EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

Binary Adder (Parallel Adder):

The 4-bit binary adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in figure below.

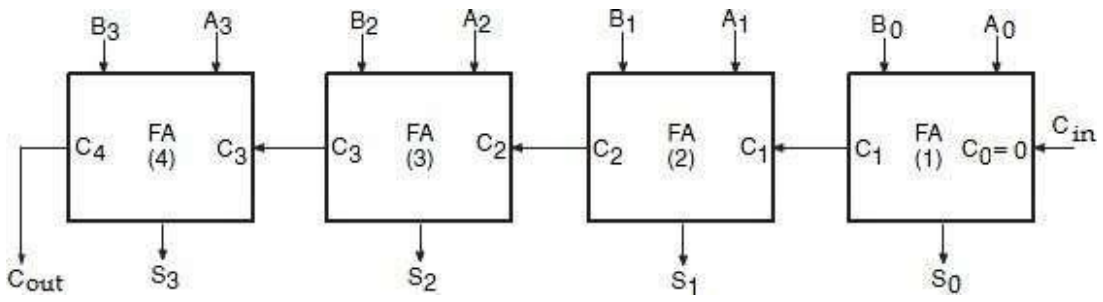


Fig : 2.11 - 4-bit binary parallel Adder

www.binils.com

Since all the bits of augend and addend are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as parallel adder.

Let the 4-bit words to be added be represented by, $A_3A_2A_1A_0$= 1111 and $B_3B_2B_1B_0$= 0011.

Significant place		4	3	2	1	
Input carry		1	1	1	0	
Augend word A :		1	1	1	1	
Addend word B :		0	0	1	1	
	1	0	0	1	0	← Sum
	↑ Output Carry					

The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C_0 in the least significant position must be 0.The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

In the least significant stage, A_0, B_0 and C_0 (which is 0) are added resulting in sum S_0 and carry C_1. This carry C_1 becomes the carry input to the second stage. Similarly in the second stage, A_1, B_1 and C_1 are added resulting in sum S_1 and carry $C_{2,}$ in the third stage, A_2, B_2 and C_2 are added resulting in sum S_2 and carry $C_{3,}$ in the third stage, A_3, B_3 and C_3 are added resulting in sum S_3 and $C_{4,}$ which is the output carry. Thus the circuit results in a sum ($S_3S_2S_1S_0$) and a carry output (C_{out}).

www.binils.com

Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. However, there are several methods to reduce this delay.

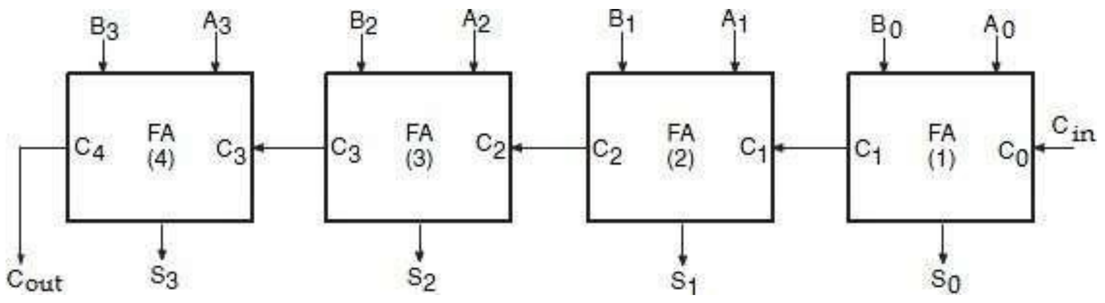
One of the methods of speeding up this process is look-ahead carry addition which eliminates the ripple-carry delay.

Carry Propagation–Look-Ahead Carry Generator:

In Parallel adder, all the bits of the augend and the addend are available for computation at the same time. The carry output of each full-adder stage is connected to the carry input of the next high-order stage. Since each bit of the sum output depends on the value of the input carry, time delay occurs in the addition process. This time delay is called as **carry propagation delay.**

For example, addition of two numbers (0011+ 0101) gives the result as 1000. Addition of the LSB position produces a carry into the second position. This carry when added to the bits of the second position, produces a carry into the third position. This carry when added to bits of the third position, produces a carry into the last position. The sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous position. i.e., the adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full adder is considered to have a propagation delay of

30nsec, then S_3 will not react its correct value until 90 nsec after LSB is generated. Therefore total time required to perform addition is 90+ 30 = 120nsec.



www.binils.com

Fig : 2.12 - 4-bit Parallel Adder

The method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions: carry generate and carry propagate.

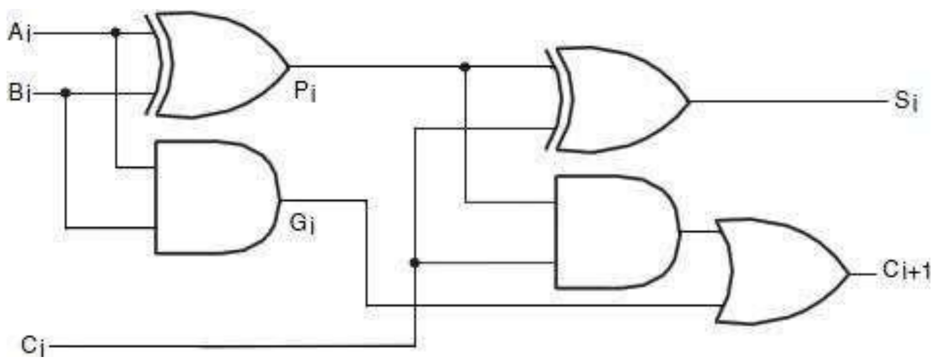


Fig 2.13-
Full-Adder circuit

www.binils.com for Anna University | Polytechnic and schools

Consider the circuit of the full-adder shown above. Here we define two functions: carry generate (G_i) and carry propagate (P_i) as,

Carry generate, $\mathbf{G_i = A_i B_i}$

Carry propagate, $\mathbf{P_i = A_i B_i}$ the output sum and carry can be expressed as,

Si = Pi Ci

Ci+1 = Gi

PiCi

G_i (carry generate), it produces a carry 1 when both Ai and Bi are 1, regardless of the

input carry C_i.

Pi (carry propagate) because it is the term associated with the propagation of the carry from C_i to C_{i+1}.

The Boolean functions for the carry outputs of each stage and substitute for each C_i its value from the previous equation:

C_0= input carry

C1= G0 + P0C0

C2= G1 + P1C1 = G1 + P1 (G0 + P0C0)

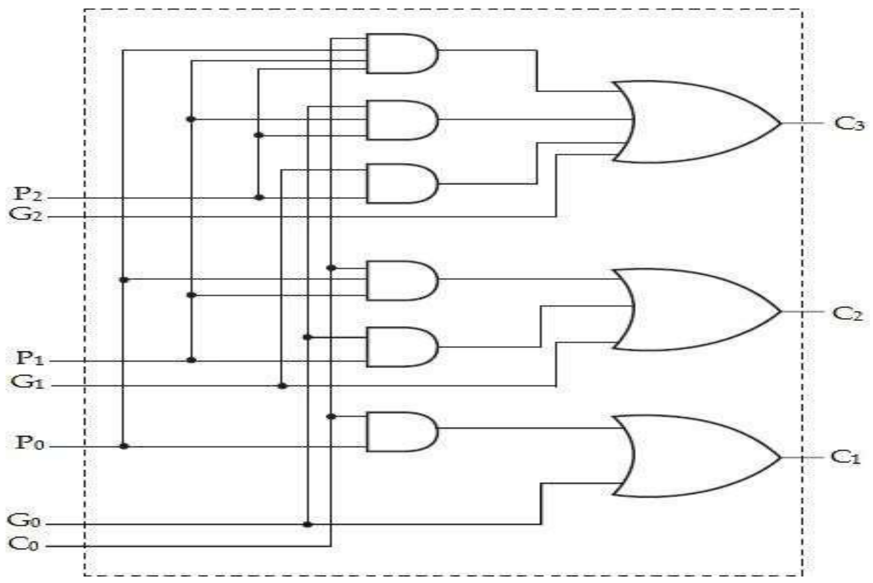
www.binils.com

Download Binils Android App in Playstore Download Photoplex

$= G1 + P1G0 + P1P0C0$

$C3 = G2 + P2C2 = G2 + P2\,(G1 + P1G0 + P1P0C0)$

$= G2 + P2G1 + P2P1G0 + P2P1P0C0$



www.binils.com

Fig : 2.14 - Logic diagram of Carry Look-ahead Generator

Since the Boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate. The three Boolean functions for C_1, C_2 and C_3 are implemented in the carry look-ahead generator as shown below. Note that C_3 does not have to wait for C_2 and C_1 to propagate; in fact C_3 is propagated at the same time as C_1 and C_2.

Using a Look-ahead Generator we can easily construct a 4-bit parallel adder with a Look-ahead carry scheme. Each sum output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the P_i variable, and the AND gate generates the G_i variable. The carries are propagated through the carry look-ahead generator and applied as inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times.

www.binils.com

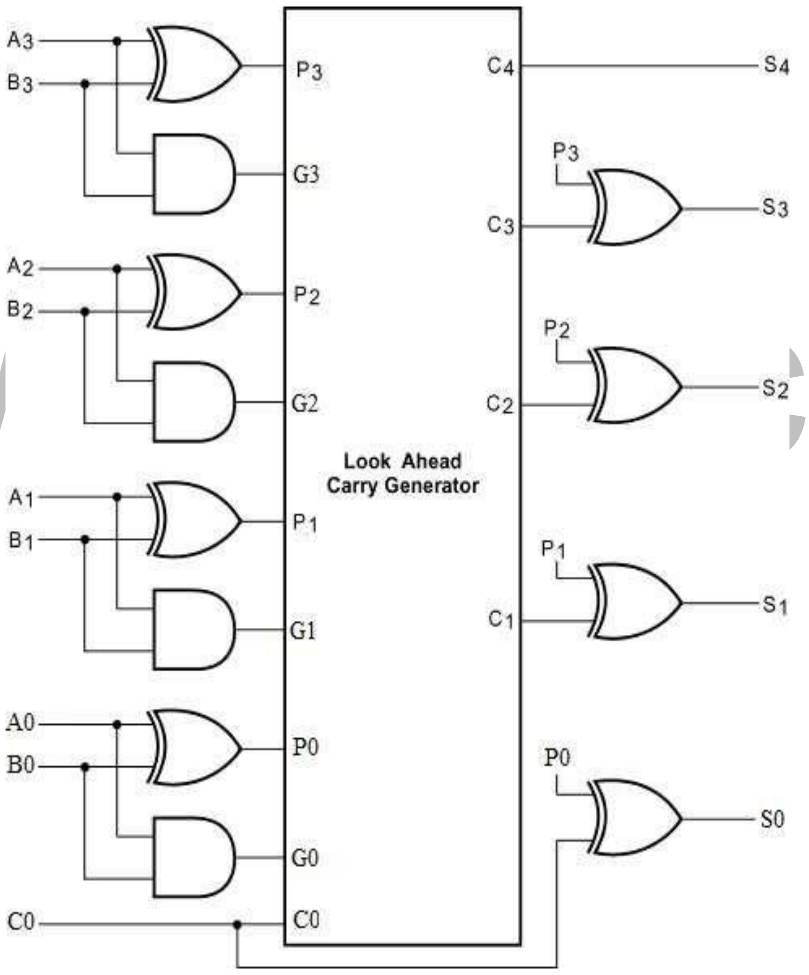


Fig : 2.15 - 4-Bit Adder with Carry Look-ahead

Binary Subtractor (Parallel Subtractor):

The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction A-B can be done by taking the 2‘s complement of B and adding it to A. The 2‘s complement can be obtained by taking the 1‘s complement and adding 1 to the least significant pair of bits. The 1‘s complement can be implemented with inverters and a 1 can be added to the sum through the input carry.

The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction. The operation thus performed becomes A, plus the 1‘s complement of B, plus1. This is equal to A plus the

2‘s complement of B.

www.binils.com

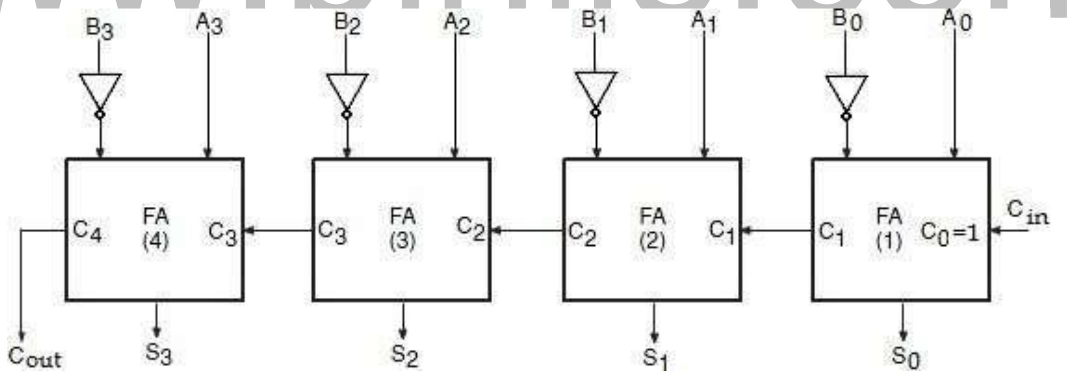


Fig : 2.16 - 4-bit Parallel Subtractor

Parallel Adder/ Subtractor:

The addition and subtraction operation can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder. A 4-bit adder Subtractor circuit is shown below.

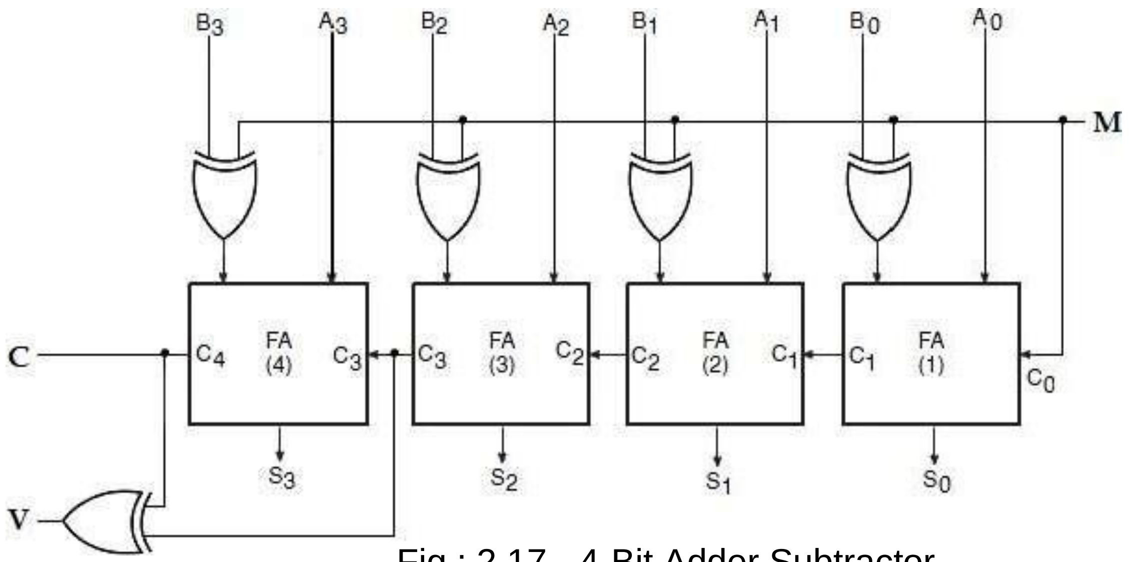


Fig : 2.17 - 4-Bit Adder Subtractor

The mode input M controls the operation. When M= 0, the circuit is an adder and when M=1, the circuit becomes a Subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When M=0, we have B 0= B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When M=1, we have

B 1= B' and C_0=1. The B inputs are all complemented and a 1 is added through the

input carry. The circuit performs the operation A plus the complement of B. The 2's exclusive-OR with output V is for detecting an overflow.

www.binils.com

Decimal Adder (BCD Adder):

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+ 9+1 = 19; the 1 is the sum being an input carry. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

These binary numbers are labeled by symbols K, Z_8, Z_4, Z_2, Z_1, K is the carry. The columns under the binary sum list the binary values that appear in the outputs of the 4- bit binary adder. The output sum of the two decimal digits must be represented in BCD.

Binary Sum					**BCD Sum**					**Decimal**
K	**Z8**	**Z4**	**Z2**	**Z1**	**C**	**S8**	**S4**	**S2**	**S1**	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

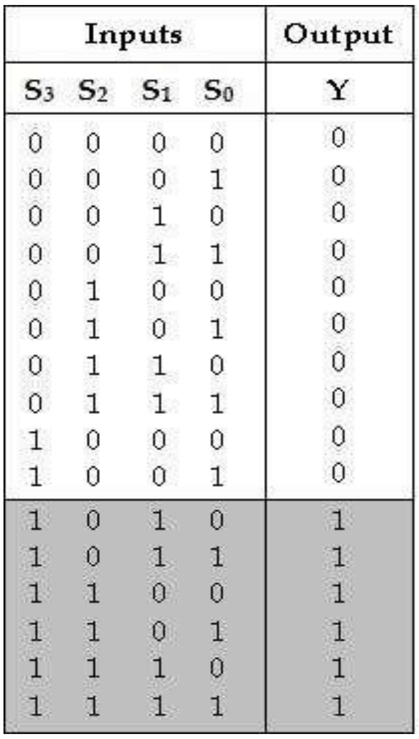
www.binils.com

0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

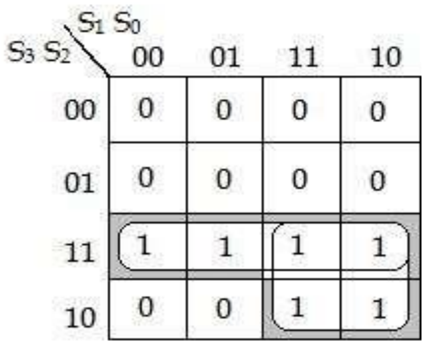
www.binils.com

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 9 (1001), we obtain a non- valid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of the given truth table.



Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



$S_3 S_2$ \ $S_1 S_0$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$Y = S_3S_2 + S_3S_1$

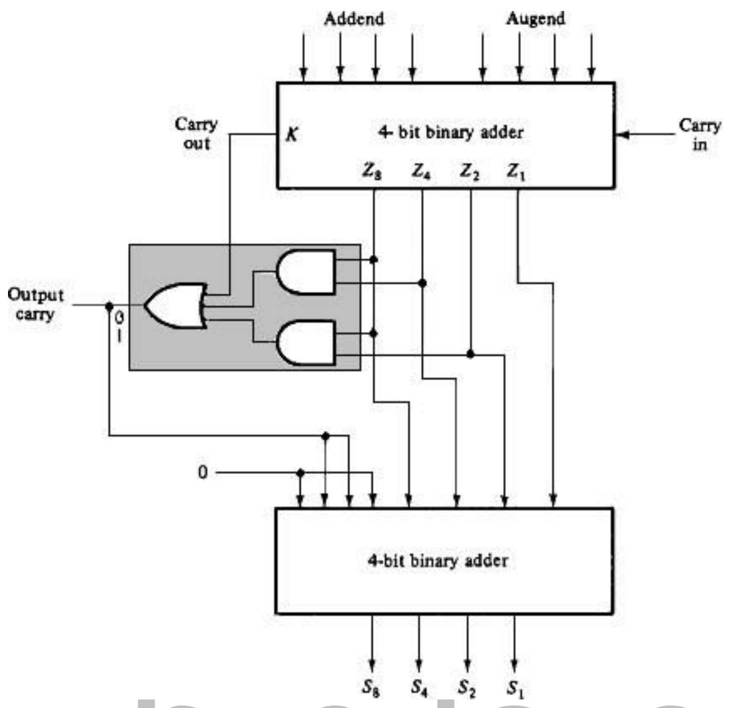
www.binils.com

To implement BCD adder we require:

x4-bit binary adder for initial addition xLogic circuit to detect sum greater than 9 and

xOne more 4-bit adder to add 0110_2 in the sum if the sum is greater than 9 or carry is 1.

