Fault Tolerant Computing

CS 530

Test Generation

Yashwant K. Malaiya

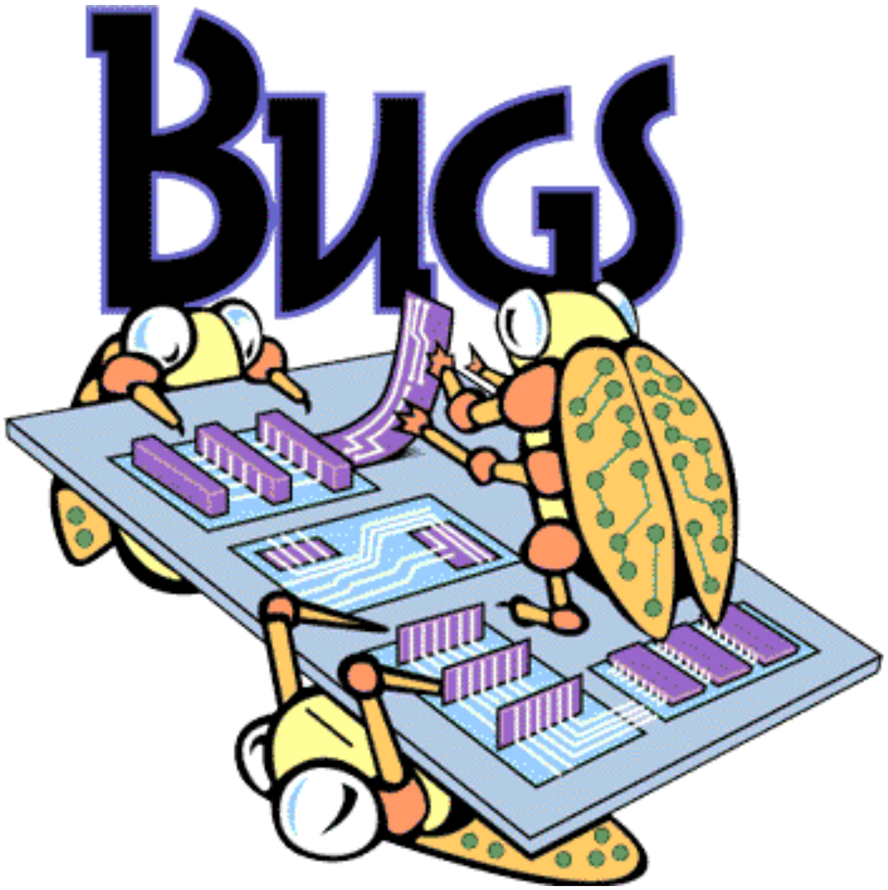
Colorado State University

January 28, 2021

Test Generation: Combinational

- Algebraic: Boolean difference
- Structural: D-notation
- Sensitized path, single-path propagation
- D-algorithm
- Fault-collapsing, Test set minimization

Testing for bugs



Testing

- We assume that tests are applied at the inputs and the response is observed at the outputs of the unit-under-test.
- A test detects the presence of a fault(s), if the output is different from the expected output.
- Two test approaches:
  - Functional (or Black-box): uses only the functional description of the unit, not its structure to obtain tests.
  - Structural testing: uses the structural information to generate tests. Requires more effort, but can be more thorough.

Random Testing

- Random testing is a form of functional testing. In random testing, each test is chosen such that it does not depend on past tests.
- In actual practice, the “random” tests are generated using Pseudo-random algorithms that approximate randomness.
- As we will discuss later, random testing can be effective for moderate degree of testing, but not for thorough testing.

Test coverage

- A single test typically covers (i.e. tests) for several potential faults.
- The coverage obtained by a test-set can be obtained using fault simulators for hardware.
- The test coverage achieved by a test-set is given by ratio:

$$\text{coverage} = \frac{\text{Number of faults covered}}{\text{Total number of possible faults}}$$

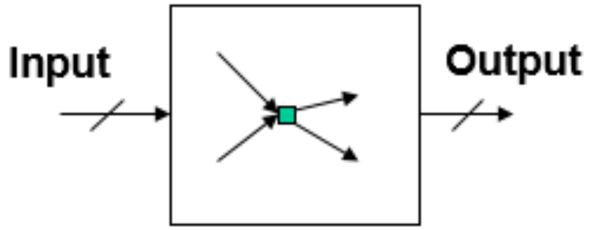
- By convention, coverage is evaluated for stuck-at 0/1 faults in hardware, often given in percentage.

Testing for Individual Faults

- First we consider structural testing for individual faults (test generation problem).
- We then consider reducing the number of faults to be considered (fault collapsing problem).
- Next we consider reducing the number of tests that need to be applied (test-set compaction problem).

