

CS8351- DIGITAL PRINCIPLES AND SYSTEM DESIGN UNIT II COMBINATIONAL CIRCUIT DESIGN

Design of half and full adders, half and full subtractors, Binary parallel adder-carry look ahead adder, BCD Adder, multiplexer, Demultiplexer, magnitude comparator, Decoder, Encoder, priority encoder.

Combinational Circuits

Q1a) Define combinational circuit

(or)

b)What is combinational circuit?

A *combinational circuit* is one where the output at any time depends only on 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.

Figure shows the block schematic representation of a generalized combinational circuit having n input variables and m output variables or simply outputs. Since the number of input variables is n , there are 2^n possible combinations of bits at the input.



Figure: Generalized combinational circuit.

Design procedure:

Q2a)Write the Design procedure in combinational logic circuit?

(or)

b)Explain the steps involved in the design of a combinational logic circuit

The different steps involved in the design of a combinational logic circuit are as follows:

1. Statement of the problem.
2. Identification of input and output variables.
3. Expressing the relationship between the input and output variables.
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. Minimization of Boolean expressions.

Arithmetic operations

In this 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. These include half-adder, full adder, half-subtractor, full subtractor and controlled inverter.

Q3a) Explain the circuit diagram of half adder

(or)

b) Draw and explain the block diagram of half adder

Half-Adder:

A *half-adder* is an arithmetic circuit block that can be used to add two bits. Such a circuit thus 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. Figure shows the truth table of a half-adder, showing all possible input combinations and the corresponding outputs.

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

$$\text{SUM } S = A\bar{B} + \bar{A}B$$

$$\text{CARRY } C = A.B$$

An examination of the two expressions tells that there is no scope for further simplification. While 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.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Figure: Truth table of a half-adder.

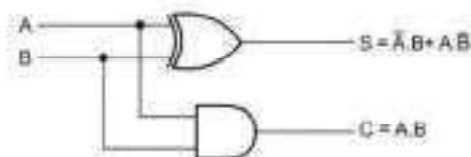


Figure: Logic implementation of a half-adder.

www.AllAbtEngg.com

Full Adder:

Q4a) Explain the circuit diagram of full adder

(or)

b) Draw and explain the block diagram of full adder

A *full adder* circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output.

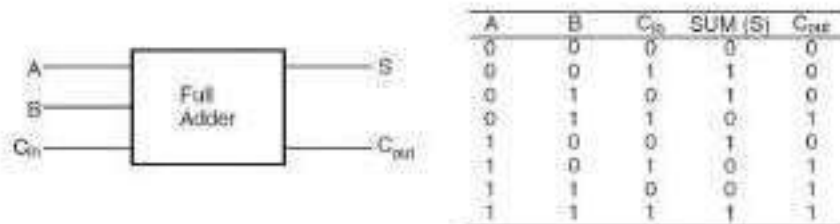


Figure: Truth table of a full adder:

Figure shows the truth table of a full adder circuit showing all possible input combinations and corresponding outputs. In order to arrive at the logic circuit for hardware implementation of a full adder, we will firstly write the Boolean expressions for the two output variables, that is, the SUM and CARRY outputs, in terms of input variables.

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + AB C_{in}$$
$$C_{out} = \bar{A}B C_{in} + A\bar{B}C_{in} + A\bar{B}\bar{C}_{in} + AB C_{in}$$

The next step is to simplify the two expressions. We will do so with the help of the Karnaugh mapping technique. Boolean expression for C_{out} is given out by the equation

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

A full adder can also be seen to comprise two half-adders and an OR gate. The expressions for SUM and CARRY outputs can be rewritten as follows:

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

www.AllAbtEngg.com

Similarly, the expression for CARRY output can be rewritten as follows:

$$\begin{aligned}
 C_{out} &= \bar{B}C_{in}(A+\bar{A}) + A\bar{B}C_{in} + A\bar{B}C_{in}(B+\bar{B}) \\
 &= A\bar{B} + A\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}C_{in} + A\bar{B}C_{in} = A\bar{B} + A\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}C_{in} \\
 &= A\bar{B}(1+C_{in}) + C_{in}(\bar{A}B + A\bar{B})
 \end{aligned}$$



Figure :Karnaugh maps for the sum and carry-out of a full adder.

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

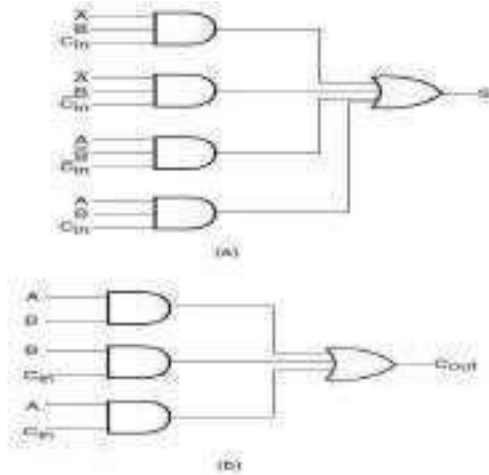


Figure :Logic circuit diagram of a full adder.