The two decimal digits, together with the input carry, are first added in the top4- bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit adder. The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage.



www.binils.com

Fig : 2.18 - Block diagram of BCD adder

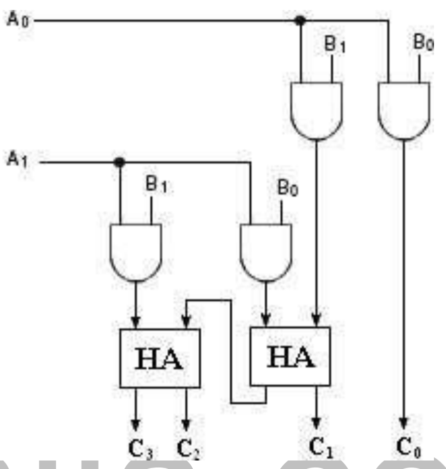
Binary Multiplier:

Multiplication of binary numbers is performed in the same way as in 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. Such partial products are shifted one position to the left. The final product is obtained from the sum of partial products.

Consider the multiplication of two 2-bit numbers. The multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is C_3, C_2, C_1 and C_0. The first partial

product is formed by multiplying A_0 by B_1B_0. The multiplication of two bits such as A_0 and B_0 produces a 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 in the diagram below.
The second partial product is formed by multiplying A_1 by B_1B_0 and shifted one position to the left. The two partial products are added with two half adder (HA) circuits.

$$\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\cline{3-4}
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}$$



www.binils.com

Fig : 2.19 - 2-bit by 2-bit Binary multiplier

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products. The least significant bit of the product does not have to go through an adder since it is formed by the output of the first AND gate.

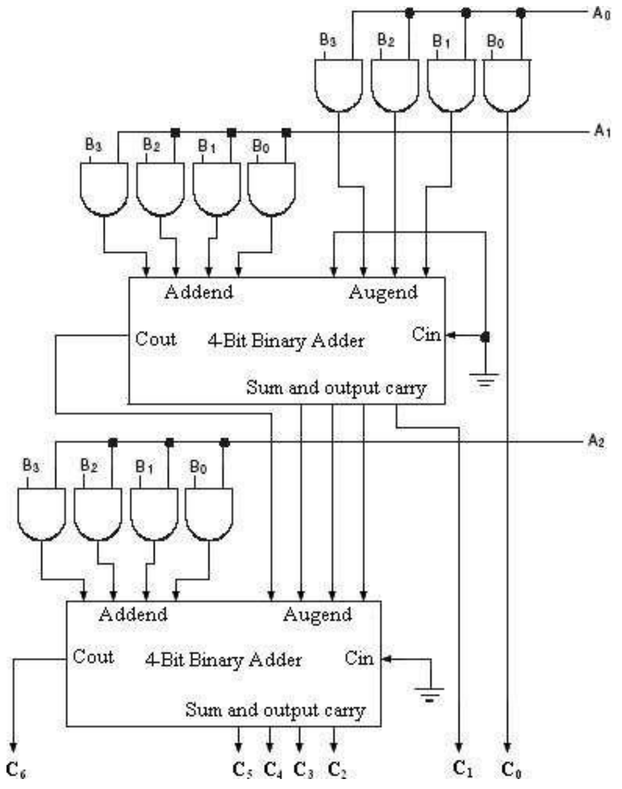
A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates are added with the partial product of the previous level to form a new partial product. The last level produces the product. For J multiplier bits and K multiplicand bits we need (J x K) AND gates and (J-1) k-bit adders to produce a product of J+K bits.

Consider a multiplier circuit that multiplies a binary number of four bits by a

number of three bits. Let the multiplicand be represented by B3, B2, B1, B0 and the multiplier by A_2, A_1, and A_0. Since K= 4 and J= 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits. The logic diagram of the multiplier is shown below.

Fig : 2.20 - 4-bit by 3-bit Binary multiplier

www.binils.com



EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

DECODERS:

A decoder is a combinational circuit that converts binary information from ‗n' input lines to a maximum of ‗2^n' unique output lines. The general structure of decoder circuit is –

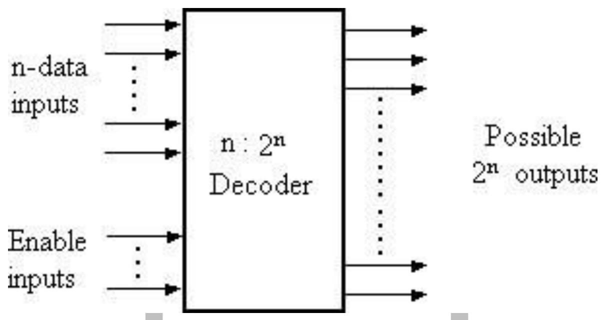


Fig : 2.24 - General structure of decoder

www.binils.com

The encoded information is presented as ‗n' inputs producing ‗2^n' possible outputs. The 2^n output values are from 0 through 2^n-1. A decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

Binary Decoder (2 to 4 decoder):

A binary decoder has ‗n' bit binary input and a one activated output out of 2^n outputs. A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.

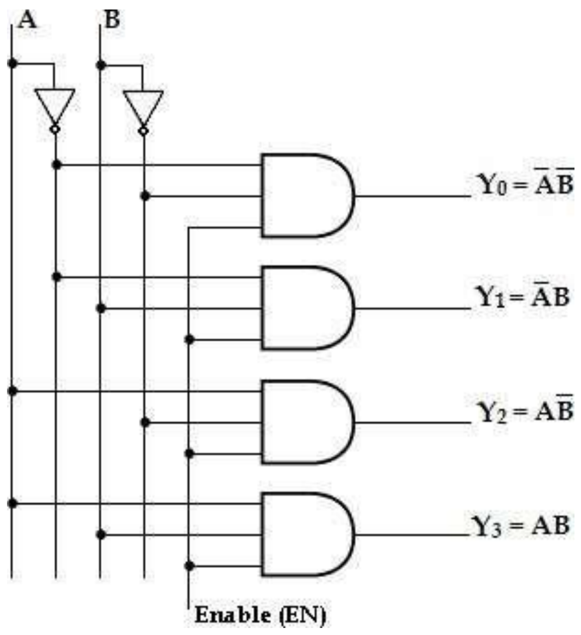


Fig : 2.25 - 2-to-4 Line decoder

Here the 2 inputs are decoded into 4 outputs, each output representing one of the minterms of the two input variables.

www.binils.com

Inputs			Outputs			
Enable	**A**	**B**	**Y3**	**Y2**	**Y1**	**Y0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

As shown in the truth table, if enable input is 1 (EN= 1) only one of the outputs ($Y_0 – Y_3$), is active for a given input.

The output Y_0 is active, ie., Y_0= 1 when inputs A= B= 0,

Y_1 is active when inputs, A= 0 and B= 1,

Y_2 is active, when input A= 1 and B= 0,

Y_3 is active, when inputs A= B= 1.

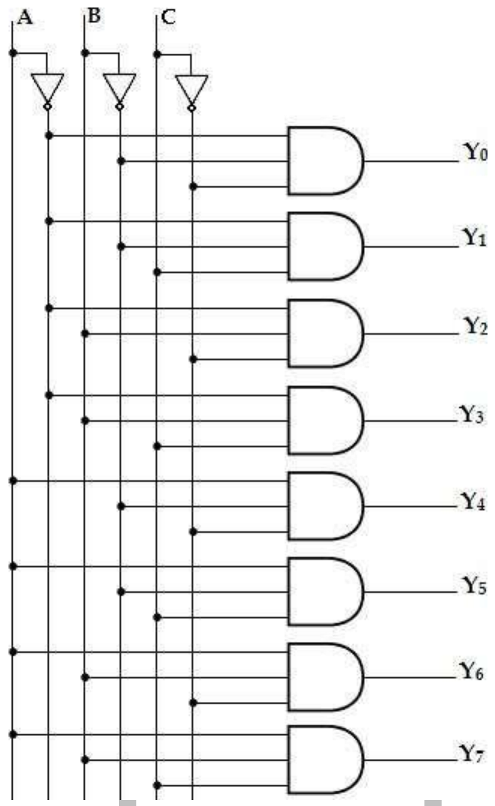
3-to-8 Line Decoder:

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y_0- Y_7). Based on the 3 inputs one of the eight outputs is selected.

The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

www.binils.com

Inputs			Outputs							
A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
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

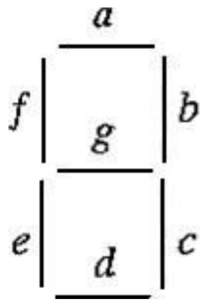


www.binils.com

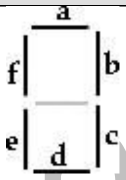
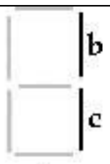
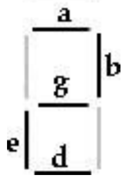
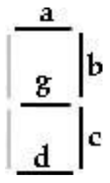
Fig : 2.26 - 3-to-8 line decoder

BCD to 7-Segment Display Decoder:

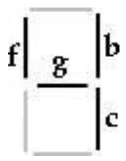
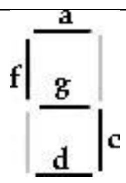
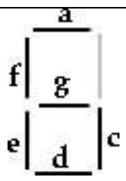
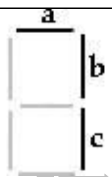
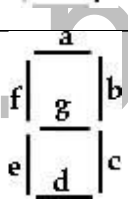
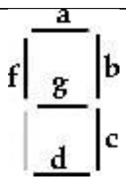
A seven-segment display is normally used for displaying any one of the decimal digits, 0 through 9. A BCD-to-seven segment decoder accepts a decimal digit in BCD and generates the corresponding seven-segment code.



Each segment is made up of a material that emits light when current is passed through it. The segments activated during each digit display are tabulated as—

Digit	Display	Segments Activated
0		a, b, c, d, e, f
1		b, c
2		a, b, d, e, g
3		a, b, c, d, g

www.binils.com

4		b, c, f, g
5		a, c, d, f, g
6		a, c, d, e, f, g
7		a, b, c
8		a, b, c, d, e, f, g
9		a, b, c, d, f, g

www.binils.com

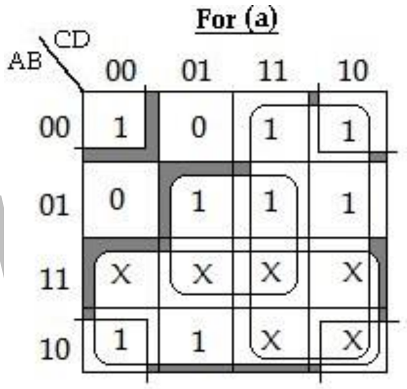
Truth table:

	BCD code				7-Segment code						
Digit	**A**	**B**	**C**	**D**	**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	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

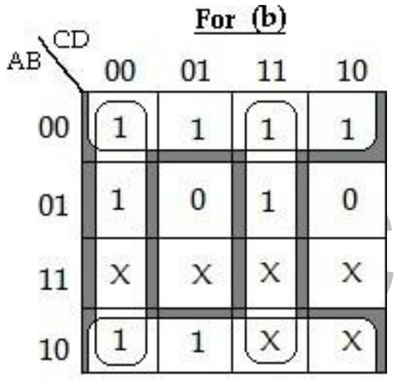
K-map Simplification:

For (a)



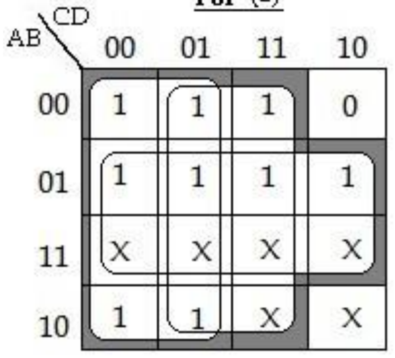
a= A+ C+ BD+ B'D'

For (b)



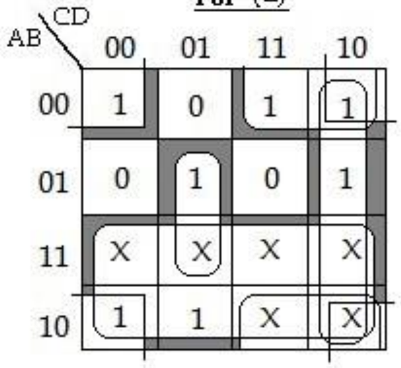
b= B'+ C'D'+ CD

For (c)

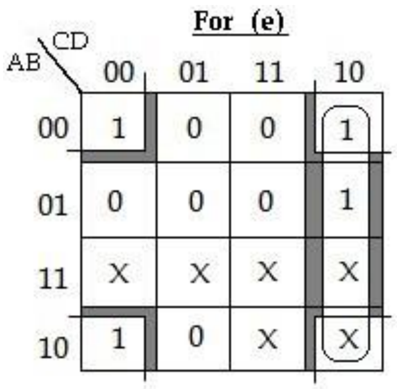


c= B+ C'+ D

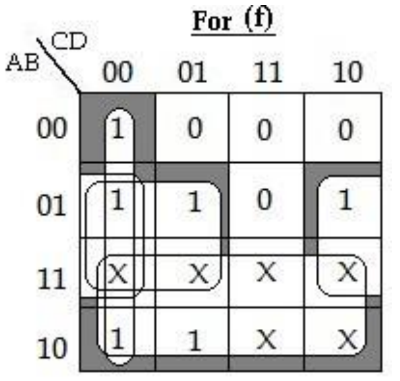
For (d)



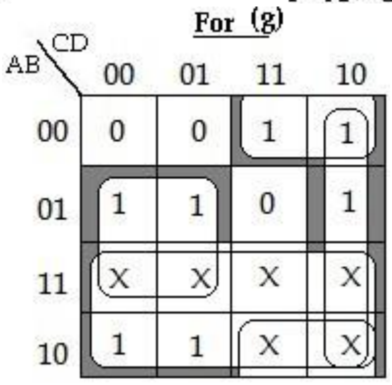
d= B'D'+ CD'+ BC'D+ B'C+ A



$e = B'D' + CD'$



$f = A + C'D' + BC' + BD'$



$g = A + BC' + B'C + CD'$

www.binils.com

Applications of decoders:

1. Decoders are used in counter system.
2. They are used in analog to digital converter.
3. Decoder outputs can be used to drive a display system.

ENCODERS:

An encoder is a digital circuit that performs the inverse operation of a decoder. Hence, the opposite of the decoding process is called encoding. An encoder is a combinational circuit that converts binary information from 2^n input lines to a maximum of ‗n' unique output lines. The general structure of encoder circuit is –

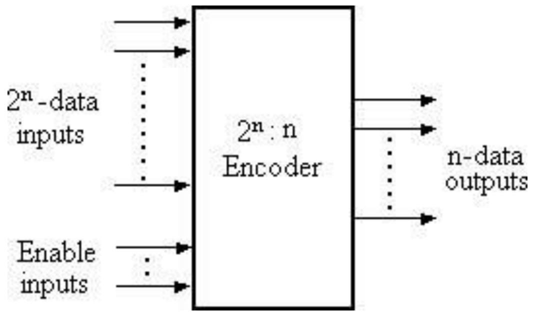


Fig : 2.27 - General structure of Encoder

It has 2^n input lines, only one which 1 is active at any time and ‗n' output lines. It encodes one of the active inputs to a coded binary output with ‗n' bits. In an encoder, the number of outputs is less than the number of inputs.

Octal-to-Binary Encoder:

It has eight inputs (one for each of the octal digits) and the three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time.

www.binils.com

Inputs								**Outputs**		
D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**A**	**B**	**C**
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

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7 and the output is 1 for digits 4, 5, 6 or 7. These conditions can be expressed by the following output Boolean functions:

$z= D_1+ D_3+ D_5+ D_7$

$y= D_2+ D_3+ D_6+ D_7$ $x= D_4+$

$D_5+ D_6+ D_7$

The encoder can be implemented with three OR gates. The encoder defined in the below table, has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination.

www.binils.com

For eg., if D_3 and D_6 are 1 simultaneously, the output of the encoder may be 111. This does not represent either D_6 or D_3. To resolve this problem, encoder circuits must establish an input priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers and if D_3 and D_6 are 1 at the same time, the output will be 110 because D_6 has higher priority than D_3.

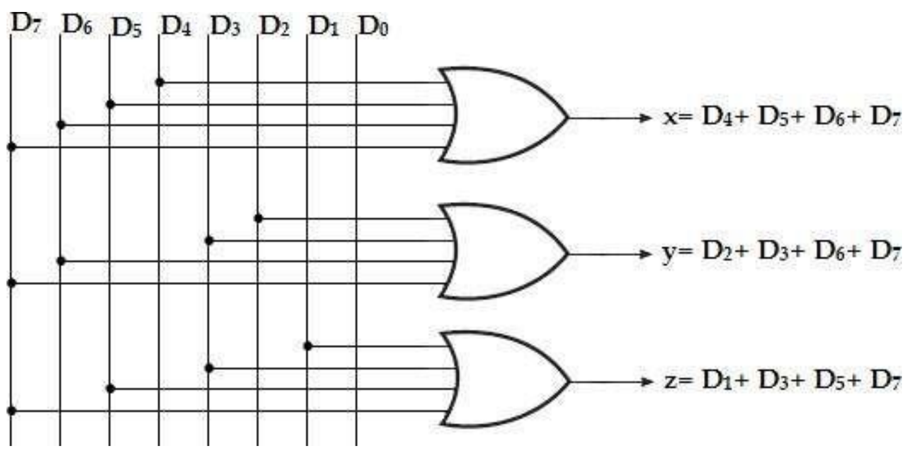


Fig : 2.28 - Octal-to-Binary Encoder

Another problem in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; this output is same as when D_0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate that atleast one input is equal to 1.

Priority Encoder:

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

In addition to the two outputs x and y, the circuit has a third output, V (valid bit indicator). It is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.

The higher the subscript number, higher the priority of the input. Input D_3, has the highest priority. So, regardless of the values of the other inputs, when D_3 is 1, the output for xy is 11.

www.binils.com

D_2 has the next priority level. The output is 10, if D_2= 1 provided D_3= 0. The output for D_1 is generated only if higher priority inputs are 0, and so on down the priority levels.

Truth table:

Inputs				**Outputs**		
D0	**D1**	**D2**	**D3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1

x	x	x	1	1	1	1

Although the above table has only five rows, when each don‘t care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the third row in the table with X100 represents minterms 0100 and 1100. The don‘t care condition is replaced by 0 and 1 as shown in the table below.

