# Draft Proposal for Unifying Path Names

Peter Ashenden

30 April 2004

## 1. Introduction

Path names occur in various forms in VHDL, both as it stands in IEEE Std 1076-2002 and in proposed revisions:

1. In expanded names
2. In the 'path_name and 'instance_name attributes
3. In the DefName and FullName prperties proposed for VHPI
4. In proposed cross module references (XMRs)
5. In proposed PSL descriptions

Path names are used for two purposes:

- As an informative identifier for a named entity
- To uniquely identify a named entity

It would be desirable to have a uniform syntax for all of these path names, while preserving backward compatibility as much as is possible. Current proposals for VHPI and XMRs are based on the path name syntax used in the 'path_name attribute. However, use of the colon character as a delimiter is problematic for PSL, which needs to deal with mixed VHDL/Verilog descriptions. A similar problem may arise with other occurrences of path names in a mixed-language design. Expanded names use the dot character as a delimiter, which is inconsistent with the 'path_name attribute.

This proposal examines the forms of path name that are required in different contexts and suggests a syntax that is uniform across those contexts. Consideration is also given to interoperability with Verilog and mixed VHDL/Verilog designs.

## 2. Path Name Contexts

Three contexts for path names have been identified:

1. Library unit path names

   This form of name identifes a named entity in an uninstantiated design unit that resides in a library. The named entity may be the design unit itself, or a named entity nested within one or more declarative regions.

2. Design hierarchy path names

   This form of name identifies an instance of a named entity in the elaborated design hierarchy. The named entity may be the root design entity itself; or, recursively, a named entity declared within the design entity, a named entity declared with a nested declarative region, a named entity declared within an instance of a protected type, or a named entity declared within a design entity bound to a component instantiation statement.

3. Wildcard path names

This form of name identifies zero or more instances of a named entity in the elaborated design hierarchy. It consists of a head part that denotes a design entity in a library and a tail part that denotes a path to an instance of a named entity in any instance of the design entity.

In addition to supporting these contexts, a name syntax should support specification of relative path names, with the name of a declarative region or region instance giving the scope for interpretation of the relative name.

### 2.1. Expanded names

Expanded names occur in context (1) and are used as unique identifiers of named entities. They are relative path names, interpreted relative to the innermost visible occurrence of the first element of the path name. For example, an expanded name A.B.C.D is interpreted by finding the innermost visible occurrence of A, then treating B.C.D as a relative path from the region denoted by A.

### 2.2. Path_name and Instance_name attributes

The 'path_name and 'instance_name attributes occur in context (2) and are used as informative identifiers of named entities. They don't uniquely identify an instance of a named entity. There are at least two forms of ambiguity. First, if a library containing a package in the design hierarchy has the same name as the root design entity, the 'path_name attributes for two named entities can be the same. For example ":L:P:X" might be a named entity X in a package P in library L, or a named entity X in a region P in the root design entity L. Second, for a named entity in a subprogram, there is no way to distinguish between different instances occurring in different activations of the subprogram. Similarly, for a named entity in a protected type body, there is no way to distinguish between different instances occurring in different shared variables of the protected type.

### 2.3. VHPI name properties

The DefName property in the current VHPI proposal occurs in context (1) and is intended to uniquely identify a named entity. It is of the form

@ <library_name>
. <primary_unit_name> [ : <architecture_name> | body ]
{ . <item_name> [ <signature> ] }

The combination of the leading '@' character and the use of '.' delimiters indicates that it is a library unit path name.

The FullName property in the current VHPI proposal occurs in context (2) and is intended to uniquely identify a named entity. It has various forms. If the named entity is statically elaborated and occurs within an elaborated package, the path name is of the form

@ <library_name> :
<package_name> :
[ <shared_variable_name> : ]
[ <item_name> [ <signature> ] ]

The shared variable name is included if the named entity is elaborated as part of a shared variable of protected type, where the shared variable is declared in the package.

If the named entity is statically elaborated and occurs within the design hierarchy other than in a package, the path name is of the form

: <root_entity_name > :
{ <region_name> : }
[ <item_name> [ <signature> ] ]

Each <region_name> is the label of a statement (block, generate, process, component instantiation) or shared variable of protected type in the hierarchy between the root and the named entity. In the case of a for-generate, the <region_name> includes the value of the generate parameter.

