TABLE OF CONTENTS

Unit 1

Unit 2

4 Design using SSI Chips

4.1 : Code Converters

Q.1 State the procedure to design code converters.

Ans. : Step 1 : Write the truth table showing the relationship between input code and output code.

Step 2 : For each output code bit determine the simplified Boolean expression using K-map.

Step 3 : Realize the code converter using logic gates.

Q.2 Design a logic circuit to convert the 8421 BCD to Excess-3 code.

☞ [SPPU : Dec.-12,13, Marks 8]

Ans. : Step 1 : Form the truth table relating BCD and Excess-3 code

Excess-3 code is a modified form of a BCD number. The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as shown in Table Q.2.1.

Decimal	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1

5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table Q.2.1

Input code : BCD code : $D_3\ D_2\ D_1\ D_0$ (D_0 LSB)

Output code : Excess-3 code : $E_3\ E_2\ E_1\ E_0$ (E_0 LSB)

Step 2 : K-map simplification for each Excess-3 code output.

For E_3

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$\therefore E_3 = D_3 + D_2(D_0 + D_1)$

For E_2

D_3D_2 \ D_1D_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

$\therefore E_2 = D_2\overline{D}_1\overline{D}_0 + \overline{D}_2(D_0 + D_1)$

For E_1

D_3D_2 \ D_1D_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

$E_1 = \overline{D}_1\overline{D}_0 + D_1D_0$
$= D_1 \odot D_0$

For E_0

D_3D_2 \ D_1D_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

$E_0 = \overline{D}_0$

Step 3 : Realization of code converter.

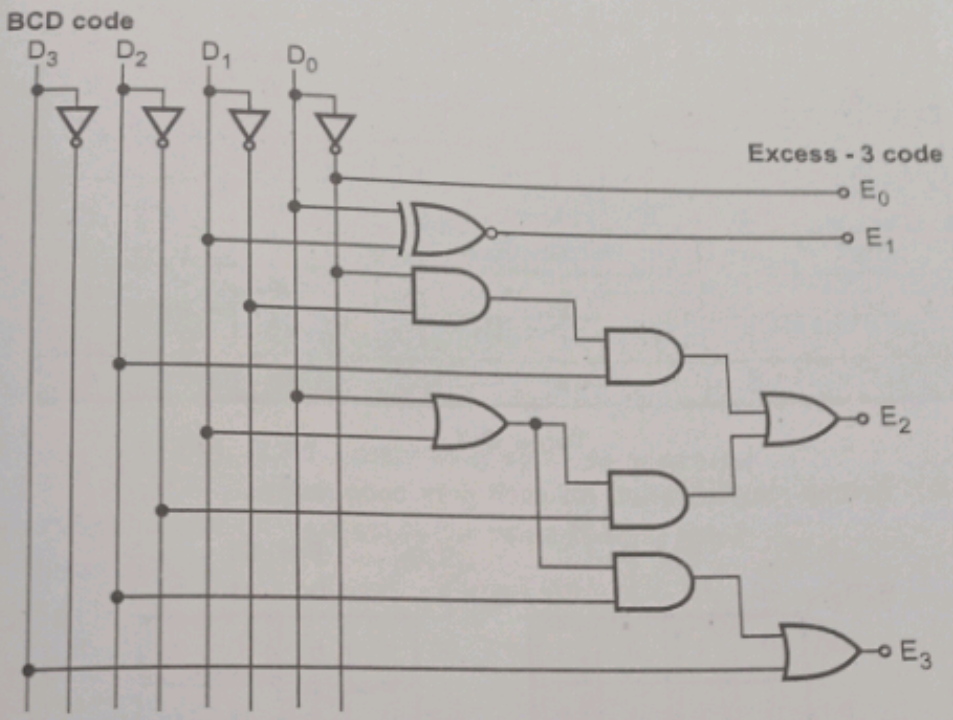


Fig. Q.2.2 BCD to excess-3 code converter

Q.3 Design and implement a 8421 to Gray code converter. Realize the converter using only NAND gates. ☞ [SPPU : May-12, Marks 4]

Ans. : Step 1 : Form the Truth table relating 8421 binary code and Gray code

Input code : Binary code : $B_3\ B_2\ B_1\ B_0$

Output code : Gray code : $G_3\ G_2\ G_1\ G_0$

Decimal	Binary code				Gray code			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0

5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Table Q.3.1

Step 2 : K-map simplification for each gray code output

For G_0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$G_0 = B_1\overline{B}_0 + \overline{B}_1B_0$
$= B_1 \oplus B_0$

For G_1

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1$
$= B_2 \oplus B_1$

For G_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$G_2 = \overline{B}_3B_2 + B_3\overline{B}_2 = B_3 \oplus B_2$

For G_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$G_3 = B_3$

Fig. Q.3.1

Step 3 : Realization of code converter using XOR-gates

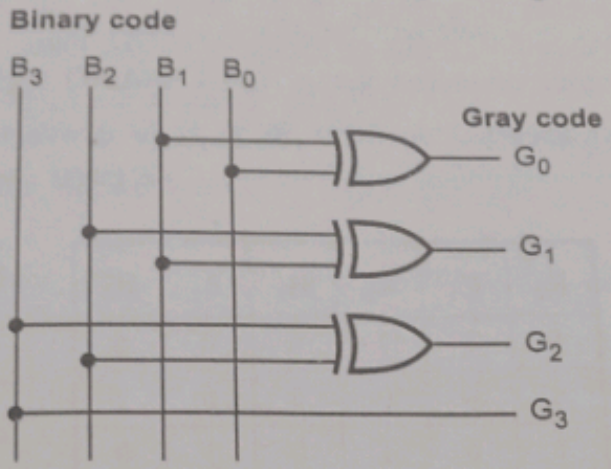
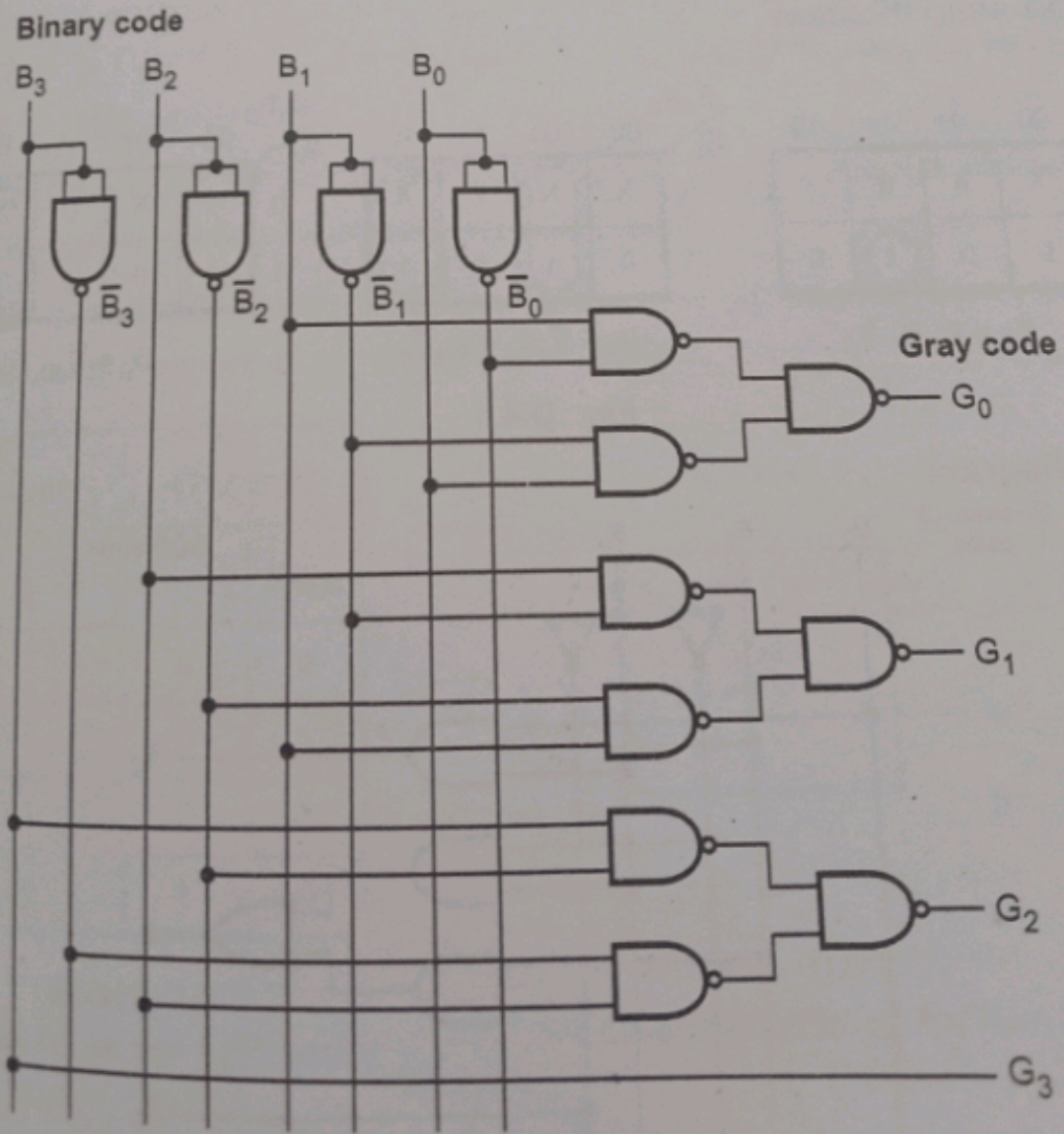


Fig. Q.3.2 Binary to gray code converter

Step 4 : Realization of code converter using NAND gates



For this converter we have derived the Boolean expressions for each gray code output in the sum of product (SOP) form. We can implement SOP expression using AND-OR logic or NAND-NAND logic. Let us see the implementation of code converter using NAND-NAND logic.

Q.4 Design a 3-bit excess 3 to 3-bit BCD code converter using logic gates.

☞ [SPPU : May-15, Marks 8]

Ans. :

E_2	E_1	E_0	B_2	B_1	B_0
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	0	0

K-map simplification

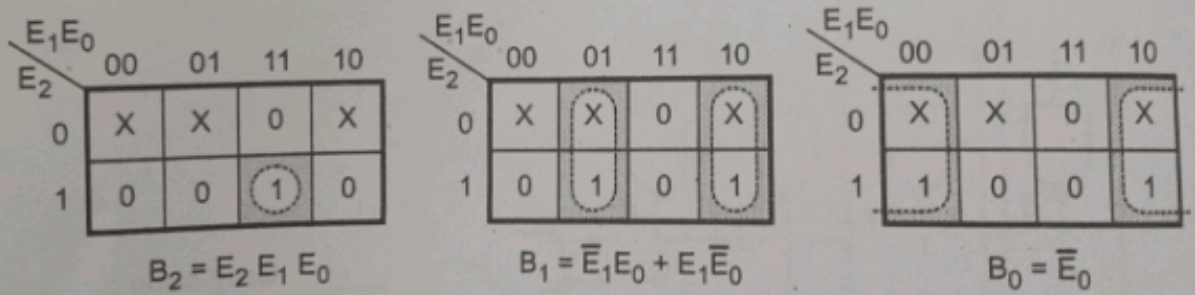
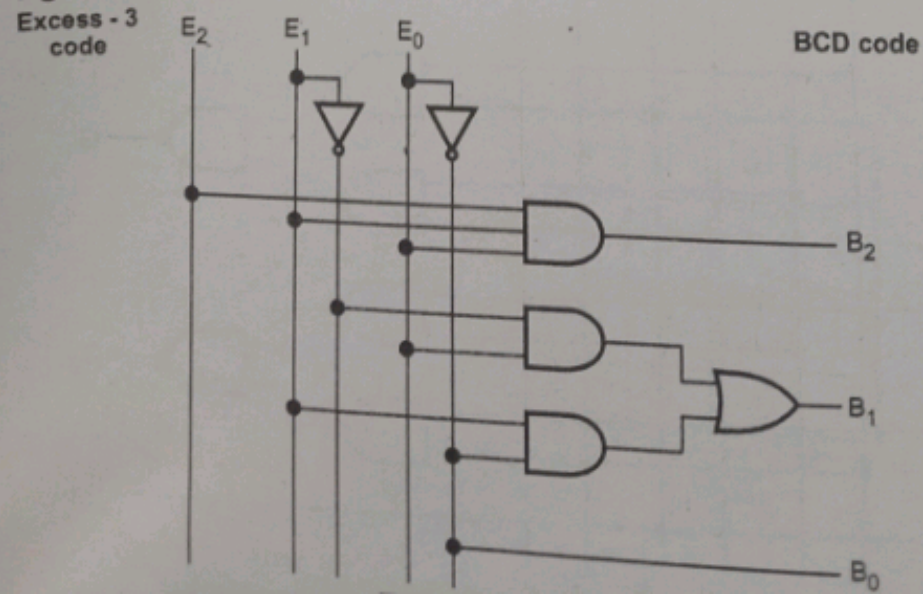


Fig. Q.4.1

Logic diagram



4.2 : Half - Adder, Full Adder, Half Subtractor, Full Subtractor, Binary Adder (IC 7483)

Q.5 What is half adder and full adder ?

