Combinational circuits

By

K.sudheer

INTRODUCTION

4.1 Combinational Circuits

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**. In combinational logic, the output variables are at all times dependent on the combination of input variables.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the input variables and generate output signals. This process transforms binary information from the given input data to the required output data. Fig. 4.1 shows the block diagram of a combinational circuit. As shown in Fig. 4.1, the combinational circuit accepts n-input binary variables and generates output variables depending on the logical combination of gates.

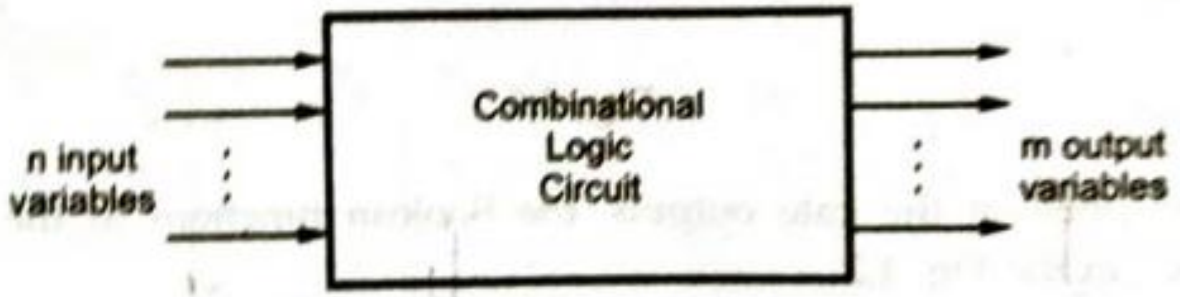


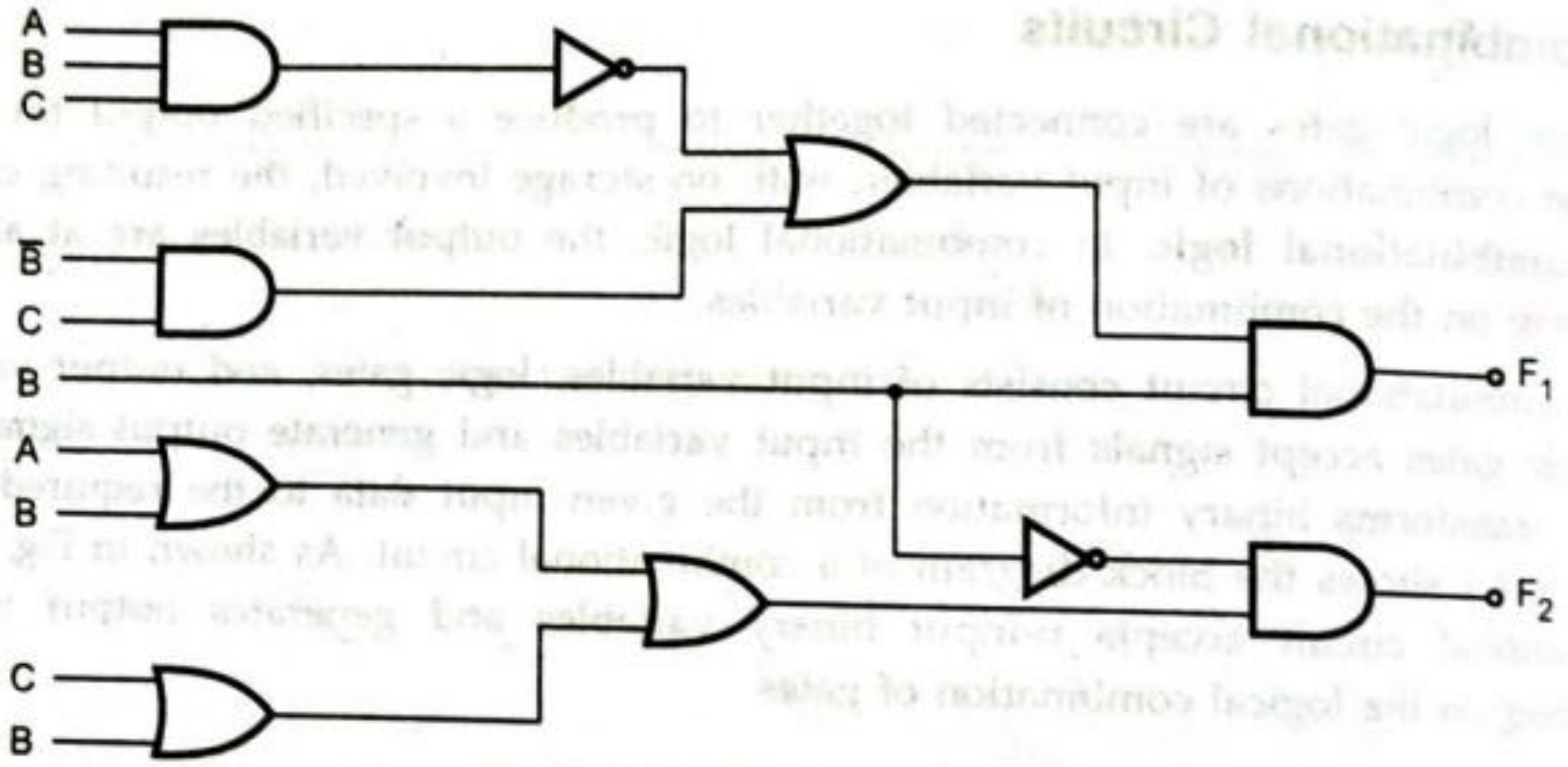
Fig. 4.1 Block diagram of a combinational circuit

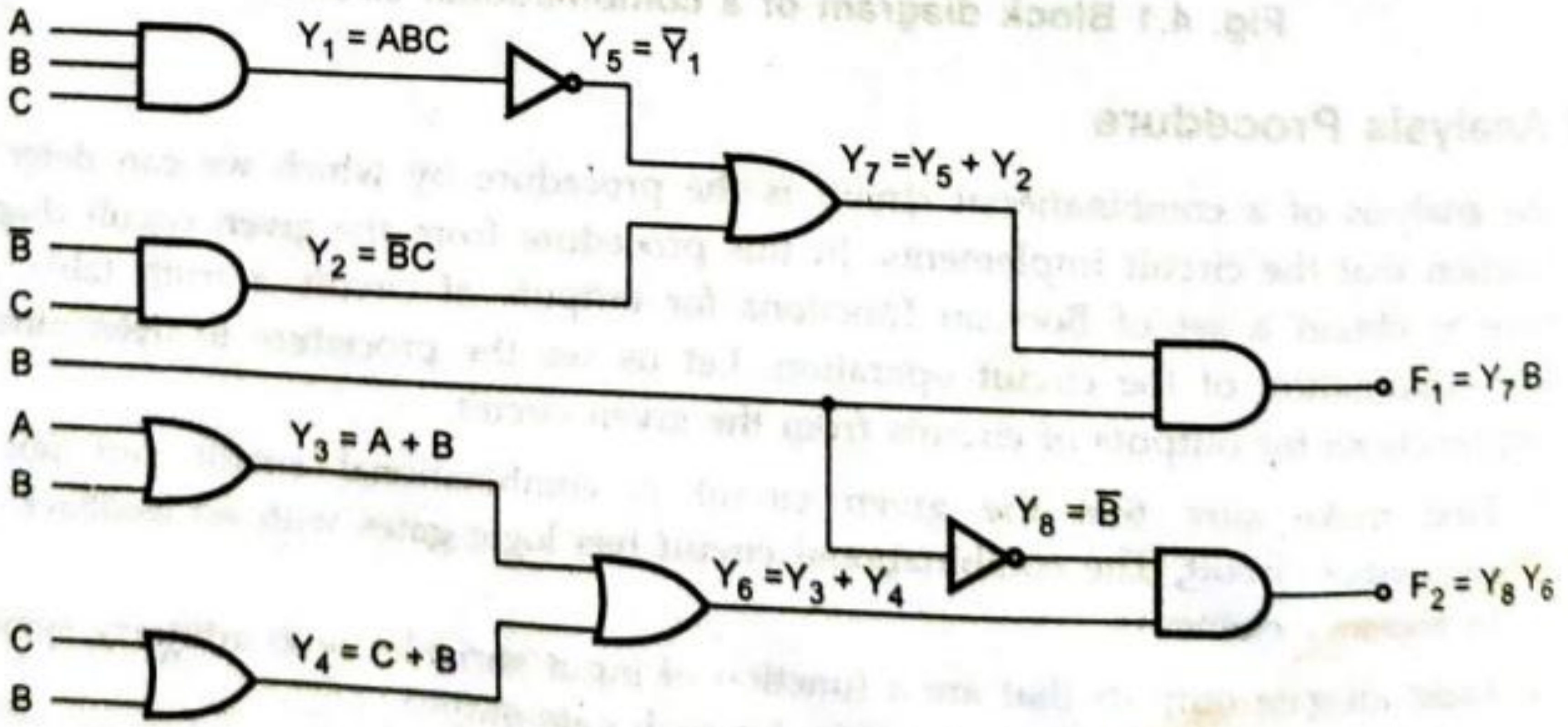
4.2 Analysis Procedure

The analysis of a combinational circuit is the procedure by which we can determine the function that the circuit implements. In this procedure from the given circuit diagram we have to obtain a set of Boolean functions for outputs of circuit, a truth table, or a possible explaination of the circuit operation. Let us see the procedure to determine the Boolean functions for outputs of circuits from the given circuit.

1. First make sure that the given circuit is combinational circuit and not the sequential circuit. The combinational circuit has logic gates with no feedback path or memory elements.
2. Label all gate outputs that are a function of input variables with arbitrary symbols, and determine the Boolean functions for each gate output.
3. Label the gates that are a function of input variables and previously labeled gates and determine the Boolean function for them.
4. Repeat the step 3 until the Boolean function for outputs of the circuit are obtained.
5. Finally, substituting previously defined Boolean functions, obtain the output Boolean functions in terms of input variables.

Example 4.1 : *Obtain the Boolean functions for outputs of the given circuit.*





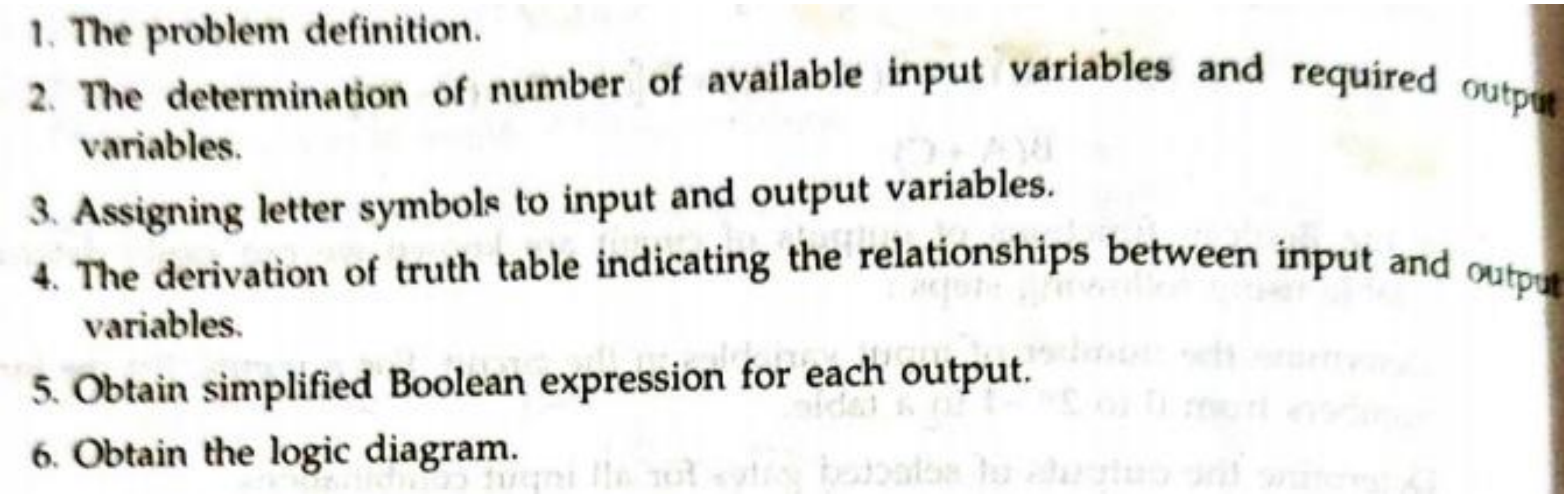
$$F_1 = Y_7 B = \left(\overline{Y_1} + Y_2\right) B = \left(\overline{ABC} + \bar{B}C\right) B$$

$$= \left[\left(\bar{A} + \bar{B} + \bar{C}\right) + \bar{B}C\right] B = \left(\bar{A} + \bar{C}\right) B$$

$$F_2 = Y_8 Y_6 = \bar{B}(Y_3 + Y_4) = \bar{B}[(A + B) + (C + B)]$$

$$= \bar{B}(A + C)$$

DESIGN PROCDURE



1. The problem definition.
2. The determination of number of available input variables and required outpu variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between input and outpu variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

Example 4.2 : *Design a combination logic circuit with three input variables that will produce a logic 1 output when more than one input variables are logic 1.*

Solution : Given problem specifies that there are three input variables and one output variable. We assign A, B and C letter symbols to three input variables and assign Y letter symbol to one output variable. The relationship between input variables and output variable can be tabulated as shown in truth table 4.1.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 4.1 Truth table

Now we obtain the simplified Boolean expression for output variable Y using K-map simplification.

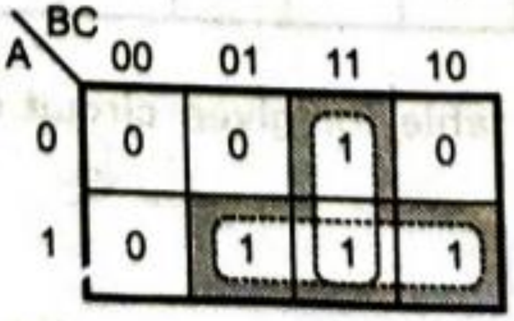
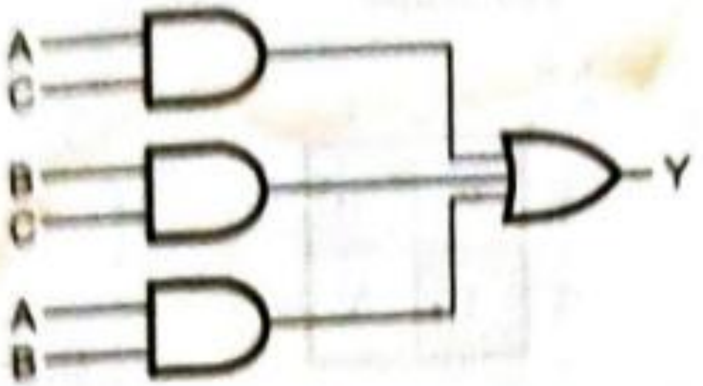


