Appendix A : VHDL'93 AND VHDL'87 SYNTAX SUMMARY

```
abstract_literal ::= decimal_literal | based_literal

access_type_definition ::= access subtype_indication

actual_designator ::=
      expression
      | signal_name
      | variable_name
      | file_name
      | open

actual_parameter_part ::= parameter_association_list

actual_part ::=
      actual_designator
      | function_name ( actual_designator )
      | type_mark ( actual_designator )

adding_operator ::= + | - | &

aggregate ::=
      ( element_association { , element_association } )

alias_declaration ::=
      alias alias_designator [ : subtype_indication ] is
name [ signature ] ;

alias_designator ::= identifier | character_literal |
operator_symbol

allocator ::=
      new subtype_indication
      | new qualified_expression

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

architecture_declarative_part ::=
      { block_declarative_item }

architecture_statement_part ::=
      { concurrent_statement }

array_type_definition ::=
      unconstrained_array_definition   |
      constrained_array_definition

assertion ::=
      assert condition
                  [ report expression ]
                  [ severity expression ]

assertion_statement ::=   [ label : ] assertion ;
```

```
association_element ::=
    [ formal_part => ] actual_part

association_list ::=
    association_element { , association_element }

attribute_declaration ::=
    attribute identifier : type_mark ;

attribute_designator ::= attribute_simple_name

attribute_name ::=
    prefix [ signature ] ' attribute_designator
     [ (expression ) ]

attribute_specification ::=
    attribute attribute_designator of
     entity_specification is expression ;

base ::= integer

base_specifier ::= B | O | X

base_unit_declaration ::= identifier ;

based_integer ::=
    extended_digit { [ underline ] extended_digit }

based_literal ::=
    base # based_integer [ . based_integer ] #
    [ exponent ]

basic_character ::=
    basic_graphic_character | format_effector

basic_graphic_character ::=
    upper_case_letter | digit | special_character|
     space_character

basic_identifier ::=
    letter { [ underline ] letter_or_digit }

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

bit_string_literal ::=  base_specifier " bit_value "

bit_value ::= extended_digit { [ underline ]
    extended_digit }

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

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

block_declarative_part ::=
    { block_declarative_item }

block_header ::=
    [ generic_clause
    [ generic_map_aspect ; ] ]
    [ port_clause
    [ port_map_aspect ; ] ]

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

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

block_statement_part ::=
    { concurrent_statement }
```

```
case_statement ::=
    [ case_label : ]
        case expression is
          case_statement_alternative
          { case_statement_alternative }
        end case [ case_label ] ;

case_statement_alternative ::=
    when choices =>
        sequence_of_statements

character_literal ::= ' graphic_character '

choice ::=
    simple_expression
    | discrete_range
    | element_simple_name
    | others

choices ::= choice { | choice }

component_configuration ::= -- VHDL'87
    for component_specification
        [ use binding_indication ; ]
        [ block_configuration ]
    end for ;

component_configuration ::= -- VHDL'93
    for component_specification
        [binding_indication ; ]
        [ block_configuration ]
    end for ;

component_declaration ::=
    component identifier [ is ]
        [ local_generic_clause ]
        [ local_port_clause ]
    end component [ component_simple_name ] ;

component_instantiation_statement ::= -- VHDL'87
    instantiation_label :
        component_name
          [ generic_map_aspect ]
          [ port_map_aspect ] ;


component_instantiation_statement ::= -- VHDL'93
    instantiation_label :
        instantiated_unit
          [ generic_map_aspect ]
          [ port_map_aspect ] ;

component_specification ::=
    instantiation_list : component_name

composite_type_definition ::=
    array_type_definition
    | record_type_definition

concurrent_assertion_statement ::=
    [ label : ] [ postponed ] assertion ;

concurrent_procedure_call_statement ::=
    [ label : ] [ postponed ] procedure_call ;


concurrent_signal_assignment_statement ::=
      [ label : ] [ postponed ]
          conditional_signal_assignment
    | [ label : ] [ postponed ]
          selected_signal_assignment

concurrent_statement ::=
    block_statement
    | process_statement
    | concurrent_procedure_call_statement
    | concurrent_assertion_statement
    | concurrent_signal_assignment_statement
    | component_instantiation_statement
    | generate_statement

condition ::= boolean_expression

condition_clause ::= until condition

conditional_signal_assignment ::=
    target          <= options
        conditional_waveforms ;

conditional_waveforms ::=
    { waveform when condition else }
    waveform [ when condition ]

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

configuration_declarative_item ::=
    use_clause
    | attribute_specification
    | group_declaration

configuration_declarative_part ::=
    { configuration_declarative_item }
```

```
configuration_item ::=
      block_configuration
      | component_configuration

configuration_specification ::= -- VHDL'87
      for component_specification use
        binding_indication ;

configuration_specification ::= -- VHDL'93
      for component_specification binding_indication ;

constant_declaration ::=
      constant identifier_list : subtype_indication [ :=
expression ] ;

constrained_array_definition ::=
      array index_constraint of
       element_subtype_indication

constraint ::=
      range_constraint
      | index_constraint

context_clause ::= { context_item }

context_item ::=
      library_clause
      | use_clause

decimal_literal ::= integer [ . integer ] [ exponent ]

declaration ::=
      type_declaration
      | subtype_declaration
      | object_declaration
      | interface_declaration
      | alias_declaration
      | attribute_declaration
      | component_declaration
      | group_template_declaration
      | group_declaration
      | entity_declaration
      | configuration_declaration
      | subprogram_declaration
      | package_declaration

delay_mechanism ::= -- VHDL'93
      transport
      | [ reject time_expression ] inertial

design_file ::= design_unit { design_unit }

design_unit ::= context_clause library_unit

designator ::= identifier | operator_symbol

direction ::= to | downto

disconnection_specification ::=
      disconnect guarded_signal_specification after
        time_expression ;

discrete_range ::= discrete_subtype_indication | range

element_association ::=
      [ choices => ] expression

element_declaration ::=
      identifier_list : element_subtype_definition ;

element_subtype_definition ::= subtype_indication

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

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

entity_class_entry ::= entity_class [ <> ]

entity_class_entry_list ::=
      entity_class_entry { , entity_class_entry }

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

