EE/CE 6301: Advanced Digital Logic

Bill Swartz

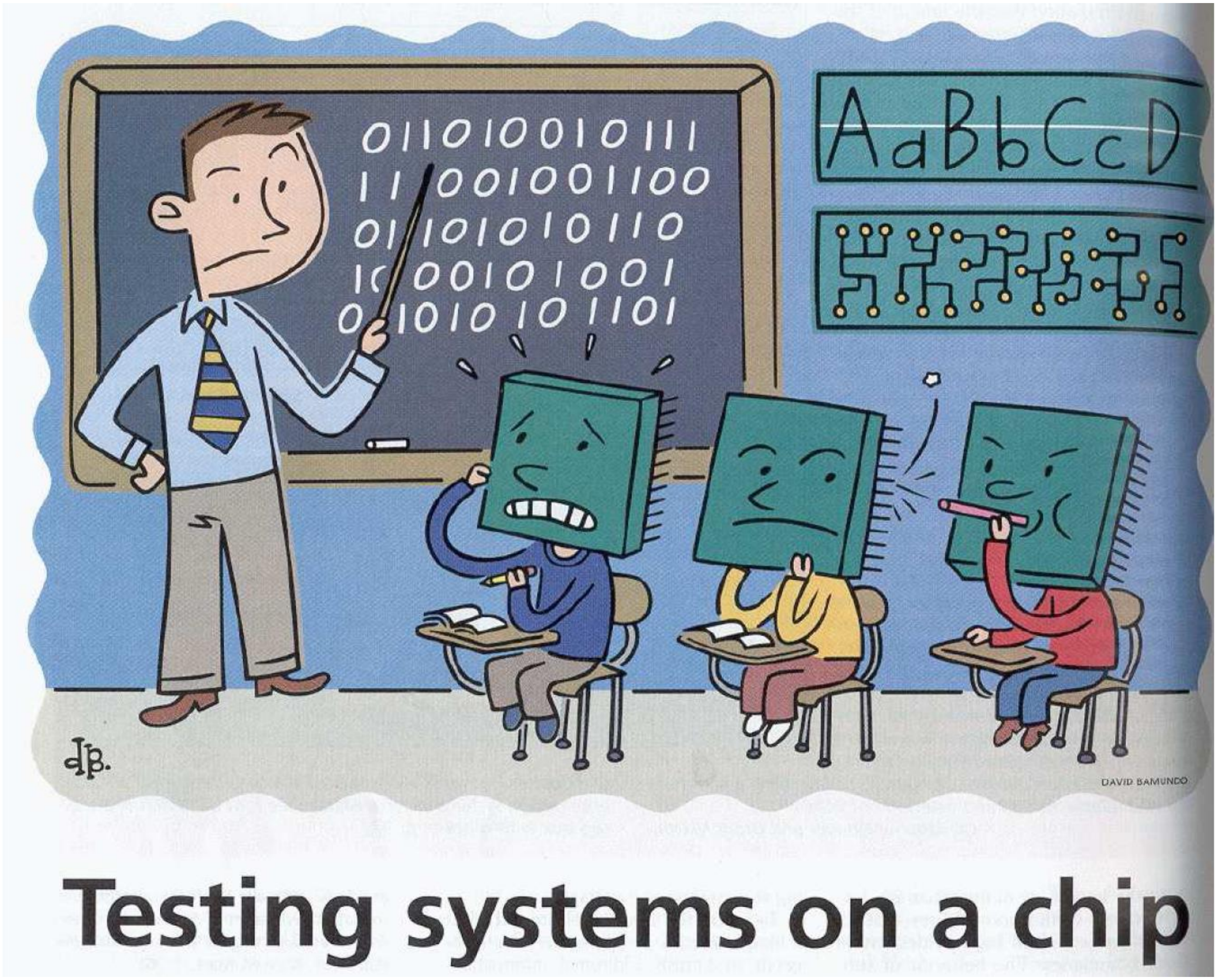
Dept. of EE
Univ. of Texas at Dallas

Session 11

Testing of Digital Systems

VLSI Test Philosophy

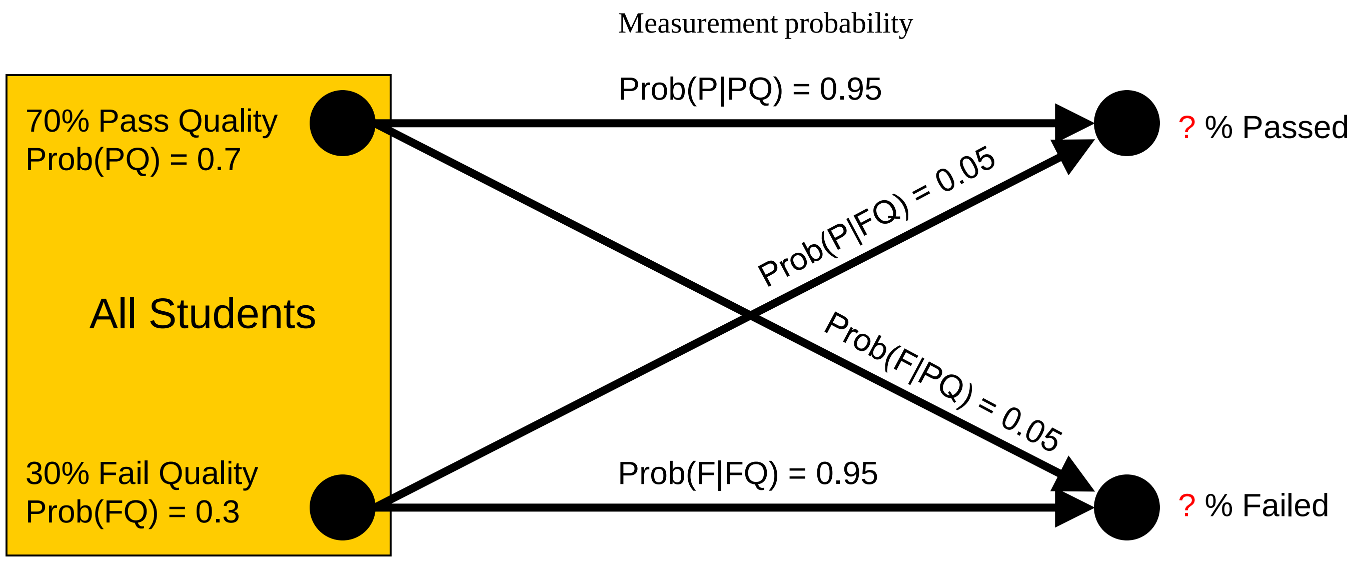
Testing



Source: IEEE Spectrum

Test Philosophy

- A Pass/Fail test:



PQ: student is pass quality
FQ: student is fail quality

P: student passes the test
F: student fails the test

Test Philosophy (cont'd)

- Probability of passing:

$$\begin{aligned}\text{Prob}(P) &= \text{Prob}(P \mid PQ) * \text{Prob}(PQ) + \text{Prob}(P \mid FQ) * \text{Prob}(FQ) \\ &= 0.95 * 0.70 + 0.05 * 0.30 = 0.68\end{aligned}$$

$$\begin{aligned}\text{Prob}(F) &= \text{Prob}(F \mid FQ) * \text{Prob}(FQ) + \text{Prob}(F \mid PQ) * \text{Prob}(PQ) \\ &= 1 - \text{Prob}(P) = 0.32\end{aligned}$$

- Student risk:

$$\Rightarrow \text{Prob}(PQ \mid F) = \frac{\text{Prob}(F \mid PQ) * \text{Prob}(PQ)}{\text{Prob}(F)} = \frac{0.05 * 0.7}{0.32} = 0.11$$

- 11% of failed students should have passed

$$\text{Prob}(A|B) = \frac{\text{Prob}(B|A) * \text{Prob}(A)}{\text{Prob}(B)} \qquad \text{Bayes Theorem}$$

Test Philosophy (cont'd)

- Student's risk (manufacturer's risk or yield loss):

$$\Rightarrow \text{Prob}(PQ \mid F) = \frac{\text{Prob}(F \mid PQ) * \text{Prob}(PQ)}{\text{Prob}(F)} = \frac{0.05 * 0.7}{0.32} = 0.11$$

- Teacher's risk (consumer's risk):

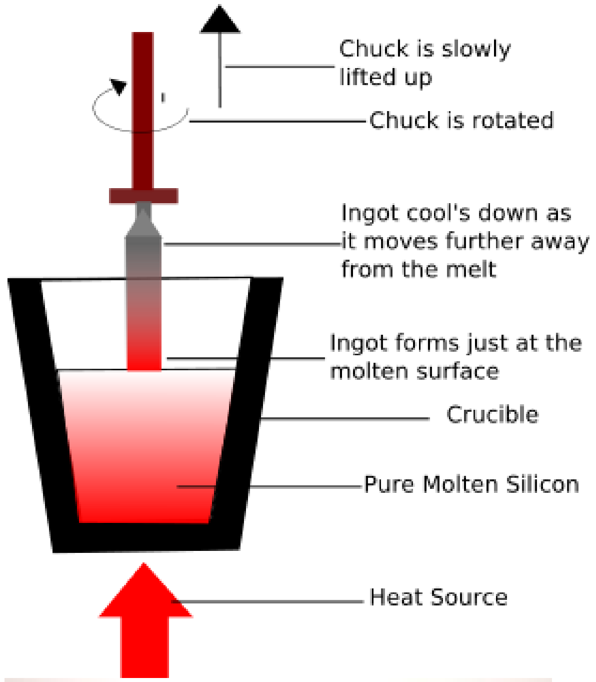
$$\Rightarrow \text{Prob}(FQ \mid P) = \frac{\text{Prob}(P \mid FQ) * \text{Prob}(FQ)}{\text{Prob}(P)} = \frac{0.05 * 0.3}{0.68} = 0.022$$

VLSI Chips – Present and Future

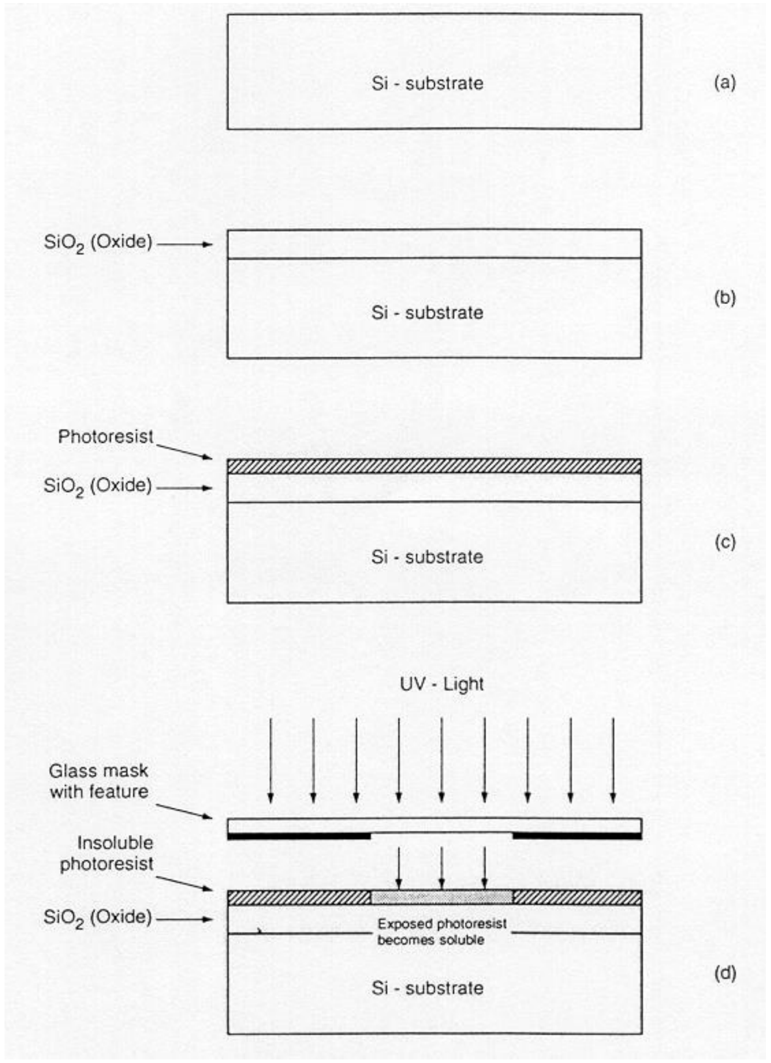
Year	1997-2001	2003-2006	2009-2012
Feature Size, µm	0.25 – 0.15	0.13 – 0.10	0.07 – 0.05
Millions of Transistors/cm^2	4 – 10	18 – 39	84 – 180
Number of wiring layers	6 – 7	7 – 8	8 – 9
Die Size, mm^2	50 – 385	60 – 520	70 – 750
Pin count $\boxed{\propto \sqrt{N_{trans.}}}$	100 – 900	160 – 1475	260 – 2690
Clock Rate, MHz	200 – 730	530 – 1100	840 – 1830
Voltage, V	1.2 – 2.5	0.9 – 1.5	0.5 – 0.9
Power, W	1.2 – 61	2 – 96	2.8 – 109

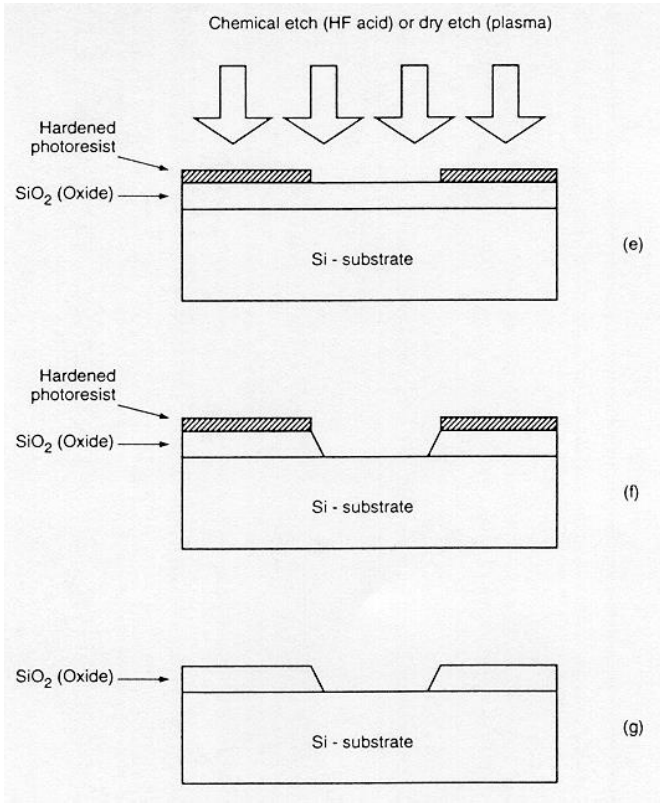
VLSI Fabrication Process

- A seed crystal is dipped into the melted silicon to initiate single-crystal growth
- Diameter: 75 – 230 mm
- Growth rate: 30 – 180 mm/hour
- The output is a silicon ingot
- Using diamond blades, the wafers are produced (thickness = 0.25 – 1 mm)
- At least one surface is polished to a flat, scratch-free mirror finish

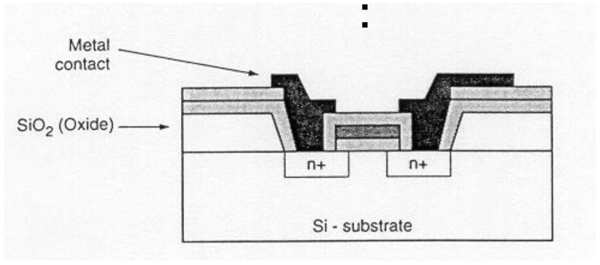

Chuck is slowly lifted up
Chuck is rotated
Ingot cool's down as it moves further away from the melt
Ingot forms just at the molten surface
Crucible
Pure Molten Silicon
Heat Source

VLSI Fabrication Process (cont'd)



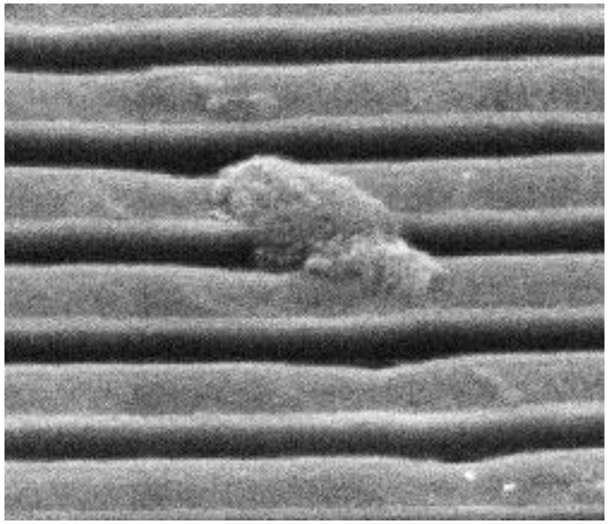


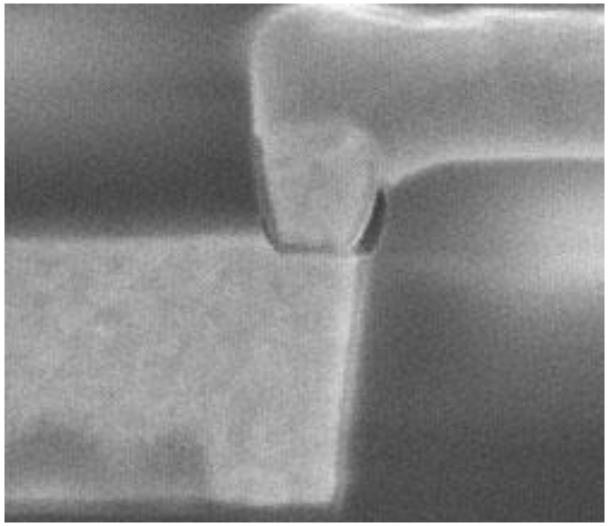
⋮



What May Go Wrong?

- Shorts between two points (bridges)
- Open in a line
- Improper doping
- Masking error
- Improper thickness of a line
- Particles on surface
- Electron migration due to heat
- Corrosion
- …





What Can be Done?

1. Wafer probe testing.
2. Use microscope to find faulty dices. Mark with a small ink. Reject after separation. Package the unmarked dies.
3. Test for electrical and mechanical characteristics.
4. Final inspection for cosmetic defects.
5. Pack and ship.

Test Definition

- "Testing" of a system is an experiment in which the system is exercised and its resulting responses are analyzed to ascertain whether it behaved correctly
  - How to send input data (test patterns)?
  - How to collect output results (test signatures)?
  - How long the system should run?
  - How to make the conclusion?
  - How to locate the cause of misbehavior (diagnosis)?
  - What causes misbehavior?
  - What test equipment are needed?
  - …

Test Importance

- Test is an integral part of any manufacturing process.
  - Production Cost = Fabrication & Packaging + Testing
  - According to Semiconductor Industry Association (SIA) roadmap, the cost of testing a die will surpass its manufacturing cost in the near future.
- Each VLSI chip needs to be tested. Sampling is not adequate.

When and Where to Test

- Try to detect defects at the earliest point possible.
- Costs increases dramatically (typically by an order of magnitude) as faulty components find their way to the higher level of integration
- Rule of 10:
  - —$1 to detect a faulty chip. Throw it out!
  - —$10 to find (and replace) a bad IC on a board.
  - —$100 to find a bad PC board in a system.
  - —$1000 to find a bad component in the fielded system

Taxonomy of Testing

- Failure: incorrect operation of a system.
- Fault: the physical failure mechanism.
- Fault effect: the logical effect of a fault on a signal or carrying net.
- Error:
  - The condition (or state) of a system containing a fault, i.e. deviation from correct state (e.g. wrong component, incorrect wiring…).
  - An instance of an incorrect operation of a system

Taxonomy of Testing (cont'd)

- An error does not necessarily lead to a failure.
  Example:
  - Error: existence of erroneous state of air pressure.
  - Failure: one of the operational tires on your car goes flat.
- Defect: physical problems due to imperfect manufacturing process.
  - Defects are not directly attributable to a human error.
  - Examples: shorts, opens, improper doping profiles, mask misalignment…
- Sometimes fabrication defects and errors are collectively referred to as physical faults

Taxonomy of Testing (cont’d)

- Fault model: the basic assumptions regarding the nature of the logical (malignant) fault.
- A fault is “detected” by observing an error caused by it.

- Briefly:

  Defect → Fault → Error → Failure

  (e.g. dust particle → stuck-at-1 → erroneous logic value → circuit failing)

Taxonomy of Testing (cont'd)

- Yield (Y): the probability (usually expressed as percent) that items (e.g. ICs) will not be defective (fraction of manufactured parts that is defect-free).
  - For VLSI chips yields range from 10% to 90% depending on the process, circuit complexity, lambda (λ), etc.
  - Y=1 (100%) means defect free production.
  - Y=0 (0%) means all circuits are faulty.

Taxonomy of Testing (cont'd)

- Fault coverage (FC) is a measure to grade the quality of a test:
  - FC= $\dfrac{\text{Number of faults detected}}{\text{Total number of faults}} * 100\%$
  - FC=1 (100%) : all possible faults detected
  - FC=0 (0%) : no fault detected.
- Defect level (DL): the probability of shipping a defective product (the fraction of bad parts that pass all tests).
  - An empirical equation: $DL = \left[1 - Y^{(1-FC)}\right] * 100\%$
  - Both Y and FC must be continually improved to maintain reliability of shipped products

Failure Rate
(Fault Occurrence Frequency)

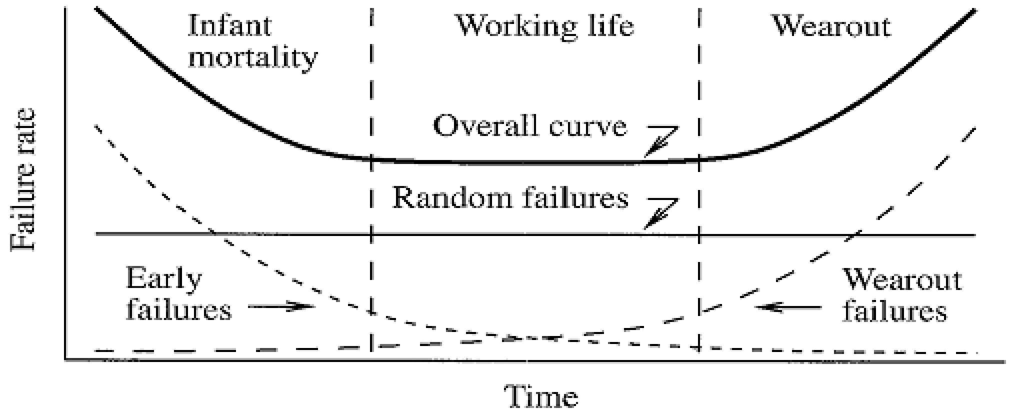
Type of Faults

- **Permanent fault**: always being present after their occurrence (in existence long enough to be observed at test time).
- **Intermittent fault**: existing only during some intervals (appears and disappears at regular intervals).
- **Transient fault**: a one-time occurrence, e.g. caused by a temporary change in some environmental factors (appears and disappears in short time intervals). These faults are crucial for real-time systems.

