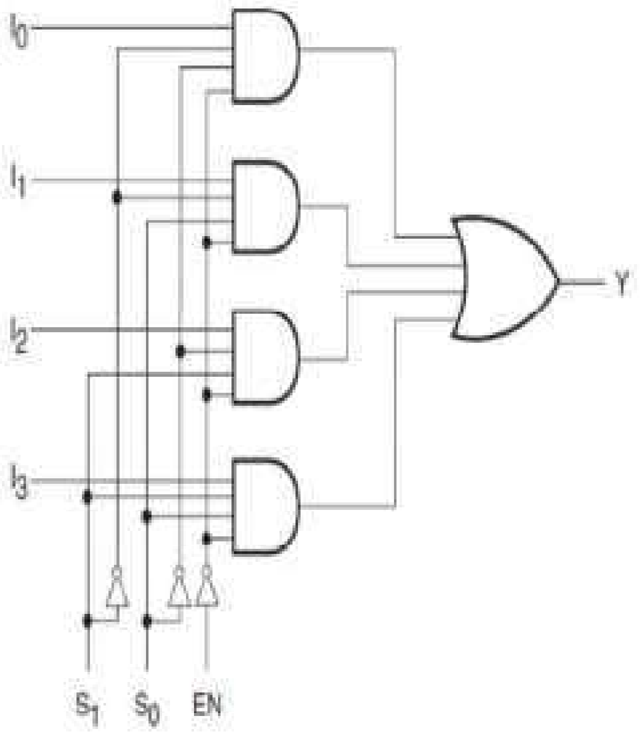
Q12a)Implementing Boolean Functions with Multiplexers
(or)
b)Implement the following Boolean function using 8:1 Multiplexer

One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2^n-to-1 MUX to implement an n-variable Boolean function. The input lines corresponding to each of the minterms present in the Boolean function are made equal to logic '1' state. The remaining minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'. As an example, Fig. (a) shows the use of an 8-to-1 MUX for implementing the Boolean function given by the equation

$$f(A, B, C) = \sum 2, 4, 7$$



S_1	S_0	EN	Y
X	X	1	0
0	0	0	I_0
0	1	0	I_1
1	0	0	I_2
1	1	0	I_3

Figure 4-to-1 multiplexer with an ENABLE input.

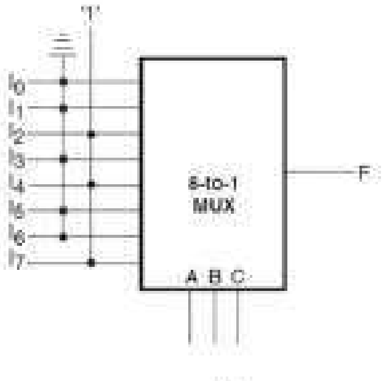
As shown in Fig. the input lines corresponding to the three minterms present in the given Boolean function are tied to logic '1'. The remaining five possible minterms absent in the Boolean function are tied to logic '0'.

However, there is a better technique available for doing the same. In this, a 2^n-to-1 MUX can be used to implement a Boolean function with $n + 1$ variables. The procedure is as follows. Out of $n + 1$ variables, n are connected to the n selection lines of the 2^n-to-1 multiplexer. The left-over variable is used with the input lines. Various input lines are tied to one of the following: '0', '1', the left-over variable and the complement of the left-over variable. Which line is given what logic status can be easily determined with the help of a simple procedure.

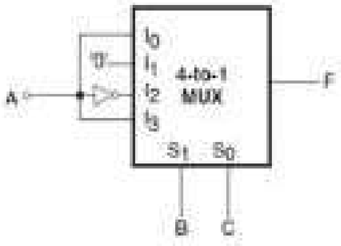
It is a three-variable Boolean function. Conventionally, we will need to use an 8-to-1 multiplexer to implement this function. We will now see how this can be implemented with a 4-to-1 multiplexer. The chosen multiplexer has two selection lines.

In the next step, two of the three variables are connected to the two selection lines, with the higher-order variable connected to the higher-order selection line. For instance, in the present case, variables B and C are the chosen variables for the selection lines and are respectively connected to selection lines S_1 and S_0. In the third step, a table of the type shown in Table is constructed. Under the inputs to the multiplexer, minterms are listed in two rows, as shown. The first row lists those terms where remaining variable A is complemented, and second row lists those terms where A is uncomplemented. This is easily done with the help of the truth table.