```
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

entity_declarative_part ::=
      { entity_declarative_item }

-- VHDL'87
entity_designator ::= simple_name | operator_symbol

-- VHDL'93
entity_designator ::= entity_tag [ signature ]

entity_header ::=
      [ formal_generic_clause ]
      [ formal_port_clause ]

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

entity_specification ::=
      entity_name_list : entity_class

entity_statement ::=
      concurrent_assertion_statement
      | passive_concurrent_procedure_call_statement
      | passive_process_statement

entity_statement_part ::=
      { entity_statement }

-- VHDL'93
entity_tag :: simple_name | character_literal |
    operator_symbol

enumeration_literal ::= identifier | character_literal

enumeration_type_definition ::=
      ( enumeration_literal { , enumeration_literal } )

exit_statement ::=
      [ label : ] exit [ loop_label ] [ when condition ] ;

exponent ::= E [ + ] integer | E - integer

expression ::=
       relation { and relation }
      | relation { or relation }
      | relation { xor relation }
      | relation [ nand relation ]
      | relation [ nor relation ]
      | relation { xnor relation }

extended_digit ::= digit | letter

extended_identifier ::=
      \ graphic_character { graphic_character } \

factor ::=
      primary [ ** primary ]
      | abs primary
      | not primary

-- VHDL'87
file_declaration ::=
      file identifier : subtype_indication is [ mode ]
        file_logical_name ;
-- VHDL'93
file_declaration ::=
      file identifier_list : subtype_indication
        file_open_information ] ;

file_logical_name ::= string_expression

-- VHDL'93
file_open_information ::=
      [ open file_open_kind_expression ] is
        file_logical_name

file_type_definition ::=
      file of type_mark

floating_type_definition := range_constraint

formal_designator ::=
      generic_name
      | port_name
      | parameter_name

formal_parameter_list ::= parameter_interface_list
```

```
formal_part ::=
      formal_designator
      | function_name ( formal_designator )
      | type_mark ( formal_designator )

full_type_declaration ::=
      type identifier is type_definition ;

function_call ::=
      function_name [ ( actual_parameter_part ) ]

-- VHDL'87
generate_statement ::=
      generate_label : generation_scheme generate
       {concurrent_statement }
      end generate [ generate_label ];