☞ [SPPU : May-07, 14, Marks 4]

Ans. : • The logic circuit which performs addition of two bits is called a **half-adder.**

• The circuit which performs addition of three bits (two significant bits and a previous carry) is a **full-adder.**

Q.6 Draw and explain the function of half-adder and full adder with suitable diagram.

OR Draw the suitable diagram of full adder.

OR What are the full adder's inputs that will produce each of the following outputs ? i) $\Sigma = 0$, $C_{out} = 0$ ii) $\Sigma = 1$, $C_{out} = 0$ iii) $\Sigma = 1$, $C_{out} = 1$ iv) $\Sigma = 0$, $C_{out} = 1$ ☞ [SPPU : Dec.-15, Marks 2]

Ans. : Half-adder

• The half-adder operation needs two binary inputs : augend and addend bits; and two binary outputs : sum and carry.

• The truth table shown in Table Q.6.1 gives the relation between input and output variables for half-adder operation.

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table Q.6.1
Truth table for half-adder

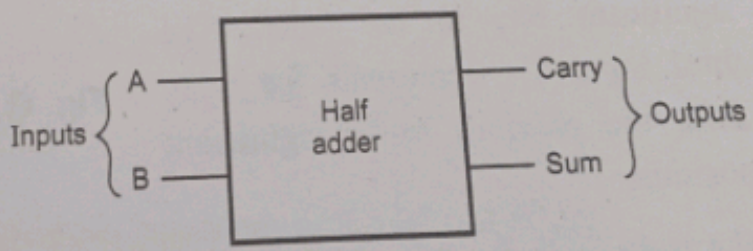


Fig. Q.6.1
Block schematic of half-adder

K-map simplification for carry and sum

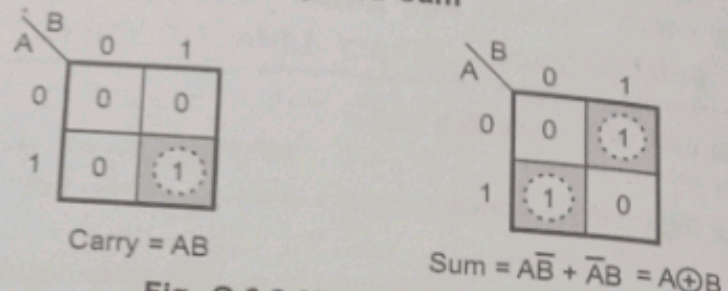


Fig. Q.6.2 Maps for half-adder

Logic diagram

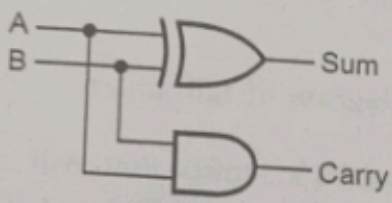


Fig. Q.6.3 Logic diagram for half-adder

Full adder

- A full-adder is a combinational circuit that forms the arithmetic sum of three input bits.
- It consists of three inputs and two outputs.
- Two of the input variables, denoted by A and B, represent the two significant bits to be added. The third input C_{in}, represents the carry from the previous lower significant position.

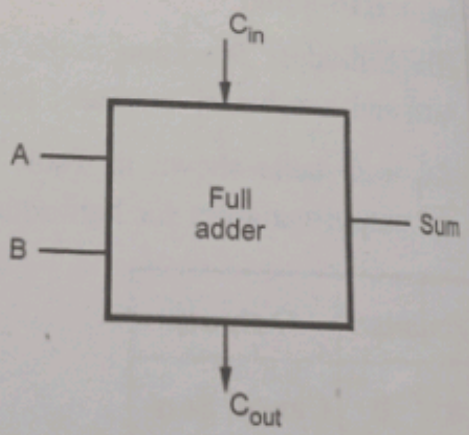


Fig. Q.6.4 Block schematic of full-adder

The truth table for full-adder is shown in Table Q.6.2.

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1

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

Table Q.6.2 Truth table for full-adder

K-map simplification for carry and sum

For carry (C_{out})

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

For sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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

$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

Fig. Q.6.5 Maps for full-adder

Logic diagram

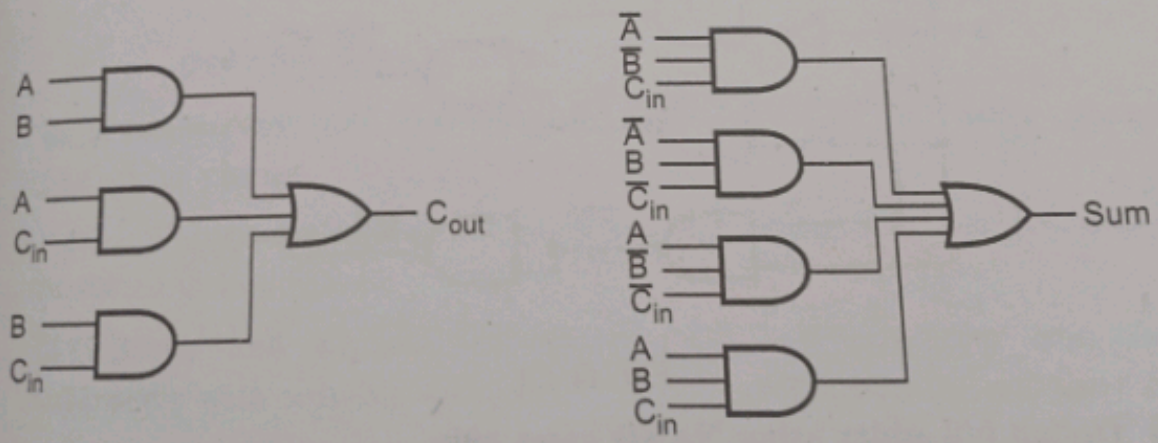


Fig. Q.6.6 Sum of product implementation of full-adder

- The Boolean function for sum can be further simplified as follows :

$$Sum = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= C_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}\,(\overline{A}\,B + A\,\overline{B})$$

$$= C_{in}\,(A \odot B) + \overline{C}_{in}\,(A \oplus B)$$

$$= C_{in}\,(\overline{A \oplus B}) + \overline{C}_{in}\,(A \oplus B) = C_{in} \oplus (A \oplus B)$$

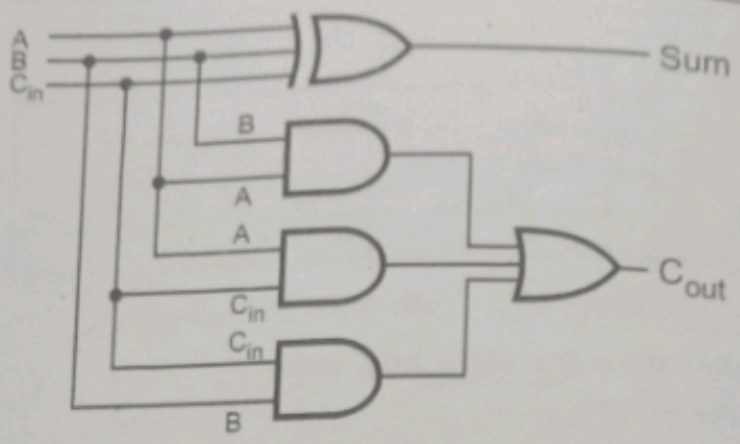


Fig. Q.6.7 Implementation of full-adder

Q.7 Draw half adder using NAND gates.

Ans. : For half adder :

$$\text{Sum} = A\overline{B} + \overline{A}B \qquad C_{out} = AB$$

$$= \overline{A\overline{B} + \overline{A}B} \qquad = \overline{\overline{AB}}$$

$$= \overline{\overline{A\overline{B}} \cdot \overline{\overline{A}B}}$$

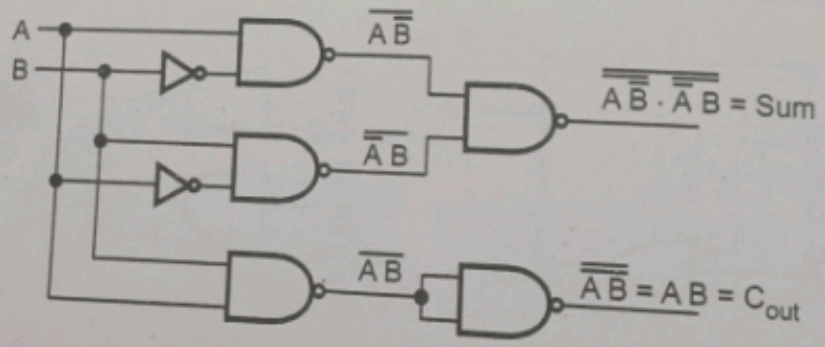


Fig. Q.7.1

Q.8 Design full adder using NAND gates only.

Ans. : Fig. Q.6.6 shows sum of product implementation of full adder using basic gates. We know that sum of product expressions can be implemented by NAND-NAND logic. Thus by replacing AND and OR gates by NAND gates in Fig. Q.6.6. We can implement full adder using NAND gates.

Q.9 Implement full adder using two half adders.

☞ [SPPU : May-07,14, Marks 3]

Ans. : • A full-adder can also be implemented with two half-adders and one OR gate, as shown in the Fig. Q.9.1.

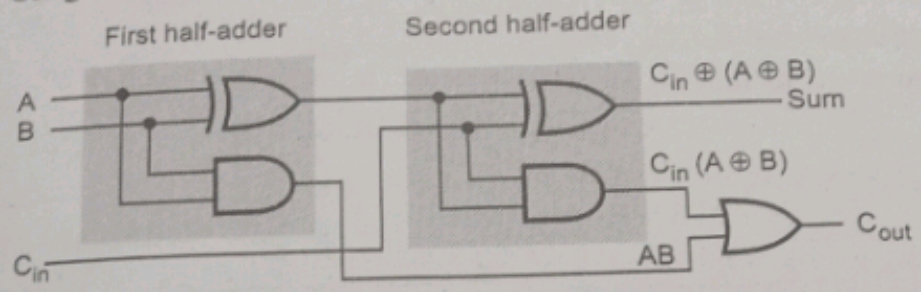


Fig. Q.9.1 Implementation of a full-adder with two half-adders and an OR gate

• The sum output from the second half-adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

$$\begin{aligned} C_{out} &= AB + A\,C_{in} + B\,C_{in} \\ &= AB + A\,C_{in}\,(B + \overline{B}) + B\,C_{in}\,(A + \overline{A}) \\ &= AB + ABC_{in} + A\,\overline{B}\,C_{in} + ABC_{in} + \overline{A}\,BC_{in} \\ &= AB\,(1 + C_{in} + C_{in}) + A\,\overline{B}C_{in} + \overline{A}\,BC_{in} \\ &= AB + A\overline{B}C_{in} + \overline{A}BC_{in} = AB + C_{in}\,(A\overline{B} + \overline{A}B) \\ &= AB + C_{in}\,(A \oplus B) \end{aligned}$$

Q.10 Implement the following Boolean function F using three half-adder circuits : F(A, B, C) = A ⊕ B ⊕ C.

Ans. : We can implement the given function using only two half-adders as shown in Fig. Q.9.1.

Q.11 Draw and explain the function of half-subtractor and full subtractor with suitable diagram. ☞ **[SPPU : Dec.-08, Marks 8]**

Ans. : Half subtractor :

• A half-subtractor is a combinational circuit that subtracts two-bits and produces their difference.

• It also has an output to specify if a 1 has been borrowed.

• Let us designate minuend bit as A and the subtrahend bit as B. The result of operation A – B for all possible values of A and B is tabulated in Table Q.11.1.

- The Boolean expression for the outputs of half-subtractor can be determined as follows.

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table Q.11.1 Truth table for half-subtractor

K-map simplification for half-subtractor

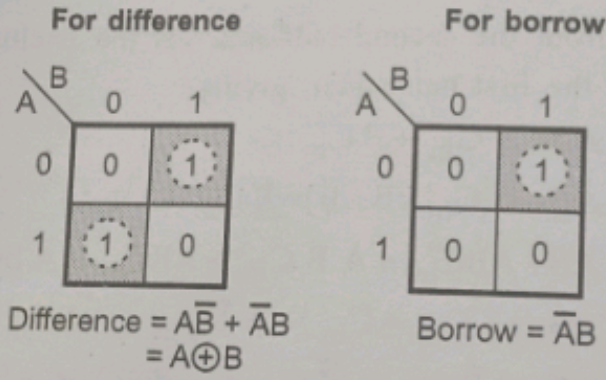


Fig. Q.11.1

Logic diagram

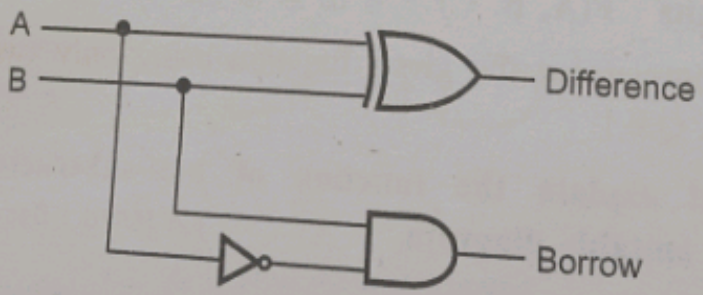


Fig. Q.11.2 Implementation of half-subtractor

Full-subtractor

- A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage.
- This circuit has three inputs and two outputs.

- The three inputs are A, B and B_{in}, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, D and B_{out}, represent the difference and output borrow, respectively.
- The Table Q.11.2 shows the truth table for full-subtractor.

