

Outline

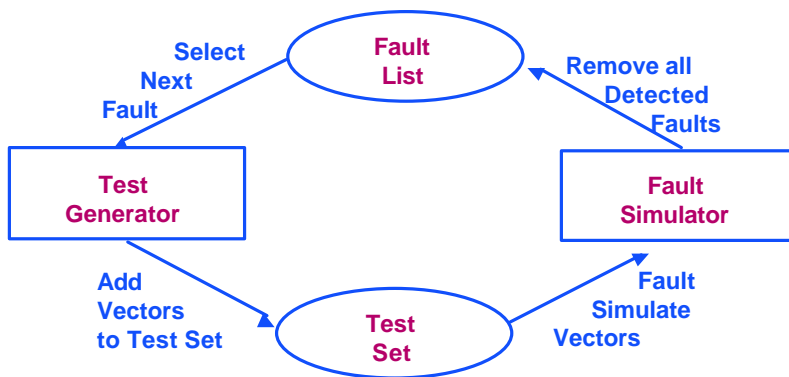
- Applications
- Why Two Fault Simulators Never Agree
- General Techniques
- Parallel Pattern Simulation
- Inactive Fault Removal
- Critical Path Tracing
- Fault Sampling
- Statistical Fault Analysis

Applications

- Fault grading a test set
- An aid to an ATPG
- Generates knowledge used in some ATPG systems
- Used to compute coverage loss due to signature analysis
- Generates fault dictionaries for use in fault diagnosis and location

Test Generation System

- Since ATPG is much slower than fault simulation, the fault list is trimmed with use of a fault simulator after each vector is generated

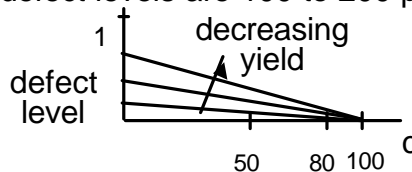


ECE 1767

University of Toronto

Fault Grading

- It has been shown that the quality of a shipped part heavily depends on the quality of a test.
 - ◆ **Defect Level = Reject Rate** = $\frac{\text{shipped defective parts}}{\text{total parts shipped as good}}$
 - ◆ c = single stuck-at fault coverage
 - ◆ d = defect coverage
 - ◆ **Assumption: $c \gg d$**
 - ◆ Y = inherent yield of a process = % of chips that are good
 - ◆ **defect level = $1 - Y^{1-c}$**
 - ◆ Current defect levels are 100 to 200 per million.



ECE 1767

University of Toronto

Why Two Fault Simulators Never Agree

- They simulate different sets of faults.
- Some report coverage on the collapsed fault list rather than the complete (uncollapsed) list.
- Fault collapsing algorithms may differ.
 - ◆ **Some count ALL gate input and output faults as different, even if a gate output without any fanout is identical to the input of the next gate.**
- Coverage reported on different classes of faults (e.g., exclude *untestable* faults from the fault list).
- Generate less accurate fault lists (e.g., do not distinguish between fanout and stem faults).

Why Two Fault Simulators Never Agree

- Some include potential detects in the regular fault coverage.
- Some will find a fault definitely detected while others may find it undetected due to *inaccuracy* in logic simulation involving unknown values
- Some start all flip-flops in a known state, since they do not handle unknowns

Fault Coverage Measures

$$\text{Fault Coverage} = \frac{\text{Detected Faults}}{\text{Total Faults}}$$

$$\text{Testable Coverage} = \frac{\text{Detected Faults}}{\text{Total Faults} - \text{Untestable}}$$

$$\text{ATG Efficiency} = \frac{\text{Detected} + \text{Untestable Faults}}{\text{Total Faults}}$$

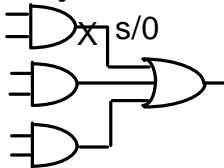
NOTE: 'Untestable' definition may be tool dependent!

General Techniques

- Serial
- Bit-Parallel
- Deductive
- Concurrent
- Parallel concurrent
- Parallel Pattern
- Inactive Fault Removal:
 - ◆ Star Algorithm
 - ◆ Critical Path Tracing

Serial Fault Simulation

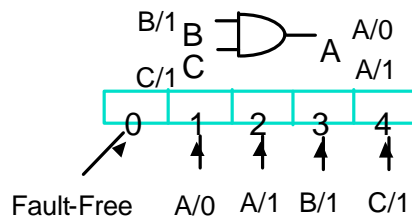
- Advantages
 - ◆ Easiest to implement if you have a logic simulator.
 - ◆ Works with any logic or timing simulator.
 - ◆ Fault insertion/injection is done by modifying the circuit.



