Problem Set #4

The **due date** for this homework is **Tue 9 Apr 2013 12:59 PM EDT -0400**.

Question 1

Single Cube Divisor Extraction from Multi-level Logic

Consider this Boolean logic network with variables a, b, c, d, e, f:

$R = abf + abcd + abce + cdf$

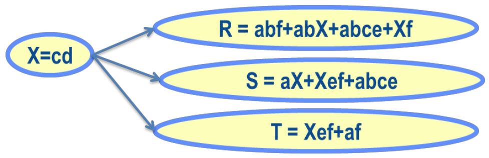
$S = acd + cdef + abce$

$T = cdef + af$

Build the *cube-literal matrix* associated with this set of functions. Look at what prime rectangles are possible in this matrix. Which are correct statements about extractions using this matrix?

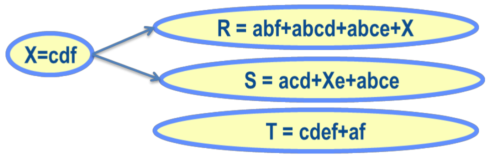
☐ Single cube divisor cd can be extracted as a prime rectangle. It has 2 columns and 4 rows $(abcd, acd, cdef, cdf)$. Extracting this divisor saves 3 literals.

☐ The picture below is a correct single-cube extraction based on a prime rectangle:



☐ Single cube divisor abc can be extracted as a prime rectangle. It has 3 columns, and 2 rows $(abcd, abce)$. Extracting this divisor saves 3 literals.

☐ The picture below is a correct single-cube extraction based on a prime rectangle:



☐ Single cube divisor bc can be extracted as a prime rectangle which has 2 columns and 2 rows $(abcd, abce)$. Extracting this divisor saves 3 literals.

Question 2

Multiple Cube Divisor Extraction from Multi-level Logic

Suppose we have these two Boolean functions, defined over 9 variables $p, q, r, s, t, u, v, w, x$:

$$G = qt + rst + pqr + quvw + rsuvw$$
$$H = pu + qtx + qu + rsu$$

Use the ideas from the multiple-cube extraction methods from lecture, to build a *Co-kernel-Cube matrix*, and look for any good prime rectangles to extract a common divisor to reduce the complexity of the network.

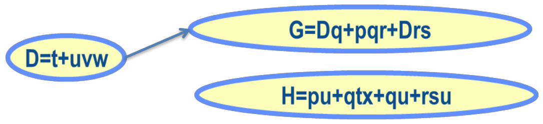
To assist in this construction, here are all the kernels and co-kernels for these functions:

```
Function:  G = qt + rst + pqr + quvw + rsuvw
        Kernel t+pr+uvw         co-kernel q
        Kernel t+uvw            co-kernel rs
        Kernel st+pq+suvw       co-kernel r
        Kernel q+rs             co-kernel t
        Kernel q+rs             co-kernel uvw
Function:  H = pu + qtx + qu+ rsu
        Kernel u+tx             co-kernel q
        Kernel p+q+rs           co-kernel u
```

Which of the following are correct statements about this extraction process?

☐ Multiple-cube divisor $D = t + uvw$ can be extracted as a prime rectangle from this matrix. The prime rectangle has 2 columns and 2 rows: (Function Co-kernel) rows are: $(G\ q), (H\ u)$. This extraction saves 3 literals.

☐ The picture below is a correct multi-cube extraction from this network.



☐ Before any common divisor extraction, the network with just the G and H nodes in it has 27 literals.

☐ Multiple-cube divisor $D = q + rs$ can be extracted as a prime rectangle from this matrix. The prime rectangle has 2 columns and 3 rows: (Function Co-kernel) rows are: $(G\ t), (G\ uvw), (H\ u)$. This extraction saves 8 literals.

☐ Multiple-cube divisor $D = q + tx$ can be extracted as a prime rectangle from this matrix. The prime rectangle has 2 columns and 3 rows: (Function Co-kernel) rows are: $(G\ q), (G\ rs), (H\ q)$. This extraction saves 4 literals.

Question 3

Basic Facts About Multi-Level Don't Cares

Which of these are correct statements about multi-level don't cares?

Select all the correct answers.

☐ Satisfiability Don't Cares (SDCs) are associated with wires inside a Boolean network model, and represent impossible patterns of inputs to, and the output of internal nodes.

