UNIT – 1

COMBINATIONAL CIRCUIT DESIGN:

NMOS IMPLEMENTATION OF SWITCH

For the purpose of understanding how logic circuits are built, we can assume that a transistor operates as a simple switch. figure 1.1a shows a switch controlled by a logic signal , x when x is low, the switch is open, and when x is high, the switch is closed. The most popular type of transistor field-effect transistor (MOSFET) There are two different types of MOSFERs, Known as n-channel, abbreviated NMOS, and p-channel, denoted PMOS.

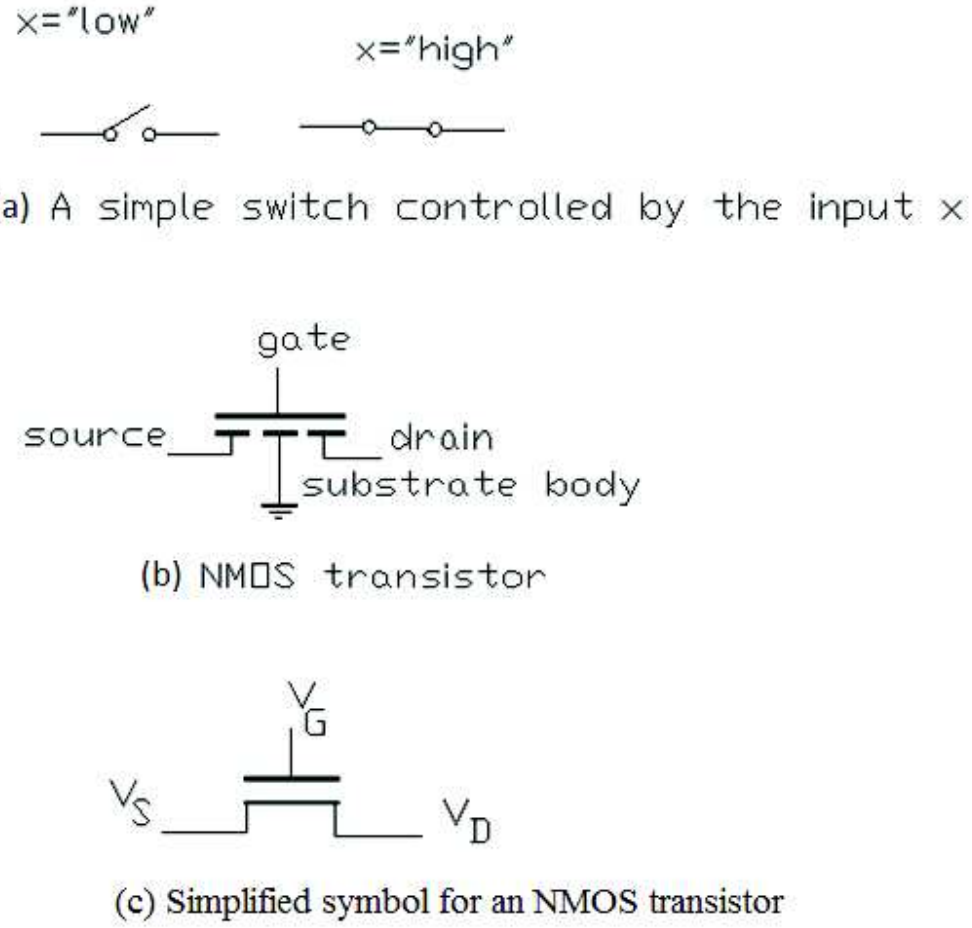


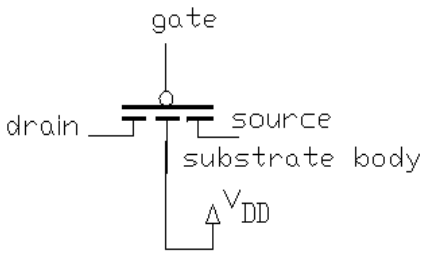
Figure 1.1 NMOS transistor as a switch

Figure 1.1b gives a graphical symbol for an NMOS transistor. it has four electrical terminals, called the source, drain, gate, and substrate. in logic circuits the substrate (also called body) terminal is connected to Gns. we will use the simplified graphical symbol in figure 1.1c, which omits the source and train terminals. They are distinguished in practice by the voltage levels applied to the transistor. The terminal with the lower voltage level is assumed as source.

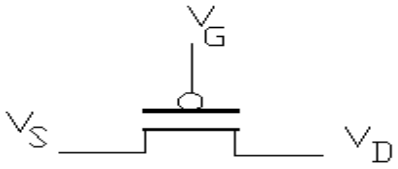
If Vg is low, Then there is no connection between the source and drain, the transistor is turnetoff.If Vg is high, then the transistor is turned on and acts as a closed switch that connects the source and train terminals.

PMOS transistors have the opposite behavior of NMOS transistors. the type of switch is open when the control input x is high and closed when x is low . A symbol is shown in figure 1.2b.

(a) A Switch with the opposite behavior of Figure 1.2a



1.2 (b) PMOS transistor



1.2 (c) Simplified symbol for an PMOS transistor

In logic circuits the substrate of the PMOS transistor is always connected to V_{DD} leading to the simplified symbol in figure 1.2c. if Vg is high, then the PMOS transistor is turned on and acts as a closed switch that connect the source and drain. In the PMOS transistor the source is the node with the higher voltage.

Figure 1.3 summarizes the typical use of NMOS and PMOS transistor in logic circuits. An NMOS transistor is turned on when its gate terminal is high.

A PMOS transistor is turned on when the NMOS transistor is turned on, its drain is pulled down to Gns, and when the PMOS transistor is turned on its drain is pulled up to V_{DD}.

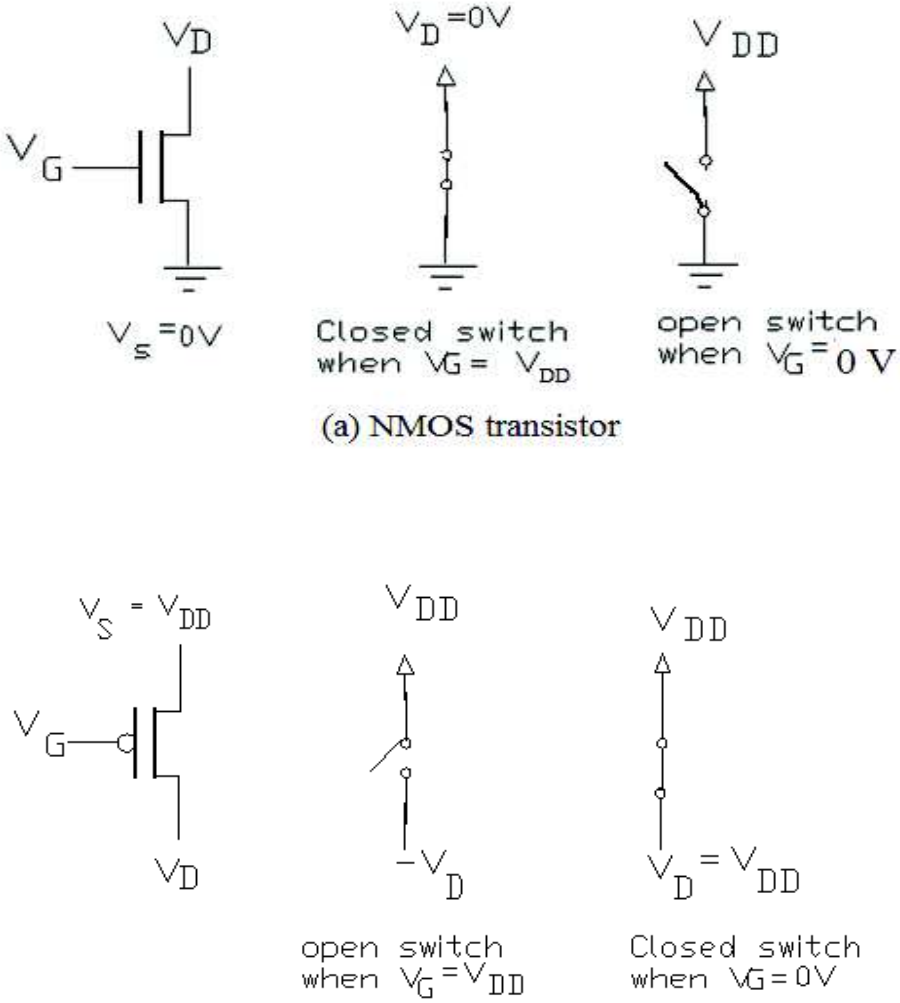


Figure 1.3 NMOS and PMOS transistor in logic circuits.

NMOS Implementation of NOT Gate

NMOS Implementation of NOT Gate in the circuit in figure 1.4a, when Vx = 0v, the NMOS transistor is turned of f. No current flows through the resistor R, and Vf to a low voltage level

If Vf is viewed as a function of Vx then the the circuit is an NMOS implementation of a NOT gate. in logic terms this circuit implements the function f = x Figure 1.4b gives a simplified circuit diagram in which the connection to the positive terminal on the power supply is indicated by an arrow labeled Vdd and the connection to the negative power supply terminal is indicated by the Gnd symbol.

Figure 1.4c presents the graphical symbols for a NOT gate. The left symbol shows the input, output, power, and ground terminals, and the right the symbol shows only the input and output terminals. In practice only the simplified symbol is used. another name often used for the NOT gate is inverter.

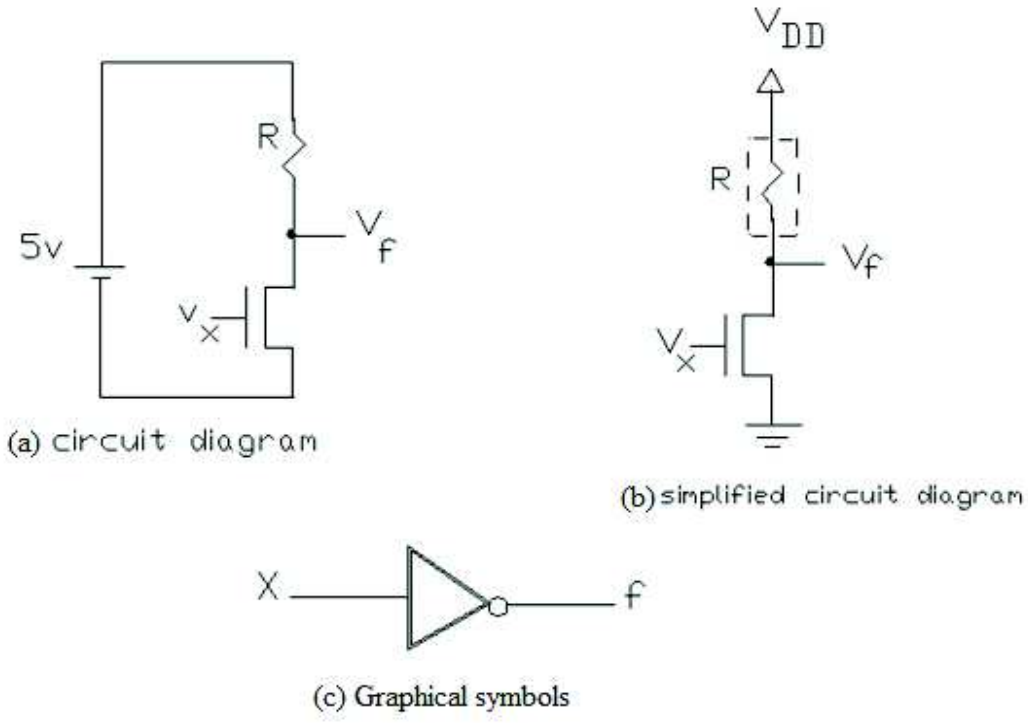
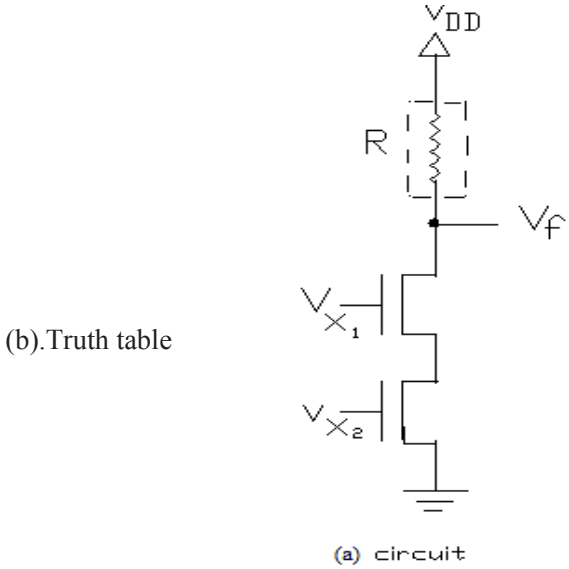


Figure 1.4 A NOT gate built using NMOS technology

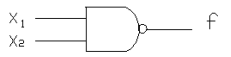
NMOS Implementation of NAND Gate

Using NMOS transistor, we can implement the series connection as depicted in figure 1.5a. If $V_{x1} = V_{x2} = 5V$, both transistors will be on and V_f will be close to 0V. But if either V_{x1} or V_{x2} is 0, then no current will flow through the series – connected transistors and V_f will be pulled up to 5V. The resulting truth table for f, provided in terms of logic values, is given in figure 1.5b. Its graphical symbols are shown in figure 15c.



