Fault Modeling

- Why model faults?
- Some real defects in VLSI and PCB
- Common fault models
- Stuck-at faults
  - Single stuck-at faults
  - Fault equivalence
  - Fault dominance and checkpoint theorem
  - Classes of stuck-at faults and multiple faults
- Transistor faults
- Summary

Why Model Faults?

- I/O function tests inadequate for manufacturing (functionality versus component and interconnect testing)
- Real defects (often mechanical) too numerous and often not analyzable
- A fault model identifies targets for testing
- A fault model makes analysis possible
- Effectiveness measurable by experiments

Some Real Defects in Chips

- Processing defects
  - Missing contact windows
  - Parasitic transistors
  - Oxide breakdown
  - . . .
- Material defects
  - Bulk defects (cracks, crystal imperfections)
  - Surface impurities (ion migration)
  - . . .
- Time-dependent failures
  - Dielectric breakdown
  - Electromigration
  - . . .
- Packaging failures
  - Contact degradation
  - Seal leaks
  - . . .

Ref.: M. J. Howes and D. V. Morgan, *Reliability and Degradation - Semiconductor Devices and Circuits,* Wiley, 1981.

Observed PCB Defects

Defect classes	Occurrence frequency (%)
Shorts	51
Opens	1
Missing components	6
Wrong components	13
Reversed components	6
Bent leads	8
Analog specifications	5
Digital logic	5
Performance (timing)	5

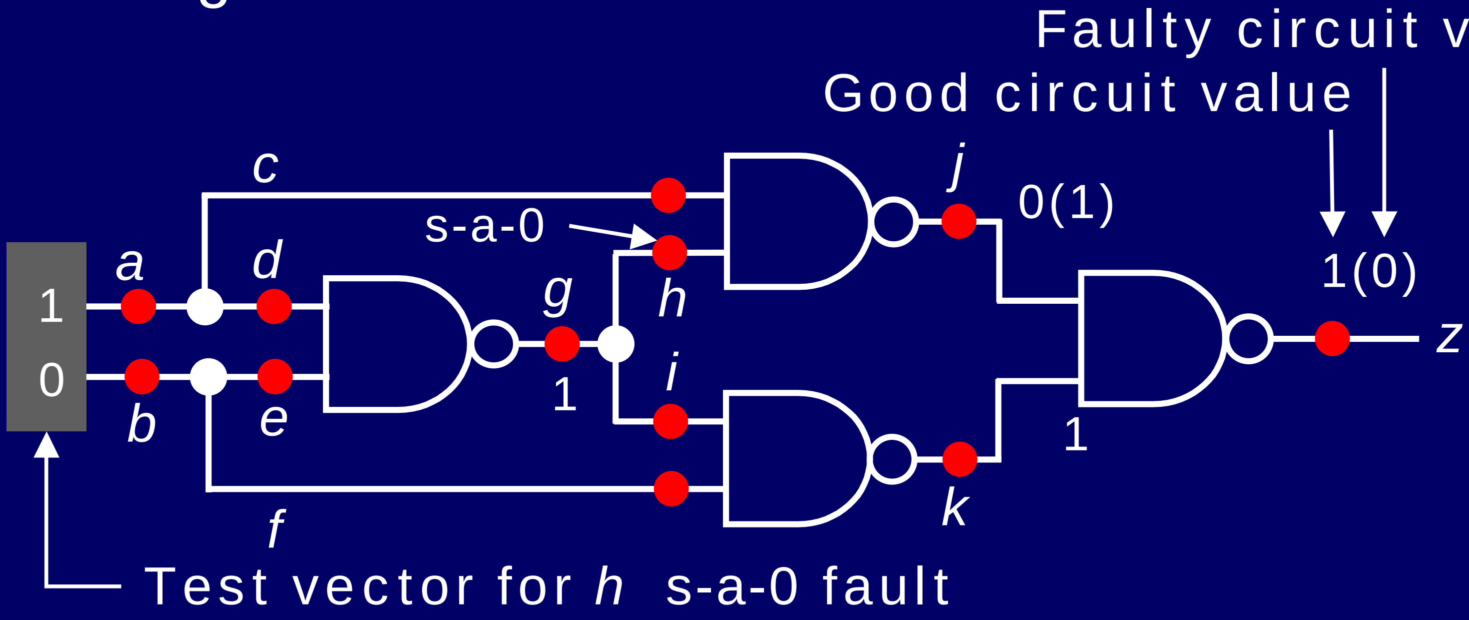
Ref.: J. Bateson, *In-Circuit Testing*, Van Nostrand Reinhold, 1985.

Common Fault Models

- Single stuck-at faults
- Transistor open and short faults
- Memory faults
- PLA faults (stuck-at, cross-point, bridging)
- Functional faults (processors)
- Delay faults (transition, path)
- Analog faults
- For more examples, see Section 4.4 (p. 60-70) of the book.

Single Stuck-at Fault

- Three properties define a single stuck-at fault
  - Only one line is faulty
  - The faulty line is permanently set to 0 or 1
  - The fault can be at an input or output of a gate
- Example: XOR circuit has 12 fault sites (●) and 24 single stuck-at faults



