

Appendix B – Digital Logic on pp. 700-712

The amazing reality is that the two fundamental components, gates and memory cells, are all that are needed for the logic of a digital computer. I would like to spend a little time discussing Appendix B – Digital Logic so that we have an understanding of what is going on at the most basic level in a computer.

Boolean algebra is the algebra used in maintaining and designing logical circuitry. Boolean variables can take on one of two values: 0 or 1.

The basic logical operations are AND, OR, and NOT and are represented by different symbols in different textbooks:

- 1) AND will be represented by an * (e.g. A*B)
- 2) OR will be represented by a + (e.g. A+B)
- 3) NOT will be represented by a ' (e.g. A')

Logical Addition (Logical OR)

0+0=0 0+1=1 1+0=1 1+1=1

Logical Multiplication (Logical AND)

0*0=0 0*1=0 1*0=0 1*1=1

Logical NOT

0'=1 1'=0

Note: Notationally, \overline{X} is X' is $\sim X$.

Let's examine the Boolean Operators Table on p. 702.

Table B.1 Boolean Operators

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

Definition: a gate is an electronic circuit that operates on one or more signals producing an output signal.

A gate is the fundamental building block of ALL digital logic circuits. Interconnections of these gates allow us to build simple or complicated logical functions. The following picture is the basic logic gates used in designing logic circuits.

The basic logic gates found on p. 704 are:



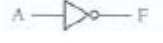

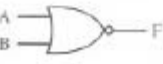
Name	Graphic Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{A}$ or $F = A'$	<table border="1"> <thead> <tr> <th>A</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{(AB)}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{(A + B)}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

Figure B.1 Basic Logic Gates

Basic axioms of Boolean algebra are:

- 1) $0 + X = X$
- 2) $1 + X = 1$
- 3) $X + X = X$
- 4) $X + X' = 1$
- 5) $0 * X = 0$
- 6) $1 * X = X$
- 7) $X * X = X$
- 8) $X * X' = 0$
- 9) $X'' = X$

Basic laws of Boolean algebra are:

- | | | |
|----------------------------------|--------------------------------|------------------|
| 1) $X + Y = Y + X$ | 2) $X * Y = Y * X$ | Commutative Law |
| 3) $X + (Y + Z) = (X + Y) + Z$ | 4) $X * (Y * Z) = (X * Y) * Z$ | Associative Law |
| 5) $X * (Y + Z) = X * Y + X * Z$ | | Distributive Law |

Note: * has precedence over +.

Other laws of Boolean algebra are:

- 6) $X + XZ = X$

P#1: Prove $X + XZ = X$ using: (a) the axioms and basic laws of Boolean algebra (b) perfect induction using truth tables.

7) $X(X+Y)=X$ 8) $(X+Y)(X+Z)=X+YZ$ 9) $X+X'Y=X+Y$ 10) $XY+YZ+Y'Z=XY+Z$

An important theorem is called DeMorgan's Theorem which is:

1) $(X+Y)'=X'Y'$

2) $(XY)'=X'+Y'$

Note: The complement of any boolean expression can be found by means of this theorem.

P#2: Find the complement of $X+(YZ)$

P#3: Show through the use of truth tables that you really found the complement of $X+(YZ)$.

Not all gates are used in implementation. Functionally complete sets of gates are those gates can implement any Boolean function:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR

P#4: How can you synthesize OR using only AND and NOT gates?

P#5: How can you synthesize AND, OR, and NOT gates using only NAND gates? How about only NOR gates?

When designing a logical circuit, the designer works from two sets of known values:

- (1) The input states that the logical circuit can take
- (2) The output states for each input

Problem: A logical network has two inputs, X and Y, and a single output Z. The relationship between inputs and outputs is as follows:

- 1) When X and Y are 0, Z is 0
- 2) When X and Y are 1, Z is 0
- 3) When X is 0 and Y is 1, Z is 0
- 4) When X is 1 and Y is 0, Z is 1

Express this using a truth table.

X	Y	Z
0	0	0
0	1	0
1	0	1
1	1	0

Add another column consisting of product terms - an AND formula whose structure is given by ANDing the row of input values to produce the desired output.

X	Y	Z	Product Terms
0	0	0	$X'Y'$
0	1	0	$X'Y$
1	0	1	XY'
1	1	0	XY

When Z equals 1, the product term is used in the sum-of-products.

Note: A product term containing all input variables is called canonic. A canonical expansion is the logical sum of the product terms where in the above case, $Z=1$. Therefore, the canonical expansion is: $Z=XY'$.

We are interested in finding the least expensive logic network; therefore, we might use some Boolean algebra to simplify the canonical expansion.

P#6: Write the Boolean expression (in sum-of-products form) for a logic network that will have a 1 output when: $X=1, Y=0, Z=0$; $X=1, Y=1, Z=0$; and $X=1, Y=1, Z=1$. All other combinations of input will produce an output of 0. Simplify the expression derived and draw a block diagram for the simplified expression.

