5. Combinational Circuit

5.1 Design Procedure

5.2 Adders and Subtractors

5.3 Code conversion

5.4 Analysis Procedure

5.5 Multilevel NAND and NOR circuits

5.6 Parity generation and checking

Combinational circuit

- The output of combinational circuit at any instant of time depends only on the present input terminals.
- It does not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have ‘n’ number of inputs, logic gates and ‘m’ number of outputs.
- The logic gates accept signal from input variables and generate output signals.

Examples:

- ✓ Adders/Subtractor
- ✓ Encoders/Decoders
- ✓ Multiplexer/De-multiplexer

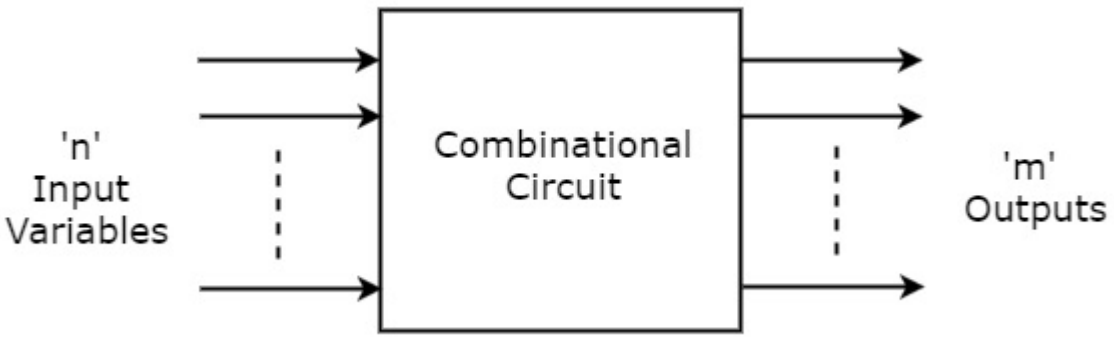


Figure: Combinational Circuit

5.1 Design Procedures

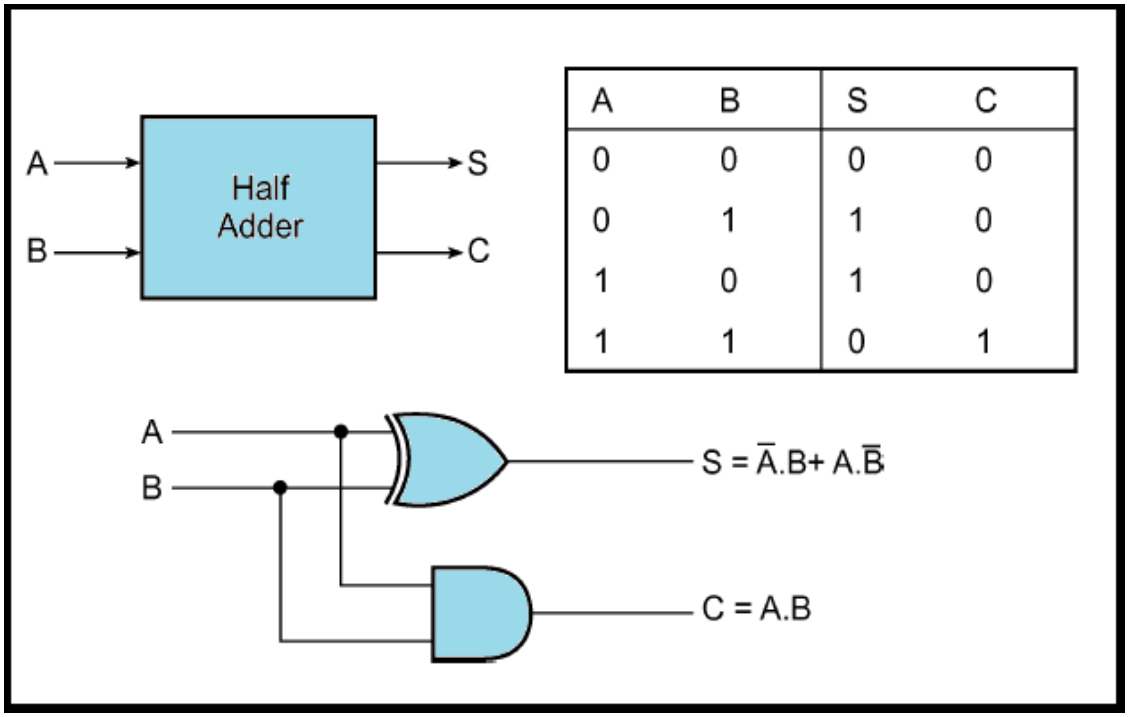
The design procedure involves the following steps:

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 relationship between inputs and outputs is derived.
5) The logic diagram is drawn.

5.2 Adders and Subtractors

Half-Adder

- A combinational circuit that performs addition of two single bit numbers.
- The circuit has two input variables A and B and two outputs: sum(S) and carry(C).

