VHDL Coding Guidelines
for
High Performance

QUICK REFERENCE GUIDE
Version 1.0

If Statement
Priority encoded

Original Design

```
entity mult_if is
     port(
       a,
       b_is_late_arriving,
       c,
       d,
       e    : in  std_logic;
       sel  : in  std_logic_vector(3 downto 0);
       z    : out std_logic
     );
end;

architecture orig of mult_if is
begin
     process(a, b_is_late_arriving, c, d, e, sel)
     begin
       z <= e;
       if sel(0) = '1' then
         z <= a;
       end if;
       if sel(1) = '1' then
         z <= b_is_late_arriving;
       end if;
       if sel(2) = '1' then
         z <= c;
       end if;
       if sel(3) = '1' then
         z <= d;
       end if;
     end process;
end;
```

Improved Design

```
entity mult_if is
    port(
      a,
      b_is_late_arriving,
      c,
      d,
      e    : in  std_logic;
      sel  : in  st _logic_vector(3 downto 0);
      z    : out std_logic
    );
end;

architecture improved of mult_if is
begin
    process(a, b_is_late_arriving, c, d, e, sel)
      variable higher_priority  : boolean;
      variable Z1               : std_logic;
    begin
      Z1 := e;
      if sel(0) = '1' then
        Z1 := a;
      end if;
      if sel(2) = '1' then
        Z1 := c;
      end if;
      if sel(3) = '1' then
        Z1 := d;
      end if;

      higher_priority := sel(2) = '1' or sel(3) = '1';

      if  (not higher_priority and sel(1) = '1') then
        Z <= b_is_late_arriving;
      else
        Z <= Z1;
      end if;
    end process;
end;
```

If Statement
Parallel encoded

Original Design

```
entity single_if is
    port(
      A    : in  std_logic_vector(5 downto 0);
      sel  : in  std_logic_vector(4 downto 0);
      CTRL_is_late_arriving
           : in  std_logic;
      Z    : out std_logic
    );
end;

architecture orig of single_if is
begin
    process(A, sel, CTRL_is_late_arriving)
    begin
      if sel(0) = '1' then
        Z <= A(0);
      elsif sel(1) = '0' then
        Z <= A(1);
      elsif sel(2) = '1' then
        Z <= A(2);
      elsif sel(3) = '1' and
          CTRL_is_late_arriving = '0' then
        Z <= A(3);
      elsif sel(4) = '0' then
        Z <= A(4);
      else
        Z <= A(5);
      end if;
    end process;
end;
```

Improved Design

```
entity single_if is
    port(
      A    : in  std_logic_vector(5 downto 0);
      sel  : in  std_logic_vector(4 downto 0);
      CTRL_is_late_arriving
           : in  std_logic;
      Z    : out std_logic
    );
end;

architecture improved of single_if is
begin
    process(A, sel, CTRL_is_late_arriving)
      variable Z1        : std_logic;
      variable prev_cond : boolean;
    begin
      if sel(0) = '1' then
        Z1 := A(0);
      elsif sel(1) = '0' then
        Z1 := A(1);
      elsif sel(2) = '1' then
        Z1 := A(2);
      elsif sel(4) = '0' then
        Z1 := A(4);
      else
        Z1 := A(5);
      end if;

      prev_cond := sel(0) = '1' or
             sel(1) = '0' or
             sel(2) = '1';

      if sel(3) = '1' and
                   CTRL_is_late_arriving = '0' then
        if prev_cond then
          Z <= Z1;
        else
          Z <= A(3);
        end if;
      else
        Z <= Z1;
      end if;
    end process;
end;
```