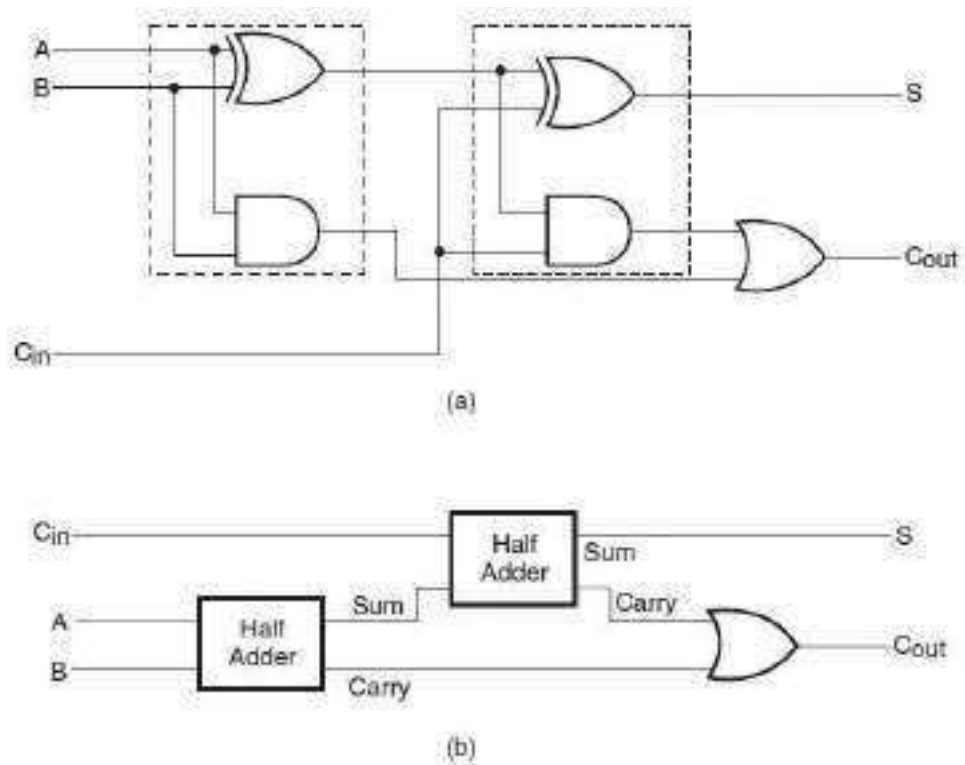


Figure :Logic implementation of a full adder with half-adders.

Half-Subtractor:

Q5a) Explain the circuit diagram of *Half-Subtractor*

(or)

b) Draw and explain the block diagram of *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. The truth table of a half-subtractor. The Boolean expressions for the two outputs are given by the equations,

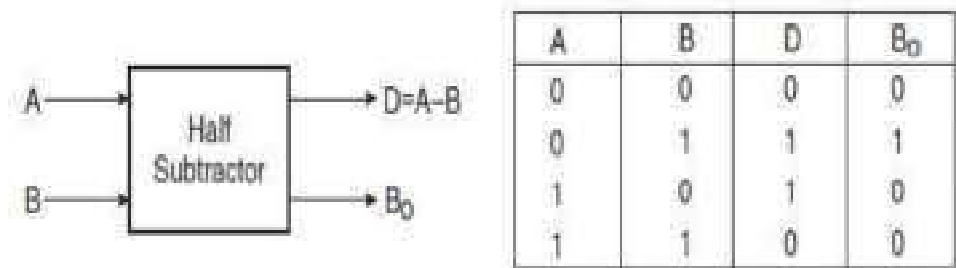


Figure Half-subtractor.

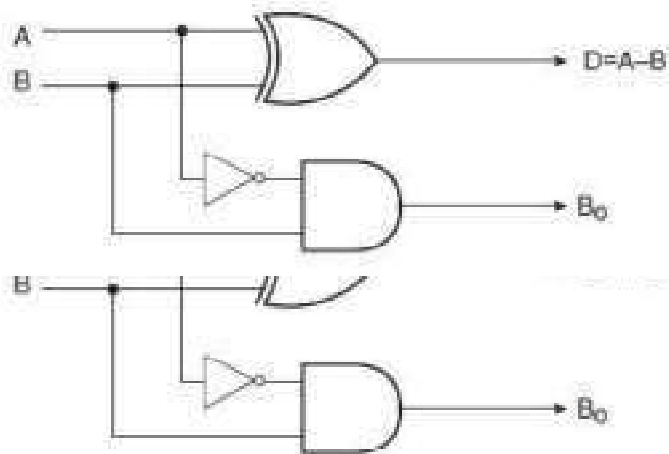


Figure Logic diagram of a half-subtractor.

Full Subtractor:

Q6a) Explain the circuit diagram of full-Subtractor

(or)

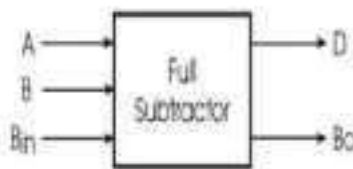
b) Draw and explain the block diagram of full-Subtractor

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 B₀. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.

The Boolean expressions for the two output variables are given by the equations

$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + AB B_{in}$$

$$B_o = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}B_{in} + AB B_{in}$$



Minuend (A)	Subtrahend (B)	Borrow In (B _{in})	Difference (D)	Borrow Out (B _o)
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

Figure: Truth table of a full subtractor.

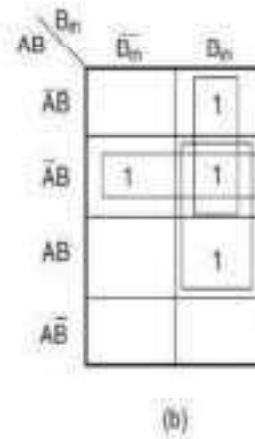
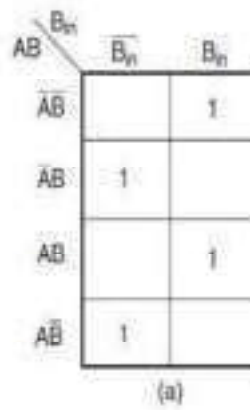


Figure:Karnaugh maps for difference and borrow outputs.