(b).Truth table

		f
0	0	1
0	1	1
1	0	1
1	1	0

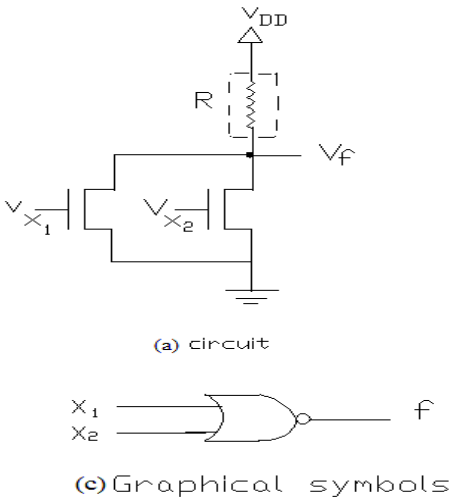


(c) Graphical symbols

Figure 1.5 NMOS realization of a NAND Gate.

NMOS Implementation of NOR Gate :

The parallel connection of NMOS transistors is given in Figure 1.6a. Here, if either $V_{x1} = 5$ or $V_{x2} = 5$ V, then V_{x2} will be close to 0 V. Only if both V_{x1} and V_{x2} are 0 will V_f be pulled up to 5V . A corresponding truth table is given in Figure 1.6b. The graphical symbols for the NOR gate appear in Figure 1.6c.



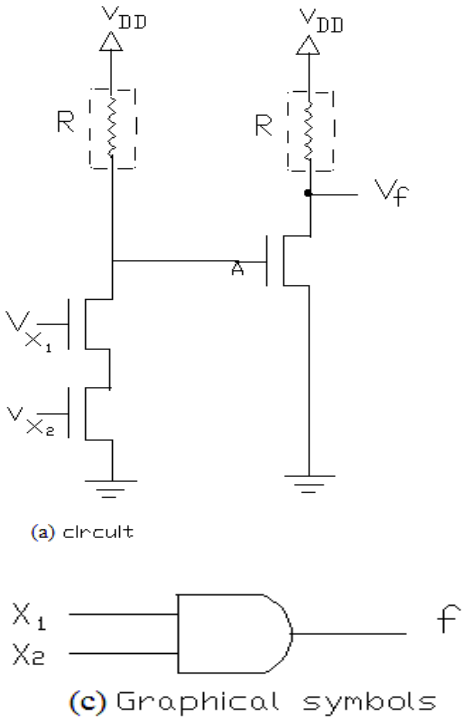
		f
0	0	1
0	1	0
1	0	0
1	1	0

(b).Truth table

Figure 1.6 NMOS realization of a NOR gate.

NMOS implementation of AND Gate

Figure 1.7 indicates how an AND gate is built in NMOS technology by following a NAND gate with an inverter. Node A realizes the NAND of inputs x_1 and x_2 and f represents the AND function.



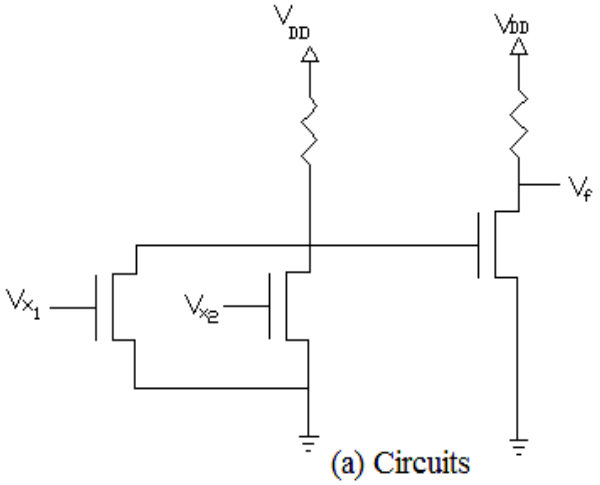
		F
0	0	0
0	1	0
1	0	0
1	1	1

(b).Truth table

Figure 1.7 NMOS realization of an AND gate.

NMOS implementation of OR Gate

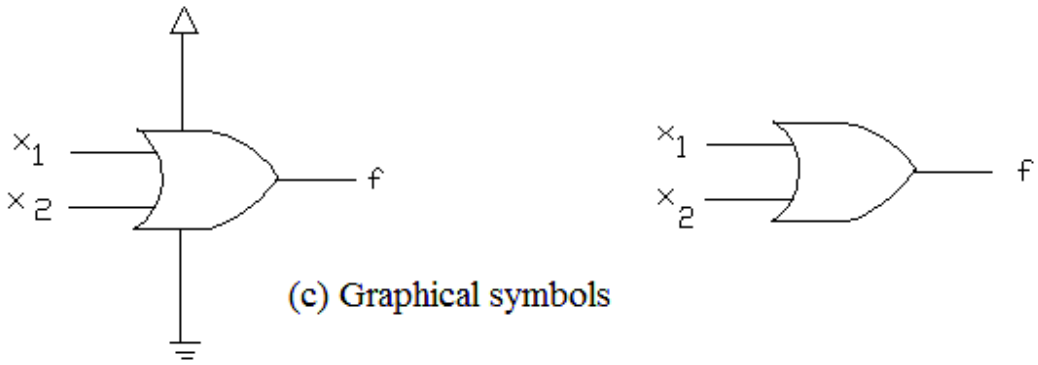
Figure 1.52 indicates how an or gate is built in NMOS technology by following NOR Gate with an inverter



(a) Circuits

		f
0	0	0
0	1	1
1	0	1
1	1	1

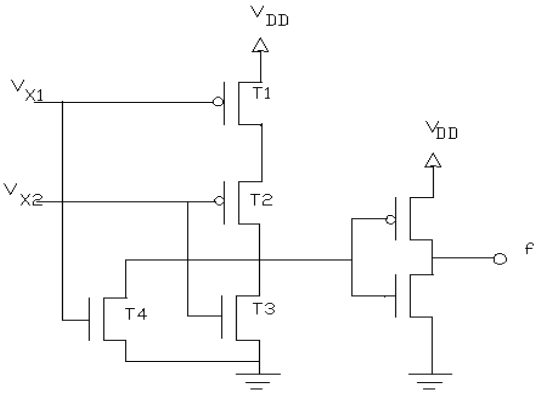
(b) Truth Table



(c) Graphical symbols

CMOS OR Gate

A CMOS OR gate is built with a NOR gate followed by a NOT gate.

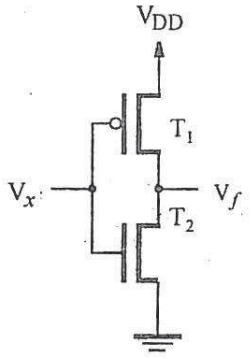


		f
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1.8a Circuit 1.8 b.Truth Table

CMOS NOT Gate:

The simplest example of a CMOS circuit, a NOT gate, is shown in figure 1.9. whenVx = 0 v, transistor T2 is off and transistor T1 is on This makes Vf = 5v, and since T2 is off and no current flows through the transistor. When Vx = 5V, T2 is on and T1 is off ThusVf = 0v, and no current flows because T1 is off



(a) Circuit

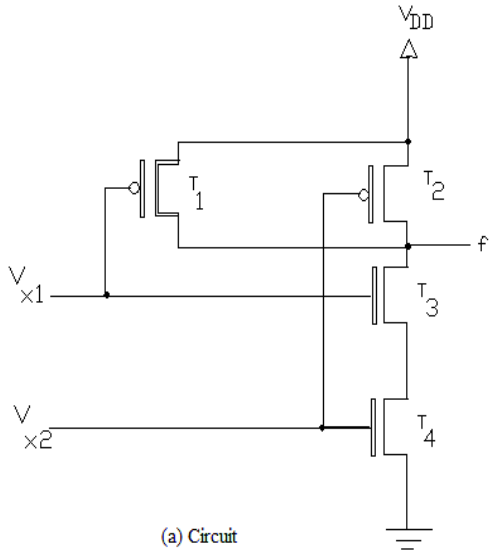
x	T_1	T_2	f
0	on	off	1
1	off	on	0

(b) Truth table and transistor states

Figure 1.9a Circuit 1.9b Truth table and transistor states

CMOS NAND Gate

Figure 1.10 shows a circuit diagram of CMOS NAND gate. The truth table in the figure specifies the state of each of the four transistors for each logic valuation of inputs X1 and X2 The circuit properly implements the NAND function Under static conditions no patch exists for current flow from Vdd to Gnd.



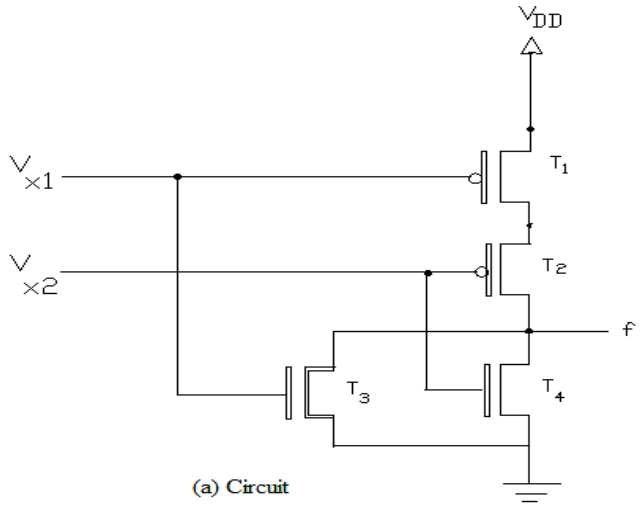
						f
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

(b).Truth Table

Figure 1.10 CMOS realization of a NAND gate.

CMOS NOR Gate

The circuit for a CMOS NOR gate is shown in Fig. 1.11. This Circuit functions as per the truth table.



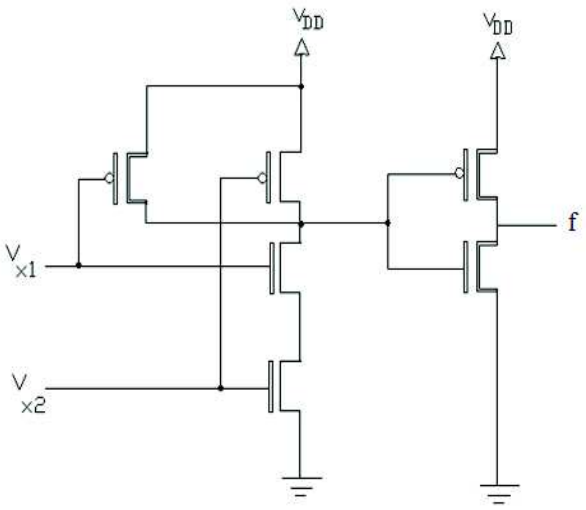
						f
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

(b).Truth Table

Figure 1.11 CMOS realization of a NOR gate.

CMOS AND Gate

A CMOS AND gate is built by connecting a NAND gate to an inverter, as illustrated in figure 1.12. Similarly, an OR gate is constructed with a NOR gate followed by a NOT gate.



		f
0	0	0
0	1	O
1	0	O
1	1	1

(b).1.12 b Truth Table

Figure 1.12a CMOS realization of a AND gate.

CMOS TRANSMISSION GATE

Basic Operation

A transmission gate, or analog switch, is defined as an electronic element that will selectively block or pass a signal level from the input to the output. This solid-state switch is comprised of a pMOS transistor and nMOS transistor. The control gates are biased in a complementary manner so that both transistors are either on or off.

