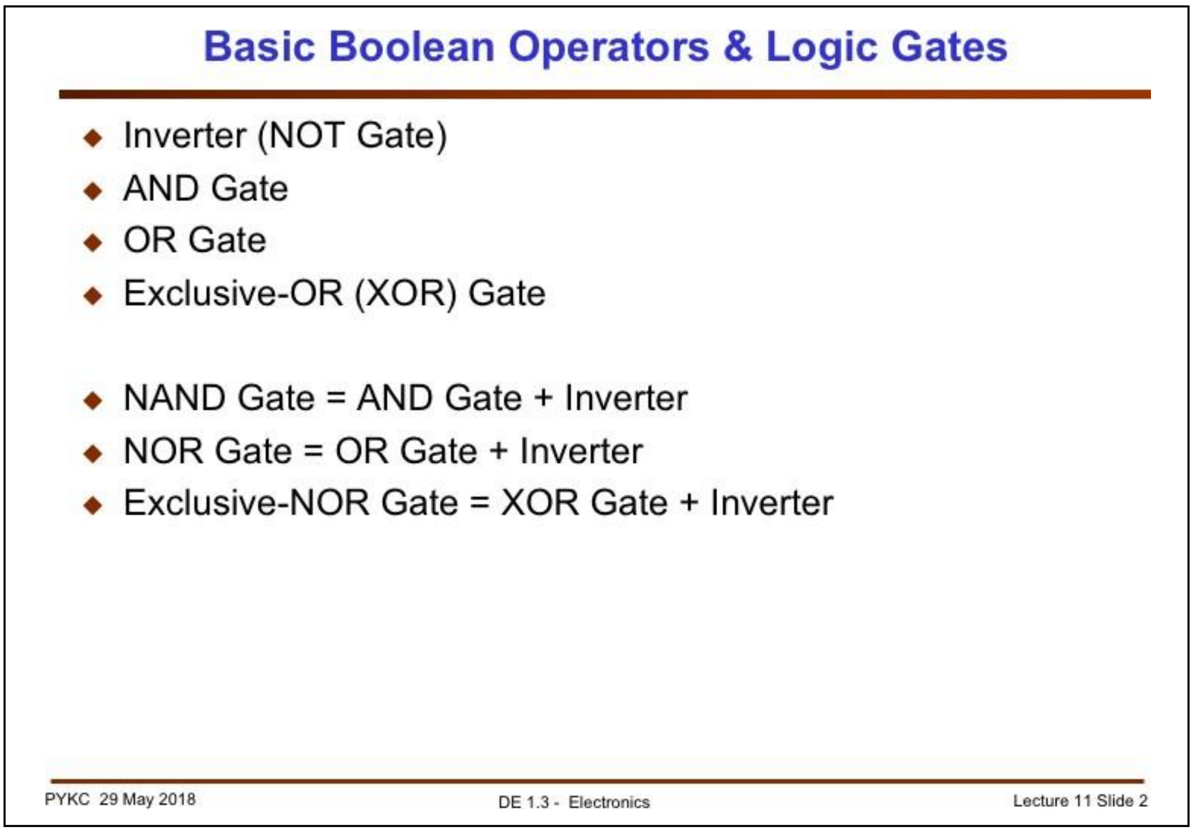
جامعة الفرات الاوسط التقنية

المعهد التقني /بابل

قسم تقنيات الاجهزة الطبية/المرحلة الاولى

Logic Gates



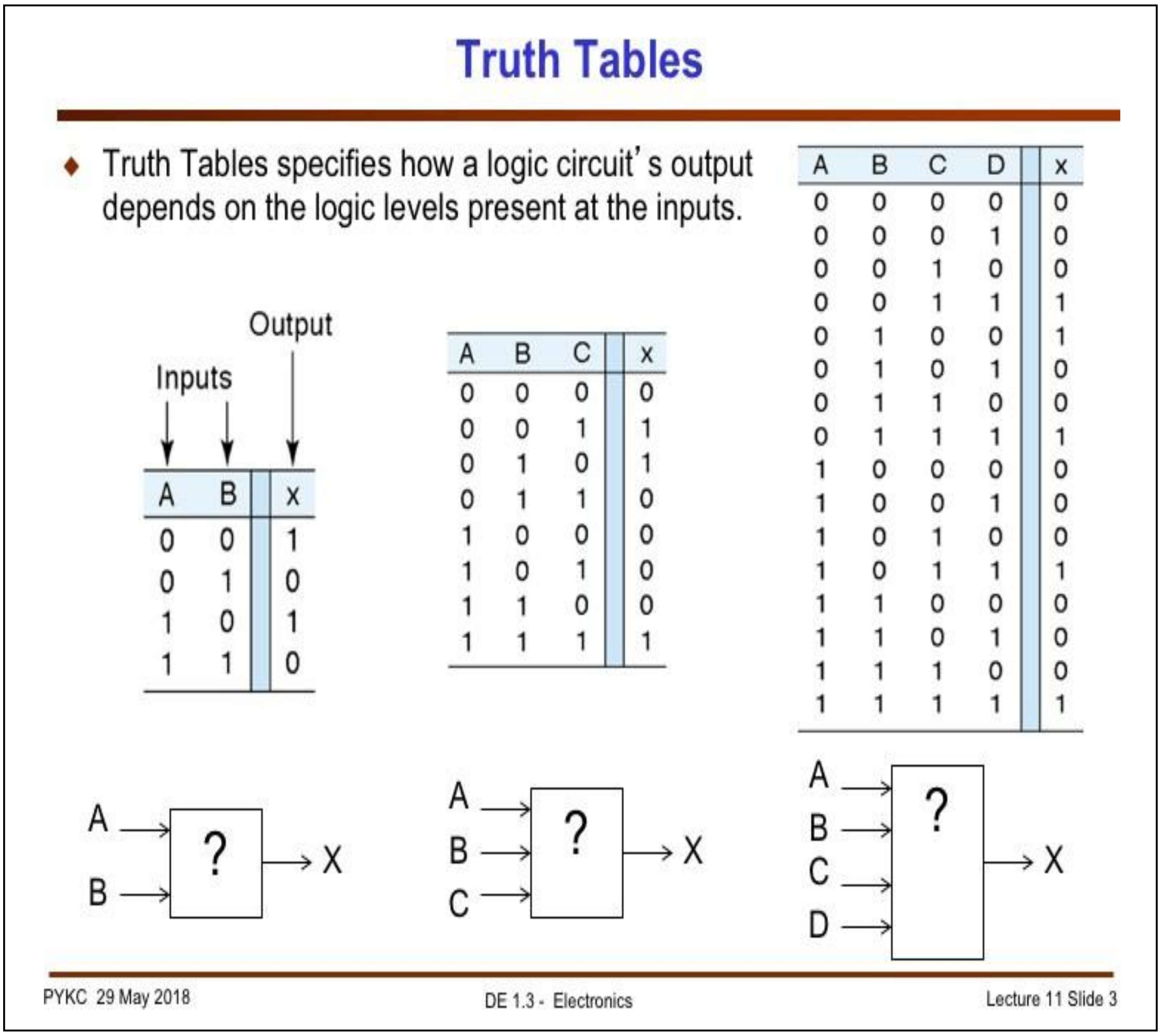
Basic Boolean Operators & Logic Gates

- Inverter (NOT Gate)
- AND Gate
- OR Gate
- Exclusive-OR (XOR) Gate

- NAND Gate = AND Gate + Inverter
- NOR Gate = OR Gate + Inverter
- Exclusive-NOR Gate = XOR Gate + Inverter

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 2

The basic logic gates are the inverter (or NOT gate), the AND gate, the OR gate and the exclusive-OR gate (XOR). If you put an inverter in front of the AND gate, you get the NAND gate etc.



Truth Tables

- Truth Tables specifies how a logic circuit's output depends on the logic levels present at the inputs.

Inputs

Output

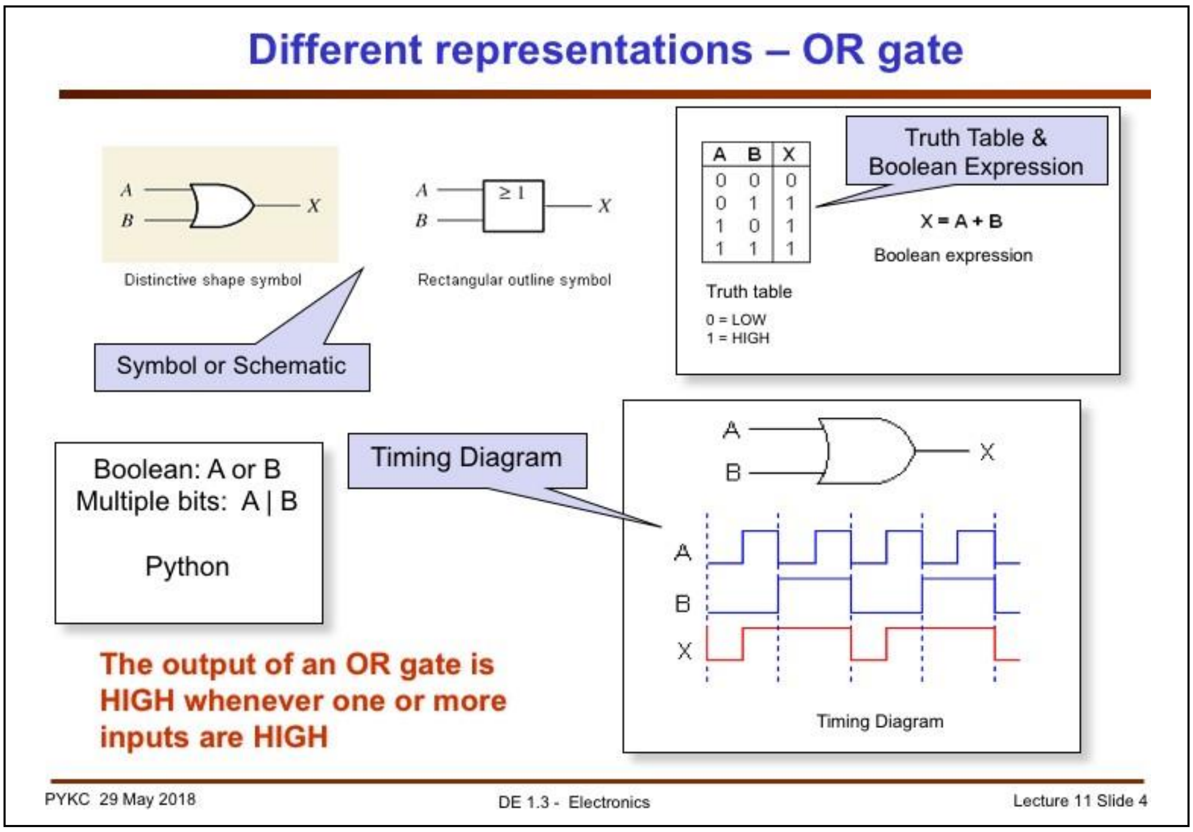
A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

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

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

A
B
?
X
A
B
C
?
X
A
B
C
D
?
X

One of the common tool in specifying a gate function is the truth table. All possible combination of the inputs A, B … etc, are enumerated, one row for each possible combination. Then a column is used to show the corresponding output value. If two logic circuits share identical truth table, they are functionally equivalent. Here are examples of truth tables for logic gate with 2, 3 and 4 inputs.



