Annex A

Syntax summary

(informative)

This annex provides a summary of the syntax for VHDL. Productions are ordered alphabetically by left-hand non-terminal name. The clause number indicates the clause where the production is given.

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 ::= % $1.2$
architecture identifier of entity_name is
architecture_declarative_part
begin
architecture_statement_part
end [ architecture ] [ architecture_simple_name ] ;

architecture_declarative_part ::= % $1.2.1$
{ block_declarative_item }

architecture_statement_part ::= % $1.2.2$
{ concurrent_statement }

array_type_definition ::= % $3.2.1$
unconstrained_array_definition | constrained_array_definition

assertion ::= % $8.2$
assert condition
[ report expression ]
[ severity expression ]

assertion_statement ::= [ label : ] assertion ; % $8.2$

association_element ::= % $4.3.2.2$
[ formal_part => ] actual_part

association_list ::= % $4.3.2.2$
association_element { , association_element }

attribute_declaration ::= % $4.4$
attribute identifier : type_mark ;

attribute_designator ::= attribute_simple_name % $6.6$

attribute_name ::= % $6.6$
prefix [ signature ] . attribute_designator [ ( expression ) ]

attribute_specification ::= % $5.1$
attribute attribute_designator of entity_specification is expression ;

base ::= integer % $13.4.2$

base_specifier ::= B | O | X % $13.7$

base_unit_declaration ::= identifier ; % $3.4.3^1$ % $|$ % $3.4.3^1$

based_integer ::= % $13.4.2$
extended_digit { [ underline ] extended_digit }

based_literal ::= % $13.4.2$
base # based_integer [ . based_integer ] # [ exponent ]

basic_character ::= % $13.1$
basic_graphic_character | format_effector

---

1. The LHS of this production was renamed to "primary_unit_declaration" in 1076-1993.
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_part ::=  
    { block_declarative_item }  

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

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

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

§ 13.1  
§ 13.3.1  
§ 5.2.1  
§ 13.7  
§ 13.7  
§ 1.3.1  
§ 1.2.1  
§ 9.1  
§ 9.1  
§ 1.3.1
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 ::= 
   for component_specification
   [ binding_indication ; ]
   [ block_configuration ]
   end for ;

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

component_instantiation_statement ::= 
   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

Copyright © 2000, IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
concurrent_assertion_statement ::= [§ 9.4]
   [ label : ] [ postponed ] assertion ;

concurrent_procedure_call_statement ::= [§ 9.3]
   [ label : ] [ postponed ] procedure_call ;

concurrent_signal_assignment_statement ::= [§ 9.5]
   [ label : ] [ postponed ] conditional_signal_assignment
   [ [ label : ] [ postponed ] selected_signal_assignment ]

concurrent_statement ::= [§ 9]
   block_statement
   process_statement
   concurrent_procedure_call_statement
   concurrent_assertion_statement
   concurrent_signal_assignment_statement
   component_instantiation_statement
   generate_statement

condition ::= boolean_expression [§ 8.1]

condition_clause ::= until condition [§ 8.1]

conditional_signal_assignment ::= target <= options conditional_waveforms ;

conditional_waveforms ::= [§ 9.5.1]
   { waveform when condition else } waveform [ when condition ]

configuration_declaration ::= [§ 1.3]
   configuration identifier of entity_name is
   configuration_declarative_part
   block_configuration
end [ configuration ] [ configuration_simple_name ] ;

configuration_declarative_item ::= [§ 1.3]
   use_clause
   | attribute_specification
   | group_declaration

configuration_declarative_part ::= [§ 1.3]
   { configuration_declarative_item }

configuration_item ::= [§ 1.3.1]
   block_configuration
   | component_configuration

configuration_specification ::= [§ 5.2]
   for component_specification binding_indication ;

constant_declaration ::= [§ 4.3.1.1]
   constant identifier_list : subtype_indication [ := expression ] ;

constrained_array_definition ::= [§ 3.2.1]
   array index_constraint of element_subtype_indication
constraint ::= [§ 4.2]
    range_constraint
    | index_constraint

context_clause ::= { context_item } [§ 11.3]

context_item ::= library_clause
    | use_clause

decimal_literal ::= integer [. integer ] [ exponent ] [§ 13.4.1]

declaration ::= [§ 4]
    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 ::= [§ 8.4]
    transport
    | [ reject time_expression ] inertial

