# ~~Section~~ Clause[1] 5

# Specifications

This ~~section~~ clause[2] describes *specifications*, which may be used to associate additional information with a VHDL description. A specification associates additional information with a named entity that has been previously declared. There are three kinds of specifications: attribute specifications, configuration specifications, and disconnection specifications.

A specification always relates to named entities that already exist; thus a given specification must either follow or (in certain cases) be contained within the declaration of the entity to which it relates. Furthermore, a specification must always appear either immediately within the same declarative part as that in which the declaration of the named entity appears, or (in the case of specifications that relate to design units or the interface objects of design units, subprograms, or block statements) immediately within the declarative part associated with the declaration of the design unit, subprogram body, or block statement.

## 5.1 Attribute specification

An attribute specification associates a user-defined attribute with one or more named entities and defines the value of that attribute for those entities. The attribute specification is said to *decorate* the named entity.

attribute_specification ::=
    **attribute** attribute_designator **of** entity_specification **is** expression ;

entity_specification ::=
    entity_name_list : entity_class

entity_class ::=
      **entity** | **architecture** | **configuration**
    | **procedure** | **function** | **package**
    | **type** | **subtype** | **constant**
    | **signal** | **variable** | **component**
    | **label** | **literal** | **units**
    | **group** | **file**

entity_name_list ::=
      entity_designator { , entity_designator }
    | **others**
    | **all**

entity_designator ::= entity_tag [ signature ]

entity_tag ::= simple_name | character_literal | operator_symbol

1. To conform to IEEE rules.
2. To conform to IEEE rules.

The attribute designator must denote an attribute. The entity name list identifies those named entities, both implicitly and explicitly defined, that inherit the attribute, as described below:

— If a list of entity designators is supplied, then the attribute specification applies to the named entities denoted by those designators. It is an error if the class of those names is not the same as that denoted by the entity class.

— If the reserved word **others** is supplied, then the attribute specification applies to named entities of the specified class that are declared in the immediately enclosing declarative part, provided that each such entity is not explicitly named in the entity name list of a previous attribute specification for the given attribute.

— If the reserved word **all** is supplied, then the attribute specification applies to all named entities of the specified class that are declared in the immediately enclosing declarative part.

An attribute specification with the entity name list **others** or **all** for a given entity class that appears in a declarative part must be the last such specification for the given attribute for the given entity class in that declarative part. ~~No~~ It is an error if a[3] named entity in the specified entity class ~~may be~~ is[4] declared in a given declarative part following such an attribute specification.

If a name in an entity name list denotes a subprogram or package, it denotes the subprogram declaration or package declaration. Subprogram and package bodies cannot be attributed.

An entity designator that denotes an alias of an object is required to denote the entire object, not a member of an object.

The entity tag of an entity designator containing a signature must denote the name of one or more subprograms or enumeration literals. In this case, the signature must match (see 2.3.2) the parameter and result type profile of exactly one subprogram or enumeration literal in the current declarative part; the enclosing attribute specification then decorates that subprogram or enumeration literal.

The expression specifies the value of this attribute for each of the named entities inheriting the attribute as a result of this attribute specification. The type of the expression in the attribute specification must be the same as (or implicitly convertible to) the type mark in the corresponding attribute declaration. If the entity name list denotes an entity ~~interface~~ declaration[5], architecture body, or configuration declaration, then the expression is required to be locally static (see 7.4).

An attribute specification for an attribute of a design unit (i.e., an entity ~~interface~~ declaration[6], an architecture, a configuration, or a package) must appear immediately within the declarative part of that design unit. Similarly, an attribute specification for an attribute of an interface object of a design unit, subprogram, or block statement must appear immediately within the declarative part of that design unit, subprogram, or block statement. An attribute specification for an attribute of a procedure, a function, a type, a subtype, an object (i.e., a constant, a file, a signal, or a variable), a component, literal, unit name, group, or a labeled entity must appear within the declarative part in which that procedure, function, type, subtype, object, component, literal, unit name, group, or label, respectively, is explicitly or implicitly declared.

