Combinational Circuits

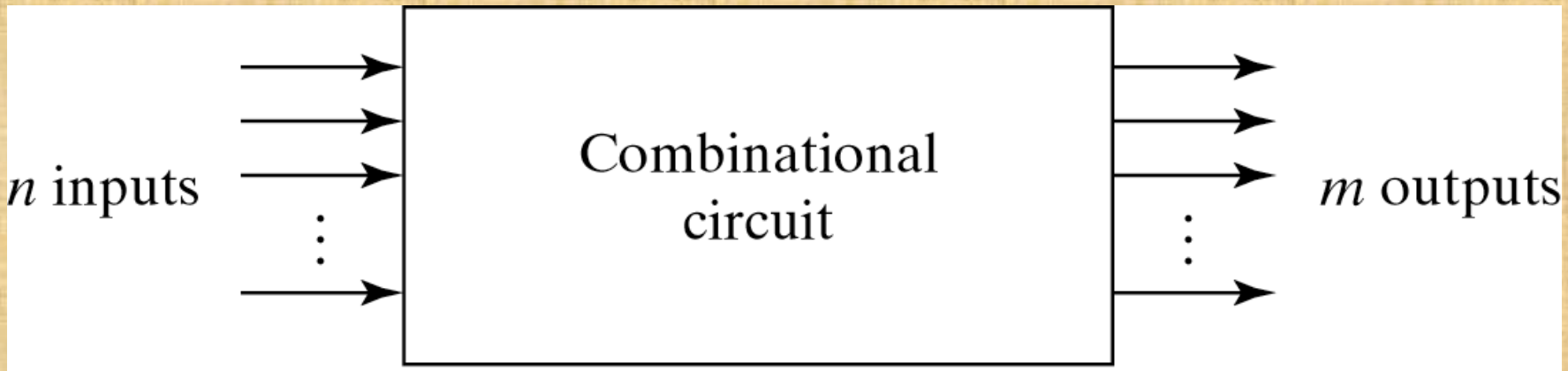


Fig. 4-1 Block Diagram of Combinational Circuit

What is Combinational Circuits?

- A Combinational Circuit is a combination of Logic gates, the output depends upon the current value of the inputs.

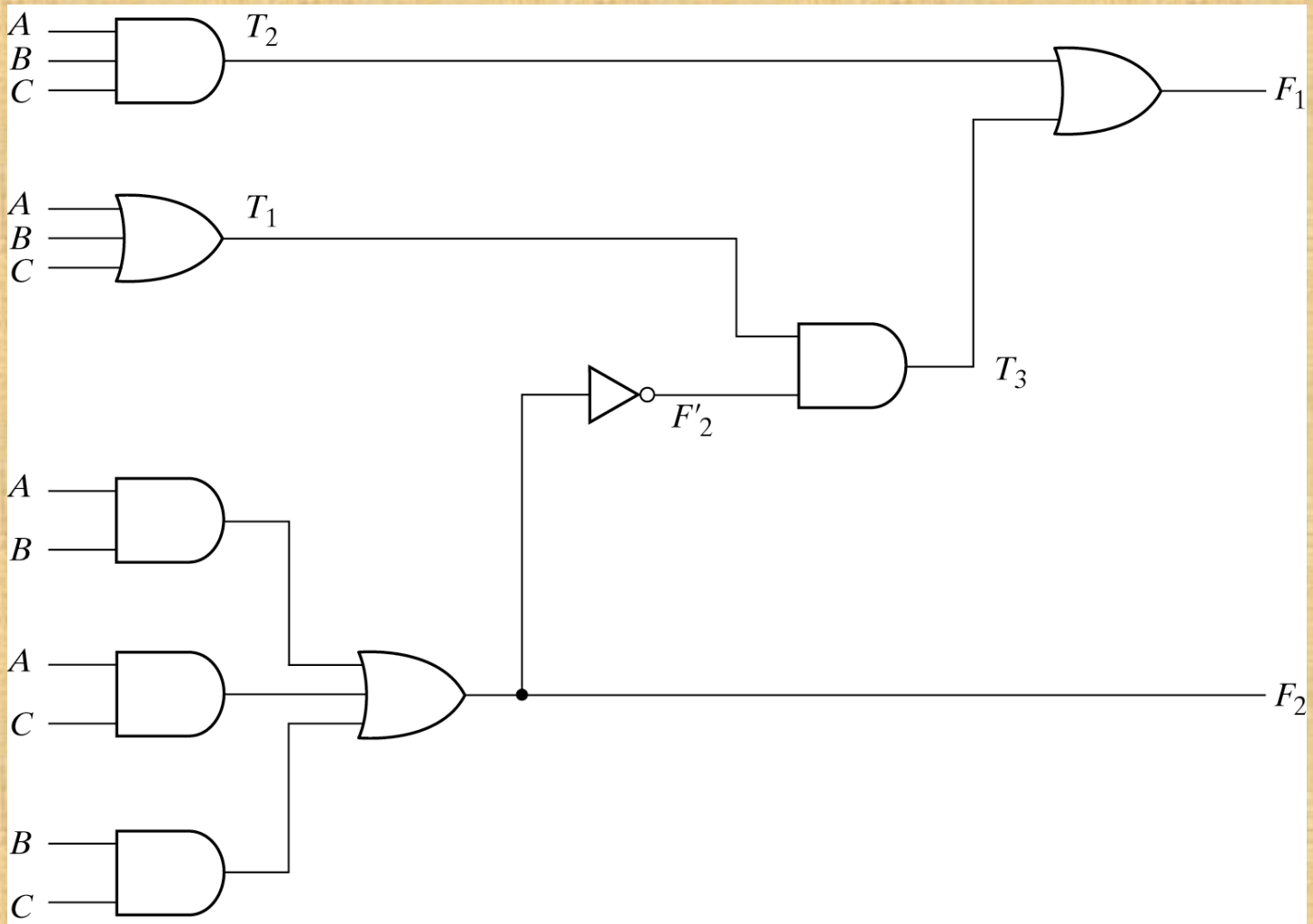


Fig. 4-2 Logic Diagram for Analysis Example

Examples of Combinational Circuits

- Addition:
  - **Half Adder (HA).**
  - **Full Adder (FA).**
  - **BCD(Decimal) Adder.**
- Subtraction:
  - **Half Subtractor.**
  - **Full Subtractor.**
- Multiplication:
  - **Binary Multipliers.**
- Comparator:
  - **Magnitude Comparator.**

Examples of Combinational Circuits

- Multiplexers
- Demultiplexers
- Encoders
- Decoders
- Converters
  - Binary to Gray Code
  - Gray to Binary Code
  - Binary to BCD Code

Two types of questions come in the exam based on Combinational Circuit:

1.Designing of a combinational Circuit

2.Analysis of Combinational Circuit

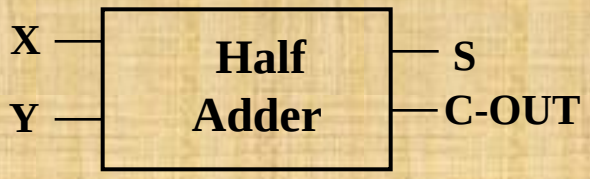
Designing Combinational Circuits

In general we have to do following steps:

1. Problem description
2. Input/output of the circuit
3. Define truth table
4. Simplification for each output
5. Draw the circuit

Half Adder

- Adding two single-bit binary values, X, Y produces a sum S bit and a carry out C-out bit.
- This operation is called half addition and the circuit to realize it is called a half adder.



Half Adder Truth Table

Inputs		Outputs	
X	Y	S	C-out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

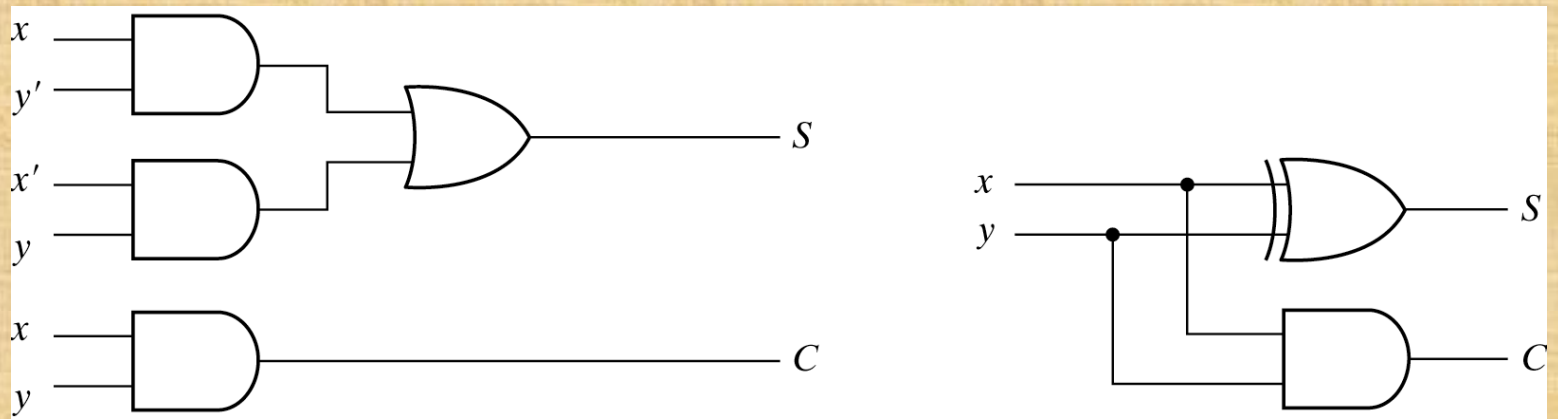
$S(X,Y) = \Sigma\,(1,2)$

$S = X'Y + XY'$

$S = X \oplus Y$

$C\text{-out}(x, y) = \Sigma\,(3)$

$C\text{-out} = XY$



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

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

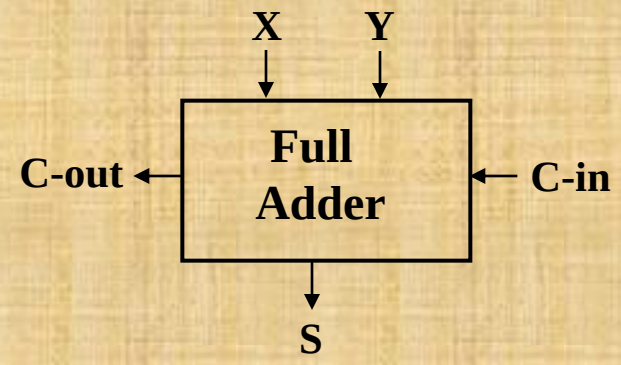
Fig. 4-5 Implementation of Half-Adder

Full Adder

- Adding two single-bit binary values, X, Y with a carry input bit C-in produces a sum bit S and a carry out C-out bit.

Full Adder Truth Table

Inputs			Outputs	
X	Y	C-in	S	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Full Adder

X	Y	C-in	S	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

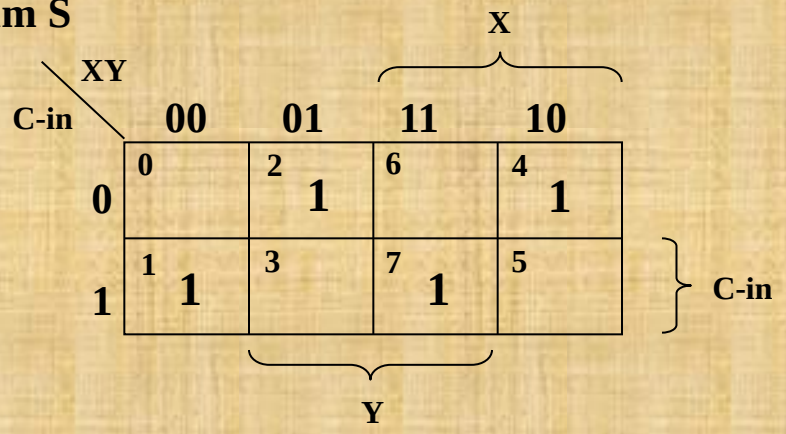
Inputs Outputs

Full Adder Truth Table

$S(X,Y, C\text{-in}) = \Sigma\ (1,2,4,7)$

$C\text{-out}(x, y, C\text{-in}) = \Sigma\ (3,5,6,7)$

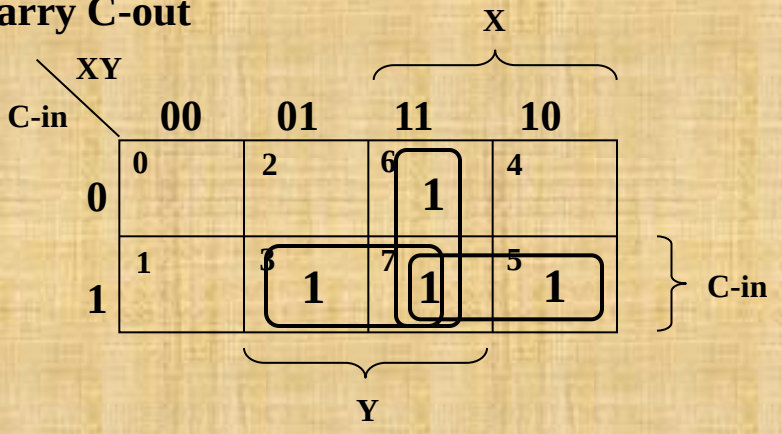
Sum S



$S = X'Y'(C\text{-in}) + XY'(C\text{-in})' + XY'(C\text{-in})' + XY(C\text{-in})$

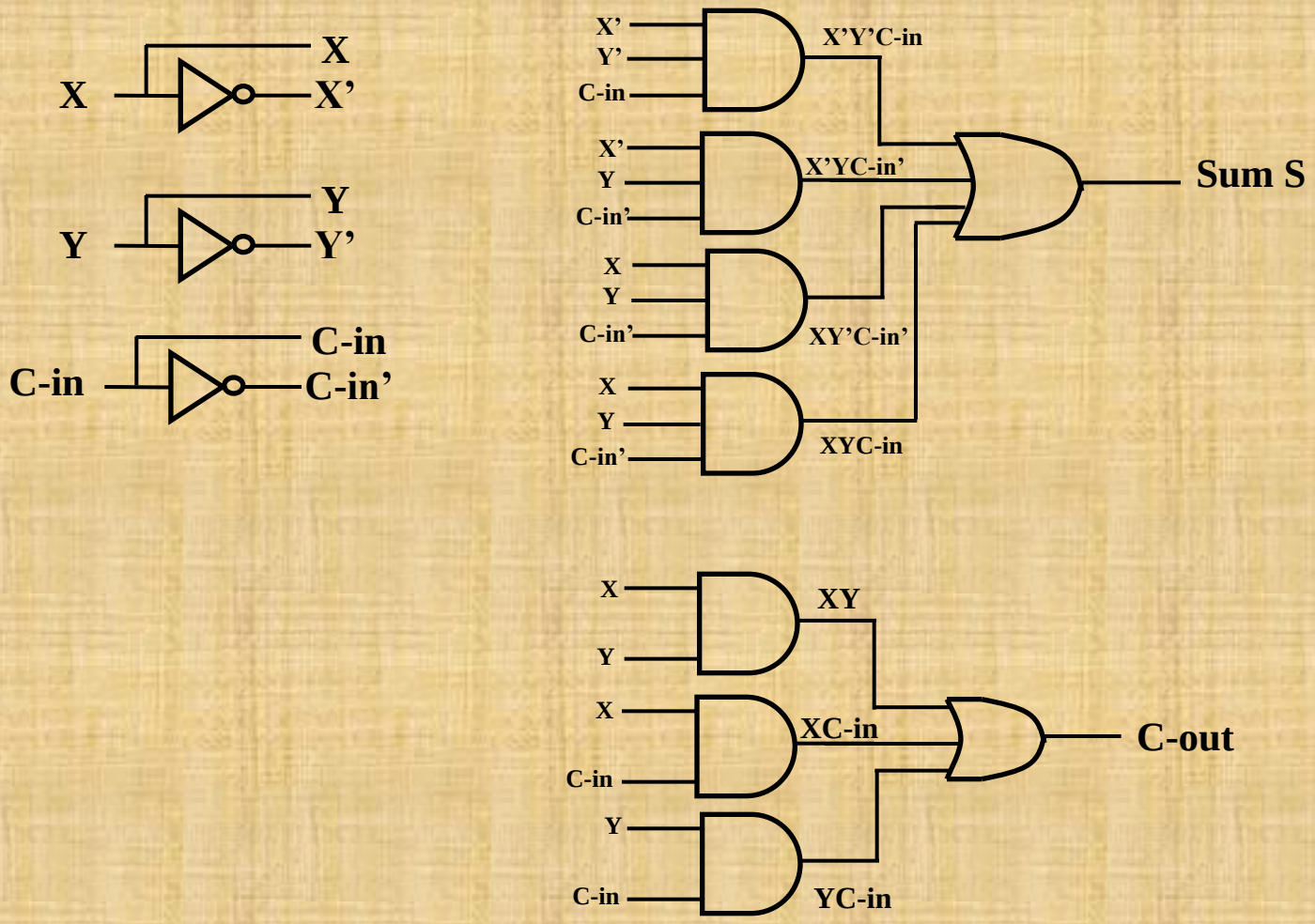
$S = X \oplus Y \oplus (C\text{-in})$

Carry C-out

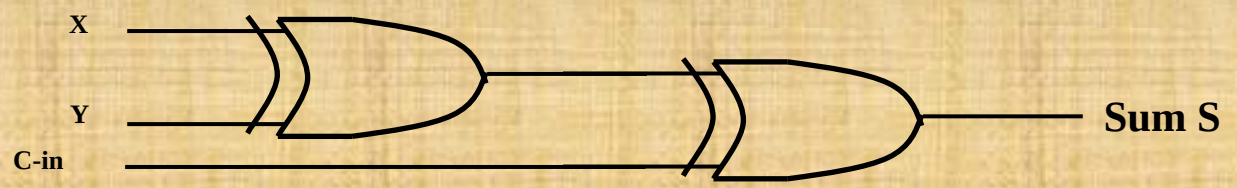


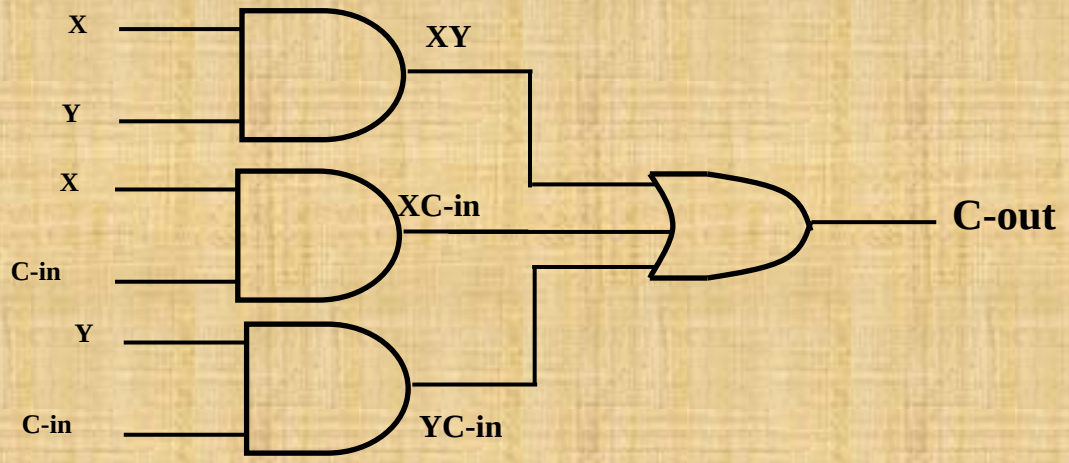
$C\text{-out} = XY + X(C\text{-in}) + Y(C\text{-in})$

Full Adder Circuit Using AND-OR



Full Adder Circuit Using Ex-OR





Full Adder Circuit Using two half - Adders

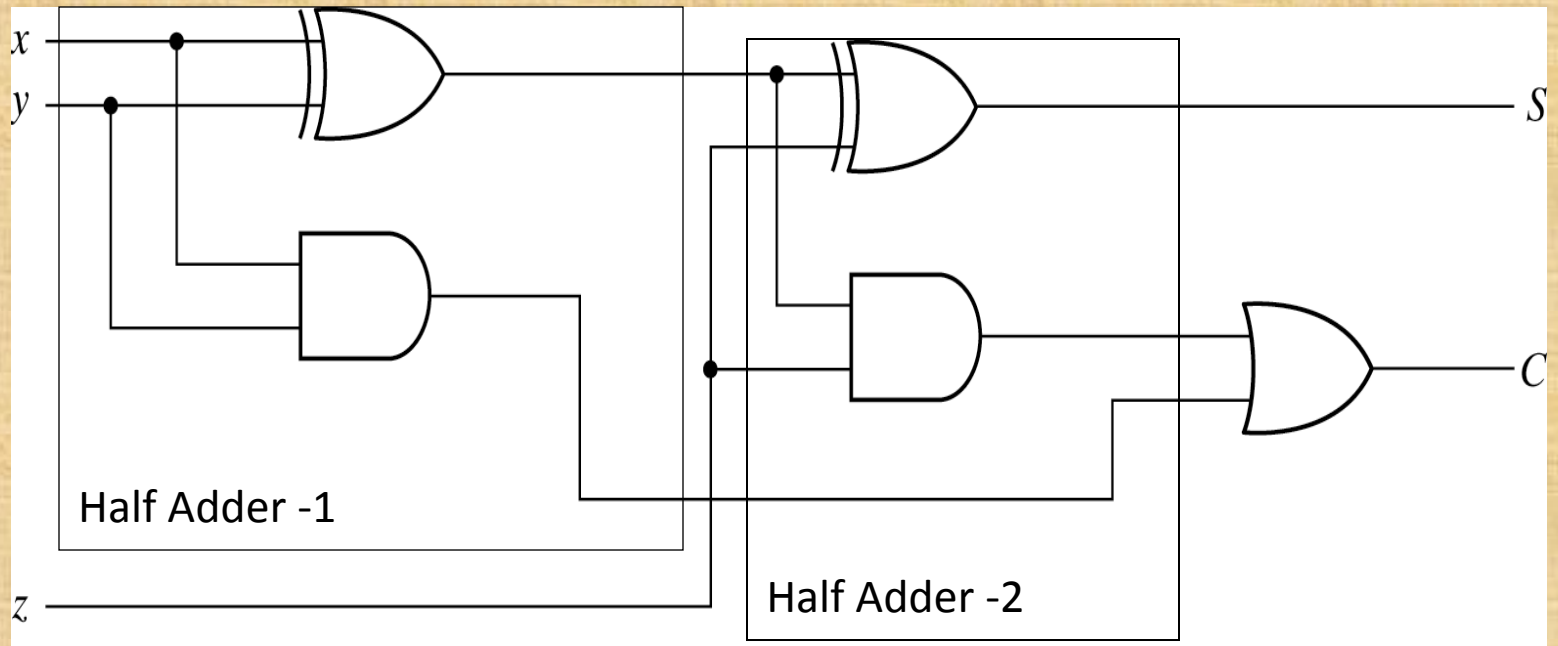


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary adder

- Binary adder that produces the arithmetic sum of binary numbers can be constructed with full adders connected in cascade, with the output carry from each full adder is connected to the input carry of the next full adder in the chain
- Note that the input carry C_0 in the least significant position must be 0.