- ◆ Any type of fault can be handled.
- Disadvantage
 - ◆ Consumes too much CPU time.

Bit-Parallel Fault Simulation

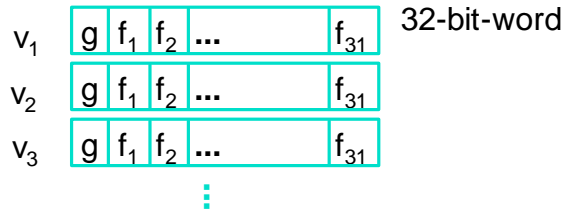
- A number of faults can be simulated in parallel by using the separate bits of a binary word to represent different logic values.
- Thus, the four faults that a 2-input AND gate can have can be associated with four bits as shown:



- Effectively, there are five different logic circuits being simulated, but the bitwise operation $A = B \cdot C$ is performed in one step.

Bit-Parallel Fault Simulation

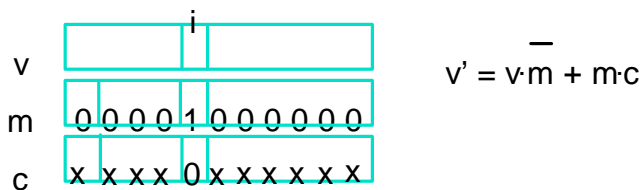
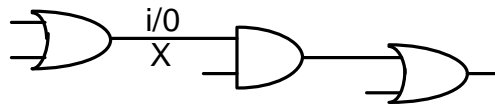
- If the word size W is smaller than the number of faults N_F , then a parallel fault simulator has to do multiple passes.
- 31 faults and the fault-free circuit are typically simulated in parallel using the bitwise parallelism of the computer word:



- ♦ Actually need two words to encode 3 values: 0, 1, x

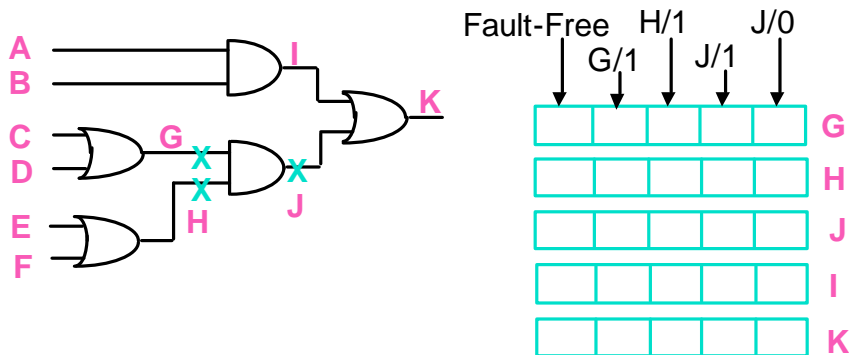
Bit-Parallel Fault Simulation

- Fault injection done using masks.
 - ♦ If fault needs to be injected on line then override the line value by fault value.



Bit-Parallel Fault Simulation Example

- Simulate faults for inputs ABCDEF = 000001



Deductive Fault Simulation

- Based on the concept of **fault-list propagation**.
- We associate with each node a list of faults that are detectable at that node.
- When we simulate a gate, we propagate the fault lists at its inputs to its output.
- The output fault list is *deduced* from those at inputs.
- Key attributes:
 - ◆ one pass, carry all faults at the same time
 - ◆ carry only active faults
 - ◆ symbolic representation of faults
 - ◆ basic operations used: set \subset , \cup , and difference

Deductive Fault Simulation

$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 1 \\
 \text{AND} \\
 1
 \end{array}
 \begin{array}{l}
 L_Z = \{a,c,d,f,g\} \\
 L_Z = L_A \mathbf{U} L_B
 \end{array}$$

$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 0 \\
 \text{AND} \\
 1
 \end{array}
 \begin{array}{l}
 L_Z = \{a,c\} \\
 L_Z = L_A - L_B = L_A \mathbf{C} \bar{L}_B \\
 \text{\{all faults in A which are not in B\}}
 \end{array}$$