For a given named entity, the value of a user-defined attribute of that entity is the value specified in an attribute specification for that attribute of that entity.

It is an error if a given attribute is associated more than once with a given named entity. Similarly, it is an error if two different attributes with the same simple name (whether predefined or user-defined) are both associated with a given named entity.

---

3. IR1000.4.7.
4. IR1000.4.7.
5. Terminological correction.
6. Terminological correction.

An entity designator that is a character literal is used to associate an attribute with one or more character literals. An entity designator that is an operator symbol is used to associate an attribute with one or more overloaded operators.

~~The decoration of a named entity that can be overloaded attributes all named entities matching the specification already declared in the current declarative part.~~ If the entity tag is overloaded and the entity designator does not contain a signature, all named entities already declared in the current declarative part and matching the specification are decorated.[7]

If an attribute specification appears, it must follow the declaration of the named entity with which the attribute is associated, and it must precede all references to that attribute of that named entity. Attribute specifications are allowed for all user-defined attributes, but are not allowed for predefined attributes.

An attribute specification may reference a named entity by using an alias for that entity in the entity name list, but such a reference counts as the single attribute specification that is allowed for a given attribute and therefore prohibits a subsequent specification that uses the declared name of the entity (or any other alias) as the entity designator.

An attribute specification whose entity designator contains no signature and identifies an overloaded subprogram or enumeration literal[8] has the effect of associating that attribute with each of the designated overloaded subprograms or enumeration literals[9] declared within that declarative part.

*Examples:*

```
attribute PIN_NO of CIN: signal is 10;
attribute PIN_NO of COUT: signal is 5;
attribute LOCATION of ADDER1: label is (10,15);
attribute LOCATION of others: label is (25,77);
attribute CAPACITANCE of all: signal is 15 pF;
attribute IMPLEMENTATION of G1: group is "74LS152";
attribute RISING_DELAY of C2Q: group is 7.2 ns;
```

NOTES

1—User-defined attributes represent local information only and cannot be used to pass information from one description to another. For instance, assume some signal X in an architecture body has some attribute A. Further, assume that X is associated with some local port L of component C. C in turn is associated with some design entity E(B), and L is associated with E's formal port P. Neither L nor P has attributes with the simple name A, unless such attributes are supplied via other attribute specifications; in this latter case, the values of P'A and X'A are not related in any way.

2—The local ports and generics of a component declaration cannot be attributed, since component declarations lack a declarative part.

3—If an attribute specification applies to an overloadable named entity, then declarations of additional named entities with the same simple name are allowed to occur in the current declarative part unless the aforementioned attribute specification has as its entity name list either of the reserved words **others** or **all**.

4—Attribute specifications supplying either of the reserved words **others** or **all** never apply to the interface objects of design units, block statements, or subprograms.

5—An attribute specification supplying either of the reserved words **others** or **all** may apply to none of the named entities in the current declarative part, in the event that none of the named entities in the current declarative part meet all of the requirements of the attribute specification.

---

7. IR1000.4.4.
8. IR1000.3.3.
9. IR1000.3.3.

## 5.2 Configuration specification

A configuration specification associates binding information with component labels representing instances of a given component declaration.

configuration_specification ::=
**for** component_specification binding_indication ;

component_specification ::=
instantiation_list : *component*_name

instantiation_list ::=
*instantiation*_label { , *instantiation*_label }
| **others**
| **all**

The instantiation list identifies those component instances with which binding information is to be associated, as defined below:

— If a list of instantiation labels is supplied, then the configuration specification applies to the corresponding component instances. Such labels must be (implicitly) declared within the immediately enclosing declarative part. It is an error if these component instances are not instances of the component declaration named in the component specification. It is also an error if any of the labels denote a component instantiation statement whose corresponding instantiated unit does not name a component.