www.AllAbtEngg.com

The simplified expression for B_0 is given by the equation

$$B_0 = \bar{A}.B + \bar{A}.B_{in} + B.B_{in}$$

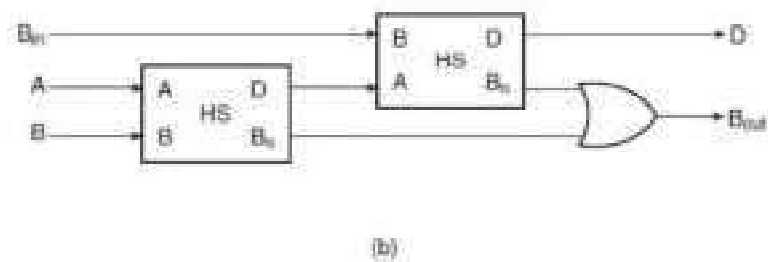
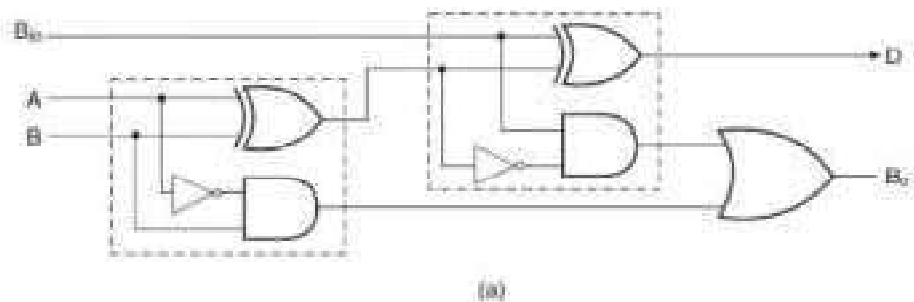


Figure: Logic implementation of a full subtractor with half-subtractors.

If we compare these expressions with those derived earlier in the case of a full adder, we find that the expression for DIFFERENCE output D is the same as that for the SUM output. Also, the expression for BORROW output B_0 is similar to the expression for CARRY-OUT C_0 . In the case of a half-subtractor, the A input is complemented. By a similar analysis it can be shown that a full subtractor can be implemented with half-subtractors in the same way as a full adder was constructed using half-adders.

Again, more than one full subtractor can be connected in cascade to perform subtraction on two larger binary numbers.

www.AllAbtEngg.com

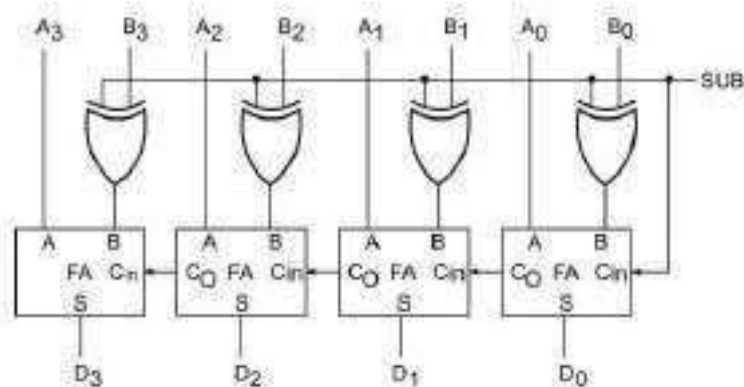
Parallel Adder-Subtractor:

Q7a) Design of a 4-bit Parallel Adder-Subtractor

(or)

b) Draw and explain the block diagram of 4-bit Parallel Adder-Subtractor

Subtraction of two binary numbers can be accomplished by adding 2's complement of the subtrahend to the minuend and disregarding the final carry, if any. If the MSB bit in the result of addition is



a 0, then the result of addition is the correct answer. If the MSB bit is a 1, this implies that the answer has a negative sign. The true magnitude in this case is given by 2's complement of the result of addition.

Full adders can be used to perform subtraction provided we have the necessary additional hardware to generate 2's complement of the subtrahend and disregard the final carry or overflow. Figure shows one such hardware arrangement. Let us see how it can be used to perform subtraction of two four-bit binary numbers. A close look at the diagram would reveal that it is the hardware arrangement for a four-bit binary adder, with the exception that the bits of one of the binary numbers are fed through controlled inverters. The control input here is referred to as the SUB input.

www.AllAbtEngg.com

When the SUB input is in logic '0' state, the four bits of the binary number ($B_3 B_2 B_1 B_0$) are passed on as such to the B inputs of the corresponding full adders. The outputs of the full adders in this case give the result of addition of the two numbers. When the SUB input is in logic '1' state, four bits of one of the numbers, ($B_3 B_2 B_1 B_0$) in the present case, get complemented. If the same '1' is also fed to the CARRY-IN of the LSB full adder, what we finally achieve is the addition of 2's complement and not 1's complement. Thus, in the adder arrangement of Fig., we are basically adding 2's complement of ($B_3 B_2 B_1 B_0$) to ($A_3 A_2 A_1 A_0$). The outputs of the full adders in this case give the result of subtraction of the two numbers. The arrangement shown achieves $A - B$. The final carry (the CARRY-OUT of the MSB full adder) is ignored if it is not displayed.

LOOK AHEAD CARRY ADDER:

Q8a) Design a 4 bit carry look ahead adder

(or)

b) Draw the logic diagram of look ahead carry adder

The four-bit binary adder described in the previously that can be used to add two four-bit binary numbers. Multiple numbers of such adders are used to perform addition operations on larger-bit binary numbers. Each of the adders is composed of four full adders (FAs) connected in cascade. This type of adder is also called a parallel binary adder because all the bits of the augend and addend are present and are fed to the full adder blocks simultaneously. Theoretically, the addition operation in various full adders takes place simultaneously. What is of importance and interest to users, more so when they are using a large number of such adders in their overall computation system, is whether the result of addition and carry-out are available to them at the same time. In other words, we need to see if this addition operation is truly parallel in nature. We will soon see that it is not. It is in fact limited by what is known as *carry propagation time*. Here, C_i and C_{i+1} are the input and output CARRY; P_i and G_i are two new binary variables called CARRY PROPAGATE and CARRY GENERATE and will be addressed a little later.