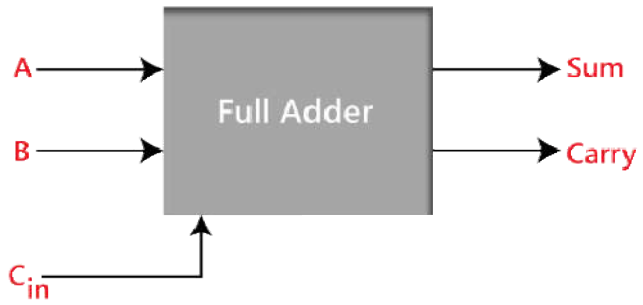


A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure: Block diagram, truth table and circuit diagram of half adder

Full-Adder

- A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is called a full adder.
- Two of the inputs, denoted by A and B, represent the two bits to be added and third input C_{in} represent the carry from the previous lower significant position.



Inputs			Outputs	
A	B	C_{in}	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

$$\text{Sum}(S) = A'B'C + A'BC' + AB'C' + ABC$$
$$= A'(B'C + BC') + A(B'C' + BC)$$
$$= A'(B \oplus C) + A(B \oplus C)'$$
$$= A \oplus B \oplus C$$

$$\text{Carry Out}(Co) = A'BC + AB'C + ABC' + ABC$$
$$= C(A'B + AB') + AB(C + C')$$
$$= (A \oplus B)C + AB$$

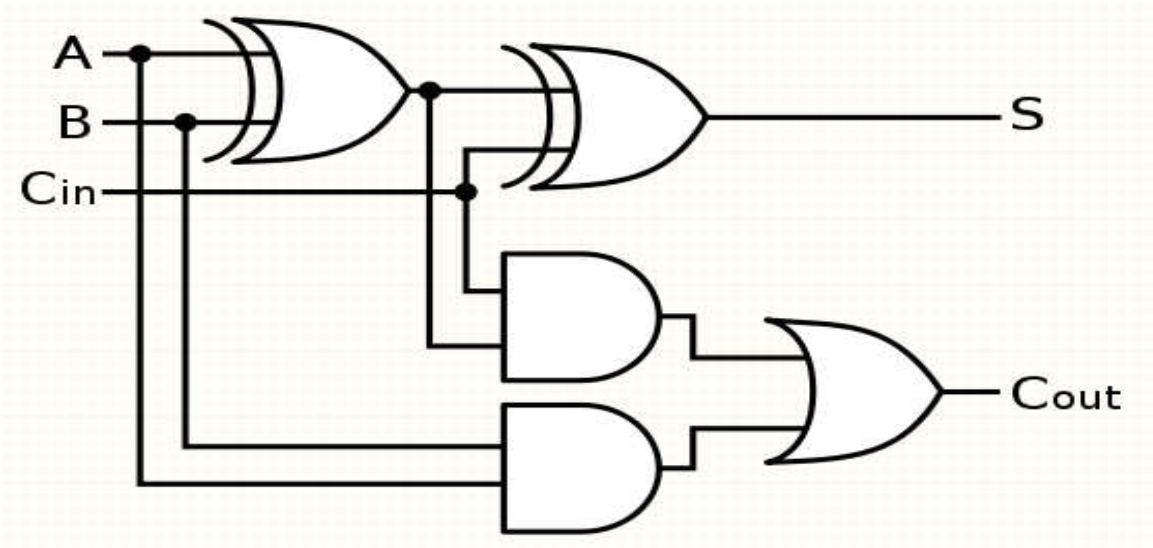
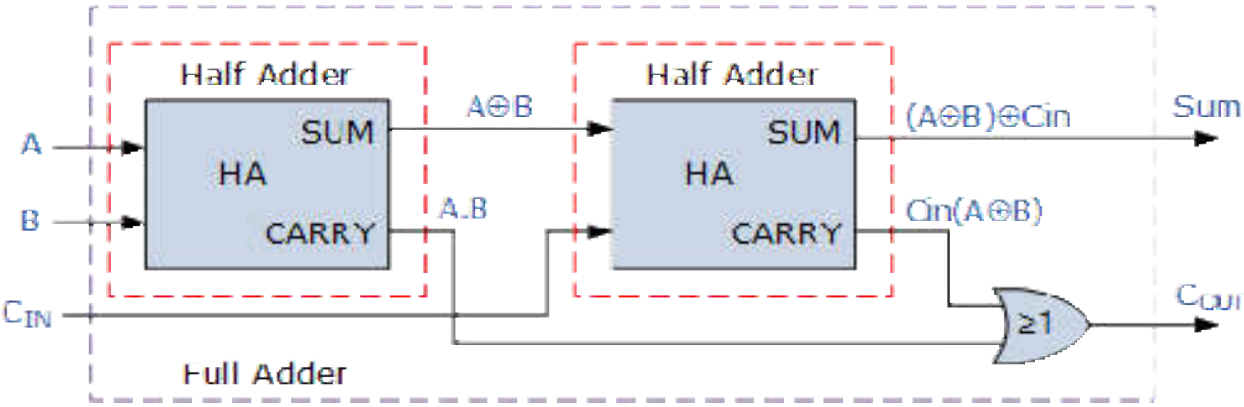
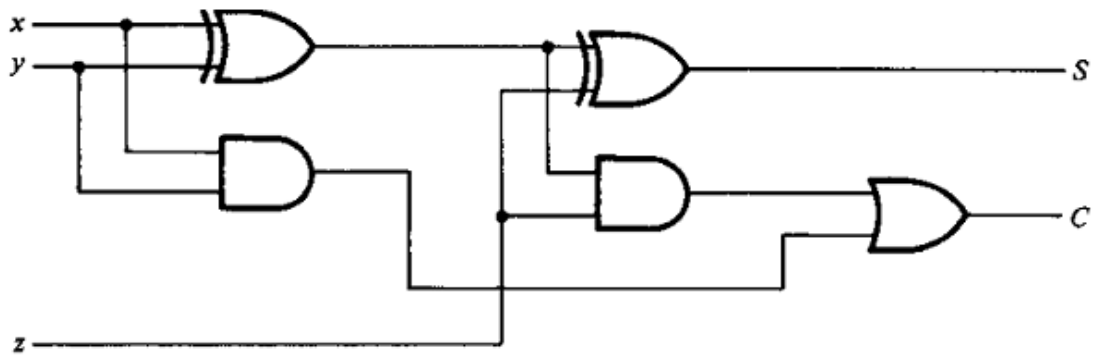


