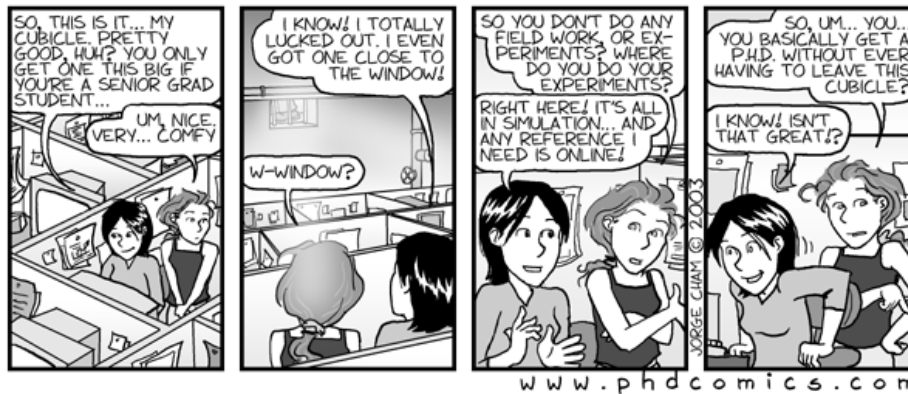


Lecture 10 Chapter 10


Simulation



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

1

Main Entry: *sim-ulation* 

Pronunciation: "sim-y&- 'l&-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]

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

2

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



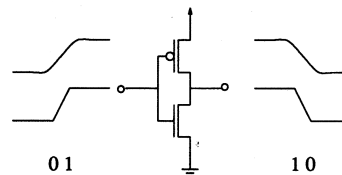
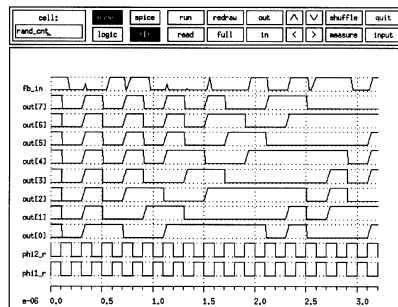
Most compute capacity used for simulation!

Lecture 1

Simulation

Goal

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



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

5

Simulation (2)

Repeat

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

Simulation is a trade-off

- Accuracy \Leftrightarrow computation time

Depends on

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

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

6

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)

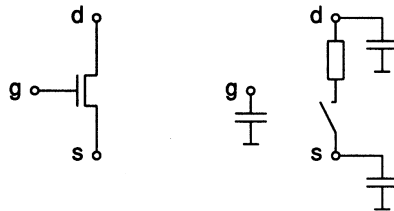
- Time** Particular representation of time
- Nanoseconds
 - Unity step
 - Delay less
 - ...

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)		

Switch Level

Function	transistor as controlled switch, R, C
Signal	logic, sometimes analog waveform
Time	vary

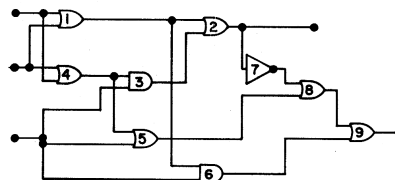


- transistors are modeled as bidirectional switches;
 - mainly digital;
 - circuits extracted from mask patterns can directly be simulated.

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”.



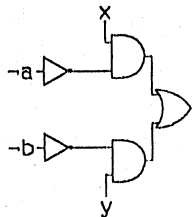
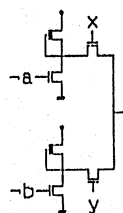
Gate Level (ctd)

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

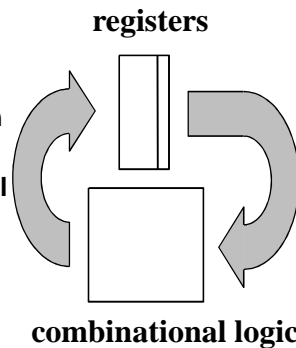
\bar{b}	\bar{a}	r
0	0	1
0	1	0
1	0	0
1	1	0

\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 \Leftrightarrow memory-places in RTL model
 - Signals in circuit \nleftrightarrow 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

