Section: 29.2.3 Callbacks

Change:

2) “Assertion system”

```
cbAssertionSysInitialized
cbAssertionSysStart
cbAssertionSysStop
cbAssertionSysEnd
cbAssertionSysReset
```

To:

2) “Assertion system”

```
cbAssertionSysInitialized
cbAssertionSysStartOn
cbAssertionSysStopOff
cbAssertionSysKill
cbAssertionSysEnd
cbAssertionSysReset
```

Section: 29.2.4 Control constants

Change:

This subclause lists the system control constant callbacks. Refer to 29.5 for details on how to use `vpi_control()` with these constants to obtain assertion control information, and control the assertion system.

1) Assertion

```
vpiAssertionDisable
vpiAssertionEnable
vpiAssertionReset
vpiAssertionKill
vpiAssertionEnableStep
vpiAssertionDisableStep
```

2) Assertion stepping

```
vpiAssertionClockSteps
```

3) “Assertion system”

```
vpiAssertionSysStart
vpiAssertionSysStop
vpiAssertionSysEnd
vpiAssertionSysReset
```

To:

This subclause lists the system control constants ~~callbacks~~. Refer to 29.5 for details on how to use `vpi_control()` with these constants to obtain assertion control information, and control the assertion system.

1) Assertion

```
vpiAssertionDisable
vpiAssertionEnable
vpiAssertionReset
vpiAssertionKill
vpiAssertionEnableStep
vpiAssertionDisableStep
```

2) Assertion stepping

```
vpiAssertionClockSteps
```

3) “Assertion system”

```
vpiAssertionSysStartOn
```

vpiAssertionSys~~Stop~~Off
vpiAssertionSysKill
vpiAssertionSysEnd
vpiAssertionSysReset

Section: 29.4.1 Placing assertion system callbacks

Change:

Use vpi_register_cb(), setting the cb_rtn element to the function to be invoked and the reason element of the s_cb_data structure to one of the following values, to place an assertion system callback.

cbAssertionSysInitialized
occurs after the system has initialized. No assertion-specific actions can be performed until this callback completes. The assertion system can initialize before cbStartOfSimulation does or afterwards.

cbAssertionSysStart
the assertion system has become active and starts processing assertion attempts. This always occur after cbAssertionSysInitialized. By default, the assertion system is "started" on simulation startup, but the user can delay this by using assertion system control actions.

cbAssertionSysStop
the assertion system has been temporarily suspended. While stopped no assertion attempts are processed and no assertion-related callbacks occur. The assertion system can be stopped and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysEnd
occurs when all assertions have completed and no new attempts shall start. Once this callback occurs no more assertion-related callbacks shall occur and assertion-related actions shall have no further effect. This typically occurs after the end of simulation.

cbAssertionSysReset
occurs when the assertion system is reset, e.g., due to a system control action.

To:

Use vpi_register_cb(), setting the cb_rtn element to the function to be invoked and the reason element of the s_cb_data structure to one of the following values, to place an assertion system callback.

cbAssertionSysInitialized
occurs after the system has initialized. No assertion-specific actions can be performed until this callback completes. The assertion system can initialize before cbStartOfSimulation does or afterwards.

cbAssertionSys~~Start~~On
the assertion system has become active and starts processing assertion attempts. This always occur after cbAssertionSysInitialized. By default, the assertion system is "started" on simulation startup, but the user can delay this by using assertion system control actions.

cbAssertionSys~~Stop~~Off
the assertion system has been temporarily stopped. While stopped no new assertion attempts are processed and no new assertion-related callbacks occur. Assertions already executing are not affected. The assertion system can be stopped and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysKill
the assertion system has been temporarily suspended. While suspended no assertion attempts are processed and no assertion-related callbacks occur. The assertion system can be suspended and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysEnd
occurs when all assertions have completed and no new attempts shall start. Once this callback occurs no more assertion-related callbacks shall occur and assertion-related actions shall have no further effect. This typically occurs after the end of simulation.

cbAssertionSysReset
occurs when the assertion system is reset, e.g., due to a system control action.

Section: 29.5.1 Assertion system control

Change:

Use `vpi_control()`, with one of the following operators and no other arguments, to obtain assertion system control information.

