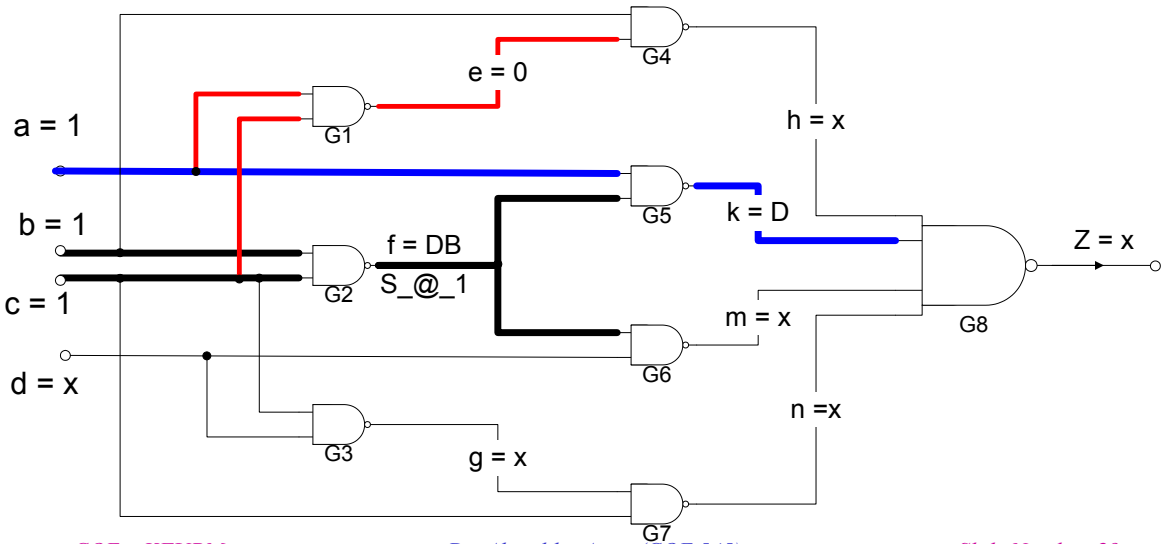
	a	b	c	d	e	f	g	h	k	m	n	z
tc(0)	x	1	1	x	x	DB	x	x	x	x	x	x
PDC(G5)	1					DB			D			
tc(1)= tc(0) & PDC(G5)	1	1	1	x	x	DB	x	x	D	x	x	x
D - Frontier	{G8 , G6 }											
J - Frontier	{ }											



Example

IMPLICATION (Forward)
(Through G1))

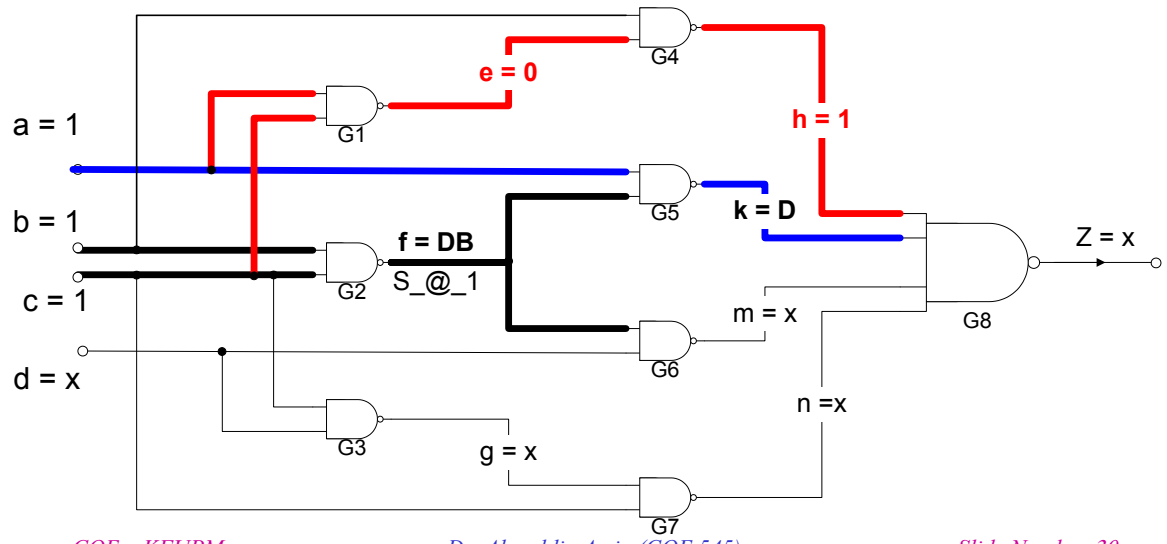
	a	b	c	d	e	f	g	h	k	m	n	z
tc(1)	1	1	1	x	x	DB	x	x	D	x	x	x
SC(G1)	1		1		0							
tc(2) = tc(1) & SC(G1)	1	1	1	x	0	DB	x	x	D	x	x	x
D - Frontier	{G8 , G6 }											
J - Frontier	{ }											



