5/28/2004

Proposal for VHPI model PSL assertion extensions

This proposal has been prepared by Cadence Design Systems, Inc. for consideration by the IEEE 1076 working group for inclusion in the next revision of the IEEE 1076 standard.

Contents

1 Introduction 4

1.1 Overview 4

1.2 Scope 4

2 VHPI model UML notation....5

2.1 UML notation quick reference 5

2.2 VHPI interface interpretation of the model 6

3 VHPI assertion class diagram....8

4 VHPI_user.h header file modifications 9

1 Introduction

1.1 Overview

[1] It is expected that the Property Specification Language standard (PSL) [1] will be adopted and included by reference in the VHDL IEEE 1076 standard [2]. This will add assertion verification capabilities to the VHDL Hardware description language. The objective of this proposal is to provide VHPI extensions to access the PSL assertions which will be adopted by the VHDL standard.

[2] The VHPII model described in this proposal constitutes an initial subset of the PSL assertion access. We expect that this access will be updated and revised accordingly with the part of the PSL language which will be accepted by the VHDL 1076 committee.

[3] This proposal is divided in four sections. The first section describes the scope and purpose of this proposal. The second section is a quick reference guide to the formal graphical notation used by the VHPI diagrams. The third section provides the VHPI diagram to access PSL assertions, and the fourth section details the VHPI standard header file additions necessary to provide access to PSL assertions.

1.2 Scope

[4] It is assumed that PSL assertions will be part of the VHDL source and would provide the basis of VHDL assertion syntax and semantics. In a similar manner, the VHPI model for accessing PSL assertions coexists with the VHPI model to access a VHDLinstantiated design. The proposed VHPI extensions include traversal of all PSL assertions declared within a design unit (library informal model context) or instance (elaborated information model context) and queries of certain characteristics of these assertions. This proposal is intended to provide the minimal required access for co-simulation and debugging tools.

2 VHPI model UML notation

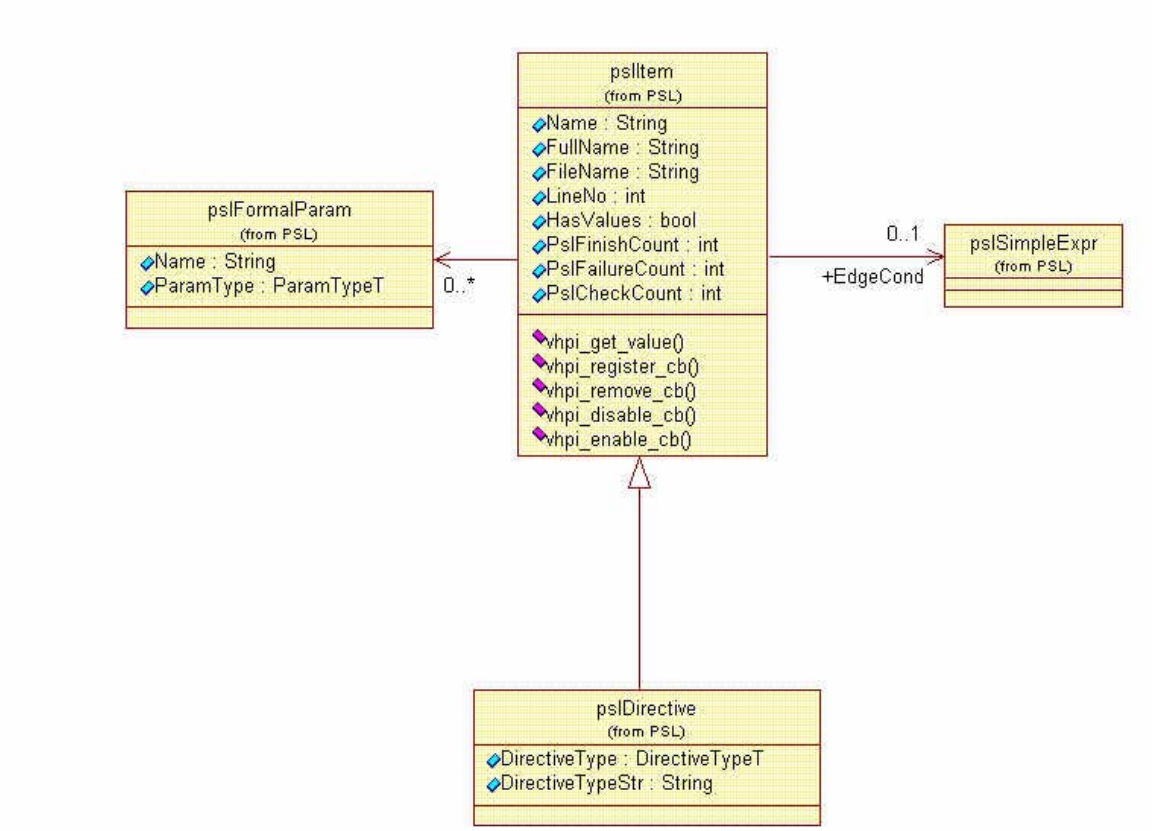
[5] The Unified Modeling Language (UML, [3]) is used to describe the VHPI information model. UML is a formal graphical language. It defines a rigorous notation and a meta model of the notation (diagrams) that can be used to describe object-oriented software design. UML is an OMG standard (Object Management Group) which is being proposed to the International Organization for Standards (ISO). The following sub sections provide a quick reference guide to interpret the VHPI diagrams. For a more complete specification of the UML notation, consult [3].

