VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity mytt is
port( A, B:      in std_logic_vector(5 downto 0);
      Y:  out std_logic_vector(11 downto 0));
end mytt;

architecture arch_tt of mytt is
constant C: std_logic_vector(2 downto 0) := "100";
begin
 process(A,B)
 begin
   Y(2 downto 0) <= A(2 downto 0);
   Y(3)          <= A(3) and B(3);
   Y(5 downto 4) <= (A(5)and B(5))&(A(4) or B(4));
   Y(8 downto 6) <= B(2 downto 0);
   Y(11 downto 9)<= C;
 end process;
end arch_tt;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity tt is
port( A, B:      in std_logic;
      C:         in std_logic_vector(1 downto 0);
      Y1,Y2:     out std_logic_vector(3 downto 0);
      Y3A, Y3B, Y3C, Y3D: out std_logic);
end tt;

architecture arch_tt of tt is
begin
  -- positional association aggregate
  Y1 <= ('0','1', A nor B, A nand B);

 -- named association aggregate
  Y2 <= (2=>A or B, 3=>A and B, 0=>'0', 1=>'1');

 -- aggregate target
  (Y3A, Y3B, Y3C, Y3D) <= not C;

end arch_tt;
```

```
--- Concatenation example:
```

```
library ieee;
use ieee.std_logic_1164.all;
entity mytt2 is
port( A, B:     in std_logic_vector(2 downto 0);
     Y:  out std_logic_vector(14 downto 0));
end mytt2;

architecture arch_tt of mytt2 is
constant C: std_logic_vector(2 downto 0) := "100";
begin
 Y <= A & B & C & C & "110";
end arch_tt;
```

Flip-flop with Positive-Edge Clock

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(
        CLK, D : in std_logic;
        Q : out std_logic
        );
end flop;
architecture archi of flop is
  begin
    process (CLK)
    begin
     if(CLK'event and CLK='1') then
     --if (rising_edge(CLK)) then
         Q <= D;
     end if;
    end process;
end archi;
```

Flip-flop with Negative-Edge Clock and Asynchronous Clear

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(
        C, D, CLR: in std_logic;
        Q : out std_logic
        );
end flop;
architecture archi of flop is
  begin
    process (C, CLR)
    begin
      if (CLR = '1')then
            Q <= '0';
      elsif (C'event and C='0')then
      --if (falling_edge(CLK)) then

            Q <= D;
      end if;
    end process;
end archi;
```

Flip-flop with Positive-Edge Clock and Synchronous Set

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(
        C, D, S : in std_logic;
        Q : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
            if (S='1') then
                Q <= '1';
            else
            Q <= D;
            end if;
        end if;
    end process;
end archi;
```

4-bit Register with Positive-Edge Clock, Asynchronous Set and Clock Enable

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(
  C, CE, PRE : in std_logic;
  D: in std_logic_vector(3 downto 0);
  Q : out std_logic_vector (3 downto 0)
        );
end flop;
architecture archi of flop is
  begin
    process (C, PRE)
```

```
      begin
        if (PRE='1') then
            Q <= "1111";
        elsif (C'event and C='1')then
            if (CE='1') then
                Q <= D;
            end if;
        end if;
    end process;
end archi;
```

Latches

```
library ieee;
use ieee.std_logic_1164.all;
entity latch is
  port(
        G, D : in std_logic;
        Q : out std_logic
        );
end latch;
architecture archi of latch is
  begin
    process (G, D)
      begin
        if (G='1') then
            Q <= D;
        end if;
    end process;
end archi;
```

Tristates

