KTU NOTES

The learning companion.

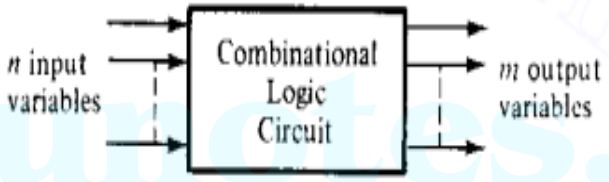
KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

MODULE 3- Binary Adders and Subtractors

Combinational circuit

Combinational circuit is a circuit in which we combine the different gates in the circuit. Some of the characteristics of combinational circuits are following −

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do 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 an n number of inputs and m number of outputs.
- Examples of Combinational Circuits: Half Adder, Full Adder, Half Subtractor, Full Subtractor



Half Adder

- Half adder is a combinational logic circuit with two inputs and two outputs.
- The half adder circuit is designed to add two single bit binary numbers.
- It is the basic building block for addition of two single bit numbers.
- This circuit has two outputs carry and sum.

- Block Diagram

Design of a Half Adder

Step-01:

Identify the input and output variables-

- Input variables = A, B (either 0 or 1)
- Output variables = S, C where S = Sum and C = Carry
-

Step-02:

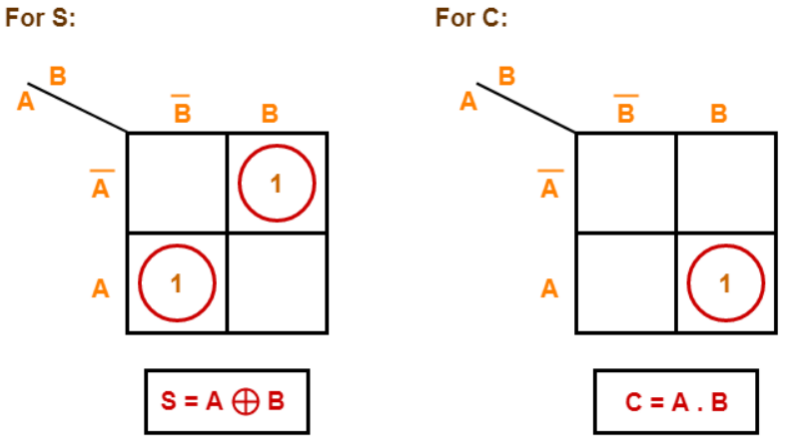
Draw the truth table-

Inputs		Outputs	
A	B	C (Carry)	S (Sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table

Step-03:

Draw K-maps using the above truth table and determine the simplified Boolean expressions-



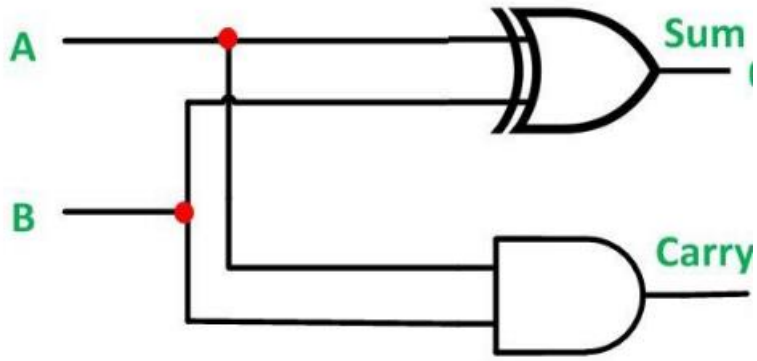
K Maps

eri

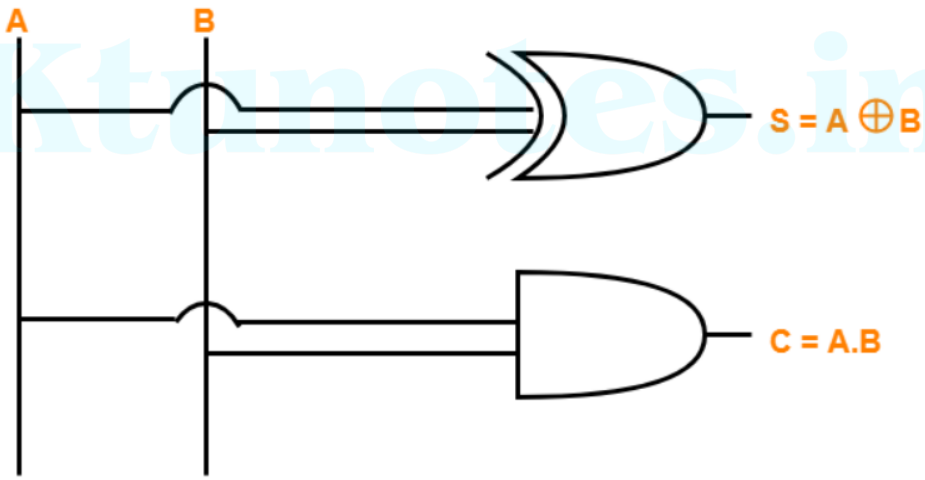
Step-04:

Draw the logic diagram.

Implementation of half adder using XOR gate and AND gate is as shown below-

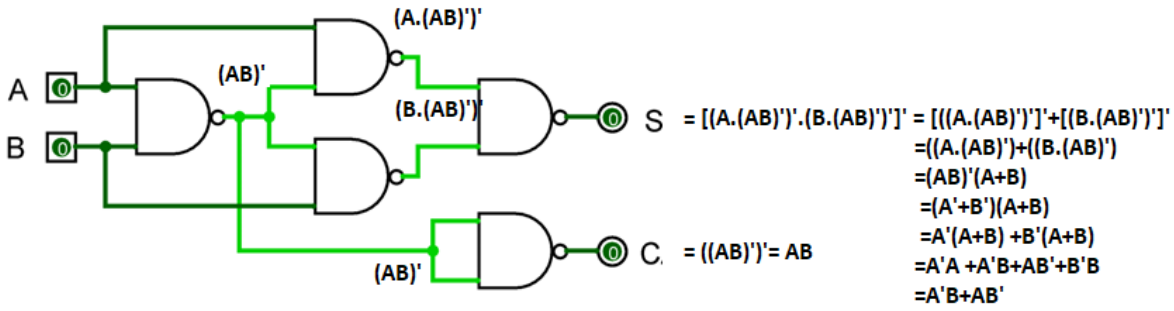


Or

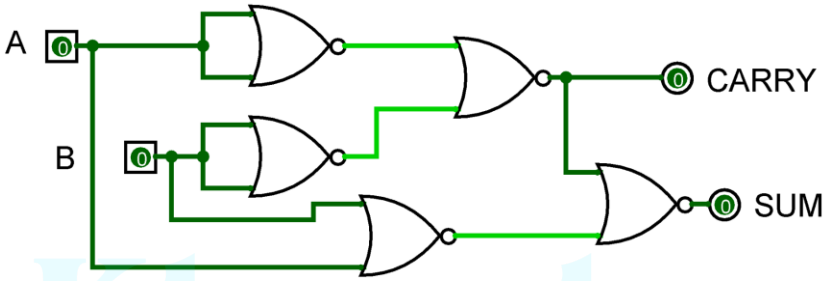


Half Adder Logic Diagram

Implementation of half adder using NAND gates

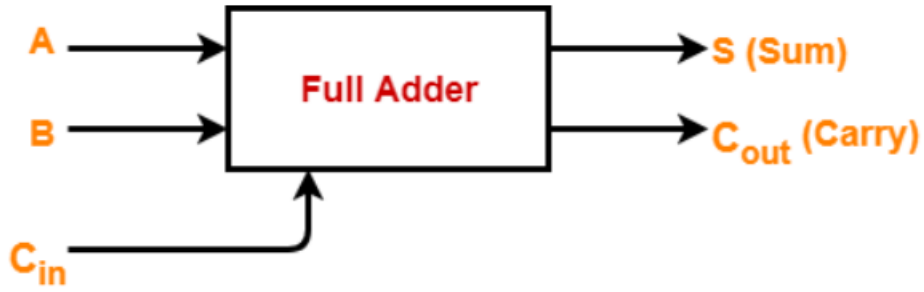


Implementation of half adder using NOR gates



Full Adder

- Full adder is developed to overcome the drawback of Half Adder circuit.
- It can add three one-bit numbers.
- The full adder is a three input and two output combinational circuit.
- Block diagram



Full adder is designed in the following steps-

Step-01:

Identify the input and output variables-

- Input variables = A, B, C_{in} (either 0 or 1)
- Output variables = S, C_{out} where S = Sum and C_{out} = Carry

Step-02:

Draw the truth table-

Inputs			Outputs	
A	B	C_{in}	C_{out} (Carry)	S (Sum)
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

Truth Table

Step-03:

Draw K-maps using the above truth table and determine the simplified Boolean expressions-

For S:

$$S = A \oplus B \oplus C_{in}$$

$S = A' B' C_{IN} + A' B C'_{IN} + A B' C'_{IN} + A B C_{IN}$

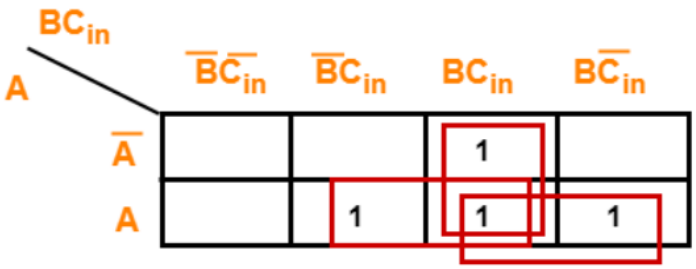
$= C_{IN} (A' B' + A B) + C'_{IN} (A' B + A B')$

$= C_{IN}$ (A Ex-NOR B) + C_{IN} (A Ex-OR B)

$= C_{IN} (A \oplus B)' + C_{IN} (A \oplus B)$

$= C_{IN} \oplus (A \oplus B)$

For C_{in}:

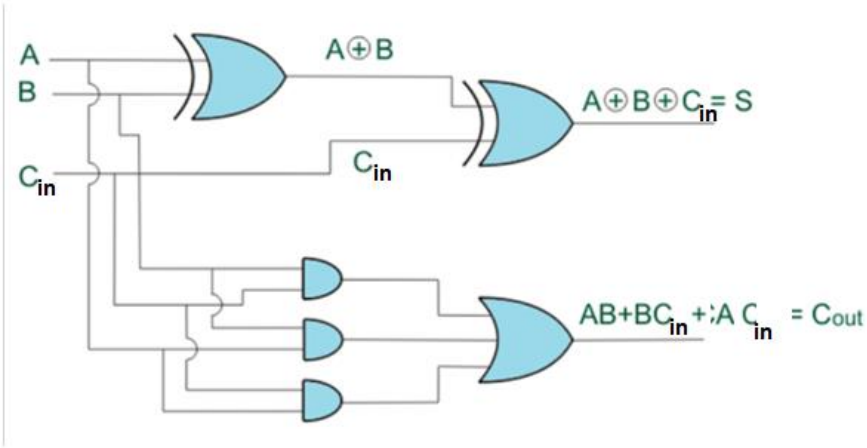


$$C_{out} = AB + BC_{in} + C_{in}A$$

Step-04:

Draw the logic diagram.

