Logic Simulation

Dr. Shubhajit Roy Chowdhury,

Centre for VLSI and Embedded Systems Technology,

IIIT Hyderabad, India

Email: src.vlsi@iiit.ac.in

Logic Simulation

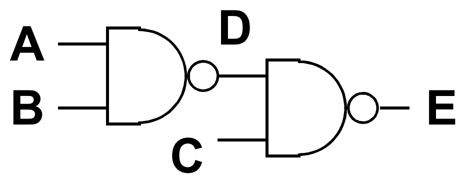
- What is simulation?
- Design verification
- Circuit modeling
- True-value simulation algorithms
    - Compiled-code simulation
    - Event-driven simulation
- Summary

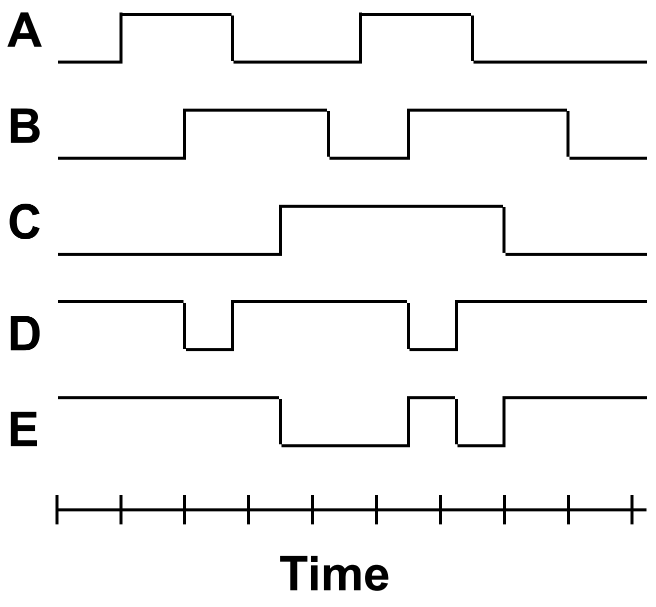
Simulation Defined

- 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

What is Logic Simulation?

- Simulate temporal behavior of logic design
- Logic design description
  - *netlist, network*
  - components
    - e.g. AND, OR, RAM, Pentium
  - component interconnections
- Logic models
  - component behavior
  - interconnect behavior
  - signal values
- Timing models
  - component behavior
  - interconnect behavior
  - signal delays

