==========

VHDL-200X Assertions Change Proposal

ID: ????

Proposer: Erich Marschner
email: erichm@cadence.com

Status: ready for review
Proposed: June 8, 2004: rev. 1
- first complete draft
July 6, 2004: rev. 2
- updated to address remaining issues:
  - ambiguity of "assert condition"
  - handling of PSL identifiers within VHDL
  - changes to allow endpoints in VHDL expressions generally
Analyzed: Date
Resolved: Date

Enhancement Summary: addition of PSL to VHDL 200x
Related issues: syntax for pathnames; assertion API
Relevant LRM section: affects many LRM sections - see below

Enhancement Detail:

Specific changes to incorporate (portions of) PSL into VHDL 200x, by reference to the Accellera PSL v1.1 LRM (final approved LRM dated June 9, 2004).

Goals

1. Support the 'simple subset' of PSL.
2. Support the Boolean, Temporal, Verification, and Modeling layers.
3. Support the VHDL 'flavor' of PSL, but also give consideration to portable PSL.
4. Support both embedded PSL and separate PSL units.

Guidelines

1. Be conservative in incorporating PSL - we can add more later, but we cannot subtract easily.
2. Avoid decisions that would make it difficult to use PSL in a mixed-language design.

3. Avoid decisions that might be in conflict with potential changes/extensions in PSL v1.2.
4. Accomplish the inclusion of PSL with minimal change to the VHDL LRM.

Preliminaries

1. Use Model for PSL

We would like to incorporate PSL in such a way that it supports well both simulation and LTL-based static verification. This involves supporting at least the assert/assume/restrict/cover directives. The others - assume_guarantee, restrict_guarantee, fairness - are not clearly required for this. (Fairness is needed for CTL-based model checkers; assume_guarantee and restrict_guarantee have to do with tracking proof obligations, which is in some sense a tool issue rather than a language issue.)

We would also like to support both embedded PSL (entered by designers as the design progresses, and used primarily, or at least initially, for white-box, module-level testing) and separate PSL files (probably created by verification engineers, and used either as library 'checkers', or for black-box testing). PSL embedded in the design probably ought to be more or less invariant, i.e., it should not require continual editing. PSL presented as separate files is more easily modified or swapped into/out of the build, without the risk of accidentally changing the design itself.

We need to decide whether the same PSL capabilities should be supported in both embedded and separate contexts. For several of the four directives (assert/assume/restrict/cover), there is an argument that the directive should be used only in separate files: 'assume's are often temporary, and may need to change with each static verification run; the same argument might be made for 'restrict'; some believe that 'cover' directives should be specified separately from the design (so that different coverage objectives can be pursued with each simulation).

If we allow assume/restrict to be embedded in a VHDL design unit, then we will have to specify what these mean for simulation, as the PSL definitions for these are focused on static verification. One possibility is to interpret both as assertions to check (noting that 'restrict' has omega semantics, so 'restrict a[*]' would need to be checked by 'assert {a[*]; EOS}', where EOS holds only at the end of simulation). For assume/restrict on primary inputs of the DUT, these might also be interpreted as constraints for constrained random input generation, but for this case, it might be better to put the assume/restrict in the testbench, or in a separate PSL vunit.

2. Embedded vs. Separate PSL

The PSL LRM defines only PSL in verification units. Allowing PSL to be embedded in a VHDL design unit M should be explained in terms of equivalent PSL in a vunit bound to M. This will make embedded PSL 'backward compatible', in some sense, with the PSL LRM definition.

Since more than one vunit can be bound to the same module, if we define embedded PSL in terms of vunits, then we have to account for the possibility of multiple vunits having different default clocks, and whether those different default clocks can all be embedded in the same VHDL design unit. This also relates to whether PSL declarations and directives can be intermixed anywhere in a VHDL design unit, or whether PSL declarations can only appear in the declarative part, and directives only in the statement part, of a VHDL design unit.