The implementation of full adder using 1 XOR gate, 3 AND gates and 1 OR gate is as shown below-



Implementation of Full Adder using Half Adders is as shown below-

$C_{OUT} = A\ B + A\ C_{IN} + B\ C_{IN}$

$= A\ B + A\ C_{IN} + B\ C_{IN}\ (A + A')$

$= A\ B + A\ C_{IN} + A\ B\ C_{IN} + A'\ B\ C_{IN}$

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

$= A\ B + A\ C_{IN} + A'\ B\ C_{IN}$

$= A\ B + A\ C_{IN}\ (B + B') + A'\ B\ C_{IN}$

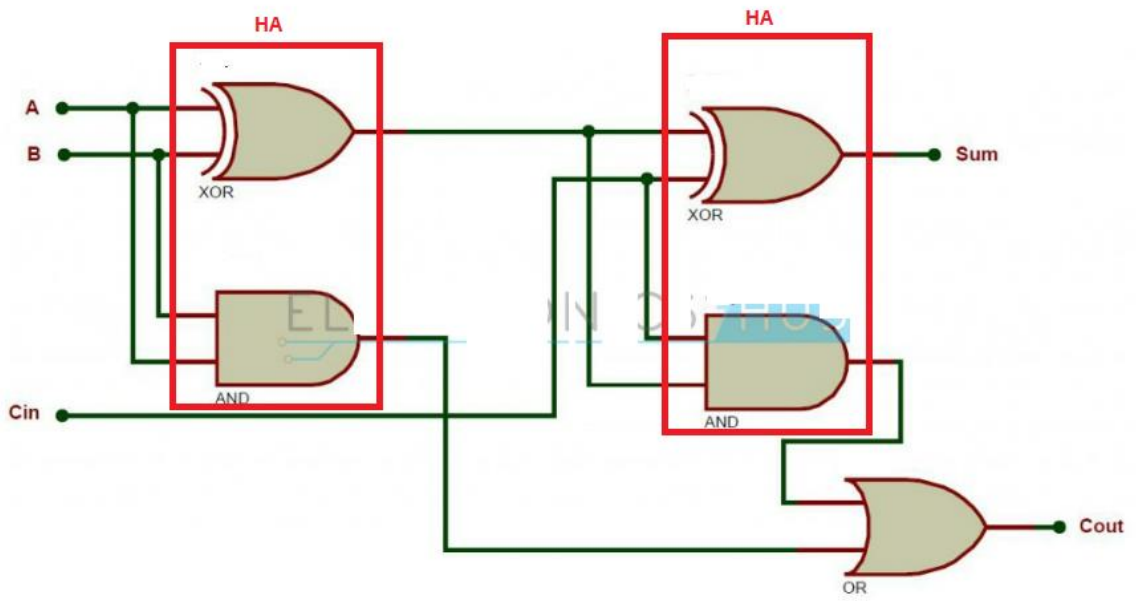
$= A\ B + A\ B\ C_{IN} + A\ B'\ C_{IN} + A'\ B\ C_{IN}$

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

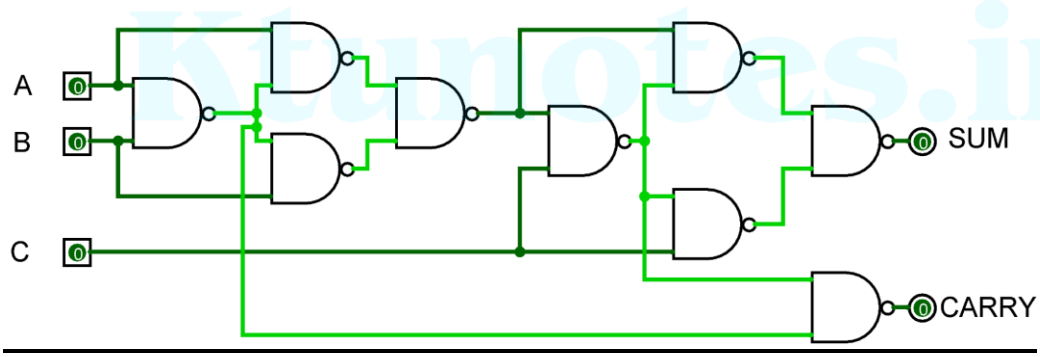
$= A\ B + C_{IN}\ (A\ B' + A'\ B)$

$= A\ B + C_{IN}\ (A\ Ex\text{-}OR\ B)$

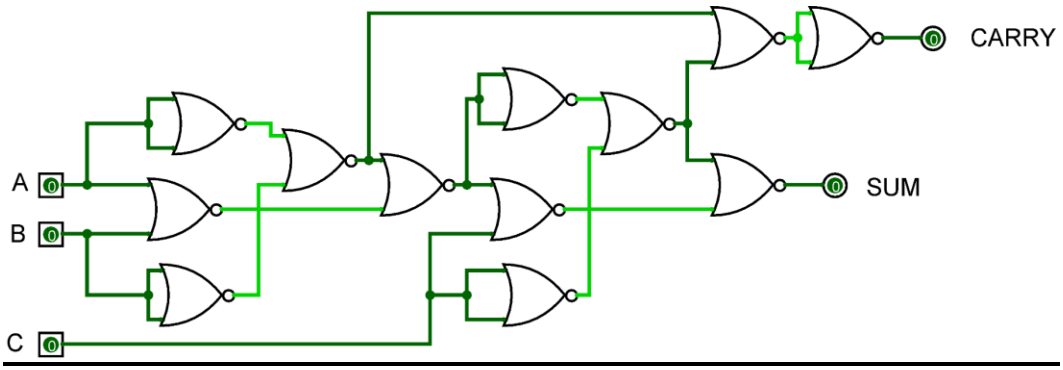
$= A\ B + C_{IN}\ (A \oplus B)$



Implementation of Full Adder using NAND gates

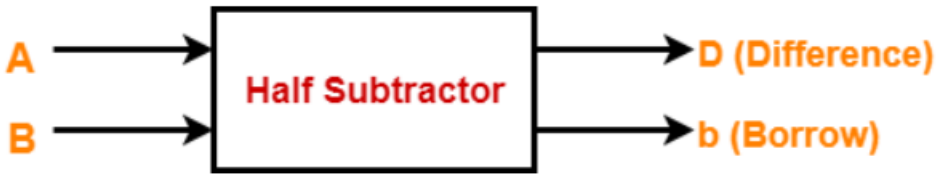


Implementation of Full Adder using NOR gates



Half Subtractor

- Half Subtractor is a combinational logic circuit.
- It is used for the purpose of subtracting two single bit numbers.
- It contains 2 inputs and 2 outputs (difference and borrow).



Designing a Half Subtractor

Step-01:

Identify the input and output variables-

- Input variables = A, B (either 0 or 1)
- Output variables = D, b where D = Difference and b = borrow

Step-02:

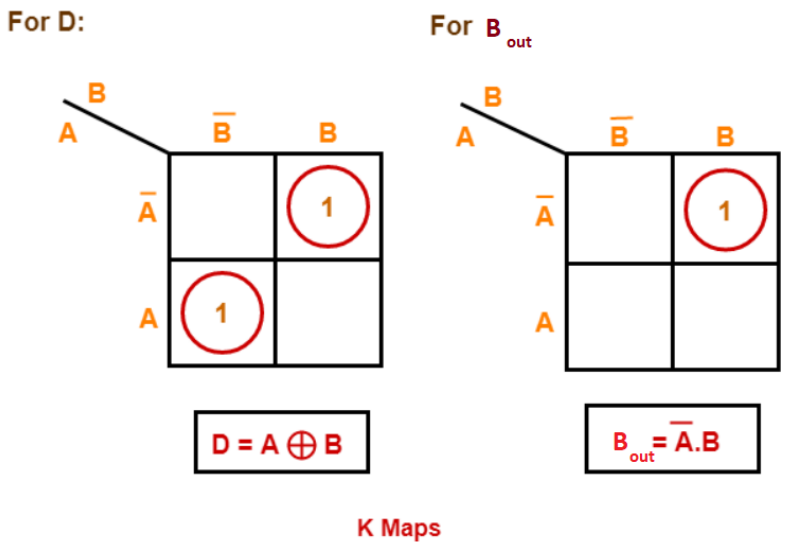
Draw the truth table-

Inputs		Outputs	
A	B	D (Difference)	B_{out} (Borrow)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table

Step-03:

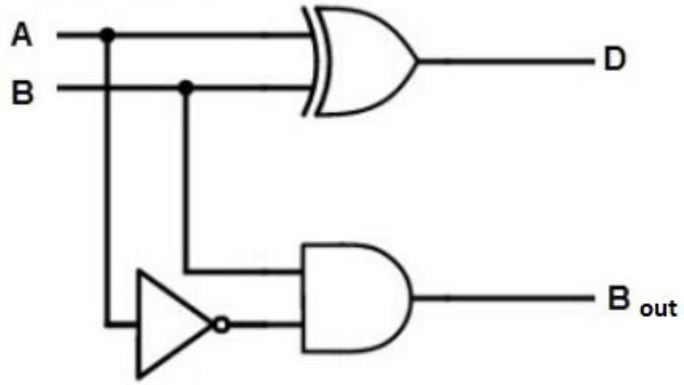
Draw K-maps using the above truth table and determine the simplified Boolean expressions-



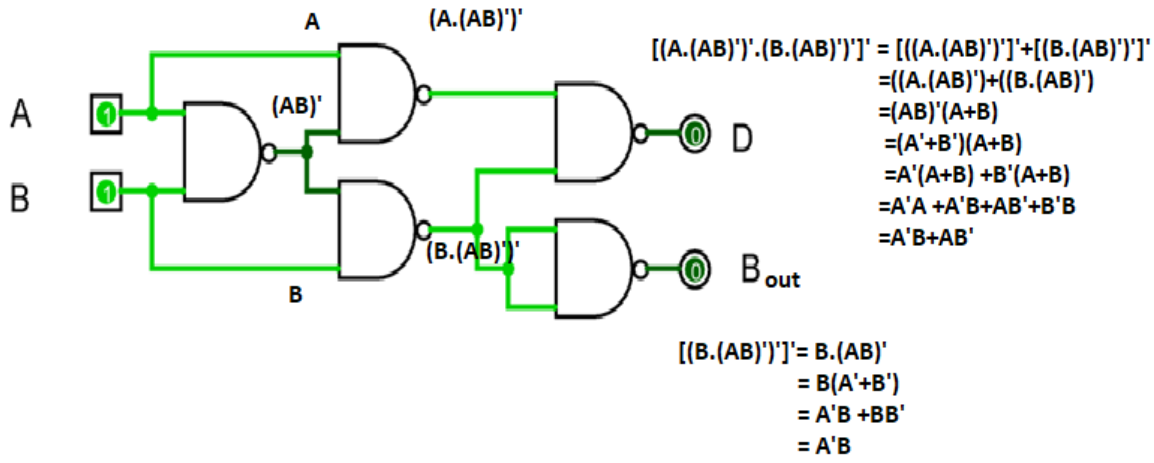
Step-04:

Draw the logic diagram

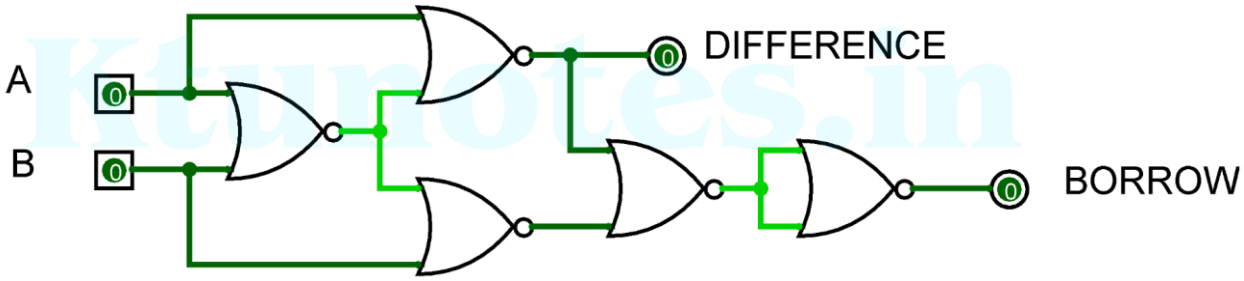
The implementation of half subtractor using 1 XOR gate, 1 NOT gate and 1 AND gate is as shown below-



implementation of half subtractor using NAND gates

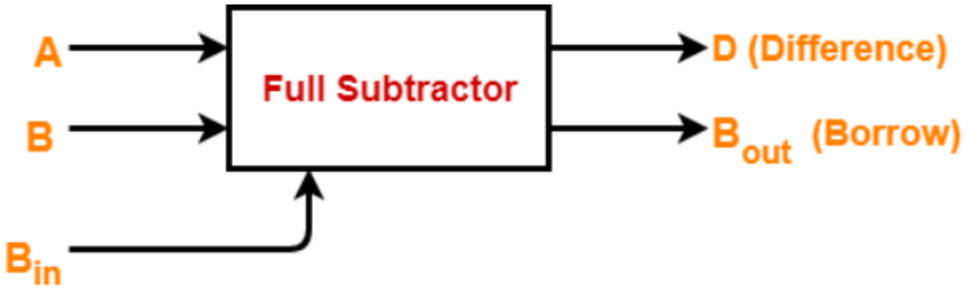


implementation of half subtractor using NOR gates



Full Subtractor

- Full Subtractor is a combinational logic circuit.
- It is used for the purpose of subtracting two single bit numbers. It also takes into consideration borrow of the lower significant stage.
- Thus, full subtractor has the ability to perform the subtraction of three bits.
- Full subtractor contains 3 inputs and 2 outputs (Difference and Borrow) as shown-



Designing a Full Subtractor-

Full subtractor is designed in the following steps-

Step-01:

Identify the input and output variables-

- Input variables = A, B, B_{in} (either 0 or 1)
- Output variables = D, B_{out} where D = Difference and B_{out} = Borrow

Step-02:

Draw the truth table-

Inputs			Outputs	
A	B	B_{in}	B_{out} (Borrow)	D (Difference)
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

Truth Table

Step-03:

Draw K-maps using the above truth table and determine the simplified Boolean expressions-

