Pedal Faster! Or Make Your Verification Environment More Efficient. You Choose.

Rich Edelman, Raghu Ardeishar and Rohit Jain

Mentor Graphics

accellera SYSTEMS INITIATIVE™

2014 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Testbenches are

- as complicated (or more) than the RTL they verify.
- more and more like software.

Testbenches create and use

- complex data structures. (Scoreboards, checkers, etc.)
- Object-oriented techniques (SV UVM Class)

- Keep testbenches as fast as possible.

Easy RTL – high frequency loops – wait on the slow signal

The A loop

```
forever begin
  @(posedge clk);
  if (interrupt == 1)
    $display("interrupt=%0d", interrupt_count++);
end
```

The B loop

```
forever begin
  wait(interrupt == 1);
  @(posedge clk);
  if (interrupt == 1)
    $display("interrupt=%0d", interrupt_count++);
end
```

Use ‘ref’ arguments if possible. (minutes → seconds)

```
`define N 1000000
module top();
  int memory[`N];

  function           void check_copyin( input int i, input int m[`N] );
    // Using 'input'
    // Code checking validity of memory 'm'
  endfunction

  function automatic void check_ref    ( input int i,   ref int m[`N] );
    // Using 'ref'
    // Code checking validity of memory 'm'
  endfunction

  initial begin
    bit use_ref;
    ...
    if (use_ref) check_ref(   i, memory);
    else         check_copyin(i, memory);
    ...
endmodule
```

Randomize() – use post_randomize() for things that really aren’t random – the order is not random – get rid of the order constraint

```
class C;
  rand int ascending[`NUMBER_OF_INTS];
  constraint values { foreach (ascending[i])
    ascending[i] inside {[0:1000]}; }
  constraint order  { foreach (ascending[i])
    if (i!=0) ascending[i] > ascending[i-1]; }
endclass

class D;
  rand int ascending[`NUMBER_OF_INTS];
  constraint values { foreach (ascending[i])
    ascending[i] inside {[0:1000]}; }

  function void post_randomize();
    ascending.sort();
  endfunction
endclass
```

Watch out for “accidental” copy of LARGE structures

```
typedef int memory[int];
memory PAGES[int];

function automatic void assign_memory(int value, int address,
      int page_address);
    int m[int];
    memory p[int];

    if (fast) begin                          // Fast LEG
      PAGES[page_address][address] = value; // No accidental copying
    end
    else begin                               // Slow LEG
      if (PAGES.exists(page_address)) begin
        m = PAGES[page_address];             // Accidental copy
      end
      m[address] = value;                    // Update value into the copy
      PAGES[page_address] = m;               // Copy the copy back in
    end
  endfunction
```

Watch out for “accidental” copy of LARGE structures

```
typedef int memory[int];
memory PAGES[int];

function automatic void assign_memory(int value, int address,
      int page_address);
    int m[int];
    memory p[int];

    if (fast) begin                          // Fast LEG
      PAGES[page_address][address] = value; // No accidental copying
    end
    else begin                               // Slow LEG
      if (PAGES.exists(page_address)) begin
        m = PAGES[page_address];             // Accidental copy
      end
      m[address] = value;                    // Update value into the copy
      PAGES[page_address] = m;               // Copy the copy back in
    end
  endfunction
```

Rich Edelman
rich_edelman@mentor.com
Raghu Ardeishar
raghu_ardeishar@mentor.com
Rohit Jain
rohit_jain@mentor.com

Mentor Graphics®

Questa™ ADVANCED VERIFICATION

Conclusions

1) Don’t let your testbench get slow.
2) Watch out for unintendted copying.