-- VHDL'93
generate_statement ::=
      generate_label :
            generation_scheme generate
              [ { block_declarative_item }
            begin ]
              { concurrent_statement }
            end generate [ generate_label ] ;

generation_scheme ::=
      for generate_parameter_specification
      | if condition

generic_clause ::=
      generic ( generic_list ) ;

generic_list ::= generic_interface_list

generic_map_aspect ::=
      generic map ( generic_association_list )

graphic_character ::=
      basic_graphic_character    | lower_case_letter |
      other_special_character

group_constituent ::= name | character_literal

group_constituent_list ::= group_constituent
        { , group_constituent }

group_declaration ::=
      group identifier : group_template_name
        ( group_constituent_list ) ;

group_template_declaration ::=
      group identifier is ( entity_class_entry_list ) ;

guarded_signal_specification ::=
      guarded_signal_list : type_mark

-- VHDL'87
identifier ::=
      letter { [ underline ] letter_or_digit }

-- VHDL'93
identifier ::=
      basic_identifier | extended_identifier

identifier_list ::= identifier { , identifier }

if_statement ::=
      [ if_label : ]

            if condition then
              sequence_of_statements
            { elsif condition then
              sequence_of_statements }
            [ else
              sequence_of_statements ]
            end if [ if_label ] ;

incomplete_type_declaration ::=  type identifier ;

index_constraint ::= ( discrete_range
                        { , discrete_range } )

index_specification ::=
      discrete_range
      | static_expression

index_subtype_definition ::= type_mark range <>

indexed_name ::= prefix ( expression { , expression } )

-- VHDL'93
instantiated_unit ::=
      [ component ] component_name
      | entity entity_name [ ( architecture_identifier ) ]
      | configuration configuration_name

instantiation_list ::=
      instantiation_label { , instantiation_label }
      | others
      | all

integer ::= digit { [ underline ] digit }

integer_type_definition ::= range_constraint

interface_constant_declaration ::=
      [ constant ] identifier_list : [ in ]
        subtype_indication [ := static_expression ]
```

```
interface_declaration ::=
      interface_constant_declaration
      | interface_signal_declaration
      | interface_variable_declaration
      | interface_file_declaration

interface_element ::= interface_declaration

-- VHDL'93
interface_file_declaration ::=
      file identifier_list : subtype_indication

interface_list ::=
      interface_element { ; interface_element }

interface_signal_declaration ::=
      [signal] identifier_list : [ mode ]
     subtype_indication [ bus ] [ := static_expression ]

interface_variable_declaration ::=
      [variable] identifier_list : [ mode ]
     subtype_indication [ := static_expression ]

iteration_scheme ::=
      while condition
      | for loop_parameter_specification

label ::= identifier

letter ::= upper_case_letter | lower_case_letter

letter_or_digit ::= letter | digit

library_clause ::= library logical_name_list ;

library_unit ::=
      primary_unit
      | secondary_unit

literal ::=
      numeric_literal
      | enumeration_literal
      | string_literal
      | bit_string_literal
      | null

logical_name ::= identifier

logical_name_list ::= logical_name { , logical_name }

logical_operator ::= and | or | nand | nor | xor | xnor

loop_statement ::= [ loop_label : ]
            [ iteration_scheme ] loop
              sequence_of_statements
            end loop [ loop_label ] ;

miscellaneous_operator ::= ** | abs | not

mode ::= in | out | inout | buffer | linkage

multiplying_operator ::= * | / | mod | rem

name ::=
      simple_name
      | operator_symbol
      | selected_name
      | indexed_name
      | slice_name
      | attribute_name

next_statement ::=
      [ label : ] next [ loop_label ] [ when condition ] ;

null_statement ::= [ label : ] null ;

numeric_literal ::=
      abstract_literal
      | physical_literal

object_declaration ::=
      constant_declaration
      | signal_declaration
      | variable_declaration
      | file_declaration

operator_symbol ::= string_literal

-- VHDL'87
options ::=
      [ guarded ] [ transport ]

-- VHDL'93
options ::= [ guarded ] [ delay_mechanism ]

package_body ::=
     package body package_simple_name is
            package_body_declarative_part
     end [ package body ] [ package_simple_name ] ;
```

```
package_body_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | use_clause
    | group_template_declaration
    | group_declaration

package_body_declarative_part ::=
    { package_body_declarative_item }

package_declaration ::=
    package identifier is
        package_declarative_part
    end [ package ] [ package_simple_name ] ;

package_declarative_item ::=
    subprogram_declaration
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | component_declaration
    | attribute_declaration
    | attribute_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration

package_declarative_part ::=
    { package_declarative_item }

parameter_specification ::=
    identifier in discrete_range

physical_literal ::= [ abstract_literal ] unit_name

physical_type_definition ::=
    range_constraint
        units
          base_unit_declaration
          { secondary_unit_declaration }
        end units [ physical_type_simple_name ]

port_clause ::=
    port ( port_list ) ;

port_list ::= port_interface_list

port_map_aspect ::=
    port map ( port_association_list )

