

SystemVerilog Assertions Semantics

Draft, 10 April 2003

1 Abstract Syntax

1.1 Abstract grammars

In the following abstract grammars, b denotes a boolean expression, v denotes a local variable name, and e denotes an expression.

The abstract grammar for unlocked sequences is

```
 $R ::= b$  // “boolean expression” form
| ( 1,  $v = e$  ) // “local variable sampling” form
| (  $R$  ) // “parenthesis” form
| (  $R \## 1 R$  ) // “concatenation” form
| (  $R \## 0 R$  ) // “fusion” form
| (  $R$  or  $R$  ) // “or” form
| (  $R$  intersect  $R$  ) // “intersect” form
| first_match (  $R$  ) // “first match” form
|  $R$  [*0] // “null repetition” form
|  $R$  [*1:$] // “unbounded repetition” form
```

The abstract grammar for clocked sequences is

```
 $S ::= @(b) R$  // “clock” form
| (  $S \## S$  ) // “concatenation” form
```

The abstract grammar for unlocked properties is

```
 $P ::=$  [disable iff (  $b$  )] [not]  $R$  // “sequence” form
| [disable iff (  $b$  )] [not] (  $R \mid \rightarrow$  [not]  $R$  ) // “implication” form
```

The abstract grammar for clocked properties is

```
 $Q ::= @(b) P$  // “clock” form
| [disable iff (  $b$  )] [not]  $S$  // “sequence” form
| [disable iff (  $b$  )] [not] (  $S \mid \rightarrow$  [not]  $S$  ) // “implication” form
```

The abstract grammar for assertions is

```
 $A ::=$  always assert property  $Q$  // “always” form
| always @(b) assert property  $P$  // “always with clock” form
| initial assert property  $Q$  // “initial” form
| initial @(b) assert property  $P$  // “initial with clock” form
```

1.2 Notations

We will use the following auxiliary notions in defining the semantics.

- φ is an *unclocked property fragment* provided “disable iff (b) φ ” is an unclocked property.
- ν is a *negation specifier* if ν is either the empty token or “not”.

Throughout the sequel, we use the following notational conventions: b, c denote boolean expressions; v denotes a local variable name; e denotes an expression; φ denotes an unclocked property fragment; ν, ν_1, ν_2 denote negation specifiers; R, R_1, R_2 denote unclocked sequences; S, S_1, S_2 denote clocked sequences; P denotes an unclocked property; Q denotes a clocked property; A denotes an assertion; i, j, k, m, n denote non-negative integer constants.

1.3 Derived forms

Internal parentheses are omitted in compositions of the associative operators “##1” and “or”.

1.3.1 Derived non-overlapping implication operator

- $(R_1 \mid \Rightarrow \nu R_2) \equiv ((R_1 \text{ ##1 } 1) \mid \rightarrow \nu R_2) .$
- $(S_1 \mid \Rightarrow \nu S_2) \equiv ((S_1 \text{ ## } @ (1) 1) \mid \rightarrow \nu S_2) .$

1.3.2 Derived consecutive repetition operators

- Let $m > 0$. $R [*m] \equiv (R \text{ ##1 } R \text{ ##1 } \dots \text{ ##1 } R) // m \text{ copies of } R .$
- $R [*0:\$] \equiv (R [*0] \text{ or } R [*1:\$]) .$
- Let $m \leq n$. $R [*m:n] \equiv (R [*m] \text{ or } R [*m+1] \text{ or } \dots \text{ or } R [*n]) .$
- Let $m > 1$. $R [*m:\$] \equiv (R [*m-1] \text{ ##1 } R [*1:\$]) .$

1.3.3 Derived delay and concatenation operators

Let $m \leq n$.