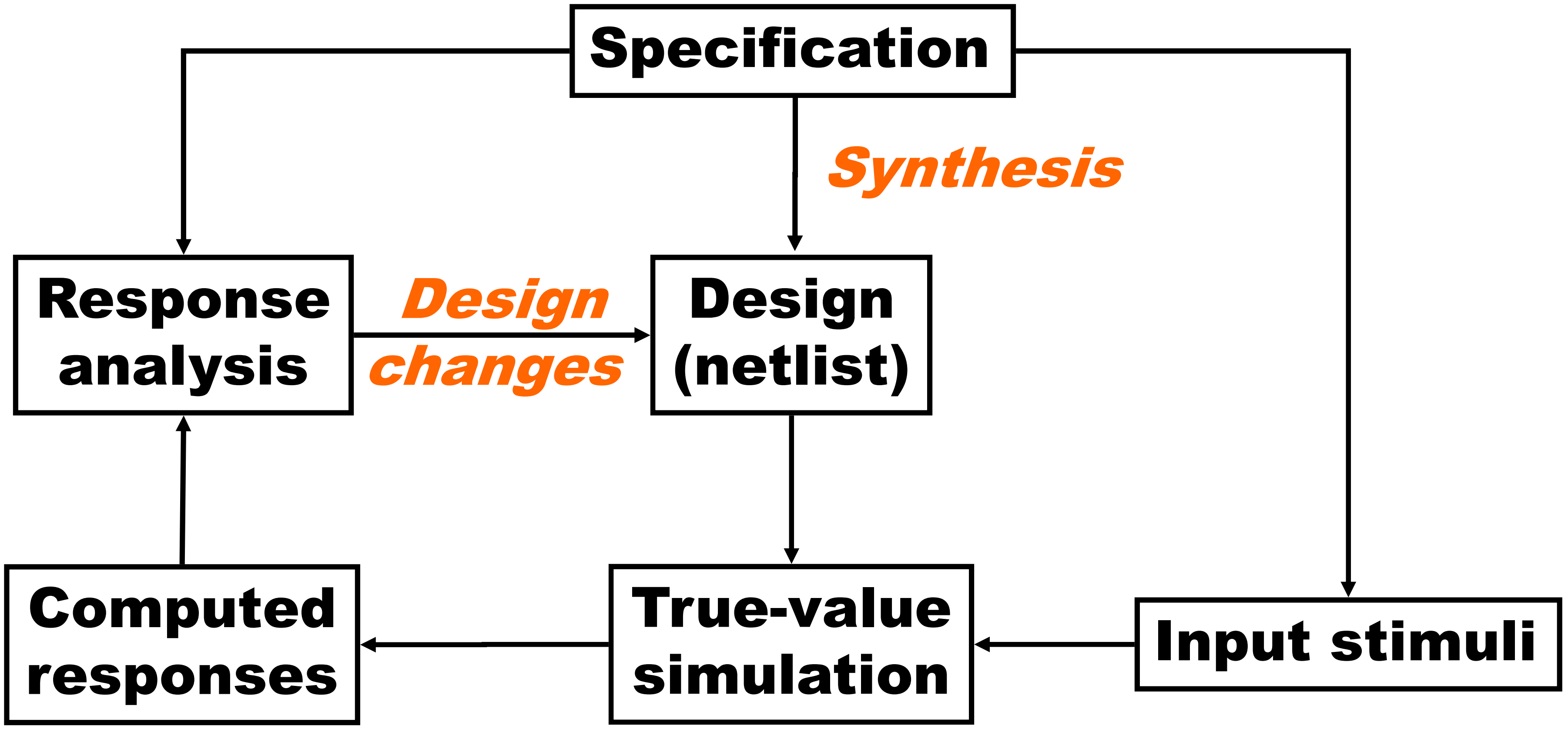




Logic Simulation Goals

- Functional correctness
  - circuit does what you want
  - validate by using lots of input stimuli
- Performance
  - circuit runs fast enough
  - no hazards or races
  - validate using lots of stimuli
- Test generation
  - simulate faulty circuits
  - does input stimulus cause faulty output?

Simulation for Verification

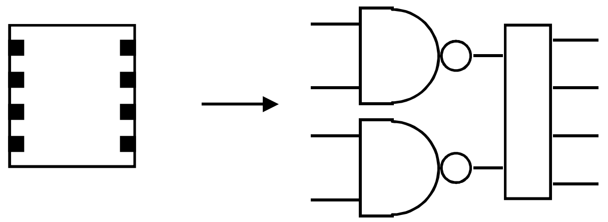


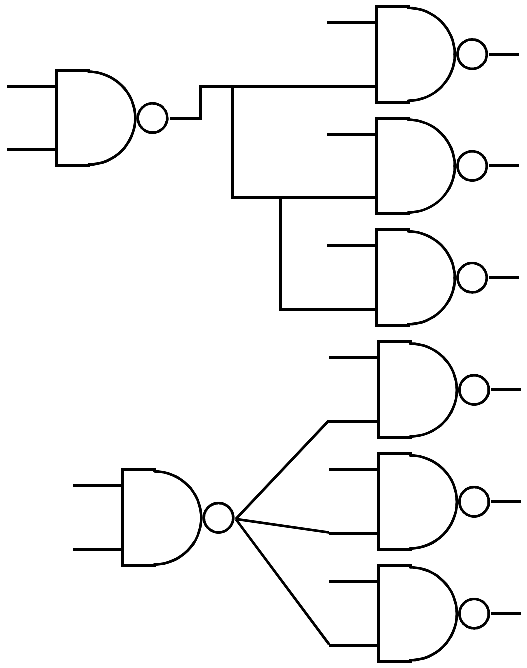
Modeling for Simulation

- 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.

Logic Design Description

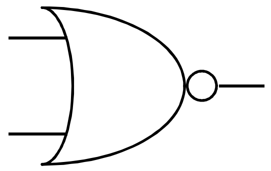
- Components
  - *modules*, *cells*,...
  - primitive - e.g. AND, OR, NOT
  - predefined - from library
    - functional behavior
    - timing behavior
  - composite - user-defined
    - *subnetwork*
    - hierarchy
- Component connections
  - wiring - *nets*
  - attachment points - *pins*, *ports*, *terminals*
  - can include wiring structure
    - fan-in, fan-out
    - parasitics

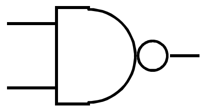




Logic Models

- Logical Primitive
  - Boolean logic operations
    - AND, OR, NAND, NOR, NOT, XOR, XNOR
  - often special-cased in simulator for speed
- Behavioral Model
  - finite state machine
    - outputs are function of inputs, next state
  - often supplied by model library vendors
    - implementation is secret
- Subnetwork
  - composed of primitives, behavioral models, subnetworks
  - use hierarchy, regularity in logic design

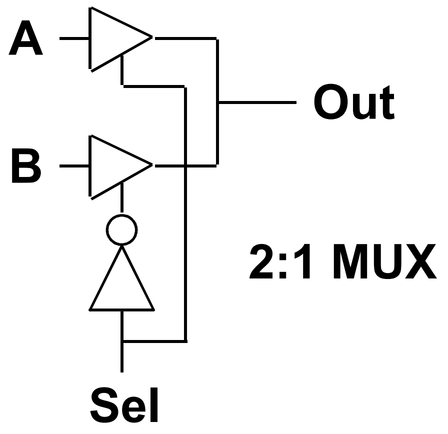


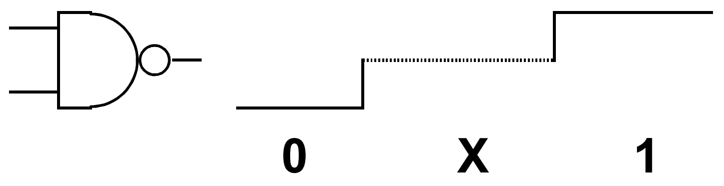


C = f(A,B)