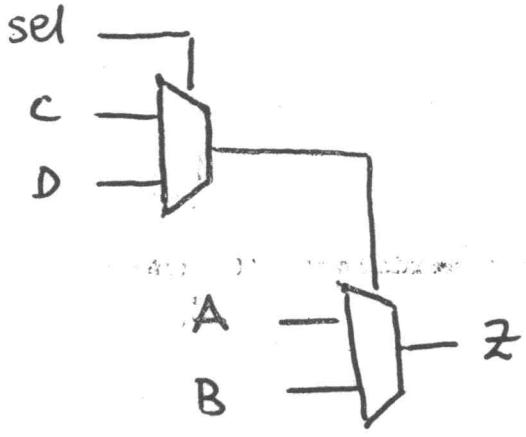
If Statement
Cascaded Select Logic

Original Design

```
entity ctrl_cascade is
    port(
      sel_is_late_arriving
                  : in  std_logic;
      A, B, C, D : in  std_logic;
      Z           : out std_logic
    );
end;

architecture orig of ctrl_cascade is
begin
    process(sel_is_late_arriving, A, B, C, D)
      variable i_sel  : std_logic;
    begin
      if sel_is_late_arriving = '1' then
        i_sel := C;
      else
        i_sel := D;
      end if;

      if i_sel = '1' then
        Z <= A;
      else
        Z <= B;
      end if;
    end process;
end;
```



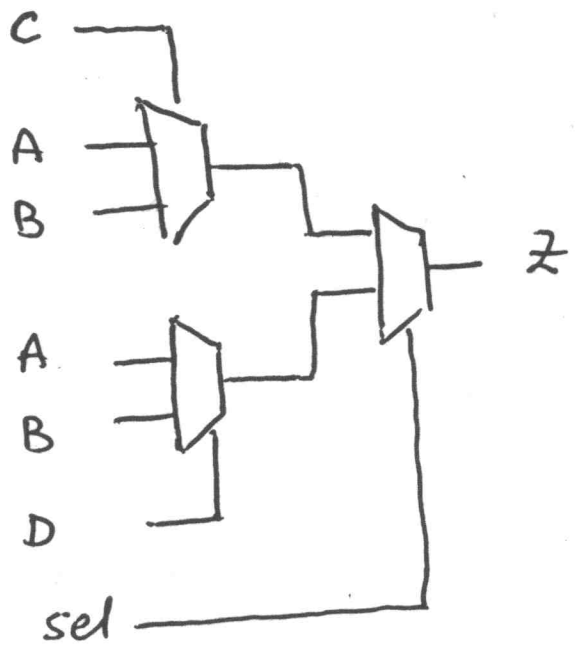
Improved Design

```
entity ctrl_cascad is
    port(
      sel_is_late_arriving
                  : in  std_logic;
      A, B, C, D : in  std_logic;
      Z           : out std_logic
    );
end;

architecture improved of ctrl_cascade is
begin
    process(sel_is_late_arriving, A, B, C, D)
      variable C_Z, D_Z  : std_logic;
    begin
      if C = '1' then
        C_Z := A;
      else
        C_Z := B;
      end if;

      if D = '1' then
        D_Z := A;
      else
        D_Z := B;
      end if;

      if sel_is_late_arriving = '1' then
        Z <= C_Z;
      else
        Z <= D_Z;
      end if;
    end process;
end;
```