Different representations – OR gate

A
B
X
Distinctive shape symbol
Symbol or Schematic
A
B
≥ 1
X
Rectangular outline symbol
Truth Table & Boolean Expression
Truth table
0 = LOW
1 = HIGH
X = A + B
Boolean expression
Timing Diagram
Timing Diagram

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Boolean: A or B
Multiple bits: A | B

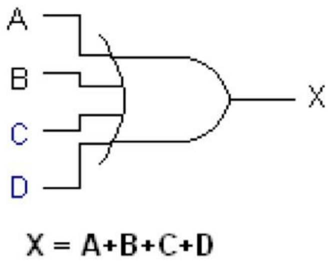
Python

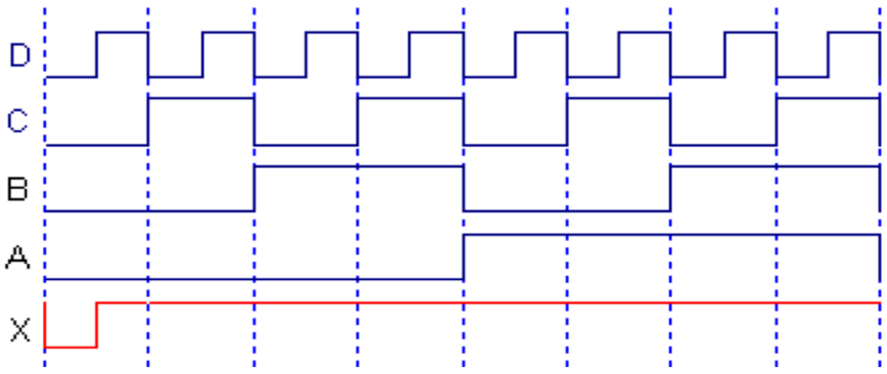
The output of an OR gate is HIGH whenever one or more inputs are HIGH

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 4

Here we show five different representations of the OR gate or OR) function. They are:

1. Schematic diagram in a logic symbol
2. Truth table
3. Boolean expression
4. Timing diagram
5. Expression in programming language (e.g. Python)



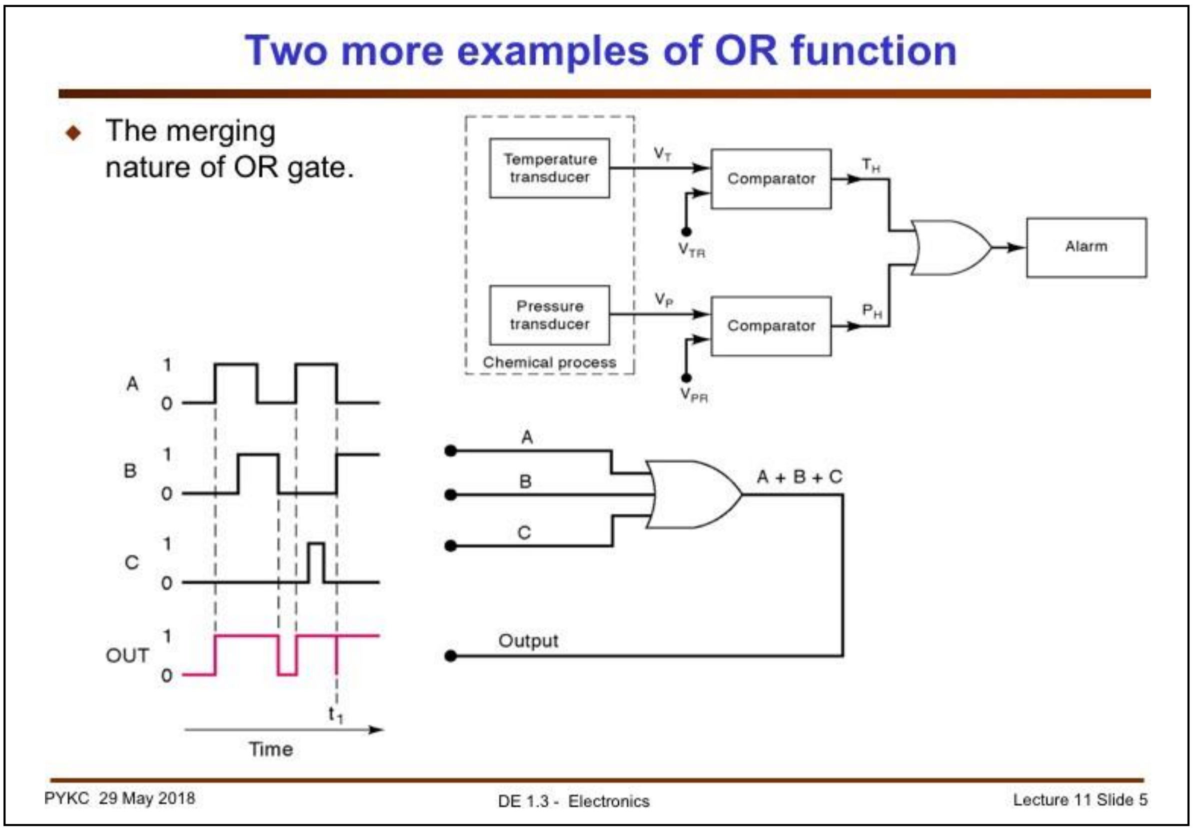


A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

In summary, OR operation produces as result of 1 whenever any input is 1. Otherwise 0.

An OR gate is a logic circuit that performs an OR operation on the circuit's input.

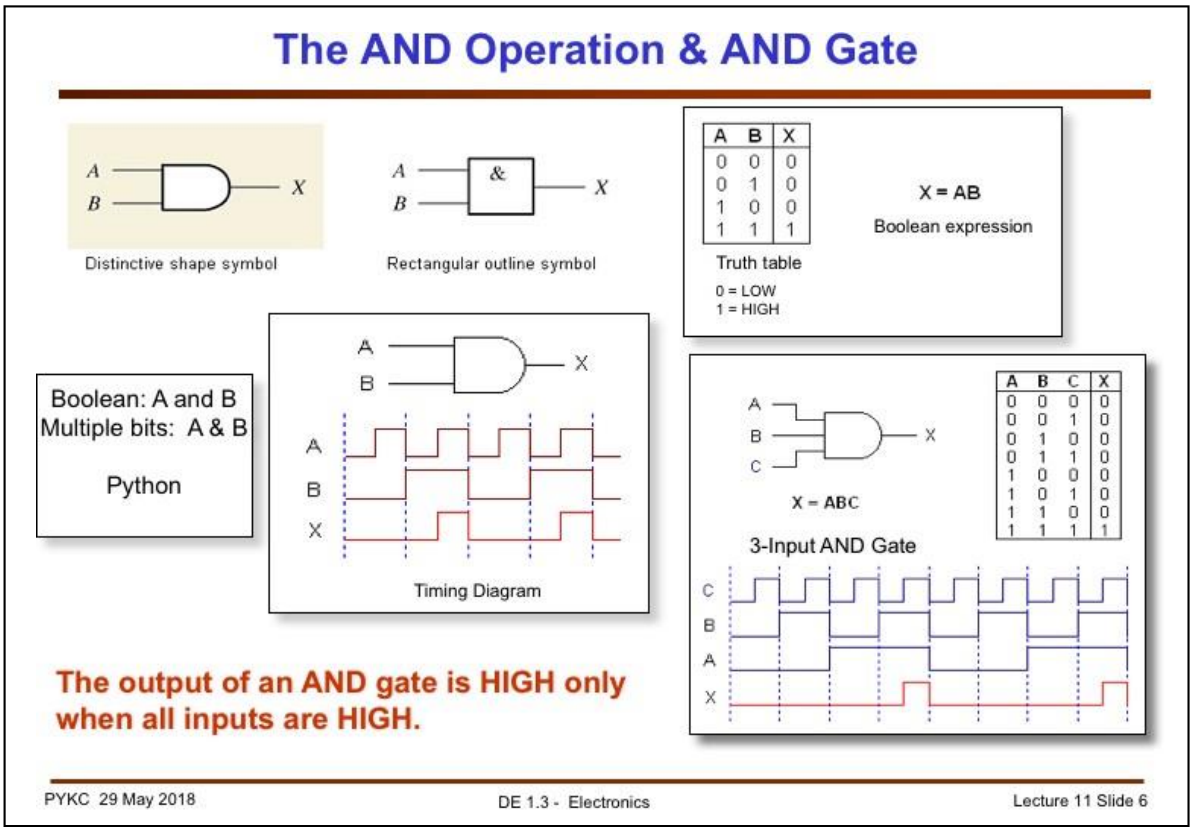
The expression x=A+B is read as "**x equals A OR B**"



Two more examples of OR function

- The merging nature of OR gate.

