~~Section~~ Clause[1] 1

Design entities and configurations

The *design entity* is the primary hardware abstraction in VHDL. It represents a portion of a hardware design that has well-defined inputs and outputs and performs a well-defined function. A design entity may represent an entire system, a subsystem, a board, a chip, a macro-cell, a logic gate, or any level of abstraction in between. A *configuration* can be used to describe how design entities are put together to form a complete design.

A design entity may be described in terms of a hierarchy of *blocks*, each of which represents a portion of the whole design. The top-level block in such a hierarchy is the design entity itself; such a block is an *external* block that resides in a library and may be used as a component of other designs. Nested blocks in the hierarchy are *internal* blocks, defined by block statements (see 9.1).

A design entity may also be described in terms of interconnected components. Each component of a design entity may be bound to a lower-level design entity in order to define the structure or behavior of that component. Successive decomposition of a design entity into components, and binding those components to other design entities that may be decomposed in like manner, results in a hierarchy of design entities representing a complete design. Such a collection of design entities is called a *design hierarchy*. The bindings necessary to identify a design hierarchy can be specified in a configuration of the top-level entity in the hierarchy.

This ~~section~~ clause[2] describes the way in which design entities and configurations are defined. A design entity is defined by an *entity declaration* together with a corresponding *architecture body*. A configuration is defined by a *configuration declaration*.

1.1 Entity declarations

An entity declaration defines the interface between a given design entity and the environment in which it is used. It may also specify declarations and statements that are part of the design entity. A given entity declaration may be shared by many design entities, each of which has a different architecture. Thus, an entity declaration can potentially represent a class of design entities, each with the same interface.

```
entity_declaration ::=
    entity identifier is
        entity_header
        entity_declarative_part
  [ begin
        entity_statement_part ]
    end [ entity ] [ entity_simple_name ] ;
```

The entity header and entity declarative part consist of declarative items that pertain to each design entity whose interface is defined by the entity declaration. The entity statement part, if present, consists of concurrent statements that are present in each such design entity.

If a simple name appears at the end of an entity declaration, it must repeat the identifier of the entity declaration.

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

1.1.1 Entity header

The entity header declares objects used for communication between a design entity and its environment.

entity_header ::=
[*formal*_generic_clause]
[*formal*_port_clause]

generic_clause ::=
generic (generic_list) ;

port_clause ::=
port (port_list) ;

The generic list in the formal generic clause defines generic constants whose values ~~may be~~ are[3] determined by the environment. The port list in the formal port clause defines the input and output ports of the design entity.

In certain circumstances, the names of generic constants and ports declared in the entity header become visible outside of the design entity (see 10.2 and 10.3).

Examples:

--An entity declaration with port declarations only:

```
entity Full_Adder is
    port (X, Y, Cin: in Bit; Cout, Sum: out Bit) ;
end Full_Adder ;
```

--An entity declaration with generic declarations also:

```
entity AndGate is
    generic
        (N: Natural := 2);
    port
        (Inputs:  in      Bit_Vector (1 to N);
         Result:  out     Bit) ;
end entity AndGate ;
```

--An entity declaration with neither:

```
entity TestBench is
end TestBench ;
```

1.1.1.1 Generics

Generics provide a channel for static information to be communicated to a block from its environment. The following applies to both external blocks defined by design entities and to internal blocks defined by block statements.

generic_list ::= *generic*_interface_list

The generics of a block are defined by a generic interface list; interface lists are described in 4.3.2.1. Each interface element in such a generic interface list declares a formal generic.

3. IR1000.4.7.

The value of a generic constant may be specified by the corresponding actual in a generic association list. If no such actual is specified for a given formal generic (either because the formal generic is unassociated or because the actual is **open**), and if a default expression is specified for that generic, the value of this expression is the value of the generic. It is an error if no actual is specified for a given formal generic and no default expression is present in the corresponding interface element. It is an error if some of the subelements of a composite formal generic are connected and others are either unconnected or unassociated.

NOTE

—Generics may be used to control structural, dataflow, or behavioral characteristics of a block, or may simply be used as documentation. In particular, generics may be used to specify the size of ports; the number of subcomponents within a block; the timing characteristics of a block; or even the physical characteristics of a design such as temperature, capacitance, location, etc.

