

The following changes are relative to P1800-2008 Draft 4

Motivation

The proposal introduces immediate assume and cover statements. The text assumes that Mantis items 1361 (Action block control tasks) and 1461 (Severity tasks) have been accepted.

In 16.3, REPLACE

16.3 Immediate assertions

The immediate assertion statement is a test of an expression performed when the statement is executed in the procedural code. The expression is nontemporal and is interpreted the same way as an expression in the condition of a procedural **if** statement. In other words, if the expression evaluates to **x**, **z**, or **0**, then it is interpreted as being false, and the assertion is said to fail. Otherwise, the expression is interpreted as being true, and the assertion is said to pass.

The immediate **assert** statement is a *statement_item* and can be specified anywhere a procedural statement is specified. The execution of immediate assertions can be controlled by using assertion control system tasks (See 19.10).

The information about assertion failure can be printed using one of the following severity system tasks in the action block.

- `$fatal` is a run-time fatal.
- `$error` is a run-time error.
- `$warning` is a run-time warning.
- `$info` indicates that the assertion failure carries no specific severity.

The syntax for these severity system tasks is shown in 19.9.

If an assertion fails and no **else** clause is specified, the tool shall, by default, call `$error`, unless `$assertfailoff` is used to suppress the failure.

```
procedural_assertion_statement ::=                               // from A.6.10
...
    | immediate_assert_statement
immediate_assert_statement ::=
    assert ( expression ) action_block
action_block ::=                                              // from A.6.3
    statement_or_null
    | [ statement ] else statement_or_null
```

Syntax 16-1—Immediate assertion syntax (excerpt from Annex A)

The *action_block* specifies what actions are taken upon success or failure of the assertion. The statement associated with the success of the **assert** statement is the first statement. It is called the *pass statement* and is executed if the expression evaluates to true. The pass statement can, for example, record the number of successes for a coverage log, but can be omitted altogether. If the pass statement is omitted, then no user-specified action is taken when the **assert** expression is true. The statement associated with **else** is called a *fail statement* and is executed if the expression evaluates to false. The **else** statement can also be omitted.

The action block is executed immediately after the evaluation of the assert expression. The execution of pass and fail statements can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

The severity system tasks can be used in assertion pass or fail statements. These tasks shall print the same tool-specific message when used either in a pass or a fail statement. For example:

```
assert_foo : assert(foo) $info("passed"); else $error("failed");
```

If more than one of these system tasks is included in the action block, then each shall be executed as specified.

If the severity system task is executed at a time other than when the assertion fails, the actual failure time of the assertion can be recorded and displayed programmatically. For example:

```
time t;  
  
always @(posedge clk)  
  if (state == REQ)  
    assert (req1 || req2);  
    else begin  
      t = $time;  
      #5 $error("assert failed at time %0t",t);  
    end
```

If the assertion fails at time 10, the error message shall be printed at time 15, but the user-defined string printed shall be “assert failed at time 10”.

WITH

16.3 Immediate assertions

The immediate assertion statement is a test of an expression performed when the statement is executed in the procedural code. The expression is non-temporal and is interpreted the same way as an expression in the condition of a procedural **if** statement. In other words, if the expression evaluates to X, Z, or 0, then it is interpreted as being false, and the assertion **statement** is said to fail. Otherwise, the expression is interpreted as being true, and the assertion **statement** is said to pass or, equivalently, to succeed.

The ~~immediate assert statement~~ *immediate_assertion_statement* is a *statement_item* and can be specified anywhere a procedural statement is specified. The execution of immediate assertions can be controlled by using assertion control system tasks (See 19.10).

The information about assertion failure can be printed using one of the following severity system tasks in the action block.

- \$fatal is a run-time fatal.
- \$error is a run-time error.
- \$warning is a run-time warning.
- \$info indicates that the assertion failure carries no specific severity.

The syntax for these severity system tasks is shown in 19.9.

