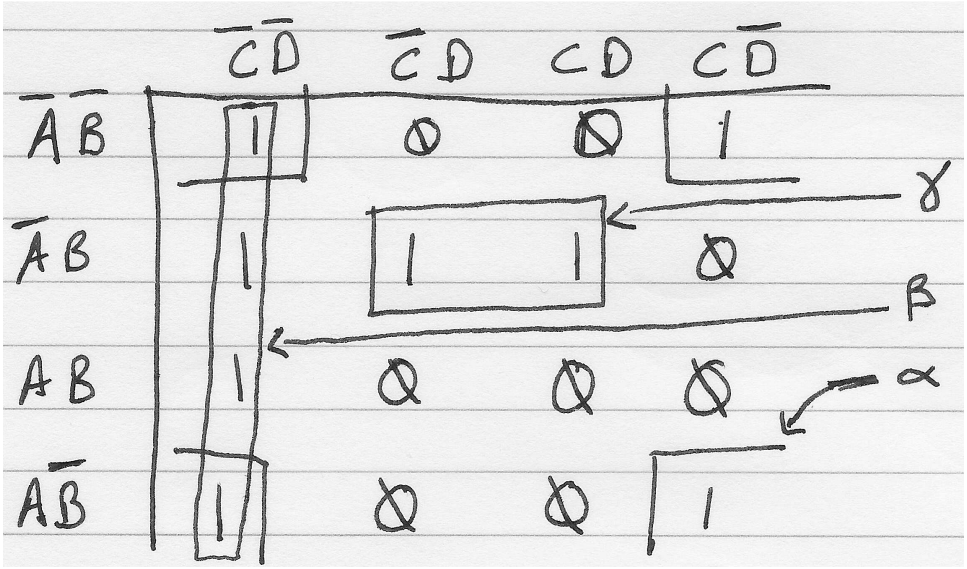
CS 16 — Fall 2009 — Mid-term exam

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 use your time wisely. All work should be written in your blue book. Note that for all of these questions, you may use high-level structures (e.g., adders, multiplexers, etc.) where appropriate. If in doubt about whether using a particular component is allowed, **ask me**.

1. (20 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	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

ANSWER: Below is the Karnaugh map for this function, with rectangles identifying terms that can be simplified:



The rectangles simplify to the following conjunctive terms:

$$\alpha = \overline{B}\,\overline{D}$$

$$\beta = \overline{C}\,\overline{D}$$

$$\gamma = \overline{A}BD$$

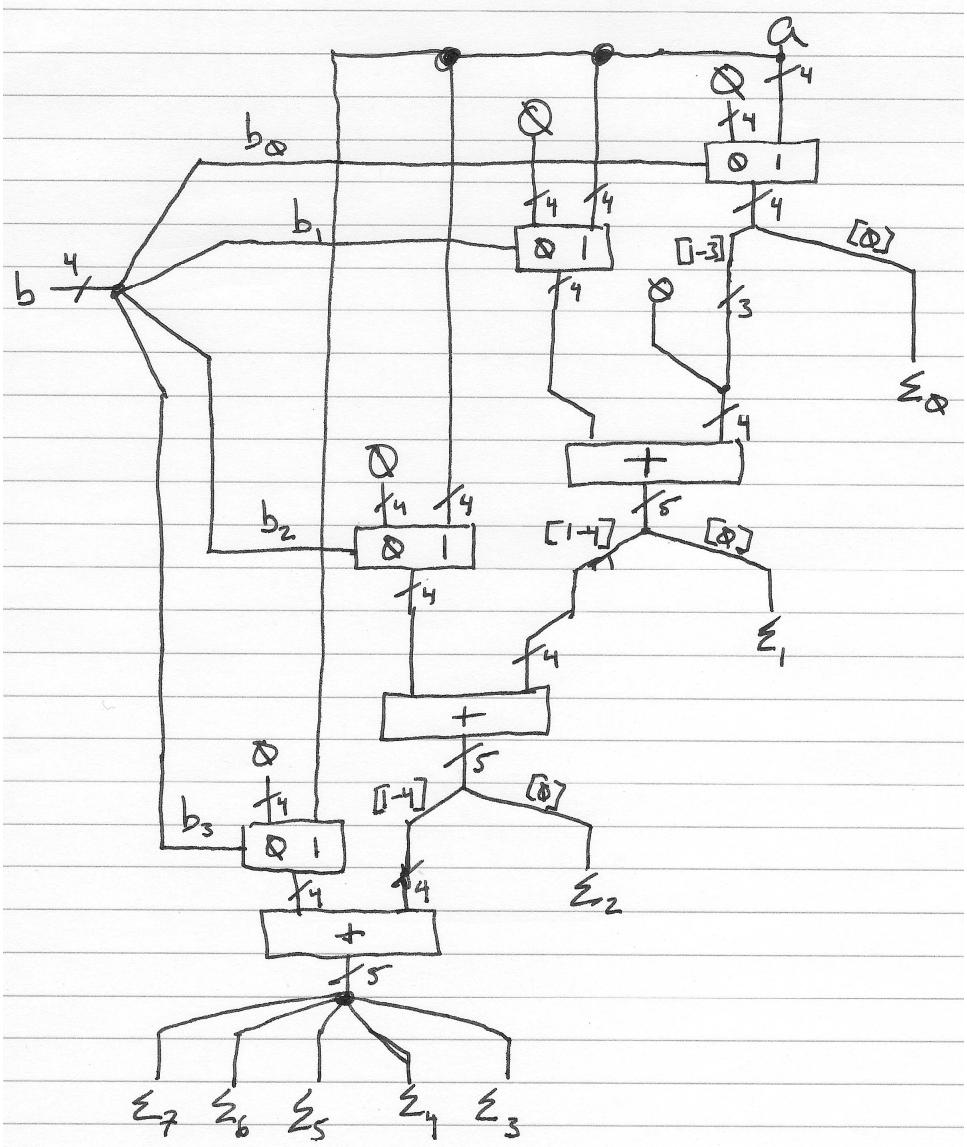
Finally, the final, simplified function is formed by disjoining those conjunctive terms:

$$Y = \overline{B}\,\overline{D} + \overline{C}\,\overline{D} + \overline{A}BD$$

DISCUSSION: The results for this problem were, overall, quite good. The common mistakes were the ones that you would expect. Some forgot the proper ordering of the row and column labels, where each must have one term in common with its adjacent rows/columns. Many committed transcription errors, simply placing the output values in the wrong locations on the map. The most common error was to fail to observe the four-corners case for α.

2. (20 points) **Draw** a 4-bit multiplier that is *combinational* (in contrast to the *sequential* multiplier from Lab 4).

   Answer: The following diagram shows a circuit that will multiply two 4-bit numbers combinationally:



   Specifically, this circuit computes $a \times b = \Sigma$, where each is decomposed into the bits $a = (a_3, a_2, a_1, a_0)$, $b = (b_3, b_2, b_1, b_0)$, and $\Sigma = (\Sigma_7, \Sigma_6, \ldots, \Sigma_0)$.

DISCUSSION: This problem won the award for the greatest variety of attempted solutions. Some failed catastrophically, providing a circuit with no meaningful structure or replicating the sequential multiplier from the lab.[1] Others indicated a basic intuition about the structure of such a multiplier, but failed to get the details right, mangling the handling of carried values into more significant digit positions.

Most interestingly, many of you presented a solution based on a collection of 1-bit half- and full-adders. This solution required a complex cascade of carry values, and when I first encountered it, I admit that I thought it was unlikely to work. However, after working through it, I could see how it worked correctly. Some presented it with flaws, but many got the details right and earned full credit.

