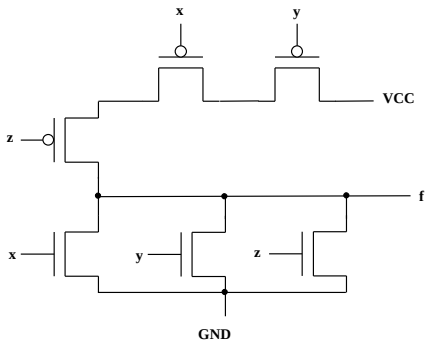
The University of Alabama in Huntsville
ECE Department
CPE 426 01
Final Exam Solution
Spring 2013

1. (6 points) Draw the transistor-level diagram of a three-input CMOS NOR gate.



2. (6 points) Has the following VHDL model been created following good synthesis style? If not, identify any problems

No COUNT is an unconstrained integer

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PAR_TO_SER is
port(START,SHCLK: in STD_LOGIC; PAR_IN: in STD_LOGIC_VECTOR(7 downto 0);
      SO: out STD_LOGIC);
end PAR_TO_SER;

architecture ALG1 of PAR_TO_SER is
begin
  P1:process(START,SHCLK)
    variable COUNT: INTEGER := 0;
    variable DONE: BOOLEAN;
  begin
    if  START = '1' then
      COUNT := 7;
      DONE := FALSE;
    elsif SHCLK'event and SHCLK = '1'  then
      if DONE = FALSE then
        SO <= PAR_IN(COUNT);
        COUNT := COUNT - 1;
      end if;
      if COUNT < 0 then
        DONE := TRUE;
      else
        DONE  := FALSE;
      end if;
    end if;
  end process;
end ALG1;
```

3. (15 points) For the following VHDL architecture, give the values of A, B, C, D, and E, each time a change occurs. Carry this out until no further change occurs.

```
entity prob is
  port (D : inout bit);
end prob;

architecture PROB of PROB is
  signal A, B, C, E : bit;
begin
  P1: process (A, C)
  begin
    B <= A after 3 ns;
    E <= C after 5 ns;
  end process P1;
  C1: C <= A after 10 ns;
  C2: D <= ((A or B) and C)
            or E after 1 ns;
  process
  begin process
    A <= '1' after 5 ns;
    wait;
  end process;
end PROB;
```

Time	A	B	C	D	E
0 ns	0	0	0	0	0
5 ns	**1**	**0**	**0**	**0**	**0**
8 ns	**1**	**1**	**0**	**0**	**0**
15 ns	**1**	**1**	**1**	**0**	**0**
16 ns	**1**	**1**	**1**	**1**	**0**
20 ns	**0**	**1**	**1**	**1**	**1**

Time	**Event**	**Processes Triggered**	**Scheduled Transactions**	**Event?**
5 ns	**A → 1**	**P1**	**B '1', 8 ns**	**Yes**
		P1	**E '0', 10 ns**	**No**
		C1	**C '1', 15 ns**	**Yes**
		C2	**D '0', 6 ns**	**No**
8 ns	**B → 1**	**C2**	**D '0', 9 ns**	**No**
15 ns	**C → 1**	**P1**	**B '1', 18 ns**	**No**
		P1	**E '1', 20 ns**	**Yes**
		C2	**D '1', 16 ns**	**Yes**
16 ns	**D → 1**	**None**		
20 ns	**E → 1**	**C2**	**D'1', 21 ns**	**No**

Scheduling Rules	Transport	Inertial
New before existing	Overwrite existing	Overwrite existing
New after existing	Append new	If $v_{new} = v_{existing}$, append new Elsif t_{new}-$t_{existing}$ > reject append new Else overwrite existing

4. (10 points) Develop an entity and architecture for an inhibited toggle flip-flop. This flip-flop has inputs I0, T, and Reset, and outputs Q and QN. Reset is active high and overrides the action of the other inputs. The flip-flop works as follows. If I0 = '1', the flip-flop changes state on the rising edge of T, if I0 = '0', no state change occurs (except on reset). Make the propagation delays from T to output and reset to output generics.

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

entity INHIBITED_TFF is
  generic (TPHTQ, TPHRQ : time);
  port (I0, T, RESET : in std_logic;
        Q, QN : out std_logic);
end INHIBITED_TFF;

architecture BEHAV of INHIBITED_TFF is
begin
  process (I0, T, RESET)
  begin
    if (RESET = '1') then
      Q <= '0' after TPHRQ;
      QN <= '1' after TPHRQ;
    elsif (T'event and T = '1' and I0 = '1') then
      Q <= not Q after TPHTQ;
      QN <= not QN after TPHTQ;
    end if;
  end process;
end BEHAV;
```