Failure Rate Over Product Lifetime

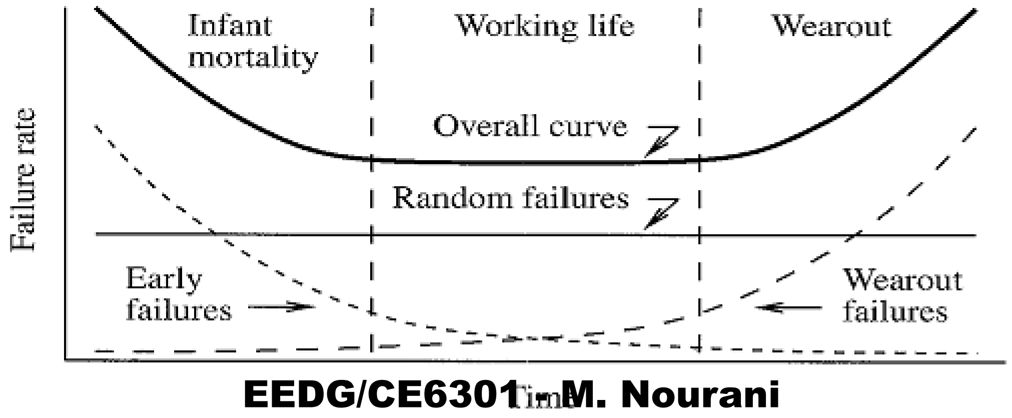
- A well-know graphical representation of the failure rate, is the bathtub curve. It consists of three regions:

1. Infant Mortality
2. Working Life
3. Wearout



Failure Rate Over Product Lifetime (cont'd)

- Infant mortality: failures in this region are attributed to poor quality due to variations in the production process.
- Working life: Constant failure rate λ. Failures are considered to occur randomly in time.
- Wear out: Increasing failure rate. This represents the end-of-life period of a system.



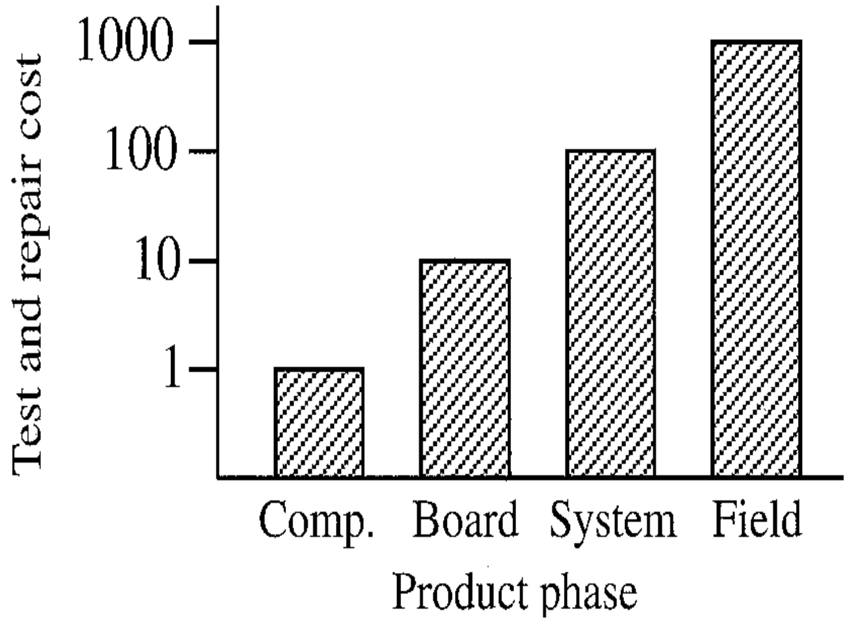
Failure Rate Over Product Lifetime (cont'd)

- It should be clear that a system should be shipped after it has passed the infant mortality period, in order to reduce the number of field returns.
- Shipping a system after the infant mortality period can be done by:
  1. Aging the system for that period (this can be several months)
  2. Aging the system under stress (this accelerates the aging process)

Test Economics

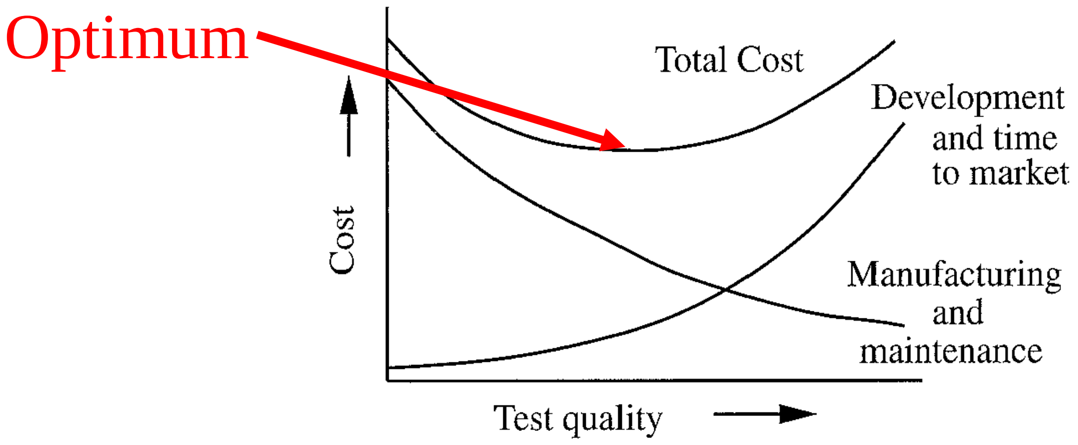
Repair Cost During the Product Phases

- A move from one product phase to the next causes the volume of parts and the test & repair cost to increase by a factor of *10.* *This is the rule-of-ten*



Economics and Liability of Testing

- *Good tests*
  - reduce test & repair cost (see rule-of-ten)
  - can reduce development time & time-to market
  - can reduce field maintenance costs
  - reduce personal injury and law suits
- There is an optimum in test development cost and its contribution to profit. Too many tests require a long test development time and test cost



Main Fault Models

Goals of Fault Modeling

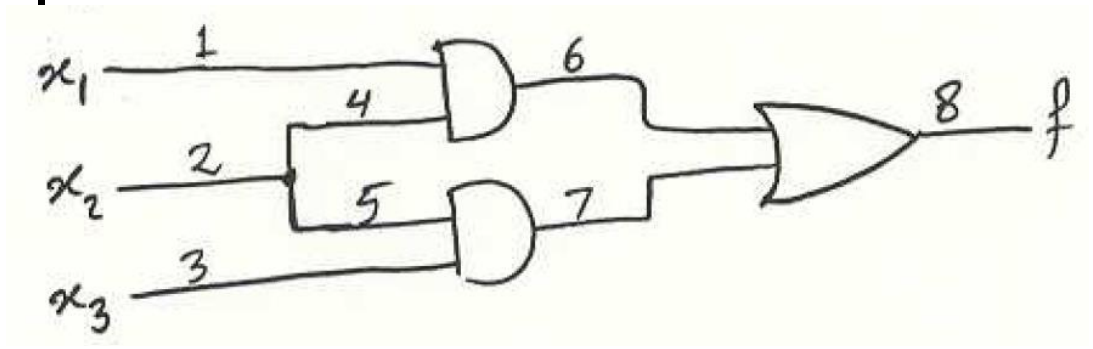
- To model physical defects in the circuit at high level of abstraction.
- To allow test generation and fault/coverage analysis to be done early in the design process.
- To model high percentage of the actual physical defects that can occur in components:
  - To reduce number of individual defects that have to be considered (e.g. find equivalent or dominant faults).
  - To reduce the complexity of the component/circuit description for test generation/analysis.

Stuck-at Fault Model

- Most commonly used fault model.
- May consider “single” or “multiple” stuck-at faults.
- The components are assumed to be (internally) fault-free.
- The effect of faults is modeled by having a line segment tied to either:
  - Vcc (s-a-1)
  - GND (s-a-0)

Stuck-at Fault Model (cont'd)

- Example:



- Fault-free function: $f = x_1x_2 + x_2x_3$
- Faulty functions:
  - Fault 1/0 → $f^* = x_2x_3$
  - Fault 2/1 → $f^* = x_1 + x_3$
  - Fault 6/0 → $f^* = x_2x_3$

Line Segment
Or
Wire

Stuck-at Value

Features of Single s-a-f Model

- Can be applied at the logic/RTL/system levels.
- Reasonable and manageable number of faults (i.e. less or equal 2 * # of line segments ≈ 2 * # of circuit nodes). So, is computationally feasible to deal with.
- Well-developed algorithms exist for automatic test patter generation (ATPG)
- Other useful fault models like stuck-open, bridging, etc. can be applied into (sequence of) stuck-at faults

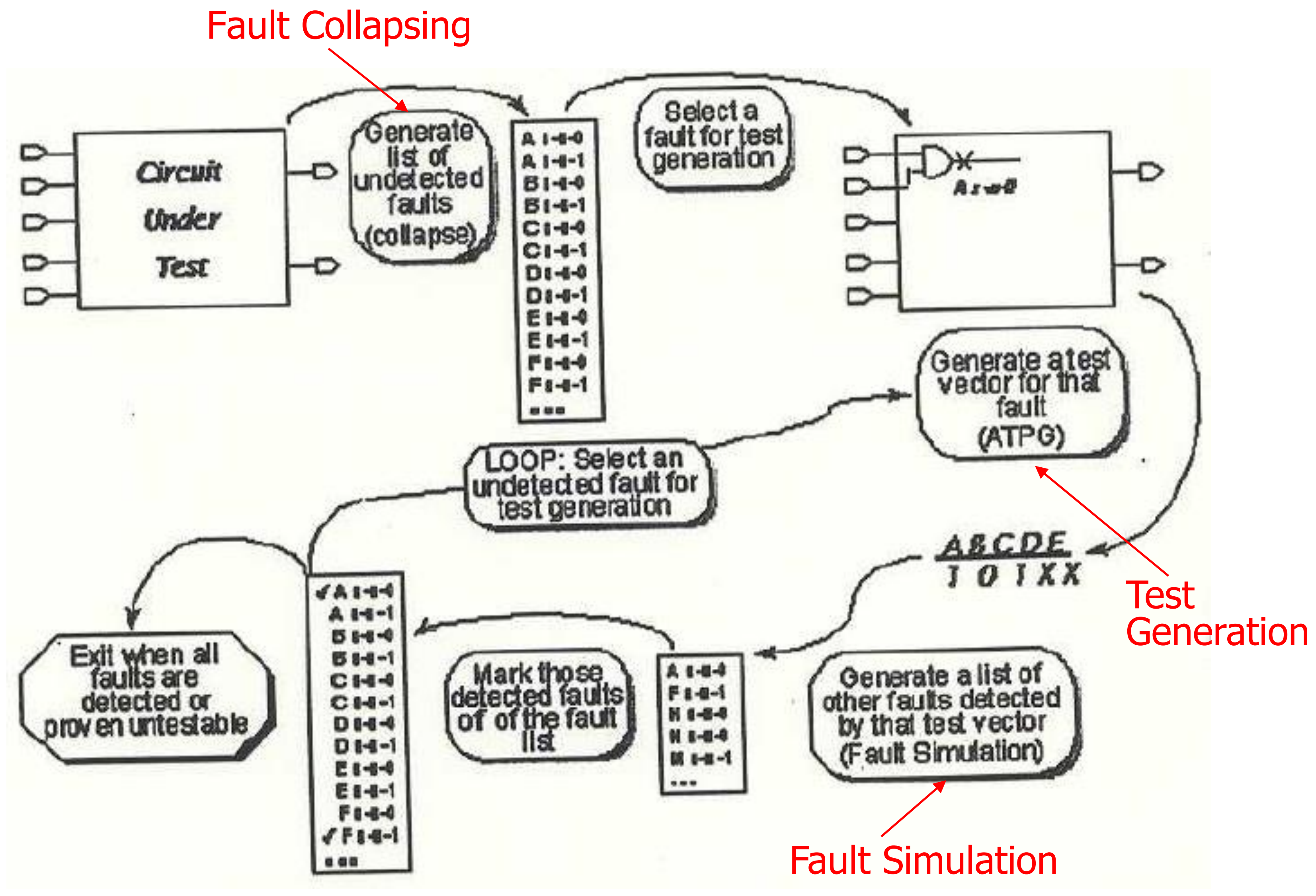
Features of Single s-a-f Model (cont’d)

- Empirical evidence shows that single stuck-at fault model covers the majority (about 90%) of the possible manufacturing defects in circuits such as:
  - Source-drain shorts
  - Oxide pinholes
  - Diffusion contaminants
  - Metallization shorts
  - …

Other Fault Models

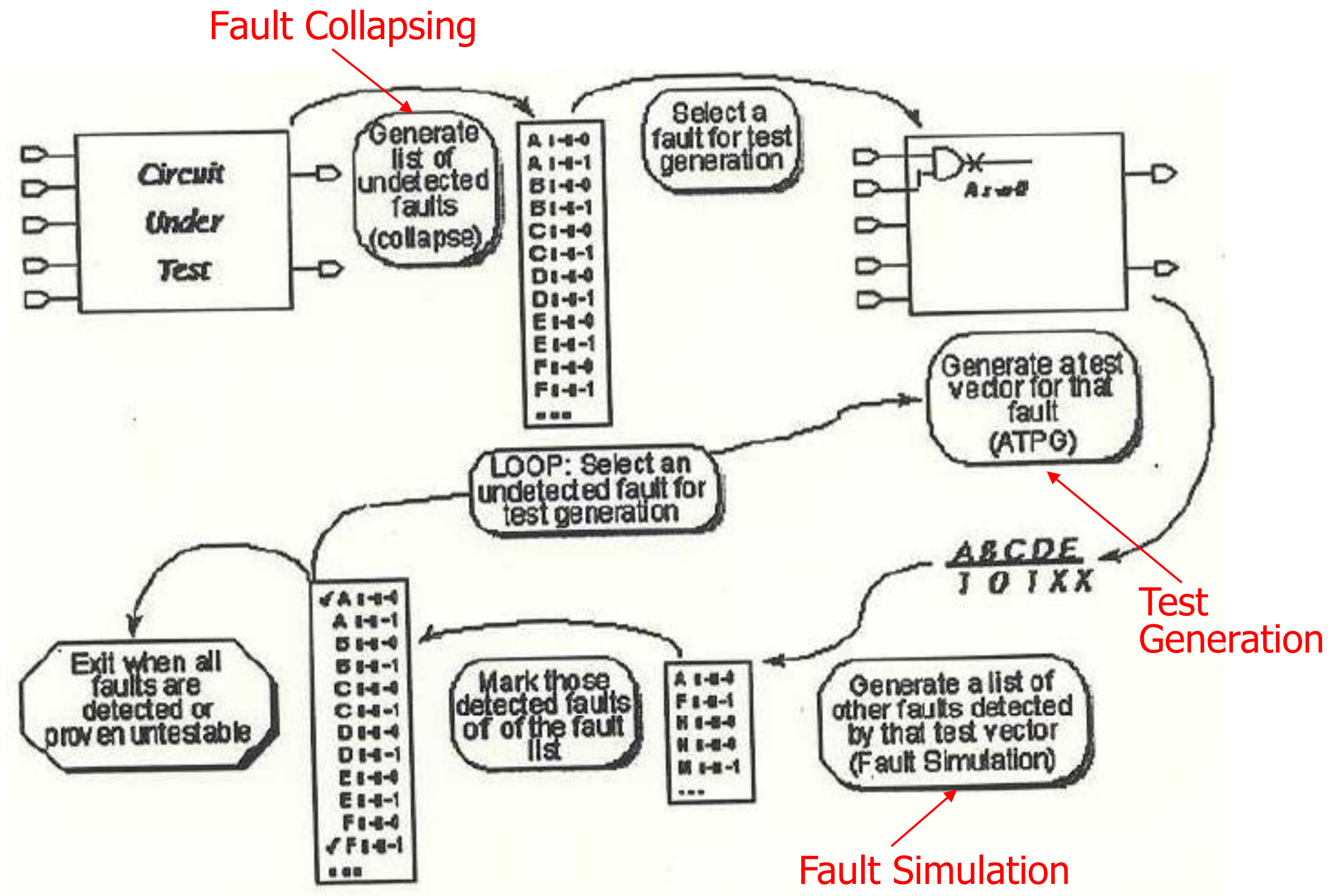
- Stuck-open fault
  - —Assumption: a single physical line in the circuit is broken (usually internal to gates).
- Bridging fault
  - —two (or more) lines are shorted together. The functional effect (on the logic nodes) is of a wired logic AND or OR depending on the technology
- Delay fault
  - —Assumptions: The logic function of the circuit-under-test is error-free. Some physical defects (developed during fabrication) makes some delays in the circuit greater than some predefined bounds.

Fault Analysis System


Fault Collapsing
Circuit Under Test
Generate list of undetected faults (collapse)
A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
F s-a-1
...
Select a fault for test generation
A s-a-0
Generate a test vector for that fault (ATPG)
LOOP: Select an undetected fault for test generation
ABCDE
1 0 1 X X
Test Generation
Exit when all faults are detected or proven untestable
✓A s-a-0
✓F s-a-1
Mark those detected faults of of the fault list
A s-a-0
F s-a-1
H s-a-0
M s-a-1
Generate a list of other faults detected by that test vector (Fault Simulation)
Fault Simulation

Fault Analysis System

Fault Analysis System


Fault Collapsing
Circuit Under Test
Generate list of undetected faults (collapse)
A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
F s-a-1
...
Select a fault for test generation
A s-a-0
Generate a test vector for that fault (ATPG)
Test Generation
LOOP: Select an undetected fault for test generation
ABCDE
1 0 1 X X
Exit when all faults are detected or proven untestable
✓A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
✓F s-a-1
...
Mark those detected faults off of the fault list
A s-a-0
F s-a-1
H s-a-0
H s-a-0
M s-a-1
...
Generate a list of other faults detected by that test vector (Fault Simulation)
Fault Simulation

Fault Collapsing

Definitions

- Let Z(x) be the logic function of a combinatorial logic circuit where x is a specific input vector *x*
- Let Z_f(x) be the logic function of a combinatorial logic circuit with the fault *f* present where x is a specific input vector *x*

Definitions

- Circuit is tested by applying a sequence T of test vectors $\{t_1, t_2, \ldots t_m\}$ and

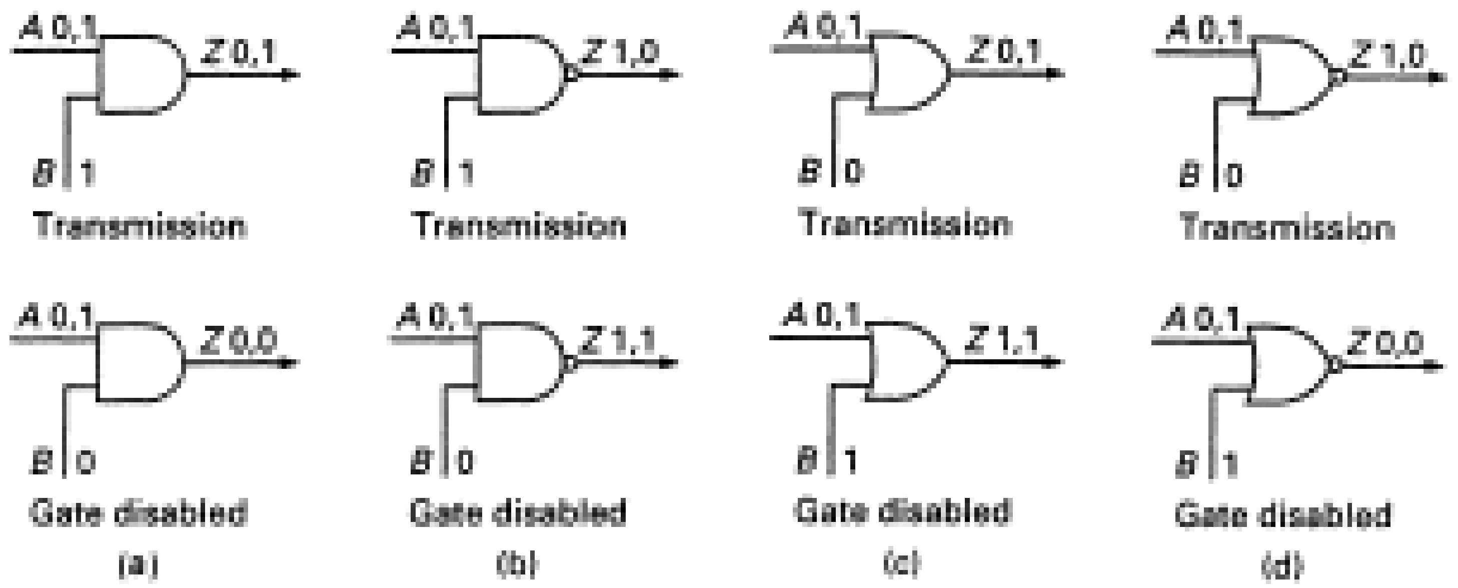
  Comparing output response with expected output response $Z(t_1), Z(t_2), \ldots Z(t_m)$.

