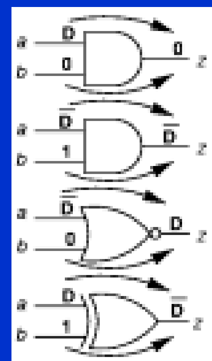
Major Combinational Automatic Test-Pattern Generation Algorithms

- Definitions
- D-Algorithm (Roth) -- 1966
  - *D-cubes*
  - Bridging faults
  - Logic gate function change faults
- PODEM (Goel) -- 1981
  - *X-Path-Check*
  - *Backtracing*
- Summary

Forward Implication

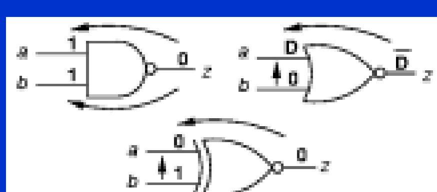


- Results in logic gate inputs that are significantly labeled so that output is uniquely determined
- AND gate forward implication table:

a \ b	0	1	X	D	$\overline{D}$
0	0	0	0	0	0
1	0	1	X	D	$\overline{D}$
X	0	X	X	X	X
D	0	D	X	D	0
$\overline{D}$	0	$\overline{D}$	X	0	$\overline{D}$

Backward Implication

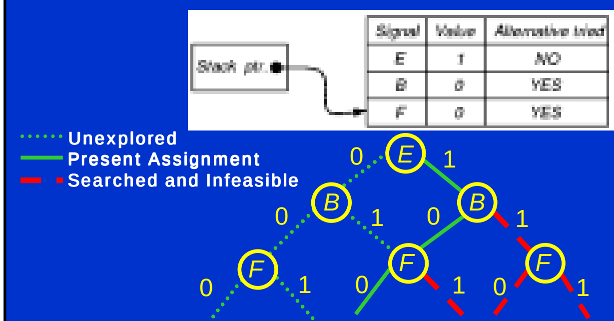
- Unique determination of all gate inputs when the gate output and some of the inputs are given



Implication Stack

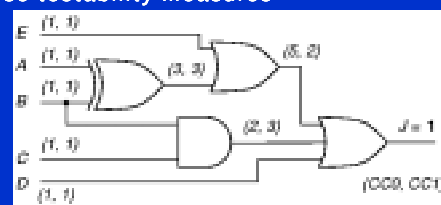
- Push-down stack. Records:
  - Each signal set in circuit by ATPG
  - Whether alternate signal value already tried
  - Portion of binary search tree already searched

Implication Stack after Backtrack



Objectives and Backtracing of ATPG Algorithm

- *Objective* – desired signal value goal for ATPG
  - Guides it away from infeasible/hard solutions
- *Backtrace* – Determines which primary input and value to set to achieve objective
  - Use testability measures



Branch-and-Bound Search

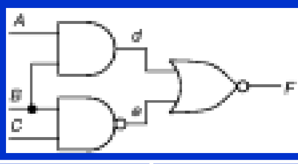
- Efficiently searches binary search tree
- *Branching* – At each tree level, selects which input variable to set to what value
- *Bounding* – Avoids exploring large tree portions by artificially restricting search decision choices
  - Complete exploration is impractical
  - Uses *heuristics*

D-Algorithm -- Roth IBM (1966)

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - *D-Cube*
  - *D-Calculus*
  - *Implications* – forward and backward
  - *Implication stack*
  - *Backtrack*
  - Test Search Space

Singular Cover Example

- Minimal set of logic signal assignments to show *essential prime implicants* of *Karnaugh map*



Gate	Inputs		Output	Gate	Inputs		Output
AND	*A*	*B*	*d*	NOR	*d*	*e*	*F*
1	0	X	0	1	1	X	0
2	X	0	0	2	X	1	0
3	1	1	1	3	0	0	1

D-Cube

- Collapsed truth table entry to characterize logic
- Use Roth's 5-valued algebra
- Can change all D's to $\overline{D}$'s and $\overline{D}$'s to D's (do both)
- AND gate:

	A	*B*	*d*
Rows 1 & 3	D	1	D
Reverse inputs	1	D	D
And two cubes	D	D	D
Interchange D and $\overline{D}$	$\overline{D}$	$\overline{D}$	$\overline{D}$
	1	$\overline{D}$	$\overline{D}$
	$\overline{D}$	1	$\overline{D}$

D-Cube Operation of D-Intersection

- ψ – undefined (same as ϕ)
- μ or λ – requires inversion of D and $\overline{D}$
- *D-intersection*: $0 \cap 0 = 0 \cap X = X \cap 0 = 0$
  $1 \cap 1 = 1 \cap X = X \cap 1 = 1$
  $X \cap X = X$
- *D-containment* – Cube *a* contains Cube *b* if *b* is a subset of *a*

∩	0	1	X	D	$\overline{D}$
0	0	ϕ	0	ψ	ψ
1	ϕ	1	1	ψ	ψ
X	0	1	X	D	$\overline{D}$
D	ψ	ψ	D	μ	λ
$\overline{D}$	ψ	ψ	$\overline{D}$	λ	μ