Test generation: Some Basics (1)

- Approaches:
  - Symbolic
  - Based on heuristics
- Needed:
  - Fault excitation: triggering the fault to create error
  - Error propagation: propagating error to the output
- Notation:
  - normal function f,
  - faulty f_α with fault α
- Vector $\hat{a}=(a_1, a_2, ..a_n)$ is a test if $f(\hat{a}) \neq f_\alpha(\hat{a})$
- All tests are contained in expression $T = f \oplus f_\alpha$

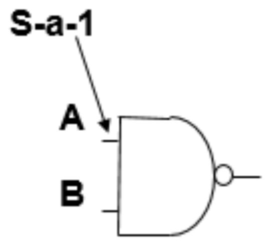


Test generation: Some Basics (2)

- **All tests are contained in T, where T = $f \oplus f_\alpha$**

i.e. T is the set of vectors for which normal and faulty outputs are different.

Example:



A \ B	0	1
0	1	1
1	1	0

f = (AB)’

A \ B	0	1
0	1	0
1	1	0

f_α = B’

A \ B	0	1
0	0	1
1	0	0

$f \oplus f_\alpha$

T = A’B (01) is a test. The only test.

$f \oplus f\alpha$ is 1 for combinations for which Karnaugh maps of f and $f\alpha$ are different.

Boolean Difference Method

Theorem: Assume input x_i has fault α which is s-a-0. Then set of tests is given by

$$T = x_i \frac{df}{dx_i}$$

$$\text{where } \frac{df}{dx_i} = f(x_1, \cdots x_{i-1}, x_i, x_{i+1}, \cdots x_n) \oplus f(x_1, \cdots x_{i-1}, \bar{x}_i, x_{i+1}, \cdots x_n)$$

$$= f(x_1, \cdots x_{i-1}, 0, x_{i+1}, \cdots x_n) \oplus f(x_1, \cdots x_{i-1}, 1, x_{i+1}, \cdots x_n)$$

$$= f_i(0) \oplus f_i(1)$$

This is compact notation for expression above

- **Note that *Boolean Difference* df/dx_i represents conditions for which output is susceptible to input x_i.**

Boolean difference (2)

- **Proof:**

Using Shanon's expansion theorem which states that

$f(x_i) = \bar{x}_i f_i(0) + x_i f_i(1)$

Note that $f_\alpha(X) = f_i(0)$

$$
\begin{aligned}
T &= f(X) \oplus f_\alpha(X) \\
&= (\bar{x}_i f_i(0) + x_i f_i(1)) \oplus f_i(0) \\
&= x_i f_i(1) \bar{f}_i(0) + x_i \bar{f}_i(1) f_i(0) \\
&= x_i (f_i(1) \oplus f_i(0))
\end{aligned}
$$

What about

x_i s-a-1?

Answer: use $\bar{x}_i$

Boolean difference (3)

- **Proof: Details**

$$T = f(X) \oplus f_\alpha(X)$$

$$= (\bar{x}_i f_i(0) + x_i f_i(1)) \oplus f_i(0)$$

$$= (\bar{x}_i f_i(0) + x_i f_i(1)).\overline{f_i(0)} + \overline{(\bar{x}_i f_i(0) + x_i f_i(1))}.f_i(0)$$

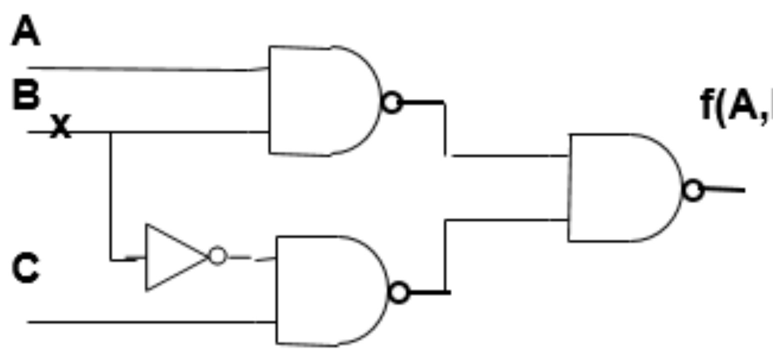
$$= x_i f_i(1)\overline{f_i(0)} + [(x_i + \bar{f}_i(0))(\bar{x}_i + \bar{f}_i(1)]f_i(0)$$

$$= x_i f_i(1)\overline{f_i(0)} + [x_i\bar{x}_i + x_i\bar{f}_i(1) + \bar{x}_i\bar{f}_i(0) + \bar{f}_i(0)\bar{f}_i(1)]f_i(0)$$

$$= x_i f_i(1)\bar{f}_i(0) + x_i\bar{f}_i(1)f_i(0)$$

$$= x_i(f_i(1) \oplus f_i(0))$$

Boolean Difference: Example



$f(A,B,C)=AB+\overline{B}C$

- Consider fault B s-a-1
- Tests are $T=\overline{B}\ df/dB$

$df/dB \quad = f(A,0,C) \oplus f(A,1,C)$

$\quad =C\oplus A \ \ =A\overline{C}+\overline{A}C$

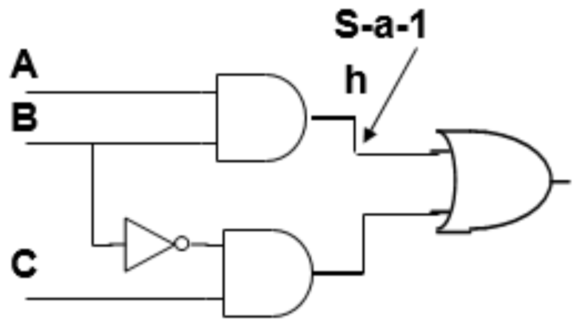
Hence

$T \quad =\overline{B}\ (A\overline{C}+\overline{A}C) = A\overline{B}\overline{C}+\overline{A}\overline{B}C$