When the voltage on node A is a Logic 1, the complementary Logic 0 is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a Logic 0, the complementary Logic 1 is applied to node A, turning both transistors off and forcing a high-impedance condition on both the IN and OUT nodes. This high-impedance condition represents the third "state" (high, low, or high-Z) that the DS3690 channel may reflect downstream.

The schematic diagram (**Figure 1**) includes the arbitrary labels for IN and OUT, as the circuit will operate in an identical manner if those labels were reversed. This design provides true bidirectional connectivity without degradation of the input signal.

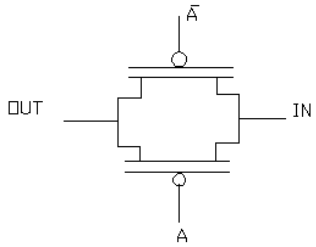


Figure 1. Schematic representation of a transmission gate.

The common circuit symbol for a transmission gate depicts the bidirectionalnature of the circuit's operation (**Figure 2**).

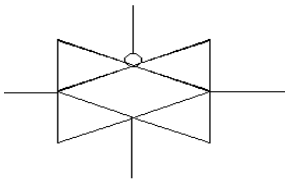


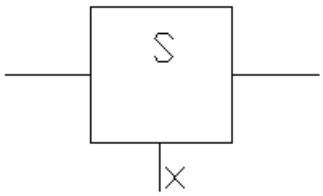
Figure 2. Circuit symbol.

Digital Logic Variables & Functions

In digital systems, binary circuits are used because the binary element is switch that has two states if a given switch is controlled by an input variable x, then we will say that the switch is open if x = o and closed if x=1, as illustrated in figure 1.13a. The graphical symbol in figure 1.13b

Figure 1.13a Two staes of a switch

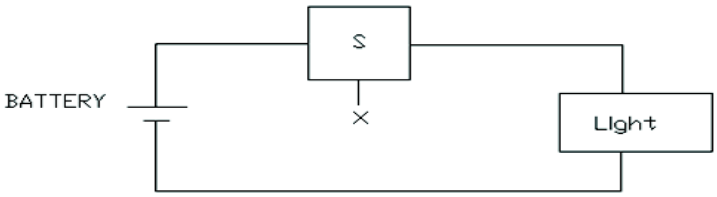
to represent such switches in the diagrams that follow Note that the control input x is shown explicitly in the symbol.



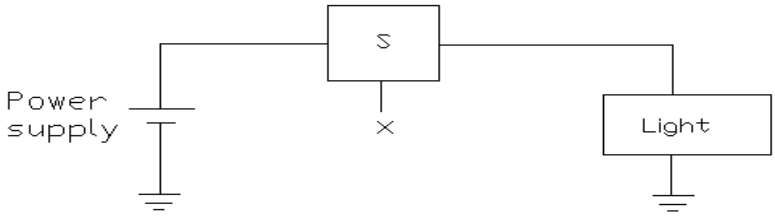
(b) Symbol for a switch

Figure 1.13b binary switch

Consider a simple application of a switch turns a small light bulb on or off this action is accomplished with the circuit in figure 1.14a. A battery provides the power source The current flows when the switch is closed, that is, when x = 1. In this example the input that causes changes in the behavior of the circuit is the switch control x.



(a) Simple connection to a battery



(b) Using a ground connection as the return path

Figure1.14 A light controlled by a switch

The output is defined as the state (or condition) the light, which we will denote by the letter L. if the light is on, we will say that L=1.if the light is off, L=0. using this convention, we can describe the state of the light as a function of the input variable x. since L=1 if x=1 and l=0 if x=0, we can say that

$$L(x) = x$$

We say that L(x) = x is a logic function and that x is an input variable.

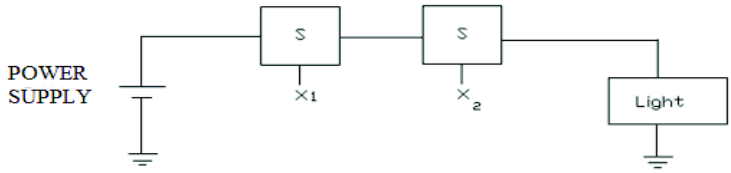
The circuit in figure 1.14a in an ordinary flashlight, where the switch is a simple mechanical device. in an electronic circuit the switch is implement as a transistor and the light may be a light-emitting diode (LED). An electronic circuit is powered by a power supply of a certain voltage, like 5 volts, One side of the power supply is connected to ground, as shown in figure 1.14b. the ground connection is used as the return path for current, to close the loop. This is achieved by connecting one side of the light to ground as indicated in the figure.

Consider now the possibility of the using two switches to control the state of the light let x1 and x2 be the control inputs for these switches the switches can be connected either in series or in parallel as shown in figure 1.15. using a series connection, the light will be turned on only if both switches are closed. if either switches are closed if either switch is open the light will be off. this behavior can be described by the expression

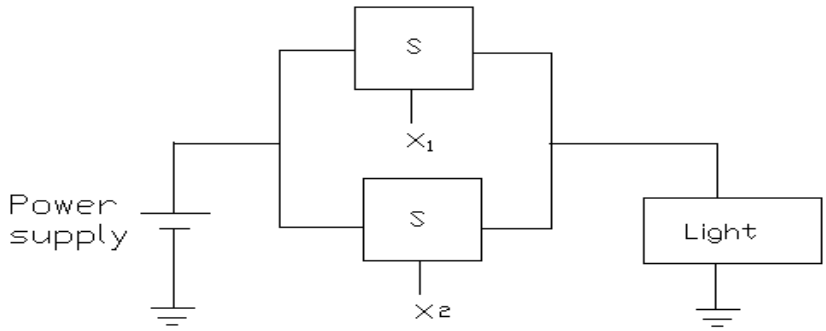
$$L(x_1, x_2) = x_1.x_2$$

Where L = 1 if $x_1 = 1$ and $x_2 = 1$,

L = 0 otherwise.



(a) The logical AND function (series connection)



(b) The logical OR function (parallel connection)

Figure 1.15 Two basic functions.

The "." symbol is called the AND operator, and the circuit in figure 1.15a is said to implement a logical AND function

The parallel connection of two switches is given in figure 1.15b. in this case the light will also be off only if both switches are open. This behavior can be on if either x1 or x2 switch is closed. The light will also be on if both switches are open . This behavior can be stated as

$$L(x1, x2)=x1+x2$$

Where L=1 if x1=1 or if x1=x2=1,

L=0 if x1=x2=0.

The + symbol is called the OR operator, and the circuit in Figure 1.15b is said to implement a logical of function

In the above expressions for AND and OR, the output L(x1,x2) is a logic function with input variables x1 and x2 the AND and OR functions are two of the most important logic functions. Together with some other simple function they can be used as building blocks for the implementation of all logic circuits. figure 1.16 illustrates how there switches can be used control the light in a more complex way. This series-parallel connection of switches realizes the logic function

$$L(x1, x2, x3)=(x1+x2.x3)$$

The light is on if x2, = 1 and, at the same time, at least one of the x1 or x2 inputs is equal to 1.

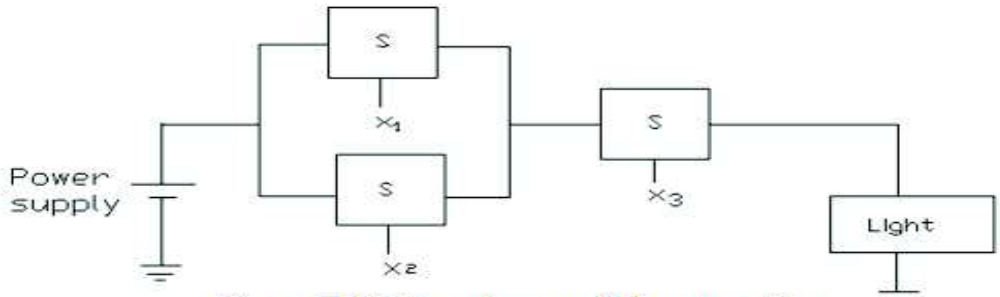


Figure 1.16 A series parallel connection

Inversion

A positive action takes place when a switch is opened. Suppose that we connect the light as shown in Figure 1.17. in this case the switch is connected in parallel with the light, rather than I series. Consequently, a closed switch will short-circuit the light and prevent the current from following through it. an extra resistor in this circuit dose not short-circuit the power supply. The light will be turned on when the switch is opened. formally, we express this functional behavior as

$$L(x)= x$$

Where $L=1$ if $x=0$,

$$L=0 \text{ if } x = 1$$

The value of this function is the inverse of the value of the input variable instead of using the word inverse, it is more common to use the term complement. thus we say that L(x) is a complement of x in this example another frequently used term for the same operation is the NOT operation.

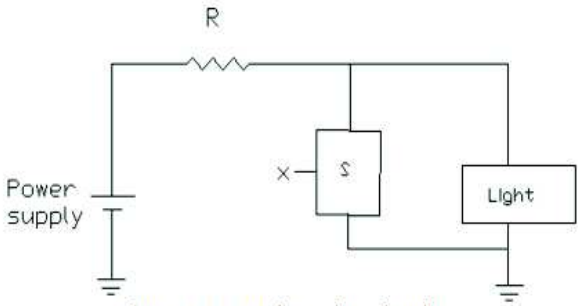
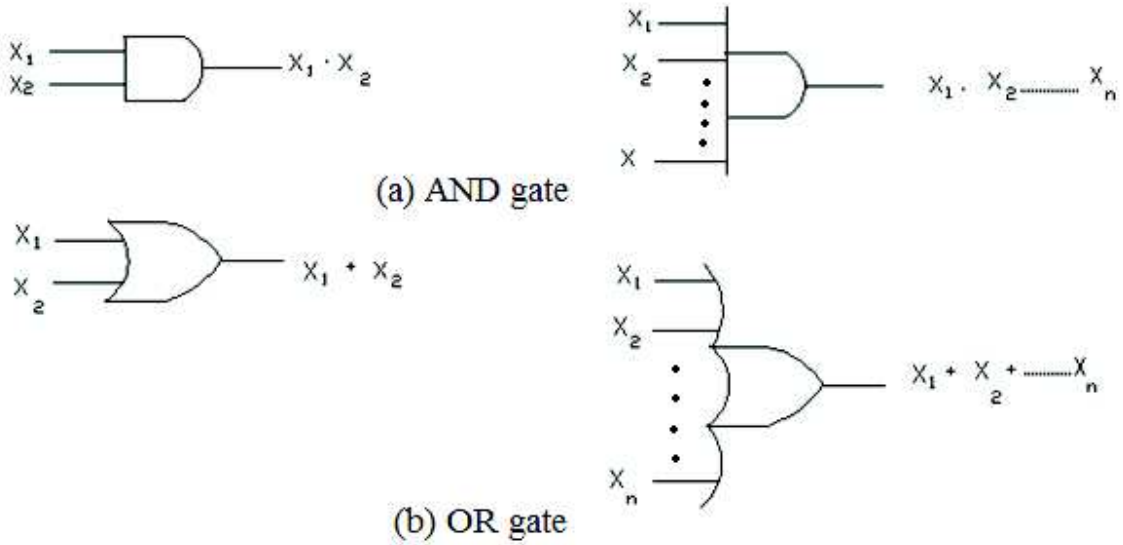


Figure 1.17 An inverting circuit

Logic Gates and Networks

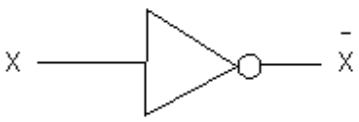
Each logic operation can be implemented with transistors, resulting In a circuit element called logic gate a logic gate has one or more inputs and output that is a function of its inputs. A logic circuit diagram, consisting of graphical symbols representing the logic gates. the graphical symbols for the AND, OR, and NOT gates are shown in Figure 1.18. The figure indicates on the left side how the AND and OR gates are drawn when there are only a few inputs. On the right side it shows how the symbols are enlarged to accommodate a greater number of inputs.

A larger circuit is implemented by a network of gates for example, the logic function from figure 1.19. A given logic function can be implemented with a number of different networks.



