

ECE 553: TESTING AND TESTABLE DESIGN OF DIGITAL SYSTEMS

Combinational ATPG

Overview

Major ATPG algorithms

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

9/30/2009 2

Forward Implication

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

	b	0	1	X	D	\bar{D}
a	0	0	0	X	D	\bar{D}
1	0	0	1	X	D	\bar{D}
X	0	X	X	X	X	X
D	0	D	X	D	D	0
\bar{D}	0	D	X	0	\bar{D}	D

9/30/2009 3

Backward Implication

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

9/30/2009 4

Implication Stack, Decision Tree, and Backtrack

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

9/30/2009 5

Objectives and Backtracing in ATPG

- **Objective** – desired signal value goal for ATPG
 - Guides it away from infeasible/hard solutions
 - Uses heuristics
- **Backtrace** – Determines which primary input and value to set to achieve objective
 - Use heuristics such as nearest PI
- **Forward trace** – Determines gate through which the fault effect should be sensitized
 - Use heuristics such as output that is closest to the present fault effect

9/30/2009 6

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*
- *Backtracking* – Search fails, therefore undo some of the work completed and start searching from a location where search options still exist

9/30/2009 7

D-Algorithm – Roth (1966)

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

9/30/2009 8

Singular Cover Example

- Minimal set of logic signal assignments to represent a function
 - show *prime implicants and prime implicates* of Karnaugh map (with explicitly showing the outputs too)

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

9/30/2009 9

Primitive D-Cube of Failure

- **Models circuit faults:**
 - *Stuck-at-0*
 - *Stuck-at-1*
 - Other faults, such as *Bridging fault* (short circuit)
 - Arbitrary change in logic function
- AND Output sa0: “1 1 **D**”
- AND Output sa1: “0 X **D**”
“X 0 **D**”
- Wire sa0: “**D**”
- *Propagation D-cube* – models conditions under which fault effect propagates through gate

9/30/2009 10

Construction of Primitive D-Cubes of Failure

1. Make cube set α_1 when good machine output is 1 and set α_0 when good machine output is 0
2. Make cube set β_1 when failing machine output is 1 and β_0 when it is 0
3. Change α_1 outputs to 0 and D-intersect each cube with every β_0 . If intersection works, change output of cube to D
4. Change α_0 outputs to 1 and D-intersect each cube with every β_1 . If intersection works, change output of cube to D

9/30/2009 11

Gate Function Change D-Cube of Failure

Cube-set	a b c	Cube-set	a b c
α_0	0 X 0 X 0 0	PDFs for AND changing to OR	0 1 \overline{D}
α_1	1 1 1		1 0 \overline{D}
β_0	0 0 0		
β_1	1 X 1 X 1 1		

9/30/2009 12

Propagation D-Cube

- Collapsed truth table entry to characterize logic
- Use Roth's 5-valued algebra
- AND gate: use the rules given earlier using α and β but in this case work with good circuit only

Write all primitive Cubes of AND gate and then create propagation cubes	A	B	d
	D	1	D
	1	D	D
	\overline{D}	\overline{D}	\overline{D}
	\overline{D}	\overline{D}	\overline{D}
	1	D	\overline{D}
	\overline{D}	1	\overline{D}

9/30/2009 13

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** –

\cap	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}
\overline{D}	ψ	ψ	\overline{D}	μ	λ
\overline{D}	ψ	ψ	\overline{D}	λ	μ

9/30/2009 14

Implication Procedure

- Model fault with appropriate *primitive D-cube of failure* (PDF)
- Select *propagation D-cubes* to propagate fault effect to a circuit output (*D-drive* procedure)
- 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

9/30/2009 15

D-Algorithm – Top Level

- Number all circuit lines in increasing level order from PIs to POs;
- 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*;
- D-drive* ();
- Consistency* ();
- return ();

9/30/2009 16

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 D in test cube) Backtrack ();
  else if (intersection fails) break;
if (all fault effects unpropagatable) Backtrack ();
    
