# CHAPTER 4
# COMBINATIONAL LOGIC

Logic circuits

**Combinational circuits**

(The circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs)

✓ ch4: Analysis and design

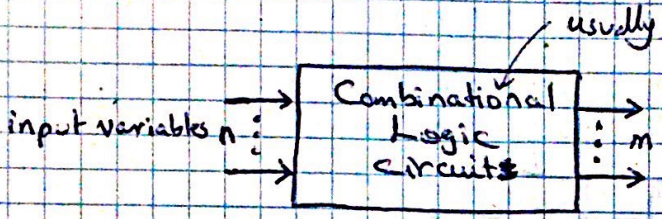
✓ ch5: Common Combinational devices

**Sequential circuits**

(The circuit employs memory elements (flip-flops) in addition to logic gates. The outputs depend not only on present inputs, but also on past inputs [system state])

✓ ch.6: Analysis and design

ch7: Common sequential devices.




Block diagram of a Combinational circuit.

## DESIGN PROCEDURE:

The procedure for the design of a combinational circuit involves the following steps:

1- The problem is stated verbally.

2- The number of available input and required output variables is determined.

3- The input and output variables are assigned letter symbols.

4- The truth table that defines the relationships between inputs and outputs is found.

5- The simplified Boolean function for each output is obtained.

6- The logic diagram is drawn.

## ADDERS:

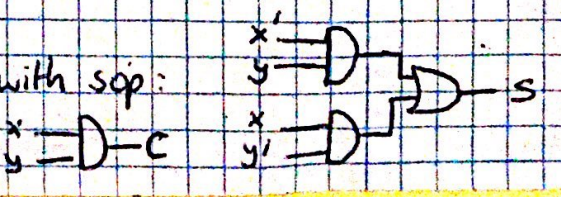
**Half-Adder:** A combinational circuit that performs the addition of two bits is called a half-adder. It needs two binary inputs (augend and addend) and two binary outputs (sum and carry)

| x | y | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

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

$C = xy$

$\Rightarrow$ with sop:



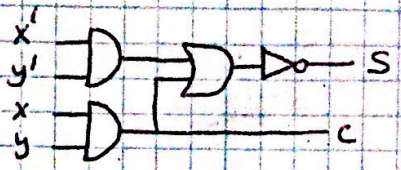

Scanned with CamScanner

Note that $S = x \oplus y \Rightarrow S' = x \odot y = xy + x'y'$ ; $C = xy$

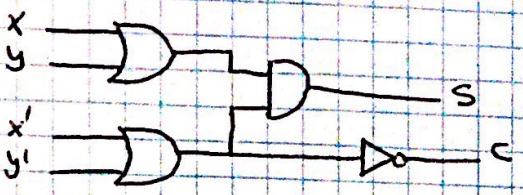
$\Rightarrow S' = C + x'y' \Rightarrow S = (C + x'y')'$ ∴

If we express S in POS:

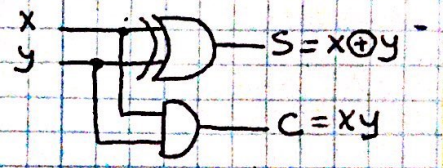
$\Rightarrow S = (x+y)(x'+y')$ ; $C = xy = (x'+y')'$




∴



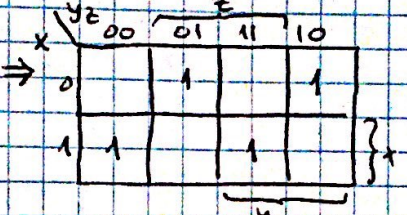

or simply;




**FULL-ADDER:** A combinational circuit that performs the addition of three bits is called a full-adder. It has 3 inputs and 2 outputs.

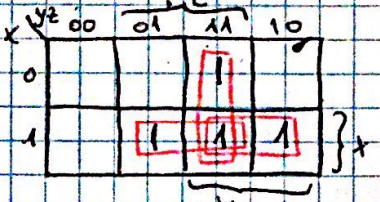
| x | y | z | C | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

