EEL 4712
Midterm 1 – Spring 2010

Name: Solution

UFID: _____

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:

Problem#:	Points
1 (8 points)	
2 (8 points)	
3 (16 points)	
4 (16 points)	
5 (16 points)	
6 (16 points)	
7 (16 points)	
8 (8 points)	

Total:

Regrade Info:

1) (8 points) For the entity given below, explain how the generic WIDTH gets its value:

```
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;
```

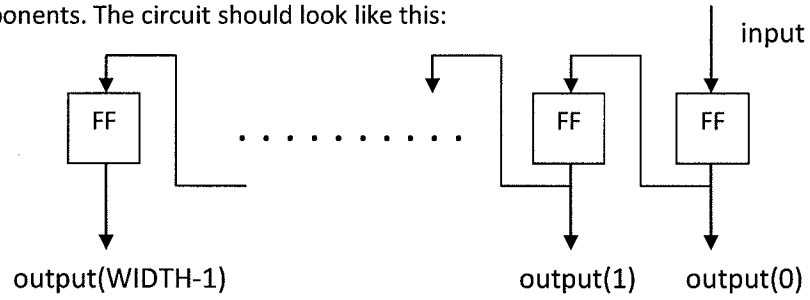
When a component of type ALU is instantiated, a generic map defines WIDTH. If no generic map is used, WIDTH gets the default value of 16.

2) (8 points) Describe the violation of the synthesis guidelines for *combinational logic* in the following example:

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

entity MUX is
  port (
    input1, input2 : in  std_logic_vector(15 downto 0);
    sel            : in  std_logic;
    en             : in  std_logic;
    output         : out std_logic_vector(15 downto 0));
end MUX;

architecture BHV of MUX is
begin  -- BHV

  process(input1, input2, sel)
  begin
    if en = '1' then
      output <= (others => '0');
    elsif sel = '0' then
      output <= input1;
    else
      output <= input2;
    end if;
  end process;

end BHV;
```

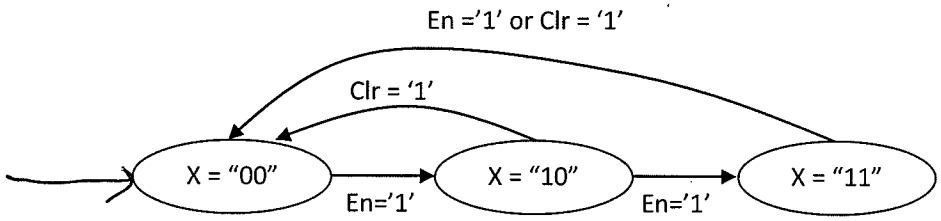
en is not in the sensitivity list

4) (16 points) Fill in the code provided below to create a shift register with generic width. *You must use a structural architecture with the provided generate loop* that connects together the flip-flop (FF) components. The circuit should look like this:




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

entity SH_REG is
  generic (WIDTH    :     positive := 16);
  port (
    clk, rst : in  std_logic;
    input    : in  std_logic;
    output   : out std_logic_vector(WIDTH-1 downto 0));
end SH_REG;

architecture STR of SH_REG is

  component FF
    port (
      clk, rst : in  std_logic;
      input    : in  std_logic;
      output   : out std_logic);
  end component;

  signal temp : std_logic_vector (WIDTH downto 0);

begin  -- STR

  temp(0) <= input;

  U_SH_REG : for i in 0 to WIDTH-1 generate

    U_FF : FF port map (
      clk    => clk,
      rst    => rst,
      input => temp(i),
      output => temp(i+1)

    );
  end generate U_SH_REG;

  output <= temp(WIDTH downto 1);

end STR;
```

6) (16 points) Fill in the skeleton code to implement the following Moore finite state machine, *using the 1-process FSM model*. Assume the "Clr" has a higher priority than "En", so that if "Clr" is asserted, the "Clr" edges are taken instead of the "En" edges. For other possible conditions that are not shown, assume that each state transitions back to itself (e.g. en='0'). Use the next page if extra room is needed.




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

entity FSM is
  port (
    clk, rst, en, clr : in  std_logic;
    x                 : out std_logic_vector(1 downto 0));
end FSM;

architecture BHV of FSM is

  type STATE_TYPE is ( STATE_0, STATE_1, STATE_2

                     );
  signal state : STATE_TYPE;

begin -- BHV
```

process (clk, rst)
begin

```
    if (rst = '1') then
        state <= STATE_0;
        x <= "00";

    elsif (clk'event and clk = '1') then
        case state is
            when STATE_0 =>
                x <= "00";
                if (en = '1') then
                    state <= STATE_1;
                else
                    state <= STATE_0;
            when STATE_1 =>
                x <= "10";
                if (clr = '1') then
                    state <= STATE_0;
```

c(1) - c(3)

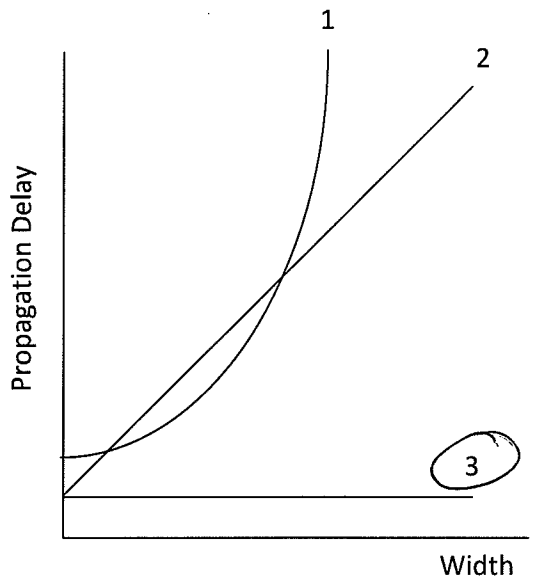
7) a. (8 points) Define each carry bit of a 3-bit carry lookahead adder in terms of the propagate and generate functions, and the carry in c(0).

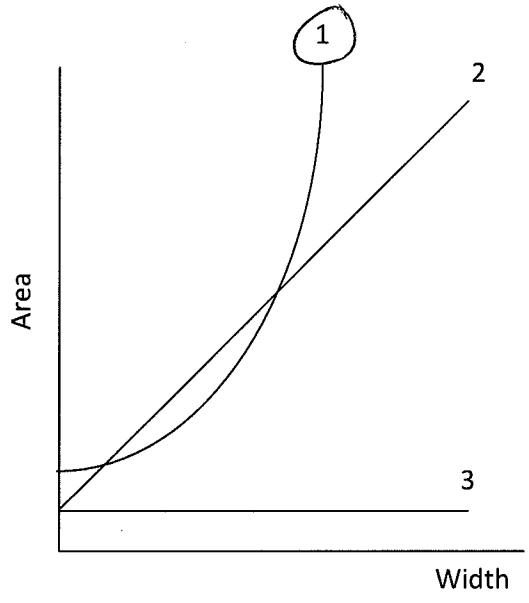
$c(1) = g(0) + p(0)c(0)$

$c(2) = g(1) + p(1)g(0) + p(1)p(0)c(0)$

$c(3) = g(2) + p(2)g(1) + p(2)p(1)g(0) + p(2)p(1)g(0)c(0)$

b. (8 points) For each graph below, circle the number that most closely represents a carry lookahead adder.





8

8) (free points) In the space below, describe every difference between numeric_std and std_logic_arith. Alternatively, leave blank for full credit.