Introduction

- Motivation
- Types of logic simulation
  - Compiled code
  - Event-driven
- Delay models
- Element evaluation
- Hazard detection
Motivation

• A design verification technique (functional and timing)
• Compare results obtained with expected responses specified by the specification
• Use software model

Stimuli and control \[\rightarrow\] Simulation program \[\rightarrow\] Results
  \[\downarrow\] Internal model

Motivation

• Correctness, independent of initial (power-on) state
• Insensitive to small variations in component delays
• Free of races, oscillations, “illegal” input combinations, “unsafe” states
• Evaluation of design alternatives (what-if scenarios)
• Documentation (generation of timing diagrams)
Logic Simulation

- What is simulation?
- Design verification
- Circuit modeling
- True-value simulation algorithms
  - Compiled-code simulation
  - Event-driven simulation
- Summary

Simulation Defined

- Definition: Simulation refers to modeling of a design, its function and performance.
- A software simulator is a computer program; an emulator is a hardware simulator.
- Simulation is used for design verification:
  - Validate assumptions
  - Verify logic
  - Verify performance (timing)
- Types of simulation:
  - Logic or switch level
  - Timing
  - Circuit
  - Fault
**Simulation for Verification**

- **Specification**
- **Design (netlist)**
  - **Design changes**
  - **True-value simulation**
  - **Input stimuli**
  - **Response analysis**
  - **Computed responses**

**Synthesis**

---

**Modeling for Simulation**

- Modules, blocks or components described by
  - Input/output (I/O) function
  - Delays associated with I/O signals
  - Examples: binary adder, Boolean gates, FET, resistors and capacitors
- Interconnects represent
  - ideal signal carriers, or
  - ideal electrical conductors
- Netlist: a format (or language) that describes a design as an interconnection of modules. Netlist may use hierarchy.
Example: A Full-Adder

HA; inputs: a, b; outputs: c, f;
AND: A1, (a, b), (c);
AND: A2, (d, e), (f);
OR: O1, (a, b), (d);
NOT: N1, (c), (e);

FA; inputs: A, B, C; outputs: Carry, Sum;
HA: HA1, (A, B), (D, E);
HA: HA2, (E, C), (F, Sum);
OR: O2, (D, F), (Carry);

Logic Model of MOS Circuit

pMOS FETs V_{DD}

a

\[ C_a \]

b

\[ C_b \]

C_a, C_b and C_c are parasitic capacitances

D_a and D_b are interconnect or propagation delays

D_c is inertial delay of gate
Options for Inertial Delay
(simulation of a NAND gate)

Inputs
- a
- b
- c (CMOS)
- c (zero delay)
- c (unit delay)
- c (multiple delay)
- c (minmax delay)

Logic simulation

Time units

Transient region

Inputs
- a
- b
- c (CMOS)
- c (zero delay)
- c (unit delay)
- c (multiple delay)
- c (minmax delay)

Logic simulation

Time units

Transient region

Signal States

- Two-states (0, 1) can be used for purely combinational logic with zero-delay.
- Three-states (0, 1, X) are essential for timing hazards and for sequential logic initialization.
- Four-states (0, 1, X, Z) are essential for MOS devices. See example below.
- Analog signals are used for exact timing of digital logic and for analog circuits.
### Modeling Levels

<table>
<thead>
<tr>
<th>Modeling level</th>
<th>Circuit description</th>
<th>Signal values</th>
<th>Timing</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function, behavior, RTL</td>
<td>Programming language-like HDL</td>
<td>0, 1</td>
<td>Clock boundary</td>
<td>Architectural and functional verification</td>
</tr>
<tr>
<td>Logic</td>
<td>Connectivity of Boolean gates, flip-flops and transistors</td>
<td>0, 1, X and Z</td>
<td>Zero-delay unit-delay, multiple-delay</td>
<td>Logic verification and test</td>
</tr>
<tr>
<td>Switch</td>
<td>Transistor size and connectivity, node capacitances</td>
<td>0, 1 and X</td>
<td>Zero-delay</td>
<td>Logic verification</td>
</tr>
<tr>
<td>Timing</td>
<td>Transistor technology data, connectivity, node capacitances</td>
<td>Analog voltage</td>
<td>Fine-grain timing</td>
<td>Timing verification</td>
</tr>
<tr>
<td>Circuit</td>
<td>Tech. Data, active/passive component connectivity</td>
<td>Analog voltage, current</td>
<td>Continuous time</td>
<td>Digital timing and analog circuit verification</td>
</tr>
</tbody>
</table>

### True-Value Simulation Algorithms

- Compiled-code simulation (oblivious simulation)
  - Applicable to zero-delay combinational logic
  - Also used for cycle-accurate synchronous sequential circuits for logic verification
  - Efficient for highly active circuits, but inefficient for low-activity circuits
  - High-level (e.g., C language) models can be used
- Event-driven simulation (exclusive simulation of activity)
  - Only gates or modules with input events are evaluated (*event means a signal change*)
  - Delays can be accurately simulated for timing verification
  - Efficient for low-activity circuits
  - Can be extended for fault simulation
Types of Simulation (Contd.)

