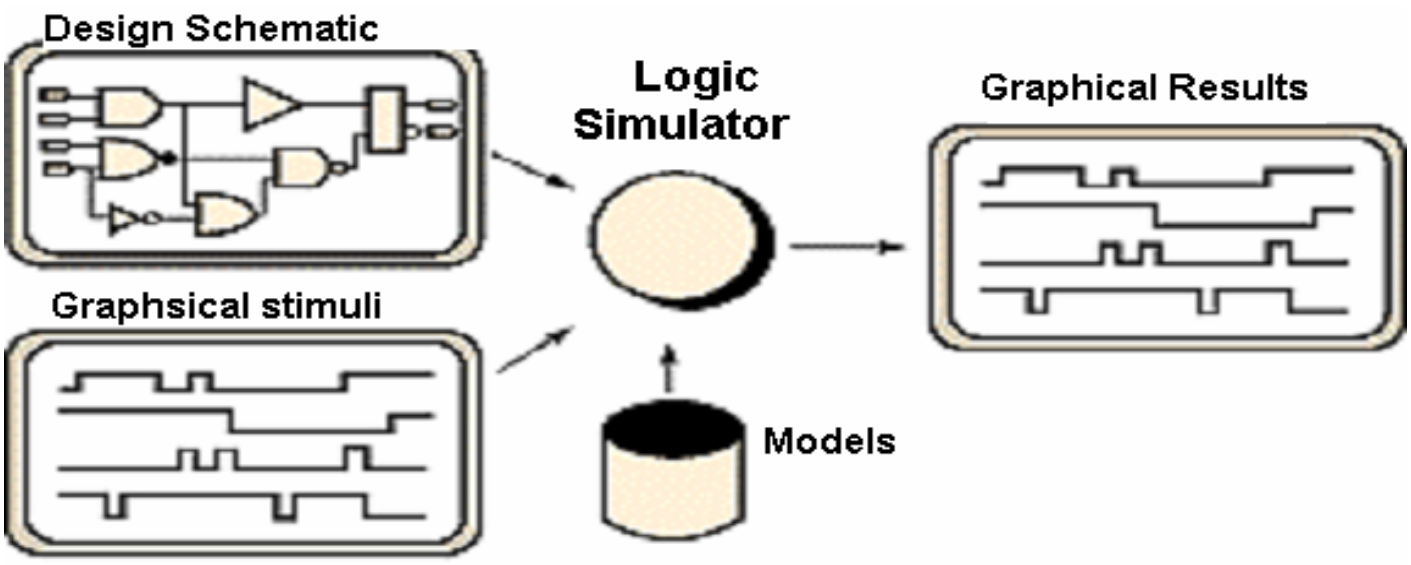
Unit 7: Simulation

- Content
  - Circuit Simulation
  - Switch-Level Simulation
  - Gate-Level (Logic) Simulation
  - RTL-Level (Cycle-Based) Simulation
  - Behavior-Level Simulation
  - Instruction Set Simulation
  - System-Level Simulation
    - Hardware-Software Co-simulation
    - Mixed-Level Simulation
    - Mixed-Signal Simulation
  - Timing-Simulation
- Reading
  - Chapter 10

Simulation

- **Simulation** is a design validation process for checking a circuit's function, timing, etc., encompassing from the lowest through the highest design levels.
- Simulation makes a computing model of the circuit, executes the model for a set of input signals (stimuli, patterns, or vector), and verifies the output signals.



Why Simulation Tools?

- Murphy's Law: “Anything that can go wrong, will!”
- Circuits are too large and complex!
- Hard to fix a chip!
- Long manufacturing turnaround time!
- Although it can not guarantee 100% validation (except that one can exhaust all the input stimuli), it is easiest and direct way to validate a circuit’s function or timing.

Types of Simulation

- Device-level simulation
  - tests the effect of fabrication parameters
- Circuit-level simulation
  - detailed analysis of voltage and current
- Switch-level simulation
  - treats transistors as switches.
- Gate-level simulation
  - uses gates as the basic elements.
- Register-transfer-level (RTL) simulation
  - register + combinational logic.
- Behavior-level simulation
  - describes designs in higher level abstraction such as by an algorithm, a data flow graph, etc. using hardware description languages or high level languages.

Types of Simulation (cont'd)

- Instruction set simulation
  - verifies the design of a CPU and evaluate its performance.
- System-level simulation
  - verifies the function of a system whose constituent components could be everything, electrical, mechanical, optical elements, etc.
- Timing simulation
  - Inspects timing behavior based on a given timing model for logic components and interconnects.
- Fault simulation
  - checks whether a given set of test vectors could attain a certain level of fault coverage for production test.

Device, Circuit, Timing Simulation

- Device-level simulation
  - Used to test the effect of fabrication parameters
  - Used by technologists, not by circuit or system designers
- Circuit-level simulation (e.g., SPICE)
  - Analog
  - Nodal/tableau equations: KCL, KVL laws
  - Numerical integration
- Timing simulation
  - Intrinsically analog, use a circuit simulator such as SPICE to obtain signal waveforms for compute timing,
  - But simplifications using macro models, look-up tables, piecewise-linear models, etc. for tackling large designs.

Switch, Gate, RTL Simulation

- Switch-level simulation
  - Transistors are modeled as bidirectional switches
  - Mainly digital
  - Circuits extracted from mask patterns can directly be simulated
- Gate-level (or logic) simulation
  - ‘‘Gate’’ mainly refers to elements found in a component library (e.g. for standard-cell design)
    - NAND, NOR, multiplexer, D-flip-flop, latch, etc.
  - Unidirectional signal flow
  - Closely related to ‘‘fault simulation’’
- Register-transfer-level (RTL) simulation
  - Circuit is seen as composed of registers to store the state and combinational logic to compute the next state (finite state machine model)

Behavior-Level Simulation

- Its goal is to achieve highest simulation speed.
- Designs are described in higher-level abstraction using HDL languages, e.g. VHDL (VHSIC Hardware Description Language), Verilog, System C, C, C++, etc.
- The behavior model focuses more on verifying the function of a design rather than the timing performance of the design, i.e., we don’t even have to know how long the design will take to perform its function.

Instruction Set Simulator

- Its goal is to verify the design of a particular CPU and evaluate its performance under certain workload.
- The simulator is normally written in high-level languages such as C, C++, etc. to achieve highest execution speed.
- The program input to an instruction set simulator is an executable code segment. Programs written in assembly language or higher level languages should be compiled into machine code.
- It plays an important role in hardware/software co-simulation.

System-Level of Simulation

- System-Level Simulation:
  - Validates the whole system described in different design levels with different types of components.
- It might have to use a variety of aforementioned simulators integrated as a whole to validate a system. For example,
  - Mixed-level simulation: For the designs described in different levels of abstractions within the same simulation environment
  - mixed-mode simulation: For the designs with different kinds of signal abstractions such as digital versus analog signals, etc.
  - hardware-software co-simulation: For the designs with software running on a target hardware. It becomes more and more important

Circuit Simulation

Circuit Simulation

- Circuit simulators like **SPICE** numerically solve device models and Kirchoff's Voltage and Current Laws (KVL, KCL) to determine time(frequency)-domain circuit behavior.
  - KVL (Mesh Analysis): The algebraic sum of the voltage drops around a closed path is zero.
  - KCL(Nodal Analysis): The algebraic sum of all the currents incident on a node is zero.
- Unlike resistors and capacitors, transistors are **non-linear** devices $\Rightarrow$ shall apply numerical approaches for circuit simulation.
- Numerical solution allows more sophisticated models, non-functional (table-driven) models, etc.
- SPICE Home Page
  - http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/

Numerical Solutions

- For Nodal Analysis, the (non)linear system describing a circuit network can be formulated as follows:

$$\mathbf{I}=\mathbf{YU}$$

  - **I** is the vector consisting of *independent* current sources.
  - **Y** is the system matrix consisting of admittance terms from all components *except* from independent sources.
  - **U** is the vector of node voltages to be solved.
- The above equation can describe a huge system with hundreds or thousands of elements. However, the system matrix **Y** is usually *sparse* such that both memory usage and calculation time can be reduced with *sparse matrix* solution techniques.
- Its simulation time step is decided by the accuracy requirement of numerical algorithms for solving the system equations. The simulator can not predict the next simulation time point based on the current time.

Types of Circuit Analysis

- DC Analysis
  - It finds the operating point of a circuit.
- AC (Small-Signal) Analysis
  - It finds the frequency response of a circuit.
- Transient Analysis
  - It finds the time domain response for a circuit when it is excited with a sinusoidal signal.
- Harmonic Balance Analysis
  - find the harmonic steady state response for a circuits when it is excited with a sinusoidal signal.

DC Analysis

- All dynamic components such as capacitances and inductances are ignored in this analysis.
  - Capacitances are removed from the circuit and inductances are short circuited. If the circuit is nonlinear, the analysis is performed using an iterative method.

Building the netlist (a SPICE-like format)

#define RLoad 1k

Model EMoll ALPHA=0.995 ALPHAR=0.5 RB=25

+ FLAG=DIODE_BC CJO=22p RS=5

+ FLAG=DIODE_BE RS=1

Volt Vcc n1 0 DC=10 R=1

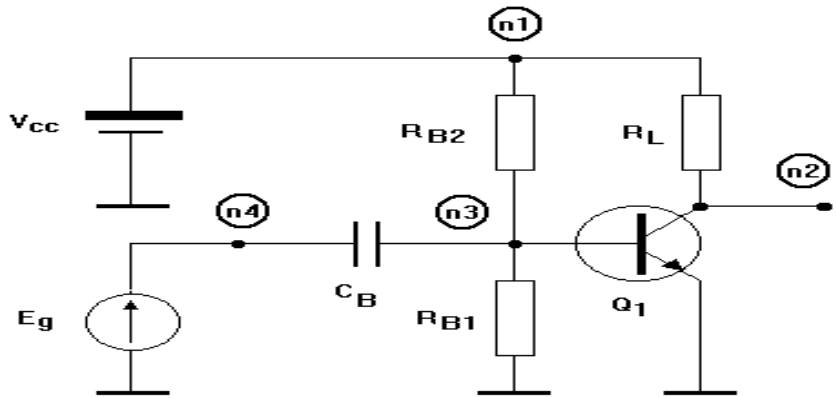
Res RB1 n3 0 21k

Res RB2 n1 n3 170k

Res RL n1 n2 RLoad

Trans Q1 n2 n3 0 MODEL=EMoll

An amplifier circuit



DC analysis results

Operating point

Vce = 5.72

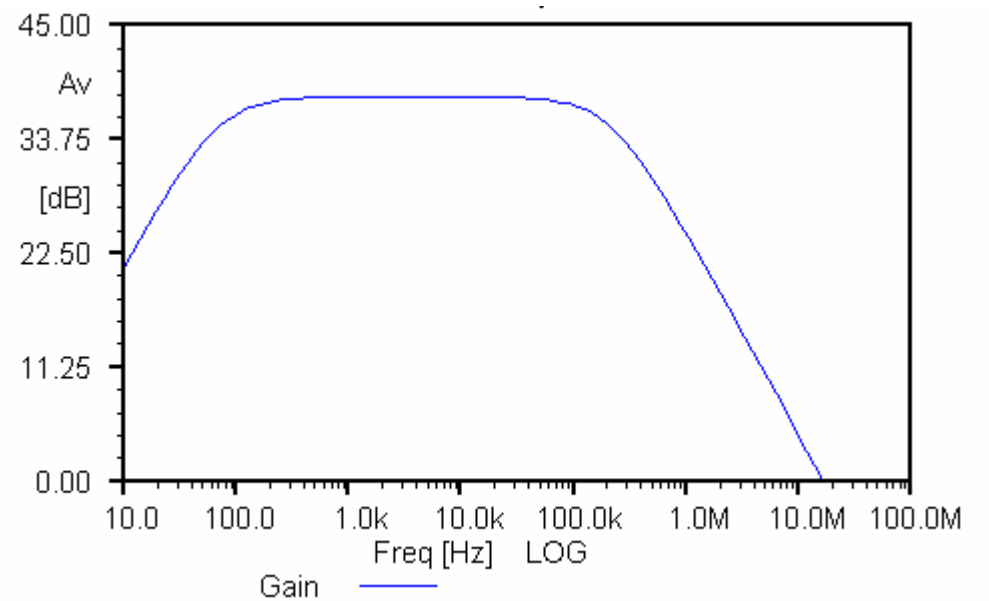
Vbe = 697.65m

Ic = 4.27m

AC Analysis

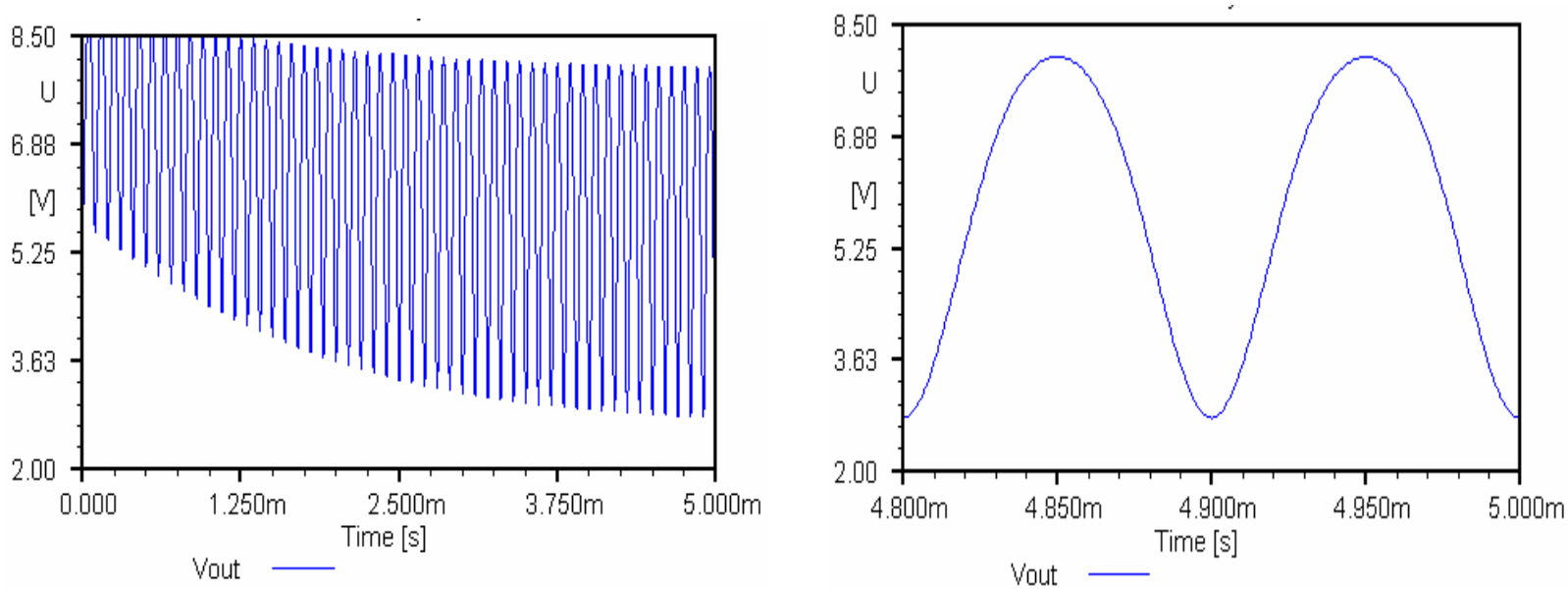
- AC analysis is used mainly in connection with amplifiers and filters, where the frequency response is of interest.
  - The AC simulation is usually based on a sweep over a range of frequencies. The excitation consists of a single frequency at a time and the signal levels are assumed to be so low as not to affect the operating point.

Result of AC analysis



Transient Analysis

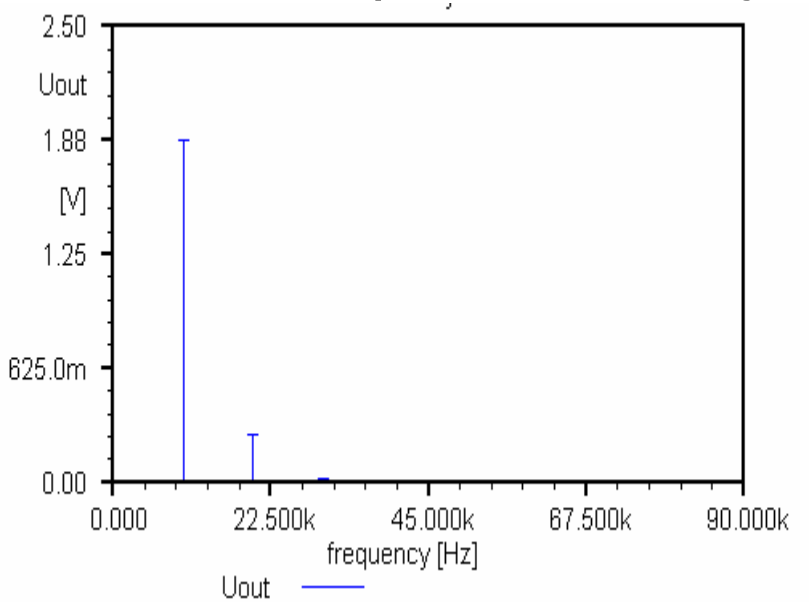
- Transient, or time-domain, analysis most closely simulates the phenomena seen in the real circuit by means of an oscilloscope.
  - A simulation consists usually of a time sweep starting at t=0. When required, a DC analysis precedes the transient analysis and defines the initial conditions for dynamic components unless set by the user.



