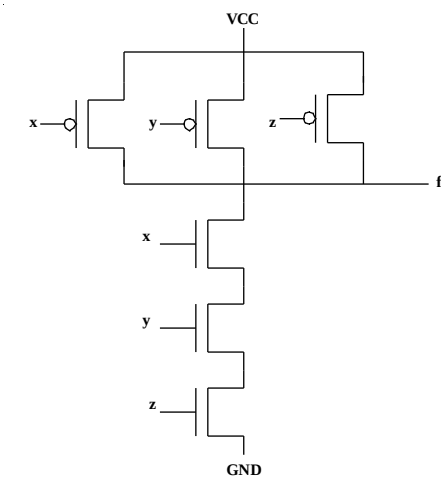
The University of Alabama in Huntsville
ECE Department
CPE 526 01
Final Exam Solution
Spring 2020

1. (5 points) Draw the transistor-level diagram of a three input CMOS NAND gate.



2. (4 points) If the NRE costs for FPGA and CBIC circuits are $50,000 and $800,000, respectively, and the cost of individual parts for FPGA and CBIC circuits are $52 and $11, respectively, what is the break-even manufacturing volume for these two types of circuits?

800000 + 52x = 50000 + 11x
41x = 750000
x = 182293

3. (1 point) A _**testbench**_ provides stimuli and captures responses.

4 (1 point) A _**constraint**_ is a set of relational expressions that must be true for the chosen value of the variables.

5. (1 point) _**Assertions**_ are pieces of declarative code that check the relationships between design signals, either once or over a period of time.

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 + 3 after 4 ns;
  A := A + 5;
  C <= F + 2;
  D <= A + 5 after 12 ns;

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	1	0	0	0	1	1	
20 ns	**1**	**0**	**0**	**0**	**1**	**6**	**F := A + 5;**
20 ns	**6**	**0**	**0**	**0**	**1**	**6**	**A := A + 5;**
20 ns + Δ	**6**	**0**	**8**	**0**	**1**	**6**	**C <= F + 2;**
24 ns	**6**	**4**	**8**	**0**	**1**	**6**	**B <= E + 3 after 4 ns;**
32 ns	**6**	**4**	**8**	**11**	**1**	**6**	**D <= A + 5 after 12 ns;**

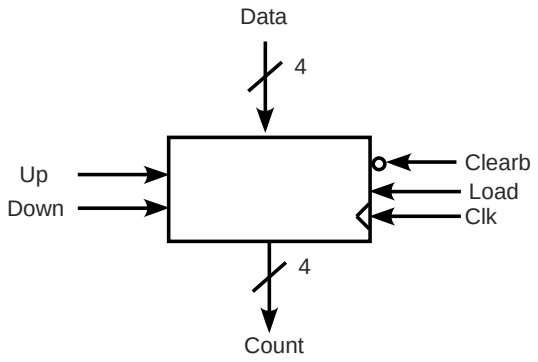
7. (1 point) _**False**_ (True/False) Gate delay is the dominant factor in determining the delay of a path.

8. (1 point) _**False**_ (True/False) Design verification is a trivial task.

9. (1 point) _**Standard cells**_ are primitives that are all the same height and varying widths.

10. (1 point) A _**configuration**_ is used in VHDL to bind an architecture to an instantiation of an entity.

11. (8 points) Create a System Verilog interface for a 4 bit counter that has an asynchronous clear input that is active low. It also has active high LOAD, UP and DOWN inputs. When data is loaded into the counter, counting proceeds from that loaded value. Include as part of the interface a clocking block that uses a rising edge clocking scheme. Also include modports from a testbench and from a DUT perspective.




```
interface counter_if(input bit clk);
  logic Clearb, Load, Up, Down; //asynch Clearb - do not include in
clocking
  bit [3:0] Data;
  bit [3:0] Count;

clocking cb(posedge clk);
  output Data, Up, Down, Load;
  input Count;
endclocking

  modport testbench (clocking cb, output Clearb);
  modport DUT(input clk, Data, Up, Down, Clearb, Load, output Count);

endinterface
```

12. (4 points) List the four types of paths that must be considered when determining the maximum frequency of a circuit.

