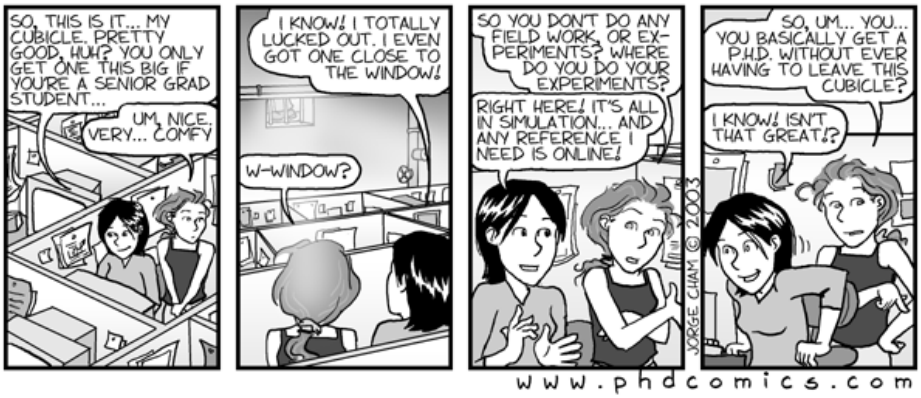
Lecture 10 Chapter 10
Simulation



Main Entry: **sim·u·la·tion**
Pronunciation: "sim-y&-'lA-sh&n
Function: *noun*
Etymology: Middle English *simulacion*, from Middle French, from Latin *simulation-*, *simulatio*, from *simulare*
1 : the act or process of simulating
2 : a sham object : COUNTERFEIT
3 a : the imitative representation of the functioning of one system or process by means of the functioning of another <a computer *simulation* of an industrial process> **b :** examination of a problem often not subject to direct experimentation by means of a simulating device

[Merriam-Webster, www.m-w.com]

Simulation: Definition, Motivation

Simulation: to construct and test a computer model of the circuit to be built.

- **Costs of simulation are far less than the costs of fabricating the circuit directly.**
- **Simulation only models those aspects of the circuit relevant to the level of abstraction concerned.**
- **Avoids problems of physical observation (measuring) to influence the DUT (device under test)**
- **For VLSI circuits simulation is not a guaranteed way of verification**
  - **Impossible to enumerate all combinations of input patterns and internal states.**
  - **However, simulation can increase the belief in the correctness of the design.**
  - **More simulation (hopefully) promotes more belief: huge dedicated simulation compute capacity**

NVIDIA Example (A.D. 2000)

- **~ 850 employees (worldwide total incl. sales, mgmt, …)**
- **~ $85M of CAD tools**
- **~ $20M emulation**
- **Engineering Compute Resources**
  - **Desktops: 200 Sun / 2150 pc's**
  - **Servers:**
    278 Sun / 634 Linux / 496 Gbytes RAM
  - **14 Terabytes of storage**

ET 4255 - Electronic Design Automation 2005 © Nick van der Meijs *24*

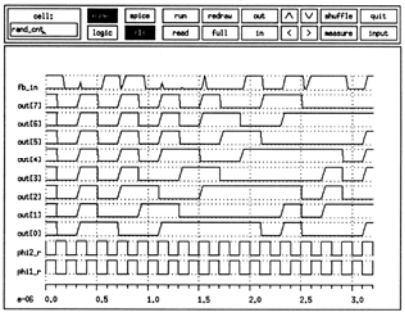
Most compute capacity used for simulation!

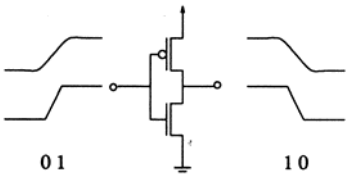
Lecture 1

Simulation

Goal

- Predicting/checking of correct behavior (electrical/functional)
- Checking/determination of performance
- Debugging of circuits





Simulation (2)

Repeat

- Simulation generally proves incorrect behavior only
- Rarely proves correctness of circuit

Simulation is a trade-off

- Accuracy <=> computation time

Depends on

- Phase in design process
- Type of circuit
- Size of circuit
- Preference of designer

Simulation (3)

Additional Reading

Section 1 to 4 of:
VLSI Circuit Analysis, Simulation and Optimization
A.E. Ruehli, D.K. Beece
CompEuro 1986 tutorial

Gate-Level Simulation
M.A. d'Abreu
IEEE Design & Test of Integrated Circuits
December 1985, page 63 - 71

RSIM - A Logic-Level Timing Simulation
C.J. Terman
Proceedings ICCD 1983, page 437 - 440

Simulation Abstractions (models)

Fundamental characteristics: function, signal, time

Function Every simulation level has its own primitives which express the electrical behavior

- Transistor
- Nand-gate
- Processor
- …..

Signal Particular representation of signal

- Logic
- Analog wave form
- Current, Voltage
- …

Signal Strength

- MOS Simulation hinges on implementation aspects outside of the pure Boolean Logic model
  - Bi-directional elements (pass gates)
  - Wired logic
  - Charge sharing
- A signal is represented using value and strength
  - Signal strength is discrete model for signal impedance
  - Signal strength models behavior when signals directly combine.
- Usually: strongest signal wins
  - Instead of voltage division
- Handling of strength depends on simulator type
  - Depends on 'analog capacity' of simulator

Simulation Abstractions (models)

Time Particular representation of time

- Nanoseconds
- Unity step
- Delay less
- …

Simulation Abstraction (models)

	Domain		
Level	**Behavior**	**Structure**	**Geometry**
Architecture	Performance	Processors	Basic Partitions
	Instruction Set	Memory	Macrocells
	Exceptions	Buses	
Register-Transfer	Algorithms	Registers	Floorplan
	Operation	Functional	
	Sequences	Units	
Logic	State transitions	Latches	Cells
	Boolean eq's	Logic gates	
	Truth Tables		
Device	Network equations	Transistor	Exact geometry
	Frequency	Capacitors	
	Response	Resistors	
	V(t), I(t)		

Circuit Level

Function Equivalent circuit of transistor, resistors, capacitors, etc. differential equations

Signal Analog waveform

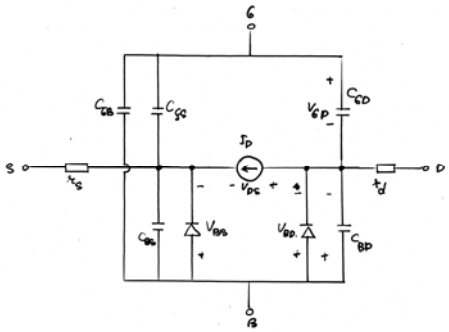
Time Integration-time step

Eg SPICE

- Most detailed
  - analog;
  - nodal / tableau equations;
  - numerical integration;

Related: timing-level simulation:

- analog, but with simplifications (macromodels, look-up tables);
- piecewise-linear methods.

