CS 16 Fall 2008 — Mid-term solutions

This is a closed-book, closed-note exam. Answer all of the questions **clearly, completely, and concisely**. You have 50 minutes, so be sure to budget your time. All work should be written in your blue book.

1. **Question:** (10 points) Use a Karnaugh map to simplify the boolean function described by the truth table below. Draw your rectangles clearly and express your result as a boolean algebraic equation—**do not draw a circuit.**

```
A   B   C   D  |  Y
---------------|----
0   0   0   0  |  1
0   0   0   1  |  1
0   0   1   0  |  1
0   0   1   1  |  1
0   1   0   0  |  0
0   1   0   1  |  0
0   1   1   0  |  0
0   1   1   1  |  1
1   0   0   0  |  0
1   0   0   1  |  0
1   0   1   0  |  0
1   0   1   1  |  1
1   1   0   0  |  1
1   1   0   1  |  0
1   1   1   0  |  1
1   1   1   1  |  0
```

Answer: The map, when laid out, looks something like this:

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	0	0	1	0
AB	1	0	0	1
$A\bar{B}$	0	0	1	0

When the appropriate rectangles are drawn, the simplified form is:
$Y = \bar{A}\bar{B} + \bar{A}CD + AB\bar{D} + \bar{B}CD$.

Discussion: I actually intended a slightly different function, but committed a transcription error of my own in writing up the question, leaving a slightly less interesting version of this question.
Difficulties with this question were the common ones: failure to see a wrap-around opportunity; putting rectangles around values already completely captured by other, existing rectangles; transcription errors. Overall, though, the answers revealed a good basic understanding of how Karnaugh maps are used.

2. **Question:** (15 points) Recall the basic rules for *two's complement addition overflow:* If the two inputs have the same sign and the output has a different sign from those two inputs, then overflow has occurred.
Prove that when the carry-in and carry-out of the most significant bit of a ripple-carry addition differ, that also indicates overflow. That is, show the equivalence of these two methods of overflow detection.

   Answer: Although there are a number of methods for proving this theorem, the most straightforward is via truth table. For an n-bit number, a_{n-1} and b_{n-1} are the most significant bits of each input, y_{n-1} is the most significant result bit, and c^i_{n-1} and c^o_{n-1} are the carry-in and carry-out bits, respectively, for the most significant bit.

c^i_{n-1}	a_{n-1}	b_{n-1}	c^o_{n-1}	y_{n-1}
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

   The bold lines show the two cases where overflow occur. To prove the two methods of detecting overflow equivalent, notice that when the first technique ($a_{n-1} = b_{n-1} \neq y_{n-1}$) obtains, the second technique ($c^i_{n-1} \neq c^o_{n-1}$) does also. Of equal importance, when the first technique does **not** obtain, neither does the second.

   Discussion: The most common problems occurred when people attempted a case-based approach (similar to the one above), but did not thoroughly enumerate the cases. In particular, it was common to show that...

$$(a_{n-1} = b_{n-1} \neq y_{n-1}) \Rightarrow (c^i_{n-1} \neq c^o_{n-1})$$

   ...but many people forgot also to show that ...

$$(a_{n-1} = b_{n-1} \neq y_{n-1}) \Leftarrow (c^i_{n-1} \neq c^o_{n-1})$$

3. **Question:** (25 points) The multiplier that we built for lab-4 worked only on positive integers; it can produce incorrect results on negative numbers (using two's complement).

   Given that multiplier (4-bit multiplier and multiplicand, yielding an 8-bit product), how can it be modified or augmented to correctly multiply any combination of positive and negative inputs? **Draw a circuit** to show how this improved multiplier would be structured.

   NOTE: You may assume high-level components. For example, if you need an adder, just draw a box and label it with an addition-sign. Just be sure it is clear what each component does. If you must invent new components that we've not used, make clear by description or diagram how it would be constructed.

   Answer: In lieu of a diagram, I will describe one possible solution. Given the two 4-bit inputs, a and b, ensure that both values are non-negative before feeding them into the multiplier. So, for a, feed it into input 0 of a multiplexer; then take a copy of a and pass it through a negation circuit to form $-a$, feeding this result into input 1 of the multiplexer. Finally, the control on this multiplexer should be a_3—that is, if the value is non-negative, select a, and if it is negative, select $-a$. The output of the multiplexer should be one of the inputs to the multiplier. Do the same for b, such that the second input to the multipler is either b (if b is non-negative) or $-b$ (if b is negative).

   The multipler is now operating on two non-negative values, and will produce a non-negative, 8-bit product y. However, depending on the signs of the original a and b values, the sign of final product should perhaps be negative. Therefore, again feed y into input 0 of a multiplexer, and feed $-y$ into input 1. The selection on this multiplexer should be determined by $a_3 \oplus b_3$—that is, if the signs are the same, then the result should be y, but if the signs of a and b differ, the true product is $-y$.

