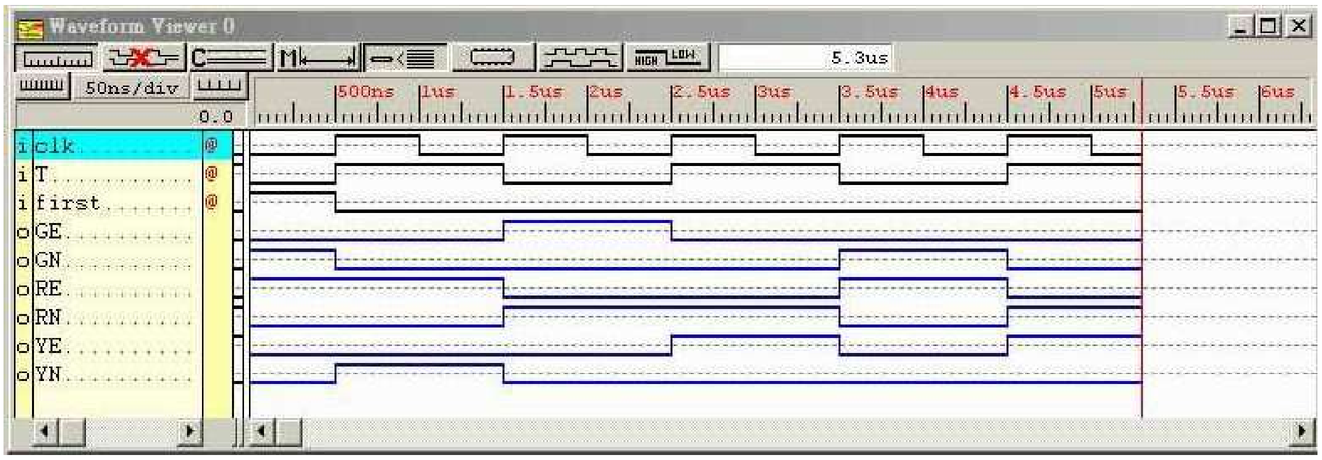
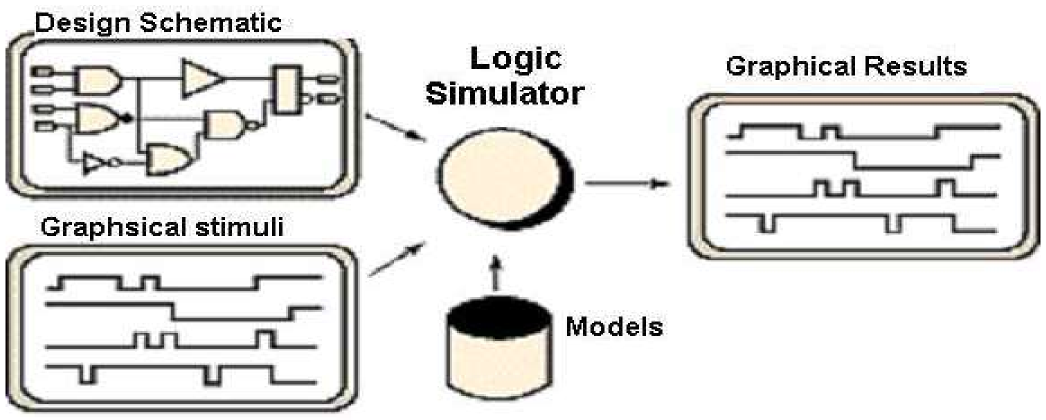
Unit 7: Simulation

- Content
  - Circuit Simulation
  - Event-Driven Simulation
  - Gate-Level (Logic) 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, it 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.

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

- Behavior-level simulation
  - The goal is to achieve highest simulation speed
  - Designs are described in higher-level abstraction using HDL languages (VHDL, Verilog, SystemC, C/C++, …)
  - Focuses more on verifying the function of a design rather than the timing performance
- Instruction set simulator
  - The goal is to verify the design of a particular CPU and evaluate its performance under certain workload
    - Input/output values only, no hardware details
  - The program input to an instruction set simulator is an executable code segment (machine code)
  - Normally written in high-level languages (ex: C/C++) to achieve highest execution speed
  - It plays an important role in hardware/software co-simulation

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   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

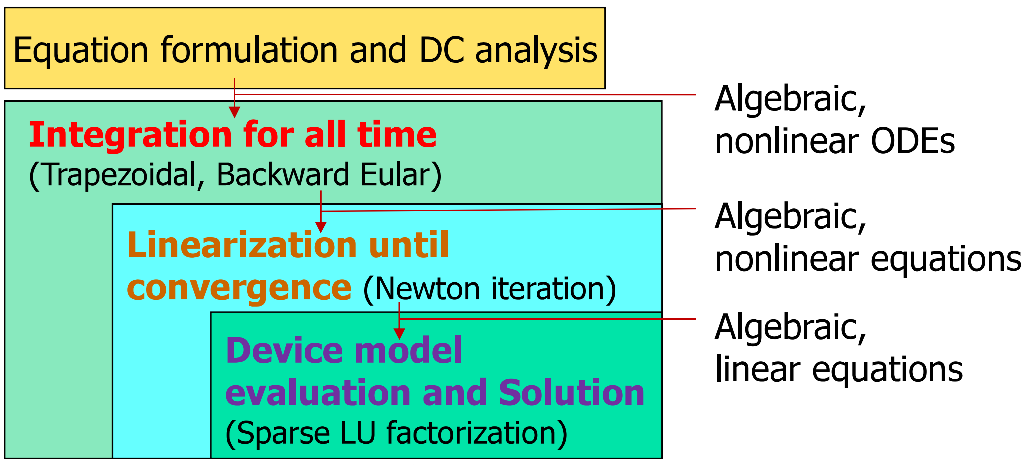
- For Nodal Analysis, the (non)linear system describing a circuit network can be formulated as follows:

$$\mathbf{I}=\mathbf{Y}\mathbf{U}$$

  - $\mathbf{I}$ is the vector consisting of *independent* current sources.
  - $\mathbf{Y}$ is the system matrix consisting of admittance terms from all components *except* from independent sources.
  - $\mathbf{U}$ 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.


