SESSION 8

VHDL’92

Language Change Specifications

Declarative part in generate statements
LCS-0001(3)

Problem
- Generate statement constitutes a declarative region.
- Configuration specifications for component instantiations nested immediately inside generate statements is a problem.
- Component instantiation statements must be enclosed in an artificial block statement in order to have a declarative part in which to place the configuration specifications

Solution  upward compatible

generate_label :
  generation_scheme GENERATE
    [ { block_declarative_item } -- optional declarative part
    BEGIN ]
    { concurrent_statement }
  END GENERATE [generate_label ] ;
… Declarative part in generate statements

entity E is end E;
architecture A of E is
    component C end component;
begin
    X:for I in 1 to 10 generate
        for L1:C use entity ...;
        begin
            L1:C;
            L2:C;
        end generate;
    end A;

configuration C of Work.E is
    for A
        for X
            for L2:C use entity ...;
        end for;
    end for;
end C;

VHDL'92 8.3

… Declarative part in generate statements

block
begin
    l1:for i in 1 to 9 generate
        l2:comp port map(x(i), x(i+1)); -- regular case
        end generate;
    l3:comp port map(X(10), y); -- end case
end block;

drawback: there is a need for two different configuration specifications
  to bind the component instantiation statements

VHDL'92 8.4
Access to hierarchical pathname and simple name
LCS-0003(6)

Problem
- A VHDL model, when elaborated, is represented by a hierarchical tree of design instances with a single root.
- The location of a particular instance within the hierarchy is useful for handling instance specific information such as estimated timing.
- When reporting messages it is useful to refer to named items by the string representation of their names.

Solution
- Two new predefined attributes that are available on certain named items: 'Simple_Name and 'Path_Name. They yield a value of type String.
- The 'Simple_Name attribute is available on any declared item. The result is the simple name, character literal, or operator symbol of the declared item.
- The 'Path_Name attribute only be available for statically-elaborated declared items.

entity Bottom is generic (GBottom : Integer); port (PBottom : Integer);
end Bottom;

architecture BottomArch of Bottom is
  signal SBottom : Integer;
begin
  ProcessBottom : process
  variable V : Integer;
  begin
    if GBottom = 4 then
    elsif GBottom = -1 then
      assert V'Simple_Name = "V" and V'Path_Name = "/Top(Top).L2/Bottom(BottomArch).ProcessBottom.V";
      assert GBottom'Simple_Name = "GBottom" and GBottom'Path_Name = "/Top(Top).L2/Bottom.GBottom";
    end if;
    wait;
  end process;
end BottomArch;
... Access to hierarchical pathname and simple name

architecture Top of Top is
  component BComp
    generic (GComp : Integer)
    port (PComp : Integer);
  end component;
  signal S : integer;
begin
  B1 : block
    signal S : Integer;
  begin
    B2 : block
      signal S : Integer;
    begin
      G1 : for I in 1 to 10 generate
        B3 : block ...
        end generate;
      end block;
      end block;
    L2 : BComp generic map (-1) port map (S);
  end block;
end Top;

entity Top is end Top;

... Access to hierarchical pathname and simple name

B3: block
  signal S : Integer;
  for L1 : BComp use entity work.Bottom(BottomArch)
    generic map (GBottom => GComp)
    port map (PBottom => PComp);
  begin
    L1 : BComp generic map (I) port map (S);
    P1 : process
      variable V : Integer;
    begin
      if I = 7 then
      end if;
      wait;
    end process;
  end block;
configuration TopConf of Top is
for Top
for L2 : BComp use
    entity Work.Bottom(BottomArch)
    generic map (GBottom => GComp)
    port map (PBottom => PComp);
end for;
end for;
end TopConf;

Processes that run after the last "delta" at a time point
LCS-0004(3)

There is on occasion a need to check various conditions in a model after a "steady state" has been reached at a particular time point.

A postponed process is distinguished from other processes by the existence of the reserved word POSTPONED immediately preceding the reserved word PROCESS in its definition.

• No signal assignment statement in 0 fs inside a postponed process
• No wait for 0 fs inside a postponed process.
• If a postponed process P is currently sensitive to a signal S and an event occurs on S during the signal-update half of the simulation cycle, P does *not* wake up in the execute-process half of the cycle simulation cycle. Rather it resumes (along with all other such processes in any order) immediately prior to the next non-zero advancement of simulation time

• If the wait statement contains a condition clause, the condition is evaluated in the simulation cycle in which the event on S occurs, not immediately prior to the next non-zero advancement of simulation time. Thus, it is possible the condition clause will be FALSE when the postponed process actually does resume.
• If a postponed process wakes up due to the expiration of a timeout clause, it runs during the last simulation cycle that occurs at its wake-up time.
... Processes that run after the last "delta"

entity assert_delta is
end assert_delta;
architecture test of assert_delta is
  signal a,b,c,d,e,f : bit := 0;
begin
  c <= a after 10 ns;
  d <= b after 10 ns;
  e <= c and d;
  f <= a and b after 10 ns;
  postponed assert f=e severity warning; -- No "false alarm"
end test;

Private types
LCS-005(4)

problem

• It is impossible in VHDL 87 to confine knowledge of the implementation of an abstract type to the package that declares the type and provides operations (subprograms) for data of the type.
• The lack of this capability allows clients of the package to rely on the representation of the abstract data.
• The effect of this is that it is difficult to develop and use standard packages that allow implementation-dependent flexibility in the representation of abstract data.
... Private types

- A package declaration consists of a *visible part* and an optional *private part*
- A private type declaration is of the form
  
  \[
  \text{TYPE identifier IS PRIVATE \{} \text{ ALLOW contract_detail \} \} ;
  \text{contract_detail ::= ACCESS | RANGE <> | FILE | ...}?
  \]

- A contract detail of ACCESS means the provider has the option of completing the
  private type with an access type definition but the clients may not declare
  constants or signals of the private type.

