FCPS Packets

Boolean Algebra

by Charles W. Brewer, July 2000
Based upon material developed by
Sally Bellacqua, Mary Johnson, and Janet Mulloy
Edited to Standard Notation by Connie Scoggins
Last Revision July 2016

Fairfax County Public Schools
Fairfax, Virginia

INTRODUCTION

George Boole (1815-1864), the inventor of symbolic logic, showed that logic could be rendered as algebraic equations and that logic should be considered a part of mathematics, not as a part of metaphysics**.** Claude E. Shannon introduced the use of Boolean algebra in the analysis and design of circuits in 1937. That is, binary arithmetic can be done with pure logic, and pure logic can be done by circuits in computers.

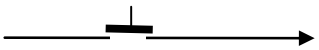
Boolean algebra can be performed with wiring diagrams, logic gates, Venn Diagrams, and truth tables. We will also develop the circuits for half-adders, full-adders, and 4-bit adders.

Wiring Diagrams

Boolean statements evaluate as either true or false. How does the computer circuitry do this? Some elemental ideas of electricity are all that we need to understand it. Although computers today use chips for transporting electrical impulses, we can think of these as consisting of wires that at one time may be carrying a current and at other times may not.

We can consider a **true** Boolean statement as current flowing through a switch in a wire and a **false** Boolean statement as current not flowing through. Thus, in a normally-open (NO) switch, a **true** condition is shown as a closed switch—the current flows. A **false** statement is shown as an open switch—the current does not flow.

NO Switch, open = false

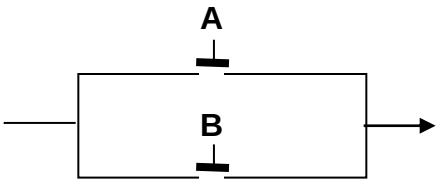


We can build a Boolean AND statement with switches in *series.* The only way that current can flow through the circuit (**true**) is when both switches are closed (**true**). An example is a car radio – it normally requires that the ignition switch be closed as well as radio being turned on to operate.

$A \bullet B$: NO Switches in Series

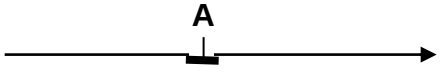
The Boolean OR statement can be built with switches in *parallel*. Current would flow (**true**) when one, the other, or both switches are closed. An example is the interior light of a car – it comes on when any of the doors are opened thus closing the switch for that door.

$A + B$: NO Switches in Parallel



The Boolean NOT statement is built with a switch (actually, a relay) that is *normally closed*. Current flows (true) when the relay is not energized and stops (false) when it is energized.

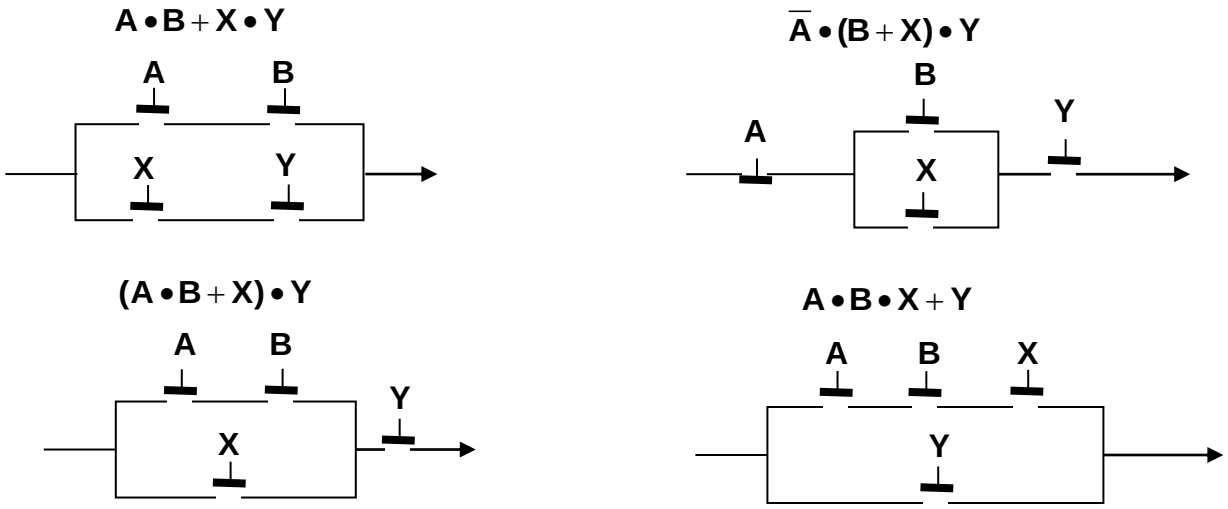
$\overline{A}$: NC Switch



NC – normally closed

In logic, AND is represented by $\bullet$, OR by $+$, and NOT by $^{-}$.
In Java, AND is represented by &&, OR by ||, and NOT by !.
When combined in expressions, the order of precedence is Parentheses, NOT, AND, OR.

EXAMPLES: A, B, X, and Y are Boolean expressions that may be evaluated as true or false. When combined in expressions, the order of precedence is parentheses, $\bar{}$, $\bullet$, $+$.



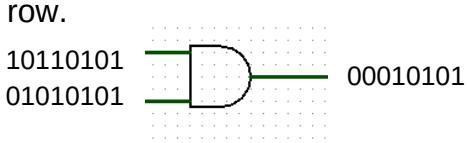
LOGIC CIRCUITS

Digital computers actually use low- and high-voltages to represent binary numbers. It is convenient for us to represent these low and high voltages by 0 and 1. A binary number such as 10101010 represents a sequence of *on* and *off* values. Similarly, the binary number 00000011 represents six *off* values followed by two *on* values.