Consider a full adder circuit,

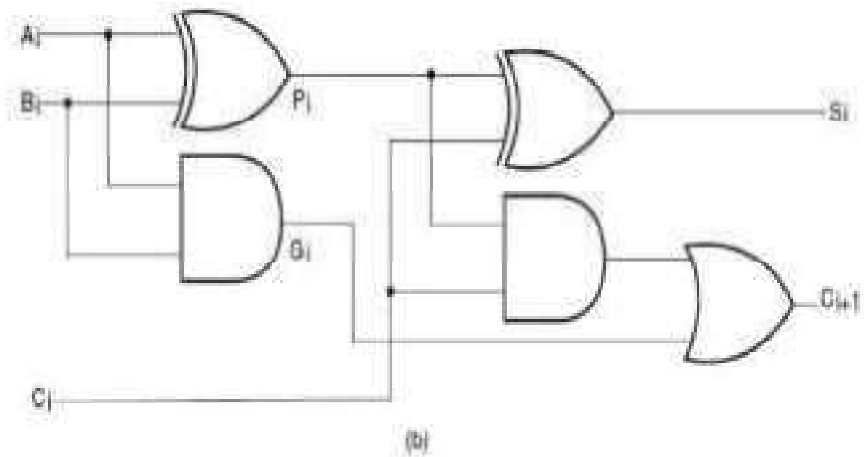
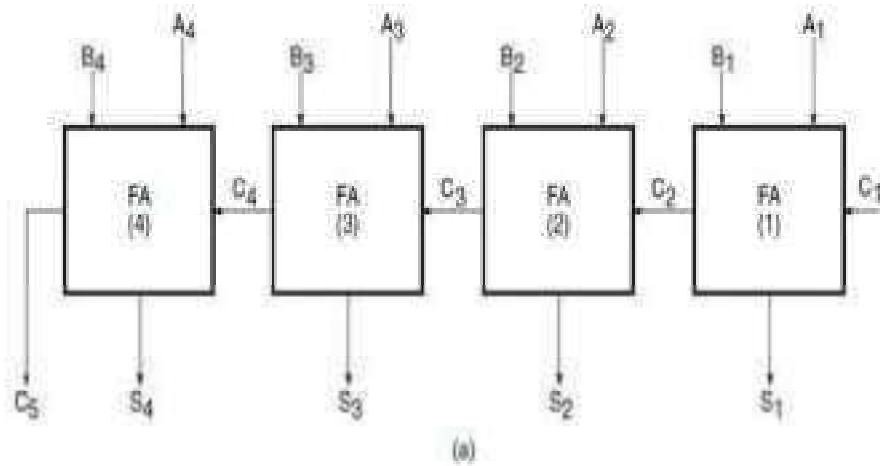


Figure: Four-bit binary adder.

www.AllAbtEngg.com

In order to explain the concept, let us define two new binary variables: P_i called CARRY PROPAGATE and G_i called CARRY GENERATE. Binary variable G_i is so called as it generates a carry whenever A_i and B_i are '1'. Binary variable P_i is called CARRY PROPAGATE as it is instrumental in propagation of C_i to C_{i+1} . CARRY, SUM, CARRY GENERATE and CARRY PROPAGATE parameters are given by the following expressions:

$$\begin{aligned}P_i &= A_i \oplus B_i \\G_i &= A_i \cdot B_i \\S_i &= P_i \oplus C_i \\C_{i+1} &= P_i \cdot C_i + G_i\end{aligned}$$

In the next step, we write Boolean expressions for the CARRY output of each full adder stage in the four-bit binary adder. We obtain the following expressions:

$$\begin{aligned}C_2 &= G_1 + P_1 \cdot C_1 \\C_3 &= G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot (G_1 + P_1 \cdot C_1) = G_2 + P_2 \cdot G_1 + P_1 \cdot P_2 \cdot C_1 \\C_4 &= G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_1 \cdot P_2 \cdot C_1) \\C_4 &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_1 \cdot P_2 \cdot P_3 \cdot C_1\end{aligned}$$

From the expressions for C_2 , C_3 and C_4 it is clear that C_4 need not wait for C_3 and C_2 to propagate. Similarly, C_3 does not wait for C_2 to propagate. Hardware implementation of these expressions gives us a kind of look-ahead carry generator. A look-ahead carry generator that implements the above expressions using AND-OR logic is shown in Fig.

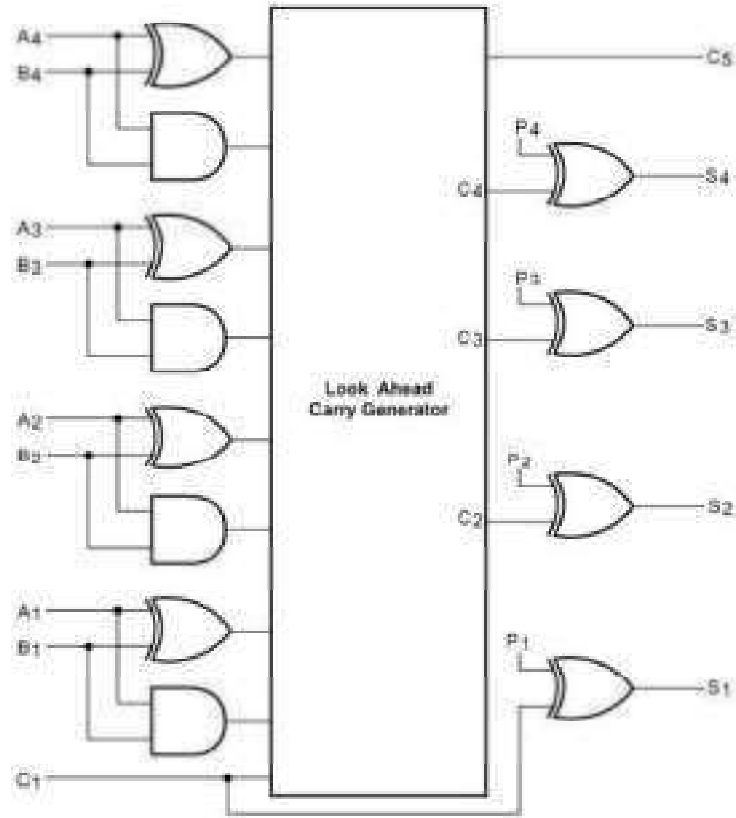


Figure: Look-ahead carry generator.

