*Name: Solution

*Registration: _____

Department of Computer Systems Engineering
University of Engineering & Technology Peshawar

Digital System Design

CSE 308

Midterm Examination Spring 2017

27 March 2017 Duration: 120 Minutes

Exam Rules

Please read carefully before proceeding.

1- This exam is OPEN books/notes, but CLOSED computers/cell phones.
2- No calculators of any kind are allowed.
3- Sharing of books, notes, and other materials during this exam is not permitted.
4- There are 4 problems in total. Some problems are harder than others. Answer the easy ones first to maximize your score.
5- Answer all problems on the problem sheet.
6- Questions will not be interpreted during the exam.
7- This exam booklet contains 11 pages, including this cover. Count them to be sure you have them all.

Problem 1	_____	(25 pts)
Problem 2	_____	(30 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(30 pts)
Exam Total	_____	(100 pts)

Good Luck!

Problem 1:

(a) The module below is in **explicit structural** form. Re-write the module in **behavioral** form. **(5 pts)**

```
module expl_str(x, y, a, b, c);
     input a, b, c;
     output x, y;
     wire a, b, c, x, y;
     wire na, nb, nc, t3, t5, t6;

     not n1(na, a);
     not n2(nb, b);
     not n3(nc, c);
     and a1(t3, na, b, c);
     and a2(t5, a, nb, c);
     and a3(t6, a, b, nc);
     or o1(x, t3, t6);
     or o2(y, a, t5);
endmodule

//SOLUTION

module behav(x, y, a, b, c);
     input a, b, c;
     output x, y;
     wire a, b, c;
     reg x, y;
     reg t3;

     always @( a or b or c ) t3 <= !a & b & c;
     always @( a or b or c or t3 ) x <= t3 | a & b & !c;
     //code below can be simplified to y <= a;
     always @( a or b or c ) y <= a | a & !b & c;
endmodule
```

(b) Convert the following **behavioral** code to **explicit structural** code. **(5 pts)**

```
module btos(x, a, b);
     input a, b;
     output x;
     wire a, b;
     reg x;

     always @( a or b ) if( a ) x = b; else x = ~b;
endmodule

//SOLUTION
```

If you don't see the logical function performed, draw a truth table. The function, $x = \overline{a \oplus b}$, can be performed by a primitive gate (**xnor**), a solution consisting of several other gates realizing the same function would also receive full credit.

```
module explicit(x, a, b);
     input a, b;
     output x;
     wire a, b;
     wire x;        //wire, not reg.

     xnor (x, a , b);
endmodule
```

(c) The module below is in **explicit structural** form, in which only primitive gates (and module instantiations) are used. Will the synthesis program synthesize exactly that arrangement and types of gates? Explain. **(5 pts)**

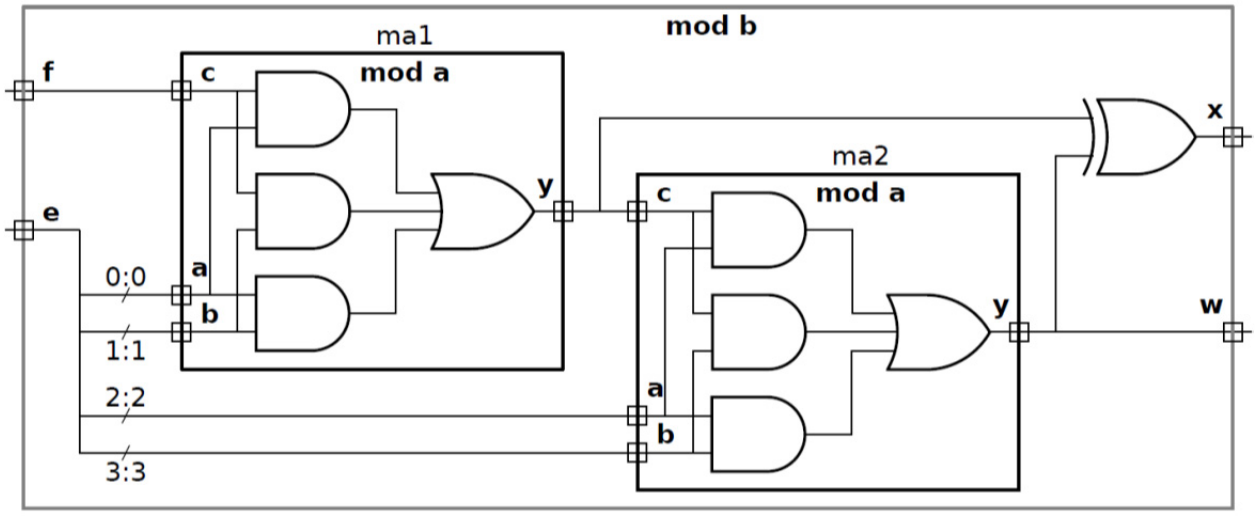
```
module bfa_structural( output sum, cout, input a, b, cin );
     wire term001, term010, term100, term111;
     wire ab, bc, ac;
     wire na, nb, nc;

     not n1( na, a);
     not n2( nb, b);
     not n3( nc, cin);
     and a1( term001, na, nb, cin);
     and a2( term010, na, b, nc);
     and a3( term100, a, nb, nc);
     and a4( term111, a, b, cin);
     or o1( sum, term001, term010, term100, term111);
     and a10( ab, a, b);
     and a11( bc, b, cin);
     and a12( ac, a, cin);
     or o2( cout, ab, bc, ac);
endmodule
```

```
//SOLUTION
```

