Department of Computer Science & Engineering (CSE)

Course Title: Digital Logic Design

Shahriar Rahman Khan
Lecturer
Dept. of CSE, MIST

Course Code: CSE 103
Credit Hr: 3.00
Contact Hr: 3.00

Overview

Map Method
- Two and Three Variable Maps
- Four Variable Map
- Five Variable Map

Product of Sums Simplification

Don’t Care Conditions

NAND Implementation

NOR Implementation

Other Two Level Implementations

NAND and NOR Implementation

Digital circuits are more frequently constructed with NAND & NOR gates than with AND & OR gates.

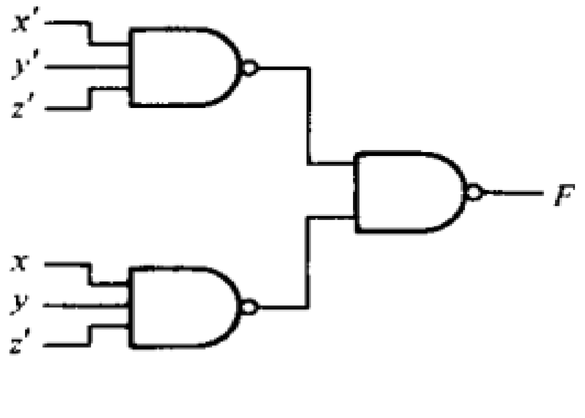
Because these gates are easier to fabricate with electronic components.

That’s why, rules and procedures were developed for the conversion from the Boolean functions given in terms of AND, OR and NOT into equivalent NAND or NOR logic diagrams.

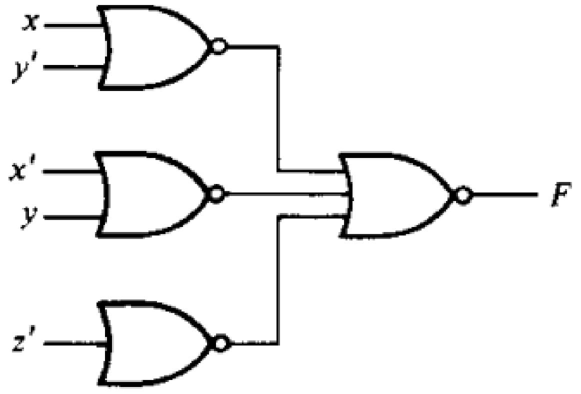
To facilitate the conversion to NAND and NOR logic, the following two graphical symbols are needed.

Two-Level Implementation

Two-level implementation means that any path from input to output contains maximum two gates hence the name two-level for the two levels of gates.



(b) $F = x'y'z' + xyz'$

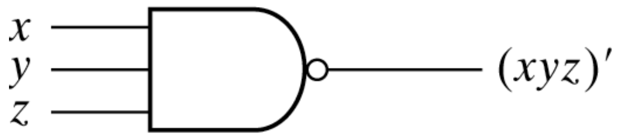


(a) $F = (x + y')(x' + y)z'$

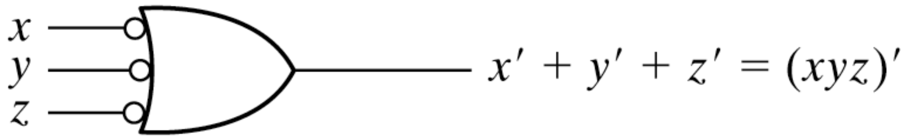
Implementing Two-Level logic using NAND gate requires the Boolean expression to be in Sum of Product (SOP) form.

Implementing Two-Level logic using NOR gate requires the Boolean expression to be in Product of Sum (POS) form.

