COE 571 Digital System Testing

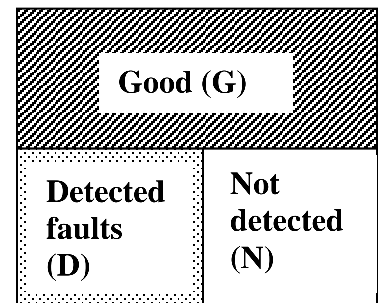
Dr. Aiman El-Maleh

Fault Simulation

1. **Defect level, test quality and yield**
2. **Serial fault simulation**
3. **Deductive fault simulation**
4. **Critical path tracing**
5. **Parallel fault simulation**
6. **Parallel pattern single fault propagation**
7. **Fault sampling**
8. **Statistical fault analysis**

How Much Testing is Required

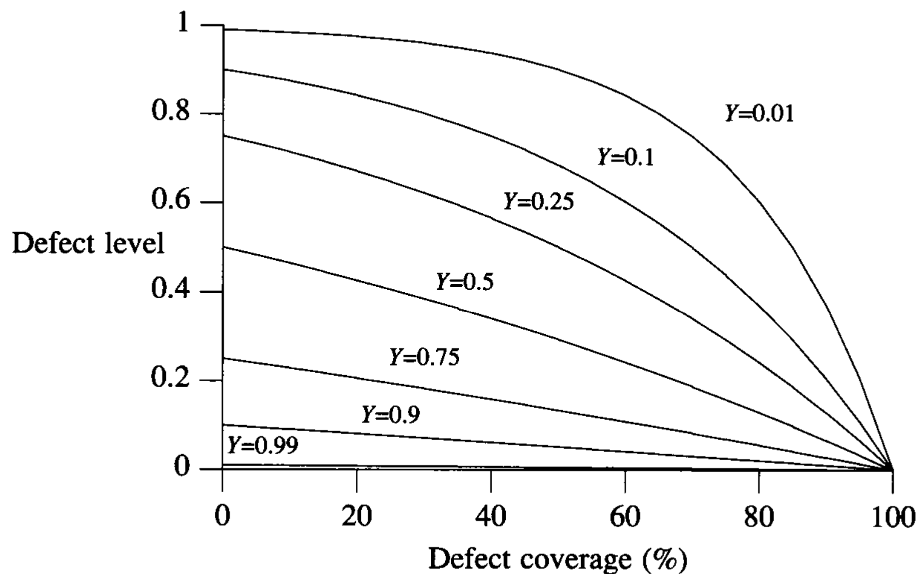
- **Defect Level (DL): probability of having a defective component in a batch of tested chips**
- **Goal of testing is to ensure DL is sufficiently small**
- **DL depends on Yield (Y) and Test Quality (Q)**
- **Yield (Y): percentage of good components of the total number fabricated**
- **Test Quality (Q): probability of detecting a fault in a defective component**



- **B = D + N**
- **Yield = Y = G/ (B+G)**
- **Quality = Q = D/(D+N) =(B-N)/B**
- **N = B(1-Q)**
- **G = YB/(1-Y)**
- $DL = \frac{N}{N+G} = \frac{(1-Q)(1-Y)}{Y+(1-Q)(1-Y)}$

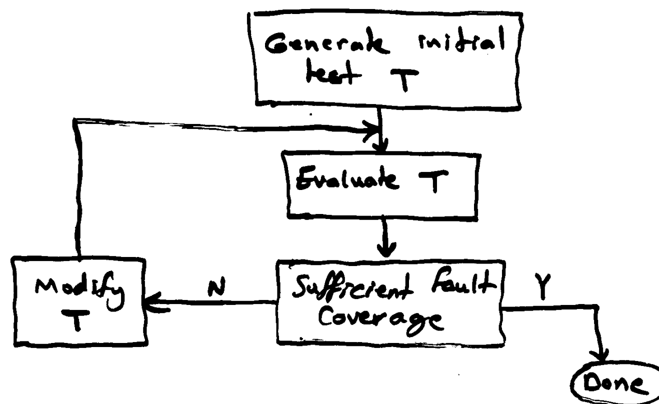
Defect Level, Test Quality and Yield

- **$DL = 1 - Y^{(1-Q)}$ (Williams & Brown)**
- **Example**
  - **Y=0.5, to get DL=0.01, Q=0.99**
  - **Y=0.5, Q=0.95 ⇒ DL=0.035**
  - **Y=0.8, Q=0.95 ⇒ DL=0.01**

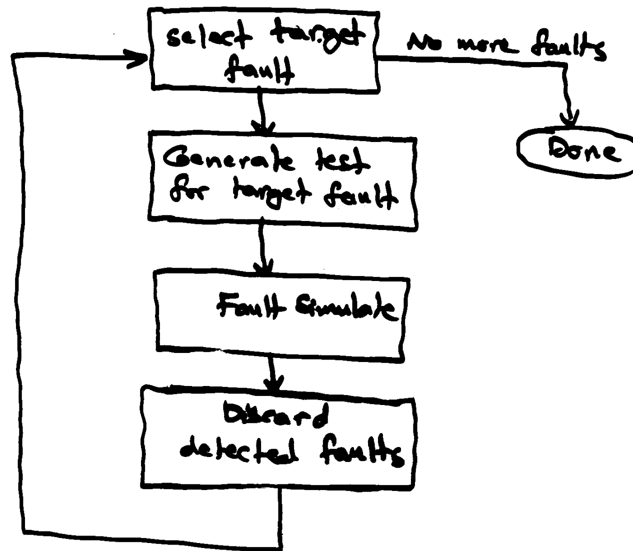


Fault Simulation

- Fault simulation consists of simulating a circuit in the presence of faults.
- Faults detected by a test T determined by comparing fault-free and faulty simulation results.
- Fault simulation is used to evaluate (grade) a test T.
- The grade of T is given by its fault coverage.
- Fault Coverage = $\frac{\text{No. of detected faults}}{\text{Total No. of faults}}$
- Fault simulation plays an important role in test generation.