prefix ::=
    name
    | function_call

primary ::=
    name
    | literal
    | aggregate
    | function_call
    | qualified_expression
    | type_conversion
    | allocator
    | ( expression )

primary_unit ::=
    entity_declaration
    | configuration_declaration
    | package_declaration

procedure_call ::= procedure_name
        [ ( actual_parameter_part ) ]

procedure_call_statement ::=
    [ label : ] procedure_call ;

process_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | use_clause
    | group_template_declaration
    | group_declaration

process_declarative_part ::=
    { process_declarative_item }
```

```
process_statement ::=
   [ process_label : ]
      [ postponed ] process [ ( sensitivity_list ) ] [ is ]
        process_declarative_part
      begin
        process_statement_part
      end [ postponed ] process [ process_label ] ;

process_statement_part ::=
      { sequential_statement }

qualified_expression ::=
      type_mark ' ( expression )
      | type_mark ' aggregate

range ::=
      range_attribute_name
      | simple_expression direction simple_expression

range_constraint ::= range range

record_type_definition ::=
      record
            element_declaration
            { element_declaration }
      end record [ record_type_simple_name ]

relation ::=
      shift_expression [ relational_operator
shift_expression ]

relational_operator ::=
  = |/= | < | <=     | > | >=

return_statement ::=
      [ label : ] return [ expression ] ;

report_statement ::=
      [ label : ]
            report expression
              [ severity expression ] ;

scalar_type_definition ::=
      enumeration_type_definition  |
      integer_type_definition      |
      floating_type_definition     |
      physical_type_definition

secondary_unit ::=
      architecture_body
      | package_body

secondary_unit_declaration ::=
    identifier = physical_literal ;

selected_name ::= prefix . suffix

selected_signal_assignment ::=
      with expression select
            target   <= options selected_waveforms ;

selected_waveforms ::=
      { waveform when choices , }
      waveform when choices

sensitivity_clause ::= on sensitivity_list

sensitivity_list ::= signal_name { , signal_name }

sequence_of_statements ::=
      { sequential_statement }

sequential_statement ::=
      wait_statement
      | assertion_statement
      | report_statement
      | signal_assignment_statement
      | variable_assignment_statement
      | procedure_call_statement
      | if_statement
      | case_statement
      | loop_statement
      | next_statement
      | exit_statement
      | return_statement
      | null_statement

-- VHDL'93
shift_expression ::=
      simple_expression
       [ shift_operator simple_expression ]

-- VHDL'93
shift_operator ::= sll | srl | sla | sra | rol | ror

sign ::= + | -

-- VHDL'87
signal_assignment_statement ::=
      target <= [ transport ] waveform ;

-- VHDL'93
signal_assignment_statement ::=
      [ label : ] target <= [ delay_mechanism ]
                            waveform ;

signal_declaration ::=
      signal identifier_list : subtype_indication
                [ signal_kind ] [ := expression ] ;

signal_kind ::=  register | bus
```

```
signal_list ::=
      signal_name { , signal_name }
      | others
      | all

-- VHDL'93
signature ::= [ [ type_mark { , type_mark } ]
          [ return type_mark ] ]

simple_expression ::=
      [ sign ] term { adding_operator term }

simple_name ::=       identifier

slice_name ::=        prefix ( discrete_range )

string_literal ::= " { graphic_character } "

subprogram_body ::=
      subprogram_specification is
            subprogram_declarative_part
      begin
            subprogram_statement_part
      end [ subprogram_kind ] [ designator ] ;

subprogram_declaration ::=
      subprogram_specification ;

subprogram_declarative_item ::=
      subprogram_declaration
      | subprogram_body
      | type_declaration
      | subtype_declaration
      | constant_declaration
      | variable_declaration
      | file_declaration
      | alias_declaration
      | attribute_declaration
      | attribute_specification
      | use_clause
      | group_template_declaration
      | group_declaration

subprogram_declarative_part ::=
      { subprogram_declarative_item }

subprogram_kind ::= procedure | function

subprogram_specification ::=
   procedure designator [ ( formal_parameter_list ) ]
   | [ pure | impure ] function designator
        [ ( formal_parameter_list ) ]
            return type_mark

subprogram_statement_part ::=
      { sequential_statement }

subtype_declaration ::=
      subtype identifier is subtype_indication ;

subtype_indication ::=
      [ resolution_function_name ] type_mark
            [ constraint ]

suffix ::=
      simple_name
      | character_literal
      | operator_symbol
      | all