Inputs			Outputs	
A	B	B_{in}	D	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

Table Q.11.2 Truth table for full-subtractor

K-map simplification of D and B_{out}

For D

A \ BB_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$D = \overline{A}\,\overline{B}B_{in} + \overline{A}B\overline{B}_{in} + A\overline{B}\,\overline{B}_{in} + ABB_{in}$$

For B_{out}

A \ BB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$$

Fig. Q.11.3 Maps for full-subtractor

Logic diagram

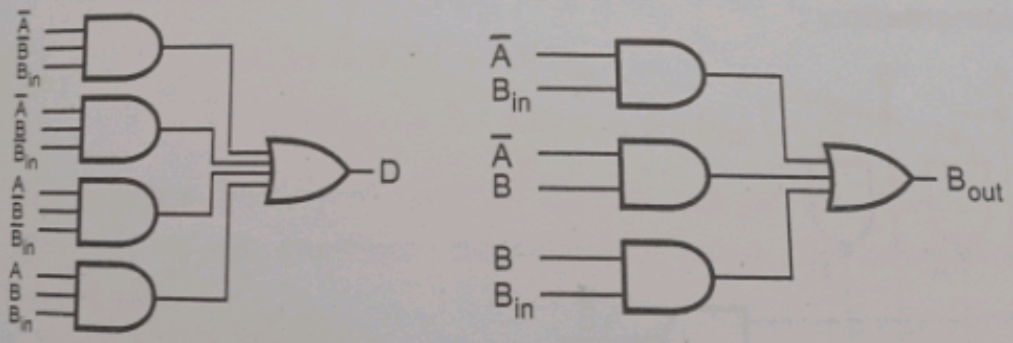


Fig. Q.11.4 Sum of product implementation of full-subtractor

- The Boolean function for D (difference) can be further simplified as follows :

$$D = \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B\,\overline{B}_{in} + A\,\overline{B}\,\overline{B}_{in} + A\,B\,B_{in}$$

$$= B_{in}(\overline{A}\,\overline{B} + AB) + \overline{B}_{in}(\overline{A}\,B + A\,\overline{B})$$

$$= B_{in}(A \odot B) + \overline{B}_{in}(A \oplus B)$$

$$= B_{in}(\overline{A \oplus B}) + \overline{B}_{in}(A \oplus B) = B_{in} \oplus (A \oplus B)$$

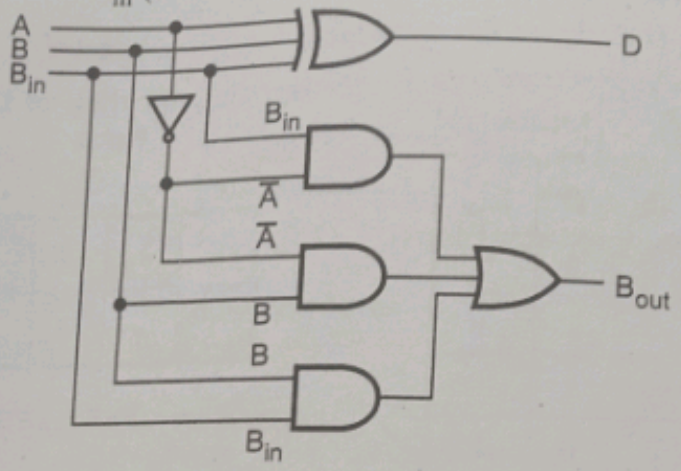


Fig. Q.11.5 Implementation of full-subtractor

Q.12 Draw half subtractor using NAND gates.

Ans. : For half subtractor :

Difference $= A\,\overline{B} + \overline{A}\,B$

$$= \overline{A\,\overline{B} + \overline{A}\,B} = \overline{\overline{A\,\overline{B}} \cdot \overline{\overline{A}\,B}}$$

Borrow $= \overline{A}\,B$

$$= \overline{\overline{A}\,B}$$

Implementation :

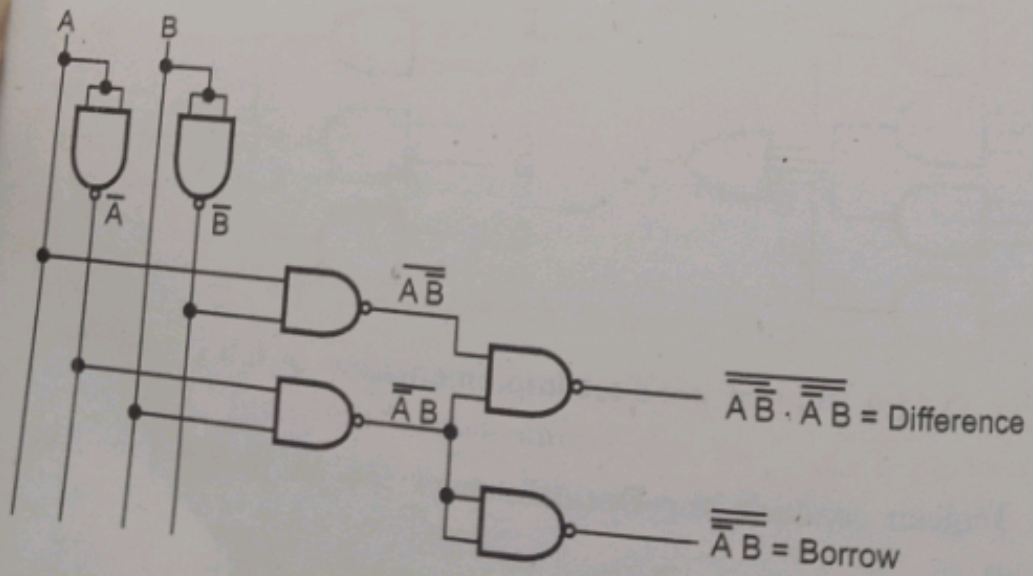


Fig. Q.12.1

Q.13 Implement a full subtractor with two half subtractors and an OR gate.

Ans. : • A full subtractor can also be implemented with two half-subtractors and one OR gate, as shown in the Fig. Q.13.1.

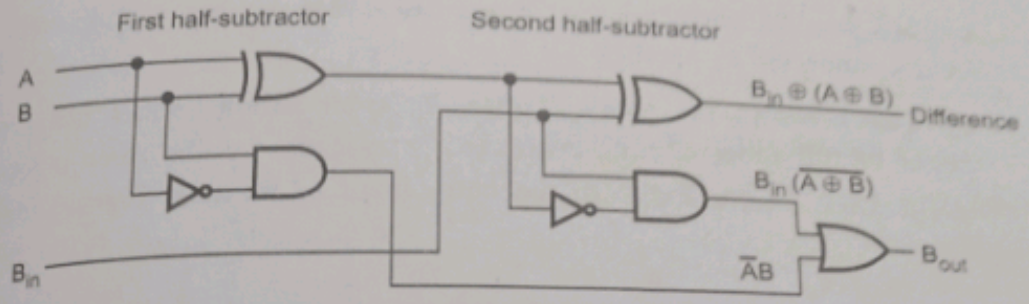


Fig. Q.13.1 Implementation of a full-subtractor with two half-subtractors and an OR gate

- The difference output from the second half-subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, which is same as difference output of full-subtractor.
- The borrow output for full subtractor is given as,

$$\begin{aligned} B_{out} &= \overline{A}\,B_{in} + \overline{A}\,B + B\,B_{in} \\ &= \overline{A}\,B_{in}\,(B + \overline{B}) + \overline{A}\,B + B\,B_{in}\,(A + \overline{A}) \\ &= \overline{A}\,B\,B_{in} + \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B + AB\,B_{in} + \overline{A}\,B\,B_{in} \\ &= \overline{A}\,B\,(B_{in} + 1 + B_{in}) + \overline{A}\,\overline{B}\,B_{in} + AB\,B_{in} \\ &= \overline{A}\,B + \overline{A}\,\overline{B}\,B_{in} + AB\,B_{in} \\ &= \overline{A}\,B + B_{in}\,(\overline{A}\,\overline{B} + AB) = \overline{A}\,B + B_{in}\,\overline{(A \oplus B)} \end{aligned}$$

- This Boolean function is same as borrow out of the full-subtractor. Therefore, we can implement full-subtractor using two half-subtractors and OR gate.

4.3 : n-bit Binary Adder

Q.14 Explain the working of n-bit binary adder.

Ans. : Parallel Adder

- A single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders must be employed.
- A n-bit, parallel adder can be constructed using number of full adder circuits connected in parallel.
- Fig. Q.14.1 shows the block diagram of n-bit parallel adder using n number of full-adder circuits connected in cascade, i.e. the carry output of each adder is connected to the carry input of the next higher-order adder.

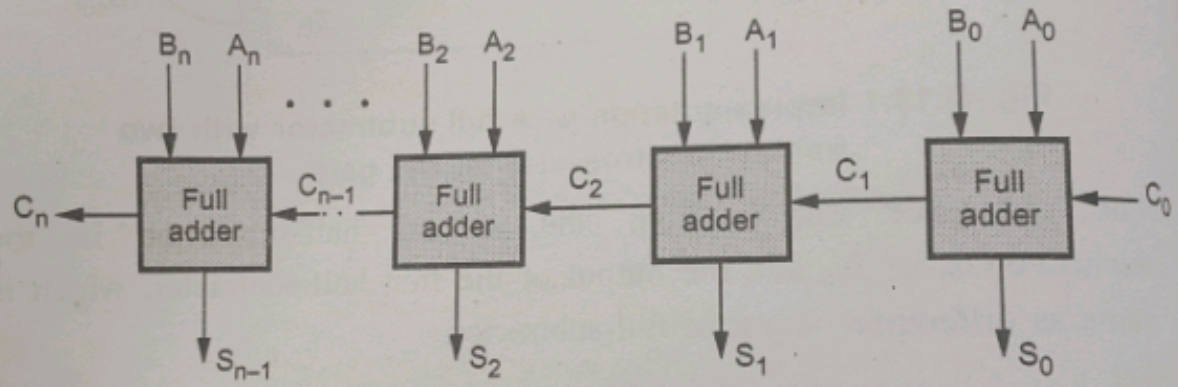


Fig. Q.14.1 Block diagram of n-bit parallel adder

- It should be noted that either a half-adder can be used for the least significant position or the carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

END...

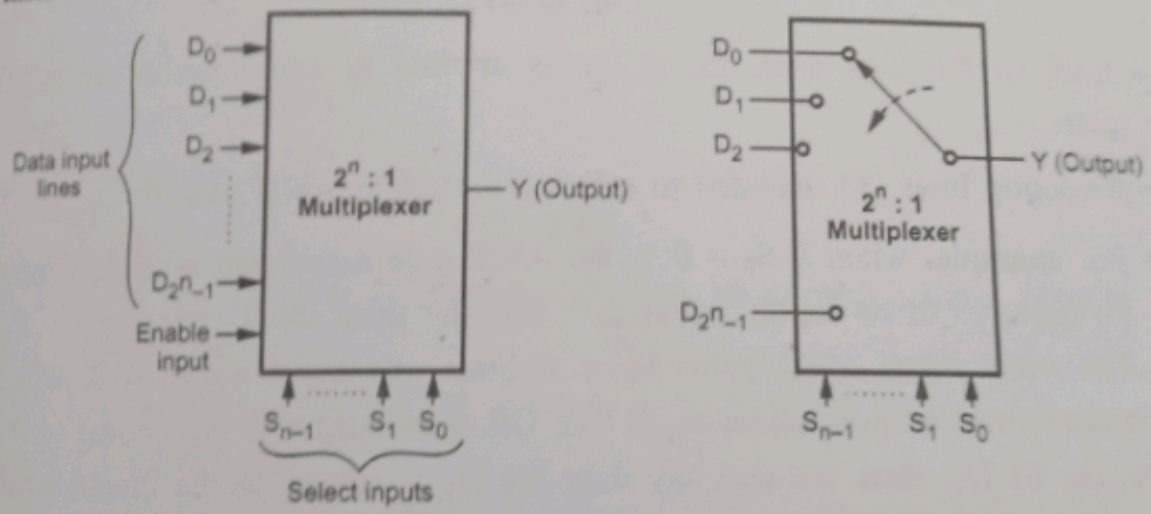
Unit 2

5 Introduction to MSI Chips and Design using MSI Chips

5.1 : Multiplexers (IC 74153) and Implementation of Logic Function using IC 74153

Q.1 What is multiplexer ?

Ans. : • In digital systems, many times it is necessary to select single data line from several data-input lines, and the data from the selected data line should be available on the output. The digital circuit which does this task is a **multiplexer**.