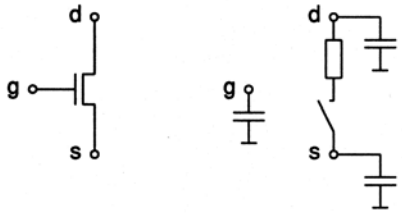


Switch Level

Function transistor as controlled switch, R, C

Signal logic, sometimes analog waveform

Time vary



- transistors are modeled as bidirectional switches;
  - mainly digital;
  - circuits extracted from mask patterns can directly be simulated.

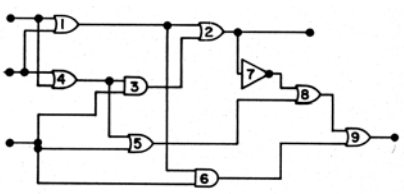
Gate Level for Digital Circuits

Function Logic function of small sub-circuits

Signal Discrete, logic values e.g. {0, 1, x}

Time varying

- ‘‘gate’’ mainly refers to elements to be found in a component library (e.g. for standard-cell design): NAND,
- NOR, MULTIPLEXER, D-FLIPFLOP, LATCH, etc.;
- unidirectional signal flow;
- closely related to ‘‘fault simulation’’.



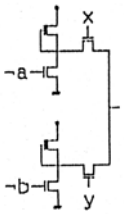
Gate Level (ctd)

Advantages

- Higher simulation rate
- Independent of technology
- Connected with standard cell lib.
- Automatic test vector generation possible

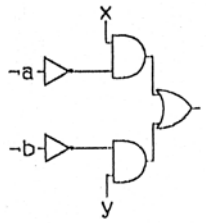
Disadvantages

- May be incompatible with design style
- e.g: pass transistors are bi-directional and gates are uni-directional



x = 1
y = 1

$\bar{b}$	$\bar{a}$	r
0	0	1
0	1	0
1	0	0
1	1	0

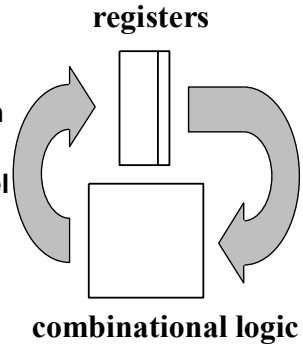


$\bar{b}$	$\bar{a}$	r
0	0	1
0	1	1
1	0	1
1	1	0

Register Transfer level

Function	Registers and transfer functions
Signal	Arithmetic values, bit-vectors
Time	clock-cycles

- **Sequential circuits**, early in design
- circuit is seen as composed of registers to store the state and combinational logic to compute the next state (FSM model).
  - Registers in circuit <=> memory-places in RTL model
  - Signals in circuit <≠> values in RTL model
- Further reduction of simulation time
- Fully independent of technology



Behavioral Level

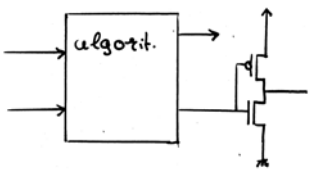
Function	procedures in high-level language describes complex components like alu's, multiplexers, counters
Signal	Arithmetic values, bit-vectors
Time	Clock-cycle, nominal time

```
module mux (out, p0, p1, select);
input  p0, p1;
input  select;
output mux_out;
always @ (select or p0 or p1)
    case (select)
          1'b0 : out = p0 ;
          1'b1 : out = p1 ;
    endcase
endmodule
```

- description in high-level language, e.g. Verilog
- Need not model all registers
- Faster simulation again
- Useful in the first stages of design
- In later stages to model 'surroundings' of module under detailed analysis

Mixed Level and Mixed-Mode.

- Simulation of a circuit with each part at the most effective level
- descriptions at different levels of abstractions coexist within the same simulation environment;
- critical parts of the design are described at a lower level than non-critical parts, while it is inefficient or infeasible to model the whole circuit at the level of the most critical part;
- it might be easier to test a subsystem with stimuli from the system itself, rather than describing the stimuli explicitly;
- Test-bench concept



Hardware-software co-simulation:

- useful in hardware-software codesign;
- becomes more and more important

Components of a Simulator (1)

Simulator Kernel

- the routines for doing the "real" simulation.
- detailed description for event-driven simulation follows.

Routines for Processing of Circuit Description

- input format: either written by the designer or obtained through an interface with a schematic entry tool.
- internal format: machine code or graph-based description.
- input format has to be compiled into internal format.

Components of a Simulator (2)

Routines for Stimuli Processing

- stimuli: the input patterns for all time instants during the simulation.
- they have to provide the kernel with the correct input patterns.

Routines for Output Processing

- the simulator results are numbers; they have to be presented in a user-friendly form, e.g. as tables or waveforms.

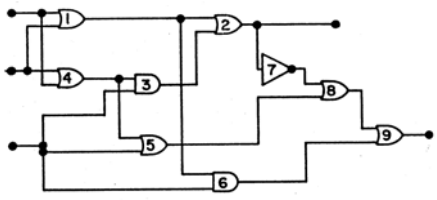
Zoom-in on Some Simulation Types

- Gate-Level (§ 10.2)
- Switch-Level (§ 10.3)

Mainly discuss simulator kernel issues

Gate Level Simulation

Function	Logic function of small sub-circuits (nand, nor, invert)
Signal	Discrete, logic values, strength of signal
Time	varying



Signal Modeling

- Discrete Signal values
- Many different models

IEEE std_logic	
U	Un-initialized
X	forcing unknown
0	forcing 0
1	forcing 1
Z	high impedance
W	weak unknown
L	weak 0
H	weak 1
-	don't care

Minimum set for any simulator

Note the mixture of value and strength

Signal Modeling (2)

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

Similar tables for:

- More inputs
- More logic values
- Other gates

	0	1	X
0	1	1	1
1	1	0	X
X	1	X	x

0 - Logic zero
1 - logic one
X - unknown

Three-value NAND gate

Delay Models for Gate-Level Simulation

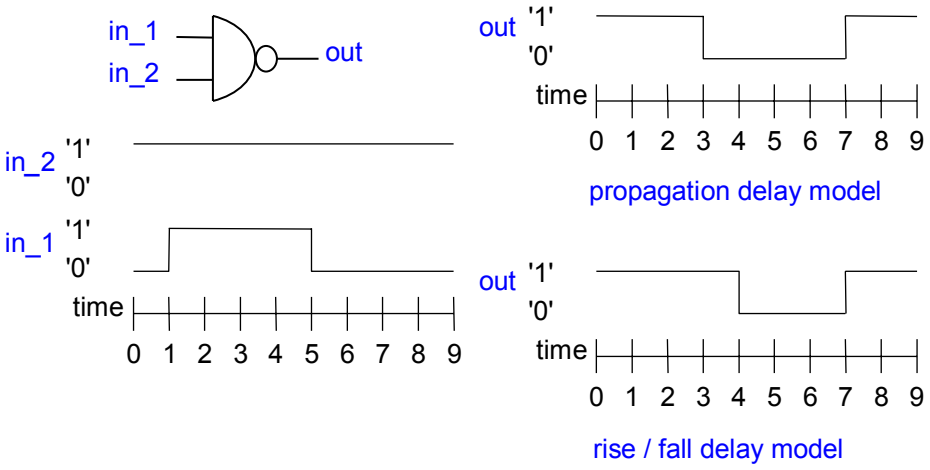
- inertial delay: a change to an input signal has to last at least a certain time before it can trigger any reaction.
- propagation delay: some time passes between the start of a signal change at the gate input and the start of a signal change at its output.
  - rise / fall delay: allow for high to low and low to high assymetry.

