

## The following changes are relative to P1800-2008 Draft 3a

### Change 16.3 on page 306

#### 16.3 Immediate assertions, assumptions and covers

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.

The ~~immediate\_assert\_statement~~ *immediate\_assertion\_statement* is a *statement\_item* and can be specified anywhere a procedural statement is specified.

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

The immediate **assert** statement is an expression of required values on design variables. A failure of an **assert** statement indicates a violation of the requirement and thus a potential error in the design.

The immediate **assume** statement is an expression of assumed values on variables. They are usually employed with formal verification tools to specify constraints on signal values under which assertions should hold. They specify the expected behavior of the environment of the design, as opposed to the design itself. In simulation assumptions are checked in the same manner as an assertion to verify that the environment behaves as expected.

The immediate **cover** statement is used to detect the occurrence of specific signal values in the procedural code. The tools can collect such information in a database and then report the results at the end of simulation. The reporting should include the number of times the **cover** statement expression was true and in that sense it is equivalent to recording the success of an **assert** statement on the same expression. **cover** statements can also be used as search targets in formal tools.

The results of a **cover** statement shall contain the following:

- Number of times succeeded
- Number of times failed

In addition, *statement\_or\_null* is executed every time expression is true.

```
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
```

## Change paragraph after Syntax 16-1

The *action\_block* specifies what actions are taken upon success or failure of the assertion. The statement associated with the success of the **assert**, **assume** or **cover** 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.

## Replace

```
assert_foo : assert(foo) $display("%m passed"); else $display("%m failed");
```

## With

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

The assertion `assert_foo` may be proven under the assumption `assume_inputs` expressing the fact that both `in_a` and `in_b` are not 0 at the same time. The cover statement detects whether both `in_a` and `in_b` were simultaneously 1.

## Change in 38.3.1, page 916, item d

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

## Add in 38.3.2

```
...  
— ImmediateAssert  
— ImmediateAssume  
— ImmediateCover
```

## Replace

## Change on page 986

### A.6.10 Assertion statements

```
procedural_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

```

## Add to Figures 36.42, 36.51

```

immediate assume
immediate cover

```

## Add to Figure 36.47

```

-----> stmt
expr <|---- immediate assume ---|
    |vpiElseStmt
    ----->stmt

```

```

expr <|---- immediate cover -----> stmt

```

## Annex L on page 1102

### Change

```

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

```