↑ previous carry



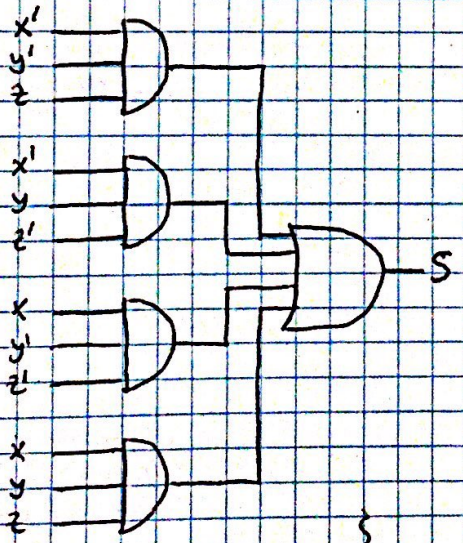

$S = \sum(1,2,4,7)$

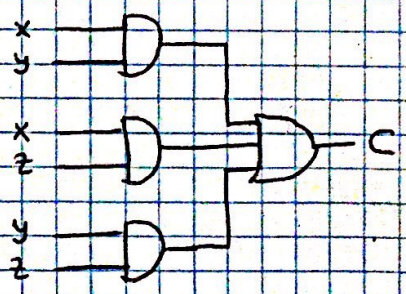
$= x'y'z + x'yz' + xy'z' + xyz$




$\Rightarrow C = xy + xz + yz$

$= \sum(3,5,6,7)$



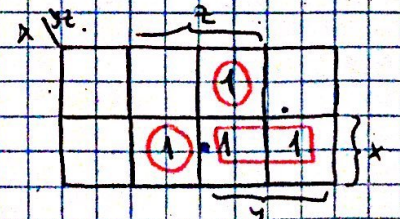




Now,

$S = x'(y'z + yz') + x(y'z' + yz)$

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

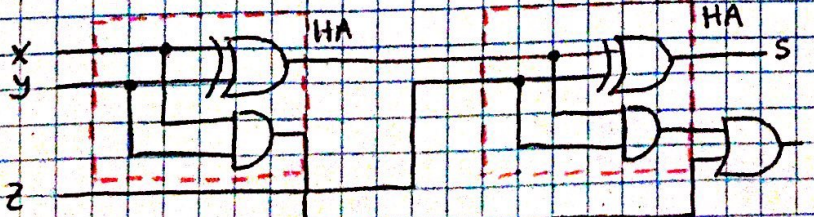
$= x'w + xw' = x \oplus w = x \oplus y \oplus z$




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

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

$= xy + (x \oplus y)z$




⇒ A full-Adder can be implemented with two half-adders and one OR gate

Scanned with CamScanner

## SUBTRACTORS:

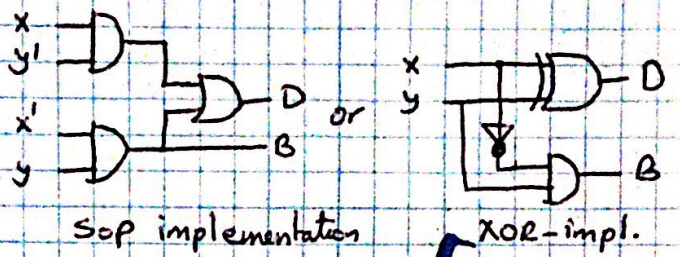
**Half-Subtractor:** A combinational circuit that subtracts two bits and produces their difference is called a half-subtractor. It has also an output to specify if a 1 has been borrowed.

| x | y | B | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

Borrow, Difference

$\Rightarrow D = x'y + xy' = x \oplus y$

$B = x'y$

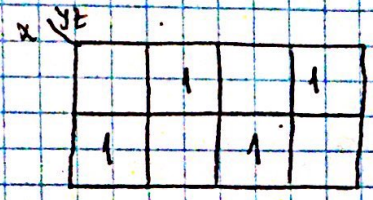


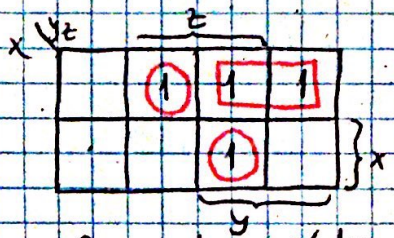

SOP implementation — XOR-impl.

**Full-Subtractor:** A full subtractor is a combinational circuit that performs a subtraction between 3 bits. 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 |

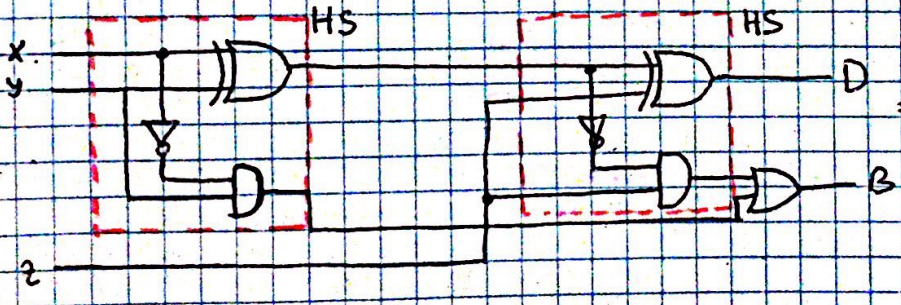
minuend, subtrahend, previous borrow



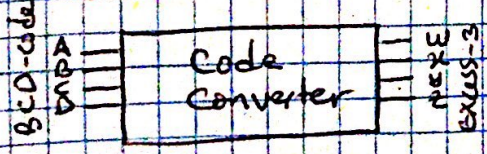

$\Rightarrow D = x \oplus y \oplus z$
$= x'y'z' + xy'z' + x'y'z + xyz$
$= (x'y + xy')z' + (x'y + xy')'z$

$B = x'y + x'y'z + xyz$
$= x'y + (x'y' + xy)z$
$= x'y + (x \odot y)z$
$= x'y + (x \oplus y)'z$




⇒ A full-subtractor can be implemented with two half-subtractors and an OR gate.