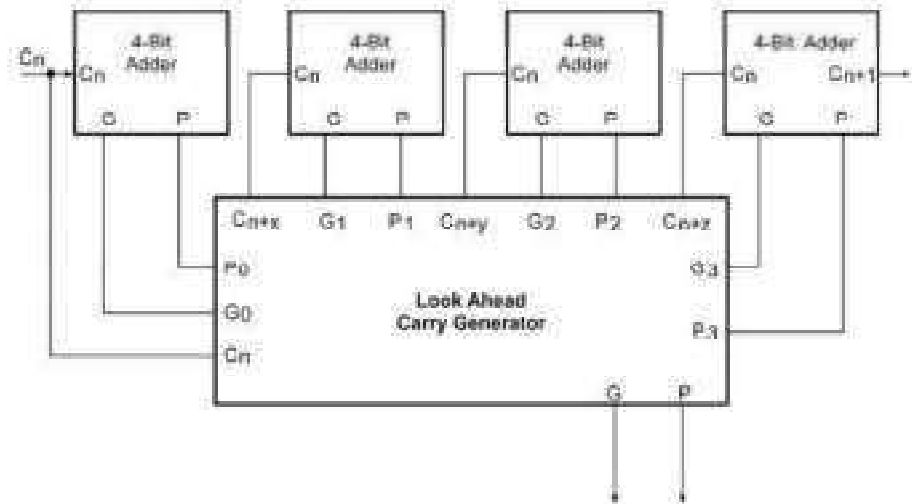


Figure IC 74182 interfaced with four four-bit adders.

Magnitude Comparator

Q9a) Design 4 bit comparator using gates

(or)

b) Draw the logic diagram of Magnitude Comparator

A *magnitude comparator* is a combinational circuit that compares two given numbers and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions $A = B$, $A > B$ and $A < B$, if A and B are the two numbers being compared. Depending upon the relative magnitude of the two numbers, the relevant output changes state. If the two numbers, let us say, are four-bit binary numbers and are designated as $(A_3 A_2 A_1 A_0)$ and $(B_3 B_2 B_1 B_0)$, the two numbers will be equal if all pairs of significant digits are equal, that is, $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. In order to determine whether A is greater than or less than B we inspect the relative magnitude of pairs of significant digits, starting from the most significant position. The comparison is done by successively comparing the next adjacent lower pair of digits if the digits of the pair under examination are equal. The comparison continues until a pair of unequal digits is reached. In the pair of unequal digits, if $A_i = 1$ and $B_i = 0$, then $A > B$, and if $A_i = 0$, $B_i = 1$ then $A < B$. If X , Y and Z are three variables respectively representing the $A = B$, $A > B$ and $A < B$ conditions, then the Boolean expression representing these conditions are given by the equations

$$X = x_3 \cdot x_2 \cdot x_1 \cdot x_0 \quad \text{where } x_i = A_i \cdot B_i + \overline{A_i} \cdot \overline{B_i}$$

$$Y = A_3 \cdot \overline{B_3} + x_3 \cdot A_2 \cdot \overline{B_2} + x_3 \cdot x_2 \cdot A_1 \cdot \overline{B_1} + x_3 \cdot x_2 \cdot x_1 \cdot A_0 \cdot \overline{B_0}$$

$$Z = \overline{A_3} \cdot B_3 + x_3 \cdot \overline{A_2} \cdot B_2 + x_3 \cdot x_2 \cdot \overline{A_1} \cdot B_1 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{A_0} \cdot B_0$$

Let us examine equation. x_3 will be '1' only when both A_3 and B_3 are equal. Similarly, conditions for x_2 , x_1 and x_0 to be '1' respectively are equal A_2 and B_2 , equal A_1 and B_1 and equal A_0 and B_0 . ANDing of x_3 , x_2 , x_1 and x_0 ensures that X will be '1' when x_3 , x_2 , x_1 and x_0 are in the logic '1' state. Thus, $X = 1$ means that $A = B$.

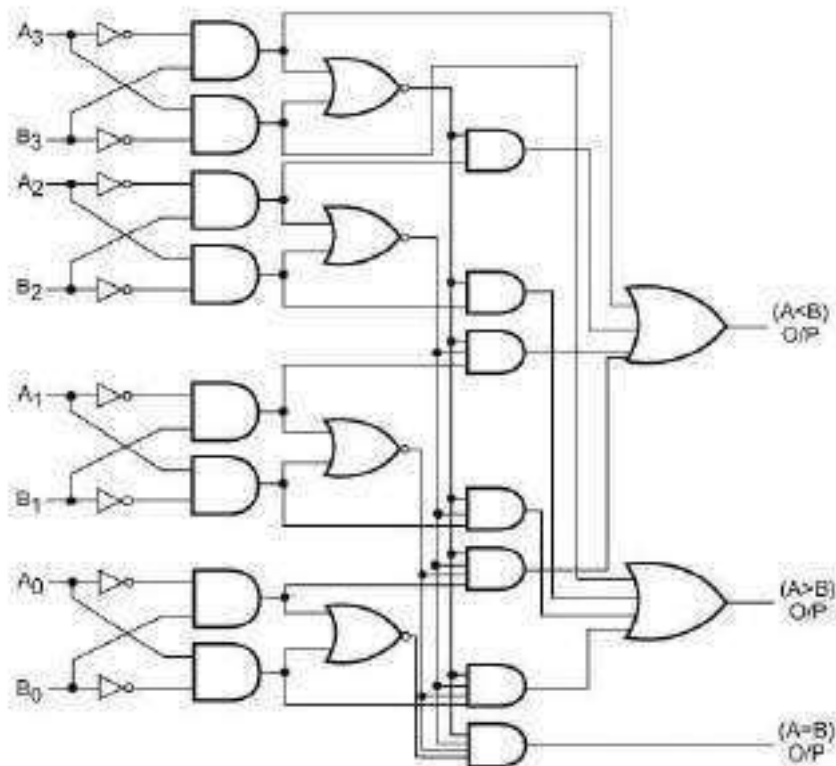


Figure: Four-bit magnitude comparator.

