Name ______________________________

Digital Design Using Verilog
EE 4702-1
Final Examination

9 May 2001 7:30-9:30 CDT

Problem 1	________	(15 pts)
Problem 2	________	(18 pts)
Problem 3	________	(17 pts)
Problem 4	________	(18 pts)
Problem 5	________	(12 pts)
Problem 6	________	(20 pts)
Exam Total	________	(100 pts)

Alias ______________________________

Good Luck!

Problem 1: The module below is in an explicit structural form.

(*a*) Re-write the module in behavioral form. The delays can be assumed to be pipeline delays. (10 pts)

(*b*) What is the difference between pipeline and inertial delays? Which kind of delay is used in your solution to the problem above? (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 #1 a1(t3,na,b,c);
   and a2(t5,a,nb,c);
   and a3(t6,a,b,nc);
   or o1(x,t3,t6);
   or #3 o2(y,a,t5);

endmodule
```

```
module behavioral(x,y,a,b,c);
   input a, b, c;
   output x, y;
   // Solution here.  Don't forget types for ports!









endmodule
```

Problem 2: The module below sets output `rot` to the number of times that input `a` must be rotated (end-around shifted) to obtain the value on input `b`, or to 32 if `a` is not a rotated version of `b`.

(*a*) Write a testbench module that tests `rots` with input pairs `a`=0,`b`=0; `a`=0,`b`=1; `a`=0,`b`=2; and `a`=0,`b`=3. (The `rot` output should be zero for the first pair and 32 for the others.) The testbench should include an integer `err` and set it to the number of incorrect outputs.

It is important that the testbench makes correct use of `ready` and `start`. (Part of the problem is determining just what is "correct use.") The testbench should use `ready` rather than assumed timing. Also, test only a single instance of `rots` and don't forget the clock. (18 pts)

```
module rots(ready, rot, start, a, b, clk);
   input a, b, start, clk;        output ready, rot;
   reg         ready;             wire [31:0] a, b;
   reg [5:0]   rot;               wire        start, clk;
   reg [31:0]  acpy;
   initial rot = 0;
   always @( posedge clk ) begin
      ready = 1;   while ( !start ) @( posedge clk );
      ready = 0;   while (  start ) @( posedge clk );
      rot = 0;  acpy = a;
      while (  acpy != b  &&  rot < 32  ) @( posedge clk ) begin
         acpy = { acpy[30:0], acpy[31] };
         if ( acpy == a ) rot = 32; else rot = rot + 1;
      end
   end
endmodule


module testrot();
 integer err;
```

```
endmodule
```

Problem 3: Convert the `rots` module (repeated below) to synthesizable Form 2 (edge-triggered flip-flops). Do not change the ports or what it does. In particular, `ready` and `start` must be used the same way. Ignore reset. (17 pts)

```
module rots(ready, rot, start, a, b, clk);
   input a, b, start, clk;       output ready, rot;
   reg        ready;             wire [31:0] a, b;
   reg [5:0]  rot;               wire        start, clk;
   reg [31:0] acpy;
   initial rot = 0;
   always @( posedge clk ) begin
      ready = 1;   while ( !start ) @( posedge clk );
      ready = 0;   while (  start ) @( posedge clk );
      rot = 0;  acpy = a;
      while (  acpy != b  &&  rot < 32  ) @( posedge clk ) begin
         acpy = { acpy[30:0], acpy[31] };
         if ( acpy == a ) rot = 32; else rot = rot + 1;
      end
   end
endmodule
```

```
module rots(ready, rot, start, a, b, clk);
   input a, b, start, clk;
   output ready, rot;  // Don't forget port types and other declarations.









         acpy = { acpy[30:0], acpy[31] };
         if ( acpy == a ) rot = 32; else rot = rot + 1;





endmodule
```

Problem 4: Two synthesizable descriptions appear below.

(*a*) In what synthesizable form is the Verilog description below? (2 pts)

(*b*) Draw a schematic showing the approximate RTL-level description generated by a synthesis program like Leonardo. (7 pts)

```
module whatsyna(x, y, z, a, b, op);
   input a, b, op;
   output x, y, z;
   wire [7:0] a, b;
   wire [1:0] op;
   reg [7:0]  x, y, z;

   always @( op or a or b ) begin

      if ( a == 0 ) y = b;

      if ( a < b ) z = a; else z = b;

      case ( op )
        0: x = a + b;
        1: x = a;
        2: x = b;
      endcase

