cs281: Computer Organization

Lab3 Prelab

Our objective in this prelab is to lay the groundwork for simplifying boolean expressions in order to minimize the complexity of the resultant digital logic circuit. We saw in Lab02 that mechanical adherence to arriving at a boolean expression for a digital logic output from a complete truth table and yielding a sum-of-products form of resultant expression, while it works, may not give us the simplest circuit. In this prelab, we will explore two means of simplification.

1 Boolean Algebra Simplification

1.1 Basic Equivalences

In the following table of equivalencies, think of the variables A, B, and C not as atomic, but rather as being able to match any boolean expression. So for the first identity, $(X + Y) \cdot 1$ is still equivalent to $(X + Y)$ by the multiplicative identity. Also, the equivalences can go in either direction, so if it is helpful for a subsequent step, we can go from $(X + Y)$ to $(X + Y) \cdot 1$ by the multiplicative identity.

Equivalence	Name	Comment
$A + 0 = A$	additive identity	
$A + 1 = 1$	additive identity	
$A + A = A$	additive identity	
$A + \overline{A} = 1$	additive identity	
$A \cdot 0 = 0$	multiplicative identity	
$A \cdot 1 = A$	multiplicative identity	
$A \cdot A = A$	multiplicative identity	
$A \cdot \overline{A} = 0$	multiplicative identity	
$\overline{\overline{A}} = A$	double complement	
$A + B = B + A$	commutative property of addition	
$A \cdot B = B \cdot A$	commutative property of multiplication	
$A + (B + C) = (A + B) + C$	associative property of addition	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	associative property of multiplication	
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	distributive property	

Recall that two boolean expressions are *equivalent* if, for all possible input combinations, the two expressions yield the same output. With this definition of equivalence, we can use a truth table whose input side consists of the variables for any of the lines of the above table, and derive columns for the expressions on the left and right of the equals sign. If these are indeed equivalent, the output columns will have identical values for all input combinations. And since the truth table contains all possible input combinations, this would be one way to prove, by a complete case-based analysis, the above equivalences.

1.2 Algebraic Proof Equivalences

Based on these basic equivalences, we can build proofs of other equivalences. Consider the boolean expression $A + AB$. In the following "two-column" proof, fill in the reason that allows us to conclude each subsequent step from the initial given expression:

$A + (A \cdot B)$	Given
$(A \cdot 1) + (A \cdot B)$	
$A \cdot (1 + B)$	
$A \cdot (1)$	
A	

Let's try another two-column proof that makes use of the above proof that $A + AB = A$:

$A + \overline{A}B$	Given
$(A + AB) + \overline{A}B$	
$A + (AB + \overline{A}B)$	
$A + (BA + B\overline{A})$	
$A + B(A + \overline{A})$	
$A + B(1)$	
$A + B$	

So we have proved that $A + \overline{A}B$ is equivalent to $A + B$.

1.3 DeMorgan's Theorems

Another very useful class of boolean algebra equivalences is given by DeMorgan's Theorems, which give us the power to break a negation of a product or a negation of a sum into smaller pieces. Here are the two forms:

Equivalence	Name	Comment
$\overline{AB} = \overline{A} + \overline{B}$	NAND to Negative-OR	
$\overline{A + B} = \overline{A} \cdot \overline{B}$	NOR to Negative-AND	

So now write a two column proof that starts with an application of one of DeMorgan's theorems on the given left hand side to prove the following equivalence:

$$\overline{\overline{A + BC} + \overline{A\overline{B}}} = A\overline{B}$$

Take care and go slowly. Remember that the length of the negation is forcing a precedence as if the terms under the bar were grouped together with parenthesis.

1.4 Truth Table to Simplified Expression

Suppose we have a desired digital logic circuit defined by the following truth table:

row	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

The sum of products form of Y is given by:

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Give reasons for each of the following steps in the proof that the above is equivalent to the simpler $AB + BC + AC$. The steps below may combine some basic identities of applications of the commutative property of addition and multiplication along with a more "substantive" step, so try and capture the spirit of the major steps in your answers. Remember that you can use, in a single step, any proof that has already been covered.

$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$	Given
$BC(\overline{A} + A) + A\overline{B}C + AB\overline{C}$	
$BC(1) + A\overline{B}C + AB\overline{C}$	
$BC + A\overline{B}C + AB\overline{C}$	
$B(C + A\overline{C}) + A\overline{B}C$	
$B(C + A) + A\overline{B}C$	
$BC + AB + A\overline{B}C$	
$BC + A(B + \overline{B}C)$	
$BC + A(B + C)$	
$BC + AB + AC$	
$AB + BC + AC$	

How many gates are required for the sum-of-products form and how many are required for the simplified form, assuming only 2-input AND and OR gates and 1-input NOT gates?

There are a number of drawbacks for the application of the previous exercises to obtaining simplified expressions when realizing combinational logic:

1. It will almost never be the case that a generous person has given you all of the steps, and you must only give the reasons for each step.
2. In fact, the simplified form will not be given at all, and a proof may require trying lots of possible paths to try and arrive at a simpler form.
3. It is relatively easy to make mistakes, and then an assumed simplified form may not, in fact, be equivalent to the original sum-of-products form.

2 Karnaugh Map Simplification

2.1 Truth Table Variation

Consider the following truth table for a desired boolean function:

row	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Now suppose that we re-write our truth table in a new form. In this form, in which we can represent up to 4 input variables, we will use both rows and columns to represent a matrix giving the possible combinations of the input variables, and then the entries in this two-dimensional matrix represent the values of the output (Y). We call this form of truth table a Karnaugh-map, or K-map.

Here is a template example where we have three input variables, A, B, and C:

	BC			
A	0 0	0 1	1 1	1 0
0	Y_0	Y_1	Y_3	Y_2
1	Y_4	Y_5	Y_7	Y_6

So the two possible values of A of 0 and 1 define the two rows of this truth table variant, and the four possible combinations of B and C together $(00, 01, 11, 10)$ define the columns of the K-map. The entries in the table are given by the Y_i, where the subscript gives the row number of the entry from the original truth table form. Notice the sequence of the inputs BC ... they do not proceed in binary counting order. That is why the sequence of Y outputs across the first row is Y_0, Y_1, Y_3, Y_2. After the case where BC is 01 we jump to the case where BC is 11, and then return to the case of 10. The thing to note about this sequence is that, when we proceed from case to case, only *1 bit changes from column to column*, never both.

Filling in a K-map from the truth table presented originally in this section, we get:

	BC			
A	0 0	0 1	1 1	1 0
0	0	0	1	0
1	0	0	1	0

Consider where the 1 values of the function "line up" in this alternate view. We can clearly see that this function has value 1 exactly when B and C both have value 1. So we can immediately write down the simplified equivalence that $Y = BC$. Note that, in this simple case, we can also see this from a boolean algebra simplification point of view. $Y = \overline{A}BC + ABC$ and we can conclude that $Y = (A + \overline{A}) \cdot BC = (1) \cdot BC = BC$.

Fill in the following K-map for the following truth table:

row	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

	BC			
A	0 0	0 1	1 1	1 0
0				
1				

Now give a simplified form for Y:

$Y =$

If you have three terms that are "or'd" together, look again. You should be able to express this in two terms. At the beginning of Lab03, your instructor will give you additional guidance on how to use the K-map to get the greatest simplification of this sort, based on the groundwork covered here.