## CODE CONVERSION:

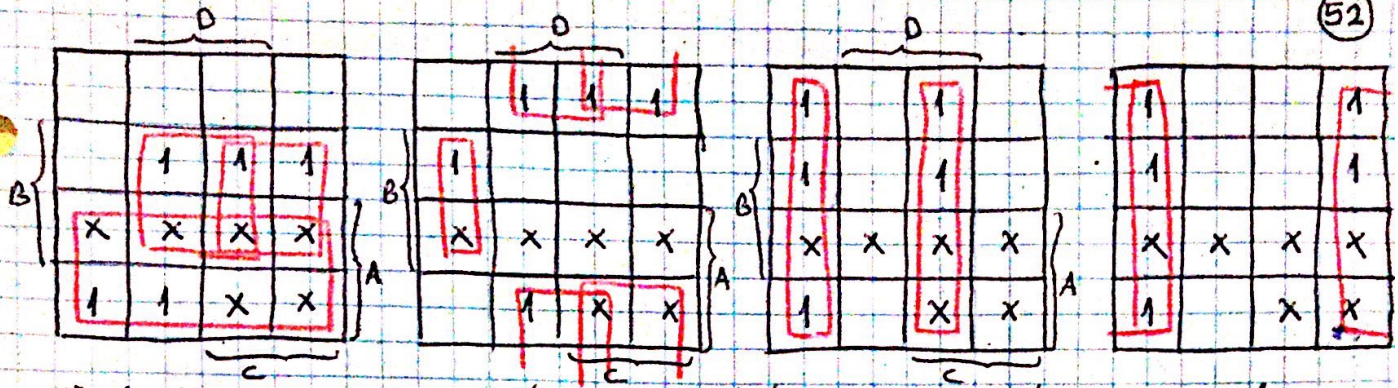



This combinational logic circuit converts BCD coded decimal digits to Excess-3 code.

| Inputs | | | | outputs | | | |
|---|---|---|---|---|---|---|---|
| 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 |

⇒ use the K-map to simplify each output.