The required minterms are identified or marked in some manner in this table. In the given table, these entries have been highlighted. Each column is inspected individually. If neither minterm of a certain column is highlighted, a '0' is written below that. If both are highlighted, a '1' is



(a)



(b)

Figure Hardware implementation of the Boolean function given by equation

Truthtable

Minterm	A	B	C	$f(A,B,C)$
0	0	0	0	0
1	0	0	1	0
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

written. If only one is highlighted, the corresponding variable (complemented or uncomplemented) is written. The input lines are then given appropriate logic status. In the present case, I_0, I_1, I_2 and I_3 would be connected to A, 0, $\overline{A}$ and A respectively. Figure (b) shows the logic implementation.

Table Implementation table for multiplexers.

	I_0	I_1	I_2	I_3
$\overline{A}$	0	1	2	3
A	4	5	6	7
	A	0	$\overline{A}$	A

Table Implementation table for multiplexers.

	I_0	I_1	I_2	I_3	I_3
$\overline{A}$	0	1	2	3	6
A	4	5	6	7	7
	A	0	$\overline{A}$	A	C

It is not necessary to choose only the leftmost variable in the sequence to be used as input to the multiplexer. Any of the variables can be used provided the implementation table is constructed accordingly. In the problem illustrated above, A was chosen as the variable for the input lines, and accordingly the first row of the implementation table contained those entries where 'A' was complemented and the second row contained those entries where A was uncomplemented. If we consider C as the left-out variable, the implementation table will be as shown in Table

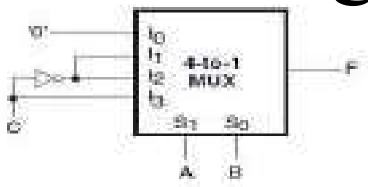


Figure :Hardware implementation using a 4-to-1 multiplexer.

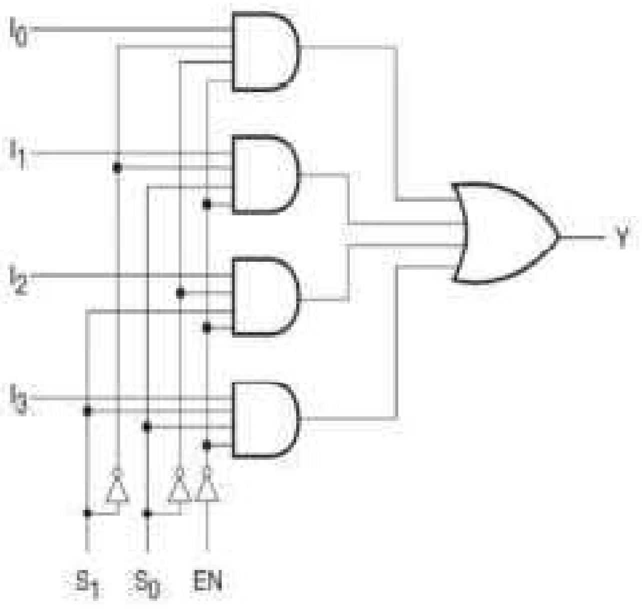
Table Implementation table for multiplexers.

	I_0	I_1	I_2	I_3
$\overline{B}$	0	1	4	5
B	2	3	6	7
	B	0	$\overline{B}$	B

Q13a)Implementing Boolean Functions with Multiplexers
(or)
b)Implement the following Boolean function using 8:1 Multiplexer

One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2^n-to-1 MUX to implement an n-variable Boolean function. The input lines corresponding to each of the minterms present in the Boolean function are made equal to logic '1' state. The remaining minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'. As an example, Fig. (a) shows the use of an 8-to-1 MUX for implementing the Boolean function given by the equation

$$f(A, B, C) = \sum 2, 4, 7$$



S_1	S_0	EN	Y
X	X	1	0
0	0	0	I_0
0	1	0	I_1
1	0	0	I_2
1	1	0	I_3

Table Implementation table.