```

9/30/2009 17

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;
    
```

9/30/2009 18

Example 7.3 – Step 2 *u sa1*

- Forward and backward implications

(a,b) means that the line has $CC0 = a$ and $CC1 = b$

9/30/2009 25

Inconsistent

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

9/30/2009 26

Example 7.3 – Backtrack

- Need alternate propagation D-cube for v

(a,b) means that the line has $CC0 = a$ and $CC1 = b$

9/30/2009 27

Example 7.3 – Step 3 *u sa1*

- Propagation D-cube for v

(a,b) means that the line has $CC0 = a$ and $CC1 = b$

9/30/2009 28

Example 7.3 – Step 4 *u sa1*

- Propagation D-cube for Z

(a,b) means that the line has $CC0 = a$ and $CC1 = b$

9/30/2009 29

Example 7.3 – Step 4 *u sa1*

- Propagation D-cube for Z and implications

(a,b) means that the line has $CC0 = a$ and $CC1 = b$

9/30/2009 30

PODEM -- Goel (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 $D(\bar{D})$ to PO
 - *Backtracing*

9/30/2009

31

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

9/30/2009

32

PODEM High-Level Flow

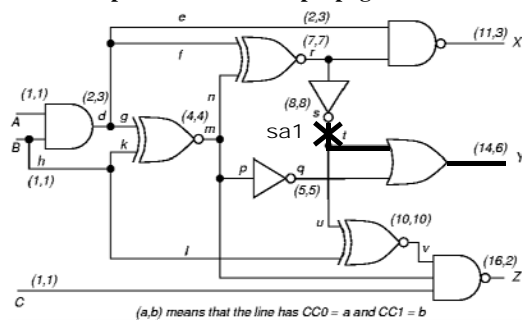
1. Assign binary value to unassigned PI
2. Determine implications of all PIs
3. Test Generated? If so, done.
4. Test possible with more assigned PIs? If maybe, go to Step 1
5. Is there untried combination of values on assigned PIs? If not, exit: untestable fault
6. Set untried combination of values on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