Example

IMPLICATION (Forward) (Through G4))

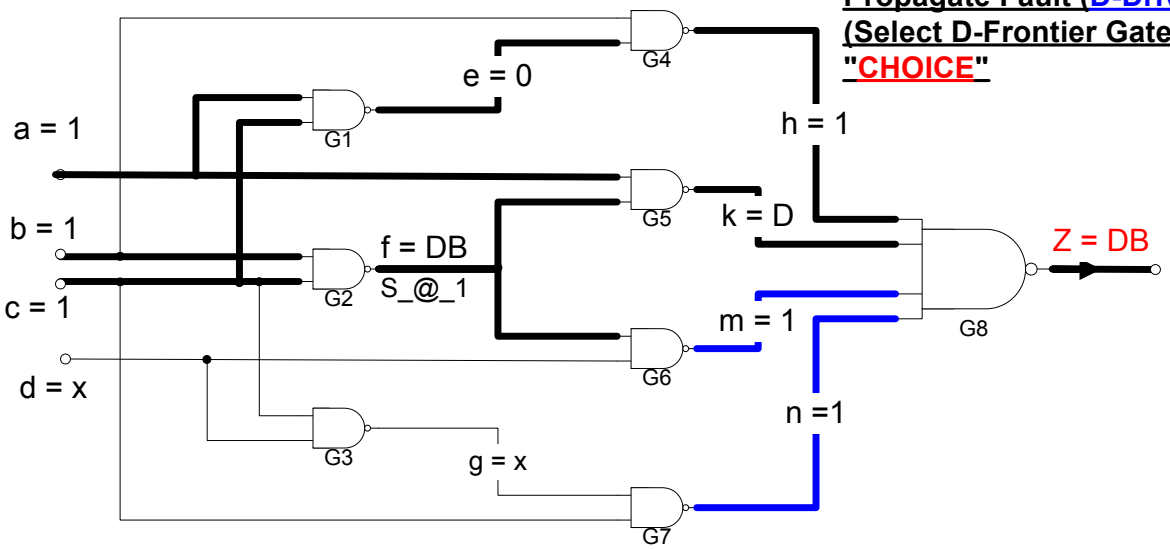
	a	b	c	d	e	f	g	h	k	m	n	z
tc(2)	1	1	1	x	0	DB	x	x	D	x	x	x
SC(G4)		1			0			1				
tc(3) = tc(2) & SC(G4)	1	1	1	x	0	DB	x	1	D	x	x	x
D - Frontier	{G8 , G6 }											
J - Frontier	{ }											



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(3)	1	1	1	x	0	DB	x	1	D	x	x	x
PDC(G8)								1	D	1	1	DB
tc(4) = tc(3) & PDC(8)	1	1	1	x	0	DB	x	1	D	1	1	DB
D - Frontier	{ }											
J - Frontier	{ G6 , G7 }											