Propagation Delay

- AKA transport delay
- Associated with output of gate

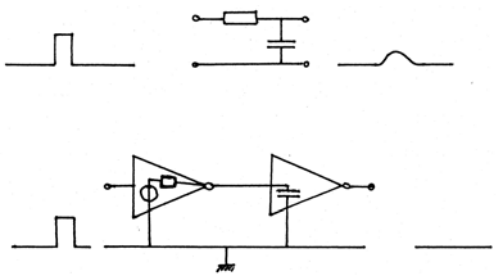
Zero Delay	Specific for type of gate
Unit Delay	Equal delay for all gates
Nominal Delay	Specific for type of gate
Rise/fall Delay	Specific for type of gate, different for rising/falling edge
Ambiguity Delay	Minimum and maximum values for both edges

Delay Model Example



Inertial Delay.

- Associated with input of gate
- Can also include effects of (RC) wiring
- Filter for short pulses

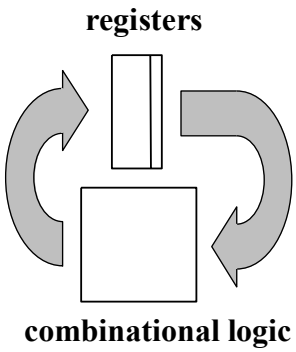


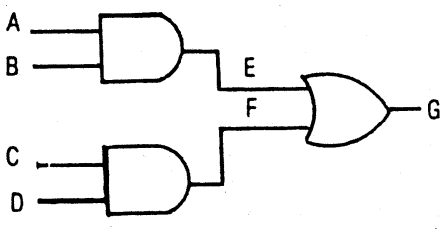
Gate-Level Simulation Kernel

- Compiler-Driven Simulation
  - Execute all instructions in pre-determined sequence
- Event-Driven Simulation
  - Simulator reacts to signals produced

Compiler-Driven Simulation

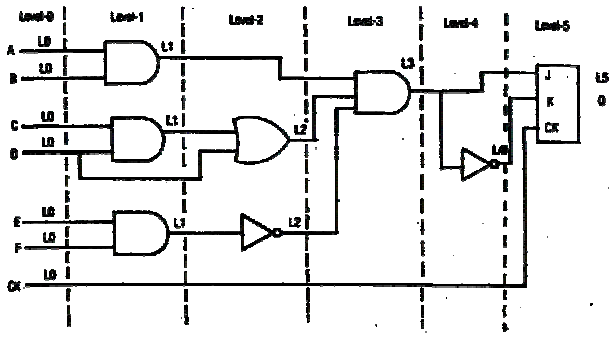
- Based on making an executable-code model of circuit;
- Efficient simulation mechanism (few machine instructions per gate);
- Applicable to few delay models in synchronous circuits (e.g. zero-delay model).





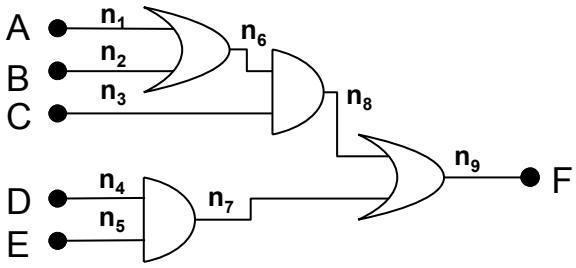

```
LDA A   %   Load accumulator with
        %   value of signal A
AND B   %   AND it with signal B.
STA E   %   store result in E.
LDA C   %   C into accumulator.
AND D   %   AND it with D.
OR  E   %   OR it with E
STA G   %   output of the circuit.
```

Compiler-Driven Simulation (2)



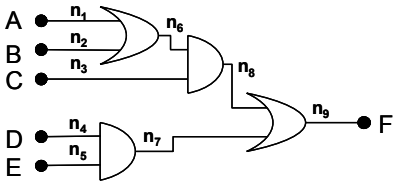
- Leveling to specify evaluation order
- Topological sort ~ longest path algorithm

Zero-Delay Example



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

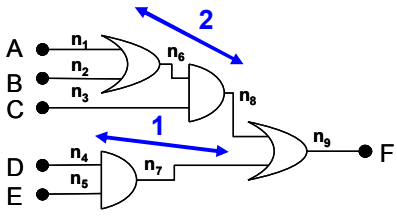
Compiled Code for Unit-Delay Simulation

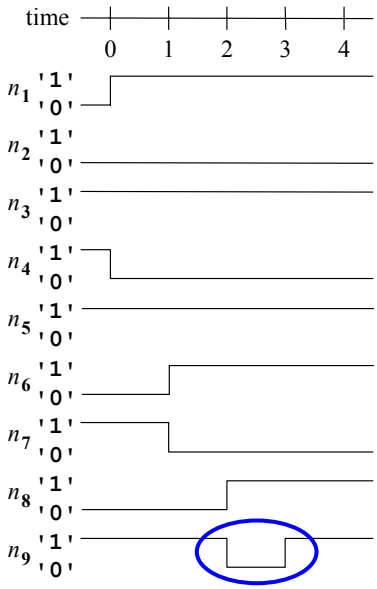



```
for (t ← t_start; t ≤ t_end; t ← t + 1) {
  new[1] ← A;
  new[2] ← B;
  new[3] ← C;
  new[4] ← D;
  new[5] ← E;
  new[6] ← OR(old[1], old[2]);
  new[7] ← AND(old[4], old[5]);
  new[8] ← AND(old[6], old[3]);
  new[9] ← OR(old[7], old[8]);
  F ← new[9];
  old ← new;
}
```


Unit-Delay Simulation

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





Event-Driven Simulation

Latency

- Only a small part of the circuit is active at any time
- Compiler-driven simulation becomes inefficient

Event-Driven Simulation Principles

Simulation	predicting a sequence of state-changes based on a sequence of input states
Event	A gate need only to be evaluated when there is a change in the input

State-transition ⇔ event = (time, net, new value)

Sequence of state-transitions ⇔ event queue

event queue aka event list, time queue
net aka node

Event-Driven Simulation (2)

Event = (time, net, new value)

