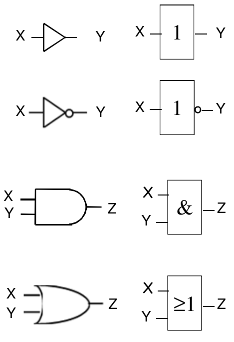
Logic Gates

Logic gates are physical devices, which implement simple Boolean functions.
Some of the simple gates:

ANSI/IEEE-1973 ANSI/IEEE-1984 Truth Table:



BUFFER Y=X

X	Y
0	0
1	1

INVERTER (*NOT*) $\mathbf{Y = \overline{X}}$

X	Y
0	1
1	0

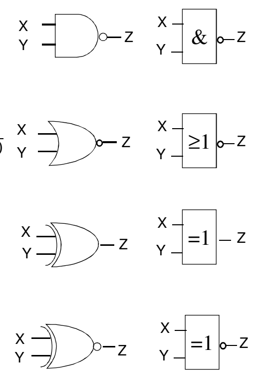
AND Z = X • Y

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

OR Z= X + Y

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

Some of the simple gates (cont'd): ANSI/IEEE-1973 ANSI/IEEE-1984 Truth Table:



NAND (NOT AND) $\mathbf{Z = \overline{(XY)}}$

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

NOR (NOT OR) $\mathbf{Z = \overline{(X+Y)}}$

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

XOR (Difference) $\mathbf{Z = X \oplus Y}$
$\mathbf{Z = X\overline{Y} + \overline{X}Y}$

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

XNOR (Equality) $\mathbf{Z = X \odot Y}$
$\mathbf{Z = XY + \overline{X}\,\overline{Y}}$

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

Integrated Circuits – IC

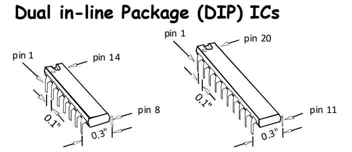
Logic gates are manufactured in integrated circuit (IC) (chip) form.

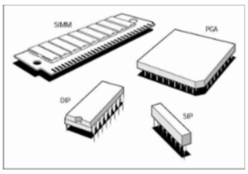
Often, a large number of mixed logic gates are packaged in a single integrated circuit.

For example, an ULSI (Ultra large-scale integration) chip can include more than 100,000 gates.

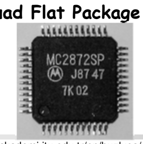
ICs, themselves, come in different types of packages.

Dual in-line Package (DIP) ICs

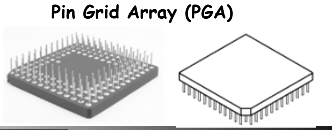




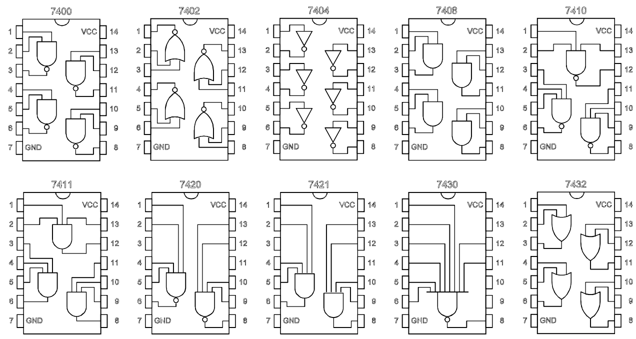
Quad Flat Package (QFP)



Pin Grid Array (PGA)



Examples of 74xx Series



You may find necessary information about ICs in their datasheet catalogs.

Positive and Negative Logic

Boolean values (zero and one) represent physical quantities such as voltage or state of an entity (door is open, light is off).

Assigning "1" to high value, and "0" to low value is called positive logic, and assigning "0" to high value, and "1" to low value is called negative logic.

Example:

Function table of a physical device with 2 inputs and one output is shown below.

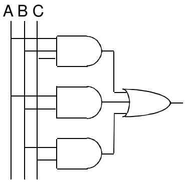
If we use the positive logic, the device can be implemented with an AND gate.

