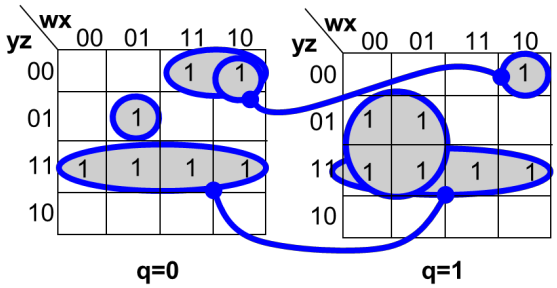
Problem Set #3

The **due date** for this homework is **Tue 2 Apr 2013 12:59 PM EDT -0400**.

Question 1

2-Level Optimization Terminology.

Consider this 5-variable Karnaugh map (Kmap), which shows a function of variables q, w, x, y, z. Kmap groupings (covers) that span both sides of the Kmap ($q = 0$ and $q = 1$) are linked with a solid line with small circles at each end.



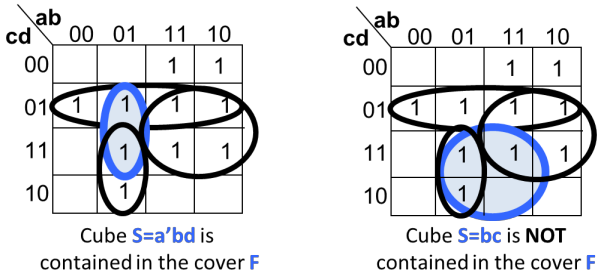
Looking at the cover of this function, select which of the following are correct statements.

- [ ] If we translate this cover into a standard SOP form with AND gates and an OR gate, that logic will have 18 literals.
- [ ] There are no literals present in any two-level SOP equation for this design.
- [ ] There are no cubes in this cover.
- [ ] This is a prime cover of this function.
- [ ] This cover has 5 cubes in it.

Question 2

How does the ESPRESSO Irredundant Operator Work?

Here is a question we need to be able to answer, to perform 2-level logic optimization: Given a cover of a function F represented as a set of cubes $\{C_1, C_2, \ldots, C_k\}$, and another cube S, does this cover of F *contain* cube S? The question is easily illustrated with a small Kmap. Function $F = \bar{c}d + ad + \bar{a}bc$. Cube $S = \bar{a}bd$. Does this cover of F also cover S? In this case, the answer is **yes**: we can see it. What if $S = bc$ instead? Then **no**: F does not cover S, and we can again see it. How can we *compute* the answer to this question?



There is a very simple way to do this via a URP algorithm. We first need some new notation:

The *cofactor* of cover F *with respect to cube* S, written F_S, is the set of cubes that result by cofactoring *each* cube C_i with respect to all the variables in the product term represented by cube S.

In this example, since we represent F with the SOP cover $\bar{c}d + ad + \bar{a}bc$, and cube $S = \bar{a}bd$, then F_S is just the set of cubes we get by setting $a = 0$ and $b = 1$ and $d = 1$ in this list of cubes in F. The result is $F_S = \bar{c} + c + 0 = 1$. This leads us to our big result:

A cover $F = \{C_1, C_2, \ldots, C_k\}$ contains cube S *if and only if* $F_S = 1$.

This is a very useful result, since it means we can test if a cover F contains any

cube S by computing the cube list for cofactor F_S and then just calling our well-understood URP tautology algorithm on F_S.

Why do we care about this result? Remember that in ESPRESSO, the big idea is the *reduce-expand-irredundant* loop. One simple way to do the irredundant calculation (indeed, this is a bit *simpler* than the way EPSRESSO *really* does it, but this way will work) is to check, for *each* cube C in the current cover F, if C is covered by all the *other* cubes in F. So, you remove C from from F, call this $[F - \{C\}]$, and then you perform the cofactor operation $[F - \{C\}]_C$, and then you call URP tautology on this new cubelist. If it is tautology, then C is redundant, and you can remove it!