No. Or at best, not necessarily. The synthesis program will map the gates above to the most appropriate gates in the target technology, it will then perform optimization. It's possible, for example, that the target technology does not have a three-input AND gate, so either two 2-input gates will be used, or maybe a 4-input AND gate will be used with one input tied to logic 1. Or perhaps, the technology has a special binary full adder primitive.

(d) Write a Verilog description of the hardware illustrated below. Base the names of ports, wires, and instances on labels in the illustration. The description can be **behavioral** or **structural**, but it must be synthesizable. **(10 pts)**




```
//SOLUTION
```

Grading Note: Many students chose to provide an explicit structural description, which is the most tedious descriptive style. In an explicit structural description `moda` *uses four primitive instantiations plus a declaration for three wires. As can be seen from the solution the implicit structural description is just one line.*

In many solutions for `modb` *the output of* `ma2` *was connected to an intermediate wire, and an* `assign` *statement was used to connect the wire to the module output. As can be seen from the solution, the* `ma2` *output can connect directly to the* `modb` *output.*

```
module moda(output wire y, input wire c, a, b);
     assign y = a && c || a && b || b && c;
endmodule

module modb(output wire x, w, input wire [3:0] e, input
wire f);
wire y1;

     moda ma1(y1,f,e[0],e[1]);
     moda ma2(w,y1,e[2],e[3]);

     assign x = y1 ^ w;
endmodule
```

Problem 2:

Draw a schematic of the hardware the synthesis system will synthesize for the following Verilog code examples. If flip flops are used, indicate if they are level triggered or edge triggered. Otherwise, don't worry about using the precisely correct gate or symbol, as long as it's functionally correct.

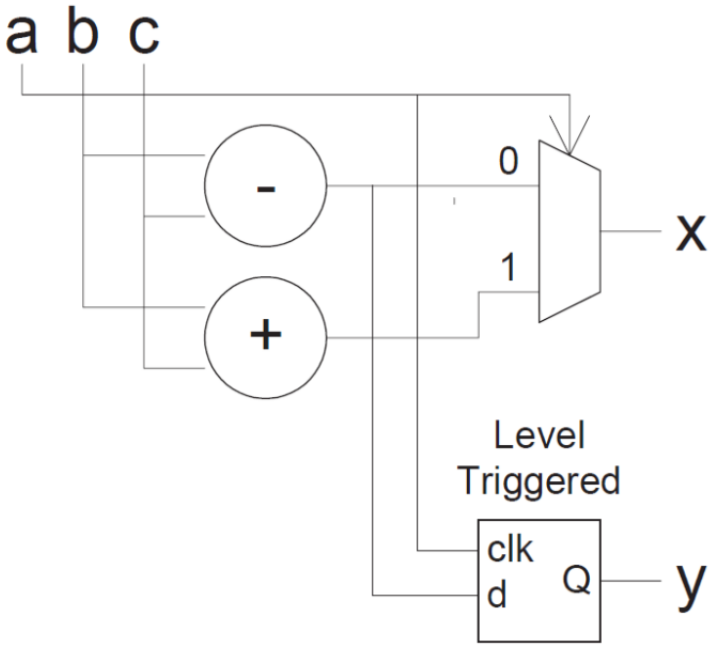
(a) Show an approximate schematic for the module below. In what synthesizable form is the Verilog description? **(15 pts)**
Hint: think about what form the code is in.

```
module mod_a(x, y, a, b, c);
     input a, b, c;
     output x, y;
     wire [7:0] b, c;
     reg [8:0] x, y;

     always @( a or b or c ) begin
          if( a ) begin
               x = b + c;
               y = b - c;
          end else begin
               x = b - c;
          end
     end
endmodule

//SOLUTION
```

combinational logic, level triggered flip-flops.