1.1.1.2 Ports

Ports provide channels for dynamic communication between a block and its environment. The following applies to both external blocks defined by design entities and to internal blocks defined by block statements, including those equivalent to component instantiation statements and generate statements (see 9.7).

port_list ::= *port*_interface_list

The ports of a block are defined by a port interface list; interface lists are described in 4.3.2.1. Each interface element in the port interface list declares a formal port.

To communicate with other blocks, the ports of a block can be associated with signals in the environment in which the block is used. Moreover, the ports of a block may be associated with an expression in order to provide these ports with constant driving values; such ports must be of mode **in**. A port is itself a signal (see 4.3.1.2); thus, a formal port of a block may be associated as an actual with a formal port of an inner block. The port, signal, or expression associated with a given formal port is called the *actual* corresponding to the formal port (see 4.3.2.2). The actual, if a port or signal, must be denoted by a static name (see 6.1). The actual, if an expression, must be a globally static expression (see 7.4).

After a given description is completely elaborated (see ~~Section~~ Clause[4] 12), if a formal port is associated with an actual that is itself a port, then the following restrictions apply depending upon the mode (see 4.3.2) of the formal port:

a) For a formal port of mode **in**,
the associated actual ~~may only~~ must[5] be a port of mode **in**, **inout**, or **buffer**.

b) For a formal port of mode **out**,
the associated actual ~~may only~~ must[6] be a port of mode **out** ~~or~~ ,[7] **inout**, or **buffer**[8].

c) For a formal port of mode **inout**,
the associated actual ~~may only~~ must[9] be a port of mode **inout** or **buffer**[10].

d) For a formal port of mode **buffer**,
the associated actual ~~may only~~ must[11] be a port of mode **out**, **inout**, or[12] **buffer**.

4. To conform to IEEE rules.
5. IR1000.4.7.
6. IR1000.4.7.
7. LCS 24.
8. LCS 24.
9. IR1000.4.7.
10. LCS 24.
11. IR1000.4.7.
12. LCS 24.

e) For a formal port of mode **linkage**,
the associated actual may be a port of any mode.

~~A **buffer** port may have at most one source (see 4.3.1.2 and 4.3.2). Furthermore, after a description is completely elaborated (see Section 12), any actual associated with a formal buffer port may have at most one source.~~[13]

If a formal port is associated with an actual port, signal, or expression, then the formal port is said to be *connected*. If a formal port is instead associated with the reserved word **open**, then the formal is said to be *unconnected*. ~~A~~ It is an error if a[14] port of mode **in** ~~may be~~ is[15] unconnected or unassociated (see 4.3.2.2) ~~only if~~ unless[16] its declaration includes a default expression (see 4.3.2). ~~A~~ It is an error if a[17] port of any mode other than **in** ~~may be~~ is[18] unconnected or unassociated ~~as long as~~ and[19] its type is ~~not~~[20] an unconstrained array type. It is an error if some of the subelements of a composite formal port are connected and others are either unconnected or unassociated.

NOTE

—Ports of mode **linkage** may be removed from a future version of the language. See Annex F.[21]

1.1.2 Entity declarative part

The entity declarative part of a given entity declaration declares items that are common to all design entities whose interfaces are defined by the given entity declaration.

```
entity_declarative_part ::=
    { entity_declarative_item }

entity_declarative_item ::=
        subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration
```

Names declared by declarative items in the entity declarative part of a given entity declaration are visible within the bodies of corresponding design entities, as well as within certain portions of a corresponding configuration declaration.

13. LCS 24.
14. IR1000.4.7.
15. IR1000.4.7.
16. IR1000.4.7.
17. IR1000.4.7.
18. IR1000.4.7.
19. IR1000.4.7.
20. IR1000.4.7.
21. LCS 25.

