Subject: Proposal to change interface ref-port default mode -or- documentation

Hi, All -

Thanks to Jay and Dave Rich for responses.

Included below:
**Proposal to change wording in section 9.6**
**Proposal to change wording in section 18.8.1**
**Proposal to change default behavior of interface variables, including new "default ref;" declaration.**

I would like to discuss and vote on these proposals at the next sv-bc meeting.

Regards - Cliff

Jay noted section 9.6 Continuous Assignments.
" ... Variables can only be driven by one continuous assignment or one primitive output. It shall be an error for a variable driven by a continuous assignment or primitive output to have an initializer in the declaration or any procedural assignment. See also Section 5.6."

And Dave Rich noted section 18.8.1
— A ref port shall be connected to an equivalent variable data type. References to the port variable shall be treated as hierarchal references to the variable it is connected to in its instantiation. This kind of port can not be left unconnected.

Taken together, I can now see that we have sort-of shown that two continuous assignments through ref-ports could be illegal. I think the wording must be more clear than just this (proposals below). The problem is that ref-ports have been defined as this last-assignment-wins variable at a higher level of hierarchy, which is why I thought the rest of the documentation indicated that a continuous assignment through a ref-port was equivalent to a hierarchical procedural assignment to the upper data type; hence, not a true continuous assignment.

I believe the following clarifications are required to disambiguate the continuous assignment behavior to a ref port.

**Proposed change to section 9.6**
" ... Variables can only be driven by one continuous assignment or one primitive output. ***This also means that a variable cannot be driven by two continuous assignments from ref ports through two different modules to a common variable, nor can the common variable be driven by a continuous assignment through a ref port and by any other procedural assignment, including a procedural assignment through another ref port.*** It shall be an error for a variable driven by a continuous assignment or primitive output to have an initializer in the declaration or any procedural assignment. See also Section 5.6."

**Proposed change to section 18.8.1**
— A ref port shall be connected to an equivalent variable data type. References to the port variable shall be treated as hierarchal references to the variable it is connected to in its instantiation. ***If a continuous assignment is made to ref-port connected variable data type, no other continuous or procedural assignment can be made to the variable data type, even if the assignments were made through other ref ports.*** This kind of port can not be left unconnected.

For the record, we have defined a ref-port without really describing why or how they should be used. No real good examples of ref-port usage is shown in the standard.

Ref-ports are already strange to the normal Verilog user. I will accept on faith that ref-ports can be valuable for behavioral modeling, per Dave Rich's earlier mention of cache-modeling (I still would like to see a full, useful

behavioral model done using ref-ports so I could evaluate the feature based on an actual design and my own design background, but I don't think that is going to happen).

One of my big concerns is that for the first time in Verilog-coding history, an engineer will now have to know how a variable was assigned in an instantiated module because the nasty and difficult-to-debug ref-port behavior is going to be the default in SystemVerilog (interfaces without modports default to this whole new ref-port type and you better understand how ref-ports are assigned in the sub-modules).

With the previously mentioned ref-port behaviors, it is now possible for an engineer to correctly code an interface, successfully attach the interface to block tests and fail once the interface and blocks are connected inside a top-level model. See example below.

I personally believe that the default interface ref-port will yield lots of support calls to EDA vendors, a whole lot more than the .* that Karen Pieper fears so much. Engineers are going to unknowingly use ref-ports wrong and they are going to call and place the blame on the EDA tools. Even after EDA support personnel explain the nuances of the ref-port and how the engineer needs to pay more attention to the assignment types in the interface-connected modules, the engineers are going to tell your support teams that this behavior is absurd and you need to fix this in your tool. Look at how long it has taken to explain this to Cliff and I know my Verilog and SystemVerilog better than almost every RTL design engineer who is going to run into this.

If I were an EDA vendor who was trying to slow down the adoption of SystemVerilog by the engineering community, I would be all over this. I would show the "much touted interface" and how default interface ref-ports can cause debugging nightmares and require more scrutiny of instantiated modules.

I really should not be fighting this because this is obviously going to require a lot of training and examples and exercises and labs to help engineers get it right. This could add a half-day to any good SystemVerilog training class (more billable hours!)

**PROPOSAL:**
I would like to propose that an interface be allowed to include a new declaration: "default ref;" and without the "default ref" declaration, all interface nets AND VARIABLES default to inout ports. Good reasons to implement this change include:

(1) Since "default" and "ref" are already keywords, this introduces no new keywords to SystemVerilog.
(2) Since this declaration requires only two words to be added to interfaces, it still allows rapid interface prototyping without the more verbose modport declarations (seemed to be an important point with Matt Maidment).
(3) The engineer has to at least make this declaration before cutting his own throat with the new and unfamiliar ref-port types.
(4) SystemVerilog is still young and not widely used. Now is the time to fix the ref-port defaults in interfaces.

Dave Rich gave a couple of good error examples using ref-ports, but I think the examples are too simple and not indicative of the types of problems engineers will discover. Consider the following example (not tested). When just sub1 is compiled, it should work(block test #1). Turn on the switch +define+SUB2 and it should still work (block test #2). Turn on the switch +define+TOP and it fails. Non-intuitive debugging follows.

```
interface example_if;
  logic a; // a is a ref-port (no modport)
endinterface

module sub1 (example_if s1_if);
  assign s1_if.a = 1;
endmodule

module sub2 (example_if s2_if);
  assign s1_if.a = 0;
```

```
endmodule

`ifdef TOP
module top;
  logic sel, x2;
  example_if x_if;

  sub1 u1 (.s1_if(x_if));
  sub2 u2 (.s2_if(x_if));

  // Illegal
  assign x1 = !sel ? x_if.a : 'z; // u1.s1_if.a
  assign x1 =  sel ? x_if.a : 'z; // u2.s2_if.a

  initial begin
    sel = 0;
    #10 x2 = x1;
    sel = 1;
    #10 x2 = x1;
    // ...
    $finish;
  end
endmodule
`elsif SUB2
module top;
  logic sel, x2;
  example_if x_if;

  sub2 u2 (.s2_if(x_if));

  // Legal
  assign x1 =  sel ? x_if.a : 'z; // u2.s2_if.a

  initial begin
    sel = 0;
    #10 x2 = x1;
    sel = 1;
    #10 x2 = x1;
    // ...
    $finish;
  end
endmodule
`else
module top;

example_if x_if;
  logic sel, x2;
  sub1 u1 (.s1_if(x_if));

  // Legal
  assign x1 = !sel ? x_if.a : 'z; // u1.s1_if.a

  initial begin
    sel = 0;
    #10 x2 = x1;
    sel = 1;
    #10 x2 = x1;
```

```
    // ...
    $finish;
  end
endmodule
`endif
```