When receiving and processing two input signals, the timing is important. Each input pair must be received simultaneously and must have time to produce the correct output before the next input pair appears. All CPUs run at a specified timing, measured in gigahertz, or billions of cycles per second.

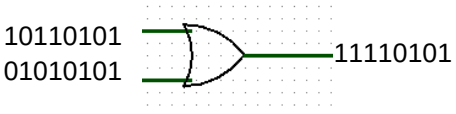
Our AND and OR gates receive 8 bits of information at two inputs and output the result.

In logic circuit notation we represent the AND gate as shown. Inside the gate, 8 pairs of bits are ANDed to produce the output. You apply the AND truth table eight times in a row.



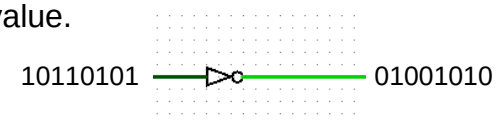
A	B	A **AND** B
0	0	
0	1	
1	0	
1	1	

Inside the OR gate, 8 pairs of bits are ORed to produce the output. You apply the OR truth table eight times in a row:



A	B	A **OR** B
0	0	
0	1	
1	0	
1	1	

A NOT gate has one input and one output. Each of the eight bits will be changed to its opposite value.

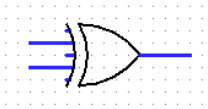


A	**NOT**-A
0	
1	

Other gates include the NAND (not-AND), NOR (not-OR), and XOR (excluded-OR).

The XOR operation's symbol is $\oplus$. The XOR truth table is

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

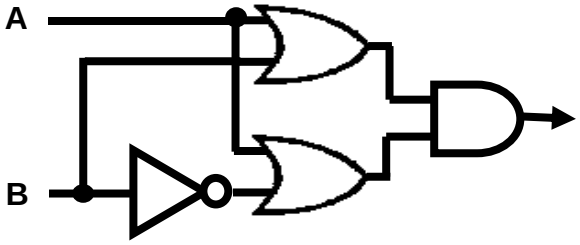
The XOR circuit picture is 

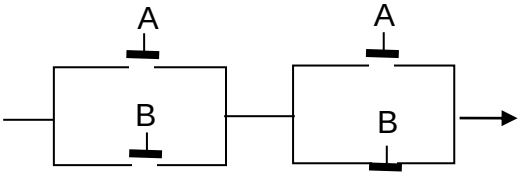
It has been proven that any logical circuit can be built from just NAND gates or from just NOR gates. With our current technology, NAND gates are the cheapest to produce.

EXAMPLE

This Boolean expression, logic circuit, and wiring diagram are all equivalent:

$$\mathbf{(A+B)\bullet(A+\overline{B})}$$





EXERCISES

If A, B, C, and D represent Boolean expressions, draw a logic circuit for the following.