3. PSL vs. Existing Assertions

VHDL already has assertions. In particular, VHDL concurrent assertions are quite similar to the simplest of PSL assertions. For example, an invariant check could be expressed both by the PSL assertion

```
assert always Condition report "Condition failed";
```

or by the VHDL concurrent assertion

```
assert Condition report "Condition failed";
```

We need to decide whether these two should be merged into one, or should remain separate. If the latter, we need to explain when to use one or the other.

UPDATE:
~~~~~~
~~~~~~~~~~~~~~~~~~~~

In fact, existing VHDL concurrent assertions are the same form as one kind of PSL assertion, but the semantics are different, so there is a potential ambiguity we need to avoid. Specifically, the syntax

```
assert Condition ;
```

could be interpreted as either an existing VHDL concurrent assertion that is implicitly required to hold at every simulation cycle, or as a PSL assertion that is required to hold only at the beginning of simulation. Essentially, VHDL concurrent assertions have an

implicit 'always' temporal operator, whereas PSL assertions have an explicit temporal operator (always, never, next, eventually!, etc.), and in the absence of such an operator, a PSL assertion is only checked once, at the beginning of simulation. We will need to either change VHDL concurrent assertions to require an explicit 'always' operator, or exclude PSL assertions that apply only to time 0, in order to avoid this ambiguity.

END UPDATE:
~~~~~~
~~~~~~~~~~~~~~~

4. Level of Integration

PSL is defined in the PSL v1.1 LRM. We need to decide how little or much needs to be said about PSL in the VHDL 200x LRM. Clearly the VHDL grammar must be extended to identify the places in which PSL constructs may appear. But to what extent do we need to specify PSL semantics in the context of VHDL? Any explanation of PSL semantics in the VHDL LRM would be at best redundant, and at worst in conflict, w.r.t. the PSL LRM, unless we are discussing semantics that are left undefined in PSL - and any such undefined semantics may become defined in the next version of PSL. So it is probably appropriate to minimize any discussion of PSL semantics in the VHDL LRM.

The granularity of PSL integration is also an issue. Integration at the design unit level (allowing PSL verification units as design units) and at the declaration/statement level (to allow embedding of PSL code within VHDL design units) seems reasonable and relatively easy. Integration at the expression level - e.g., allowing PSL built-in functions, or endpoints, or even sequences, within VHDL declarations or statements (other than in the Modeling Layer within a PSL verification unit, where PSL already extends VHDL expressions in this manner) would require substantially more effort. So it seems best to lean towards design unit/declaration/statement level integration.

A related issue is whether to allow PSL code in VHDL packages. PSL directives are essentially concurrent statements. VHDL does not currently allow concurrent statements in packages, so allowing PSL directives in packages would be a bit odd. Such directives would also be of limited utility, since they could only refer to global signals declared in those packages. On the other hand, PSL declarations might be allowed in packages without any issues; they would be analogous to declarations of procedures that are invoked by concurrent procedure calls.

Approach

1. Allow PSL vunits in VHDL 200x as design units, with all the contents allowed by the PSL v1.1 LRM--i.e., PSL declarations and directives, and modeling layer code. This includes allowing a PSL vunit to have a context clause.

2. Allow PSL declarations in packages and in the declarative part of an entity, architecture, or block statement.

3. Allow PSL directives as a new kind of statement, in any concurrent statement part.

4. Allow (at most) only a single default clock declaration per declarative part, in line with PSL's requirement for at most a single default clock declaration in a verification unit. (Note that a verification unit V bound to an architecture A could then be thought of as a block V' in A, where the declarations of V appear in the declarative part of V', and the directives of V appear in the statement part of V'.)

5. Support the temporal layer, as restricted for the Simple Subset of PSL (defined in PSL v1.1 LRM, section 4.4.4).

(** NEW **)
5a. Clarify that an assertion of the form "assert condition", with no temporal operators, will be interpreted as a VHDL concurrent assertion (i.e., equivalent to a PSL assertion of the form "assert always condition") rather than as a PSL property that applies only at time 0. [Note that any other form of PSL assertion that applies to time 0 (e.g., assert {a} |=> {b}) will be interpreted as PSL, as usual; similarly, properties of the form "assert P" where P is a PSL property of whatever form, are not affected. Only properties that are simple Boolean conditions are affected.]