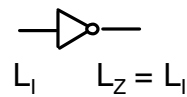
$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 0 \\
 \text{AND} \\
 0
 \end{array}
 \begin{array}{l}
 L_Z = \{f\} \\
 L_Z = L_A \mathbf{C} L_B
 \end{array}$$

Deductive Fault Simulation

$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 0 \\
 \text{AND} \\
 0
 \end{array}
 \begin{array}{l}
 L_Z = \{a,c,d,f,g\} \\
 L_Z = L_A \mathbf{U} L_B
 \end{array}$$

$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 1 \\
 \text{AND} \\
 0
 \end{array}
 \begin{array}{l}
 L_Z = \{a,c\} \\
 L_Z = L_A - L_B = L_A \mathbf{C} \bar{L}_B \\
 \text{\{all faults in A which are not in B\}}
 \end{array}$$

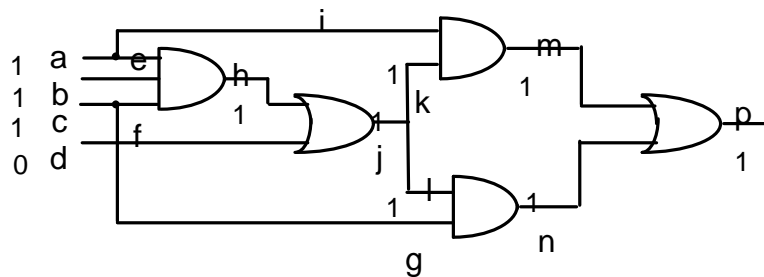
$$\begin{array}{l}
 L_A: \{a,c,f\} \\
 L_B: \{d,f,g\}
 \end{array}
 \begin{array}{c}
 1 \\
 \text{AND} \\
 1
 \end{array}
 \begin{array}{l}
 L_Z = \{f\} \\
 L_Z = L_A \mathbf{C} L_B
 \end{array}$$



Deductive Fault Simulation: Fault Injection

$L_Z = \{L_A \text{ } L_B\} \cup \{Z/1\}$	$L_Z = L_A \cup L_B \cup \{Z/1\}$	A=0, B=0
$L_Z = L_A - L_B \cup \{Z/1\}$	$L_Z = L_B - L_A \cup \{Z/0\}$	A=0, B=1
$L_Z = L_B - L_A \cup \{Z/1\}$	$L_Z = L_A - L_B \cup \{Z/0\}$	A=1, B=0
$L_Z = L_A \cup L_B \cup \{Z/0\}$	$L_Z = \{L_A \text{ } L_B\} \cup \{Z/0\}$	A=1, B=1

Deductive Fault Simulation Example



Excited faults:

$$a_0 = \{a_0\}$$

$$c_0 = \{c_0\}$$

$$j_0 = \{j_0\}$$

$$h_0 = \{e_0, b_0, f_0, h_0\}$$

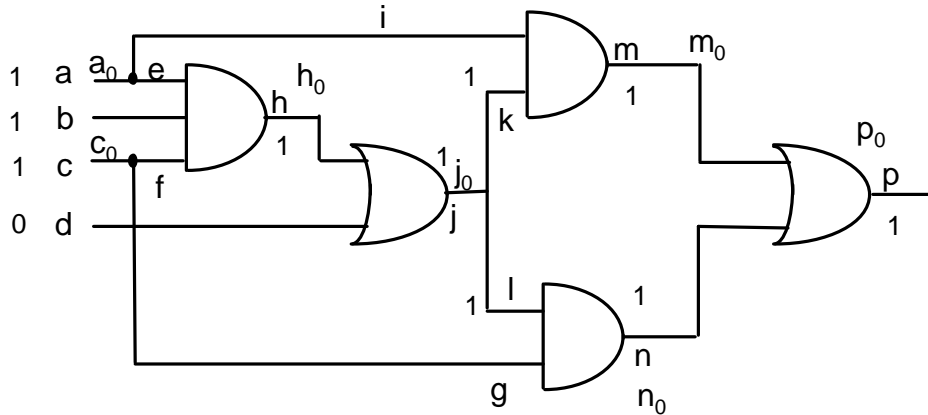
$$m_0 = \{i_0, k_0, m_0\}$$

$$n_0 = \{l_0, g_0, n_0\}$$

$$p_0 = \{p_0\}$$

collapsed fault list with
equivalence classes

Deductive Fault Simulation Example



ECE 1767

University of Toronto

Deductive Fault Simulation with Unknowns

- Less pessimistic (two lists per node)

