since 2001

COSMOS COLLEGE
OF MANAGEMENT AND TECHNOLOGY

Affiliated to Pokhara University

Combinational Logic

Manish Man Shrestha

Introduction

- A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms binary information from the given input data to the required output data

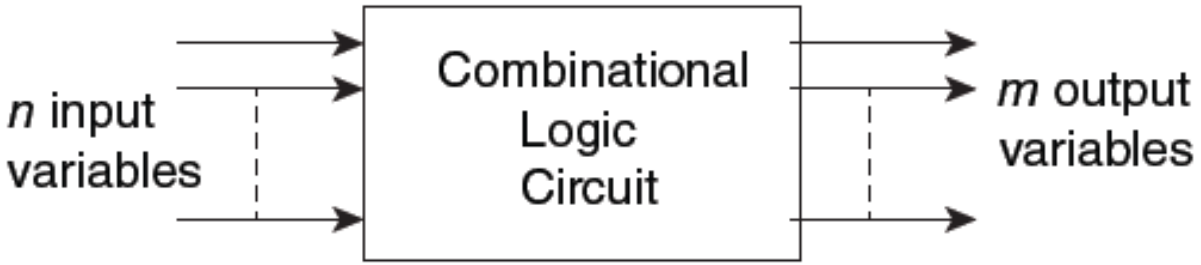


Figure 1: Block diagram of a combinational circuit

Adders

- This simple addition consists of four possible elementary operations, namely, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10. Here, when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a *carry*.
- A combinational circuit that performs the addition of two bits is called a half-adder.
- A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is a full-adder.

Half adder

- We arbitrarily assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The is shown below

a) Truth table

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

b)
K-map for S

x \ y	0	1
0		(1)
1	(1)	

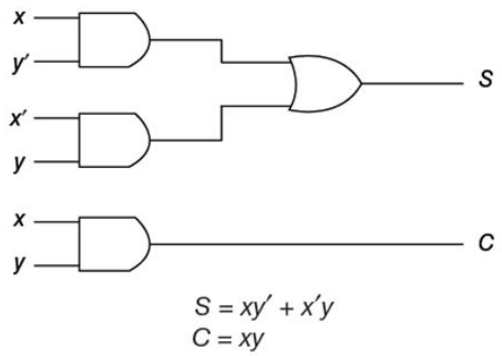
$S = x'y + xy'$
$= x \oplus y$

K-map for C

x \ y	0	1
0		
1		(1)

$C = xy$

c) Circuit implementation



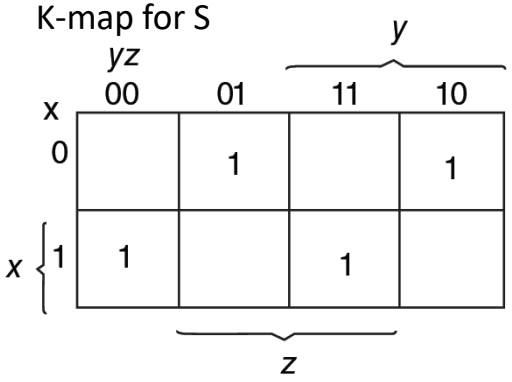
Various implementations of a half-adder

Full adder

- 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. The two outputs are designated by the symbols S for sum and C for carry.

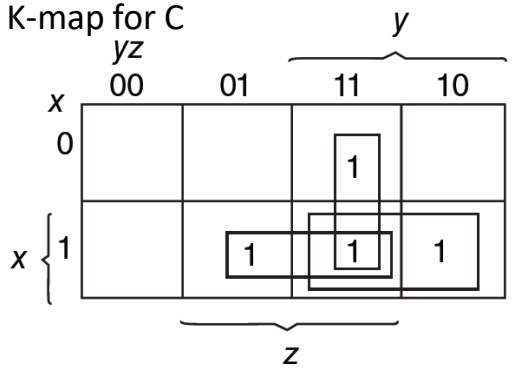
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map for S