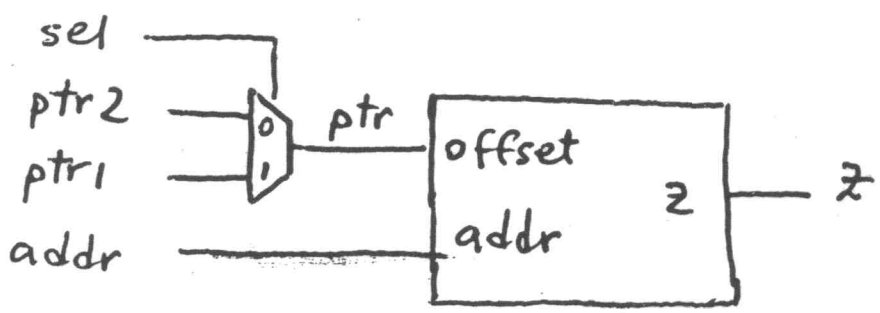


Original Design

```
entity reorder is
     generic(
       N : integer := 8
     );
     port(
       sel_is_late_arriving
            : in  std_logic;
       ptr1,
       ptr2 : in std_logic_vector(N-1 downto 0);
       addr : in std_logic_vector(N-1 downto 0);
       Z    : out std_logic_vector(N-1 downto 0)
     );
end;

architecture orig of reorder is
     component comp
          port(
            Z      : out std_logic_vector(N-1 downto 0);
            addr   : in  std_logic_vector(N-1 downto 0);
            offset : in  std_logic_vector(N-1 downto 0)
          );
     end component;
begin
     process(addr, ptr1, ptr2, sel_is_late_arriving)
       variable ptr: std_logic_vector(N-1 downto 0);
     begin
       if sel_is_late_arriving = '1' then
         ptr := ptr1;
       else
         ptr := ptr2;
       end if;
     end process;

     U1: comp port map (Z, addr, ptr);
end;
```

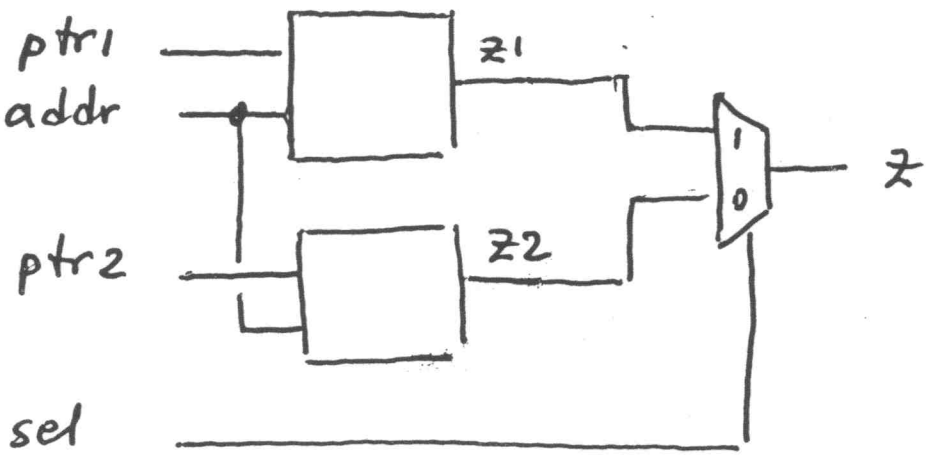


Improved Design

```
entity reorder is
    generic(
      N : integer := 8
    );
    port(
      sel_is_late_arriving
          : in  std_logic;
      ptr1,
      ptr2 : in  std_logic_vector(N-1 downto 0);
      addr : in  std_logic_vector(N-1 downto 0);
      Z    : out std_logic_vector(N-1 downto 0)
    );
end;

architecture improved of reorder is
    component comp
        port(
          Z      : out std_logic_vector(N-1 downto 0);
          addr   : in  std_logic_vector(N-1 downto 0);
          offset : in  std_logic_vector(N-1 downto 0)
        );
    end component;
    signal Z1, Z2 : std_logic_vector(N-1 downto 0);
begin
    U1   : comp port map (Z1, addr, ptr1);
    U2   : comp port map (Z2, addr, ptr2);

    process(addr, ptr1, ptr2, sel_is_late_arriving)
    begin
      if sel_is_late_arriving = '1' then
        Z <= Z1;
      else
        Z <= Z2;
      end if;
    end process;
end;
```