— If the reserved word **others** is supplied, then the configuration specification applies to instances of the specified component declaration whose labels are (implicitly) declared in the immediately enclosing declarative part, provided that each such component instance is not explicitly named in the instantiation list of a previous configuration specification. This rule applies only to those component instantiation statements whose corresponding instantiated units name components.

— If the reserved word **all** is supplied, then the configuration specification applies to all instances of the specified component declaration whose labels are (implicitly) declared in the immediately enclosing declarative part. This rule applies only to those component instantiation statements whose corresponding instantiated units name components.

A configuration specification with the instantiation list **others** or **all** for a given component name that appears in a declarative part must be the last such specification for the given component name in that declarative part.

The elaboration of a configuration specification results in the association of binding information with the labels identified by the instantiation list. A label that has binding information associated with it is said to be *bound*. It is an error if the elaboration of a configuration specification results in the association of binding information with a component label that is already bound, unless the binding indication in the configuration specification is an incremental binding indication (see 5.2.1). It is also an error if the elaboration of a configuration specification containing an incremental binding indication results in the association of binding information with a component label that is already incrementally bound[10].

NOTE

—A configuration specification supplying either of the reserved words **others** or **all** may apply to none of the component instances in the current declarative part. This is the case when none of the component instances in the current declarative part meet all of the requirements of the given configuration specification.

10. Ashenden's review of P1076-2000/D1.

### 5.2.1 Binding indication

A binding indication associates instances of a component ~~declaration~~[11] with a particular design entity. It may also associate actuals with formals declared in the entity ~~interface~~ declaration[12].

binding_indication ::=
[ **use** entity_aspect ]
[ generic_map_aspect ]
[ port_map_aspect ]

The entity aspect of a binding indication, if present, identifies the design entity with which the instances of a component are associated. If present, the generic map aspect of a binding indication identifies the expressions to be associated with formal generics in the ~~design entity interface~~ entity declaration[13]. Similarly, the port map aspect of a binding indication identifies the signals or values to be associated with formal ports in the ~~design entity interface~~ entity declaration[14].

When a binding indication is used in ~~a~~ an explicit[15] configuration specification, it is an error if the entity aspect is absent.

A binding indication appearing in a component configuration ~~need not~~ must[16] have an entity aspect ~~under the following condition: The~~ unless the[17] block corresponding to the block configuration in which the given component configuration appears ~~is required to have~~ has[18] one or more configuration specifications that together configure all component instances denoted in the given component configuration. ~~Under this circumstance, these~~ The[19] binding indications appearing in these configuration specifications[20] are the corresponding[21] *primary binding indications*. ~~It is an error if a binding indication appearing in a component configuration does not have an entity aspect and there are no primary binding indications. It is also an error if, under these circumstances, the binding indication has neither a generic map aspect nor a port map aspect. This form of binding indication is the *incremental binding indication*, and it~~ A binding indication need not have an entity aspect; in that case, either or both of a generic map aspect or a port map aspect must be present in the binding indication. Such a binding indication is an *incremental binding indication*. An incremental binding indication[22] is used to *incrementally rebind* the ports and generics of the denoted instance(s) under the following conditions:

— For each formal generic appearing in the generic map aspect of the incremental binding indication and denoting a formal generic that is unassociated or associated with **open** in any of the primary binding indications, the given formal generic is bound to the actual with which it is associated in the generic map aspect of the incremental binding indication.

— For each formal generic appearing in the generic map aspect of the incremental binding indication and denoting a formal generic that is associated with an actual other than **open** in one of the primary binding indications, the given formal generic is *rebound* to the actual with which it is associated in the generic map aspect of the incremental binding indication. That is, the association given in the primary binding indication has no effect for the given instance.

---

11. IR1000.2.9.
12. Terminological correction.
13. Terminological correction.
14. Terminological correction.
15. LCS 8.
16. LCS 8.
17. LCS 8.
18. LCS 8.
19. LCS 8.
20. LCS 8.
21. LCS 8.
22. LCS 8.