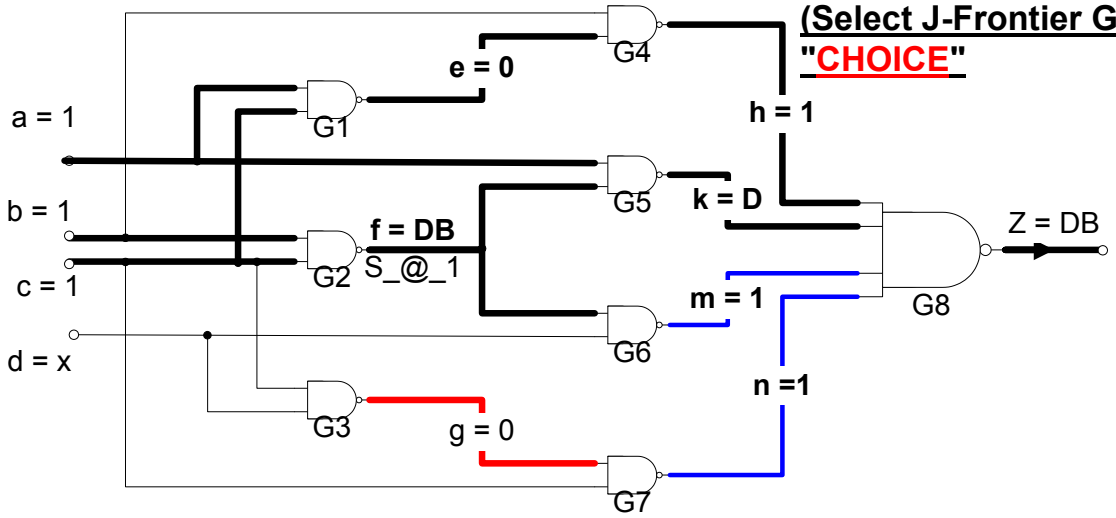
Propagate Fault (D-Drive) (Select D-Frontier Gate "CHOICE"



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(4)	1	1	1	x	0	DB	x	1	D	1	1	DB
SC(G7)			1				0				1	
tc(5) = tc(4) & SC(G7)	1	1	1	x	0	DB	0	1	D	1	1	DB
D - Frontier	{ }											
J - Frontier	**{G6 , G3 }**											

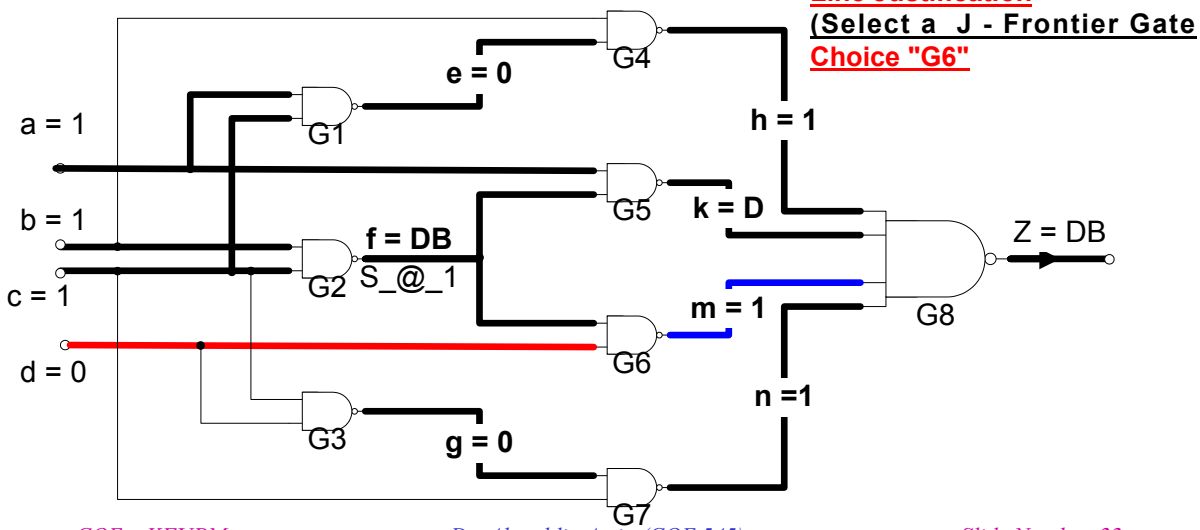
Line Justification
(Select J-Frontier Gate)
"CHOICE"



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(5)	1	1	1	x	0	DB	0	1	D	1	1	DB
SC(G6)				0		x				1		
tc(6) = tc(5) & SC(G6)	1	1	1	0	0	DB	0	1	D	1	1	DB
D - Frontier	{ }											
J - Frontier	{G3 }											

