Unit-4
Combinational Logic

Unit-4
Combinational Logic

Combinational circuit is a circuit which consist of logic gates whose outputs at any instant of time are determined directly from the present combination of inputs without regard to previous input. The combinational circuit do not use any memory.

- There will be 2^n combination of input variable for n inputs.
- A combinational circuit can have n number of inputs and m number of outputs.
- For e.g. adders, subtractors, decoders, encoders etc.

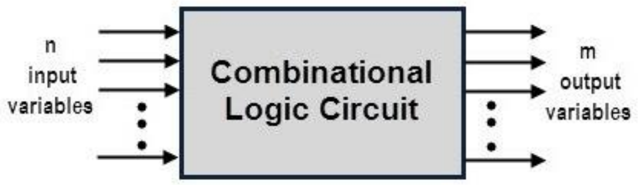


Fig: Block diagram of combinational circuit

Combinational logic circuit design procedure:

1. The problem is stated.
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationships between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

Adders

Adders are the combinational circuits which is used to add two or more than two bits at a time.

Types of adders:
- Half Adder
- Full Adder

1. Half Adder:

A combinational circuit that performs the addition of bits is called half adder. This circuit needs two binary inputs and two binary outputs. The input variables designate the augend(A) and addend(B) bits; the output variables produce the sum(S) and carry(C).

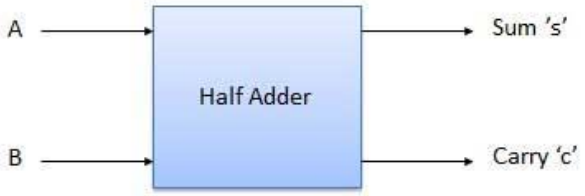
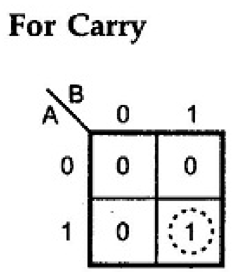


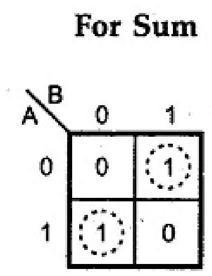
Fig: Block diagram

Truth table to identify the function of half adder:

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map:



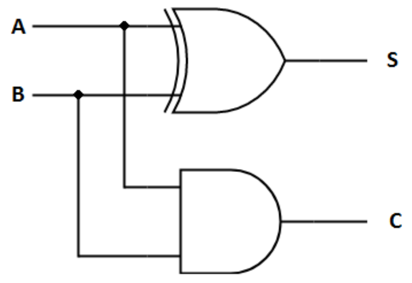


From k-map the logical expression for sum and carry is:

$C = AB$

$S = \bar{A}B + A\bar{B} = A \oplus B$

Logic diagram:



Q. Design a half adder using only NAND gates.

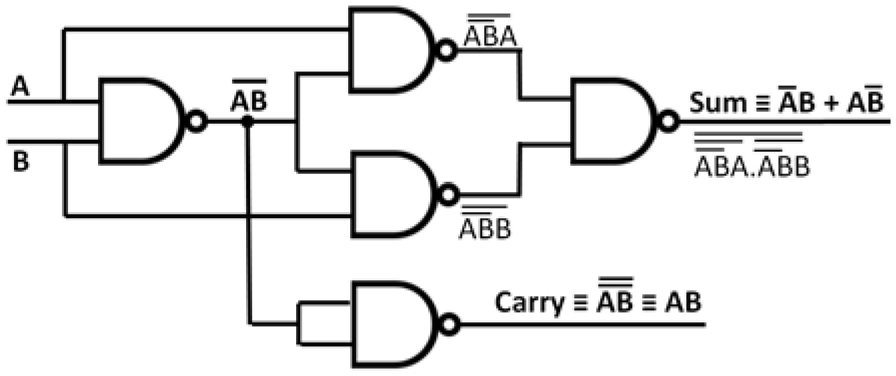
Solⁿ:

Input variables: A & B, Output variables: sum(S) and carry(C)

$S = \bar{A}B + A\bar{B}$

$C = AB$

Logic diagram of half adder using NAND gates only:



Q. Design a half adder logic circuit using NOR gates only.

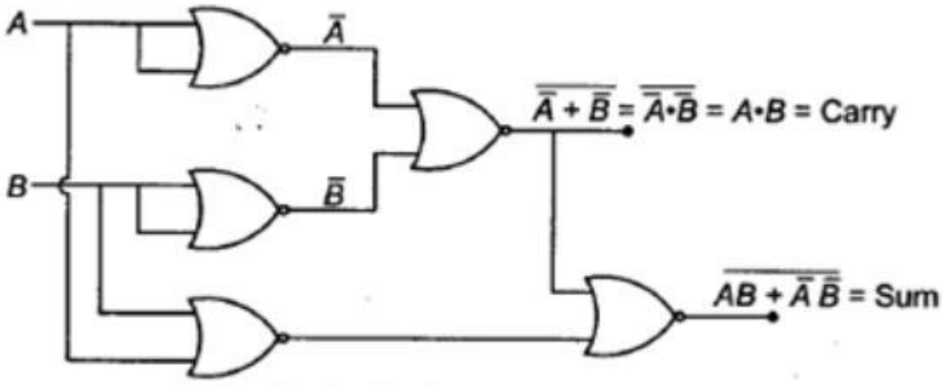
Sol^n:

Input variables: A & B, Output variables: sum(S) and carry(C)

$S = \bar{A}B + A\bar{B}$

$C = AB$

Logic diagram of half adder using NOR gates only:


A
B
$\bar{A}$
$\bar{B}$
$\overline{\bar{A}+\bar{B}} = \overline{\bar{A}} \cdot \overline{\bar{B}} = A \cdot B = \text{Carry}$
$\overline{AB + \bar{A}\bar{B}} = \text{Sum}$

Half adder logic diagram using NOR gate

2. **<u>Full Adder</u>**:

A combinational circuit that performs the addition of three bits at a time is called full adder. It consists of three inputs and two outputs, two inputs are the bits to be added, the third input represents the carry from the previous position.

