VHDL Syntax Reference

By Prof. Taek M. Kwon
EE Dept, University of Minnesota Duluth

This summary is provided as a quick lookup resource for VHDL syntax and code examples. Please click on the topic you are looking for to jump to the corresponding page.

Contents

1. Bits, Vectors, Signals, Operators, Types

1.1 Bits and Vectors in Port

Bits and vectors declared in port with direction.

Example:

```
    port ( a : in std_logic;   -- signal comes in to port a from outside
           b : out std_logic;  -- signal is sent out to the port b
           c : inout std_logic; -- bidirectional port
           x : in std_logic_vector(7 downto 0);  -- 8-bit input vector
           y : out std_logic_vector(7 downto 0)   -- no ';' for the last item
          );
```

1.2 Signals

Signals are declared without direction.

Example:

```
    signal s1, s2 : std_logic;
    signal X, Y : std_logic_vector(31 downto 0);
```

1.3 Constants

Constants are useful for representing commonly-used values of specific types.

Example:

```
   In the declaration area:
    constant init : std_logic_vector(3 downto 0) := "1100";
    signal sig_vec : std_logic_vector(3 downto 0);
  In the body:
    sig_vec <= init;
```

1.4 Relational Operators

Return a Boolean result and thus used in **if** or **when** clauses.

=	equal to:	*highest precedence*
/=	not equal to	
<	less than	
<=	less than equal	
>	greater than	
>=	greater than equal:	*lowest precedence*

1.5 Logical Operators

Bit-by-bit logical operations.

not	example) (**not** a)	*highest precedence*
and		
or		
nand		
nor		
xor		
xnor		*lowest precedence*

1.6 Assignments

<=	signal assignment
:=	variable assignment, signal initialization

Example:

signal q: **std_logic_vector**(3 **downto** 0);

Multiple bits are enclosed using a pair of double quotations:

```
q  <=  “1011”;
```

Hexadecimals are represented using X”….”:

```
q  <=  X”B”;
```

A single bit is enclosed using single quotations:

```
q <=  (‘1’, ’0’, ’1’, ’1’);
```

You may use named association:

```
q <=  (3=>’1’, 2=>’0’, 1=>’1’, 0=>’1’);
```

Named association allows position independence, i.e., you can write

```
q <=  (0=>’1’, 2=>’0’, 1=>’1’, 3=>’1’);
```

You may combine indices.

```
q <= (3|1|0 => ‘1’, 2 => ‘0’);
```

Use the keyword ‘others’ to simplify the expression.

```
q <= (2=>’0’, others => ‘1’);
```

We frequently use **others** for initialization or setting bits.

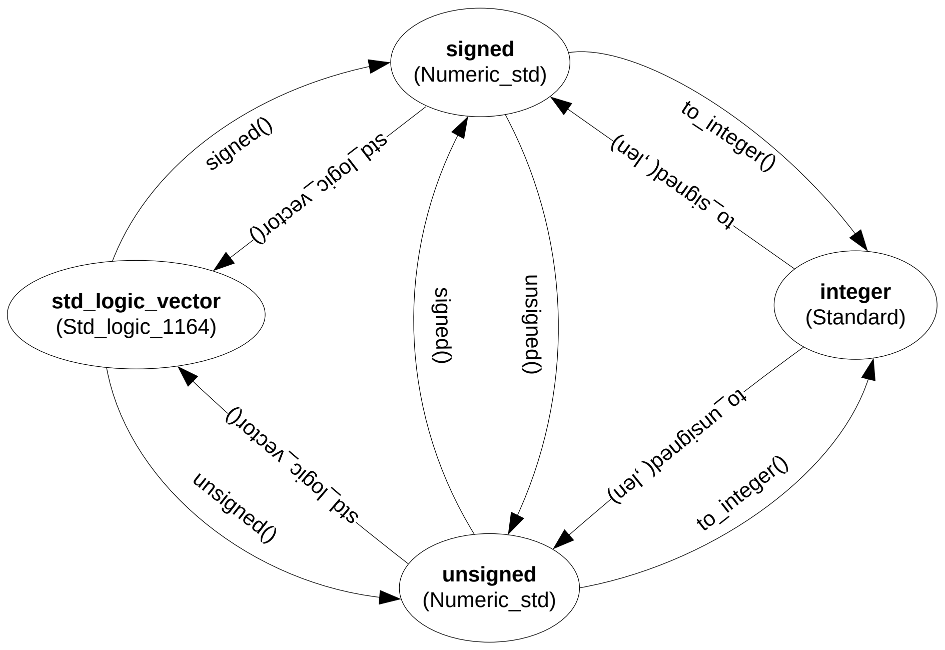
```
x <= “00000000”;      -- is same as
x <= (others => ‘0’);
```

1.7 Concatenation, &

Example: **signal** a, b **: std_logic_vector**(7 **downto** 0) := “10111111”;

b **<=** a(7 **downto** 2) **&** “00”; -- b contains “10111100”

1.8 Type Conversion Chart



2. Concurrent Statements

Any statement placed in architecture body is concurrent. Only one type of conditional statements is allowed as concurrent which are shown here.