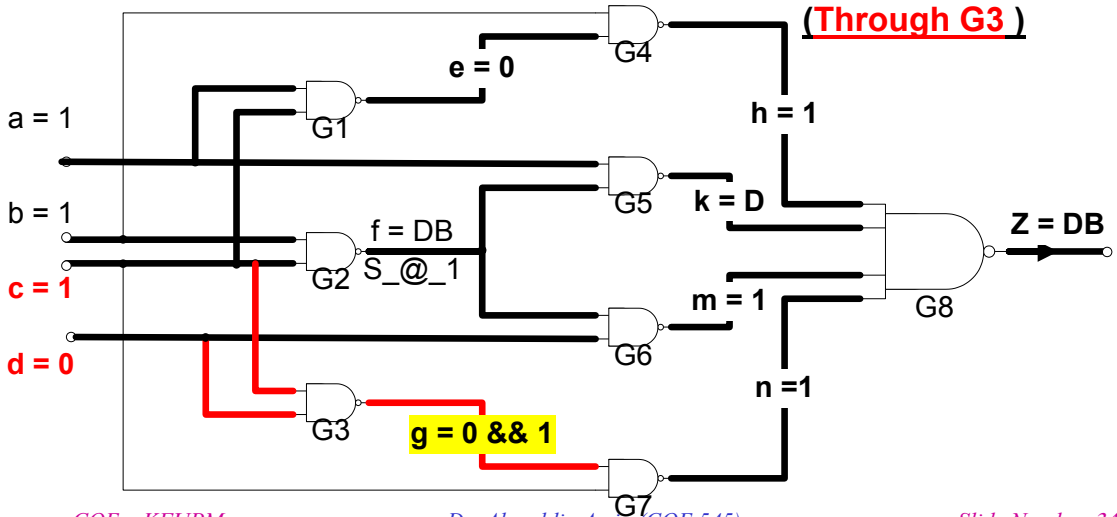
Line Justification
(Select a J - Frontier Gate
Choice "G6"



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(6)	1	1	1	0	0	DB	0	1	D	1	1	DB
SC(G3)			1	0			1					
tc(7) = tc(6) & SC(G3)	INCONSISTENCY											
D - Frontier	{ }											
J - Frontier	{G3 }											

IMPLICATION (Forward)
(Through G3)



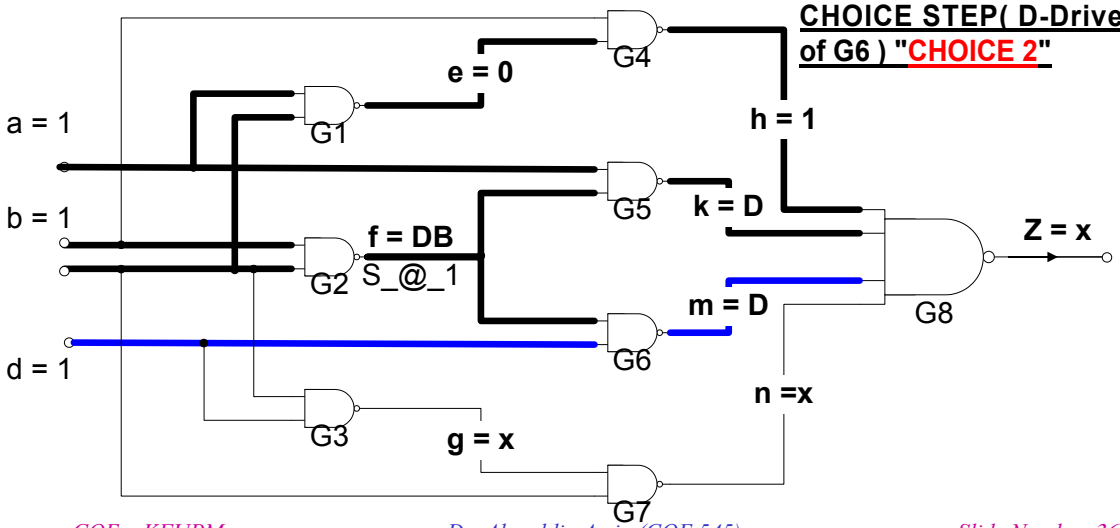
Example

1. **Inconsistency** → Backtrack to the Last Choice Step
   - Line Justification Choice of G6 or G3 → Choice was G6
   - Backtrack and Choose G3 instead for Line Justification
2. Another **Inconsistency** Develops→ Backtrack to the Last Choice Step
   - Line Justification Choice of G6 or G7 → Choice was G7
   - Backtrack and Choose G6 instead for Line Justification
3. Another **Inconsistency** Develops→ Backtrack to the Last Choice Step
   - D-Drive Choice of G6 or G8 → Choice was G8
   - Backtrack to D-Drive Step and Choose G6 instead

Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(3)	1	1	1	x	0	DB	x	1	D	x	x	x
PDC(G6)				1		DB				D		
tc(4)= tc(3) & PDC(G6)	1	1	1	1	0	DB	x	1	D	D	x	x
D - Frontier	{G8 }											
J - Frontier	{ }											

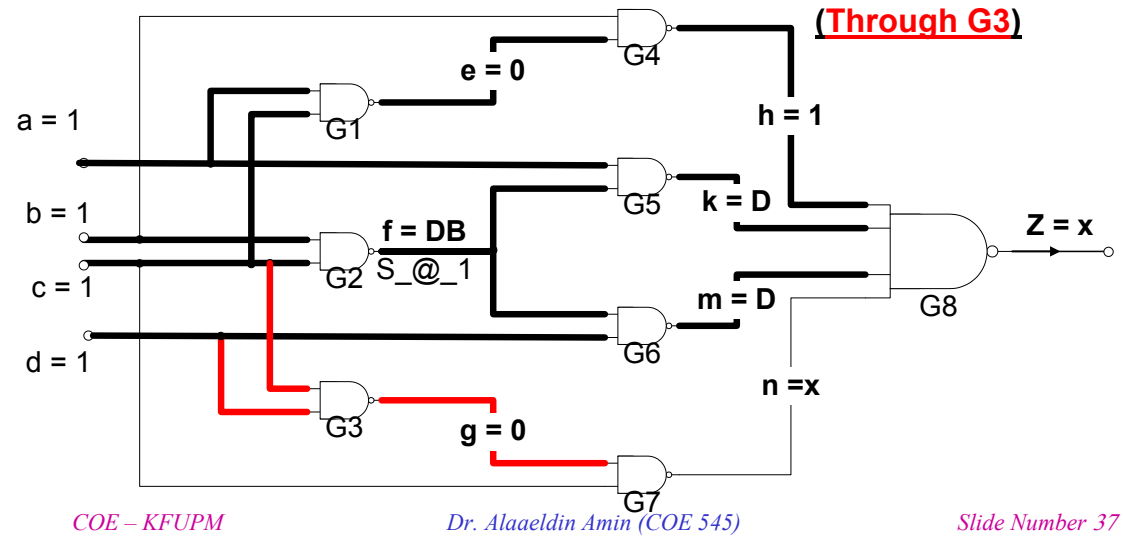
BACKTRACK to LAST CHOICE STEP(D-Drive PDC of G6) "CHOICE 2"



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(4)	1	1	1	1	0	DB	x	1	D	D	x	x
SC(G3)			1	1			0					
tc(5) = tc(4) & SC(G3)	1	1	1	1	0	DB	0	1	D	D	x	x
D - Frontier	{G8 }											
J - Frontier	{ }											

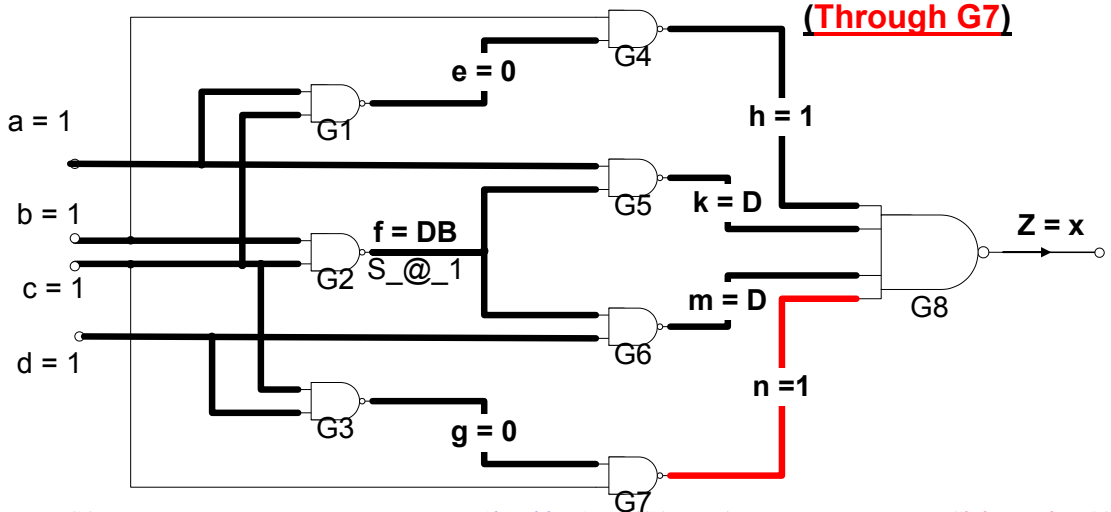
IMPLICATION (Forward) (Through G3)



