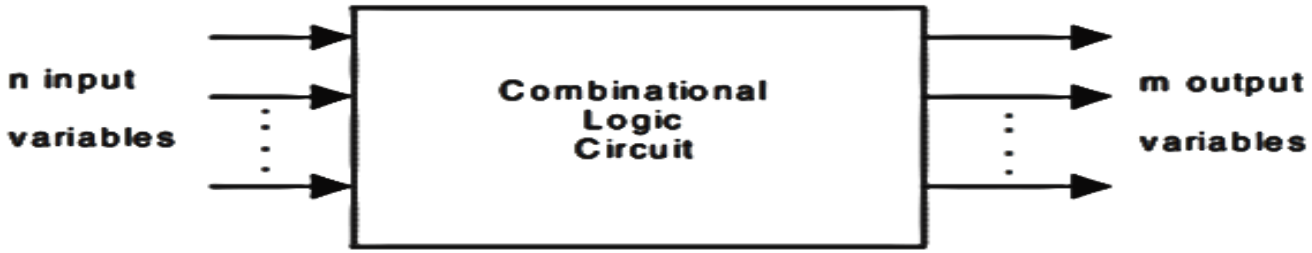
CHAPTER-5

ANALYSIS AND SYNTHESIS OF COMBINATIONAL LOGIC CIRCUIT

Introduction

- The digital system consists of two types of circuits, namely
  I. Combinational logic circuits and
  II. Sequential circuits
- A combinational circuit consists of logic gates, where outputs are at any instant and are determined only by the present combination of inputs.
- Sequential circuits contain logic gates as well as memory cells. Their outputs depend on the present inputs and also on the states of memory elements.
- A combinational logic circuit consists of input variables, logic gates, and output variables.
- The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it
- Figure below shows a block diagram of a combinational logic circuit.
- There are n number of input variables coming from an electric source and m number of output signals go to an external destination.



DESIGN PROCEDURE

- Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.
2. Identify the input variables and output functions.
3. The input and output variables are assigned letter symbols.
4. The truth table is prepared that completely defines the relationship between the input variables and output functions.
5. The simplified Boolean expression is obtained by any method of minimization—algebraic method, or Karnaugh map method.
6. A logic diagram is realized from the simplified expression using logic gates.

- The output Boolean functions from the truth table are simplified by any available method, such as algebraic manipulation, the map method, or the tabulation procedure.
- Usually, there will be a variety of simplified expressions from which to choose.

Cont.…

- However, in any particular application, certain restrictions, limitations, and criteria will serve as a guide in the process of choosing a particular algebraic expression.
- A practical design method would have to consider such constraints as
  a. Minimum number of gates,
  b. Minimum number of inputs to a gate,
  c. Minimum propagation time of the signal through the circuit,
  d. Minimum number of interconnections, and
  e. Limitations of the driving capabilities of each gate.
- Since all these criteria cannot be satisfied simultaneously, and since the importance of each constraint is dictated by the particular application, it is difficult to make a general statement as to what constitutes an acceptable simplification.
- In most cases, the simplification begins by satisfying an elementary objective, such as producing a simplified Boolean function in ***a standard form***, and from that proceeds to meet any other performance criteria

COMBINATIONAL LOGIC USING NAND AND NOR GATES

a) NAND Implementation

- The NAND gate is said to be a universal gate because any digital system can be implemented with it.
- To facilitate the conversion to NAND logic, it is convenient to use the two alternate graphic symbols shown in Figure below.

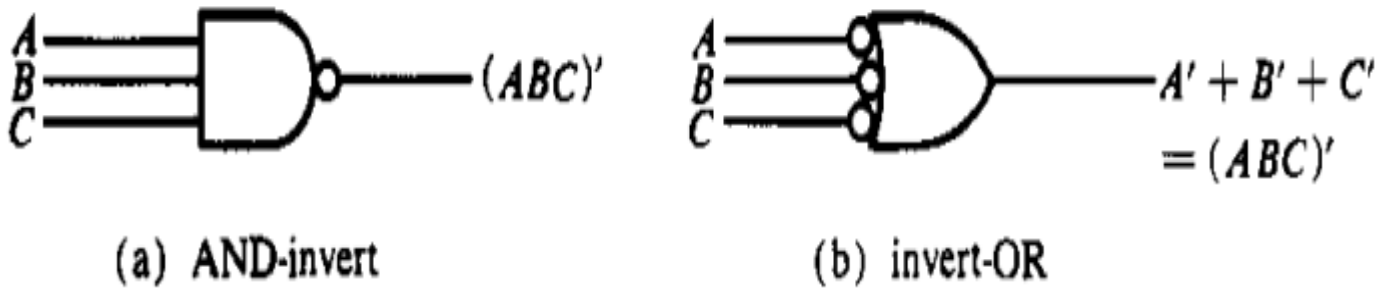


FIGURE. Two graphic symbols for a NAND gate

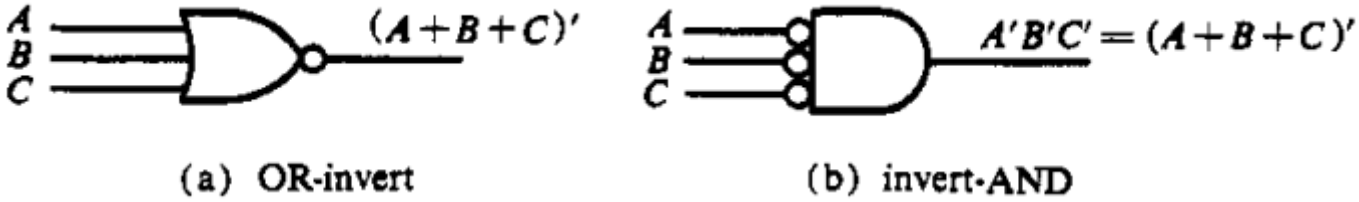
- The AND-invert graphic symbol consists of an AND graphic symbol followed by a small circle.
- The invert-OR graphic symbol consists of an OR graphic symbol that is preceded by small circles in all the inputs.
- Either symbol can be used to represent a NAND gate.

Cont..

- To obtain a multilevel NAND diagram from a Boolean expression, proceed as follows:
  a. From the given Boolean expression, draw the logic diagram with AND, OR, and inverter gates. Assume that both the normal and complement inputs are available.
  b. Convert all AND gates to NAND gates with AND-invert graphic symbols.
  c. Convert all OR gates to NAND gates with invert-OR graphic symbols.
  d. Check all small circles in the diagram. For every small circle that is not compensated by another small circle along the same line, insert an inverter (one-input NAND gate) or complement the input variable.
- In general, the number of NAND gates required to implement a Boolean expression is equal to the number of AND-OR gates except for an occasional inverter.
- Example: implement using only NAND gates.
  a) F = A + (B’ + C)(D’ + BE’)
  b) F = (CD + E)(A + B’)
  c) F = BC’ + A(B + CD)
  d) F = (AB' + CD')E + BC(A + B)
  e) F = w(x + y + z) + xy;

b) NOR implementation

- The two graphic symbols for the NOR gate are shown in Fig. below.

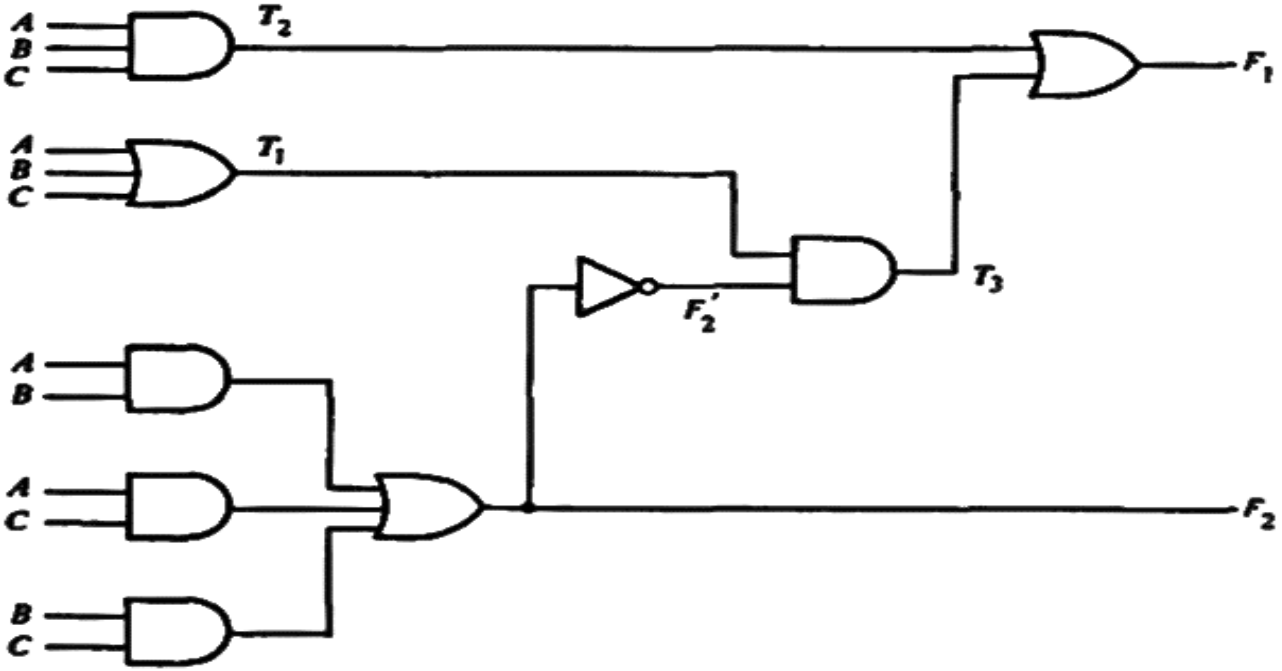


(a) OR-invert (b) invert-AND

- The OR-invert symbol defines the NOR operation as an OR followed by a complement.
- The invert-AND symbol complements each input and then performs an AND operation.
- The two symbols designate the same NOR operation and are logically identical because of DeMorgan's theorem.
- The procedure for implementing a Boolean function with NOR gates is similar to the procedure outlined in the previous section for NAND gates.
  a. Draw the AND-OR logic diagram from the given algebraic expression Assume that both the normal and complement inputs are available.
  b. Convert all OR gates to NOR gates with OR-invert graphic symbols.
  c. Convert all AND gates to NOR gates with invert-AND graphic symbols.
  d. Any small circle that is not compensated by another small circle along the same line needs an inverter or the complementation of the input variable.

Cont…

- **Exercise** : implement the following Boolean function using only NOR gates.
  a. F = (AB + E)(C + D)
  b. F = A + (B’ + C)(D’ + BE’)
  c. F = (CD + E)(A + B’)
  d. F = [(C + D)B' + A](B + C')
  e. F = (AB' + CD')E + BC(A + B)
  f. F = w(x + y + z) + xy;
- Convert the logic diagram the following to a multiple-level NAND and NOR circuit



Example #1.

Design a logic circuit that has three inputs, A, B and C whose output will be high only when a majority of the inputs are high.

Solution: **Step 1 :Set up the truth table**

Truth Table

A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\leftarrow \bar{A}BC$
1	0	0	0	
1	0	1	1	$\leftarrow A\bar{B}C$
1	1	0	1	$\leftarrow AB\bar{C}$
1	1	1	1	$\leftarrow ABC$

AND Terms

STEP 2

Step 3. Write the Sum-of-products expression for the output.

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Step 4. Simplify the output expression.

- ✓ **This expression can be simplified in several ways.**
- ✓ **Perhaps the quickest way is to realize that the last term ABC two variables in common with each of the other terms.**

$$Y = \overline{A}BC + ABC + A\overline{B}C + ABC + AB\overline{C} + ABC$$

- ✓ **Factoring the appropriate pairs of terms , we have**

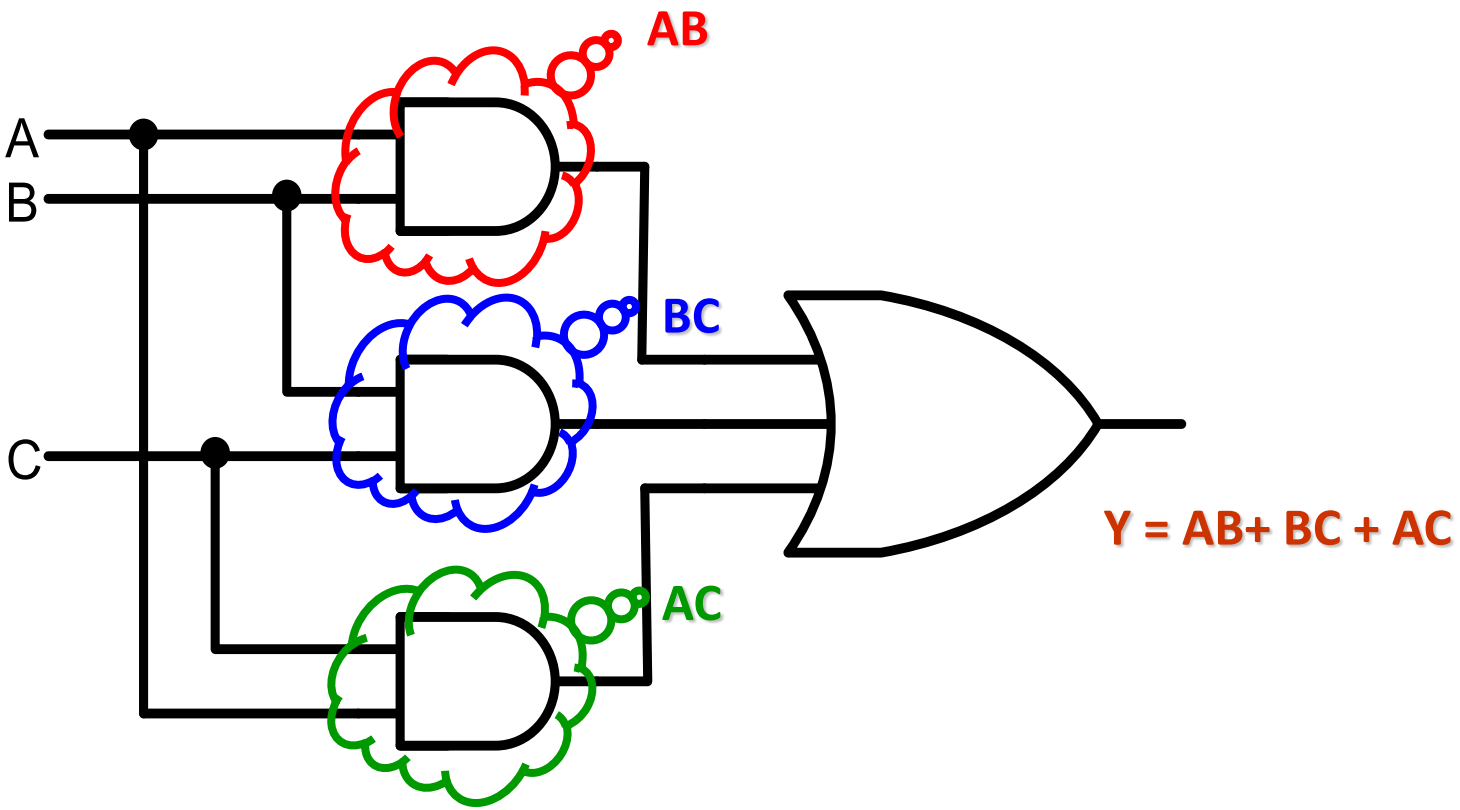
$$Y = BC(\overline{A} + A) + AC(\overline{B} + B) + AB(\overline{C} + C)$$

- ✓ **Since each term in the parenthesis is equal to 1, we have**

$$\boxed{Y = BC + AC + AB}$$

Step 5. Implement the circuit for the final expression.

- ✓ The Y= AB + BC + AC expression is implemented in fig. below.
- ✓ Since the expression is in SOP, the circuit consists of a group of AND gates working into a single OR gate.



