DIGITAL LOGIC

LECTURE 5

COVERAGE

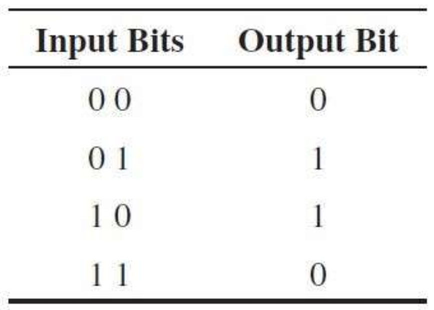
- Continuation of Lecture 4
- Quiz 1 (30 mins)
- Logic Gates
- Boolean Algebra
- Combinational Circuits
- Sequential Circuits
- Programmable Logic Devices

Continuation of Lecture 4

MODULO 2 OPERATION

- Modulo-2 addition (or subtraction) is the same as binary addition with the carries discarded, as shown in the table below

Modulo-2 operation.



Input Bits	Output Bit
0 0	0
0 1	1
1 0	1
1 1	0

- **Truth tables are widely used to** describe the operation of logic circuits, as you will learn in the next lecture.
- With two bits, there is a total of four possible combinations, as shown in the table.
- This particular table describes the modulo-2 operation also known as *exclusive-OR and can be implemented with a logic gate*

- Another convention is called *BCD (binary coded decimal"). In this case each decimal* digit is separately converted to binary. Therefore, since 7 = 0111_2 and 9 = 1001_2, then 79 = 01111001 (BCD).
- It is very often quite useful to represent blocks of 4 bits by a single digit. Thus in base 16 there is a convention for using one digit for the numbers 0,1,2,*. . .,15 which is called hexadecimal. It follows decimal for 0-9, then uses letters A-F.*
- How do you count in hexadecimal once you get to F? Simply start over with another column and continue as follows:
- …, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F,

  20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Binary to Hexadecimal

- Converting a binary number to hexadecimal is a straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol.

EXAMPLE:

1. Convert the following binary numbers to hexadecimal:

(a) 1100101001010111 (b) 111111000101101001

Solution

a.1100101001010111 b.00111111000101101001

$CA57_{16}$ **$3F169_{16}$**

Two zeros have been added in part (b) to complete a 4-bit group at the left.

Hexadecimal to Binary

- To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

EXAMPLE:

1. Determine the binary numbers for the following hexadecimal numbers:

(a) $10A4_{16}$ (b) $CF8E_{16}$ (c) 9742_{16}

Solution

(a)1000010100100 (b)1100111110001110 (c)1001011101000010

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

Quiz 1

QUIZ 1 (30 mins) (10 Marks)

1. Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ (b) $B2F8_{16}$ (c) $1C_{16}$ (d) $A85_{16}$

2. Add the signed numbers: 01000100, 00011011, 00001110, and 00010010.

3. Determine the 1's complement of each binary numbers:

(a) 00011010 (b) 11110111

4. Determine the 2's complement of each binary numbers:

(a) 00010110 (b) 11111100 (c) 10010001

Lecture 5

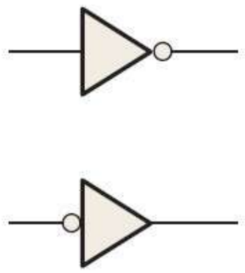
Logic Gates

Introduction

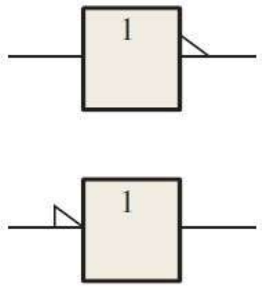
- Logic gates are one of the fundamental building blocks of digital systems.
- Most of the functions in a computer, with the exception of certain types of memory, are implemented with logic gates used on a very large scale.
- For example, a microprocessor, which is the main part of a computer, is made up of hundreds of thousands or even millions of logic gates
- The term *gate is used to describe a circuit that performs* a basic logic operation.

THE INVERTER/NOT GATE

- The inverter (NOT circuit) performs the operation called *inversion or complementation. The* inverter changes one logic level to the opposite level. In terms of bits, it changes a 1 to a 0 and a 0 to a 1.
- Standard logic symbols for the **inverter are shown below.** (a) shows the *distinctive shape symbols, and (b) shows the rectangular outline symbols.*



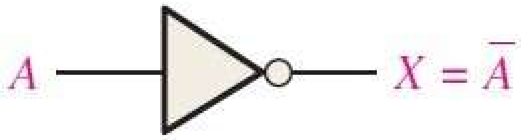
(a) Distinctive shape symbols with negation indicators



(b) Rectangular outline symbols with polarity indicators

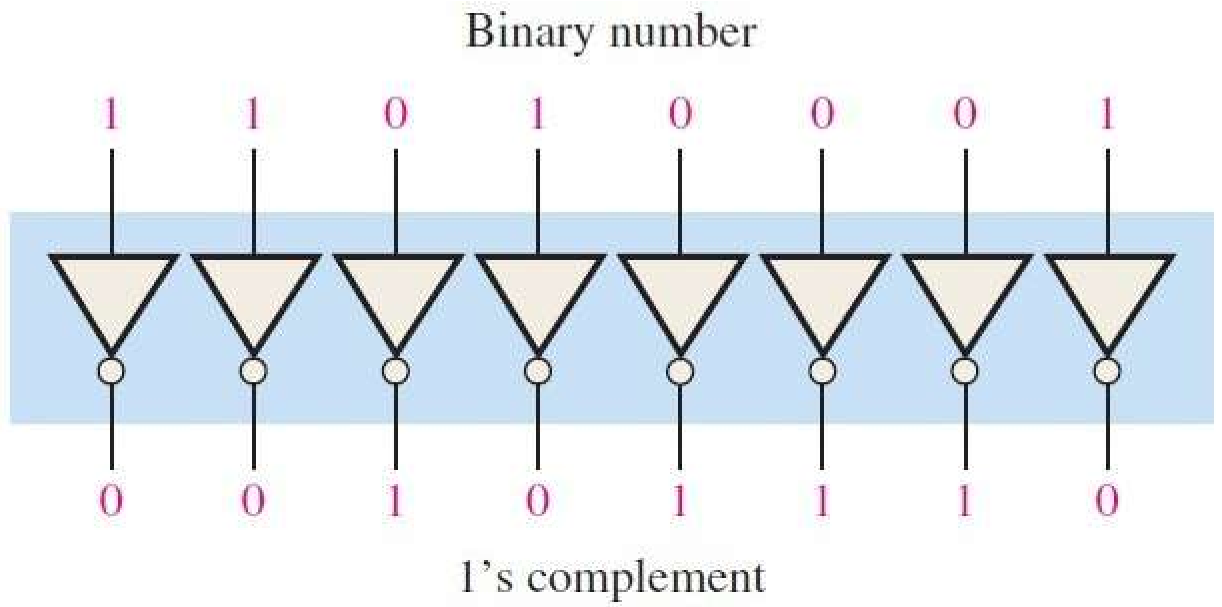
Inverter truth table.

Input	Output
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)



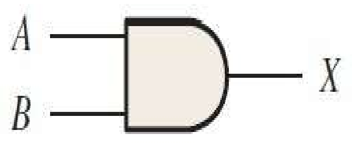
APPLICATION

- The Figure below shows a circuit for producing the 1’s complement of an 8-bit binary number.
- The bits of the binary number are applied to the inverter inputs and the 1’s complement of the number appears on the outputs.