Primary inputs to primary outputs

Primary inputs to storage element inputs

Storage element outputs to storage element inputs

Storage element outputs to primary outputs

13. (13 points) a. (2 points) VHDL entity, b. (11 points) VHDL architecture Design a finite state machine for control of lights used to start a race, which works as follows.
Inputs: Reset, Start, Clock Outputs: Red, Yellow, Green

- Only one light can be on at any time.
- The Reset signal forces the circuit into a state in which the red light is turned on.
- When the Start signal is activated, the red light stays on for at least one second, then the yellow light is turned on.
- The yellow light stays turned on one second and then the green light is turned on.
- The green light stays on for at least three seconds and then the red light is turned on and the circuit returns to its reset state.
- The input clock has a period of 1 sec.

```
entity RACE is
  port (CLK, S, RESET  : in std_logic;
        R, Y, G : out std_logic);
end RACE;

architecture SYNTH of RACE is
  type STATE_TYPE is (Reset_State, Stay_Red, Yellow, First_Green,
                      Second_Green, Third_Green);
  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
begin
  process(CLK, RESET)
  begin
    if (RESET = '1') then
      CURRENT_STATE <= Reset_State;
    elsif (CLK'event and CLK = '1') then
      CURRENT_STATE <= NEXT_STATE;
    end if;
  end process;
  process(S, CURRENT_STATE)
  begin
    case CURRENT_STATE is
      when Reset_State => if (S = '1') then
                            NEXT_STATE <= Stay_Red;
                          else
                            NEXT_STATE <= Reset_State;
                          end if;
      when Stay_Red => NEXT_STATE <= Yellow;
      when Yellow => NEXT_STATE <= First_Green;
      when First_Green => NEXT_STATE <= Second_Green;
      when Second_Green => NEXT_STATE <= Third_Green;
      when Third_Green => NEXT_STATE <= Reset_State;
    end case;
  end process;
  process (CURRENT_STATE)
  begin
    R <= '0'; Y <= '0'; G <= '0';
    case CURRENT_STATE is
      when Reset_State | Stay_Red => R <= '1';
      when Yellow => Y <= '1';
      when First_Green to Third_Green => G <= '1';
    end case;
  end process;
end SYNTH;
```

14. (15 points) Create a VHDL architecture for a 3-to-8 decoder with an active-low enable input which uses a case statement to represent the functionality of the decoder. Create an entity and its accompanying architecture that represents a 4-to-16 decoder by using two instances of the `en_dec_328` entity. a. (6 points) 3-to-8 decoder architecture, b. (3 points) 4-to-8 decoder entity, c. (6 points) 4-to-8 decoder architecture

```
entity en_dec_328 is
  port (a : in std_logic_vector (2 downto 0);
        en : std_logic;
        d : out std_logic_vector (7 downto 0));
end entity en_dec_328;

architecture problem14 of en_dec_328 is
begin
    process (en, a)
    begin
        if (en = '1') then
            d <= "00000000";
        elsif (en = '0') then
            case (a) is
                when "000" => d <= "00000001";
                when "001" => d <= "00000010";
                when "010" => d <= "00000100";
                when "011" => d <= "00001000";
                when "100" => d <= "00010000";
                when "101" => d <= "00100000";
                when "110" => d <= "01000000";
                when "111" => d <= "10000000";
                when others => d <= "00000000";
            end case;
        end if;
    end process;
end architecture problem14;

entity en_dec_4216 is
    port(a : in std_logic_vector (3 downto 0);
         en : in std_logic;
         d : out std_logic_vector (15 downto 0));
end entity en_dec_4216;

architecture problem14b_c of en_dec_4216 is
    signal a_not : std_logic;
begin
    a_not <= '1' when a(3)='0' else
             '0' when a(3)='1';

    U0 : entity work.en_dec_328 port map (a => a (2 downto 0),
                                          en => a(3),
                                          d => d(7 downto 0));

    U1 : entity work.en_dec_328 port map (a => a (2 downto 0),
                                          en => a_not,
                                          d => d(15 downto 8));

end architecture problem14b_c;
```

15. (1 point) _**False**_ (True/False) Multiple assignments to a signal within a process can cause that signal to have multiple drivers.

