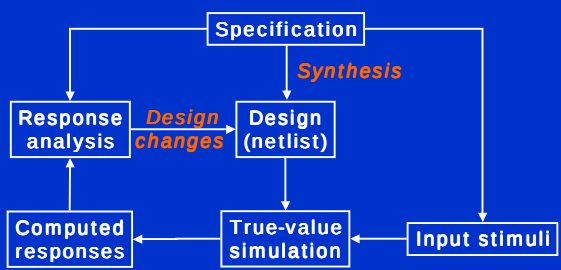
Logic Simulation

- 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

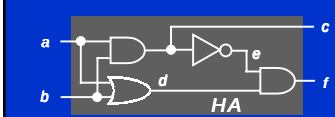
Simulation for Verification



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.

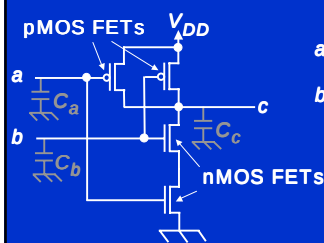
Example: A Full-Adder



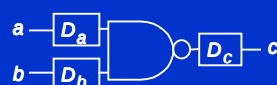
HA;
inputs: a, b;
outputs: c, f;
AND: A1, (a, b), (c);
AND: A2, (d, e), (f);
OR: O1, (a, b), (d);
NOT: N1, (c), (e);

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

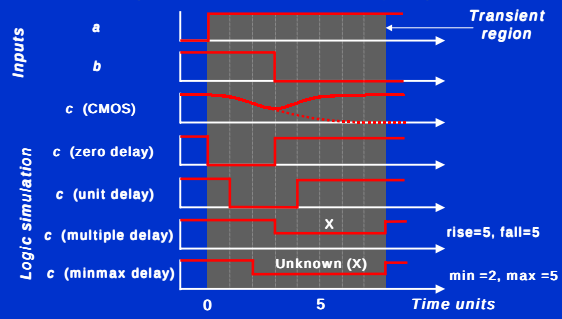


D_a and D_b are interconnect or propagation delays

D_c is inertial delay of gate

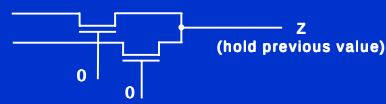
Options for Inertial Delay

(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

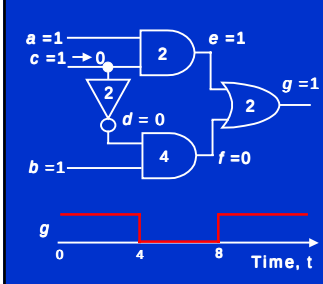
- Compiled-code simulation
  - Applicable to zero-delay combinational logic
  - Also used for cycle-accurate synchronous sequential circuits for logic verification
  - Efficient for highly active circuits, but inefficient for low-activity circuits
  - High-level (e.g., C language) models can be used
- 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

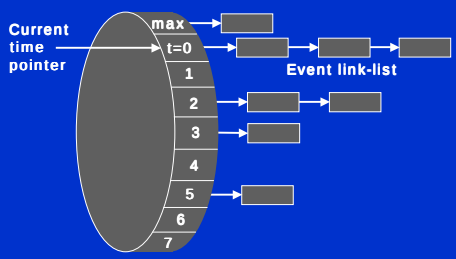
(Example)



Time stack

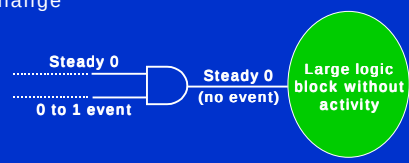
	Scheduled events	Activity list
t = 0	c = 0	d, e
1		
2	d = 1, e = 0	f, g
3		
4	g = 0	
5		
6	f = 1	g
7		
8	g = 1	

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.

