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

What is Combinational Logic

Adders
- Half-Adder
- Full-Adder

Subtractors
- Half-Subtractor
- Full-Subtractor

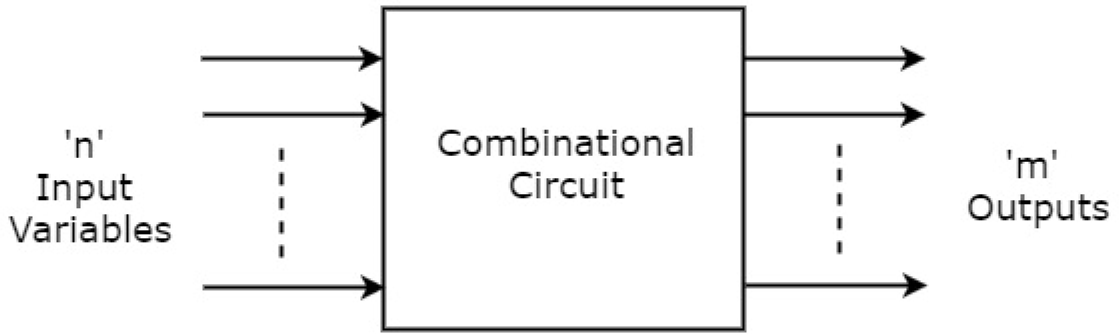
Code Conversion

Multilevel NAND and NOR Circuits

The Universal Property of NAND and NOR gates

Combinational Logic

- Logic circuits for the digital systems maybe combinational or sequential.
- Combinational circuits consist of Logic gates. These circuits operate with binary values.
- The outputs of combinational circuit depend on the combination of present inputs. The following figure shows the block diagram of combinational circuit.



Combinational Logic Vs. Sequential Logic

Combinational Logic	Sequential Logic
In Combinational Logic, output is only dependent of the present input. Example: Adder, Subtractor, Multiplexer, Comparator.	But in Sequential Logic, output depends on the present input as well as the previous output/outputs. Example: Counter, Register.
In this example 1+0=1, the output is not depending on any previous output. It's only depending on the present inputs. So, Adder is a combinational Circuit.	In this example 3+1=4 -> 4+1=5, the present output is depending on the previous output. The counter is incrementing the previous output. So, Counter is a Sequential Circuit.
In Combinational Logic, there is no memory required to store previous outputs.	In Sequential Logic, there is a memory element required to store previous outputs, which is called flip-flops.
There is no positive feedback in Combinational Circuit.	There is positive feedback in Sequential Circuit.

Design Procedure of Combinational Circuit

- Problem Definition
- The no of available input variables and required output variables are determined.
- The I/P and O/P variables are assigned letter symbols.
- The truth table that defines the required relationships between inputs and outputs is derived.
- The simplified Boolean function for each output is obtained.
- The logic diagram is drawn

N.B: The specifications may indicate that some input combinations will not occur. This combinations become don’t care conditions.

Design Procedure of Combinational Circuit

- Design a combinational circuit that accepts a 3-bit binary number whose output is equal to 1 if there are two consecutive 1’s in the input. The output is 0 otherwise.

- Solve this using the design procedure of combinational circuit.

Adders

- Digital computers perform a variety of information processing tasks which includes various arithmetic operations.
- The most basic arithmetic operation, is the addition of two binary digits.
- We know, 0+0=00 ; 0+1=01 ; 1+0=01 ; 1+1=10.
- In the last operation, the binary sum consists of two digits. The higher significant bit of the result is called a carry.
- Two types:
  - Half-adder: A combinational circuit that performs the addition of two bits is a half-adder.
  - Full-adder: A combinational circuit that performs the addition of three bits is a full-adder.

Half-Adder

- From the verbal explanation of half adder, we understand that it needs two binary inputs and two binary outputs.
- Assign symbols x and y to two inputs and S (Sum) and C (Carry) to two outputs.
- Now formulate a truth table to address the relation between the inputs and the outputs. The truth table is shown below:

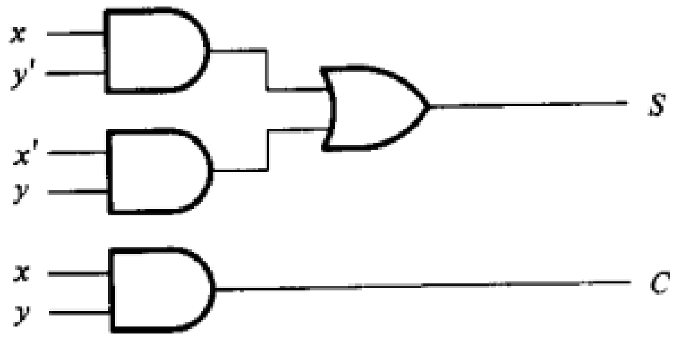
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- The simplified Boolean functions for the two outputs are:

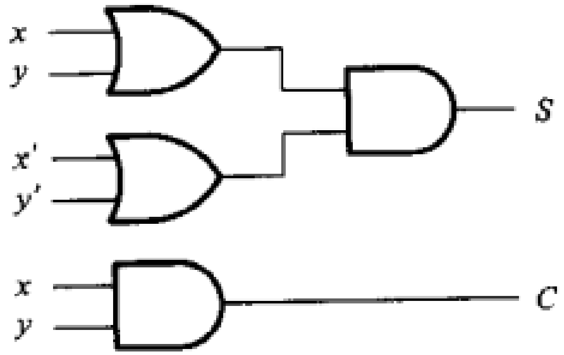
$$S = x'y + xy' \quad D = x'y + xy'$$

$$C = xy \quad B = x'y$$

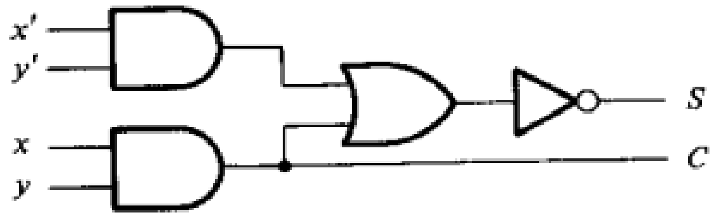
Half-Adder



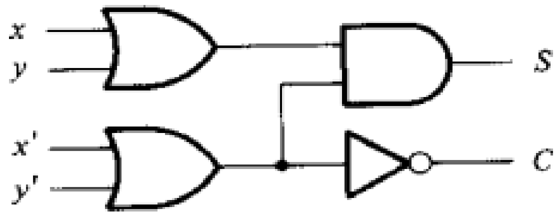
(a) $S = xy' + x'y$
$C = xy$



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

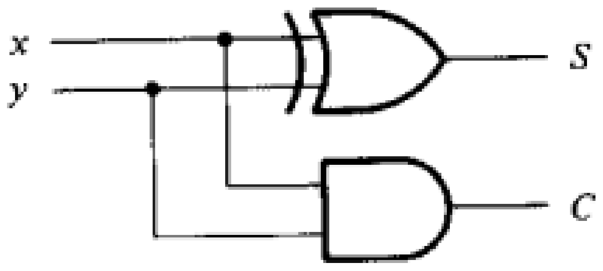


(c) $S = (C + x'y')'$
$C = xy$



(d) $S = (x + y)(x' + y')$
$C = (x' + y')'$

Half-Adder



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

Half-Adder

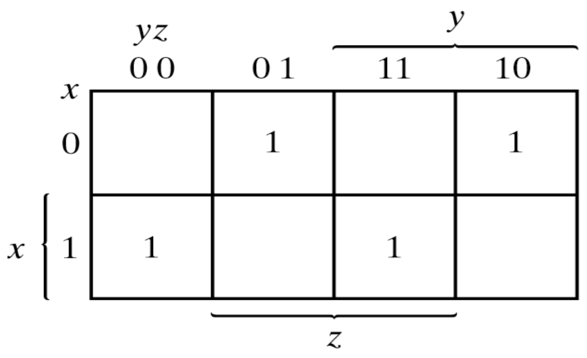
- Figure a is the implementation of the half-adder in SOP.
- Figure b is the implementation of the half-adder in POS.
- In figure c, if we take the minterms of 0’s for S function, so we get
  - S’ = x’y’ + xy => S = (x’y’ + xy)’
  - But C = xy, hence S = (x’y’ + C)’
- In figure d, we are taking the maxterms of 1’s for C function, so we get,
  - S = (x + y) . (x’ + y’)
  - C = (x’+y’)’
- In figure e, we implement S using XOR gate.

Full-Adder

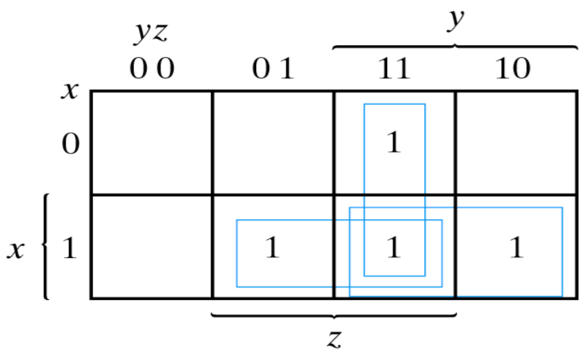
- As this is a combinational circuit that adds three 1 bit binary digits, so there will be 3 input variables and 2 output variables.

