Mantis 2089: Allow checker construct to include final blocks with immediate assertions

Motivation

Mantis 1900 added a new construct called a checker to encapsulate related verification constructs like assertions into one entity. This proposal extends the checker to allow **final** procedures to be included. These final procedures can be used to check status or print out statistics at the end of the simulation. This is required to make the checker more usable in verification libraries.

Note: Some changes to VPI diagrams may be required.

Note to editor: This proposal relies on the following other language-extending Mantis items:

- 1900: checker

Note to editor: Mantis 2088 will touch similar areas. The proposal is written without showing the 2088 changes.

16.18.1 Overview

Note to editor: from Mantis 1900 Text

REPLACE

The checkers may contain concurrent assertions, free variable declarations and assignments, structural procedures, function declarations, **`let`** declarations, sequences and properties, and generate regions. The following procedures are allowed in a checker (see 16.18.4):

— **`initial_check`** procedure, encapsulating assertions monitored on the first clock tick (similar to 16.14.5).

— **`always_check`** procedure, encapsulating concurrent assertions and non-blocking assignments of free variables.

WITH

The checkers may contain concurrent assertions, free variable declarations and assignments, structural procedures, function declarations, **`let`** declarations, sequences and properties, and generate regions. The following procedures are allowed in a checker (see 16.18.4):

— **`initial_check`** procedure, encapsulating assertions monitored on the first clock tick (similar to 16.14.5).

— **`always_check`** procedure, encapsulating concurrent assertions and non-blocking assignments of free variables.

— **`final`** procedure, encapsulating immediate assertions or display statements, for reporting status at the end of simulation.

16.18.2 Checker declaration

Mantis 1900

REPLACE

checker_or_generate_item ::=
{ attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct[5]
| always_construct[2]
| concurrent_assertion_item
| checker_generate_item

WITH

checker_or_generate_item ::=
{ attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct[5]
| always_construct[2]
| final_construct
| concurrent_assertion_item
| checker_generate_item

REPLACE

Action blocks of assertions within a checker will be referred to as *checker action blocks*, and the rest of the checker will be referred to as *checker body*.

A checker body may contain the following elements:

— Declaration of **`let`**, sequences, properties and functions.

— Concurrent assertions.

— Free variables and their assignments (see 16.18.5).

— Default clocking and disable declarations.

— **`initial_check`** and **`always_check`** procedures (see 16.18.4).

— Generate blocks, containing any of the above elements.

Checker action blocks shall not write into free variables, but they may contain any other code which is normally allowed in action blocks in modules.

WITH

Action blocks of assertions within a checker will be referred to as *checker action blocks*, and the rest of the checker, with the exception of any final block code, will be referred to as *checker body*.

A checker body may contain the following elements:

— Declaration of **`let`**, sequences, properties and functions.

— Concurrent assertions.

— Free variables and their assignments (see 16.18.5).

— Default clocking and disable declarations.

— **initial_check**, ~~and~~ **always_check**, and **final** procedures (see 16.18.4).

— Generate blocks, containing any of the above elements.

Checker action blocks or final procedures shall not write into free variables, but they may contain any other code which is normally allowed in action blocks or final procedures in modules.

16.18.4 Checker procedures Mantis 1900

REPLACE

The following procedures are allowed inside a checker body:

— **initial_check** procedure, and

— **always_check** procedure

An **initial_check** procedure may contain concurrent assertions and event controls only. As explained in 16.14.5, assertions inside an *initial* procedure are monitored only on the first clock tick, while in any other location they are always monitored (see 16.14.5). It shall be an error to specify an **initial_check** procedure outside of the checker body.

An **always_check** procedure may be specified in the checker body only and may contain concurrent assertions, non-blocking free variable assignments (see 16.18.5.1) and event control statements. All other statements shall not appear inside an **always_check** procedure.

WITH

The following procedures are allowed inside a checker body:

— **initial_check** procedure, and

— **always_check** procedure

— **final** procedure

An **initial_check** procedure may contain concurrent assertions and event controls only. As explained in 16.14.5, assertions inside an *initial* procedure are monitored only on the first clock tick, while in any other location they are always monitored (see 16.14.5). It shall be an error to specify an **initial_check** procedure outside of the checker body.

An **always_check** procedure may be specified in the checker body only and may contain concurrent assertions, non-blocking free variable assignments (see 16.18.5.1) and event control statements. All other statements shall not appear inside an **always_check** procedure.

A **final** procedure may be specified within a checker in the same manner as in a module (see 9.2.3). This allows for the checker to check conditions with immediate assertions or print out statistics at the end of simulation. A final block within a checker may include any construct which is allowed in a non-checker final block. Any variables defined within **final** procedures obey all the normal scope rules (see 22.8). However, statements within **final** procedures shall not write into free variables.

A.1.8 Checker items

Mantis 1900

REPLACE

checker_or_generate_item ::=
{ attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct[5]
| always_construct[2]
| concurrent_assertion_item
| checker_generate_item

WITH

checker_or_generate_item ::=
{ attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct[5]
| always_construct[2]
| final_construct
| concurrent_assertion_item
| checker_generate_item