國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing

Chapter 3
Fault Simulation

Outline

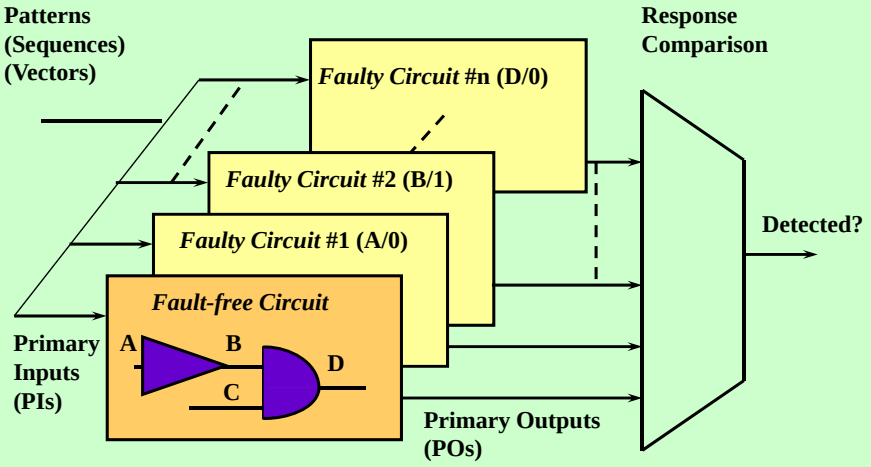
- Fault Simulation for Comb. Ckt
  - Basic of Logic Simulation
  - Parallel Fault Simulation
  - Deductive Fault Simulation
  - Concurrent Fault Simulation
- Approximation Approach
- Techniques for Sequential Circuits

Note: Comb. Ckt: Combinational Circuits

Why Fault Simulation ?

- To evaluate the quality of a test set
  - I.e., to compute its fault coverage
- Part of an ATPG program
  - A vector usually detects multiple faults
  - Fault simulation is used to compute the faults accidentally detected by a particular vector
- To construct fault-dictionary
  - For post-testing diagnosis
- To Evaluate the fault coverage of a functional patterns

Conceptual Fault Simulation

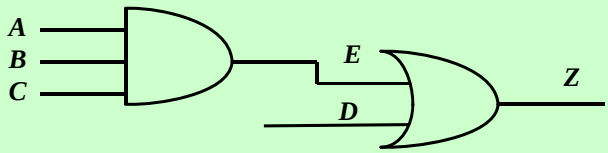


Logic simulation on both good (fault-free) and faulty circuits

Some Basics for Logic Simulation

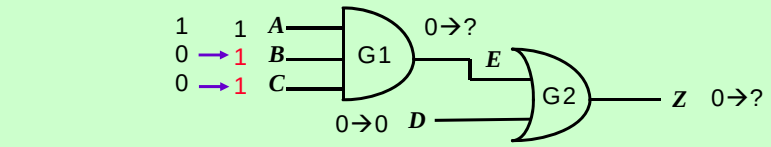
- For fault simulation purpose,
  - mostly the gate delay is assumed to be zero unless the delay faults are considered. Our main concern is the functional faults
- The logic values
  - can be either two (0, 1) or three values (0, 1, X)
- Two simulation mechanisms:
  - Oblivious compiled-code:
    - circuit is translated into a program and all gates are executed for each pattern. (may have redundant computation)
  - Interpretive event-driven:
    - Simulating a vector is viewed as a sequence of value-change events propagating from the PI's to the PO's
    - Only those logic gates affected by the events are re-evaluated

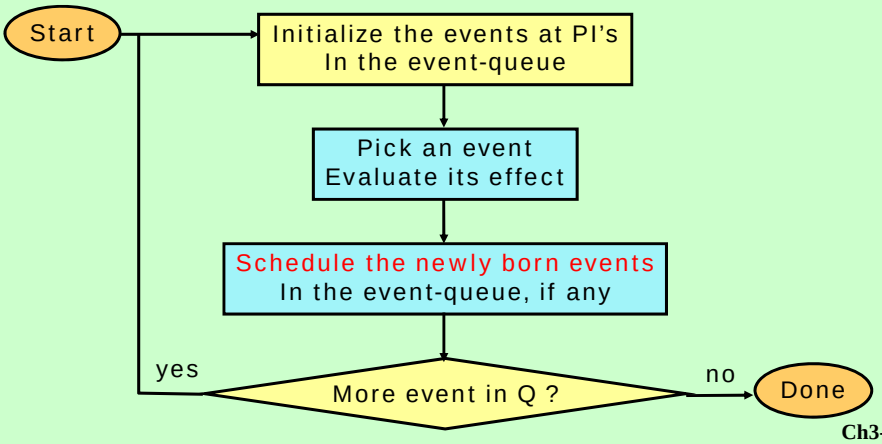
Compiled-Code Simulation



- Compiled code
  - LOAD *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* */
  - STORE *Z* /* store result of *Z* */

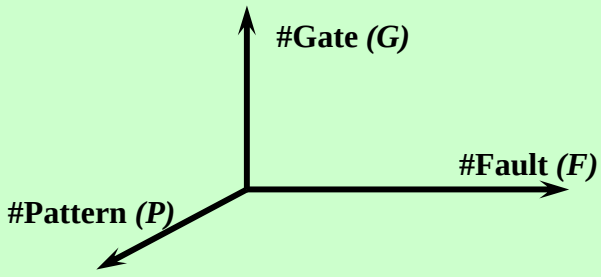
Event-Driven Simulation





Ch3-7

Complexity of Fault Simulation

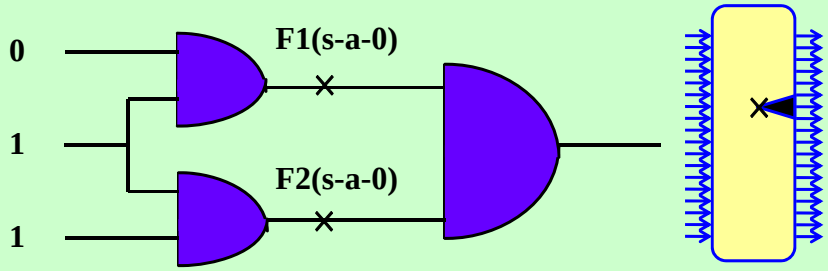


- Complexity ~ $F \cdot P \cdot G \sim O(G^3)$, where G is the no. of gates
- The complexity is higher than logic simulation by a factor of *F*, while usually is much lower than ATPG
- The complexity can be greatly reduced using
  - Fault dropping and other advanced techniques

Ch3-8

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.



