Digital Electronics

Chapter 4
Combinational Logic

Outline of Chapter 4

- 4.1 Introduction
- 4.2 Design Procedure
- 4.3 Hazards
- 4.4 Adders
- 4.5 Subtractors
- 4.6 Code Conversion
- 4.7 Analysis Procedure
- 4.8 Multilevel NAND Circuits
- 4.9 Multilevel NOR Circuits
- 4.10 Exclusive OR & Equivalent Functions

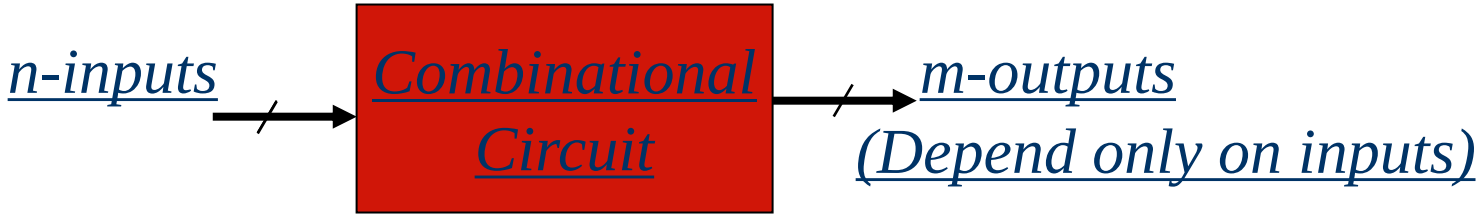
INTRODUCTION

- A combinational circuit consists of logic gates whose outputs, at any time, are determined by combining the values of the inputs.
- For n input variables, there are 2^n possible binary input combinations.
- For each binary combination of the input variables, there is one possible output.

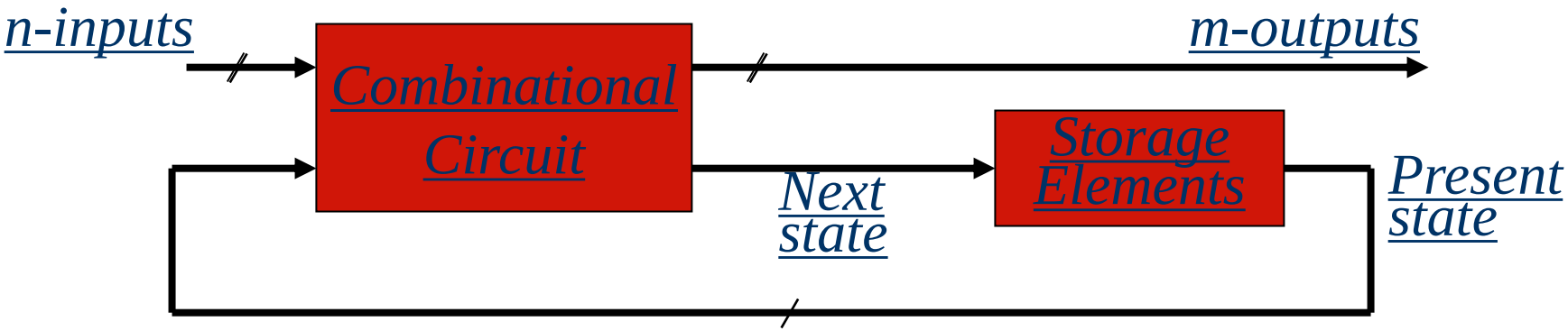
INTRODUCTION

- Hence, a combinational circuit can be described by:
  1. A truth table that lists the output values for each combination of the input variables, or
  2. *m* Boolean functions, one for each output variable.

INTRODUCTION



Combinational Circuit



Sequential Circuit

DESIGN PROCEDURE

- The procedure involves the following steps:

1. State the problem.
2. Determine no. of available input variables and required output variables.
3. Assign letter symbols to the input and output variables.
4. Derive the truth table that defines the required relationship between inputs and outputs.
5. Obtain simplified Boolean function for each output.
6. Draw the logic diagram.

HAZARDS

- A **hazard**, if exists, in a digital circuit causes a temporary fluctuation in output of the circuit.
- In other words, a hazard in a digital circuit is a temporary disturbance in ideal operation of the circuit which if given some time, gets resolved itself.
- These disturbances or fluctuations occur when different paths from the input to output have different delays and due to this fact, changes in input variables do not change the output instantly but do appear at output after a small delay caused by the circuit building elements, i.e., logic gates.

HAZARDS

- There are three different kinds of hazards found in digital circuits

1. Static hazard

   A **static hazard** takes place when change in an input causes the output to change momentarily before stabilizing to its correct value.

2. Dynamic hazard

   A **dynamic hazard** is the possibility of an output changing more than once as a result of a single input change.

3. Functional/Hybrid hazard

   Functional hazards are non-solvable hazards which occurs when more than one input variable changes at the same time.

ADDERS(HALF ADDER)

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

Table 4-3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

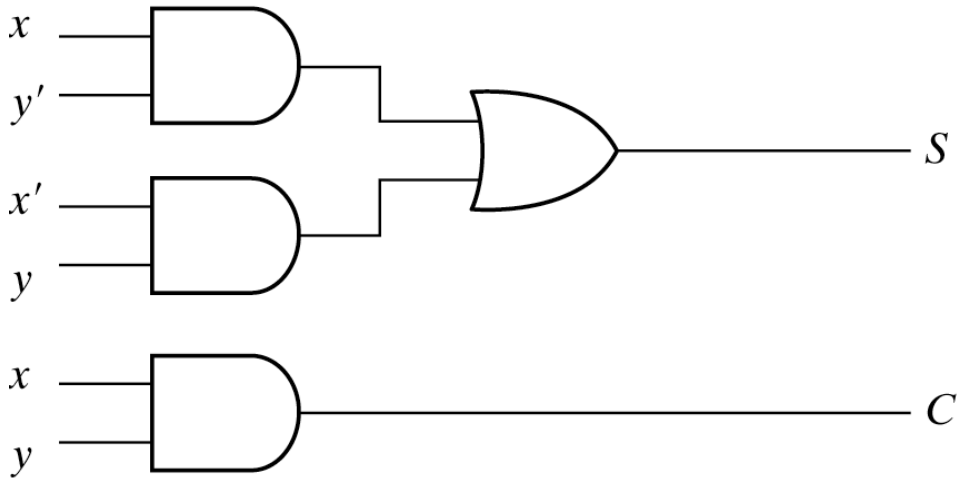
S: Sum
C: Carry

$S = x'y + xy'$

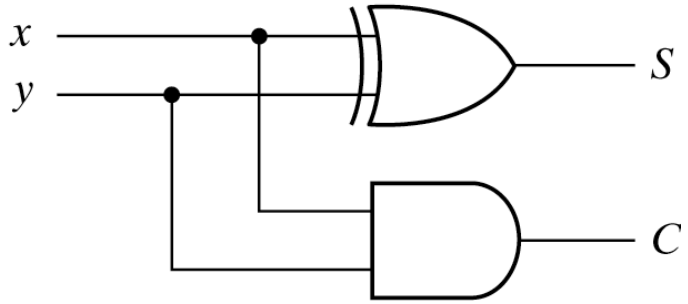
$C = xy$

ADDERS(HALF ADDER)

Implementation of HALF ADDER



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

ADDERS (FULL ADDER)

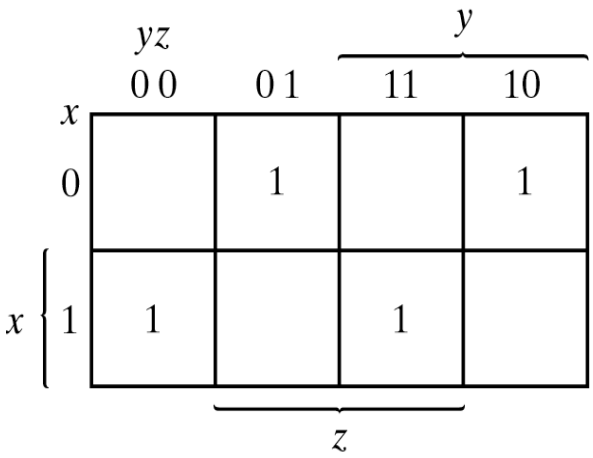
- One that performs the addition of three bits(two significant bits and a previous carry) is a full adder.

