Digital Design with VHDL

CSE 560M
Lecture 5
Shakir James
Plan for Today

• Announcement
  – Commentary due Wednesday
  – HW1 assigned today. **Begin immediately!**

• Questions

• VHDL help session

• Assignment
Design Entity

• Hardware abstraction

• Entity definition
  – entity declaration
  – architecture body
Entity and Architecture

```
entity and2 is
  port (x, y: in std_logic;
       f: out std_logic);
end and2;

architecture a1 of and2 is
begin
  process(x, y)
  begin
    f <= x and y;
  end process;
end a2;

architecture a2 of and2 is
begin
  f <= '1' when a='1' and b='1' else '0';
end a2;
```

- Entity declaration
  - inputs and outputs

- Architecture body
  - Process description
  - Structural description

Concurrent assignment
  (“same” as process)
Multiple Assignments

- Concurrent region — a “wire”

```vhdl
architecture a1 of e is
begin
  x <= '0';
  y <= x;
  x <= '1';
end a1;
```

- Within a process — “pseudo-sequential”

```vhdl
architecture a2 of e is
begin
  p: process(x, y)
  begin
    x <= '0';
    y <= x;
    x <= '1';
  end process;
end a2;
```

```
x = 'X';
y = 'X';
```

```
x = '1';
y = '1';
```
**Structural Description**

```vhdl
entity and4 is
  port (i0, i1, i2, i3: in std_logic;
       f: out std_logic);
end and4;

architecture circuit of and4 is

  component and2
    port (x, y: in std_logic;
          f: out std_logic);
  end component;

  signal n1, n2: std_logic;

  begin
    a2_1: and2 port map (i0, i1, n1);
    a2_2: and2 port map (i2, i3, n2);
    a2_3: and2 port map (n1, n2, f)
  end circuit;
```
entity and4 is
port (i0, i1, i2, i3: in std_logic;
      f: out std_logic);
end and4;

architecture a2 of and4 is
signal n1, n2: std_logic;
begin
  p: process (i0, i1, i2, i3, n1, n2)
  begin
    n1 <= a and b;
    n2 <= c and d;
    f  <= n1 and n2;
  end process;

end a2;
Combinatorial and Sequential

<table>
<thead>
<tr>
<th></th>
<th>Combinatorial Logic</th>
<th>Sequential Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Examples</td>
<td>• Logic functions</td>
<td>• Latches</td>
</tr>
<tr>
<td></td>
<td>• Arithmetic functions</td>
<td>• Flip-flops</td>
</tr>
<tr>
<td></td>
<td>• Multiplexers</td>
<td>• Registers</td>
</tr>
<tr>
<td></td>
<td>• Decoders</td>
<td>• Counters</td>
</tr>
<tr>
<td>Architecture</td>
<td>• Processes</td>
<td>• Processes</td>
</tr>
<tr>
<td></td>
<td>• Concurrent statements</td>
<td></td>
</tr>
</tbody>
</table>

Concurrent statements are simpler!
Combinatorial Logic as Process

• Common error #1

entity and3 is
  port (a, b, c: in std_logic;
       f: out std_logic);
end and3;

architecture a of and3 is
begin
  p: process (a, b)
  begin
    o <= a and b and c;
  end process;
end a;

c missing in sensitivity list

• Common error #2

architecture a of e is
begin
  p: process (a, b)
  begin
    if a = '0' then
      x <= a; y <= b;
    elsif a = b then
      x <= '0'; y <= '1';
    else
      x <= not a;
    fi
  end process;
end a;

y not defined in else
Clocked Sequential Circuit

entity fsm is
    port (x, rst_l, clk: in std_logic;
          z : out std_logic);
end fsm;

architecture mealy of fsm is
    type states is (s0, s1);
    signal state: states := s0;
begin
  state_trans: process(clk)
  begin
    if (rising_edge(clk)) then
      if (rst_l = '0') then
        state <= s0;
      else
        case state is
          when s0 =>
            if (x = '1') then
              state <= s1;
            end if;
          when s1 =>
            if (x = '1') then
              state <= s0;
            end if;
          end case;
    end if;
  end process state_trans;
  -- code that defines outputs
  z <= '1' when (state = s0 and x = '1')
    or (state = s1 and x = '0')
    else '0';
end mealy;
Assignment

