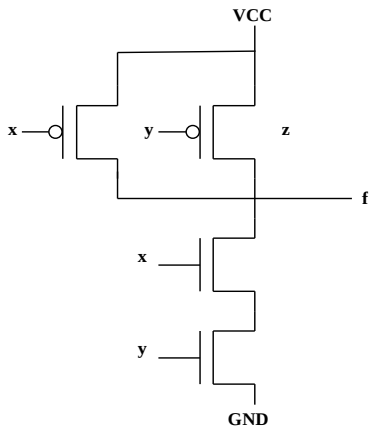
The University of Alabama in Huntsville
ECE Department
CPE 426 01
Final Exam Solution
Spring 2016

1. (6 points) Draw the transistor-level diagram of a two input CMOS NAND gate.



2. (5 points) If the NRE costs for CBIC and ASIC circuits are $650,000 and $3,500,000, respectively, and the cost of individual parts for CBIC and ASIC circuits are $42 and $16, respectively, what is the break-even manufacturing volume for these two types of circuits?

650000 + 42x = 3500000 + 16x
26x = 2850000
x = 109616

3. (1 point) _**'event, 'range, 'high, 'low, etc.**_ is an example of a VHDL attribute.

4 (1 point) _**Synthesis**_ is the process of transforming a behavioral description into a structural gate-level circuit.

5. (1 point) A(n) _**SDF file**_ provides the gate-level circuit with accurate timing backannotated from the layout.

6. (6 points) For the process given below, A, B, C, and D are all integers that have a value of 0 at time = 10 ns. If E changes from '0' to '1' at time 20 ns, specify all resulting changes. Indicate the time at which each change will occur, the signal/variable affected, and the value to which it will change.

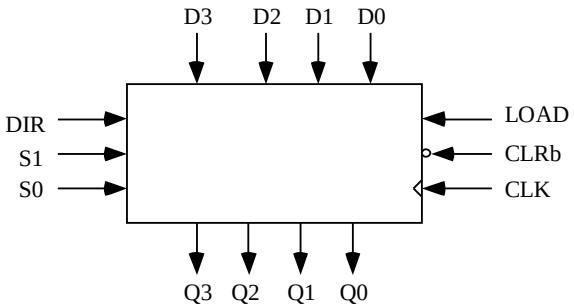
```
process
  variable F : integer := 1; variable A : integer := 0;
begin
  wait on E;
  A := 1;
  F := A + 5;
  B <= E + 1 after 5 ns;
  C <= F + 2;
  D <= A + 5 after 15 ns;
  A := A + 5;
end process;
```

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

Time	A	B	C	D	E	F	
10 ns	0	0	0	0	0	1	
20 ns	0	0	0	0	1	1	given
20 ns	1	0	0	0	1	1	After executing A := 1
20 ns	1	0	0	0	1	6	After executing F := A + 5
20 ns	6	0	0	0	1	6	After executing A := A + 5
20 + Δ ns	6	0	8	0	1	6	As a result of C <= F + 2
25 ns	6	2	8	0	1	6	As a result of B <= E + 1 after 5 ns
35 ns	6	2	8	6	1	6	As a result of D <= A + 5 after 15 ns

7. (1 point) A process is triggered whenever a signal in its _**sensitivity list**_ has an event on it.

8. (18 points) A barrel shifter is a shift register in which the data can be shifted either by one bit position, as in a normal shift register, or by multiple positions. Design a four-bit barrel shifter that can shift to the right or left by 0, 1, 2, or 3 bits and has synchronous clear and parallel load capabilities. Clear loads all zeros into the shift register and has priority over load. If DIR = '0', shift left, else shift right. The binary value of S1 and S0 dictate the amount of shift, for example S1S0 = 10 means shift by 2 bit positions. Shift in 0s as needed. (a) (4 points) Write an entity for the barrel shifter. (c) (11 points) Write an architecture for the barrel shifter




