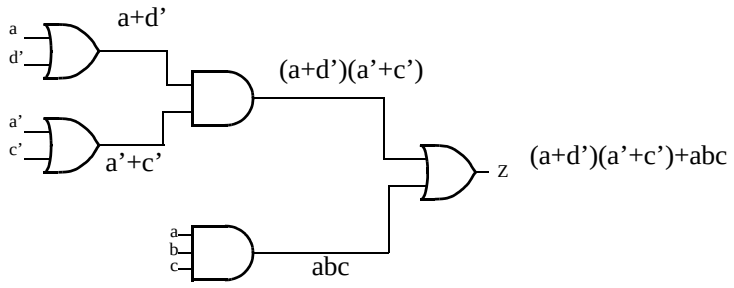
CPE/EE 422/522 **EXAM 2** **Name** KEY

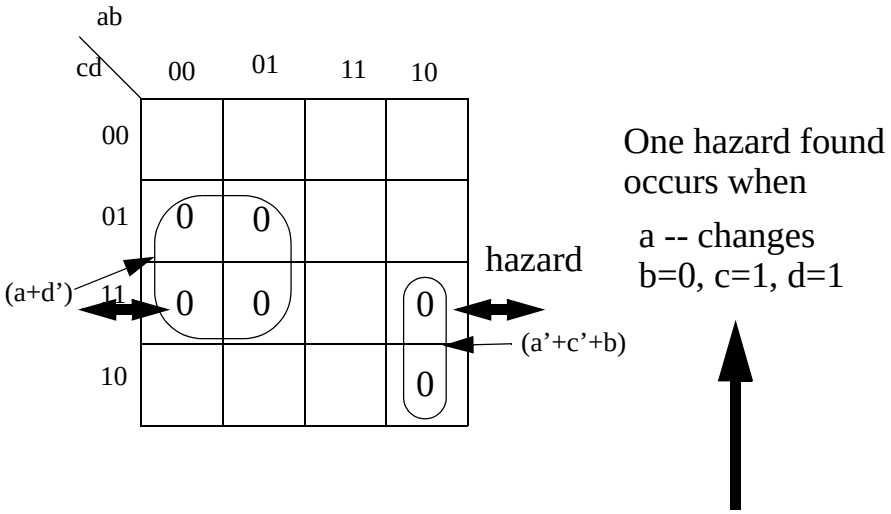
Take Home **Class** _____

Spring Semester 2010

1. [10 points] For the network shown below, find all static 0-hazards. For each 0-hazard found, specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K'map to show why this is the case.



(a+d')(a'+c')+abc	
[(a+d')+abc][(a'+c')+abc]	applying X + YZ = (X + Y)(X + Z)
[{(a+d')+a}{(a+d')+bc}][{(a'+c')+a}{(a'+c')+bc}]	applying X + YZ = (X + Y)(X + Z)
[{a+d'+a}{(a+d'+b)(a+d'+c)}][{a'+c'+a}{(a'+c'+b)(a'+c'+c)}]	applying X + YZ = (X + Y)(X + Z)
(a+a+d')(a+d'+b)(a+d'+c)(a'+c'+a)(a'+c'+b)(a'+c'+c)	removing excess parenthesis
(a+d')(a+d'+b)(a+d'+c)(a'+c'+a)(a'+c'+b)(a'+c'+c)	applying X + X = X
(a+d')(a'+c'+a)(a'+c'+b)(a'+c'+c)	applying X(X+Y)= X



2. [15 points] Write an overloading function for 'not', where the input and returned value are standard logic vectors. The 'not' function should basically simulate a group of inverters. The output bits should be one of the following: 'U', '0', '1', or 'X'. an uninitialized input should give an uninitialized output.

```
function "not" (vec: std_logic_vector)
    return std_logic_vector is
  variable inv_vec: std_logic_vector(vec'length-1 downto 0);
  alias vector: std_logic_vector(vec'length-1 downto 0) is vec;
  begin
    for i in 0 to vec'length-1 loop
      inv_vec(i) := not vector(i);
    end loop;
    return inv_vec;
  end function;
```