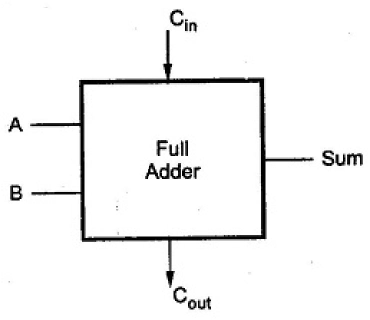


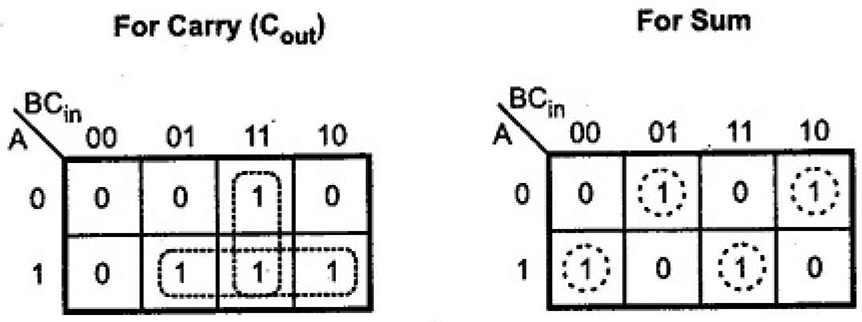
Fig: Block diagram

Truth table for full adder:

Input			Output	
A	B	Cin	Sum	Carry
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
1	1	1	1	1

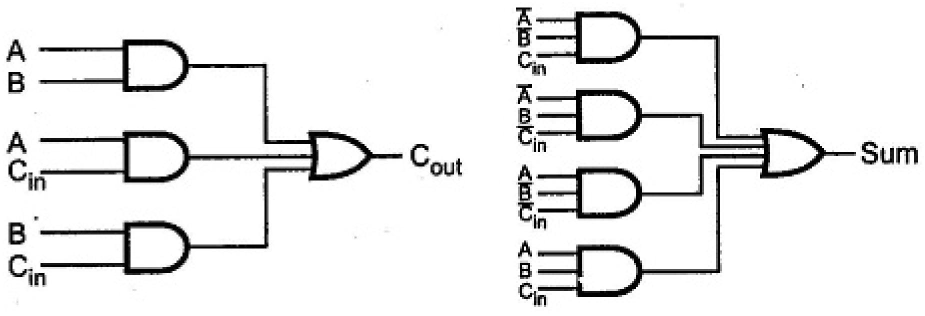
- The sum(S) output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1.
- The carry output (C_{out}) has a carry 1 if two or three inputs are equal to 1.

Simplified expression using k-map in SOP can be obtained as;



$$Sum(S) = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Logic Diagram:

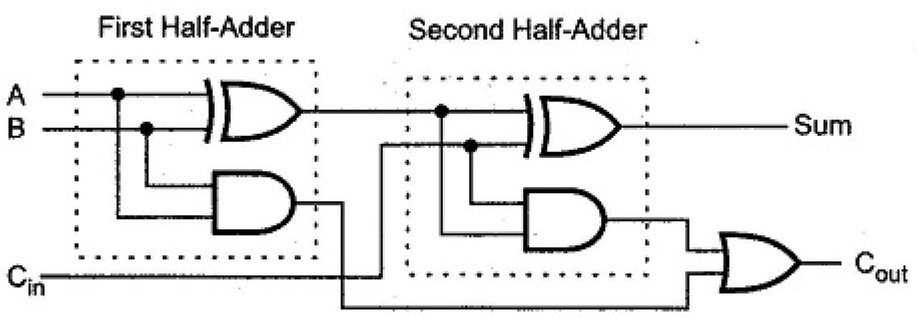


<u>For carry</u> <u>For sum</u>

Fig: SOP implementation of full-adder

Note: *It can also be implemented in POS form. (Try yourself)*

<u>Implementation of a full-adder with two half-adders and an OR gate:</u>



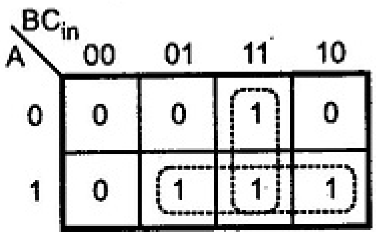
Logic expression for sum:

$$(A\oplus B)\oplus C_{in} = (\bar{A}B + A\bar{B})\oplus C_{in} = \left(\overline{\bar{A}B + A\bar{B}}\right)C_{in} + (\bar{A}B + A\bar{B})\bar{C}_{in}$$
$$= (A + \bar{B})(\bar{A} + B)C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in}$$
$$= \bar{A}\bar{B}C_{in} + ABC_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in}$$