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

Full-Adder



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



$$\begin{aligned} S &= xy + xz + yz \\ &= xy + xy'z + x'yz \end{aligned}$$

Fig. 4-6 Maps for Full Adder

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Full-Adder

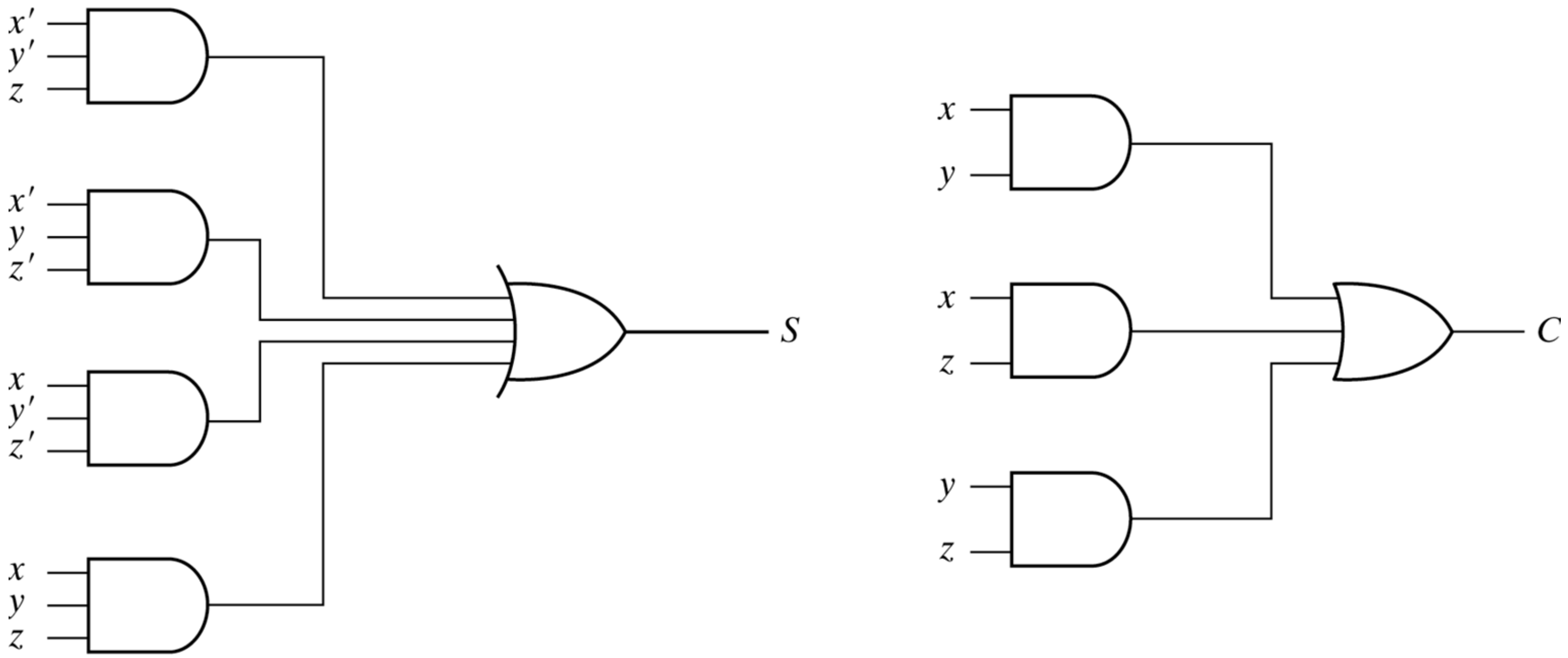


Fig. 4-7 Implementation of Full Adder in Sum of Products

Full-Adder

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

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

$C = xy + yz + xz$

- $= xy+z(x+y)$
- $= xy+z(x+x'y)$ [Apply distributive law, A+A'B=A+B]
- $= xy+xz+x'yz$
- $= x(y+z)+x'yz$
- $= x(y+y'z)+x'yz$ [Apply distributive law, A+A'B=A+B]
- $= xy+xy'z+x'yz$
- $= xy + z(xy'+x'y)$
- $= xy + z(x\oplus y)$

Full-Adder

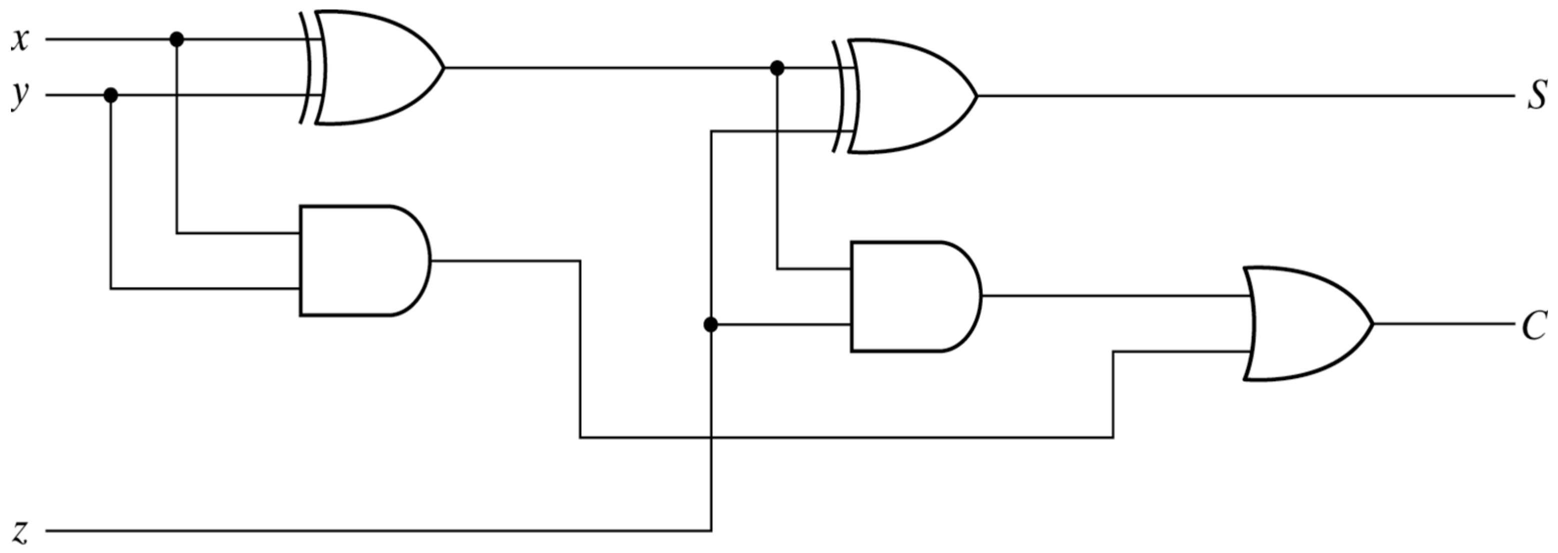


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Half-Subtractor

- A combinational circuit that subtracts two bits and produces their difference.
- It also has an output to specify if a 1 has been borrowed
- Assign symbols x and y to two inputs and D (Difference) and B (Borrow) to two outputs. If x>=y, then no B is needed. If x<y, that means (0-1) operation. Then we borrow a 1 from the next higher stage.
- So, D=2B+x-y if x=0, y=1. then B=1 (as we have to borrow 1 from higher stage. So D = 2+0-1=1
- Now formulate a truth table

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

- The simplified Boolean functions for the two outputs are:

$$D = x'y + xy'$$
$$B = x'y$$

Full-Subtractor

- A full-subtractor is a combinational circuit that performs a subtraction between two bits, assuming that 1 may have been borrowed by a lower significant stage.
- It has 3 inputs and 2 outputs.