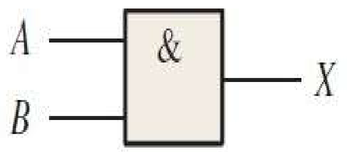


AND GATE

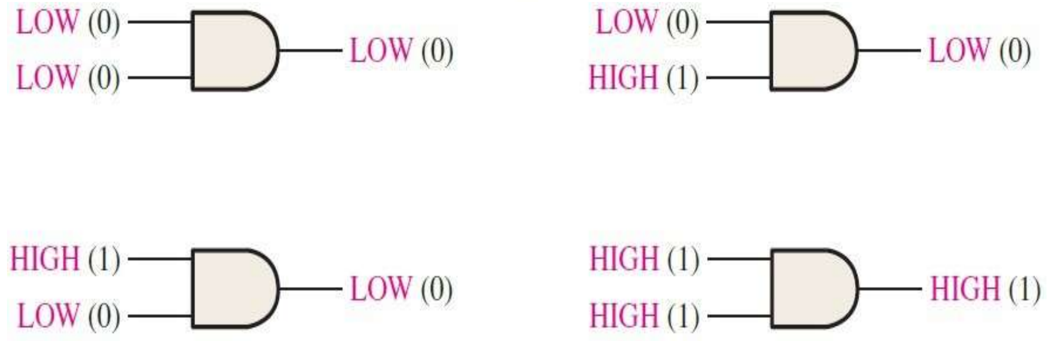
- The AND gate is one of the basic gates that can be combined to form any logic function.
- An AND gate can have two or more inputs and performs what is known as logical multiplication.
- The AND gate is indicated by the standard logic symbols shown in the Figure below. Inputs are on the left, and the output is on the right in each symbol.

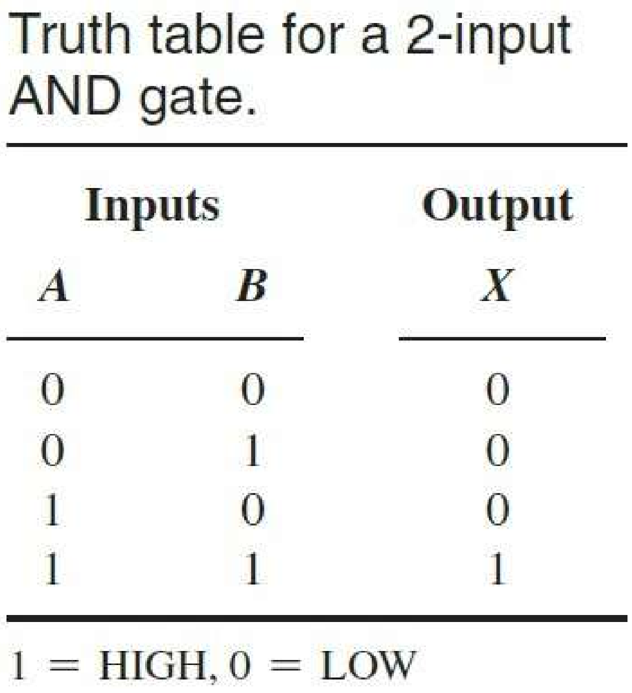


(a) Distinctive shape



(b) Rectangular outline with the AND (&) qualifying symbol





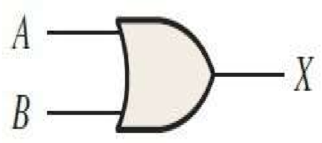
Truth table for a 2-input AND gate.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

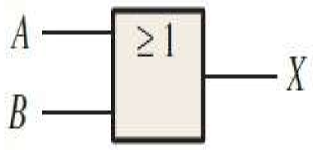
1 = HIGH, 0 = LOW

OR GATE

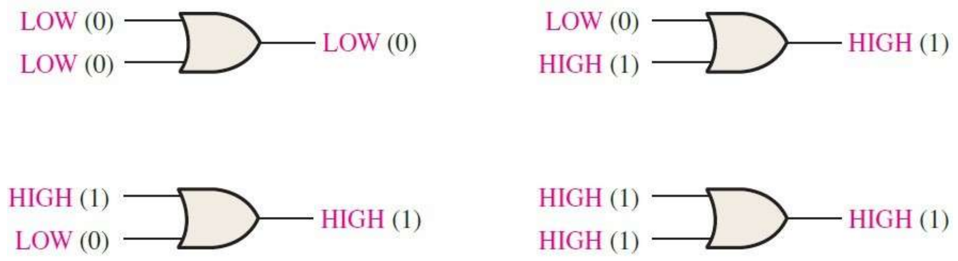
- The OR gate is another of the basic gates from which all logic functions are constructed.
- An OR gate can have two or more inputs and performs what is known as logical addition.
- An **OR gate has two or more inputs and one output, as indicated by the standard logic** symbols in the Figure below, where OR gates with two inputs are illustrated.
- An OR gate can have more than one inputs.



(a) Distinctive shape



(b) Rectangular outline with the OR (≥ 1) qualifying symbol



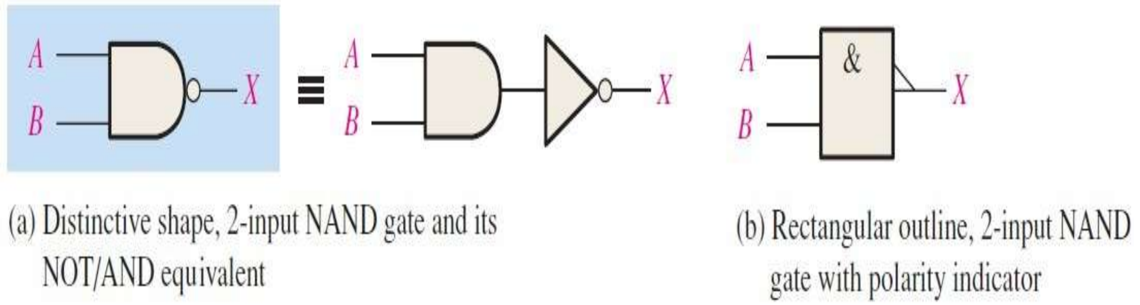
Truth table for a 2-input OR gate.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