Event-Driven Simulation Algorithm

```
Insert stimulus events into queue
While event queue not empty:
    fetch event e of queue
    t:= e.time
    for all gates g with input connected to e.net:
        evaluate g with new input e.value
        if output of g changes:
            schedule new event for output of g at t + Δt where
            Δt is the delay associated with the transition
```

Data Structures and Functions for Event-Driven Simulation

```
struct event {
    struct time;
    struct net *node;
    struct signal_value value;
    …
};

struct event_queue {
    …
};
```

Main functions:

- **new_queue: to create a new event queue;**
- **first_event: to remove and return the earliest event in the queue.**
- **insert_event: to add an event to the queue.**
- **(reschedule): if the time of an event changes**

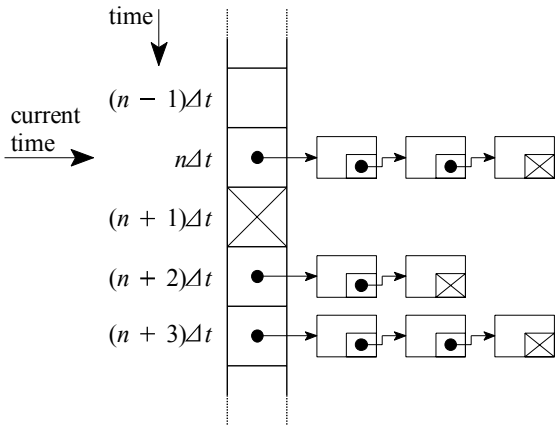
ADT: Priority Queue

Implementation of Event Queue

- **An assumption that is often valid: all gate delays are small integer multiples of minimum-resolution delay Δ_t .**
- **The event queue can then be implemented by an array containing linked lists of simultaneous events (events at $k\Delta_t$ are stored at array index position k).**

Array-Based Event Queue

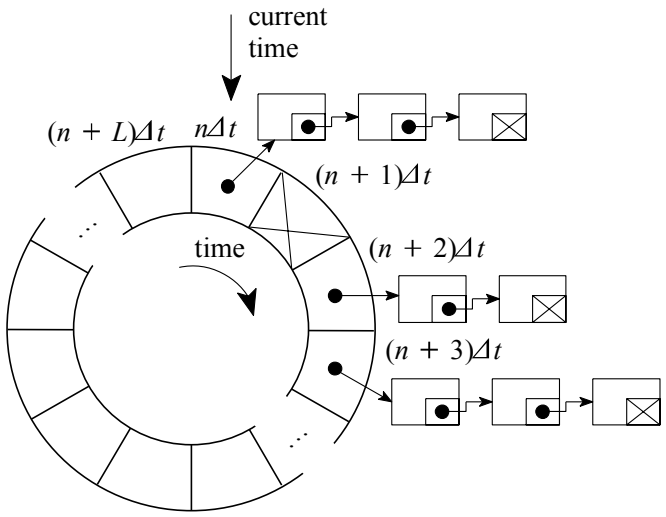
- Often: all gate delays are small integer multiples of minimum-resolution delay Δ_t .
- The event queue → array containing linked lists of simultaneous events (events at $k\Delta_t$ are stored at array index position k).



Time complexity?
Insertion: O(1)
Deletion: O(1)

The Time Wheel

- An indexing modulo L leads to the time wheel data structure.
- Useful if most events occur within time window $L << (t_{end} - t_{start})$



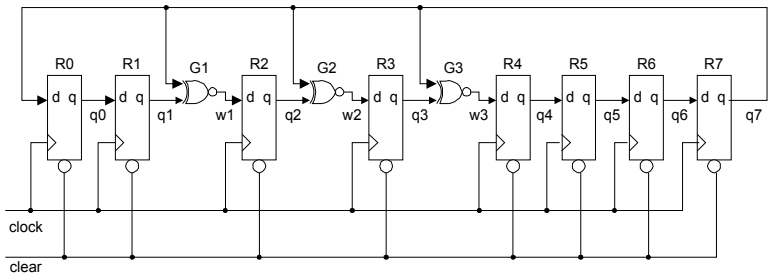
More on Implementation

- Events that take place more than later than $L\Delta_t$ after the current time should be stored in an overflow list.
- If necessary, the overflow list itself can be implemented as a time wheel with a coarser resolution, e.g. $L\Delta_t$ instead of Δ_t .
- If the variance in delays in the system is larger than can be handled by time wheels, a priority queue should be used: adding and removing events will require O(log n) time instead of O(1) time (with n the number of events).

Gate-Level Simulation: Discussion.

- Compiler-driven simulation evaluates many more circuit nets, but does not have the overhead of event-queue manipulation (the overhead can reach a factor of 100).
- Event-driven simulation can handle sophisticated delay models.
- Some simulators use a combination of both methods.
- Yet another method is demand-driven simulation: it processes the circuit backwards from the outputs that the user wants to observe back to the inputs (but it can't deal with circularities).
- Simulation is always too slow → hardware accelerated simulation

Home Brew Simulator




```
xnor2 ("G1","q7","q1","w1");
xnor2 ("G2","q7","q2","w2");
xnor2 ("G3","q7","q3","w3");
dff ("R0","clear","clock","q7","q0");
dff ("R1","clear","clock","q0","q1");
dff ("R2","clear","clock","w1","q2");
dff ("R3","clear","clock","w2","q3");
dff ("R4","clear","clock","w3","q4");
dff ("R5","clear","clock","q4","q5");
dff ("R6","clear","clock","q5","q6");
dff ("R7","clear","clock","q6","q7");
```

Homebrew.zip

See http://www.maxmon.com → free stuff → Homebrew [EDN, July '94]


```
/* Usage: sim stimulus_file response_file */

#include "models.c"                    /* Include the models  */
#include "sim.c"                       /* Include the simulator */
Main (int argc, char *argv[])
{
   char *stimulus = argv[1],           /* stimulus file */
   char *response = argv[2];           /* response file */

   initialize();                       /* initialize */

   xnor2 ("G1","q7","q1","w1");        /* begin circuit description */
   xnor2 ("G2","q7","q2","w2");
   xnor2 ("G3","q7","q3","w3");
   dff ("R0","clear","clock","q7","q0");
   dff ("R1","clear","clock","q0","q1");
   dff ("R2","clear","clock","w1","q2");
   dff ("R3","clear","clock","w2","q3");
   dff ("R4","clear","clock","w3","q4");
   dff ("R5","clear","clock","q4","q5");
   dff ("R6","clear","clock","q5","q6");
   dff ("R7","clear","clock","q6","q7");

   simulate (stimulus, response);      /* go simulate it */
}
```

Normally use 'linker'

Slightly edited homebrew simulator example

AND Model Code

```
void and2(char *name, char *net_in1, char *net_in2, char *net_out)
/*
 * To add an and2 component into the circuit description. Add the
 * component then hookup each net to each pin.
 */
{
   Cmp *cmp;
   cmp = cmp_add(cct, name, 3, and2_simulate);/* add component */
   if(cmp) {
        net_connect(cct, net_in1, cmp, 1);    /* hook pin to net */
        net_connect(cct, net_in2, cmp, 2);
        net_connect(cct, net_out, cmp, 3);
        /* set the driver pin on this component */
        pin_set_driver(cmp, 3);
   }
}
```