Logic expression for carry:

$$(A\oplus B)C_{in} + AB = (\bar{A}B + A\bar{B})C_{in} + AB = \bar{A}BC_{in} + A\bar{B}C_{in} + AB$$

Simplification of carry:



$\therefore C_{out} = AB + AC_{in} + BC_{in}$

Subtractors

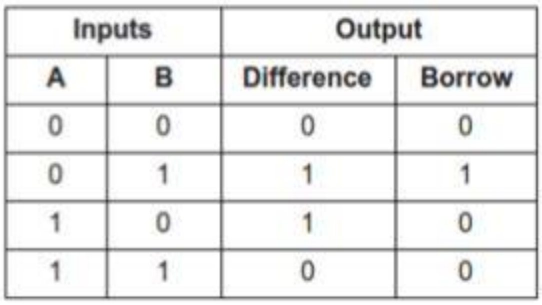
Subtractor is a combinational logic circuit which is used to subtract two or more than two bits at a time, and provides difference and borrow as an output.

Types of Subtractors:

- Half subtractor
- Full subtractor

1. **Half Subtractor:**
   A half-subtractor is a combinational logic circuit that subtract two bits at a time and produces their difference.
   It has two inputs minuend (A) & subtrahend (B) and two outputs difference and borrow. The difference is a result of subtraction and borrow is used to indicate borrow from next most significant bit. The borrow bit is present only when $A < B$.

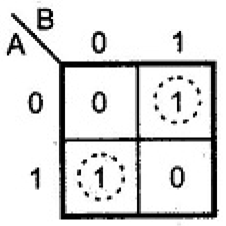
   Truth table:



Inputs		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

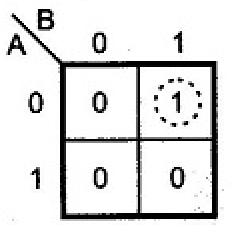
K-map:

For Difference



Difference = $A\bar{B} + \bar{A}B$
= $A \oplus B$

For Borrow



Borrow = $\bar{A}B$

Logic Diagram:

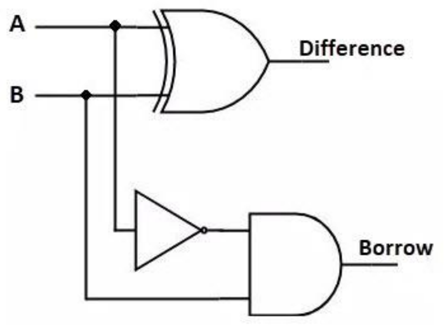
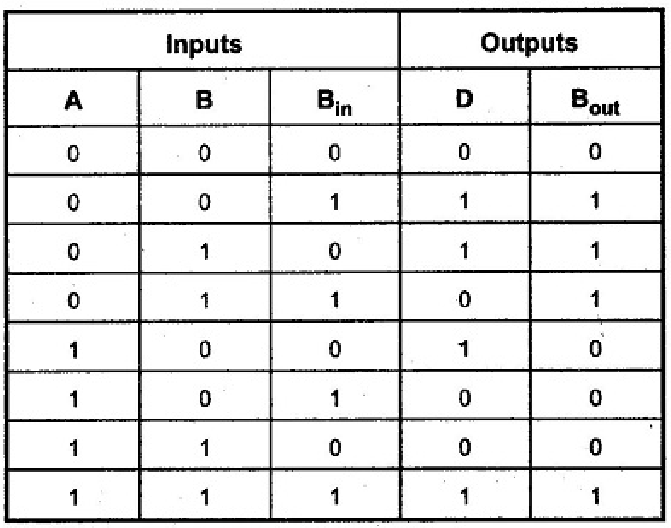


Fig: Implementation of half-subtractor

2. **<u>Full Subtractor</u>:**
   A combinational logic circuit used to subtract three binary digits at a time is called full subtractor.
   This circuit has three input and two outputs. The three inputs are $A, B \text{ and } B_{in}$, denote the minuend, subtrahend and previous borrow respectively. The two outputs, $D \text{ and } B_{out}$ represent the difference and output borrow, respectively.

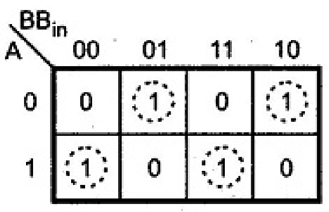
Truth table for full subtractor:



Inputs			Outputs	
A	B	B_{in}	D	B_{out}
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

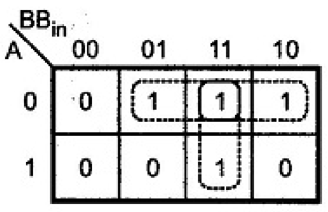
Simplified expression of D and B_{out} using k-map in SOP can be obtained as;