Logic Values

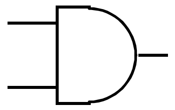
- Component output values - *states*
- 2-state simulation
  - 0 - logical 0
  - 1 - logical 1
- 3-state simulation
  - add X state
    - unknown, uninitialized, intermediate
- 5-state simulation
  - add rising, falling states
- Other states
  - Z - high impedance - for buses
  - U - unknown, but 0 or 1
    - X for intermediate value
  - D, D' - fault signals - for fault simulation
  - more states => closer to analog simulation





Logic Model Implementation

- Truth table
  - list of all input/output choices
  - fast - table lookup evaluation
    - use 0, 1, 2 for 3 states
  - impractical for many inputs, logic values
- Latch
  - truth table with knowledge of previous state
- Logic equations
  - can be compiled and executed fast
  - must support value system
- Behavioral description
  - HLL description of input/output relationship
    - hardware description language - Verilog, VHDL
    - general-purpose language
  - usually precompiled for speed



	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

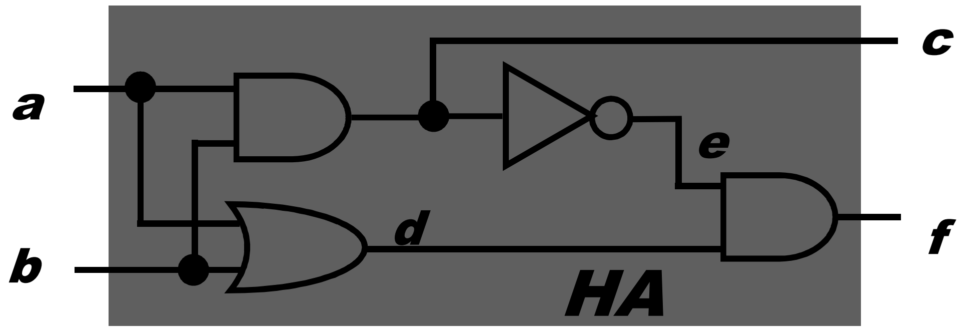
```
if (gate == and2)
    out = in1 && in2;
else if (gate == or2)
    out = in1 || in2;
```

Hardware Description Languages

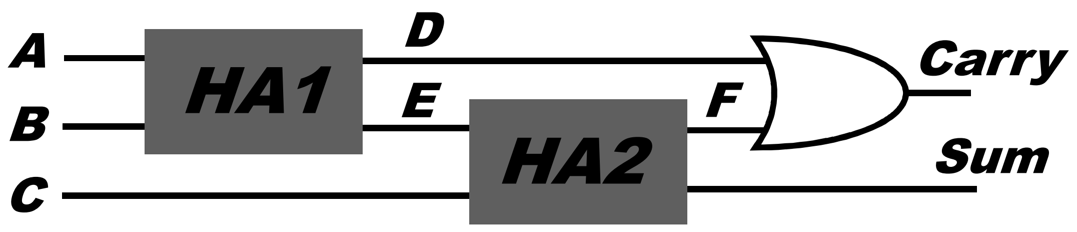
- Special-purpose languages
  - special constructs for hardware description
  - timing, concurrency, bit vectors, etc.
  - can usually include structural description
  - examples
    - Verilog
    - VHDL

- Standard programming languages
  - add data types and objects for hardware description
  - examples
    - C
    - C++
    - Matlab