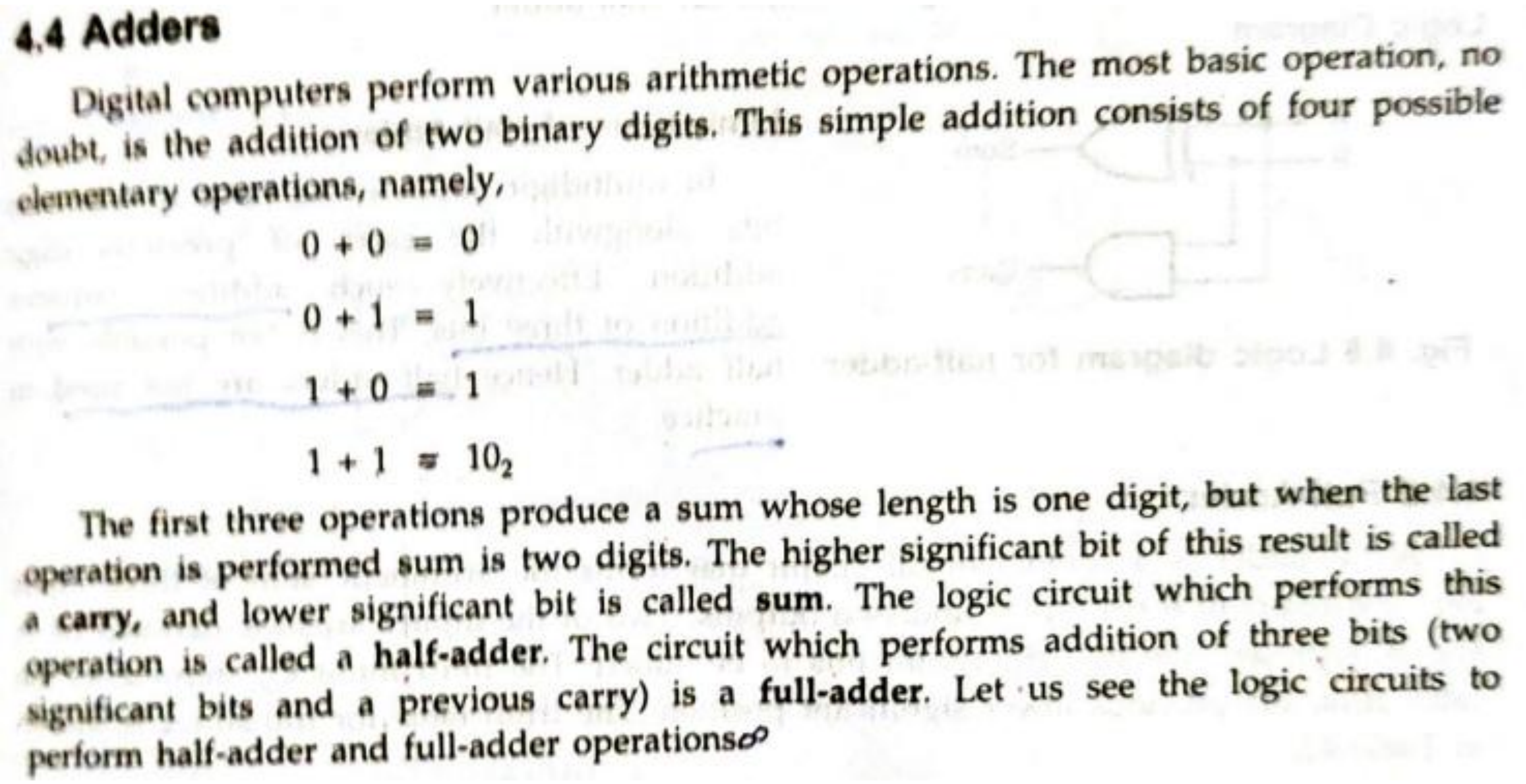
Fig. 4.4

$$Y = AC + BC + AB$$

Logic Diagram



ADDERS



4.4 Adders

Digital computers perform various arithmetic operations. The most basic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely,

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10_2$$

The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits. The higher significant bit of this result is called a **carry,** and lower significant bit is called **sum**. The logic circuit which performs this operation is called a **half-adder**. The circuit which performs addition of three bits (two significant bits and a previous carry) is a **full-adder**. Let us see the logic circuits to perform half-adder and full-adder operations.

4.4.1 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 4.2 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 4.2 Truth table for half-adder

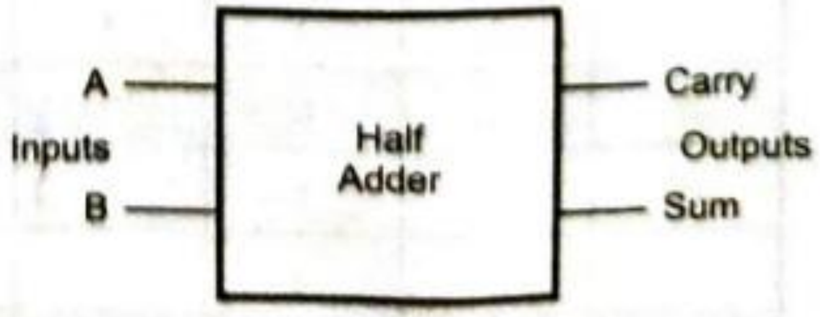
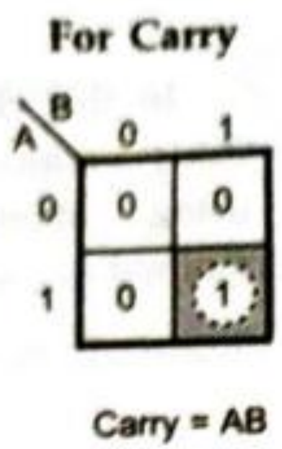


Fig. 4.6 Block schematic of half-adder

K-map simplification for carry and sum



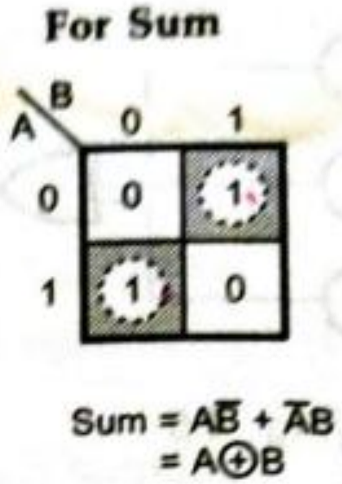


Fig. 4.7 Maps for half-adder

Logic Diagram

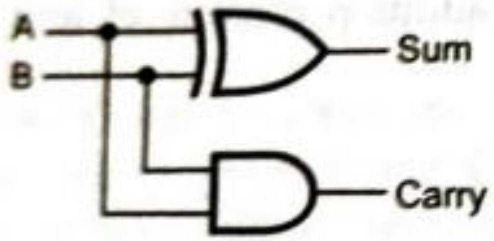


Fig. 4.8 Logic diagram for half-adder

Limitations of Half-Adder :

In multidigit addition we have to add two bits alongwith the carry of previous digit addition. Effectively such addition requires addition of three bits. This is not possible with half adder. Hence half-adders are not used in practice.

4.4.2 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. The truth table for full-adder is shown in Table 4.3.

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 4.3 Truth

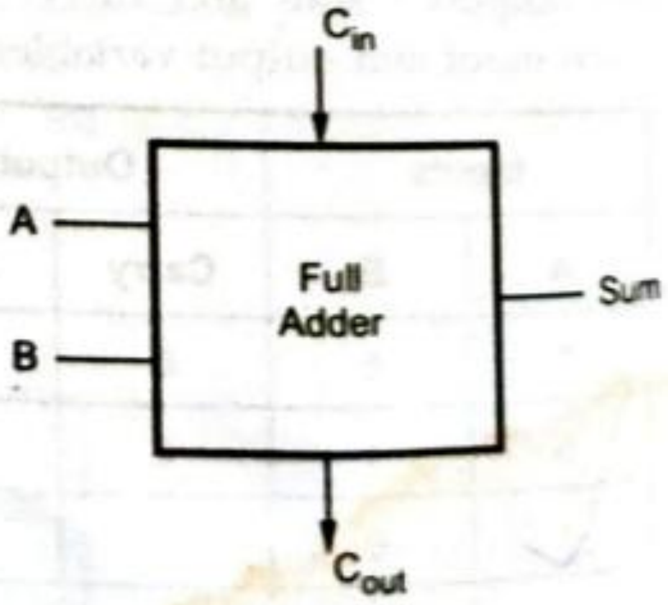
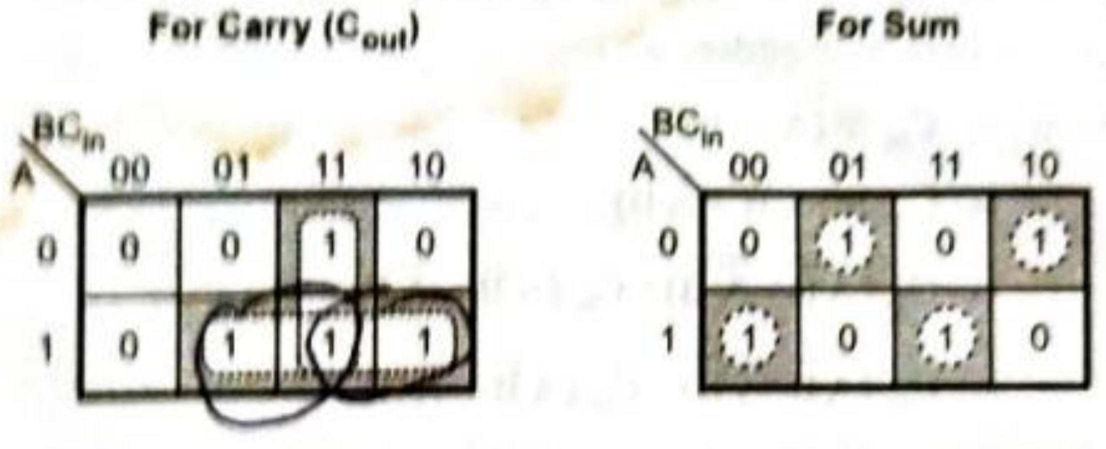


Fig. 4.9 Block schematic of

K-map simplification for carry and sum



$C_{out} = AB + A\,C_{in} + B\,C_{in}$ $\quad$ $Sum = \bar{A}\,\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\,\bar{C}_{in} + ABC_{in}$

Fig. 4.10 Maps for full-adder

Logic Diagram

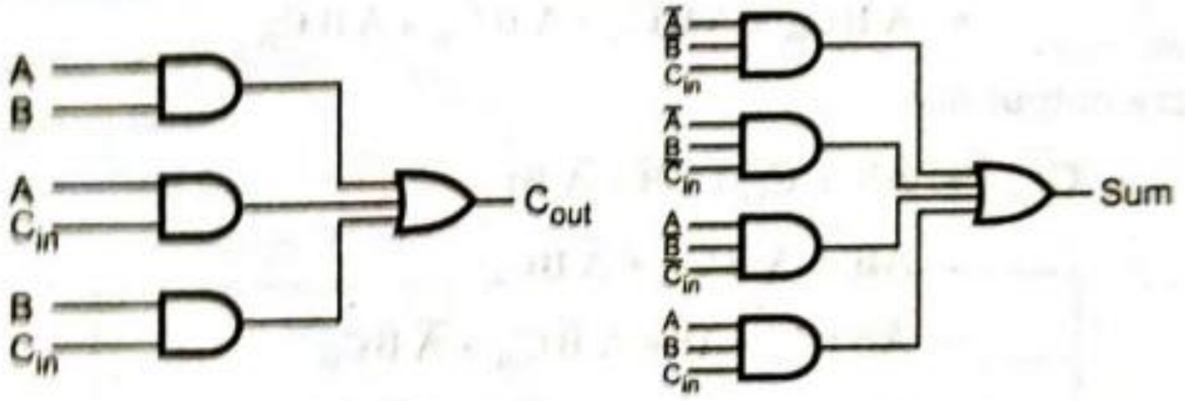

A
B
A
C_{in}
B
C_{in}
C_{out}
$\bar{A}$ $\bar{B}$ C_{in}
$\bar{A}$ B $\bar{C}_{in}$
A $\bar{B}$ $\bar{C}_{in}$
A B C_{in}
Sum

Fig. 4.11 Sum of product implementation of full-adder

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

$$\begin{aligned}
\text{Sum} &= \bar{A}\,\bar{B}\,C_{in} + \bar{A}\,B\,\bar{C}_{in} + A\,\bar{B}\,\bar{C}_{in} + A\,B\,C_{in} \\
&= C_{in}\,(\bar{A}\,\bar{B} + AB) + \bar{C}_{in}\,(\bar{A}\,B + A\,\bar{B}) \\
&= C_{in}\,(A \odot B) + \bar{C}_{in}\,(A \oplus B) = C_{in}\,(\overline{A \oplus B}) + \bar{C}_{in}\,(A \oplus B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}$$

With this simplified Boolean function circuit for full-adder can be implemented as shown in the Fig. 4.12.

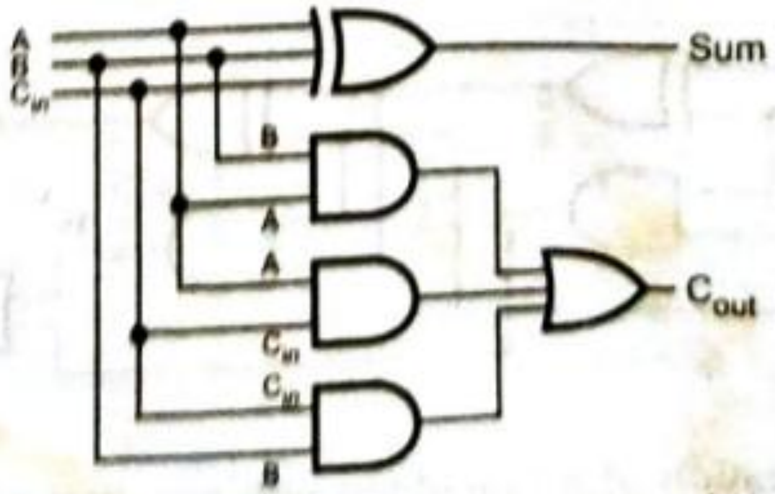
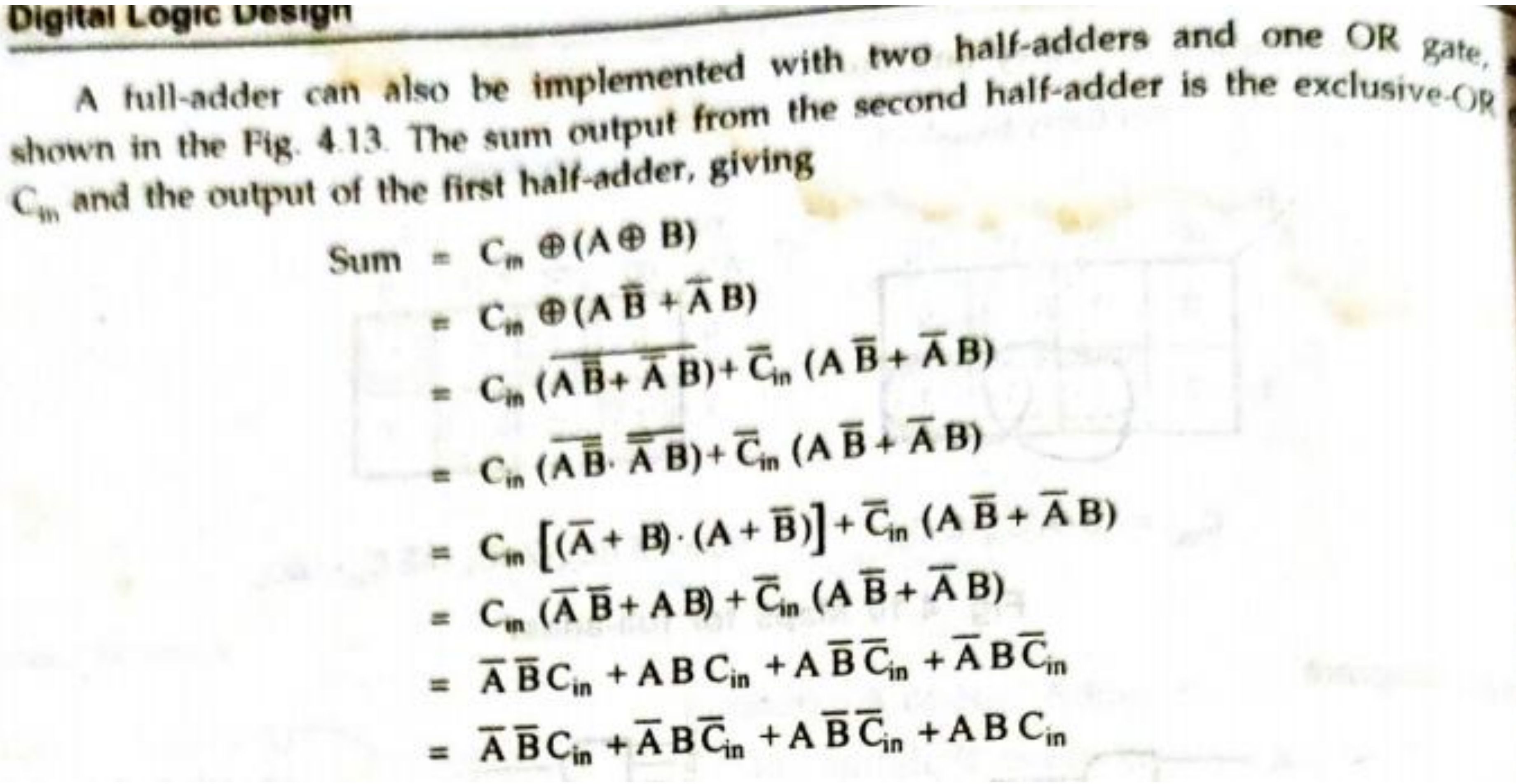