$$\begin{array}{c}
 \{ \}_0 \{ \}_U \text{ A} \\
 \{ \}_0 \{ \}_U \text{ B}
 \end{array}
 \begin{array}{c}
 1 \\
 \text{AND} \\
 1
 \end{array}
 \begin{array}{c}
 1 \\
 \text{Z}
 \end{array}
 \quad
 \begin{array}{l}
 L_Z^0 = L_A^0 \cup L_B^0 \\
 L_Z^U = \{L_A^U - L_B^0\} \cup \{L_B^U - L_A^0\}
 \end{array}$$

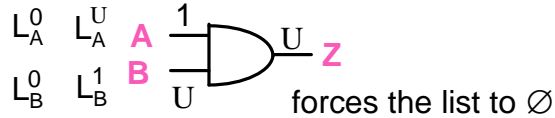
$$\begin{array}{c}
 L_A^0 \quad L_A^U \text{ A} \\
 L_B^0 \quad L_B^1 \text{ B}
 \end{array}
 \begin{array}{c}
 1 \\
 \text{AND} \\
 U
 \end{array}
 \begin{array}{c}
 U \\
 \text{Z}
 \end{array}
 \quad
 \begin{array}{l}
 L_Z^0 = L_A^0 \cup L_B^0 \\
 L_Z^1 = L_B^1 - L_A^0 - L_A^U
 \end{array}$$

ECE 1767

University of Toronto

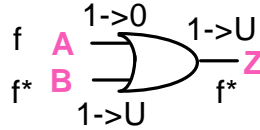
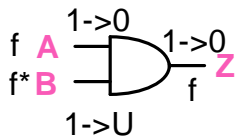
Deductive Fault Simulation with Unknowns

- More pessimistic (single list per node)
 - ◆ If good value is unknown, then no fault propagates on that line.



- ◆ If faulty value is unknown, mark fault with * and follow propagation rules below.

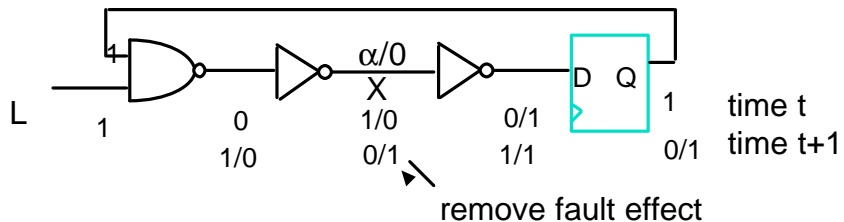
$$\begin{aligned}
 f \cap f^* &= f^* & f \cup f^* &= f & f - f^* &= f^* & f^* - f &= \emptyset \\
 g \cap f^* &= \emptyset & g \cup f^* &= \{g, f^*\} & g - f^* &= g & f^* - g &= f^*
 \end{aligned}$$



use same rules as for 2-value deductive fault simulation

Sequential Circuits and Feedback

- A fault list L containing fault $\alpha/0$ may propagate back to line α which now has a good value of 0. Then fault $\alpha/0$ must be removed from the list.



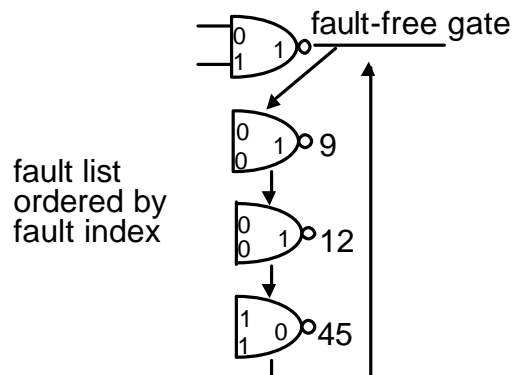
Performance

- In theory, deductive fault simulation can be very efficient.
 - ◆ **One pass simulates all faults.**
 - ◆ **Only active faults that will cause events are present in the list.**
- However, lists can be very large.
 - ◆ **Must implement efficient set union, intersection.**
 - ◆ **Keep lists sorted so that operations are linear.**
- Very hard to extend to non-logical fault behavior, e.g., multi-valued levels, hard to incorporate timing (essentially 0-delay simulator).

