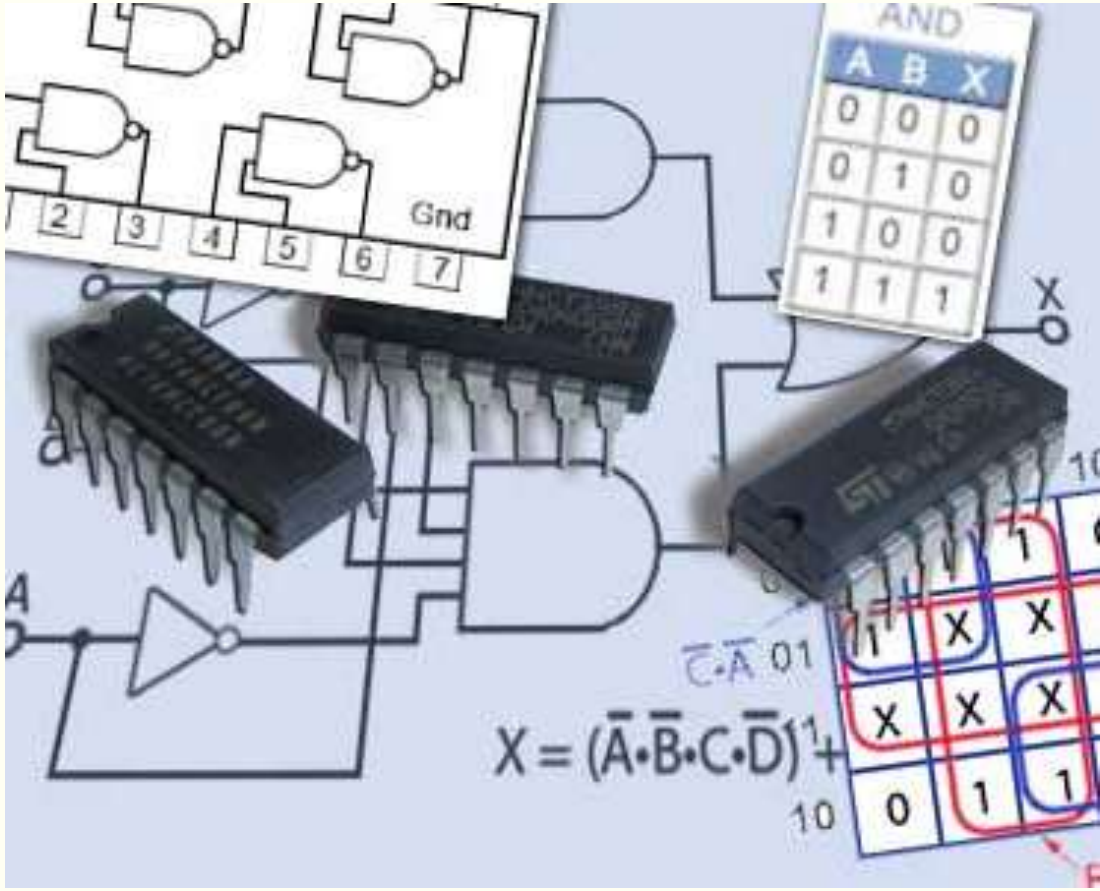
Digital Logic

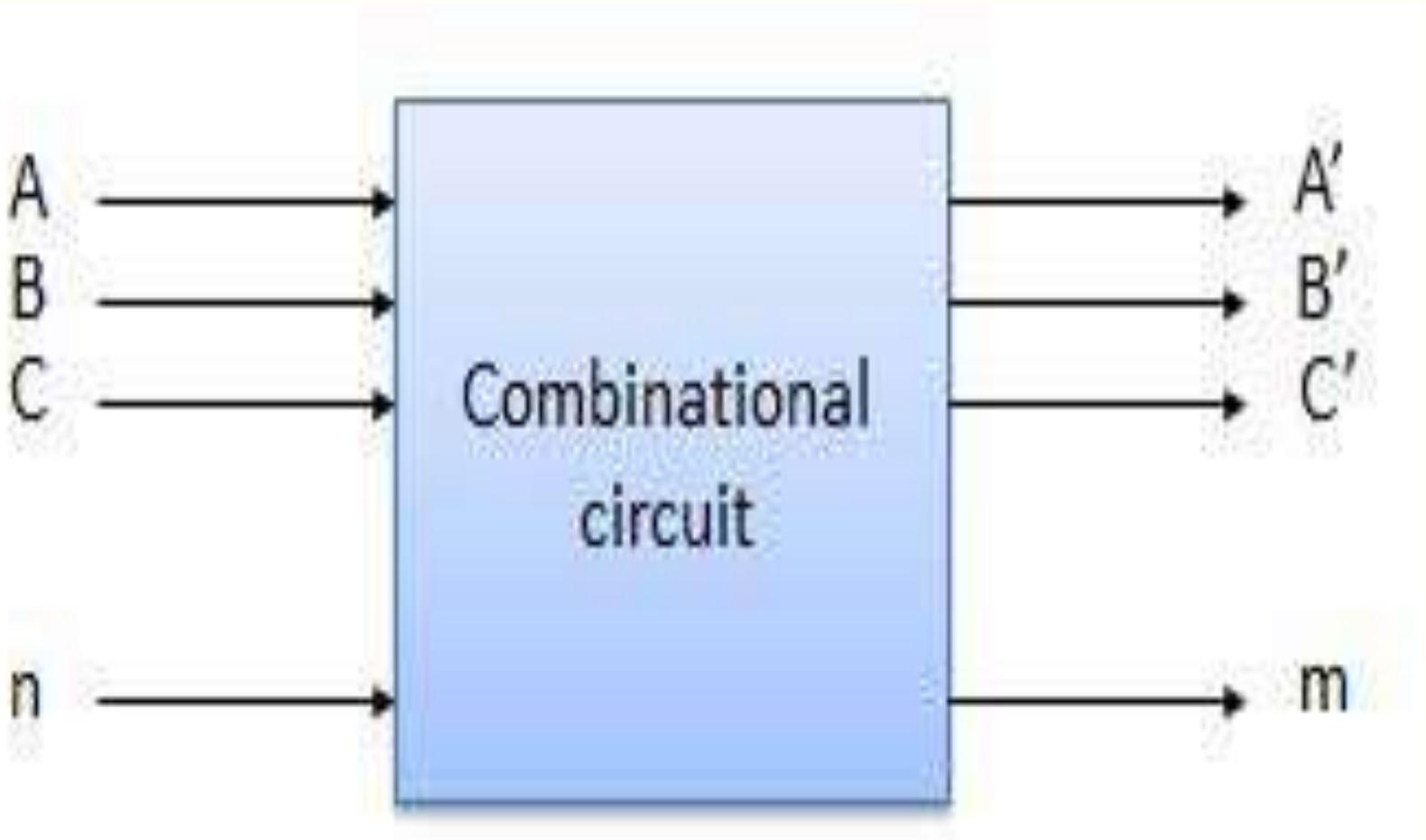
Chapter 4: COMBINATIONAL LOGIC

Er. Nawaraj Bhandari

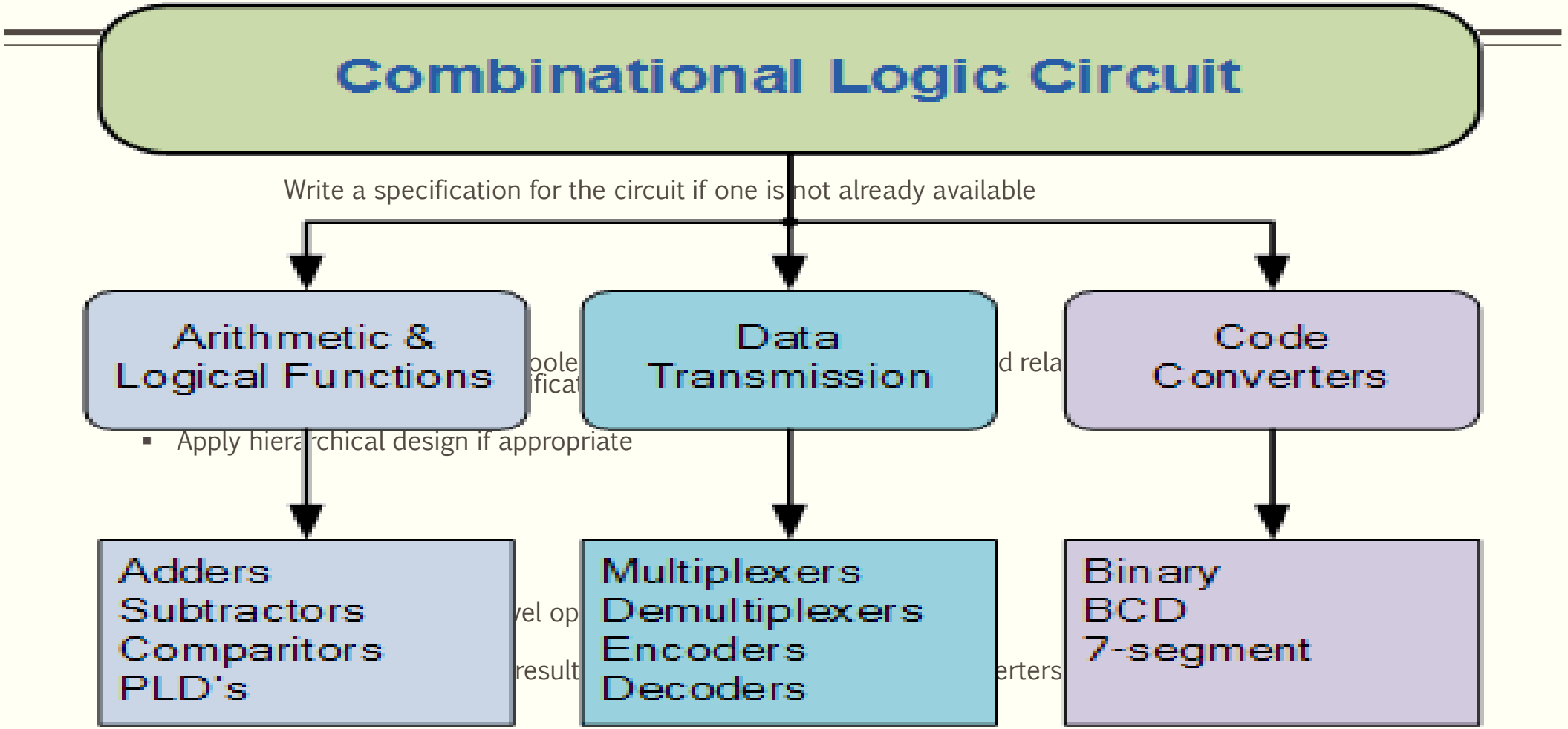


INTRODUCATION

- **combinational logic** is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only.