For D



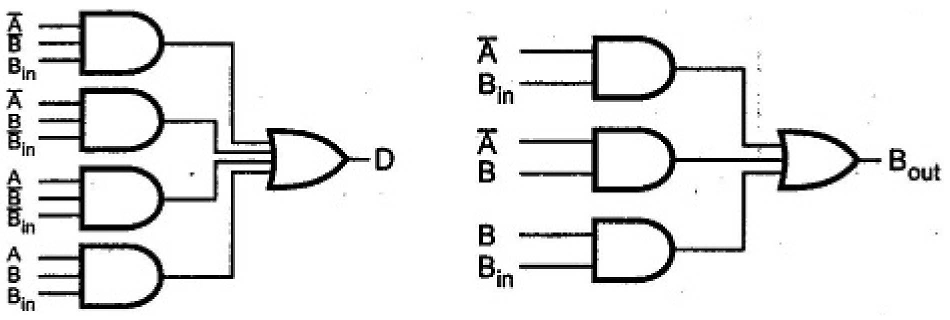
$$D = \overline{A}\,\overline{B}B_{in} + \overline{A}B\overline{B}_{in} + A\overline{B}\,\overline{B}_{in} + ABB_{in}$$

For B_{out}



$$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$$

Logic circuit for full subtractor:

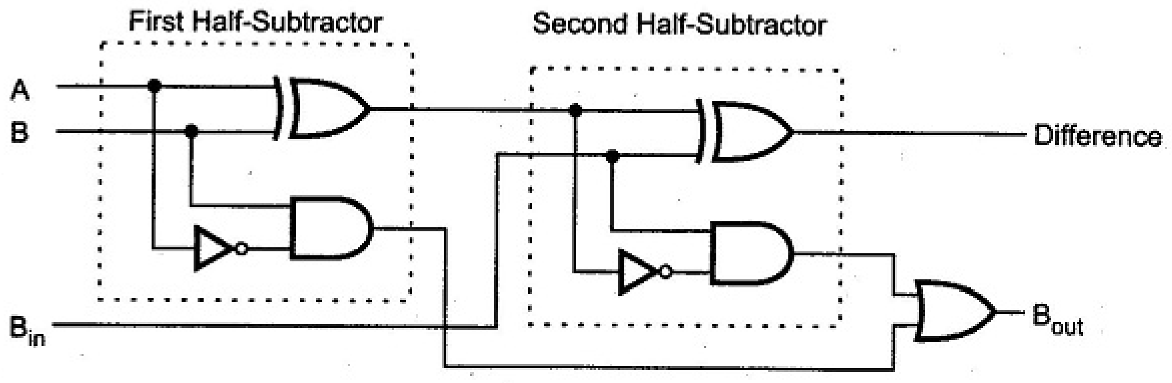


For D For B_{out}

Fig: SOP implementation of full-subtractor

Note: *It can also be implemented in POS form. (Try yourself)*

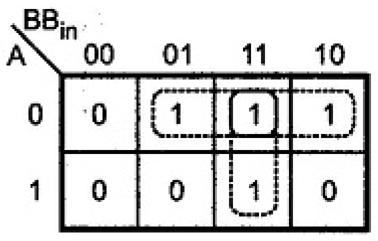
Implementation of full subtractor using two half subtractor and one OR gate:



$$D = (A \oplus B) \oplus B_{in} = (\bar{A}B + A\bar{B}) \oplus B_{in} = \left(\overline{\bar{A}B + A\bar{B}}\right)B_{in} + (\bar{A}B + A\bar{B})\bar{B}_{in}$$
$$= \{(A+\bar{B})(\bar{A}+B)\}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in}$$
$$= \bar{A}\bar{B}B_{in} + ABB_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in}$$

$$B_{out} = (\overline{A \oplus B})B_{in} + \bar{A}B = \left(\overline{\bar{A}B + A\bar{B}}\right)B_{in} + \bar{A}B = \{(A+\bar{B})(\bar{A}+B)\}B_{in} + \bar{A}B$$
$$= \bar{A}\bar{B}B_{in} + ABB_{in} + \bar{A}B$$

Using k-map:



A \ BB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$$\therefore B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$$

Code Conversion

- The availability of large variety of codes for the same discrete elements of information results in the use of different codes by the different system.
- A conversion circuit must be inserted between the two systems if each use different codes for same information.
- Thus a code converter is a circuit that makes the two systems compatible even though each uses a different binary information.

BCD to excess-3 Code Conversion:

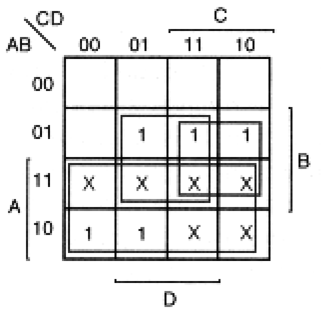
BCD Excess-3 circuit will convert numbers from their binary representation to their excess-3 representation. Since each code uses four bits to represent a decimal digit, there must be four input variables and four outputs variables. Let the input four binary variables are A, B, C & D and the four output variables are W, X, Y & Z.

Truth table:

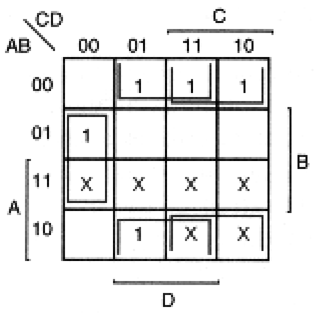
BCD Inputs				excess-3 Outputs			
A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Note: Four binary variables may have 16 bit combinations, and only 10 of which are listed in truth table i.e. from 0 to 9. The rest six bit combinations not listed for input variables are don't care combinations.

