# BCD Adder, Binary Multiplier, Multiplexers, De-Multiplexers, Decoders, Encoders, Parity Circuits and Comparators

## BCD Adder

### The Rules for BCD addition:

1) When the BCD sum is less than or equal to 1001 without a carry, the corresponding BCD digit is correct. No correction needed.

2) When the BCD sum is greater than 1001, without a carry. the result is invalid. The addition of binary $6(0110)_2$ to the sum converts it to the correct digit.

3) When the BCD sum is less than or equal to 1001, but with a carry '1', the result is again invalid. The addition of binary $6\ (0110)_2$ to the sum converts it to the correct digit.

example: Perform BCD addition of 448 & 489.

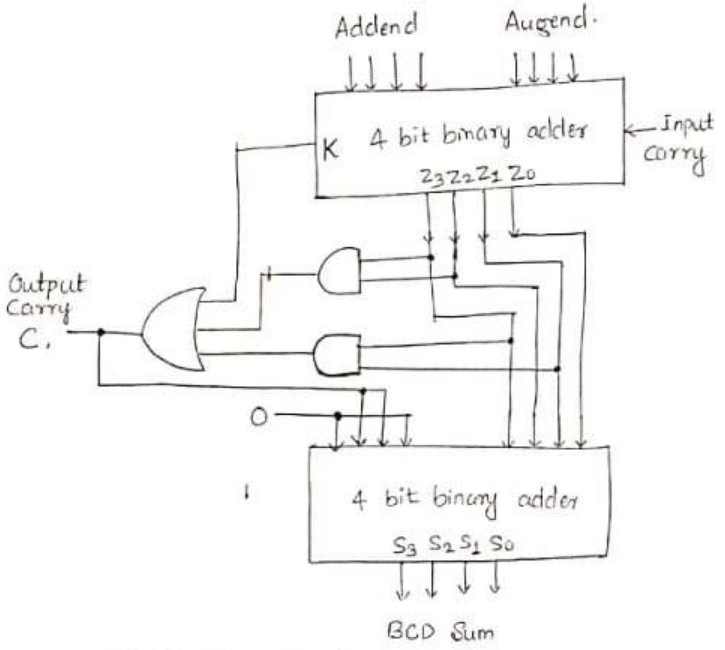
Sol$^n$:

```
   11
   448
 + 489
 -----
   937

           1            1
     1     ↓            ↓
 +  0100 | 0100     |  1000
    0100 | 1000     |  1001
 ----------------------------
    1001 | 1101     (1) 0001
         | 0110         0110
 ----------------------------
    1001 (1) 0011       0111
    Rule 1   Rule 2     Rule 3.
```

P.T.O

A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown below.



Block Diagram of BCD Adder

It has two 4-bit binary adders and correction logic. The two decimal digits, together with an input carry, are added in the first 4-bit binary adder, to produce the binary sum.

The condition for correction can be expressed by the boolean function $\boxed{C = K + Z_3 Z_1 + Z_3 Z_2}$

Here 'C' is the output carry from the BCD adder, & 'K' is the output carry from the first binary adder. The two terms with the 'Z' variables detect the binary outputs from 1010 through 1111.

P.T.O

When the BCD carry is equal to '0', '0000' is added to the binary sum. This condition occurs if the sum of two BCD numbers and carry is less than or equal to 1001

When the BCD carry is equal to '1', '0110' is added to the binary sum through the second 4-bit binary adder. This condition occurs when the sum of BCD is greater than 1001 without carry and also for when BCD sum is less or equal to 1001 without carry.

## BINARY MULTIPLIERS.

Multiplication of binary numbers is performed in the same way as with decimal numbers.

The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.

Each such multiplication forms a partial product.

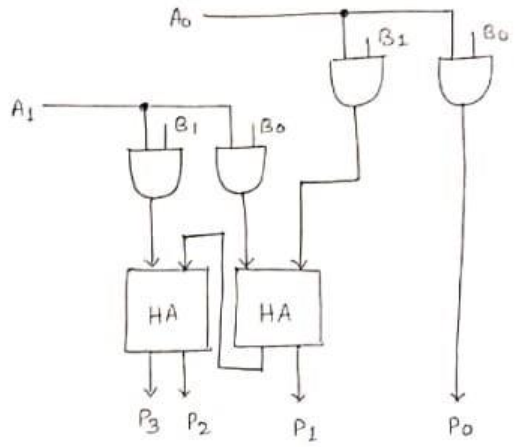
Successive partial products are shifted one bit to the left.

The final product is obtained from the sum of the partial products.

To see how a binary multiplier can be implemented with a combinational circuit, consider the multiplication of two 2-bit numbers, as shown below.

The multiplicand bits are $B_1$ & $B_0$, the multiplier bits are $A_1$ & $A_0$, and the product is $P_3P_2P_1P_0$.

| | | | |
|---|---|---|---|
| | | $B_1$ | $B_0$ |
| | | $A_1$ | $A_0$ |
| | | $A_0B_1$ | $A_0B_0$ |
| | $A_1B_1$ | $A_1B_0$ | + |
| $P_3$ | $P_2$ | $P_1$ | $P_0$ |



A 2x2 Binary Multiplier

The first partial product is formed by multiplying $B_1B_0$ by $A_0$.

The multiplication of two bits such as $A_0$ and $B_0$ produces '1' if both bits are '1', otherwise it produces a '0'. This is identical to an AND operation. Therefore, the partial product can be implemented with AND gates as shown above.

The second partial product is formed by multiplying $B_1B_0$ by $A_1$. and is shifted one position to the left. The two partial products are added with two half adder (HA) circuits.

Note that the least significant of the product goes not have to go through an adder, since it is formed by the o/p of 1st AND gate.

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.

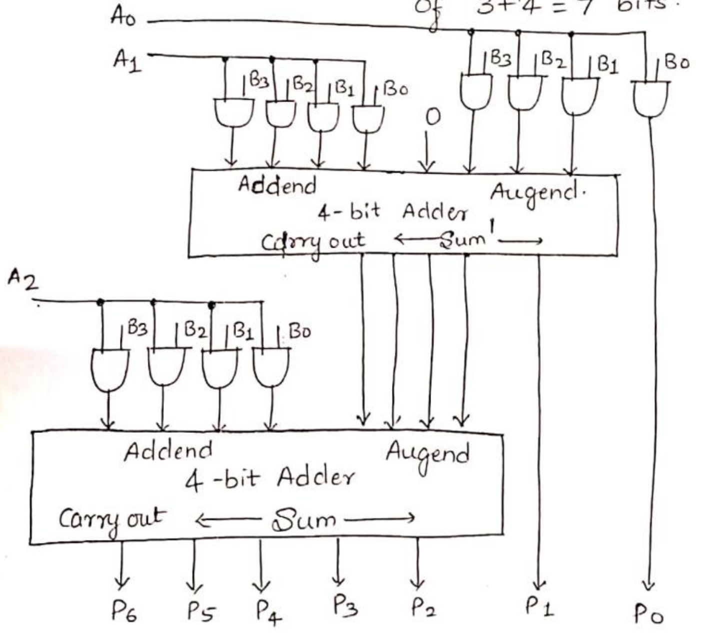
A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in multiplier.

The binary o/p in each level of AND gates is added in parallel with the partial product of the previous level to form a new partial product. The last level produces the product.

For M multiplier bits and N multiplicand bits,
we need M×N AND gates
(M-1), N bit adders to produce a product of
M+N bits.

for example: Let Multiplicand is $B_3 B_2 B_1 B_0$ &
Multiplier is $A_2 A_1 A_0$

Since $M=3$ & $N=4$, we need $3 \times 4 = 12$ AND gates
(3-1) 4 bit Adders
= 2, 4-bit Adders to
produce a product
of $3+4 = 7$ bits.



A 4 X 3 bit Binary Multiplier

## MULTIPLEXERS (MUX)

Multi Inputs ; Single Output

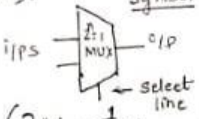
A multiplexer is a combinational circuit that selects binary information from one of many input lines and sends the information to a single output line.

The selection of a particular input line is controlled by a set of input variables, called Selection inputs.

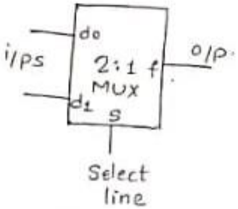
Normally, there are '$2^n$' input lines an 'n' selection inputs whose bit combinations determine which input is Selected. ( $2^n : 1$ is the MUX configuration).

↑ inputs ↑ output.

Symbol



A simple 2:1 MUX is as shown below ($2:1 = 2^1:1$ ∴ n = 1 select line)



Truth Table

| S | do | d1 | f |
|---|---|---|---|
| 0 | 1 | 0 | do |
| 1 | 0 | 1 | d1 |

OR

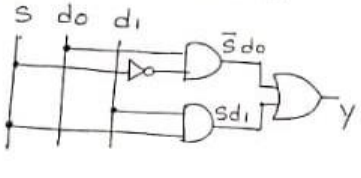
| S | Y |
|---|---|
| 0 | do |
| 1 | d1 |

Note: In MUX any one input bit will be high at an instant of time.