target ::=
      name
      | aggregate

term ::=
      factor { multiplying_operator factor }

timeout_clause ::= for time_expression

type_conversion ::= type_mark ( expression )

type_declaration ::=
      full_type_declaration
      | incomplete_type_declaration

type_definition ::=
      scalar_type_definition
      | composite_type_definition
      | access_type_definition
      | file_type_definition

type_mark ::=
      type_name
      | subtype_name

unconstrained_array_definition ::=
      array ( index_subtype_definition
                  { , index_subtype_definition } )
                    of element_subtype_indication

use_clause ::=
      use selected_name { , selected_name } ;

variable_assignment_statement ::=
      [ label : ] target := expression ;

variable_declaration ::=
      [ shared ] variable identifier_list :
        subtype_indication [ := expression ] ;
```

wait_statement ::=
    [label :] **wait** [sensitivity_clause]
    [condition_clause] [timeout_clause] ;

waveform ::=
    waveform_element { , waveform_element }
    | unaffected

waveform_element ::=
    value_expression [**after** *time*_expression]
    | **null** [**after** *time*_expression]

Appendix B : PACKAGE STANDARD

```
-- This is Package STANDARD as defined in the VHDL 1992 Language Reference Manual.
--  Reprinted by permission from Model Technology Inc.
--   NOTE: VCOM and VSIM will not work properly if these declarations
--         are modified.

-- Version information: @(#)standard.vhd

package standard is
    type boolean is (false,true);
    type bit is ('0', '1');
    type character is (
                nul, soh, stx, etx, eot, enq, ack, bel,
                bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
                dle, dc1, dc2, dc3, dc4, nak, syn, etb,
                can, em,  sub, esc, fsp, gsp, rsp, usp,

                ' ', '!', '"', '#', '$', '%', '&', ''',
                '(', ')', '*', '+', ',', '-', '.', '/',
                '0', '1', '2', '3', '4', '5', '6', '7',
                '8', '9', ':', ';', '<', '=', '>', '?',

                '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
                'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
                'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
                'X', 'Y', 'Z', '[', '\', ']', '^', '_',

                '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
                'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
                'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
                'x', 'y', 'z', '{', '|', '}', '~', del,

                c128, c129, c130, c131, c132, c133, c134, c135,
                c136, c137, c138, c139, c140, c141, c142, c143,
                c144, c145, c146, c147, c148, c149, c150, c151,
                c152, c153, c154, c155, c156, c157, c158, c159,

                -- the character code for 160 is there (NBSP),
                -- but prints as no char
```

```
                ' ', '¡', '¢', '£', '¤', '¥', '¦', '§',
                '¨', '©', 'ª', '«', '¬', '-', '®', '¯',
                '°', '±', '²', '³', '´', 'µ', '¶', '·',
                '¸', '¹', 'º', '»', '¼', '½', '¾', '¿',

                'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç',
                'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï',
                'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×',
                'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß',

                'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç',
                'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
                'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷',
                'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ' );

    type severity_level is (note, warning, error, failure);
    type integer is range -2147483648 to 2147483647;
    type real is range -1.0E308 to 1.0E308;
    type time is range -2147483647 to 2147483647
                units
                    fs;
                    ps = 1000 fs;
                    ns = 1000 ps;
                    us = 1000 ns;
                    ms = 1000 us;
                    sec = 1000 ms;
                    min = 60 sec;
                    hr = 60 min;
                end units;
    subtype delay_length is time range 0 fs to time'high;
    impure function now return delay_length;
    subtype natural is integer range 0 to integer'high;
    subtype positive is integer range 1 to integer'high;
    type string is array (positive range <>) of character;
    type bit_vector is array (natural range <>) of bit;
    type file_open_kind is (
                read_mode,
                write_mode,
                append_mode);
    type file_open_status is (
                open_ok,
                status_error,
                name_error,
                mode_error);
    attribute foreign : string;
end standard;
```

Appendix C : PACKAGE TEXTIO

```
------------------------------------------------------------------------
-- Package TEXTIO as defined in Chapter 14 of the IEEE Standard VHDL
--   Language Reference Manual (IEEE Std. 1076-1987), as modified
--   by the Issues Screening and Analysis Committee (ISAC), a subcommittee
--   of the VHDL Analysis and Standardization Group (VASG) on
--   10 November, 1988.  See "The Sense of the VASG", October, 1989.
-- Reprinted by permission from Model Technology Inc.
------------------------------------------------------------------------
-- Version information: %W% %G%
------------------------------------------------------------------------

package TEXTIO is
    type LINE is access string;
    type TEXT is file of string;
    type SIDE is (right, left);
    subtype WIDTH is natural;