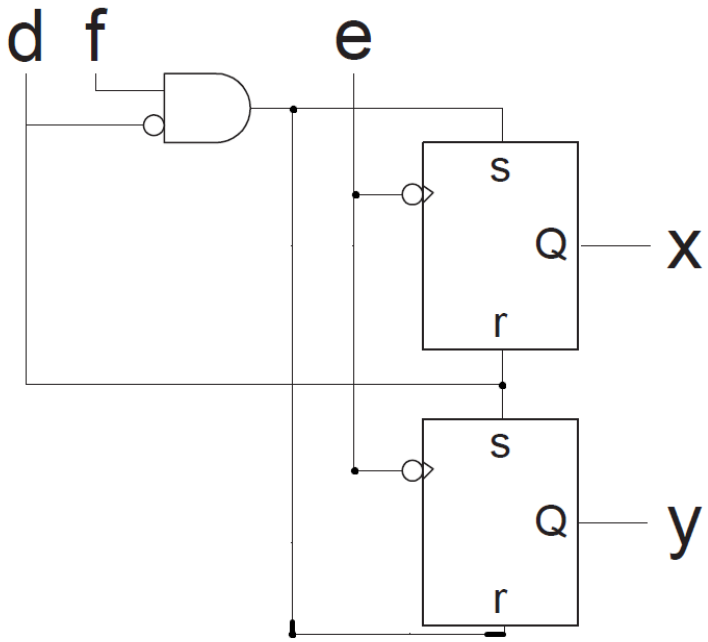
(b) Show an approximate schematic for the module below. What form is the description in? **(15 pts)**
Hint: think about what form the code is in.

```
module mod_b(x, y, d, e, f);
      input d, e, f;
      output x, y;
      reg x, y;

      always @( negedge e  )
            if( d ) begin
                  x = 0;
                  y = 1;
            end else if ( f ) begin
                  x = 1;
                  y = 0;
            end
endmodule

//SOLUTION
```

Edge triggered flip flops.



Problem 3:

(a) Show the values of the variables as indicated below: **(10 pts)**

```
module tryout();
     reg [15:0] a;
     reg [0:15] b;
     reg [3:0] e [3:0];
     reg [3:0] x1, x2, x3;
     reg x4;

     initial begin
         a = 16'h1234;
         b = 16'h1234;
         x1 = a[3:0];          //SOLUTION: x1 = 4'b0100
         x2 = b[0:3];          //SOLUTION: x2 = 4'b0001
         x3 = a[0:3] & b[0:3];//SOLUTION: x3 = 4'b0000
         x3 = a[4:7] | b[3:0];//SOLUTION: x3 = 4'b0011
         x3 = a[4:7] ^ b[7:4];//SOLUTION: x3 = 4'b0001
         x4 = & a[8:11];       //SOLUTION: x4 = 1'b0
         x4 = & a[11:8];       //SOLUTION: x4 = 1'b0
         x4 = | b[12:15];      //SOLUTION: x4 = 1'b1
         e = 16'h1234;
         e[0] = e[0] + 4'hf;   //SOLUTION: e = 16'h1233
         e = 16'h1234;
         e[0][0] = e[0][0] + 1'b1; //SOLUTION: e = 16'h1235
     end
endmodule
```

(b) Do the two code fragments below do the same thing? If not, how do they differ? **(5 pts)**

```
// Fragment A.
if ( foo > bar ) x = x + 1; else y = y + 1;

// Fragment B.
case ( foo > bar )
     1: x = x + 1;
     default: y = y + 1;
endcase

//SOLUTION
```

They do not differ.

Problem 4:

(a) Write the hardware description of an 8-bit register with shift-left and shift-right modes of operation. The suggested skeleton file has been written below: **(10 pts)**

```
module slsr(sl, sr, din, clk, reset, Q);
     input sl, sr, din, clk, reset;
     output [7:0] Q;

     //WRITE YOUR CODE HERE
     reg [7:0] Q;

     always @ (posedge clk) begin
          if (~reset) begin
               if (sl) begin
                    Q <= {Q[6:0],din};
               end
               else if (sr) begin
                    Q <= {din, Q[7:1]};
               end
          end
     end
     always @ (posedge reset) begin
          Q<= 8'b00000000;
     end
endmodule
```

(b) Write the hardware description of a 4-bit mod-13 counter. The suggested skeleton file has been written below: **(10 pts)**

```
module mod13Cntr(clk, reset, Q);
     input clk, reset;
     output [3:0] Q;

     //WRITE YOUR CODE HERE
     reg [3:0] Q;

     always @ (posedge clk) begin
          if (~reset) begin
               if (Q == 4'b1100) begin
                    Q <= 4'b0;
               end
               else begin
                    Q <= Q+1;
               end
          end
     end
     always @ (posedge reset) begin
          Q <= 4'b0000;
     end
endmodule
```

(c) Write the hardware description of a 4-bit adder/subtractor. An adder/subtractor is a piece of hardware that can give the result of addition or subtraction of the two numbers based on a control signal. Assume that the numbers are in 2's complement notation. Please keep in mind that this is a combinatorial circuit. The suggested skeleton file to start with is given below: **(10 pts)**

```
module addsub (a, b, sel, res);
     input [3:0] a, b;
     input sel;
     output [3:0] res;

     //WRITE YOUR CODE HERE
     wire [3:0] res;

     assign res = (sel)? A + B : A - B;
endmodule
```

〈This page is intentionally left blank. This page can be used for scratch work or as extra space. If you write work here that you want me to grade, be sure to clearly indicate which problem(s) the work corresponds to!〉