	I_0	I_1	I_2	I_3
$\overline{C}$	0	1	2	3
C	4	5	6	7
	1	0	C	1

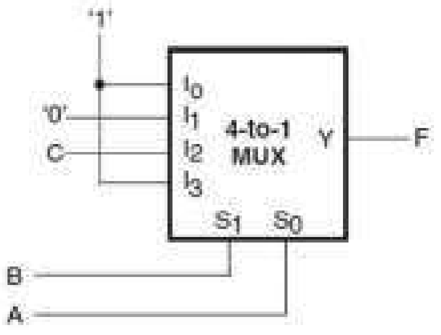
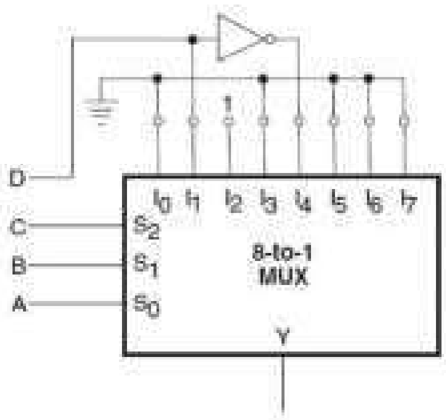


Figure Example



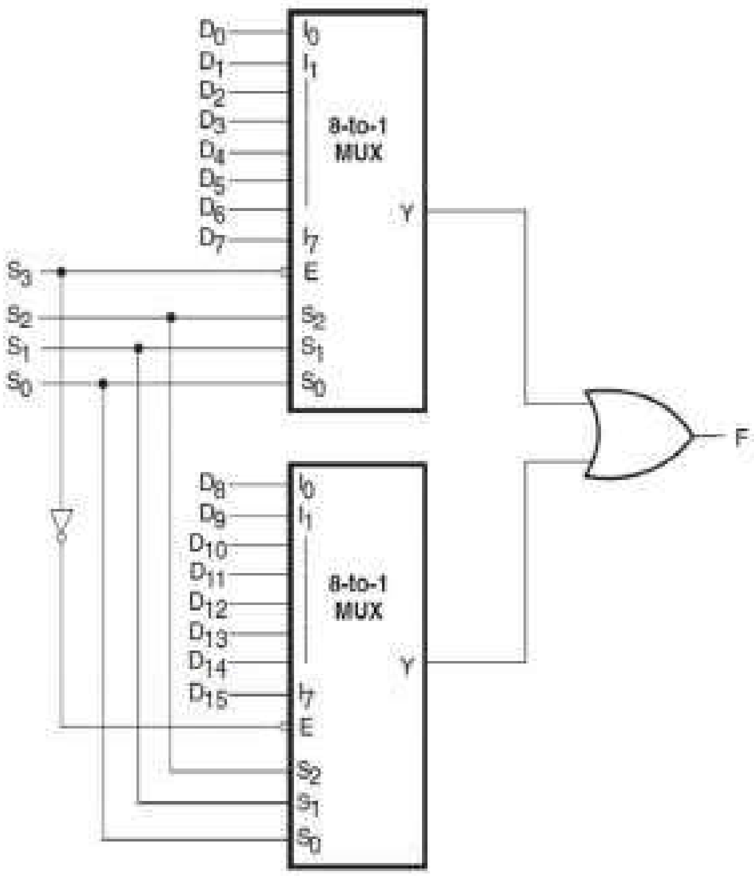
Encoders

Q14a)Write a note on encoders

(or)

b)Design of encoder circuit

An *encoder* is a multiplexer without its single output line. It is a combinational logic function that has 2^n (or fewer) input lines and n output lines, which correspond to n selection lines in a multiplexer. The n output lines generate the binary code for the possible 2^n input lines. Let us take the case of an octal-to-binary encoder. Such an encoder would have eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent. The truth table of such an encoder is given in Table In the truth table, D_0 to D_7 represent octal digits 0 to 7. A, B and C represent the binary digits.



The eight input lines would have $2^8 = 256$ possible combinations. However, in the case of an octal-to-binary encoder, only eight of these 256 combinations would have any meaning. The remaining combinations of input variables are 'don't care' input combinations. Also, only one of the input lines at a time is in logic '1' state. Figure shows the hardware implementation of the octal-to-binary encoder described by the truth table in Table . This circuit has the shortcoming that it produces an all 0s output sequence when all input lines are in logic '0' state. This can be overcome by having an additional line to indicate an all 0s input sequence.