☐ Multi-level don't cares are called "implicit" because they arise from the structure of a Boolean network model, and we need to go find them.

☐ Observability Don't Cares (ODCs) are constructed from SDCs.

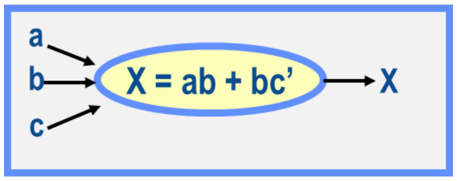
☐ Controllability Don't Cares (CDCs) can be computed from internal SDCs and knowledge of any external network don't care patterns.

☐ In any multi-level Boolean network, it generally true that we extract all the implicit don't cares (CDCs and ODCs), no matter the size and complexity of the network.

Question 4

Satisfiability Don't Cares (SDCs)

Consider this node computing the value X somewhere inside a Boolean logic network shown below. Which are the correct statements about the Satisfiability Don't Care SDC_X for the wire labeled X in this network?

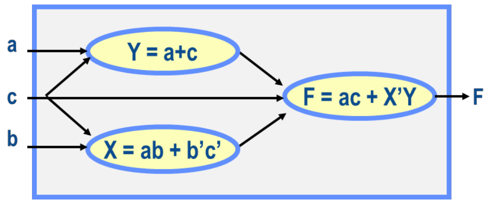


- ☐ SDC_X includes the impossible patterns $Xabc = 1-0-$, among others.
- ☐ SDC_X includes the impossible patterns $Xabc = 11--$, among others.
- ☐ SDC_X can be represented as $\bar{X}ab + \bar{X}b\bar{c} + X\bar{b} + X\bar{a}c$
- ☐ SDC_X is computed as $X \oplus (ab + b\bar{c})$
- ☐ SDC_X is a Boolean function of inputs X, a, b and c, which makes a 1 for impossible patterns of these variables.

Question 5

Controllability Don't Cares (CDCs) in Multi-Level Logic

Consider this small Boolean logic network:



Use the methods from the lecture to extract the CDC_F patterns for node F.

Hint: You can use the **kbdd** BDD calculator to do this, if you want. This is, after all, *computational* Boolean algebra. The only tricky part is that you need to define the internal wires as *both* variables and functions. Since we already saw in lecture how to compute the SDC for a node that is a simple OR gate, we can repeat the computation here for node Y, now done with **kbdd**:

```
boolean a b c Yvar
eval Ynode a + c
eval SDC_Y  Ynode ^ Yvar
sop SDC_Y
```

Observe that we define "**Yvar**" so we can use it as a variable in the SDC computation, but we also define a function "**Ynode**" to compute "the expression for **Y**". We can then evaluate the SDC_Y cover, and can ask **kbdd** to print out an SOP form. **Kbdd** prints the following for "sop SDC_Y":

```
KBDD: sop SDC_Y
  a & !Yvar
+ !a & c & !Yvar
+ !a & !c & Yvar
```

which is $a\bar{Y} + \bar{a}c\bar{Y} + \bar{a}\bar{c}Y$. Of course, this will not usually be optimized (i.e., a small equation). But you could use this as the start of a simple Kmap optimization to clean this up.

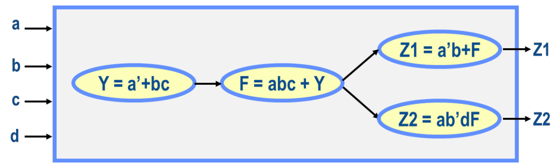
Which of the following statements are correct, in describing the CDC_F don't cares?

- ☐ There are 11 total impossible CDC patterns of $acXY$ for node F.
- ☐ Pattern $acXY = 00--$ is impossible at the inputs to F.
- ☐ There are 10 total impossible CDC patterns of ($acXY$) for node F.
- ☐ Pattern $acXY = 0001$ is impossible at the inputs to F.
- ☐ $CDC_F = (\forall b)[(Y \oplus (a+c) + X \oplus (ab + \bar{b}\bar{c})]$

Question 6

Observability Don't Cares (ODCs) in Multi-Level Logic

Consider this small Boolean logic network:



Use the methods from the lecture to extract the ODC_F patterns for node F.