Do this: Suppose
$F = A\bar{X} + AB\bar{Y} + B\bar{X}\bar{Y} + ABC\bar{Z} + AC\bar{Y}\bar{Z} + BC\bar{X}\bar{Z} + C\bar{X}\bar{Y}\bar{Z}$

Suppose $S = B\bar{X}Z$.

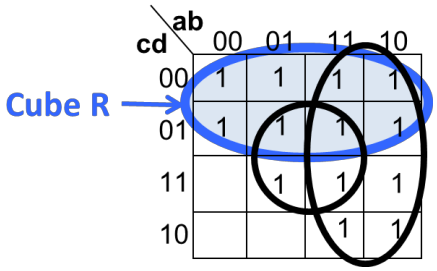
Does F cover S?

- No, F does not cover S.
- Yes, F covers S.

Question 3

How does the ESPRESSO Reduce Operator Work?

Consider the cover of the function $F(a, b, c, d)$ shown below. We want to apply the REDUCE operator to the single shaded cube R in the map.



The PCN cube list for this starting cover (3 cubes) of F is as follows. We write cubes as **[aa bb cc dd]**. So, the $F = \bar{c} + a + bd$ cube list is:

[11 11 10 11], [01 11 11 11], [11 01 11 01].

Next, note that the PCN cube list for the cover that has the single cube R =**[11 11 10 11]** *removed* – write this as $F - \{R\}$ – is as follows: **[01 11 11 11], [11 01 11 01]**.

Now, to illustrate how REDUCE does its job, do the following:

1. Draw a Kmap for the *complement* of this reduced cover $F - \{R\}$. This means: look at these two cubes in $F - \{R\}$. In a new Kmap, mark the cells where these cubes cover. Then, look at all the *other* cells in the Kmap – the cells these two cubes do *not* cover. Puts 1s in these other cells. This is the function we need to build: this is the complement of $F-\{R\}$.
2. By hand, circle a good cover of the 1s in this new Kmap, for the complement of $F - \{R\}$. Write a PCN cubelist to describe this cover of this complement.
3. Intersect each cube of this complement with the original R =**[11 11 10 11]** cube. You do this by **bitwise ANDing** the PCN bit slots between any two cubes you want to intersect. So, for example, intersect **[11 10 01 01]** with **[01 10 11 01]** gives **[01 10 01 01]**. Each intersection gives you one new cube, whose entries are the bitwise AND of the two cubes you start with.
4. Take this new intersected cubelist, and do a **bitwise OR** across all the PCN bit slots in all the cubes (if there is more than one cube). For example, the bitwise OR of **[11 10 01 01]** with **[01 01 11 01]** yields **[11 11 11 01]**.

This results in a single cube with the property we need: this is the *smallest single cube that covers the 1s that are outside of* $F-\{R\}$. That is, this is the smallest cube to which we can reduce R and still cover all the 1s outside of $F - \{R\}$.

Roughly speaking, Step 3 in the above recipe computes a set of "small" cubes that give how R overlaps with $\overline{F - \{R\}}$, and step 4 builds the smallest *one* cube that covers these, and is contained in R. Again, the desirable feature of this method is that it is fully *automatic* – we can write software to do this, and it is not a very complex operation.

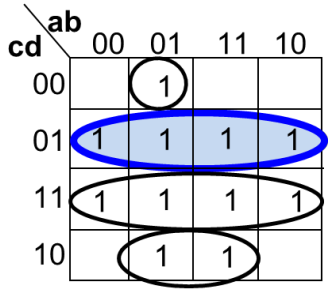
If we perform the steps of this recipe, which of the following are true? Select all the correct answers:

- A cover of $\overline{F - \{R\}}$ can be written as: [10 10 11 11], [01 10 11 11]
- It's a trick! There is no way to reduce R at all!
- The reduced result for R will be computed as [10 11 10 10]
- A cover of $\overline{F - \{R\}}$ can be written as: [10 10 11 11], [10 11 10 10], [10 11 01 10]
- The correct reduced result for R may still overlap some 1s covered by $F - \{R\}$, because there is no way to create a legal, single, reduced cube otherwise
- The reduced result for R will be computed as [10 11 10 11]

Question 4

How does the ESPRESSO Expand Operation Work?