Figure: Circuit diagram of full adder

Implementation of full-adder with two half-adder





Implementation of full adder

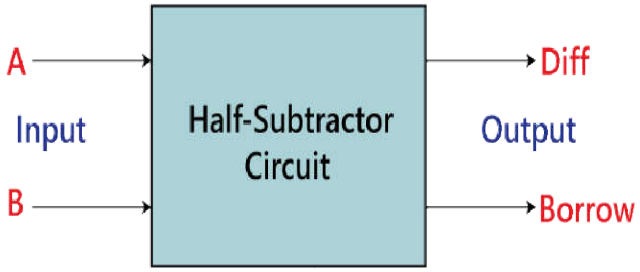
This implementation uses the following Boolean expressions:

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

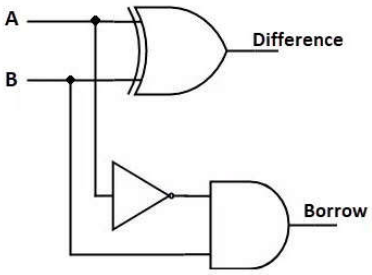
$$C = xy + xz + yz$$

Half-Subtractor

- Combinational circuit that subtracts two bits and produces their difference.
- It has two outputs; difference and borrow.



Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Difference (D) = A'B + AB'

Borrow (Bo) = A'B

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 A, B and Bin denote the minuend, subtrahend and previous borrow. The two outputs, D and Bo, represent difference and borrow out.

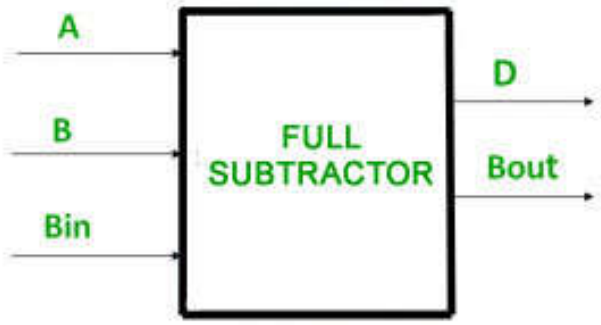


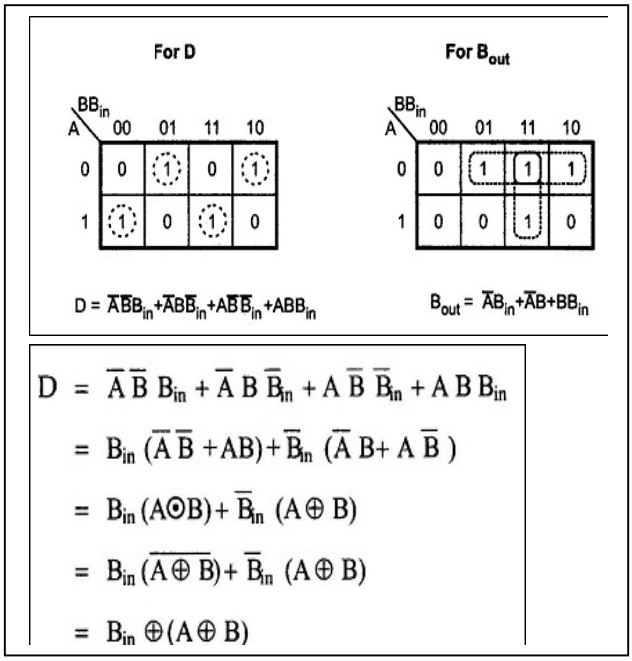
Figure: Block diagram of full subtractor