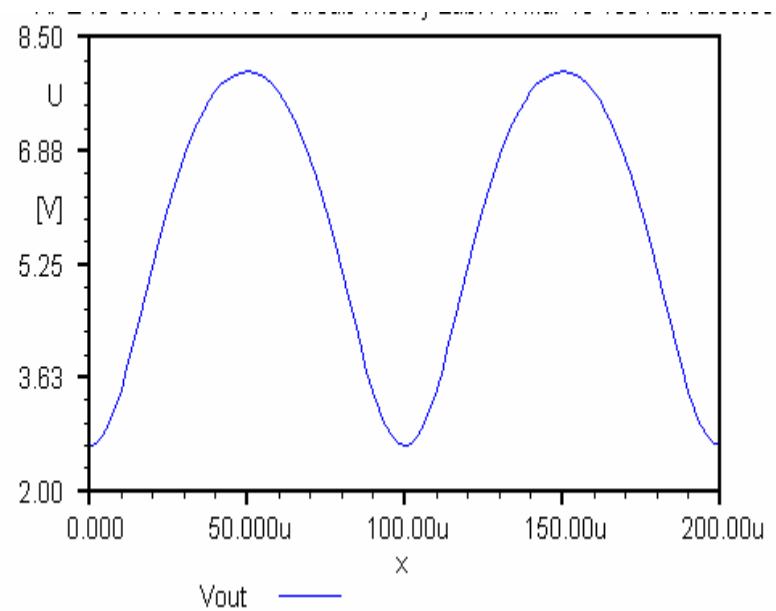
Transient Response at n2

Harmonic Balance Analysis

- When a nonlinear circuit is excited with one or several independent periodic signals, mixing products will be generated.
- Harmonic balance analysis calculates, in the steady state, the spectrum of the signals in the circuit and thus mimics a spectrum analyzer.



Spectrum

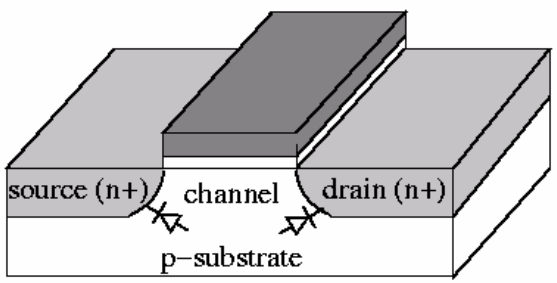


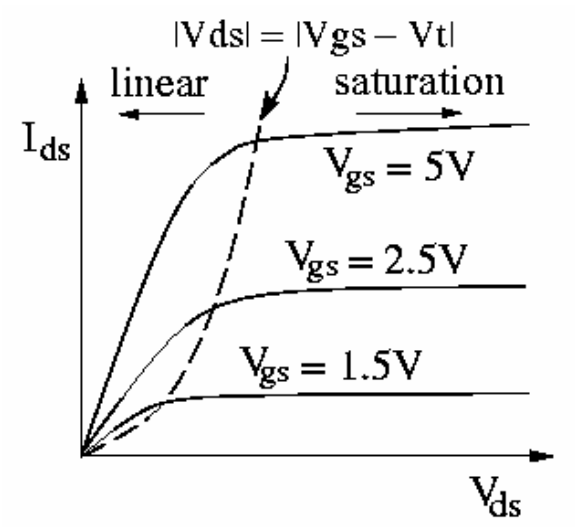
Waveform

Spice Transistor Simulation Models

- MOS transistor models
  - Level 1: basic transistor equations; not very accurate.
  - Level 2: more accurate determination of effective channel length, transition between the linear and saturation regions.
  - Level 3: empirical model
  - Level 4 (BSIM): more efficient empirical model.
  - Level 28 (BSIM2).
  - Level 47 (BSIM3): recent model for deep submicron transistors.
- Some Spice model parameters
  - L, W: transistor length, width (L, W)
  - VT0: zero-bias threshold voltage ($V_{t\,0}$)
  - KP: transconductance (k')
  - GAMMA: body bias factor (γ)
  - TOX: oxide thickness (t_{ox})
  - NSUB: substrate doping (N_a, N_d)
- Commercially available circuit simulation tools: HSPICE, ST-SPICE, etc.

Basic Transistor Equations





- **Cutoff region:** $V_{gs} < V_t$

$$I_{ds} = 0$$

- **Linear region:** $V_{ds} < V_{gs} - V_t$

$$I_{ds} = \beta\left((V_{gs} - V_t)V_{ds} - \frac{1}{2}V_{ds}^2\right)$$

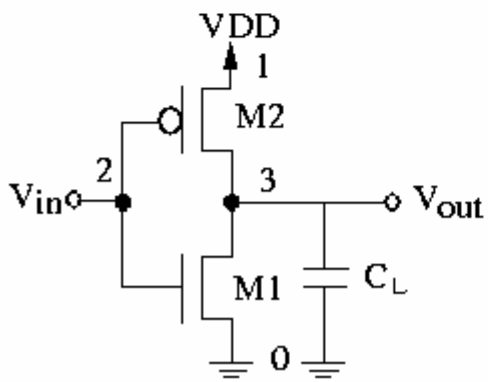
- **Saturated region:** $V_{ds} \geq V_{gs} - V_t$

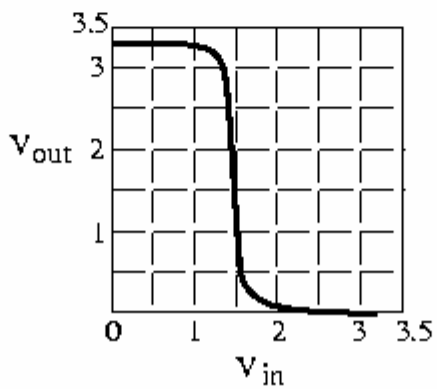
$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_t)^2$$

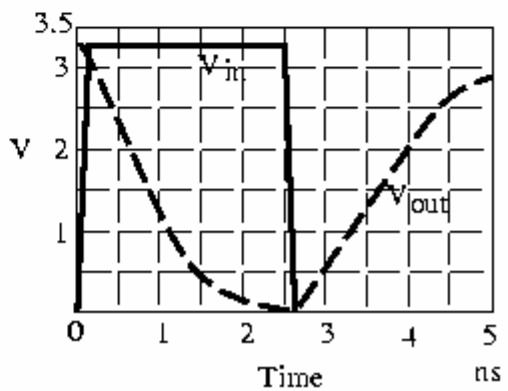
  - V_{gs} (V_{ds}): gate-to-source (drain-to-source) voltage
  - I_{ds}: current between the drain and source
  - V_t: transistor **threshold voltage**
  - β: **Transistor gain factor** $\left(\beta = k'\frac{W}{L}\right)$
  - k': transistor **transconductance** $\left(k' = \frac{\mu\varepsilon}{t_{ox}} = \frac{\partial I_{ds}}{\partial V_{gs}}\Big|_{V_{ds}=constant}\right)$ (μ: carrier mobility; ε: **permittivity** of the gate insulator; transconductance unit: siemens/mho)
  - W/L: width-to-length ratio

Circuit Simulation of a CMOS Inverter (0.6 μm)

```
M1 3 2 0 0 nch W=1.2u L=0.6u AS=2.16p PS=4.8u AD=2.16p PD=4.8u
M2 3 2 1 1 pch W=1.8u L=0.6u AS=3.24p PS=5.4u AD=3.24p PD=5.4u
CL 3 0 0.2pF

VDD 1 0 3.3
VIN 2 0 DC 0 PULSE (0 3.3 0ns 100ps 100ps 2.4ns 5ns)

.LIB '../mod_06' typical

.OPTION NOMOD POST INGOLD=2 NUMDGT=6 BRIEF
.DC VIN 0V 3.3V 0.001V
.PRINT DC V(3)
.TRAN 0.001N 5N
.PRINT TRAN V(2) V(3)
.END
```



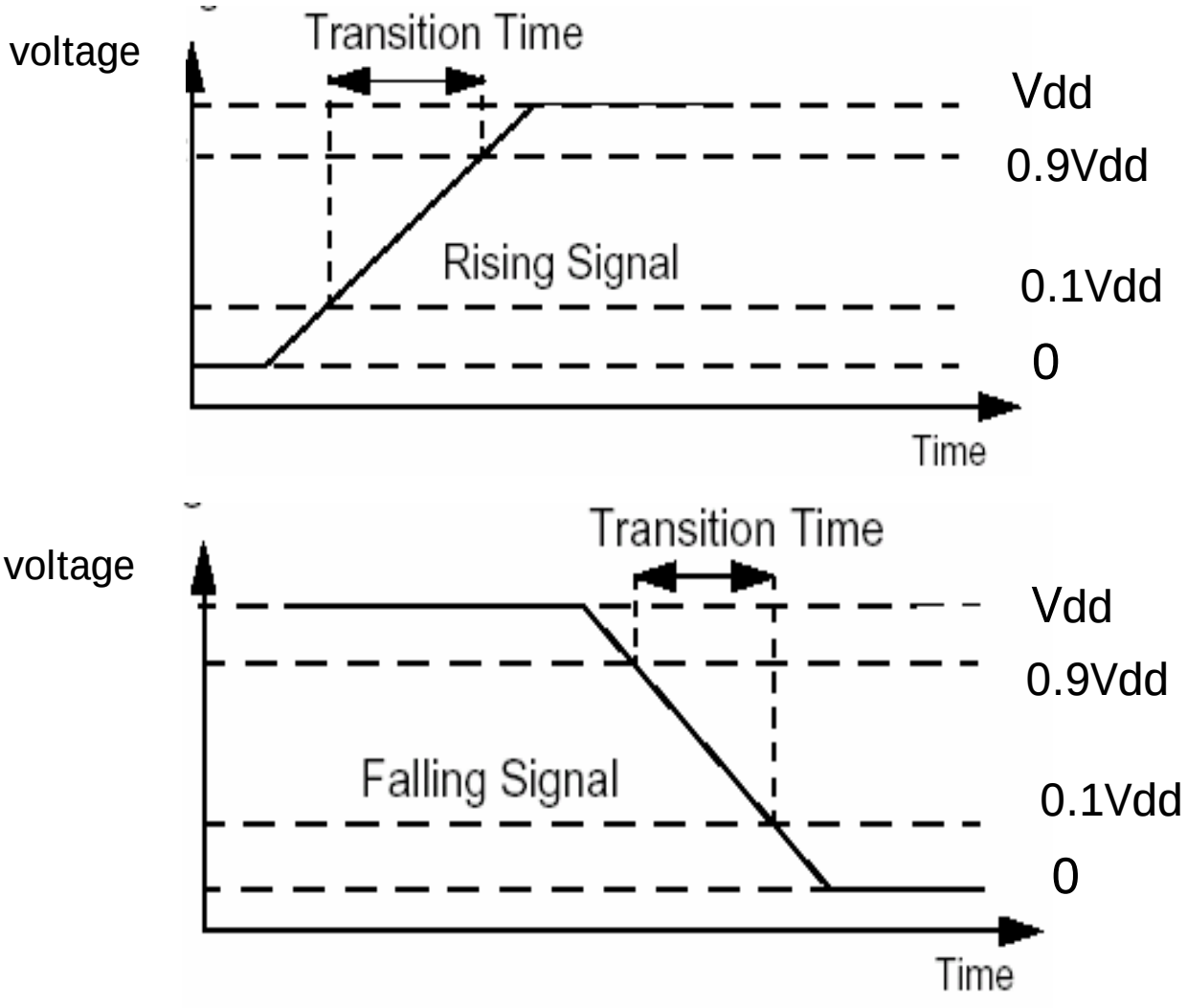




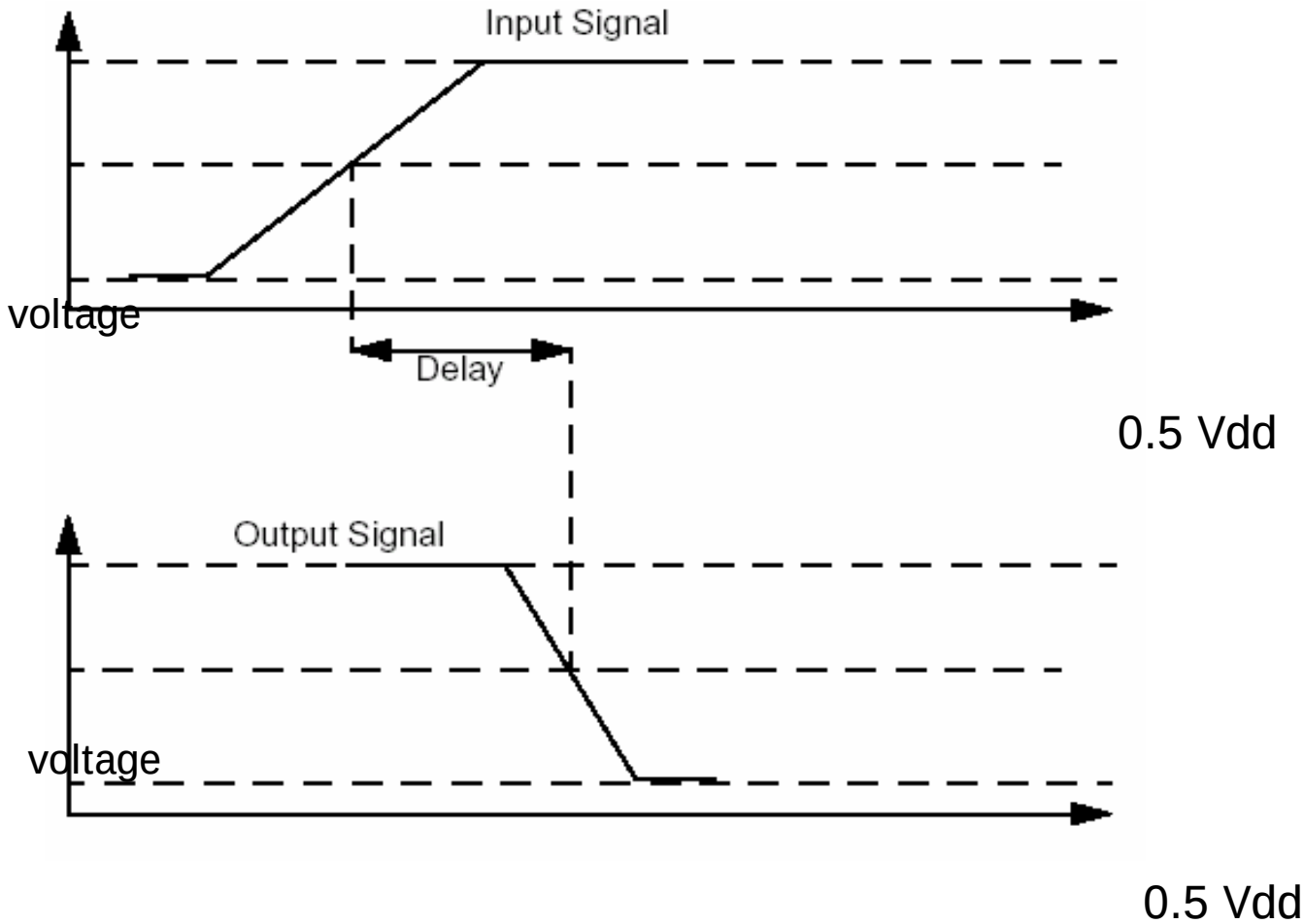
Circuit Simulation for Delay Calculation

- Used only for small circuits or a small portion of a circuit which is timing critical.
- Definition of delays
  - rise (fall) time
    - As a convention, it is the time period for a signal to rise (fall) from 10% (90%) Vdd to 90% (10%) Vdd.
  - Rise (fall) delay
    - As a convention, rise delay is measured from the 50% Vdd of the input transition to the 50% Vdd of output rise (fall).
  - Note
    - Some uses 20% and 80% thresholds for calculating delays.
- Propagation delay
  - The time period between the moment when an input signal transition reaches 50%Vdd and the moment when the output signal transition reaches 50%Vdd.

Rise Time and Fall Time

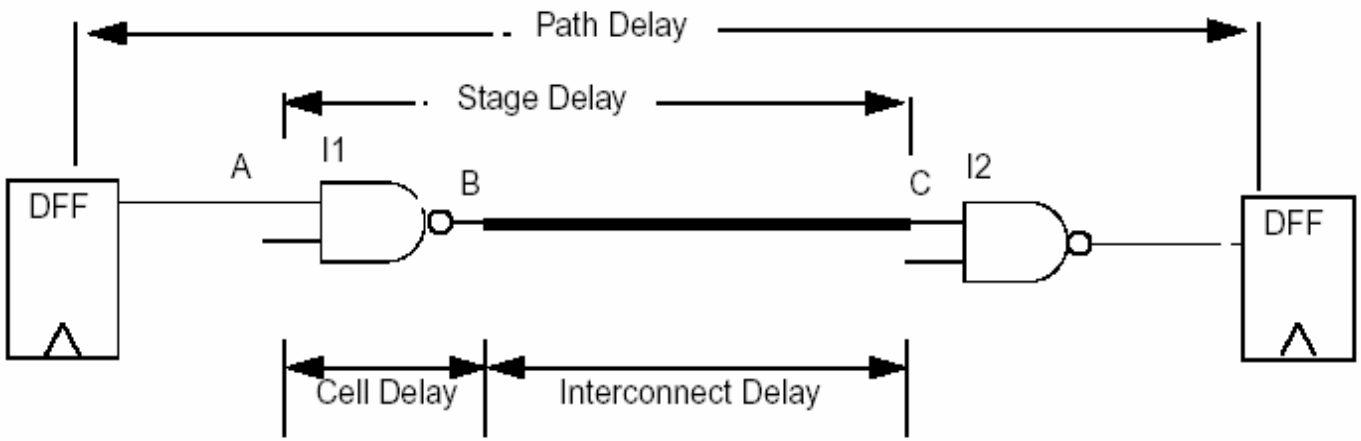


Propagation Delay



Path Definition and Delay

- Path definition
  - A path is defined as from a source to a sink.
    - A source can be the output of a flip-flop (latch) or a primary input (i.e., the input pin of an IC).
    - A sink can be the input of a flip-flip (latch) or a primary output (i.e., the output pin of an IC).
- Path delay comprises
  - The delay of a flip-flop (latch) which serves as a source.
  - The delay of any device along the underlying path.
  - The delay of any interconnect along the underlying path.