The solution presented above was also one that many attempted or approximated. Using higher-level, 4-bit adders makes the problem of handling carry values simpler, and maps more cleanly onto the addition steps performed in the sequential multiplier, which itself is modeled after the paper-and-pencil form of multiplication.

[1] Those who provided such answers have little excuse. When we developed the sequential multiplier, I stated, explicitly, that a combinational multiplier was a certain exam question. You had time to contemplate the question and develop an answer, even in collaboration with others in the class.

3. (20 points) For Labs 2 and 3, you created a 4-bit counter that could count from 0 to 15 and wrap around 0 again. In doing so, you found that there was a regular pattern to the logic for each bit that you added to the counter. With this pattern, you could easily define the Boolean functions for a k-bit counter, for any tractable integer value of k.

   Now, consider creating a reverse counter. For example, for a 4-bit reverse counter, it should count from 15 down to 0 and wrap around to 15 again. Of course, some of you tackled this problem as the second part of Lab 3. However, many of those who did so simply inverted the bits of the output of their forwards counter—a clever trick, but of no help here.

   Find the pattern to the Boolean logic functions for a k-bit reverse counter. That is, write the logic functions for a reverse counter up to 4 bits. Simplify them so that you can express the form of the $(k-1)^{st}$ bit of a k-bit counter. (For example, for a 10-bit counter, what is the logic function for the 9^{th}, most significant bit, recalling that the least significant bit is the 0^{th}.)

   ANSWER: First, here is the truth table for the current counter state $Y = (Y_3, Y_2, Y_1, Y_0)$ and the next counter state $X = (X_3, X_2, X_1, X_0)$:

Y_3	Y_2	Y_1	Y_0	X_3	X_2	X_1	X_0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0

Next, we move through the functions for X_i, moving from less to more significant digits.

$X_0 = \overline{Y_0}$: This simplification can be obtained with a Karnaugh map.

$$\begin{aligned} X_1 &= \overline{Y_1}\,\overline{Y_0} + Y_1 Y_0 \\ &= \overline{Y_1 \oplus Y_0} \end{aligned}$$

$$\begin{aligned} X_2 &= \overline{X_2}\,\overline{X_1}\,\overline{X_0} + X_2\overline{X_1}X_0 + X_2 X_1\overline{X_0} + X_2 X_1 X_0 \\ &= \overline{X_2}\,\overline{X_1}\,\overline{X_0} + X_2(X_1 + X_0) \\ &= \overline{X_2}\,\overline{(X_1 + X_0)} + X_2(X_1 + X_0) \\ &= \overline{X_2 \oplus (X_1 + X_0)} \end{aligned}$$

$$\begin{aligned} X_3 &= \overline{X_3}\,\overline{X_2}\,\overline{X_1}\,\overline{X_0} + X_3(X_2 + X_1 + X_0) \\ &= \overline{X_3}\,\overline{(X_2 + X_1 + X_0)} + X_3(X_2 + X_1 + X_0) \\ &= \overline{X_3 \oplus (X_2 + X_1 + X_0)} \end{aligned}$$

Finally, we generalize the result for an arbitrary bit k:

$$Y_k = \overline{X_k \oplus (X_{k-1} + X_{k-2} + \ldots + X_1 + X_0)}$$

Discussion: First, it is important to note that there is at least one other, equally valid simplification:

$$Y_k = X_k \oplus \overline{X_{k-1}}\,\overline{X_{k-2}} \ldots \overline{X_1}\,\overline{X_0}$$

Most who presented this alternate simplification got it exactly correct, which is fortunate because none of them showed their work to reach this simplification.
The most common errors in solving this problem were failures in Boolean algebraic manipulation. Many used Karnaugh maps as a first step to simplification, but then reached a dead end with expressions such as $X_2 = \overline{X_2}\,\overline{X_1}\,\overline{X_0} + X_2 X_1 + X_2 X_0$. Failure to apply some of the algebraic rules in an attempt to change the form of the expression and extract some regularity was pervasive. Although time was short on this exam, the complete lack of attempted algebraic manipulation suggested that some knew not where to start with such a problem. More practice may be in order.
A less common, but not utterly infrequent error was the malformation of the truth table itself. Some provided this table:

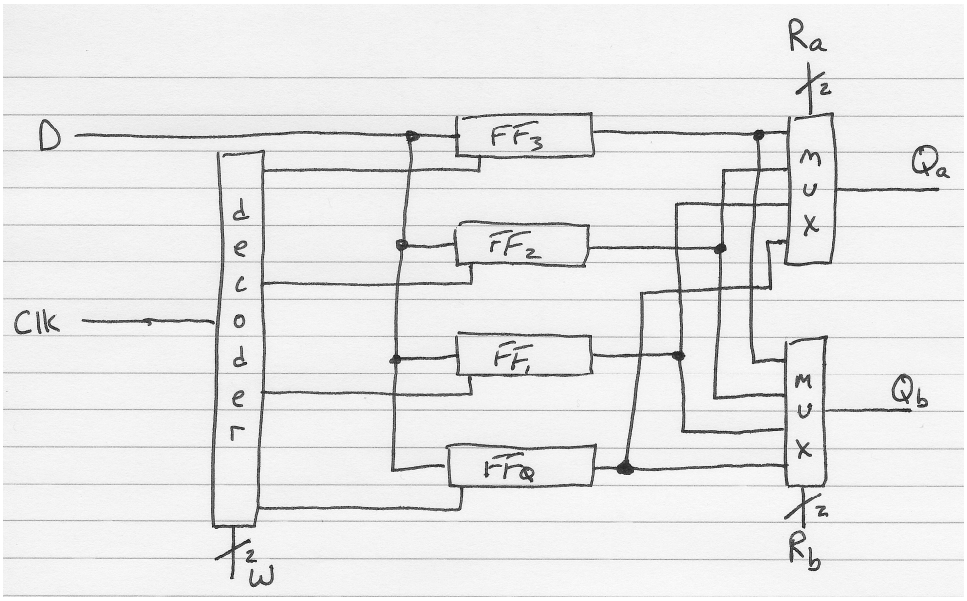
Y_3	Y_2	Y_1	Y_0	X_3	X_2	X_1	X_0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

While this table superficially shows X in the reverse order, it does **not** result in a counter that counts backwards through the integer sequence. For any given value for Y, any device that implements this truth table will ping-pong between two values. For example, if $Y = 1101$, then $X = 0010$. At the next clock cycle, $Y = 0010$, and thus $X = 1101$. Essentially, this device would simply invert the bits at every clock tick. Those who made this mistake strangely continued onward, writing simplified formulae such as $X_k = \overline{Y_k}$. What is troubling is that such simplifications did not set off alarm bells, leading the exam-taker to recognize that something was dramatically wrong. Such alarm bells **should** have gone off.

4. (20 points) Draw an addressable memory that has 4 addresses and 1 $\frac{\text{bit}}{\text{address}}$ and that has the following properties:

   - **Read two registers at once:** There should be two separate selector inputs R_a and R_b and two separate selected register outputs Q_a and Q_b such that two of the four registers can be read simultaneously. For example, if I want to read both registers 1 and 3, then I should be able to set $R_a = 01_2$ and $R_b = 11_2$ and see the value from register 1 on output Q_a and the value of register 3 on Q_b.
   - **Write a register not being read:** There should be two inputs for assigning a new value to a register. Specifically, W is the address of the register that should adopt a new value, and D is the value to be adopted.
   - **Simultaneity of reads and writes:** The reading of two registers should occur simulatenously with the write of a register. These operations may be performed on three different registers, or on the same register(s).

   ANSWER: Here is a circuit that fulfills the register file characteristics described above:



Writing to the register file is structured as always, with a decoding of the clock signal to the desired register. What is different from the addressable memory that we developed during class is that there is a dedicated selector input, W, to that decoder. That allows us to select into which register we wish to write without also simultaneously selecting a register from which to read.
Meanwhile, the outputs of each register is routed to two multiplexers. Each multiplexer has its own selector input, R_a and R_b respectively. Thus, these two inputs allow us to read any choice of two registers to read simulateously.