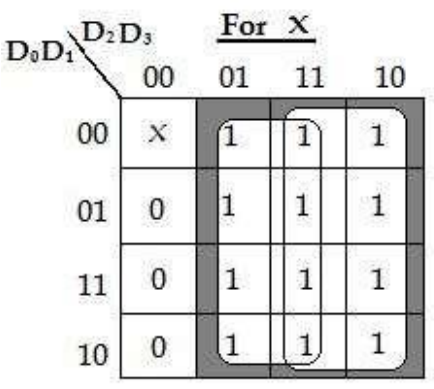
Modified Truth table:

Inputs				Outputs		
D0	**D1**	**D2**	**D3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0	1	0	1
0	1	1	0			
1	0	1	0			
1	1	1	0			
0	0	0	1	1	1	1
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

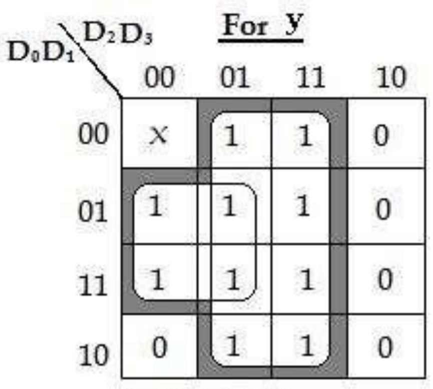
www.binils.com

www.binils.com for Anna University | Polytechnic and schools

K-map Simplification:

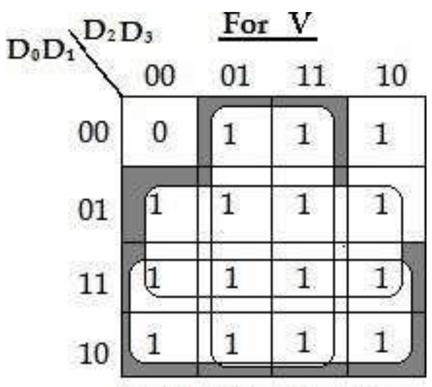


$x = D_2 + D_3$



$y = D_3 + D_1D_2$

www.binils.com



$V = D_0 + D_1 + D_2 + D_3$

The priority encoder is implemented according to the above Boolean functions.

Download Binils Android App in Playstore Download Photoplex

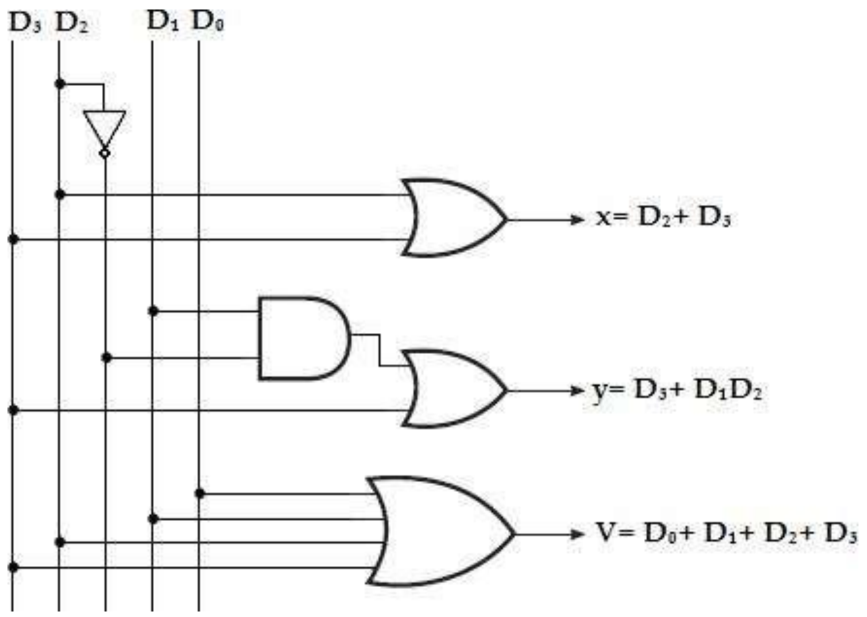


Fig : 2.29 - 4-Input Priority Encoder

www.binils.com

EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

INTRODUCTION:

The digital system consists of two types of circuits,

namely (i). Combinational circuits

(ii). Sequential circuits

Combinational circuit consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions.

Sequential logic circuit comprises both logic gates and the state of storage elements such as flip-flops. As a consequence, the output of a sequential circuit depends not only on present value of inputs but also on the past state of inputs.

www.binils.com

In the previous chapter, we have discussed binary numbers, codes, Boolean algebra and simplification of Boolean function and logic gates. In this chapter, formulation and analysis of various systematic designs of combinational circuits will be discussed.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it. Binary information from the given data transforms to desired output data in this process. Both input and output are obviously the binary signals, *i.e.*, both the input and output signals are of two possible states, logic 1 and logic 0.

For *n* number of input variables to a combinational circuit, 2^n possible combinations of binary input states are possible. For each possible combination, there is one and only one possible output combination. A combinational logic circuit can be described by *m* Boolean functions and each output can be expressed in terms of *n* input variables.

DESIGN PROCEDURE:

Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.
2. Identify the input and output variables.
3. The input and output variables are assigned letter symbols.
4. Construction of a truth table to meet input -output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. The simplified Boolean expression is obtained by any method of minimization— algebraic method, Karnaugh map method, or tabulation method.
7. A logic diagram is realized from the simplified boolean expression using logic gates.

www.binils.com

The following guidelines should be followed while choosing the preferred form for hardware implementation:

1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
2. There should be a minimum number of interconnections. 3.Limitation on the driving capability of the gates should not be ignored.

ARITHMETIC CIRCUITS – BASIC BUILDING BLOCKS

In this section, we will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction.

The basic building blocks that form the basis of all hardware used to perform the arithmetic operations on binary numbers are half-adder, full adder, half-subtractor, full- subtractor.

Half-Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.

www.binils.com

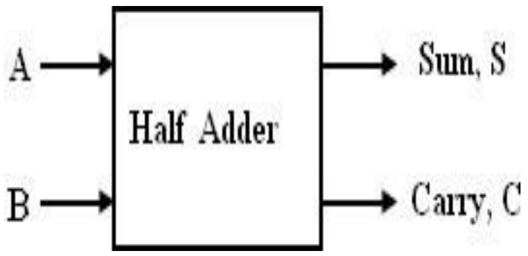


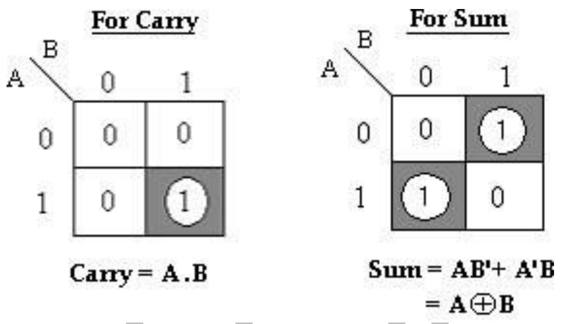
Fig 2.1 - Block schematic of half-adder

The truth table of a half-adder, showing all possible input combinations and the corresponding outputs are shown below.

Inputs		Outputs	
A	1. B	2. Carry (C)	Sum (S)
0	3. 0	4. 0	0
0	5. 1	6. 0	1
1	0	0	1
1	1	1	0

Truth table of half-adder

-map simplification for carry and sum:



www.binils.com

The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A'B+ AB'= A B

Carry, C = A . B

The first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate. The logic diagram of the half adder is,

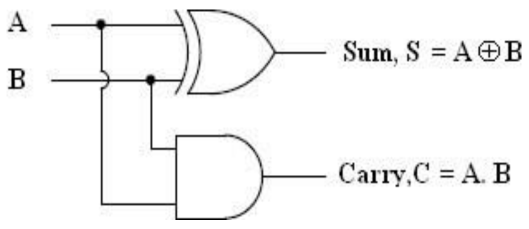
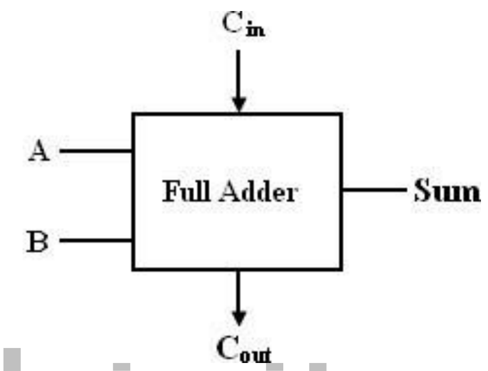


Fig 2.2 : Logic Implementation of Half-adder

Full-Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of 3 inputs and 2 outputs.

Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,



www.binils.com

Fig 2.3 - Block schematic of full-adder

The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible. The truth table is shown below,

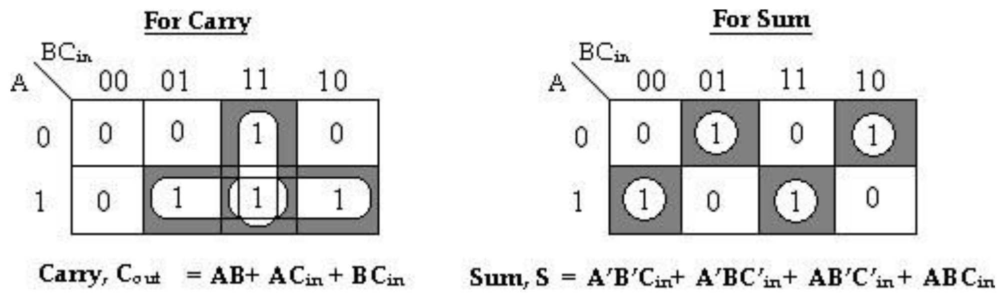
Truth Table:

Inputs			**Outputs**	
A	**B**	**Cin**	**Sum (S)**	**Carry (C_{out})**
0	0	0	0	0

0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

To derive the simplified Boolean expression from the truth table, the Karnaugh map method is adopted as,

www.binils.com


For Carry
For Sum
Carry, Cout = AB+ ACin + BCin
Sum, S = A'B'Cin+ A'BC'in+ AB'C'in+ ABCin

The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A’B’Cin+ A’BC’in + AB’C’in + ABCin Carry,

Cout = AB+ ACin + BCin .

The logic diagram for the above functions is shown as,

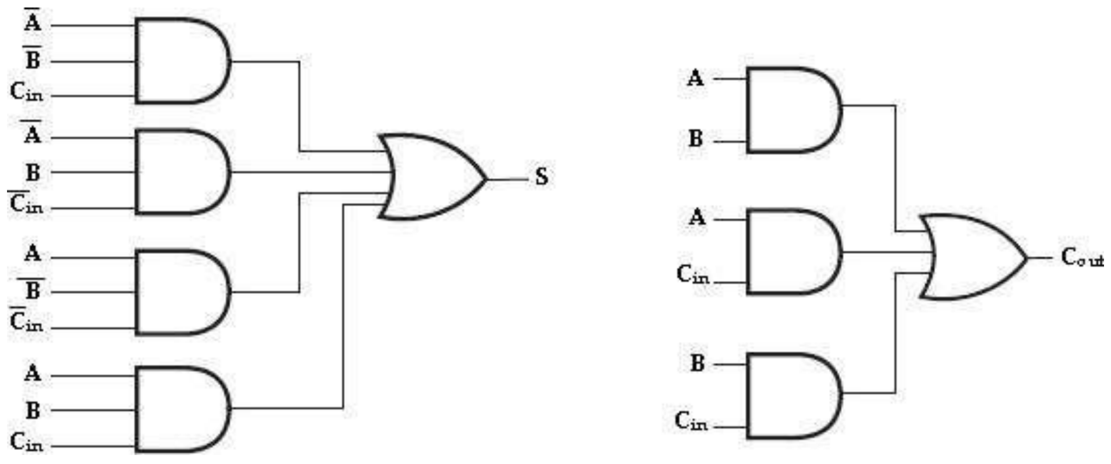


Fig : 2.4 -Implementation of full-adder in Sum of Products

The logic diagram of the full adder can also be implemented with two half-adders and one OR gate. The S output from the second half adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

www.binils.com

Sum = **C_{in} (A B)** [x y = x'y+xy']

= Cin

(A'B+AB')

= C'in (A'B+AB') + Cin (A'B+AB')' [(x'y+xy')'= (xy+x'y')]

= C'in (A'B+AB') + Cin (AB+A'B')

= A'BC'in + AB'C'in + ABCin + A'B'Cin .

and the carry output is,

Carry, C_{out} = **AB+ C_{in} (A'B+AB')**

= AB+ A'BCin+ AB'Cin

= AB (C_{in}+1) + A'BC_{in}+ AB'C_{in} [C_{in}+1= 1]

= ABCin+ AB+ A'BCin+ AB'Cin

= AB+ ACin (B+B') + A'BCin

= $AB + AC_{in} + A'BC_{in}$

= $AB(C_{in}+1) + AC_{in} + A'BC_{in}$ [$C_{in}+1 = 1$]

= $ABC_{in} + AB + AC_{in} + A'BC_{in}$

= $AB + AC_{in} + BC_{in}(A + A') =$

$AB + AC_{in} + BC_{in}$.

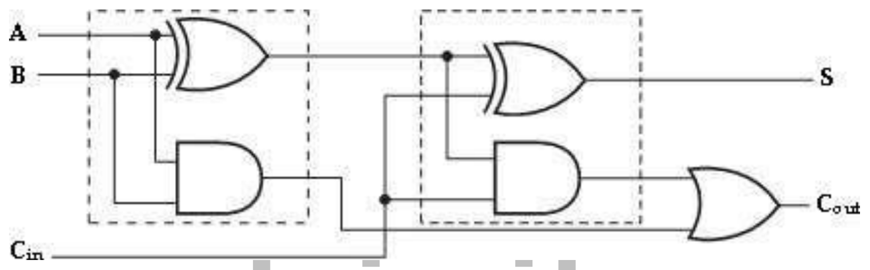


Fig : 2.5 - Implementation of full adder with two half-adders and an OR gate

www.binils.com

Half -Subtractor:

A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a ‗1‘ has been borrowed to perform the subtraction.

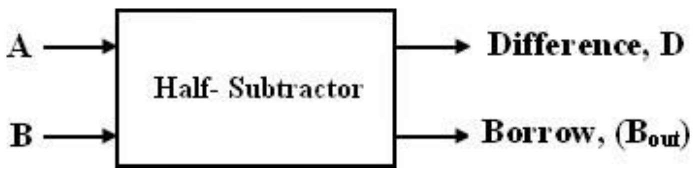


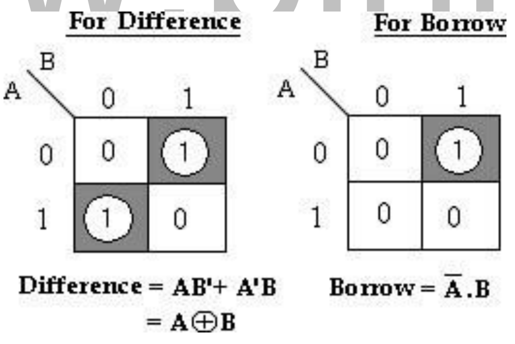
Fig : 2.6 - Block schematic of half-subtractor

The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

Input		**Output**	
A	**B**	**Difference (D)**	**Borrow (B_{out})**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map simplification for half subtractor:

www.binils.com



The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A’B+ AB’= A B Borrow, B_{out}

= A’ .B

The first one representing the DIFFERENCE (**D**)output is that of an exclusive-OR gate, the expression for the BORROW output (**B_{out}**) is that of an AND gate with input A complemented before it is fed to the gate.

The logic diagram of the half adder is,

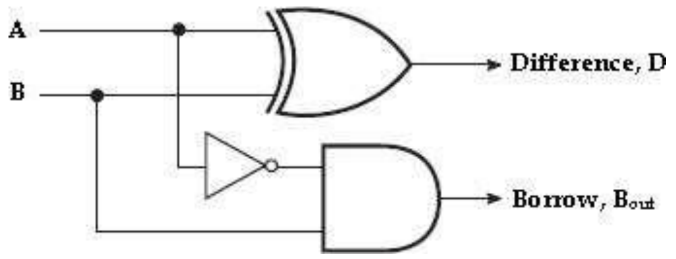


Fig : 2.7 - Logic Implementation of Half-Subtractor

Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output.

Full Subtractor:

www.binils.com

A *full subtractor* performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a ‗1' has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The

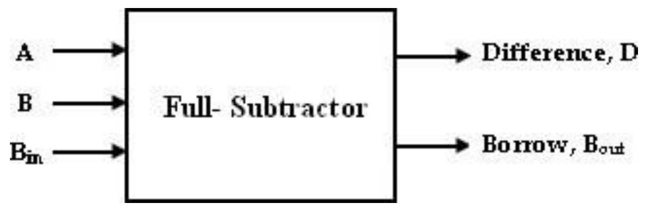


Fig : 2.8 - Block schematic of full-adder

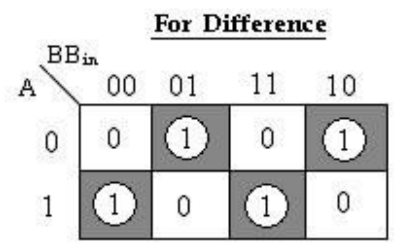
BORROW output bit tells whether the minuend bit needs to borrow a _1‘ from the next possible higher minuend bit.

The truth table for full-subtractor is,

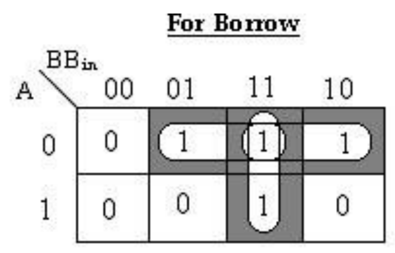
Input s			**Outputs**	
A	**B**	**Bin**	**Difference(D)**	**Borrow(B_{out})**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

www.binils.com

K-map simplification for full-subtractor:



Difference, D = $A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$



Borrow, $B_{out} = A'B + A'B_{in} + BB_{in}$

The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A'B'Bin+ A'BB'in + AB'B'in + ABBin

Borrow, Bout = A'B+ A'Cin + BBin .