~~If an assertion fails and no else clause is specified, the tool shall, by default, call \$error, unless \$assertfailoff is used to suppress the failure.~~

```

procedural_assertion_statement ::=                                //from A.6.10
    ...
    | immediate_assert_statement
    | immediate_assertion_statement
immediate_assertion_statement ::=
    immediate_assert_statement
    | immediate_assume_statement
    | immediate_cover_statement
immediate_assert_statement ::=
    assert ( expression ) action_block
immediate_assume_statement ::=
    assume ( expression ) action_block
immediate_cover_statement ::=
    cover ( expression ) statement_or_null
action_block ::=                                              //from A.6.3
    statement_or_null
    | [ statement ] else statement_or_null

```

Syntax 16-1—Immediate assertion syntax (excerpt from Annex A)

There are three types of immediate assertions: immediate **assert**, immediate **assume**, and immediate **cover**.

The immediate **assert** statement specifies that its *expression* is required to hold. Failure of an immediate **assert** statement indicates a violation of the requirement and thus a potential error in the design. If an **assert** statement fails and no **else** clause is specified, the tool shall, by default, call `$error`, unless `$assertfailoff` is used to suppress the failure.

The immediate **assume** statement specifies that its *expression* is assumed to hold. For example, immediate **assume** statements can be used with formal verification tools to specify assumptions on design inputs that constrain the verification computation. When used in this way, they specify the expected behavior of the environment of the design as opposed to that of the design itself. In simulation, an immediate **assume** may behave as an immediate **assert** to verify that the environment behaves as assumed. A simulation tool shall provide the capability to check the immediate **assume** statement in this way.

The *action_block* of an immediate **assert** or **assume** statement specifies what actions are taken upon success or failure of the immediate assertion. The statement associated with ~~the success of the assert-statement~~ is the first statement. It is called the *pass statement* and ~~is~~ shall be executed if the ~~expression~~ *expression* evaluates to true. The pass statement can, for example, record the number of successes for a coverage log, but can be omitted altogether. If the pass statement is omitted, then no user-specified action is taken when the ~~assert-expression~~ *expression* of the immediate **assert** or **assume** is true. The statement associated with **else** is called ~~a~~ the *fail statement* and ~~is~~ shall be executed if the ~~expression~~ *expression* evaluates to false. The **else** statement can also be omitted. The action block ~~is executed~~ shall be enabled to execute immediately after the evaluation of the ~~assert-expression~~ *expression* of the immediate **assert** or **assume**.

The immediate **cover** statement specifies that successful evaluation of its *expression* is a coverage goal. Tools shall collect coverage information and report the results at the end of simulation or on demand via an assertion API (see Clause 38). The results of coverage for an immediate **cover** statement shall contain the following:

- Number of times evaluated
- Number of times succeeded

A pass statement for an immediate **cover** may be specified in *statement_or_null*. The pass statement shall be executed if the *expression* evaluates to true. The pass statement shall be enabled to execute immediately after the evaluation of the expression of the immediate **cover**.

The execution of pass and fail statements can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

The severity system tasks can be used in assertion pass or fail statements. These tasks shall print the same tool-specific message when used either in a pass or a fail statement. For example:

```
assert_foo : assert(foo) $info("passed"); else $error("failed");  
assume_inputs: assume (in_a || in_b) $info("assumption holds");  
                else $error("assumption does not hold");  
cover_a_and_b: cover (in_a && in_b) $info("in_a && in_b == 1 covered");
```

For example, a formal verification tool might prove `assert_foo` under the assumption `assume_inputs` expressing the condition that `in_a` and `in_b` are not both 0 at the same time. The `cover` statement detects whether `in_a` and `in_b` are both simultaneously 1.

If more than one of these system tasks is included in the action block, then each shall be executed as specified.

If the severity system task is executed at a time other than when the **assertion immediate assert or assume** fails, the actual failure time of the **assertion immediate assert or assume** can be recorded and displayed programmatically. For example:

```
time t;  
  
always @(posedge clk)  
  if (state == REQ)  
    assert (req1 || req2);  
    else begin  
      t = $time;  
      #5 $error("assert failed at time %0t",t);  
    end
```

If the **assertion immediate assert** fails at time 10, the error message shall be printed at time 15, but the user-defined string printed shall be “assert failed at time 10”.

