Chapter 4 Fault Simulation

錯誤模擬

Outline

- Introduction to Fault Simulation
- Fault Simulation for Combinational Circuits
- Techniques for Sequential Circuits
- Fault Grading

What is Fault Simulation?

Given :

- **A circuit**
- **A test (sequence of test vectors)**
- **A fault model**

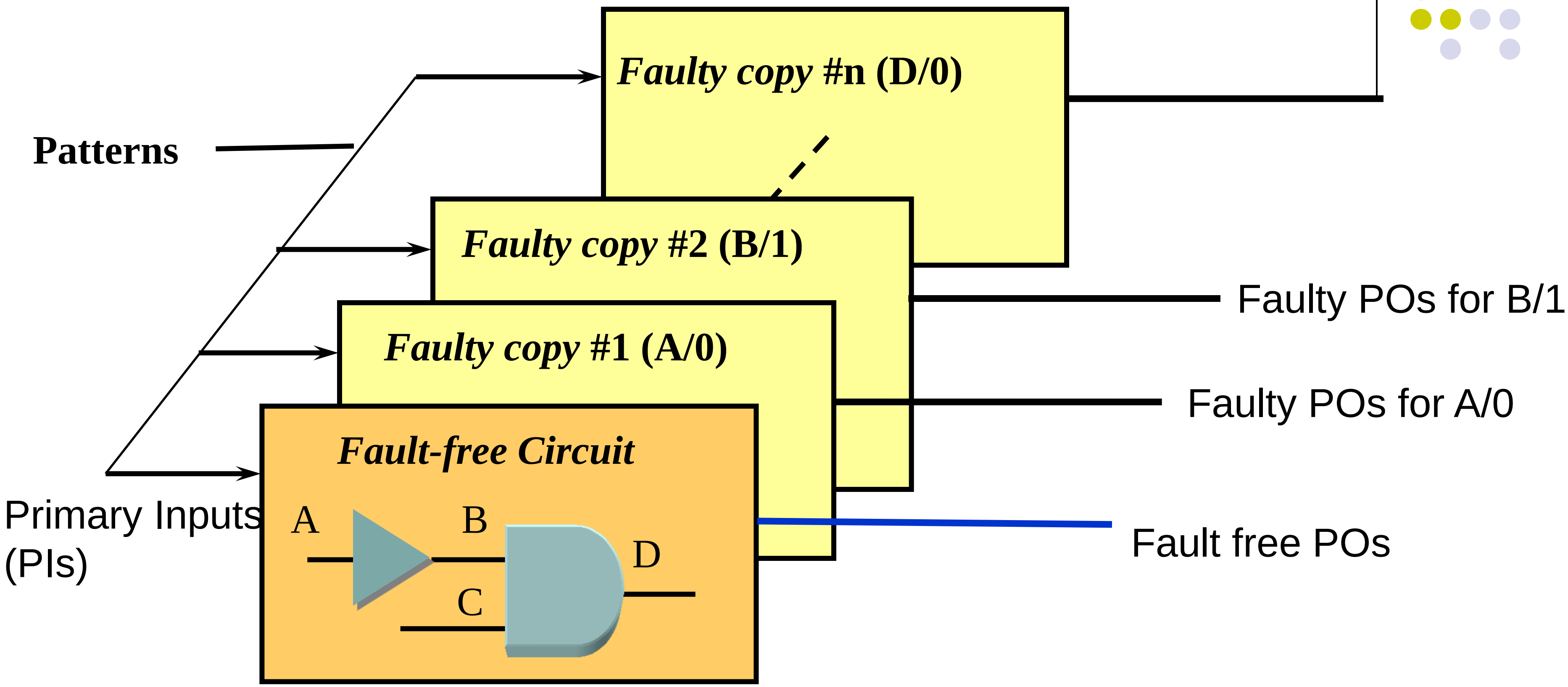
Determine:

- **Fault coverage (the fraction of modeled faults detected)**
- **Undetected faults**

Applications of Fault Simulation

- Determine fault coverage of a given test sequence
  - Measure test quality by fault coverage
- Determine undetected faults for test generation
- Construct fault dictionary for diagnosis
  - Use a fault simulator to compute & store the response for every fault
  - If the output response of the circuit under test matches any of the response in the fault dictionary, the fault location is identified

Conceptual Fault Simulation



- One logic simulation for every fault-free and faulty circuits
- Complexity ~ O(*F* x *P* x *G)*
  - *F: # of faults; P: # of patterns; G: # of gates*
- A simple technique: fault dropping.

Fault Simulation Techniques

- Parallel-Fault (Single-Pattern) Simulation
- Deductive Fault Simulation
- Concurrent Fault Simulation
- Parallel-Pattern Single-Fault Simulation
- Critical Path Tracing

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 (one for each fault) to be simultaneously processed is limited by the word length, e.g., 32 circuits for a 32-bit computer
- The fault-free logic simulation is repeated for each pass
  - This can be avoided by storing fault-free values in another word.

Speedup of Parallel Fault Simulation

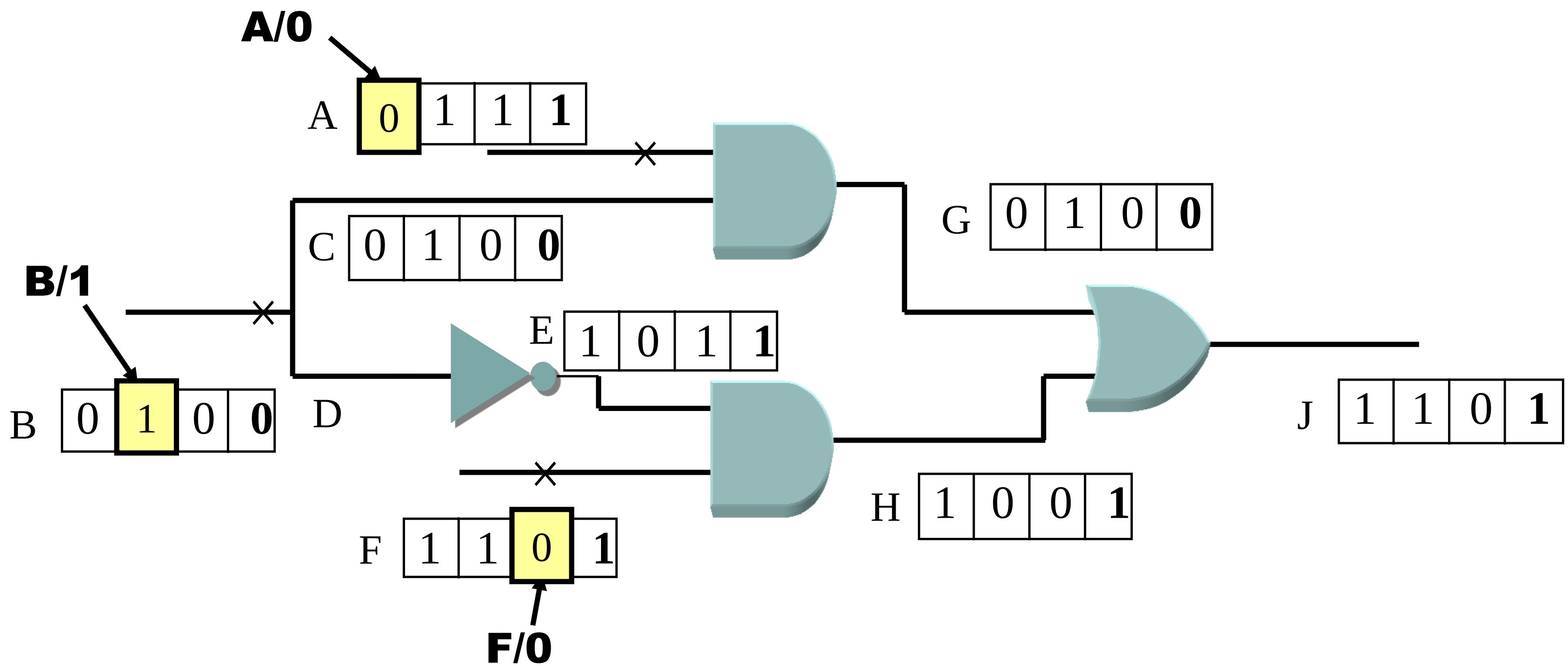
- There are extra costs associated with parallel fault simulation, so the speed up is only sub-linear.
  - An event, a value-change of a single fault or fault-free circuit leads to the computation of the entire word
  - A proper fault grouping is needed.
    - Group *similar* faults in the same simulation

Example: Parallel Fault Simulation

- Assume 4-bit computer words
- Consider three faults: J s-a-0, B s-a-1, and F s-a-0
- Bit allocation:

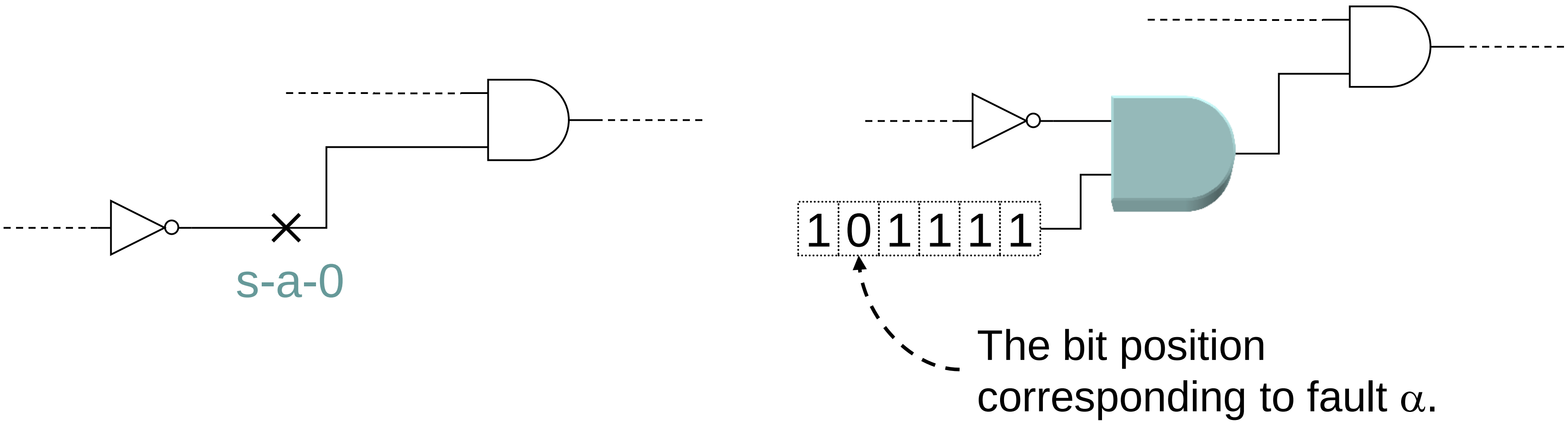
A/0	B/1	F/0	FF

← Fault free bit



Fault Injection by Circuit Modeling

- By changing only the circuits, we can use a parallel logic simulator to implement a parallel fault simulation.

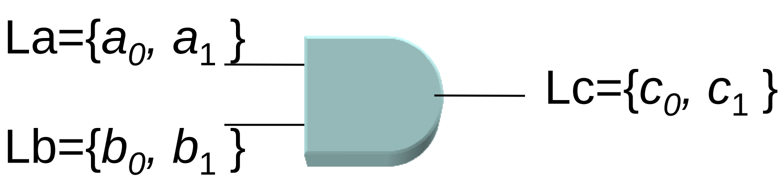


Parallel Fault Simulation – Summary

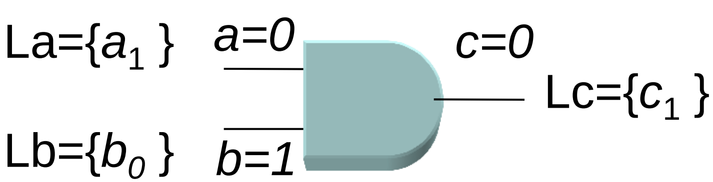
- Most effective for
  - Interconnected logic gate netlist
  - Stuck-at faults
  - Two-valued logic
- Little resource required
  - Easy to implement
  - Fixed and low memory requirement
- Some extra cost
  - Faults injected in the same word may not generate the same events.
  - The fault-free circuit is simulated in each pass.

