ENSC E-123, HW 1: Combinational Logic

Total Points: 10 plus 2 more for Grads

DUE Thursday, Feb. 5, 2015

Contents

1 Why Binary? A better idea? (3 points) 1
  1.1 How many combinations, for a given number of binary lines? (1 point) 1
  1.2 How much improvement in density of info, with more levels? (1 point) 2
  1.3 ...then why is anything other than binary rare? (1 point) 2

2 Numbers (2 points) 2

3 Implement this and that function (2 points) 3

4 Say it with VERILOG (PAL talk) (1 point) 4

5 Active Low (1 point) 5
  5.1 What function? 5
  5.2 Draw it 5

6 Two Forms of Digital Comparator (1 point, plus 2 more for GRADS) 6
  6.1 4-bit Equality Detector (1 point) 6
  6.2 2-bit A > B Detector (GRADS: 2 points) 6

1 Why Binary? A better idea? (3 points)

The binary logic widely used nowadays permits each wire or line or storage location in memory to represent only one of two possibilities, as you know. So, two lines permit 4 possible combinations (the two “binary digits” or “bits” can encode 4 distinct combinations). 3 bits can encode 8 combinations; and so on.

1.1 How many combinations, for a given number of binary lines? (1 point)

A “binary digit” or “bit” is provided by each wire or line carrying a conventional digital signal. Early microprocessors used 16 lines to address memory, providing \(2^{16}\) combinations and thus \(2^{16}\) distinct addresses or locations in memory space. The processors on early IBM PC’s worked this way—limiting address space, directly addressable, to \(2^K\) ( \(K = 1024\): the number of combinations, incidentally, provided by ten bits: this is a useful fact, worth memorizing; 20 lines provide \(2^{10} \times 2^{10}\); and so on). (IBM worked around this by letting the processor switch among such small “segments” of memory. The version of the 8051 processor that you’ll soon be using also uses just 16 lines to address memory, and also offers this somewhat-clumsy ‘paging’ option.) Later processors, including the 68000 family, used more lines to address memory. The 68000 addresses were defined using 32 lines, allowing it to address ______________ locations. The old IBM PC bus (EISA) used 20 lines, permitting it to address ______________ locations.

1Revisions: fixed unprintable \(A > B\), corrected “2-bit” to “4-bit” in caption for equality detector; tidied verbatim ABEL template form (11/13/04).

1Revisions: fixed unprintable \(A > B\), corrected “2-bit” to “4-bit” in caption for equality detector; tidied verbatim ABEL template form (11/13/04).
1.2 How much improvement in density of info, with more levels? (1 point)

If one were to adopt a 3-level scheme (“ternary” rather than binary), then each line could represent one of 3 values. How many combinations could be encoded by three ternary lines or digits?

How many could be encoded by three quaternary lines or digits (a 4-level scheme)?

1.3 ...then why is anything other than binary rare? (1 point)

Then why not use ternary, quaternary or more?

2 Numbers (2 points)

Write the decimal equivalents of the following binary numbers (even if you’ve learned another algorithm for evaluating 2’s comp, try using the fact that the MSB carries its usual weight—but is negative). (See Class Notes D1/14.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Decimal</th>
<th>--if Unsigned binary</th>
<th>--if signed (2’s comp) binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 Implement this and that function (2 points)

Show how to use AND, OR and NOT gates to implement the following functions:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>x2 x1</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4 Say it with VERILOG (PAL talk) (1 point)

Suppose inputs $X_1$ and $x_2$, in the preceding question, are assigned to two inputs of a PAL; each of the functions is assigned to a separate output pin, as stated below. Your task is just to write (with a pencil, not computer program!) the 3 equations in VERILOG’s terms, for the 3 functions we have not done for you (we did the first).

```
module func1(x1, x2, f1,f2,f3,f4); // here we list all the signals that go in and out of the thing we’re designing
  input x1; // ...then we say whether each is an INPUT or an OUTPUT (or both)
  input x2;
  output f1,f2,f3,f4;
  wire x1, x2; // ...finally, we say what type of signal each is (wire vs reg vs net)
  wire f1,f2,f3,f4; // this is mysterious jargon that means, roughly, ‘this thing doesn’t remember’

assign f1 = !x1 & !x2; //...and here you write the Boolean equations: AND is &
                        // OR is | ("A & B"), OR is | ("A | B")
assign f2 = ; // XOR is "A ^ B", NOT is ! "˜" (as in "!A")
assign f3= ;
assign f4 = ;
endmodule
```
5 Active Low (1 point)

Note: this question is easy. Don’t let all the talk here confuse you!

Here are the symbol and truth table for a 3-input OR gate, as you know:

![3-input OR symbol]

**Figure 1: 3-input OR**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Both the symbol and the name “OR” assume the convention that “1” is “true,” “0” is false.” But what happens if you adopt the contrary assumption—and many circuits make it convenient to make this change of assumptions:

5.1 What function?

What function would this same piece of hardware perform for you, if you treated all inputs and outputs as active low? That is, what logical function of input LOW’s (0’s) would this piece of hardware carry out?

5.2 Draw it

Draw the same gate but choose the “assertion level” symbol appropriate when you assume the gate is taking in and putting out signals that are “active low.” See sec. 8.07 if you’re puzzled.
6 Two Forms of Digital Comparator (1 point, plus 2 more for GRADS)

One of these exercises is taken from Lab D1 (the first question is expanded slightly—from 2-bits to four):

6.1 4-bit Equality Detector (1 point)

Use any gates to make a comparator that detects equality between two 4-bit numbers. (This circuit, widened, is used a lot in computers, where a device often needs to watch the public “address bus,” so as to respond upon seeing its own distinctive “address.”)

Hint: the XOR function is a big help.... You can think of XOR as a 1-bit equality/inequality detector.

6.2 2-bit $A > B$ Detector (GRADS: 2 points)

(This is harder.)

Use any gates to make a comparator that detects when one of a pair of two bit number (call it “A”) is larger than the other (call it “B”).

You need not make the circuit symmetrical: it need not detect $B > A$, only $A > B$, versus the contrary case “$A$ not-greater-than $B$”—formally, $A \leq B$).

Draw gates, or do the job with VERILOG\(^2\), if you prefer. You need not show assignment of variables to pins, or show a complete .v file. Just show us your Verilog equation. If your output variable is called $A_{\text{gt}}B$, for example, you need only write

```verilog
assign A_{\text{gt}}B = ...
```

---

\(^2\)If you’re a VERILOG wizard, and want to use the language’s high-level powers to do this, OK. But Please show, as well, the Boolean equivalent.