Scanned with CamScanner




$W = A + BC + BD$




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




$y = CD + C'D'$



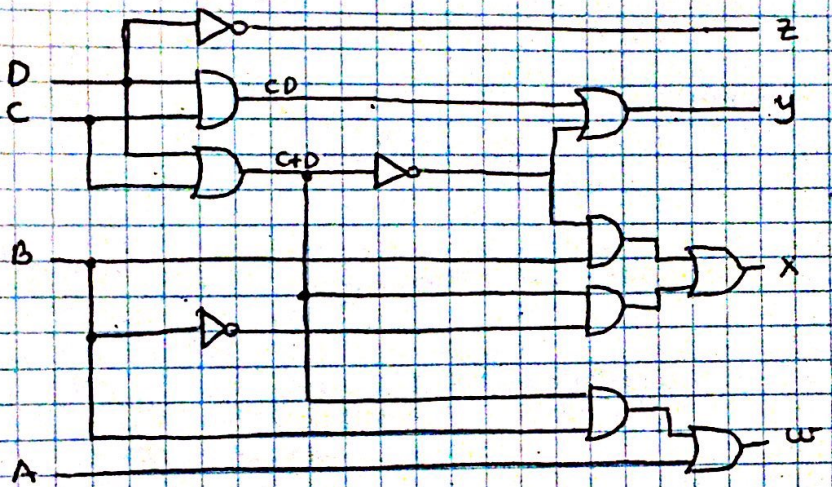

$z = D'$

$\therefore\ z = D'$

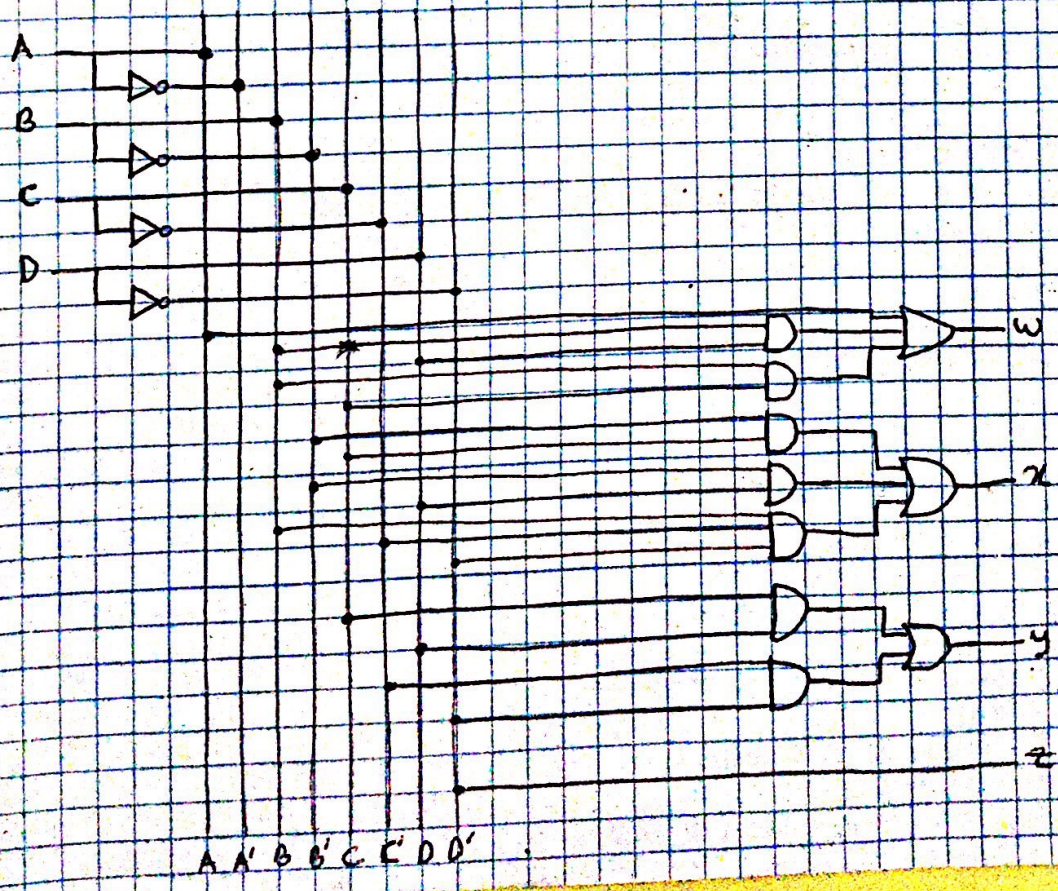
$y = CD + C'D' = CD + (C+D)'$

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

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




or simply;




Scanned with CamScanner

## Analysis Procedure:

The analysis of a combinational circuit starts with a given logic diagram and ends with a set of Boolean functions, a truth table, or a verbal explanation of the circuit operation.

Note that combinational circuits have gates with no feedback or memory elements.

Steps of the analysis:

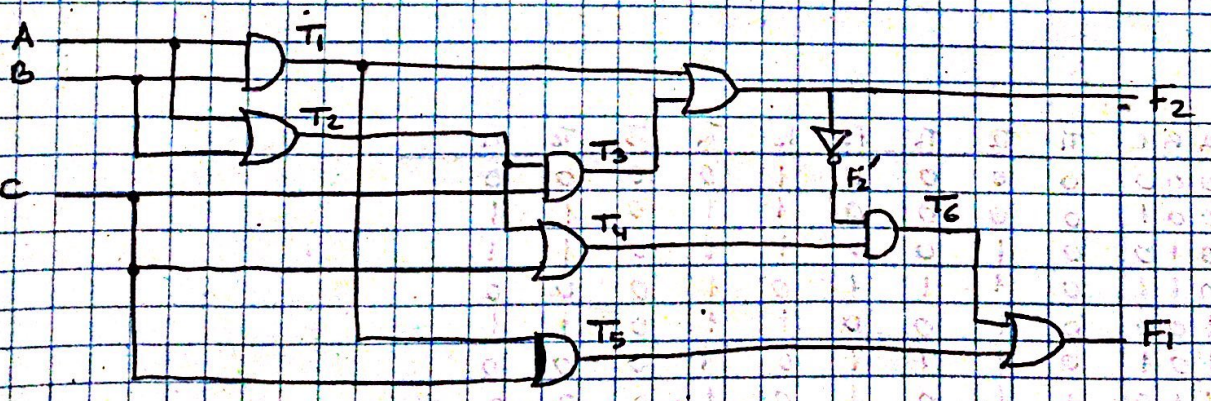
1- Label with arbitrary symbols all gate outputs that are a function of only the input variables. Obtain the Boolean functions for each gate.

2- Label with other arbitrary symbols those gates that are a function of input variables and/or previously labeled gates. Find the Boolean functions of these gates.

3- Repeat step2 until you reach the outputs of the circuit.

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

**Example:** For the following circuit, find the Boolean functions for the two outputs as a function of the inputs and explain the circuit operation. Derive the truth table of the circuit also.




$T_1 = AB$ ; $T_2 = A+B$ ; $T_3 = T_2 \cdot C$ ; $T_4 = T_2 + C$ ; $T_5 = T_1 \cdot C$ ; $T_6 = F_2' \cdot T_4$

$F_2 = T_1 + T_3 = AB + T_2 \cdot C = AB + (A+B)C = AB + AC + BC$

$F_1 = T_5 + T_6 = T_1 \cdot C + F_2' \cdot T_4 = ABC + (AB+AC+BC)' \cdot (T_2 + C)$

$= ABC + (A'+B')(A'+C')(B'+C')(A+B+C) = ABC + AB'C' + A'BC' + A'B'C$

Scanned with CamScanner

| A | B | C | $T_1$ | $T_2$ | $T_3$ | $F_2$ | $F_2'$ | $T_4$ | $T_5$ | $T_6$ | $F_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

↑ Carry output

↑ Sum output

⇒ This circuit is a full-adder with $F_1$ being the Sum and $F_2$ the carry output.

