

Replace

17.2 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,

The immediate assert statement is a `statement_item` and can be specified anywhere a procedural statement is specified.

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

With

17.2 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 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 statement is said to fail. Otherwise, the expression is interpreted as being true, and the assertion statement is said to pass.

The immediate assertion statements is a `statement_items` and can be specified anywhere a procedural statement is specified.

There are three types of immediate assertions: immediate **assert**, immediate **assume**, and an 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 the equivalent assert statement. Cover statements can also be used as search targets in formal tools.

The results of 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_assert_statement ::=
    immediate_assert
    | immediate_assume
    | immediate_cover
immediate_assert ::=
    assert ( expression ) action_block
immediate_assume ::=
    assume ( expression ) action_block
immediate_cover ::=
    cover ( expression ) statement_or_null
action_block ::=                                                  //from A.6.3
    statement_or_null
    | [ statement ] else statement

```

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 28.2.1, 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 28.2.2

```

...
— Immediate Assert
— Immediate Cover
— Immediate Assume

```

Replace

A.6.10 Assertion statements

```

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

```

With

A.6.10 Assertion statements

```

procedural_assertion_statement ::=
    concurrent_assertion_statement
    | immediate_assert_statement
immediate_assert_statement ::=
    immediate_assert
    | immediate_assume
    | immediate_cover
immediate_assert ::=
    assert ( expression ) action_block
immediate_assume ::=
    assume ( expression ) action_block
immediate_cover ::=
    cover ( expression ) statement_or_null

```

Add to Figures 27.31, 27.40

```

immediate assume
immediate cover

```

Add to Figure 27.36

```

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

```

```

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

```

Annex I

Change

```

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

```