Example # 2:

- ✓ refer to fig (a), where four logic-signal lines A, B, C, D are being used to represent a 4-bit binary number with A as the MSB and D as the LSB. The binary inputs are fed to a logic circuit that produces a HIGH output only when the binary number is greater than $0110_2 = 6_{10}$.
- ✓ Design this circuit

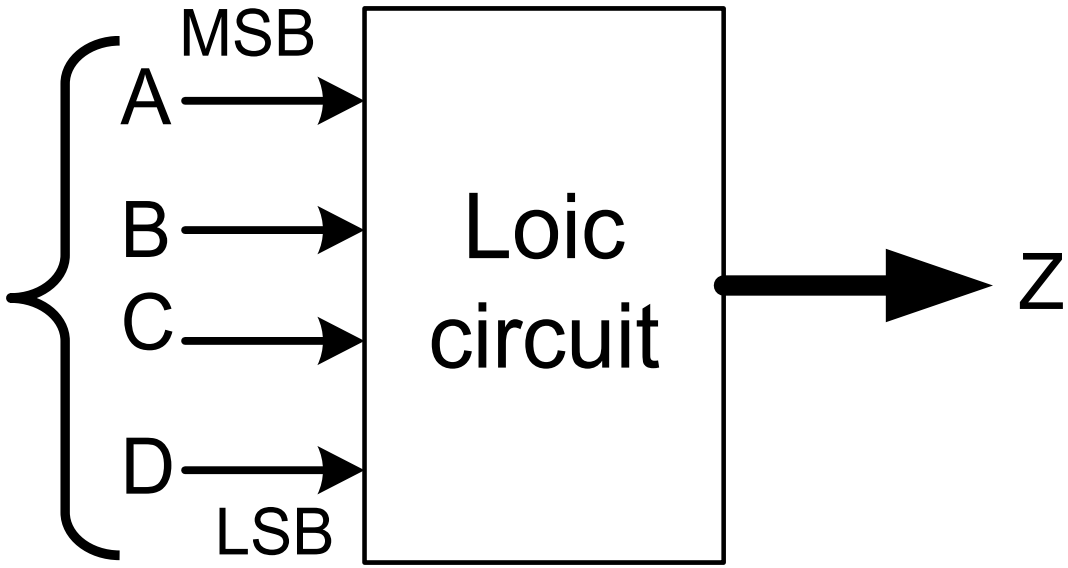
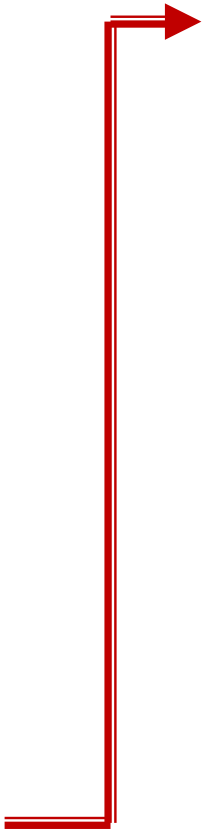


Fig.(a)

TRUTH TABLE

	A	B	C	D	Z
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	



	A	B	C	D	Z
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

STEP: 2

	A	B	C	D	Z
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1

$\overline{A}BCD$

	A	B	C	D	Z	
8	1	0	0	0	1	$A\overline{B}\overline{C}\overline{D}$
9	1	0	0	1	1	$A\overline{B}\overline{C}D$
10	1	0	1	0	1	$A\overline{B}C\overline{D}$
11	1	0	1	1	1	$A\overline{B}CD$
12	1	1	0	0	1	$AB\overline{C}\overline{D}$
13	1	1	0	1	1	$AB\overline{C}D$
14	1	1	1	0	1	$ABC\overline{D}$
15	1	1	1	1	1	$ABCD$

Step 3. SOP expression

$$Z = \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$$

Step:4 Simplification

$$Z = \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$$

$$Z = \bar{A}BCD + A\bar{B}\bar{C}(\bar{D} + D) + A\bar{B}C(\bar{D} + D) + AB\bar{C}(\bar{D} + D) + ABC(\bar{D} + D)$$