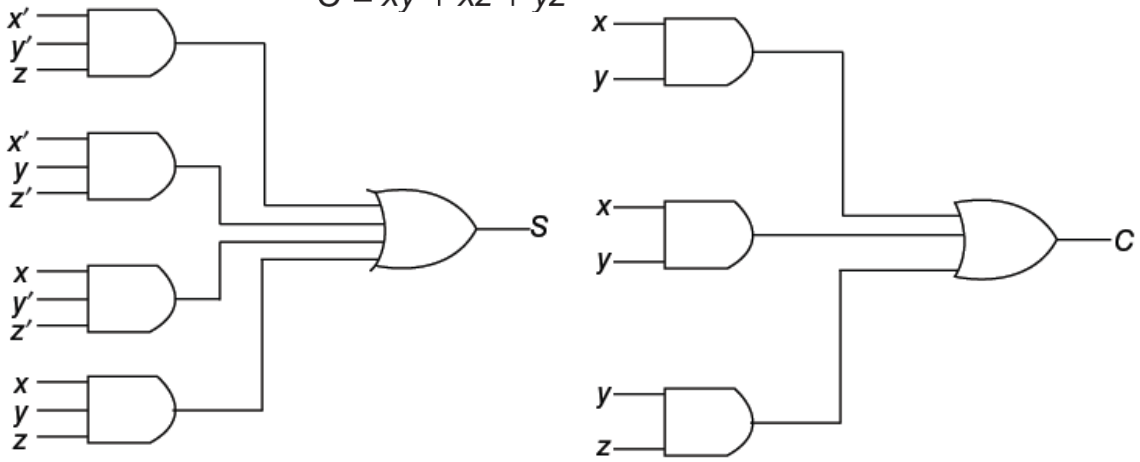


$S = x'y'z + x'yz' + xy'z' + xyz$

K-map for C



$C = xy + xz + yz$

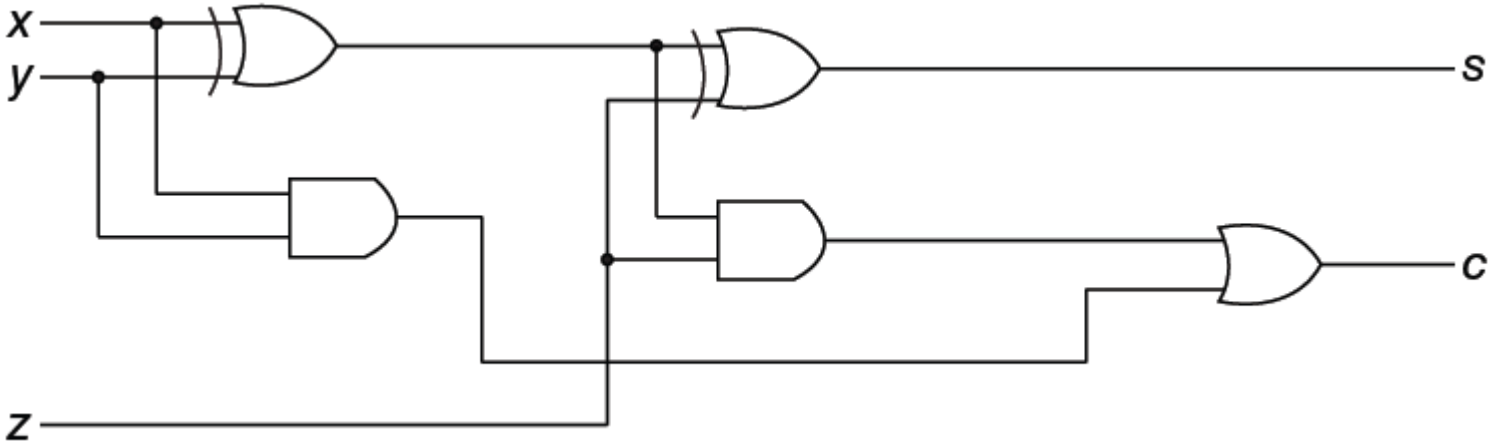


Full adder using the half adders

We have,

S = x'y'z + x'yz' + xy'z' + xyz
= z'(xy' + x'y) + z(xy + x'y')
= z'(xy' + x'y) + z(xy' + x'y)' [since complement of XNOR is XOR]
= z ⊕ (x ⊕ y)

C = xy + xz + yz
= xy + xz + yz(x + x')
= xyz + xy + xz + x'yz
= xy(z + 1) + xz + x'yz
= xy + xz + x'yz
= xy + xz(y+y') + x'yz
= xyz + xy + xy'z + x'yz
= xy(z + 1) + xy'z + x'yz
= xy + z(xy' + x'y)
= xy + z(x ⊕ y)

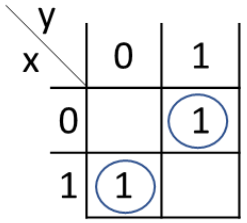


Half subtractor

- A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. Designate the minuend bit by x and the subtrahend bit by y. The half-subtractor needs two outputs. One output generates the difference and will be designated by the symbol D. The second output, designated B for borrow, generates the binary signal that informs the next stage that a 1 has been borrowed.

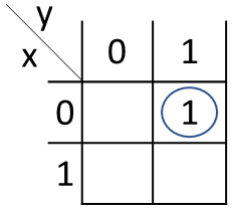
x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-map for D



$$D = x'y + x\,y'$$

K-map for B

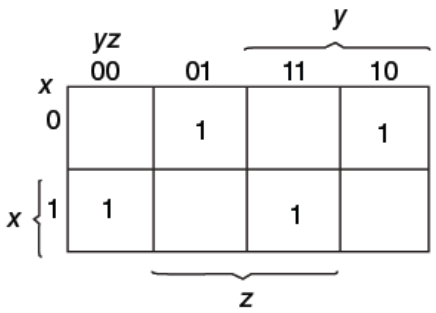


$$B = x'y$$

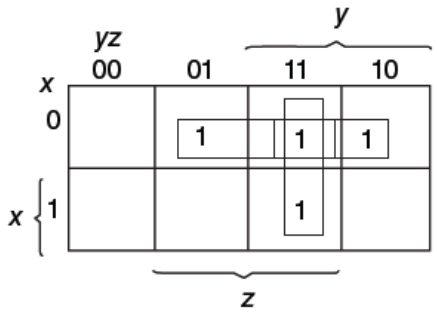
Full subtractor

- A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs. The three inputs, x, y, and z, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and B, represent the difference and output borrow, respectively.