Ch3-9

Fault Simulation Techniques

- Serial Fault Simulation
  - trivial single-fault single-pattern
- Parallel Fault Simulation
- Deductive Fault Simulation
- Concurrent Fault Simulation

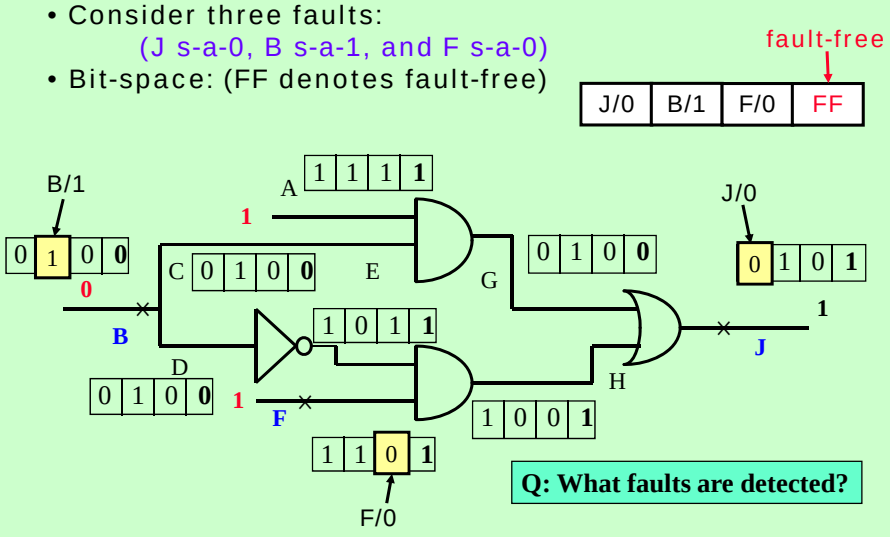
Ch3-10

Parallel Fault Simulation

- Simulate multiple circuits at a time:
  - The 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 simultaneously is limited by the word length, e.g., 32 circuits for a 32-bit computer
- Extra Cost:
  - 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 each pass

Example: Parallel Fault Simulation

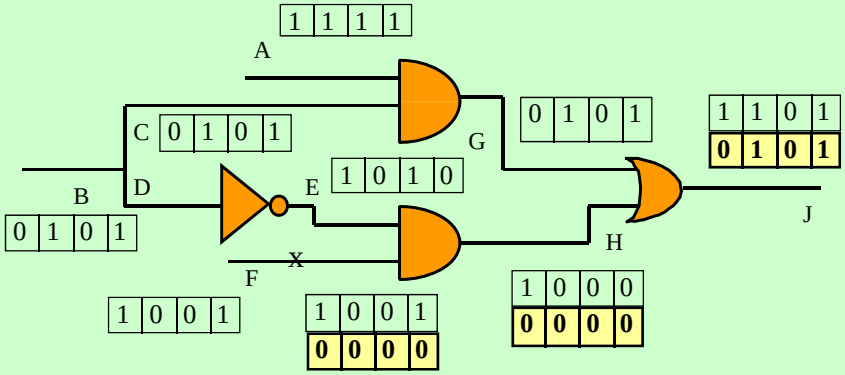
- Consider three faults:
  (J s-a-0, B s-a-1, and F s-a-0)
- Bit-space: (FF denotes fault-free)



Example: Parallel-Pattern Simulation

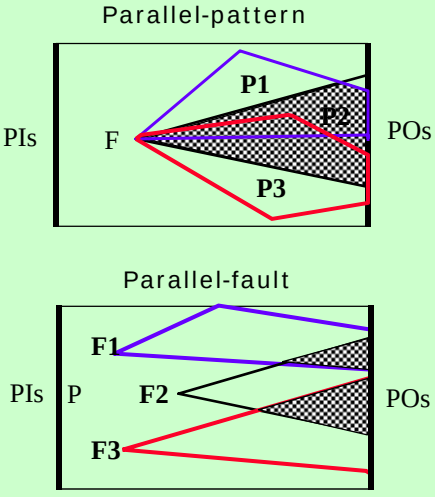
- Consider one fault F/0 and four patterns: P3,P2,P1,P0

Bit-Space: | P3 | P2 | P1 | P0 |



Ch3-13

Parallel-Pattern v.s. Parallel-Fault



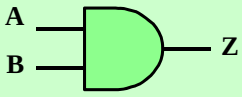
- P1, P2, P3 are patterns events
- F1, F2, F3 are faults
- Complexity
  - Is proportional to the events that need to be processed
  - The value-change events (upper figure) seems to be fewer than the fault-events (lower figure)
  - Hence, parallel-pattern seems to be more efficient than parallel-fault methods

Ch3-14

Deductive Fault Simulation

- Simulate all faulty circuits in one pass
  - For each pattern, sweep the circuit from PI's to PO's.
  - During the process, a list of faults is associated with each line
  - The list contains faults that would produce a fault effect on this line
  - The union fault list at every PO contains the detected faults by the simulated input vector
- Major operation: fault list propagation
  - Related to the gate types and values
  - The size of the list may grow dynamically, leading to a potential memory explosion problem

