

# 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 [ /oop_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 ::= srl | sll | 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
```

```

'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç',
'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç',
'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç',
'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç', 'ç',
'À', 'À', 'À', 'À', 'À', 'À', 'È', 'Ç',
'È', 'È', 'È', 'È', 'Ì', 'Í', 'Î', 'Î',
'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 vhd192 syntax:
  file input : TEXT open read_mode is "STD_INPUT";
  file output : TEXT open write_mode is "STD_OUTPUT";

  -- changed for vhd192 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 vhd192 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 ENDLIN(variable L : in LINE) return BOOLEAN;
--
-- Function ENDLIN 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 package 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 thereof, or that denotes a constrained array subtype

### Summary of the VHDL Signal Attributes

S'event	Function returning a <b>Boolean</b> 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 ... .... <b>wait until</b> Clk'event and Clk = '1'; -- rising edge of clock
S'active	Function returning a <b>Boolean</b> 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	<b>Implicit signal</b> 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. <b>wait on</b> ReceivedData'transaction; -- process is -- sensitive to ReceiveData changing value

S'delayed(T)	<p><b><u>Implicit signal</u></b> 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.</p> <pre> wait on Data'transaction; case BV2'(Data'Delayed &amp; Data) is -- Data @ last delta time   when "X0" =&gt; ... -- from X to 0 transition   when "10:" =&gt; ... -- from 1 to 0 transition   when others =&gt; ... -- end case;</pre>
S'stable(T)	<p><b><u>Implicit signal</u></b> 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.</p> <pre> if Data'stable(40 ns) then -- met set up time</pre>
S'quiet(T)	<p><b><u>Implicit signal</u></b> 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.</p> <pre> if Data'quiet(40 ns) then -- Really quiet, not even an assignment of   -- the same value during the last T time units</pre>
S'last_event	<p>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.</p> <pre> variable : TsinceLastEvent : time; -- .. TsinceLastEvent := Data'last_event;</pre>
S'last_active	<p>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.</p> <pre> variable : TsinceLastEvent : time; -- .. TsinceLastEvent := Data'last_active;</pre>
S'last_value	<p>Function of the base type of S returning the previous value of S, immediately before the last change of S.</p> <pre> wait on Data'transaction; case BV2'(Data'last_value &amp; Data) is -- Data @ last value   when "X0" =&gt; ... -- from X to 0 transition   when "10" =&gt; ... -- from 1 to 0 transition   when others =&gt; ... -- end case;</pre>

# INDEX

---

## A

- Access type, 72
- Active, 121, 123
- Aggregates, 62-66, 159
- Alias, 81, 82
- Anonymous array, 70
- Architecture, 16, 21, 150
- Array, 61
  - Constrained, 61
  - Multidimensional, 68
  - Unconstrained, 63, 178
- Ascending attribute, 78, 80
- Assertion, 169-170
- Assignment, 39, 156-157
- Associations
  - element, 62
  - port, 162-164
  - subprogram, 194
- Attributes, 77
  - Declarations, 231
  - Predefined
    - 'active, 123
    - 'ascending, 78, 80
    - 'base, 78
    - 'delayed, 123
    - 'event, 123
    - 'high, 78, 80
    - 'image, 79
    - 'last\_active, 124
    - 'last\_event, 124
    - 'last\_value, 124
    - 'left, 78, 80
    - 'leftof, 79
    - 'low, 78, 80
    - 'pos, 79
    - 'pred, 79
    - 'quiet, 123
    - 'range, 80
    - 'reverse\_range, 80
    - 'right, 78, 80
    - 'rightof, 79
    - 'stable, 123
    - 'succ, 79
    - 'transaction, 123
    - 'val, 79
    - 'value, 79
  - Specification, 234, 236
  - User defined, 232-234

## B

- Based literals, 40
- Base attribute, 78
- Binding
  - Configured components, 246
  - Default, 239
  - Deferred, 246
  - Explicit, 240
- Bit type, 41
- Bit\_vector, 41
- Block statement, 171
- Boolean type, 56
- Bus functional model (BFM), 249
  - Architectural command, 250, 256
  - Instruction file command, 250, 253
- Buffer port, 16

## C

- Case, 92-98
- Character type, 54
- Class, 5, 6, 180
- Code examples
  - also see "Models"*
  - Access type, 72
  - Aggregates in signal assignment, 159
  - Alias'87, 82
  - Alias'93, 83
  - Architecture, 22
  - Architectural command (BFM), 256
  - Array, 63
    - Array of Boolean, 63
    - Constrained, 65
    - Timing specification, 69
    - Unconstrained, 66
  - Assert statement, 170
  - Attributes
    - Declarations, 233
    - Predefined, 78
    - Specifications, 237
    - User defined, 237
  - Binding - default, 240
  - Bit string literal, 42
  - Block statement, 172
  - Case, 94, 96, 97, 98
  - Component, 149
  - Component instantiation, 161
  - Concatenation, 46