Function pointer

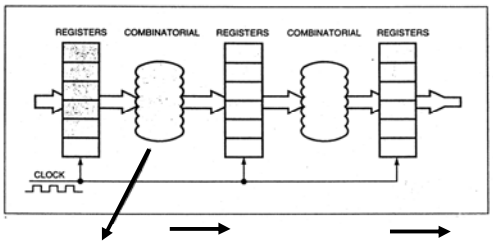
AND Simulation Code.

```
void and2_simulate(Cmp *cmp, int not_used, Event *ev)
{
   int val;
   Pin *pin = PIN_ADDR(cmp, 1);       /* first input pin */
   Net *net;
/*
 * 'AND' each of the input pins to determine output.
 */
   val  = pin->net->value;     /* first input pin */
   pin++;                      /* second input pin    */
   val &= pin->net->value;     /* and2        */
   pin++;                      /* output pin      */
   net = pin->net;             /* net on output pin   */
/*
 *  Schedule an event to appear on output pin. Event will happen at
 *  current time plus one unit.
 */
   event_schedule(net, ev->time+1, val);
}
```

Backup

Cycle Based Simulation

- For synchronous networks



- Convert to Boolean equations
- Not a gate level network
- No timing data! Faster Simulation
- Use static timing analysis instead

Levels of Simulation (1)

From the lowest level to higher levels:

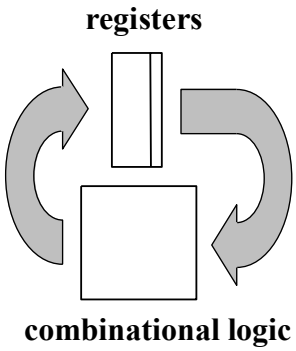
- Device-level simulation:
  - used to test the effect of fabrication parameters;
  - used by technologists, not by circuit or system designers.
- Circuit-level simulation (e.g. SPICE):
  - analog;
  - nodal / tableau equations;
  - numerical integration;
- Timing-level simulation:
  - analog, but with simplifications (macromodels, look-up tables);
  - piecewise-linear methods.

Levels of Simulation (2)

- 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 to be found in a component library (e.g. for standard-cell design): NAND, NOR, MULTIPLEXER, D-FLIPFLOP, LATCH, etc.;
  - unidirectional signal flow;
  - closely related to ‘‘fault simulation’’.

Levels of Simulation (3)

- **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).



- **Behavioral-level simulation:**
  - description in high-level language, e.g. VHDL (VHSIC Hardware Description Language).

Levels of Simulation (4)

- **Mixed-level and mixed-mode simulation:**
  - descriptions at different levels of abstractions coexist within the same simulation environment;
  - critical parts of the design are described at a lower level than noncritical parts, while it is inefficient or infeasible to model the whole circuit at the level of the most critical part;
  - it might be easier to test a subsystem with stimuli from the system itself, rather than describing the stimuli explicitly;
- **Hardware-software cosimulation:**
  - useful in hardware-software codesign;
  - becomes more and more important.

Gate Modeling

3-valued NAND truth table

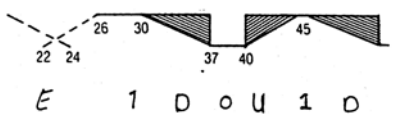
in_1	in_2	out
'0'	'0'	'1'
'0'	'1'	'1'
'0'	'X'	'1'
'1'	'0'	'1'
'1'	'1'	'0'
'1'	'X'	'X'
'X'	'0'	'1'
'X'	'1'	'X'
'X'	'X'	'X'

Signal Values

Depending on Delay Model

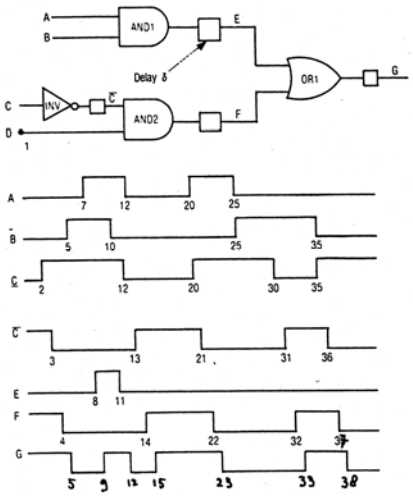
0	Logic zero
1	Logic high
X	Logic unknown
U	Rising
D	Falling
E	Ambiguity

Delay model	Signal Model
Zero delay	0, 1
Unit Delay	0, 1
Rise/fall delay	0, 1, x
Ambiguity delay	0,1 ,U, D, E



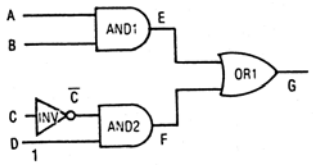
Propagation Delay Model (1)

Nominal/Unit Delay

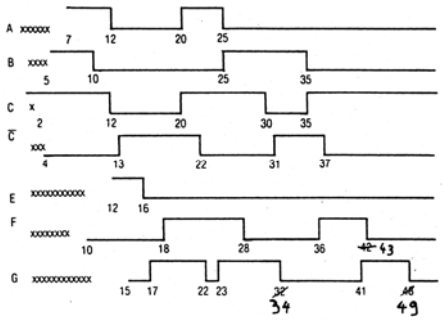


Propagation Delay Model (2)

Rise/Fall Delay

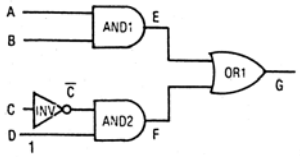


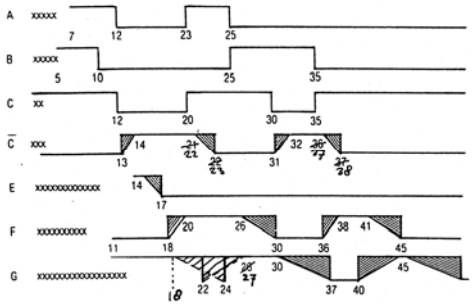
	Rise	Fall	
AND gates	5	6	Units
Inverter	1	2	Units
OR gate	5	6	units



Propagation Delay Model (3)