Hint: You can use the **kbdd** BDD calculator to do this, if you want. **Kbdd** has a very general mechanism to replace any variable in a function, with any arbitrary other function. (Example: take a piece of hardware with an output F and inputs a, b, c, d, and make a new piece of hardware with output G; then connect the G output to the "d" input on F, replacing the variable "d" in F with the function G. It's a very natural thing for hardware composition.) This also lets us cofactor things easily. Here is a small example, where we set function $F = x \oplus y \oplus z$ and then calculate the two Shannon cofactors **F(x=1)** (which we call **F_x** in **kbdd**) and **F(x=0)** (which we call **F_xbar** in **kbdd**):

```
KBDD: boolean x y z
KBDD: eval F x^y^z
F: x^y^z
KBDD: sop F
  x & y & z
+ x & !y & !z
+ !x & y & !z
+ !x & !y & z
KBDD: replace F_x F x 1
KBDD: sop F_x
  y & z
+ !y & !z
KBDD: replace F_xbar F x 0
```

```
KBDD: sop F_xbar
  y & !z
+ !y & z
```

Which of the following statements are correct, in describing the ODC_F don't cares?

- ☐ Patterns $abcY = 10--$ make all Z outputs insensitive to F.
- ☐ Patterns $abcY = 00--$ make all Z outputs insensitive to F.
- ☐ $ODC_F = (\forall d)[\ \overline{(\partial Z1/\partial F)} \cdot \overline{(\partial Z2/\partial F)}\ ]$
- ☐ There are 4 ODC patterns of $abcY$ for node F, that mask F at all Z outputs.
- ☐ There are 2 ODC patterns of $abcY$ for node F, that mask F at all Z outputs.

Question 7

Q7 LET'S TRY SIS!

The best way to understand what a real multi-level optimizer can do is to try it. So, let's go run **SIS** and see what happens. **SIS** is a multi-level logic synthesis tool originally developed at the University of California at Berkeley by many students working with Professors Bob Brayton and Alberto Sangiovanni-Vincentelli. A good reference about SIS is http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html

If you go look in the TOOLS section of our course website, you will find a short tutorial document about **SIS**. You run **SIS** exactly like you ran **ESPRESSO** in the last homework: you edit a small textfile with the right inputs; you upload it through our portal to the Coursera servers; we run it and send you back the results; you look at the results to answer this question. Please go look at the tutorial document 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 two 3-bit unsigned integers, **a2 a1 a0** and **b2 b1 b0**. We want you to MULTIPLY these two numbers, and then compute the 4-bit result **s3 s2 s1 s0** modulo 13. So, think of this as the following computation:

> **s[3:0] = (a[2:0] * b[2:0]) mod 13**

Keep in mind that you (conceptually) do the 3-bit multiply first to compute the **6-bit** product, then do the mod-13 remainder (a number between 0 and 12) as the result. If this was a C program, the inner loop would be something like:

```
unsigned int a, b s;
s = (a * b)%13;
s = s & 0x0000000f;
```

As one concrete example, if **a2 a1 a1 = 110** (the number 6), and **b2 b1 b0 = 101** (the number 5), then 6*5 = 30, and 30 mod 13 = 4 (since 30 = 2*13 + 4). So the resulting 4-bit output is **s3 s2 s1 s0 = 0100**.

SIS can read many different input format, but in particular, it can read exactly the same truth table format (the so-called PLA format) that ESPRESSO can read. Since you already did a problem on the last assignment in this format, we will use this.

Do this: Make a file that describes this truth table. That file should start like this:

```
.i 6
.o 4
.ilb a2 a1 a0 b2 b1 b0
.ob s3 s2 s1 s0
```

And it should have lines that look this, for our 6*5 = 4 mod 13:

```
110101 0100
```

We will use a standard synthesis script to optimize this result. Look at the SIS output and answer this question:

How many nodes are this Boolean logic network?

Question 8

(*This question is a continuation of Question 7. Please refer to the description of the problem there*).

How many literals are in this Boolean logic network?
(Hint: Literal is the appearance of variable in true or complemented form, which shows on the RHS of an "=" in any Boolean Logic node.)

Question 9

(*This question is a continuation of Question 7. Please refer to the description of the problem there*).

How many AND gates (product terms with at least 2 variables in them) are in this Boolean logic network?

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

Submit Answers Save Answers