The various kinds of declaration are described in Clause 4, and the various kinds of specification are described in Clause 5. The use clause, which makes externally defined names visible within the block, is described in Clause 10.[22]

Example:

--An entity declaration with entity declarative items:

```
entity ROM is
    port ( Addr: in   Word;
           Data: out  Word;
           Sel:  in   Bit);
    type   Instruction is array (1 to 5) of Natural;
    type   Program is array (Natural range <>) of Instruction;
    use    Work.OpCodes.all, Work.RegisterNames.all;
    constant ROM_Code: Program :=
        (
            (STM,  R14,  R12,  12,  R13) ,
            (LD,    R7,   32,   0,  R1  ) ,
            (BAL,  R14,    0,   0,  R7  ) ,
                    • • •        -- etc.
        ) ;
end ROM;
```

NOTE

—The entity declarative part of a design entity whose corresponding architecture is decorated with the 'FOREIGN attribute is subject to special elaboration rules. See 12.3.

1.1.3 Entity statement part

The entity statement part contains concurrent statements that are common to each design entity with this interface.

```
entity_statement_part ::=
    { entity_statement }

entity_statement ::=
      concurrent_assertion_statement
    | passive_concurrent_procedure_call
    | passive_process_statement
```

~~Only~~ It is an error if any statements other than[23] concurrent assertion statements, concurrent procedure call statements, or process statements ~~may~~[24] appear in the entity statement part. All such statements must be passive (see 9.2). Such statements may be used to monitor the operating conditions or characteristics of a design entity.

22. Suggested in Lance Thompson's review of P1076-2000/D1.
23. IR1000.4.7.
24. IR1000.4.7.

Example:

--An entity declaration with statements:

```
entity Latch is
    port (  Din:    in    Word;
            Dout:   out   Word;
            Load:   in    Bit;
            Clk:    in    Bit );
    constant Setup: Time := 12 ns;
    constant PulseWidth: Time := 50 ns;
    use Work.TimingMonitors.all;
begin
    assert Clk='1' or Clk'Delayed'Stable (PulseWidth);
    CheckTiming (Setup, Din, Load, Clk);
end ;
```

NOTE

—The entity statement part of a design entity whose corresponding architecture is decorated with the 'FOREIGN attribute is subject to special elaboration rules. See 12.4.

1.2 Architecture bodies

An architecture body defines the body of a design entity. It specifies the relationships between the inputs and outputs of a design entity and may be expressed in terms of structure, dataflow, or behavior. Such specifications may be partial or complete.

```
architecture_body ::=
    architecture identifier of entity_name is
        architecture_declarative_part
    begin
        architecture_statement_part
    end [ architecture ] [ architecture_simple_name ] ;
```

The identifier defines the simple name of the architecture body; this simple name distinguishes architecture bodies associated with the same entity declaration. For the purpose of interpreting the scope and visibility of the identifier (see 10.2 and 10.3), the declaration of the identifier is considered to occur after the final declarative item of the entity declarative part of the corresponding entity declaration.[25]

The entity name identifies the name of the entity declaration that defines the interface of this design entity. For a given design entity, both the entity declaration and the associated architecture body must reside in the same library.

If a simple name appears at the end of an architecture body, it must repeat the identifier of the architecture body.

More than one architecture body may exist corresponding to a given entity declaration. Each declares a different body with the same interface; thus, each together with the entity declaration represents a different design entity with the same interface.

NOTE

—Two architecture bodies that are associated with different entity declarations may have the same simple name, even if both architecture bodies (and the corresponding entity declarations) reside in the same library.

25. LCS 3.

1.2.1 Architecture declarative part

The architecture declarative part contains declarations of items that are available for use within the block defined by the design entity.

```
architecture_declarative_part ::=
    { block_declarative_item }

block_declarative_item ::=
      subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | component_declaration
    | attribute_declaration
    | attribute_specification
    | configuration_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration
```

The various kinds of declaration are described in ~~Section~~ Clause[26] 4, and the various kinds of specification are described in ~~Section~~ Clause[27] 5. The use clause, which makes externally defined names visible within the block, is described in ~~Section~~ Clause[28] 10.

NOTE

—The declarative part of an architecture decorated with the 'FOREIGN attribute is subject to special elaboration rules. See 12.3.