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

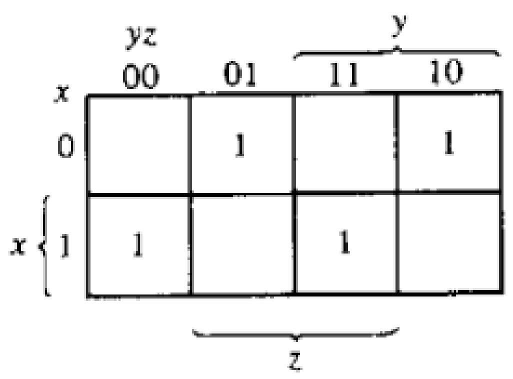
- 1 0 0
- 0 0 1
- 0 1 1

- x-y-z
- x=0, y=0, z=1
- D=0-0-1= 0-1, B=1
- x=1, y=0, z=1
- D=1-0-1=1-1=0, B=0

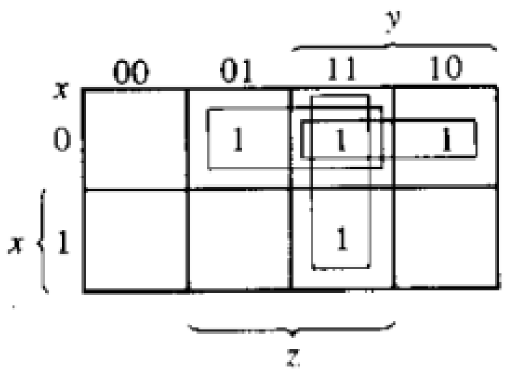
Full-Subtractor

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

$$B = x'y + x'z + yz$$



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



$B = x'y + x'z + yz$

Full-Subtractor

- Q1. Implement a full-subtractor using 2 half-subtractor and an OR gate.
- Q2. Show, how a full adder can be converted to a full subtractor with the addition of one inverter.

D = (x xor y) xor z

B = x'y+x'z+yz

B=x'y+z.(x'+y)

B=x'y+z.1.(x'+y)

B=x'y+z(x'+x)(x'+y)

B=x'y+z(x'+xy) [Distributive law]

B=x'y+x'z+xyz

B=x'(y+z)+xyz

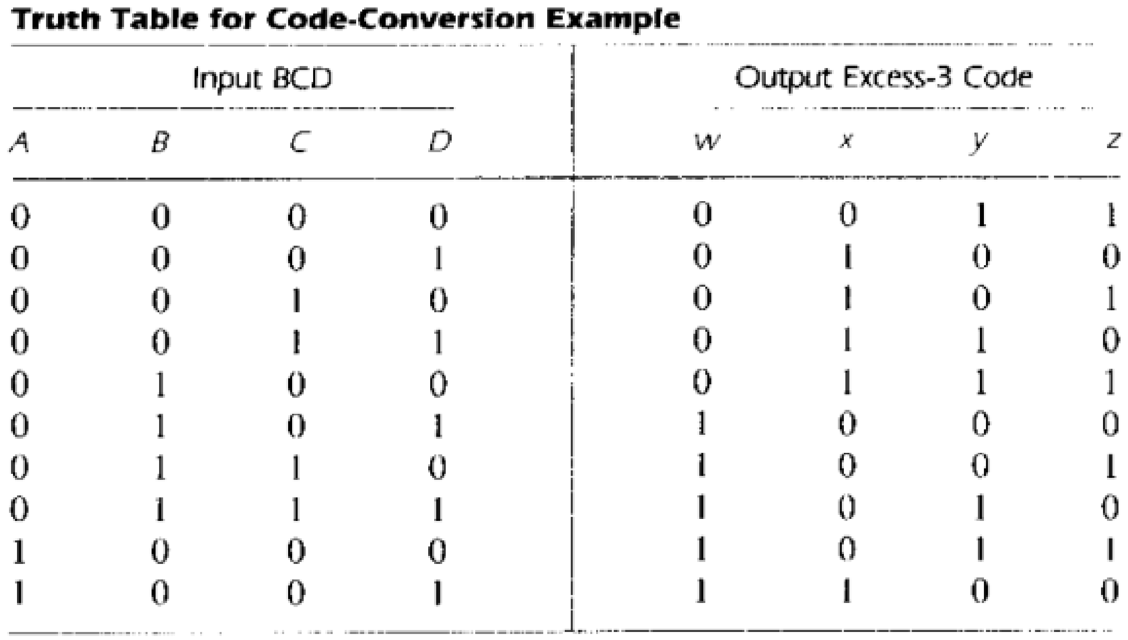
B=x'(y+y'z)+xyz

B=x'y+x'y'z+xyz

B=x'y+z(x'y'+xy) = x'y+z(x xor y)'

Code Conversion

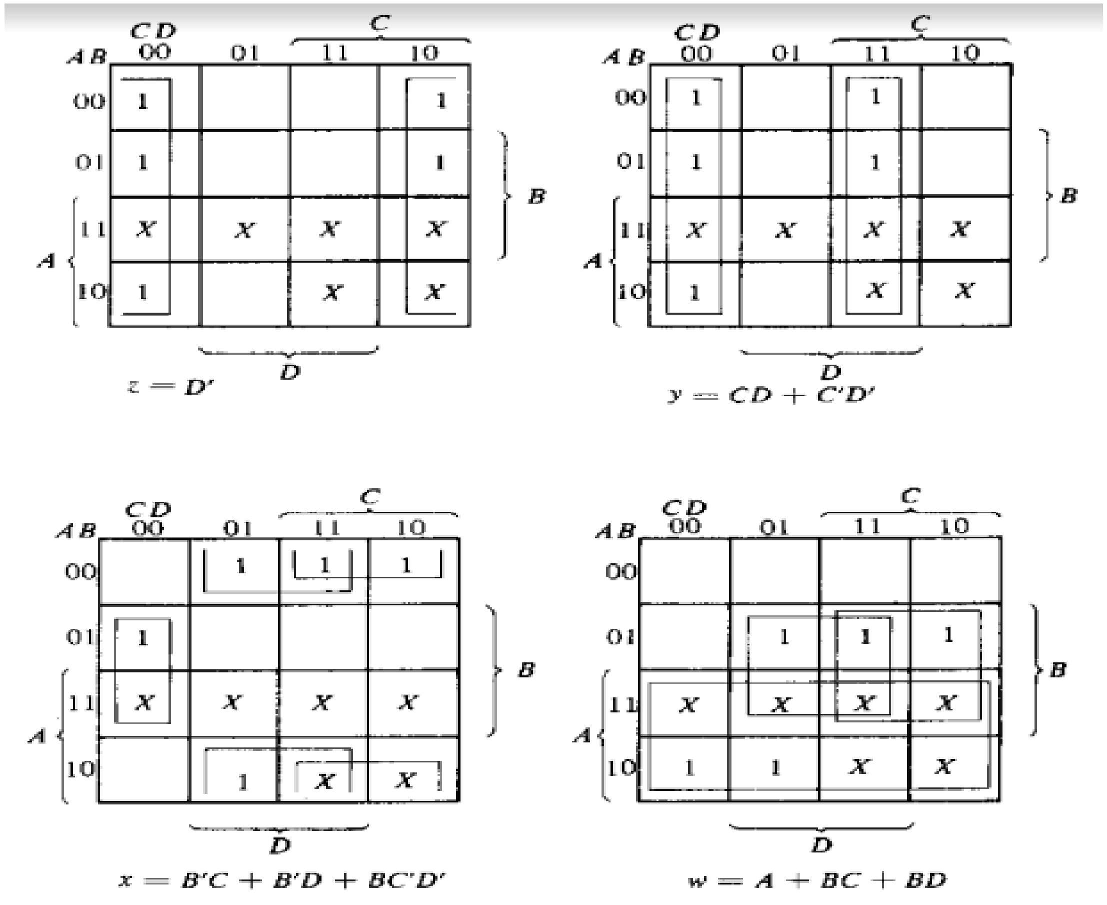
- A conversion circuit must be inserted between the two systems if each uses different codes for the same information.
- Suppose, one system uses BCD and another system is using Excess-3 code. So, to communicate between these two system, a conversion circuit is needed.



Truth Table for Code-Conversion Example

Input BCD				Output Excess-3 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