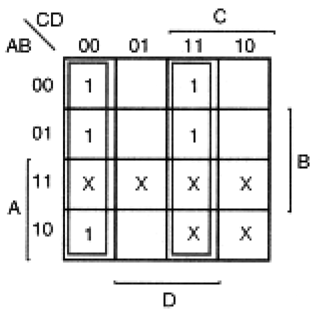
K-maps for BCD to excess-3 code converter:



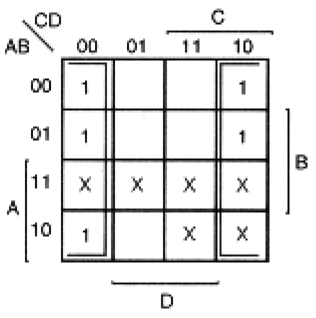
$W = A + BC + BD$



$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$



$Y = CD + \bar{C}\bar{D}$

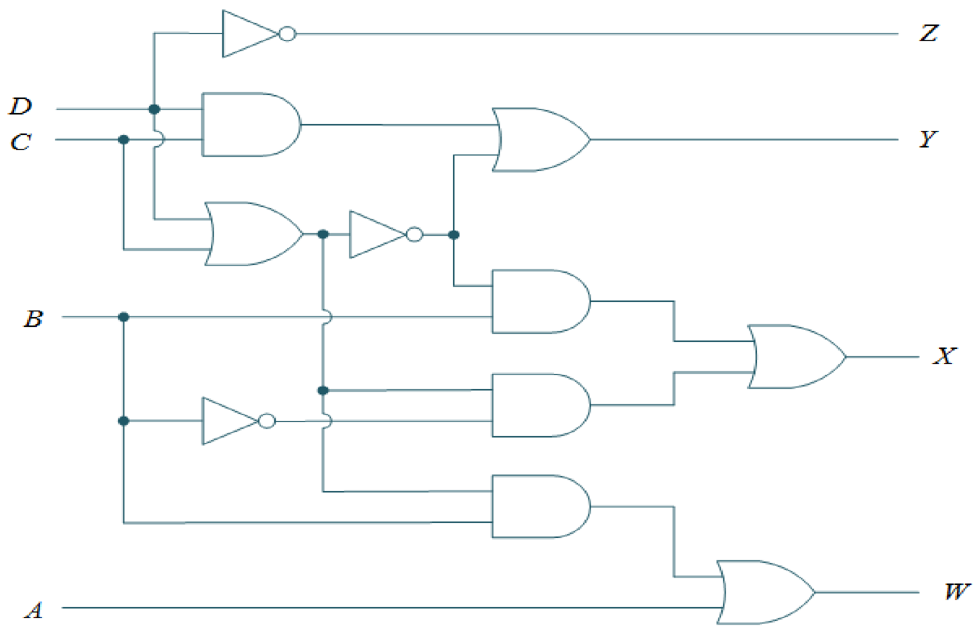


$Z = \bar{D}$

The Boolean functions for the outputs lines of the circuit are derived from k-maps which are:

$W = A + BC + BD = A + B(C + D)$
$X = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$
$Y = CD + C'D' = CD + (C + D)'$
$Z = D'$

Logic diagram for BCD to excess-3 converter:



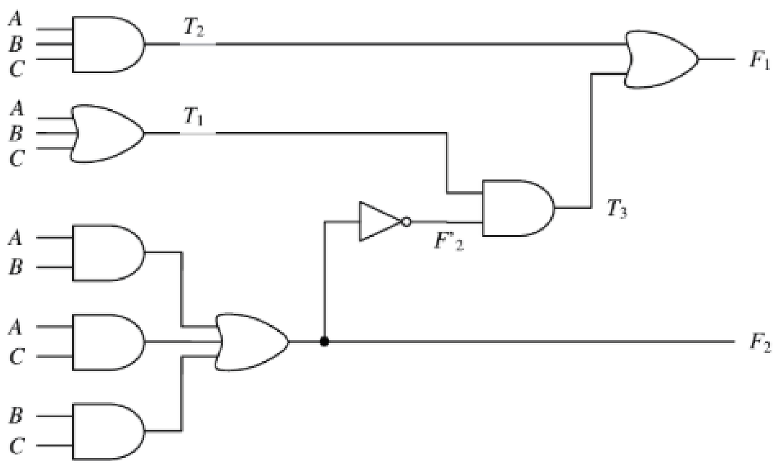
Analysis Procedure

To obtain the Boolean expressions and truth tables from the combinational logic circuit, we need to analyse the circuit. First ensure that the circuit is combinational - that is there is no feedback of an output to an input that the output depends on.

1st step: make sure that circuit is combinational.
2nd step: obtain the output Boolean functions or the truth table.

- **Obtaining Boolean functions from logic diagram:**
  Steps:
  1. Label all gate outputs that are a function only of input variables or their complements with arbitrary symbols. Determine the Boolean functions for each gate output.
  2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for the outputs of 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.

A straight-forward procedure:



Step 1:

$F_2 = AB + AC + BC$

$T_1 = A + B + C$

$T_2 = ABC$

Step 2 & 3:

$T_3 = F_2'T_1$

$F_1 = T_3 + T_2$

Step 4:

$$\begin{aligned} F_1 = T_3 + T_1 = F_2'T_1 + ABC &= (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

- **Obtaining truth table from logic diagram:**

  Steps:

  1. Determine the number of input variables in the circuit. For n inputs, list the binary numbers from 0 to $2^n - 1$ in a table.
  2. Label the output of selected gates.
  3. Obtain the truth table for the output 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, until all outputs are determined.

