LOGIC GATES

- **NOT Gate**
- **AND Gate**
- **OR Gate**
- **NAND Gate**
- **NOR Gate**
- **EXOR Gate**
- **EXNOR Gate**
- **Implementing logic circuits from logical expressions**

LOGIC GATES

Gates are the basic components of logic circuits. Gates are electronic circuits that perform Boolean operations. Input(s) of logic gates are variable values (0/1 or 0V/+5V) and the output is the result of the Boolean operation.

Gates are built with transistors, diodes, resistors, and capacitors. They are sold in the form of integrated circuits.

There are 7 basic gates:

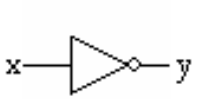
AND, OR, NOT,

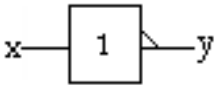
NAND, NOR,

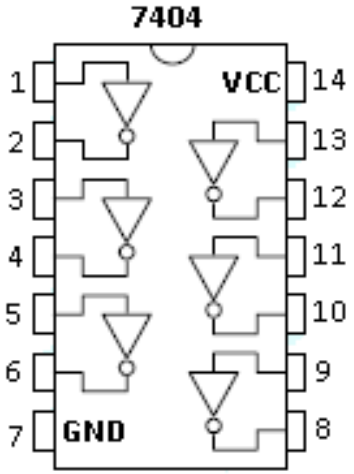
EXOR, and EXNOR gates.

NOT Gate

NOT gate complements the input signal. It outputs 1 when the input is 0, and outputs 0 when the input is 1. It is represented with the following symbols.



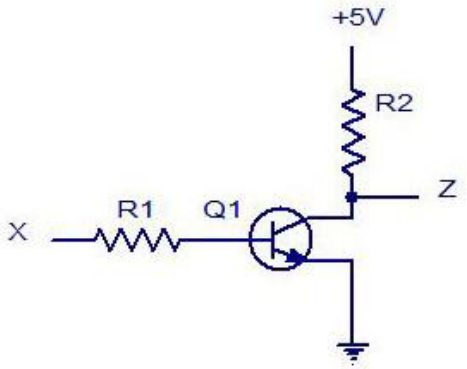


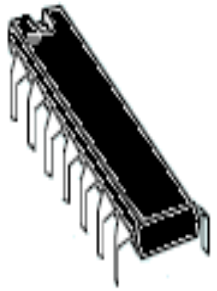


Truth Table

x	y
0	1
1	0

Electronic Circuit



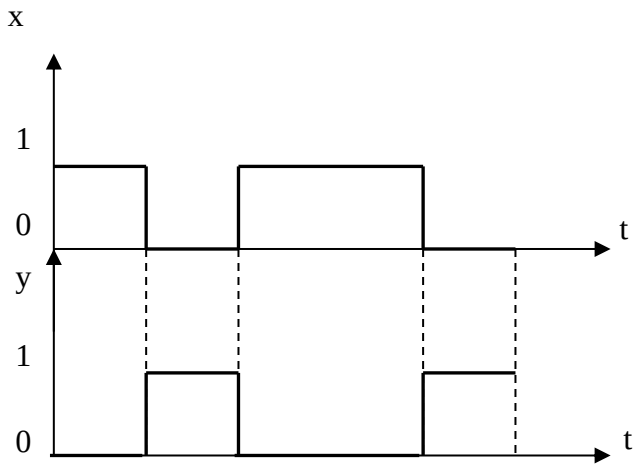


Logical Expression

$y = x'$ $y = \bar{x}$

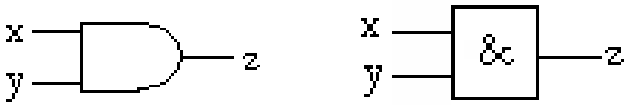
NOT Gate

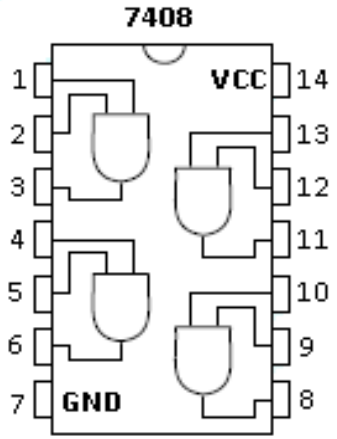
Example: If we input the signal x to a NOT gate, the output signal (y) would be as shown in the timing diagram below.