- A test vector t detects a fault *f* iff

  $$Z_f(t) \neq Z(t)$$

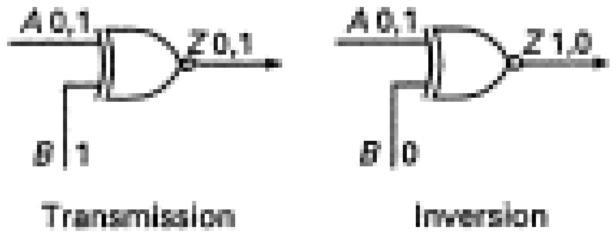
Switching Characteristics

- AND, NAND, OR, NOR gates



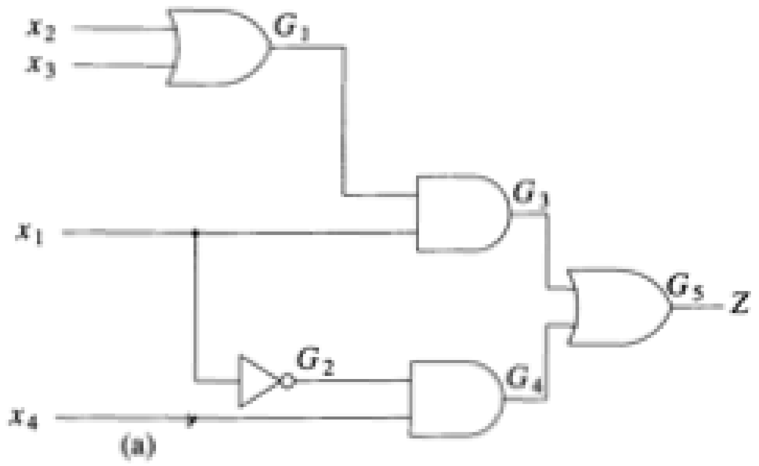
Switching Characteristics

- XOR, XNOR gates



Example Circuit

- Z = (x2+x3) x1 + x1′x4

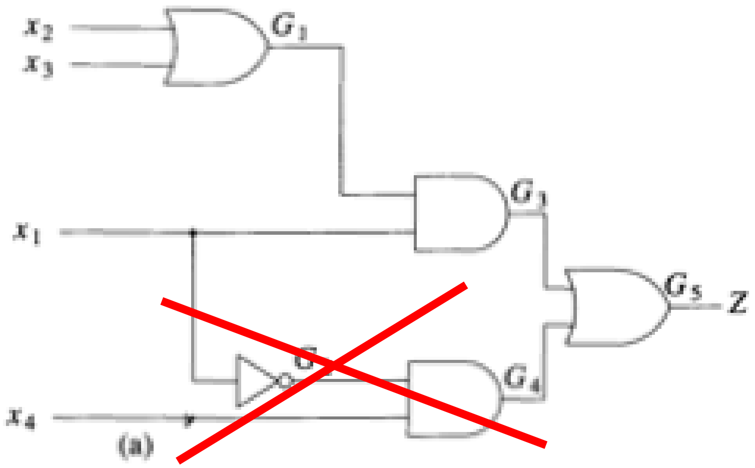


Example Circuit : f = x4 s-a-0

- $Z = (x2+x3)\, x1 + x1'x4$

  becomes

- $Z_f = (x2+x3)\, x1$



Fault Detection

- For single-output circuits, a test that detects fault *f* makes

  $Z(t) = 0$ and $Z_f(t) = 1$

  or

  $Z(t) = 1$ and $Z_f(t) = 0$

- Thus the set of all tests that detect f is given by solutions to

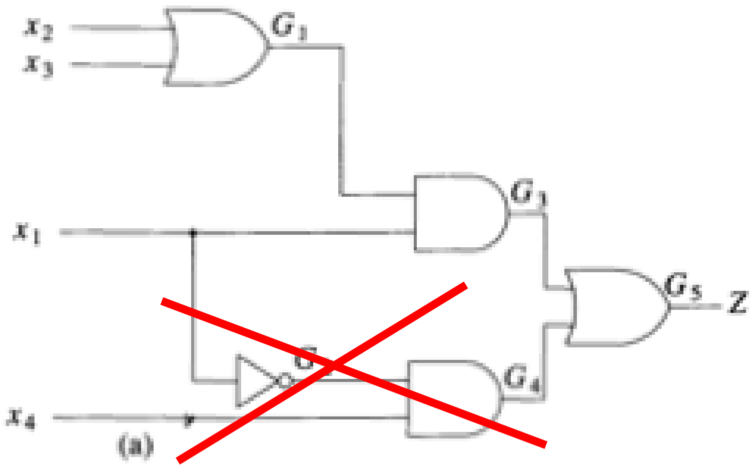
  $$Z(x) \bigoplus Z_f(x) = 1$$

Example Circuit : f = x4 s-a-0

- $Z = (x2+x3)\ x1 + x1'x4$

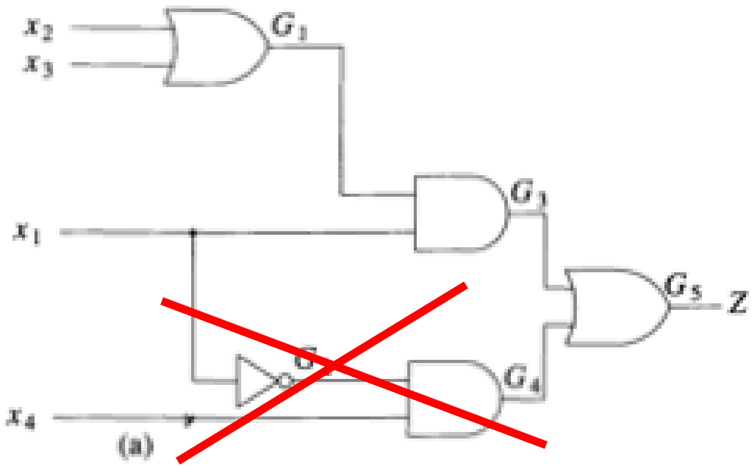
  becomes

- $Z_f = (x2+x3)\ x1$



Example Circuit : f = x4 s-a-0

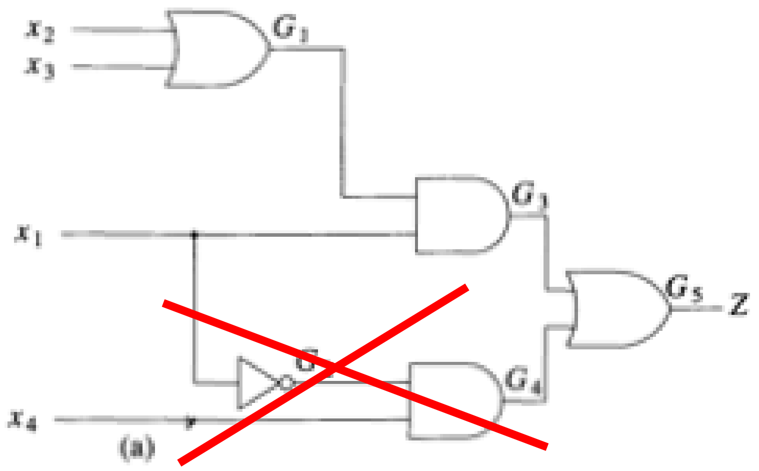
- $Z(x) \oplus Z_f(x) = 1$
- (x2+x3) x1 + x1′x4 $\oplus$ (x2+x3) x1 = 1
- => x1′x4



Example Circuit : f = x4 s-a-0

- $Z(x) \oplus Z_f(x) = 1$
- $(x2+x3)\, x1 + x1'x4 \oplus (x2+x3)\, x1 = 1$
- $\Rightarrow\ x1'x4$

Means vectors (x4,x3,x2,x1)
1000
1100
1010
1110
all detect f

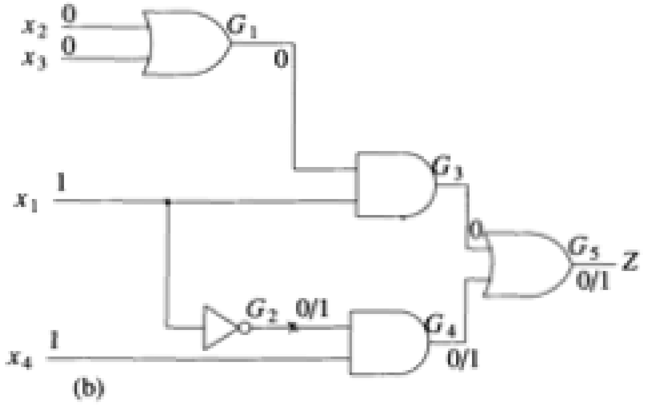


Fault Sensitization

- A test t that detects a fault f activates or generates an error (or fault effect) by creating different Boolean values at site of fault.
- A test t propagates the error to primary output along a path known as **fault propagation**
- A line whose value in the test *t* changes the presence of the fault *f* is said to be **sensitized** to the fault *f* by test *t*.
- The resulting path is said to be the **sensitized path**.

f = G2 s-a-1

- Sensitized path: (G2, G4, G5)



Fault Equivalence

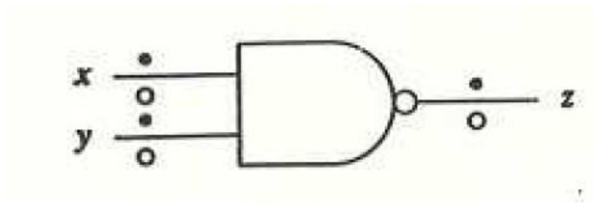
- Two faults f and g are functionally equivalent iff $z_f(t) = z_g(t)$ under any test set T ($t \in T$).
- Test (vector) t distinguishes between two faults f and g if $z_f(t) \neq z_g(t)$ (i.e. $z_f(t) \oplus z_g(t) = 1$ for a single-output function).
- Equivalency and distinguishability are defined for the faults of the same nature (fault-universe).

Stuck-at-fault Notation

- We will denote stuck-at-faults using dots as a shorthand
- A s-a-1 fault is a black dot •
- A s-a-0 fault is a open circle ○

Fault Equivalence Example

- NAND-2 gate:
  - s-a-1: •
  - s-a-0: ○



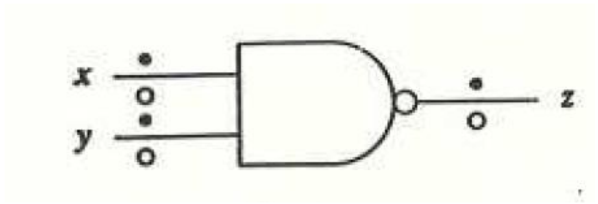
The uncollapsed state has 6 stuck-at-faults

  - For the NAND-2 gate:

    The s-a-0 @ x is equivalent to the s-a-0 @ y which is also equivalent to the s-a-1 @ z
  - Test pattern xy=11 can detect all of them.

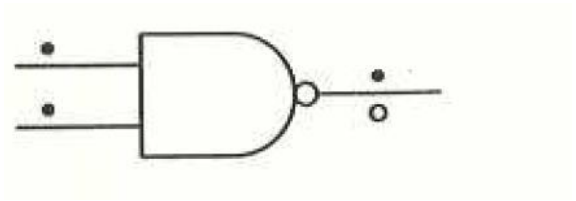
Fault Equivalence Example

- NAND-2 gate:
  - s-a-1: •
  - s-a-0: ○



The uncollapsed state has 6 stuck-at-faults

  - We can show the collapsed state where we remove equivalent faults (remove input faults) as



The collapsed state has 4 stuck-at-faults

Controlling and Non-controlling Value

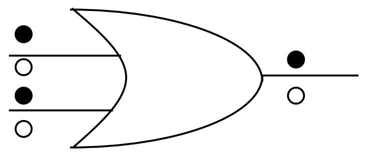
- Controlling value : when it present on at least one input of a gate, it forces the output to a known value
  - AND gate, NAND gate : 0
  - OR gate, NOR gate : 1
- Non-controlling value : the complement of the controlling value or (sensitizing value)
  - AND gate, NAND gate : 1
  - OR gate, NOR gate : 0

Fault Equivalence

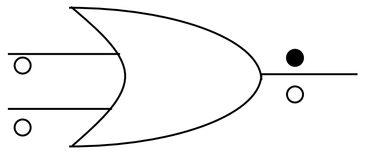
- In general, for a gate with controlling value *c* and inversion *i*, all input s-a-*c* faults and the output s-a-($c \oplus i$) fault are functionally equivalent.

- For NAND gate, controlling value *c* is 0 and inversion is 1, so input stuck-at-zero faults are equivalent to the output stuck-at-one fault

Another Example: Or Gate



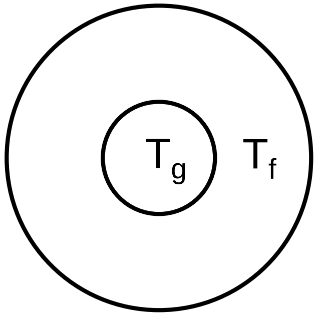
- For an OR gate, controlling value c is 1 and inversion is 0, so all input s-a-1 is equivalent to output s-a-1
- Test pattern 00 will find all of these faults



Equivalent collapsed faults

Fault Dominance

- Let T_f and T_g be the set of all tests that detect fault f and g, respectively. These are equivalent statements:
  - Fault f dominates g ($g \subseteq f$)
  - f and g are functionally equivalent under T_g
  - $T_g \subseteq T_f$
  - Any test that detects g, i.e. $z_g(t) \neq z(t)$ will also detect f on the same primary output because $z_f(t) = z_g(t)$



- To reduce efforts in testing we can identify and remove the dominating faults (such as f) from the set of faults.

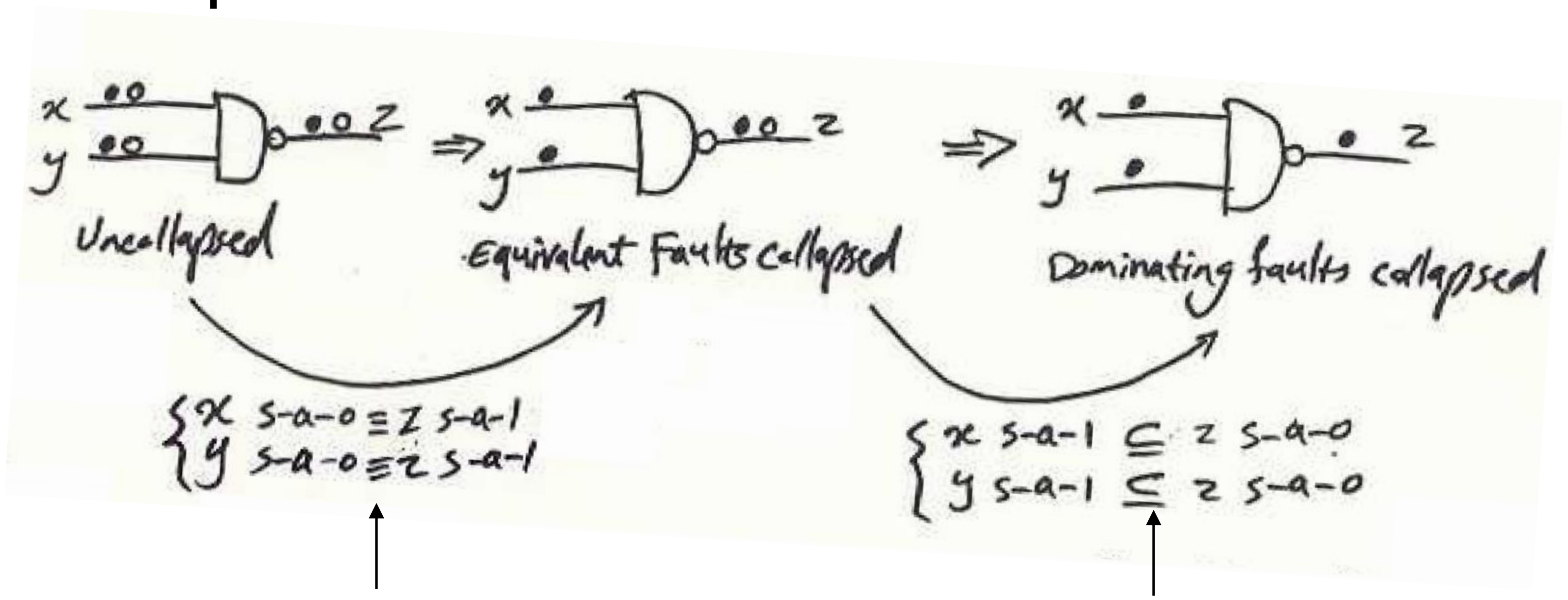
Fault Dominance

- In general, for a gate with controlling value *c* and inversion *i*, the output s-a-($\bar{c} \oplus i$) fault dominates any s-a-$\bar{c}$ input fault equivalent.

- For NAND gate, controlling value *c* is 0 and inversion is 1, so the output s-a-0 fault dominates the s-a-1 input faults

Fault Dominance (cont'd)

- Example:



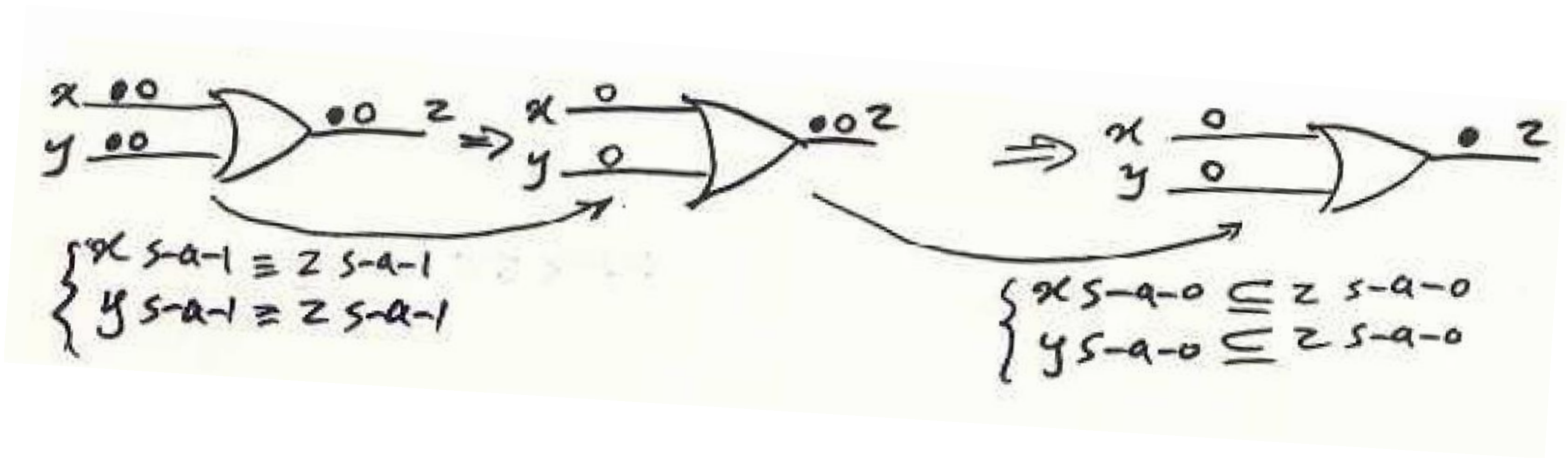
equivalence

dominance

Fault Dominance

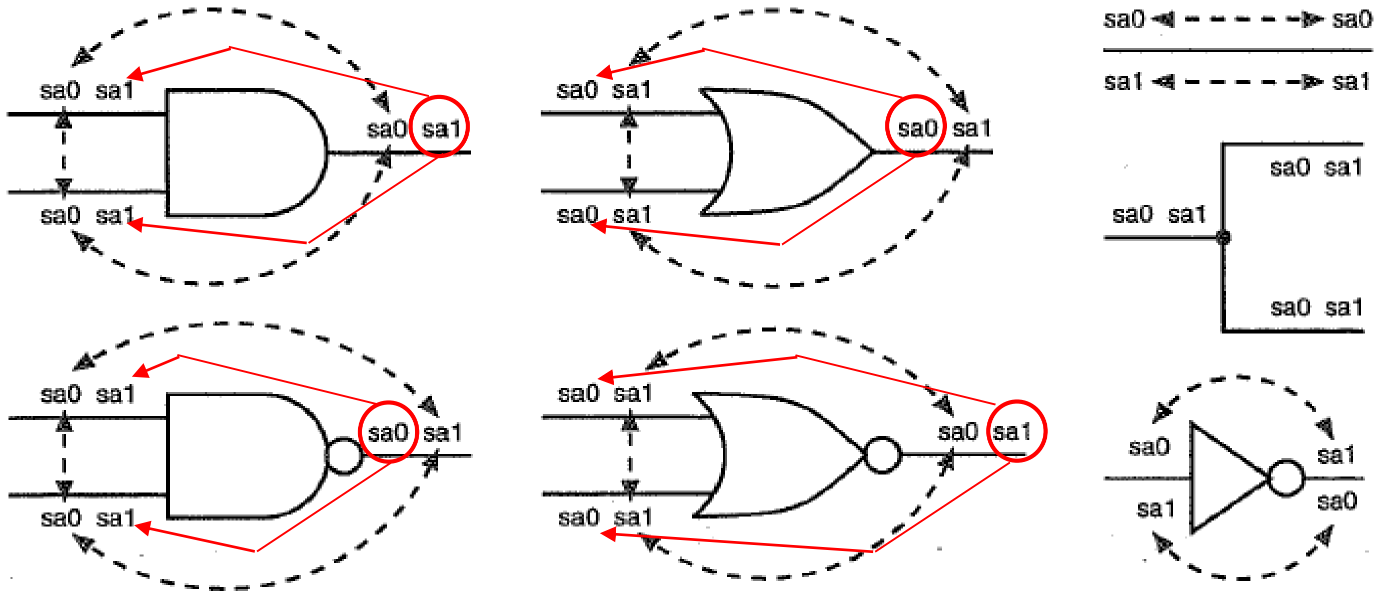
- In general, for a gate with controlling value *c* and inversion *i*, the output s-a-($\bar{c}$⊕*i*) fault dominates any s-a-$\bar{c}$ input fault equivalent.

