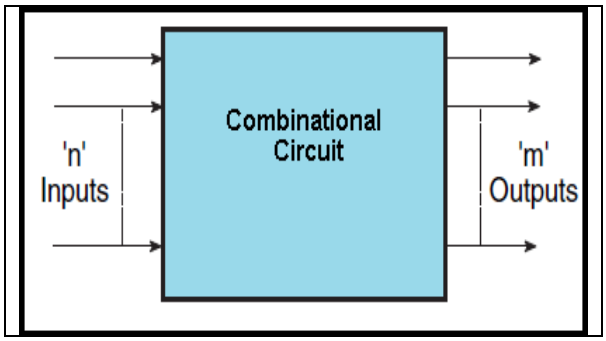



A *combinational circuit* is one where the output of that circuit at any time only depends on the present combination values of inputs. And the output is never depending on any past state value of input combination as well as the previous state output value. Now if we take any example then the logic gate is the most basic building block of combinational logic. Now as we know logic gates is work according the specified logic for specified gate. And those combinational circuits which are made by logic gate maintain Boolean expression. In above see the block diagram of generalized combinational circuit. In above block diagram we can see that combinational logic circuit has "n" inputs that mean it can take $2^n$combination of input values. The output "m" is depending on the Boolean expression of combinational logic circuit.

***HALF-ADDER*:** As the name suggests *half-adder* is an arithmetic circuit block by using this circuit block we can be used to add two bits. As we know it can add two bit number so it has two inputs terminals (***A*** & ***B***) and as well as two outputs terminals, with one producing the SUM (***S***) output and the other producing the CARRY OUT ($C_{OUT}$).

TRUTH TABLE FOR HALF ADDER

| A | B | S | $C_{OUT}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

K-map for S

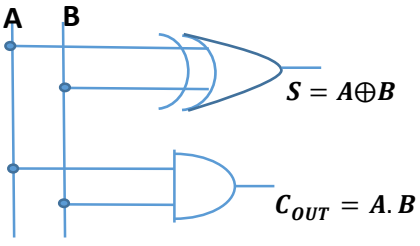
| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

K-map for $C_{OUT}$

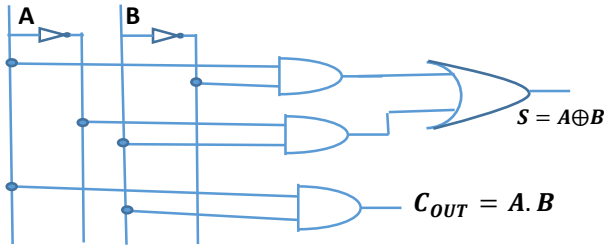
| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Then for S we have $f(A,B) = \sum(1,2) = \prod(0,3)$ **and** $C_{OUT}$ for we have $f(A,B) = \sum(3) = \prod(0,1,2)$

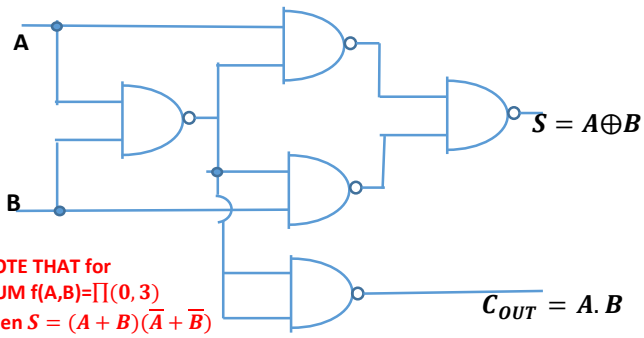
Then **Sum (S)**$= \overline{A}B + A\overline{B} = A \oplus B$ **& Carry out** $(C_{OUT}) = AB$




**Half adder using X-OR and AND gates**

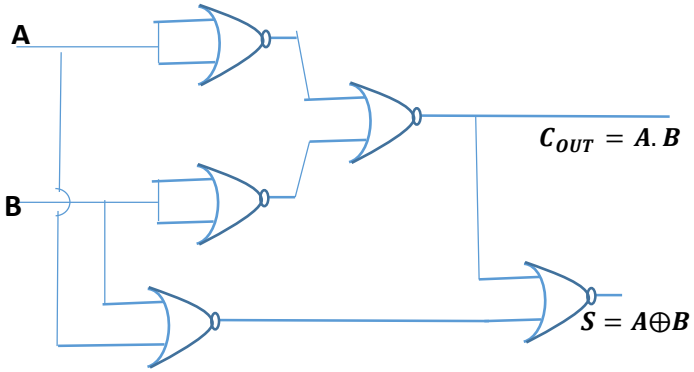



**Half adder using basic gates**




NOTE THAT for SUM f(A,B)=$\prod(0,3)$ Then $S = (A+B)(\overline{A}+\overline{B})$

**Half adder using NAND gates**




**Half adder using NOR gates**

***Full Adder*:** As we read early half adder can add only two bit of numbers so if we want to add two bit of input along with carry then we have to add three bits. But in half adder we can add only two bits so we need some extra. A *full adder* circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY OUTPUT. By using full adder we can add large numbers because it can add two bit number with carry. Let us recall the procedure for adding larger binary numbers. We begin with the addition of LSBs of the forward to the next higher column bits. As a result, when we add the next adjacent higher column bits, we would be required to add three bits if there were a carry from the previous addition. We have a similar situation for the other higher column bits also until we reach the MSB. A full adder is therefore essential for the hardware implementation of an adder circuit capable of adding larger binary numbers. For full adder circuit the relation between input and output expressed by the Boolean expressions for the SUM(**S**) and CARRY OUTPUTS ($C_{OUT}$)

TRUTH TABLE FOR FULL ADDER

| $C_{IN}$ | A | B | S | $C_{OUT}$ |
|---|---|---|---|---|
| 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 |

K-map for S

| $C_{in}$A / B | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

K-map for $C_{OUT}$

| $C_{in}$A / B | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

Then for **S** we have $f(C_{IN}, A, B) = \sum(1,2,4,7) = \prod(0,3,5,6)$ **and** $C_{OUT}$ for we have $f(C_{IN}A, B) = \sum(3,5,6,7) = \prod(0,1,2,4)$

From truth table $S = \sum m(1,2,4,7) = \overline{C_{IN}}\,\overline{A}\,B + \overline{C}_{IN}A\,\overline{B} + C_{IN}\overline{A}\,\overline{B} + C_{IN}AB = \overline{C_{IN}}\,(\overline{A}\,B + A\,\overline{B}) + C_{IN}(\overline{A}\,\overline{B} + AB)$

$= \overline{C_{IN}}\,(A \oplus B) + C_{IN}\,(A \odot B) = \overline{C_{IN}}\,(A \oplus B) + C_{IN}\overline{(A \oplus B)} = \mathbf{A \oplus B \oplus} C_{IN}$

Again $C_{OUT} = \sum m(3,5,6,7) = \overline{C_{IN}}A\,B + C_{IN}\overline{A}B + C_{IN}A\overline{B} + AB\,C_{IN} = (\overline{A}\,B + A\overline{B})\,C_{IN} + AB(\overline{C}_{IN} + C_{IN}) =$ **(A ⊕ B)** $C_{IN}$ **+ A B**

**NOTE THAT**

**1.** $C_{OUT} = \sum m(3,5,6,7) = \overline{C_{IN}}A\,B + C_{IN}\overline{A}B + C_{IN}A\overline{B} + AB\,C_{IN} = \overline{C_{IN}}A\,B + C_{IN}\overline{A}B + C_{IN}A\overline{B} + AB\,C_{IN} + AB\,C_{IN} + AB\,C_{IN}$

$= AB(C_{IN} + \overline{C_{IN}}) + C_{IN}B(A + \overline{A}) + C_{IN}A(B + \overline{B}) = AB + C_{IN}B + C_{IN}A.$