- T
  d
  h
- c
- T
  d
  s
- T
  fe
  t
  o



- Design procedure



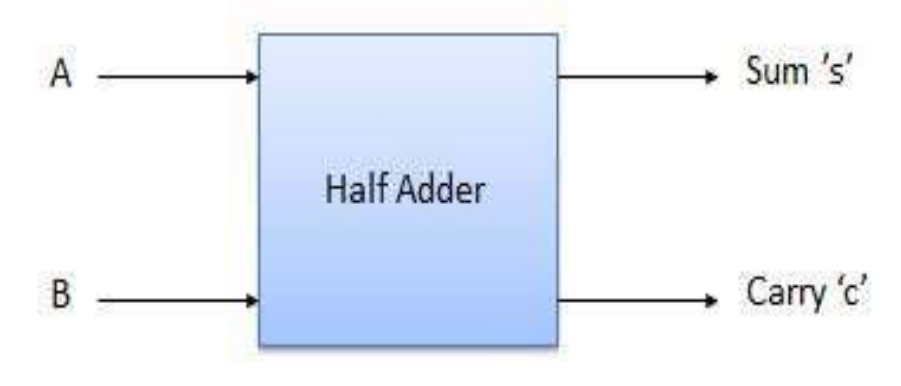
- Apply hierarchical design if appropriate

- 4. Technology Mapping
- Map the logic diagram to the implementation technology selected