VHDL Counter

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all; -- for the unsigned type

entity counter_example is
  generic ( WIDTH : integer := 32);
  port (
    CLK, RESET, LOAD : in std_logic;
    DATA : in unsigned(WIDTH-1 downto 0);
    Q : out unsigned(WIDTH-1 downto 0));
end entity counter_example;

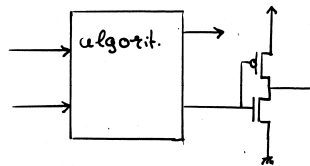
architecture counter_example_a of counter_example is
  signal cnt : unsigned(WIDTH-1 downto 0);
begin
  process(RESET, CLK)
  begin
    if RESET = '1' then
      cnt <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD = '1' then
        cnt <= DATA;
      else
        cnt <= cnt + 1;
      end if;
    end if;
  end process;

  Q <= cnt;
end architecture counter_example_a;

```

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 other 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.

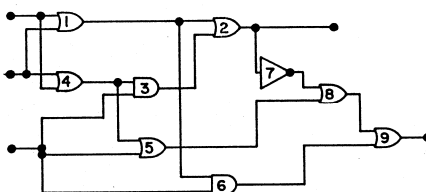
Zoom-in on Some Simulation Types

- 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.

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

Similar tables for:

- More inputs
- More logic values
- Other gates

0 - Logic zero
1 - logic one
X - unknown

Three-value NAND gate

Delay Models for Gate-Level Simulation

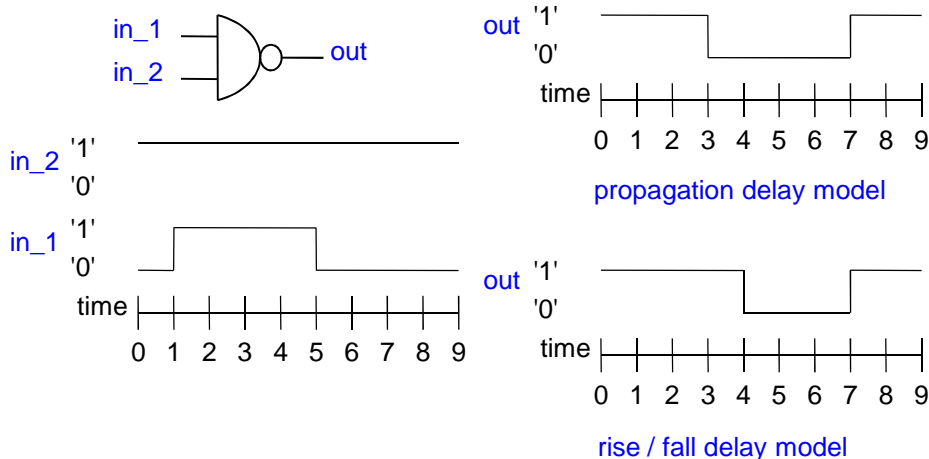
- **inertial delay**: a change to an input signal has to last at least a certain time before it can trigger any reaction.
- **propagation delay**: some time passes between the start of a signal change at the gate input and the start of a signal change at its output.
 - **rise / fall delay**: allow for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

Propagation Delay

- AKA transport delay
- Associated with **output** of gate

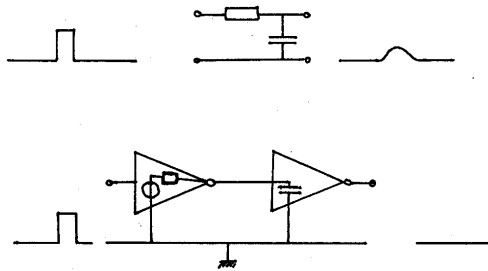
Zero Delay	Specific for type of gate
Unit Delay	Equal delay for all gates
Nominal Delay	Specific for type of gate
Rise/fall Delay	Specific for type of gate, different for rising/falling edge
Ambiguity Delay	Minimum and maximum values for both edges

Delay Model Example



Inertial Delay.

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



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

29

Gate-Level Simulation Kernel

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

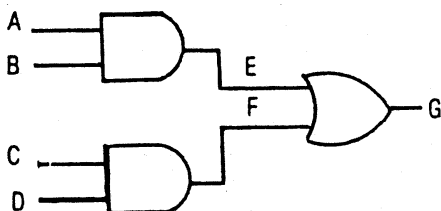
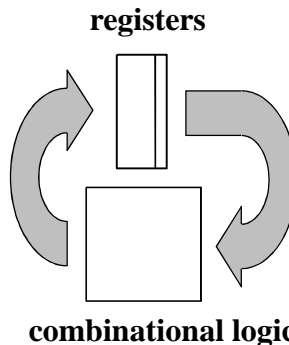
ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

30

Compiler-Driven Simulation

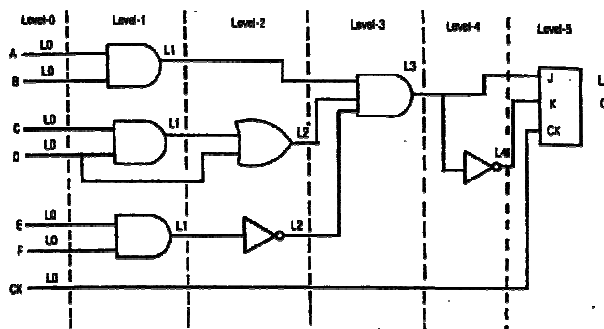
- Based on making an executable-code model of circuit;
- Efficient simulation mechanism (few machine instructions per gate);
- Applicable to few delay models in synchronous circuits (e.g. zero-delay model).