Primitive D-Cube of Failure

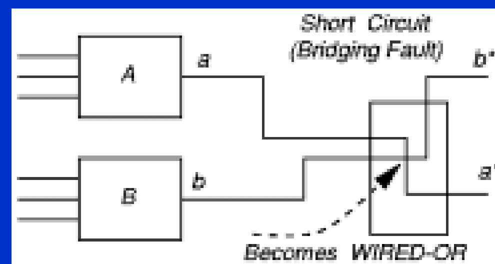
- Models circuit faults:
  - *Stuck-at-0*
  - *Stuck-at-1*
  - *Bridging fault* (short circuit)
  - Arbitrary change in logic function
- AND Output sa0: "1 1 D"
- AND Output sa1: "0 X $\overline{D}$ "
  "X 0 $\overline{D}$ "
- Wire sa0: "D"
- *Propagation D-cube* – models conditions under which fault effect propagates through gate

Implication Procedure

1. Model fault with appropriate ***primitive D-cube of failure*** (PDF)
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 (*Consistency* procedure)

- Put signal assignments in ***test cube***
- Regrettably, cubes are selected very arbitrarily by D-ALG

Bridging Fault Circuit



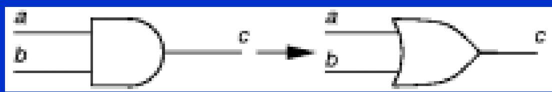
Construction of Primitive D-Cubes of Failure

1. Make cube set $\alpha 1$ when good machine output is 1 and set $\alpha 0$ when good machine output is 0
2. Make cube set $\beta 1$ when failing machine output is 1 and $\beta 0$ when it is 0
3. Change $\alpha 1$ outputs to 0 and D-intersect each cube with every $\beta 0$. If intersection works, change output of cube to D
4. Change $\alpha 0$ outputs to 1 and D-intersect each cube with every $\beta 1$. If intersection works, change output of cube to $\overline{D}$

Bridging Fault D-Cubes of Failure

Cube-set	*a*	*b*	*a**	*b**	Cube-set	*a*	*b*	*a**	*b**
$\alpha 0$	0	X	0	X	PDFs for Bridging fault	1	0	1	$\overline{D}$
	X	0	X	0		0	1	$\overline{D}$	1
$\alpha 1$	1	X	1	X					
	X	1	X	1					
$\beta 0$	0	0	0	0					
$\beta 1$	X	1	1	1					
	1	X	1	1					

Gate Function Change D-Cube of Failure



Cube-set	a	b	c	Cube-set	a	b	c
α0	0	X	0	PDFs for AND changing to OR	0	1	$\overline{D}$
	X	0	0				
α1	1	1	1		1	0	$\overline{D}$
β0	0	0	0				
β1	1	X	1				
	X	1	1				

D-Algorithm – Top Level

1. Number all circuit lines in increasing level order from PIs to POs;
2. Select a primitive D-cube of the fault to be the *test cube*;
   - Put logic outputs with inputs labeled as D ($\overline{D}$) onto the *D-frontier*;
3. *D-drive* ();
4. *Consistency* ();
5. return ();

D-Algorithm – *D-drive*

while (untried fault effects on D-frontier)
    select next untried D-frontier gate for propagation;
    while (untried fault effect fanouts exist)
        select next untried fault effect fanout;
        generate next untried propagation D-cube;
        D-intersect selected cube with test cube;
        if (intersection fails or is undefined) continue;
        if (all propagation D-cubes tried & failed) break;
        if (intersection succeeded)
            add propagation D-cube to test cube -- recreate *D-frontier*;
            Find all forward & backward implications of assignment;
            save *D-frontier*, algorithm state, test cube, fanouts, fault;
            break;
        else if (intersection fails & D and $\overline{D}$ in test cube) *Backtrack* ();
        else if (intersection fails) break;
    if (all fault effects unpropagatable) *Backtrack* ();

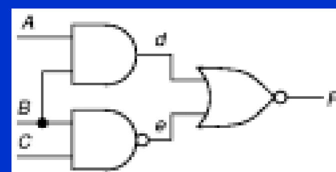
D-Algorithm -- *Consistency*

g = coordinates of test cube with 1's & 0's;
if (*g* is only PIs) fault testable & stop;
for (each unjustified signal in *g*)
    Select highest # unjustified signal *z* in *g*, not a PI;
    if (inputs to gate *z* are both D and $\overline{D}$) break;
    while (untried singular covers of gate *z*)
        select next untried singular cover;
        if (no more singular covers)
            If (no more stack choices) fault untestable & stop;
            else if (untried alternatives in *Consistency*)
                pop implication stack -- try alternate assignment;
            else
                Backtrack ();
                D-drive ();
        If (singular cover D-intersects with *z*) delete *z* from *g*, add inputs to singular cover to *g*, *find all forward and backward implications of new assignment*, and break;
        If (intersection fails) mark singular cover as failed;