Controlling Value of a Logic Gate



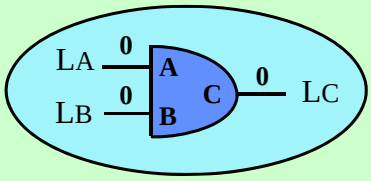
Whenever there is a '0' in the inputs, Z will be '0'
➔ Controlling value for NAND gate is '0'
➔ Non-Controlling value is '1'

Gate Type	Controlling Value	Non-Controlling Value
AND	'0'	'1'
OR	'1'	'0'
NAND	'0'	'1'
NOR	'1'	'0'

Example: Fault List Propagation

Fault-free simulation results: {A=0, B=0, C=0}
Q: What is the detected fault list at line C?

(Reasoning) To create a fault effect at line C, we need {A=1, B=1}
➔ which means that we need a fault effect at A as well as B
➔ It can be achieved in faulty circuits LA · LB
➔ Also C/1 is a new fault to be included in the fault list of C



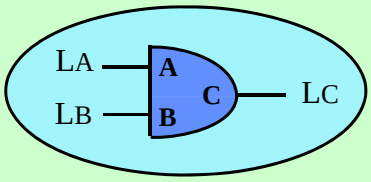
LA, LB, LC are fault list propagated to their respective lines

$\overline{L_A}$ is the set of all faults not in LA

Example: Fault List Propagation

LA, LB, LC are detected fault list at their respective lines

Consider a two-input AND-gate:



Non-controlling case: **Case 1:** A=1, B=1, C=1 at fault-free,
$L_C = L_A + L_B + \{C/0\}$

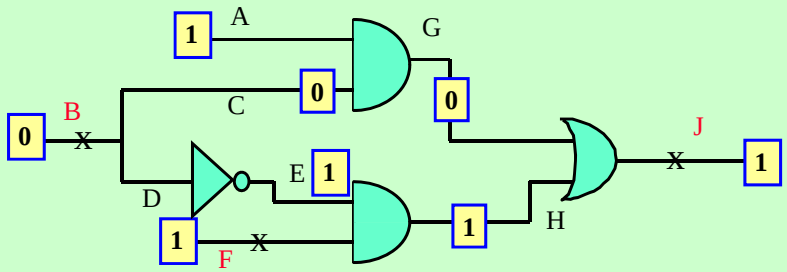
Controlling cases: **Case 2:** A=1, B=0, C=0 at fault-free,
$L_C = \overline{L_A} \cdot L_B + \{C/1\}$

Case 3: A=0, B=0, C=0 at fault-free,
$L_C = L_A \cdot L_B + \{C/1\}$

$\overline{L_A}$ is the set of all faults not in LA

Example: Deductive Simulation (1)

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

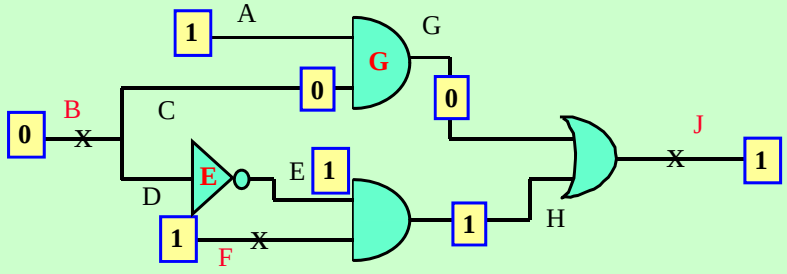


Fault List at PI's:

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

Example: Deductive Simulation (2)

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

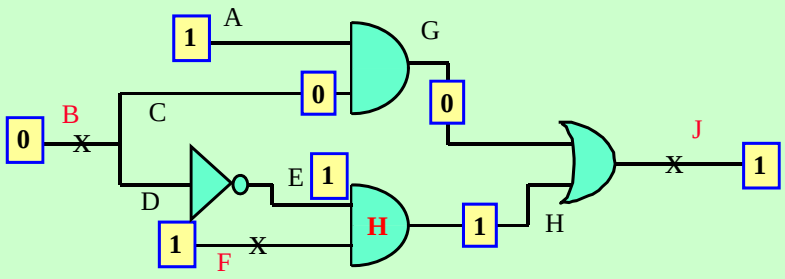


Fault Lists at G and E:

L_B = {B/1}, L_F = {F/0}, L_A = ϕ, L_C=L_D = {B/1},
L_G = ($\overline{L_A}$ * L_C) = {B/1}
L_E = (L_D) = {B/1}

Example: Deductive Simulation (3)

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

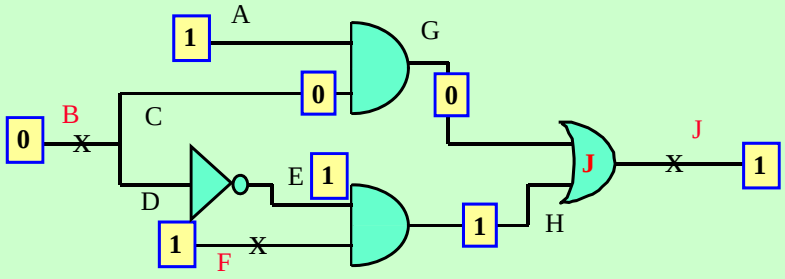


Computed Fault List at H:

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

Example: Deductive Simulation (4)