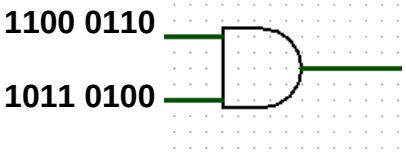
1. $(A+B)\bullet C$

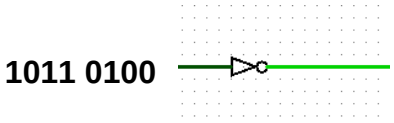
2. $A+\overline{B}\bullet C$

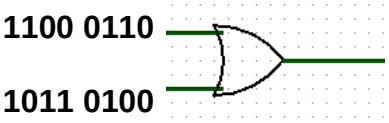
3. $A\bullet\overline{B\bullet C}+D$

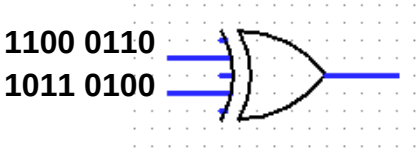
4. $A\bullet B+C\bullet D$

5. How would the pairs of sequences be processed by these gates?

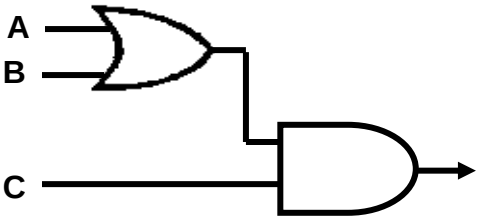


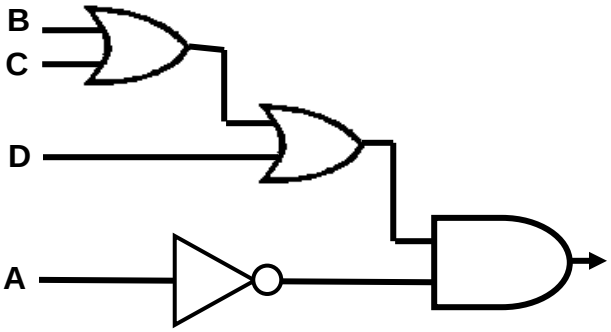


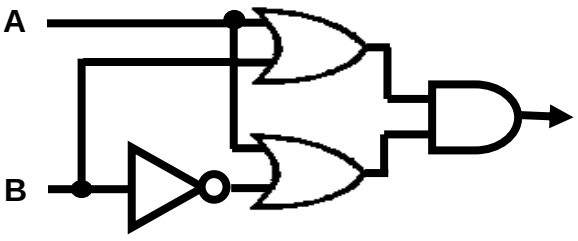


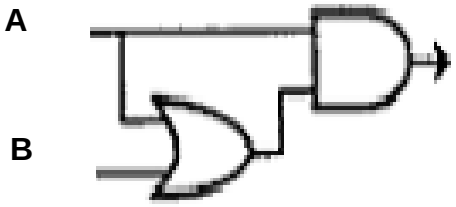


6. Write a Boolean expression for each of these logic circuits.





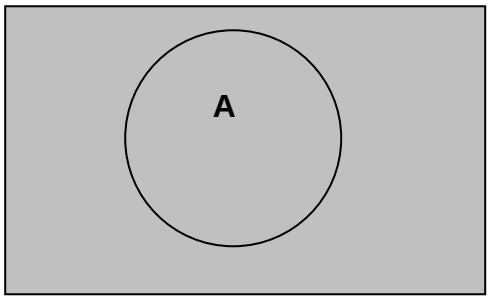




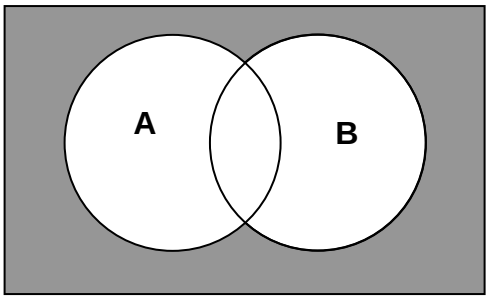
VENN DIAGRAMS

If we use 1 to represent the Universal set and 0 to represent the null or EMPTY set, then we can illustrate the following with Venn diagrams. Note that **AND** ($\bullet$) corresponds to the **intersection** of sets while **OR** ($+$) corresponds to the **union** of the sets.

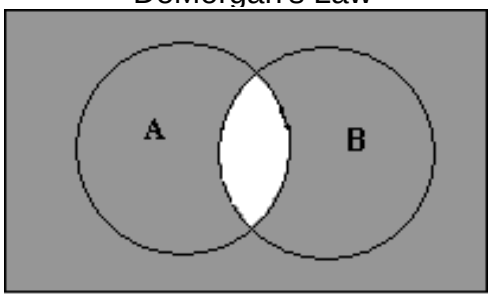
$\overline{\mathbf{A}} + \mathbf{A} = 1$



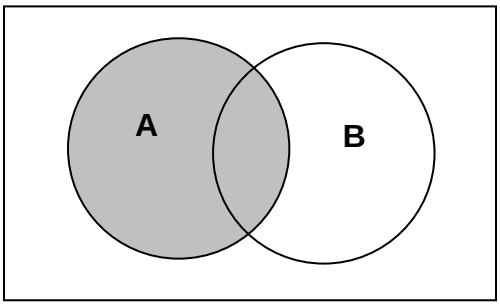
$\overline{\mathbf{A} + \mathbf{B}}$



$\overline{\mathbf{A}} + \overline{\mathbf{B}} = \overline{\mathbf{A} \bullet \mathbf{B}}$
DeMorgan's Law

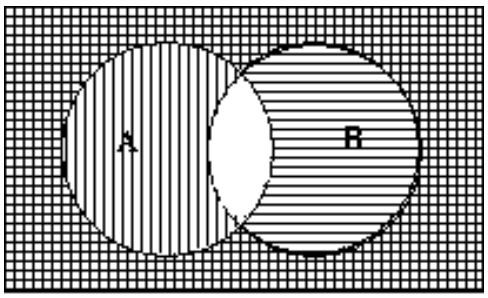


$\mathbf{A} + \mathbf{A} \bullet \mathbf{B}$



EXERCISES

Match each of the following sets to the Venn diagram to the right**.**

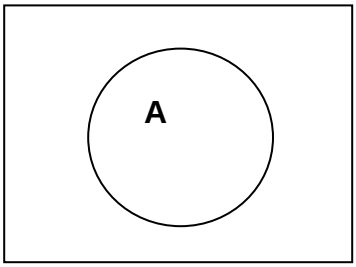


___1. Vertical lines
___2. Horizontal lines
___3. Shaded area
___4. The unshaded area
___5. Crosshatched area
___6. The uncrosshatched area

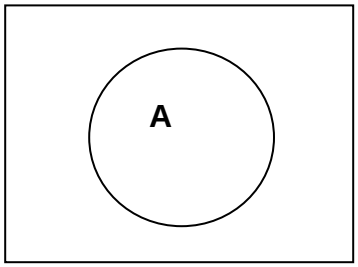
A. $A + B$
B. $A \bullet B$
C. $\overline{A}$
D. $\overline{B}$
E. $\overline{A} + \overline{B}$
F. $\overline{A} \bullet \overline{B}$

Shade a Venn diagram to represent each boolean expression.