Backtrack

if (PO exists with fault effect) ***Consistency*** ();
else pop prior implication stack setting to try alternate assignment;
if (no untried choices in implication stack)
  fault untestable & stop;
else return;

Circuit Example 7.1 and Truth Table



Inputs			Output
a	*b*	*c*	*F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Singular Cover & D-Cubes

A	*B*	*C*	*d*	*e*	*F*
1	1		1		
0			0		
	0		0		
	1	1		0	
	0			1	
		0		1	
				1	0
			1		0
			0	0	1
D	1		D		
1	D		D		
D	D		D		
	D	1		$\overline{D}$	
	1	D		$\overline{D}$	
	D	D		$\overline{D}$	
			D	0	$\overline{D}$
			0	D	$\overline{D}$
			D	D	$\overline{D}$

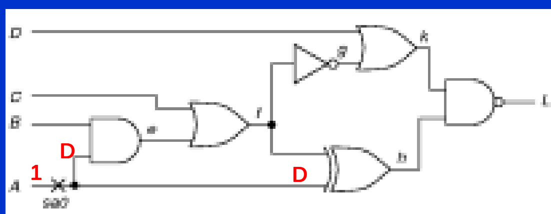
- ***Singular cover*** – Used for justifying lines
- ***Propagation D-cubes*** – Conditions under which difference between good/failing machines propagates

Steps for Fault *d* sa0

Step	*A*	*B*	*C*	*d*	*e*	*F*	Cube type
1	1	1		D			PDF of AND gate
2				D	0	$\overline{D}$	Prop. D-cube for NOR
3		1	1		0		Sing. Cover of NAND

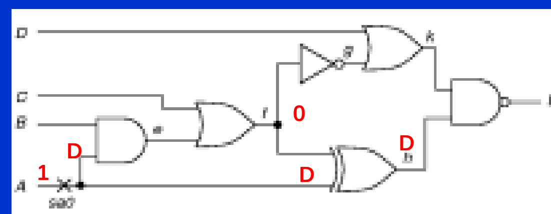
Example 7.2 Fault *A* sa0

- Step 1 – *D-Drive* – Set *A = 1*



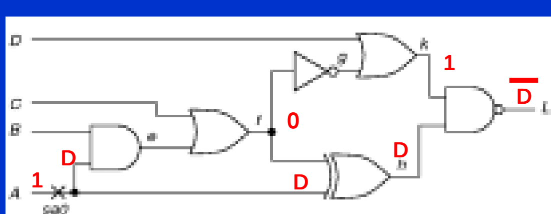
Step 2 -- Example 7.2

- Step 2 – *D-Drive* – Set *f* = 0



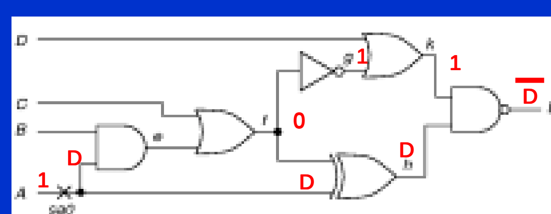
Step 3 -- Example 7.2

- Step 3 – *D-Drive* – Set *k* = 1



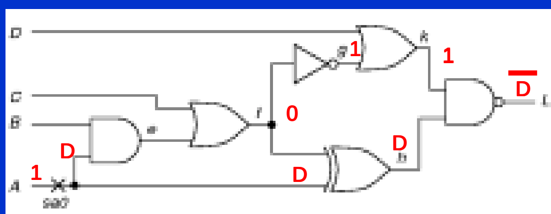
Step 4 -- Example 7.2

- Step 4 – *Consistency* – Set *g* = 1



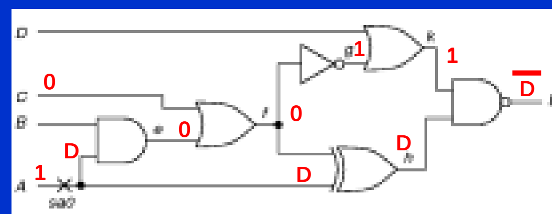
Step 5 -- Example 7.2

- Step 5 – *Consistency* – *f* = 0 Already set



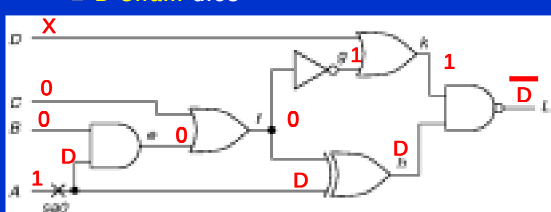
Step 6 -- Example 7.2

- Step 6 – *Consistency* – Set *c* = 0, Set *e* = 0



D-Chain Dies -- Example 7.2

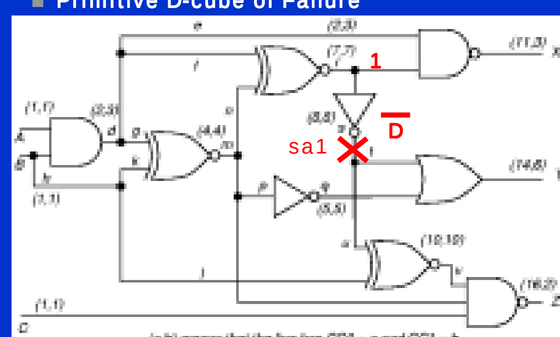
- Step 7 – *Consistency* – Set *B* = 0
- *D-Chain* dies