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

entity BARREL is
  port (D : in std_logic_vector(3 downto 0);
        CLK, LOAD, CLR, DIR : in std_logic;
        S : in std_logic_vector(1 downto 0);
        Q : out std_logic_vector(3 downto 0));
end BARREL;

architecture BARREL of BARREL is
  signal TEMP : std_logic_vector (3 downto 0);
begin
  process (CLK)
  begin
    if (CLK'event and CLK = '1') then
      if (CLR = '0') then
        TEMP <= "0000";
      elsif (LOAD = '1') then
        TEMP <= D;
      elsif (DIR = '0') then
        case S is
          when "00" => TEMP <= TEMP;
          when "01" => TEMP <= TEMP(2 downto 0) & '0';
          when "10" => TEMP <= TEMP(1 downto 0) & "00";
          when "11" => TEMP <= TEMP(0) & "000";
          when others => TEMP <= TEMP;
        end case;
      else
        case S is
          when "00" => TEMP <= TEMP;
          when "01" => TEMP <= '0' & TEMP(3 downto 1);
          when "10" => TEMP <= "00" & TEMP(3 downto 2);
          when "11" => TEMP <= "000" & TEMP(3);
          when others => TEMP <= TEMP;
        end case;
      end if;
    end if;
  end process;
  Q <= TEMP;
end BARREL;
```

9. (1 point) _**Boolean, bit**_ is a predefined enumerated type in VHDL.

10. (18 points) (a) (9 points) Create a VHDL entity named mux4to1 that represents a 4-to-1 multiplexer which has an architecture which uses a case statement to represent the functionality of the multiplexer. (b) (9 points) Create a second entity and its accompanying architecture that represents a 16-to-1 multiplexer by using four instances of the mux4to1 entity.

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

entity MUX4TO1 is
  port ( DATA_IN : in std_logic_vector (3 downto 0);
         SEL : in  std_logic_vector (1 downto 0);
         DATA_OUT : out std_logic);
end MUX4TO1;

architecture CASEMUX of MUX4TO1 is
begin
  process (DATA_IN, SEL)
  begin
    case SEL is
      when "00"    => DATA_OUT <= DATA_IN(0);
      when "01"    => DATA_OUT <= DATA_IN(1);
      when "10"    => DATA_OUT <= DATA_IN(2);
      when "11"    => DATA_OUT <= DATA_IN(3);
      when others  => DATA_OUT <= 'X';
    end case;
  end process;
end CASEMUX;

library ieee;
use ieee.std_logic_1164.all;
use work.all;

entity MUX16TO1 is
  port (DATA_IN  : in std_logic_vector (15 downto 0);
        SEL      : in std_logic_vector (3 downto 0);
        DATA_OUT : out std_logic);
end MUX16TO1;

architecture STRUCT of MUX16TO1 is
  component MUX4TO1C
    port (DATA_IN  : in std_logic_vector (3 downto 0);
          SEL      : in std_logic_vector (1 downto 0);
          DATA_OUT : out std_logic);
  end component;
  for all : MUX4TO1C use entity MUX4TO1(CASEMUX);
  signal INTERNAL : std_logic_vector (3 downto 0);
begin
  U1 : MUX4TO1C port map (DATA_IN(3 downto 0), SEL(1 downto 0), INTERNAL(3));
  U2 : MUX4TO1C port map (DATA_IN(7 downto 4), SEL(1 downto 0), INTERNAL(2));
  U3 : MUX4TO1C port map (DATA_IN(11 downto 8), SEL(1 downto 0), INTERNAL(1));
  U4 : MUX4TO1C port map (DATA_IN(15 downto 12), SEL(1 downto 0), INTERNAL(0));
  U5 : MUX4TO1C port map (INTERNAL(3 downto 0), SEL(3 downto 2), DATA_OUT);
end STRUCT;
```

11. (1 point) One can create _**assertions**_ about signals in your design to check their behavior and temporal relationships.

12. (1 point) Hardware description languages add provide the _concurrency_ that regular programming languages do not have.

13. (1 point). _Routing_ is the process of making the connections between standard cells.

14. (4 points) List the four types of paths that must be considered when doing timing analysis of sequential circuits.

    inputs to outputs

    inputs to inputs of registers

    outputs of registers to inputs of registers

    outputs of registers to outputs

15. (1 point) _Standard cells_ are primitives that are all the same height and varying widths.

16. (6 points) Create a class `Exercise1` containing two random variables, 8-bit `data` and 4-bit `address`. (a) Create a constraint block that keeps address to 3 or 4. (b) In an initial block, construct an `Exercise1` object and randomize it.

```
package mine;
  class Exercise1;
    // The random variables
    rand bit [7:0] data;
    rand bit [3:0] address;
    // Limit the values for address
    constraint address_dist
    { address inside{[3:4]};}
  endclass : Exercise1
endpackage : mine

  Exercise1 p;
  initial begin
    p = new();
    p.randomize();
  end
```

17. (12 points) Consider a tape player controller that has five inputs and three outputs. Four of the inputs are the control buttons on the tape player. The input PL is 1 if the play button is pressed, the input RE is 1 if the rewind button is pressed, the input FF is 1 if the fast forward button is pressed, and the input ST is 1 if the stop button is pressed. The fifth input to the control circuit is M, which 1 is the special "music sensor" detects music at the current tape position. The three outputs of the control circuit are P, R, and F, which make the tape play, rewind, and fast forward, respectively, when 1. Even though the RE and FF buttons will be held down for more than one clock cycle, a circuit has been created that makes them one for only one cycle. To test this tape player using System Verilog, the following interface, top module, packet files have been created. There is also an incomplete test module that repeatedly reads inputs from a text file and applies them. Add the following functionality to the test module.