Faulty circuit value

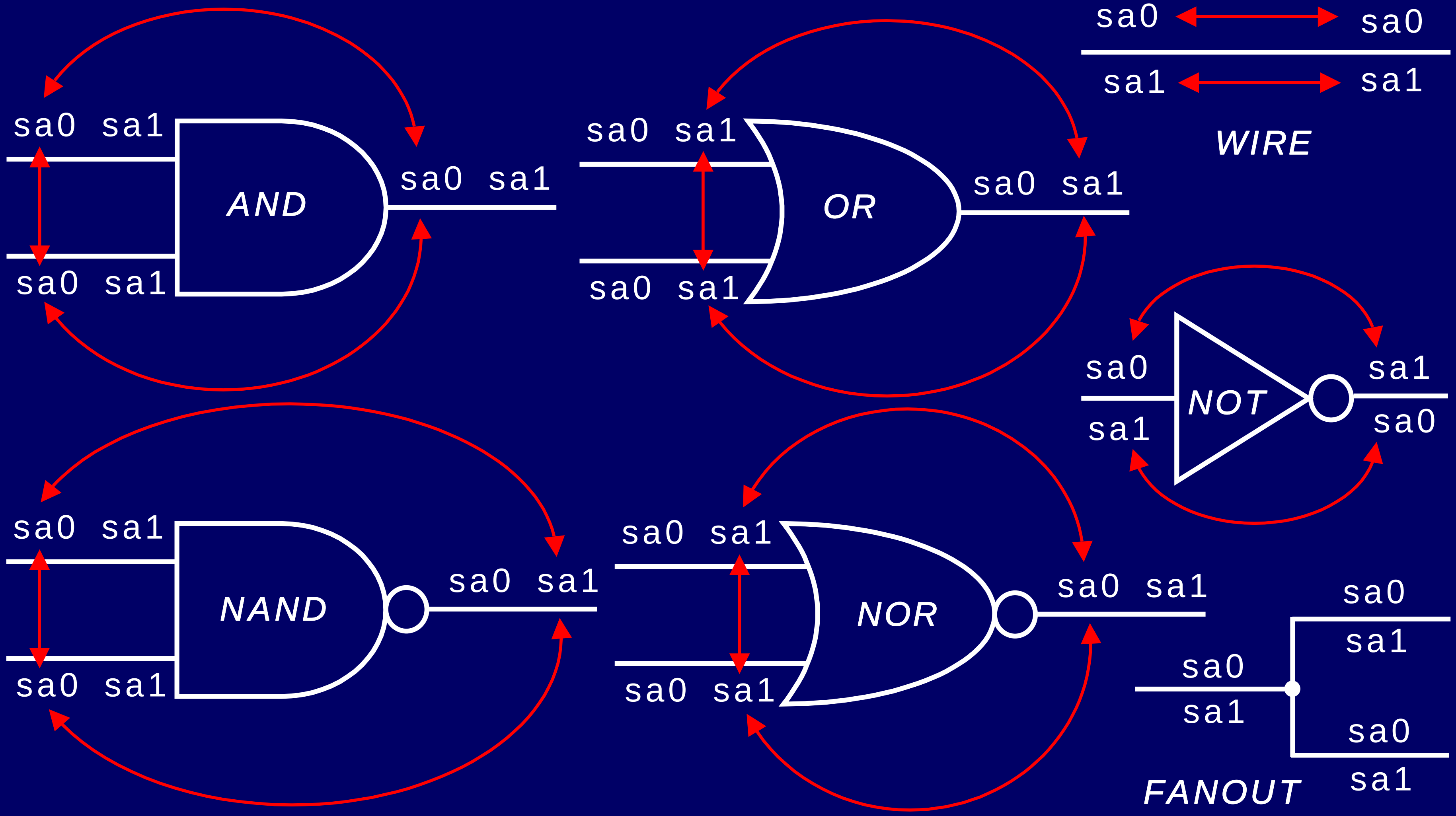
Good circuit value

Test vector for *h* s-a-0 fault

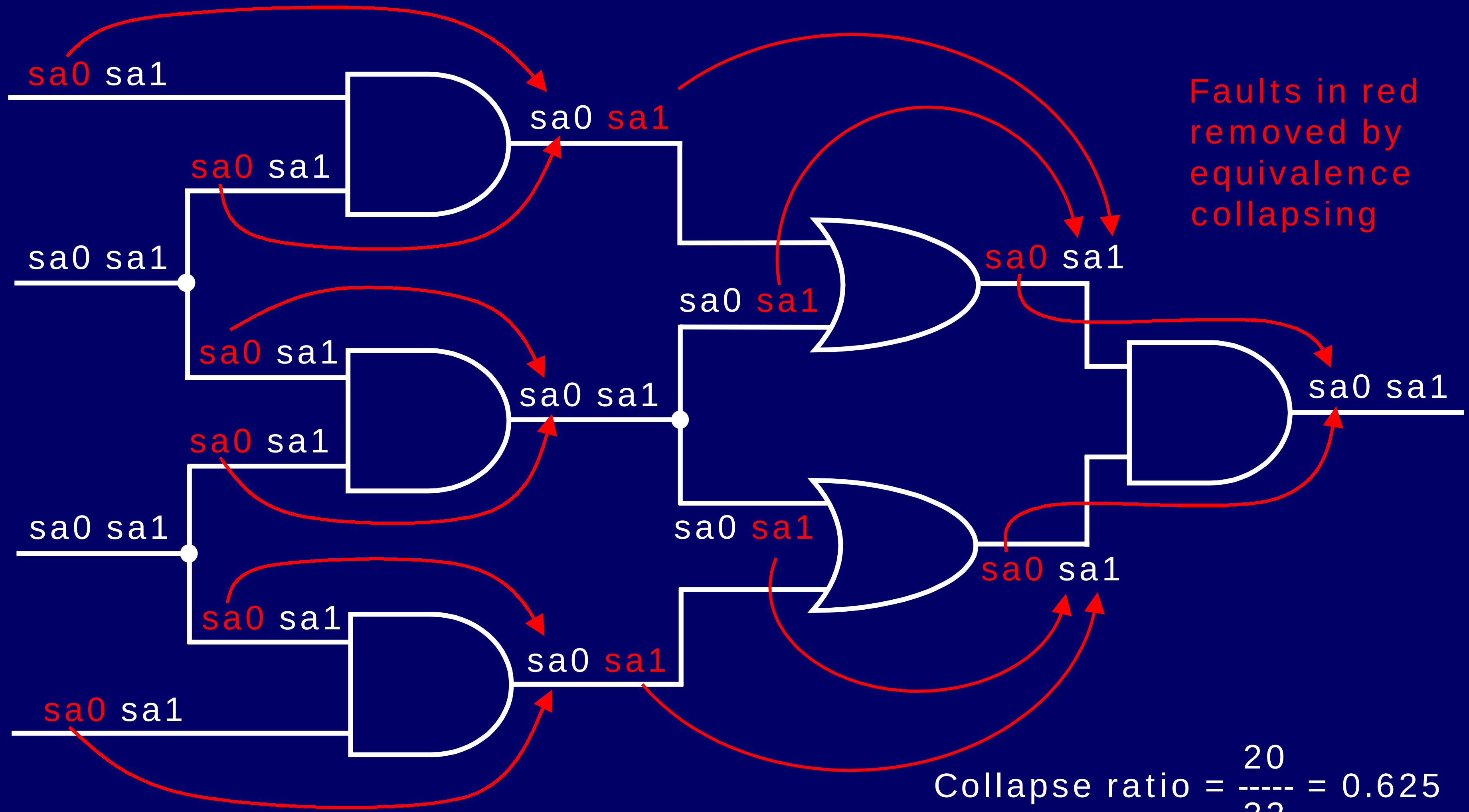
Fault Equivalence

- Number of fault sites in a Boolean gate circuit = #PI + #gates + # (fanout branches).
- Fault equivalence: Two faults f1 and f2 are equivalent if all tests that detect f1 also detect f2.
- If faults f1 and f2 are equivalent then the corresponding faulty functions are identical.
- Fault collapsing: All single faults of a logic circuit can be divided into disjoint equivalence subsets, where all faults in a subset are mutually equivalent. A collapsed fault set contains one fault from each equivalence subset.