Case Statement
Embedded If

Original Design

```
entity case_with_if is
    port(
      C_is_late_arriving,
      D_is_late_arriving
            : in  std_logic;
      i_sel : in  std_logic;
      A, B  : in  std_logic;
      sel   : in  std_logic_vector(2 downto 0);
      Z     : out std_logic
    );
end;

architecture orig of case_with_if is
begin
    process(sel, i_sel, A, B, C_is_late_arriving,
            D_is_late_arriving)
    begin
      case sel is
        when "000"  =>  Z <= A;
        when "001"  =>  Z <= B;
        when "010"  =>
          if i_sel = '1' then
            Z <= C_is_late_arriving;
          else
            Z <= D_is_late_arriving;
          end if;
        when "100"  =>  Z <= A XOR B;
        when "101"  =>  Z <= A NAND B;
        when "111"  =>  Z <= NOT A;
        when others =>  Z <= NOT B;
      end case;
    end process;
end;
```

Improved Design

```
entity case_with_if is
    port(
      C_is_late_arriving,
      D_is_late_arriving
            : in  std_logic;
      i_sel : in  std_logic;
      A, B  : in  std_logic;
      sel   : in  std_logic_vector(2 downto 0);
      Z     : out std_logic
    );
end;

architecture improved of case_with_if is
begin
    process(sel, i_sel, A, B, C_is_late_arriving,
            D_is_late_arriving)
      variable Z1: std_logic;
    begin
      case sel is
        when "000"  =>  Z1 := A;
        when "001"  =>  Z1 := B;
        when "100"  =>  Z1 := A XOR B;
        when "101"  =>  Z1 := A NAND B;
        when "111"  =>  Z1 := NOT A;
        when others =>  Z1 := NOT B;
      end case;

      if sel = "010" then
        if i_sel = '1' then
          Z <= C_is_late_arriving;
        else
          Z <= D_is_late_arriving;
        end if;
      else
          Z <= Z1;
      end if;
    end process;
end;
```

Case Statement
Late-arriving Selector

Original Design

```
entity split_case is
     port(
       sel0                    : in  std_logic;
       sel1_is_late_arriving   : in  std_logic;
       sel2                    : in  std_logic;
       X                       : in  std_logic;
       A, B, C, D              : in  std_logic;
       Z                       : out std_logic
     );
end;

architecture orig of split_case is
     signal sel: std_logic_vector(2 downto 0);
begin
     sel <= sel2 & sel1_is_late_arriving & sel0;
     process(sel, X, A, B, C, D)
     begin
       case sel is
         when "000"  =>  Z <= A;
         when "001"  =>  Z <= B;
         when "010"  =>
                    if X = '1' then
                      Z <= C;
                    else
                      Z <= D;
                    end if;
         when "100"  =>  Z <= A XOR B;
         when "101"  =>  Z <= A NAND B;
         when "111"  =>  Z <= NOT A;
         when others =>  Z <= NOT B;
       end case;
     end process;
end;
```

Improved Design

```
entity split_case is
    port(
      sel0                    : in  std_logic;
      sel1_is_late_arriving   : in  std_logic;
      sel2                    : in  std_logic;
      X                       : in  std_logic;
      A, B, C, D              : in  std_logic;
      Z                       : out std_logic
    );
end;

architecture improved of split_case is
    signal sel  : std_logic_vector(1 downto 0);
begin
    sel <= sel2 & sel0;
    process(sel, sel1_is_late_arriving, X, A, B, C, D)
      variable Z1, Z2 : std_logic;
    begin
      case sel is  -- sel1_is_late_arriving = '0'
        when "00"   =>  Z1 := A;
        when "01"   =>  Z1 := B;
        when "10"   =>  Z1 := A XOR B;
        when "11"   =>  Z1 := A NAND B;
        when others =>  Z1 := NOT B;
      end case;

      case sel is  -- sel1_is_late_arriving = '1'
        when "00"   => if X = '1' then
                         Z2 := C;
                       else
                         Z2 := D;
                       end if;
        when "11"   => Z2 := NOT A;
        when others => Z2 := NOT B;
      end case;

      if sel1_is_late_arriving = '1' then
        Z <= Z2;
      else
        Z <= Z1;
      end if;
    end process;
end;
```