- *Test cube*: *A, B, C, D, e, f, g, h, k, L*

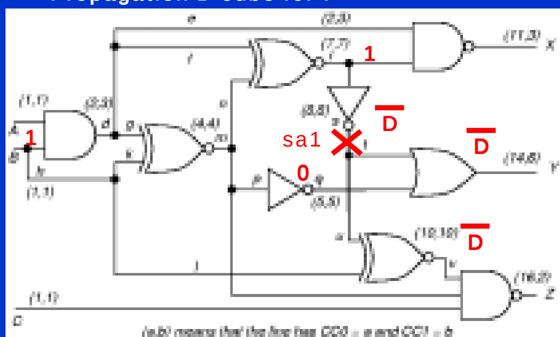
Example 7.3 – Fault *s* sa1

- Primitive D-cube of Failure



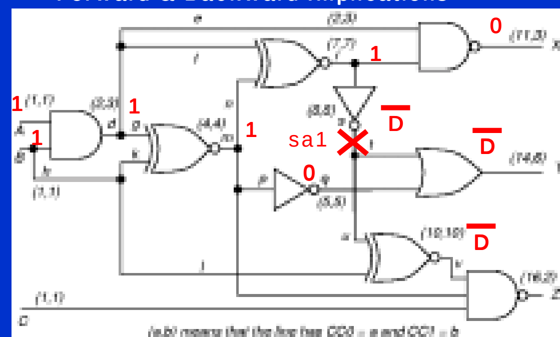
Example 7.3 – Step 2 *s* sa1

- Propagation D-cube for *v*



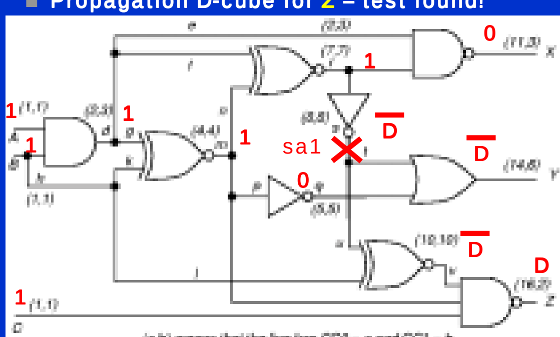
Example 7.3 – Step 2 *s* sa1

- Forward & Backward Implications



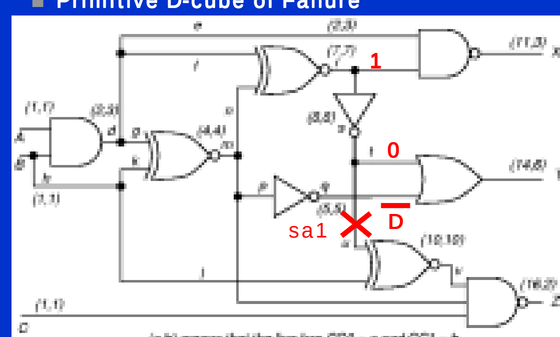
Example 7.3 – Step 3 *s* sa1

- Propagation D-cube for *Z* – test found!