   end

endmodule
```

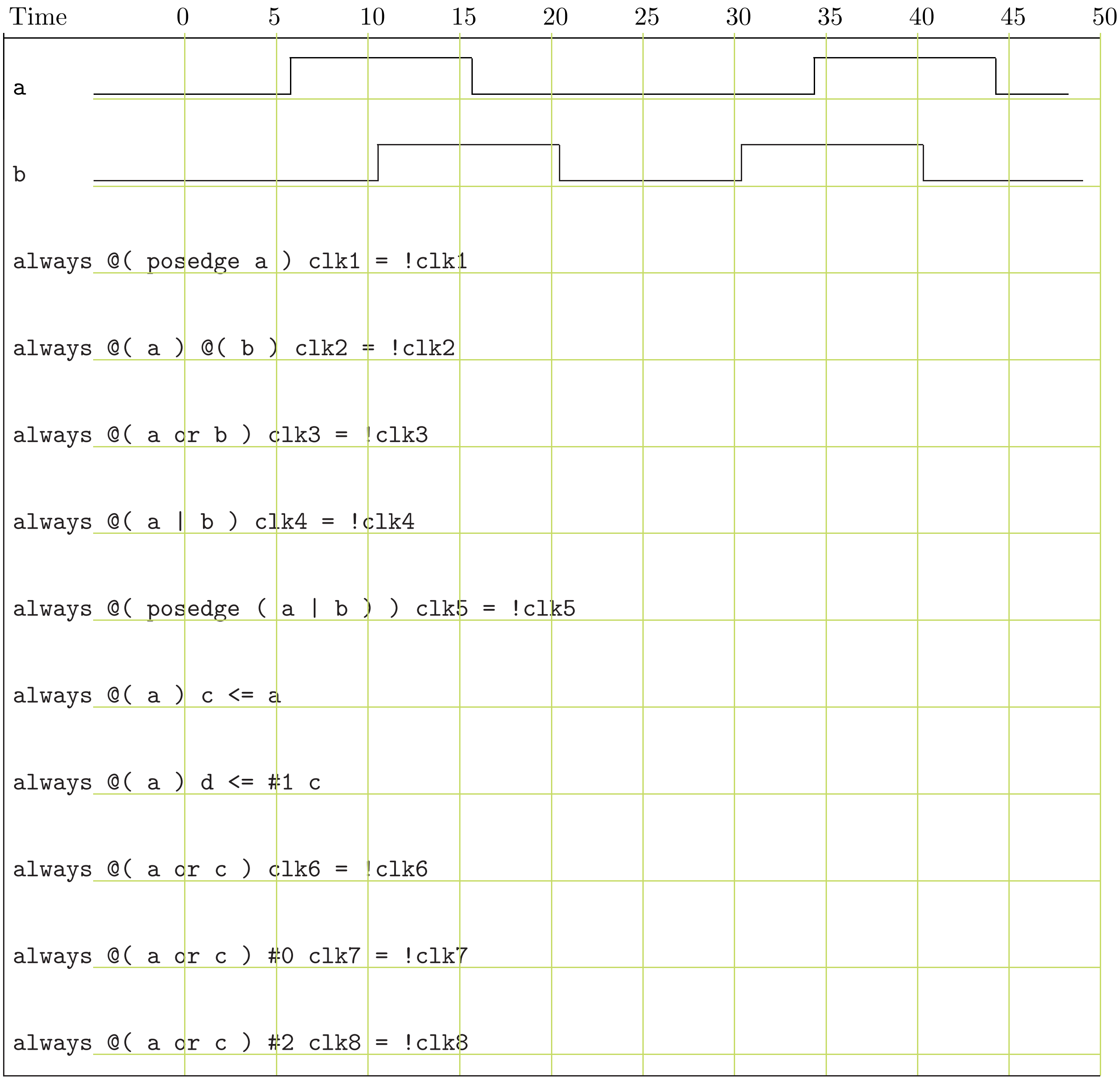
Problem 4, continued:

(*c*) In what synthesizable form is the Verilog description below? (2 pts)

(*d*) Draw a schematic showing the approximate RTL-level description generated by a synthesis program like Leonardo. (7 pts)

```
module whatsyn2(sum, nibbles, a, b, c);
   input nibbles, a, b, c;
   output sum;
   wire [15:0] nibbles;
   wire        a, b, c;
   reg [6:0]   sum;

   reg [15:0]  n2;
   reg         last_c;
   integer     i;

   always @( posedge a or negedge b )
     if ( !b ) begin

        sum = 0;

     end else begin

        if ( c != last_c ) begin
           n2 = nibbles;
           for ( i=0; i < 4; i = i + 1 ) begin
              sum = sum + n2[3:0];
              n2 = n2 >> 4;
           end
        end
        last_c = c;

     end

endmodule
```

Problem 5: In the diagram below `c`, `d`, and identifiers starting with `clk` are all initialized to zero. Complete the timing diagram. (12 pts)



```
always @( posedge a ) clk1 = !clk1
always @( a ) @( b ) clk2 = !clk2
always @( a or b ) clk3 = !clk3
always @( a | b ) clk4 = !clk4
always @( posedge ( a | b ) ) clk5 = !clk5
always @( a ) c <= a
always @( a ) d <= #1 c
always @( a or c ) clk6 = !clk6
always @( a or c ) #0 clk7 = !clk7
always @( a or c ) #2 clk8 = !clk8
```

Problem 6: Answer each question below.

(*a*) The code below, based on the Homework 3 solution, simulates properly before synthesis but in the post-synthesis simulation the testbench reports an incorrect beep time.

What goes wrong? Fix the problem without modifying the code below the indicated line. *Hint: The beep can start (and stop) at a slightly different time than the code below.* (5 pts)

```
module beepprob(beep, clk);
   input clk;
   output beep;

   assign beep = | beep_timer;


   // DO NOT MODIFY CODE BELOW THIS LINE.
   always @( posedge clk ) begin
      // Lots of stuff;

      if ( beep_timer ) beep_timer = beep_timer - 1;

   end

endmodule
```

(*b*) Describe something that a parameter can be used for that an ordinary input port cannot and something that an input port can be used for that a parameter cannot. (5 pts) (*c*) What is the difference between `case`, `casex`, and `casez`? (5 pts)

(*d*) Explain how each of the three statements below behave differently with unknown values. In particular, explain what has to be unknown and how the results of each statement is different. (5 pts)

```
m1 = a > b ? c : d;

if ( a > b ) m2 = c; else m2 = d;

case ( a > b )
  1: m3 = c;
  default: m3 = d;
endcase
```