Priority Encoder:

Q15a)**Design of** ***Priority Encoder***

(or)

b)Explain the decimal to BCDPriority Encoder

A *priority encoder* is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded. We will illustrate the concept of priority encoding with the help of an example. Let us assume that the octal-to-binary encoder described in the previous paragraph has an input priority for higher-order digits. Let us also assume that input lines D_2, D_4 and D_7 are all simultaneously in logic '1' state. In that case, only D_7 will be encoded and the output will be 111. The truth table of such a priority

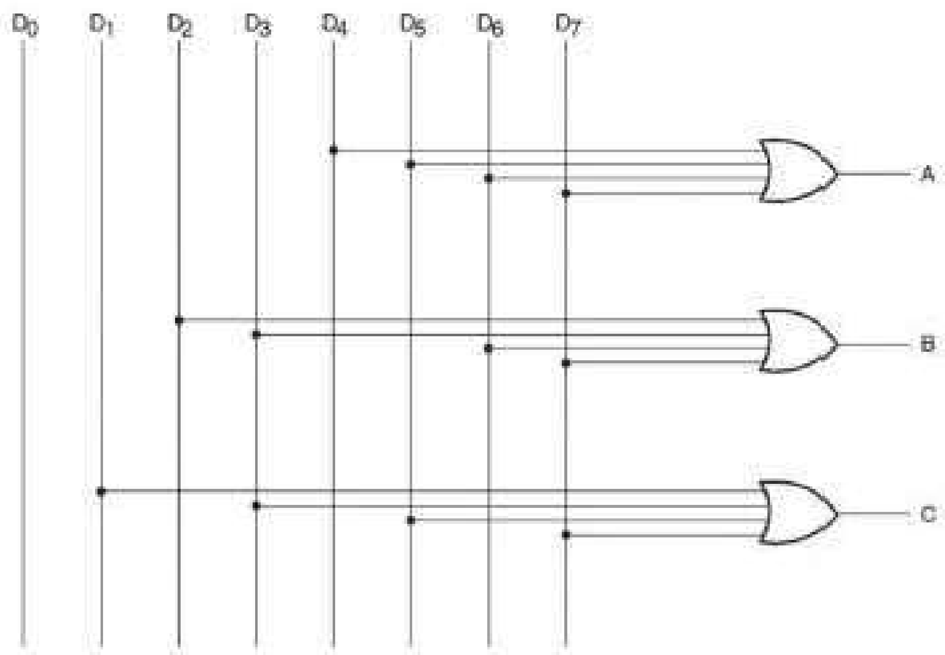


Figure Octal-to-binary encoder.

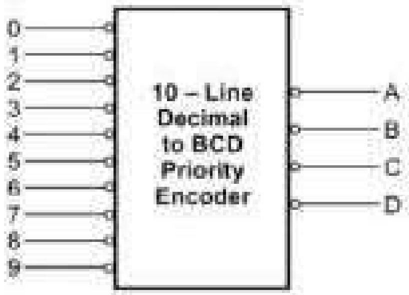
Table Truth table of an encoder.

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	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

encoder will then be modified to what is shown in Table Looking at the last row of the table, it implies that, if $D_7 = 1$, then, irrespective of the logic status of other inputs, the output is 111 as D_7 will only be encoded. As another example, Fig. shows the logic symbol and truth table of a 10-line decimal to four-line BCD encoder providing priority encoding for higher-order digits, with digit 9 having the highest priority. In the functional table shown, the input line with highest priority having a LOW on it is encoded irrespective of the logic status of the other input lines.

Table Priority encoder.

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1
X	X	1	0	0	0	0	0	0	1	0
X	X	X	1	0	0	0	0	0	1	1
X	X	X	X	1	0	0	0	1	0	0
X	X	X	X	X	1	0	0	1	0	1
X	X	X	X	X	X	1	0	1	1	0
X	X	X	X	X	X	X	1	1	1	1



Inputs										Outputs			
0	1	2	3	4	5	6	7	8	9	D	C	B	A
X	X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	X	0	1	1	1	1	1	1	1	1	0	0
X	X	0	1	1	1	1	1	1	1	1	1	0	1
X	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 10-line decimal to four-line BCD priority encoder.