- $(\text{ ## } [m : n] R) \equiv (1 [*m:n] \text{ ##1 } R) .$
- $(\text{ ## } [m : \$] R) \equiv (1 [*m:\$] \text{ ##1 } R) .$
- $(\text{ ## } m R) \equiv (1 [*m] \text{ ##1 } R) .$
- Let $m > 0$. $(R_1 \text{ ## } [m : n] R_2) \equiv (R_1 \text{ ##1 } 1 [*m-1:n-1] \text{ ##1 } R_2) .$
- Let $m > 0$. $(R_1 \text{ ## } [m : \$] R_2) \equiv (R_1 \text{ ##1 } 1 [*m-1:\$] \text{ ##1 } R_2) .$
- Let $m > 1$. $(R_1 \text{ ## } m R_2) \equiv (R_1 \text{ ##1 } 1 [*m-1] \text{ ##1 } R_2) .$
- $(R_1 \text{ ## } [0 : 0] R_2) \equiv (R_1 \text{ ##0 } R_2) .$
- Let $n > 0$. $(R_1 \text{ ## } [0 : n] R_2) \equiv ((R_1 \text{ ##0 } R_2) \text{ or } (R_1 \text{ ## } [1 : n] R_2)) .$
- $(R_1 \text{ ## } [0 : \$] R_2) \equiv ((R_1 \text{ ##0 } R_2) \text{ or } (R_1 \text{ ## } [1 : \$] R_2)) .$

1.3.4 Derived non-consecutive repetition operators

Let $m \leq n$.

- $b [*->m:n] \equiv (!b [*0:\$] \##1 b) [*m:n]$
- $b [*->m:\$] \equiv (!b [*0:\$] \##1 b) [*m:\$]$
- $b [*->m] \equiv (!b [*0:\$] \##1 b) [*m]$
- $b [*=m:n] \equiv (b [*->m:n] \##1 !b [*0:\$])$
- $b [*=m:\$] \equiv (b [*->m:\$] \##1 !b [*0:\$])$
- $b [*=m] \equiv (b [*->m] \##1 !b [*0:\$])$

1.3.5 Other derived operators

- $(R_1 \text{ and } R_2)$
 $\equiv (((R_1 \##1 1[*0:\$]) \text{ intersect } R_2) \text{ or } (R_1 \text{ intersect } (R_2 \##1 1[*0:\$])))$.
- $(R_1 \text{ within } R_2) \equiv ((1[*0:\$] \##1 R_1 \##1 1[*0:\$]) \text{ intersect } R_2)$.
- $(b \text{ throughout } R) \equiv ((b [*0:\$]) \text{ intersect } R)$.
- $(b, v = e) \equiv (b \##0 (1, v = e))$.

2 Semantics

Let \mathbf{P} be the set of atomic propositions.

The semantics of assertions and properties is defined via a relation of satisfaction by empty, finite, and infinite words over the alphabet $\Sigma = 2^{\mathbf{P}} \cup \{\top, \perp\}$. Such a word is an empty, finite, or infinite sequence of elements of Σ . The number of elements in the sequence is called the *length* of the word, and the length of word w is denoted $|w|$. Note that $|w|$ is either a non-negative integer or ∞ .

The sequence elements of a word are called its *letters* and are assumed to be indexed consecutively beginning at zero. If $|w| > 0$, then the first letter of w is denoted w^0 ; if $|w| > 1$, then the second letter of w is denoted w^1 ; and so forth. $w^{i\cdot}$ denotes the word obtained from w by deleting its first i letters. If $i < |w|$, then $w^{i\cdot} = w^i w^{i+1} \dots$. If $i \geq |w|$, then $w^{i\cdot}$ is empty.

If $i \leq j$, then $w^{i,j}$ denotes the finite word obtained from w by deleting its first i letters and also deleting all letters after its $(j + 1)$ st. If $i \leq j < |w|$, then $w^{i,j} = w^i w^{i+1} \dots w^j$.

If w is a word over Σ , define \bar{w} to be the word obtained from w by interchanging \top with \perp . More precisely, $\bar{w}^i = \top$ if $w^i = \perp$; $\bar{w}^i = \perp$ if $w^i = \top$; and $\bar{w}^i = w^i$ if $w^i \in 2^{\mathbf{P}}$.