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



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



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

The 1's and 0's for the output variables are determined from the subtraction of x - y - z.

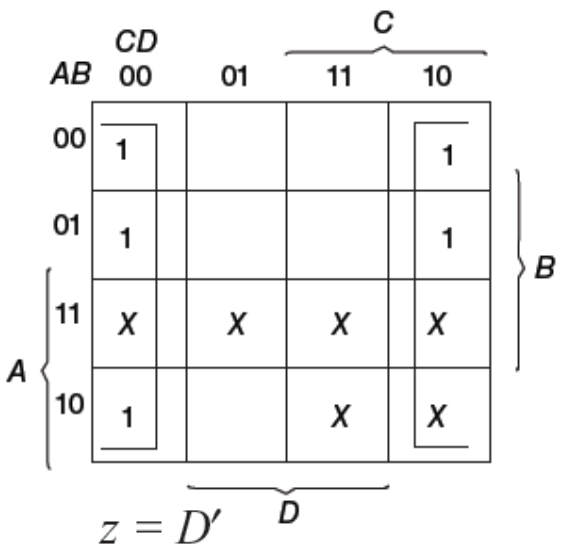
For x = 0, y = 0, and z = 1, we have to borrow a 1 from the next stage, making B = 1 and adds 2 to x. Since 2 - 0 - 1 = 1, D = 1.
For x = 0 and yz = 11, we need to borrow again, making B = 1 and x = 2. Since 2 - 1 - 1 = 0, D = 0.
For x = 1 and yz = 01, we have x - y - z = 0, which makes B = 0 and D = 0.
For x = 1, y = 1, z = 1, we have to borrow 1, making B = 1 and x = 3, and 3 - 1 - 1 = 1, making D = 1.

BCD to excess-3 code

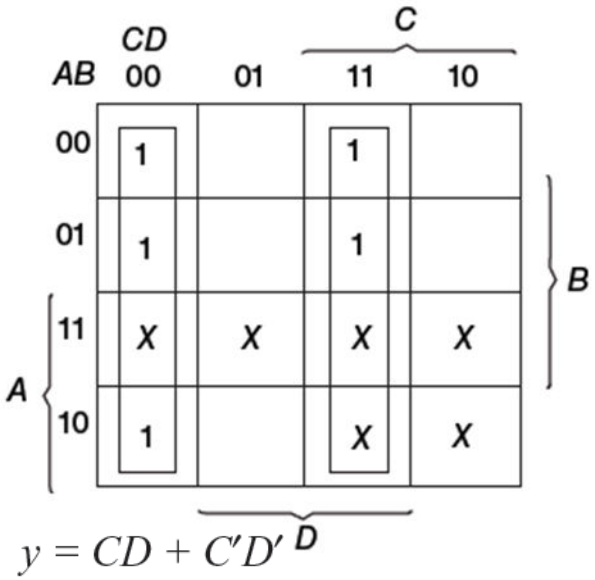
Input BCD				Output Excess-3 code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

The six bit combinations not listed for the *input* variables are don't-care combinations. Since they will never occur, we are at liberty to assign to the output variables either a 1 or a 0, whichever gives a simpler circuit.