$\quad =(100,001)$

Boolean Difference: Internal Nodes

- **Consider an internal node h=h(X) s-a-1. Express the original function f(X) as $f_h(X,h)$. Tests for h s-a-1 are given by $\bar{h}(X)\, df_h(X,h)/dh$.**



$f(A,B,C)=AB+\bar{B}C \quad h(A,B)=AB$

$f_h(B,C,h)=h+\bar{B}C$

$df_h/dh = f_h(0,B,C) \oplus f_h(1,B,C) = (\bar{B}C) \oplus 1$

$= \overline{\bar{B}C} = B+\bar{C}$

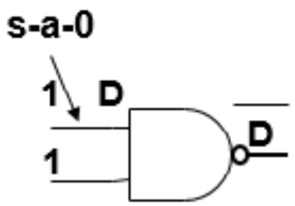
$T = \bar{h}\, df_h/dh = \overline{(AB)}(B+\bar{C}) = (\bar{A}+\bar{B})(B+\bar{C}) = \bar{A}B+\bar{A}\bar{C}+\bar{B}\bar{C}$

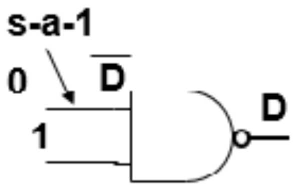
=010, 011, 000, 100 (four vectors!)

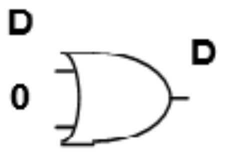
A \ BC	00	01	11	10
0	1	0	1	1
1	1	0	0	0

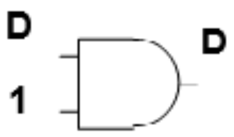
D-Notation

- Notation: Line has value D if it is 1 normally and 0 in presence of the fault. Line has value $\overline{D}$ if it is 0 normally and 1 in presence of the fault.





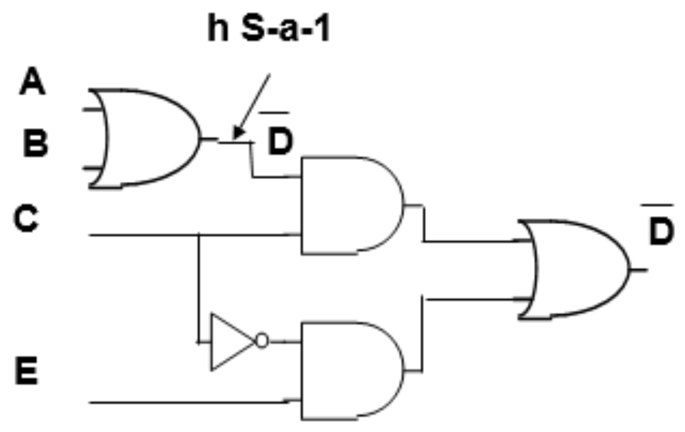




Rules of error propagation:

Gate	All other inputs
AND, NAND	1
OR, NOR	0
XOR	0, 1

Single Path Propagation



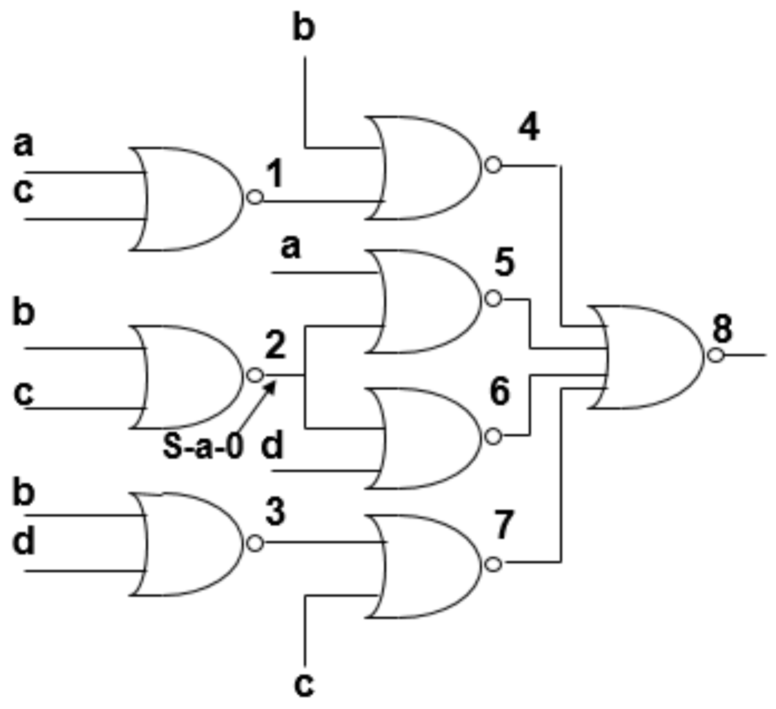
Single path propagation attempts to propagate error using a single path from the fault site to an output.