The semantics of clocked sequences and properties is defined in terms of the semantics of unclocked sequences and properties. See the subsection on rewrite rules for clocks below.

It is assumed that the satisfaction relation $\ell \models b$ is defined for elements $\ell \in 2^{\mathbf{P}}$ and boolean expressions b . For any boolean expression b , define

$$\top \models b \quad \text{and} \quad \perp \not\models b .$$

2.1 Rewrite rules for clocks

The semantics of clocked sequences and properties is defined in terms of the semantics of unclocked sequences and properties. The following rewrite rules define the transformation of a clocked sequence or property into an unclocked version that is equivalent for the purposes of defining the satisfaction relation.

- $@(c) b \mapsto (!c [*0:\$] \#\#1 c \& b) .$
- $@(c) (1, v = e) \mapsto (@(c) 1 \#\#0 (1, v = e)) .$
- $@(c) (R) \mapsto @(c) R .$
- $@(c) (R_1 \#\#1 R_2) \mapsto (@(c) R_1 \#\#1 @(c) R_2) .$
- $@(c) (R_1 \#\#0 R_2) \mapsto (@(c) R_1 \#\#0 @(c) R_2) .$
- $@(c) (R_1 \text{ or } R_2) \mapsto (@(c) R_1 \text{ or } @(c) R_2) .$
- $@(c) (R_1 \text{ intersect } R_2) \mapsto (@(c) R_1 \text{ intersect } @(c) R_2) .$
- $@(c) \text{ first_match } (R) \mapsto \text{ first_match } (@(c) R) .$
- $@(c) R [*0] \mapsto (@(c) R) [*0] .$
- $@(c) R [*1:\$] \mapsto (@(c) R) [*1:\$] .$
- $@(c) \text{ disable iff } (b) \varphi \mapsto \text{ disable iff } (b) @(c) \varphi .$
- $@(c) \text{ not } b \mapsto @(c) !b .$
- $@(c) \text{ not } R \mapsto \text{ not } @(c) R$, provided R is not a boolean expression.
- $@(c) \nu_1 (R_1 \mid\rightarrow \nu_2 R_2) \mapsto \nu_1 (@(c) R_1 \mid\rightarrow @(c) \nu_2 R_2) .$
- $(S_1 \#\# S_2) \mapsto (S_1 \#\#1 S_2)$

2.2 Tight satisfaction without local variables

Tight satisfaction is denoted by \models . For unclocked sequences without local variables, tight satisfaction is defined as follows. w, x, y, z denote finite words over Σ .

- $w \models b$ iff $|w| = 1$ and $w \models b$.
- $w \models (R)$ iff $w \models R$.
- $w \models (R_1 \#\#1 R_2)$ iff there exist x, y such that $w = xy$ and $x \models R_1$ and $y \models R_2$.
- $w \models (R_1 \#\#0 R_2)$ iff there exist x, y, z such that $w = xyz$ and $|y| = 1$, and $xy \models R_1$ and $.$
- $w \models (R_1 \text{ or } R_2)$ iff either $w \models R_1$ or $w \models R_2$.
- $w \models (R_1 \text{ intersect } R_2)$ iff both $w \models R_1$ and $w \models R_2$.

- $w \models \text{first_match} (R)$ iff both
 - $w \models R$ and
 - if there exist x, y such that $w = xy$ and $x \models R$, then y is empty.
- $w \models R [*0]$ iff $|w| = 0$.
- $w \models R [*1:\$]$ iff there exist w_1, w_2, \dots, w_j ($j \geq 1$) such that $w = w_1 w_2 \dots w_j$ and for every i such that $1 \leq i \leq j$, $w_i \models R$.

If S is a clocked sequence, then $w \models S$ iff $w \models S'$, where S' is the unlocked sequence that results from S by applying the rewrite rules.

2.3 Satisfaction without local variables

2.3.1 Satisfaction by infinite words

w denotes an infinite word over Σ . Assume that all properties, sequences, and unlocked property fragments do not involve local variables.