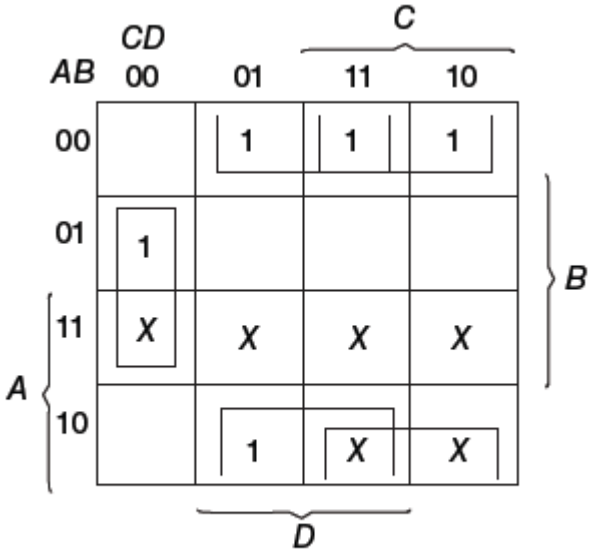
BCD to excess-3 code: K-map



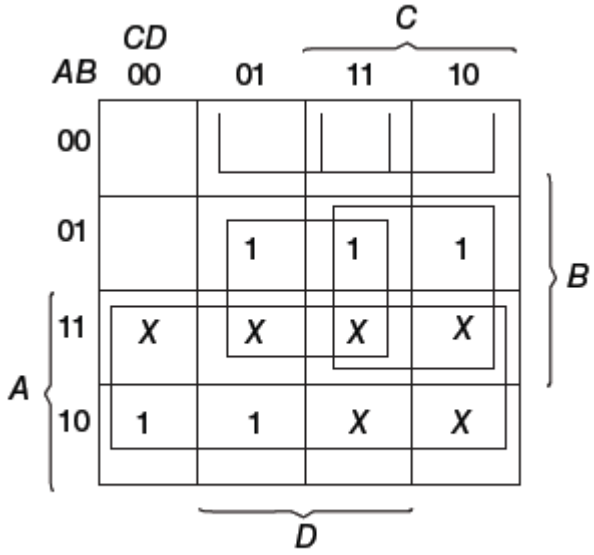
$z = D'$



$y = CD + C'D'$



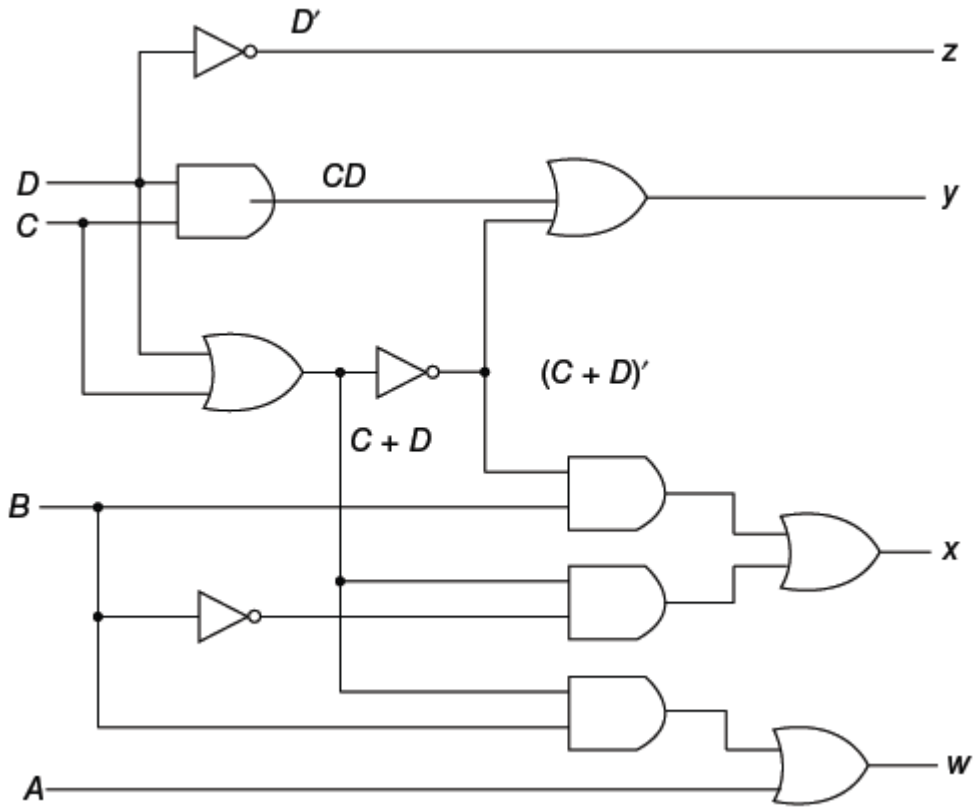
$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

BCD to excess-3 code: Logic diagram

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$



Combination circuit example

A) Problem description

y is 1 if a is to 1, or b and c are 1. z is 1 if b or c is to 1, but not both, or if all are 1.

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

C) Minimized output equations

y

a \ bc	00	01	11	10
0	0	0	1	0
1	1	1	1	1

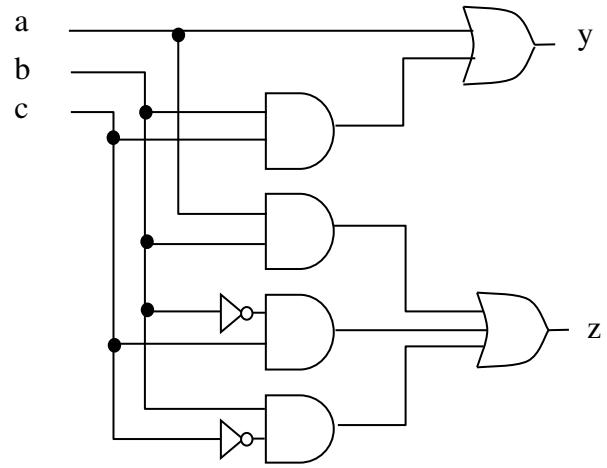
y = a + bc

z

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

z = ab + b'c + bc'

D) Logic Gates



Question

- Binary to Gray code conversion
- A circuit has four inputs and two outputs. One of the outputs is high when majority of inputs are high. The second output is high only when all inputs are of same type. Design the combinational circuit.
  (Ans: Y1 = ABD+BCD+ACD+ABC; Y2 = A'B'C'D' + ABCD)