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 5



The AND Operation & AND Gate

A
B
X
Distinctive shape symbol
A
B
&
X
Rectangular outline symbol

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

0 = LOW
1 = HIGH

X = AB
Boolean expression

Boolean: A and B
Multiple bits: A & B

Python

A
B
X
A
B
X
Timing Diagram

A
B
C
X
X = ABC
3-Input AND Gate
C
B
A
X

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

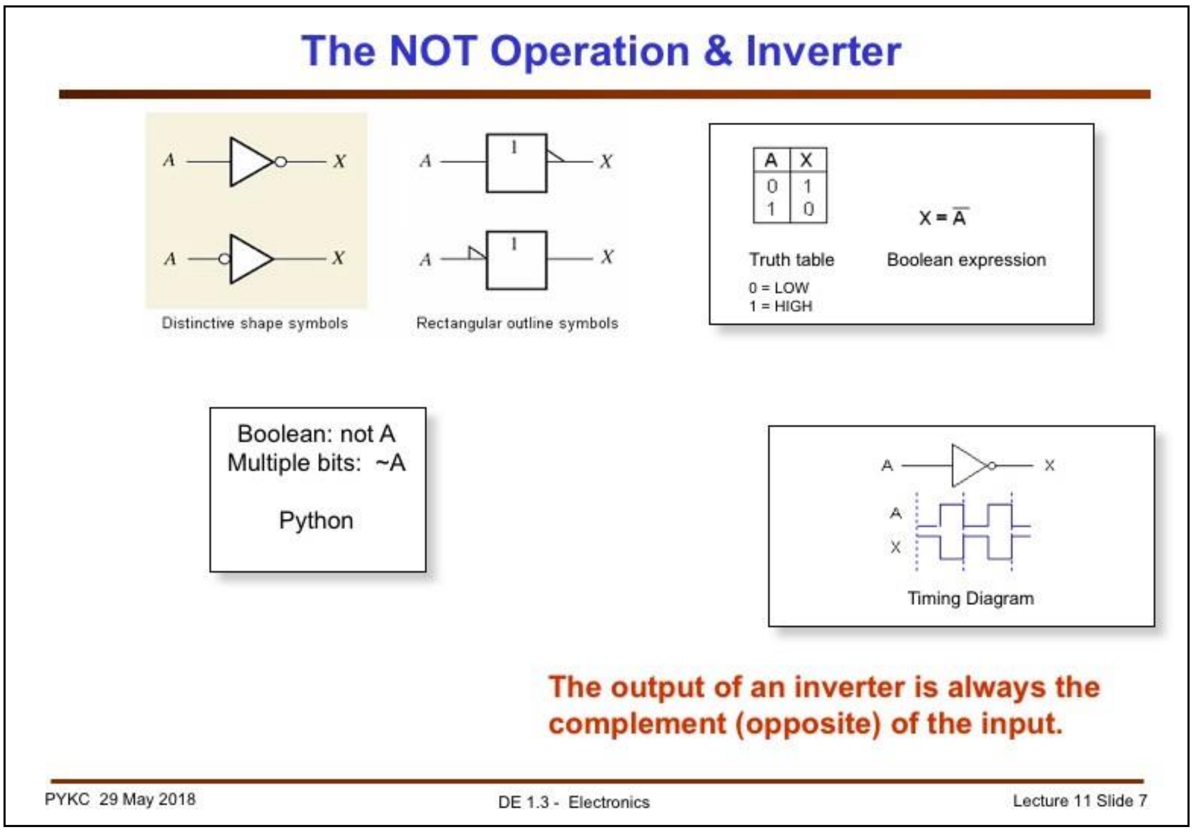
The output of an AND gate is HIGH only when all inputs are HIGH.

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 6

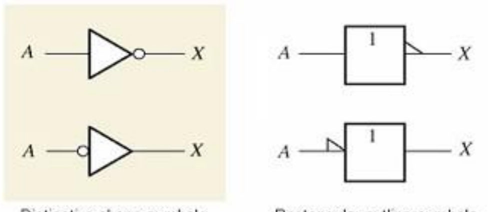
The **AND** operation is performed the same as ordinary multiplication of 1s and 0s. An **AND** gate is a logic circuit that performs the AND operation on the circuit's inputs.

An AND gate output will be 1 **only** when **all** inputs are 1; for all other cases the output will be 0.

The expression **x=A.B** is read as "x equals A AND B."



The NOT Operation & Inverter



A	X
0	1
1	0

Truth table

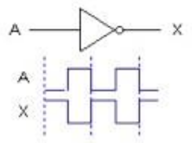
0 = LOW
1 = HIGH

$X = \overline{A}$

Boolean expression

Boolean: not A
Multiple bits: ~A

Python

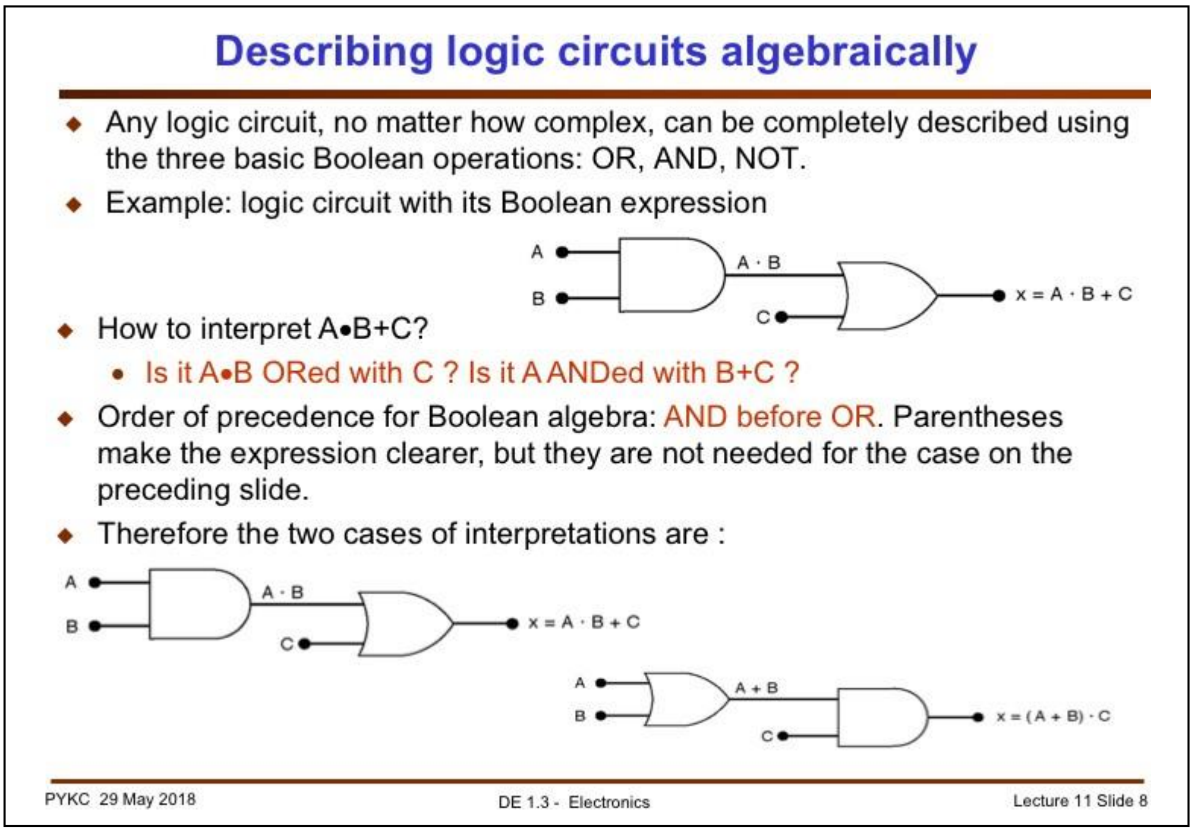


Timing Diagram

The output of an inverter is always the complement (opposite) of the input.

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 7

The NOT gate (inverter) is simple, but important. Note the difference between a Boolean operator "not A", where A is a Boolean variable (i.e. True or False), and that for a multiple bit variable. In multiple bit case, ~A results in EACH BIT within A being inverted. This is also known as "bitwise" operation.



Describing logic circuits algebraically

- Any logic circuit, no matter how complex, can be completely described using the three basic Boolean operations: OR, AND, NOT.
- Example: logic circuit with its Boolean expression
- How to interpret A•B+C?
  - Is it A•B ORed with C ? Is it A ANDed with B+C ?
- Order of precedence for Boolean algebra: AND before OR. Parentheses make the expression clearer, but they are not needed for the case on the preceding slide.
- Therefore the two cases of interpretations are :