Equivalence Rules



Equivalence Example



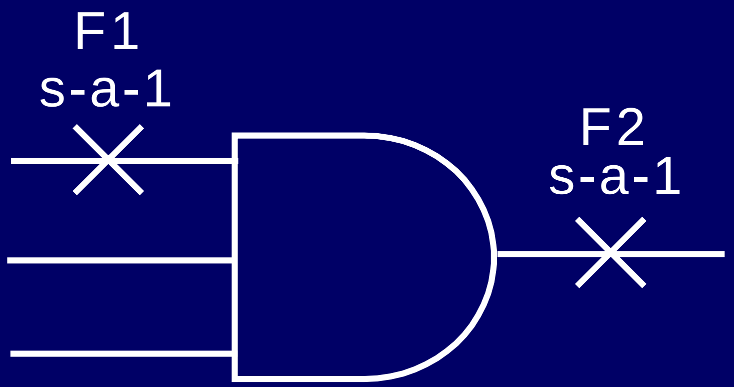
Faults in red removed by equivalence collapsing

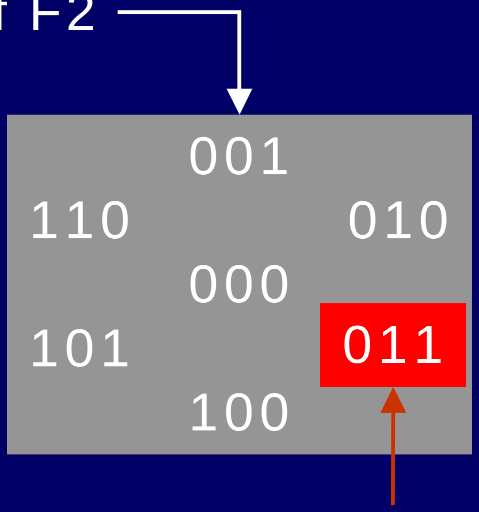
$$\text{Collapse ratio} = \frac{20}{32} = 0.625$$

Fault Dominance

- If all tests of some fault F1 detect another fault F2, then F2 is said to dominate F1.
- Dominance fault collapsing: If fault F2 dominates F1, then F2 is removed from the fault list.
- When dominance fault collapsing is used, it is sufficient to consider only the input faults of Boolean gates. See the next example.
- In a tree circuit (without fanouts) PI faults form a dominance collapsed fault set.
- If two faults dominate each other then they are equivalent.

Dominance Example





All tests of F2

001

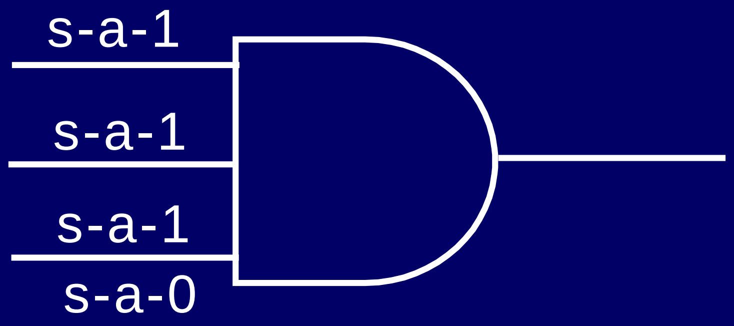
110 010

000

101 011

100

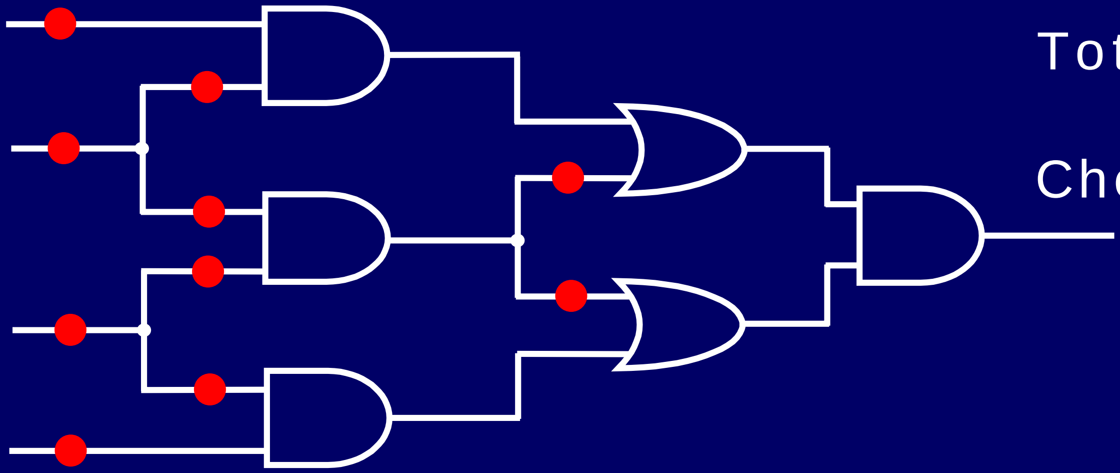
Only test of F1



A dominance collapsed fault set

Checkpoints

- Primary inputs and fanout branches of a combinational circuit are called *checkpoints*.
- Checkpoint theorem: A test set that detects all single (multiple) stuck-at faults on all checkpoints of a combinational circuit, also detects all single (multiple) stuck-at faults in that circuit.



Total fault sites = 16

Checkpoints (●) = 10

Classes of Stuck-at Faults

- Following classes of single stuck-at faults are identified by fault simulators:
  - *Potentially-detectable fault* -- Test produces an unknown (X) state at *primary output* (PO); detection is probabilistic, usually with 50% probability.
  - *Initialization fault* -- Fault prevents initialization of the faulty circuit; can be detected as a potentially-detectable fault.
  - *Hyperactive fault* -- Fault induces much internal signal activity without reaching PO.
  - *Redundant fault* -- No test exists for the fault.
  - *Untestable fault* -- Test generator is unable to find a test.