(a) AND gate

(b) OR gate



(c) NOT gate

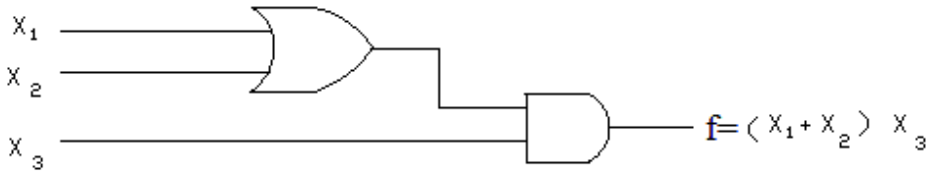


Figure 1.19 The function from Figure 1.18

BOOLEAN ALGEBRA

In 1849 George Boole published a scheme for the algebraic description of processes. it involved in logical thought and reasoning this scheme and its further refinements became known as Boolean algebra provides It was almost 100 years later that this algebra found application in the engineering sense. in the late 1930s claude Shannon showed that Boolean algebra provides an effective means of describing circuits built with switches. The algebra can, therefore, be used to describe logic circuits This algebra is a powerful tool that can be used for designing and analyzing logic circuits

Axioms of Boolean Algebra

Like any algebra, Boolean algebra is based on a set rules that are derived from a small number of basic assumptions are called axiom let us assume that Boolean algebra values, 0 and 1. Assume that the following axioms are true

1a 0.0=0

1b 1+1=2

2a 1.1=1

2b 0+0=0

3a 0.1=1.0=0

3b 1+0=0+1=1

4a If x = 0, then x = 1

4b If x= 1, then x=0

Single-Variable Theorems

From the axiom we can define some rules for single variables. these rules are often called theorems if x is variables in B, the n the following thermos hold:

5a. x.o=0

5b. x+1=1

6a. x.1=x

6b. x+0=x

7a. x.x=x

7b. x+x=x

8a. x.x=0

8b. x+x=1

9. x=x

it is easy to prove the validity of these theorems by substituting the values x=o and x=1 into the expressions and using the axioms given above. for example, in theorem 5a, if x = 0, then the theorem states that that 0.0 =0, which is true according to axiom la similarly, if x = 1, then theorem 5a status that 1.0 = 0, which is also true according to axiam 3a.

Duality

Given a logic expression, its dual is obtained by replacing all+ operators, and vice versa, and by the replacing all 0s with 1s, and vice versa. The dual of any true statement (axiom or theorem) in Boolean algebra is also a true statement.

Two-and Three – variable Properties

If x,y, and z are the variables in B, then the following properties hold:

10a.	x.y = y.x	Commutative
10b.	x+y = y+x	
11a.	x.(y.z) = (x.y).z	Associative
11b.	x+(y+z) = (x+y) +z	
12a.	x.(y+z) =x.y + x.z	Distributive
12b.	x+y.z = (x+y). (x+z)	
13a.	x+x.y = x	Absorption
13b.	x. (x+y) = x	
14a.	x.y + x.y = x	Combining
14b.	(x+y). (x+y) = x	
15a.	x.y. = x+y	De Morgan's theorem
15b.	x+y = x .y	
16a.	x+x .y = x+y	
16b.	x. (x+y) = x.y	
17a.	x. y+y. z+x . z=x .y+x.z	Consensus
17b.	(x+y). (y+z). (x+z) = (x+y). (x+z)	

INPUT		LHS		RHS		
x	**y**					
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Figure 1.20 Proof of Demorgan's theorem in 15a.

Again, we can prove the validity of these properties either by perfect induction or by performing algebraic manipulation. figure 1.20 illustrates how perfect induction of a truth table. The evaluation of left-hand and right-hand sides of the identity in 15 a gives the same result

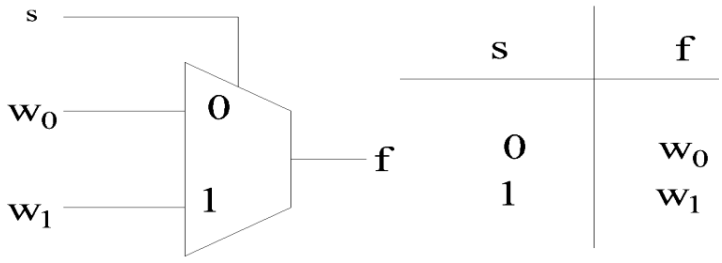
1.2. COMBINATIONAL CIRCUIT BUILDING BLOCKS

MULTIPLEXERS

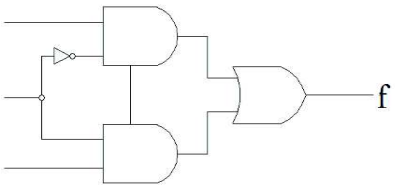
A multiplexer circuit has a number of data inputs, one or more select inputs, and one output. it passes the signal value on one of the data inputs to the output. the data input is selected by the values of the select inputs figure1.21shows a 2-to1 multiplexer

part 1.21(a) gives the symbol commonly used the select input,s, chooses as the output of the multiplexer either input W0 or W1. the multiplexer's functionality can be described in the form of a truth table as shown in part 1.21b of the figure part 1.21(c) gives a sum-of-products implementation of the 2 to 1 multiplexer and part 1.21(d) illustrates haw can be constructed with transmission gates.

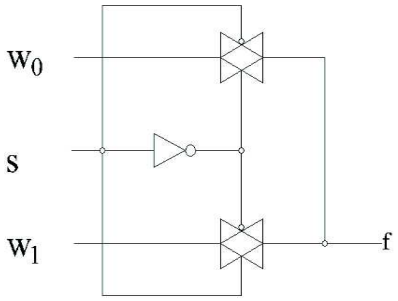
Figure 1.22a shows a – larger multiplexer with four data inputs , w_0, ….., w_3 and two select inputs, s_1 and s_0. As shown in the truth table in part (b) of the figure, the two-bit number represented by s_1s_0 selects one of the data inputs as the output of the multiplexer.



(a)graphical symbol (b)truth table

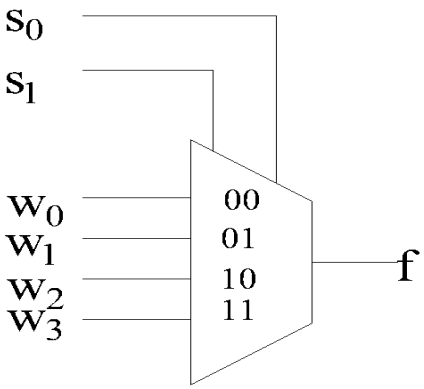


(c)sum -of-products circuit



(d)circuit with transmission gates

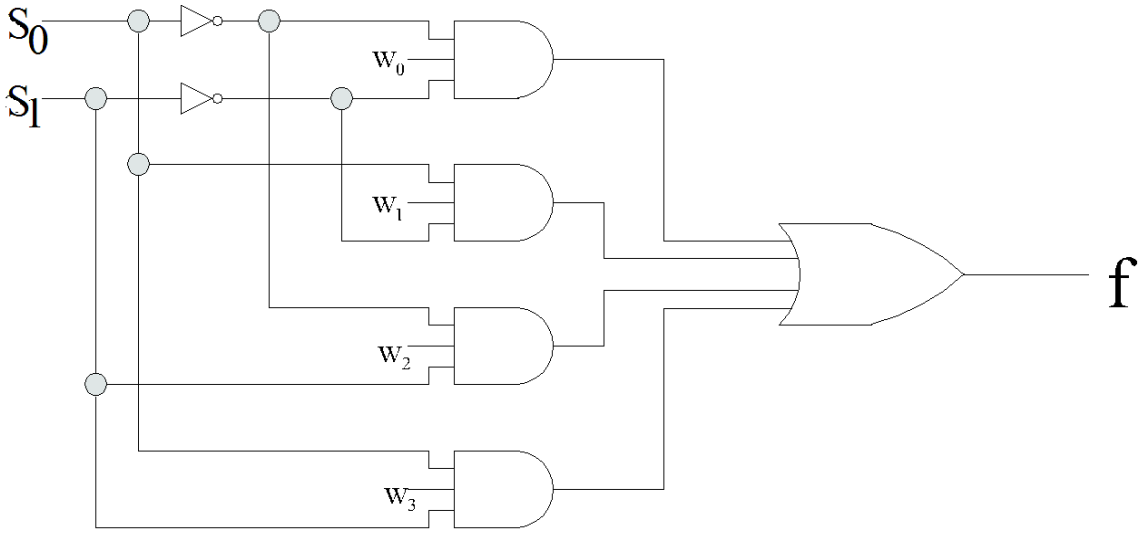
Figure 1.2 A 2-to-1 MULTIPLEXER



(a)graphical symbol

s_0 s_1	f
00	w_0
01	w_1
10	w_2
11	w_3

(b)truth table



(C) CIRCUIT

Figure 1.22 1 A 4-to-1 MULTIPLEXER

A sum – of – products implementation of the 4 to multiplexer appears in figure 1.22c. It realizes the multiplexer function.

$$F = s_1s_0w_0 + s_1s_0w_1 + s_1s_0w_2 + s_1s_0w_3$$

It is possible to build larger multiplexers using the same approach. Usually, the number of data inputs, n is a integer power of two. A multiplexer that has n data inputs w_0, ….., w_{n-1}, requires $[\log_2 n]$ select inputs. Larger multiplexer can also be constructed from smaller multiplexers. For examples , the 4 to 1 multiplexer can be built using three 2 to -1 multiplexers as illustrated in figure 1.23.

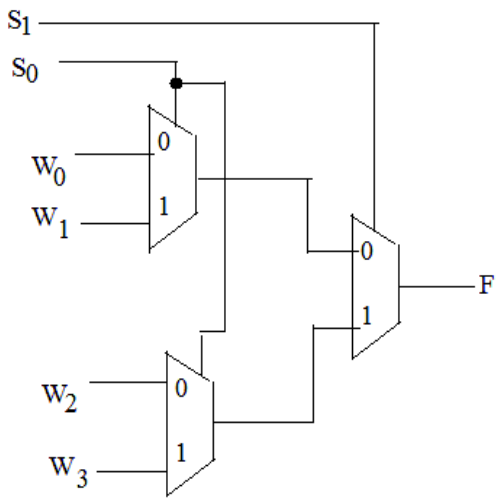


FIGURE 1.23 USING 2-to-1 MULTIPLEXER TO BUILD A 4-to-1 MULTIPLEXER

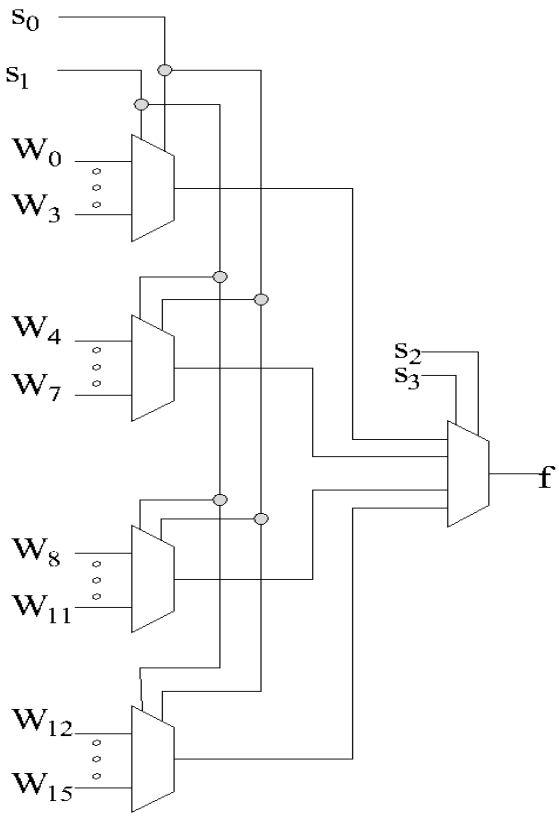


Figure 1.24 shows how a16 to 1 multiplexer is constructed with five 4 to 1 multiplexer

DEMULTIPLEXERS

The purpose of the multiplexer circuit is to multiplex then n data inputs onto the single data output under control of the select inputs

A circuit that performs the opposite function, namely, placing the value of a single data input onto multiple data outputs is called a **demultiplexer.** Thedemultiplexer can be implemented using a decoder circuit.

DECODERS

Decoders circuits are used to decode encoded information A binary decoder shown in the figure 1.25 is a logic circuit with n inputs and 2n outputs Only one outputs is asserted at a time, and each output corresponds to one valuation of the inputs .