Two-Level Implementation



(a) AND–invert



(b) Invert–OR

Fig. 3-19 Two Graphic Symbols for NAND Gate

Two-Level NAND Implementation

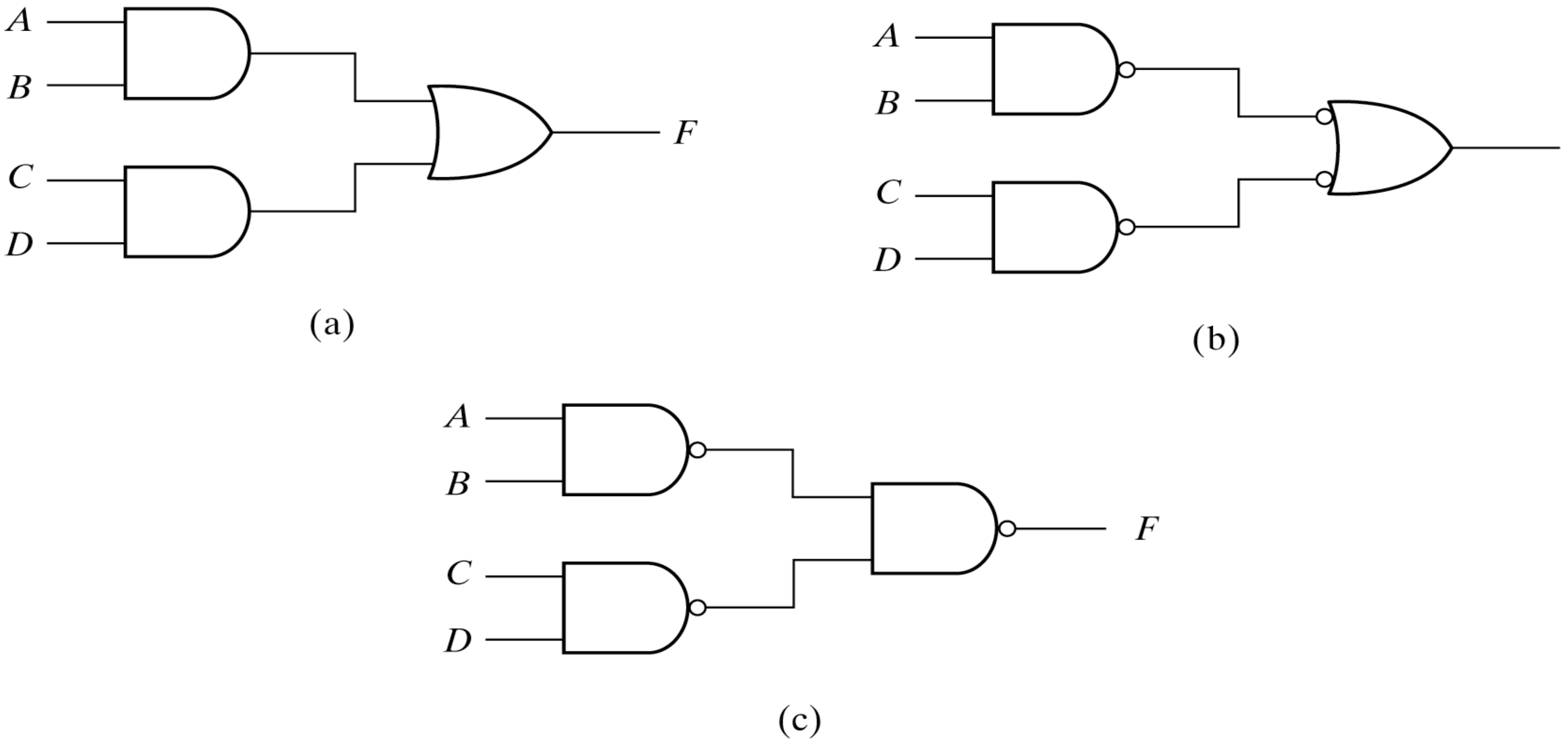


Fig. 3-20 Three Ways to Implement $F = AB + CD$

Conversion to NAND Logic

- Simplify the function and express the Boolean function in SOP. After simplification, let

$$F(x,y,z) = x'y' + yz + x'$$

- Pass the literals of each product term to each NAND gate. Each NAND gate should have at least two literals. This group of NAND gates are called first-level gates. In first level, no of NAND gates will be no of product terms.
- Pass the outputs of the first-level to a single NAND gate. This is called second-level. This NAND gate maybe either AND-invert or Invert-OR.
- A term with a single literal requires an inverter in the first level or maybe complemented and used as an input in the second level NAND gate.

Conversion to NAND Logic

Let, F(x,y,z)=x'y' + yz + x'

First Level

i. Pass x' and y' to NAND gate = (x'y')'

ii. Pass y and z to NAND gate = (yz)'

iii. Complement x' = x

Second Level

i. Pass (x'y')' , (yz)' and x to another NAND gate, which is called Second Level gate

ii. F(x,y,z)= ((x'y')' . (yz)' . x)'

iii. F(x,y,z)= ((x'y')')' + ((yz)')' + (x)' [Apply de-morgan]

iv. F(x,y,z)= x'y'+ yz + x' [Apply de-morgan again]

Another way of Conversion to NAND Logic

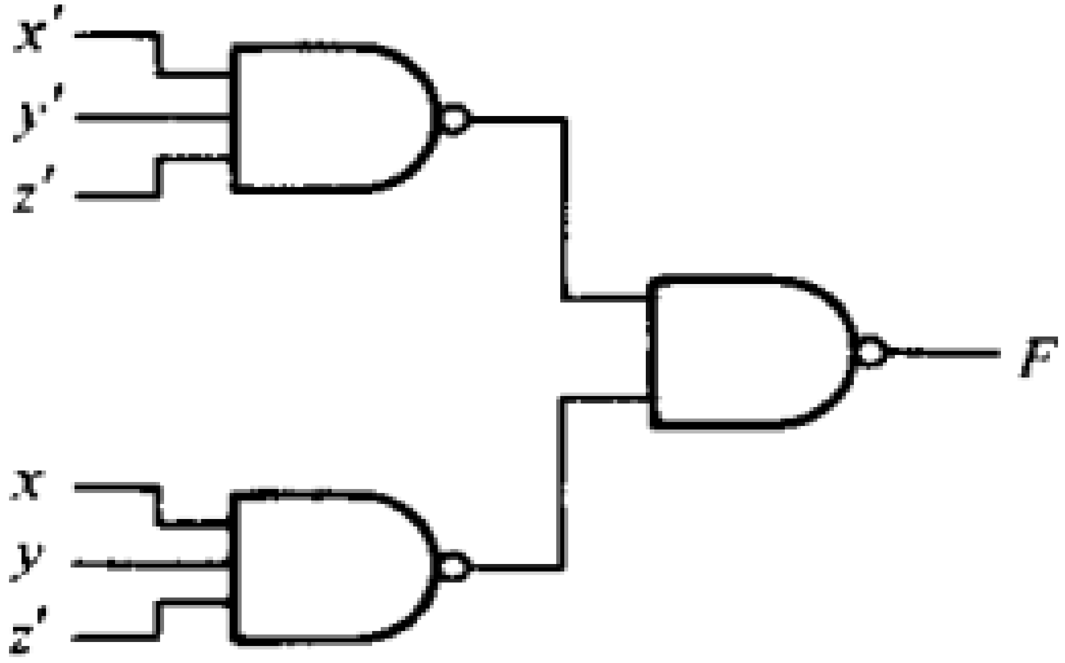
Given, F(x,y,z)=Σ(0,6) . Simplify the function in SOP form using K-map