Certain types of Boolean expressions lead to circuits that are more desirable from an implementation viewpoint.

Product term - a single variable or the logical product of several variables. The variables may or may not be complemented.

Sum term - a single variable or the logical sum of several variables. The variables may or may not be complemented.

X is both a sum term and a product term

$X+Y$ is a sum term

XY is a product term

$X+YX$ is neither

Sum-of-products - a product term or several product terms logically added

Product-of-sums - a sum term or several sum terms logically multiplied

We have already seen how to develop sum-of-product expressions. To develop product-of-sum expressions, do the following:

- 1) Construct a table of input and output values
- 2) Input value of 1 complements variable and 0 does not
- 3) Desired expression is the product of sum terms whose output is 0

P#7: Consider the following table:

Inputs		Output
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

Give a sum-of-products expression and a product-of-sums expression for the output F and draw the logical circuitry that realizes each expression that represents the output F.

NAND Gate Design

NAND gates are widely used in the design of modern computers. In general, NAND gates take less resistors than AND and NOR gates take less resistors than OR. To design a NAND-to-NAND two level gate network, we produce the sum-of-products expression. A sum-of-products expression can be drawn using AND gates at the first level and an OR gate at the second level. Finally, we can replace all AND and OR gates with NAND gates.

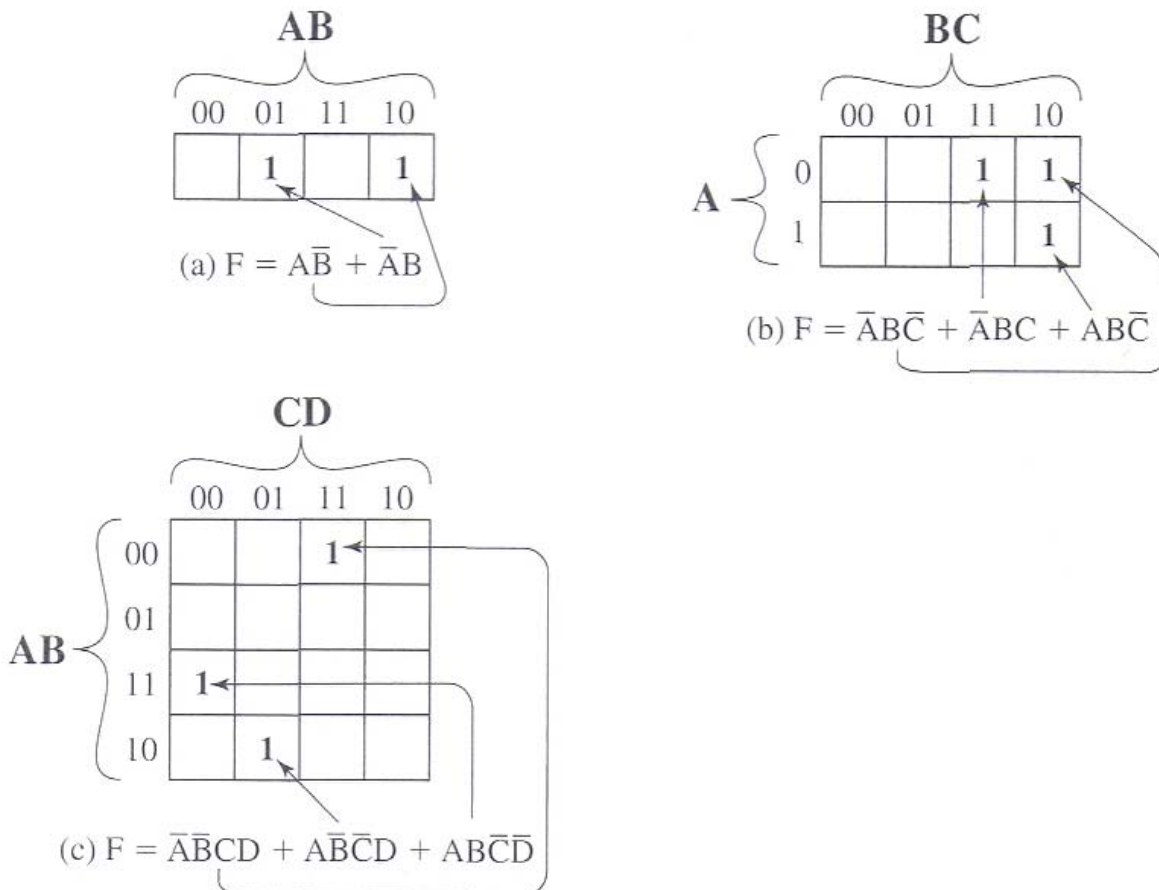
P#8: Develop the sum-of-products that describe the function of the following truth table (X,Y, and Z are inputs, A is output):

X	Y	Z	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Finally, draw the logical circuitry using only AND and OR logic. Then design the logical circuitry using only NAND gates.