Example: A Full-Adder

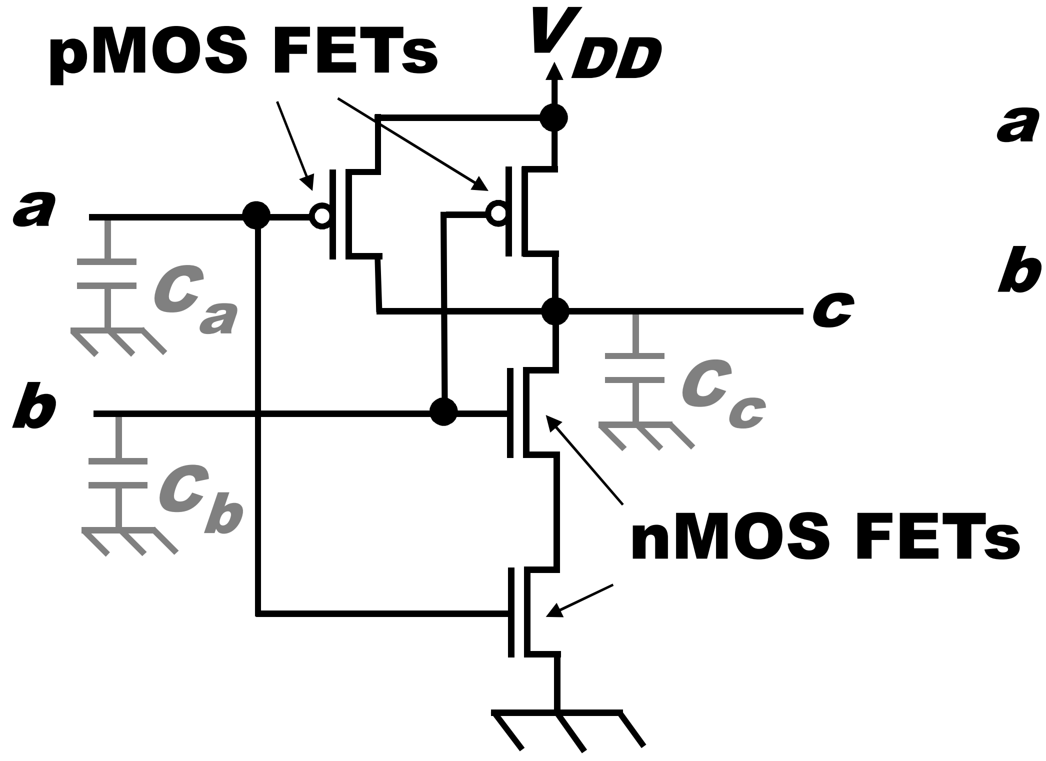


HA;
inputs: a, b;
outputs: c, f;
AND: A1, (a, b), (c);
AND: A2, (d, e), (f);
OR: O1, (a, b), (d);
NOT: N1, (c), (e);

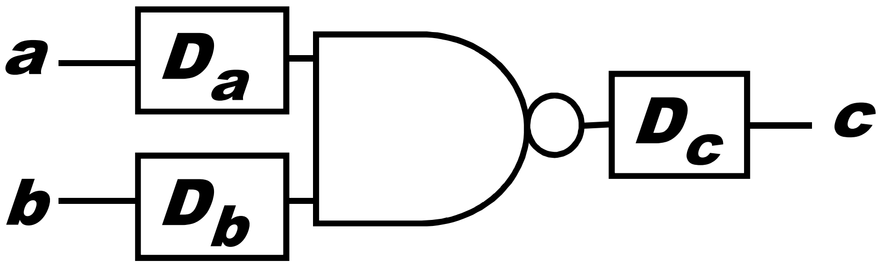


FA;
inputs: A, B, C;
outputs: Carry, Sum;
HA: HA1, (A, B), (D, E);
HA: HA2, (E, C), (F, Sum);
OR: O2, (D, F), (Carry);

Logic Model of MOS Circuit



C_a , C_b *and* C_c are
parasitic capacitances

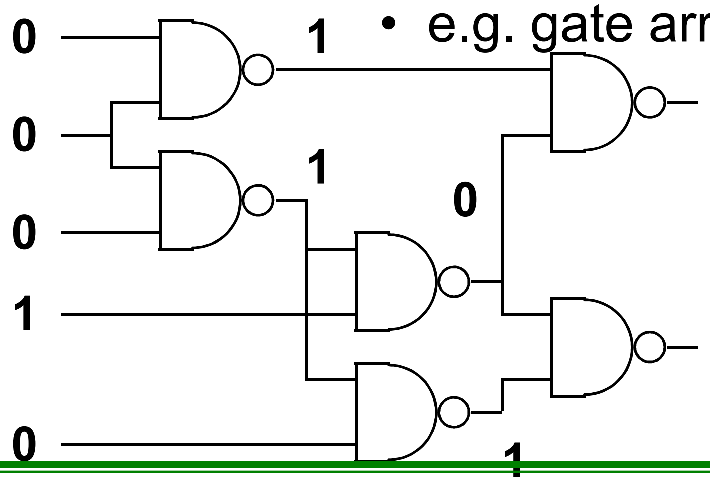


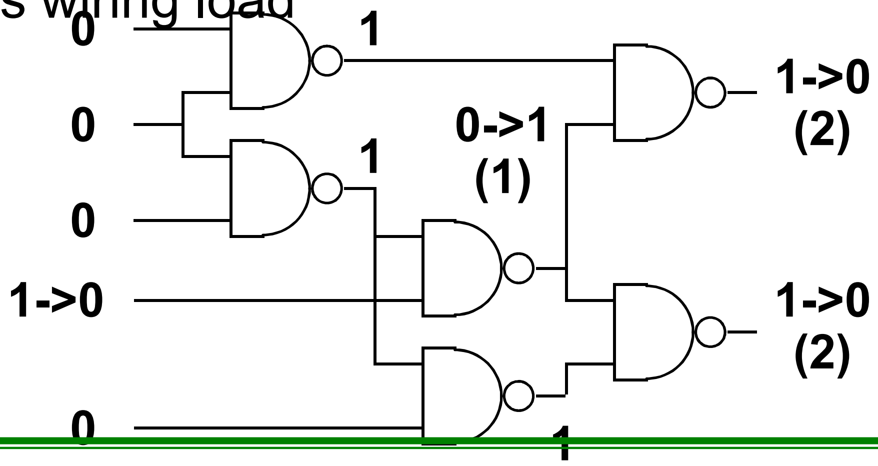
D_a **and** D_b **are interconnect or propagation delays**

