



Some Problems with the PSL language definition

Version 1.1
March 25, 2004

Infineon Technologies AG

Authors:

Adriana Maggiore, Christian Pichler, Klaus Winkelmann

Abstract:

This document describes some problems with the current PSL definition (as of LRM 1.1 draft a March 2004) and proposes changes which are required for easy and efficient implementation.

TABLE OF REVISIONS

Issue	Date	Description + Reason for the Modification	Affected Sections
0.9	Mar 16, 2004	Creation	
1.0	Mar 19, 2004	First distributed version	
1.1	March 24, 2004	Adapted to LRM 1.1 of March 22, 2004	

CONTENTS

1	SYNTAX	4
1.1	HDL embedding.....	4
1.1.1	Problem description	4
1.1.2	Proposed solution.....	4
1.2	Tokenisation	5
1.2.1	Problem description	5
1.2.2	Proposed solution.....	5
1.3	Case-Sensitivity	5
1.3.1	Problem description	5
1.3.2	Proposed solution.....	5
1.4	Operator precedences.....	5
1.4.1	Problem description	5
1.4.2	Proposed solution.....	5
1.5	PSL_Identifier	5
1.6	Sequence	5
1.7	Sequence operators (&, &&,).....	6
1.8	Suffix implication.....	6
2	SEMANTICS.....	6
2.1	Length matching	6
2.1.1	Problem description	6
2.1.2	Proposed solution.....	6
2.2	SERE concatenation (;)	7
2.3	Inheritance	7
2.4	Replicated properties.....	7
2.5	Verification directives.....	8
3	EDITORIAL COMMENTS.....	8
4	CORE LANGUAGE	9

This document is motivated by Infineon’s effort to support a PSL subset with our verification tool CVE. However, the issues encountered are not specific to CVE, but stem from our general experience in designing languages and language-based software.

All observations discussed herein refer to the following document, briefly called “LRM”:

“Property Specification Language Reference Manual Draft a, Version 1.1, March 22, 2004”.

Note that in some cases we explicitly propose changes to the language, while in other cases what are required are only clarifications or additional definitions.

1 Syntax

1.1 HDL embedding

1.1.1 Problem description

The PSL syntax definitions make no clear distinction between PSL and HDL expressions. This makes it difficult to write clean parsers, and it makes it also complicated for a user to read PSL expressions.

Example:

- `{a} |-> b /*` in this case “{a}” has to be interpreted as `FL_Property`
- `{a} == b /*` in this case `{a}` has to be interpreted as HDL expression
- `{{a} & {b}} | => c /*` how should we interpret this ?*/
 assume that a and b are bitvectors: does the expression say that if the bitwise conjunction of a and b is different from zero, then in the next step c should be true?
 Or does it say that if a is different from zero and b is different from zero ...
 i.e. are “{a}” and “{b}” SERE’s or is “{a} & {b}” an HDL_expression??

1.1.2 Proposed solution

We propose the following syntactic changes:

Either put every HDL_expression into a wrapper like “true(...)” or “p(...)”

or change the syntax as follows

1. HDL-expressions in `FL_Properties` have to be put always in “{“ ”} brackets.
2. “{“ ”} brackets in an SERE-Context are always interpreted as SERE’s unless they are put into “(“ ”) brackets.

```

/* SERE = Boolean_without_curly_brackets
    | “(“ Boolean “)”
    | ...
FL_Property =
    Sequence
    | ...
*/
  
```

Example:

```

{{a}} |-> c    “{a}” is a SERE
{{a,b}} |-> c  syntax error
{{{a,b}}} |-> c  ok, “{a,b}” is an HDL_expression
  
```

1.2 Tokenisation

1.2.1 Problem description

In the current definition it is unclear what the tokens of the language are.

Example: is “[*]” a single token (i.e. it is not allowed that there are white spaces in the token); or are these two or even three tokens (“[*” and “]” or “[“,”*” and “]”)?

1.2.2 Proposed solution

The EBNF should have a more formal format.

1.3 Case-Sensitivity

1.3.1 Problem description

The current LRM states that keywords in PSL are case-sensitive, regardless of the underlying HDL rules for identifiers. However, in VHDL, not only identifiers are case-insensitive but also the VHDL keywords. Therefore it seems very unnatural that all PSL-keywords are case-sensitive but all keywords of the embedded VHDL are case-insensitive (especially because some keywords like **and**, **or**, **not** are shared between PSL and the embedded VHDL).

Example: (A AND B) would be legal FLProperty; however (A AND (B |-> (C AND D))) would result in a syntax error (however “(A and (B |-> (C AND D)))” would be syntactically correct).

1.3.2 Proposed solution

It should depend on the embedded HDL, if keywords in PSL are case-sensitive or case-insensitive (i.e. if the embedded HDL is Verilog, then the keywords should be case-sensitive; if the embedded HDL is VHDL they should be case-insensitive).

1.4 Operator precedences

1.4.1 Problem description

Refers to LRM section 4.2.2.

The precedence/associativity for some operators is missing: **union**, **!**, **[+]**, **()** (in Sequence (FLProperty))

1.4.2 Proposed solution

Should be described in Table 2 (FL operator precedence and associativity)

1.5 PSL_Identifier

Refers to LRM section 4.2., appendix A.

The definition of a PSL_Identifier is missing. It is also not clear, if identifier names are case-sensitive or case-insensitive, if the tokens of the embedded HDL are case-insensitive (as VHDL).

1.6 Sequence

Refers to LRM section 6.1.2 and A.3.5

The syntax definition of sequences in LRM 6.1.2 and A.3.5 don't match.