$$= \bar{A}BCD + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

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

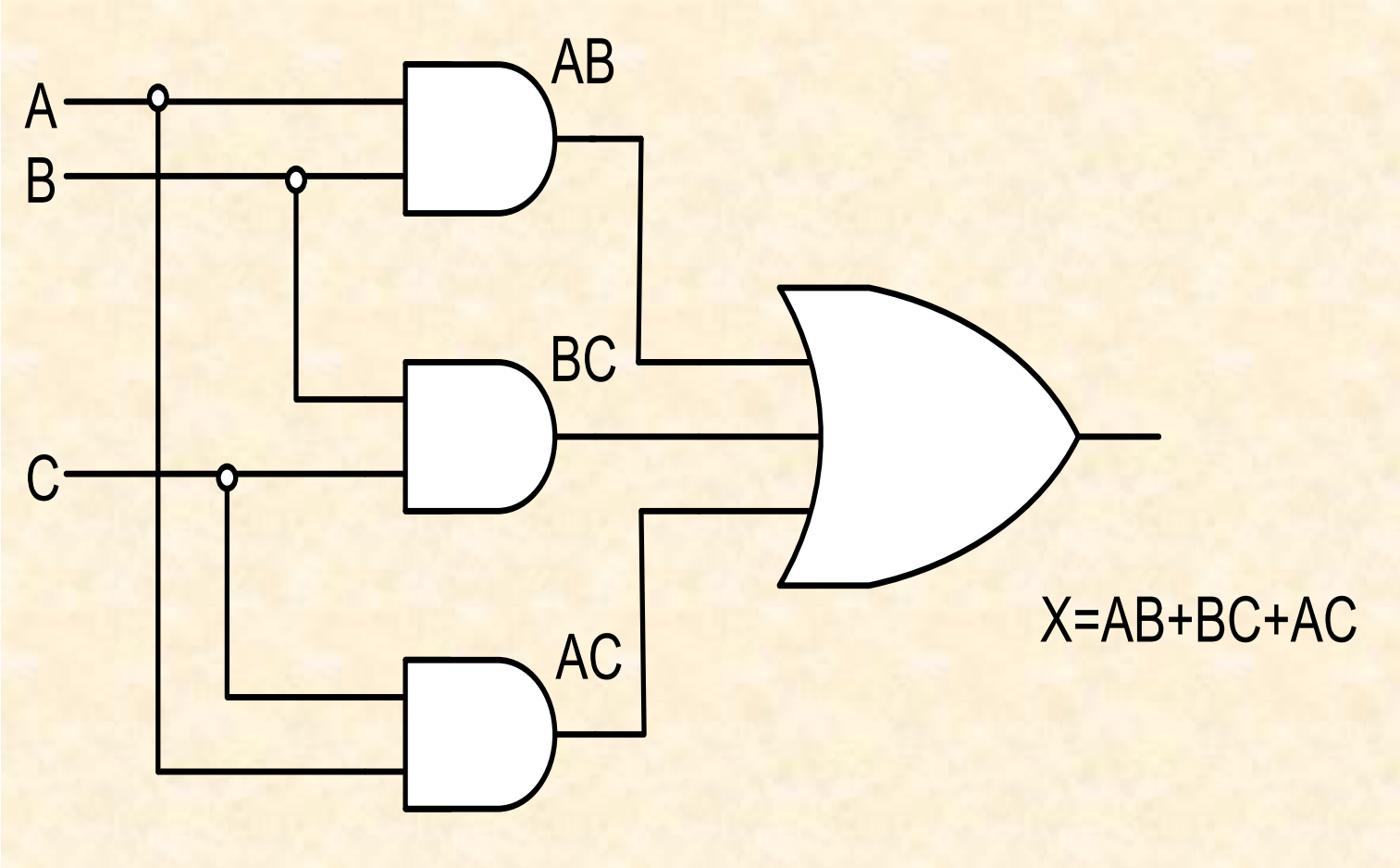
$$= \bar{A}BCD + A\bar{B} + AB$$

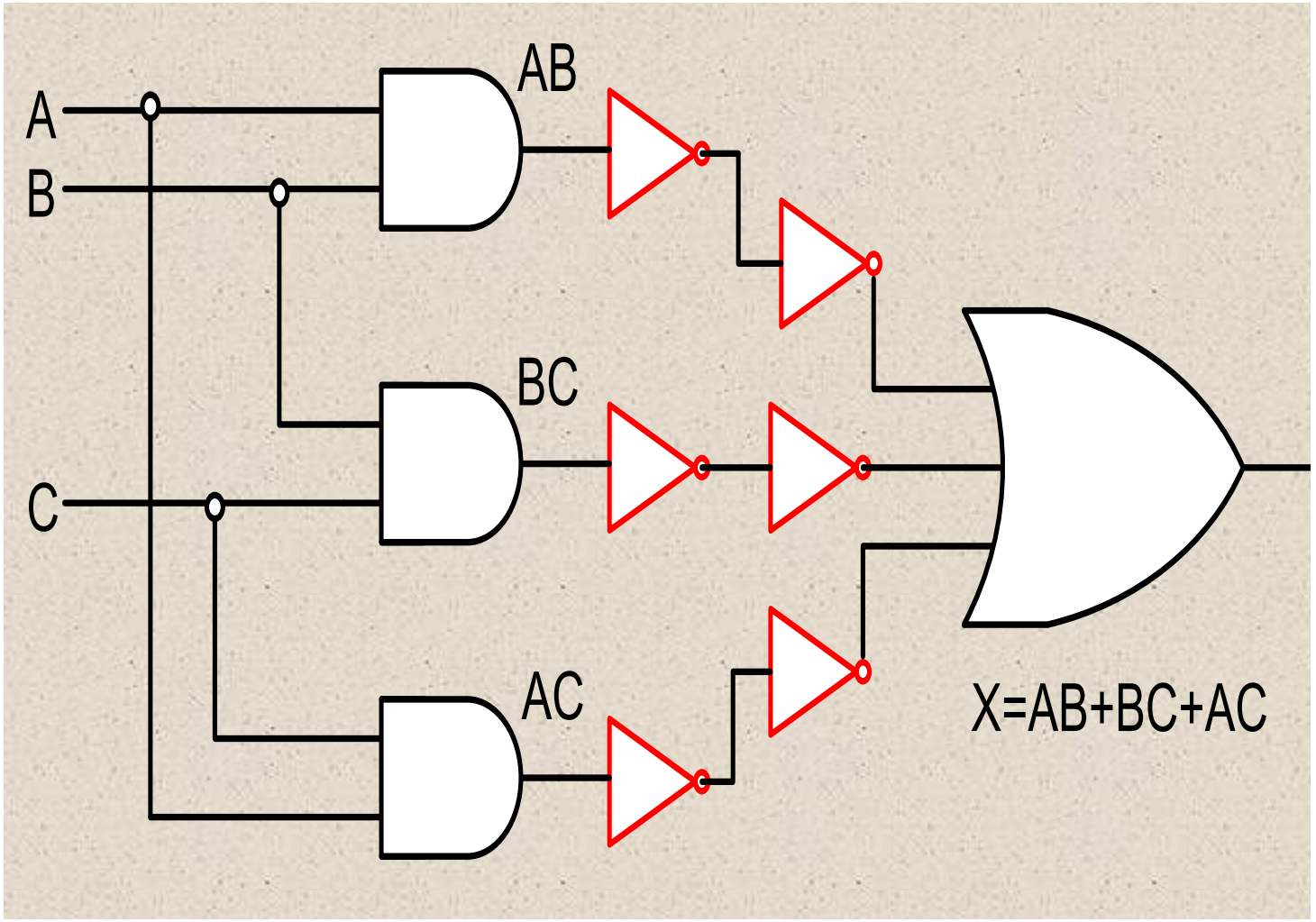
$$= \bar{A}BCD + A(\bar{B} + B)$$

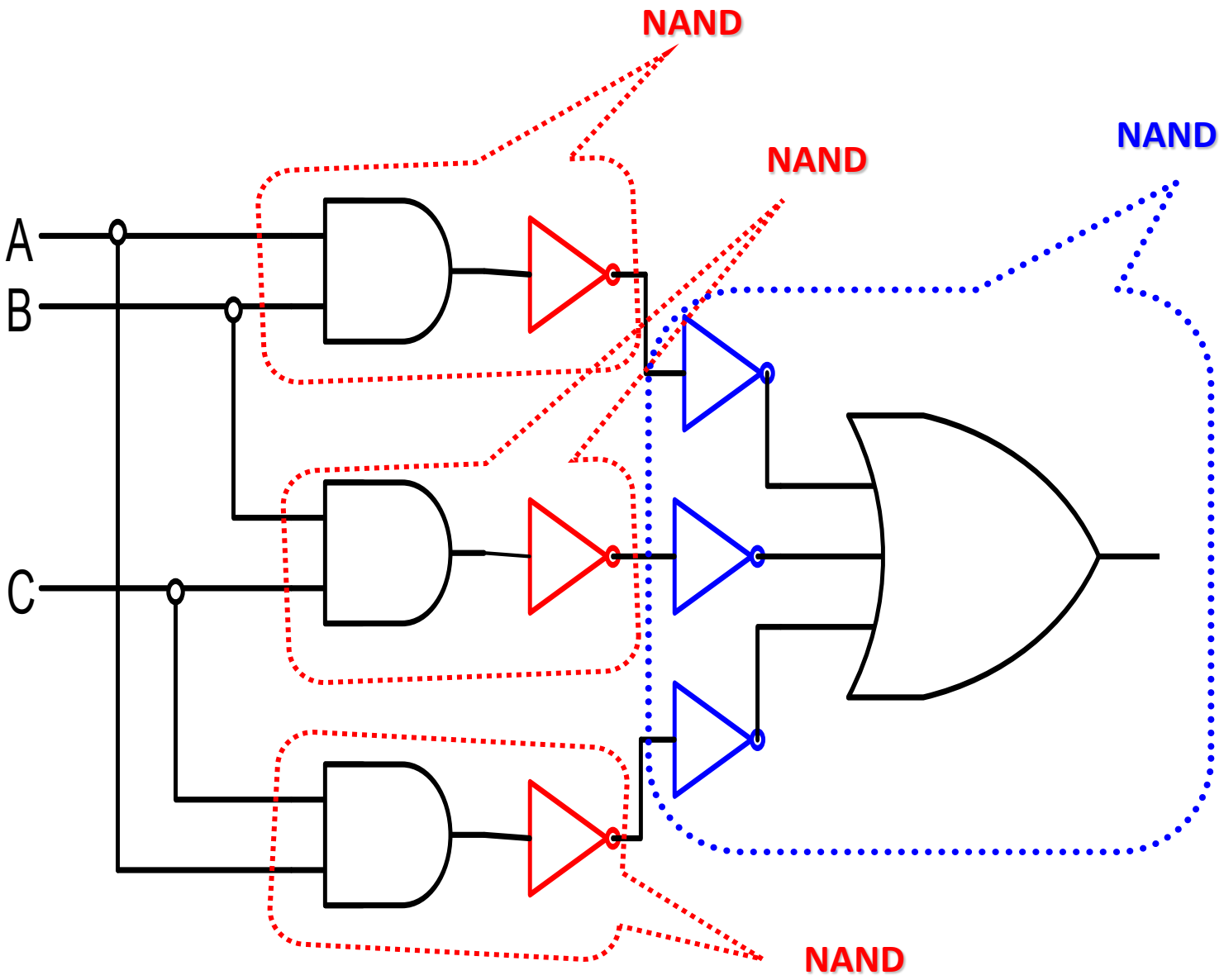
$$= \bar{A}BCD + A$$

$$Z = BCD + A$$

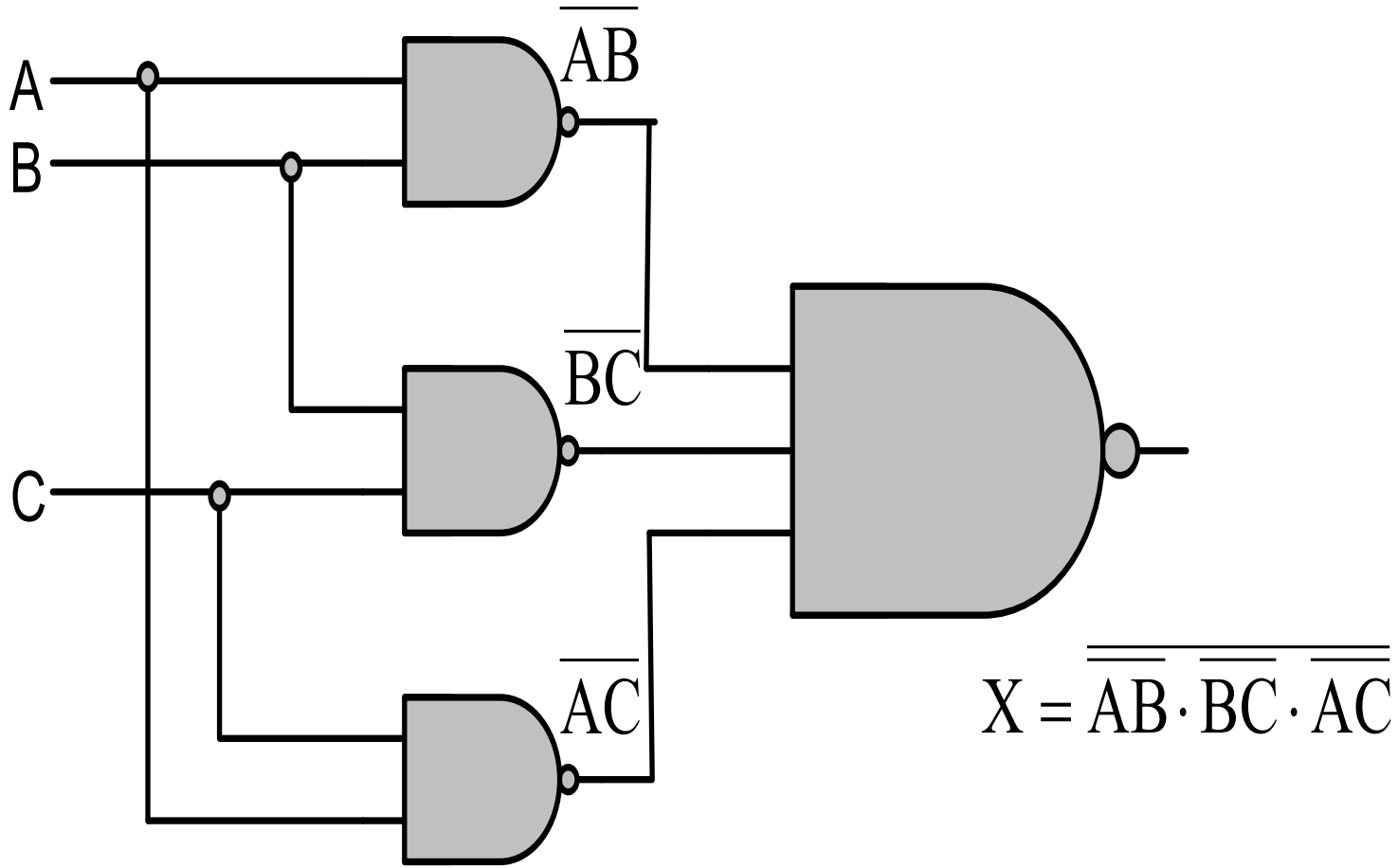
AOI LOGIC CIRCUIT
(for example # 1)



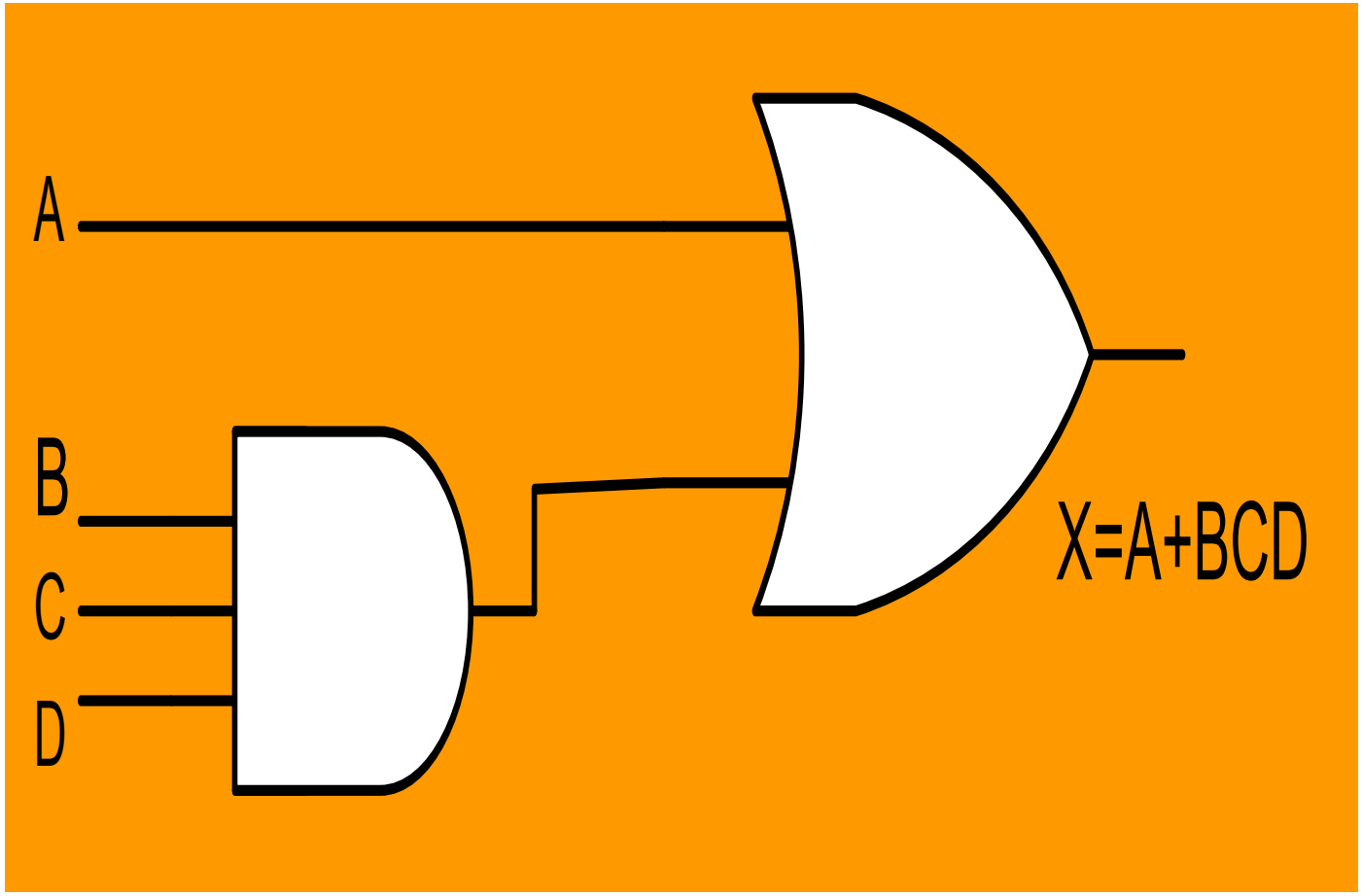

A
B
C
AB
BC
AC
X=AB+BC+AC


NAND
NAND
NAND
A
B
C
NAND

NAND/NAND LOGIC CIRCUIT



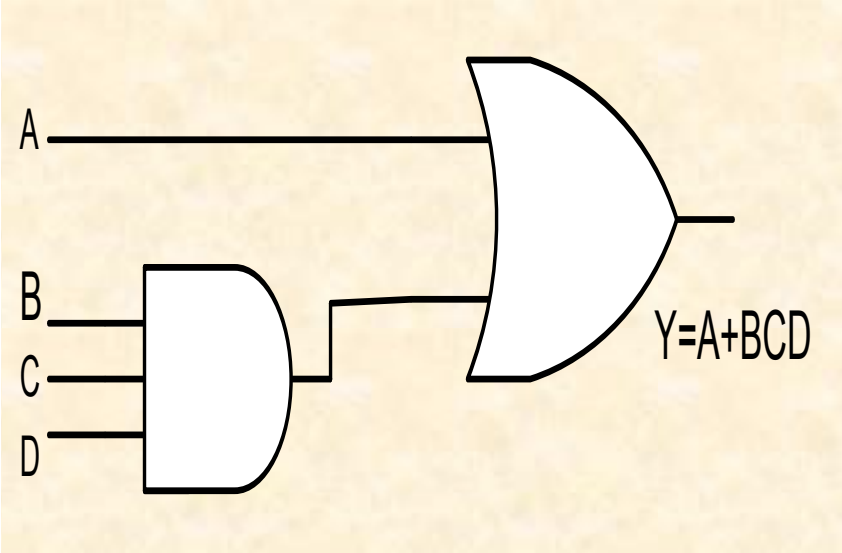
AND-OR-INVERTER LOGIC CIRCUIT
(for example # 2)



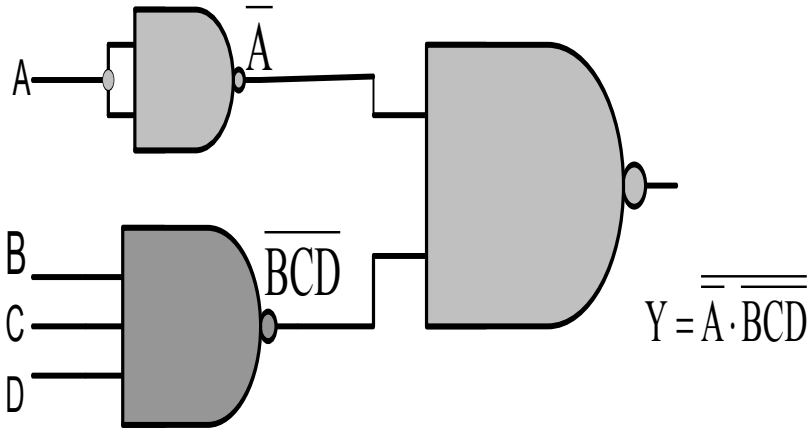
- ✓ **We can streamline the process of converting a sum-of-products circuit from AND/OR to NAND gates as follows:**

1. **Replace each AND gate, OR gate, and Inverter by a single NAND gate.**
2. **Use a NAND gate to invert any single variable that is feeding the final OR gate.**

AND-OR-INVERTER LOGIC CIRCUIT



NAND/NAND LOGIC CIRCUIT



Example #3: Design a combinational circuit with four inputs and four outputs. The output generates the 2's complement of the input binary number.

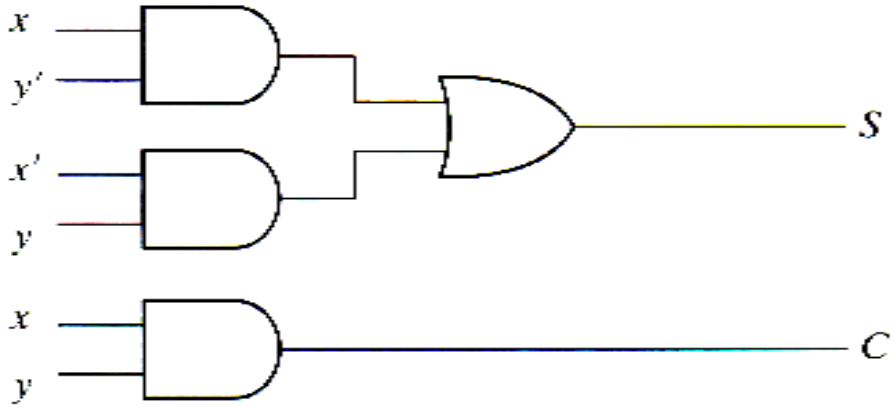
FUNCTIONS OF COMBINATIONAL LOGIC

1. HALF-ADDER

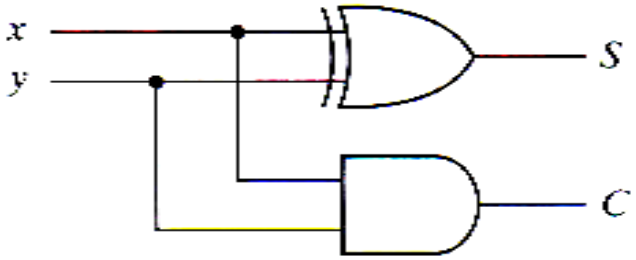
- The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit.
- The Sout represents the least significant bit of the sum.
- The Cout represents the most significant bit of the sum.

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- The Boolean functions for the two output from the truth table:
- **S = x'y+ xy' = x⊕y**
- **C = xy**



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

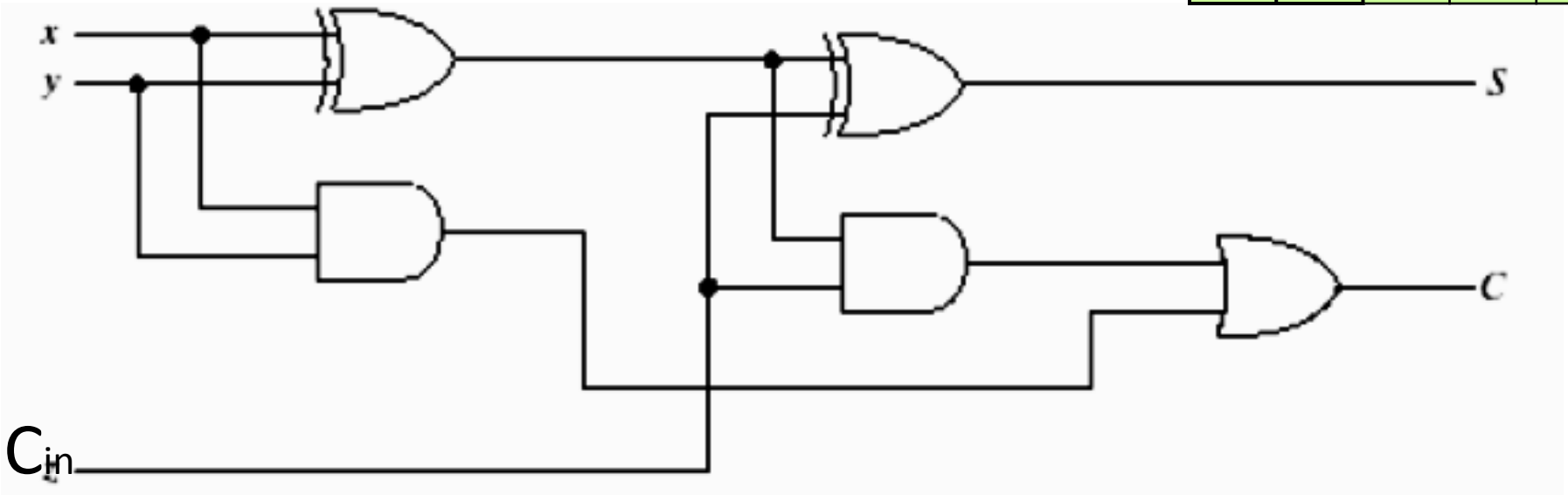
2. The Full-Adder

- The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.
- The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry.
- The Boolean functions for the two out put from the truth tabl
- Using x-OR, $S = X'Y'C_{in} + X'YC'_{in} + XY'C'_{in} + XYC_{in} = X \oplus Y \oplus C_{in}$

$$C = X'YC_{in} + XYC'_{in} + XYC'_{in} + XYC_{in} = XY + C_{in}(X \oplus Y)$$

X	Y	C_{in}	C_{out}	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

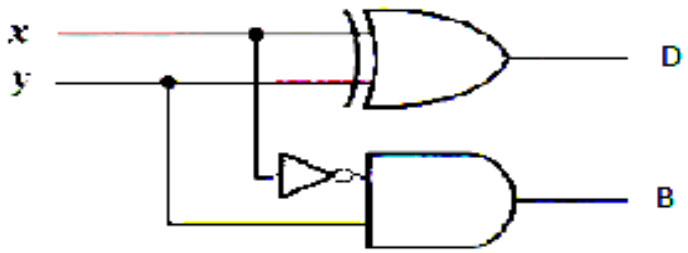
Implementation of Full Adder with Two Half Adders and an OR Gate



3. HALF-SUBTRACTOR

- A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output to specify if a 1 has been borrowed.
- The Boolean functions for the two outputs of the half-subtractor are derived directly from the truth table:
- D = x'y + xy' = x⊕y
- B = x'y

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

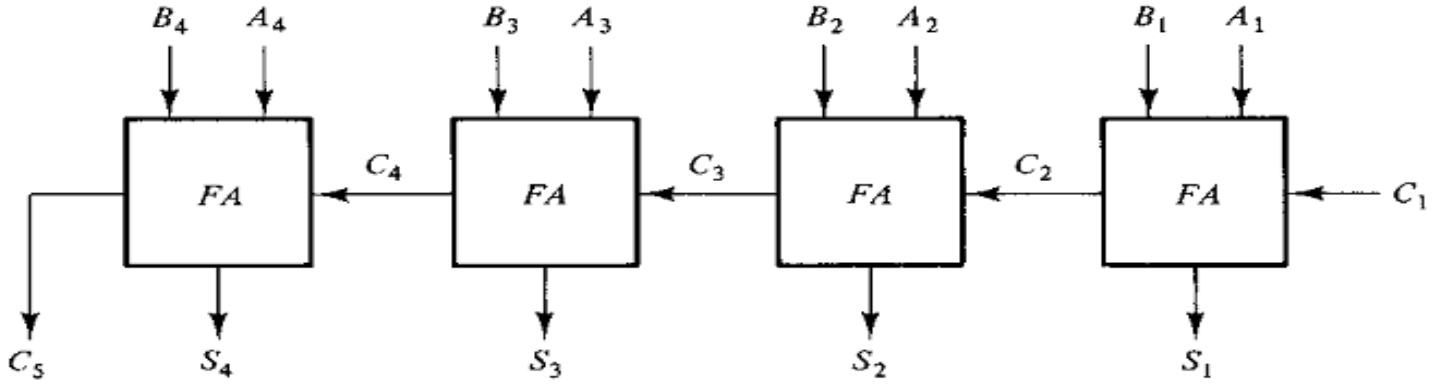


4. FULL-SUBTRACTOR

- A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage.
- This circuit has three inputs and two outputs.
- Exercise:- design the full-subtractor combinational logic ckt.