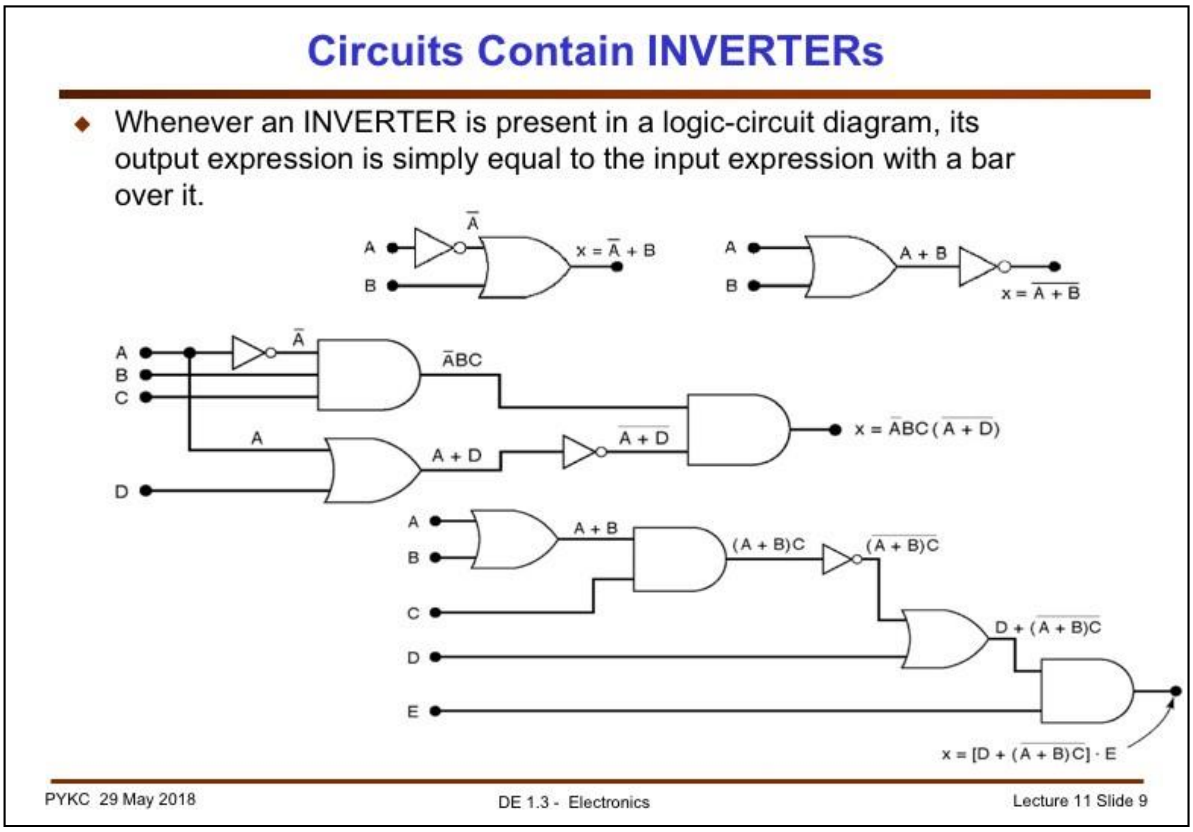
PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 8

Using symbolic diagram or truth table to specify or describe logic gates and logic functions is cumbersome. A much better way is to use algebraic expression. Here a “dot” represents the AND operation, and a “+” represents and OR operation. Furthermore, a bar over a variable or a ‘/’ in front of the variable represents an inversion (NOT function).

The convention is that AND has precedence over OR.

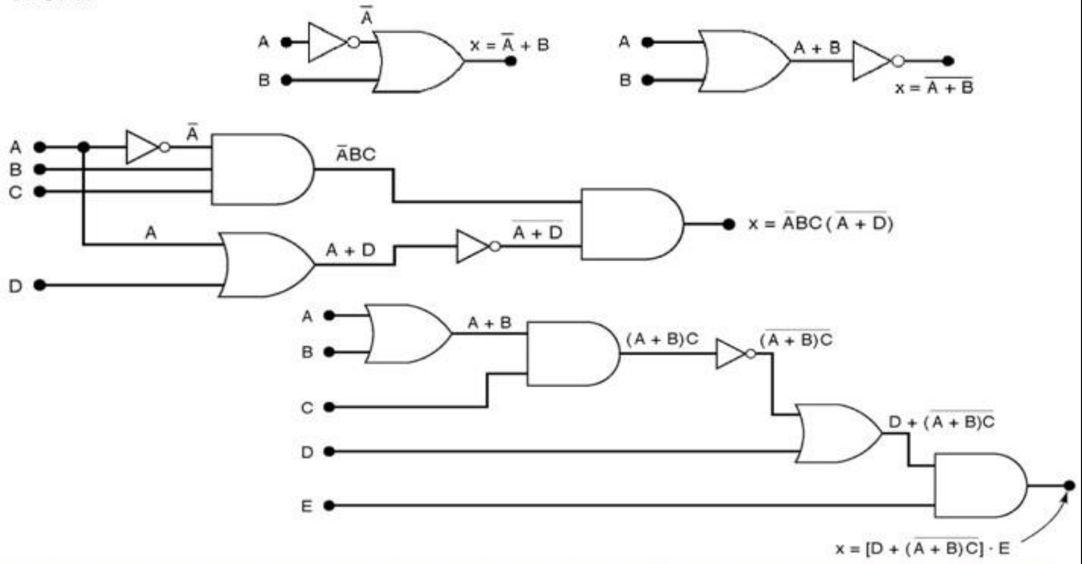
Precedence rules in Boolean algebra:

1. First, perform all inversions of single terms
2. Perform all operations with parentheses
3. Perform an AND operation before an OR operation unless parentheses indicate otherwise
4. If an expression has a bar over it, perform the operations inside the expression first and then invert the result



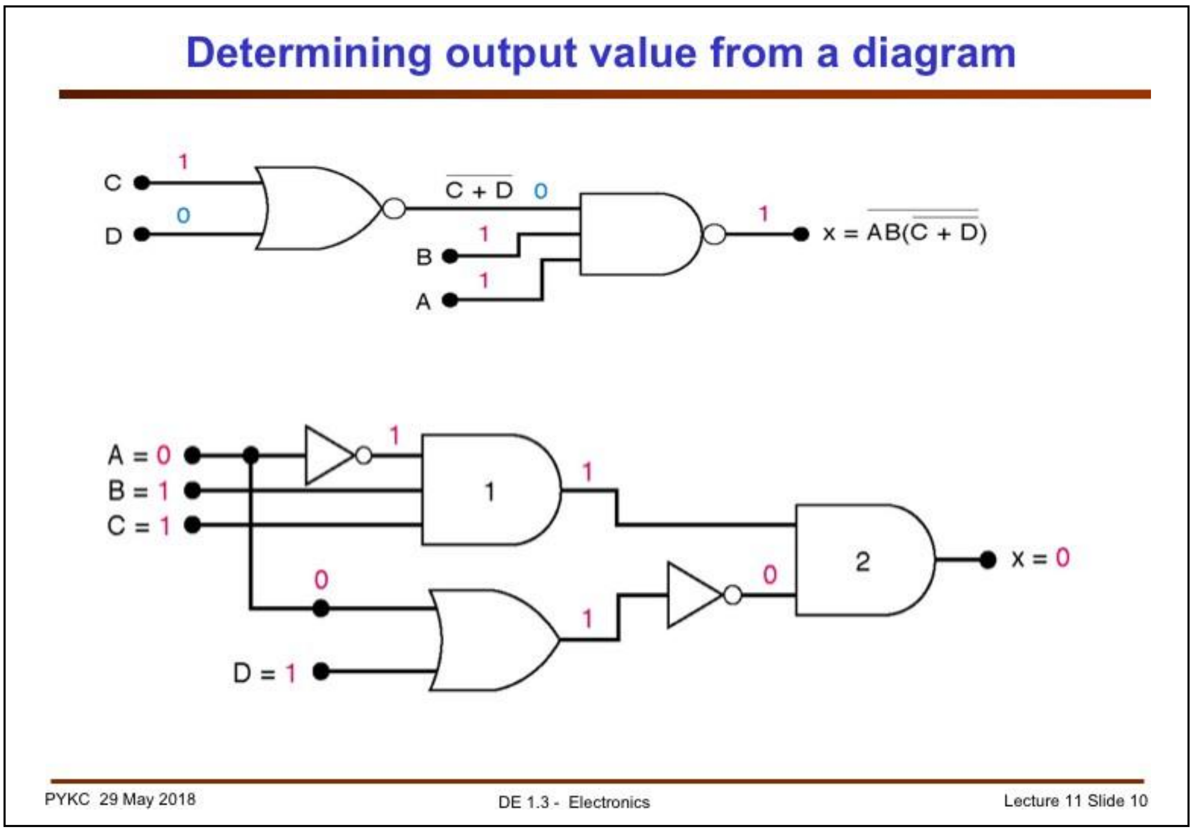
Circuits Contain INVERTERs

- Whenever an INVERTER is present in a logic-circuit diagram, its output expression is simply equal to the input expression with a bar over it.