D_c **is inertial delay of gate**

Component Timing Models

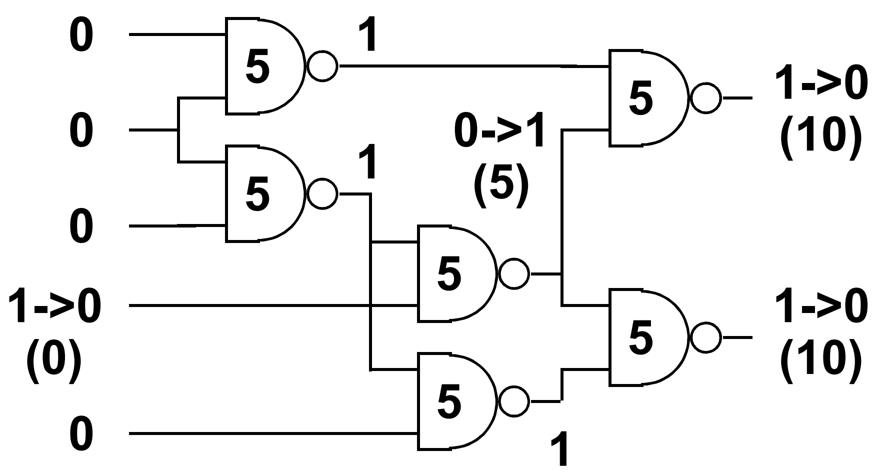
- Zero delay
  - no delay from gate inputs to outputs - a boolean equation
  - good when just considering functional behavior
    - e.g. combinational fault simulation
- Unit delay
  - gate delay is one unit
  - appropriate when all gates are same delay
    - e.g. gate array, ignores wiring load

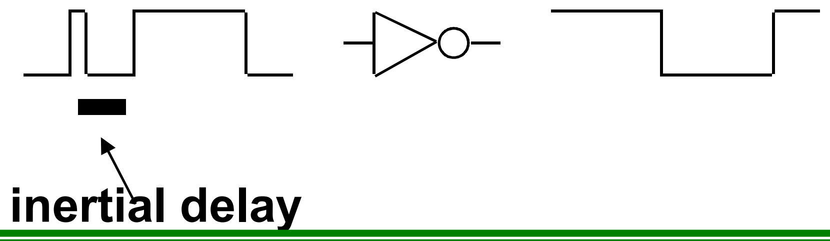




Component Timing Models

- Propagation delay
  - fixed delay from inputs to outputs
  - can vary with fan-out, output load
  - delay = time or multiple unit delays
  - varies among gate types
    - XOR vs. inverter
  - varies between gate instances
    - manufacturing variations
  - make output unknown for interval
    - rise/fall, min/max delay
- Inertial delay
  - delay time before gate responds
  - input value must be present long enough for the gate to respond
  - gates are low-pass filters

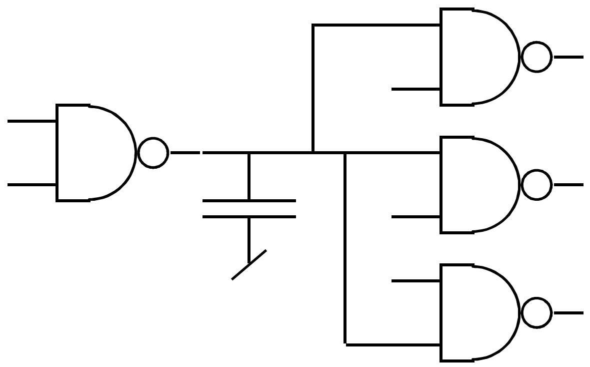




Component Timing Models

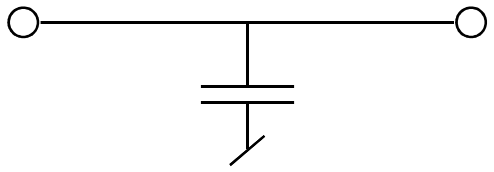
- Output Loading Delay
  - gate delay is function of load it drives
  - wiring capacitance
  - type of gate
    - e.g. small vs. driver
  - number of fan-outs
  - types of fan-outs
  - depends on interconnect delay model
- Compute prior to simulation

PropDelay = f(load)

