**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 (5 points)</td>
<td></td>
</tr>
<tr>
<td>2 (5 points)</td>
<td></td>
</tr>
<tr>
<td>3 (12 points)</td>
<td></td>
</tr>
<tr>
<td>4 (12 points)</td>
<td></td>
</tr>
<tr>
<td>5 (16 points)</td>
<td></td>
</tr>
<tr>
<td>6 (16 points)</td>
<td></td>
</tr>
<tr>
<td>7 (16 points)</td>
<td></td>
</tr>
<tr>
<td>8 (15 points)</td>
<td></td>
</tr>
<tr>
<td>9 (3 points)</td>
<td>3</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
SIGNAL _signal_name : STD_LOGIC;
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) (5 points) Given the following ALU entity:

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

entity ALU is
  generic (
    WIDTH : positive := 16);
  port ( 
    input1, input2 : in std_logic_vector(WIDTH-1 downto 0);
    sel       : in std_logic_vector(3 downto 0);
    output    : out std_logic_vector(WIDTH-1 downto 0));
end ALU;

show the corresponding value of the width generic next to each of the following instantiations:

UUT : alu
generic map (width => 8)
port map ( 
  input1  => input1,  
  input2  => input2,  
  sel     => sel,      
  output  => output);  
Width = 8

UUT2 : alu
port map ( 
  input1 => input3,   
  input2 => input4,   
  sel    => sel2,      
  output => output2);  
Width = 16
```

2) (5 points) True/false. If an entity doesn’t specify a default value for a generic, and an instantiation of the entity does not use a generic map, then the generic defaults to 32.

```vhdl
false
```
3) (12 points) Identify the violation of the synthesis coding guidelines for combinational logic discussed in class (there is only one), and the effect on the synthesized circuit.

```plaintext
process(state, go)
begin

next_state <= state;

case state is
when STATE_0 =>
  done <= '0';
  output <= "00";
  if (go = '1') then
    next_state <= STATE_1;
  end if;

when STATE_1 =>
  output <= "01";
  if (go = '1') then
    next_state <= STATE_2;
  end if;

when STATE_2 =>
  output <= "10";
  if (go = '1') then
    next_state <= STATE_3;
  end if;

when STATE_3 =>
  done <= '1';
  output <= "11";
  if (go = '1') then
    next_state <= STATE_0;
  end if;

when others => null;
end case;
end process;
```
4) (12 points) For the following entity, fill in the waveform for “output” of each architecture assuming the values shown are in decimal format:

declares entity ADD is
    port (input1, input2 : in std_logic_vector(15 downto 0);
          output : out std_logic_vector(15 downto 0);
          overflow : out std_logic);
end ADD;

architecture BHV1 of ADD is
    signal temp : unsigned(16 downto 0);
begin
    process(input1, input2)
    begin
        temp <= unsigned("0"&input1) + unsigned("0"&input2);
        output <= std_logic_vector(temp(15 downto 0));
        overflow <= std_logic(temp(16));
    end process;
end BHV1;

architecture BHV2 of ADD is
begin
    process(input1, input2)
    begin
        variable temp : unsigned(16 downto 0);
        begin
            temp := unsigned("0"&input1) + unsigned("0"&input2);
            output <= std_logic_vector(temp(15 downto 0));
            overflow <= std_logic(temp(16));
        end process;
end BHV2;

architecture BHV3 of ADD is
    signal temp : unsigned(16 downto 0);
begin
    temp <= unsigned("0"&input1) + unsigned("0"&input2);
    output <= std_logic_vector(temp(15 downto 0));
    overflow <= std_logic(temp(16));
end BHV3;

<table>
<thead>
<tr>
<th>Input1</th>
<th>0</th>
<th>0</th>
<th>10</th>
<th>10</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Output (BHV1)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Output (BHV2)</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>12</td>
<td>101</td>
</tr>
<tr>
<td>Output (BHV3)</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>12</td>
<td>101</td>
</tr>
</tbody>
</table>
5) (16 points) Fill in the code provided below to create a ripple carry adder with generic width. You must use a structural architecture with the provided generate loop that connects together the full-adder (FA) components. The circuit should look like this:

![Circuit Diagram]

```
entity adder is
  generic (WIDTH : positive := 8);
  port (x, y, cin : in std_logic_vector(WIDTH-1 downto 0);
        s, cout : out std_logic);
end entity;

architecture RIPPLE_CARRY of adder is
  component fa
    port (x, y, cin : in std_logic;
          s, cout : out std_logic);
  end component;

  signal carry : std_logic_vector(WIDTH downto 0);

begin -- RIPPLE_CARRY
  carry(0) <= cin;
  cout    <= carry(WIDTH);