Concurrent Fault Simulation

- Like deductive fault simulation, this is also a one-pass approach, given enough memory, but several passes are often required in practice.
- Only those gates in the faulty circuit whose inputs/outputs are different from the fault-free circuit are (explicitly) simulated.
 - ◆ **Input and output values are stored for these gates.**
 - ◆ **The simulation algorithm is similar to event-driven logic simulation, but events are for the fault-free and faulty circuits.**

Concurrent Fault Simulation



- Concurrent fault simulation is faster, but more memory intensive than deductive fault simulation.
- Various timing models can be incorporated, so asynchronous circuits can be simulated accurately.

Parallel Concurrent Fault Simulation

- Faults are evaluated in parallel in groups of size w , where w is the size of the computer word.
- Fault effects are propagated using concurrent fault simulation algorithm.
- Faster and less memory than concurrent fault simulation.

Parallel Pattern Fault Simulation

- For combinational circuits only
- Evaluate 32 patterns (test vectors) in parallel for each fault
 - ◆ **vectors can be applied in any order**
 - ▲ evaluate good circuit
 - ▲ inject fault and evaluate
- Used in random testing
 - ◆ **keep vector that detects fault**
- Used in commercial fault simulator available from Mentor Graphics

Inactive Fault Removal

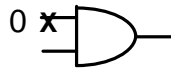
- Temporarily removing faults known to be inactive in a given time frame will speed up the fault simulation.
- Any faults having different states from the fault-free circuit must be simulated.

Inactive Fault Removal

- **Local Analysis:**

zero-level inactive

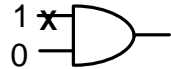
stuck-at-0



Fault not excited

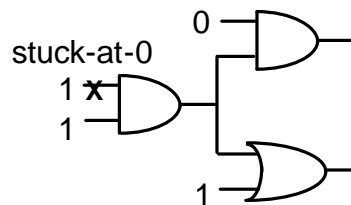
one-level inactive

stuck-at-0



Fault excited but not propagated one level

two-level inactive

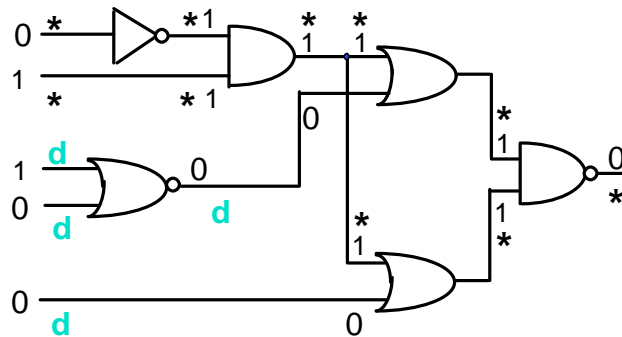


Fault excited but not propagated two levels

Inactive Fault Removal

- **Global Analysis:** Star Algorithm
[Akers, Krishnamurthy, Park, Swaminathan, ITC, 1990]
 - ◆ Evaluate good circuit
 - ◆ Place Star (*) on subset of nodes which adequately justify primary output values
 - ◆ Nodes not marked * are inactive, and faults on these nodes cannot be detected

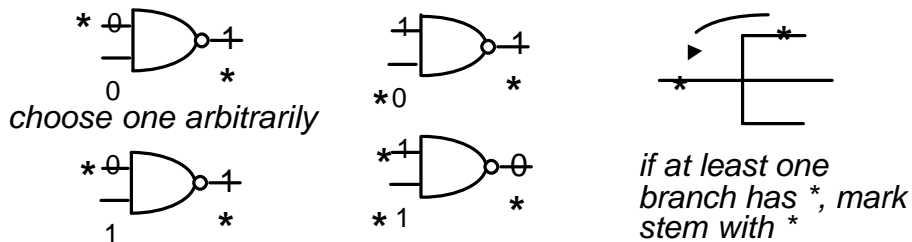
Star Algorithm for Combinational Circuits



d = don't care

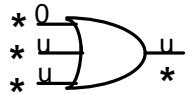
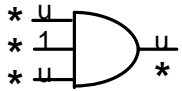
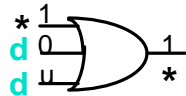
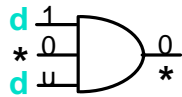
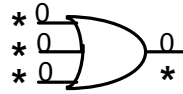
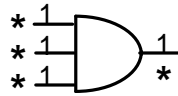
Star Algorithm