```
library ieee;
use ieee.std_logic_1164.all;
entity three_st is
  port(
        T : in std_logic;
        I : in std_logic;
        O : out std_logic
        );
end three_st;
architecture archi of three_st is
  begin
    process (I, T)
      begin
        if (T='0') then
            O <= I;
        else
            O <= 'Z';
        end if;
    end process;
end archi;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity three_st is
  port(
        T : in std_logic;
        I : in std_logic;
        O : out std_logic         );
end three_st;
architecture archi of three_st is
  begin
    O <= I when (T='0') else 'Z';
end archi;
```

Counters .4-bit up/down counter with an asynchronous clear

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity counter is
  port(
   C, CLR, UP_DOWN : in std_logic;
   Q: out std_logic_vector(3 downto 0)
);
end counter;
architecture archi of counter is
signal tmp: std_logic_vector(3 downto 0);
begin
    process (C, CLR)
      begin
        if (CLR='1') then
            tmp <= "0000";
        elsif (C'event and C='1') then
            if (UP_DOWN='1') then
                tmp <= tmp + 1;
            else
                tmp <= tmp - 1;
            end if;
        end if;
    end process;
    Q <= tmp;
end archi;
```

8-bit Shift-Left Register with Positive-Edge Clock, Serial In, and Serial Out

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity shift is
  port(
        C, SI : in std_logic;
        SO : out std_logic
  );
end shift;
architecture archi of shift is
signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
            for i in 0 to 6 loop
                tmp(i+1) <= tmp(i);
            end loop;
            tmp(0) <= SI;
        end if;
    end process;
    SO <= tmp(7);
end archi;
```

8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Clear, Serial In, and Serial Out

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(
        C, SI, CLR : in std_logic;
        SO : out std_logic
        );
end shift;
architecture archi of shift is
signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
            tmp <= (others => '0');
        elsif (C'event and C='1') then
            tmp <= tmp(6 downto 0) & SI;
        end if;
    end process;
    SO <= tmp(7);
end archi;
```

8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Parallel Load, Serial In, and Serial Out

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(
  C, SI, ALOAD : in std_logic;
  D  : in std_logic_vector(7 downto 0);
  SO : out std_logic
        );
end shift;
architecture archi of shift is
signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, ALOAD, D)
      begin
        if (ALOAD='1') then
            tmp <= D;
        elsif (C'event and C='1') then
            tmp <= tmp(6 downto 0) & SI;
        end if;
    end process;
    SO <= tmp(7);
end archi;
```

8-bit Shift-Left Register with Negative-Edge Clock, Clock Enable, Serial In, and Serial Out

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity shift is
  port(
        C, SI, CE : in std_logic;
        SO : out std_logic
        );
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='0') then
            if (CE='1') then
                for i in 0 to 6 loop
                    tmp(i+1) <= tmp(i);
                end loop;
                tmp(0) <= SI;
            end if;
        end if;
    end process;
    SO <= tmp(7);
end archi;
```

4-to-1 1-bit MUX using IF Statement

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (
    a, b, c, d : in std_logic;
    s : in std_logic_vector (1 downto 0);
    o : out std_logic         );
end mux;
architecture archi of mux is
  begin
    process (a, b, c, d, s)
    begin
      if (s = "00") then
          o <= a;
      elsif (s = "01") then
          o <= b;
      elsif (s = "10") then
          o <= c;
      else
          o <= d;
      end if;
  end process;
end archi;
```

4-to-1 MUX Using CASE Statement

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (
        a, b, c, d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic
        );
end mux;
architecture archi of mux is
  begin
    process (a, b, c, d, s)
    begin
      case s is
        when "00" => o <= a;
        when "01" => o <= b;
        when "10" => o <= c;
        when others => o <= d;
      end case;
  end process;
end archi;
```

Decoders

```
library ieee;
use ieee.std_logic_1164.all;
entity dec is
```

```
  port (
  sel: in std_logic_vector (2 downto 0);
  res: out std_logic_vector (7 downto 0)
        );
end dec;
architecture archi of dec is
  begin
    res <= "00000001" when sel = "000"
      else "00000010" when sel = "001"
      else "00000100" when sel = "010"
      else "00001000" when sel = "011"
      else "00010000" when sel = "100"
      else "00100000" when sel = "101"
      else "01000000" when sel = "110"
      else "10000000";
end archi;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity dec is
  port (
  sel: in std_logic_vector (2 downto 0);
  res: out std_logic_vector (7 downto 0)
        );
end dec;
architecture archi of dec is
  begin
   with sel select
    res <= "00000001" when "000",
           "00000010" when "001",
           "00000100" when "010",
           "00001000" when "011",
           "00010000" when "100",
           "00100000" when "101",
           "01000000" when "110"
           "10000000" when others;
end archi;
```