design_file ::= design_unit { design_unit } [§ 11.1]

design_unit ::= context_clause library_unit [§ 11.1]

designator ::= identifier | operator_symbol [§ 2.1]

direction ::= to | downto [§ 3.1]

disconnection_specification ::= [§ 5.3]
    disconnect guarded_signal_specification after time_expression ;

discrete_range ::= discrete_subtype_indication | range [§ 3.2.1]

element_association ::= [§ 7.3.2]
    [ choices => ] expression

element_declaration ::= [§ 3.2.2]
    identifier_list : element_subtype_definition ;

element_subtype_definition ::= subtype_indication [§ 3.2.2]

element_association ::= [§ 7.3.2]
    identifier_list : element_subtype_definition ;

element_subtype_definition ::= subtype_indication [§ 3.2.2]

element_aspect ::= [§ 5.2.1.1]
    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 } |

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 }

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

file_declaration ::=  
  file identifier_list : subtype_indication [ file_open_information ] ;

file_logical_name ::=  string_expression

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 ) ]  

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  

identifier ::= basic_identifier | extended_identifier  

identifier_list ::= identifier { , identifier }
if_statement ::=                           [§ 8.7]
  [ 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 ;                           [§ 3.3.1]

index_constraint ::= ( discrete_range { , discrete_range } )                   [§ 3.2.1]

index_specification ::= discrete_range |
                       static_expression                           [§ 1.3.1]

index_subtype_definition ::= type_mark range <>                               [§ 3.2.1]

indexed_name ::= prefix ( expression { , expression } )                        [§ 6.4]

instantiated_unit ::=                           [§ 9.6]
  [ component ] component_name
  | entity entity_name [ ( architecture_identifier ) ]
  | configuration configuration_name

instantiation_list ::=                           [§ 5.2]
  instantiation_label { , instantiation_label }
  | others
  | all

integer ::= digit { [ underline ] digit }                                      [§ 13.4.1]

integer_type_definition ::= range_constraint                                   [§ 3.1.2]

interface_constant_declaration ::=                           [§ 4.3.2]

interface_declaration ::=                           [§ 4.3.2]
  interface_constant_declaration
  | interface_signal_declaration
  | interface_variable_declaration
  | interface_file_declaration

interface_element ::= interface_declaration                                    [§ 4.3.2.1]

interface_file_declaration ::=                           [§ 4.3.2]
  file identifier_list : subtype_indication

interface_list ::=                           [§ 4.3.2.1]
  interface_element { ; interface_element }

interface_signal_declaration ::=                           [§ 4.3.2]
  [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  
      [§ 9.7]

letter ::=  upper_case_letter | lower_case_letter  
      [§ 13.3.1]

letter_or_digit ::=  letter | digit  
      [§ 13.3.1]

library_clause ::=  library logical_name_list ;  
      [§ 11.2]

library_unit ::=  
      primary_unit  
      | secondary_unit  
      [§ 11.1]

literal ::=  
      numeric_literal  
      | enumeration_literal  
      | string_literal  
      | bit_string_literal  
      | null  
      [§ 7.3.1]

logical_name ::=  identifier  
      [§ 11.2]

logical_name_list ::=  logical_name { , logical_name }  
      [§ 11.2]

logical_operator ::=  and | or | nand | nor | xor | xnor  
      [§ 7.2]

loop_statement ::=  
    [ loop_label : ]  
    [ iteration_scheme ] loop  
    sequence_of_statements  
    end loop [ loop_label ] ;  
      [§ 8.9]

miscellaneous_operator ::=  ** | abs | not  
      [§ 7.2]

mode ::=  in | out | inout | buffer | linkage  
      [§ 4.3.2]

multiplying_operator ::=  * | / | mod | rem  
      [§ 7.2]

name ::=  
      simple_name  
      | operator_symbol  
      | selected_name  
      | indexed_name  
      | slice_name  
      | attribute_name  
      [§ 6.1]

next_statement ::=  
    [ label : ] next [ loop_label ] [ when condition ] ;  
      [§ 8.10]

null_statement ::=  [ label : ] null ;  
      [§ 8.13]
numeric_literal ::= [§ 7.3.1]
  abstract_literal
  | physical_literal

object_declaration ::= [§ 4.3.1]
  constant_declaration
  | signal_declaration
  | variable_declaration
  | file_declaration

operator_symbol ::= string_literal [§ 2.1]