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(5)	1	1	1	1	0	DB	0	1	D	D	x	x
SC(G7)			1				0				1	
tc(6) = tc(5) & SC(G7)	1	1	1	1	0	DB	0	1	D	D	1	x
D - Frontier	{G8 }											
J - Frontier	{ }											

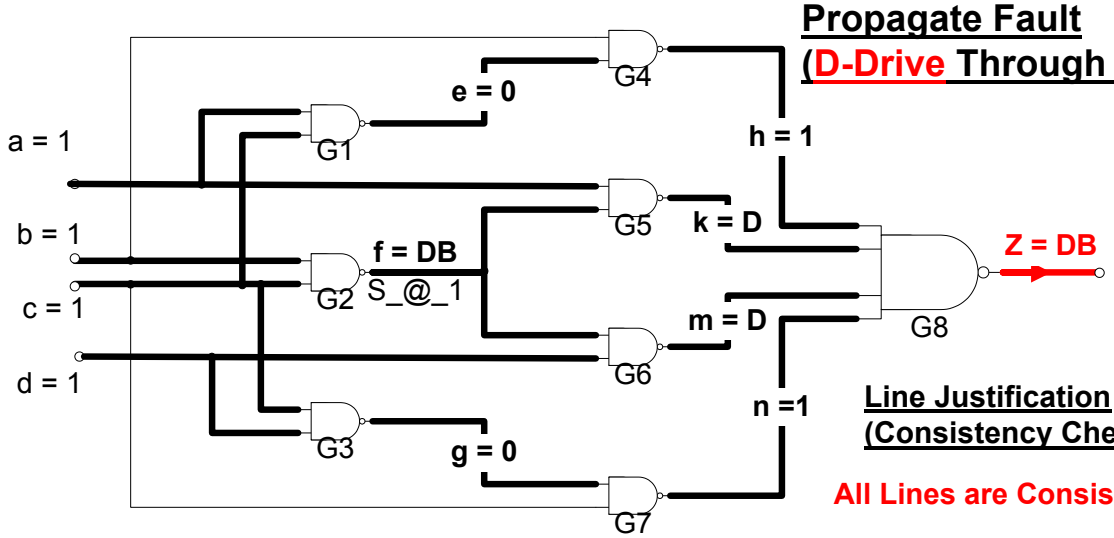
IMPLICATION (Forward)
(Through G7)



Example

	a	b	c	d	e	f	g	h	k	m	n	z
tc(6)	1	1	1	1	0	DB	x	1	D	D	x	x
PDC(G8)								1	D	D	1	DB
tc(7)= tc(6) & PDC(G8)	1	1	1	1	0	DB	0	1	D	D	1	DB
D - Frontier	{ }											
J - Frontier	{ }											

Propagate Fault
(D-Drive Through G8)

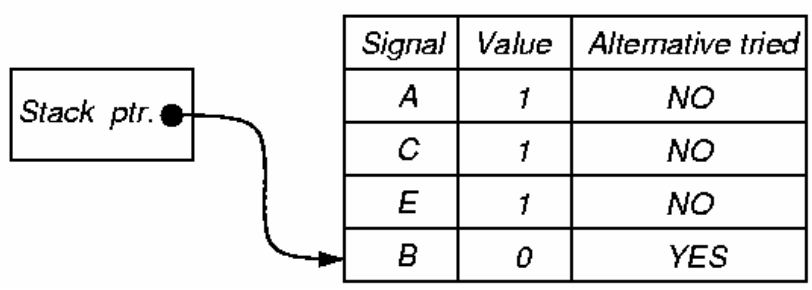


Line Justification
(Consistency Check)