Fault List in Deductive Fault Simulation

- Maintain a fault list of every signals in a logic set.
- A fault is in a fault list of each signal if
  - The fault changes the fault-free value of the signal, or
  - The fault is propagated to the signal



Fault list before simulation

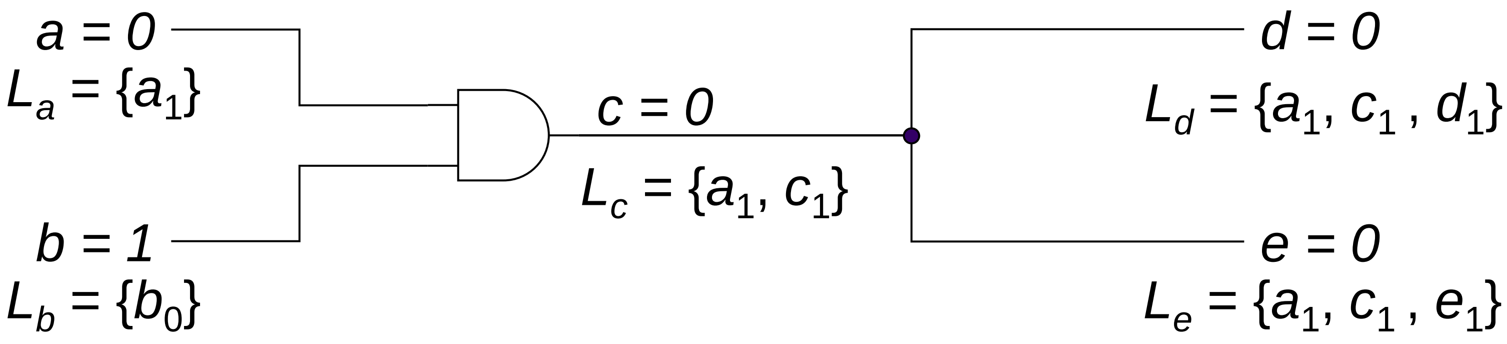


Fault list after logic simulation

Propagation of Fault List

- Create an initial fault list at every signal (inject collapsed faults)
- Propagate fault list starting from PIs.
- Propagate through gates level by level (logic deduction)
  - Propagation rules are set operations and depends only on gate type and its input values.
- A fault is detected if it appears in the fault list of a primary output.

An Example of Fault List Propagation



- Input $a = 0$, $b = 1$
- $a = 0 \rightarrow L_a = \{a_1\}$
  - a_1: a stuck-at-1
- b_0 in L_b is not propagated to L_c because $a = 0$ is the controlling value.

Rule of Fault List Propagation

	a	*b*	*z*	Output fault list
AND	0	0	0	$\{L_a \cap L_b\} \cup z_1$
	0	1	0	$\{L_a - L_b\} \cup z_1$
	1	0	0	$\{L_b - L_a\} \cup z_1$
	1	1	1	$\{L_a \cup L_b\} \cup z_0$
OR	0	0	0	$\{L_a \cup L_b\} \cup z_1$
	0	1	1	$\{L_b - L_a\} \cup z_0$
	1	0	1	$\{L_a - L_b\} \cup z_0$
	1	1	0	$\{L_a \cap L_b\} \cup z_0$
NOT	0		1	$L_a \cup z_0$
	1		0	$L_a \cup z_1$

Fault List Propagation Rule

- Notations:
  - Let *I* be the set of inputs of a gate *Z* with a controlling value *c* and inversion *i*

    (i.e., i is 0 for AND, OR and 1 for NAND, NOR)
  - Let *C* be the set of inputs with value *c*

Non-controlling case: **union**

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

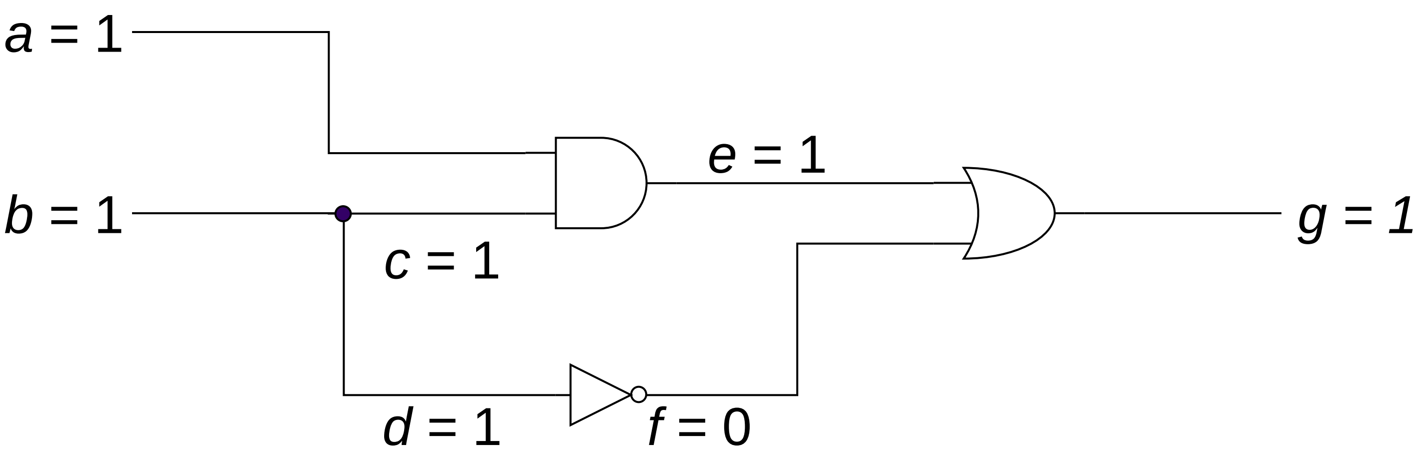
Controlling cases: **intersection**

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

A Generalized Propagation Rule – Cont'd

- If no input has value c, any fault effect on any 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.

An Example of Deductive Fault Simulation



$$L_a = \{a_0\}$$

$$L_b = \{b_0\}$$

$$L_c = \{b_0, c_0\}$$

$$L_d = \{b_0, d_0\}$$

$$L_e = L_a \cup L_c \cup \{e_0\} = \{a_0, b_0, c_0, e_0\}$$

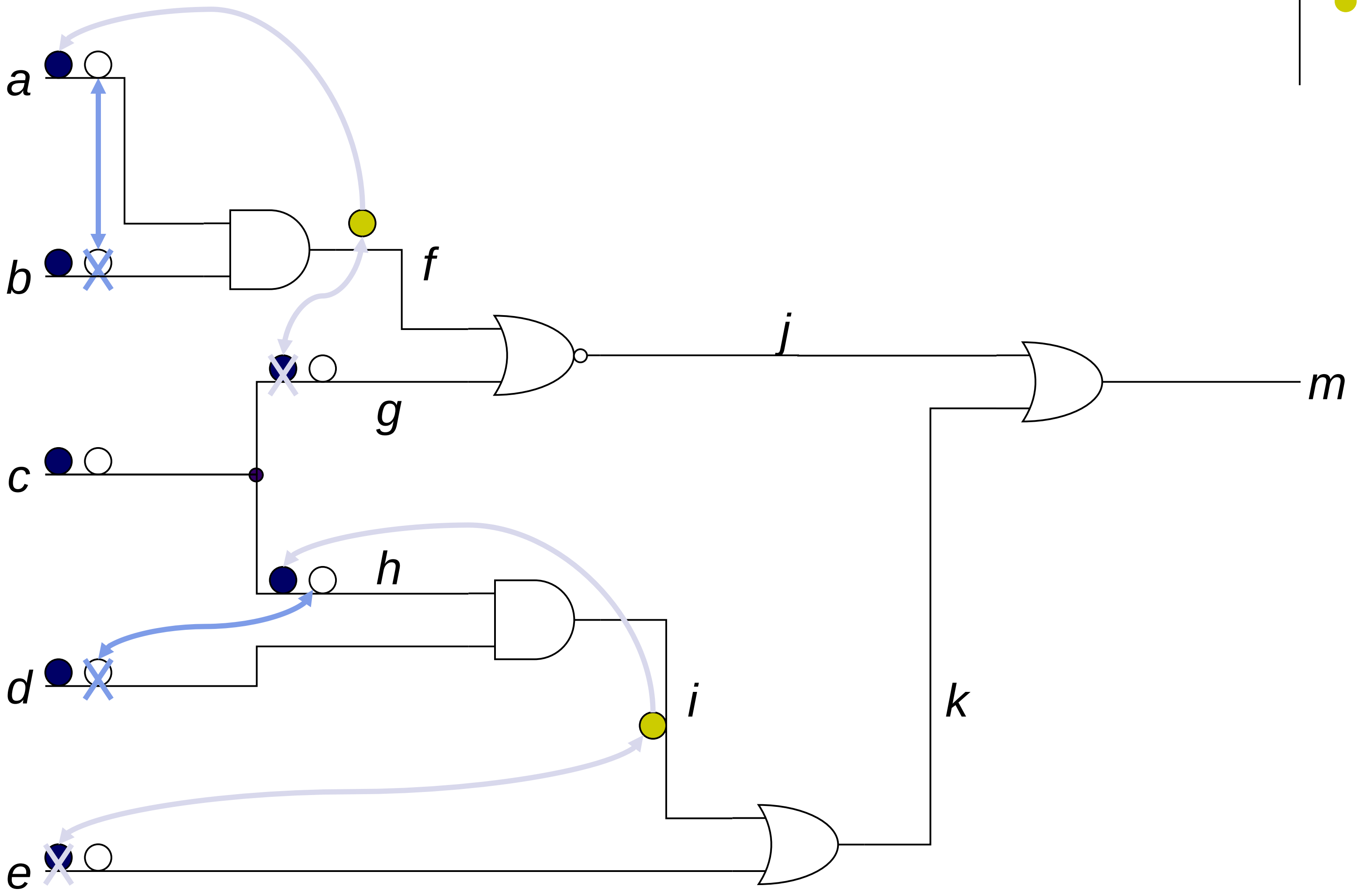
$$L_f = \{b_0, d_0, f_1\}$$

$$L_g = \{L_e - L_f\} \cup \{g_0\} = \{a_0, c_0, e_0, g_0\}$$

The Deductive Fault Simulation Algorithm

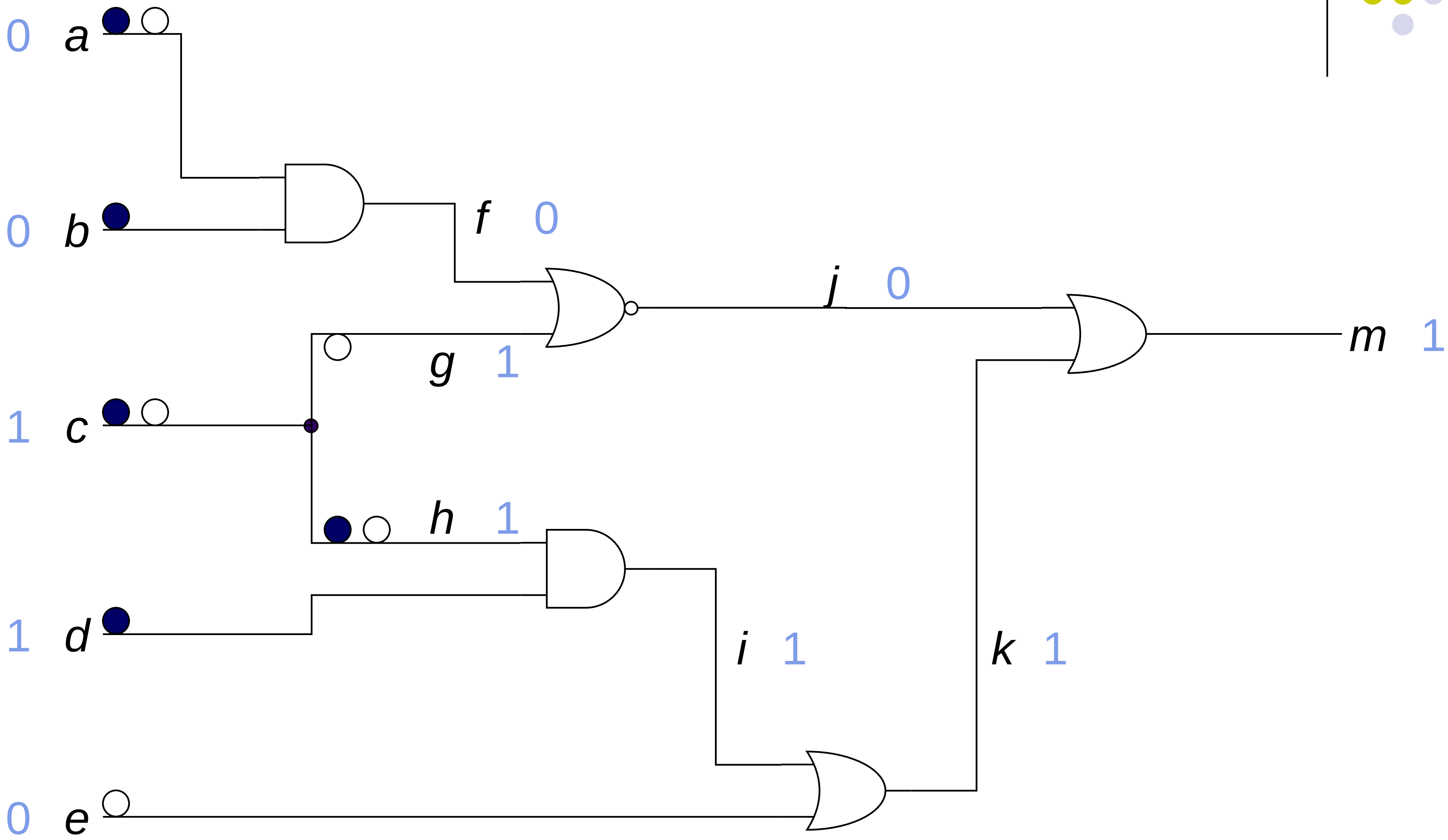
- Levelize the circuit from PIs to POs.
- Step 1: Read a test vector.
- Step 2: Perform logic simulation.
- Step 3: Perform fault list propagation.
- Step 4: Determine detected faults.
- Goto step 1