    -- changed for vhdl92 syntax:
    file input : TEXT open read_mode is "STD_INPUT";
    file output : TEXT open write_mode is "STD_OUTPUT";

    -- changed for vhdl92 syntax (and now a built-in):
    procedure READLINE(file f: TEXT; L: out LINE);

    procedure READ(L:inout LINE; VALUE: out bit; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out bit);

    procedure READ(L:inout LINE; VALUE: out bit_vector; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out bit_vector);

    procedure READ(L:inout LINE; VALUE: out BOOLEAN; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out BOOLEAN);

    procedure READ(L:inout LINE; VALUE: out character; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out character);

    procedure READ(L:inout LINE; VALUE: out integer; GOOD : out BOOLEAN);
```

```
    procedure READ(L:inout LINE; VALUE: out integer);

    procedure READ(L:inout LINE; VALUE: out real; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out real);

    procedure READ(L:inout LINE; VALUE: out string; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out string);

    procedure READ(L:inout LINE; VALUE: out time; GOOD : out BOOLEAN);
    procedure READ(L:inout LINE; VALUE: out time);

    -- changed for vhdl92 syntax (and now a built-in):
    procedure WRITELINE(file f : TEXT; L : inout LINE);

    procedure WRITE(L : inout LINE; VALUE : in bit;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in bit_vector;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in BOOLEAN;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in character;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in integer;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in real;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0;
          DIGITS: in NATURAL := 0);

    procedure WRITE(L : inout LINE; VALUE : in string;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0);

    procedure WRITE(L : inout LINE; VALUE : in time;
          JUSTIFIED: in SIDE := right;
          FIELD: in WIDTH := 0;
          UNIT: in TIME := ns);

    -- is implicit built-in:
    -- function ENDFILE(file F : TEXT) return boolean;

    -- function ENDLINE(variable L : in LINE) return BOOLEAN;
    --
    -- Function ENDLINE as declared cannot be legal VHDL, and
    --   the entire function was deleted from the definition
    --   by the Issues Screening and Analysis Committee (ISAC),
    --   a subcommittee of the VHDL Analysis and Standardization
    --   Group (VASG) on 10 November, 1988.  See "The Sense of
    --   the VASG", October, 1989, VHDL Issue Number 0032.
end;

--*******************************************************
--**                                                   **
--** Copyright (c) Model Technology Incorporated 1991  **
--**               All Rights Reserved                 **
--**                                                   **
--*******************************************************
```

Appendix D : PACKAGE STD_LOGIC_1164

```
-- ------------------------------------------------------------------
--
--   Title     :  std_logic_1164 multi-value logic system
--   Library   :  This package shall be compiled into a library
--             :  symbolically named IEEE.
--             :
--   Developers:  IEEE model standards group (par 1164)
--   Purpose   :  This packages defines a standard for designers
--             :  to use in describing the interconnection data types
--             :  used in vhdl modeling.
--             :
--   Limitation:  The logic system defined in this package may
--             :  be insufficient for modeling switched transistors,
--             :  since such a requirement is out of the scope of this
--             :  effort. Furthermore, mathematics, primitives,
--             :  timing standards, etc. are considered orthogonal
--             :  issues as it relates to this package and are therefore
--             :  beyond the scope of this effort.
--             :
--   Note      :  No declarations or definitions shall be included in,
--             :  or excluded from this package. The "package declaration"
--             :  defines the types, subtypes and declarations of
--             :  std_logic_1164. The std_logic_1164 package body shall be
--             :  considered the formal definition of the semantics of
--             :  this package. Tool developers may choose to implement
--             :  the package body in the most efficient manner available
--             :  to them.
--             :
-- ------------------------------------------------------------------
--   modification history :
-- ------------------------------------------------------------------
--  version | mod. date:|
--   v4.200 | 01/02/92  |
-- ------------------------------------------------------------------

PACKAGE std_logic_1164 IS
    -- logic state system  (unresolved)
    ------------------------------------------------------------------
```

```
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                         '-'   -- Don't care
                       );
    ------------------------------------------------------------------
    -- unconstrained array of std_ulogic for use with the resolution function
    ------------------------------------------------------------------
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

    ------------------------------------------------------------------
    -- resolution function
    ------------------------------------------------------------------
    FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

    ------------------------------------------------------------------
    -- *** industry standard logic type ***
    ------------------------------------------------------------------
    SUBTYPE std_logic IS resolved std_ulogic;