Scanned with CamScanner

## Exclusive-OR (Revisited)

Remember that $x \oplus y = xy' + x'y$

$x \odot y = xy + x'y' = (x \oplus y)'$

$\Rightarrow$ $x \oplus 0 = x$ since $x \cdot 1 + x' \cdot 0 = x + 0 = x$

$x \oplus 1 = x'$ since $x \cdot 0 + x' \cdot 1 = 0 + x' = x'$

$x \oplus x = 0$ since $x \cdot x' + x' \cdot x = 0 + 0 = 0$

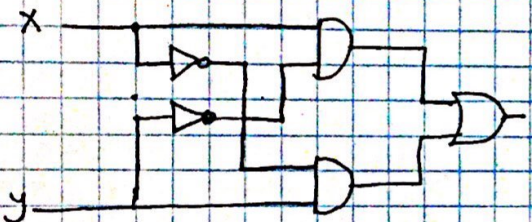
$x \oplus x' = 1$ since $x \cdot x + x' \cdot x' = x + x' = 1$

$x \oplus y' = (x \oplus y)'$ since $x \oplus y' = x \cdot y + x'y' = x \odot y = (x \oplus y)'$

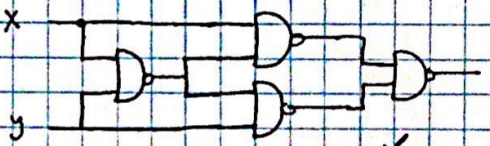
$x' \oplus y = (x \oplus y)'$ since $x' \oplus y = x' \cdot y' + xy = x \odot y = (x \oplus y)'$

$x \oplus y = y \oplus x$ ; $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$ ← Associativity

Implementation:




;

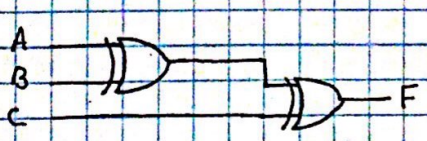



$F = \left( (xy)' \cdot x \right)' \cdot \left( (xy)' \cdot y \right)'$

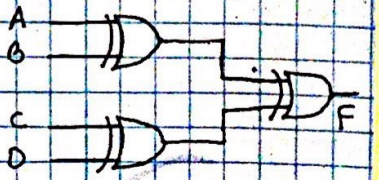
$= (xy)' \cdot x + (xy)' \cdot y$

$= (x' + y') \cdot x + (x' + y') y = xy' + x'y = x \oplus y$

$A \oplus B \oplus C$




$A \oplus B \oplus C \oplus D$

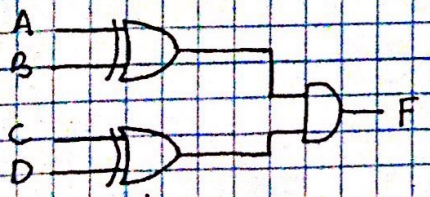



**Example:** Implement the function

$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$ using exclusive-OR and AND gates only.

$\Rightarrow F = (AB' + A'B) CD' + (AB' + A'B) C'D$

$= (A \oplus B) CD' + (A \oplus B) C'D = (A \oplus B)(CD' + C'D) = (A \oplus B)(C \oplus D)$




Scanned with CamScanner

## MULTILEVEL NAND CIRCUITS: (NAND implementation Revisited!)

To obtain a NAND diagram from a Boolean expression:

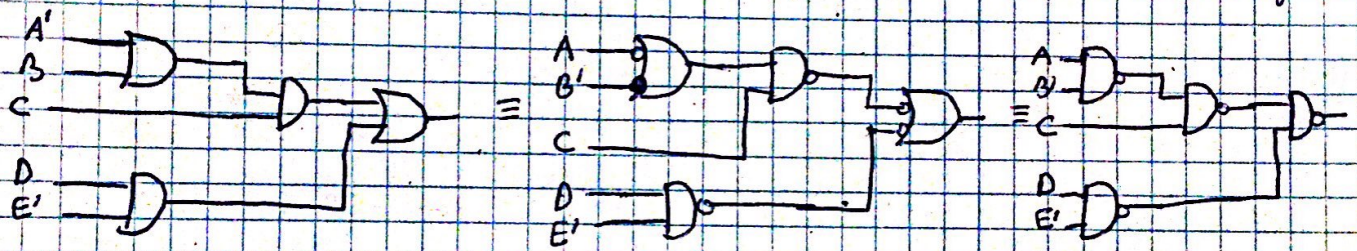
1). Using the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume that both normal and complemented inputs are available.

2). Convert all AND gates to NAND gates with AND-invert graphic symbols.

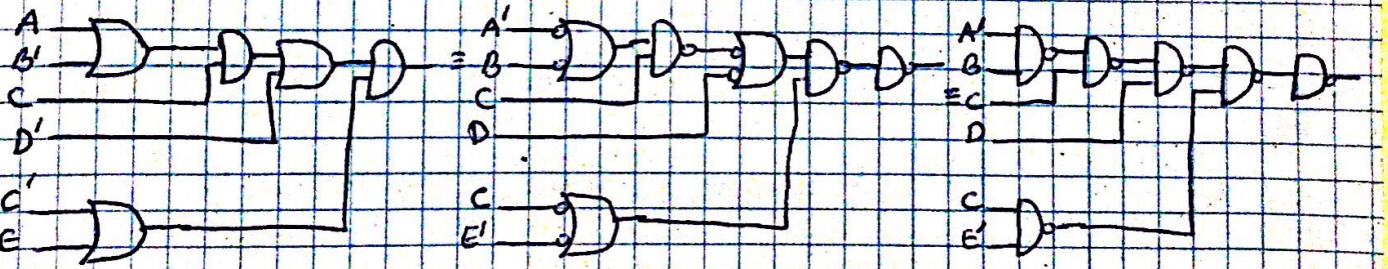
3). Convert all OR gates to NAND gates with Negative-OR graphic symbols.

