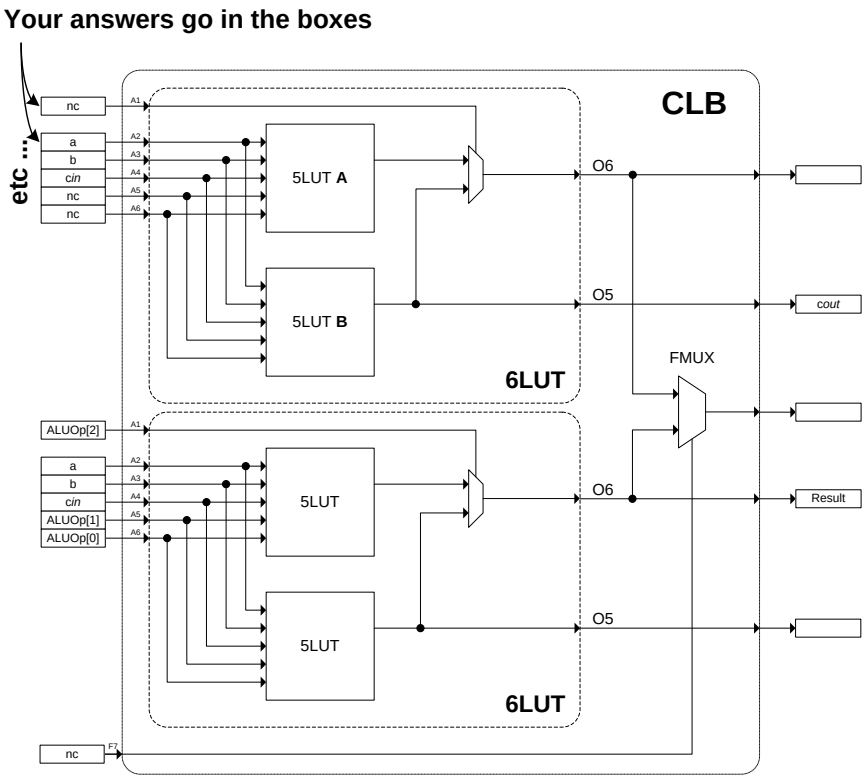
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS150, Spring 2010

Quiz 2: February 5^{th}
Solution

One possible input/output assignment is shown below:



Many others exist as well. The important part to notice is that this circuit cannot be packed into $2\times$ 5-LUTs. First, in order to build the 6-input function, you need to (at least) cascade the $2\times$ 5-LUTs together and use the MUX between them. In addition, however, you have to ensure that 5 of the 6 inputs are shared between the $2\times$ 5-LUTs *and* that the right output comes out of each of the two output ports (one for the $Result$ and one for the c_{out}) for every possible function. As it turns out, you cannot create a correct assignment with this CLB and ALUOp encoding.

Rationale: The rationale will be in two parts. The first part will explain why you have to use a bit of the ALUOp as the select bit into the MUX between the $2\times$ 5-LUTs. The second part will show why the assignment (even with one of the three ALUOp bits as the select bit) still doesn't work out.

To derive the correct result in all cases, you need to account for the $Result$ regardless of the ALUOp value, and the c_{out} for when the ALUOp is addition or subtraction. In all cases, you must get the c_{out} from the LUT whose output only *conditionally* goes through the MUX (we will call this the 5-LUT **B** - see the top 6-LUT in the diagram for which 5-LUT is which). This is because you can always determine the c_{out} through only knowing a, b, and c_{in} (three inputs). $Result$, on the other hand, depends on the ALUOp as well (which is why it must be determined from the output of the MUX between the two 5-LUTs.

Now, given that c_{out} must come from the 5-LUT **B**, we must make sure that it is always generated correctly and that it does not conflict with the $Result$. If the select bit into the MUX is one of a, b, or c_{in}, then 5-LUT **B** and 5-LUT **A** generate these functions (respectively) during an addition operation:

1. Sum when $a = 0$, Sum when $a = 1$
2. Sum when $b = 0$, Sum when $b = 1$
3. Sum when $c_{in} = 0$, Sum when $c_{in} = 1$

Likewise for subtraction. Unfortunately, one of the two 5-LUTs must be generating the c_{out}. Since both are busy generating variants of the sum, we can't get a c_{out} using any of these encodings.

Thus, an ALUOp bit must instead be used as the MUX select bit. With this scheme, 5-LUT **B** can generate the c_{out} and 5-LUT **A** can generate the result. Now, we have to check to see if we always get the correct answer (for both the $Result$ and c_{out}) when we use one of the ALUOp bits as the MUX select bit.

We will start with ALUOp[0] first. If we use ALUOp[0], then 5-LUT **B** generates the $+$, &, $\otimes$, and passthrough functions (the rest are generated by 5-LUT **A**, or visa-versa depending on which input to the MUX corresponds to the select bit being high and low). Notice that with this encoding (if $+$ does indeed come out of 5-LUT **B**), $-$ must come out of 5-LUT **A**. But this means that the c_{out} and $Result$ come out of different outputs, depending on the function, which can't ever work! Thus, ALUOp[0] will not yield the correct result in all cases.

We look to ALUOp[1] next. Consider only the $+$, $-$, &, and $|$ operations. If we use ALUOp[1] as the select bit, $+$ and $-$ must come from 5-LUT **A** and the other two operations must come from 5-LUT **B**. This is because while we computing $+$ and $-$, 5-LUT **B** must generate the c_{out}. But now we have a problem: the ALUOp encoding for those four operations (if you remove ALUOp[1]) is ambiguous. See below:

```
000 +
001 -
010 &
011 |

becomes (if we remove ALUOp[1]):

0 0 +
0 1 -
0 0 &
0 1 |
```

We said that 5-LUT **A** generates the sum/difference and that 5-LUT **B** generates the c_{out}. Because of the ambiguity, however, 5-LUT **B** *still* generates the c_{out} when we really want it to generate the & and $|$. So, this encoding won't work either.

Finally, we turn our focus to ALUOp$[2]$. With this bit as the select into the MUX, we have the same type of problem as we had with ALUOp$[1]$ (consider the $+$, $-$, $\otimes$ and $\sim$ operations this time around).

Thus, we can't pack this circuit into a single 6-LUT and must instead use $2\times$ 6-LUTs.