All Lines are Consistent

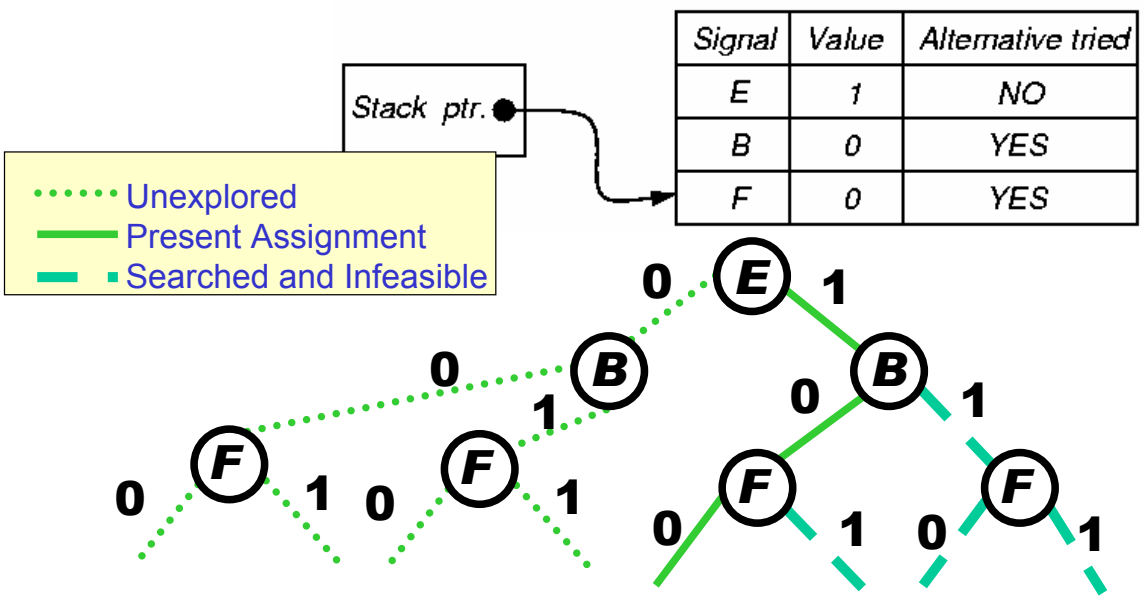
Implication Stack

- **Push-down stack. Records:**
  - **Each signal set in circuit by ATPG**
  - **Whether alternate signal value already tried**
  - **Portion of binary search tree already searched**



Signal	Value	Alternative tried
A	1	NO
C	1	NO
E	1	NO
B	0	YES

Implication Stack after Backtrack



Signal	Value	Alternative tried
E	1	NO
B	0	YES
F	0	YES

Alternate Solution Approach

Fault is Line l/v

Fault Provoking

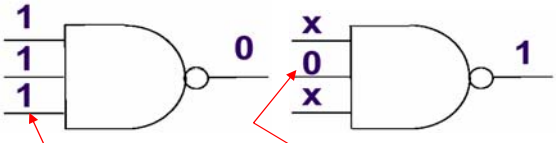
Fault Propagation

```
Fanout_Free()
begin
  Set all Values to x
  Justify (l, vB)
  if (v = 0)
    then Propagate (l, D)
    else  Propagate (l, DB)
End
```

Test Generation for l S_@_v Fault in a Fanout-Free CUT

Alternate Solution Approach

- Justify is a Recursive Procedure



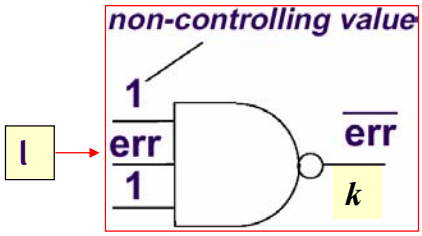
Non-Controlling

Controlling

C	*i*	Gate
0	0	AND
0	1	NAND
1	0	OR
1	1	NOR

```
Justify(l, val) - - For Fanout-Free CUTs
Begin
l = val
If (l is PI) then return
/* (l is a Gate Output */
C = Control Value of l
i = inversion of l
inval = val ⊕ i
if (inval = not(c))
    then for Every I/P j of l
          Justify (j, inval)
    else
      begin
          Select One I/P j of l
          Justify (j, inval)
      end
end
```