Fig. 4.12 Implementation of full-adder

FULL ADDER WITH 2 HALF ADDERS



A full-adder can also be implemented with two half-adders and one OR gate, shown in the Fig. 4.13. The sum output from the second half-adder is the exclusive-OR C_{in} and the output of the first half-adder, giving

$$\begin{aligned}
\text{Sum} &= C_{in} \oplus (A \oplus B) \\
&= C_{in} \oplus (A\bar{B} + \bar{A}B) \\
&= C_{in}\,\overline{(A\bar{B} + \bar{A}B)} + \bar{C}_{in}\,(A\bar{B} + \bar{A}B) \\
&= C_{in}\,(\overline{A\bar{B}} \cdot \overline{\bar{A}B}) + \bar{C}_{in}\,(A\bar{B} + \bar{A}B) \\
&= C_{in}\,[(\bar{A} + B) \cdot (A + \bar{B})] + \bar{C}_{in}\,(A\bar{B} + \bar{A}B) \\
&= C_{in}\,(\bar{A}\bar{B} + AB) + \bar{C}_{in}\,(A\bar{B} + \bar{A}B) \\
&= \bar{A}\bar{B}C_{in} + ABC_{in} + A\bar{B}\bar{C}_{in} + \bar{A}B\bar{C}_{in} \\
&= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}
\end{aligned}$$

the carry output is

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

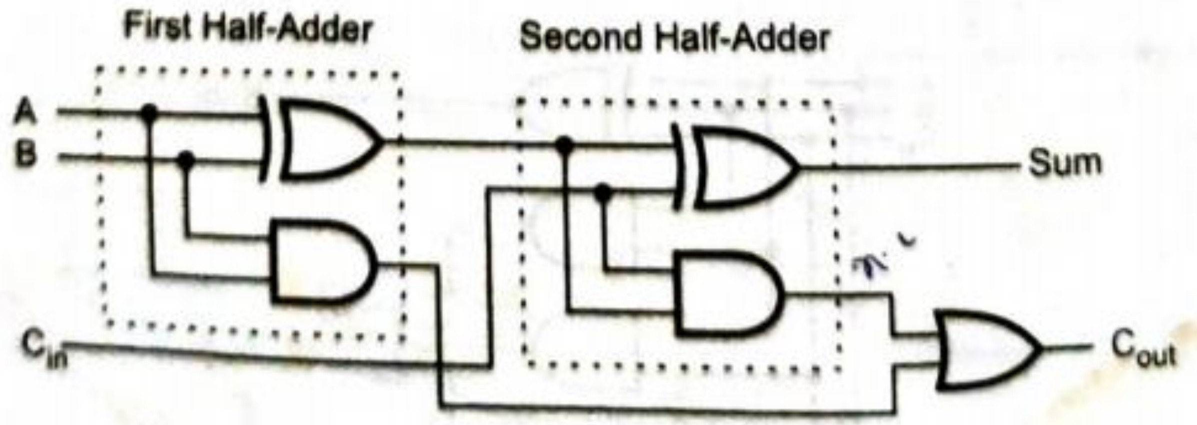


Fig. 4.13 Implementation of a full-adder with two half-adders and an OR gate

4.5 Subtractors

The subtraction consists of four possible elementary operations, namely,

0 – 0 = 0

0 – 1 = 1 with 1 borrow

1 – 0 = 1

1 – 1 = 0

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed. Just as there are half and full-adders, there are half and full-subtractors.

4.5.1 Half-Subtractor

A half-subtractor is a combinational circuit that does the subtraction between 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 4.4.

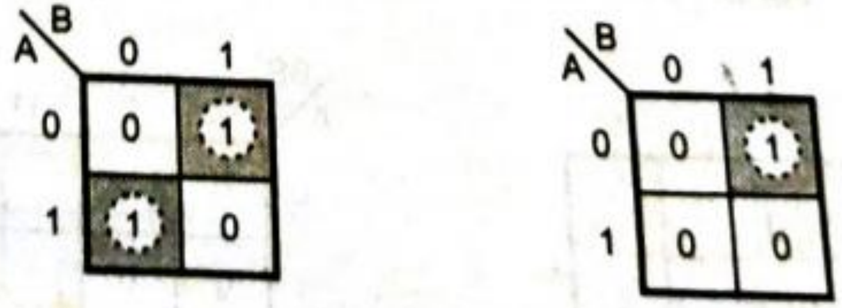
Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 4.4 Truth table for half-subtractor

As shown in the Table 4.4, half-subtractor has two input variables and two output variables. The Boolean expression for the outputs of half-subtractor can be determined as follows.

K-map simplification for half-subtractor

For Difference **For Borrow**



$$\text{Difference} = A\overline{B} + \overline{A}B$$
$$= A \oplus B$$

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

Fig. 4.14

Logic Diagram

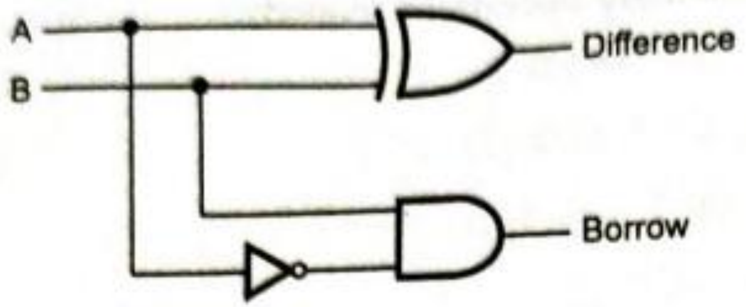


Fig. 4.15 Implementation of half-subtractor

Limitations of Half Subtractor :

In multidigit subtraction, we have to subtract bit alongwith the borrow of the previous digit subtraction. Effectively such subtraction requires subtraction between three bits. This is not possible with half-subtractor.

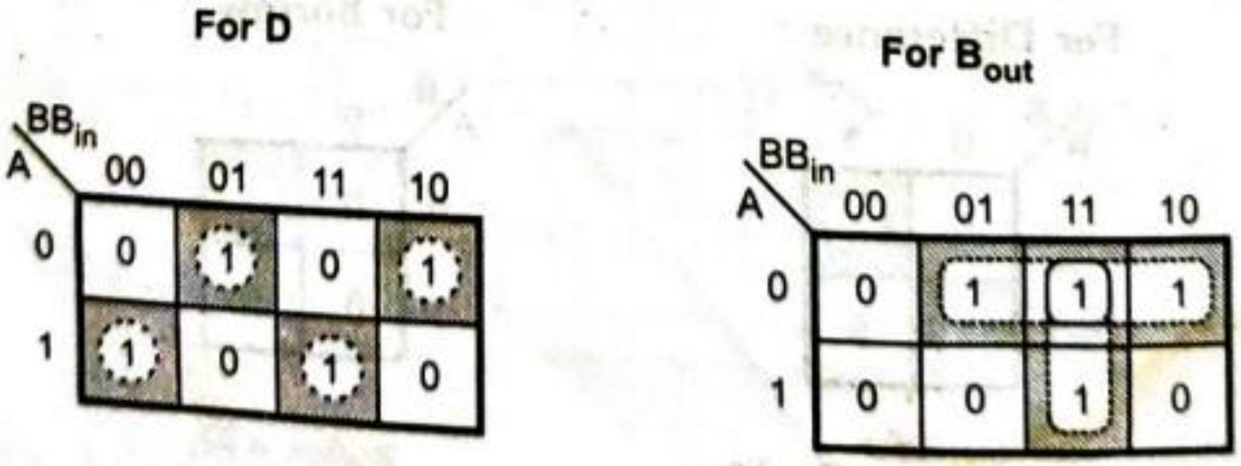
4.5.2 Full-Subtractor

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 4.5 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 4.5 Truth table for full-subtractor

K-map simplification of D and B_{out}



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

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

Fig. 4.16 Maps for full-subtractor

Logic circuit

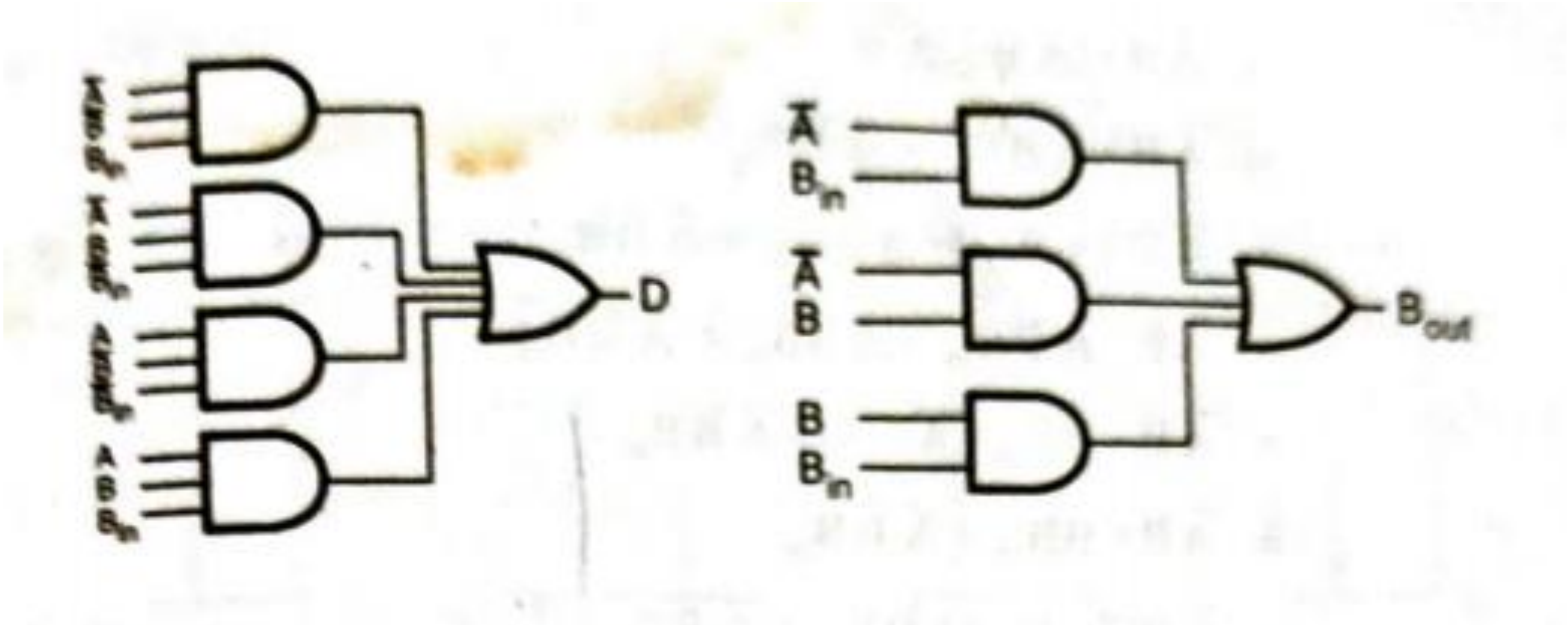


Fig. 4.17 Sum of product implementation of full-subtractor

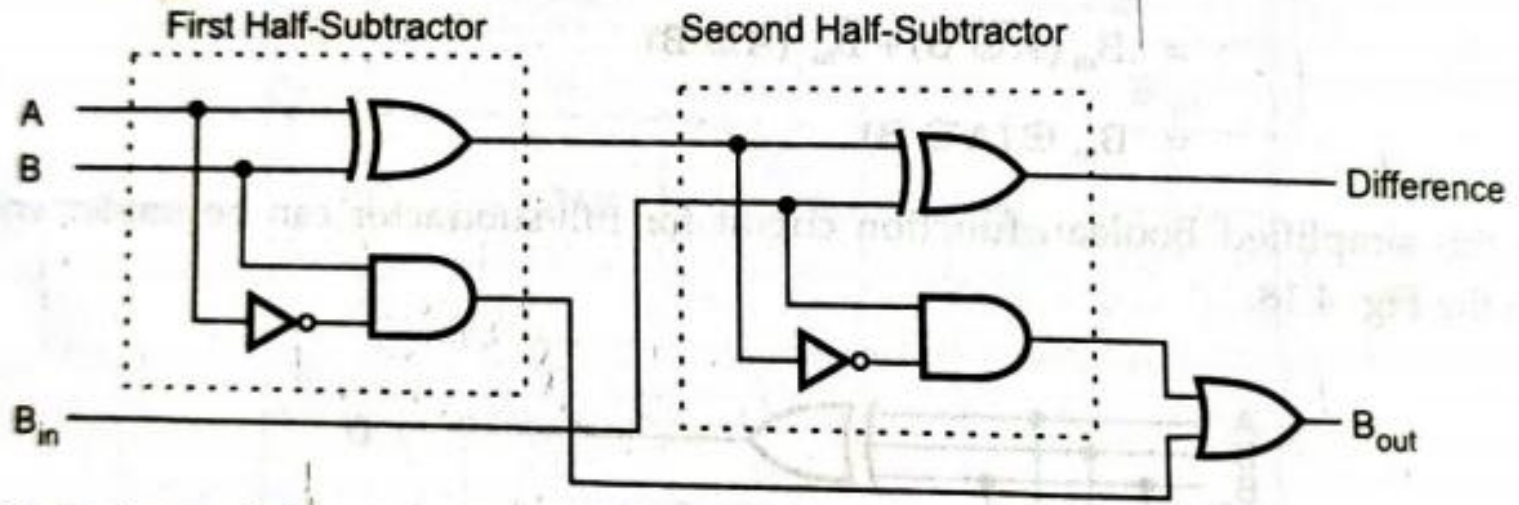


Fig. 4.19 Implementation of a full-subtractor with two half-subtractors and an OR gate

4.6 Binary Adder / Parallel Adder

We have seen, 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. 4.20 shows the block diagram of n-bit parallel adder using 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.

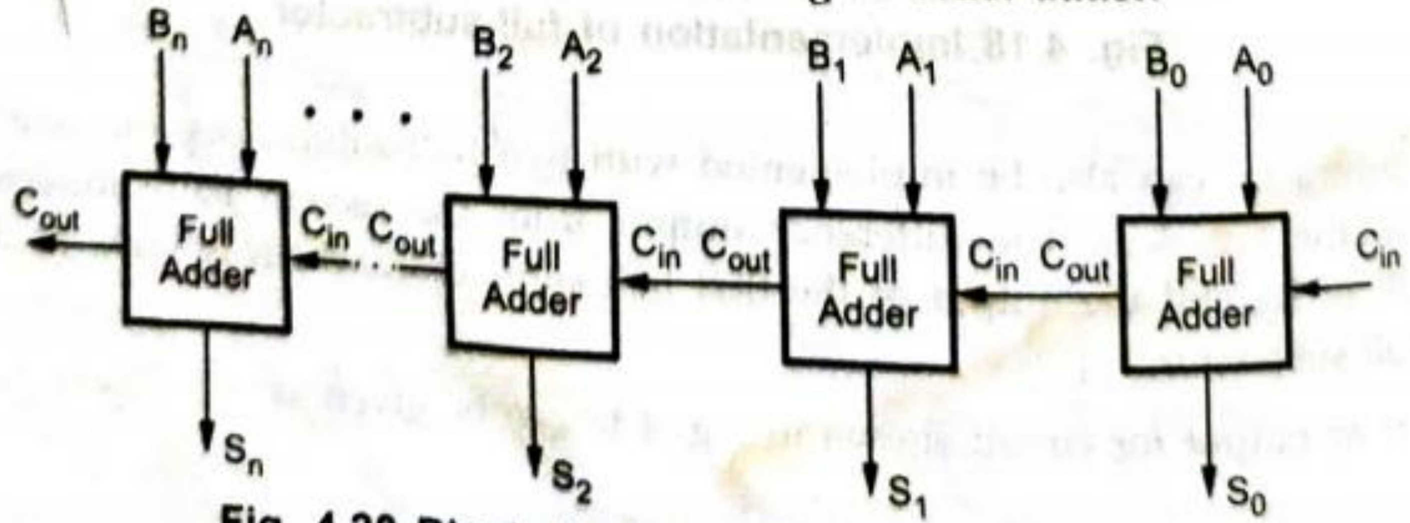


Fig. 4.20 Block diagram of n-bit parallel adder

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.