Example

```vhdl
package WAVES_STANDARD is

... type EVENT_VALUE is private allow constant, port, parameter;

... function "+" (X, Y : EVENT_VALUE) return EVENT_VALUE;

... private
  -- Implementation defined
end WAVES_STANDARD;
```

Clean up for VHDL 87 user-defined attributes
LCS-0007(2)

- The language will permit a label on any statement. This will allow any statement to be attributed indirectly by attributing its label.
- Physical units and enumeration literals are attributable in VHDL 92. UNIT and LITERAL are added to the attribute-specification concept of an entity class.
...User-defined attributes

```vhdl
package P is
  attribute A : Integer;
  type MVL is (Low, High, ...);
  function F(X : Bit) return Boolean;
  function F(X : MVL) return Boolean;
  attribute A of Low : literal is 0;
  -- Low is not overloaded; no parameter result profile needed
  -- Note "unit" as an entity class
  attribute A of F(MVL return Boolean) : function is 1;
  -- F is overloaded; parameter result profile required
end P;
package body P is
  constant C : Integer := F(MVL return Boolean)'A;
  function F(X : Bit) return Boolean is
    attribute A of L : label is 3;
    begin
      ...
      L: X := Y;
      -- Label on sequential assignment statement
    end F;
  end P;
```

Improved aliasing and import/export
LCS-0008(2)

**Problem**
- There is often a need to build new interfaces out of existing packages.
- Awkward in VHDL because of limited facilities for "importing" and "exporting" declarations.
- The use clause in VHDL 87 provides a general way of importing declarations but the language does not provide a way to export a declaration imported with a use clause.
- Alternatively, an alias declaration can be used to both import an export a declaration but the alias declaration is rather limited in the kinds of declarations it deals with.

**Solution**
- Allow aliases for any declared item with the exception of design unit declared items.

This proposal is upward compatible.
… Aliasing and import/export

examples

package P1 is
  function Is_Zero(signal S : Bit) return Boolean;
  function Is_Zero(signal S : MVL) return Boolean;
  ...
end P1;

package P2 is
  alias Character is Std.Standard.Character;
  -- Enumeration literals for Std.Standard.Character
  -- automatically aliased in here
  ...
  alias Is_Zero is Work.P1.Is_Zero(MVL return Boolean);
  -- Picks up second Is_Zero in P1
  ...
end P2;

Allowing "no change" conditions in concurrent signal assignments
LCS-0009(5)

Problem
The ELSE clause is commonly used to save the existing value of the target signal in a conditional signal assignment:
S <= A when COND else S;
but this is done at the expense of already scheduled future activity. A transaction still occurs on the target signal.

Solution
• The reserved word NO_TRANSACTION should be allowed as a substitute for any waveform in a selected signal assignment or conditional signal assignment.
  • NO_TRANSACTION is not allowed in a sequential signal assignment

This is upward compatible.
... "no change" conditions

<table>
<thead>
<tr>
<th>Existing:</th>
<th>Proposition:</th>
</tr>
</thead>
</table>
| S <= NO_TRANSACTION when COND1  
  else VALUE2 after T2 when COND2  
  else VALUE3A after T3A, VALUE3B after T3B when COND3; | S <= NO_TRANSACTION when COND1  
  else VALUE2 after T2 when COND2  
  else VALUE3A after T3A, VALUE3B after T3B when COND3; |

-- Conditional signal assignment
-- The above is equivalent to

<table>
<thead>
<tr>
<th>conditional_waveforms ::=</th>
<th>waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ waveform when condition else }</td>
<td>{ when condition else waveform }</td>
</tr>
<tr>
<td>waveform</td>
<td>[ when condition ]</td>
</tr>
</tbody>
</table>

extend the operator set to include the xnor operator

LCS-0010(3)

A new reserved word (xnor) is added. All previous descriptions using this as an identifier will be in error.
Easing problems of composites differing only in subelement resolvedness
LCS-0011(2)

Problem
It is impossible for two identically-structured objects to be of the same type in VHDL if there exist matching subelements of the objects that differ in their associated resolution functions.