2.1 UML notation quick reference

[6] *Class diagrams*

[7] We use the class diagram technique of UML to express the VHPI information model. A class diagram specifies the VHPI class types and the way classes are connected together. In UML, class inheritance is denoted by a hollow arrow directed towards the parent class.

[8] *A class*



[9] *A derived class*

[10] An expanded class shows two compartments, the top one displays the **properties** with their names and return type, the bottom one displays the **operations** that are defined for this class. Properties and operations

inherited from parent classes may not appear in the compartment boxes of the derived classes but are available for all derived classes. In the example above, many properties, for example “PslFinishCount”, “PslFailureCount” are defined for the “pslItem” class. Many operations, for example “vhpi_get_value”, vhpi_register_cb” are defined on the “pslItem” class. Two additional properties “DirectiveType” and “DirectiveTypeStr” are defined for the class “pslDirective” which are not available for the parent class “pslItem”.

[11] The link between the “pslDirective” class and the “pslItem” class shows the **inheritance** between a derived class and its parent class. A derived class inherits properties and operations from its parent classes. The hollow arrow points to the parent class.

[12] *Associations*

[13] Relationships between classes are called associations and are denoted by straight lines between classes. Associations have descriptive parameters such as multiplicity, navigability and role names.

[14] **Associations** are links between classes that depict their inter-relationships.

[15] Navigability, multiplicity and role names can be used to further describe the relationship.

Navigability expresses the direction of access and is represented by an arrow. An association can be bi-directional in which case arrows may be shown at both ends.

[16] **Multiplicity** expresses the type of relationship between the classes: singular (one, zero or one), multiple (zero or more, one or more) and is represented by numbers at the end of the association to which it applies. It can be one the following:

[17] 1 for access to one object handle (singular relationship)

[18] 0..1 for access to zero or one object handle (singular relationship)

[19] 0..* for access to zero or more object handles of the same class (iteration relationship)

[20] 1..* for access to one or more object handles of the same class (iteration relationship)

[21] A **role name** is a tag name on one end of the association. It may be used to indicate more precisely the relationship or to distinguish this relationship from another relationship that leads to an object of the same class. In the figure above, “EdgeCond” is the name of the relation that accesses an object of class “pslExpr” from an object of class “pslItem”. The relationship it denotes is an iteration relationship.

[22] In the diagrams, the following convention is used: if a role name is not specified, the method name for accessing the object pointed by the arrow is the target class name. From the “pslItem” class, zero or more objects of the “pslFormalParam” class can be obtained, the default method name is “pslFormalParams”.

2.2 VHPI interface interpretation of the model

When interpreting the VHPI class diagrams, “VHPI” must be added as a prefix to any class, property, method or operation name in order to obtain the standard defined constant listed in the VHPI standard header file (vhpi_user.h).

A VHPI iteration (also called one-to-many method) is modeled by an association with a multiplicity of either zero or more (0..*), or one or more (1..*) to indicate that the iteration may contain zero handles or will contain at least one handle. In order to traverse iteration relationships, use vhpi_iterator() and

vhpi_scan(). The direction or navigability indicates the class of the handles created by the iteration. In the example above, we show that there is a one-to-many relationship between a “pslItem” class and a “pslFormalParam” class.