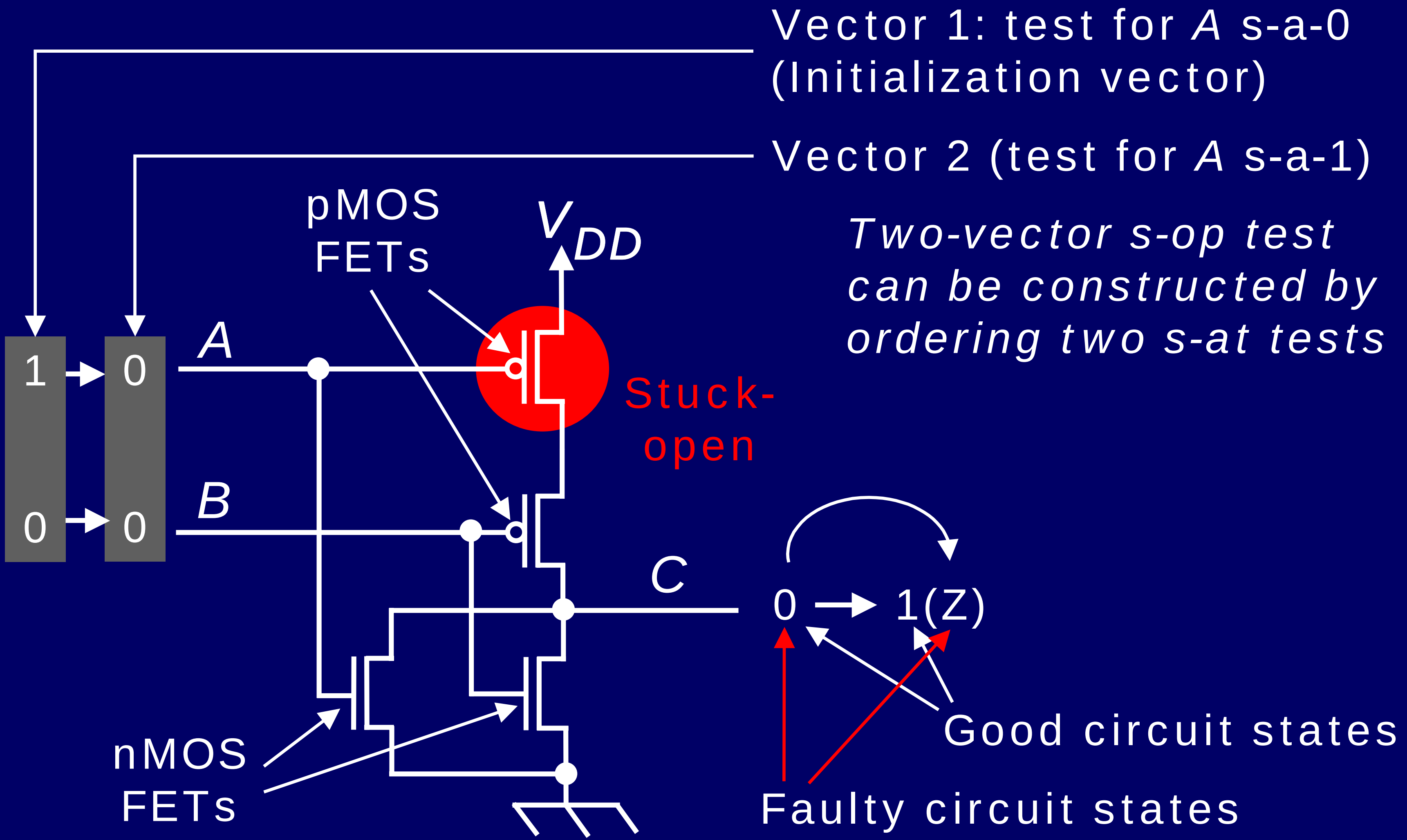
Multiple Stuck-at Faults

- A multiple stuck-at fault means that any set of lines is stuck-at some combination of (0,1) values.
- The total number of single and multiple stuck-at faults in a circuit with k single fault sites is 3^k-1.
- A single fault test can fail to detect the target fault if another fault is also present, however, such masking of one fault by another is rare.
- Statistically, single fault tests cover a very large number of multiple faults.

Transistor (Switch) Faults

- MOS transistor is considered an ideal switch and two types of faults are modeled:
  - Stuck-open -- a single transistor is permanently stuck in the open state.
  - Stuck-short -- a single transistor is permanently shorted irrespective of its gate voltage.
- Detection of a stuck-open fault requires two vectors.
- Detection of a stuck-short fault requires the measurement of quiescent current (I_{DDQ}).

Stuck-Open Example



Vector 1: test for *A* s-a-0 (Initialization vector)

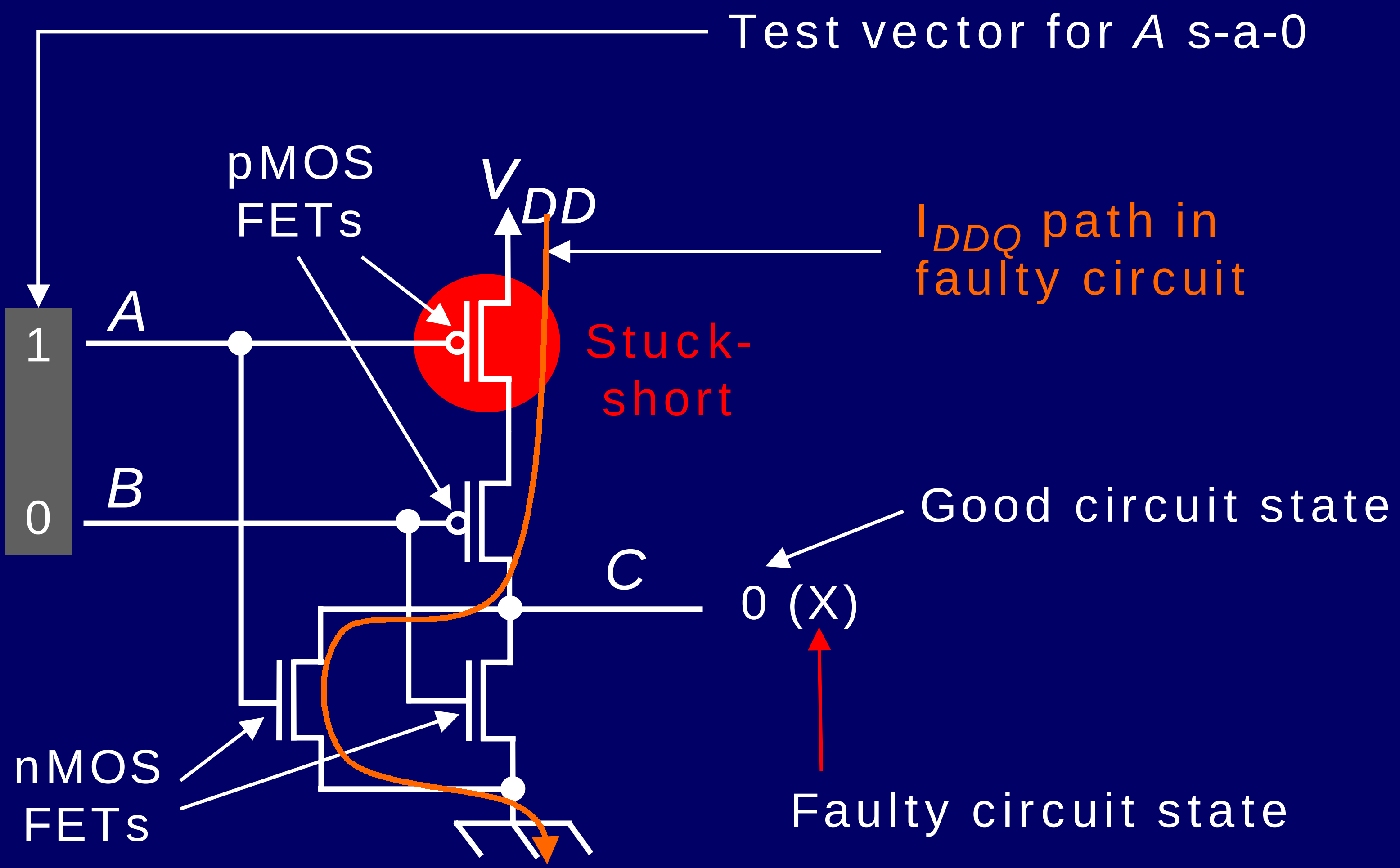
Vector 2 (test for *A* s-a-1)

