Clause: 16.3

Description

Currently, LRM mentions that pass action block is executed whenever property returns true. This causes pass actions to be executed for vacuous succeses. This makes pass action blocks unusable in a lot of cases.

Suggested Resolution

(Note to the Editor: A new sub-clause is getting added after Action Control Tasks in section 19. Please adjust the number of the new sub-clause based on that. Green color for the clause number is used to remind about it.)

Clause 16.3, add the following on page 307:

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.

Clause 16.14.1, change the following on page 362:

The **`assert`** statement is used to enforce a **`property`** as a checker. When the property for the **`assert`** statement is evaluated to be true, the pass statements of the action block are executed. When the property for the **`assert`** statement is evaluated to be false, the fail statements of the *action_block* are executed. When the property for the **`assert`** statement is evaluated to be disabled, no `action_block` statement is executed. The execution of pass and fail actions can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

Clause 16.14.2, change the following on page 362:
(Note to the Editor: The following change will be done after incorporating changes for 1460)

If the property has a disabled evaluation, neither pass nor fail statements of the *action block* are executed. The execution of pass and fail actions can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

Clause 16.14.3, change the following on page 364:
(Note to the Editor: The following changes will be made if 1768 proposal does not get approved)

In addition, *statement_or_null* is executed every time a property is being evaluated to true. The execution of *statement_or_null* can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

In addition, statement_or_null gets executed for every match. If there are multiple matches at the same time, the statement gets executed multiple times, one for each match. The execution of *statement_or_null* can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

(Note to the Editor: The following changes will be made if 1768 is approved)

The pass statement specified in statement_or_null shall be executed once for each successful evaluation attempt of the underlying property_spec. The pass statement shall be executed in the Reactive region of the timestep in which the corresponding evaluation attempt succeeds. The execution of *statement_or_null* can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

The pass statement specified in statement_or_null shall be executed, with multiplicity, for each match that is counted toward the total for the attempt. The pass statement shall be executed in the Reactive region of the timestep in which the corresponding match completes. The execution of *statement_or_null* can be controlled by using assertion action control tasks. The assertion action control tasks are described in 19.11.

Clause 16.16, change the following on page 364:

When executed, the **expect** statement starts a single thread of evaluation for the given property on the subsequent clocking event, that is, the first evaluation shall take place on the next clocking event. If the property fails at its clocking event, the optional **else** clause of the action block is executed. If the property succeeds the optional pass statement of the action block is executed. 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.

Clause 19.1, add the following:

Assertion Action control (19.11)

```
$assertpasson           $assertpassoff
$assertfailon           $assertfailoff
$assertnonvacuouson     $assertvacuousoff
```

(Note to the Editor: Shift the numeration of subsequent items accordingly)

ADD the following new section:

19.11 Assertion action control system tasks

Note to Editor: This is a new section following 19.10 (Assertion control), all subsequent numbering of clauses and Syntax tables should be incremented by one.

```
assert_action_control_task ::=
assert_action_control_task [ ( levels [ , list_of_modules_or_assertions ] ) ] ;
assert_action_control_task ::=
$assertpasson
| $assertpassoff
| $assertfailon
| $assertfailoff
| $assertnonvacuouson
| $assertvacuousoff
list_of_modules_or_assertions ::=
module_or_assertion { , module_or_assertion }
module_or_assertion ::=
module_identifier
| assertion_identifier
| hierarchical_identifier
```

Syntax 19.11 —Assertion action control syntax (not in Annex A)

SystemVerilog provides six system tasks to control execution of assertion action blocks for concurrent assertions:

— *$assertpassoff* shall stop execution of the pass action for vacuous and nonvacuous success of all the specified assertions. Execution of the pass action for both vacuous and nonvacuous successes can be re-enabled subsequently by $assertpasson, while the execution of pass action for only nonvacuous successes can be enabled subsequently by $assertnonvacuouson. An assertion that is already executing, including execution of the pass or fail action, is not affected. By default, the pass action is executed.

— `$assertfailoff` shall stop execution of the fail action of all the specified assertions until a subsequent `$assertfailon`. An assertion that is already executing, including execution of the pass or fail action, is not affected. By default, the fail action is executed. This task also affects the execution of default fail action block.

— `$assertvacuousoff` shall stop execution of the pass action of all the specified assertions on vacuous success until a subsequent `$assertpasson`. An assertion that is already executing, including execution of the pass or fail action, is not affected. By default, the pass action is executed on vacuous success. Refer to section 16.14.6 for the definition of vacuous success.

— `$assertpasson` shall enable execution of the pass action for vacuous and nonvacous success of all the specified assertions. An assertion that is already executing, including execution of the pass or fail action, is not affected.