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

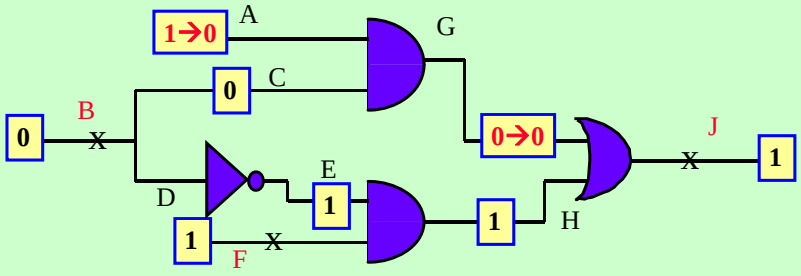


Final Fault List at the output J:

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},
$L_J = (\overline{L_G} \cdot L_H)$ {F/0, J/0}

Example: Even-Driven Deductive Fault Simulation

• When A changes from 1 to 0



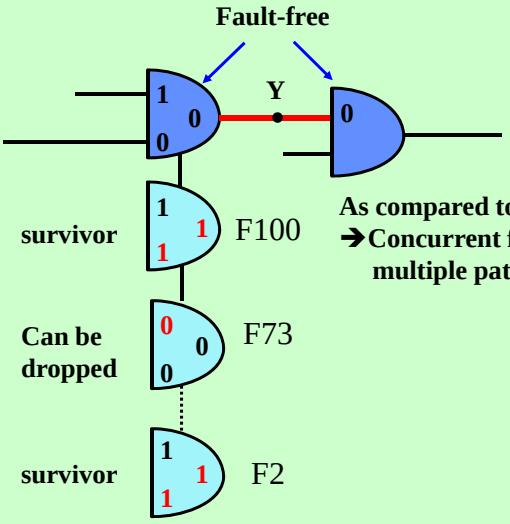
Event-driven operation:

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

Concurrent Fault Simulation

- Simulate all faulty circuits in one pass:
  - 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.
- Theoretically,
  - all faults in a circuit can be processed in one pass
- Practically,
  - memory explosion problem may restrict the number of faults that can be processed in each pass

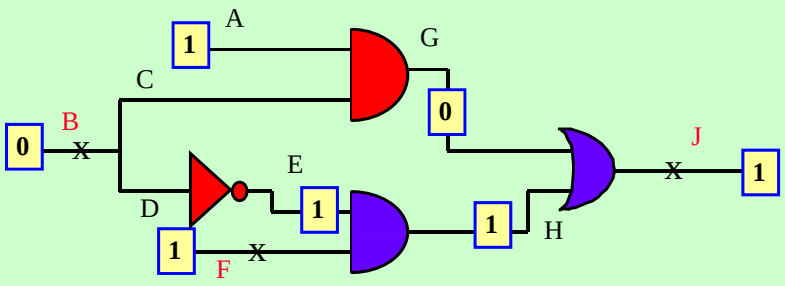
Concurrent Fault Simulation



As compared to deductive fault simulation
➔Concurrent fault simulation can process multiple patterns in a single run of simulation

Example: Concurrent Simulation (1)

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

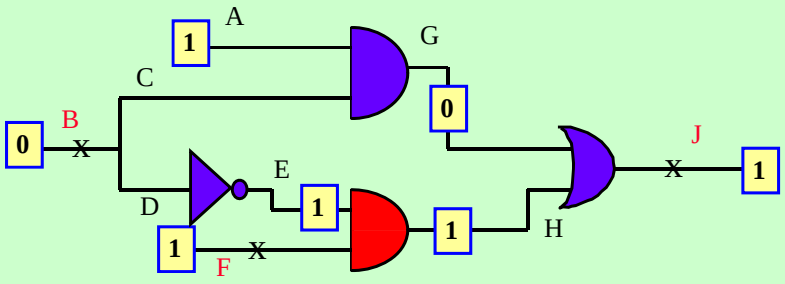


L_G = {10_0, B/1:11_1} **L_E = {0_1, B/1:1_0}**

Fault Free **A fault B/1**

Example: Concurrent Simulation (2)

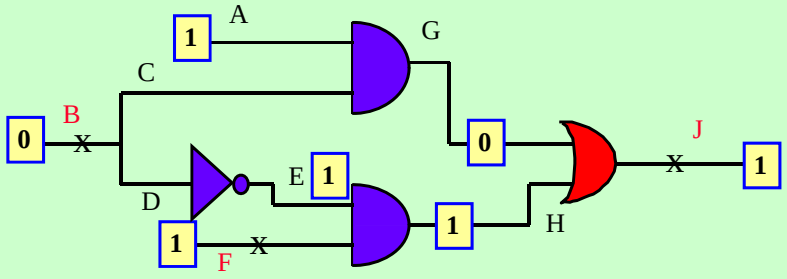
• Consider 3 faults: B/1, F/0, and J/0



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}

Example: Concurrent Simulation (3)

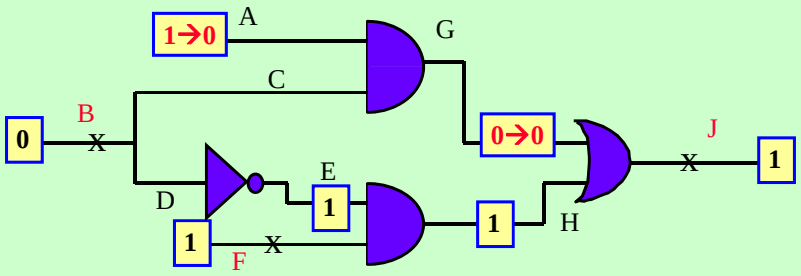
• Consider 3 faults: B/1, F/0, and J/0



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}
dropped

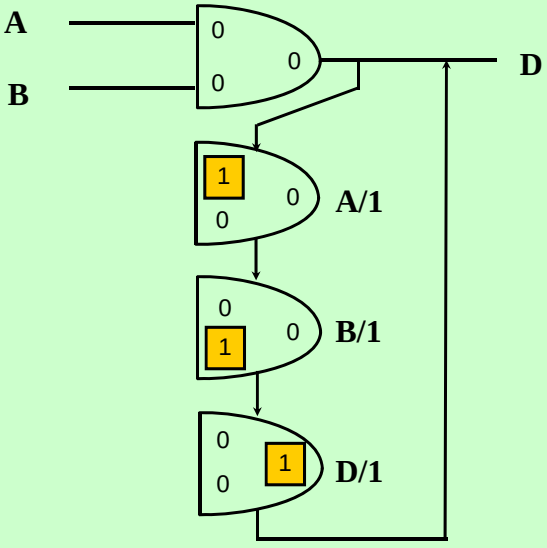
Example: Concurrent Simulation (4)