Fault Collapsing

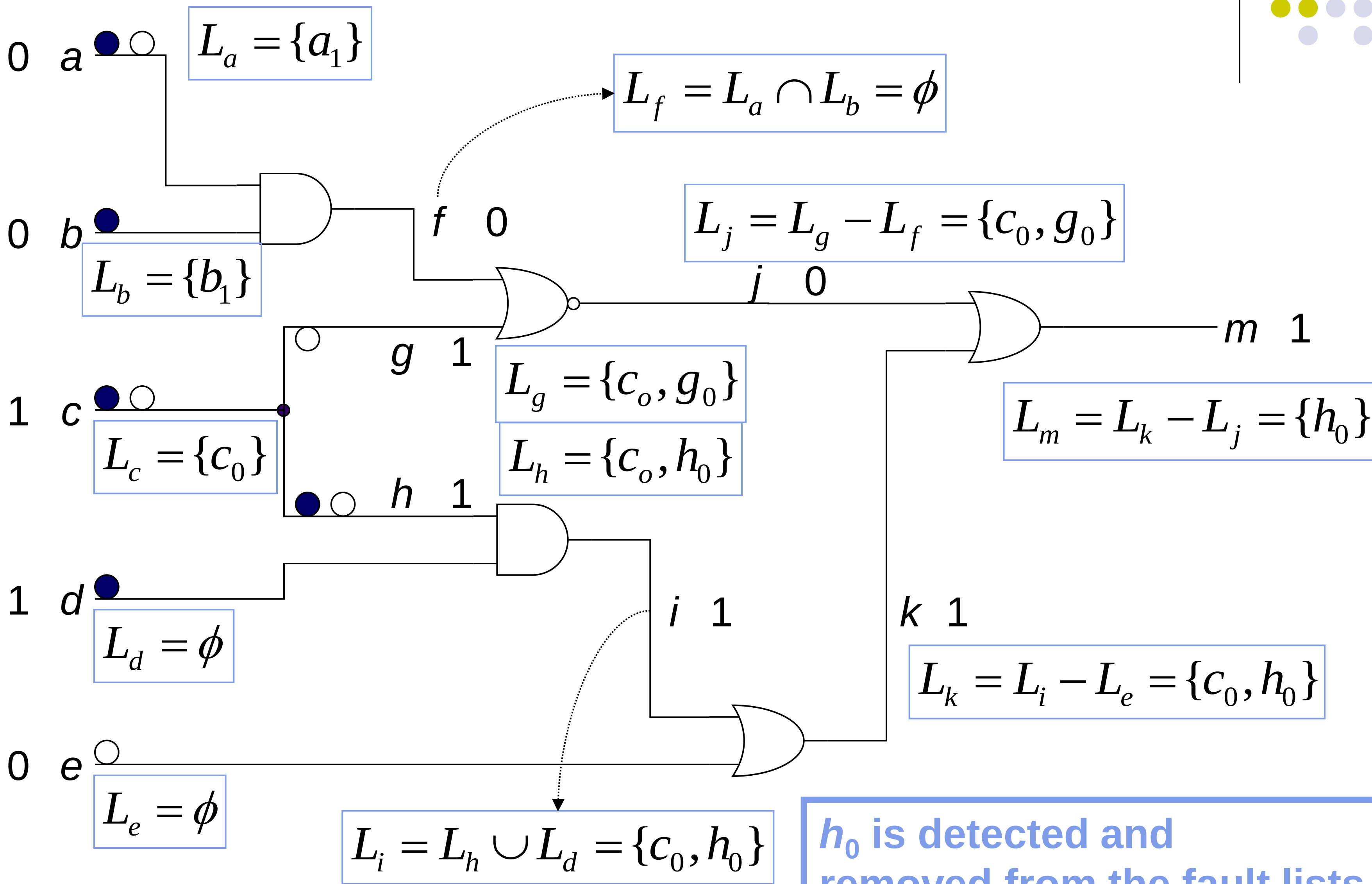


Collapsed fault list: $\{a_0, a_1, b_1, c_0, c_1, d_1, e_0, g_0, h_0, h_1\}$

Logic Simulation

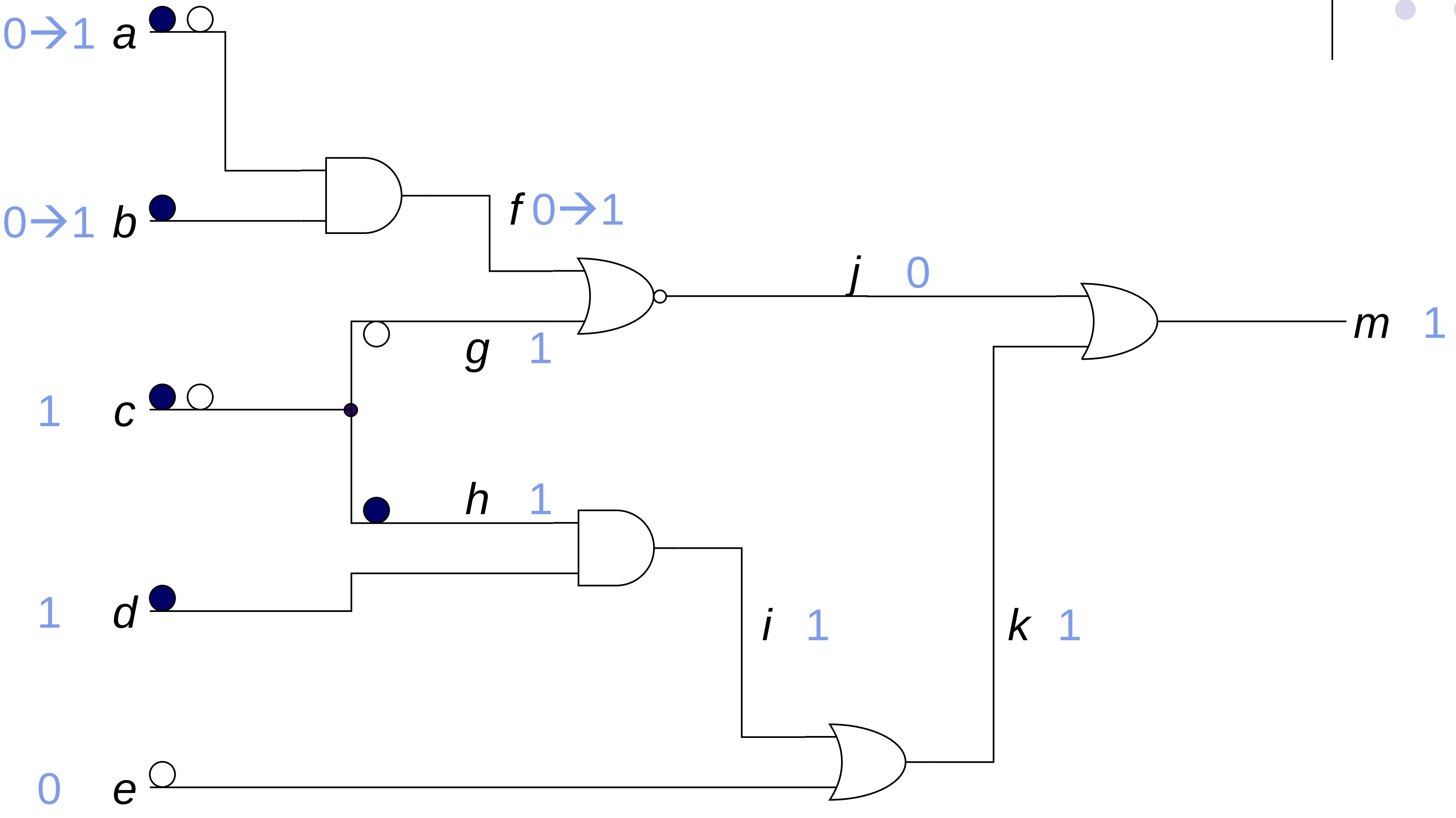


Fault List Propagation

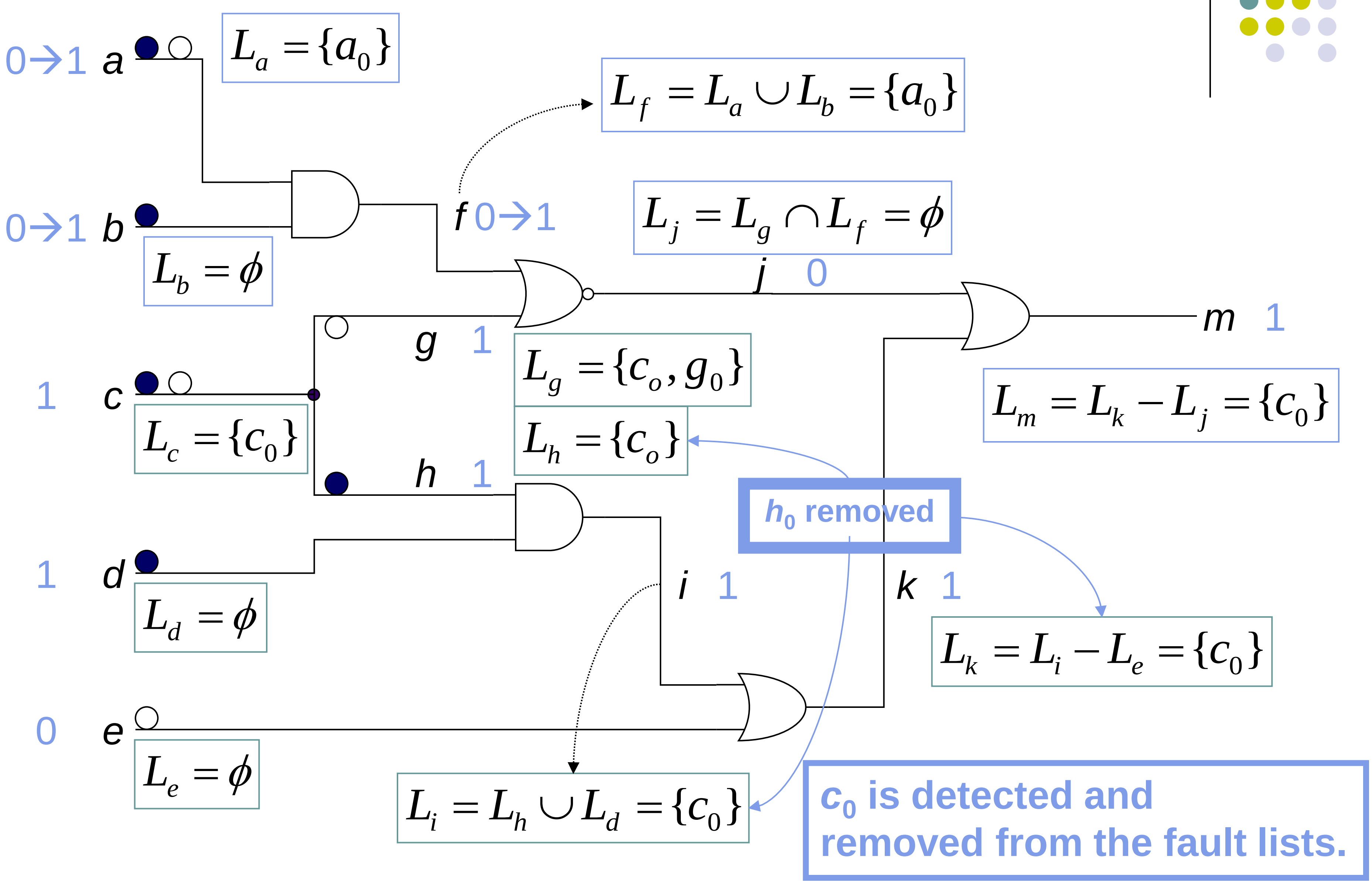


h_0 is detected and removed from the fault lists.

Apply the Next Vector



Fault List Propagation

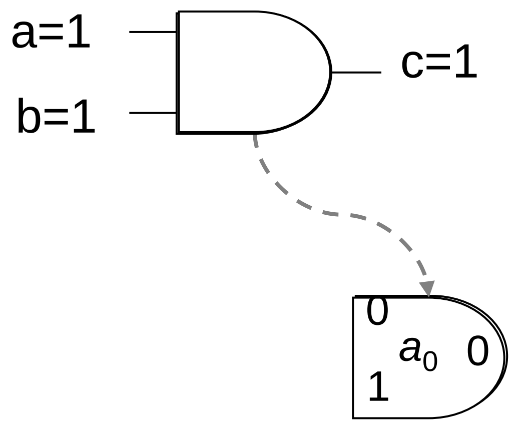


Faults in Concurrent Fault Simulation

- Maintain a fault list for every signal in a list of faulty gates.
  - Each faulty gate stores the status of a fault
    1. If a fault is activated (i.e., the signal is the fault site) or
    2. Propagated to this gate.

