EE 3755

Verilog handout #1

```
/// LSU EE 3755 -- Fall 2001 -- Computer Organization
//
/// Verilog Notes 1 -- Verilog Basics

/// Contents

 /// Module
 /// Module Ports
 /// Wire
 /// Gates
 /// Module Instantiation
 /// Logic Value Set
 /// Binary Full Adder

/// 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"

////////////////////////////////////////////////////////////////////////////////
/// Module

// :P: 4.1     Good description.
// :H: 2.3     Material covered out of order.
// :LRM: 12.1  Advanced.

//  Describes a part,
//    can be simpler than a gate
//    or more complex than an entire computer.
//
//  A module is defined once (see example) and can be /instantiated/,
//  used by other modules, many times.

// :Example:
//
// A definition of a module with no inputs or outputs, and that does nothing.

module my_first_module();

endmodule

// :Keywords: module, endmodule


////////////////////////////////////////////////////////////////////////////////
/// Module Ports

// :P: 4.2     Good description.
// :H: 2.3     Material covered out of order.
// :LRM: 12.3  Advanced.

//  A /port/ is either an input or an input [or an inout] of a module.

//  Each port has:
//   a name    E.g.,  clock, q, my_input
//   a direction:  input, output, [or inout]
//

// :Example:
//
// A module with two ports, an input and an output.
// Though it has two ports it still does nothing.

module my_second_module(my_first_output,my_first_input);
   input my_first_input;
```

```
   output my_first_output;

endmodule

// :Keywords: input, output

 /// More Information
//
// The port declarations must be at the top of a module.
// A module can have any number of ports.
// Each port name must appear near the module name (my_second_module)
//   and with a port direction below (input my_first_input).
// The input and output keywords can be followed by any positive of names.
// Inputs and outputs can appear in any order.
//   In class outputs will appear first.

// :Example:
//
// A module with four ports.  Also does nothing.

module my_third_module(a,b,c,d);
   input a, c;
   output b;
   output d;

endmodule


////////////////////////////////////////////////////////////////////////////////
/// Wire

// :P: 3.2.2
// :H: 2.4.1        Material covered out of order.
// :LRM: 3.2, 3.7   Advanced.

// Objects of type /wire/ (among other things) are used to connect
// items within a module.
//

// :Example:
//
// A module with a wire, c, declared. It doesn't connect to anything.

module my_first_wire_module(a,b);
   output a;
   input  b;

   wire   c;

endmodule

// :Example:
//
// A module with three wires declared: c, d, and e. None connect
// to anything.

module my_second_wire_module(a,b);
   output a;
   input  b;

   wire   c,d;
   wire   e;

endmodule

// :Keyword: wire
//   Used to declare wires.
```

```
 /// More Information
//
// The wire declarations can appear in many places in a module.
// wire is one of several /net/ data types.
//   Other net data types not covered in this course.

////////////////////////////////////////////////////////////////////////////////
/// Gates

// :P: 5.1                Good.
// :LRM: 7.1, 7.2, 7.3    Advanced.


// A /gate/ is a /primitive/ component.
//
// Some Verilog gates: /and/, /or/, /xor/.
//
// /primitive/: something that is not defined in terms of something else.


// :Example:
//
// A module implementing: x = ab + c using gates.
// Finally, a module that does something!

module my_umm_lost_count_module(x,a,b,c);
   input a, b, c;
   output x;

   wire   ab;

   and and1(ab,a,b);
   or or1(x,ab,c);

endmodule

 /// More Information
//
//  Gates are instantiated in modules.
//    E.g.: and and1(ab,a,b);
//  The instantiation specifies a gate type (and), a name (and1),
//    and a list of port connections (ab,a,b).
//
//  There are many rules for what can appear in a port connection.
//    Based on material covered so far...
//    ...port connections can be inputs, outputs, or wires.
//    They can also be (based on material covered later)...
//    ...sized constants, variables, and expressions (subject to restrictions).
//
//  Several gates, including "and" and "or" can have one or more inputs.
//    That is, there is no need for separate 3-input AND gates, 4-input AND,
//    gates, etc.
//
//  It does not matter what order gates are instantiated in. See
//    two examples below.
//

 ///  Gates to be used in class:
//    :Keywords: and, or, not, xor, nand, nor, xnor.


// :Example:
//
// Annotated version of module above.

module my_umm_lost_count_module_a(x,a,b,c);
   input a, b, c;
   output x;
```

```
   // Wire declarations can be omitted, but please don't.  Could have
   // used "d" or "fred", "v12012", etc, instead of "ab", but didn't
   // because "ab" helps the person reading this description remember
   // that "ab" is the output of an AND gate with inputs "a" and "b".

   wire   ab;

   // Code below /instantiates/ an AND gate. Instance name is "and1".