- **Excitation:**
  - h=0 normally. Need A,B=0,0
- **Propagation:**
  - Other AND input:1
  - Other OR input: 0
- **Justification:**
  - C=1 already. E=x (*don't care*)
- **Test is (0,0,1,x)**

Single Path Propagation may fail

- Single Path Propagation may fail in some cases even when a test exists.
- In the example in the next slide, an attempt to propagate an error using a strictly single path fails.
- In this example to propagate an error, the error needs to be propagated through multiple paths simultaneously.

Schneider's Counterexample



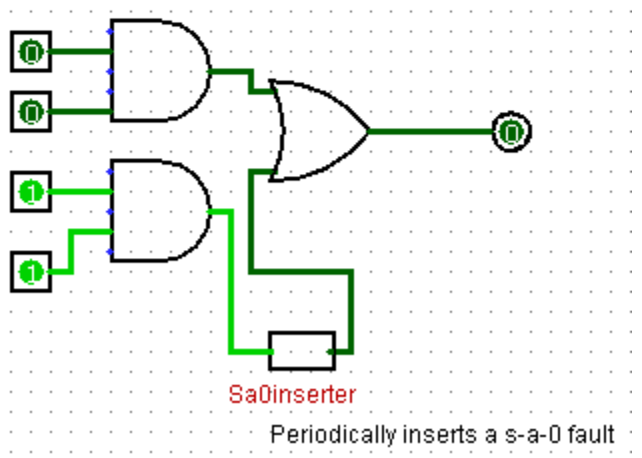
Try single path 2-6-8

- **Excitation: D at 2: b,c=0,0**
- **Forward trace:**
  - **$\overline{D}$ at 6: d=0**
  - **D at 8: 4,5,7=0,0,0**
- **Implication:**
  - **Since b=d=0, 3=1, 7=0**
- **Line Justification (backward trace):**
  - **For 5=0: a=1**
  - **Since abc=100, 1=0, 4=1 (!)**
  - **Inconsistency.**
- **Single path propagation fails.**

- **Multiple path propagation thru 5 and 6 works!**
- **b,c=0,0; a,d=0,0 Thus (0,0,0,0) is a test.**

Using Logisim

- Demonstration of Logisim
- Minimization
- Fault insertion



D-Algorithm

- Extension of single-path propagation
- Applicable for any type of elements (inc. gates)

Info used:

- Each normal element:
  - What other elements it is connected to
  - Its functional description
  - How to drive a D or $\overline{D}$ through it
- Faulty element:
  - how to get a D or $\overline{D}$ at its output

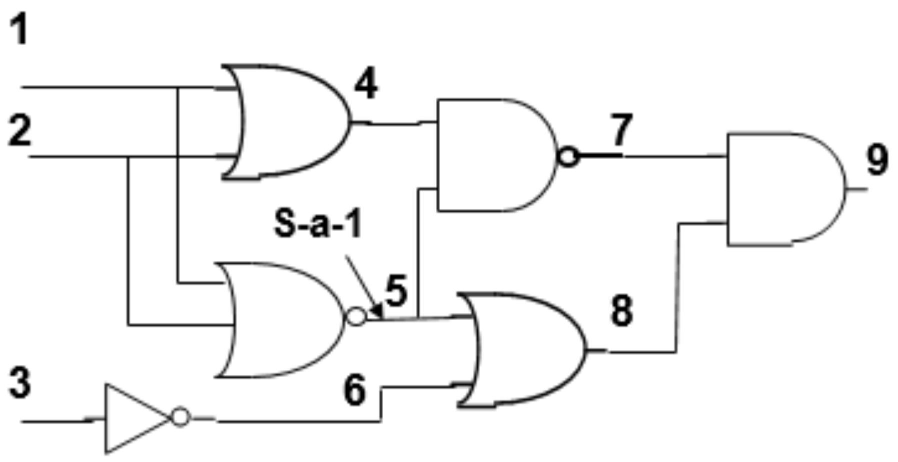
D-Algorithm: To find test for a given fault

- Excitation: Get D or $\overline{D}$ at a faulty element output
  - Do implication of the 0/1 values chosen*.
- D-drive: move D-frontier forward
  - Implication*
  - Repeat until a D or $\overline{D}$ at one output*
- Line justification
  - Justify all specified outputs of elements by having suitable inputs*
- * Backtrack to last point a choice existed