The logic diagram for the above functions is shown as,

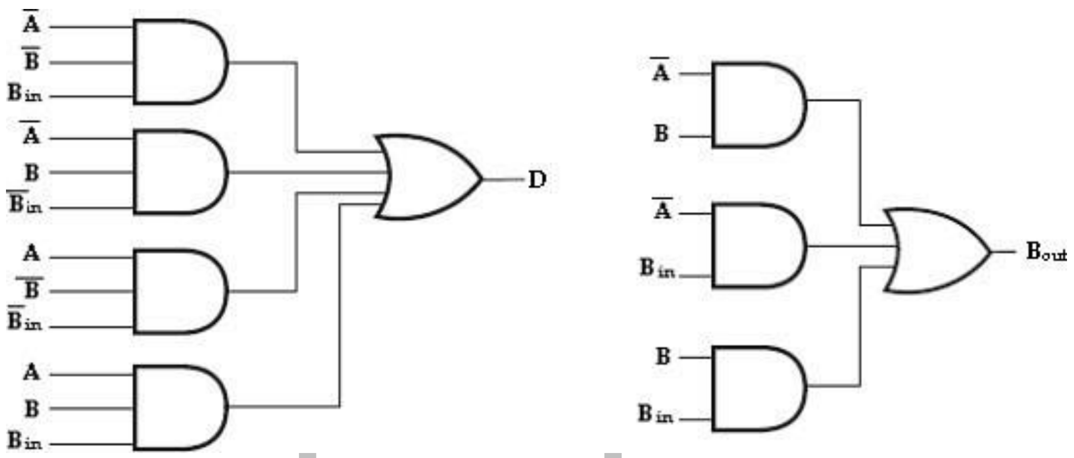


Fig : 2.9 - Implementation of full-adder in Sum of Products

www.binils.com

The logic diagram of the full-subtractor can also be implemented with two half- subtractors and one OR gate. The difference,D output from the second half subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, giving

Difference,D= **B_{in} (A B)** = [x y = x'y+xy']

Bin (A'B+AB')

= B'in (A'B+AB') + Bin (A'B+AB')' [(x'y+xy')'= (xy+x'y')]

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

= A'BB'in + AB'B'in + ABBin + A'B'Bin .

and the borrow output is,

Download Binils Android App in Playstore Download Photoplex

Borrow, Bout = **A’B+ Bin (A’B+AB’)’** [(x‘y+xy‘)‘=(xy+x‘y‘)]

= A‘B+ Bin (AB+A‘B‘)

= A‘B+ ABBin+ A‘B‘Bin

= A‘B (Bin+1) + ABBin+ A‘B‘Bin [C_{in}+1= 1]

= A‘BBin+ A‘B+ ABBin+ A‘B‘Bin [A+A‘= 1]

= A‘B+ BBin (A+A‘) + A‘B‘Bin

= A‘B+ BBin+ A‘B‘Bin

= A‘B (Bin+1) + BBin+ A‘B‘Bin [C_{in}+1= 1]

= A‘BBin+ A‘B+ BBin+ A‘B‘Bin

= A‘B+ BBin+ A‘Bin (B +B‘)= A‘B+ BBin+ A‘Bin.

www.binils.com

Therefore, we can implement full-subtractor using two half-subtractors and OR gate as,

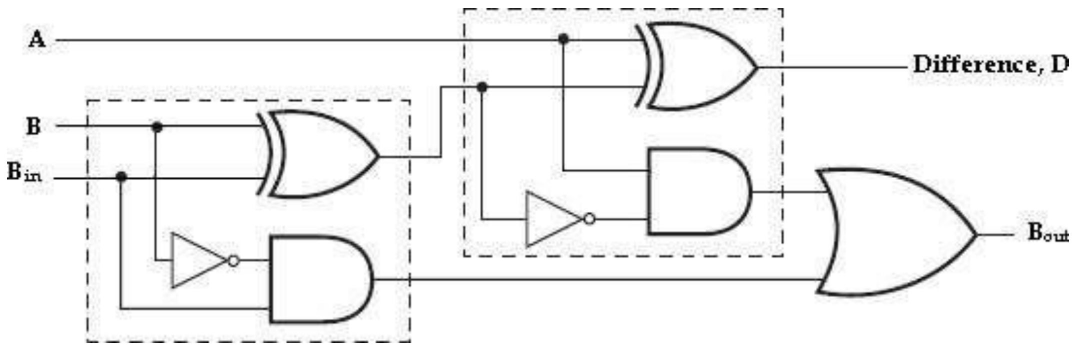


Fig : 2.10 - Implementation of full-subtractor with two half-subtractors and an OR gate

EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

MULTIPLEXER: (Data Selector)

A ***multiplexer*** or *MUX*, is a combinational circuit with more than one input line, one output line and more than one selection line. A multiplexer selects binary information present from one of many input lines, depending upon the logic status of the selection inputs, and routes it to the output line. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The multiplexer is often labeled as MUX in block diagrams.

A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line.

www.binils.com

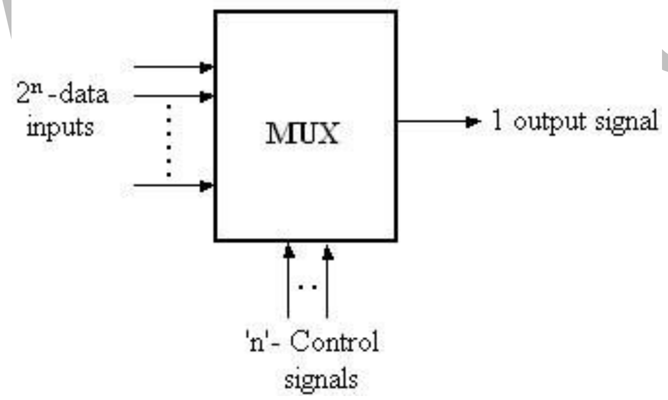


Fig : 2.30 - Block diagram of Multiplexer

2-to-1- line Multiplexer:

The circuit has two data input lines, one output line and one selection line, S. When S= 0, the upper AND gate is enabled and I_0 has a path to the output.

When S=1, the lower AND gate is enabled and I_1 has a path to the output.

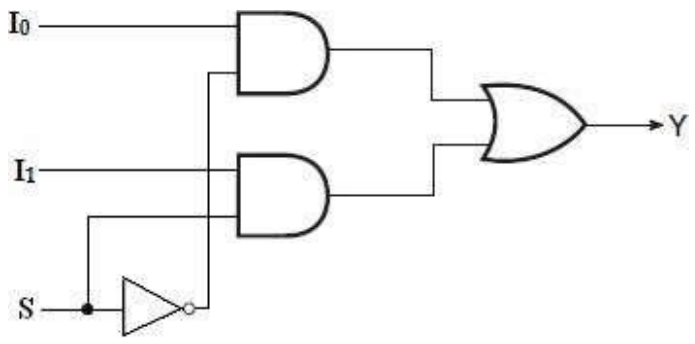


Fig : 2.31 - Logic diagram

The multiplexer acts like an electronic switch that selects one of the two sources.

Truth table:

S	Y
0	I0
1	I1

www.binils.com

4-to-1-line Multiplexer:

A 4-to-1-line multiplexer has four (2n) input lines, two (n) select lines and one output line. It is the multiplexer consisting of four input channels and information of one of the channels can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input channel is possible by two selection inputs.

Each of the four inputs I_0 through I_3, is applied to one input of AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.

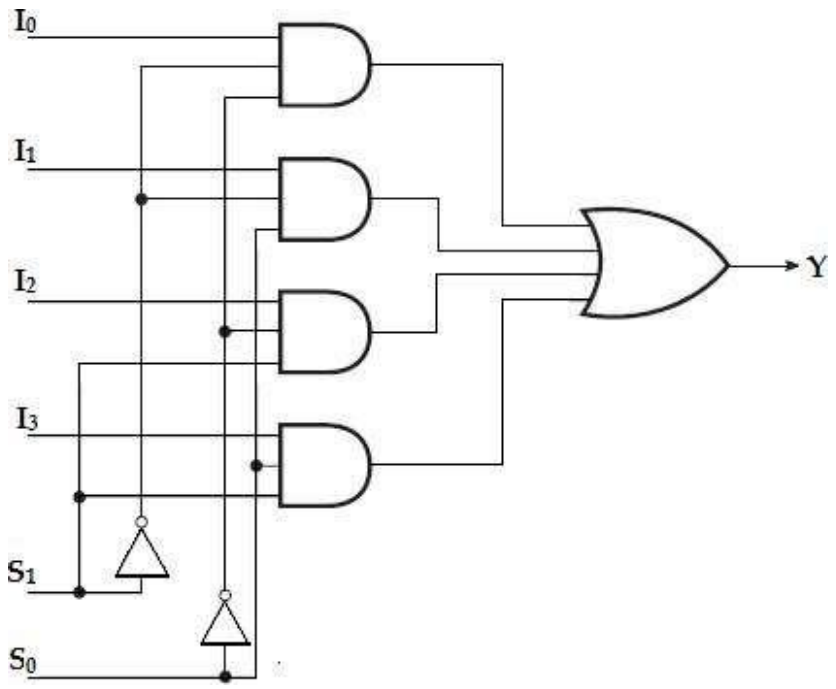


Fig : 2.32 - 4-to-1-Line Multiplexer

www.binils.com

Function table:

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

To demonstrate the circuit operation, consider the case when S_1S_0= 10. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I_0 only if S_1= 0 and S_0= 0; Y= $I_0S_1'S_0'$. The data output is equal to I_1 only if S_1= 0 and S_0= 1; Y= $I_1S_1'S_0$.

The data output is equal to I_2 only if S_1= 1 and S_0= 0; Y= $I_2S_1S_0'$.

The data output is equal to I_3 only if S_1= 1 and S_0= 1; Y= $I_3S_1S_0$. When these terms are ORed, the total expression for the data output is,

Y= I0S1'S0'+ I1S1'S0 +I2S1S0'+ I3S1S0.

As in decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer.

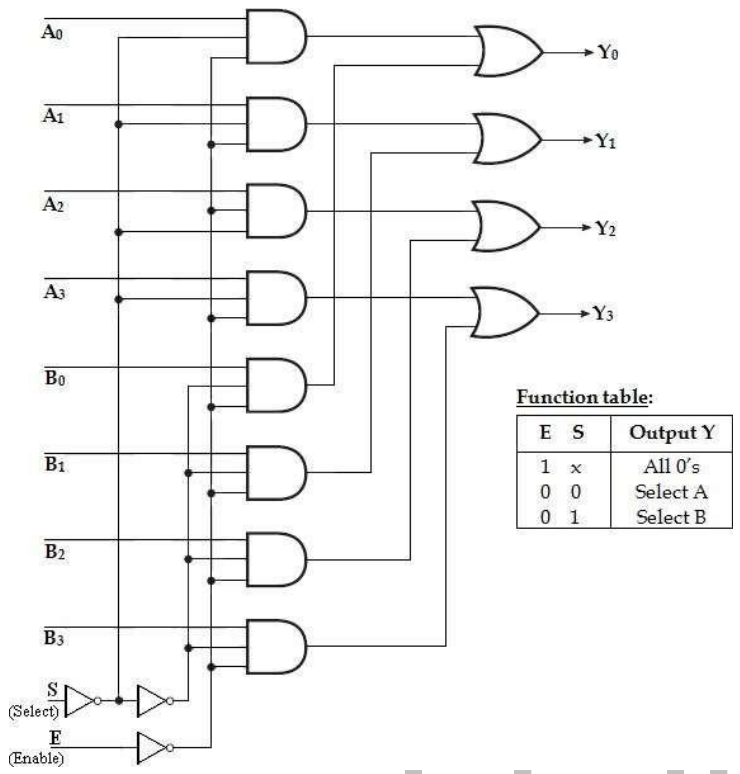
www.binils.com

Quadruple 2-to-1 Line Multiplexer:

This circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either A0 or B0. Similarly, output Y1 may have the value of A1 or B1, and so on. Input selection line, S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal operation.

Although the circuit contains four 2-to-1-Line multiplexers, it is viewed as a circuit that selects one of two 4-bit sets of data lines. The unit is enabled when E= 0. Then if S= 0, the four A inputs have a path to the four outputs. On the other hand, if

S=1, the four B inputs are applied to the outputs. The outputs have all 0's when E= 1, regardless of the value of S.



Function table:

E	S	Output Y
1	x	All 0's
0	0	Select A
0	1	Select B

Fig : 2.33 – Quadruple 2 to 1 MUX

www.binils.com

Application:

The multiplexer is a very useful MSI function and has various ranges of applications in data communication. Signal routing and data communication are the important applications of a multiplexer. It is used for connecting two or more sources to guide to a single destination among computer units and it is useful for constructing a common bus system. One of the general properties of a multiplexer is that Boolean functions can be implemented by this device.

Implementation of Boolean Function using MUX:

Any Boolean or logical expression can be easily implemented using a multiplexer. If a Boolean expression has (n+1) variables, then ‗n' of these variables can be connected to the select lines of the multiplexer. The remaining single variable along with constants 1 and 0 is used as the input of the multiplexer. For example, if C is the single variable, then the inputs of the multiplexers are C, C', 1 and 0. By this method any logical expression can be implemented.

In general, a Boolean expression of (n+1) variables can be implemented using a multiplexer with 2^n inputs.

www.binils.com

1. **Implement the following boolean function using 4: 1 multiplexer,**
   F (A, B, C) = ∑m (1, 3, 5, 6).

   Solution: Variables, n= 3 (A, B, C)

   Select lines= n- 1 = 2 (**S_1, S_0**) 2^{n-1} to MUX i.e., 2^2 to 1 = 4 to 1 MUX

   Input lines= $2^{n-1} = 2^2 = 4$ (**D_0, D_1, D_2, D_3**)

Implementation table:

Apply variables A and B to the select lines. The procedures for implementing the function are:

i. List the input of the multiplexer

ii. List under them all the minterms in two rows as shown below.

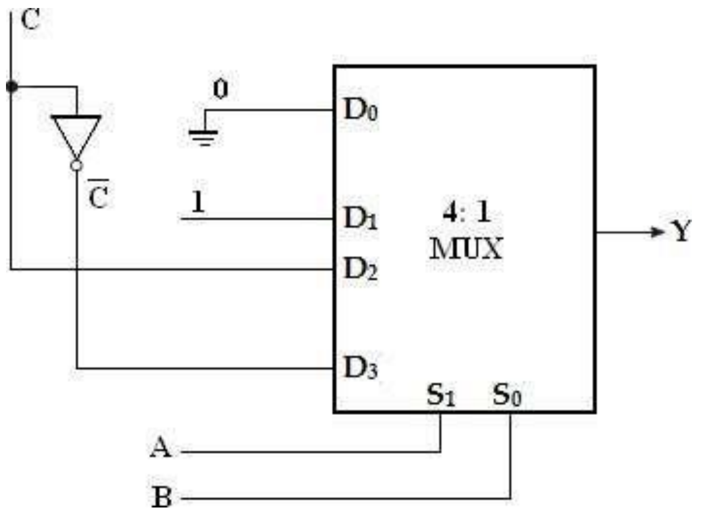
The first half of the minterms is associated with A‘ and the second half with A. The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer.

1. If both the minterms in the column are not circled, apply 0 to the corresponding input.
2. If both the minterms in the column are circled, apply 1 to the corresponding input.
3. If the bottom minterm is circled and the top is not circled, apply C to the input. 4.If the top minterm is circled and the bottom is not circled, apply C‘ to the input.

www.binils.com

	D_0	D_1	D_2	D_3
$\bar{C}$	0	①	2	③
C	4	⑤	⑥	7
	0	1	C	$\bar{C}$

Multiplexer Implementation:

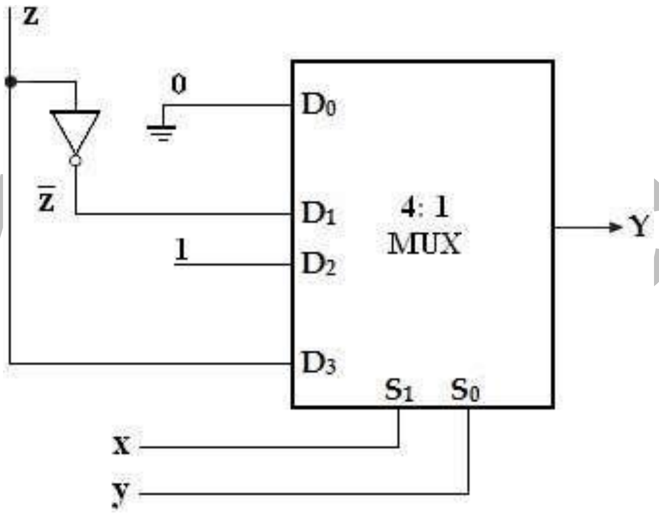


2. **F (x, y, z) = ∑m (1, 2, 6, 7) <u>Solution:</u>**

<u>Implementation table:</u>

	D_0	D_1	D_2	D_3
$\bar{z}$	0	1	2	3
z	4	5	6	7
	0	$\bar{z}$	1	z

<u>Multiplexer Implementation:</u>



3. **F (A, B, C) = ∑m (1, 2, 4, 5)**

<u>Solution:</u> Variables, n= 3 (A, B, C)

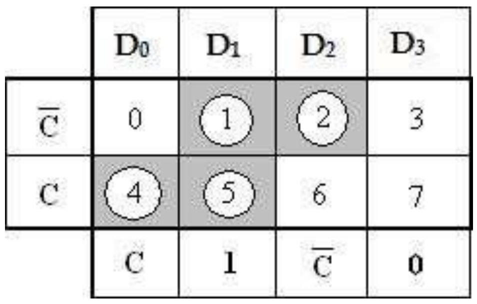
Select lines= n- 1 = 2 (**S_1, S_0**) 2^{n-1} to

MUX i.e., 2^2 to 1 = 4

to 1 MUX

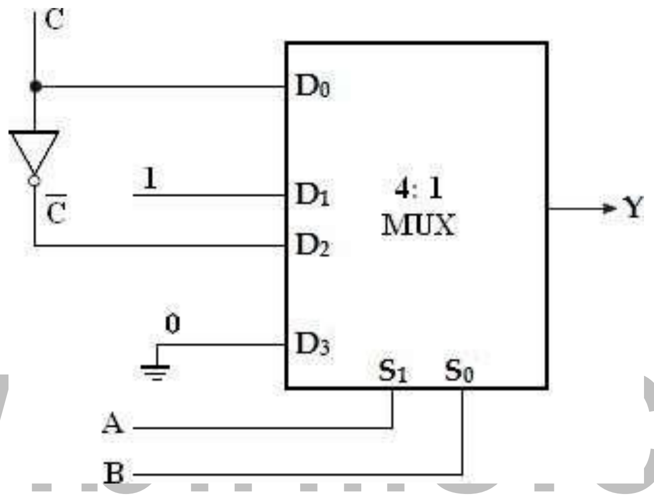
Input lines= 2^{n-1} = 2^2 = 4 (**D_0, D_1, D_2, D_3**) **<u>Implementation</u>**

<u>table:</u>



	D_0	D_1	D_2	D_3
$\overline{C}$	0	1	2	3
C	4	5	6	7
	C	1	$\overline{C}$	0

Multiplexer Implementation:



www.binils.com

4. **F(P, Q, R, S)= ∑m (0, 1, 3, 4, 8, 9, 15)**

Solution:

Variables, n= 4 (P, Q, R, S) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

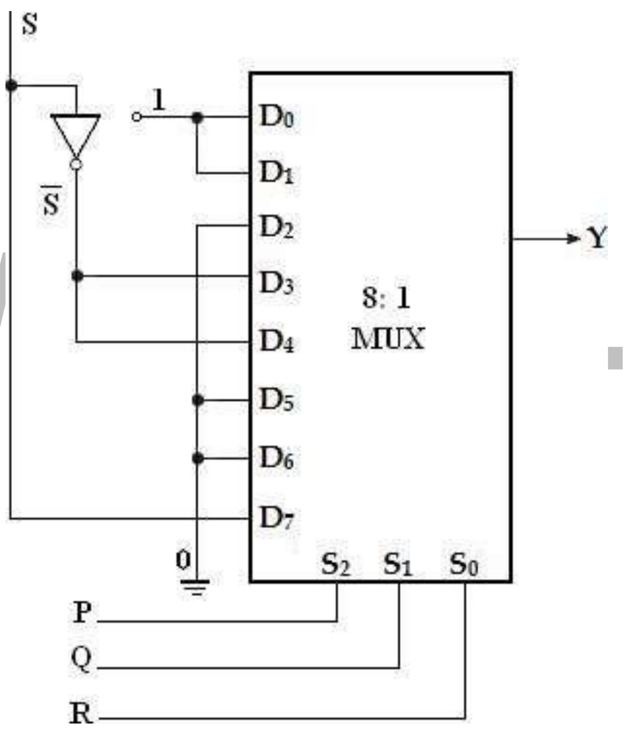
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	0	1	2	3	4	5	6	7
S	8	9	10	11	12	13	14	15
	1	1	0	$\overline{S}$	$\overline{S}$	0	0	S