If the named entity is dynamically elaborated (ie, it is declared within a subprogram that is activated), the path name is of the form

<ancestor_full_name> : { <subprogram_name> : }
[ <item_name> [ <signature> ] ]

where the <parent_full_name> is the hierachical path name to the process from which the subprogram was called, and each <subprogram_name> is the name of a subprogam on the call chain from the process to the named entity. This form of path by itself does not uniquely identify a named entity, since it does not take account of overloaded versions of subprograms. The VHPI specification allows for augmenting the subprogram names with signatures in a search path used to locate a dynamically elaborated named entity.

For both the DefName and FullName properties, implicitly defined labels are used for unlabelled statements whose labels are needed as path name elements. The labels are of the form _Pn for process statements and _Ln for for-loop statements.

The VHPI name properties also provide for identifying an element of an array or record or a slice of a record by appending an index, element name or range using the same syntax as VHDL names. Indices and range bounds have to be literals.

### 2.4. XMR names

XMR names occur in context (2) and are intended to be a reference from one location in a design hierarchy to a named entity elaborated in another part of the design hierarchy. They are limited to references to statically elaborated objects of class signal, variable and constant (thus including ports and generics).

The requirements for XMR names include both relative and absolute path names. In the case of relative path names, there is a requirement for up-level references as well as references to named entities lower in the design hierarchy than the location of the reference. There is a requirement for reference to named items in elaborated packages.

### 2.5. PSL names

*Erich to advise...*

## 3. Abstract syntax for path names

### 3.1. Library unit path name

A library unit path name that needs to uniquely identify a named entity can be formed from the following elements:

<library_name>
  <primary_unit_name> [ <secondary_unit_name> ]
  { <item_name> [ <signature> ] }

The <library_name> is the logical name of the library containing the design unit. The <primary_unit_name> is the either the simple name of the entity if the named entity is declared in an entity declartion or architecture body, or the simple name of the package if the named entity is declared in a package declaration or body. The <secondary_unit_name> is either the simple name of the architecture if the named entity is declared in an architecture body, or an indicator of some sort if the named entity is declared in a package body. The sequence of item names are names of nested declarative regions within which the named entity is declared. If any of those is a subprogram, the signature is included. The simple name of the named entity (if the named entity is other than the design unit itself) is the last item name in the sequence.

### 3.2. Design Hierarchy path name

A design hierachy path name that needs to uniquely identify an instance of a named entity must include elements that identify parent region instances, whether those instances be statically or dynamically elaborated.

For a statically elaborated named entity declared within a package, or instantiated as part of a shared variable that is declared within a package, the elements in the path name are

<library_name>
  <package_name>
  [ <region_name> ]
  [ <item_name> [ <signature> ] ]

The <library_name> is the logical name of the library containing the package, and the <package_name> is the simple name of the package. The <region_name> is included if the named entity is within a shared variable; in that case, the <region_name> is the simple name of the shared variable. The simple name of the declared entity (if the named entity is other than the package itself) is the <item_name> at the end of the path name.

For a statically elaborated named entity declared within a design entity, or instantiated as part of a shared variable that is declared within a design entity, the elements of the path name are

<root_entity_name>
  { <region_name>}
  [ <item_name> [ <signature> ] ]

Each <region_name> is the label of a statement (block, generate, process, component instantiation), or simple name of a shared variable, in the hierarchy between the root and the named entity. In the case of a for-generate statement, the <region_name> includes the value of the generate parameter. In the case of an unlabeled process statement, the <region_name> is a surrogate identifier, unique in the region in which the process statement occurs, but deterministically generated so that the process can be identified. The simple name of the declared entity (if the named entity is other than the root entity ) is the <item_name> at the end of the path name.

For a dynamically elaborated named entity, a design hierarchy path only has validity for the lifetime of the named entity. At that time, there is a statically elaborated ancestor region from which dynamic elaboration commenced. The ancestor region is either a statically elaborated declarative region, if the subprogram is called as part of elaborating a declaration or an interface element in a port map or generic map; or it is a process statement or equivalent, if the program is

called from a statement. In either case, the design hierarchy path name of the ancestor can be used as a head part of the named entity's path name to uniquely identify the site at which dynamic elaboration commenced. There can be at most one active site at a time for a given potential ancestor, since elaboration of a declarative region involves elaborating declarations in order, and execution of statments in a process occurs sequentially.

