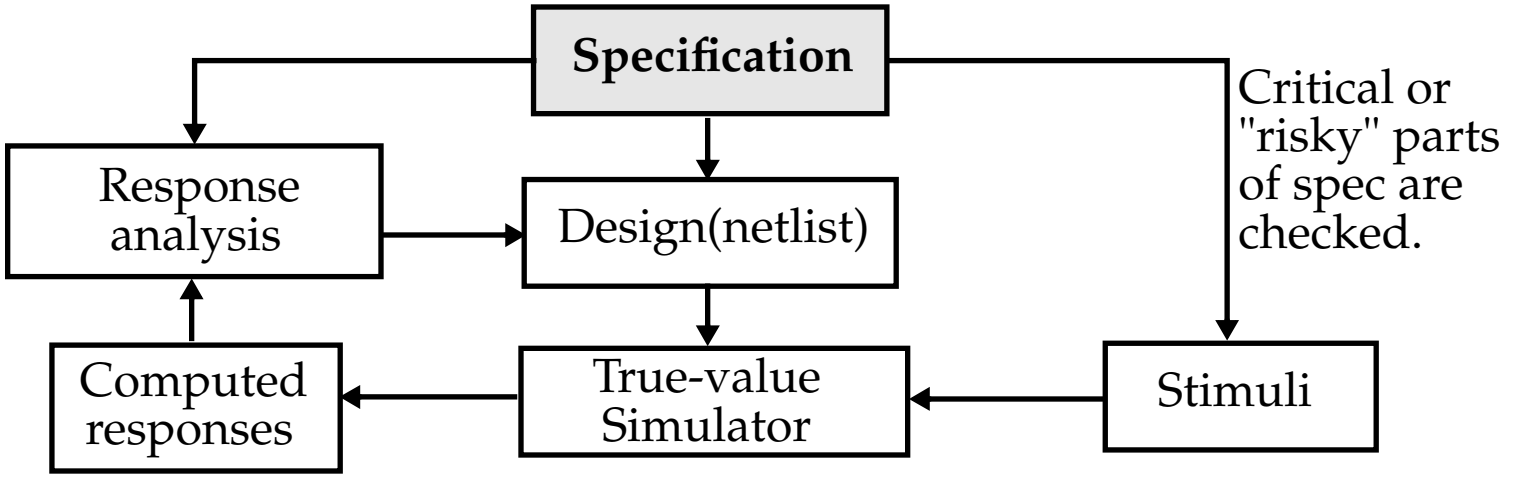
Design Verification

Simulation used for 1) *design verification*: verify the correctness of the design and 2) *test* verification.

Design verification:



- Adv. include ability to verify at multiple levels of design abstraction, e.g., RTL, logic, switch, circuit for different purposes, e.g., timing, function.
- Disadv. include lack of a guarantee that the design conforms to spec.

Formal verification mathematically proves the correctness of a design, but is only applicable in limited forms at higher levels of abstraction.

Design Verification

Design verification involves verifying

- *Function*
- *Timing* (**Static Timing Analysis** can also be used instead of simulation)

The verification vectors needed for each of these are usually different.

For example, the test for a critical path such as the carry in a ripple carry adder is not likely to be part of the test set designed to test the function.

Design verification patterns are often used in manufacturing test because

- They are already available (although it is non-trivial to translate them to a production tester).
- They may provide high fault coverage.