1.2.2 Architecture statement part

The architecture statement part contains statements that describe the internal organization and/or operation of the block defined by the design entity.

```
architecture_statement_part ::=
    { concurrent_statement }
```

All of the statements in the architecture statement part are concurrent statements, which execute asynchronously with respect to one another. The various kinds of concurrent statements are described in ~~Section~~ Clause[29] 9.

26. To conform to IEEE rules.
27. To conform to IEEE rules.
28. To conform to IEEE rules.
29. To conform to IEEE rules.

Examples:

--A body of entity Full_Adder:

```
architecture DataFlow of Full_Adder is
    signal A,B: Bit;
begin
    A <= X xor Y;
    B <= A and Cin;
    Sum <= A xor Cin;
    Cout <= B or (X and Y);
end architecture DataFlow ;
```

--A body of entity TestBench:

```
library Test;
use Test.Components.all;
architecture Structure of TestBench is
    component Full_Adder
        port (X, Y, Cin: Bit; Cout, Sum: out Bit);
    end component;

    signal A,B,C,D,E,F,G: Bit;
    signal OK: Boolean;
begin
    UUT:        Full_Adder   port map (A,B,C,D,E);
    Generator:  AdderTest    port map (A,B,C,F,G);
    Comparator: AdderCheck port map (D,E,F,G,OK);
end Structure;
```

--A body of entity AndGate:

```
architecture Behavior of AndGate is
begin
    process (Inputs)
        variable Temp: Bit;
    begin
        Temp := '1';
        for i in Inputs'Range loop
            if Inputs(i) = '0' then
                Temp := '0';
                exit;
            end if;
        end loop;
        Result <= Temp after 10 ns;
    end process;
end Behavior;
```

NOTE

—The statement part of an architecture decorated with the 'FOREIGN attribute is subject to special elaboration rules. See 12.4.

1.3 Configuration declarations

The binding of component instances to design entities is performed by configuration specifications (see 5.2); such specifications appear in the declarative part of the block in which the corresponding component instances are created. In certain cases, however, it may be appropriate to leave unspecified the binding of component instances in

a given block and to defer such specification until later. A configuration declaration provides the mechanism for specifying such deferred bindings.

```
configuration_declaration ::=
    configuration identifier of entity_name is
        configuration_declarative_part
        block_configuration
    end [ configuration ] [ configuration_simple_name ] ;

configuration_declarative_part ::=
    { configuration_declarative_item }

configuration_declarative_item ::=
      use_clause
    | attribute_specification
    | group_declaration
```

The entity name identifies the name of the entity declaration that defines the design entity at the apex of the design hierarchy. For a configuration of a given design entity, both the configuration declaration and the corresponding entity declaration must reside in the same library.

If a simple name appears at the end of a configuration declaration, it must repeat the identifier of the configuration declaration.

NOTES

1—A configuration declaration achieves its effect entirely through elaboration (see ~~Section~~ Clause[30] 12). There are no behavioral semantics associated with a configuration declaration.

2—A given configuration may be used in the definition of another, more complex configuration.

Examples:

--An architecture of a microprocessor:

```
architecture Structure_View of Processor is
    component ALU port ( ••• );  end component;
    component MUX port ( ••• );  end component;
    component Latch port ( ••• );  end component;
begin
    A1: ALU port map ( ••• ) ;
    M1: MUX port map ( ••• ) ;
    M2: MUX port map ( ••• ) ;
    M3: MUX port map ( ••• ) ;
    L1: Latch port map ( ••• ) ;
    L2: Latch port map ( ••• ) ;
end Structure_View ;
```

--A configuration of the microprocessor:

```
library TTL, Work ;
configuration V4_27_87 of Processor is
    use Work.all ;
```

30. To conform to IEEE rules.

```
        for Structure_View
            for A1: ALU
                use configuration TTL.SN74LS181 ;
            end for ;
            for M1,M2,M3: MUX
                use entity Multiplex4 (Behavior) ;
            end for ;
            for all: Latch
                -- use defaults
            end for ;
        end for ;
    end configuration V4_27_87 ;
```

1.3.1 Block configuration

A block configuration defines the configuration of a block. Such a block ~~may be~~ is[31] either an internal block defined by a block statement or an external block defined by a design entity. If the block is an internal block, the defining block statement ~~may be~~ is[32] either an explicit block statement or an implicit block statement that is itself defined by a generate statement.

```
block_configuration ::=
    for block_specification
        { use_clause }
        { configuration_item }
    end for ;

block_specification ::=
      architecture_name
    | block_statement_label
    | generate_statement_label [ ( index_specification ) ]

index_specification ::=
      discrete_range
    | static_expression

configuration_item ::=
      block_configuration
    | component_configuration
```

The block specification identifies the internal or external block to which this block configuration applies.