- For OR gate, controlling value *c* is 1 and inversion is 0, so the output s-a-0 fault dominates the s-a-0 input faults



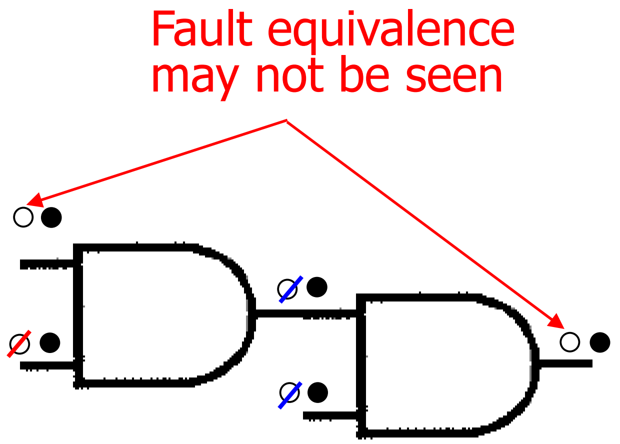
Summary of Equivalent/Dominant Faults

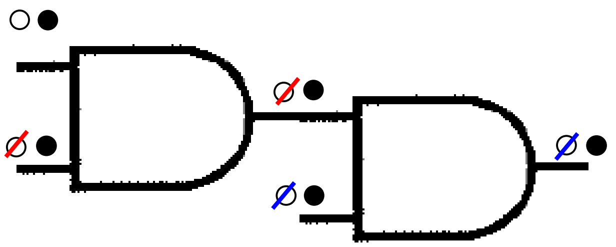
- Equivalence – dashed lines
- Dominance – red lines



Caution in Fault Collapsing

- Try to work on faults in one direction. For example, start from the output gate, apply both equivalency and dominancy to the gate and and remove as many faults as possible in the output (keep the rest in the input) before going to the next level.
- If you work on both directions when removing fault, some equivalency/dominancy may not be seen.
- Example (consider only fault equivalence):

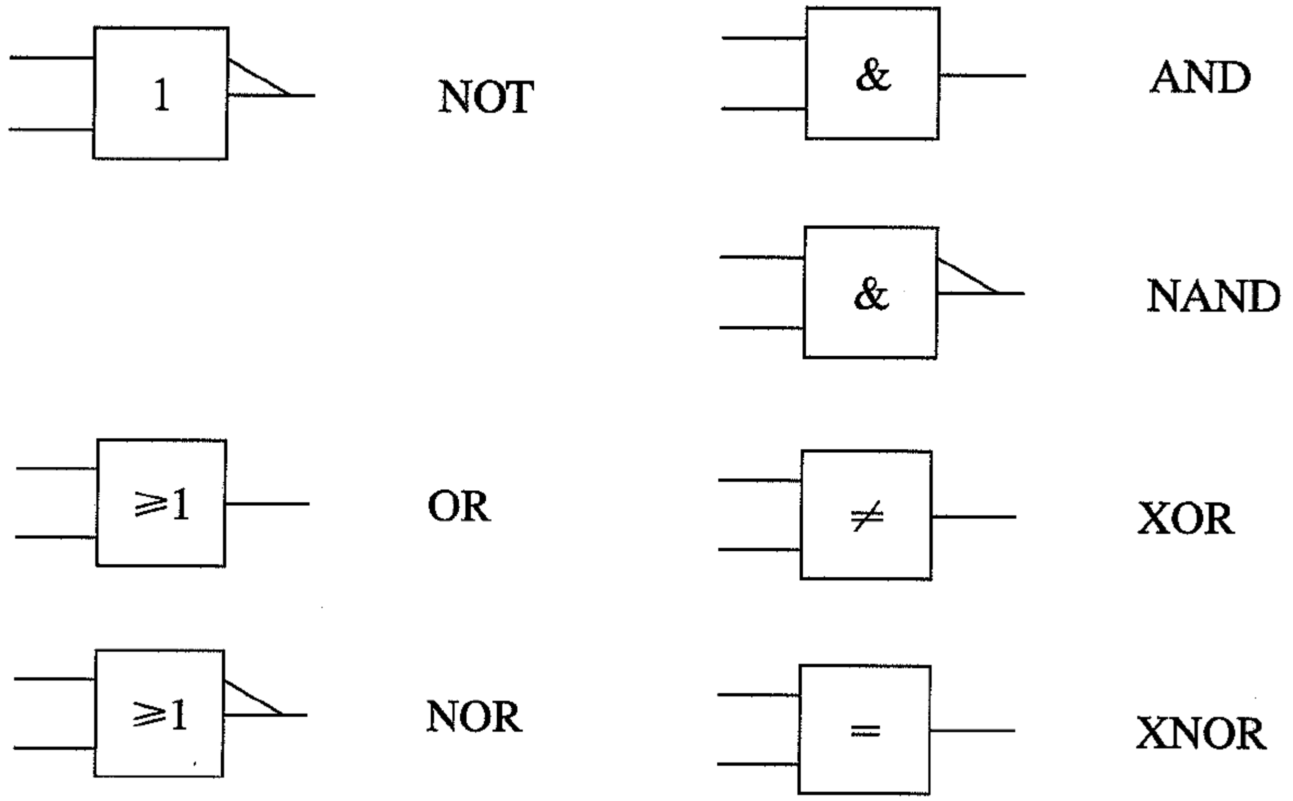




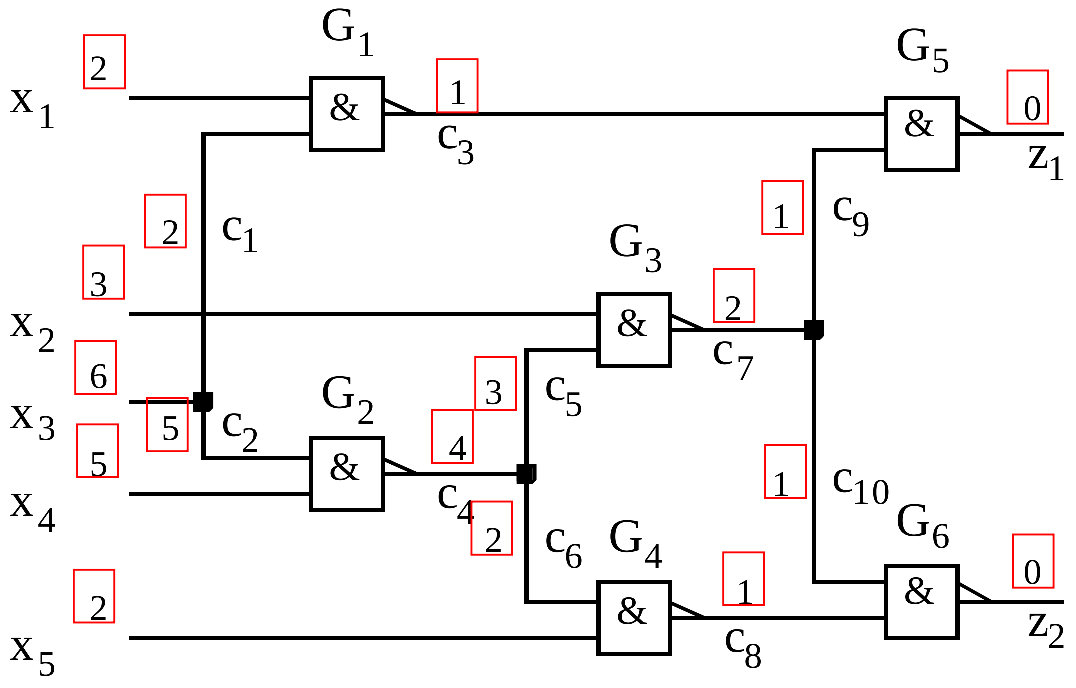
Step 1: /
Step 2: /

IEEE Gate Symbols

- In some of the examples, the standard IEEE symbols of the logic gates will be used

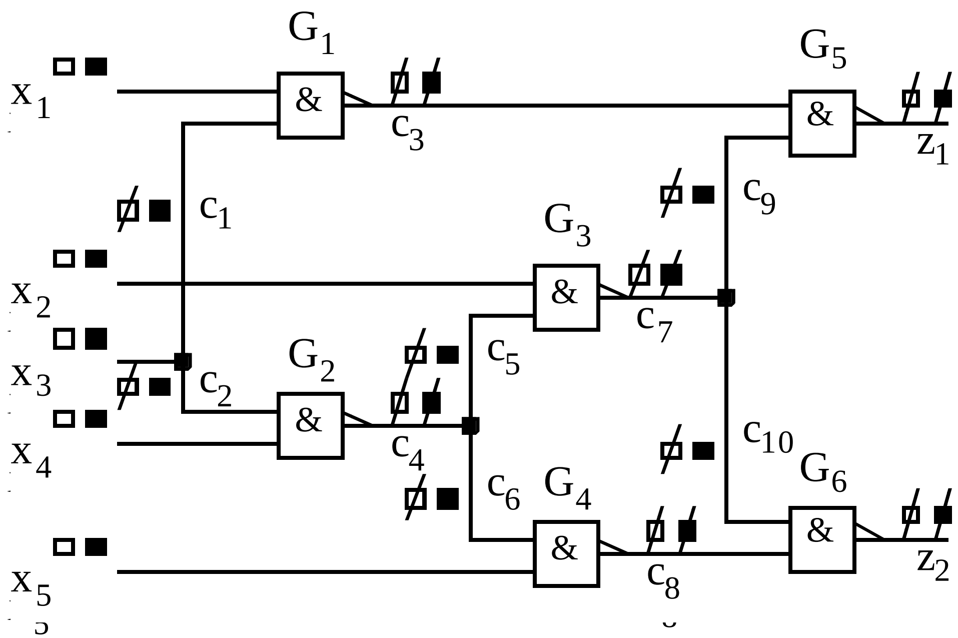


Fault Collapsing (Dropping) – An Example



- Circuit lines ordered in non-decreasing order of their output level values as $\{z_2, z_1, c_{10}, c_9, c_8, c_7, c_6, c_5, c_4, c_3, c_2, c_1, x_5, x_4, x_3, x_2, x_1\}$

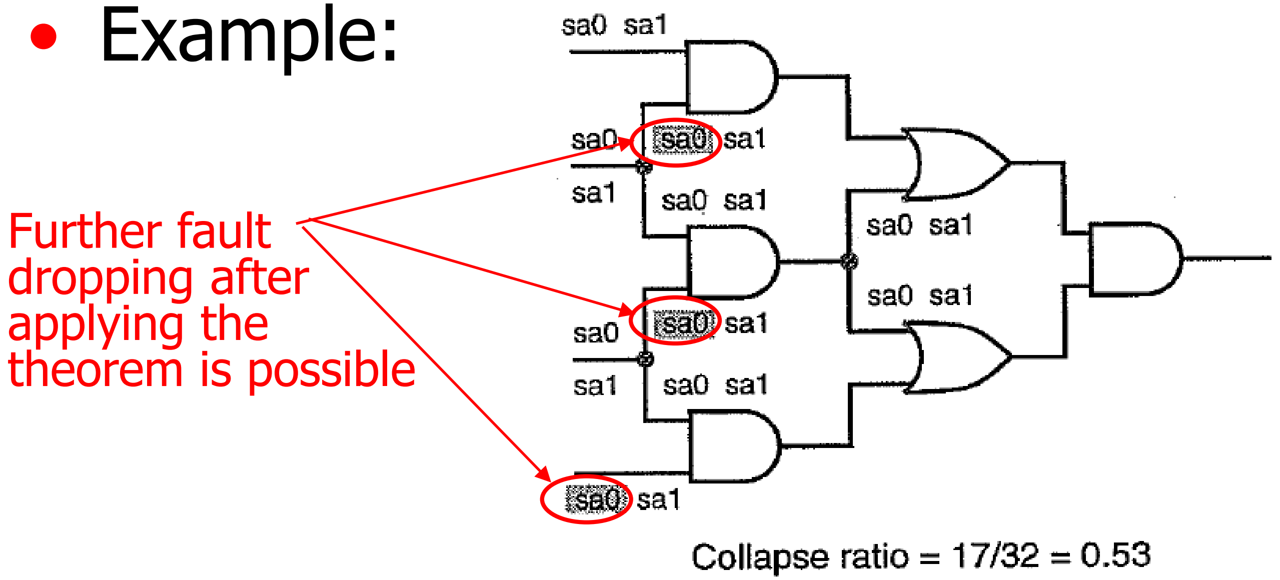
Fault Collapsing (Dropping) – An Example



- Circuit lines ordered in non-decreasing order of their output level values as $\{z_2, z_1, c_{10}, c_9, c_8, c_7, c_6, c_5, c_4, c_3, c_2, c_1, x_5, x_4, x_3, x_2, x_1\}$
- Collapse Ratio=$N_{\text{remaining faults}}/N_{\text{all\_faults}}$ =16/34=0.47
- Note: Faults remain only at primary inputs and fanout branches (Why?)

Checkpoint Theorem

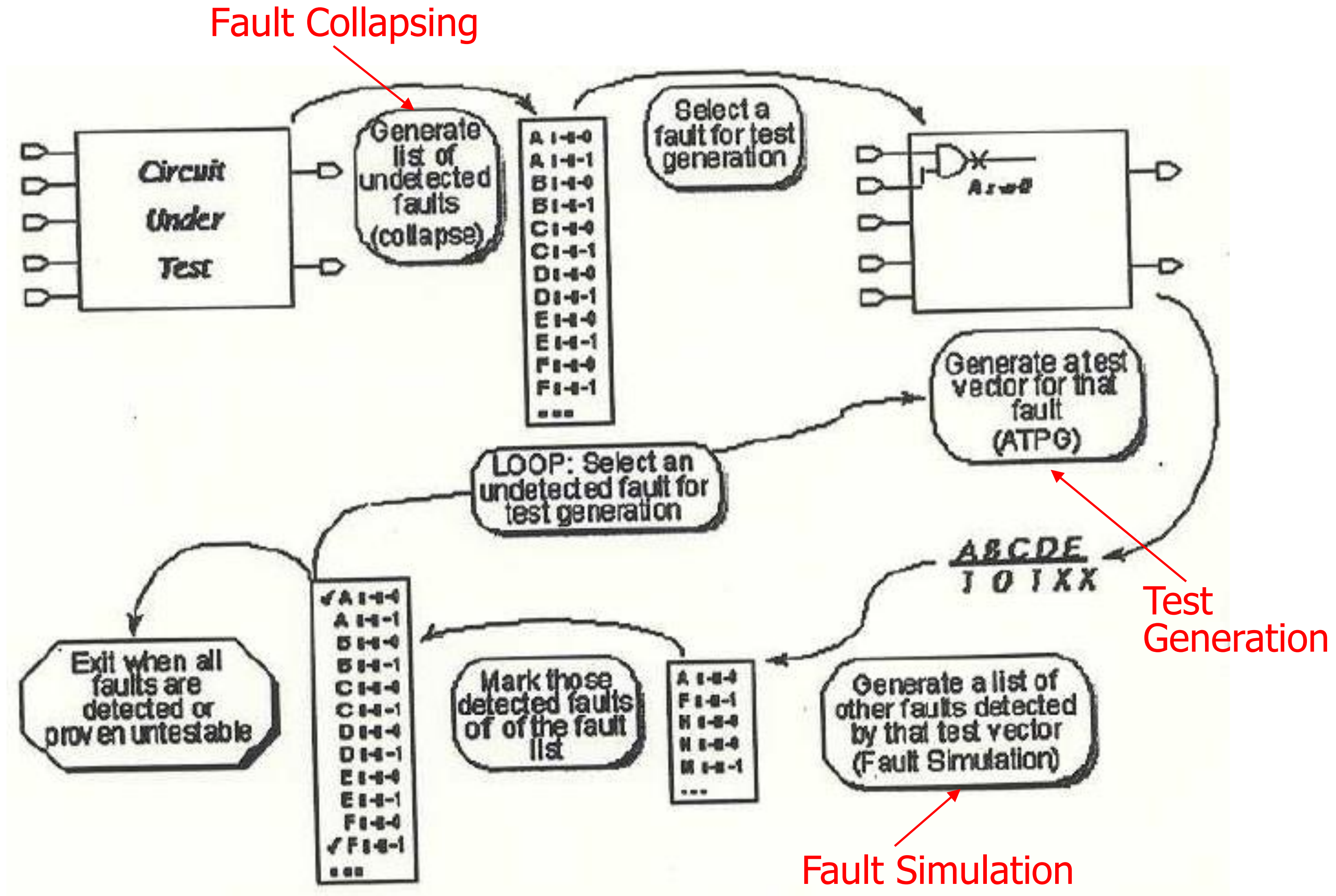
- Definition: Primary inputs and fanout branches of a combinational circuit are called **checkpoints**.
- A test set that detects all single stuck-at faults **at the checkpoints** of a combinational circuit detects all single stuck-at faults in that circuit.
- Example:


Further fault dropping after applying the theorem is possible
sa0 sa1
sa0
sa0 sa1
sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0
sa0 sa1
sa1
sa0 sa1
sa0 sa1
Collapse ratio = 17/32 = 0.53

- Proof: By contradiction.

Test Generation

Fault Analysis System (Review)


Fault Collapsing
Circuit Under Test
Generate list of undetected faults (collapse)
A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
F s-a-1
...
Select a fault for test generation
A s-a-0
Generate a test vector for that fault (ATPG)
Test Generation
ABCDE
1 0 1 X X
LOOP: Select an undetected fault for test generation
Generate a list of other faults detected by that test vector (Fault Simulation)
Fault Simulation
A s-a-0
F s-a-1
H s-a-0
H s-a-0
M s-a-1
...
Mark those detected faults of of the fault list
✓A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
✓F s-a-1
...
Exit when all faults are detected or proven untestable

Test Generation Techniques

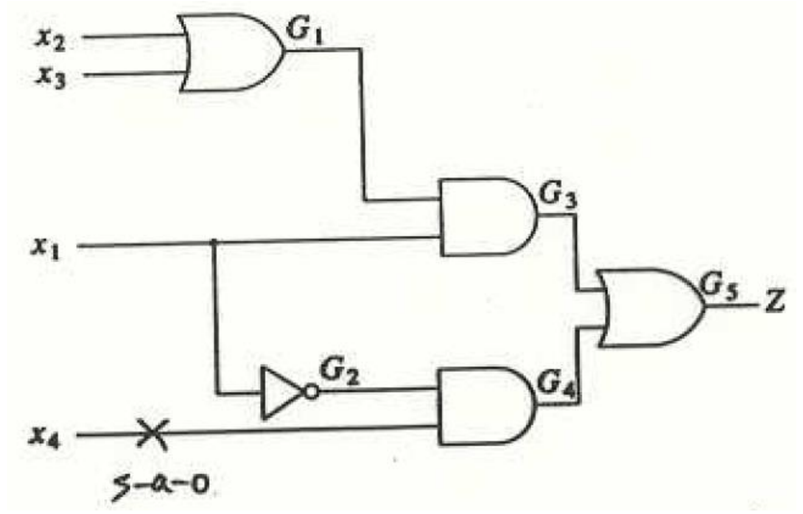
- There are two main test vector generation techniques

1. **Non-Structural**
   - Analyzes the gate-level description of a circuit and implicitly enumerate all possible input combinations to find a test vector for a target fault.
2. **Structural**
   - Analyzes the structure of a given circuit to generate a test vector for a given target fault, or declare it untestable.

Non-Structural Test Generation

Fault Detection in Combinational Circuits

- A test (vector) t detects a fault f if and only if $Z_f(t) \neq Z(t)$ (i.e., at least one of the outputs are different in N and N_f).
- For a single output circuit $Z_f(t) \neq Z(t)$ is equivalent to $Z_f(t) \oplus Z(t) = 1$
- Example: find tests to detect s-a-0 at x_4.



Fault Detection in Combinational Circuits (cont'd)

- Fault-free output: $Z = (x_2 + x_3) \cdot x_1 + \overline{x}_1 \cdot x_4$
- Faulty output: $Z_f = (x_2 + x_3) \cdot x_1$
- $Z \oplus Z_f = \left[(x_2 + x_3) \cdot x_1 + \overline{x}_1 \cdot x_4\right] \oplus \left[(x_2 + x_3) \cdot x_1\right] = \ldots = \overline{x}_1 \cdot x_4$
- Test vectors to detect s-a-0 at x_4 satisfy $\overline{x}_1 x_4 = 1$ that are:

x_1	x_2	x_3	x_4
0	x	x	1

$\equiv$

x_1	x_2	x_3	x_4
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1

Boolean Difference

- Consider a Boolean function f(x) where $x=(x_1,x_2,\ldots,x_i,\ldots,x_n)$ is the input vector.
- Cofactors of f:

$$\begin{cases} f_{x_i} = f|_{x_i=1} = f(x_1,\ldots,x_i=1,\ldots,x_1) \\ f_{\overline{x}_i} = f|_{x_i=0} = f(x_1,\ldots,x_i=0,\ldots,x_1) \end{cases}$$

- Shannon Expansion Theorem:

$$\begin{cases} f(x) = x_i \cdot f_{x_i} + \overline{x}_i \cdot f_{\overline{x}_i} \\ f(x) = \left(x_i + f_{\overline{x}_i}\right) \cdot \left(\overline{x}_i + f_{x_i}\right) \end{cases}$$

Boolean Difference (cont'd)

- Definition of Boolean difference:

$$\frac{df}{dx_i} = f_{x_i} \oplus f_{\overline{x}_i}$$

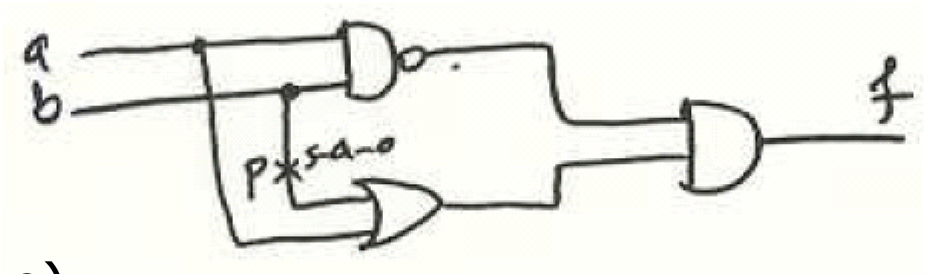
- $\frac{df}{dx_i} = 0$ means s-a-0 or s-a-1 at x_i cannot be observed at the output. In other words, s-a-f at x_i is undetectable.
- If $\frac{df}{dx_i} = 1$ then:
  - $\frac{df}{dx_i} \cdot x_i = 1$ gives all test vectors that can detect s-a-0 at x_i
  - $\frac{df}{dx_i} \cdot \overline{x}_i = 1$ gives all test vectors that can detect s-a-1 at x_i

Stimulate the fault

Propagate the fault effect

Boolean Difference (cont'd)

- Example:



$$f = \overline{ab} \cdot (a + p)$$

$$\frac{df}{dp} = f|_{p=0} \oplus f|_{p=1} = \left[\overline{ab} \cdot a\right] \oplus \left[\overline{ab}\right] = \left[(ab + \bar{a}) \cdot \overline{ab}\right] + (\overline{ab} \cdot a) \cdot ab$$

$$= ab \cdot \overline{ab} + (\bar{a} \cdot \overline{ab}) = \bar{a} \cdot (\bar{a} + \bar{b}) = \bar{a}$$

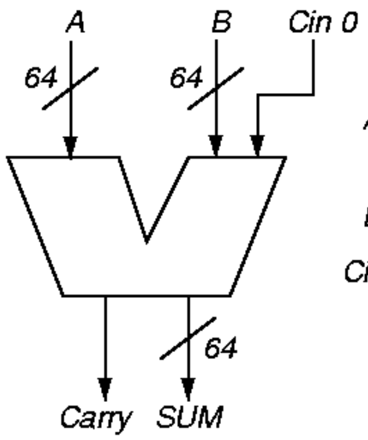
$$\Rightarrow \frac{df}{dp} \cdot p = \bar{a} \cdot b = 1 \Rightarrow ab = 01$$

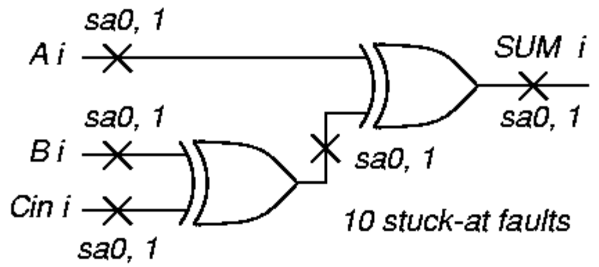
- So ab=01 is the only test vector that can detect s-a-0 at line p.

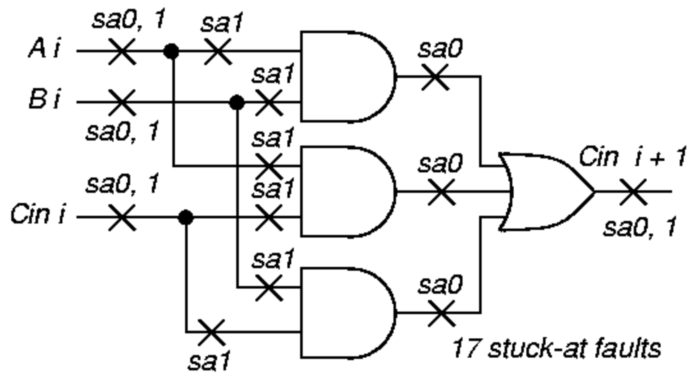
Structural Test Generation

Motivation

- Consider a 64-bit ripple-carry adder







Motivation (cont.)