Equation formulation and DC analysis
Integration for all time
(Trapezoidal, Backward Eular)
Linearization until convergence (Newton iteration)
Device model evaluation and Solution
(Sparse LU factorization)
Algebraic, nonlinear ODEs
Algebraic, nonlinear equations
Algebraic, linear equations

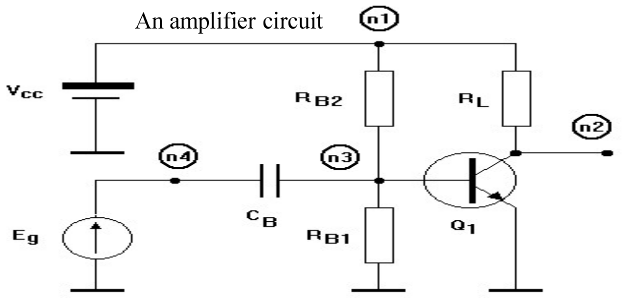
DC Analysis

- All dynamic components such as capacitances and inductances are ignored in this analysis.
  - Capacitors -> open; inductors -> short
  - If the circuit is nonlinear, Newton method is repeated until converge

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

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
```



DC analysis results

Operating point

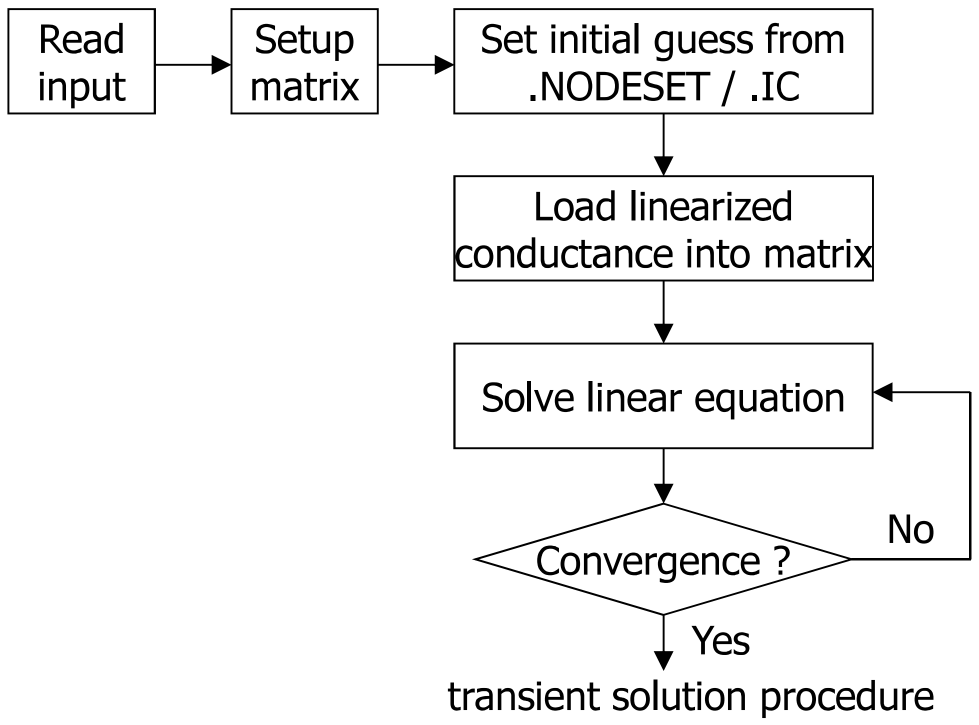
Vce = 5.72

Vbe = 697.65m

Ic = 4.27m

DC Analysis Algorithm

- Linearization and matrix calculations are the key !!

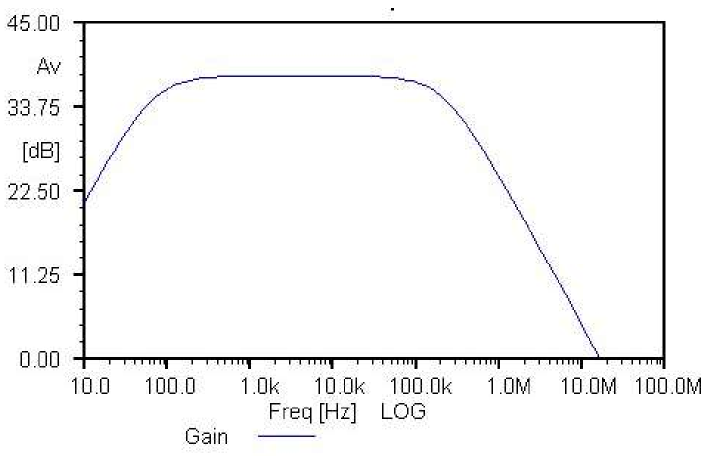


Read input
Setup matrix
Set initial guess from .NODESET / .IC
Load linearized conductance into matrix
Solve linear equation
Convergence ?
No
Yes
transient solution procedure

AC Analysis

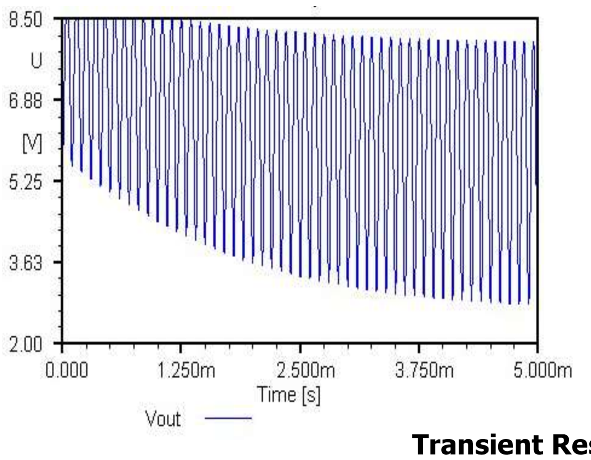
- AC analysis is used for the frequency response
  - Mainly in connection with amplifiers and filters
  - Usually based on a sweep over a range of frequencies
- Electronic device are represented by a linearized model of complex admittance about it operating point
  - Ex: C = 1 / jωC, L = jωL

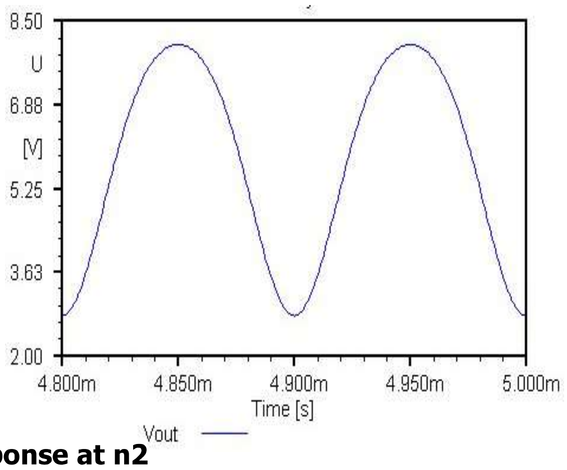
Result of AC analysis



Transient Analysis

- Transient, or time-domain, analysis simulates the phenomena seen in the real circuit by an oscilloscope
  - A simulation consists of a time sweep starting at t=0