4) Check all small circles in the diagram. For every that is not compensated by another along the same line, insert an inverter (one-input NAND) or complement the input variable.

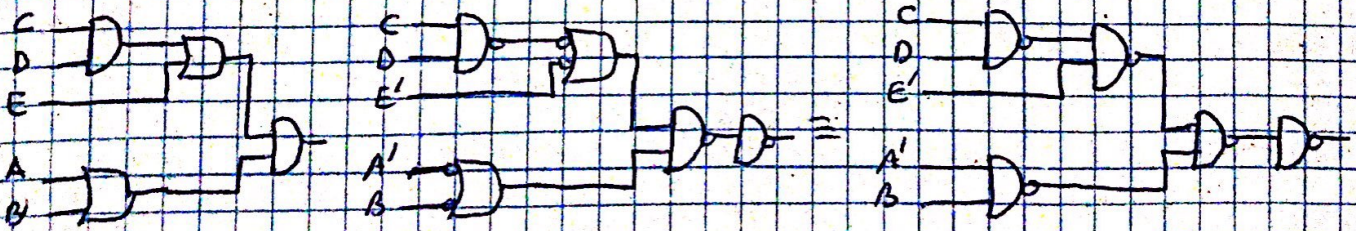
Example: Implement the function $F = (A'+B)C + DE'$ with NAND gates.



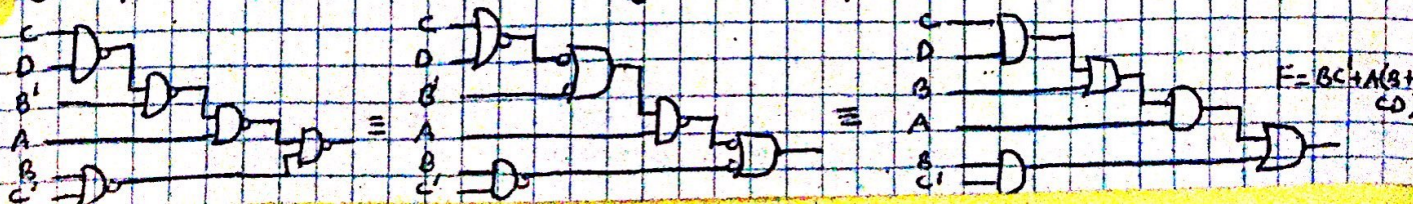

Example: Implement the function $F = ((A+B')C+D')(C'+E)$




Example: Implement the function $F = (CD+E)(A+B')$ with NAND gates

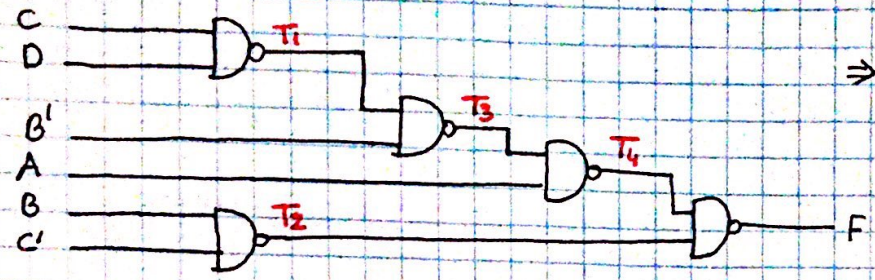



Example: Transform the following Logic diagram to AND-OR




Scanned with CamScanner

Example: Derive the following Boolean function F.




$\Rightarrow T_1 = (CD)'$
$T_2 = (BC')'$
$T_3 = (T_1 \cdot B')'$
$= T_1' + B$
$= CD + B$

$T_4 = (A \cdot T_3)'$
$= A' + T_3'$
$= A' + (CD + B)$

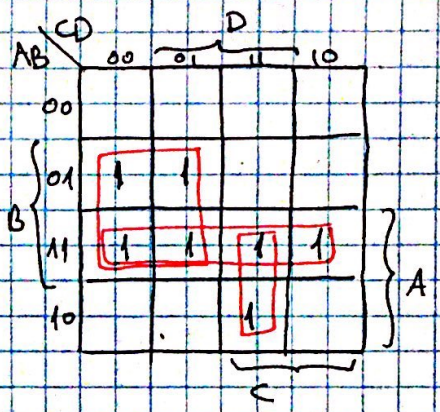
$\therefore F = (T_2 \cdot T_4)' = T_2' + T_4' = BC' + A \cdot T_3 = BC' + A(CD + B)$

$= BC' + AB + ACD$

Truth table:

| A | B | C | D | $T_1$ | $T_2$ | $T_3$ | $T_4$ | F |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

$\Rightarrow$

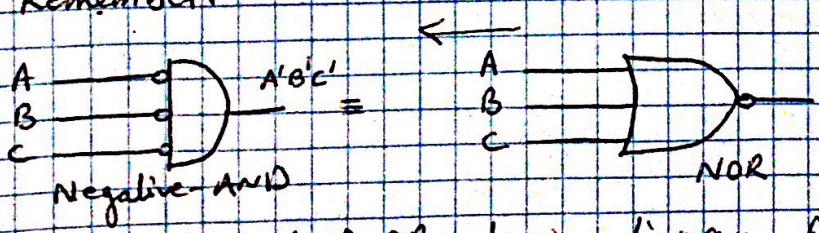



$\Rightarrow F = AB + BC' + ACD$

## MULTILEVEL NOR CIRCUITS:

Remember that the NOR function is the dual of the NAND.