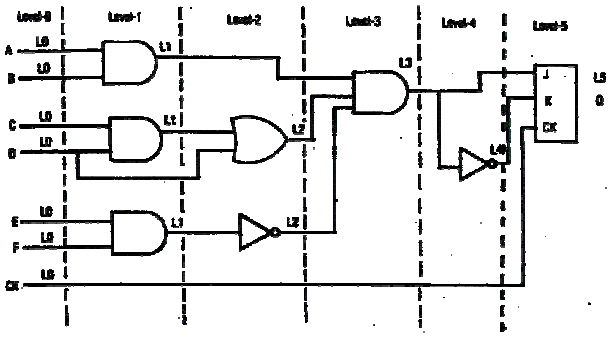
Ambiguity Delay

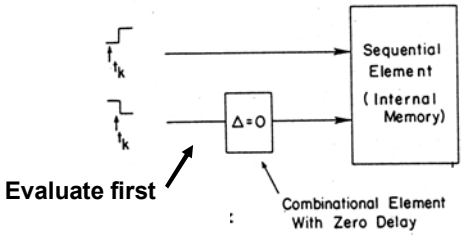




Gate	Delay Min	Max	
AND	5	6	Rise
	4	7	Fall
OR	4	7	Rise
	4	7	Fall
Inverter	1	2	Rise
	2	3	Fall

Compiler-Driven Simulation (2)





Event-Driven Simulation

- **Event-driven simulation is a widely-used mechanism in gate-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 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.**

Event-Driven Simulation Algorithm

```
event_driven_simulation ()
{
  struct event_queue *Q;
  Q ← new_queue();
  "insert stimuli in Q";
  "initialize: all network nodes connected to a memory to 'U' and
               all other nodes to 'X'";
  for (t ← t_start; t < t_end;) {
    current_event ← first_event(Q);
    t ← current_event->time;
    "process current_event and add new events to Q at
     time t + appropriate delay";
  }
}
```

Signal Strength

MOS Simulation hinges on implementation aspects outside of the pure Boolean Logic model

- Bi-directional elements
- Wired logic
- Charge sharing

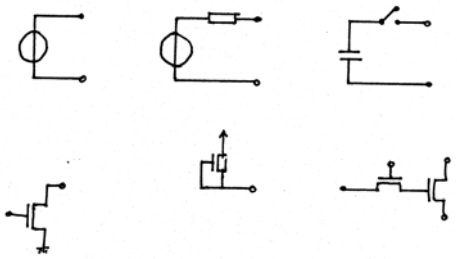
Signal Impedance: also discrete

- For example

Forcing see (a)

Non-forcing see (b)

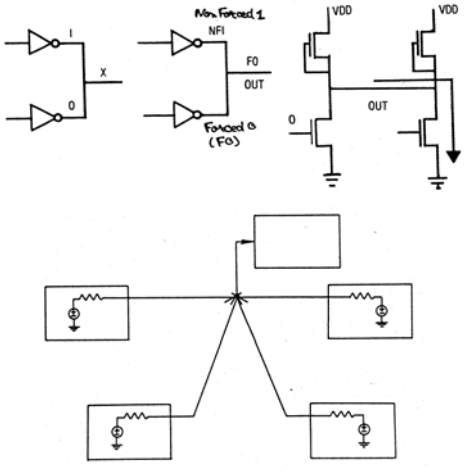
High-Impedance see (c)



(a) (b) (c)

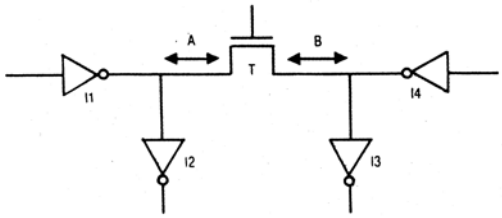
Signal Strength (2)

Strongest signal wins



Signal Strength (3)

Heuristic for bi-directional elements



Signal direction A → B if …

A → B if Strength (I_1) > Strength (I_4)

B → A if Strength (I_1) < Strength (I_4)

Signal Strength (4).

- **Sometimes several value-strength pairs**

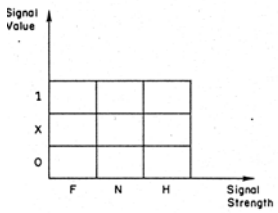


Table 1. Logic States.

State	Comments
Forcing high (low)	Unlimited charge source (sink)
Non-forcing high (low)	Restricted charge source (sink)
High-impecance high (low)	Maximum (minimum) trapped charge
Forcing indeterminate	Short circuit
Non-forcing indeterminate	Unknown charge
High-impedance indeterminate	Unknown trapped charge

Switch-Level Simulation Model (1)

- Circuit model: nets interconnected by transistors.
- A signal is a pair (s, v):
  - strength(s): associated with (possibly discrete) impedance.
  - level(v): associated with voltage.
    Possible values include: '0', '1' and 'X'.

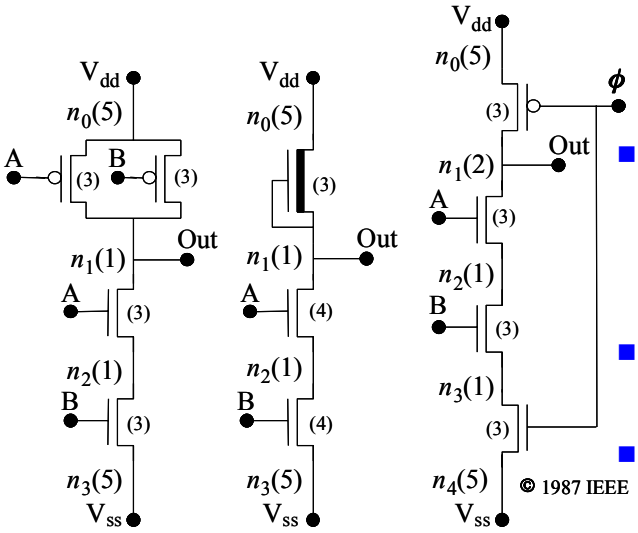
Switch-Level Simulation Model (2)

- There are two types of nets:
  - storage nets: they have a capacitance value; often the set of values is discrete.
  - input nets: they act as sources of fixed value and can supply unlimited current.
- The transistors:
  - act as bidirectional switches;
  - have a strength value (signals passing through a transistor have their strength reduced to this value).

Bryant's Model of Strength Values

- There are w distinct strength values: 1, 2, ... , k, ... , w.
- s = w ⇒ s is the strength of an input signal.
- k < s < w ⇒ s is the strength of a transistor.
- 1 ≤ s ≤ k ⇒ s is the strength of a storage net.