Two-vector s-op test can be constructed by ordering two s-at tests

Good circuit states

Faulty circuit states

Stuck-Short Example



Summary

- Fault models are analyzable approximations of defects and are essential for a test methodology.
- For digital logic single stuck-at fault model offers best advantage of tools and experience.
- Many other faults (bridging, stuck-open and multiple stuck-at) are largely covered by stuck-at fault tests.
- Stuck-short and delay faults and technology-dependent faults require special tests.
- Memory and analog circuits need other specialized fault models and tests.

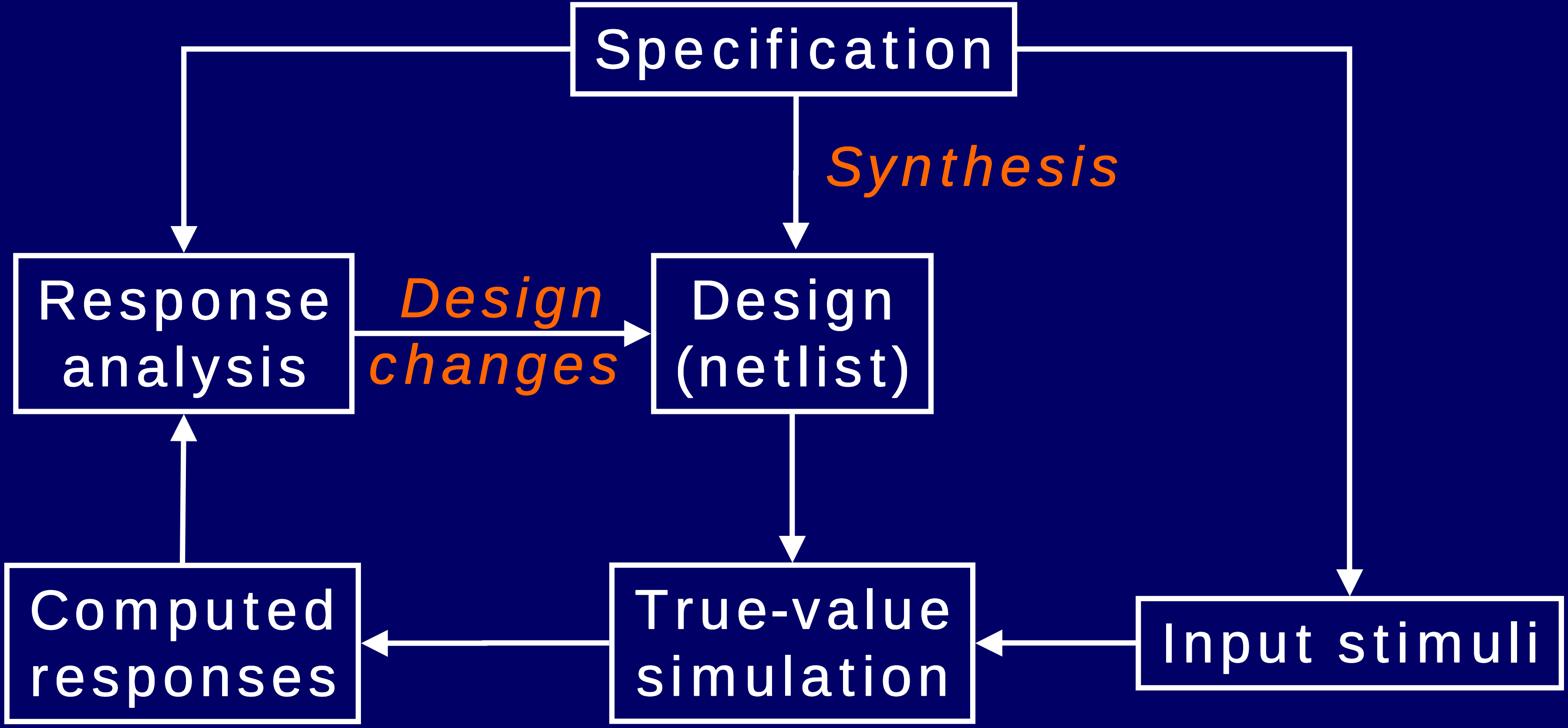
Logic Simulation

- What is simulation?
- Design verification
- Circuit modeling
- True-value simulation algorithms
  - Compiled-code simulation
  - Event-driven simulation
- Summary

Simulation Defined

- Definition: Simulation refers to modeling of a design, its function and performance.
- A software simulator is a computer program; an emulator is a hardware simulator.
- Simulation is used for design verification:
  - Validate assumptions
  - Verify logic
  - Verify performance (timing)
- Types of simulation:
  - Logic or switch level
  - Timing
  - Circuit
  - Fault

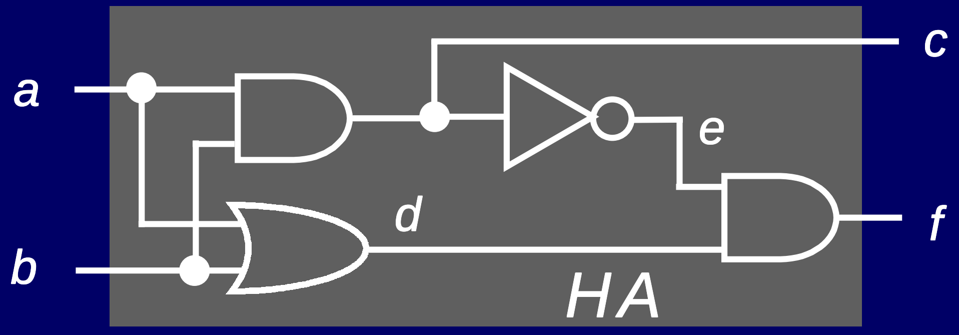
Simulation for Verification



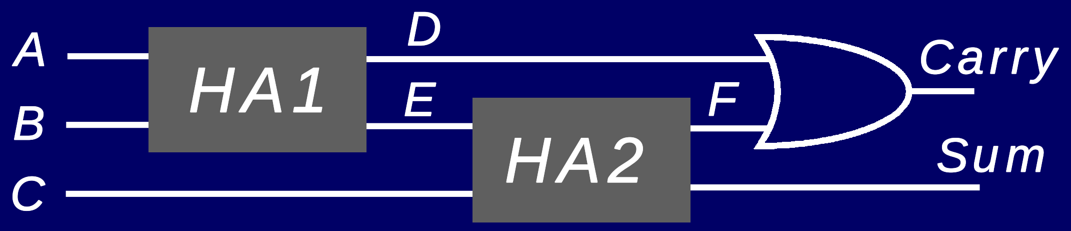
Modeling for Simulation

- Modules, blocks or components described by
  - Input/output (I/O) function
  - Delays associated with I/O signals
  - Examples: binary adder, Boolean gates, FET, resistors and capacitors
- Interconnects represent
  - ideal signal carriers, or
  - ideal electrical conductors
- Netlist: a format (or language) that describes a design as an interconnection of modules. Netlist may use hierarchy.