AND Gate

AND gates performs the Boolean AND operation. It has two inputs and a single output. It is represented with the following symbols.





Truth Table

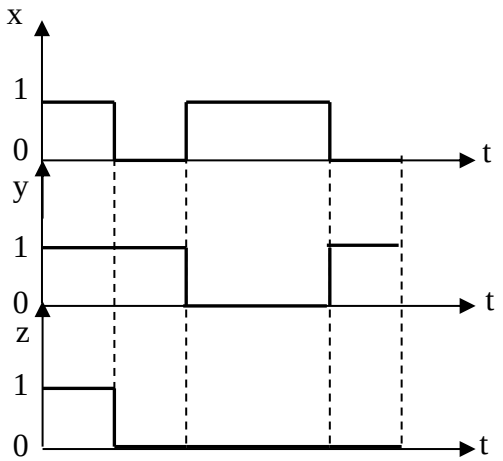
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Logical Expression:

$z = x.y \qquad z = xy$

AND Gate

Example: If we input the signals x and y to an AND gate, the output signal (z) would be as shown in the timing diagram below.

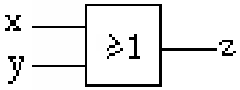


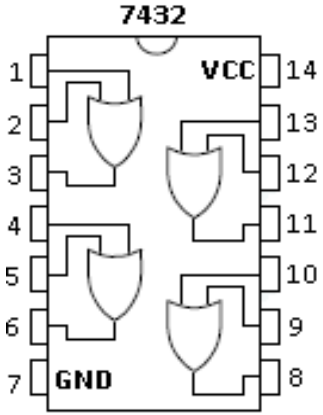
Remember that, AND operation is the equivalent of binary multiplication.

$0 \cdot 0 = 0,\ 0 \cdot 1 = 0,\ 1 \cdot 0 = 0,\ 1 \cdot 1 = 1$

OR Gate

OR gates performs the Boolean OR operation. It has two inputs and a single output. It is represented with the following symbols.





Truth Table

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

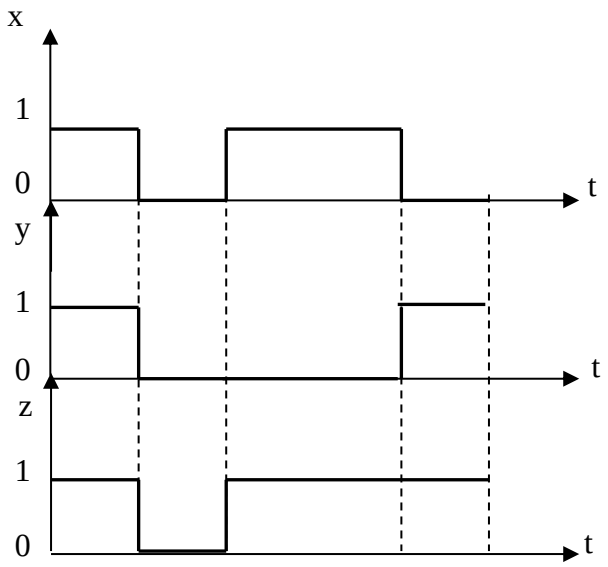
Logical Expression

$$z = x + y$$

It is similar to binary addition, but the output is 1 when both inputs are 1.

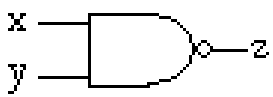
OR Gate

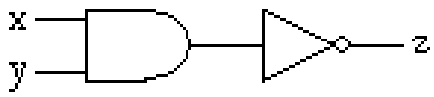
Example: If we input the signals x and y to an OR gate, the output signal (z) would be as shown in the timing diagram below.

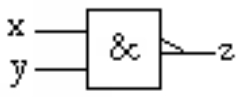


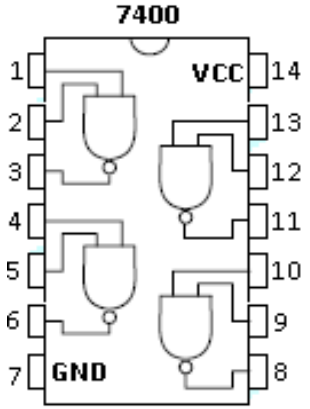
NAND Gate

NAND gates are universal gates. We can implement AND, OR, and NOT gates with NAND gates. We can consider NAND gates as AND gates which have a NOT gate right after the output. It is represented with the following symbols.