- Test generation is fault-oriented. A test generated for a fault often detects other faults



- Fault simulation is also used to construct <u>fault dictionaries</u>

- A fault dictionary stores the output response to T (or a signature $S(R_f)$) of every faulty circuit N_f.

- Another application is to analyze the operation of a circuit in presence of faults.

Common Concepts & Terminology

- Serial fault Simulation:
  - Models the faulty circuit by transforming the fault-free N to N_f.
  - N_f is simulated using a logic simulator
  - process is repeated for each fault
  - can handle any type of fault provided the model of N_f is known
  - no special fault simulator is required
  - Impractical for simulating a large number of faults

- Other fault simulation techniques: parallel, deductive, concurrent
  - determine the behavior of circuit N in presence of faults without explicitly changing the model of N.
  - capable of simultaneously simulating a set of faults.
  - simultaneously simulate the fault-free circuit N and a set of faulty circuits $\{N_f\}$.

- Fault simulation involves the following tasks: fault specification, fault insertion, fault-effect generation & propagation, fault detection & discarding.

- Fault specification: defining set of modeled faults & performing fault collapsing

- Fault insertion: selecting a subset of faults to be simulated in one pass & creating data structures that indicate presence of faults to simulator.

- Data structures are used to generate effects of the inserted faults during simulation

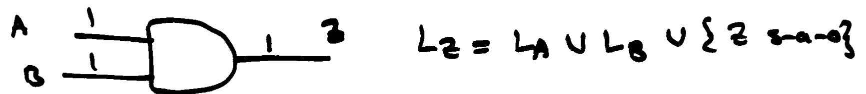
- Fault discarding (fault dropping): marking the fault as detected & removing it from list of faults to be simulated.

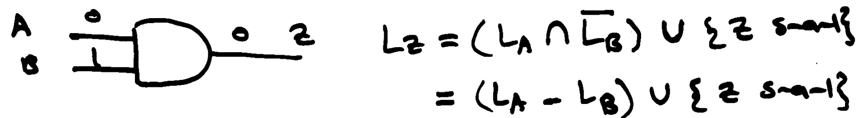
- <u>Deductive Fault Simulation:</u>
  - simulates the good circuit and deduces the behavior of all faulty circuits.
  - A fault list L_i is associated with every signal line i
  - L_i is the set of all faults f that cause the values of i in N and N_f to be different at the current simulated time.

- If i is a primary output and all values are binary, then L_i is the set of faults detected at i

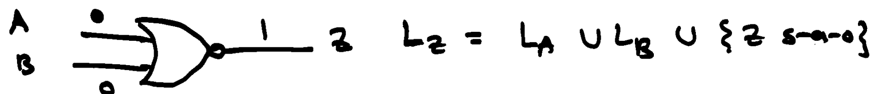
- Given the fault-free values and the fault lists of the inputs of an element, it computes the fault-free output and the output fault list.

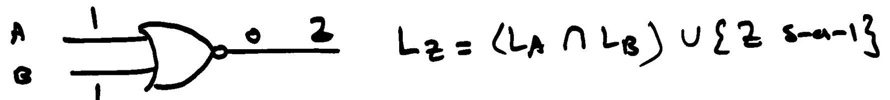
- A deductive fault simulator also propagates list events.





• $\overline{L_B}$ is the set of faults not in L_B





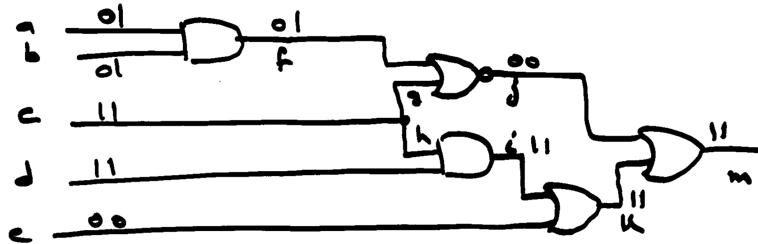
- Let I be the set of inputs of a gate z with controlling value c and inversion i.
- Let C be the set of inputs with value c.
- The fault list of z is computed as:

if $C = \emptyset$ then $L_z = \{\bigcup_{j \in I} L_j\} \cup \{z \text{ s-a-}(c \oplus i)\}$

else $L_z = (\{\bigcap_{j \in C} L_j\} - \{\bigcup_{j \in I-C} L_j\}) \cup \{z \text{ s-a-}(c \oplus i)\}$

- If no input has value c, any fault effect on an input propagates to the output
- If some inputs have value c, only a fault effect that affects all the inputs at c without affecting any of the inputs at $\bar{c}$ propagates to the output.
- The local fault of the output is always added.

- Examples:



- The set of faults to be simulated is
  $\{a_0, a_1, b_1, c_0, c_1, d_1, e_0, g_0, h_0, h_1\}$

- α_v denotes α s-a-v.

- 1st applied vector is (abcde) = 00110

$L_a = \{a_1\}$, $L_b = \{b_1\}$, $L_c = \{c_0\}$, $L_d = \phi$, $L_e = \phi$

$L_f = L_a \cap L_b = \phi$, $L_g = L_c \cup \{g_0\} = \{c_0, g_0\}$

$L_h = L_c \cup \{h_0\} = \{c_0, h_0\}$

$L_j = L_g - L_f = \{c_0, g_0\}$

$L_i = L_d \cup L_h = \{c_0, h_0\}$

$L_K = L_i - L_e = \{c_0, h_0\}$

$L_m = L_K - L_j = \{h_0\}$

- Since h_0 is detected, we drop it by deleting h_0 from every fault list where it appears (L_h, L_i, L_K, and L_m).

