March 26,2004

EE3755 EXAM 2:

Do not turn over the page till I say so.
Some problems are very easy so do not spend
Too much time on them.
If you think a problem is difficult, try to solve
The easy ones First.

Name:

SSN:

Problem 01(10pts): 05mins
Problem 02(07pts): 01mins
Problem 03(10pts): 05mins
Problem 04(10pts): 05mins
Problem 05(06pts): 05mins
Problem 06(10pts): 05mins
Problem 07(20pts): 12mins
Problem 08(15pts): 05mins
Problem 09(10pts): 02mins
Bonus 10(2pts): 01mins

Total 100pts 46mins

Solution

Problem 1: Fill the output of ripple carry counter after time 150.(q value and time)// (10pts. 5mins)

```
module ripple_carry_counter(q,clk,reset);

output [3:0]  q;
input clk, reset;
T_FF tff0(q[0],clk,reset);
T_FF tff1(q[1],q[0],reset);
T_FF tff2(q[2],q[1],reset);
T_FF tff3(q[3],q[2],reset);

endmodule

module T_FF(q,clk,reset);

output q;
input clk,reset;
wire d;

D_FF dffo(q,d,clk,reset);
not n1(d,q); //not is a Verilog-provided primitive. case sensitive
endmodule

//module D_FF with synchronous reset
module D_FF(q,d,clk,reset);
output q;
input d,clk,reset;
reg q;

always @(posedge reset or negedge clk)

if (reset)
        q = 1'b0;
// module D_FF with synchronous reset
else
        q = d;
endmodule

// Test bench or Stimulus Block
module stimulus;

reg clk;
reg reset;
wire [3:0] q;

ripple_carry_counter r1(q, clk, reset);

// Control the clk signal that drives the design block.Cycle time =10
initial
        clk = 1'b0; //set clk to 0
always
        #5 clk = ~clk; // toggle clk every 5 time units

//Control the reset signal that drives the design block
//rest is asserted from 0 to 20 and from 200 to 220.
```

```
initial
begin
        reset = 1'b1;
        #15 reset = 1'b0;
        #180 reset = 1'b1;
        #10 reset = 1'b0;
        #20 $finish; //terminate the simulation
end
// Monitor the outputs
initial
        $monitor($time, "Output q = %d",q);
endmodule
```

```
//Solution
//Output of the simulation
//#  0 Output q =0
//# 20 Output q =1
//# 30 Output q =2
//# 40 Output q =3
//# 50 Output q =4
//# 60 Output q =5
//# 70 Output q =6
//# 80 Output q =7
//# 90 Output q =8
//#100 Output q =9
//#110 Output q =10
//#120 Output q =11
//#130 Output q =12
//#140 Output q =13
//#150 Output q =14
//#160 Output q =  15          // fill  q
//#170 Output q =  0           // fill   q
//#180 Output q =  1           // fill   q
//#190 Output q =  2           // fill    q
//#195 Output q =0          // fill time
//# 210Output q =1          // fill time
//#220  Output q =2             // fill q
```

problem 2: Complete the table. (7pts. 1 Mins)

```
     a     b     |   a % b
   // """"""""""""""""""""
   // 3     5     |  3
   // 5     3     |  2
   // 7     7     |  0

      a     b     |   a  && b
   // """"""""""""""""""""
   // 000  000   |  0
   // 000  001   |  0
   // 010  101   |  1
   // 100  110   |  1
```

problem 3: Convert the module below to an implicit form.(10pts. 5mins)

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

//Solution comes here

```
module bfa_implicit(sum,cout,a,b,cin);
   input a,b,cin;
   output sum,cout;
   assign sum =

         ~a & ~b &  cin |

         ~a &  b & ~cin |

          a & ~b & ~cin |

          a &  b &  cin;

   assign cout = a & b | b & cin | a & cin;

endmodule
```

Problem 4: Fill in the values for x below.(10pts. 5mins)

```
##################################################
module opp_example();
     reg [7:0] x,a;
     reg [3:0] b,c,;
     reg [2:0] s;
initial begin
       a = 19;
       x = a < 10 ? 10 : a < 20 ? 20 : 30;   //x = 20
    b= 4'h6;
    c= 4'b0001;
    x = b & c;                   // x  =   0
    x = b && c;                   // x  =  1
    b  = 4'b0001;
    x = { 3'd2,1'h1,b};              // x =  01010001

       x = 8'b00001011;
    s = 2;
    x = x << s;                  // x = 00101100
    b = -3;
    c =  1;
    x  = b > c ;                  // Explain this one , why?
                                                      //  x = 1, unsigned number
    b = -1;
    c = 3;
    x = b <= c;                  // Explain this one, why?
                                 // x =  0 unsigned number
    b = -1;
    c = 15;
    x = b === c;                  // x =  1

        b = 2;
    c = 3;
    x = b == c;                  //x =  0

    end
endmodule;
```

problem 5: Draw the figure for module who_am_I(6pts. 3mins)

```
module who_am_I(x,a,b);
        output x;
        input a,b;

        wire na, nb, na_b, a_nb;

        not n1(na,a);
        not n2(nb,b);


        and a1(na_b,na,b);
        and a2(a_nb,a,nb);

        or o1(x,na_b,a_nb);
endmodule
```