9/30/2009

33

Example 7.3 Again

- Select path $s - Y$ for fault propagation

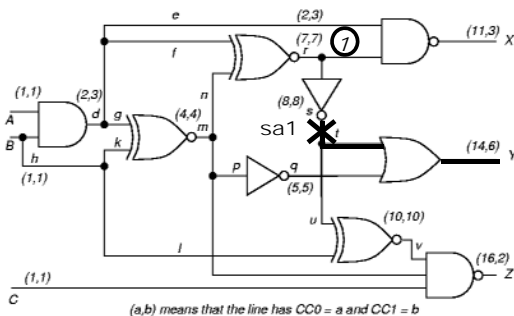


9/30/2009

34

Example 7.3 -- Step 2 s sa1

- Initial objective: Set r to 1 to excite fault

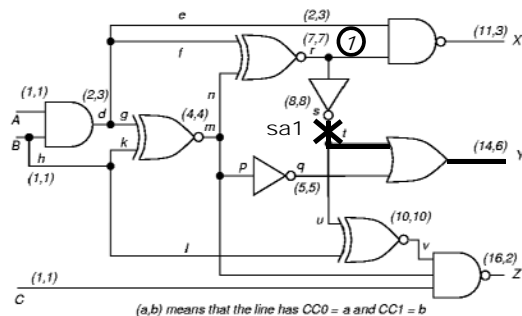


9/30/2009

35

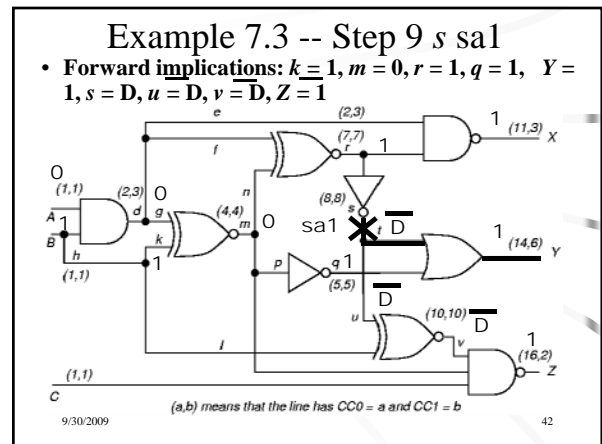
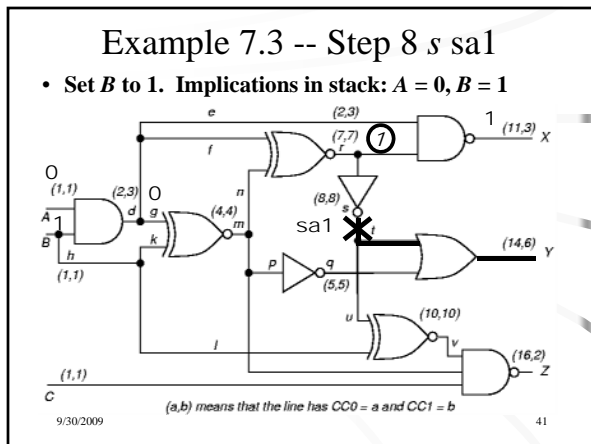
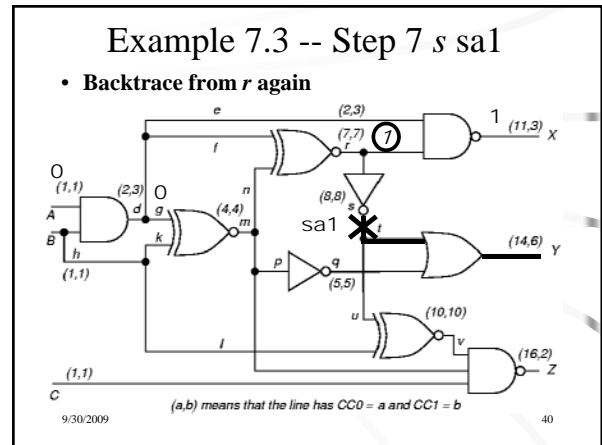
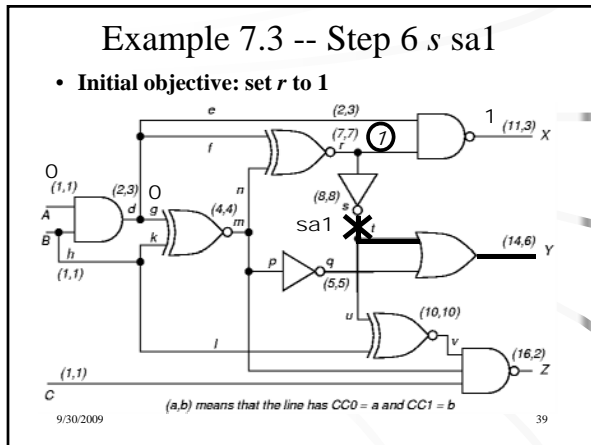
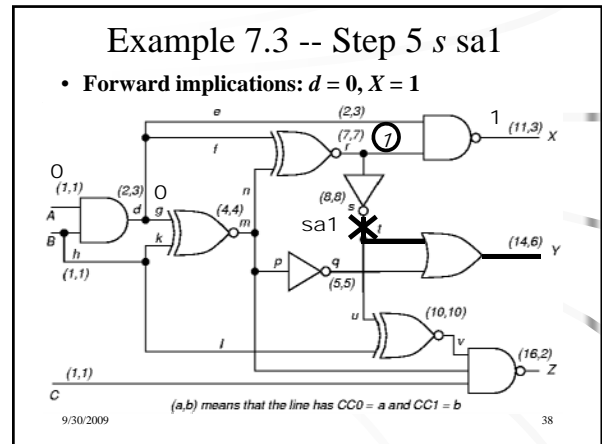
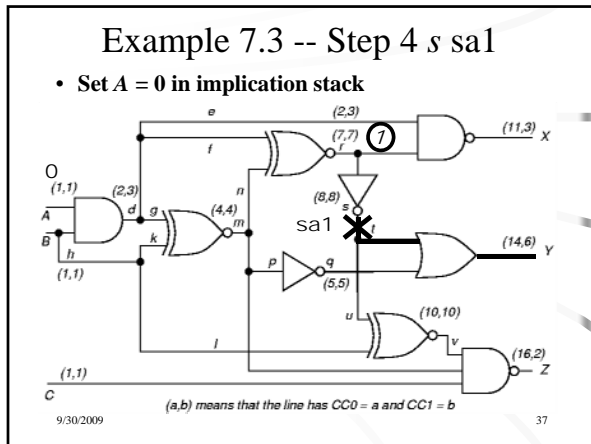
Example 7.3 -- Step 3 s sa1

