EVENT DRIVEN SIMULATION

- Verilog simulation uses Event Driven model of computation
- Basic knowledge of Event Driven simulation can help in writing and Debugging Verilog descriptions
- We will examine a simplified model of ED digital circuit simulation
- Goal is to understand enough about ED simulation to help in writing good Verilog
- We will not write our own ED simulator but we will attempt to understand how they work

Example Circuit and Terms

A 1
B 1
C 0
D 0
E 1
F 0

- Test Vector – Input Stimulus (1,1,0,0,1,0)
- Net – Electrically Common Point in Circuit
- Line – Can be Part of a Net
- Gate – Transistor Circuit that Performs a Function
ED SIMULATOR STRUCTURE

Event Queue

Gate Simulation Queue

Input Vector

Basic Timing Wheel

EVENT DRIVEN SIMULATION

- Test Input Vector For Changes
- A Change is an EVENT
- Schedule Events for Primary Inputs
- Repeat Scheduling Until Finished
  - Process Each Event (Change in a net Value)
  - Schedule Gate Simulations
  - Simulate All (in no particular order)
  - Check for New Events
ED Simulation

• ED – Eliminates Unnecessary Simulations
  – Only Simulates Gates with Events on an Input
  – Used to model parallelism in circuits
• Event – Change in a NET Value
  – Each Net has a Data Structure
• ED Simulator
  – Detects Events
  – Schedules the Simulations in Response
• Dynamic Scheduling
• Test For Event When
  – New Input Vector
  – Immediately After Simulating a Gate

ED SIMULATION EXAMPLE

Suppose Input Test Vector Changes From
(1, 1, 0, 0, 1, 0) → (0, 0, 0, 0, 1, 1)
Suppose Input Test Vector Changes From 
(1, 1, 0, 0, 1, 0) \rightarrow (0, 0, 0, 0, 1, 1) 
Events are in Red

The EVENT QUEUE at TIME=1
The ED QUEUES at TIME=1

Event Queue at TIME=1

Gate Queue at TIME=1

Suppose Input Test Vector Changes From
(1, 1, 0, 0, 1, 0) \rightarrow (0, 0, 0, 0, 1, 1)
Events are in Red
The ED QUEUES at TIME=2

Simulation Time
1  2

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

Event Queue

Simulation Time
1  2

| G1 | 0, 0 |
| G3 | 1, 1 |
| G4 | 0, 0 |
| G5 | 0, 1 |

Gate Queue

ED SIMULATION EXAMPLE

Suppose Input Test Vector Changes From
(1, 1, 0, 0, 1, 0) → (0, 0, 0, 0, 1, 1)
Events are in Red
The ED QUEUES at TIME=2

Simulation Time

Event Queue

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
</tr>
</tbody>
</table>

Gate Queue

<table>
<thead>
<tr>
<th>G1</th>
<th>0, 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
<td>1, 1</td>
</tr>
<tr>
<td>G4</td>
<td>0, 0</td>
</tr>
<tr>
<td>G5</td>
<td>0, 1</td>
</tr>
<tr>
<td>G6</td>
<td>0</td>
</tr>
</tbody>
</table>

ED SIMULATION EXAMPLE

Suppose Input Test Vector Changes From
(1, 1, 0, 0, 1, 0) → (0, 0, 0, 0, 1, 1)
Events are in Red
Suppose Input Test Vector Changes From 

\[(1, 1, 0, 0, 1, 0) \rightarrow (0, 0, 0, 0, 1, 1)\]

Events are in Red

ED SIMULATION COMMENTS

• ORDER In Which Gates Are Simulated 
  (at a Given Time-Epoch) DOES NOT MATTER!
• The Event Queue is Processed Causing the Gate Queue to be Filled 
• The Simulator Alternately Processes the Event then the Gate Queue

**BASIC TIMING WHEEL**

1. Process All Events in Event Queue
2. Simulate All Gates in Simulation Queue
3. During Simulation Update EQ with New Events
4. If Event Queue Not Empty, GO TO Step 1
Timing for LCC versus ED

(a) LCC is 0-Delay (zero-Delay) Simulation
Basic ED is 1-Delay (Unit-Delay)

```
SimCkt() {
    C = ~B;
    Y = C & A;
}
```

(b) Event Driven
(0,0) → (1,1)

Event Queue

Gate Queue

Common Problem

```
REGISTER
Din
Dout
Reset

RESET CIRCUIT
CLK
```

One Way to Fix it

REGISTER

Din

Dout

Reset

Avoids Glitch – Synchronizes Register Reset Signal

TIMING

- Previous Example: Gate G2 was Simulated Twice!
- At Time=1 and Time=2
- Previous Example: 2 Simulations for G2
- ED Simulators Can Show Hazards/Races
- Intermediate Values are Stored in Event Queue Instead of Netlist Itself

EXAMPLE

- G2 after G1, but if G2 causes X1 Event, G1 and G3 sims use a different value!
- X1 must be held constant until all entries in Gate Queue are simulated
Functional versus Mapped Timing

(a) Pre-synthesis Verilog functional simulation

The example below is zero-delay as no delays are specified.

```verilog
always @*
begin
  C = ~B;
  Y = C & A;
end
```

(b) Post-synthesis simulation of Implementation X

Timing (and glitches!) depend on implementation technology.