(a) Draw the figure for module who_am_I

(b) what kind of logic is that and what will be verilog code for that?

exclusive or;   xor(x,a,b);

problem 6: Complete the module , think about the input size(10 pts. 5mins)

```
module signed_adder_different_input_size(sum,overflow,a,b);
  input [7:0] a;
  input [3:0] b;
  output [7:0] sum;
  output      overflow;

  wire   sa = a[7];
  wire   sb = b[3];
  wire   ssum = sum[7];
  assign      sum = a + { sb ? 4'b1111 : 4'b0, b };                //Fill code here.
  assign overflow = sa != sb   ? 0 :
                    sb == ssum ? 0 : 1;                  //Fill code here.
  endmodule
```

Problem 7: Population Counter: (20pts. 12 mins)

```
module pop(p,a,clk);
   input [31:0] a;
   input        clk;
   output       p;

   reg [5:0]    p;
   reg [31:0]   acopy;
   reg [5:0]    pcopy;

   initial acopy = 0;
   initial pcopy = 0;
   always @( posedge clk )
     begin

        if( acopy == 0 )
          begin
             p = pcopy;
             pcopy = 0;
             acopy = a;
          end
        else
          begin
             pcopy = pcopy + acopy[0];
             acopy = acopy >> 1;
          end


     end

endmodule
```

(a) When will be earliest time we know the correct output p?(how many clock cycles) and what will be input at that time?(3pts) clock 1, all 0.

(b) What will be the worst case input? (which input will produce the correct output p at the latest time).(3pts) any input with 1 at bit position 31.

(c) What is the problem of the above module?(hint: think about (a) and (b)).(2pts)
Don't know when the correct output is.

(d) If we divide the above module by 2, we roughly have a speed up of 2 (see the table :actually 32/17).
If we divide the above module by 4 , we roughly have a speed up of 4 (see the table : actually 32/10).
Assuming we use an adder unit which will add only 2 numbers at a time, we need 3 adder units when we divide the above module by 4.(see the table : 3 adders with 2 stages(levels))

# of counting units	# of adder stages(levels)	# of adders	Processing time(cycles)
8	3	7	7
4	2	3	10
2	1	1	17
1	0	0	32

Table 1. Multi-unit population counter.

(d)-1. How many number of adders do we need when we have 16 counting units? (2pts) 15

(d)-2. What will be the processing time when we have 16 counting units? (2pts) 6

(e) Write a verilog code to finish the job 2 times faster than the above one.(assume adding the two intermediate p values takes 1 clock cycle and it should produce correct output at least after 20 cycles).(8 pts)

```
module add(p,a,b,clk);
input [5:0] a,b;
input clk;
output p;
reg [5:0] p;
always @(posedge clk)
begin
     p = a+b;
     end
endmodule


 module pop16(p,a,clk);
   input [15:0] a;
   input        clk;

   output       p;
   reg [5:0]    p;
   reg [15:0]   acopy;
   reg [5:0]    pcopy;
   initial acopy = 0;
   initial pcopy = 0;
   always @( posedge clk )
     begin
        if( acopy == 0 )
          begin
             p = pcopy;
             pcopy = 0;
             acopy = a;
          end
        else
          begin
             pcopy = pcopy + acopy[0];

             acopy = acopy >> 1;
          end
     end
endmodule

module popfast(p,a,clk)
wire  [5:0] p1,p2;
output  p;
input [31:0] a;
reg [5:0] p;
 pop16 pa(p1,a[31:16],clk);

 pop16 pb(p2,a[15:0],clk);

 add pc(p,p1,p2,clk);

endmodule
```

problem 8: Write a verilog program to convert a Packed BCD number to ASCII value.(15pts. 5mins)

```
     format: Packed BCD(8bits)
                              A        B
                 bit position: 0111    0100
                   decimal      7        4
        there are 2 numbers in an 8 bits input.
        each number is 4 bits long.
     for example:
                  input X   0000 0001
                  output Y  = 8'd49   //ASCII value for 1;
                  output Z = 8'd48  //ASCII value for 0;
 module packed_bcd_2_ascii(Z,Y,X)
 input [7:0] X;
 output [7:0] Z,Y;
  // code comes here.
           assign  Z = X[7:4] + 48;
           assign  Y  = X[3:0] + 48;
endmodule
```

problem 9: Write a verilog program to compare 4bits numbers.(10pts. 2mins)

You are given a module.
The module name is mag_comp_slice.
mag_comp_slice(eqout,smallout,bigout,a,b,eqin,smallin,begin);// code for this is omitted.
you may use mag_comp_slice module which implements the fig.1.