BCD Adder:

www.AllAbtEngg.com

Q10a) Design an 4bit BCD adder

(or)

b) Draw and explain 4- bit BCD adder using IC7483.

A *BCD adder* is used to perform the addition of BCD numbers. A BCD digit can have any of the ten possible four-bit binary representations, that is, 0000, 0001, ..., 1001, the equivalent of decimal numbers 0, 1, ..., 9. When we set out to add two BCD digits and we assume that there is an input carry too, the highest binary number that we can get is the equivalent of decimal number 19 ($9 + 9 + 1$).

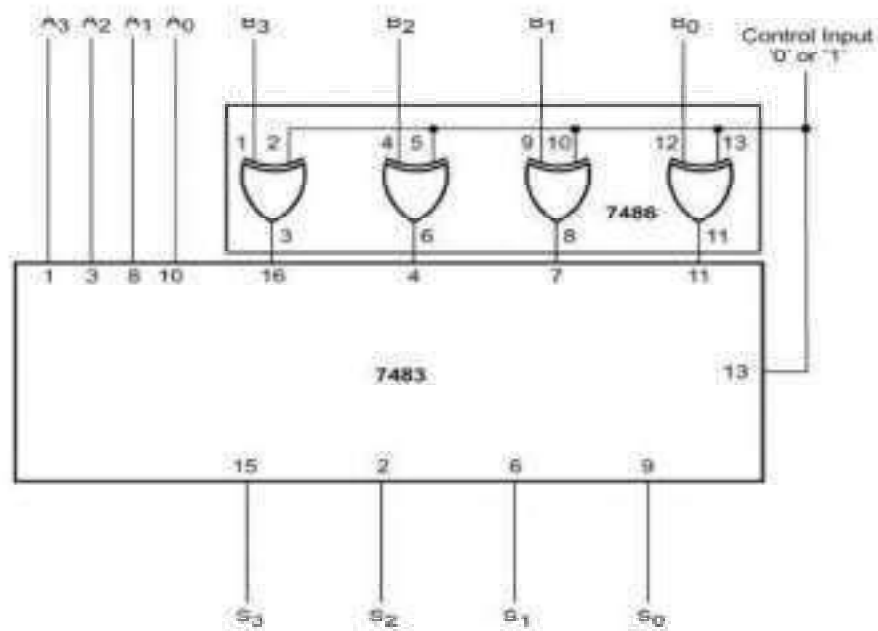


Figure: Four-bit adder-subtractor.

This binary number is going to be $(10011)_2$. On the other hand, if we do BCD addition, we would expect the answer to be $(0001\ 1001)_{BCD}$. And if we restrict the output bits to the minimum required, the answer in BCD would be $(1\ 1001)_{BCD}$. Table lists the possible results in binary and the expected results in BCD when we use a four-bit binary adder to perform the addition of two BCD digits. It is clear from the table that, as long as the sum of the two BCD digits remains equal to or less than 9, the four-bit adder produces the correct BCD output.

The binary sum and the BCD sum in this case are the same. It is only when the sum is greater than 9 that the two results are different. It can also be seen from the table that, for a decimal sum greater than 9 (or the equivalent binary sum greater than 1001), if we add 0110 to the binary sum, we can get the correct BCD sum and the desired carry output too. The Boolean expression that can apply the necessary correction is written as

$$C = K + Z_3.Z_2 + Z_3.Z_1$$

www.AllAbtEngg.com

Equation implies the following. A correction needs to be applied whenever $K = 1$. This takes care of the last four entries. Also, a correction needs to be applied whenever both Z_3 and Z_2 are '1'. This takes care of the next four entries from the bottom, corresponding to a decimal sum equal to 12, 13, 14 and 15. For the remaining two entries corresponding to a decimal sum equal to 10 and 11, a correction is applied for both Z_3 and Z_1 , being '1'. While hardware-implementing, 0110 can be added to the binary sum output with the help of a second four-bit binary adder. The correction logic as dictated by the Boolean expression should ensure that (0110) gets added only when the above expression is satisfied. Otherwise, the sum output of the first binary adder should be passed on as such to the final output, which can be accomplished by adding (0000) in the second adder. Figure shows the logic arrangement of a BCD adder capable of adding two BCD digits with the help of two four-bit binary adders and some additional combinational logic.

Decimal sum	Binary sum				BCD sum					
	K	Z_3	Z_2	Z_1	Z_0	C	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

www.AllAbtEngg.com

The BCD adder described in the preceding paragraphs can be used to add two single-digit BCD numbers only. However, a cascade arrangement of single-digit BCD adder hardware can be used to perform the addition of multiple-digit BCD numbers. For example, an n-digit BCD adder would require n such stages in cascade. As an illustration, Fig. shows the block diagram of a circuit for the addition of two three-digit BCD numbers. The first BCD adder, labelled LSD (Least Significant Digit), handles the least significant BCD digits. It produces the sum output ($S_3 S_2 S_1 S_0$), which is the BCD code for the least significant digit of the sum. It also produces an output carry that is fed as an input carry to the next higher adjacent BCD adder. This BCD adder produces the sum output ($S_7 S_6 S_5 S_4$), which is the BCD code for the second digit of the sum, and a carry output. This output carry serves as an input carry for the BCD adder representing the most significant digits. The sum outputs ($S_{11} S_{10} S_9 S_8$) represent the BCD code for the MSD of the sum.

