15. Non-synthesizable Constructs
Most tools will not synthesize:
- access, after, alias, assert, bus, disconnect, file, guarded, initial, impure, label, linkage, new, on, open, postponed, pure, reject, report, severity, shared, transport, units, with.

16. Standard Packages
Samplings of a subset of standard packages the language provides.

16.1 IEEE.STD_LOGIC_1164 Package
- type std_logic is ('U', 'X', '0', '1', 'Z', 'W', 'l', 'H', '\'');
- type std_logic_vector is array (natural range <>) of std_logic;
- function resolved(s : std_logic_vector) return std_logic;
- function to_bit(s : std_logic; xmap : bit := '0') return bit;
- function to_bitvector(s : std_logic_vector; xmap : bit := '0') return bit_vector;
- function std_logic_vector to bitvector(b : bit_vector)
- function rising_edge(s : std_logic_vector) return boolean;
- function falling_edge(s : std_logic_vector) return boolean;

16.2 STD.TEXTIO Package
- type text is file of string;
- type line is (right, left);
- type width is natural;
- file input : text open read_mode is "std_input";
- file output : text open write_mode is "std_output";
- procedure readline(file f : text; l : line);
- procedure writeln(file f : text; l : in line);
- procedure read(l : in line);
- procedure write(l : out line);
- procedure justified in line := right;
- type character is integer;
- type string is string;
- there is no standard package for textual operations on std_logic. Tool vendors may provide their own.

16.3 IEEE.NUMERIC_STD Package
- type unsigned is array (natural range <>) of std_logic;
- type signed is array (natural range <>) of std_logic;
- function shift_left(arg : unsigned; count : natural) return unsigned;
- function rotate_left(arg : unsigned; count : natural) return unsigned;
- function resize(arg : signed; new_size : natural) return signed;


1. Introduction
VHDL is an case insensitive and strongly typed language. Comments start with two adjacent hyphens (--). end and end at the end of line.

2. Compiler Unit
Library Usage Declarations -- ref. 11
Entity Declarations -- ref. 3
Architecture Declarations -- ref. 4
Package Declarations -- ref. 10
Configuration Declarations -- ref. 14

3. Entity Declaration
- entity n_input : out
  generic (n : integer := 2);
  port (data : in bit_vector(1 to n);
  result : out bit);
- end n_input;
- -- port directions in I OUT I INPUT I BUFFER I LINKAGE

4. Architecture Declaration
architecture behave of n_input is
-- declarations -- ref. 7
begin
-- concurrent statements -- ref. 9
end behave;

5. Operators
- logical operators: and, or, xor, nand, nor, xnor, not
- relational operators: =, /=, <, <=, >, >=
- shift operators: shift_left, shift_right
- arithmetic operators: +, -, *, /, mod, abs, rem
- character operators: character, integer
- boolean operators: true, false
- natural operators: natural
- positive operators: positive
- real operators: real

6. Data Types
6.1 Predefined Data Types
- bit "0" and "1"
- bit_vector Array of "bit"
- boolean true and false
- character 7-bit ASCII
- integer signed 32 bit at least
- natural integer >= 0
- positive integer > 0
- real Floating point, min. +1e38 to -1e38
string Array of characters.
time hr, min, sec, ms, us, ns, ps, fs
file_open_kind read_mode, write_mod, append_mode
file_open_status open_ck, status_error, name_error, mode_error

6.2 User Defined Data Types
• type distance is range 0 to 100000 units
  meter; -- base unit
  kilometer = 1000 meter;
end units distance;
• type number is integer;
• type voltage is range 0 to 5;
• type current is range 1000 downto 0;
• type d_bus is array (range <= 1) of bit;
• type instruction is record
  opcode : bit;
  operand : bit;
end record;
• type int_file is file of integer;
• type pointer_to_integer is access integer;
• subtype positive_number is integer range 0 to 100000;
• type fourval is (X, Y, Z, W);
• subtype resolve_n is resolve twoval;