— For each formal port appearing in the port map aspect of the incremental binding indication and denoting a formal port that is unassociated or associated with **open** in any of the primary binding indications, the given formal port is bound to the actual with which it is associated in the port map aspect of the incremental binding indication.

It is an error if a formal port appears in the port map aspect of the incremental binding indication and it is a formal port that is associated with an actual other than **open** in one of the primary binding indications.[23]

If the generic map aspect or port map aspect of a primary[24] binding indication is not present, then the default rules as described in 5.2.2 apply.

It is an error if an explicit entity aspect in an incremental binding indication does not adhere to any of the following rules:

— If the entity aspect in the corresponding primary binding indication is of the first form (fully bound), as specified in 5.2.1.1, then the entity aspect in the incremental binding indication must also be of the first form and must denote the same entity declaration as that of the primary binding indication. An architecture name must be specified in the incremental binding indication if and only if the primary binding indication also identifies an architecture name; in this case, the architecture name in the incremental binding indication must denote the same architecture name as that of the primary binding indication.

— If the entity aspect in the primary binding indication is of the second form (that is, identifying a configuration), then the entity aspect of the incremental binding indication must be of the same form and must denote the same configuration declaration as that of the primary binding indication.

NOTES

1—The third form (open) of an entity aspect does not apply to incremental binding indications as this form cannot include either a generic map aspect or a port map aspect and incremental binding indications must contain at least one of these aspects.

2—The entity aspect of an incremental binding indication in a component configuration is optional.

3—The presence of an incremental binding indication will never cause the default rules of 5.2.2 to be applied.[25]

*Examples:*

```
entity AND_GATE is
    generic(I1toO, I2toO: DELAY_LENGTH := 4 ns);
    port     (I1, I2: in BIT;O: out BIT);
end entity AND_GATE;

entity XOR_GATE is
    generic(I1toO, I2toO : DELAY_LENGTH := 4 ns);
    port     (I1, I2: in BIT;O : out BIT);
end entity XOR_GATE;

package MY_GATES is
    component AND_GATE is
        generic(I1toO, I2toO: DELAY_LENGTH := 4 ns);
        port     (I1, I2: in BIT;O: out BIT);
    end component AND_GATE;
```

23. LCS 8 (reformat only).
24. LCS 8.
25. LCS 8.

```
    component XOR_GATE is
        generic(I1toO, I2toO: DELAY_LENGTH := 4 ns);
        port      (I1, I2: in BIT;O : out BIT);
    end component XOR_GATE;
end package MY_GATES;

entity Half_Adder is
    port(X, Y: in BIT;
              Sum, Carry: out BIT);
end entity Half_Adder;

use WORK.MY_GATES.all;
architecture Structure of Half_Adder is
    for L1: XOR_GATE use
        entity WORK.XOR_GATE(Behavior)          -- The primary binding indication
            generic map (3 ns, 3 ns)            -- for instance L1.
            port map (I1 => I1, I2 => I2, O => O);
    for L2: AND_GATE use
        entity WORK.AND_GATE(Behavior)          -- The primary binding indication
            generic map (3 ns, 4 ns)            -- for instance L2.
            port map (I1, open, O);
begin
    L1: XOR_GATEport map (X, Y, Sum);
    L2: AND_GATEport map (X, Y, Carry);
end architecture Structure;

use WORK.GLOBAL_SIGNALS.all;
configuration Different of Half_Adder is
    for Structure
        for L1: XOR_GATE
            generic map (2.9 ns, 3.6 ns);   -- The  incremental binding
        end for;                            -- indication of L1; rebinds its generics.
        for L2: AND_GATE
            generic map (2.8 ns, 3.25 ns)   -- The incremental binding
            port map (I2 => Tied_High);     -- indication L2; rebinds its generics
        end for;                            -- and binds its open port.
    end for;
end configuration Different;
```

#### 5.2.1.1 Entity aspect

An entity aspect identifies a particular design entity to be associated with instances of a component. An entity aspect may also specify that such a binding is to be deferred.

```
entity_aspect ::=
      entity entity_name [ ( architecture_identifier) ]
    | configuration configuration_name
    | open
```