   Discussion: Many people thought to take the XOR of the most significant bits of each input in order to determine when to negate the result from the multiplier. However, many people either (a) forgot also to negate any negative inputs before feeding them into the multiplier, or (b) thought that it would be sufficient to strip off that most significant bit, ignoring that negation of a two's complement number involves more than just removal of the sign bit.

4. **Question:** (25 points) Consider the following sequence of 1-bit values:

   $0, 1, 1, 1, 0$

   Construct a clocked circuit that repeatedly emits this pattern of 1-bit output values every 5 clock cycles.
   NOTE: You may express any combinational functions either as explicit gates or by use of a ROM. If you use a ROM, it must be clear which lines provide the input address, which lines carry the output data, and what the complete contents of the ROM are.

   Answer: Consider the basic feedback-based counter structure that we have used: a register stores the counter value, which then serves as the input to a combinational "incrementor" circuit that calculates the *next* value of the counter; the incrementor's output then serves as input to the register. For this problem, the counter should count from 0 to 4 and then repeat, tracking the *internal* sequence number. We must then add one more combinational circuit that maps the internal sequence number to the external output.
   Therefore, we need a 3-bit register to count from 0 to 4 and wrap around to 0. Here, $y = y_2, y_1, y_0$ is the current counter value (that is, the output of the register and the input into the incrementor), while $d = d_2, d_1, d_0$ is the next counter value (that is, the output of the incrementor and the input into the register). Thus, we need to express d as a function of y.
   Additionally, we need a combinational "externalizer"—a circuit that maps y onto a final, 1-bit output, z. Therefore, the truth table for all of these input and outputs is:

y_2	y_1	y_0	d_2	d_1	d_0	z
0	0	0	0	0	1	0
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	1	0	0	1
1	0	0	0	0	0	0

   The functions are, therefore:

$$d_2 = \bar{y_2} y_1 y_0$$

$$d_1 = \bar{y_2}(y_1 \oplus y_0)$$

$$d_0 = \bar{y_2}\bar{y_0}$$

$$z = \bar{y_2}(y_1 + y_0)$$

Discussion: Overall, this problem went well for most. There were often errors in determining the logic for the incrementor, often not counting through quite the correct sequence (e.g., 0 to 5 instead of 0 to 4). Some tried completely novel circuits to emit this sequence, and some of them worked, although validating their correctness was more difficult.

5. **Question:** (25 points) Consider the following sequence of 1-bit values:

   $$0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, \ldots$$

   That is, between 0's values, there is first zero 1's, and then one 1, and then two 1's, and then three 1's, and so on. This sequence ends with a 0 is followed by 255 1's, at which point it should repeat.

   Construct a clocked circuit that repeatedly emits this pattern of 1-bit output values every 32,896 clock cycles.

   NOTE: First, obviously, you cannot just enumerate this pattern—you must devise a new approach. Second, again assume that you have high-level components at your disposal. Third, you again may express any combinational functions using either gates or a ROM, using the same rules for a ROM as above.

   Answer: To solve this problem, you need **two** counters: an *outer* counter, a, that keeps track of how many 1's should appear in this portion of the sequence, and an *inner* counter, b that counts how many 1's have been printed in this portion of the sequence so far. If y is the output of the circuit at each step, then we can list the register values at each step along with the output produced:

outer counter a	**inner counter** b	**sequence output** y
0	0	0
1	0	0
1	1	1
2	0	0
2	1	1
2	2	1
3	0	0
3	1	1
3	2	1
3	3	1
4	0	0
4	1	1
4	2	1
...	...	...
255	254	1
255	255	1
0	0	0

   First, notice that a increments only when $a = b$. Also notice that $b = 0 \Rightarrow y = 0$, and that $b \neq 0 \Rightarrow y = 1$. To control the progression of each counter register, we need a multiplexer that determines the input into each. Specifically, these multiplexers should operate as follows:

   - **Multiplexer for a input:** Input 0 is the output of a—that is, the same value that a already has. Input 1 should be $a + 1$. The selector for this multiplexer is the output of a *comparitor*—that is, a circuit that emits 1 if

$a = b$, and 0 otherwise. Thus, when $a = b$, register a will be incremented; otherwise, a will retain its previous value.

- **Multiplexer for b input:** Input 0 is $b+1$, while input 1 is 0. The selector on this multiplexer is also the result of the comparitor $a = b$. Therefore, when $a = b$, register b is reset to 0; otherwise, b is incremented.

Assuming this arrangement of the registers, one needs only a simple combinational circuit to compute $y = b_2 + b_1 + b_0$.

Discussion: Not many people got far with this question. There were some misguided attempts to employ our datapath and control constructs, but those are for implementing machine code instructions, not for straight sequence counting. Some people ignored the text of the question and described or drew solutions that involved some brute-force enumeration of all 32,896 entries in the sequence. A few managed to approach something like the nested-counters described in the solution above. Any such ideas were given some non-trivial credit.