Table 4-4
Full Adder

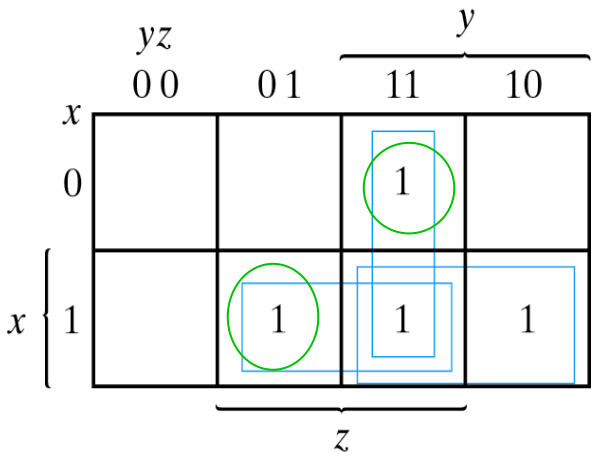
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

ADDERS (FULL ADDER)

Simplified Expression



$S = x'y'z + x'yz' + xy'z' + xyz$



$C = xy + xz + yz$
$= xy + xy'z + x'yz$

Fig. 4-6 Maps for Full Adder

S = x'y'z + x'yz' + xy'z' + xyz

C = xy + xz + yz

ADDERS (FULL ADDER)

Full Adder Implementation in SOP

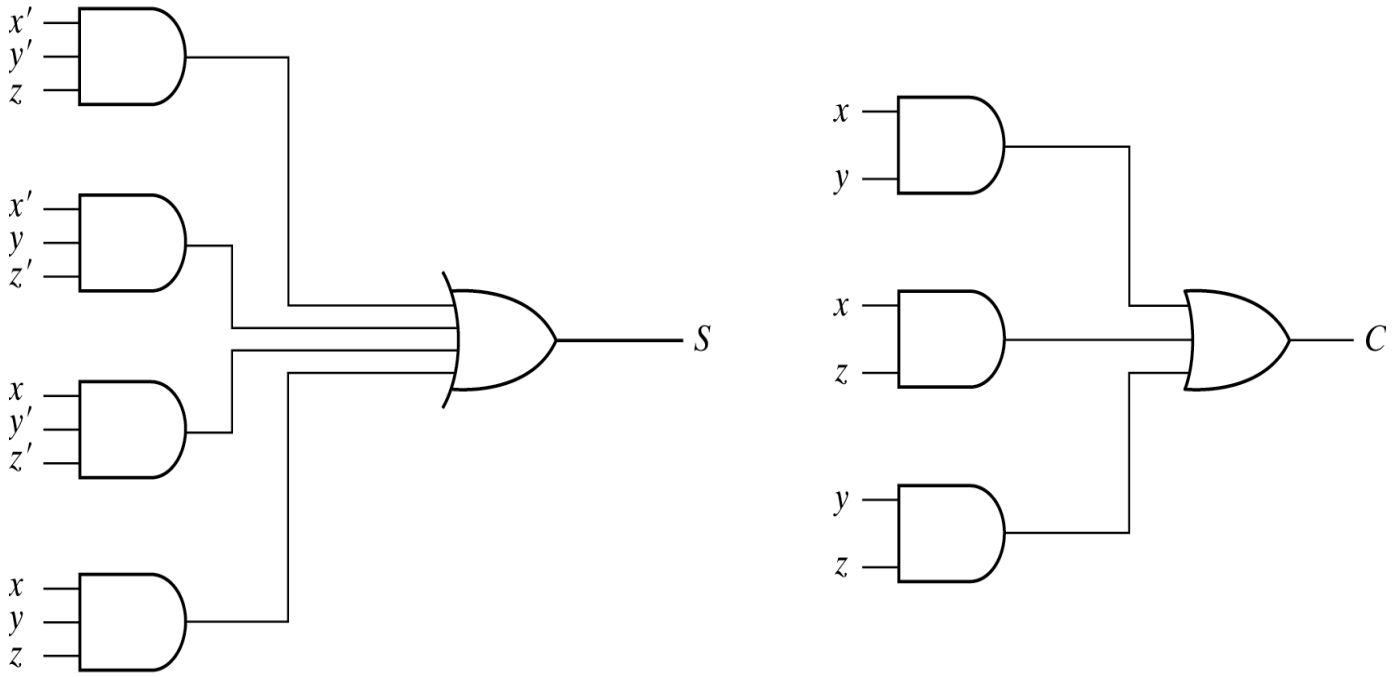


Fig. 4-7 Implementation of Full Adder in Sum of Products

ADDERS (FULL ADDER)

- Full-adder can also implemented with two half adders and one OR gate (Carry Look-Ahead adder).

$$S = z \oplus (x \oplus y)$$
$$= z'(xy' + x'y) + z(xy' + x'y)'$$
$$= xy'z' + x'yz' + xyz + x'y'z$$
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

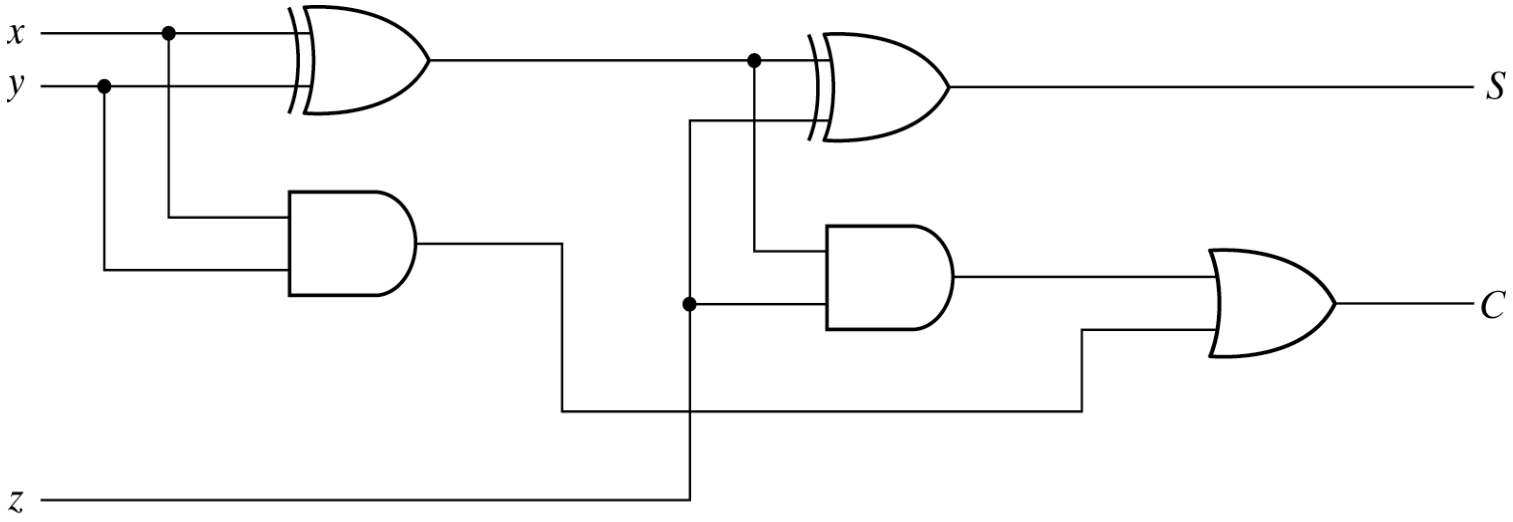


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

SUBTRACTORS(HALF SUBTRACTOR)

- A combinational circuit that performs the subtraction of two bits is called a half subtractor.
- The truth table for the half subtractor is listed below:

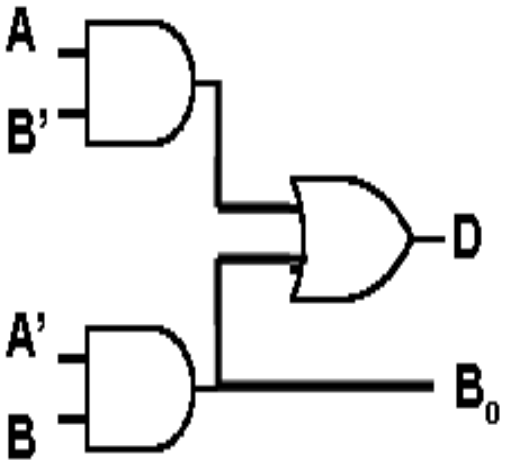
Input		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Difference = A'B+AB'

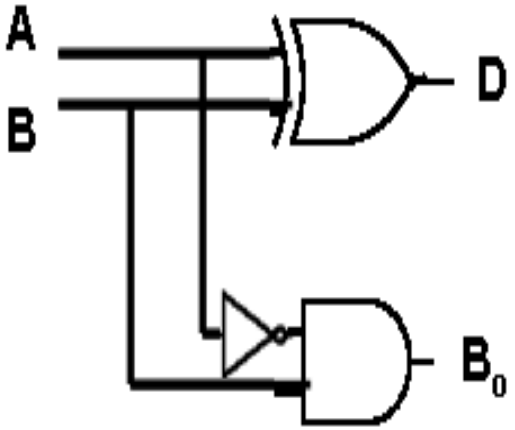
Borrow = A'B

SUBTRACTORS(HALF SUBTRACTOR)

Implementation of HALF SUBTRACTOR



(a)



(b)

D = A'B+AB'
B = A'B

D = A'B+AB'
B = A'B

SUBTRACTORS(FULL SUBTRACTOR)

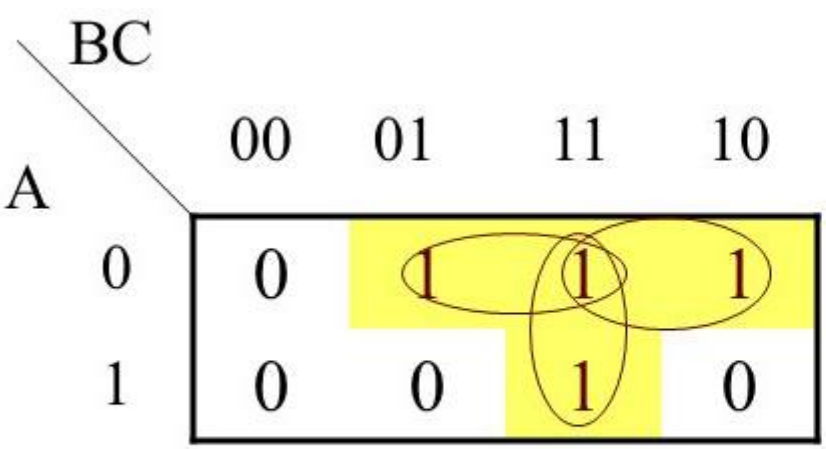
- One that performs the subtraction of three bits is a full subtractor.