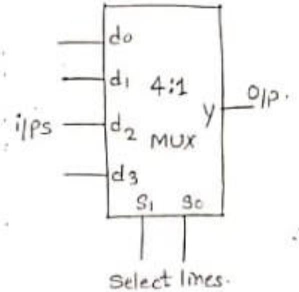
From the truth table, it is seen that
if S=0, do input is Selected and sent to the o/p.
S=1, d1 input is Selected and sent to the o/p

$\therefore \boxed{Y = \bar{S}d_0 + Sd_1}$

Realization of 2:1 MUX using gates →



## 4:1 MUX ($2^2:1$, n=2)



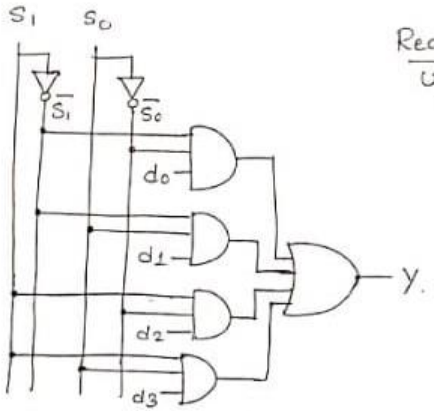
Truth Table

| S1 | S0 | Y |
|---|---|---|
| 0 | 0 | do |
| 0 | 1 | d1 |
| 1 | 0 | d2 |
| 1 | 1 | d3 |

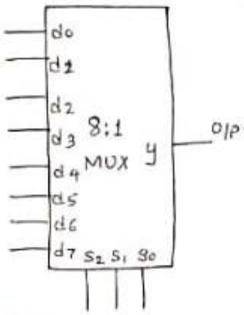
$\therefore \boxed{Y = \bar{S}_1\bar{S}_0 d_0 + \bar{S}_1 S_0 d_1 + S_1\bar{S}_0 d_2 + S_1 S_0 d_3}$

Realization of 4:1 MUX Using gates

4:1 MUX Circuit diagram



## 8:1 MUX ($2^3:1$, n=3).

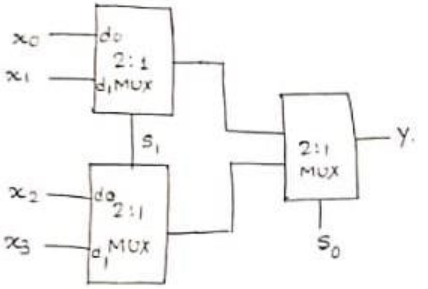


| S2 | S1 | S0 | Y |
|---|---|---|---|
| 0 | 0 | 0 | do |
| 0 | 0 | 1 | d1 |
| 0 | 1 | 0 | d2 |
| 0 | 1 | 1 | d3 |
| 1 | 0 | 0 | d4 |
| 1 | 0 | 1 | d5 |
| 1 | 1 | 0 | d6 |
| 1 | 1 | 1 | d7 |

$$\therefore \boxed{\begin{aligned} Y = {} & \bar{S}_2\bar{S}_1\bar{S}_0 d_0 \\ & + \bar{S}_2\bar{S}_1 S_0 d_1 \\ & + \bar{S}_2 S_1\bar{S}_0 d_2 \\ & + \bar{S}_2 S_1 S_0 d_3 \\ & + S_2\bar{S}_1\bar{S}_0 d_4 \\ & + S_2\bar{S}_1 S_0 d_5 \\ & + S_2 S_1\bar{S}_0 d_6 \\ & + S_2 S_1 S_0 d_7 \end{aligned}}$$

Exercise : Configure 4:1 MUX using 2:1 MUX.

Soln

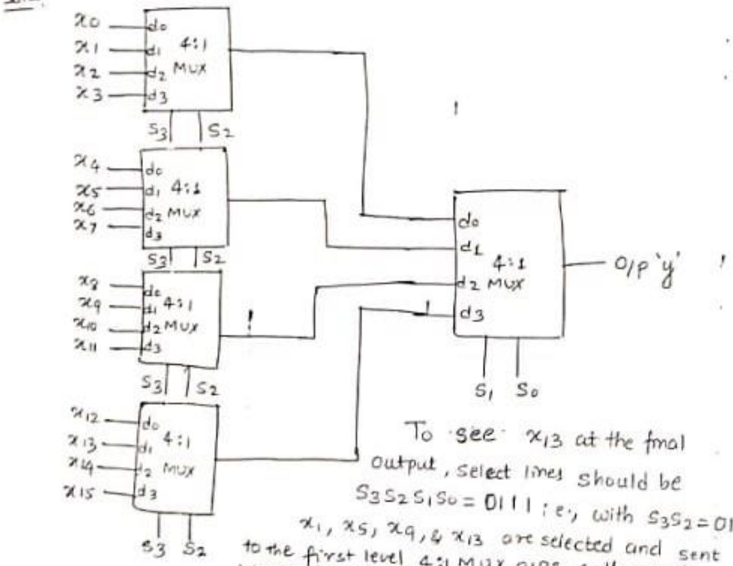


Let us say address or select lines $S_1S_0 = 10$ then for $S_1 = 1$ places $x_1$ at $d_1$ & $x_3$ at $d_1$

for $S_0 = 0$ places $x_1$ at the final output

Exercise : Configure 16:1 MUX using 4:1 MUX.

Soln.



To see $x_{13}$ at the final output, select lines should be $S_3S_2S_1S_0 = 0111$ i.e., with $S_3S_2 = 01$ $x_1, x_5, x_9, \& \, x_{13}$ are selected and sent to the first level 4:1 MUX o/ps & these acts like i/ps to second level 4:1MUX, $S_1S_0 = 11$, $x_{13}$ is selected at Y.

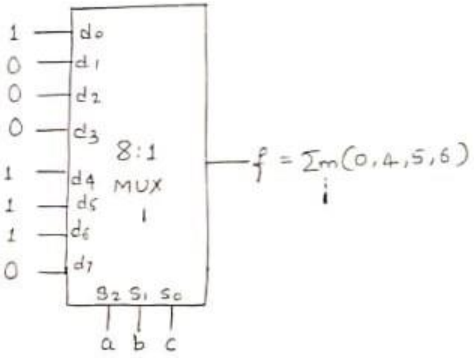
Exercise: Implement the following function using 8:1 MUX

$$f(a\,b\,c) = \Sigma m(0, 4, 5, 6)$$

Soln,

Let $d_0 = d_4 = d_5 = d_6 = 1$ &

$d_1 = d_2 = d_3 = d_7 = 0$ in a 8:1 MUX



Exercise : Implement the following function using 4:1 MUX

$$f(a\,b\,c) = \Sigma m(0, 1, 2, 7)$$

Soln. The given function is 3 variable function i.e., we get 8 combination of i/p variables. So it can be directly implemented using 8:1 MUX as we did in previous example. But asked is to implement using 4:1 MUX. So we have to minimize the given function.

The algebraic form of given function is

$$f(a\,b\,c) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc$$

$$= \bar{a}\bar{b}(\bar{c} + c) + \bar{a}b\bar{c} + abc$$

$$f(a\,b\,c) = \bar{a}\bar{b}(1) + \bar{a}b\bar{c} + abc$$

P.T.O

$f(abc) = \bar{a}\bar{b}(1) + \bar{a}b\bar{c} + bbc$

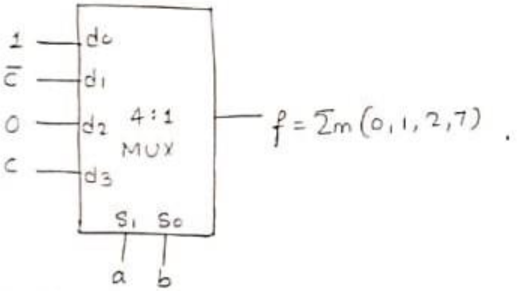
Lets compare this eqn with 4:1 MUX general equation

$$f = \bar{S}_1\bar{S}_0(d_0) + \bar{S}_1 S_0(d_1) + S_1\bar{S}_0(d_2) + S_1 S_0(d_3)$$

∴ Here a & b will be acting as select lines & 'c' will be the data lines

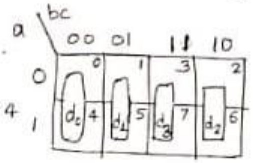
∴ $d_0 = 1$, $d_1 = \bar{c}$, $d_2 = 0$, $d_3 = c$

↑ as we dont have $a\bar{b}$ combination in the simplified equation for $d_2$



Exercise: Implement $f(abc) = \Sigma m(1,4,5,7)$ using 4:1 MUX using 'b' and 'c' as select lines

Soln. $f(abc) = \Sigma m(1,4,5,7)$.