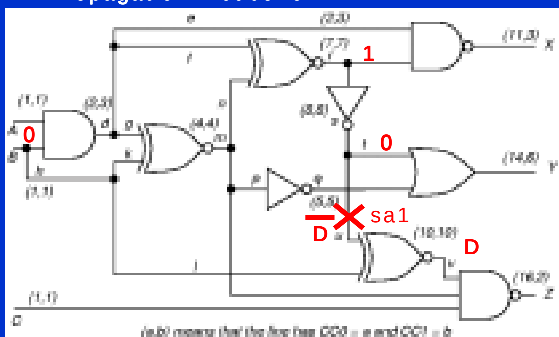
Example 7.3 – Fault *u* sa1

- Primitive D-cube of Failure



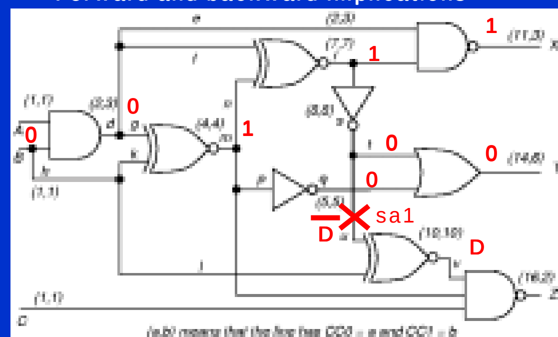
Example 7.3 – Step 2 *u* sa1

- Propagation D-cube for *v*



Example 7.3 – Step 2 *u* sa1

- Forward and backward implications

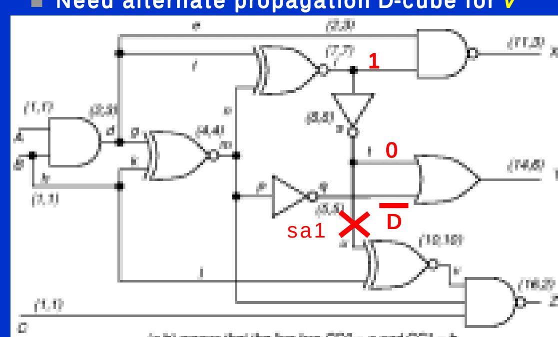


Inconsistent

- *d* = 0 and *m* = 1 cannot justify *r* = 1 (equivalence)
  - Backtrack
  - Remove *B* = 0 assignment

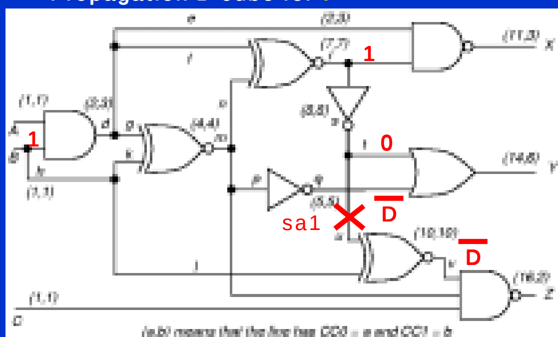
Example 7.3 – Backtrack

- Need alternate propagation D-cube for *v*



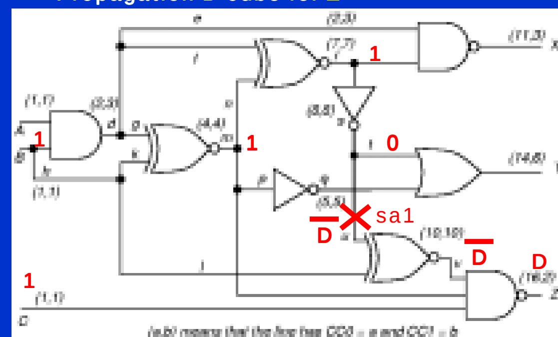
Example 7.3 – Step 3 *u* sa1

- Propagation D-cube for *v*



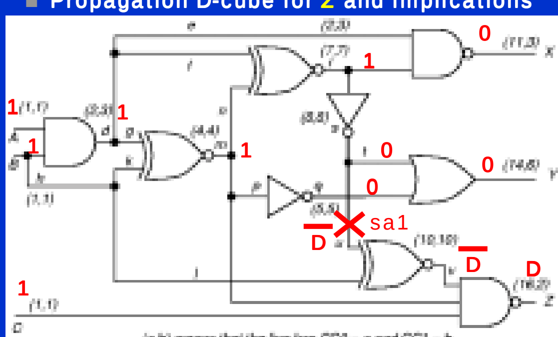
Example 7.3 – Step 4 *u* sa1

- Propagation D-cube for *Z*



Example 7.3 – Step 4 *u* sa1

- Propagation D-cube for *Z* and implications



PODEM -- Goel IBM (1981)

- New concepts introduced:
  - Expand binary decision tree only around primary inputs
  - Use *X-PATH-CHECK* to test whether *D-frontier* still there
  - *Objectives* -- bring ATPG closer to propagating D ($\overline{D}$) to PO
  - *Backtracing*

Motivation

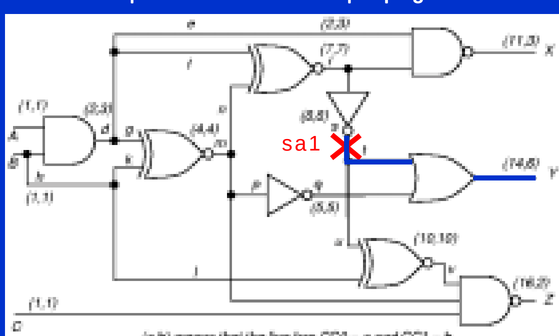
- IBM introduced semiconductor DRAM memory into its mainframes – late 1970's
- Memory had error correction and translation circuits – improved reliability
  - D-ALG unable to test these circuits
    - Search too undirected
    - Large XOR-gate trees
    - Must set all external inputs to define output
  - Needed a better ATPG tool

PODEM High-Level Flow

1. Assign binary value to unassigned PI
2. Determine implications of all PIs
3. Test Generated? If so, done.
4. Test possible with more assigned PIs? If maybe, go to Step 1
5. Is there untried combination of values on assigned PIs? If not, exit: untestable fault
6. Set untried combination of values on assigned PIs using objectives and backtrace. Then, go to Step 2