• HW1 assigned, due 9/23

• Readings
  – For Wednesday
    • H&P: Appendix A, sections A3- A.5
    • Commentary: Retrospective on HLL Computers
      – submit commentary to newsgroup before class
  – For Monday
    • H&P: Appendix A, sections A6- A.9
    • V&L: Ch. 4
Exercise 1

• Write a VHDL module implementing a 4 input multiplexer. Each input should be a 32 bit wide `std_logic_vector`.

• Hint
  – Entity:
    • inputs:
    • outputs:
  – Architecture:
    • combinatorial or sequential logic?
library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port (p0,p1,p2,p3: in std_logic_vector(31 downto 0);
    sel   : in std_logic_vector(1 downto 0);
    output : out std_logic_vector(31 downto 0));
end mux;

architecture beh of mux is
begin
  output <= p0 when sel="00" else
           p1 when sel="01" else
           p2 when sel="10" else
           p3 when sel="11";
end beh;
Exercise 2

• Write a VHDL module implementing a synchronous 32 bit counter. A “reset” signal resets the counter to 0. An “en” signal enables the counter modification. An “up” signal indicates whether the counter must be incremented (1)/decremented(0).

• The output of the module is the value of the signal, represented as 32 bit wide std_logic_vector.
library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port (p0, p1, p2, p3: in std_logic_vector(31 downto 0);
        sel : in std_logic_vector(1 downto 0);
        output : out std_logic_vector(31 downto 0));
end mux;

architecture beh of mux is
begin
  output <= p0 when sel="00" else
            p1 when sel="01" else
            p2 when sel="10" else
            p3 when sel="11";
end beh;
architecture beh of counter is
  signal count : integer;
begin
  counter_p: process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        count <= 0;
      else
        if en = '1' then
          if up = '1' then
            count <= count + 1;
          else
            count <= count - 1;
          end if;
        end if;
      end if;
    end if;
  end process;

  output <= conv_std_logic_vector(count,32);
end beh;
Exercise 3

• Use the modules written for exercises 1 and 2 in order to implement a synchronous register file containing 4 counters. The multiplexer will allow to determine which counter will be seen on the single output.

• The module should present a global reset signal, a write enable signal, a read and a write selector, and an up signal.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity counter_reg is
  port (clk : in std_logic;
         write_en : in std_logic;
         up       : in std_logic;
         reset    : in std_logic;
         read_sel : in std_logic_vector(1 downto 0);
         write_sel: in std_logic_vector(1 downto 0);
         output   : out std_logic_vector(31 downto 0));
end counter_reg;

architecture struct of counter_reg is

  component mux
    port (p0,p1,p2,p3: in std_logic_vector(31 downto 0);
          sel : in std_logic_vector(1 downto 0);
          output : out std_logic_vector(31 downto 0));
  end component;

  component counter
    port (clk    : in std_logic;
          en     : in std_logic;
          up     : in std_logic;
          reset  : in std_logic;
          output : out std_logic_vector(31 downto 0));
  end component;

Shakir James
begin

wr_en <= "0000" when write_en = '0' else
    "0001" when write_sel="00" else
    "0010" when write_sel="01" else
    "0100" when write_sel="10" else
    "1000";

cloop: for i in 3 downto 0 generate
    cnt: counter port map (clk,wr_en(i),up,reset,outputs(i));
    end generate;

m: mux port map (outputs(0),outputs(1),outputs(2),outputs(3),
    read_sel,output);

end struct;