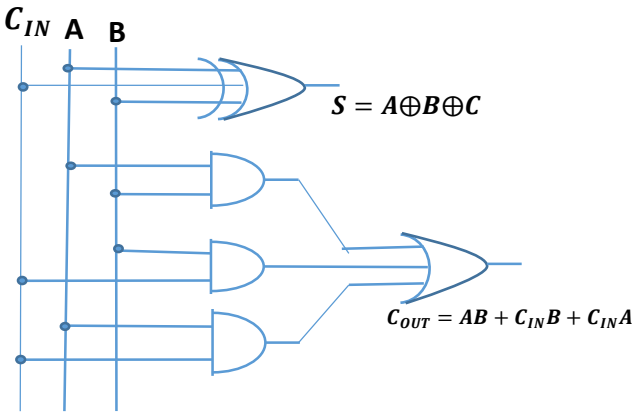
2. $\overline{C_{OUT}} = \overline{AB + C_{IN}B + C_{IN}A} = \overline{(AB)}.\overline{(C_{IN}B)}.\overline{(C_{IN}A)} = (\overline{A} + \overline{B})(\overline{C_{IN}} + \overline{B})(\overline{C_{IN}} + \overline{A}) = (\overline{A}\,\overline{C_{IN}} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C_{IN}} + \overline{B})(\overline{C_{IN}} + \overline{A})$

$= \overline{A}\,\overline{C_{IN}} + \overline{A}\,\overline{C_{IN}} + \overline{A}\,\overline{B}\,\overline{C_{IN}} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C_{IN}} + \overline{B}\,\overline{C_{IN}}\,\overline{A} + \overline{B}\,\overline{C_{IN}} + \overline{B}\,\overline{A} = \overline{A}\,\overline{B} + \overline{B}\,\overline{C_{IN}}(1 + \overline{A}) + \overline{A}\,\overline{C_{IN}}$

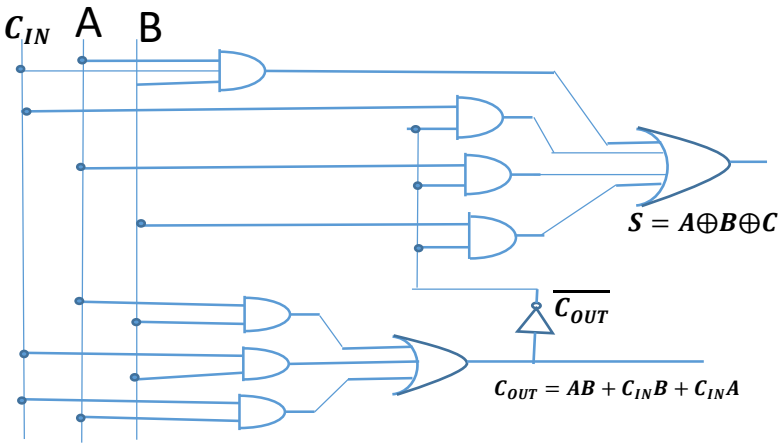
$= \overline{A}\,\overline{B} + \overline{C_{IN}}\,\overline{B} + \overline{C_{IN}}\,\overline{A}$

3. Again $(C_{IN} + A + B)\overline{C_{OUT}} + AB\,C_{IN} = C_{IN}\overline{C_{OUT}} + A\overline{C_{OUT}} + B\overline{C_{OUT}} + AB\,C_{IN} = C_{IN}\overline{A}\,\overline{B} + A\overline{C}_{IN}\,\overline{B} + B\overline{C_{IN}}\,\overline{A} + AB\,C_{IN} = S$

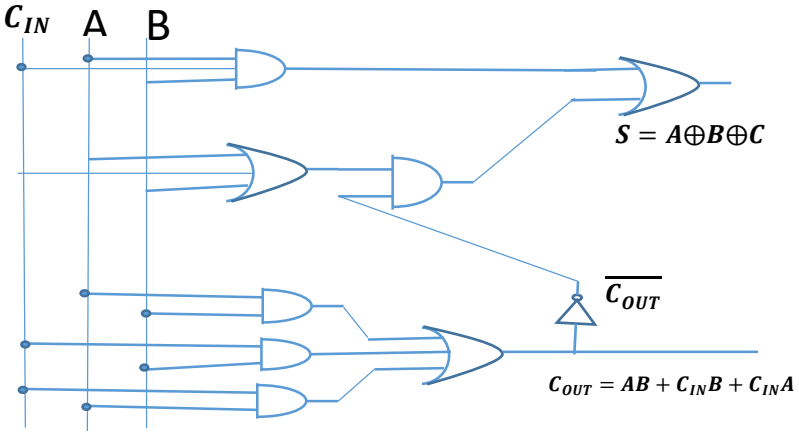
Then $= (C_{IN} + A + B)\overline{C_{OUT}} + AB\,C_{IN}.$



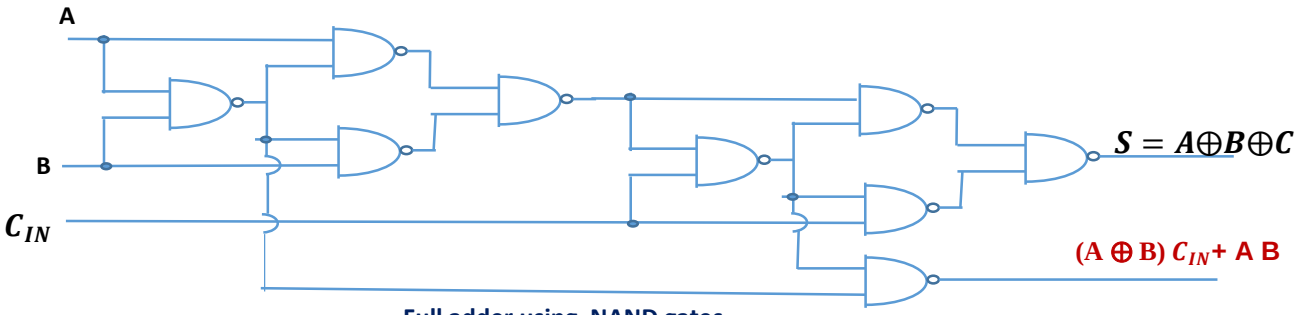

full adder using X-OR and AND and OR gates



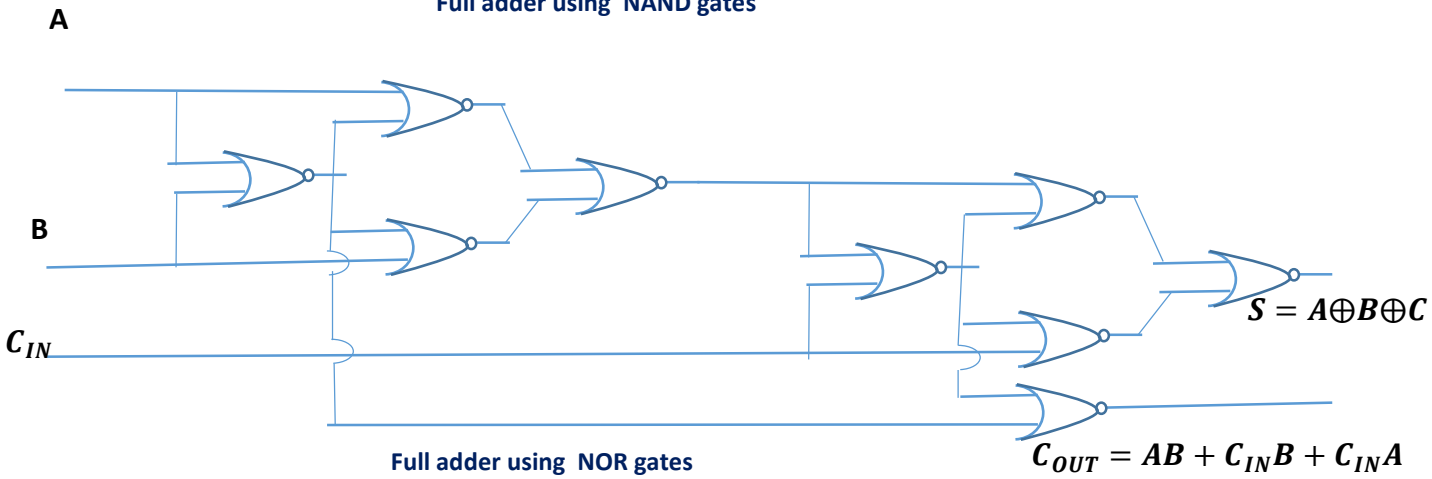

Full adder using basic gates




Full adder using basic gates




Full adder using NAND gates




Full adder using NOR gates

***HALF-SUBTRACTOR*:** As the name suggests *half-subtractor* is an arithmetic circuit block by using this circuit block we can be used to subtract two bits. As we know it can subtract two bit number so it has two inputs terminals (***A*** & ***B***) and as well as two outputs terminals, with one producing the DIFFEENCE ($D$) output and the other producing the BORROW OUT ($B_{OUT}$).

