VHDL 2 – Combinational Logic Circuits

Reference: Roth/John Text: Chapter 2
Combinational logic

-- Specify behavior via concurrent signal assignments

entity Gates is
  port (a, b, c: in STD_LOGIC; d: out STD_LOGIC);
end Gates;

architecture behavior of Gates is
  signal e: STD_LOGIC;
begin
  -- concurrent signal assignment statements
  e <= (a and b) xor (not c); -- synthesize gate-level ckt
  d <= a nor b and (not e); -- in target technology
end;
Conditional Signal Assignment

```vhdl
signal a, b, c, d, y: std_logic;
signal S: std_logic_vector(0 to 1);
begin
    with S select
    y <= a after 1 ns when "00",
        b after 1 ns when "01",
        c after 1 ns when "10",
        d after 1 ns when "11";
--Alternative "default":
    d after 1 ns when others;
```

4-to-1 Mux
32-bit-wide 4-to-1 multiplexer

signal a, b, c, d, y: bit_vector(0 to 31);
signal S: bit_vector(0 to 1);
begin
  with S select
    y <= a after 1 ns when "00",
    b after 1 ns when "01",
    c after 1 ns when "10",
    d after 1 ns when "11";
--a, b, c, d, y can be any type, as long as they match
Conditional Signal Assignment – Alternate Format

\[ y \leq a \text{ after } 1 \text{ ns when } (S=“00”) \text{ else b after } 1 \text{ ns when } (S=“01”) \text{ else c after } 1 \text{ ns when } (S=“10”) \text{ else d after } 1 \text{ ns; } \]

Use any boolean expression for each condition:

\[ y \leq a \text{ after } 1 \text{ ns when } (F=‘1’) \text{ and } (G=‘0’) \ldots \]
Conditional assignment can model the truth table of a switching function

Model a switching function without deriving logic equations

\[ S \leftarrow A \& B; \quad \text{--} S \text{ is a 2-bit vector with } S \text{ select} \]

\[ Y \leftarrow '0' \text{ when } \text{"00"}, \quad '1' \text{ when } \text{"01"}, \quad '1' \text{ when } \text{"10"}, \quad '0' \text{ when } \text{"11"}, \quad 'X' \text{ when others}; \]

Concatenate operator & combines scalars/vectors into larger vectors
Structural architecture example
(no “behavior” specified)

architecture structure of full_add1 is

component xor -- declare component to be used
    port (x,y: in bit;
        z: out bit);
end component;

for all: xor use entity work.xor(eqns); -- if multiple arch’s
signal x1: bit; -- signal internal to this component
begin -- instantiate components with “map” of connections
    G1: xor port map (a, b, x1); -- instantiate 1st xor gate
    G2: xor port map (x1, cin, sum); -- instantiate 2nd xor gate

…add circuit for carry output…
end;
Associating signals with formal ports

component AndGate port
  (Ain_1, Ain_2 : in BIT; -- formal parameters
   Aout : out BIT);
end component;

-- positional association of “actual” to “formal”:  
A1:AndGate port map (X, Y, Z1);

-- named association (usually improves readability):  
A2:AndGate port map (Ain_2=>Y, Aout=>Z2, Ain_1=>X);

-- both (positional must begin from leftmost formal):  
A3:AndGate port map (X, Aout => Z3, Ain_2 => Y);
Example: D flip-flop

entity DFF is
  port (Preset: in bit;
       Clear: in bit;
       Clock: in bit;
       Data: in bit;
       Q: out bit;
       Qbar: out bit);
end DFF;
7474 D flip-flop equations

architecture eqns of DFF is
    signal A,B,C,D: bit;
    signal QInt, QBarInt: bit;
begin
    A <= not (Preset and D and B) after 1 ns;
    B <= not (A and Clear and Clock) after 1 ns;
    C <= not (B and Clock and D) after 1 ns;
    D <= not (C and Clear and Data) after 1 ns;
    QInt <= not (Preset and B and QbarInt) after 1 ns;
    QBarInt <= not (QInt and Clear and C) after 1 ns;
    Q <= QInt; -- Can drive but not read “outs”
    QBar <= QBarInt; -- Can read & drive “internals”
end;
4-bit Register (Structural Model)

entity Register4 is
  port ( D: in bit_vector(0 to 3);
         Q: out bit_vector(0 to 3);
         Clk: in bit;
         Clr: in bit;
         Pre: in bit);
end Register4;
architecture structure of Register4 is

component DFF -- declare library component to be used
  port (Preset: in bit;
       Clear: in bit;
       Clock: in bit;
       Data: in bit;
       Q: out bit;
       Qbar: out bit);
  end component;

signal Qbar: bit_vector(0 to 3); -- dummy for unused FF Qbar outputs

begin
  -- Signals connect to ports in order listed above
  F3: DFF port map (Pre, Clr, Clk, D(3), Q(3), Qbar(3));
  F2: DFF port map (Pre, Clr, Clk, D(2), Q(2), Qbar(2));
  F1: DFF port map (Pre, Clr, Clk, D(1), Q(1), Qbar(1));
  F0: DFF port map (Pre, Clr, Clk, D(0), Q(0), Qbar(0));
end;
Register Structure

architecture structure of Register4 is
  component DFF -- declare library component to be used
    port (Preset: in bit;
        Clear: in bit;
        Clock: in bit;
        Data: in bit;
        Q: out bit;
        Qbar: out bit);
  end component;
begin
  -- Signals connect to ports in order listed above
  F3: DFF port map (Pre, Clr, Clk, D(3), Q(3), OPEN);
  F2: DFF port map (Pre, Clr, Clk, D(2), Q(2), OPEN);
  F1: DFF port map (Pre, Clr, Clk, D(1), Q(1), OPEN);
  F0: DFF port map (Pre, Clr, Clk, D(0), Q(0), OPEN);
end;

Keyword OPEN indicates an unconnected output
VHDL “Process” Construct

[label:] process (sensitivity list)
  declarations
  begin
    sequential statements
  end process;

- Process statements are executed *in sequence*
- Process statements are executed once at start of simulation
- Process halts at “end” until an event occurs on a signal in the “sensitivity list”
- Allows conventional programming language methods to describe circuit behavior
Combinational logic via process

-- All signals referenced in process must be in the sensitivity list.

entity And_Good is
    port (a, b: in BIT; c: out BIT);
end And_Good;

architecture Synthesis_Good of And_Good is
begin
    process (a,b)     -- gate sensitive to events on (a, b)
    begin
        c <= a and b;  -- c updated on a or b “events”
    end process;
end;
Combinational logic via process

-- This example produces unexpected results.
entity And_Bad is
    port (a, b: in BIT; c: out BIT);
end And_Bad;
architecture Synthesis_Bad of And_Bad is
    begin
        process (a) -- this should be process (a, b)
            begin
                c <= a and b; -- will not react to changes in b
            end process;
    end Synthesis_Bad;
-- synthesis tool may generate a flip flop, triggered by signal a