The ugly part

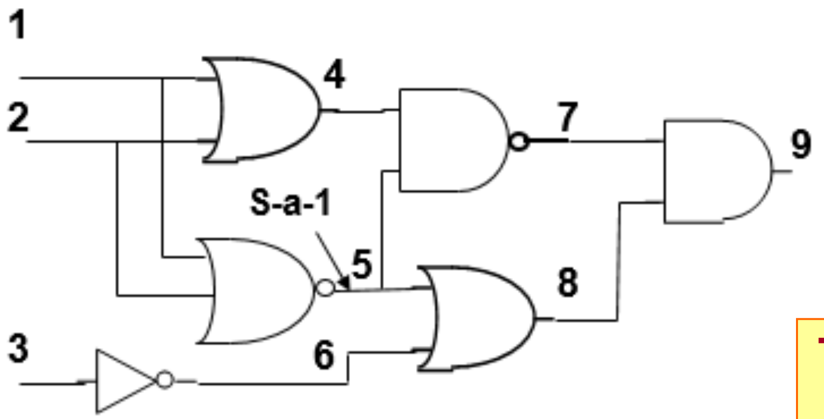
This is a compact description of the algorithm.

D-Algo: example: Part 1



- **Fault: NOR output s-a-1**
- **Excitation: 1,2 = 1,0 gets a $\overline{D}$ at 5**
- **Propagation to 9: through 7 or 8?** (*Choice*)
    - **Try 5-8-9 first**

D-Algorithm Ex (part 2)



Try: path 5-8-9

Table gives step-by-step values, until an inconsistency is observed

Step	1	2	3	4	5	6	7	8	9
Initial	1	0			$\overline{D}$				
5→8	1	0			$\overline{D}$	0		$\overline{D}$	
8→9	1	0			$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
4←7	1	0		0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
3←6	1	0	1	0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
1,2←4	ϕ	0	1	0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$

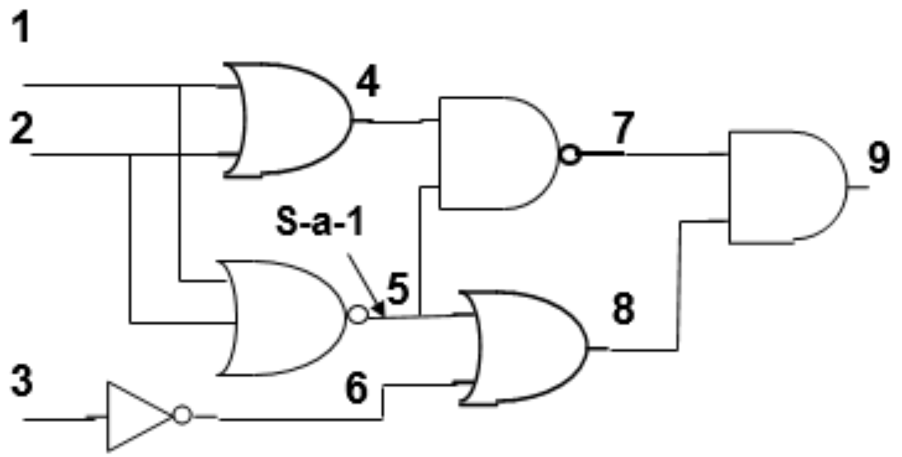
D-drive

Justifi-cation

Inconsistency!

Need to Backtrack

D-Algorithm Ex (3)



Try now: 5-7-9

Yes!

	1	2	3	4	5	6	7	8	9
	1	0			$\overline{D}$				
5→7	1	0		1	$\overline{D}$		D		
7→9	1	0		1	$\overline{D}$		D	1	D
6←8	1	0		1	$\overline{D}$	1	D	1	D
3←6	1	0	0	1	$\overline{D}$	1	D	1	D

D-drive

Justifi-
cation

Improved Algorithms

- While the D-algorithm is basic and historically important, it is not efficient.
- Several efficient test generation algorithms have been developed and compared using large example circuits.

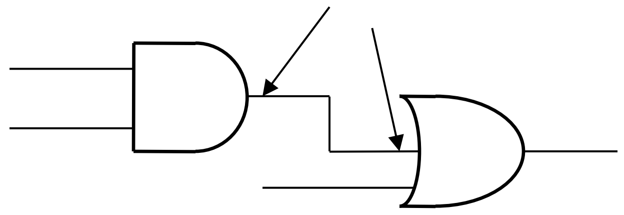
Combinational ATPG Algorithms

- **Automatic Test Pattern Generation (ATPG)** algorithms: searches are based on heuristics that generally work faster
  - **PODEM 1981**: x7 speedup relative to D-algorithm
  - **FAN 1983**: x23
  - **SOCRATES 1988**: x1574
  - **EST87651991**: x8765
  - **Tafertshofer 1997**: x25057
- Test generation is an "np-complete problem". No algorithm is known which will solve it in polynomial time (i.e. in n^r time, n=number of elements, r is some finite constant)
- It has been suggested that often computation time needed is of the order of n^3. **Prabhu Goel – PODEM, Verilog**

Fault Collapsing (1)

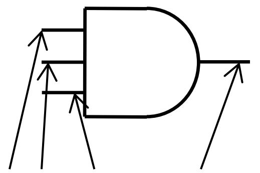
- **Fault Collapsing**: reducing the number of faults to be considered.
- Collapsing can be done using these
  - Equivalence property
  - Dominance property
- **Equivalence**: Faults α and β are equivalent if $f_\alpha = f_\beta$, i.e. if the two always generate the same response.