Assertion Satisfaction:

For the definition of assertion satisfaction, b denotes the boolean expression representing the enabling condition for the assertion. Intuitively, b is derived from the conditions in the context of a procedural assertion, while b is “1” for a declarative assertion.

- $w, b \models \text{always } \textcircled{c} \text{ assert property } P$ iff for every $i \geq 0$ such that $w^i \models c$, if $\bar{w}^i \models b$ then $w^{i\cdot} \models \textcircled{c} P$.
- $w, b \models \text{always assert property } Q$ iff for every $i \geq 0$, if $\bar{w}^i \models b$ then $w^{i\cdot} \models Q$.
- $w, b \models \text{initial } \textcircled{c} \text{ assert property } P$ iff if there exists $i \geq 0$ such that $w^i \models c$, then for the first such i , if $\bar{w}^i \models b$ then $w^{i\cdot} \models \textcircled{c} P$.
- $w, b \models \text{initial assert property } Q$ iff (if $\bar{w}^0 \models b$ then $w \models Q$).

Property Satisfaction:

- $w \models Q$ iff $w \models Q'$, where Q' is the unlocked property that results from Q by applying the rewrite rules.
- $w \models \text{disable iff } (b) \varphi$ iff either $w \models \varphi$ or there exists $k \geq 0$ such that $w^k \models b$ and $w^{0,k-1} \top^\omega \models \varphi$. Here, $w^{0,-1}$ denotes the empty trace.
- $w \models \text{not } \varphi$ iff $\bar{w} \not\models \varphi$.
- $w \models R$ iff there exists $j \geq 0$ such that $w^{0,j} \models R$.
- $w \models (R_1 \mid \rightarrow \nu R_2)$ iff for every $j \geq 0$ such that $\bar{w}^{0,j} \models R_1$, $w^{j\cdot} \models \nu R_2$.

Remark. It can be proved that $w \models \text{not } b$ iff $w \models !b$.

2.3.2 Satisfaction by finite words

For an assertion A , we define both weak and strong satisfaction, denoted \models^- and \models^+ (respectively) by a finite word w over Σ . These relations are defined in terms of the relation of satisfaction by infinite words as follows:

- $w \models^- A$ iff $w \top^\omega \models A$.
- $w \models^+ A$ iff $w \perp^\omega \models A$.

A tool checking for satisfaction of A by the finite word w should return

- “true” if $w \models^+ A$.
- “false” if $w \not\models^- A$.
- “unknown” otherwise.

2.4 Local variable flow

In this subsection, we define inductively how local variable names flow through unlocked sequences. Below, “ \cup ” denotes set union, “ \cap ” denotes set intersection, “ $-$ ” denotes set difference, and “ $\{\}$ ” denotes the empty set.

The function “*block*” takes a sequence as input and returns a set of local variable names as output. Intuitively, this function returns the set of local variable names that are blocked from flowing out of the sequence.

The function “*flow*” takes a set X of local variable names and a sequence as input and returns a set of local variable names as output. Intuitively, this function returns the set of local variable names that flow out of the sequence given the set X of local variable names that flow into the sequence.

The function “*block*” is defined by

- $block(b) = \{\}$.
- $block((1, v = e)) = \{\}$.
- $block((R)) = block(R)$.
- $block((R_1 \##1 R_2)) = (block(R_1) - flow(\{\}, R_2)) \cup block(R_2)$.
- $block((R_1 \##0 R_2)) = (block(R_1) - flow(\{\}, R_2)) \cup block(R_2)$.
- $block((R_1 \text{ or } R_2)) = block(R_1) \cup block(R_2)$.
- $block((R_1 \text{ intersect } R_2))$
 $= block(R_1) \cup block(R_2) \cup (flow(\{\}, R_1) \cap flow(\{\}, R_2))$.
- $block(first_match (R)) = block(R)$.
- $block(R [*0]) = \{\}$.
- $block(R [*1:\$]) = block(R)$.