Strength Model Examples



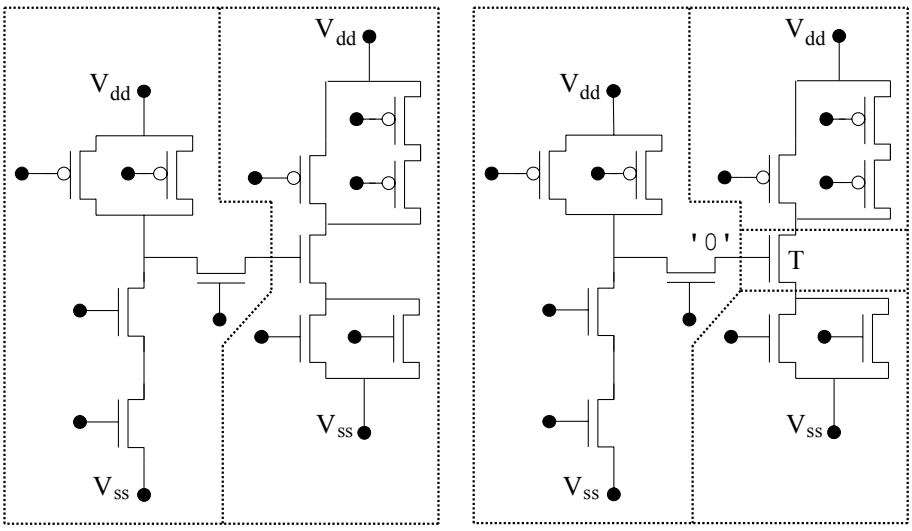
- Key point is choosing strength values consistent with electrical characteristics of circuit
- Transistor limits the strength of its output signal
- Strongest signal wins

Switch-Level Simulation Techniques

Main principles:

- Partition the circuit into unidirectional subcircuits
  - Channel-connected components are bi-directional
  - Gates bound uni-directional elements
  - Off-state transistors (and input nets) can also bound uni-directional elements
  - Interaction between these subcircuits can be handled similar to gate-level simulation.
- Two types of partitioning exist: static and dynamic (= accounting for signal values).
- Apply special methods to compute the ‘‘steady-state’’ of the channel-connected components.

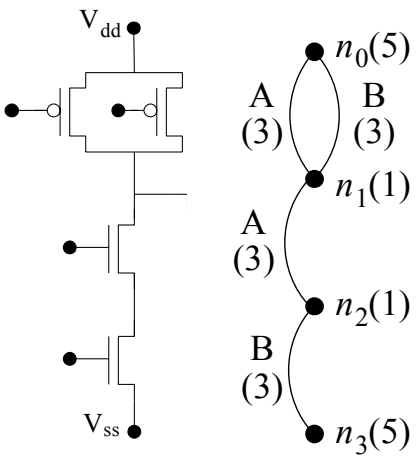
Circuit Partitioning Example



Static partitioning Dynamic partitioning

Circuit Representation

- A convenient representation for switch-level circuits is a multigraph G(V, E) 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 (1)

- A signal on a vertex $u \in V$ is denoted by (σ_u , λ_u).
- 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 $v \in V$.

$$\sigma_{u \to v} = \min(\sigma_u , \varepsilon_{u,v})$$

- 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 (2)

- Suppose that a driven net $v \in V$ has edges $(u_1,v), \ldots, (u_m,v) \in V$, then:

$$\sigma_v = \max_{1 \le i \le m} \sigma_{u_i \to v}$$ Strength of signal on net v

- For a charged net, the net's own signal should be taken into account:

$$\sigma_v = \max (\sigma_v, \max_{1 \le i \le m} \sigma_{u_i \to v})$$

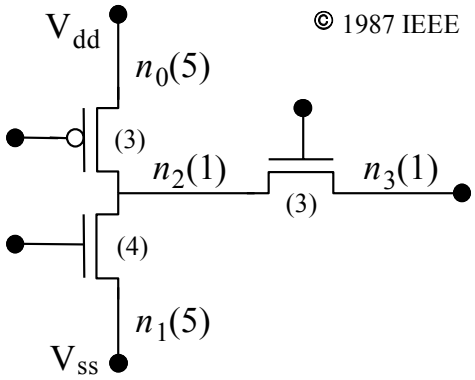
- When combining signals from different directions, the level of the new signal equals the level of the strongest signals. In case of multiple signals with equal strength and different levels, the new level becomes 'X'.

Simulation Algorithm Principles

- The algorithm is based on a repeated application of:

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

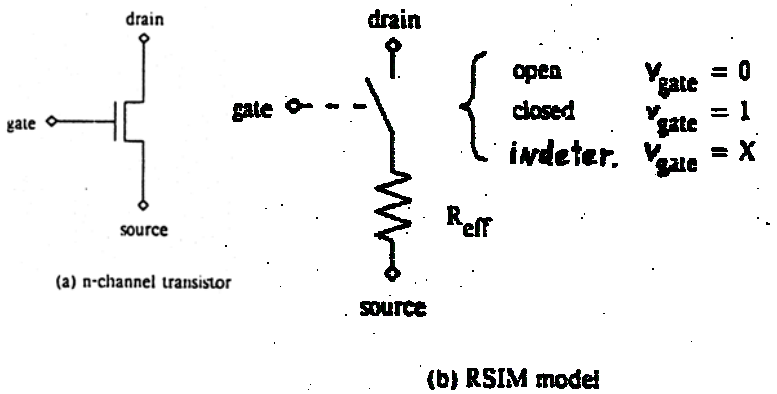
- This should be done carefully: propagate the strongest signals first.
- Implement with an array of queues, one array position for each strength value.



Simulation Algorithm: Discussion

- The algorithm operates in linear time with respect to the number of nets and transistors.
- The algorithm is static: changes to input signals require repeating the complete propagation. It can, however, be modified for dynamic simulation.
- This algorithm does not incorporate any type of delays related to the physical implementation. Switch-level timing simulation can deal with actual R and C values derived from the layout.

RSIM Linear Switch Level Model



V_{gate}	R_{ds} N	R_{ds} P
0	∞	R_{eff}
1	R_{eff}	∞
X	$[R_{eff} - \infty]$	$[R_{eff} - \infty]$

Interval Arithmetic for X-states

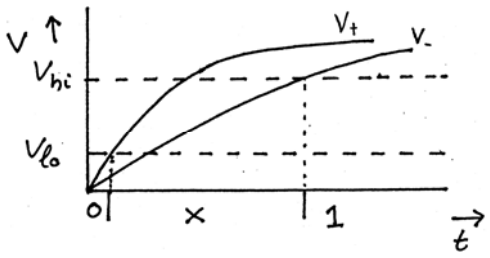
Linear Switch Level Model

R_{eff} depends on:

- Transistor dimension
- Transistor type
- Transistor context
  pull-down, pass, …

$$R_{eff} = f(type, context) \times \frac{length}{width}$$

Logic Level

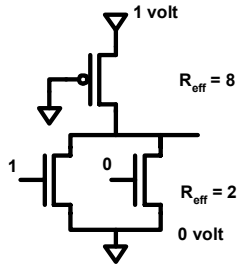


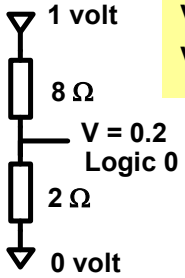
$$V = [V_-, V_+]$$

$\Leftrightarrow$

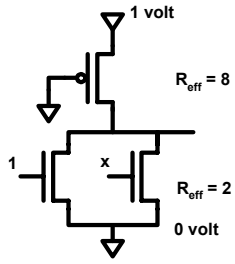
Logic States	
0	$V_+ \leq V_{lo}$
1	$V_- \geq V_{hi}$
X	otherwise