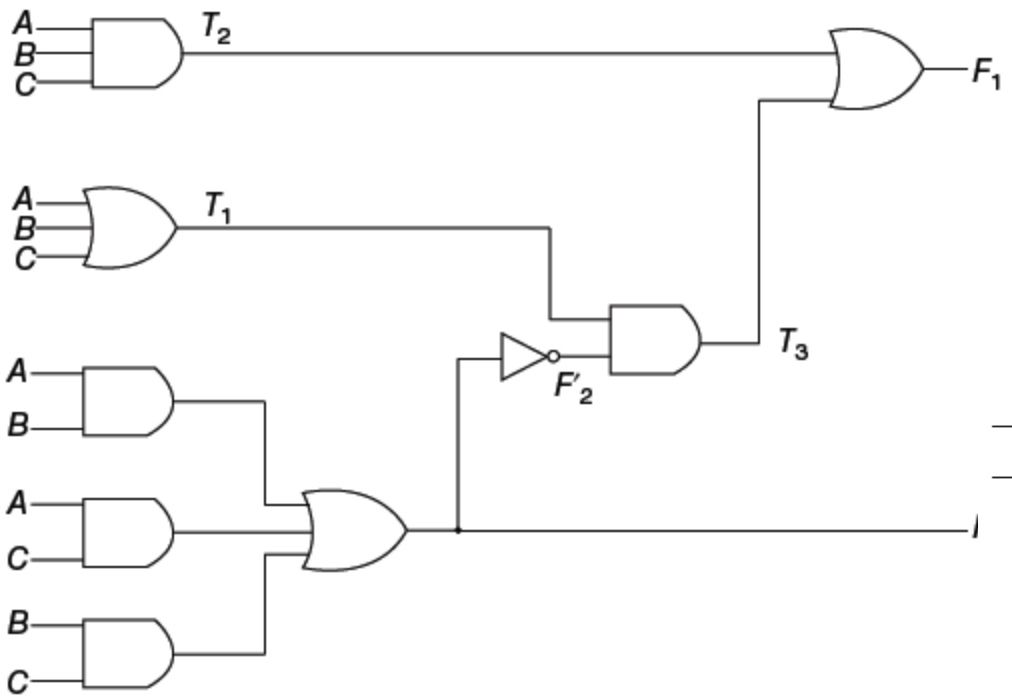
Analysis procedure

It starts with a given logic diagram and culminates with a set of Boolean functions, a truth table

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

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

Analysis procedure: Example



Logic diagram for analysis example

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

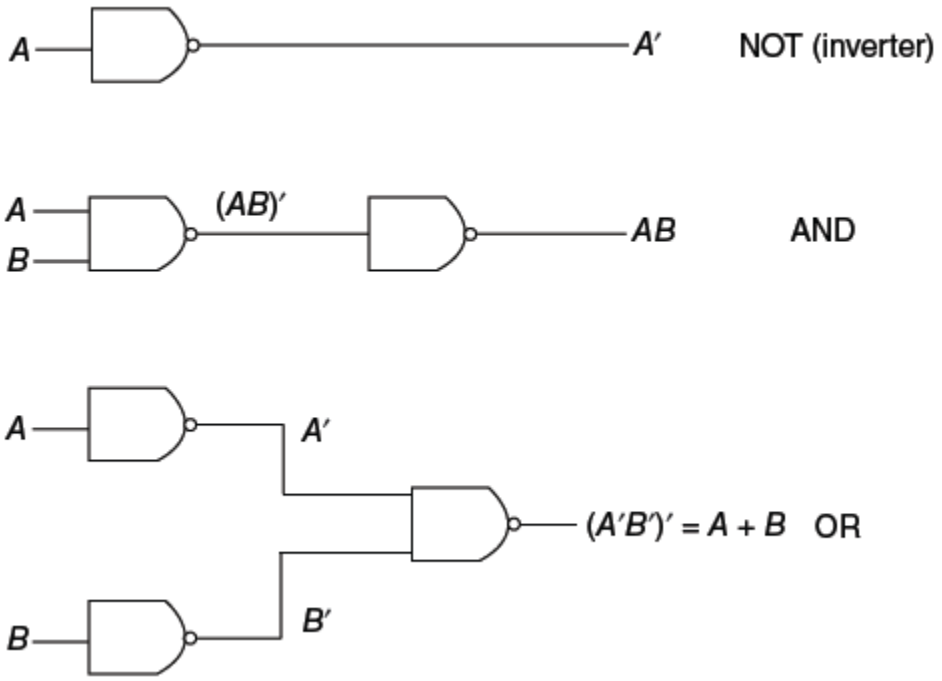
$$T_3 = F'_2\, T_1$$
$$F_1 = T_3 + T_2$$

$$F_1 = T_3 + T_2 = F'_2\, T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC$$
$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$
$$= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC$$
$$= A'BC' + A'B'C + AB'C' + ABC$$

Table: Truth table for logic diagram

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	I	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