- When A changes from 1 to 0

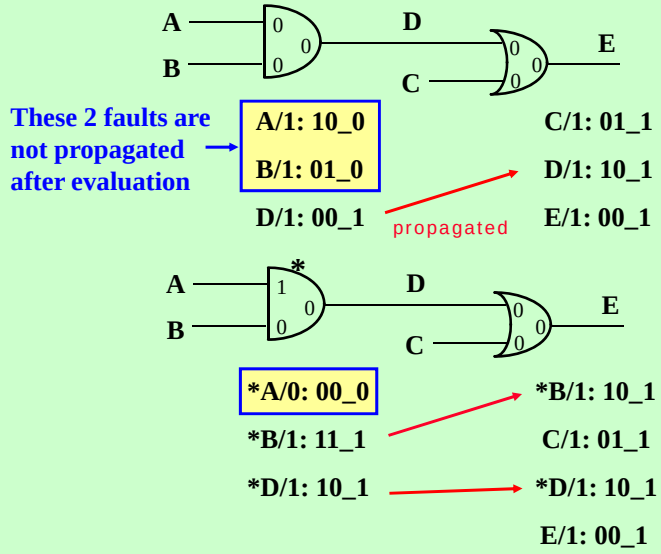


L_G = {00_0, B/1:01_0} L_E = {0_1, B/1:1_0}
L_H = {11_1, B/1:01_0, F/0:10_0}
L_J = {01_1, B/1:00_0, F/0:00_0, J/0:01_0}

Fault List Including New Borns



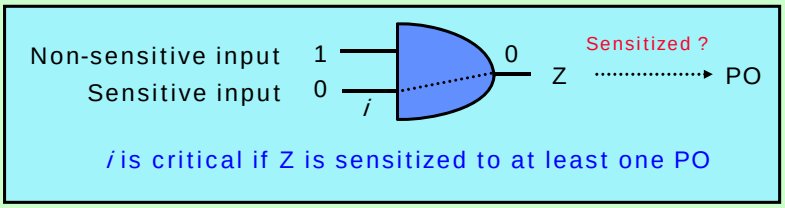
Fault List Propagation



Outline

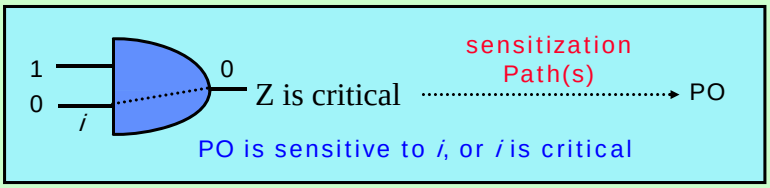
- Fault Simulation for Comb. Circuits
- Approximation Approach
  - Critical Path Tracing
  - Probabilistic Approach
- Techniques for Sequential Circuits

Sensitive Input and Critical Path



- Sensitive Input of a gate:
  - A gate input *i* is *sensitive* if complementing the value of *i* changes the value of the gate output
- Critical line
  - Assume that the fault-free value of *w* is *v* in response to *t*
  - A line *w* is critical w.r.t. a pattern *t* iff *t* detects the fault *w* stuck-at $\overline{v}$
- Critical paths
  - Paths consisting of critical lines only

Basics of Critical Path Tracing



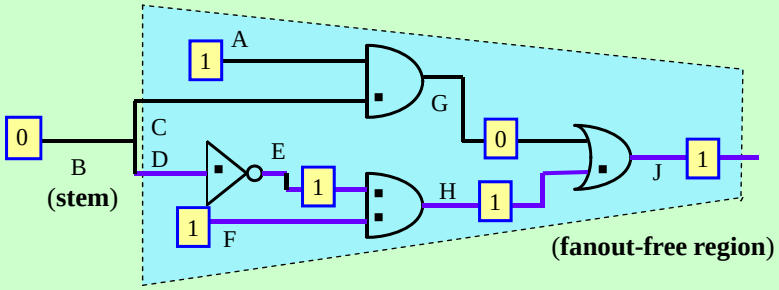
- A gate input *i* is critical w.r.t. a pattern *t* if
  - (1) the gate output is critical and
  - (2) *i* is a sensitive input to *t*
  - Use recursion to prove that *i* is also critical
- In a fanout-free circuit
  - the criticality of a line can be determined by backward traversal to the sensitive gate's inputs from PO's, in linear time

Analysis of Critical Path Tracing

- Three-step Procedure:
  - Step 1: Fault-free simulation
  - Step 2: Mark the sensitive inputs of each gate
  - Step 3: Identification of the critical lines by backward critical path tracing)
- Complexity is O(G)
  - Where G is the gate count
  - for fanout-free circuits --- very rare in practice
- Application
  - Applied to fanout-free regions, while stem faults are still simulated by parallel-pattern fault simulator.