- Functional (exhaustive) ATPG – generate complete set of tests for circuit input-output combinations
  - 129 inputs, 65 outputs:
  - 2^{129} = 680,564,733,841,876,926,926,749,214,863,536,422,912 patterns
  - Using 1 GHz ATE, would take 2.15×10^{22} years
- Structural test:
  - No redundant adder hardware, 64 bit slices
  - Each with 27 faults (using fault equivalence)
  - At most 64 x 27 = 1728 faults (tests)
  - Takes 0.000001728 s on 1 GHz ATE
- Designer gives small set of functional tests – augment with structural tests to boost coverage to 98^+ %

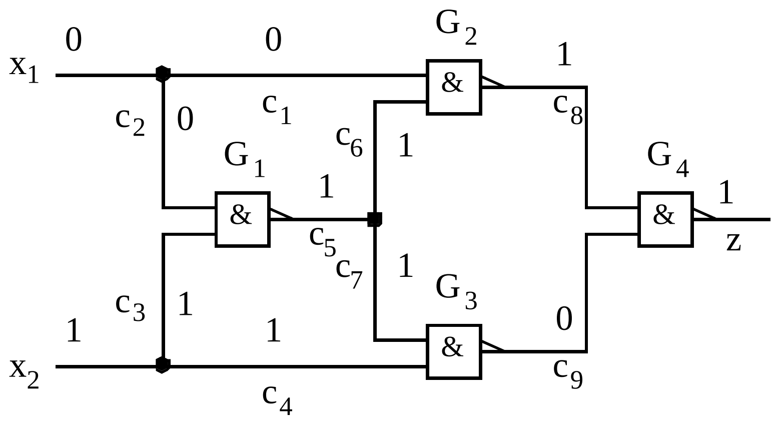
Algebra for Structural Test

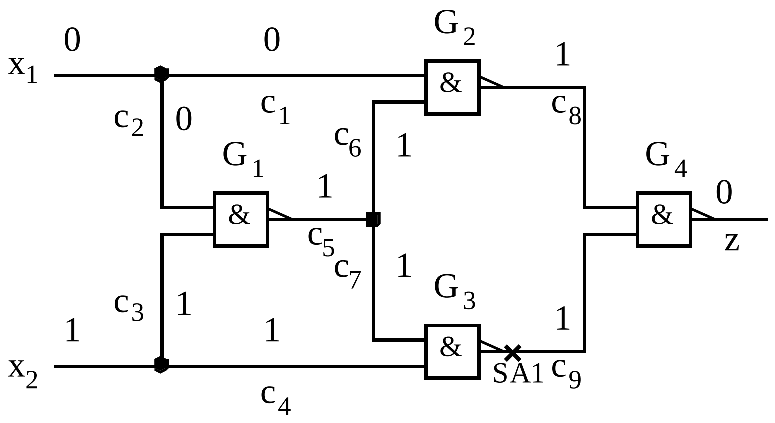
- Represent two machines, which are simulated simultaneously by a computer program:
  - Good circuit machine (1st value)
  - Bad circuit machine (2nd value)

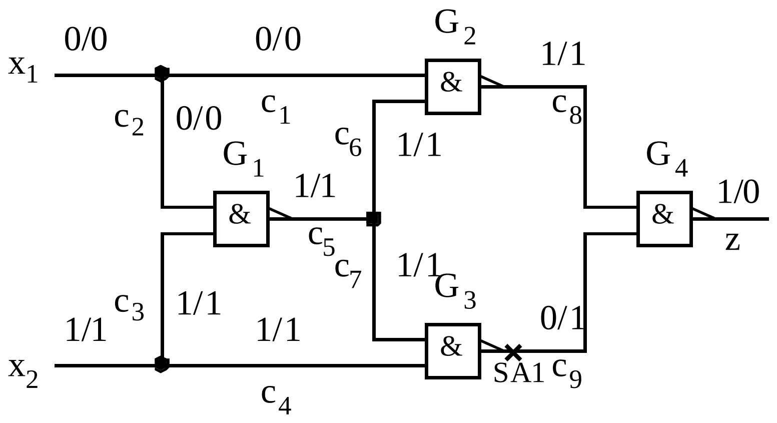
- Better to represent both in the algebra:
  - Need only 1 pass of ATPG to solve both
  - Good machine values that preclude bad machine values become obvious sooner & vice versa

Circuit Representation

- It's more efficient to carry (simulate) fault-free and faulty values at the same time.







Algebra for Structural Test (cont.)

- Roth’s 5-Valued and Muth’s 9-Valued Algebra

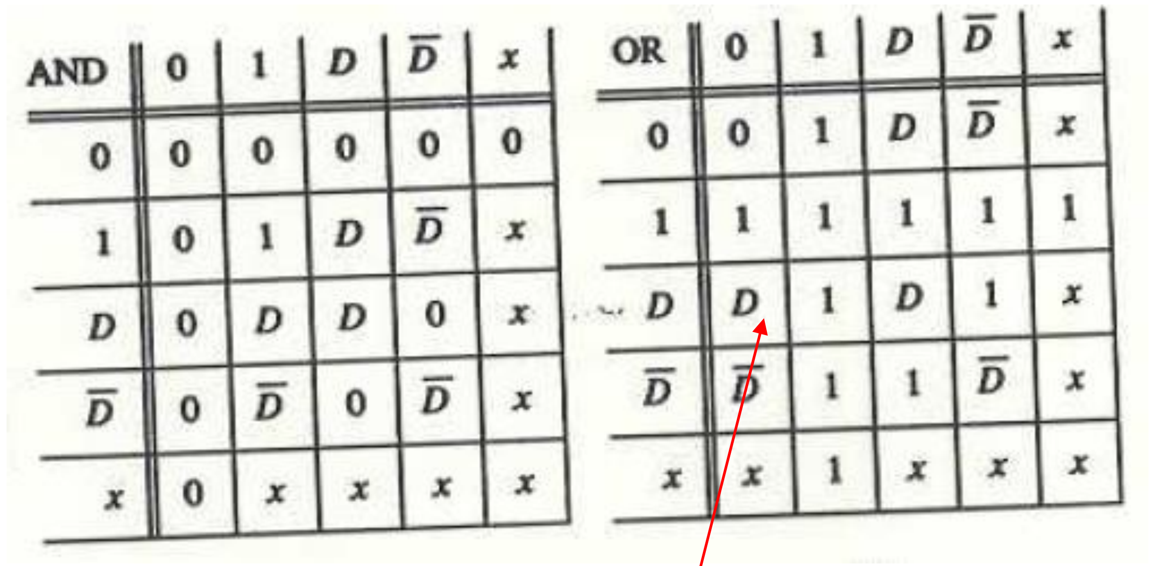
Symbol	Meaning	Fault-Free (Good)	Faulty (Bad)
0	0/0	0	0
1	1/1	1	1
D	1/0	1	0
$\overline{D}$	0/1	0	1
X	X/X	X	X
G0	0/X	0	X
G1	1/X	1	X
F0	X/0	X	0
F1	X/1	X	1

5-Valued

9-Valued

Operations in Multi-Valued Logic

- Examples (2-input AND and OR gates)



AND	0	1	D	$\overline{D}$	x
0	0	0	0	0	0
1	0	1	D	$\overline{D}$	x
D	0	D	D	0	x
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$	x
x	0	x	x	x	x

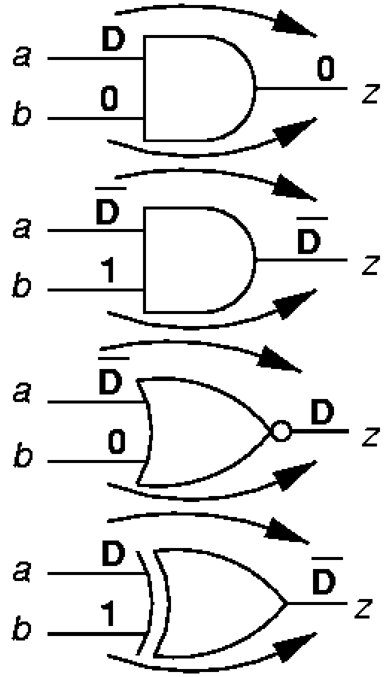
OR	0	1	D	$\overline{D}$	x
0	0	1	D	$\overline{D}$	x
1	1	1	1	1	1
D	D	1	D	1	x
$\overline{D}$	$\overline{D}$	1	1	$\overline{D}$	x
x	x	1	x	x	x

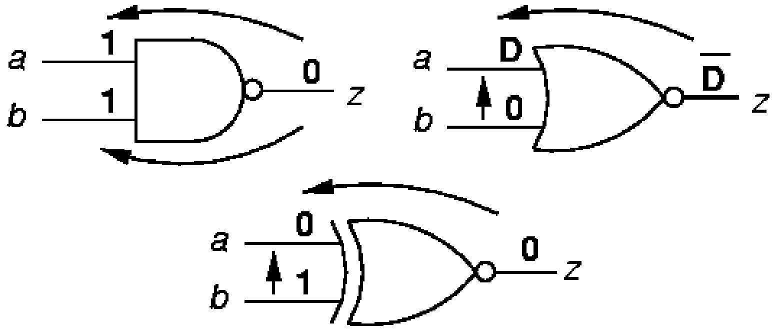
- e.g. D+0=1/0+0/0=1/0=D

Path Sensitization

- Three key tasks are needed
  1. Fault Sensitization (Stimulation)
  2. Fault (Effect) Propagation
  3. Line Justification
- Forward and Backward Implications

Examples of Forward Implications

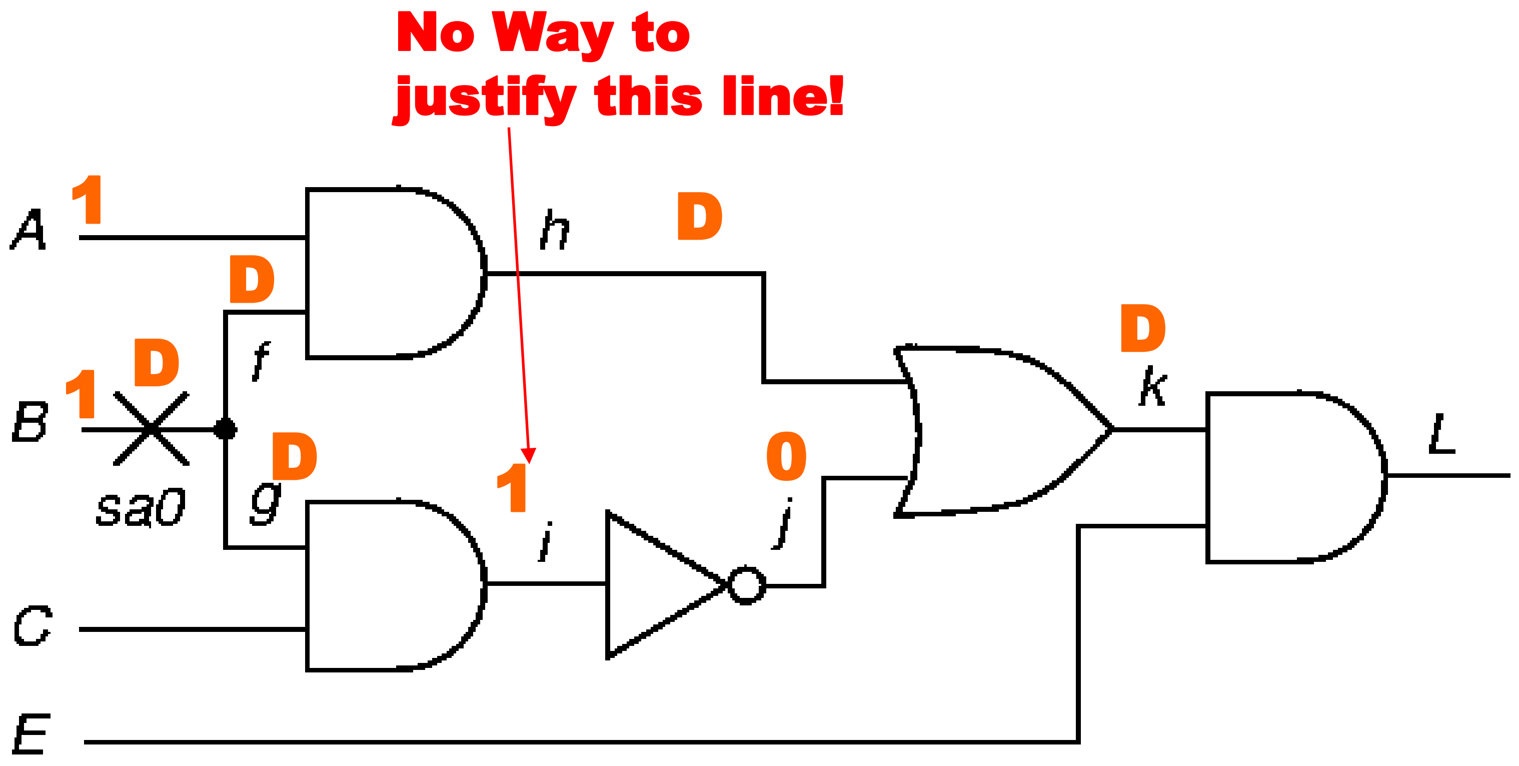




Examples of Backward Implications

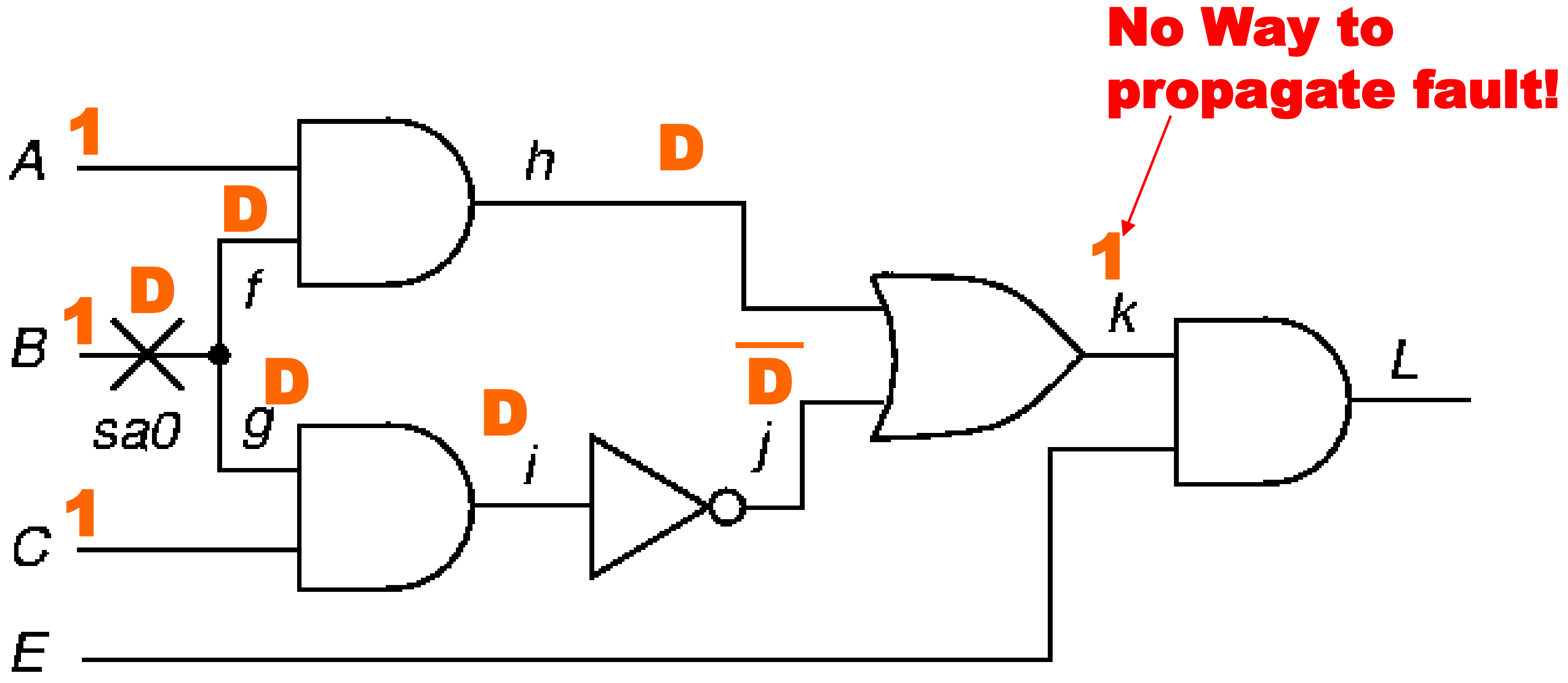
Path Sensitization Example

- Propagate through top path



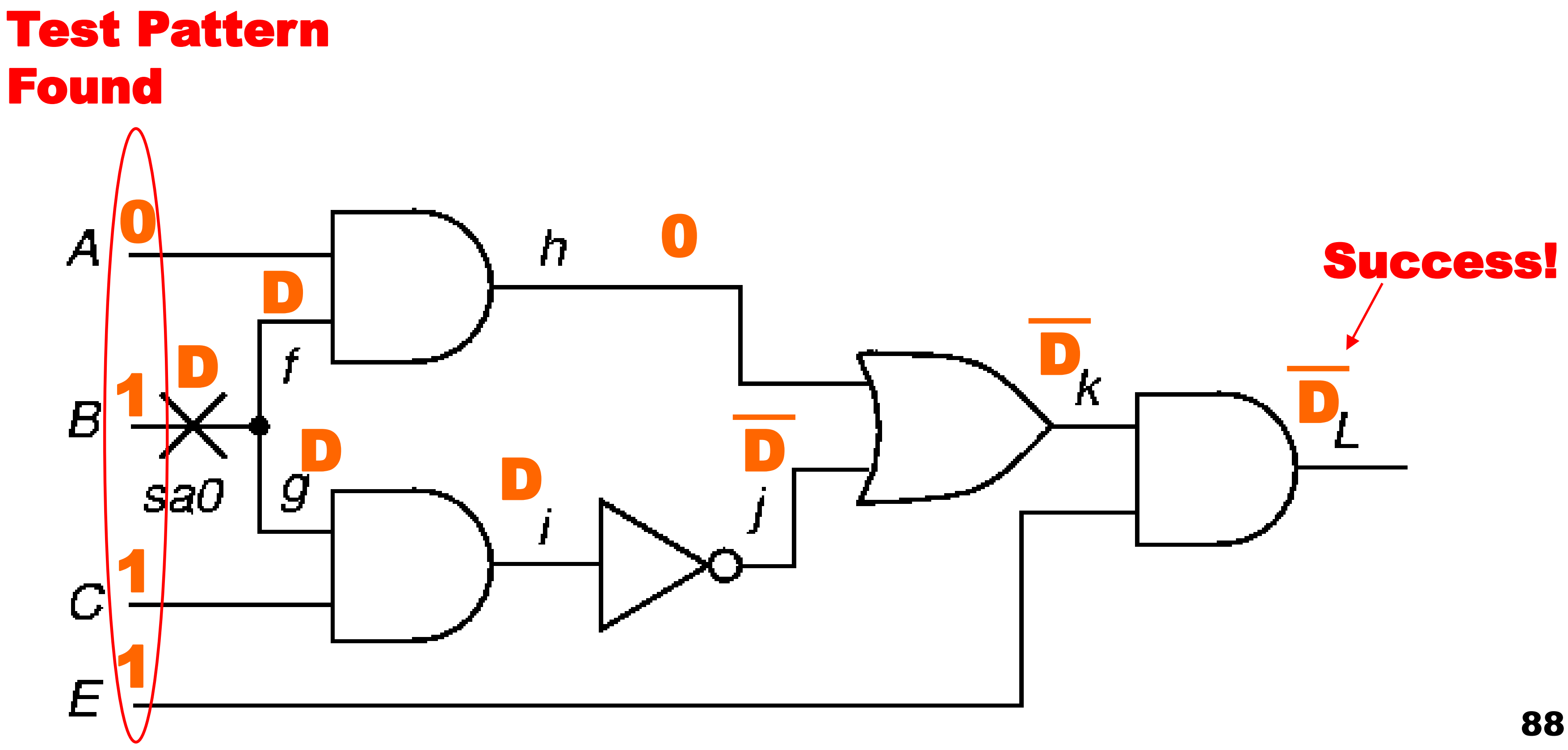
Path Sensitization Example (cont.)

- Propagate through top and bottom paths simultaneously



Path Sensitization Example (cont.)

- Propagate through the bottom path simultaneously



D-Algorithm

- Initialization
- Identification of test generation sub-tasks (TGSTs)
  0. If impossible, BACKTRACK; if complete, STOP
  1. Fault-effect excitation (FEE) – **necessary**
  2. Fault-effect propagation (FEP)
     – Each gate in D-frontier D an **alternative**
  3. Justification
     – Every gate in unjustified set U – **necessary**
- Select a TGST
  — Identify local value assignment for selected TGST
     – FEE – values at **outputs** of faulty circuit element
     – FEP – value at the **output** of gate selected for FEP
     – Justification – values at **inputs** of gate selected for justification

D-Algorithm (cont.)