If a block configuration appears immediately within a configuration declaration, then the block specification of that block configuration must be an architecture name, and that architecture name must denote a design entity body whose interface is defined by the entity declaration denoted by the entity name of the enclosing configuration declaration.

If a block configuration appears immediately within a component configuration, then the corresponding components must be fully bound (see 5.2.1.1), the block specification of that block configuration must be an architecture name, and that architecture name must denote the same architecture body as that to which the corresponding components are bound.

If a block configuration appears immediately within another block configuration, then the block specification of the contained block configuration must be a block statement or generate statement label, and the label must denote

31. IR1000.4.7.
32. IR1000.4.7.

a block statement or generate statement that is contained immediately within the block denoted by the block specification of the containing block configuration.

If the scope of a declaration (see 10.2) includes the end of the declarative part of a block corresponding to a given block configuration, then the scope of that declaration extends to each configuration item contained in that block configuration, with the exception of block configurations that configure external blocks. Similarly, if a declaration is visible (either directly or by selection) at the end of the declarative part of a block corresponding to a given block configuration, then the declaration is visible in each configuration item contained in that block configuration, with the exception of block configurations that configure external blocks. Additionally, if a given declaration is a homograph of a declaration that a use clause in the block configuration makes potentially directly visible, then the given declaration is not directly visible in the block configuration or any of its configuration items. See 10.3 for more information.

For any name that is the label of a block statement appearing immediately within a given block, a corresponding block configuration may appear as a configuration item immediately within a block configuration corresponding to the given block. For any collection of names that are labels of instances of the same component appearing immediately within a given block, a corresponding component configuration may appear as a configuration item immediately within a block configuration corresponding to the given block.

For any name that is the label of a generate statement immediately within a given block, one or more corresponding block configurations may appear as configuration items immediately within a block configuration corresponding to the given block. Such block configurations apply to implicit blocks generated by that generate statement. If such a block configuration contains an index specification that is a discrete range, then the block configuration applies to those implicit block statements that are generated for the specified range of values of the corresponding generate parameter; the discrete range has no significance other than to define the set of generate statement parameter values implied by the discrete range. If such a block configuration contains an index specification that is a static expression, then the block configuration applies only to the implicit block statement generated for the specified value of the corresponding generate parameter. If no index specification appears in such a block configuration, then it applies to exactly one of the following sets of blocks:

— All implicit blocks (if any) generated by the corresponding generate statement, if and only if the corresponding generate statement has a generation scheme including the reserved word **for**

— The implicit block generated by the corresponding generate statement, if and only if the corresponding generate statement has a generation scheme including the reserved word **if** and if the condition in the generate scheme evaluates to TRUE

— No implicit or explicit blocks, if and only if the corresponding generate statement has a generation scheme including the reserved word **if** and the condition in the generate scheme evaluates to FALSE

If the block specification of a block configuration contains a generate statement label, and if this label contains an index specification, then it is an error if the generate statement denoted by the label does not have a generation scheme including the reserved word **for**.

Within a given block configuration, whether implicit or explicit, an implicit block configuration is assumed to appear for any block statement that appears within the block corresponding to the given block configuration, if no explicit block configuration appears for that block statement. Similarly, an implicit component configuration is assumed to appear for each component instance that appears within the block corresponding to the given block configuration, if no explicit component configuration appears for that instance. Such implicit configuration items are assumed to appear following all explicit configuration items in the block configuration.

It is an error if, in a given block configuration, more than one configuration item is defined for the same block or component instance.