Original Design

```
entity reorder is
    generic(
      N     : integer := 8
    );
    port (
        A_is_late_arriving,
        B         : in  integer range -256 to 255;
        C, D      : in  std_logic_vector(N-1 downto 0);
        Z         : out std_logic_vector(N-1 downto 0)
    );
end;

architecture original of reorder is
begin
    process(A_is_late_arriving, B, C, D)
    begin
      if (A_is_late_arriving + B < 24) then
        Z <= C;
      else
        Z <= D;
      end if;
   end process;
end;
```

Improved Design

```
entity reorder is
    generic(
      N     : integer := 8
    );
    port (
        A_is_late_arriving,
        B          : in  integer range -256 to 255;
        C, D       : in  std_logic_vector(N-1 downto 0);
        Z          : out std_logic_vector(N-1 downto 0)
    );
end;

architecture improved of reorder is
begin
    process(A_is_late_arriving, B, C, D)
    begin
      if (A_is_late_arriving < 24 - B) then
        Z <= C;
      else
        Z <= D;
      end if;
    end process;
end;
```

Carry-in Speedup

Original Design

```
entity carry_in is
    generic(
      N           : integer := 8
    );
    port(
      A, B        : in  std_logic_vector(N-1 downto 0);
      Cin_is_late : in  std_logic;
      Z           : out std_logic_vector(N-1 downto 0)
    );
end;

architecture original of carry_in is
begin
    process(A, B, Cin_is_late)
    begin
      Z <= A + B + Cin_is_late;
    end process;
end;
```

Improved Design

```
entity carry_in is
    generic(
      N           : integer := 8
    );
    port(
      A, B        : in  std_logic_vector(N-1 downto 0);
      Cin_is_late : in  std_logic;
      Z           : out std_logic_vector(N-1 downto 0)
    );
end;

architecture improved of carry_in is
begin
    process(A, B, Cin_is_late)
    begin
      if Cin_is_late = '1' then
        Z <= A + B + 1;
      else
        Z <= A + B;
      end if;
    end process;
end;
```

Eliminating Common Subexpressions
Reducing Area

Original Design

```
entity expressions is
    generic(
      N           : integer := 8
    );
    port(
      A, B        : in  std_logic_vector(N-1 downto 0);
      Q, R, S     : in  std_logic_vector(N-1 downto 0);
      C1, C2, C3  : in  std_logic_vector(N-1 downto 0);
      Z           : out std_logic_vector(N-1 downto 0)
    );
end;

architecture original of expressions is
begin
    process(A, B, Q, R, S, C1, C2, C3)
    begin
      if (C1 > A + B + Q) then
        Z <= R - S;
      end if;
      if (C2 > A + B + R) then
        Z <= S - Q;
      end if;
      if (C3 > A + B + S) then
        Z <= Q - R;
      end if;
    end process;
end;
```

Improved Design

```
entity expressions is
    generic(
      N           : integer := 8
    );
    port(
      A, B        : in  std_logic_vector(N-1 downto 0);
      Q, R, S     : in  std_logic_vector(N-1 downto 0);
      C1, C2, C3  : in  std_logic_vector(N-1 downto 0);
      Z           : out std_logic_vector(N-1 downto 0)
    );
end;

architecture improved of expressions is
begin
    process(A, B, Q, R, S, C1, C2, C3)
      variable temp : std_logic_vector(N-1 downto 0);
    begin
      temp := A + B;
      if (C1 > temp + Q) then
        Z <= R - S;
      end if;
      if (C2 > temp + R) then
        Z <= S - Q;
      end if;
      if (C3 > temp + S) then
        Z <= Q - R;
      end if;
    end process;
end;
```

Linear Structures
Reduce OR function

Original Design

```
function OR_reduce_chain(data: std_logic_vector)
                                    return std_logic is
    variable result: std_logic;
begin
    result := '0';
    for I in data'RANGE loop
      result := result OR data(I);
    end loop;
    return result;
end;
```