(c) Post-synthesis simulation of Implementation Y

4x1 Memory
(4 locations, each location has 1 bit)

Functional Simulation  Timing Simulation

Blocking versus Non-blocking

begin and end typos – fixed in book p. 35

(a) Blocking assignments - RHS values applied to LHS immediately

```verilog
always @(posedge clk)
begin
  q1 <= d;
  q2 <= q1;
end
```

(b) Non-blocking assignments - all RHS values applied to LHS after `always` block exit

```verilog
always @(posedge clk)
begin
  q1 <= d;
  q2 <= q1;
end
```
Combinational Loops

(a) A combinational loop

```verilog
always @(a)
begin
    y = y + a;
end
```

(b) Sequential element in feedback path

```verilog
always @(posedge clk)
begin
    y <= y + a;
end
```

Static Hazards

- Condition where Single Input Change Produces Momentary Output Change (glitch) When no Change is Intended to Occur
- **Static-1 Hazard**: Output Should Remain Logic-1 but Glitches to Logic-0
- **Static-0 Hazard**: Output Should Remain Logic-0 but Glitches to Logic-1
Static Hazard-1 Example

- Steady State Behavior: Inputs/Outputs are Stable Values
- Transient Behavior: Output may Exhibit a “Glitch” after Input(s) change Value(s)

\[
Y = x_1 x_2 + \overline{x}_2 x_3
\]

• SOP (circuit) Form can cause Static-1 Hazard

\[
Y = (x_1 + \overline{x}_2) (x_2 + x_3)
\]

• POS (circuit) Form can cause Static-0 Hazard

Static 1 Hazard Eliminated

Static Hazard Elimination Requires Adding a Redundant Gate
Removing Static Hazards

For short time, OR inputs are both “0”

This Event Occurs AFTER This Event

Static-1 Hazard

Removing Static Hazards

Static-1 Hazard Eliminated

Static Hazard Elimination Requires
Adding a Redundant Gate
Question

\[ Y = x_1 x_2 + \bar{x}_2 x_3 \]

• If a Combinational circuit is to be Implemented in 2-level form whose function is specified by the above equation, is it possible to have a Static-0 Hazard?

Answer

• YES!!! The equation is a symbolic BEHAVIORAL description of the circuit, and the hazard depends on the technology mapped circuit. For example, it could be implemented as a 2-level POS form with no redundant gates causing a hazard, or in SOP form not allowing the static-0 hazard to occur. We must have a logic diagram to determine if a hazard exists.
Where are Static Hazards?

\[ F = X \cdot Y' \cdot Z' + W' \cdot Z + W \cdot Y \]

\[ F = X \cdot Y' \cdot Z' + W' \cdot Z + W \cdot Y + W' \cdot X \cdot Y' + Y \cdot Z + W \cdot X \cdot Z' \]
Dynamic Hazards

- These types of Hazards cause the Output to Have a Transient Output that Changes Three or More Times
- A Glitch Generally Means that the Output Changes Two Times for a Single Input Vector Change

![Dynamic Hazard Example](image)

Dynamic Hazard Example

*example from Prof. G. Dueck*
ED SIMULATOR INITIALIZATION

• So Far, Considered Events (NET changes) Only
• Cannot Initialize All NETS to 0 Initially!

• Commonly Used Method is Multiple-Valued Logic
  EXAMPLE: 3-Valued (ternary) Logic
  Use \{0, 1, U\} U is Unknown
  (Verilog uses X – Not a Don’t Care!!!!)
  Initialize All Nets to U

• Event Occurs on B Whether A is 0 or 1

Unknown Values – X in HDLs

• X (or U) means Simulator DOES NOT KNOW the Logic Value
• Don’t Cares are Assigned to 0 or 1 in REAL Circuits
• Sometimes a Simulator Can Schedule a Gate for Simulation with an Unknown Input

• Unknown Values in HDL Simulations are Usually a Sign of Trouble With Your Design!!!!!!
Multiple Valued Logic (VHDL)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Uninitialized</td>
<td>initial default value for literal</td>
</tr>
<tr>
<td>X</td>
<td>Forcing Unknown</td>
<td>forced unresolved logic value</td>
</tr>
<tr>
<td>0</td>
<td>Forcing 0</td>
<td>forced logic value 0</td>
</tr>
<tr>
<td>1</td>
<td>Forcing 1</td>
<td>forced logic value 1</td>
</tr>
<tr>
<td>Z</td>
<td>High Impedance</td>
<td>logic value of open circuit</td>
</tr>
<tr>
<td>W</td>
<td>Weak Unknown</td>
<td>weak unresolved logic value</td>
</tr>
<tr>
<td>L</td>
<td>Weak 0</td>
<td>weak logic value 0</td>
</tr>
<tr>
<td>H</td>
<td>Weak 1</td>
<td>weak logic value 1</td>
</tr>
<tr>
<td>-</td>
<td>Don’t Care</td>
<td>used for synthesis</td>
</tr>
</tbody>
</table>

Table 1.1: IEEE Standard 1164 MVL Values for VHDL

Multiple Valued Logic (Verilog)

- Verilog uses 4-valued logic for basic values

0 - represents a logic zero, or a false condition
1 - represents a logic one, or a true condition
x - represents an unknown logic value
z - represents a high-impedance state

The values 0 and 1 are logical complements of one another.

Also Incorporates “strength attribute” values:

- strength1 \{supply1, strong1, pull1, weak1, highz1\}
- strength0 \{supply0, strong0, pull0, weak0, highz0\}
### Verilog Logic Gate Primitives

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>x</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>nand</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>nor</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>x</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>xnor</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
|     | z   | x   | x   | x

### Summary

- Verilog simulation uses Event Driven model of computation
- Event Driven simulation allows for Modeling of Timing behavior
- Timing behavior can cause problems: glitches, static hazards, dynamic hazards, races
- Multiple Valued Logic needed to Model Binary-valued Logic with ED Model
- MVL values Cause Simple Logic Functions to have more complex truth tables
- Blocking versus Non-blocking used for different circuit behavior