Set-up and Hold Time Requirement

- Set-up time and hold time of a certain memory element forms a time window around the clock arrival edge.
- Within the time window, signal should not change its logic value, otherwise the memory element may goes into meta-stable state or be loaded with a wrong logic value.
- That is to say, a signal being propagated from the source of a path can be safely loaded into a memory element if the signal can be stable before the set-up time and remain stable during the time window.

Purposes of Path Delay Calculation

- Check whether a signal change traversing through a path violates the set-up time and hold-time constraints at a certain memory element (flip-flop or latch).
- To achieve this goal, we should be able to calculate the path delays.
- Delay calculation is usually done on per path basis for checking
  - Whether any long path in the circuit would violate setup time constraint of flip-flops or latches.
  - Whether any short path would violate hold time constraint of flip-flops or latches.

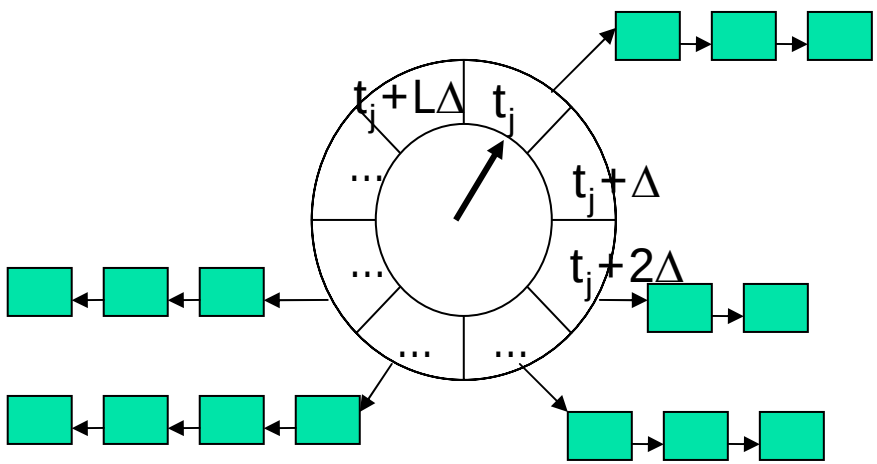
Event Driven Simulation

Event-Driven Simulation

- **Event-driven simulation** is a widely-used mechanism in gate- and switch-level simulators.
- An **event** is a change of a signal value that may trigger new changes.
- There is a queue of events ordered by the time when the event is going to happen.
- Basic steps:
  - The output of a gate G changes at time t_i.
  - The fanout of the gate is inspected; it consists of the inputs of the gates G_k that are connected to the output of gate G.
  - If the outputs of the gates G_k change, they are scheduled to change at time $t_i + \Delta_k$, where Δ_k is the delay associated with the transition.

Timing Wheel

- Instead of using an event queue, a timing wheel is most often used to manage events.
- It is a circularly linked list which contains the scheduled events based on their temporal information about when the events ought to occur.



t_j is current time

Δ is the minimum time resolution used in the simulation

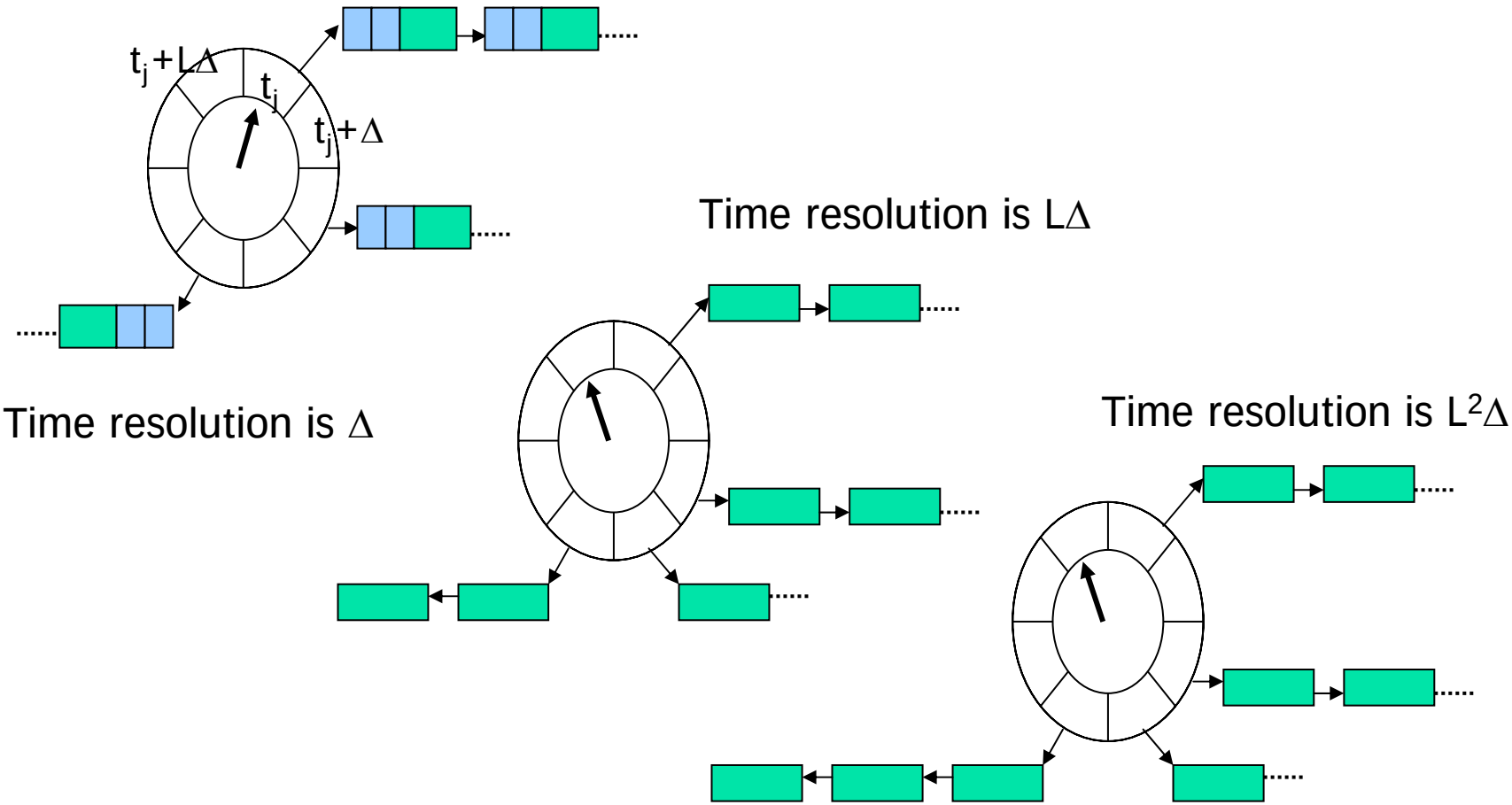
L+1 is the number of entries in the timing wheel

Operation of a Timing Wheel

- Remove an event from the linked list pointed by the current time.
- Evaluate the event.
- Schedule all events triggered by the evaluation of the above event to the linked-list based on their occurring times.
- When the linked list pointed by the current time is empty, advance the current time by Δ.
  - More directly, the time is advanced to the time slot pointing to a non-empty list. Then, the simulation time step is simply the time period between two non-empty time slots.
- This process is repeated until the wheel is empty.

Hierarchical Timing Wheel

- If the event occurring time exceeds the maximum allowable time permitted by a timing wheel, a hierarchy of timing wheel can be used.



Switch-Level Simulation

Basics of Switch-Level Simulation

- Treating transistor as bidirectional switches.
- Signals are with discrete values.
- Transistor and interconnect parasitic resistance and capacitance can be included in the circuit network to have a better approximation of real circuit behavior.
- Purposes of switch-level simulation
  - Validate the function of a circuit
  - Simulate timing behavior of a circuit
  - Estimate power consumption of a circuit

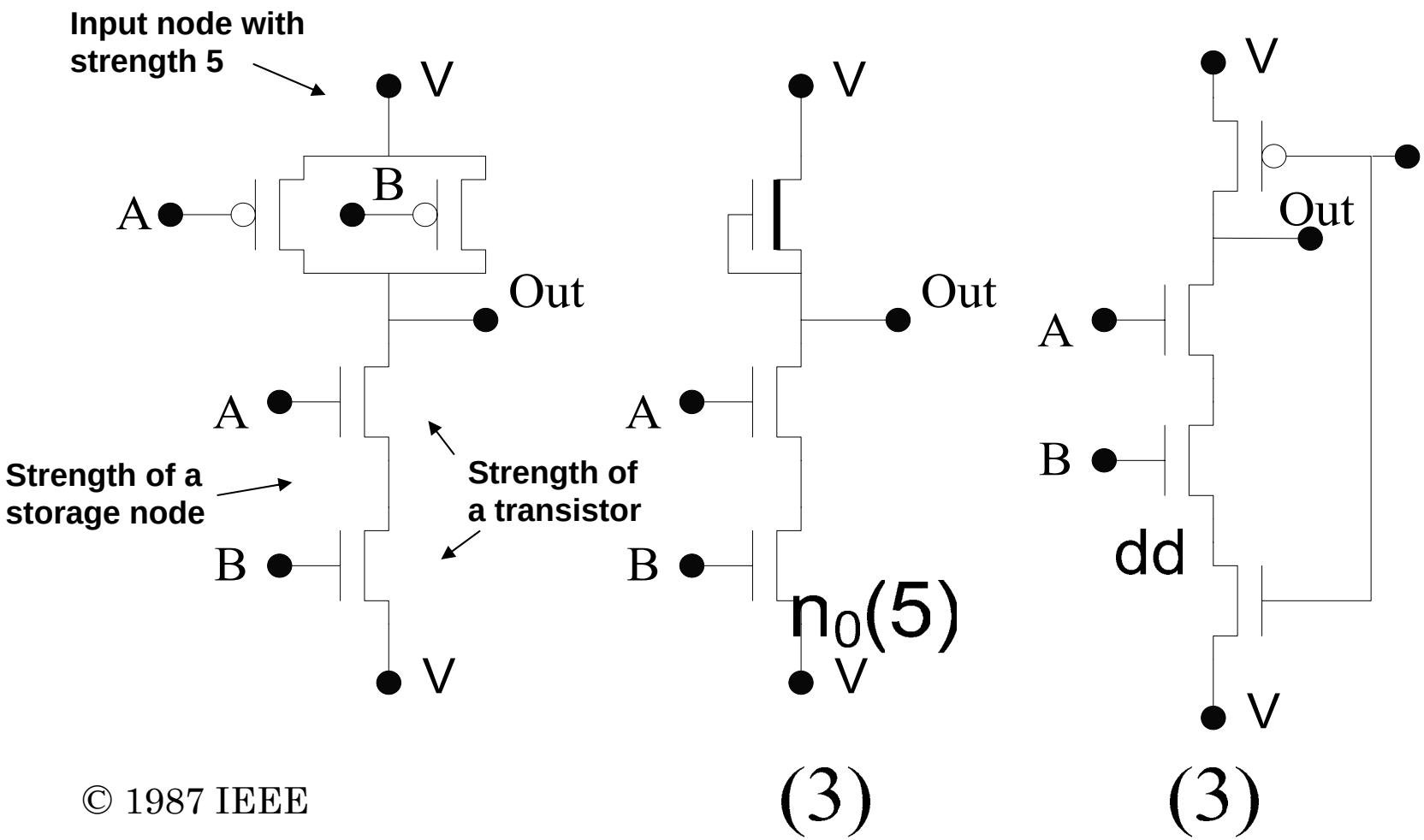
Validation of Functional Behavior

- Transistors are modeled as bidirectional switches.
- Transistor on-resistance (related to transistor size) is modeled as a strength (from a set of discrete values) driving the load.
- Transistor network is modeled as a graph model where
  - Transistor is modeled as an edge between two circuit nodes.
  - Circuit node is modeled as either the input node or storage node.
    - Input node can be Vdd, Gnd or a strong logic 1 or 0.
    - Storage (charged) node is a node with a capacitance.
    - A node is also associated with a strength.
    - An input node has the largest strength.
    - A storage node has a strength proportional to its capacitance.

Strength Value and Signal Representation

- Strength
  - k < s <w: s is the strength of a transistor;
  - 1 ≤ s ≤ k: s is the strength of a storage node;
  - w: is the strength of an input node;
- Signal Representation
  - Represented by a pair of (s, v), where
    - **s** is a strength
    - **v** is a level, a voltage denoted with discrete values at least including 1, 0, and X, where X represents the unknown signal value.

Strength Model Example



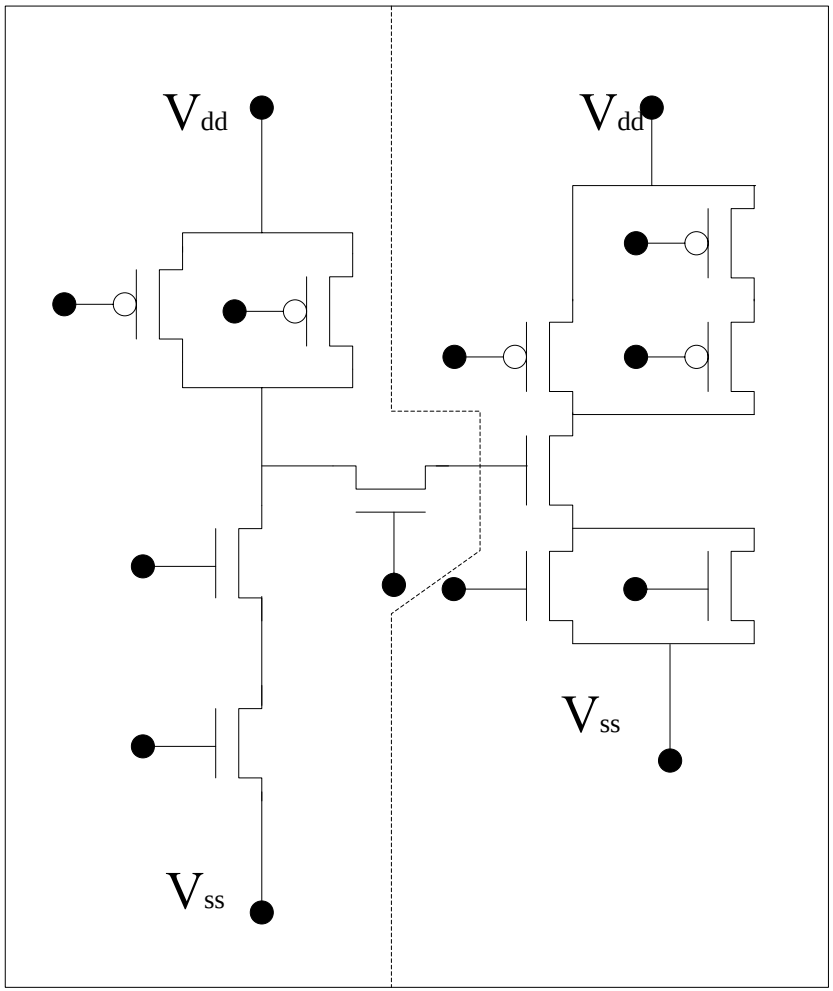
Switch-Level Simulation Techniques

- Partitioning the circuit into subcircuits that can be treated as unidirectional components.
  - Static partitioning: Connections to the gate of a transistor determine subcircuit boundaries irrespective of the signals carried by the nets.
  - Dynamic partitioning: Known signal values in the network are taken into account such that further partitioning of subcircuits is possible.
- Each subcircuit is then modeled as a channel-connected component or a switch graph (multigraph) G=(V, E), where
  - V is a set of vertices representing input or storage nodes labeled with node (net) names and strengths.
  - E is a set of edges representing transistors labeled with a transistor name and strength.