For D:

$$D = A \oplus B \oplus B_{in}$$

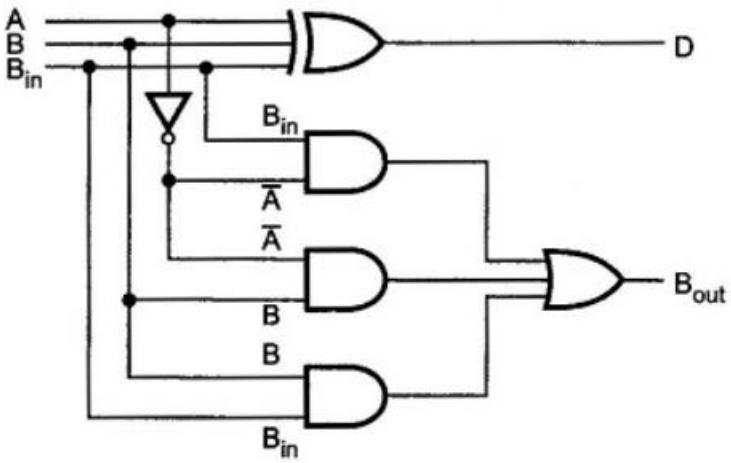
$$\begin{aligned} D &= \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in} \\ &= B_{in}(\bar{A}\bar{B} + AB) + \bar{B}_{in}(\bar{A}B + A\bar{B}) \\ &= B_{in}(A \odot B) + \bar{B}_{in}(A \oplus B) \\ &= B_{in}(\overline{A \oplus B}) + \bar{B}_{in}(A \oplus B) \\ &= B_{in} \oplus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

For B_{in}:

$$B_{out} = \bar{A}B + (\bar{A} + B)B_{in}$$

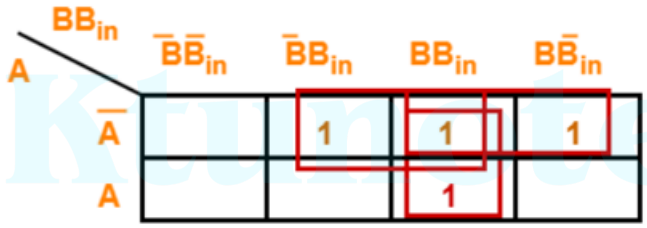
Step-04:

Draw the logic diagram.



Full Subtractor using Half Subtractor

For B_{in}:

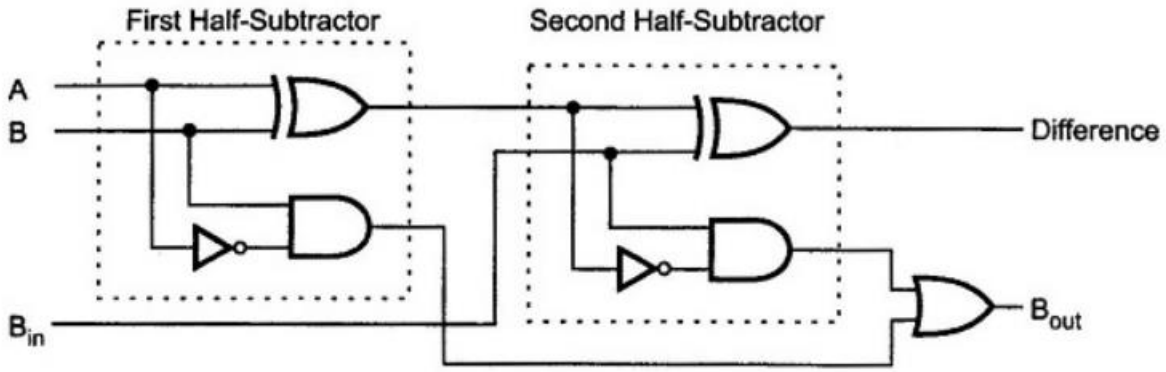


D = (A XOR B) XOR Bin

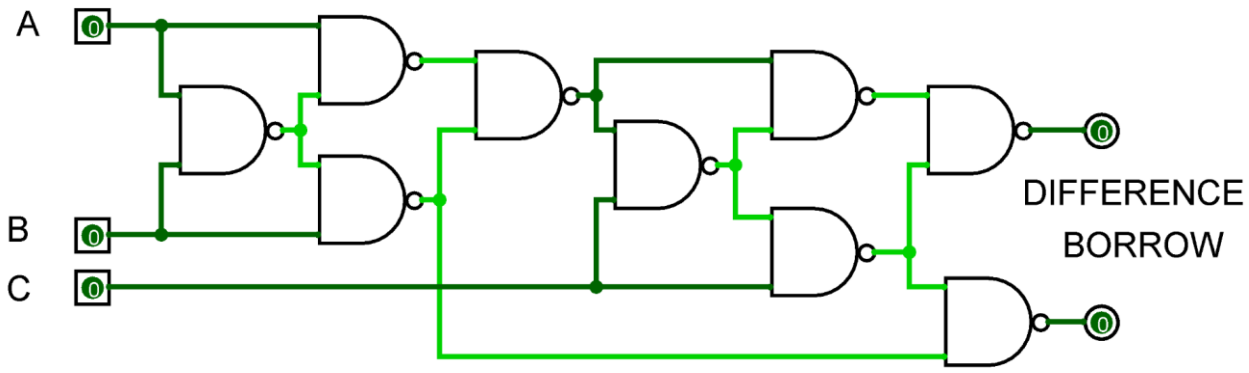
Bin = A'B'Bin +A'BBin + A'BB'in + ABBin

= A'B (Bin + B'in) + Bin(A'B'+AB)

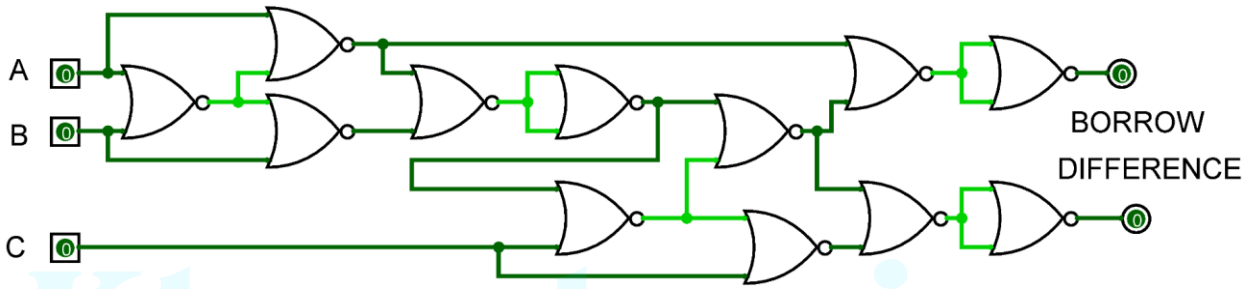
= A'B + Bin (A XOR B)'



Full Subtractor using NAND gates



Full Subtractor using NAND gates



Binary to BCD Code Convertor

Input variables are B3, B2, B1, B1 representing a 4 bit Binary number.

Output variables are D4, D3, D2, D1, D0, representing its equivalent BCD.

Since highest decimal value of 4 bit binary number is 15, 5 bits are required to represent its BCD. D4 for the first digit, and D3D2D1D0 for the second digit

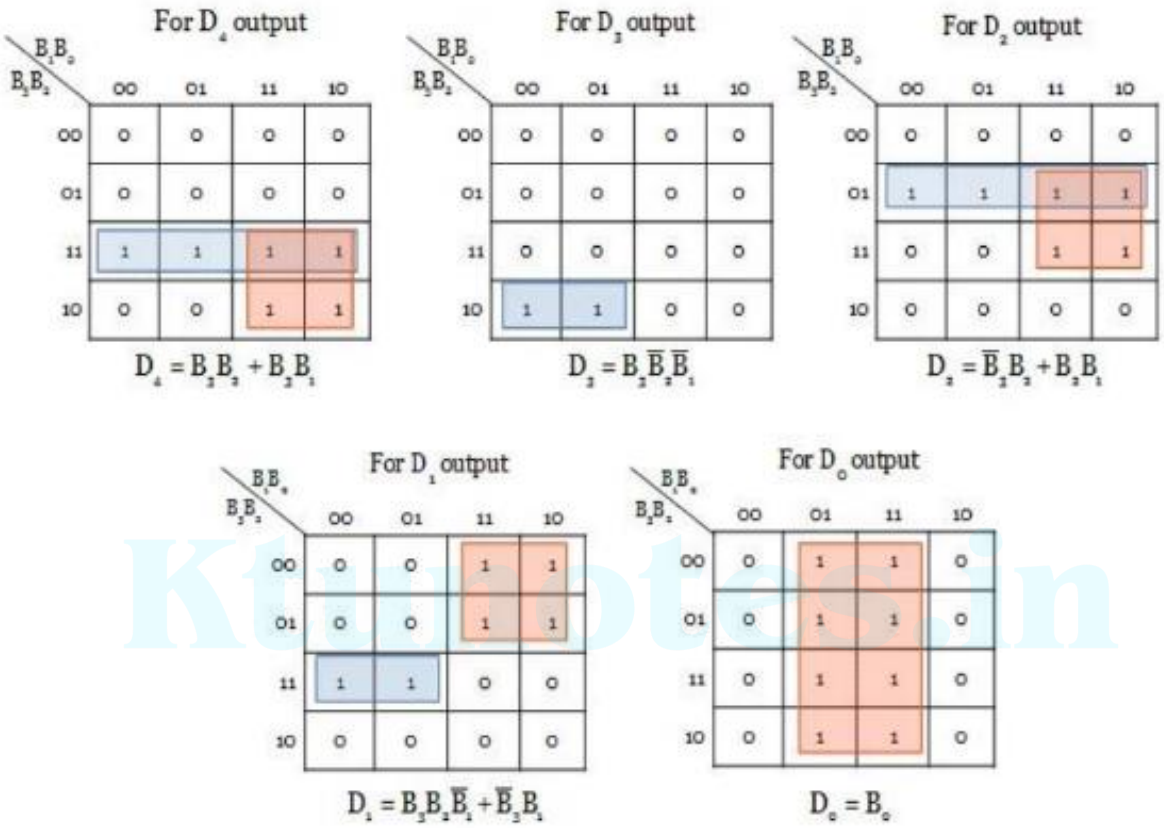
Truth Table

Decimal Number	Binary code (Input)				BCD code (Output)				
	B3	B2	B1	B0	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

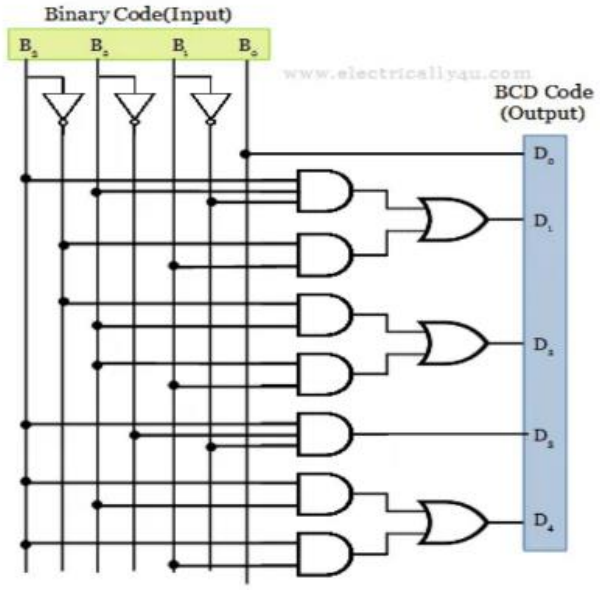
K-maps

4 input variables, so 4 variable K map with 16 cells.