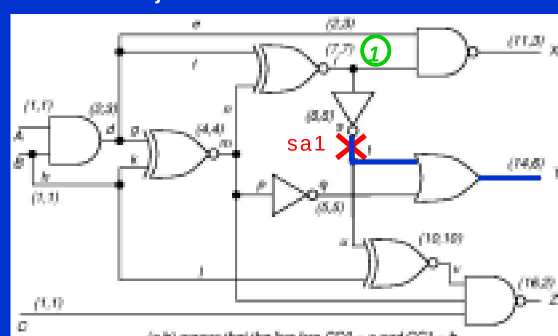
Example 7.3 Again

- Select path *s – Y* for fault propagation



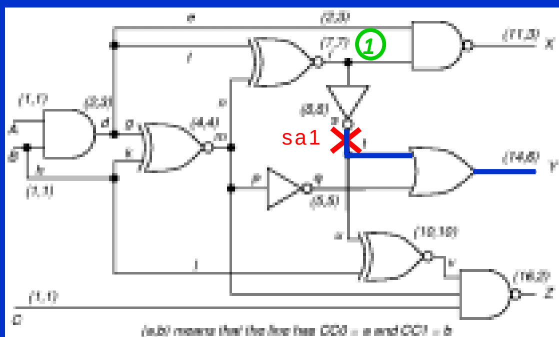
Example 7.3 -- Step 2 *s* sa1

- Initial objective: Set *r* to 1 to sensitize fault



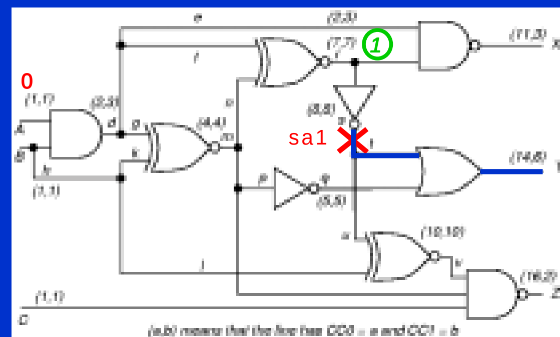
Example 7.3 -- Step 3 *s* sa1

- Backtrace from *r*



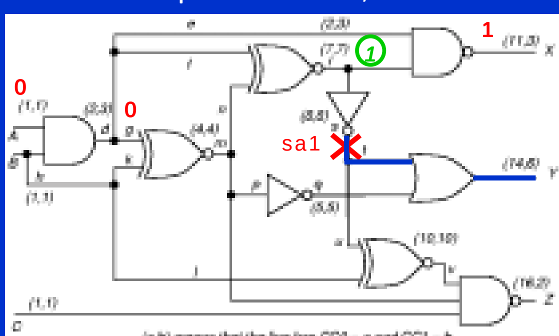
Example 7.3 -- Step 4 *s* sa1

- Set *A* = 0 in implication stack



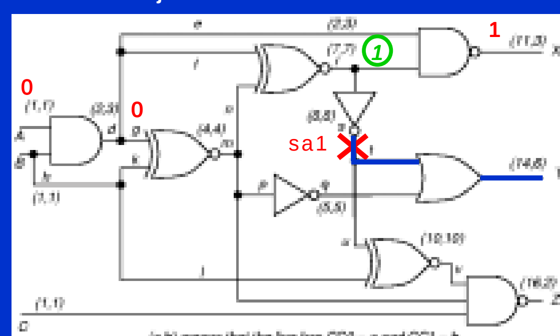
Example 7.3 -- Step 5 *s* sa1

- Forward implications: *d* = 0, *X* = 1



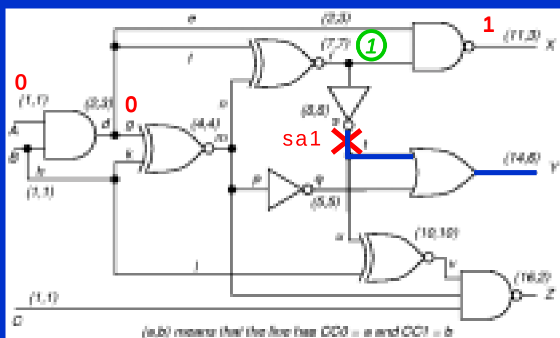
Example 7.3 -- Step 6 *s* sa1

- Initial objective: set *r* to 1



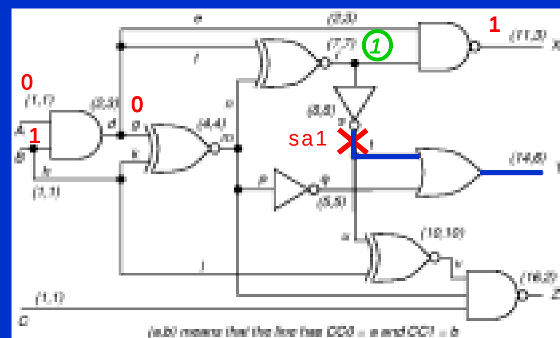
Example 7.3 -- Step 7 *s* sa1

- Backtrace from *r* again



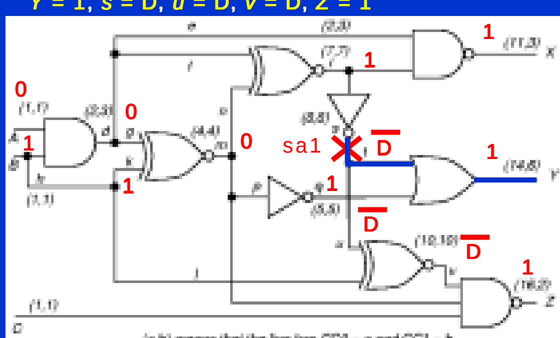
Example 7.3 -- Step 8 *s* sa1