	00	01	11	10
0	1	0	0	0
1	0	0	0	1

F=x’y’z’+xyz’

Now, we can implement this using Two-level implementation. Again, we can try to simplify the complement of the function in SOP, but combining the 0’s in the K-Map

Another way of Conversion to NAND Logic



(b) $F = x'y'z' + xyz'$

Another way of Conversion to NAND Logic

F(x,y,z)=Σ(0,6)

	00	01	11	10
0	1	0	0	0
1	0	0	0	1

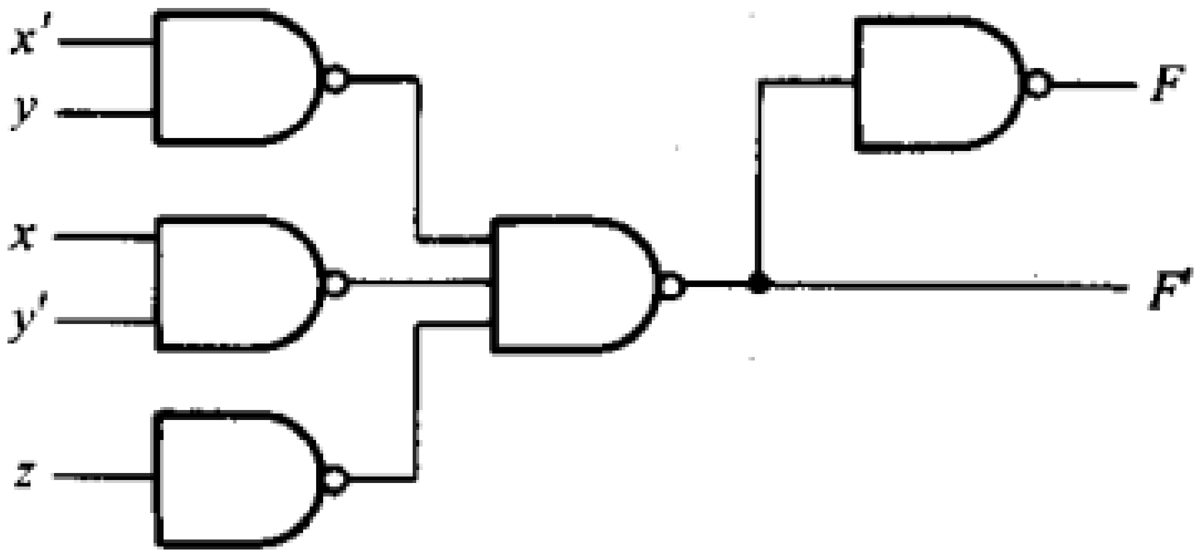
F’=x’y+xy’+z

We can implement this using Two-level NAND implementation

Now, if the output F is required, then we add a one-input NAND gate to invert the function. This gives a three-level implementation.

N.B. It is assumed that, the input variables are available in true form or complement form. If not, then we have to use inverter to complement.

Another way of Conversion to NAND Logic



(c) $F' = x'y + xy' + z$

Examples

F(A,B,C,D)=A’B’C+AC’+ACD+ACD’+A’B’D’

Step 1; Simplify the function in SOP form using K-map

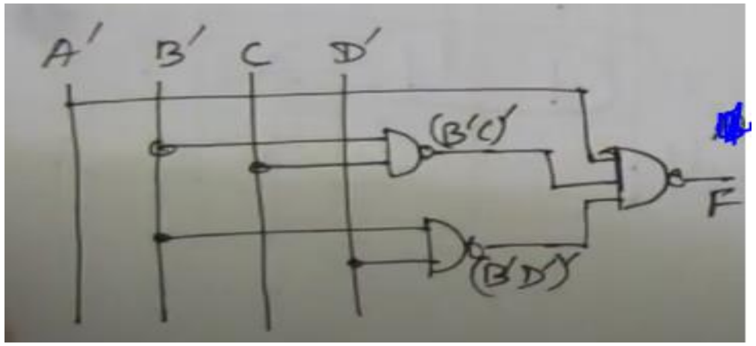
	00	01	11	10
00	1	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

F=A+B’C+B’D’