**Code examples**

- Concatenation Operator, 68
- Concurrent procedure call, 166
- Concurrent procedures, 203
- Concurrent statement, 137
- Configuration declaration, 244, 302
- Configuration specification, 241
- Deferring binding, 247
- Entity, 11, 147
- Enumeration type, 52
- File, 222, 224, 225
- Functions, 195
- Generate statement, 168
- Incomplete signal assignments, 327
- Integer operations, 51
- Lexically identical types, 53
- Loop,
  - For, 101, 102
  - Simple, 100
  - While, 100
- Memory declaration & initialization, 68
- Mod operator, 47
- Operations on Real, 60
- Operations on type time, 58
- Overloaded enumeration literals, 54
- Overloaded operators, 200, 201
- Physical types, 58
- Port association and type conversion, 164
- Port association rules, 162
- Process rules, 154
- Pseudo-random Generator, 47
- Pseudo-random generator, 155
- Register inference, 324, 325
- Resolution function, 199, 218
- Setup and Hold, 125, 203, 271
- Shift operators '93, 45
- Short-circuit, 43
- State machine
  - Explicit, 319
  - Implicit, 319
- Static expression, 88
- String to 80 characters, 212
- Subelement association, 190
- Subprogram, 179
  - Drivers, 188
  - Implicit signals, 185
  - Initialization rules, 183
  - Restrictions, 181
  - Static signals, 188
  - Subtypes, 186
- Task/protocol, 192

- Testbench of a memory, 271
- TextIO, 222, 224
- Transmission line, 293
- Type conversion, 59
- UART Receiver, 281
- UART Testbench, 296
- UART Transmitter, 279
- Verifier (for a UART), 295
- VITAL counter, 310, 314
- Wait for 0, 137
- Comments, 12
- Compilation, 10, 22
  - Example, 25
  - Order, 226
- Component
  - instantiation, 159
  - configuration, 246
  - declaration, 160
- Composite, 61
- Component Instantiation, 159-162
- Concatenation operator, 45
- Concurrent assertion, 169
- Concurrent procedure call, 166
- Concurrent procedures, 202-206
- Concurrent signal assignment, 156
- Conditional signal assignment, 157
- Configuration
  - Configured components, 246
  - Configuration declaration, 243-245
  - Deferred binding, 246
  - Specification, 239
- Constant, 5, 6
  - Deferred, 213
  - in Subprograms, 180
- Control structure
  - Case, 92, 95
  - If 89
  - Loop, 99

**D**

- Deferred constants, 213
- Delimiters, 38
- Delayed attribute, 123
- Delta time
  - Concept, 128
  - Concurrent statements, 136
  - Modeling methods, 135
  - Use of variables, 137
  - Wait for 0 ns, 135

Design units, 8  
Digit, 40  
Discrete Range, 64  
Discrete type, 49  
Driver, 111- 114, 133, 187

## E

Elaboration, 22. 24. 25  
Entity, 10, 145  
Enumeration  
    Ordering, 54  
    Overload, 54  
    Predefined, 54  
    Type, 51  
Event attribute, 121, 123  
Exit statement. 104  
Expression, qualified 54, 73

## F

File, 5, 8, 74-77, 220-225  
Function  
    Definition, 175  
    Impure, 195  
    Pure, 195  
    Resolution, 109, 198, 218  
Functional model, 249-261  
    Architectural command, 256-261  
    Intruction file command, 253-256

## G

Generate statement, 167  
Generic, 12  
Guidelines  
    Alias'93, 81  
    Array directions, 67  
    Array -Unconstrained, 67  
    Arrays, 64  
    Attributes, 77, 232, 235  
    Block statement, 173  
    Boolean, 56  
    Case statement, 92  
    Buffer, 178  
    Buffer port, 16  
    Capitalization, 37  
    Class in subprograms, 182  
    Comments, 12  
    Component association list, 162

Component declaration, 239  
Concurrent assertion statement, 169  
Concurrent procedures, 203  
Configuration declaration, 240  
Constants, 6  
Defaults in subprograms, 184  
Defining a condition, 91  
Delimiters, 40  
Entity, 150  
Enumeration, 51  
External Stimuli, 263  
File and signal rule, 72  
File naming convention, 9  
For loop, 101  
Function, 195  
Generic, 11, 13  
Header, 12  
If statement, 91  
Identifiers, 30, 34, 35  
Indentation, 13  
Initialization, 113  
Interface declarations, 14  
Libraries, 25  
Library, 25  
Line length, 14  
Loop, 99  
Loop statements, 104, 105  
Mode in subprograms, 183  
Named notation, subprograms, 194  
Naming identifiers, 31-33  
Next, 103  
Packages  
    Contents, 209  
    Information hiding, 211  
Paths with identifiers, 36  
Port interface of array types, 146  
Procedures, 176  
Records, 70  
Side effects, 190, 191  
Signal attributes, 124  
Signal vs. variable, 8  
State machine, 320  
Subprogram array size and  
    direction, 179  
Subprogram Arrays, 178  
Subprograms modes, 180  
Subprograms with signals, 184  
Suffixes for classes, 6, 12