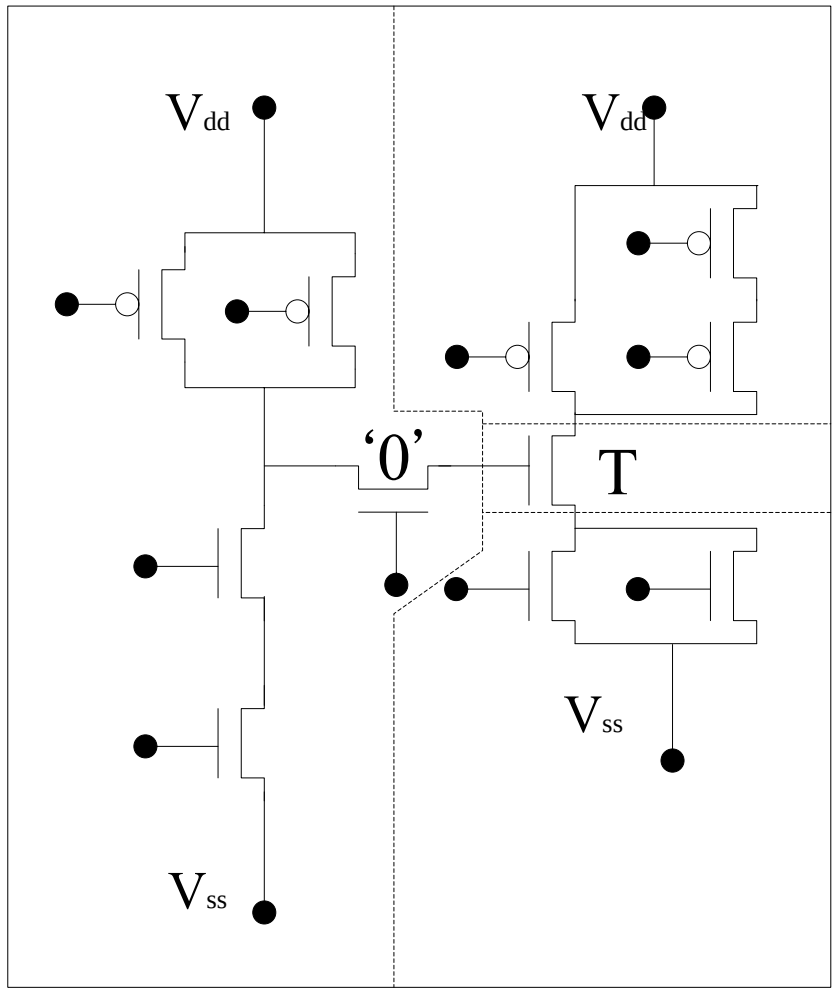
Switch-Level Simulation Techniques (cont'd)

- Apply special methods to compute the steady state of a subcircuit whose inputs are given with logic values.
- The steady state output value of a subcircuit (called B) is then served as the input value to the other subcircuits that have a connection to the output of B.
- Event-driven simulation is normally used to propagate the signal changes and find the steady state.
- The simulation time step is decided by the two most adjacent events. It is normally not a fixed time value, but the simulator can predict the next simulation time point.
- This process is repeated until the steady values of all subcircuits are computed.

Static Versus Dynamic Partitioning



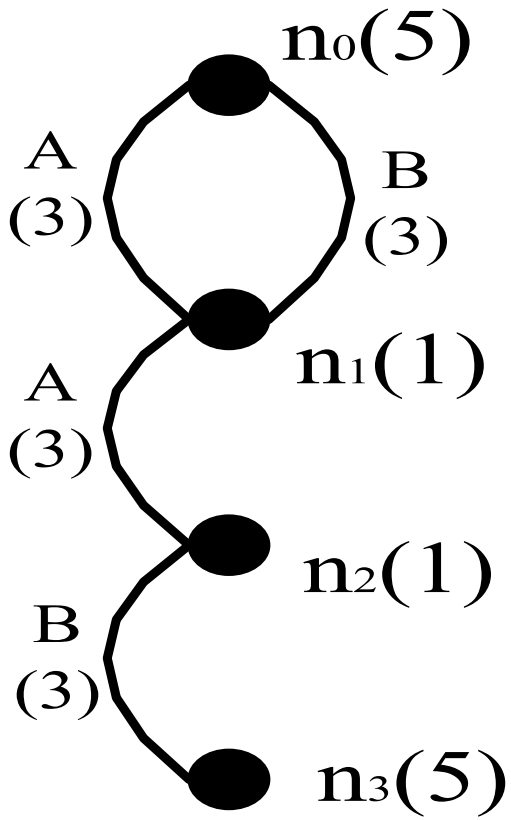
Static partitioning



Dynamic partitioning

Example of Switch Graph

- A convenient representation for switch-level circuits is a multigraph rather than the more general cell-port net model.
- Vertices represent nets and are labeled with the net name and strength.
- Edges represent transistors and are labeled with a transistor ID and strength.



Signals and Signal Propagation

- A signal on a vertex $u \in V$ is denoted by $\langle \sigma_u , \lambda_u \rangle$.
- The strength of a transistor $(u,v) \in E$ is given by $\varepsilon_{u,v}$. $\varepsilon_{u,v} = 0$ when the transistor is off.
- $\sigma_{u \to v}$ denotes the strength of the signal flowing from $u \in V$ to $v \in V$ $\qquad \sigma_{u \to v} = \min\left(\sigma_u, \varepsilon_{u,v}\right)$
- The level of the signal flowing remains λ_u.
- There are two types of nets:
  - *driven* nets: nets having a conducting path to an input net
  - *charged* nets: nets electrically isolated from input nets

Signals and Signal Propagation (cont'd)

- Suppose that a driven net $v \in V$ has edges (u_1, v), …, $(u_m, v) \in V$, then $$\sigma_v = \max_{1 \le i \le m} \sigma_{u_i \to v}$$
- For a charged net, the net's own signal should be taken into account, $$\sigma_v = \max(\sigma_v, \max_{1 \le i \le m} \sigma_{u_i \to v})$$
- When combining signals from different directions, the level of the new signal equals the level of the strongest signals. In case of multiple signals with equal strength and different levels, the new level becomes 'X'.

Simulation Algorithm Principles

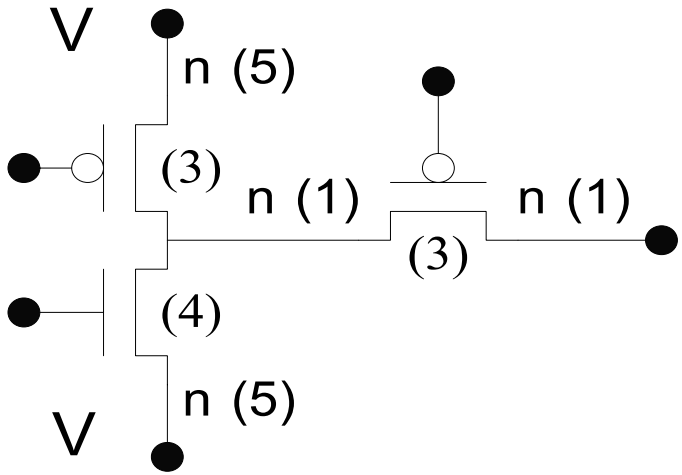
- The algorithm is based on a repeated application of:

$$\sigma_v = \max(\sigma_v,\ \sigma_{u \to v})$$

- This should be done carefully
  - propagate the strongest signals first, i.e., the algorithm must wait until all the signal activity subsides at a node before the signal value is propagated forward.
- Implement with an array of queues, one array position for each strength value.

Example

- The table below shows how input signal changes are propagated to the output.



Propagate from → to	State of n_2	State of n_3
"Initial state"	(1, X)	(1, X)
$n_o \rightarrow n_2$	(3, 1)	(1, X)
$n_1 \rightarrow n_2$	(4, 0)	(1, X)
$n_2 \rightarrow n_3$	(4, 0)	(3, 0)

Strength Logic value

Discussion

- The above algorithm operates in linear time with respect to the number of nets and transistors.
- The algorithm is static, i.e.,
  - Changes to input signals require repeating the complete propagation.
  - The algorithm does not propagate any type of delays related to the physical implementation

Switch-Level Timing Simulation

- Simulate the timing behavior of a circuit to find out timing problem.
- Need delay models to account for
  - Transistor on-resistance and capacitance
  - Interconnect resistance and capacitance
- Delay Models
  - Lumped RC model (overestimating delay)
  - Lumped RC model + input slope (slew rate)
  - Distributed RC model + input slope
- Need to find input vectors that activate the longest paths and shortest paths. This is no guarantee of finding such vectors.

Model Library for Switch-Level Simulation

- Just like the SPICE, we need a model for each type of transistors to perform switch-level simulation
  - For functional verification, a transistor is modeled as a bidirectional switch with various strengths.
  - For timing simulation, the strength of a transistor is replaced with an on-resistance of the transistor.
- Switch-level transistor model is relatively simpler than the models of components for other types of simulators
  - What we need is the on-resistance of a transistor for a given process technology under certain operating condition.
  - The computation of on-resistance can be implemented as a part of a simulator using the information provided by a process technology file.

Switch-Level Timing Verification

- Need delay models for transistors and interconnects.
- Input vector independent (it actually uses all the input vector combinations at the same time).
- Propagate all the inputs with rise and fall transitions to find out the worst-case delay path.
- The algorithms to propagate the rise and fall transitions from sources to sinks are similar to those used for simulation
  - A circuit is first decomposed into stages.
  - The output change of a stage would turn on or turn off some transistors in other stages and as such the changes are propagated from sources to sinks.
  - On the propagating the change from stages to another stages, transistor and interconnect delay models are used compute the path delay.

Gate-Level (Logic) Simulation

Signal Modeling for Gate-Level Simulation

- Signal values are discrete.
- The minimum set consists of ‘0’, ‘1’ and ‘X’.
  - ‘X’ means ‘‘unknown’’.
- Many models use more signal values.
  - IEEE std_logic data type with 9 values: mixture of level and strength.
    - ‘U’ (uninitialized)
    - ‘X’ (forcing unknown)
    - ‘0’ (forcing 0); ‘1’ (forcing 1)
    - ‘Z’ (high impedance)
    - ‘W’(weak unknown)
    - ‘L’ (weak 0), ‘H’, (weak 1)
    - ‘–’ (don’t care).

Gate Modeling

- Gate models should deal with multiple-valued logic.
- Gate behavior can be represented by truth tables or compiled code.

3-valued NAND truth table

in_1	in_2	out
'0'	'0'	'1'
'0'	'1'	'1'
'0'	'X'	'1'
'1'	'0'	'1'
'1'	'1'	'0'
'1'	'X'	'X'
'X'	'0'	'1'
'X'	'1'	'X'
'X'	'X'	'X'

Delay Models for Gate-Level Simulation

- **Inertial delay**
  - a change to an input signal has to last at least a certain time before it can trigger any reaction.
- **Propagation (or transport) delay**
  - some time passes between the start of a signal change at the gate input and the start of a signal change at its output.
- **Rise/fall delay(time)**
  - due to capacitances that have to be charged or discharged, there is a time difference between the moment when an output starts to change and the moment when the output has reached its final value.

More Accurate Gate Delay Model

- Timing (delay) model for each gate in the library should contain
  - rise and fall delays as a function of gate size, load capacitance (as well as resistance if wire is long), and Input slope
  - propagation delay as a function of gate size, load capacitance, and input slope.
- Typically, three kinds of delay are of interest
  - Worst case delay
    - Using T(emperature) = 125° C, supply voltage= 90% Vdd, and worst case SPICE model for delay characterization.
  - Best case delay
    - Using T = 0° C, supply voltage= 110% Vdd, and best case SPICE model for delay characterization.
  - Typical case delay
    - Using T = 27° C, supply voltage= 100% Vdd, and typical case SPICE model for delay characterization.

Timing Library For Logic Simulation

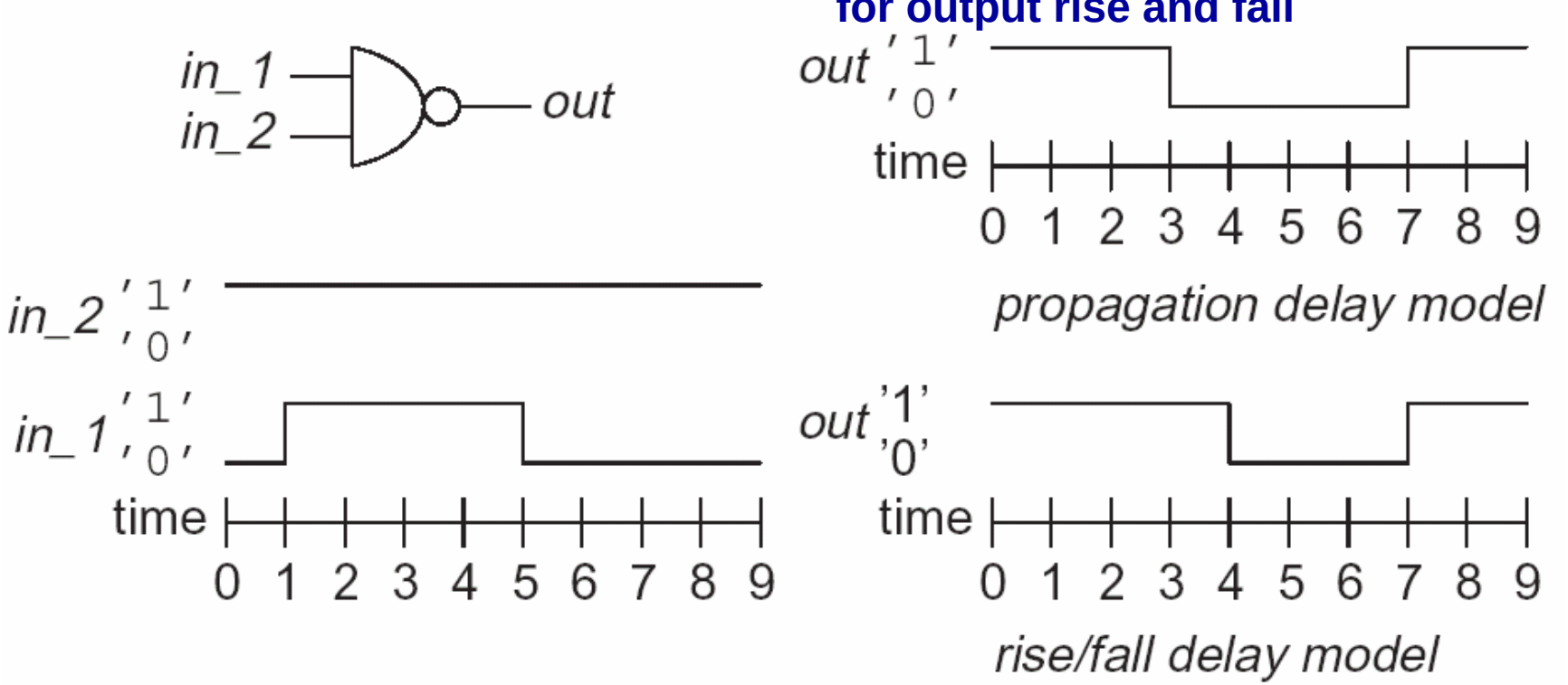
- Usually organized as a table when given
  - a cell, its input slew rate, and its output loading, looking up the table will return to you the output transition time (rise or fall time) and the propagation delay.

Delay table for some cell in Cadence TLF format

```
Model(delayTemplateModel
     (Spline
      (Input_Slew_Axis 0.050 0.200 1.000 4.000 20.000)
      (Load_Axis 0.0446 0.892 3.568 14.275)
        data()
     )
)
Cell(...
     Model(ioDelayRiseModel  delayTemplateModel
     (Spline
        data((0.7210 0.8471 1.2849 3.05673)
             (0.8119 0.9380 1.3758 3.1475)
             (0.9975 1.1236 1.5612 3.3322)
             (1.4293 1.5552 1.9922 3.7609)
             (3.3955 3.5204 3.9542 5.7101))
...
)
```

Delay Model Example

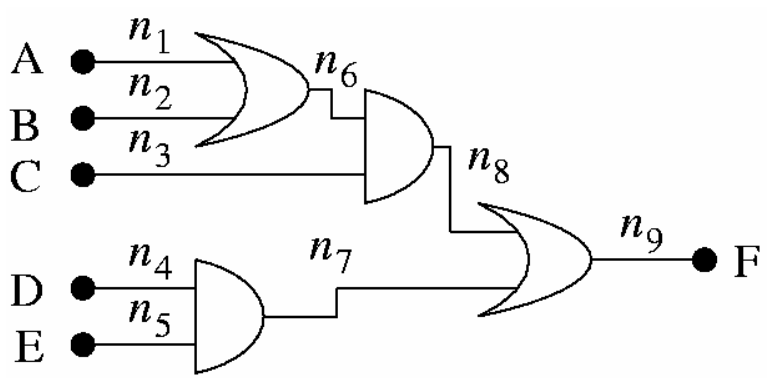
With a single delay value 3 units for output rise and fall



With a fall delay value 3 units and rise delay value 2 units

Compiler-Driven Simulation

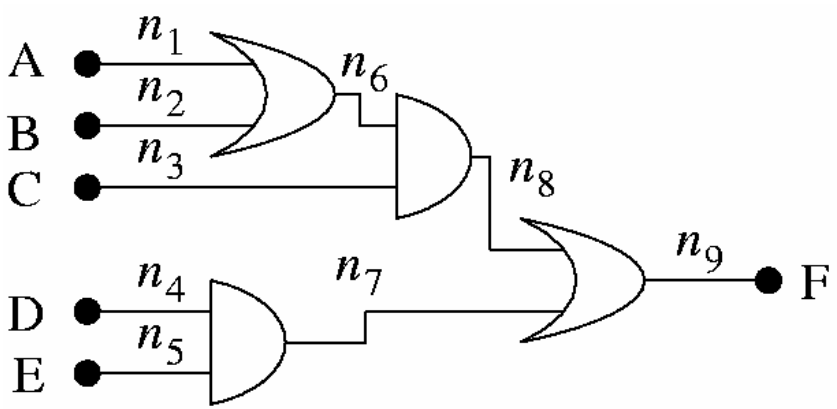
- **Compiler-driven simulation** is often used in gate-level simulators.
- Based on making an executable-code model of a circuit.
- Efficient simulation mechanism (few machine instructions per gate).
- Applicable to few delay models in synchronous circuits (e.g. zero-delay model).