TRUTH TABLE FOR HALF ADDER

| A | B | D | $B_{OUT}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

K-map for D

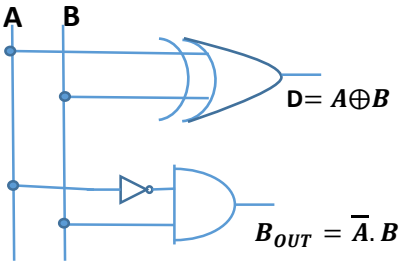
| A \ B | 0 | 1 |
|---|---|---|
| 0 | **0** | **1** |
| 1 | 1 | 0 |

K-map for $B_{OUT}$

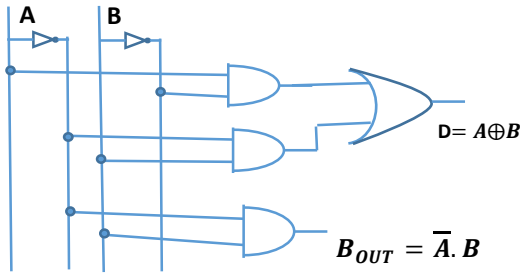
| A \ B | 0 | 1 |
|---|---|---|
| 0 | **0** | **0** |
| 1 | 1 | 0 |

Then for **D** we have $f(A,B) = \sum(1,2) = \prod(0,3)$ **and** $B_{OUT}$ for we have $f(A,B) = \sum(1) = \prod(0,2,3)$

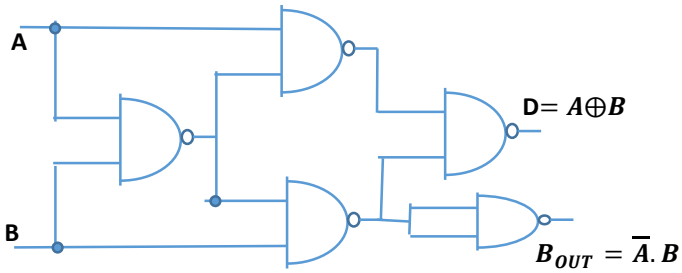
Then **DIFFERENCE (D)**$= \overline{A}B + A\overline{B} = A \oplus B$ **&** **BORROW OUT** $(B_{OUT}) = \overline{A}B$




Half subtractor using X-OR and NOT and AND gates

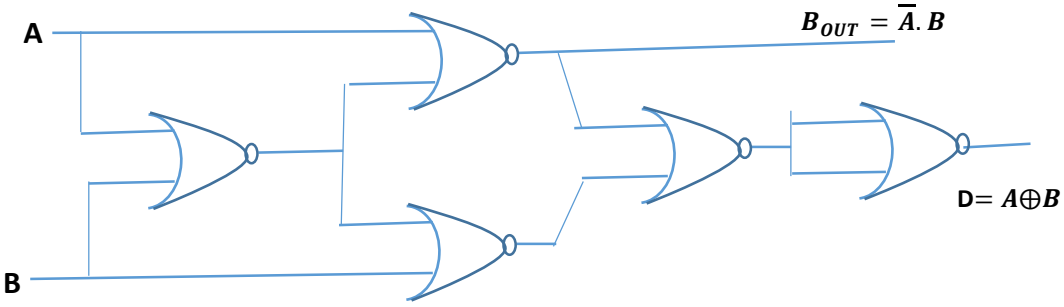



Half subsector using basic gates




NOTE THAT for DIFFERENCE f(A,B)=$\prod(0,3)$
Then D= $(A+B)(\overline{A}+\overline{B})$

Half subtractor using NAND gates




Half sub tractor using NOR gates

***Full Subtractor*:** As we read early half subtractor can subtract only two bit of numbers so if we want to subtract two bit of input along with borrow then we have to subtract three bits. But in half subtractor we can subtract only two bits so we need some extra. A *full subtractor* circuit is an arithmetic circuit block that can be used to subtract three bits to produce a DIFFERENCE and a BORROW OUTPUT. By using full subtractor we can subtract large numbers because it can subtract two bit number with borrow. Let us recall the procedure for subtracting larger binary numbers. We begin with the subtraction of LSBs of the forward to the next higher column bits. As a result, when we subtract the next adjacent higher column bits, we would be required to subtract three bits if there were a borrow from the previous subtraction. We have a similar situation for the other higher column bits also until we reach the MSB. A full subtractor is therefore essential for the hardware implementation of an subtractor circuit capable of subtracting larger binary numbers. For full subtractor circuit the relation between input and output expressed by the Boolean expressions for the DIFFERENCE**(D)** and BORROW OUTPUTS ($B_{OUT}$)

TRUTH TABLE FOR FULL SUBTRCTOR

| $B_{IN}$ | A | B | D | $B_{OUT}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

K-map for D

| $B$ \ $D_{in}$A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

K-map for $B_{OUT}$

| $B$ \ $D_{in}$A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |

Then for **D** we have $f(B_{IN}, A, B) = \sum(1,2,4,7) = \prod(0,3,5,6)$ **and** $B_{OUT}$ for we have $f(B_{IN}A, B) = \sum(1,4,5,7) = \prod(0,2,3,6)$

From truth table **D** = $\sum m(1,2,4,7) = \overline{B_{IN}}\,\overline{A}\,B + \overline{B}_{IN}A\,\overline{B} + B_{IN}\overline{A}\,\overline{B} + B_{IN}AB = \overline{B_{IN}}\,(\overline{A}\,B + A\,\overline{B}) + B_{IN}(\overline{A}\,\overline{B} + AB)$

$= \overline{B_{IN}}\,(A \oplus B) + B_{IN}\,(A \odot B) = \overline{B_{IN}}\,(A \oplus B) + B_{IN}\overline{(A \oplus B)} = \mathbf{A \oplus B \oplus B_{IN}}$

Again $B_{OUT} = \sum m(1,4,5,7) = \overline{B_{IN}}\,\overline{A}\,B + B_{IN}\overline{AB} + B_{IN}\overline{AB} + AB\,B_{IN} = (\overline{A}\,\overline{B} + AB)\,B_{IN} + \overline{A}B(\overline{B}_{IN} + B_{IN}) =$ **$(A \odot B)\,B_{IN} + \overline{A}\,B$**

**NOTE THAT**

**1.** $B_{OUT} = \sum m(1,4,5,7) = \overline{B_{IN}}\,\overline{A}\,B + B_{IN}\overline{AB} + B_{IN}\overline{A}B + AB\,B_{IN} = \overline{B_{IN}}\,\overline{A}\,B + B_{IN}\overline{AB} + B_{IN}\overline{A}B + AB\,B_{IN} + AB\,B_{IN} + AB\,B_{IN}$

$= \overline{A}B(\overline{B_{IN}} + B_{IN}) + B_{IN}\overline{A}(\overline{B} + B) + B_{IN}B(A + \overline{A}) = \overline{A}B + B_{IN}\overline{A} + B_{IN}B.$