Input			Output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

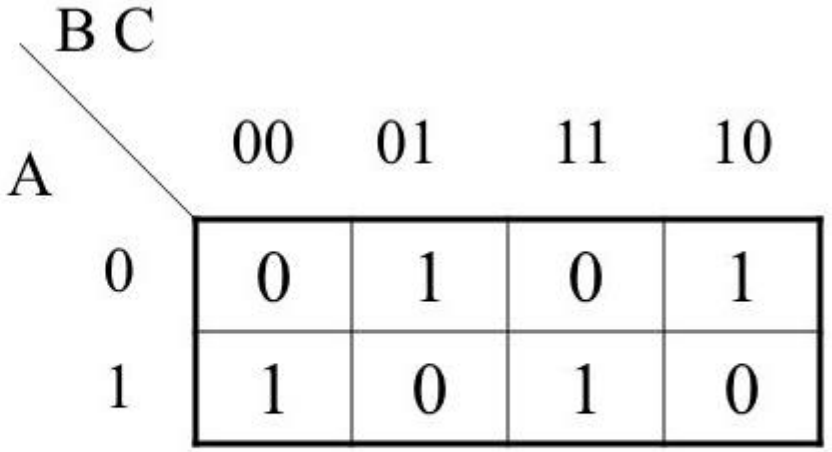
Truth table of full subtractor

SUBTRACTORS(FULL SUBTRACTOR)

Simplified Expression



For Borrow



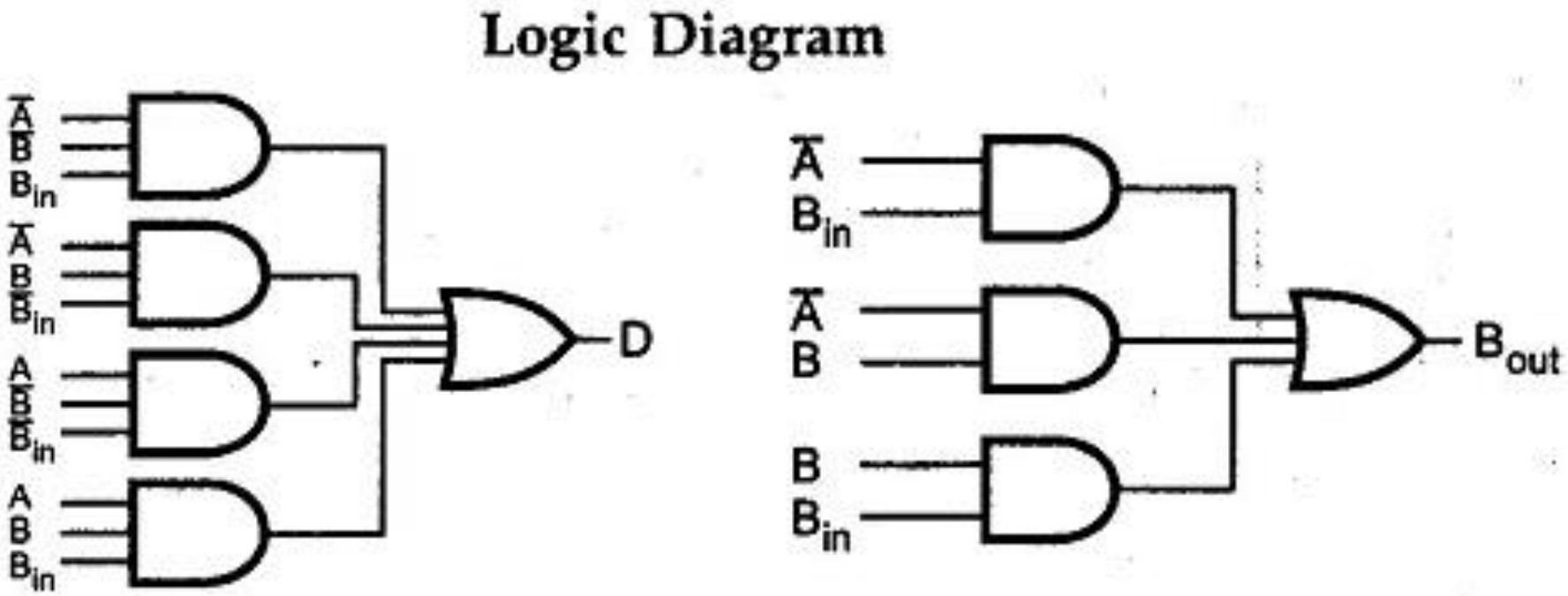
For Difference

D = A’B’C+A’BC’+AB’C’+ABC

C = A’C+A’B+BC

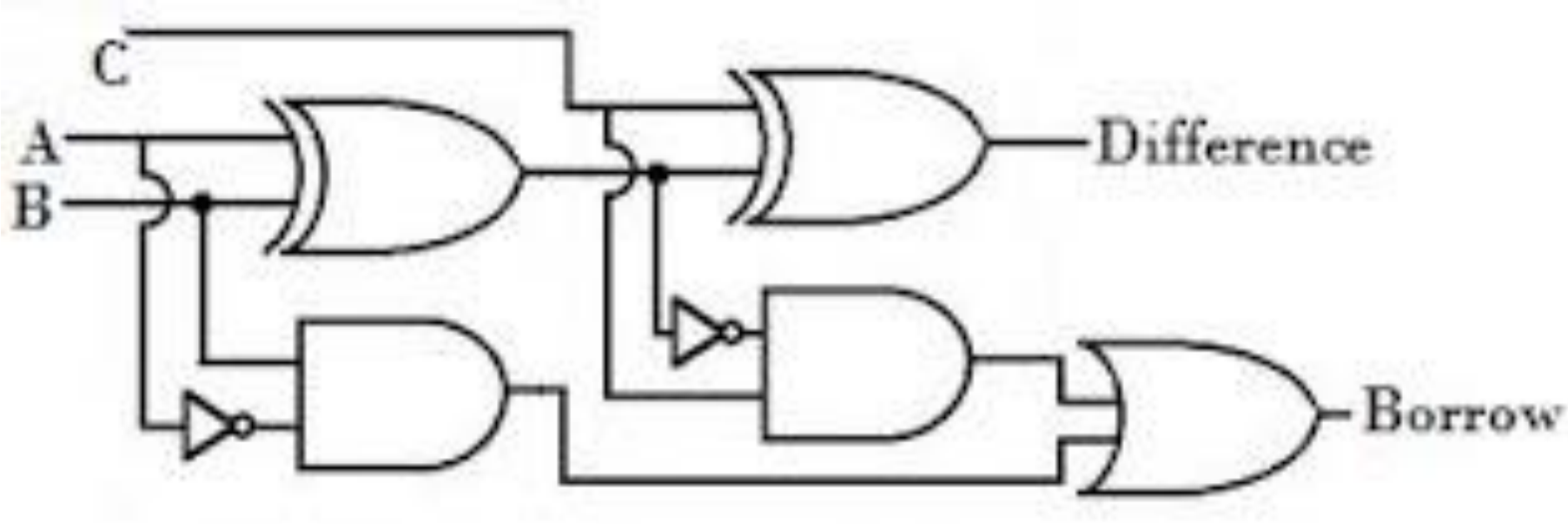
SUBTRACTORS(FULL SUBTRACTOR)

Full Subtractor Implementation in SOP



SUBTRACTORS(FULL SUBTRACTOR)

- Logic diagram of Full Subtractor has been shown in the figure below:



CODE CONVERSION

- Code conversion is a process that converts data from one type of binary code to another type of binary code.

- A code converter is a logic circuit that changes data presented in one type of binary code to another type of binary code, such as BCD to binary, BCD to 7–segment, binary to BCD, BCD to XS3, binary to Gray code, and Gray code to binary.