**Guidelines****Synthesis**

Conditional statements, 326

Levels, 317

RTL, 321

**Testbenches**, 262, 263

Approach, 264

Architectures, 268

**Testbenches**

BFM, 252

Global signals, 270

Type identifiers, 36

Types, 52, 58

Types and subtypes, 49

Unconstrained arrays, 64

Use clause, 214

Verification, 263

Waveform element, 139

Wait, 131

**H**

High attribute, 78, 80

**I**

Identifiers, 29–37

If statement, 89

Image attribute, 79

Inertial delay, 138–142

Initialization, 24

Integer type, 49

Integer -- universal, 59

Interface declaration, 117

**L**

Last\_active attribute, 124

Last\_event attribute, 124

Last\_value attribute, 124

Levels of Descriptions, 2

Language Reference Manual (LRM)

section #

1.1 -- Entity, 10, 145

1.1.1.1 -- Generic, 12

1.1.1.2 -- Ports, 162

1.2 -- Architecture, 16, 150

2.1 -- Subprograms, 175

2.3 -- Subprogram overloading, 194

2.3.1 -- Operator overloading, 200

2.4 -- Resolution function, 109, 198

2.5 -- Package declarations, 209

2.6 -- Deferred constants, 213

2.6 -- Package bodies, 211

3.1 -- Scalar types, 49

3.1.1 -- Enumeration types, 51

3.1.1.1 -- Enumeration - predefined, 54

3.1.3 -- Physical types, 57

3.1.4.1 -- Real type, 60

3.2.1 Array types, 61

3.3 -- Access types, 72

4.1 Type declarations, 4, 47

4.2 Subtype declarations, 48

4.3 -- Class, 5

4.3.1.1 -- Constant, 6

4.3.2 -- Interface declaration, 117

4.3.3. -- Alias, 81

4.4 -- Attribute declarations, 231

5 -- Specifications, 234

5.2.1.2 -- Component instantiation, 159

7.2 -- Operators, 42

7.2.2 -- Relational operators, 43

7.2.3 -- Shift operators, 44

7.2.4 -- Concatenation Operator, 45

7.2.6 -- Mod and Rem, 46

7.3.1 -- Logical operators, 43

7.3.2 Aggregates, 62

7.3.5 -- type conversion, 58

7.4 -- Static expression, 87

8.1 -- Wait statement, 127

8.12 -- Return Statement, 195

8.4 -- Inertial/transport delays, 139

8.7 -- If statement, 89

8.8 -- Case statement, 92

8.9 -- Loop statement, 99

9.1 -- Block statement, 171

9.2 Process, 18, 152

9.3 -- Concurrent procedure call, 166

9.4 -- Concurrent assertion, 169

9.5.1 -- Conditional signal  
assignment, 157

9.5.2 -- Selected signal assignment, 157

9.7 -- Generate statement, 167

10.4 -- Use clauses, 214

10.5 -- Overload context, 54

11.1 --Design units, 8

12 -- Elaboration, 25

12.6.1 -- Driver, 112

12.6.1 -- Drivers, 111

**Language Reference Manual (LRM)**

## section #

12.6.2 -- Propagation of signal  
values, 117

12.6.4 -- Simulation cycle, 121, 133

13.3 -- Identifiers, 29

13.4.1 -- Decimal literals, 40

13.4.2 -- Based literals, 40

13.7 -- Bit string literal, 41

13.8 --- Comments, 12

14.1 -- Predefined attributes, 77

Left attribute, 78, 80

Leftof attribute, 79

Library, 22-23

Low attribute, 78, 80

**Literals**

Based, 40

Bit string, 41

Character, 41

Logical operators, 43

**Loop**

For, 101

For loop, 101

Simple loop, 99

While loop, 100

**M**

Methodology, 3

**Models***also see "Code examples"*

Bit reversal, 101

Count 1's in bit vector, 103

Memory, 254

Physical types, 58

Pseudo-random generator, 155

Resolved Boolean, 199

Setup and Hold, 125, 203

Setup and Hold package, 271

String to 80 characters, 212

Transmission line, 293

UART Package, 287

UART Receiver, 281

UART Testbench, 296

UART Transmit protocol, 290

UART Transmitter, 279

UART Verifier, 295

Units package, 232

Modulus operator, 46

**N**

Named notation, 194

Natural subtype, 49

**Naming**

Component instantiations, 31

Timing parameters, 31

Next statement, 103

Now function, 57

Null, 73

**O**

Object, 5

**Operators**

Concatenation, 45

Definition, 42

Logical, 43

Mod, 46

Precedence, 42

Relational, 43

Rem, 46

Shift, 44

Short circuit, 43

overloading, 200-202