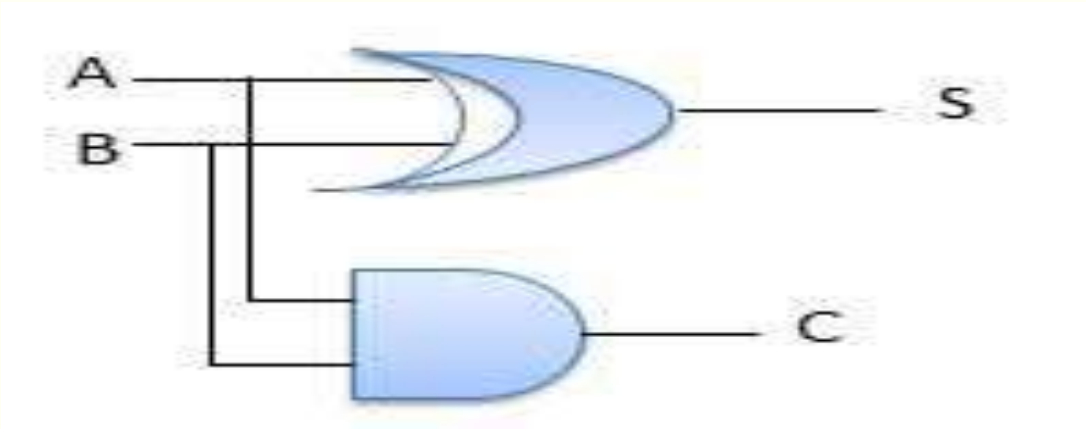
Adder

- Digital computers perform a variety of information-processing tasks. Among the basic functions encountered are the various arithmetic operations.
- The most basic arithmetic operation, no doubt, is the addition of two binary digits.
- **Half-Adder**

- A combinational circuit that performs the addition of two bits is called a *half-adder.*
- Circuit needs **two inputs** and **two outputs**. The input variables designate the augend (x) and addend (y) bits; the output variables produce the sum (S) and carry (C).
- In half adder we do not add carry from previous one.
- It is the basic building block for addition of two **single** bit numbers.



Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full-Adder

- A *full-adder* is a combinational circuit that forms the arithmetic sum of three input bits.
- It consists of **three inputs** and **two outputs**. Two of the input variables, denoted by *x* and *y*, represent the **two significant bits** to be added. The third input, z, represents the **carry** from the previous lower significant position.