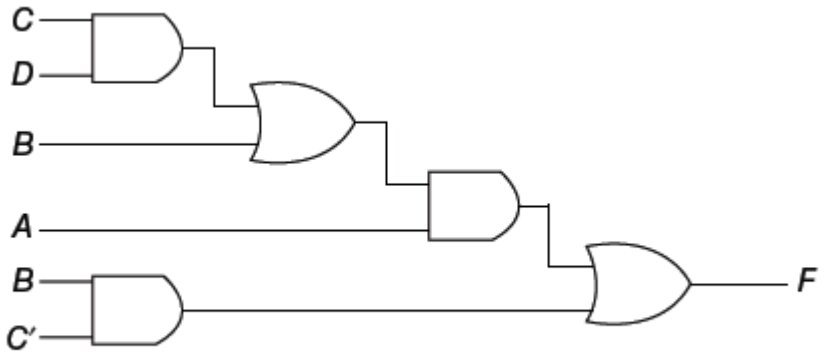
Multilevel NAND circuits



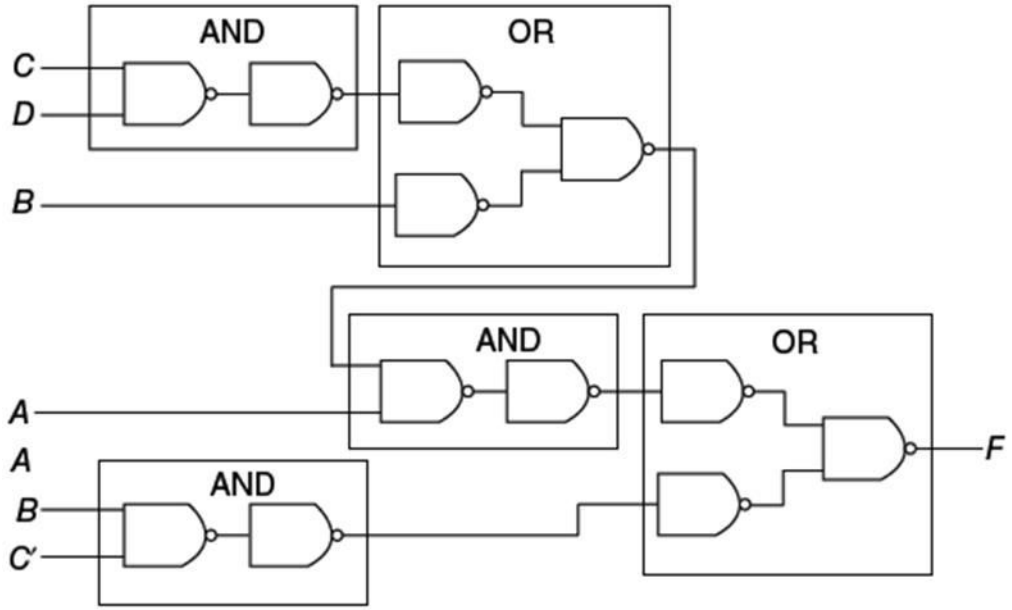
Implementation of NOT. AND, or OR by NAND gates

Multilevel NAND circuits: Block diagram method

F = A(B + CD) + BC′

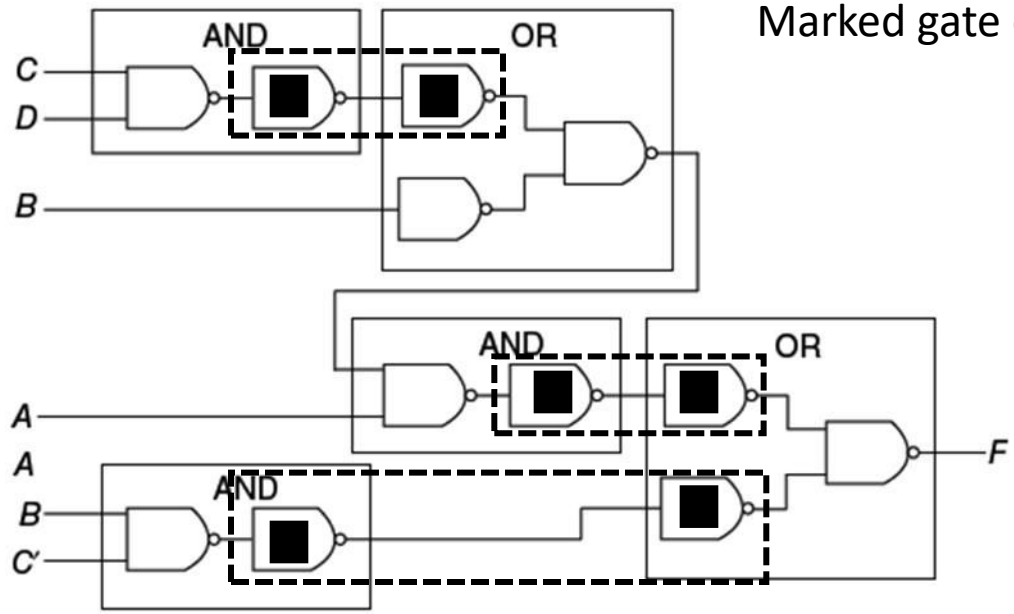


(a) AND/OR implementation



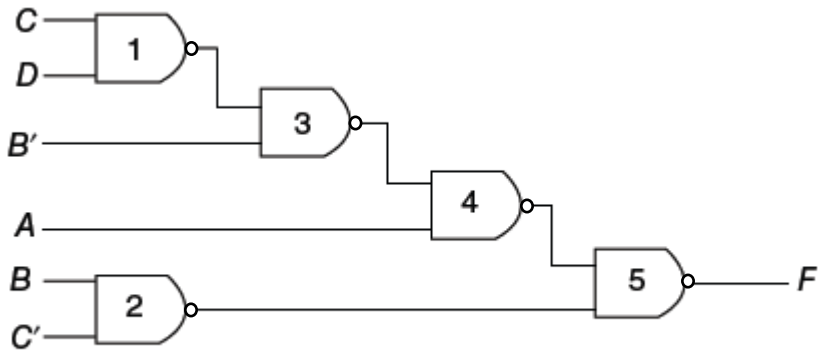
(b) Substituting equivalent NAND functions

