Motivation

It is often the case that all the concurrent assertions that are placed in a design unit share the same clock and disable iff condition. While it is possible to define a default clocking for the assertions, it is not possible to do so for the disabling condition. It is then necessary to repeat the same code in all the assertions. This proposal remedies the situation by introducing a default disable declaration. Note that its scoping rules are different from the current default clocking scoping rules in that default disable can be redefined in nested module declarations. However, there is a Mantis ticket #1799 which asks to modify the default clocking scoping rules in the same way.

ADD

16.15 Disable resolution

Note to editor: Shift the numeration of the following subsections accordingly.

Note to the editor: Add a Syntax Box containing the following text:

```
module_or_generate_item_declaration ::=                    // from A.1.4
        ...
        | default clocking clocking_identifier ;
        | default disable expression_or_dist ;
```

A default disable condition may be declared as an item within a module, interface, or program. A default disable applies only within its scope which can be the module, interface, or program, but its effect is independent of the position of the declaration within the scope. Declaring more than one default disable item within the same module, interface, or program shall result in a compilation error. Furthermore, the scope also includes any nested module, interface, or program declaration. The scope does not include instantiated modules or interfaces, however, if a nested module, interface, or program declaration itself has a default disable declaration, then that default disable applies within the nested declaration and overrides any default disable from without.

In the following example, the inferred reset condition of both assertions `a1` and `a2` is `rst1.`

```
module m1;
   bit clk, rst1;
   default disable rst1;
   a1: assert property (@clk p1); // property p1 is defined elsewhere
   ...
   module m2;
      bit rst2;
      ...
      a2: assert property (@(posedge clk) p2); // property p2 is defined
elsewhere
   endmodule
   ...
endmodule
```

However, when a different default disable condition is specified in `m2`, it overrides the default condition in `m1`, therefore in the following example the inferred reset condition of `a1` is `rst1`, but the inferred reset condition of `a2` is `rst2`.

```
module m1;
   bit clk, rst1;
   default disable rst1;
   a1: assert property (@clk p1); // property p1 is defined elsewhere
   ...
   module m2;
      bit rst2;
      default disable rst2;
      ...
      a2: assert property (@(posedge clk) p2); // property p2 is defined
elsewhere
   endmodule
   ...
endmodule
```

The following rules apply for the disable condition resolution:

a) If an assertion has a **disable iff** clause, then the disable specified in this clause will be used, and any **default disable** statement will be ignored.

b) If an assertion does not contain a **disable iff** clause, but the assertion is within the scope of a **default disable** statement, then the disabling value for the assertion is inferred from the **default disable** statement.

c) Otherwise, no inference is performed (this is equivalent to the inference of a `1'b0` disable condition).

Below are two example modules illustrating the application of these rules.

```
module examples_with_default (input logic a, b, clk, rst, rst1);

default disable rst;

property p1;
   disable iff (rst1) a |=> b;
endproperty

// Disable condition is rst1 - explicitly specified within a1
a1 : assert property (@(posedge clk) disable iff (rst1) a |=> b);

// Disable condition is rst1 - explicitly specified within p1
a2 : assert property (@(posedge clk ) p1);

// Disable condition is rst - no explicit specification, inferred from
// default disable statement
a3 : assert property (@(posedge clk) a |=> b);

// Disable condition is 1'b0 . This is the only way to
// cancel the effect of default disable.
a4 : assert property (@(posedge clk) disable iff (1'b0) a |=> b);

endmodule

module examples_without_default (input logic a, b, clk, rst);

property p2;
   disable iff (rst) a |=> b;
endproperty

// Disable condition is rst - explicitly specified within a5
a5 : assert property (@(posedge clk ) disable iff (rst) a |=> b);

// Disable condition is rst - explicitly specified within p2
a6 : assert property (@ (posedge clk ) p2);

// No Disable condition
a7 : assert property (@ (posedge clk ) a |=> b);
```

```
// Only enable condition and clocking event are inferred from an always block
// Assertion a8 is equivalent to
//    @(posedge clk)  !bit'(rst!='b0) |-> (a |=> b)

always @(posedge clk or posedge rst)
if (rst)
   ...
else begin
   a8 : assert property ( a |=> b);
   ...
end

endmodule
```

In assertion `a8` the inferred enabling condition is from the else clause of the **if-else** statement, and thus it has to represent the complementary interpretation of the four-valued expression in the **if** condition. One such form is as indicated in `a8`. Other equivalent forms may be used, such as `((rst != 'b0) !== 1'b1)`.

14-12 Default clocking

Change in Syntax 14-3 from

module_or_generate_item_declaration ::= *// from A.1.4*
    ...
    | **default clocking** clocking_identifier ;

to

module_or_generate_item_declaration ::= *// from A.1.4*
    ...
    | **default clocking** clocking_identifier ;
    ...

A.1.4 Module items

REPLACE

module_or_generate_item_declaration ::=
    package_or_generate_item_declaration
    | genvar_declaration
    | clocking_declaration
    | **default clocking** clocking_identifier **;**

WITH

module_or_generate_item_declaration ::=
    package_or_generate_item_declaration
    | genvar_declaration
    | clocking_declaration
    | **default clocking** clocking_identifier **;**
    | **default disable** expression_or_dist **;**