   // Output is always first, followed by 1 or more inputs. (Gates
   // can have any number of inputs.)
   and and1(ab,a,b);
   or or1(x,ab,c);

endmodule

// :Example:
//
// Same example except for different instantiation order. The
// different order does NOT matter.  Choose order for human
// readability.

module my_umm_lost_count_module_b(x,a,b,c);
   input a, b, c;
   output x;

   wire   ab;

   or or1(x,ab,c);
   and and1(ab,a,b);

   // This module is essentially equivalent to my_umm_lost_count_module.
   // It doesn't matter that input ab to or1 is driven by a gate (and1)
   // on the next line because the simulator does not execute Verilog
   // lines such as these in order, instead it follows signals from
   // inputs to outputs.

endmodule


// :Example:
//
// Two modules implementing an exclusive or gate.  The
// first module uses AND, OR, and NOT gates, and is shown to
// illustrate how gates are used.  The second module uses
// Verilog's xor gate, which really makes both modules unnecessary.

module my_xor_module(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


module my_xor_module2(x,a,b);
   output x;
   input  a, b;
```

```
   xor x1(x,a,b);
endmodule


////////////////////////////////////////////////////////////////////////////////
/// Module Instantiation

// :P: 4.1, 4.2
// :H: 2.3       Covered out of order.
// :LRM: 12.1    Advanced.

// A module is used by another module by /instantiating/ it.
//
// A module instantiation is similar to a gate instantiation.
//

// :Example:
//
// An xnor module which uses the xor module defined above and a not gate.

module my_xnor_module(x,a,b);
   input a, b;
   output x;

   my_xor_module x1(y,a,b);
   not n1(x,y);

   // Here a module defined above, my_xor_module, is being used, or
   // instantiated.  The name of the instance is x1.

endmodule

 /// More Information
//
// The differences between a gate instantiation and a module
//   instantiation should not make a difference in this class.
//
// If a module name (not to be confused with the instance name) is misspelled:
//   The Modelsim compiler will probably not report an error.
//   The Modelsim simulator will report the error when the design is loaded.


////////////////////////////////////////////////////////////////////////////////
/// Logic Value Set

// :P: 3.2.1   Good.
// :H: 2.4.1   Good.
// :LRM: 3.1   Good.


// Wires [other nets, and regs] can take on four values:

// 0: Logic 0, false, off, no,  sad,   :-(
// 1: Logic 1, true,  on,  yes, happy, :-)
// z: High impedance.
//       Nothing driving wire. (E.g.., not connected to anything or output hi Z.)
// x: Unknown.
//       Simulator cannot determine value.
//       Usually indicates a bug in Verilog code.
//       Sometimes this means two sources driving wire.

// :Example:
//
// Module below generates the four logic values.  See comments.
// Assume that input "a" is either zero or one.  The intent of this
// module is to generate the four physical values for demonstration
// purposes.  There are easier ways to generate the physical values.
```

```
module values_demo(v0,v1,vx,vz,a);
   input a;
   output v0, v1, vx, vz;

   wire   na;

   not n1(na,a);

   and a1(v0,a,na);  // v0 will be logic 0.

   or o1(v1,a,na);   // v1 will be logic 1.

   not n2(vx,a);
   not n3(vx,na);    // vx is driven by a 0 and 1, so its value is x.

                     // vz unconnected, so its value is z.

endmodule

// :Example:
//
// Testbench for values_demo. Material for this testbench
// not yet covered.  Ignore it if you like.

module demo_values();

   wire v0, v1, vx, vz;
   reg  in;

   values_demo d1(v0,v1,vx,vz,in);

   initial begin

      in = 0;

      #5;

      in = 1;

      #5;

   end

endmodule


////////////////////////////////////////////////////////////////////////////////
/// Binary Full Adder

// :PH: 4.5    Mixed with material to be covered later.
// :P: 5.1.3   Different design.

// A /binary full adder/ adds three one-bit integers.
//
// It is usually used as part of a larger, say 32-bit, adder.

// Inputs: a, b, cin (carry in)
// Outputs: sum, cout
//
// Computes a+b+cin and sets
//   sum to the LSB of a+b+cin and
//   cout to the MSB of a+b+cin.

// Truth Table

// a b cin | cout sum
```

```
// ====================
// 0 0 0   | 0 0
// 0 0 1   | 0 1
// 0 1 0   | 0 1
// 0 1 1   | 1 0
// 1 0 0   | 0 1
// 1 0 1   | 1 0
// 1 1 0   | 1 0
// 1 1 1   | 1 1

// cout = a b  +  a cin  +  b cin
// sum = a xor b xor cin
//
// Won't use xor here though.

// :Example:
//
// Example of a binary full adder.  There will be several other
// examples of this circuit.

module bfa_structural(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
```