- Mark all outputs *
- Back propagate *'s to primary inputs using the following rules:

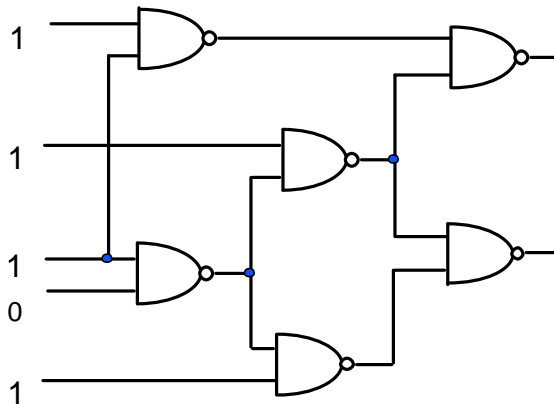


- All lines **not marked *** are **not required** to determine the outputs; hence, no faults on unstarred lines can be detected.

Star Backtracing for Sequential Circuits



Star Algorithm Example



Star Algorithm

- **Theorem:** All single and multiple stuck-at faults marked with “d” during a *single execution* of the algorithm are not detected by the vector and they need not be simulated
- **Theorem:** All single stuck-at faults marked with “d” in the *union of many executions* of the algorithm for different runs are also not detected

Critical Path Tracing

[Abramovici, Menon, Miller, IEEE D&T, 1984]

- Identify critical lines in circuit to implicitly determine detected faults for each test vector.
- Perform logic simulation, identify sensitive inputs of each gate, then backtrace from the primary outputs, connecting the dots.
- Exact algorithm for fanout-free circuits, but requires *stem analysis* for circuits with reconvergent fanout
- It can act as a “sandwich” with Star Algorithm to speed fault simulation

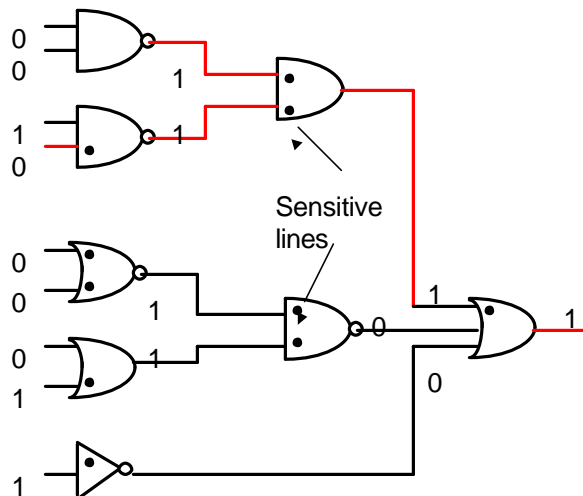
Critical Path Tracing

- Gate input is **sensitive** if complementing its value it changes the value of the gate output. A line l has **critical** value v for test vector t iff t detects the fault l stuck-at- v
- **Lemma:** If the output of a gate is critical then all sensitive inputs for this gate are also critical

ALGORITHM

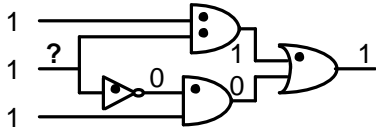
Simulate a vector and from every primary output, mark it as critical and Trace backwards towards primary inputs marking sensitive lines as Critical according to the lemma

Critical Path Tracing: Example

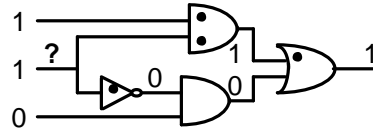


Critical Path Tracing

- CPT needs extra care for circuits with reconvergent fanouts



Line is not critical for vector 111
(s-a-0 not detected)



Line is critical for vector 110
(s-a-0 detected)

- Needs *stem analysis*, simulate s-a-0 on the line under consideration until next *capture line* and mark it if it is critical