Separate Kmaps for 5 output variables



Logic Circuit Diagram



$$D_4 = B_3B_2 + B_3B_1$$

$$D_3 = B_3\overline{B}_2\overline{B}_1$$

$$D_2 = \overline{B}_3B_2 + B_2B_1$$

$$D_1 = B_3B_2\overline{B}_1 + \overline{B}_3B_1$$

$$D_0 = B_0$$

BCD to Excess 3 Convertor

Valid BCD is only from 0 to 9 (10 to 15 are invalid BCD)

Excess 3 code is obtained by adding 3 (0011) to the BCD code. Since BCD 10 to 15 are invalid, corresponding entries in kmap are marked with don't cares (X)

Truth Table:

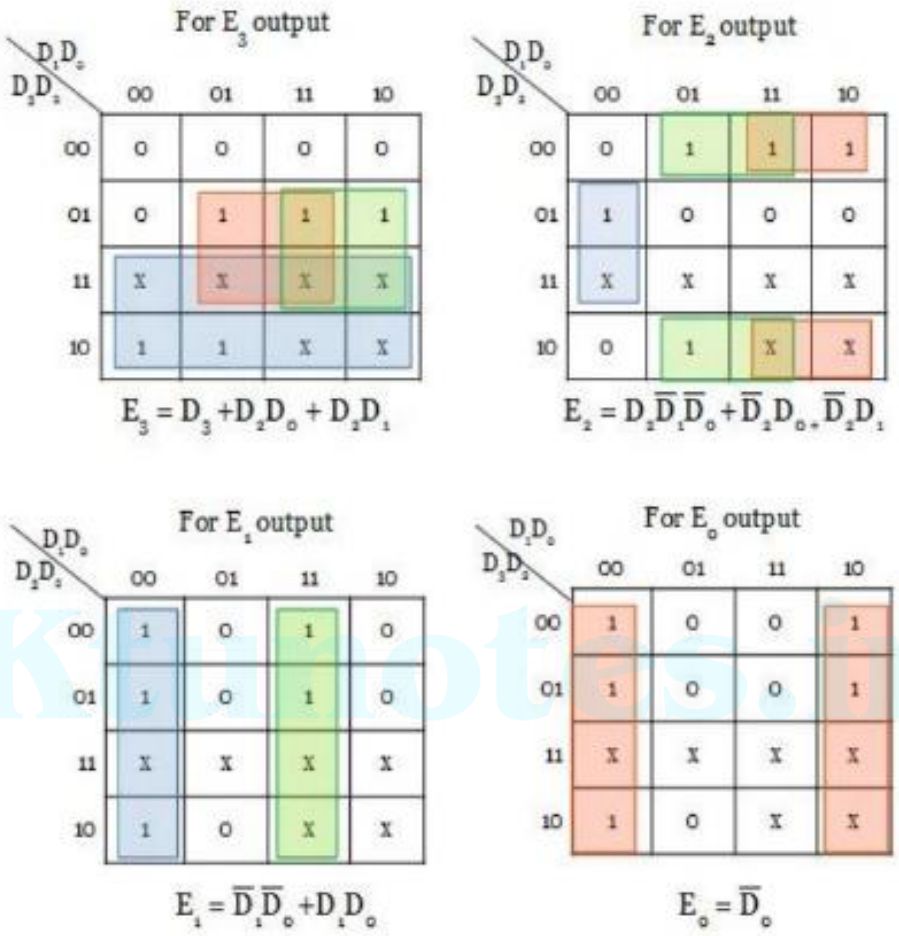
Decimal Number	BCD code (Input)				Excess-3 code (Output)			
	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

i

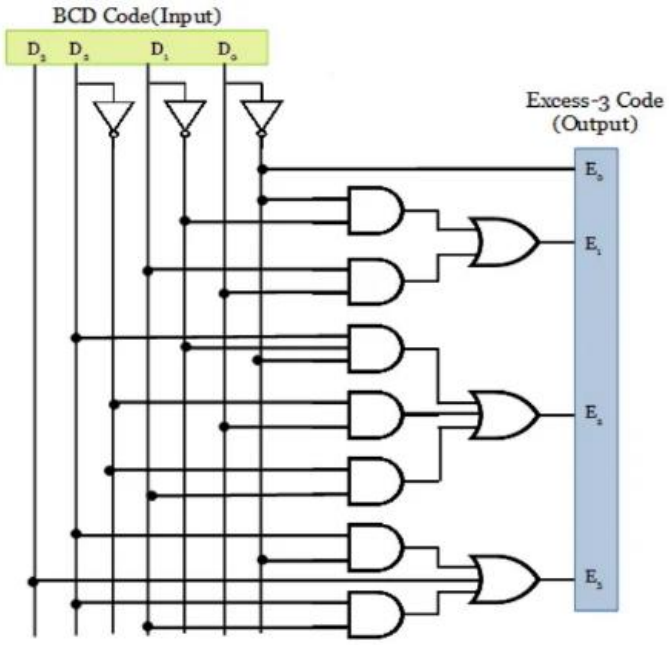
K-maps

4 input variables, so 4 variable K map with 16 cells.

Separate Kmaps for 4 output variables



Logic Circuit Diagram



$$E_3 = D_3 + D_2D_0 + D_2D_1$$

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

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

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

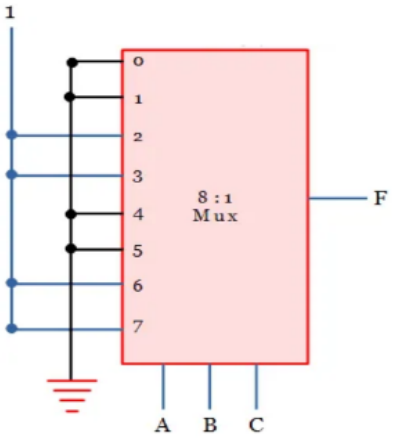
Implementation of Boolean function in multiplexer

Problem #1

Implement the boolean expression F(A, B, C) = ∑ m(2, 3, 6, 7) using a multiplexer

Solution

- There are 3 variables in the given expression, hence $2^n = 2^3 = 8 : 1$ multiplexer.
- So, the mux has 8 input lines, 3 selection lines, and one output.
- The inputs, corresponding to the minterms (2, 3, 6, 7) are connected to logic 1 and the remaining terms to logic 0(grounded).
- The given input variables are connected as three selection lines

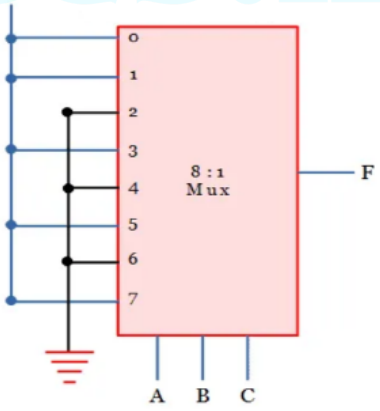


Problem #2

Implement the boolean expression F(A, B, C) = ∑ m(0, 1, 3, 5, 7) using a multiplexer.

Solution

- there are 3 variables and hence 8 : 1 multiplier is used to solve the expression.
- The three input variables(A, B, C) are connected as three selection lines.
- The inputs, corresponding to the minterms (0, 1, 3, 5, 7) are connected to logic 1 and the remaining terms to the logic 0(grounded).



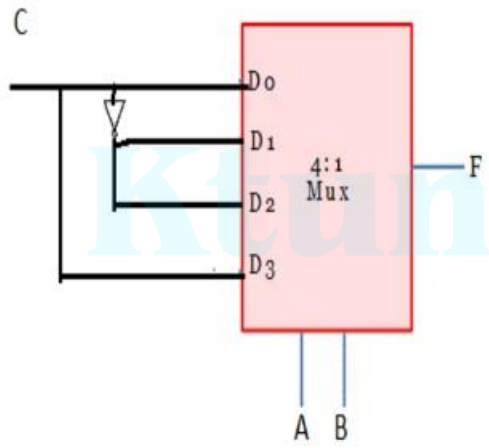
Problem #3

Implement the boolean expression F(A, B, C) = ∑ m(1, 2, 4, 7) using 4 : 1 multiplexer.

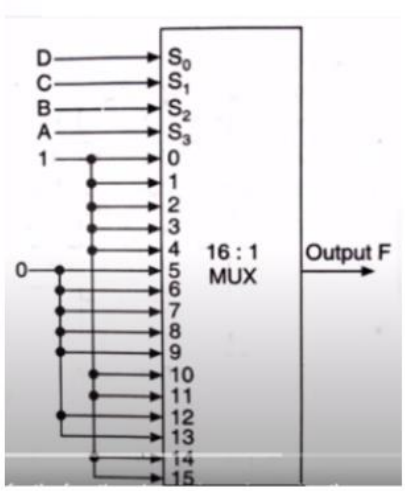
Solution

- In the given boolean expression, there are 3 variables. We should use 2^3 : 1 = 8 : 1 multiplexer. But as per the question, it is to be implemented with 4 : 1 mux.
- For 4 : 1 multiplexer, there should be 2 selection lines. So from the given 3 variables, the 2 variables(A, B) are used as selection line inputs.
- Let us derive the four inputs of 4 : 1 multiplexer using the implementation table

A	B	C	F	
0	0	0	0	F = C
0	0	1	1	
0	1	0	1	F = C'
0	1	1	0	
1	0	0	1	F = C'
1	0	1	0	
1	1	0	0	F = C
1	1	1	1	

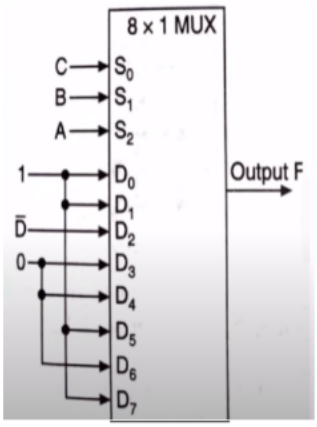


Problem #4
Implement the
F(A, B, C) = ∑ m (0, 1, 2, 3, 4, 10, 11,14, 15)
using 16 : 1 multiplexer.
Solution



Minterm	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

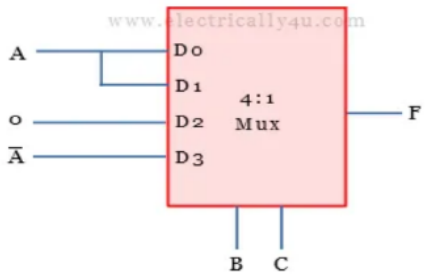
Problem #4
Implement the
F(A, B, C) = ∑ m (0, 1, 2, 3, 4, 10, 11,14, 15)
using 8 : 1 multiplexer.
Solution



MINTERM	A	B	C	D	F	
0	0	0	0	0	1	
1	0	0	0	1	1	F = 1 D0
2	0	0	1	0	1	
3	0	0	1	1	1	F = 1 D1
4	0	1	0	0	1	
5	0	1	0	1	0	F = D' D2
6	0	1	1	0	0	
7	0	1	1	1	0	F = 0 D3
8	1	0	0	0	0	
9	1	0	0	1	0	F = 0 D4
10	1	0	1	0	1	
11	1	0	1	1	1	F = 1 D5
12	1	1	0	0	0	
13	1	1	0	1	0	F = 0 D6
14	1	1	1	0	1	
15	1	1	1	1	1	F = 1 D7

Problem #5

For the given multiplexer circuit, determine the logic function.



Solution:

- In the given multiplexer circuit, B and C are the selection inputs. Hence the possible input combinations are 00, 01, 10 and 11.
- if BC = 00, D_0 = A ,is selected as the output.
- if BC = 01, D_1 = A, is selected as the output.
- If BC = 10, D_2 = 0, is selected as the output.
- If BC = 11, D_3 = A' is selected as the output.

B	C	F
0	0	A
0	1	A
1	0	0
1	1	$\bar{A}$

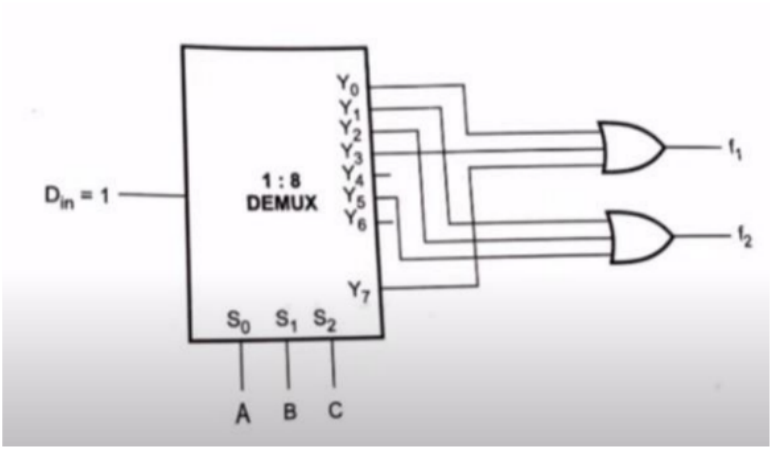
- From the truth table, the output expression can be written as F = AB'C' + AB'C + A'BC
  = AB' (C' + C) + A'BC
  = AB' + A'BC

Problem #6

Implement the functions F1 = ∑ m (0, 3, 7), F2 = ∑ m (1, 2, 5) using demultiplexer.

Solution

- Largest term in the functions is 7 = $(111)_2$. Hence we require 3 selection lines and a 1:8 demultiplexer.
- Consider A, B, C as the three variables used as selection lines.