1 = HIGH, 0 = LOW

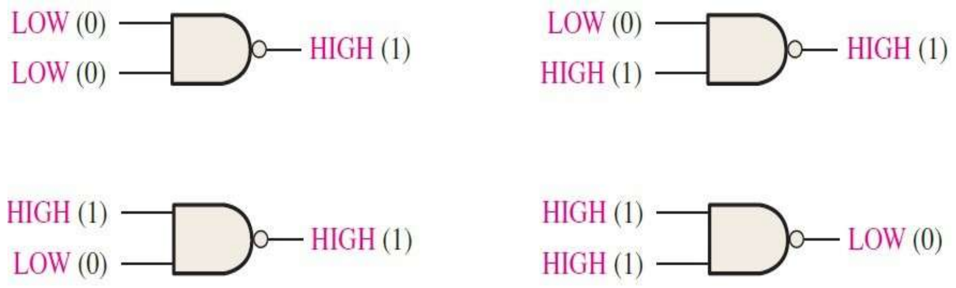
NAND GATE

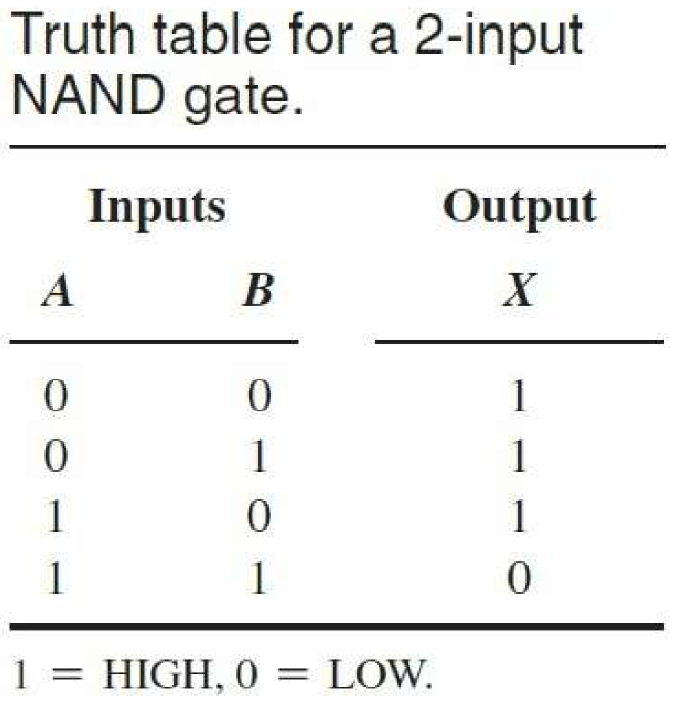
- The NAND gate is a popular logic element because it can be used as a universal gate; i.e., NAND gates can be used in combination to perform the AND, OR, and inverter operations.
- The term *NAND is a contraction of NOT-AND*. (*“NOT-ted” AND GATE*)
- The standard logic symbol for a 2-input NAND gate and its equivalency to an AND gate followed by an inverter are shown below



(a) Distinctive shape, 2-input NAND gate and its NOT/AND equivalent

(b) Rectangular outline, 2-input NAND gate with polarity indicator





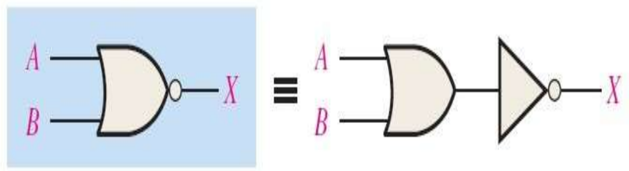
Truth table for a 2-input NAND gate.

Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

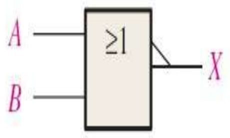
1 = HIGH, 0 = LOW.

NOR GATE

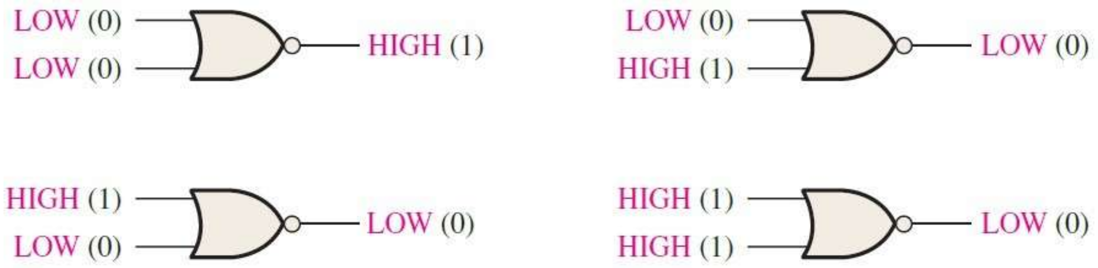
- The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations.
- The term *NOR is a contraction of NOT-OR and implies an OR function with an inverted* (complemented) output.
- The standard logic symbol for a 2-input NOR gate and its equivalent OR gate followed by an inverter are shown below



(a) Distinctive shape, 2-input NOR gate and its NOT/OR equivalent



(b) Rectangular outline, 2-input NOR gate with polarity indicator



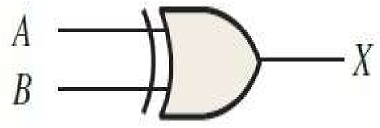
Truth table for a 2-input NOR gate.

Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

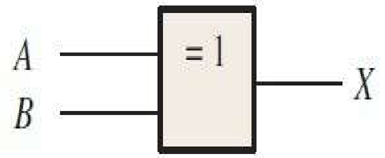
1 = HIGH, 0 = LOW.

EXCLUSIVE OR/NOR GATES (XOR/XNOR)

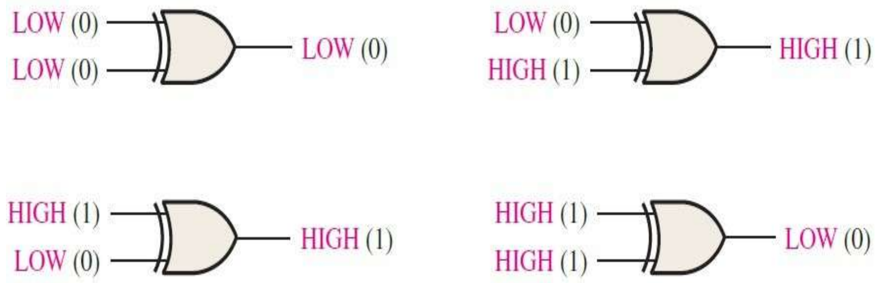
- Exclusive-OR and exclusive-NOR gates are formed by a combination of the other gates already discussed
- However, because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.
- Standard symbols for an exclusive-OR (XOR for short) gate are shown below.



(a) Distinctive shape



(b) Rectangular outline



Truth table for an exclusive-OR gate.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Algebra

OBJECTIVES

- The basic operations, laws and theorems of Boolean algebra.
- Use truth tables to prove Boolean theorems
- Express the output of a logic circuit using Boolean algebra.

Laws of Boolean Algebra

- A set of laws similar to Algebra
- Commutative Laws
- Associative Laws
- Distributive Laws
- Identity Laws
- Complement Laws