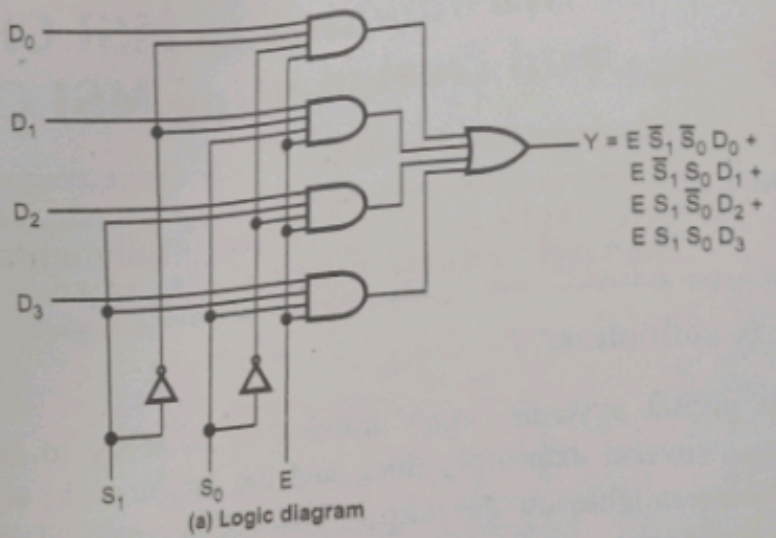
(a) Block diagram of $2^n : 1$ multiplexer (b) Equivalent circuit

Fig. Q.1.1

- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. Therefore, multiplexer is '**many into one**' and it provides the digital equivalent of an analog selector switch.

Q.2 Draw and explain the logic diagram of 4 : 1 line multiplexer.

Ans. : Fig. Q.2.1 (a) shows 4-to-1 line multiplexer.



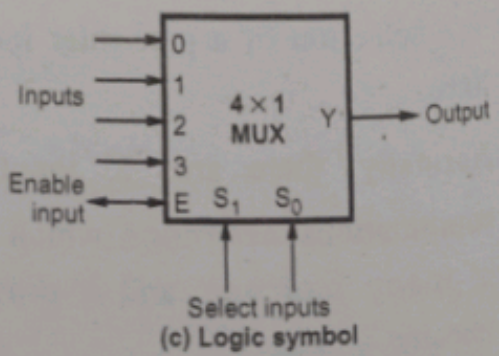
(a) Logic diagram

Fig. Q.2.1

- Each of the four lines, D_0 to D_3, is applied to one input of an AND gate.
- Selection lines are decoded to select a particular AND gate.
- For example, when $S_1S_0 = 0\ 1$, the AND gate associated with data input D_1 has two of its inputs equal to 1 and the third input connected to D_1. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of D_1, thus we can say data bit D_1 is routed to the output when $S_1\ S_0 = 0\ 1$.

E	S_1	S_0	Y
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3
0	X	X	0

(b) Function table



(c) Logic symbol

Fig. Q.2.1 4 to 1 line multiplexer

Q.3 Show how 4-input multiplexer can be realize using 2-input multiplexers.

Ans. : • It is possible to expand range of inputs for multiplexer beyond the available range by interconnecting several multiplexers in cascade.

• The circuit with two or more multiplexers connected to obtain the multiplexer with more number of inputs is known as **multiplexer tree.**

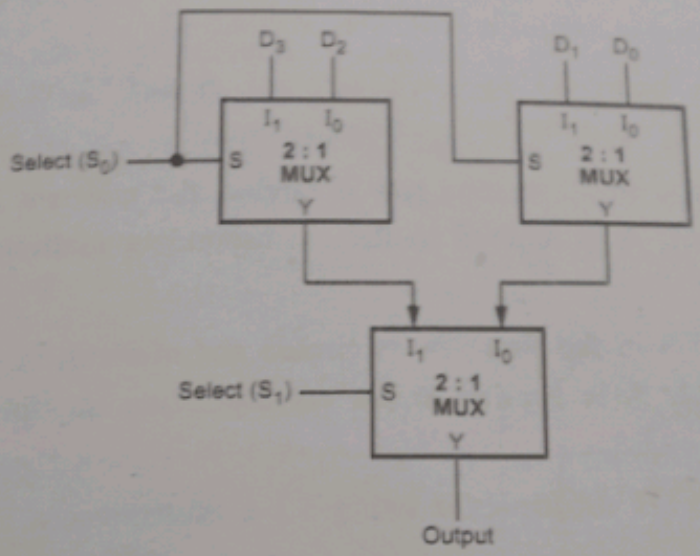


Fig. Q.3.1

Q.4 Explain the process of implementation of combinational circuit using multiplexer.

Ans. : Let us implement $F(A, B, C) = \sum m\,(1, 3, 5, 6)$ Boolean function using 4 : 1 multiplexer.

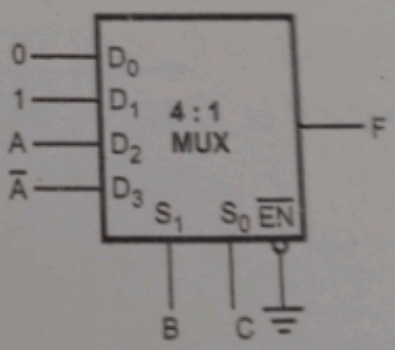
Step 1 : Connect least significant variables as select inputs of multiplexer. Here, connect C to S_0 and B to S_1.

Step 2 : Derive inputs for multiplexer using implementation table.

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

Most significant variable

(a) Implementation table



(b) Multiplexer implementation

Fig. Q.4.1

As shown in the Fig. Q.4.1 (a) the implementation table is nothing but the list of the inputs of the multiplexer and under them list of all the minterms in two rows. The first row lists all those minterms where A is complemented, and the second row lists all the minterms with A uncomplemented. The minterms given in the function are circled and then each column is inspected separately as follows :

- If the two minterms in a column are not circled, 0 is applied to the corresponding multiplexer input (see column 0).
- If the two minterms in a column are circled, 1 is applied to the corresponding multiplexer input (see column 1).
- If the minterm in the second row is circled and minterm in the first row is not circled, A is applied to the corresponding multiplexer input (see column 2).
- If the minterm in the first row is circled and minterm in the second row is not circled, $\overline{A}$ is applied to the corresponding multiplexer input (see column 3).

Q.5 Design 16 : 1 multiplexer using 4 : 1 multiplexers.

☞ **[SPPU : May-11, Marks 8]**

Ans. : Since there are 16-inputs for the multiplexers we require four 4 : 1 multiplexers to satisfy input needs. The four outputs of 4 : 1 multiplexers are again multiplexed by 4 : 1 multiplexer to generate final output.

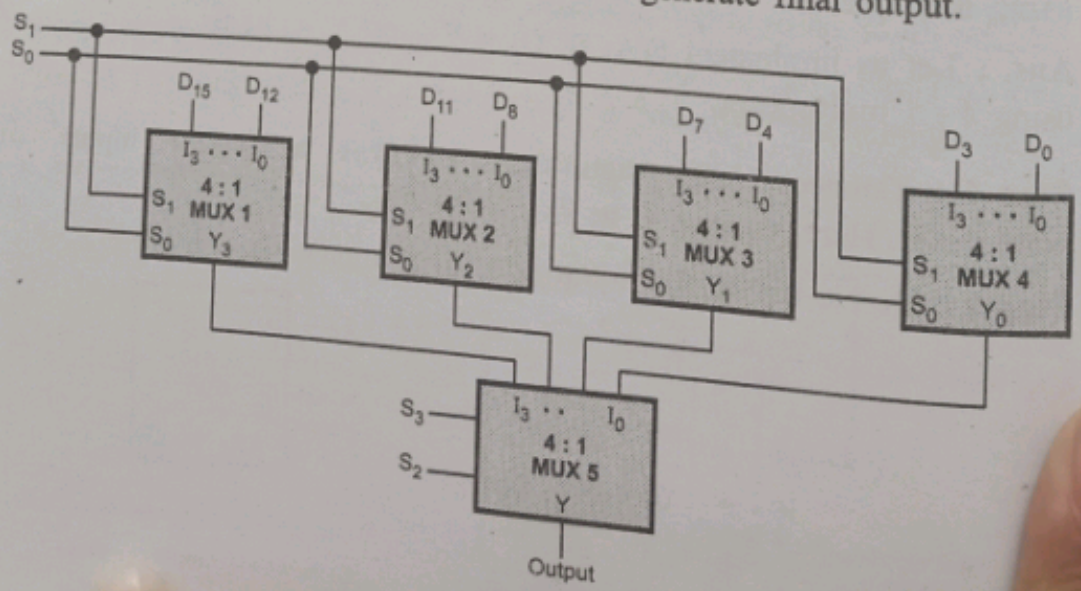


Fig. Q.5.1

Step 1 : Connect the select lines (S_1 and S_0) of four multiplexers in parallel.

Step 2 : Connect the most significant select lines (S_3 and S_2) to the MUX 5.

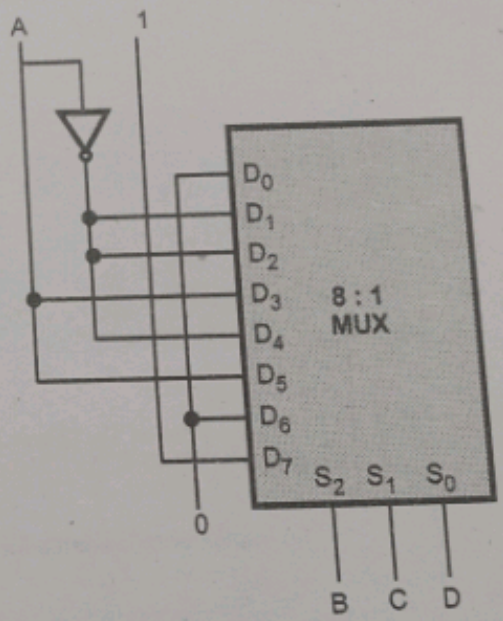
Step 3 : Connect the outputs Y_0, Y_1, Y_2 and Y_4 of four multiplexers as data inputs for the MUX 5, as shown in the Fig. Q.5.1.

Q.6 Implement the following Boolean function with 8 : 1 multiplexer
$F(A, B, C, D) = \pi M (0, 3, 5, 8, 9, 10, 12, 14)$

Ans. : Here, instead of minterms, maxterms are specified. Thus, we have to circle maxterms which are not included in the Boolean function. Fig. Q.6.1 shows the implementation of Boolean function with 8 : 1 multiplexer.

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

(a) Implementation table



(b) Multiplexer implementation

Fig. Q.6.1

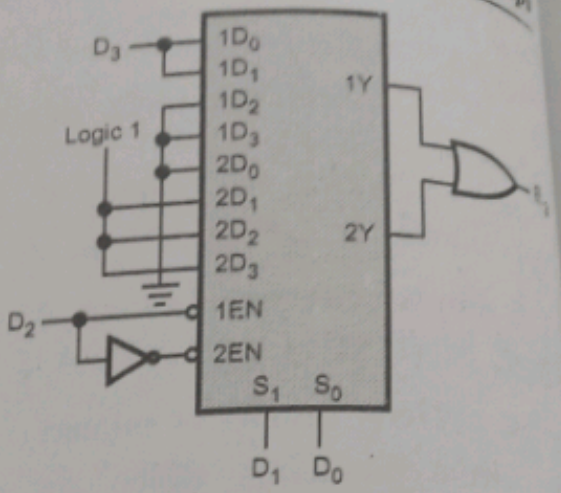
Q.7 Design and implement BCD to Excess-3 code converter using dual 4:1 multiplexers and some logic gates. ☞ [SPPU : May-13, Marks 8]

Ans. : Refer Q.2 of Chapter - 4 for truth table of BCD to excess - 3 code conversion.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}_3$	0	1	2	3	4	(5)	(6)	(7)
D_3	(8)	(9)	10	11	12	(13)	(14)	(15)
	D_3	D_3	0	0	0	1	1	1

Note : Don't care minterms 13, 14 and 15 are considered as logic 1.

(a) Implementation table for E_3



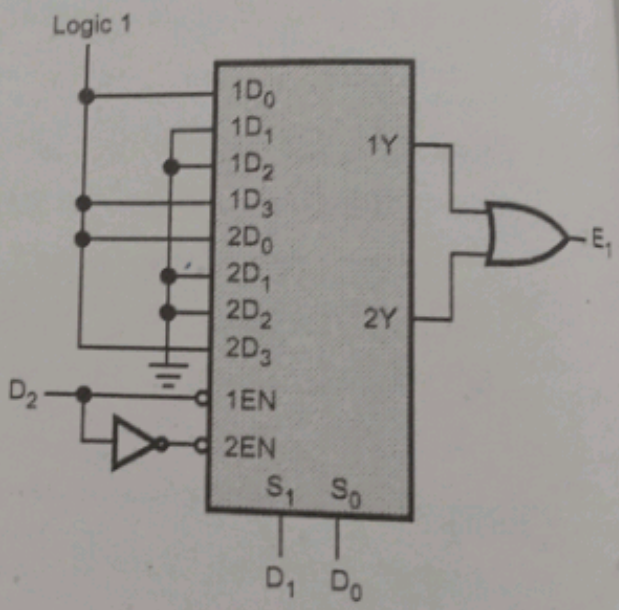
(b) Implementation

Fig. Q.7.1

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}_3$	(0)	1	2	(3)	(4)	5	6	(7)
D_3	(8)	9	10	(11)	(12)	13	14	(15)
	1	0	0	1	1	0	0	1

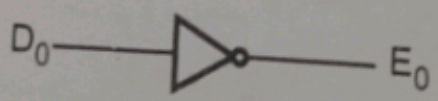
Note : Don't care minterms 11, 12 and 15 are considered as logic 1.