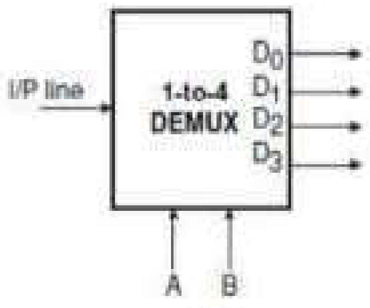
Some of the encoders available in IC form provide additional inputs and outputs to allow expansion. IC 74148, which is an eight-line to three -line priority encoder, is an example. ENABLE-IN (EI) and ENABLE-OUT (EO) terminals on this IC allow expansion. For instance, two 74148s can be cascaded to build a 16-line to four-line priority encoder.

Demultiplexers and Decoders:

***Q16a) Design of*Demultiplexers and Decoders**
(or)
b)Explain the decimal to BCDPriority Encoder

A *demultiplexer* is a combinational logic circuit with an input line, 2^n output lines and n select lines. It routes the information present on the input line to any of the output lines. The output line that gets the information present on the input line is decided by the bit status of the selection lines. A *decoder* is a special case of a demultiplexer without the input line. Figure (a) shows the circuit representation of a 1-to-4 demultiplexer. Figure (b) shows the truth table of the demultiplexer when the input line is held HIGH.

A decoder, as mentioned earlier, is a combinational circuit that decodes the information on n input lines to a maximum of 2^n unique output lines. Figure shows the circuit representation of 2-to-4, 3-to-8 and 4-to-16 line decoders. If there are some unused or 'don't care' combinations in the n-bit code, then there will be fewer than 2^n output lines. As an illustration, if there are three input lines, it



(a)

I/P	Select		O/P			
	A	B	D_0	D_1	D_2	D_3
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

(b)

Figure 1-to-4 demultiplexer.

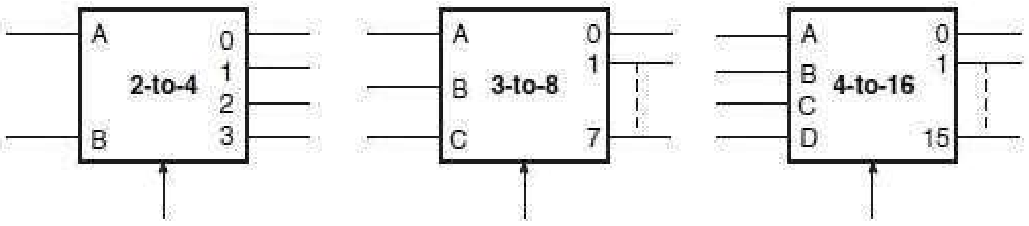


Figure Circuit representation of 2-to-4, 3-to-8 and 4-to-16 line decoders.

can have a maximum of eight unique output lines. If, in the three-bit input code, the only used three-bit combinations are 000, 001, 010, 100, 110 and 111 (011 and 101 being either unused or don't care combinations), then this decoder will have only six output lines. In general, if n and m are respectively the numbers of input and output lines, then $m \leq 2^n$.

A decoder can generate a maximum of 2^n possible minterms with an n-bit binary code. In order to illustrate further the operation of a decoder, consider the logic circuit diagram in Fig. This logic circuit, as we will see, implements a 3-to-8 line decoder function. This decoder has three inputs designated as A, B and C and eight outputs designated as D_0, D_1, D_2, D_3, D_4, D_5, D_6 and D_7. From the truth table given along with the logic diagram it is clear that, for any given input combination, only one of the eight outputs is in logic '1' state. Thus, each output produces a certain minterm that corresponds to the binary number currently present at the input. In the present case, D_0, D_1, D_2, D_3, D_4, D_5, D_6 and D_7 respectively represent the following minterms:

$$D_0 \to \overline{A}.\overline{B}.\overline{C},\ D_1 \to \overline{A}.\overline{B}.C,\ D_2 \to \overline{A}.B.\overline{C},\ D_3 \to \overline{A}.B.C$$