1.7 Sequence operators (&, &&, |)

Refers to LRM section 4.2.2

The sequence operators (&, && and |) have different priorities in Verilog and PSL. So you will get ambiguities when parsing expressions (ex: “{a & b[*2]}”: if & is interpreted as Verilog operator then the expression is equivalent to “{(a&b)[*2]}”; if & is interpreted as a PSL sequence operator then the expression is equivalent to “{a & {b[*2]}}”. Both interpretations are legal according to the syntax definition in LRM (however, their semantic is different).

A possible interpretation of the LRM is that, because the HDL operators have the highest precedence, then the example must be interpreted as {(a & b)[*2]}.

However, it appears quite confusing that two operators with different precedence are denoted by the same token (Verilog “&” and PSL “&”).

1.8 Suffix implication

Refers to LRM section 6.2.1.6., appendix B3.2

The form

```
Sequence "( " FL_Property ")"
```

is defined in the appendix B to be syntactic sugar for

```
Sequence "|->" FL_Property
```

But the informal semantics given in 6.2.1.6 use different wording. This is confusing. The informal semantics should make it clear that the two are the same. Even better, *drop the first form completely*. It is inconsistent with the style of all other operators, which are denoted by explicit symbols rather than in this implicit way.

2 Semantics

2.1 Length matching

Refers to LRM section 6.1.1.6 and appendix B.

2.1.1 Problem description

Let's have a truncated input of just one letter “a” (i.e. {a} is a word of length 1).

Would the following properties hold?

- {a} |= {{a;false} && {a;a}}
- {a} |= {{a} && {a;a}}

The LRM 1.1 version states that at the end of the input sequence you have to check that the length of the expressions r1 and r2 could be the same.

Therefore

```
{a} |= {{a;false} && {a;a}}
```

would hold but

```
{a} |= {{a} && {a;a}}
```

would fail.

2.1.2 Proposed solution

I would propose that both properties should hold. The semantic rules for “r1 && r2” should be adapted appropriately.

2.2 SERE concatenation (;)

Refers to LRM section 6.1.1.1

In the informal semantics, why it is said that "A;B holds on a path" rather than "A;B holds tightly on a path"?

2.3 Inheritance

Refers to LRM section 5.2.1

Are the following examples legal according to the rules for names in HDL expressions?

```
vunit V1 (top) {
wire temp;
assign temp = ...
}
```

```
vunit V2 (top) {
inherit V1;
wire temp;
assign temp = ...
}
```

```
vunit V3 (top) {
inherit V2;
assert always temp && x;
}
```

How many declaration of "temp" are in the transitive closure of V3? The rules in 5.2.1 requires that there is a single declaration.

Also,

```
vunit V1 (top) {
wire temp;
assign temp = ...
}
```

```
vunit V2 (top) {
inherit V1;
...
}
```

```
vunit V3 (top) {
inherit V1;
...
}
```

```
vunit V4 (top) {
inherit V2, V3;
...
}
```

How many declarations of "temp" are there?

What is the exact definition of transitive closure for vunit inheritance?

2.4 Replicated properties

Refers to LRM section 6.2.3

The definition suggests that the forall Name can be used as a repetition count "an implementation shall support at least ... use of the Name as an index or repetition count."

The definition also states that "The Name defined by a replicator represents a non-static variable."

In section 6.1.2.1, a repetition count is required to be statically computable.

This problem was already present in LRM 1.01 and had been raised in:

<http://www.eda.org/vfv/hm/1039.html>

2.5 Verification directives

Refers to LRM section 7.1

What is the scope of a label? Is it the vunit? How does vunit inheritance and vunit binding effect labels?

3 Editorial Comments

4.1.3.3

There is no reference to the document which defines GDL.

4.2.3 Macros

page 19, line 50: should "PSL directives" be "PSL macros"?

4.2.3.1 %for

note 5 on page 20: should "similar" be "the same as" ?

4.2.5 Comments

The way comments are expressed in the SystemVerilog flavour is missing.

5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5

The definitions of bit, boolean, bitvector, numeric and string HDL_or_PSL_expressions is missing for the GDL flavour.

5.2.2

There is no reference to the GDL flavour.

6.1.1.6 SERE length-matching and (&&)

In the restrictions, the symbol "&" should be replaced by "&&"

6.1.2 and A.3.5

The definition of Sequence is different in the two sections (while still equivalent).

7.2

page 88, lines 32 – 34: Redundant "is not explicitly bound" makes the sentence incorrect.

4 Core language

PSL (syntax and especially semantics) as defined in the current LRM is too complex for most users, and simply too large. It is more likely to be used if it can be presented in a concise way. This criticism refers both to syntax and semantics:

- There are too many redundancies and even ambiguities, see above.
- The semantics must be described completely within the LRM itself, on a reasonable number of pages. As long as additional references to about 80 pages of scientific papers are needed to clarify open issues, the language will simply not be accepted by practising engineers.

Furthermore, there are too many changes between different releases of the LRM. This indicates that as a whole, the language is not yet mature enough to be a standard.

Therefore standardization should be split into at least two part. The first PSL standard should only define a ***small subset*** of the current language, which should be syntactically and semantically clean. Extensions of this subset should be ***postponed***.

Possible starting points for the core language could be (to be discussed):

- a subset aligned with SVA
- the simple subset.

Parts of the language which should better be kept out of the core language in order to keep it manageable and clear include:

- OBE,
- clocked expressions,
- redundancies such as “X” (= “next”),
- inheritance for unbound modules.