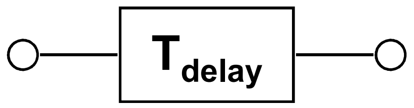


Interconnect Timing Models

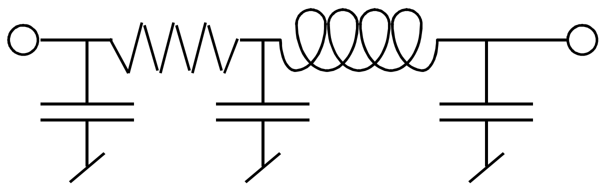
- Isochronic
  - zero delay
  - interconnect acts as capacitor
  - all delay assigned to driving gate



- Wave propagation
  - transmission line
  - fixed delay
    - function of wire "length"
  - can model as different propagation delay times for different fan-outs

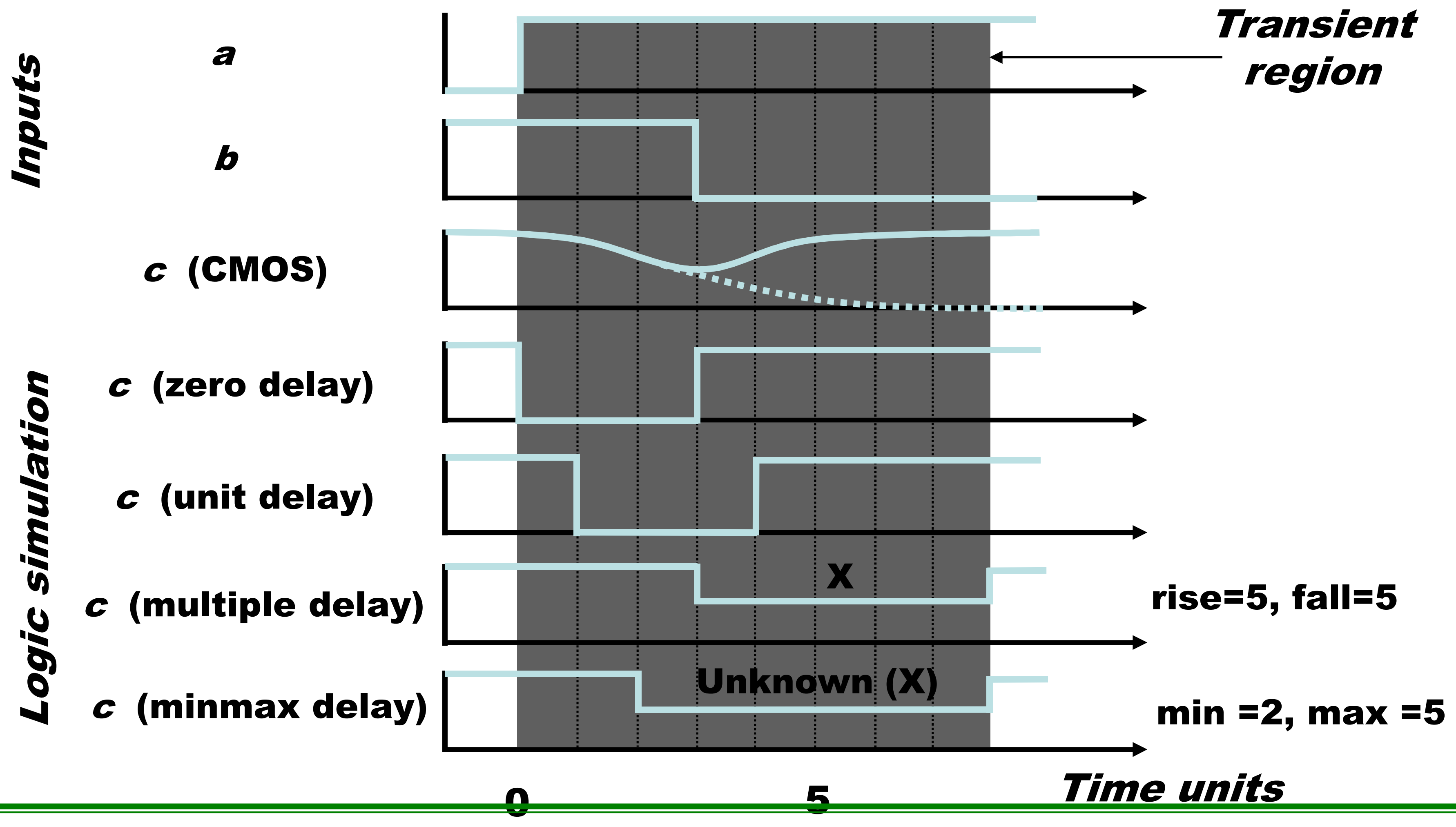


- Signal diffusion
  - distributed RLC parasitics
  - inertial delay
  - usually combine with driver timing model