- Now assume that the next vector applied is $(abcde) = 11110$

- $L_a = \{a_0\}$, $L_b = \phi$, $L_f = \{a_0\}$

- The evaluation of gate j generates no logic event, but $L_j = L_f \cap L_g = \phi$
  ⇒ A list event may occur even without a corresponding logic event

- $L_m = L_k - L_j = \{c_0\}$
  ⇒ c_0 is detected

- To determine whether a list event has occurred, the new L_α must be compared with the old L_α (L'_α).

- <u>Fault list Computation for a general function:</u>

  - Let z be the output of a combinational block implementing the function $f(a,b,c,--)$.

  - Let us define an XOR operation between a variable x and its fault list L_x by

$$x \oplus L_x = \begin{cases} L_x & \text{if } x=0 \\ \overline{L_x} & \text{if } x=1 \end{cases}$$

- $x \oplus L_x$ is the list of faults that cause x to take value 1.

- Denote by $F(A, B, C, \ldots)$ the set function obtained by replacing all AND and OR operations in $f(a,b,c,\ldots)$ by $\cap$ and $\cup$ respectively.

- The list of faults that cause z to take value 1 (ignoring z s-a-1) is given by:

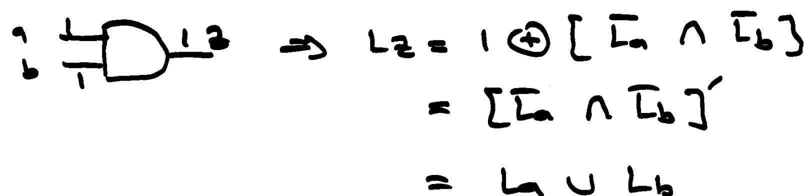
$$z \oplus L_z = F(a \oplus L_a,\ b \oplus L_b,\ c \oplus L_c, \ldots)$$

$$\Rightarrow \boxed{L_z = f(a,b,c,\ldots) \oplus F(a \oplus L_a,\ b \oplus L_b, \ldots)}$$

- **Example:**

$f = ab$

$$L_z = ab \oplus [\, a \oplus L_a \cap b \oplus L_b \,]$$



$$\Rightarrow L_z = 1 \oplus [\, \bar{L}_a \cap \bar{L}_b \,]$$

$$= [\, \bar{L}_a \cap \bar{L}_b \,]'$$

$$= L_a \cup L_b$$

- Example: $F = a \oplus b$

$$L_z = (a \oplus b) \oplus \left[\{ (a \oplus L_a) \cap \overline{(b \oplus L_b)} \} \cup \{ \overline{(a \oplus L_a)} \cap (b \oplus L_b) \} \right]$$

$$L_z = (L_a \cup L_b) - (L_a \cap L_b)$$

$a = 0$, $b = 0$, $z = 0$

$$L_z = (L_a \cap \overline{L_b}) \cup (\overline{L_a} \cap L_b)$$

$$L_z = (L_a \cup L_b) - (L_a \cap L_b)$$

$a = 1$, $b = 1$, $z = 0$

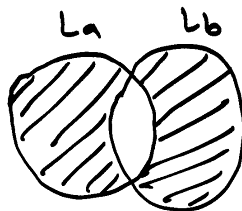
$$L_z = (\overline{L_a} \cap L_b) \cup (L_a \cap \overline{L_b})$$

$$L_z = (L_a \cup L_b) - (L_a \cap L_b)$$

$a = 1$, $b = 0$, $z = 1$

$$L_z = 1 \oplus \left[(\overline{L_a} \cap \overline{L_b}) \cup (L_a \cap L_b) \right]$$

$$= \left[(\overline{L_a} \cap \overline{L_b}) \cup (L_a \cap L_b) \right]'$$



- Oscillation & active faults:
  - Faults that cause oscillation & those that result in large amount of logic activity should be removed from fault lists
  - they can be easily detected
- Three-valued Deductive Simulation:
- Two lists are associated with each line α
  L_α^δ and L_α^ε, where $\{\delta, \varepsilon\} = \{0, 1, u\} - \{v\}$
  and the value of line α is v.
- if $v = 1$ for a line α, then the lists L_α^0 and L_α^u are associated with line α
- L_α^0 (L_α^u) represents the set of faults that cause α to have value 0 (u).
- Since the set of all faults processed is $L_\alpha^0 \cup L_\alpha^1 \cup L_\alpha^u \Rightarrow L_\alpha^v = \overline{L_\alpha^\varepsilon \cup L_\alpha^\delta}$.

$$L_d^0 = (L_a^1 \cap L_b^1 \cap L_c^1) \cup L_d^0$$

(NAND gate: inputs a = 0, b = u, c = 1; output d = 1)

$$L_d^u = (\overline{L_a^0} \cap \overline{L_b^0} \cap \overline{L_c^0}) - (L_a^1 \cap L_b^1 \cap L_c^1)$$

- If line α has a fault-free value $v \in \{0,1\}$, and f is a fault in L_α, and α is a primary output, then f is called a potentially detected fault.

- <u>Deductive fault Simulation Limitations:</u>

  - Propagation of fault lists through an element is based on the Boolean equations
    ⇒ applicable only to models using Boolean equations

  - Limited in practice to two or three logic values

  - Cannot take full advantage of activity-directed simulation

Critical Path Tracing

- For every input vector, it simulates fault-free circuit and determines detected faults by finding critical lines.

- **Definition:** A line l has a critical value v in the test vector t iff t detects the fault l s-a-$\bar{v}$.

- A line with a critical value in t is said to be critical in t.

- **Definition:** A gate input is sensitive (in a test t) if complementing its value changes the value of the gate output.