Faulty Gate Structure in Concurrent Fault Simulation

- Each element in the fault list contains
  - Fault index
    - For example, a0 in the following
  - Fault states: input and output values of an element



This faulty gate (for a stuck-at 0) changes a faulty input to 0 and output to 0.

Concurrent Fault Simulation

- An extension of event-driven simulation
  - Event-driven simulation for both fault free events and faulty events.
- Events in concurrent fault simulation
  - **Fault-free event**: (signal, value)
    - Generated by PI changes
    - Applied to both fault free and faulty circuits.
  - **Faulty event**: (fault index, signal, value)
    - Generated by faults
    - Applied only to the faulty circuit.

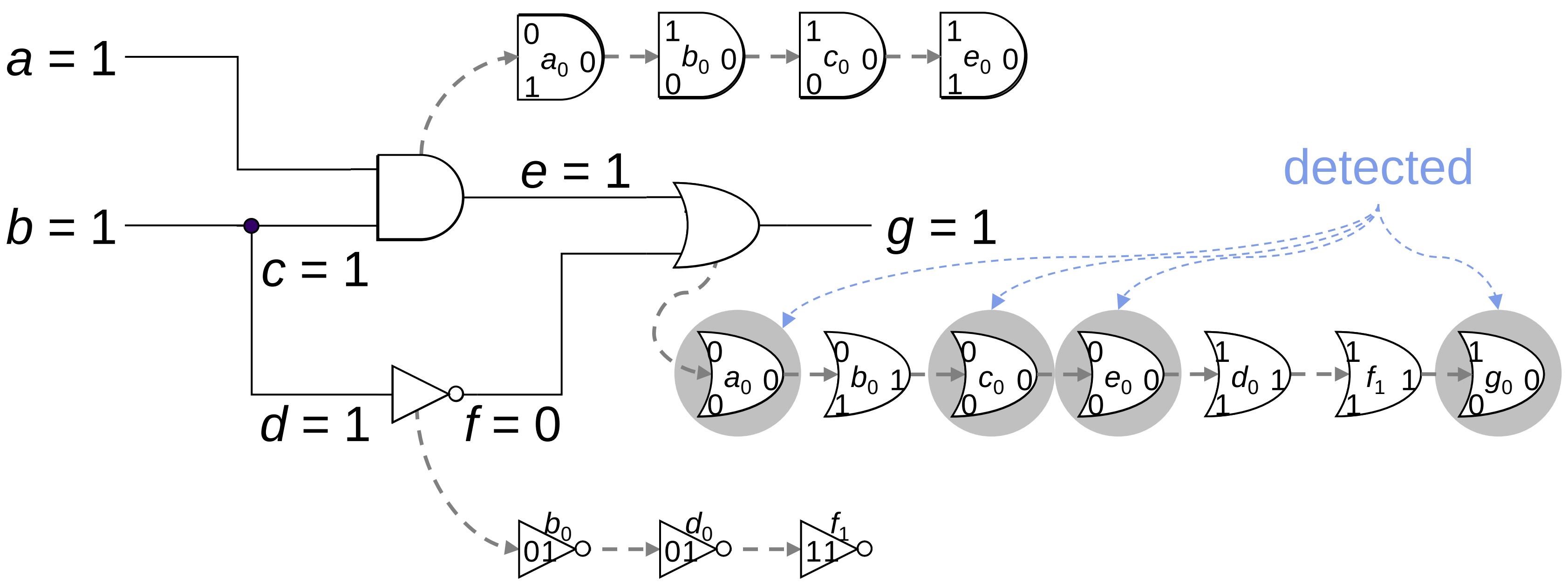
Maintain the Faulty List

- After fault-free and faulty event propagation, an element is in the list iff the I/O values of the faulty circuit are different from the fault-free values.
- Faulty gates with the same output values as true circuits are kept,
  - Fault can propagate through multiple paths: we might need to evaluate a faulty gate several times before we know no faulty effects pass
- A local fault (at inputs or outputs of a gate) will remain in the list even there is no difference of I/O values with faulty free circuits.
  - Note that dropped faults will be deleted.

Fault Dropping

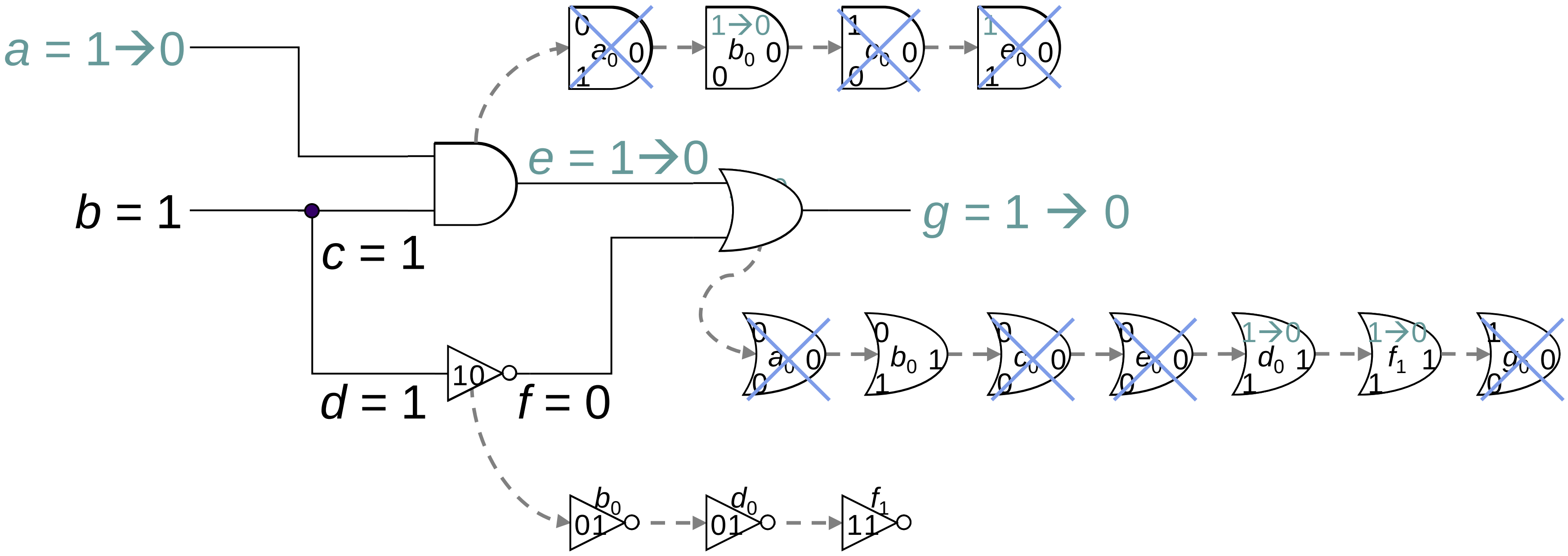
- We can hash every dropped faults, and process every faulty list to remove dropped faults.
- We can also remove dropped faults at fault site, and remove faulty gates if no fanin faulty list contains the fault.

Concurrent Fault Simulation – Example

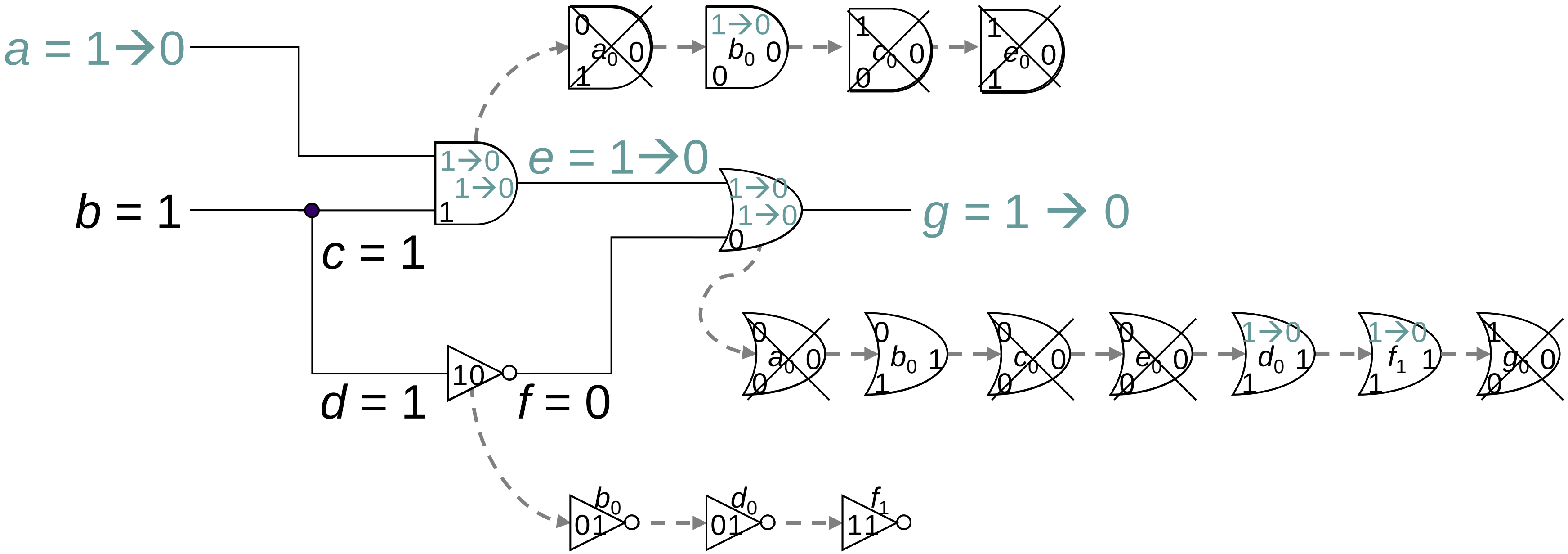


Concurrent Simulation - Convergence

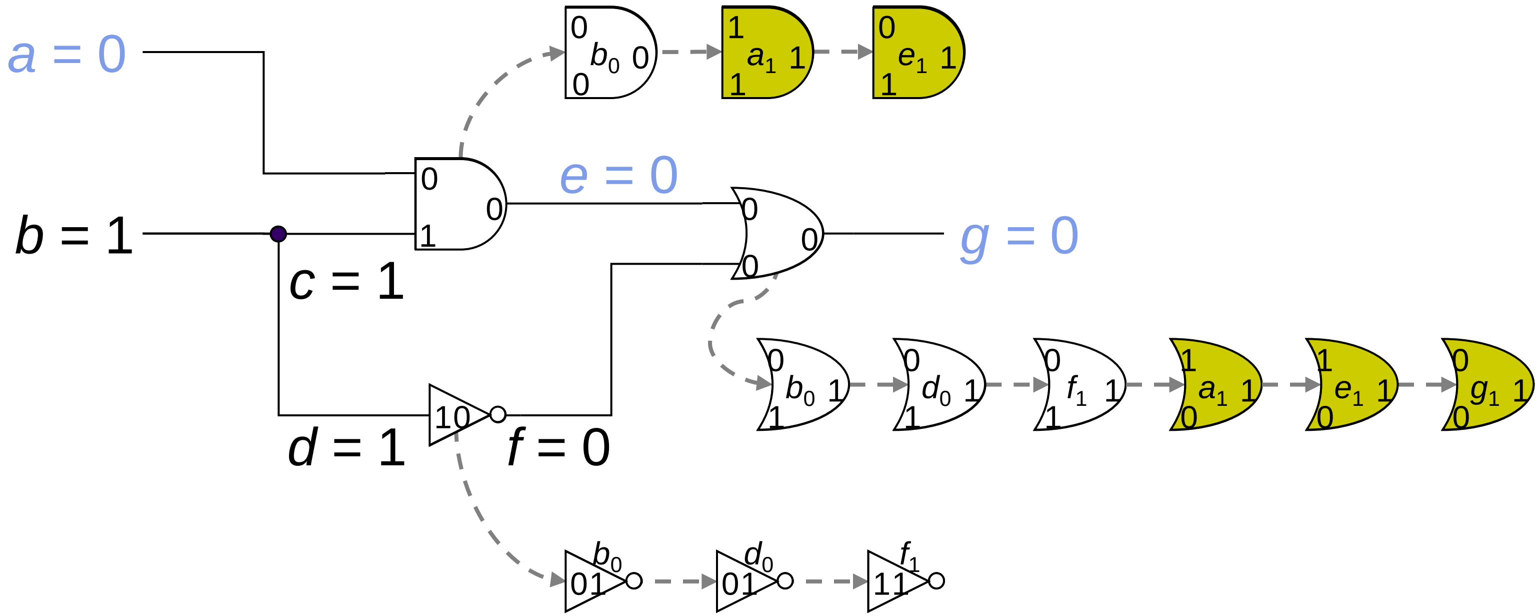
a_0, c_0 and e_0 converge into good gates.