  U_ADD : for i in 0 to WIDTH-1 generate
    U_FA : fa port map (x => x(i),
                         y => y(i),
                         cin => carry(i),
                         s  => s(i),
                         cout => carry(i+1));
  end generate U_ADD;
end RIPPLE_CARRY;
```
6) (15 points) Draw a schematic for the register-transfer-level (RTL) circuit that will be synthesized from the following VHDL code. Clearly label all inputs, outputs, and show the corresponding location of internal signals.

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

entity bhw_test is
  port (
    clk, rst : in std_logic;
    input1, input2 : in std_logic_vector(7 downto 0);
    input3, input4 : in std_logic_vector(7 downto 0);
    input5 : in std_logic_vector(7 downto 0);
    output : out std_logic_vector(7 downto 0));
end bhw_test;

architecture BHV of bhw_test is

signal add1, add2, add3 : std_logic_vector(7 downto 0);
signal delay1, delay2 : std_logic_vector(7 downto 0);

begin
  process(clk, rst)
  begin
    if (rst = '1') then
      add1 <= (others => '0');
      add2 <= (others => '0');
      add3 <= (others => '0');
      delay1 <= (others => '0');
      delay2 <= (others => '0');
    elsif (clk'event and clk = '1') then
      add1 <= std_logic_vector(unsigned(input1)+unsigned(input2));
      add2 <= std_logic_vector(unsigned(input3)+unsigned(input4));
      add3 <= std_logic_vector(unsigned(add1)+unsigned(add2));
      delay1 <= input5;
      delay2 <= delay1;
    end if;
  end process;

  output <= std_logic_vector(unsigned(add3)+unsigned(delay2));

end BHV;
```

7) (16 points) Fill in the skeleton code to implement the following Moore finite state machine, using the 2-process FSM model. Use the next page if extra room is needed.

entity fsm is
  port (  
    clk, rst, up_dn : in std_logic;  
    x        : out std_logic_vector(1 downto 0));
end fsm;

architecture TWO_PROC of fsm is

  type STATE_TYPE is (STATE0, STATE1, STATE2, STATE3);
  signal state, next_state : STATE_TYPE;

begin

  process(clk, rst)
  begin
    if (rst = '1') then
      state <= STATE0;
    elsif(clk'event and clk = '1') then
      state <= next_state;
    end if;
  end process;

  process(state, up_dn)
  begin
    case state is
      when STATE0 =>
        x <= "00";
        if (up_dn = '1') then
          next_state <= STATE1;
        else
          next_state <= STATE3;
        end if;

      when STATE1 =>
        x <= "01";
        if (up_dn = '1') then
          next_state <= STATE2;
        else
          next_state <= STATE0;
        end if;

      when STATE2 =>
        x <= "10";
        if (up_dn = '1') then

        end case;
  end process;

end TWO_PROC;
next_state <= STATE3;
else
    next_state <= STATE1;
end if;

when STATE3 =>
    x <= "11";
    if (up_dn = '1') then
        next_state <= STATE0;
    else
        next_state <= STATE2;
    end if;

when others => null;
end case;
end process;

end TWO_PROC;
8) a. (5 points) Define the logic for the generate and propagate bits for a single bit $i$ of a carry-lookahead adder in terms of the $x$ and $y$ inputs.

$$
g_i = x_i y_i
$$

$$
\rho_i = x_i + y_i
$$

b. (5 points) Define the block propagate and block generate for a 4-bit CLA, assuming $p_i$ is the propagate of a single bit and $g_i$ is the generate of a single bit.

$$
B_p = p_3 p_2 p_1 p_0
$$

$$
B_g = g_3 + p_3 g_2 + p_2 p_1 g_1 + p_3 p_2 p_1 g_0
$$

c. (5 points) Describe the tradeoffs between a ripple-carry adder and a carry-lookahead adder.

*ripple carry*: delay and area increase linearly with width

*CLA*: delay is constant, area increases super-linearly

d. (3 extra credit points) What realistic limitation prevents a carry-lookahead adder from achieving a constant delay for any width?

fan in

9) (3 free points) Why do I try to avoid multiple choice questions on tests?

a) because they are confusing

b) because there are multiple possible answers

c) because there are too many choices

d) because there are multiple possible answers

e) a + b

f) b + c

g) a + b + c

h) b + c + d

i) a + b + c + d

j) all of the above

k) some of the above

l) b xor c nand h

m) true

n) false

o) neither