Name _____

Computer Organization
EE 3755
Midterm Examination

22 March 2002, 12:40-13:30 CST

Problem 1	_____	(15 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(40 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: The incomplete module below is part of a multi-level carry-lookahead adder similar to the one in Homework 3. Add code for module outputs `P` and `G` and for connections `ci0`, `ci1`, and `ci2`. Outputs `P` and `G` are propagate and generate signals for higher-level adders and connections `ci0`, `ci1`, and `ci2` are carry-in values for the lower-level adders.

These signals must be generated by *carry lookahead logic*. Structural or procedural code (or both) can be used. (15 pts)

```
module adder9(sum,P,G,a,b,cin);
   input [8:0] a, b;
   input       cin;
   output [8:0] sum;
   output       P, G;


   adder3 a0(sum[2:0],p0,g0,a[2:0],b[2:0],ci0);
   adder3 a1(sum[5:3],p1,g1,a[5:3],b[5:3],ci1);
   adder3 a2(sum[8:6],p2,g2,a[8:6],b[8:6],ci2);









endmodule
```

Problem 2: Module `the_hard_way` performs a common operation the hard way. (15 pts)

```
module the_hard_way(x,a,b);
   input [3:0] a, b;                    reg         x;
   output      x;                       integer     i;

   always @( a or b )
     begin
        x = 0;
        for(i=0; i<4; i=i+1) x = x | ( a[i] ^ b[i] );
        x = ~x;
     end

endmodule
```

(*a*) Show the hardware that would be synthesized for this module **without** optimization. Be sure to show module ports and registers, if any.

(*b*) Which Verilog operator performs the same function as the module?

Problem 3: Show the hardware that would be synthesized for the module below **without** optimization. Be sure to show the module ports and registers, if any. (15 pts)

```
module stuff(x,c,s,r,a,b);
   input [7:0] a, b;          input [1:0] s;
   input       r;             output      x, c;
   reg [7:0]   x, c;

   always @( posedge r ) begin

      case( s )
        0: x = a + b;
        1: x = a - b;
        2: x = a & b;
      endcase

     if( a - b < a & b ) x = x + b; else c = 0;

     c = c + 1;

   end

endmodule
```

Problem 4: The output of the first module below is the sum of two quotients; the values on the inputs and output are integers. Complete the second module so that it performs the same operation as the first one, but produces an output after several clock cycles. The second module should use the instantiated divider. This divider produces an output two clock cycles after its inputs change (it does not use `start` or `ready` signals). The second module should be coded so that when `n` or `b` changes the `x` output is set to the correct value within several cycles and the correct value is held until `n` or `b` change again. *Hint: A state variable can be a simple counter.*

(15 pts)

```
module qsums_comb(x,n,b);
   input [31:0] n, b;
   output [31:0] x;

   assign x = ( n/b ) + ( n/(b+1) );

endmodule

module qsums_seq(x,n,b,clk);
   input [31:0] n, b;
   input        clk;
   output [31:0] x;   // Output should remain valid if n and b don't change.

   // Use only one divider.
   div your_divider(q,d1,d2,clk); // q = d1/d2 after 2 cycles.









endmodule
```

Problem 5: Answer each question below.

(*a*) Fill in the values for `x` below, any radix will do. (8 pts)

```
module misc();
   reg [7:0] x;
   reg [3:0] a, b, c;
   initial begin

      a = 4'h5;
      x = { 2'd1, 2'o0, a };       // x =

      a = 4'b1010;
      b = 4'b0001;

      x = a & b;                   // x =

      x = a && b;                  // x =

      x = a < b ? a | b : a ^ b;  // x =

      a = -3;  b = 5;

      x = a < b;                   // x =

      c = 0;
      // Be careful.
      x = a == b == c;        // Explain this one:  x =

   end

endmodule
```

(*b*) A single-level (flat) carry lookahead adder can produce a sum in just six gate delays, no matter the size of the numbers being added. Besides cost, why is the delay of six not a good measure of actual delay? (8 pts)

(*c*) Convert 0.75 and 0.375 to normalized binary scientific notation and then add them as floating-point hardware would. (The numbers do **not** have to be converted to IEEE 754.)(8 pts)

(*d*) Exponents in the IEEE 754 format are biased. What does that mean? In what other way might the exponent be represented? (8 pts)

(*e*) The ALU in the Patterson & Hennessey text and presented in class realizes three operations, addition, subtraction, and set less than (`slt`), all sharing an adder. How are the subtraction and set less than operations performed using the adder? (8 pts)