- The sensitive inputs of a gate with two or more inputs are easily determined as follows:

  1. If only one input j has the controlling value of the gate (c), then j is sensitive.

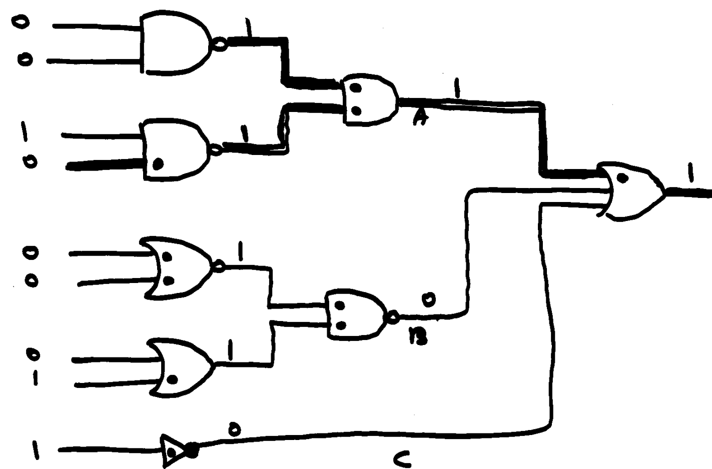
  2. If all inputs have value $\bar{c}$, then all inputs are sensitive.

  3. Otherwise, no input is sensitive.

Lemma: If a gate output is critical, then its sensitive inputs are also critical.

- Critical path tracing starts by marking the primary output as critical.
- Other critical lines are identified by recursive application of the lemma.

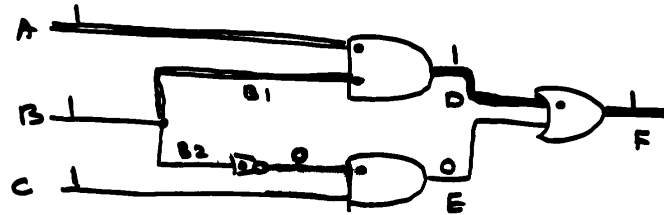
- Example:



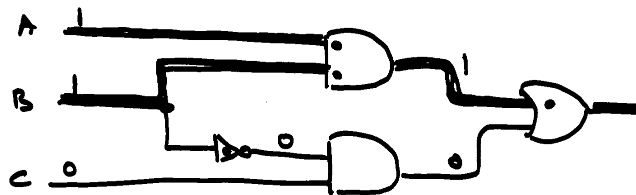
* sensitive inputs are marked by dots and critical paths shown as heavy lines

- Observe that CPT completely avoided areas bordered by B & C since they are not critical.
- Conventional fault simulators would propagate fault effects to B & C before discovering at the output that they are not detected.

- Example:



The fault B s-a-0 is not critical here



The fault B s-a-0 is critical here

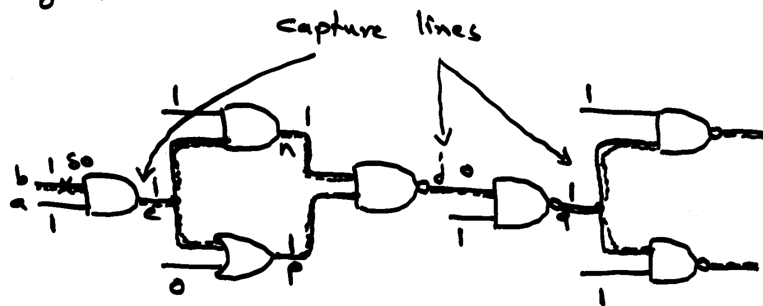
⇒ We need to determine whether a stem X is critical given that some of its fanout branches are critical

- One obvious solution is to use a regular fault simulator to simulate the fault on the stem. If it is detected, then the stem fault is marked critical.

- **Definition**: Let t be a test that activates fault f in a single-output combinational circuit. Let y be a line with level l_y, sensitized to f by t. If every path sensitized to f either goes through y or does not reach any line with level greater than l_y, then y is said to be a capture line of f in t.

- If t detects f, there exists at least one capture line of f, namely the primary output.

- If the effect of f propagates on a single path, then every line on that path is a capture line of f.



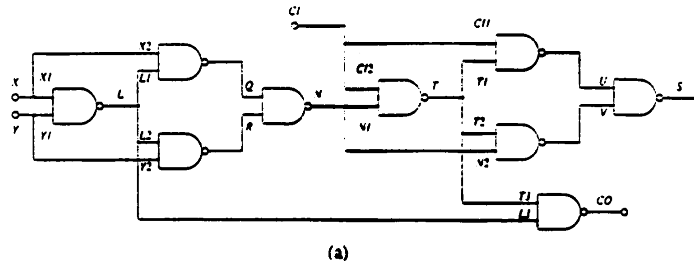
- Capture lines of a fault form a transitive chain
- Lines e, j, and q are capture lines of the fault b s-a-0.
- Lines j and q are capture lines of the fault e s-a-0.
- Line q is a capture line of the fault j s-a-1.

Theorem: A test t detects the fault f iff all the capture lines of f in t are critical in t.

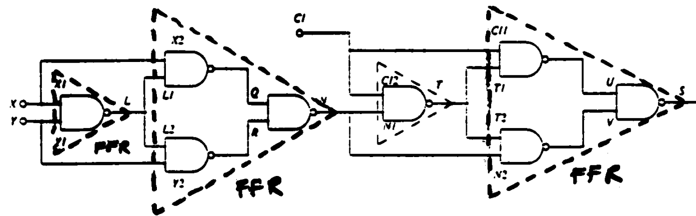
- To determine whether a stem is critical, it is sufficient to propagate the fault effects only to the first capture line of the stem fault.
- Since capture lines are defined for single-output circuits, a circuit with m primary outputs is partitioned into m single-output circuits called cones.
- A cone contains all the logic feeding one primary output.
- Critical path tracing is very simple for fanout-free regions (FFRs).
- Logic simulation and marking of sensitive gate inputs is performed before critical path tracing.