Solution
actual_part ::= actual_designator | function_name ( actual_designator ) | type_mark ( actual_designator ) -- add a third syntactic possibility

This solution is upward compatible

… Subelement resolvedness

subtype or_bit is wired_or bit;
subtype and_bit is wired_and bit;
type or_array is array(natural range <>) of or_bit;
type and_array is array(natural range <>) of and_bit;
-- or_array and and_array are different types

component c
  port(p : inout and_array);
end c;
signal x : or_array(1 to 10);
...
-- want to associate x with port p of c
l:c port map(or_array(y) => and_array(x));
Associating expressions with IN ports in association lists
LCS-0012(2)

Problem
• An "actual" that is associated with a formal port must be a "signal name".
• Unnecessary restriction for IN ports.
• Inconvenient in the case where the instantiator wishes to associate a particular "value" with an IN port on the instanced entity/component.

Solution
• VHDL allows an IN port to include a default expression (defines the value of the unconnected port).
• Associating an expression with an IN port in an association list should be equivalent to leaving the IN port unconnected and replacing the default expression of the IN port with one that yields the same value as the expression in the association list.
• For this suggested analogy to hold, the expression in the association list must be a static expression.

... Expressions with IN ports

component c
  port (p : in time);
end component;

l:c port map(17 ns);

This proposal is upward compatible.
Annotating related items with a single value (i.e., relations)  
LCS-0013(3)

Problem

The present language lacks facilities for building relationships between declared items and for attributing collections of declared items.

Example of use: the ability to document that two statements have identical resource-allocation requirements.

Solution

Introducing into the language the concepts of a "group" and a "group type". Intuitively, a "group" is a named list of declared items.

Every group belongs to a "group type" which defines the allowable kinds of items that can be in the group.

... Relations

```

 group_type_declaration ::=  
    GROUP TYPE identifier IS ( entity_class_entry [, entity_class_entry ] ) ;  
 entity_class_entry ::= entity_class | entity_class <>  

group_declaration ::=  
    GROUP identifier : group_type_name ( item_name_list ) ;  

Examples

 group type Resource is (label <>); -- arbitrary number of labels  
group type Path is (signal, signal); -- two signals

 group G1 : Resource(L1, L2, L5);  
group G2 : Resource(L3, L4);  
group D_to_Q : Path(D, Work.P.Q);  

 attribute Implementation of G1 : group is "cla";  
 attribute Rising_Pin_to_Pin_Delay of D_to_Q : group is 2 ns;  

 group type Dont_Merge is (group <>); -- "groups of groups"  
group Constraint_3 : Dont_Merge(G1, G2);  
```


Improvements in language-defined attributes
LCS-0014(4)

Problem
• Need for additional pre-defined mechanisms for performing commonly-needed operations and returning basic information.
• 'BEHAVIOR and 'STRUCTURE useless.

Solution
• 'BEHAVIOR and 'STRUCTURE removed.
• S'DRIVING: attribute of a signal S.
  – Available from within a process that has a driver for S or in a subprogram with an INOUT or OUT mode signal parameter S.
  – The result is TRUE if the signal is currently driven FALSE otherwise.
• S'DRIVING_VALUE
  The result is the current value of the driver for S within the process or passed to the subprogram

... Language-defined attributes

T'ASCENDING:
  T is a scalar type or subtype. TRUE if T is ascending and FALSE otherwise.

A'ASCENDING[(N)]
  TRUE if the associated index range is ascending and FALSE otherwise.

TIMAGE(Y)
  T is a scalar type and Y is an expression of this type.
  The result is the string representation of the value of the expression.
  Details should be consistent with the conversions defined for TextIO.

TVALUE(Y)
  T is a scalar type and Y is an expression of type STRING.
  The type of the result is T.
  This is the inverse of 'IMAGE.
Straightening out "static expressions" vs. elaboration-time evaluable expressions

LCS-0015(3)

This notion is intended to approximate the class of expressions that can be evaluated during model elaboration.

Problem

The present concept of a static expression is both too restrictive and too permissive.

• It does not allow as a static expression forms such as indexed names and function calls even though under certain conditions it is perfectly reasonable for such forms to be static expressions.

• It allows a reference to a subprogram-resident constant to be a static expression when this violates language notions about when elaboration of these items occurs.

• The requirements on expressions in a generation scheme of a generate statement are different from those on an index expression in a signal name that appears as an actual designator in a port association list.

Solution

• A static expression may not contain a reading reference to a signal or variable.

• May not contain an invocation of an "impure" function.

• May not contain a name that denotes an item declared in a subprogram declarative part.

Logging assert-like messages without asserting anything

LCS-0016(3)

case x is
    when 0 => ...
    when 1 => ...
    ...
    when others =>
        report "unexpected value for x!" severity error;
end case;
Extending the base character set of the language
LCS-0017(4)

VHDL 92 character set will be based on 8-bit ISO LATIN-1 (ISO 8859-1) rather than the seven-bit coded character set ISO 646.

Constraint-error problems
wrt concatenation and DOWNTO arrays
LCS-0018(2)

Problem

variable A, B : Bit_Vector(7 downto 0);
...
B := A(6 downto 3) & A(5 downto 2);
gives a run-time error because the lower bound of the result is -1 which violates the NATURAL index subtype of BIT_VECTOR.