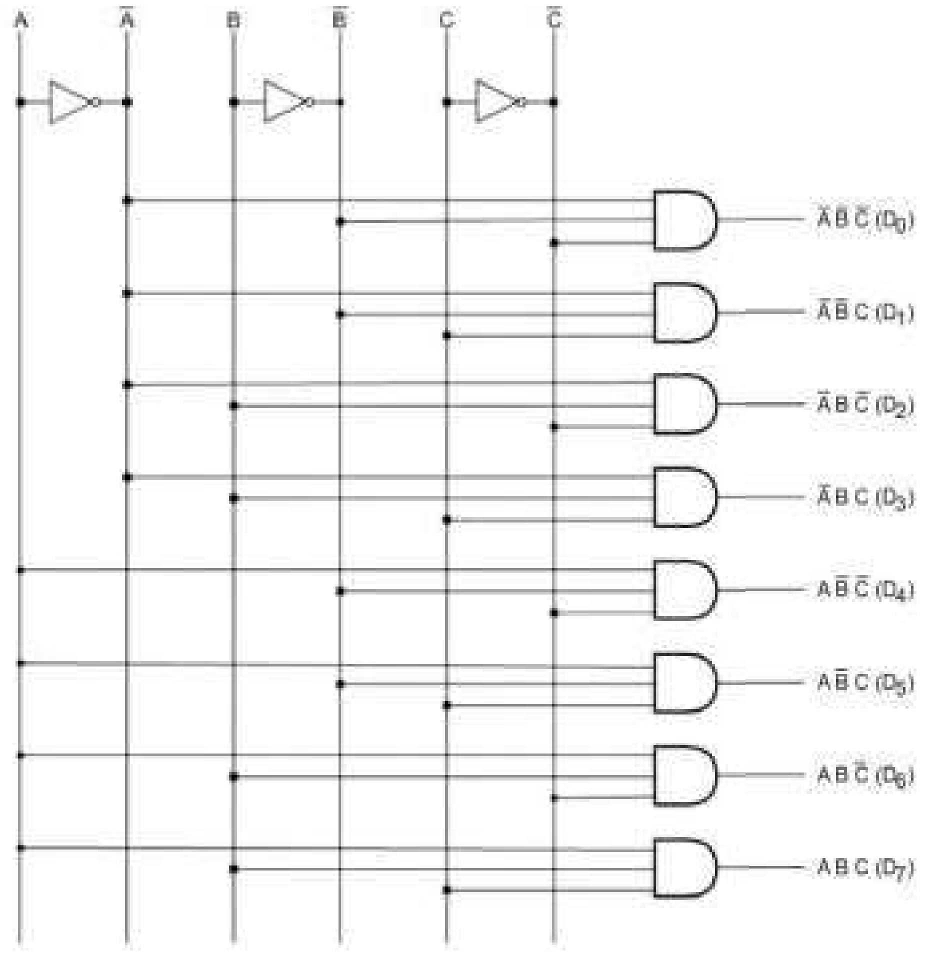
$$D_4 \to \overline{A}.\overline{B}.\overline{C},\ D_5 \to A.\overline{B}.C,\ D_6 \to A.B.\overline{C},\ D_7 \to A.B.C$$

Implementing Boolean Functions with Decoders

A decoder can be conveniently used to implement a given Boolean function. The decoder generates the required minterms and an external OR gate is used to produce the sum of minterms. Figure shows the logic diagram where a 3-to-8 line decoder is used to generate the Boolean function given by the equation

$$Y = A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.B.C + \overline{A}.\overline{B}.\overline{C}$$

In general, an n-to-2^n decoder and m external OR gates can be used to implement any combinational circuit with n inputs and m outputs. We can appreciate that a Boolean function with a large number of minterms, if implemented with a decoder and an external OR gate, would require an OR gate with an equally large number of inputs. Let us consider the case of implementing a four-variable Boolean function with 12 minterms using a 4-to-16 line decoder and an external OR gate. The OR gate here needs to be a 12-input gate. In all such cases, where the number of minterms in a given Boolean function with n variables is greater than $2^n/2$ (or 2^{n-1}), the complement Boolean function will have fewer minterms. In that case it would be more advantageous to do NORing of minterms of the complement Boolean function using a NOR gate rather than doing ORing of the given function using a OR gate. The output will be nothing but the given Boolean function.



A
$\bar{A}$
B
$\bar{B}$
C
$\bar{C}$
$\bar{A}\bar{B}\bar{C}$ (D_0)
$\bar{A}\bar{B}C$ (D_1)
$\bar{A}B\bar{C}$ (D_2)
$\bar{A}BC$ (D_3)
$A\bar{B}\bar{C}$ (D_4)
$A\bar{B}C$ (D_5)
$AB\bar{C}$ (D_6)
ABC (D_7)

INPUTS			OUTPUTS							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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

Figure Logic diagram of a 3-to-8 line decoder.