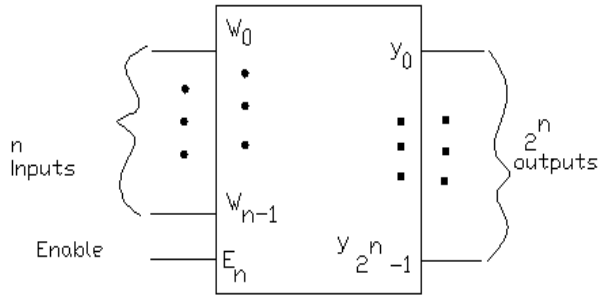
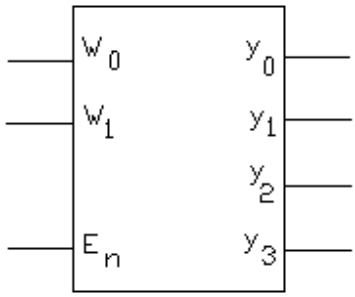


Figure 1.25 An n-to-2^n binary decoder

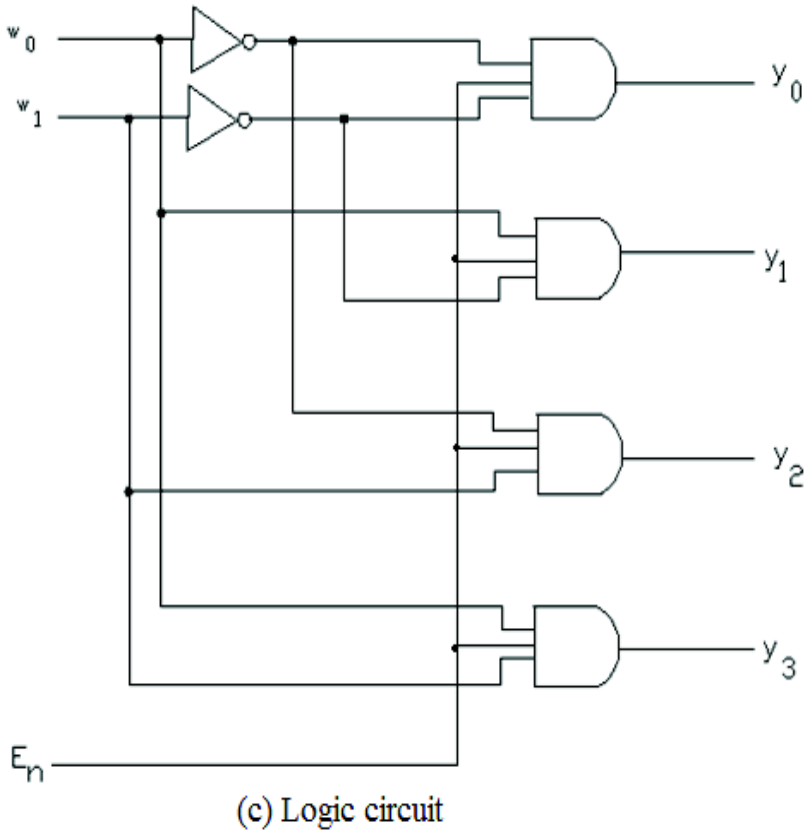
The decoder also has an enable input. En, that is used to disable the outputs; if En = 1, the valuation of wn-1….. w1w0 determines which of the outputs Is asserted

1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

(a) Truth Table



(b) Graphic symbol



(c) Logic circuit

Figure 1.26 A-2-to-decoder

For example, the 2-to4 decoder in Figure 1.26 can be used as a 2-to4 Demultiplexer in this case the en input serves as the data input for the Demultiplexer, and they y0 to y3 outputs are the data input the valuation of w1 w0 determines which of the outputs is set to the value of En.

To see how the circuit works, consider the truth table in figure 1.26a. when En=0, all the outputs are set to 0, including the one selected by the valuation of w1w0 sets the appropriate to 1.

ENCODERS

An encoder performs the opposite function of a decoder in encodes given information into a more compact from.

BINARY ENCODERS

A binary encoder encodes information from 2n inputs into an n-bit code, as indicated in figure 1.27. exactly one of the input signals should have a value of 1, and the outputs present the binary number that identifies which input is equal to 1.

The truth table for a 4 to 2 encoder is provided in figure 1.27b. observe that the output y0 is 1 when either input y0 is 1 when either input w1 or w3 Is 1, and output y1 is 1 when input w2 or w3 is 1. Hence these outputs can be generate by the circuit Figure 1.27c

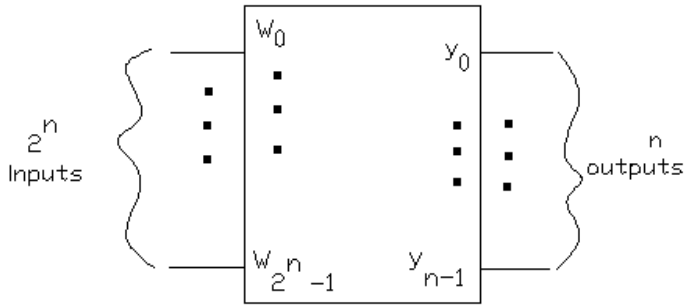


Figure 1.27 A 2^n-to-n binary encoder

Encoders are used to reduce the number of the bits needed to represent given information A practical use of encoders is for transmitting information in a digital system.

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

(b) Truth Table

BINNARY ADDER – SUBTRACTOR

A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is a full adder Two half adders can be employed to implement a full adder.

A binary adder subtractor is combinational circuit that performs the arithmetic operation of addition and subtraction with binary numbers the helf adder designs carried out first, from which we developthe full adder for two n bit numbers the subtraction circuit is included a complementing circuit

HALF ADDER

A half adder, needs two binary inputs and two binary outputs. the input variables designate the augends and addend bits; the output variables produce the sum and carry we assign symbols x and y to the two inputs and s (for sum) and C (for carry) to the outputs the block diagram of a half adder is shown in fig. 1.28. the truth table for the half adder is listed in Table 4.1. The c output represents the least significant bit of the sum.

The simplified Boolean functions for the two outputs can be obtained directly from the truth table the simplified sum of products expressions are the logic diagram of the half adder implemented in sum of products is shown in fig. 1.29(a) it can be also implemented with an exclusive OR and an AND gate as shown in fig 1.29(b). This from is used to show that two half adders can be used to construct a full adder.

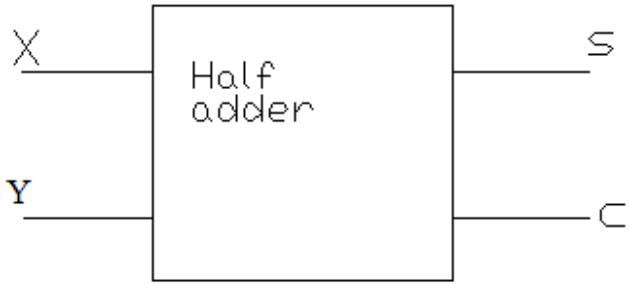


Figure1.28 Block diagram of half adder

Truth Table of Half Adder

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

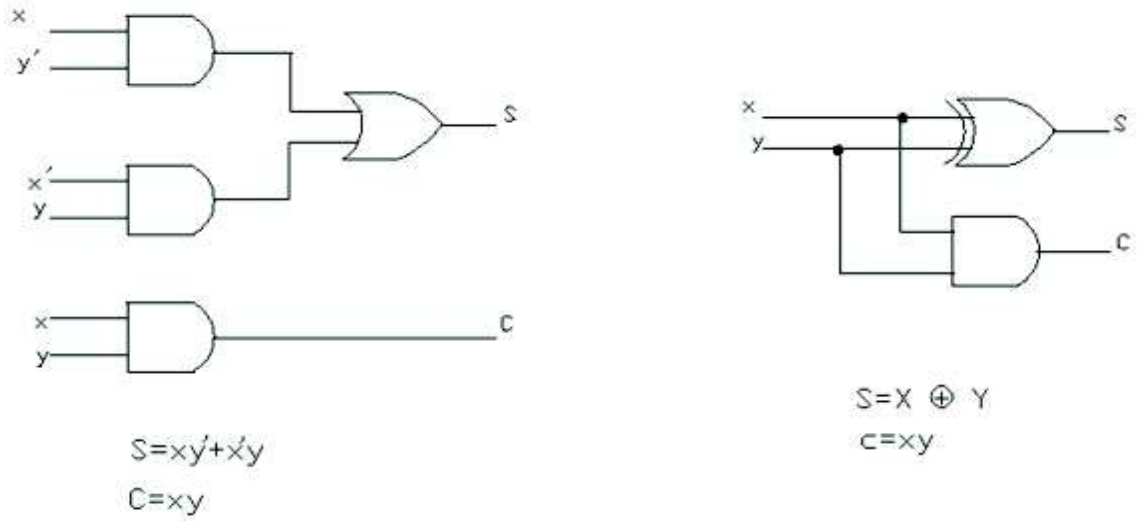


Figure 1.29 Implementation of half adder

FULL ADDER

A full adder is a combinational circuit that forms the arithmetic sum of three bits it consists of three inputs and two output. two of the input variables denoted by x and represents the two bits to be added the third input, z, represents the carry from the previous lower significant position

The block diagram of a full adder is shown in fig.1.30.

The truth table of the full adder is listed in table 1.2. the eight rows under the input variables designate all possible combinations of the three variables the output variables are determined from the arithmetic sum of the input bits.

When all input bits are 0, the output is 0. the S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to1. The c output has a carry of 1 if two or three inputs are equal to 1.

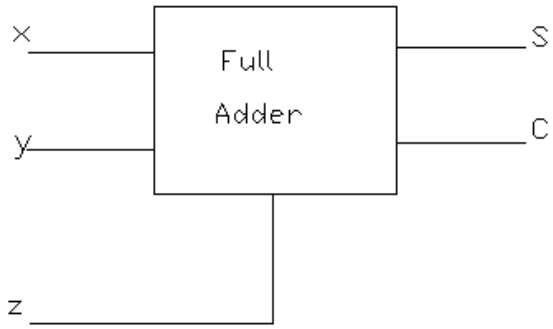


Figure 1.30 Block diagram of full adder

X	Y	Z	C	S
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

The maps for the outputs of the full adder are shown in fig.1.31 the simplified expressions are

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

The logic diagram for the full adder implemented in sum-of-products form is shown in fig 1.32. it can also be implemented with two half address and one OR gate, as shown in fig.1.33. the S output from the second half adder, giving