Multiplexer Implementation:



www .com

5. **Implement the Boolean function using 8: 1 and also using 4:1 multiplexer F (A, B, C, D) = ∑m (0, 1, 2, 4, 6, 9, 12, 14)**

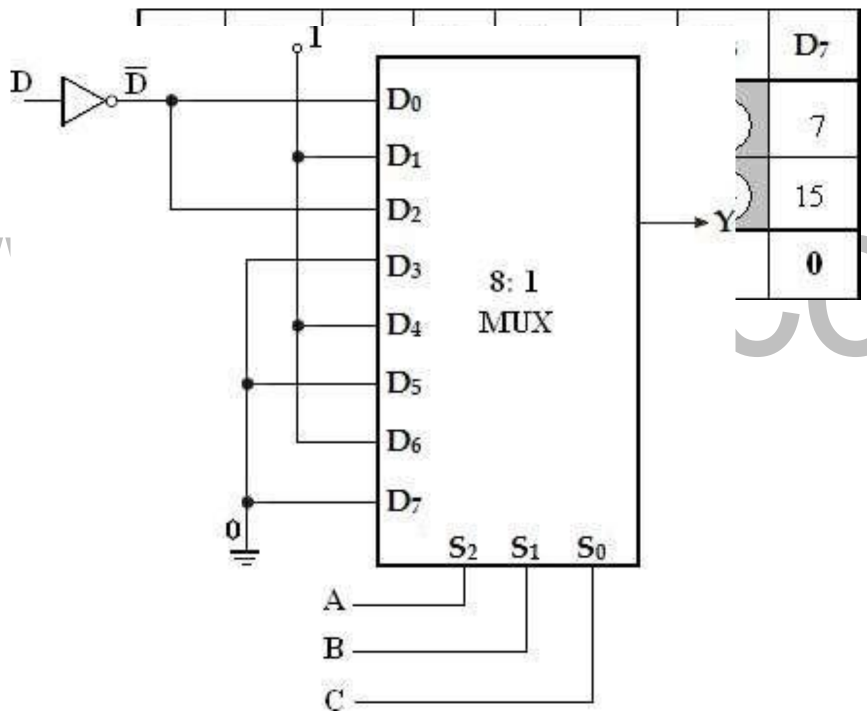
Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

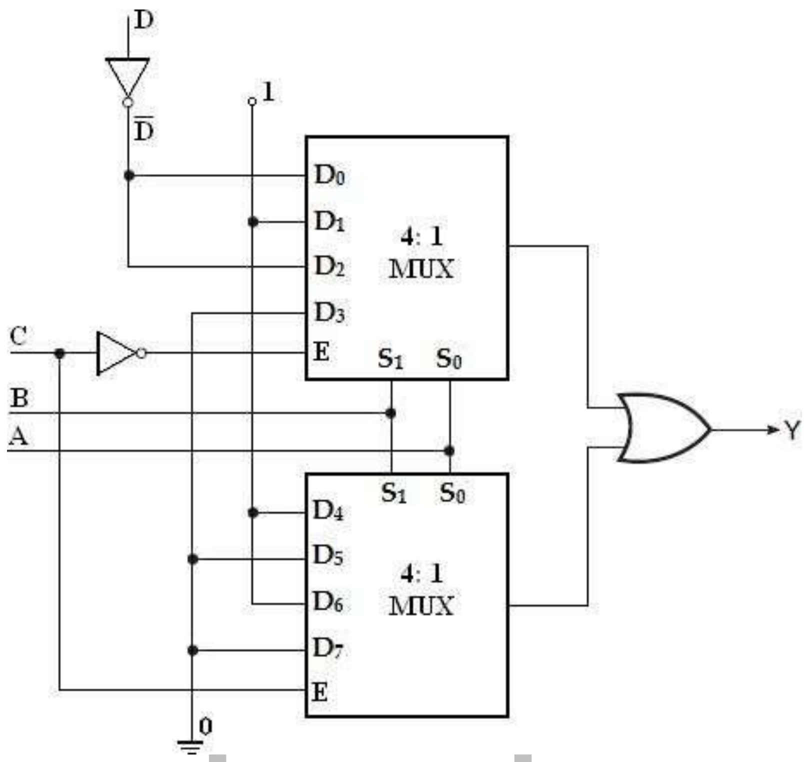
Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:



Multiplexer Implementation (Using 8: 1 MUX):

Using 4: 1 MUX:



www.binils.com

6. F (A, B, C, D) = ∑m (1, 3, 4, 11, 12, 13, 14, 15)

Solution:

Variables, n= 4 (A, B, C, D) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

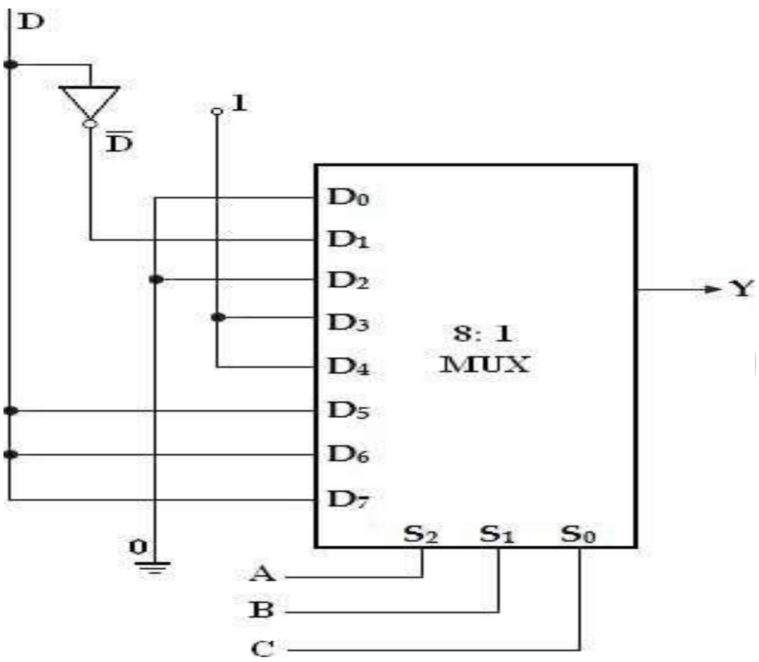
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	①	2	③	④	5	6	7
D	8	9	10	⑪	⑫	⑬	⑭	⑮
	0	$\overline{D}$	**0**	**1**	**1**	D	D	D

Multiplexer Implementation:



www m

7. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = A’BD’ + ACD + B’CD + A’C’D.

Solution:

Convert into standard SOP form,

= A‘BD‘ (C‘+C) + ACD (B‘+B) + B‘CD (A‘+A) + A‘C‘D (B‘+B)

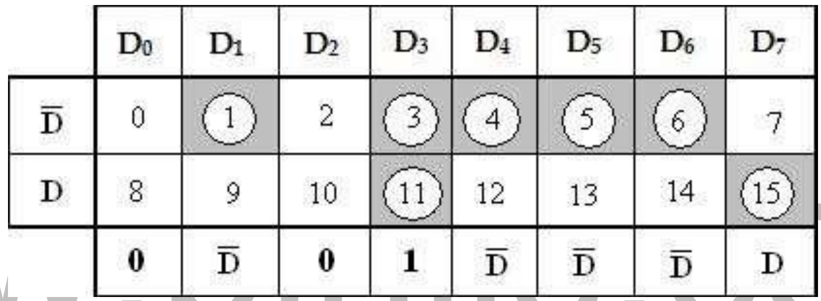
= A‘BC‘D‘ + A‘BCD‘+ AB‘CD+ ABCD +A‘B‘CD + AB‘CD+A‘B‘C‘D+ A‘BC‘D

= A‘BC‘D‘ + A‘BCD‘+ AB‘CD + ABCD +A‘B‘CD +A‘B‘C‘D+ A‘BC‘D

= m4+ m6+ m11+ m15+ m3+ m1+ m5

= ∑m (1, 3, 4, 5, 6, 11, 15)

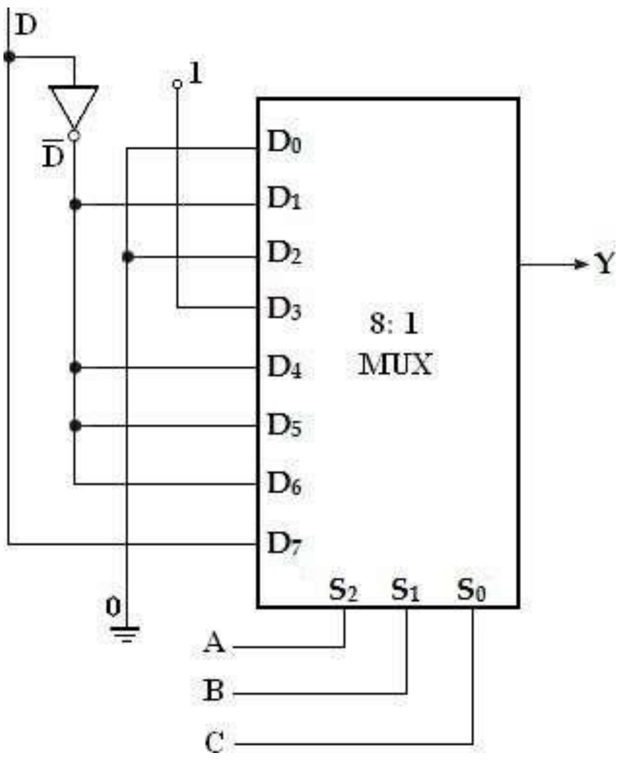
Implementation table:



	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	0	$\overline{D}$	**0**	**1**	$\overline{D}$	$\overline{D}$	$\overline{D}$	D

www.binils.com

Multiplexer Implementation:



www.binils.com

8. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = AB'D + A'C'D + B'CD' + AC'D.

Solution:

Convert into standard SOP form,

= AB'D (C'+C) + A'C'D (B'+B) + B'CD' (A'+A) + AC'D (B'+B)

= AB'C'D+ AB'CD+ A'B'C'D + A'BC'D +A'B'CD' + AB'CD' +AB'C'D+ ABC'D= AB'C'D

+ AB'CD+ A'B'C'D + A'BC'D +A'B'CD' + AB'CD'+ ABC'D = m9+ m11+

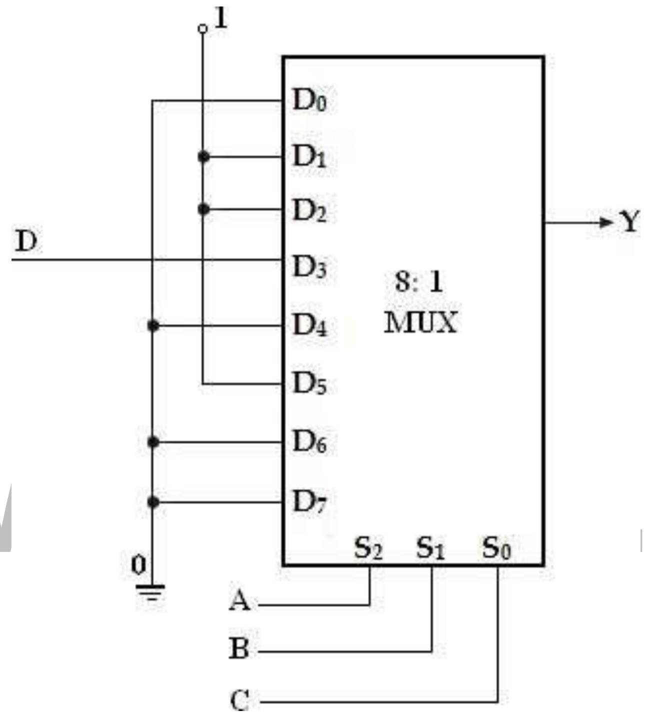
m1+ m5+ m2+ m10+ m13

= ∑m (1, 2, 5, 9, 10, 11, 13).

Implementation Table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	0	**1**	**1**	D	**0**	**1**	**0**	**0**

Multiplexer Implementation:



www.binils.com

9. Implement the Boolean function using 8: 1 and also using 4:1 multiplexer **F (w, x, y, z) = ∑m (1, 2, 3, 6, 7, 8, 11, 12, 14)**

Solution:

Variables, n= 4 (w, x, y, z)

Select lines= n-1 = 3 (**S_2, S_1, S_0**)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

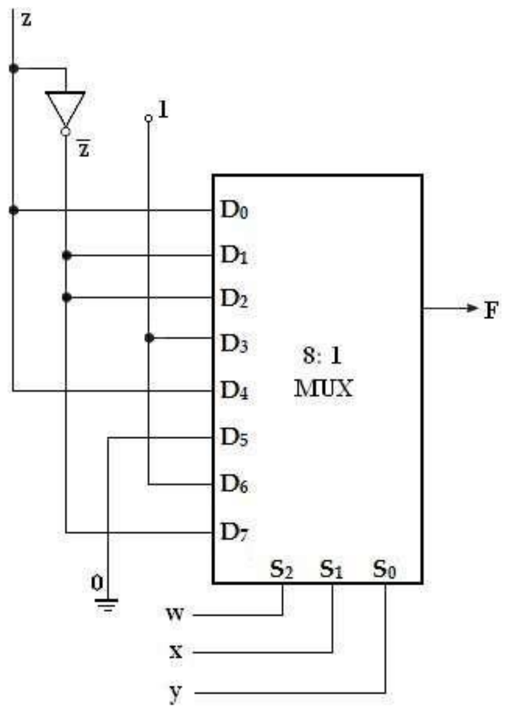
Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

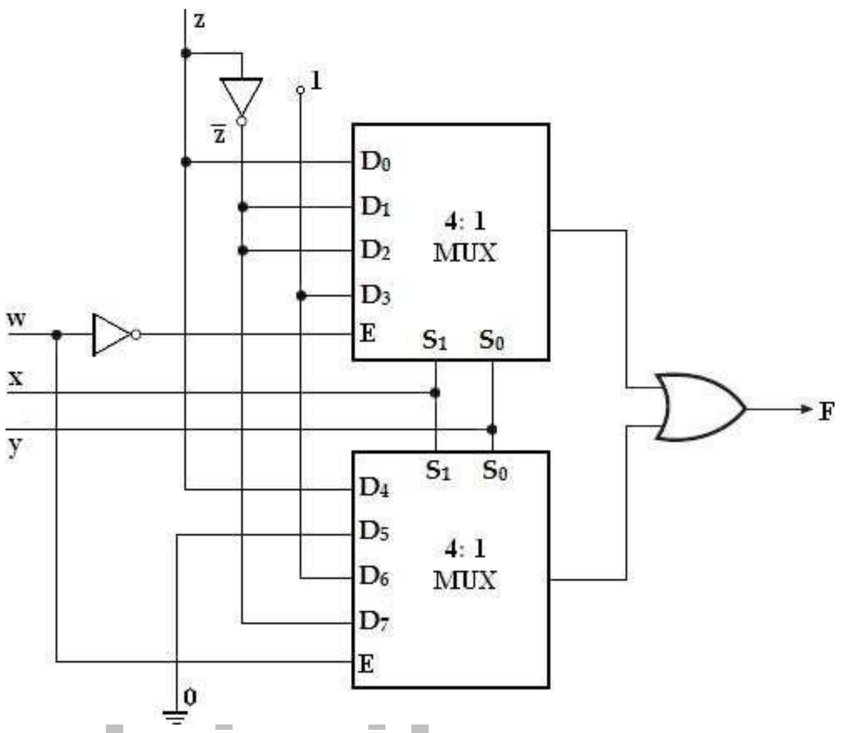
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{z}$	0	①	②	③	4	5	⑥	⑦
z	⑧	9	10	⑪	⑫	13	⑭	15
	z	$\bar{z}$	$\bar{z}$	**1**	z	**0**	**1**	$\bar{z}$

Multiplexer Implementation (Using 8:1 MUX):

www.binils.com



(Using 4:1 MUX):



www.binils.com

10. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∏m (0, 3, 5, 8, 9, 10, 12, 14) **Solution:**

Variables, n= 4 (A, B, C, D) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

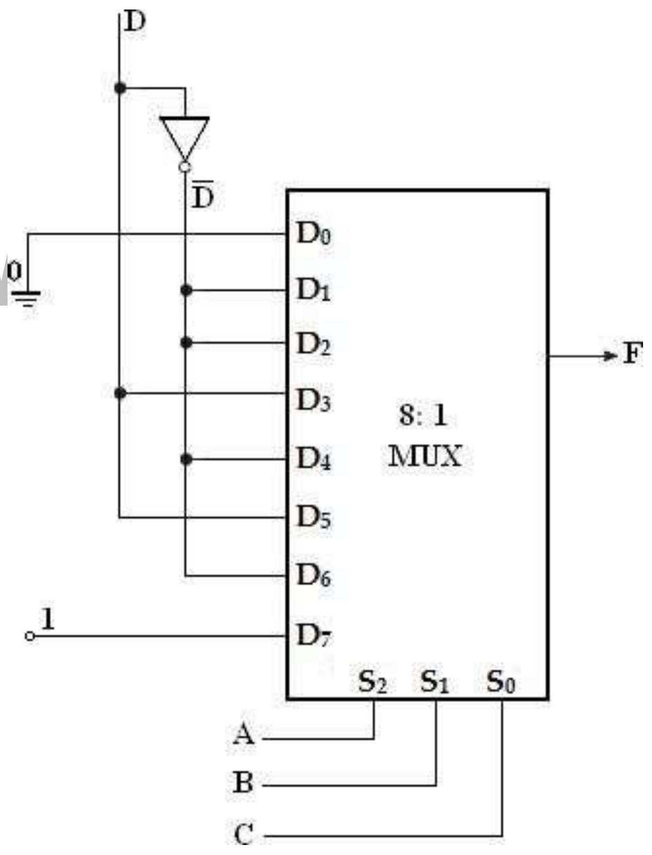
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{D}$	0	①	②	3	④	5	⑥	⑦
D	8	9	10	⑪	12	⑬	14	⑮
	0	$\bar{D}$	$\bar{D}$	D	$\bar{D}$	D	$\bar{D}$	**1**

Multiplexer Implementation:



www com

11. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∑m (0, 2, 6, 10, 11, 12, 13) + d (3, 8, 14)

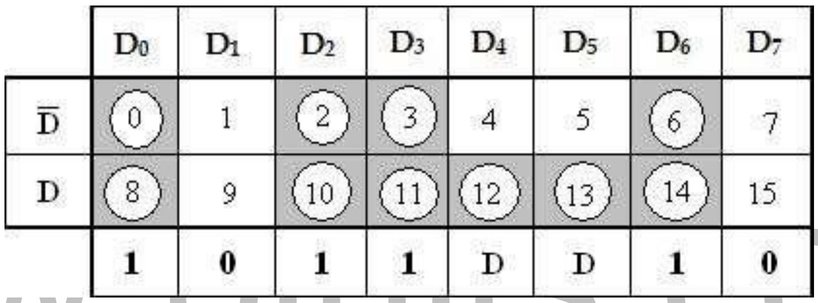
Solution:

Variables, n= 4 (A, B, C, D) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

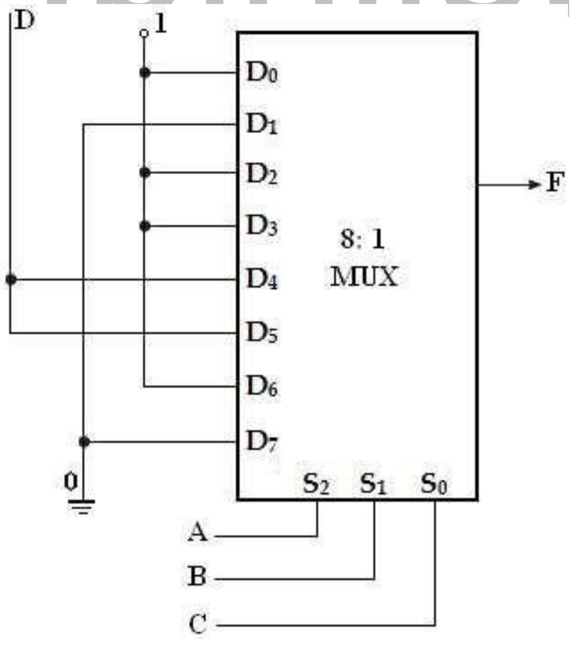
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**) **Implementation Table:**



	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	1	**0**	**1**	**1**	D	D	**1**	**0**

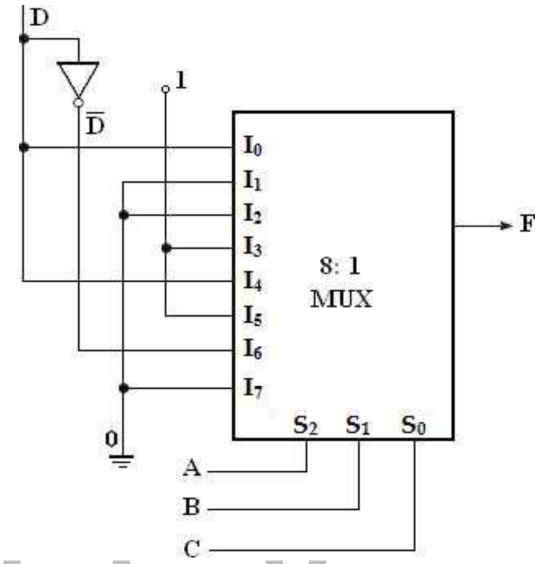
www.binils.com



Multiplexer Implementation:

An 8×1 multiplexer has inputs A, B and C connected to the selection inputs S_2, S_1, and S_0 respectively. The data inputs I_0 to I_7 are as follows **I_1=I_2=I_7= 0; I_3=I_5= 1; I_0=I_4= D** and **I_6= D'.**

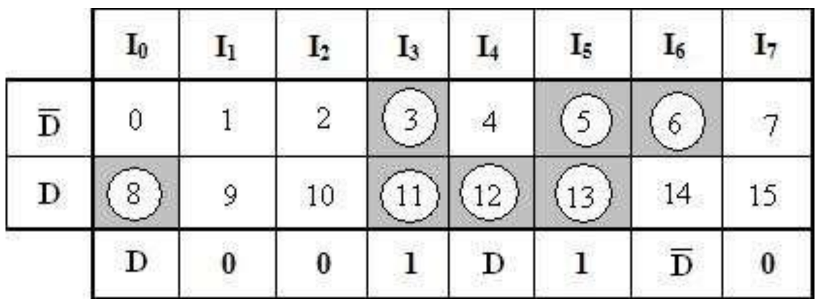
Determine the Boolean function that the multiplexer implements.



www.binils.com

Multiplexer Implementation:

Implementation table:



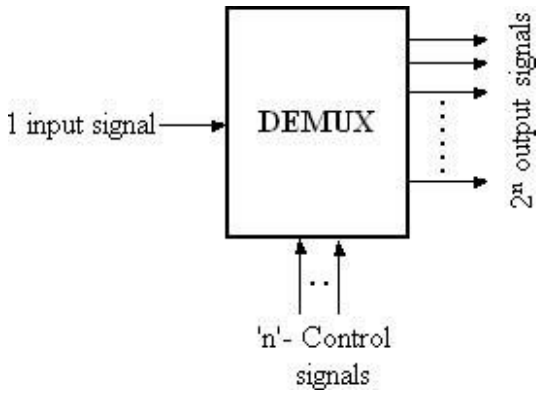
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	0	1	2	③	4	⑤	⑥	7
D	⑧	9	10	⑪	⑫	⑬	14	15
	D	0	0	1	D	1	$\overline{D}$	0

F (A, B, C, D) = ∑m (3, 5, 6, 8, 11, 12, 13).

DEMULTIPLEXER:

Demultiplex means one into many. Demultiplexing is the process of taking information from one input and transmitting the same over one of several outputs.

A demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several (2n) output lines.



www.binils.com

Fig : 2.34 - **Block diagram of demultiplexer**

The block diagram of a demultiplexer which is opposite to a multiplexer in its operation is shown above. The circuit has one input signal, ‗n' select signals and 2^n output signals. The select inputs determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input caused to appear on one of the n output lines, the demultiplexer is also called a ―***data distributer***‖ or a ―***serial-to- parallel converter***‖ .

1-to-4 Demultiplexer:

A 1-to-4 demultiplexer has a single input, **D_{in}**, four outputs (**Y_0 to Y_3**) and two select inputs (**S_1 and S_0**).

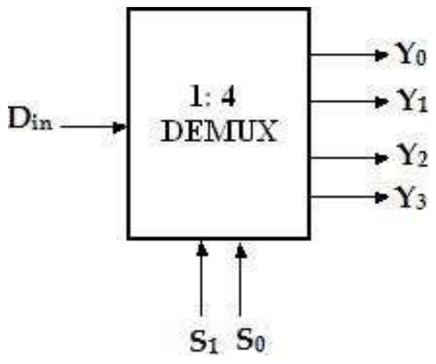


Fig : 2.35 - Logic Symbol

www.binils.com

The input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines. The truth table of the 1-to-4 demultiplexer is shown below.

Enable	S1	S0	Din	Y0	Y1	Y2	Y3
0	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0

1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Truth table of 1-to-4 demultiplexer

From the truth table, it is clear that the data input, D_{in} is connected to the output Y_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 and S_0= 1. Similarly, the data input is connected to output Y_2 and Y_3 when S_1= 1 and S_0= 0 and when S_1= 1 and S_0= 1, respectively. Also, from the truth table, the expression for outputs can be written as follows,

www.binils.com

Y0=

S1’S0’Din

Y1=

S1’S0Din

Y2=

S1S0’Din

Y3=

S1S0Din

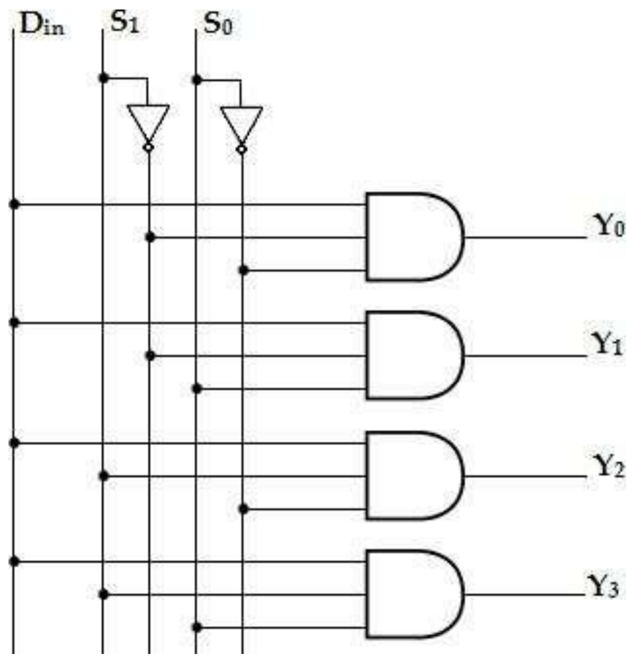


Fig : 2.36 - Logic diagram of 1-to-4 demultiplexer

www.binils.com

Now, using the above expressions, a 1-to-4 demultiplexer can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line D_{in}, is connected to all the AND gates. The two select lines S_1, S_0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

1-to-8 Demultiplexer:

A 1-to-8 demultiplexer has a single input, **D_{in}**, eight outputs (**Y_0 to Y_7**) and three select inputs (**S_2, S_1 and S_0**). It distributes one input line to eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

Di n	S 2	S1	S 0	Y 7	Y 6	Y 5	Y 4	Y 3	Y2	Y 1	Y 0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Truth table of 1-to-8 demultiplexer

www.binils.com

From the above truth table, it is clear that the data input is connected with one of the eight outputs based on the select inputs. Now from this truth table, the expression for eight outputs can be written as follows:

Y0= S2‘S1‘S0‘Din   Y4= S2

S1‘S0‘Din Y1= S2‘S1‘S0Din

Y5= S2

S1‘S0Din Y2= S2‘S1S0‘Din Y6=

S2 S1S0‘Din Y3= S2‘S1S0Din

Y7= S2S1S0Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawn as shown below. Here, the single data line, D_{in} is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if $S_2S_1S_0$= 000, then only AND gate-0 will be enabled and thereby the data input, D_{in} will appear at Y_0. Similarly, the different combinations of the select inputs, the input D_{in} will appear at the respective output.

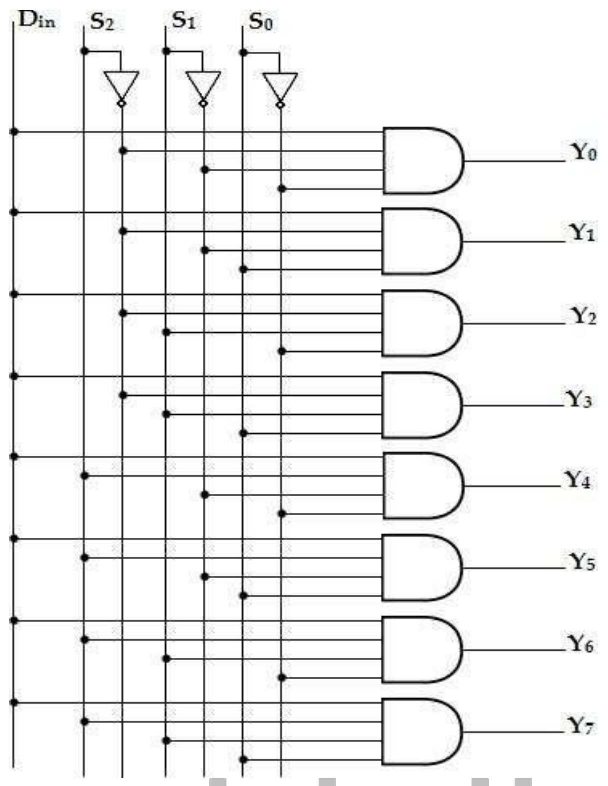
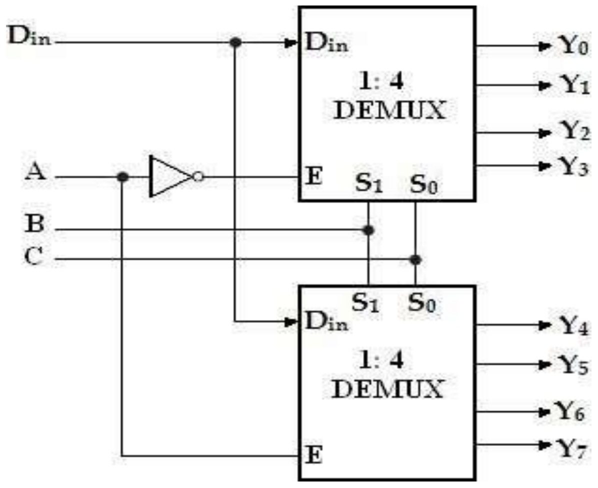


Fig : 2.37 - Logic diagram of 1-to-8 demultiplexer

www.binils.com

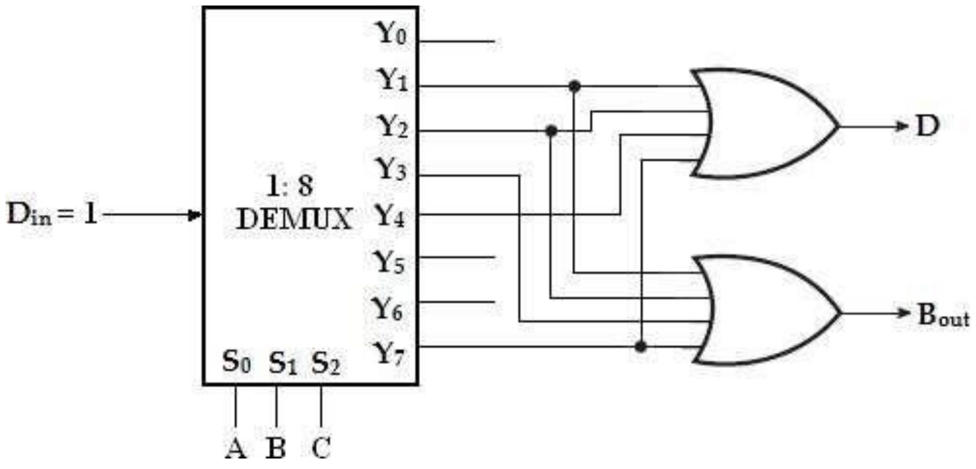
1. Design 1:8 demultiplexer using two 1:4 DEMUX.



2. Implement full subtractor using demultiplexer.

www.binils.com

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



www.binils.com

EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

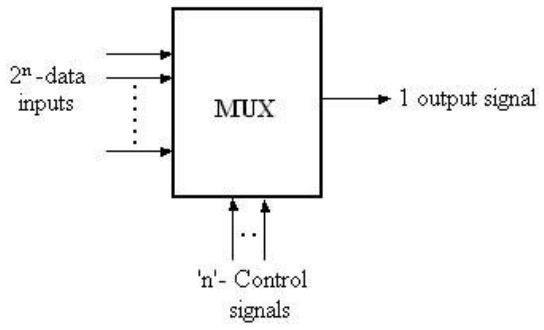
MULTIPLEXER: (Data Selector)

A ***multiplexer*** or *MUX*, is a combinational circuit with more than one input line, one output line and more than one selection line. A multiplexer selects binary information present from one of many input lines, depending upon the logic status of the selection inputs, and routes it to the output line. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The multiplexer is often labeled as MUX in block diagrams.

A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line.

www.binils.com

Fig : 2.30 - Block diagram of Multiplexer



<u>2-to-1- line Multiplexer</u>:

The circuit has two data input lines, one output line and one selection line, S. When S= 0, the upper AND gate is enabled and I_0 has a path to the output.

When S=1, the lower AND gate is enabled and I_1 has a path to the output.

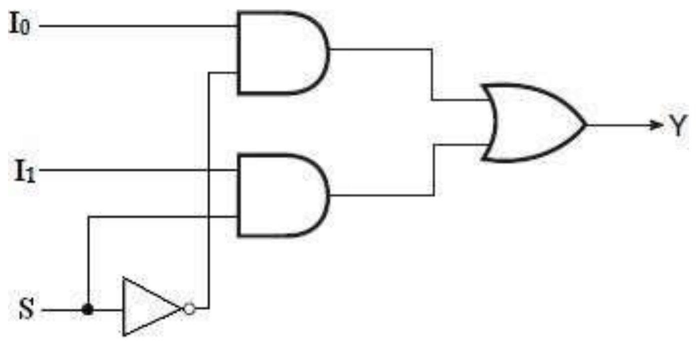


Fig : 2.31 - Logic diagram

The multiplexer acts like an electronic switch that selects one of the two sources.

Truth table:

S	Y
0	I0
1	I1

www.binils.com

4-to-1-line Multiplexer:

A 4-to-1-line multiplexer has four (2n) input lines, two (n) select lines and one output line. It is the multiplexer consisting of four input channels and information of one of the channels can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input channel is possible by two selection inputs.

Each of the four inputs I_0 through I_3, is applied to one input of AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.

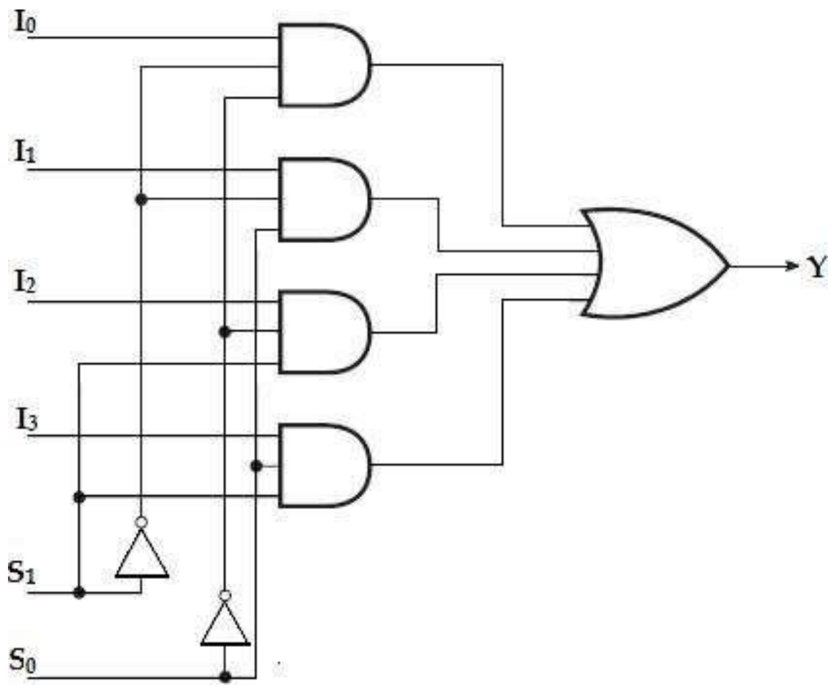


Fig : 2.32 - 4-to-1-Line Multiplexer

www.binils.com

Function table:

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

To demonstrate the circuit operation, consider the case when S_1S_0= 10. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I_0 only if S_1= 0 and S_0= 0; Y= I_0S_1‘S_0‘. The data output is equal to I_1 only if S_1= 0 and S_0= 1; Y= I_1S_1‘S_0.

The data output is equal to I_2 only if S_1= 1 and S_0= 0; Y= $I_2S_1S_0$‘.

The data output is equal to I_3 only if S_1= 1 and S_0= 1; Y= $I_3S_1S_0$.
When these terms are ORed, the total expression for the data output is,

Y= I0S1’S0’+ I1S1’S0 +I2S1S0’+ I3S1S0.

As in decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer.