Example: A Full-Adder

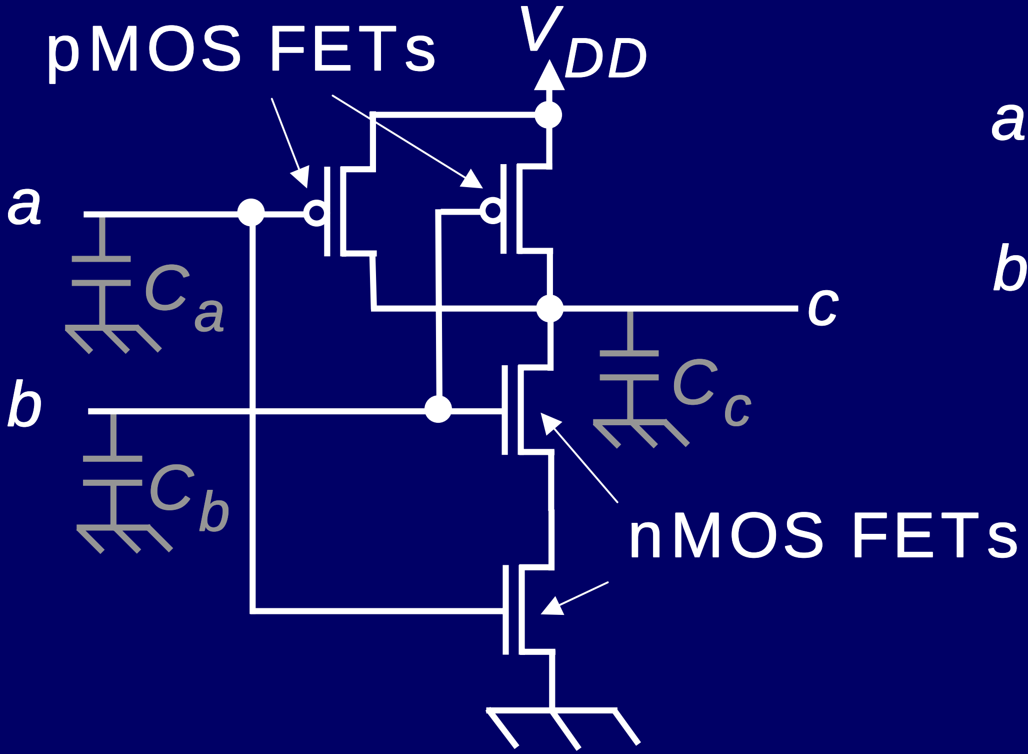


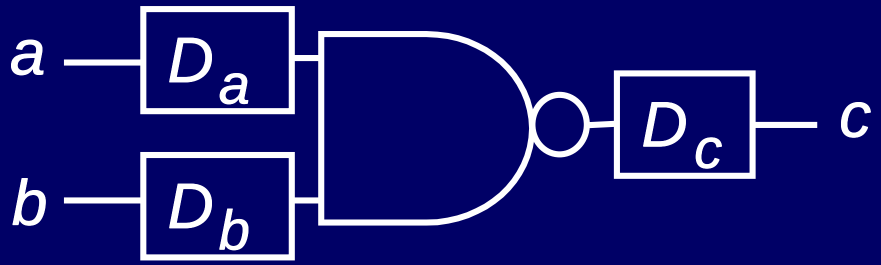
HA;
inputs: a, b;
outputs: c, f;
AND: A1, (a, b), (c);
AND: A2, (d, e), (f);
OR: O1, (a, b), (d);
NOT: N1, (c), (e);



FA;
inputs: A, B, C;
outputs: Carry, Sum;
HA: HA1, (A, B), (D, E);
HA: HA2, (E, C), (F, Sum);
OR: O2, (D, F), (Carry);

Logic Model of MOS Circuit





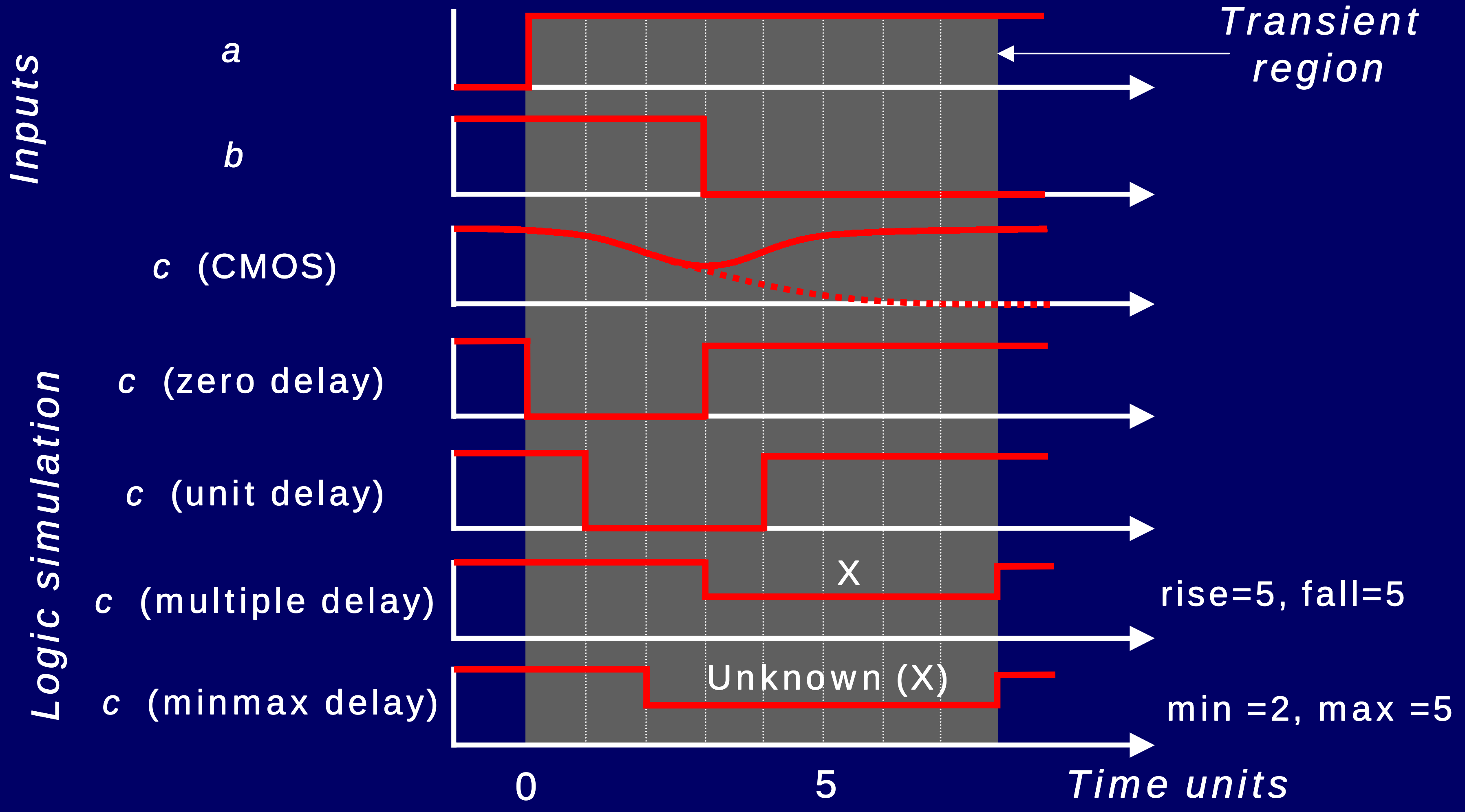
C_a , C_b *and* C_c are parasitic capacitances