options ::= [ guarded ] [ delay_mechanism ] [§ 9.5]

package_body ::= [§ 2.6]
  package body package simple_name is
  package_body_declarative_part
  end [ package body ] [ package_simple_name ];

package_body_declarative_item ::= [§ 2.6]
  subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | shared_variable_declaration
  | file_declaration
  | alias_declaration
  | group_declaration

package_body_declarative_part ::= [§ 2.6]
  { package_body_declarative_item }

package_declaration ::= [§ 2.5]
  package identifier is
  package_declarative_part
  end [ package ] [ package_simple_name ];

package_declarative_item ::= [§ 2.5]
  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  

primary_unit_declaration ::= identifier ;  

procedure_call ::= procedure_name [ ( actual_parameter_part ) ]  

procedure_call_statement ::= [ label : ] procedure_call ;
process_declarative_item ::=                  [§ 9.2 ]
   [ § 9.2 ]
   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 ::=                  [§ 9.2 ]
   [ § 9.2 ]
   { process_declarative_item }

process_statement ::=                        [§ 9.2 ]
   [ § 9.2 ]
   [ postponed ] process [ ( sensitivity_list ) ] [ is ]
   process_declarative_part
   begin
   process_statement_part
   end [ postponed ] process [ process_label ] ;

process_statement_part ::=                   [§ 9.2 ]
   [ § 9.2 ]
   { sequential_statement }

protected_type_body ::=                      [§ 3.5.2]
   [ protected body ]
   protected_type_body_declarative_part
end protected body [ protected_type_simple_name ]

protected_type_body_declarative_item ::=     [§ 3.5.2]
   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

protected_type_body_declarative_part ::=     [§ 3.5.2]
   [ § 3.5.2 ]
   { protected_type_body_declarative_item }

protected_type_declaration ::=               [§ 3.5.1]
   [ protected ]
   protected_type_declarative_part
end protected [ protected_type_simple_name ]
protected_type_declarative_item ::=  
    subprogram_declaration  
    | attribute_specification  
    | use_clause  

protected_type_declarative_part ::=  
    { protected_type_declarative_item }  

protected_type_definition ::=  
    protected_type_declaration  
    | protected_type_body  

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

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

return_statement ::=  
    [ label : ] return [ 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 ::= [§ 9.5.2]
    { waveform when choices , }
    waveform when choices

sensitivity_clause ::= on sensitivity_list [§ 8.1]

sensitivity_list ::= signal_name { , signal_name } [§ 8.1]

sequence_of_statements ::= [§ 8]
    { sequential_statement }

sequential_statement ::= [§ 8]
    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

shift_expression ::= [§ 7.1]
    simple_expression [ shift_operator simple_expression ]

shift_operator ::= sll | srl | sla | sra | rol | ror [§ 7.2]

sign ::= + | – [§ 7.2]

signal_assignment_statement ::= [§ 8.4]
    [ label : ] target <= [ delay_mechanism ] waveform ;

signal_declaration ::= [§ 4.3.1.2]
    signal identifier_list : subtype_indication [ signal_kind ] [ := expression ] ;

signal_kind ::= register | bus [§ 4.3.1.2]

signal_list ::= [§ 5.3]
    signal_name { , signal_name }
    | others
    | all

signature ::= [ [ type_mark { , type_mark } ] [ return type_mark ] ] [§ 2.3.2]

simple_expression ::= [§ 7.1]
    [ sign ] term { adding_operator term }

simple_name ::= identifier [§ 6.2]

slice_name ::= prefix ( discrete_range ) [§ 6.5]

string_literal ::= " { graphic_character } " [§ 13.6]
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

[§ 8.1]

type_conversion ::= type_mark ( expression )

[§ 7.3.5]

type_declaration ::= full_type_declaration
| incomplete_type_declaration

[§ 4.1]

type_definition ::= scalar_type_definition
| composite_type_definition
| access_type_definition
| file_type_definition
| protected_type_definition

[§ 4.1]

type_mark ::= type_name
| subtype_name

[§ 4.2]

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

[§ 3.2.1]

use_clause ::= use selected_name { , selected_name } ;

[§ 10.4]

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

[§ 8.5]

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

[§ 4.3.1.3]

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

[§ 8.1]

waveform ::= waveform_element { , waveform_element }
| unaffected

[§ 8.4]

waveform_element ::= value_expression [ after time_expression ]
| null [ after time_expression ]

[§ 8.4.1]