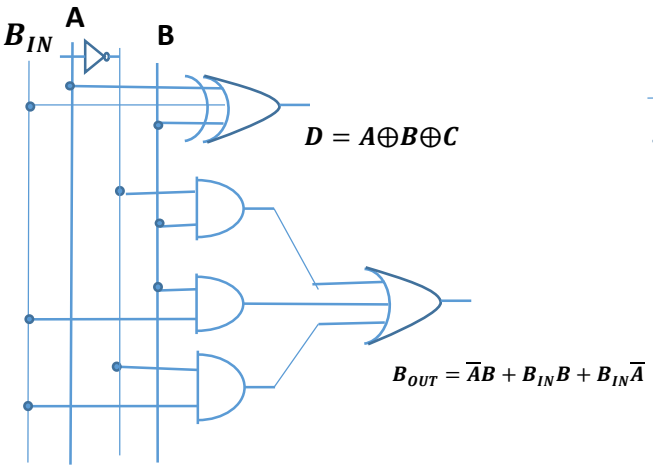
2. $\overline{B_{OUT}} = \overline{\overline{A}B + B_{IN}B + B_{IN}\overline{A}} = \overline{(\overline{A}B)}.\overline{(B_{IN}B)}.\overline{(B_{IN}\overline{A})} = (A + \overline{B})(\overline{B_{IN}} + \overline{B})(\overline{B_{IN}} + A) = (A\,\overline{B_{IN}} + A\,\overline{B} + \overline{B}\,\overline{B_{IN}} + \overline{B})(\overline{B_{IN}} + A)$

$= A\,\overline{B_{IN}} + A\,\overline{B_{IN}} + A\,\overline{B}\,\overline{B_{IN}} + A\,\overline{B} + \overline{B}\,\overline{B_{IN}} + \overline{B}\,\overline{B_{IN}}\,A + \overline{B}\,\overline{B_{IN}} + \overline{B}\,A = A\,\overline{B} + \overline{B}\,\overline{B_{IN}}(1 + A) + A\,\overline{B_{IN}}$

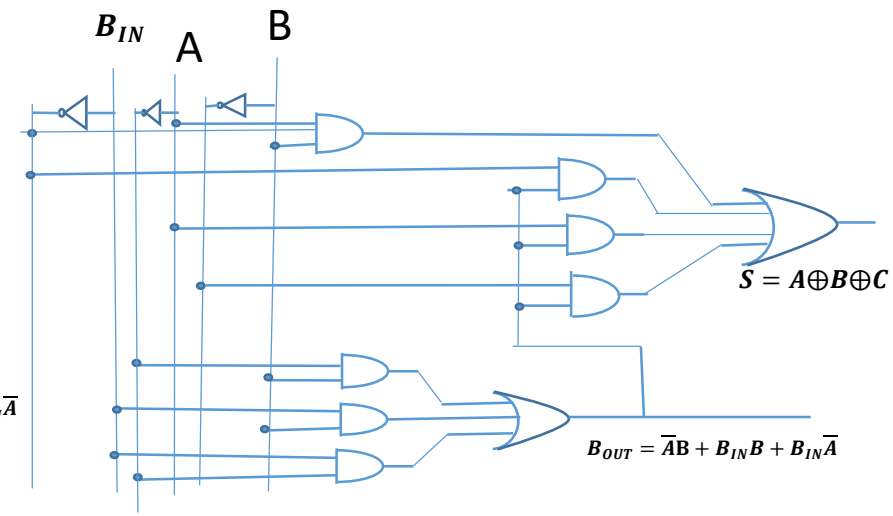
$= A\,\overline{B} + \overline{B_{IN}}\,\overline{B} + A\overline{B_{IN}}$

3. Again $(\overline{B_{IN}} + A + \overline{B})B_{OUT} + \overline{B}_{IN}A\,\overline{B} = \overline{B}_{IN}\,B_{OUT} + AB_{OUT} + \overline{B}B_{OUT} + \overline{B}_{IN}A\,\overline{B} = \overline{B_{IN}}\,\overline{A}\,B + AB_{IN}\,B + \overline{B}B_{IN}\overline{A} + \overline{B}_{IN}A\,\overline{B} = D$

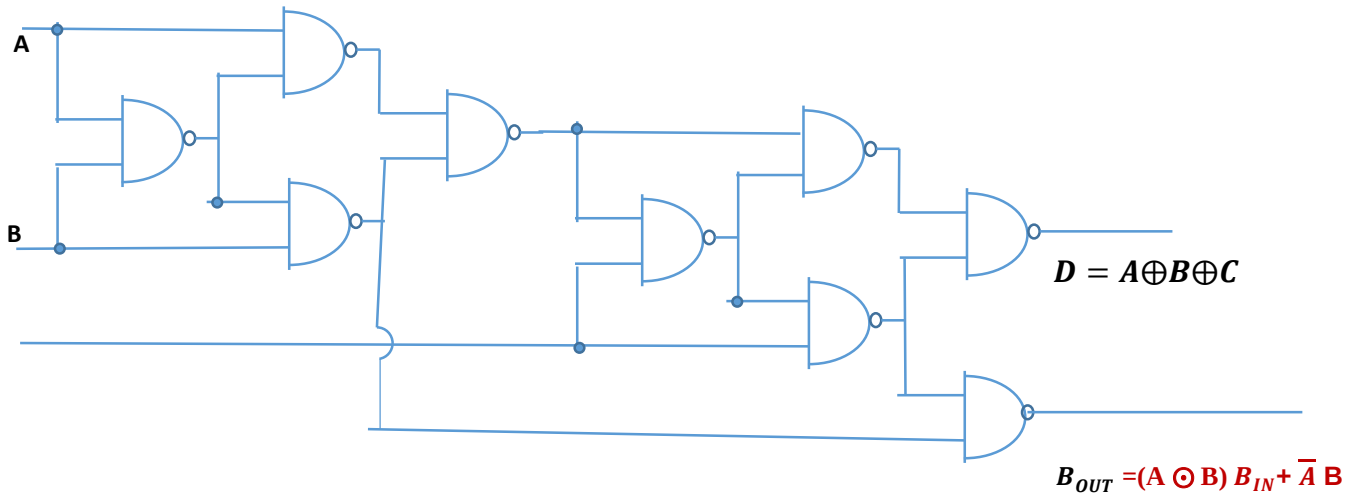
Then $= (\overline{B_{IN}} + A + \overline{B})B_{OUT} + \overline{B}_{IN}A\,\overline{B}.$



