## **LSU EE 4755**

The questions below can be answered without using EDA software, paper and pencil will suffice. Please turn in the solution on paper. Homework 2 will require the use of Verilog implementations. Nevertheless, runnable SystemVerilog code for this assignment can be found at https://www.ece.lsu.edu/koppel/v/2016/hw01.v (plain Verilog) and

https://www.ece.lsu.edu/koppel/v/2016/hw01.v.html (syntax-highlighted HTML).

Those who are rusty about the correspondence between Verilog code and hardware might want to look at the solution to EE 3755 Fall 2013 Homework 1, at http://www.ece.lsu.edu/ee3755/2013f/hw01\_sol.pdf.

**Problem 1:** Show a Verilog explicit structural description of the module illustrated below. In this assignment it is okay to use primitives (and, not,...), but don't get in the habit of using them.



- Base the names of ports, wires, and instances on labels in the illustration.
- Of course, use only primitives and wires. See Table 28-1 of IEEE Std 1800-2012 for a list of gates.

Solution appears below. In order to be explicitly structural NOT gates were instantiated to provide the inverted inputs for the AND gates. In real life, there would be no disadvantage using !a[2] in place of na2. (That may not be 100% true, because working for a company with super-strict HDL style rules is a real-life situation.)

```
module ezmod( output uwire x, y,
                                   input uwire [3:0] a ); // SOLUTION
          na0, na1, na2, na3;
  uwire
  not n0(na0,a[0]);
  not n1(na1,a[1]);
  not n2(na2,a[2]);
  not n3(na3,a[3]);
          alpha, beta, gamma;
  uwire
   and one(alpha,
                   na3, a[2], na1,
                                     a[0]);
  and two(beta,
                   a[3], a[2], na1,
                                     na0
                                          );
   and three(gamma, na3, na2, na1,
                                     na0 );
   or four(x, alpha, beta );
   xor five(y, beta, gamma );
endmodule
```

**Problem 2:** Answer the following questions about Verilog primitives as defined in IEEE Std 1800-2012. (See Chapter 28.) Indicate the exact section number where the answer is found.

(a) The standard provides a not primitive and a nor primitive, among others. One can easily argue that a 1-input nor gate is the same as a not gate. Does the standard actually allow Verilog code to instantiate a 1-input nor gate?

Yes, see Section 28.4.

Grading Note: It is not correct to answer "table 28-1 because it is shown as an *n*-input gate", because the table does not explicitly state that n == 1 is acceptable for a nor gate.

(b) Based on the standard, is there anything that can be done with a **not** primitive that can't be done with a 1-input **nor** primitive? (Don't try to answer this too deeply, just show an instantiation.)

Yes, a not primitive can have more than one output. The outputs all have the same value under unstressed circumstances. Multiple-output not gates will not be used for designs in this class.

**Problem 3:** Output match of module is\_1133, shown below, is 1 iff its input d (digits) is 1133 in BCD (which has the same representation as 1133\_16). The module instantiates BCD digit detection modules is\_1 and is\_3.

```
module is_1( output uwire match, input uwire [3:0] d );
  uwire
           z321;
  nor o0(z321,d[3],d[2],d[1]);
   and a1(match,z321,d[0]);
endmodule
module is 3( output uwire match, input uwire [3:0] d );
  uwire z32;
  nor o0(z32,d[3],d[2]);
   and a1(match,z32,d[1],d[0]);
endmodule
module is_1133( output uwire match, input uwire [15:0] d );
  uwire
                  m1, m2, m3, m4;
  and a1(match, m1, m2, m3, m4);
  is_1 i0(m1, d[15:12]);
  is_1 i1(m2, d[11:8]);
  is_3 i2(m3, d[7:4]);
```

endmodule

(a) Draw a diagram of is\_1133 based on the explicit structural description above. Show the insides of the is\_1 and is\_3 modules. Label the diagram using the same wire and instance names used in the Verilog descriptions.

is\_3 i3(m4, d[3:0]);

Solution appears to the right.



(a) Design a module is\_1133\_is that does the same thing as is\_1133, but that uses implicit structural code. The correct solution requires adding only one short line to the shell shown below. Don't forget that the value in d is in BCD. Note: The word short was added after the original assignment.

Solution appears below. The comparison operator checks for the correct value. We need to compare d to the BCD representation of 1133. Verilog does not have literal format just for BCD, as it does for binary, octal, decimal, and hexadecimal. But it doesn't need one because the BCD representation of 1133 is the same as the binary representation of 1133\_16, which in Verilog is 16'h1133. That means that the is\_1333 module (either version) has an output that's one iff the input is the BCD representation of 1333 or the unsigned binary representation of 4403 (because  $4403_10 = 1133_16$ ).

```
// SOLUTION
module is_1133_is( output uwire match, input uwire [15:0] d );
    assign    match = d == 16'h1133;
endmodule
```

**Problem 4:** When completed the output of module is\_1235 is 1 iff the input is 1235 in BCD. module is\_1235( output uwire match, input uwire [15:0] d );

## endmodule

(a) Complete the module. The module must be explicitly structural except for the use of the concatenation operator (see Section 11.4.12). The module **must** use is\_1 and is\_3 to detect the digits. Do not assume or design an is\_2 or is\_5 and don't put in logic to detect those digits.

Solution appears below. The  $is_1$  module is used to detect a 2 by swapping the two least-significant bits. (The same method can be used to detect a 4 or an 8.) Similarly, the  $is_3$  is used to detect a 5 by swapping the two middle digits. (The same method can be used to detect a 6 or a 9.)

```
// SOLUTION
```

```
module is_1235( output uwire match, input uwire [15:0] d );
    uwire m1, m2, m3, m4;
    is_1 i0(m1, d[15:12]);
    is_1 i1(m2, {d[11:10],d[8],d[9]}); // Actually detect 2.
    is_3 i2(m3, d[7:4]);
    is_3 i3(m4, {d[3],d[1],d[2],d[0]} ); // Actually detect 5.
    and a1(match, m1, m2, m3, m4);
endmodule
```

(b) Draw a diagram of the completed module, which should be very similar to the diagram from the previous problem.

Solution appears to the right.

![](_page_4_Figure_8.jpeg)