- Energy storage elements contribute ordinary differential equations (ODEs) to system equations (ex: I = C*dV/dt)
- Error-prone due to the truncation of Taylor series
  - Need time-step control engine → active circuit needs small steps

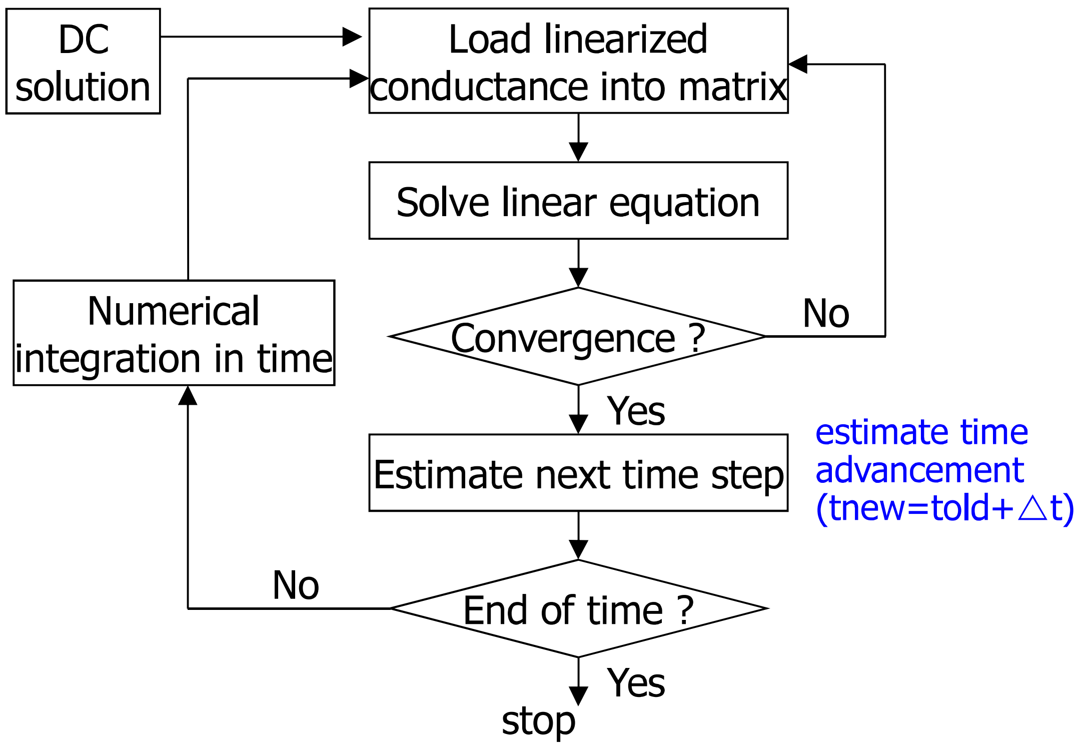




Transient Response at n2

Transient Analysis Algorithm

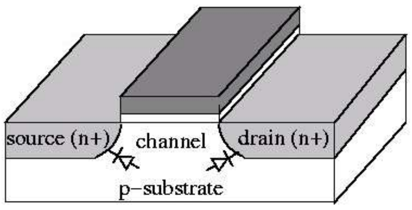
- ODE solvers and time step controls are the key !!

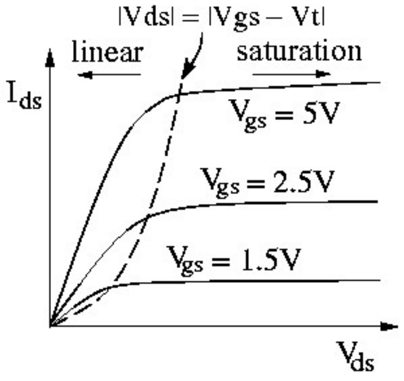

DC solution
Load linearized conductance into matrix
Solve linear equation
Convergence ?
No
Yes
Numerical integration in time
Estimate next time step
estimate time advancement (tnew=told+△t)
End of time ?
No
Yes
stop

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_{t0})
  - 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:** $V_{ds} < V_{gs} - V_t$

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

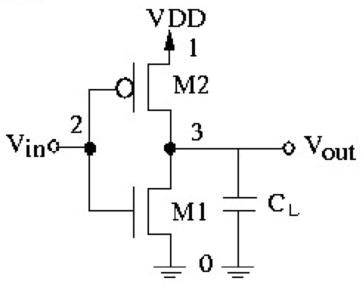
- **Saturated region:** $V_{ds} \geq V_{gs} - V_t$

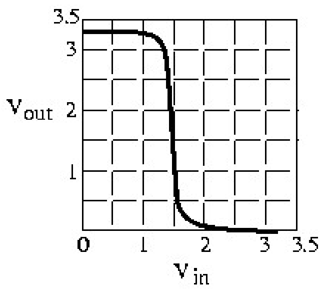
$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_t)^2$$

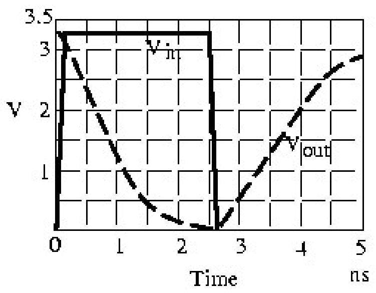
  - V_{gs} (V_{ds}): gate-to-source (drain-to-source) voltage
  - I_{ds}: current between the drain and source
  - V_t: transistor **threshold voltage**
  - β: **Transistor gain factor** $\left(\beta = k'\frac{W}{L}\right)$
  - k': transistor **transconductance** $\left(k' = \frac{\mu\varepsilon}{t_{ox}} = \frac{\partial I_{ds}}{\partial V_{gs}}\Big|_{V_{ds}=constant}\right)$ (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
```







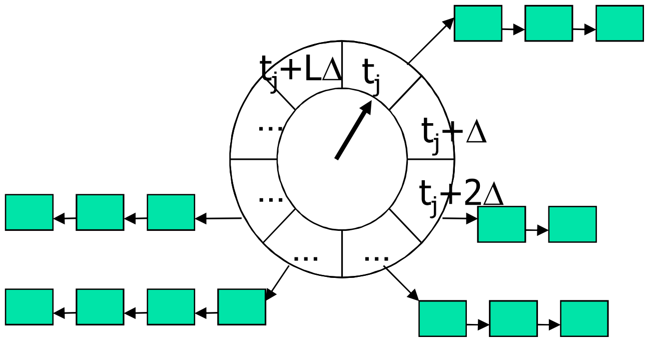
Event Driven Simulation

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.