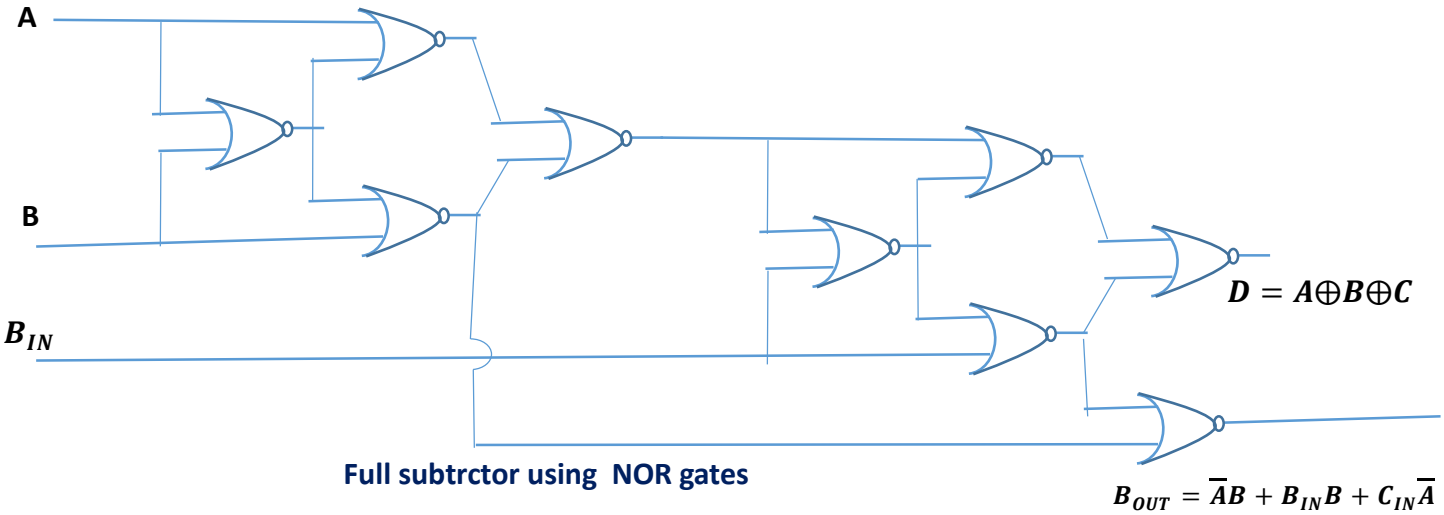

full sdubtractor using X-OR and AND and OR gates




Full subtractor using basic gates

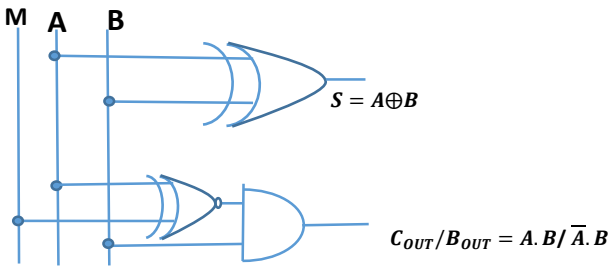



Full subtractor using NAND gates



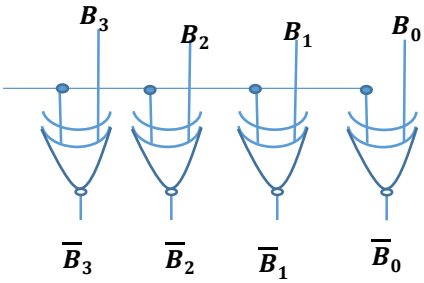

Full subtrctor using NOR gates

***HALF-ADDER//SUBTRACTOR:*** As X-OR gate can be used as an inverter when one of the input is high state for positive logic.

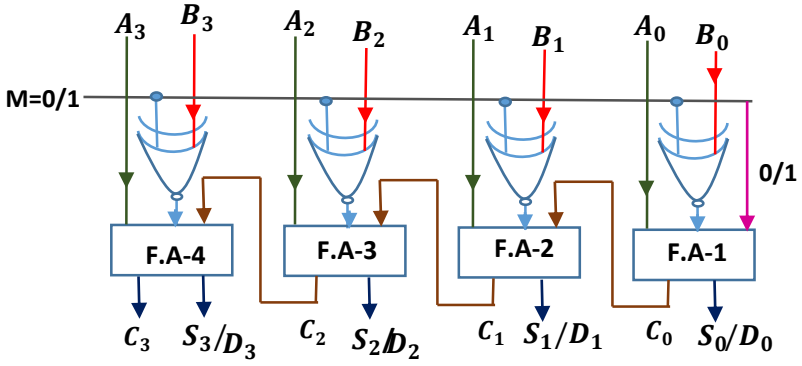




Half adder(M=0)/subtractor(M=1) using X-OR and AND gates

***FULL ADDER//2's COMPLEMENT SUBTRACTOR:***




1's complement circuit




Full adder(M=0) //2's complement subtractor (M=1) circuit

**DECODER :** In digital electronics, a **binary decoder** is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of $2^n$ unique outputs.

They are used in a wide variety of applications, including data demultiplexing, seven segment displays, and memory address decoding.

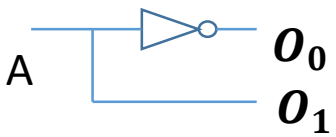
There are several types of binary decoders, but in all cases a decoder is an electronic circuit with multiple input and multiple output signals, which converts every unique combination of input states to a specific combination of output states. In addition to integer data inputs, some decoders also have one or more "enable" inputs. When the enable input is negated (disabled), all decoder outputs are forced to their inactive states.

Depending on its function, a binary decoder will convert binary information from n input signals to as many as $2^n$ unique output signals. Some decoders have less than $2^n$ output lines; in such cases, at least one output pattern will be repeated for different input values.

The simplest is the 1-to-2 line decoder. The truth table and the circuit are

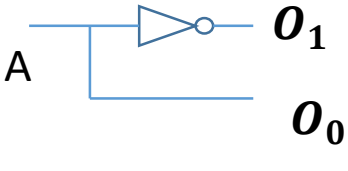
The simplest is **the 1-to-2 line decoder (active HIGH output)**.
The truth table and the circuit are

| A | $O_1$ | $O_0$ |
|---|---|---|
| 0 | O | 1 |
| 1 | 1 | 0 |



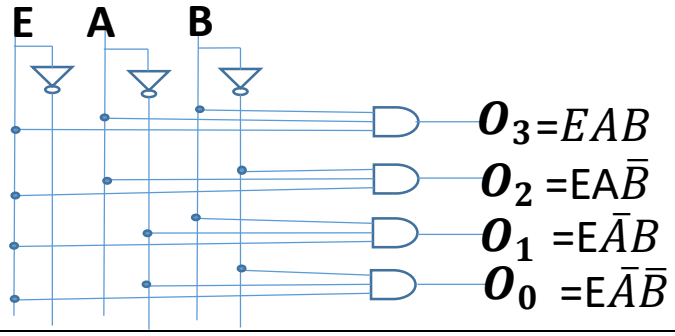

The simplest is **the 1-to-2 line decoder (active LOW output)**.
The truth table and the circuit are

| A | $O_1$ | $O_0$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 1 |



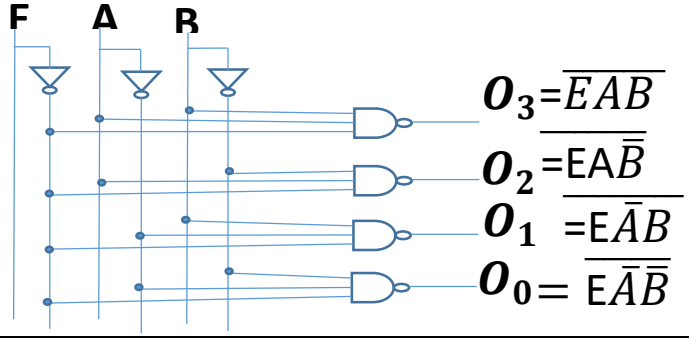
**The 2-to-4 line decoder (active HIGH output).**

| E | A | B | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | X | X | 0 | 0 | 0 | 0 |



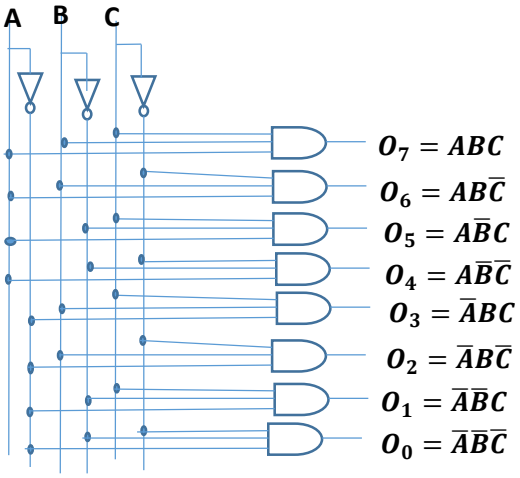

**The 2-to-4 line decoder (active LOW output).**

| E | A | B | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | X | X | 1 | 1 | 1 | 1 |