5. (1 point) _**SDF**_ files provide realistic delays for post-layout simulation.

6. (1 point) _**Xilinx**_ and _**Altera**_ are the two main FPGA vendors.

7. (1 point) _**False**_ (True or False) Gate delay is by far the biggest contributor to circuit delay.

8. (1 point) _**True**_ (True or False) A synthesis tool will ignore all **`after`** clauses in a VHDL model.

9. (5 points) If the NRE costs for CBIC and ASIC circuits are $100,000 and $750,000, respectively, and the cost of individual parts for CBIC and ASIC circuits are $20 and $6, respectively, what is the break-even manufacturing volume for these two types of circuits?

   $100,000 + $20x = $750,000 + $6x
   14x = 650,000
   x = 46429

10. (10 points) Write a VHDL procedure called `scan_results` with an **in**-mode `std_logic_vector` signal parameter `results`, and **out**-mode variable parameters `majority_value` of type `std_logic`, `majority_count` of type natural and `tie` of type Boolean. The procedure counts the occurrences of '0' and '1' values in results. It sets `majority_value` to the most frequently occurring value, `majority_count` to the number of occurrences and `tie` to true if there are an equal number of occurrences of '0' and '1'. If the number of occurrences is equal, the procedure sets `majority_value` to 'X' and `majority_count` to the number of occurrences.

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

package MINE is
  procedure scan_results (signal results : in std_logic_vector;
                          variable majority_value : out std_logic;
                          variable majority_count : out natural;
                          variable tie : out Boolean);
end package MINE;

package body MINE is
  procedure scan_results (signal results : in std_logic_vector;
                          variable majority_value : out std_logic;
                          variable majority_count : out natural;
                          variable tie : out Boolean) is
    variable one_count, zero_count : natural:= 0;
  begin
    tie := false;
    majority_value := 'X';
    for i in results'range loop
      if results(i) = '0' then
        zero_count := zero_count + 1;
      elsif results(i) = '1' then
        one_count := one_count + 1;
      end if;
    end loop;
    if (one_count = zero_count) then
      tie := true;
      majority_count := zero_count;
    elsif (one_count > zero_count) then
      majority_value := '1';
      majority_count := one_count;
    else
      majority_value := '0';
      majority_count := zero_count;
    end if;
    return;
  end scan_results;
end package body MINE;
```

11. (15 points) Design an FSM circuit for controlling a simple home security system. The operation of the system is as follows.

Inputs: Front gate switch (FS)
Motion detector switch (MS)
Asynchronous reset switch (R)
Clear switch (C)

Outputs: Front gate melody (FM)
Motion detector melody (MM)

- When the reset switch (R) is asserted, the FSM goes to the initialization state (S_init) immediately.
- From state S_init, the FSM unconditionally goes to the wait state (S_wait).
- From state S_wait, the FSM waits for one of the four switches to be activated. All the switches are active-high, so when a switch is pressed or activated, it sends out a 1. The following actions are taken when a switch is pressed:
    - When FS is pressed, the FSM goes to state S_front. In state S_front, the front gate melody is turned on by setting FM = 1. The FSM remains in state S_frontuntil the clear switch is pressed. Once the clear switch is pressed, the FSM goes back to S_wait.
    - When MS is activated, the FSM goes to state S_motion. In state S_motion, MM is turned on with a 1. MM will remain on for two more clock periods and then the FSM will go back to S_wait.
    - From any state, as soon as the reset switch is pressed, the FSM immediately goes back to state S_init.
    - Pressing the clear switch only affects the FSM when it is in state S_front. The clear switch had no effect on the FSM when it is in any other state.
    - Any unused state encoding will have S_init as their next state.

