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.

Changes to the LRM:

Update Syntax box in 19.2 with BNF changes proposed below.

Add this section to the LRM.

19.7 Assertions in interfaces.

Assertion (and cover) statements are very valuable in an interface, providing additional information about the definition of the interface and the associated protocol it must follow. Assertions and properties can be included in interfaces to define procotols and errors to be reported when the protocols are violated. When an error is reported, the scope of where the interface is instantiated is reported. For example:

```
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
```

If the protocol is not followed, an error message will be reported, with some similarity as:

```
Error at time 200ns, in file <file>, line ###, parent.intf.A1,
      “This message is here.”
```

Note, the scope (“parent.intf.A1”) reflects the instance of the interface and the label of the property. This scope may not reflect the particular scope that caused the error. A module connecting to this interface and driving the signals may be the better scope to report as the originator of the error. This can be done by importing assertions into a modport definition.

19.7.1 Importing assertions in modports

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 sees the specific scope and investigate the problem at that scope. The following interface defines a simple bus with two modport definitions, a `slave` and a `master`. The master drives the requests of the bus and the slave responds to the requests. The master modport imports the assertion `goodMode` for execution where the

modport will be used.

```
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 checking the outputs
                );

  // Assertion to be imported into the module using the “master” modport.
  goodMode: assert property @(posedge clk)
    not (mode == 3)
     else $error(“Simple_bus Mode set to illegal value of 3.”);
endinterface: simple_bus

The modules defined below show the usage of the interface and the modports.

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
```

The use of the import statement in the master modport has these effects:

1) Imports the assertion statement `goodMode` into the module `cpuMod` where it will execute and report errors using the scope of where `cpuMod` is instantiated.

2) The assertion or property is still bound to the specific signals, properties or sequences defined in the interface.

3) Disables execution of the assertion statement in the interface scope. In the example, the assertion `goodMode` will not be executing in the scope `top.intf` . Regardless of the use of the modport, the assertion will not execute in the interface scope once imported into a modport definition.

If an error occurrs in the above system, due to the assertion goodMode, the error message would be similar to:

```
Error at time 200ns., in file <file>, line ###, top.cpu.goodMode,
      “Simple_bus Mode set to illegal value of 3.”
```

Assertions can be imported into multiple modport definitions. Imported assertions reference the signals, properties or sequences from the interface instance.

Changes to the BNF:

Rename “modport_tf_ports_declaration” as “modport_import_export_declaration”
in: modport_ports_declaration,

Rename “modport_tf_port” as “modport_decl_port”

Replace “modport_tf_port”

modport_tf_port ::= task named_task_proto { , named_task_proto } | function named_function_proto { , named_function_proto } | task_or_function_identifier { , task_or_function_identifier }

with the new, “modport_decl_port”

modport_decl_port ::=

task named_task_proto { , named_task_proto }

| function named_function_proto { , named_function_proto }

| import_export_identifier

Replace definition “task_or_function_identifier” with

import_export_identifier ::=

task_identifier | function_identifier

| assert_statement_identifier | cover_statement_identifier

Thought for friendly amendment:

Is this legal? I believe it is. Should we provide an example of this?

```
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;

  ...
```



```
sequence valid_trans;
   @(posdge clk)
   ( $rose(start) [->1]; rdy [->1]);
endsequence

endinterface: simple_bus


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.

completeAddr0: cover property @(posedge clk)
   ($rose(start), addr == 0 |-> b.valid_trans); // Reference property ***

endmodule
```