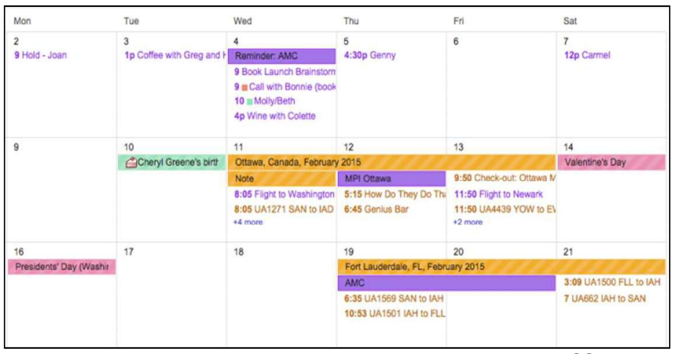
t_j is current time

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

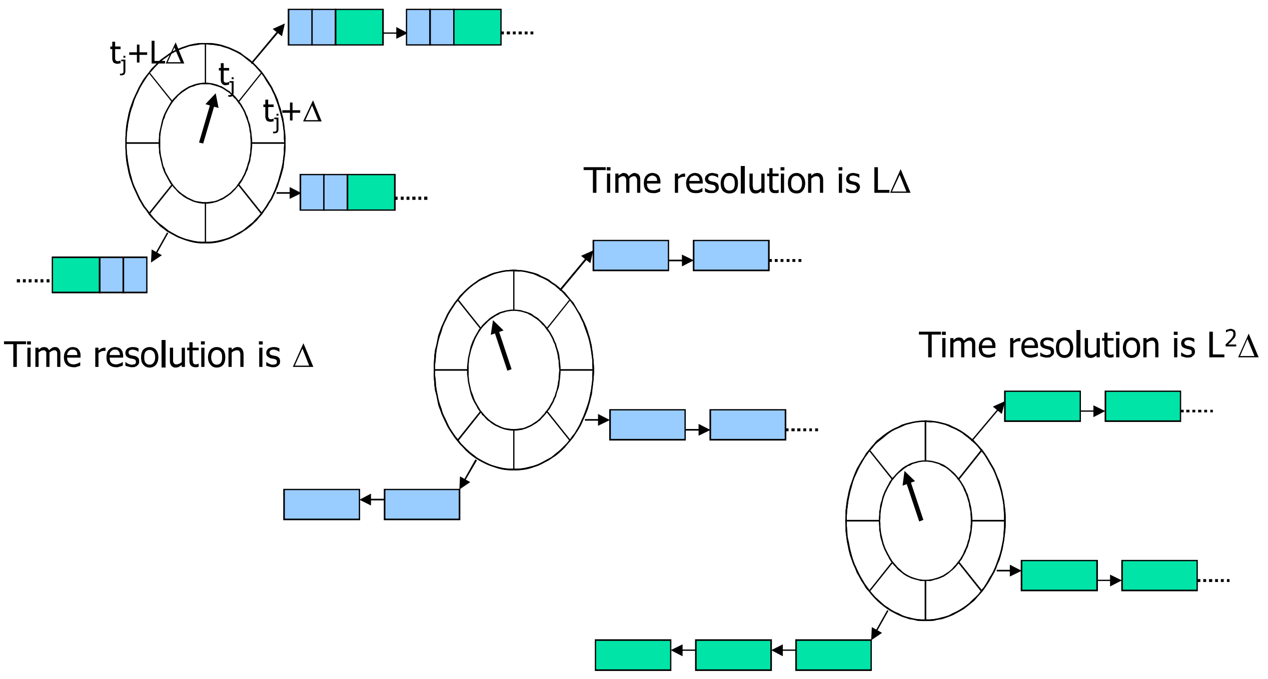
- Remove an event from the linked list pointed by the current time
- Evaluate the event.
- Schedule all events triggered by the evaluation of the above event to the linked-list based on their occurring times.
- When the linked list pointed by the current time is empty, advance the current time by Δ.
  - 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.
- Similar to manage the calendar
  - Add events in arbitrary order, but execute them in order



Mon	Tue	Wed	Thu	Fri	Sat
2 9 Hold - Joan	3 1p Coffee with Greg and H	4 Reminder: AMC 9 Book Launch Brainstorm 9 Call with Bonnie (book 10 Molly/Beth 4p Wine with Colette	5 4:30p Genny	6	7 12p Carmel
9	10 Cheryl Greene's birth	11 Ottawa, Canada, February 2015 Note 8:05 Flight to Washington 8:05 UA1271 SAN to IAD +4 more	12 MPI Ottawa 5:15 How Do They Do Th 6:45 Genius Bar	13 9:50 Check-out: Ottawa M 11:50 Flight to Newark 11:50 UA4439 YOW to EV +2 more	14 Valentine's Day
16 Presidents' Day (Washir	17	18	19 Fort Lauderdale, FL, February 2015 AMC 6:35 UA1569 SAN to IAH 10:53 UA1501 IAH to FLL	20	21 3:09 UA1500 FLL to IAH 7 UA662 IAH to SAN

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.