The first form of entity aspect identifies a particular entity declaration and (optionally) a corresponding architecture body. If no architecture identifier appears, then the immediately enclosing binding indication is said to *imply* the design entity whose interface is defined by the entity declaration denoted by the entity name and whose body is defined by the default binding rules for architecture identifiers (see 5.2.2). If an architecture identifier appears, then the immediately enclosing binding indication is said to *imply* the design entity consisting of the entity declaration denoted by the entity name together with an architecture body associated with the entity declaration; the architecture identifier defines a simple name that is used during the elaboration of a design hierarchy to select the appropriate architecture body. In either case, the corresponding component instances are said to be *fully bound*.

At the time of the analysis of an entity aspect of the first form, the library unit corresponding to the entity declaration denoted by the entity name is required to exist; moreover, the design unit containing the entity aspect depends on the denoted entity declaration. If the architecture identifier is also present, the library unit corresponding to the architecture identifier is required to exist only if the binding indication is part of a component configuration containing explicit block configurations or explicit component configurations; only in this case does the design unit containing the entity aspect also depend on the denoted architecture body. In any case, the library unit corresponding to the architecture identifier is required to exist at the time that the design entity implied by the enclosing binding indication is bound to the component instance denoted by the component configuration or configuration specification containing the binding indication; if the library unit corresponding to the architecture identifier was required to exist during analysis, it is an error if the architecture identifier does not denote the same library unit as that denoted during analysis. The library unit corresponding to the architecture identifier, if it exists, must be an architecture body associated with the entity declaration denoted by the entity name.

The second form of entity aspect identifies a design entity indirectly by identifying a configuration. In this case, the entity aspect is said to *imply* the design entity at the apex of the design hierarchy that is defined by the configuration denoted by the configuration name.

At the time of the analysis of an entity aspect of the second form, the library unit corresponding to the configuration name is required to exist. The design unit containing the entity aspect depends on the configuration denoted by the configuration name.

The third form of entity aspect is used to specify that the identification of the design entity is to be deferred. In this case, the immediately enclosing binding indication is said to *not imply* any design entity. Furthermore, the immediately enclosing binding indication must not include a generic map aspect or a port map aspect.

#### 5.2.1.2 Generic map and port map aspects

A generic map aspect associates values with the formal generics of a block. Similarly, a port map aspect associates signals or values with the formal ports of a block. The following applies to both external blocks defined by design entities and to internal blocks defined by block statements.

generic_map_aspect ::=
    **generic map** ( *generic*_association_list )

port_map_aspect ::=
    **port map** ( *port*_association_list )

Both named and positional association are allowed in a port or generic association list.

The following definitions are used in the remainder of this subclause:

— The term *actual* refers to an actual designator that appears either in an association element of a port association list or in an association element of a generic association list.

— The term *formal* refers to a formal designator that appears either in an association element of a port association list or in an association element of a generic association list.

The purpose of port and generic map aspects is as follows:

— Generic map aspects and port map aspects appearing immediately within a binding indication associate actuals with the formals of the ~~design entity interface~~ entity declaration[26] implied by the immediately enclosing binding indication. ~~No~~ It is an error if a[27] scalar formal ~~may be~~ is[28] associated with more

26. Terminological correction.
27. IR1000.4.7.
28. IR1000.4.7.

than one actual. ~~No~~ It is an error if a[29] scalar subelement of any composite formal ~~may be~~ is[30] associated more than once in the same association list.

Each scalar subelement of every local port of the component instances to which an enclosing configuration specification or component configuration applies must be associated as an actual with at least one formal or with a scalar subelement thereof. The actuals of these associations for a given local port ~~may be~~ must be either[31] the entire local port or any slice or subelement (or slice thereof) of the local port. The actuals in these associations must be locally static names.