Solution

• When the result of the operation is ascending, the left bound of the result is the same as the left bound of the left operand (unless the left operand is null in which case the result is the same as the right operand)

• When the result of the operation is descending, the right bound of the result is the same as the right bound of the right operand (unless the right operand is null in which case the result is the same as the left operand).
**New inertial keyword and specifying "pulse-rejection" time**  
**LCS-0019(4)**

signal_assignment_statement ::=  
  target <= [ TRANSPORT | [ USE time_expression ] INERTIAL] waveform ;

output_signal <= USE 0 ns INERTIAL input_signal AFTER 10 ns ;  
is the same as  
output_signal <= TRANSPORT input_signal AFTER 10 ns ;

output_signal <= USE 10 ns INERTIAL input_signal AFTER 10 ns ;  
is the same as  
output_signal <= INERTIAL input_signal AFTER 10 ns ;

output_signal <= USE 5 ns INERTIAL input_signal AFTER 10 ns ;

Upward-compatibility  
The new keyword INERTIAL is being added.

---

**Universal arithmetic**  
**LCS-0020(2)**

The VHDL LRM requires "exact" evaluation of static universal expressions.

Solution  
- Drop the requirement for exact evaluation of static universal expressions.
- The precision/accuracy requirements of static (or locally static) universal arithmetic should be no more demanding than those for run-time universal arithmetic.
Provide access to predefined operators for any given type
LCS-0022(2)

The user must be able to overload a predefined operator but still have the possibility accessing the predefined operator.

```vhdl
function "+"(a,b : in integer) return integer is
begin
  if float(a)+float(b) > float(integer'right) then
    return integer'right;
  else
    return std.standard."+"(a,b);
  end if;
end;
```

```vhdl
package log is
  type bit4 is ('x', '0', '1', 'z');
  alias same is "="(bit4, bit4 return boolean);
  function "="(a,b : in bit4) return boolean;
end log;
```

```vhdl
package body log is
  function "="(a,b : in bit4) return boolean is
  begin
    if same(a, 'x') or same(b, 'x') then
      return false;
    else
      return same(a, b);
    end if;
  end;
end;
```

Overloading of resolution functions
LCS-0023(5)

Problem
Overloading resolution functions can be an interesting modeling feature of VHDL 92. Adding this possibility leads to some situations where the compiler will be unable to disambiguate the name of the resolution function. The purpose of this LCS is to define such cases.

Solution
A resolution function given in a subtype indication is ambiguous (and therefore illegal) if and only if

- the given name of the function, the required return type of the function (i.e., the base type of the type mark that follows the function name), and
- the fact that the function must have a single parameter of a composite type are not sufficient to uniquely identify the function among all visible declarations.
... Overloading of resolution functions

EXAMPLE 1:
function F1(X : BIT_VECTOR) return BIT;
function F1(X : STRING) return CHARACTER;
signal S : F1 BIT; -- Unambiguous and legal

EXAMPLE 2:
function F1(X : BIT_VECTOR) return BIT;
function F1(X : STRING) return BIT;
signal S : F1 BIT; -- Ambiguous (and therefore illegal)

?? EXAMPLE 3:
function F1(X : BIT VECTOR; Y : INTEGER) return BIT;
function F1(X : STRING) return BIT;
signal S : F1 BIT; -- Unambiguous but illegal

EXAMPLE 4:
function F1(X : BIT_VECTOR) return BIT;
function F1(X : STRING; Y : INTEGER) return BIT;
signal S : F1 BIT; -- Unambiguous and legal

Shift and rotate operators

LCS-0024(5)

The language shall include a collection of operators for performing shift/rotate operations on single dimensional arrays whose element types are BIT or BOOLEAN:

Operation: Fill Value:
Shift left logical A SLL N '0' for BIT and FALSE for boolean
Shift right logical A SRL N "
Shift left arithmetic A SLA N A(A'left)
Shift right arithmetic A SRA N A(A'Right)
Rotate left A ROL N
Rotate right A ROR N
Direct instantiation
LCS-0025(2)

Problem
The amount of syntax required to instantiate and bind a simple component to an existing design entity.

Solution
Extend the capabilities of the component instantiation statement

```
component_instantiation_statement ::= 
  instantiation_label : instantiated_unit [generic_map_aspect] [port_map_aspect];

instantiated_unit ::= 
  [ COMPONENT ] component_name 
  | ENTITY entity_name [(architecture_identifier)] 
  | CONFIGURATION configuration_name
```

---

... Direct instantiation

```
entity AND_GATE is 
  generic (tLH, tHL : time := 5 ns);
  port (I1, I2 : in Bit; O : out Bit);
end AND_GATE;

architecture BEHAVIOR of AND_GATE is 
begin 
  ...
end BEHAVIOR;

entity TEST_AND_GATE is end TEST_AND_GATE;

architecture TEST of TEST_AND_GATE is 
  signal S1, S2, S3 : Bit;
begin 
  L1 : entity WORK.AND_GATE(BEHAVIOR) 
  generic map (7 ns, 8 ns) 
  port map ($S1, S2, S3$);
end TEST;
```
Foreign interface
LCS-0026(3)

Problem
Designers often make use of general-purpose programming languages or other simulation models besides VHDL. General-purpose languages often have functionality or runtime capabilities which would be useful to access from within VHDL models. In order to facilitate access to these capabilities a formalized, standard way of identifying and accessing models written in other languages should be provided in VHDL.

