

## Proposal for Mantis 1668: Local Variable Initializers

NOTE: All LRM references are to P1800-2008-draft3aPROTECTED.pdf.

The goal of this proposal is to allow a declaration assignment to be written as part of the declaration of a local variable in either a sequence or a property declaration. The intended semantics is that declaration assignments be performed in the order of declaration at the beginning of each evaluation attempt of an instance of the named sequence or property.

The formal semantics of local variable declaration assignments and proposed changes to Annex F are described in a separate document.

Some care is needed in determining when the local variable declaration assignments should be performed if the clock is changing as the evaluation proceeds to the instance of the named sequence or property in which the declaration assignments appear. For consistency with the singly-clocked cases, the local variable declaration assignments must be performed after aligning to the new clock. For a multiply-clocked sequence, the LRM currently requires a unique semantic leading clock, and this is the clock to which the local variable declaration assignments should be aligned. For a multiply-clocked property, there can be more than one semantic leading clock. Therefore, there is a formal definition of a recursive function *push* that attaches the local variable declaration assignments to each of the subproperties that might be differently clocked. The details are defined in the separate document that describes proposed changes to Annex F.

REMARK: In the future, if the restriction that multiply-clocked sequences have a unique semantic leading clock is relaxed, then a similar recursive function can be used to attach the local variable declaration assignments to the beginnings of the various differently-clocked subsequences.

In the proposed Syntax 16-13, the `subroutine_call` option for `sequence_match_item` is replaced by an ellipsis because that feature is not discussed in this section.

This proposal clarifies, in alignment with Mantis 1704, that a local variable assignment cannot be attached to a sequence that admits an empty match.

The description of the declaration assignments involves expansion of the initial description of local variables. Parts of the text of 16.9 have been rewritten and expanded to improve clarity. For example, there was before no clear description of when a local variable can be referenced and the relationship between a local variable's "not flowing" and its "becoming unassigned".

### Syntax 16-4, p. 315. CHANGE

```
assertion_variable_declaration ::=  
  var_data_type list_of_variable_identifiers ;
```

TO

```
assertion_variable_declaration ::=  
  var_data_type list_of_variable_identifiers list_of_variable_decl_assignments ;
```

## 16.9, p. 335: REPLACE

The dynamic creation of a variable and its assignment is achieved by using the local variable declaration in a sequence or property declaration and making an assignment in the sequence.

```
[  
  sequence_expr ::=  
    ...  
    | ( sequence_expr { , sequence_match_item } ) [ sequence_abbrev ]  
    ...  
  sequence_match_item ::=  
    operator_assignment  
    | inc_or_dec_expression  
    | subroutine_call  
]  
]
```

Syntax 16-12 -- Variable assignment syntax (excerpt from Annex A)

The type of variable is explicitly specified. The variable can be assigned at the end point of any syntactic subsequence by placing the subsequence, comma separated from the sampling assignment, in parentheses. For example, if in

```
a ##1 b[->1] ##1 c[*2]
```

it is desired to assign  $x = e$  at the match of  $b[->1]$ , the sequence can be rewritten as

```
a ##1 (b[->1], x = e) ##1 c[*2]
```

The local variable can be reassigned later in the sequence, as in

```
a ##1 (b[->1], x = e) ##1 (c[*2], x = x + 1)
```

For every attempt, a new copy of the variable is created for the sequence. The variable value can be tested like any other SystemVerilog variable.

Hierarchical references to a local variable are not allowed.

## WITH

The dynamic creation of a local variable is achieved by using ~~the local variable declaration in a sequence or property declaration and making an assignment in the sequence~~ an assertion variable declaration within the declaration of a named sequence or property (see 16.12).

```
[
  assertion_variable_declaration ::=
    var_data_type list_of_variable_decl_assignments ;
```

```
]
Syntax 16-12 -- Assertion variable declaration syntax (excerpt from Annex A)
[Note to readers and the editor: The original Syntax 16-12 appears below, slightly
modified, as Syntax 16-13.]
```

The data type of an assertion variable declaration ~~is explicitly specified~~ shall be specified explicitly. The data type shall be one of the types allowed within assertions as defined in Subclause 16.5.1. The data type shall be followed by a comma-separated list of one or more identifiers with optional declaration assignments. A declaration assignment, if present, defines the initial value to be placed in the corresponding local variable. The initial value is defined by an expression, which need not be constant.

At the beginning of each evaluation attempt of an instance of a named sequence or property, a new copy of each of its local variables shall be created and, if present, the corresponding declaration assignment shall be performed. Declaration assignments shall be performed in the order that they appear in the sequence or property declaration. Non-local variables appearing in the expression of a declaration assignment to a local variable shall be evaluated using the Preponed values from the time slot in which the evaluation attempt begins. The expression of a declaration assignment to a given local variable may refer to a previously declared local variable. In this case the previously declared local variable shall itself have a declaration assignment, and the initial value assigned to the previously declared local variable shall be used in the evaluation of the expression assigned to the given local variable. Local variables do not have default initial values. A local variable without a declaration assignment shall be unassigned at the beginning of the evaluation attempt.