Commutative Laws

- Commutative Law of Boolean Addition

$$A + B = B + A$$

- Commutative Law of Boolean Multiplication

$$AB = BA$$

Associative Laws

- Associative Law of Boolean Addition

$$(A + B) + C = A + (B + C)$$

- Associative Law of Boolean Multiplication

$$(AB)C = A(BC)$$

Distributive Laws

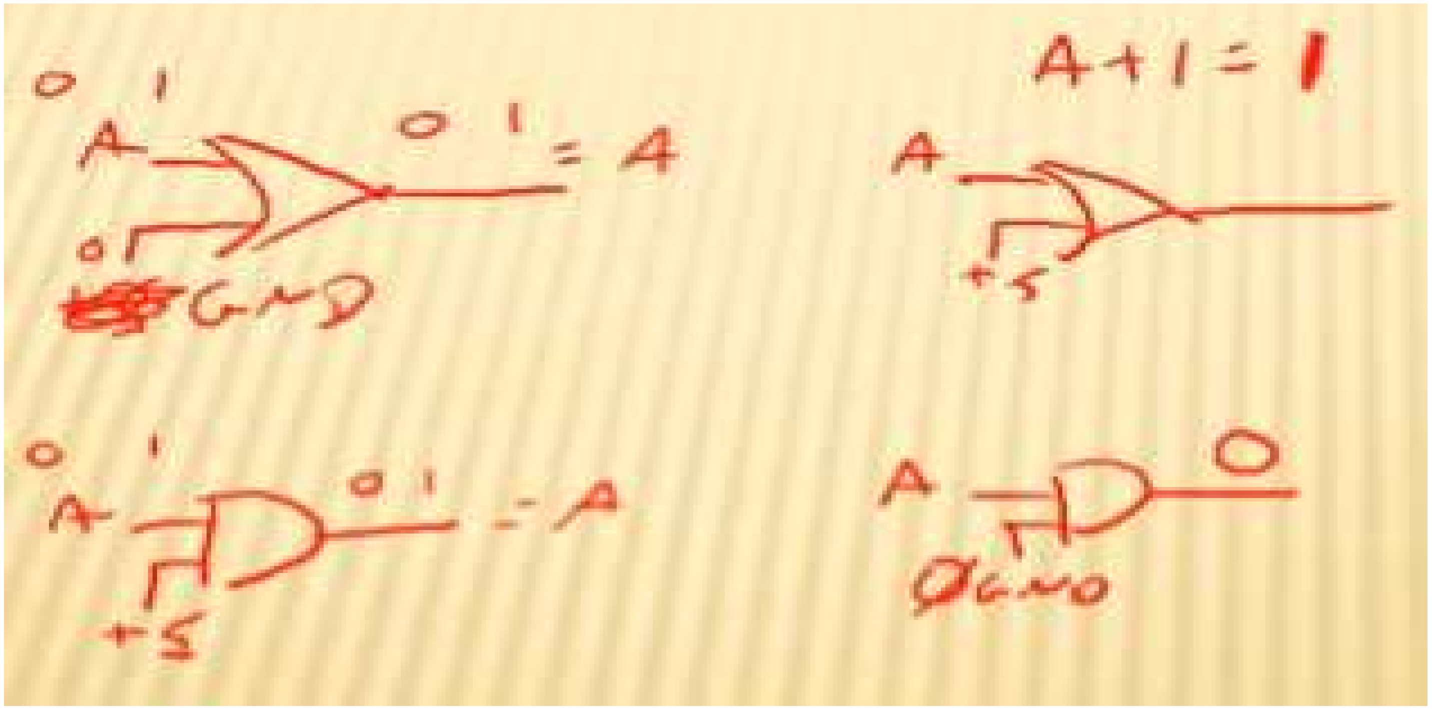
$$A(B + C) = AB + AC$$

$$A + BC = (A + B)(A+C)$$

- NOTE: The first distributive law is similar to arithmetic
- The second distributive law does not apply in arithmetic

Identity Laws

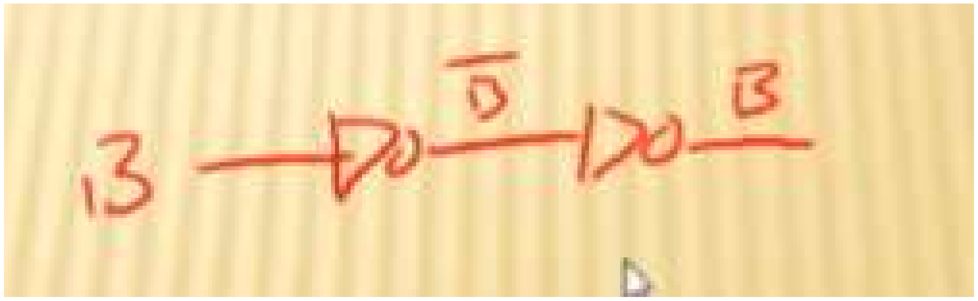
- A + 0 = A (Anything ORed with 0 is itself)
- A * 1 = A (Anything ANDed with 1 is itself)



Complement Laws

- A + A' = 1 (Anything ORed with its complement is 1
- A * A' = 0 (Anything ANDed with its complement is 0)
- Any variable inverted twice is itself

B'' = B



Dual Property

Statement	Dual
0' = 1	1' = 0
A + 1 = 1	A * 0 = 0
A + A = A	A * A = A
A + A' = 1	A * A' = 0

Using the Laws

- Simplify A + AB
- Apply the distributive law:

$$A + AB = A(1 + B)$$

- Apply the commutative law:

$$A(1 + B) = A(B + 1)$$

$$A(B + 1) = A(1) = A * 1$$

- By the identity law: A * 1 = A

Simplification of Combinational Logic Circuits

- Complex combinational logic circuits must be reduced without changing the function of the circuit.
- Reduction of a logic circuit means the same logic function with fewer gates and/or inputs.
- The first step to reducing a logic circuit is to write the Boolea Equation for the logic function.
- The next step is to apply as many rules and laws as possible in order to decrease the number of terms and variables in the expression.
- To apply the rules of Boolean Algebra it is often helpful to first remove any parentheses.
- After removal of the paretheses, common terms or factors may be removed leaving terms that can be reduced by the Boolean Algebra.

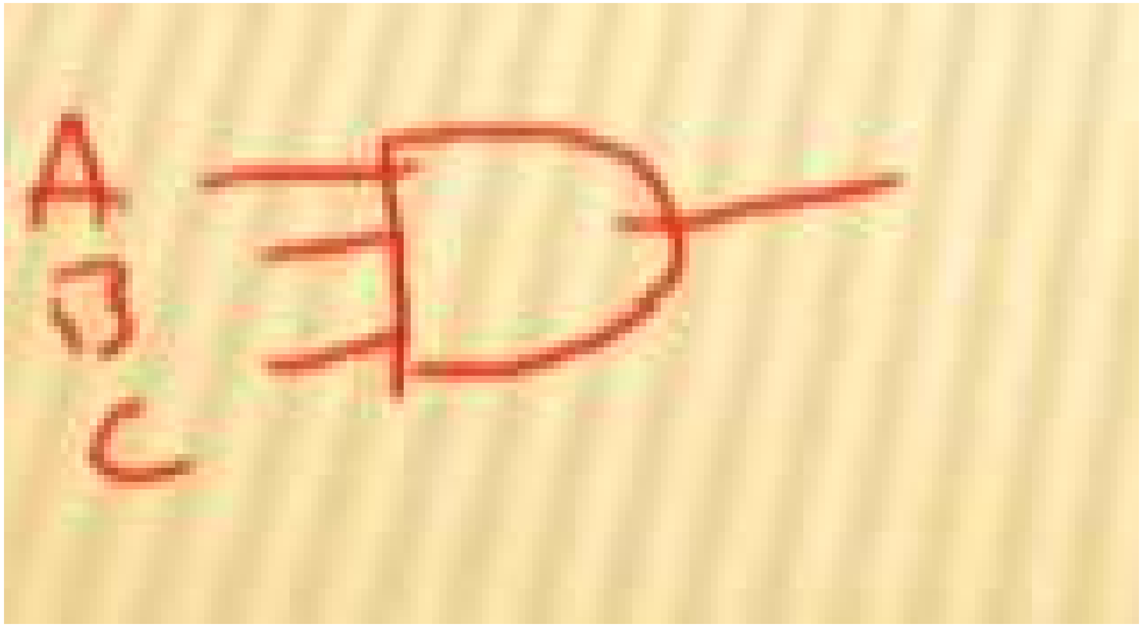
Simplification of Combinational Logic Circuits

- Construct a truth table to prove the simplified expression is the same.
- The final step is to draw the logic diagram for the reduced Boolean Expression.