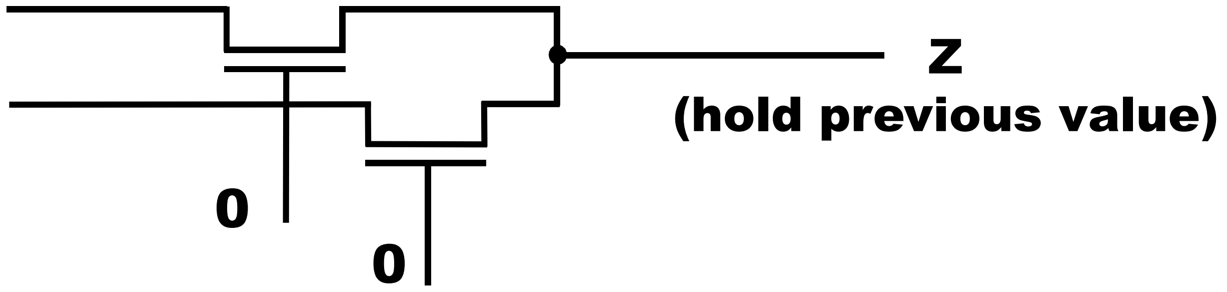
Options for Inertial Delay

(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

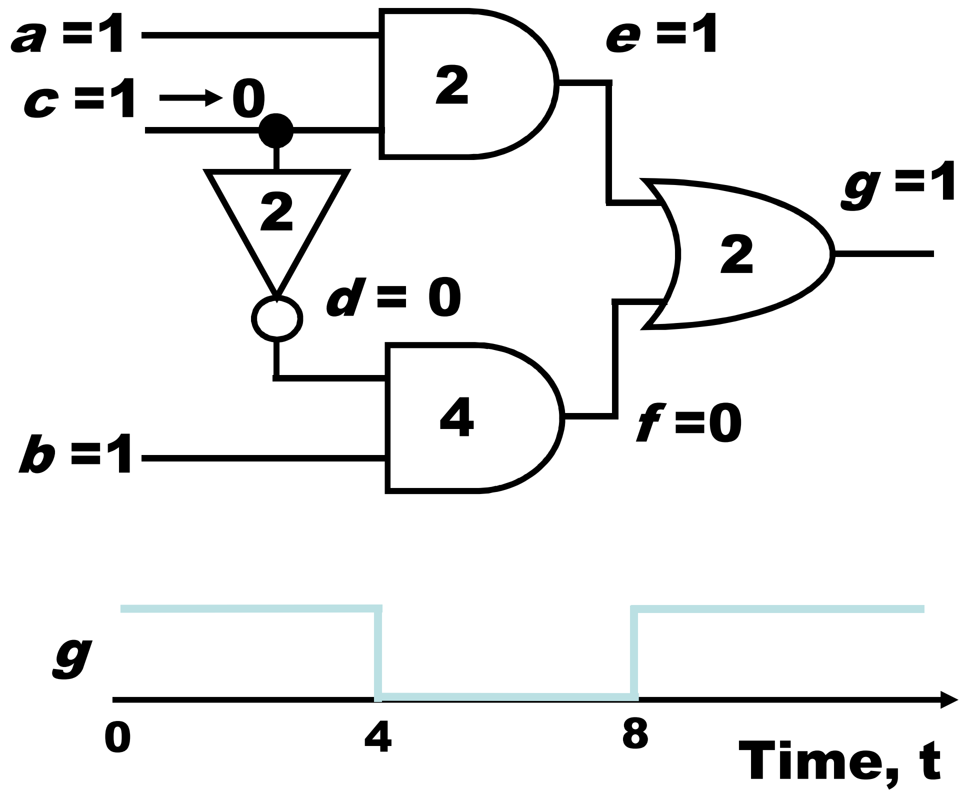
True-Value Simulation Algorithms

- 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

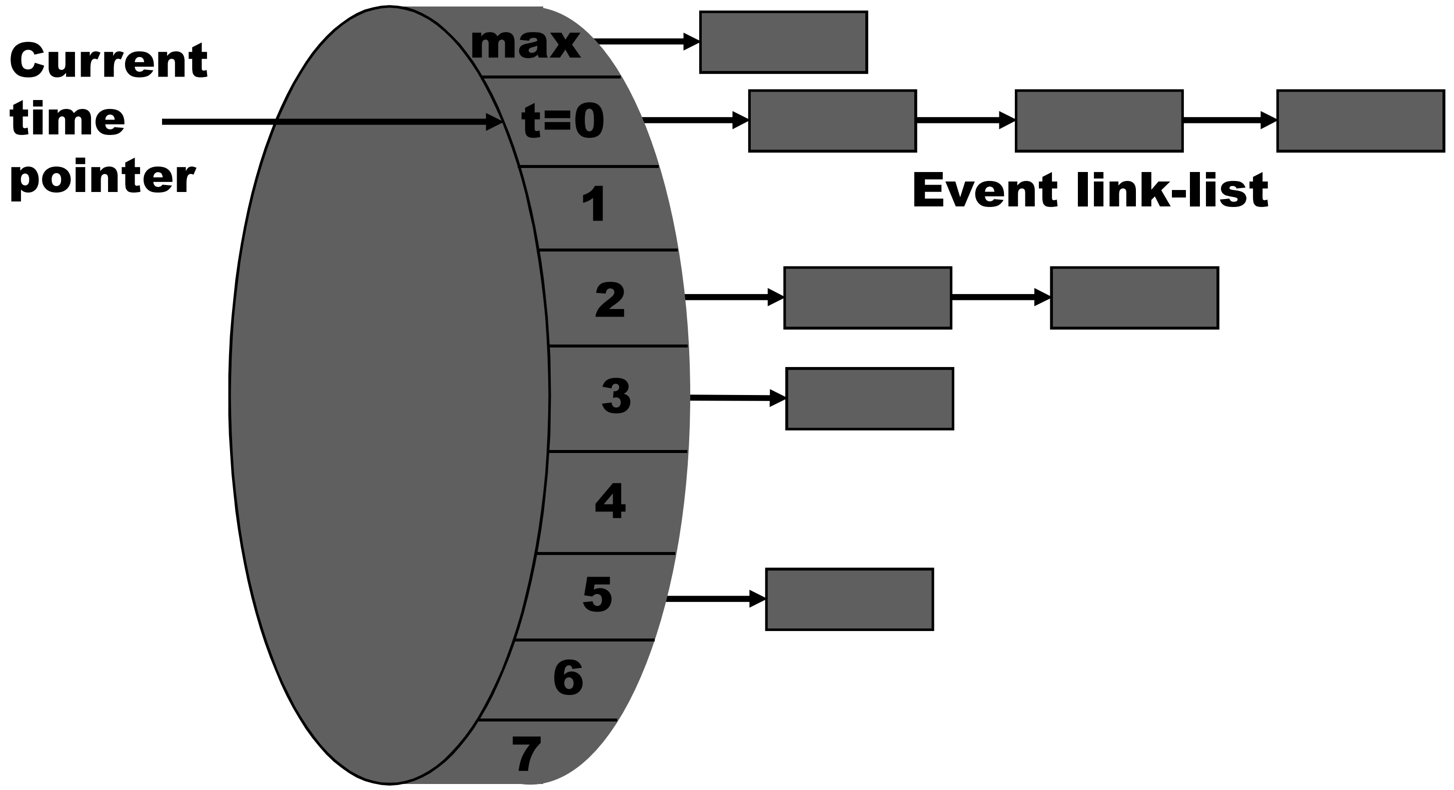
- Step 1: Levelize combinational logic and encode in a compilable programming language
- Step 2: Initialize internal state variables (flip-flops)