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

Table: Truth table of full-subtractor

B_{in} = Previous borrow

$D = (A - B - B_{in})$



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

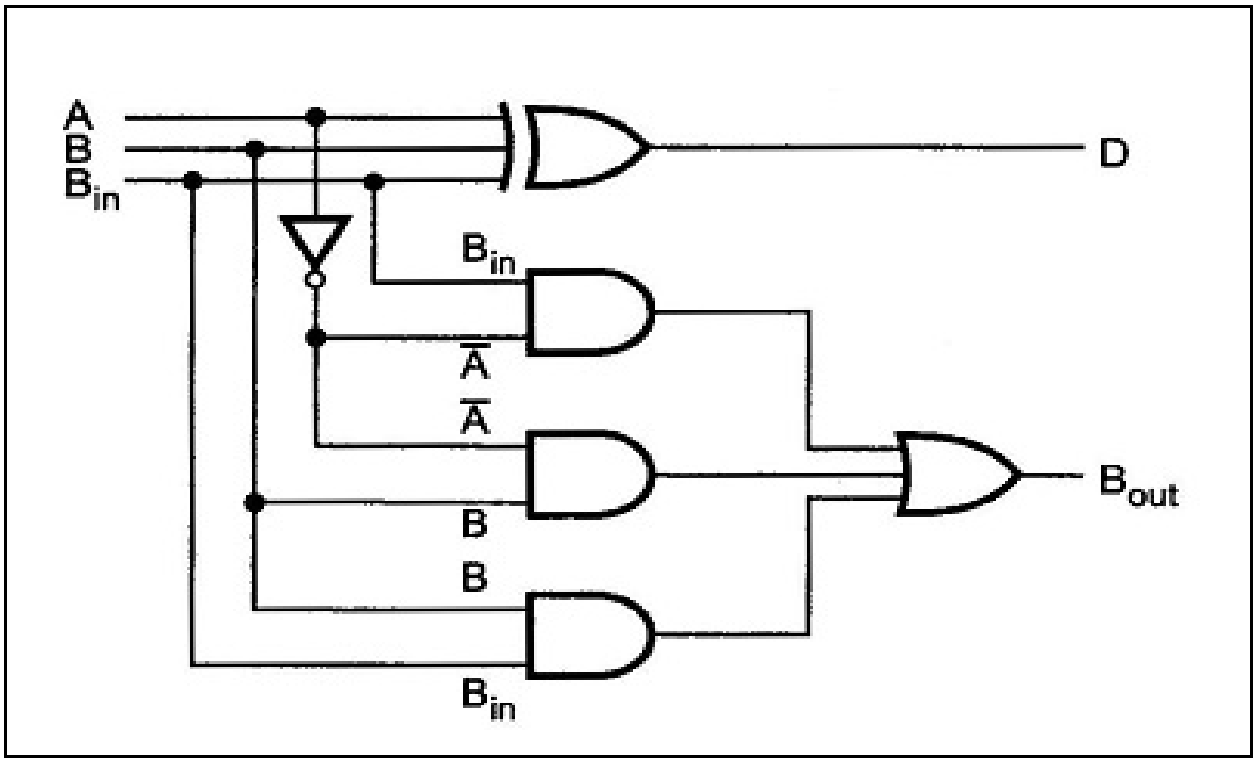


Figure: Circuit diagram of full-subtractor

5.3 Code Conversion

BCD to Excess-3 Code Converter

- Each code uses four bits to represent a decimal digit.
- There must be four input variables and four output variables.
- Let us designate the four input binary variables by the symbols *D3, D2,* D1, and *D0,* and the four output variables by *E3, E2, E1,* and E0.
- The truth table relating the input and output variables is shown in Table 4-1.
- The truth table shown below has only the valid 4-bit BCD codes. For the remaining input combinations, the output cannot be predicted. So they are don't care outputs.

Decimal Number	BCD code (Input)				Excess-3 code (Output)			
	D3	D2	D1	Do	E3	E2	E1	Eo
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

From the truth table, the min-terms are obtained for each outputs (E3, E2, E1, E0).

$E_3 = \sum m(5, 6, 7, 8, 9)$,

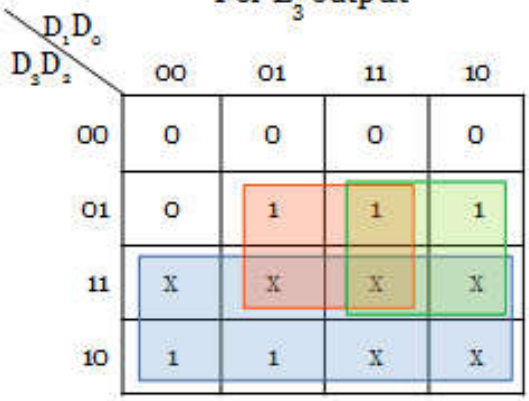
$E_2 = \sum m(1, 2, 3, 4, 9)$,

$E_1 = \sum m(0, 3, 4, 7, 8)$,

$E_0 = \sum m(0, 2, 4, 6, 8,)$

The min-terms of each output is plotted in k-map and simplified expression is obtained.

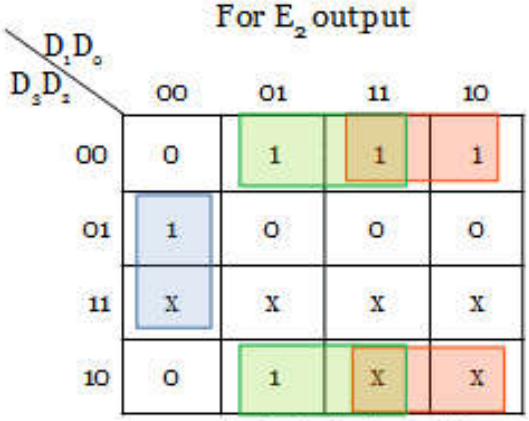
For E_3 output



D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$E_3 = D_3 + D_2D_0 + D_2D_1$$