b=0, c=0 corresponds to cells 0 & 4

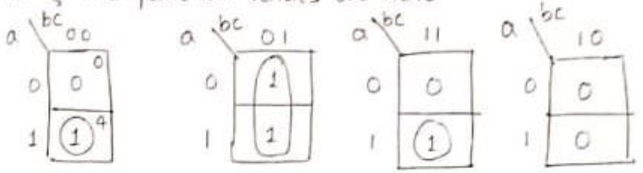
b=0, c=1 corresponds to cells 1 & 5

b=1, c=0 corresponds to cells 2 & 6

b=1, c=1 corresponds to cells 3 & 7

P.T.O

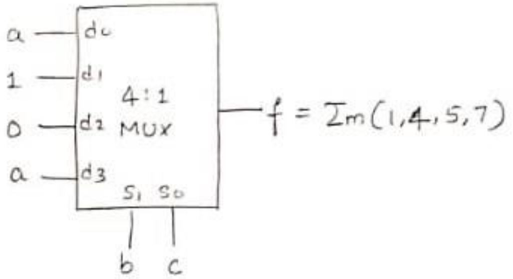
Filling the function values we have



d0-map d1-map d3-map d2-map

∴ $\boxed{d_0 = a}$ $d_1 = \bar{a} + a$ $\boxed{d_1 = 1}$ $\boxed{d_3 = a}$ $\boxed{d_2 = 0}$

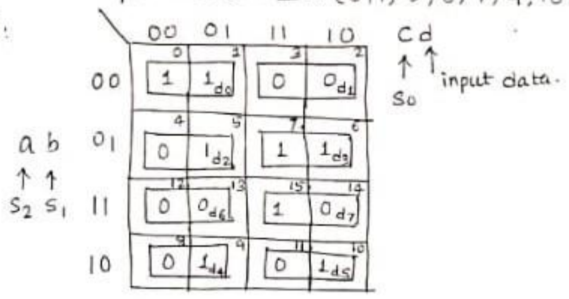
Here 'a' is acting as data line & (b,c) acting as Select lines. ∴ $d_0 d_1 d_2 d_3$ takes values only interms of 'a'.



Example: Implement the following function using 8:1 MUX. Treat a, b & c as select lines.

$f(abcd) = \Sigma m(0,1,5,6,7,9,10,15)$.

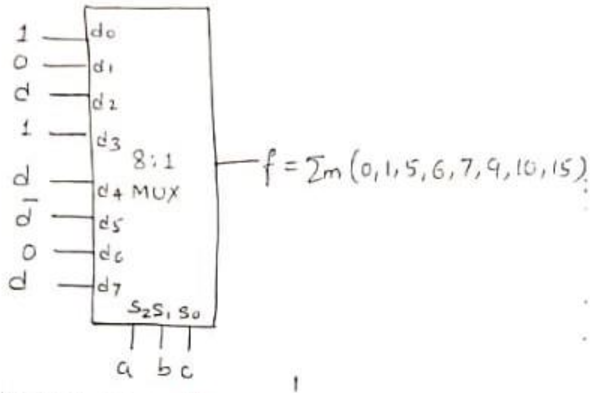
Soln.



P.T.O.

From the map.

$d_0 = 1$, $d_1 = 0$, $d_2 = d$, $d_3 = 1$, $d_4 = d$, $d_5 = \bar{d}$, $d_6 = 0$, $d_7 = d$



Exercise: Implement $f(a\,b\,c\,d) = \Sigma m(0,1,5,6,7,9,10,15)$ using 4:1 MUX with a & b as select lines.

Soln.

| ab ↑ ($S_1$ $S_0$) \ cd | 00 | 01 | 11 | 10 | |
|---|---|---|---|---|---|
| 00 | $1^{0}$ | $1^{1}$ | $0^{3}$ | $0^{2}$ | → $d_0$ |
| 01 | $0^{4}$ | $1^{5}$ | $1^{7}$ | $1^{6}$ | → $d_1$ |
| 11 | $0^{12}$ | $0^{13}$ | $1^{15}$ | $0^{14}$ | → $d_3$ |
| 10 | $0^{8}$ | $1^{9}$ | $0^{11}$ | $1^{10}$ | $d_2$. |

| ab \ cd | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 0 | 0 |

$d_0$ map ∴ $\boxed{d_0 = \bar{c}}$

| ab \ cd | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 01 | 0 | 1 | 1 | 1 |

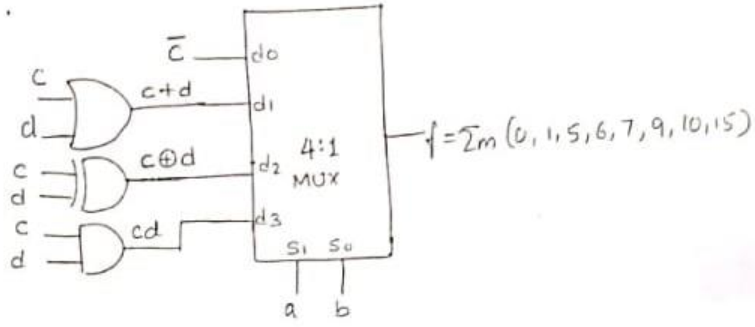
$d_1$ map ∴ $\boxed{d_1 = c + d}$

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 11 | 0 | 0 | 1 | 0 |

$d_3$ map ∴ $\boxed{d_3 = cd}$

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 10 | 0 | 1 | 0 | 1 |

$d_2$ map ∴ $\boxed{d_2 = \bar{c}d + c\bar{d} = c \oplus d}$



Exercise: Implement the boolean function $f(a\,b\,c\,d) = \Sigma m(0,2,4,5,7,9,10,14)$ using Multiplexers with two 4:1 MUX with variables (a, d) connected to their select lines in the first level. And one 2:1 MUX with variable 'c' connected to its select line in the second level.

Soln. $f(a\,b\,c\,d) = \Sigma m(0,2,4,5,7,9,10,14)$

| ab \ cd | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $1^{0}$ | $0^{1}$ | $0^{3}$ | $1^{2}$ |
| 01 | $1^{4}$ | $1^{5}$ | $1^{7}$ | $0^{6}$ |
| 11 | $0^{12}$ | $0^{13}$ | $0^{15}$ | $1^{14}$ |
| 10 | $0^{8}$ | $1^{9}$ | $0^{11}$ | $1^{10}$ |

|← c=0 →|← c=1 →|

Such problems can be solved in simpler way. Divide the whole K-map with single variable acting as select line in the problem definition.

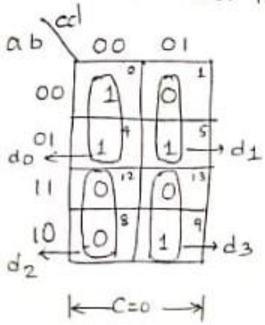
From problem definition, variable 'c' is acting as a select line of 2:1 MUX in second level.

C = 0 & C = 1 if its a select line for 2:1 MUX

∴ Look into the K-map whereever we have C=0 & C=1

So when we divide K-map, we get as shown above.

Let us consider the C=0 map for final 2:1 MUX



(a=0, d=0) represents d0

(a=0, d=1) represents d1

(a=1, d=0) represents d2

(a=1, d=1) represents d3.

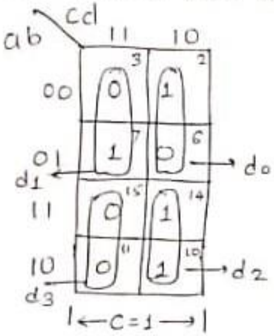
Here a & d are select lines of 4:1 MUX in first level.

Let us write $d_0, d_1, d_2, d_3$ in terms of 'b'

$$\boxed{d_0 = 1,\quad d_1 = b,\quad d_2 = 0,\quad d_3 = \overline{b}}$$

These will be the i/ps of first 4:1 MUX in first level.

Now Let us look at C=1 map.

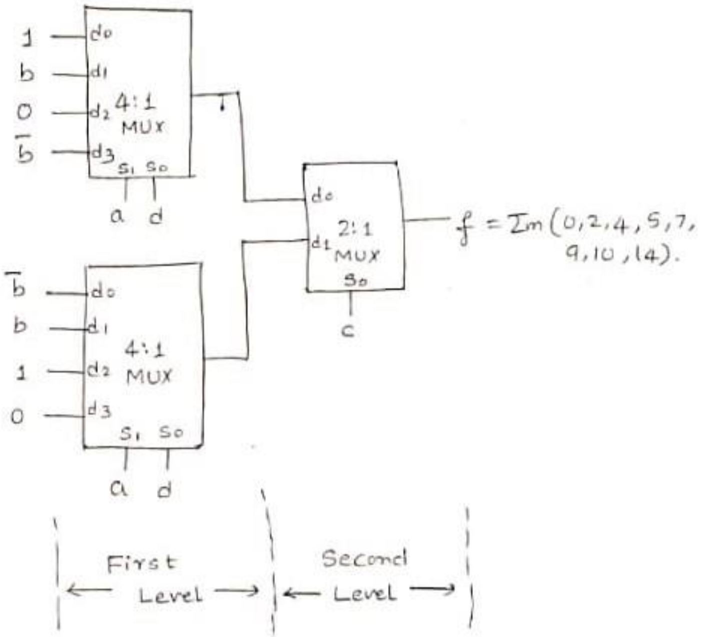


$$\therefore \boxed{\begin{aligned} d_0 &= \overline{b} \\ d_1 &= b \\ d_2 &= 1 \\ d_3 &= 0 \end{aligned}}$$

These will be the i/ps of second 4:1 MUX in first level.