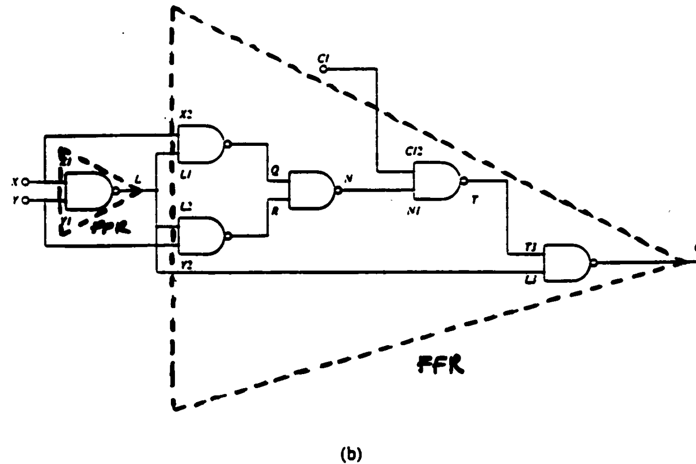
Circuit Partitioning into Cones

- Full adder circuit:



(a)





(b)

Outline of Critical Path Tracing

- Critical path tracing algorithm (CPT)

```
for every primary output z
  begin
    Stems_to_check = ∅
    Extend(z)
    while (Stems_to_check ≠ ∅)
      begin
        j = the highest level stem in Stems_to_check
        remove j from Stems_to_check
        if Critical(j) then Extend(j)
      end
  end
```

- Critical path tracing in a fanout-free region

```
Extend(i)
begin
  mark i as critical
  if i is a fanout branch then
    add stem(i) to Stems_to_check
  else
    for every input j of i
      if sensitive(j) then Extend(j)
end
```

- *Extend* stops at FFR inputs and collects all the stems reached in the set Stems_to_check.

- Stem analysis

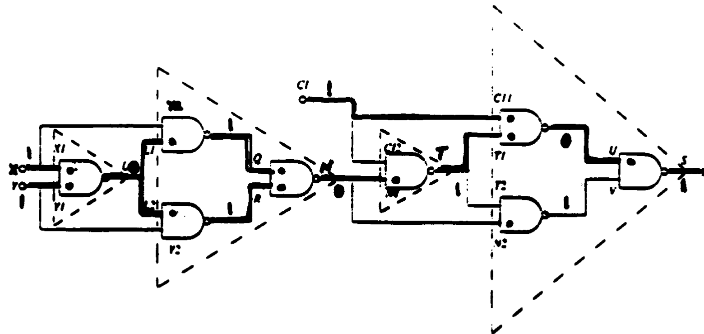
```
Critical(j)
begin
  Frontier = {fanouts of j}
  repeat
    begin
      i = lowest level gate in Frontier
      remove i from Frontier
      if (Frontier ≠ Ø) then
        begin
          if Propagates(i) then add fanouts of i to Frontier
        end
      else
        begin
          if Propagates(i) and i is critical then return TRUE
          return FALSE
        end
    end
end
```

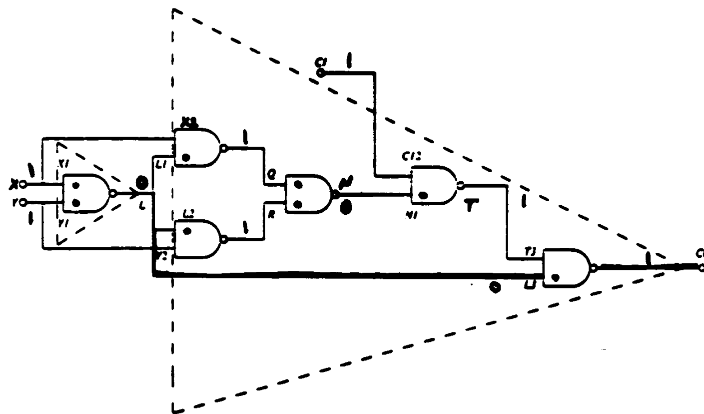
- The function Propagates(i) determines whether gate i propagates the fault effects reaching its inputs

- Lemma: A gate i propagates fault effects iff:

1. Either fault effects arrive only on sensitive inputs of i, OR
2. Fault effects arrive on all the nonsensitive inputs of i with controlling value and only on these inputs.

- Critical paths in the full-adder circuit for the test 111.



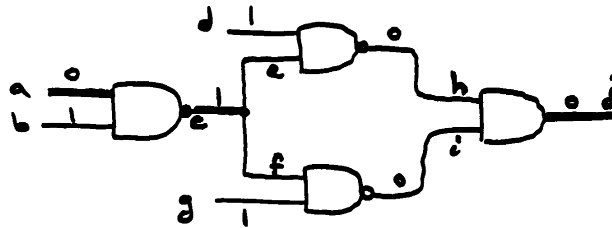


- Execution trace of critical path tracing

FFR traced	Critical lines	Stems_to_check	Stem checked	Capture line
S	S, U, CI1, T1	T, CI		
		CI	T	U or S
T	T, N1	CI, N		
		CI	N	U or S
N	N, Q, R, L1, L2	CI, L		
		CI	L	N
L	L, X1, Y1	CI, X, Y		
		X, Y	CI	U
CI	CI	X, Y		
		Y	X	R or N
X	X	Y		
		Ø	Y	Q or N
Y	Y	Ø		
CO	CO, L3	L		
		Ø	L	—

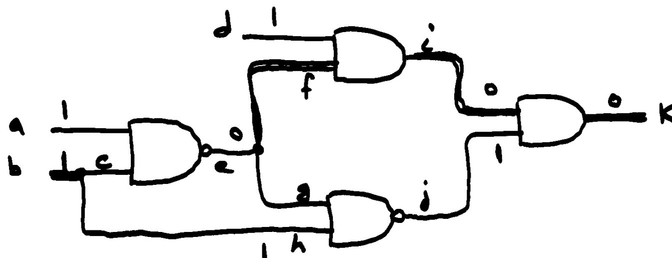
- Critical path tracing may not identify all the faults detected by a test.