Improved Design

```
function OR_reduce_tree(val: std_logic_vector)
                                    return std_logic is
    variable   UPPER_TREE,
               LOWER_TREE    : std_logic;
    variable   MID,
               LEN           . integer;
    variable   L_BOUND,
               R_BOUND       : integer;
    variable result          : std_logic;
    variable   i_val  :
        std_logic_vector(val'LENGTH-1 downto 0);
begin
    i_val := val;
    LEN := i_val'LENGTH;
    L_BOUND := i_val'LEFT;
    R_BOUND := i_val'RIGHT;

    if LEN = 1 then
      result := i_val(L_BOUND);
    elsif LEN = 2 then
      result :=  i_val(L_BOUND) OR i_val(R_BOUND);
    else
      MID := (LEN + 1)/2 + R_BOUND;
      UPPER_TREE :=
        OR_reduce_tree(i_val(L_BOUND downto MID));
      LOWER_TREE :=
        OR_reduce_tree(i_val(MID-1 downto R_BOUND));
      result := UPPER_TREE OR LOWER_TREE;
    end if;
    return result;
end;
```

Original Design

```
function mux_chain(sel, data: std_logic_vector)
                                  return std_logic is
    variable i_sel :
      std_logic_vector(sel'LENGTH-1 downto 0);
    variable i_data:
      std_logic_vector(data'LENGTH-1 downto 0);
    variable result       : std_logic;
begin
    i_sel    := sel;
    i_data   := data;

    result := i_data(i_data'LEFT);
    for I in i_sel'LENGTH - 1 downto 0 loop
      if i_sel(I) = '1' then
        result := i_data(I);
      end if;
    end loop;
    return result;
end;
```

Improved Design

```
function mux_tree(sel, data: std_logic_vector)
                                  return std_logic is
    variable resul        : std_logic;
    variable upper_tree,
             lower_tree   : std_logic;
    variable i_sel :
      std_logic_vector(sel'LENGTH-1 downto 0);
    variable i_data:
      std_logic_vector(data'LENGTH-1 downto 0);
    variable final_sel: std_logic;
    variable val: std_logic_vector(1 downto 0);
    variable  SEL_LEN,
              DATA_LEN,
              MID : integer;
begin
    i_sel    := sel;
    i_data   := data;
    DATA_LEN := i_data'LENGTH;
    SEL_LEN  := i_sel'LENGTH;

    if SEL_LEN = 0 or DATA_LEN = 0 then
    elsif SEL_LEN = 1 then
      result := mux_2_1(i_sel(0), i_data);
    elsif SEL_LEN = 2 then
      upper_tree := mux_2_1(i_sel(1),
                            i_data(2 downto 1));
      val := (upper_tree, i_data(0));
      result := mux_2_1(i_sel(0), val);
    elsif SEL_LEN = 3 and DATA_LEN = 3 then
      upper_tree := mux_2_1(i_sel(2),
                            i_data(2 downto 1));
      val := (upper_tree, i_data(0));
      final_sel := i_sel(1) OR i_sel(0);
      result := mux_2_1(final_sel, val);
    elsif SEL_LEN = 3 and DATA_LEN = 4 then
      upper_tree := mux_2_1(i_sel(2),
                            i_data(3 downto 2));
      lower_tree := mux_2_1(i_sel(0),
                            i_data(1 downto 0));
      val := (upper_tree, lower_tree);
      final_sel := i_sel(1) OR i_sel(0);
      result := mux_2_1(final_sel, val);
    else
      MID := (DATA_LEN + 1)/2 + i_data'RIGHT;
      upper_tree :=
        mux_tree(i_sel(i_sel'LEFT downto MID),
                 i_data(i_data'LEFT downto MID));
      lower_tree :=
        mux_tree(i_sel(MID - 2 downto i_sel'RIGHT),
                 i_data(MID - 1 downto i_data'RIGHT));
      val := (upper_tree, lower_tree);
      final_sel :=
              OR_reduce_tree(i_sel(MID - 1 downto 0));
      result := mux_2_1(final_sel, val);
    end if;
    return result;
end;
```