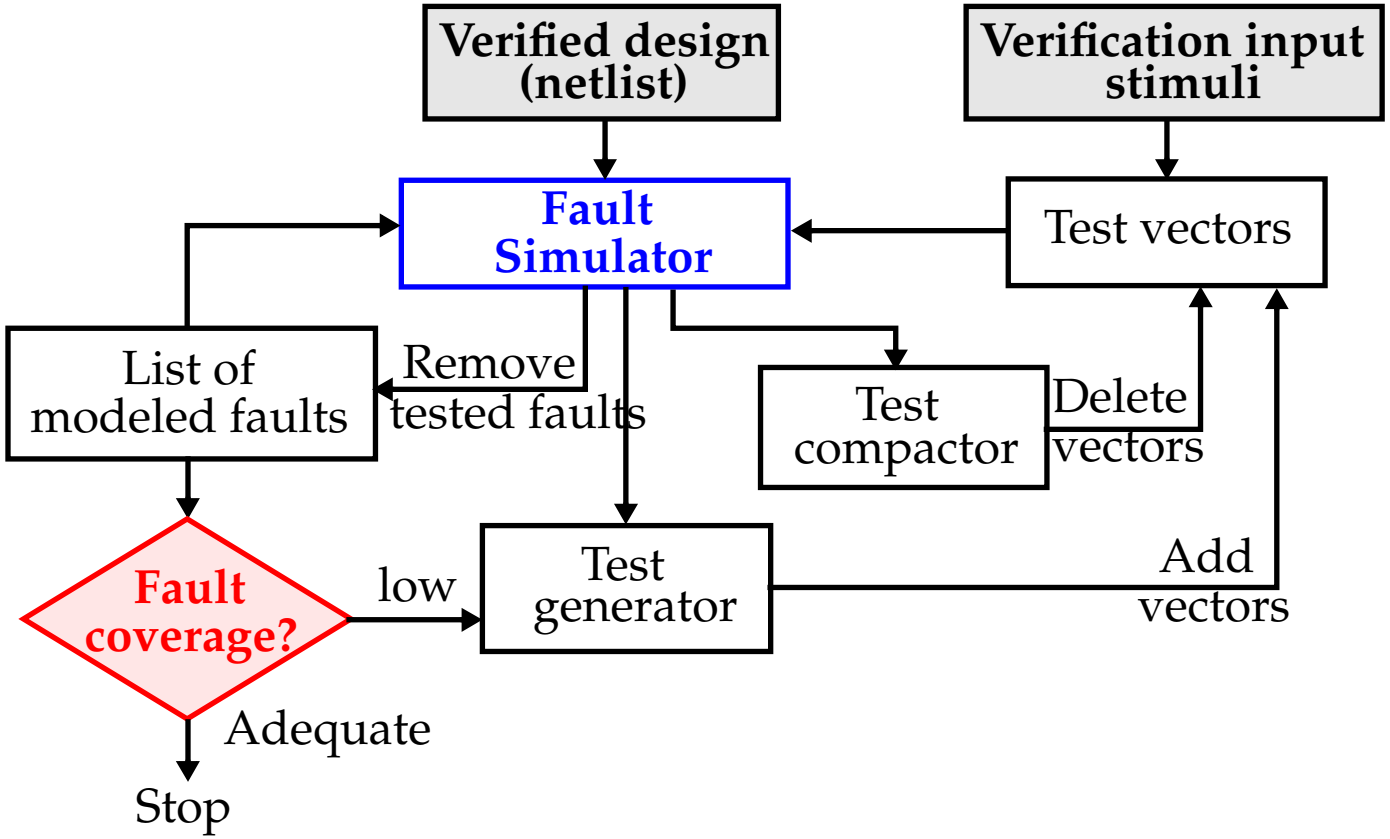
  If not, they are often augmented with ATPG patterns.

The functional test patterns for **sequential circuits** are more likely to possess lower fault coverage.

Particularly when the specification of the transition graph is *incomplete*, making it difficult to use stategies such as "test all transitions in the state diagram".

Test Evaluation

A fault simulator is used in the development of manufacturing tests:



Verification patterns are used as input to fault simulator and their coverage under some fault model is determined.

Test Evaluation

The fault simulator can also, with the help of a test generator, produce a set of vectors with a given fault coverage for manufacturing test.

If no fault list is supplied, the fault simulator will generate the **fault list** for the specified fault model.

The fault simulator result can also be used for *compaction* -- removal of vectors that do not detect any additional faults.

Fault dropping is typically performed during this process.

This causes faults detected by each vector to be removed, i.e., they are not considered in the fault simulation of the remaining patterns.

Fault dropping makes it **impossible** to determine the "overlap" in fault coverage among the vectors.

Having this information allows the "best" vectors to be selected and can further reduce the test set size.

Modeling Levels and Simulation Types

Focus of fault analysis is mainly at the logic and switch levels.

Modeling Level	Circuit Description	Signal Values	Timing Resolution	Application
Function, behavior or RTL	VHDL, verilog	0, 1	Clock boundaries	Architecture and functional verification
Logic	Gates and transistors	0, 1, X, Z	0/unit/multiple-delay	Logic verification and test
Switch	Transistor connectivity, node caps	0, 1, X	0-delay	Logic verification
Timing	Transistor and tech data, node caps	analog voltage	fine-grained time	Timing verification
Circuit	Active and passive components, tech data	analog voltage/current	continuous time	Digital timing and analog verification

Signal States

Pure combinational logic can be modeled with two states, [0,1].