A VHPI singular (also called one-to-one method) will be represented by a navigable association with a multiplicity of one (1) if the method always returns a handle of the destination class or a multiplicity of zero or one (0..1) if the method may not return a handle. In order to traverse a singular relationship, use vhpi_handle(). In the example above, the diagram shows a one-to-one relationship that allows traversal from a “pslItem” class to the “pslExpr” class.

Note that the diagrams only express the possible access flow and not all access is presented in a single diagram.

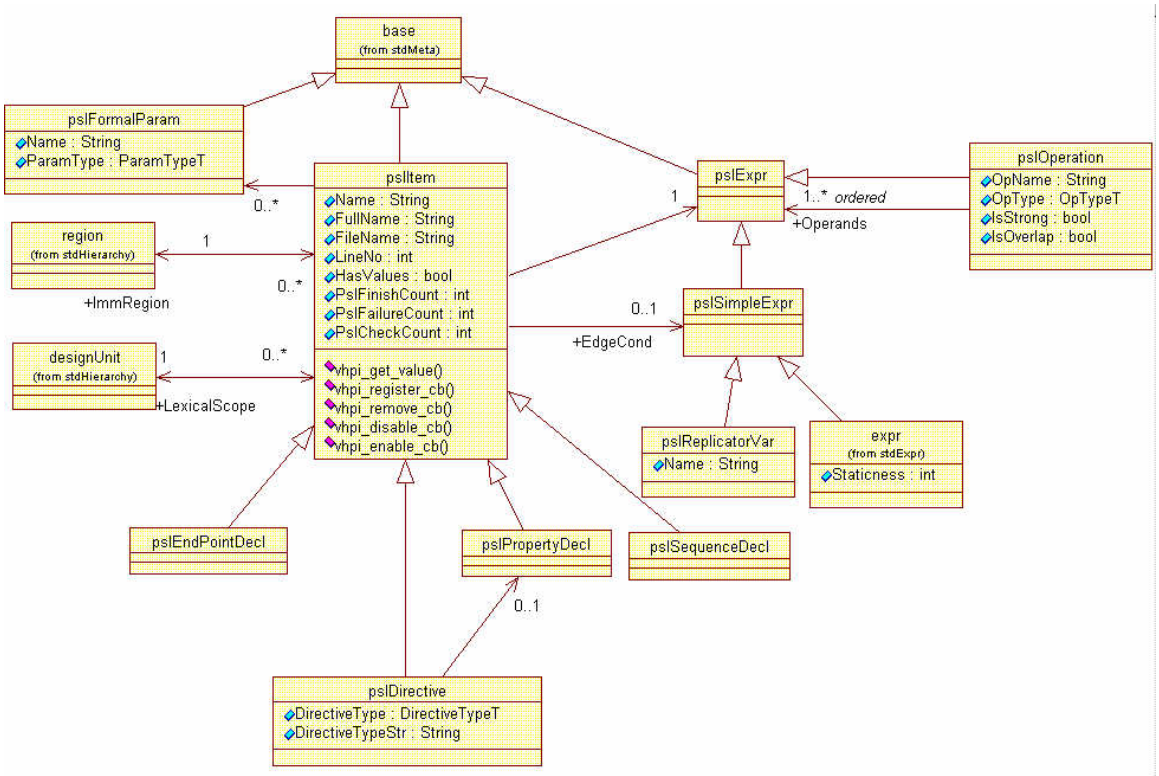
A VHPI property which appears in the top compartment of a class can be queried with one of the following VHPI interface functions:

vhpi_get() for a boolean or integer property,

vhpi_get_str() for a string property.

Additional VHPI functions can be available for a certain class and are listed in the bottom compartment of the class. Such functions are for example be vhpi_get_value() or vhpi_put_value().