— `$assertfailon` shall enable execution of the fail action of all the specified assertions. An assertion that is already executing, including execution of the pass or fail action, is not affected. This task also affects the execution of the default fail action block.
— `$assertnonvacuouson` shall enable execution of the pass action of all the specified assertions on nonvacuous success. An assertion that is already executing, including execution of the pass or fail action, is not affected. Refer to section 16.14.6 for the definition of vacuous success.

When invoked with no arguments, the system task shall apply to all the assertions. When the system task is specified with arguments, the first argument indicates levels of the hierarchy, consistent with the corresponding argument to the Verilog $dumpvars system task. Subsequent arguments specify which scopes of the model to control. These arguments can specify entire scopes (module, program, interface, always block or initial block) or individual assertions. Please refer to 20.7.1.2 for the definition of $dumpvars.

These system tasks shall not affect the execution of pass or fail actions until the system task is executed. These system tasks shall not affect the statistics counters for the assertions.

Clause 38.4.2, Add the following:

- — `cbAssertionKill`. An attempt is killed (e.g., as a result of a control action).
- — `cbAssertionDisablePassAction`. The pass action is disabled for vacuous and nonvacuous success for the assertion (e.g., as a result of control action).
- — `cbAssertionEnablePassAction`. The pass action is enabled for vacuous and nonvacuous success for the assertion (e.g., as a result of control action).
- — `cbAssertionDisableFailAction`. The fail action is disabled for the assertion (e.g., as a result of control action).
- — `cbAssertionDisableVacuousAction`. The pass action is disabled for vacuous success of the assertion (e.g., as a result of control action).
- — `cbAssertionEnableNonvacuousAction`. The pass action is enabled for nonvacuous success of the assertion (e.g., as a result of control action).

— On disable, enable, reset, ~~and~~ kill, pass action, fail action, vacuous action and nonvacuous action callbacks, the returned `p_vpi_attempt_info` info pointer is `NULL`, and no attempt information is available.

Clause 38.5.2, change the following on page 920:

- — Usage example: `vpi_control(vpiAssertionEnable, assertionHandle)`
  - — `vpiAssertionEnable` enables starting new attempts for this assertion. This has no effect on any existing attempts or if the assertion is already enabled.

- Usage example: `vpi_control(vpiAssertionDisablePassAction, assertionHandle)`
  - `vpiAssertionDisablePassAction` disables execution of pass action for vacuous and nonvacuous success of this assertion. This has no effect on any existing attempts or if the assertion pass action is already disabled. By default, all pass actions are enabled.

- Usage example: vpi_`control(vpiAssertionEnablePassAction, assertionHandle)`
  - `vpiAssertionEnablePassAction` enables execution of pass action for vacuous and nonvacuous success of this assertion. This has no effect on any existing attempts or if the assertion pass action is already enabled.

- Usage example: `vpi_control(vpiAssertionDisableFailAction, assertionHandle)`
  - `vpiAssertionDisableFailAction` disables execution of fail action for this assertion. This has no effect on any existing attempts or if the assertion fail action is already disabled. By default, all fail actions are enabled.

- Usage example: `vpi_control(vpiAssertionEnableFailAction, assertionHandle)`
  - `vpiAssertionEnableFailAction` enables execution of fail action for this assertion. This has no effect on any existing attempts or if the assertion fail action is already enabled.

- Usage example: `vpi_control( vpiAssertionDisableVacuousAction, assertionHandle)`
  - `vpiAssertionDisableVacuousAction` disables execution of pass action on vacuous success of this assertion. This has no effect on any existing attempts or if the execution of pass action on vacuous success is already disabled. By default, all vacuous actions are enabled.

- Usage example: `vpi_control(vpiAssertionEnableNonvacuousAction, assertionHandle)`
  - `vpiAssertionEnableNonvacuousAction` enables execution of pass action on nonvacuous success of this assertion. This has no effect on any existing attempts or if the pass action is already enabled for nonvacuous success of this assertion.

Annex M, change the following on page 1107:

Note to Editor: Fill-in appropriate (non-overlapping) numbers for added VPI properties

```
/* assertion control constants */
#define vpiAssertionDisable 620
#define vpiAssertionEnable 621
```

```
#define vpiAssertionReset 622
#define vpiAssertionKill 623
#define vpiAssertionEnableStep 624
#define vpiAssertionDisableStep 625
#define vpiAssertionClockSteps 626
#define vpiAssertionSysOn 627
#define vpiAssertionSysOff 628
#define vpiAssertionSysKill 632
#define vpiAssertionSysEnd 629
#define vpiAssertionSysReset 630
#define vpiAssertionDisablePassAction <editor to fill-in>
#define vpiAssertionEnablePassAction <editor to fill-in>
#define vpiAssertionDisableFailAction <editor to fill-in>
#define vpiAssertionEnableFailAction <editor to fill-in>
#define vpiAssertionDisableVacuousAction <editor to fill-in>
#define vpiAssertionEnableNonvacuousAction <editor to fill-in>
```