(a) Implementation table for E_1



(b) Implementation

Fig. Q.7.2



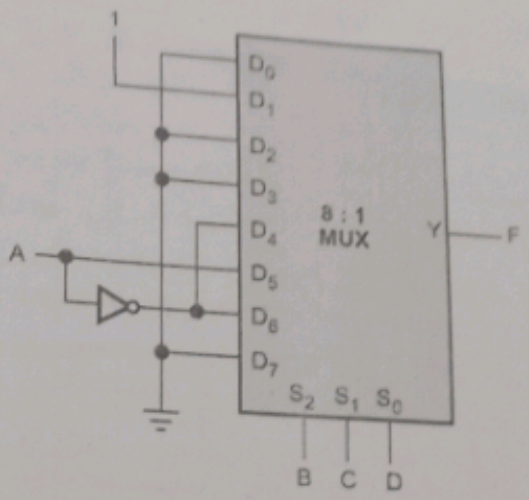
Q.8 Implement the following Boolean function using single 8 : 1 multiplexer.

$$F(A, B, C, D) = \sum m(1, 4, 6, 9, 13).$$ ☞ **[SPPU : May-17, Marks 6]**

Ans. :

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

(a) Implementation table



(b) Implementation

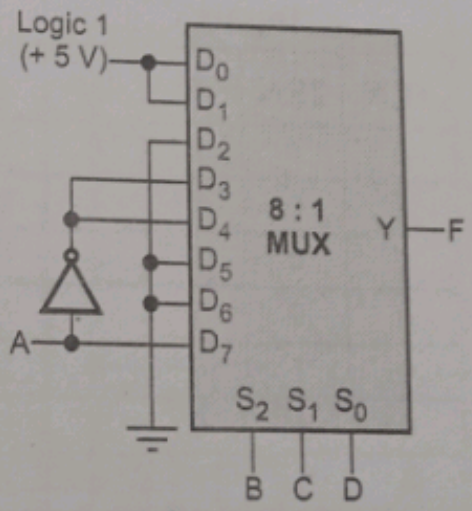
Fig. Q.8.1

Q.9 Implement given function using 8 : 1 MUX and logic gates $F(A, B, C, D) = \sum m(0, 1, 3, 4, 8, 9, 15)$ ☞ [SPPU : Dec.-18, Marks 6]

Ans. :

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

Implementation Table



Implementation

Fig. Q.9.1

Q.10 Design and implement 8 : 1 MUX using two 4 : 1 MUX and implement given function $F(X, Y, Z) = \sum m(1,3,4,7)$.

☞ [SPPU : May-19, Marks 6]

Ans. :

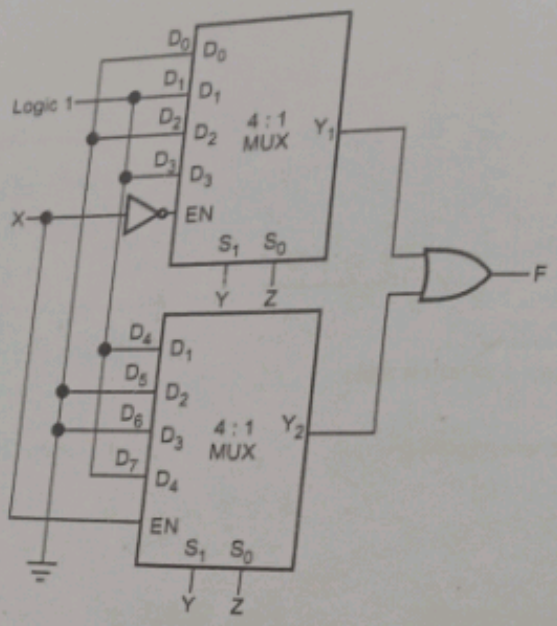


Fig. Q.10.1

Q.11 Write a note on IC MUX 74153.

Ans. :

Inputs				Outputs	
1EN	2EN	B	A	1Y	2Y
0	0	0	0	$1D_0$	$2D_0$
0	0	0	1	$1D_1$	$2D_1$
0	0	1	0	$1D_2$	$2D_2$
0	0	1	1	$1D_3$	$2D_3$
0	1	0	0	$1D_0$	0
0	1	0	1	$1D_1$	0
0	1	1	0	$1D_2$	0
0	1	1	1	$1D_3$	0
1	0	0	0	0	$2D_0$
1	0	0	1	0	$2D_1$
1	0	1	0	0	$2D_2$
1	0	1	1	0	$2D_3$
1	1	x	x	0	0

Table Q.11.1 Truth table for 74XX153, dual 4-to-1 multiplexer

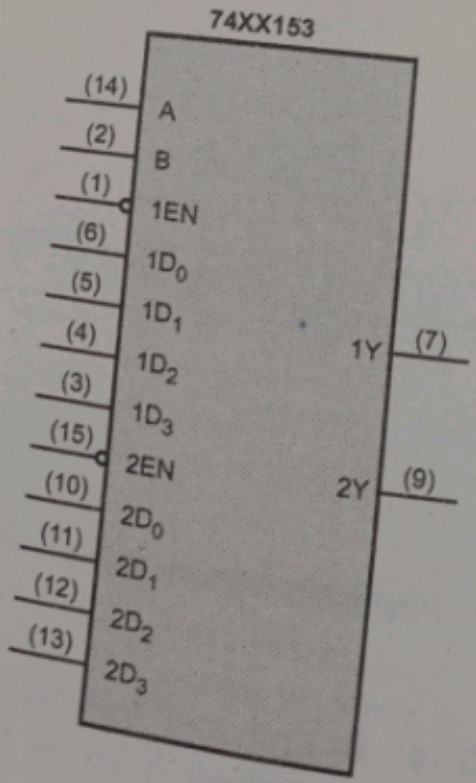


Fig. Q.11.1 Logic symbol for 74XX153

The 74XX153 is a dual 4 to 1 multiplexer. Fig. Q.11.1 shows the logic symbol for 74XX153. It contains two identical and independent 4-to-1 multiplexers. Each multiplexer has separate enable inputs. The Table Q.11.1 shows the truth table for 74XX153.

5.2 : Decoder / Demultiplexer (IC 74138, IC 74238) and Implementation of Logic Function using IC 74138

Q.12 What is decoder ? Draw and explain a 2 to 4 line decoder.

Ans. : A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different.

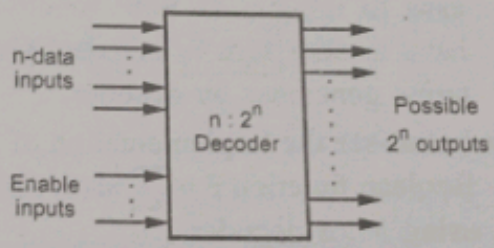


Fig. Q.12.1 General structure of decoder

- Fig. Q.12.2 shows 2-to-4 decoder. 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables.

Inputs			Outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_0
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

Table Q.12.1 Truth table for a 2 to 4 decoder

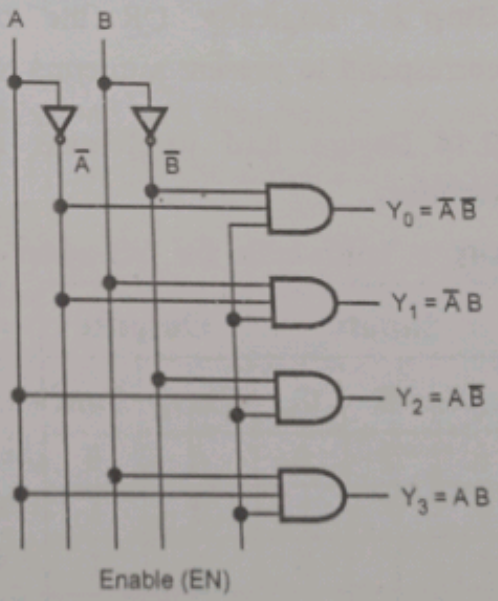


Fig. Q.12.2 2 to 4 line decoder

- The two inverters provide the complement of the inputs, and each one of four AND gates generates one of the minterms.
- The Table Q.12.1 shows the truth table for a 2-to-4 decoder.
- If enable input is 1 (EN = 1), one, and only one, of the outputs Y_0 to Y_3, is active for a given input.

- The output Y_0 is active, i.e. $Y_0 = 1$ when inputs A = B = 0, the output Y_1 is active when inputs A = 0 and B = 1.
- If enable input is 0, i.e. EN = 0, then all the outputs are 0.

Q.13 Explain the procedure to implement combination circuit using decoder with the help of example.

Ans. : • The combination of decoder and external logic gates can be used to implement single or multiple output functions.

- When decoder output is active high, it generates minterms (product terms) for input variables; i.e. it makes selected output logic 1. In such case to implement SOP function we have to take sum of selected product terms generated by decoder.
- Let us see the Implementation of Boolean function $F = \sum m(1, 2, 3, 7)$ using 3 : 8 decoder.

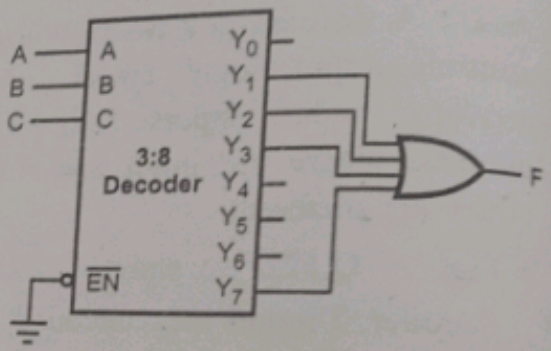


Fig. Q.13.1

Step 1 : Connect function variables as inputs to the decoder.

Step 2 : Logically OR the outputs correspond to present minterms to obtain the output.

Q.14 Design and implement a full adder circuit using a 3 : 8 decoder.

☞ [SPPU : May-10, Dec.-10,16 Marks 4]

Ans. : Truth table for full adder is as shown in the Table Q.14.1.

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table Q.14.1 Truth table for full-adder

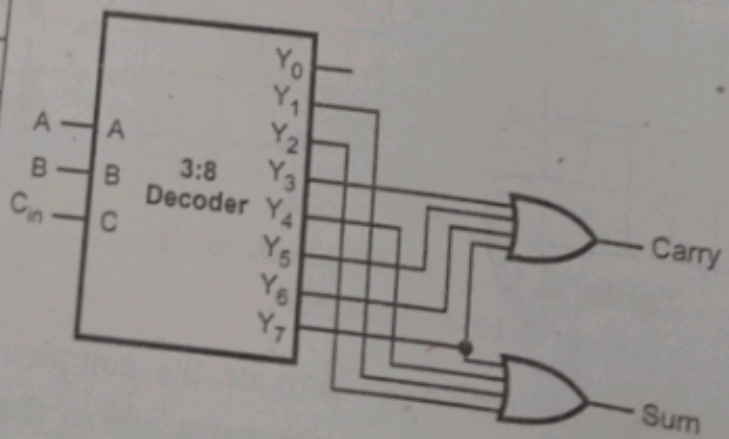


Fig. Q.14.1

Q.15 Design a 4 × 16 decoder using 3 × 8 decoders.

Ans. : • Fig. Q.15.1 shows the 4 × 16 decoder using two 3 × 8 decoders.

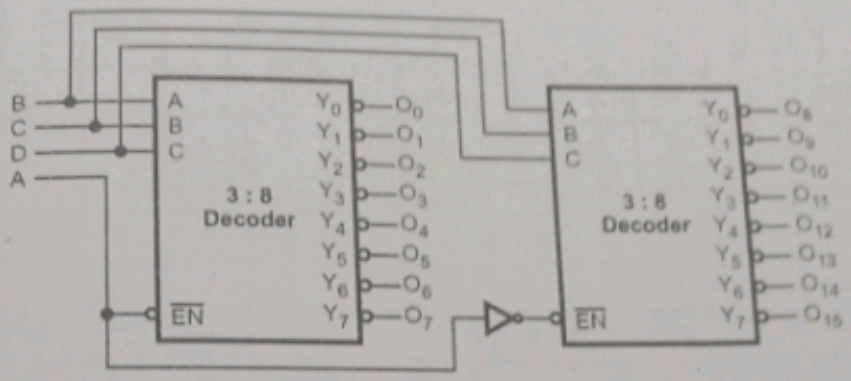


Fig. Q.15.1

- Here, one input line (D) is used to enable/disable the decoders.
- When D = 0, the top decoder is enabled and the other is disabled. Thus the bottom decoder outputs are all 1s, and the top eight outputs generate minterms 0 0 0 0 to 0 1 1 1.
- When D=1, the enable conditions are reversed and thus bottom decoder outputs generate minterms 1000 to 1Y111, while the outputs of the top decoder are all 1s.

Q.16 Write a note on IC decoder 74138.

Ans. : The 74X138 is a commercially available 3-to-8 decoder. It accepts three binary inputs (A, B, C) and when enabled, provides eight individual active low outputs (Y_0 - Y_7). The device has three enable inputs : two active low ($\overline{G}_{2A}$, $\overline{G}_{2B}$) and one active high (G_1). Fig. Q.16.1 and Table Q.16.1 show logic symbol and function table respectively.