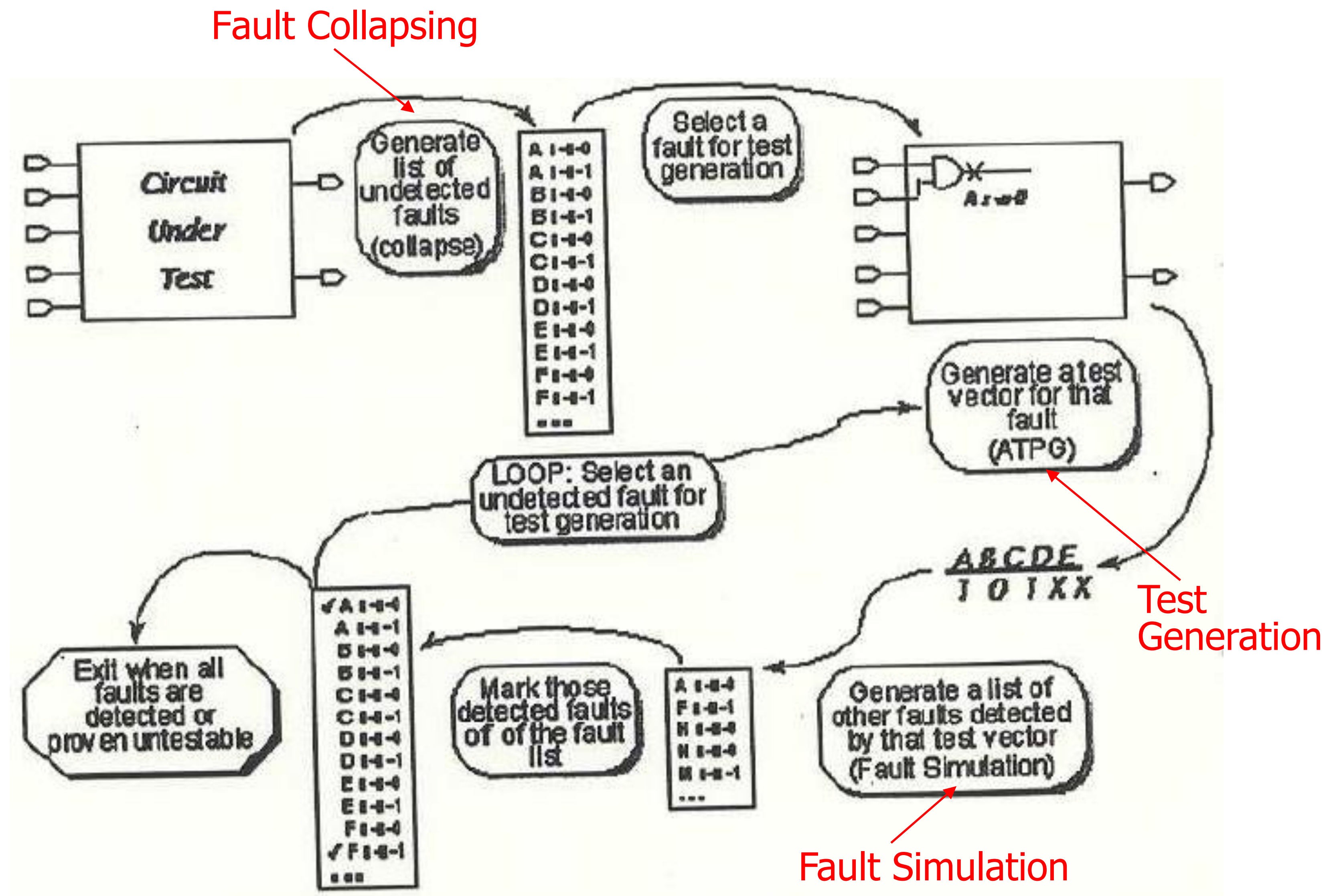
- Value assignment
  - Store untried alternative TGSTs and untried alternative value assignment for selected TGST
  - Store the values at all circuit lines
  - Make selected local value assignment (VA)
  - Perform implication – if CONFLICT, then BACKTRACK
- BACKTRACK
  - Find most recently selected VA or TGST for which one (or more) untried alternative exists
  - Restore values to **before** above assignment was made
  - Select an untried alternative TGST and/or untried local VA and repeat the value assignment step

History of Algorithms

Algorithm	Est. speedup over D-ALG (normalized to D-ALG time)	Year
D-ALG	1	1966
PODEM	7	1981
FAN	23	1983
TOPS	292	1987
SOCRATES	1574 † ATPG System	1988
Waicukauski et al.	2189 † ATPG System	1990
EST	8765 † ATPG System	1991
TRAN	3005 † ATPG System	1993
Recursive learning	485	1995
Tafertshofer et al.	25057	1997

Fault Simulation

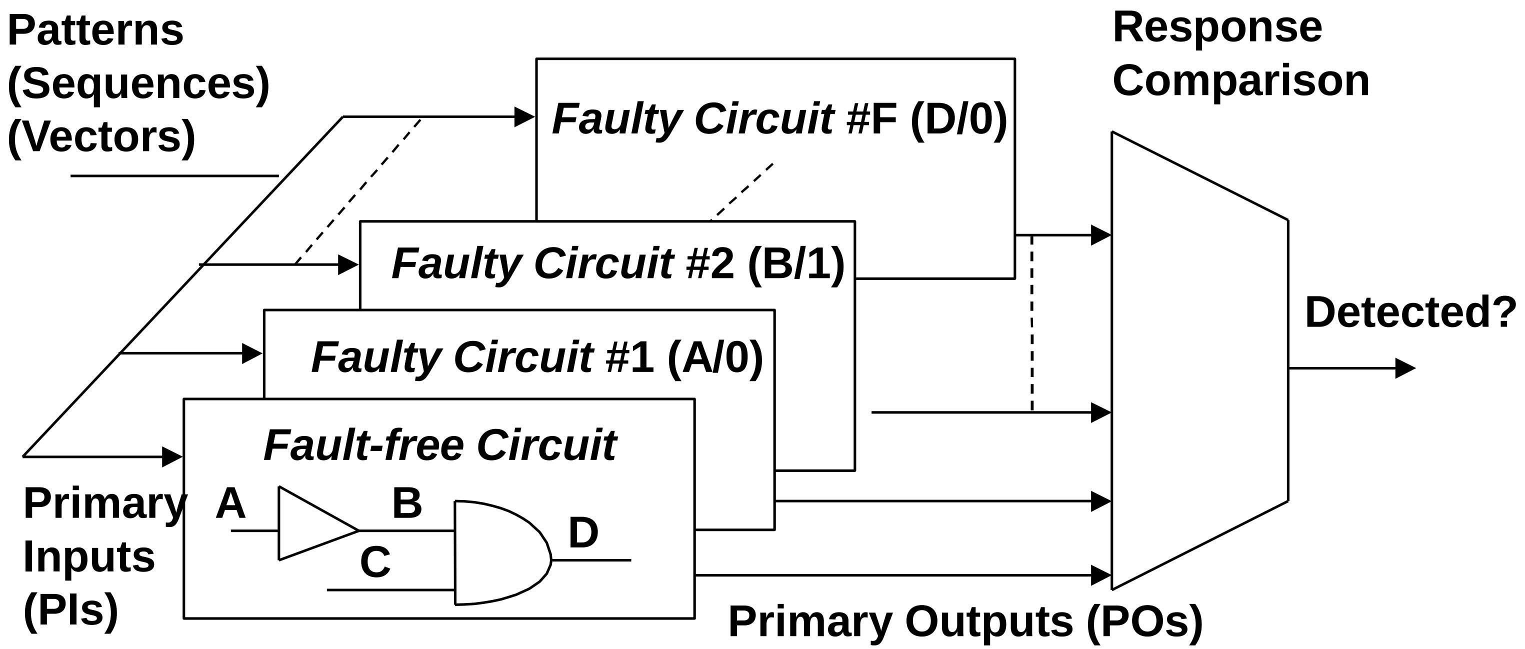
Fault Analysis System (Review)


Fault Collapsing
Circuit Under Test
Generate list of undetected faults (collapse)
A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
F s-a-1
...
Select a fault for test generation
A s-a-0
Generate a test vector for that fault (ATPG)
LOOP: Select an undetected fault for test generation
ABCDE
1 0 1 X X
Test Generation
✓A s-a-0
A s-a-1
B s-a-0
B s-a-1
C s-a-0
C s-a-1
D s-a-0
D s-a-1
E s-a-0
E s-a-1
F s-a-0
✓F s-a-1
...
Exit when all faults are detected or proven untestable
Mark those detected faults of of the fault list
A s-a-0
F s-a-1
H s-a-0
N s-a-0
M s-a-1
...
Generate a list of other faults detected by that test vector (Fault Simulation)
Fault Simulation

Why Fault Simulation?

1. To evaluate the quality of a test set
   - usually in terms of fault coverage
2. To incorporate into ATPG for test generation
   - due to its lower complexity
3. To construct fault dictionary
   - for post-test diagnosis

Conceptual Fault Simulation

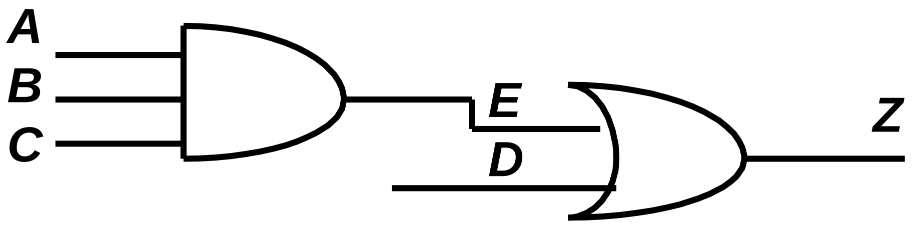


- Logic simulation on both good (fault-free) and faulty circuits

Some Basics for Logic Simulation

- For fault simulation purpose, mostly the gate delay is assumed to be zero unless the delay faults are considered. Our main concern is the functional faults.
- The logic values can be either two (0, 1) or three values (0, 1, X). For delay fault, more values will be needed.
- Two simulation mechanisms:
  - Oblivious compiled-code: circuit is translated into program and all gates are executed for each pattern.
  - Interpretive event-driven: circuit structure and gate status are stored in the table and only those gates needed to be updated with a new pattern are processed.

Oblivious Compiled Code

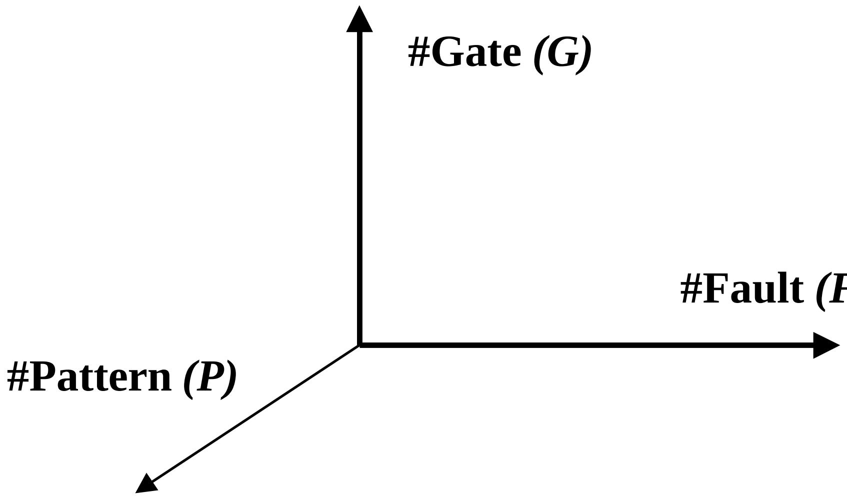


- Compiled codes
  - LDA *A* /* load accumulator with value of *A* */
  - AND *B* /* calculate *A and B* */
  - AND *C* /* calculate *E = AB and C* */
  - OR *D* /* calculate *Z = E or D* */
  - STA *Z* /* store result of *Z* */

Event-Driven

- While (event list not empty) begin
  - *t* = next time in the list
  - for every event *(i, t)* begin
    - update value of gate *i*
    - schedule fanout gates of *i* in the event list if value changes are expected
  - end
- end

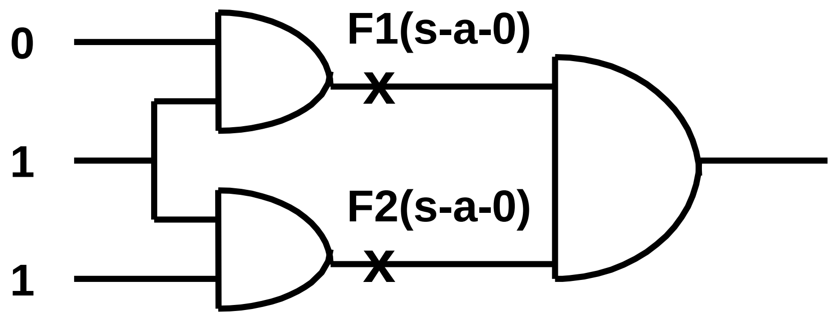
Complexity of Fault Simulation



- Complexity = *P* * *F* **G*~ $O(G^3)$ with single s-a faults
- The complexity is higher than logic simulation, $O(G^2)$, but is much lower than test pattern generation.
- In reality, the complexity is much lower due to fault dropping and advanced techniques.

Characteristics of Fault Simulation

- Fault activity with respect to fault-free circuit is often sparse both in time and in space.
- For example, F1 is not activated by the given pattern, while F2 affects only the lower part of this circuit.



- The efficiency of a fault simulator depends on its ability to exploit these characteristics.

Fault Simulation

- **Goal**: to determine the list of faults in a CUT (Circuit Under Test) and to simulate the fault-free and faulty circuits efficiently to determine if their outputs are different

- Methodologies
  - Parallel
  - Deductive
  - Concurrent
  - Critical Path Tracing

Classical Fault Simulation Techniques

- Common Characteristics:
  - In general, no restriction on the circuit types.
  - Developed before VLSI era.
- Serial Fault Simulation
  - trivial single-fault single-pattern
- Parallel Fault Simulation
- Deductive Fault Simulation
- Concurrent Fault Simulation

Parallel Fault Simulation

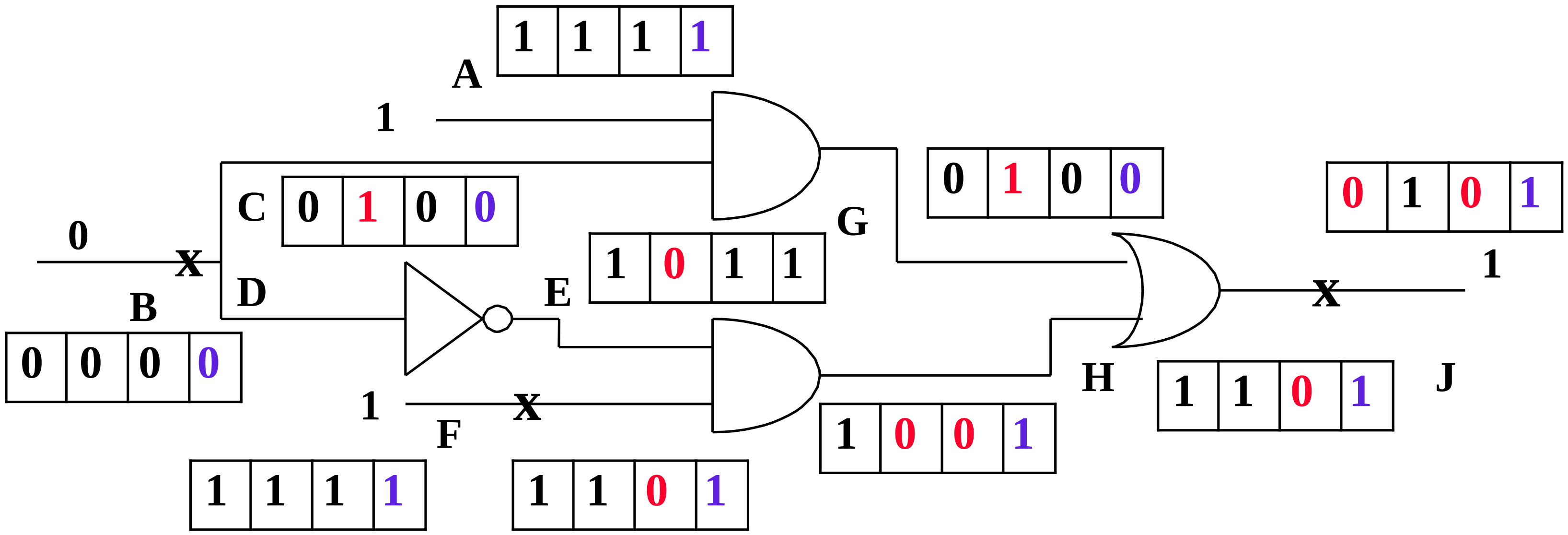
- Taking advantage of inherent parallel operation of computer words to simulate faulty circuits in parallel with fault-free circuit
  - the number of faulty circuits, or faults, can be processed parallelly is limited by the word length.
- Straightforward and memory efficient
- Some weaknesses:
  - An event, a value change, of a single fault or fault-free circuit leads to the computation of the entire word.
  - The fault-free logic simulation is repeated for the number of passes.

Example of Parallel Fault Simulation

- Consider three faults: B/1, F/0, and J/0
  - Bit-space:

J/0	B/1	F/0	FF

where FF = Fault-free



Deductive Fault Simulation

- A list of faults associated with each line containing the identifier of each fault which produces a fault effect on this line.
  - Only the faults with fault effect, or difference w.r.t. fault-free circuit, is retained in the list.
- The propagation of such lists can be based on set operation derived the gate types and values.
  - The list update is performed with each new pattern, which is not efficient.
  - The list may dynamically grow is size, which incurs memory explosion problem.

Fault List Propagation Rule

- **Let the gate G = F(A,B) with input lists L_A and L_B and output list L_G to be updated.**
- **If in the fault-free circuit,**
  - **the value of G is 0(1), use F(F-)**
  - **the value of gate input A is 0(1), replace A in logic expression by L_A(L_A-) and A- by L_A-(L_A).**
- **Replace * by *intersection* and + by *union*.**
- **Add G/1(G/0) to the list L_G.**

Fault List Propagation Rule

- 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 follows:

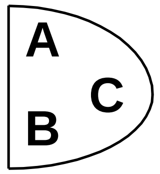
$$if\ C = \emptyset\ then\ L_Z = \left\{\bigcup_{j\in I} L_j\right\} \bigcup \{Z\ s-a-(c\oplus i)\}$$

$$else\ L_Z = \left\{\bigcap_{j\in I} L_j\right\} - \left\{\bigcup_{j\in I-C} L_j\right\} \bigcup \{Z\ s-a-(\bar{c}\oplus i)\}$$

In other words, 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 c propagates to the output. In both cases we add the local fault of the output.