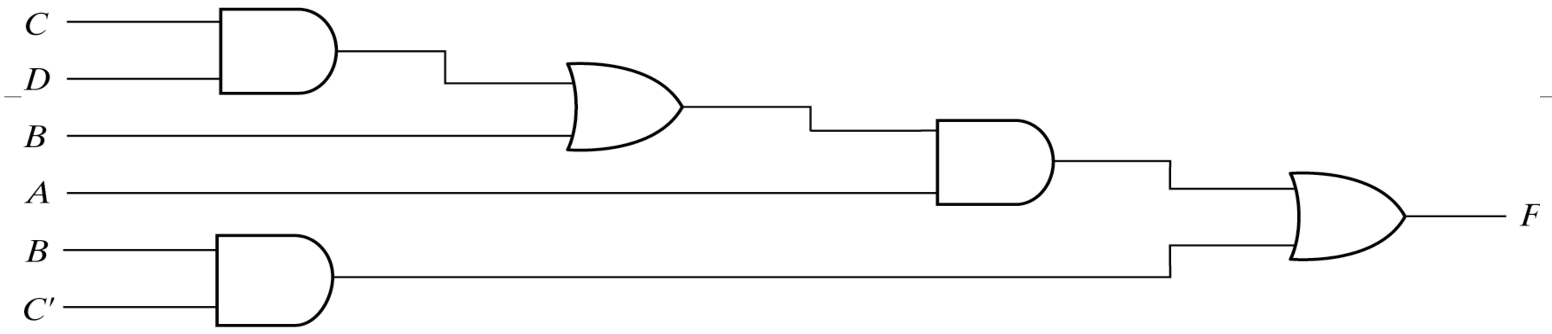
F={(A+B’C+B’D’)’}’

F={A’ . (B’C)’ . (B’D’)’}’

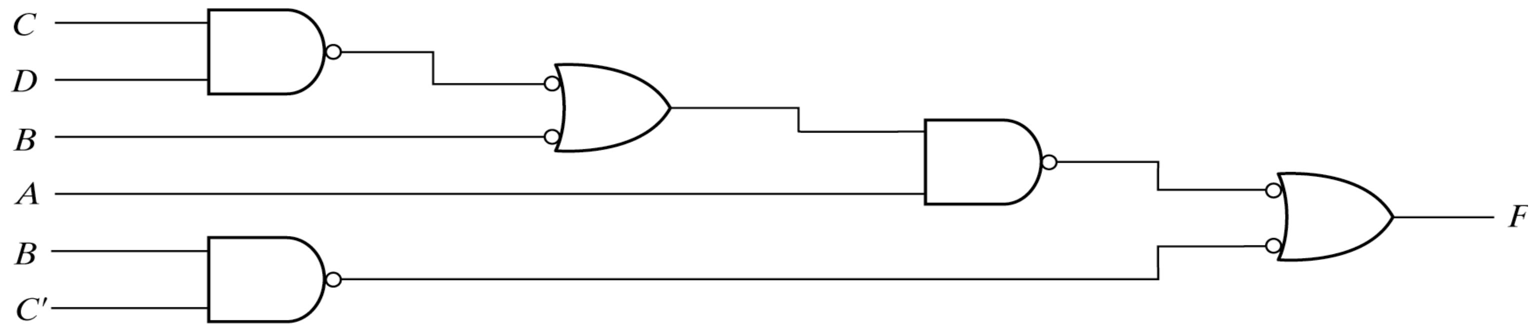
You learnt this in intermediate



Examples



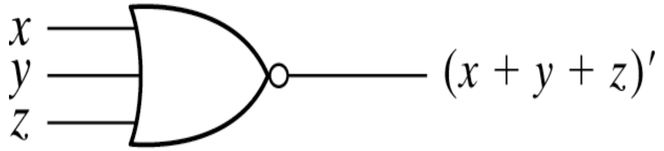
(a) AND-OR gates



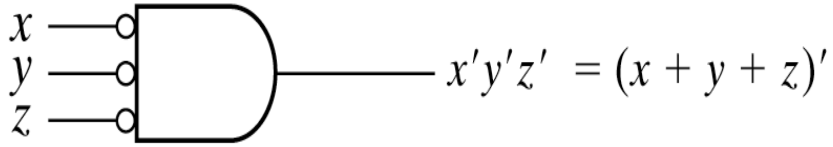
(a) NAND gates

Fig. 3-22 Implementing $F = A(CD + B) + BC$

Two-Level NOR implementation



(a) OR–invert



(a) Invert–AND

Fig. 3-25 Two Graphic Symbols for NOR Gate

Conversion to NOR Logic

Simplify the function using K-Map and express the Boolean function in POS. After simplification, let

$$F(x, y, z) = (x' + y') \cdot (y + z) \cdot x'$$

Pass each **sum** term to each NOR gate. Each NOR gate should have at least two literals. This group of NOR gates are called first-level (level-1) gates.

Pass each output of the first-level to a single NOR gate. This is called second-level gate.

If there is a single literal in the simplified Boolean function then invert it in the first-level using a NOR gate.