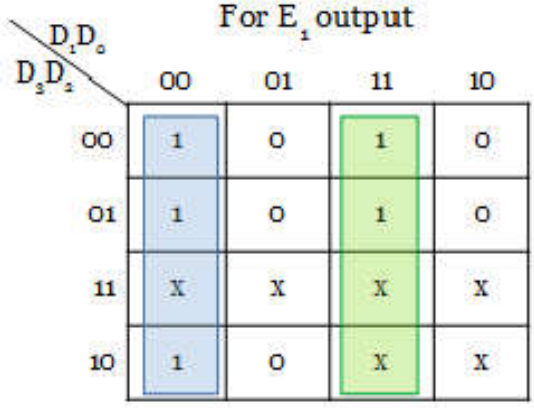
For E_2 output



D_3D_2 \ D_1D_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

$$E_2 = D_2\overline{D}_1\overline{D}_0 + \overline{D}_2D_0 + \overline{D}_2D_1$$

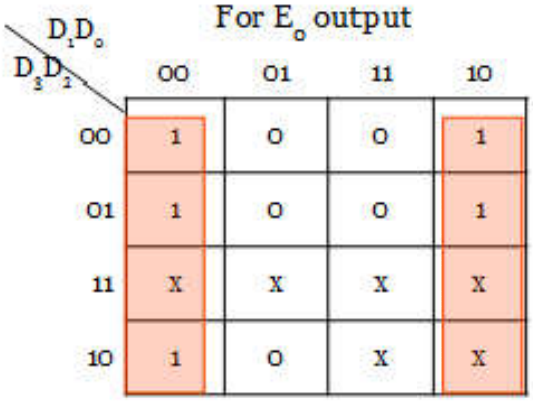
For E_1 output



D_3D_2 \ D_1D_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

$$E_1 = \overline{D}_1\overline{D}_0 + D_1D_0$$

For E_0 output



D_3D_2 \ D_1D_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

$$E_0 = \overline{D}_0$$

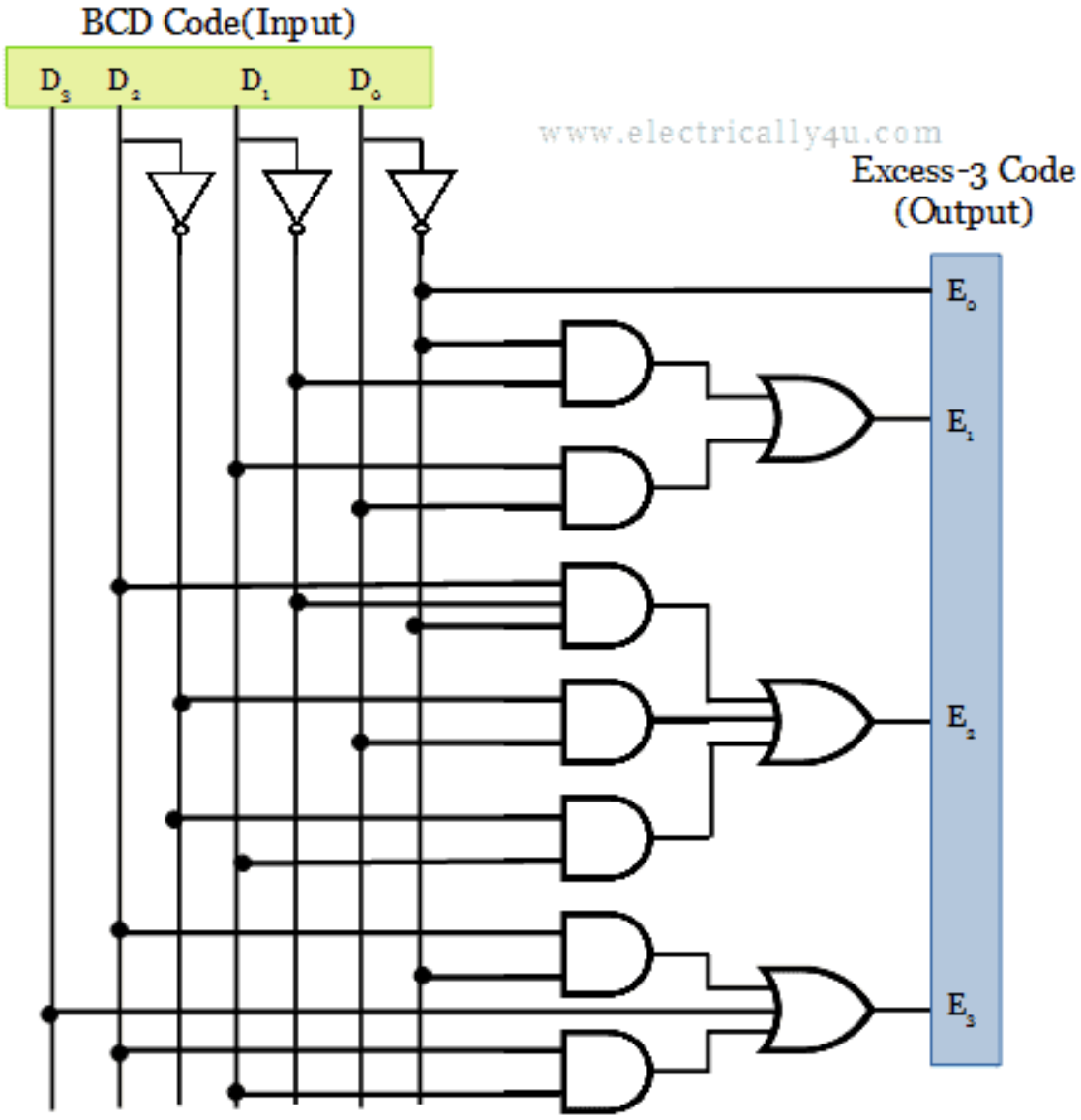


Figure: BCD to Excess-3 code converter

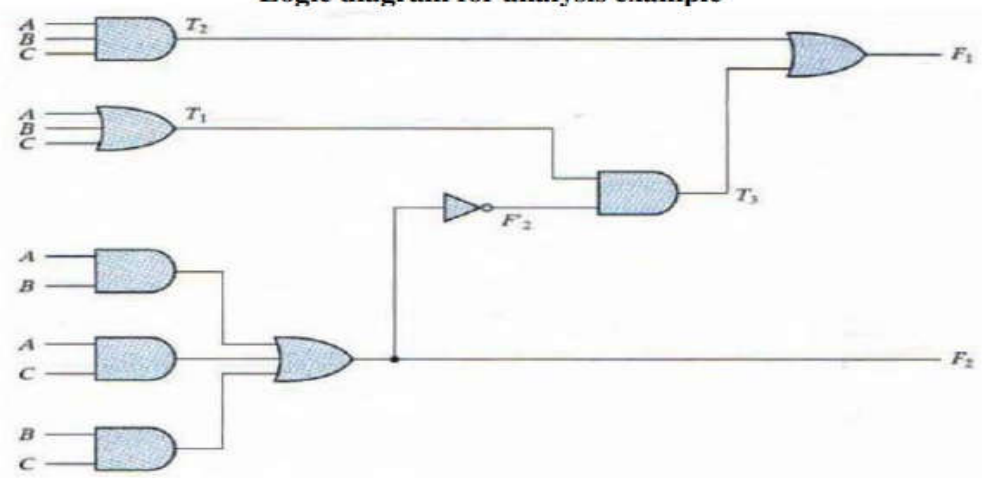
5.4 Analysis Procedure

- The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- To obtain the output Boolean functions from a logic diagram, we proceed as follows:
    1. Label all gate outputs that are a function of input variables 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 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.