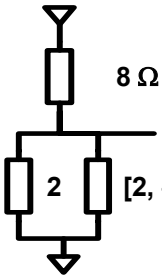
Determining Node Potentials





V_{lo} = 0.25
V_{hi} = 0.75

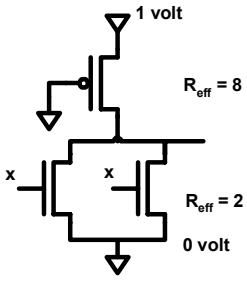


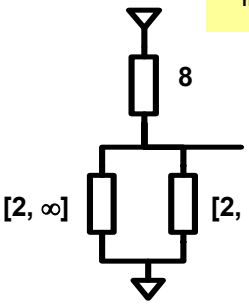


V = [0.11, 0.2]
Logic 0

Determining Node Potentials

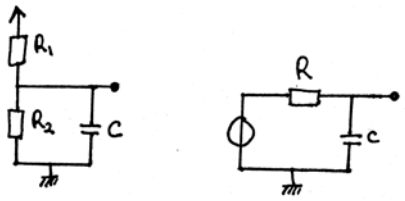
V_{lo} = 0.25
V_{hi} = 0.75





V = [0.11, 1]
Logic x

Determining Transition Time



Calibration

R_{static}	Final node potential
R_{dynlow}	Used for RC time high → low
$R_{dynhigh}$	Used for RC time low → high

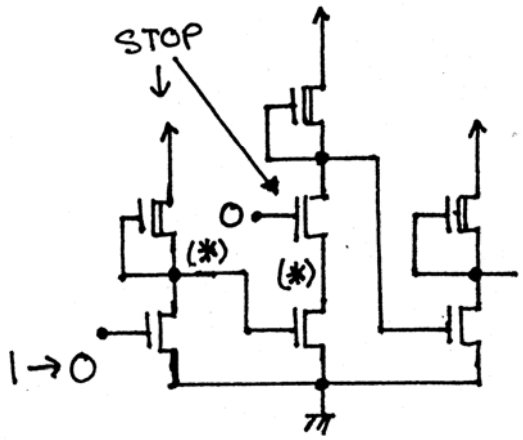
RSIM Simulation Algorithm

Event Driven

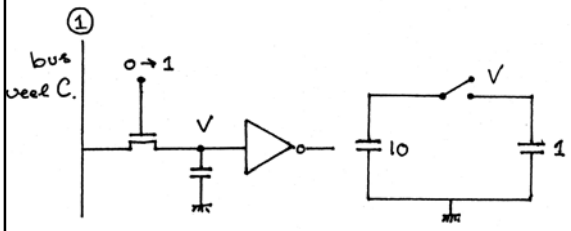
- Event = (net, new logic value, time)
- Algorithm: While event list ≠ ∅
  1. take event from list
  2. set node on new value
  3. determine effect on other nodes
     - Limited number of nodes (stage, vicinity)
     - Determine charge-sharing (immediately)
     - Determine final value (RC time)

RSIM Simulation Algorithm

- (*) = nodes which are influenced



Charge Sharing



Logic States	
0	$V_+ \leq V_{lo}$
1	$V- \geq V_{hi}$
X	otherwise

$$V_- = \frac{\textit{total capacitance on logic level 1}}{\textit{total capacitance}}$$

$$V_+ = \frac{\textit{total capacitance on logic level 1 or x}}{\textit{total capacitance}}$$

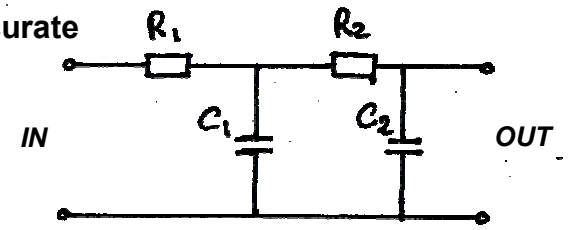
Voltage on V	
Before switched on	After switched on
1	1
0	10/11
X	[10/11, 1]

RC Time

R: R_{dynhi} or R_{dynlo}

C: total cap. on logic level 0 or x if final state 1
total cap. on logic level 1 or x if final state 0

RC time inaccurate



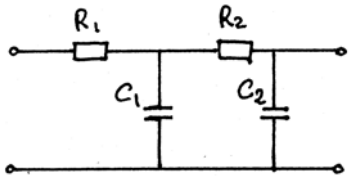
Better:

- $R_1C_1 + (R_1 + R_2)C_2$ (Elmore Delay)

Switch Level Simulation Conclusion

RSIM

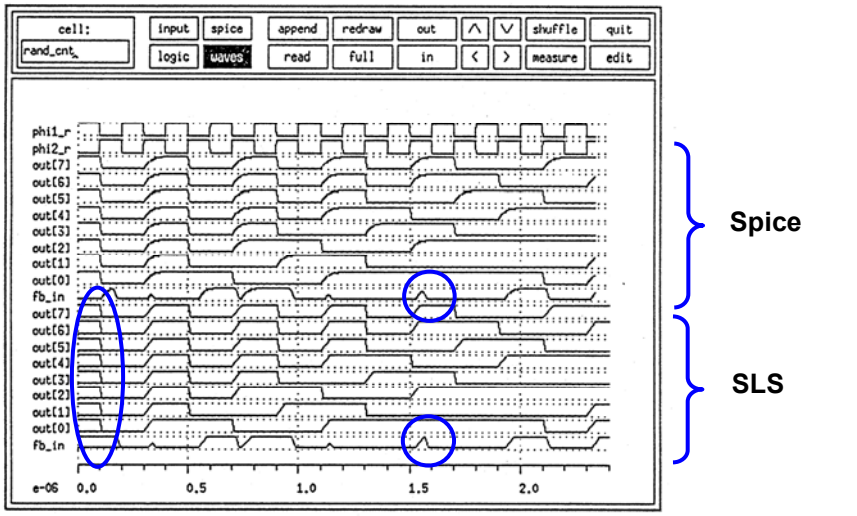
- Linear switch level (switch level timing)
- Interval arithmetic
- Conversion to logic states
- Lumped (concentrated) capacitances



SLS

- Linear switch level
- Interval arithmetic (consistent)
- Conversion to logic states only for transistor state
- Elmore delay

Switch Level Timing (SLS) vs SPICE



SPICE: 39 min 41 sec SLS: 2.4 sec (HP9000840, ~anno 1990)

Simulation Conclusion

Gate level	**huge circuits, not appropriate for MOS**
Discrete switch level	**appropriate for MOS, no timing information**
Linear switch level	**appropriate for MOS, timing estimation**
Timing simulation	**accurate timing, circuit size limited**
Circuit simulation	**most accurate, "small" circuits**
Cycle based simulation	**fast, no timing, synchronic circuits**