Solution
Include a predefined STRING attribute called FOREIGN which can be placed on subprograms or architecture bodies. Whenever this attribute is placed on a subprogram or architecture it identifies that subprogram or architecture as foreign which means it will have a non-VHDL implementation. The string value can be used to supply implementation-specific information.

The existence of this attribute means that any VHDL statements or declarations associated with the subprogram or architecture are ignored.

Foreign functions may possess "local state"
This capability need not be provided by all implementations.

---

… Foreign interface

-- Example 1: Foreign Subprograms
package TRIG_FUNCTIONS is
  ....
  function COSINE (I : REAL) return REAL;
  attribute FOREIGN of COSINE (REAL return REAL) : function
    is "LANGUAGE -> C; MAPPING -> cos;"
  ....
end TRIG_FUNCTIONS;

-- Example 2: Foreign Models
architecture ON_MODELER of CHIP_386i is
  attribute FOREIGN of ON_MODELER : architecture is "/foo/file.lmsi";
begin
end ON_MODELER;
Clean-up for rules on OTHERS within aggregates  
LCS-0027(1)

The choice OTHERS will only be allowed within an element association if the element association appears last within the aggregate and consists of only this one choice.

Actual parts in binding-indication map aspects, slices in formal parts  
LCS-0028(3)

entity E1 is  port (in0, in1, in2: out bit); end E1;

entity E is  end E;

architecture ILSA of E is  
component c port (i0: out bit_vector (0 to 2));  end component;
signal s : bit;
for u1:c use entity work.E1(a)
  port map (in0 => i0(0), in1 => i0(1), in2 => i0(2));  -- this notation is now legal
signal x1: bit_vector (0 to 2);
signal x2: bit_vector (4 to 5);
signal x3: bit;
begin
  u1:c port map (x1);
  u2:c port map (i0(0 to 1) => x2, i0(2) => x3);
  -- This notation is also legal, it means the same as:
  u3:c port map (i0(0) => x2(4), i0(1) => x2(5), i0(2) => x3);
end ILSA;
Barring non-language-defined units from residing in STD
LCS-0029(2)

A conforming implementation of VHDL may not provide non-language-defined units in library STD nor may it allow non-language-defined units to be placed in STD. The reference manual should make this clear.

New TIME subtype in STANDARD for delays, etc.
LCS-0030(3)

The following declaration should be added to Std.Standard:

subtype Delay_LENGTH is Time range 0 fs to Time'High;
Presentation improvements in the language reference manual
LCS-0031(2)

- The LRM should include an index.
- It should include an appendix identifying those constructs which are potentially non-portable.
- A strong effort should be made to include only legal VHDL in the examples in the LRM.
- It is absolutely essential that the LRM contain paragraph numbers.

Disambiguating names
LCS-0032(2)

Problem
The VHDL LRM does not explain the meaning of selected names when multiple interpretations exist.

Solution
(1) If the prefix in a selected name has an interpretation as the name of an enclosing subprogram then the selected name can only be an expanded name.
   If there exist multiple interpretations of the prefix as an enclosing subprogram then the selected name is ambiguous.
   If a selected name cannot be interpreted as an expanded name based on its prefix as described above, then the suffix and the innermost complete context are used as necessary to disambiguate the name.

(2) If a name or primary has an interpretation as both an indexed name as well as a function call then the innermost complete context is used to disambiguate the name.
... Disambiguating names

Example 1 -- The following is legal:

```vhdl
type A is array (1 to 10) of Integer;
function AF (P : integer := 3) return A;
variable I : Integer;
variable J : A;
...
I := AF (3); -- indexed name
J := AF (3); -- function call
```

Example 2 -- The following is also legal:

```vhdl
type R1 is record
  I: Integer;
end record;
type R2 is record
  I: Real;
end record;
function F return R1;
function F return R2;
...
constant C: Integer := F.I; -- call on first F
```

Example 3 -- The following is illegal regardless of the context of the "f.x":

```vhdl
function f return bit is
  x : integer;
function f return boolean is
  function f return integer is
    x : real;
    begin
      ... f.x ...
      end f;
    ...
end f;
...
end f;
```

Example 4 -- The following is legal with the expanded name interpretation being the correct one:

```vhdl
type r is record f : integer; end record;
function x return r is
  variable f : integer;
  variable v : integer;
  begin
    v := x.f;
    ...
  end x;
```

VHDL'92 8.49

VHDL'92 8.50
Allowing design root to have generics/ports
LCS-0033(3)

Problem
The LRM does not say anything about whether or not the root design entity may have ports or generics.

Solution
• An implementation may allow the root of a design to have ports and/or generics. The means by which these ports and/or generics are associated with the external environment of the VHDL model are outside the scope of the language.
• An implementation is not required to support ports or generics on the root of the VHDL model.
• If supported, an implementation may impose restrictions on the types and modes of these ports and/or generics.

Locally-static, locally/globally static names, case statements
LCS-0034(2)

(1) A literal of type STANDARD.TIME should not be allowed as a locally static primary.

(2) The requirement that a constant be "of a locally static subtype" in order for a reference to the constant to be a locally static primary should be dropped. Similarly, the requirement that the type mark in a qualified expression denote a locally static subtype for the qualified expression to be a locally static primary should also be dropped.