- For example (AB)(BC)

Simplification of Combinational Logic Circuits

A	B	C	AB	BC	(AB)(BC)	ABC
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1


A
B
C

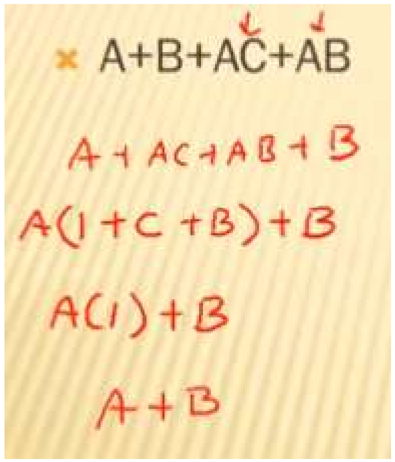
Simplification of Combinational Logic Circuits

- In some cases the question arises as to the order of operations. If an AND and an OR appear in the same expression, which is to be done first? The order of operations of Boolean Algebra are the same as standard algebra. The AND operation (same as multipication) is performed first. Of course, parentheses can be used to alter the order of operations just as in standard algebra. In the expression AB + C, A is ANDed to B then ORed with C. In the expression, A(B + C), B is ORed with C first, then ANDed with A.
- A function inside a parentheses must be accomplished first before any functions outside the parentheses.
- A bar over several variables can also act as a parentheses. Any function under the bar must be done before any function not under the bar

Simplification of Combinational Logic Circuits

- A + B + AC + AB

A	B	C	AC	AB	A + B + AC + AB	A + B
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	1	1
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	0	1	1
1	1	0	0	1	1	1
1	1	1	1	1	1	1



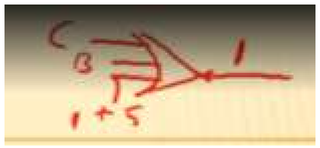
- A+B+AC+AB

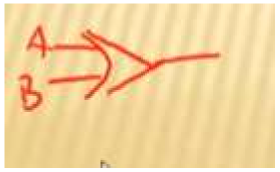
$A + AC + AB + B$

$A(1 + C + B) + B$

$A(1) + B$

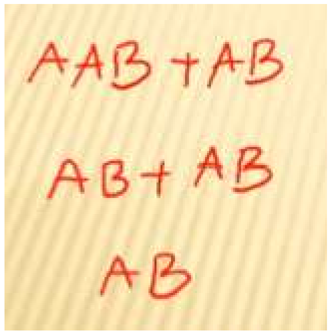
$A + B$





Simplification of Combinational Logic Circuits

- A(AB + B)

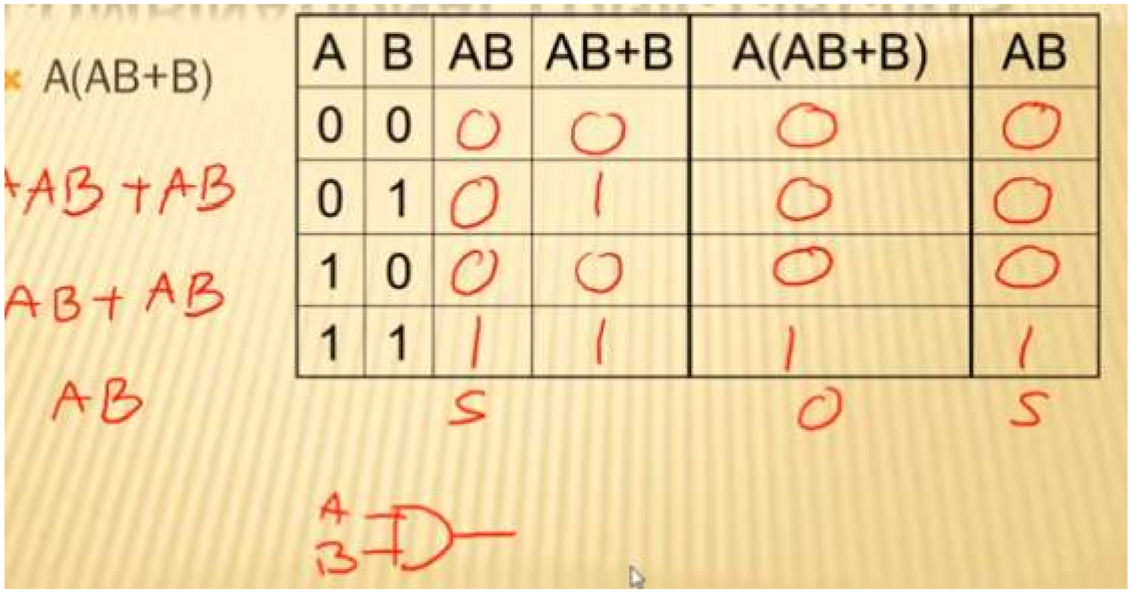


$AAB + AB$

$AB + AB$

AB

Simplification of Combinational Logic Circuits



A(AB+B)

AB + AB

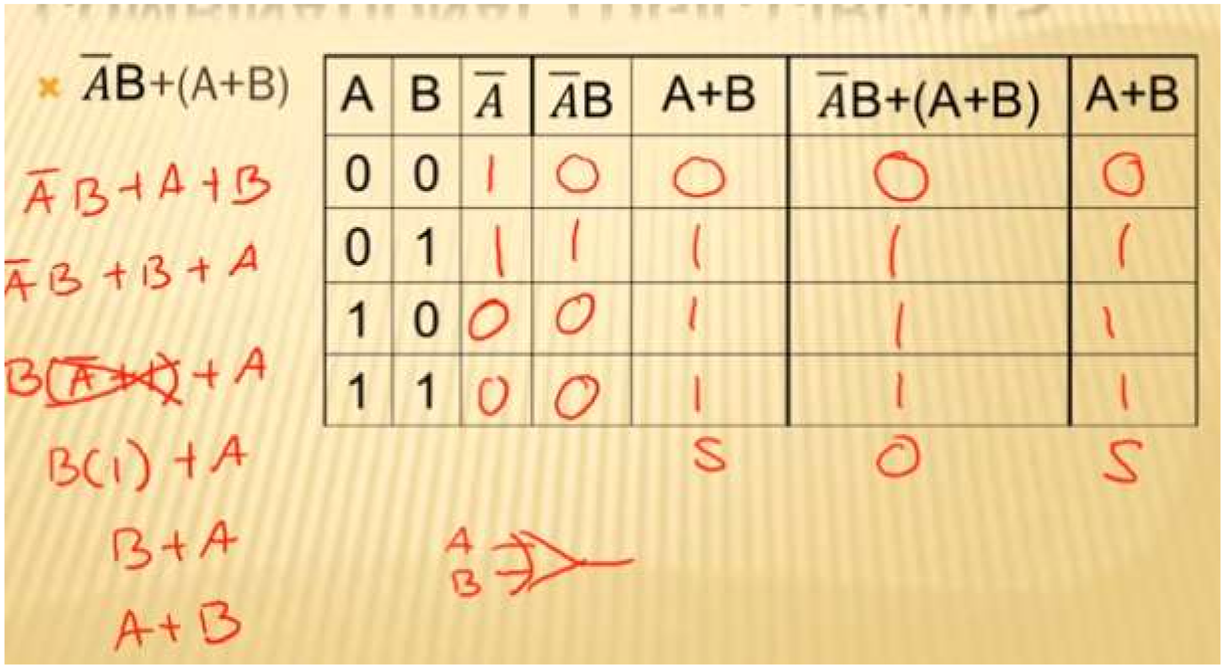
AB + AB

AB

A	B	AB	AB+B	A(AB+B)	AB
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	0	0	0
1	1	1	1	1	1
		S		O	S