Truth Table

x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

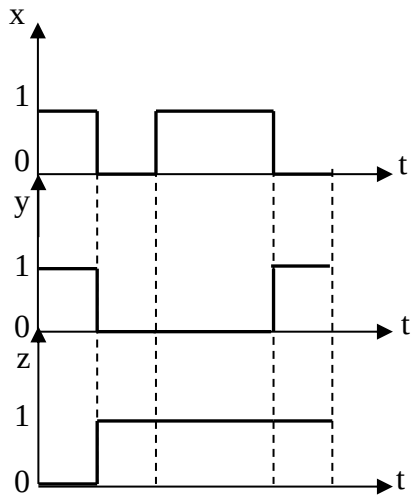
Logical Expression

$z = (x.y)'$ $\qquad z = \overline{x.y}$

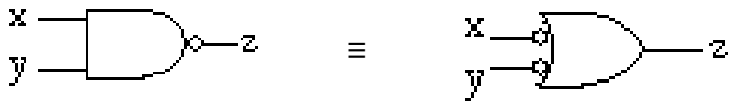
NAND gate outputs 0 if all the inputs are 1. All the other combinations make the gate output 0.

NAND Gate

If we input the signals x and y to an NAND gate, the output signal (z) would be as shown in the timing diagram below.

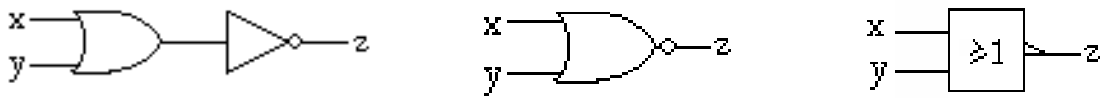


There's a similarity between OR gate and NAND gate. OR gate outputs 1 if at least one input is 1, and NAND gate outputs 1 if at least one input is 0.



NOR Gate

NOR gates are universal gates just like NAND gates. We can consider NOR gates as OR gates which have a NOT gate right after the output. It is represented with the following symbols.



Truth Table

x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

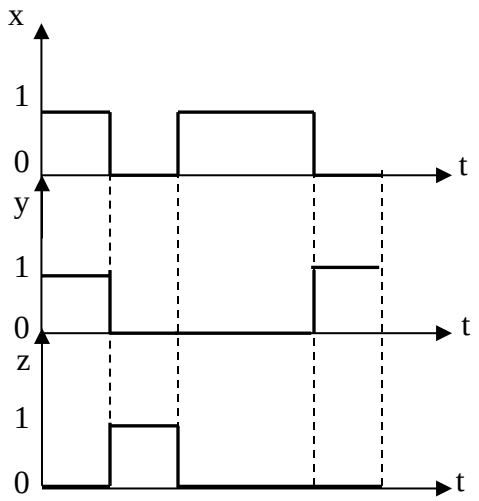
Logical Expression

$z = (x + y)'$ $\quad z = \overline{x + y}$

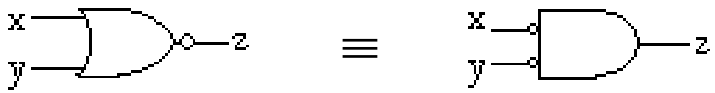
NOR gate outputs 1 if all the inputs are 0. All the other combinations make the gate output 0.

NOR Gate

If we input the signals x and y to an NOR gate, the output signal (z) would be as shown in the timing diagram below.

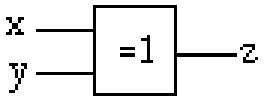


There's a similarity between AND gate and NOR gate. AND gate outputs 1 if all the inputs are 1, and NOR gate outputs 1 if all the inputs are 0.



EXOR Gate

EXOR gate can be implemented with AND and OR gates. But since we need this operation frequently, there's a basic logic gate for it. It checks if the inputs are different or not. It is represented with the following symbols.