3 VHPI PSL class diagram



4 PSL related additions to the vhpi_user.h header file

[23] Below are the additional objects types, relationships, properties and properties constant values which should inserted in the vhpi_user.h file in order to provide assertion access. The constant values attributed to the defined constants were chosen to be outside the range used currently by VHPI. A range of values should be reserved for the VHPI assertion access and that range should be large enough to allow future extensions. The range and values used in the following header file are temptative.

```
#ifndef PSL_CLASSES
#define PSL_CLASSES,
/* Object types */
      vhpiPslSequenceDecl      1252,  /* PSL Named Sequence Declaration
                                        (LRM Sect 6.1.2) */

      vhpiPslEndPointDecl      1254,  /* PSL Named Endpoint
                                         Declaration (LRM Sect 6.1.3) */
      vhpiPslPropertyDecl      1255,  /* PSL Named Property Declaration
(LRM Sect 6.2.4) */
      vhpiPslDirective   1257  /* PSL Verification Directives (LRM Sect
7.1) */
#endif

#ifndef PSL_MANY_METHODS
#define PSL_MANY_METHODS,
/* Relationships */

/* One to many relationship from a vhpi region or designUnit handle to
   pslItems


   Iteration returns handles of type:
   vhpiPslEndPointDecl, vhpiPslPropertyDecl,  vhpiPslSequenceDecl,
vhpiPslDirective
*/
       vhpiPslItems           1703
#endif

#ifndef PSL_ONE_METHODS
#define PSL_ONE_METHODS,
/* One to one relationship from a pslItem
   to a pslExpr
   pslItem --EdgeCond-> pslExpr

   (vhpiPslEndPointDecl, vhpiPslPropertyDecl,
    vhpiPslSequenceDecl, vhpiPslDirective)  --EdgeCond-> pslExpr

   Returned handle is the HDL clock expression.
*/
       vhpiEdgeCond         1455
#endif

/* Properties */
#ifndef PSL_INT_PROPERTIES
#define PSL_INT_PROPERTIES,
/* Integer properties for derived classes of vhpiPslItem
```

```
   Retrieve with vhpi_get()
*/
       vhpiPslFinishCount   1202, /* Number of times a property has
reached the state of vhpiAssertFinished */
       vhpiPslFailureCount  1203, /* Number of times a property has
reached the state of vhpiAssertFailed */
       vhpiPslCheckCount    1204, /* Number of times a property has been
checked */


/* vhpiDirectiveType -- Integer property for handles of type
vhpiPslDirective.
   Retrieve with vhpi_get() - Returned values defined by
vhpiDirectiveTypeT.
*/
       vhpiDirectiveType  1207,
       vhpiParamType      1211  /* The parameter type one of the values
defined by vhpiParamTypeT */
#endif

/* Enumeration constants and defined constants */
#ifndef PSL_ENUMTYPES
#define PSL_ENUMTYPES


/* Directive type codes */
typedef enum {
      vhpiDirAssert              = 1, /* PSL Assert Directive (LRM Sect
7.1.1) */
      vhpiDirAssume              = 2, /* PSL Assume Directive (LRM Sect
7.1.2) */
      vhpiDirCover               = 6, /* PSL Cover  Directive (LRM Sect
7.1.6) */
      vhpiDirRestrict            = 8, /* PSL Restrict  Directive (LRM
Sect 7.1.6) */
} vhpiDirectiveTypeT;

/* Assertion state value encodings
To query, use vhpi_get_value() with reference handle of type
vhpiPslPropertyDecl and integer or string format (vhpiIntVal or
vhpiStringVal) */

#define vhpiAssertInactive  1  /* There are currently no partial matches
of the sequence of conditions described by the property */

#define vhpiAssertActive    2  /* The first term of the enabling
condition is satisfied, and the property has not finished or failed */

#define vhpiAssertFinished  3  /* The fulfilling condition has evaluated
to true, or the property has terminated without failing. */

#define vhpiAssertFailed    4  /* The fulfilling condition has evaluated
to false. */

#define vhpiAssertDisabled  5  /* The property is disabled, and is not
being checked. */

/* enumeration type for returned values of vhpiParamTypeP property */
typedef enum {
      vhpiConstParamType,
```

```
      vhpiBoolParamType,
      vhpiPropertyParamType,
      vhpiSequenceParamType
}vhpiParamTypeT;
#endif

#ifndef PSL_STR_PROPERTIES
#define PSL_STR_PROPERTIES,
      vhpiDirectiveTypeStr 1550, /* The directive type of the
pslDirective */
#endif
```

Annex A: References

[1] Accellera Property Specification Language Reference Manual version 1.01, approved Accellera standard.

[2] IEEE Std 1076-2002, IEEE Standard VHDLHardware Description Language.

[3] OMG UML Unified Modeling Language v. 1.3, Object Management Group, June 1999.