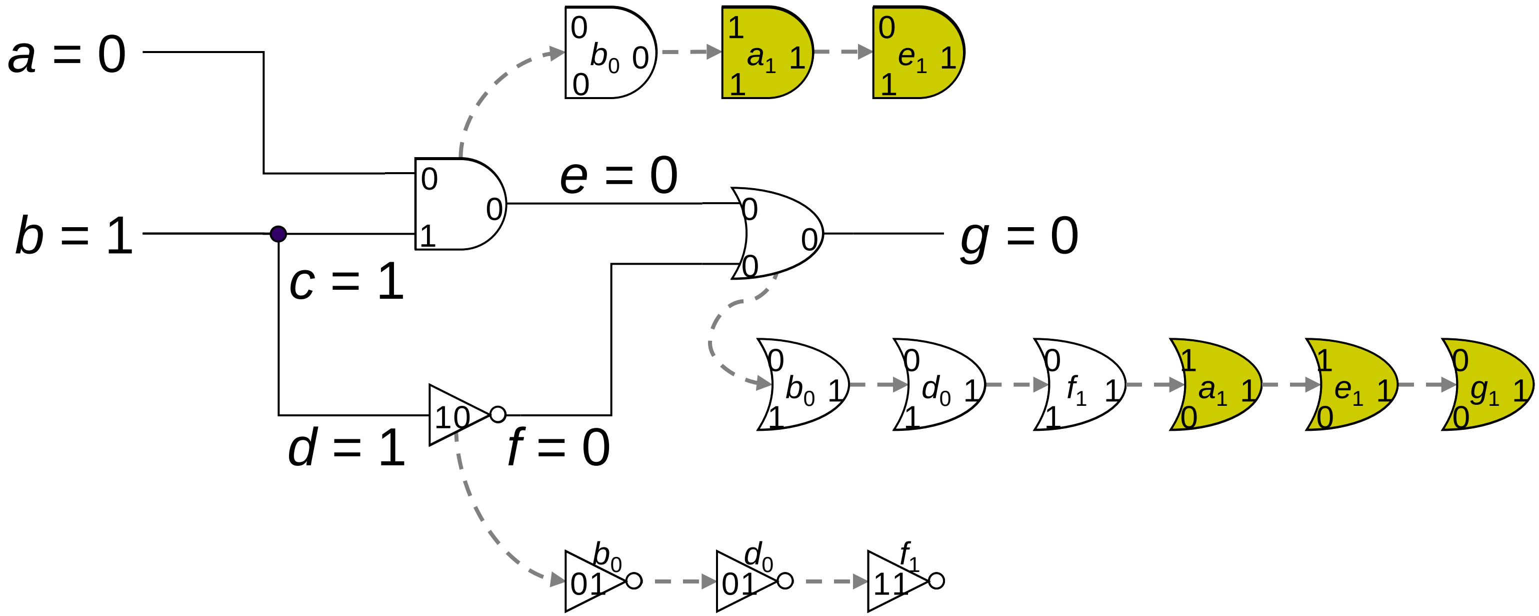
Concurrent Simulation - Convergence



Concurrent Simulation - Divergence



Concurrent Simulation - Divergence



Disadvantages of Deductive & Concurrent Fault Simulation

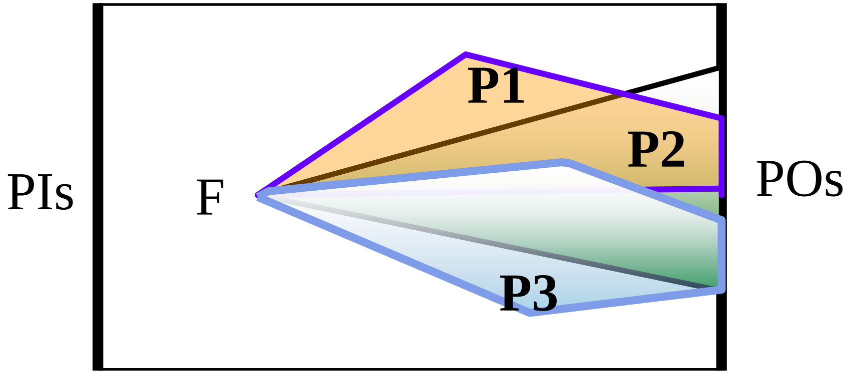
- Large memory to record the status of all circuits (fault-free & faulty).
  - Dynamic memory (linked lists)
- Evaluation overhead on the linked lists
  - Comparing fault lists
  - Removing faults
  - Adding faults
- Performance overhead in memory management
- However, concurrent fault simulation is very popular because it is flexible.
  - Different fault types, timing, etc.

Parallel-Pattern Single-Fault Propagation (PPSFP)

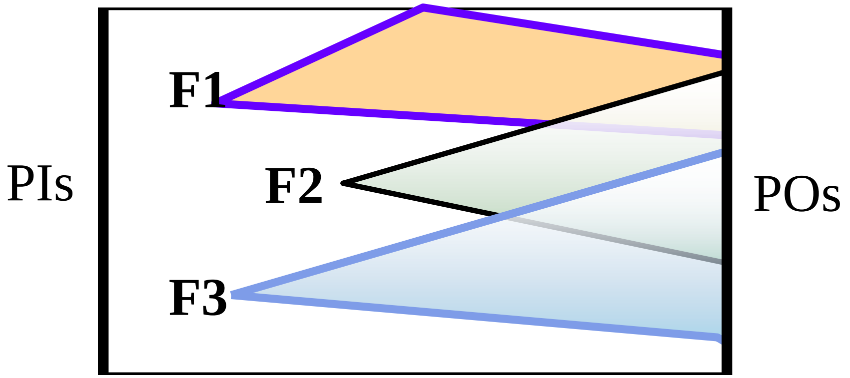
- Use multiple computer words to simulate multiple input patterns.
- It can be based on an event-driven simulation.
- Simple and Extremely Efficient
  - Basis of most modern combinational fault simulators
  - Note that this is mostly the case we use a fault simulator, since ATPG is also limited to combinational circuits.
- It is also applicable to sequential circuits.

Parallel-Pattern v.s. Parallel-Fault

Parallel-pattern
(P1, P2, P3 are events by patterns)



Parallel-fault
(F1, F2, F3 are events by faults)



- Parallel fault simulation will be more efficient if
  - Faulty events are generated and propagated to POs.
  - Otherwise, events are useless.
- In general, parallel faults generate different events, and they are usually not detected simultaneously by a single PI pattern.
- Parallel pattern simulation is more efficient.

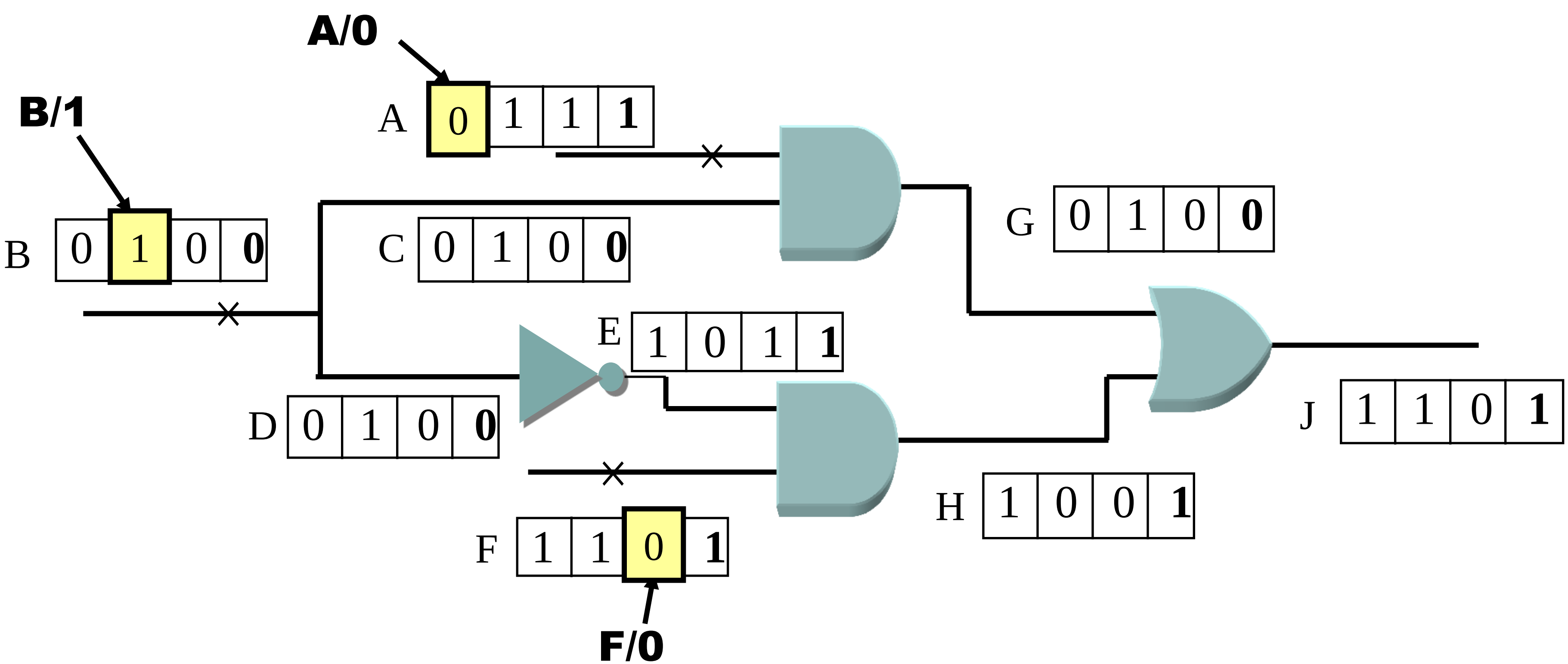
Re-visit Parallel Fault Simulation

- Bit allocation:

A/0	B/1	F/0	FF

← Fault free bit

- There are 9 events, but only one fault is detected!

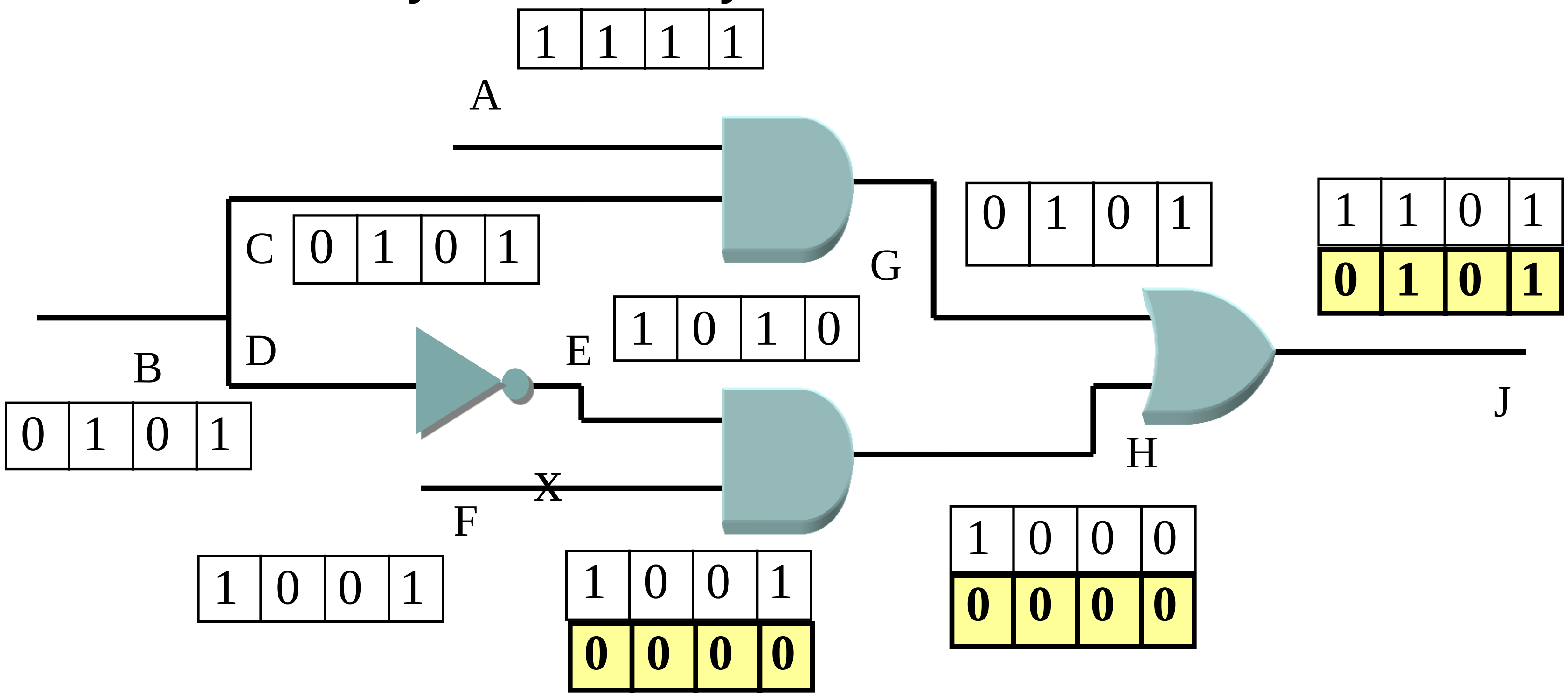


Example: Parallel Pattern Simulation

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

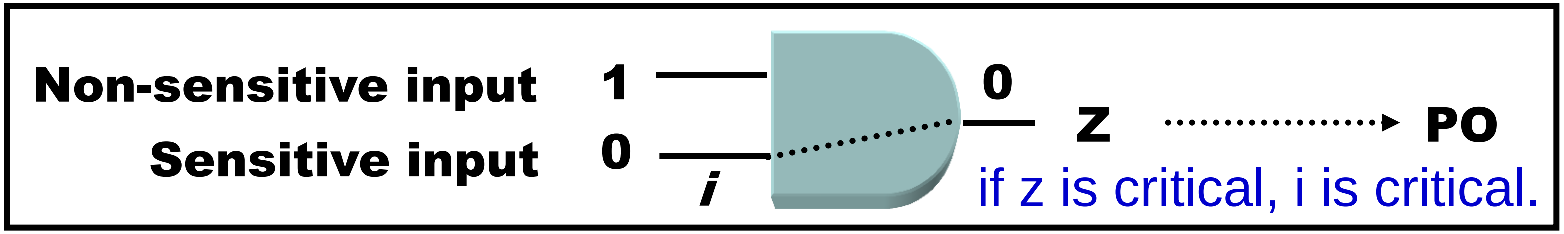
P3	P2	P1	P0

- There are only three faulty events!