Obtaining Boolean function from logic circuit

Logic diagram for analysis example



The Boolean functions for the above outputs are,

$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C \quad \& \quad T_2 = ABC$$

$$T_3 = \overline{F_2} T_1$$

$$F_1 = T_3 + T_2$$

Finally the output F_1 can br obtained as,

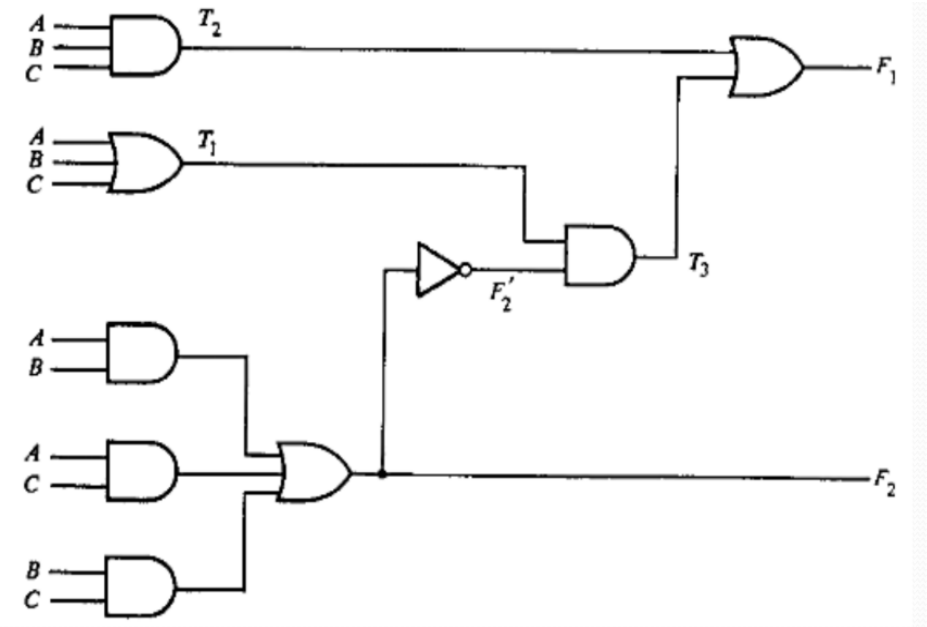
$$F_1 = \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$

Obtaining truth table from logic diagram and get the Boolean function

To obtain the truth table directly from the logic diagram without going through the derivation of the Boolean functions, we proceed as follows:

1) Form 2^n possible input combinations from n inputs.
2) Label the outputs of selected gates.
3) Obtain the truth table for the outputs of those gates which are functions of the input variables only.
4) Proceed to obtain the truth table of outputs of those gates which are a function of previously defined value until all outputs are determined.