Priority Encoder

```
library ieee;
use ieee.std_logic_1164.all;
entity priority is
port (
  sel : in std_logic_vector (7 downto 0);
  code :out std_logic_vector (2 downto 0)
        );
end priority;
architecture archi of priority is
begin
  code <= "000" when sel(0) = '1' else
          "001" when sel(1) = '1' else
          "010" when sel(2) = '1' else
          "011" when sel(3) = '1' else
          "100" when sel(4) = '1' else
          "101" when sel(5) = '1' else
          "110" when sel(6) = '1' else
          "111" when sel(7) = '1' else
          "000";
end archi;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity adder is
  port(  A, B : in std_logic_vector(7 downto 0);
        SUM  : out std_logic_vector(7 downto 0);
        CO   : out std_logic        );
end adder;
architecture archi of adder is
signal tmp: std_logic_vector(8 downto 0);
begin
  tmp <= conv_std_logic_vector((conv_integer(A) +
                             conv_integer(B)),9);
  SUM <= tmp(7 downto 0);
  CO  <= tmp(8);
end archi;

-- Or: Res <= ("0" & A) + ("0" & B);
```

Unsigned 8-bit Adder/Subtractor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity addsub is
  port(  A, B : in std_logic_vector(7 downto 0);
        OPER : in std_logic;
        RES  : out std_logic_vector(7 downto 0)  );
end addsub;
architecture archi of addsub is
  begin
    RES <= A + B when OPER='0'
    else A - B;
end archi;
```

Comparators

VHDL: **(=, /=,<, <=, >, >=)**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity compar is
  port(
  A, B : in std_logic_vector(7 downto 0);
  CMP  : out std_logic
```

```
        );
end compar;
architecture archi of compar is
  begin
    CMP <= '1' when A >= B else '0';
end archi;
```

Unsigned 8x4-bit Multiplier

```
module multiplier(A, B, RES);
  input  [7:0] A;
  input  [3:0] B;
  output [11:0] RES;
  assign RES = A * B;
endmodule
```

Finite State Machine

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fsm is
  port (
        clk, reset, x1 : IN std_logic;
        outp : OUT std_logic
        );
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state, next_state: state_type;
begin
  process1: process (clk, reset)
  begin
      if (reset ='1') then
          state <= s1;
      elsif (clk = '1' and clk'Event) then
          state <= next_state;
    end if;
  end process process1;
  process2 : process (state, x1)
  begin
    case state is
      when s1 =>
        if x1='1' then
            next_state <= s2;
        else
            next_state <= s3;
        end if;
      when s2 => next_state <= s4;
      when s3 => next_state <= s4;
      when s4 => next_state <= s1;
    end case;
  end process process2;
  process3 : process (state)
```

```
    begin
      case state is
        when s1 => outp <= '1';
        when s2 => outp <= '1';
        when s3 => outp <= '0';
        when s4 => outp <= '0';
      end case;
  end process process3;
end beh1;
```

Hierarchical Design

```
library ieee;
use ieee.std_logic_1164.all;
entity my_block is
  port(
        I1 : in std_logic;
        I2 : in std_logic;
        O : out std_logic        );
end my_block;
architecture arch1 of my_block is
  . . .
  . . .
end arch1;


library ieee;
use ieee.std_logic_1164.all;
entity top is
  port(
        DI_1, DI_2, DI_3, DI_4 : in std_logic;
        DOUT1, DOUT2 : out std_logic
        );
end top;
architecture top_arch of top is
  component my_block
    port (
        I1 : in std_logic;
        I2 : in std_logic;
        O : out std_logic
        );
  end component;
  . . .
  . . .
begin
  inst1: my_block port map (
        I1=>DI_1,
        I2=>DI_2,
        O=>DOUT1
        );
  Inst2: my_block port map (
        DI_3, DI_4, DOUT2    );
  . . .
  . . .
end top;
```