Example of Critical Path Tracing

▪ sensitive input, — **critical line**

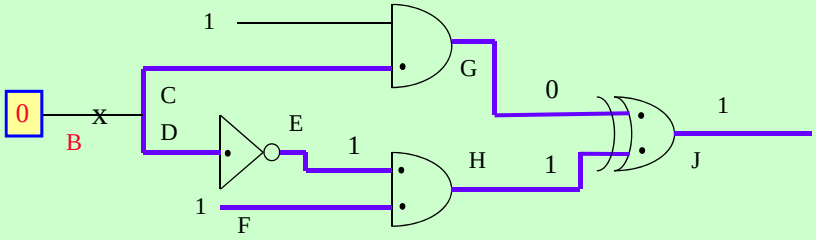


Detected faults in the fanout-free region: {J/0, H/0, F/0, E/0, D/1}

Question: is B stuck-at-1 detected ?

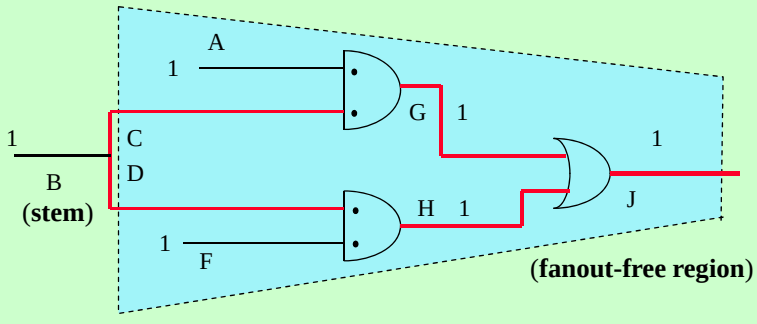
Anomaly of Critical Path Tracing

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



- **It turns out that B/1 is not detectable even though both C and D are critical, because their effects cancel out each other at gate J, (i.e., fault masking problem)**
- **There is also a so-called multiple path sensitization problem.**

Multiple Path Sensitization



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

Parallel and Distributed Simulation

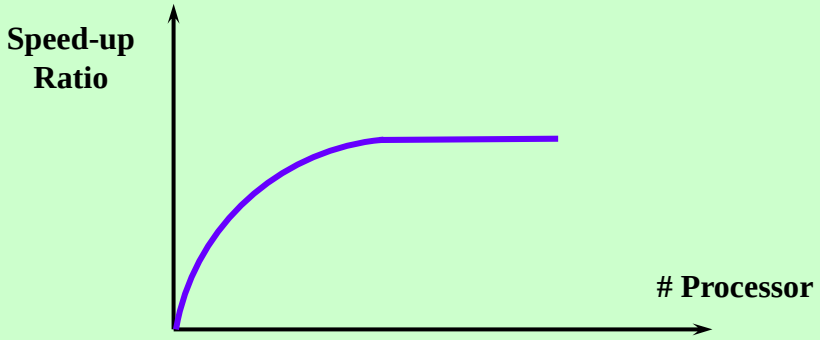
- To share the fault simulation effort
  - by a number of processors either tightly connected as in parallel computation or loosely connected as in distributed computation.
- The speed-up
  - with respect to the processor number depends on the degree of duplicated computation, and the communication overhead among processors.
- The distributed simulation
  - on a cluster of networked workstations is especially appealing.

Distributed Simulation Techniques

- Fault Partition
  - Distributes faults among many processors.
  - Works relatively well for both combinational and sequential circuits.
- Pattern Partition
  - Distributes patterns among processors.
    - no duplicated logic simulation
  - Works well for combinational circuits.
- Circuit Partition
  - Difficult to achieve synchronization without incurring excessive communication overhead.

Distributed Fault Simulation

• Typical Speed-up versus No. of Processors



• Diminished increase of speed-up ratio with more processors

Ch3-41

Fault Grading

- Approximate fault coverage
  - Can be obtained in much shorter computational time than regular fault simulation.
  - Not suitable for high fault-coverage requirement.
- Typical fault grading methods:
  - Toggle test, e.g. DATAS
  - Detection probability computation, e.g. STAFAN
  - Fault sampling
    - estimate from a selected subset of total faults
  - Test set sampling
    - estimate from a subset of complete test sequence

Ch3-42

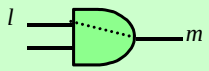
STAFAN

- Compute fault detection probability from logic simulation.
  - d_l = detection probability of s-a-0 on l = $C1(l)O(l)$
  - d_l = detection probability of s-a-1 on l = $C0(l)O(l)$

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

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

$$O(l) = S(l)O(m)$$



- m is the immediate successor of *l*
- observability can be computed backwards from POs

STAFAN (cont.)

$$d_f^n = 1-(1-d_f)^n \quad n \text{ is the no. of vectors}$$

Statistical Fault Coverage $= \dfrac{\sum_{\Phi} d_f^n}{|\Phi|}$ ← **the summation of each fault's detection probability**

Φ is the set of faults of interest

- More sophisticated than toggle test with same computation complexity

Outline

- Fault Simulation for Comb. Circuits
- Approximation Approach
  - Toggle Counting
  - Critical Path Tracing
  - Probabilistic Approach
- Techniques for Sequential Circuits

Fault Grading for Functional Input Sequence

Inputs:

(1) A test application program
(2) A sequential design

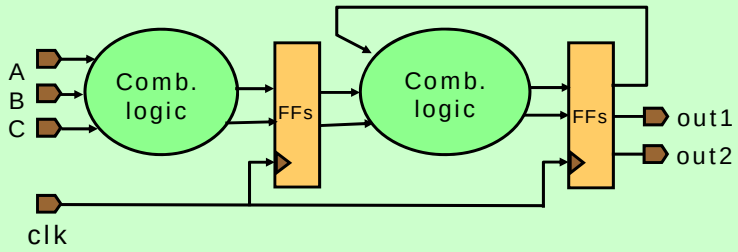
Output: The fault coverage

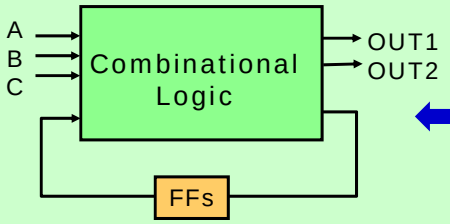
Application: High-Performance CPU Designs