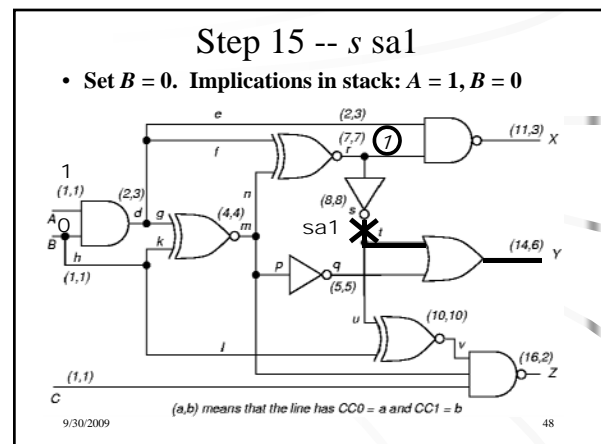
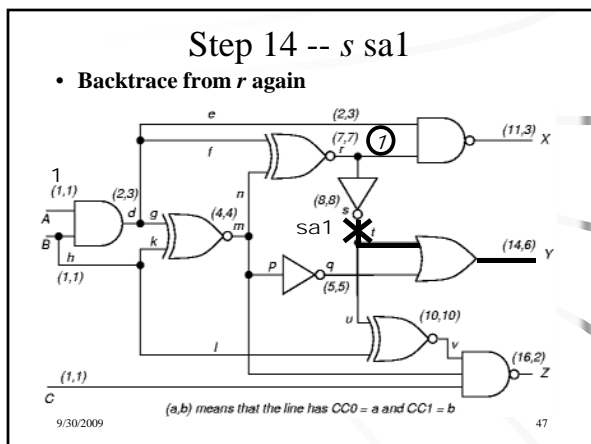
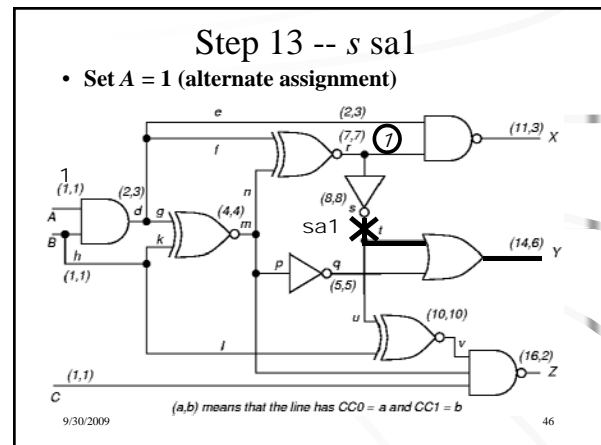
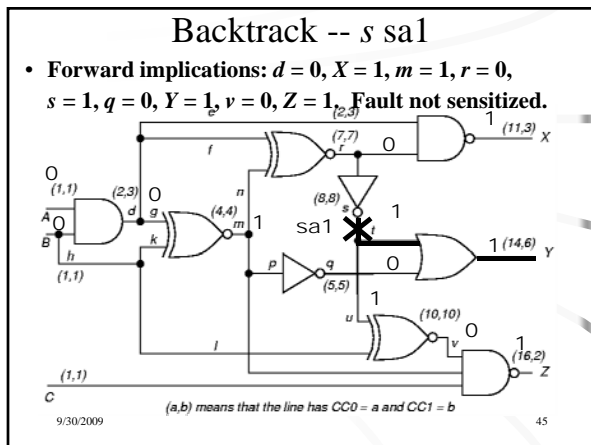
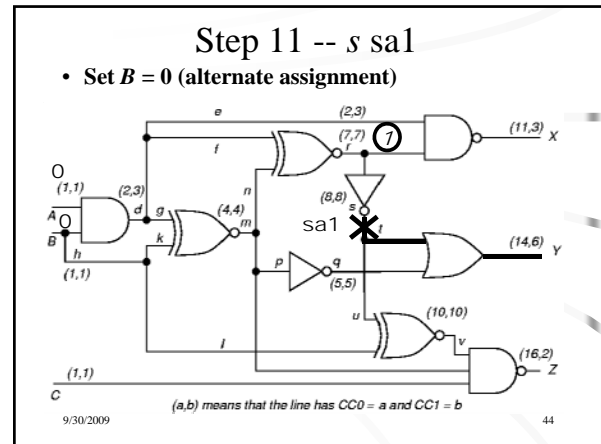
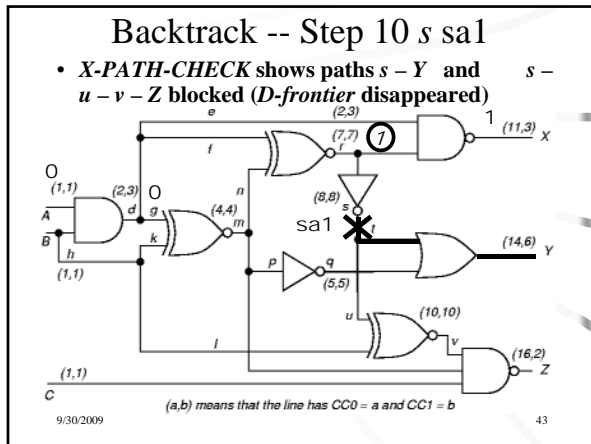
- Backtrace from r