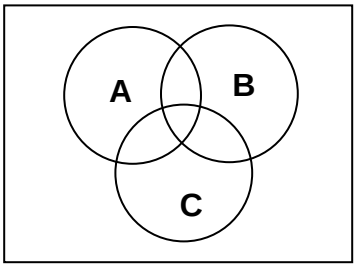
$A \bullet A$



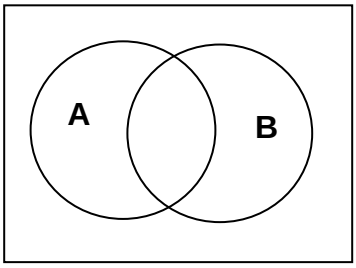
$A \bullet \overline{A}$



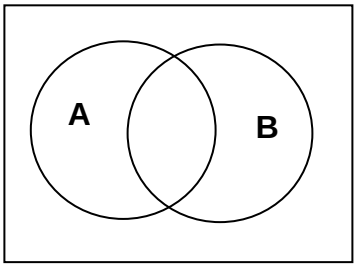
$(A \bullet B) \bullet C$



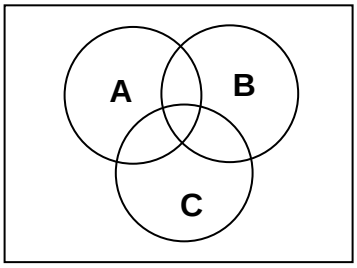
$\overline{A + B}$



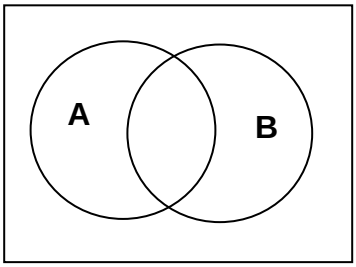
$\overline{A \bullet B}$



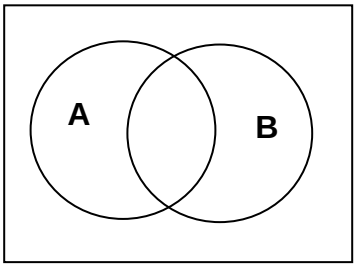
$A \bullet (B + C)$



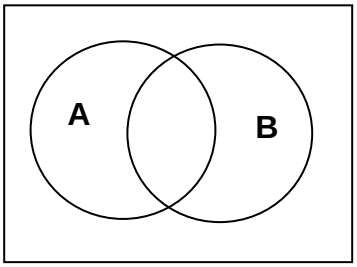
$\overline{A} \bullet \overline{B}$



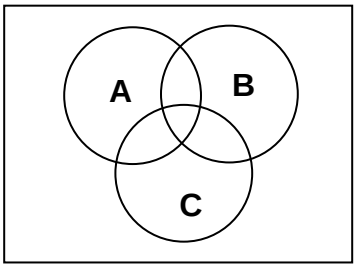
$\overline{A} + \overline{B}$



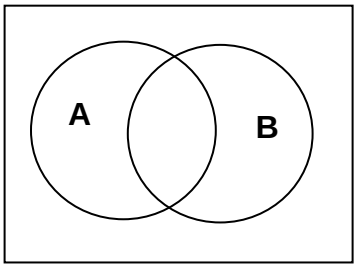
$A \bullet (A + B)$



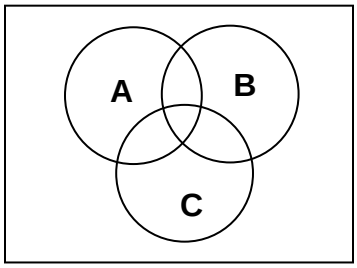
$(A + B) \bullet (B + C)$



$A + (\overline{A} \bullet B)$



$\overline{A} \bullet \overline{B} \bullet \overline{C}$



TRUTH TABLES

There are four possible combinations for two switches that are either open or closed. Using a 0 for open (no current flowing or FALSE) and a 1 for closed (current flowing or TRUE), we can represent all of these combinations with a truth table.

A	B	$A \bullet B$	$A + B$	$\overline{A}$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

With three elements we would have eight possibilities. Could you list them?

A && (A || B)= A

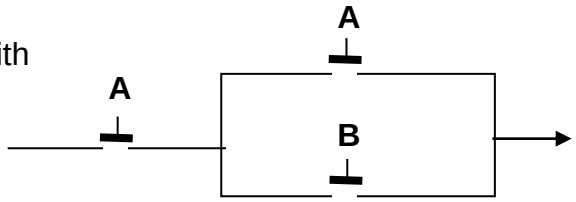
We could illustrate this Java identity with a Boolean truth table.

A	B	$A + B$	$A \bullet (A + B)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Notice that we have demonstrated that this identity is true, since the last column has the same truth as column A.

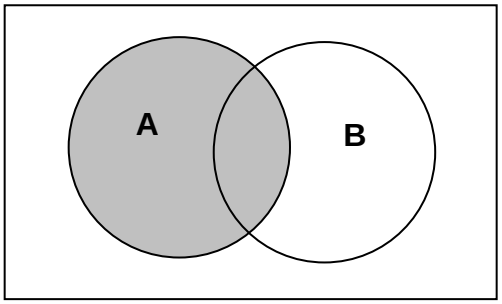
We could also illustrate this with

a wiring diagram,



a logic diagram,

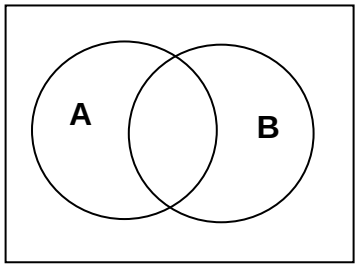
or a Venn diagram



EXERCISES

1. Given the identity: $\mathbf{A \bullet (\overline{A} + B) = A \bullet B}$

ILLUSTRATE IT WITH A VENN DIAGRAM:

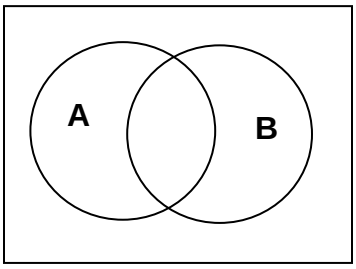


PROVE IT WITH A TRUTH TABLE

A	B	$\overline{A}$	$\overline{A}+B$	$A \bullet (\overline{A}+B)$	$A \bullet B$

2. Given the identity: $\mathbf{A + A \bullet B = A}$

ILLUSTRATE WITH A VENN DIAGRAM:



PROVE WITH A TRUTH TABLE

A	B	$A \bullet B$	$A + A \bullet B$

3. Use a truth table to prove: $\mathbf{(A + B) \bullet (\overline{A} + C) = (A \bullet C) + (\overline{A} \bullet B)}$

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

BASIC LAWS OF BOOLEAN ALGEBRA

These laws simplify boolean expressions. A, B, and C are booleans. 0 is False, 1 is True.

