

## Mantis 2089: Allow checker construct to include final procedures 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.

### Change Log

2/25/2008: Added text to clarify that the operation of final procedures does not depend on the instantiation context. Note that Mantis 2110 (not approved yet) would allow checkers within procedural loops. This Mantis item contains additional clarifications about the operation of procedures within the checker in this context.

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

- 1900: checker
- 2182: VPI Diagrams for checkers

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, display statements, or other procedural code, for reporting status at the end of simulation or doing any other end-of-simulation processing.

## 16.18.2 Checker declaration

Mantis 1900

REPLACE

```
checker_or_generate_item ::=
    { attribute_instance } continuous_assign
  | checker_or_generate_item_declaration
  | initial_construct5
  | always_construct2
  | concurrent_assertion_item
  | checker_generate_item
```

WITH

```
checker_or_generate_item ::=
    { attribute_instance } continuous_assign
  | checker_or_generate_item_declaration
  | initial_construct5
  | always_construct2
  | 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.

Checkers may assign values to their formal arguments, treating them as output arguments, though no explicit notation of this is required in the checker declaration statement. Each formal argument used in this way may be assigned a value either in a checker body or in checker action blocks. If a formal argument is written in the checker body, its corresponding actual argument shall be a checker variable or a formal argument in another

checker. If a formal argument is assigned a value in a checker, it shall be untyped. The output actual argument shall have a static lifetime.

WITH

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**, 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 each of these constructs they may contain any other code which is normally allowed when that construct appears in a module ~~in action blocks in modules~~.

Checkers may assign values to their formal arguments, treating them as output arguments, though no explicit notation of this is required in the checker declaration statement. Each formal argument used in this way may be assigned a value either in a checker body or in checker action blocks. However, **final procedures are not allowed to write to the checker formal arguments in this way**. If a formal argument is written in the checker body, its corresponding actual argument shall be a checker variable or a formal argument in another checker. If a formal argument is assigned a value in a checker, it shall be untyped. The output actual argument shall have a static lifetime.

### 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. The operation of the final procedure is independent of the instantiation context of the checker that contains it. It will be executed once at the end of simulation for every unique instantiation statement of that checker. There is no implied ordering in the execution of multiple final blocks. A final procedure within a checker may include any construct which is allowed in a non-checker final procedure. Checker variable declarations are not allowed. However declarations of other variable types, which are not allowed within a checker body, are allowed, and code within a final procedure is allowed to reference (but not write to) any visible static variable, including checker variables declared in the checker body which contains the procedure.

REPLACE

### 16.18.5 Checker variables

Any variable declared within a checker preceded by the keyword `checkvar` is a *checker variable*, a variable object introduced into SystemVerilog to support modeling for assertions and for formal verification. Individual bits of a checker variable are referred to as *check bits*. A declaration without `checkvar` in a checker body or a declaration of a checker variable outside of it shall be illegal, for example:

```

module m;
  // Illegal: checker variables may be declared in a checker body only
  checkvar bit a;
  ...
endmodule : m
checker c;
  // Illegal: non-checker variables shall not be declared in a checker body
  bit a;
  ...
endchecker : c

```

WITH

### 16.18.5 Checker variables

Any variable declared within a checker preceded by the keyword `checkvar` is a *checker variable*, a variable object introduced into SystemVerilog to support modeling for assertions and for formal verification. Individual bits of a checker variable are referred to as *check bits*. A declaration without `checkvar` in a checker body or a declaration of a checker variable outside of it shall be illegal, for example:

```

module m;
  // Illegal: checker variables may be declared in a checker body only

```

```

    checkvar bit a;
    ...
endmodule : m
checker c;
// Illegal: non-checker variables shall not be declared in a checker body
bit a;
...
final begin
// Illegal: checkvar variables are not allowed in final procedures
checkvar bit total2;

// Legal: inside a final procedure
bit total3;
...
end
endchecker : c

```

### A.1.8 Checker items

Mantis 1900

REPLACE

```

checker_or_generate_item ::=
    { attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct5
| always_construct2
| concurrent_assertion_item
| checker_generate_item

```

WITH

```

checker_or_generate_item ::=
    { attribute_instance } continuous_assign
| checker_or_generate_item_declaration
| initial_construct5
| always_construct2
| final_construct
| concurrent_assertion_item
| checker_generate_item

```