P.T.O

∴ The implementation is shown below.



## DEMULTIPLEXERS (DeMUX)

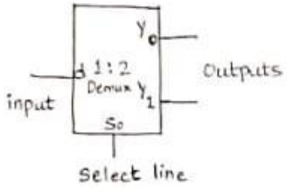
A demultiplexer is a digital function that performs the inverse of the multiplexer operation. i.e., a demultiplexer receives information from a single line and transmits it to one of $2^n$ possible output lines.

The selection of the specific output is controlled by the bit combination of 'n' selection lines.

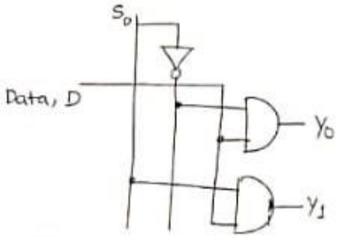
$\boxed{\text{Single Input : Multi Outputs}}$

P.T.O

## 1:2 DeMUX $(1:2^1, n=1)$



| $S_0$ | $Y_0$ | $Y_1$ |
|---|---|---|
| 0 | D | 0 |
| 1 | 0 | D |



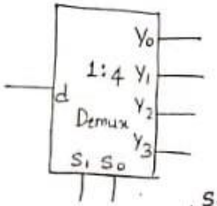
Realization of 1:2 Demux using gates

NOTE: In DeMUX any one O/P bit will be high at an instant of time.

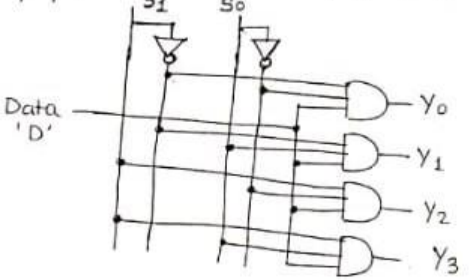
From truth table,

if S=0, D is sent to $Y_0$
S=1, D is sent to $Y_1$

## 1:4 DeMUX $(1:2^2, n=2)$

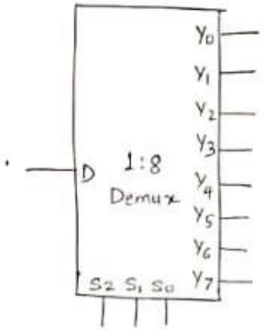


| $S_1$ | $S_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|
| 0 | 0 | D | 0 | 0 | 0 |
| 0 | 1 | 0 | D | 0 | 0 |
| 1 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 0 | 0 | 0 | D |

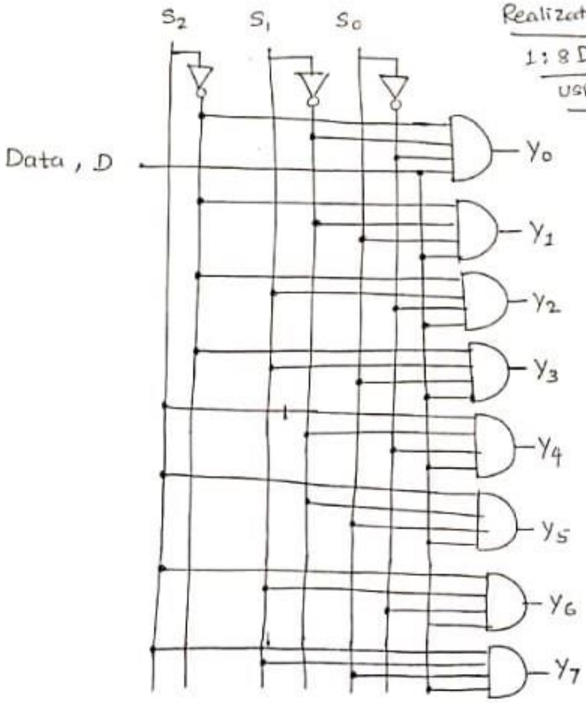


Realization of 1:4 DeMUX using gates.

## 1:8 DeMUX $(1:2^3, n=3)$



| $S_2$ | $S_1$ | $S_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ | $Y_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | D | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | D | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | D | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | D | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | D | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | D | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D |

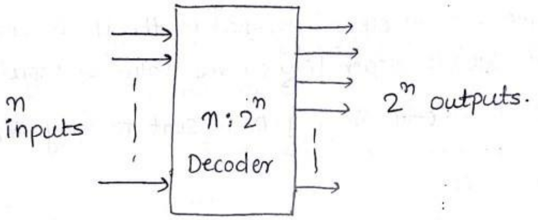


Realization of 1:8 DeMUX using gates

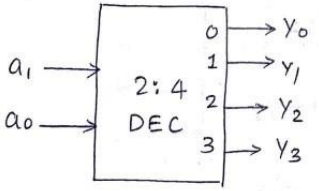
# DECODERS

A decoder is a combinational circuit that converts binary information from the 'n' coded i/ps to a maximum of '$2^n$' unique o/ps. i.e., n inputs = $2^n$ output
$\therefore (n : 2^n)$.

Multi Input = Multi Output

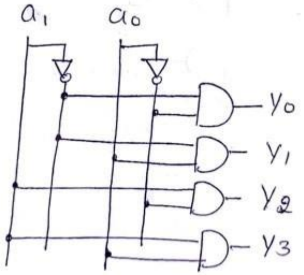


## 2:4 Decoder $(2 : 2^2, n=2)$



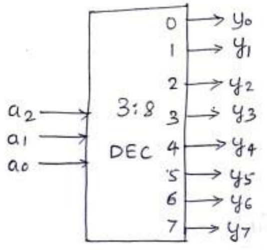
| Inputs | | Outputs. | | | |
|---|---|---|---|---|---|
| $a_1$ | $a_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

$\therefore Y_0 = \bar{a}_1\bar{a}_0$ ; $Y_1 = \bar{a}_1 a_0$ ; $y_2 = a_1\bar{a}_0$ & $y_3 = a_1 a_0$



Note: Any one o/p will be high at an instant of time

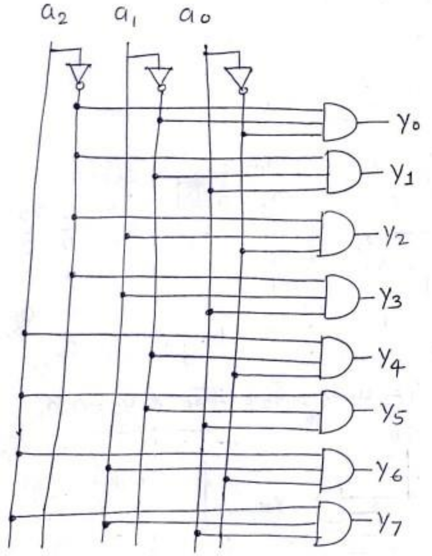
## 3:8 Decoder ($3:2^3$, $n=3$)



| $a_2$ | $a_1$ | $a_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ | $Y_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

We can observe here in decoders, that is the i/p is sent to the particular output depending on the value of input .

for eg, 000 is to only $Y_0$ ; 110 is sent to only $Y_6$



## Logic Design Using Decoders.

We observe that each o/p of the decoder generates a minterm (i.e; product of i/ps combinations)

2:4 DEC generates 4 minterms using two input variables.

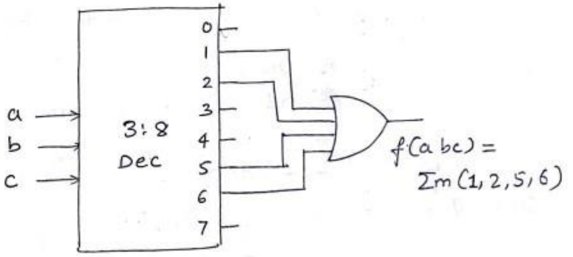
3:8 DEC generates 8 minterms using three input variables.

Thus, $n:2^n$ DEC generates '$2^n$' minterms using 'n' i/p variables & finally realized a SOP expression.

Exercise: Implement $f(abc) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}c + ab\bar{c}$ using 3:8 Decoder.

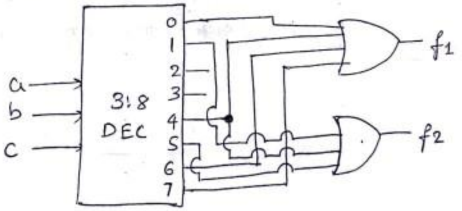
Soln: $f(abc) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}c + ab\bar{c}$

$\therefore f(abc) = \sum m(1,2,5,6)$.