Conversion to NOR Logic

Let, $F(x,y,z) = (x' + y') . (y + z) . x'$

- First Level
  1. Pass x' and y' to NOR gate = $(x' + y')'$
  2. Pass y and z to NOR gate = $(y + z)'$
  3. Complement x' = x

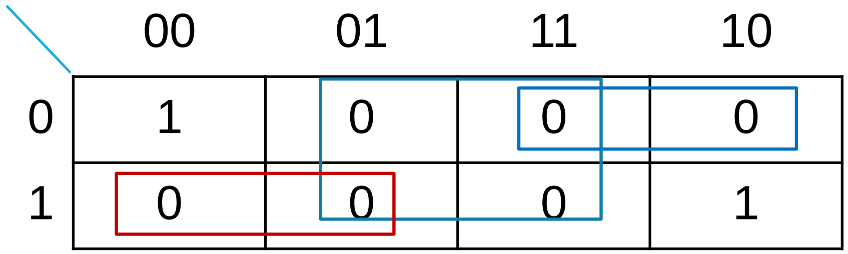
- Second Level
  1. Pass $(x' + y')'$, $(y + z)'$ and x to another NOR gate, which is called Second Level gate
  2. $F(x,y,z)= \{(x' + y')' + (y + z)' + x\}'$
  3. $F(x,y,z)= ((x' + y')')' . ((y + z)')' . (x)'$ [Apply de-morgan]
  4. $F(x,y,z)= (x' + y') . (y + z) . x'$ [Apply de-morgan again]

Another way of Conversion to NOR Logic

F(x,y,z) = Σ(0,6)

F(x,y,z)=Π(1,2,3,4,5,7)

Step 1; Simplify the function in POS form using K-map



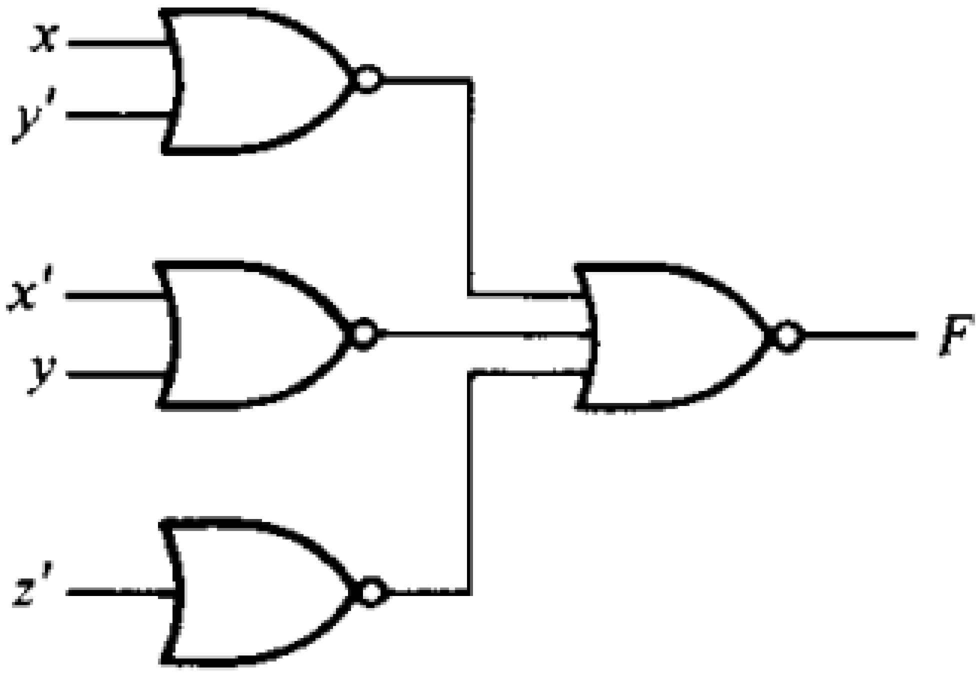
	00	01	11	10
0	1	0	0	0
1	0	0	0	1

F=(x + y’) . (x’ + y) . z’

Now, we can implement this using Two-level NOR implementation.

Or, we can try to simplify the complement of the function in POS, but combining the 1’s in the K-Map

Another way of Conversion to NOR Logic



(a) $F = (x + y')(x' + y)z'$

Another way of Conversion to NOR Logic

F(x,y,z)=Σ(0,6)

F(x,y,z)=Π(1,2,3,4,5,7)

Step 1; Simplify the function in POS form using K-map

	00	01	11	10
0	1	0	0	0
1	0	0	0	1

F’=(x + y + z) . (x’ + y’ + z)

F = {(x + y + z) . (x’ + y’ + z)}’

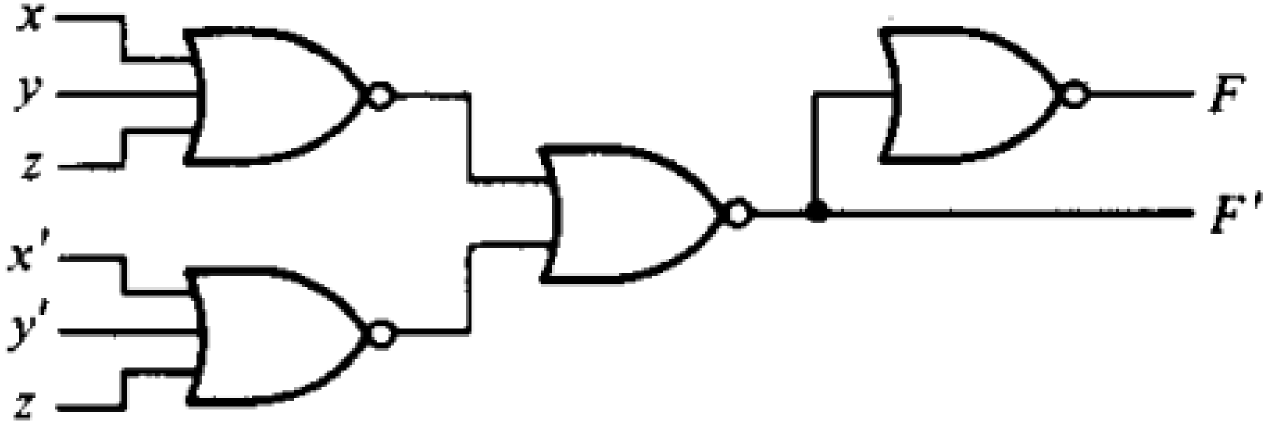
={(x+y+z)’+(x’+y’+z)’}

=x’y’z’+xyz’