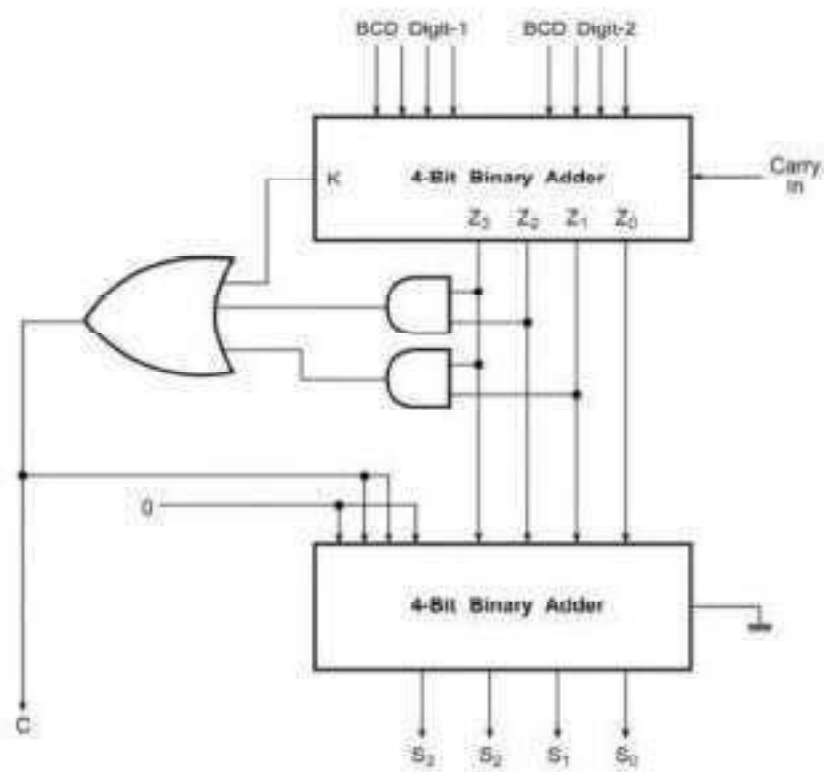


Figure Single-digit BCD adder.

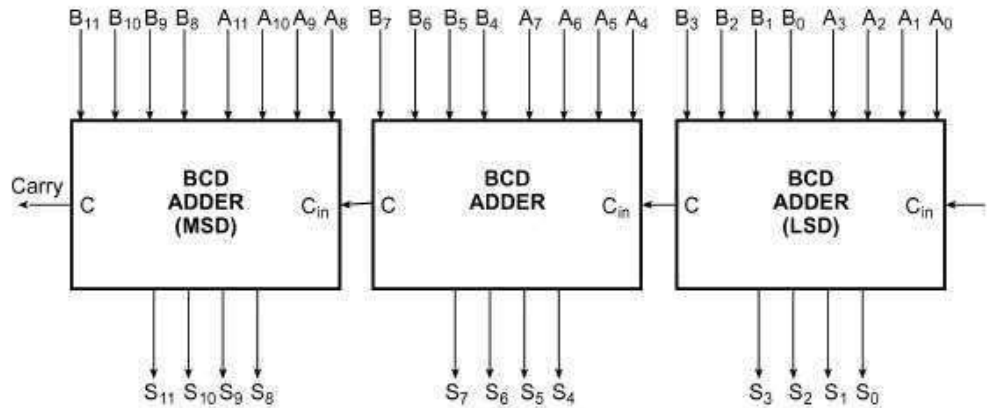


Figure Three-digit BCD adder.

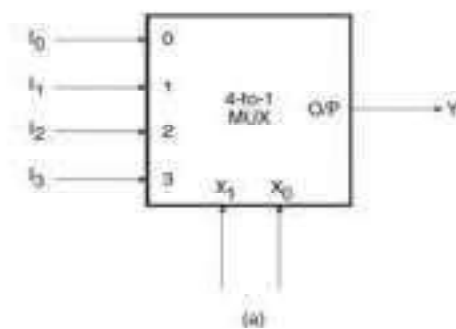
Multiplexer

Q11a) Draw the logic diagram of Multiplexer

(or)

b) Explain in detail about the multiplexers

A multiplexer or MUX, also called a *data selector*, is a combinational circuit with more than one input line, one output line and more than one selection line. There are some multiplexer ICs that provide complementary outputs. Also, multiplexers in IC form almost invariably have an ENABLE or STROBE input, which needs to be active for the multiplexer to be able to perform its intended function. A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line. If there are n selection lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n -to-1 multiplexer or $2^n \times 1$ multiplexer. Figures (a) and (b) respectively show the circuit representation and truth table of a basic 4-to-1 multiplexer.



X_1	X_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b)

Figure (a) 4-to-1 multiplexer circuit representation and (b) 4-to-1 multiplexer truth table

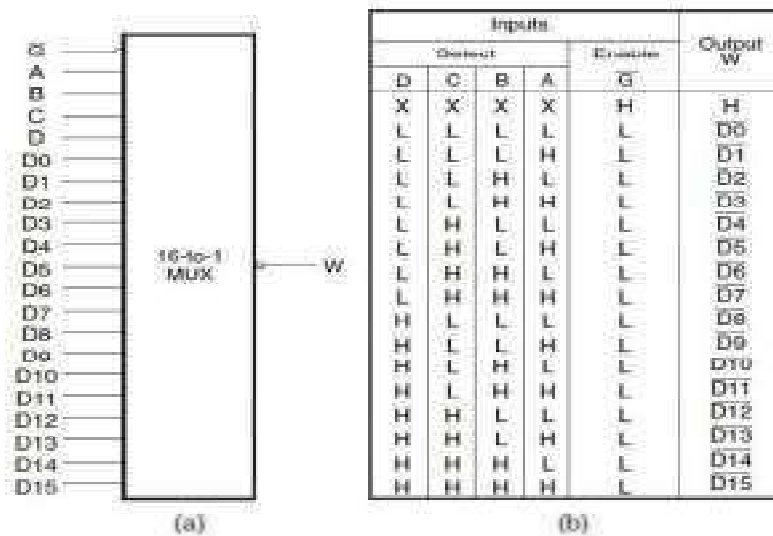


Figure (a) 16-to-1 multiplexer circuit representation and (b) 16-to-1 multiplexer truth table.