3. [15 points] Write a VHDL procedure that will add two bit vectors that represent signed binary numbers. Negative numbers are represented in 2's complement. If the vectors are of different lengths, the shorter one should be sign-extended during the addition. Make no assumpitions about the range for either vector. The procedure call should be of the form **Add2(A,B,SUM)**, where **A** and **B** are the input bit vectors and **SUM** is the output bit vector.

```
procedure Add2 (A, B: in bit_vector;
                signal Sum: inout bit_vector;) is
alias pA: bit_vector(A'length-1 downto 0) is A;
alias pB: bit_vector(B'length-1 downto 0) is B;
variable S: bit_vector(Sum'length-1 downto 0);
variable C: bit;
begin
  for i in S'reverse_range loop
    if i < pA'length and i < pB'length then
      S(i) := pA(i) xor pB(i) xor C;
      C := (pA(i) and pB(i)) or (pA(i) and C) or (pB(i) and C);
    elsif i < pA'length and i >= pB'length then
      S(i) := pA(i) xor pB(pB'left) xor C;
      C := (pA(i) and pB(pB'left)) or (pA(i) and C) or
           (pB(pB'left) and C);
    elsif i >= pA'length and i < pB'length then
      S(i) := pA(pA'left) xor pB(i) xor C;
      C := (pA(pA'left) and pB(i)) or (pA(pA'left) and C) or
           (pB(i) and C);
    else
      S(i) := pA(pA'left) xor pB(pB'left) xor C;
      C := (pA(pA'left) and pB(pB'left)) or (pA(pA'left) and C) or
           (pB(pB'left) and C);
    end if;
  end loop;
  Sum <= S;
end Add2;
```

4. [15 points] A VHDL entity has inputs A and B, and outputs C and D. A and B are initially high. Whenever A goes low, C will go high 5 ns later, and if A changes again, C will change 5 ns later. D will change if B does not change for 3 ns after A changes.
(a) Write the VHDL architecture section using a single process that determines the ouputs C and D.
(b) Introduce another process in the same architecture section that will check that B is stable 2 ns before and 1 ns after A goes high. If it is not stable this process should report an error to the simulator. The process should also report an error if B goes low for a time interval less than 10 ns.

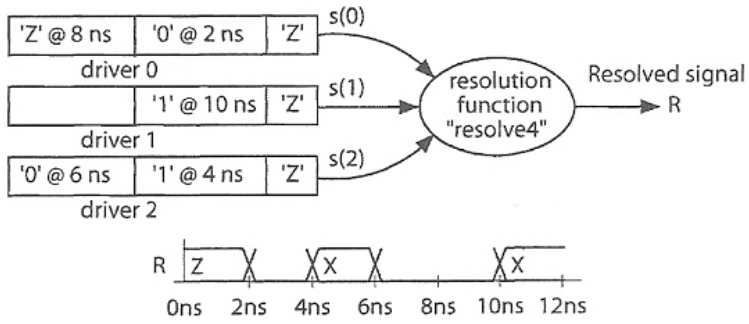
```
entity Solution is
  port(A, B: in bit := '1';
       C, D: inout bit);
end Solution;

architecture Solution of Solution is
begin
  -- a
  process
  begin
    wait until A'event;
    C <= transport not A after 5 ns;
    wait for 3 ns;
    if B'stable(3 ns) then D <= not D; end if;
  end process;

  --b
  process(A, B)
  begin
    if A'event and A = '1' then     -- A has changed
      assert B'stable(2 ns)       -- False if B has changed w/in past 2ns
        report "B not stable for the 2ns before A'event."
        severity error;
    end if;
    if B'event then           -- B has changed
      assert A'stable(1 ns)       -- False if A has changed w/in past 1ns
        report "B not stable for the 1ns following A'event."
        severity error;
      if B = '1' then           -- B changed to high
        assert (B'delayed'last_event >= 10 ns)
          report "B was low for less than 10ns."
          severity error;
      end if;
    end if;
  end process;
end Solution;
```

5. [10 points] Consider the following three concurrent statements, where R is a resolved signal of type X01Z:

R <= transport '0' after 2 ns, 'Z' after 8 ns;
R <= transport '1' after 10 ns;
R <= transport '1' after 4 ns, '0' after 6 ns;

Draw the multiple drivers that will be created and the resolved output signal R from time 0 until time 12 ns.

Time	0	2	4	6	8	10
S(0)	Z	0	0	0	Z	Z
S(1)	Z	Z	Z	Z	Z	1
S(2)	Z	Z	1	0	0	0
R	Z	0	X	0	0	X



6. [15 points] Use Shannon's expansion theorem around **a** and **b** for the function

Y = ab'cdef + a'b'c'd'e + b'c'ef' + abcde'f

so that it can be implemented using only four-variable function generators (4 variable LUT). Draw a block diagram to indicate how **Y** can be implemented using only four-variable function generators. Indicate the function realized by each four-variable function generator.

