EE 3755

Verilog Handout #3

```
/// LSU EE 3755 -- Fall 2001 -- Computer Organization
//
/// Verilog Notes 3 -- Delay

/// Contents

 /// Delays
 /// Ripple Adder and Delays

/// References

// :P:   Palnitkar, "Verilog HDL"
// :Q:   Qualis, "Verilog HDL Quick Reference Card Revision 1.0"
// :H:   Hyde, "Handbook on Verilog HDL"
// :LRM: IEEE, Verilog Language Reference Manual  (Hawaii Section Numbering)
// :PH:  Patterson & Hennessy, "Computer Organization & Design"


////////////////////////////////////////////////////////////////////////////////
/// Delays

// :P: 6.2, 6.2.2, 6.2.3
// :H: Not covered for continuous assignments.
// :LRM: 6.1

// A /delay/ is a specification of elapsed time, the amount of
// time the simulator is supposed to wait.

// The simulator keeps track of simulated time, measured in cycles.
// This is unlike conventional languages.

// The default delay is zero.
//
// If delays are not specified things happen instantly.
//
// The default delay for a gate is zero.
//
// The default delay for an assign is zero.
//
// The default delay for an assign with a really complex expression is zero.
//
// The default delay for a module containing a zillion gates is zero.
//
// Zero is NOT the same as one.
//

// Delays can be specified in a variety of ways.
//
//   The following will be covered in class:
//
//      Delays in wires and assigns (this set).
//      Delays in behavioral statements (later this semester).
//
//   See "delays" module, below, for further description.


// :Example:
//
// A module that just passes the signal through, like wire.  Suppose
// at t=5 input a changes from 0 to 1, when does x change?

module no_delays(x,a);
  input a;
  output x;

  assign x = a;
  wire   w = x;
```

```
endmodule

// Ans: x changes at 5 + 0 = 5.


// :Example:
//
// A module that just passes the signal through unchanged, though it
// passes through two inverters.  Suppose at t=5 input a changes from
// 0 to 1, when does x change?

module no_delays2(x,a);
   input a;
   output x;

   wire   b;

   not n1(b,a);
   not n2(x,b);

endmodule

// Ans: x changes at 5.


// :Example:
//
// Illustration and explanation of delays in assigns and wires.

module delays(x,a);
   input a;
   output x;

   // Delay (For nets.)
   //
   wire #7   w = a;
   //
   // w gets value of a 7 cycles after each change in a (a must hold
   // steady for at least 7 cycles).

   // Delay (For continuous assignments.)
   //
   assign #3 x = w;
   //
   // x gets value of "w" 3 cycles after each change in "w" (w must
   // hold steady for at least 3 cycles).

endmodule


////////////////////////////////////////////////////////////////////////////////
/// Ripple Adder and Delays


// :Example:
//
// The binary full adder presented earlier.  There are no delays 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


// :Example:
//
// The same BFA with delays.
// Suppose "a" changes at t=5 and "b" changes at t=20.
// When do sum and cout change?

module bfa_implicit_d(sum,cout,a,b,cin);
   input a,b,cin;
   output sum,cout;

   assign #3 sum =
          ~a & ~b &  cin |
          ~a &  b & ~cin |
           a & ~b & ~cin |
           a &  b &  cin;

   assign #2 cout = a & b | b & cin | a & cin;

endmodule

// Ans: sum: t=8 and t=23;  cout t=7 and t=22.


 /// More information
//
// If a changes at t=5 and b changes at t=6 the situation is
// different because there are two changes in "sum" within the
// 3-cycle delay.


// :Example:
//
// A 4-bit ripple adder constructed using the undelayed BFA.
// If a changes at t=5 when does sum change?

module ripple_4(sum,cout,a,b);
   input [3:0] a, b;
   output [3:0] sum;
   output       cout;

   wire         c0, c1, c2;

   bfa_implicit bfa0(sum[0],c0,a[0],b[0],1'b0);
   bfa_implicit bfa1(sum[1],c1,a[1],b[1],c0);
   bfa_implicit bfa2(sum[2],c2,a[2],b[2],c1);
   bfa_implicit bfa3(sum[3],cout,a[3],b[3],c2);

endmodule

// Ans: I'm sure everyone got it.


// :Example:
//
// A 4-bit ripple adder constructed using the BFA with delays.
// Suppose at t=0 a=0 and b=4'b1111
// If a changes to 1 at t=100 when does sum stop changing?

module ripple_4_d(sum,cout,a,b);
   input [3:0] a, b;
   output [3:0] sum;
   output       cout;
```

```
   wire         c0, c1, c2;

   bfa_implicit_d bfa0(sum[0],c0,a[0],b[0],1'b0);
   bfa_implicit_d bfa1(sum[1],c1,a[1],b[1],c0);
   bfa_implicit_d bfa2(sum[2],c2,a[2],b[2],c1);
   bfa_implicit_d bfa3(sum[3],cout,a[3],b[3],c2);

endmodule

// Ans t=109  (Not checked)


// :Example:
//
// A 4-bit adder using operators and no delays. This would be
// a better way of coding a 4-bit adder in Verilog, but it's
// not a good way to demonstrate the low speed of a ripple adder
// to a computer organization class.

module another_adder(sum,cout,a,b);
   input [3:0] a, b;
   output [3:0] sum;
   output       cout;

   wire [4:0]   sum_with_carry = a + b;
   assign       sum = sum_with_carry[3:0];
   assign       cout = sum_with_carry[4];

endmodule


// :Example:
//
// This is not really an example and students are not expected
// to understand it.  This is a testbench for the adder, it
// instantiates the adders, generates inputs for them, and checks
// for the correct output.

module demoadd();

   wire [4:0] sum1, sum2, sum3;
   reg [4:0] shadow_sum;
   reg [3:0]  a, b;
   integer    i;

   ripple_4 adder1(sum1[3:0],sum1[4],a,b);
   ripple_4_d adder2(sum2[3:0],sum2[4],a,b);
   another_adder adder3(sum3[3:0],sum3[4],a,b);

   task check_sum;
      input [4:0] s;
      input [79:0] name;

      if( s != shadow_sum ) begin
         $display("Wrong sum in %s: %d + %d = %d != %d\n",
                  name, a, b, shadow_sum, s);
         $stop;
         end

   endtask

   initial begin

      for(i=0; i<255; i=i+1) begin
         a = i[3:0];
         b = i[7:4];
         shadow_sum = a + b;
```

```
        #10;
        check_sum(sum1,"Ripple");
        check_sum(sum2,"Ripple_d");
        check_sum(sum3,"Operator");
      end

      $display("Tests completed.");

   end

endmodule
```