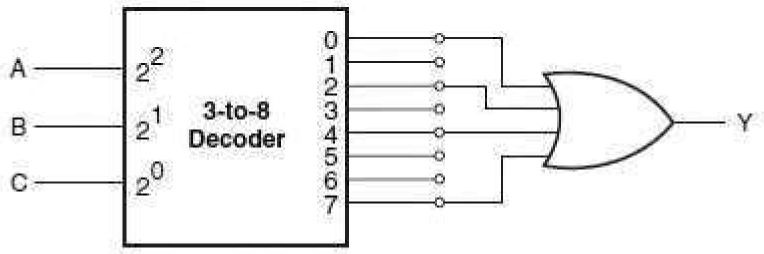


Figure Implementing Boolean functions with decoders.

Cascading Decoder Circuits

There can possibly be a situation where the desired number of input and output lines is not available in IC decoders. More than one of these devices of a given size may be used to construct a decoder that can handle a larger number of input and output lines. For instance, 3-to-8 line decoders can be used to construct 4-to-16 or 5-to-32 or even larger decoder circuits. The basic steps to be followed to carry out the design are as follows:

1. If n is the number of input lines in the available decoder and N is the number of input lines in the desired decoder, then the number of individual decoders required to construct the desired decoder circuit would be 2^{N-n}.
2. Connect the less significant bits of the input lines of the desired decoder to the input lines of the available decoder.
3. The left-over bits of the input lines of the desired decoder circuit are used to enable or disable the individual decoders.
4. The output lines of the individual decoders together constitute the output lines, with the outputs of the less significant decoder constituting the less significant output lines and those of the higher-order decoders constituting the more significant output lines. The concept is further illustrated in solved example 8.8, which gives the design of a 4-to-16 decoder using 3-to-8 decoders.

Problem

Implement a full adder circuit using a 3-to-8 line decoder.

Solution

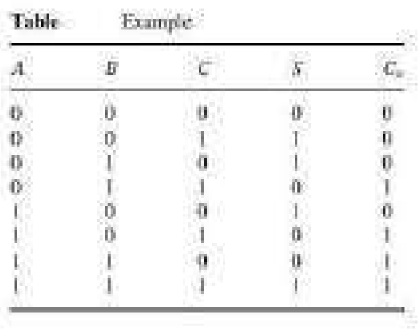
A decoder with an OR gate at the output can be used to implement the given Boolean function. The decoder should at least have as many input lines as the number of variables in the Boolean function to be implemented. The truth table of the full adder is given in Table , and Fig. shows the hardware implementation.

From the truth table, Boolean functions for SUM and CARRY outputs are given by the following equations:

$$\text{Sum output } S = \Sigma\ 1, 2, 4, 7$$

$$\text{Carry output } C_o = \Sigma\ 3, 5, 6, 7$$

Table Example



A	B	C	S	C_o
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

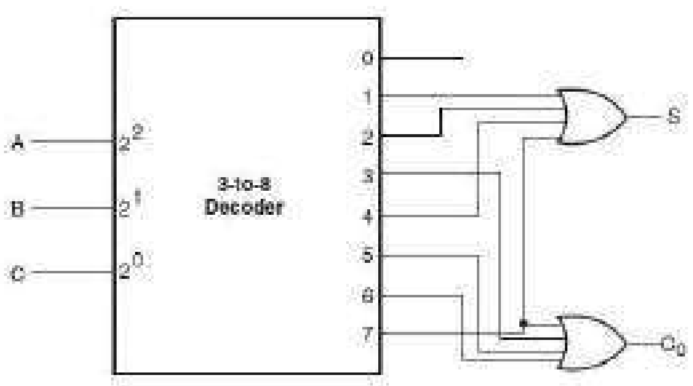


Figure Example

2 MARKS

1. **Define combinational logic.**
   (or)
   What is combinational logic
   When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called combinational logic.

2. **Explain the design procedure for combinational circuits.**
   (or)
   Write short notes on design procedure for combinational circuits
   The problem definition
   - Determine the number of available input variables & required O/P variables.
   - Truth Table Construction
   - Obtain simplified Boolean expression for each O/P (using K-Map).
   - Obtain the logic diagram.

3. **Define Half adder and full adder**
   (or)