DISCUSSION: The answers to this question were, by and large, good. The primary challenges for these problem were (a) recalling the structure of an addressable memory, and (b) separating the addresses used to select different components, thus allowing reads and writes to be separately selected. With those observations, the modifications to the in-class addressable memory were modest.

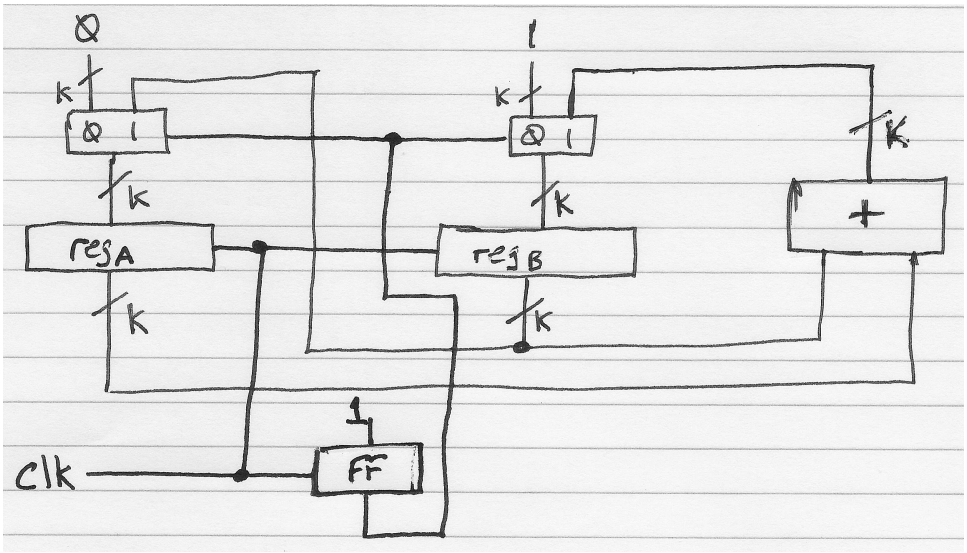
The most common error was introduce some explicit circuitry to handle the simultinaeity requirement. However, no such circuitry is necessary. The circuit above can accept a value to write into one register and read values from two registers, all at the same time, without any additional components.

5. (20 points) The Fibonacci sequence is defined as:

$$F(n) = \begin{cases} n & \text{if } n < 2 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

For example, starting at the 0^{th} element, the first 10 values of this sequence are: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$.
Design and draw a sequential logic circuit that calculates the n^{th} Fibonacci number. Since the sequence is infinite, you may assume that your circuit can handle only k-bit values—that is, for sufficiently large n, $F(n) > 2^k - 1$, and therefore your circuit can no longer validly compute $F(n)$. The user of this circuit should be able to reset it, and then clock it n times to obtain $F(n)$. Show sufficient circuitry to demonstrate how the circuit is initialized, and then how it computes the correct result.

ANSWER: The following circuit solves this problem:



Specifically, clock cycle 0 is used to load the initial values into the registers, where $reg_A = 0$ and $reg_B = 1$. Since $F(0) = 0$, we can say that $reg_A = F(0)$, and $reg_B = F(1)$. If t is the cycle number, then at any moment, $reg_A = F(t)$ and $reg_B = F(t+1)$.

DISCUSSION: The errors on this problem came in two flavors: (a) failure to handle the loading of initial values; and (b) a lack of clue. For those who suffered from (b), this problem was one in which you either had the insight, or you didn't. Problem (a) was a lack of rigor, which for some was clearly exacerbated by the time pressure. In particular, some simply failed to acknowledge the need to handle the loading of initial values properly. Others addressed this problem, but erred in the details.