Binary Adder

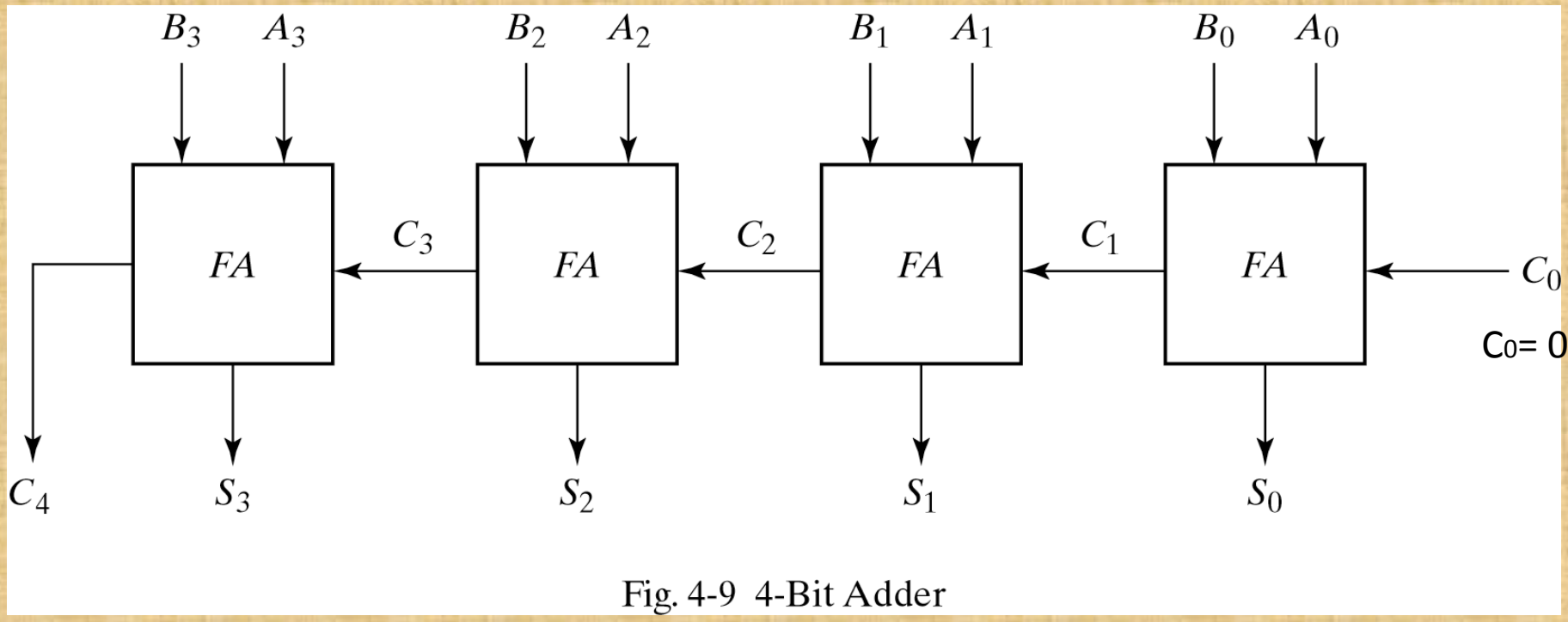


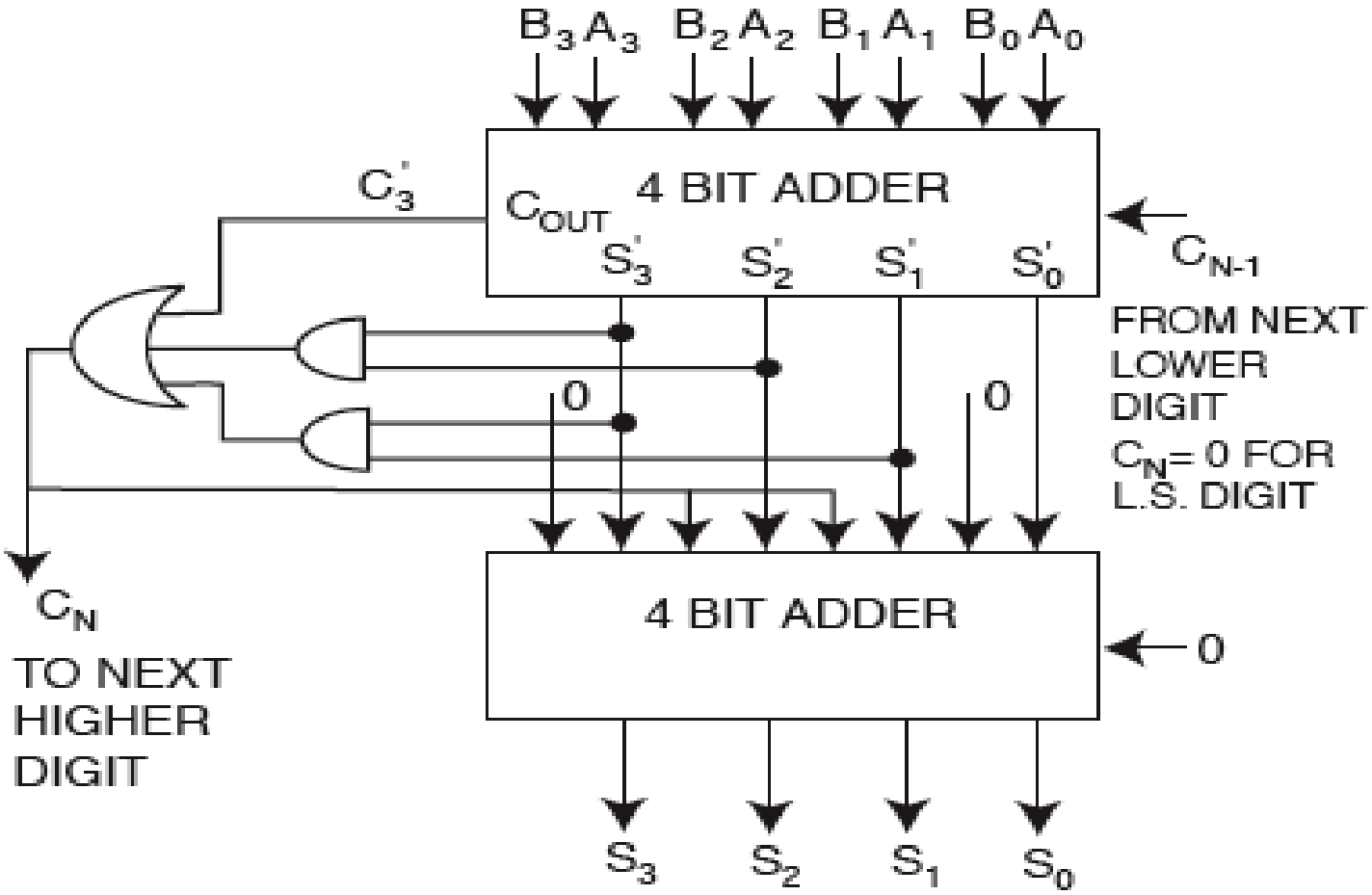
Fig. 4-9 4-Bit Adder

Binary Adder

- For example to add A= 1011 and B= 0011

subscript i:	3	2	1	0	
Input carry:	0	1	1	0	C_i
Augend:	1	0	1	1	A_i
Addend:	0	0	1	1	B_i
	---	---	---	---	
Sum:	1	1	1	0	S_i
Output carry:	0	0	1	1	C_{i+1}

DECIMAL/BCD ADDER



- ADD **0110** WHEN C_N=1
- ADD **0000** WHEN C_N=0

Assignment

- Explain half Adder and full Adder? Expalin Full Adder using Half adders?

Subtractors

Combinational Arithmetic Circuits

- Addition:
  - **Half Adder (HA).**
  - **Full Adder (FA).**
  - **Binary Adder**
  - **BCD(Decimal) Adder.**
- Subtraction:
  - **Half Subtractor.**
  - **Full Subtractor.**
- Multiplication:
  - **Binary Multipliers.**
- Comparator:
  - **Magnitude Comparator.**

Combinational Arithmetic Circuits

- Multiplexers
- Demultiplexers
- Encoders
- Decoders

Half Subtractor

- Subtracting a single-bit binary value Y from anther X (I.e. X -Y) produces a difference bit D and a borrow out bit B-out.
- This operation is called half subtraction and the circuit to realize it is called a half subtractor.

$D(X,Y) = \Sigma(1,2)$

$D = X'Y + XY'$

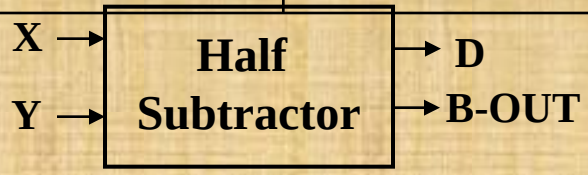
$D = X \oplus Y$

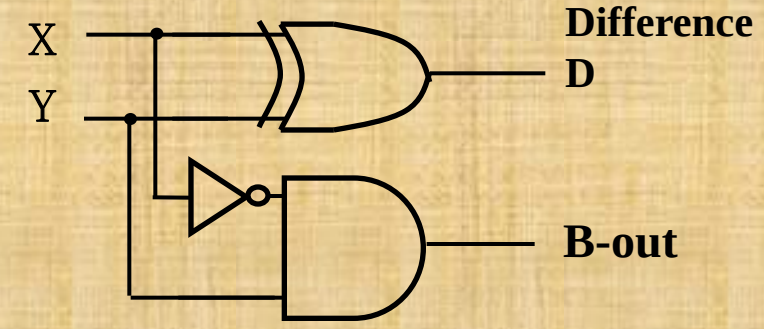
$B\text{-out}(x, y, C\text{-in}) = \Sigma(1)$

$B\text{-out} = X'Y$

Half Subtractor Truth Table

Inputs		Outputs	
X	Y	D	B-out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0





Binary Arithmetic Operations Subtraction

- Two binary numbers are subtracted by subtracting each pair of bits together with borrowing, where needed.
- Subtraction Example:

		0 0 1 1 1 1 1 0 0	Borrow
X	229	1 1 1 0 0 1 0 1	
Y -	46	- 0 0 1 0 1 1 1 0	
	183	1 0 1 1 0 1 1 1	

Full Subtractor

- Subtracting two single-bit binary values, Y, B-in from a single-bit value X produces a difference bit D and a borrow out B-out bit. This is called full subtraction.

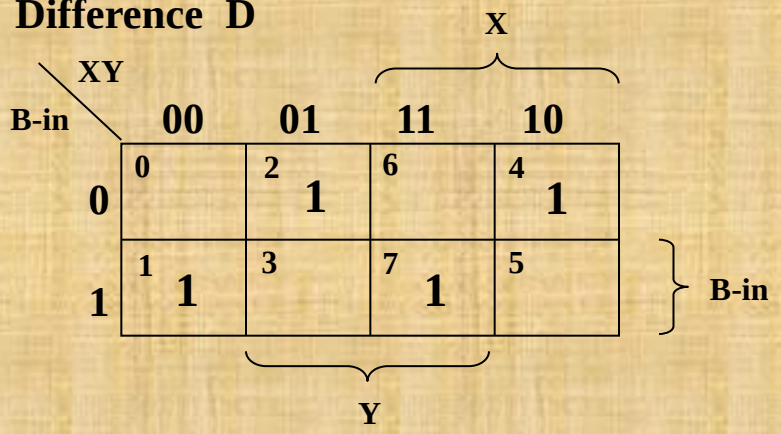
Full Subtractor Truth Table

Inputs			Outputs	
X	Y	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

$S(X,Y, C\text{-in}) = \Sigma\ (1,2,4,7)$

$C\text{-out}(x, y, C\text{-in}) = \Sigma\ (1,2,3,7)$

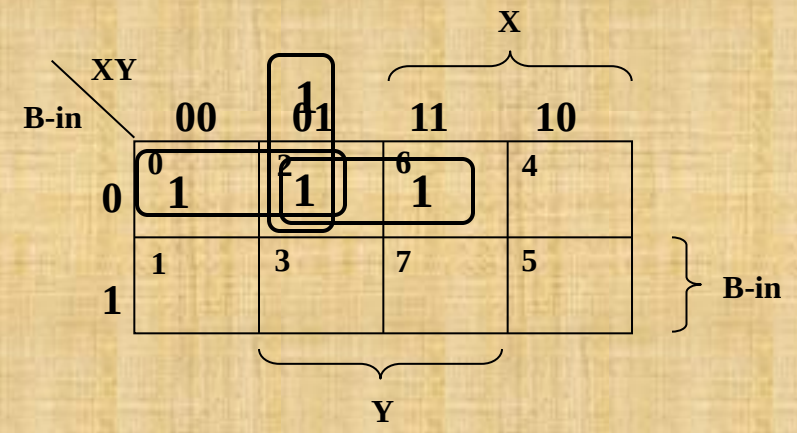
Difference D



$S = X'Y'(B\text{-in}) + XY'(B\text{-in})' + XY'(B\text{-in})' + XY(B\text{-in})$

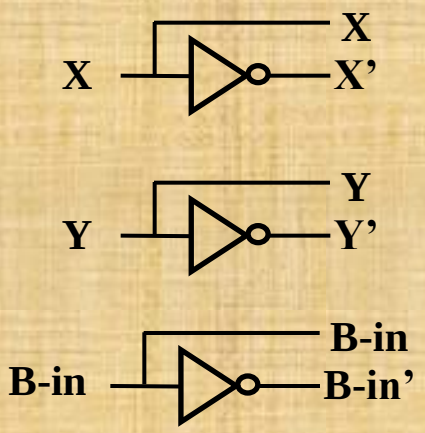
$S = X \oplus Y \oplus (C\text{-in})$

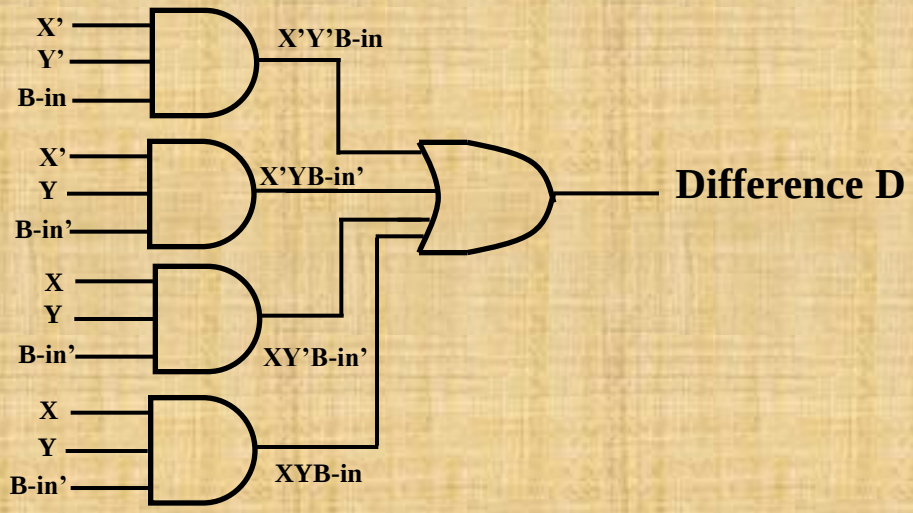
Borrow B-out

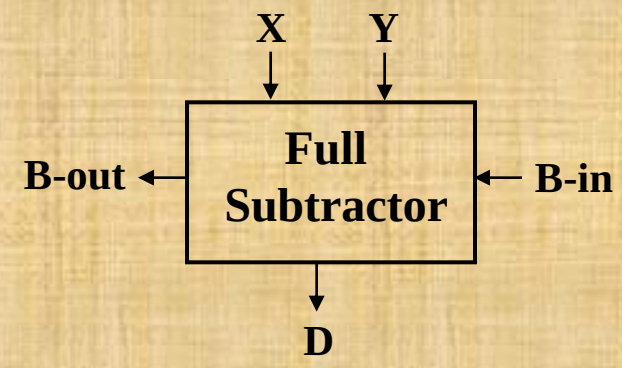


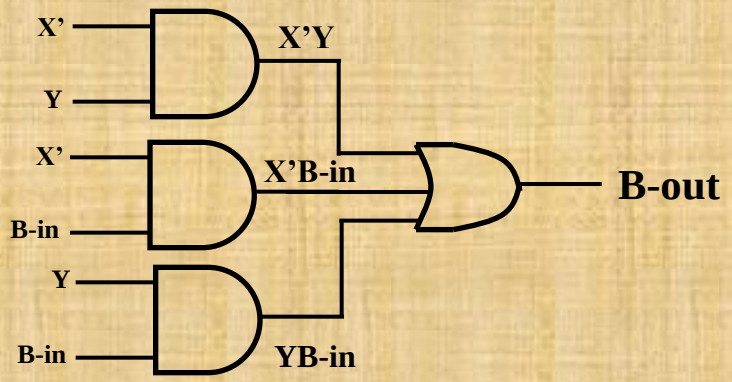
$B\text{-out} = X'Y + X'(B\text{-in}) + Y(B\text{-in})$