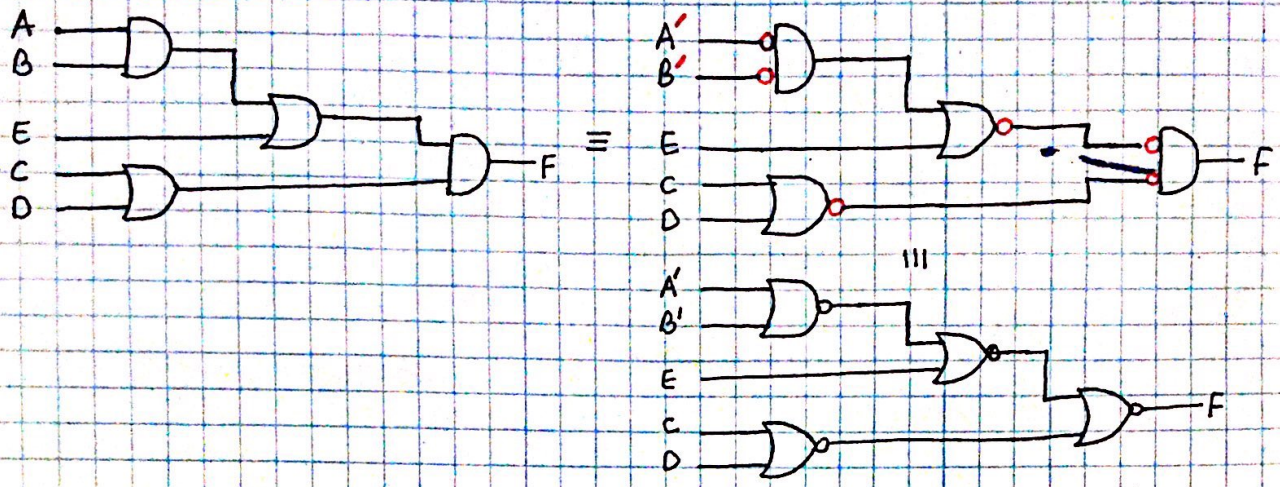
Remember:



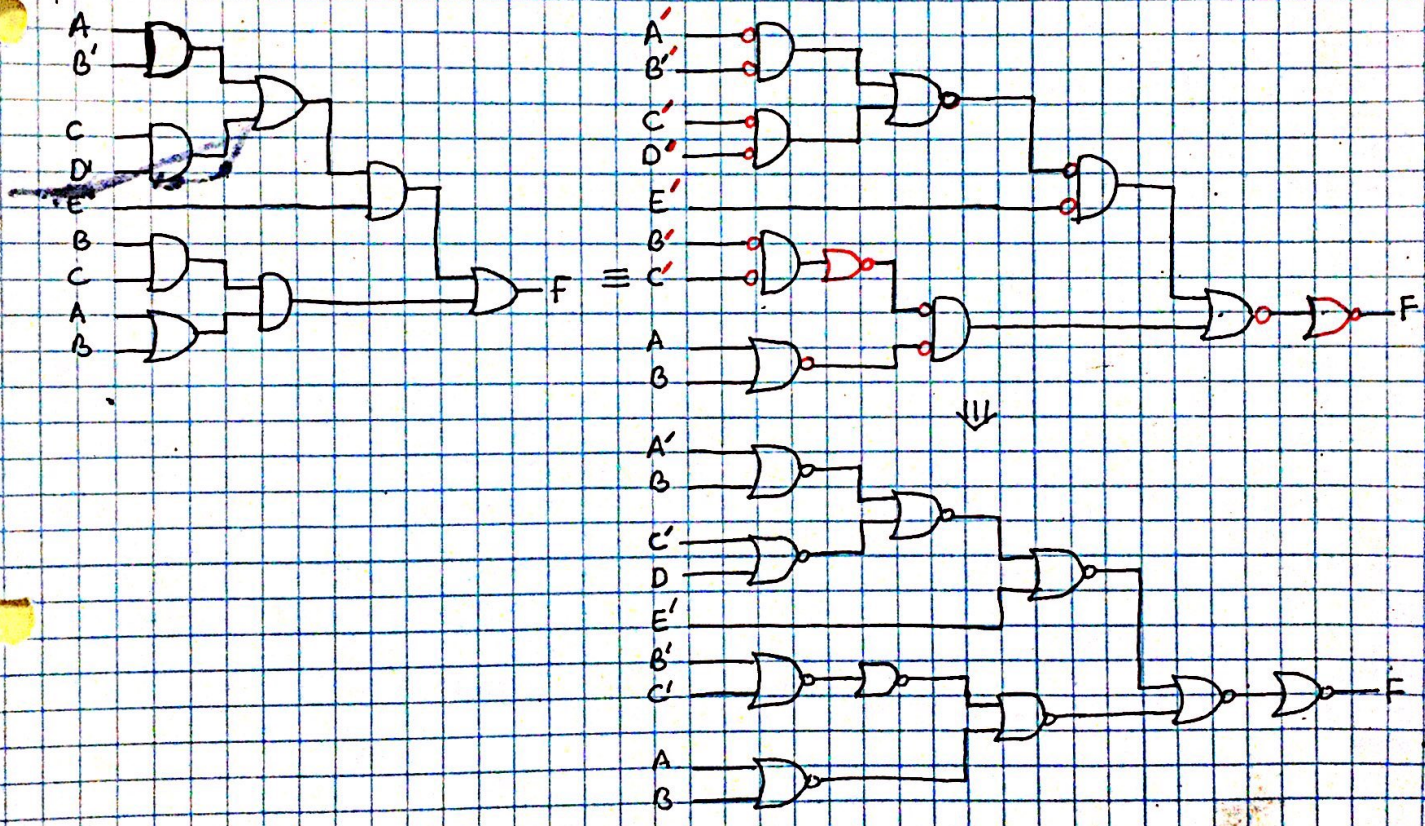

1. Draw the AND-OR logic diagram for the given expression. Assume that both the normal and complement inputs are available.
2. Convert all OR gates to NOR gates.
3. Convert all AND gates to NOR gates.
4. Any small circle that is not compensated by another small circle along the same line needs an inverter or the input variable can be complemented.

Scanned with CamScanner

Example: Implement the function $F = (AB+E)(C+D)$ with NOR gates.

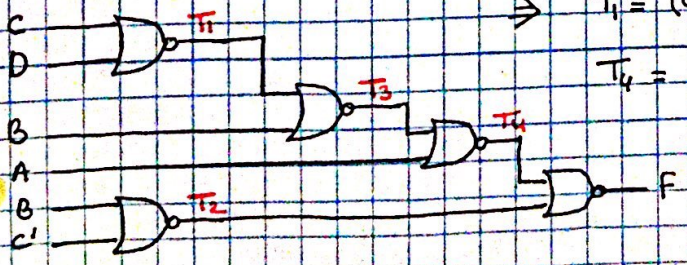



Example: Implement the function $F = (AB'+CD')E + BC(A+B)$ with NOR.




Analysis Procedure of NOR circuits:

Determine the Boolean function F in the following circuit




$\Rightarrow$ $T_1 = (C+D)'$ ; $T_2 = (B+C')'$ ; $T_3 = (T_1+B)'$

$T_4 = (T_3+A)'$

$F = (T_2+T_4)' = T_2' \cdot T_4'$

$= (B+C')(T_3+A) = (B+C')(T_1' \cdot B' + A)$

$= (B+C')((C+D) \cdot B' + A)$

$= (B+C')(A+B') \cdot (A+C+D)$

Scanned with CamScanner

## Parity Generation and checking:

A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. It is used for detecting one bit errors. The parity bit is generated by the transmitter and checked by the receiver.

Example: Consider a 3-bit message transmittion with even parity.

| 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 |

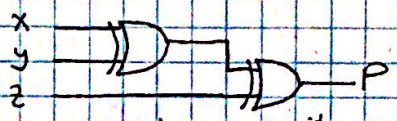
| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | 1 | | 1 |
| 1 | 1 | | 1 | |

$\Rightarrow P = x'y'z + x'yz' + xy'z' + xyz$

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

$= x'(y \oplus z) + x(y \oplus z)'$

$= x \oplus y \oplus z$



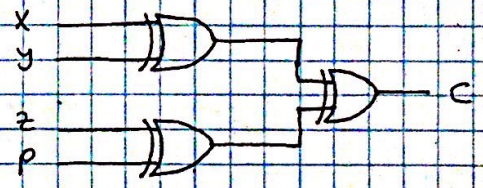

3-bit even parity generator.

$\Rightarrow$ at the receiver site, the four bits received must have an even number of 1's. In case of error the output of the parity checker is 1.

| 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 |

| xy \ zP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | | 1 |
| 01 | 1 | | 1 | |
| 11 | | 1 | | 1 |
| 10 | 1 | | 1 | |

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

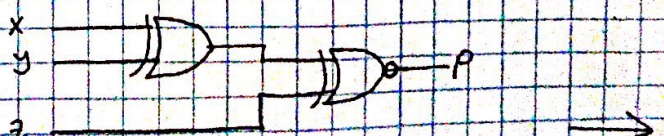



Example: Design a 3-bit odd parity generator and a 4-bit odd parity checker using Exclusive-OR gats.

| x | y | z | P |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | | | 1 |
| 1 | | 1 | | |

$\Rightarrow P = (x \oplus y \oplus z)'$




Scanned with CamScanner