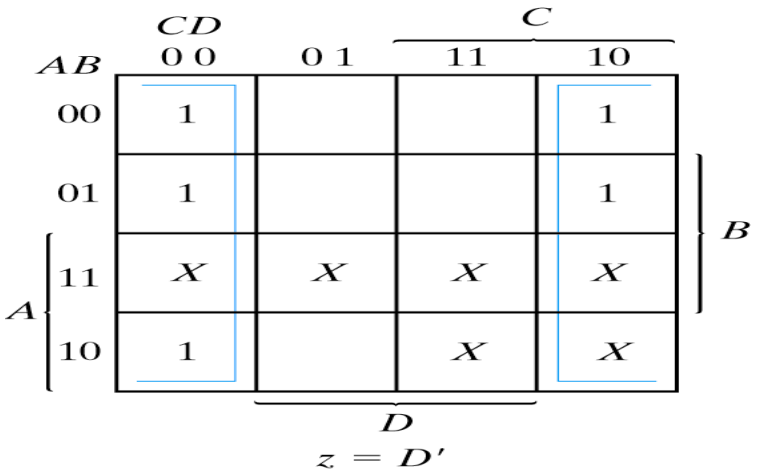
CODE CONVERSION

- BCD to XS-3 Code Conversion

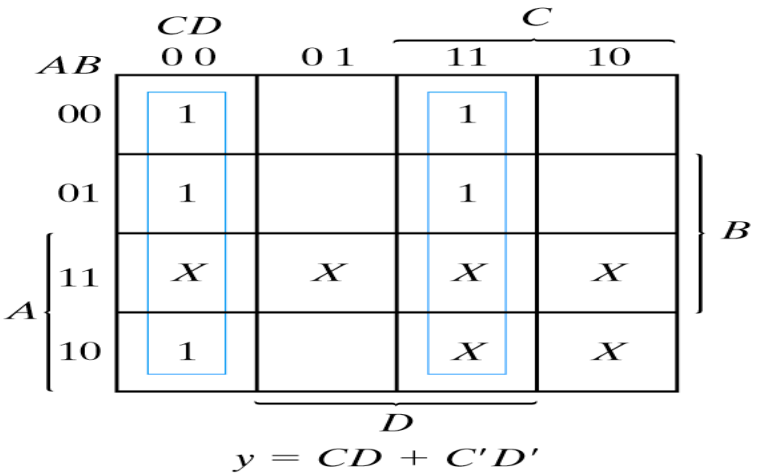
Input (Std BCD code)					Output (XS3 Code)			
A	B	C	D		w	x	y	z
0	0	0	0		0	0	1	1
0	0	0	1		0	1	0	0
0	0	1	0		0	1	0	1
0	0	1	1		0	1	1	0
0	1	0	0		0	1	1	1
0	1	0	1		1	0	0	0
0	1	1	0		1	0	0	1
0	1	1	1		1	0	1	0
1	0	0	0		1	0	1	1
1	0	0	1		1	1	0	0
1	0	1	0		X	X	X	X
1	0	1	1		X	X	X	X
1	1	0	1		X	X	X	X
1	1	1	0		X	X	X	X
1	1	1	1		X	X	X	X

CODE CONVERSION

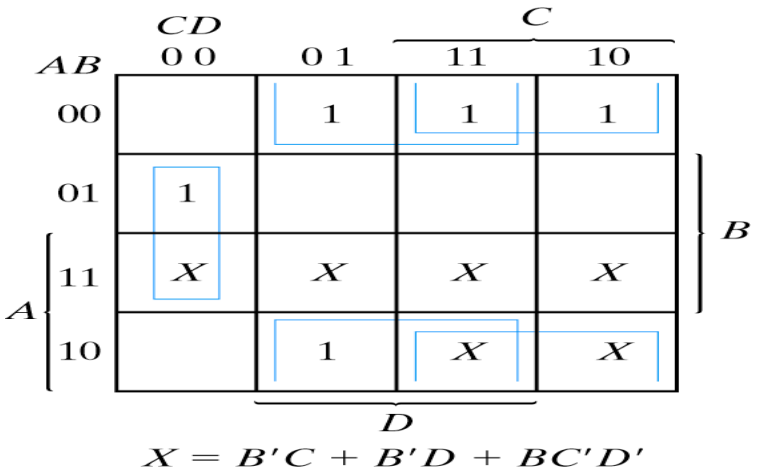
K- Maps for Simplification and Simplified Boolean Functions



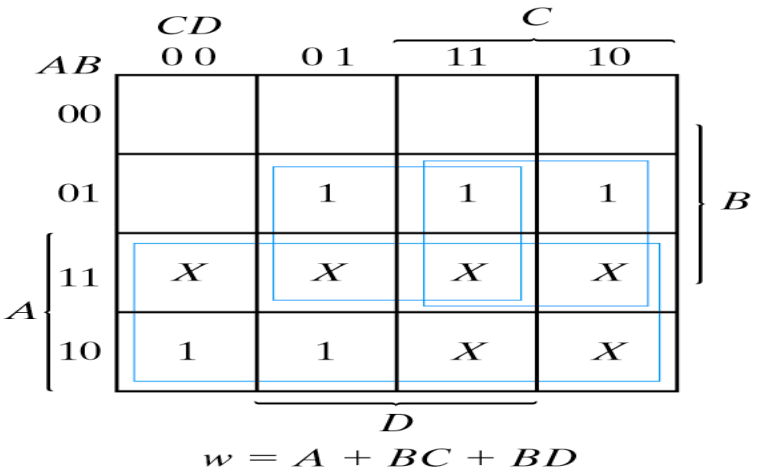
$z = D'$



$y = CD + C'D'$

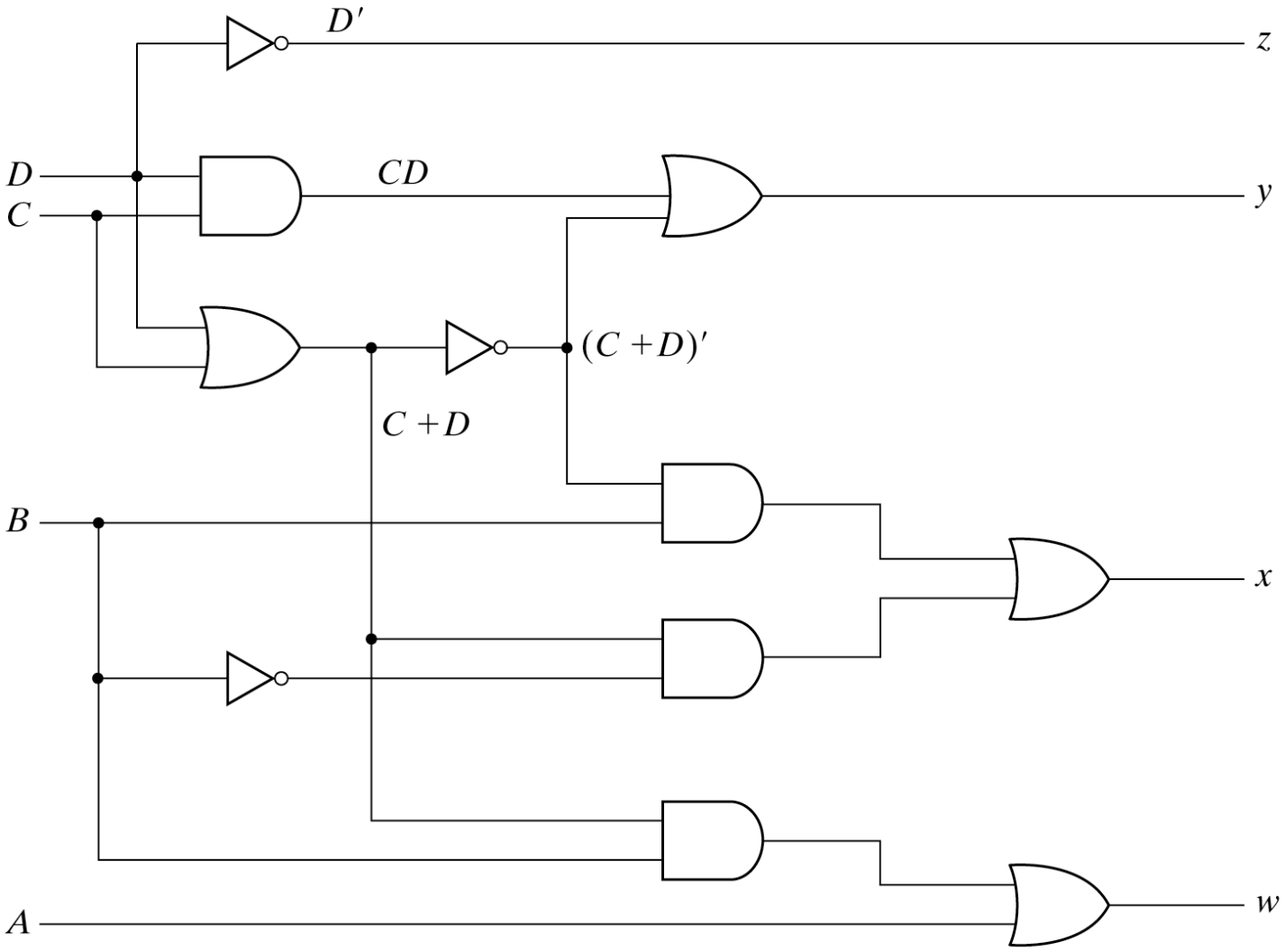


$X = B'C + B'D + BC'D'$



$w = A + BC + BD$

CODE CONVERSION



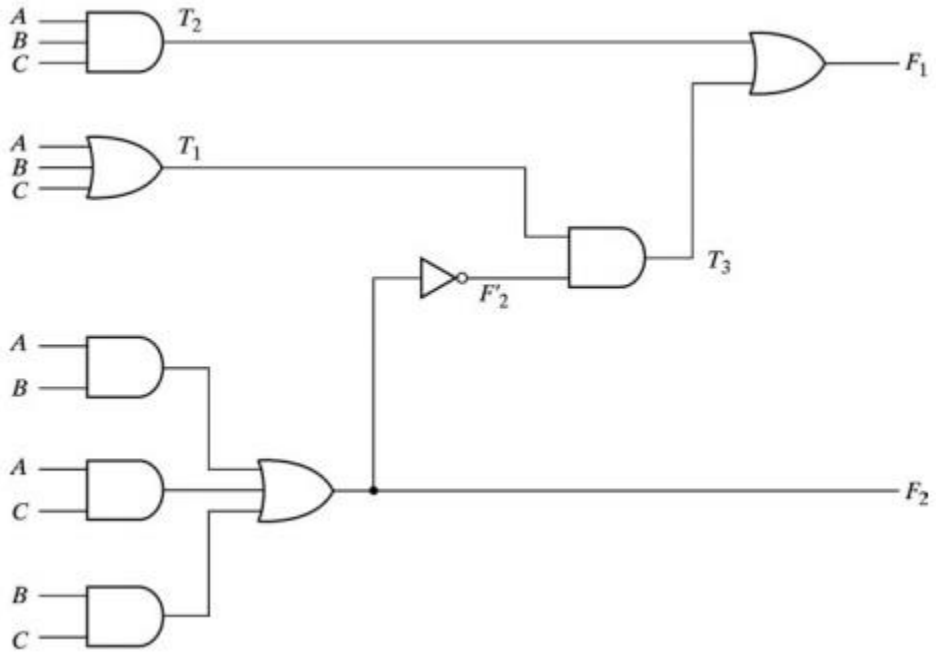
ANALYSIS PROCEDURE

- Analysis:

  determine the function that the circuit implements.
- Often start with a given logic diagram.
- The analysis can be performed by:
  1. Manually finding Boolean functions
  2. Manually finding truth table
  3. Using a computer simulation program
- First step:
  - Make sure that circuit is combinational
  - Without feedback paths or memory elements
- Second step:
  - Obtain the output Boolean functions or the truth table