Full Subtractor Circuit Using AND-OR

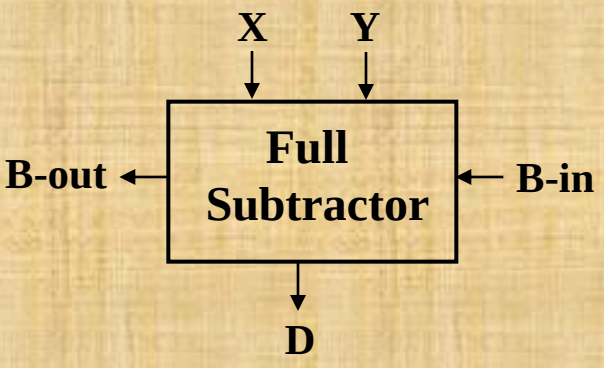


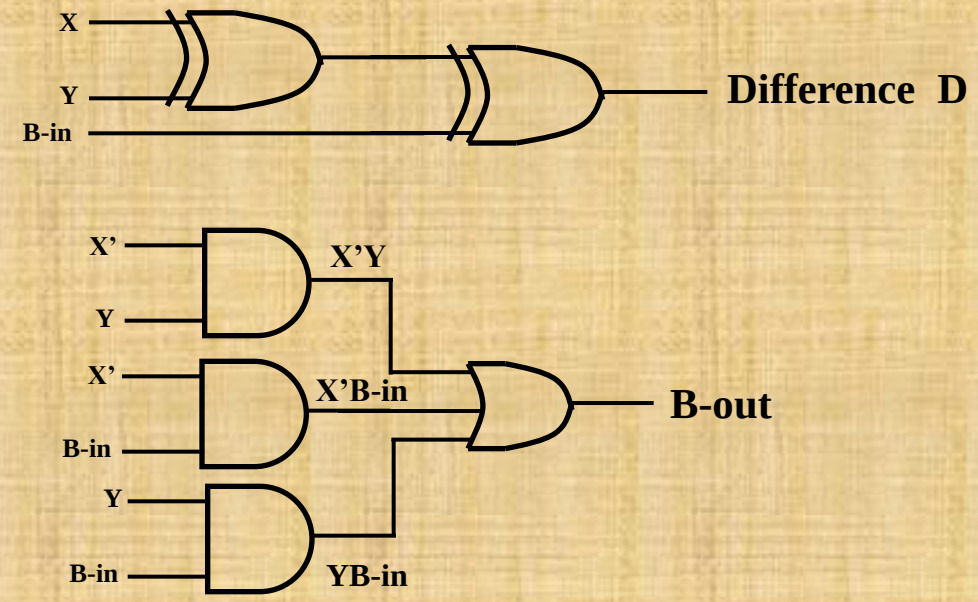






Full Subtractor Circuit Using XOR





n-bit Subtractors

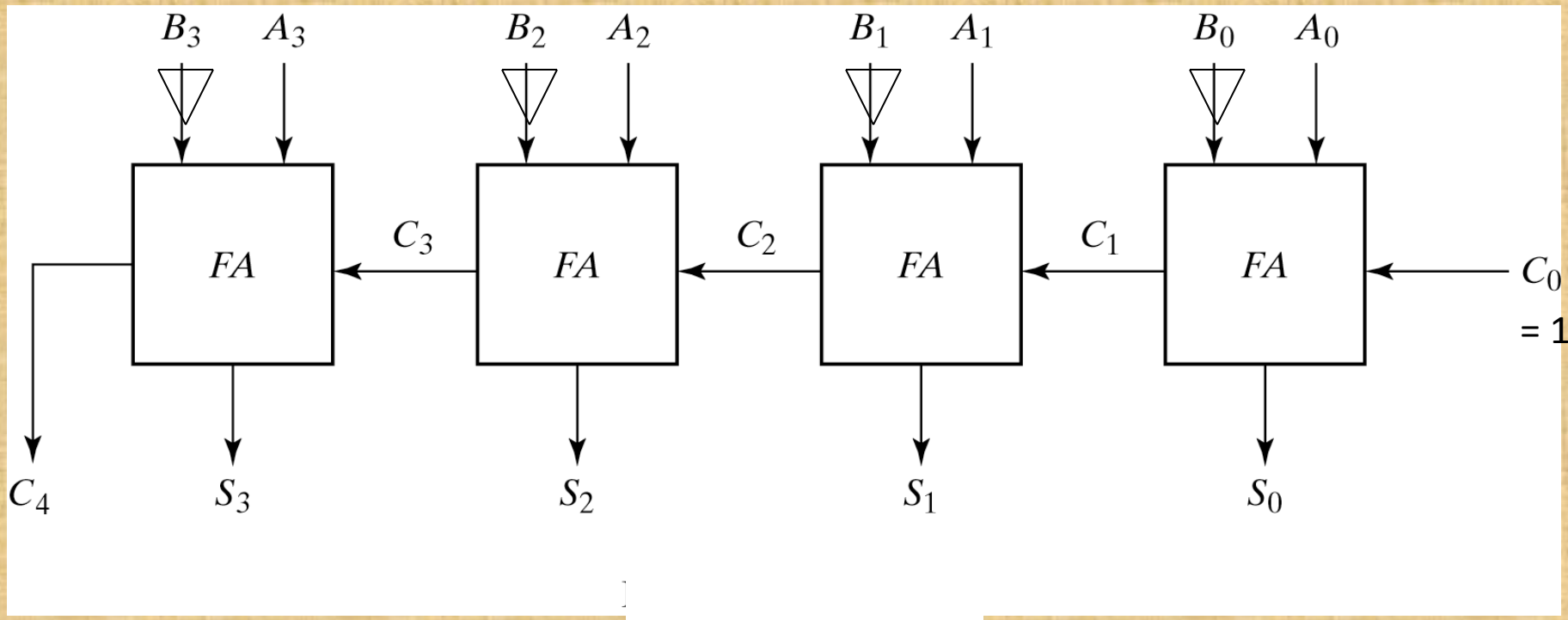
An n-bit subtracor used to subtract an n-bit number Y from another n-bit number X (i.e X-Y) can be built in one of two ways:

- By using n full subtractors and connecting them in series, creating a borrow ripple subtractor:
  - **Each borrow out B-out from a full subtractor at position j is connected to the borrow in B-in of the full subtracor at the higher position j+1.**
- By using an n-bit adder and n inverters:
  - **Find two's complement of Y by:**
    - Inverting all the bits of Y using the n inverters.
    - Adding 1 by setting the carry in of the least significant position to 1
  - **The original subtraction (X - Y) now becomes an addition of X to two's complement of Y using the n-bit adder.**

Binary Subtractor

- The subtrcation A – B can be done by taking the 2's complement of B and adding it to A because A- B = A + (-B)
- It means if we use the inveters to make 1's complement of B (connecting each Bi to an inverter) and then add 1 to the least significant bit (by setting carry C0 to 1) of binary adder, then we can make a binary subtractor.

4 bit 2's complement Subtractor



Adder Subtractor

- The addition and subtraction can be combined into one circuit with one common binary adder (see next slide).
- The mode M controls the operation. When M=0 the circuit is an adder when M=1 the circuit is subtractor. It can be don by using exclusive-OR for each Bi and M. Note that $1 \oplus x = x'$ and $0 \oplus x = x$

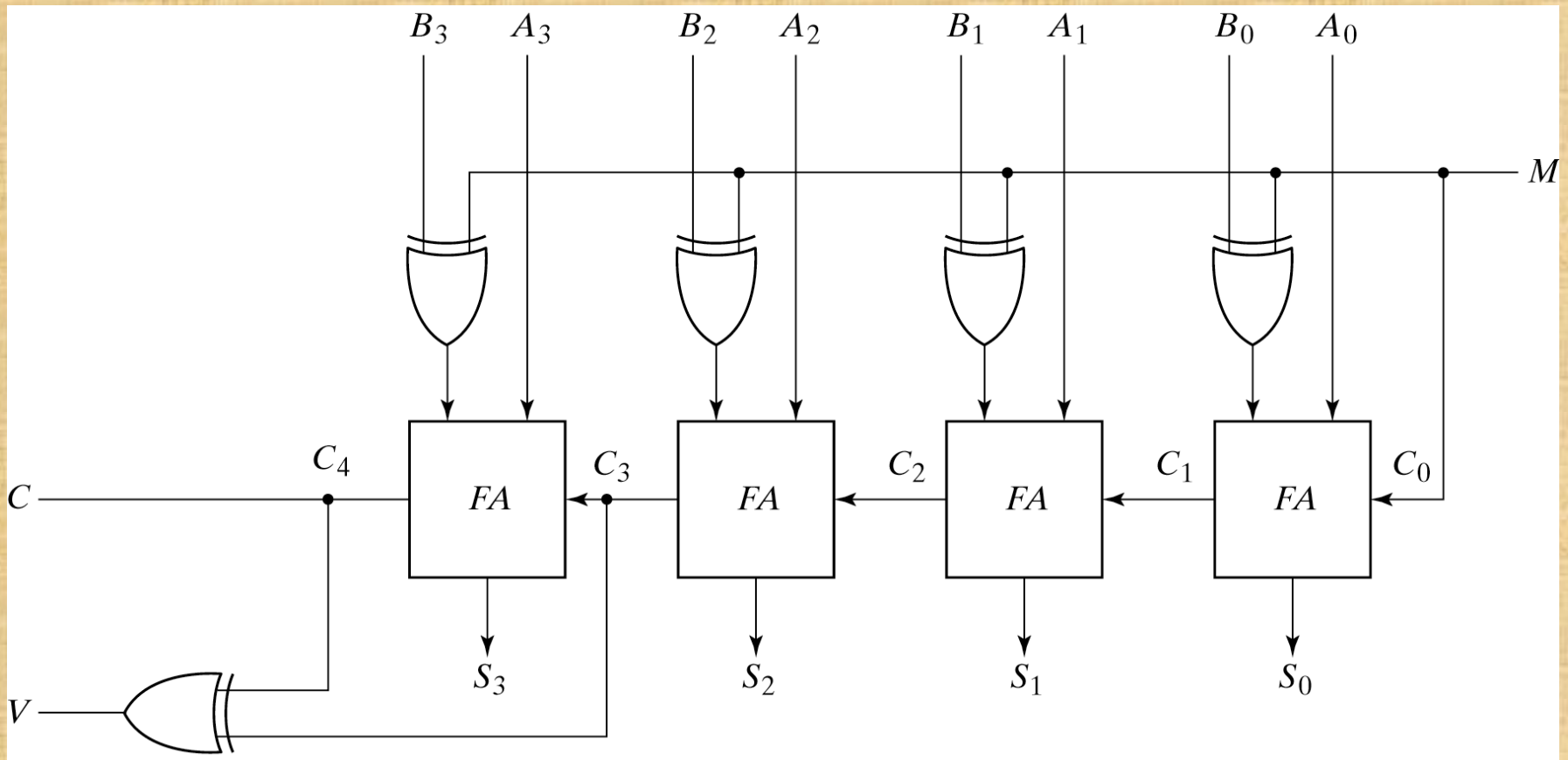
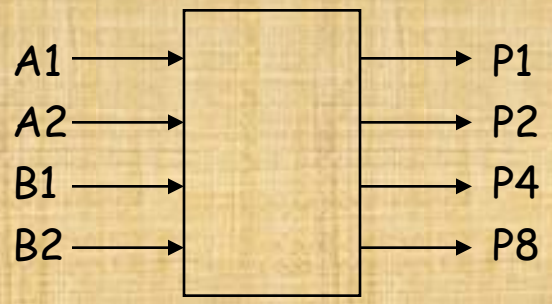


Fig. 4-13 4-Bit Adder Subtractor

Checking Overflow

- Note that in the previous slide if the numbers considered to be signed V detects overflow. V=0 means no overflow and V=1 means the result is wrong because of overflow
- Overflow can be happened when adding two numbers of the same sign (both negative or positive) and result can not be shown with the available bits. It can be detected by observing the carry into sign bit and carry out of sign bit position. If these two carries are not equal an overflow occurred. That is why these two carries are applied to exclusive-OR gate to generate V.

Design example: 2x2-bit multiplier

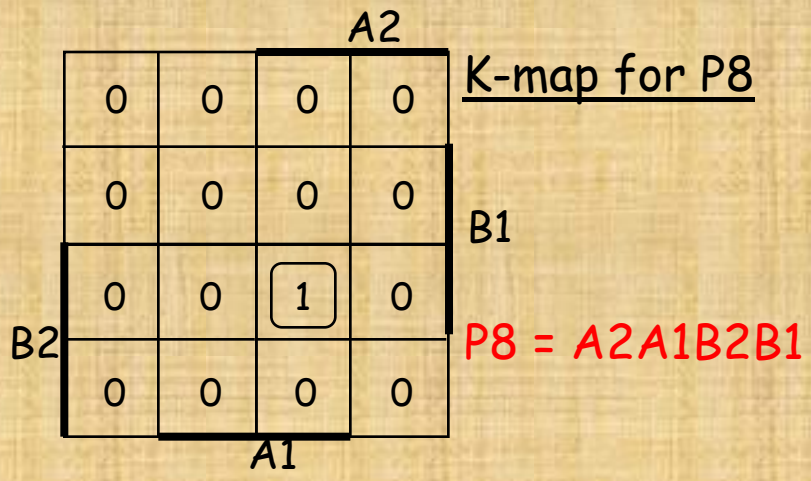


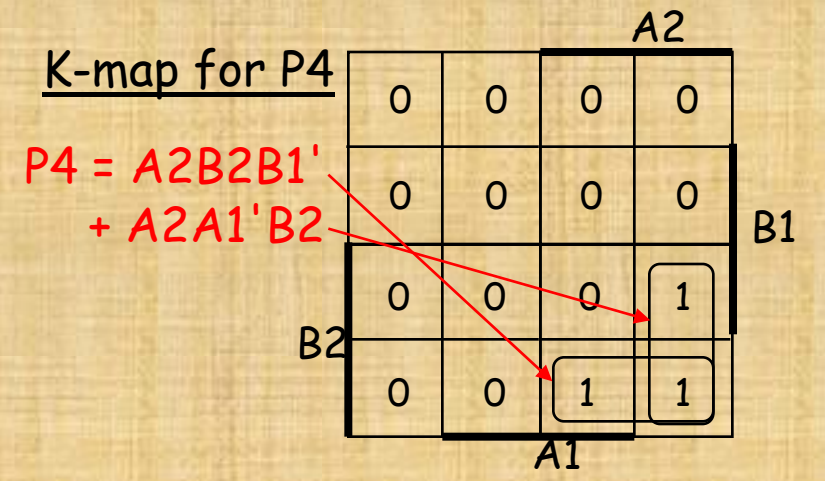
block diagram
and
truth table

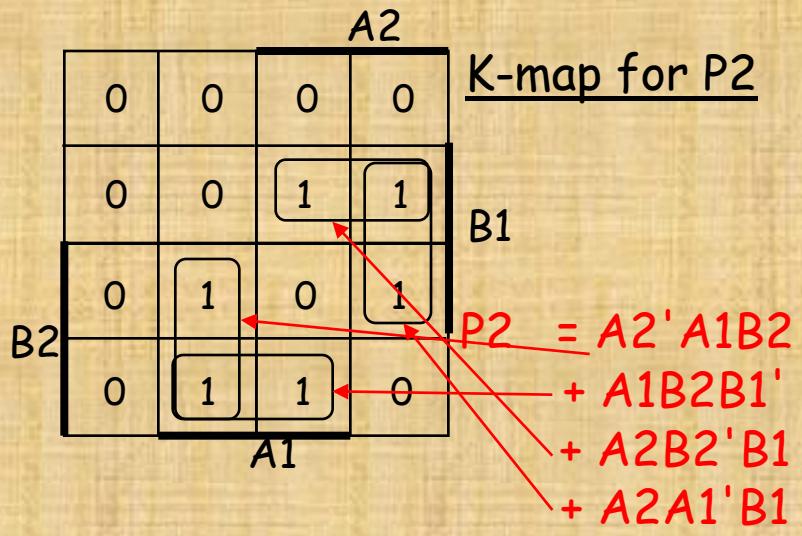
A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

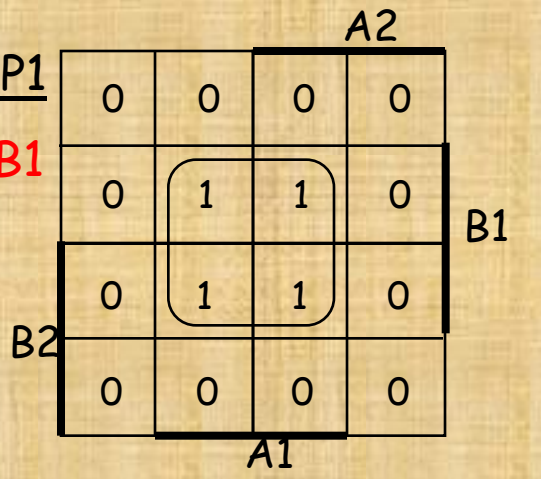
4-variable K-map
for each of the 4
output functions

Design example: 2x2-bit multiplier (cont'd)









Assignment-

- Explain half Subtractor and Full Subtractor.

Magnitude Comparator

Magnitude Comparator

- It is a combinational circuit that compares two numbers and determines their relative magnitude
- The output of comparator is usually 3 binary variables indicating: A>B

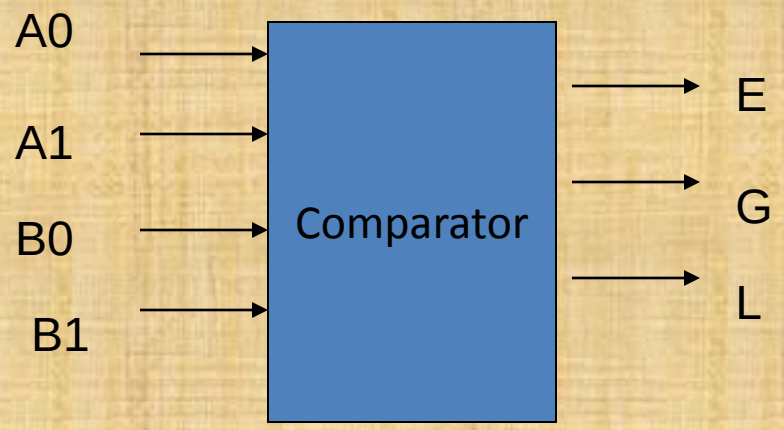
  A=B

  A<B
- For example to design a comparator for 2 bit binary numbers A (A1A0) and B (B1B0) we do the following steps:

Comparators

- **For a 2-bit comparator we have four inputs A1A0 and B1B0 and three output E (is 1 if two numbers are equal), G (is 1 when A > B) and L (is 1 when A < B) If we use truth table and KMAP the result is**
- **E= A'1A'0B'1B'0 + A'1A0B'1B0 + A1A0B1B0 + A1A'0B1B'0**

  or E=((A0 ⊕ B0) + (A1 ⊕ B1))' (see next slide)
- **G = A1B'1 + A0B'1B'0 + A1A0B'0**
- **L= A'1B1 + A'1A'0B0 + A'0B1B0**



Truth Table

A0	A1	B0	B1	E	L	G
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1

Magnitude Comparator

- From the truth table:

E = (0,5,10,15)

= A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0’B1B0’ +A1A0B1B0

Magnitude Comparator

- A>B means

A1	B1	Y1
0	0	0
0	1	0
1	0	1
1	1	0

if A1=B1 (X1=1) then A0 should be 1 and B0 should be 0

A0	B0	Y0
0	0	1
0	1	0
1	0	0
1	1	0

For A> B: A1 > B1 or A1 =B1 and A0 > B0

It means Y= A1B’1 + X1A0B’0 should be 1 for A>B

Magnitude Comparator

- For B>A B1 > A1

  or

  A1=B1 and B0> A0

  z= A’1B1 + X1A’0B0

- The procedure for binary numbers with more than 2 bits can also be found in the similar way. For example next slide shows the 4-bit magnitude comparator, in which

