Advanced Timing

This lecture is mostly based on contents of Chapter 8, from “The Verilog Hardware Description Language” book [1], 5th edition. Example figures and (modified) code are from the textbook unless otherwise specified.

Topics covered:

• Verilog simulator timing models
• Gate-level timing
• Procedural timing
• Parallelism and non-determinism
Advanced Timing
Verilog Timing Models

• Verilog can be used to simulate design
• When simulating a design, timing models are used to evaluate behavior of the design and advance time

• Two timing models are often used
  • Timing model for gate-level description
  • Timing model for procedural description

• Both essentially determine when to update the values of wires and registers
• Main difference is what triggers the update to the values
  • Any input change for gate-level description
  • Only specific events for procedural description
Gate-Level and Procedural Timing Models

• Gate-Level Timing Model
  • Any input change will cause output to change
  • If input changes in time < propagation delay, output change is not visible
  • Schedule update in N time units in future, could be 0 units

• Procedural Timing Model
  • Only events can trigger output changes
  • Updates to events trigger new output changes, even if previous output change has not happened yet
  • Schedule update in N time units in future, could be 0 units

• Procedural timing model is more flexible
  • E.g. recall we can describe combinatorial, gate-level logic, with procedural always statements

```verilog
module nandLatch
    (output q, qBar,
     input  set, reset);

    nand #2
    (q, qBar, set),
    (qBar, q, reset);
endmodule

module DFF
    (output reg q,
     input  d, clock);

    always @ (posedge clock)
        #5 q = d;
endmodule

module behavioralNand
    #(parameter delay = 5)
    (output reg out,
     input  in1, in2, in3);

    always @ (in1 or in2 or in3)
        #delay out = ~(in1 & in2 & in3);
endmodule
```
Event-Driven Simulation Basics

- The Verilog simulator tools keep track of all events in a time-ordered queue or list.
- At each time instance, check current event, then update outputs and evaluate procedural blocks, and schedule new events for future.
Gate-Level Simulation Example

- Textbook example, each gate has \( d \) time units propagation delay:

<table>
<thead>
<tr>
<th>a</th>
<th>Update</th>
<th>A=1 at ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Update</td>
<td>A=1 at ( t )</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>B=0 at ( t+d )</td>
</tr>
<tr>
<td>c</td>
<td>Update</td>
<td>A=1 at ( t )</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>B=0 at ( t+d )</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>C=1 at ( t+2d )</td>
</tr>
</tbody>
</table>

Schedule updates after \( d \) time units, no other updates between \( t \) and \( t+d \)

\[ \text{Example Diagrams} \]

- a. Initial values.
- b. Values at end of time \( t \).
- c. Values at end of time \( t + d \).
- d. Values at end of time \( t + 2d \)
Procedural Timing

• Behavioral models are evaluated only when an event happens

```verilog
module twoPhiLatch
    (input  phi1, phi2,
     output reg q,
     input  d);
    reg qInternal;
    always begin
        @ (posedge phi1)
            qInternal = d;
        @ (posedge phi2)
            q = qInternal;
    end
endmodule
```

```verilog
module twoPhiLatchWithDelay
    (input  phi1, phi2, d,
     output reg q);
    reg qInternal;
    always begin
        @ (posedge phi1)
            #2 qInternal = d;
        @ (posedge phi2)
            #2 q = qInternal;
    end
endmodule
```

Changes to d are ignored, only use d when phi1 event happens

Evaluate the change to q after 2 time units, use qInternal at the time of evaluation
Parallelism and Non-Deterministic Behavior

• Each always block, initial block, module, gate, etc. conceptually execute in parallel
  • They represent different hardware that can run in parallel
• One or more events can be updated at the same time, then there is no determinism which is updated first
  • In simulation this depends on the algorithm of the simulator
  • In hardware, this depends on small variations in delays of wires, manufacturing variations, etc.

```verilog
module stupidVerilogTricks
(output reg f,
  input a, b);

reg q;

initial
  f = 0;

always @ (posedge a)
  #10 q = b;

not (qBar, q);
always @ q
  f = qBar;
endmodule
```

Value of f depends on whether not or always block are evaluated first

In practice, want to update all registers, f, on a clock to avoid races and non-determinism
Event-Driven Simulation with Non-Blocking Events

- Event driven simulation first evaluates regular events, then non-blocking events, and monitor events.
References