1. IDENTITY
$$A+0=A$$
$$A\bullet 1=A$$

2. INVERSE
$$A+\overline{A}=1$$
$$A\bullet\overline{A}=0$$

3. INVOLUTION
$$\overline{\overline{A}}=A$$
not (not-A) equals A

4. IDEMPOTENT
$$A+A=A$$
$$A\bullet A=A$$

5. BOUNDEDNESS
$$A+1=1$$
$$A\bullet 0=0$$

6. COMMUTATIVE
$$A+B=B+A$$
$$A\bullet B=B\bullet A$$

7. ASSOCIATIVE
$$A+(B+C)=(A+B)+C$$
$$A\bullet(B\bullet C)=(A\bullet B)\bullet C$$

8. DISTRIBUTIVE
$$A+(B\bullet C)=(A+B)\bullet(A+C)$$
$$A\bullet(B+C)=(A\bullet B)+(A\bullet C)$$

9. ABSORPTION
$$A+(A\bullet B)=A$$
$$A\bullet(A+B)=A$$

10. DERIVED
$$A+(\overline{A}\bullet B)=(A+\overline{A})\bullet(A+B)=A+B$$
$$A\bullet(\overline{A}+B)=(A\bullet\overline{A})+(A\bullet B)=A\bullet B$$

11. DeMORGAN'S
$$\overline{A+B}=\overline{A}\bullet\overline{B}$$
The NOT of the ORs equals the AND of the NOTS.
"Since it is false that either thing is true, then both things must be false."

$$\overline{A\bullet B}=\overline{A}+\overline{B}$$
The NOT of the ANDs equals the OR of the NOTS.
"Since it is false that two things together are true, at least one of them must be false."

Boolean algebra also has precedence rules.

The order of precedence is: parentheses (), NOT $\bar{}$, AND $\bullet$, XOR $\oplus$, and OR $+$.
Example: A $+$ B $\bullet$ C is evaluated by the order of precedence as A $+$ (B $\bullet$ C).

SIMPLIFYING A BOOLEAN EXPRESSION USING THE LAWS

As in regular algebra, Boolean algebraic expressions can be simplified by using the laws. Here are two examples:

$X + \overline{Y} + X + Y =$		$X \bullet \overline{X \bullet Y} =$	
$X + X + \overline{Y} + Y =$	Commutative	$X \bullet (\overline{X} + \overline{Y}) =$	DeMorgan's
$(X + X) + (\overline{Y} + Y) =$	Associative	$(X \bullet \overline{X}) + (X \bullet \overline{Y}) =$	Distributive
$X + 1 =$	Idempotent and inverse	$0 + (X \bullet \overline{Y}) =$	Inverse
$1 =$	Boundedness	$X \bullet \overline{Y}$	Identity

EXERCISES

PART I. Match each expression to the law that can be used to simplify it, then simplify each expression.

___ 1. $X + 1 =$

___ 2. $\overline{X} \bullet (X + B) =$

___ 3. $\overline{X} \bullet \overline{X} =$

___ 4. $\overline{X} \bullet \overline{Y} =$

___ 5. $A + 0 =$

___ 6. $X \bullet 1 =$

___ 7. $(X \bullet Y) + \overline{X} =$

___ 8. $(X + Y) \bullet (X + Z) =$

___ 9. $(\overline{X} + \overline{Y}) \bullet \overline{X} =$

__ 10. $\overline{X} + X =$

__ 11. $\overline{A \bullet (B + C)} =$

__ 12. !((n<0) && (m > 10))

A. Commutative

B. Distributive

C. Identity

D. Inverse

E. Idempotent

F. Boundedness

G. Absorption

H. Associative

I. Involution

J. Derived

K. DeMorgan's

PART II. Simplify the following using the Laws of Boolean Algebra. Show the steps and the law used for each step.

1. $Z \bullet ((\overline{X} + Y) + \overline{Y})$

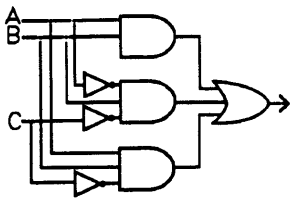
2. $\overline{(\overline{X} + Y)} + (X \bullet \overline{Y})$

3. $\overline{(X + Y) \bullet X}$

4. $(M + N) \bullet (M + P) + \overline{M}$

PART III.

a. Write the Boolean expression for the (unsimplified) logic circuit.

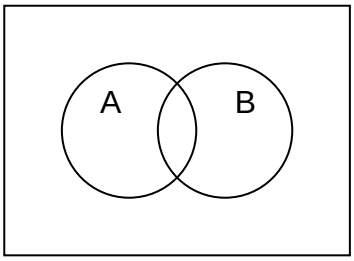


b. Simplify the expression using Boolean identities.

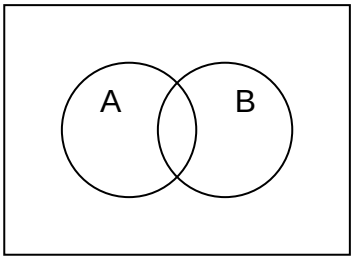
c. Draw the simplified logic circuit.

Part IV. Draw an appropriate Venn diagram for each of the following. Shade the area indicated by the expression. Using the Venn diagram and/or the Boolean laws, write the expression in its simplest form.