- Set *B* to 1. Implications in stack: *A* = 0, *B* = 1



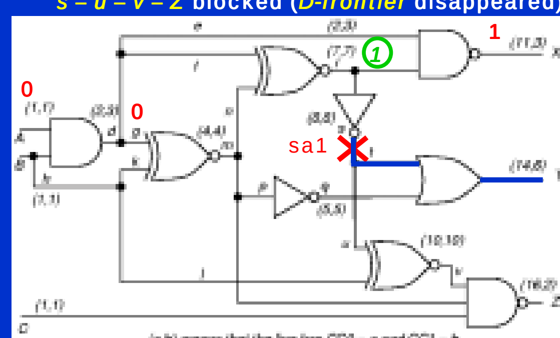
Example 7.3 -- Step 9 *s* sa1

- Forward implications: *k* = 1, *m* = 0, *r* = 1, *q* = 1, *Y* = 1, *s* = $\overline{D}$, *u* = $\overline{D}$, *v* = $\overline{D}$, *Z* = 1



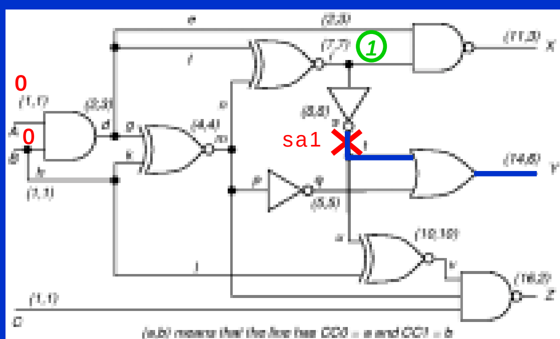
Backtrack -- Step 10 *s* sa1

- *X-PATH-CHECK* shows paths *s* – *Y* and *s* – *u* – *v* – *Z* blocked (*D-frontier* disappeared)



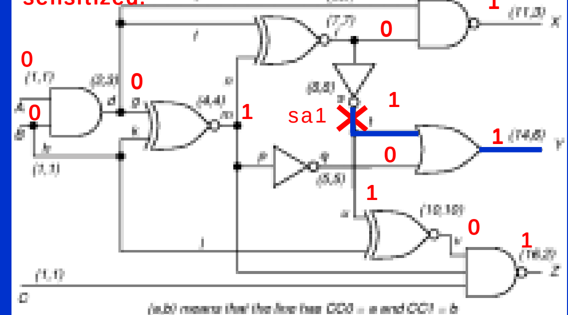
Step 11 -- *s* sa1

- Set *B* = 0 (alternate assignment)



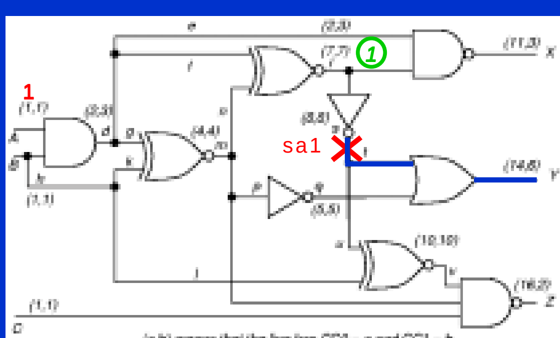
Backtrack -- *s* sa1

- Forward implications: *d* = 0, *X* = 1, *m* = 1, *r* = 0, *s* = 1, *q* = 0, *Y* = 1, *v* = 0, *Z* = 1. Fault not sensitized.



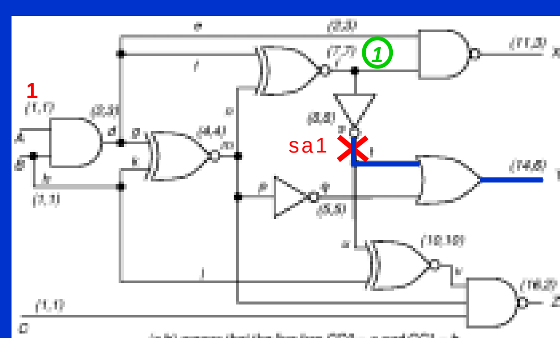
Step 13 -- *s* sa1

- Set *A* = 1 (alternate assignment)