The internal states of sequential circuits are unknown at power up.

Even after power up and initialization, unknown states occur.

X is used to represent them:

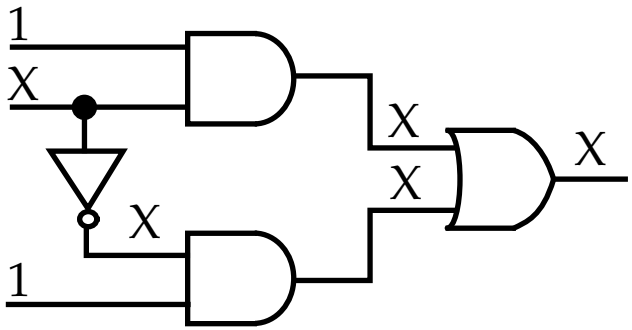
Inputs	**Output**		
a/b	**AND**	**OR**	**NOT($\bar{a}$)**
0/0	0	0	1
0/1	0	1	1
1/X	0	X	1
1/0	0	1	0
1/1	1	1	0
1/X	X	1	0
X/0	0	X	X
X/1	X	1	X
X/X	X	X	X

Other gates can be represented using these three types.

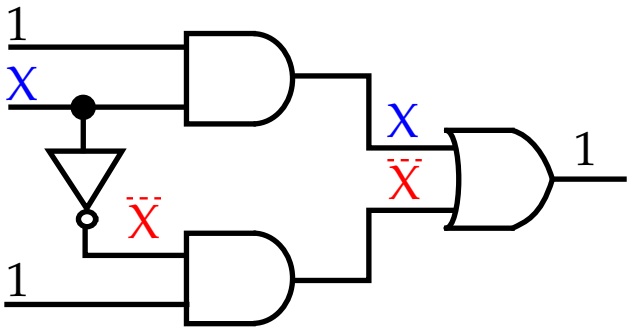
Signal States

3-state logic is pessimistic.

Here, the output of the mux can be uniquely determined if a **symbolic** simulation was performed (not practical for large circuits).

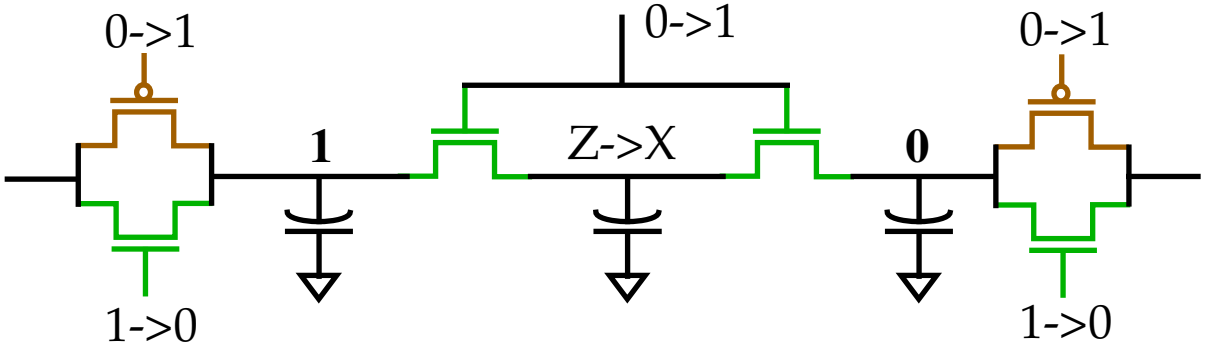


3-state simulation



Symbolic simulation

MOS circuits require a 4th state:

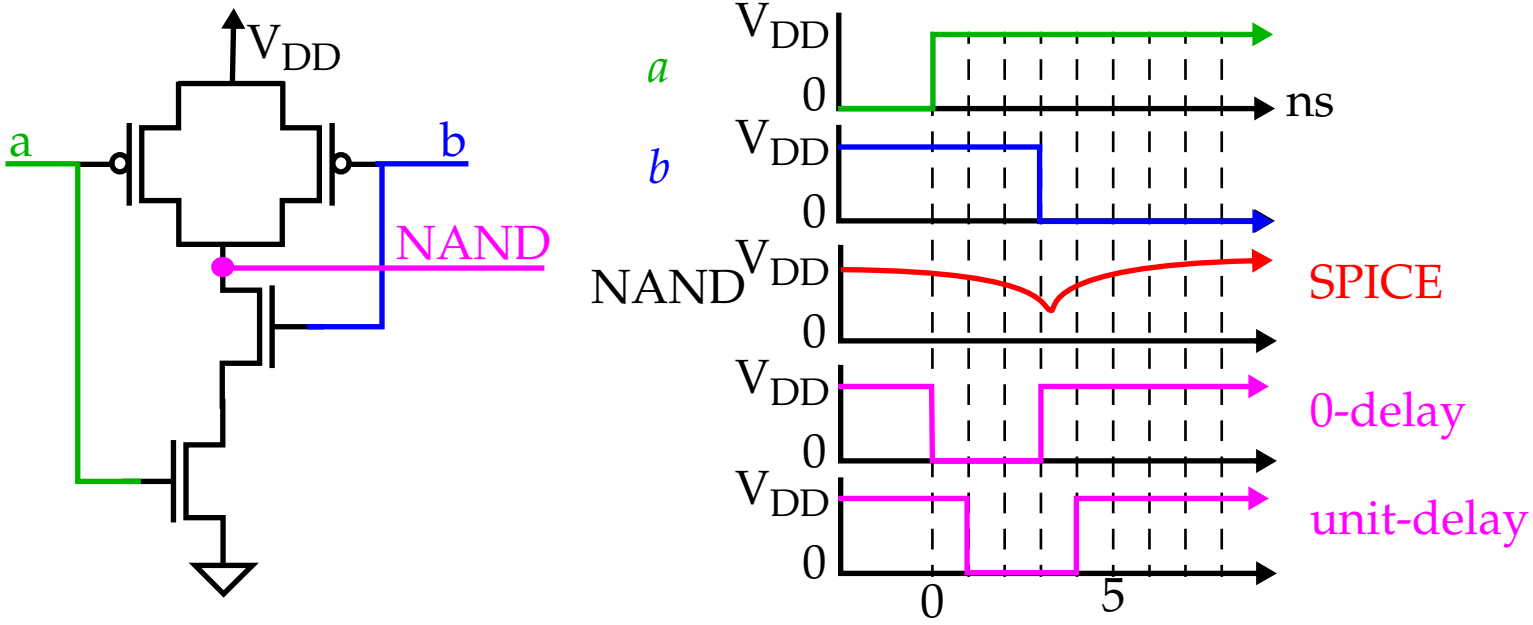


Z interpreted as state before node floated or X if charge sharing occurs.

Timing

Signals experience two types of delays

- **Inertial delay**: Time interval between an input change and output change of a gate.
- **Propagation delay** (transport delay): Time interval between output change and arrival at the input of a gate.



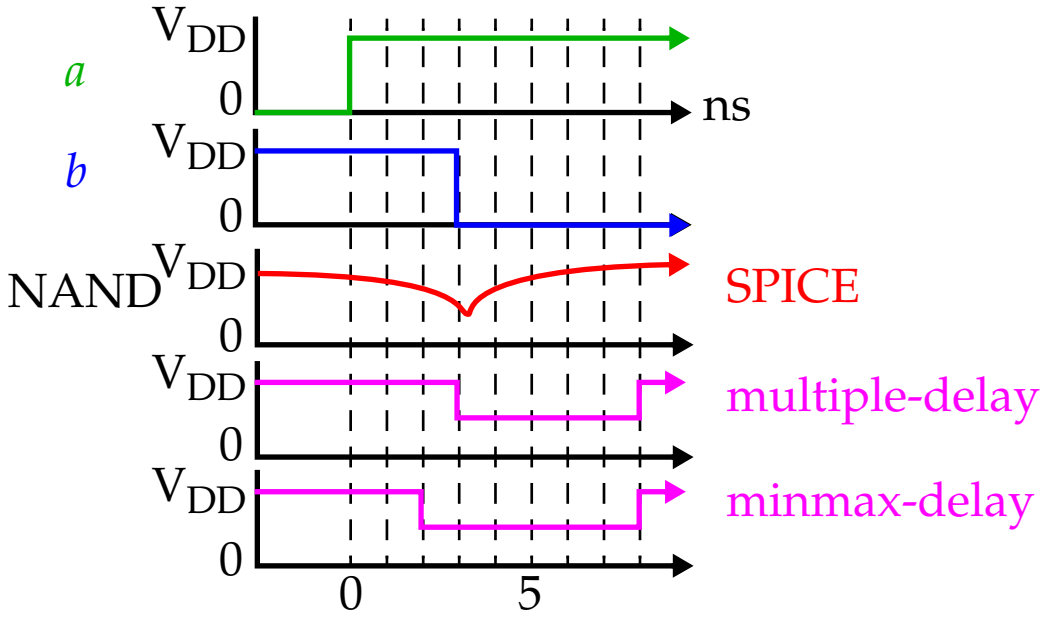
Unit delay: All gates have one unit of delay.