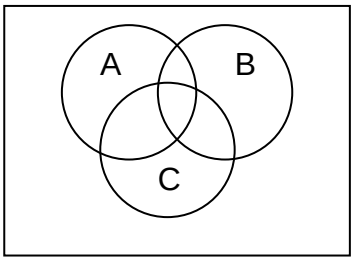
1. $\overline{\overline{\overline{A \bullet B} \bullet A}} =$



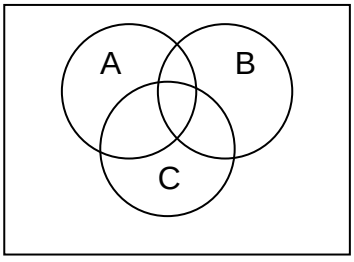
2. $A \bullet (\overline{A} + A \bullet B)$



3. $\overline{A} \bullet \overline{B} \bullet \overline{C} + A + B + C$

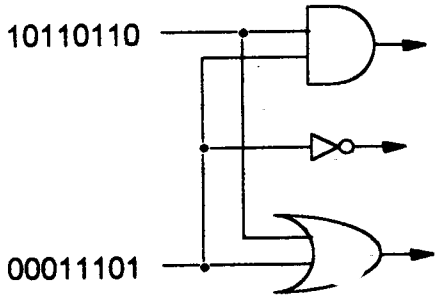


4. $\overline{(\overline{A} + \overline{B} + \overline{C}) \bullet C})$



REVIEW EXERCISES

1. Determine the output of each gate in the following logic diagram.



_____

_____

_____

2. Given the indicated input values for X, Y, and Z, determine the value of each of the following Boolean expressions.

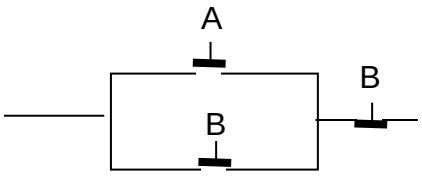
X = 11100100

Y = 00111110

Z = 00110010

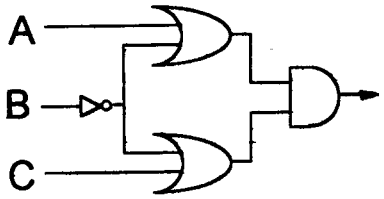
$X + Y$	_____
$\overline{Z} \bullet Z$	_____
$Y \bullet Z$	_____
$\overline{X} \bullet \overline{Y} + \overline{Z}$	_____
$\overline{X + Y + Z}$	_____

3. Write the Boolean expression for this WIRING DIAGRAM.



4. Draw the LOGIC CIRCUIT for $(A + \overline{B}) \bullet (B + \overline{C})$

5. Write the Boolean expression that describes the LOGIC CIRCUIT to the right.



6. Complete the truth table to prove $\overline{A} + (A \bullet \overline{B}) = (\overline{A} + \overline{B})$

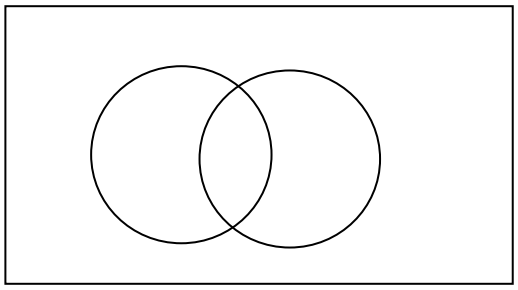
A	B	$\overline{A}$	$\overline{B}$	
0	0			
0	1			
1	0			
1	1			

7. Given: $(A + B) \bullet (\overline{A} + B)$

a. Draw the equivalent logic circuit.

b. Simplify the expression using Boolean algebra. Show your work.

c. Shade the Venn diagram to illustrate the expression



8. Apply DeMorgan's Law:

a. $\overline{(A == B) + (C == D)}$

b. $(X\,!= Y) \bullet (X > Z)$

c. It is not true that he took both Algebra and CS. →

9. Prove the following identities with truth tables.

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

A	B	
0	0	
0	1	
1	0	
1	1	

$A + (\overline{A} \bullet B) = A + B$

A	B	
0	0	
0	1	
1	0	
1	1	

10. Write Boolean expressions for the lettered parts of the Venn diagram in terms of the sets X, Y, and Z.

a. $\overline{X} \bullet Y \bullet Z$

b. _____

c. _____

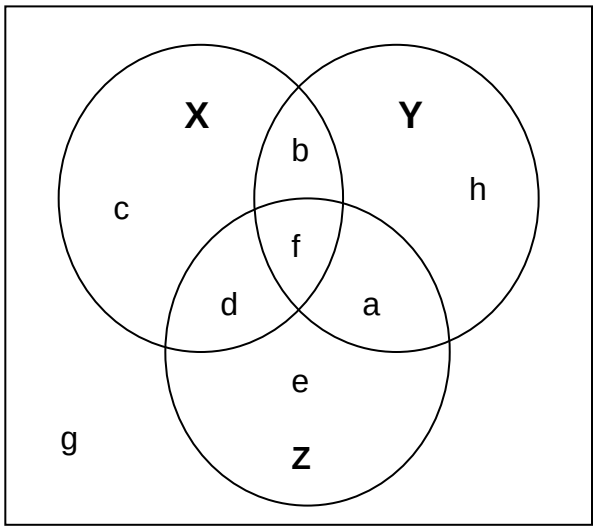
d. _____

e. _____

f. _____

g. _____

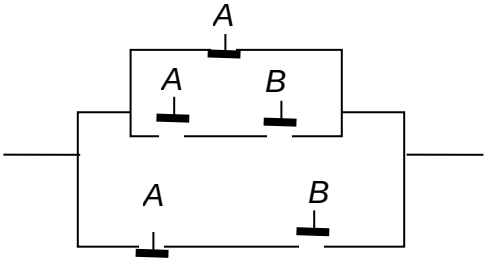
h. _____



11. Draw a logic circuit for $\overline{A \bullet B}$

Draw a logic circuit for $\overline{A} + \overline{B}$

12. Write the Boolean expression for the circuit below.

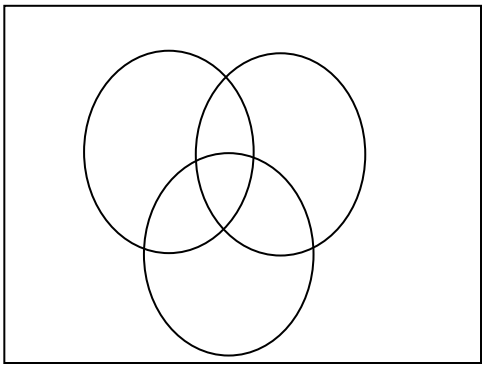


13. GIVEN: $(A+B+C)\bullet(A+B)\bullet(A+C)$

a. Simplify the expression using Boolean algebra. Show your work.

b. Using the simplified expression, draw a wiring diagram.

b. Using the simplified expression, shade the Venn diagram



d. Using the simplified expression, draw the logic circuit.