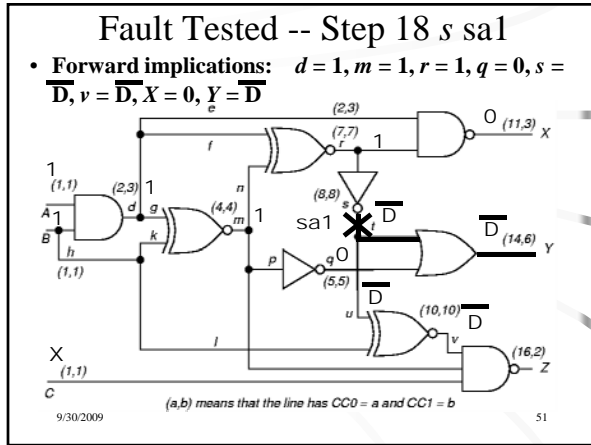
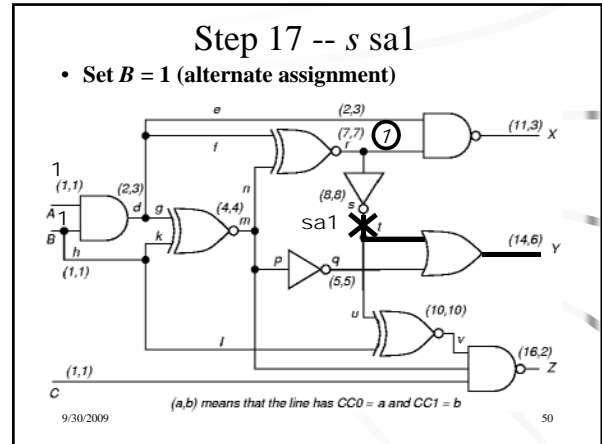
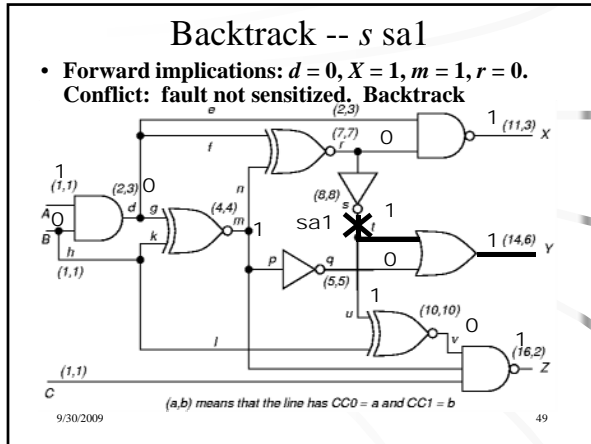


