Homework 5

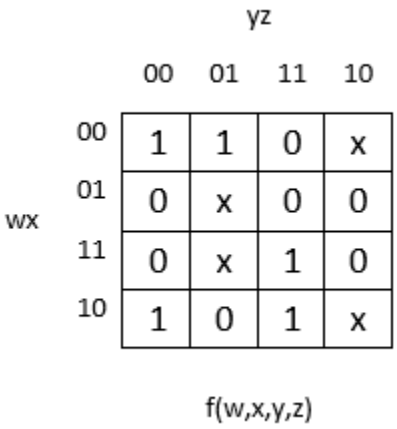
Homework 5 is due on Wednesday, March 25, at the start of the lecture. Remember to include your *Discussions section* (e.g. ED1) and follow the complete Homework submission guidelines.

Please ask all questions about this assignment during the office hours, or post them on piazza.

Boolean algebra and two-level design

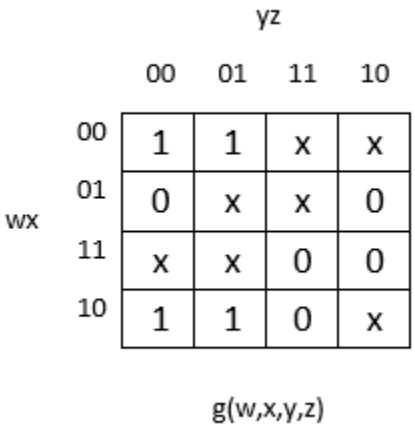
1. Simplifying with Don't Cares

Consider functions f(w,x,y,z) and g(w,x,y,z) specified by the Karnaugh maps shown below.



wx \ yz	00	01	11	10
00	1	1	0	x
01	0	x	0	0
11	0	x	1	0
10	1	0	1	x

f(w,x,y,z)



wx \ yz	00	01	11	10
00	1	1	x	x
01	0	x	x	0
11	x	x	0	0
10	1	1	0	x

g(w,x,y,z)

For each function, f and g:

1. Give a minimal SOP expression and show the corresponding loops on the K-map.
2. Give a minimal POS expression and show the corresponding loops on the K-map.
3. Is your minimal SOP expression in part 1 equivalent to your minimal POS expression in part 2? Explain, using 1-2 sentences. *Hint:* Refer to your K-maps and observe the loops. (Do not try to verify algebraically.)

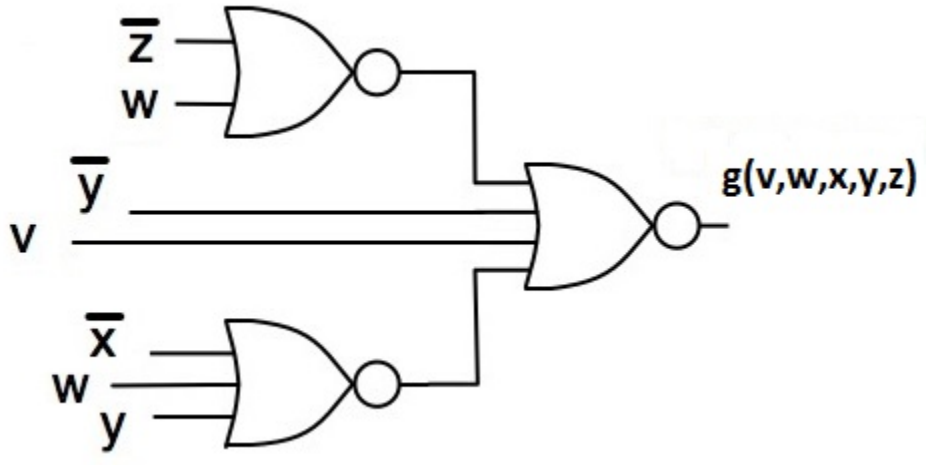
2. Boolean algebra, duality

Let *x* and *y* be Boolean variables.

1. Using perfect induction, prove that x(x'+y)=xy.
2. State the dual of x(x'+y)=xy. By what principle do you know the dual statement is also true?
3. Using dual of x(x'+y)=xy, simplify the following Boolean expression: y'+z+x(y'+z)'.

3. NAND/NOR circuits

1. Implement wx + x'z + ywz using NAND gates only. Do not simplify the expression.
2. The figure below shows a 2-level NOR circuit that implements a function g(v,w,x,y,z). Give a product-of-sums expression for g(v,w,x,y,z) and draw a 2-level OR-to-AND circuit for function g. Assume complemented inputs are available.



4. Logic design word problem

You notice an advertisement in a campus newspaper seeking an ECE student for a part-time job working in Psychology Professor Zapper's laboratory, and you apply for and get the job! Professor Zapper assigns you the task of building logic circuitry for the following experiment studying the learning ability of rats. The experimenter enters a number 0,1,2, or 3 on a keypad, which encodes the number as a two-bit unsigned integer as the binary input $\mathbf{A_1A_0}$ to your circuit. The experiment cage has two levers to which the left and right paws of the rat are strapped. The output of the levers, **L** and **R**, respectively, which will also be inputs to your circuit, are 1 if pressed by the rat, and 0 otherwise.

There are three lights in a row in the cage, corresponding to the input numbers 1, 2, and 3; call these $\mathbf{F_1}$, $\mathbf{F_2}$, and $\mathbf{F_3}$; the corresponding light should be on (output 1) only for the corresponding input number. There is also an output **Z**, which when activated (**Z = 1**) gives the rat a mild electric shock.