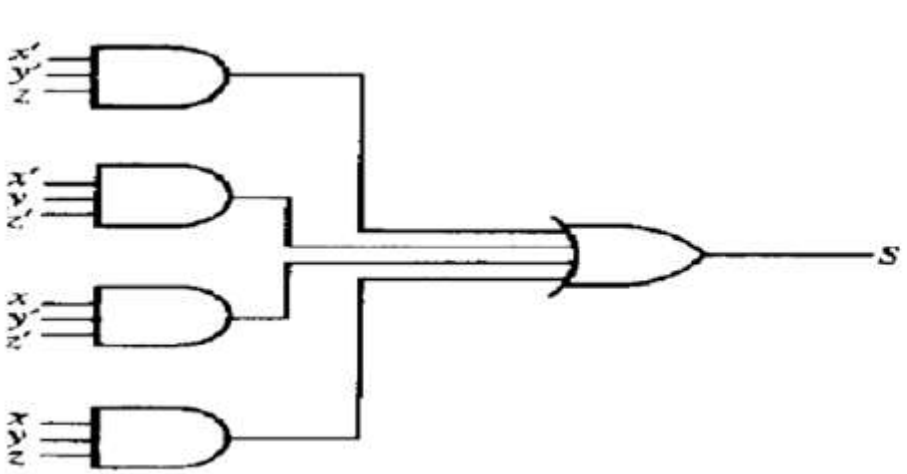
	1		1
1		1	

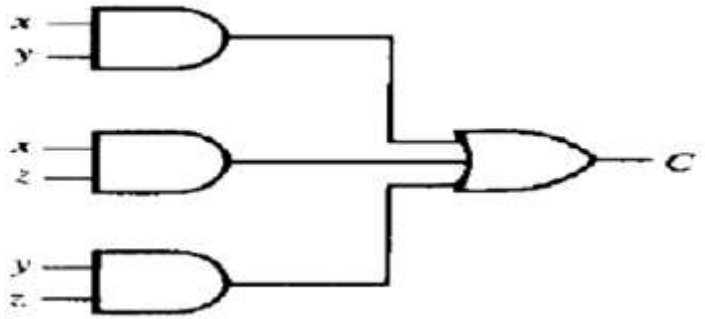
K-MAP For
S=A’B’C+A’BC’+AB’C’+ABC

		1	
	1	1	1

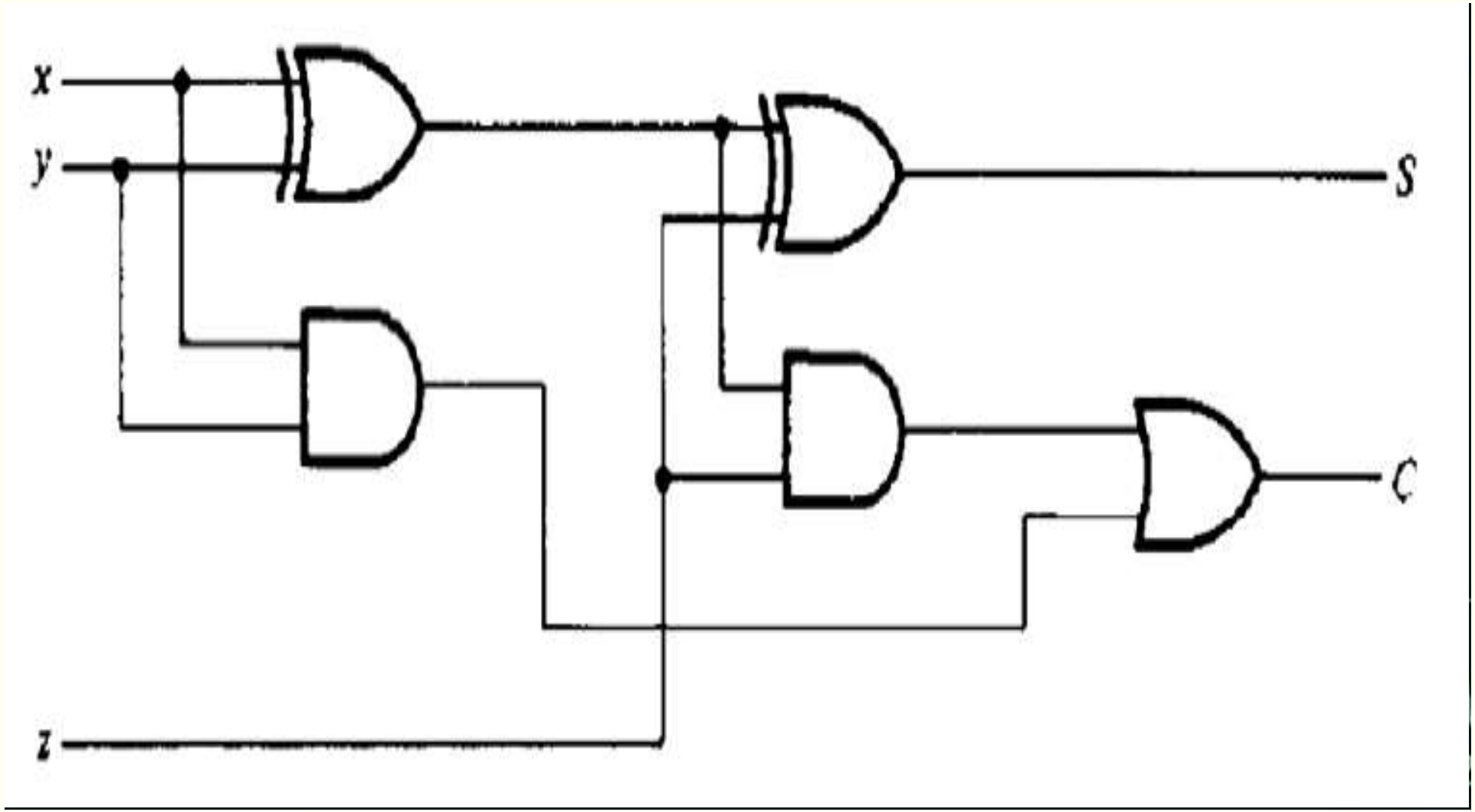
K-MAP For C0=AB+AC+BC

Inputs			Output	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1





IMPLIMENTATION OF FULL ADDER WITH TWO HALF ADDER

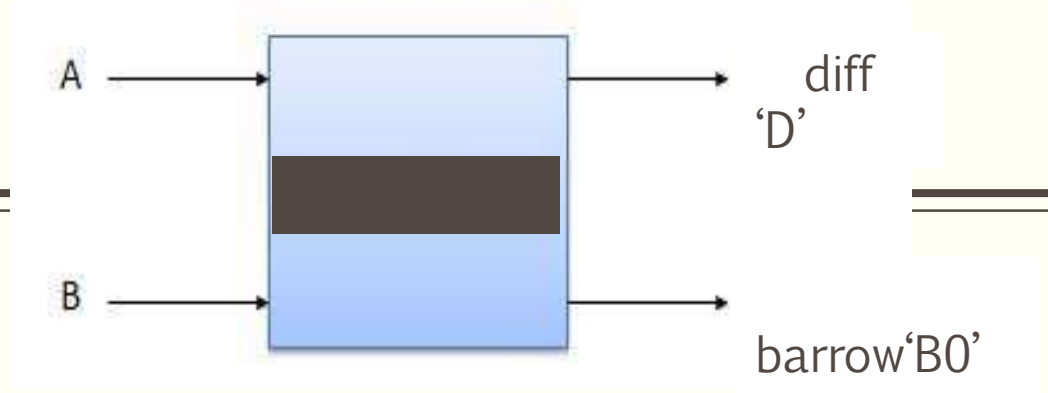


Substractor

- The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend.
- By this method, the subtraction operation becomes an addition operation requiring full-adders for its machine implementation.
- It is possible to implement subtraction with logic circuits in a direct manner, as done with paper and pencil.
- By this method, each subtrahend bit of the number is subtracted from its corresponding **significant minuend bit** to form a **difference bit**.
- If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position