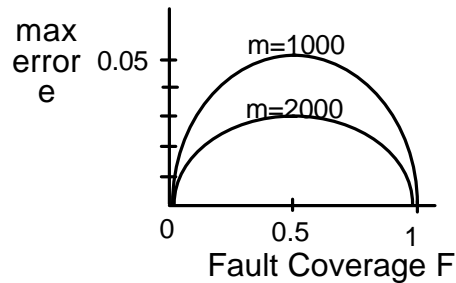
Fault Sampling

- Let
 - ◆ **T** = test set
 - ◆ **M** = total number of faults in collapsed fault list
 - ◆ **K** = number of faults detected in collapsed fault list
- Then fault coverage $F = K/M$.
- Select a random sample of **m** faults from **M** faults, and let **k** be the number of faults detected in the sample. Then the sample fault coverage $f = k/m$.
- How big should the sample be to get $F = f \pm e$ for a given error?

Fault Sampling

- Probability that test T will detect k faults in a random sample of size m is:

$$\frac{\binom{K}{k} \binom{M-K}{m-k}}{\binom{M}{m}}$$



$M \gg m$

Statistical Fault Analysis (STAFAN)

- No fault simulation is performed
 - Only a good circuit simulation is performed.
 - Based on signal probabilities.
- Let T = test set of n independent random vectors
- d_f = probability that a random vector from T detects the fault f
- Probability that none of the n vectors detects f = $(1-d_f)^n$
- Then the probability d_f^n of detecting f by n vectors = $1 - (1-d_f)^n$

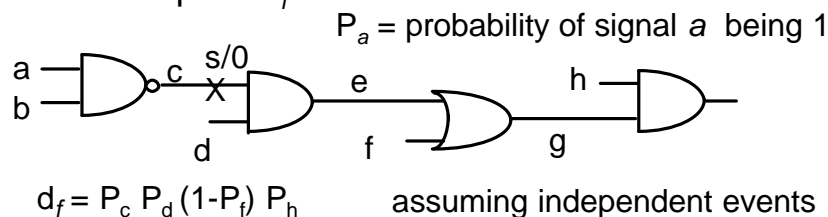
not power, just a notation

Statistical Fault Analysis (STAFAN)

- The the expected number of faults detected (D_n) from a set Φ of faults is: $D_n = \sum_{f \in \Phi} (1 - (1 - d_f)^n)$

- Expected fault coverage = $\frac{D_n}{|\Phi|}$

- How to compute d_f ?



Statistical Fault Analysis (STAFAN)

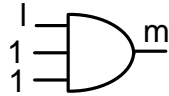
- 1-Controllability of line $l = C1(l) =$ probability that a randomly selected vector from T produces logic 1 on line l .
- $C0(l) = 1 - C1(l)$ if no unknowns.
- $O(l) =$ observability of line $l =$ probability that a randomly selected vector from T propagates the fault effect from l to a primary output (PO).
- During good circuit simulation, keep 0-count, 1-count

$$C0(l) = \frac{0\text{-count}}{n} \quad C1(l) = \frac{1\text{-count}}{n}$$

(signal probabilities)

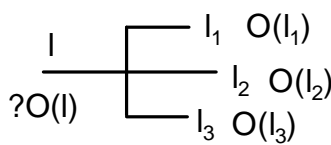
Statistical Fault Analysis (STAFAN)

- We also keep a sensitization count on each gate input
 - ◆ $S(l)$ = probability that line l is sensitized to next gate



line l is sensitized

- Observability of $l = S(l) O(m)$ $O(PO) = 1$
 - ◆ Traverse backwards from PO's to compute



lower bound: $O(l) = \max O(l_k)$

upper bound:

$O(l) = O(l_1) + O(l_2) - O(l_1)(l_2)$
(assuming independent lines)

Statistical Fault Analysis (STAFAN)

- Detection probabilities for line/fault $l/0$, $l/1$
 - ◆ $d_{l/0} = C1(l) O(l)$
 - ◆ $d_{l/1} = C0(l) O(l)$
- Group faults in 3 ranges:
 - ◆ high range: $d_f > 0.9$ --> considered detected
 - ◆ low range: $d_f < 0.1$ --> considered not detected
 - ◆ middle range: $0.1 < d_f < 0.9$ --> undecided
- Experimental results [Jain and Singer, 1986]
 - ◆ 91% to 98% of high range indeed detected
 - ◆ 78% to 98% of low range not detected
 - ◆ middle range contained 10% of faults on average