This allows circuits with feedback to be simulated since the proper sequencing of signals is maintained.

Timing

Multiple-delay: All delays modeled as multiples of some time unit, e.g. 1 ns.

Each gate has a **rising** delay, d_r, and a **falling** delay, d_f, which is the delay from input to output change.



Here, $d_r = d_f = 5$ns. *b* falling at time 3 indicates output will be 1 at 8ns. At time 3, simulator indicates output is unknown, *X*.

Minmax-delay: Statistical model that uses d_{min} and d_{max} to account for process variations.

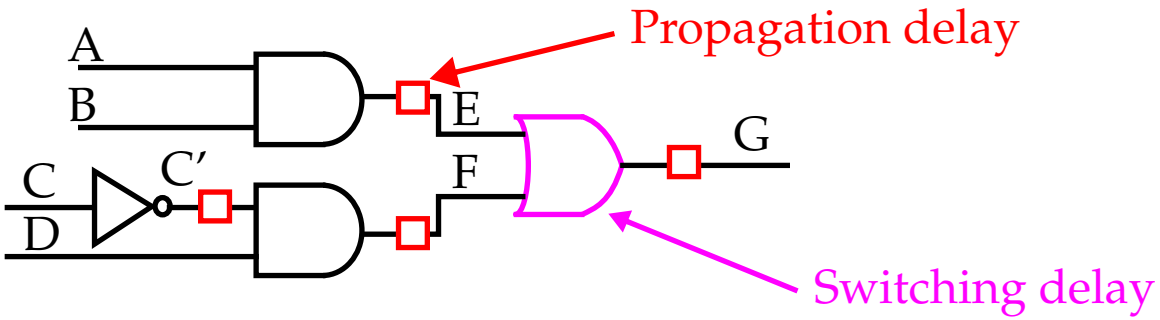
Here, d_{min}=2 and d_{max}=5 which results in ambiguity interval (2,5).

Timing

Transmission lines: Interconnect gives rise to delay, i.e., gate output does not instantaneously change "driven" gate inputs.

Propagation delay can be implemented at gate inputs to allow separate modeling of delay at each fanout branch.

Propagation delay can also be modeled by treating the entire fanout net as a circuit element, similar to the treatment of gates.



Separate rise and fall propagation delays can be modeled yielding up to 8 different delay conditions for a 2-input gate.

Algorithms for True-Value Simulation

Def: Simulation is the process of computing a circuit's signals as a function of time.

For digital circuits, only certain discrete values of signals are meaningful, i.e. the transients can be skipped.

Discrete event simulation

Time is advanced in discrete "jumps" and signals acquire values from a meaningful set.

The change of a signal from one value to another is called an **event**.