For example, at the beginning of an evaluation attempt of an instance of

```
sequence s ;
  logic u, v = a, w = v || b ;
  ...
endsequence
```

the assignment of a to v is performed first and the assignment of v || b to w is performed second. The value assigned to w is the same as that which would result from the declaration assignment w = a || b. The local variable u is unassigned at the beginning of the evaluation attempt.

Local variables may be assigned and re-assigned within the body of the sequence or property in which they are declared.

```
[
sequence_expr ::=
...
| ( sequence_expr { , sequence_match_item } ) [ sequence_abbrev ]
...
sequence_match_item ::=
operator_assignment
| inc_or_dec_expression
...
subroutine_call
]
```

Syntax 16-~~12~~13 -- Variable assignment syntax (excerpt from Annex A)

[Note to editor: all subsequent syntax box numbers in Clause 16 must be adjusted.]

~~The variable can~~ One or more local variables may be assigned at the end point of **any** a syntactic subsequence by placing the subsequence, comma separated from the ~~sampling-assignment~~ list of local variable assignments, in parentheses. At the end of any non-empty match of the subsequence, the local variable assignments are performed in the order that they appear in the list. For example, if in

```
a ##1 b[->1] ##1 c[*2]
```

it is desired to assign  $x = e$  and then  $y = x \ \&\& \ f$  at the match of  $b[->1]$ , the sequence can be rewritten as

```
a ##1 (b[->1], x = e, y = x && f) ##1 c[*2]
```

~~The A~~ local variable ~~can~~ may be reassigned later in the sequence or property, as in

```
a ##1 (b[->1], x = e, y = x && f) ##1 (c[*2], x = x + 1 &=
g)
```

The subsequence to which a local variable assignment is attached shall not admit an empty match. For example, the sequence

```
a ##1 (b[*0:1], x = e) ##1 c[*2] // illegal
```

is illegal because the subsequence  $b[*0:1]$  can match the empty word. The sequence

```
(a ##1 b[*0:1], x = e) ##1 c[*2] // legal
```

is legal because the concatenated subsequence `a ##1 b[*0:1]` cannot match the empty word.

~~For every attempt, a new copy of the variable is created for the sequence. The variable value can be tested like any other SystemVerilog variable.~~ A local variable may be referenced within the sequence or property in which it is declared. The structure of the sequence or property shall guarantee that the local variable be assigned a value prior to the point at which the reference is made. The prior assignment may be a declaration assignment or an assignment attached to a subsequence. There is an implicit reference associated with the use of an `inc_or_dec_operator` or an assignment operator other than “=”. Therefore, a local variable shall be assigned a value prior to being updated with an `inc_or_dec_operator` or with an assignment operator other than “=”.

Under certain circumstances, a local variable that is assigned later becomes unassigned. If a local variable does not flow out of a subsequence (see below), then the local variable shall become unassigned at the end of that subsequence, regardless of whether it was assigned a value prior to that point. The local variable shall not be referenced after the point from which it does not flow until after it has again been assigned a value. See Annex F for precise conditions defining local variable flow.

Hierarchical references to a local variable are not allowed.

### 16.9, p. 336. CHANGE

Local variables ~~can~~ may be used in sequences or properties.

### 16.9, p. 336. REPLACE

Local variables can be written on repeated sequences and accomplish accumulation of values.

```
sequence rep_v;
  int x;
  `true,x = 0 ##0
  (!a [*0:$] ##1 a, x = x+data)[*4] ##1 b ##1 c &&
  (data_out == x);
endsequence
```

### WITH

Local ~~variables~~ variable assignments ~~can~~ may be ~~written on repeated sequences~~ attached to the operand sequence of a repetition and accomplish accumulation of values.

```
sequence rep_v;
  int x = 0;
  `true,x = 0 ##0
```

```

(!a [*0:$] ##1 a [->1], x =x+data += data) [*4] ##1 b
##1 c && (data_out == x);
endsequence

```

An accumulating local variable may be used to count the number of times a condition is repeated, as in the following example:

```

sequence count_a_cycles;
  int x;
  ($rose(a), x = 1)
  ##1 (a, x++) [*0:$]
  ##1 !a && (x <= MAX);
endsequence

```

**RATIONALE:** The original `rep_v` example showed how to work around lack of local variable declaration assignments and is awkward with the feature added. Also, the idiom `!a [*0:$]` `##1 a` is more compactly written as `a [->1]`. The second example shows a useful idiom and illustrates the use of an `inc_or_dec_expression` to update a local variable.