ANALYSIS PROCEDURE(CONTD.)

- Step 1:
  - Label all gate outputs that are a function of input variables
  - Determine Boolean functions for each gate output

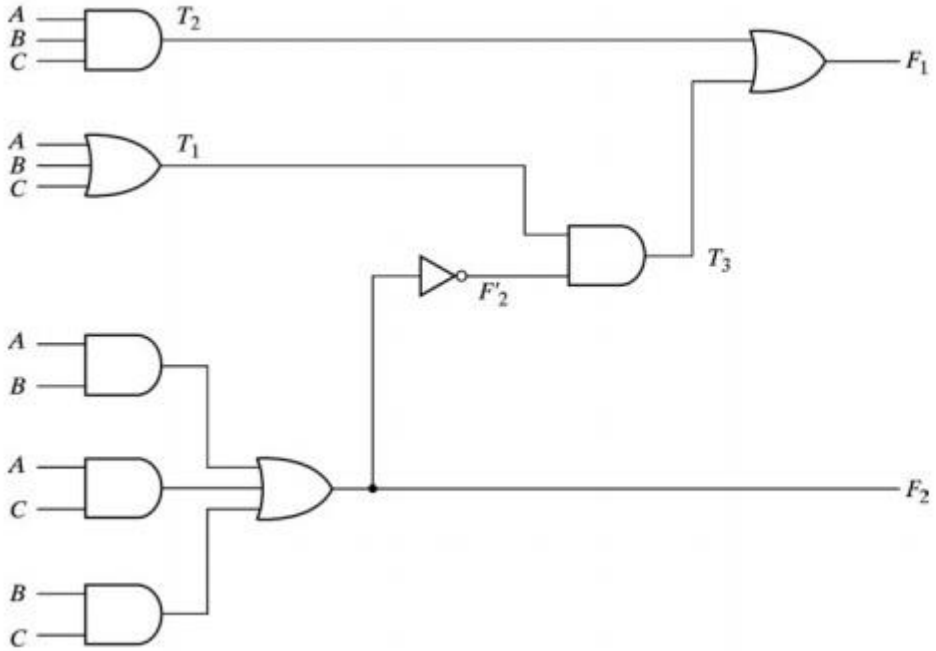


$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Figure 4.1

ANALYSIS PROCEDURE(CONTD.)

- Step 2:
  - Label the gates that are a function of input variables and previously labeled gates
  - Find the Boolean function for these gates



$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

ANALYSIS PROCEDURE(CONTD.)

- Step 3:
  - Obtain the output Boolean function in term of input variables
    - By repeated substitution of previously defined functions

$$
\begin{aligned}
F_1 &= T_3 + T_2 = F'_2\, T_1 + ABC \\
&= (AB + AC + BC)'\,(A + B + C) + ABC \\
&= (A' + B')(A' + C')(B' + C')\,(A + B + C) + ABC \\
&= (A' + B'\,C')(AB' + AC' + BC' + B'\,C) + ABC \\
&= A'\,BC' + A'\,B'\,C + AB'\,C' + ABC
\end{aligned}
$$

ANALYSIS PROCEDURE(CONTD.)

- To obtain the truth table from the logic diagram:

  1. Determine the number of input variables For n inputs:
     - 2^n possible combinations
     - List the binary numbers from 0 to 2^n-1 in a table
  2. Label the outputs of selected gates
  3. Obtain the truth table for the outputs of those gates that are a function of the input variables only
  4. Obtain the truth table for those gates that are a function of previously defined variables at step 3
     - Repeatedly until all outputs are determined

ANALYSIS PROCEDURE(CONTD.)

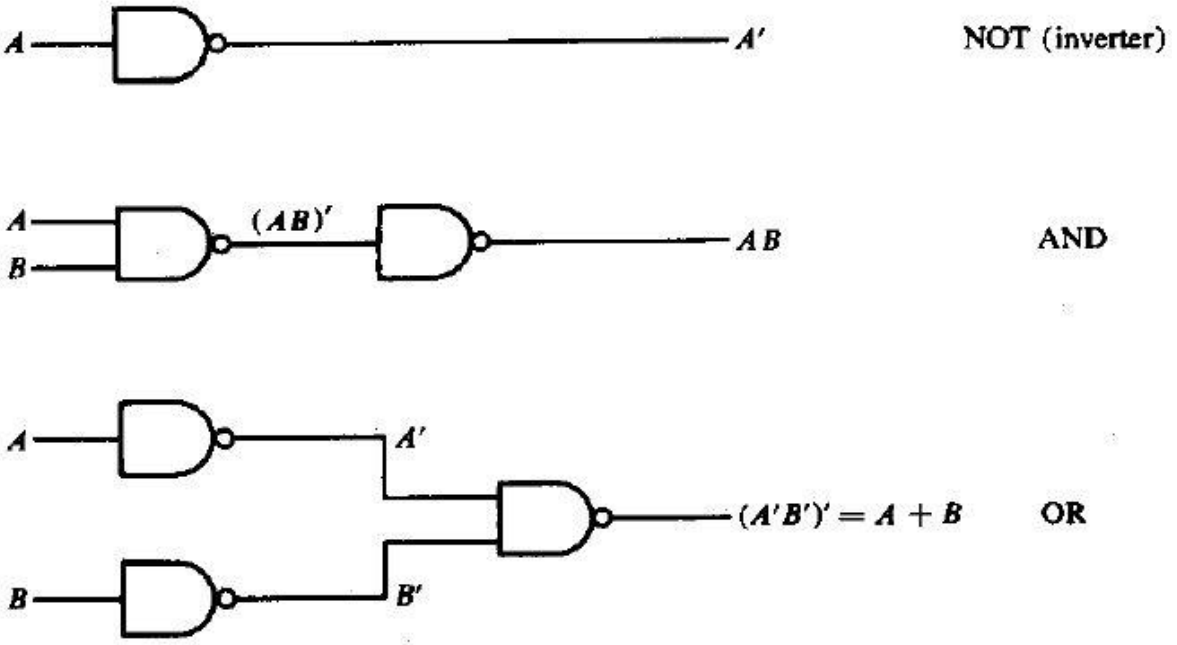
- Truth Table for Figure 4.1:

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

MULTI LEVEL NAND AND NOR CIRCUITS

- The NAND gate is said to be a universal gate because any digital system can be implemented with it.

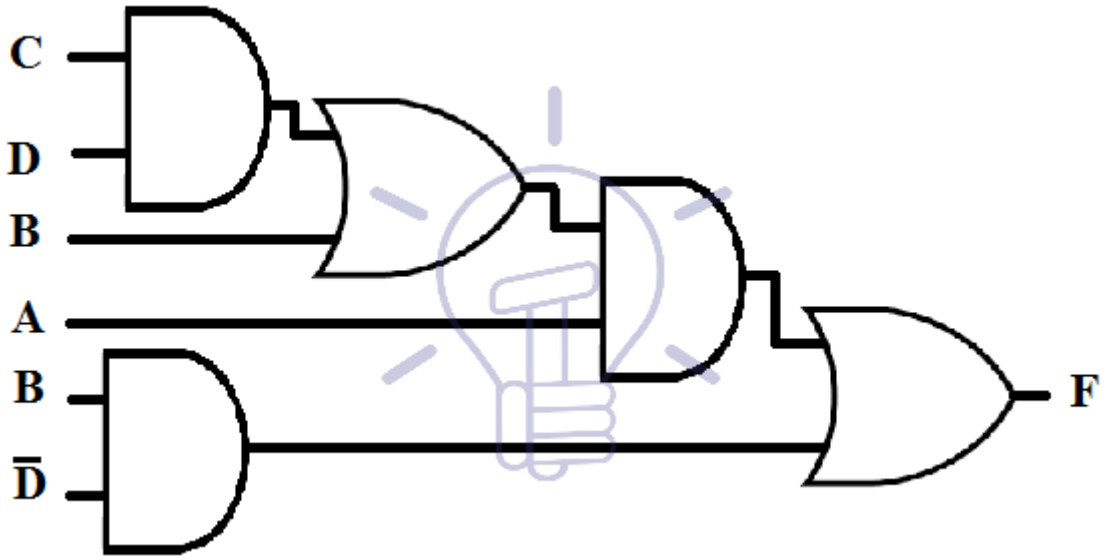
- The implementation of the AND,OR and NOT operations with NAND gate is shown in Fig. below.



MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a multi-level function be;

$$F = A (B + CD) + BD'$$

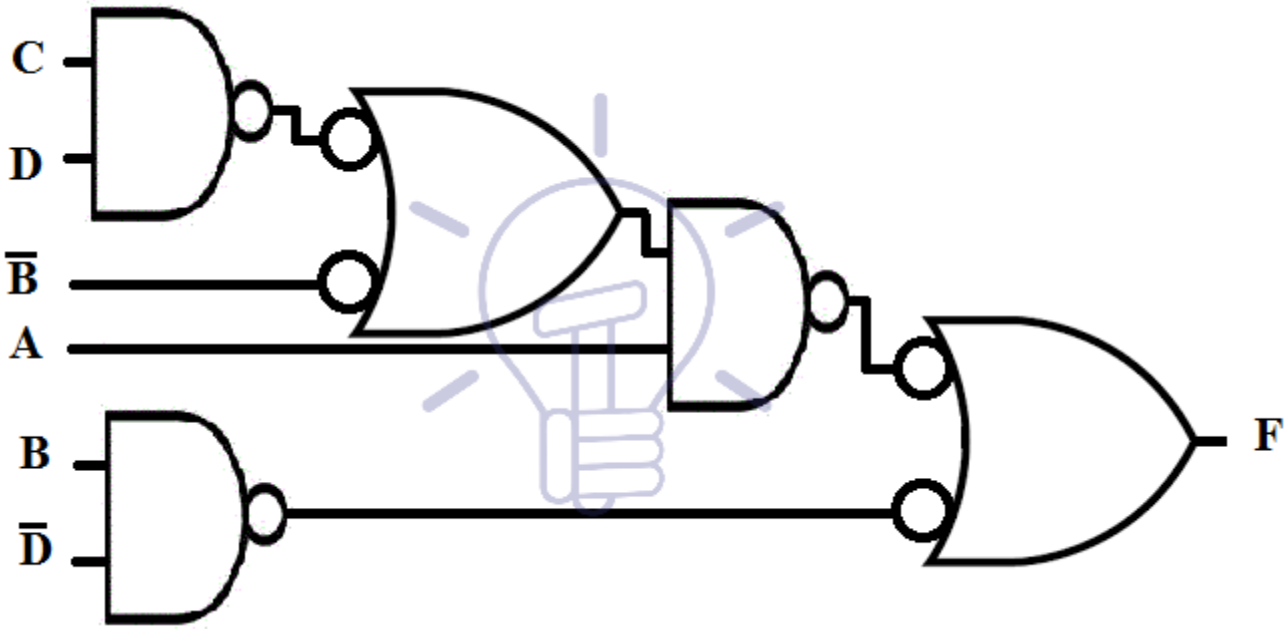


AND-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a multi-level function be;

$$F = A (B + CD) + BD'$$

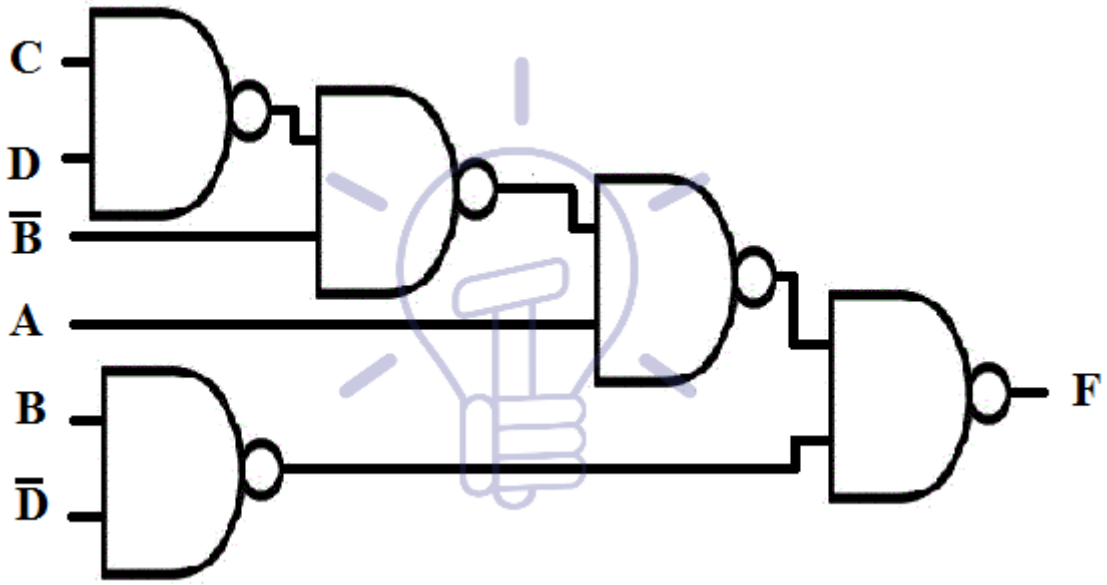


AND-INVERT and INVERT-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a multi-level function be;

$$F = A (B + CD) + BD'$$

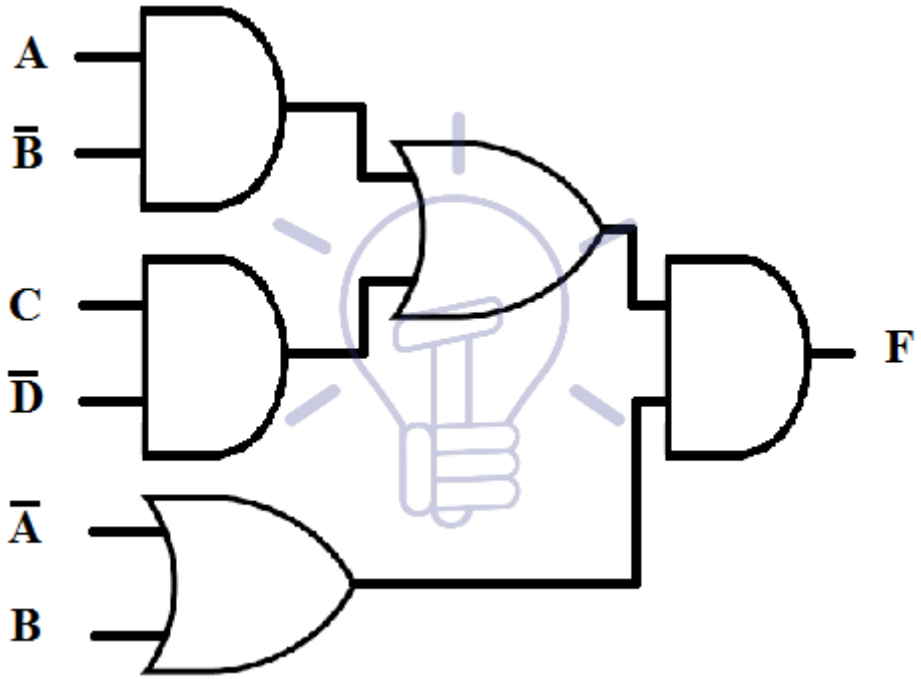


NAND Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose 3-level function be

F = (AB'+CD') (A'+B)