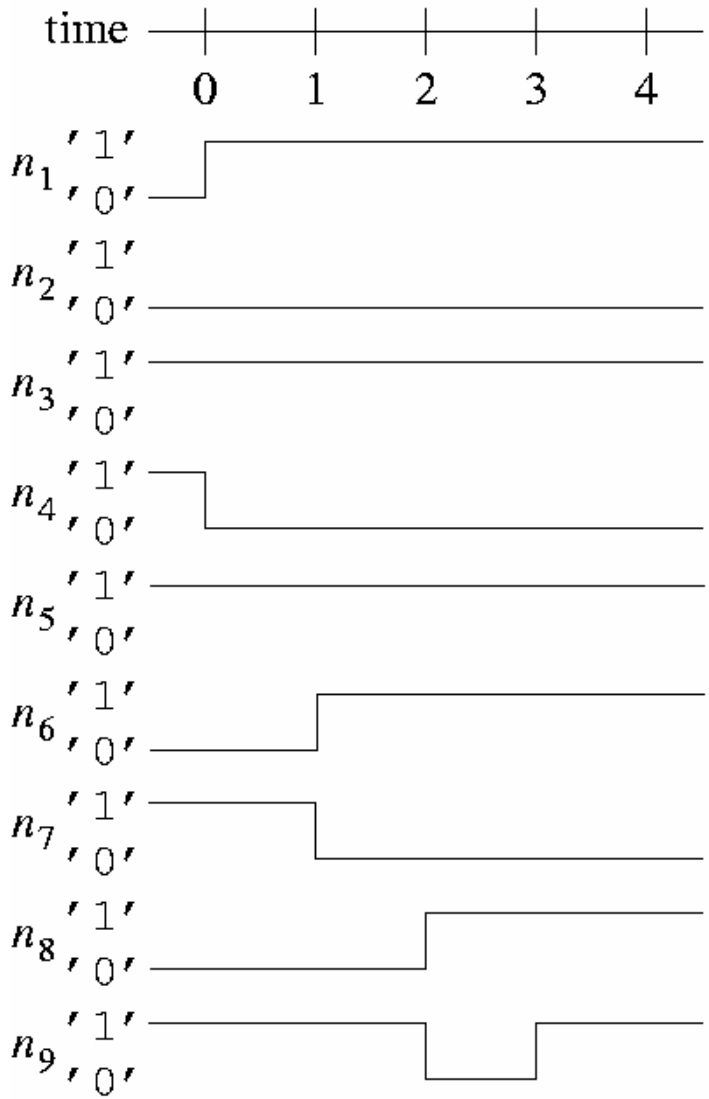


$n_1 \leftarrow$ A;
$n_2 \leftarrow$ B;
$n_3 \leftarrow$ C;
$n_4 \leftarrow$ D;
$n_5 \leftarrow$ E;
$n_6 \leftarrow$ OR(n_1, n_2);
$n_7 \leftarrow$ AND(n_4, n_5);
$n_8 \leftarrow$ AND(n_6, n_3);
$n_9 \leftarrow$ OR(n_7, n_8);
F $\leftarrow n_9$;

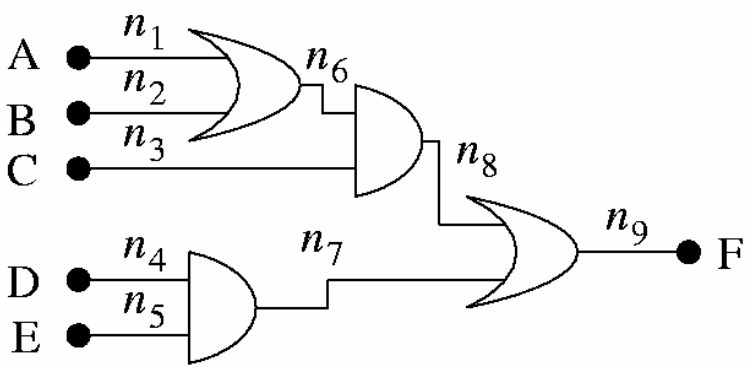
Unit-Delay Simulation

- Assumes that all gate delays equal 1.
- Provides some information about signal evolution in time, especially to detect *glitches*.





Compiled Code for Unit-Delay Simulation




```
for (t ← t_start; t ≤ t_end; t ← t + 1) {
    new[1] ← A;
    new[2] ← B;
    new[3] ← C;
    new[4] ← D;
    new[5] ← E;
    new[6] ← OR(old[1], old[2]);
    new[7] ← AND(old[4], old[5]);
    new[8] ← AND(old[6], old[3]);
    new[9] ← OR(old[7], old[8]);
    F ← new[9];
    old ← new;
}
```

Event-Driven Logic Simulation (EDLS)

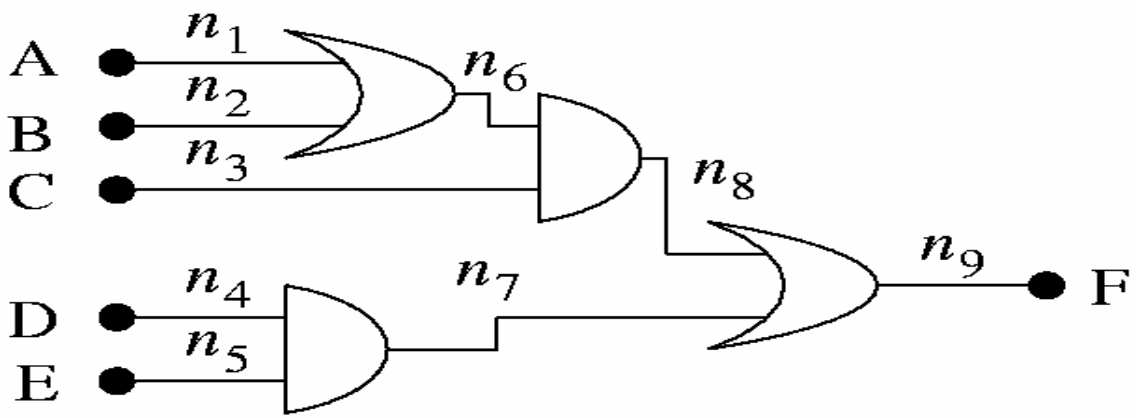
- Normally use a timing wheel to keep track of event occurrences.
- The occurrence of an event is a change of the logic value on a signal. It is possible that there are multiple events occurring for the same signal due to the different arrival times of triggering events.
- More accurate delay model can be used
  - to find out hazards or glitches
  - to perform more accurate path delay calculation
  - to perform more accurate estimation of node switching activity for power computation
- It is slower than compiled-code simulation.
- Its simulation time step is decided by the two most adjacent events. It is not a fixed time value, but the simulator knows which event will be processed next and thus know the next time point.

Algorithm for EDLS

```
Event_Driven_Simulation()
{
    struct event_queue *Q; // Used as a timing wheel
    Q← new_queue();  // Create timing wheel
    insert_stimuli(); // Put the events caused by the given stimuli
    initialize_network(); // Set all network nodes representing  memory
                          // elements to 'U' and all other nodes to 'X'
    for(t=t_start; t<t_end){
        current_event← first_event(Q, t);
        while (current_event != NULL){
            process_current_event(current_event);
            add_new_events(current_event, Q);
            current_event← first_event(Q, t);
        }
    Advance_current_time(t);
    }
}
```

This is only an outline of a simple timing wheel algorithm. For more complex timing wheel implementation, one has to add codes to manage the timing wheel.

Example (1/10)



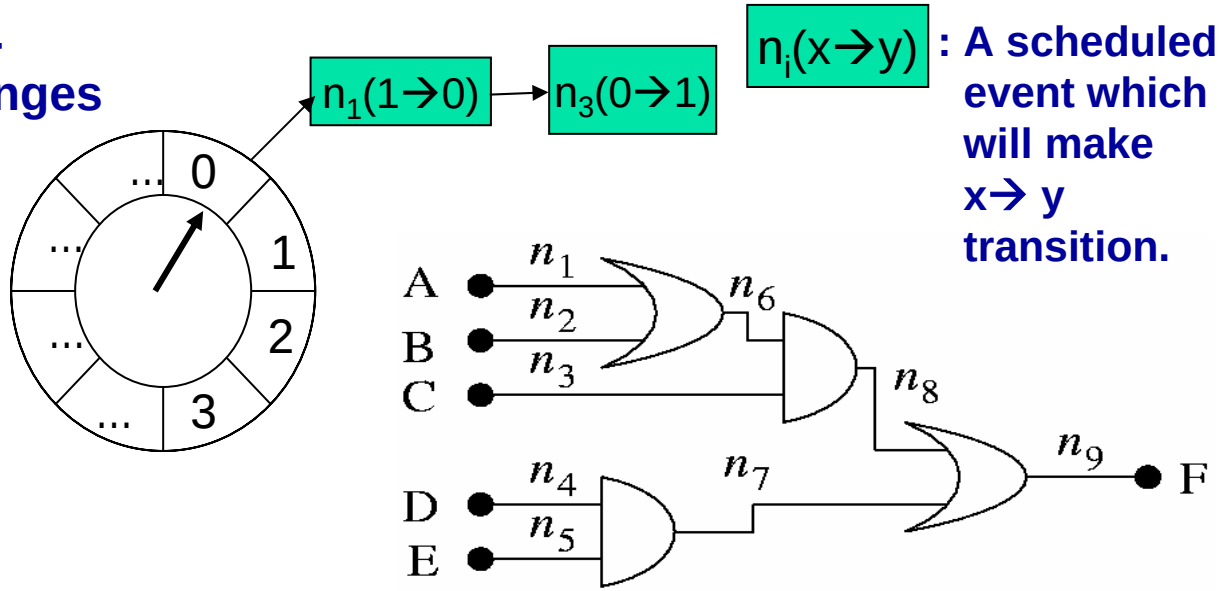
- Suppose the two-input ORs have a propagation delay of 2 ns and two input AND gates have a propagation delay of 3 ns. Suppose the time resolution for simulation is 1 ns (i.e., Δ = 1ns).
- Suppose at time zero, the five inputs go through the following logic value changes:
  - A: 1→ 0, B:0→0, C: 0→1, D: 0→0, E: 0→0
- Perform logic simulation using a timing-wheel with 8 slots (one slot = 1ns).

Example (2/10)

- **Set up the timing-wheel for the changes of input A and C.**

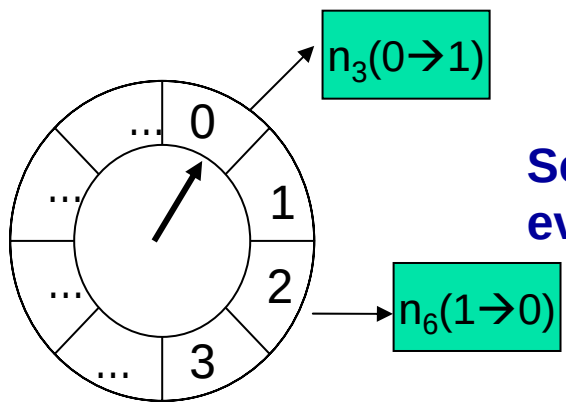
Current time t=0

$n_1=1$, $n_2=0$, $n_3=0$, $n_4=0$, $n_5=0$, $n_6=1$, $n_7=0$, $n_8=0$, $n_9=0$



Process event $n_1(1\rightarrow0)$ at t= 0

$n_1=1\rightarrow0$, $n_2=0$, $n_3=0$, $n_4=0$, $n_5=0$, $n_6=1$, $n_7=0$, $n_8=0$, $n_9=0$

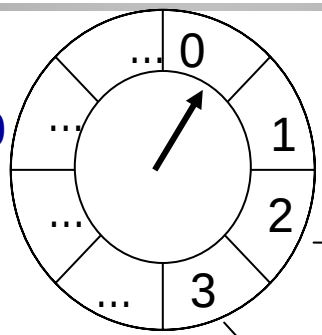


Schedule the triggered event $n_6(1\rightarrow0)$

Example (3/10)

Process event n_3(0→1) at t=0

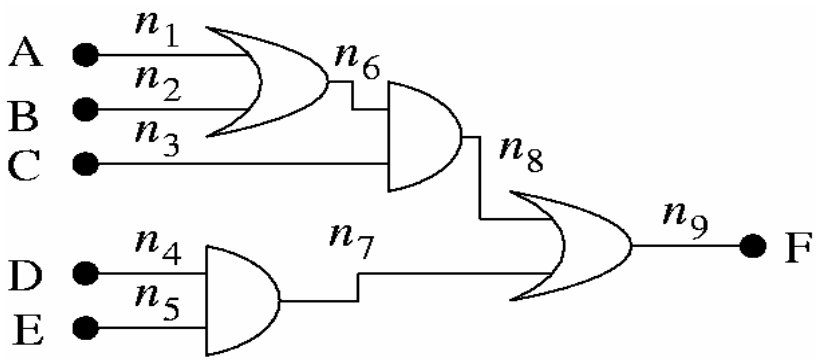
n_1=0, n_2=0, n_3=0→1, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0



n_6(1→0)

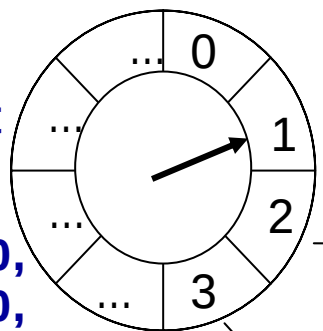
n_8(0→1)

Schedule the triggered event n_8 (0→1)



Advance current time by one resolution unit to t = 1

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0



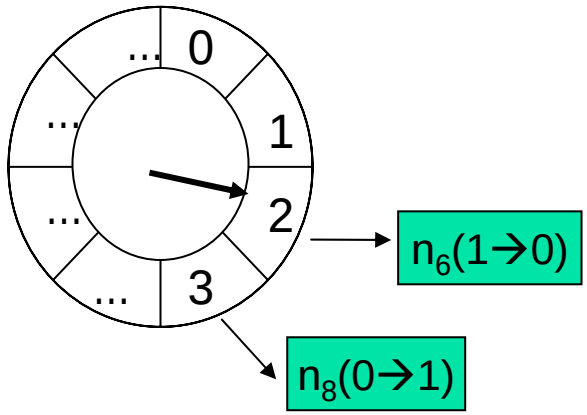
n_6(1→0)

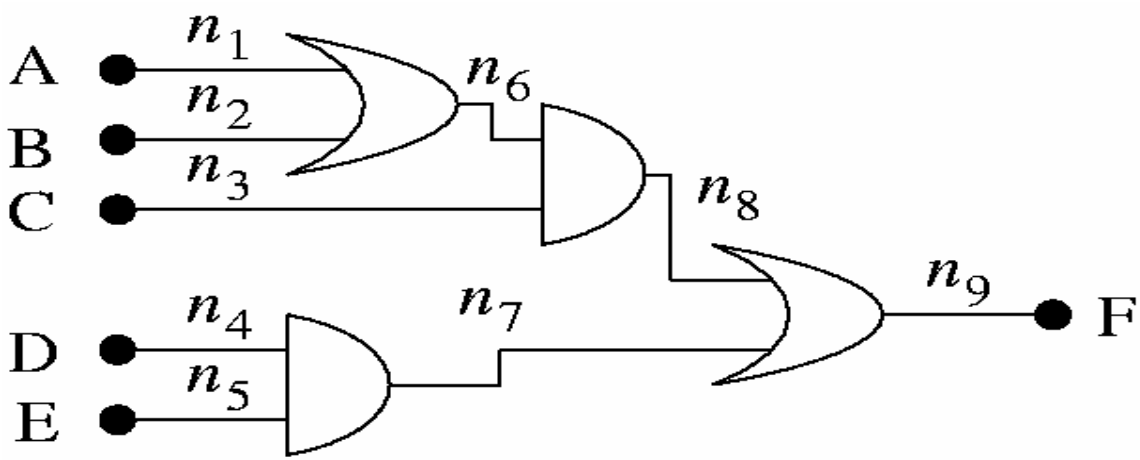
n_8(0→1)

Example (4/10)

Advance current time by one resolution unit to t=2

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0

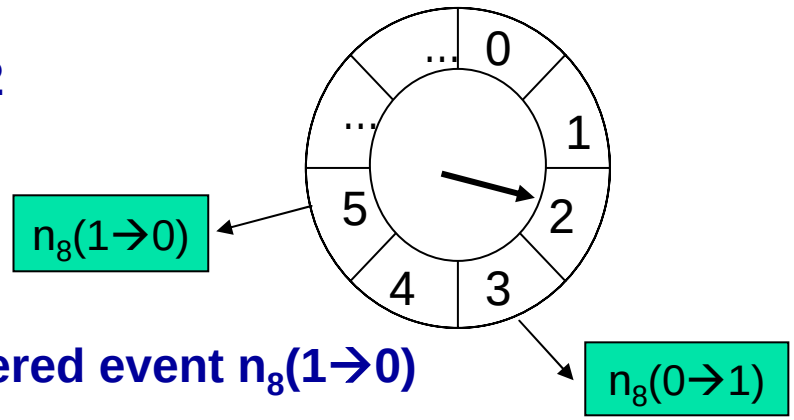




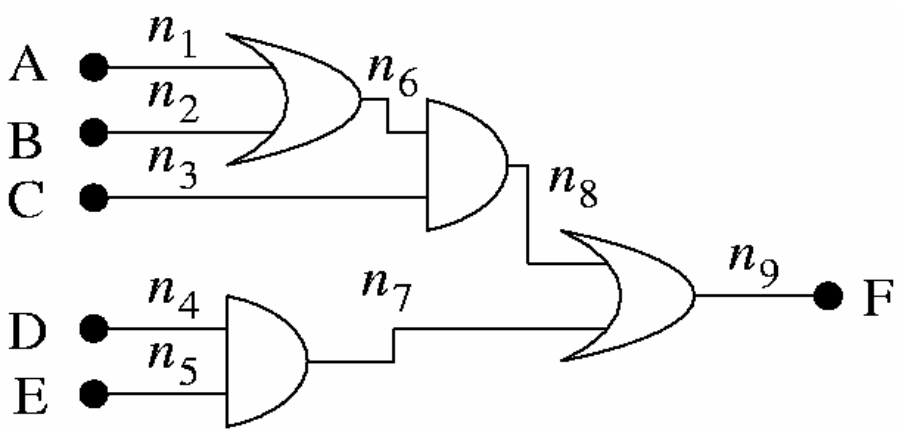
Example (5/10)