Multilevel NAND circuits: Final implementation of example



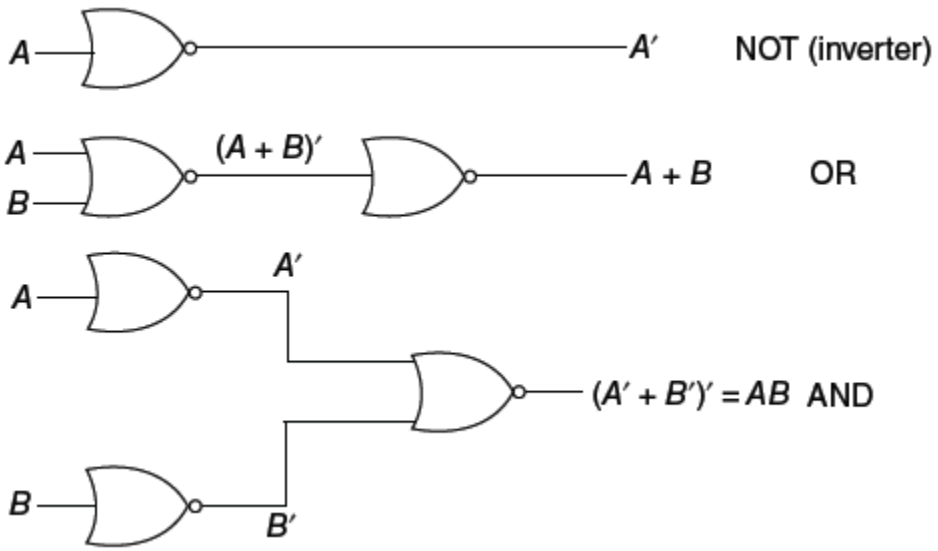
(b) Substituting equivalent NAND functions

Marked gate can be eliminated since they are inverter of inverter



(c) NAND implementation

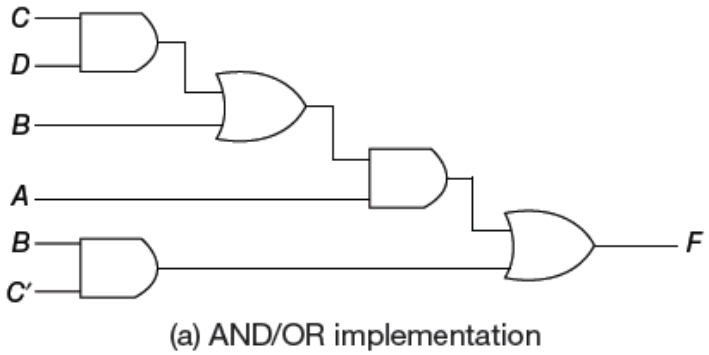
Multilevel NOR circuits



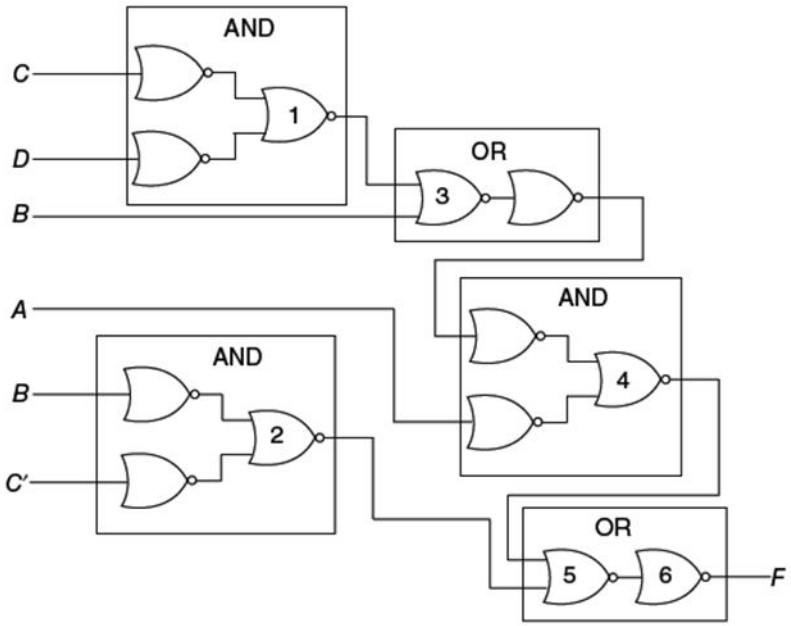
Implementation of NOT, OR, and AND by NOR gates