5. BINARY PARALLEL ADDER

- A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers in parallel.
- It consists of full-adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.
- Figure below shows the interconnection of four full-adder (FA) circuits to provide a
- 4-bit binary parallel adder.
- An n-bit parallel adder requires n full-adders.



(a) 4-bit parallel adder

- Example:

a) What are the sum outputs when 111 and 101 are added by the 3-bit parallel adder?

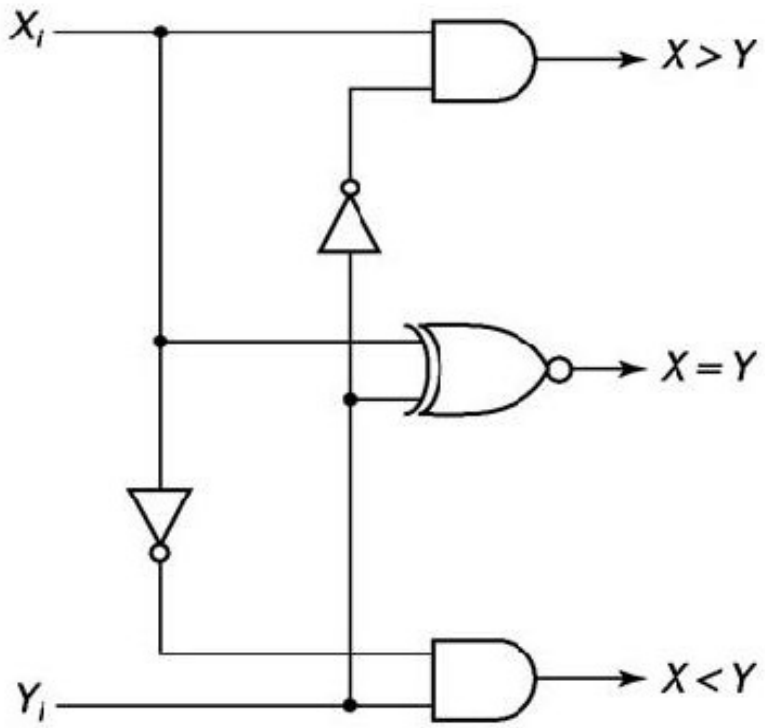
b) What are the sum outputs when 1110 and 1011 are added by the 4-bit parallel adder?

6. MAGNITUDE COMPARATOR

- The comparison of two numbers is an operation that determines if one number is greater than, less than, or equal to the other number.
- A magnitude comparator is a combinational circuit that compares two numbers, X and Y, and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether X > Y, X = Y, or X<Y .

- Compares two *n*-bit binary values to determine which one is larger.
- Inputs : 2 single-bit inputs (X and Y).
- Outputs : 3 lines: *X>Y*, *X=Y*, *X<Y*.

Cont….



(a)

X_i	Y_i	$X > Y$	$X = Y$	$X < Y$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

(b)

7. DECODERS

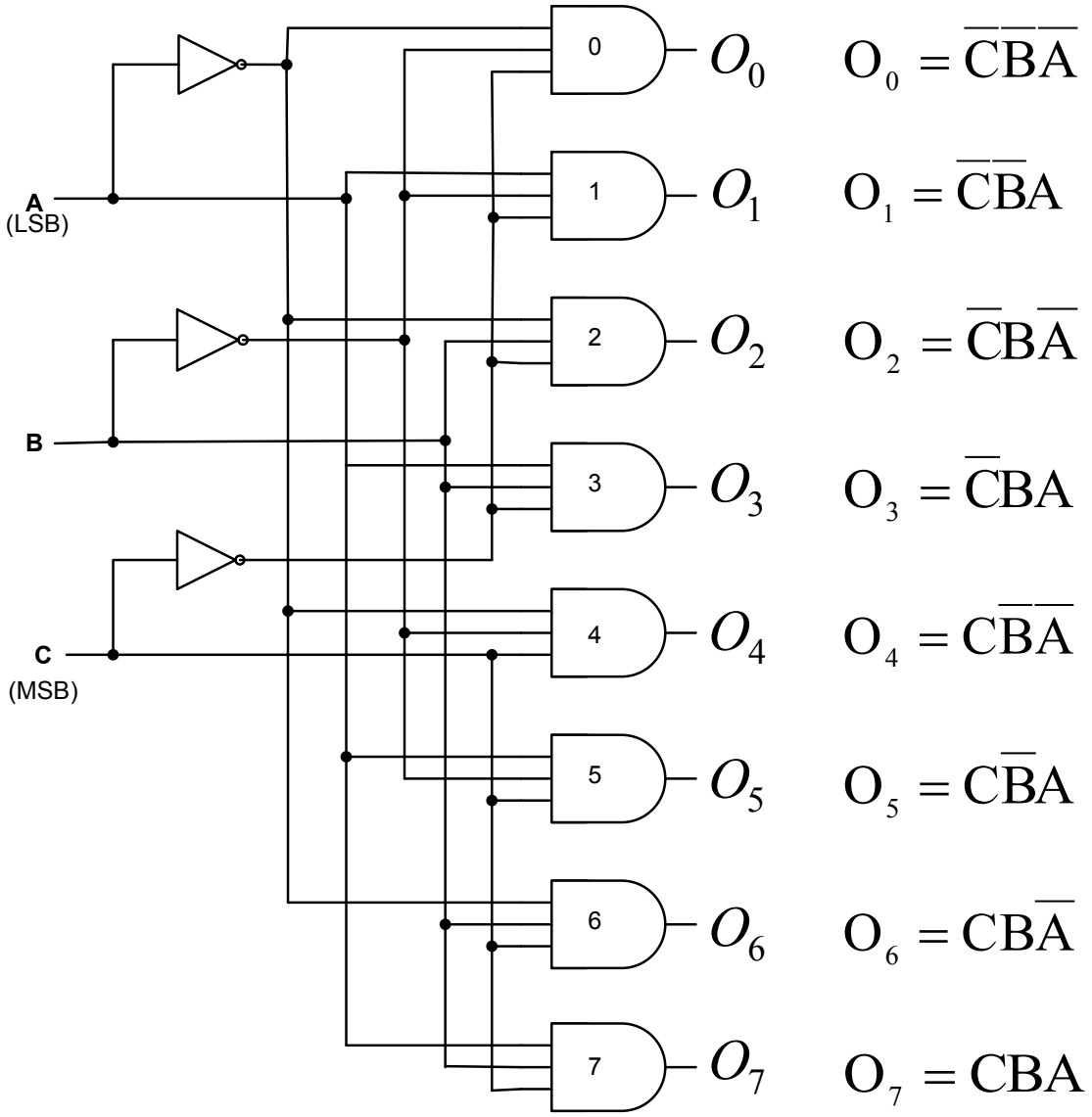
- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- If the n-bit decoded information has unused or don't-care combinations, the decoder output will have fewer than 2^n output.
- The decoders presented here are called n-to-m-line decoders, where m <= 2^n.
- Their purpose is to generate the 2^n (or fewer) minterms of n input variables.
- The name decoder is also used in conjunction with some code converters such as a BCD-to-seven segment decoder.
- Only one output can be active (high) at any times.

3-TO-8 - LINE DECODER

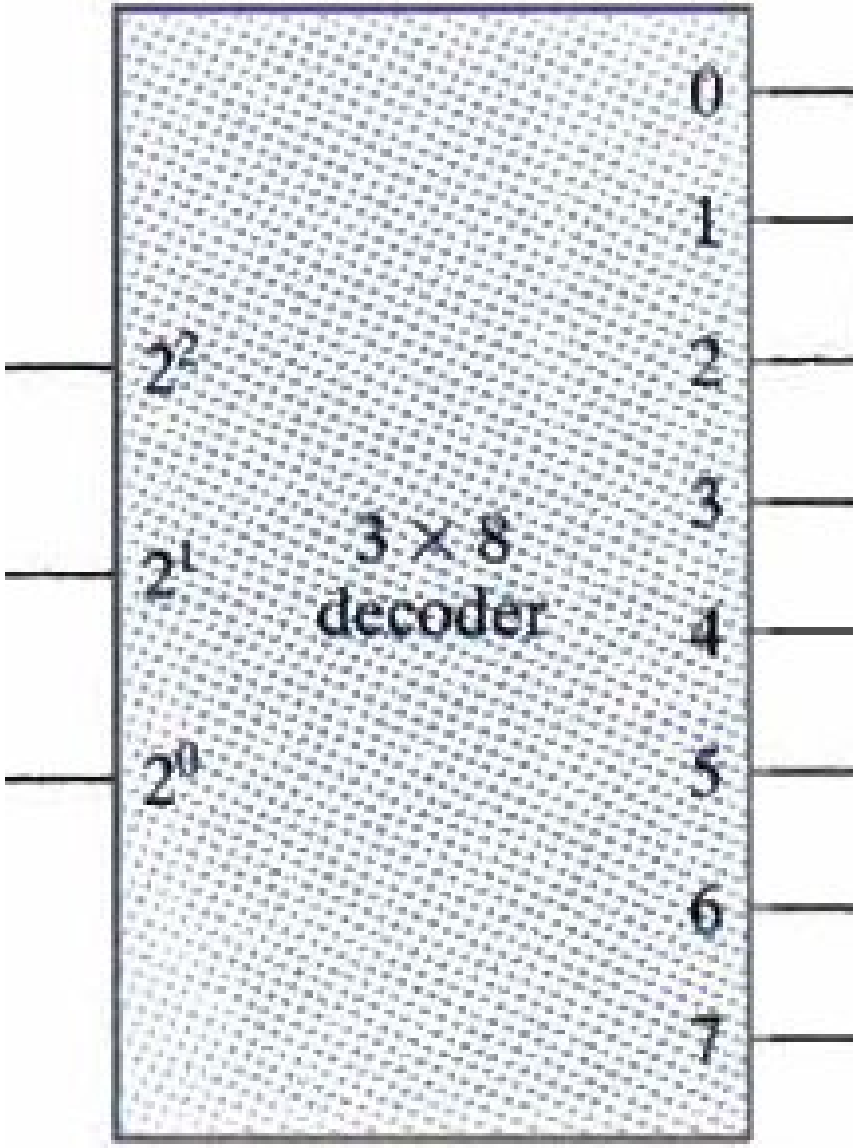
TRUTH TABLE

Inputs			Outputs							
C	**B**	**A**	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic ckt 3-line-to-8-line Decoder



BLOCK DIAGRAM 3-LINE-TO-8-LINE DECODER



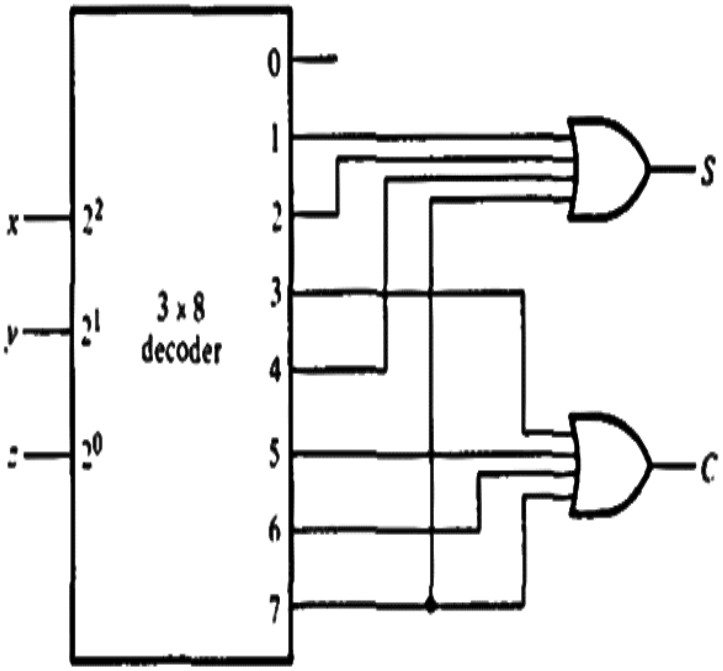
Combinational Logic Implementation using decoder

- A decoder provides the 2^n minterm of n input variables. Since any Boolean function can be expressed in sum of minterms canonical form, one can use a decoder to generate the minterms and an external OR gate to form the sum.
- In this way, any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n- line decoder and m OR gates.
- The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean functions for the circuit be expressed in sum of minterms.
- **Example:-**Implement a full-adder circuit with a decoder and two OR gates.
- From the truth table of the full-adder, we obtain the functions for this combinational circuit in sum of minterms:

$$S(x, y, z) = \sum(1, 2, 4, 7)$$
$$C(x, y, z) = \sum(3, 5, 6, 7)$$

Example: A combinational circuit is defined by the following three Boolean functions.
Design the circuit with a decoder and external gates.

F1 = x'y'z' + XZ
F2 = xy'z' + x'y
F3 = x'y'z + xy



Decoder with enable input

- Normally every commercially available decoder ICs have a special input other than normal working input variables called ENABLE.
- The use of this ENABLE input is that when activated the complete IC comes to the working condition for its normal functioning.
- If ENABLE input is deactivated the IC goes to sleep mode, the normal functioning is suspended, and all the outputs become logic 0 irrespective of normal input variables conditions.
- Its function is build higher decoder from lower decoders.