- Example 1



Approximations of CPT due to multiple path sensitization.

- Example 2



Approximations of CPT due to partial self-masking.

Distinctive Features of CPT

1. It directly identifies the faults detected by a test, without simulating the set of all possible faults.

2. It deals with faults only implicitly. Fault enumeration, fault collapsing, fault partitioning (for multipass simulation), fault insertion, and fault dropping are not needed.

3. It is based on a path tracing algorithm that does not require computing values in faulty circuits by gate evaluations or fault list processing.

4. It is an approximate method; pessimistic.

5. Critical path tracing is faster & requires less memory than conventional fault simulation.

* Note that the drop in the number of detected faults marked by CPT compared to exact fault simulation is small because:
    - A fault is often detected by several test vectors in a test set.
    - If a fault is not correctly recognized as detected in one test, it is likely that it will be detected in other tests.

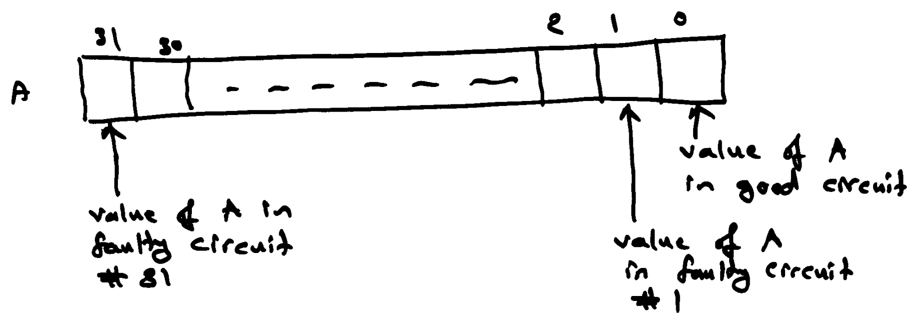
Parallel Fault Simulation

- The good circuit and a fixed number, say W, of faulty circuits are simultaneously simulated.

- A set of F faults requires $\lceil F/W \rceil$ fault simulation passes.

- The values of a signal in the good circuit and the W faulty circuits are represented in a computer word.

- Example: Assume 2-valued logic and a 32-bit word ⇒ W = 31



⇒ 31 faults will be simultaneously simulated with the fault-free machine in one simulation pass

- In this example, each signal in the circuit will need 32-bit word during fault simulation

- Let f represent the fault j s-a-c.
- Let v_i be the value propagating onto line i in the faulty circuit N_f.

- Every line $i \neq j$ takes the value v_i but the value of j should always be c.
- To compute the new values on lines resulting from injecting fault f on line j:

$$v_i' = v_i \overline{\delta_{ij}} + \delta_{ij} c$$

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

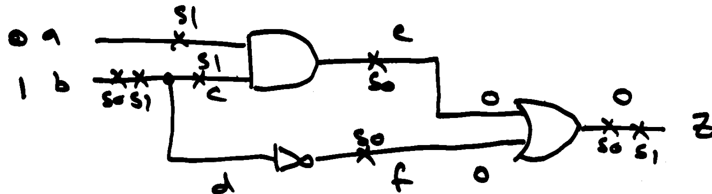
- Each line has two masks I and S
- Mask I indicates whether faults should be inserted on that line & in what bit position.
- Mask S defines the stuck values of the faults on the line
- To insert faults on line Z

$$Z' = Z \cdot \overline{I_Z} + I_Z \cdot S$$

Parallel Fault Simulation

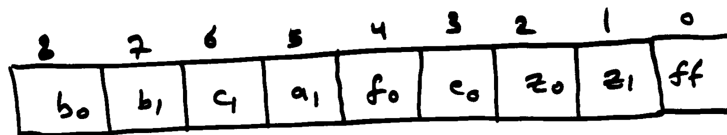
Example

- Consider the circuit shown below and the faults indicated to be fault simulated



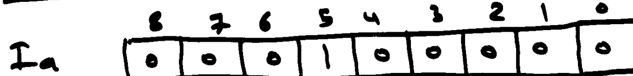
- we have 8 faults, so we need to use 9-bits to simultaneously simulate all of them.

- Fault representation:



- For line a:

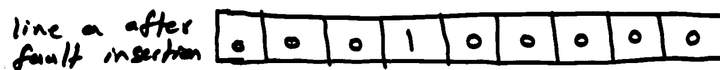
I_a



S_a

line a before fault insertion

line a after fault insertion



- **For line b:**

	8	7	6	5	4	3	2	1	0
I_b	1	1	0	0	0	0	0	0	0
S_b	0	1							
line b before fault insertion	1	1	1	1	1	1	1	1	1
line b after fault insertion	0	1	1	1	1	1	1	1	1

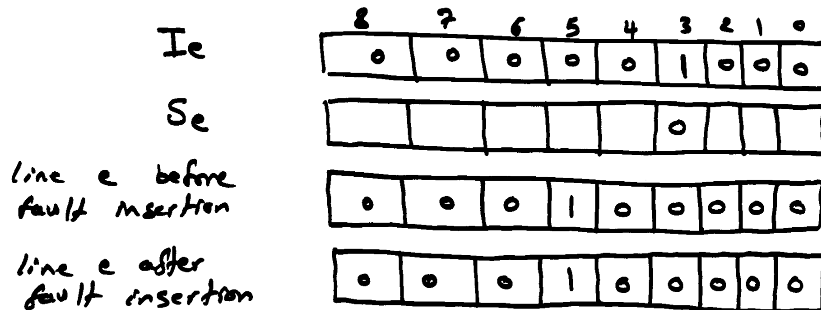
- **For line c:**

	8	7	6	5	4	3	2	1	0
I_c	0	0	1	0	0	0	0	0	0
S_c			1						
line c before fault insertion	0	1	1	1	1	1	1	1	1
line c after fault insertion	0	1	1	1	1	1	1	1	1