Exercise: Implement $f_1(abc) = \sum m(0,4,6,7)$ & $f_2(abc) = \sum m(1,4,5)$ using 3:8 Decoder

Soln



Observe that the no.of inputs to each OR-gate would be the no.of minterms in the expression.

We can reduce the no.of i/ps by taking complement of given function (i.e, $\bar{f}$) instead of f and finally invert the OR o/p by f. This can simply be done by replacing the ORgate with a NOR gate.

This can be adopted when the no.of minterms in a function is more than half the total no.of possible minterms.

Exercise: Implement the following functions using a decoder minimizing the no.of i/ps to be summed.

$f_1(abc) = \Sigma m(0,2,3,5,6,7)$.

$f_2(abc) = \Sigma m(1,3,4,6,7)$.

Soln $f_1(abc) = \Sigma m(0,2,3,5,6,7)$

Here no. of minterms are more than half of the total minterms

$\therefore \bar{f_1}(abc) = \Sigma m(1,4)$ in SOP form.

OR $f_1(abc) = \Pi M(1,4)$ in POS form.

Similarly for $f_2(abc) = \Sigma m(1,3,4,6,7)$

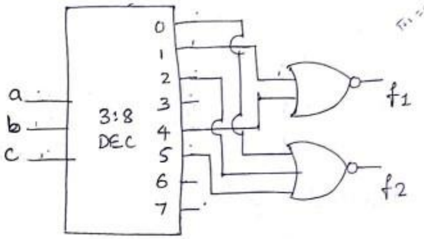
$\bar{f_2}(abc) = \Sigma m(0,2,5)$ SOP form.

or $f_2(abc) = \Pi M(0,2,5)$ POS form.

We can use any one of the two forms to implement.

P.T.O

The implementation is shown below.



Exercise: Implement the following functions expressed in maxterm canonical form using 3to 8 line decoder minimizing the no.of i/ps.

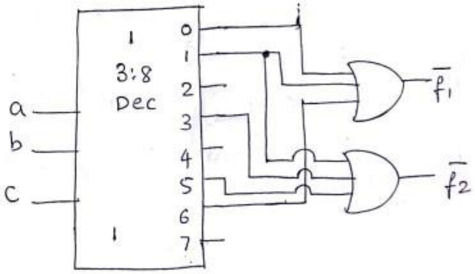
$f_1(abc) = \Pi M(2,3,4,5,7)$

$f_2(abc) = \Pi M(0,2,4,6,7)$.

Soln: By looking into the expressions, both have its terms more than half of the possible total term.

∴ we take complement of these functions.

$\therefore f_1 = \Sigma m(0,1,6)$ & $f_2 = \Sigma m(1,3,5)$.



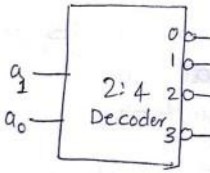
Note: for SOP, our goal is to get '1'
for POS, our goal is to get '0'

In the first example, SOP is converted to POS ∴ our goal is to get '0', & to get '0' we have to use NOR gate

In the second example, POS is converted to SOP ∴ our goal is to get '1', to get '1' we have to use OR gate

∴ In general, we have to concentrate only on the complemented function, if its POS use NOR gate
if its SOP use OR gate.

Decoders in IC packages usually have an active low output. The truth table and symbol of such 2:4 decoder is shown below.



$\overline{\bar{a}\,\bar{b}} = a+b = M_0$

$\overline{\bar{a}\,b} = a+\bar{b} = M_1$

$\overline{a\,\bar{b}} = \bar{a}+b = M_2$

$\overline{a\,b} = \bar{a}+\bar{b} = M_3$

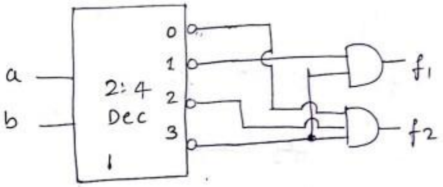
∴ These generate maxterms at their outputs and are useful in implementing POS expressions.

| Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| $a_1$ | $a_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

Exercise: Implement the following functions in maxterm canonical form using 2:4 decoder with active low outputs.

$f_1(a,b) = \Pi M(1,3)$
$f_2(a,b) = \Pi M(0,2,3)$.

Sol<sup>n</sup>.

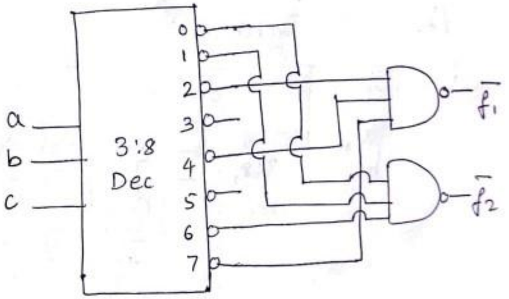


Exercise: Implement the following functions using 3:8 decoder with active low outputs.

$f_1(a\,b\,c) = \Pi M(0,1,3,5,6)$
$f_2(a\,b\,c) = \Pi M(2,3,4,5,7)$

Sol<sup>n</sup>

$\overline{f}_1(a\,b\,c) = \Sigma m(2,4,7)$
$\overline{f}_2(a\,b\,c) = \Sigma m(0,1,6)$



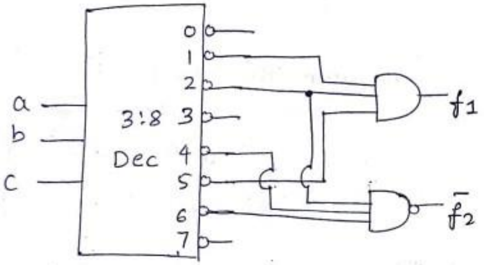
Exercise: $f_1 = \Pi M(1,2,5)$, $f_2 = \Pi M(0,1,3,5,7)$.
Implement using 3:8 Dec. with active low O/ps.

Sol^n: $f_1$ is having its terms less than the half of its total possible terms. ∴ No need of complementing;

$f_2$ is having more than half of its total terms
∴ It has to be complemented.

$$\therefore\ f_1(abc) = \Pi M(1,2,5)$$

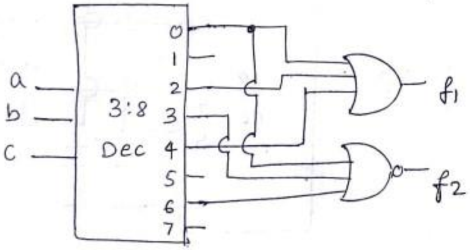
$$\overline{f_2}(abc) = \Sigma m(2,4,6).$$



Excercise: $f_1(abc) = \Sigma m(0,2,4)$; $f_2(abc) = \Sigma m(1,2,4,5,7)$
Implement with uncomplemented O/ps (active high O/ps).

Sol^n: $f_1(abc) = \Sigma m(0,2,4)$

$\overline{f_2}(abc) = \Pi M(0,3,6)$.



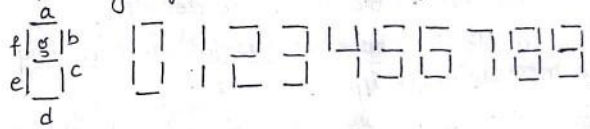
Note:
In active high output decoder
Use OR gate to implement SOP form
Use NOR gate to implement POS form

In active low output decoder
Use NAND gate to implement SOP form
Use AND gate to implement POS form

## BCD to Seven Segment Decoder

- Digital readouts found in electronic calculators and digital watches use LEDs. Each digit of the readout is formed from seven segments, each consisting of one LED that can be illuminated by digital signals.

A BCD-to-Seven segment decoder is a combinational circuit that accepts a digits in BCD and generates the appropriate O/ps for the selection of segments that display the digit. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display. as shown below.



| Cell No. | W | X | Y | Z | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | $0_1$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 10,11,12,13,14,15 | All other i/ps. | | | | x | x | x | x | x | x | x |

When we map all the seven segment variables into K-map individually, we get expression as.

$$a = w+y+xz+\bar{x}\bar{z}$$
$$b = \bar{x}+\bar{y}\bar{z}+yz$$
$$c = x+\bar{y}+z$$
$$d = w+y\bar{z}+\bar{x}\bar{z}+\bar{x}y+x\bar{y}z$$
$$e = y\bar{z}+\bar{x}\bar{z}$$
$$f = w+\bar{y}\bar{z}+x\bar{y}+x\bar{z}$$
$$g = w+x\bar{y}+\bar{x}y+y\bar{z}$$

## Decoders with Enable Input

Observe that in the decoders dealt with so, far, one of the o/ps is always '1' in the uncomplemented o/p version (active high o/p version) or one of the o/p is always '0' in the complemented o/p version (active low o/p version). There are several applications where we would want all the o/ps to be '0' or '1' respectively. This can be acheived by including an enable i/p to the decoder.

The enable i/ps are also useful in cascading decoders to increase their i/p-o/p lines.

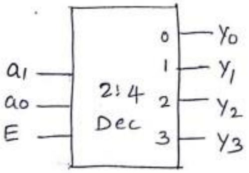
For eg: In cascading 2:4 decoders to obtain 4:16 line decoder.