(A= B) = x3x2x1x0

(A> B) = A3B’3 + x3A2B’2 + x3x2A1B’1+ x3x2x1A0B’0

(A< B) = A’3B3 + x3A’2B2 + x3x2A’1B1+ x3x2x1A’0B0

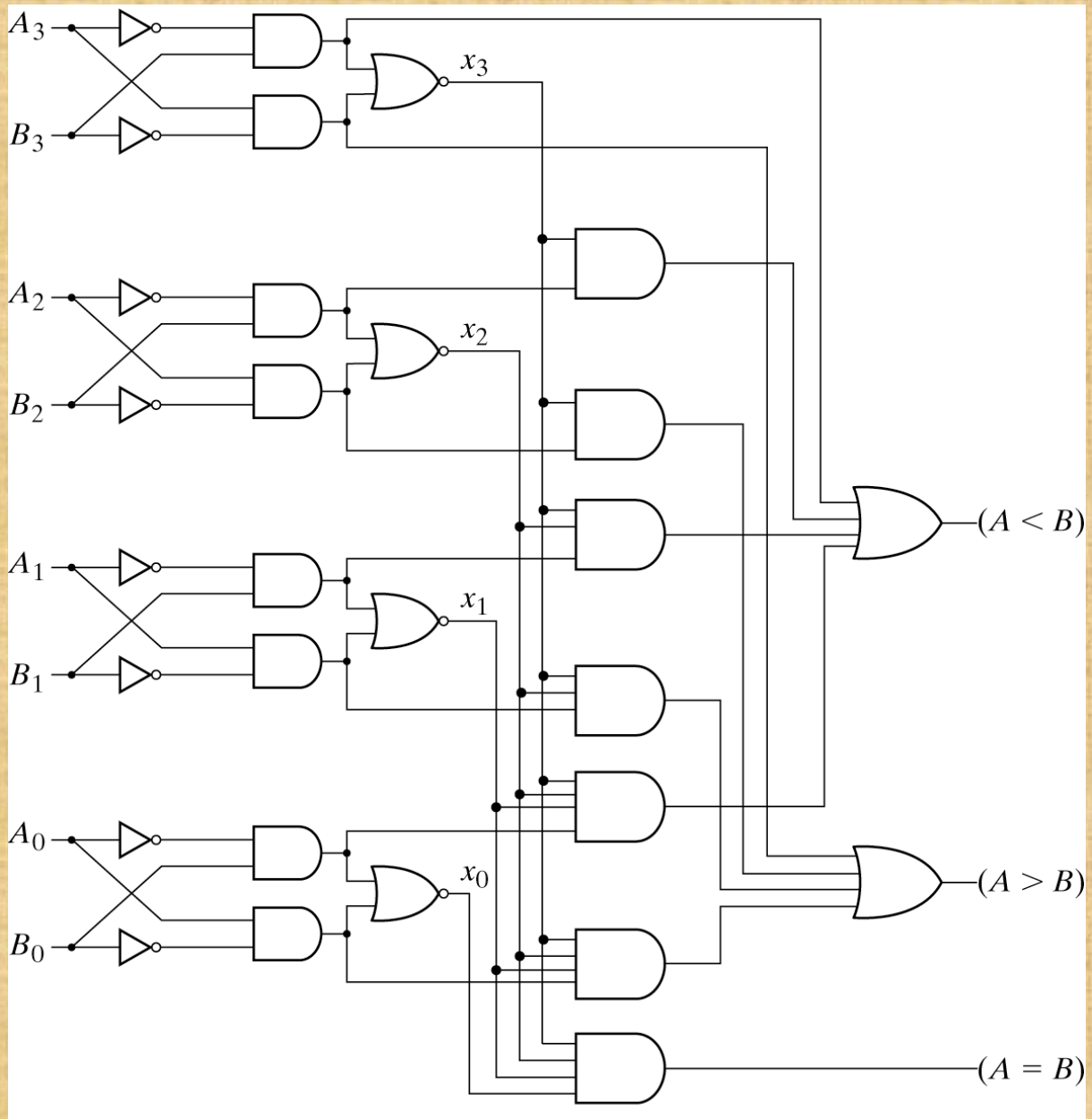
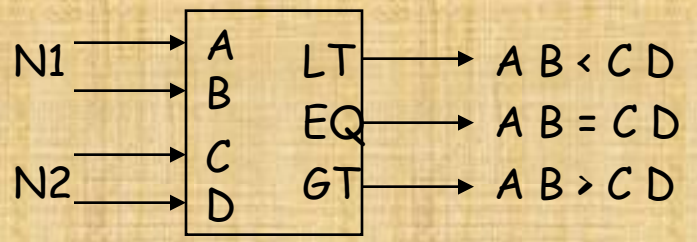


Fig. 4-17 4-Bit Magnitude Comparator

Design example: two-bit comparator

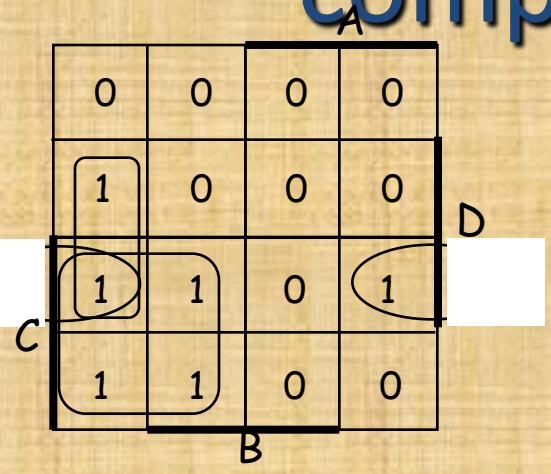


block diagram and truth table

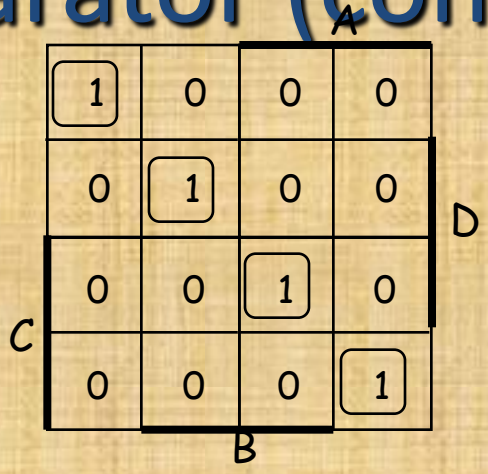
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

we'll need a 4-variable Karnaugh map for each of the 3 output functions

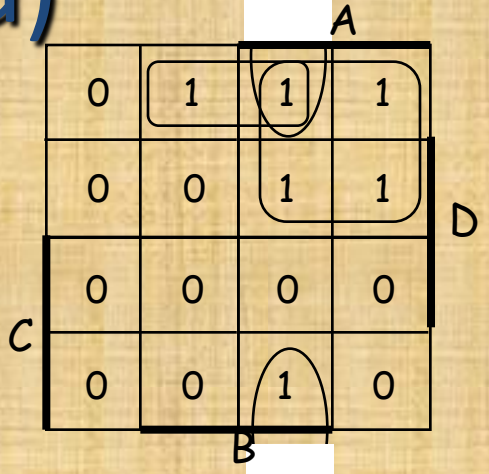
Design example: two-bit comparator (cont'd)



K-map for LT



K-map for EQ



K-map for GT

LT = A' B' D + A' C + B' C D

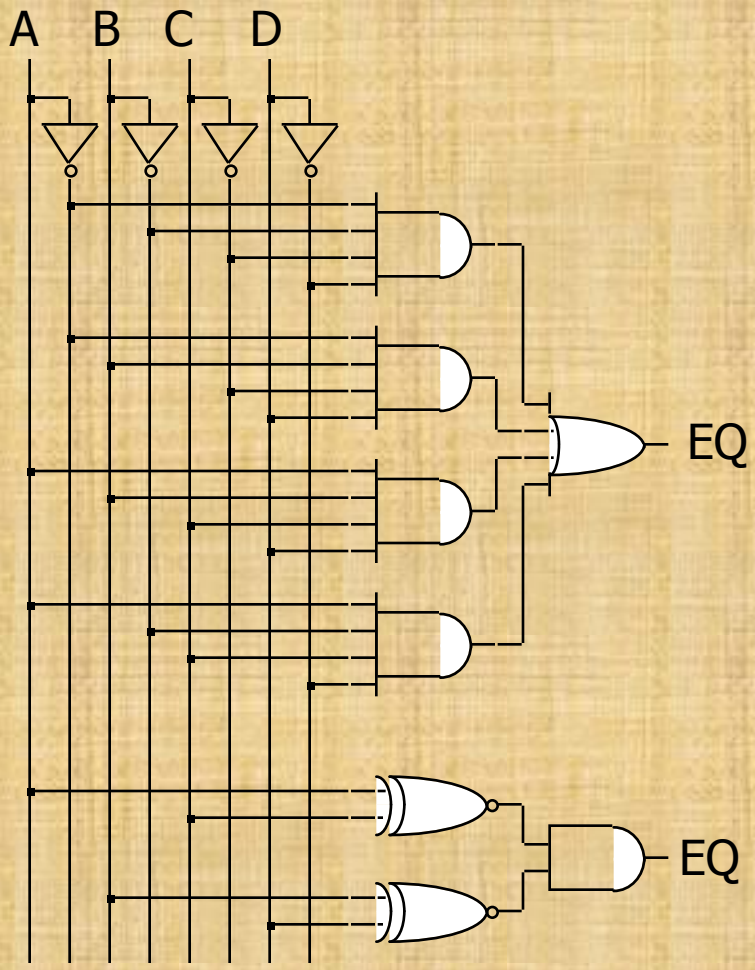
EQ = A'B'C'D' + A'BC'D + ABCD + AB'CD' = (A xnor C) • (B xnor D)

GT = B C' D' + A C' + A B D'

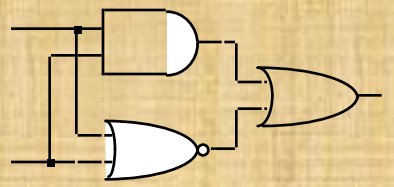
Canonical PofS vs minimal?

LT and GT are similar (flip A/C and B/D)

Design example: two-bit comparator (cont'd)



two alternative implementations of EQ with and without XOR



XNOR is implemented with at least 3 simple gates

Assignment

Explain 2-bit comparator.

Code Converter

Binary to Gray Code Converter

The Gray Code is unweighted and is not an arithmetic code: that is, there are no specific weights asssigned to the bit positions. The important feature of the Gray Code is that it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence.

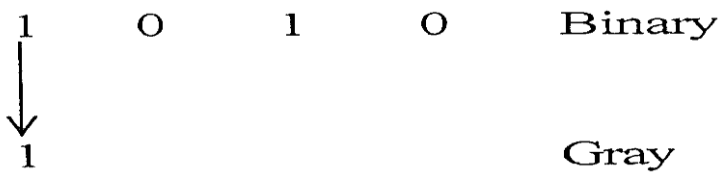
To convert a binary number to a Gray Code number, the following rules apply.

1. The most significant digit (Left Most Bit) in the Gray Code is the same as the corresponding digit in the binary number.

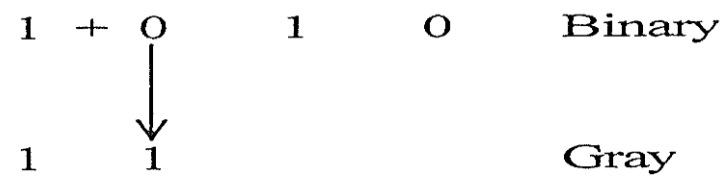
2. Going from left to right, add each adjacent pair of binary digits to get the next Gray code digit, regradless carries.

For instance - Let us convert the binary number 1010 to Gray Code.

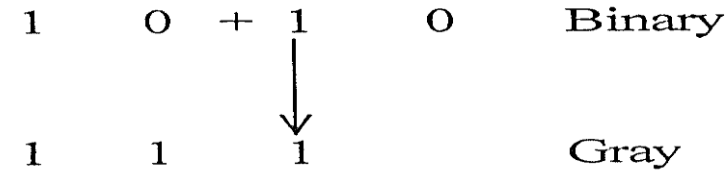
Step 1 - The left most Gray digit is the same as the left most binary



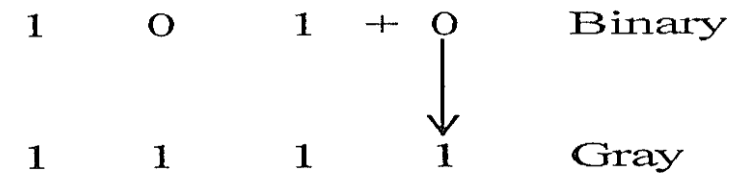
Step 2 - Add the left most binary digit to the adjacent one.



Step 3 - Add the next adjacent pair



Step 4 -Add the last adjacent pair



The conversion is now complete and the Gray Code is 1111.

Steps to design the converter

1. Design a converter by the following procedures:

a. Write down the truth table of both input and output bits of the converter.

b. Apply Karnaugh Map to look for the minimized logic expression for the output bits.

c. Implement the logic gates by using Circuit Maker.

Example:

For Binary to Gray Code Converter, binary bits are input and gray code bits are output. So first write the truth table for binary bits and gray code. Then k-map for the all bits of gray code, find the simplified expression for each bit of gray code. Then design the logical circuit.

Truth Table

Decimal	Binary				Gray			
	A	B	C	D	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

K-Map for each bit of Gray code

For practical consideration code conversions are made for each bit.

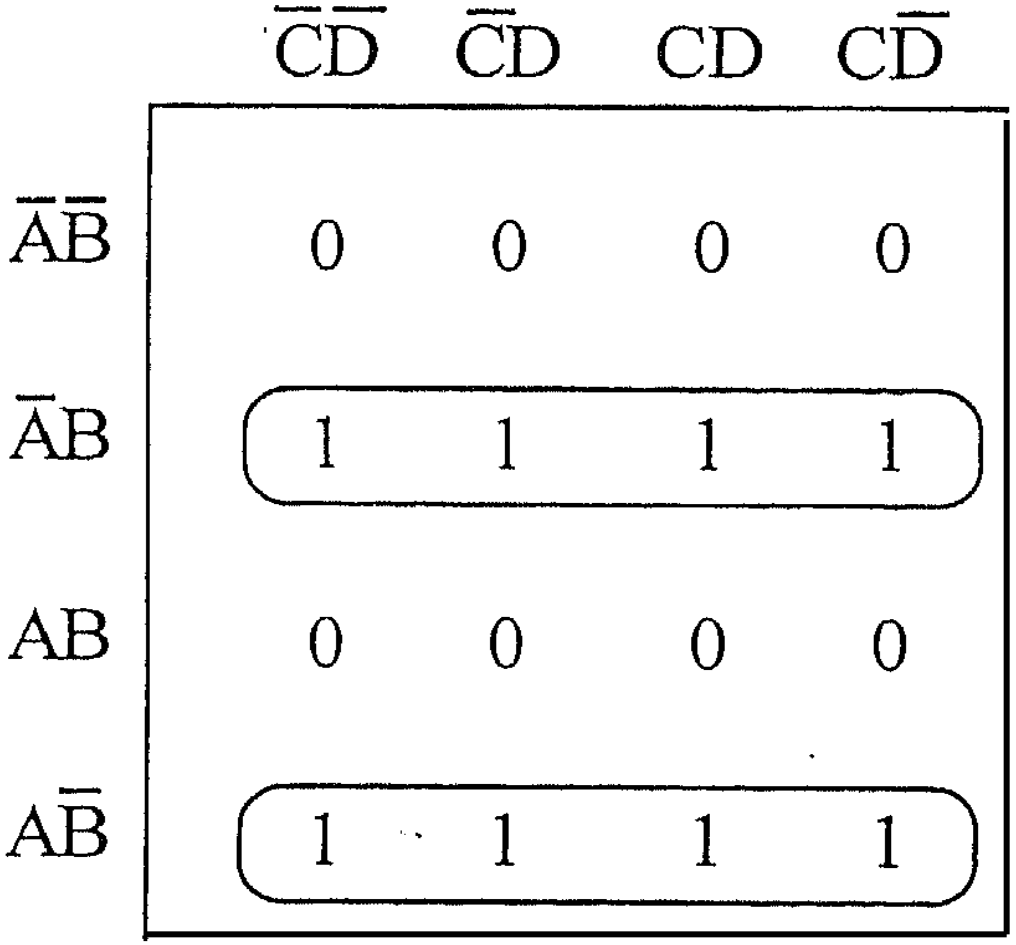
(a) For Y_3, the Karnaugh map can draw as follow.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	1	1	1	1
$A\bar{B}$	1	1	1	1

(a)

By minimization, $Y_3 = A$

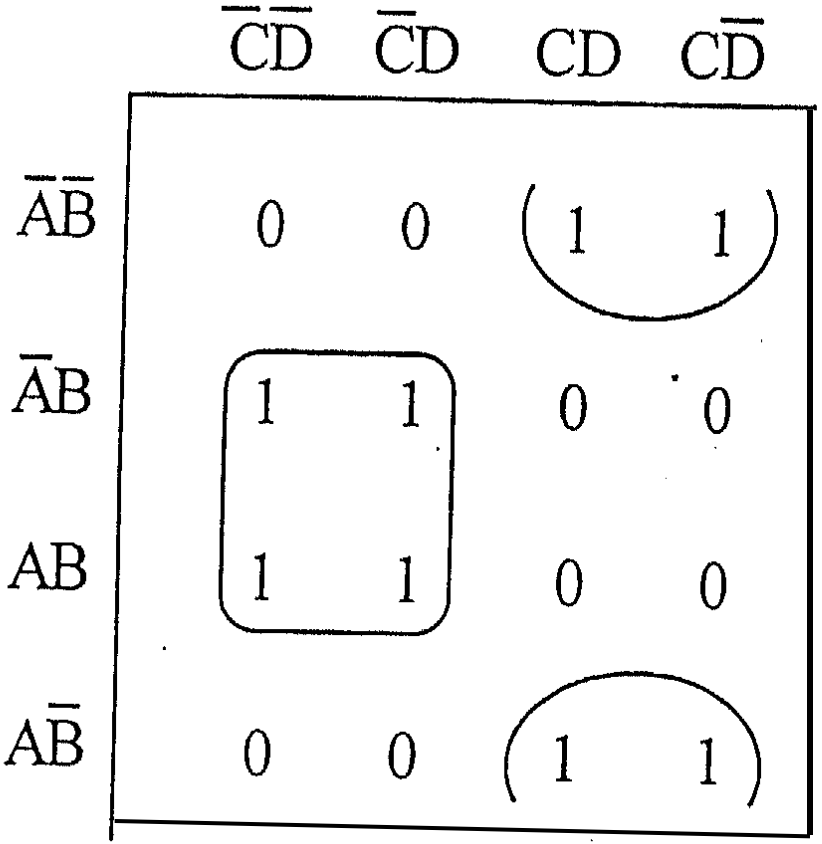
(b) For Y_2, the Karnaugh map can draw as follow.



(b)

By minimization, $Y_2 = A \oplus B$

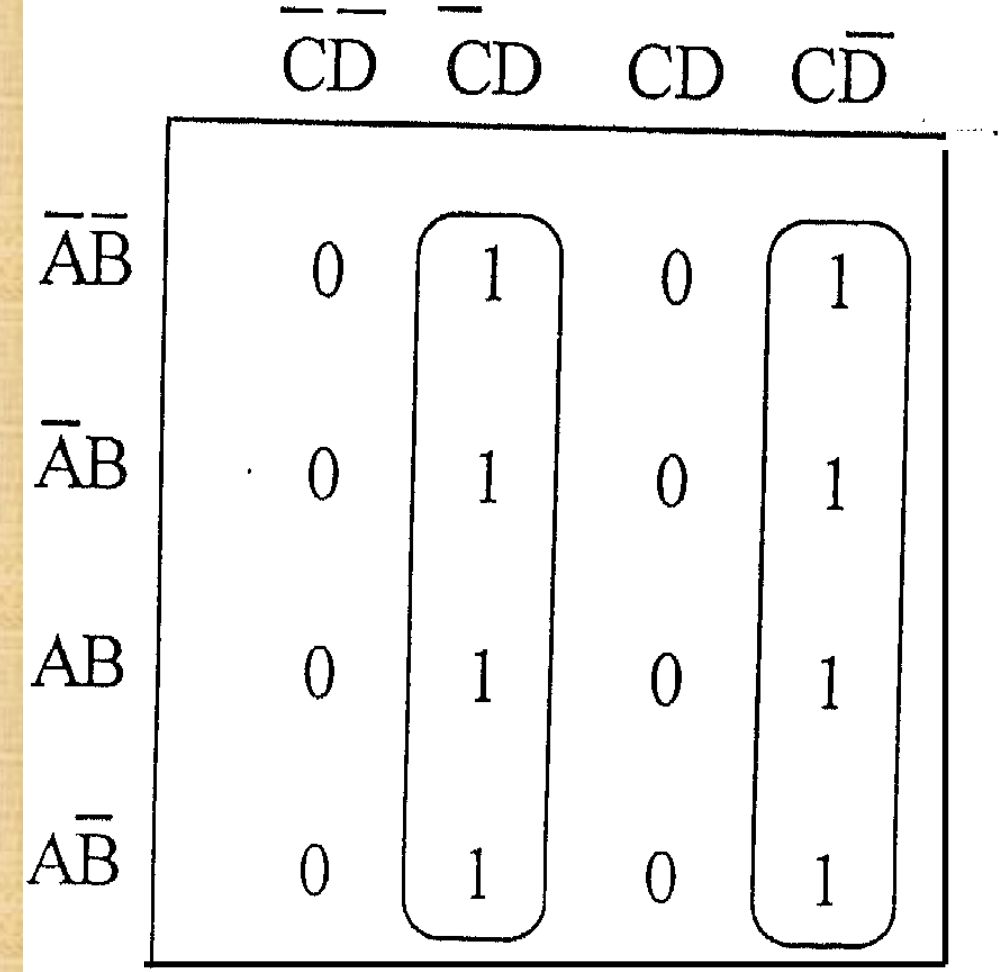
(c) For Y_1, the Karnaugh map can draw as follow.



(a)

By minimization, $Y_1 = B \oplus C$

(d) For Y_0, the Karnaugh map can draw as follow.