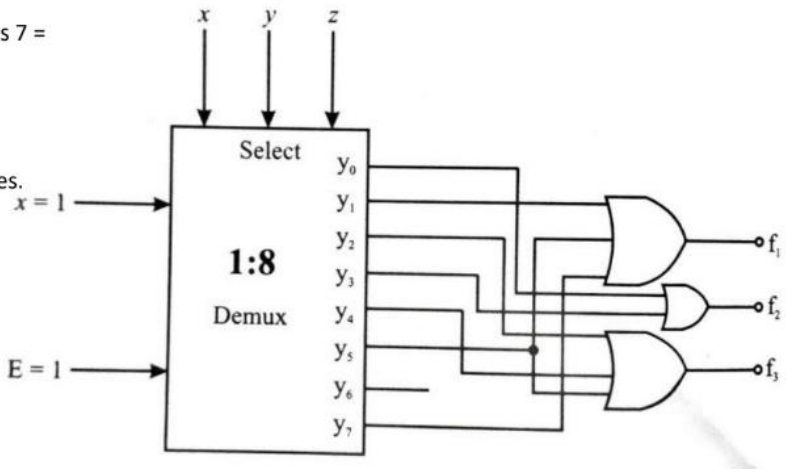


Problem #7

Implement the functions F1 = ∑ m (1, 5, 7), F2 = ∑ m (0, 3), F3 = ∑ m (2, 4, 5) using demultiplexer.

Solution

- Largest term in the functions is 7 = $(111)_2$. Hence we require 3 selection lines and a 1:8 demultiplexer.
- Consider x, y, z as the three variables used as selection lines.

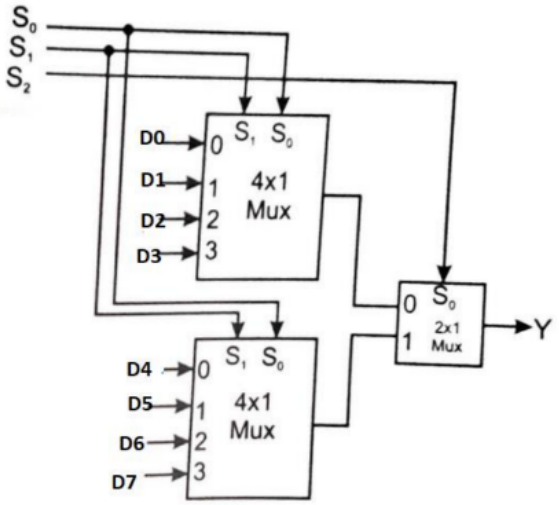


Problem #8

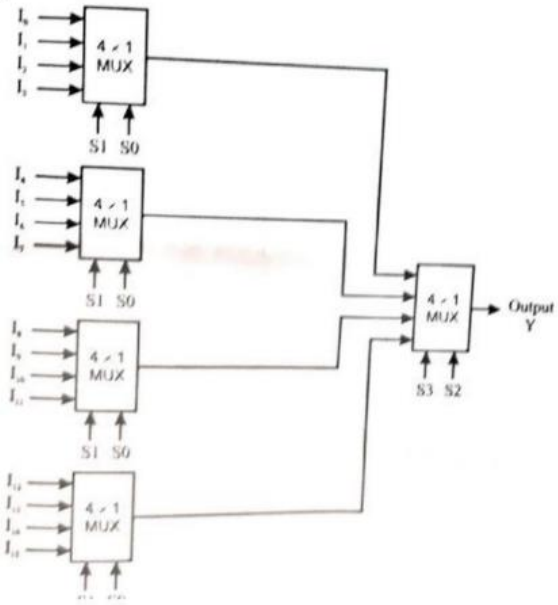
Configure an 8:1 Multiplexer using 4 : 1 MUX and 2 : 1 MUX

Solution

D7	D6	D5	D4	D3	D2	D1	D0	S2	S1	S0	Y
0	0	0	0	0	0	0	1	0	0	0	D0
0	0	0	0	0	0	1	0	0	0	1	D1
0	0	0	0	0	1	0	0	0	1	0	D2
0	0	0	0	1	0	0	0	0	1	1	D3
0	0	0	1	0	0	0	0	1	0	0	D4
0	0	1	0	0	0	0	0	1	0	1	D5
0	1	0	0	0	0	0	0	1	1	0	D6
1	0	0	0	0	0	0	0	1	1	1	D7

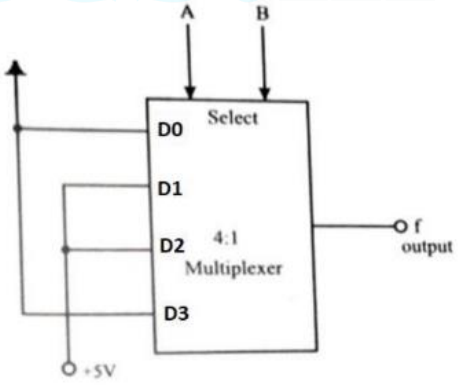


Problem #9
Configure an 16 : 1 Multiplexer using 4 : 1 MUX
Solution

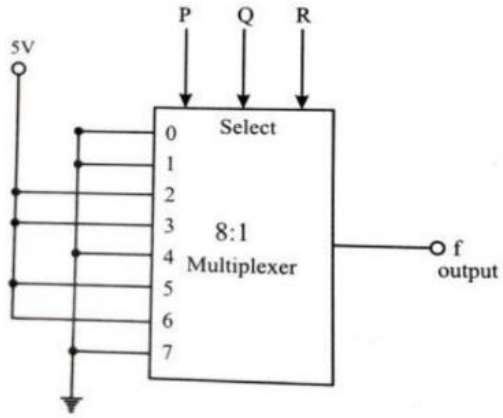


Problem #10
Implement the boolean expression F(A, B) = A'B + AB' using a multiplexer
Solution

A	B	D	F
0	0	D0	0
0	1	D1	1
1	0	D2	1
1	1	D3	0



Problem #11
Implement the boolean expression F(P, Q, R) = ∑ m (2, 3, 5, 6), using a multiplexer
Solution

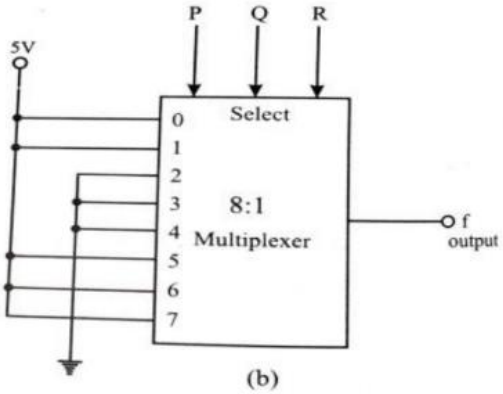


Problem #12
Implement the boolean expression F(P, Q, R) = P'Q'+ PR + PQ using a multiplexer
Solution

F(P, Q, R) = P'Q'+ PR + PQ
=P'Q'(R+R') + PR(Q + Q') +PQ (R + R')
=P'Q'R + P'Q'R' + PQR + PQ'R + PQR + PQR'
= P'Q'R + P'Q'R' + PQR + PQ'R + PQR'

P	Q	R	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)

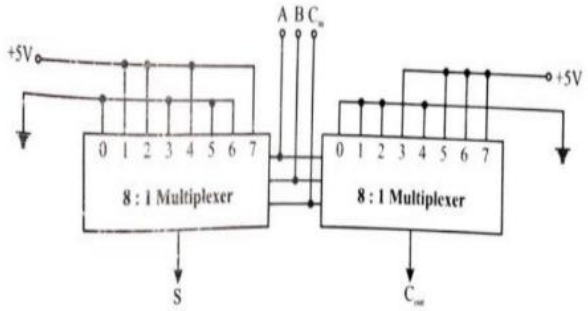
Problem #13
Implement a full adder using a multiplexer

S = A ⊕ B ⊕ C
= AB'C' + A'B'C + A'BC' + ABC
= ∑ m (1, 2, 4, 7)

Cout = AB + ACin + Bcin
= AB(Cin+ C'in) + ACin (B +B') + BCin(A+ A')
= ABCin + ABC'in + ABCin + AB'Cin + ABCin + A'Bcin
= ABCin + ABC'in + AB'Cin + A'Bcin
= ∑ m (3, 5, 6, 7)

A	B	Cin	S
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

A	B	Cin	Cout
0	0	0	0
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	1



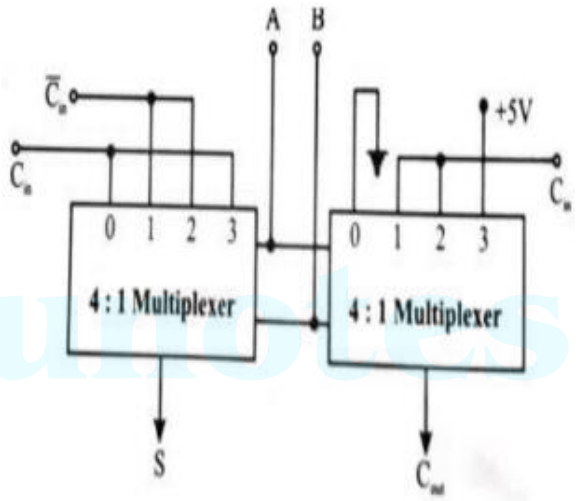
Problem #14
Implement a full adder using 4 : 1 multiplexers

S = A ⊕ B ⊕ C
= AB'C' + A'B'C + A'BC' + ABC
= ∑ m (1, 2, 4, 7)

A	B	Cin	S		
0	0	0	0	S = Cin	D0
0	0	1	1		
0	1	0	1	S = C'in	D1
0	1	1	0		
1	0	0	1	S = C'in	D2
1	0	1	0		
1	1	0	0	S = Cin	D3
1	1	1	1		

Cout = AB + ACin + Bcin
= AB(Cin+ C'in) + ACin (B +B') + BCin(A+ A')
= ABCin + ABC'in + ABCin + AB'Cin + ABCin + A'Bcin
= ABCin + ABC'in + AB'Cin + A'Bcin
= ∑ m (3, 5, 6, 7)

A	B	Cin	Cout		
0	0	0	0	Cout = 0	D0
0	0	1	0		
0	1	0	0	Cout = Cin	D1
0	1	1	1		
1	0	0	0	Cout = Cin	D2
1	0	1	1		
1	1	0	1	Cout = 1	D3
1	1	1	1		



Problem #15

Realize the boolean expression $F(A, B, C) = \sum m(0, 1, 4, 6)$ using

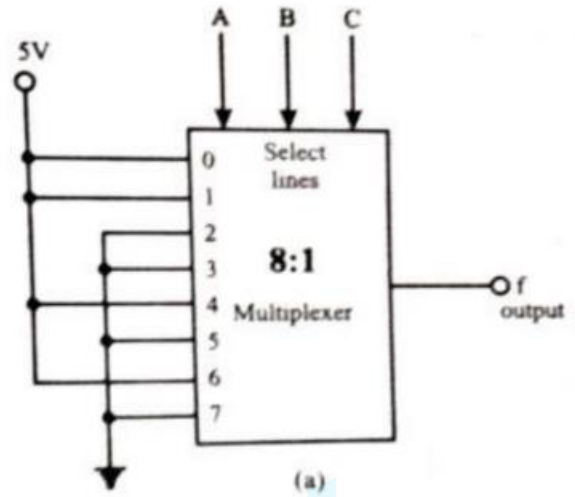
a) 8 : 1 multiplexers

b) 4 : 1 multiplexers

c) 2 : 1 multiplexers

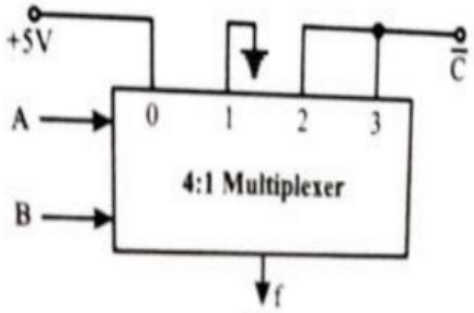
Solution:

8 : 1 multiplexers



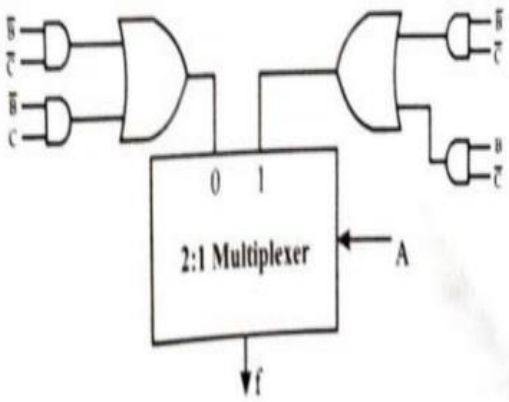
(a)

A	B	C	F		
0	0	0	1	F = 1	D0
0	0	1	1		
0	1	0	0	F = 0	D1
0	1	1	0		
1	0	0	1	F = C'	D2
1	0	1	0		
1	1	0	1	F = C'	D3
1	1	1	0		



(b)