$t_j+L\Delta$
t_j
$t_j+\Delta$
Time resolution is $L\Delta$
Time resolution is Δ
Time resolution is $L^2\Delta$

Gate-Level (Logic) Simulation

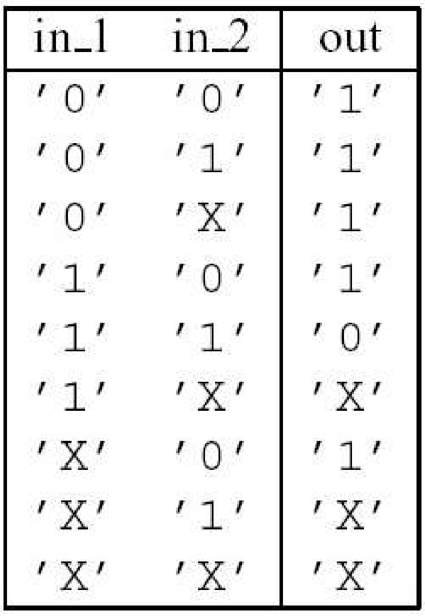
Signal Modeling for Gate-Level Simulation

- 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.

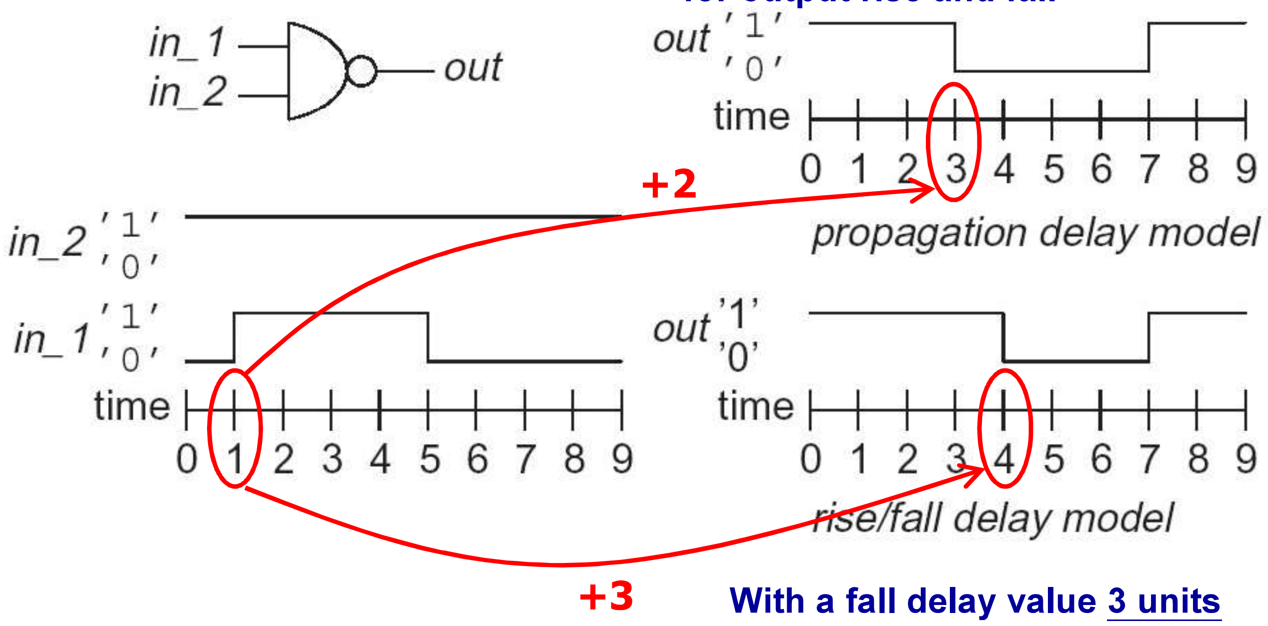
Timing Library For Logic Simulation

- 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

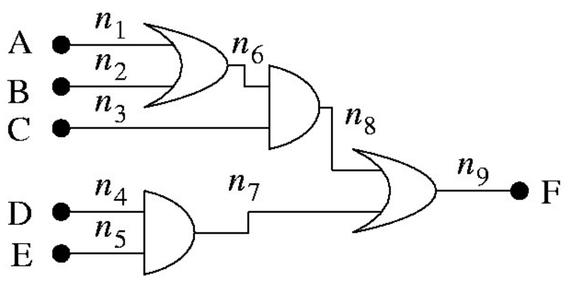


With a single delay value 2 units for output rise and fall

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

Compiler-Driven Simulation

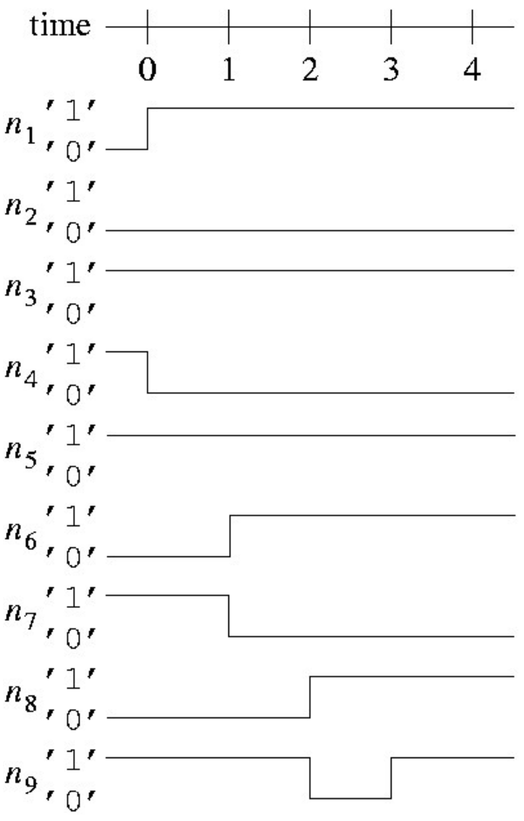
- **Compiler-driven simulation** is often used in gate-level simulators.
- Based on making an executable-code model of a circuit.
- Efficient simulation mechanism (few machine instructions per gate).
- Applicable to few delay models in synchronous circuits (e.g. zero-delay model).