Y = ab'cdef + a'b'c'd'e + b'c'ef' + abcde'f

$Y = a'b'Y_0 + a'bY_1 + ab'Y_2 + abY_3$ **Shannon's Decompostion Theorem**

case 1: a=0, b=0

$Y_0 = c'd'e + c'ef'$

case 2: a=0, b=1

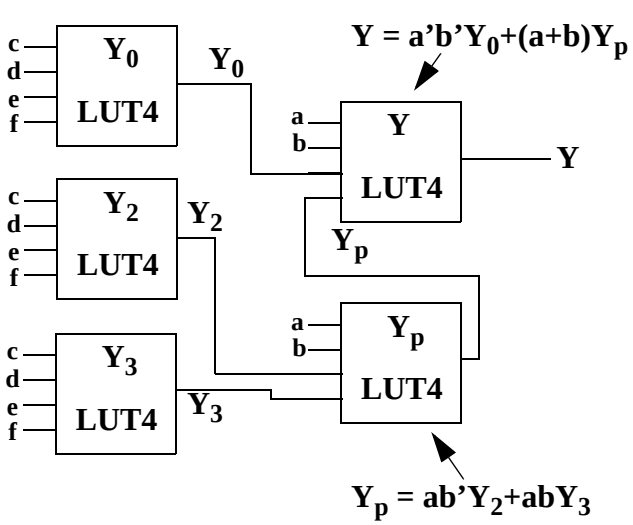
$Y_1 = 0$

case 3: a=1, b=0

$Y_2 = cdef + c'ef'$

case 4: a=1, b=1

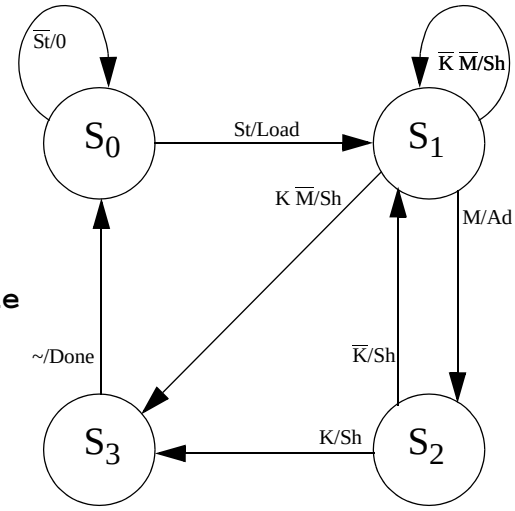
$Y_3 = cde'f$



7. [10 points] For the given state graph, derive the simplified next-state and output equations by inspection. Use the following one-hot state assignments for the flip-flops $Q_0Q_1Q_2Q_3$: S_0, 1000; S_1, 0100; S_2, 0010; S_3, 0001;

Inputs:
St,K,M

Outputs:
Load,Sh, Ad, Done



assuming flip-flops have both resets and presets allowing state S_0 to be represented with $Q_0 = 1$ and the remaining 3 flip-flops at 0.

$Q_0^+ = \overline{St}Q_0 + Q_3$

$Q_1^+ = StQ_0 + \overline{K}\,\overline{M}\,Q_1 + \overline{K}\,Q_2$

$Q_2^+ = MQ_1$

$Q_3^+ = K\,Q_2 + K\,\overline{M}\,Q_1$

$Load = St\,Q_0$

$Sh = \overline{K}\,\overline{M}\,Q_1 + \overline{K}\,Q_2 + K\,Q_2 = \overline{K}\,\overline{M}\,Q_1 + Q_2$

$Ad = M\,Q_1$

$Done = Q_3$

8. [10 points] Summerize your design experience in Laboratory 3 (Capacitance Measurment Laboratory). What were the major problems that you encountered. How did you solve there problems? How stable was the readout? How did your behavioral design implemenation differ from the structural design in terms of ease of implemenation, performance, and FPGA resource usage?

Answer: Many answers possible here.

Comment on the two structural implementation that were presented to you by the instructor. Which of these two implementations would be considered to be better design practice because it adheres more closely to synchronous design methodologies?

The second design is a synchronous design. It has all its elements fed synchronously with a single clock signal. It utilizes Enable signals to determine when the elements will respond to the clock. It is the better design practice because it does not suffer from the nondeterministic timing characteristics that are present when gated clocks are utilized.