6. Support the verification layer directives assert/assume/restrict/cover, both embedded in other VHDL units, and in PSL verification units. Leave unspecified how assume/restrict are interpreted in simulation. (This leaves room for their use in constrained random input generation, e.g., if they appear in a testbench, and also allows them to be interpreted as assertions, if they appear in the DUT.)

7. Support the modeling layer in vunits - i.e., auxiliary VHDL code in a verification unit. Note that this allows a PSL vunit to contain a use clause.

8. Support the VHDL flavor of PSL in all of the above.

9. Extend the scopes of objects declared in a VHDL design unit into any corresponding verification unit (as with configurations), so any code in a verification unit can refer to objects declared in the corresponding design unit.

(** NEW **)
9a. Handle trailing underscore/bang keywords in PSL (e.g., before_, before!) by (a) generally removing the restriction prohibiting trailing underscores in VHDL identifiers, and (b) define a class of special identifiers, which are only used as reserved words, that have a final '!'.

10. Allow endpoint instances to be referenced anywhere within VHDL. These would be very useful in building reactive testbenches, in particular, and would be a simple extension of VHDL expression syntax.

11. Allow attribute specifications to apply to PSL property, sequence, and endpoint declarations, as well as labels on PSL directives.

12. Mention that elaboration of a design unit also involves elaboration of any associated verification units, as well as elaboration of any embedded PSL declarations/statements (but don't define elaboration of PSL constructs).

13. Define the updating of PSL endpoints (which are effectively implicit signals like S'Stable) to occur, in dependency order, in much the same way that implicit signals are updated, and immediately after implicit signals are updated.

14. Define execution of PSL directives to occur at the beginning of each simulation cycle, after all signals and endpoints have been updated, but prior to execution of any (non-PSL) processes.

Limitations (items deferred for now, if not forever)

1. Do not allow PSL declarations and directives in a configuration, because it would confuse two very different functions - controlling the composition of the design hierarchy on the one hand (configurations), vs. instrumenting the design hierarchy with verification monitors on the other (vunits).

2. Do not allow PSL declarations in declarative parts associated with procedural code (i.e., in a process or subprogram).

3. Don't extend VHDL concurrent assertion statements to merge with PSL assert directives. This would require explaining how 'postponed' affects a PSL assert directive, in particular a multi-cycle assertion, and that would be really messy. Instead, leave concurrent assertions as they are, and add PSL assert directives as another layer. (Sequential assertions, Concurrent assertions, PSL directives)

4. Don't attempt to define the elaboration of PSL declarations and directives.

5. Don't attempt to define PSL 'parameter passing', which is essentially text substitution rather than any of the parameter passing mechanisms that VHDL supports today.

6. Don't attempt to define 'equivalent processes' for PSL directives. It would be far too difficult to define the algorithm portion of the equivalent process, and in any case it would be redundant with the PSL LRM.

7. Later, consider allowing PSL_Expressions (e.g., a->b for Boolean a,b) and Built_In_Functions (e.g., onehot) to appear anywhere an expression is allowed in VHDL.

8. Later, consider allowing a configuration to specify which vunits are to be included. This might be accomplished by defining a new kind of configuration_item that allows a configuration to apply an unbound vunit to a particular instance in the configuration hierarchy.

9. Later, consider extending PSL vunit pathnames to allow binding of PSL vunits to nested blocks within an architecture.

10. Later, consider allowing PSL to reference shared variables.

11. Later, consider treating endpoints as just another kind of implicit signal, and therefore allow endpoints and other implicit signals to intermix - e.g., for endpoint E, allow E'delayed(T), ...