Multilevel NOR circuits: Block Diagram Method

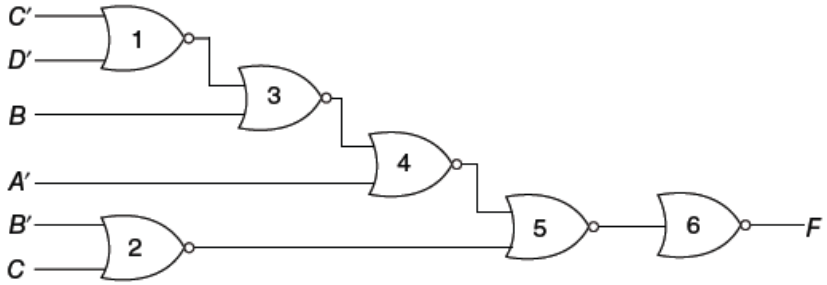
$F = A(B + CD) + BC'$



(a) AND/OR implementation



(b) Substitution equivalent NOR functions

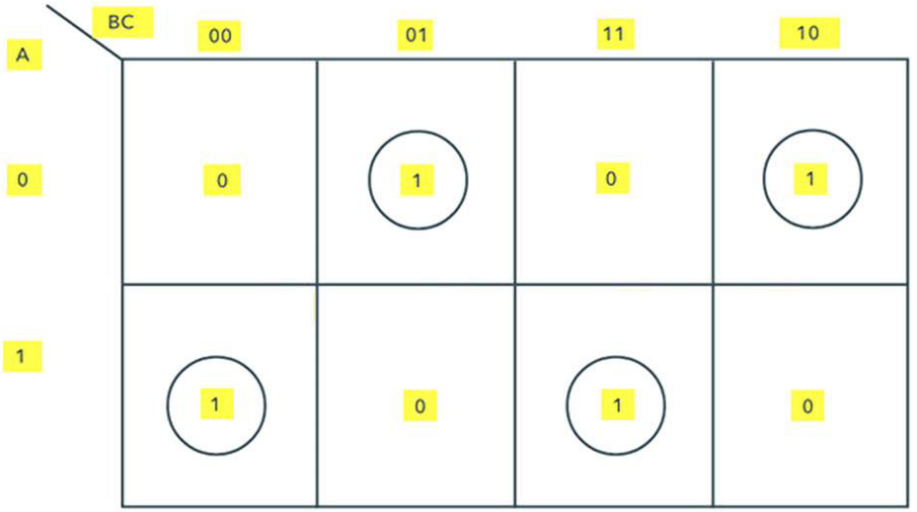


(c) NOR implementation

3-bit even parity generator

3-bit message			Even parity bit generator (P)
A	B	C	Y
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

K-Map For 3-Bit Even Parity Generator

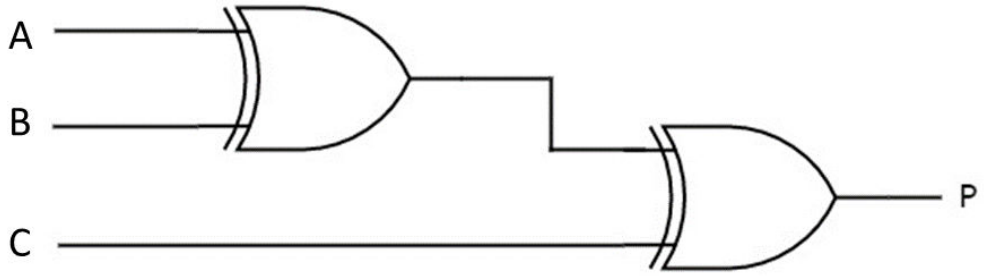


$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

$$= \bar{A}\,(B \oplus C) + A\,(\overline{B \oplus C})$$

$$P = A \oplus B \oplus C$$



Even Parity Checker

4-bit received message				Parity error check C_p
A	B	C	P	
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

The table shows the truth table for the Even Parity Checker in which PEC = 1 if the error occurs, i.e., the four bits received have odd number of 1s and PEC = 0 if no error occurs, i.e., if the 4-bit message has even number of 1s.

Even Parity Checker: K-map and logic circuit

K-Map For Even Parity Checker

AB \ cp	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

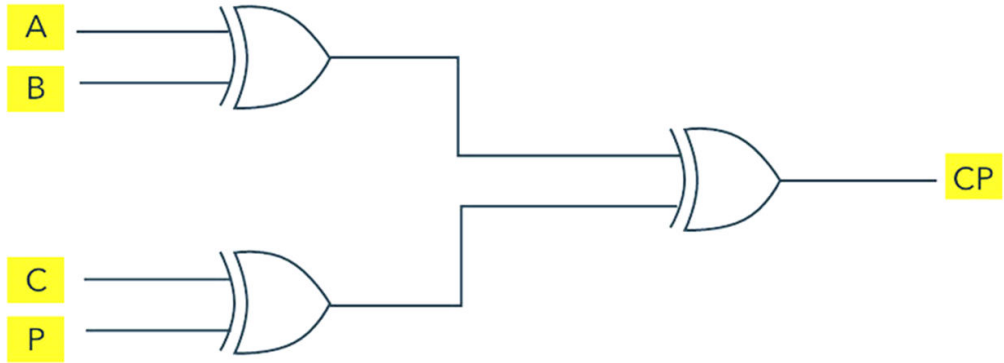
$$PEC = \overline{A}\ \overline{B}\ (\overline{C}\ P + C\overline{P}) + \overline{A}\ B(\overline{C}\ \overline{P} + CP) + AB(\overline{C}\ P + C\overline{P}) + A\overline{B}\ (\overline{C}\ \overline{P} + CP)$$

$$= \overline{A}\ \overline{B}\ (C \oplus P) + \overline{A}\ B(\overline{C \oplus P}) + AB(C \oplus P) + A\overline{B}\ (\overline{C \oplus P})$$

$$= (\overline{A}\ \overline{B} + AB)(C \oplus P) + (\overline{A}\ B + A\overline{B})(\overline{C \oplus P})$$

$$= (A \oplus B) \oplus (C \oplus P)$$

Logic Circuit Of Even Parity Checker



Assignment

- Odd parity generator
- Odd parity checker