(b)

By minimization, $Y_0 = C \oplus D$

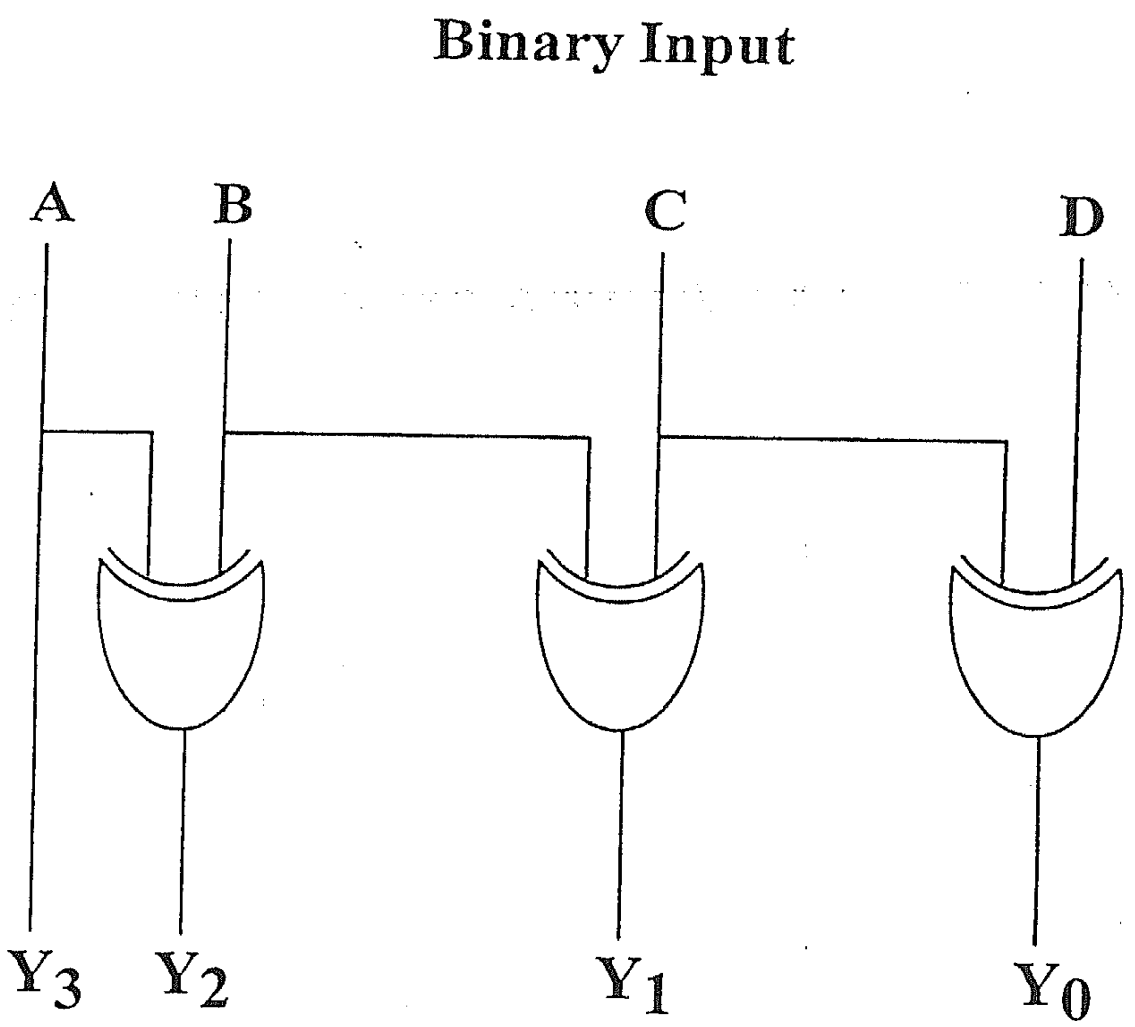


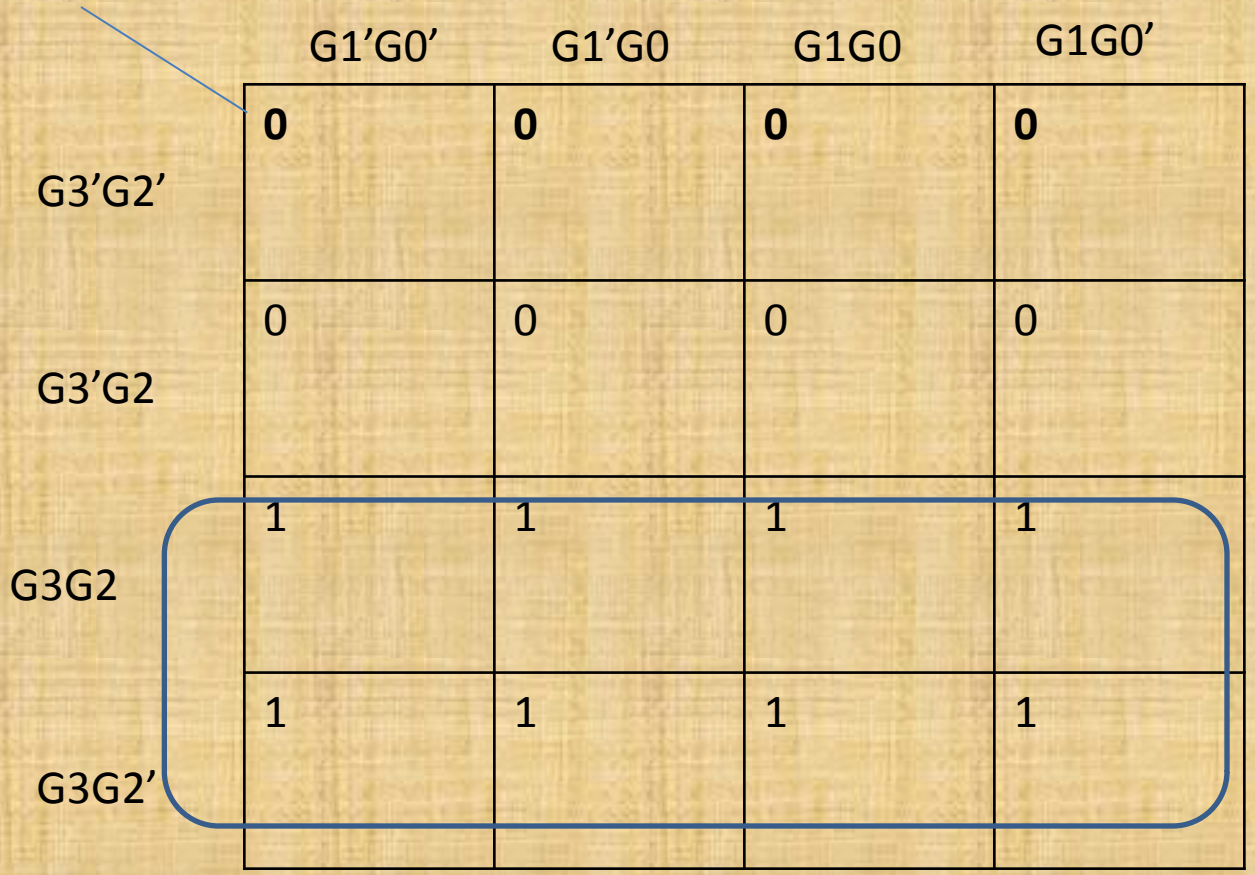
Figure 3.1 Binary-to-Gray Code

Gray to Binary Converter

Truth Table

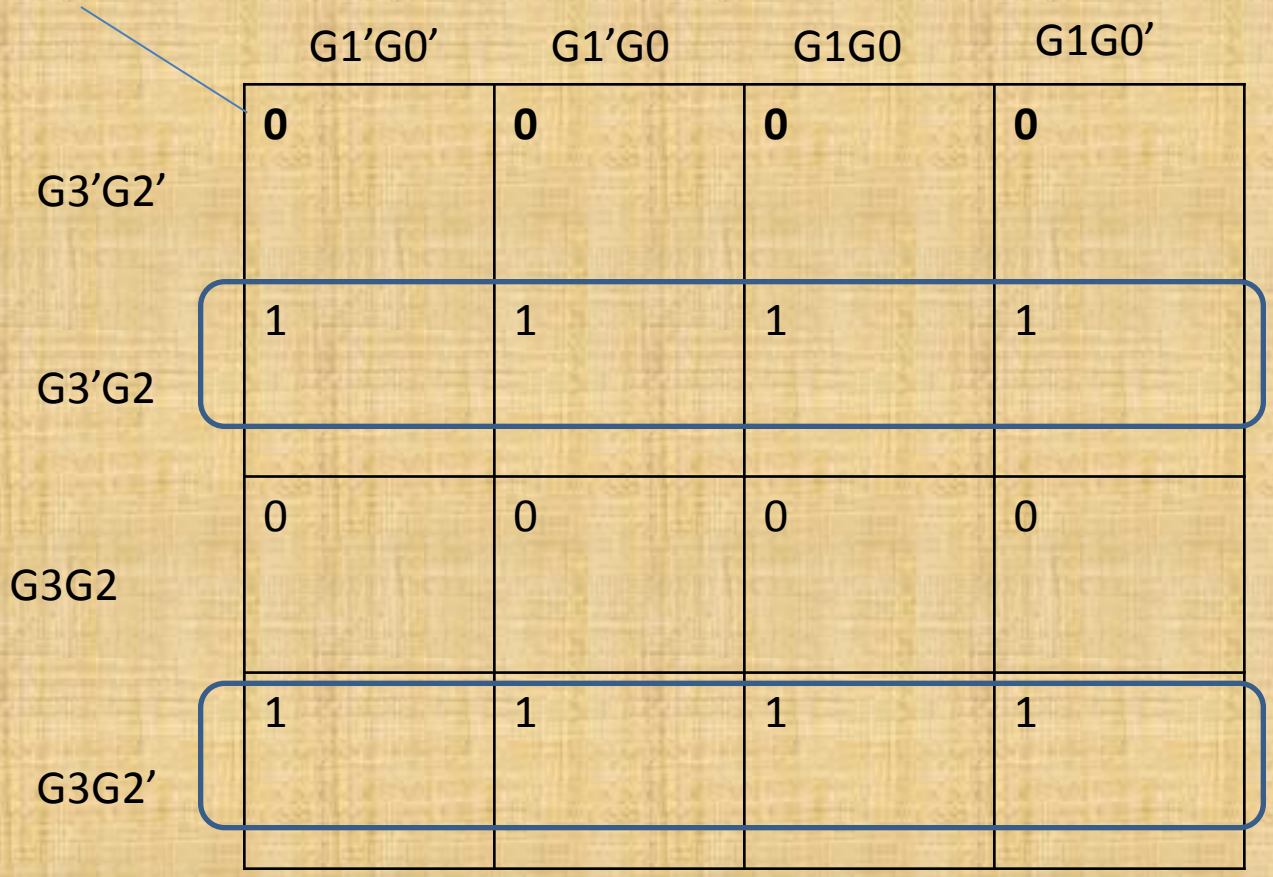
Dec	G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
3	0	0	1	1	0	0	1	0
2	0	0	1	0	0	0	1	1
6	0	1	1	0	0	1	0	0
7	0	1	1	1	0	1	0	1
5	0	1	0	1	0	1	1	0
4	0	1	0	0	0	1	1	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	0	0	1
15	1	1	1	1	1	0	1	0
14	1	1	1	0	1	0	1	1
10	1	0	1	0	1	1	0	0
11	1	0	1	1	1	1	0	1
9	1	0	0	1	1	1	1	0

K MAP For B3



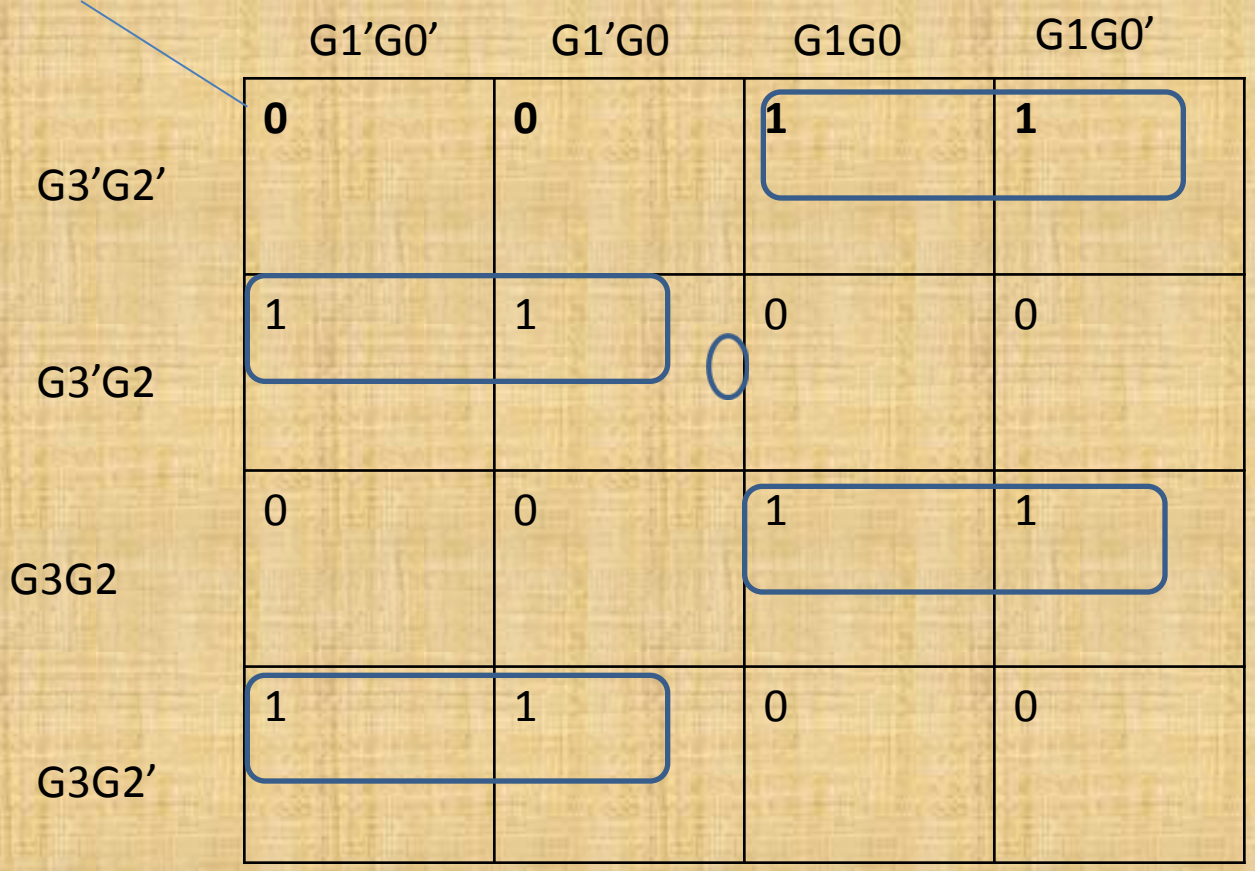
B3 = G3

K MAP For B2



$$B1 = G3 \oplus G2 \oplus G1$$

K MAP For B1

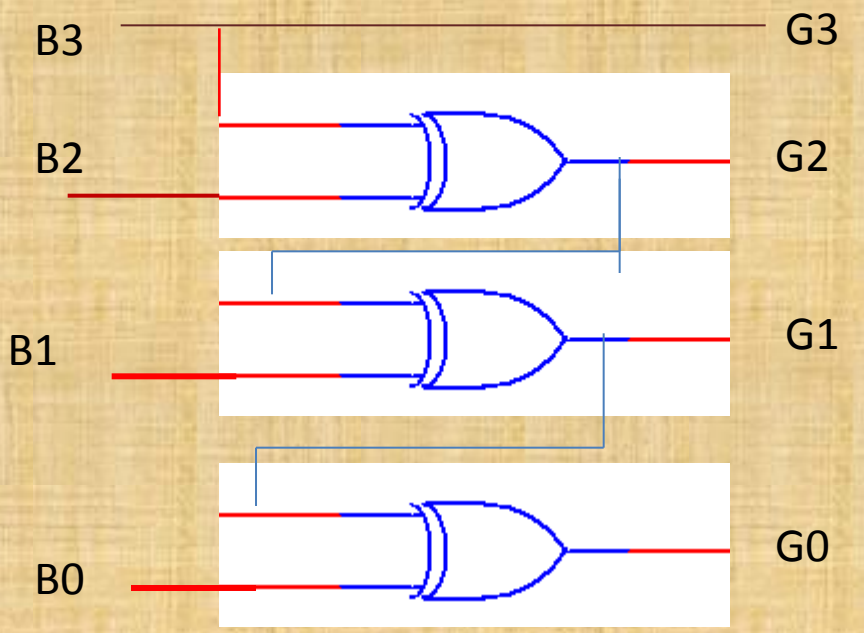


$$B1 = G3 \oplus G2 \oplus G1$$

K MAP For B0

	G1'G0'	G1'G0	G1G0	G1G0'
G3'G2'	**0**	**1**	**0**	**1**
G3'G2	1	0	1	0
G3G2	0	1	0	1
G3G2'	1	0	1	0

$$B0 = G3 \oplus G2 \oplus G1 \oplus G0$$


B3
G3
B2
G2
B1
G1
B0
G0

Assignment

Design the Converter for

1. Binary to BCD
2. BCD to Gray
3. BCD to Binary
4. BCD to Excess

Analysis of Combinational Circuits

Designing Combinational Logic Circuits

•A logic circuit having 3 inputs, A, B, C will have its output HIGH only when a majority of the inputs are HIGH.

Step 1 Set up the truth table

Step 2
Write the AND term for each case where the output is a 1.

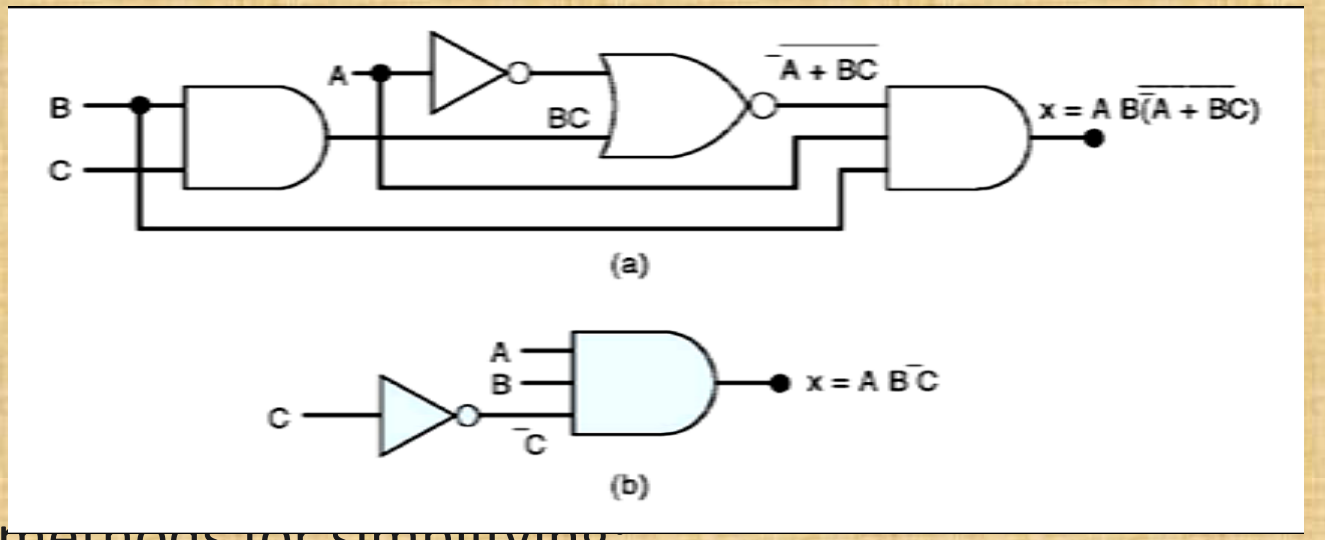
A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \bar{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\bar{B}C$
1	1	0	1	$\rightarrow AB\bar{C}$
1	1	1	1	$\rightarrow ABC$

Sum-Of-Products Form

- SOP is useful in simplification and design
- Two or more AND terms OR together
  - Ex: $ABC+\overline{A}B\overline{C}$
  - the inversion sign cannot cover more than one variable (ABC)
- Another general form for logic expressions is sometimes used in logic-circuit design. It called product-of-sum (POS)
- Consist 2 or more OR terms that are AND together.
  - Ex: $(A+\overline{B}+C)(A+C)$

Analysis of Logic Circuits

- First obtain one expression for the circuit, then try to simplify.
- Example:

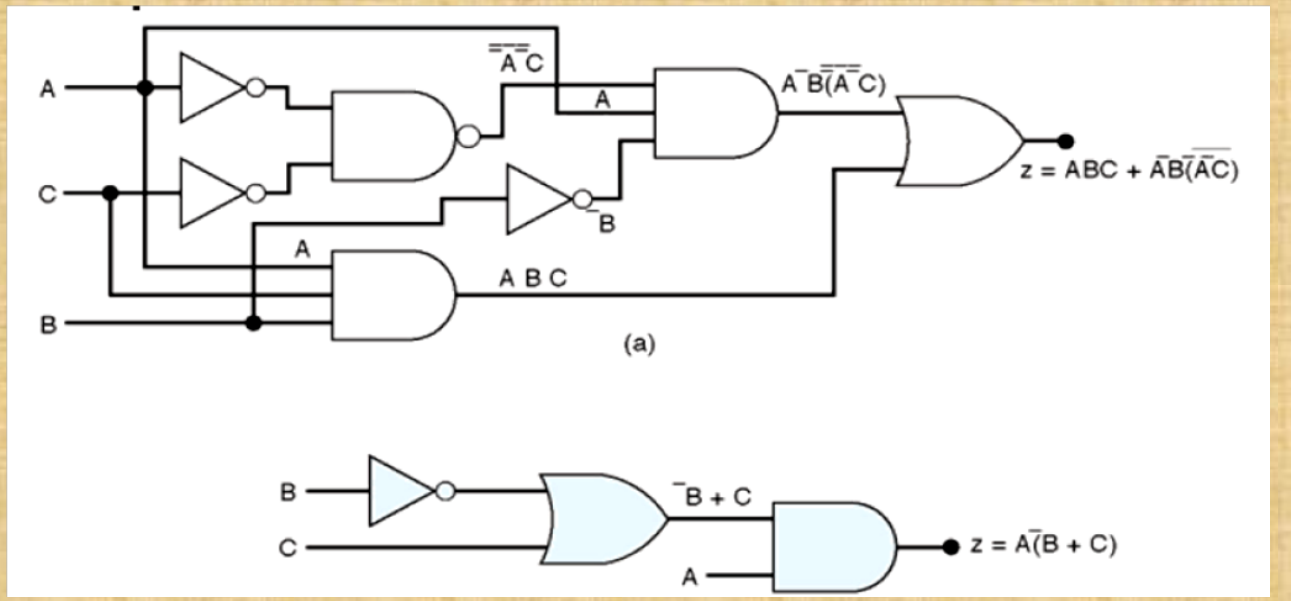


- Two methods for simplifying:
  - Algebraic method (use Boolean algebra theorems)
  - Karnaugh mapping method (systematic, step-by-step approach)

Algebraic Simplification

1. Put the original expression into SOP form by repeated application of *DeMorgan's theorems*
2. Once in SOP form, check for *common factors* and factor whenever possible.