```
entity ALARM is
      Port (CLK, R, MS, FS, C : in bit;
         MM, FM : out bit);
end ALARM;

architecture BEHAV of ALARM is
  type STATES is (S_INIT, S_WAIT, S_FRONT, S_MOTION0, S_MOTION1, S_MOTION2);
  signal NEXT_STATE, CURRENT_STATE : STATES;
begin
  process (CURRENT_STATE, C, MS, FS)
  begin
    case (CURRENT_STATE) is
      when S_INIT => NEXT_STATE <= S_WAIT;
      when S_WAIT => if (FS = '1') then
                       NEXT_STATE <= S_FRONT;
                     elsif (MS = '1') then
                       NEXT_STATE <= S_MOTION0;
                     else
                       NEXT_STATE <= S_WAIT;
                     end if;
      when S_FRONT => if (C = '1') then
                        NEXT_STATE <= S_WAIT;
                      else
                        NEXT_STATE <= S_FRONT;
                      end if;
      when S_MOTION0 => NEXT_STATE <= S_MOTION1;
      when S_MOTION1 => NEXT_STATE <= S_MOTION2;
      when S_MOTION2 => NEXT_STATE <= S_WAIT;
    end case;
  end process;

  process (R, CLK)
  begin
    if (R = '1') then
      CURRENT_STATE <= S_INIT;
    elsif (CLK'event and CLK = '1') then
      CURRENT_STATE <= NEXT_STATE;
    end if;
  end process;
```

```
  process (CURRENT_STATE)
  begin
    case CURRENT_STATE is
      when S_INIT | S_WAIT => MM <= '0';
                              FM <= '0';
      when S_FRONT => MM <= '0';
                      FM <= '1';
      when S_MOTION0 | S_MOTION1 | S_MOTION2 => MM <= '1';
                                                FM <= '0';
    end case;
  end process;

end BEHAV;
```

12. (1 point) The starting point from which a verification plan can be created is a **_design specification_**.

13. (1 point) List one type of coverage considered during the verification process. **_code, path, assertion, functional_**

14. (2 points) List two layers that may be present in a layered testbench:
    __scenario____
    __transaction__

15. (1 point) A(n) **_interface_** is a construct in System Verilog that represents a bundle of wires and has intelligence.

16. (10 points) Create a VHDL entity named `en_mux_821` that represents a 8:1 multiplexer with an active-low enable input which has an architecture that uses a case statement to represent the functionality of the multiplexer. Create a second entity and its accompanying architecture that represents a 16:1 multiplexer by using two instances of the `en_mux_821` entity.

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

entity EN_MUX_821 is
  port (EN : in std_logic;
        S : in std_logic_vector(2 downto 0);
        D : in std_logic_vector(7 downto 0);
        O : out std_logic);
end EN_MUX_821;
```

```
architecture BEHAVE of EN_MUX_821 is
begin
  process (EN, S, D)
    variable TEMP : std_logic;
  begin
    case S is
      when "000" => TEMP := d(0);
      when "001" => TEMP := d(1);
      when "010" => TEMP := d(2);
      when "011" => TEMP := d(3);
      when "100" => TEMP := d(4);
      when "101" => TEMP := d(5);
      when "110" => TEMP := d(6);
      when "111" => TEMP := d(7);
      when others => TEMP := d(0);
    end case;
    if (EN = '0') then
      O <= TEMP;
    else
      O <= 'Z';
    end if;
  end process;
end BEHAVE;

library ieee;
use ieee.std_logic_1164.all;

entity EN_MUX_1621 is
  port (EN : in std_logic;
        S : in std_logic_vector(3 downto 0);
        d : in std_logic_vector(15 downto 0);
        O : out std_logic);
end EN_MUX_1621;

architecture STRUCT of EN_MUX_1621 is
  signal T0, T1 : std_logic;
begin
  T0 <= not S(3);
  U1: entity work.EN_MUX_821(behave)
      port map (EN => T0, S => S(2 downto 0),
                D => D(7 downto 0), O => t1);
  U2: entity work.EN_MUX_821(behave)
      port map (EN => s(3), s => S(2 downto 0),
                D => D(15 downto 8), O => T1);
  O <= T1 when EN = '0' else 'Z';
end STRUCT;
```

17. (1 point) _**False**_ (True or False) A signal initialized in its declaration will be properly initialized post-synthesis.