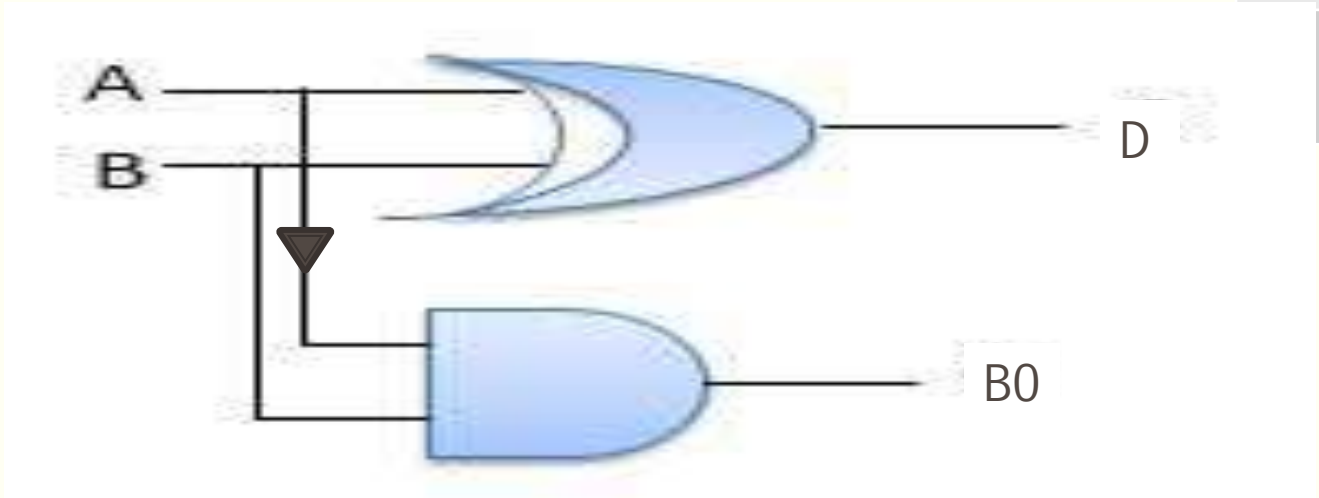
Half-Subtractor

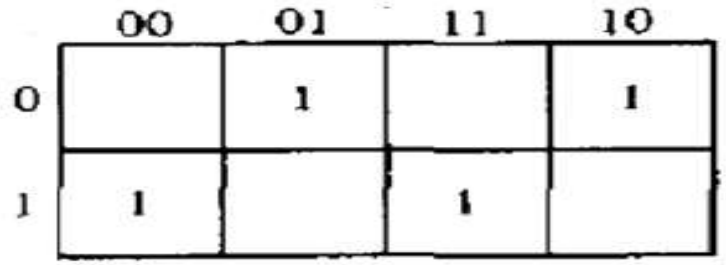
- A half-subtractor is a combinational circuit that subtracts two bits and produces their difference bit.
- Denoting minuend bit by x and the subtrahend bit by *y*.
- To perform *x - y,* we have to check the relative magnitudes of *x* and *y*:
- If $x \geq y$, we have three possibilities: 0 - 0 = 0, 1 - 0 = 1, and 1 - 1 = 0.
- If $x < y$, we have 0 - 1, and it is necessary to borrow a 1 from the next higher stage.
- The half-subtractor needs **two outputs**, difference (D) and borrow (B).



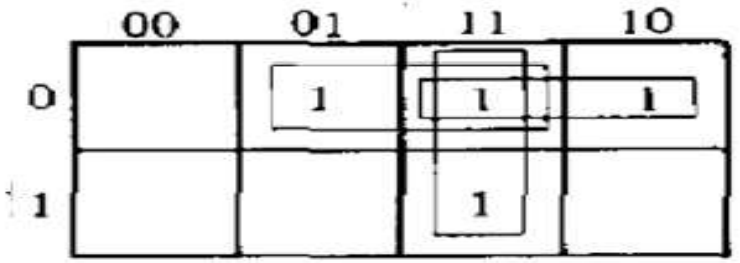
Inputs		Output
A	B	D
0	0	0
0	1	1
1	0	1
1	1	0