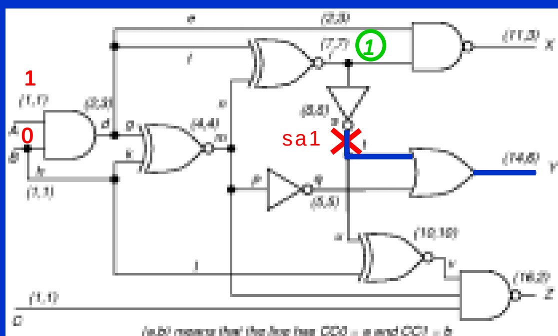
Step 14 -- *s* sa1

- Backtrace from *r* again



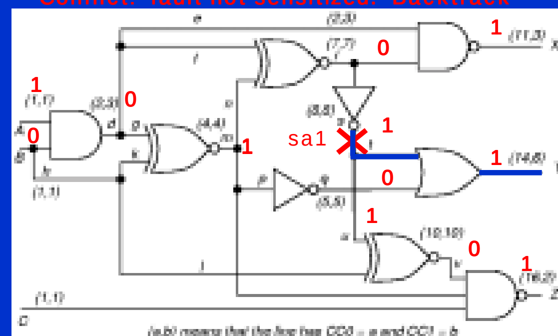
Step 15 -- *s* sa1

- Set *B* = 0. Implications in stack: *A* = 1, *B* = 0



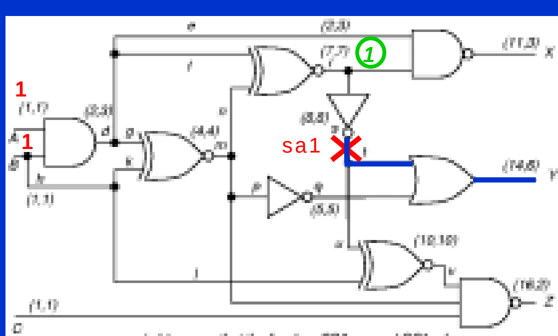
Backtrack -- *s* sa1

- Forward implications: *d* = 0, *X* = 1, *m* = 1, *r* = 0. Conflict: fault not sensitized. Backtrack



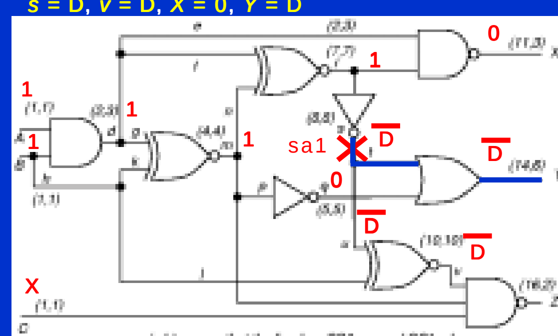
Step 17 -- *s* sa1

- Set *B* = 1 (alternate assignment)



Fault Tested -- Step 18 *s* sa1

- Forward implications: *d* = 1, *m* = 1, *r* = 1, *q* = 0, *s* = D, *v* = D, *X* = 0, *Y* = D



Backtrace (s, v_s) Pseudo-Code

$v = v_s$;
while (*s* is a gate output)
    if (*s* is NAND or INVERTER or NOR) $v = \overline{v}$;
    if (objective requires setting all inputs)
        select unassigned input *a* of *s* with
          hardest controllability to value *v*;
    else
        select unassigned input *a* of *s* with
          easiest controllability to value *v*;
    s = *a*;
return (*s*, *v*) /* Gate and value to be assigned */;

Objective Selection Code

if (gate *g* is unassigned) return (*g*, $\overline{v}$);
select a gate *P* from the D-frontier;
select an unassigned input *l* of *P*;
if (gate *g* has controlling value)
    c = controlling input value of *g*;
else if (0 value easier to get at input of
  XOR/EQUIV gate)
    c = 1;
else *c* = 0;
return (*l*, $\overline{c}$);

PODEM Algorithm

while (no fault effect at POs)
    if (*xpathcheck* (D-frontier)
        (l, v_l) = *Objective* (*fault*, v_{fault});
        (pi, v_{pi}) = *Backtrace* (l, v_l);
        Imply (pi, v_{pi});
        if (*PODEM* (*fault*, v_{fault}) == SUCCESS) return (SUCCESS);
        (pi, v_{pi}) = *Backtrack* ();
        Imply (pi, v_{pi});
        if (*PODEM* (*fault*, v_{fault}) == SUCCESS) return
          (SUCCESS);
        Imply (*pi*, “X”);
        return (FAILURE);
    else if (implication stack exhausted)
        return (FAILURE);
    else *Backtrack* ();
return (SUCCESS);

Summary

- D-ALG – First complete ATPG algorithm
  - *D-Cube*
  - *D-Calculus*
  - *Implications* – forward and backward
  - *Implication stack*
  - *Backup*
- PODEM
  - Expand decision tree only around PIs
  - Use *X-PATH-CHECK* to see if *D-frontier* exists
  - *Objectives* -- bring ATPG closer to getting D ($\overline{D}$) to PO
  - *Backtracing*