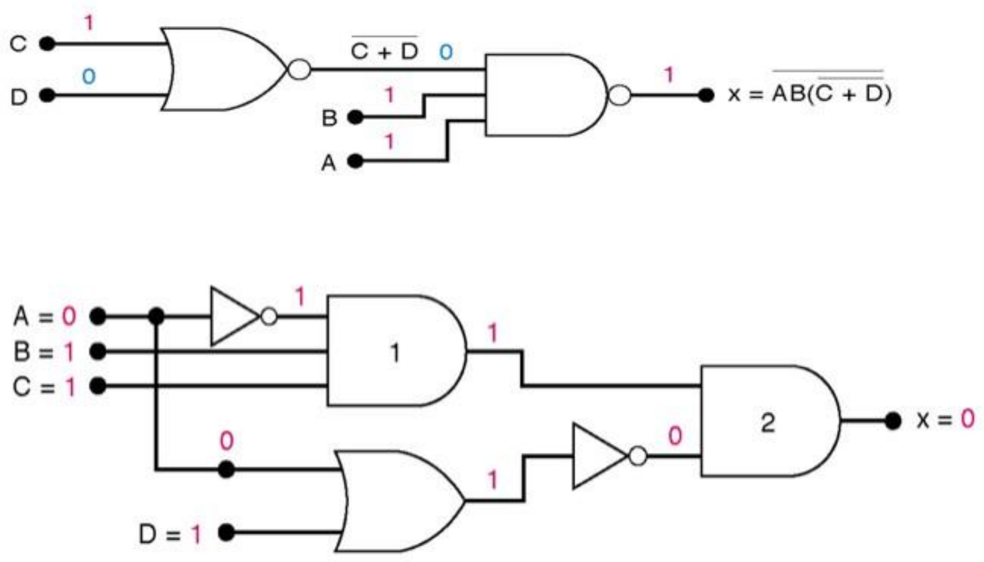


PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 9

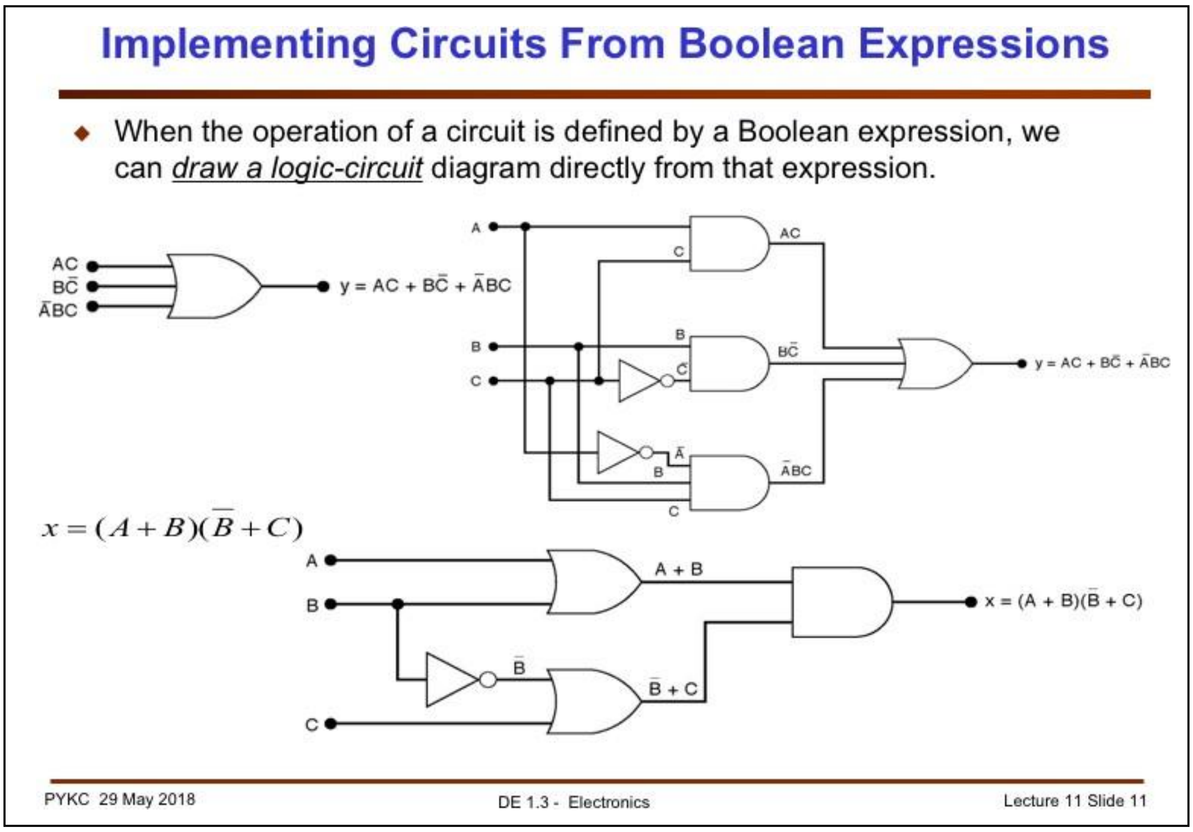
Inversion in Boolean expression has a bar over the Boolean variable. Here are a number of examples.



Determining output value from a diagram



We often do not draw the full inverter, but use a circle to indicate inversion. Therefore shown here on the top circuit, there is a 2- input OR gate followed by an inverter, making it a NOR gate. To evaluate the output of this circuit for inputs shown, we propagate the input values through the gates from left to right.



Implementing Circuits From Boolean Expressions

- When the operation of a circuit is defined by a Boolean expression, we can *draw a logic-circuit* diagram directly from that expression.

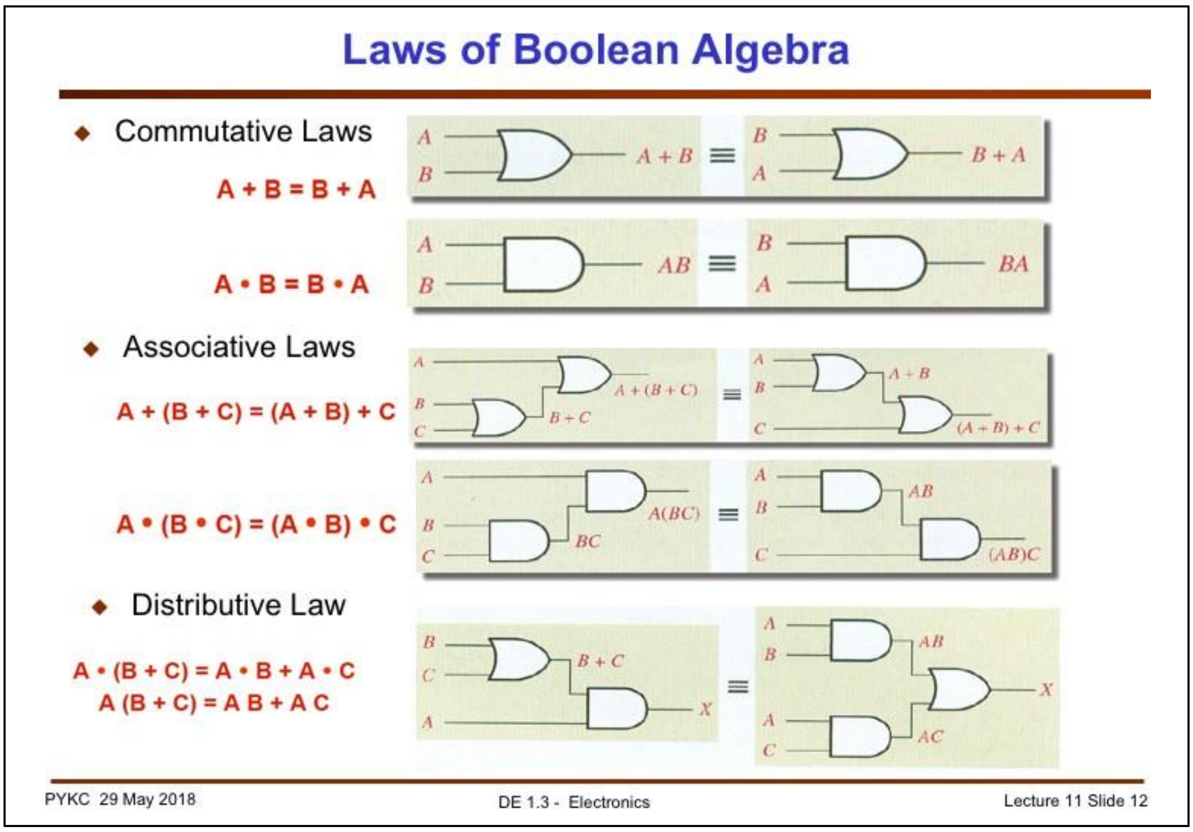
AC
$B\bar{C}$
$\bar{A}BC$
$y = AC + B\bar{C} + \bar{A}BC$
A
C
B
C
$\bar{C}$
$\bar{A}$
B
C
AC
$B\bar{C}$
$\bar{A}BC$
$y = AC + B\bar{C} + \bar{A}BC$

$x = (A + B)(\overline{B} + C)$

A
B
C
$\bar{B}$
A + B
$\bar{B}$ + C
$x = (A + B)(\bar{B} + C)$

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 11

Given a Boolean expression, we can easily translate it to symbolic representation of gates. This is quite easy to do.



Laws of Boolean Algebra

- Commutative Laws

  A + B = B + A

  A • B = B • A

- Associative Laws

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

  A • (B • C) = (A • B) • C

- Distributive Law

  A • (B + C) = A • B + A • C
  A (B + C) = A B + A C

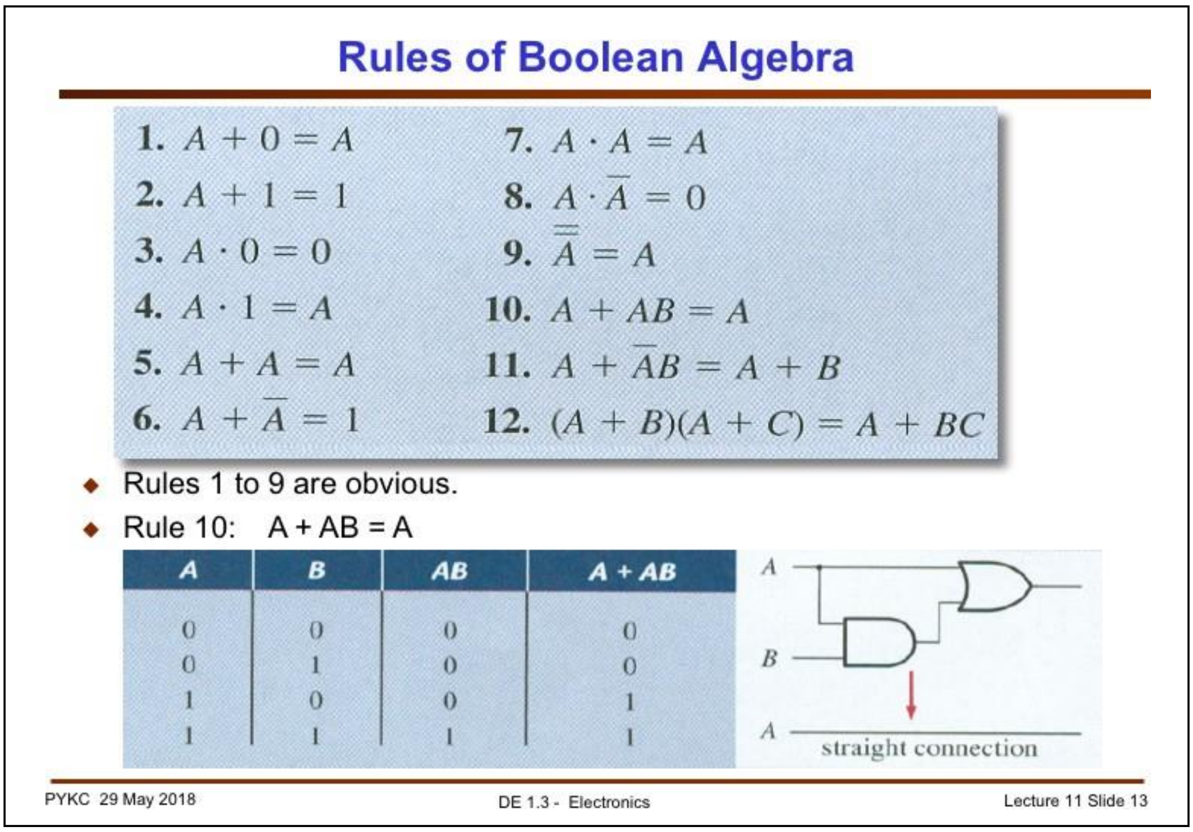
A
B
A + B
≡
B
A
B + A
AB
BA
A + (B + C)
B + C
A + B
(A + B) + C
A(BC)
BC
AB
(AB)C
X
AC

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 12

Just like normal algebra, Boolean algebra allows us to manipulate the logic equation and perform transformation and simplification.