We can implement this F’ using Two-level NOR implementation

Now, if the output F is required, then we add a one-input NOR gate to invert the F’ function. This gives a three-level implementation.

Another way of Conversion to NOR Logic



(b) $F' = (x + y + z)(x' + y' + z)$

Other Two-Level Implementations

Consider the four Logic gates AND, OR, NAND & NOR. Since, there are 4 Logic gates, we will get 16 possible ways of realizing two level logic. Those are

- AND-AND, AND-OR, AND-NAND, AND-NOR, OR-AND, OR-OR, OR-NAND, OR-NOR, NAND-AND, NAND-OR, NAND-NAND, NAND-NOR, NOR-AND, NOR-OR, NOR-NAND, NOR-NOR.

These two level logic realizations can be classified into the following two categories.

- Degenerative form
- Non-degenerative form

Degenerative Form

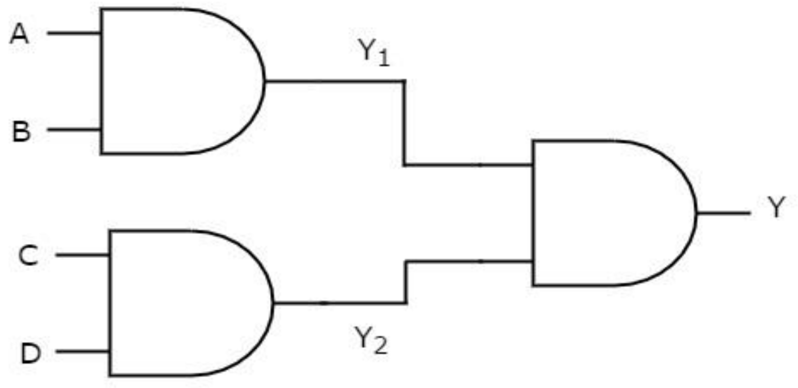
If the output of two level logic realization can be obtained by using single Logic gate, then it is called as **degenerative form**. Obviously, the number of inputs of single Logic gate increases. Due to this, the fan-in of Logic gate increases. Fan-in is **the number of inputs a logic gate can handle**.

Only **6 combinations** of two level logic realizations out of 16 combinations come under degenerative form. Those are

AND-AND	OR-OR
AND-NAND	OR-NOR
NAND-NOR	NOR-NAND

AND-AND Logic

In this logic realization, AND gates are present in both levels. Below figure shows an example for **AND-AND logic** realization.

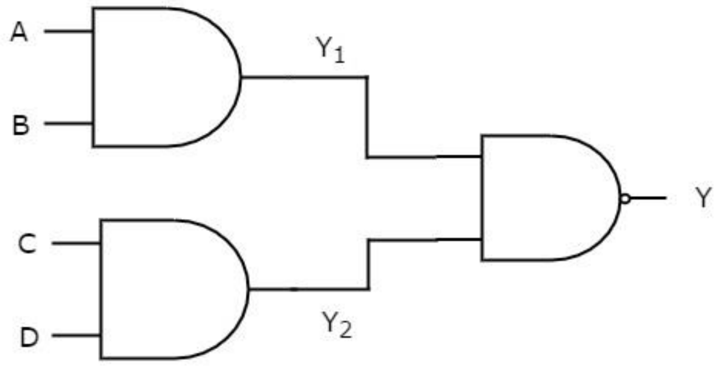


We will get the outputs of first level logic gates as $Y_1=AB$ and $Y_2=CD$

In second level. So, the output of this AND gate is Y=Y1Y2=(AB)(CD)=ABCD. Which can be implemented using a single 4 input AND gate.

AND-NAND Logic

In this logic realization, AND gates are present in first level and NAND gate is present in second level. Below figure shows an example for **AND-NAND logic** realization.



We will get the outputs of first level logic gates as Y_1=AB and Y_2=CD

In second level. So, the output of this NAND gate is Y=(Y1Y2)’=((AB)(CD))’=(ABCD)’. Which can be implemented using a single 4 input NAND gate.

Non-Degenerative Form

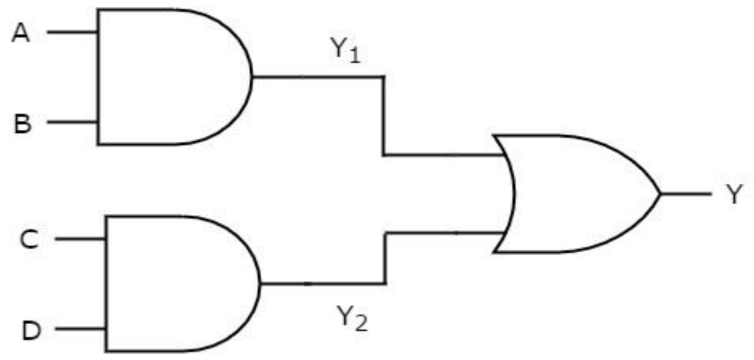
If the output of two level logic realization can’t be obtained by using single logic gate, then it is called as **non-degenerative form**.