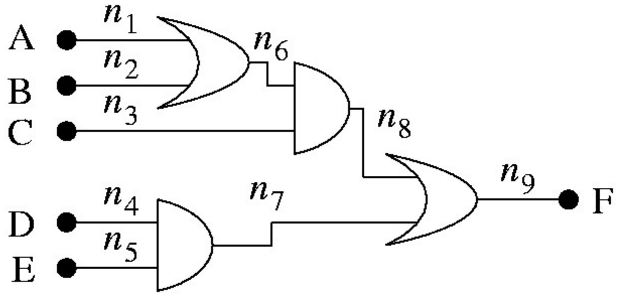


$n_1 \leftarrow$ A;
$n_2 \leftarrow$ B;
$n_3 \leftarrow$ C;
$n_4 \leftarrow$ D;
$n_5 \leftarrow$ E;
$n_6 \leftarrow$ OR(n_1, n_2);
$n_7 \leftarrow$ AND(n_4, n_5);
$n_8 \leftarrow$ AND(n_6, n_3);
$n_9 \leftarrow$ OR(n_7, n_8);
F $\leftarrow n_9$;

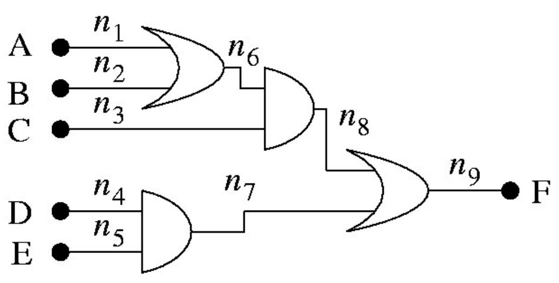
Unit-Delay Simulation

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





Compiled Code for Unit-Delay Simulation



```
for ($t \leftarrow t_{start}$; $t \leq t_{end}$; $t \leftarrow t + 1$) {
    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;
}
```

Unit-delay help to keep the execution in order !!

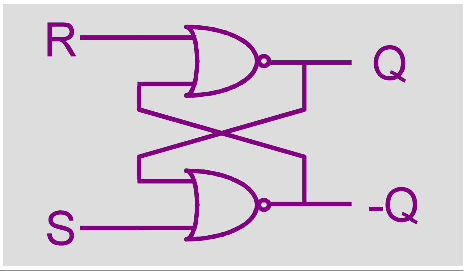
Event-Driven Logic Simulation (EDLS)

- 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

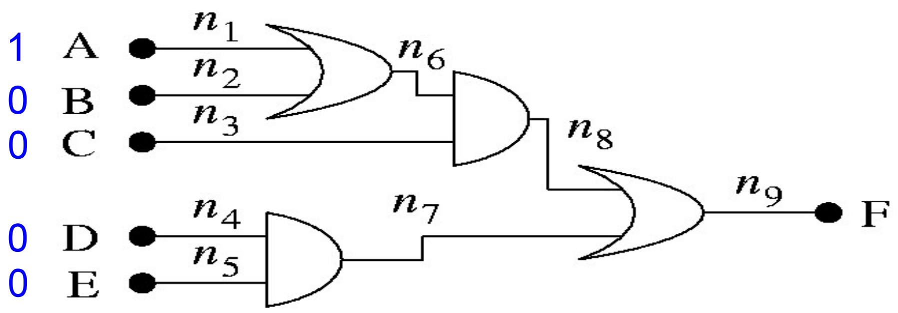
- Process a series of ordered events
- event-list : pending events
- time-wheel: simulation time

```
While (event-list != 0)
    advance time wheel
    determine current events
    update values
    propagate events
    evaluate active events
    schedule new events
```



Time	R	S	Q	-Q
0	0	1	1	0
1	-	0	-	-
2	1	-	-	-
2.1	-	-	0	-
2.2	-	-	-	1
3	0	-	-	-
4	-	1	-	-
4.1	-	-	-	0
4.2	-	-	1	-

Example (1/8)



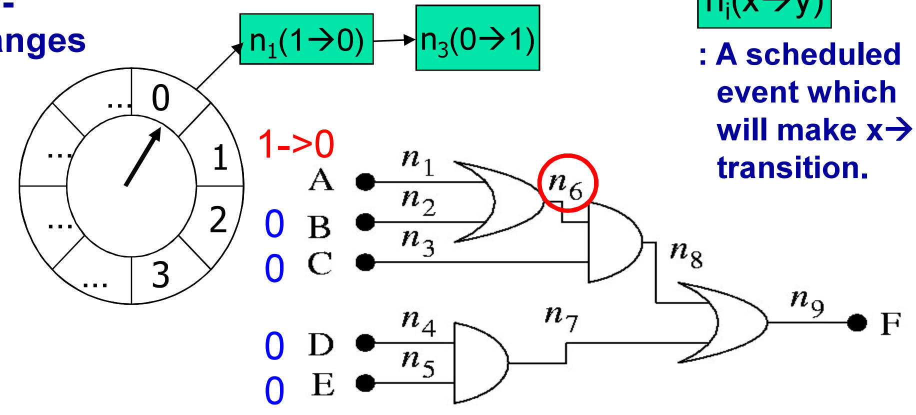
- Suppose the two-input ORs have a propagation delay of **2 ns** and two input AND gates have a propagation delay of **3 ns**. Suppose the time resolution for simulation is 1 ns (i.e., Δ = 1ns).
- Suppose at time zero, the five inputs go through the following logic value changes:
  – A: 1→ 0, B:0→0,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

Example (2/8)

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

Current time t=0

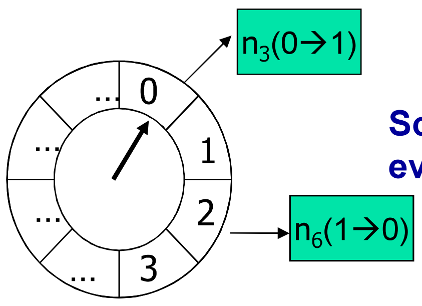
$n_1=1, n_2=0, n_3=0, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0$



$n_i(x\to y)$: A scheduled event which will make $x\to y$ transition.