Stuck-at-0 at these two nodes are equivalent

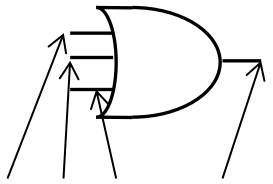


Fault Collapsing (2)

- **Equivalence: Faults α and β are equivalent if $f_\alpha = f_\beta$. Then α and β affect the output in exactly the same way.**



All s-a-0 equivalent



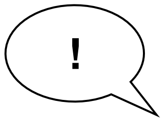
All s-a-1 equivalent

- **For an N-input gate only n+2 faults need to be considered**
- **Ex: NAND gate: we only need to consider**
  - **Any input s-a-0 or output s-a-1 (count as 1)**
  - **One input s-a-1 (total n such inputs)**
  - **Output s-a-0 (1)**
- **Termed *Equivalence fault collapsing***

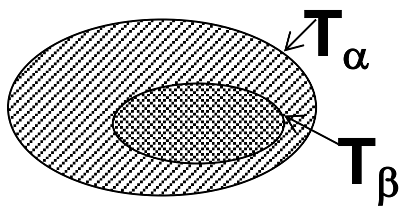
"Equivalence partitioning" in software testing

Fault Collapsing (2)

- **Dominance**: A fault α dominates fault β if $T_\beta \subset T_\alpha$.

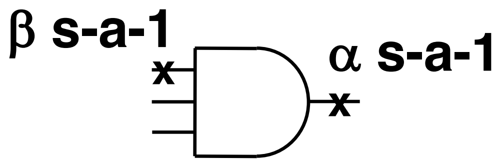


For detection only fault β needs to be considered. For location, both need to be considered separately (if distinguishable)



Detection only attempts to identify that the unit under test is faulty.

Example:



T_α= 0xx, x0x, xx0

T_β= 011

$\therefore T_\beta \subset T_\alpha$

(0,1,1) will test for both α and β. No need to use other tests if only detection is needed.

Check-points (1)

Here is a nice theorem:

- **Theorem:** In a fan-out free combinational circuit, any test set that detects all stuck faults on primary inputs will detect all stuck faults in the network.
- Note that the primary inputs are inputs to the unit-under-test coming from outside.

- **If there is fan-out? Here is a nice extension.**

SSFs=single stuck faults

Fault Collapsing: Check-points (2)

- **Theorem:** In a combinational circuit, any test set that detects all stuck faults on
  - all primary inputs and
  - All branches of fanout points

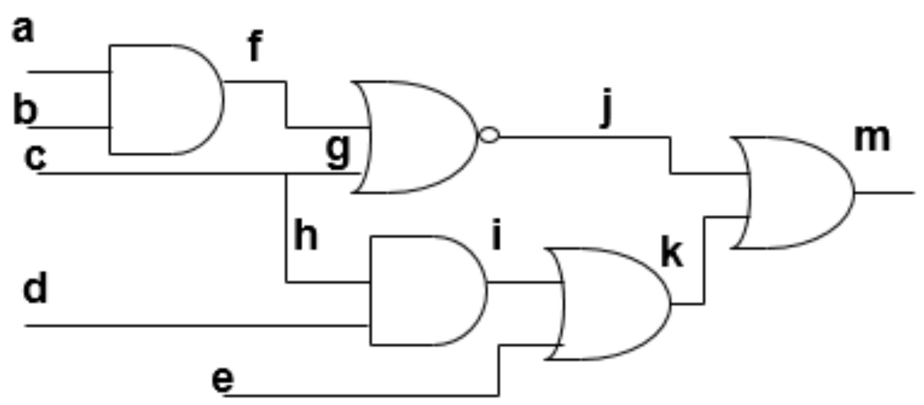
  will detect all stuck faults in the network.

These are appropriately called Checkpoints

Incidentally a check-point concept is also applicable for software testing

H. Yin, Z. Lebne-Dengel and Y. K. Malaiya, " Automatic Test Generation using Checkpoint Encoding and Antirandom Testing" Int. Symp. on Software Reliability Engineering, 1997, pp. 84-95.

Checkpoints: Example



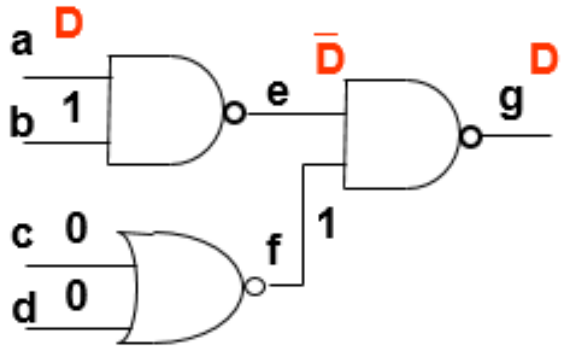
- 12 nodes, two faults at each node (s-a-0, s-a-1) thus 24 faults before collapsing.
- Checkpoints are:
  - Primary inputs: a,b,c,d, e
  - All branches of fan-out points: g,h
  - Faults at checkpoints 7x2=14 faults
- Thus only 14 out of 24 need to be considered.