$$S = z \oplus (x \oplus y)$$

$$= z'(xy' + x'y) + z(xy' + xy)'$$

$$= z'(xy' + x'y) + z(cy' + x'y')$$

$$= xy'z' + x'yz' + xyz + x'y'z$$

The carry output is

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

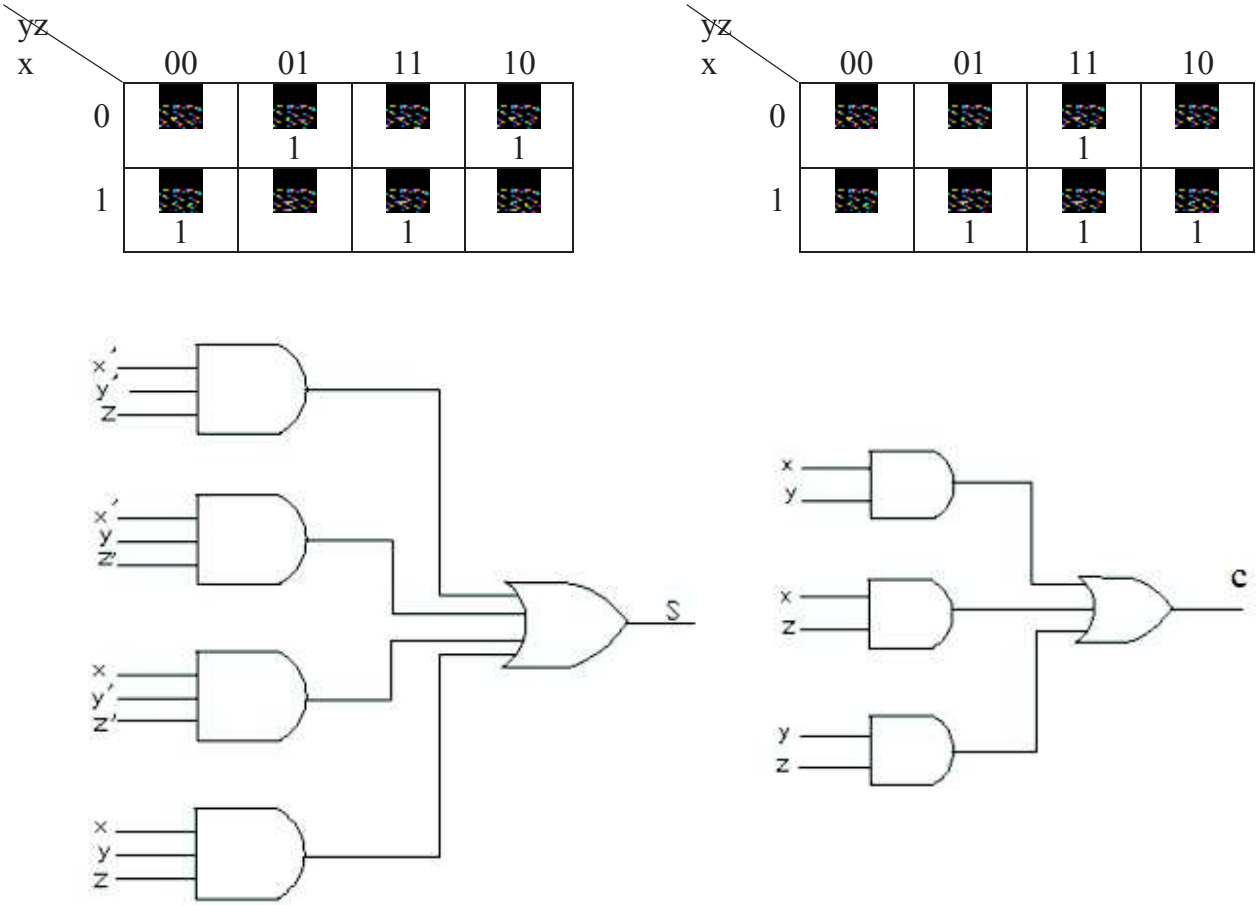


Figure 1.32 Implementation of full adder in sum-of- products

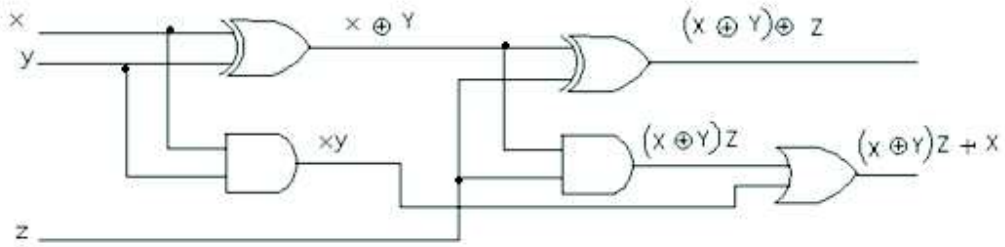


Figure 1.33 Implementation of full adder with two half adders an OR gate

Binary Adder

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. it can be constructed with full adders connected in cascade, with the output carry from each full adders connected to the input carry of the next of four full adder (FA) circuits to provide a four-bit binary ripple carry adder.

The input carry to the adder is C0, and it ripples through the full adders to the full adders to the output carry c4. the S outputs generate the required sum bits.

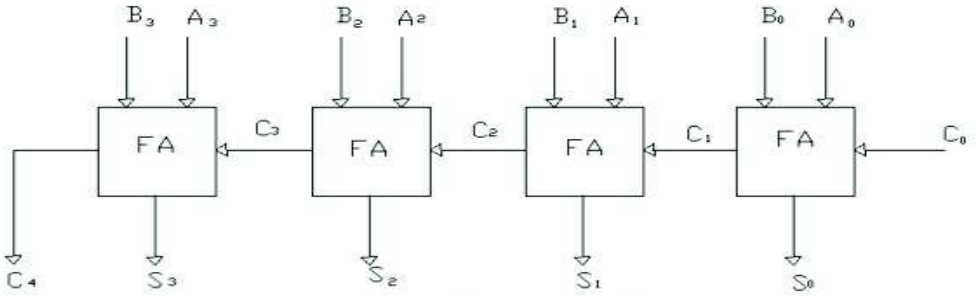


Figure 1.34 Four -bit adder

To demonstrate with a specific example, consider the two binary numbers A=1011and B=0011 Their sum S=1110 is formed with the four-bit adder as follows:

Subscript i:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	
Augends	1	0	1	1	
Addend	0	0	1	1	
Sum	1	1	1	0	
Output carry	0	0	1	1	

Half Subtractor

The block diagram shown in fig 1.35 is a half subtractor and it has two inputs and two outputs. The two inputs and y form the minuend and the subtrahend D is the difference output and B is the borrow output. the function table explains the working of the half subtract or (Table 1.3). The simplified sum of products expressions are

$$D = x'y + xy'$$

$$B = x'y$$

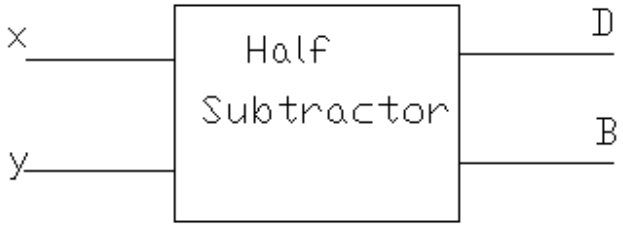


Figure 1.35 Block diagram of half subtractor

Table 1.3 Half Subtractor

x	y	D	B
0	0	0	1
0	1	1	1
1	0	1	0
1	1	0	0

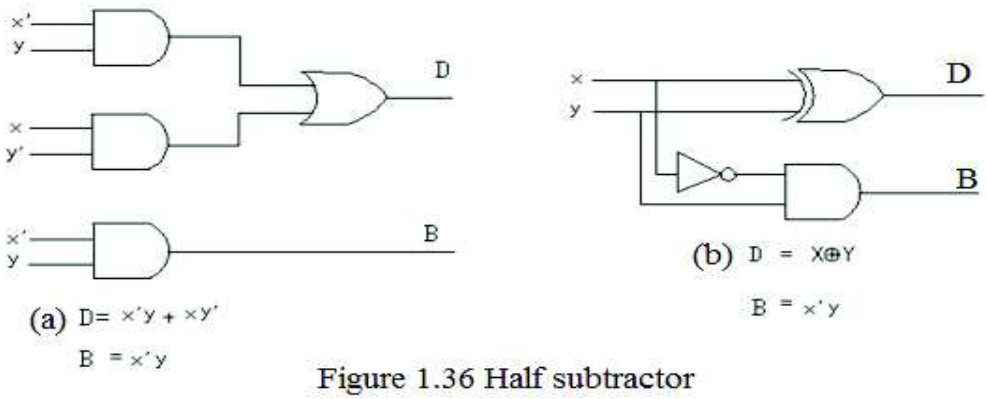


Figure 1.36 Half subtractor

The logic diagram implementation of these two expressions using basic gates is shown in fig 1.36(a) It can also be implemented using and EX-OR gate and an AND gate as indicated in Fig.1.36(b).

Full Subtractor

A full subtractor has three inputs and two outputs x,y and z are the inputs to be subtracted in which z represents borrow from the next stage. D and B are the outputs. The block diagrams of a full subtract or is shown in Fig. 1.37. Table 1.4 represents the truth table for a full subtractor and Fig. 1.38(a,b) shows the maps for outputs.

$$D = x'y'z + x'yz' + xy'z' + xyz$$

$$B = x'z + x'y + yz$$

The simplified expressions for D and B are implemented using basic gates are shown in fig 1.39.

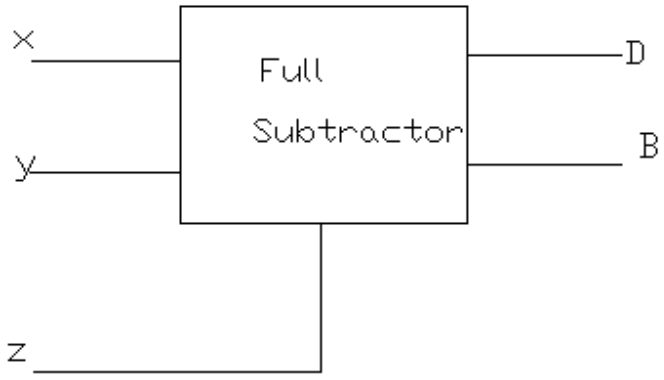


Figure 1.37 Block diagram of full subtractor

Table 1.4 Full Substractor

x	Y	Z	D	B
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

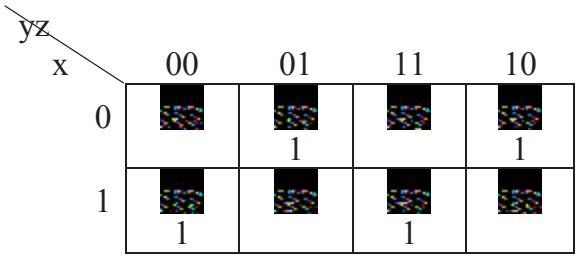


Figure 1.38a Maps for full subtractor

(a) K map for D=x'y'z+x'yz'+xy'z'+xyz

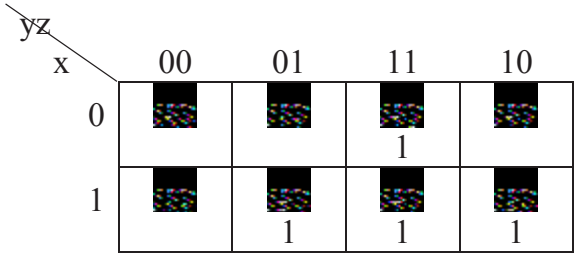


Figure 1.38b Maps for full substractor

(b) K map for B=x'z+x'y+yz

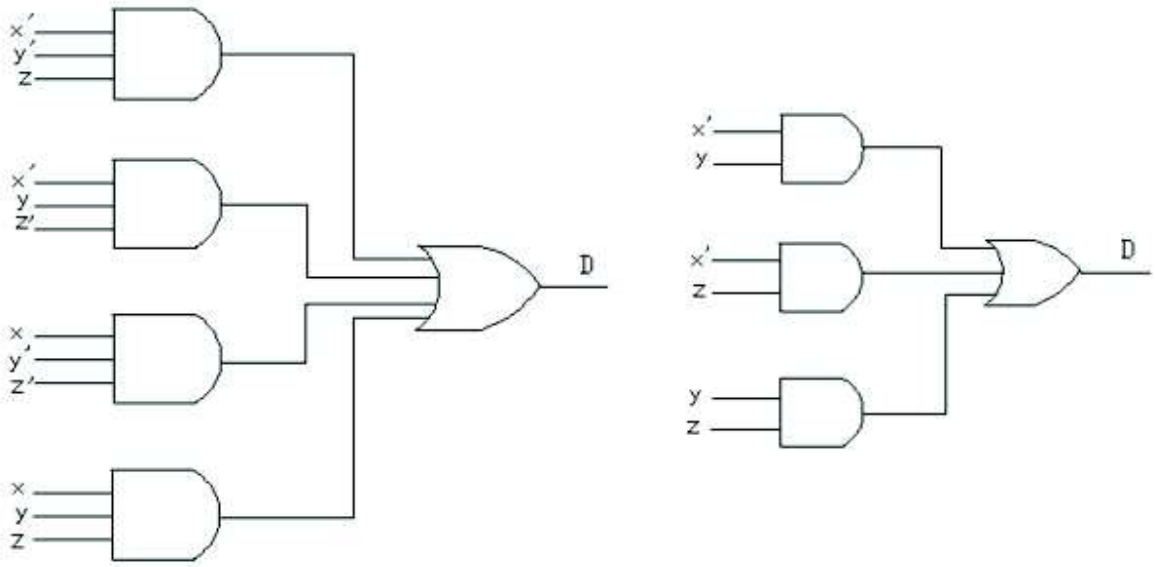


Figure 1.39 Implementation of full substractor

Binary subtractor

The subtractor of unsigned binary numbers is done by means of complements, Remember that the subtraction A-B is done by taking the 2's complement of B and adding it to A.

The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C0 must be equal to 1 when subtraction operations can be combined into one circuit with one common binary adder by including an exclusive – OR gate with each full adder Subtraction can be realized using an adder by controlling inputs to a parallel adder.

Fig.1.40(a) shows adder – subractor units using parallel adder

Considering the table, expressions for x and y can be obtained using K-map. The resulting expressions are

$x_1 = A_1$

$y_1 = B_1 \oplus M$ and

$C_1 = M$

These equations are implemented to obtain an adder. Subtractor logic diagram circuit and is shown in Fig. 1.40b.