— Generic map aspects and port map aspects appearing immediately within a component instantiation statement associate actuals with the formals of the component instantiated by the statement. ~~No~~ It is an error if a[32] scalar formal ~~may be~~ is[33] associated with more than one actual. ~~No~~ It is an error if a[34] scalar subelement of any composite formal ~~may be~~ is[35] associated with more than one scalar subelement of an actual.

— Generic map aspects and port map aspects appearing immediately within a block header associate actuals with the formals defined by the same block header. ~~No~~ It is an error if a[36] scalar formal ~~may be~~ is[37] associated with more than one actual. ~~No~~ It is an error if a[38] scalar subelement of any composite formal ~~may be~~ is[39] associated with more than one actual or with a scalar subelement thereof.

An actual associated with a formal generic in a generic map aspect must be an expression or the reserved word **open**; an actual associated with a formal port in a port map aspect must be a signal, an expression, or the reserved word **open**.

Certain restrictions apply to the actual associated with a formal port in a port map aspect; these restrictions are described in 1.1.1.2.

A formal that is not associated with an actual is said to be an *unassociated* formal.

NOTE

—A generic map aspect appearing immediately within a binding indication need not associate every formal generic with an actual. These formals may be left unbound so that, for example, a component configuration within a configuration declaration may subsequently bind them.

*Example:*

```
entity Buf is
    generic (Buf_Delay: TIME := 0 ns);
    port (Input_pin: in Bit; Output_pin: out Bit);
end Buf;
```

29. IR1000.4.7.
30. IR1000.4.7.
31. IR1000.4.7.
32. IR1000.4.7.
33. IR1000.4.7.
34. IR1000.4.7.
35. IR1000.4.7.
36. IR1000.4.7.
37. IR1000.4.7.
38. IR1000.4.7.
39. IR1000.4.7.

```
architecture DataFlow of Buf is
begin
    Output_pin <= Input_pin after Buf_Delay;
end DataFlow;

entity Test_Bench is
end Test_Bench;

architecture Structure of Test_Bench is
    component Buf is
        generic (Comp_Buf_Delay: TIME);
        port (Comp_I: in Bit; Comp_O: out Bit);
    end component;

    -- A binding indication; generic and port map aspects within a binding indication
    -- associate actuals (Comp_I, etc.) with formals of the ~~design entity interface~~ entity declaration[40]
    -- (Input_pin, etc.):
    for UUT: Buf
        use entity Work.Buf(DataFlow)
            generic map (Buf_Delay => Comp_Buf_Delay)
            port map (Input_pin => Comp_I, Output_pin=> Comp_O);
    signal S1,S2: Bit;
begin
    -- A component instantiation statement; generic and port map aspects within a
    -- component instantiation statement associate actuals (S1, etc.) with the
    -- formals of a component (Comp_I, etc.):
    UUT: Buf
        generic map(Comp_Buf_Delay => 50 ns)
        port map(Comp_I => S1, Comp_O => S2);

    -- A block statement; generic and port map aspects within the block header of a block
    -- statement associate actuals (4, etc.) with the formals defined in the block header:
    B: block
        generic (G: INTEGER);
        generic map(G => 4);
    begin
    end block;
end Structure;
```

NOTES

1[41]—A local generic (from a component declaration) or formal generic (from a block statement or from the entity declaration of the enclosing design entity) may appear as an actual in a generic map aspect. Similarly, a local port (from a component declaration) or formal port (from a block statement or from the entity declaration of the enclosing design entity) may appear as an actual in a port map aspect.

2—If a formal generic is rebound by an incremental binding indication, the actual expression associated by the formal generic in the primary binding indication is not evaluated during the elaboration of the description.

*Cross-References:* Generics, 1.1.1; Ports, 1.1.2; Interface Declarations, 4.3.2.[42]

40. Terminological correction.
41. LCS 15.
42. LCS 15.

### 5.2.2 Default binding indication

In certain circumstances, a default binding indication will apply in the absence of an explicit binding indication. The default binding indication consists of a default entity aspect, together with a default generic map aspect and a default port map aspect, as appropriate.