Code Conversion



$z = D'$

$y = CD + C'D'$

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

$w = A + BC + BD$

Code Conversion

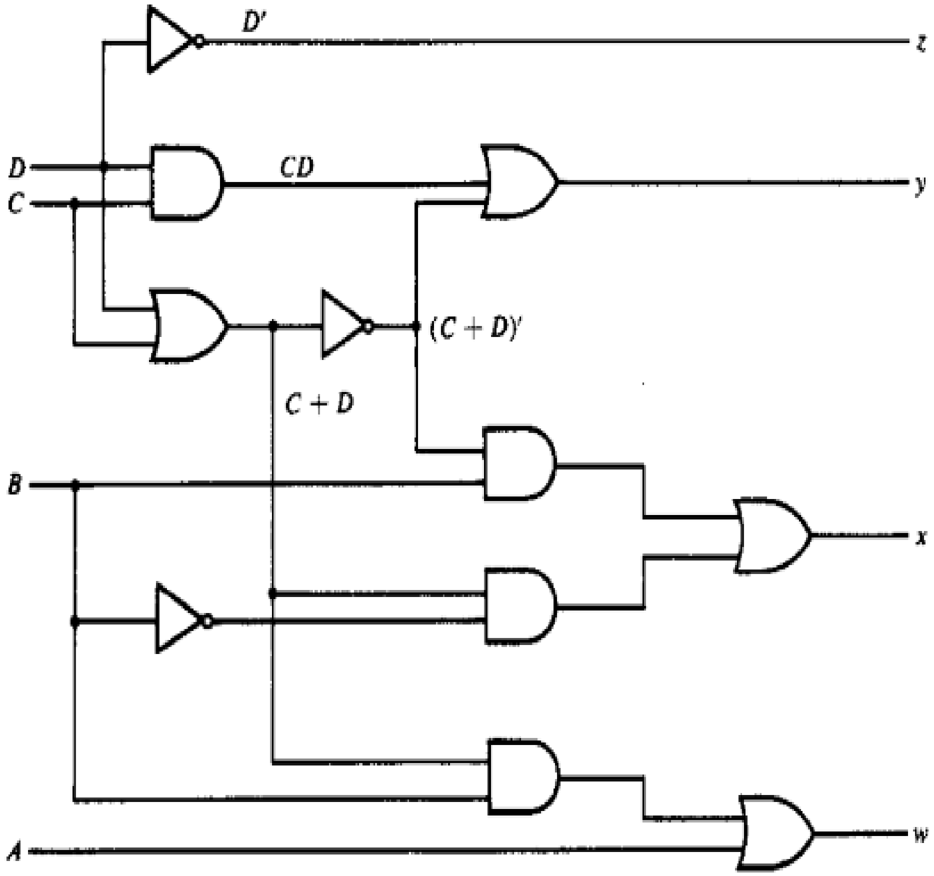
$z = D'$

$y = CD + C'D' = CD + (C + D)'$

$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$

$\quad = B'(C + D) + B(C + D)'$

$w = A + BC + BD = A + B(C + D)$



Analysis Procedure

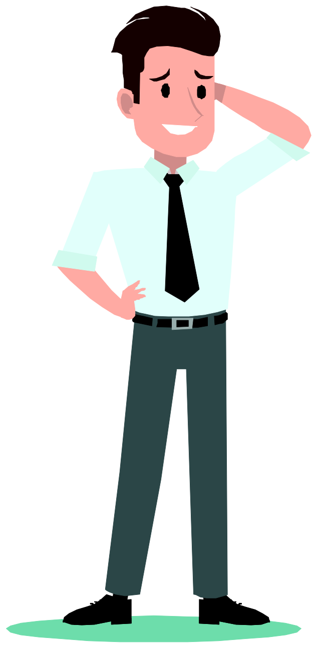
- We know, the design of a combinational circuit starts from the verbal specification of a required function and culminates with a set of Boolean functions or a logic diagram.
- The analysis of a combinational circuit is kind of a reverse process.
- That means, it starts with a logic diagram and culminates with a set of Boolean functions, a truth table or a verbal explanation of the circuit operation.

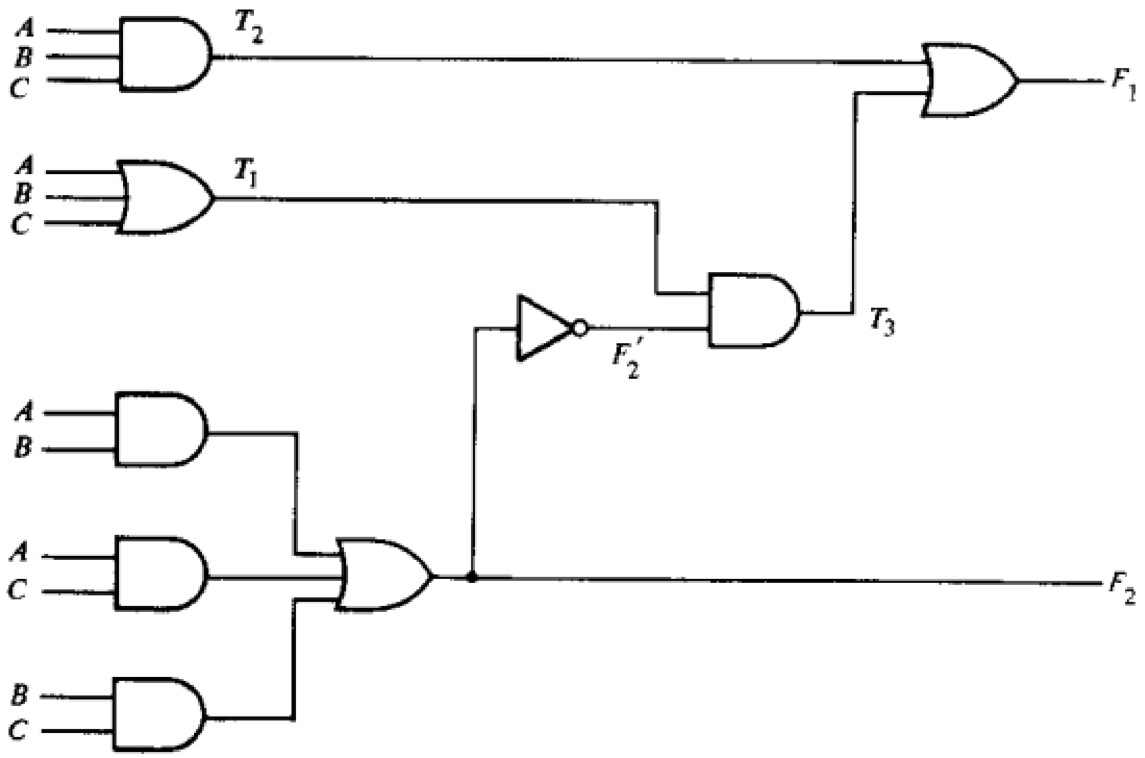
Analysis Procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

- Label with arbitrary symbols all gate outputs that are a function of the input variables. Obtain the Boolean functions for each gate.
- Label with other arbitrary symbols those gates which are a function of the input variables and/or previously labeled gates. Find the Boolean functions for these gates.
- Repeat the process in step 2 until the outputs of the circuit are obtained.
- By repeated substitution of previously defined functions, obtain the output Boolean Functions in terms of input variables only.

Analysis Procedure





Analysis Procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

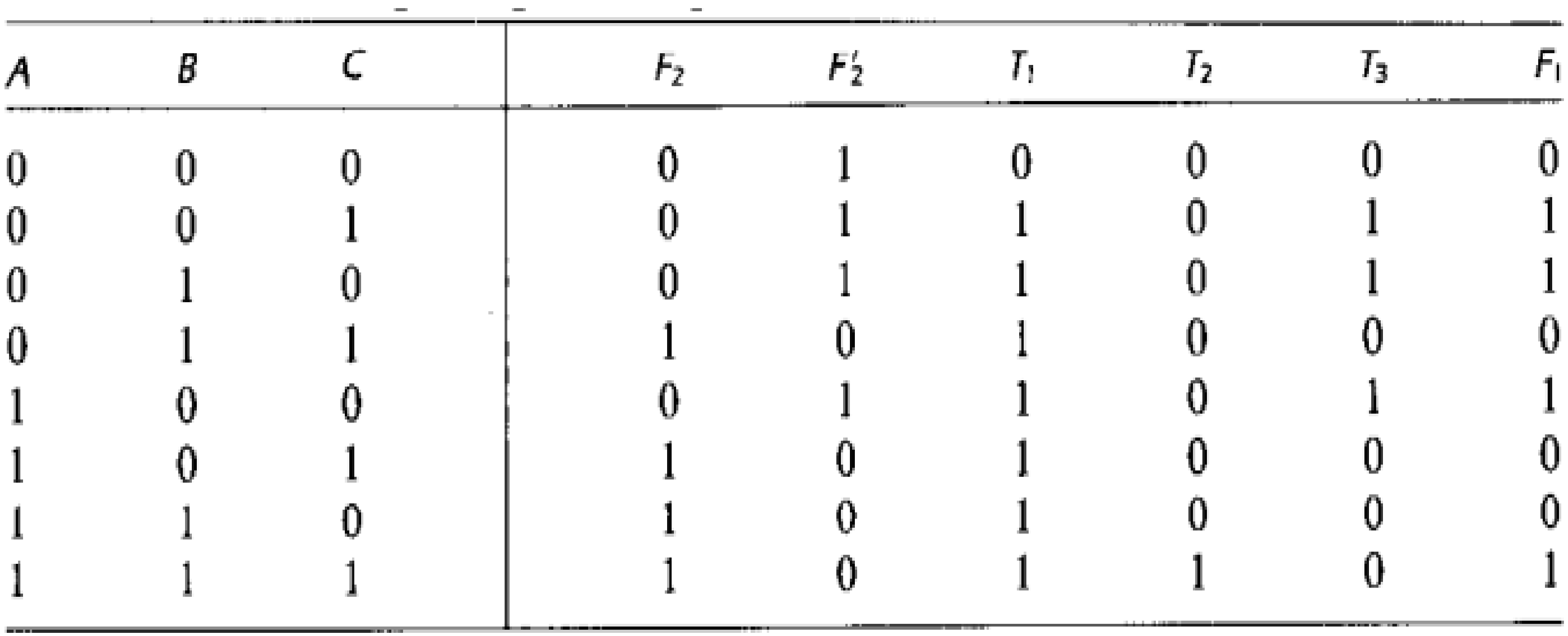
- Step 1: Label with arbitrary symbols all gate outputs that are a function of the input variables. Obtain the Boolean functions for each gate.
- T1 = A + B + C
- T2 = ABC
- T3 = AB
- T4 = AC
- T5 = BC
- Step 2: Label with other arbitrary symbols those gates which are a function of the input variables and/or previously labeled gates. Find the Boolean functions for these gates.
- F2 = T3 + T4 + T5
- T3 = F2'T1
- F1 = T3 + T2
- By repeated substitution of previously defined functions, obtain the output Boolean Functions in terms of input variables only.