In Figures 36.42 and 36.51, REPLACE

immediate assert

WITH

immediate assert

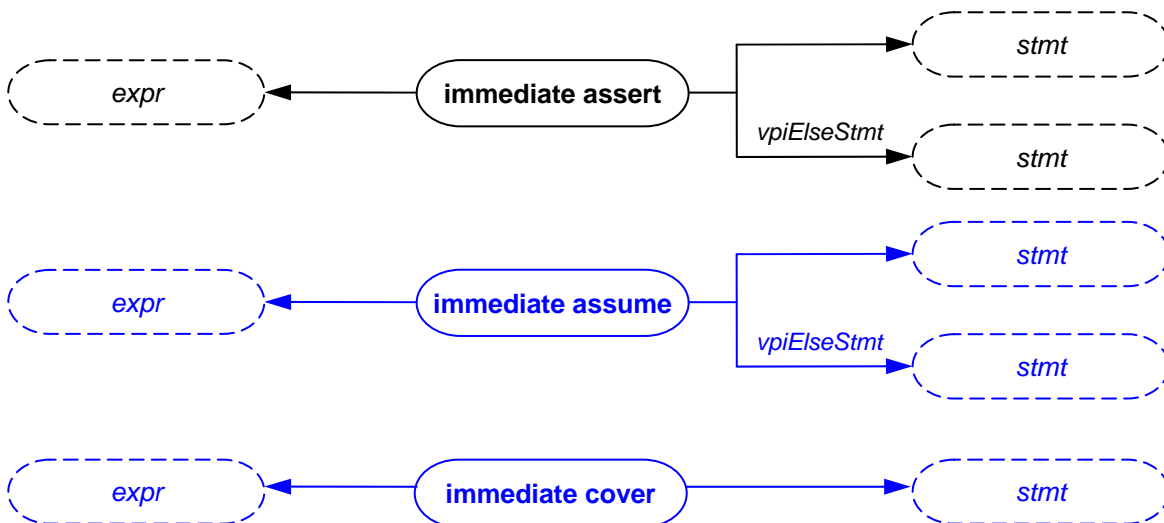
immediate assume

immediate cover

In Figure 36.47, REPLACE



WITH



In 38.3.1, REPLACE

d) To obtain an assertion of a specific type, e.g., cover assertions, the following approach should be used:

WITH

d) To obtain an assertion of a specific type, e.g., ~~cover assertions~~ concurrent ~~cover~~ **property** statements, the following approach should be used:

In 38.3.2, REPLACE

- Assertion type
 - Sequence
 - Assert
 - Assume
 - Cover
 - Property
 - ImmediateAssert

WITH

- Assertion type
 - Sequence
 - Assert
 - Assume
 - Cover
 - Property
 - ImmediateAssert
 - ImmediateAssume
 - ImmediateCover

In A.6.10, REPLACE

```
procedural_assertion_statement ::=
    concurrent_assertion_statement
    | immediate_assert_statement
immediate_assert_statement ::=
    assert ( expression ) action_block
```

WITH

```
procedural_assertion_statement ::=
    concurrent_assertion_statement
    | immediate_assert_statement
    | immediate_assertion_statement
immediate_assertion_statement ::=
    immediate_assert_statement
    | immediate_assume_statement
    | immediate_cover_statement
immediate_assert_statement ::=
    assert ( expression ) action_block
immediate_assume_statement ::=
    assume ( expression ) action_block
immediate_cover_statement ::=
    cover ( expression ) statement_or_null
```

Annex L on page 1102, REPLACE

```
#define vpiImmediateAssert 665
```

WITH

```
#define vpiImmediateAssert 665  
#define vpiImmediateAssume editor to fill  
#define vpiImmediateCover editor to fill
```