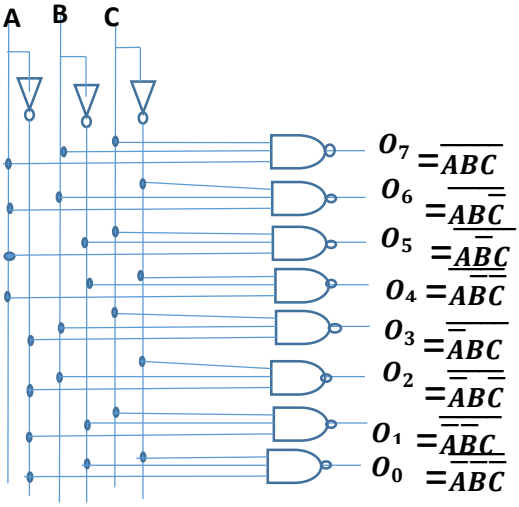

**The 3-to-8 line decoder (active HIGH output).**

| INPUTS | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



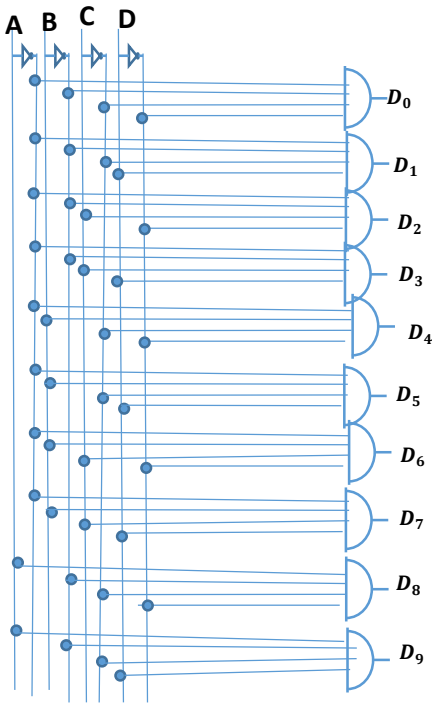

**The 3-to-8 line decoder (active LOW output).**

| INPUTS | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |



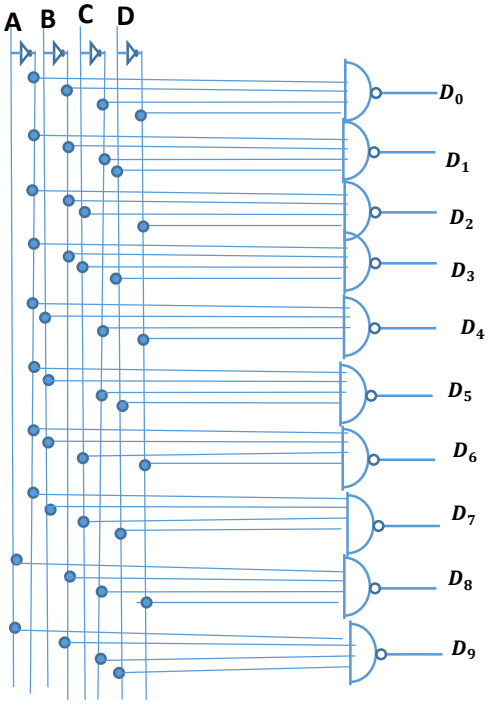
BCD to decimal decoder **(active HIGH output)**.

| INPUTS | | | | OUTPUTS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $O_9$ | $O_8$ | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |




BCD to decimal decoder **(active LOW output)**.

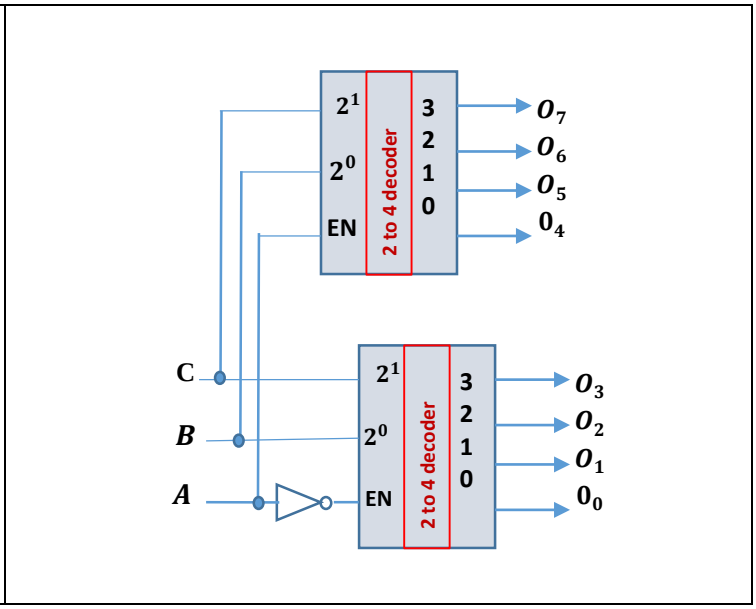
| INPUTS | | | | OUTPUTS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $O_9$ | $O_8$ | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_1$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |



## CASCADING OF DECODER

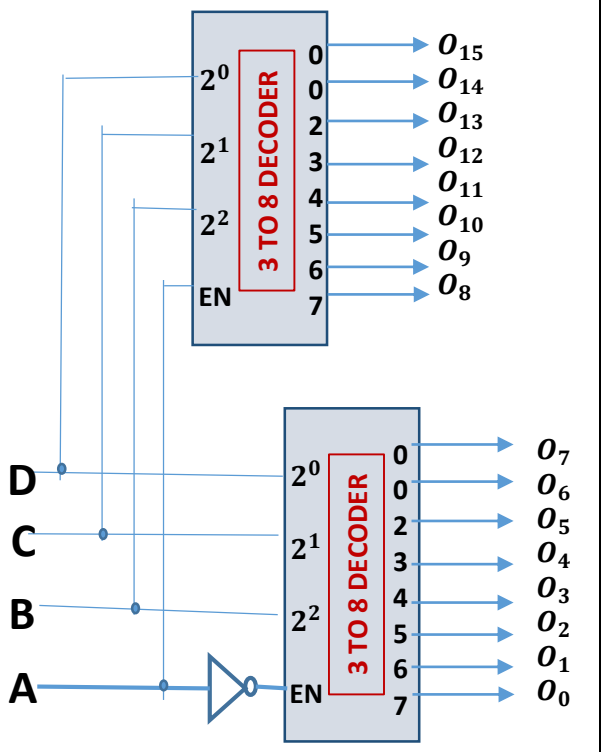
### 3 to 8 Decoder using two 2 to 4 Decoder

| INPUTS | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



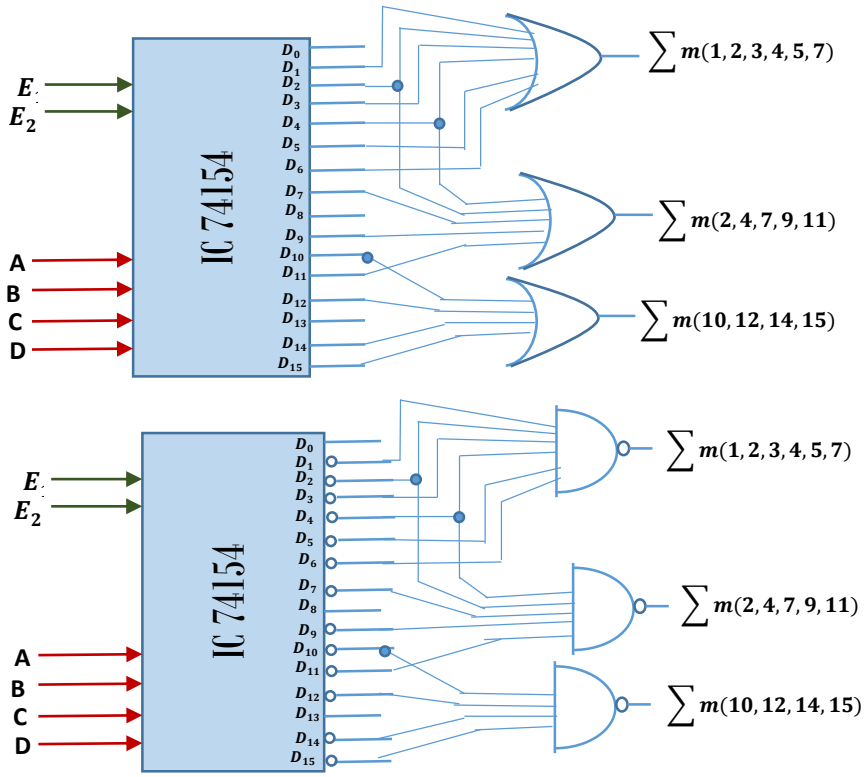

### 4 to 16 Decoder using two 3 to 8 Decoder

| INPUTS | | | | OUTPUTS | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $O_{15}$ | $O_{14}$ | $O_{13}$ | $O_{12}$ | $O_{11}$ | $O_{10}$ | $O_9$ | $O_8$ | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Design combinational logic circuits for the logic function (i) $F(A,B,C,D) = \sum m(2,4,7,9,11)$**