The function “*flow*” is defined by

- $flow(X, b) = X$.
- $flow(X, (1, v = e)) = X \cup \{v\}$.
- $flow(X, (R)) = flow(X, R)$.
- $flow(X, (R_1 \#\#1 R_2)) = flow(flow(X, R_1), R_2)$.
- $flow(X, (R_1 \#\#0 R_2)) = flow(flow(X, R_1), R_2)$.
- $flow(X, (R_1 \text{ or } R_2)) = flow(X, R_1) \cap flow(X, R_2)$.
- $flow(X, (R_1 \text{ intersect } R_2))$
 $= (flow(X, R_1) \cup flow(X, R_2)) - block((R_1 \text{ intersect } R_2))$.
- $flow(X, first_match(R)) = flow(X, R)$.
- $flow(X, R [*0]) = X$.
- $flow(X, R [*1:\$]) = flow(X, R)$.

Remark: It can be proved that

$$flow(X, R) = (X \cup flow(\{\}, R)) - block(R) .$$

It follows that $flow(\{\}, R) \cap block(R) = \{\}$. Intuitively, $flow(\{\}, R)$ is the set of local variable names that are sampled in R and are not blocked by R .

2.5 Tight satisfaction with local variables

A *local variable context* is a function that assigns values to local variable names. If L is a local variable context, then $\text{dom}(L)$ denotes the set of local variable names that are in the domain of L . If $D \subseteq \text{dom}(L)$, then by $L|_D$ we mean the local variable context obtained from L by restricting its domain to D .

In the presence of local variables, tight satisfaction is a four-way relation defining when a finite word w over the alphabet Σ together with an input local variable context L_0 satisfies an unlocked sequence R and yields an output local variable context L_1 . This relation is denoted

$$w, L_0, L_1 \models R .$$

and is defined below. It can be proved that the definition guarantees that

$$w, L_0, L_1 \models R \quad \text{implies} \quad \text{dom}(L_1) = flow(\text{dom}(L_0), R) .$$

- $w, L_0, L_1 \models (1, v = e)$ iff $|w| = 1$ and $w^0 \models 1$ and

$$L_1 = \{(v, e[L_0, w^0])\} \cup L_0|_{\text{dom}(L_0) - \{v\}} ,$$

where $e[L_0, w^0]$ denotes the value obtained from e by evaluating first according to L_0 and second according to w^0 .

- $w, L_0, L_1 \models b$ iff $|w| = 1$ and $w^0 \models b[L_0]$ and $L_1 = L_0$. Here $b[L_0]$ denotes the expression obtained from b by substituting values from L_0 .

- $w, L_0, L_1 \models (R)$ iff $w, L_0, L_1 \models R$.
- $w, L_0, L_1 \models (R_1 \ \#\#1 \ R_2)$ iff there exist x, y, L' such that $w = xy$ and $x, L_0, L' \models R_1$ and $y, L', L_1 \models R_2$.
- $w, L_0, L_1 \models (R_1 \ \#\#0 \ R_2)$ iff there exist x, y, z, L' such that $w = xyz$ and $|y| = 1$, and $xy, L_0, L' \models R_1$ and $yz, L', L_1 \models R_2$.
- $w, L_0, L_1 \models (R_1 \ \text{or} \ R_2)$ iff there exists L' such that both of the following hold:
 - either $w, L_0, L' \models R_1$ or $w, L_0, L' \models R_2$, and
 - $L_1 = L'|_D$, where $D = \text{flow}(\text{dom}(L_0), (R_1 \ \text{or} \ R_2))$.
- $w, L_0, L_1 \models (R_1 \ \text{intersect} \ R_2)$ iff there exist L', L'' such that $w, L_0, L' \models R_1$ and $w, L_0, L'' \models R_2$ and $L_1 = L'|_D \cup L''|_D$, where $D = \text{flow}(\text{dom}(L_0), (R_1 \ \text{intersect} \ R_2))$.
- $w, L_0, L_1 \models \text{first_match} (R)$ iff both
 - $w, L_0, L_1 \models R$ and
 - if there exist x, y, L' such that $w = xy$ and $x, L_0, L' \models R$, then y is empty.
- $w, L_0, L_1 \models R \ [*0]$ iff $|w| = 0$ and $L_1 = L_0$.
- $w, L_0, L_1 \models R \ [*1 : \$]$ iff there exist $L_{(0)} = L_0, w_1, L_{(1)}, w_2, L_{(2)} \dots, w_j, L_{(j)} = L_1$ ($j \geq 1$) such that $w = w_1 w_2 \dots w_j$ and for every i such that $1 \leq i \leq j$, $w_i, L_{(i-1)}, L_{(i)} \models R$.