E	A	B	D_0	D_1	D_2	$_D3$
0	X	X	0	0	0	0
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

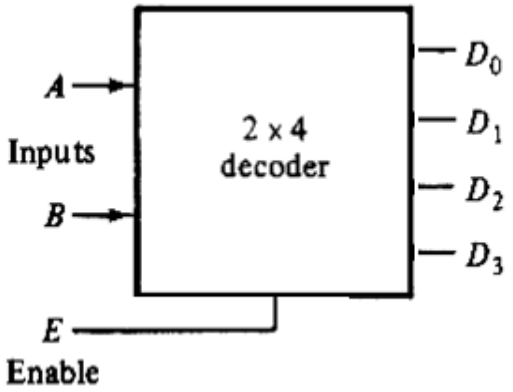
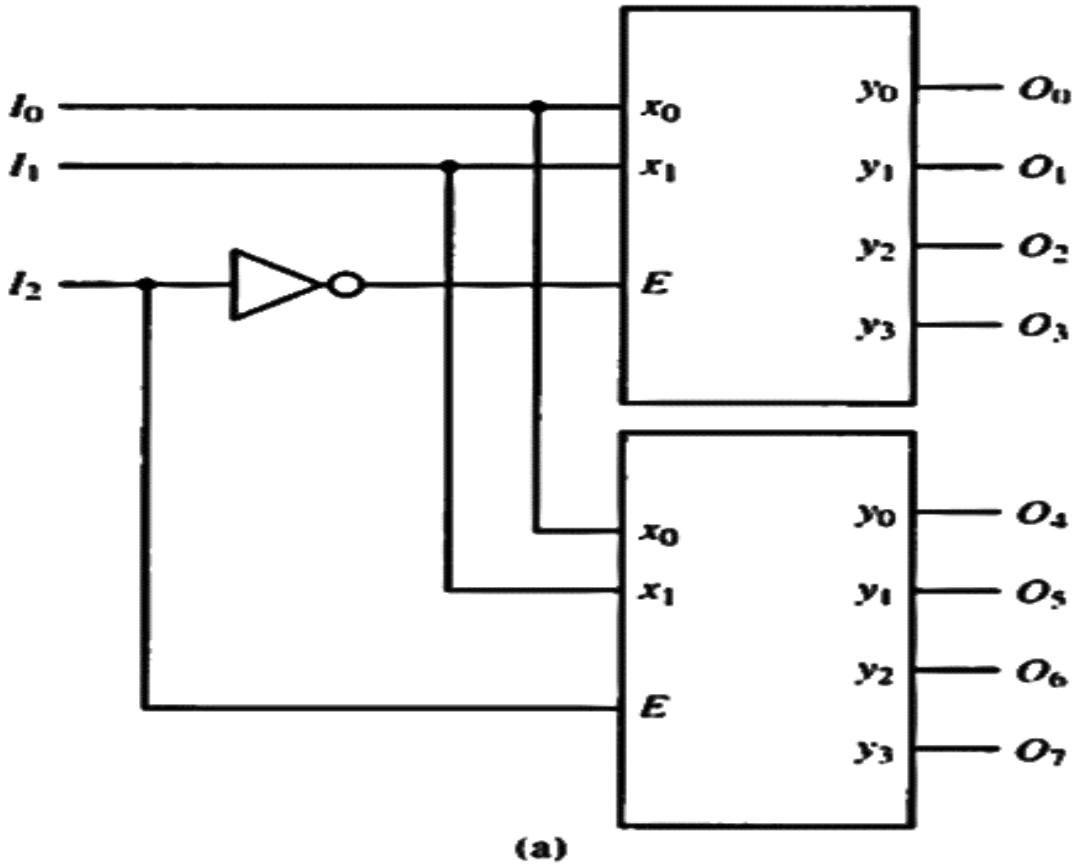


Figure: 2-to-4 decoder with enable input E.

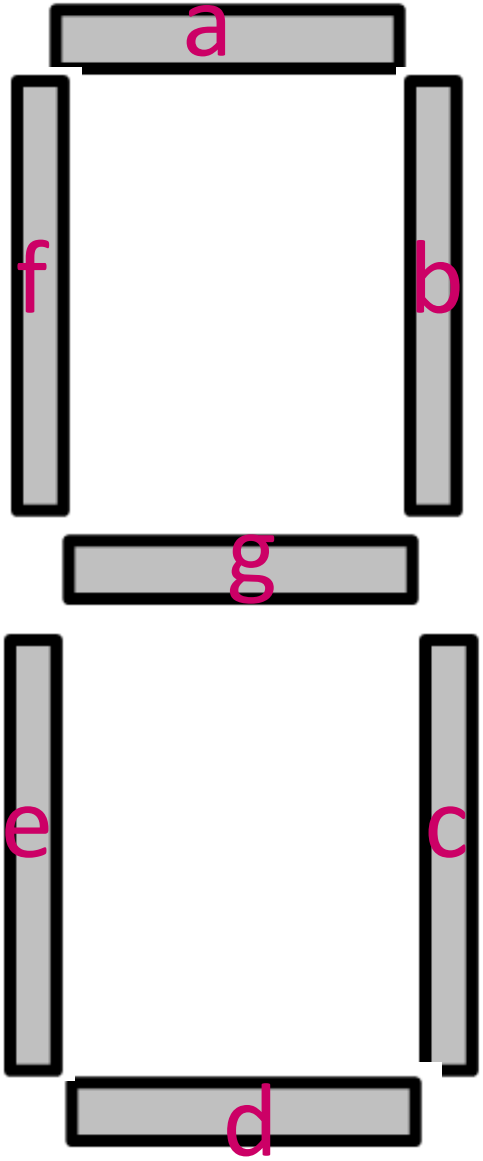
Cont…

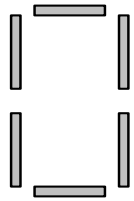
- Example:- Construct a 3-to-8 line decoder with the use of a 2-to-4 line decoder.



(a)

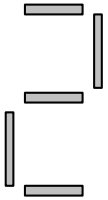
BCD-TO-7-SEGMENT DECODER



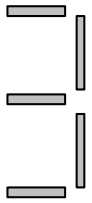


a, b, c, d, e & f

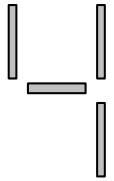
b & c



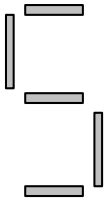
a, b, d, e & g



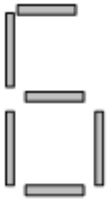
a, b, c, d, & g



b, c, f & g

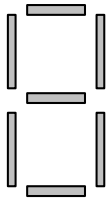


a, c, d, f & g

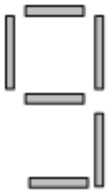


a, c, d, e, f & g

a, b & c

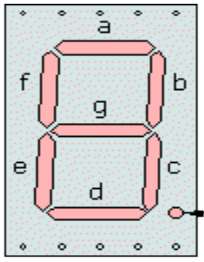


a, b, c, d, e, f & g



a, b, c, d, f & g

7-segment display



Truth table of 7-segments decoder

BCD inputs				*segment outputs*							*Decimal display*
D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9
1	0	1	0	x	x	x	x	x	x	x	----
:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	x	x	x	x	x	x	x	----

