Problem: Assertion error messages report the wrong scope.

Context: Assertions defined in an interface.

Description: When assertions (from an interface) fail, they report an error with the scope of where they are instantiated.

```
interface simple;
   // Report a failure when the protocol is not adhered to.
   A1 : assert property @(posedge clk)
       (.......)
    else $error(“This message is here.”);
endinterface

module parent;
   simple intf; // Declare the interface with a name.

   intf_driver D1 (.intf(intf), ...); // Connect the interface in here.
endmodule

module intf_driver (...);
   // Report a failure of this module to adhere to the protocol.
   A2 : assert property @(posedge clk)
       (.......)
    else $error(“This message is here.”);
```

If the assertion **A1** fails, it would report an error message something like:

```
Error at time N, in file XXX, line YYY, parent.intf.A1,
      “This message is here.”
```

Consider the error message when assertion **A2** (from module **intf_driver**) fails:

```
Error at time N, in file XXX, line YYY, parent.D1.A2,
      “This message is here.”
```

This error message points the reader to the module where the error was generated - **intf_driver**. With the message from assertion A1, the reader has to:

1) Examine the parent scope to determine what modules are connected to the interface **intf**.

2) Then they need to examine which module was driving the particular signals in common (by looking at the module and the interface definition.)

3) Then they now know which module caused the problem.

Solution:

Modports can import tasks and functions into or out from the module instantiating the modport. This gives visibility to the task in both contexts (the module and the interface.) Importing of an assertion statement following the task model would allow the assertion to be executed from the proper scope and thus report the proper scope in its error message.

Section 19
Interfaces

19.1 Introduction

Add additional paragraph.

Labeled assertion and coverage statements can be imported into the module scope using the interface. This provides locality of the statements for the purposes of reporting error messages or coverage information to the user. Locality of reported error messages directs the user to the chosen imported scope. With careful selection of the scope for the specific assertion statements, a user can be directed to the actual module that is causing (or forwarding) the failure. Properties and sequences defined within an interface can also be imported for an imported assertion or coverage statement to reference.

19.5.5 Example of imported assertions.

This interface contains a simple bus definition with two modports for a bus master and a bus slave. The master imports the assertion so that if it violates the assertion, the error message reports its scope to the user. The user can then see the specific scope and investigate the problem from that scope.

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, wdata, rdata;
  logic [1:0] mode; // Only legal values of 0, 1, 2
  logic start, rdy;

  modport slave (input req, addr, mode, start, clk, wdata
                 output gnt, rdy, rdata,
                );
  modport master(input gnt, rdy, clk, rdata,
                 output req, addr, mode, start, wdata,
                 import goodMode, // imported assertion.
                );
  // Assertion to be imported into the module using the master modport.
  goodMode: assert property @(posedge clk)
    not (mode == 3)
     else $error(“Mode set to illegal value of 3.”);
endinterface: simple_bus
```



```
module memMod (
   simple_bus.slave a
); // interface name and modport name
logic avail;
always @(posedge a.clk) // the clk signal from the interface
a.gnt <= a.req & avail; // the gnt and req signal in the interface
endmodule

module cpuMod (simple_bus.master b);
...
  // Fixme - this could create a burst write, but that's illegal.
  assign mode = {burst, write|read}; // Can do read, burst read, or write.

endmodule

module top;
logic clk = 0;
simple_bus sb_intf(clk); // Instantiate the interface
initial repeat(10) #10 clk++;
memMod mem(.a(sb_intf)); // Connect the interface to the module instances
cpuMod cpu(.b(sb_intf));
endmodule
```

Use of the import statement in the master modport has these effects:

1) Use of the modport 'master' (as the module cpuMod) does, imports the assertion statement *goodMode* into the module *cpuMod* where it will execute and report errors with the scope of where cpuMod is instantiated.

2) Disable execution of the assertion statement in the interface scope. In the example, the assertion *goodMode* will not be executing in the scope *top*.

If assertion, or coverage statements are imported into modports that are not instantiated, the statement will execute in the interface scope.

When assertions that are imported, the signals referenced by the assertion must be part of the modport definition. It shall be an error for an imported assertion to reference a signal that is not defined in the module scope. It shall be an error for imported assertions to reference a property not defined in the module scope. The property must also be imported in the modport definition.