(3) A range that is in the form of an attribute name (e.g., a 'RANGE attribute reference) should be a locally static range if and only if the prefix is a locally static subtype.

(4) It should be clarified that Items 4 & 5 in the first LRM 7.4 list are talking about attribute names whose prefix is a locally static subtype.

(5) A primary that is a reference to an alias is a locally static primary if and only if the aliased name given in the corresponding alias declaration is a locally static primary.
… Locally-static, locally/globally static names, case statements

(6) A name is a locally static name unless one of the following apply:

(a) It is a simple name, denotes an alias, and the aliased object in the alias declaration is not given with a locally static name.
(b) It is an indexed name and one or more of the indexing expressions is not a locally static expression.
(c) It is a slice name and the discrete range is not a locally static discrete range.
(d) It is an indexed name or a slice name and the prefix is an alias whose subtype is not locally static.
(e) It has a prefix and the prefix is of an access type.
(f) It has a prefix and the prefix is not a static name.

(7) For the sake of consistency similar changes should probably be made to the definition of a globally static name although (a) is not an issue because the aliased object is required by the language to be given with a static name, (d) may not be a problem depending on how globally static subtypes turn out in the revision of LCS-0015 that is now underway, and (e) is not problem (this is only of relevance for signals).

… Locally-static, locally/globally static names, case statements

(8) The type of the select expression in a case statement must either be a discrete type or a one-dimensional array type whose element base type is a character type.

(9) For a one-dimensional array select expression, the expression must be one of:

(a) A slice whose discrete range is locally static.
(b) A declared object or declared object subelement of a locally static subtype.
(c) A function call where the return type mark of the function denotes a locally static subtype.
(d) A qualified expression or type conversion where the type mark denotes a locally static subtype.

(10) For a scalar select expression, a function call where the return type mark of the function denotes a locally static subtype is added to the list of possibilities where base-type coverage is not required (and the range of the locally static subtype is used instead).
Reference to formal from within its own interface list  
LCS-0035(1)

Problem
The LRM is unclear about the meaning when referencing a formal interface object from within its own interface list.

Solution
The use of a name that denotes an interface object is not allowed in the default expressions or constraints of an interface list if the declaration of the interface object is itself within the interface list.

Examples
ENTITY E IS  
GENERIC (G1 : INTEGER ; G2 : INTEGER := G1 ); -- ILLEGAL  
PORT (P1 : STRING ; P2 : STRING(P1'RANGE) ); -- ILLEGAL  
PROCEDURE X(Y1, Y2: INTEGER; Y3 : INTEGER RANGE Y1 TO Y2); -- ILLEGAL  
END E;

Association-list type-conversions matching dataflow  
LCS-0036(2)

Problem
- VHDL presently allows both the formal part and the actual part in an association element to be in the form of a function call (i.e., specifying a type conversion).
- This is not WYSIWYG since these type conversions may not be applicable depending on the direction of dataflow.

Solution
- The formal part of a (named) association element may only be in the form of a function call when the corresponding interface object is of mode OUT, INOUT, BUFFER, or LINKAGE.
- The actual part of a (named or positional) association element may only in in the form of a function call when the corresponding interface object is of the mode IN, INOUT, or LINKAGE.
… Association-list type-conversions matching dataflow

package p is
    procedure p1(variable x : in  integer);
    procedure p2(variable x : out integer);
end p;
package body p is
    procedure example is
        variable v : integer := 0;
        function f(y : integer) return integer is
            begin
            ...
        end f;
        begin
            p1(x => f(v)); -- OK
            p2(f(x) => v); -- OK
            p1(f(x) => v); -- ILLEGAL
            p2(x => f(v)); -- ILLEGAL
        end example;
    ...
end p;

Clean-up for guarded signals
LCS-0037(3)

Problem
Assignments of null waveform elements to signal parameters.

Solution
• The reserved word BUS should not be allowed in an interface signal declaration for a subprogram parameter.
• A signal parameter is a guarded signal if and only if the corresponding actual signal is a guarded signal.
• It continues to be an error to assign a null waveform element to a guarded signal (parameter ???). For null-waveform assignments to signal parameters, the check for this condition must be performed, in general, at run time.
Clarifying alias declarations for objects, attributes on subelements
LCS-0038(2)

Problem
• The meaning of the constraint on a scalar alias is unclear.
• It is unclear whether aliases can be established for items such as S'STABLE(2
NS) or pre-defined GUARD signals.
• We also consider the related issues of placing attributes on non-declared objects
(e.g., subelements of objects) and of the requirements on the prefix in a user-
defined attribute reference.

Solution
• Aliases of scalars: the bounds and direction implied by the subtype indication
from the alias declaration must match that of the object being aliased. This check
is performed, in general, at the time the alias declaration is elaborated.
• Aliases of 1-dimensional arrays: there must exist matching elements between the
constraint implied by the subtype indication from the alias declaration and that of
the object being aliased.