Process event n_6(1→0) at t=2

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=1→0, n_7=0, n_8=0, n_9=0



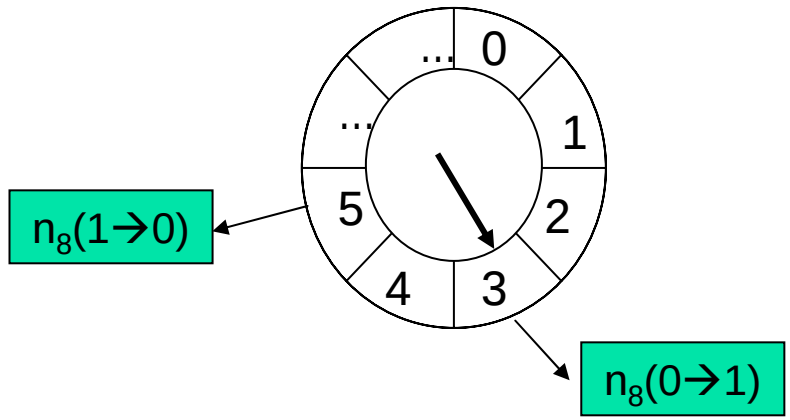
Schedule triggered event n_8(1→0)

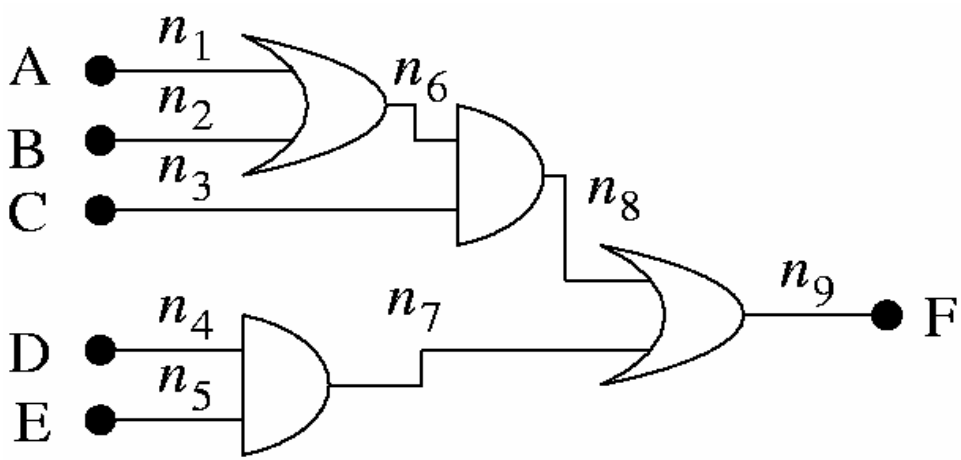


Example (6/10)

Advance time to t=3

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=0, n_7=0, n_8=0, n_9=0

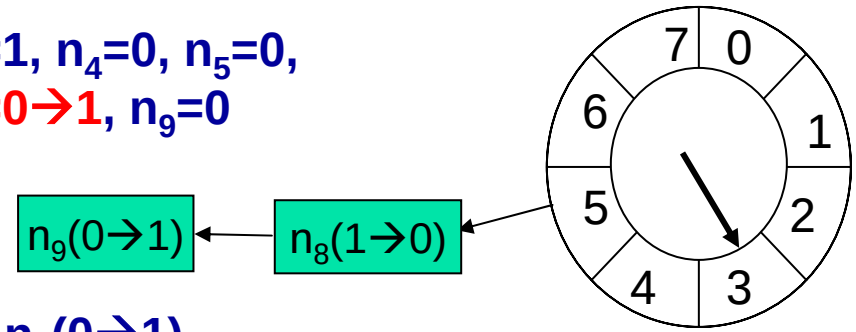




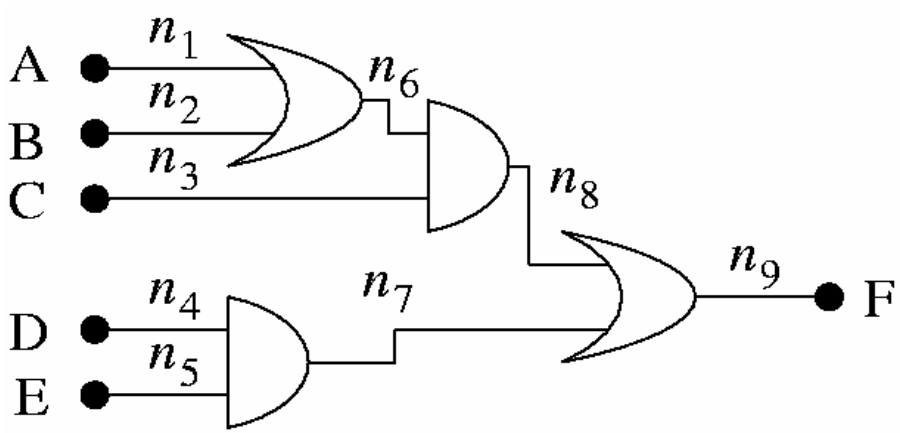
Example (7/10)

Process event n_8(0→1) at t=3

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=0, n_7=0, n_8=0→1, n_9=0



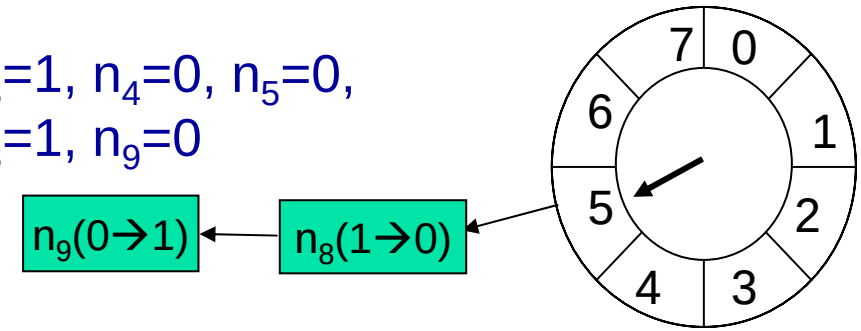
Schedule event n_9(0→1)

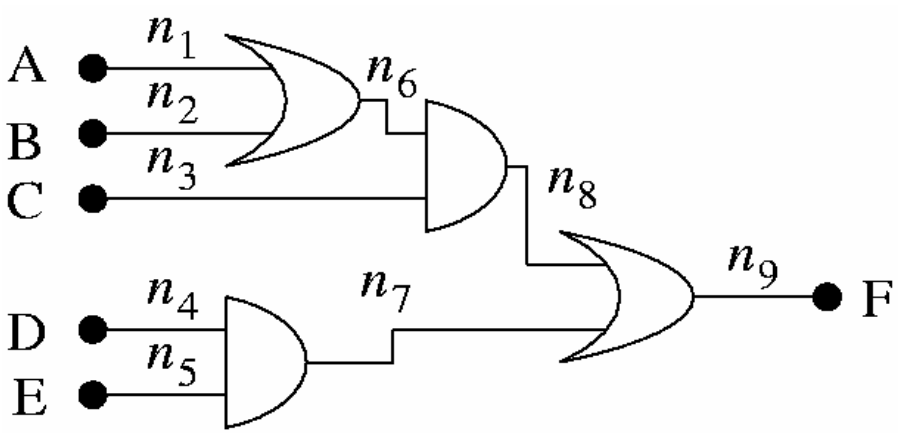


Example (8/10)

Advance time to t=4 and then to t=5

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=0, n_7=0, n_8=1, n_9=0

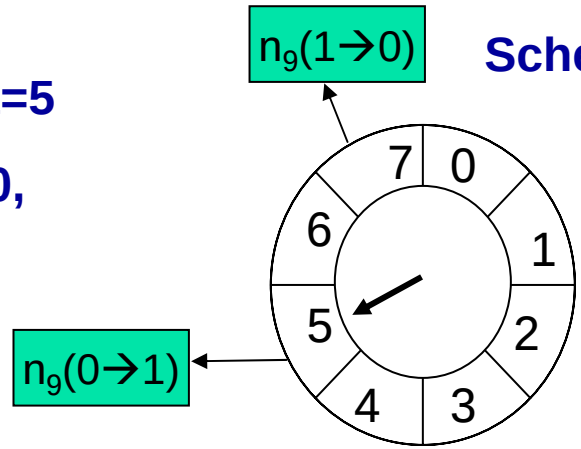


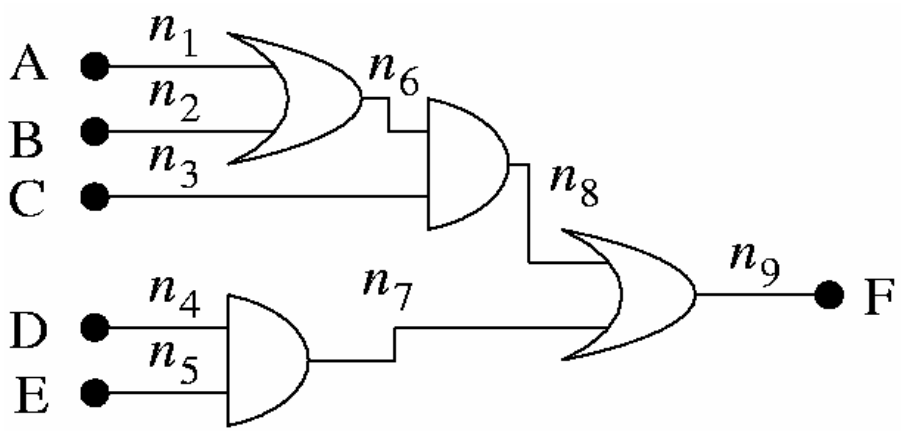


Example (8/10)

Process event $n_8(1\rightarrow0)$ at t=5

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=0, n_7=0, n_8=1→0, n_9=0

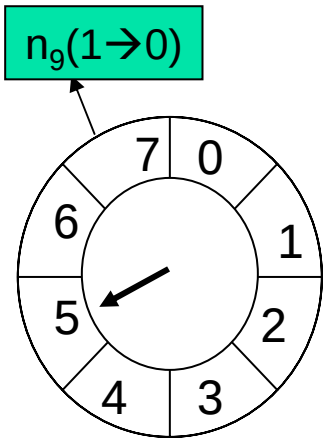


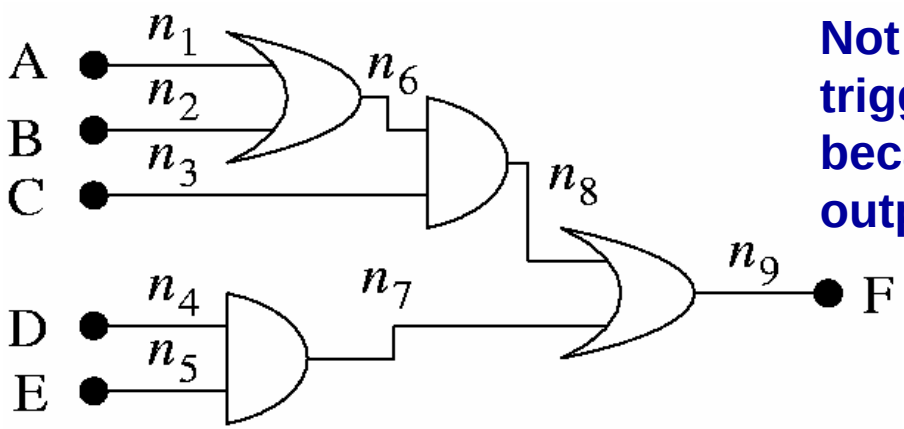


Example (9/10)

Process event $n_9(0\rightarrow1)$ at t=5

$n_1=0$, $n_2=0$, $n_3=1$, $n_4=0$, $n_5=0$, $n_6=0$, $n_7=0$, $n_8=0$, $n_9=0\rightarrow1$



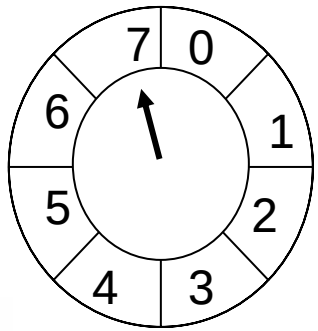


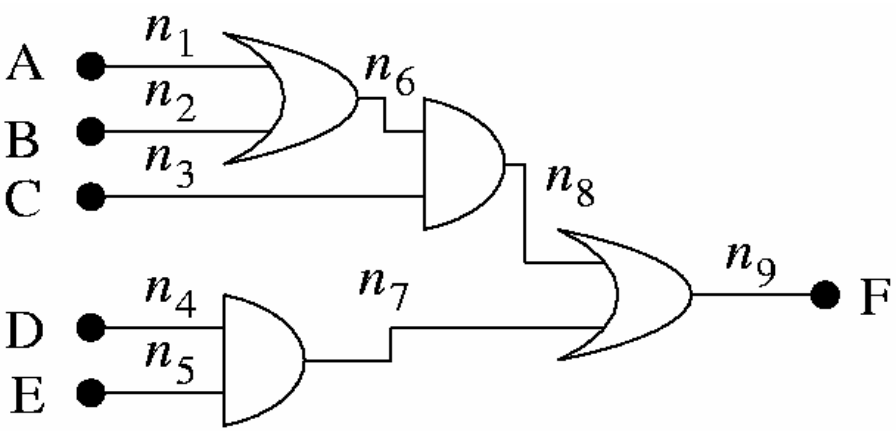
Not scheduling any triggered event because n_9 is an output.

Example (10/10)

Advance time to t=6, t=7 and then process event n_9(1→0) at t=7

n_1=0, n_2=0, n_3=1, n_4=0, n_5=0, n_6=0, n_7=0, n_8=0, n_9=1->0





Not scheduling any triggered event because n_9 is an output.

- There are hazards on n8(0→1→0) and n9 (0→1→0).
- It takes 7ns to propagate the input change to the output.

Parallel Logic Simulation

- Logic simulation is a very important task in designing of a VLSI circuit. It is used to verify a design, estimate timing performance, perform fault simulation, estimate circuit switching activity, etc. It is imperative that the simulation is done as fast as possible.
- Parallel logic simulation over a multiprocessor or a network of computers is a way to speed up the simulation performance.
- Logic partitioning that minimizes the number of events passing among processors is the key to successful implementation of parallel logic simulation.

Discussion

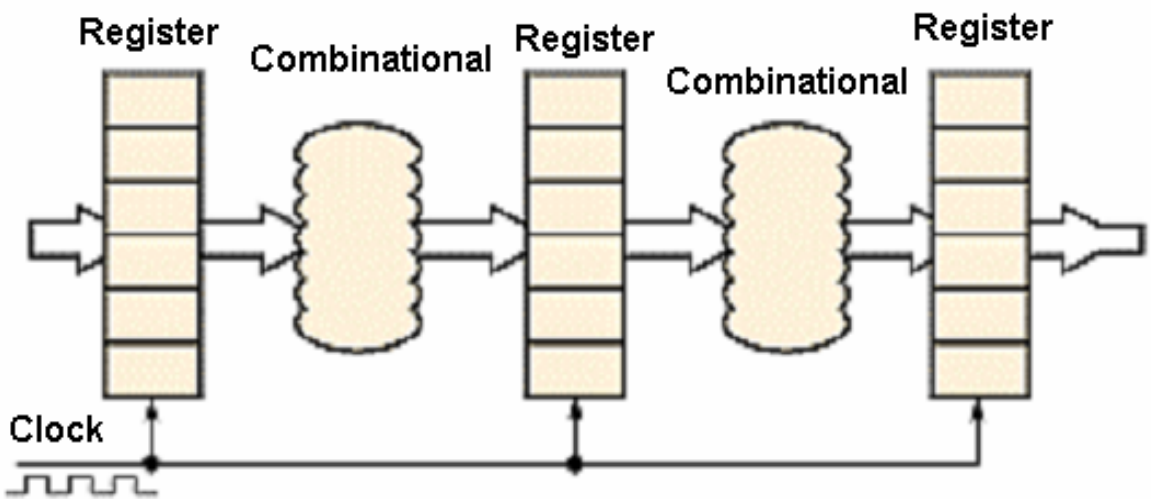
- It is more complex and difficult to use compiled code logic simulation to handle variable gate and interconnect delay.
- Event-driven simulation presented previously more flexible in handling delay.
- The bottleneck of logic simulation is the time required for simulating a large design. Logic simulation is well suited to be conducted in parallel computing systems if a design can be partitioned into many loosely connected subcircuits

Register Transfer Level (Cycle-Based) Simulation

Register Transfer Level (RTL) Simulation

- A functional simulation performed for the designs described in RTL descriptions.
- In RTL designs, the transferring of logic value into a register is governed by the arrival of a clock signal. Because a clock arrives at a register only once per cycle, the RTL simulation is best carried out by the so-called cycle-based simulation (CBS).
- The simulation time step is 1 cycle.