Inside the Multiplexer

We will briefly describe the type of combinational logic circuit found inside a multiplexer by considering the 2-to-1 multiplexer in Fig. (a), the functional table of which is shown in Fig.

(b). Figure (c) shows the possible logic diagram of this multiplexer. The circuit functions as follows:

- For $S = 0$, the Boolean expression for the output becomes $Y = I_0$.
- For $S = 1$, the Boolean expression for the output becomes $Y = I_1$.

Thus, inputs I_0 and I_1 are respectively switched to the output for $S = 0$ and $S = 1$. Extending the concept further, Fig. shows the logic diagram of a 4-to-1 multiplexer. The input combinations 00, 01, 10 and 11 on the select lines respectively switch I_0, I_1, I_2 and I_3 to the output.

$$Y = I_0 \overline{S_1} \overline{S_0} + I_1 \overline{S_1} S_0 + I_2 S_1 \overline{S_0} + I_3 S_1 S_0$$

$$Y = I_0 \overline{S_2} \overline{S_1} \overline{S_0} + I_1 \overline{S_2} \overline{S_1} S_0 + I_2 \overline{S_2} S_1 \overline{S_0} + I_3 \overline{S_2} S_1 S_0 + I_4 S_2 \overline{S_1} \overline{S_0} + I_5 S_2 \overline{S_1} S_0 + I_6 S_2 S_1 \overline{S_0} + I_7 S_2 S_1 S_0$$

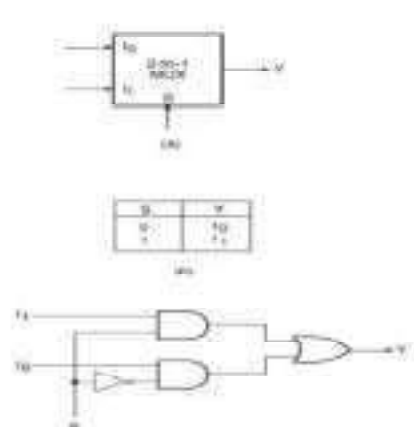


Figure (a) 2-to-1 multiplexer circuit representation, (b) 2-to-1 multiplexer truth table and (c) 2-to-1 multiplexer logic diagram.

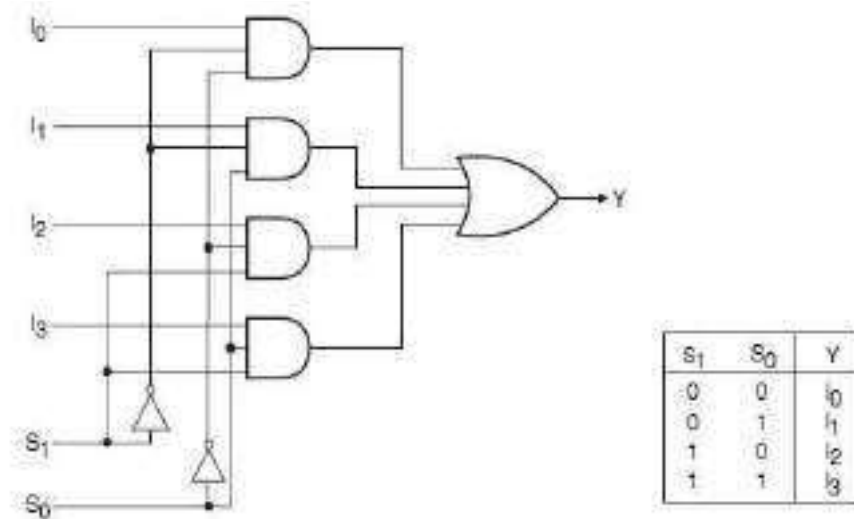


Figure Logic diagram of a 4-to-1 multiplexer.

As outlined earlier, multiplexers usually have an ENABLE input that can be used to control the multiplexing function. When this input is enabled, that is, when it is in logic '1' or logic '0' state, depending upon whether the ENABLE input is active HIGH or active LOW respectively, the output is enabled. The multiplexer functions normally. When the ENABLE input is inactive, the output is disabled and permanently goes to either logic '0' or logic '1' state, depending upon whether the output is uncomplemented or complemented. Figure 8.6 shows how the 2-to-1 multiplexer of Fig. can be modified to include an ENABLE input. The functional table of this modified multiplexer is also shown in Fig. The ENABLE input here is active when HIGH. Some IC packages have more than one multiplexer. In that case, the ENABLE input and selection inputs are common to all multiplexers within the same IC package.

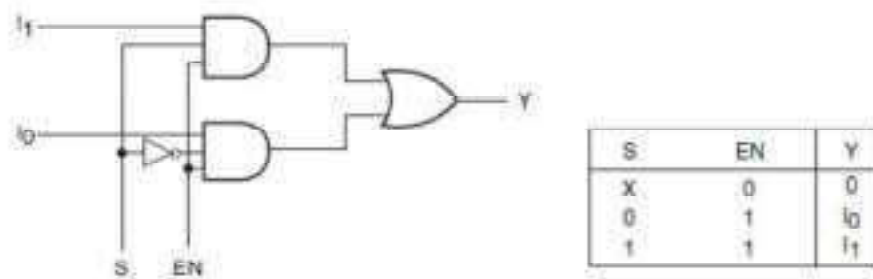


Figure 2-to-1 multiplexer with an ENABLE input.