Alternate Solution Approach



Fault Propagation Problem Converted into a Line-Justification Problem

Propagate **(**l, err**)**
- - For Fanout-Free CUTs
/ err is D or DB */*
Begin
l = err
If (l is PO) **then return**
k = Fanout Gate of l
C = Control Value of *k*
i = inversion of *k*
For Every I/P *j of k Other Than* l
  Justify **(**j, not(C)**)**
  Propagate **(***k* , err ⊕ *i* **)**
end

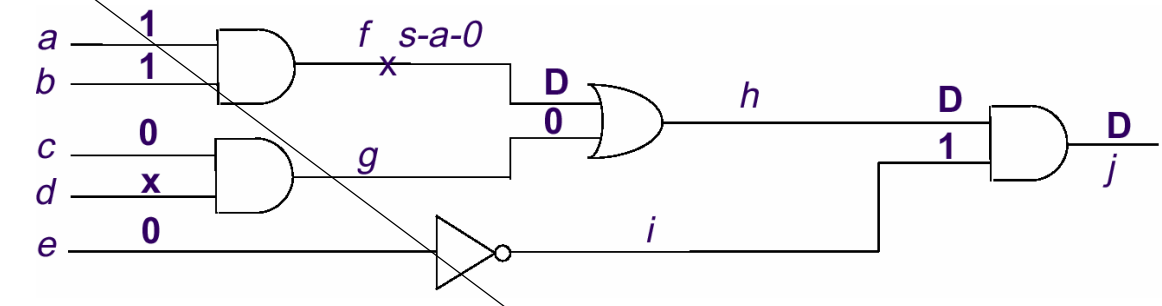
Reduces Fault Propagation to a set of line- justification problems


```
D-alg()
begin
If ( Imply_and_check() = FAILURE ) then return FAILURE
if (error not at PO) then begin
    if (D-frontier = void ) then return FAILURE
    repeat
        begin
            select an untried gate(G) from D-frontier
            c = controlling value of G
            assign c to every input of G with value x
            if (D-alg() = SUCCESS) then return SUCCESS
        end
    until all gates from D-frontier have been tried
    return FAILURE
end
If (J-frontier = void) then return SUCCESS  -- Fault Appeared at PO
select a gate (G) from J-frontier
c = controlling value of G
repeat
    begin
        select an input (j) of G with value x, assign c to j
        if (Solve() = SUCCESS) then return SUCCESS
        assign C̄ to j                          -- Reverse Decision
    end
until all inputs of G are specified
return FAILURE
end
```

Alternate Solution Approach

```
Solve()
begin
If (Imply_and_check() = FAILURE ) then return FAILURE
if (error at PO and all lines are justified)
    then return SUCCESS
if (error can't be propagated to a PO)
    then return FAILURE
Select an unsolved problem
repeat
    begin
        select one untried way to solve it
        if (Solve() = SUCCESS) then return SUCCESS
    end
until all ways to solve it have been tried
return FAILURE
end
```


Example (Fanout-Free CUT)



- In Fanout-Free CUTs, Line-Justification Problems Can be Solved *Independently* of all Others (All PIs are usually Exclusive)
- Fanout CUTS
  - More than One Possibility of Propagating Faults to POs
  - Once a Path is Chosen to Propagate a Fault, this Can Also Be Reduced to a *Set of Line Justification Problems*
- For Reconvergent Fanout CUTS, Line Justification Problems are No Longer Independent (Inconsistencies may arise)

```
D-alg()
begin
If ( Imply_and_check( ) = FAILURE )
    then  return FAILURE
if (error not at PO) then
 begin
    if (Empty D-frontier)
          then return FAILURE
    repeat
     begin
       select untried gate(G) from D-frontier
       c = controlling value of G
       assign c to every I/P of G with value x
       if (D-alg() = SUCCESS)
          then return SUCCESS
     end
    until all D-frontier gates from are  tried
    return FAILURE
end
```

```
If (J-frontier = void) then return
    SUCCESS
select a gate (G) from J-frontier
c = controlling value of G
repeat
    begin
        select an input (j) of G with
    value x, assign c to j
        if (Solve() = SUCCESS)
    then return SUCCESS
    end
until all inputs of G are specified
return FAILURE
end
```