Reserved words are in boldface type. Square brackets enclose optional items. Curly brackets enclose items that are repeated zero or more times. A vertical bar (|) indicates or.

**Disclaimer:** This VHDL summary is not complete and contains some special cases. Only VHDL statements used in this text are listed. For a complete description of VHDL syntax, refer to references [7] and [17].

*signal assignment statement:* (sequential or concurrent statement)

\[ \text{signal} \leftarrow \text{[reject pulse-width | transport]} \text{expression [after delay_time]}; \]

*Note:* If concurrent, signal value is recomputed every time a change occurs on the right-hand side. If after time-spec is omitted, signal is updated after delta time.

*variable assignment statement:* (sequential statement only)

\[ \text{variable} \leftarrow \text{expression;} \]

*Note:* This can be used only within a process, function, or procedure. The variable is always updated immediately.

*conditional assignment statement:* (concurrent statement only)

\[
\begin{align*}
\text{signal} & \leftarrow \text{expression1 when condition1} \\
& \hspace{1cm} \text{else expression2 when condition2} \\
& \hspace{1cm} \ldots \\
& \hspace{1cm} \text{[else expression]};
\end{align*}
\]

*selected signal assignment statement:* (concurrent statement only)

\[
\begin{align*}
\text{with expression select} \\
\text{signal} & \leftarrow \text{expression1 [after delay_time] when choice1,} \\
& \hspace{1cm} \text{expression2 [after delay_time] when choice2,} \\
& \hspace{1cm} \ldots \\
& \hspace{1cm} \text{[expression [after delay_time] when others]};
\end{align*}
\]
entity declaration:

  entity entity-name is
    [generic (list-of-generics-and-their-types );]
    [port (interface-signal-declaration);]
    [declarations]
  end entity-name;

interface-signal declaration:

  list-of-interface-signals: mode type [ := initial-value ]
  { ; list-of-interface-signals: mode type [ := initial-value ]}

Note: An interface signal can be of mode in, out, inout, or buffer.

architecture declaration:

  architecture architecture-name of entity-name is
    [declarations] -- variable declarations not allowed
  begin
    architecture-body
  end architecture-name;

Note: The architecture body may contain component-instantiation statements, processes, blocks, assignment statements, procedure calls, etc.

integer type declaration:

  type type-name is range integer_range;

enumeration type declaration:

  type type-name is (list-of-names-or-characters);

subtype declaration:

  subtype subtype-name is type-name [index-or-range-constraint];

variable declaration:

  variable list-of-variable-names : type-name [ := initial_value ];

signal declaration:

  signal list-of-signal-names : type-name [ := initial_value ];

constant declaration:

  constant constant-name : type-name := constant_value;

alias declaration:

  alias identifier [ : identifier-type ] is item-name;

Note: Item-name can be a constant, signal, variable, file, function name, type name, etc.

array type and object declaration:

  type array_type_name is array index_range of element_type;
  signal | variable | constant array_name: array_type_name [ := initial_values ];
process statement (with sensitivity list):
  [process-label:] process (sensitivity-list)
  [declarations]           -- signal declarations not allowed
  begin
    sequential statements
  end process [process-label];

Note: This form of process is executed initially and thereafter only when an item on
the sensitivity list changes value. The sensitivity list is a list of signals. No wait statements are
allowed.

process statement (without sensitivity list):
  [process-label:] process
  [declarations]           -- signal declarations not allowed
  begin
    sequential statements
  end process [process-label];

Note: This form of process must contain one or more wait statements. It starts execution
immediately and continues until a wait statement is encountered.

wait statements can be of the form:
  wait on sensitivity-list;
  wait until boolean-expression;
  wait for time-expression;

if statement:                           (sequential statement only)
  if condition then
    sequential statements
  { elsif condition then
    sequential statements }           -- 0 or more elsif clauses may be included
  [else sequential statements]
  end if;

case statement:                        (sequential statement only)
  case expression is
    when choice1 => sequential statements
    when choice2 => sequential statements
    ...
    [when others => sequential statements]
  end case;

for loop statement:                    (sequential statement only)
  [loop-label:] for identifier in range loop
    sequential statements
  end loop [loop-label];

Note: You may use exit to exit the current loop.
while loop statement: (sequential statement only)
   [loop-label:] while boolean-expression loop
   sequential statements
   end loop [loop-label];

exit statement: (sequential statement only)
   exit [loop-label] [when condition];

assert statement: (sequential or concurrent statement)
   assert boolean-expression
       [report string-expression]
       [severity severity-level];

report statement: (sequential statement only)
   report string-expression
       [severity severity-level];

procedure declaration:
   procedure procedure-name (parameter list) is
      · [declarations]
   begin
      sequential statements
   end procedure-name;
Note: Parameters may be signals, variables, or constants.

procedure call:
   procedure-name (actual-parameter-list);
Note: An expression may be used for an actual parameter of mode in; types of the actual
parameters must match the types of the formal parameters; open cannot be used.

function declaration:
   function function-name (parameter-list) return return-type is
      · [declarations]
   begin
      sequential statements -- must include return return-value;
   end function-name;
Note: Parameters may be signals or constants.

function call:
   function-name (actual-parameter list)
Note: A function call is used within (or in place of) an expression.

library declaration:
   library list-of-library-names;
**use statement:**

```
use library_name.package_name.item;  \(.item may be .all)\n```

**package declaration:**

```
package package-name is
    package declarations
end [package][package-name];
```

**package body:**

```
package body package-name is
    package body declarations
end [package body][package name];
```

**component declaration:**

```
component component-name
    [generic (list-of-generics-and-their-types);]
    port (list-of-interface-signals-and-their-types);
end component;
```

**component instantiation:**

```
label: component-name
    [generic map (generic-association-list);]
    port map (list-of-actual-signals);
```

**Note:** Use open if a component output has no connection.

**generate statements:**

```
generate_label: for identifier in range generate
    [begin]
        concurrent statement(s)
    end generate [generate_label];
```

```
generate_label: if condition generate
    [begin]
        concurrent statement(s)
    end generate [generate_label];
```

**file type declaration:**

```
type file_name is file of type_name;
```

**file declaration:**

```
file file_name: file_type [open mode] is "file_pathname";
```

**Note:** Mode may be read_mode, write_mode, or append_mode.