Analysis Procedure

- By repeated substitution of previously defined functions, obtain the output Boolean Functions in terms of input variables only.

- F1 = T3 + T2 = F2’T1 + ABC
- F1 = (T3 + T4 + T5)’T1 + ABC
- F1 = (AB + AC + BC)’(A + B + C) + ABC
- F1 = (A’+B’)(A’+C’)(B’+C’)(A + B + C) + ABC
- F1 = (A’ + B’C’)(B’+C’)(A+B+C)+ABC
- F1 = (A’ + B’C’)(AB’ + AC’ + BC’ + B’C) + ABC
- F1 = A’BC’ + A’B’C + AB’C’ + ABC

Analysis Procedure



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

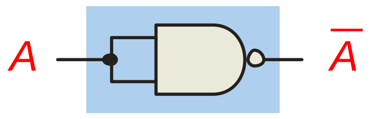
Multilevel NAND NOR Circuits

- Combinational circuits are more frequently constructed with NAND or NOR gates rather than AND and OR gates.
- NAND and NOR gates are more common from the hardware point of view as they are readily available in integrated circuit form.
- You have already seen the Two-level implementation of NAND and NOR logic. Here we will see general cases of multilevel circuits.

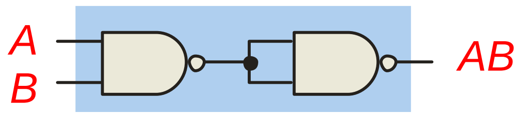
Universal Property of NAND Gate

- The NAND gate is said to be a universal gate because any digital system can be implemented with it.
- Combinational and Sequential circuits both can be constructed with this gate because the flip-flop circuit (memory element) can be constructed from two NAND gates.
- Before implementing any Boolean function with NAND gates, we need to know the implementation of NOT, AND, and OR gates by NAND gates.

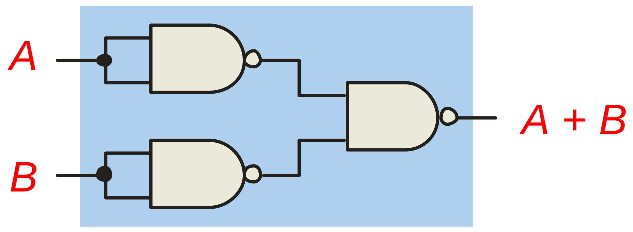
Universal Property of NAND Gate



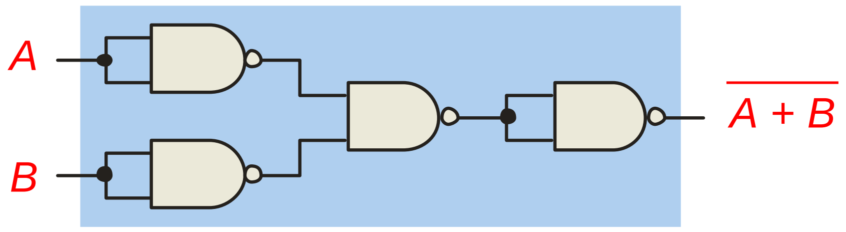
Inverter



AND gate



OR gate



NOR gate

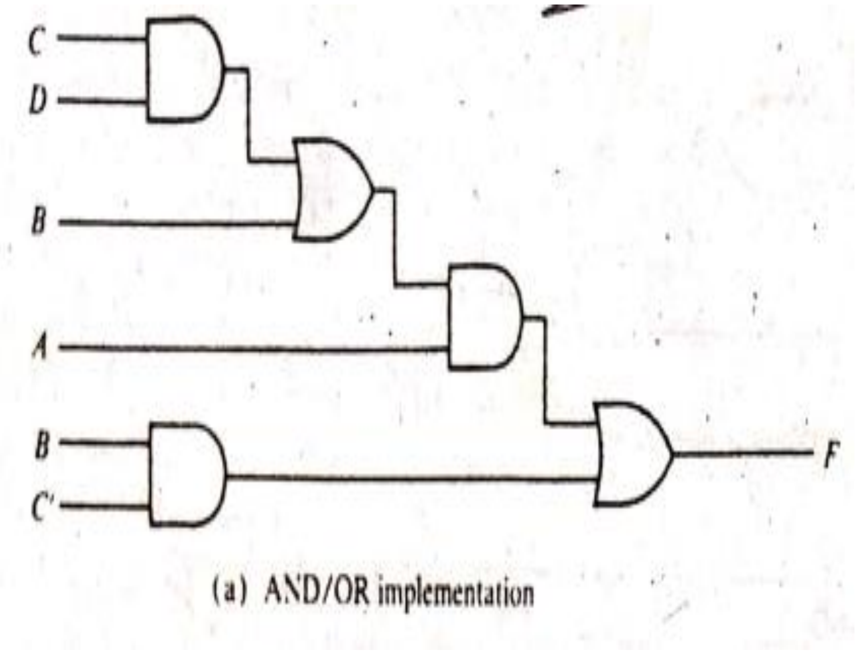
Boolean Function Implementation using NAND Gate

To obtain a multilevel NAND diagram from a Boolean expression, follow these steps:

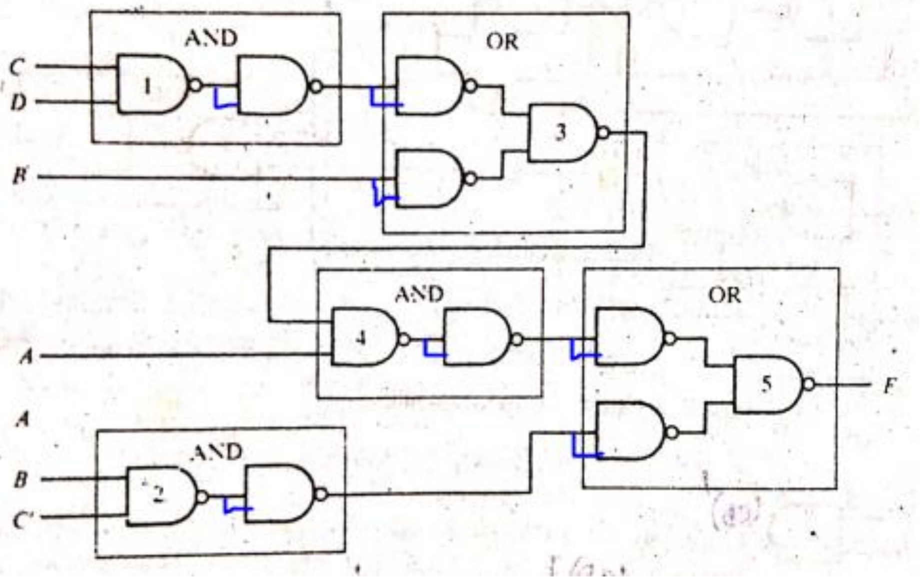
- From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume, both the normal and complement inputs are available.
- Draw a second logic diagram with the equivalent NAND logic, as given in previous slide, substituted for each AND, OR and NOT gate.
- Remove any two cascaded inverters from the diagram, since double inversion doesn't perform a logic function. Remove inverters connected to single external inputs and complement the corresponding input variable.