Example 4.3 : *Design a 4-bit parallel adder using full adders.*

Solution : Fig. 4.21 shows the block diagram and Fig. 4.22 shows logic symbol for 4-bit parallel adder. Here, for least significant position, carry input of full-adder is made 0.

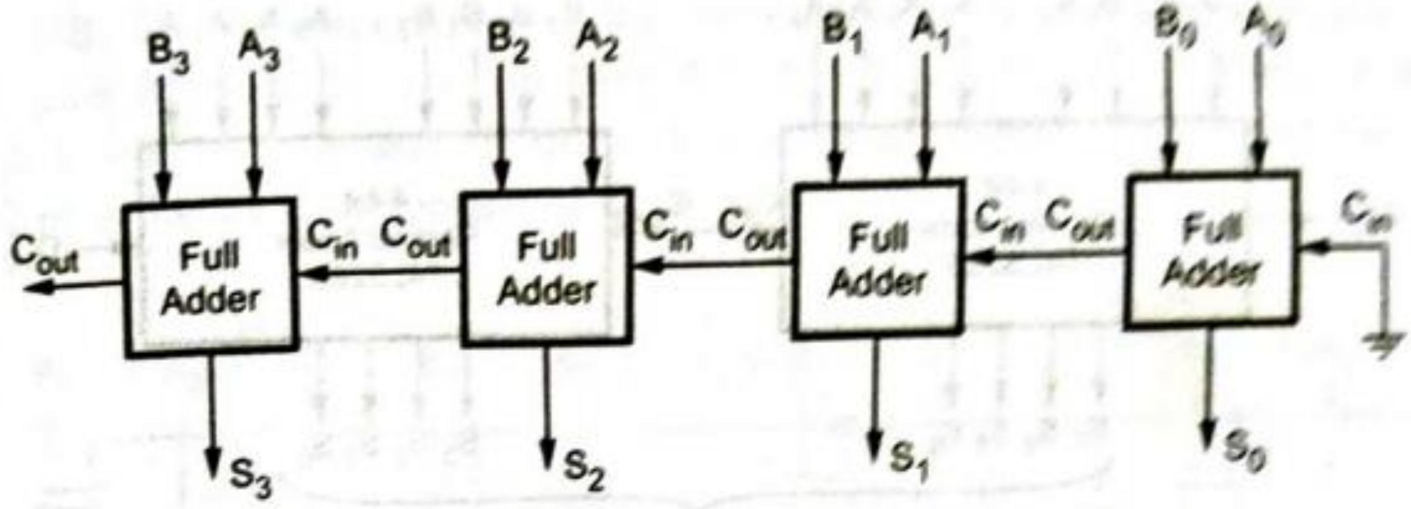


Fig. 4.21 Block diagram of 4-bit full adder

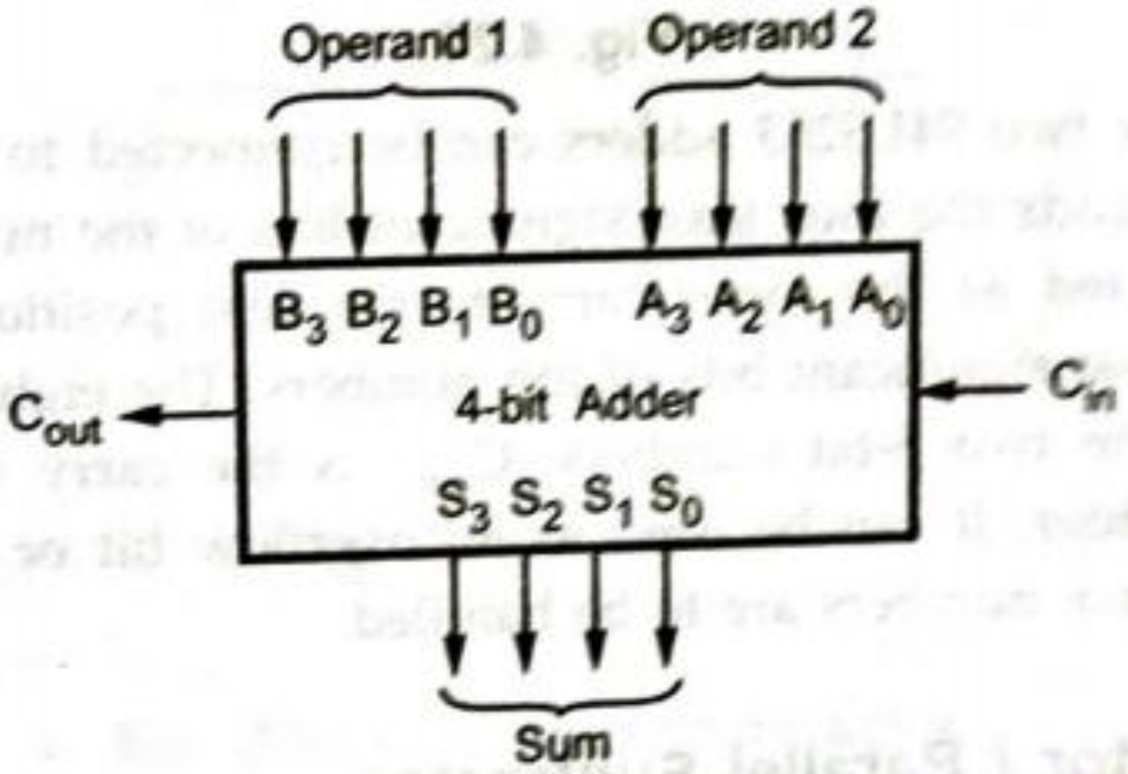


Fig. 4.22 Logic diagram of 4-bit parallel adder

Example 4.4 : *Design an 8-bit adder using two 74283s.*

Solution :

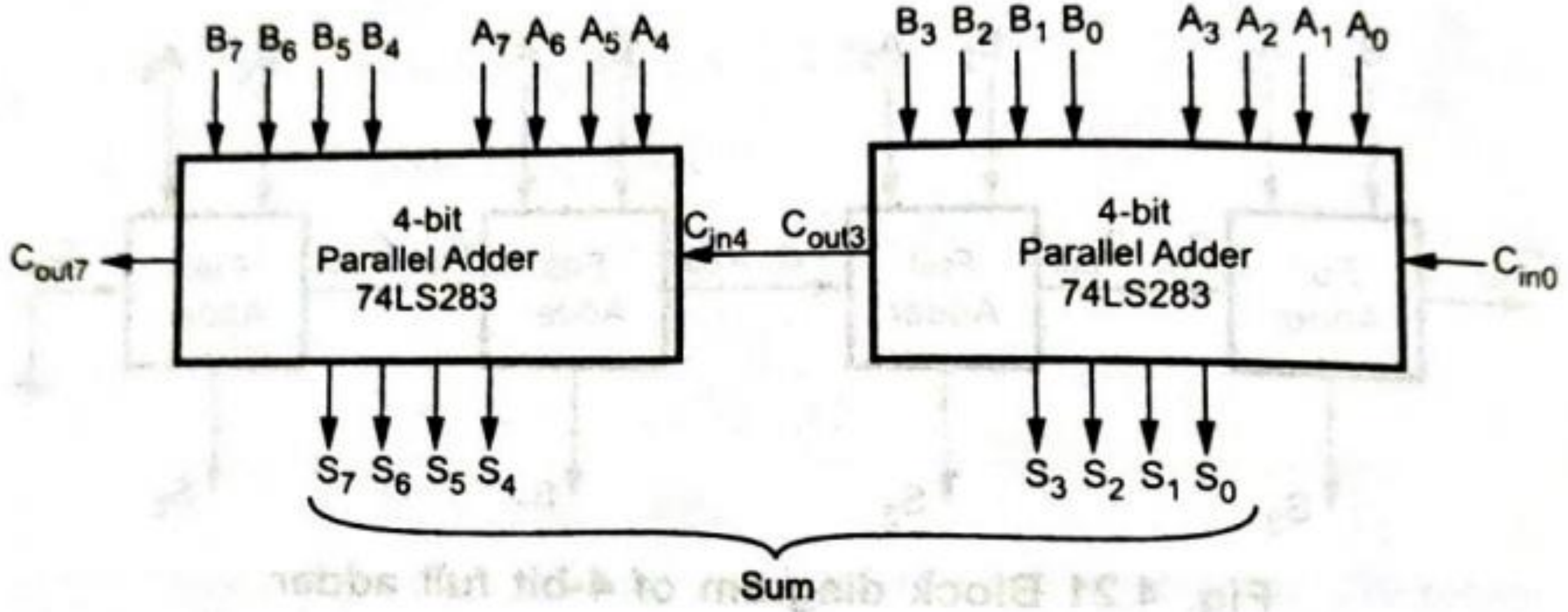


Fig. 4.24

Fig. 4.24 shows how two 74LS283 adders can be connected to add two 8-bit numbers. The adder on the right adds the four least significant bits of the numbers. The C_{out3} output of this adder is connected as the input carry to the first position of the second adder, which adds the four most significant bits of the numbers. The eight sum outputs represent the resultant sum of the two 8-bit numbers. $C_{out\,7}$ is the carry out of the last position (MSB) of the second adder. It can be used as an overflow bit or as a carry into another adder stage if large binary numbers are to be handled.

4.10 Decimal Adder

The digital systems handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0 0 0 0 to 1 0 0 1, i.e. each BCD digit is represented as a 4-bit binary number. When we write BCD number say 526, it can be represented as

$$\begin{array}{ccc} 5 & 2 & 6 \\ \downarrow & \downarrow & \downarrow \\ 0\,1\,0\,1 & 0\,0\,1\,0 & 0\,1\,1\,0 \end{array}$$

Here, we should note that BCD cannot be greater than 9.

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Sum Equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

```
      6        0  1  1  0   ←  BCD for 6
   +  3        0  0  1  1   ←  BCD for 3
   -----      ------------
      9        1  0  0  1   ←  BCD for 9
```

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

```
      6        0  1  1  0   ←  BCD for 6
   +  8        1  0  0  0   ←  BCD for 8
   -----      ------------
     14        1  1  1  0   ←  Invalid BCD number
```

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

$$
\begin{array}{rr}
 & 6 \\
+ & 8 \\
\hline
 & 14
\end{array}
\qquad
\begin{array}{rcccl}
 & 0 & 1 & 1 & 0 \leftarrow \text{BCD for 6} \\
 & 1 & 0 & 0 & 0 \leftarrow \text{BCD for 8} \\
\hline
 & 1 & 1 & 1 & 0 \leftarrow \text{Invalid BCD number} \\
+ & 0 & 1 & 1 & 0 \leftarrow \text{Add 6 for correction} \\
\hline
\end{array}
$$

$$
\underbrace{0\ 0\ 0\ 1}_{1} \qquad \underbrace{0\ 1\ 0\ 0}_{4} \quad \leftarrow \text{BCD for 14}
$$

After addition of 6 carry is produced into the second decimal position.

Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD

8					1	0	0	0	←	BCD for 8
+ 9					1	0	0	1	←	BCD for 9
17	0	0	0	1	0	0	0	1	←	Incorrect BCD result

In this, case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown below

8					1	0	0	0	←	BCD for 8
+ 9					1	0	0	1	←	BCD for 9
17	0	0	0	1	0	0	0	1	←	Incorrect BCD result
+	0	0	0	0	0	1	1	0	←	Add 6 for correction
	0	0	0	1	0	1	1	1	←	BCD for 17

Going through these three cases of BCD addition we can summarise the BCD addition procedure as follows :

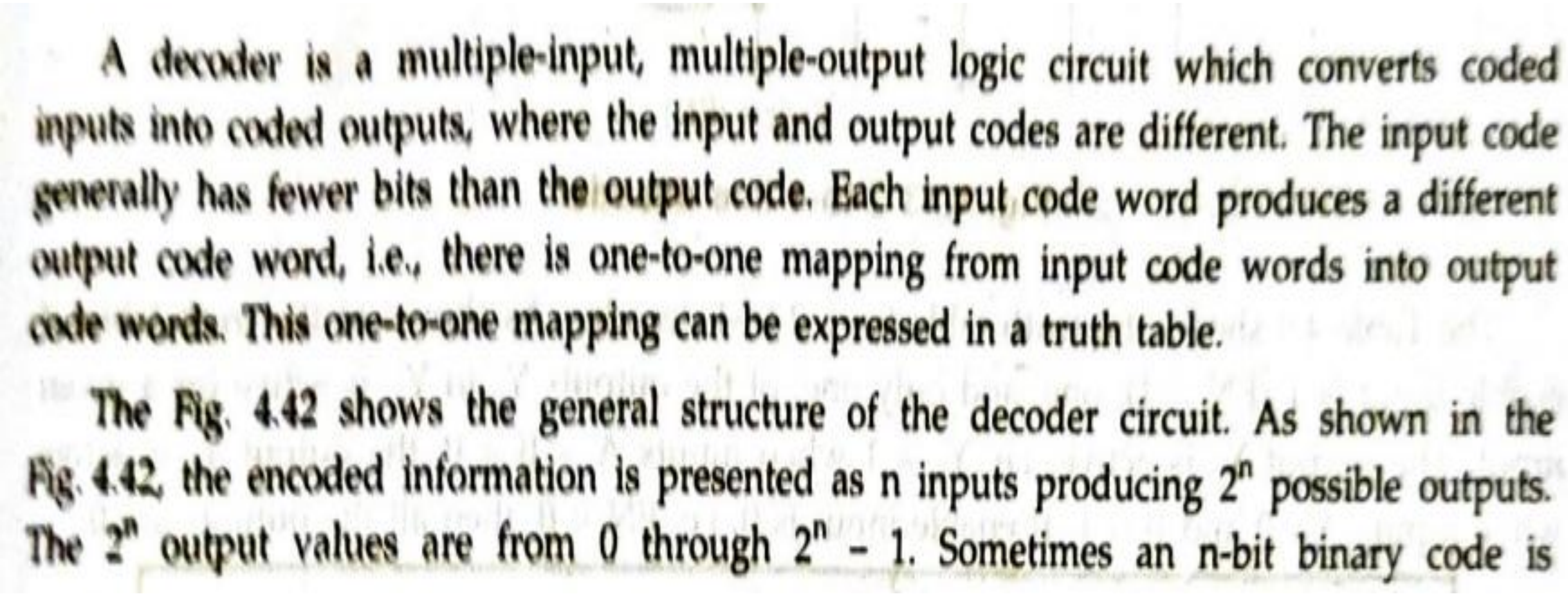
1. Add two BCD numbers using ordinary binary addition.
2. If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
3. If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Thus 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.

decoder



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. The input code generally has fewer bits than the output code. Each input code word produces a different output code word, i.e., there is one-to-one mapping from input code words into output code words. This one-to-one mapping can be expressed in a truth table.