In negative logic system, the device is implemented with an OR gate.

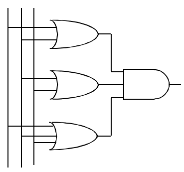
Physical Device			Positive Logic			Negative Logic		
Inputs:		Output:	Inputs:		Output:	Inputs:		Output:
x1	x2	z	x1	x2	z	x1	x2	z
L	L	L	0	0	0	1	1	1
L	H	L	0	1	0	1	0	1
H	L	L	1	0	0	0	1	1
H	H	H	1	1	1	0	0	0

Implementation of Boolean Functions Using Logic Gates

- Sum of Products (SoP)
  - AND gates implement the products.
  - OR gate implements the sum.



- Product of Sums (PoS)
  - OR gates implement the sums.
  - AND gate implements the product.

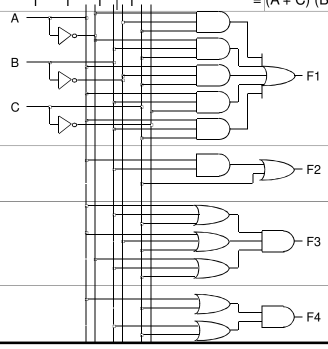


NOT gates can be also used where necessary.

Example: Implementation of a Boolean Function represented (given) by truth table

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

F(A, B, C)= Σm(1,3,5,6,7) 1. canonical form
= A'B'C + A'BC + AB'C + ABC' + ABC = F1
= AB+C = F2 (minimized)
F(A, B, C)= ΠM(0,2,4) 2. canonical form
= (A + B + C) (A + B' + C) (A' + B + C) = F3
= (A + C) (B + C) = F4 (minimized)



F1 = F2 = F3 = F4

1. canonical form (SoP)

minimized (SoP)

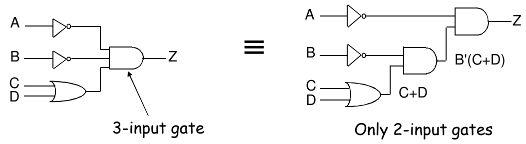
2. canonical form (PoS)

minimized (PoS)

Implementation of Boolean Functions Using Logic Gates (cont'd)

There are many ways to express a Boolean function. We implement each one using different logic gates.

Example: Z = A' • B' • (C + D) = (A' • (B' • (C + D))) (Associative Law)



3-input gate Only 2-input gates

Sometimes, it is necessary to manipulate logic expressions of functions based on the types of available gates (for example, if we have only 2-input AND gates).

Reduction of logic equations is still necessary in order to fit the equations into a small number of ICs.

Functionally Complete Sets of Logic Gates

A set of logic operations is said to be **functionally complete**, if any Boolean function can be expressed using only this set of operations.

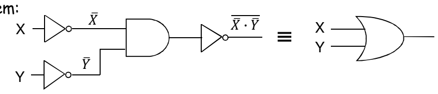
- The set {AND, OR, NOT} is obviously functionally complete because AND, OR, and NOT are main operations that are defined in of the Boolean algebra.

  Any function can be expressed in sum-of-products (or product-of-sums) form, and a sum-of-products expression uses only the AND, OR, and NOT operations.
- Since the set of operations {AND, OR, NOT} is functionally complete, any set of logic gates which can realize {AND, OR, NOT} is also functionally complete.
- For example, {AND, NOT} is also a functionally complete set of gates because OR can be realized using only AND and NOT.

  To prove it we can use De Morgan's theorem.

De Morgan's Theorem:

$\overline{\bar{X} \cdot \bar{Y}} = X + Y$



Since {AND, NOT} is functionally complete, we can express any Boolean function using only AND and NOT.

Universal Logic Gates

If a single gate forms a functionally complete set by itself, then any Boolean function can be realized using only gates of that type.

This type of a gate is called **universal logic gate**.

- The NAND gate is an example of such a gate.

  Remember: the NAND gate performs the AND operation followed by inversion (AND-NOT).
- NOT, AND, and OR can be realized using only NAND gates.
- Thus, any Boolean function can be realized using only NAND gates.
- Similarly, the set consisting only of the binary operator NOR is also functionally complete.
- All other logic functions can be realized using only NOR gates.