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



5.5 Multi Level NAND and NOR Circuit

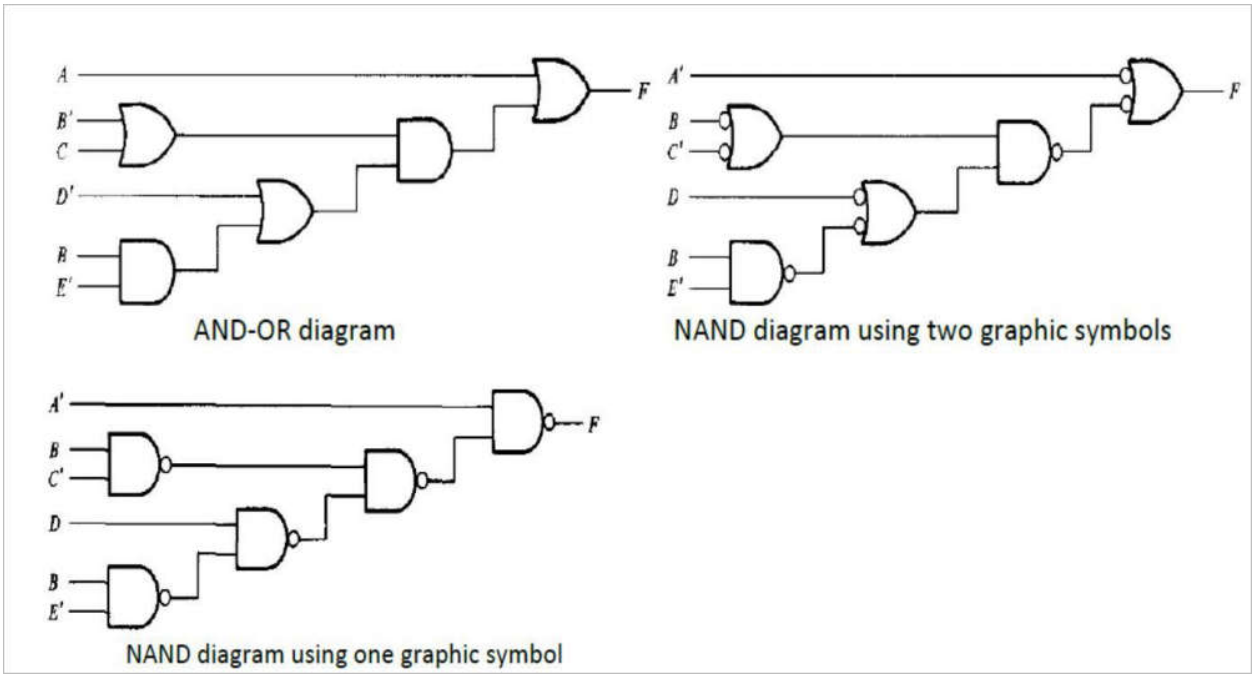
Multi Level NAND Circuit

- 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.

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.

Example:

F = A + (B' + C)(D' + BE')



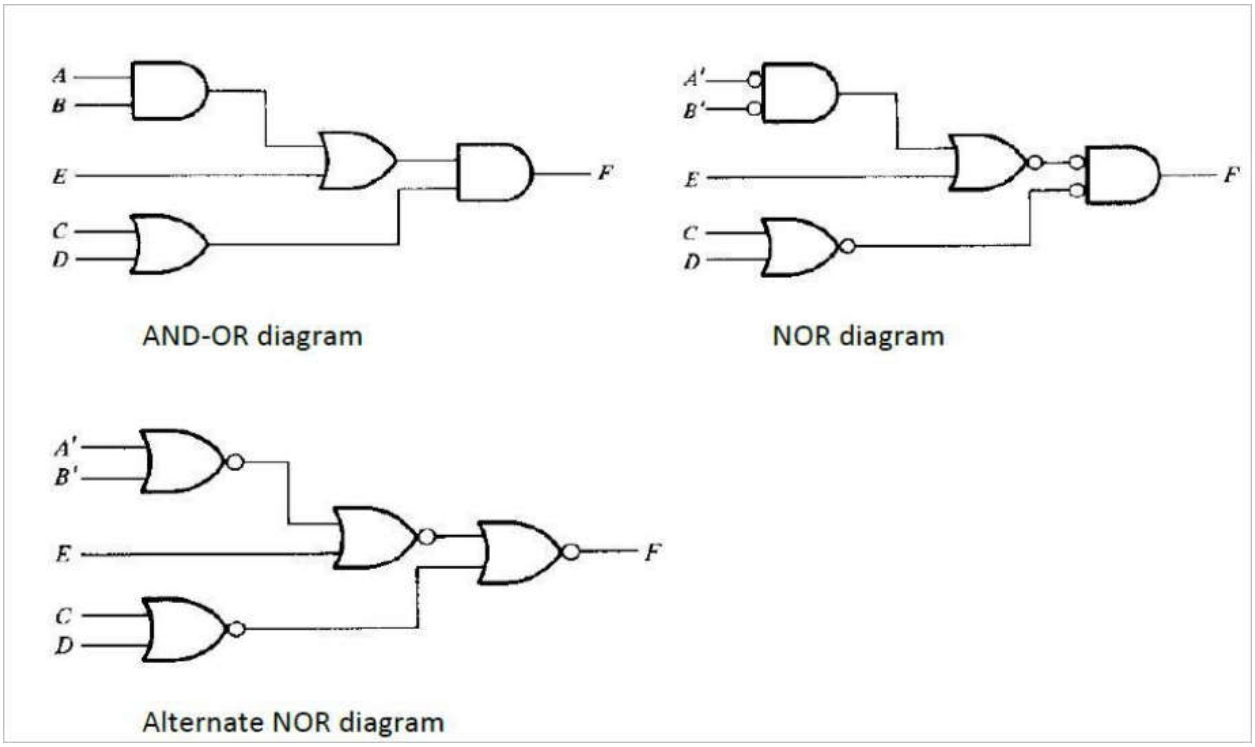
Multi Level NOR Circuit

- The procedure for implementing a Boolean function with NOR gates is similar to the procedure outlined in the previous section for NAND gates:
    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.