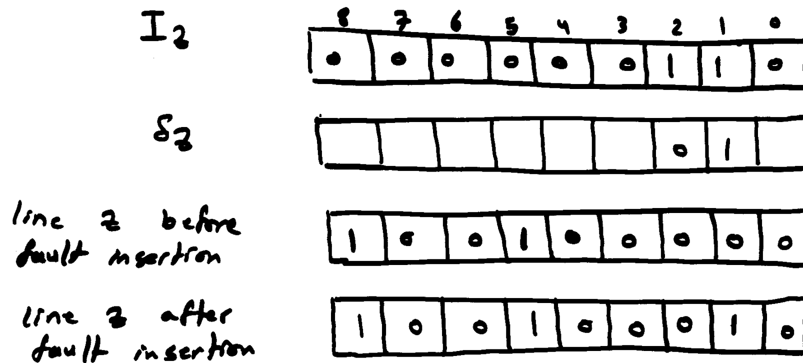
- **For line f:**

	8	7	6	5	4	3	2	1	0
I_f	0	0	0	0	1	0	0	0	0
S_f					0				
line f before fault insertion	1	0	0	0	0	0	0	0	0
line f after fault insertion	1	0	0	0	0	0	0	0	0

- For line e:



- For line z:



- Since z is a primary output, all values in the faulty machine that are different from the fault-free machine indicate fault detection
  ⇒ detected faults: z_1, a_1, b_0

- Let S_i denote the set of lines in the circuit that can be affected by the value of line i.

- Faults defined on lines i and j such that $S_i \cap S_j = \phi$ are said to be <u>independent</u>.

- Independent faults cannot affect the same part of the circuit & can be simulated in the same bit position

- It is possible to reduce the number of passes by simulating several independent faults simultaneously

- Limitations of parallel fault simulation:

  - requires elements to be modeled by Boolean equations
  - impractical for multivalued logic
  - cannot take advantage of the reduction in the number of faults caused by fault dropping

Parallel-Pattern Single-Fault Propagation (PPSFP)

- The PPSFP method combines single-fault propagation with prallel-pattern evaluation.
- Single-fault propagation:
  - Serial fault simulation method for combinational circuits
  - SSFs are simulated one at a time using a fault-free simulator
  - Computation of faulty values starts at the fault site & continues until all faulty values become identical to good values or fault detected
- Parallel-pattern evaluation:
  - Simulates W vectors concurrently
  - For 2-valued logic, the values of a signal in W vectors are stored in W-bit memory
  - Valid for combinational circuits where vector order is not relevant
  - Simulator cannot be event-driven
  - all the gates in the circuit are evaluated in the order of their level

- Let $a < 1$ denote the average activity in a circuit i.e. average fraction of gates that have events on their inputs in one vector

  ⇒ parallel-pattern evaluation will simulate $1/a$ more gates than an event-driven simulator

- Parallel-pattern evaluation is more efficient if $W > 1/a$

- The overall speedup $= W a$

- Example:

  if $a = 0.05$ and $W = 32$ ⇒ overall

  speedup $= 0.05 \times 32 = 1.6$

- Commercial tools use $W = 256$

- PPSOF successfully used to evaluate large sets of random vectors (½ million)

- Cannot support an algorithmic test generation process in which we need to know faults detected by one vector before we generate the next.

Fault Sampling

- Let M be the number of SSFs in a circuit and K be the number of faults detected by a test set T.
- Fault coverage of T is $F = \frac{K}{M}$
- Fault sampling is a technique that reduces the cost of fault simulation by simulating only a random sample of $m < M$ faults.
- Let k by be the number of faults detected by T when simulating m faults.
- Estimated fault coverage $f = \frac{k}{m}$
- The problem is to determine m such that the estimated fault coverage f is bounded by $[F - e_{max}, F + e_{max}]$ with a probability c.
- The probability that T will detect k faults from a random sample of size m, given that it detects K faults from the entire set M is

$$P_k(m, M, K) = \frac{\binom{K}{k}\binom{M-K}{m-k}}{\binom{M}{m}}$$ (Hypergeometric distrib.)

- The mean $\mu_x = m \frac{K}{M} = mF$
- The variance $\sigma_x^2 = m \frac{K}{M}\left[1 - \frac{K}{M}\right]\frac{M-m}{M-1}$

$$\simeq mF(1-F)(1-\frac{m}{M})$$

- For large M, this distribution can be approximated by a normal distribution with
  - mean $\mu_f = \frac{\mu_x}{m} = F$
  - variance $\sigma_f^2 = \frac{\sigma_x^2}{m^2} = \frac{1}{m} F(1-F)(1-\frac{m}{M})$
- With confidence level of 99.7 %, the estimated fault coverage is in the interval