**(ii) $F(A,B,C,D) = \sum m(2,4,7,9,11)$ (iii) $F(A,B,C,D) = \sum m(10,12,14,15)$**





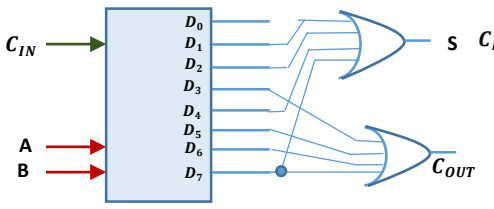


## BINARY FULL ADDER USING DECODER

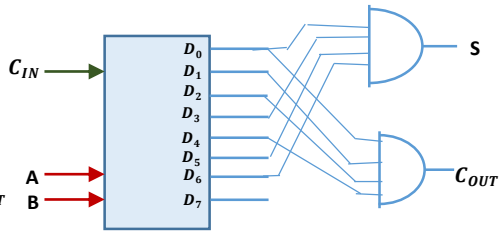
Truth table for full adder

| $C_{IN}$ | A | B | S | $C_{OUT}$ |
|---|---|---|---|---|
| 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 |

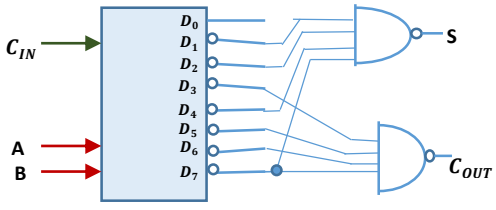
Then for S we have $f(C_{IN},A,B) = \sum(1,2,4,7) = \prod(0,3,5,6)$ **and** $C_{OUT}$ for we have $f(C_{IN}A,B) = \sum(3,5,6,7) = \prod(0,1,2,4)$



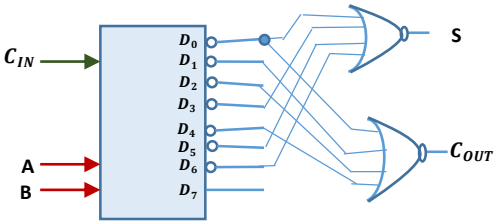

Using DECODER and OR gate




Using DECODER and AND gate




Using DECODER and NAND gate




Using DECODER and NOR gate

## BCD to 7 segment display

BCD (Binary Coded Decimal) is an encoding scheme which represents each of the decimal numbers by its equivalent 4-bit binary pattern. Seven segment displays comprise of seven individual segments formed by either Light Emitting Diodes (LEDs) or Liquid Crystal Displays (LCDs) arranged in a definite pattern (Figure 1).

| Decimal Digit | Input lines | | | | Output lines | | | | | | | Display pattern |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | a | b | c | d | e | f | g | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |

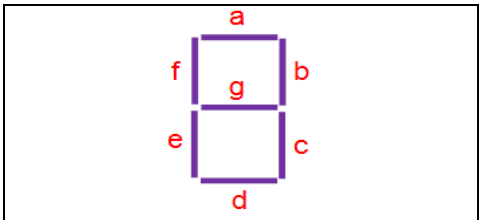
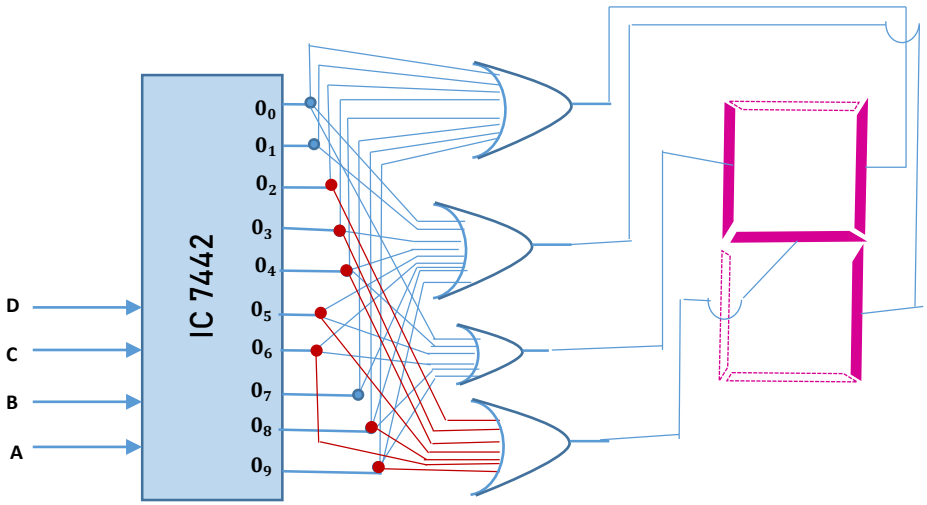


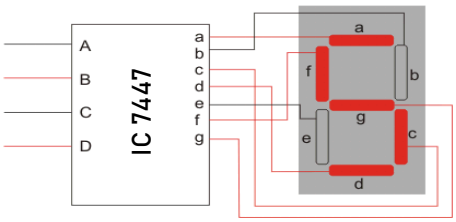
Figure1 Seven segment display

$a = F_1(A,B,C,D) = \sum m(0,2,3,5,6,7,8,9)$
$\boldsymbol{b = F_2(A,B,C,D) = \sum m(0,1,2,3,4,7,8,9)}$
$\boldsymbol{c = F_3(A,B,C,D) = \sum m(0,1,3,4,5,6,7,8,9)}$
$d = F_4(A,B,C,D) = \sum m(0,2,3,5,6,8,)$
$e = F_5(A,B,C,D) = \sum m(0,2,6,8,)$
$\boldsymbol{f = F_6(A,B,C,D) = \sum m(0,4,5,6,8,9)}$
$\boldsymbol{g = F_7(A,B,C,D) = \sum m(2,3,4,5,6,8,9).}$

### Displaying digit '4' using BCD to decimal decoder(IC 7442)




### Displaying digit '4' using BCD to 7- segment Decoder(IC 7447)



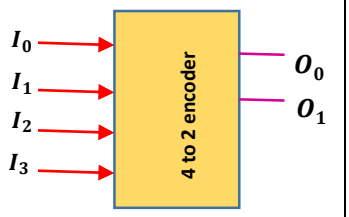
**ENCODER:** An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of $2^n$ input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes $2^n$ input lines with 'n' bits. It is optional to represent the enable signal in encoders.

**4 to 2 Encoder**Let 4 to 2 Encoder has four inputs $I_3$, $I_2$, $I_1$ & $I_0$ and two outputs $O_1$ & $O_0$. The **block diagram** of 4 to 2 Encoder is shown in the following figure 1 .At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.

The **Truth table** of 4 to 2 encoder is shown below.