Critical Path Tracing

- Find faults detected by a test without simulating all faults.
- The key is to find critical lines in the circuits by tracing from POs.
  - line *w* is critical for a pattern *t* iff *t* detects *w* stuck-at *v*
    - fault-free value of *w* is *v’* (under *t)*
- Any PO is critical.
- Find whether any inputs of gates to PO are critical recursively
  - If a line of a gate output is critical, the critical property can be transferred to its sensitive inputs.
  - A gate input *i is sensitive* if complementing the value of *i* changes the value of the gate output

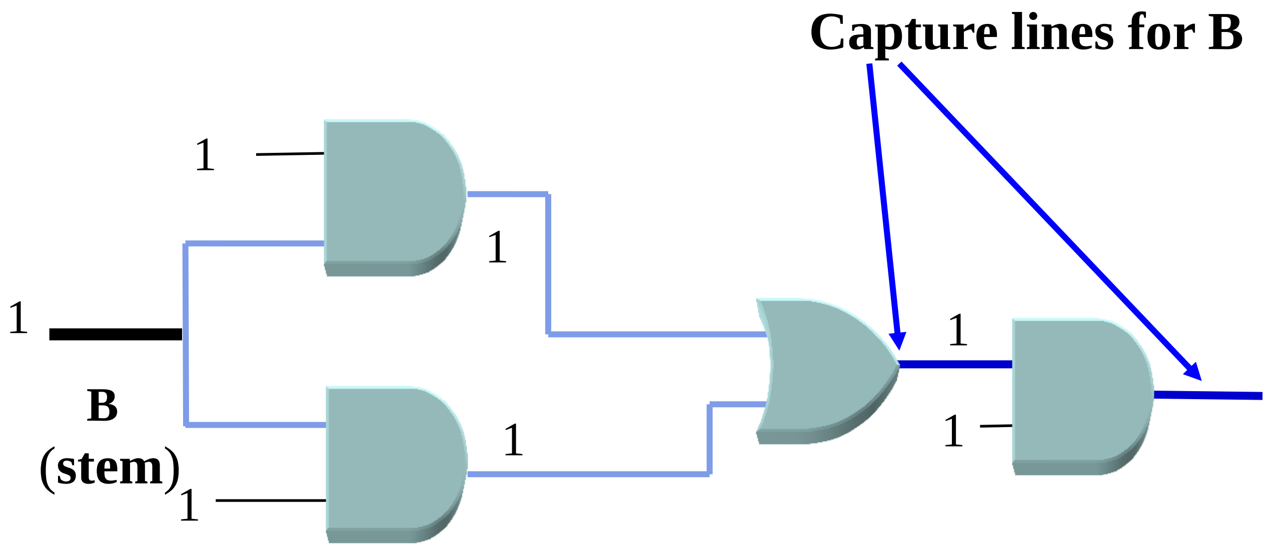


Algorithm of Critical Path Tracing

- For a fanout-free circuit,
  - Critical lines can be determined by backward traversing the sensitive gate inputs from PO's, in linear time
  - In general, special processing has to be done for stems
- General Critical path tracing:
  1. Fault-free simulation
  2. Extract single-PO logic cones and process each cone
     1. Process stems with highest levels
     2. If the stem is critical, mark critical lines in the fanout free region according to the sensitivity properties.

Critical Property of a Stem

- A stem is critical
  - If all the capture lines are critical.
  - All sensitized paths to POs for the stem pass through the capture lines (dominator).
- Note that this is only an approximation!

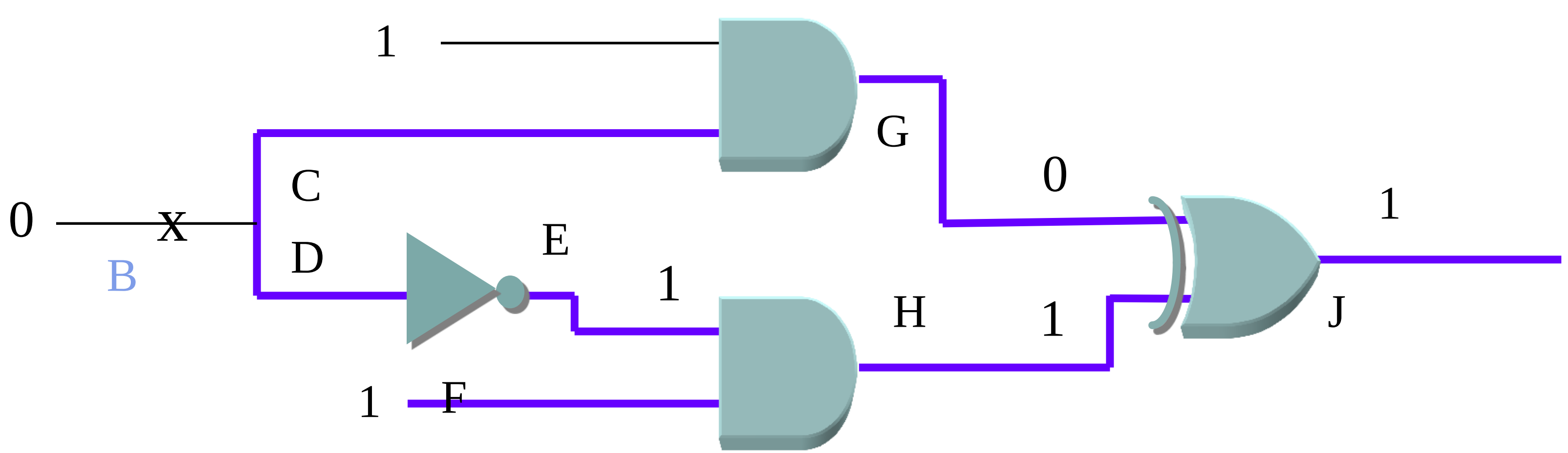


Problems of Critical Path Tracing

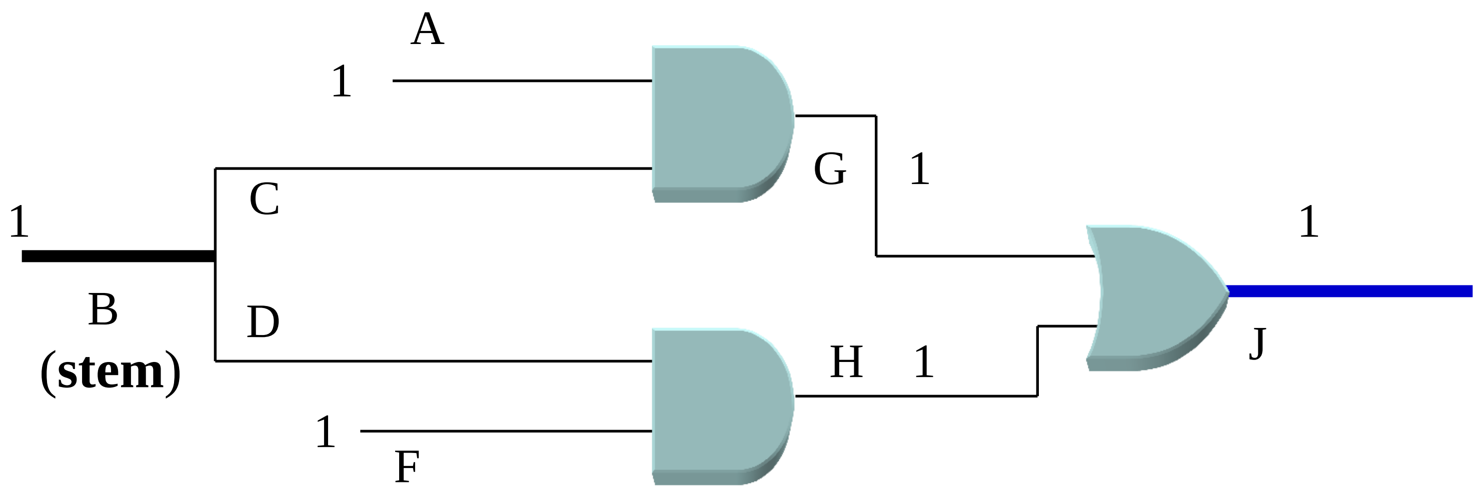
- The fault coverage by complete critical path tracing is approximate.
  - Self-masking paths lead to over-estimation
  - Multiple-path sensitization leads to under-estimation.
- Not very useful as a standalone simulator.
- Applied to individual fanout-free regions
  - Note that stem faults are only simulated to its dominators and then faults at dominators become representatives.

Self-masking Paths

- Detected faults in the fanout-free region: {J/0, H/0, G/1, F/0, E/0, D/1, C/1}
- Stem criticality is hard to infer from branches.
  - For example, is B/1 detectable by the given pattern?
- B/1 is not detectable even though both C and D are critical, because their effects cancel out each other at gate J
  - Self masking when paths go through different inversions



Multiple Path Sensitization



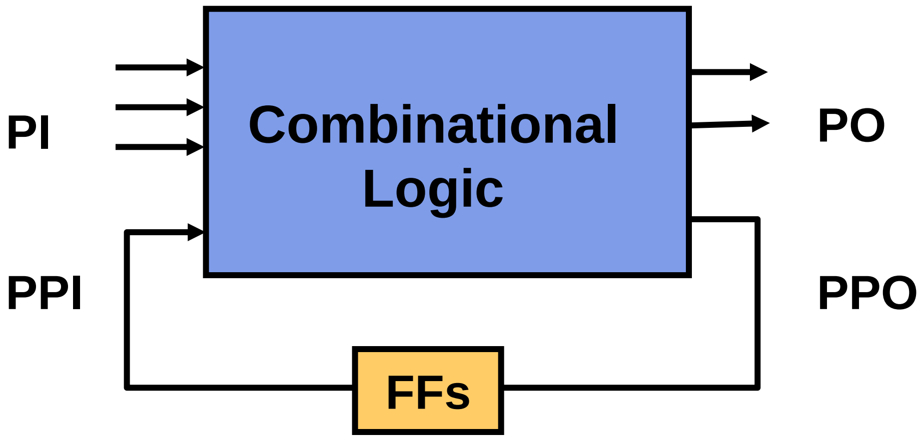
Both C and D are not critical, yet B is critical and B/0 can be detected at J by multiple path sensitization.