- Step 3: For each input vector
  - Set primary input variables
  - Repeat (until steady-state or max. iterations)
    - Execute compiled code
  - Report or save computed variables

Event-Driven Algorithm
(Example)



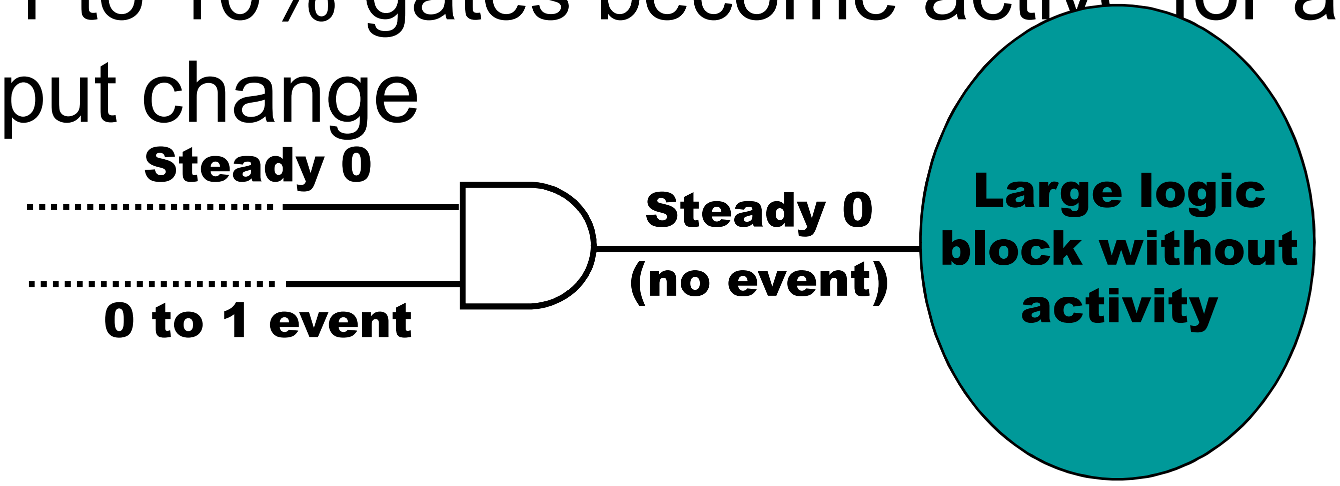
Time stack	Scheduled events	Activity list
t = 0	*c* = 0	*d, e*
1		
2	*d* = 1, *e* = 0	*f, g*
3		
4	*g* = 0	
5		
6	*f* = 1	*g*
7		
8	*g* = 1	

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.

Questions?