D_a and D_b are interconnect or propagation delays

D_c is inertial delay of gate

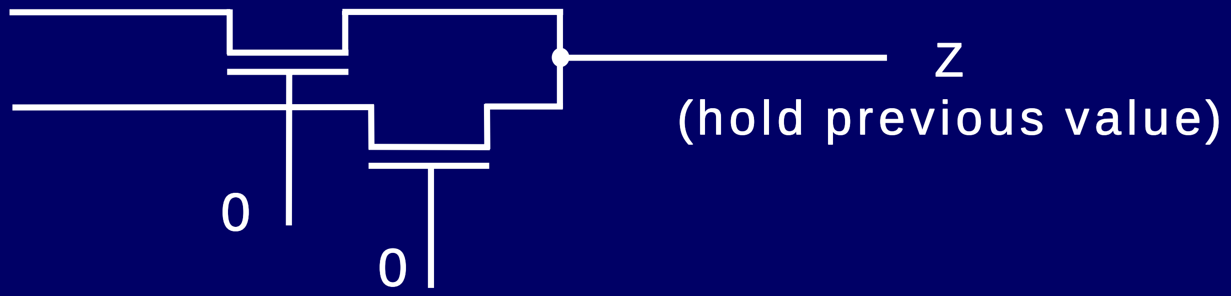
Options for Inertial Delay

(simulation of a NAND gate)



Signal States

- Two-states (0, 1) can be used for purely combinational logic with zero-delay.
- Three-states (0, 1, X) are essential for timing hazards and for sequential logic initialization.
- Four-states (0, 1, X, Z) are essential for MOS devices. See example below.
- Analog signals are used for exact timing of digital logic and for analog circuits.



Modeling Levels

Modeling level	Circuit description	Signal values	Timing	Application
Function, behavior, RTL	Programming language-like HDL	0, 1	Clock boundary	Architectural and functional verification
Logic	Connectivity of Boolean gates, flip-flops and transistors	0, 1, X and Z	Zero-delay unit-delay, multiple-delay	Logic verification and test
Switch	Transistor size and connectivity, node capacitances	0, 1 and X	Zero-delay	Logic verification
Timing	Transistor technology data, connectivity, node capacitances	Analog voltage	Fine-grain timing	Timing verification
Circuit	Tech. Data, active/ passive component connectivity	Analog voltage, current	Continuous time	Digital timing and analog circuit verification

True-Value Simulation Algorithms

- Compiled-code simulation
  - Applicable to zero-delay combinational logic
  - Also used for cycle-accurate synchronous sequential circuits for logic verification
  - Efficient for highly active circuits, but inefficient for low-activity circuits
  - High-level (e.g., C language) models can be used
- Event-driven simulation
  - Only gates or modules with input events are evaluated (*event means a signal change*)
  - Delays can be accurately simulated for timing verification
  - Efficient for low-activity circuits
  - Can be extended for fault simulation

Compiled-Code Algorithm

- Step 1: Levelize combinational logic and encode in a compilable programming language
- Step 2: Initialize internal state variables (flip-flops)
- Step 3: For each input vector

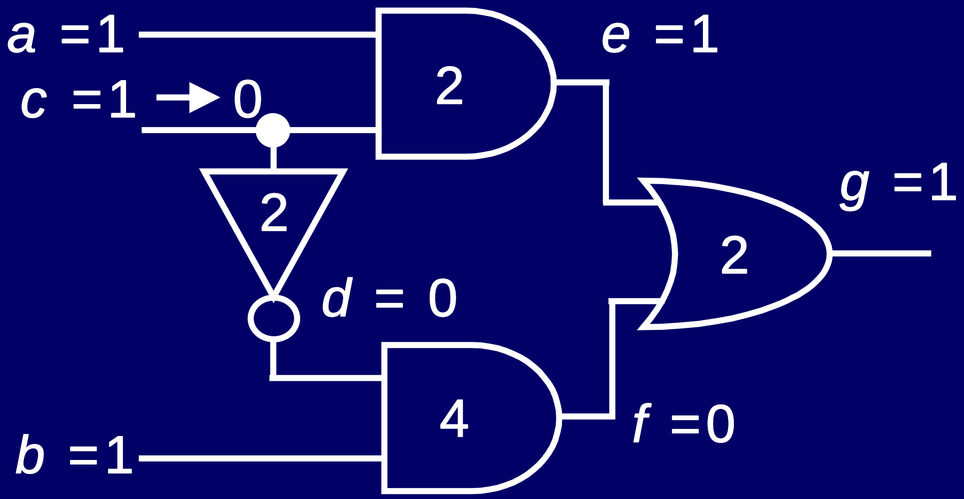
  Set primary input variables

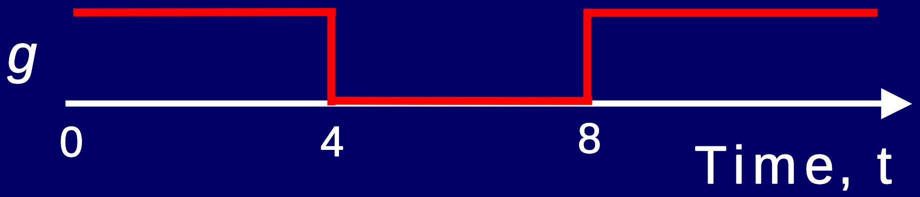
  Repeat (until steady-state or max. iterations)