Karnaugh Map Examples

A Karnaugh Map is a way of representing and simplifying a Boolean function for up to 4 variables. The map represents all possible combinations of n binary variables and thus contains 2^n possible squares (where n is the number of variables). Below you will see K-maps of 2, 3, and 4 variables. There are other ways to set up a K-map, but I will use the figures below for all in class, homework, and exam examples/questions.



As I mentioned earlier, a K-map can be used to simplify a Boolean function. In general, the rules are as follows:

- 1) Among the squares with a 1, find those that belong to the largest "block" of 1, 2, 4, or 8. Circle those blocks.
- 2) Select additional blocks so that every 1 in the square is a member of some block. 1's can be included in multiple blocks.

Here are examples of legal blocks:

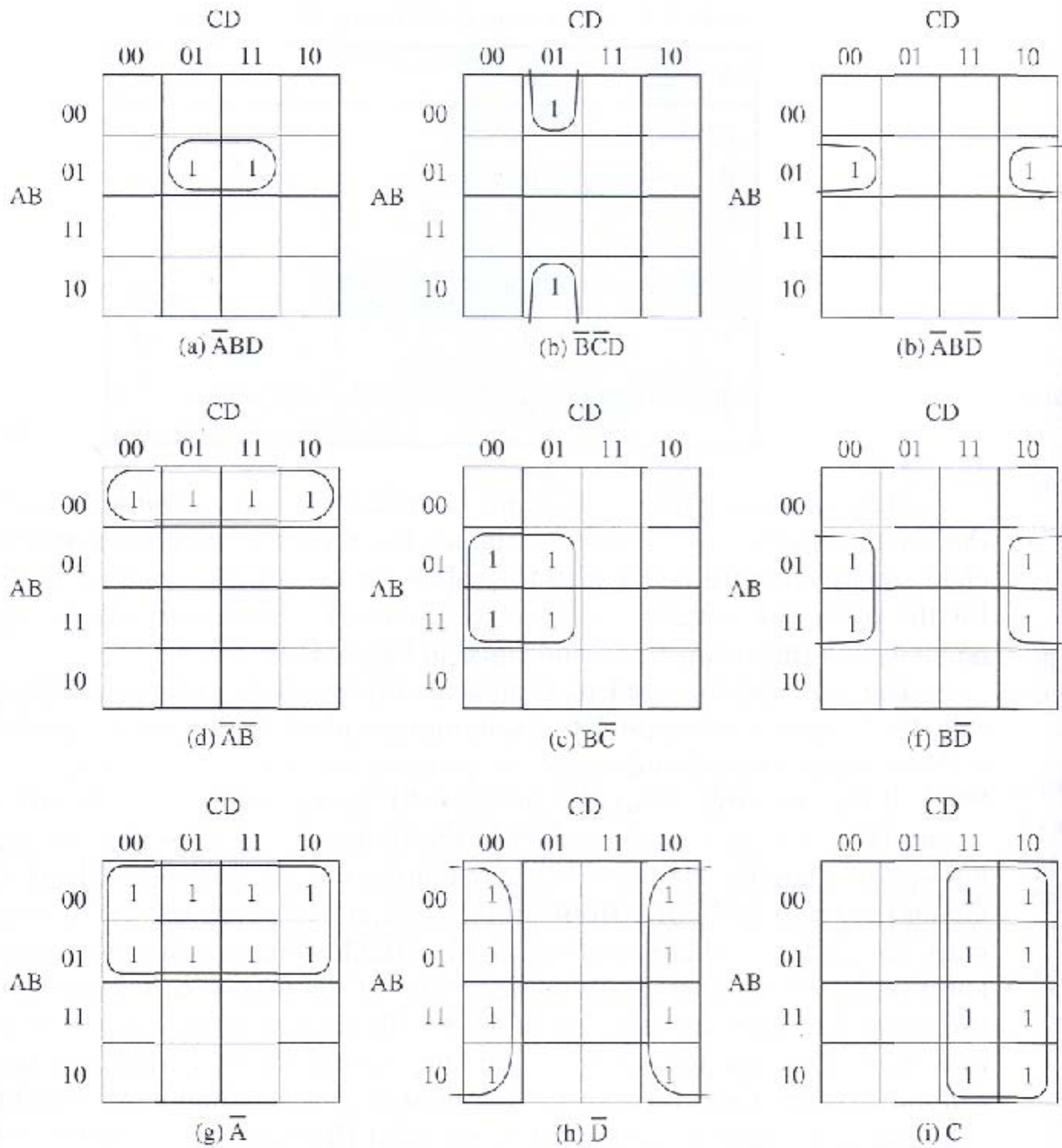


Figure B.8 The Use of Karnaugh Maps

A nice explanation of K-maps can be found at: <http://www.facstaff.bucknell.edu/mastascu/elessonshtml/Logic/Logic3.html>

P#9: Given the truth table below:

- a) Draw the K-map
- b) Give the Boolean function for the simplest sum-of-products form
- c) Draw the circuit using all NANDs

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0