Boolean algebra obeys the same laws as normal algebra:

1. the commutative law – the order of the Boolean **variables** do not matter
2. the associative law – the order of the Boolean **operators** do not matter
3. The distributive law –one can distribute a Boolean operator into the parenthesis



Rules of Boolean Algebra

1. $A + 0 = A$	7. $A \cdot A = A$
2. $A + 1 = 1$	8. $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	9. $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	10. $A + AB = A$
5. $A + A = A$	11. $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	12. $(A + B)(A + C) = A + BC$

- Rules 1 to 9 are obvious.
- Rule 10: A + AB = A

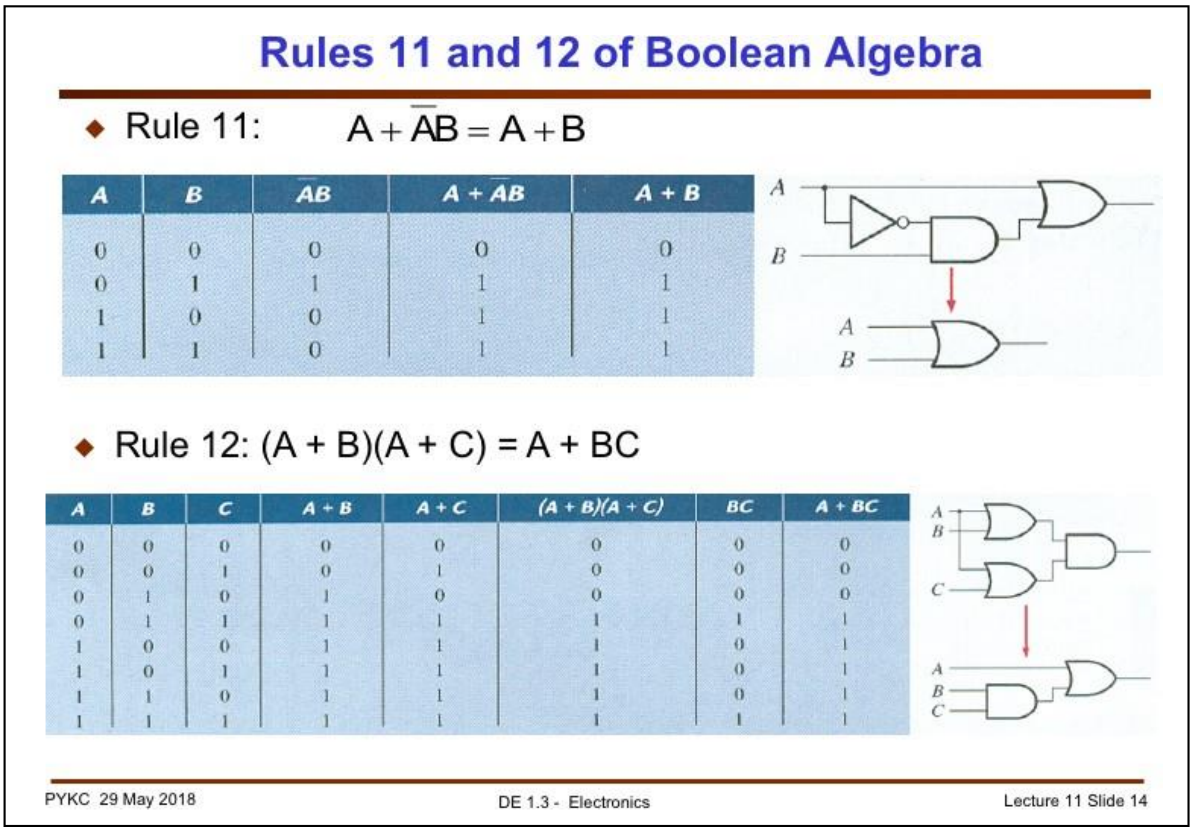
A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

A
B
A
straight connection

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 13

There are also a number of rules to help simplification of Boolean expression. The first 9 rules listed here are obvious.

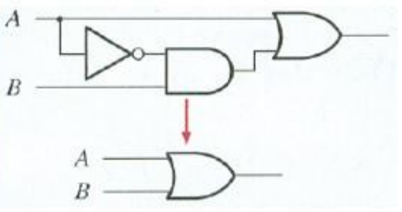
Rule 10: Less obvious, but it is clearly shown here that it is true.



Rules 11 and 12 of Boolean Algebra

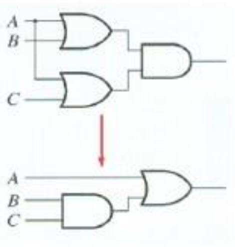
- Rule 11: $A + \overline{A}B = A + B$

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



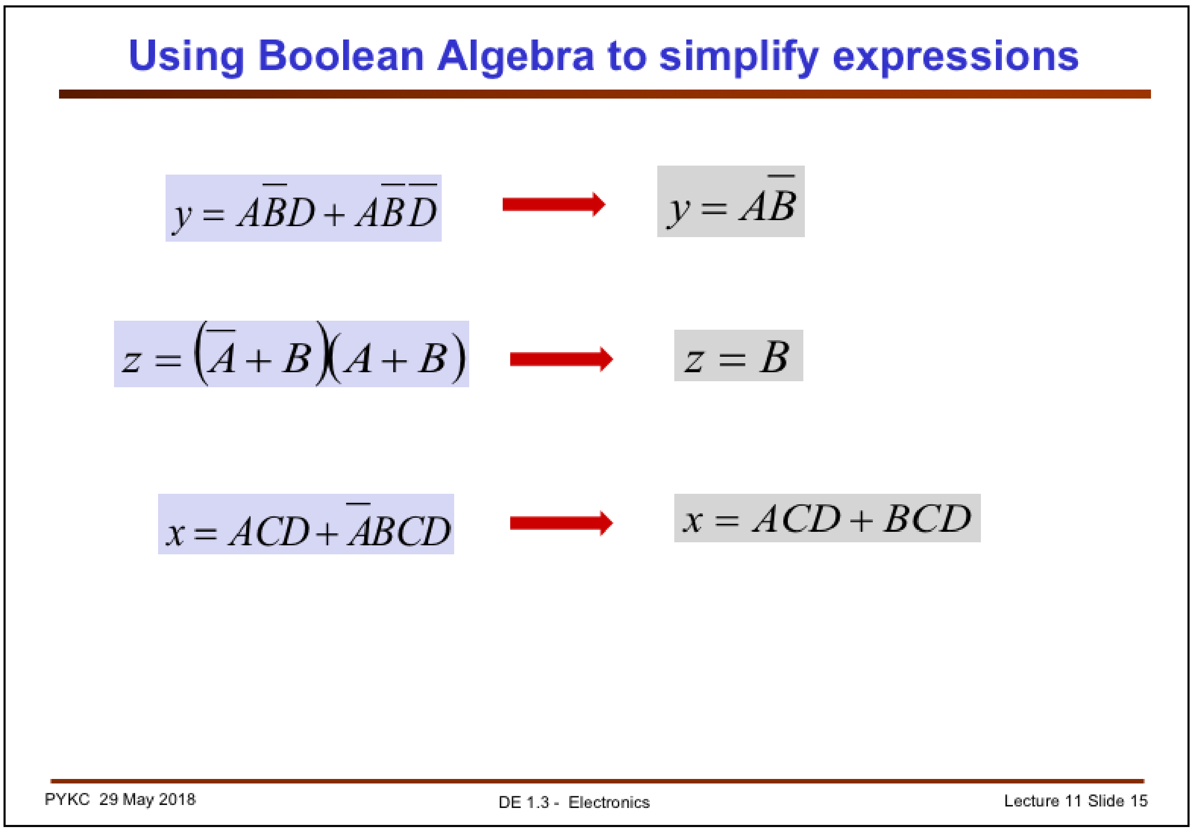
- Rule 12: $(A + B)(A + C) = A + BC$

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1



PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 14

Rule 11 and Rule 12 are more difficult. You may need to remember them in order to apply them for the purpose of simplifying Boolean expressions.