- Let, F = BC' + A(B + CD)

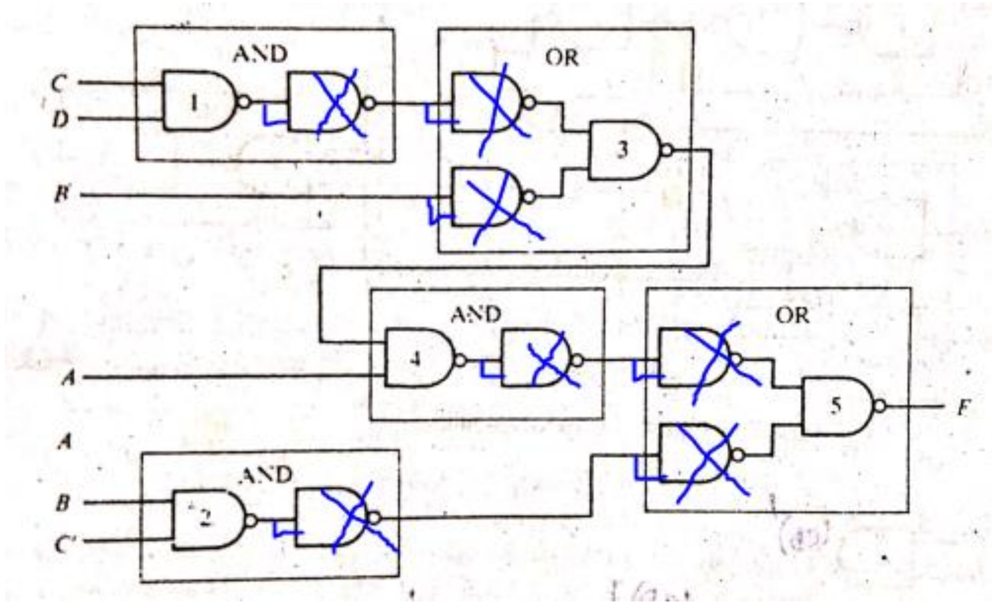
Boolean Function Implementation using NAND Gate

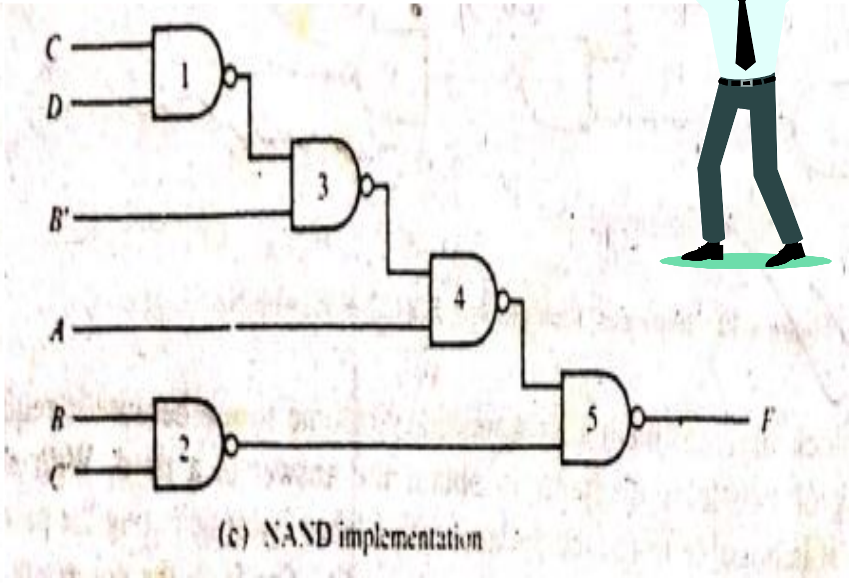


(a) AND/OR implementation



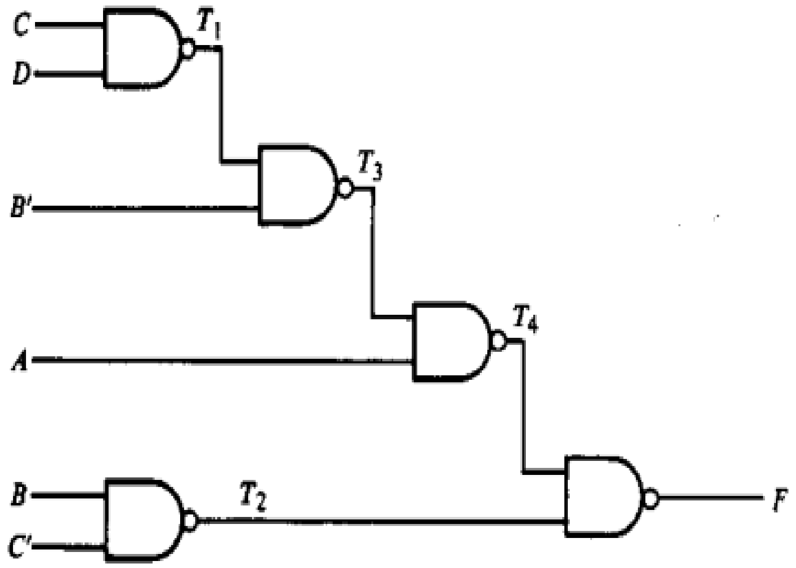
Boolean Function Implementation using NAND Gate





(c) NAND implementation

Analysis procedure of Boolean Function with NAND Gate



$$T_1 = (CD)' = C' + D'$$
$$T_2 = (BC')' = B' + C$$

$$T_3 = (B'T_1)' = (B'C' + B'D')'$$
$$= (B + C)(B + D) = B + CD$$
$$T_4 = (AT_3)' = [A(B + CD)]'$$
$$F = (T_2T_4)' = \{(BC')'[A(B + CD)]'\}'$$
$$= BC' + A(B + CD)$$

Another way to Implement Boolean Function using NAND Gate

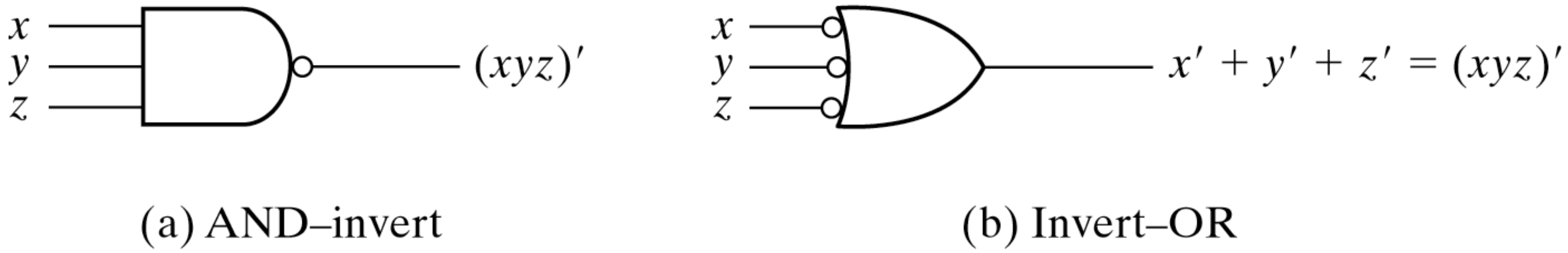


Fig. 3-19 Two Graphic Symbols for NAND Gate

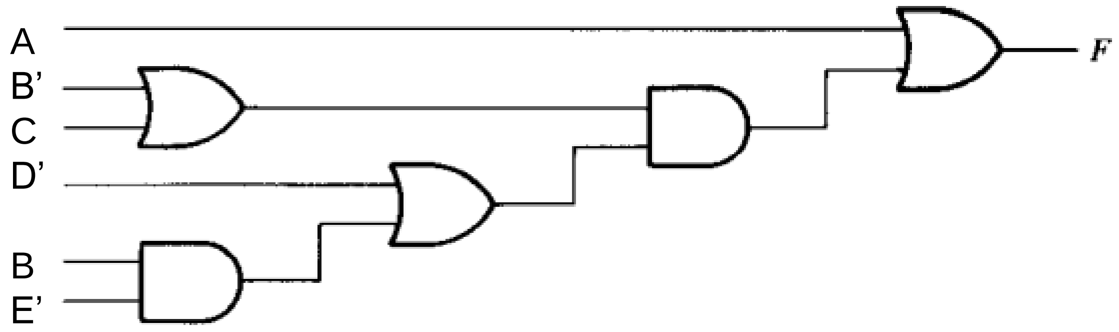
Another way to Implement Boolean Function using NAND Gate

To obtain a multilevel NAND diagram from a Boolean expression, follow these steps:

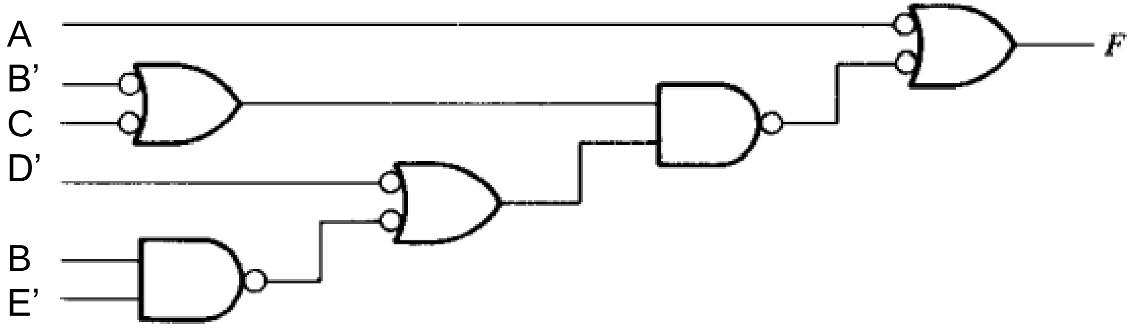
- From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume, both the normal and complement inputs are available.
- Convert all AND gates to NAND gates with AND-invert graphic symbol.
- Convert all OR gates to NAND gates with invert-OR graphic symbol.
- Check all small circles in the diagram. For every small circle that's not compensated by another small circle along the same line, insert an inverter(one input NAND gate) or complement the input variable.