The remaining **10 combinations** of two level logic realizations come under non-degenerative form. Those are

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-NAND	AND-NOR
NAND-OR	NOR-AND

AND-OR Logic

In this logic realization, AND gates are present in first level and OR gate is present in second level. Below figure shows an example for **AND-OR logic** realization.

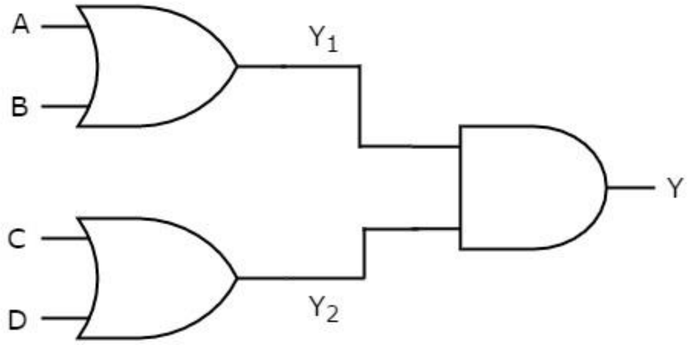


We will get the outputs of first level logic gates as Y1=AB and Y2=CD

In second level. So, the output of this OR gate is Y = Y1+Y2 = AB+CD. This Boolean function is in **Sum of Products** form. Since, we can’t implement it by using single logic gate, this AND-OR logic realization is a **non-degenerative form.**

OR-AND Logic

In this logic realization, OR gates are present in first level and AND gate is present in second level. Below figure shows an example for OR-**AND logic** realization.



We will get the outputs of first level logic gates as Y1=A+B and Y2=C+D

In second level. So, the output of this OR gate is Y = Y1Y2 = (A+B)(C+D). This Boolean function is in **Products of Sums** form. Since, we can't implement it by using single logic gate, this OR-AND logic realization is a **non-degenerative form.**

AND-OR-INVERT Implementation

When the output of an AND-OR circuit is complemented, it results in an AND-OR-Invert circuit.

Recall that, AND-OR logic directly implements SOP expressions. POS expressions can be implemented with AND-OR-Invert logic.

- X=$(\bar{A} + \bar{B})(\bar{C} + \bar{D})$
- X=$(\overline{AB})(\overline{CD})$
- X=$\overline{\overline{\overline{(AB)}\;\overline{(CD)}}}$
- X=$\overline{\overline{\overline{AB}} + \overline{\overline{CD}}}$
- X=$\overline{AB + CD}$

For a 4-input AND-OR-Invert circuit, the output X is 0 if both input A and input B are 1 or both input C and input D are 1

AND-OR-INVERT Implementation

The two forms NAND-AND and AND-NOR are equivalent forms and both perform the AND-OR-Invert function.