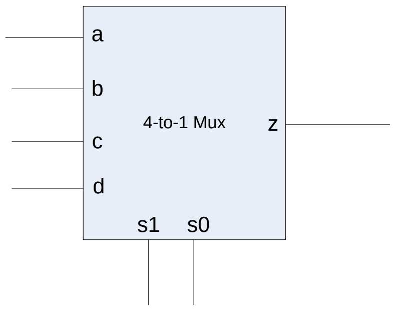
2.1 Conditional Signal Assignment

Syntax:

```
signal_name <= value_expr_1 when Boolean_expr_1 else
               value_expr_2 when Boolean_expr_2 else
               value_expr_3 when Boolean_expr_3 else
               ….
               value_expr_n;
```

Example: 4-to-1 Mux

```
z <=  a when (s=”00”) else
      b when (s=”01”) else
      c when (s=”10”) else
      d when (s=”11”) else
      ‘X’;
```



A better way would be:

```
z <=  a when (s=”00”) else
      b when (s=”01”) else
      c when (s=”10”) else
      d;
```

2.2 Selected Signal Assignment

Syntax:

```
        with select_expression select
            signal_name <= value_expr_1 when choice_1,
                           value_expr_2 when choice_2,
                           ….
                           value_expr_n when choice_n;
```

Example: 4-to-1 Mux

```
        with s select
            z <= a when “00”,
                  b when “01”,
                  c when “10”,
                  d when others;
```

3. Sequential Statements

3.1 Variables

Variables are objects used to store intermediate values between sequential VHDL statements. Variables are only allowed in processes, procedures and functions, and they are always local to those functions. When a value is assigned to a variable, “:=” is used.

Example:

```
        signal Grant, Select: std_logic;

        process(Rst, Clk)
            variable Q1, Q2, Q3: std_logic;
        begin
             if Rst=’1’ then
                 Q1 := ‘0’;   Q2 := ‘0’;  Q3 := ‘0’;
             elsif (Clk=’1’ and Clk’event) then
                 Q1 := Grant;
                 Q2 := Select;
                 Q3 := Q1 or Q2;
             end if;
          end process;
```

Note: Both signals and variables carry data from place to place. However, you must always use signals to carry information between concurrent elements of your design.

3.2 If-then-else Statement

Syntax:

```
        if Boolean_expr_1 then
                sequential_statements;
        elsif Boolean_expr_2 then
                sequential_statements;
        elsif Boolean_expr_3 then
                ...
        else
                sequential statements;
        end if;
```

Example:

```
        process ( a, b, m, n)
        begin
                if m  =  n then
                        r <= a + b;
                elsif   m > 0 then
                        r <= a – b;
                else
                        r <= a + 1;
                 end if;
        end;
```

3.3 Case Statement

Syntax:

```
        case sel is
                when choice_1 =>
                        sequential_statements;
                when choice_2 =>
                        sequential_statements;
                .  .  .
                when others =>
                        sequential_statements;
        end case;
```

Example:

```
case sel is
        when "00" =>
                r <= a + b;
        when "10"
                r <= a – b;
        when others =>
                r <= a + 1;
end case;
```

3.4 For Loop

Syntax:

```
for index in loop_range loop
        sequential statements;
end loop;
```

Example:

```
constant MAX: integer := 8;
signal a, b, y: std_logic_vector(MAX-1 downto 0);
…
…
for i in (MAX-1) downto 0 loop
        y(i) <= a(i) xor b(i)
end loop;
```

3.5 While Loop

Syntax:

```
loop_name: while (condition) loop
    ---repeated statements
end loop loop_name;
```

Example:

```
while error_flag /= ‘1’ and done /=’1’ loop
        Clock <= not Clock;
        wait for CLK_PERIOD/2;
end loop;
```

3.6 Infinite Loop

Syntax:

```
loop_name: loop
        …
        exit when (condition)
end loop loop_name;
```

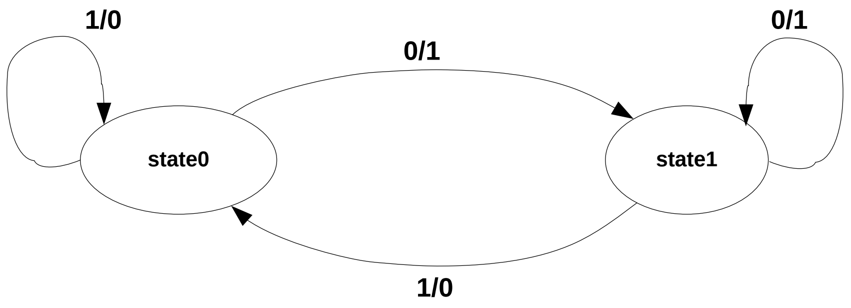
Example:

```
loop
    Clock <= not Clock;
    wait for CLK_PERIOD/2;
    if done = ‘1’ or error_flag = ‘1’ then
        exit;
    end if;
end loop;
```

3.7 Wait Statements

```
wait on signals;
wait until Boolean_expr;
wait for time_expr;
```

3.8 Finite State Machine (FSM) Implementation



Finite state machines in VHDL can be implemented by following a typical programming structure such as given below. It consists of two processes: one for combinational logic process that sets the next state and output, and a clock handling process that loads the next state to present state. This implementation is a Mealy machine.

```
Entity state_machine is
  Port( reset, clk, x: in std_logic;
                    Z: out std_logic);
End state_machine;
```

--Architecture portion of the code is shown in the next page.

```
Architecture bhv of state_machine is

        Type statetype is (state0, state1);  -- define states
        Signal Pstate, Next_state: statetype;
Begin

Logic_proc: process(pstate, x)
  Begin
        Case pstate is
            When state0 =>
                  If x='0' then
                        Next_state <= state1;
                        Z <= '1';
                  Else
                        Next_state <= state0;
                        Z <= '0';
                  End if;
            When state1 =>
                  If x='1' then
                        Next_state <= stateo0;
                        Z <= '0';
                  Else
                        Next_state <= state1;
                        Z <= '1';
                  End if;
        End case;
End process Logic_proc;


Clock_proc: process
Begin
        Wait until (clk'event and clk ='1');
        If reset = '1' then
            Pstate <= statetype'left;
        Else
            Pstate <= next_state;
        End if;
End process Clock_proc;


End bhv;
```