UPDATE:
~~~~~~
~~~~~~~~~~~~~~~~

Issues to be addressed elsewhere:

1. Resolve pathname syntax in PSL vs. VHDL vs. mixed-language designs

2. Define APIs for access to PSL directive state during simulation

END UPDATE:
~~~~~~
~~~~~~~~~~~~

Modifications to 1076-2002

0. Overview of this standard

Add a new clause:

0.3 Incorporation of PSL by reference

Portions of this standard refer to syntactic nonterminals and semantic concepts defined in the Accellera Property Specification Language (PSL) Language Reference Manual (LRM), version 1.1.

References to a nonterminal in the PSL LRM all begin with the qualifier "PSL_". All such references implicitly refer to the VHDL flavor of PSL.

[and anything else that needs to be said about this]

1. Design entities and configurations

1.1.2 Entity declarative part

Add PSL_declaration to the production for entity_declarative_item.

1.1.3 Entity statement part

Add PSL_directive to the production for entity_statement.

1.2.1 Architecture declarative part

Add PSL_declaration to the production for block_declarative_item.

1.2.2, Architecture statement part

No change needed, because concurrent_statement is defined elsewhere.

2. Subprograms and packages

2.5 Package declarations

Add PSL_declaration to the production for package_declarative_item.

3. Types

No change, other than possibly adding notes where appropriate to reiterate PSL LRM definitions of what is a Boolean/Bit/Integer/Vector type.

4. Declarations

In the introductory paragraph, add PSL_declaration as an alternative in the syntax for 'declaration', and explain in the following text that PSL_declaration is a reference to nonterminal PSL_Declaration in the PSL v1.1 LRM.

Add a new clause:

4.8 PSL declarations

Declarations in the Accellera Property Specification Language (PSL) may also appear in a VHDL description. Specifically, the following declarations are supported:

* default clock declaration
* sequence declaration
* endpoint declaration
* property declaration

The syntax and semantics of these declarations are defined in [ref. to PSL v1.1 LRM]. Some limitations apply to the use of these declarations in VHDL. These limitations are described in the following sections.

4.8.1 Default clock declarations

A default clock declaration may appear at most once in a given declarative region. The scope of a given default clock declaration extends from the end of the declaration to the end of the immediately enclosing declarative region.

4.8.2 Property declarations

A property declaration must declare a property that conforms to the Simple Subset of PSL.

5. Specifications

5.1 Attribute specification

Extend entity_class to include 'property', 'sequence', 'endpoint'.

Add these to par. 6 on page 76, which lists all the possible kinds of entities that can have attributes associated.

6. Names

No change.

7. Expressions (** UPDATED **)

Add 'endpoint' to the production for 'primary', after 'allocator'.

7.3 Operands (** UPDATED **)

7.3.7 Endpoints (** NEW **)

Add a production

```
Endpoint ::= PSL_Endpoint_Instance
```

Explain that an endpoint is an expression of type Boolean. Refer to the PSL LRM for more details.

8. Sequential statements

No change.

9. Concurrent statements

In the introductory paragraph, add PSL_directive as an alternative in the syntax for 'concurrent_statement'.

(** DELETE THIS **)
, and explain in the following text that PSL_directive is a reference to nonterminal PSL_Directive in the PSL v1.1 LRM.
(** END **)

Add a new clause:

9.8 PSL directives

Certain directives in the Accellera Property Specification Language (PSL) may also appear in a VHDL description. Specifically, the following directives are supported:

* assert directive
* assume directive
* restrict directive
* cover directive

The syntax and semantics of these directives are defined in [ref. to PSL v1.1 LRM]. Some limitations and interpretations apply to the use of these directives in VHDL. These limitations and interpretations are described in the following sections.

(** MOVED **)
For any given PSL directive, there is an equivalent process that implements that directive. The form and contents of the equivalent process are determined by the semantics of the VHDL flavor of PSL and are not defined in the VHDL LRM.

9.8.1 Assert directives

An assert directive must involve a property that conforms to the Simple Subset of PSL.