Simplification of Combinational Logic Circuits

A’B+(A+B



- $\bar{A}B+(A+B)$

$\bar{A}B+A+B$

$\bar{A}B+B+A$

$B(\cancel{\bar{A}+1})+A$

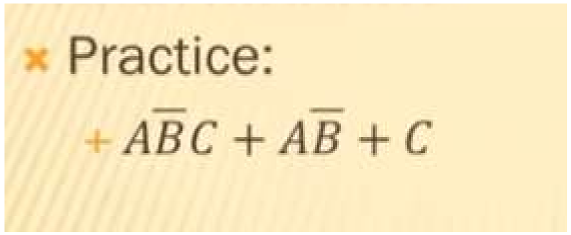
$B(1)+A$

$B+A$

$A+B$

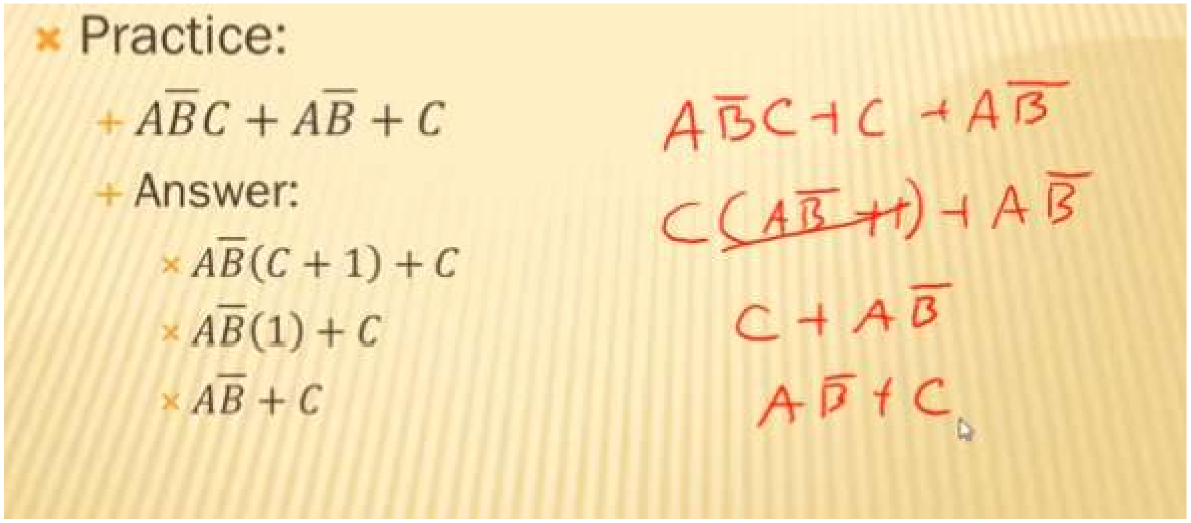
A	B	$\bar{A}$	$\bar{A}B$	A+B	$\bar{A}B+(A+B)$	A+B
0	0	1	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	1	0	0	1	1	1
				S	0	S

Simplification of Combinational Logic Circuits



- Practice:
  - $A\overline{B}C + A\overline{B} + C$

Simplification of Combinational Logic Circuits



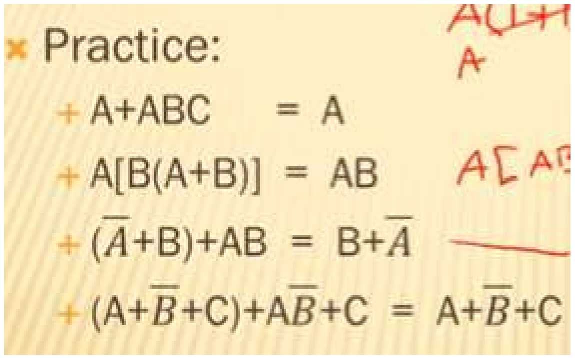
- Practice:
  - $A\overline{B}C + A\overline{B} + C$
  - Answer:
    - $A\overline{B}(C + 1) + C$
    - $A\overline{B}(1) + C$
    - $A\overline{B} + C$

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

$C(\cancel{A\overline{B} + 1}) + A\overline{B}$

$C + A\overline{B}$

$A\overline{B} + C$

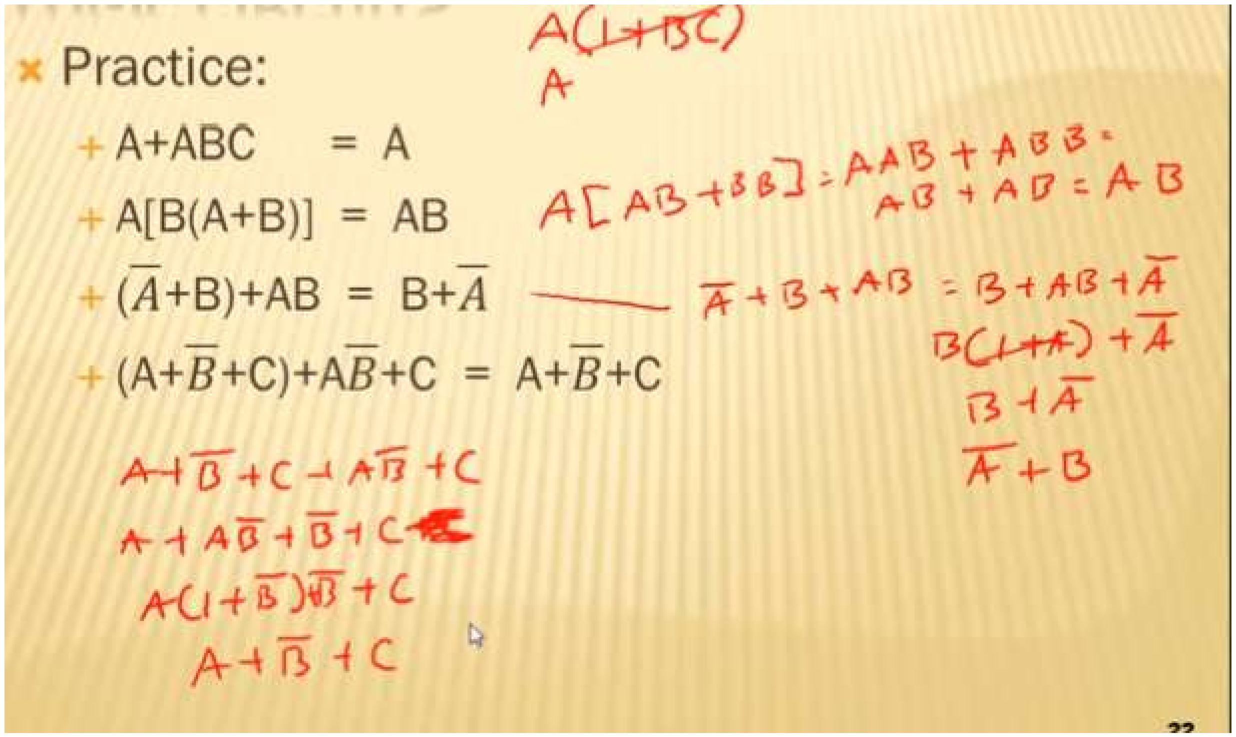


- Practice:
  - A+ABC = A
  - A[B(A+B)] = AB
  - $(\overline{A}+B)+AB = B+\overline{A}$
  - $(A+\overline{B}+C)+A\overline{B}+C = A+\overline{B}+C$