Fault Simulation

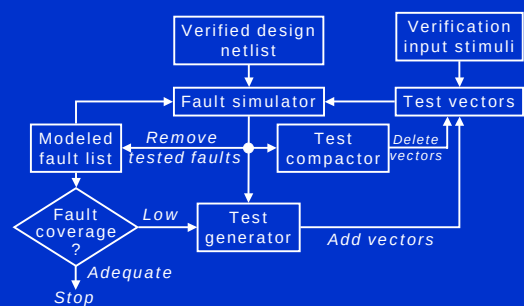
- Problem and motivation
- Fault simulation algorithms
  - Serial
  - Parallel
  - Deductive
  - Concurrent
- Random Fault Sampling
- Summary

Problem and Motivation

- Fault simulation Problem: Given
  - A circuit
  - A sequence of test vectors
  - A fault model

  Determine
  - Fault coverage - fraction (or percentage) of modeled faults detected by test vectors
  - Set of undetected faults
- Motivation
  - Determine test quality and in turn product quality
  - Find undetected fault targets to improve tests

Fault simulator in a VLSI Design Process



Fault Simulation Scenario

- Circuit model: mixed-level
  - Mostly logic with some switch-level for high-impedance (Z) and bidirectional signals
  - High-level models (memory, etc.) with pin faults
- Signal states: logic
  - Two (0, 1) or three (0, 1, X) states for purely Boolean logic circuits
  - Four states (0, 1, X, Z) for sequential MOS circuits
- Timing:
  - Zero-delay for combinational and synchronous circuits
  - Mostly unit-delay for circuits with feedback

Fault Simulation Scenario (continued)

- Faults:
  - Mostly single stuck-at faults
  - Sometimes stuck-open, transition, and path-delay faults; analog circuit fault simulators are not yet in common use
  - Equivalence fault collapsing of single stuck-at faults
  - Fault-dropping -- a fault once detected is dropped from consideration as more vectors are simulated; fault-dropping may be suppressed for diagnosis
  - Fault sampling -- a random sample of faults is simulated when the circuit is large

Fault Simulation Algorithms

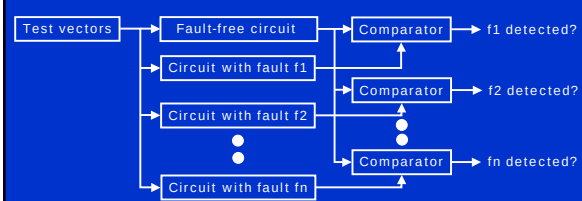
- Serial
- Parallel
- Deductive
- Concurrent
- Differential

Serial Algorithm

- Algorithm: Simulate fault-free circuit and save responses. Repeat following steps for each fault in the fault list:
  - Modify netlist by injecting one fault
  - Simulate modified netlist, vector by vector, comparing responses with saved responses
  - If response differs, report fault detection and suspend simulation of remaining vectors
- Advantages:
  - Easy to implement; needs only a true-value simulator, less memory
  - Most faults, including analog faults, can be simulated

Serial Algorithm (Cont.)

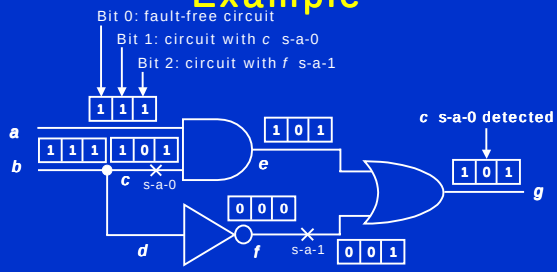
- Disadvantage: Much repeated computation; CPU time prohibitive for VLSI circuits
- Alternative: Simulate many faults together



Parallel Fault Simulation

- Compiled-code method; best with two-states (0,1)
- Exploits inherent bit-parallelism of logic operations on computer words
- Storage: one word per line for two-state simulation
- Multi-pass simulation: Each pass simulates *w*-1 new faults, where *w* is the machine word length
- Speed up over serial method ~ *w*-1
- Not suitable for circuits with timing-critical and non-Boolean logic

Parallel Fault Sim. Example

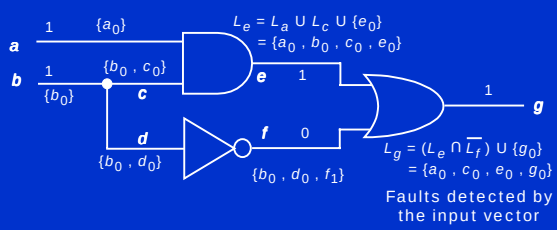


Deductive Fault Simulation

- One-pass simulation
- Each line k contains a list L_k of faults detectable on k
- Following true-value simulation of each vector, fault lists of all gate output lines are updated using set-theoretic rules, signal values, and gate input fault lists
- PO fault lists provide detection data
- Limitations:
  - Set-theoretic rules difficult to derive for non-Boolean gates
  - Gate delays are difficult to use