### 16.9, p. 336. REPLACE

The local variables declared in one sequence are not visible in the sequence where it gets instantiated.

#### WITH

The local variables declared ~~in one~~ within a sequence or property are not visible in the context where ~~it gets~~ the sequence or property is instantiated.

**RATIONALE:** The beginning of the proposal discusses local variables for both sequences and properties.

### 16.9, p. 338. CHANGE

A local variable may have been assigned a value before the start of the evaluation of the composite sequence. Such a local variable is said to flow in to each of the operand sequences.

#### TO

A local variable may have been assigned a value before the start of the evaluation of the composite sequence, either from an initial declaration assignment or from an

assignment attached to a preceding subsequence. Such a local variable is said to flow in to each of the operand sequences.

**RATIONALE:** To clarify that a declaration assignment counts as an assignment “before the start of evaluation of the composite sequence”, as suggested by Jiang Long.

### 16.9, p. 338. CHANGE

If these conditions are satisfied, then the local variable is said to flow out of the composite sequence. An intuitive description of the conditions for local variable flow follows:

#### TO

If these conditions are satisfied, then the local variable is said to flow out of the composite sequence. [Otherwise, the local variable shall become unassigned at the end of the composite sequence.](#) An intuitive description of the conditions for local variable flow follows:

**RATIONALE:** To complete the correlation of the concepts of “not flowing” and “becoming unassigned”.

### Syntax 16-14, p. 341. CHANGE

```
assertion_variable_declaration ::=  
  var_data_type list_of_variable_identifiers ;
```

#### TO

```
assertion_variable_declaration ::=  
  var_data_type list_of_variable_identifiers list_of_variable_decl_assignments ;
```

**In Subclause 16.13, ADD the following Subsubclause:**

#### [16.13.7 Local variable declaration assignments](#)

[For singly-clocked sequences and properties, a local variable declaration assignment for an evaluation attempt of an instance of a named sequence or property is performed when the evaluation attempt begins. Such an evaluation attempt always begins in a timestep in which there is a tick of the single governing clock.](#)

[For multiclock sequences and properties, a local variable declaration assignment for an evaluation attempt of an instance of a named sequence or property with a single semantic leading clock \(see 16.15.1\) shall be performed at the earliest tick of the semantic leading clock that is at](#)

or after the beginning of the evaluation attempt. If there are two or more distinct semantic leading clocks for an instance of a named property, then a separate copy of the local variable shall be created for each semantic leading clock. For each copy of the local variable, the declaration assignment shall be performed at the earliest tick of the corresponding semantic leading clock that is at or after the beginning of the evaluation attempt, and that copy of the local variable shall be used in the evaluation of the subproperty associated with the corresponding semantic leading clock.

For example, let

```

property p;
  logic v = e;
  (@(posedge clk1) (a == v)[*1:$] |-> b)
  and
  (@(posedge clk2) c[*1:$] |-> d == v)
  ;
endproperty

a1: assert property (@(posedge clk) f |=> p);

```

The instance of `p` in assertion `a1` has two semantic leading clocks, `posedge clk1` and `posedge clk2`. Therefore, there are separate copies of the local variable `v` for the two subproperties governed by these clocks. Let `t0` be a timestep in which `posedge clk` occurs and in which the sampled value of `f` is true. According to the structure of `a1`, an evaluation attempt of the instance of `p` starts strictly after `t0`. Let `t1` be the earliest timestep after `t0` in which `posedge clk1` occurs, and let `t2` be the earliest timestep after `t0` in which `posedge clk2` occurs. Then a declaration assignment `v = e` is performed in `t1`, and the value is assigned to the copy of `v` associated with `posedge clk1`. This value is used in the evaluation of the subproperty `(a == v)[*1:$] |-> b`. Similarly, a declaration assignment `v = e` is performed in `t2`, and the value is assigned to the copy of `v` associated with `posedge clk2`. This value is used in the evaluation of the subproperty `c[*1:$] |-> d == v`.

An equivalent declaration of `p` that does not use local variable declaration assignments is the following:

```

property p;
  logic v;
  (@(posedge clk1) (1, v = e) ##0 (a == v)[*1:$] |-> b)
  and
  (@(posedge clk2) (1, v = e) ##0 c[*1:$] |-> d == v)
  ;
endproperty

```

**A.2.10, p. 976. CHANGE**

assertion\_variable\_declaration ::=  
var\_data\_type list\_of\_variable\_identifiers ;

**TO**

assertion\_variable\_declaration ::=  
var\_data\_type ~~list\_of\_variable\_identifiers~~ list\_of\_variable\_decl\_assignments ;

**[Proposed changes to Annex F are described in a separate document.]**