For a given site in an ancestor, there may be multiple activations of the subprogram containg the named entity (since subprograms may be recursive), so some indication destinquishing among them is required. (In the VHPI FullName property, this is the call chain.)

Putting all of this together, the elements of the path name are

<ancestor_path_name> { <activation_indicator> }
[ <item_name> [ <signature> ] ]

Note that this is different from the 'path_name attribute, in which the path given leads to the declarative region in which the subprogram is declared. The site at which dynamic elaboration starts might be further nested within concurrent statements in that region (in which the subprogram is visible), leading to a longer path to the dynamic elaboration site.

Considering the activation indicator, it would seem desirable to use the name of the subprogram containing the named entity as part of the indicator, since that would help make the path name intelligible to the human user. However, there may be several overloaded versions of the subprogram name. Furthermore, there may be different subprograms with the same name and signature visible in different subprograms invoked in the call chain leading to the named entity. Thus, in general, the full call chain of subprogram names with signatures included would be needed as the activation indicator. At each level in the call chain, the subprogram name would be interpreted as denoting the subprogram of the given name visible in the region of the subprogram at the preceding level

For deeply nested call chains, this form of activiation indicator would become very cumbersome. An alternative form is the numerical depth of the call chain. This was originally used in the VHPI FullName property, but subsequently superceded by the chain of names.

In the case of a named entity dynamically elaborated from a protected type method, the subprogram name used in the activation indicator would have to be the selected name of the method, with the prefix being the name (possibly an expanded name) of the shared variable, visible at the given level of the call chain, whose method is invoked. If this is seen asdifficult to implement or interpret, an alternative path name for named entities dynamically elaborated from protected type methods could use the path name of the shared variable as the ancestor full name, since at most one method of that variable can be active at a time. The activation indicator would then be the call chain (or call chain depth) starting from the activation of the method.

In order to support VHPI properties that refer to subelements, slices and attributes of named entities, further forms of design hierarchy path name are needed.

For a subelement of an array, the elements of the path name are

<array_path_name> <index_value> { <index_value> }

where each <index_value> is a literal value, one per index range of the array, of the type of the corresponding index type.

For a subelement of a record, the elements of the path name are

<record_path_name> <element_simple_name>

For a slice of an array, the elements of the path name are

<array_path_name> <left_bound_value> <direction> <right_bound_value>

For an attribute of a named entity, e.g., an implicit signal such as s'delayed(n), the elements of the path name are

<prefix_path_name> <attribute_simple_name> { <actual_parameter_value> }

where <actual_parameter_value> elements may be included if the attribute is a function. Each value is a literal value of the type of the correponding function parameter. Multiple parameters are allowed in order to cater for VHDL-AMS attributes.

### 3.3. Wildcard path names

*Erich to review and advise...*

A wildcard path name can be formed from the following elements

<library_name>
<entity_name> [ <architecture_name> ]
{ <region_name>}
[ <item_name> [ <signature> ] ]

The is a hybrid of a library unit and a design hierachy path name, and matches zero or more instances of a named entity in a design hierarchy. The <library_name> and <entity_name> specify an entity declaration; the wildcard path name only matches named entities for which the specified entity declaration is an ancestor in the design hierachy. If the <architecture_name> is included, matching is further constrained to require the design entity formed by the entity and the architecture to be an ancestor.

The remainder of the path specifies a chain of containing regions in the design hierarchy down to a named entity. The path matches any named entity whose simple name is the same as the item name and which is contained in the specified chain of regions starting from an instance of the specified design entity. If the chain of regions and the <item_name> is omitted, the path matches just the design entity instance.

### 3.4. Relative path names

Relative path names can be constructed based on the abstract syntax for absolute path names. A relative path name needs to be interpreted with respect to a reference location. The application will determine whether the relative name is interpreted in the library unit context, design hierarchy context or wildcard context. Specifically,

- Expanded names always relative, and are interpreted in the library context. The first element of an expanded name denotes an enclosing declaration, based on scope and visibility rules, and the remainder of the expanded name is interpreted relative to that declaration.
- VHPI relative path name search names are interpreted based on a context supplied to the lookup function. That context is either a library unit or design hierarchy path name. The search name is appended to the context name to form a complete path name.
- XMR relative path names are interpreted in the design hierarchy context. The relative name is appended to the path name of the region in which the relative name occurs.

  *Steve: Is this right?*

- PSL relative names are interpreted in the wildcard context.