Why Test Set Reduction works

Generally one pattern tests for several faults, as

- On a sensitized path a s-a-0 (s-a-1) on all nodes with D ($\overline{D}$) will be detected.

Example:



Sensitized path: a-e-g

(1100) will detect a s-a-0,

e s-a-1 and g s-a-0

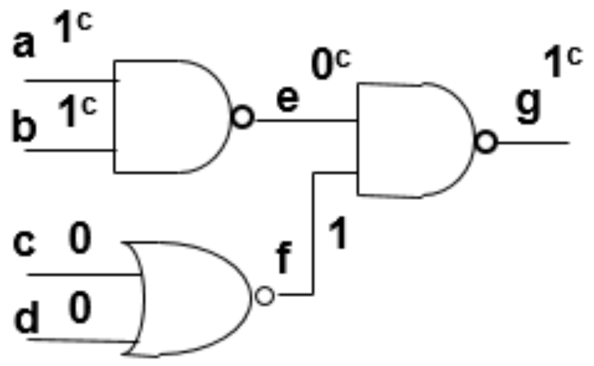
Why Test Set Reduction works

Generally one pattern tests for several faults, because

- **With a given vector, several nodes will be critical.**

A node is critical if a change in its logic value will change the output.

Example: Here the critical nodes are marked with a c. A node is critical only under a specific input vector, here (1,1,0,0).



(1100) will detect a s-a-0, b s-a-0, e s-a-1 and g s-a-0

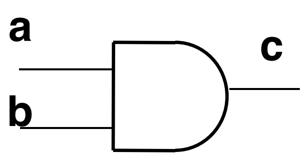
Test Set Reduction

- In test set reduction, object is to minimize the number of tests without losing fault detection capability.
- One approach, shown in the next slide to compile lists of faults detected by each vector.
- The other approach uses simulation. In a test set a test is dropped if it does not test for any new faults.

Test Set Compaction

- Minimize the number of patterns.

Example:



tests \ faults	a-0	a-1	b-0	b-1	c-0	c-1
00						√
01		√				√
10				√		√
11	√		√		√	

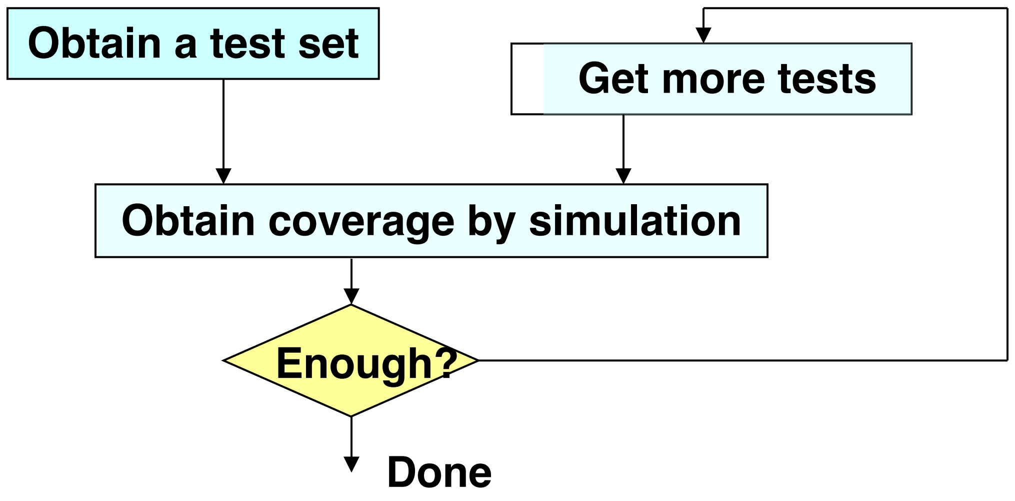
Minimum set: 01, 10, 11

Answer: 01, 10,11 will test for all the faults. Thus no need to apply 00.

In practice heuristics are used, complete optimization is not needed.

Coverage & Simulation

- **Coverage:** fraction of *all possible faults* covered by a test set. **Undetectable faults may or may not be counted.**
- Simulation can be used to determine coverage.
- Complete (100%) coverage is not feasible for very complex systems.



Fault distinction

- **Fault distinction** attempts to identify the specific fault that is present. The problem goes well beyond fault detection.
- In the next slide, the fault distinction problem is illustrated. In the **adaptive approach**, the results obtained in the past are exploited to cut down on tests needed. Test t1 tests for faults f1 and f2, but does not test for fault f3. Assuming there is one (and only one) fault, if there is no error when t1 is applied, fault f3 must be the one present.

Fault distinction

- **Preset test set:** no decision making during testing
- **Adaptive:** successive narrowing down

Problem: There is a fault. Is it f1, f2 or f3?