18. (1 point) _**True**_ (True or False) Wait statements can be used to imply clocked behavior to the synthesis tool.

19. (1 point) The @ construct in System Verilog is similar to the _**wait**_ construct in VHDL.

20. (1 point) A _**constrained random**_ test environment allows you to run hundreds of tests without having to hand check the results.

21. (10 points) Develop a functional model of a 4-bit carry-look-ahead adder. The-adder has two 4-bit data inputs a (3 downto 0) and b (3 downto 0); a 4-bit data output, s (3 downto 0); a carry input, c_in; a carry output, c_out; a carry generate output, g, and a carry propagate output, p. The adder is described by the logic equations:

$S_i = A_i \oplus B_i \ C_{i-1} \oplus$ (delay is 5 ns)
$G_i = A_iB_i$ (delay is 2 ns)
$P_i = A_i + B_i$ (delay is 3 ns)
$C_i = G_i + P_iC_{i-1}$
$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$
$P = P_3P_2P_1P_0$

where the G_i are the intermediate carry generate signals, the P_i are the intermediate carry propagate signals and the C_i are the intermediate carry signals. C_{-1} is c_in and C_3 is c_out. Use a loop construct for S_i, G_i, P_i and C_i.

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

entity CLA_ADDER is
  port (A, B : in std_logic_vector(3 downto 0);
        S : out std_logic_vector(3 downto 0);
        C_IN : in std_logic;
        C_OUT, P, G : out std_logic);
end CLA_ADDER;

architecture BEHAVE of CLA_ADDER is
  signal PS, GS : std_logic_vector(3 downto 0);
begin
  process (A, B, C_IN)
    variable C : std_logic_vector(3 downto 0);
  begin
    S(0) <= A(0) xor B(0) xor C_IN after 5 ns ;
    PS(0) <= A(0) or B(0) after 3 ns;
    GS(0) <= A(0) and B(0) after 2 ns;
    C(0) := GS(0) or (PS(0) and C_IN);
    for i in 1 to 3 loop
      S(i) <= A(i) xor B(i) xor C(i-1) after 5 ns;
      PS(i) <= A(i) or B(i) after 3 ns;
      GS(i) <= A(i) and B(i) after 2 ns;
      C(i) := GS(i) or (PS(i) and C(i-1));
    end loop;
    C_OUT <= C(3);
    G <= GS(3) or (PS(3) and GS(2)) or (PS(3) and PS(2) and GS(1))
         or (PS(3) and PS(2) and PS(1) and GS(0));
    P <= PS(3) and PS(2) and PS(1) and PS(0);
  end process;
end BEHAVE;
```

Bonus:

(5 points) A synchronous (4-bit) up/down decade counter with outputs COUNT and CO works as follows: All state changes occur on the rising edge of the CLK input, except the asynchronous clear (CLR). When CLR = 0, the counter is reset regardless of the values of the other inputs. The inputs are DATA, CLR, LOAD, EN, UP.

If the LOAD input is 0, the data input DATA is loaded into the counter.
If LOAD = EN = UP =1, the counter is incremented.
If LOAD = EN =1, and UP = 0, the counter is decremented.
If LOAD = EN = UP = 1, the carry output (CO) =1 when the counter is in state 9.
If LOAD = EN = 1 and UP = 0, the carry output (CO) =1 when the counter is in state 0.

Create a SystemVerilog interface for this circuit that has a clocking block that defines directions relative to the test bench.

```
interface counter_if(input bit CLK);
  logic LOAD, EN, UP, CO, CLR;
  logic [3:0] COUNT, DATA;

  clocking cb @(posedge CLK);
    input COUNT, CO;
    output LOAD, EN, UP, CLR, DATA;
  endclocking

  modport TEST (clocking cb);
endinterface
```