Achieving Determinism in SystemVerilog 3.1 Scheduling Semantics

Phil Moorby, Arturo Salz, Peter Flake
Surrendra Dudani, Tom Fitzpatrick
Synopsys, Inc.

DVCon 2003
Motivation

• Enable SystemVerilog 3.1 charter
  ▪ Increase the verification capabilities of Verilog
  ▪ One language for design, testbench and assertions

• Reduce the need to use PLI to get verification environment to work

• Predictable semantics across all tools
  ▪ Simulation
  ▪ Synthesis
  ▪ Formal verification

• No performance degradation

• Backwards compatible with Verilog IEEE 1364-2001
Can IEEE 1364-2001 do the job?

• Problems
  ▪ Verilog zero-delay simulation races
    • Lack of predictability
  ▪ Lack of consistency across design and verification tools
    • Different semantics between event-driven and cycle-accurate

• Proposed solution
  ▪ Extend the scheduling semantics of the Verilog 2K1 standard
  ▪ Apply partial ordering of design, testbench and assertion-based code using 3 new event scheduling regions
Proposed new scheduling algorithm

• Why?
  ▪ Simulation races between design, testbench and assertion-based event executions are hard to avoid
  ▪ Users resort to the PLI for synchronizing and de-racing the interactions
    • Synchronize at beginning of time slot to sample data
    • Synchronize at beginning of time slot react and drive new stimulus

• Proposed solution uses three new event regions
  ▪ Preponed
  ▪ Observe
  ▪ Reactive

• Allows non-zero delay models to work with cycle-accurate models

• Enables significant performance improvements
Verilog standard flow of event regions within a time slot

- active
- inactive
- NBA
- Read-only
- pre-active
- pre-NBA
- post-NBA
- PLI

from previous time slot

iterative regions

to next time slot

Achieving Determinism in SystemVerilog 3.1 Scheduling Semantics, DVCon 2003 (5)
2-stage shift register

FF1

FF2

clk

a

b

c

drive here

sample here

clock trigger
dffGate FF1(b, a, clk);
dffGate FF2(c, b, clk);
Achieving Determinism in SystemVerilog 3.1 Scheduling Semantics, DVCon 2003

dffUdp FF1(b, a, clk);
dffUdp FF2(c, b, clk);
dffUdp FF1(b, a, clk);
dffUdp FF2(c, b, gclk2);
and G1(gclk2, clk, w1);
always @(posedge clk) begin
  c = b; // FF2
  b = a; // FF1
end

always @(posedge clk)
  b = a; // FF1
always @(posedge clk)
  c = b; // FF2
always @(posedge clk) bTemp = a;
always @(posedge clk) c <= b;
always @(posedge gclk2) b <= a;
and G1(gclk2, clk, w1);
Continuous invariant assertions on non-overlapping clocks

non-overlapping clocks

clk1

clk2

what order?

assert ((clk1 && clk2) == 0);

clk2 = clk;

clk1 <= clk;

active

inactive

NBA

clk1

observe

read-only

must read and evaluate here
assign #0 gclk = clk;
always @(posedge gclk) b = a; // FF1
always @(posedge clk) c = b; // FF2

sequence @(posedge clk) sa = (!a ; a ; !a);
sequence @(posedge clk) sc = (!c ; c ; !c);

property p = (sa => [2] sc); // clocked assertion

assert (p) pass_statement;
else; fail_statement;
Conclusions

- SystemVerilog 3.1 charter brings together design, testbench, and assertion-based code into one language

- Need for consistent semantics and results across design and verification tools, from simulation to formal verification

- A new event scheduling algorithm has been proposed that enables design and verification code to consistently work together without the need to resort to PLI synchronization