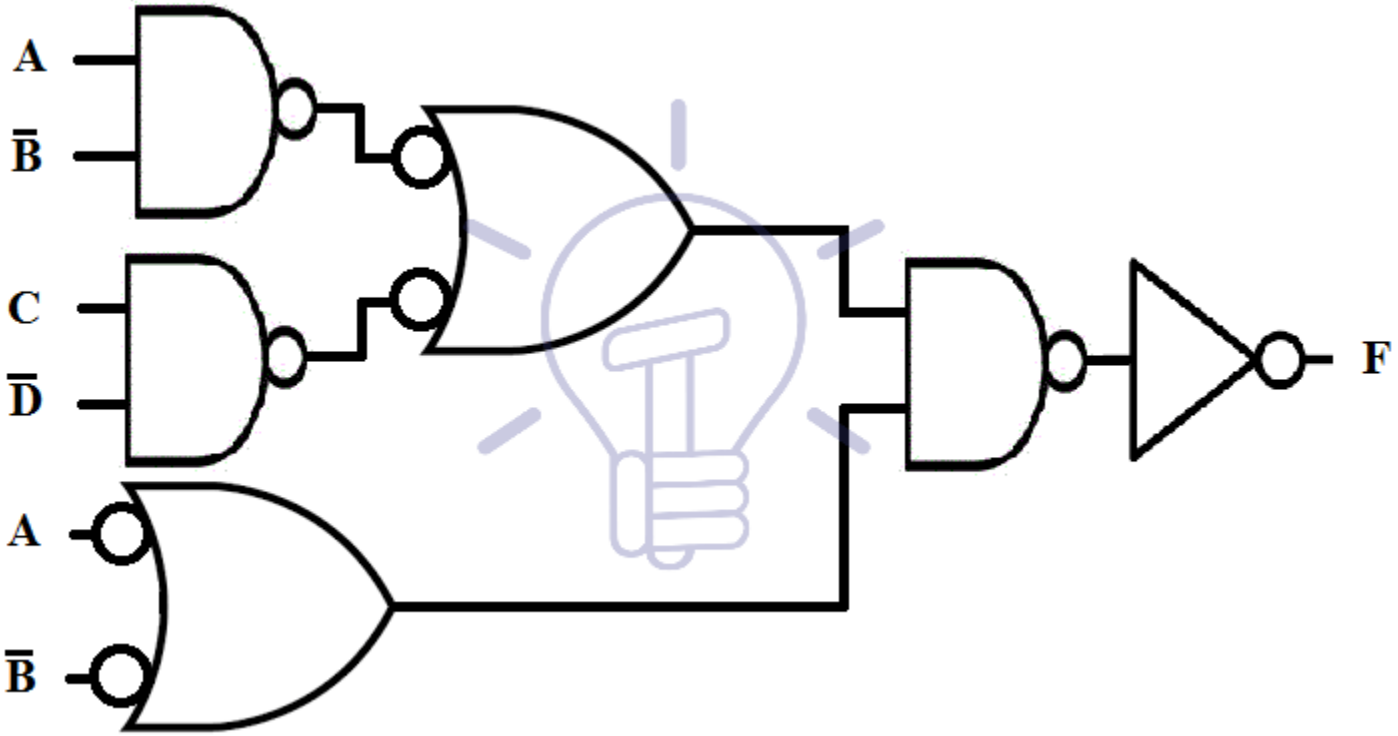


AND-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose 3-level function be

F = (AB'+CD') (A'+B)

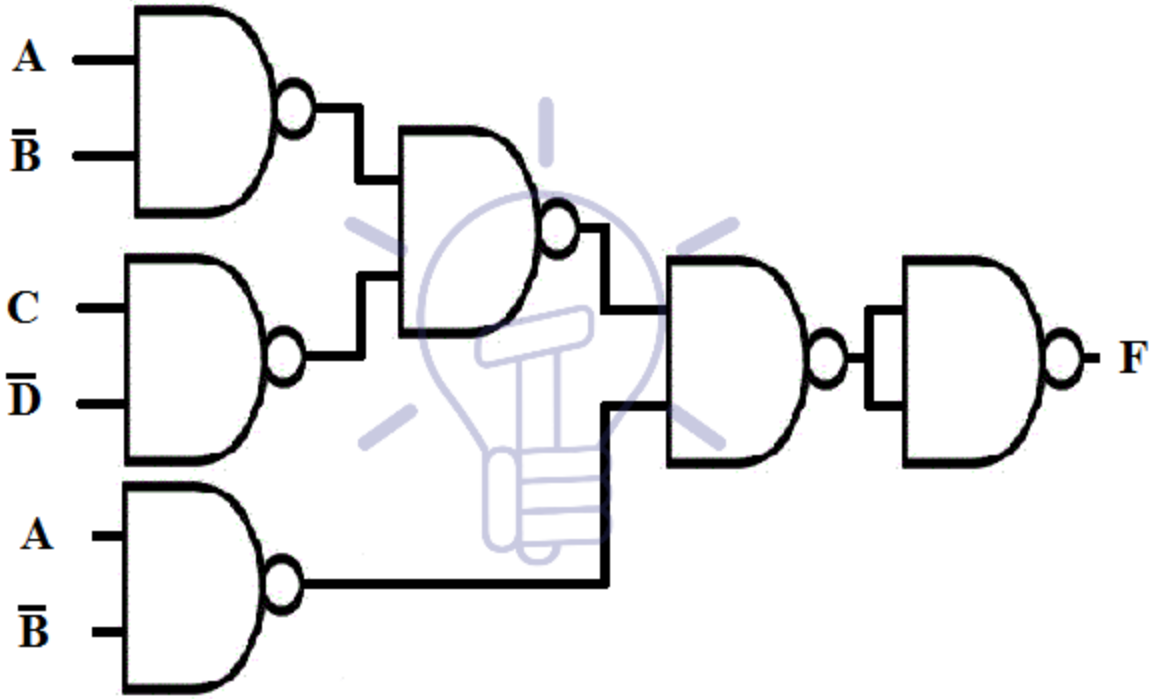


AND-INVERT and INVERT-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a multi-level function be;

F = (AB'+CD') (A'+B)



NAND Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- The NOR gate is said to be a universal gate because any digital system can be implemented with it.

- The implementation of the AND,OR and NOT operations with NOR gate is shown in Fig. below.

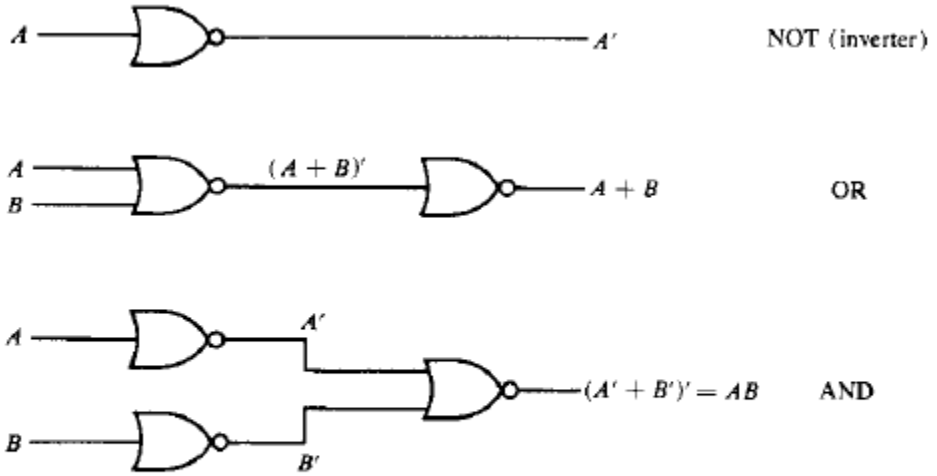
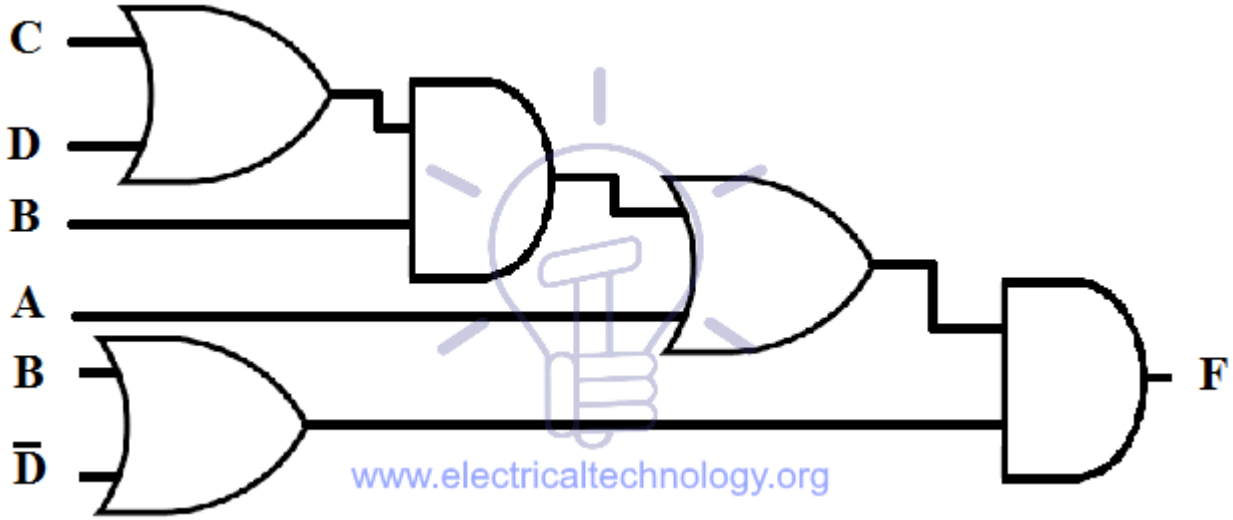


FIGURE 4-17
Implementation of NOT, OR, and AND by NOR gates

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (A + B (C + D)) (B + D')$$

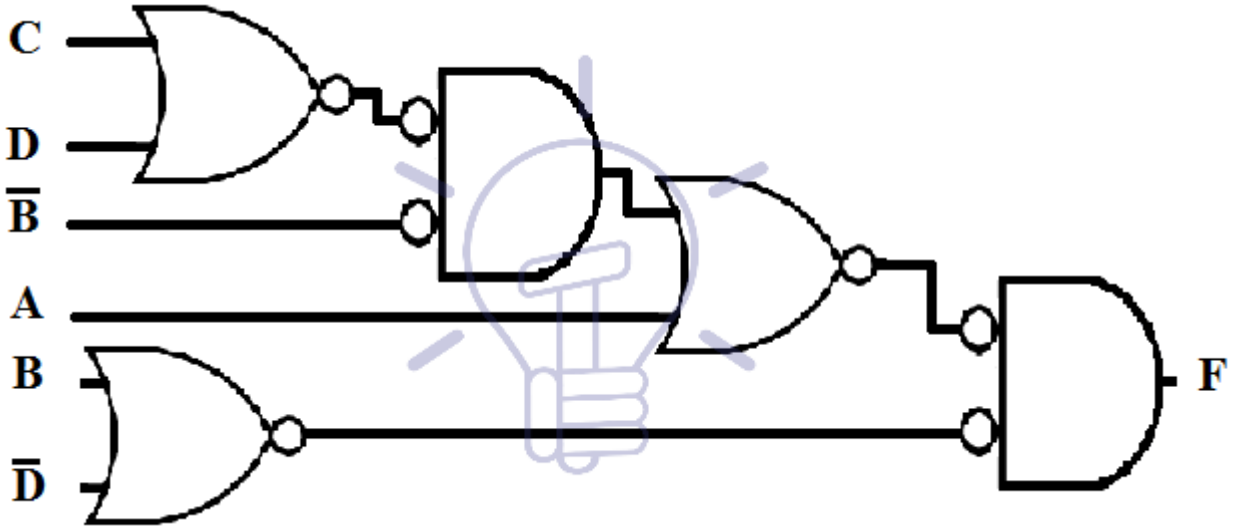


AND-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (A + B (C + D)) (B + D')$$