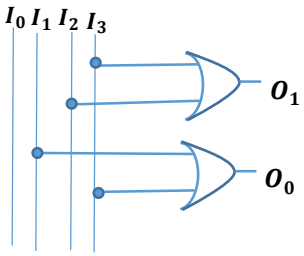
| Inputs | | | | Outputs | |
|---|---|---|---|---|---|
| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

**figure 1**




**From truth table we get**

$O_0 = I_1 + I_3$ & $O_1 = I_2 + I_3$




**8 to 3 Encoder:**

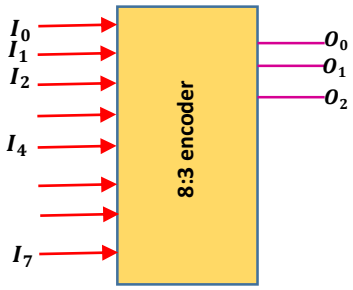
| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

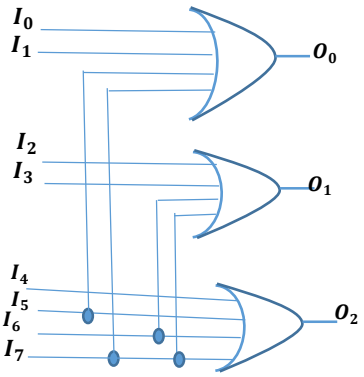
**from truth table we get**

$O_0 = I_1 + I_3 + I_5 + I_7$

$O_1 = I_2 + I_3 + I_6 + I_7$

& $O_2 = I_4 + I_5 + I_6 + I_7$




**BCD Encoder:**

| Inputs | | | | | | | | | | Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I_9$ | $I_8$ | $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

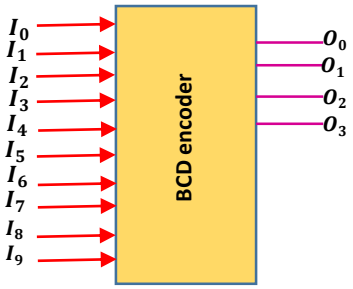
**from truth table we get**

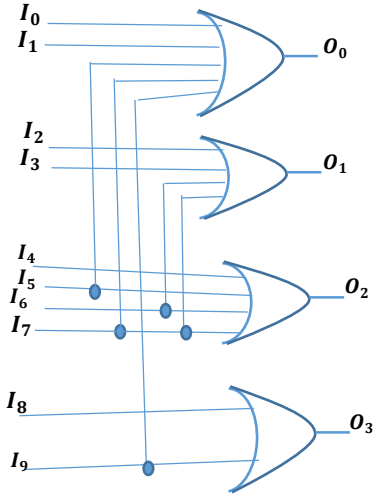
$O_0 = I_1 + I_3 + I_5 + I_7$

$O_1 = I_2 + I_3 + I_6 + I_7$

& $O_2 = I_4 + I_5 + I_6 + I_7$

& $O_2 = I_8 + I_9$



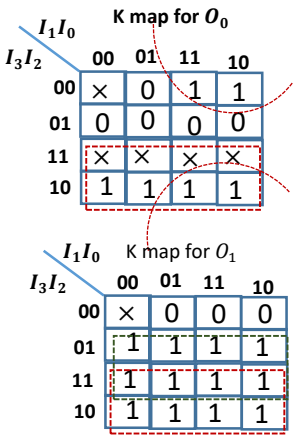
**PRORITY ENCODER:** A priority encoder provide n bits of binary coded output representing the position of the highest order active input of $2^n$ inputs. If two or more inputs are high at the same time, the input having the highest priority will take precedence.

**4 to 2 priority encoder**

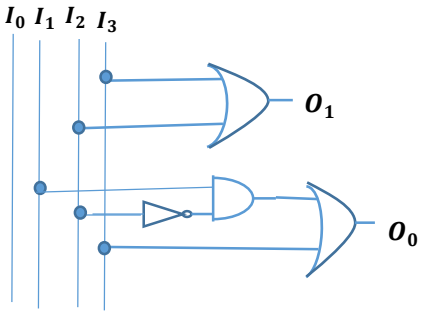
A 4-to-2 priority encoder takes 4 input bits and produces 2 output bits. In this truth table, for all the non-explicitly defined input combinations (i.e. inputs containing 2, 3, or 4 high bits) the lower priority bits are shown as don't cares (X). Similarly when the inputs are 0000, the outputs are not valid and therefore they are XX. **Note that we take that $I_3$ have the highest priority and $I_0$ have lowest priority**

**Truth table for 4:2 priority encoder**

| Inputs | | | | outputs | |
|---|---|---|---|---|---|
| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | × | × |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | × | 0 | 1 |
| 0 | 1 | × | × | 1 | 0 |
| 1 | × | × | × | 1 | 1 |




Then $O_0 = I_3 + I_1\overline{I_2}$ & $O_1 = I_3 + I_2$




## PARITY GENERATOR AND PARITY CHECKER:

What is Parity Bit?

The parity generating technique is one of the most widely used error detection techniques for the data transmission. In digital systems, when binary data is transmitted and processed, data may be subjected to noise so that such noise can alter 0's (of data bits) to 1's and 1's to 0's.

Hence, **parity bit** is added to the word containing data in order to make number of 1's either even or odd. Thus it is used to detect errors, during the transmission of binary data .The message containing the data bits along with parity bit is transmitted from transmitter node to receiver node.

At the receiving end, the number of 1's in the message is counted and if it doesn't match with the transmitted one, then it means there is an error in the data.

**Parity generator and checker**

A parity generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called parity checker. A combined circuit or devices of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data word.

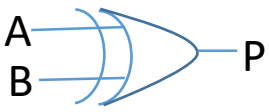
The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1's an even amount whereas in odd parity the added parity bit will make the total number of 1's odd amount.

The basic principle involved in the implementation of parity circuits is that sum of odd number of 1's is always 1 and sum of even number of 1's is always zero. Such error detecting and correction can be implemented by using Ex-OR gates (since Ex-OR gate produce zero output when there are even number of inputs).

To produce two bits sum, one Ex-OR gate is sufficient whereas for adding three bits two Ex-OR gates are required as shown in below figure.

**Even parity generator for two inputs**

| A | B | P |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



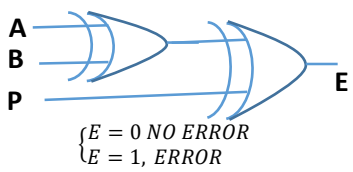

**Odd parity generator for two inputs**

| A | B | P |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## Even parity checker for two inputs

| A | B | P | E |
|---|---|---|---|
| 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 |

Here $F(A,B,P) = \sum m(1,2,4,7)$




$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$

## Odd parity checker for two inputs

| A | B | P | E |
|---|---|---|---|
| 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 |

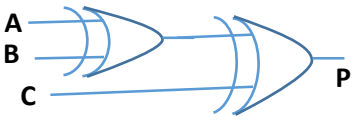
Here $F(A,B,P) = \sum m(0,3,5,6)$


$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$

## Even parity generator for three inputs

| A | B | C | 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 |

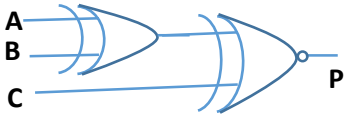
Here $F(A,B,P) = \sum m(1,2,4,7)$




## Odd parity generator for three inputs

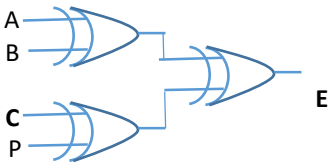
| A | B | C | P |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Here $F(A,B,P) = \sum m(0,3,5,6)$




## Even parity checker for three inputs

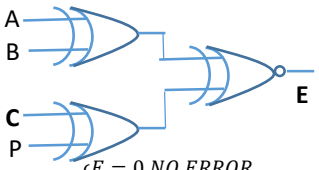
| A | B | C | P | E |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |




$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$

## Odd parity checker for three inputs

| A | B | C | P | E |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

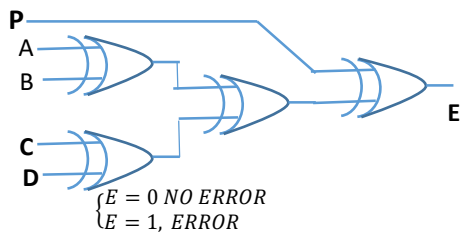



$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$

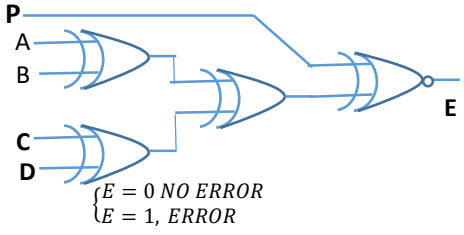
**HOME TUSK (i) Even/ Odd parity generator for four inputs (ii) Even/ Odd parity checker for four inputs**

**Hints: (i) see Even parity checker for three inputs**

**(ii)**




$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$




$\begin{cases} E = 0 \; NO \; ERROR \\ E = 1, \; ERROR \end{cases}$