Using Boolean Algebra to simplify expressions

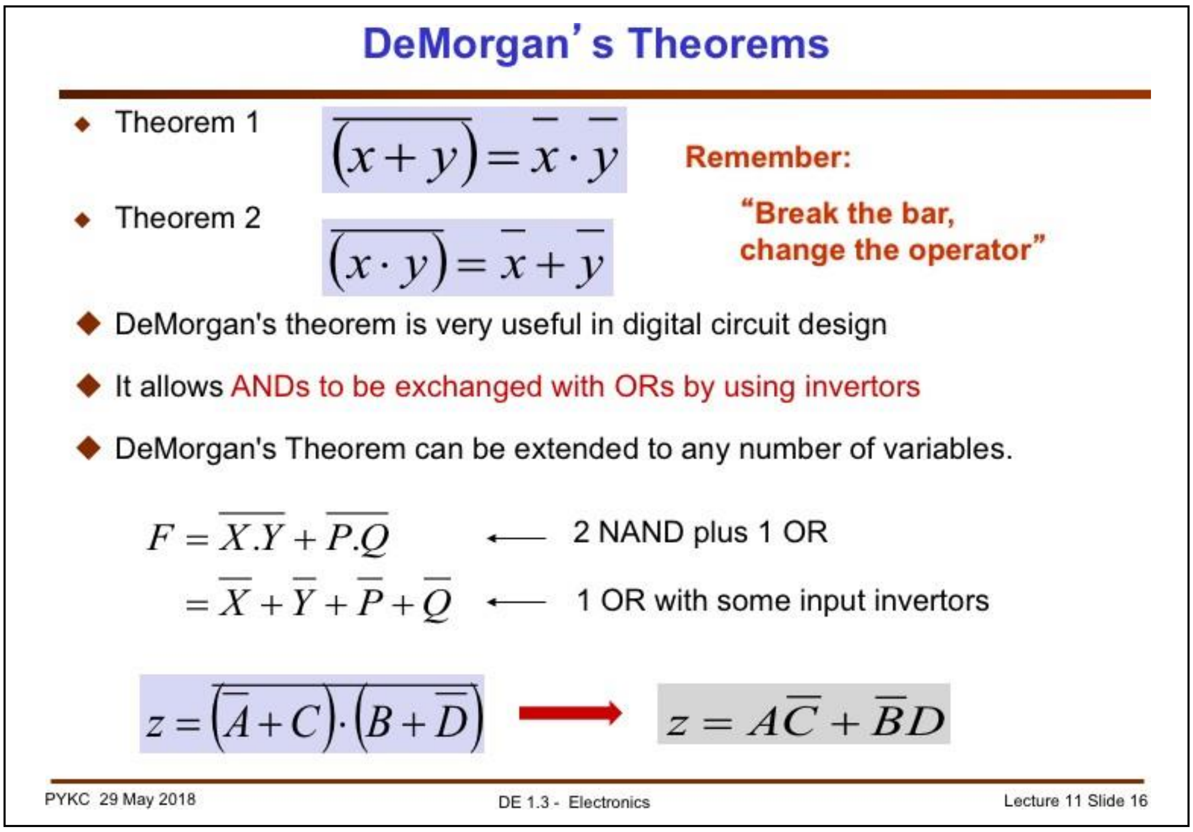
$y = A\overline{B}D + A\overline{B}\,\overline{D} \rightarrow y = A\overline{B}$

$z = (\overline{A} + B)(A + B) \rightarrow z = B$

$x = ACD + \overline{A}BCD \rightarrow x = ACD + BCD$

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 15

Above are three examples of simplification of Boolean expressions.



DeMorgan's Theorems

- Theorem 1 $\overline{(x+y)}=\overline{x}\cdot\overline{y}$
- Theorem 2 $\overline{(x\cdot y)}=\overline{x}+\overline{y}$

Remember:

"Break the bar, change the operator"

- DeMorgan's theorem is very useful in digital circuit design
- It allows ANDs to be exchanged with ORs by using invertors
- DeMorgan's Theorem can be extended to any number of variables.

$$F=\overline{X.Y}+\overline{P.Q} \longleftarrow \text{2 NAND plus 1 OR}$$

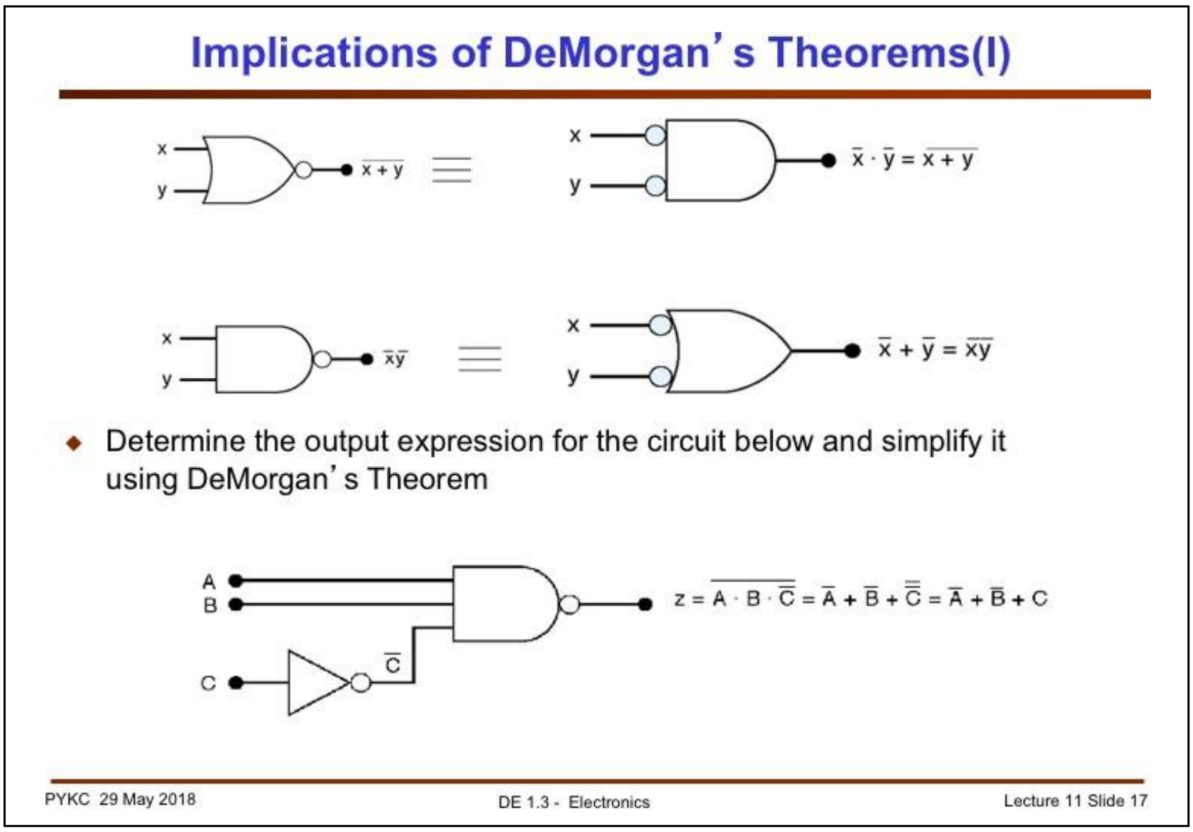
$$=\overline{X}+\overline{Y}+\overline{P}+\overline{Q} \longleftarrow \text{1 OR with some input invertors}$$

$$z=\overline{(\overline{A}+C)\cdot(B+\overline{D})} \longrightarrow z=A\overline{C}+\overline{B}D$$

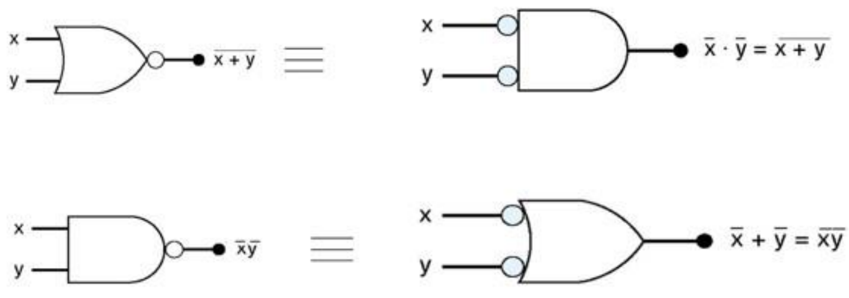
PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 16

De'Morgan's Theorems is important to Boolean logic. They allow us to exchange OR operation with AND operation and vice versa.

Applying De'Morgan, we can also simplify Boolean expression in many cases.

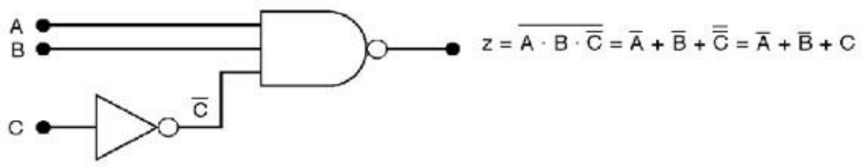


Implications of DeMorgan's Theorems(I)



x
y
$\overline{x+y}$
$\equiv$
x
y
$\bar{x}\cdot\bar{y}=\overline{x+y}$
x
y
$\overline{xy}$
$\equiv$
x
y
$\bar{x}+\bar{y}=\overline{xy}$

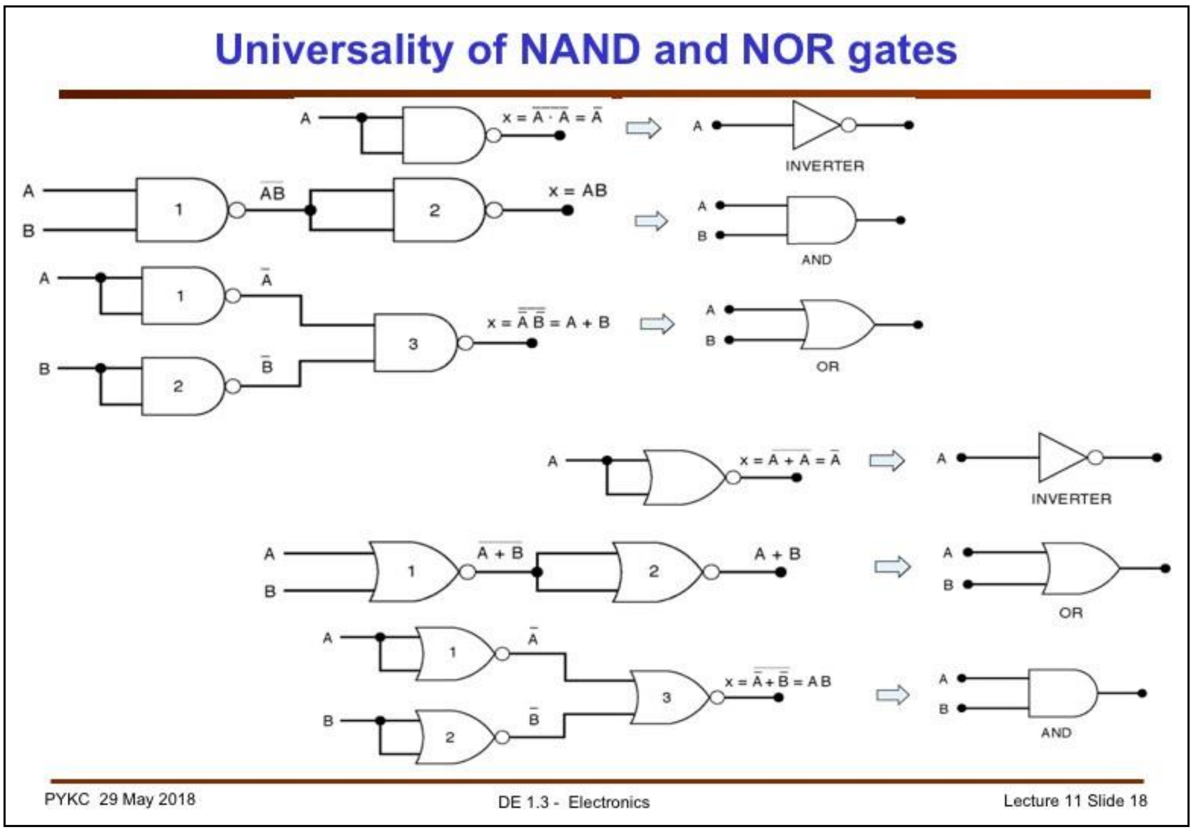
- Determine the output expression for the circuit below and simplify it using DeMorgan's Theorem



A
B
C
$\bar{C}$
$z=\overline{A\cdot B\cdot\bar{C}}=\bar{A}+\bar{B}+\bar{\bar{C}}=\bar{A}+\bar{B}+C$

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 17

De'Morgan requires an inverter output. Here is symbolic representation of De'Morgan: move the inversion to the inputs, and change OR to AND, or AND to OR.