The Fig. 4.42 shows the general structure of the decoder circuit. As shown in the Fig. 4.42, the encoded information is presented as n inputs producing 2^n possible outputs. The 2^n output values are from 0 through $2^n - 1$. Sometimes an n-bit binary code is

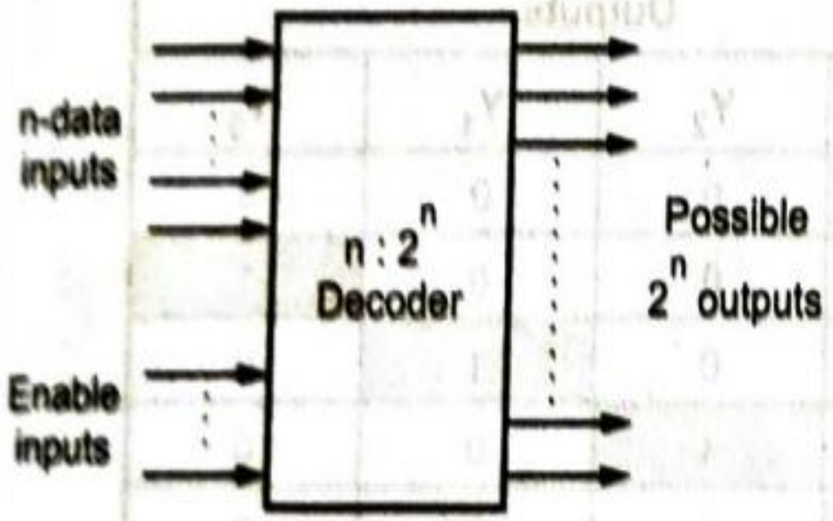


Fig. 4.42 General structure of decoder

truncated to represent fewer output values than 2^n. For example, in the BCD code, the 4-bit combinations 0000 through 1001 represent the decimal digits 0-9, and combinations 1010 through 1111 are not used. Usually, a decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

4.13.1 Binary Decoder

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

Fig. 4.43 shows 2 to 4 decoder. Here, 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables. The two inverters provide the complement of the inputs, and each one of four AND gates generates one of the minterms.

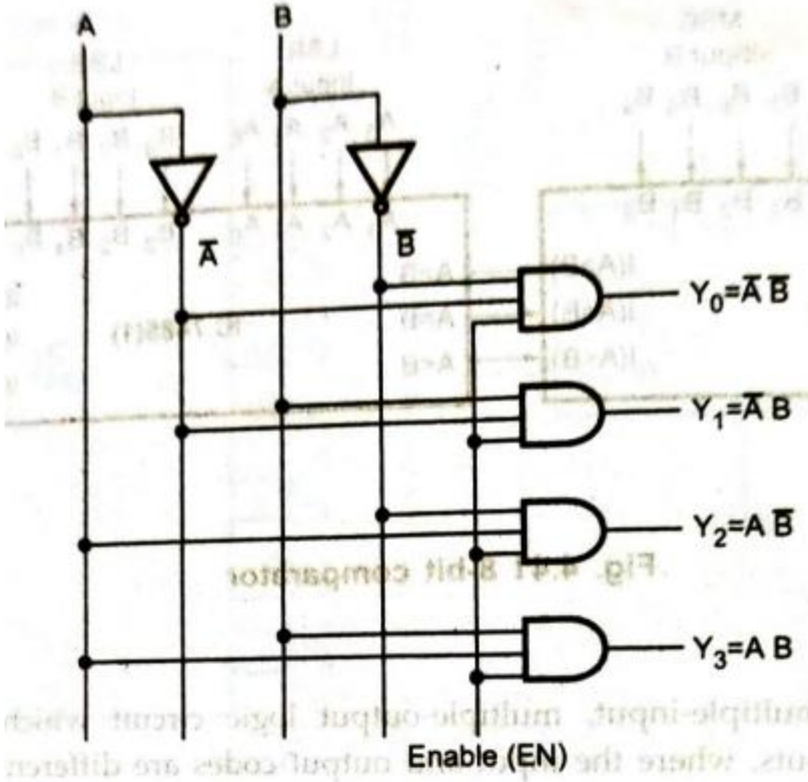


Fig. 4.43 2-to-4 line decoder

The Table 4.9 shows the truth table for a 2-to-4 decoder. As shown in the truth table, 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.

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 4.9 Truth table for a 2-to-4 decoder

Example 4.10 : *Draw the circuit for 3-to-8 decoder and explain.*

Solution : Fig. 4.44 shows 3-to-8 line decoder. Here, 3 inputs are decoded into eight outputs, each output represent one of the minterms of the 3-input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. Enable input is provided to activate decoded output based on data inputs A, B, and C. The table shows the truth table for 3 to 8 decoder.

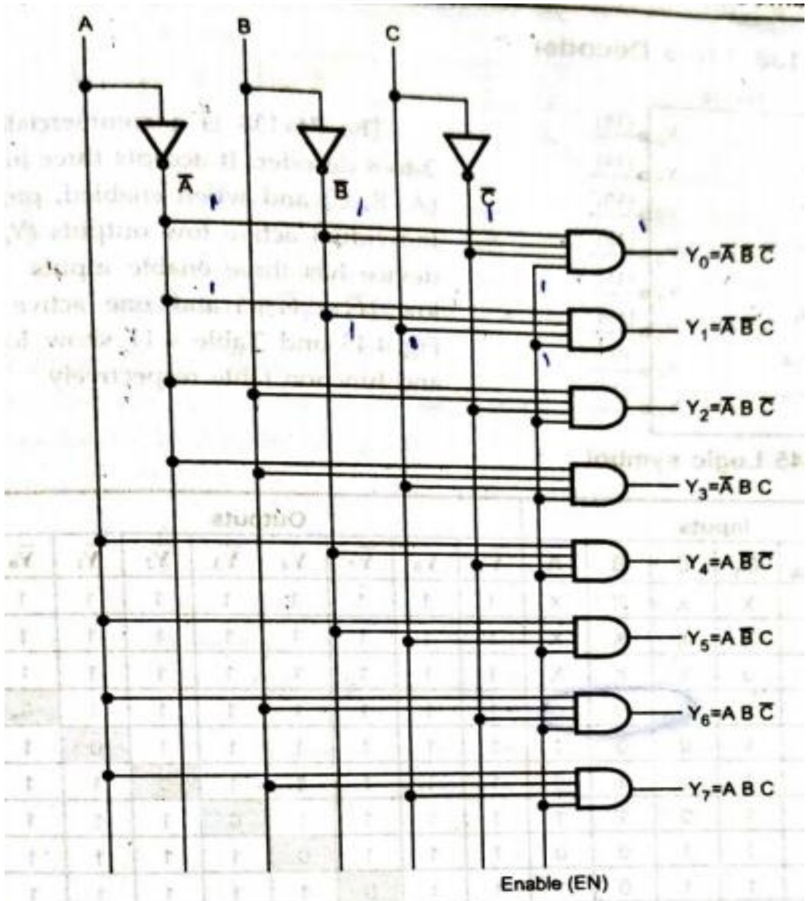

A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
$Y_0=\overline{A}\,\overline{B}\,\overline{C}$
$Y_1=\overline{A}\,\overline{B}\,C$
$Y_2=\overline{A}\,B\,\overline{C}$
$Y_3=\overline{A}\,B\,C$
$Y_4=A\,\overline{B}\,\overline{C}$
$Y_5=A\,\overline{B}\,C$
$Y_6=A\,B\,\overline{C}$
$Y_7=A\,B\,C$
Enable (EN)

Fig. 4.44 3:8 line decoder

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 4.10 Truth table for a 3-to-8 decoder

Encoder

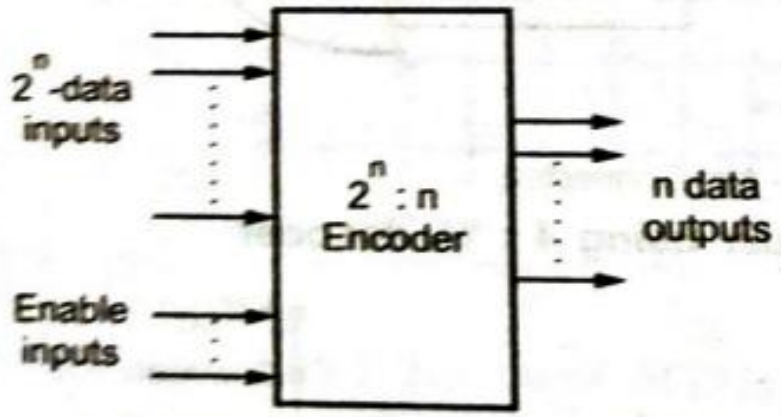


Fig. 4.58 General structure of encoder

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. 4.58 shows the general structure of the encoder circuit. As shown in the Fig. 4.58, the decoded information is presented as 2^n inputs producing n possible outputs.

4.14.1 Decimal to BCD Encoder

The decimal to BCD encoder, usually has ten input lines and four output lines. The decoded decimal data acts as an input for encoder and encoded BCD output is available on the four output lines.

The Fig. 4.59 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 4.16.

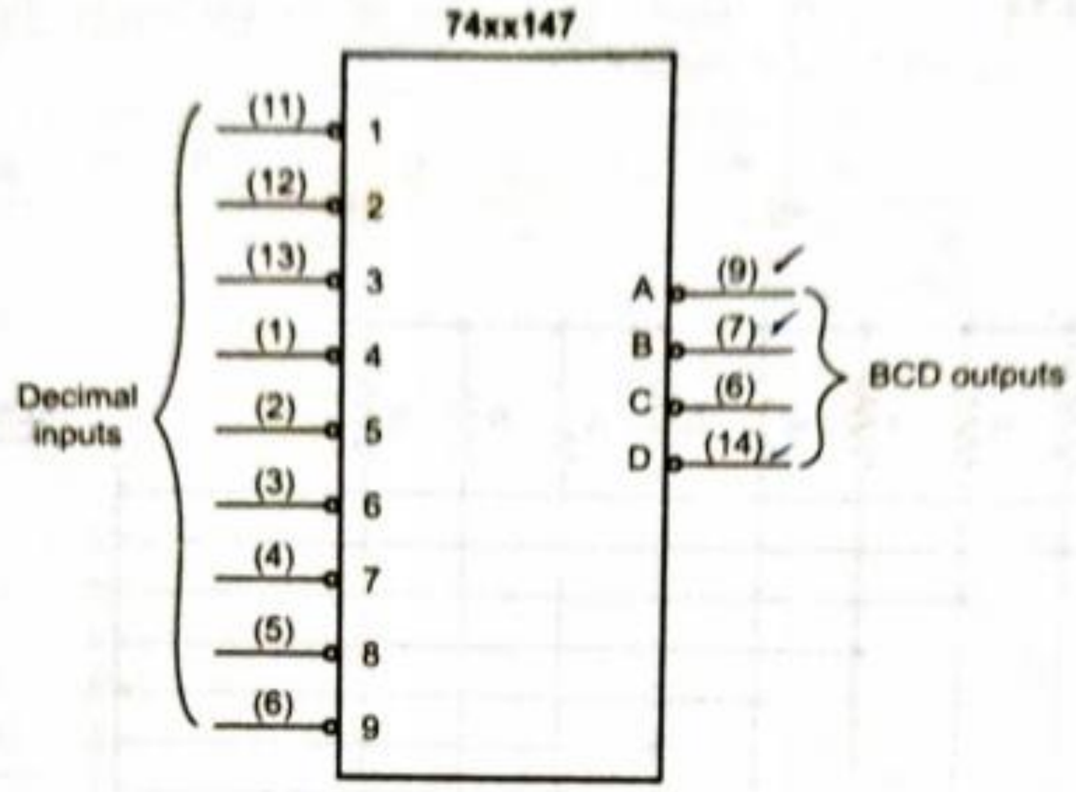


Fig. 4.59 Logic symbol for 74xx147 (Decimal to BCD encoder)

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 4.16 Truth table for decimal to BCD encoder

4.14.3 Priority Encoder

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

Table 4.18 shows truth table of 4-bit priority encoder.

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

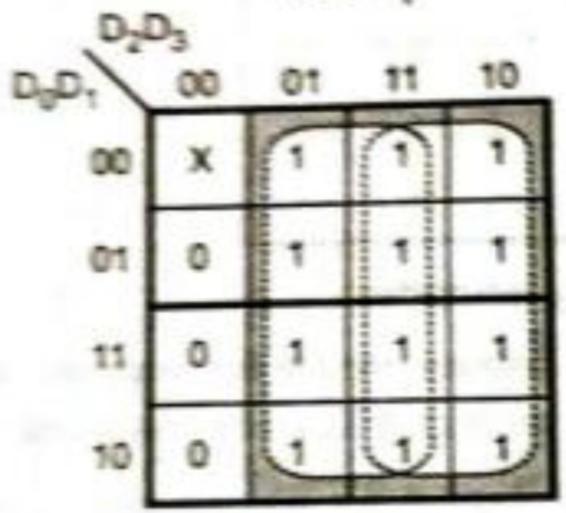
Table 4.18 Truth table of 4-bit priority encoder

3-
2-
[illegible]
0:

Table 4.15 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 11. The D_2 has the next priority. Thus, when $D_3 = 0$ and $D_2 = 1$, regardless of other two lower priority inputs output is 10. The output for D_1 is generated only if higher priority inputs are 0, and so on. 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

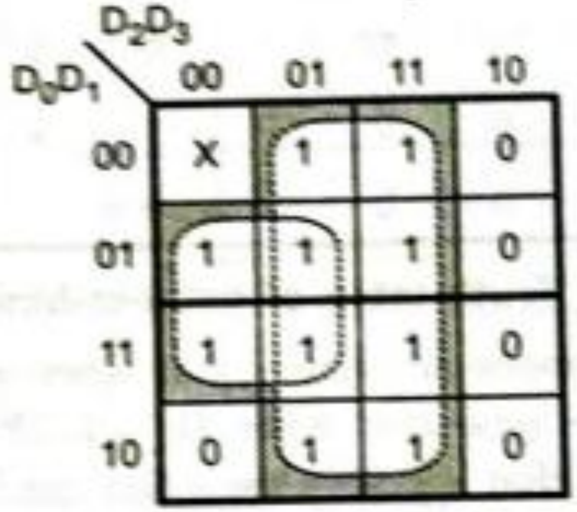
For Y_1



D_0D_1 \ D_2D_3	00	01	11	10
00	X	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$Y_1 = D_2 + D_3$

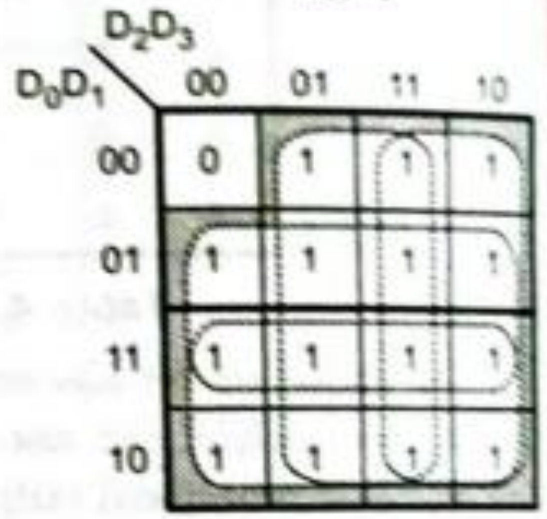
For Y_0



D_0D_1 \ D_2D_3	00	01	11	10
00	X	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

$Y_0 = D_3 + D_1\overline{D_2}$

For V



D_0D_1 \ D_2D_3	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$V = D_0 + D_1 + D_2 + D_3$

Logic diagram

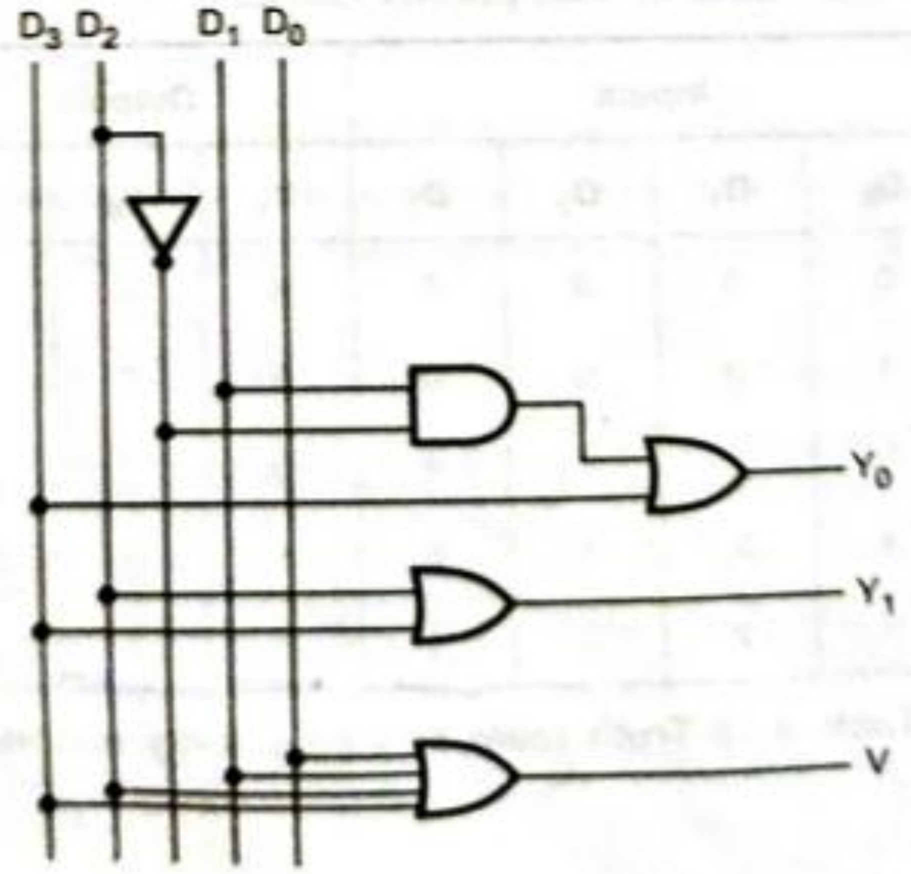


Fig. 4.62 (b)

4.15 Multiplexers

Multiplexer is a digital switch. It allows digital information from several sources to be routed onto a single output line, as shown in the Fig. 4.65. The basic multiplexer has several data-input lines and a single output line. 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.