- Let, F = A+(B'+C)(D'+BE')

Another way to Implement Boolean Function using NAND Gate

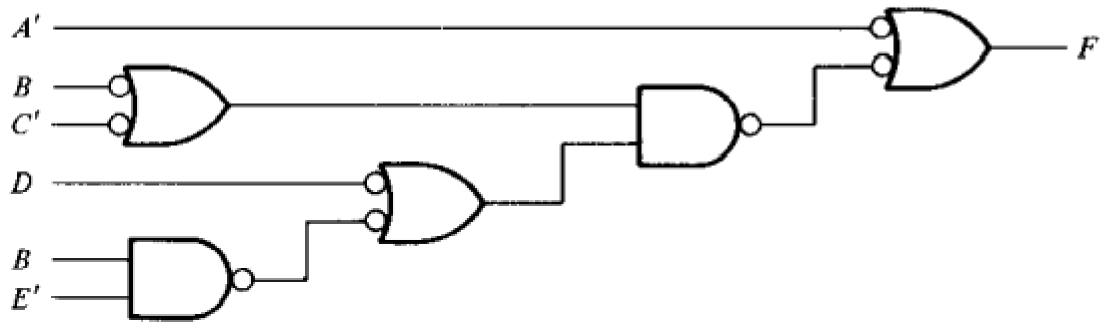


(a) AND-OR diagram

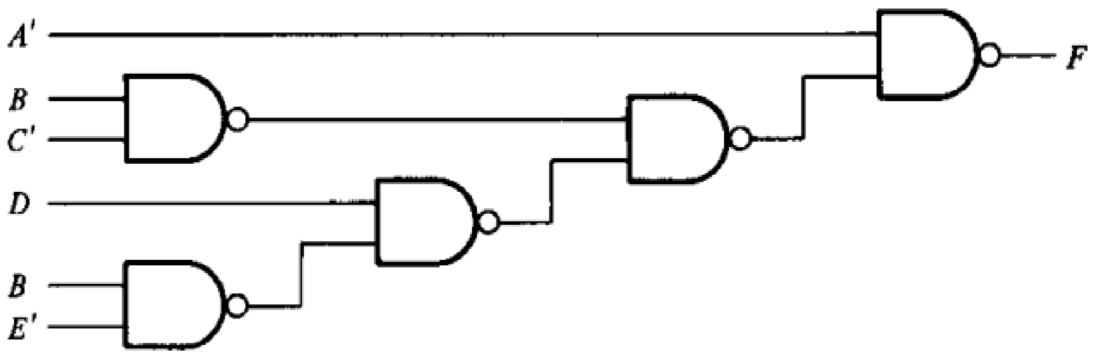


(b) NAND diagram using two graphic symbols

Another way to Implement Boolean Function using NAND Gate

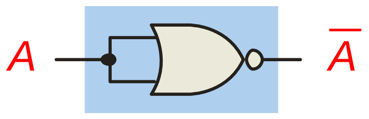


(b) NAND diagram using two graphic symbols

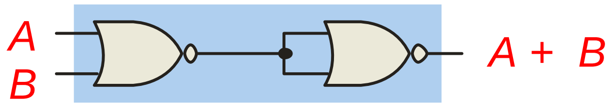


(c) NAND diagram using one graphic symbol

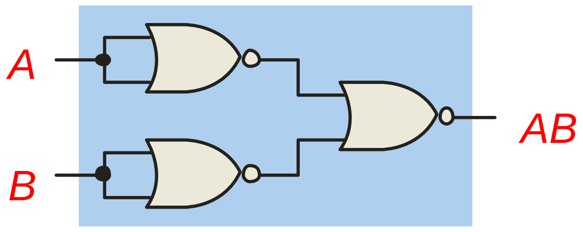
Universal Property of NOR Gate



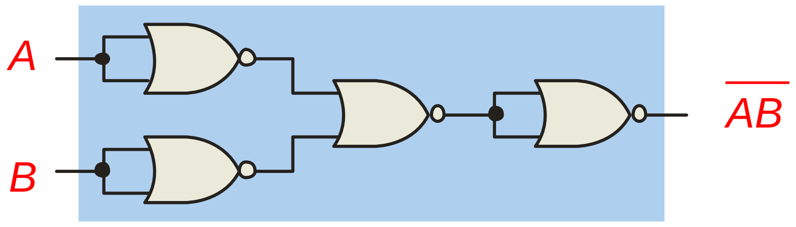
Inverter



OR gate



AND gate



NAND gate

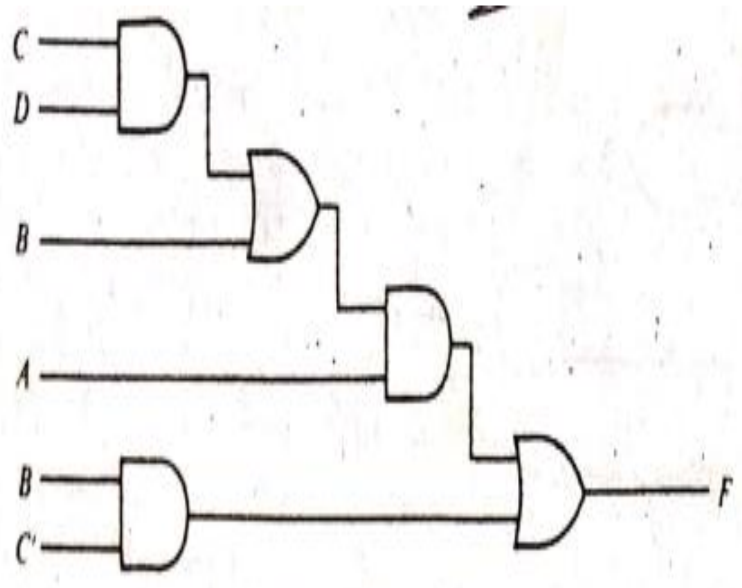
Boolean Function Implementation using NOR Gate

To obtain a multilevel NOR diagram from a Boolean expression, follow these steps:

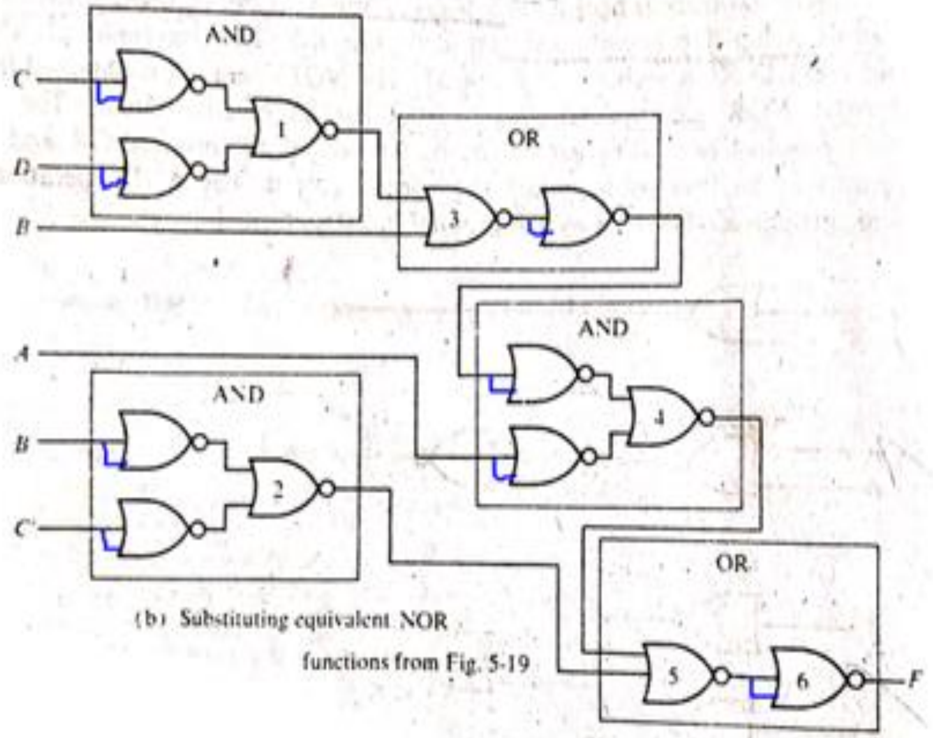
- From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume, both the normal and complement inputs are available.
- Draw a second logic diagram with the equivalent NOR logic, as given in previous slide, substituted for each AND, OR and NOT gate.
- Remove pairs of cascaded inverters from the diagram, since double inversion doesn't perform a logic function. Remove inverters connected to single external inputs and complement the corresponding input variable.