→ Whenever enable pin is high (1), Decoder is said to be activated and it is enabled to perform operation.

→ Whenever enable pin is low (0), Decoder is said to be de-activated and it is not-enabled to perform operation.

## 2:4 Decoder with Enable

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| E | $a_1$ | $a_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
| 0 | X | X | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |



fig(a): 2:4 line Active High o/p (Un complemented) Decoder.

| Inputs | | | Outputs. | | | |
|---|---|---|---|---|---|---|
| E | $a_1$ | $a_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
| 1 | X | X | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |

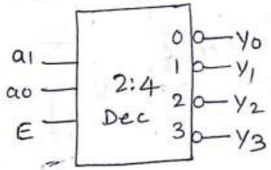
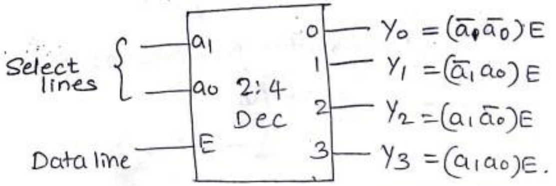


fig (b): 2:4 Active Low o/p (Complemented) Decoder.

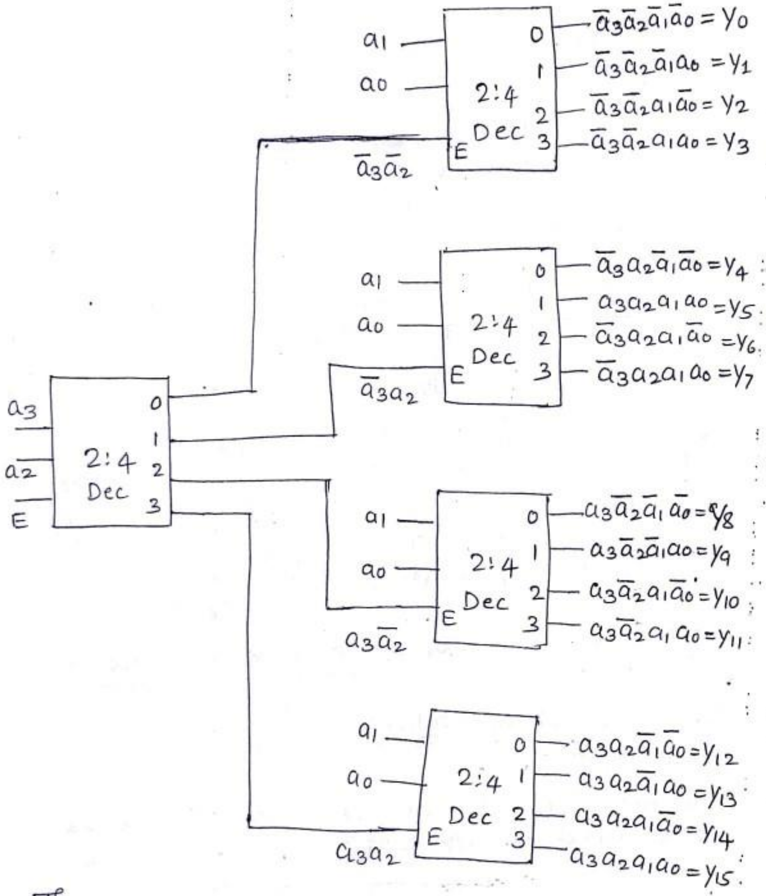
One important application of a decoder is its use as a demultiplexer.

The i/ps of decoder acts as select lines of demux & enable of decoder acts as i/p line of demux.



Decoder as a Demultiplexer.

## 4:16 Decoder using 2:4 Decoders.


$a_1$
$a_0$
2:4 Dec
E
0
1
2
3
$\bar a_3\bar a_2\bar a_1\bar a_0 = y_0$
$\bar a_3\bar a_2\bar a_1 a_0 = y_1$
$\bar a_3\bar a_2 a_1\bar a_0 = y_2$
$\bar a_3\bar a_2 a_1 a_0 = y_3$
$\bar a_3\bar a_2$
$\bar a_3 a_2\bar a_1\bar a_0 = y_4$
$a_3a_2a_1a_0 = y_5$
$\bar a_3 a_2 a_1\bar a_0 = y_6$
$\bar a_3 a_2 a_1 a_0 = y_7$
$\bar a_3 a_2$
$a_3\bar a_2\bar a_1\bar a_0 = y_8$
$a_3\bar a_2\bar a_1 a_0 = y_9$
$a_3\bar a_2 a_1\bar a_0 = y_{10}$
$a_3\bar a_2 a_1 a_0 = y_{11}$
$a_3\bar a_2$
$a_3a_2\bar a_1\bar a_0 = y_{12}$
$a_3a_2\bar a_1 a_0 = y_{13}$
$a_3a_2a_1\bar a_0 = y_{14}$
$a_3a_2a_1a_0 = y_{15}$
$a_3a_2$
$a_3$
$a_2$
E

This 4:16 Decoder generates 16 minterms from $\bar a_3\bar a_2\bar a_1\bar a_0$ to $a_3a_2a_1a_0$.

## ENCODERS

An encoder is a digital function that performs the inverse operation of a decoder.

An encoder has $2^n$ input lines an 'n' output lines.

$2^n$ inputs $= n$ outputs.

$\therefore\ (2^n : n)$

Multi inputs = Multi Outputs.

NOTE: Any one i/p will be high at an instant of time

eg: 4:2 Encoder, 8:3 Encoder

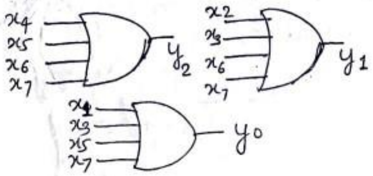
### 8:3 Encoder ( $2^3 : 3$ , $n = 3$ )

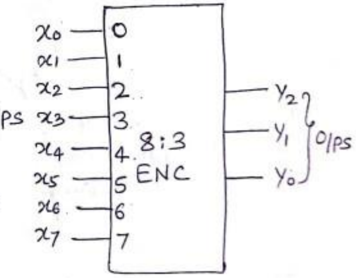
| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$\therefore Y_2 = x_4 + x_5 + x_6 + x_7$

$Y_1 = x_2 + x_3 + x_6 + x_7$

$y_0 = x_1 + x_3 + x_5 + x_7$





There are some problems associated with encoders.

It is possible that when more than one inputs are at logic 1 (high) there may be an error in the output code

For eg: if $x_3 = x_4 = 1$, then $y_2 y_1 y_0 = 111$, whereas this o/p should have resulted for only $x_7 = 1$.

The other problem is that the o/p is 000 when all the i/ps are at logic 0 (low) as well as when $x_0 = 1$.

These problems can be overcome by the priority encoder with a validity indicator. The truth table of a 8:3 priority encoder is shown below.

| Row No. | Inputs |  |  |  |  |  |  |  | Outputs |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $Y_2$ | $Y_1$ | $Y_0$ | Valid. |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | x | x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | x | x | x | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 5 | x | x | x | x | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | x | x | x | x | x | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 7 | x | x | x | x | x | x | 1 | 0 | 1 | 1 | 0 | 1 |
| 8 | x | x | x | x | x | x | x | 1 | 1 | 1 | 1 | 1 |

Observe in the table that higher order bits have higher priority. For eg: if $x_3$ is at logic 1, then irrespective of the logic values of $x_0, x_1$ & $x_2$ the o/p code will be 011. The valid o/p being '1' indicates that the i/p code is valid.

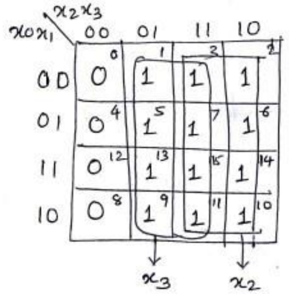
The row 0 & row 1 o/ps are now distinguished by the logic level of the 'Valid' output.

Excercise: Write the condensed truth table for 4:2 priority encoder with a valid o/p where the highest priority is given to the highest bit position or input with highest index and obtain the minimal sum expressions for the o/ps.

Soln:

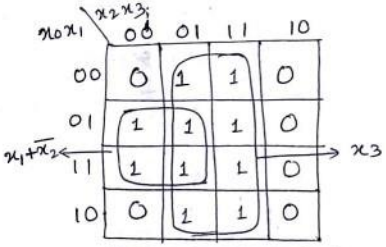
| Cell No. | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $Y_1$ | $Y_0$ | Valid. |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4,12 | x | 1 | 0 | 0 | 0 | 1 | 1 |
| 2, 6, 10, 14 | x | x | 1 | 0 | 1 | 0 | 1 |
| 1, 3, 5, 7, 9, 11, 13, 15 | X | X | X | 1 | 1 | 1 | 1 |

Let us now construct the K-maps for the o/ps.