Usage example: `vpi_control(vpiAssertionSysReset)`

`vpiAssertionSysReset`

discards all attempts in progress for all assertions and restore the entire assertion system to its initial state.Any pre-existing `vpiAssertionStepSuccess` and `vpiAssertionStepFailure` callbacks shall be removed; all other assertion callbacks shall remain.

Usage example: `vpi_control(vpiAssertionSysStop)`

`vpiAssertionSysStop`

considers all attempts in progress as unterminated and disable any further assertions from being started. This control has no effect on pre-existing assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysStart)`

`vpiAssertionSysStart`

restarts the assertion system after it was stopped (e.g., due to `vpiAssertionSysStop`). Once started, attempts shall resume on all assertions. This control has no effect on prior assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysEnd)`

`vpiAssertionSysEnd`

discard all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Note that once this control is issued, no further assertion related actions shall be permitted.

To:

Use `vpi_control()`, with one of the following ~~operator~~constants and ~~no other arguments~~a second handle argument which is either a `vpiHandle` for a scope or a `vpiCollection` of handles for a list of scopes, to ~~obtain~~ control the assertion system ~~control information~~. A `NULL` handle signifies that the control applies to all assertions regardless of scope.

Usage example: `vpi_control(vpiAssertionSysReset, handle)`

`vpiAssertionSysReset`

discards all attempts in progress for all assertions and restore the entire assertion system to its initial state.Any pre-existing `vpiAssertionStepSuccess` and `vpiAssertionStepFailure` callbacks shall be removed; all other assertion callbacks shall remain.

Usage example: `vpi_control(vpiAssertionSys~~Stop~~Off, handle)`

`vpiAssertionSys~~Stop~~Off`

~~considers all attempts in progress as unterminated and~~ disables any further assertions from being started. Assertions already executing are not affected. This control has no effect on pre-existing assertion callbacks.

`vpiAssertionSysKill`

considers all attempts in progress as unterminated and disables any further assertions from being started. This control has no effect on pre-existing assertion callbacks.

Usage example: `vpi_control(vpiAssertionSys~~Start~~On, handle)`

`vpiAssertionSys~~Start~~On`

restarts the assertion system after it was stopped or suspended (e.g., due to `vpiAssertionSys~~Stop~~Off` or `vpiAssertionSysKill`). Once started, attempts shall resume on all assertions. This control has no effect on prior assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysEnd, handle)`

`vpiAssertionSysEnd`

discards all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Note that once this control is issued, no further assertion related actions shall be permitted.

Section: Annex I

Change:

```
#define cbAssertionStart 606
#define cbAssertionSuccess 607
#define cbAssertionFailure 608
#define cbAssertionStepSuccess 609
#define cbAssertionStepFailure 610
#define cbAssertionDisable 611
#define cbAssertionEnable 612
#define cbAssertionReset 613
#define cbAssertionKill 614
/* assertion "system" callback types */
#define cbAssertionSysInitialized 615
#define cbAssertionSysStart 616
#define cbAssertionSysStop 617
#define cbAssertionSysEnd 618
#define cbAssertionSysReset 619
/* 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 vpiAssertionSysStart 627
#define vpiAssertionSysStop 628
#define vpiAssertionSysEnd 629
#define vpiAssertionSysReset 630
```

To:

```
#define cbAssertionStart 606
#define cbAssertionSuccess 607
#define cbAssertionFailure 608
#define cbAssertionStepSuccess 609
#define cbAssertionStepFailure 610
#define cbAssertionDisable 611
#define cbAssertionEnable 612
#define cbAssertionReset 613
#define cbAssertionKill 614
/* assertion "system" callback types */
#define cbAssertionSysInitialized 615
#define cbAssertionSysStartOn 616
#define cbAssertionSysStopOff 617
#define cbAssertionSysKill       631
#define cbAssertionSysEnd 618
#define cbAssertionSysReset 619
/* 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 vpiAssertionSysStartOn 627
#define vpiAssertionSysStopOff 628
#define vpiAssertionSysKill      632
#define vpiAssertionSysEnd 629
#define vpiAssertionSysReset 630
```