$A(1+$

A

$A[AB$



- Practice:
  - A+ABC = A
  - A[B(A+B)] = AB
  - $(\overline{A}+B)+AB = B+\overline{A}$
  - $(A+\overline{B}+C)+A\overline{B}+C = A+\overline{B}+C$

$A(\cancel{1+BC})$

A

$A[AB+BB] = AAB + ABB =$

$AB + AB = AB$

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

$B(\cancel{1+A}) + \overline{A}$

$B + \overline{A}$

$\overline{A} + B$

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

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

$A(1+\overline{B}) + \overline{B} + C$

$A + \overline{B} + C$

De Morgan's Theorem

- De Morgan's theorem allows large bars in a Boolean Expression to be broken up into smaller bars over individual variables.
- De Morgan's theorem says that a large bar over several variables can be broken between the variables if the sign between the variables is changed.

$$\overline{A \cdot B \cdot C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

$$\overline{A \cdot B \cdot C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

De Morgan's Theorem

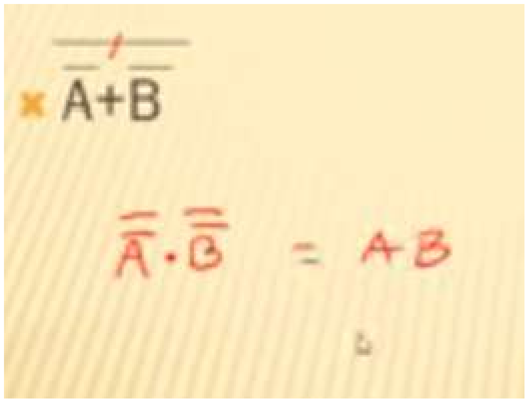
- De Morgan's theorem can be used to prove that a NAND gate is equal to an OR gate with inverted inputs.
- De Morgan's theorem can be used to prove that a NOR gate is equal to an AND gate with inverted inputs.
- In order to reduce expressions with large bars, the bars must first be broke up. This means that in some cases, the first step in reducing an expression is to use De Morgan's theorem

De Morgan's Theorem

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

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

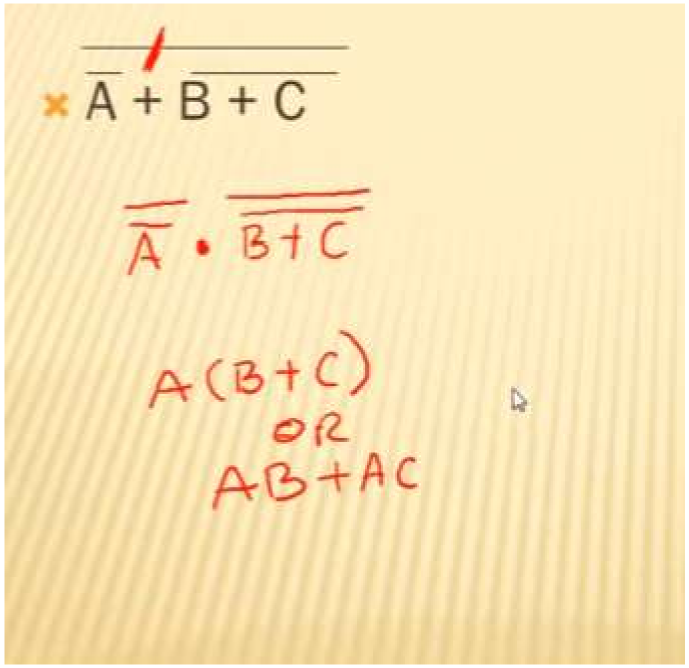
De Morgan's Theorem



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

$\overline{\overline{A}} \cdot \overline{\overline{B}} = AB$

De Morgan's Theorem



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

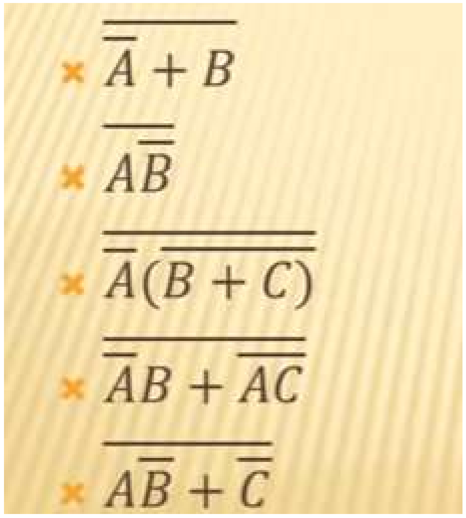
$$\overline{\overline{A}} \cdot \overline{\overline{B+C}}$$