  *Erich: Are there relative names in PSL? If so, what is the point of reference?*

## 4. Concrete syntax

In developing a concrete syntax for path names, a number of issues need to be considered:

1. Distinction between absolute and relative names.

   This can be done on the basis of presence or absence of a leading delimiter: presence of a leading delimiter indicating an absolute name and absence indicating a relative name. VHPI search path names take this approach.

   *Strawman:* Use a leading delimiter for absolute names and no leading delimiter for relative names.

2. Distinction between library unit and design hierarchy names.

   VHPI as currently proposed makes the distinction based on the separator between path elements: ‘.’ for library unit names and ‘:’ for design hierarchy names. The distinction is intended to mirror the use of ‘.’ for expanded names and ‘:’ for 'path_name attributes. The scheme is complicated, however, by the use of ‘:’ as a delimiter for the secondary unit name in a library unit path name.

   An alternative could be to use different leading delimiters to make the distinction.

   *Strawman:* Use a leading ‘.’ delimiter for absolute library unit path names.

3. For library unit names, choice of delimiters for secondary unit name and indicator for package body.

   VHPI marks the secondary unit name by preceding it with a ‘:’ delimiter. For a package body, the secondary unit name is the reserved word “body”.

   In other places in VHDL that refer to an entity/architecture pair, the architecture name is parenthesized and follows the entity name. This could be generalized to packages by using the reserved word “body” in parentheses after the package name. An approach using this form of delimiter would be natural for VHDL designers, and would avoid the overloaded use of ‘:’ as a delimiter in the current VHPI path name formats.

   *Strawman:* Use the parenthesized architecture simple name for architecture bodies, and the reserved word “body” in parentheses for package bodies.

4. For design hierarchy names, distinction between package-based and entity-based names.

   The 'path_name attribute currently makes no distinction, and this is a source of ambiguity.

   VHPI uses a leading ‘:’ delimiter to indicate an entity-based name and ‘@’ to indicate a package-based name. However, this is complicated by the use of ‘@’ for the leading delimiter in a library unit path name, irrespective of whether the design unit is a package or entity.

   An alternative could be to use distinct leading delimiters, both of which are distinct from a leading delimiter for library unit path names.

   *Strawman:* Use a leading ‘:’ delimiter for entity-based design hierarchy path names, and a leading ‘@’ delimiter for package-based design hierarchy path names.

5. Choice of separator.

As indicated above, VHPI uses '.' for library unit names and ':' for design hierachy names, motivated by the separators used in expanded names and 'path_name attributes, respectively.

The PSL committee currently uses '.' for wildcard names (a form of design hierarchy name) in order to avoid conflict with the use of ':' for ranges in Verilog.

*Strawman:* To be determined, based on an analysis of the trade-off between interoperability with Verilog/PSL and campatibility with legacy code using prototype VHPI implementations.

6. Choice of surrogate identifier for unlabelled process statements and loop statements.

   VHPI currently specifies identifiers of the form _P0, _P1, etc, for processes within a given declarative region, numbered in the same order as their occurrence in the region. Similarly, identifiers of the form _L0, _L1, etc., are specified for loop statements in a statement part.

   An alternative is to leave the form of surrogate unspecified (and hence, non-portable), and encourage designers to label statements that they want to refer to in path names.

   *Strawman:* Since VHPI specifies a surrogate scheme, adopt the same.

7. Choice of form for subprogram activation indicators.

   *Erich: Is the following so?*

   VHPI is the only place where this issue arises. Hence, the issue is not considered for path names in general.

8. Choice of syntax for subelements, slices and attributes of named entities.

   *Erich: How is this dealt with in PSL?*

   In VHPI, VHDL syntax is adopted for these forms of names. The syntax for values is constrained to that of literals of the appropriate types.

   *Strawman:* Adopt VHDL syntax, with values constrained to be literals of the appropriate types.

9. Combatibility with Verilog and mixed-language designs.

   The main aspect of this issue that has been raised to date is the choice of separator (see above). Other issues are yet to be identified.

   *Strawman:* To be determined, based on an analysis of the trade-off between interoperability with Verilog/PSL and campatibility with legacy code using prototype VHPI implementations.

## 5. Examples

The purpose of this section is to illustrate the various forms of path names based on choices made for concrete syntax.

*To be completed as an exercise in use-case analysis...*