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.

Example:

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



5.6 Parity Generation and Checking

Parity Generator

- The circuit that generates the parity bit in the transmitter is called a *parity generator*.
- A parity bit is used for the purpose of detecting errors during transmission of binary information.
- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.
- In **Even** parity, the added parity bit will make the total number of 1's in message even.
- In **Odd** parity, the added parity bit will make the total number of 1's in message odd.

Parity Checker

- The message, including the parity bit, is transmitted and then checked at the receiving end for errors.
- An error is detected if the checked parity does not correspond with the one transmitted.
- The circuit that checks the parity in the receiver is called a *parity checker*.

Even Parity Generator

Example:

- Consider a 3-bit message to be transmitted together with an even parity bit.
- The three bits, *x, y,* and z, constitute the message and are the inputs to the circuit.
- The parity bit *P* is the output.
- For even parity, the bit *P* must be generated to make the total number of 1's even (including *P).*

Table: Even Parity Generator Truth Table

Min-terms	3-bit message			Even Parity Generator (P)
	W	X	Y	
m0	0	0	0	0
m1	0	0	1	1
m2	0	1	0	1
m3	0	1	1	0
m4	1	0	0	1
m5	1	0	1	0
m6	1	1	0	0
m7	1	1	1	1

$$PE = m1 + m2 + m4 + m7$$
$$= W'X'Y + W'XY' + WX'Y' + WXY$$
$$= Y (W'X' + WX) + Y'(W'X + WX')$$
$$= Y (W \oplus X)' + Y' (W \oplus X)$$
$$= (W \oplus X) \oplus Y$$

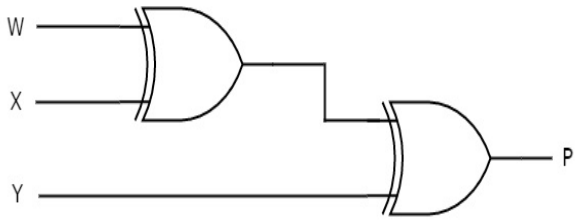


Figure: 3-bit even parity generator

Even Parity Checker

Assume a 3-bit binary input; WXY is transmitted along with an even parity bit, P. So, the resultant word data contains 4 bits, which will be received as the input of even parity checker.

Table: Truth table of Even Parity Checker

Min-terms	INPUTS				output
	W	X	Y	P	E
m0	0	0	0	0	0
m1	0	0	0	1	1
m2	0	0	1	0	1
m3	0	0	1	1	0
m4	0	1	0	0	1
m5	0	1	0	1	0
m6	0	1	1	0	0
m7	0	1	1	1	1
m8	1	0	0	0	1
m9	1	0	0	1	0
m10	1	0	1	0	0
m11	1	0	1	1	1
m12	1	1	0	0	0
m13	1	1	0	1	1
m14	1	1	1	0	1
m15	1	1	1	1	0

From the truth table:

E = m1 + m2 + m4 + m7 + m8 + m11 + m13 + m14

From the above Truth table, we can observe that the even parity check bit value is '1', when odd number of 1's present in the received data. That means the Boolean function of even parity check bit is an **odd function**. Exclusive-OR function satisfies this condition. Hence, we can directly write the **Boolean function** of even parity check bit as:

E=W⊕X⊕Y⊕P

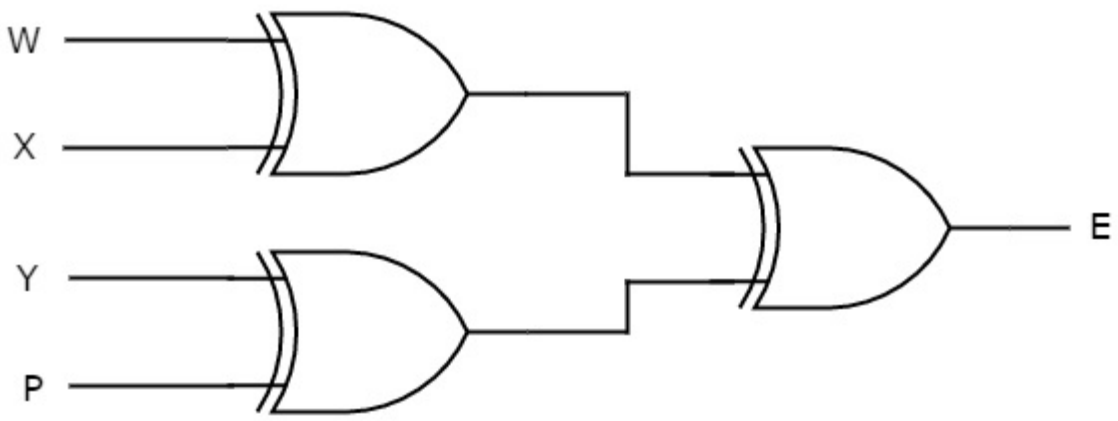


Figure: 4-bit even parity bit checker

- ✓ This circuit consists of three **Exclusive-OR gates** having two inputs each.
- ✓ The first level gates produce outputs of

  W⊕X & Y⊕P.
- ✓ The Exclusive-OR gate, which is in second level produces an output of **W ⊕ X ⊕ Y ⊕ P**

Assignment: Design odd parity bit generator and checker circuit