- D = A‘B + AB‘ B0= A‘B

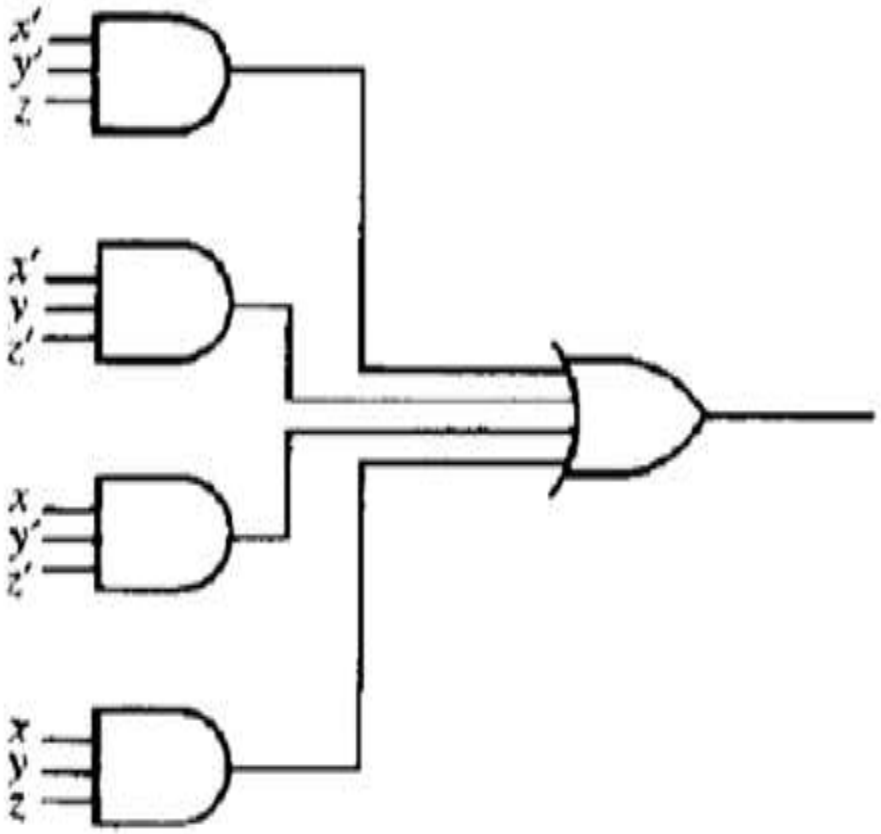


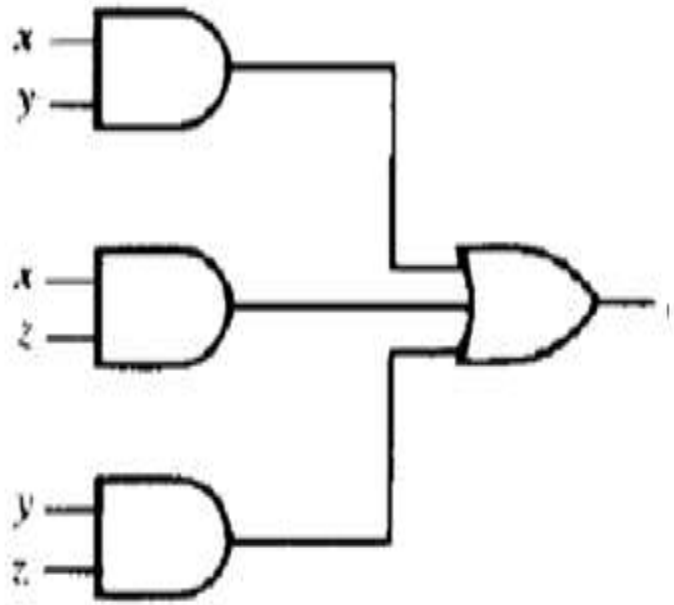


D = x'y'z + x'yz' + xy'z' + xyz



B = x'y + x'z + yz



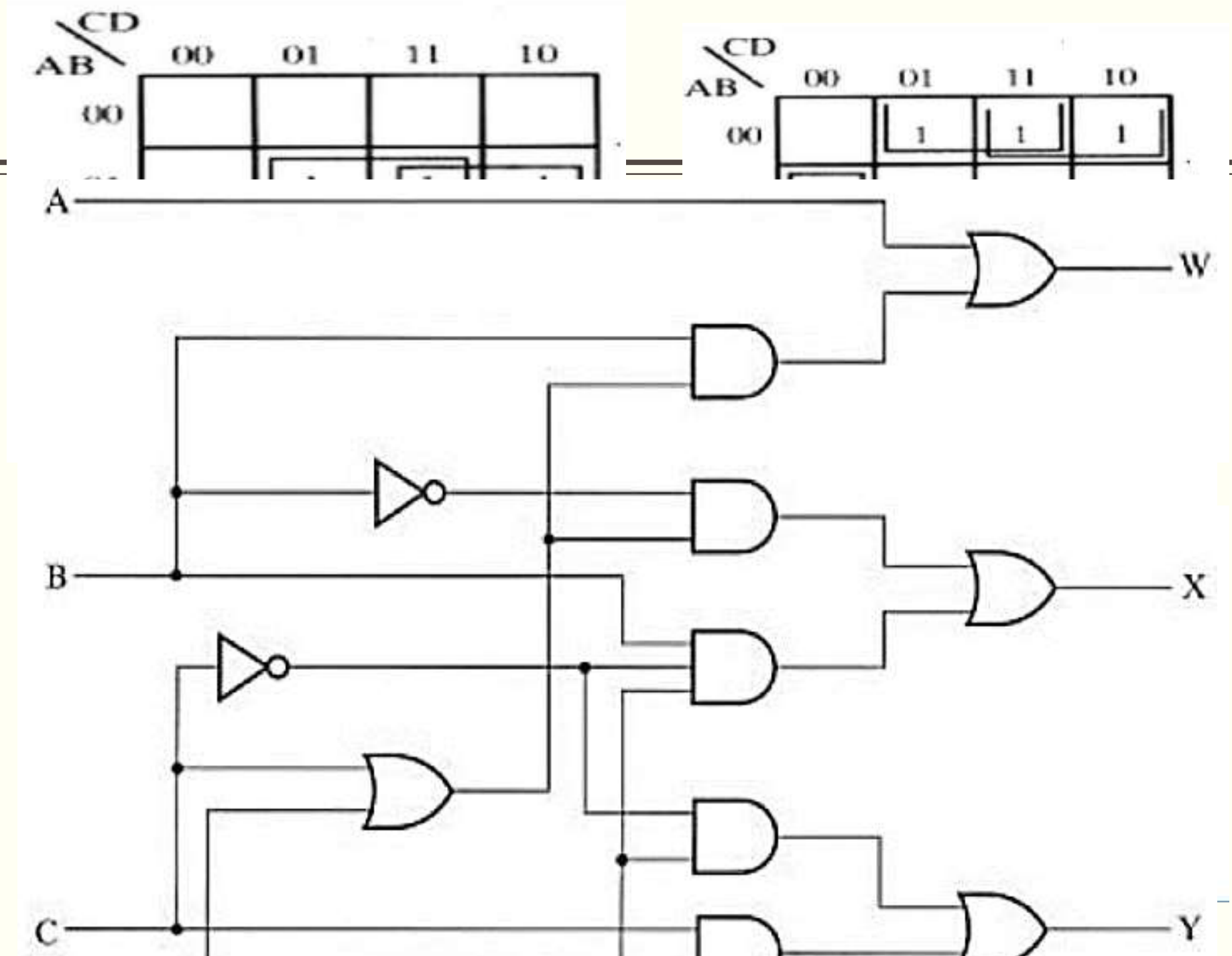


Code Conversion

- The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another.
- A conversion circuit must be inserted between the two systems if each uses different codes for the same information.
- Thus, a **code converter** is a circuit that makes the two systems compatible even though each uses a different binary code.
- To convert from binary code A to binary code B, code converter has **input lines** supplying the bit combination of elements as specified by code A and the output lines of the converter generating the corresponding bit combination of code B.
- A Code converter (combinational circuit) performs this transformation by means of logic gates.
- The design procedure of code converters will be illustrated by means of a *specific example* of conversion from the BCD to the excess-3 code.

BCD to Excess-3 code converter

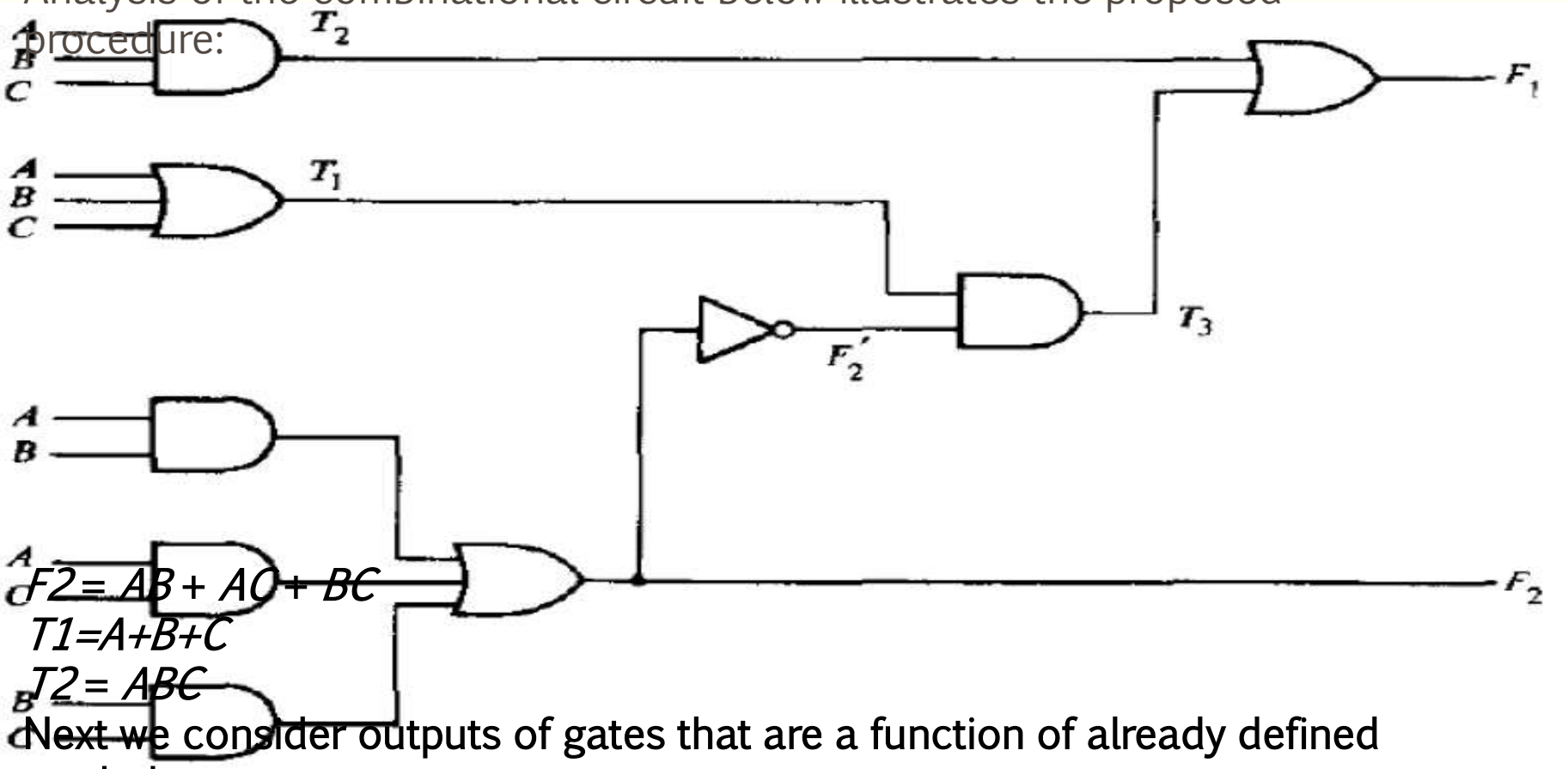
Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0