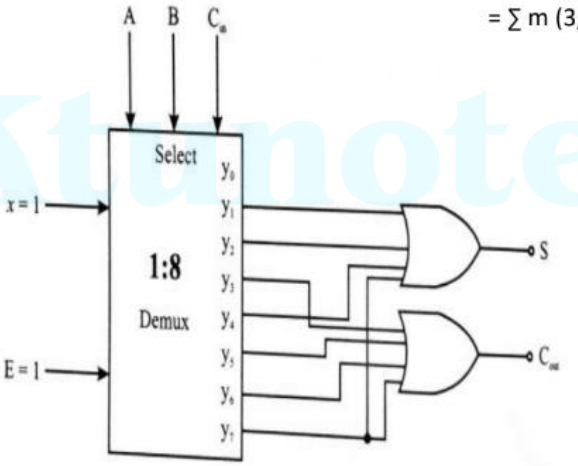
F(A, B, C) = ∑ m (0, 1, 4, 6)
= A'B'C' + A'B'C + AB'C'+ ABC'
= A'(B'C' + B'C) + A (B'C' + BC')



Problem #16
Realize a full adder using a demultiplexer

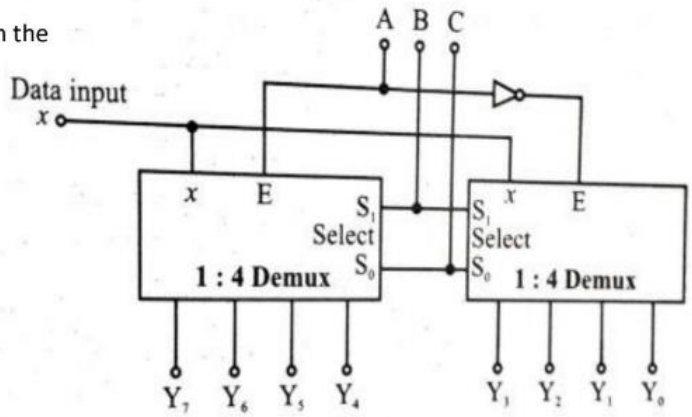
S = A ⊕ B ⊕ C
= AB'C' + A'B'C + A'BC' + ABC
== ∑ m (1, 2, 4, 7)

Cout = AB + ACin + Bcin
= AB(Cin+ C'in) + ACin (B +B') + BCin(A+ A')
= ABCin + ABC'in + ABCin + AB'Cin + ABCin + A'Bcin
= ABCin + ABC'in + AB'Cin + A'Bcin
= ∑ m (3, 5, 6, 7)



Problem #17
Design a 1:8 demultiplexer using two 1:4 demultiplexers

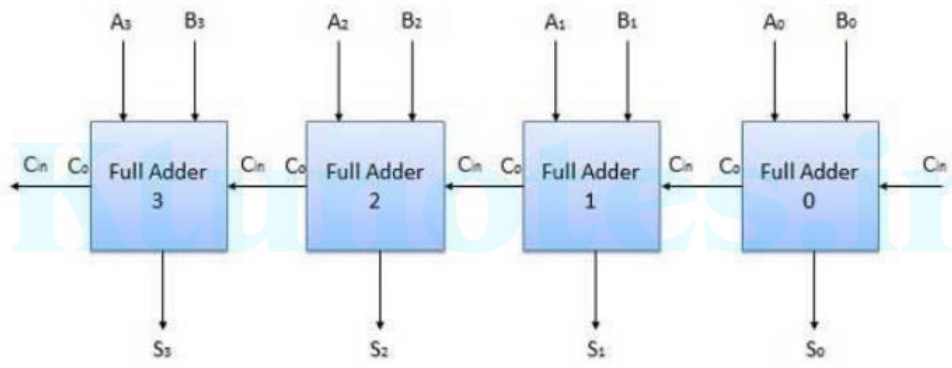
- 1:4 demultiplexer has only 2 select lines
- 1: 8 demultiplexer requires 3 select lines
- Additional select line can be derived from the enable input



N-Bit Parallel Adder

- Full Adder is capable of adding only two single digit binary number along with a carry input.
- Practically we need to add binary numbers which are much longer than just one bit.
- To add two n-bit binary numbers, we need to use the n-bit Parallel Adder.
- Parallel Adder uses a number of Full Adders in cascade.
- Carry output of the previous Full Adder is connected to carry input of the next Full Adder (hence also called ripple carry adder)

4-Bit Parallel Adder



A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder.

- Drawback: ripple carry implementation- increases propagation delay associated with each stage and decreases operating speed.

Carry Look Ahead Adder

- Overcomes the drawback of Parallel adder
- Sum and carry outputs of each stage are made independent of the results of previous stages- ripple effect is eliminated
- Uses concept of look-ahead carry
- Requires additional circuitry but speed becomes independent of number of bits (or stages)

- $C_i = A_i B_i + C_{i-1} (A_i \oplus B_i)$

- Let P_i be carry propogate, $P_i = A_i \oplus B_i$
- Let G_i be carry generate , $G_i = A_i B_i$

- $C_i = G_i + P_i C_{i-1}$
- $C_{i+1} = G_{i+1} + P_{i+1} C_i$
- $C_{i+1} = G_{i+1} + P_{i+1} G_i + P_{i+1} P_i C_{i-1}$

For first full adder

- Carry propagate: $P_0 = A_0 \oplus B_0$
- Carry generate: $G_0 = A_0 B_0$
- Carry out of first full adder: $C_0 = G_0 + P_0 C_{-1}$

For second full adder

$C_1 = G_1 + P_1 C_0$

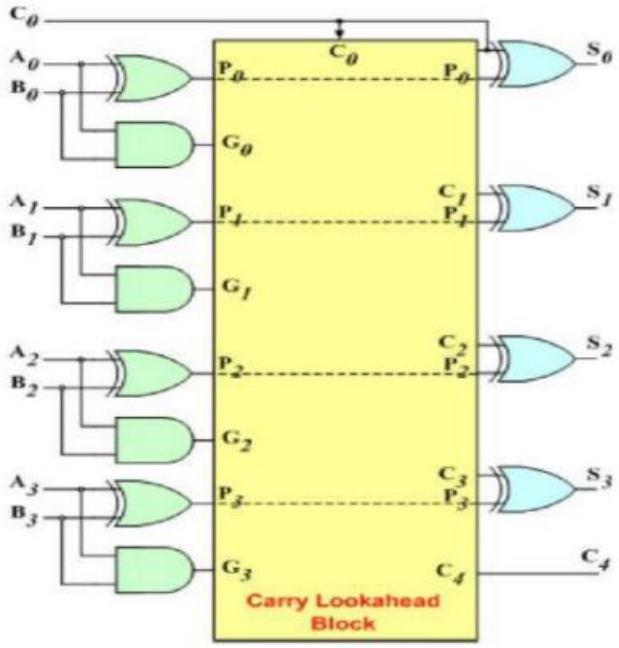
$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{-1}$

Continuing to other stages

$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1}$

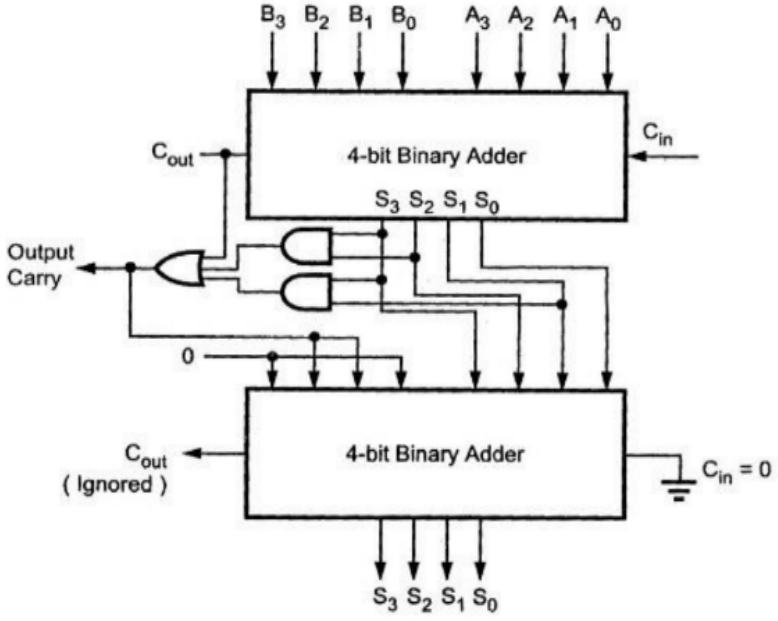
$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}$

4-bit Adder with look-ahead carry(contd)



BCD Adder

BCD addition is carried out by individually adding the corresponding 4-bit groups starting from the LSB side, and if there is any carry to the next group, or if the result belongs to any of the 6 illegal states, that we add 6 (0110) to the sum term of the group and resulting carry is added in the next group.



For partial sum outputs
1010, 1011, 1100, 1101, 1110, 1111,
Correction should be done by adding
6 (0110) to the partial sum $S_3S_2S_1S_0$

0110 is added to the partial sum if any one of the following occurs

1. C_{out} is 1
2. S_3 and S_2 simultaneously high
3. S_3 and S_1 simultaneously high

MODULE 3 : Design of Parity Generators and Parity Checkers

What is Parity?

- In digital systems when binary data is transmitted and processed, it may be subjected to noise and this noise can alter the data bits(1's become 0's or 0's become 1's)
- To detect errors, a parity bit is added to the word, to make the number of 1's odd or even.
- The message containing the data bits along with the parity bit is transmitted from the transmitter node to receiver node
- At the receiver node, the parity bit is checked with the number of 1's in the data and if it does not match, error is detected.
- Two types- Even Parity and Odd Parity

Parity Generator

It is combinational circuit that accepts an n-1 bit data and generates the additional bit that is to be transmitted with the bit stream. This additional or extra bit is called as a Parity Bit.

In even parity bit scheme, the parity bit is '0' if there are even number of 1s in the data stream and the parity bit is '1' if there are odd number of 1s in the data stream.

In odd parity bit scheme, the parity bit is '1' if there are even number of 1s in the data stream and the parity bit is '0' if there are odd number of 1s in the data stream. Let us discuss both even and odd parity generators.

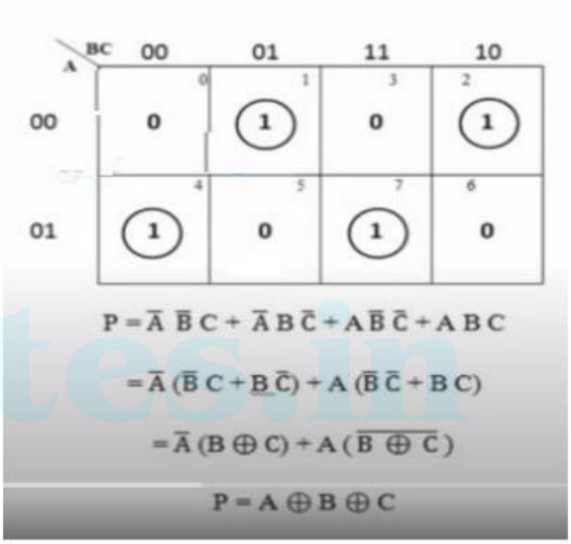
Parity Check

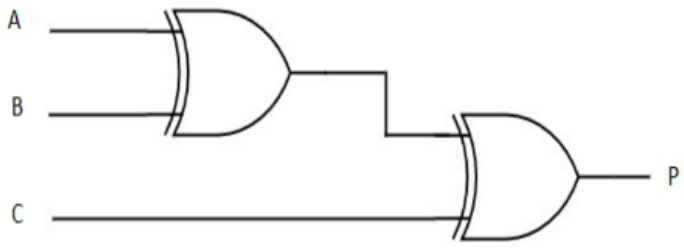
It is a logic circuit that checks for possible errors in the transmission. This circuit can be an even parity checker or odd parity checker depending on the type of parity generated at the transmission end. When this circuit is used as even parity checker, the number of input bits must always be even.

Even Parity Generator

- Let us assume that a 3 bit message is to be transmitted with an even parity bit.
- Let A, B, C be the three inputs, and output Y is the parity bit.
- Output Y is 0 if the number of 1's in (A, B, C) is even.
- Output Y is 1 if the number of 1's in (A, B, C) is odd.