Illustration of Fault List Propagation

Consider a two-input AND gate



Case 1: A=1, B=1, C=1 at fault-free,

$L_C = L_A + L_B + \{C/0\}$

Case 2: A=1, B=0, C=0 at fault-free,

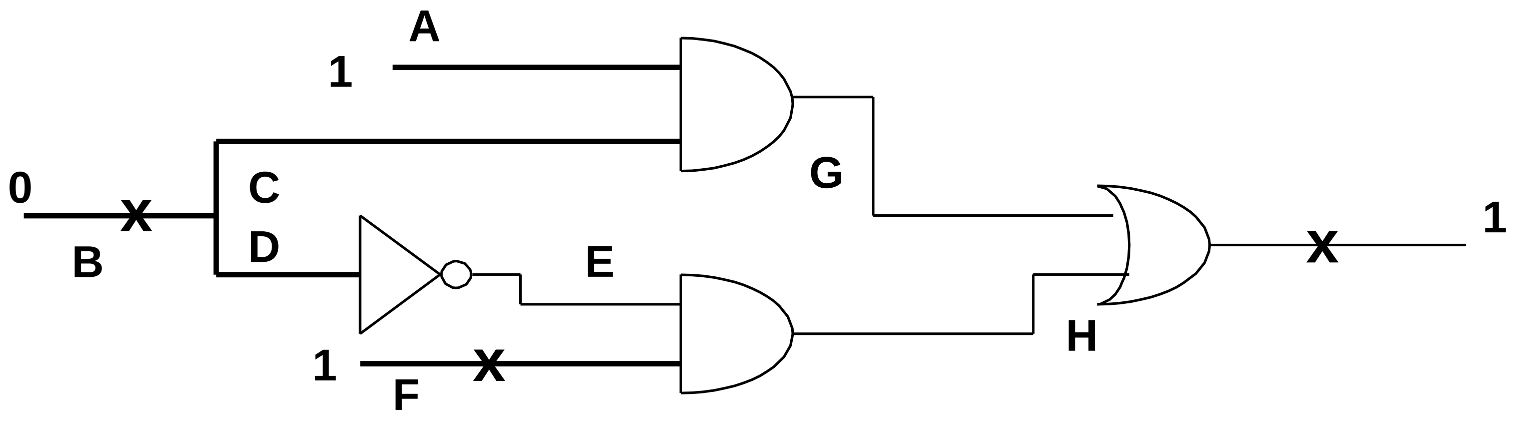
$L_C = L_{A^-} * L_B + \{C/1\}$

Case 3: A=0, B=0, C=0 at fault-free,

$L_C = L_A * L_B + \{C/1\}$

Example of Deductive Simulation Ia

- Consider 3 faults: B/1, F/0, and J/0

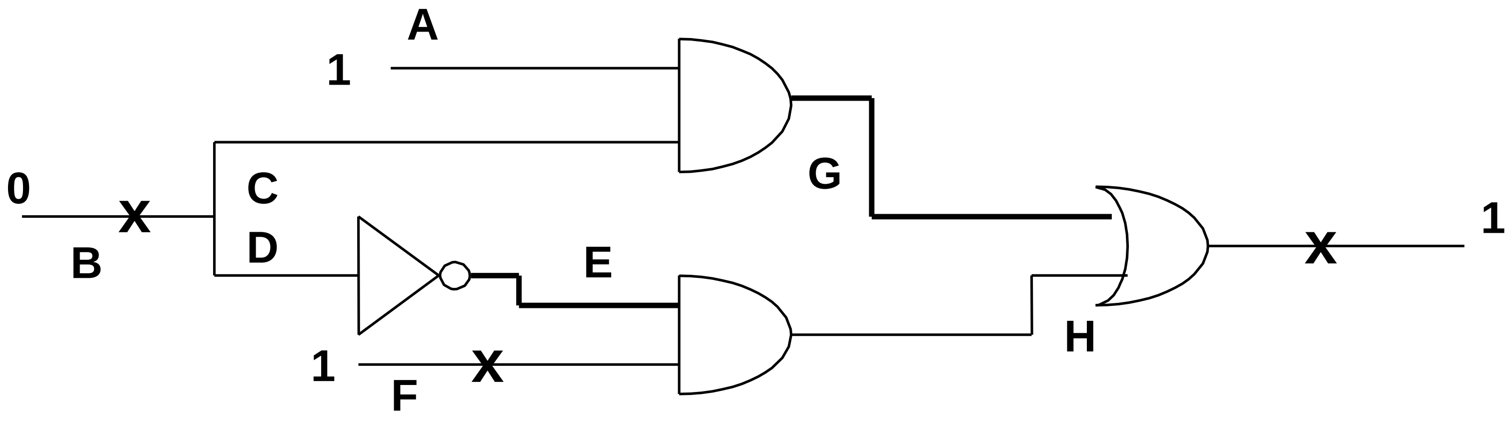


L_B = {B/1}, L_F = {F/0}, L_A = 0

L_C=L_D = {B/1}

Example of Deductive Simulation Ib

- Consider 3 faults: B/1, F/0, and J/0



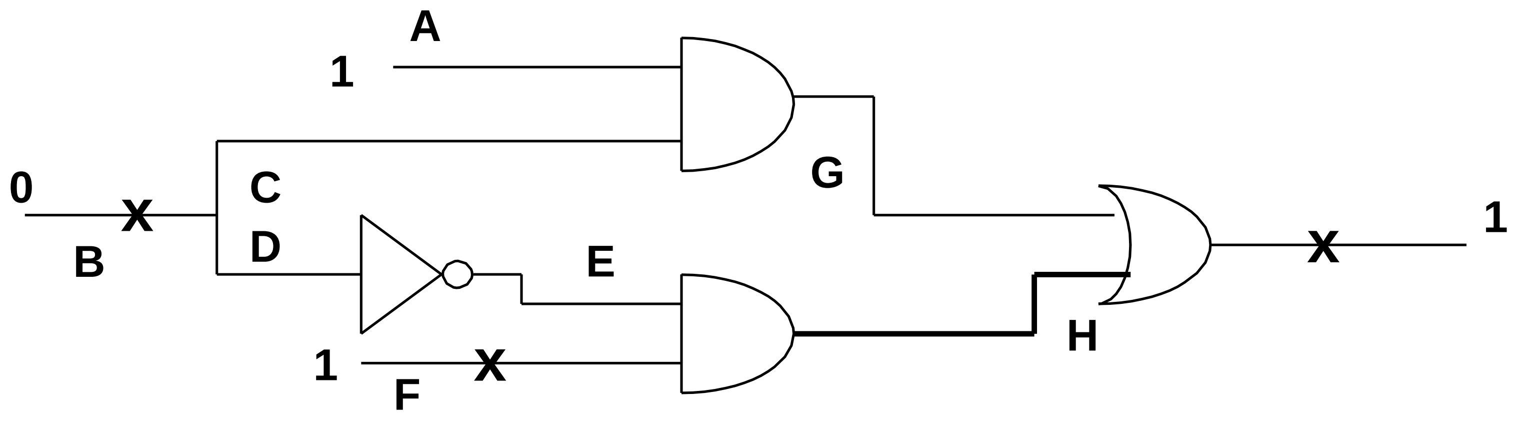
L_B = {B/1}, L_F = {F/0},

L_C=L_D = {B/1},

L_G = {B/1}, L_E = {B/1}

Example of Deductive Simulation Ic

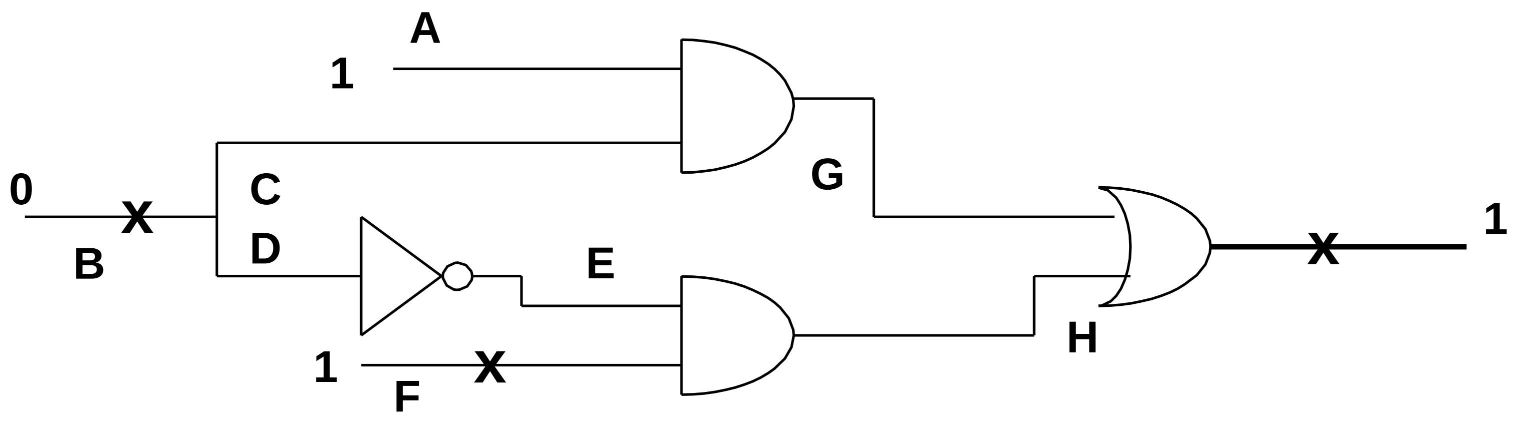
- Consider 3 faults: B/1, F/0, and J/0



L_B = {B/1}, L_F = {F/0},
L_C=L_D = {B/1}, L_G = {B/1},
L_E = {B/1}, L_H = {B/1, F/0}

Example of Deductive Simulation Id

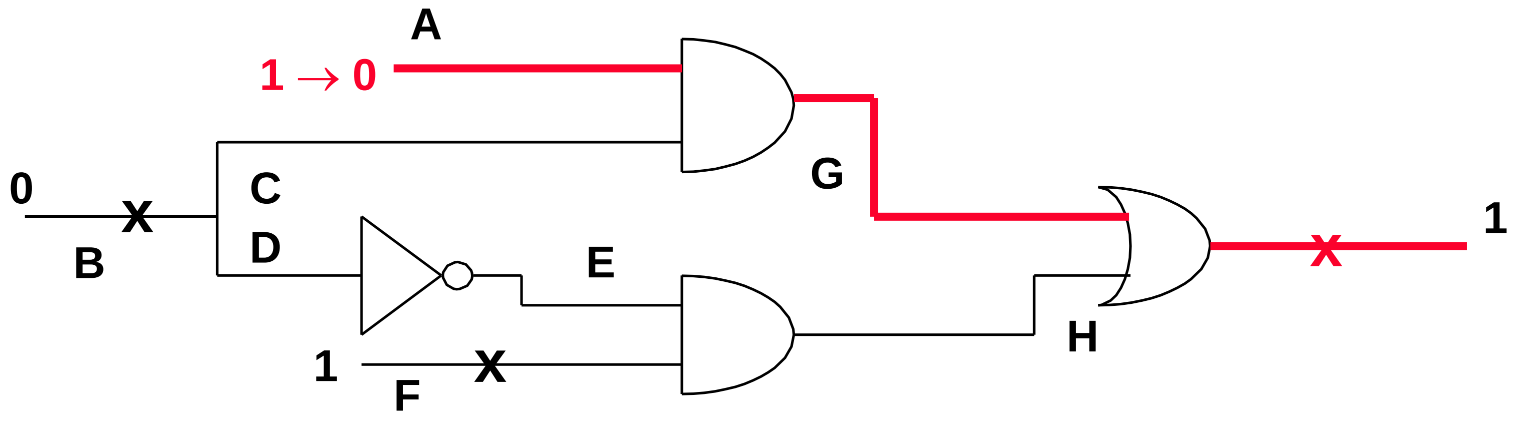
- Consider 3 faults: B/1, F/0, and J/0



L_B = {B/1}, L_F = {F/0},
L_C=L_D = {B/1}, L_G = {B/1},
L_E = {B/1}, L_H = {B/1,F/0}, L_J = {F/0,J/0}

Example of Deductive Simulation II

- When A changes from 1 to 0



L_B = {B/1}, L_F = {F/0},

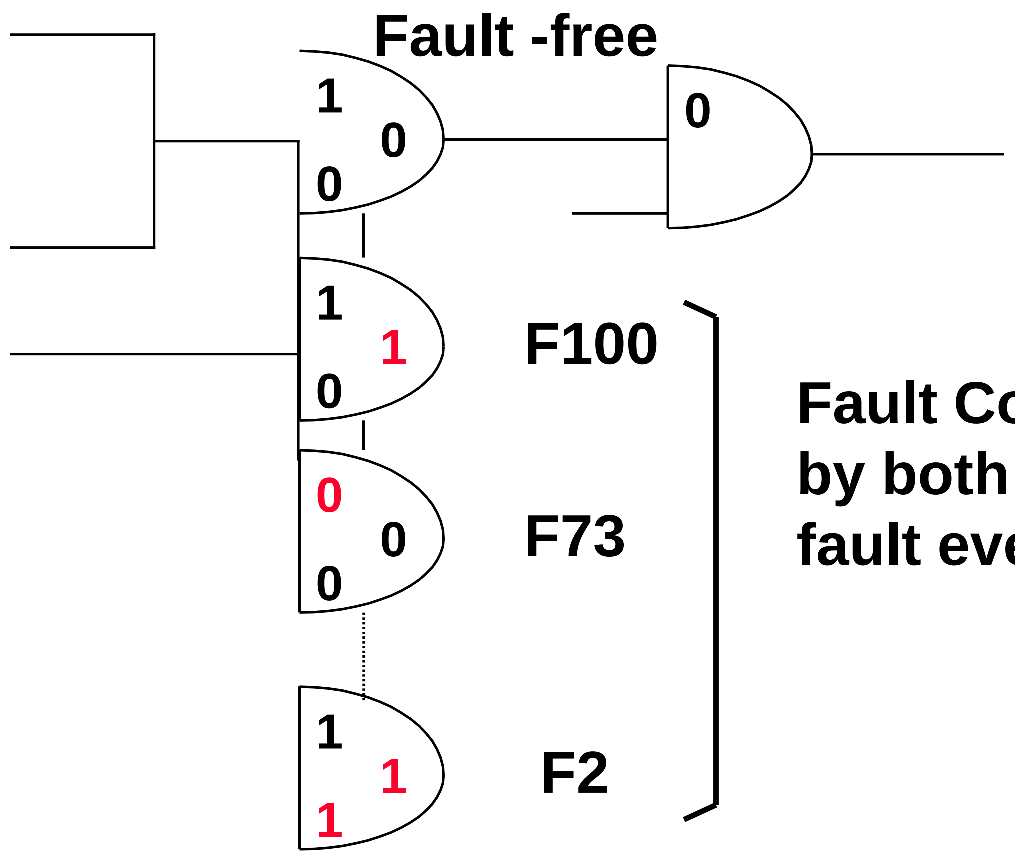
L_C=L_D = {B/1}, L_G = 0,

L_E = {B/1}, L_H = {B/1,F/0}, L_J = {B/1,F/0,J/0}

Concurrent Fault Simulation

- **Each gate retains a list of fault copies each of which stores the status of a fault exhibiting difference from fault-free values.**
- **Simulation mechanism is similar to the conceptual fault simulation except that only the dynamical difference w.r.t. fault-free circuit is retained.**
- **Very versatile in circuit types and gate delays**
- **Although theoretically all faults in a circuit can be processed in one pass, memory explosion problem restricts the fault number in each pass.**

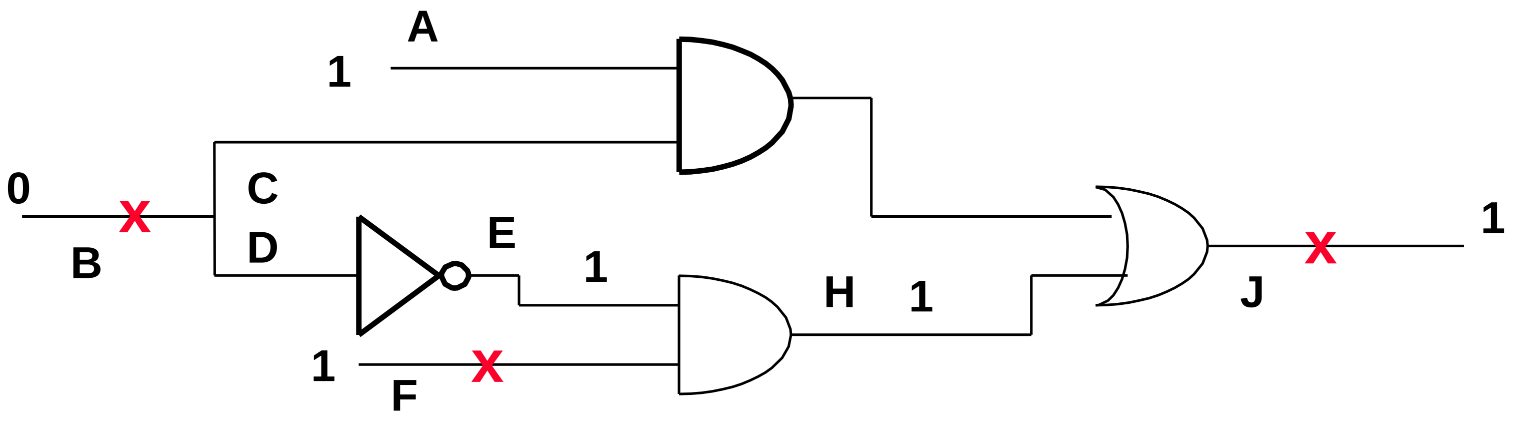
Concurrent Fault Simulation



Fault Copies: updated by both logic event and fault events

Example of Concurrent Simulation I

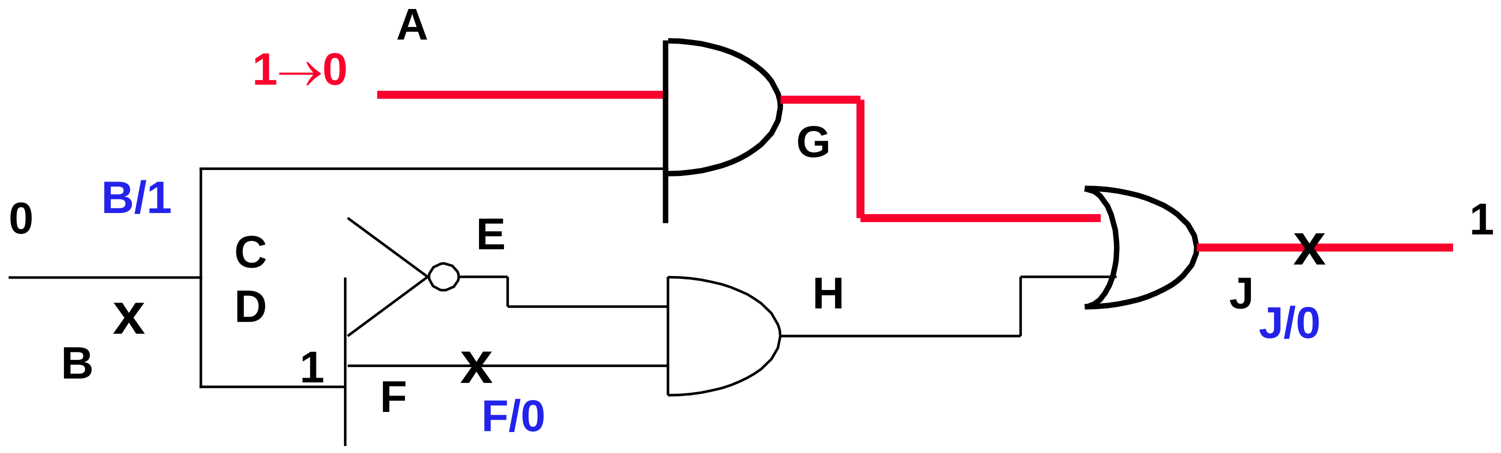
- Consider 3 faults: B/1, F/0, and J/0



L$_G$ = {10_0, B/1:11_1}   L$_E$ = {0_1, B/1:1_0}
L$_H$ = {11_1, B/1:01_0, F/0:10_0}
L$_J$ = {01_1, B/1:10_1, F/0:00_0, J/0:01_0}

Example of Concurrent Simulation II

- When A changes from 1 to 0



LG = {00_0, B/1:01_0} LE = {0_1, B/1:1_0}
LH = {11_1, B/1:01_0, F/0:10_0}
LJ = {01_1, B/1:00_0, F/0:00_0, J/0:01_0}

Modern Combinational Simulation Techniques

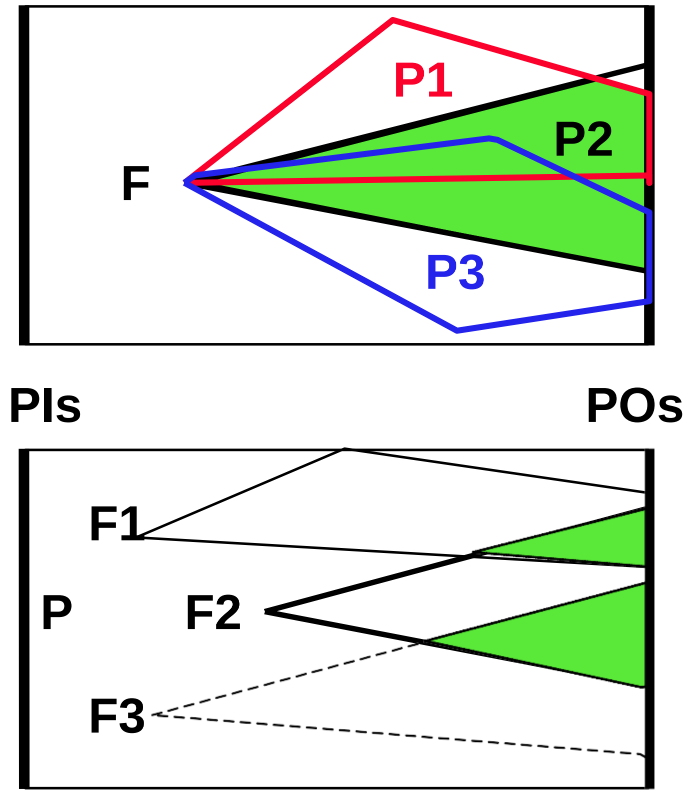
- Parallel pattern
- Critical path tracing
- Other sophisticated techniques

Parallel-Pattern Single-Fault Propagation (PPSFP)

- **Many patterns are simulated in parallel for both fault-free and faulty circuits. The number of patterns is a multiple of computer word length.**
- **Coincident logic events of fault-free circuit from these patterns can be simulated in parallel.**

  –reduction of logic event simulation time
- **Coincident fault events of faulty circuits from these patterns can also be simulated in parallel.**
  - **reduction of fault event simulation time**
- **Simple and extremely efficient**
  - **basis of all modern combinational fault simulators**

Comparison with Parallel-Fault



- Fault event maps for parallel-pattern (upper) and parallel-fault (lower)
- Parallel-Pattern case:

  Consider three patterns: P1, P2, P3.
- Parallel-Fault case:

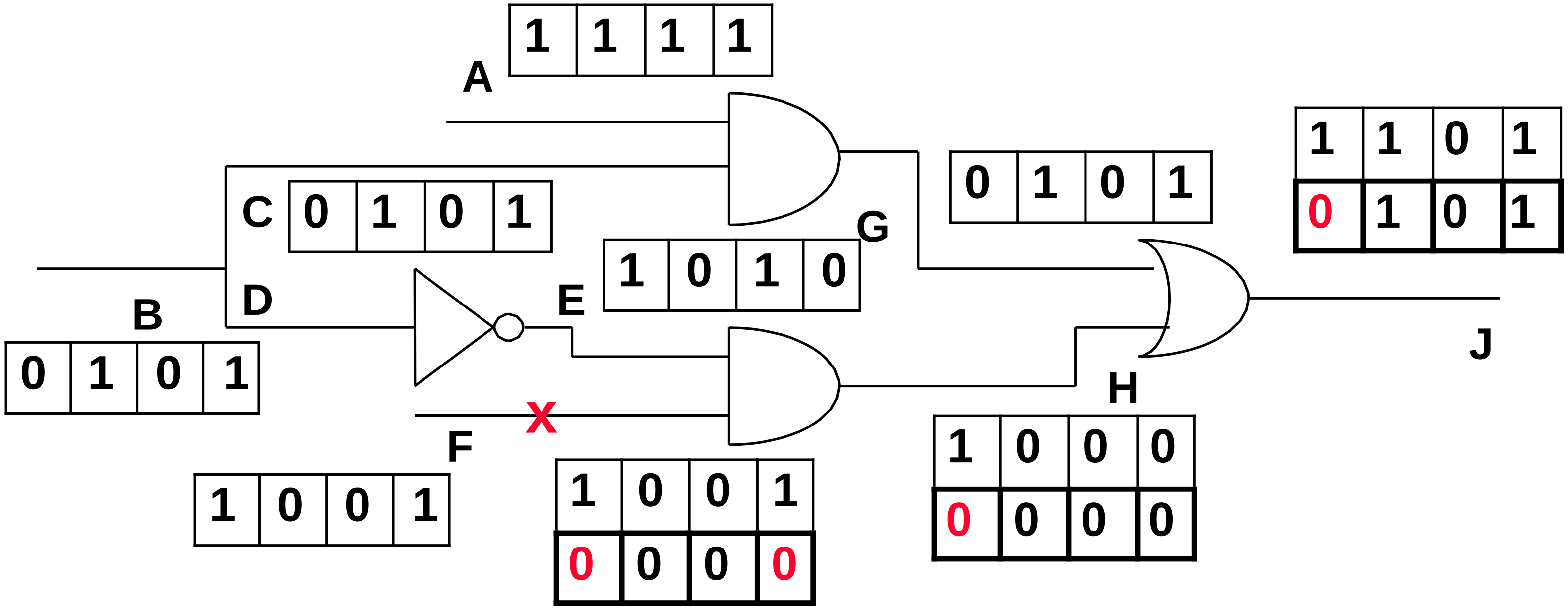
  Consider three faults: F1, F2, F3.
- The overlapping of fault events is inherently much higher in parallel-pattern simulation, and hence more event can be done at the same time.

Example of Parallel Pattern Simulation

- Consider one fault F/0 and four patterns: P3,P2,P1,P0
  - Bit-space:

P3	P2	P1	P0

faulty value in red



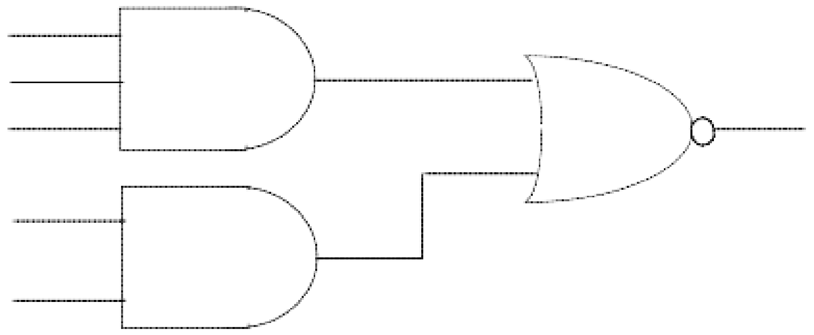
Critical Path Tracing

- Two-step Procedure:
  - Fault-free simulation and identification of the sensitive gate inputs
  - Backtracing to identify the critical lines (critical path tracing)
- O(G) complexity for fanout-free circuits --- very rare in practice
- However, it becomes effective in fanout-free region when stem faults are simulated by parallel-pattern fault simulator.