If S is a clocked sequence, then $w, L_0, L_1 \models S$ iff $w, L_0, L_1 \models S'$, where S' is the unclocked sequence that results from S by applying the rewrite rules.

2.6 Satisfaction with local variables

2.6.1 Satisfaction by infinite words

w denotes an infinite word over Σ . L_0, L_1 denote local variable contexts.

The rules defining assertion satisfaction are identical to those without local variables, but with the understanding that the underlying properties can have local variables.

Property Satisfaction:

- $w \models Q$ iff $w, \{\} \models Q'$, where Q' is the unclocked property that results from Q by applying the rewrite rules.
- $w, \{\} \models \text{disable} \ \text{iff} \ (b) \ \varphi$ iff either $w, \{\} \models \varphi$ or there exists $k \geq 0$ such that $w^k \models b$ and $w^{0,k-1} \top^\omega, \{\} \models \varphi$. Here, $w^{0,-1}$ denotes the empty trace.
- $w, L_0 \models \text{not} \ \varphi$ iff $\bar{w}, L_0 \not\models \varphi$.
- $w, L_0 \models R$ iff there exist $j \geq 0$ and L_1 such that $w^{0,j}, L_0, L_1 \models R$.
- $w, L_0 \models (R_1 \ \mid \rightarrow \ \nu \ R_2)$ iff for every $j \geq 0$ and L_1 such that $\bar{w}^{0,j}, L_0, L_1 \models R_1$, $w^{j,\cdot}, L_1 \models \nu \ R_2$.

2.6.2 Satisfaction by finite words

The definition is identical to that without local variables, but with the understanding that the underlying properties can have local variables.

3 Extended Expressions

3.1 Extended booleans

w denotes an infinite trace over Σ , T denotes a clocked or unclocked sequence.

- $w^j \models T.\text{ended}$ iff there exist $i \leq j$ and L such that $w^{i,j}, \{\}, L \models T$.
- $w^j \models @ (c) (T.\text{matched})$ iff there exists $i < j$ such that $w^i \models T.\text{ended}$ and $w^{i+1,j}, \{\}, \{\} \models (!c [*0:\$] \#\#1 c)$.
- $w^j \models @ (c) \$\text{stable}(e)$ iff there exists $i < j$ such that $w^{i,j}, \{\}, \{\} \models (c \#\#1 c [*->1])$ and $e[w^i] = e[w^j]$.
- $w^j \models @ (c) \$\text{rose}(e)$ iff there exists $i < j$ such that $w^{i,j}, \{\}, \{\} \models (c \#\#1 c [*->1])$ and $b[w^i] = 1$ and $b[w^j] \neq 1$, where b is the least-significant bit of e .
- $w^j \models @ (c) \$\text{fell}(e)$ iff there exists $i < j$ such that $w^{i,j}, \{\}, \{\} \models (c \#\#1 c [*->1])$ and $b[w^i] = 0$ and $b[w^j] \neq 0$, where b is the least-significant bit of e .

3.2 Past

w denotes an infinite trace over Σ .

- Let $n \geq 1$. If there exist $i < j$ such that $w^{i,j}, \{\}, \{\} \models (c \#\#1 c [*->n-1])$, then

$$@ (c) \$\text{past}(e, n)[w^j] = e[w^i].$$

Otherwise, $@ (c) \$\text{past}(e, n)[w^j]$ is undefined.

- $\$\text{past}(e) \equiv \$\text{past}(e, 1)$.