Example:



Step 3 Write the SOP form the output

Step 4 Simplify the output expression

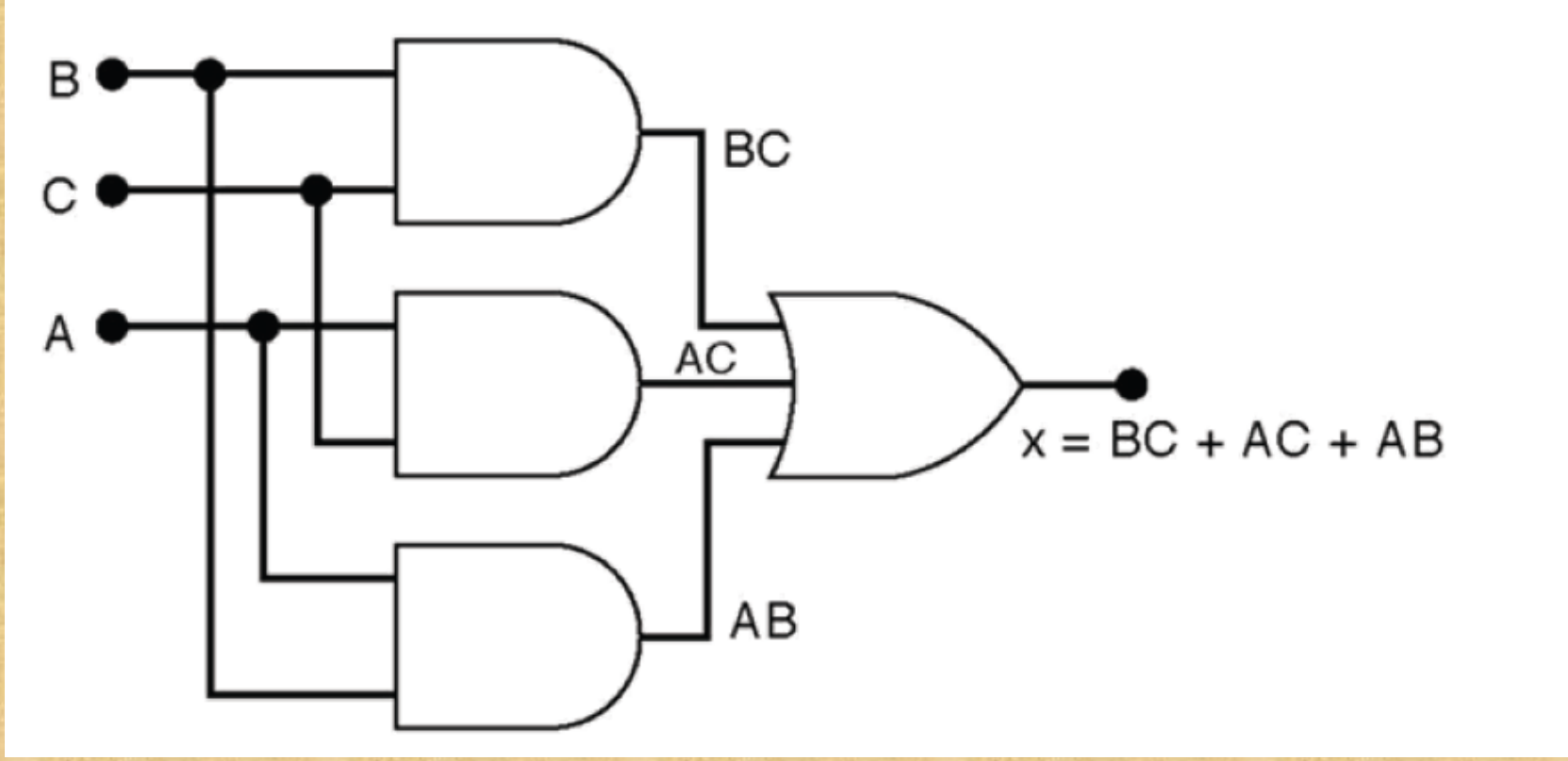
$$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

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

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

$$= BC + AC + AB$$

Step 5 Implement the circuit



Karnaugh Map (K-Map) Method

- K Map shows the relationship between inputs & outputs
- Horizontally & vertically adjacent squares differ only in one variable.

A	B	X
0	0	1 → $\bar{A}\bar{B}$
0	1	0
1	0	0
1	1	1 → AB

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

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

A	B	C	X
0	0	0	1 → $\bar{A}\bar{B}\bar{C}$
0	0	1	1 → $\bar{A}\bar{B}C$
0	1	0	1 → $\bar{A}B\bar{C}$
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1 → $AB\bar{C}$
1	1	1	0

$$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$$

(b)

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	1
$\bar{A}B$	1	0
$A\bar{B}$	1	0
AB	0	0

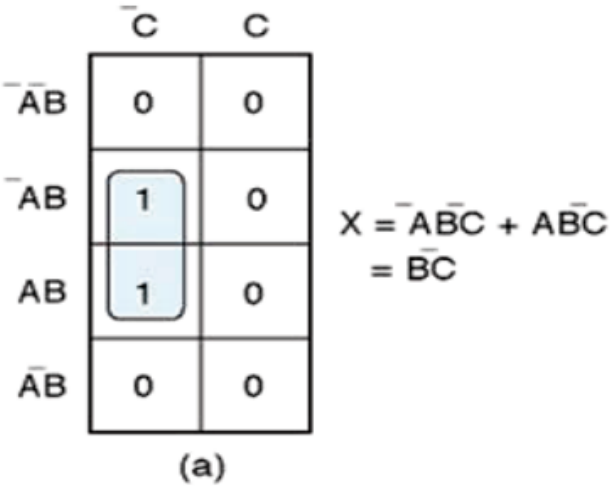
A	B	C	D	X	
0	0	0	0	0	
0	0	0	1	1	→ $\bar{A}\bar{B}\bar{C}D$
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	→ $\bar{A}B\bar{C}D$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	1	→ $AB\bar{C}D$
1	1	1	0	0	
1	1	1	1	1	→ $ABCD$

$$X = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + AB\bar{C}D + ABCD$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}B$	0	1	0	0
$\bar{A}B$	0	1	0	0
AB	0	1	1	0
$A\bar{B}$	0	0	0	0

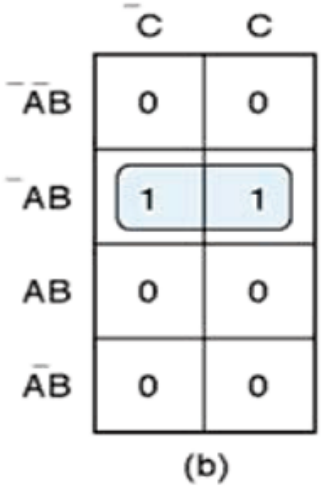
(c)

Looping is a process combining the squares which contain 1s. The output expression can be simplified by looping.



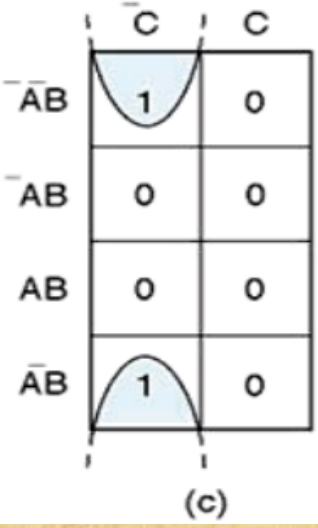
(a)

$X = \bar{A}B\bar{C} + AB\bar{C}$
$= B\bar{C}$



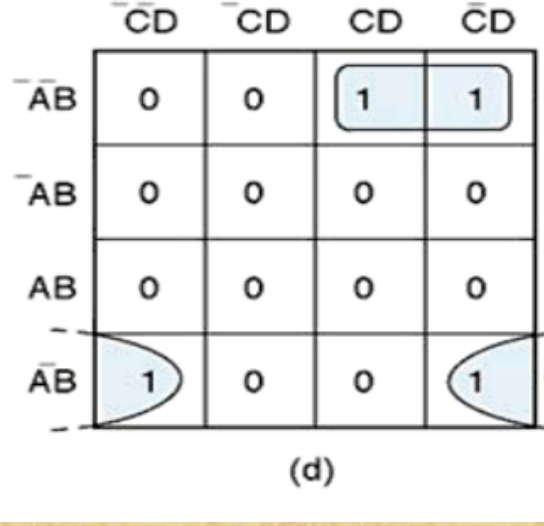
(b)

$X = \bar{A}B\bar{C} + \bar{A}BC$
$= \bar{A}B$



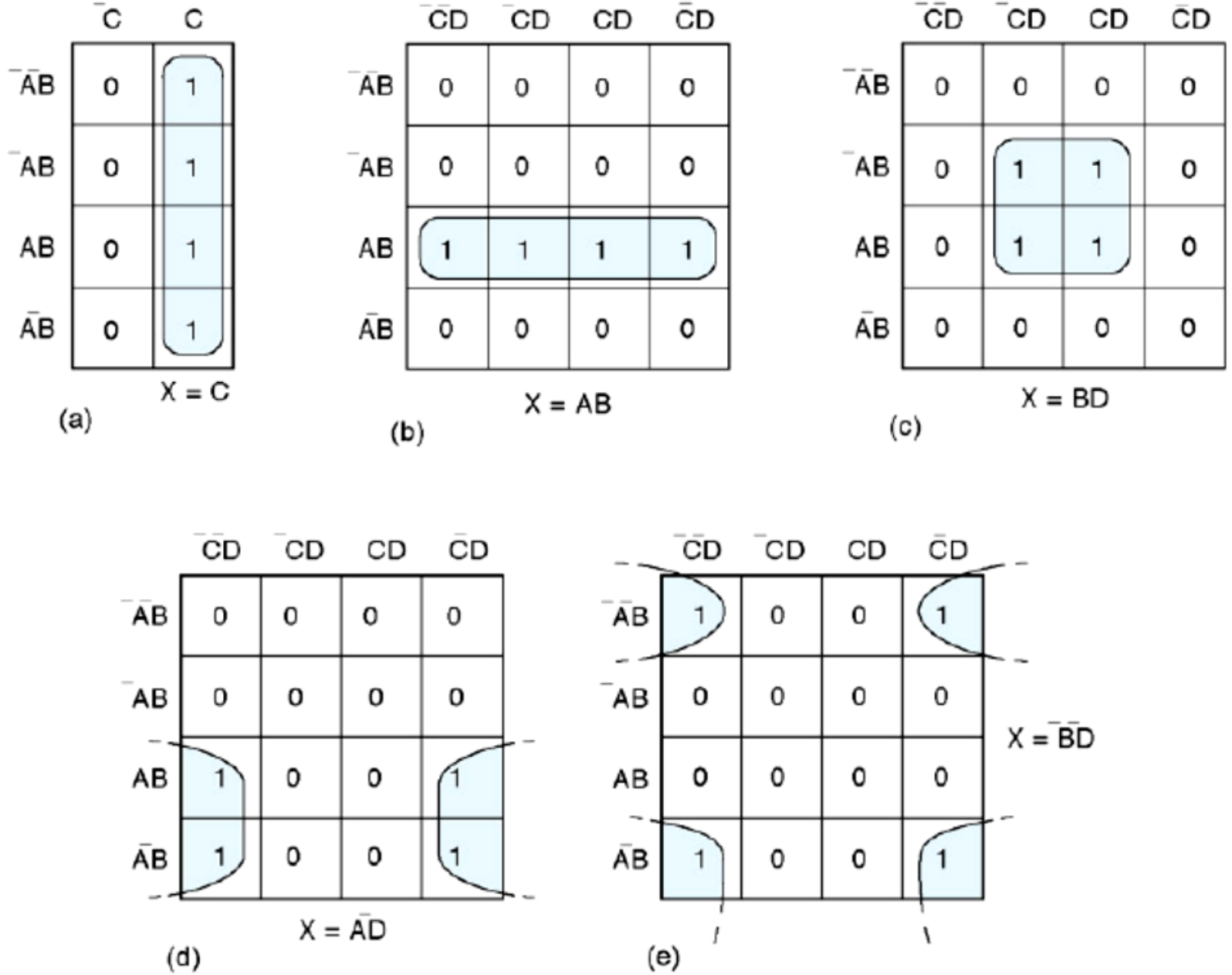
(c)

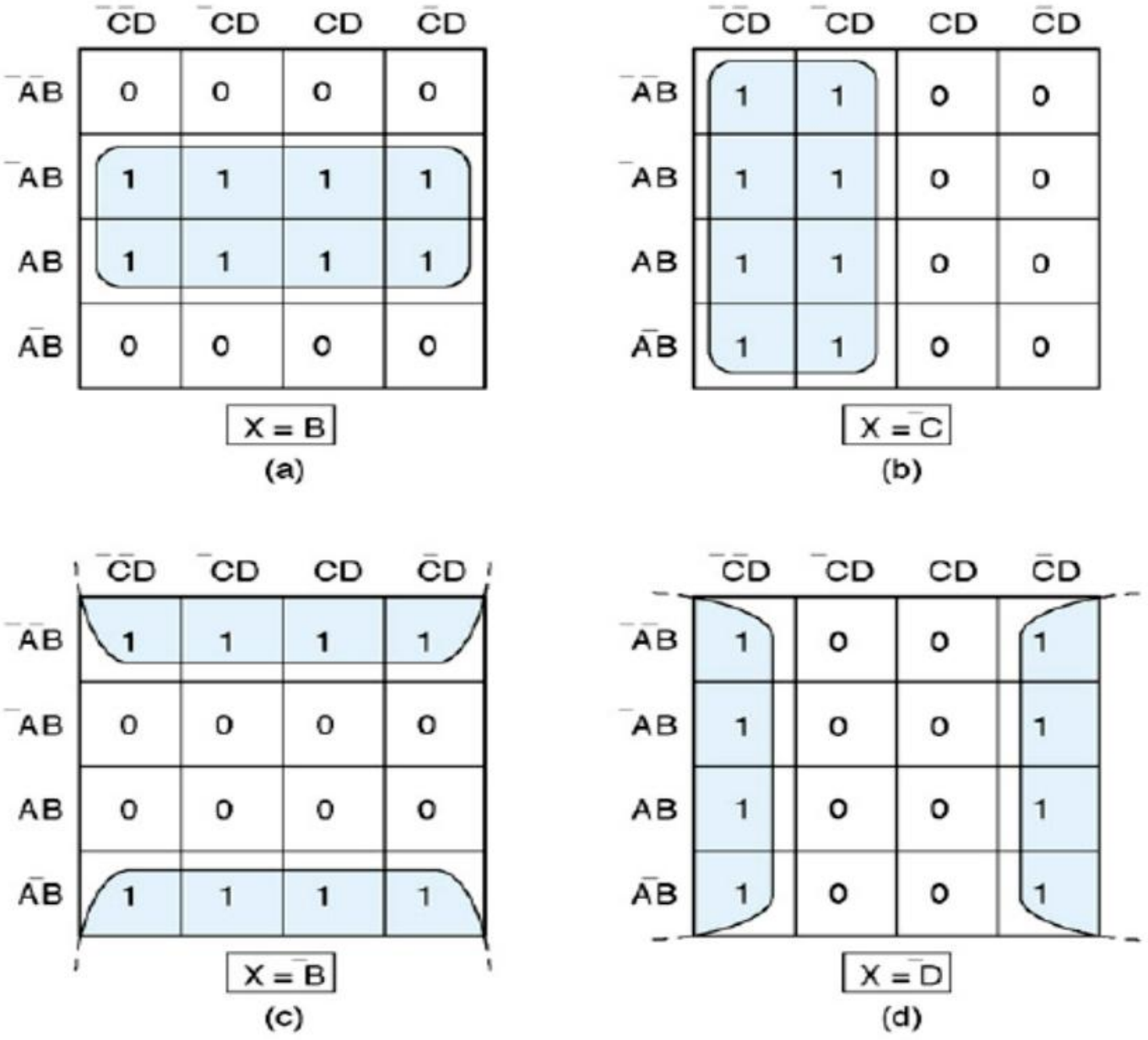
$X = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = \bar{B}\bar{C}$



(d)

$X = \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D}$
$+ \bar{A}BCD + A\bar{B}C\bar{D}$
$= \bar{A}\bar{B}C + \bar{A}\bar{B}D$


$\bar{C}$ C
$\bar{A}\bar{B}$ 0 1
$\bar{A}B$ 0 1
AB 0 1
$A\bar{B}$ 0 1
X = C
(a)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 0 0 0 0
$\bar{A}B$ 0 0 0 0
AB 1 1 1 1
$A\bar{B}$ 0 0 0 0
X = AB
(b)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 0 0 0 0
$\bar{A}B$ 0 1 1 0
AB 0 1 1 0
$A\bar{B}$ 0 0 0 0
X = BD
(c)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 0 0 0 0
$\bar{A}B$ 0 0 0 0
AB 1 0 0 1
$A\bar{B}$ 1 0 0 1
X = $A\bar{D}$
(d)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 1 0 0 1
$\bar{A}B$ 0 0 0 0
AB 0 0 0 0
$A\bar{B}$ 1 0 0 1
X = $\bar{B}\bar{D}$
(e)

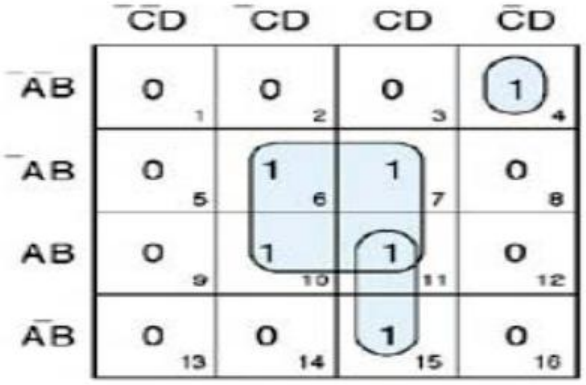

$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 0 0 0 0
$\bar{A}B$ 1 1 1 1
AB 1 1 1 1
$A\bar{B}$ 0 0 0 0
X = B
(a)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 1 1 0 0
$\bar{A}B$ 1 1 0 0
AB 1 1 0 0
$A\bar{B}$ 1 1 0 0
X = $\bar{C}$
(b)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 1 1 1 1
$\bar{A}B$ 0 0 0 0
AB 0 0 0 0
$A\bar{B}$ 1 1 1 1
X = $\bar{B}$
(c)
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ 1 0 0 1
$\bar{A}B$ 1 0 0 1
AB 1 0 0 1
$A\bar{B}$ 1 0 0 1
X = $\bar{D}$
(d)

Rule for loops of any size

When a variable appears in both complemented & uncomplemented form within a loop, that variable is eliminated from the expression. Variables that are the same for all squares of the loop must appear in the final expression.