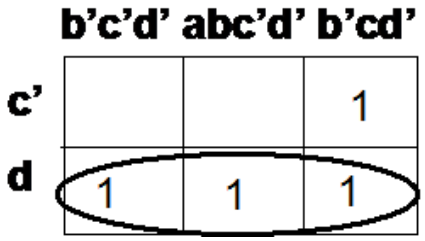
Consider this function $F(a, b, c, d)$ which we are optimizing using the reduce-expand-irredundant method like ESPRESSO. Assume we have just done a REDUCE step and we have this intermediate, nonprime 4-cube cover of F:



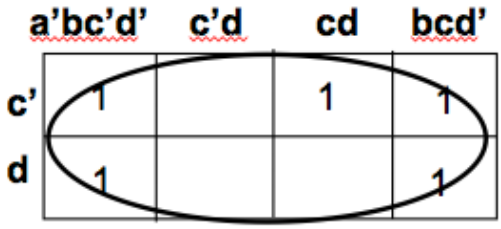
We want to perform an EXPAND operation on the highlighted cube: $\bar{c}d$. As we learned in lecture, ESPRESSO does this by building a cover of the OFF-set for this current cover, then building a Blocking Matrix for the cube we seek to expand, and then computing a cover of this matrix, which tells us how to grow this cube.

Given this recipe for the EXPAND step, which of the following are correct ways of performing this computation for this example? Select all correct answers.

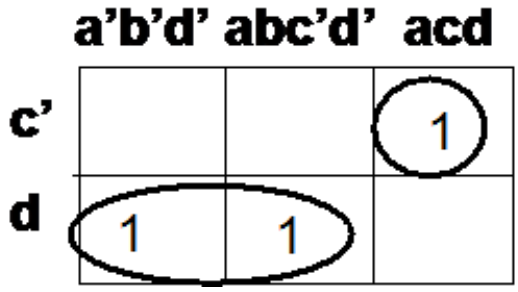
- [ ] The correct EXPAND result is d from the Blocking Matrix shown below:



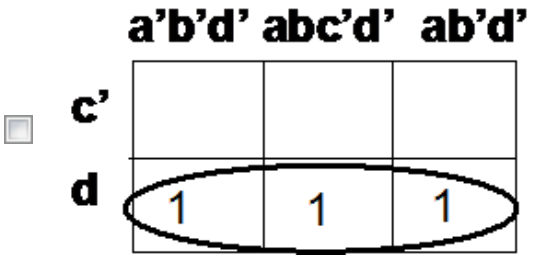
- [ ] The correct EXPAND result is $\bar{c}d$ (nothing to expand) from the Blocking Matrix cover shown below:



- [ ] The correct EXPAND result is $\bar{c}d$ (nothing to expand) from the Blocking Matrix cover shown below:



The correct EXPAND result is d from the Blocking Matrix shown below:



Question 5

LET'S TRY ESPRESSO!

The best way to understand what a real 2-level optimizer can do is to try it. So, let's go run ESPRESSO and see what happens. As we mentioned in class, ESPRESSO is a tool originally developed at the University of California at Berkeley by Rick Rudell, Bob Brayton, and Alberto Sangiovanni-Vincentelli.

If you go look in the TOOLS section of our course website, you will find two helpful tutorials about **ESPRESSO**: a written document that explains the tool and its capabilities, and also a short video tutorial about to use **ESPRESSO** in our Coursera MOOC environment. Please go look at those before you proceed further in this problem.