SOP Expression for all output

a =

b =

c =

d =

e =

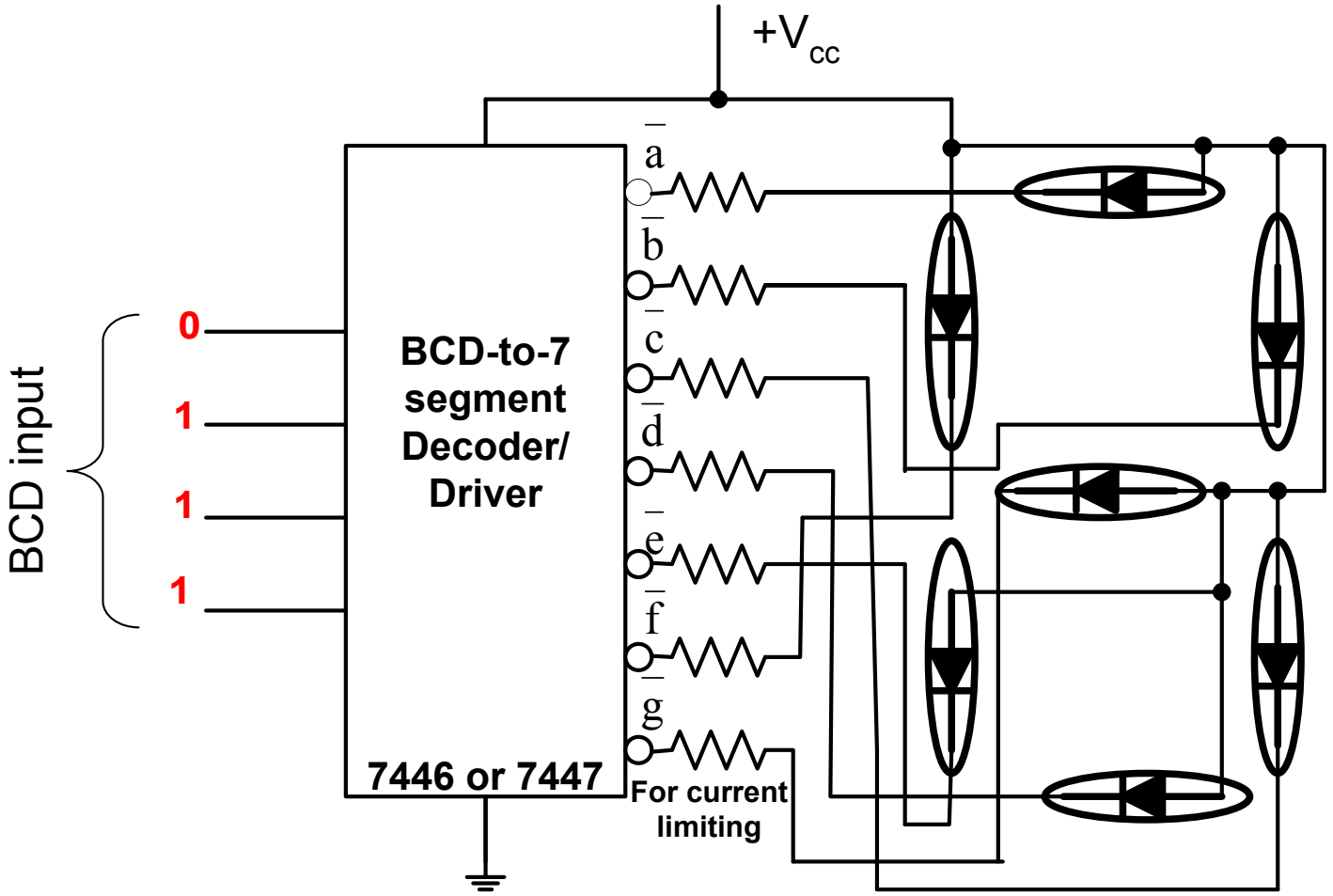
f =

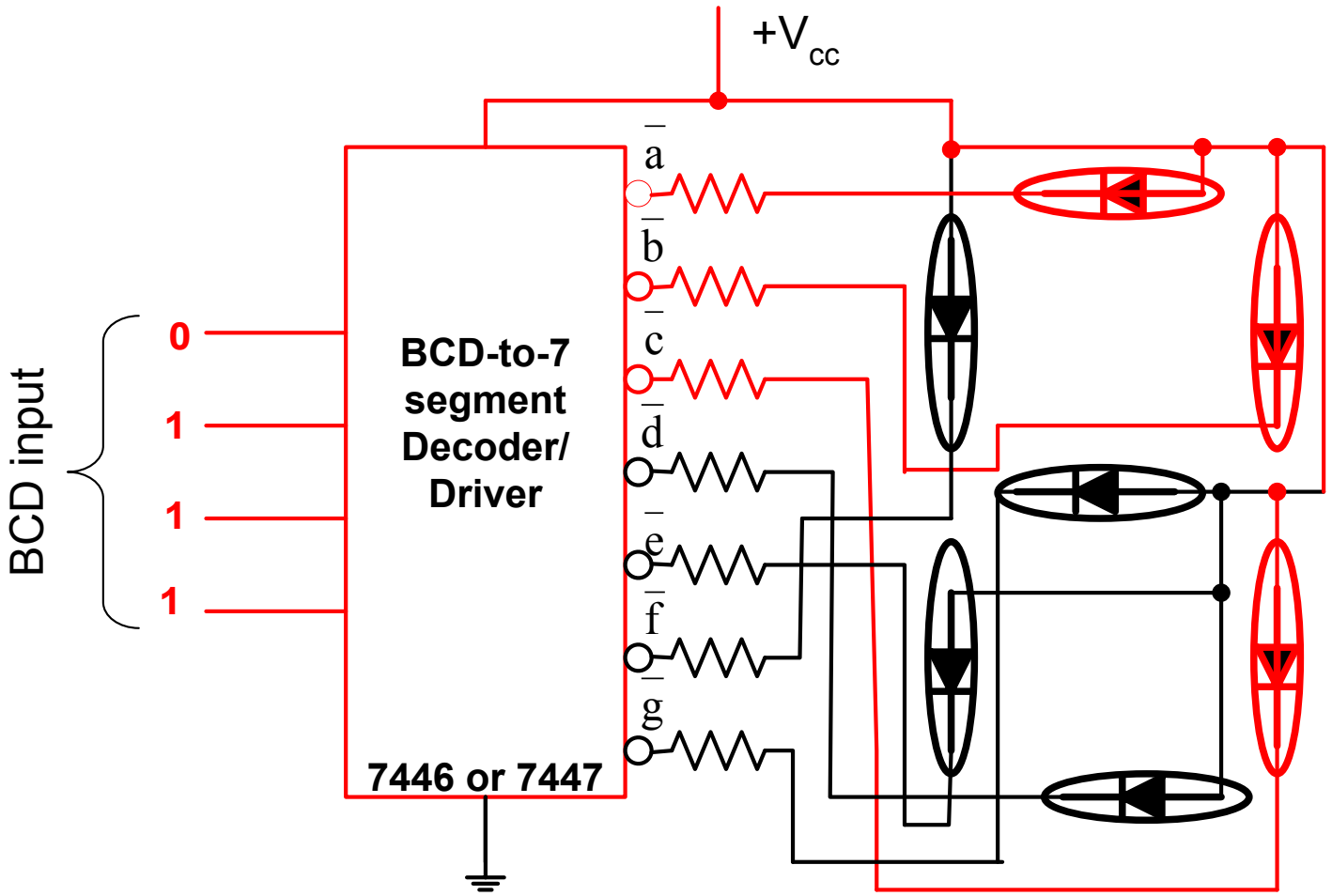
g =

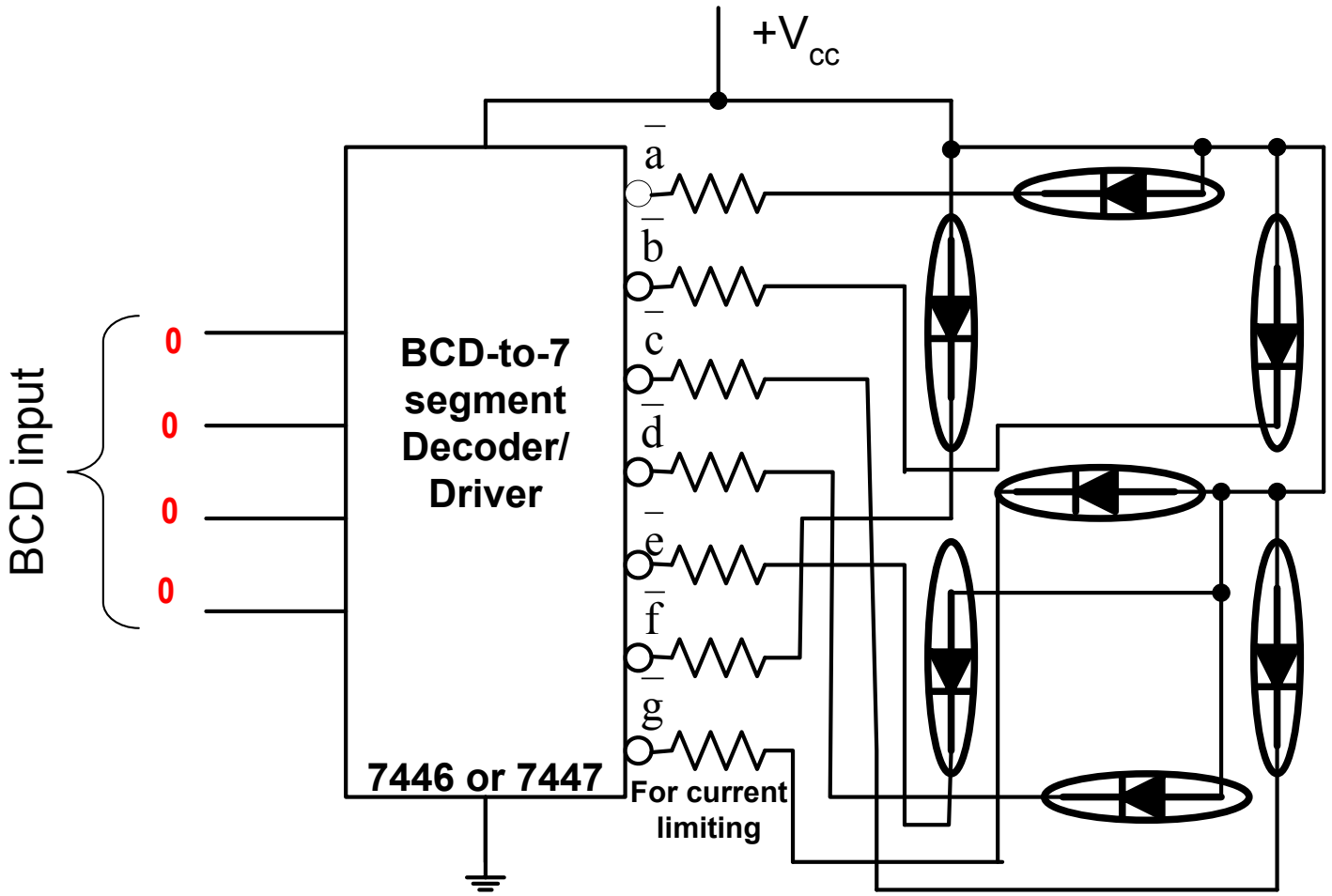
Logic Circuit of 7-segments decoder

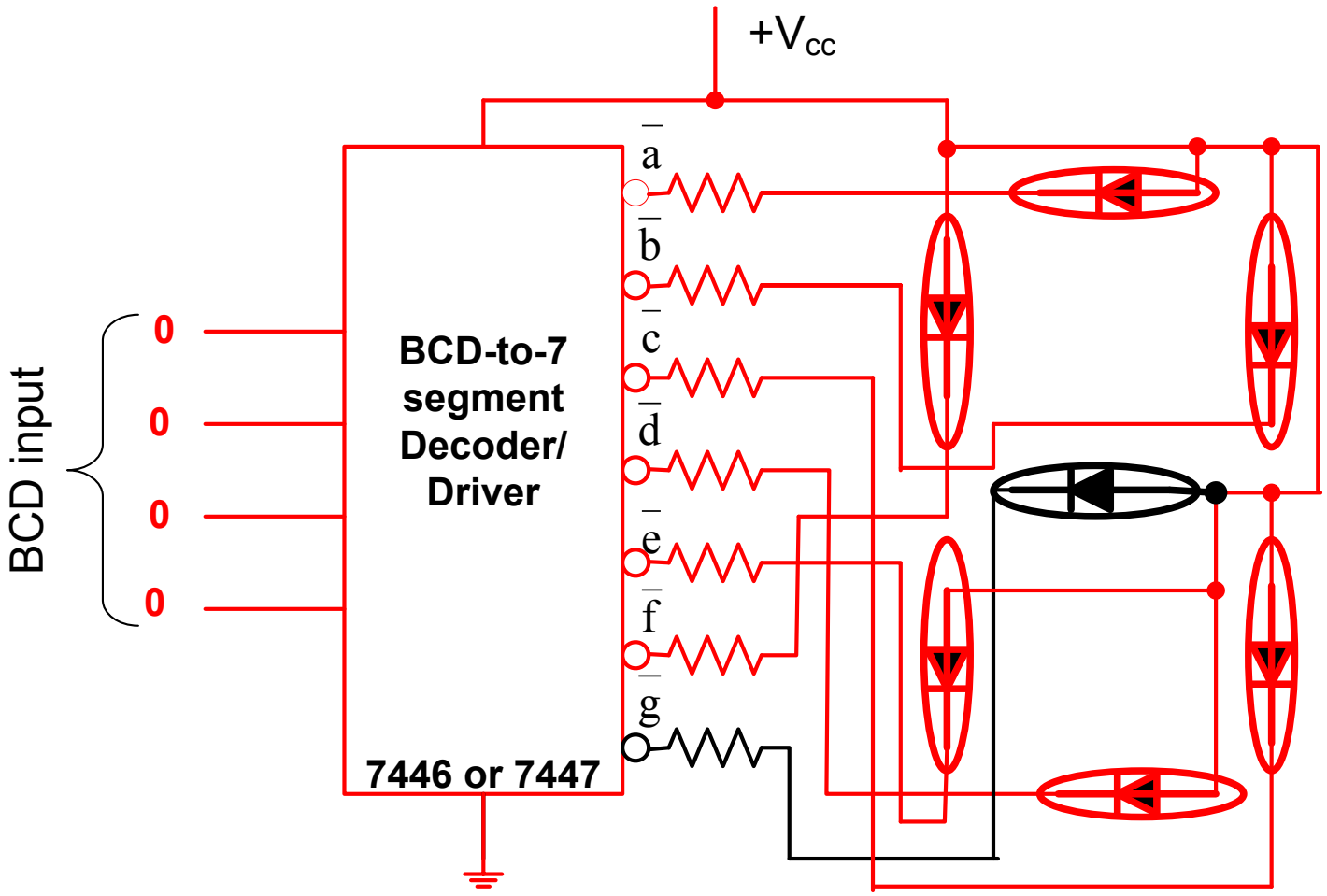
EXERCISE

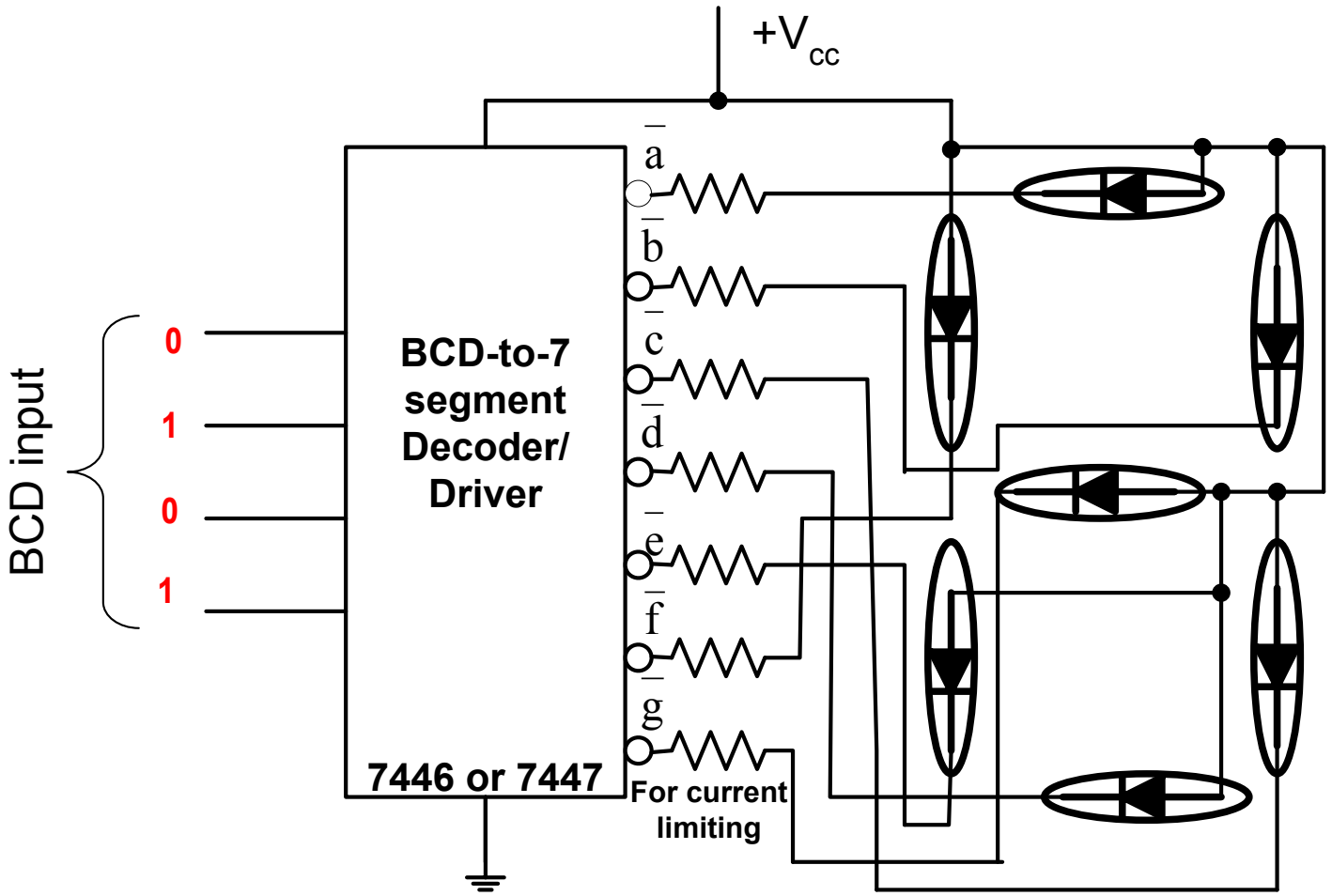
Practical Implementation BCD to7-segment decoder

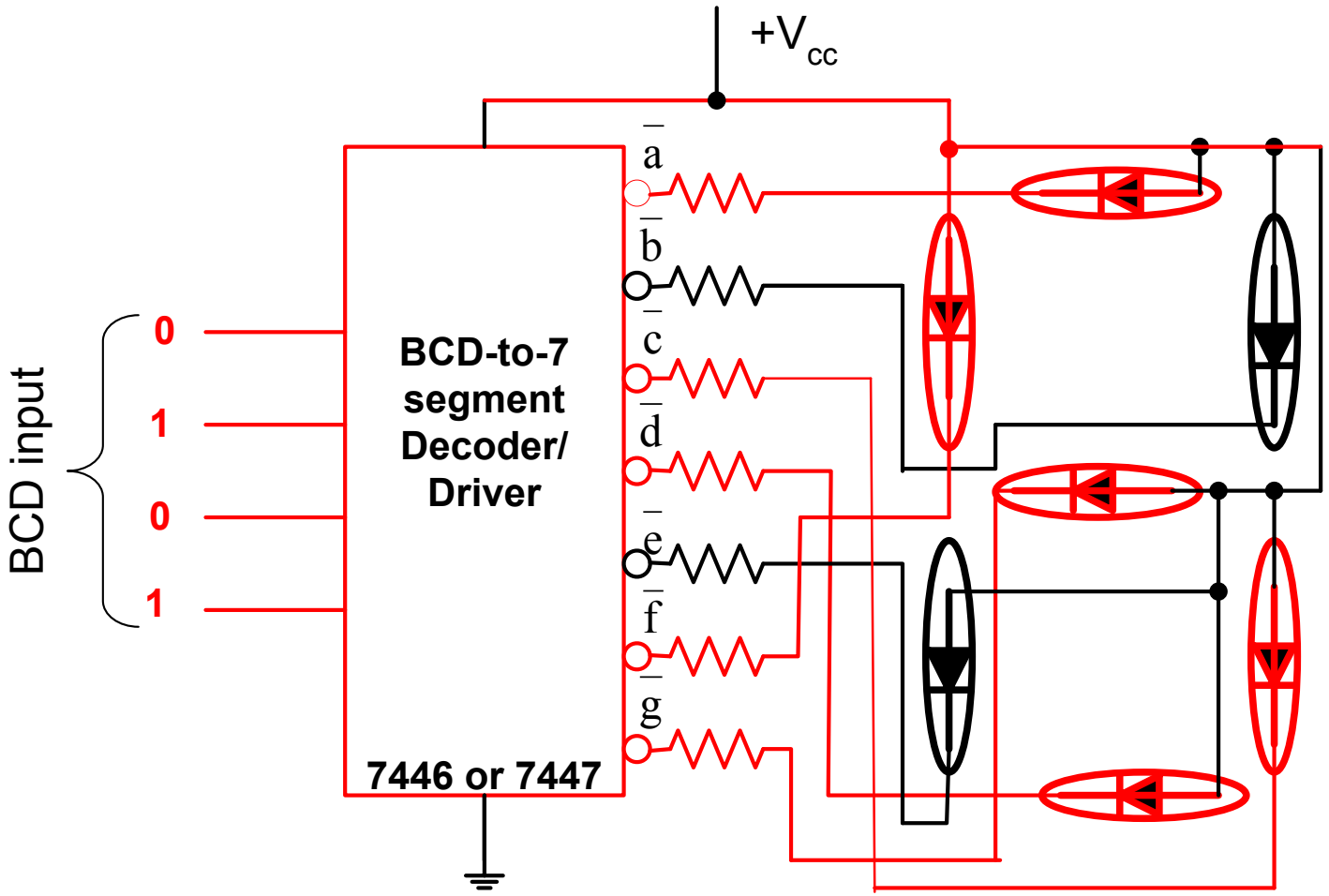



+Vcc
BCD input
0
1
1
1
BCD-to-7 segment Decoder/ Driver
7446 or 7447
a
b
c
d
e
f
g


+Vcc
BCD input
0
0
0
0
BCD-to-7 segment Decoder/ Driver
7446 or 7447
a
b
c
d
e
f
g
For current limiting


+Vcc
BCD input
0
0
0
0
BCD-to-7 segment Decoder/ Driver
7446 or 7447
a
b
c
d
e
f
g

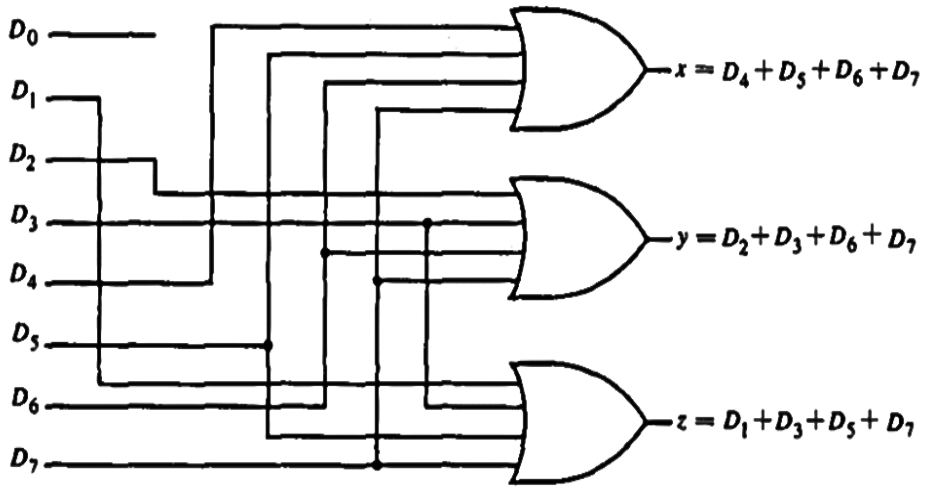

+Vcc
BCD input
0
1
0
1
BCD-to-7 segment Decoder/ Driver
7446 or 7447
a
b
c
d
e
f
g
For current limiting


+Vcc
BCD input
0
1
0
1
BCD-to-7 segment Decoder/ Driver
7446 or 7447
a
b
c
d
e
f
g

8. 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.
- The output lines generate the binary code corresponding to the input value.
- It is assumed that only one input has a value of 1 at any given time; otherwise the circuit has no meaning.
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.
- These conditions can be expressed by the following output Boolean functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$



Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

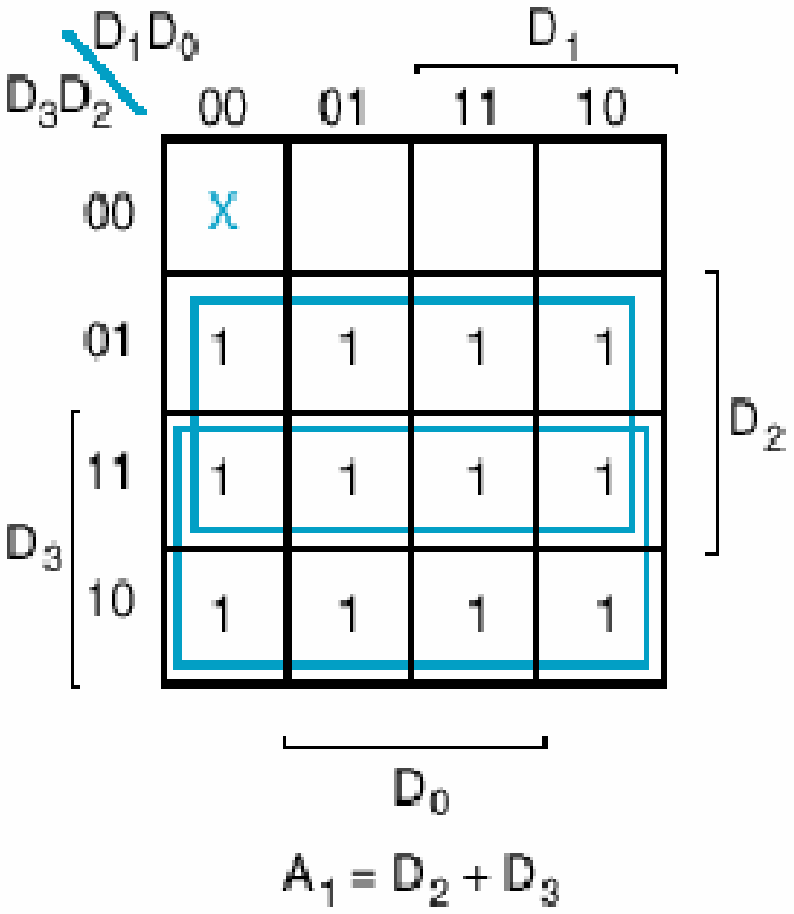
Truth table of 8x3 encoder

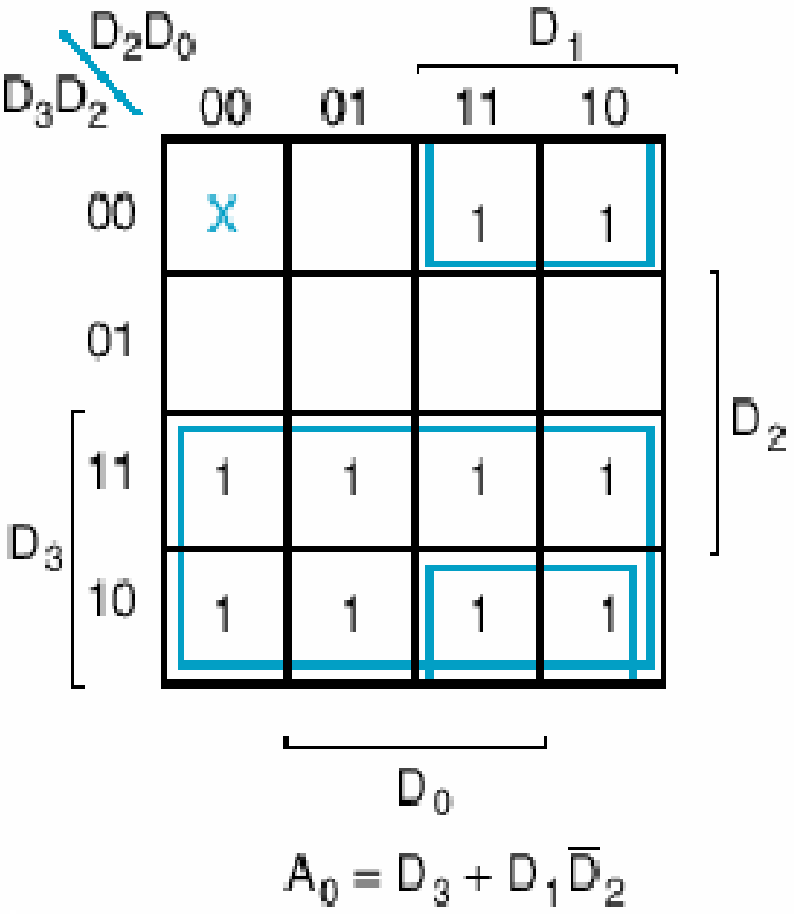
Priority Encoder

- Accepts multiple values and encodes them
  - Works when more than one input is active
- Consists of:
  - Inputs (2^n)
  - Outputs
    - when more than one output is active, sets output to correspond to highest input
    - V (indicates whether any of the inputs are active)
  - Selectors / Enable (active high or active low)

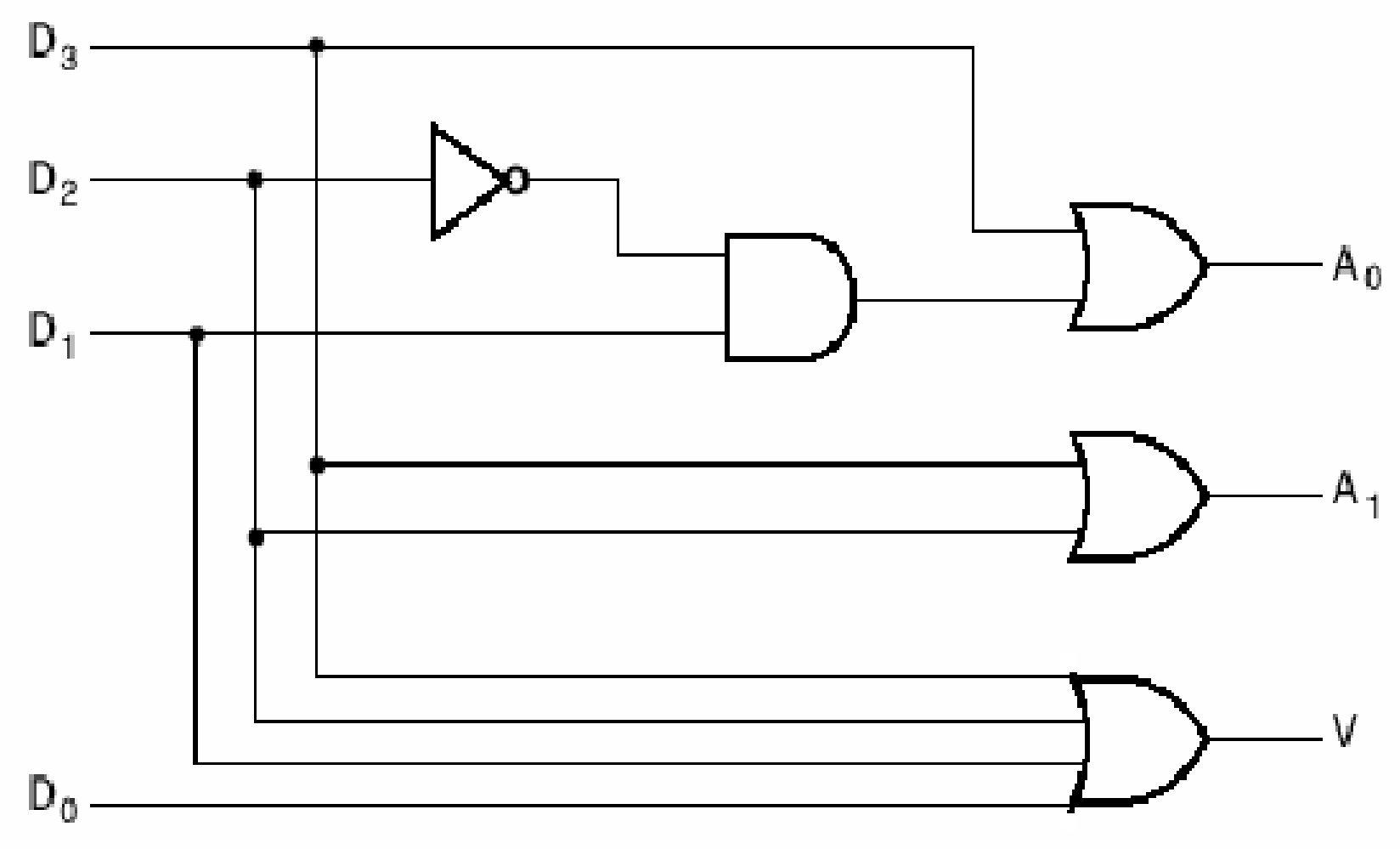
D3	D2	D1	D0	A1	A0	V
0	0	0	0	x	X	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Priority -------


D1D0
D3D2
00
01
11
10
D1
D2
D3
D0
X
1
A1 = D2 + D3

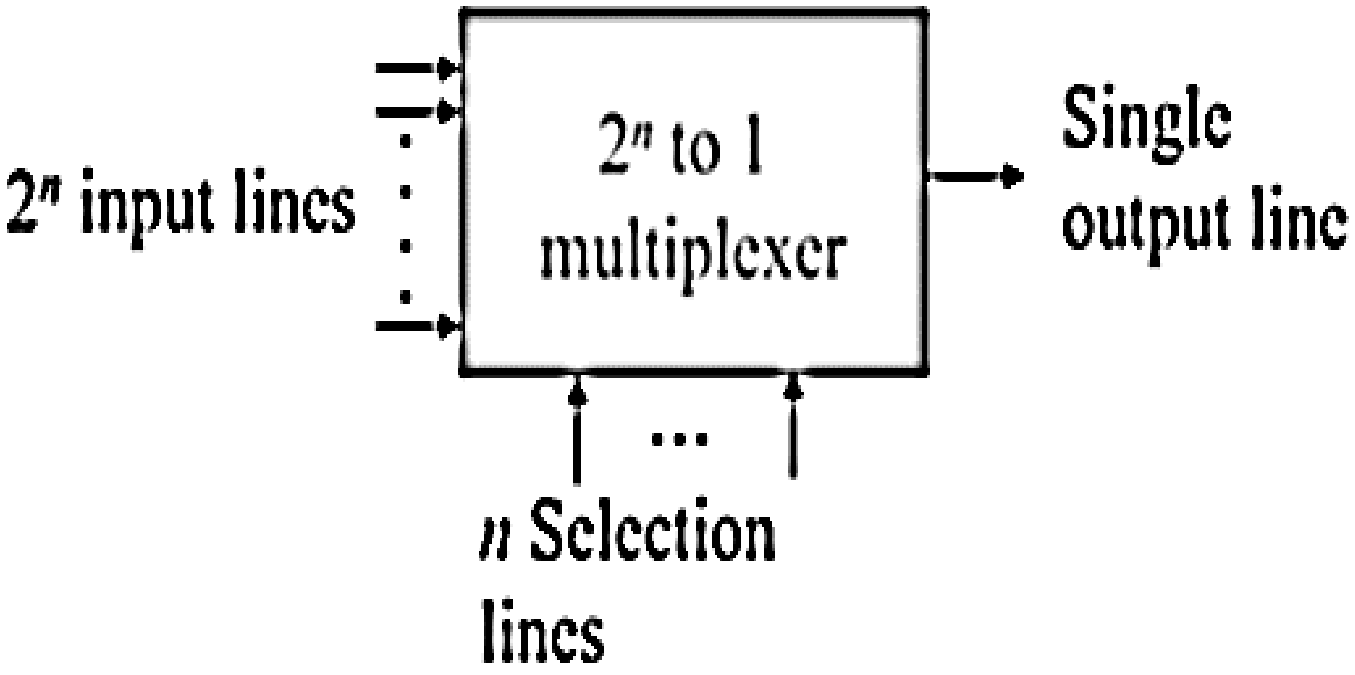

D2D0
D3D2
00
01
11
10
D1
D2
D3
D0
X
1
A0 = D3 + D1D̄2

Priority Encoder



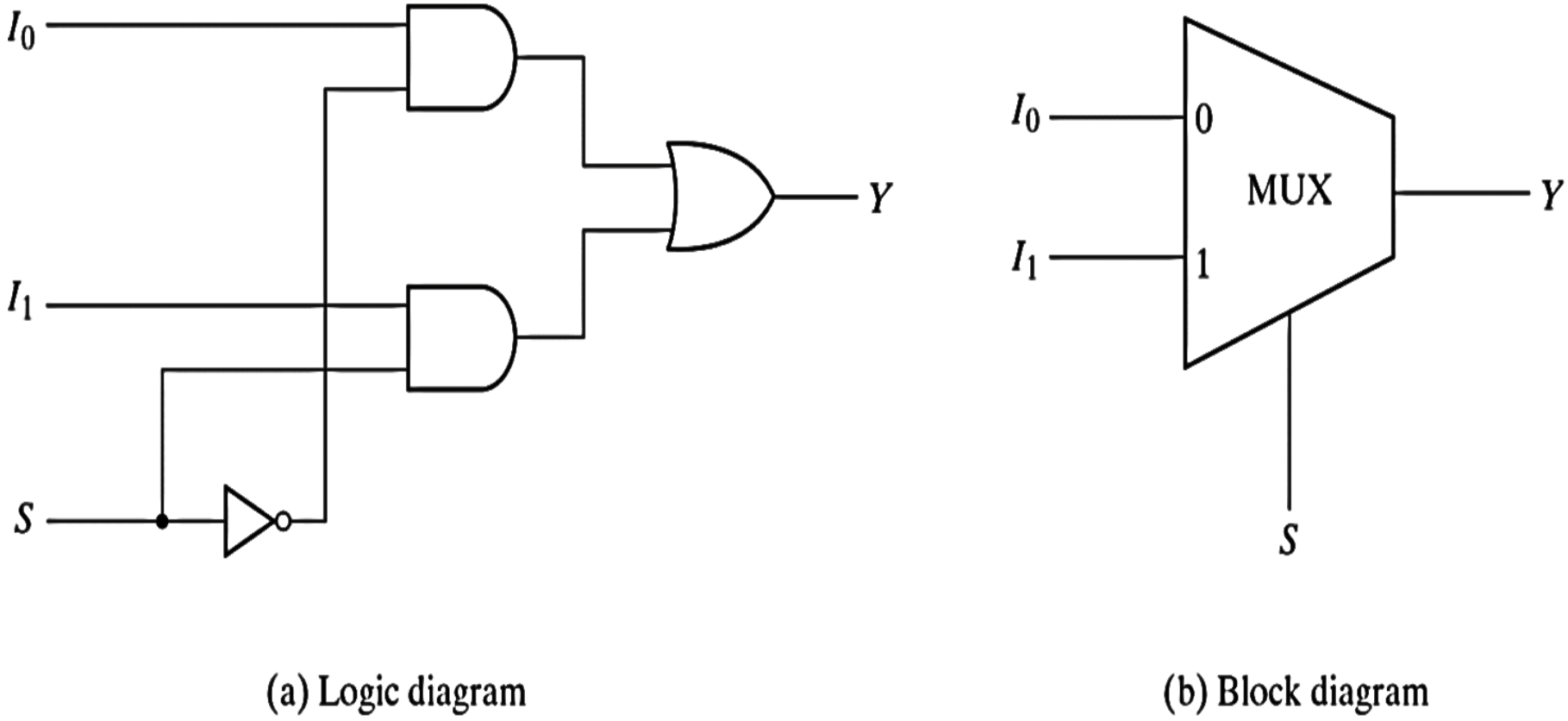
9. MULTIPLEXERS

- A digital multiplexer is a combinational circuit that selects binary information from one of the 2^n input channels and transmits to a single output line.
- That is why the multiplexers are also called data selectors.
- The selection of the particular input channel is controlled by a set of select inputs.
- Select binary information from one of many input lines and direct it to a single output line
- 2^n input lines, n selection lines and one output line