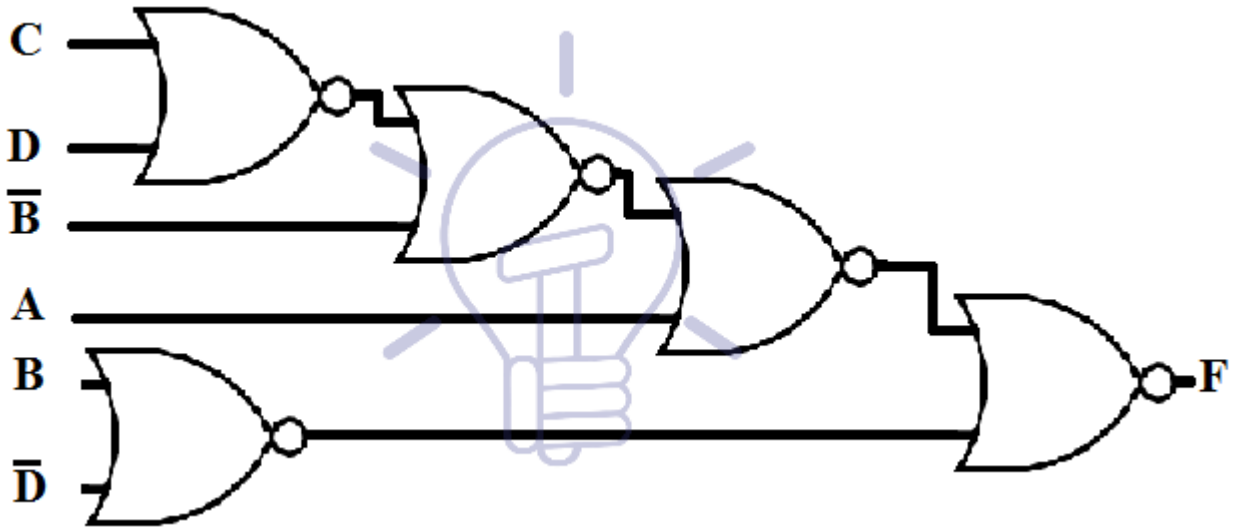


OR Invert and Invert AND Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (A + B (C + D)) (B + D')$$

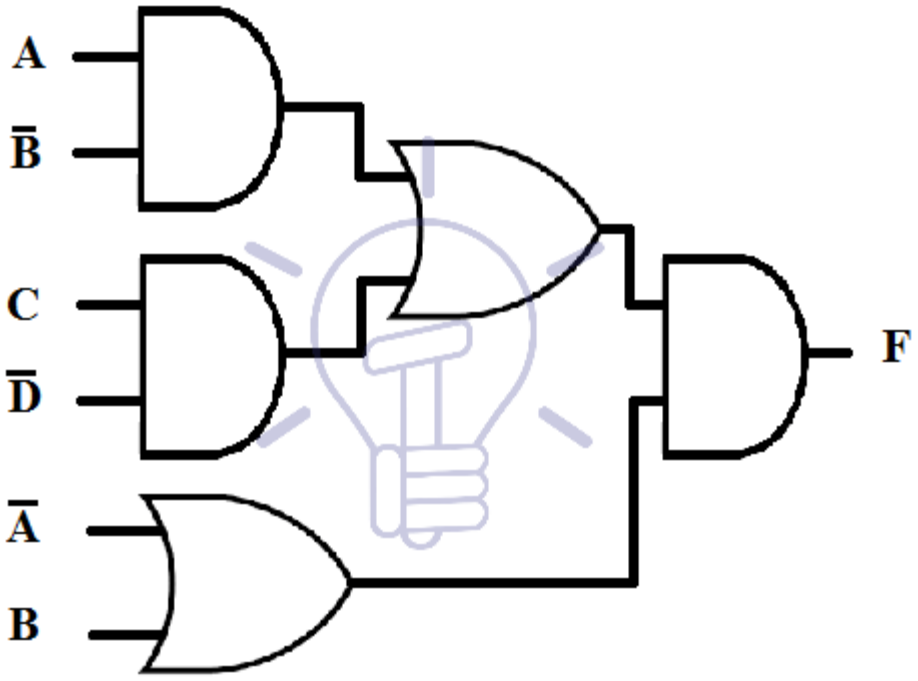


OR Invert and Invert AND Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (AB' + CD')(A' + B)$$

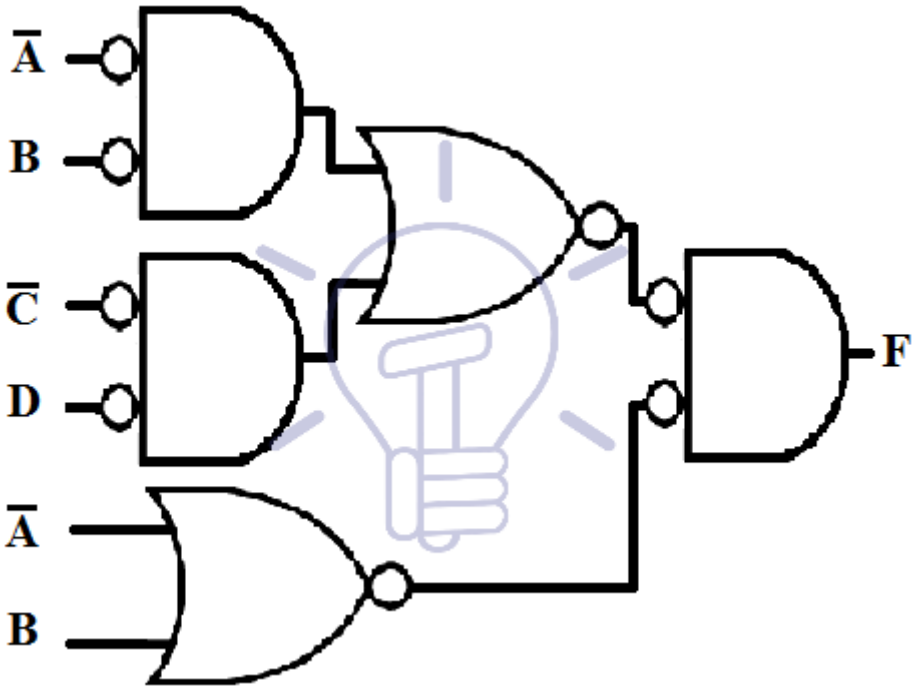


AND-OR Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (AB' + CD')(A' + B)$$

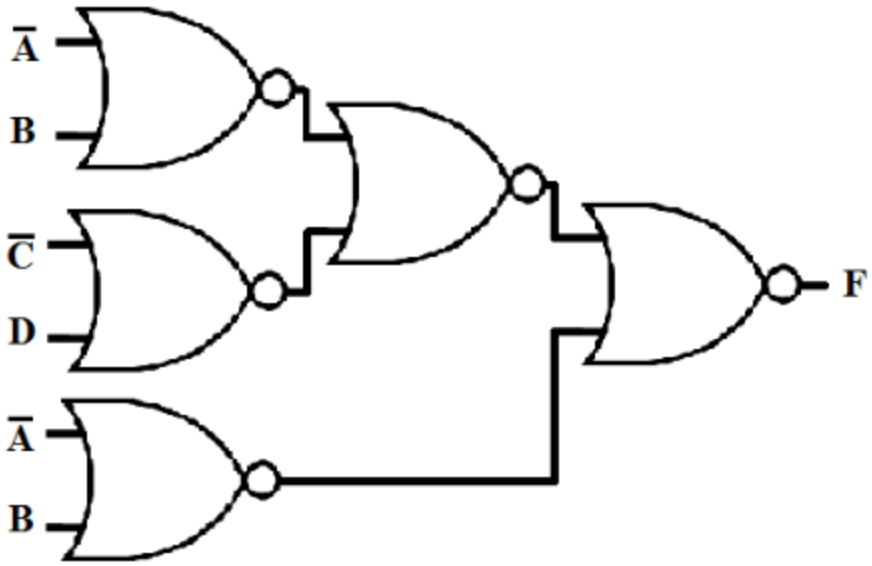


OR Invert and Invert AND Schematic

MULTI LEVEL NAND AND NOR CIRCUITS

- Suppose a 4-level function:

$$F = (AB' + CD') (A' + B)$$



OR Invert and Invert AND Schematic

Ex-OR and Equivalence Functions

- Exclusive-OR and equivalence, denoted by $\oplus$ and $\odot$, respectively, are binary operations that perform the following Boolean functions:
  - $x \oplus y = x'y + xy'$
  - $x \odot y = x'y' + xy$

- The two operations are complement of each other. Each is commutative and associative.