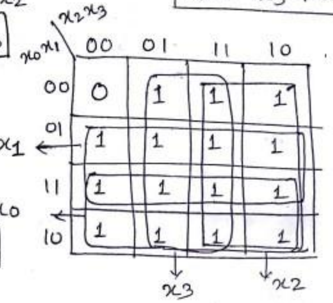
| $x_0x_1 \backslash x_2x_3$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $0^{0}$ | $1^{1}$ | $1^{3}$ | $1^{2}$ |
| 01 | $0^{4}$ | $1^{5}$ | $1^{7}$ | $1^{6}$ |
| 11 | $0^{12}$ | $1^{13}$ | $1^{15}$ | $1^{14}$ |
| 10 | $0^{8}$ | $1^{9}$ | $1^{11}$ | $1^{10}$ |

$\therefore \boxed{y_1 = x_2 + x_3}$



| $x_0x_1 \backslash x_2x_3$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 1 | 1 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |

$\therefore \boxed{Y_0 = x_3 + x_1 \overline{x_2}}$



| $x_0x_1 \backslash x_2x_3$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 1 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

$\therefore$ Valid $= x_0 + x_1 + x_2 + x_3$

Exercise : 4:2 priority encoder with highest priority to the lowest significant i/p or lower index.

Soln :

| Cell No. | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $Y_1$ | $Y_0$ | Valid. |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15, 8, 9, 10, 11, 12, 13, 14 | 1 | X | X | X | 0 | 0 | 1 |
| 4, 5, 6; 7 | 0 | 1 | X | X | 0 | 1 | 1 |
| 2, 3 | 0 | 0 | 1 | X | 1 | 0 | 1 |
| 1 | 0 | 0. | 0 | 1 | 1 | 1 | 1 |

After constructing K-maps for outputs, we get minimal sum expressions as

$$\boxed{\begin{array}{l} Y_1 = \overline{x_0}\,\overline{x_1}\,x_2 + \overline{x_0}\,\overline{x_1}\,x_3 \\ Y_0 = \overline{x_0}\,x_1 + \overline{x_0}\,\overline{x_2}\,x_3 \\ \text{Valid} = x_0 + x_1 + x_2 + x_3 \end{array}}$$

## PARITY CIRCUITS (Generator & Checker)

A parity bit is an extra bit included with a binary message to make the no. of 1's either odd or even.

The message, including the parity bit, is transmitted and then checked at the receiving end for errors.

An error is detected if the parity of the data bits in the message does not correspond to the parity bit transmitted.

The circuit that generates the parity bit in the transmitter is called a PARITY GENERATOR

The circuit that checks the parity bit in the receiver is called a PARITY CHECKER

### Even Parity Generator

Consider a 3-bit message to be transmitted with an even parity bit.

| Row No. | 3-bit Message |  |  | Parity Bit |
|---|---|---|---|---|
|  | X | Y | Z | P |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |

P.T.O

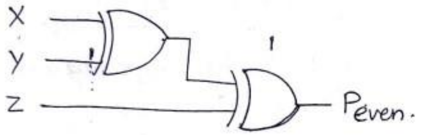
From truth table, 3 bits X, Y & Z constitute the message and are the inputs to the circuit.

The parity bit, P is the output.

For even parity, the bit 'P' must be generated to make the total number of 1's (including P) even.

| X \ YZ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $0^{0}$ | $1^{1}$ | $0^{3}$ | $1^{2}$ |
| 1 | $1^{4}$ | $0^{5}$ | $1^{7}$ | $0^{6}$ |

$$\therefore \boxed{P_{even} = X \oplus Y \oplus Z}$$



## Odd Parity Generator

3-bit message to be transmitted with an odd parity bit

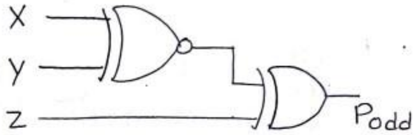
| Row No. | 3-bit Message X | Y | Z | Parity Bit P. |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 0 |

P.T.O

From truth table, for odd parity, the bit 'P' must be generated to make the total no. of 1's (including P) odd.

| X \ YZ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $1^{0}$ | $0^{1}$ | $1^{3}$ | $0^{2}$ |
| 1 | $0^{4}$ | $1^{5}$ | $0^{7}$ | $1^{6}$ |

$$\therefore \boxed{P_{odd} = (X \odot Y) \oplus Z}$$



The 3-bits in the message, together with the parity bit, are transmitted to their destination, where they are applied to a parity checker circuit to check for possible errors in the transmission.

The information transmitted with even parity (odd parity), the 4 bits received must have an even (odd) no. of 1's.

An error occurs if these bits have an odd (even) no. of 1's, indicating that at least one bit has changed its value during transmission.

If such error occurs, the o/p of the parity checker will be equal to 1.

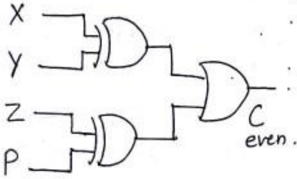
P.T.O

## Even Parity Checker

| Row No. | 4-bit Information (3 bit + 1 bit P)<br>X Y Z P | C |
|---|---|---|
| 0 | 0 0 0 0 | 0 |
| 1 | 0 0 0 1 | 1 |
| 2 | 0 0 1 0 | 1 |
| 3 | 0 0 1 1 | 0 |
| 4 | 0 1 0 0 | 1 |
| 5 | 0 1 0 1 | 0 |
| 6 | 0 1 1 0 | 0 |
| 7 | 0 1 1 1 | 1 |
| 8 | 1 0 0 0 | 1 |
| 9 | 1 0 0 1 | 0 |
| 10 | 1 0 1 0 | 0 |
| 11 | 1 0 1 1 | 1 |
| 12 | 1 1 0 0 | 0 |
| 13 | 1 1 0 1 | 1 |
| 14 | 1 1 1 0 | 1 |
| 15 | 1 1 1 1 | 0 |

| XY \ ZP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $0^{0}$ | $1^{1}$ | $0^{3}$ | $1^{2}$ |
| 01 | $1^{4}$ | $0^{5}$ | $1^{7}$ | $0^{6}$ |
| 11 | $0^{12}$ | $1^{13}$ | $0^{15}$ | $1^{14}$ |
| 10 | $1^{8}$ | $0^{9}$ | $1^{11}$ | $0^{10}$ |

$$\therefore \boxed{C_{even} = X \oplus Y \oplus Z \oplus P}$$



Here, if C = 0 → no errors, if no. of 1's in the stream (X Y Z P) are even.

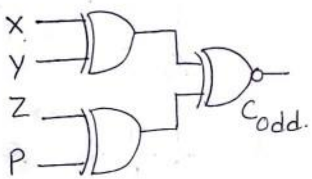
& C = 1 → error, if no. of 1's in the stream (X Y Z, P) are odd.

## Odd Parity Checker

| Row No. | 4 bit Information (3 bit + 1 bit P)<br>X Y Z P | C |
|---|---|---|
| 0 | 0 0 0 b | 1 |
| 1 | 0 0 0 1 | 0 |
| 2 | 0 0 1 0 | 0 |
| 3 | 0 0 1 1 | 1 |
| 4 | 0 1 0 0 | 0 |
| 5 | 0 1 0 1 | 1 |
| 6 | 0 1 1 0 | 1 |
| 7 | 0 1 1 1 | 0 |
| 8 | 1 0 0 0 | 0 |
| 9 | 1 0 0 1 | 1 |
| 10 | 1 0 1 0 | 1 |
| 11 | 1 0 1 1 | 0 |
| 12 | 1 1 0 0 | 1 |
| 13 | 1 1 0 1 | 0 |
| 14 | 1 1 1 0 | 0 |
| 15 | 1 1 1 1 | 1 |

| XY \ ZP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $1^{0}$ | $0^{1}$ | $1^{3}$ | $0^{2}$ |
| 01 | $0^{4}$ | $1^{5}$ | $0^{7}$ | $1^{6}$ |
| 11 | $1^{12}$ | $0^{13}$ | $1^{15}$ | $0^{14}$ |
| 10 | $0^{8}$ | $1^{9}$ | $0^{11}$ | $1^{10}$ |

$$\therefore \boxed{C_{odd} = (X \oplus Y) \odot (Z \oplus P)}$$

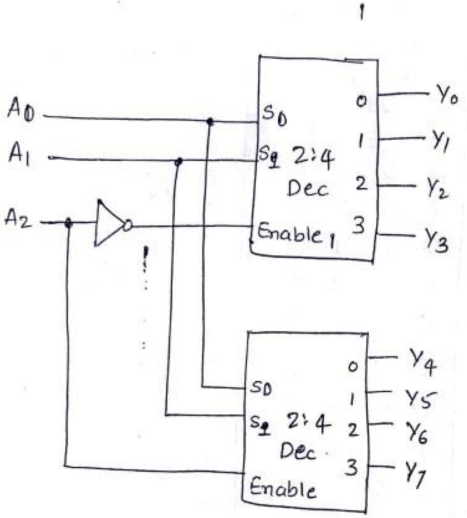


Here if C = 0 → no errors, if no. of 1's in the stream (XYZP) are odd.