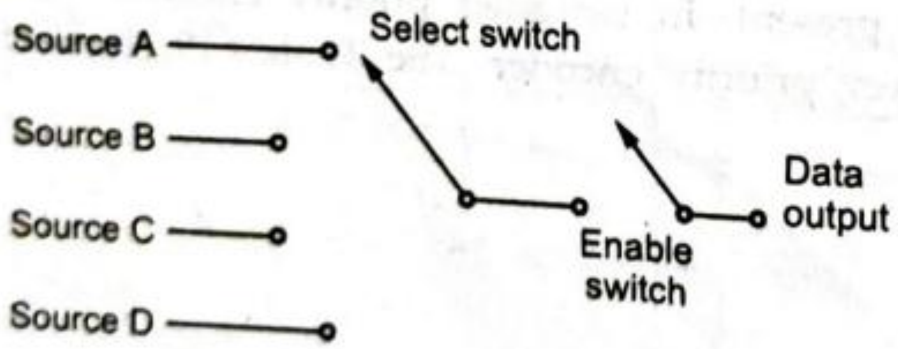
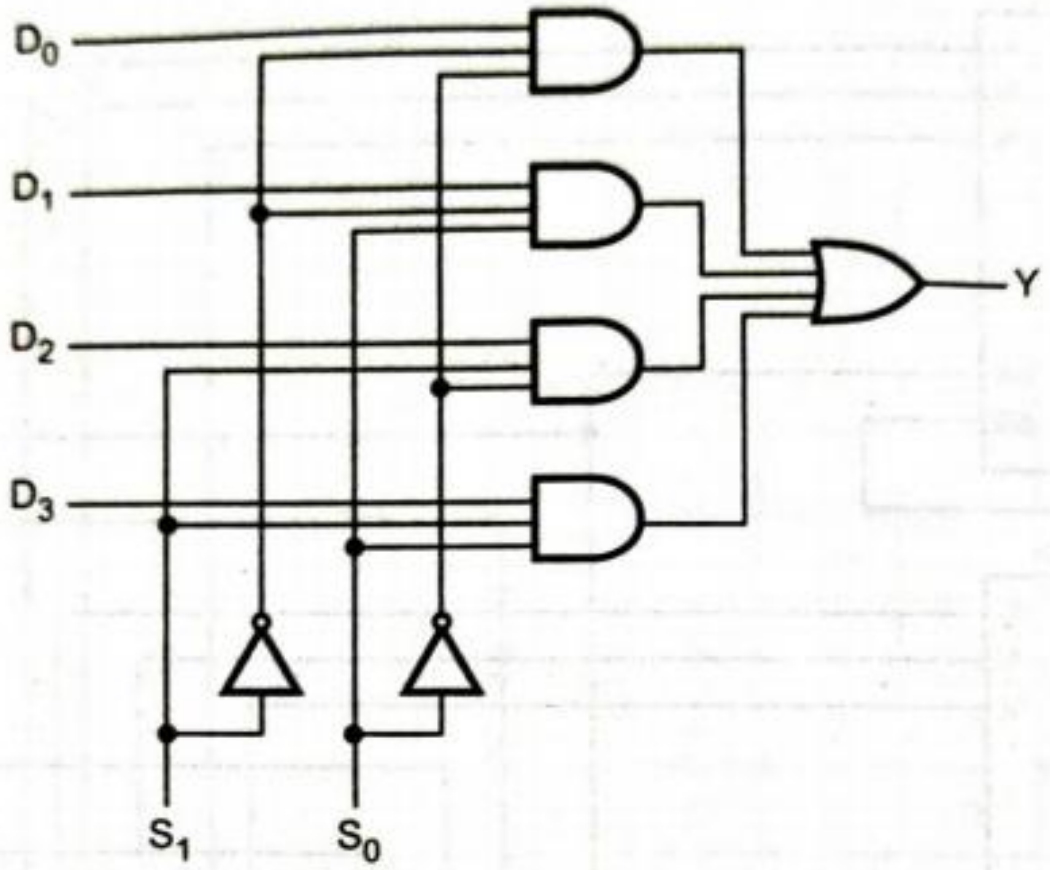


Fig. 4.65 Analog selector switch

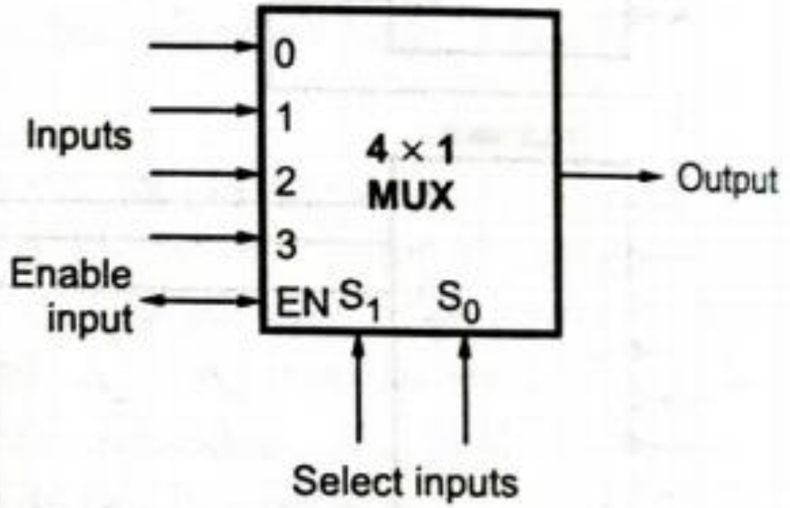
Fig. 4.66 (a) shows 4-to-1 line multiplexer. Each of the four lines, D_0 to D_3, is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



(a) Logic diagram

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

(b) Function table



(c) Logic symbol

Fig. 4.66 4-to-1 line multiplexer

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 $S_1\ S_0 = 0\ 1$.

In some cases, two or more multiplexers are enclosed within one IC package, as shown in the Fig. 4.67. The Fig. 4.67 shows quadruple 2-to-1 line multiplexer, i.e. four multiplexers, each capable of selecting one of two input lines. Output Y_1 can be selected to be equal to either A_1 or B_1. Similarly output Y_2 may have the value of A_2 or B_2, and so on. The selection line S selects one of two lines in all four multiplexers. The control input E enables the multiplexers in the 0 state and disables them in the 1 state. When E = 1, outputs have all 0's, regardless of the value of S.

Example 4.20 : *Implement the following Boolean function using 8 :1 MUX.*

$$F(P, Q, R, S) = \sum m\,(0, 1, 3, 4, 8, 9, 15)$$

Solution : Fig. 4.76 shows the implementation of given Boolean function with 8 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
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

Fig. 4.76 (a) Implementation table

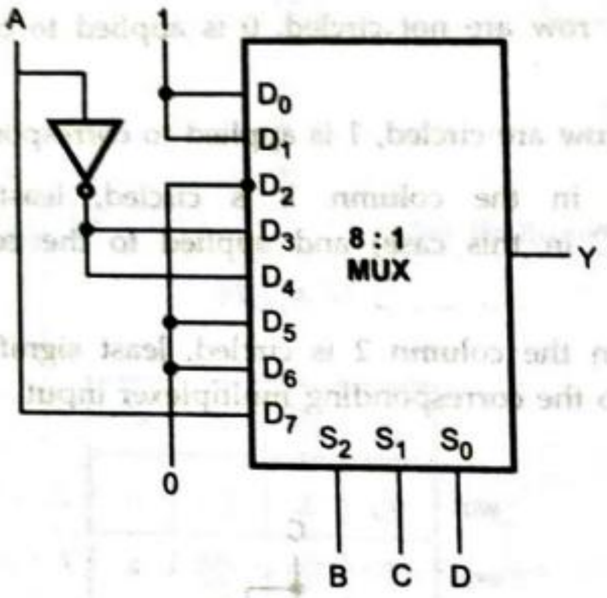


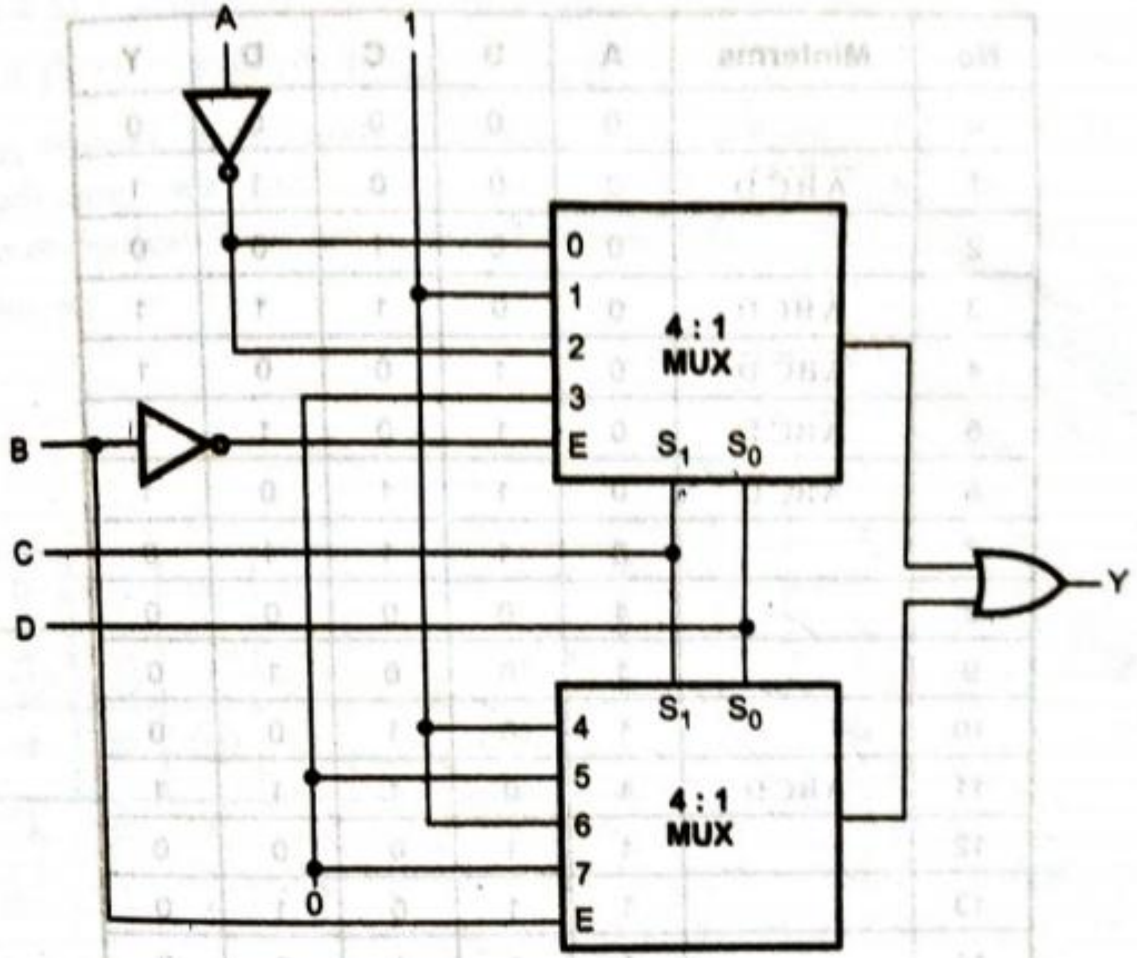
Fig. 4.76 (b) Multiplexer Implementation

Example 4.21 : *Implement the following Boolean function using 4 : 1 MUX*
$F(A, B, C, D) = \sum m\ (0, 1, 2, 4, 6, 9, 12, 14)$

Solution : The function has four variables. To implement this function we require 8 : 1 multiplexer. i.e., two 4 : 1 multiplexers. We have already seen how to construct 8 : 1 multiplexer using two 4 : 1 multiplexers. The same concept is used here to implement given Boolean function.

	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
	$\bar{A}$	1	$\bar{A}$	0	1	0	1	0

Fig. 4.77 (a) Implementation table



A
1
B
C
D
0
1
2
3
E
4 : 1
MUX
S1
S0
4
5
6
7
E
4 : 1
MUX
0
Y

4.16 Demultiplexers

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. Fig. 4.81 shows 1 : 4 demultiplexer. The single input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines.