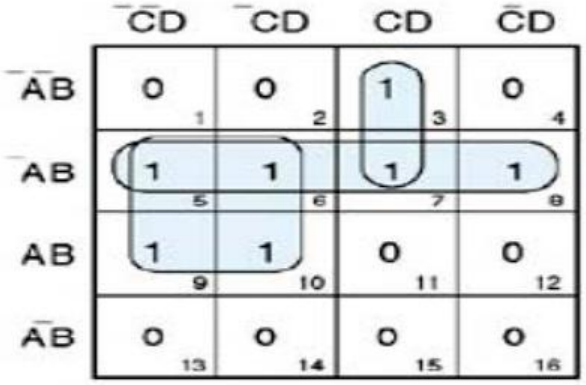
Complete Simplification Process

1. Construct the K map and place 1s and 0s in the squares according to the truth table.
2. Loop the isolated 1s which are not adjacent to any other 1s. (single loops)
3. Loop any pair which contains a 1 adjacent to only one other 1. (double loops)
4. Loop any octet even if it contains one or more 1s that have already been looped.
5. Loop any quad that contains one or more 1s that have not already been looped, making sure to use the minimum number of loops.
6. Loop any pairs necessary to include any 1s that have not yet been looped, making sure to use the minimum number of loops.
7. Form the OR sum of all the terms generated by each loop.



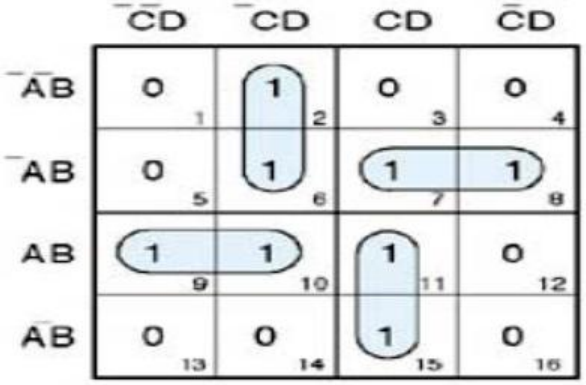
$$X = \underbrace{\bar{A}BC\bar{D}}_{\text{loop 4}} + \underbrace{ACD}_{\text{loop 11, 15}} + \underbrace{BD}_{\text{loop 6, 7, 10, 11}}$$

(a)



$$X = \underbrace{\bar{A}\bar{B}}_{\text{loop 5, 6, 7, 8}} + \underbrace{\bar{B}\bar{C}}_{\text{loop 5, 6, 9, 10}} + \underbrace{\bar{A}CD}_{\text{loop 3, 7}}$$

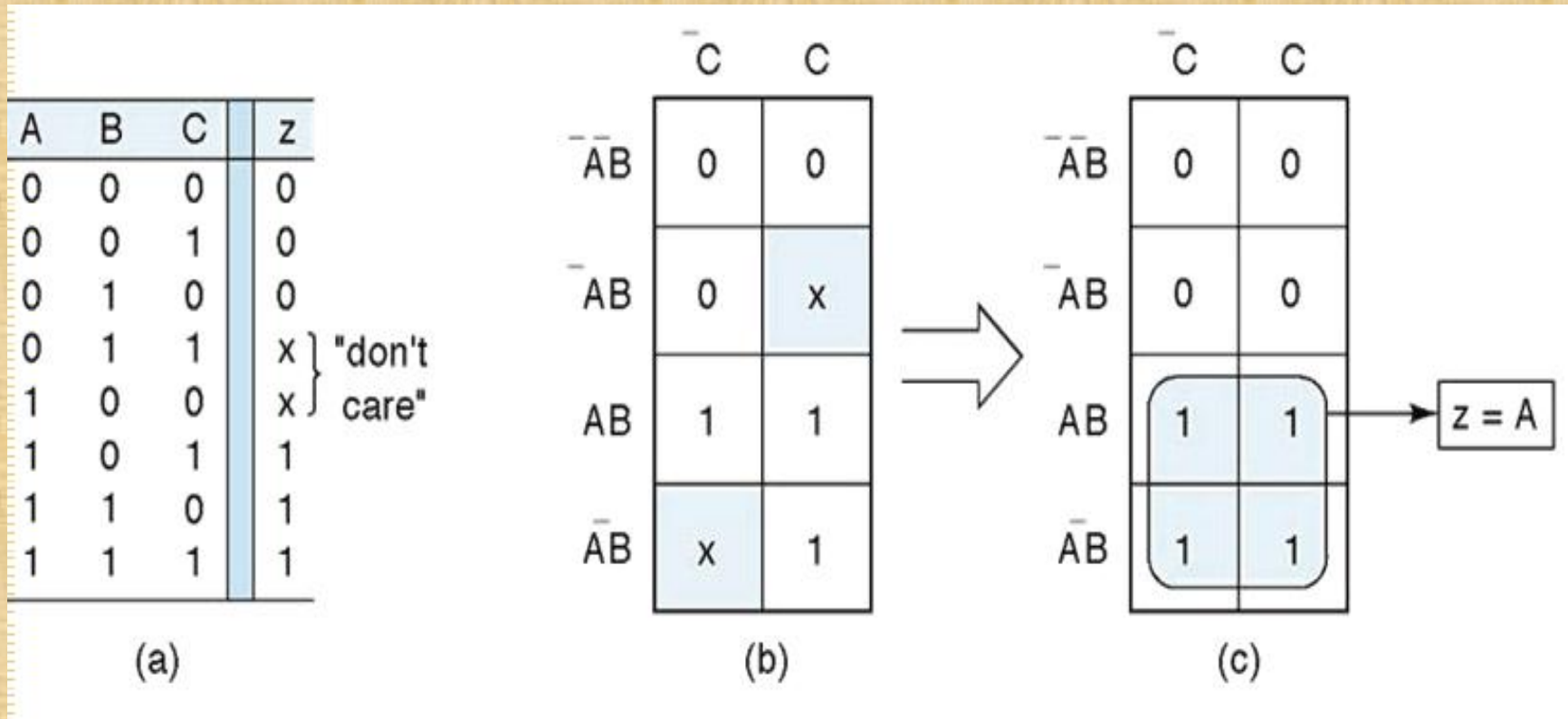
(b)



$$X = \underbrace{AB\bar{C}}_{9, 10} + \underbrace{\bar{A}CD}_{2, 6} + \underbrace{\bar{A}BC}_{7, 8} + \underbrace{ACD}_{11, 15}$$

(c)

"Don't-Care" Conditions are certain input conditions for which there are no specified output levels. "Don't-care" conditions should be changed to either 0 or 1 to produce K-map looping that yields the simplest expression.



A	B	C	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	x
1	0	0	x
1	0	1	1
1	1	0	1
1	1	1	1

"don't care"

(a)

	$\bar{C}$	C
$\bar{A}\bar{B}$	0	0
$\bar{A}B$	0	x
AB	1	1
$A\bar{B}$	x	1

(b)

	$\bar{C}$	C
$\bar{A}\bar{B}$	0	0
$\bar{A}B$	0	0
AB	1	1
$A\bar{B}$	1	1

z = A

(c)

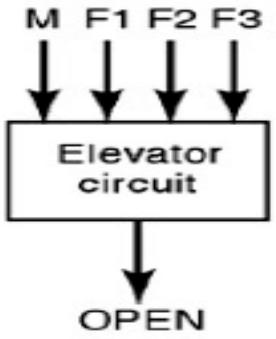
Filling K-Map from Output Expression

When the desired output is presented as a Boolean expression instead of a truth table, the K map can be filled by using the following steps:

1. Get the expression into SOP form if it is not already so.
2. For each product term in the SOP expression, place a 1 in each K-map square whose label contains the same combination of input variables. Place a 0 in all other squares.

- Don’t care condition can come about for several reasons:
  - In some situations certain input combination can never occur and so there is no specified output for these condition.
- Whenever don’t care conditions occur, we must decide which x to change to 0 and which to 1 to produce the best K-map looping (i.e the simplest expression)

Example



(a)

M	F1	F2	F3	OPEN
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

(b)

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$M\,\overline{F1}$	0	1	X	1
$\overline{M}\,F1$	1	X	X	X
$M\,F1$	0	X	X	X
$M\,\overline{F1}$	0	0	X	0

(c)

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$M\,\overline{F1}$	0	1	1	1
$\overline{M}\,F1$	1	1	1	1
$M\,F1$	0	0	0	0
$M\,\overline{F1}$	0	0	0	0

$OPEN = \overline{M}\,(F1 + F2 + F3)$

(d)

Example

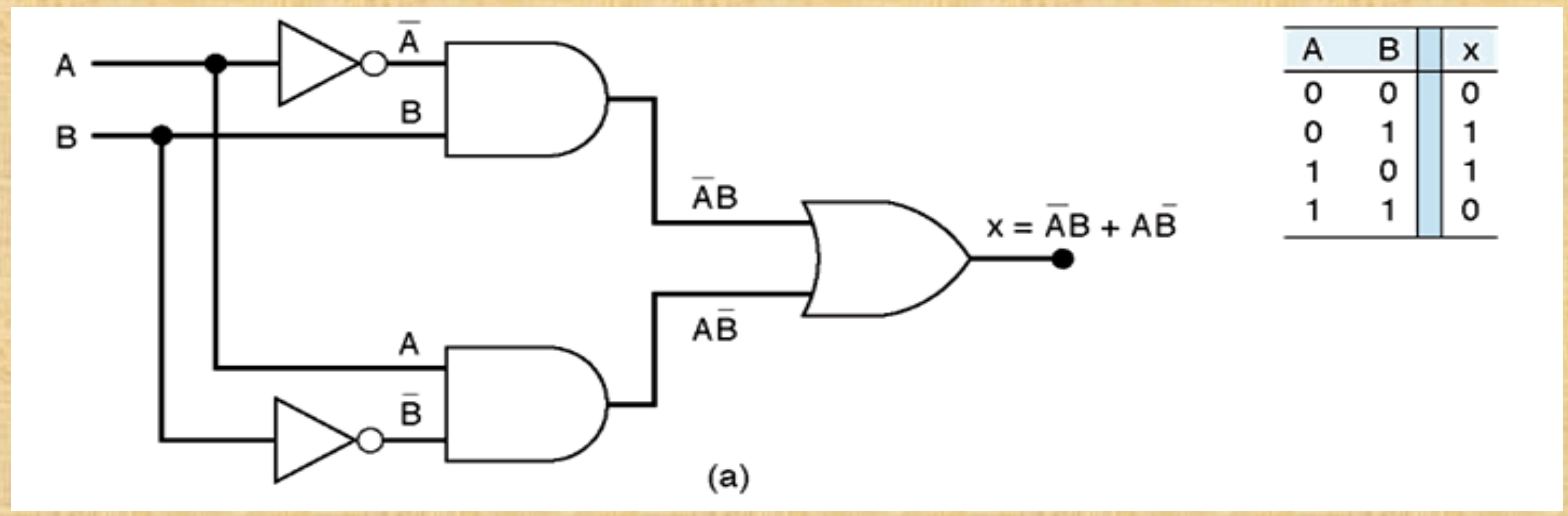
A BCD counter produces a four bit output representing the BCD code for the number of pulses hat have been applied to the counter input. For example, after 4 pulses have occurred, the counter outputs are DCBA= 01002 = 410. The counter resets to 0000 on the tenth pulse and starts counting over again. In other words, the DCBA output will never represent a number greater than 10012=910. Design the logic circuit that produces a HIGH output whenever the count is 2,3, or 9. Use K mapping and take advantage of the don't care conditions.

Summary

- Compared to the algebraic method, the K-map process is a more orderly process requiring fewer steps and always producing a minimum expression.
- For the circuits with large numbers of inputs (larger than four), other more complex techniques are used.

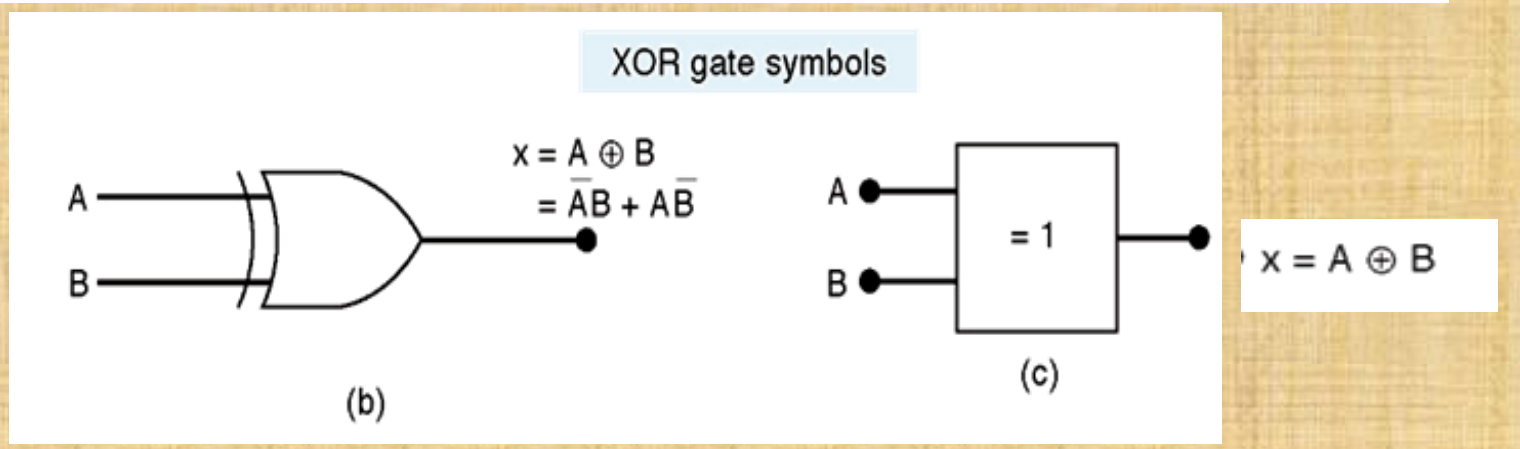
Exclusive-OR and Exclusive-NOR Circuits

Exclusive-OR (XOR) produces a HIGH output whenever the two inputs are at opposite levels.

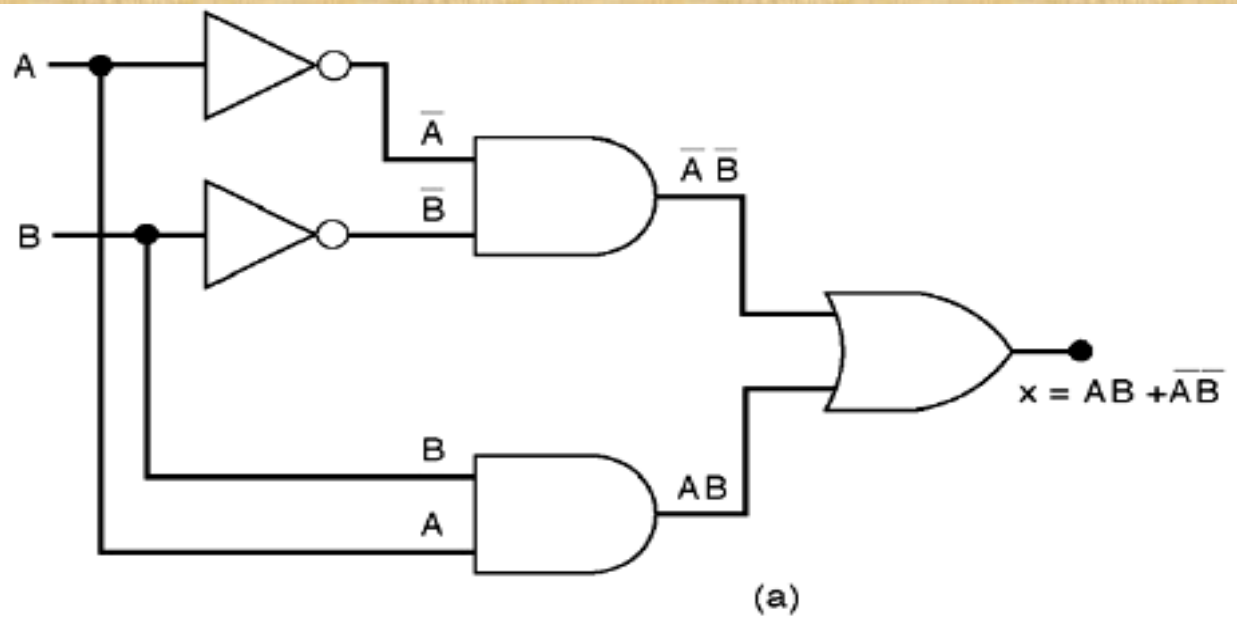


A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate symbols



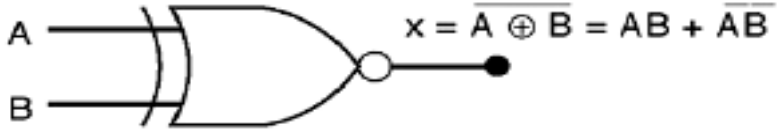
Exclusive-NOR (XNOR) produces a HIGH output whenever the two inputs are at the same level.



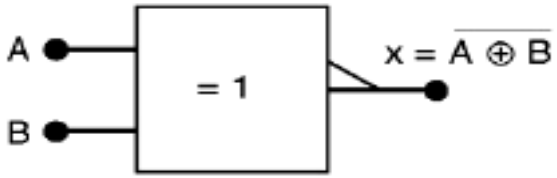
A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

(a)