Truth table for the above 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	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 Circuit

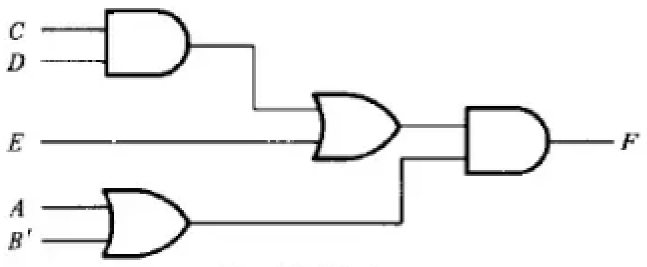
To implement a Boolean function with NAND gates we need to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic. The conversion of an algebraic expression from AND, OR, and complement to NAND can be done by simple circuit-manipulation techniques 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 symbols.
4. Check all small circles in the diagram. For every small circle that is not compensated by another small circle along the same line, insert an inverter (one-input NAND gate) or complement the input variable.

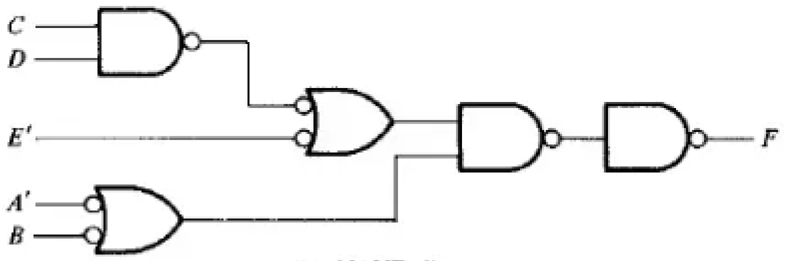
E.g.

Multilevel Boolean expression: $F = (CD + E)(A + B')$



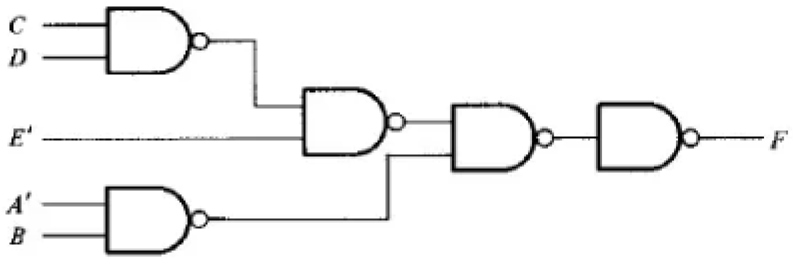
(a) AND-OR diagram

→ NAND diagram using two graphic symbols:



(b) NAND diagram

→ NAND diagram using one graphic symbol:



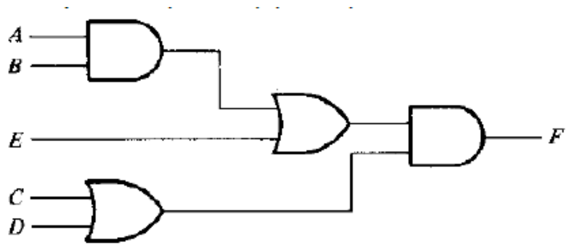
(c) Alternate NAND diagram

Multi-level NOR Circuit

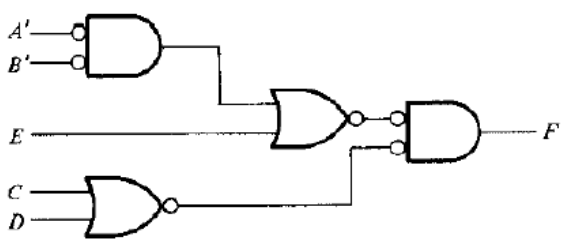
The NOR function is the dual of the NAND function. For this reason, all procedures and rules for NOR logic form a dual of the corresponding procedures and rules developed for NAND logic.

Boolean function implementation using NOR gate:
1. Draw the AND-OR logic diagram from the given algebraic expression. Assume that both the normal and complement inputs are available.
2. Convert all OR gates to NOR gates with OR-invert graphic symbols.
3. Convert all AND gates to NOR gates with invert-AND graphic symbols.
4. Any small circle that is not compensated by another small circle along the same line needs an inverter or the complementation of the input variable.

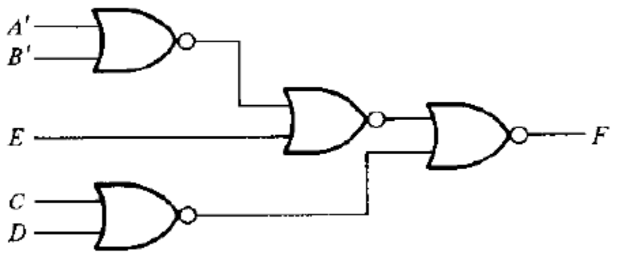
E.g.
$F = (AB + E)(C + D)$



(a) AND-OR diagram



(b) NOR diagram



(c) Alternate NOR diagram

Fig: Implementing $F = (AB + E)(C + D)$ with NOR gates

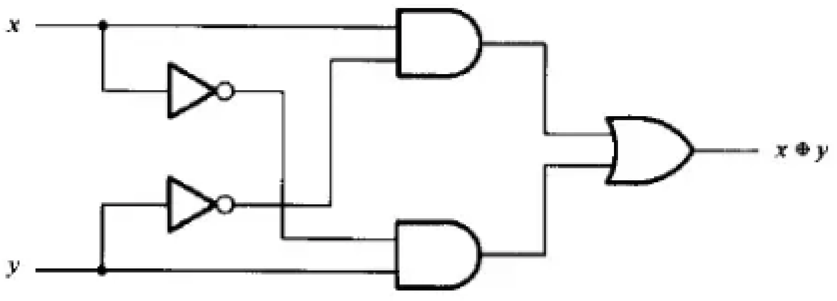
Exclusive-OR (XOR)