If no visible entity declaration has the same simple name as that of the instantiated component, then the default entity aspect is **open**. A *visible entity declaration* is either

a) An entity declaration that has the same simple name as that of the instantiated component and that is directly visible (see 10.3), ~~or~~[43]

b) An entity declaration that has the same simple name as that of the instantiated component and that would be directly visible in the absence of a directly visible (see 10.3) component declaration with the same simple name as that of the entity declaration, or

c) An entity declaration denoted by "L.C", where L is the target library and C is the simple name of the instantiated component. The *target library* is the library logical name of the library containing the design unit in which the component C is declared.[44]

These visibility checks are made at the point of the absent explicit binding indication that causes the default binding indication to apply.

Otherwise, the default entity aspect is of the form

**entity** *entity*_name ( *architecture*_identifier )

where the entity name is the simple name of the instantiated component, and the architecture identifier is the same as the simple name of the most recently analyzed architecture body associated with the entity declaration. If this rule is applied either to a binding indication contained within a configuration specification or to a component configuration that does not contain an explicit inner block configuration, then the architecture identifier is determined during elaboration of the design hierarchy containing the binding indication. Likewise, if a component instantiation statement contains an instantiated unit containing the reserved word **entity** but does not contain an explicitly specified architecture identifier, this rule is applied during the elaboration of the design hierarchy containing a component instantiation statement. In all other cases, this rule is applied during analysis of the binding indication.

It is an error if there is no architecture body associated with the entity ~~interface~~ declaration[45] denoted by an entity name that is the simple name of the instantiated component.

The default binding indication includes a default generic map aspect if the design entity implied by the entity aspect contains formal generics. The default generic map aspect associates each local generic in the corresponding component instantiation (if any) with a formal of the same simple name. It is an error if such a formal does not exist or if its mode and type are not appropriate for such an association. Any remaining unassociated formals are associated with the actual designator **open**.

The default binding indication includes a default port map aspect if the design entity implied by the entity aspect contains formal ports. The default port map aspect associates each local port in the corresponding component instantiation (if any) with a formal of the same simple name. It is an error if such a formal does not exist or if its mode and type are not appropriate for such an association. Any remaining unassociated formals are associated with the actual designator **open**.

If an explicit binding indication lacks a generic map aspect, and if the design entity implied by the entity aspect contains formal generics, then the default generic map aspect is assumed within that binding indication. Similarly,

43. LCS 5.
44. LCS 5.
45. Terminological correction.

if an explicit binding indication lacks a port map aspect, and the design entity implied by the entity aspect contains formal ports, then the default port map aspect is assumed within that binding indication.

## 5.3 Disconnection specification

A disconnection specification defines the time delay to be used in the implicit disconnection of drivers of a guarded signal within a guarded signal assignment.

disconnection_specification ::=
**disconnect** guarded_signal_specification **after** *time*_expression ;

guarded_signal_specification ::=
*guarded*_signal_list : type_mark

signal_list ::=
*signal*_name { , *signal*_name }
| **others**
| **all**

Each signal name in a signal list in a guarded signal specification must be a locally static name that denotes a guarded signal (see 4.3.1.2). Each guarded signal must be an explicitly declared signal or member of such a signal.

If the guarded signal is a declared signal or a slice thereof, the type mark must be the same as the type mark indicated in the guarded signal specification (see 4.3.1.2). If the guarded signal is an array element of an explicitly declared signal, the type mark must be the same as the element subtype indication in the (explicit or implicit) array type declaration that declares the base type of the explicitly declared signal. If the guarded signal is a record element of an explicitly declared signal, then the type mark must be the same as the type mark in the element subtype definition of the record type declaration that declares the type of the explicitly declared signal. Each signal must be declared in the declarative part enclosing the disconnection specification.

Subject to the aforementioned rules, a disconnection specification *applies to* the drivers of a guarded signal S of whose type mark denotes the type T under the following circumstances:

— For a scalar signal S, if an explicit or implicit disconnection specification of the form

**disconnect** S: T **after** *time*_expression;

exists, then this disconnection specification applies to the drivers of S.