**Open**

Port, 162-163

Ordering operators, 53

**Others**

in Array aggregate, 65

in Case statement, 92

Overloaded operators, 200

Overloading subprograms, 194

**P**

Package, 4, 162

Body, 211

declaration, 209

Physical type, 57

**Port,**

Association rules, 162-165

Array declaration, 146

Buffer, 16

Guidelines, 146-150

In, 15

InOut, 15

Initialization, 116

Out, 15

Pos attribute, 79

Positive subtype, 49  
Positional notation, 194  
Pred, 79  
Procedures, 175  
    *also see Concurrent procedure*  
Process, 18, 152-156  
Propagation of signal values, 117

## Q

Qualified Expression, 54, 73  
Quiet attribute, 123

## R

Range attribute, 80  
Real type, 60  
Record type, 70  
Reject Inertial delay, 141  
Remainder operator, 46  
Reserved Words, 38  
Resolution function, 109, 198-200, 218  
Reverse\_range attribute, 80  
Return statement, 195  
Right attribute, 78, 80  
Rightof attribute, 79

## S

Scalar type, 49-60  
Scope of visibility, 21, 148  
Selected signal assignment, 157  
Sequential statement, 152  
Shift operators, 44  
Side effects, 190  
Signal, 5, 7  
    Attributes, 121-126  
    Implicit, 122  
    Packages, 217  
    in Subprograms, 180, 188-190  
Signal assignment  
    Concurrent, 156  
    Conditional, 157  
Simulation, 24, 26, 27  
    Cycle, 121, 133  
    Time, 57  
Slice, 67

## Specifications

    Attribute, 234-238  
    Configuration, 239-243  
    Disconnect, 234  
Stable attribute, 123  
State machine, 318  
Static expression, 87  
Std\_uLogic, 55  
Std\_Logic, 198  
String type, 73, 212, 272, 348  
Subelement association, 189  
Subprogram, 175  
    drivers, 187  
    Implicit signal attributes, 184  
    Initialization, 183  
    interface class, 180  
    Matching elements in calls, 189  
    Packages, 220  
    Passing subtypes, 186  
    Overloading, 194  
    Side effects, 190-193  
    Static signals, 188

Subtype, 4, 48

Succ attribute, 79

## Synthesis

    Architectural design, 322  
    Architecture, 330  
    Attributes, 331  
    Classes of objects, 329  
    Configuration, 330  
    Constructs, 323  
    Design units, 330  
    Entity, 330  
    If statement, 326  
    Library, 330  
    Memory, 68  
    Methodology, 317  
    Operators, 329  
    Package, 330  
    Register inference with variables, 323  
    Resource sharing, 332  
    RTL, 320  
    Sequential statements, 331  
    State machine, 318  
    Structural, 321  
    Subprogram, 330  
    Types, 328

**T**

Task / low level protocol modeling, 191

**Testbench**

Architectures, 268-274

Memory, 270

Methodology, 264-267

Overview, 261-264

Validation Plan, 265

UART, 268

TextIO, 220-225, 298, 349

Time type, 57

Timeout clause (Wait for), 130

Transaction attribute, 123

Transport delay, 138-142

Type, 4, 6

Access, 72

Array, 61

Boolean, 56

Conversion, 58

Conversion in association lists, 164

Declarations, 47

Discrete, 49

Type, 4, 6

Enumeration, 51

Physical, 57

Predefined in Standard, 4

Real, 60

Record, 70

Scalar, 49

**U****UART**

Architecture, 277

Configuration, 302

Project, 277

Receive protocol, 292

Receiver, 280

Testbench, 283, 296-301

Transmission line model, 293

Transmitter, 277, 278

Verifier, 294

Unconstrained array, 63-68

Subprograms, 178-179

Use clauses, 214-216

**V**

val attribute, 79

value attribute, 79

Variable, 5, 7

in Subprograms, 180

in Packages (shared), 209

Verifier, 268-269, 294

Visibility, 21, 146-148

**VHDL**

Definition, 1

**VITAL**

Definition, 305

Distributed delay style, 313-315

Features, 306

Pin-to-Pin delay style, 308-312

Purpose, 306

Timing parameters, 308

Types, 307

**W****Wait**

Statement, 127

Wait, 131

Wait for, 130

Wait on, 128

Wait until, 128

Waveforms (Test vectors), 159