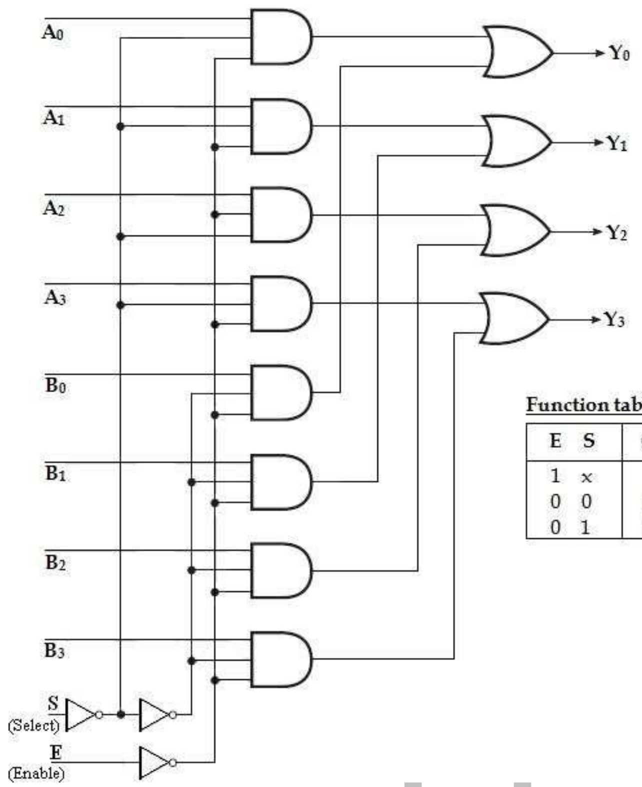
www.binils.com

Quadruple 2-to-1 Line Multiplexer:

This circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either A0 or B0. Similarly, output Y1 may have the value of A1 or B1, and so on. Input selection line, S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal operation.

Although the circuit contains four 2-to-1-Line multiplexers, it is viewed as a circuit that selects one of two 4-bit sets of data lines. The unit is enabled when E= 0. Then if S= 0, the four A inputs have a path to the four outputs. On the other hand, if

S=1, the four B inputs are applied to the outputs. The outputs have all 0‘s when E= 1, regardless of the value of S.



Function table:

E	S	Output Y
1	x	All 0's
0	0	Select A
0	1	Select B

Fig : 2.33 – Quadruple 2 to 1 MUX

www.binils.com

Application:

The multiplexer is a very useful MSI function and has various ranges of applications in data communication. Signal routing and data communication are the important applications of a multiplexer. It is used for connecting two or more sources to guide to a single destination among computer units and it is useful for constructing a common bus system. One of the general properties of a multiplexer is that Boolean functions can be implemented by this device.

Implementation of Boolean Function using MUX:

Any Boolean or logical expression can be easily implemented using a multiplexer. If a Boolean expression has (n+1) variables, then ‗n' of these variables can be connected to the select lines of the multiplexer. The remaining single variable along with constants 1 and 0 is used as the input of the multiplexer. For example, if C is the single variable, then the inputs of the multiplexers are C, C', 1 and 0. By this method any logical expression can be implemented.

In general, a Boolean expression of (n+1) variables can be implemented using a multiplexer with 2^n inputs.

www.binils.com

1. **Implement the following boolean function using 4: 1 multiplexer,**
   F (A, B, C) = ∑m (1, 3, 5, 6).

   Solution: Variables, n= 3 (A, B, C)

   Select lines= n- 1 = 2 (**S_1, S_0**) 2^{n-1} to MUX i.e., 2^2 to 1 = 4 to 1 MUX

   Input lines= $2^{n-1} = 2^2 = 4$ (**D_0, D_1, D_2, D_3**)

Implementation table:

Apply variables A and B to the select lines. The procedures for implementing the function are:

i. List the input of the multiplexer

ii. List under them all the minterms in two rows as shown below.

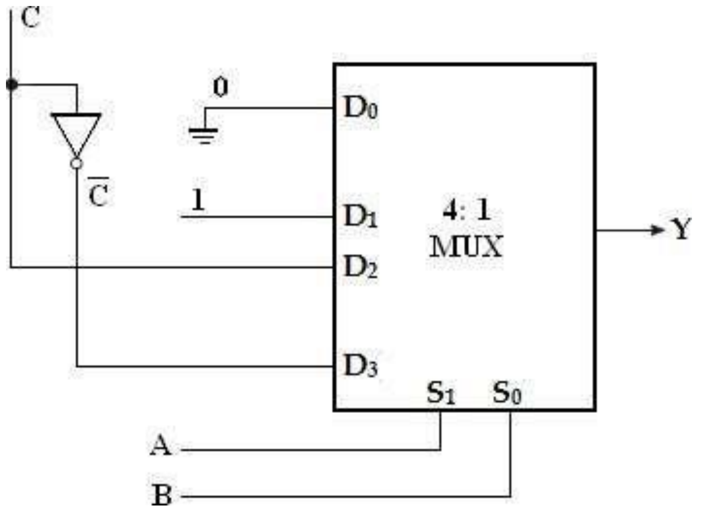
The first half of the minterms is associated with A' and the second half with A. The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer.

1. If both the minterms in the column are not circled, apply 0 to the corresponding input.
2. If both the minterms in the column are circled, apply 1 to the corresponding input.
3. If the bottom minterm is circled and the top is not circled, apply C to the input. 4.If the top minterm is circled and the bottom is not circled, apply C' to the input.

	D_0	D_1	D_2	D_3
$\overline{C}$	0	①	2	③
C	4	⑤	⑥	7
	0	1	C	$\overline{C}$

www.binils.com

Multiplexer Implementation:

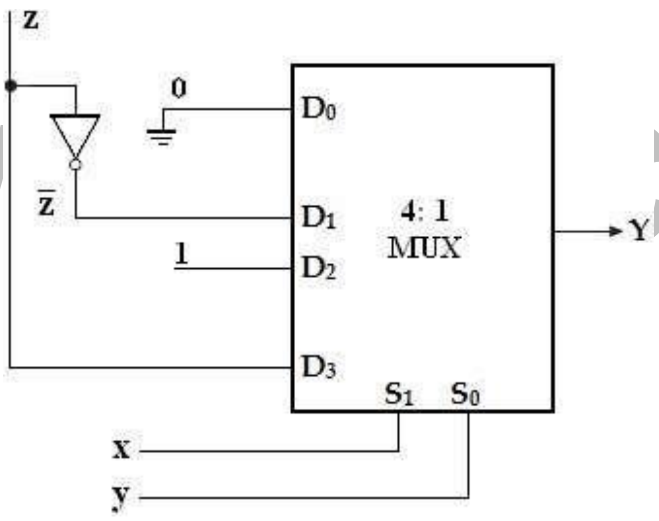


2. F (x, y, z) = ∑m (1, 2, 6, 7) Solution:

Implementation table:

	D_0	D_1	D_2	D_3
$\overline{z}$	0	1	2	3
z	4	5	6	7
	0	$\overline{z}$	1	z

Multiplexer Implementation:



www.binils.com

3. F (A, B, C) = ∑m (1, 2, 4, 5)

Solution: Variables, n= 3 (A, B, C)

Select lines= n- 1 = 2 (**S_1, S_0**) 2^{n-1} to

MUX i.e., 2^2 to 1 = 4

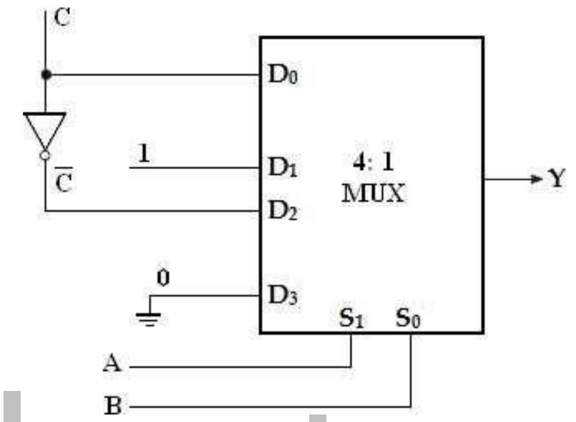
to 1 MUX

Input lines= 2^{n-1} = 2^2 = 4 (**D_0, D_1, D_2, D_3**) **Implementation**

table:

	D_0	D_1	D_2	D_3
$\overline{C}$	0	1	2	3
C	4	5	6	7
	C	1	$\overline{C}$	0

Multiplexer Implementation:



www.binils.com

4. F(P, Q, R, S)= ∑m (0, 1, 3, 4, 8, 9, 15)

Solution:

Variables, n= 4 (P, Q, R, S) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

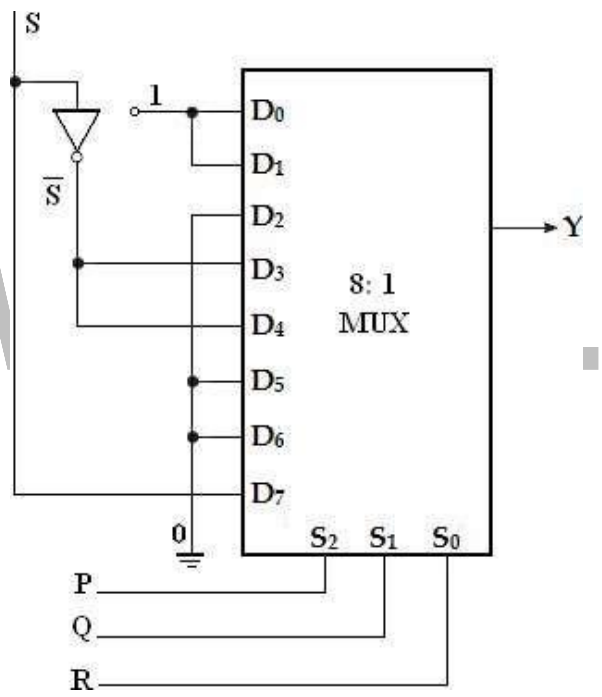
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	0	1	2	3	4	5	6	7
S	8	9	10	11	12	13	14	15
	1	1	0	$\overline{S}$	$\overline{S}$	0	0	S

Multiplexer Implementation:



www.binils.com

5. **Implement the Boolean function using 8: 1 and also using 4:1 multiplexer F (A, B, C, D) = ∑m (0, 1, 2, 4, 6, 9, 12, 14)**

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

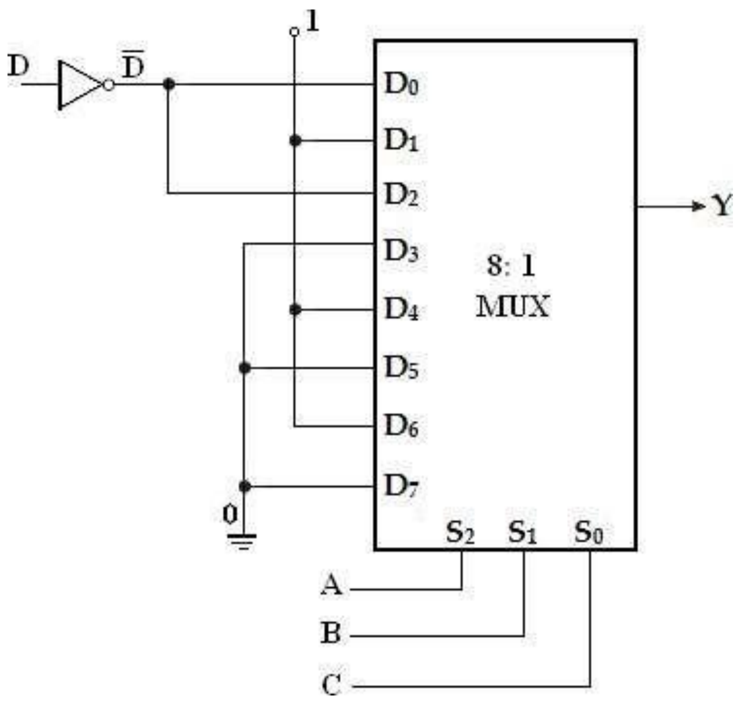
Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

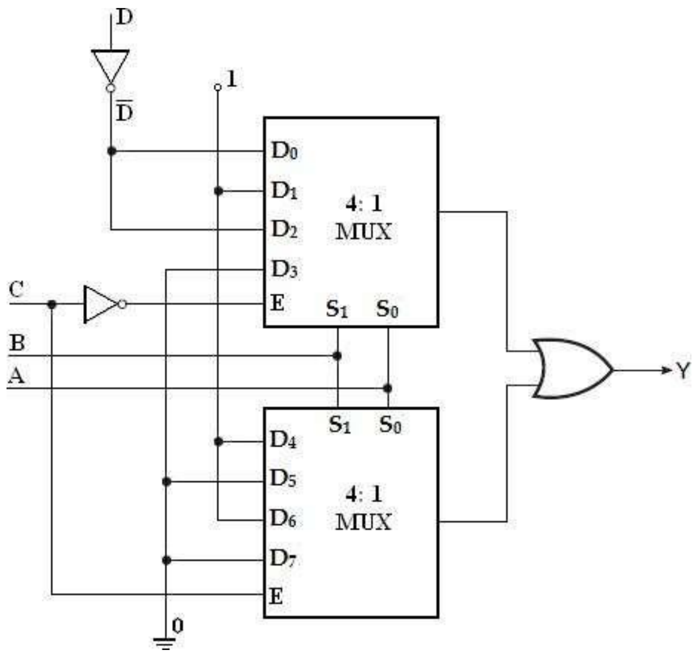
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	$\textcircled{0}$	$\textcircled{1}$	$\textcircled{2}$	3	$\textcircled{4}$	5	$\textcircled{6}$	7
D	8	$\textcircled{9}$	10	11	$\textcircled{12}$	13	$\textcircled{14}$	15
	$\overline{D}$	**1**	$\overline{D}$	**0**	**1**	**0**	**1**	**0**

www.binils.com

Multiplexer Implementation (Using 8: 1 MUX):



Using 4: 1 MUX:



6. F (A, B, C, D) = ∑m (1, 3, 4, 11, 12, 13, 14, 15)

www.binils.com

Solution:

Variables, n= 4 (A, B, C, D) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

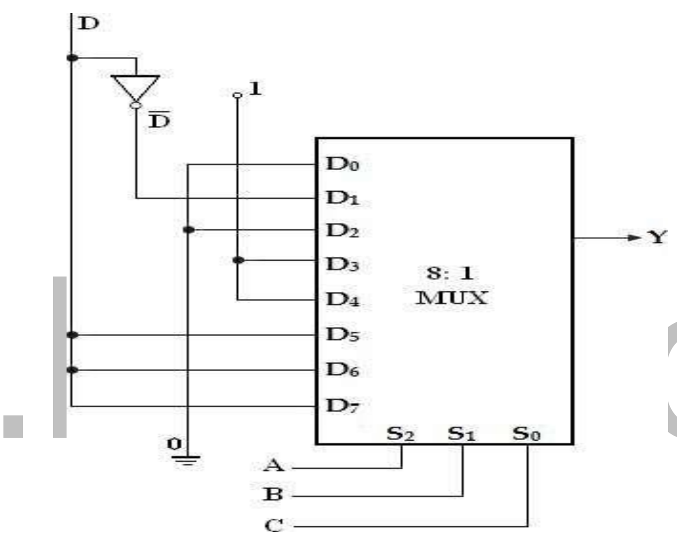
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	①	2	③	④	5	6	7
D	8	9	10	⑪	⑫	⑬	⑭	⑮
	0	$\overline{D}$	**0**	**1**	**1**	D	D	D

Multiplexer Implementation:



www.binils.com

7. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = A'BD' + ACD + B'CD + A'C'D.

Solution

Convert into standard SOP form,

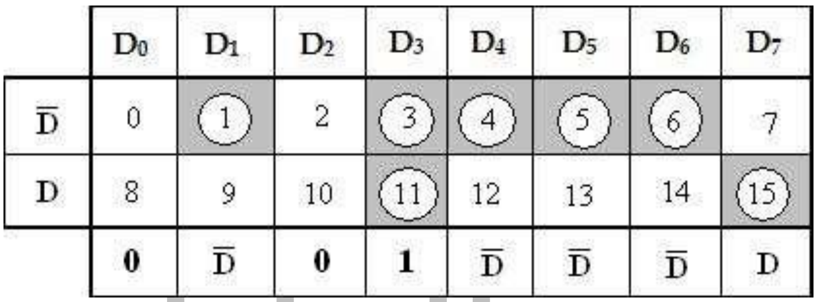
= A‘BD‘ (C‘+C) + ACD (B‘+B) + B‘CD (A‘+A) + A‘C‘D (B‘+B)

= A‘BC‘D‘ + A‘BCD‘+ AB‘CD+ ABCD +A‘B‘CD + AB‘CD+A‘B‘C‘D+ A‘BC‘D

= A‘BC‘D‘ + A‘BCD‘+ AB‘CD + ABCD +A‘B‘CD +A‘B‘C‘D+ A‘BC‘D

= m4+ m6+ m11+ m15+ m3+ m1+ m5
= ∑m (1, 3, 4, 5, 6, 11, 15)

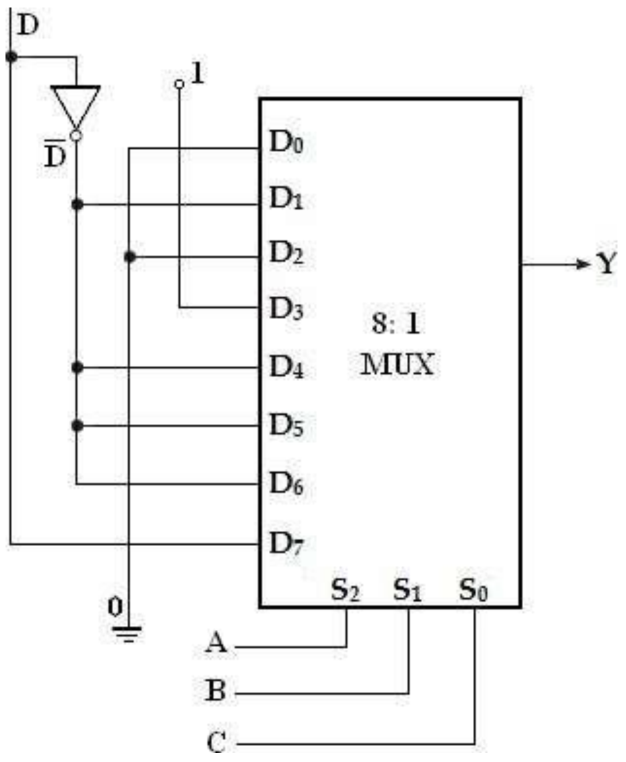
Implementation table:



	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	①	2	③	④	⑤	⑥	7
D	8	9	10	⑪	12	13	14	⑮
	0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$	$\overline{D}$	D

www.binils.com

Multiplexer Implementation:



www.binils.com

8. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = AB'D + A'C'D + B'CD' + AC'D.