Major challenge: often too time-consuming

Sequential Design Model

Sequential Circuits



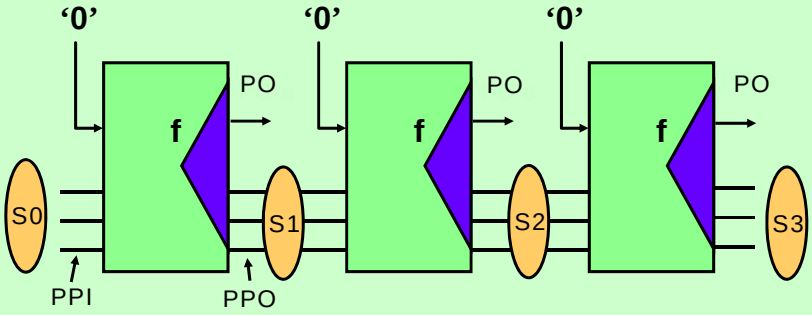


Hoffman Model

Ch3-47

Time-Frame-Expansion Model

Ex: Input Sequence ('0', '0', '0')
State Sequence (S0 → S1 → S2 → S3)



Notations: PPI: pseudo primary inputs (I.e., outputs of flip-flops)
PPO: pseudo primary outputs (I.e., inputs of flip-flops)

A single fault becomes multiple faults in the time-frame-expansion model

Ch3-48

Hypertrophic Faults

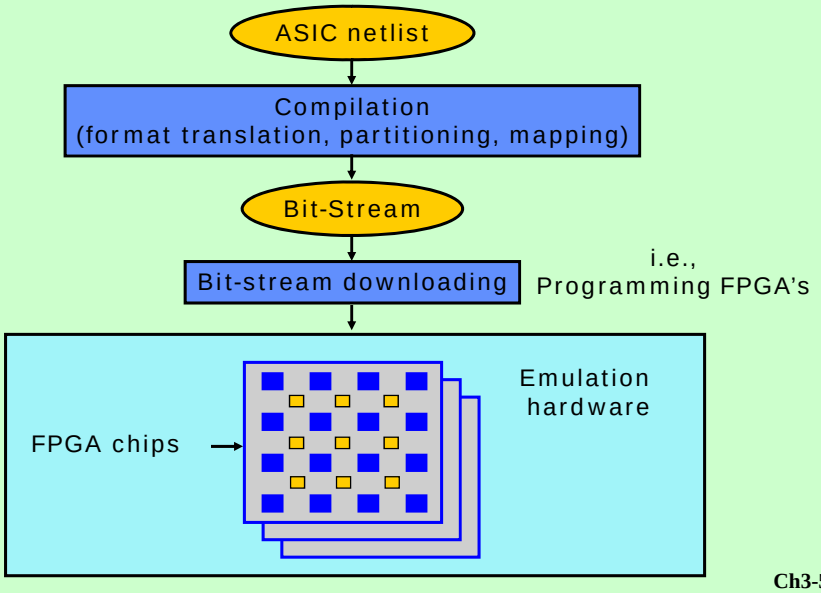
- A hypertrophic fault
  - Is a fault that diverges from the fault-free circuit with a large number of Xs, which usually is a stuck-at fault occurring at a control line and thus prevents the circuit initialization
- A small number of hypertrophic faults
  - account for a large percentage of fault events and CPU time
- These faults are sometimes dropped
  - as potentially detected faults to reduce simulation time. However, the resultant fault coverage then becomes approximate

A potentially detected fault is
a fault detected only when the circuit is
powered on in certain states, not every state.

Fault Emulation

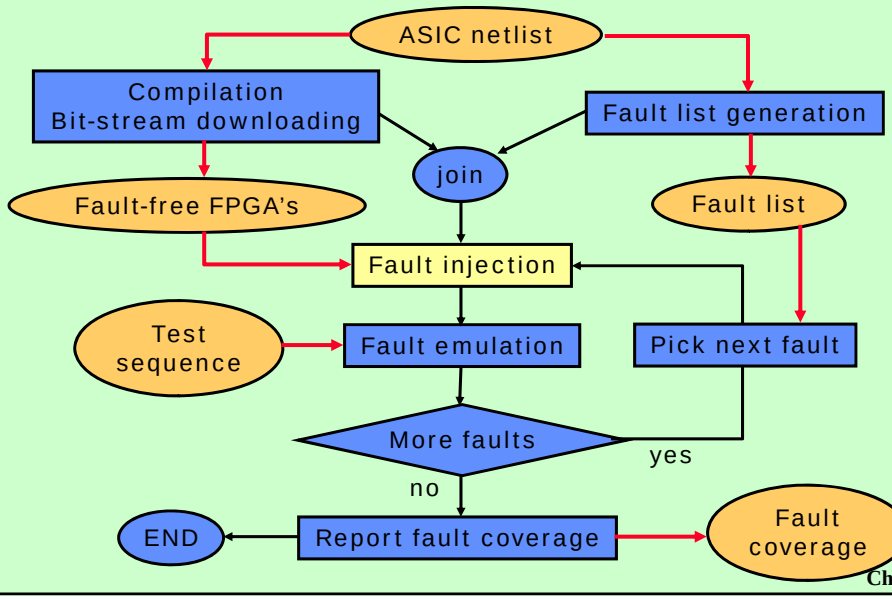
We can utilize FPGA to speed up the sequential fault grading