… Alias declarations for objects, attributes on subelements

• Implicitly declared signal GUARD can be aliased.
• A signal-valued attribute can be aliased.
• Continue to disallow placing a user-defined attribute value on a subelement, slice,
object designated by an access value, signal-valued attribute, etc.
Generalizing VHDL identifiers  
LCS-0039(3)

**Problem**
In order to allow consistency with other IEEE standards the language should generalize what is allowed for an identifier to include, e.g., VHDL reserved words, leading digits, etc. Examples of required identifiers include 68000, bus, out, in, select, etc.

**Solution**
- An extended identifier is delimited by backslashes and may include any graphic character (including the space character) but may not include format effectors (tabs, line feeds, etc).
- In order to represent a backslash within an extended identifier, a pair of adjacent backslashes must be used.
- Unlike normal identifiers, extended identifiers are case sensitive, e.g., \Joe\ is not the same identifier as \JOE\.
- An extended identifier is never the same as a normal (i.e., non-extended) identifier and vice-versa, e.g., \X\ is not the same as X or x.

**Examples**:
- \a\ b\ baker_charlie \A B__ $ & @ ! ( " /\ \\ \a\b\ \BUS\ \23\ 

Run-time constraint/subtype checks  
LCS-0041(2)

The VHDL LRM is not clear on the set of run-time constraint/subtype checks that a conforming implementation must perform. In the interest of portability, the reference manual should precisely specify each of these checks. The most incompletely-specified area concerning constraint/subtype checks in the present language are those for subprogram parameters.
Syntactic improvements, uniform bracketing
LCS-0042(2)

Users of VHDL are frustrated by inconsistencies in the syntax which require different syntactic constructs in similar circumstances.

architecture identifier of entity_name is
architecture_declarative_part
begin
architecture_statement_part
end [ architecture ] [ architecture_simple_name ];

block_label :
block [ ( guard_expression ) ] [ is ]
block_header
block_declarative_part
begin
block_statement_part
end block [ block_label ];

[ case_label : ]
case expression is
[ case_statement_alternative ]
decase [ case_label ];

component identifier [ is ]
[ local_generic_clause ]
[ local_port_clause ]
end component [ component_simple_name ];

configuration identifier of entity_name is
configuration_declarative_part
block_configuration
end [ configuration ] [ configuration_simple_name ];

[ if_label : ]
if condition then
sequence_of_statements
[ elsif condition then
sequence_of_statements ]
[ else
sequence_of_statements ]
end if [ if_label ];

package body package_simple_name is
package_body_declarative_part
end [ package body ] [ package_simple_name ];

package identifier is
package_declarative_part
end [ package ] [ package_simple_name ];
... Syntactic improvements, uniform bracketing

[ process_label :
  process [ ( sensitivity_list ) ] [ is ]
  process_declarative_part
  begin
  process_statement_part
  end process [ process_label ] :

record
  element_declaration
  { element_declaration }
end record [ record_type_simple_name ] :

subprogram_specification is
  subprogram_declarative_part
  begin
  subprogram_statement_part
  end [ subprogram_kind ] [ designator ] :
subprogram_kind ::= procedure
  function

I/O
LCS-0043(3)

• A file object is a fourth class of object in VHDL.
• A subprogram parameter may also be a file object; such a file object is declared with an "interface file declaration". Syntactically, an interface file declaration requires the reserved word FILE where SIGNAL, CONSTANT, and VARIABLE are presently allowed.
• The corresponding actual in a subprogram call must itself be a file object.
• The mode of the parameter F to the pre-defined Read procedure is changed from IN to INOUT and the mode of the parameter F to the pre-defined Write procedure is changed from OUT to INOUT.
• Functions are allowed to take file objects as parameters. The parameter F to the pre-defined EndFile function is an IN-mode file-class parameter.
• functions cannot reference file objects.
... I/O

Added to package STANDARD are the following new enumeration types:

```vhdl
type File_Open_Kind is
  (Read_Open, -- Resulting access mode is read-only
   Write_Open, -- Resulting access mode is write-only
   Append_Open); -- Resulting access mode is write-only

type File_Open_Status is
  (Open_OK, -- Open was successful
   Status_Error, -- File object was already open
   Name_Error, -- Could not find external file
   Use_Error); -- Could not use given open kind with
                 -- this external file
```

The syntax of the VHDL 87 file declaration is changed as follows:

```vhdl
file_declaration ::= FILE identifier : subtype_indication [ file_open_information ] ;

file_open_information ::= [ OPEN *file_open_kind*_expression ] IS *string*_expression
```

---

... I/O

There exist two new pre-defined procedures associated with every file type.

```vhdl
procedure File_Open(F : inout FT;
  External_Name : in String;
  Open.Kind : in File_Open.Kind := Read_Mode);

procedure File_Open(Status : out File_Open_Status;
  F : inout FT;
  External_Name : in String;
  Open.Kind : in File_Open.Kind := Read_Mode);

procedure File_Close(F : inout FT);
```
EXAMPLE 1:

type Int_IO is file of Integer;
file My_File : Int_IO open Append_Open is "~jones/collect_data";

EXAMPLE 2:

procedure ... is
  type Int_IO is file of Integer;
  file F : Int_IO;
  variable Status : File_Open_Status;
begin
  File_Open(F, "~jones/collect_data", Status);
  if Status /= Open_OK then
    ...
  end if;
end procedure;