The exclusive-OR (XOR) denoted by the symbol ⊕ is a logical operation that performs the following Boolean operation:

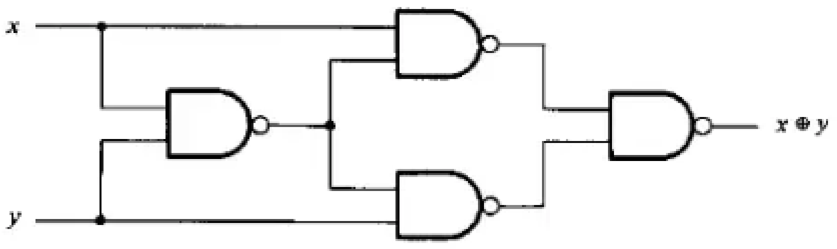
$$x \oplus y = xy' + x'y$$

It is equal to 1 if only x is equal to 1 or if only y is equal to 1 but not when both are equal to 1.

Realization of Ex-OR using basic gates and universal gates:



(a) With AND-OR-NOT gates



(b) With NAND gates

<u>Parity Generator and Checker</u>

Parity Generator:

A parity generator is a combinational logic circuit that generates the parity bit in the transmitter.

- A parity bit is used for the purpose of detecting errors during transmission of binary information. It is an extra bit included with a binary message to make the number of 1's either odd or even.
- Types of parity: Even parity & Odd parity.
- In Even parity, added parity bit will make the total number of 1's an even amount.
- In Odd parity, added parity bit will make the total number of 1's an odd amount.

3-bit even parity generator truth table:

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

Solving the truth table for all the cases where *P* is 1 using SOP method:

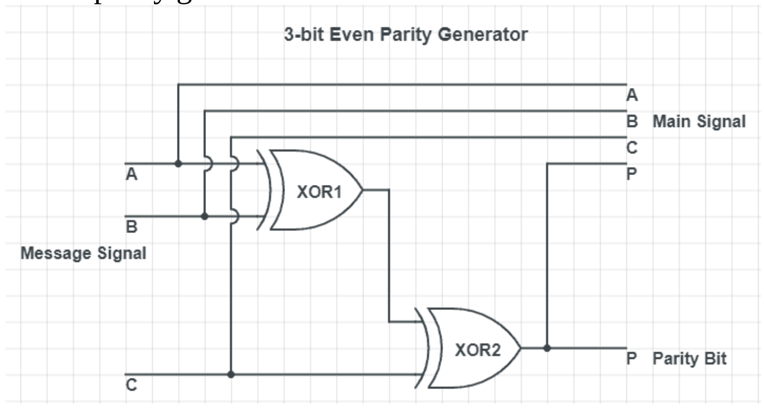
$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

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

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

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

3-bit even parity generator circuit:



Parity checker:
A circuit that checks the parity in the receiver is called parity checker. The parity checker circuit checks for possible errors in the transmission.

- Since the information transmitted with even parity, the received must have an even number of 1's. If it has odd number of 1's, it indicates that there is an error occurred during transmission.

3-bit even parity checker truth table;

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 output of the parity checker is denoted by PEC (Parity Error Checker). If there is error, that is, if it has odd number of 1's, it will indicate 1. If no then PEC will indicate 0.

$$PEC = \bar{A}\,\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(\bar{C}\bar{D} + CD) + AB(\bar{C}D + C\bar{D}) + A\bar{B}(\bar{C}\bar{D} + CD)$$

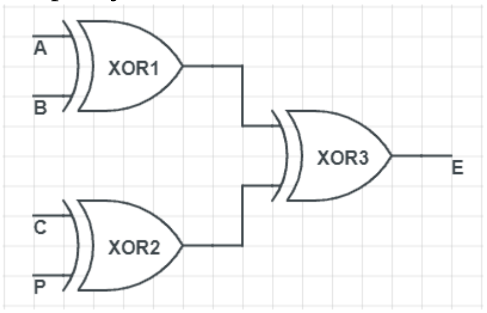
$$= \bar{A}\,\bar{B}(C \oplus D) + \bar{A}B(\overline{C \oplus D}) + AB(C \oplus D) + A\bar{B}(\overline{C \oplus D})$$

(Here truth table's $P = D$)

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

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

3-bit even parity checker circuit:



Q. *Design a combinational circuit that multiplies 2-bit numbers, a_1a_0 and b_1b_0 to produce a 4-bit product, $c_3c_2c_1c_0$. Use AND gates and half-adders.*