A classical synchronous desigr comprises globs of combinatorial logic sandwiched betwee blocks of registers.



Basic Concepts

- A cycle-based simulator evaluates register's values only when clocks arrive at the registers.
- It discards the timing information and converts the combinatorial blocks into "flat" Boolean equations or some easily evaluated formats. That is, the timing details in the combinational logic is immaterial.
- It usually uses only two logic values 0 and 1 for simulation.
- It uses memory efficiently and lets you quickly verify extremely large designs.
- It is well-suited for evaluating classical synchronous designs.

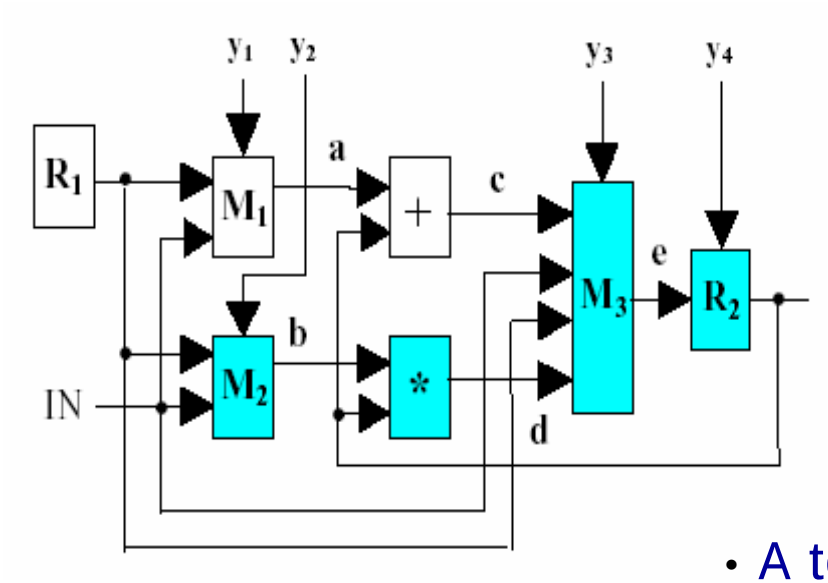
Key to Faster CBS

- Must have a efficient way to derive the output of the combinational part of the circuit given with its inputs. We can use
  - Flat Boolean expression
  - Reduced-Order Binary Decision Diagram
  - Decision Diagram if the design description is in a level higher than the gate-level description.
  - Other higher-level constructs described by C or C++ such that the design description can be directly compiled into executable code to increase simulation speed.
- The way to model the combinational part of the circuit normally decides the speed of cycle-based simulation.
  - The higher-level of abstraction implemented by high level languages will lead to more efficient simulation.

Decision Diagram (DD)

- R. Ubar, A. Morawiec, and J. Raik, “Cycle-based Simulation with Decision Diagrams,” DATE, pp. 454-458, 1999.
- A DD is a directed acyclic graph G=(M, R, x, f)
  - M: a set of nodes.
  - R: a relation in M. R(m) $\subset$ M denotes the set of successor nodes of m $\in$ M.
  - **x**: x(m) is a labeling function for the node m. The labels can be variable, algebraic expression, or constant.
  - f: f(m,n) is a labeling function on the edges between two nodes m and n where m is the parent node and n is the child node.

Example of Decision Diagram (DD)



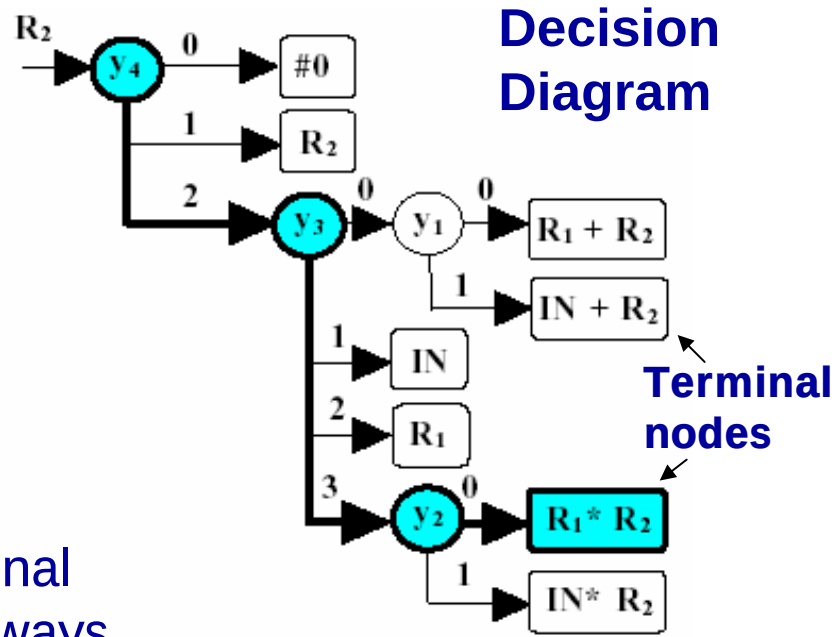
Original RTL behavior

M_i: Mux

R_i: Register

IN: Inputs

y_i: Control signal

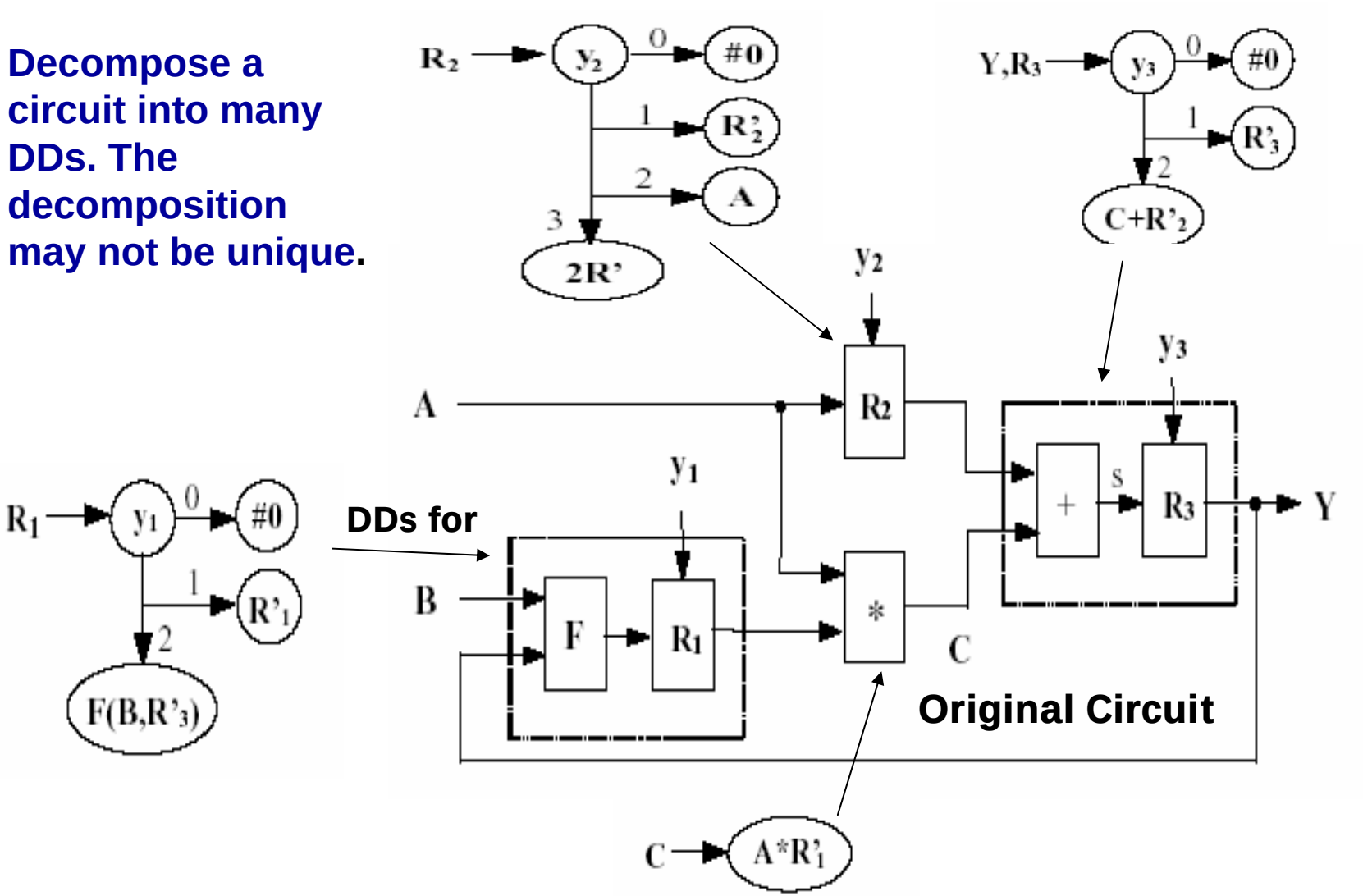


- A terminal node always leads to an evaluation of an expression.

Given y_4=2, y_3=3, and y_2=0, an **activated path** leading to R_1*R_2 is formed.

DDs for Data Path

- **Decompose a circuit into many DDs. The decomposition may not be unique.**



DDs for Control Part

- The control logic for the above example can be synthesized into a finite state machine (FSM).

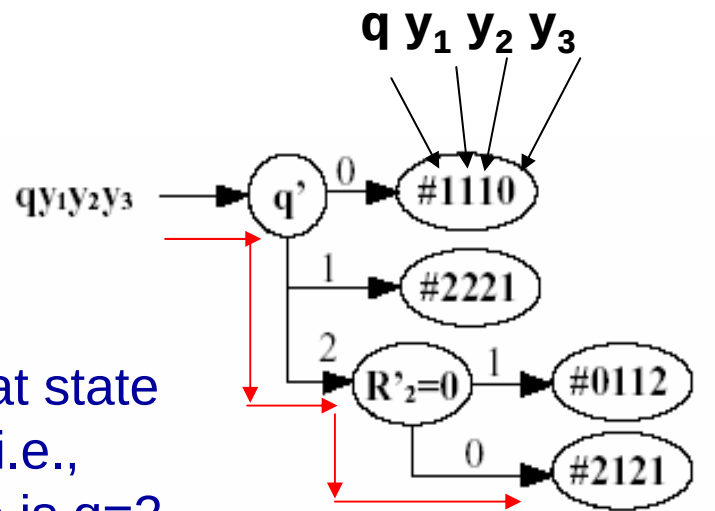
State/Output table

q'	x	q	$y_1\ y_2\ y_3$	Activities
0		1	1 1 0	y_3:R_3 := 0
1		2	2 2 1	y_1:R_1 := B, y_2:R_2 := A
2	R'_2=0	0	1 1 2	y_3:R_3 := A*B
2	R'_2≠0	2	1 2 1	y_2:R_2 := A

- Next state function
  q=F(q',x)
- Output function
- y_1=H1(q',x)

  y_2=H2(q',x)

  y_3=H3(q',x)



The highlighted path indicates when q' is at state 2 and the input x=0 (i.e., R'_2=0), the next state is q=2, the output is $y_1y_2y_3$=121.

Behavior-Level Simulation

Behavior Model for Digital Designs

- It is normally described in high level languages for efficient simulation without structural information.
  - For example, a 32-bit by 32-bit multiplication is simply modeled as A*B rather than being modeled as a Booth multiplier formed by some basic logic gates.
- For a design described in HDL, an effective behavioral simulator should be built for behavior-level simulation and a modeling library for efficient simulation should also be created.
- For a design described in System C, C, C++, or other high level languages, the design itself can be complied into executable code that can be run directly on a machine. A library should also be provided for the compiler.

Behavior Level Simulation for Digital Designs

- If the behavior model is implemented in high level language such as C, C++, etc., behavior-level simulation is fulfilled by running the executable code.
  - The way how the behavior of a design is modeled, rather than the simulator itself, mainly decides the behavior level simulation speed.
- If the behavior model is implemented in HDL, an HDL simulator must be implemented to execute the behavior model.
  - The simulation speeds also depends on the efficiency of the HDL simulator.
- The behavior of a digital design should be correctly modeled. This is what behavior level simulation would like to achieve.

Behavior Model for Analog Designs

- The behavior of an analog circuit can be described by a set of linear or non-linear equations in time domain or frequency domain.
- To know how an analog circuit responds to an external event, an evaluation of equations is performed to generate response.
- The equations can be
  - simple (first order approximation for fast simulation).
  - Second order approximation for moderate simulation speed and accuracy.
  - Higher order approximation for accurate simulation.
- There is a tradeoff between modeling accuracy and simulation speed (note that this point is quite different from the modeling of a digital design).

Example of an Analog Behavior Model

- Using MAST Language for example (MAST is a modeling language for SABER which is a mixed-signal simulator, see http://www.analogy.com).

Simple Resistor

```
template resistor p m = rnom
        electrical p, m                     # Declare pins as electrical
        number rnom                         # Nominal resistance value
        {
        branch v = v(p,m),i = i(p->m)       # Resistor voltage and current
        v = i*rnom                          # Ohm's Law
        }
```

The behavior model for a resistor must satisfy Ohm's Law.

More Complicate Analog Behavior Model

- More complicate behavior models for some analog circuits such as VCO (voltage controlled oscillator), PLL(phase locked loop), ADC (analog to digital converter) are more difficult to obtain.
  - Extensive characterization may be performed to extract important parameters for faster simulation speed.

Behavior Level Simulation for Analog Designs

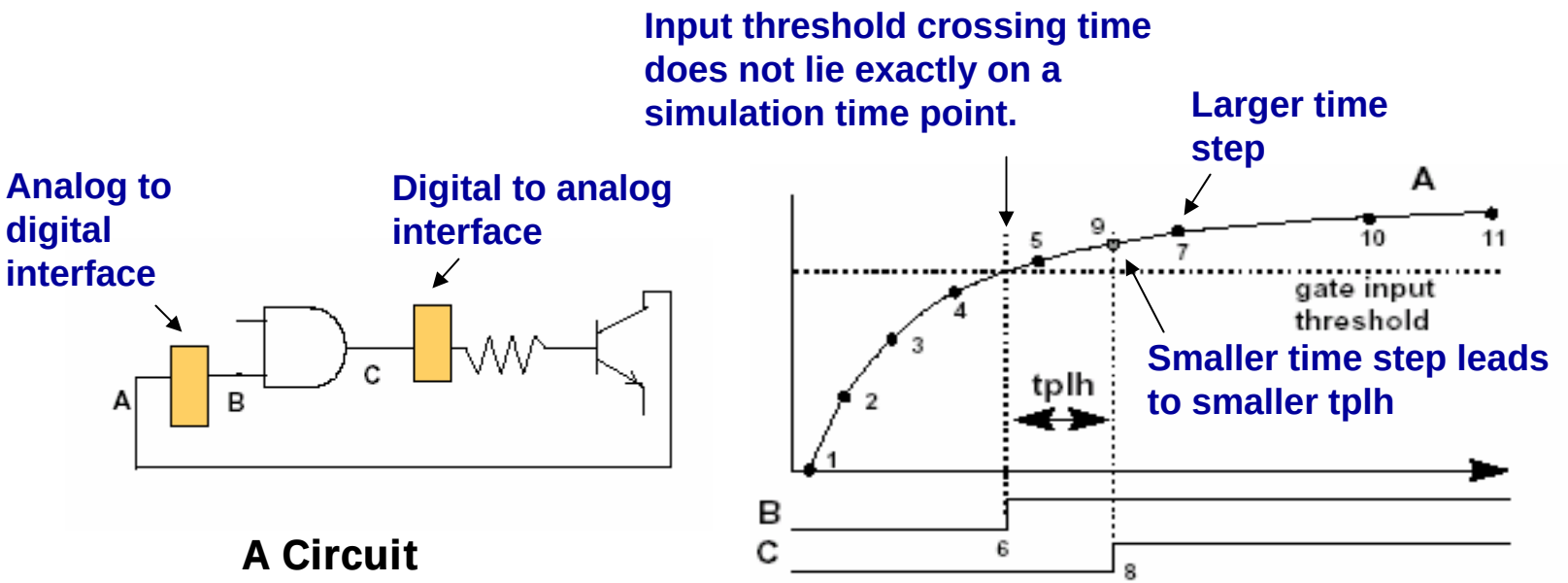
- The modeling of a design itself and the algorithms used to solve a system of linear equations is the key to the simulation speed. Normally, a simulator always faces the trade-off between accuracy and simulation speed.
- Analog simulators work by guessing the next time step based on the previous one or two time points. A better way to predict the time step is to vary it during simulation.
  - Make time step smaller when there is a lot of signal activities.
  - Make time step larger when signal is quiet.
- The predicted time step may not be able to make the solution satisfying the desired accuracy. Thus, time must be rolled back and a smaller time step is used.