NOTES

1—As a result of the rules described in the preceding paragraphs and in ~~Section~~ Clause[33] 10, a simple name that is visible by selection at the end of the declarative part of a given block is also visible by selection within any configuration item con-

tained in a corresponding block configuration. If such a name is directly visible at the end of the given block declarative part, it will likewise be directly visible in the corresponding configuration items, unless a use clause for a different declaration with the same simple name appears in the corresponding configuration declaration, and the scope of that use clause encompasses all or part of those configuration items. If such a use clause appears, then the name will be directly visible within the corresponding configuration items except at those places that fall within the scope of the additional use clause (at which places neither name will be directly visible).

2—If an implicit configuration item is assumed to appear within a block configuration, that implicit configuration item will never contain explicit configuration items.

3—If the block specification in a block configuration specifies a generate statement label, and if this label contains an index specification that is a discrete range, then the direction specified or implied by the discrete range has no significance other than to define, together with the bounds of the range, the set of generate statement parameter values denoted by the range. Thus, the following two block configurations are equivalent:

for Adders(31 **downto** 0) ••• **end for**;

for Adders(0 **to** 31) ••• **end for**;

4—A block configuration ~~may~~ is allowed to[34] appear immediately within a configuration declaration only if the entity declaration denoted by the entity name of the enclosing configuration declaration has associated architectures. Furthermore, the block specification of the block configuration must denote one of these architectures.

Examples:

--A block configuration for a design entity:

```
for ShiftRegStruct--  An architecture name.
    -- Configuration items
    -- for blocks and components
    -- within ShiftRegStruct.
end for ;
```

--A block configuration for a block statement:

```
for B1              --  A block label.
    -- Configuration items
    -- for blocks and components
    -- within block B1.
end for ;
```

1.3.2 Component configuration

A component configuration defines the configuration of one or more component instances in a corresponding block.

```
component_configuration ::=
    for component_specification
        [ binding_indication ; ]
        [ block_configuration ]
    end for ;
```

The component specification (see 5.2) identifies the component instances to which this component configuration applies. A component configuration that appears immediately within a given block configuration applies to component instances that appear immediately within the corresponding block.

33. To conform to IEEE rules.
34. IR1000.4.7.

It is an error if two component configurations apply to the same component instance.

If the component configuration contains a binding indication (see 5.2.1), then the component configuration implies a configuration specification for the component instances to which it applies. This implicit configuration specification has the same component specification and binding indication as that of the component configuration.

If a given component instance is unbound in the corresponding block, then any explicit component configuration for that instance that does not contain an explicit binding indication will contain an implicit, default binding indication (see 5.2.2). Similarly, if a given component instance is unbound in the corresponding block, then any implicit component configuration for that instance will contain an implicit, default binding indication.

It is an error if a component configuration contains an explicit block configuration and the component configuration does not bind all identified component instances to the same design entity.

Within a given component configuration, whether implicit or explicit, an implicit block configuration is assumed for the design entity to which the corresponding component instance is bound, if no explicit block configuration appears and if the corresponding component instance is fully bound.

Examples:

--A component configuration with binding indication:

```
for all: IOPort
    use entity StdCells.PadTriState4 (DataFlow)
        port map (Pout=>A, Pin=>B, IO=>Dir, Vdd=>Pwr, Gnd=>Gnd) ;
end for ;
```

--A component configuration containing block configurations:

```
for D1: DSP
    for DSP_STRUCTURE
        -- Binding specified in design entity or else defaults.
        for Filterer
            -- Configuration items for filtering components.
        end for ;
        for Processor
            -- Configuration items for processing components.
        end for ;
    end for ;
end for ;
```

NOTE

—The requirement that all component instances corresponding to a block configuration be bound to the same design entity makes the following configuration illegal:

```
architecture A of E is
    component C is end component C;
    for L1: C use entity E1(X);
    for L2: C use entity E2(X);
begin
    L1: C;
    L2: C;
end architecture A;
```

```
configuration Illegal of Work.E is
    for A
        for all: C
            for X       --  Does not apply to the same design entity in all instances of C.
              •••
            end for; -- X
        end for; -- C
    end for; -- A
end configuration Illegal ;
```