Use mag_comp_slice module and write 4bits comparator.

```
module fourbit_comp(eq,big,small,a,b);
output eq,big,small; // 1bits
input [3:0] a,b;
reg  eq,big,small;
wire eq1,gq2,eq3, big1,big2,big3,small1,small2,small3;

                mag_comp_slice(eq1,big1,small1,a[3],b[3],1,0,0);
                mag_comp_slice(eq2,big2,small2,a[2],b[2],eq1,big1,small1);
                mag_comp_slice(eq3,big3,small3,a[1],b[1],eq2,big2,small2);
                mag_comp_slice(eq,big,small,a[0],b[0],eq3,big3,small3);


                endmodule
```

endmodule

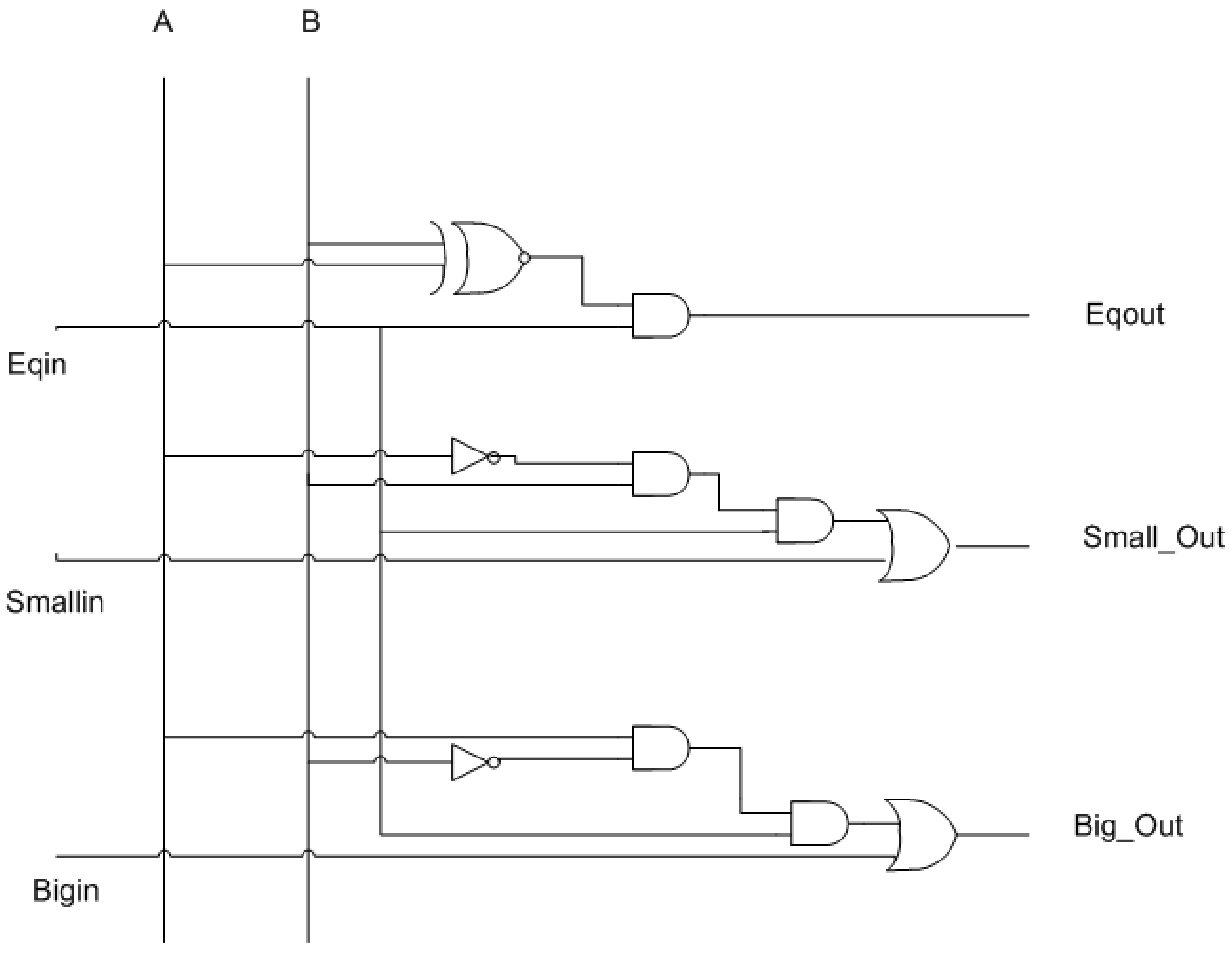

Fig.1 mag_comp_slice

problem 10 : Estimation time to solve match what you really spent?(2pts. 1mins)