16. (1 point). _**True**_(True/False) If you give a SystemVerilog simulator the same seed and the same testbench, it always produces the same results.

17. (1 point) _**True**_ (True/False) Assertions in System Verilog use negative logic.

18. (8 points An ARM Advanced High-performance Bus (AHB) has the following signals.

Signal	Width	Direction	Description
HCLK	1	Output	Clock
HADDR	21	Output	Address
HWRITE	1	Output	Write flag: 1=write, 0=read
HTRANS	2	Output	Transaction type: 2'b00=IDLE, 2'b10=NONSEQ
HWDATA	8	Output	Write data
HRDATA	8	Input	Read data

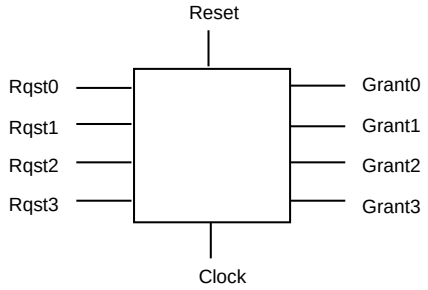
Create a class to encapsulate the AHB transactions. In this class constrain:

a. The address (HADDR) to be in the lower 5 addresses and upper 5 addresses each with probability 40% and the other addresses with probability 20%.
b. HTRANS to NONSEQ (HTRANS = 2'b10) and IDLE (HTRANS = 2'b00)..
c. All other AHB signals are randomized but unconstrained.

```
package mine;
  class AHB;
    rand bit HCLK;
    rand bit [20:0] HADDR;
    rand bit HWRITE;
    rand bit [1:0] HTRANS;
    rand bit [7:0] HWDATA;
    rand bit [7:0] HRDATA;

    constraint c
    {
      HADDR dist {[21'b000000000000000000000:21'b000000000000000000100]
      :/40, //lower 5 addresses
      [21'b111111111111111111011:21'b111111111111111111111]
      :/40, //upper 5 addresses
      [21'b000000000000000000101:21'b111111111111111111010]
      :/20  // all other addresses
      };
      HTRANS inside {2'b00, 2'b10};
    }
  endclass : AHB
endpackage : mine
```

19. (13 points) An arbiter is a circuit that allows at most one subsystem at a time to use a shared resource. A four-way arbiter is shown below. Each subsystem sets its request signal to 1 when it wants to use the resource. When the arbiter sets the grant signal to 1, the subsystem uses the resource. The subsystem sets its request back to 0 when it has finished, and waits for grant to be 0 before starting a subsequent request. While a subsystem is granted use of the resource, other requests must wait, rather than pre-empting the active subsystem. Subsystems are granted requests in order, starting with 0, then 1, 2, 3 and back to 0. A subsystem is skipped if it has no pending request.



As part of creating a SystemVerilog testbench for this device, an interface and a packet for randomization has been created. The requests are being driven by the request device modeled in VHDL below. Each req_dev has an associated test program. The test program is responsible for issuing a signal to the req_dev to reset it and providing a time parameter. After reset, the requesting device raises its request line high and keeps it high for the amount of time provided by the test program. Since req_dev works on the positive edge of the clock, the test program will provide inputs to it on the negative edge of the clock. Complete the test0 program so that it repeats the following process 100 times.