7. Declarations
• constant bus_width : integer := 32;
• variable read_flag : bit := '0'; -- only in processes
  and subprograms
• signal clock : bit;
• file file_open write_mode is "test.out";
• alias enable : bit is addr(31);
• attribute delay : time;
• attribute delay of my_signal : signal is 8 ns;
• component n_input_nand
  generic ( n : integer := 2 );
  port ( data : in bit_vector ( 1 to n );
  result : out bit );
end component n_input_nand;
• function square [1:integer] return integer;
• for store : use configuration latch;

8. Attributes
• type my_array is array ( 9 downto 0 ) of any_type;
• variable an_array : my_array;
• type fourval is ( '0', '1', '2', '3' );
• signal sig : sigtype;
• constant T : time := 10 ns;

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Result type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_array/high</td>
<td>any_type</td>
<td>9</td>
</tr>
<tr>
<td>my_array/low</td>
<td>any_type</td>
<td>0</td>
</tr>
<tr>
<td>my_array/first</td>
<td>any_type</td>
<td>0</td>
</tr>
<tr>
<td>my_array/high</td>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>an_array/ascending</td>
<td>integer</td>
<td>10</td>
</tr>
<tr>
<td>an_array/descent</td>
<td>integer</td>
<td>9 downto 0</td>
</tr>
<tr>
<td>an_array/reverse_range</td>
<td>integer</td>
<td>0 to 9</td>
</tr>
<tr>
<td>fourval/tof('0')</td>
<td>fourval</td>
<td>error</td>
</tr>
<tr>
<td>fourval/tof('1')</td>
<td>fourval</td>
<td>'0'</td>
</tr>
</tbody>
</table>

9. Statements
9.1 Concurrent Statements
• state_mach : process ( state ) -- label is optional
  begin
  -- variable declarations -- ref. 7
  end process;
• U1_n_input_nand : n_input_nand
  generic map ( n := 2 );
  port map ( data => my_data,
  result => my_res );
end process;
• top_block : block
  -- declaration -- ref. 7
  begin
    -- concurrent statements -- ref. 9
  end block;
• label1 : for i in 1 to 3 generate
  label2 : nand2 ( a(b), b(c), c(d) );
end generate;
• label3 : if ( x < y ) generate
  label4 : nor( a(i), b(c), c(d) );
end generate;

9.2 Sequential Statements
• null; -- does nothing
• wait on sig1, sig2 until ( sig = '1' ) for 30 ns;
• wait until ( clock'event and clock = '1' );
• read_flag := 0; -- read_flag is a variable
• if ( x < y ) then max := y;
  -- max is a variable
  elsif ( x > y ) then max := x;
  -- optional
  else max := x;
  -- optional
end if;
• case is
  when '1' | '0' => d <= '1';
  -- d is a signal
  when 'Z' => d <= '0';
  when others => d <= 'X';
end case;
• while ( x < y ) loop
  next when ( x = 5 ); -- usage of next
  x := x + 1; -- x is a variable
end loop;
• for i in ( 0 to 100 ) loop
  x := x + 1; -- x is a variable
end loop;

10. Package Declarations
• package two_level is
  -- type, signal, function declarations -- ref. 7
  end two_level;
• package body two_level is
  -- subprogram definitions -- ref. 12
  end two_level;

11. Library Usage Declarations
• using the two_level package;
• library work;
• use work.two_level all; -- all objects used
• use work.two_level.vcc; -- only the "vcc" object used

12. Subprograms
function bool_2_2level ( boolean : in boolean )
return two_level is
  variable return_val : two_level;
begin
  if ( in boolean = true ) then
    return_val := high;
  else
    return_val := low;
  end if;
end bool_2_2level;

procedure clock_buffer
  ( signal local_clk : in out bit; signal clk_simulation : in out bit; constant clock_skew : in time )
begin
  example of side effects in a procedure
  global clk <= local clk after clock_skew;
  local clk <= clk_simulation;
end clock_buffer;

13. Predefined Subprograms
• enable <= '1' when ( now < 2 ns) else '0';
• variable pio : pointer_to_integer;
  pio := new integer;
  -- usage of new
  deallocate( pio );
• variable status : file_open_status;
  file_open( status, my_file, "int.dat", read_mode );
• endfile( my_file ); -- returns true/false
• variable int_var : integer;
  read( my_file, int_var );
• file_close( my_file );

14. Configuration Declarations
configuration input_8 of n_command is
  for customizable