1.1 Assumptions

The purpose of this enhancement is allow properties to be considered as assumptions for formal as well as for dynamic simulation tools. When a property is assumed, the tools may constraint the environment so that the property holds. Additionally, for random simulation, biasing on the inputs provides a way to make random choices.

This proposal includes two extensions. First, it provides an assumption construct, similar to the assertion and cover construct. Second, it extends the expression syntax to allow biasing specification. The syntax of the biasing is already provided by the constraint feature of System Verilog.

1.1.1 Syntax for assumption

procedural_assertion_item ::= // from Annex A.6.10

    assert_property_statement
    | cover_property_statement
    | assume_property_statement

concurrent_assertion_item ::=

    concurrent_assert_statement
    | concurrent_cover_statement
    | concurrent_assume_statement

concurrent_assert_statement ::= // from Annex A.2.10

    [block_identifier**:**] assert_property_statement

concurrent_cover_statement ::=

    [block_identifier**:**] cover_property_statement

concurrent_assume_statement ::=

    [block_identifier**:**] assume_property_statement

assert_property_statement::=

    assert property (property_spec **)** action_block
    | **assert property (** property_instance **)** action_block

cover_property_statement::=

    cover property (property_spec **)** statement_or_null
    | **cover property (** property_instance **)** statement_or_null

assume_property_statement::=

    assume property (property_spec **) ;**
    | **assume property (** property_instance **) ;**

The procedural_assertion_item and concurrent_assertion_item are extended to include the **assume** construct.
Note that **assume** does not provide an action block, as the actions for an assumption serve no purpose.

1.1.2 Semantics for assume

For the procedural usage of **assume**, the rules governing the inference of clocks and enabling conditions are identical to **assert**. Also, the rules for interpretation when embedded in an always or an initial block are identical to **assert**. Namely, when **assume** is embedded in an always block, the property is assumed to hold for every clock tick. When **assume** is embedded in an initial block, the property is assumed to hold for the first clock tick.

The tools must constraint the environment such that the properties that are assumed shall hold. Like an assert property, an assumed property must be checked and reported if it fails to hold. There is no requirement on the tools to report successes of the assumed properties.

For formal analysis tools, there is no obligation to verify that the assumed properties hold. An assumed property may be considered as a hypothesis to prove the asserted properties.

1.1.3 Syntax for biasing

assertion_expression ::=

  expression

  | expression **dist** **{** dist_list **}** **;** // from Annex A.1.9

dist_list ::= dist_item { **,** dist_item }

dist_item ::=

  value_range **:=** expression

  | value_range **:/** expression

value_range ::=

  expression

  | **[** expression **:** expression **]**

This syntax introduces a new production named assertion_expression. In the BNF for assertions, the expression production must be replaced by assertion_expression. The operator **dist** and the production dist_list is explained in Section 12.4.4 of the System Verilog LRM.

1.1.4 Semantics for biasing

The biasing feature is only useful when properties are considered as assumptions to drive random simulation. For assertions or coverage, the biasing that is associated with any expression can be safely ignored converted to the set membership function. For example,

```
a1:assume property @(posedge clk) req dist {0:=40, 1:=60} ;
property proto
   @(posedge clk) req |-> req[*1:$] ##0 ack;
endproperty

a1_assertion:assert property req inside {0, 1} ;
property proto_assertion
  @(posedge clk) req |-> req[*1:$] ##0 ack;
endproperty
```

In the above example, signal `req` is specified with distribution in assumption `a1`, and is converted to an equivalent assertion `a1_assertion`.

Also, formal tools may follow the same conversion for assertions as well as for assumptions.

The semantics of biasing construct are explained in Section 12.4.4 of the System Verilog LRM.

It should be noted that the properties that are assumed must hold in the same way with or without biasing. The biasing simply provides a means to select values of free variables, according to the specified weights, when there is a choice of selection at a particular time.

1.1.5 Example of assume

Consider a simple synchronous request - acknowledge protocol, where the variable req can be raised at any time and must stay asserted until ack is asserted. In the next clock cycle both req and ack must be deasserted.

Properties governing req are:

```
property pr1;
   @(posedge clk) !reset_n |-> !req; //when reset_n is asserted (0),keep req 0
endproperty
property pr2;
   @(posedge clk) ack |=> !req; // one cycle after ack, req must be deasserted
end property
property pr3;
   @(posedge clk) req |-> req[*1:$] ##0 ack; // hold req asserted until
                                             // and including ack asserted
endproperty
```

Properties governing ack are:

```
property pa1;
   @(posedge clk) !reset_n || !req |-> !ack;
endproperty
property pa2;
   @(posedge clk) ack |=> !ack;
endproperty
```

When verifying the behavior of a protocol controller which has to respond to requests on req, the assertions assert_req1 and assert_req2 should be proven while assuming that statements assume_ack1, assume_ack2 and assume_ack3 hold at all times.

```
assume_ack1:assume property (pr1);
assume_ack2:assume property (pr2);
assume_ack3:assume property (pr3);

assert_req1:assert property (pa1)
   else $display("\n ack asserted while req is still deassrted");
assert_req2:assert property (pa2)
   else $display("\n ack is extended over more than one cycle");
```