- $F=(AB+CD+E)'$

AND-OR-INVERT Implementation

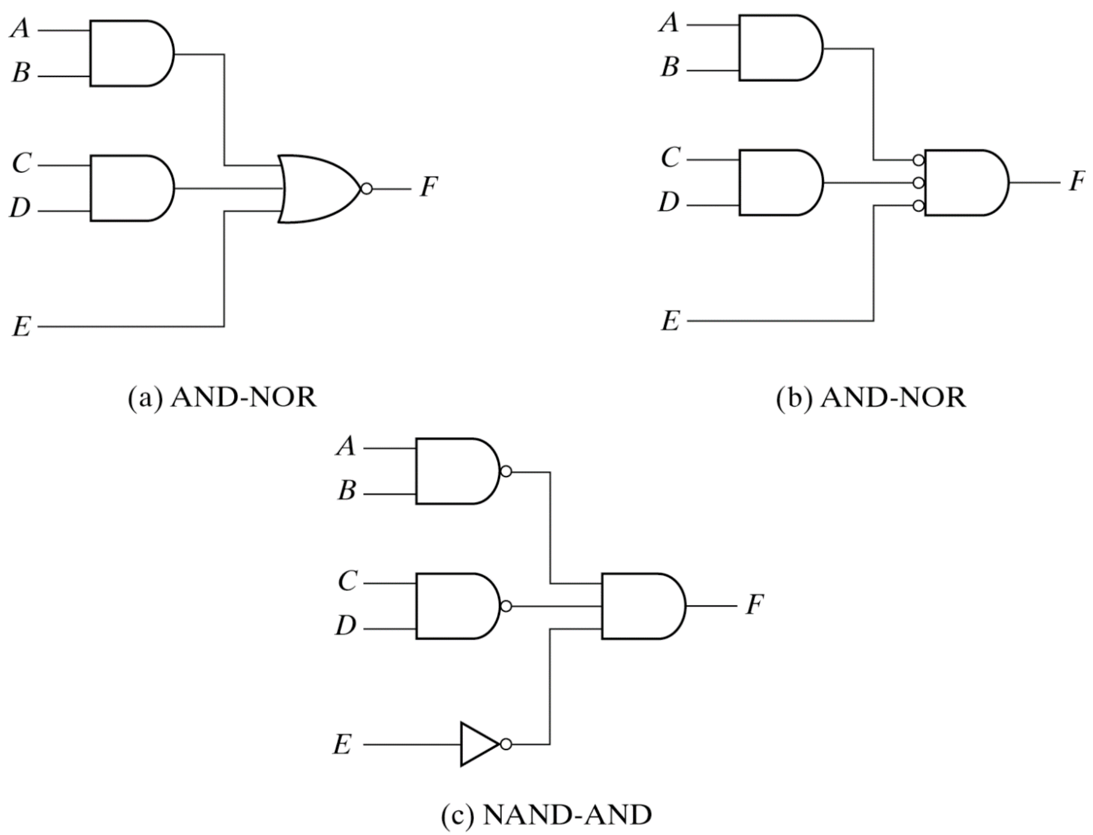


Fig. 3-29 AND-OR-INVERT Circuits; $F = (AB + CD + E)'$

OR-AND-INVERT Implementation

The two forms OR-NAND and NOR-OR are equivalent forms and both perform the OR-AND-Invert function.

- $F=[(A+B)(C+D).E]'$

OR-AND-INVERT Implementation

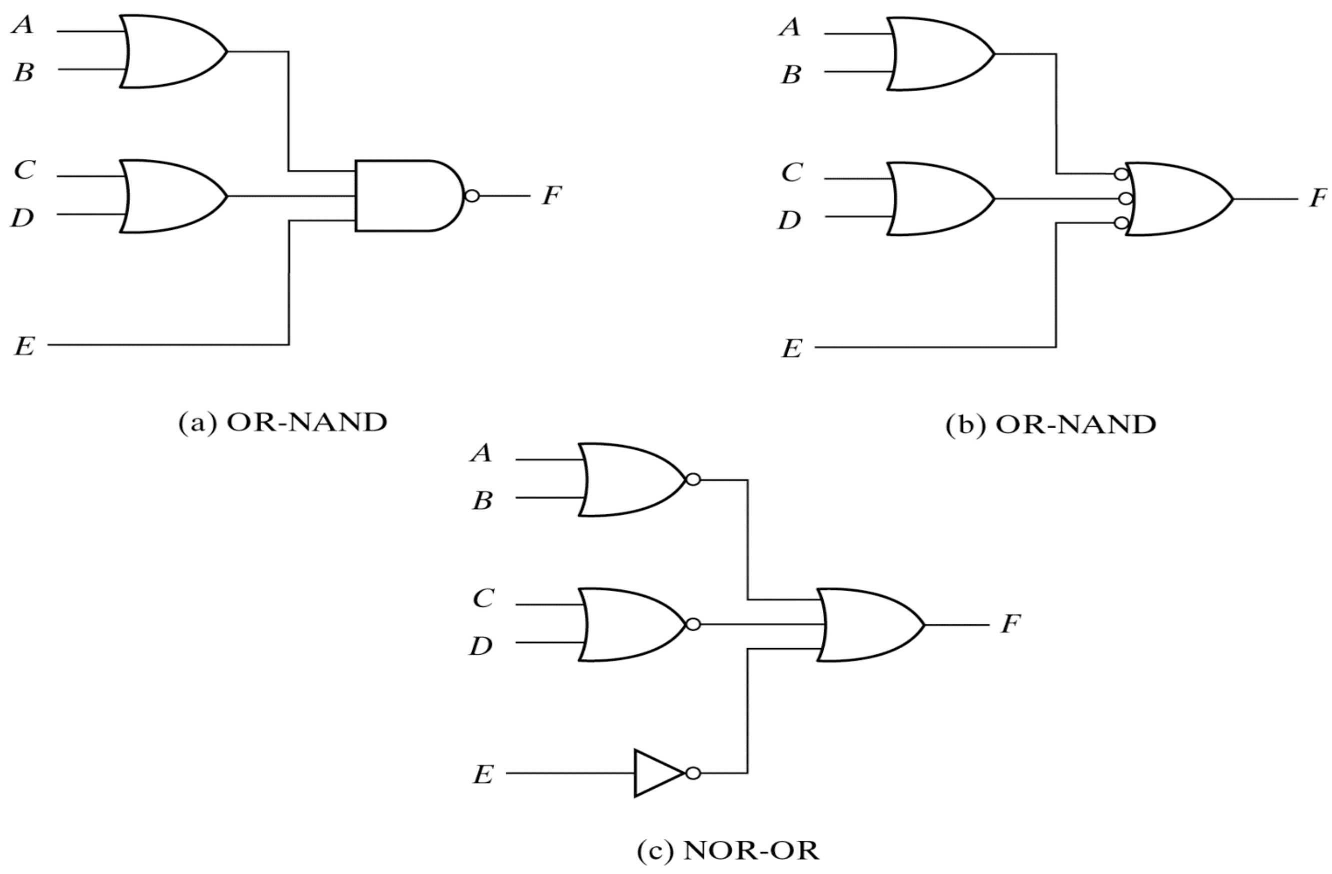


Fig. 3-30 OR-AND-INVERT Circuits; $F = [(A + B)(C + D)E]'$

GOOD NEWS!

THE CLASS IS OVER…

THANK YOU!