The purpose of this proposal is to clarify where concurrent and immediate assertions may appear and what kind of variables they may use.

Note: referenced text assumes 1995 – assertion in loops and 1729 immediate assume and cover

NOTE TO SV-AC: The assertion action controls of section 19.11 state that they apply only to concurrent assertions. However, in the VPI section, they appear to apply both to concurrent and immediate assertions. Immediate assertions do not have “vacuous” which is probably why they were targeted for concurrent assertions, but I think they should apply to both. I could fix that here if desired since I would need to add the note to them if they apply to immediates.

==========

REPLACE IN 16.4 CONCURRENT ASSERTIONS OVERVIEW

Concurrent assertions describe behavior that spans over time. Unlike immediate assertions, the evaluation model is based on a clock so that a concurrent assertion is evaluated only at the occurrence of a clock tick. The values of variables used in the evaluation (except for the special case of loop iterators as described in 16.14.5) are the sampled values. This way, a predictable result can be obtained from the evaluation, regardless of the simulator’s internal mechanism of ordering events and evaluating events. This model of execution also corresponds to the synthesis model of hardware interpretation from an register transfer language (RTL) description.

The values of variables used in assertions are sampled in the Preponed region of a time slot, and the assertions are evaluated during the Observe region.

WITH

Concurrent assertions describe behavior that spans over time. Unlike immediate assertions, the evaluation model is based on a clock so that a concurrent assertion is evaluated only at the occurrence of a clock tick. The values of variables used in the evaluation (except for the special case of loop iterators as described in 16.14.5) are the sampled values. This way, a predictable result can be obtained from the evaluation, regardless of the simulator’s internal mechanism of ordering events and evaluating events. This model of execution also corresponds to the synthesis model of hardware interpretation from an register transfer language (RTL) description.

All data referenced in a concurrent assertion, with the exception of local variables (see 16.9) and **for (see 12.7.1.)** and **foreach (see 12.7.3)** loop indices, must have a static lifetime (exist for the whole elaboration and simulation time). Similarly, concurrent assertions may only exist in blocks whose lifetime is also static. So for example,

- automatic variables and members or elements of dynamic variables cannot be referenced in an assertion. This includes class properties, dynamically sized variables, data in automatic tasks, functions, or blocks.

- Class methods (see Clause 8) are only active for the lifetime of the call and therefore may not contain concurrent assertions or have its elements referenced by a concurrent assertion.
- The lifetime of a **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** block is limited to the execution of all processes spawned by the block, and the lifetime of a scope enclosing any fork block includes the lifetime of the fork block. **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** blocks therefore may not contain concurrent assertions or have its elements referenced by a concurrent assertion.

All variables in an assertion use the value sampled in the Preponed region of a timeslot with the exception that local variables and **for** and **foreach** loop variables use the current value. ~~The values of variables used in assertions are sampled in the Preponed region of a time slot, and the~~ The assertions are evaluated during the Observe region.

REPLACE in 16.3 Immediate assertions (assumes changes from 1729)

The *immediate_assertion_statement* is a *statement_item* and can be specified anywhere a procedural statement is specified, including automatic tasks, functions, and blocks

WITH

The *immediate_assertion_statement* is a *statement_item* and can be specified anywhere a procedural statement is specified. While immediate **assume** and immediate **assert** statements can be included in automatic tasks, functions, and blocks, and in **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** blocks, they cannot be controlled by system tasks, action block controls, or VPI. An immediate **cover** statement cannot be used in automatic tasks, functions, and blocks, or in a **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** block.

REPLACE IN 19.10 Assertion control tasks

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

WITH

System tasks apply to all assertions with the exception of immediate assertions in automatic tasks, functions, and blocks and immediate assertions in a **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** block. When invoked with no arguments, the system task shall apply to all static assertions.

REPLACE IN 38.4.2 Placing assertion callbacks

These callbacks are specific to a given assertion;

WITH

Callbacks apply to all assertions with the exception of immediate assertions in automatic tasks, functions, and blocks and to immediate assertions in a **fork**…**join**, **fork**…**join_any**, or **fork**…**join_none** block.

These callbacks are specific to a given assertion;

REPLACE IN 38.5.2 Assertion control

To obtain assertion control information, use **vpi_control()** with one of the operators in this subclause.

WITH

Assertion control applies to all assertions with the exception of immediate assertions in automatic tasks, functions, and blocks and to immediate assertions in a **`fork`**…**`join`**, **`fork`**…**`join_any`**, or **`fork`**…**`join_none`** block. To obtain assertion control information, use **vpi_control()** with one of the operators in this subclause.

REPLACE in 39.5.3

— For assertion handle, the coverable entities are assertions.

WITH

— For assertion handle, the coverable entities are assertions that are not in automatic tasks, functions and blocks and for immediate assertions in a **`fork`**…**`join`**, **`fork`**…**`join_any`**, or **`fork`**…**`join_none`** block.