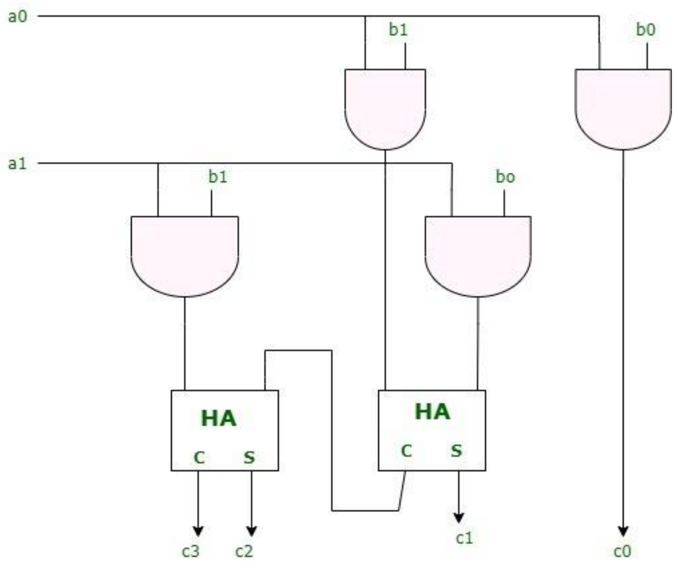
Sol^n^:

Here's how multiplication would take place:

$$\begin{array}{cccc}
 & & a_1 & a_0 \\
 & & b_1 & b_0 \\
\hline
 & & a_1b_0 & a_0b_0 \\
 & a_1b_1 & a_0b_1 & \\
\hline
c_3 & c_2 & c_1 & c_0
\end{array}$$

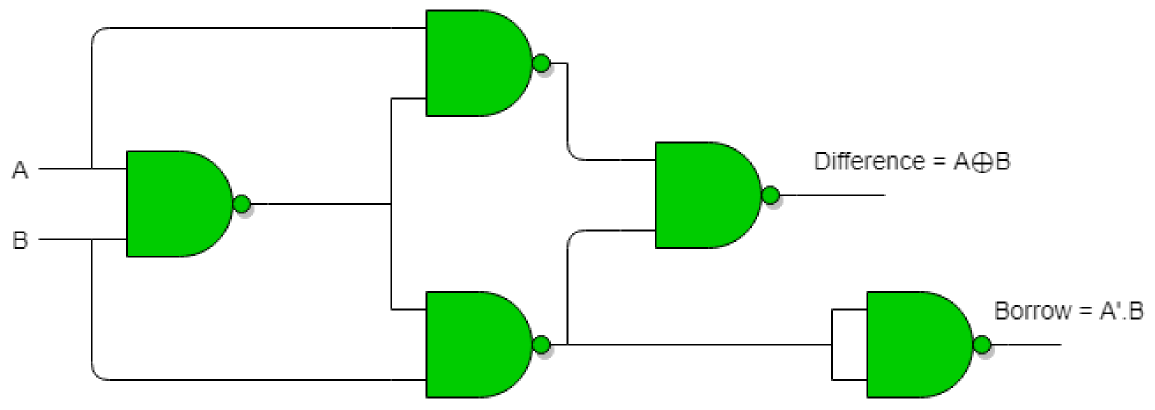
In the above calculation a_1a_0 is the multiplicand and b_1b_0 is the multiplier. The first product obtained from multiplying b_0 with the multiplicand is called as partial product 1. And the second product obtained from multiplying b_1 with the multiplicand is known as the partial product 2.

Based on the above equation, we can see that we need four AND gates and two half adders. The AND gate will performs the multiplication, and the half adders will add the partial product terms. Hence the circuit obtained is as follows:

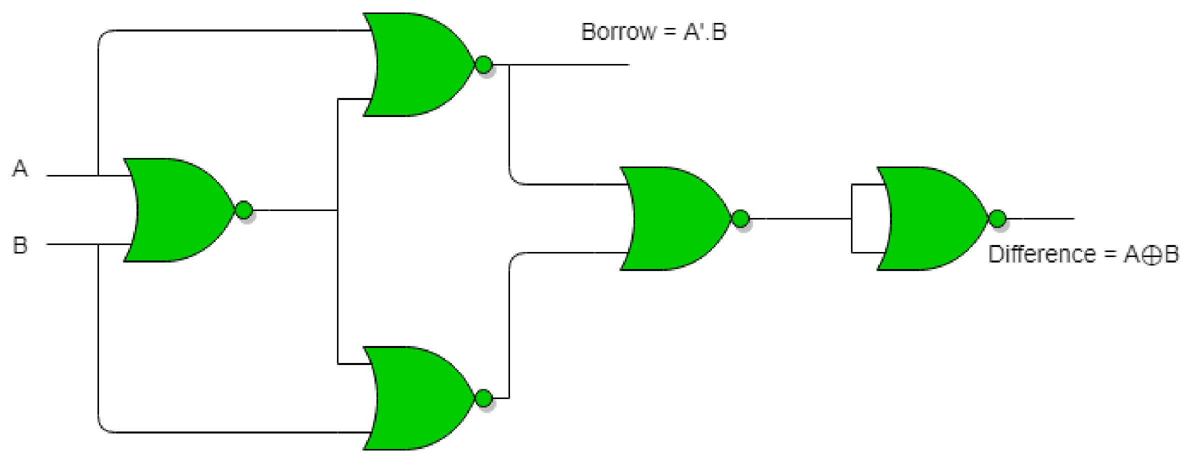


Implementation of half subtractor using universal gate:

Using NAND gate only:



Using NOR gate only:



Reference:

M. Morris Mano, "Digital Logic & Computer Design"