Compiled Simulation

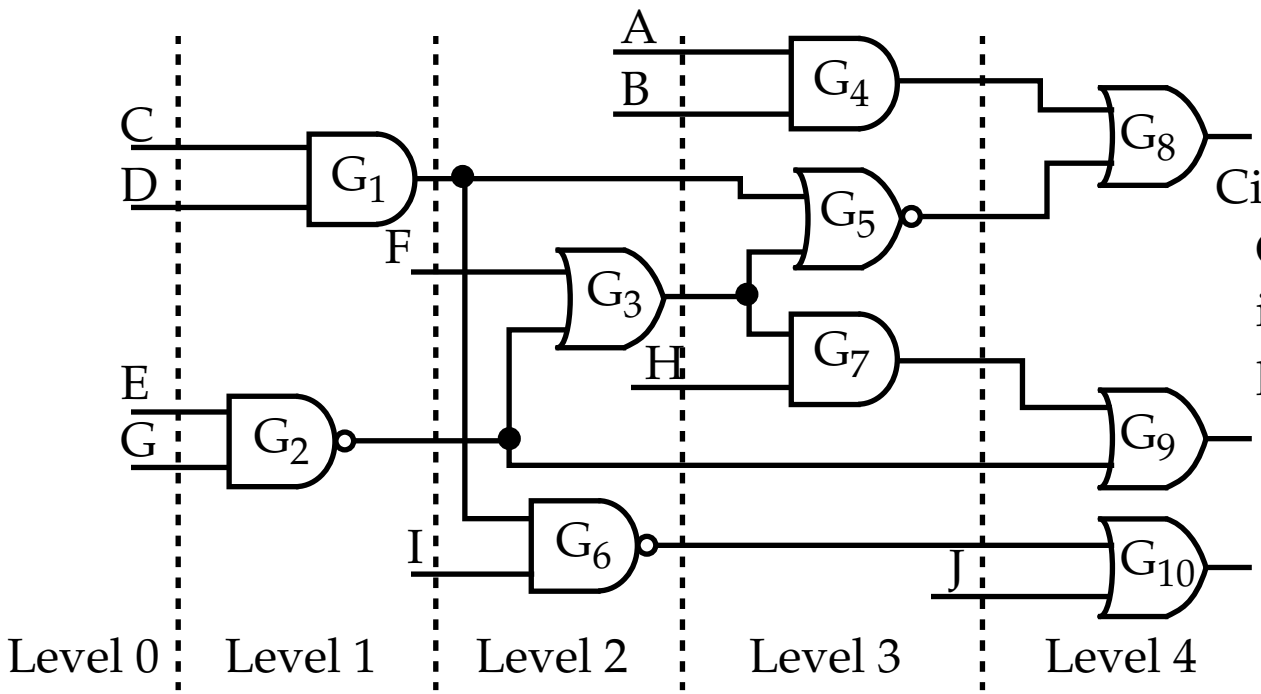
Circuit is described in an HDL.

It is levelized and converted into an executable.

Levelization ensures that the inputs are evaluated before the output.

Compiled Simulation

Levelization example:



Circuit is levelized. Gates at a level have inputs **only** from lower levels

Signals treated as variables, gates translated to opcodes for AND, OR, etc.

For each vector, code is repeatedly executed until steady state is achieved (handles feedback).

Compiled Simulation

Step 1: Levelize circuit and produce compiled code.

Step 2: Initialize data variables (FFs and other memory).

Step 3: For each input vector

Set PI variables

Repeat until steady-state or maximum iteration count reached

Execute compiled code.

Report or save variable values.

Adv:

Good when 2-state (0,1) simulation is sufficient, e.g., high level design verification.

It's fast!

Disadv:

Recompilation needed for design changes.

All nodes are evaluated including gates with steady-state values.

Typically, only 1%-10% of the gates actually change state.

Multiple delay, min-max delay are difficult to implement.