- Compiled-code (oblivious)
  - The circuit is described in a programming language and an executable model is generated
  - Circuit operation = program execution
  - Fast and efficient but inflexible; practical only for small circuits

- Event-driven
  - Exclusive simulation of activity
  - Circuit is a data structure, simulation program is same for all circuits
  - Flexible, but requires event list management (overhead)

Compiled-Code Algorithm

- Step 1: Levelize combinational logic and encode in a compilable programming language
- Step 2: Initialize internal state variables (flip-flops)
- Step 3: For each input vector
  - Set primary input variables
  - Repeat (until steady-state or max. iterations)
    - Execute compiled code
  - Report or save computed variables
Compiled-Code Simulation

Simulation program

* Delays can be modeled by explicitly adding them to the software model

Event-Driven Simulation

Advance simulation time
Determine current events
Update values
Propagate events
Evaluate activated elements
Schedule resulting events

No more events

Done
Event-Driven Simulation

Snapshot of event list

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>F = 1</td>
<td>t = 2</td>
</tr>
<tr>
<td>G = 1</td>
<td>t = 4</td>
</tr>
<tr>
<td>Z = 1</td>
<td>t = 4</td>
</tr>
<tr>
<td>Y = 1</td>
<td>t = 7</td>
</tr>
</tbody>
</table>

Event-Driven Algorithm
(Example)

Scheduled events
Activity list

t = 0
<table>
<thead>
<tr>
<th>c = 0</th>
<th>d, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>g</td>
</tr>
</tbody>
</table>

Time stack
Time Wheel (Circular Stack)

Current time pointer: $t=0$

Event link-list:

Efficiency of Event-driven Simulator

- Simulates events (value changes) only
- Speed up over compiled-code can be ten times or more; in large logic circuits about 0.1 to 10% gates become active for an input change
Delay Models

- Transport delay (pure delay)
  - Single delay value for each gate
- Rise/fall times
  - Two delay values for each gate output
- Ambiguity delay
  - Min and max delay values for each gate
  - Ambiguity region becomes enlarged towards primary outputs
- Inertial delay
  - Filter out narrow pulses (spikes and glitches)

Element Evaluation

- Simulation speed depends on fast elements (gates) that can be evaluated
- Truth tables
  - O(1) evaluation but O(2^N) storage
  - Decision step needed (which truth table?)
- Zoom table of size tS combines t individual truth tables, S is size of largest truth table
Input Scanning

\[
\text{evaluate (G,c,i)} \\
\text{begin} \\
u_{\text{values}} = \text{FALSE}; \\
\text{for every input value of G} \\
\text{begin} \\
\quad \text{if } v = \text{c then return } c \oplus i \\
\quad \text{if } v = \text{u then } u_{\text{values}} = \text{TRUE} \\
\text{end} \\
\text{if } u_{\text{values}} \text{ return } u \\
\text{return } c \oplus i \\
\text{end} \\
\text{O(1) storage, O(N) evaluation}
\]

- Controlling value c
- Inversion value i

\[
\begin{array}{c|c|c}
\text{c} & \text{X} & \text{c} \oplus i \\
\hline
\text{X} & \text{c} & \text{X} \oplus i \\
\hline
\text{X} & \text{X} & \text{c} \oplus i \\
\hline
\overline{c} & \overline{c} & \overline{c} \oplus i \\
\end{array}
\]

Input Counting

- Maintain two counters \text{c\_count} and \text{u\_count} for every gate
  - \text{c\_count}: number of inputs with c values
  - \text{u\_count}: number of inputs with u values
- These counters must be updated

\[
\text{evaluate (G,c,i)} \\
\text{begin} \\
\quad \text{if } \text{c\_count} > 0 \text{ then return } c \oplus i \\
\quad \text{if } \text{u\_count} > 0 \text{ then return } u \\
\text{return } \overline{c} \oplus i \\
\text{end} \\
\text{O(1) storage} \\
\text{O(1) evaluation}
\]

\[
\begin{array}{c|c|c}
\text{c} & \text{i} & \text{c} \oplus i \\
\hline
\text{AND} & 0 & 0 \\
\text{OR} & 1 & 0 \\
\text{NAND} & 0 & 1 \\
\text{NOR} & 1 & 1 \\
\end{array}
\]
Hazard Detection

- Simulator should detect hazards (analyze transient behavior of signals)
- 3-valued logic simulation:

```
0 → 1  A
1 → 0  B
0 → C
```

Static hazard: 0 → 1 → 0
Dynamic hazard: 0 → 1 → 0 → 1

- How many logic values are necessary?
- More the better, but…more complexity

Summary

- Logic or true-value simulators are essential tools for design verification.
- Verification vectors and expected responses are generated (often manually) from specifications.
- A logic simulator can be implemented using either compiled-code or event-driven method.
- Per vector complexity of a logic simulator is approximately linear in circuit size.
- Modeling level determines the evaluation procedures used in the simulator.