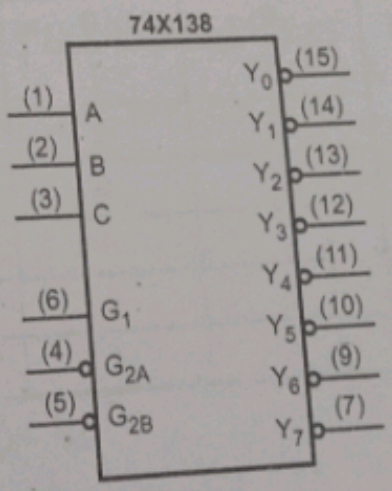


Fig. Q.16.1 Logic symbol

Inputs						Outputs							
G_{2B}	G_{2A}	G_1	C	B	A	$\bar{Y}_7$	$\bar{Y}_6$	$\bar{Y}_5$	$\bar{Y}_4$	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1

Table Q.16.1 Function table

Q.17 Design a 3-bit binary to 3-bit gray code converter using IC-74138.

☞ [SPPU : Dec.-13, Marks 4]

Ans. : Truth table

Inputs			Outputs		
B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Implementation

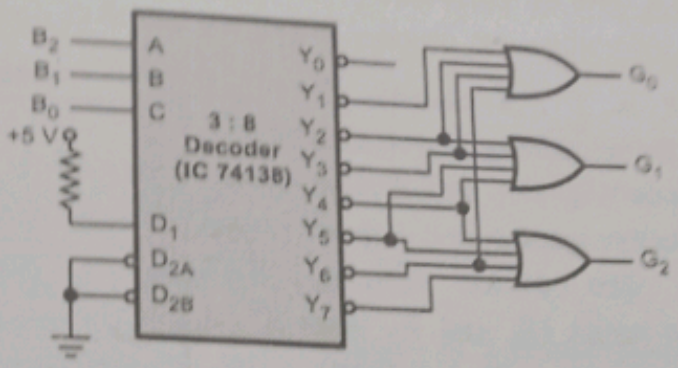


Fig. Q.17.1

Q.18 Design multiple output functions using IC 74138 and logic gates : $F_1 (A, B, C) = \pi M(0, 1, 3, 7)$ and $F_2 (A, B, C) = \pi M(2, 3, 7)$.

☞ [SPPU : May-16, Marks 6]

Ans. :

$\pi M\ (0,\ 1,\ 3,\ 7) = \Sigma m\ (2,\ 4,\ 5,\ 6)$

and

$\pi M\ (2,\ 3,\ 7) = \Sigma m\ (0,\ 1,\ 4,\ 5,\ 6)$

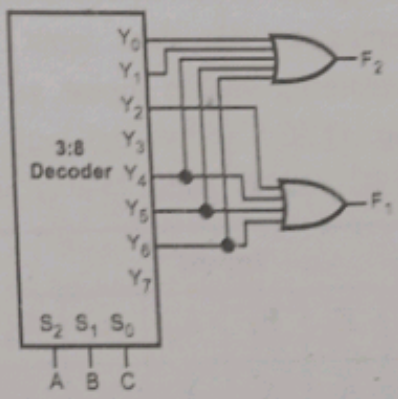


Fig. Q.18.1

Q.19 Design full subtractor using decoder IC 74138.

☞ [SPPU : Dec.-15, Marks 6]

Ans. : Refer Q.11 of Chapter 4.

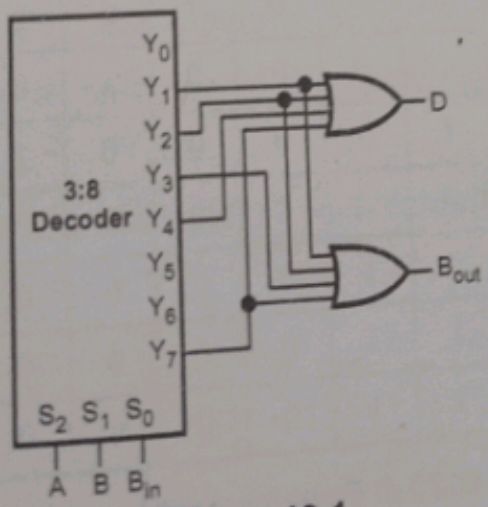


Fig. Q.19.1

Q.20 Write a note on IC 74238.

Ans. :

- The 74238 is 3 : 8 decoder. It decodes three binary weighted address inputs (A_0, A_1 and A_2) to eight mutually exclusive outputs ($Y_0 + Y_7$). The device features three enable inputs ($\overline{E}_1$ and $\overline{E}_2$ and E_3). Every output will be LOW unless $\overline{E}_1$ and $\overline{E}_2$ are LOW and E_3 is HIGH. The 74238 can be used as an eight output de-multiplexer by using one of the active LOW enable inputs as the data input and the remaining enable inputs as strobes. Fig. Q.20.1 shows the logic symbol of IC 74238. Table Q.20.1 shows function table of IC 74238.

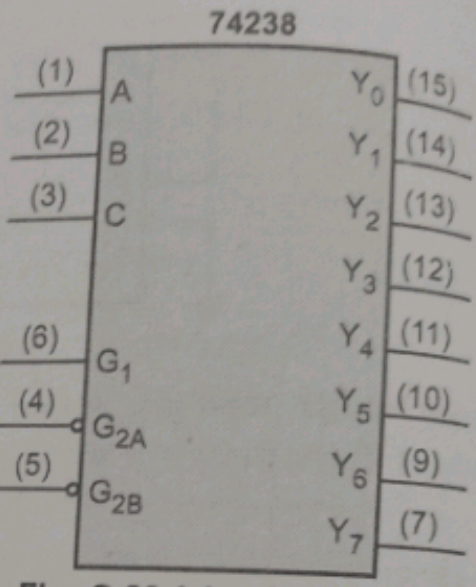


Fig. Q.20.1 Logic symbol of 74238

Inputs						Outputs							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_2	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	X	X	X	X	X	0	0	0	0	0	0	0	0
X	1	X	X	X	X	0	0	0	0	0	0	0	0
X	X	0	X	X	X	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0	0	0	1	0	0	0
0	0	1	1	0	1	0	0	0	0	0	1	0	0
0	0	1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	1	1	0	0	0	0	0	0	0	1

Table Q.20.1 Function table of IC 74238

Note : Important difference between 74138 and 74238 is that for 74138 outputs are active LOW and for 74238 outputs are active high.

Q.21 What is de-multiplexer ?

Ans. : A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines.

- The selection of specific output line is controlled by the values of n selection lines.
- The Fig. Q.21.1 shows the block diagram of a demultiplexer. It has one input data line, 2^n output lines, n select lines and one enable input.

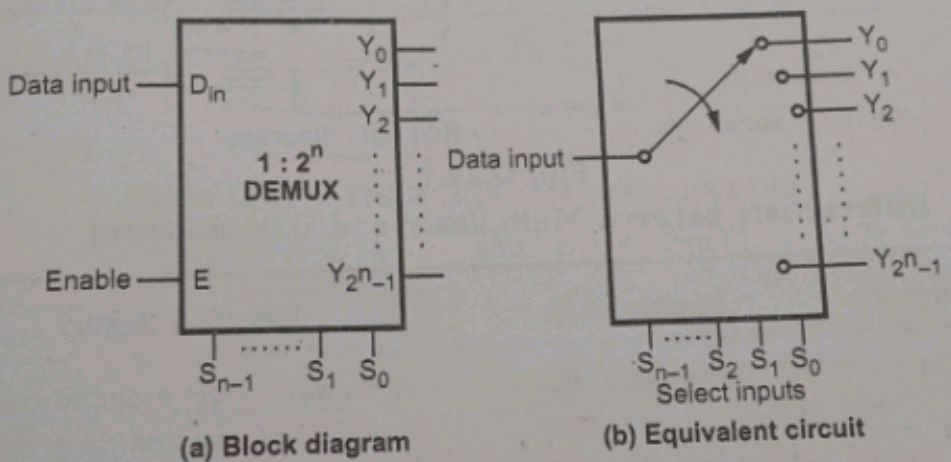


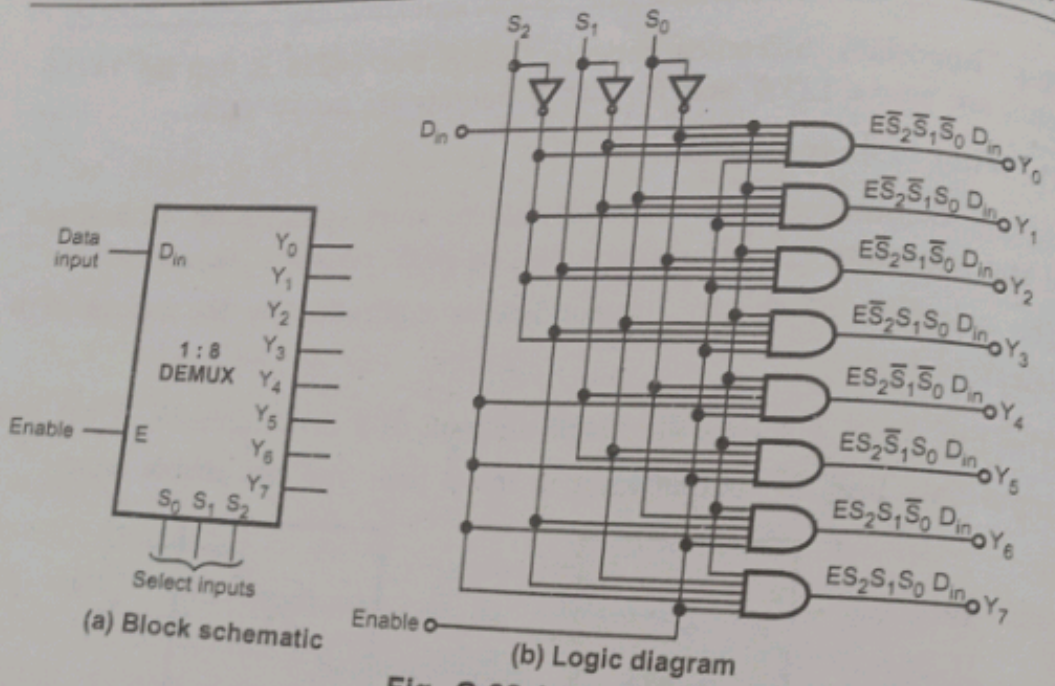
Fig. Q.21.1

Q.22 Draw and explain the logic diagram of one line to 8 - line demultiplexer.

Ans. : The Fig. Q.22.1 shows 1 : 8 demultiplexer. (Refer Fig. Q.22.1 on next page)

- The single input data D_{in} has a path to all eight outputs, but the input information is directed to only one of the output lines depending on the select inputs.

Q.23 Differentiate between multiplexer and demultiplexer.



(a) Block schematic (b) Logic diagram

Fig. Q.22.1

Ans. : Differentiate between Multiplexer and Demultiplexer

Parameter	Multiplexer	Demultiplexer
Definition	Multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line.	Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines
Number of data inputs	2^n	1
Number of data outputs	1	2^n
Relationship of input and output	Many to one	One to many
Applications	• Used as a data selector • In time division multiplexing at the transmitting end	• Used as a data distributor • In time division multiplexing at the receiving end

Q.24 Implement the full subtractor using a 1 : 8 demultiplexer.

Ans. : Table shows the truth table for full subtractor.

Inputs			Outputs	
A	B	B_{in}	D	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

Table Q.24.1 Truth table for full-subtractor

$\therefore$ $D = \sum m\,(1, 2, 4, 7)$ and $B_{out} = \sum m\,(1, 2, 3, 7)$

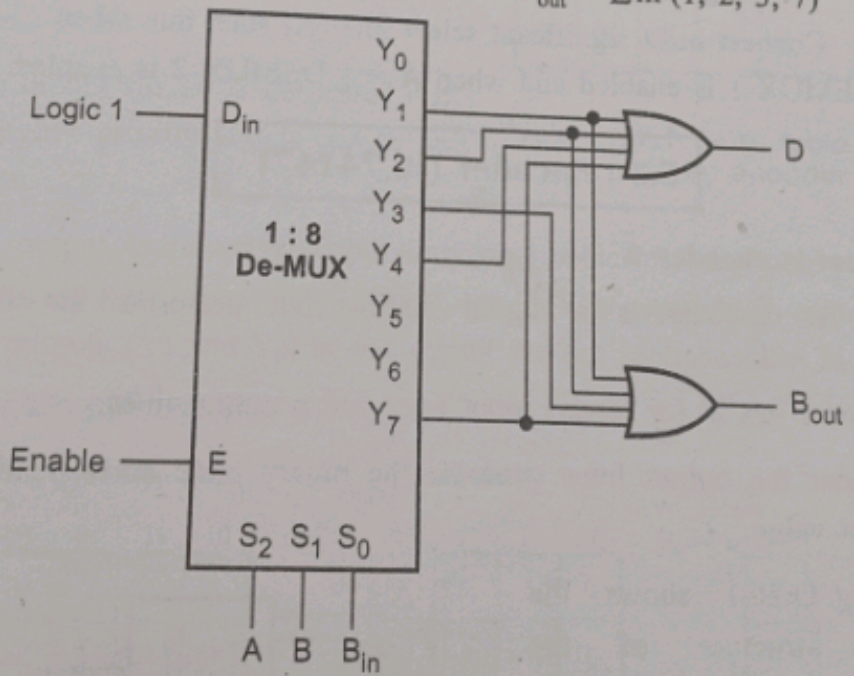


Fig. Q.24.1 Implementation

Q.25 Design 1 : 8 demultiplexer using two 1 : 4 demultiplexers.

Ans. : Step 1 : Connect D_{in} signal to D_{in} input of both the demultiplexers.