& C = 1 → error, if no. of 1's in the stream (XYZP) are even.

Exercise: Construct 3:8 Decoder using 2:4 decoders.

Sol^n.

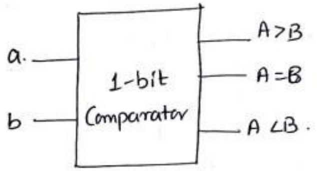


| $A_2$ | $A_1$ | $A_0$ | $Y_7$ | $Y_6$ | $Y_5$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 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 |

## COMPARATORS

Comparators are designed to compare the magnitude of two binary number and indicate whether one is greater than, less than or equal to the other.

- One Bit (1-Bit) Comparator



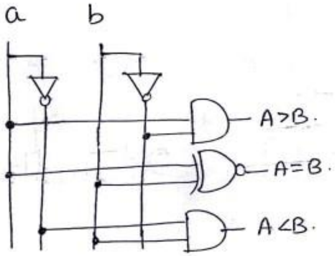
$(A>B)=1$ if $a>b$

$(A=B)=1$ if $a=b$

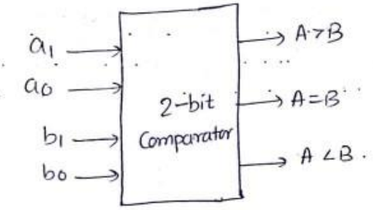
$(A<B)=1$ if $a<b$

| a | b | A>B | A=B | A<B |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

$$\therefore \boxed{(A>B) = a\bar{b} \; ; \; (A=B) = \bar{a}\bar{b} + ab = a \odot b \; ; \; (A<B) = \bar{a}b}$$



## 2-bit Comparator

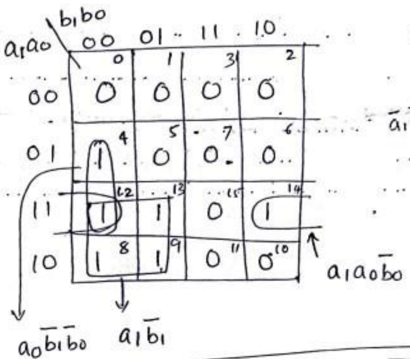


$(A>B)=1$ if $a_1a_0 > b_1 b_0$

$(A=B)=1$ if $a_1a_0 = b_1 b_0$

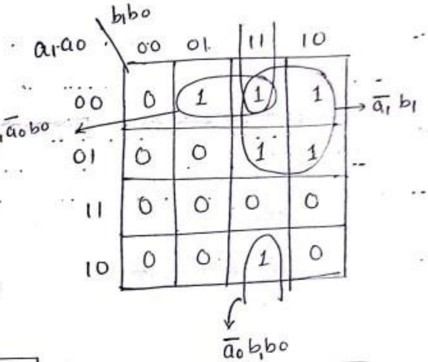
$(A<B)=1$ if $a_1a_0 < b_1b_0$

| Cell no. | $a_1$ | $a_0$ | $b_1$ | $b_0$ | A>B | A=B | A<B. |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 14 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |



| $a_1a_0$ \ $b_1b_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | 1 | 1 | 0 | 1 |
| 10 | 1 | 1 | 0 | 0 |

$\therefore \boxed{(A>B) = a_0\bar{b}_1\bar{b}_0 + a_1\bar{b}_1 + a_1a_0\bar{b}_0}$



| $a_1a_0$ \ $b_1b_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 1 |
| 01 | 0 | 0 | 1 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 1 | 0 |

$\therefore \boxed{(A<B) = \bar{a}_1\bar{a}_0b_0 + \bar{a}_0b_1b_0 + \bar{a}_1b_1}$

$(A>B) = \Sigma m(4, 8, 9, 12, 13, 14)$

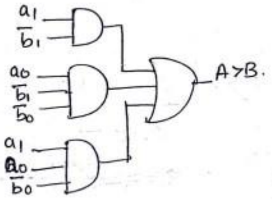
$(A<B) = \Sigma m(1, 2, 3, 6, 7, 11)$

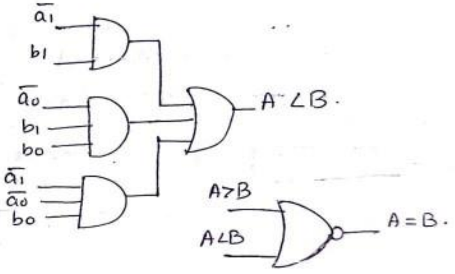
$(A=B) = \Sigma m(0, 5, 10, 15)$

We observe that $(A=B)$ minterms are the uncovered minterms of $(A>B)$ & $(A<B)$.

$\therefore (A=B)$ = Complement of $[A>B + A<B]$

$\therefore \boxed{(A=B) = \overline{(A>B)+(A<B)}}$

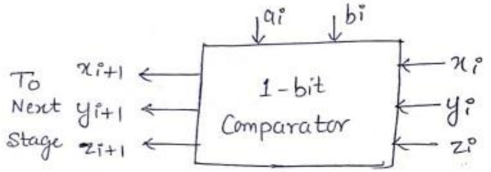




- Most often there is a requirement to compare two 4-bit, 8-bit or higher bit binary numbers.

It is worthwhile to look at a 1-bit comparator with inputs and outputs which can be used to cascade several of these to configure multiple-bit comparators.

The block diagram of 1-bit comparator at ith stage, of such a cascade is shown below.



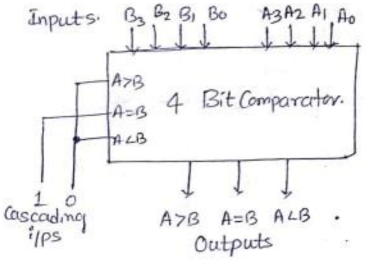
$x_i$ = A>B , $y_i$ = A=B, $z_i$ = A∠B of previous stage

$x_{i+1}$ = A>B , $y_{i+1}$ = A=B , $z_{i+1}$ = A∠B of next stage

Note: Any one of (x, y, z) will be high at an instant of time.

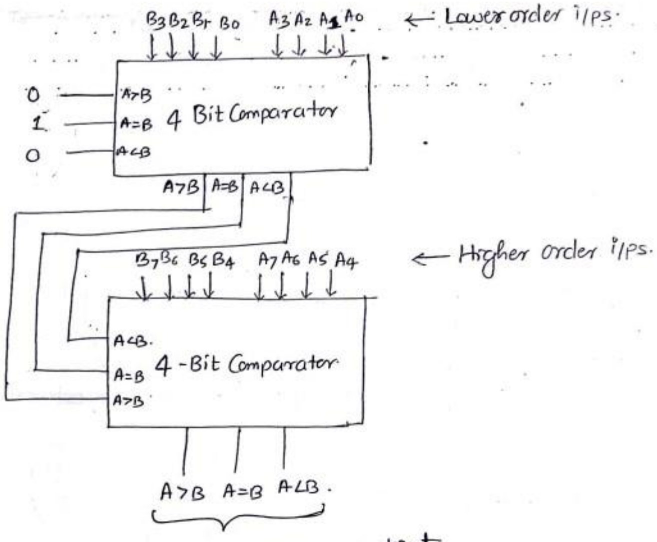
No two of them will be high at an instant of time. If such condition occurs, the outputs are don't care conditions.

## 4 Bit Comparator



| Comparing i/ps | Cascading i/ps | | | outputs. | | |
|---|---|---|---|---|---|---|
| A , B | A>B | A=B | A∠B | A>B | A=B | A∠ |
| A>B. | 0 | 1 | 0 | 1 | 0 | 0 |
| A=B | 0 | 1 | 0 | 0 | 1 | 0 |
| A∠B | 0 | 1 | 0 | 0 | 0 | 1 |
| A=B | 1 | 0 | 0 | 1 | 0 | 0 |
| | X | 1 | X | 0 | 1 | 0 |
| | 0 | 0 | 1 | 0 | 0 | 1 |

## 8-bit Comparator Using 4-bit Comparator



8-bit Comparison output

When Higher order bits of A & B are different, we can directly say that (A>B) or (A∠B). Here it doesnt caring about the lower order bit and also the cascading i/ps its getting.

When Higher order bits of A & B are same, at that time Comparator compares the lower order bits of A & B and according to that it says output (A=B).

The final 8-bit Comparator O/p will be the cascading inputs values.

| | Comparing I/ps | | Cascading I/ps | | | Outputs. | | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | A>B | A=B | A∠B | A>B | A=B | A∠B. | |
| A∠B | A>B | | X | X | X | 1 | 0 | 0 | 0 0 1 |
| | A=B | | X | 1 | X | 0 | 1 | 0 | |
| | | | 0 | 0 | 1 | 0 | 0 | 1 | |
| | | | 1 | 0 | 0 | 1 | 0 | 0 | |
| | | | 0 | 0 | 0 | 1 | 0 | 1 | |
| | | | 1 | 0 | 1 | 0 | 0 | 0 | |