Truth Table

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Logic Expression

$z = x \oplus y$ $\quad z = x'y + xy'$

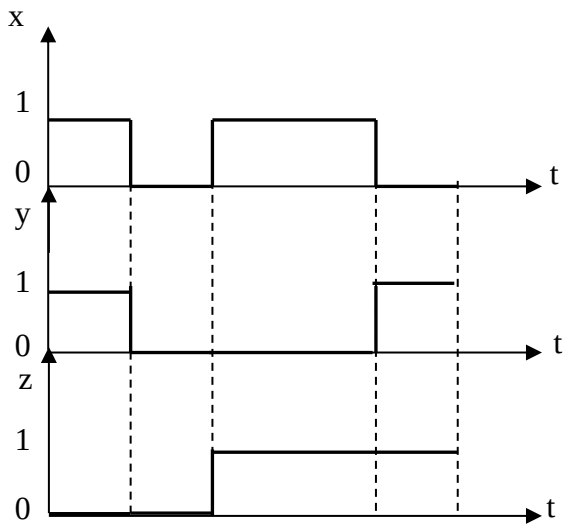
❖ If an EXOR gate has more than two inputs, when we input an odd number of 1's, the output will be 1.

E.g. $1 \oplus 0 \oplus 1 \oplus 1 = 1$, $\quad 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$,

❖ We can apply associative law. $a \oplus (b \oplus c) = (a \oplus b) \oplus c = a \oplus b \oplus c$

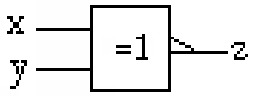
EXOR Gate

If we input the signals x and y to an EXOR gate, the output signal (z) would be as shown in the timing diagram below.



EXNOR Gate

EXNOR gate can be implemented with AND and OR gates just like EXOR gate. It checks if inputs are the same or not. It is represented with the following symbols.



Truth Table

x	y	z
0	0	1
0	1	0
1	0	0
1	1	1

Logic Expression

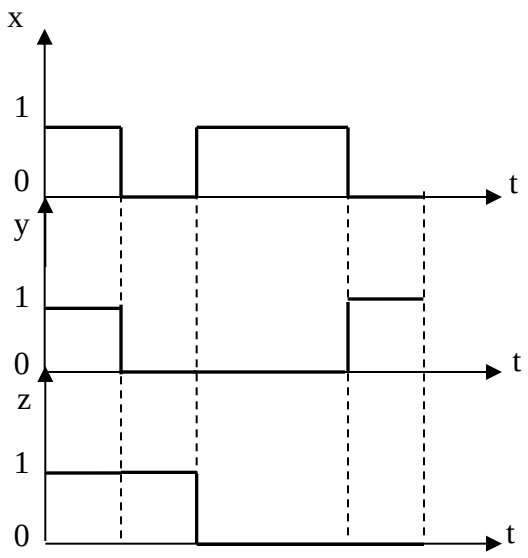
$z = x \otimes y$ $\qquad z = x.y + x'.y'$

- EXNOR gate is the complement of EXOR gate. $z = (x \oplus y)'$
- If an EXNOR gate has more than two inputs, when we input an even number of 0's, the output will be 1.

  E.g. $1 \otimes 0 \otimes 1 \otimes 1 = 0$, $1 \otimes 0 \otimes 0 \otimes 1 \otimes 1 = 1$,
- We can apply associative law. $a \otimes (b \otimes c) = (a \otimes b) \otimes c = a \otimes b \otimes c$

EXNOR Gate

If we input the signals x and y to an EXNOR gate, the output signal (z) would be as shown in the timing diagram below.



Implementing Logic Circuits From Logical Expressions

First, we check if we need complements of any variables. If we need the complement of a variable, we tie up NOT gates to those variables.

Then, we start implementing the circuit from the last logical operation and continue *backwards*.

We must be careful about the priorities in the expression. Parentheses have the highest priority. NOT operation has the next highest priority. And finally, AND operation has higher priority than OR operation.

Priorities:

Parentheses > NOT > AND > OR

Implementing Logic Circuits From Logical Expressions

Example: Let's implement the logic circuit of $F = A' (B+CD')$

First, we check the complements of variables. It can be seen in the expression that, we need A' and D'. So we tie up NOT gates to these variables. Then we start from the last operation: *A' AND (B+CD')*. We place an AND gate and tie up A' to one of the inputs. Next operation we must consider is *B OR CD'*. So, we put an OR gate and tie up B to one of the inputs, and also tie up the output of OR gate to the other input of the AND gate we placed before. Next, we implement *C AND D'* the same way.