Step 2 : Connect select lines B and C to select lines S_1 and S_0 of the both demultiplexers, respectively.

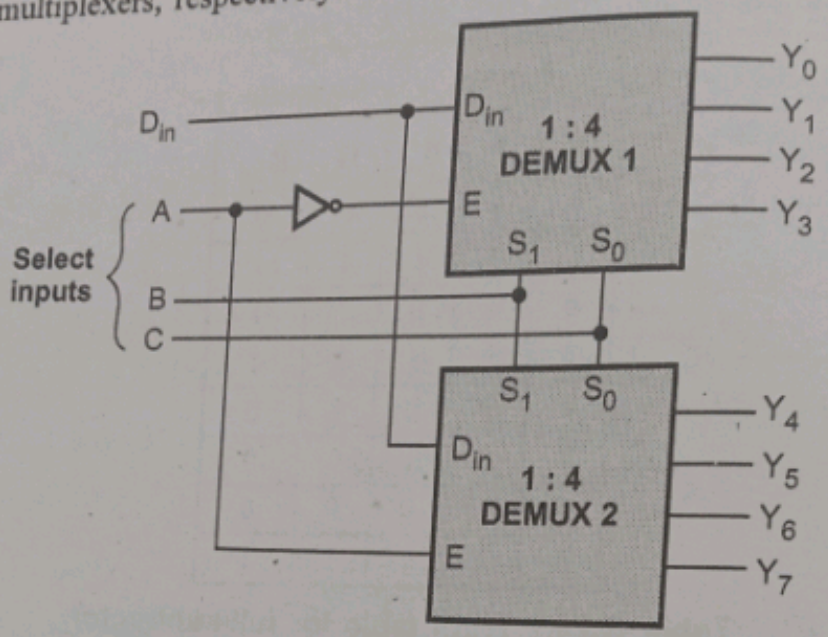


Fig. Q.25.1 Cascading of demultiplexers

Step 3 : Connect most significant select line (A) such that when A = 0 DEMUX 1 is enabled and when A = 1 DEMUX 2 is enabled.

5.3 : Encoder (IC 74147)

Q.26 What is encoder ?

Ans. : • An encoder is a digital circuit that performs the inverse operation of a decoder.

• An encoder has 2^n (or fewer) input lines and n output lines.

• In encoder the output lines generate the binary code corresponding to the input value.

• The Fig. Q.26.1 shows the general structure of the encoder circuit. As shown in the Fig. Q.26.1 the decoded information is presented as 2^n inputs producing n possible outputs.

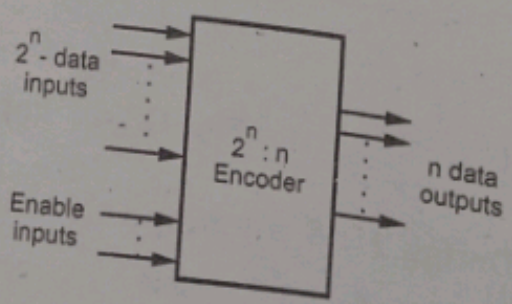


Fig. Q.26.1 General structure of encoder

Q.27 What is priority encoder ? Explain with the help of suitable example.

Ans. : • 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.

- Table Q.27.1 shows truth table of 4-bit priority encoder.
- Table Q.27.1 shows D_3 input with highest priority and D_0 input with lowest priority. When D_3 input is high, regardless of other inputs output is (Y_1 Y_0 = 11) 11.
- The D_2 has the next priority. Thus, when $D_3 = 0$ and $D_2 = 1$, regardless of other two lower priority input, output is 10.
- The output for D_1 is generated only if higher priority inputs are 0, and so on.

Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	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

Table Q.27.1 Truth table of 4-bit priority encoder

- The output V (a valid output indicator) indicates, one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs (Y_1 and Y_0) of the circuit are not used.

K-map simplification

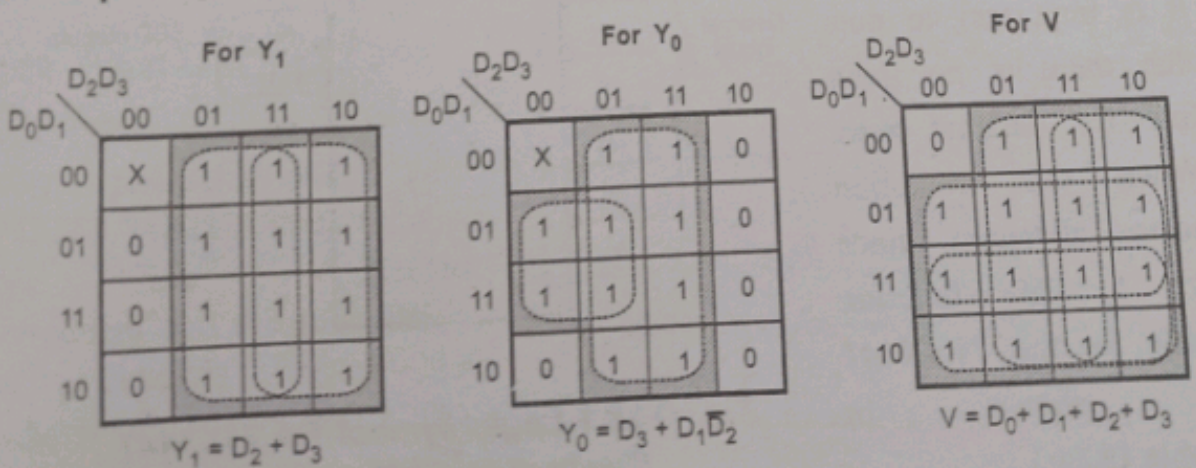


Fig. Q.27.1

Logic diagram

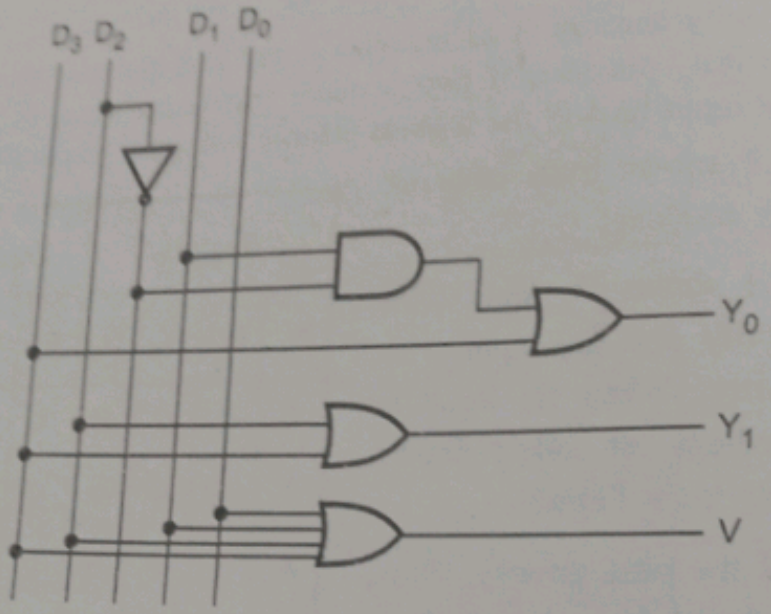


Fig. Q.27.2

Q.28 Write a note on IC 74147.

Ans. : The Fig. Q.28.1 shows the logic symbol for decimal to BCD encoder IC, IC 74XX147. It has nine input lines and four output lines. Both input and output lines are asserted active low. It is important to note that there is no input line for decimal zero. When this condition occurs, all output lines are 1. The function table for the 74XX147 is shown in Table Q.28.1.

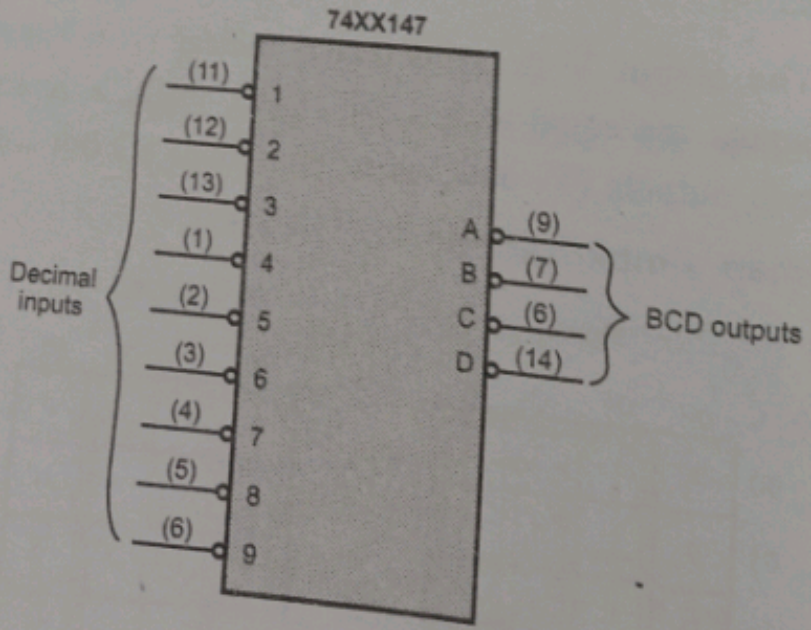


Fig. Q.28.1 Logic symbol for 74XX147 (Decimal to BCD encoder)

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

x indicates don't care condition

Table Q.28.1 Truth table for decimal to BCD encoder

5.4 : Binary Adder

Q.29 What is IC 7483 ?

Ans. : The most common is a 4-bit parallel adder IC (74LS83/74S283) that contains four inter connected full-adders and the look-ahead carry circuitry needed for high-speed operation. The 7483 and 74283 are a TTL Medium Scale Integrated (MSI) circuit with same pin

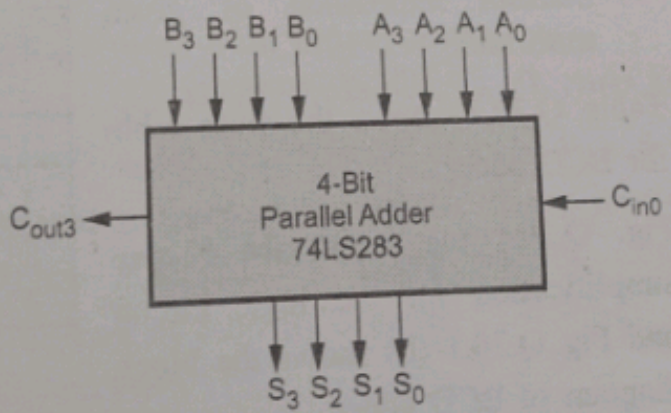


Fig. Q.14.1 Functional symbol for the 74LS283

configuration. Fig. Q.29.1 shows the functional symbol for the 74LS283 4-bit parallel adder. The inputs to this IC are two 4-bit numbers, A_3 A_2 A_1 A_0 and B_3 B_2 B_1 B_0, and the carry, C_{in0}, into the LSB position. The outputs are the sum bits S_3 S_2 S_1 S_0, and the carry, C_{out3}, output of the MSB position.

Two or more parallel adder blocks can be connected (cascaded) to accommodate the addition of larger binary numbers.

5.5 : BCD Adder

Q.30 What is BCD / Decimal adder ? Design one digit BCD / Decimal adder. ☞ [SPPU : May-05, 12, Dec.-05, 12, 14, Marks 6]

Ans. : • A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD.

- To implement BCD adder we require :
  - 4-bit binary adder for initial addition
  - Logic circuit to detect sum greater than 9 and
  - One more 4-bit adder to add 0110_2 in the sum if sum is greater than 9 or carry is 1.
- The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.
- Table Q.30.1 shows the truth table for BCD adder.
- Fig. Q.30.1 (a) shows the K-map simplification for the logic circuit and Fig. Q.30.1 (b) shows the block diagram of BCD adder.

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

Table Q.30.1

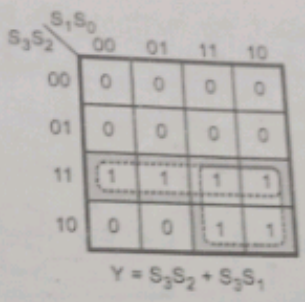


Fig. Q.30.1 (a) K-map simplification

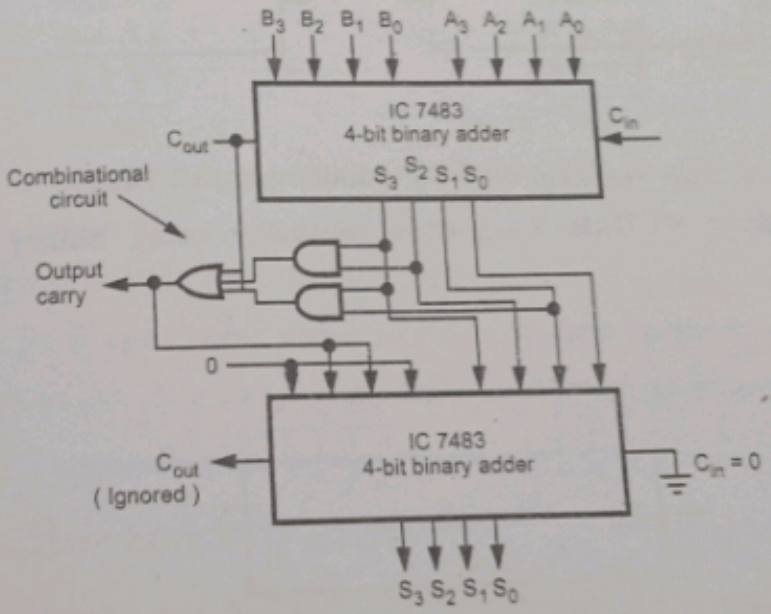


Fig. Q.30.1 (b) Block diagram of BCD adder

Q.31 Design an 8-bit BCD adder using 4-bit binary adder.