Assuming you have read/viewed these tutorial materials, here is what we want you to do. Suppose we have a function $F(x_1, x_2, x_3, x_4, x_5, x_6)$. We set $F = 1$ if there are exactly 2 **1's**, or 3 **1's**, or 4 **1's** in the input variables, else we set $F = 0$. So, for example, $F(1, 1, 1, 0, 1, 0) = 1$, because we see 4 **1's**. And we set set $F(0, 1, 1, 0, 0, 0) = 1$, because we see 2 **1's**. But $F(0, 0, 0, 0, 0, 1) = 0$, because see only one 1. Similarly, $F(1, 1, 0, 1, 1, 1) = 0$ because we see 5 **1's**. We want you to specify an ESPRESSO input file that will optimize this function as a 2-level form. You can type this out by hand, or if you like, you can write a program to do it. (This is small enough you can type it, it's just a little bit boring).

Your ESPRESSO input file should start like this:

.i 6

.o 1

.ilb x1 x2 x3 x4 x5 x6

.ob F

...and then have 64 lines of 1s and 0s.

- [ ] There are 64 literals in this design.
- [ ] There are 96 literals in this design.
- [ ] All AND gates in this 2-level ESPRESSO design have 4 inputs.
- [ ] There are 16 product terms in this design.
- [ ] There are AND gates with 2 inputs and 4 inputs in the 2-level hardware for this ESPRESSO design.
- [ ] There are 64 product terms in this design.

Question 6

Multi-Level Logic, the Boolean Network Model and the Algebraic Model

Which are these are true statements about the Boolean Network Model and the Algebraic Model?

- [ ] The Brayton-McMullen theory tells us that the only place to look for multi-cube divisors for a pair of functions F and G is in intersections of their respective kernels.
- [ ] The Brayton-McMullen theory tells us that co-kernels for a function F can be found in the intersections of the cubes that comprise the SOP form of function F; eg, if acd and cde are cube in F, then $acd \cap cde = cd$ is a potential co-kernel for F.
- [ ] The algebraic model seeks to make Boolean functions simpler to optimize by treating them as polynomials over real numbers.

- [ ] The algebraic model loses some of the “espressivity” of full Boolean logic, and this is the compromise we make to be able to do useful operations like division and factoring.
- [ ] Algebraic division is also called “weak” division since the algebraic model loses some of the “espressivity” of full Boolean logic.

Question 7

Algebraic Division

Consider these 2 functions of variables $p, q, r, s, t, u, v, w, x, y, z$

$$F = pt + xz + qt + pyz + rst + qyz + puvw + px + quvw + rsuvw$$
$$D = p + q + rs$$

Use the algebraic division algorithm from class to compute $Q = F/D$. (Hint: you want to build the table as in the lecture notes: one row for each cube in F; one column for each cube in D; do the cube-wise walk through D and build up the partial quotient solution for Q one column of this table at a time. When you are done, you can extract the remainder R from the computed quotient.)

Which of these are correct statements about the result of this division exercise?

- [ ] The quotient for $Q = F/D = t + yz$.
- [ ] The remainder R, for $F = Q * D + R$, is $R = xz + pyz + qyz + qt$.
- [ ] The quotient for $Q = F/D = t + x + yz + uvw$.
- [ ] The remainder R, for $F = Q * D + R$, is $R = xz + pyz + qyz + px$.
- [ ] The remainder R, for $F = Q * D + R$, is $R = xz + pyz$
- [ ] The quotient for $Q = F/D = t + uvw$.

Question 8

Kerneling

Here is a function represented in algebraic form:

$$F = pt + xz + qt + pyz + rst + qyz + puvw + px + quvw + rsuvw$$

Use the recursive kerneling method discussed in the lectures, and run the algorithm by hand to extract all the kernels (and their associated co-kernels) from F. Which these are correct statements about these kernels and co-kernels in F?

- ☐ Kernel = $p + z$, with co-kernel x
- ☐ Kernel = $t + yz + uv$, with co-kernel rs
- ☐ Kernel = $p + q + rs$, with co-kernel vw
- ☐ Kernel = $x + py + qy$, with co-kernel z
- ☐ Kernel = $t + yz + uvw + x$, with co-kernel p
- ☐ Kernel = $x + y$, with co-kernel t

☐ In accordance with the Honor Code, I certify that my answers here are my own work.

Submit Answers Save Answers