9/30/2009

36







Backtrace (s, v_s) Pseudo-Code

```

v = v_s;
while (s is a gate output)
  if (s is NAND or INVERTER or NOR) v = NOT 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 */;
    
```

9/30/2009 52

Objective Selection Code

```

if (gate g is unassigned) return (g, NOT 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, NOT c);
    
```

9/30/2009 53

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) = Backtrace ();
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 Backtrace ();
return (SUCCESS);
    
```

9/30/2009 54

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 (D)** to PO
 - *Backtracing*

9/30/2009 55

Appendices

9/30/2009 56

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

Stack ptr. ●

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

9/30/2009 57

Objectives and Backtracing in ATPG

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

9/30/2009 58

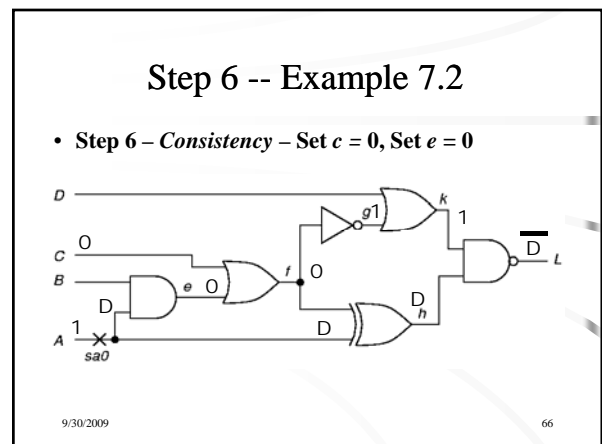