Solution:

Convert into standard SOP form,

= AB'D (C'+C) + A'C'D (B'+B) + B'CD' (A'+A) + AC'D (B'+B)

= AB'C'D+ AB'CD+ A'B'C'D + A'BC'D +A'B'CD' + AB'CD' +AB'C'D+ ABC'D= AB'C'D

+ AB'CD+ A'B'C'D + A'BC'D +A'B'CD' + AB'CD'+ ABC'D = m9+ m11+

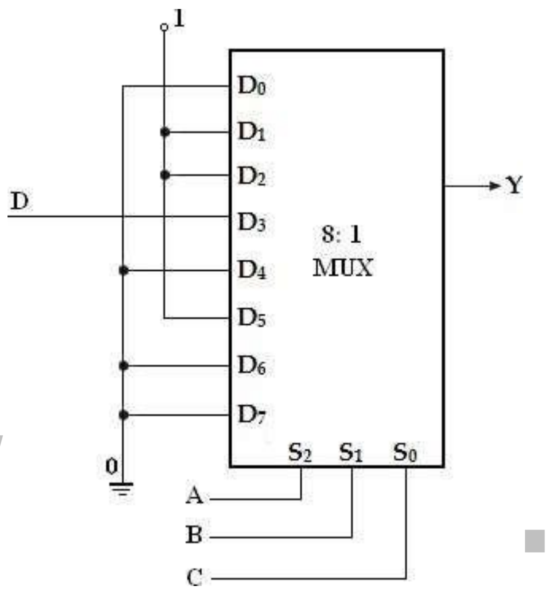
m1+ m5+ m2+ m10+ m13

= ∑m (1, 2, 5, 9, 10, 11, 13).

Implementation Table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	0	**1**	**1**	D	**0**	**1**	**0**	**0**

Multiplexer Implementation:



www .com

9. Implement the Boolean function using 8: 1 and also using 4:1 multiplexer **F (w, x, y, z) = ∑m (1, 2, 3, 6, 7, 8, 11, 12, 14)**

Solution:

Variables, n= 4 (w, x, y, z)

Select lines= n-1 = 3 (**S_2,**

S_1, S_0)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

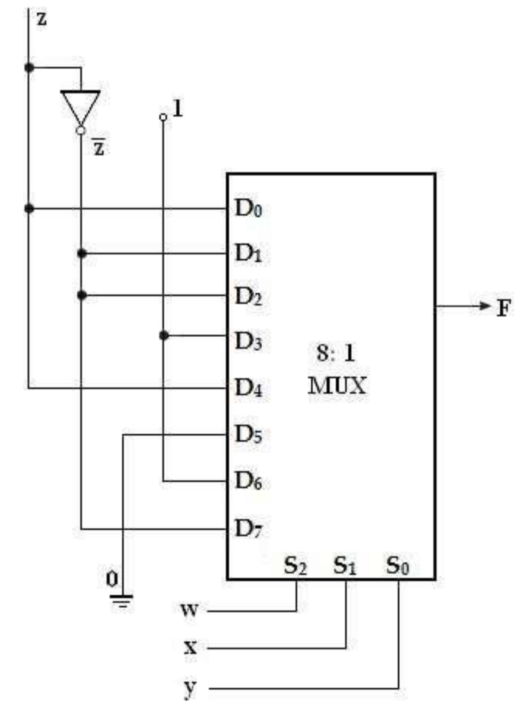
Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

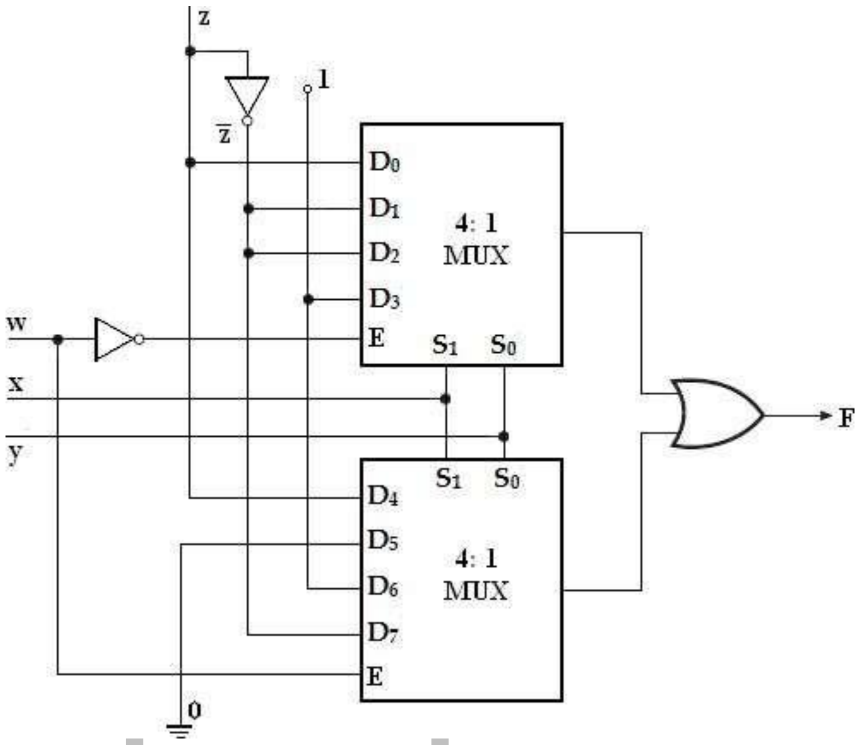
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{z}$	0	$\textcircled{1}$	$\textcircled{2}$	$\textcircled{3}$	4	5	$\textcircled{6}$	$\textcircled{7}$
z	$\textcircled{8}$	9	10	$\textcircled{11}$	$\textcircled{12}$	13	$\textcircled{14}$	15
	z	$\bar{z}$	$\bar{z}$	**1**	z	**0**	**1**	$\bar{z}$

Multiplexer Implementation (Using 8:1 MUX):

www.binils.com



(Using 4:1 MUX):



www.binils.com

10. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∏m (0, 3, 5, 8, 9, 10, 12, 14) **Solution:**

Variables, n= 4 (A, B, C, D) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

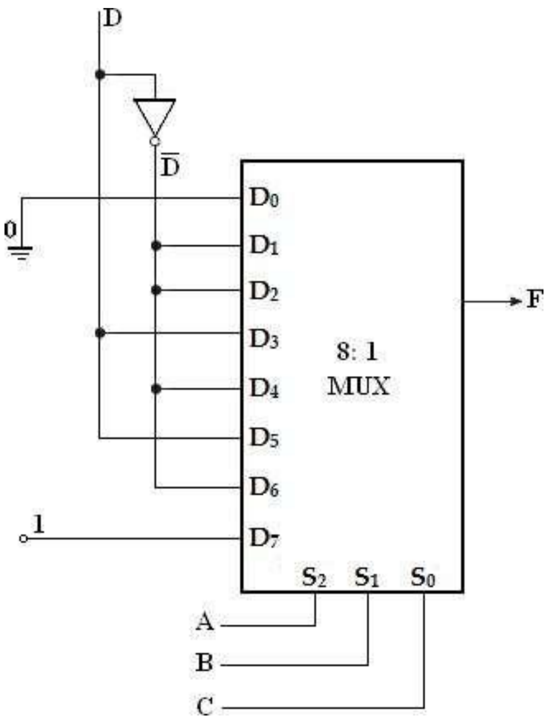
Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	0	$\overline{D}$	$\overline{D}$	D	$\overline{D}$	D	$\overline{D}$	1

Multiplexer Implementation:

www.binils.com



11. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∑m (0, 2, 6, 10, 11, 12, 13) + d (3, 8, 14)

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= 2^{n-1} = 2^3 = 8 (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

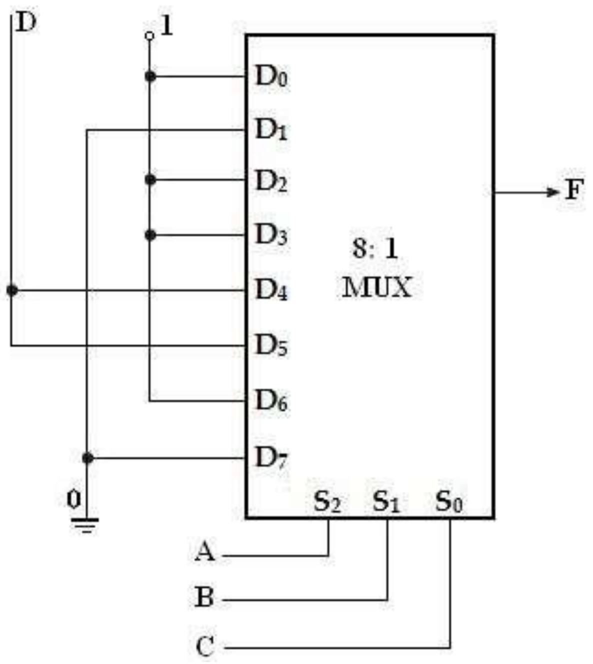
Implementation

Table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	1	0	1	1	D	D	1	0

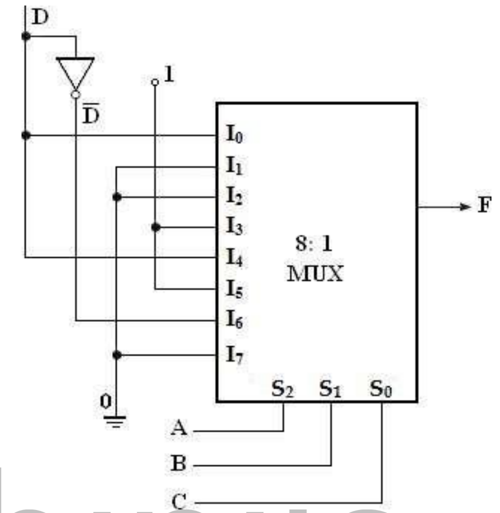
www.binils.com

Multiplexer Implementation:



12. An 8×1 multiplexer has inputs A, B and C connected to the selection inputs S_2, S_1, and S_0 respectively. The data inputs I_0 to I_7 are as follows $\mathbf{I_1=I_2=I_7=0;}$ $\mathbf{I_3=I_5=1;\ I_0=I_4=D}$ and $\mathbf{I_6=D'}$.
Determine the Boolean function that the multiplexer implements.

Multiplexer Implementation:



www.binils.com

Implementation table:

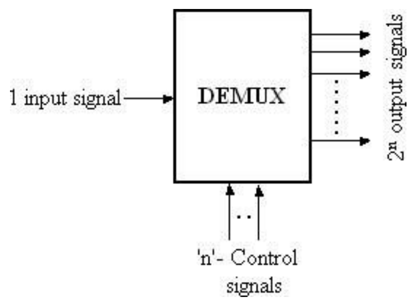
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	0	1	2	③	4	⑤	⑥	7
D	⑧	9	10	⑪	⑫	⑬	14	15
	D	0	0	1	D	1	$\overline{D}$	0

F (A, B, C, D) = ∑m (3, 5, 6, 8, 11, 12, 13).

DEMULTIPLEXER:

Demultiplex means one into many. Demultiplexing is the process of taking information from one input and transmitting the same over one of several outputs.

A demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several (2n) output lines.



www.binils.com

Fig : 2.34 - **Block diagram of demultiplexer**

The block diagram of a demultiplexer which is opposite to a multiplexer in its operation is shown above. The circuit has one input signal, ‗n' select signals and 2^n output signals. The select inputs determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input caused to appear on one of the n output lines, the demultiplexer is also called a ―***data distributer***‖ or a ―***serial-to- parallel converter***‖ .

1-to-4 Demultiplexer:

A 1-to-4 demultiplexer has a single input, **D_{in}**, four outputs (**Y_0 to Y_3**) and two select inputs (**S_1 and S_0**).

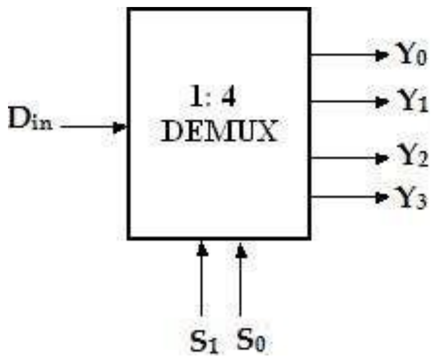


Fig : 2.35 - Logic Symbol

www.binils.com

The input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines. The truth table of the 1-to-4 demultiplexer is shown below.

Enable	**S1**	**S0**	**Din**	**Y0**	**Y1**	**Y2**	**Y3**
0	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0

1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Truth table of 1-to-4 demultiplexer

From the truth table, it is clear that the data input, D_{in} is connected to the output Y_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 and S_0= 1. Similarly, the data input is connected to output Y_2 and Y_3 when S_1= 1 and S_0= 0 and when S_1= 1 and S_0= 1, respectively. Also, from the truth table, the expression for outputs can be written as follows,

www.binils.com

Y0=

S1'S0'Din

Y1=

S1'S0Din

Y2=

S1S0'Din

Y3=

S1S0Din

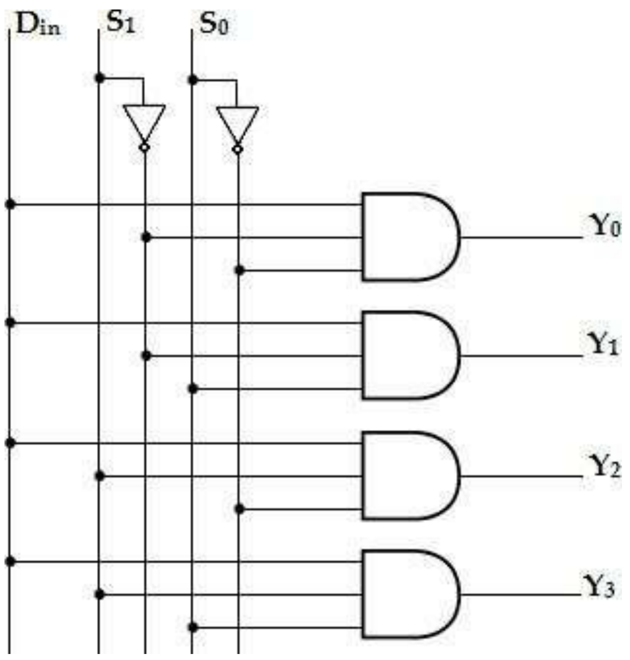


Fig : 2.36 - Logic diagram of 1-to-4 demultiplexer

www.binils.com

Now, using the above expressions, a 1-to-4 demultiplexer can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line D_{in}, is connected to all the AND gates. The two select lines S_1, S_0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

1-to-8 Demultiplexer:

A 1-to-8 demultiplexer has a single input, **D_{in}**, eight outputs (**Y_0 to Y_7**) and three select inputs (**S_2, S_1 and S_0**). It distributes one input line to eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

Di n	S 2	S1	S 0	Y 7	Y 6	Y 5	Y 4	Y 3	Y2	Y 1	Y 0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Truth table of 1-to-8 demultiplexer

www.binils.com

From the above truth table, it is clear that the data input is connected with one of the eight outputs based on the select inputs. Now from this truth table, the expression for eight outputs can be written as follows:

Y0= S2‘S1‘S0‘Din   Y4= S2

S1‘S0‘Din Y1= S2‘S1‘S0Din

Y5= S2

S1‘S0Din Y2= S2‘S1S0‘Din Y6=

S2 S1S0‘Din Y3= S2‘S1S0Din

Y7= S2S1S0Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawn as shown below. Here, the single data line, D_{in} is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if $S_2S_1S_0$= 000, then only AND gate-0 will be enabled and thereby the data input, D_{in} will appear at Y_0. Similarly, the different combinations of the select inputs, the input D_{in} will appear at the respective output.

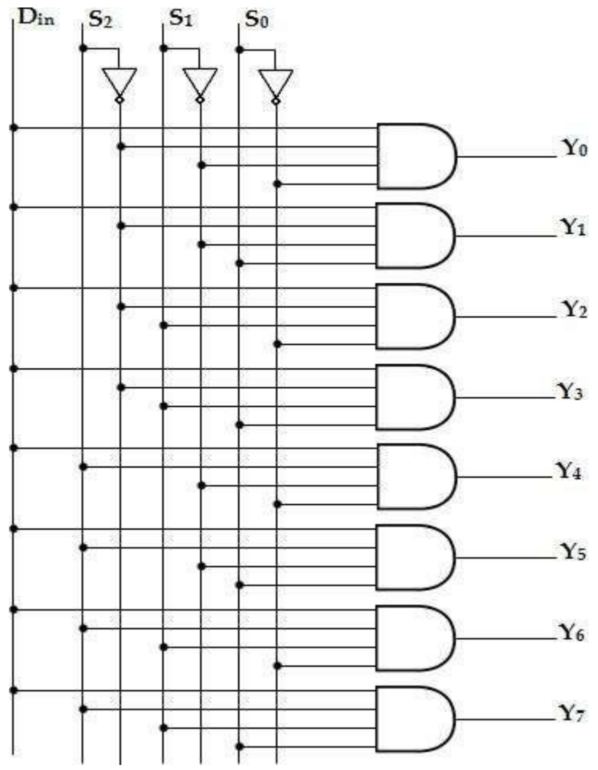
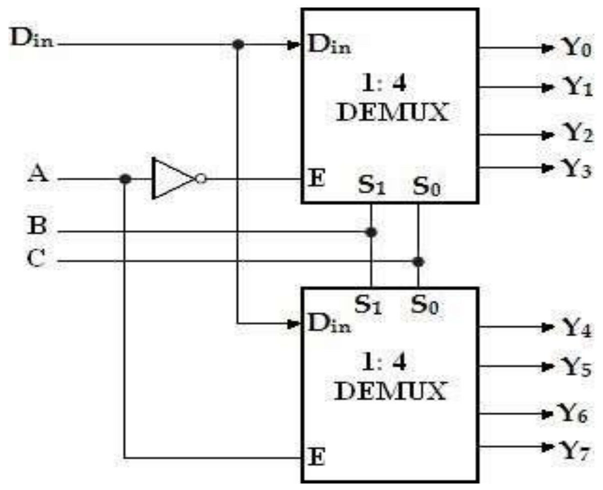


Fig : 2.37 - Logic diagram of 1-to-8 demultiplexer

www.binils.com

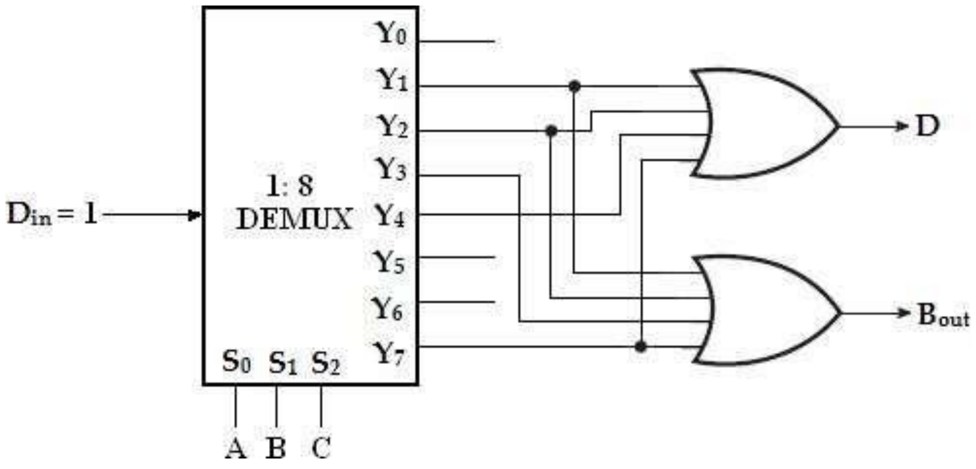
1. Design 1:8 demultiplexer using two 1:4 DEMUX.



2. Implement full subtractor using demultiplexer.

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

www.binils.com



www.binils.com