    ------------------------------------------------------------------
    -- unconstrained array of std_logic for use in declaring signal arrays
    ------------------------------------------------------------------
    TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;

    ------------------------------------------------------------------
    -- common subtypes
    ------------------------------------------------------------------
    SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
    SUBTYPE X01Z    IS resolved std_ulogic RANGE 'X' TO 'Z'; -- ('X','0','1','Z')
    SUBTYPE UX01    IS resolved std_ulogic RANGE 'U' TO '1'; -- ('U','X','0','1')
    SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z';
       -- ('U','X','0','1','Z')

    ------------------------------------------------------------------
    -- overloaded logical operators
    ------------------------------------------------------------------

    FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
--  function "xnor" ( l : std_ulogic; r : std_ulogic ) return ux01;
    FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;

    ------------------------------------------------------------------
    -- vectorized overloaded logical operators
    ------------------------------------------------------------------
    FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
```

```
    FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
--  ------------------------------------------------------------------------
--  Note : The declaration and implementation of the "xnor" function is
--  specifically commented until at which time the VHDL language has been
--  officially adopted as containing such a function. At such a point,
--  the following comments may be removed along with this notice without
--  further "official" ballotting of this std_logic_1164 package. It is
--  the intent of this effort to provide such a function once it becomes
--  available in the VHDL standard.
--  ------------------------------------------------------------------------
--  function "xnor" ( l, r : std_logic_vector  ) return std_logic_vector;
--  function "xnor" ( l, r : std_ulogic_vector ) return std_ulogic_vector;

    FUNCTION "not"  ( l : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector;

    ---------------------------------------------------------------------
    -- conversion functions
    ---------------------------------------------------------------------
    FUNCTION To_bit       ( s : std_ulogic;        xmap : BIT := '0') RETURN BIT;
    FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0')
        RETURN BIT_VECTOR;
    FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0')
        RETURN BIT_VECTOR;

    FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic;
    FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector;
    FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
    FUNCTION To_StdULogicVector ( b : BIT_VECTOR ) RETURN std_ulogic_vector;
    FUNCTION To_StdULogicVector ( s : std_logic_vector) RETURN std_ulogic_vector;

    ---------------------------------------------------------------------
    -- strength strippers and type convertors
    ---------------------------------------------------------------------
    FUNCTION To_X01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
    FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
    FUNCTION To_X01  ( s : std_ulogic        ) RETURN  X01;
    FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
    FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
    FUNCTION To_X01  ( b : BIT               ) RETURN  X01;

    FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN  std_logic_vector;
    FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
    FUNCTION To_X01Z ( s : std_ulogic        ) RETURN  X01Z;
    FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
    FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
    FUNCTION To_X01Z ( b : BIT               ) RETURN  X01Z;

    FUNCTION To_UX01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
    FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
    FUNCTION To_UX01  ( s : std_ulogic        ) RETURN  UX01;
    FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
    FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
    FUNCTION To_UX01  ( b : BIT               ) RETURN  UX01;

    ---------------------------------------------------------------------
    -- edge detection
    ---------------------------------------------------------------------
    FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
    FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;