CD
AB
00
01
11
10
00
CD
AB
00
01
11
10
00
1
1
1
A
B
C
W
X
Y

- Analysis Procedure
- The design of a combinational circuit starts from the verbal specifications of a required function and ends with a set of output Boolean functions or a logic diagram. The **analysis of a combinational circuit** is somewhat the reverse process. It starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or a verbal explanation of the circuit operation.
- Steps in analysis:

- 1. The first step in the analysis is to make sure that the given circuit is combinational and not sequential.
- 2. Assign symbols to all gate outputs that are a function of the input variables. Obtain the Boolean functions for each gate.
- 3. Label with other arbitrary symbols those gates that are a function of input variables and/or previously labeled gates. Find the Boolean functions for these gates.
- 4. Repeat step 3 until the outputs of the circuit are obtained.
- 5. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables only.

Analysis of the combinational circuit below illustrates the proposed procedure:



F2 = AB + AC + BC
T1=A+B+C
T2 = ABC
Next we consider outputs of gates that are a function of already defined symbols:
T3 = F2'T1
F1 = T3 + T2

To obtain F1 as a function of A, B, and C, forms a series of substitutions as follows:

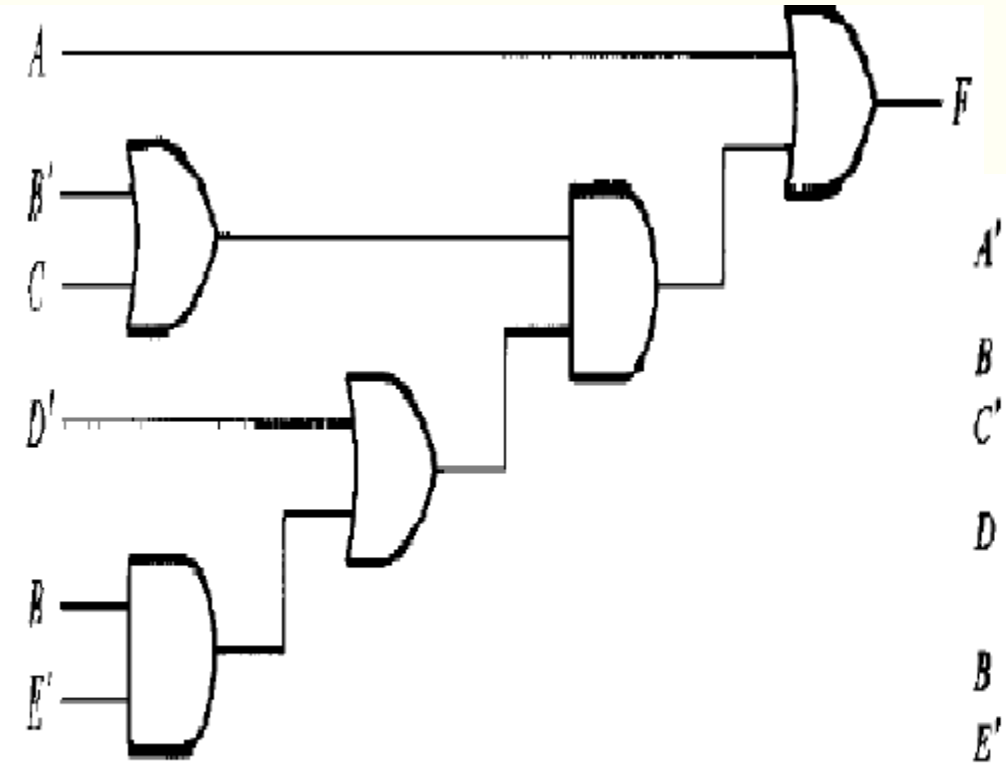
$$F_1 = T_3 + T_2 = F_2' T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC$$