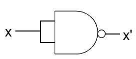
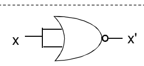
NAND (and also NOR) gates are called **universal logic gates**.

Proof of functional completeness

To prove that NAND and NOR operators are functionally complete , we have to show that AND, OR, NOT operations can be implemented using only NAND (or alternatively, NOR) gates.

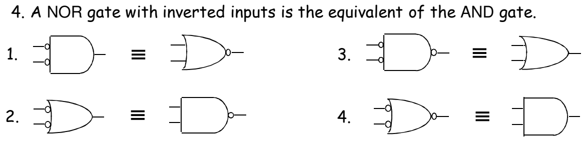
NAND is denoted by symbol |

NOR is denoted by symbol ↓

	NAND	NOR
NOT:	x' = x | x = (x·x)' = x' 	x' = x ↓ x = (x+x)' = x' 
AND:	x·y = ((x·y)')' Involution = (x | y)'	x·y = (x' + y')' de Morgan = (x' ↓ y')
OR:	x+y = (x'·y')' de Morgan x+y = (x' | y')	x+y = ((x+y)')' Involution = (x ↓ y)'

Relation between NAND and NOR

- NAND - NOR Conversions
  - de Morgan:
    1. A' • B' = (A + B)'
    2. A' + B' = (A • B)'
    3. (A' • B')' = A + B
    4. (A' + B')' = (A • B)
- These expressions show that,
  1. An AND gate with inverted inputs is the equivalent of the NOR gate.
  2. An OR gate with inverted inputs is the equivalent of the NAND gate.
  3. A NAND gate with inverted inputs is the equivalent of the OR gate.
  4. A NOR gate with inverted inputs is the equivalent of the AND gate.



Implementation of Boolean functions using only NAND (NOR) gates

There are four different combinations:

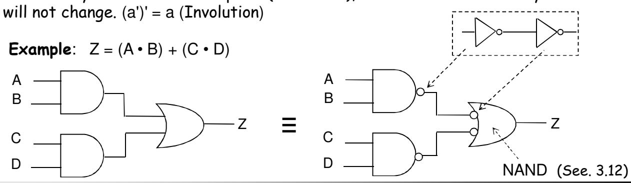
1. Expression in SOP form, implementation using NAND gates
2. Expression in SOP form, implementation using NOR gates
3. Expression in POS form, implementation using NOR gates
4. Expression in POS form, implementation using NAND gates

1.Implementation of Boolean functions in the SOP form using only NAND gates

Shortcut: If we add NOT gates to the outputs of AND gates and to the inputs of the OR gates, we obtain NAND gates. (See 3.12 - 2)

If we always add inverters in pairs (NOT-NOT), the function realized by the circuit will not change. (a')' = a (Involution)

Example: Z = (A • B) + (C • D)



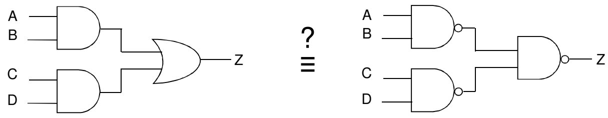
Example (cont'd):

Solution using algebraic conversion:

Expression is inverted twice. (Z')' = Z (Involution)

Z = (A • B) + (C • D) (SoP form)
= [((A • B) + (C • D))']'
= [(A • B)' • (C • D)']' (De Morgan)
= (A | B) | (C | D) (only NAND gates)

Algebraic verification:



Z = [(A • B)' • (C • D)']' Expression using NANDs (circuit on the right)
= [(A' + B') • (C' + D')]'
= [(A' + B')' + (C' + D')']
= (A • B) + (C • D) ✓ Expression of the circuit on left

Implementation using gates with limited number of inputs

Sometimes, it is necessary to implement products (or sums) with many literals using gates that accept only 2 inputs (remember the integrated circuits in 3.4).