Process event $n_1(1\to0)$ at t= 0

$n_1=1\to0, n_2=0, n_3=0, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0$

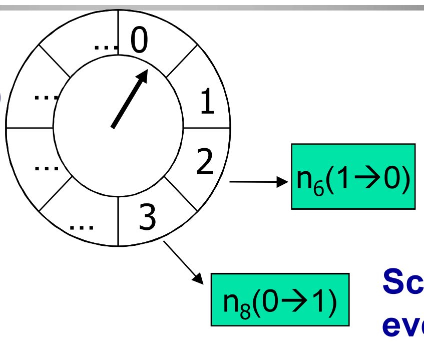


Schedule the triggered event $n_6(1\to0)$ (delay=2ns)

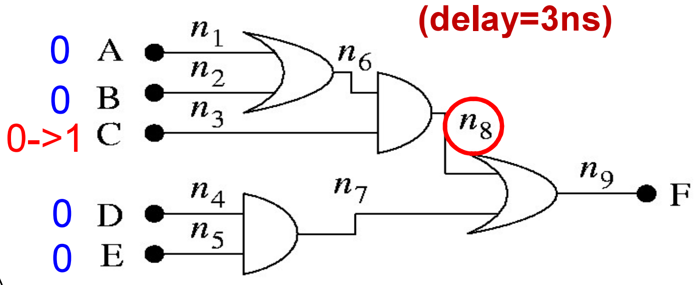
Example (3/8)

Process event $n_3(0\to1)$ at t=0

$n_1=0, n_2=0, n_3=0\to1, n_4=0, n_5=0, n_6=1, n_7=0, n_8=0, n_9=0$

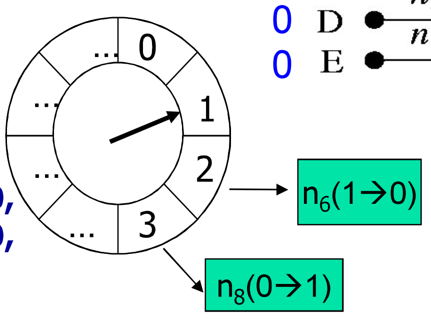


Schedule the triggered event n_8 $(0\to1)$ (delay=3ns)



Advance current time by one resolution unit to t = 1

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

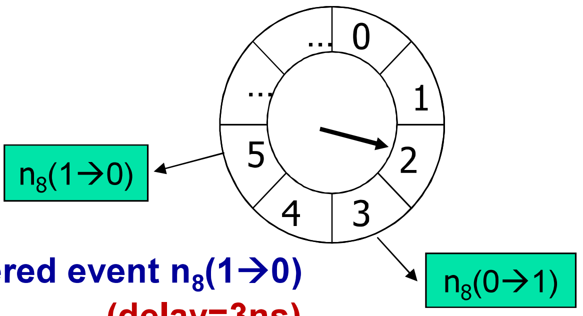


Example (4/8)

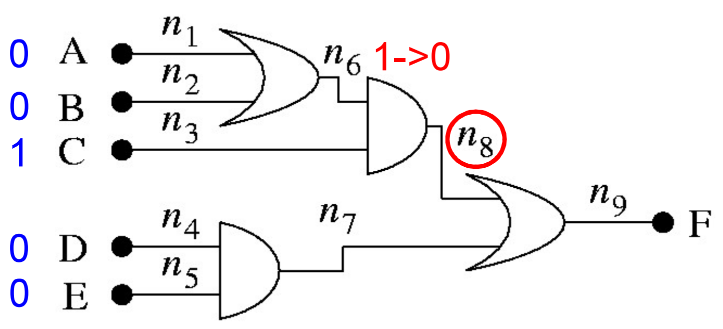
Advance current time by one resolution unit to t=2

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) (delay=3ns)

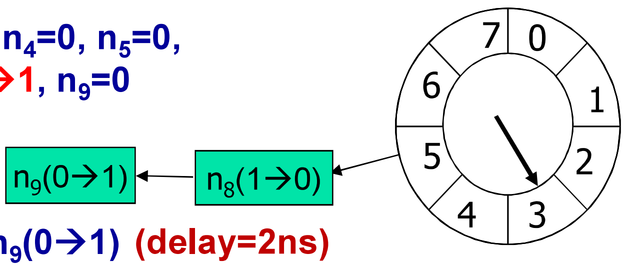


Example (5/8)

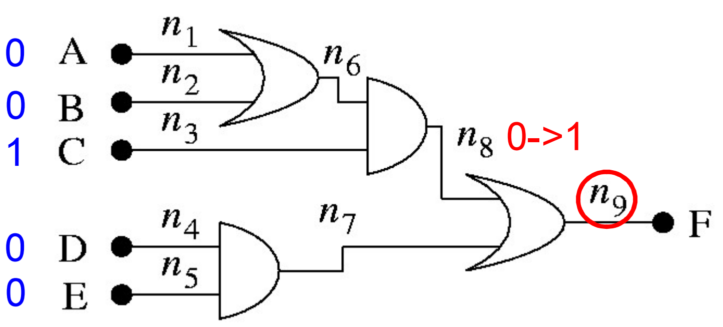
Advance time to t=3

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) (delay=2ns)

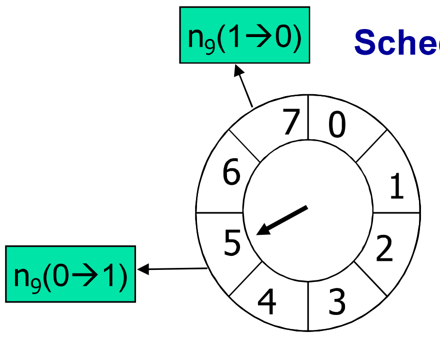


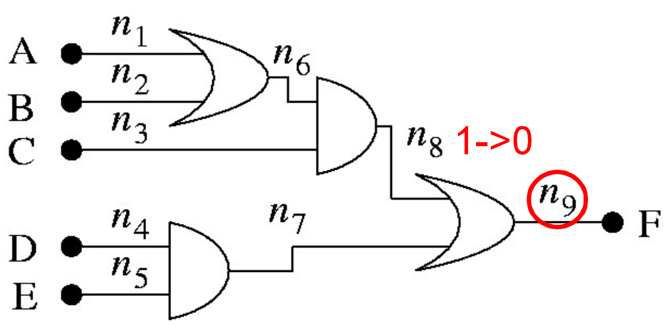
Example (6/8)

Advance time to t=4 and t=5
Process event n_8(1→0) 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

Schedule event n_9(1→0)
(delay=2ns)

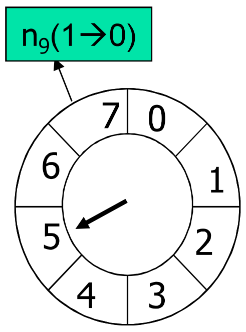


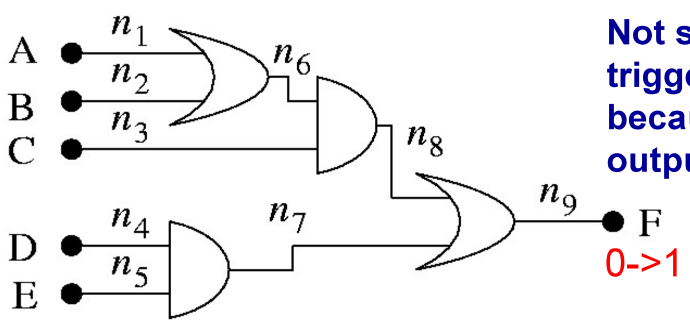


Example (7/8)