cont---

Example: 2-to-1-line multiplexer



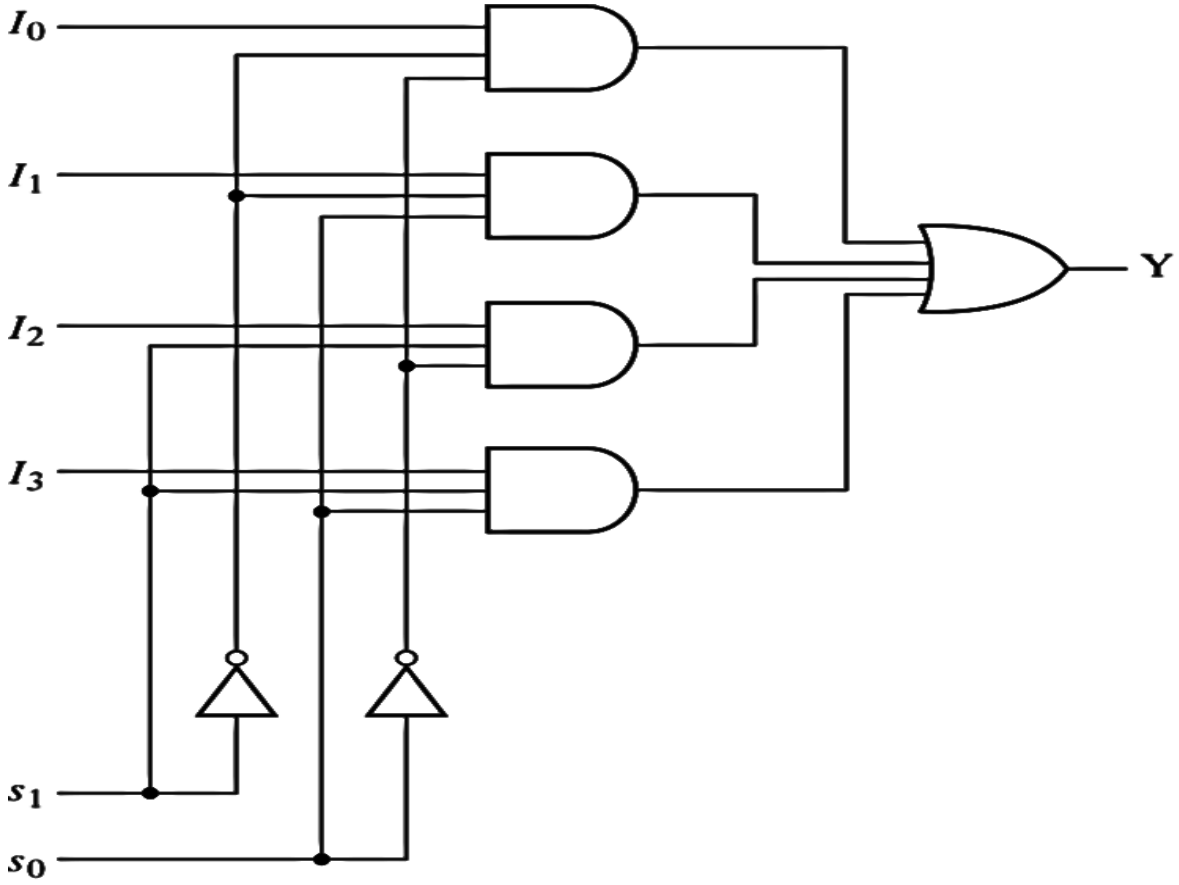
(a) Logic diagram

(b) Block diagram

Figure: 2-to-1 line multiplexer

Cont..

- **Example: a logic circuit, for 4x1 multiplexer**



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

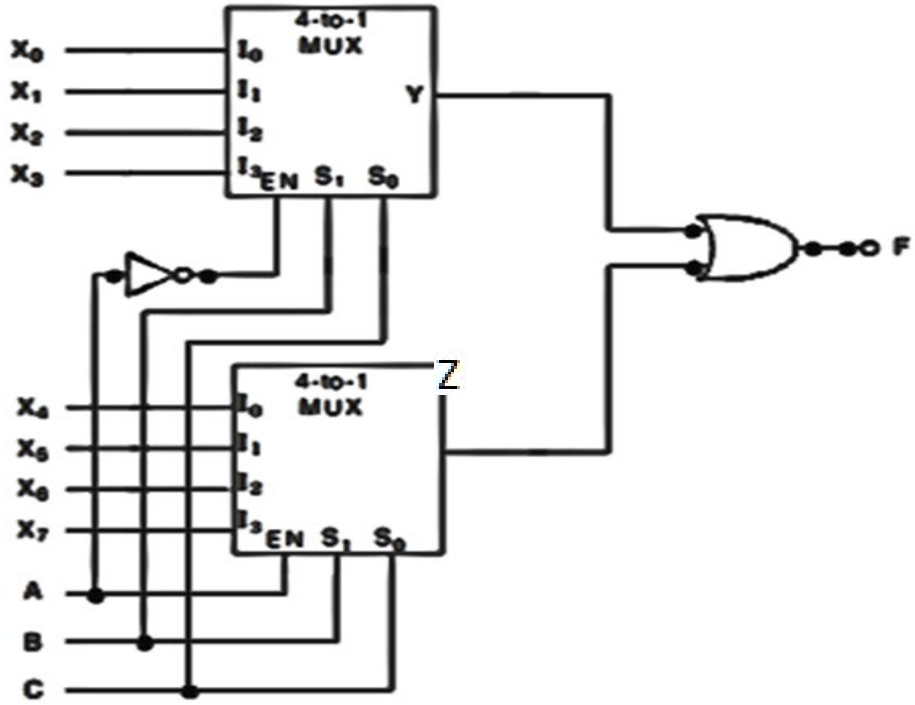
(b) Function table

Figure: 4-to-1 line multiplexer

Cont….

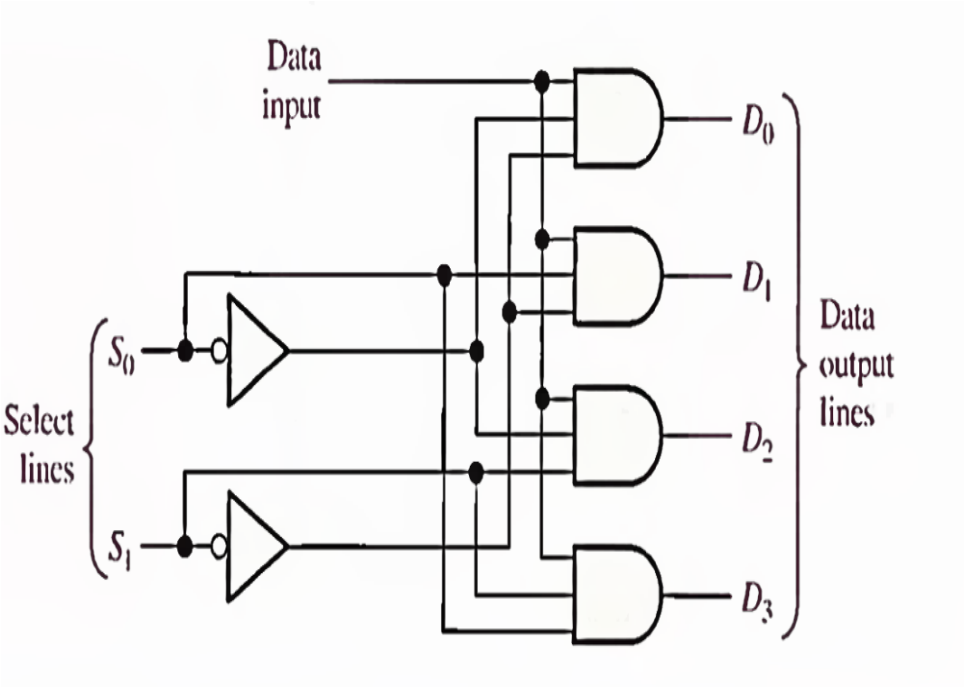
- As in decoders, multiplexer ICs may have an enable input to control the operation of the unit.
- When the enable input is in a given binary state, the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as a normal multiplexer.
- The enable input (sometimes called strobe) can be used to expand two or more multiplexer ICs to a digital multiplexer with a larger number of inputs.
- Example:- Construct a 8-to-1 multiplexer with the use of a 4-to-1 line multiplexer and external gate.

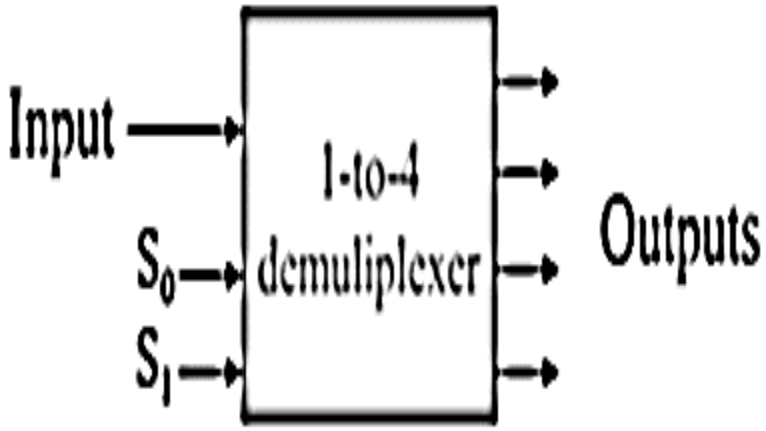
- Example:- Construct a 16-to-1 multiplexer with the use of only a 4-to-1 line multiplexer.



10. Demultiplexer

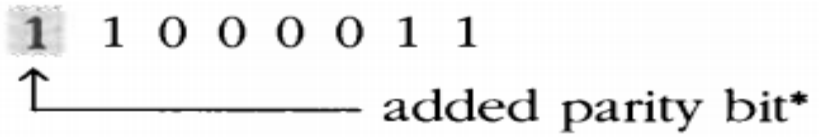
- A demultiplexer is a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- The selection of a specific output is controlled by the bit combination of n elected lines.
- The term “demultiplex” means one into many.





11. PARITY GEINERATOR AND CHECKER

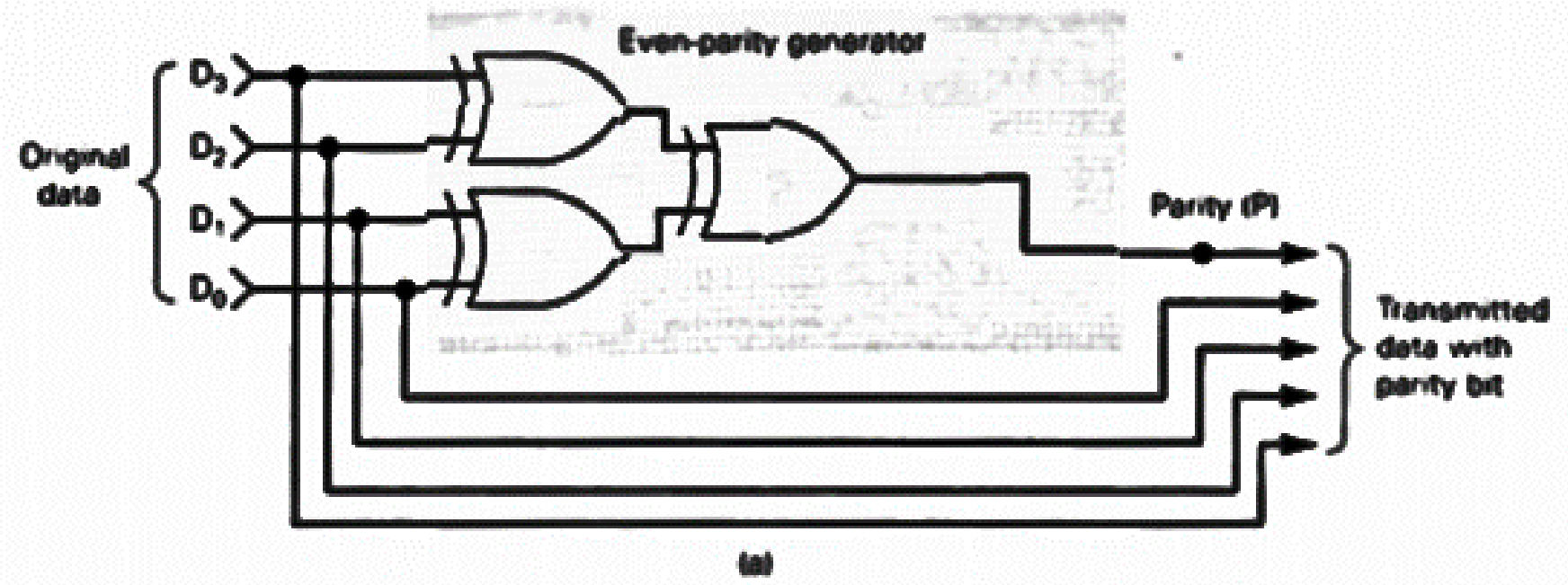
- Whenever information is transmitted from one device (the transmitter) to another device (the receiver), there is a possibility that errors can occur such that the receiver does not receive the identical information that was sent by the transmitter.
- One of the simplest and most widely used schemes for error detection is the parity method.
- **Parity Bit**:- A parity bit is an extra bit that is attached to a code group that is being transferred from one location to another.
- The parity bit is made either 0 or 1, depending on the number of 1 is that are contained in the code group. Two different methods are used.
  - ✓ The even-parity method, and
  - ✓ The odd-parity method.
- In the even-parity method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an even number.



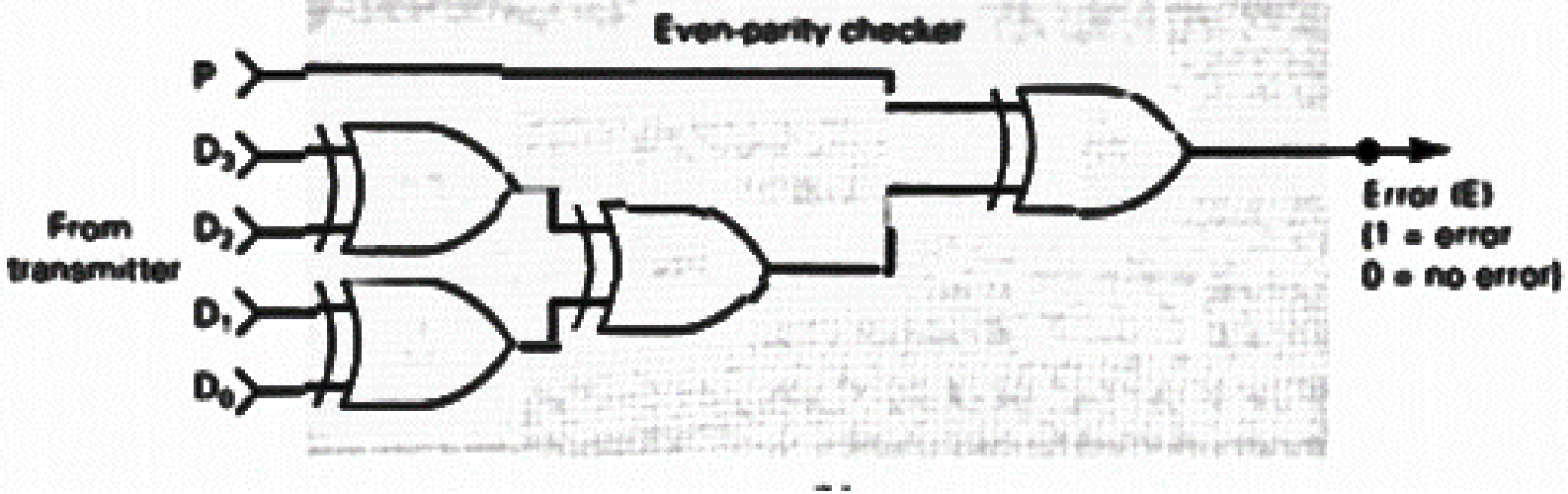
Cont…

- In the odd-parity method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an odd number.
- A parity generator is a combination logic system to generate the parity bit at the transmitting side.
- Parity Checker:- The message bits with the parity bit are transmitted to their destination, where they are applied to a parity checker circuit.
- The circuit that checks the parity at the receiver side is called the parity checker.
- The parity checker circuit produces a check bit and is very similar to the parity generator circuit.
- If the check bit is 1, then it is assumed that the received data is incorrect. The check bit will be 0 if the received data is correct.

Cont…


Even-parity generator
Original data
D_3
D_2
D_1
D_0
Parity (P)
Transmitted data with parity bit

(a)


Even-parity checker
P
D_3
D_2
D_1
D_0
From transmitter
Error (E)
(1 = error
0 = no error)

END OF CHAPTER-5

QUESTION??? COMMENT