Mixed-Level Simulation

- The design is described in different levels of abstraction.
  - More critical parts that would decide the performance of the whole design may be described in transistor or gate level, while the rest can be described in RTL or behavior level for efficient simulation.
- multiple simulators must work together to carry out mixed-level simulation.
  - At this situation, synchronization of simulation time plays an important role for correct implementation of a mixed-level simulator.
  - Recalled that the time step for SPICE (analog simulator) is not predictable, for transistor-level and gate level simulators is event dependent, for cycle-based simulation is 1 cycle, and for digital behavior level simulation is not even clearly defined.
  - Synchronization of simulation time is not a trivial task.

Mixed-Signal Simulation

- The simulation involves dealing with both digital and analog signals.
- Like mixed-level simulation, multiple simulators must work together to perform mixed signal simulation.
  - Synchronization of simulation time is the key.
  - The selection of time step in analog simulator will have an influence on the time when the input of a logic gate will pass its logic threshold.



Instruction Set Simulation

- An instruction set simulator (ISS) simulates how a CPU executes the instructions of a program.
  - For verifying the function of the CPU.
  - For evaluating the performance of the CPU.
  - For exploring the architectural trade-off.
  - For evaluating an instruction set.
- The primary design consideration for an ISS
  - Rapid execution by ignoring the timing and architectural information for functional verification, or
  - Detailed implementation of architectural feature with timing information for architectural design space exploration and performance evaluation.

Public Domain ISS: SimpleScalar

- Simplescalar is a powerful simulation tool that provides both detailed and high-performance simulation of modern microprocessors.
- The Simplescalar toolset can perform fast, flexible, and accurate simulation of modern processors to accelerate hardware development and design a high-performance machine.
- http://www.simplescalar.com

Simulator	Description	Lines of code	Simulation speed
Sim-safe	Simple function simulator	320	6MIPS
Sim-fast	Speed-optimized function simulator	780	7MIPS
Sim-profile	Dynamic program analyzer	1300	4MIPS
Sim-bpred	Branch predictor simulator	1200	5MIPS
Sim-cache	Multilevel cache memory simulator	1400	4MIPS
Sim-fuzz	Random instruction generator and tester	2300	2MIPS
Sim-outorder	Detailed micro-architectural timing model	3900	0.3MIPS

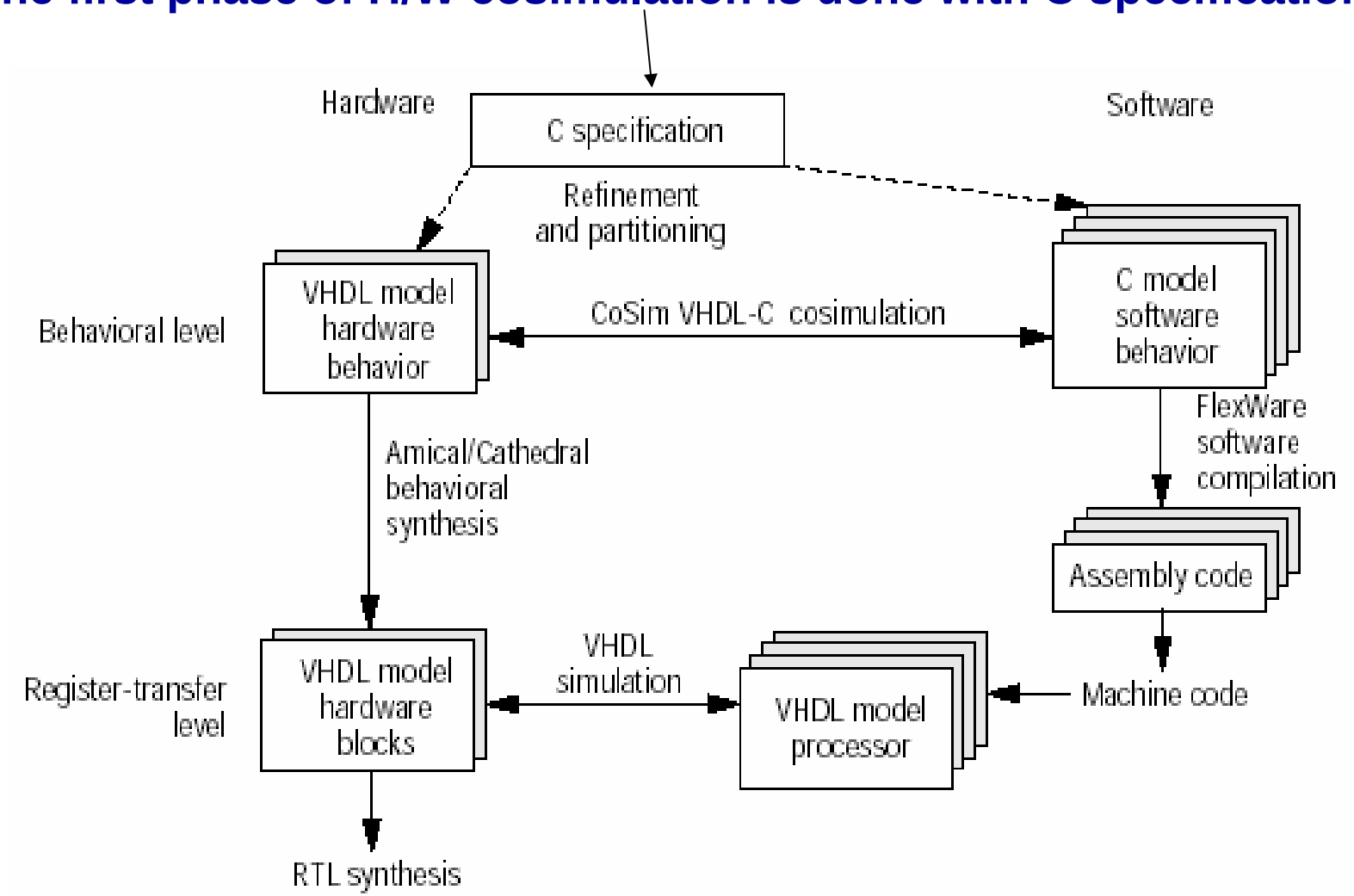
The above table shows the ISS in SimpleScalar can be configured to implement different architectural options.

Hardware/Software CoSimulation

- Hardware normally consists a processor that could execute a software program.
- The cosimulation of hardware and software can occur in different level of abstraction.
  - The hardware can be described in transistor, gate, RTL, or behavior level.
  - The software can be simply described in high level language such as C, or C++, or even in HDL.
- If hardware and software are described in the same HDL language, cosimulation will be simpler, but multiple simulators might still need to work together to perform cosimulation if hardware and software are not described in the same level of abstraction.
- If hardware and software are described in the same high level language, cosimulation can be easily done because both the hardware and software can be compiled into executable code. Then cosimulation is carried out by simply executing the executable code.

A Flow of H/W CoSimulation

The first phase of H/W cosimulation is done with C specification.



System-Level Simulation (SLS)

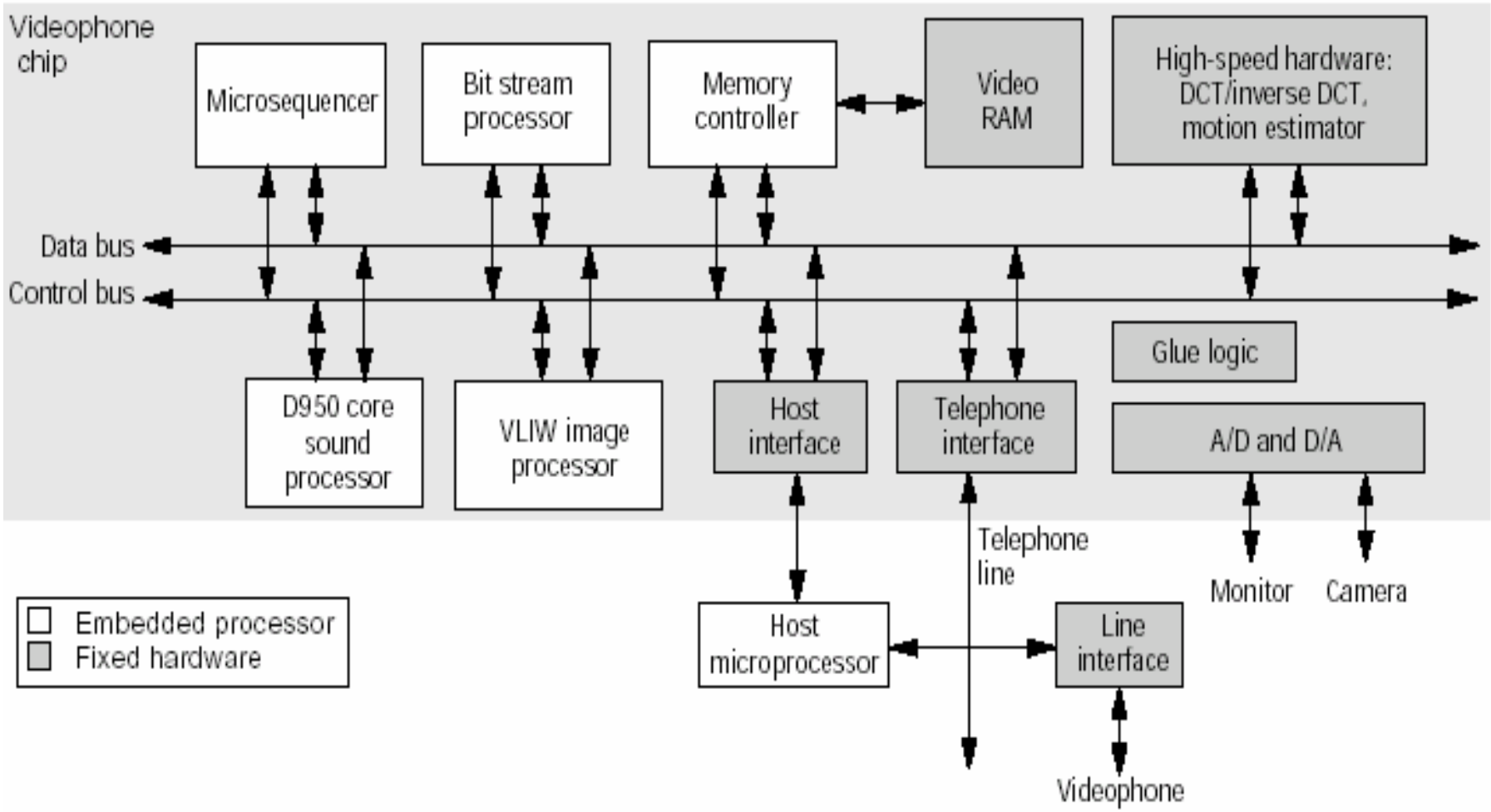
What Is a System ?

- A set or arrangement of things so related or connected as to form a unity or organic whole. (From Webster's Dictionary).
- Things can be a chair, a desk, a computer, a person, a camera, etc. These things can be so related or connected as to form a system of working place where a person sits on the chair using the computer on the desk and holding the camera to take a picture and down load it into the computer.
- From the definition, systems form a hierarchy. That is to say, a thing within a system could be a system itself.
- A system can be too complicate to be described and validated effectively. Thus, to describe a system effectively for system-level simulation requires a multi-disciplined core team that understand the details of the system.
- We satisfy ourselves here to deal with a system consisting of only electronic devices. However, the same concepts can be applied to a system that may contains some optical or mechanical devices.

An Electronic System

- What do you see in the following picture?
  - Many chips
  - Passive devices such as resistors, capacitors, …, etc.
  - Mechanical switches
  - A board holding electronic components
  - A system of connected components for doing something

An Example of an Electronic System



Single-chip videophone. DCT: discrete cosine transform.

System Modeling and Model Library

- For system-level simulation we must have a way to describe the behavior of the things (components) we see in a system and the relations among the things. This is called “system modeling.”
- Normally, the system description for simulation may involves
  - Using of different languages such as C, C++, VHDL, Verilog, etc.
  - Modeling of digital components as well as analog components.
  - Modeling of non-electronic components such as motors, MEMS (**Microelectromechanical Systems),** optical devices, etc.
- The behavioral models of components form a system model library which is used in high-level synthesis and system-level simulation (SLS).
- System modeling should be done in a manner to maximize simulation speed without losing the accuracy.
- Model development itself represents substantial extra cost of unclear value.

Purposes of SLS

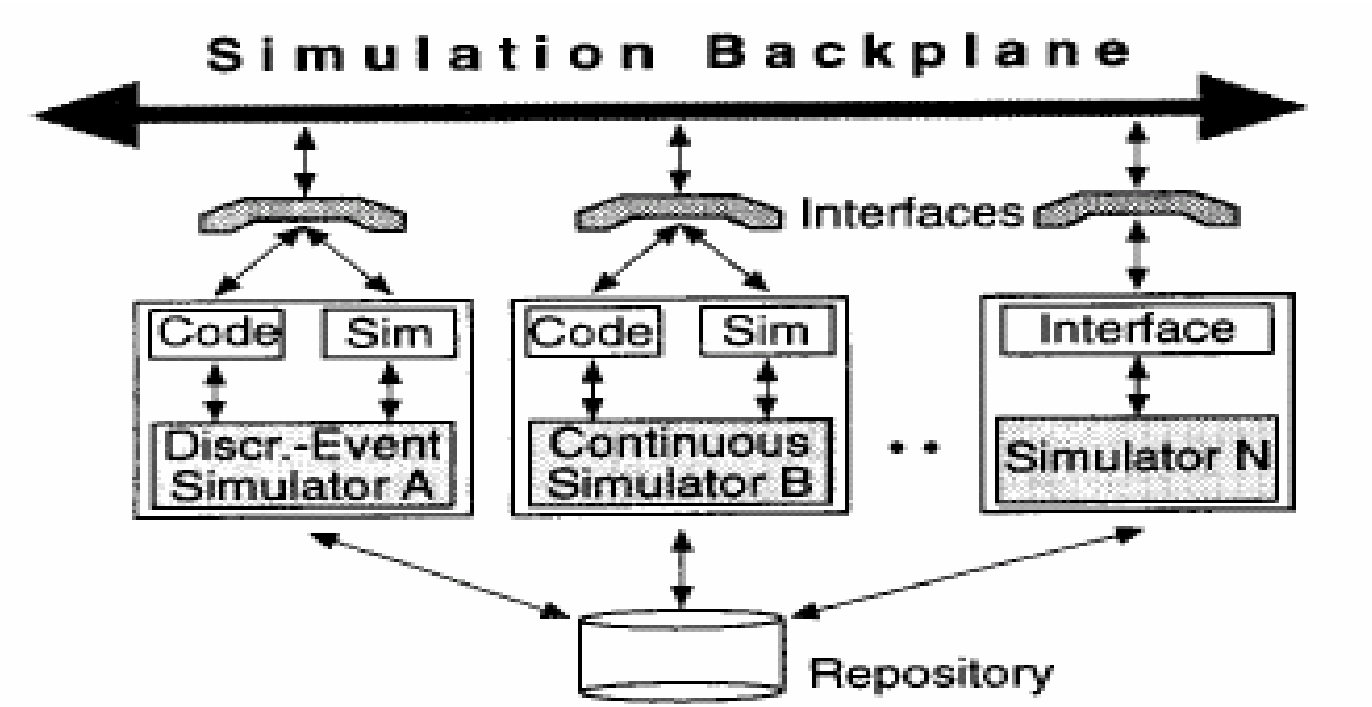
- System Validation
  - For a large system that is described in different languages and different design levels, system-level simulation (SLS) is often the only way to validate the correctness of system design.
- Design Space Exploration
  - Exploring the system design parameters to find out the trade-off among performance, cost, power, design complexity, etc.

Who Should Perform SLS?

- System designers
  - Should build a model for the targeted system using the models that describe the functions of components. The purpose is to validate the system function and system performance and carry out design space exploration
- Component designers
  - Should build a model for its component and a model for the targeted system to validate whether its component could perform its function correctly within a targeted system.
- Both need to set up a testbench to perform SLS, which itself is a complicate task.
- Generally, it requires an SLS platform that weaves various kinds of simulators together to perform SLS.

System-Level Simulation Backplane

- Mixed-level, mixed-mode (signal), hardware/software cosimulations usually work under a simulation backplane to carry out SLS.
- The synchronization of simulation time step is the foremost important task for a simulation backplane which integrates different simulators.



Discussions

- Simulation is sometimes the only way to verify a design and decide a design's performance.
  - Achieving both objectives need to develop a set of stimuli. While we are not able to exhaust all the stimuli for a large design, carefully developing a set of effective stimuli is very important.
    - For design verification, we may have to pay more attention to the corner cases.
    - For performance evaluation, we may have to find a stimulus to activate some particular path or a set of stimuli that reasonably represent to the real world situation.
- Timing simulation can be performed at different level of abstraction, for example
  - At transistor, gate, and RTL levels, we are interested in the delay of the longest path and the number of cycles to complete a certain task.
  - At behavioral level, we may concern with only the number of operations to complete a task.
- Hardware emulation is another effective way to speed up simulation speed if gate-level description of a design is available.

Summary

- We have studied different kinds of simulations for simulating the designs described in different level of abstractions.
  - Circuit simulation for analog and digital design verification.
  - Transistor-level simulation for circuits being modeled as a network of bidirectional switches.
  - Gate-level simulation for circuits being implemented into a network of basic logic gates.
  - RTL-level (Cycle-based) simulation for circuits whose combinational parts are described in a more abstracted manner without caring about timing information.
  - Behavior-level simulation for digital circuits being described in high level language or HDL without knowing the number of cycles required for perform its function. For analog circuits being described in a manner to trade off simulation speed and accuracy.
  - System-level simulation using a simulation backplane
    - Mixed-level, mixed-signal, H/W cosimulation, etc.