- Let, F = BC' + A(B + CD)

Boolean Function Implementation using NOR Gate

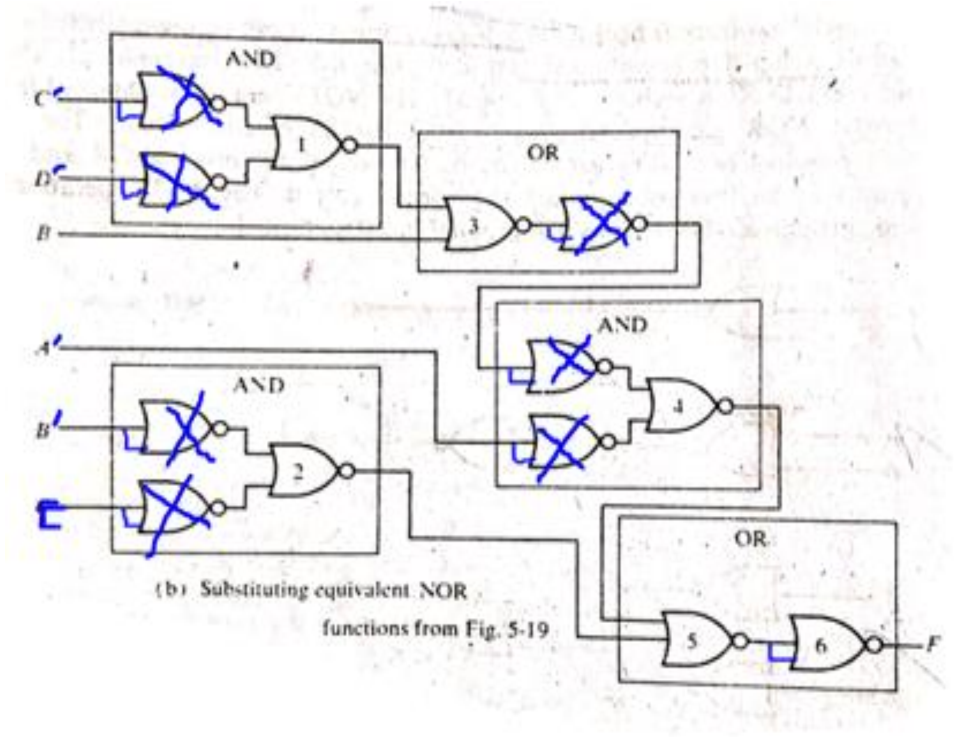


(a) AND/OR implementation

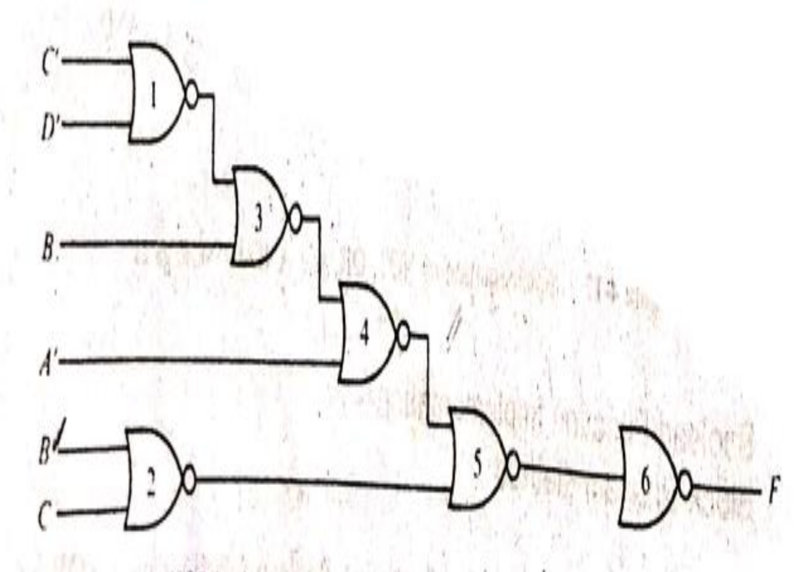


(b) Substituting equivalent NOR functions from Fig. 5-19

Boolean Function Implementation using NOR Gate

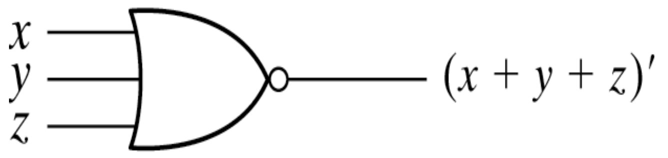


(b) Substituting equivalent NOR functions from Fig. 5-19

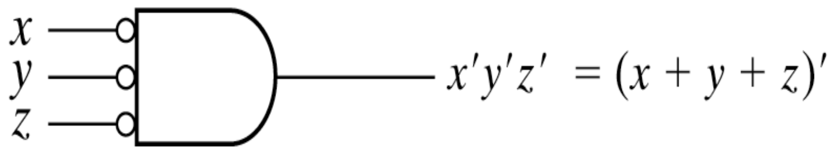


(c) NOR implementation

Another way to Implement Boolean Function using NOR Gate



(a) OR–invert



(a) Invert–AND

Fig. 3-25 Two Graphic Symbols for NOR Gate

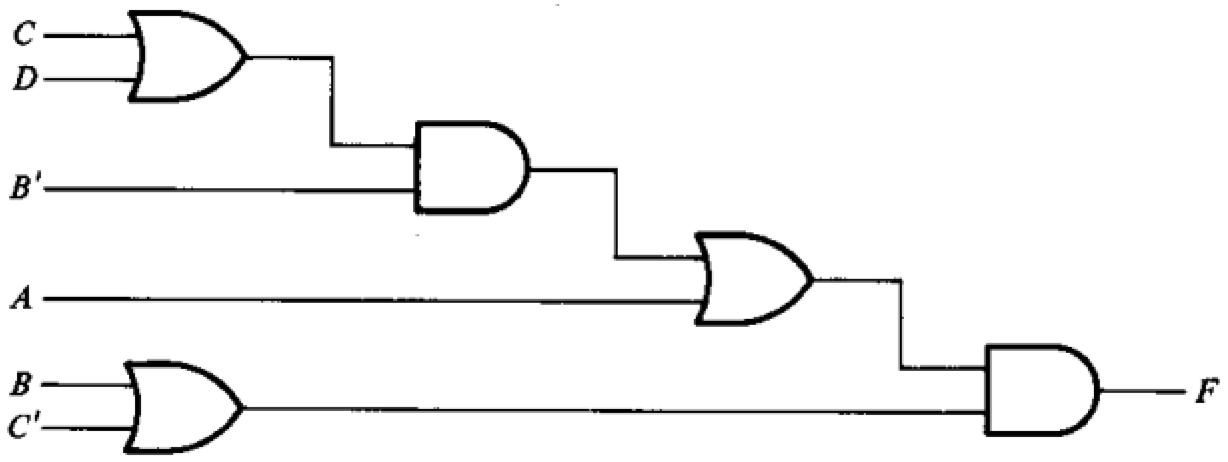
Another way to Implement Boolean Function using NOR Gate

To obtain a multilevel NOR diagram from a Boolean expression, follow these steps:

- From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume, both the normal and complement inputs are available.
- Convert all AND gates to NOR gates with invert-AND graphic symbol.
- Convert all OR gates to NOR gates with OR-invert graphic symbol.
- Check all small circles in the diagram. For every small circle that's not compensated by another small circle along the same line, insert an inverter(one input NOR gate) or complement the input variable.

- Let, F = A+(B'+C)(D'+BE')

Another way to Implement Boolean Function using NAND Gate



(c) AND-OR logic diagram

Parity Generator and Parity Checker

Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

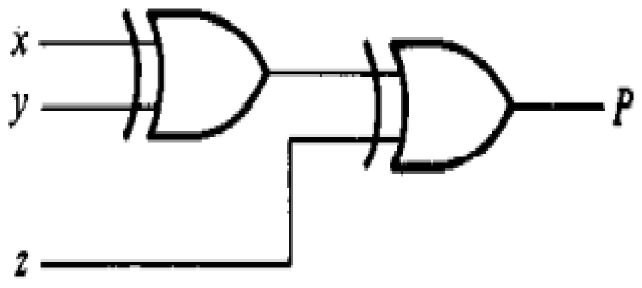
$$P = x \oplus y \oplus z$$

Even-Parity-Checker Truth Table

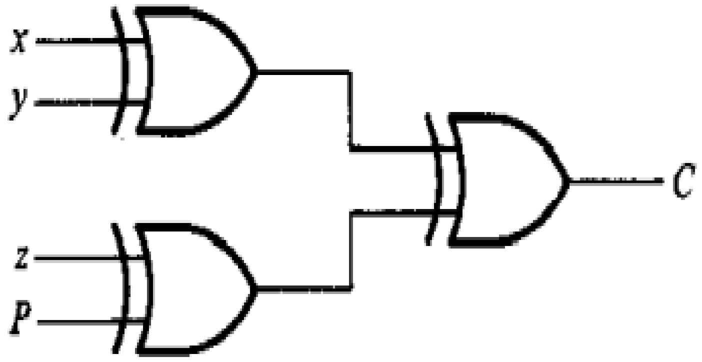
Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$

Parity Generator and Parity Checker



(a) 3-bit even parity generator



(b) 4-bit even parity checker

GOOD NEWS!

THE CLASS IS OVER…

THANK YOU!