Hi, All -

Some thoughts about interfaces for your consideration.

Interfaces, used just as a shorthand for declarations, are more verbose than their VHDL counterparts.

Consider the VHDL entity-architecture pair based on Bhasker's VHDL Primer book (pp. 11 & 15 - example converted to lower case):

```
-- VHDL half_adder
entity half_adder is
  port (a, b:       in  bit;
        sum, carry: out bit;
end half_adder;

architecture rtl of half_adder is
begin
  sum   <= a xor b;
  carry <= a and b;
end rtl;

//-------------------------------------------------------
// Verilog half_adder with interfaces
interface half_adder_if;
  logic a, b;
  logic sum, carry;
endinterface

// note: shorter interface instance names are better
module half_adder (half_adder_if ha_if);
  assign ha_if.sum   = ha_if.a ^ ha_if.b;
  assign ha_if.carry = ha_if.a & ha_if.b;
endmodule

//-------------------------------------------------------
// Verilog-2001 half_adder
module half_adder (output sum, carry,
                   input    a,     b);
  assign sum   = a ^ b;
  assign carry = a & b;
endmodule
```

As shown above, for small modules, using interfaces is much more verbose than Verilog-2001 and even more verbose than VHDL.

The VHDL architecture header announces that the model is to be used with the half_adder entity ("of half_adder") and from that point on, the entity member names can be used without a hierarchical reference (I know, hierarchical references are illegal in VHDL). SystemVerilog requires references to the interface member names to be done hierarchically using the interface instance name. This actually makes sense because a SystemVerilog module could instantiate multiple different interfaces or even multiple copies of the same interface and the interfaces could have members named the same. VHDL cannot attach to multiple entities.

I guess the best guideline we can share about interfaces is to instantiate them with very short instance names (I used a relatively long instance name with five characters, and that added significant verbosity to the design) and to not make interface usage a standard practice when doing simple instantiation of small to

medium modules (they are quite verbose and therefore just add characters and confusion to typical instantiations).

I don't know if this is part of the SystemVerilog 3.0 Standard anywhere, but the above SystemVerilog model would not compile using SystemSim ("Illegal unconnected interface port 'ha_if'") until I instantiated the model into a dummy testbench to make a connection to the half_adder interface. The inability to compile the module to at least test my SystemVerilog code was most annoying!

Also consider what it takes to put together a simple block-testbench for the above models using Verilog-2001 and SystemVerilog 3.0.

The Verilog-2001 testbench will be even simpler when .* is implemented. Note that the SystemVerilog 3.0 version again requires lots of extra references to "ha_if" hierarchical names.

```
//----- Verilog-2001 model with testbench -----
`timescale 1ns/1ns

// Verilog-2001 half_adder
module half_adder (output sum, carry,
                   input    a,     b);
  assign sum   = a ^ b;
  assign carry = a & b;
endmodule

module tb;
  reg  clk;
  reg  a, b;
  wire sum, carry;

  `ifdef DOTSTAR
  half_adder u1 (.*);
  `else
  half_adder u1 (.sum(sum), .carry(carry), .a(a), .b(b));
  `endif

  initial begin
    clk = 1;
    forever #10 clk = ~clk;
  end

  initial begin
                   a <= 0; b <= 0;
    @(posedge clk) a  = 0; b  = 1;
    @(posedge clk) a  = 1; b  = 0;
    @(posedge clk) a  = 1; b  = 1;
    @(posedge clk) $finish;
  end

  initial begin
    $timeformat(-9,0,"ns",8);
    $monitor("%t: carry=%b  sum=%b  a=%b  b=%b",
             $stime, carry, sum, a, b);
  end
endmodule
```

```
//----- SystemVerilog 3.0 model with testbench -----
`timescale 1ns/1ns

// Verilog half_adder with interfaces
interface half_adder_if;
  logic a, b;
  logic sum, carry;
endinterface

// note: shorter interface instance names are better
module half_adder (half_adder_if ha_if);
  assign ha_if.sum   = ha_if.a ^ ha_if.b;
  assign ha_if.carry = ha_if.a & ha_if.b;
endmodule

module tb;
  logic clk;
  half_adder_if ha_if;

  half_adder u1 (.ha_if(ha_if));

  initial begin
    clk = 1;
    forever #10 clk = ~clk;
  end

  initial begin
                  ha_if.a <= 0; ha_if.b <= 0;
    @(posedge clk) ha_if.a  = 0; ha_if.b  = 1;
    @(posedge clk) ha_if.a  = 1; ha_if.b  = 0;
    @(posedge clk) ha_if.a  = 1; ha_if.b  = 1;
    @(posedge clk) $finish;
  end

  initial begin
    $timeformat(-9,0,"ns",8);
    $monitor("%t: carry=%b  sum=%b  a=%b  b=%b",
           $stime, ha_if.carry, ha_if.sum, ha_if.a, ha_if.b);
  end
endmodule
```

I cannot think of a way to improve interfaces, but as a committee, we may want to tone down the hype about the great interface capability in SystemVerilog. When users try to use interfaces, they are in for a surprise (it's not as easy and efficient for small to medium module instantiations as they might have thought).

I do not want us to make the same mistake that the SystemC backers made. Promoters of SystemC would have had you believe that SystemC was the next generation design suite and that Verilog and VHDL were going to die. Then they scaled back and said we would still use Verilog and VHDL for gate-level simulation. Then they scaled back and said SystemC was not the best language for RTL coding. In essence, they lost credibility with the design community by over-hyping SystemC capabilities. I do not want SystemVerilog to similarly lose credibility due to promising a simplified instantiation capability that does not really exist.

Regards - Cliff