## Clause: 17.2

### Description

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

### LRM Changes

Clause 17.13.1, change the following on page 274:

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. Otherwise, the 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 22.9.

Clause 17.16, change the following on page 287:

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 22.9.

Clause 17.13.3, change the following on page 276:

In addition, *statement_or_null* is executed every time a property succeeds. The execution of *statement_or_null* can be controlled by using assertion action control tasks. The assertion action control tasks are described in 22.9.

Add the following new section:

### 22.9 Assertion action control system tasks

**Note to Editor: This is a new section following 22.8, 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
| $assertvacuouson
| $assertvacuousoff
list_of_modules_or_assertions ::=
module_or_assertion { , module_or_assertion }
module_or_assertion ::=
module_identifier
| assertion_identifier
| hierarchical_identifier
```

*Syntax 22-7—Assertion pass action control syntax (not in Annex A)*

SystemVerilog provides eight system tasks to control execution of assertion action block for concurrent assertions:
— `$assertpassoff` shall stop execution of pass action of all the specified assertions until a subsequent `$assertpasson`. An assertion that is already executing, including execution of the pass or fail statement, is not affected. By default, pass action is executed.
— `$assertfailoff` shall stop execution of fail action of all the specified assertions until a subsequent `$assertfailon`. An assertion that is already executing, including execution of the pass or fail statement, is not affected. By default, fail action is executed
— `$assertvacuousoff` shall stop execution of pass action of all the specified assertions on vacuous success until a subsequent `$assertvacuouson`. An assertion that is already executing, including execution of the pass or fail statement, is not affected. By default, pass action is executed on vacuous success. Refer to section 17.13.6 for the definition of vacuous success.
— `$assertpasson` shall enable execution of pass action of all the specified assertions. An assertion that is already executing, including execution of the pass or fail statement, is not affected.
— `$assertfailon` shall enable execution of fail action of all the specified assertions. An assertion that is already executing, including execution of the pass or fail statement, is not affected.
— `$assertvacuouson` shall enable execution of pass action of all the specified assertions on vacuous success. An assertion that is already executing, including execution of the pass or fail statement, is not affected. Refer to section 17.13.6 for the definition of vacuous success.

When invoked with no arguments, the system task shall apply to all assertions. When the 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.

These system tasks shall not affect the execution of pass or fail statements until the task is executed.

Clause 28.4.2, change the following on page 471:

— 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 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 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 vacuous action for this assertion. This has no effect on any existing attempts or if the assertion vacuous action is already disabled. By default, all vacuous actions are enabled.

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

Annex I, change the following on page 627:

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 vpiAssertionEnableVacuousAction <editor to fill-in>
```

Clause 28.3.2, add the following:

- cbAssertionKill. An attempt is killed (e.g., as a result of a control action).
- cbAssertionDisablePassAction. The pass action is disabled for the assertion (e.g., as a result of control action).
- cbAssertionEnablePassAction. The pass action is enabled 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).
- cbAssertionEnableFailAction. The fail action is enabled for the assertion (e.g., as a result of control action).
- cbAssertionDisableVacuousAction. The vacuous action is disabled for the assertion (e.g., as a result of control action).
- cbAssertionEnableVacuousAction. The vacuous action is enabled for the assertion (e.g., as a result of control action).

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