  - Execute compiled code

  Report or save computed variables

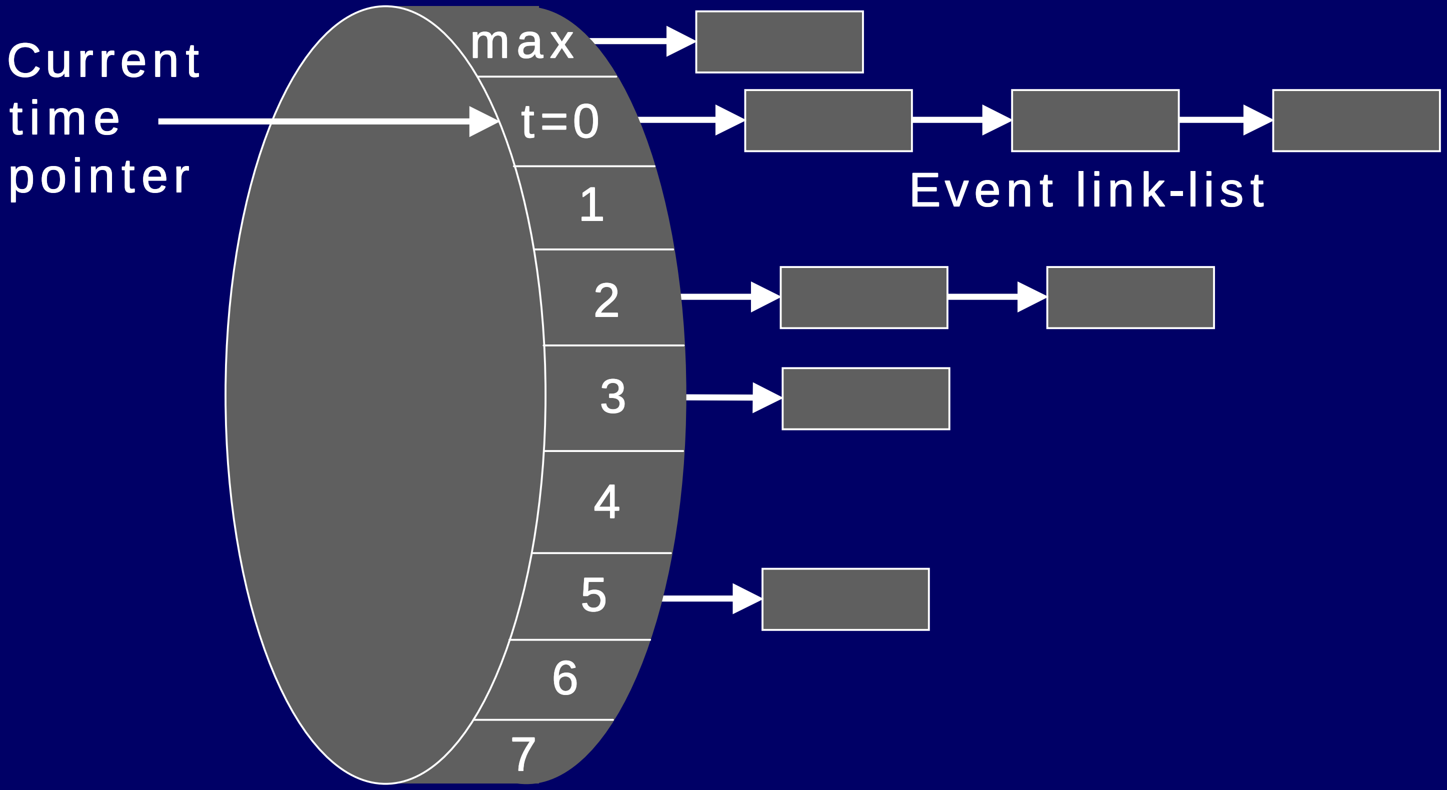
Event-Driven Algorithm (Example)





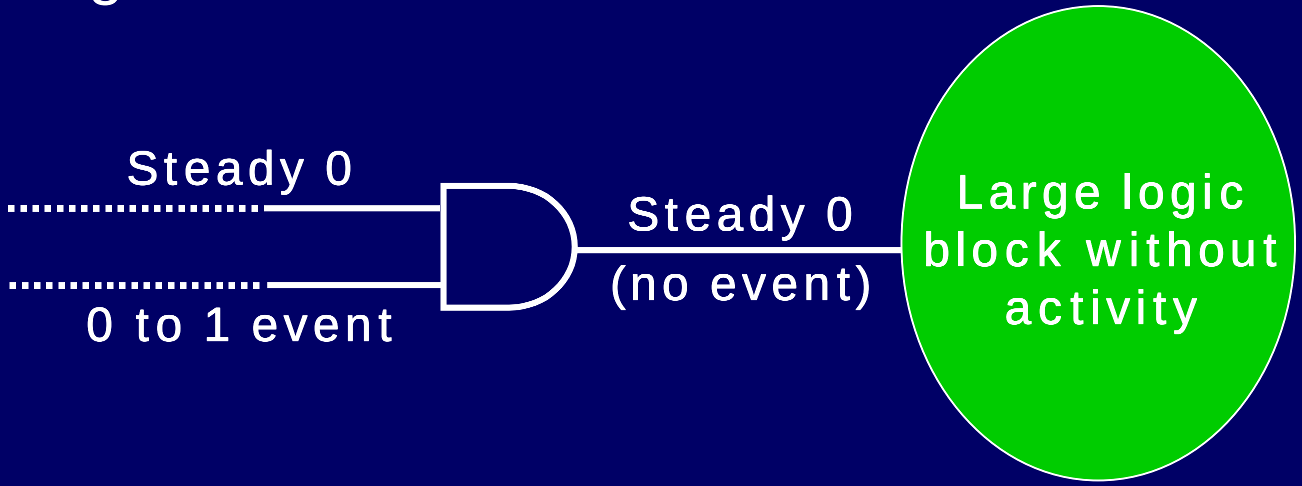
Time stack	Scheduled events	Activity list
t = 0	*c* = 0	*d*, *e*
1		
2	*d* = 1, *e* = 0	*f*, *g*
3		
4	*g* = 0	
5		
6	*f* = 1	*g*
7		
8	*g* = 1	

Time Wheel (Circular Stack)



Efficiency of Event-driven Simulator

- Simulates events (value changes) only
- Speed up over compiled-code can be ten times or more; in large logic circuits about 0.1 to 10% gates become active for an input change



Summary

- Logic or true-value simulators are essential tools for design verification.
- Verification vectors and expected responses are generated (often manually) from specifications.
- A logic simulator can be implemented using either compiled-code or event-driven method.
- Per vector complexity of a logic simulator is approximately linear in circuit size.
- Modeling level determines the evaluation procedures used in the simulator.