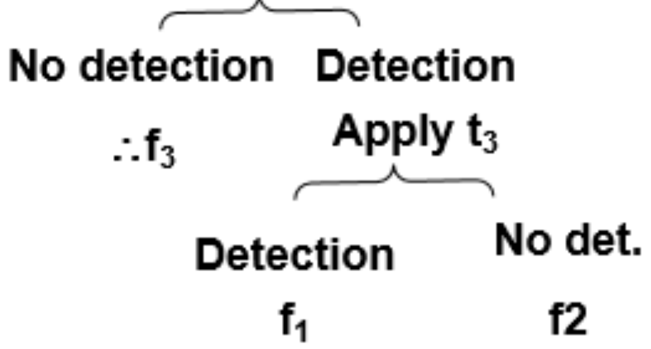
Fault	Test t_1	Test t_2	Test t_3
f_1	tests	doesn't	tests
f_2	tests	tests	doesn't
f_3	doesn't	tests	tests

•**Preset approach:**

•**Get response to t_1, t_2, t_3**

•**Then Identify.**

•**Adaptive: Apply t_1**



Assuming equal probability 1/3 for each fault, average number of tests to identify the fault= 2x 1/3+2x1/3+1 x1/3 = 1.7 vectors!

Guided probe for fault location

If you can probe inside, the fault distinction problem becomes easier.

- Apply inputs that cause an error. Start probing:
  - The error is not present at node A
  - but exists at a downstream node B,
  - Implies that the fault is somewhere between A and B.
  - Keep changing A and B until they correspond to the input and output of a single "element"
- Replace or fix the suspected element
- **Guided probe approach is applicable to both hardware and software.**

Test Generation: Summary

- Boolean difference $T = x_i \frac{df}{dx_i}$ $\frac{df}{dx_i} = f_i(0) \oplus f_i(1)$

- D (normally 1), excitation and propagation
- D-algorithm: backtrack if needed
- Equivalence/dominance collapsing, checkpoints
- Test set compaction
- Fault coverage and simulation
- Redundancy: undesirable during testing
- Fault distinction: preset vs. adaptive

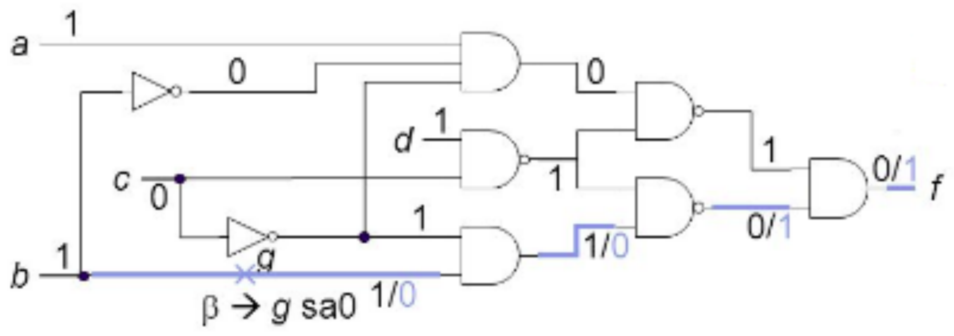
References

- **Supporting reading:** Design for Testability in Digital Integrated circuits, Bob Strunz, Colin Flanagan, Tim Hall http://www.cs.colostate.edu/~cs530/digital_testing.pdf
- "Fault tolerant and Fault Testable Digital Design" (Prentice hall International), Parag Lala.
- Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits, by Michael L. Bushnell, Vishwani D. Agrawal, Springer 2000.
- Test Pattern Generation And Test Application Time Reduction Algorithms For VLSI Circuits, Ilker Hamzaoglu, Dissertation (Introductory chapters only)
- I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits", Proc. of the Int. Conf. on Computer-Aided Design, November 1998.

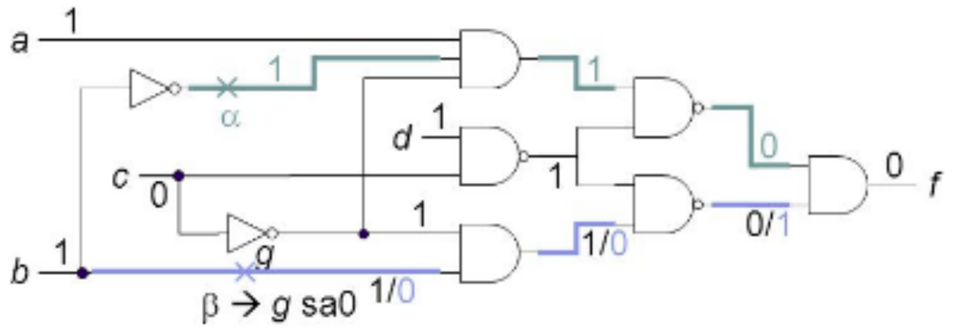
FAQ

- **Can a testable fault become untestable in presence of another fault?**

  - Fault β can be detected by test vector 1101. (X10X)



  - In the presence of the undetectabe fault α, the test vector 1101 becomes invalidate for fault β.



a
1
1
α
1
d
1
0
c
0
1
1
0
f
b
1
g
1/0
0/1
β → g sa0
1/0