M	[illegible]	[illegible]	[illegible]
0	[illegible]	[illegible]	0
1	[illegible]	[illegible]	1

M	[illegible]	[illegible]	[illegible]	[illegible]
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

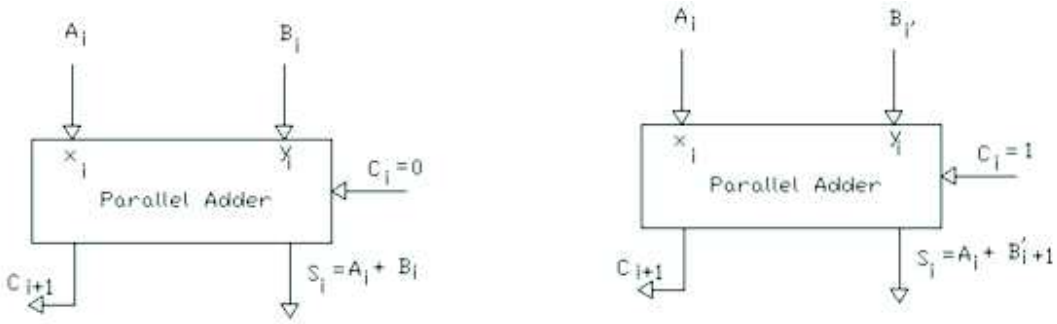


Figure 1.40 (a) Adder-Substractor units using Parallel adder

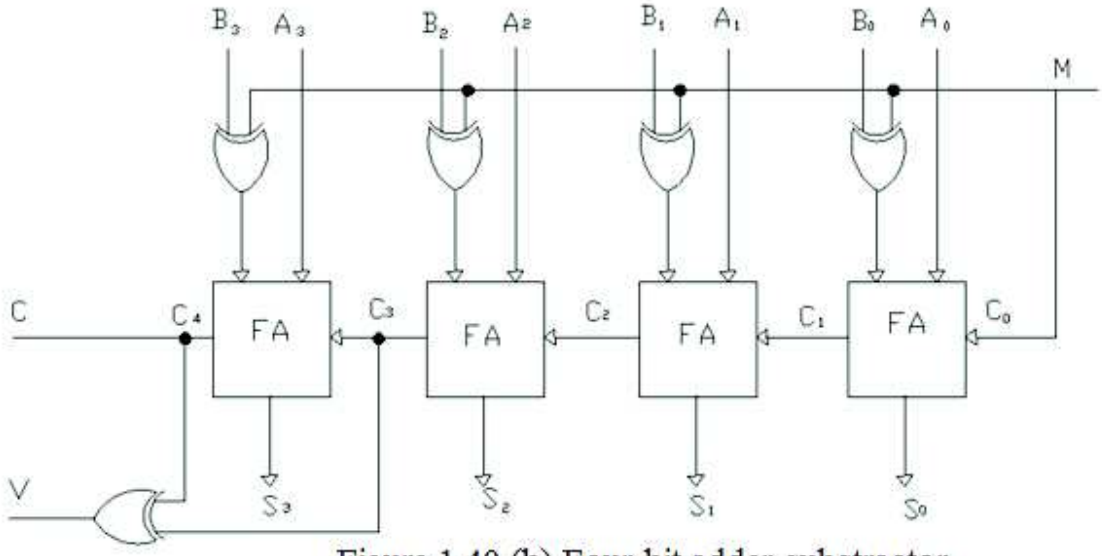


Figure 1.40 (b) Four-bit adder-substractor

MAGNITUDE COMPARATOR

The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equalto the other number.

A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes The outcome of the comparison is specified by three binary variables that indicate whether A>B,A=B or a<B

Digital function designed by means of an algorithm-a procedure which specifies a finite set of steps that, if followed, give the solution to a problem we illustrate this method here by driving an algorithm for the design of a four –bit magnitude comparator.

The algorithm is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers. consider two numbers, A and B, with four digits each. write the coefficients of the numbers in descending order of significance:

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

Each subscripted letter represents one of the digital the number are equal if all pairs of significant digits are equal: A3=B3, A2=B2, A1=B1, and A0=B0. When the numbers are

binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive – NOR functions as

$$X_i = A_i B_i + A_i' B_i' \quad \text{for I} = 0,1,3$$

Where xi=1 only if the pair of bits in position I are equal (i.e., if both are 0)

The binary variable (A=B) is equal if all pairs of significant digits of the two numbers are equal.

To determine whether A is greatest or less than B, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position if the two digit of a pair are equal, we compare the next lower significant pair of digits the comparison continues until a pair of unequal digits is reached. if the corresponding digit of A is 1 and that of B is 0, we conclude that corresponding digit of A is 0 and that B is 1, we have A<B. the sequential comparison can be expressed logically by the two Boolean function.

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_2 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$

The symbols (A>B) and (A<B) are binary output variables that are equal to 1 when A>B and A<B, respectively.

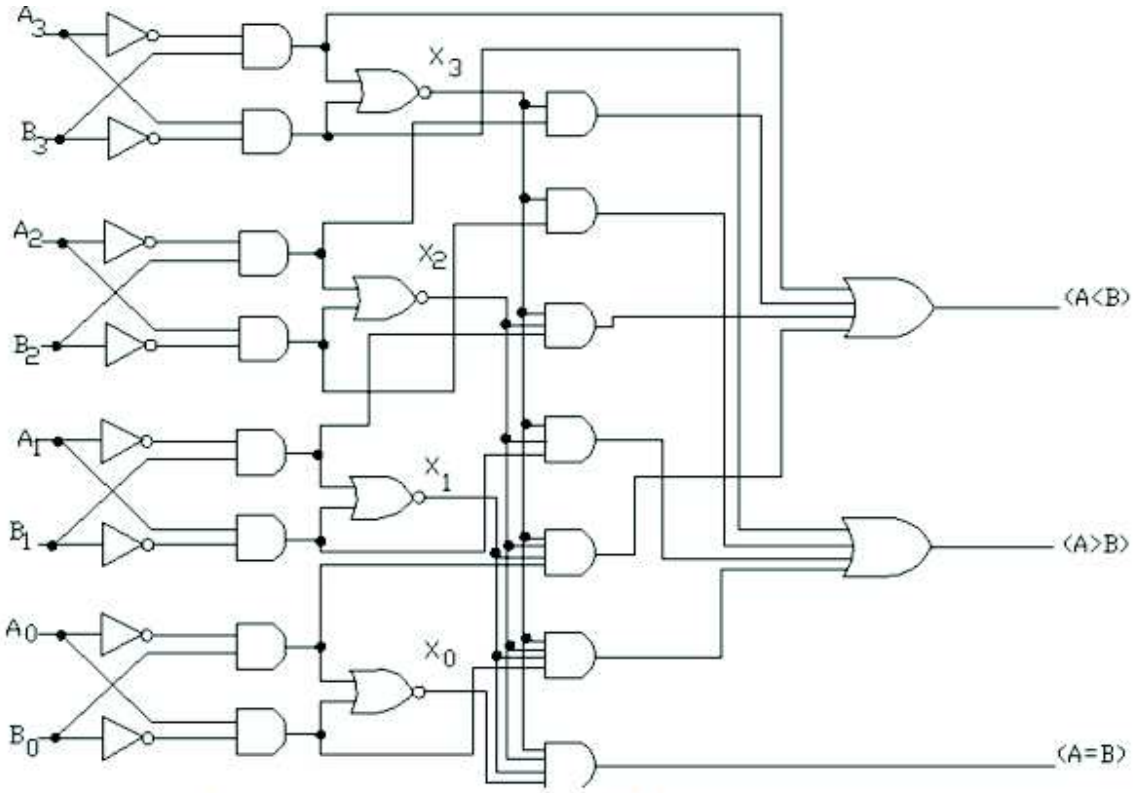


Figure 1.41 Four-bit magnitude comparator

The logic diagram of the four-bit magnitude comparator is shown in fig. 1.41. The four x outputs are generated with exclusive NOR circuits and are applied to an AND gate to give the output binary variables (A=B).

HAZARDS

In asynchronous sequential circuit it is important that undesirable glitches on signals should not occur. the glitches on signals should not occur. The glitches caused by the structure of a given circuit and propagation delays in the circuit are referred to as hazards. Two types of hazards are illustrated in Figure 1.42

(a) Static hazard

(b) Dynamic of Hazard

Figure 1.42 Definition of hazards

A static hazard exists if a signal is supposed to remain at a particular logic value, As shown in figure 1.42 a, one type of static hazard is when the signal at level 1 is supposed to remain at 1 but dips to 0 for a short time Another type is when the signal is supposed to remain at level 0 but rises momentarily to1, thus producing a glitch

A different type of hazard may occur when a signal is supposed to change involves a short oscillation before the signal settles into its new level, as illustrated in figure 1.42b, then a dynamic hazard is said to exits.

Critical and non-critical race conditions:

A critical race occurs when the order in which internal variables are changed determines the eventual state that the state machine will end up in.

A non-critical race occurs when the order in which internal variables are changed does not alter the eventual state.

Static, dynamic, and essential race conditions:

Static race conditions

These are caused when a signal and its complement are combined together.

Dynamic race Conditions:

These result in multiple transitions when only one is intended. they are due to interaction between gates

CIRCUIT SYNTHEIS USING GATES

Example : 1

Implement the function F = ∑ m {0,2,3,7} with minimal gates

SOULTION

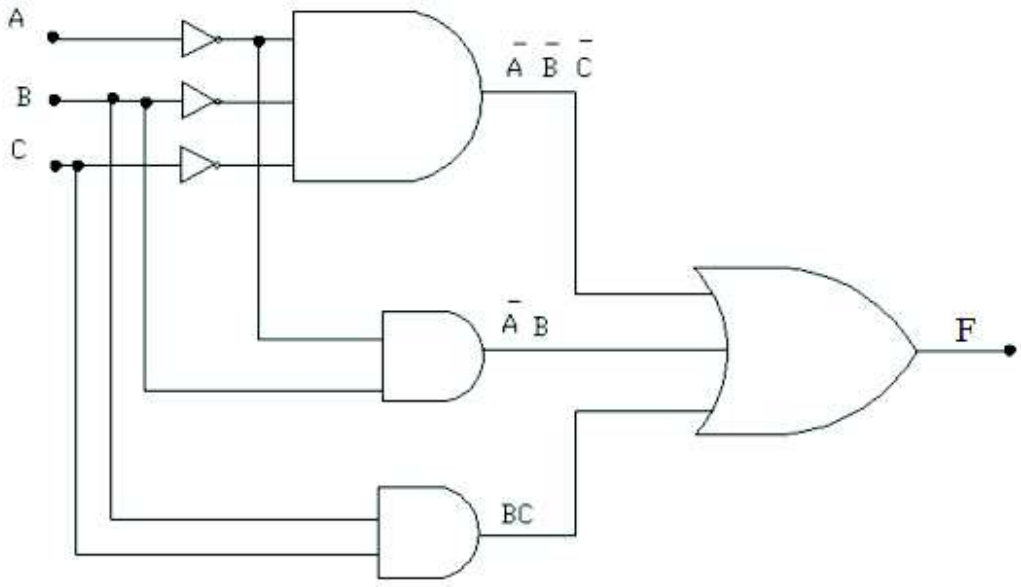
Step I

A \ BC	00	01	11	10
0	(1)	0	1	1
1	0	0	1	0

K map Simplification

$$\therefore F = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B + B\,C$$

Step II



Implementation with minimal gates

Example : 2

Implement the function F = ∑ {0,2,3,7} with do not care 4 & 6 with minimal gates.