Deductive Fault Sim. Example

Notation: L_k is fault list for line k
k_n is s-a-n fault on line k

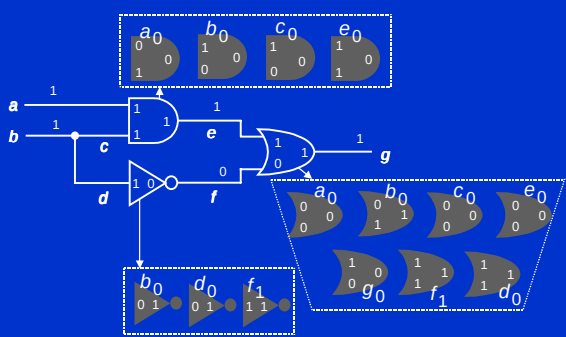


Faults detected by the input vector

Concurrent Fault Simulation

- Event-driven simulation of fault-free circuit and only those parts of the faulty circuit that differ in signal states from the fault-free circuit.
- A list per gate containing copies of the gate from all faulty circuits in which this gate differs. List element contains fault ID, gate input and output values and internal states, if any.
- All events of fault-free and all faulty circuits are implicitly simulated.
- Faults can be simulated in any modeling style or detail supported in true-value simulation (offers most flexibility.)
- Faster than other methods, but uses most memory.

Conc. Fault Sim. Example



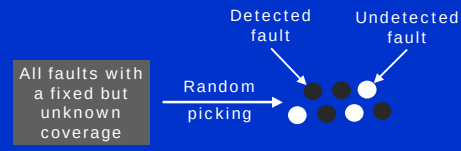
Fault Sampling

- A randomly selected subset (sample) of faults is simulated.
- Measured coverage in the sample is used to estimate fault coverage in the entire circuit.
- Advantage: Saving in computing resources (CPU time and memory.)
- Disadvantage: Limited data on undetected faults.

Motivation for Sampling

- Complexity of fault simulation depends on:
  - Number of gates
  - Number of faults
  - Number of vectors
- Complexity of fault simulation with fault sampling depends on:
  - Number of gates
  - Number of vectors

Random Sampling Model



N_p = total number of faults (population size)

C = fault coverage (unknown)

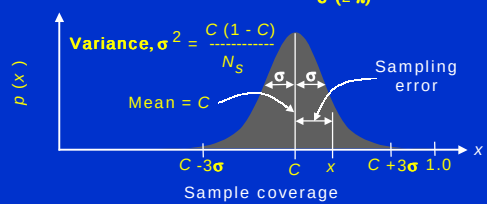
N_s = sample size

$N_s << N_p$

c = sample coverage (a random variable)

Probability Density of Sample Coverage, c

$$p(x) = \text{Prob}(x \le c \le x + dx) = \frac{1}{\sigma (2\pi)^{1/2}} e^{\frac{(x - C)^2}{2\sigma^2}}$$



Sampling Error Bounds

$$|x - C| = 3 \left[\frac{C(1 - C)}{N_s} \right]^{1/2}$$

Solving the quadratic equation for C, we get the 3-sigma (99.7% confidence) estimate:

$$C_{3\sigma} = x \pm \frac{4.5}{N_s} [1 + 0.44 N_s x (1 - x)]^{1/2}$$

Where N_s is sample size and x is the measured fault coverage in the sample.

Example: A circuit with 39,096 faults has an actual fault coverage of 87.1%. The measured coverage in a random sample of 1,000 faults is 88.7%. The above formula gives an estimate of 88.7%±3%. CPU time for sample simulation was about 10% of that for all faults.

Summary

- Fault simulator is an essential tool for test development.
- Concurrent fault simulation algorithm offers the best choice.
- For restricted class of circuits (combinational and synchronous sequential with only Boolean primitives), differential algorithm can provide better speed and memory efficiency (Section 5.5.6.)
- For large circuits, the accuracy of random fault sampling only depends on the sample size (1,000 to 2,000 faults) and not on the circuit size. The method has significant advantages in reducing CPU time and memory needs of the simulator.