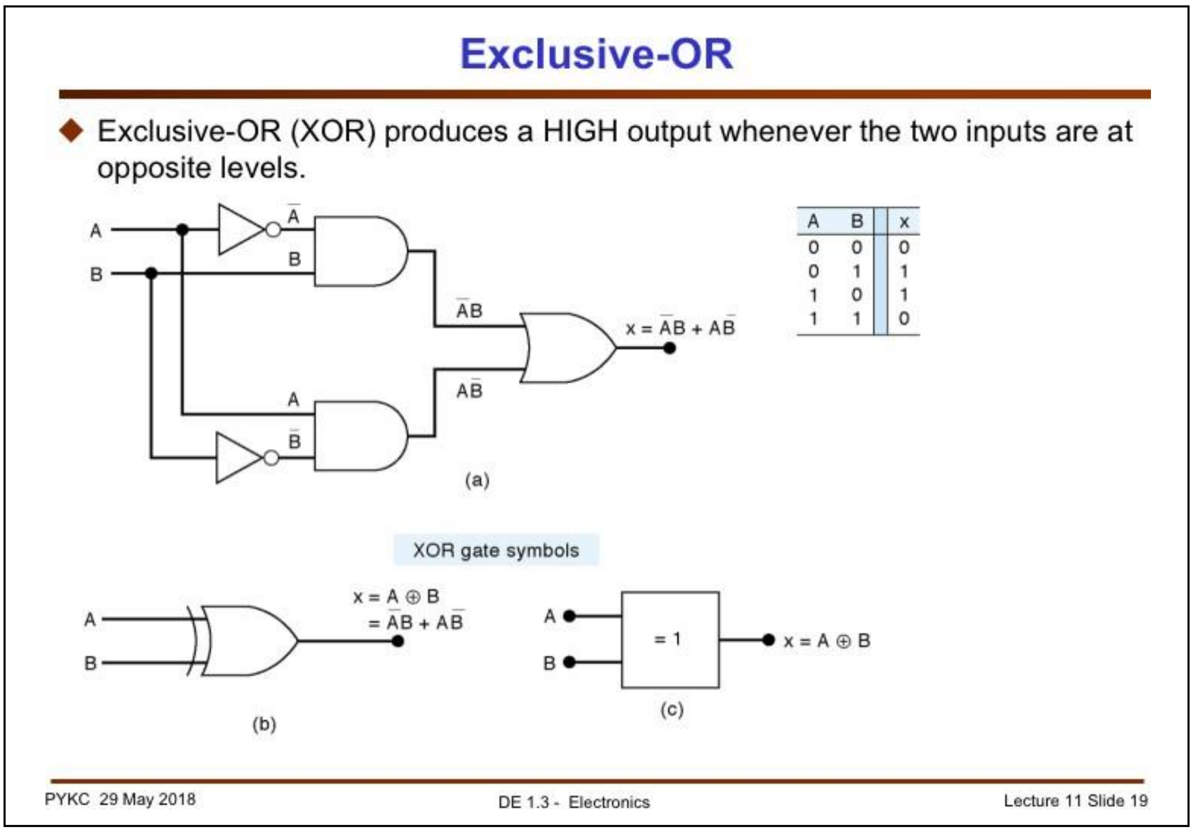


Universality of NAND and NOR gates

A
$x = \overline{A \cdot A} = \overline{A}$
INVERTER
A
B
1
$\overline{AB}$
2
$x = AB$
AND
$\overline{A}$
$\overline{B}$
3
$x = \overline{\overline{A}\,\overline{B}} = A + B$
OR
$x = \overline{A + A} = \overline{A}$
INVERTER
$\overline{A + B}$
$A + B$
OR
$x = \overline{\overline{A} + \overline{B}} = AB$
AND

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 18

Let us assume that we ONLY have 2-input NAND gate. From this, we can get an inverter, an AND gate, and, thanks to De'Morgan, we can also get an OR gate. In other words, if we have a 2-input NAND gate, we can build the three basic logic operators: NOT, AND and OR. As a result, we can build ANY logic circuit and implement any Boolean expression. Taken to limit, give me as many NAND gate as I want, in theory I can build a Pentium processor. This shows the universality of the NAND gate. Similarly, one can do the same for NOR gates.



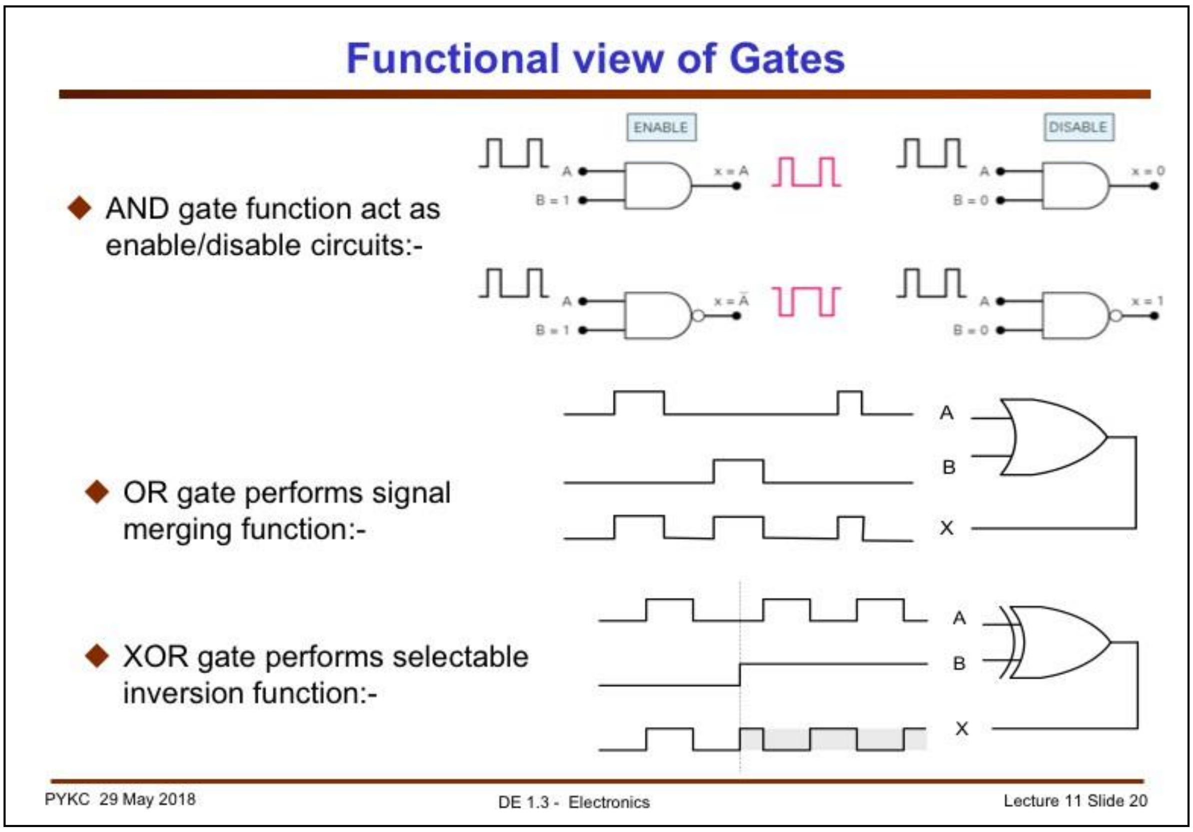
Exclusive-OR

- Exclusive-OR (XOR) produces a HIGH output whenever the two inputs are at opposite levels.

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

A
B
$\bar{A}$
B
$\bar{A}B$
$x = \bar{A}B + A\bar{B}$
$A\bar{B}$
A
$\bar{B}$
(a)
XOR gate symbols
A
B
$x = A \oplus B$
$= \bar{A}B + A\bar{B}$
(b)
A
B
= 1
$x = A \oplus B$
(c)

The exclusive-OR gate is high only if the two inputs are DIFFERENT, i.e. either A is high OR B is high but not both. The XOR gate is often used as logic comparator (output is 0 if the two inputs are equal).


ENABLE
DISABLE
A
B = 1
x = A
B = 0
x = 0
x = Ā
x = 1
A
B
X

Functional view of Gates

- AND gate function act as enable/disable circuits:-
- OR gate performs signal merging function:-
- XOR gate performs selectable inversion function:-

PYKC 29 May 2018 DE 1.3 - Electronics Lecture 11 Slide 20

Now let us take a functional view of logic operators.

The AND operator can be interpreted as having an enabling or disabling function. (We sometimes call this a ‘gating function” as if it perform a gate keeping (i.e blocking) function.) Input B here is a the gating control – if B = 1, it lets A through, otherwise if B = 0, it blocks A.

The OR operator can be interpreted as a merging function. It combines both A and B high level and merge them to form output X.

The XOR gate can be viewed as a selectable inverter. If B = 0, A is pass to the output. If B = 1, A is inverted. So B determines inversion, or no inversion of A.