SOLUTION

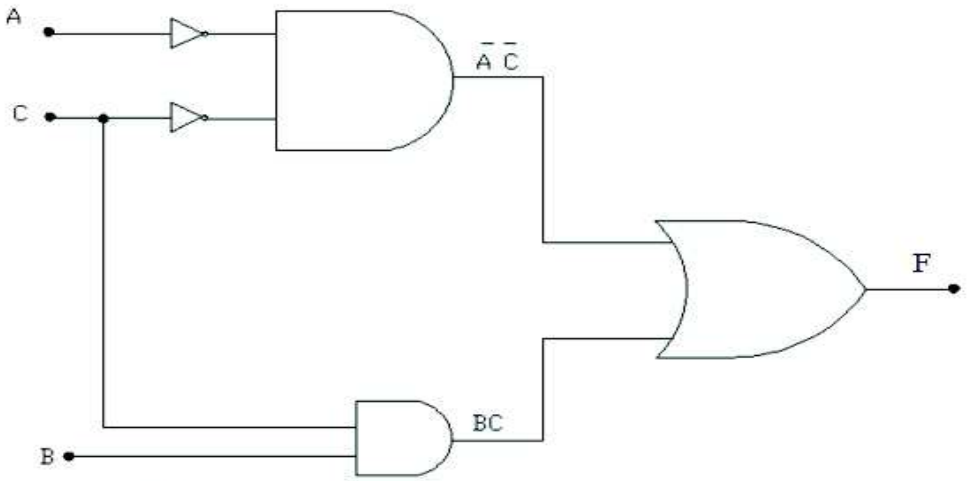
Step I

A \ BC	00	01	11	10
0	1	0	1	1
1	x	0	1	x

K map simplification

$$\therefore F = \overline{A}\,\overline{C} + B\,C$$

Step II



Implementation with mimimal gates

Example : 3 Implement the function

$F(A,B,C,D) = \sum m\{4,5,6,7,8,12\} + d\{1,2,3,9,11,14\}$ with only NAND gates

SOLUTION

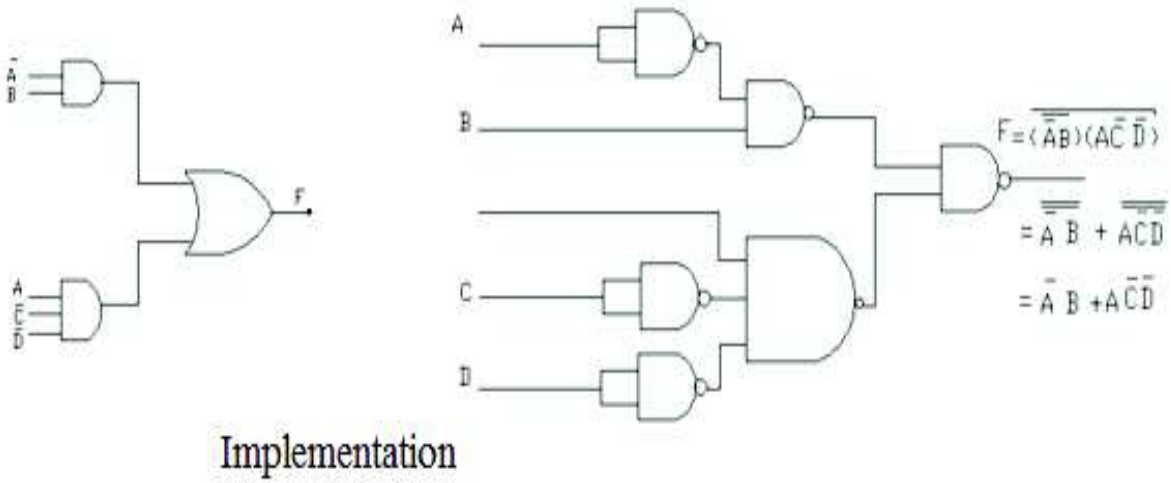
Step I

K map Simplification

AB \ CD	00	01	11	10
00	0	x	x	x
01	1	1	1	1
11	1	0	0	x
10	1	x	x	1

$$\therefore F(A,B,C,D) = \bar{A}B + A\bar{C}D$$

Step II



Implementation

SYNTHESIS OF LOGIC FUNCTION USING MULTIPLEXER

Example : 1

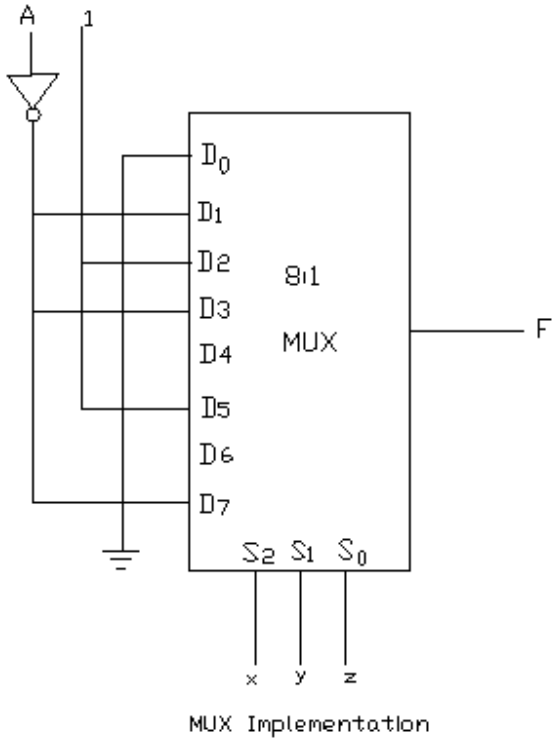
Implement the function f = ∑m{ 1,2,3,5,7,10,13} multiplexer

SOLUTION

Step I

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

Step II



MUX Implementation

Example 2 : Implement the function F = ∑m{1,2,3,5,7,10,13} with don't care of 4 & 6 with multiplexer

SOLUTION Consider 4 & 6

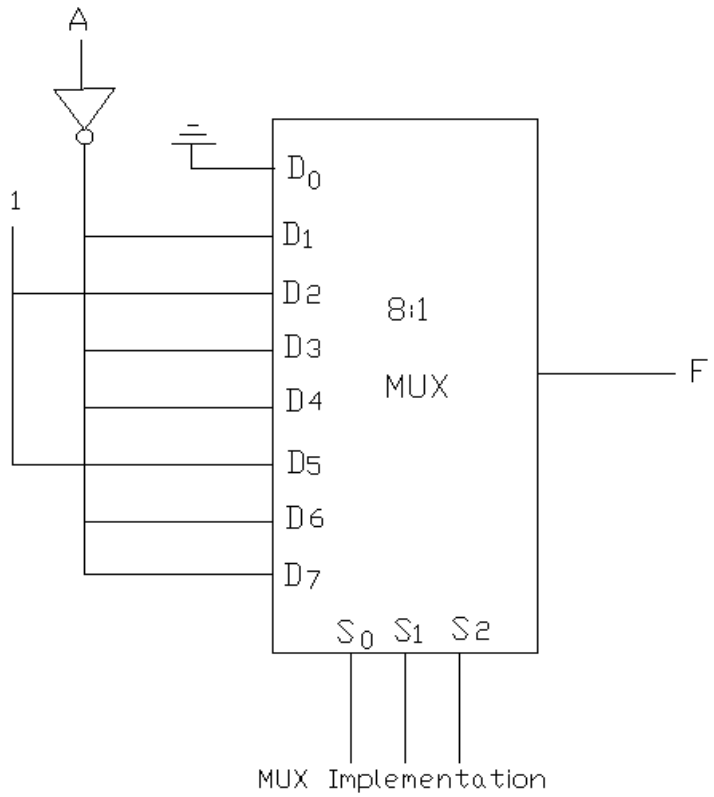
Step I

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

Here don't care as =1

Implementation Table

Step II



Example 3 : Implement the function F = ∑m {0,2,3,7} with mux

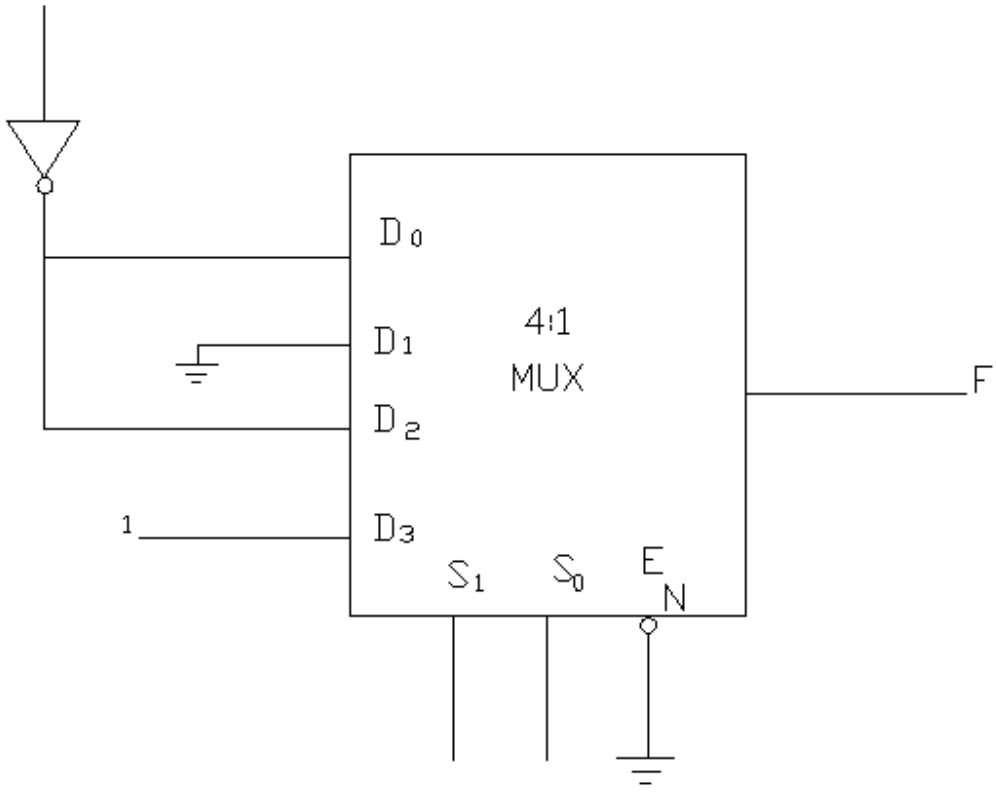
SOLUTION

Step I

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

Implementation Table

Step II



Example 4 : Implement the function F = ∑ m {0,2,3,7} with don't care 4 & 6 with mux

SOLUTION

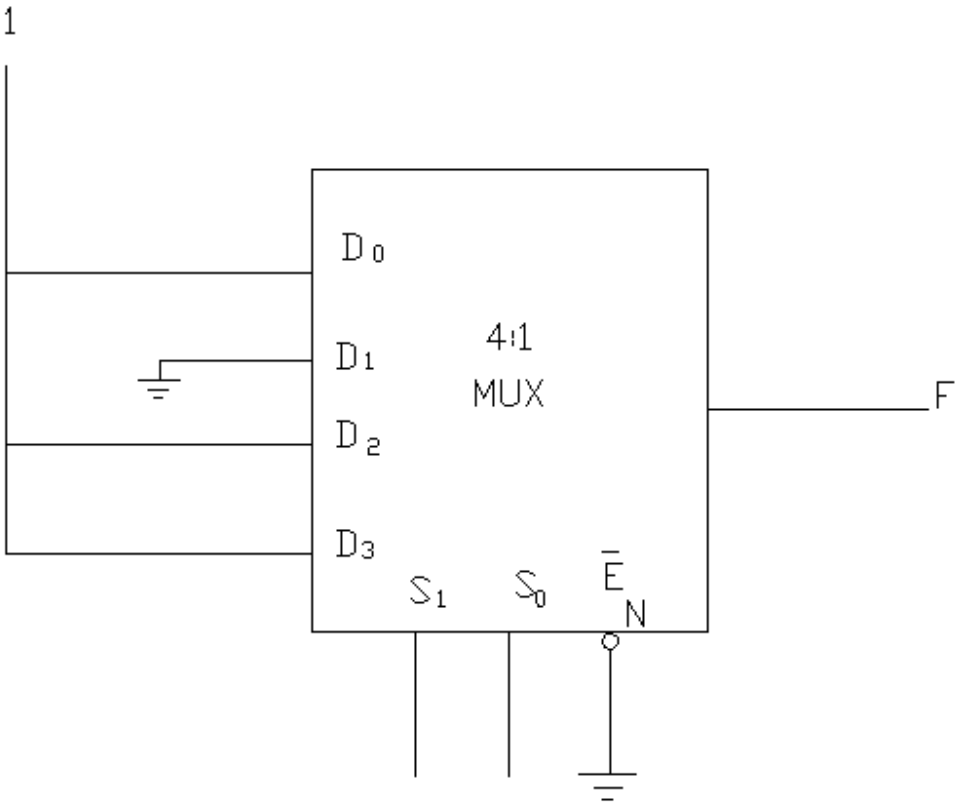
Step I

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

Here don't care as =1

Implementation Table

StepII



Example 5 :

Implement the following function using 4:1 mux F (A,B,C,D) = ∑ {0,1,2,4,6,9,12,14}

SOLUTION :

The function has four variables to implement this function, we require two 4:1 mux.

Step I

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

Implementation Table

Step II