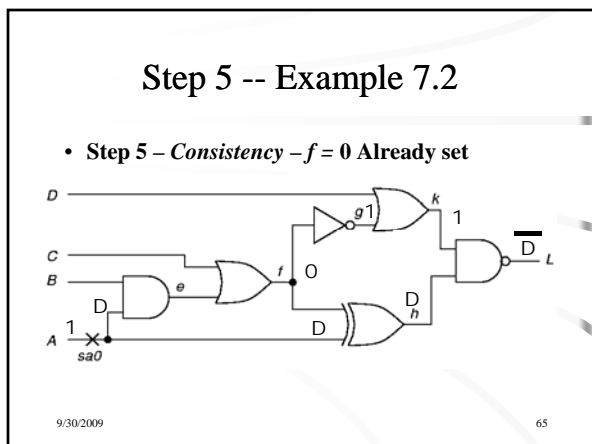
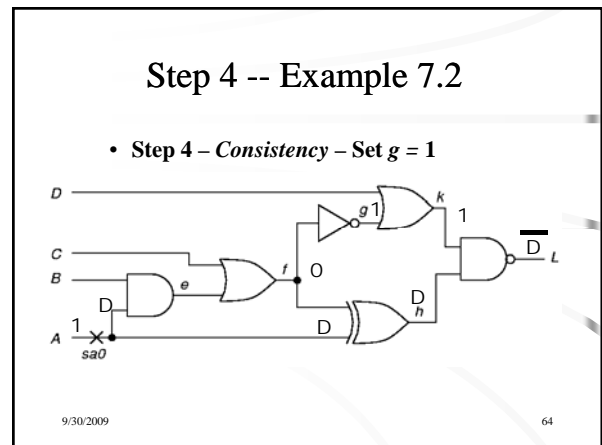
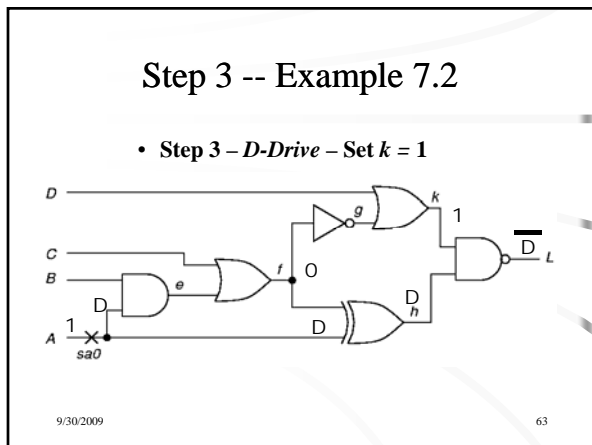
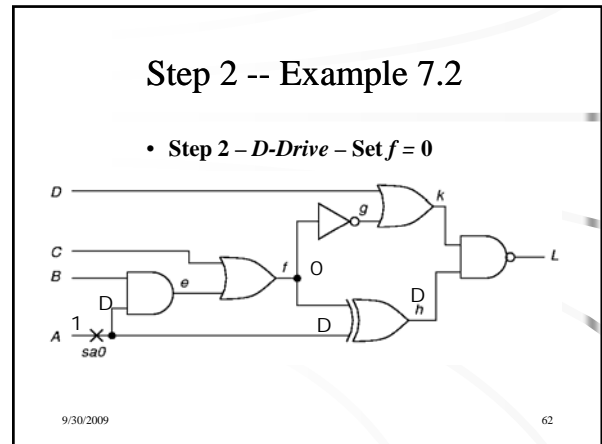
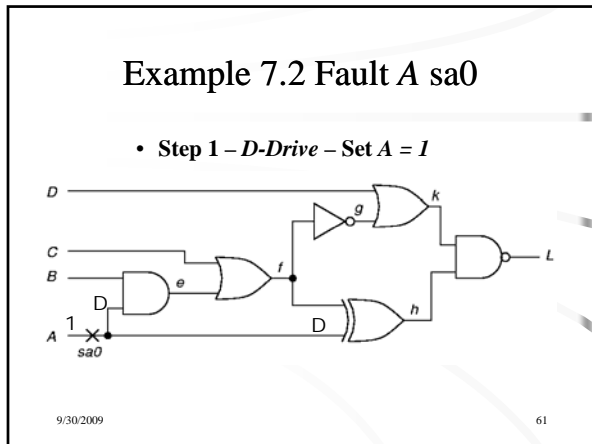
Bridging Fault Circuit

9/30/2009 59

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	D
$\alpha 1$	1	X	1	X		0	1	D	1
$\beta 0$	0	0	0	0					
$\beta 1$	X	1	1	1					
	1	X	1	1					

9/30/2009 60



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

9/30/2009 67

Example 7.3 – Fault sa1

- Primitive D-cube of Failure

(a,b) means that the line has CC0 = a and CC1 = b

9/30/2009 68

Example 7.3 – Step 2 sa1

- Propagation D-cube for v

(a,b) means that the line has CC0 = a and CC1 = b

9/30/2009 69

Example 7.3 – Step 2 sa1

- Forward & Backward Implications

(a,b) means that the line has CC0 = a and CC1 = b

9/30/2009 70

Example 7.3 – Step 3 sa1

- Propagation D-cube for Z – test found!

(a,b) means that the line has CC0 = a and CC1 = b

9/30/2009 71