    If R = '1' or F = '1', set M = '1' and keep it high for a random number of cycles as determined by the constraints set in the packet and then set it to '0'.

```
interface tape_if(input bit clk);
  logic RESET, PL, RE, FF, ST, M, P, F, R;
  logic [15:0] COUNT;

  clocking cb @(negedge clk);
    output RESET, PL, RE, FF, ST, M;
    input P, F, R;
  endclocking

  modport TEST (clocking cb);
endinterface

module top;
  bit clk;
  logic ff_one, re_one;
  always #5 clk = ~clk;

  tape_if tapeif(clk);
  tape_player u1 (.CLK (clk),
          .RESET (tapeif.RESET),
          .PL (tapeif.PL),
          .RE (re_one),
          .FF (ff_one),
          .ST (tapeif.ST),
          .M (tapeif.M),
          .COUNT (tapeif.COUNT),
          .P (tapeif.P),
          .F (tapeif.F),
          .R (tapeif.R));
  one_cycle u2 (.PUSH (tapeif.FF),
                .CLK (clk),
                .R (tapeif.RESET),
                .SINGLE (ff_one));
  one_cycle u3 (.PUSH (tapeif.RE),
                .CLK (clk),
                .R (tapeif.RESET),
                .SINGLE (re_one));
  test t1(tapeif);
endmodule : top

package mine;
  class Packet;
    // The random variables
    rand int music;
    // Limit the values for music
    constraint music_dist
    { music dist {[20:25] := 10,
                  [26:39] := 20,
                  [40:50] := 15};
    }
  endclass : Packet
endpackage : mine
```

```
import mine::*;
module test(tape_if.TEST tapeif);
  Packet p;
  integer in, status;
  logic reset, ff, re, pl, st;
  initial begin
    p = new();
  end
  initial begin
    repeat (10) @ (tapeif.cb);
    repeat (200)
    begin
      while (!$feof(in))
      begin
        status = $fscanf(in, "%b %b %b %b %b\n",
                         reset, pl, ff, re, st);
        tapeif.cb.RESET <= reset;
        tapeif.cb.FF <= ff;
        tapeif.cb.RE <= re;
        tapeif.cb.PL <= pl;
        tapeif.cb.ST <= st;
        p.randomize();
        if (tapeif.cb.F == 1'b1 || tapeif.cb.R == 1'b1)
        begin
          tapeif.cb.M <= 1'b1;
          repeat (p.music) @tapeif.cb;
          tapeif.cb.M <= 1'b0;
        end
        @(tapeif.cb);
      end // end while
      $rewind(in);
    end // repeat 200
    $finish;
  end // initial
endmodule : test
```

18. (9 points) For the following interface, add the following code.
(a) A clocking block that is sensitive to the negative edge of the clock. This clocking block needs to contain all I/O from the perspective of the master. The signals write, data_in and address are provided by the master to the slave. The slave provides data_out to the master.
(b) A modport for the testbench called `master`, and a modport for the DUT called `slave`.
(c) Use the clocking block in the I/O list for the master modport.

```
interface my_if(input bit clk);
  bit write; bit [15:0] data_in; bit [7:0] address;
  logic [15:0] data_out;

  clocking cb @(negedge clk);
     output  write, address, data_in; input  data_out;
  endclocking;

   modport slave (input clk, write, data_in, output data_out);
   modport master(clocking cb);

endinterface
```

19. (7 points) Write a single VHDL model which represents an AND gate with an arbitrary number of inputs, N. (a) (3 points) entity, (b) (3 points) architecture

```
library ieee;
use ieee.std_logic_1164.all;
entity arb_and is
  generic (N : integer := 2);
  port (X : in std_logic_vector(N - 1 downto 0);
        Y : out std_logic);
end entity arb_and;

architecture behave of arb_and is
begin
  process (X)
    variable TEMP : std_logic:= '1';
  begin
    for I in X'range loop
      TEMP := TEMP and X(I);
    end loop;
    Y <= TEMP;
  end process;
end behave;
```