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



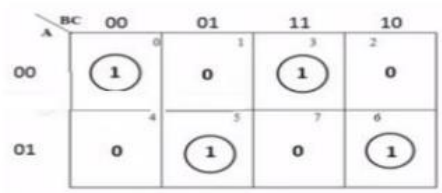


$$P = A \oplus B \oplus C$$

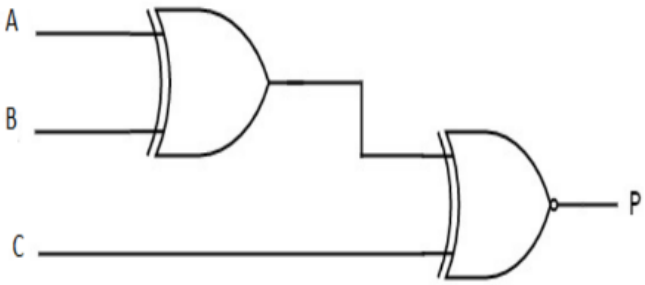
Odd Parity Generator

- Let us assume that a 3 bit message is to be transmitted with an odd parity bit.
- Let A, B, C be the three inputs, and output Y is the parity bit.
- Output Y is 0 if the number of 1's in (A, B, C) is odd.
- Output Y is 1 if the number of 1's in (A, B, C) is even.

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



$$P_{od} = (\bar{A}\bar{B}\bar{C} + A\bar{B}C) + (\bar{A}BC + AB\bar{C})$$
$$= \bar{C}(AB + \bar{A}\bar{B}) + B(A\bar{B} + \bar{A}B)$$
$$= \bar{C}(A \oplus B) + C(A \oplus B)$$
$$= \bar{C}\bar{X} + CX = C \odot X = C \odot A \oplus B$$

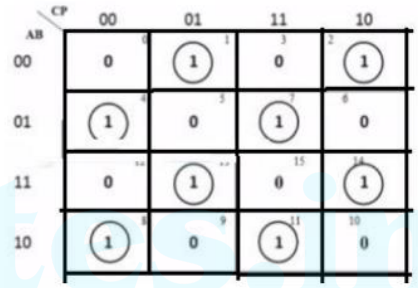


$$P = C \odot A \oplus B$$

Even Parity Checker

- Consider that a three input message along with an even parity bit is applied as input to an even parity checker circuit.
- This circuit checks the possibility of error.
- Parity Error Check (PEC) Output =1, if any error occurs
  i.e. there will be an odd number of 1's for the 4 bits received.
- Parity Error Check (PEC) Output =0, if no error occurs
  i.e. there will be an even number of 1's for the 4 bits received.

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

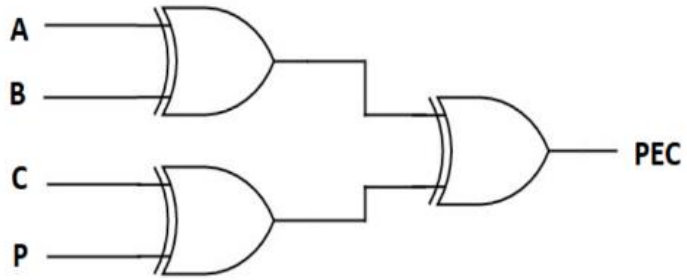


$$PEC = \overline{A}\ \overline{B}\ (\overline{C}P + C\overline{P}) + \overline{A}B(\overline{C}\ \overline{P} + CP) + AB(\overline{C}P + C\overline{P}) + A\overline{B}\ (\overline{C}\ \overline{P} + CP)$$

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

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

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

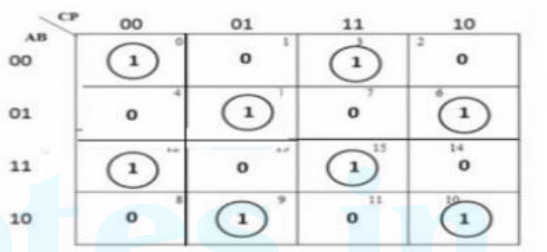


$$PEC = (A \oplus B) \oplus (C \oplus P)$$

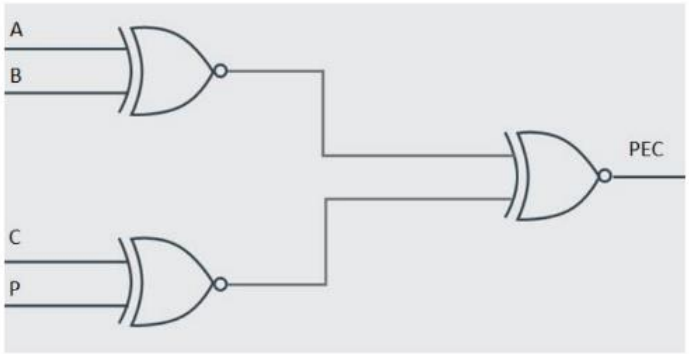
Odd Parity Checker

- Consider that a three input message along with an odd parity bit is applied as input to an odd parity checker circuit.
- This circuit checks the possibility of error.
- Parity Error Check (PEC) Output = 1, if any error occurs
  i.e. there will be an even number of 1's for the 4 bits received.
- Parity Error Check (PEC) Output = 0, if no error occurs
  i.e. there will be an odd number of 1's for the 4 bits received.

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



PEC = (A XNOR B) XNOR (C XNOR P)



PEC = (A XNOR B) XNOR (C XNOR P)

Code Converter

A code converter is a logic circuit that changes data presented in one type of binary code to another type of binary code

Examples-

- Binary to BCD
- Binary to Excess 3
- Binary to Gray Code
- Gray Code to Binary

Steps to design a code converter logic circuit

Step 1: Truth table showing relationship between input and output

Step 2 : For each output, determine the simplified Boolean expression using KMAP

Step 3: Realize the code converter using logic gates

Binary to BCD Code Convertor

Input variables are B3, B2, B1, B1 representing a 4 bit Binary number.

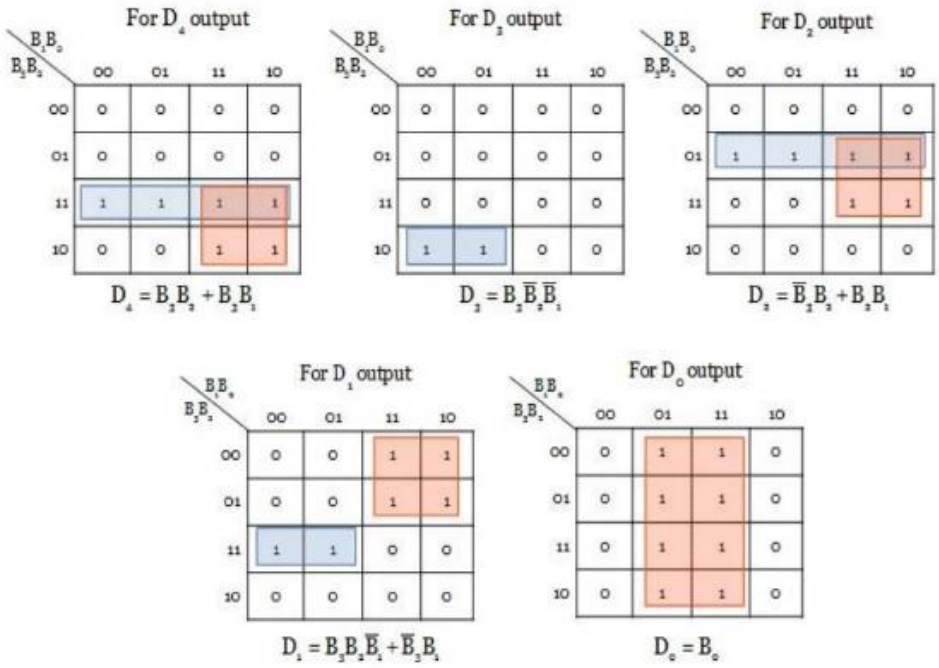
Output variables are D4, D3, D2, D1, D0, representing its equivalent BCD.

Since highest decimal value of 4 bit binary number is 15, 5 bits are required to represent its BCD. D4 for the first digit, and D3D2D1D0 for the second digit

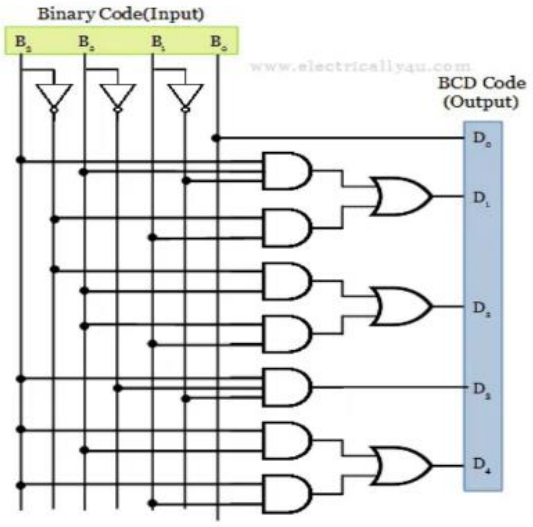
Truth Table

Decimal Number	Binary code (Input)				BCD code (Output)				
	B3	B2	B1	B0	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

K-maps



Logic Circuit Diagram



$$D_4 = B_3B_2 + B_3B_1$$

$$D_3 = B_3\overline{B}_2\overline{B}_1$$

$$D_2 = \overline{B}_3B_2 + B_2B_1$$

$$D_1 = B_3B_2\overline{B}_1 + \overline{B}_3B_1$$

$$D_0 = B_0$$

BCD to Excess 3 Convertor

Valid BCD is only from 0 to 9 (10 to 15 are invalid BCD)

Excess 3 code is obtained by adding 3 (0011) to the BCD code. Since BCD 10 to 15 are invalid, corresponding entries in kmap are marked with don't cares (X)

Truth Table:

Decimal Number	BCD code (Input)				Excess-3 code (Output)			
	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
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

i

K-maps

For E_3 output

D_2D_3 \ 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_2D_3 \ 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_2D_3 \ 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

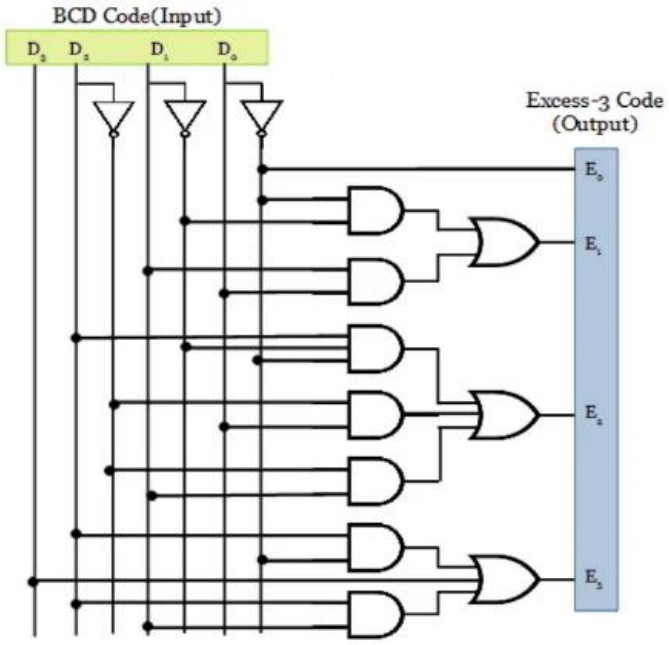
$E_1 = \overline{D}_1\overline{D}_0 + D_1D_0$

For E_0 output

D_2D_3 \ 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

$E_0 = \overline{D}_0$

Logic Circuit Diagram



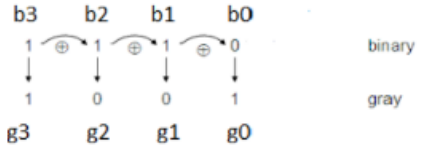
$$E_3 = D_3 + D_2D_0 + D_2D_1$$

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

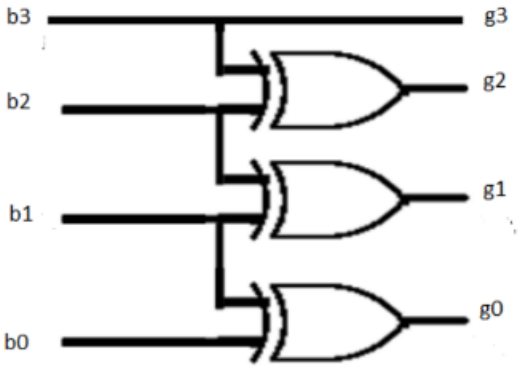
$$E_1 = \overline{D}_1\overline{D}_0 + D_1D_0$$

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

Binary to Gray Code converter



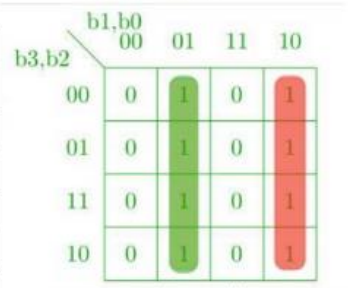
$$g_3 = b_3$$
$$g_2 = b_3 \oplus b_2$$
$$g_1 = b_2 \oplus b_1$$
$$g_0 = b_1 \oplus b_0$$