FPGA-Based Emulation Process



Ch3-51

Serial Fault Emulation by FPGA's



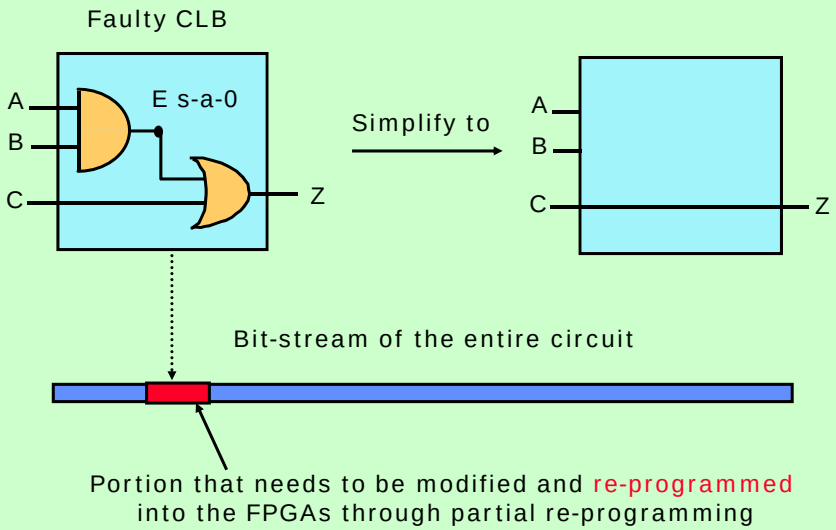
Ch3-52

Fault Injection Should Be Efficient !

- Fault Injection
  - Is to convert a fault-free FPGA implementation to a faulty one
  - If not efficient, could become the new bottleneck
- (1) Static Fault Injection
  - Directly changes the configuration of the fault-free implementation to a faulty one
- (2) Dynamic Fault Injection
  - Do not change the configuration directly
  - Fault inject is injected through the control of some hardware originally built-in to the netlist

Ch3-53

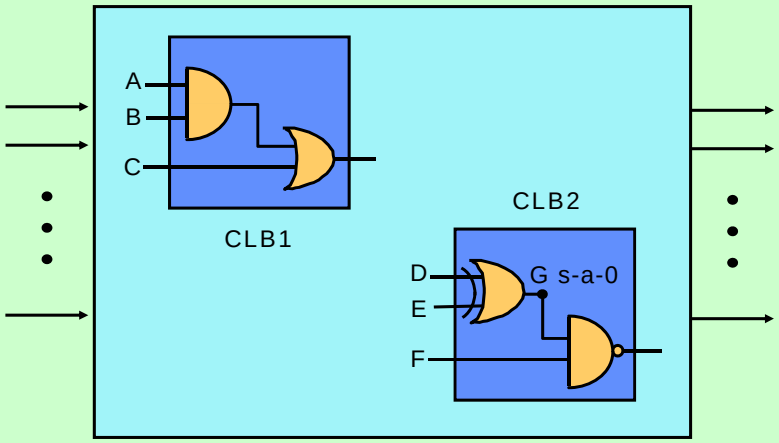
Static Fault Injection



Ch3-54

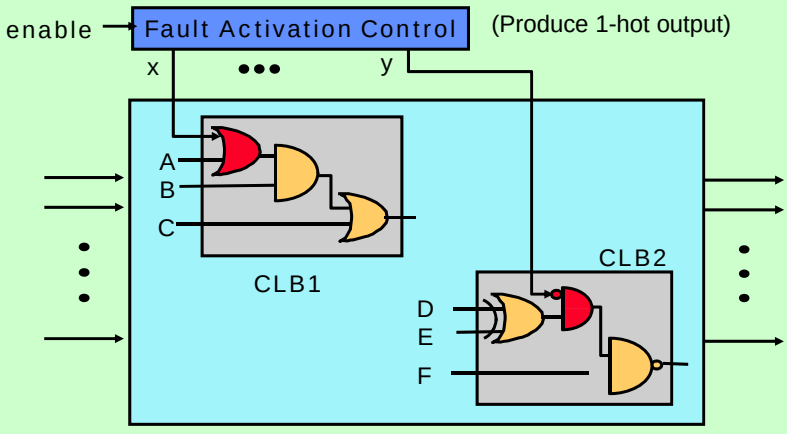
Example: FPGA-implementation

Two faults are being considered:
A stuck-at 1
G stuck-at-0



Ch3-55

Dynamic Fault Injection (I)

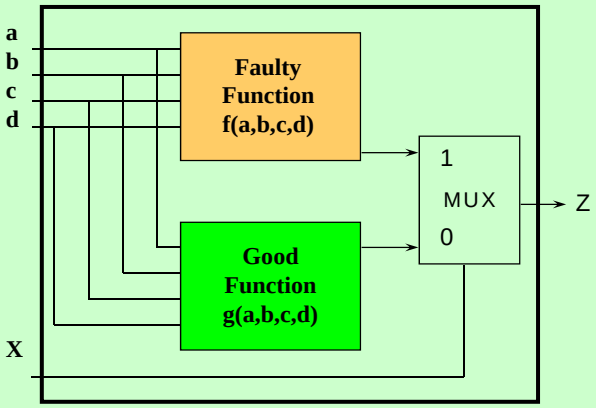


(x=1, y=0) → The above netlist behaves like A s-a-1 faulty circuit
(x=0, y=1) → The above netlist behaves like G s-a-0 faulty circuit

Ch3-56

Dynamic Fault Injection (II)

(1) Conservatively map only 4-input function to a CLB,
which is originally assumed to be capable of realizing 5-input function.
(2) Extra input, I.e., x, is reserved for the control of dynamic fault injection.



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

Overview of Dynamic Fault Injection (II)

In the following configuration:
5 faults are injected (one for each column), but only 1 is activated