(** NEW **)
If a VHDL statement can be interpreted as either a VHDL concurrent assertion or as a PSL assert directive, it shall be interpreted as a VHDL concurrent assertion.

9.8.2 Assume directives

An assume directive must involve a property that conforms to the Simple Subset of PSL.

10. Scope and visibility

State that the scope of a declaration in an entity or architecture extends into a PSL verification unit bound to that entity or architecture.

May need to mention that a PSL forall property is a declarative region, and that the scope of the forall parameter extends from the point of definition to the end of that declarative region.

11. Design units and their analysis

In 11.1, add

primary_unit ::= PSL_verification_unit

and explain in the following text that PSL_verification_unit is a reference to the nonterminal Verification_Unit in the PSL v1.1 LRM.

12. Elaboration and execution

12.1 Elaboration of a design hierarchy

Add the following sentence at the appropriate place:

Elaboration of a block statement equivalent to an external block defined by a design entity that has an associated verification unit (bound either to the design entity as a whole, or to an instance of that design entity) involves first elaborating the associated verification unit. (Perhaps explain this in turn as involving insertion of the verification unit as a nested block within the body of the design entity.)

(Note that this covers both VHDL decls/stmts and PSL decls/directives in the vunit.)

Just before 12.4, add a new section something like this:

12.4 Elaboration of a PSL Endpoint Instance

Elaboration of a PSL endpoint instance consists of construction of the equivalent process statement followed by elaboration of the equivalent process statement.

12.4.4 Other concurrent statements -- now 12.5.4

Change the last paragraph to refer to PSL directives also - e.g.,

Elaboration of all concurrent signal assignment statements,
concurrent assertion statements, and PSL directives consists
of construction of the equivalent process statement followed
by elaboration of the equivalent process statement.

Add a new section 12.6.4, Updating PSL Endpoints, after 12.6.3, Updating Implicit Signals, and before the current 12.6.4, The simulation cycle. Cover the following:

- endpoint instances are monitors that generate implicit (Boolean) signals
  - a reference to an endpoint instance is a reference to the associated implicit signal
- endpoints may be dependent upon other endpoints, so execution must be ordered appropriately
  - this should parallel the discussion of dependency of implicit signals in 12.6.3

In the simulation cycle definition, add the following between steps (c) and (d):

For each process P that is equivalent to a PSL directive,
if P is sensitive to a signal S and if an event has occurred
on S in this simulation cycle, then P resumes.

and change the following entry (originally (d)) to read as follows:

For each remaining process P, if P is ...

13. Lexical Elements

13.3 Identifiers (** NEW **)

Modify definition of basic_identifier to allow trailing underscores:

basic_identifier ::=
letter [underline] { letter_or_digit [underline] }

Reflect this in paragraph following, by deleting the phrase "inserted between a letter or digit and an adjacent letter or digit".

13.3.3 PSL Identifiers (** NEW **)

Add a section that says that defines PSL identifiers as basic identifiers that may have '!' appended:

```
PSL_identifier ::=
  basic_identifier [ '!' ]
```

Explain that PSL keywords (which are reserved words in VHDL) are the only instances of PSL_Identifier - i.e., it is not legal to declare an identifier of this form.

13.9 Reserved words

Add PSL keywords to the reserved words list.

14. Predefined language environment

No changes.

Annex A. Syntax Summary

Reflect all syntax changes for PSL incorporation.

B. Glossary

Consider adding PSL-related terminology.

C. Potentially nonportable constructs

No changes.

D. Changes from IEEE Std 1076-2002

Summarize all changes made for PSL incorporation.

E. Related standards

Add a reference to Accellera PSL v1.1.

Analysis:

Resolution:

[To be performed by the 200X Assertion Working Group]

Erich Marschner, Cadence Design Systems
Senior Architect, Advanced Verification
Phone: +1 410 750 6995 Email: erichm@cadence.com
Cell: +1 410 294 2599 Email: erichm@comcast.net