Critical Path Tracing

- Efficient since it implicitly targets all faults in a circuit in a single pass
- However, in its basic form applicable only to fanout-free circuits, i.e., circuits with no fanout systems
- Since most practical circuits have fanouts
  - —Circuits are partitioned into fanout-free regions
  - —Critical path tracing applied within each fanout-free region
  - —Additional steps are needed to check criticality of the paths across fanout-free regions (FFR)

Fanout-Free Circuits



- Definition: (1) Every line branches out to at most one gate (2) There is just one path from every line to the primary output
- *Theorem*: In a fanout-free circuit, every test set that detects all SSL faults on primary input lines is complete.
- *Corollary*: An n-input fanout-free circuit can be tested with at most 2n test patterns (non-optimal)

Critical Path Tracing

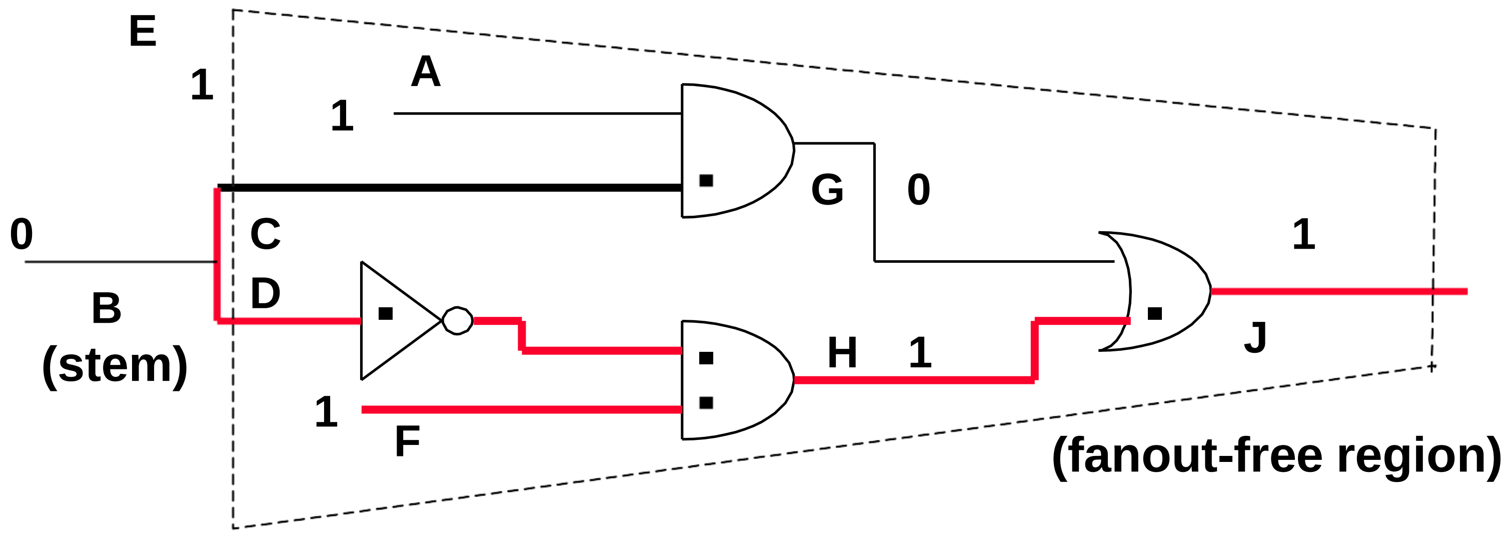
- Efficient since it implicitly targets all faults in a circuit in a single pass
- However, in its basic form applicable only to fanout-free circuits, i.e., circuits with no fanout systems
- Since most practical circuits have fanouts
  - Circuits are partitioned into fanout-free regions
  - Critical path tracing applied within each fanout-free region
  - Additional steps are needed to check criticality of the paths across fanout-free regions (FFR)

Basics of Critical Path Tracing

- A line *l* is *critical* w.r.t. a pattern *t* iff *t* detects the fault *l/v.*
- Paths of critical lines are critical paths.
- A gate input *i* is *sensitive* if complementing the value of *i* changes the value of the gate output.
- A gate input *i* is *critical* w.r.t. a pattern *t* if the gate output is critical and *i* is sensitive w.r.t. *t*.
- In a fanout-free circuit, the criticality of all lines can be determined by backward traversing the successive sensitive gate inputs from POs, in linear time.

Example of Critical Path Tracing

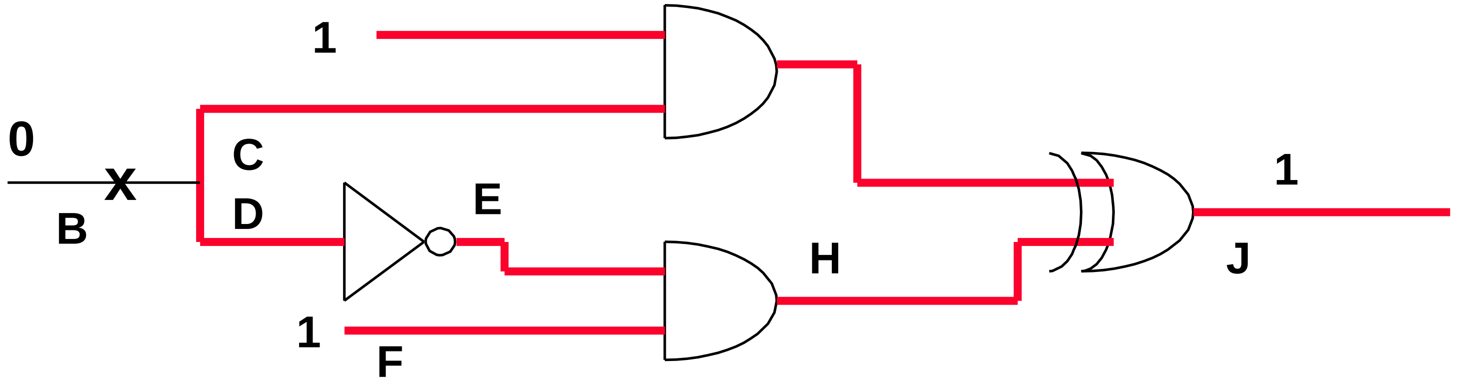
▪ sensitive input, 🟥 critical line



Detected faults in the fanout-free region:
{J/0, H/0, F/0, E/0, D/1}

Anomaly of Critical Path Tracing

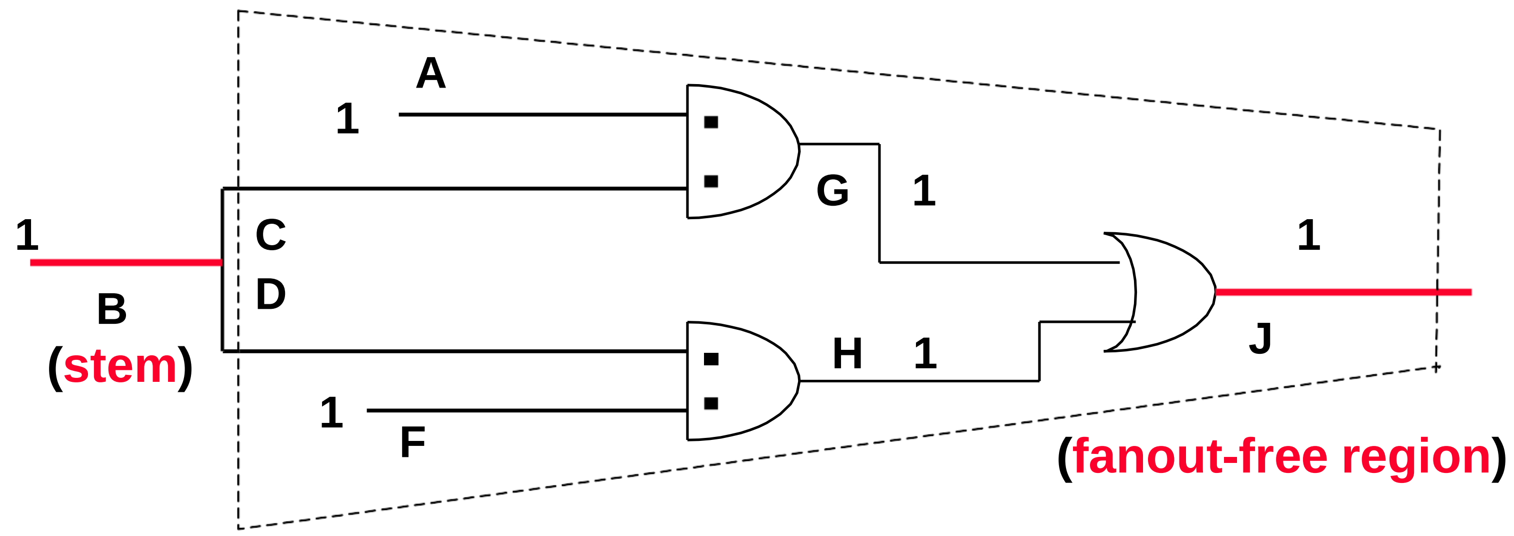
- Stem criticality is hard to infer from branches. E.g. is B/1 detectable by the given pattern?



- It turns out B/1 is not detectable even though both C and D are critical, because its effects cancel each other out at gate J.
- There is also multiple path sensitization problem.

Multiple Path Sensitization

▪ sensitive input, ▬ critical line



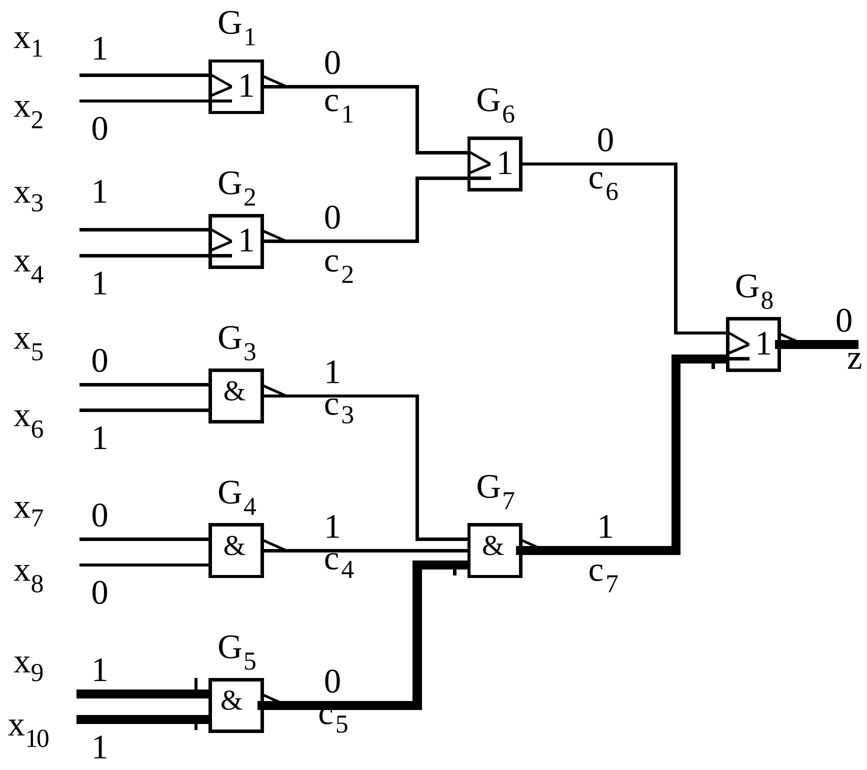
Both C and D are not critical, yet B is critical and B/0 can be detected at J by multiple path sensitization.

Key Concept

- Given a vector P = (p_1, p_2,…, p_n), assume fault-free circuit simulation has been performed to compute the value implied at each line by P
  - An input c_{i_l} of a gate G is said to be **sensitive** at its output c_j, if complementing the values at c_{i_l} (without changing values at any of G's other inputs) will complement the value at c_j
  - A line c_i in a circuit C is said to be **critical** for vector P, if P detects a SAϖ Fault at c_i, when P implies a value w at c_i

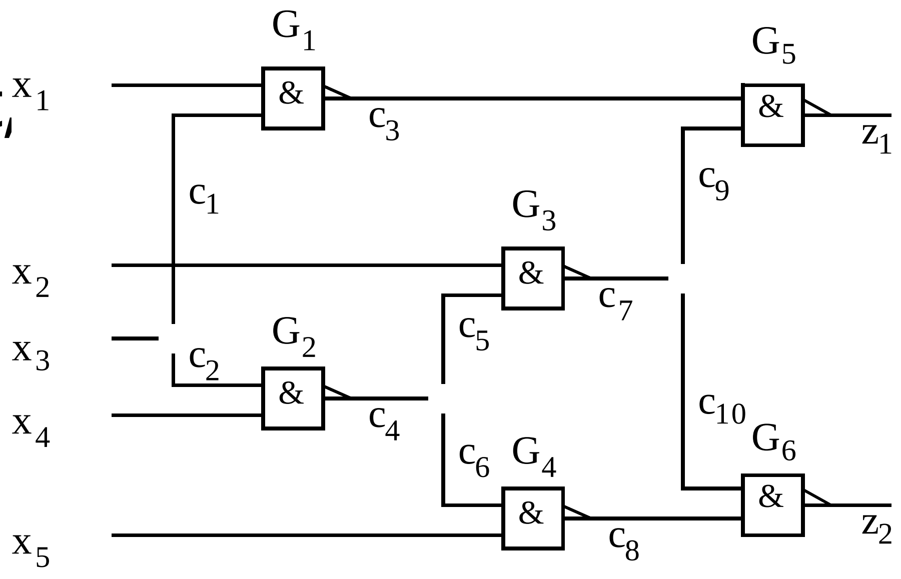
Critical Path Tracing - Process

- Perform fault-free circuit simulation
- Compute sensitivity of each input of each gate
- Mark the output as critical
  - —Traversing the circuit lines from output to inputs, compute criticality of each line



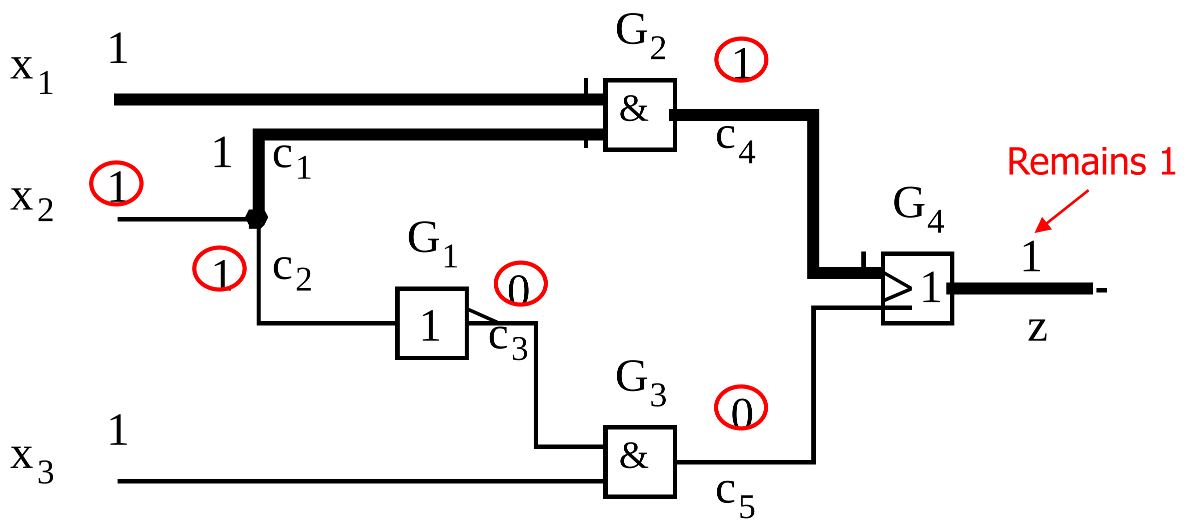
Critical Path Tracing – Process (cont'd)

- A circuit with fanouts can be partitioned into **fanout-free regions** (FFRs) by disconnecting each stem from its branches
  - Each connected sub-circuit is an FFR
  - Each FFR can be identified by its output, which is either a primary output or a stem of a fanout system
  - Hence, we will refer to an FFR by its output, as FFR(x_3), FFR(c_4), FFR(c_7), FFR(z_1), and FFR(z_2)



Critical Path Tracing – Process (cont'd)

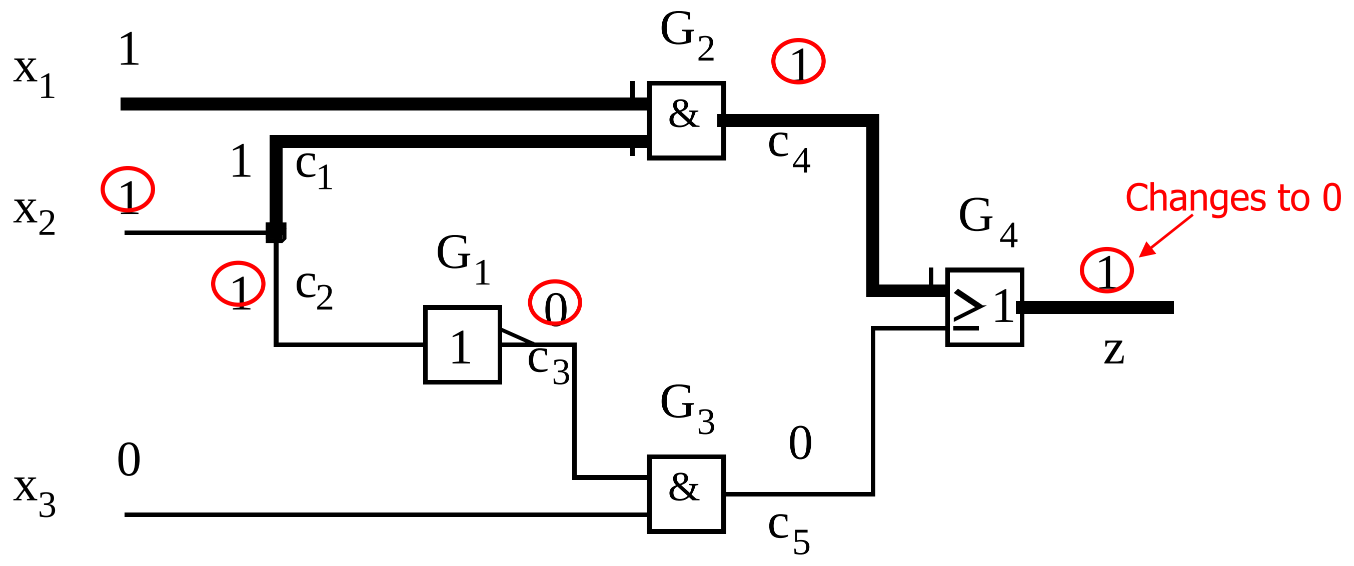
- Difficulty with applying critical path tracing to circuits with fanouts
  - This circuit can be partitioned into two FFRs: FFR(z) and FFR(x_2)



  - Critical path tracing on FFR(z) for this vector identifies x_1, c_1, c_4, and z as critical
  - Even though c_1 is critical, explicit fault simulation shows that **the stem x_2 is not critical**

Critical Path Tracing (cont'd)

- For this vector, critical path tracing of FFR(z) again identifies x_1, c_1, c_4, and z as critical

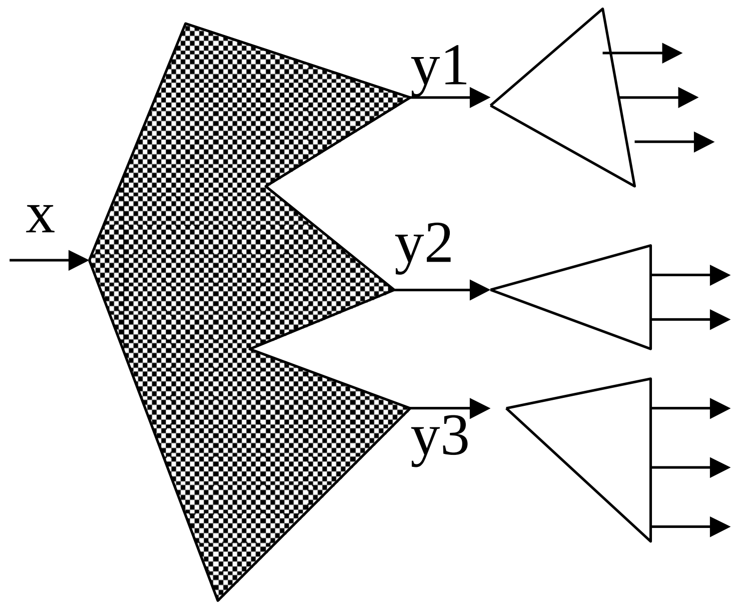


- However, in this case, **the stem x_2 is critical**
- Hence, no simple universally applicable relation exists between criticality of a stem and those of its branches

Critical Path Tracing (cont’d)

- Above examples show that no universally applicable relation exists between the criticality of a stem and those of its branches
- To obtain a complete fault simulation
  - Critical path tracing can be used within each FFR
  - Explicit fault simulation must be performed for faults at a stem to determine its criticality
  - The results of above two steps can be combined to compute the criticality of all lines in an arbitrary circuit

Other Sophisticated Techniques



d(x) = d((x-y1) d(y1) + d(x-y2) d(y2)
+ d(x-y3)d(y3)

- Stem Region Methods such as Exit Lines
  - in the right figure, d(x) is determined from its exit lines y1, y2, and y3
- Multiple Packet Simulation
  - use more memory space to improve speed
- Selective or Demand-driven Fault-free Simulation
  - reduce unnecessary fault-free simulation as undetected faults become less

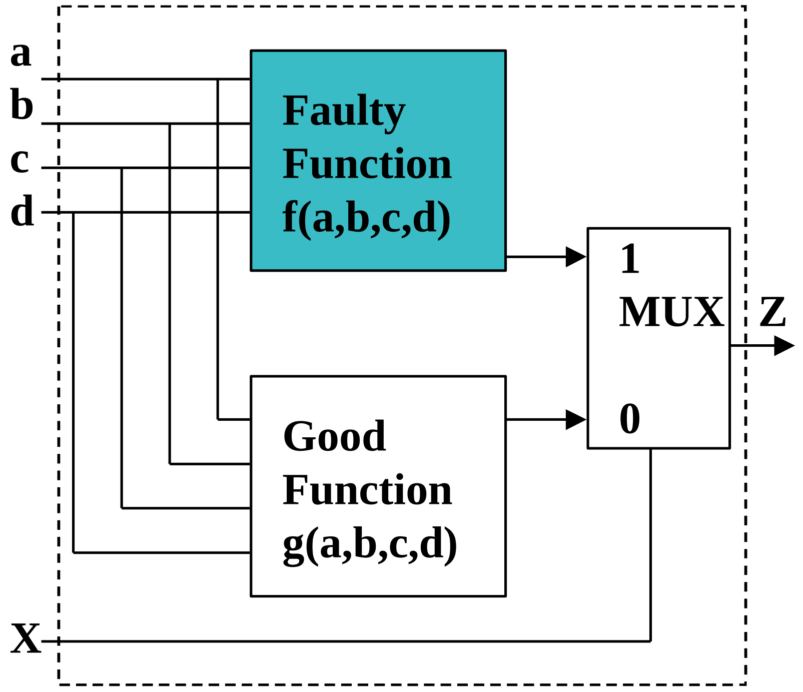
Hardware Approaches to Fault Simulation

- **Commercialized Dedicated Accelerators:**
  - **IKOS & ZYCAD**
- **Other Hardware Ideas**
  - **Associative Memory**
  - **Cellular Automata**
  - **Field-Programmable Gate Array**

Cellular Automata

- The cellular automata is a 2D array of very simple processors with interconnection to 4 immediate neighbors. It can be regarded as a massively parallel pipelined computer.
- [CA95] shows both gates and interconnects of a circuit can be mapped into the cellular automata and the CA can execute logic and fault simulation with additional comparison.
- Mainly restricted to combinational circuits.

Field Programmable Gate Array



A CLB with a dynamic fault injected (activated with x=1)

Hardware

- FPGA is a reconfigurable gate array which can be mapped from any logic circuits and emulated much faster than software simulation.
- The fault insertion process is slow. One way to minimize the insertion is to have both good gates and faulty gates within FPGA as shown left [FPGA95].