```

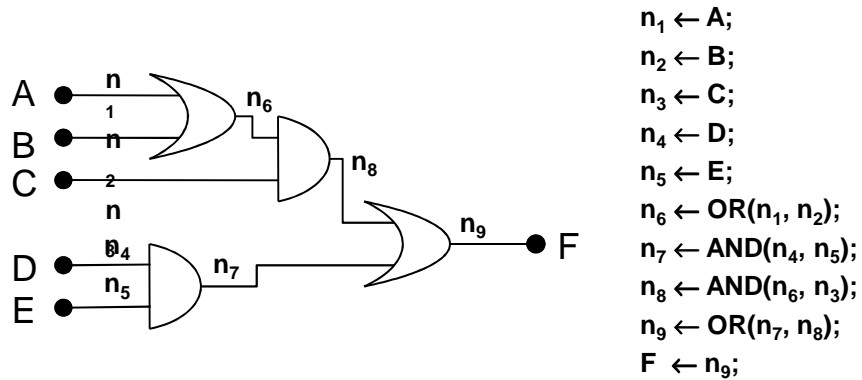
LDA A  % Load accumulator with
        % value of signal A
AND B  % AND it with signal B.
STA E  % store result in E.
LDA C  % C into accumulator.
AND D  % AND it with D.
OR E   % OR it with E
STA G  % output of the circuit.
    