Enable	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Table 4.24 Function table for 1:4 demultiplexer

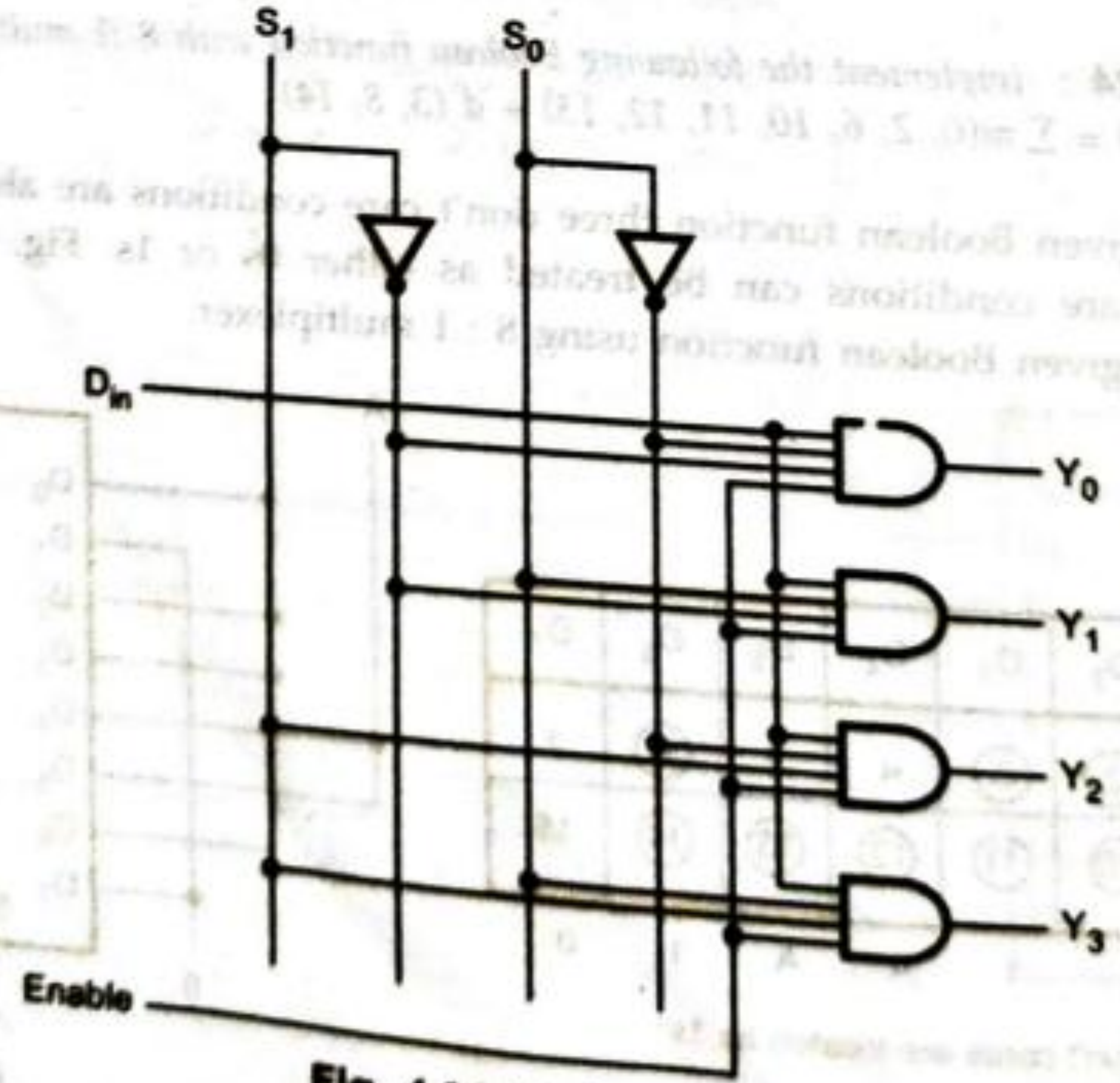


Fig. 4.81 (a) Logic diagram

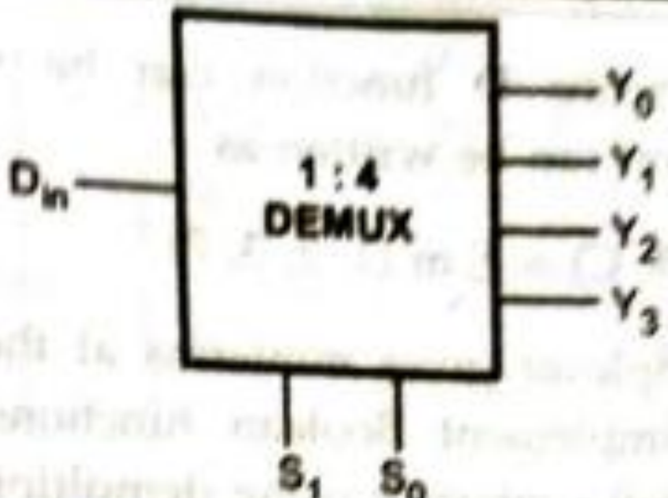


Fig. 4.81 (b) Logic symbol

Example 4.25 : *Design 1 : 8 demultiplexer using two 1 : 4 demultiplexers.*

Solution : The cascading of demultiplexers is similar to the cascading of decoder. Fig. 4.82 shows cascading of two 1 : 4 demultiplexers to form 1 : 8 demultiplexer.

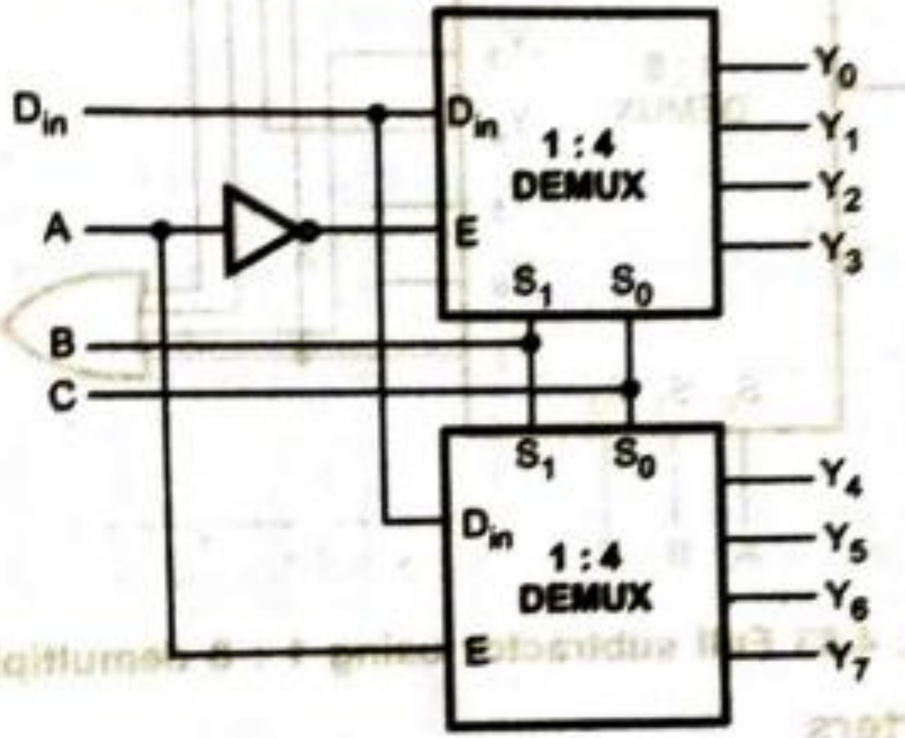


Fig. 4.82 Cascading of demultiplexers

Flip Flops

A digital computer needs devices which can store information. A flip flop is a binary storage device. It can store binary bit either 0 or 1. It has two stable states HIGH and LOW i.e. 1 and 0. It has the property to remain in one state indefinitely until it is directed by an input signal to switch over to the other state. It is also called bistable multivibrator.

The basic formation of flip flop is to store data. They can be used to keep a record or what value of variable (input, output or intermediate). Flip flop are also used to exercise control over the functionality of a digital circuit i.e. change the operation of a circuit depending on the state of one or more flip flops. These devices are mainly used in situations which require one or more of these three.

Operations, storage and sequencing.

Latch Flip Flop

The R-S (Reset Set) flip flop is the simplest flip flop of all and easiest to understand. It is basically a device which has two outputs one output being the inverse or complement of the other, and two inputs. A pulse on one of the inputs to take on a particular logical state. The outputs will then remain in this state until a similar pulse is applied to the other input. The two inputs are called the Set and Reset input (sometimes called the preset and clear inputs).

Such flip flop can be made simply by cross coupling two inverting gates either NAND or NOR gate could be used Figure 1(a) shows on RS flip flop using NAND gate and Figure 1(b) shows the same circuit using NOR gate.

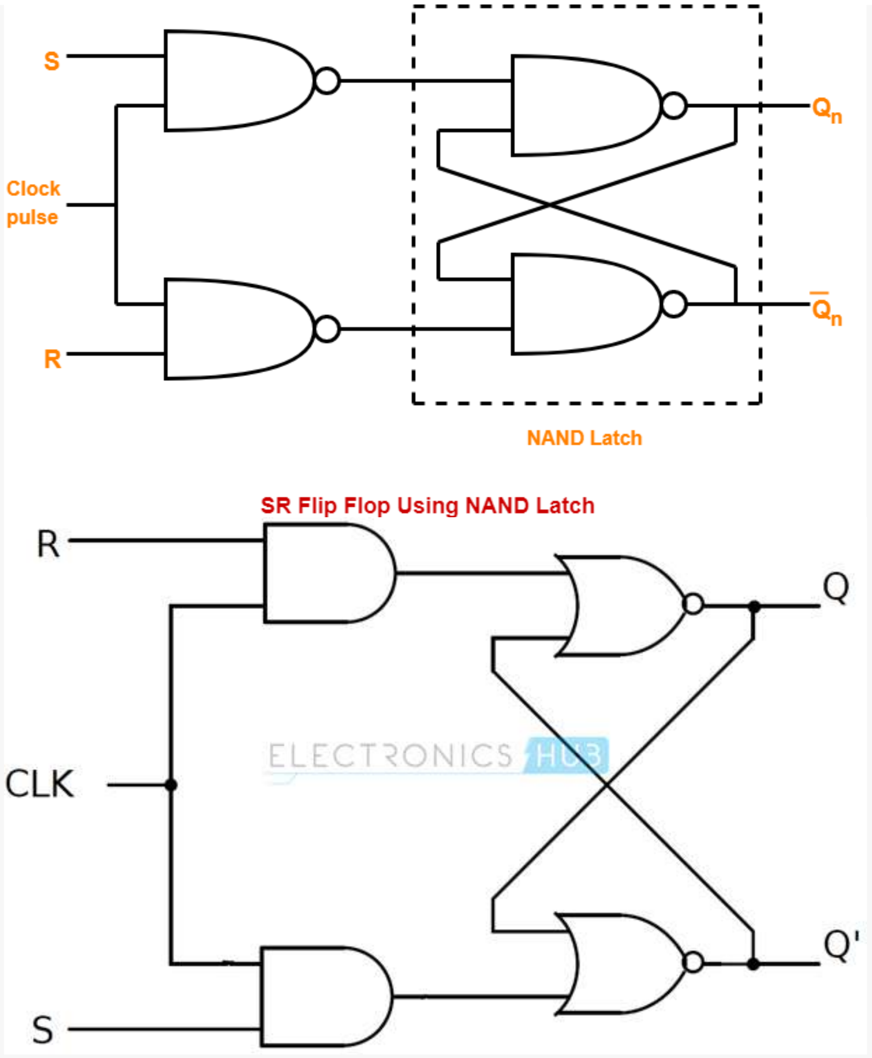


Figure 1: Latch R-S Flip Flop Using NAND and NOR Gates

To describe the circuit of Figure 1(a), assume that initially both R and S are at the logic 1 state and that output is at the logic 0 state.

Now, if Q = 0 and R = 1, then these are the states of inputs of gate B, therefore the outputs of gate B is at 1 (making it the inverse of Q i.e. 0). The output of gate B is connected to an input of gate A so if S = 1, both inputs of gate A are at the logic 1 state. This means that the output of gate A must be 0 (as was originally specified). In other words, the 0 state at Q is continuously disabling gate B so that any change in R has no effect. Also the 1 state at $\bar{Q}$ is continuously enabling gate A so that any change S will be transmitted through to Q. The above conditions constitute one of the stable states of the device referred to as the Reset state since Q = 0.

Now suppose that the R-S flip flop in the Reset state, the S input goes to 0. The output of gate A i.e. Q will go to 1 and with Q = 1 and R = 1, the output of gates B () will go to 0 with now 0 gate A is disabled keeping Q at 1. Consequently, when S returns to the 1 state it has no effect on the flip flop whereas a change in R will cause a change in the output of gate B. The above conditions constitute the other stable state of the device, called the Set state since Q = 1. Note that the change of the state of S from 1 to 0 has caused the flip flop to change from the Reset state to the Set state.

There is another input condition which has not yet been considered. That is when both the R and S inputs are taken to the logic state 0. When this happens both Q and will be forced to 1 and will remain so far as long as R and S are kept at 0. However when both inputs return to 1 there is no way of knowing whether the flip flop will latch in the Reset state or the Set state. The condition is said to be indeterminate because of this indeterminate state great care must be taken when using R-S flip flop to ensure that both inputs are not instructed simultaneously.

Table 1: The truth table for the NAND R-S flip flop

Initial Conditions	**Inputs (Pulsed)**		**Final Output**	
Q	**S**	**R**	**Q**	
1	0	0	indeterminate	
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0
0	0	0	indeterminate	
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1

or more simply shown in Table 2

Table 2: Simple NAND R-S Flip Flop Truth Table

S	**R**	**Q**
0	0	indeterminate
0	1	Set (1)
1	0	Reset(0)
1	1	No Change

When NOR gate are used the R and S inputs are transposed compared with the NAND version. Also the stable state when R and S are both 0. A change of state is effected by pulsing the appropriate input to the 1 state. The indeterminate state is now when both R and S are simultaneously at logic 1. Table 3 shows this operation.

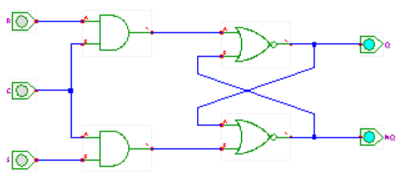
Table 3: NOR Gate R-S Flip Flop Truth Table

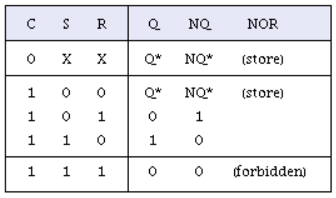
S	**R**	**Q**
0	0	No Change
0	1	Reset (0)
1	0	Set (1)
1	1	Indeterminate

Clocked RS Flip Flop

The RS latch flip flop required the direct input but no clock. It is very use full to add clock to control precisely the time at which the flip flop changes the state of its output.

In the clocked R-S flip flop the appropriate levels applied to their inputs are blocked till the receipt of a pulse from an other source called clock. The flip flop changes state only when clock pulse is applied depending upon the inputs. The basic circuit is shown in Figure 2. This circuit is formed by adding two AND gates at inputs to the R-S flip flop. In addition to control inputs Set (S) and Reset (R), there is a clock input (C) also.





C	S	R	Q	NQ	NOR
0	X	X	Q*	NQ*	(store)
1	0	0	Q*	NQ*	(store)
1	0	1	0	1	
1	1	0	1	0	
1	1	1	0	0	(forbidden)

Figure 2: Clocked RS Flip Flop

Table 4: The truth table for the Clocked R-S flip flop				
Initial Conditions	**Inputs (Pulsed)**		**Final Output**	**Comment**
Q	**S**	**R**	**Q (t + 1)**	No Change
0	0	0	0	No Change
0	0	1	0	Clear Q
0	1	0	1	Set Q
0	1	1	???	indeterminate
1	0	0	1	No Change
1	0	1	0	Clear Q
1	1	0	1	Set Q
1	1	1	???	indeterminate

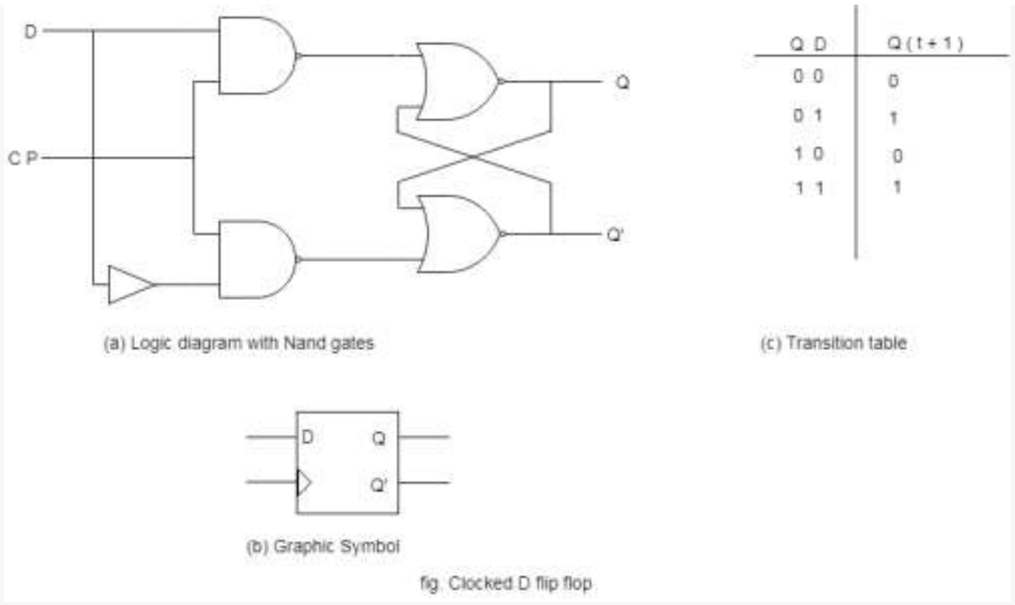
The excitation table for R-S flip flop is very simply derived as given below

Table 5: Excitation table for R-S Flip Flop

S	**R**	**Q**
0	0	No Change
0	1	Reset (0)
1	0	Set (1)
1	1	Indeterminate

D Flip Flop

A D type (Data or delay flip flop) has a single data input in addition to the clock input as shown in Figure 3.