---

14. GIVEN: $A\bullet B+B\bullet C+B\bullet\overline{C}$

a. Draw the given logic circuit

b. Simplify using Boolean algebra. Show your work.

FROM BOOLEAN ALGEBRA TO DIGITAL CIRCUITS

Half-adder

We want to design a circuit that gets the right answer for these four problems:

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 1\ 0 \end{array}$$

Carry Sum

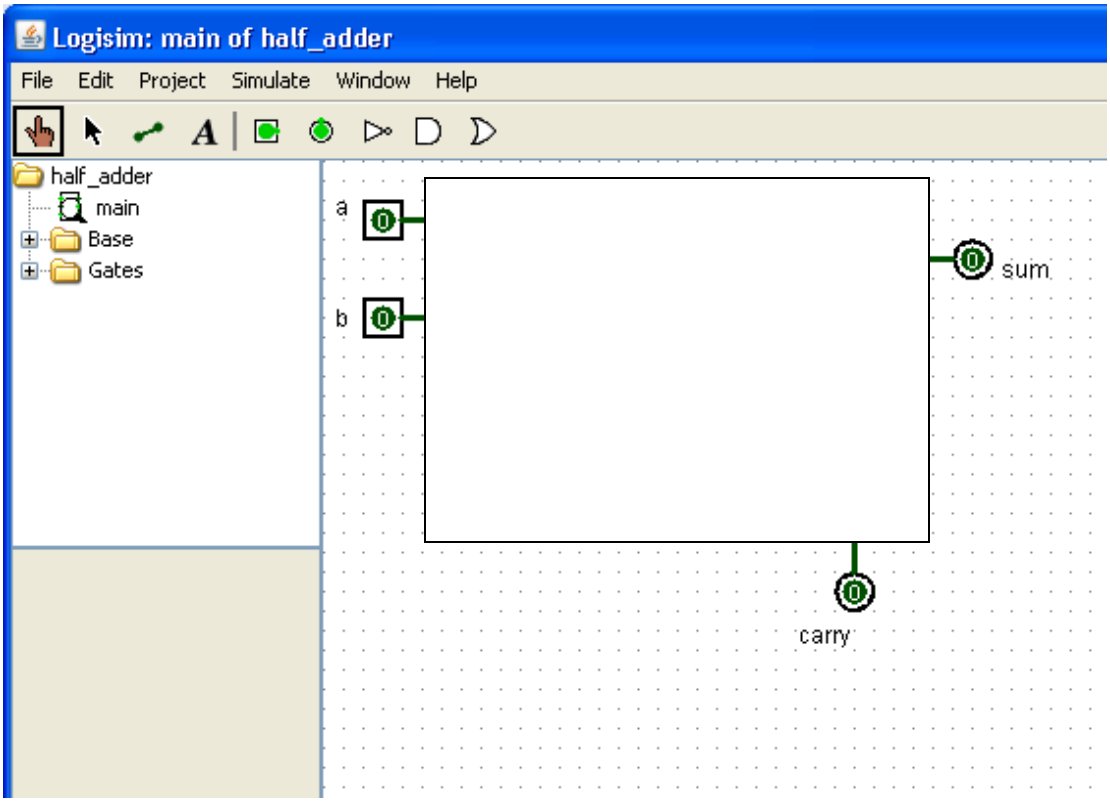
In other words, we want to design a circuit that produces this truth table:

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

1. What is the boolean expression for the Carry? _____

2. What is the boolean expression for the Sum? _____

Logisim (http://www.cburch.com/logisim/) is an educational tool for designing and simulating digital logic circuits. Here is a partial Logisim screenshot of the circuit diagram. Your job is to supply the missing gates to make the circuit work as described.



When you have a working circuit, save it as `half_adder`.

Full-Adder

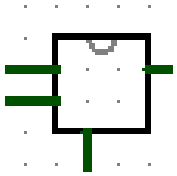
The half-adder does not handle the "carry" step. We want to design a circuit to get the right answer if we include an input carry. There are eight possible sums:

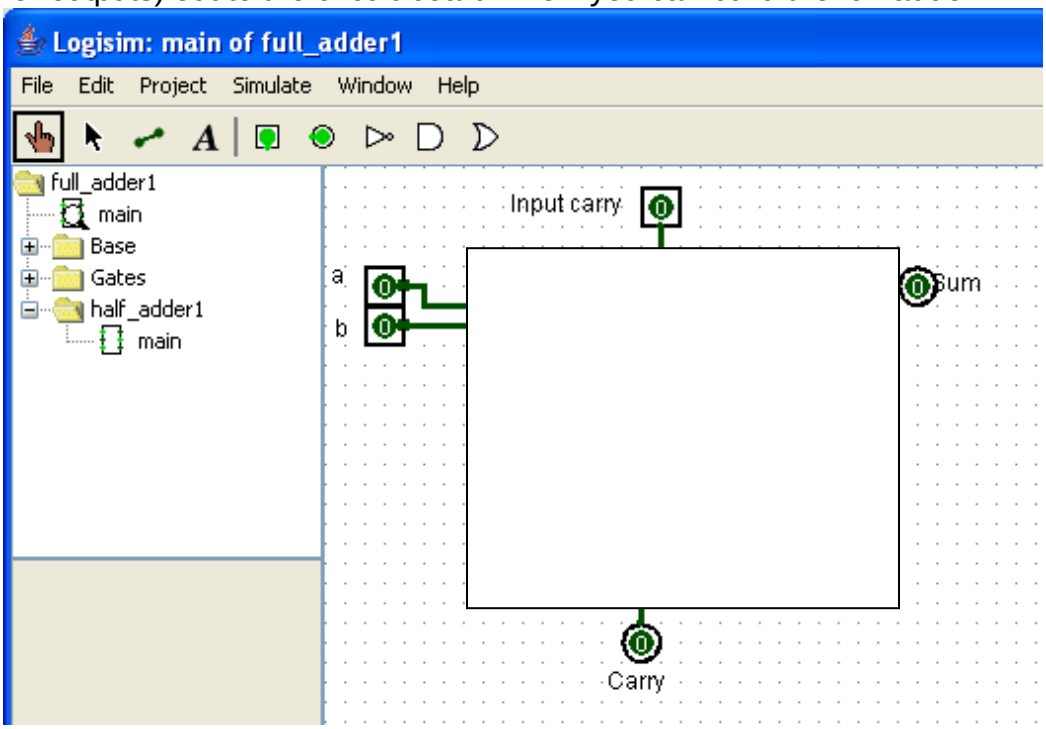
Or, in truth table form:								
	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0	1	1	1 0	1	1 0	1 0	1 1

Carry Sum

inputCarry	a	b	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Here is a partial Logisim screenshot of the circuit diagram. Your job is to supply the missing gates to make the full-adder work as described. This circuit uses two half-adders. In Logisim, go to Project, Load Library, Logisim and load the `half_adder.circ`. An icon appears on the left menu. You can click and drag your working half_adder (notice it has two pins for inputs and two pins for outputs) out to the circuit board. Now you can build the full-adder.





When you have a working circuit, save it as `full_adder`.

2-Bit Adder

Now you have a working full-adder. If you put two full-adders together in the right way, you can make a two-bit adder. In other words, you can handle sixteen addition problems:

from

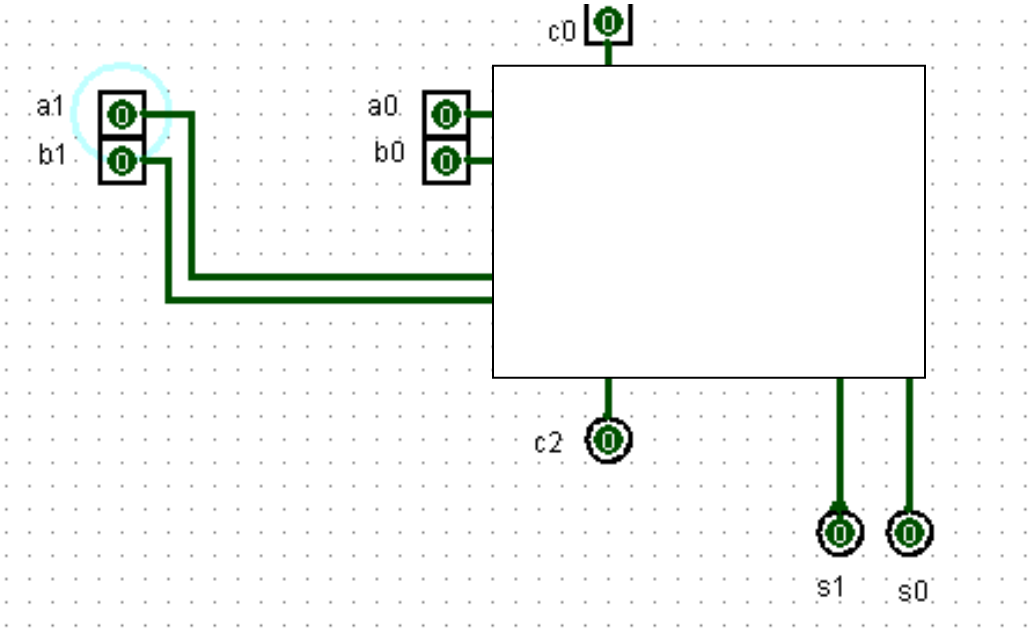
$$\begin{array}{r} 0\,0 \\ +\,0\,0 \\ \hline 0\,0 \end{array} \qquad \begin{array}{r} 0\,0 \\ +\,0\,1 \\ \hline 0\,1 \end{array} \qquad \begin{array}{r} 0\,1 \\ +\,0\,0 \\ \hline 0\,1 \end{array}$$

to

$$\begin{array}{r} 1\,1 \\ +\,0\,0 \\ \hline 1\,1 \end{array} \qquad \begin{array}{r} 1\,1 \\ +\,0\,1 \\ \hline 1\,0\,0 \end{array} \qquad \begin{array}{r} 1\,1 \\ +\,1\,1 \\ \hline 1\,1\,0 \end{array}$$

c2 s1 s0

The 2-bit adder uses two full-adders. Your job is to connect them in the right way



When you have a working 2-bit adder, save it.

Then construct a 3-bit adder, which uses three full-adders.

Then construct a 4-bit adder. Use it to solve addition problems all the way from

$$\begin{array}{r} 0\,0\,0\,0 \\ +\,0\,0\,0\,0 \\ \hline 0\,0\,0\,0 \end{array}$$

to

$$\begin{array}{r} 1\,1\,1\,1 \\ +\,1\,1\,1\,1 \\ \hline 1\,1\,1\,1\,0 \end{array}$$

c4 s3 s2 s1 s0