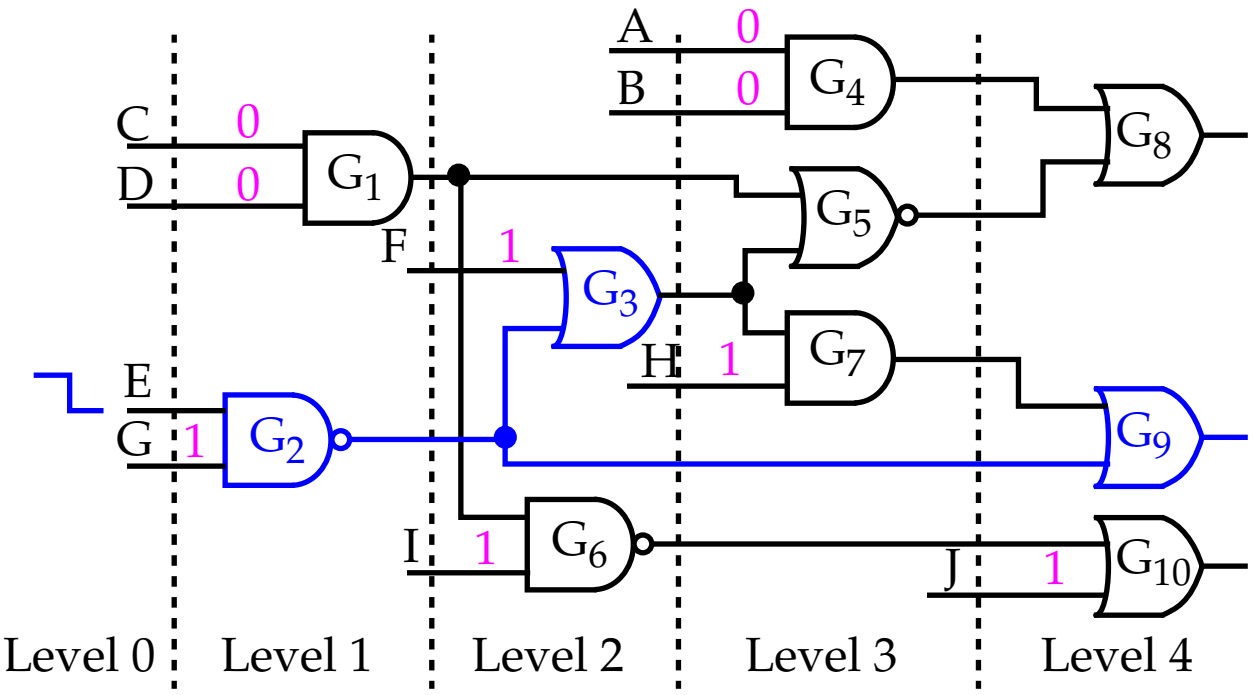
Cannot model glitches, race conditions, i.e. timing problems.

Event-Driven Simulation

Good match for discrete-event simulation.

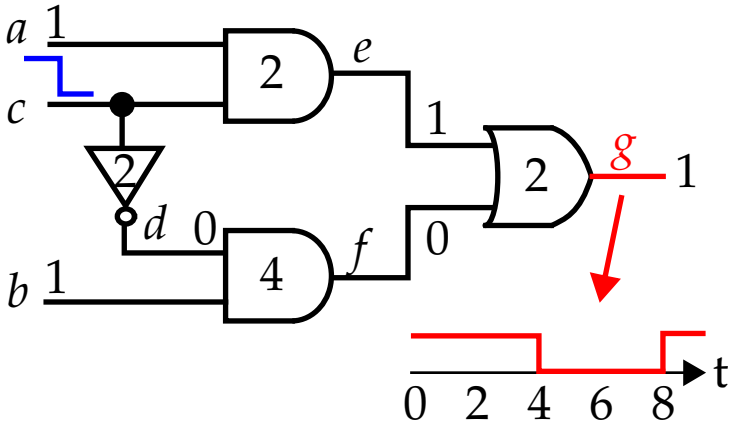
Events (changes in signal values) cause new events (future changes in other signal values).

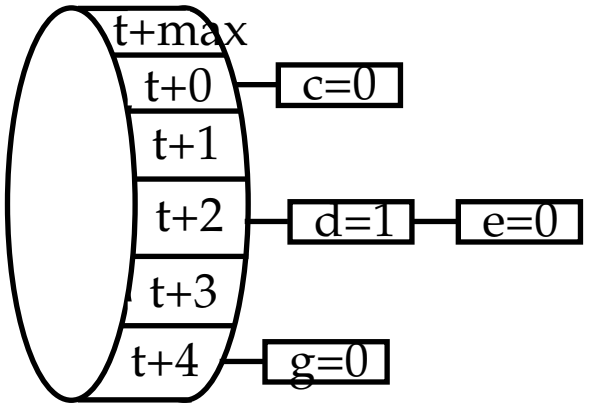
Evaluation takes place only if an input to a gate changes.



Event-Driven Simulation

Gates whose inputs change go on the *activity list*.





Timing wheel

Activity list

d, e

f, g

gates driving these outputs

Simulation involves evaluating a gate on the activity list.

If output changes, then gates at fanout added to *activity list*.

Adv:

Computationally efficient.

Ability to simulate arbitrary delays via *event scheduling*.

Event scheduler responsible for distributing events to the appropriate list.