Impure functions

package Number_Stream is
  impure function Next_Biggest return Integer;
end Number_Stream;

package body Number_Stream is
  type Int_IO is file of Integer;
  file Stream_A : Int_IO is "...";
  file Stream_B : Int_IO is "...";

  impure function Next_Biggest return Integer is
    variable A, B : Integer;
    begin
      read(Stream_A, A);
      read(Stream_B, B);
      if A > B then
        return A;
      else
        return B;
      end if;
    end Next_Biggest;
end Number_Stream;
Pre-defined impure functions  
LCS-0044(2)

Problem
The LRM does not define the semantics of Std.Standard.Now when this function is
invoked during elaboration. Furthermore, a need has been identified for a pre-defined language function that generates pseudo-random numbers.

Solution
Std.Standard.Now should return 0 fs if called during elaboration.
The subprogram specification for Now in Std.Standard should be changed to
declare this function as an “impure” function.
Add Random_Numbers package to the library Std.

package Random_Numbers is
    procedure Init_Random(Seed : Integer);
    -- Establishes the seed value that is used to guide the sequence of
    -- pseudo-random numbers returned by successive calls to Random. If
    -- Init_Random is not called, the default initial seed value is 1.
    -- Init_Random may be called multiple times to re-establish the seed
    -- value and (re-initialize the sequence returned by Random).

    impure function Random return Integer;
    -- Returns a pseudo-random Integer and generates a new seed value
    -- based on the current seed value. The language does not define
    -- the algorithm used to generate the random numbers nor does it
    -- define the "next" random number that results from any particular
    -- seed value.
end Random_Numbers;
Default binding and binding time
LCS-0045(4)

Problem

• The LRM seems to require the use of a configuration declaration in order to
complete default binding. This is an unfortunate restriction that serves to
inconvenience users.
• There is a technical problem with the definition of default binding that (in theory)
prevents the desired semantics for the default entity aspect from being the obtained
effect during the process of default binding.
• VHDL 87 does not specify the time at which (explicit or implicit) binding
information is processed by an implementation, and, in particular, the language does
not specify the point at which entities, architectures, and configurations are "bound to" a configuration declaration or instancing architecture.

---

entity Bottom1 is end Bottom1;
architecture Bottom1Body of Bottom1 is
begin ... end Bottom1Body;
entity Bottom2 is end Bottom2;
architecture Bottom2Body of Bottom2 is
begin ... end Bottom2Body;
entity Middle1 is end Middle1;
architecture Middle1Body of Middle1 is
begin ... end Middle1Body;
entity Middle2 is end Middle2;
architecture Middle2Body of Middle2 is
begin ...
component Bottom1 end component;
component Bottom2 end component;
for L4 : Bottom2 use entity
WORK.Bottom2(Bottom2Body);
begin
L3 : Bottom1;
L4 : Bottom2;
end Middle2Body;
configuration Middle2Conf of Middle2 is
for Middle2Body
for L3 : Bottom1 use entity
WORK.Bottom1(Bottom1Body); end for;
end for;
end Middle2Conf;

entity Top is end Top;
architecture TopBody of Top is
component Middle1 end component;
component Middle2 end component;
for L2 : Middle2 use configuration
WORK.Middle2Conf;
begin
L1 : Middle1;
L2 : Middle2;
end TopBody;
... Default binding and binding time

This model is simulatable despite:

- The absence of a configuration specification for component instance L1 in Top(TopBody).
- The lack of a configuration declaration for Top(TopBody).
- Note that Middle2(Middle2Body) depends on entity Bottom2 but not on Bottom2(Bottom2Body) or entity Bottom1.
- Note that the only architecture that configuration Middle2Conf depends on is Middle2(Middle2Body).
- Had the component configuration for L3 : Bottom1 in this configuration declaration contained an inner block configuration, the configuration declaration would have depended on Bottom1(Bottom1Body).

Global Variables
LCS-0046(3)

Problem
- Currently, signals are the only means of communication between processes.
- Their utility is sometimes limited by the inherent or explicit delay introduced between the assignment of a new value and the ability to "see" that new value from other processes.

Solution
Shared variables may be declared in any declarative region except processes and subprograms.

```
SHARED VARIABLE identifier_list : subtype_indication [:= expression];
```
Shared variables are updated using variable assignment statements.
Shared variables may be referenced from any process or procedure unless that procedure is nested (at any level) inside a function which is not identified as an "impure" function.
Every reading or writing reference to a shared variable must be performed inside a protected region.
--- Global Variables

protected_region ::= 
  protect ( variable_name {, variable_name} )
  begin
  sequence_of_statements
  end protect;

During the execution of the protected region, no other read or write operations may 
be performed on the variables listed in the protected region; i.e., no other protected 
region for these variables can be entered until the current protected region is 
completed.

No nesting of protected regions.

No wait statement.

No protected region is executed during the evaluation of implicit signals or during 
network evaluation (i.e., propagation of signal values through ports).

Concurrent statements which have equivalent processes and which reference one or 
more shared variables have an implicit protected placed in the equivalent process 
from the beginning of the process to just before the final wait statement.