Q	D	Q (t + 1)
0	0	0
0	1	1
1	0	0
1	1	1

Figure 3: D Flip Flop

Basically, such type of flip flop is a modification of clocked RS flip flop gates from a basic Latch flip flop and NOR gates modify it in to a clock RS flip flop. The D input goes directly to S input and its complement through NOT gate, is applied to the R input.

This kind of flip flop prevents the value of D from reaching the output until a clock pulse occurs. The action of circuit is straight forward as follows.

When the clock is low, both AND gates are disabled, there fore D can change values with out affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this case, Q is forced equal to D when the clock again goes low, Q retains or stores the last value of D. The truth table for such a flip flop is as given below in table 6.

Table 6: Truth table for D Flip Flop

S	R	Q(t + 1)
0	0	0
0	1	1
1	0	0
1	1	1

The excitation table for D flip flop is very simply derived given as under.

Table 7: Excitation table for D Flip Flop

S	**Q**
0	0
1	1

JK Flip Flop

One of the most useful and versatile flip flop is the JK flip flop the unique features of a JK flip flop are:

1. If the J and K input are both at 1 and the clock pulse is applied, then the output will change state, regardless of its previous condition.
2. If both J and K inputs are at 0 and the clock pulse is applied there will be no change in the output. There is no indeterminate condition, in the operation of JK flip flop i.e. it has no ambiguous state. The circuit diagram for a JK flip flop is shown in Figure 4.

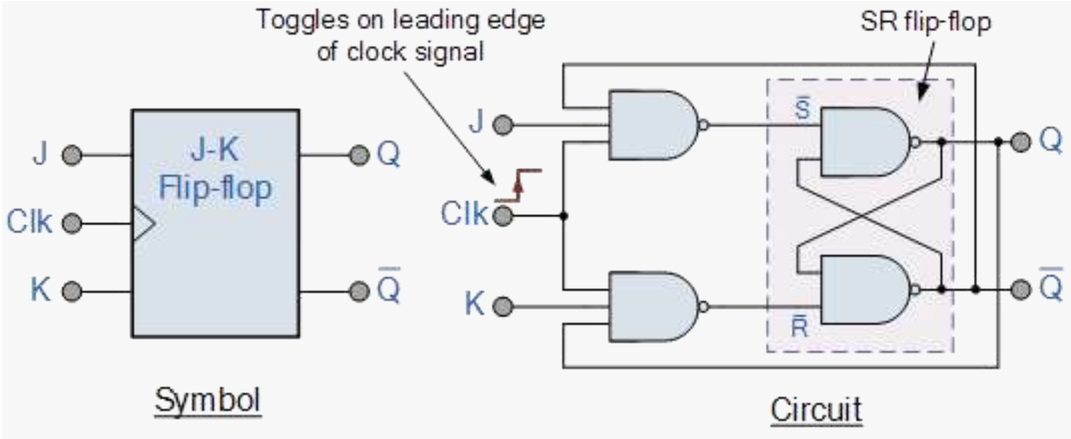


Figure 4: JK Flip Flop

When J = 0 and K = 0

These J and K inputs disable the NAND gates, therefore clock pulse have no effect on the flip flop. In other words, Q returns it last value.

When J = 0 and K = 1,

The upper NAND gate is disabled the lower NAND gate is enabled if Q is 1 therefore, flip flop will be reset (Q = 0 , =1)if not already in that state.

When J = 1 and K = 0

The lower NAND gate is disabled and the upper NAND gate is enabled if is at 1, As a result we will be able to set the flip flop (Q = 1, = 0) if not already set

When J = 1 and K = 1

If Q = 0 the lower NAND gate is disabled the upper NAND gate is enabled. This will set the flip flop and hence Q will be 1. On the other hand if Q = 1, the lower NAND gate is enabled and flip flop will be reset and hence Q will be 0. In other words , when J and K are both high, the clock pulses cause the JK flip flop to toggle. Truth table for JK flip flop is shown in table 8.

Table 8: The truth table for the JK flip flop

Initial Conditions	**Inputs (Pulsed)**		**Final Output**
Q	**S**	**R**	**Q (t + 1)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

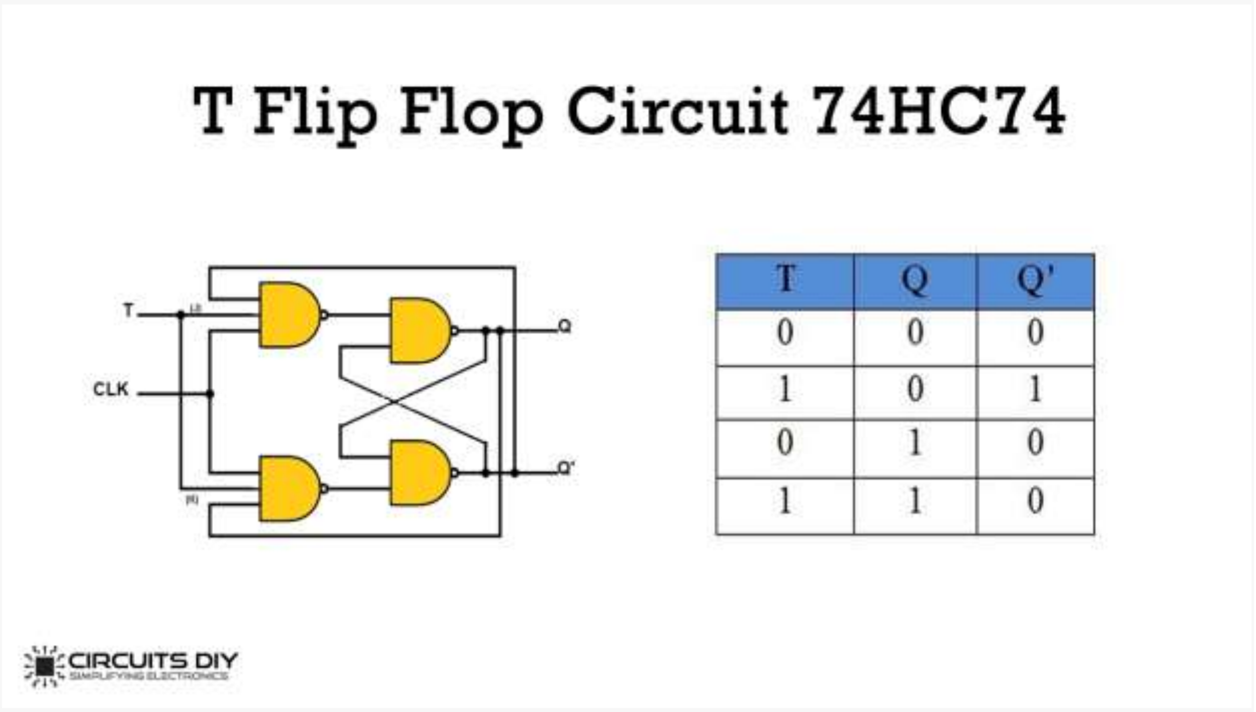
The excitation table for JK flip flop is very simply derived as given in table 9.

Table 9: Excitation table for JK Flip Flop

S	**R**	**Q**
0	0	No Change
0	1	0
1	0	1
1	1	Toggle

T Flip Flop

A method of avoiding the indeterminate state found in the working of RS flip flop is to provide only one input (the T input) such, flip flop acts as a toggle switch. Toggle means to change in the previous stage i.e. switch to opposite state. It can be constructed from clocked RS flip flop be incorporating feedback from output to input as shown in Figure 5.



T	Q	Q'
0	0	0
1	0	1
0	1	0
1	1	0

Figure 5: T Flip Flop

Such a flip flop is also called toggle flip flop. In such a flip flop a train of extremely narrow triggers drives the T input each time one of these triggers, the output of the flip flop changes stage. For instance Q equals 0 just before the trigger. Then the upper AND gate is enable and the lower AND gate is disabled. When the trigger arrives, it results in a high S input.

This sets the Q output to 1. When the next trigger appears at the point T, the lower AND gate is enabled and the trigger passes through to the R input this forces the flip flop to reset.

Since each incoming trigger is alternately changed into the set and reset inputs the flip flop toggles. It takes two triggers to produce one cycle of the output waveform. This means the output has half the frequency of the input stated another way, a T flip flop divides the input frequency by two. Thus such a circuit is also called a divide by two circuit.

A disadvantage of the toggle flip flop is that the state of the flip flop after a trigger pulse has been applied is only known if the previous state is known. The truth table for a T flip flop is as given table 7.

Table 7: Truth table for T Flip Flop

Q_n	T	$Q_n + 1$
0	0	0
0	1	1
1	0	1
1	1	0

The excitation table for T flip flop is very simply derived as shown in Table 8.

Table 8: Excitation table for T Flip Flop

T	Q
0	Q_n
1	$_n$

Generally T flip flop ICs are not available. It can be constructed using JK, RS or D flip flop. Figure 6 shows the relation of T flip flop using JK flip flop.

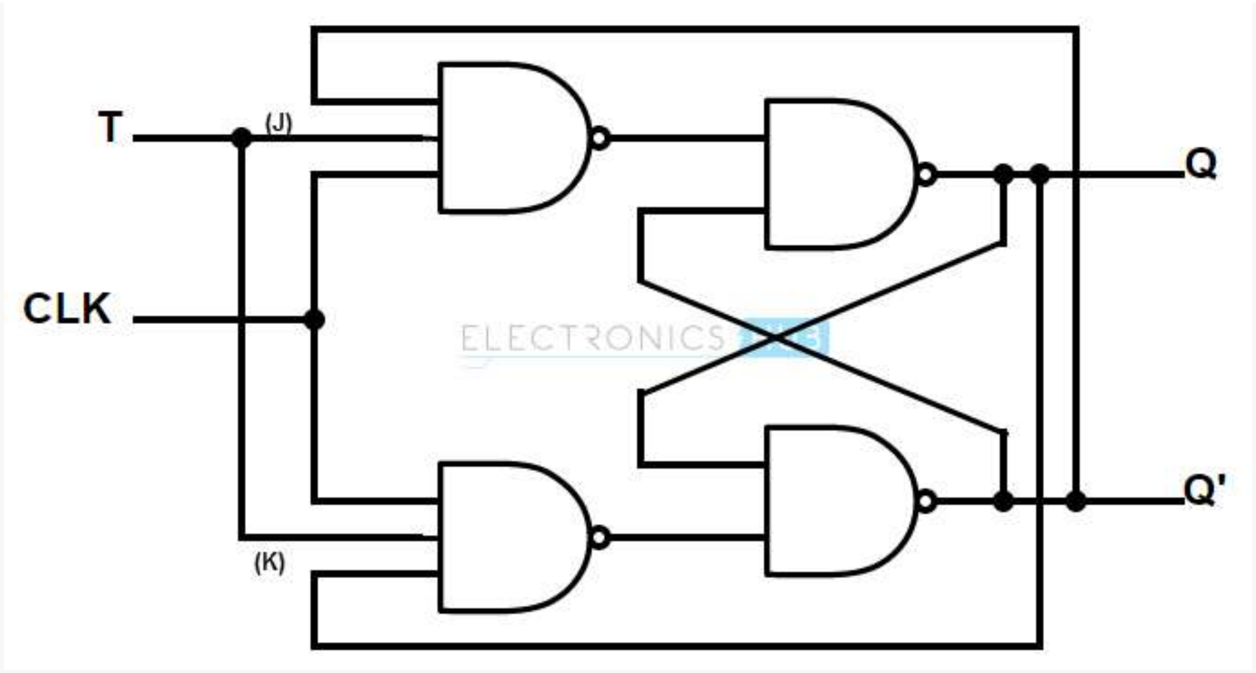


Figure 7: JK & D Flip Flop Connected as T Flip flop

A D-type flip flop may be modified by external connection as a T-type stage as shown in Figure 7. Since the Q logic is used as D-input the opposite of the Q output is transferred into the stage each clock pulse. Thus the stage having Q - 0 transistors q = 1, Providing a toggle action, if the stage had Q = 1 the clock pulse would result in Q = 0 being transferred, again providing the toggle operation. The D-type flip flop connected as in Figure 6 will thus operate as a T-type stage, complementing each clock pulse.

Master Slave Flip Flop

Figure 8 shows the schematic diagram of master slave J-K flip flop

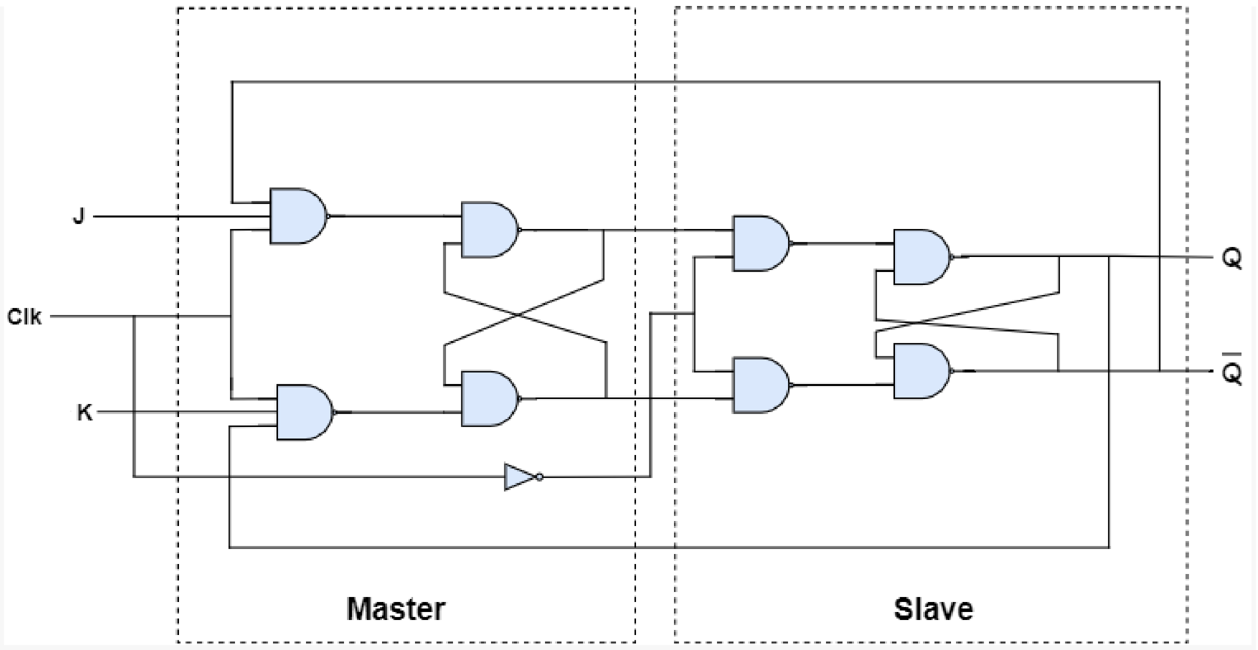


Figure 8: Master Slave JK Flip Flop

A master slave flip flop contains two clocked flip flops. The first is called master and the second slave. When the clock is high the master is active. The output of the master is set or reset according to the state of the input. As the slave is incative during this period its output remains in the previous state. When clock becomes low the output of the slave flip flop changes because it become active during low clock period. The final output of master slave flip flop is the output of the slave flip flop. So the output of master slave flip flop is available at the end of a clock pulse.