EEL 4712  
Midterm 1 – Spring 2015  
VERSION 1

Name:  
UFID:

Sign here if you want your test to be returned in class, where other students might see your score:

---

**IMPORTANT:**
- Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
- As always, the best answer gets the most points.

**COVER SHEET:**

<table>
<thead>
<tr>
<th>Problem#</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (15 points)</td>
<td></td>
</tr>
<tr>
<td>2 (12 points)</td>
<td></td>
</tr>
<tr>
<td>3 (6 points)</td>
<td></td>
</tr>
<tr>
<td>4 (12 points)</td>
<td></td>
</tr>
<tr>
<td>5 (8 points)</td>
<td></td>
</tr>
<tr>
<td>6 (6 points)</td>
<td></td>
</tr>
<tr>
<td>7 (16 points)</td>
<td></td>
</tr>
<tr>
<td>8 (5 points)</td>
<td></td>
</tr>
<tr>
<td>9 (15 points)</td>
<td></td>
</tr>
<tr>
<td>10 (5 points)</td>
<td>5</td>
</tr>
</tbody>
</table>

Total:

**Regrade Info:**
ENTITY _entity_name IS
PORT(_input_name, __input_name : IN STD_LOGIC;
     _input_vector_name : IN STD_LOGIC_VECTOR(_high downto __low);
     __bidir_name, __bidir_name : INOUT STD_LOGIC;
     __output_name, __output_name : OUT STD_LOGIC);
END _entity_name;

ARCHITECTURE a OF _entity_name IS
BEGIN
-- Process Statement
-- Concurrent Signal Assignment
-- Conditional Signal Assignment
-- Selected Signal Assignment
-- Component Instantiation Statement
END a;

_instance_name: _component_name PORT MAP (__component_port => __connect_port,
   __component_port => __connect_port);

WITH __expression SELECT
   __signal <= __expression WHEN __constant_value,
                __expression WHEN __constant_value,
                __expression WHEN __constant_value,
                __expression WHEN __constant_value;
   __signal <= __expression WHEN __boolean_expression ELSE
                __expression WHEN __boolean_expression ELSE
                __expression;

IF __expression THEN
   __statement;
   __statement;
ELSIF __expression THEN
   __statement;
   __statement;
ELSE
   __statement;
   __statement;
END IF;

CASE __expression IS
  WHEN __constant_value =>
     __statement;
     __statement;
  WHEN __constant_value =>
     __statement;
     __statement;
  WHEN OTHERS =>
     __statement;
     __statement;
END CASE;

<generate_label>: FOR <loop_id> IN <range> GENERATE
-- Concurrent Statement(s)
END GENERATE;

type array_type is array(_upperbound downto _lowerbound);
1) (15 points) Fill in the following behavioral VHDL to implement the illustrated circuit. Assume that clk and rst connect to every register. All wires and operations are width bits. Ignore overflow from the adders.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test1 is
  generic (width : positive := 8);
  port (clk, rst : in std_logic;
in1, in2, in3 : in std_logic_vector(width-1 downto 0);
out1, out2, out3 : out std_logic_vector(width-1 downto 0));
end test1;