1) waits a random number of clock cycles to set the reset for reqdev to 1.
2) then provides a random number of cycles that reqdev will hold a request high.
3) sets the reset to 0 after one clock cycle
4) waits one more cycle
5) waits for req_dev to lower request to 0
6) waits one more clock cycle

```
interface arbiter_if(input bit clk);
  logic [3:0] REQUEST, GRANT, GEN_REQ;
  logic RESET;
  int REQ_TIME3, REQ_TIME2, REQ_TIME1, REQ_TIME0;
endinterface


module top;
  bit clk;
  always #5ns clk = ~clk;
  arbiter_if arbiterif(clk);
  arbiter u1 (.CLK (clk), .REQ (arbiterif.REQUEST),
              .GRANT (arbiterif.GRANT), RESET (arbiterif.RESET));
  req_dev u2 (.GEN_REQ (arbiterif.GEN_REQ[3]), .REQ_TIME (arbiterif.REQ_TIME3),
              .REQUEST (arbiterif.REQUEST[3]), .GRANT (arbiterif.GRANT[3]),
              .RESET (arbiterif.RESET),.CLK (clk));
  req_dev u3 (.GEN_REQ (arbiterif.GEN_REQ[2]), .REQ_TIME (arbiterif.REQ_TIME2),
              .REQUEST (arbiterif.REQUEST[2]), .GRANT (arbiterif.GRANT[2]),
              .RESET (arbiterif.RESET), .CLK (clk));
  req_dev u4 (.GEN_REQ (arbiterif.GEN_REQ[1]), .REQ_TIME (arbiterif.REQ_TIME1),
              .REQUEST (arbiterif.REQUEST[1]), .GRANT (arbiterif.GRANT[1]),
              .RESET (arbiterif.RESET),.CLK (clk));
  req_dev u5 (.GEN_REQ (arbiterif.GEN_REQ[0]), .REQ_TIME (arbiterif.REQ_TIME0),
```

```
              .REQUEST (arbiterif.REQUEST[0]), .GRANT (arbiterif.GRANT[0]),
              .RESET (arbiterif.RESET), .CLK (clk));
  test3 t1(arbiterif);
  test2 t2(arbiterif);
  test1 t3(arbiterif);
  test0 t4(arbiterif);
  testr t5(arbiterif);
endmodule : top

package mine;
  class Packet_g;
    // The random variables
    rand int generate_t, request_t;
    // Limit the values
    constraint c1 {generate_t < 300; generate_t > 100;
                   request_t < 101; request_t > 0;}
  endclass : Packet_g
endpackage : mine

entity REQ_DEV is
  port(GEN_REQ : in std_logic;
       REQ_TIME : in integer range 0 to 100;
       REQUEST : out std_logic;
       GRANT : in std_logic;
       RESET, CLK : in std_logic);
end REQ_DEV;


import mine::*;
module test0(arbiter_if arbif);
  Packet_g p;
  initial begin
    p = new();
    repeat(100)
    begin
      p.randomize();
      repeat(p.generate_t) @(negedge arbif.clk);
      //wait random nunber of cycles to set reset to 1 (1)
      arbif.RESET <= 1'b1;         //set reset to 1 (1)
      arbif.REQ_TIME1 <= p.request_t; // set random number of cycles to hold high (2)
      @(negedge arbif.clk);             //wait one clock cycle (3)
      arbif.RESET <= 1'b0;       //set the reset to 0 (3)
      @(negedge arbif.clk);             //wait one more clock cycle (4)
      @(negedge arbif.REQUEST[1]);      //wait for re_dev to lower request to 0 (5)
      @(negedge arbif.clk);             //wait one more clock cycle (6)
    end
  end
endmodule : test0
```

20. (6 points) Given the following constraints, what are the solution probabilities?

```
Class MemTrans;
  Rand bit x;
  Rand bit [1:0] y;
  Constraint c_xy
  {
     y inside {[x:3]};
     solve x before y;
  }
endclass
```

Solution	x	y	Probability
A	0	0	1/8
B	0	1	1/8
C	0	2	1/8
D	0	3	1/8
E	1	0	0
F	1	1	1/6
G	1	2	1/6
H	1	3	1/6

21. (8 points) For the class below, write two covergroups to collect coverage on the test plan requirements
    1. "All ALU opcodes must be tested". Assume that the opcodes are valid on the positive edge of signal clk.
    2. "operand1 shall take on the values maximum negative (-128), zero, and maximum positive (127)". Define a coverage bin for each of these values as well as a default bin.

```
typedef enum {ADD, SUM, MULT, DIV} opcode_e;
class Transaction;
  rand opcode_e opcode;
  rand byte operand1;
  rand byte operand2;
endclass : Transaction

Transaction tr;

covergroup cover1 @(posedge clk);
  coverpoint tr.opcode_e;
endgroup

covergroup cover2;
  coverpoint tr.operand1
  {
    bins min = {-128};
    bins zero = {0};
    bins max = {127};
    bins def = default;
  }
endgroup
```