Appendix

E

VHDL Language Grammar

This appendix contains the formal grammar of the standard ANSI/IEEE Standard 1076-1993 VHDL language in BNF format. In this format, productions are on the left-hand side of an equivalence. Two colons and an equals sign are used for equivalence, vertical bars for oring, square brackets for optional parts, and curly brackets for parts that zero or more of them may be used. As in the text, we have used upper-case letters for the language-reserved words. Language productions are ordered alphabetically.

```
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                                                        [$]
```

```
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 ::=                                              [$]
        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 ::=                                    [$]
        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 ::=                                              [$]
        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 ::=                                                          [$]
          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 }

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 : 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_constitiuent { , 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 ::=                                                                            [$]
        [ 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 } )                                    [$]

instantiation_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                                                [$]

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                                        [$]

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
                        primary_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 ::=                                              [$]
```

```
          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 ::=  =  |  /=  |  <  |  <=  |  >  |  >=           [$]

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 ::=                                                        [$]
      { 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

shift_expression ::=                                                          [$]
      simple_expression [ shift_operator simple_expression ]

shift_operator ::= SLL | SRL | SLA | SRA | ROL | ROR                          [$]

sign ::=  + | -                                                               [$]

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

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 ]
```