Professor Zapper wants the experiment to work as follows:

- When the experimenter's input number is zero, the rat is never shocked (**Z = 0**), and all of the lights are off.
- When the input number is 1, the light $\mathbf{F_1}$ alone is activated, and the rat receives a shock **unless** it presses only the left lever (**L = 1**).
- When the input number is 2, the light $\mathbf{F_2}$ alone is activated, and the rat receives a shock unless it presses only the right lever (**R = 1**).
- When the input number is 3, the light $\mathbf{F_3}$ alone is activated, and the rat receives a shock if it presses either or both levers.

Based on this description, you need to design a circuit for Professor Zapper's experiment.

1. Design the desired truth table (inputs $\mathbf{A_1}$, $\mathbf{A_0}$, **L, R**) for the outputs **Z,** $\mathbf{F_3}$, $\mathbf{F_2}$, and $\mathbf{F_1}$.
2. Use a Karnaugh map to reduce the logical expression for this function **Z** to a minimal SOP form.
3. Draw the complete logic schematic circuit diagram for your system, for inputs $\mathbf{A_1}$, $\mathbf{A_0}$, **L, R** entering on the left side, and outputs $\mathbf{F_1}$, $\mathbf{F_2}$, $\mathbf{F_3}$, and **Z** on the right side.

5. 4-bit Gray code to Excess-3 converter circuit

Excess-3 is a decimal code, in which each decimal digit d is represented by the 4-bit binary representation of d + 3. For instance the decimal digit 2 would be denoted in excess-3 by 0101 (because 2 + 3 = 5 and 0101 is the binary representation of 5). Excess-3 also has 6 unused 4-bit strings, including 0000 and 1111.

Now consider a **4-bit Gray code** as defined in Section 2.1 of Prof. Lumetta's notes, where the first ten codes are used to denote decimal digits. For example, number 2_{10} is represented as 0011_2., number 9_{10} is represented as 1101_2. Other combinations should never be encountered

Decimal	Excess-3 $x_3x_2x_1x_0$	4-bit Gray code $g_3g_2g_1g_0$
0	0011	0000
1	0100	0001
2	0101	0011
3	0110	0010
...	...	...
9	1100	1101

In this problem you will design a logic circuit that takes a 4-bit Gray code encoding $g_3g_2g_1g_0$ as input and outputs the corresponding 4-bit excess-3 codeword $x_3x_2x_1x_0$.

1. Complete a truth table showing the four outputs x_3, x_2, x_1, and x_0, corresponding to each Gray code input $g_3g_2g_1g_0$. Draw all 16 rows in the truth table (the rows must be in ascending order) and use don't cares where appropriate.
2. Draw K-maps for all four outputs x_3, x_2, x_1, and x_0 , each as a function of g_3, g_2, g_1, and g_0 Notice that some input bit combinations are not valid inputs and so should never be encountered. These inputs correspond to don't cares in the K-map.
3. Using the K-maps from part 2, give a minimal SOP expression for each of x_3, x_2, x_1, and x_0. Be sure to use don't cares in order that your expressions are as simple as possible.
4. Now draw the complete logic schematic, with inputs entering from the left and outputs exiting to the right. Your circuit should be 2-level and based on the minimal SOP expressions from part 3; you can assume that complemented inputs are also available. *Note:* Keep your drawing as easy to read as possible; e.g. keep wires short and do not let wires cross.
5. Suppose we wish to include a fifth output **E**, where **E** = 1 if and only if the input was ***invalid***. (In other words, **E** = 1 indicates there was an input error.) Give a ***simple*** Boolean expression for E. You do not need to draw the circuit.

6. Programming in C

```
#include <stdio.h>
int main()
{
    unsigned int a, b, c, d;
    unsigned int f;

    /* Print header for K-map. */
    printf("         bc      \n");
    printf("     00 01 11 10 \n");
    printf("   _____________\n");

    /* row-printing loop */
    for (a = 0; 2 > a; a = a + 1)
    {
        printf("a=%u | ", a);

        /* Loop over input variable b in binary order. */
        for (b = 0; 2 > b; b = b + 1)
        {
            /* Loop over d in binary order.*/
            for (d = 0; 2 > d; d = d + 1)
            {
                /* Use variables b and d to calculate *
                 * input variable c (iterated in      *
                 * Gray code order).                  */
                /* CALCULATE c HERE. */

                /* Calculate and print one K-map entry *
                 * (function F(a,b,c) ).               */
                /* INSERT CODE HERE. */
            }
        }

        /* End of row reached: print a newline character. */
        printf("\n");
    }

    return 0;
}
```

The above program provides some initial code to generate a 3-variable K-Map for a 3-variable Boolean function. For example, for function F(a,b,c) = ab + b'c' the program should print the following:

```
         bc
     00 01 11 10
   _____________
a=0 | 1  0  0  0
a=1 | 1  0  1  1
```

1. Complete the program above. For your portion of the code at "CALCULATE c HERE", you must begin by ensuring that the variable c follows Gray code order and runs from 0 to 1 when b=0, but from 1 to 0 when b=1. At "INSERT CODE HERE", you must then calculate the function F. Do not otherwise change the program. **For full points, use C's bitwise operators (&, |, ^ and ~) to perform both of these calculations.**
2. Print the K-Map for **F(a,b,c) = (a+b') (a'+c') (a+b'+c').**
3. Turn in a printout of your code and its output.

Hint 1: You may find it helpful to verify the output on a paper version of the K-map.

Hint 2: Remember that bitwise operators operate on the entire pattern of 32-bits (here the bit pattern corresponds to unsigned int). It might be useful for you to print the output F to see for yourself what is happening with the bits. Then, before having your code printing F, you must 'discard' all bits other than the 1's position (one way would be to use an & operation with an appropriate value).

Again, you may not hardcode the K-map or change the program other than as specified.