```

Compiler-Driven Simulation (2)



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

Zero-Delay Example

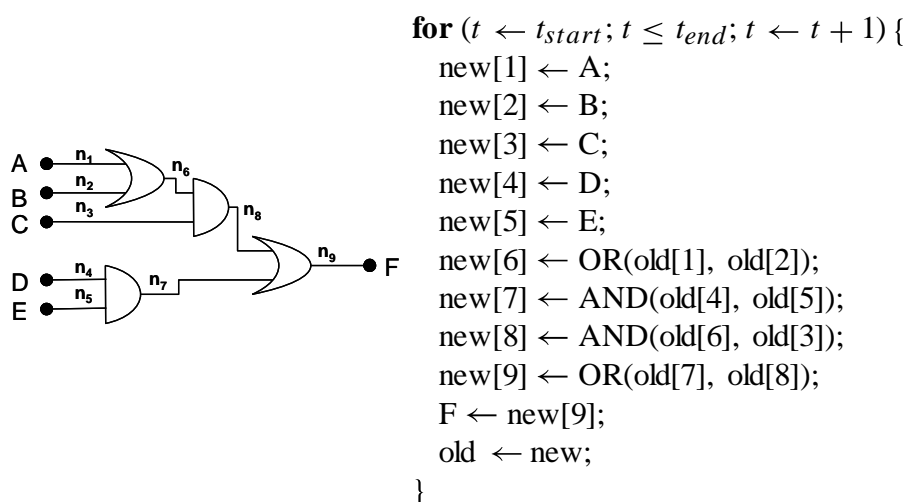


ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

33

Compiled Code for Unit-Delay Simulation



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

34

Unit-Delay Simulation

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

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs
10/7/2010
35

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

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs
10/7/2010
36

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

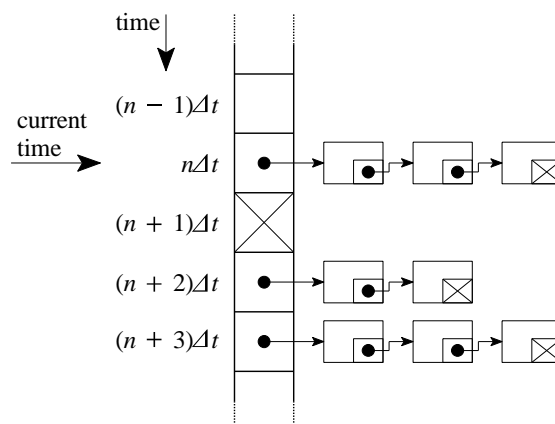
ADT: Priority Queue

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 \rightarrow array containing linked lists of simultaneous events (events at $k\Delta_t$ are stored at array index position k).



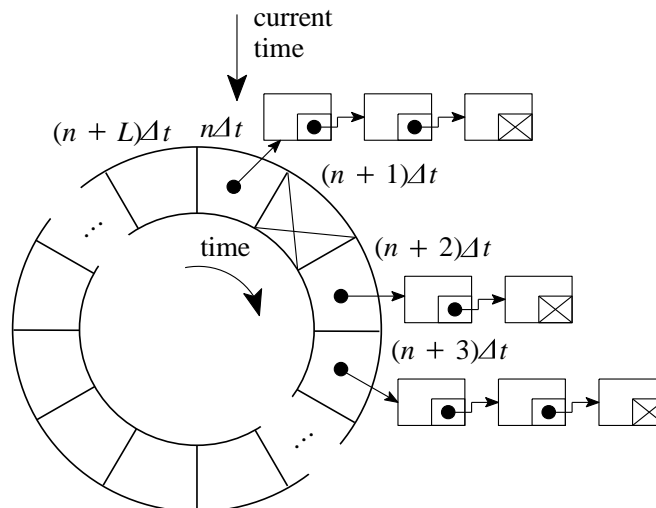
Time complexity?

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 $L \ll (t_{\text{end}} - t_{\text{start}})$



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

41

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).

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

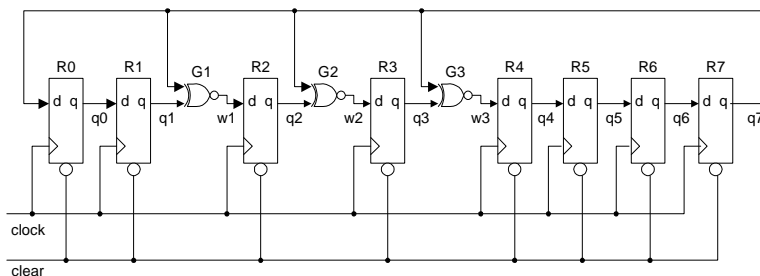
10/7/2010

42

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.maxinon.com> → free stuff → Homebrew [EDN, July '94]

```

/* Usage: sim stimulus_file response_file */
#include "models.c"
#include "sim.c"
Main (int argc, char *argv[])
{
    char *stimulus = argv[1],
    char *response = argv[2];

    initialize();

    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");

    simulate (stimulus, response);
}

```

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);
}

```

Gate Level Simulation Conclusion

- Gate Level Simulation is main stream
- Highly developed state of the art

- Disadvantage is the need for input vectors (stimuli)
- Static Timing Analysis (STA) attempts at predicting timing w/o input vectors
- When timing is important, not correct operation (can be checked by more high level simulators)
 - Find critical paths (longest path, similar to compaction)
 - Estimate arrival times (of signals at latches/flip flops)
 - Estimate required times
 - Estimate slack (difference of required and arrival times)
 - Potential problems include 'false path problem'.