architecture BHV of test1 is
  signal regIn1, regIn3 : std_logic_vector(width-1 downto 0);
begin
  process(clk, rst)
  begin
    if (rst = '1') then
      regIn1 <= (others => '0');
      regIn3 <= (others => '0');
      out2 <= (others => '0');
    elseif (rising_edge(clk)) then
      regIn1 <= in1;
      regIn3 <= in3;
      out2 <= std_logic_vector(unsigned(in2) + unsigned(regIn3));
    end if;
  end process;
  out1 <= std_logic_vector(unsigned(regIn1) + unsigned(in2));
  out3 <= regIn3;
end BHV;
```
2) (12 points) Draw the circuit that will be synthesized from the following sequential logic description. You can omit the clk and rst signals. Just show registers and add operations. For partial credit add signal labels to registers.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test2 is
  generic(
    width : integer := 0);
  port (
    clk, rst : in std_logic;
    in1, in2, in3 : in std_logic_vector(width-1 downto 0);
    out1, out2 : out std_logic_vector(width-1 downto 0));
end test2;

architecture BNV of test2 is

  signal regIn1, regIn2, regIn3 : std_logic_vector(width-1 downto 0);
  signal regAddOut1, regAddOut2 : std_logic_vector(width-1 downto 0);

begin
  process(clk, rst)
  begin
    if (rst = '1') then
      regIn1 <= (others => '0');
      regIn2 <= (others => '0');
      regIn3 <= (others => '0');
      regAddOut1 <= (others => '0');
      regAddOut2 <= (others => '0');
    elsif (rising_edge(clk)) then
      regIn1 <= in1;
      regIn2 <= in2;
      regIn3 <= in3;
      regAddOut1 <= std_logic_vector(unsigned(regIn1)+unsigned(regIn2));
      regAddOut2 <= regAddOut1;
    end if;
  end process;

  process(regAddOut1, regAddOut2, regIn3)
  begin
    out1 <= regAddOut1;
    out2 <= std_logic_vector(unsigned(regAddOut2)+unsigned(regIn3));
  end process;
end BNV;
3) (6 points) Fill in the CASE_TEST architecture with code that is semantically equivalent to the IF_TEST architecture, but use a case statement instead of an if statement. Hint: use “when others” to include as many conditions as possible.

library ieee;
use ieee.std_logic_1164.all;

entity if_case is
  port (cond : in std_logic_vector(2 downto 0);
        output : out std_logic_vector(1 downto 0));
end if_case;

architecture IF_TEST of if_case is
begin
  process (cond)
  begin
    if (cond(2) = '1') then
      output <= "00";
    elsif (cond(1) = '1') then
      output <= "01";
    elsif (cond(0) = '1') then
      output <= "10";
    else
      output <= "11";
    end if;
  end process;
end IF_TEST;

architecture CASE_TEST of if_case is
begin
  process (cond)
  begin
    case cond is
      when "000" =>
        output <= "11";
      when "001" =>
        output <= "10";
      when "010" =>
        output <= "01";
      when "011" =>
        output <= "01";
      when others =>
        output <= "00";
    end case;
  end process;
end CASE_TEST;
4) a. (6 points) For the IF_TEST architecture in question 3), what synthesis guideline would be violated if you removed the else statement?

   output not defined on all paths

b. (6 points) What type of component would synthesis infer if the else statement was removed?

   latch

5) (8 points) Identify any violations of the synthesis coding guidelines for sequential logic:

   process(clk, rst)
   begin
     if (rst = '1') then
       output <= (others => '0');
     elsif (rising_edge(clk)) then
       output <= input;
     end if;

     if (en = '1') then
       output2 <= input;
     end if;
   end process;

6) (6 points) The following code will generate an error (not a warning) when synthesized in Quartus. Describe the error (hint: it is not a syntax error):

   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

   entity test is
     generic (width : positive := 8);
     port(
       clk     : in std_logic;
       rst     : in std_logic;
       in1, in2 : in std_logic_vector(width-1 downto 0);
       out1, out2 : out std_logic_vector(width-1 downto 0))
   end test;

   architecture BHV of test is
   begin
     process(clk, rst)
     begin
       if (rst = '1') then
         out1 <= (others => '0');
         out2 <= (others => '0');
       elsif (rising_edge(clk)) then
         out1 <= in1;
       end if;
     end process;

     out2 <= std_logic_vector(unsigned(in1)+unsigned(in2));
   end BHV;
7) (16 points) Fill in the provided code to create the illustrated structural architecture using a series of pre-existing FF and adder components. Use the component declarations to determine their I/O. Make sure to use the for-generate loop for the width FF instances, where each FF connects to a single bit of the adder output and the overall output. Declare any required internal signals.

```vhls
library ieee;
use ieee.std_logic_1164.all;

entity test3 is
  generic(width : positive := 8);
  port (
    clk, rst : in std_logic;
    in1, in2 : in std_logic_vector(width-1 downto 0);
    output : out std_logic_vector(width-1 downto 0));
end test3;

architecture STR of test3 is

  component FF
  port (
    clk, rst, D : in std_logic;
    Q   : out std_logic);
end component;

  component adder
  generic(width : positive);
  port (
    in1, in2 : in std_logic_vector(width-1 downto 0);
    sum      : out std_logic_vector(width-1 downto 0));
end component;

  signal adderOut : std_logic_vector(width-1 downto 0);

begin

  U_ADD : adder generic map (width => width)
  port map (
    in1 => in1,
    in2 => in2,
    sum => adderOut
  );

  U_FFS : for i in 0 to width-1 generate
    U_FF : FF port map ( 
      clk => clk,
      rst => rst,
      0 => adderOut(i),
      Q  => output(i)
    );
  end generate U_FFS;
end STR;
```
8) (5 points) Why do generics work with a vhd file but not a vho file?

*Synthesis removes the generic before creating the vho file.*

9) a. (9 points) Define the carry out \( (c_4) \) logic of a 4-bit carry lookahead adder (CLA) in terms of the propagate signals \( (p_i) \), generate signals \( (g_i) \), and carry in \( (c_0) \).

\[
c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\]

b. (3 points) True/False. Creating a wider CLA by connecting the carry out of one CLA into the carry in of another CLA has a constant propagation delay.

*false*

c. (3 points) True/False. Creating a wider CLA by using a tree-based hierarchy of CLAs with carry generation logic at each level (e.g., the hierarchical architecture from lab 3) has a constant propagation delay. List any assumptions.

*false*

10) 5 free points for having to take a test at 8:30am.