— For a composite signal S, an explicit or implicit disconnection specification of the form

**disconnect** S: T **after** *time*_expression;

is equivalent to a series of implicit disconnection specifications, one for each scalar subelement of the signal S. Each disconnection specification in the series is created as follows: it has, as its single signal name in its signal list, a unique scalar subelement of S. Its type mark is the same as the type of the same scalar subelement of S. Its time expression is the same as that of the original disconnection specification.

The characteristics of the disconnection specification must be such that each implicit disconnection specification in the series is a legal disconnection specification.

— If the signal list in an explicit or implicit disconnection specification contains more than one signal name, the disconnection specification is equivalent to a series of disconnection specifications, one for each signal name in the signal list. Each disconnection specification in the series is created as follows: It has, as its single signal name in its signal list, a unique member of the signal list from the original

disconnection specification. Its type mark and time expression are the same as those in the original disconnection specification.

The characteristics of the disconnection specification must be such that each implicit disconnection specification in the series is a legal disconnection specification.

— An explicit disconnection specification of the form

**disconnect others**: T **after** *time*_expression;

is equivalent to an implicit disconnection specification where the reserved word **others** is replaced with a signal list comprised of the simple names of those guarded signals that are declared signals declared in the enclosing declarative part, whose type mark is the same as T, and that do not otherwise have an explicit disconnection specification applicable to its drivers; the remainder of the disconnection specification is otherwise unchanged. If there are no guarded signals in the enclosing declarative part whose type mark is the same as T and that do not otherwise have an explicit disconnection specification applicable to its drivers, then the above disconnection specification has no effect.

The characteristics of the explicit disconnection specification must be such that the implicit disconnection specification, if any, is a legal disconnection specification.

— An explicit disconnection specification of the form

**disconnect all**: T **after** *time*_expression;

is equivalent to an implicit disconnection specification where the reserved word **all** is replaced with a signal list comprised of the simple names of those guarded signals that are declared signals declared in the enclosing declarative part and whose type mark is the same as T; the remainder of the disconnection specification is otherwise unchanged. If there are no guarded signals in the enclosing declarative part whose type mark is the same as T, then the above disconnection specification has no effect.

The characteristics of the explicit disconnection specification must be such that the implicit disconnection specification, if any, is a legal disconnection specification.

A disconnection specification with the signal list **others** or **all** for a given type that appears in a declarative part must be the last such specification for the given type in that declarative part. ~~No~~ It is an error if a[46] guarded signal of the given type ~~may be~~ is[47] declared in a given declarative part following such a disconnection specification.

The time expression in a disconnection specification must be static and must evaluate to a non-negative value.

It is an error if more than one disconnection specification applies to drivers of the same signal.

If, by the aforementioned rules, no disconnection specification applies to the drivers of a guarded, scalar signal S whose type mark is T (including a scalar subelement of a composite signal), then the following default disconnection specification is implicitly assumed:

**disconnect** S : T **after** 0 ns;

A disconnection specification that applies to the drivers of a guarded signal S is the *applicable disconnection specification* for the signal S.

Thus the implicit disconnection delay for any guarded signal is always defined, either by an explicit disconnection specification or by an implicit one.

---

46. IR1000.4.7.
47. IR1000.4.7.

NOTES

1—A disconnection specification supplying either the reserved words **others** or **all** may apply to none of the guarded signals in the current declarative part, in the event that none of the guarded signals in the current declarative part meet all of the requirements of the disconnection specification.2—Since disconnection specifications are based on declarative parts, not on declarative regions, ports declared in an entity ~~interface~~ declaration[48] cannot be referenced by a disconnection specification in a corresponding architecture body.

*Cross-References:* Disconnection statements, 9.5; Guarded assignment, 9.5; Guarded blocks, 9.1; Guarded signals, 4.3.1.2; Guarded targets, 9.5; Signal guard, 9.1.

48. Terminological correction.