$$A(B+C)$$

OR

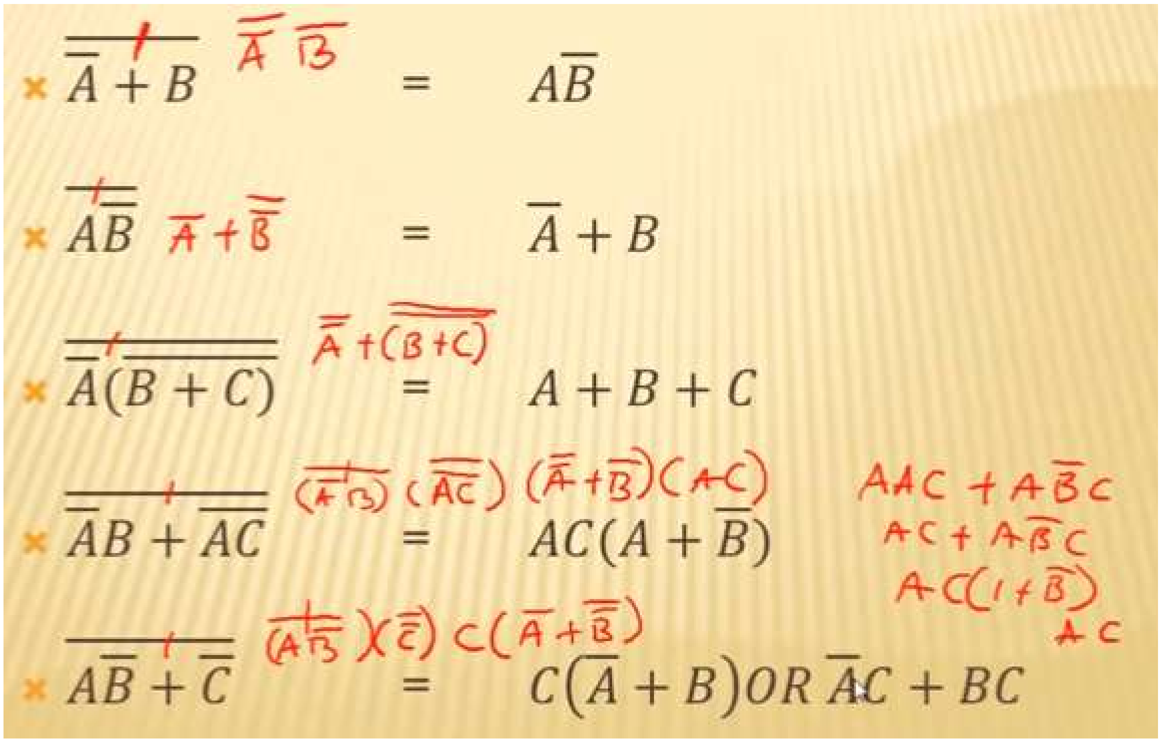
$$AB+AC$$

De Morgan's Theorem



- $\overline{\overline{A} + B}$
- $\overline{A\overline{B}}$
- $\overline{\overline{A}\,\overline{(B + C)}}$
- $\overline{\overline{A}B + \overline{AC}}$
- $\overline{A\overline{B} + \overline{C}}$

De Morgan's Theorem



- $\overline{\overline{A}+B}$ $\overline{\overline{A}}\,\overline{B}$ $=$ $A\overline{B}$
- $\overline{A\overline{B}}$ $\overline{A}+\overline{\overline{B}}$ $=$ $\overline{A}+B$
- $\overline{\overline{A}\,\overline{(B+C)}}$ $\overline{\overline{A}}+\overline{\overline{(B+C)}}$ $=$ $A+B+C$
- $\overline{\overline{A}B+\overline{AC}}$ $(\overline{\overline{A}B})(\overline{\overline{AC}})$ $(\overline{\overline{A}}+\overline{B})(AC)$ $=$ $AC(A+\overline{B})$

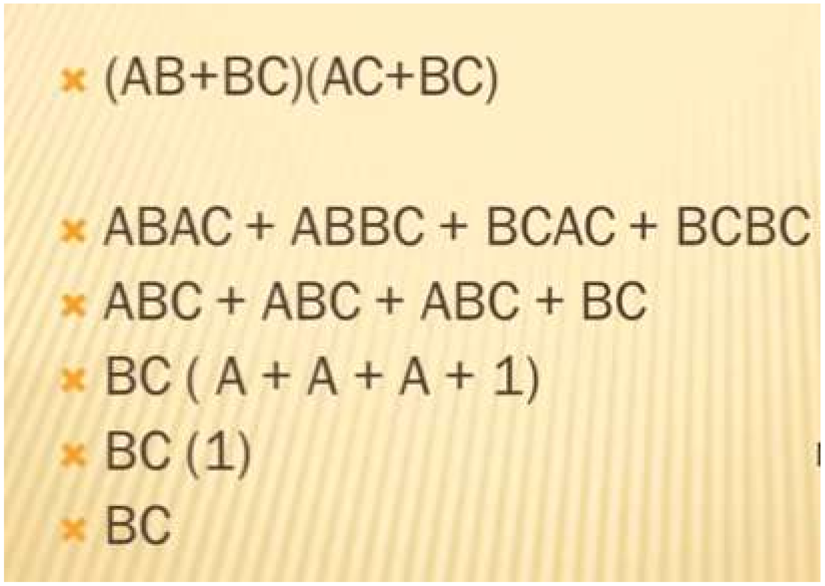
  $AAC+A\overline{B}C$

  $AC+A\overline{B}C$

  $AC(1+\overline{B})$

  AC
- $\overline{A\overline{B}+\overline{C}}$ $(\overline{A\overline{B}})(\overline{\overline{C}})$ $C(\overline{A}+\overline{\overline{B}})$ $=$ $C(\overline{A}+B)\,OR\,\overline{A}C+BC$

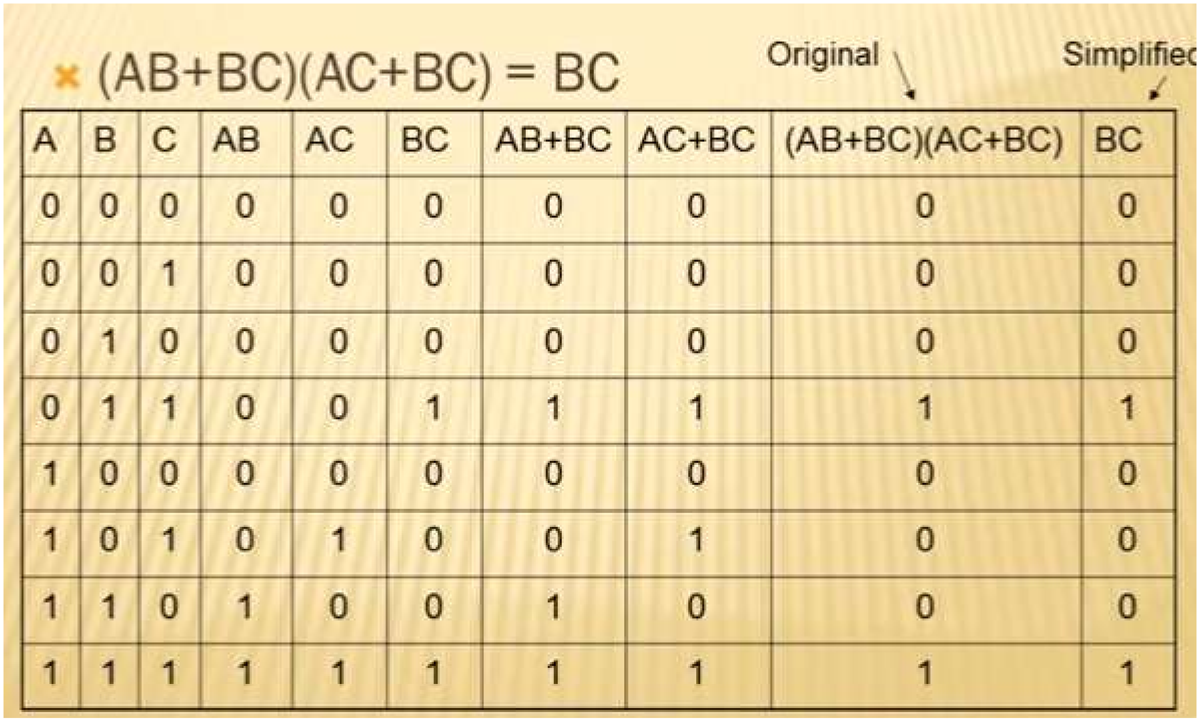
FOIL(FIRST - OUTER - INNER - LAST



- (AB+BC)(AC+BC)

- ABAC + ABBC + BCAC + BCBC
- ABC + ABC + ABC + BC
- BC (A + A + A + 1)
- BC (1)
- BC

FOIL(FIRST - OUTER - INNER - LAST


(AB+BC)(AC+BC) = BC

Original
Simplifiec

A	B	C	AB	AC	BC	AB+BC	AC+BC	(AB+BC)(AC+BC)	BC
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0
1	1	0	1	0	0	1	0	0	0
1	1	1	1	1	1	1	1	1	1