    ---------------------------------------------------------------------
    -- object contains an unknown
    ---------------------------------------------------------------------
    FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN;
    FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN;
    FUNCTION Is_X ( s : std_ulogic        ) RETURN  BOOLEAN;
END std_logic_1164;
```

Appendix E : VHDL PREDEFINED ATTRIBUTES

VHDL Attributes

Attribute	Prefix	Comments
T'base	Type	Base type of T. Must be prefix to another attribute
T'left	scalar	The left bound of T, result of type T
T'right	type/ST	The right bound of T, result of type T
T'high	scalar	The upper bound of T, result of type T
T'low	type/ST	The lower bound of T, result of type T
T'Ascending VHDL'93	scalar type/ST	TRUE if type T is ascending
T'image(X) VHDL'93	scalar type/ST	Function which converts scalar object X of type T into string
T'value(X) VHDL'93	scalar type/ST	Function which converts object X of type string into scalar of type T
T'pos(X)	discrete /PT/ST	Function which returns a universal integer representing the position number of parameter X of type T. First position = 0.
T'val(X)	discrete /PT/ST	Function which returns of base type T the value whose position is the universal integer value corresponding to X.
T'succ(X)	discrete /PT/ST	Function returning a value of type T whose value is the position number one greater than the one of the parameter. It is an error if X = T'high or if does not belong to the range T'low to T'high
T'pred(X)	discrete /PT/ST	Function returning a value of type T whose value is the position number one less than the one of the parameter. It is an error if X = T'low or if does not belong to the range T'low to T'high

T'leftof(X)	discrete /PT/ST	Function which returns the value that is to the left of parameter X of type T. Result type is of type T. Error if X = T'left
T'rightof(X)	discrete /PT/ST	Function which returns the value that is to the right of parameter X of type T. Result type is of type T. Error is X = T'right
A'left(N)	Array*	Function which returns the left bound of of the Nth index range of A. X is of type universal integer. Result type is of type of the left bound of the left index range of A. N = 1 if omitted.
A'right(A)	Array*	Same as A'left(N), except right bound is returned
A'high(N)	Array*	Function which returns the upper bound of the range of A. Result type is the type of the Nth index range of A. N = 1 if omitted.
A'low(N)	Array*	Same as A'high(N), e lower bound is returned.
A'range(N)	Array*	The range of A'left(N) to A'right(N)
A'reverse range(N)		The range of A'right(N) to A'left(N)
A'length	Array*	returns 0 is array os null. Else, returns T'pos(A'high(N)) - T'pos(A'low(N) where T is the subtype of the Nth index of A.
A'Ascending	Array*	True if Nth index range of A is defined in an ascending range, else returns false.

PT = physical type
ST = Subtype
Array* = Any prefix that is appropriate for an array object, (e.g. type, variable, signal) or alias therof, or that denotes a constrained array subtype

Summary of the VHDL Signal Attributes

S'event	Function returning a **Boolean** which identifies if signal S has a new value assigned onto this signal (i.e. value is different that last value). **if** Clk'event **then** -- if Clk just changed in value then ... . **wait until** Clk'event **and** Clk = '1'; -- rising edge of clock
S'active	Function returning a **Boolean** which identifies if signal S had a new assignment made onto it (whether the value of the assignment is the SAME or DIFFERENT. **if** Data'active **then** -- New assignment of Data
S'transaction	**Implicit signal** of type **bit** which is created for signal S when it S'transaction is used in the code. This implicit signal is NOT declared since it is implicitly defined. This signal toggles in value (between '0' and '1') when signal S had a new assignment made onto it (whether the value of the assignment is the SAME or DIFFERENT. The user should NOT rely on its VALUE. **wait on** ReceivedData'transaction; -- process is -- sensitive to ReceiveData changing value

S'delayed(T)	**Implicit signal** of the same **base type as S**. It represents the value of signal S delayed by a time Tn. Thus, the value of S'delayed(T) at time Tn is always equal to the value of S at time Tn -t. For example, the value of S'delayed(5 ns) at time 1000 ns is the value of S at time 995 ns. Note if time is omitted, it defaults to 0 ns. **wait on** Data'transaction; **case** BV2'(Data'Delayed & Data) **is** -- Data @ last delta time **when "X0" =>** ... -- from X to 0 transition **when "10:" =>** ... -- from 1 to 0 transition **when others =>** ... -- **end case;**
S'stable(T)	**Implicit signal** of **Boolean** type. This implicit signal is true when an event (change in value) has NOT occurred on signal S for T time units, and the value FALSE otherwise. If time is omitted, it defaults to 0 ns. **if** Data'stable(40 **ns**) **then** -- met set up time
S'quiet(T)	**Implicit signal** of **Boolean** type. This implicit signal is true when the signal has been quiet (i.e. no activity or signal assignment) for T time units, and the value FALSE otherwise. If time is omitted, it defaults to 0 ns. **if** Data'quiet(40 **ns**) **then** -- Really quiet, not even an assignment of -- the same value during the last T time units
S'last_event	The amount of **time** that has elapsed since the last event (change in value) occurred on signal S. If there was no previous event, it returns Time'high. **variable** : TsinceLastEvent : time; -- .. TsinceLastEvent := Data'last_event;
S'last_active	The amount of **time** that has elapsed since the last activity (assignment) occurred on signal S. If there was no previous event, it returns Time'high. **variable** : TsinceLastEvent : time; -- .. TsinceLastEvent := Data'last_active;
S'last_value	Function of the **base type of S** returning the previous value of S, immediately before the last change of S. **wait on** Data'**transaction**; **case** BV2'(Data'last_value & Data) **is** -- Data @ last value **when "X0" =>** ... -- from X to 0 transition **when "10" =>** ... -- from 1 to 0 transition **when others =>** ... -- **end case;**

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

T

U

V

W