Techniques for Sequential Fault Simulation

- Fault simulation without restoration
- Fault grouping for parallel fault simulation
- Single Event Faults
- Management of hypertrophic faults
- Parallel pattern simulation for sequential circuits

Sequential Circuit Model

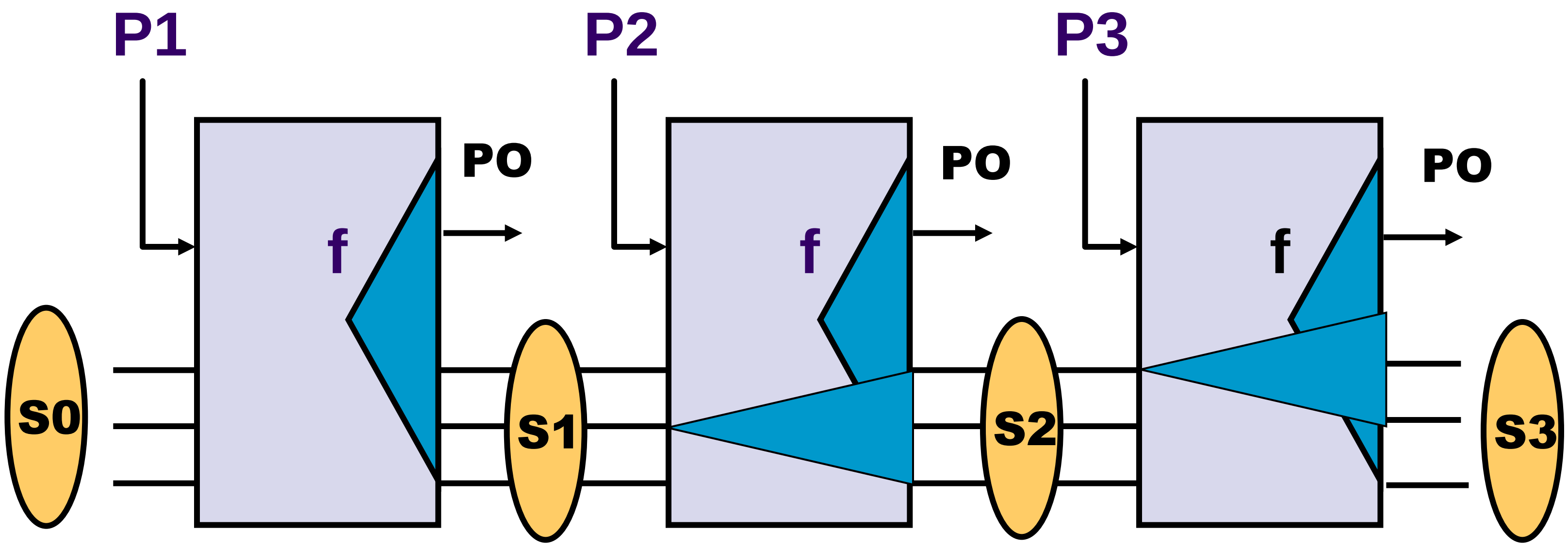
- For sequential circuits, a Huffman model will be used for the following discussion.



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

Faulty State in Sequential Simulation

- Faults will propagate to FF's
  - Need to record faulty states
  - Like multiple faults

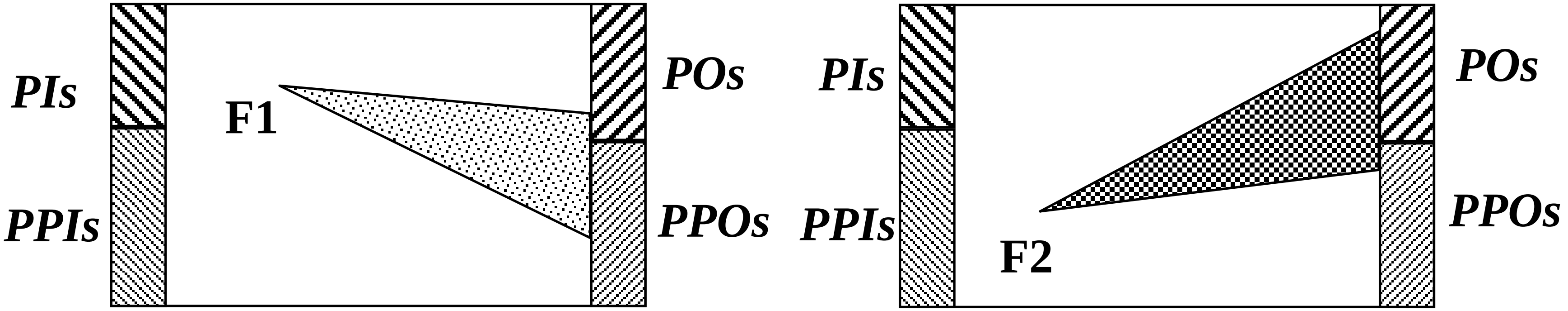


Ex: Input Sequence (P1, P2, P3)
State Sequence (S0 → S1 → S2 → S3)

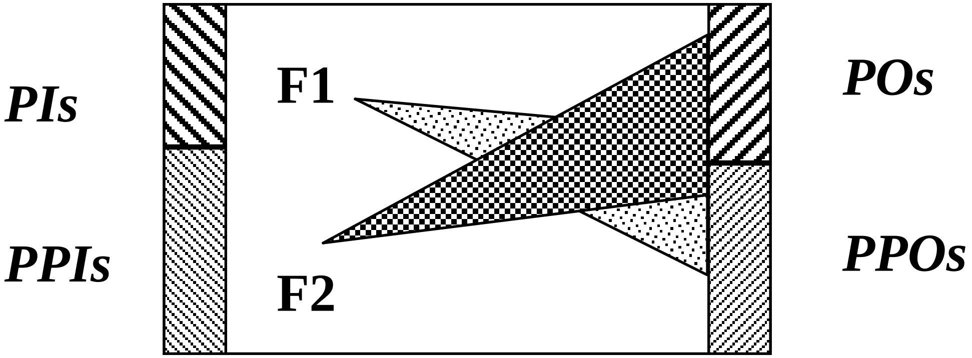
Fault Simulation without Restoration

- A new fault can be simulated without restoring status of previous fault status to fault-free value.
  - To store and restore fault-free values is inefficient
- Techniques for simulation without restoration
  - Only two sets of values are retained at each gate: one for fault-free and one for faulty ones.
  - Each active fault at any given time is given a unique identifier (ID).
    - Only faulty values with the same ID are used.
    - Otherwise, true values are used.
- Efficient in both time and memory space[ROOFS89].

Simulation without Restoration



With Restoration: Effect of F1 is cleared before simulating F2

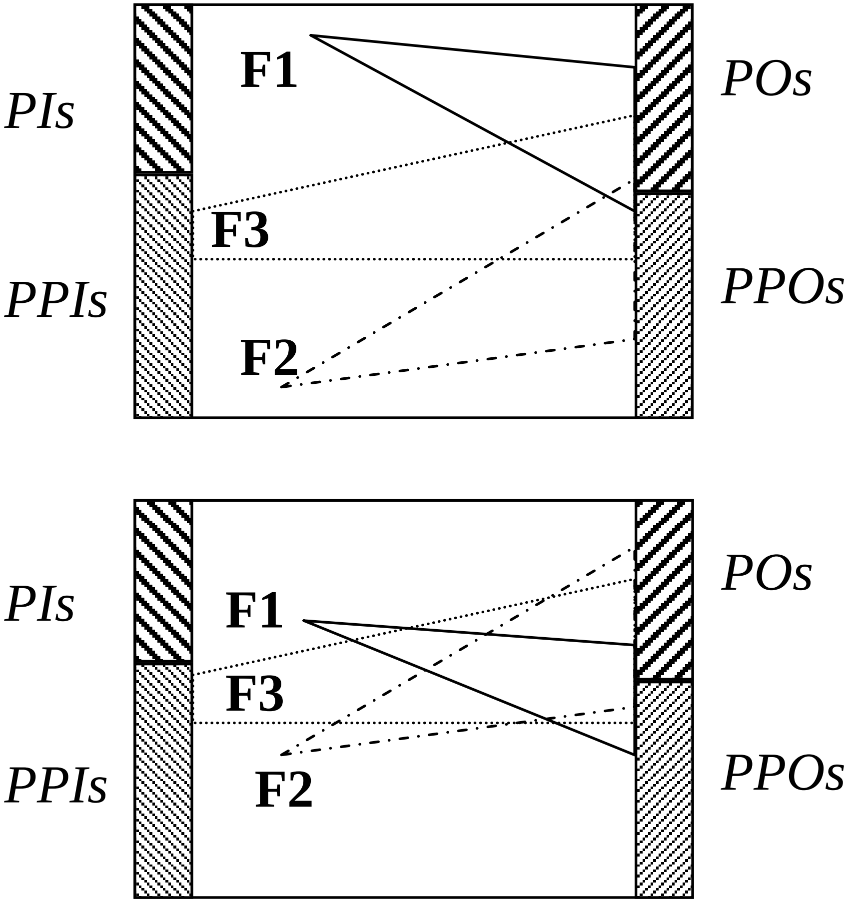


Without Restoration: F2 is simulated right over effect of F1

Fault Grouping

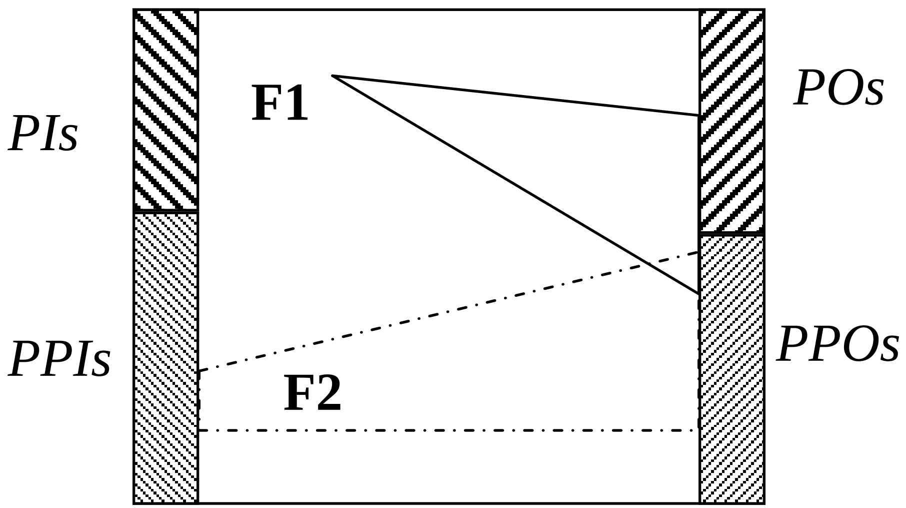
- Faults with coincident events can be grouped together for parallel simulation.
  - Static fault grouping
    - Based on circuit structure such as fan-in cone, depth-first path from POs.
  - Dynamic fault grouping
    - Fault grouped according to the similarity of faulty states
    - At the beginning of each time frame, faults are dynamically re-grouped for effective usage of all bits in a word

Fault Grouping in Parallel Fault Simulation



- The lower grouping case is evidently more desirable than the upper case because similar faulty events are simulated in parallel.
- Similar events have higher probability to be
  - dropped together
  - detected together
- Distinct faulty events will cause unnecessary simulation of inactive faults

Single Event Faults



- For circuit in a time frame, faults are classified into single event faults, e.g. , F1, and multiple event faults, e.g., F2.

- Single event faults are those whose effects originate only from the fault sites of the current time frame.
- Sophisticated combinational techniques, such as simulate-to-dominators, can then be applied with significant performance improvement.
- Only single-event stem faults or multiple-event faults are parallel-simulated.
- [HOPE DAC92]

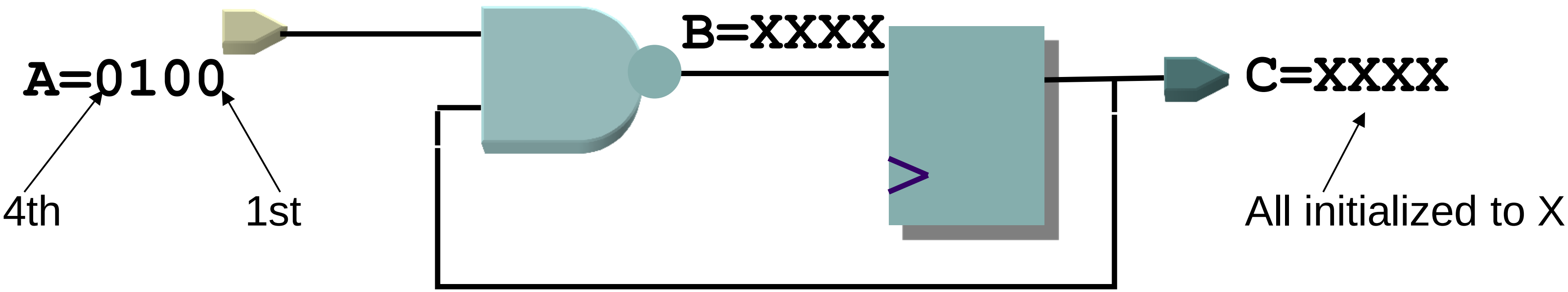
Hypertrophic Faults

- A hypertrophic fault is a fault that diverges from the fault-free circuit with a large number of X's
  - Usually a stuck-at fault occurring at a control line and thus prevents the circuit initialization
  - A small number of hypertrophic faults can account for a large percentage of fault events and CPU time
- These faults are sometimes dropped
  - as potentially detected faults to reduce simulation time. (Fault coverage is approximate)
- In parallel fault simulator
  - such as PROOFS, these faults can be simulated in parallel with fault-free circuit with little additional memory while achieving performance close to approximate simulator [HyHOPE IEE Circuits, Devices and Systems 94].

Parallel Pattern Simulation for Sequential Circuits

- Sequential circuits are state dependent by nature
  - However, many of them have a short memory, which can be exploited by parallelism of sequences or patterns.
- The basic mechanism of both parallel sequence and pattern
  - To initialize unknown FFs as Xs
  - Simulate the circuit and update the FFs by multiple passes.
  - States will converge to the true values eventually.
- The convergence rate (# of passes)
  - Determines the speed of simulation.
  - Usually small compared with the word length
  - Dependent on the circuit as well as the given sequence.