Process event n_9(0→1) 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→1



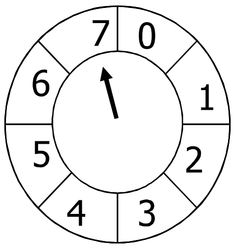


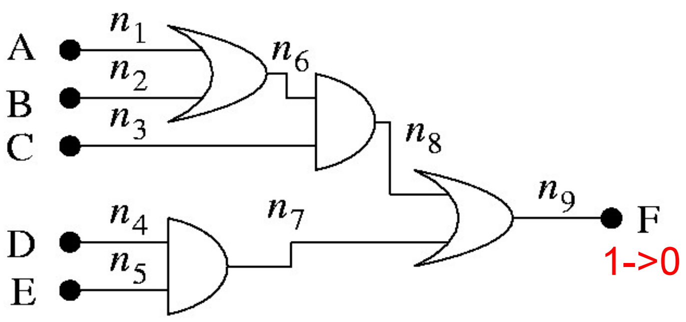
Not scheduling any triggered event because n_9 is an output.

Example (8/8)

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



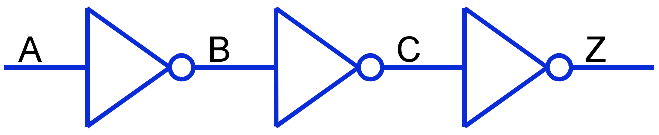


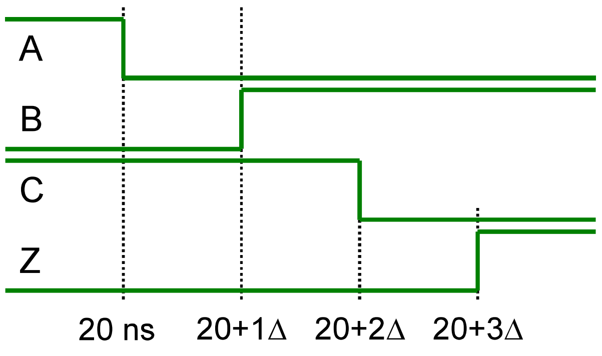
Not scheduling any triggered event because n_9 is an output.

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

Delta Delay

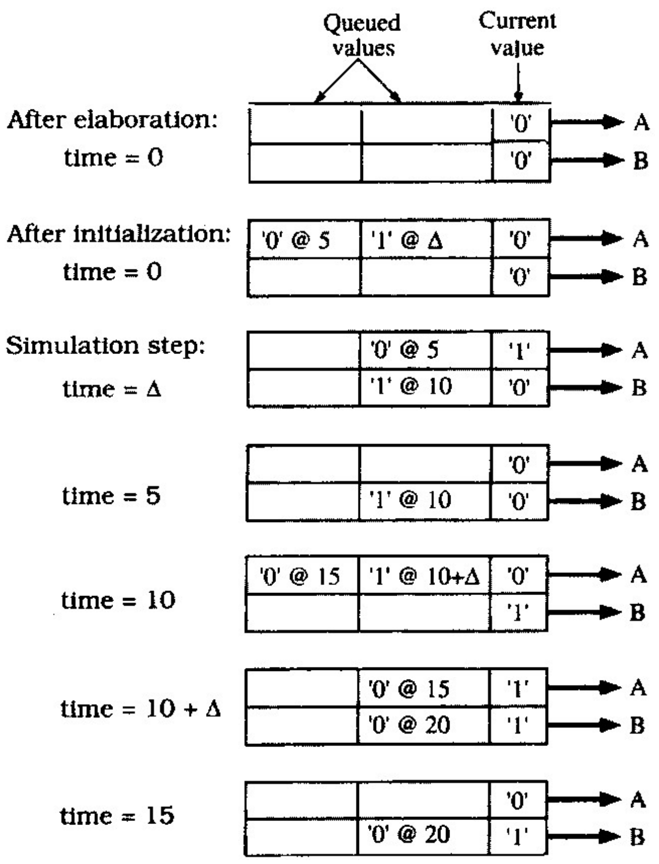
- Infinitesimally small delay
- Models hardware where a minimal amount of time is needed for a change to occur
  - Even in zero delay simulation
- Allows for ordering of events that occur at the same simulation time





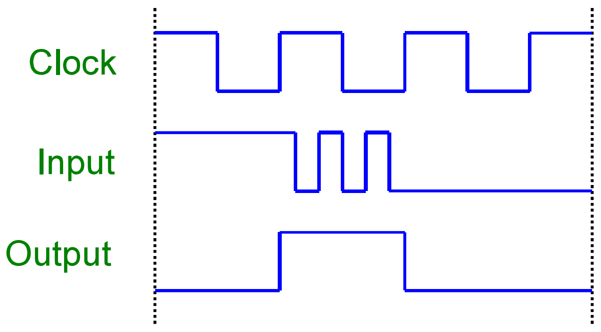
Simulation Example

```
reg A, B;

always @(B)
    begin
        A = 1;
        A = #5 0;
    end

always @(A)
    begin
        if (A==1)
            B = #10 ~B;
    end
```



Cycle-Based Simulation

- Event-Driven:
  - Timing accurate
  - Better debug environment
  - Simulation speed is slower



- Cycle-Based:
  - Perform evaluations just before the triggering clock edge
    - Repeatedly triggered events are evaluated only once in a clock cycle
    - Applicable to synchronous designs only
  - Faster simulation time (5x – 100x)
  - Only cycle-accurate
  - Require other tools (ex: STA) to check timing problems

Discussions

- Simulation is sometimes the only way to verify a design and decide a design's performance
  - Developing effective stimuli is very important
  - Pay attention to the corner cases and longest paths
- There is always a tradeoff between speed and accuracy
  - At transistor level and gate level, we are interested in the delay values of the longest path
    - Accurate but slow
  - At RTL and behavioral level, we only concern with the number of cycles or operations to complete a task
    - Fast but inaccurate (no details)
- Hardware emulation is another effective way to speed up simulation if gate-level design is available
  - Like FPGA