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

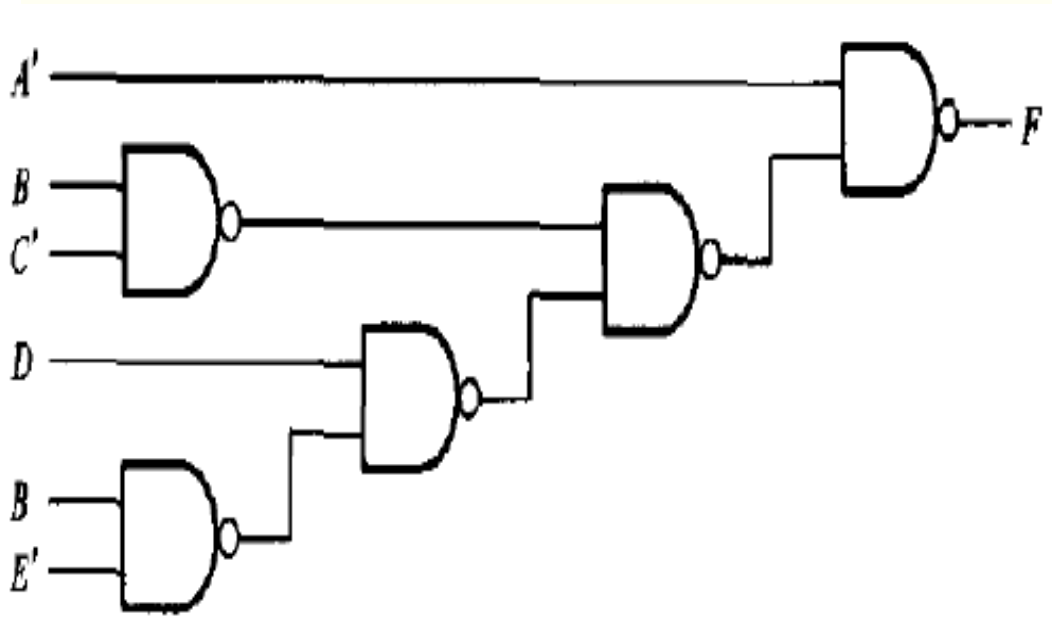
Concept of multi-level NAND and NOR circuits

- To implement a Boolean function with NAND gates we need to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic.
- The conversion of an algebraic expression from AND, OR, and complement to NAND can be done by simple circuit-manipulation techniques that change AND-OR diagrams to NAND diagrams.
- To obtain a multilevel NAND diagram from a Boolean expression, proceed as follows:
- 1. From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume that both the normal and complement inputs are available.
- 2. Convert all AND gates to NAND gates with AND-invert graphic symbols.
- 3. Convert all OR gates to NAND gates with invert-OR graphic

- Example: F = A + (B' + C) (D' + BE ')

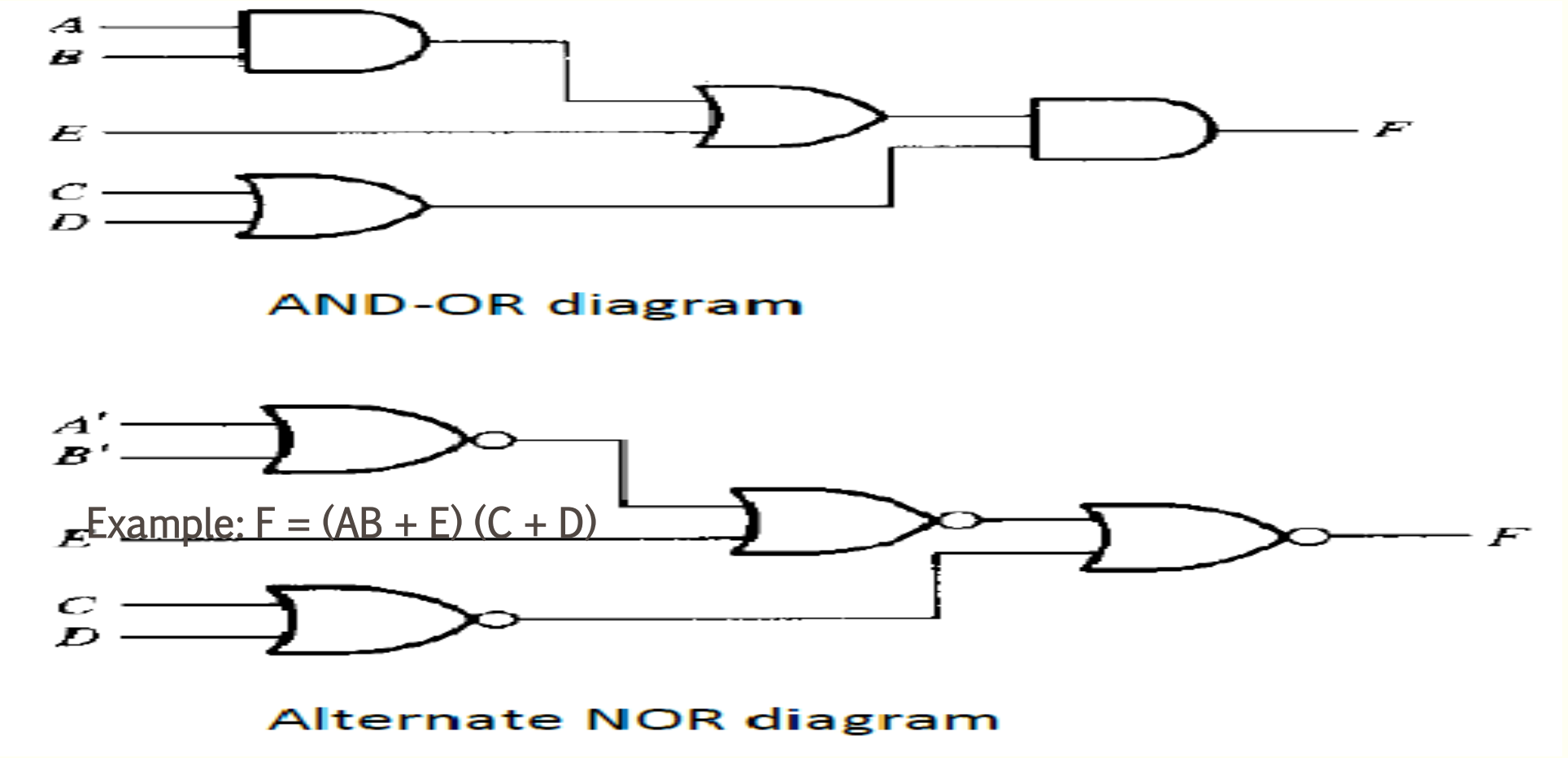


AND-OR diagram



NAND diagram using one graphic symbol

- Multi-level NOR circuits



Example: F = (AB + E) (C + D)