Sequential Simulation Example



1st **A=0** **B=XXX1** **C= XX1X**

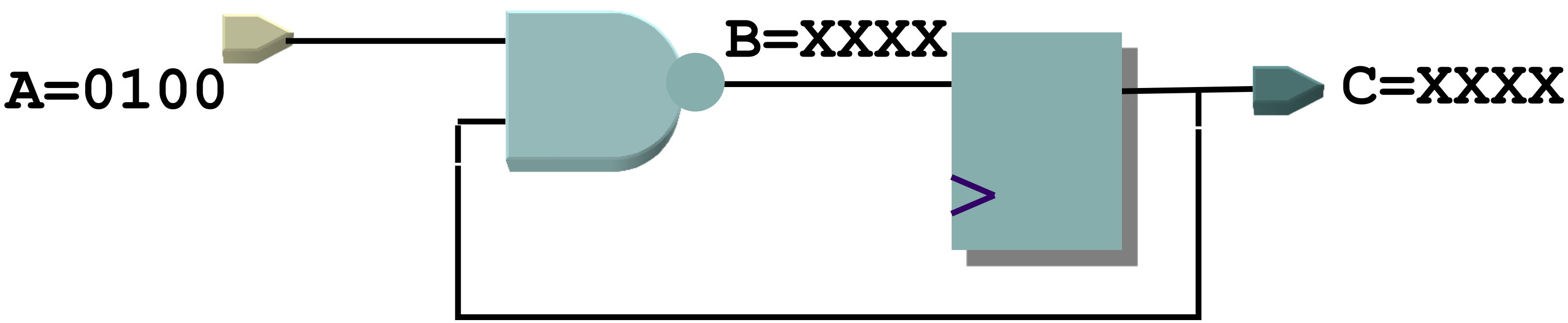
A=0 **B=XX11** **C= X11X**

A=1 **B=X011** **C= 011X**

4th **A=0** **B=1011** **C=1011X**

To be used for 5th

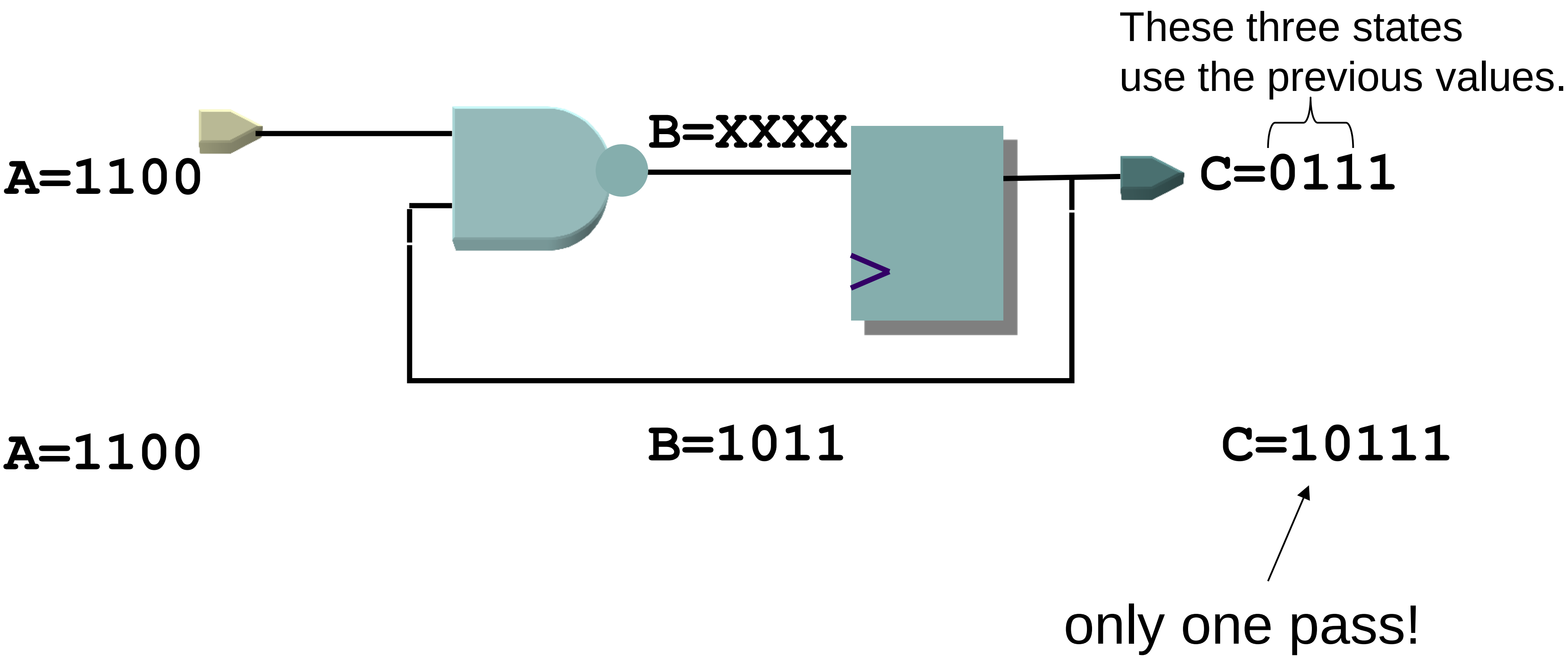
Parallel Sequential Simulation Example



A=0100	B=1X11	C=1X11X
A=0100	B=1011	C=1011X
A=0100	B=1011	C=1011X

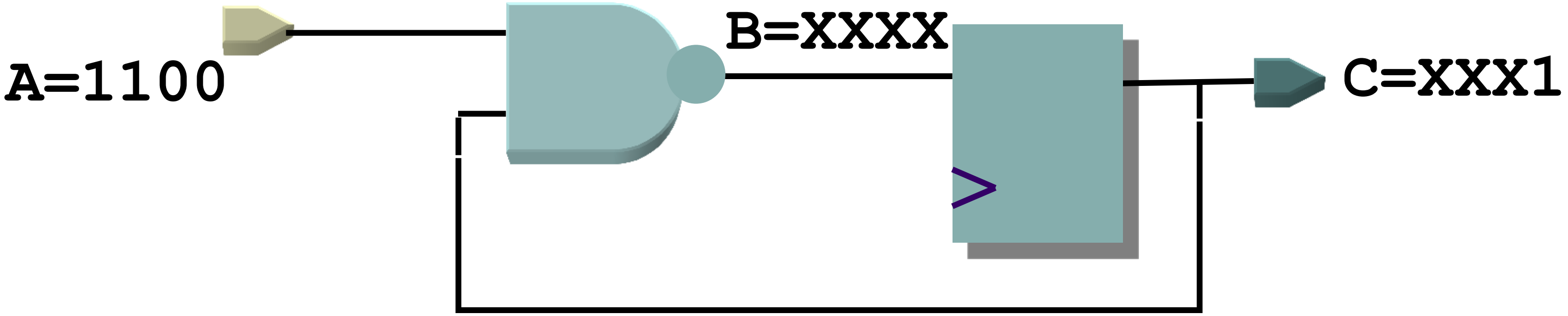
Converge for three passes

Parallel Sequential Simulation with Look-ahead Heuristic



Usually most state variables stay the same, so they are good guess for next simulation.

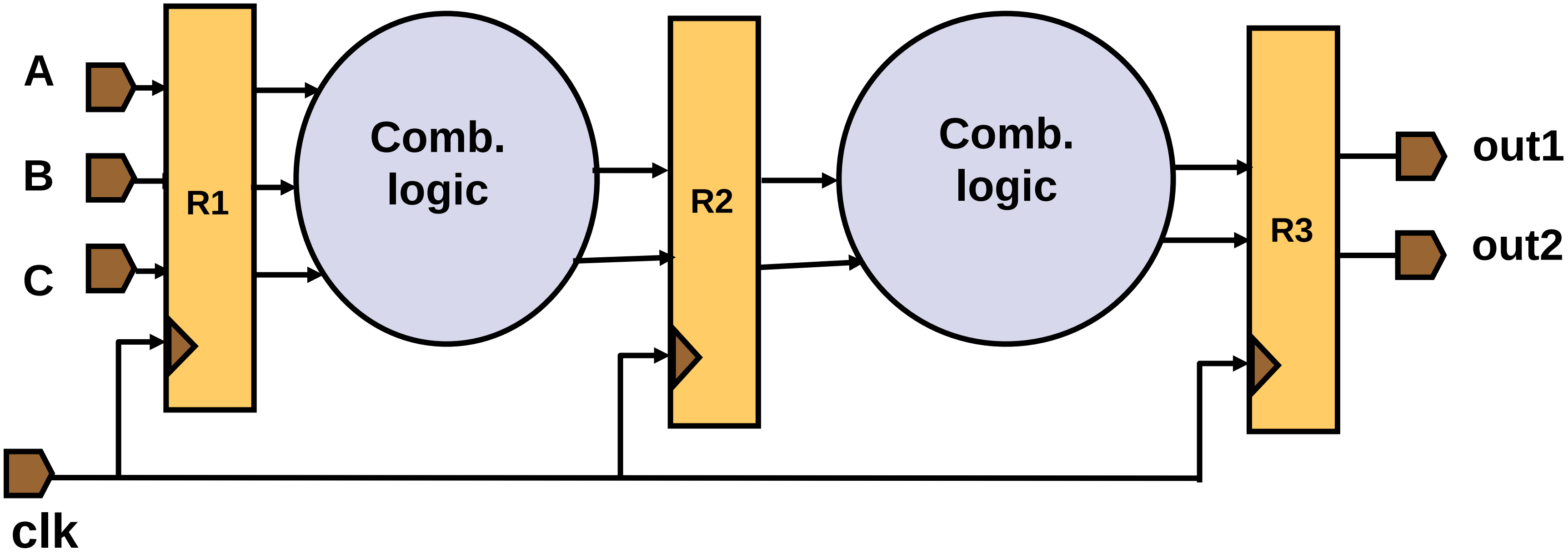
Parallel Sequential Simulation Example (If Look-ahead not used)



A=1100	B=XX11	C=XX111
A=1100	B=X011	C=X0111
A=1100	B=1011	C=10111

Also need three passes

An Extreme Example of Short-Memory Circuit



A pipelined data-path

Because there is no feedback, every patterns can be simulated independently.

Fault Grading

- Approximate fault coverage
  - Can be obtained in much shorter computational time than regular fault simulation.
  - Not suitable for high fault coverage.
    - It often occurs that the accuracy requirement exceeds error ranges of fault grading.
- 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

Toggle Test

- Circuit node transition
  - is computed during logic simulation and the number of node transition or toggle rate is used as the basis of fault coverage estimation.
  - e.g. zero toggle rate implies either s-a-0 or s-a-1 is not detected.
- Only logic simulation is performed.
  - very low computational cost
- Controllability-based estimation
  - Single stuck-at faults are only activated but not necessary propagated to POs.

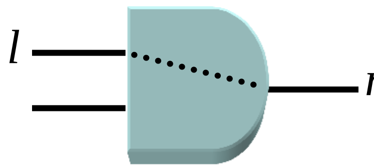
STAFAN

- 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.)

- The probability of detecting a fault with n test vectors

$$d_f^n = 1 - (1 - d_f)^n$$

  - *n is the # of vectors*
  - *$(1\text{-}d_f)^n$ is prob. of not being detected* ***after n test vectors***

- The fault coverage is $\dfrac{\sum_{f=\Phi} d_f^n}{|\Phi|}$

  - Φ is the number of total faults of interests

- More sophisticated than toggle test with same computation complexity

Fault Sampling

- M: total number of collapsed faults
  m: number of faults randomly selected
  K: number of faults detected by the test set T
  k: number of selected faults detected by T
- Actual fault coverage $F = K/M$
- $P_k(M, m, K)$: the probability that T will detect k faults from a random sample size of m, given that it detects K faults from the entire set of M faults.

- $$P_k(m, M, K) = \frac{C_k^K C_{m-k}^{M-K}}{C_m^M}$$

- ***P_k* is hypergeometric distribution with mean** $\mu_k = mK/M = mF$
  variance $\sigma_k^2 \quad = mK/M\,[1-k/M\,](M-m)/(M-1)$
  $\cong mF(1-F)(1-m/M)$

- For large *M*, **this distribution can be approximated by a normal distribution with mean μ_k and standard deviation σ_k.**

- **The estimated fault coverage *f* is a random variable with normal distribution:**

$$\mu_f = \mu_k / m = F$$
$$\sigma_f^2 = \sigma_k^2 / m^2 = (1/m)F(1-F)(1-m/M)$$

- **With a confidence level of 99.7%, the estimated fault coverage $f \in [F-3\sigma_f, F+3\sigma_f]$.**

$$e_{\max} = 3\sqrt{F(1-F)(1-m/M)1/m}$$
$$\cong 3\sqrt{F(1-F)/m} \qquad \text{if } m << M$$

Independent of *M*!!

Maximum Errors of the Estimated Fault Coverage