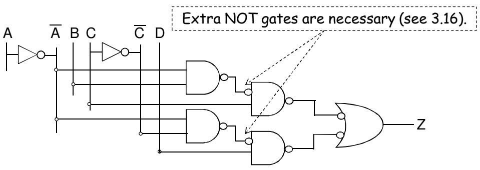
Example:

$Z=\overline{A}BC + \overline{A}\overline{C}D$

Implement this expression using **only 2-input** NAND Gates.

Solution 1:

1. Implementation with the classical gates of the Boolean algebra

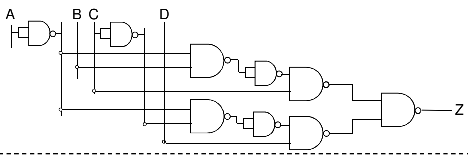


2. Inserting NOT gates to obtain NAND gates

Example (cont'd):

Solution 1:

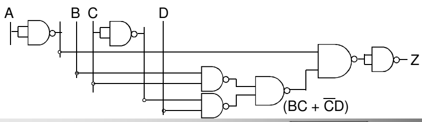
3. Implementation with 2-input NAND gates



Solution 2:

Manipulating the original expression to obtain a simpler circuit

$Z=\overline{A}BC + \overline{A}\overline{C}D = \mathbf{\overline{A}(BC + \overline{C}D)}$

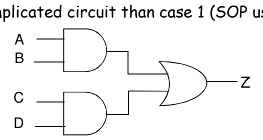


The circuit in solution 2 is cheaper to implement than the circuit in solution 1.
Therefore solution 2 is preferable to solution 1.

2. Implementation of Boolean functions in the SOP form using only NOR gates

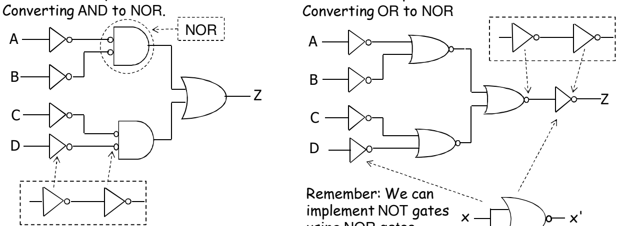
In this case, we obtain a more complicated circuit than case 1 (SOP using NAND).

Example: Z = (A • B) + (C • D)



1. Step:
Converting AND to NOR.

2. Step:
Converting OR to NOR



(a')' = a (Involution)

Remember: We can implement NOT gates using NOR gates.

3. Implementation of Boolean functions in the POS form using only NOR gates

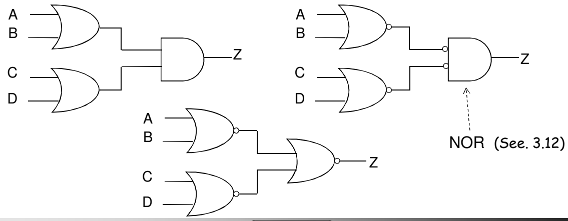
For the expressions in the POS form, using NOR gates is advantageous.

Shortcut :

If we add NOT gates to the outputs of OR gates, and to the inputs of the AND gates, we obtain NOR gates. (See 3.12 -1)

Remember: If we always add inverters in pairs, the function realized by the circuit will not change. (a')' = a (Involution)

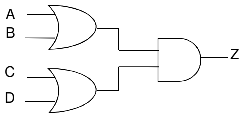
Example: Z = (A + B) • (C + D)



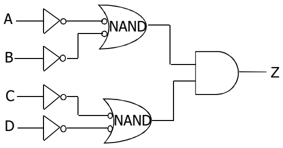
4. Implementation of Boolean functions in the POS form using only NAND gates

In this case, we obtain a more complicated circuit than case 3 (POS using NOR).

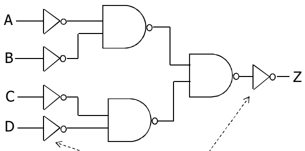
Example: Z = (A + B) • (C + D)



1. Step:
Converting OR to NAND



2. Step:
Converting AND to NAND



Remember: We can implement NOT gates using NAND gates.