☞ **[SPPU : May-05, 10, 12, Marks 8]**

Ans. : To implement 8-bit BCD adder we have to cascade two 4-bit BCD adders. In cascade connection carry output of the lower position (digit) is connected as a carry input of the higher position (digit). Fig. Q.31.1 shows the block diagram of 8-bit BCD adder.

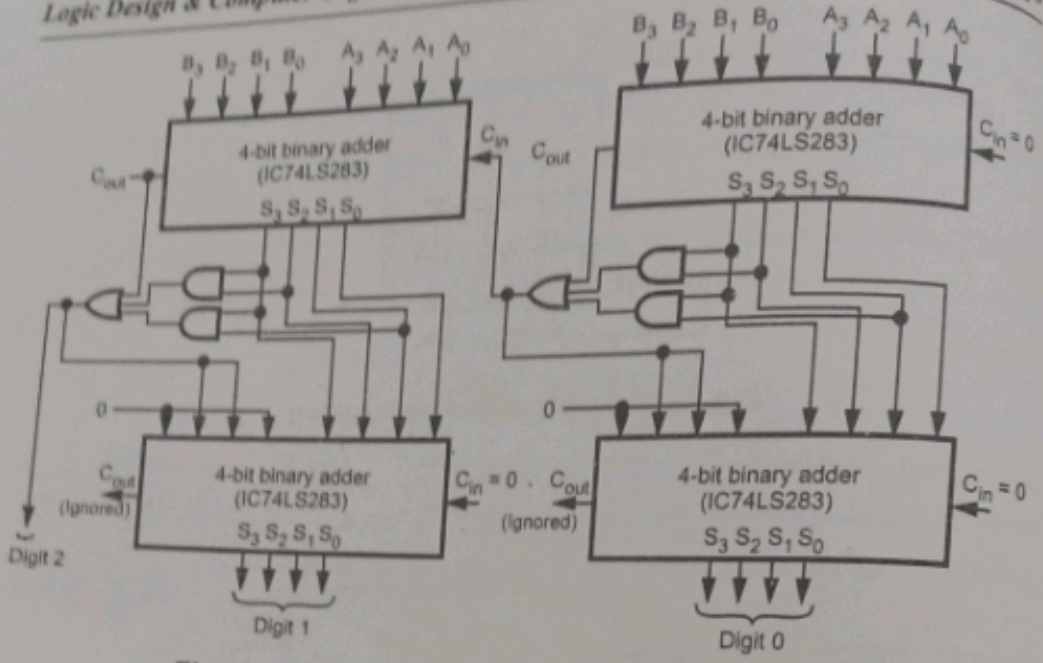


Fig. Q.31.1 8-bit BCD adder using IC 74283

Q.32 Design a BCD to Ex-3 code converter using binary parallel adder.

Ans. :

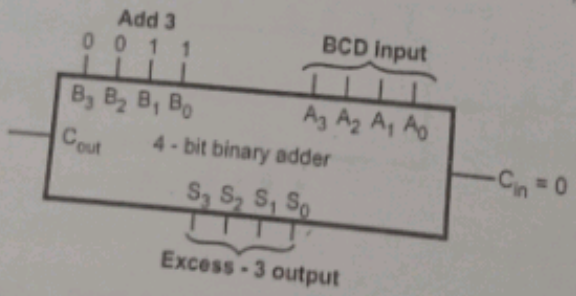


Fig. Q.32.1

5.6 : BCD Subtractor using IC 7483

Q.33 Design BCD subtractor using 9's complement method.

Ans. : The steps for 9's complement BCD subtraction as follows :

- Find the 9's complement of a negative number
- Add two numbers using BCD addition
- If carry is generated add carry to the result otherwise find the 9's complement of the result.

Fig. Q.33.1 shows the logic diagram of the circuit to implement above mentioned steps to perform BCD subtraction using 9's complement method. As shown in the Fig. Q.33.1, first binary adder finds the 9's complement of the negative number. It does this by inverting each bit of BCD number and adding 10 ($1\ 0\ 1\ 0_2$) to it. Let us find the 9's complement of 2. (Refer Fig. Q.33.1 on next page)

		0	0	1	0	← BCD for 2
	1	1	0	1		← Inverting each bit
+	1	0	1	0		← Add 10 (1010_2)
Ignore carry →	1	0	1	1	1	← 9's complement for 2

Next two 4-bit binary adders perform the BCD addition. The last adder finds the 9's complement of the result if carry is not generated after BCD addition otherwise it adds carry in the result.

Q.34 Design BCD subtractor using 10's complement method.

Ans. : The steps for 10's complement BCD subtraction as follows.

- Find the 10's complement of a negative number
- Add two numbers using BCD addition
- If carry is not generated find the 10's complement of the result.

Fig. Q.34.1 shows the logic diagram of the circuit to implement above mentioned steps to perform BCD subtraction using 10's complement method. As shown in the Fig. Q.33.1, first binary adder finds the 10's complement of the negative number (9's complement + 1). Next two 4-bit binary adders perform the BCD addition. Finally, last 4-bit binary adder finds the 10's complement of the number if carry is not generated after BCD addition.

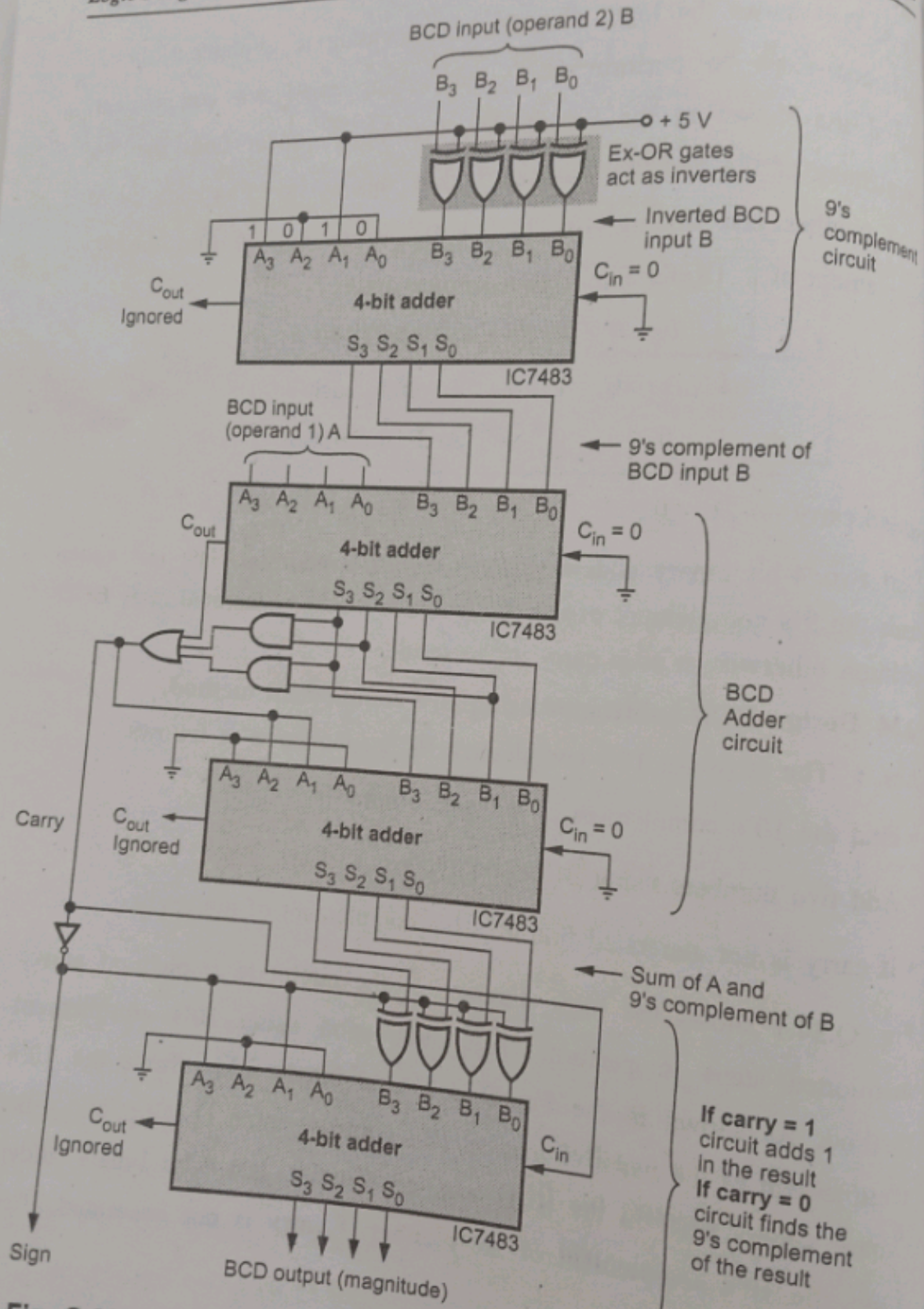


Fig. Q.33.1 4-bit BCD subtractor using 9's complement method

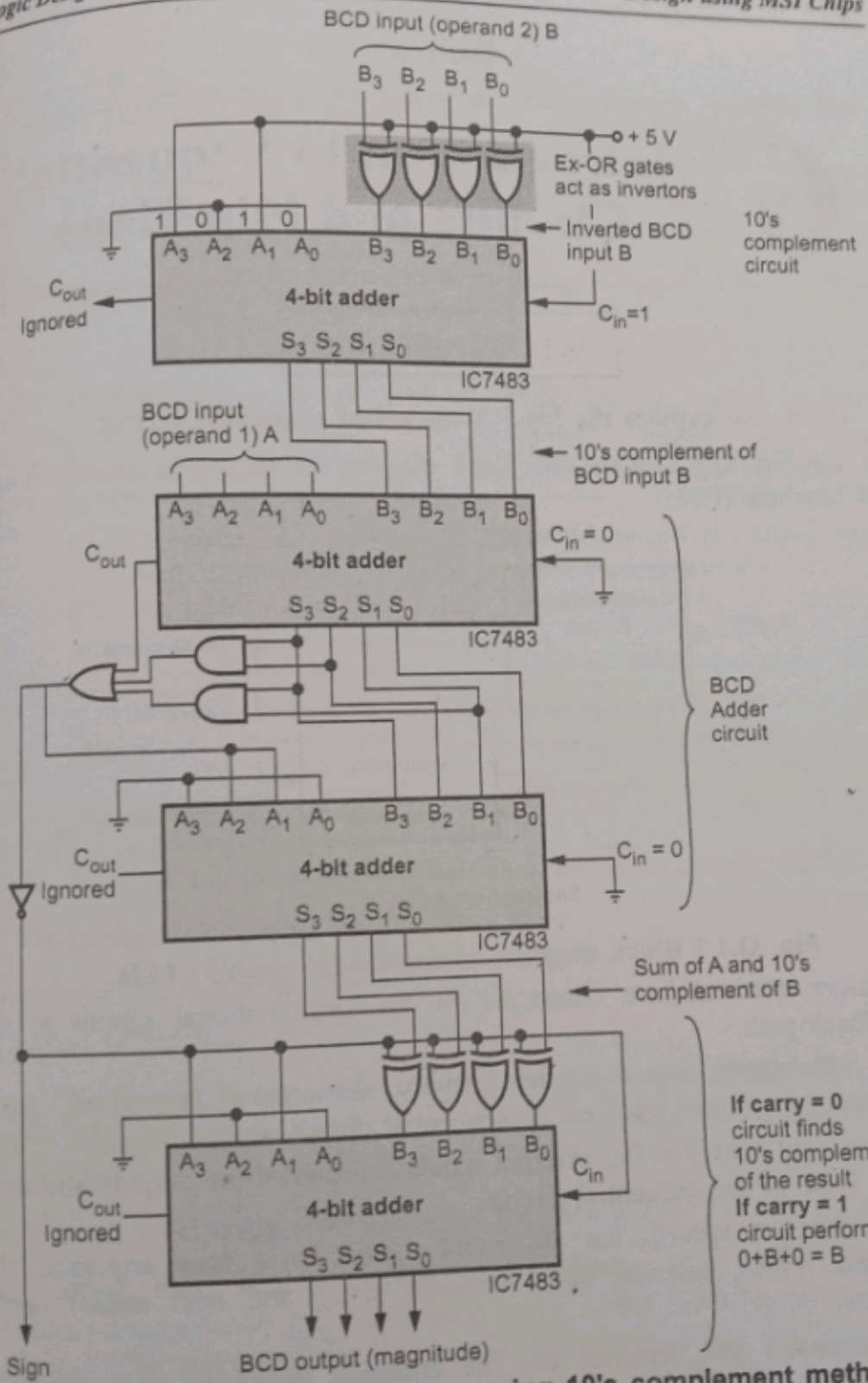


Fig. Q.34.1 4-bit BCD subtractor using 10's complement method

END...