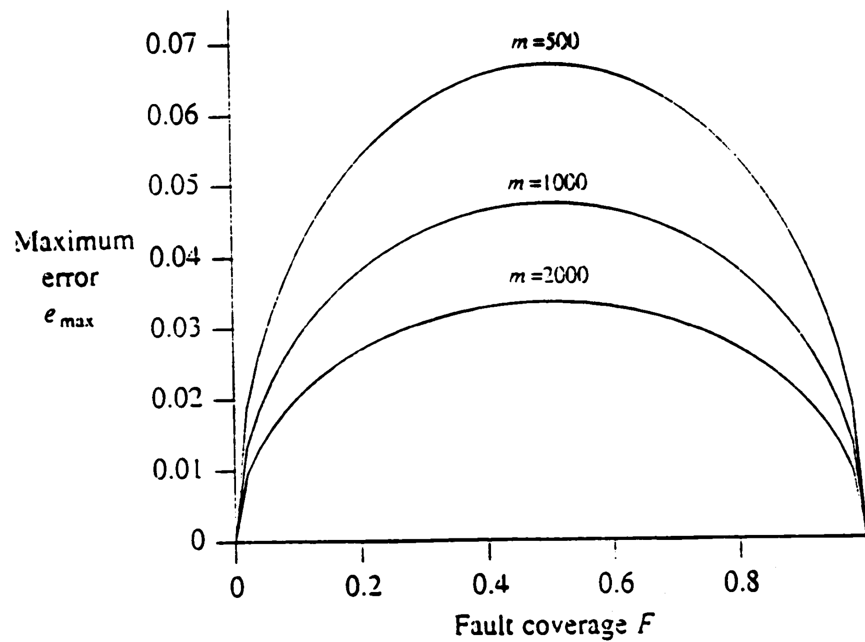
$$[F - 3\sigma_f , F + 3\sigma_f]$$

- The maximum estimation error emax is

$$e_{max} = 3\sqrt{F(1-F)(1-\frac{m}{M}) \times \frac{1}{m}}$$

- To reduce significantly the cost of fault simulation, m should be much smaller than M
- With $m \ll M$, we can approximate $(1-\frac{m}{M}) \simeq 1$ and the error becomes independent of the total number of faults M.

- Estimated fault coverage error vs. actual fault coverage



- Note that the worst case occurs when the fault coverage F is 50%.
- With a sample size of 1000, the estimation error is less than 5%.

Statistical Fault Analysis

- STAFAN is a statistical fault analysis method which provides low cost alternative to exact fault simulation
- STAFAN estimates for every SSF its probability of being detected by a test T based on logic simulation.
- Let N be a combinational circuit and T be a set of n <u>independent</u> random vectors.
- Let d_f be the probability that a random vector of T detects the fault f.
- The probability of not detecting f with n vectors is $(1 - d_f)^n$
- The probability d_f^n that a set of n vectors detects f is $d_f^n = 1 - (1 - d_f)^n$
- Let Φ be the set of faults.
- The expected number of faults detected by n vectors is $D_n = \sum_{f \in \Phi} d_f^n$

- The expected fault coverage is $\frac{D_n}{|\Phi|}$
  = average detection probability

- Let f be the s-a-v fault on line 2.

- C1(2), the 1-controllability of 2, is the probability that a randomly selected vector of T sets line 2 to value 1.

- C0(2), the 0-controllability of 2, is the probability that a randomly selected vector of T sets line 2 to value 0.

- O(2), the observability of 2, is the probability that a randomly selected vector of T propagates a fault effect from 2 to a primary output.

- STAFAN assumes that activating a fault and propagating its effects are independent events.

- The detection probability of the s-a-0 fault on line 2 is C1(2) O(2)

- The detection probability of the s-a-1 fault on line 2 is C0(2) O(2)

- To estimate controllabilities & observabilities STAFAN computes the following for every line l during fault-free simulation:

  - 0-count: incremented in every test in which l has a value 0.

  - 1-count: incremented in every test in which l has a value 1.

  - sensitization count: incremented in every test in which l is a sensitive input of the gate (done only for gate inputs).

- $C0(l) = \dfrac{0\text{-count}}{n}$

- $C1(l) = \dfrac{1\text{-count}}{n}$

- The probability $S(l)$ that a randomly selected vector propagates a fault effect from l to the gate output

$$S(l) = \frac{\text{sensitization-count}}{n}$$

- The computation of observabilities starts by setting $O(i) = 1$ for every primary output i.

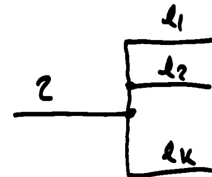
- Observabilities for the other lines are computed via a backward traversal of the circuit

- Let z be an input of a gate with output m.

  $$O(z) = S(z)\, O(m)$$

- Let z be a stem with K fanout branches

- We want to determine $O(z)$ given $O(z_i)$



- <u>Let us assume the following:</u>
  1. Let L_i (L) denote the event whose probability is $O(z_i)$ ($O(z)$)
  2. the events $\{L_i\}$ are independent
  3. the event L occurs iff any subset of $\{L_i\}$ occurs

  $$\Rightarrow L = \bigcup_{i=1}^{K} L_i$$

- Because of reconvergent fanout, L may occur even when none of the L_i events happens, and the occurrence of a subset of $\{L_i\}$ does not guarantee that L will take place.

- STAFAN assume the following lower and upper bounds:

$$\max_{1 \le i \le K} O(z_i) \le O(z) \le P\left(\bigcup_{i=1}^{K} L_i\right)$$

- Example: For $K=2$, $P(L_1 \cup L_2) = O(z_1) + O(z_2) - O(z_1)\,O(z_2)$

- STAFAN computes the observability of a stem z by

$$O(z) = (1-\alpha) \max_{1 \le i \le K} O(z_i) + \alpha\, P\left(\bigcup_{i=1}^{K} L_i\right)$$

$$\alpha \in [0,1]$$

- Based on their detection probabilities, STAFAN groups the faults into:
  1. High range with detection prob. > 0.9
  2. Low range with detection prob. < 0.1
  3. Mid range with $0.1 < d_f < 0.9$

High range ⇒ likely detected (91% - 98% det.)
Low range ⇒ likely not detected (78% - 98% not det.)
Mid range ⇒ No prediction (< 10% of faults)