Binary to Gray Code converter

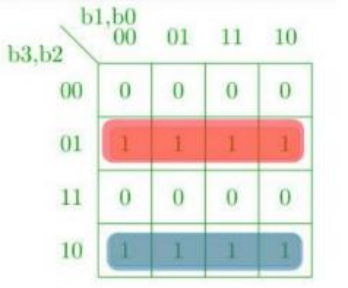
Binary				Gray Code			
b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Kmap for g0



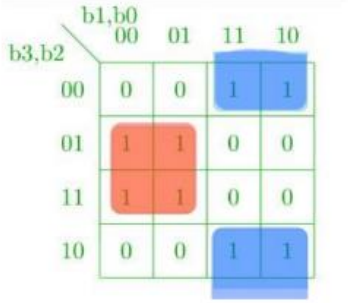
$g_0 = b_0b_1' + b_1b_0' = b_0 \oplus b_1$

Kmap for g2



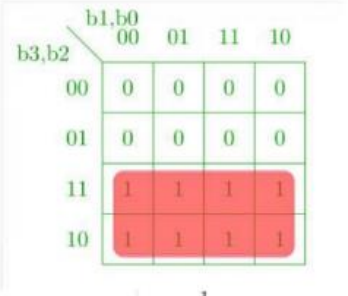
$g_2 = b_2b_3' + b_3b_2' = b_2 \oplus b_3$

Kmap for g1



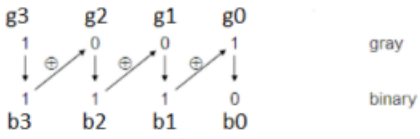
$g_1 = b_2b_1' + b_1b_2' = b_1 \oplus b_2$

Kmap for g3

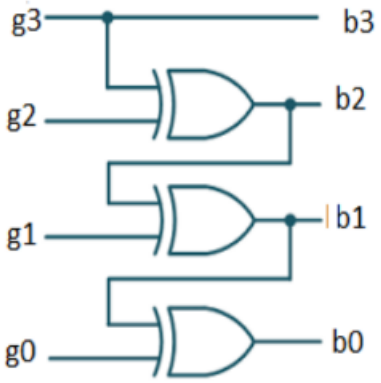


$g_3 = b_3$

Gray Code to Binary converter



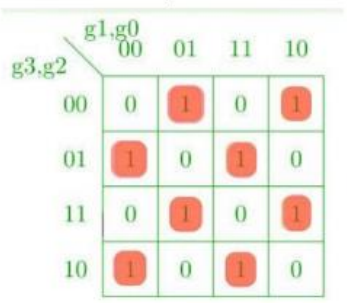
b3 = g3
b2 = b3 ⊕ g2
b1 = b2 ⊕ g1
b0 = b1 ⊕ g0



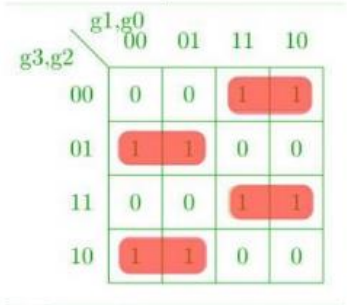
Gray code to Binary Converter

Gray Code				Binary			
g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

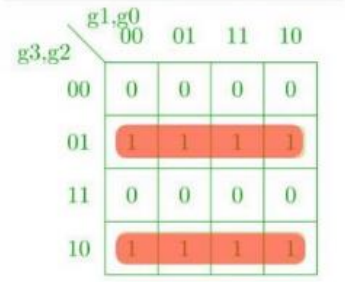
Kmap for b0



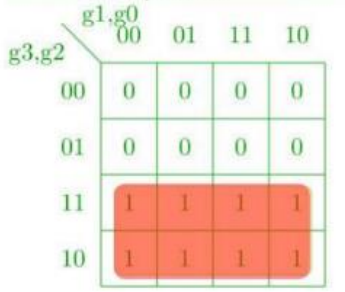
Kmap for b1



Kmap for b2



Kmap for b3

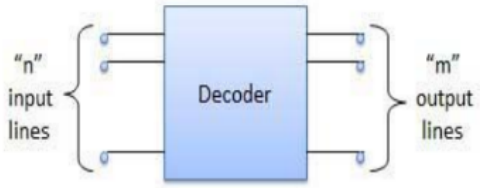


Gray code to Binary Converter (corresponding Boolean expressions)

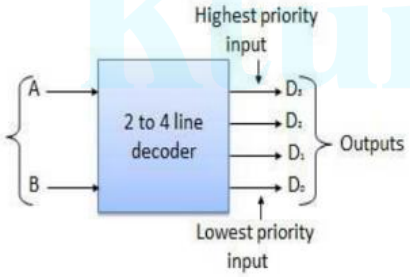
$$
\begin{aligned}
b_0 &= g_3'g_2'g_1'g_0 + g_3'g_2'g_1g_0' + g_3'g_2g_1'g_0' + g_3'g_2g_1g_0 + g_3g_2'g_1'g_0' + g_3g_2'g_1g_0 \\
&\quad + g_3g_2g_1'g_0 + g_3g_2g_1g_0' \\
&= g_3'g_2'(g_1'g_0 + g_1g_0') + g_3'g_2(g_1'g_0' + g_1g_0) + g_3g_2'(g_1'g_0' + g_1g_0) \\
&\quad + g_3g_2(g_1'g_0 + g_1g_0') \\
&= g_3'g_2'(g_0 \oplus g_1) + g_3'g_2(g_0 \odot g_1) + g_3g_2'(g_0 \odot g_1) + g_3g_2(g_0 \oplus g_1) \\
&= (g_0 \oplus g_1)(g_2 \odot g_3) + (g_0 \odot g_1)(g_2 \oplus g_3) \\
&= g_3 \oplus g_2 \oplus g_1 \oplus g_0 \\
b_1 &= g_3'g_2'g_1 + g_3'g_2g_1' + g_3g_2g_1 + g_3g_2'g_1' \\
&= g_3'(g_2'g_1 + g_2g_1') + g_3(g_2g_1 + g_2'g_1') \\
&= g_3'(g_2 \oplus g_1) + g_3(g_2 \odot g_1) \\
&= g_3 \oplus g_2 \oplus g_1 \\
b_2 &= g_3'g_2 + g_3g_2' \\
&= g_3 \oplus g_2 \\
b_3 &= g_3
\end{aligned}
$$

Decoders

- A decoder is a class of multiple-input, multiple-output combinational circuits
- It converts a set of input variables representing a code into a set of output variables representing a different code.
- Most common is n to 2^n decoder.

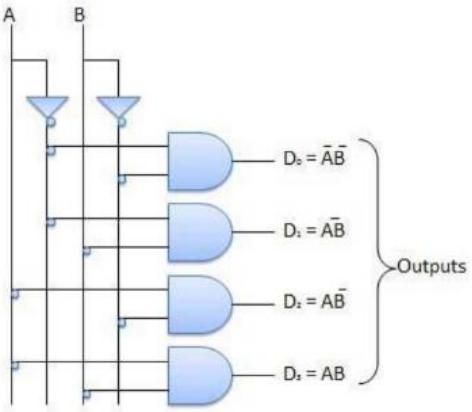


2 to 4 Line Decoder



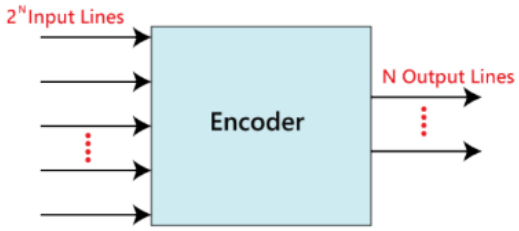
- Simplest form of decoder with 2 input lines and 4 output lines.
- Decoders can be used as minterm and maxterm generators.

Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

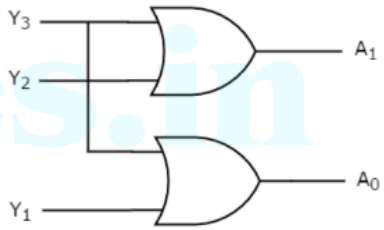


Encoder

- Convert one binary code to another
- Differ from decoders in that the number of input lines will be much more than the number of output lines
- Eg 2^n : n encoder



4 : 2 Line Encoder

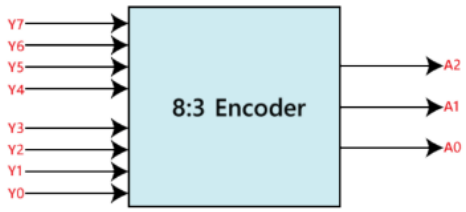


Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

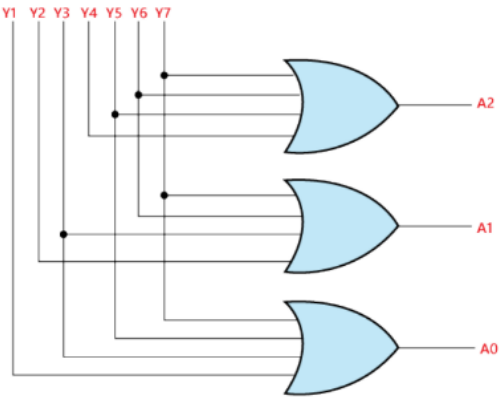
$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

8 : 3 line encoder (Octal to Binary Encoder)



INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_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



Logical expression

$A_2=Y_4+Y_5+Y_6+Y_7$

$A_1=Y_2+Y_3+Y_6+Y_7$

$A_0=Y_7+Y_5+Y_3+Y_1$

- Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, **Digital Encoder** more commonly called a **Binary Encoder** takes ALL its data inputs one at a time and then converts them into a single encoded output.
- A binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output.

Priority Encoder

Drawbacks of Normal Encoders –

- There is an ambiguity, when all inputs of encoder are equal to zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code.

To overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the code corresponding to the active High inputs, which has higher priority. This type of digital encoder is known commonly as a **Priority Encoder** or **P-encoder**

The priority encoder is an encoder with a priority function. If two or more inputs have the same value, then the output will be corresponding to the inputs having the highest priority.

4 : 2 Priority Encoder

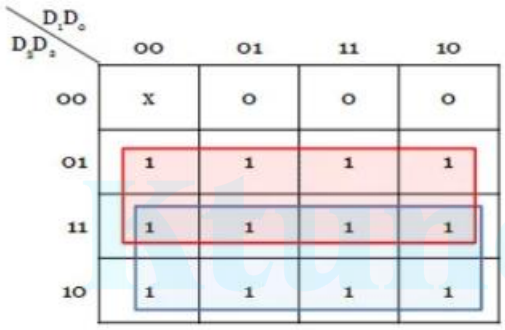
Inputs				Output	
D_3	D_2	D_1	D_0	B	A
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

The four inputs D_3, D_2, D_1, D_0 has a different priority.
D_3 has the highest priority,
whereas D_0 has the lowest priority.

When D_3 input is high, irrespective of other inputs,
the output will be AB = 11.

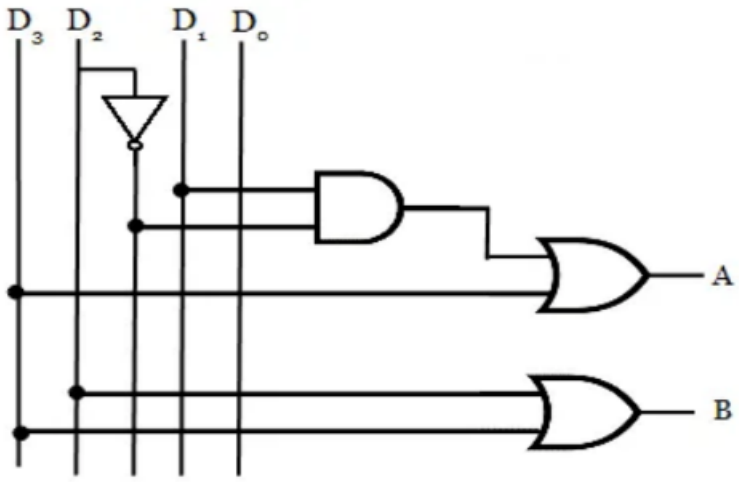
When D_3 input is low, D_2 has the next higher priority.
Irrespective of the other inputs, the output will be BA = 10.

When D_3 and D_2 inputs are low, D_1 takes the next higher priority.
Irrespective of D_0 input, the output will be BA = 01.



$B = D_2 + D_3$

$A = D_3 + D_1\overline{D}_2$



$A = D_3 + D_1\overline{D}_2$

$B = D_2 + D_3$