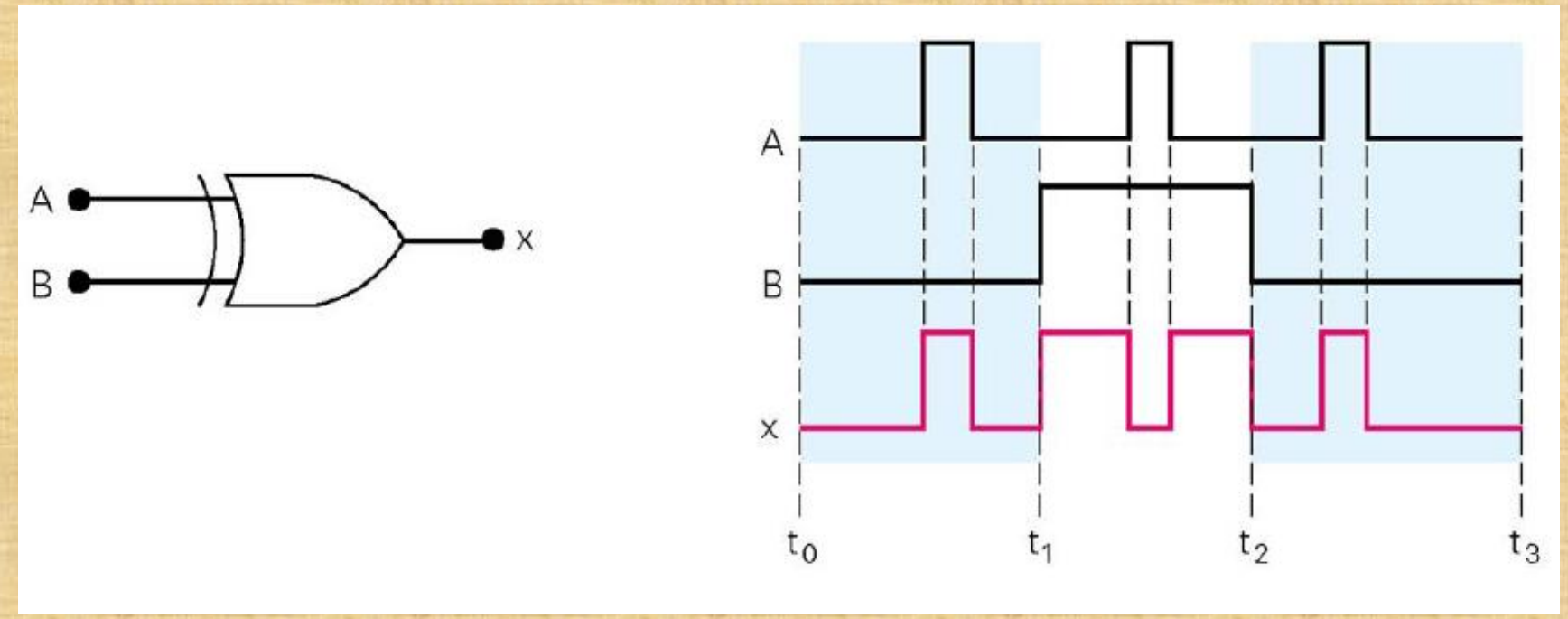
XNOR gate symbols



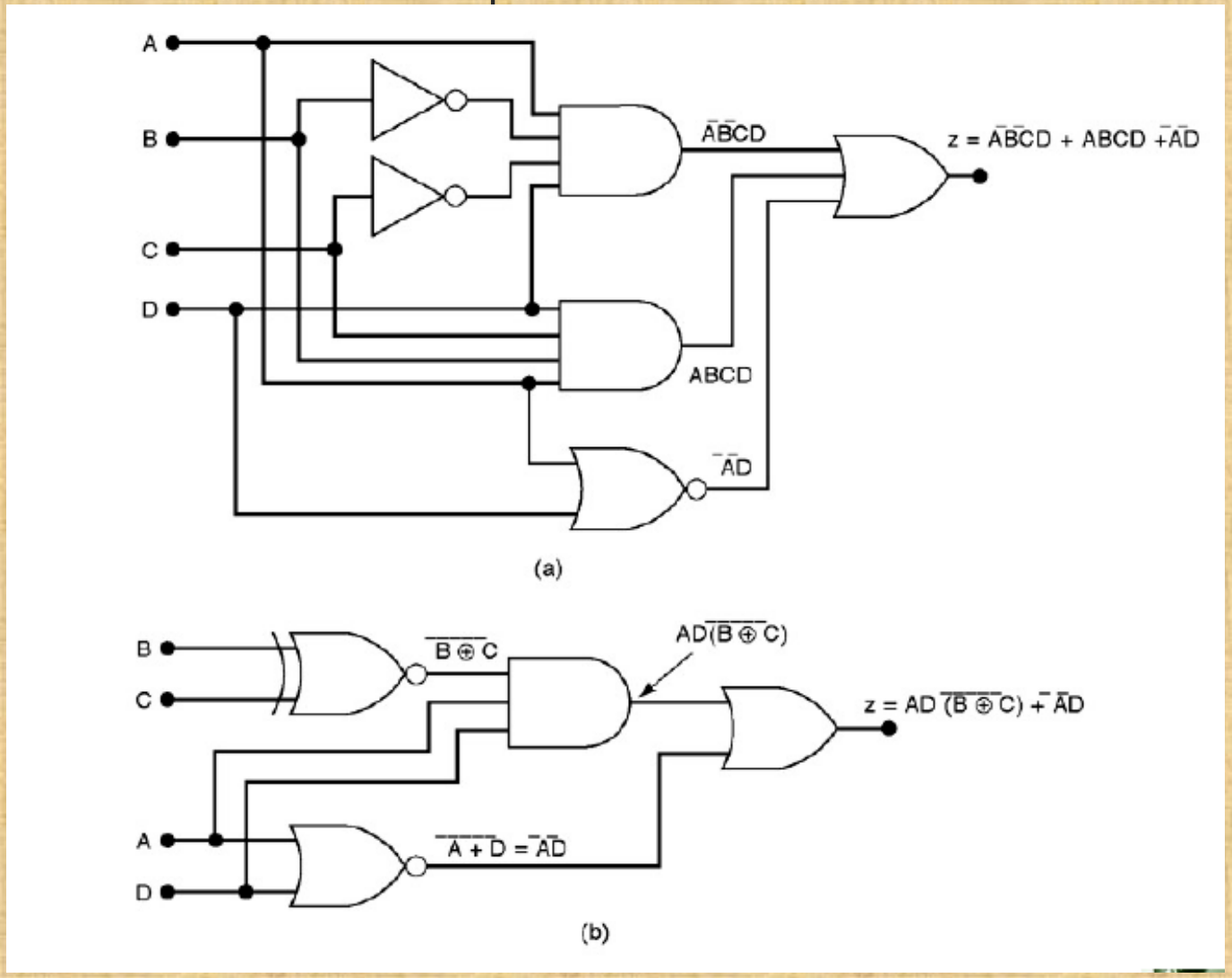
(b)



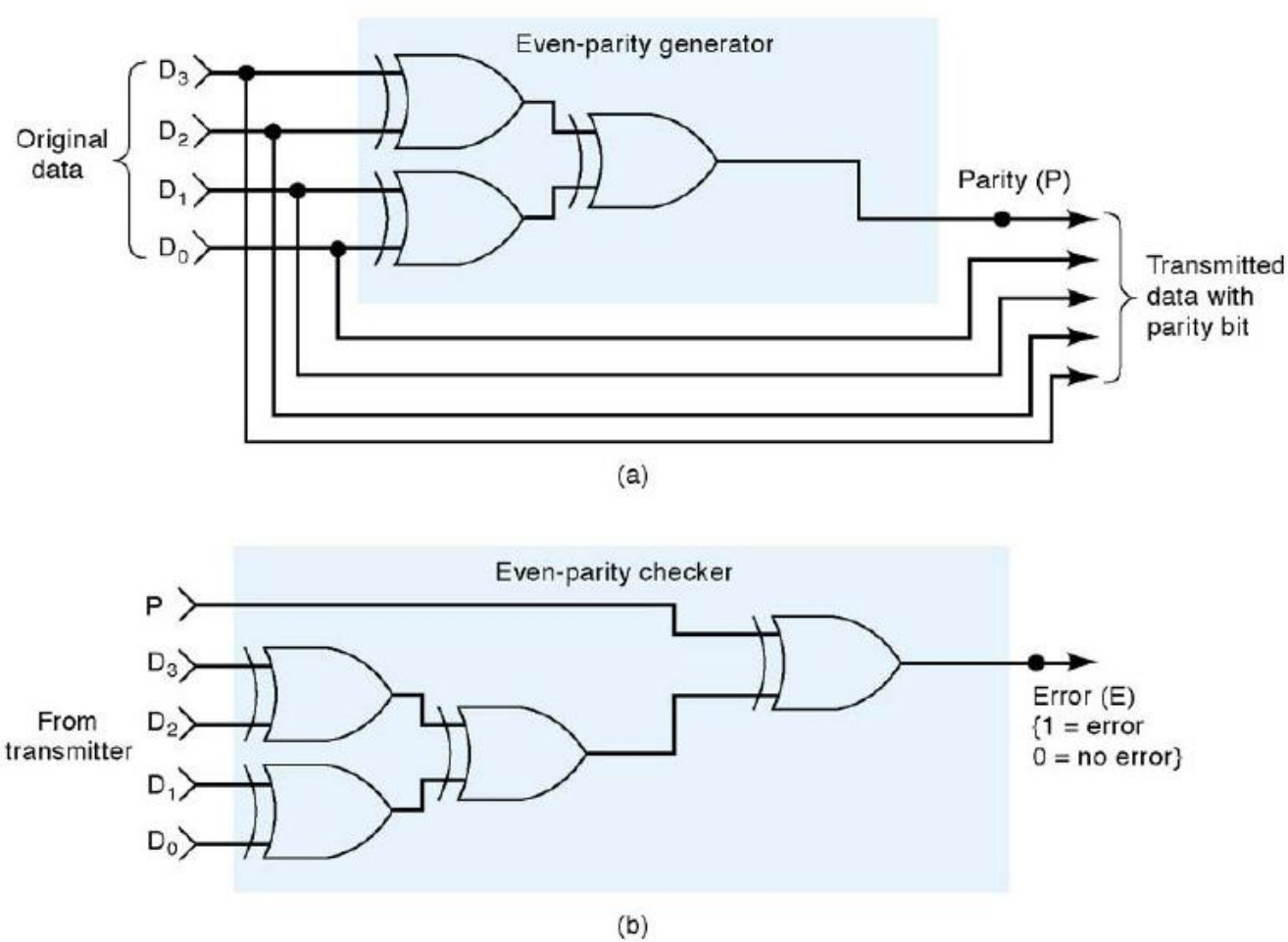
(c)


A
B
x
A
B
x
t_0
t_1
t_2
t_3

XNOR gate may be used to simplify circuit implementation.

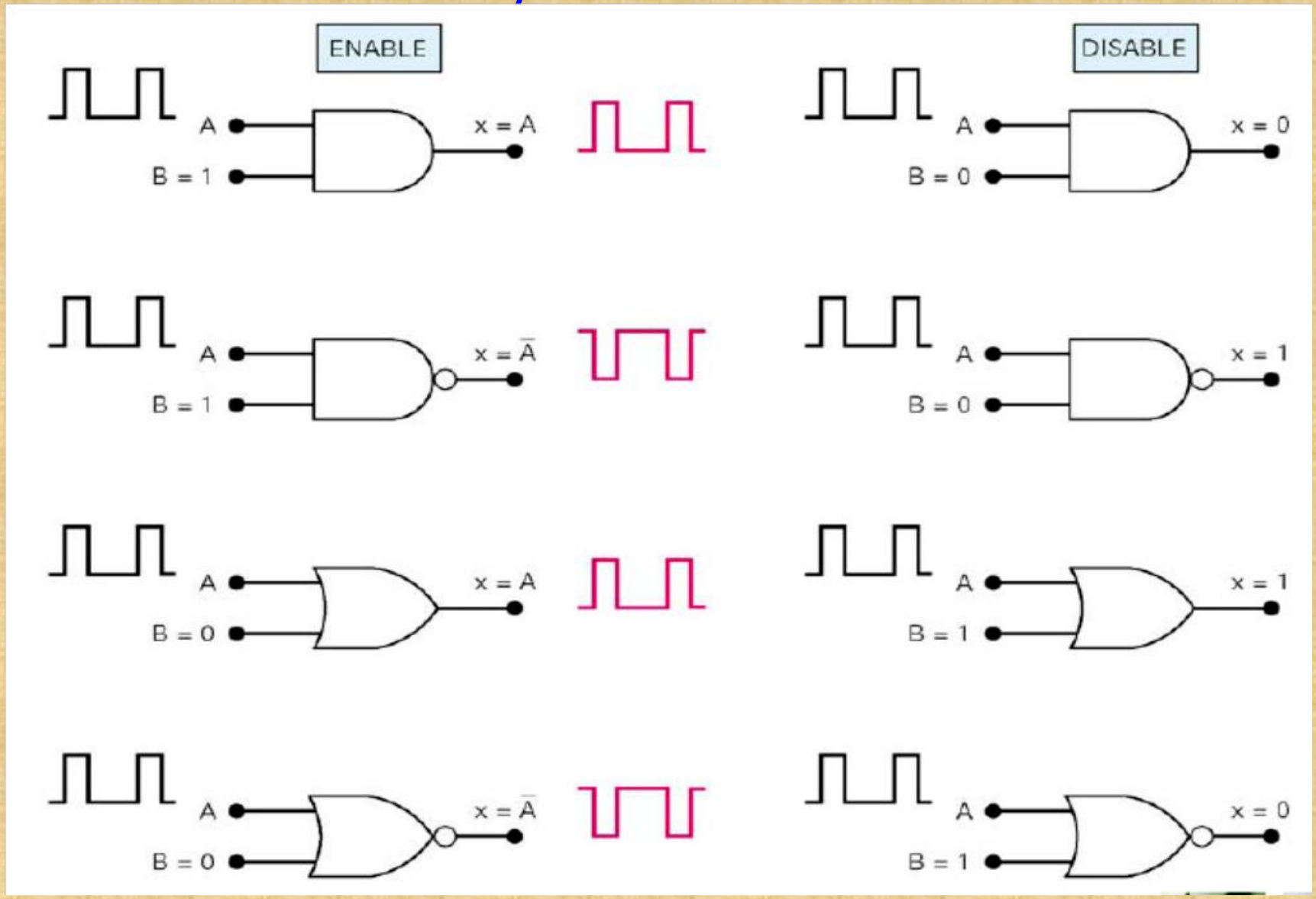


Parity Generator and Checker



- A transmitter can attach a parity bit to a set of data bits before transmitting the data bits to a receiver. The receiver will detect any single bit errors that may have occurred during the transmission.
- In figure (a) the set of data to be transmitted is applied to the parity-generator circuit, which produces the even-parity bit, P, at its output. This parity bit is transmitted to the receiver along with the original data bits, making a total of five bits.
- In figure (b) these five bits (data+parity) enter the receiver's parity-checker circuit, which produces an error output, E that indicates whether or not a single-bit error has occurred.

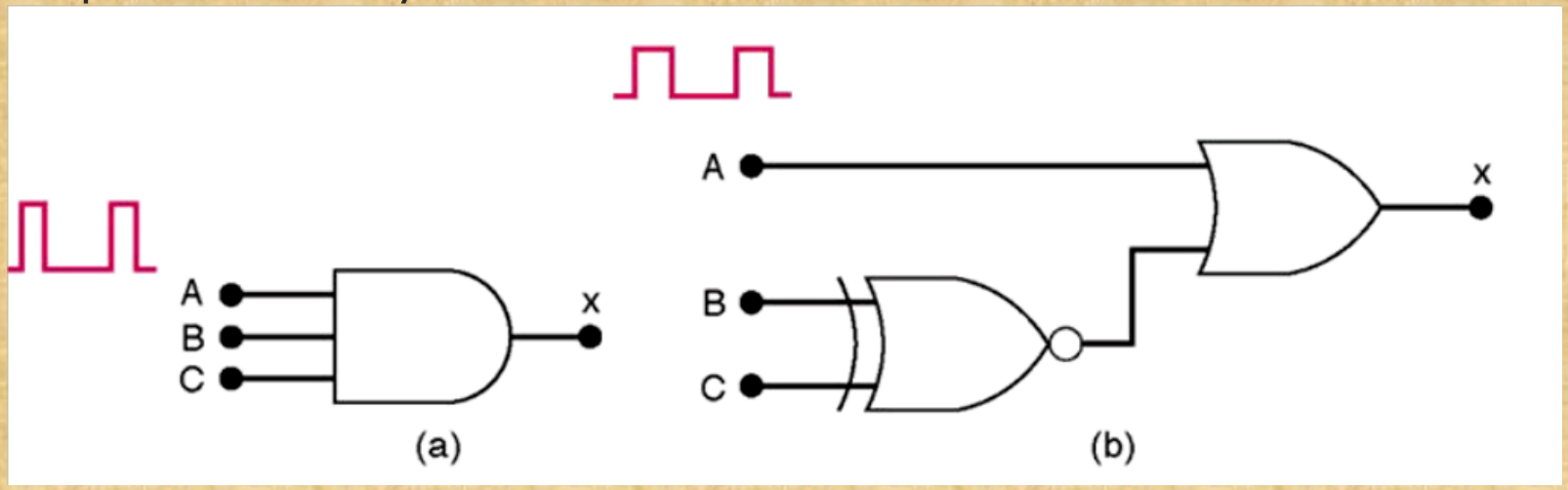
Enable/Disable Circuits



Enable/Disable Circuits cont.

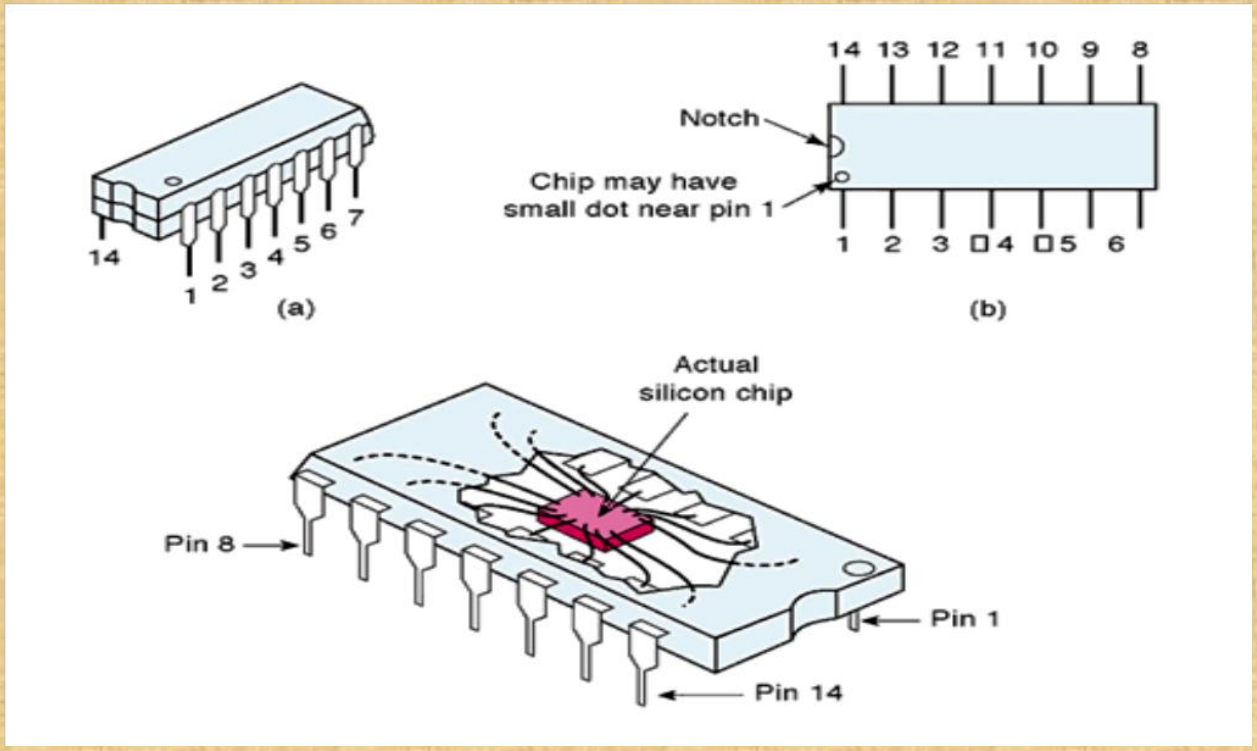
Ex. 1(Fig.a): Design a logic circuit that will allow a signal to pass to the output only when control inputs B and C are both HIGH; otherwise, the output will stay LOW.

Ex. 2(Fig.b): Design a logic circuit that will allow a signal to pass to the output only when one, but not both, of the control inputs are HIGH; otherwise, the output will stay LOW.



Basic Characteristics of Digital ICs

- Digital ICs (chips): a collection of resistors, diodes and transistors fabricated on a single piece of semiconductor materials called substrate.
- Dual-in-line package (DIP) is a common type of packages. It contains two parallel rows of pins.



- Digital ICs are often categorized according to their circuit complexity as measured by the number of equivalent logic gates on the substrates. 6 levels of complexity:

  SSI, MSI, LSI,VLSI, ULSI,GSI.

- SSI – having a small number of gates

Multiplexer/De-multiplexer

Mux/Demux Vocabulary

MULTIPLEXER (aka DATA SELECTOR)- circuit that can select one of a number of inputs and pass the logic level of that input to the output.

DEMULTIPLEXER (aka DATA DISTRIBUTOR)- circuit that depending on the status of its select inputs will channel its data input to one of several outputs.

SELECT INPUTS (aka ADDRESS LINES)- used by the mux to determine which data inputs will be switched to the output.

$$if\ 2^N\ input\ lines = N\ select\ lines$$

Example of a Combinatorial Circuit: A Multiplexer (MUX)

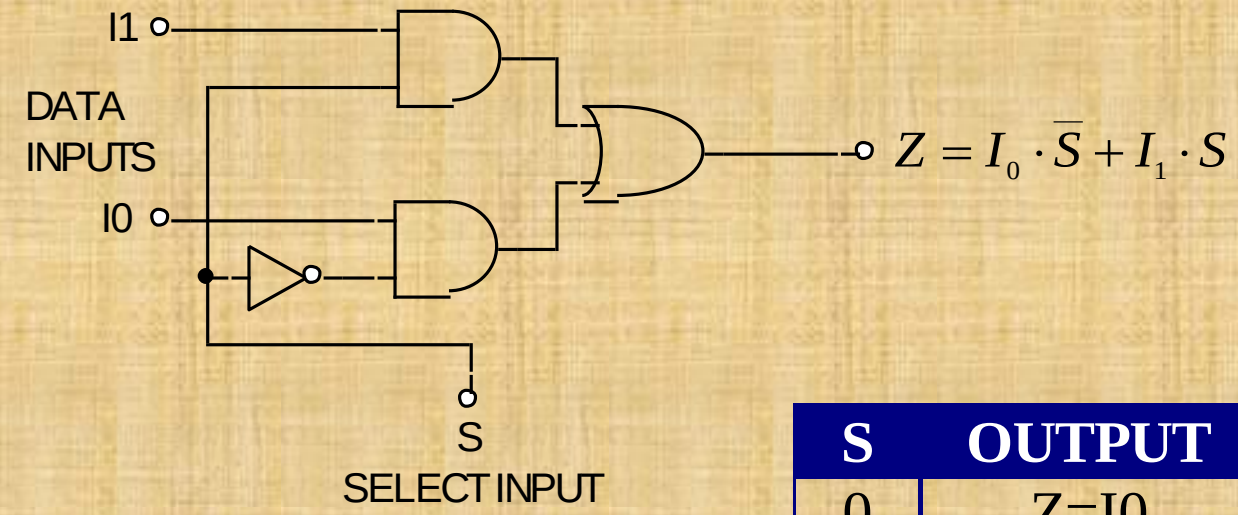
Consider an integer ‘m’, which is constrained by the following relation:

$\mathbf{m = 2^n}$, where m and n are both integers.

- **A m-to-1 Multiplexer** has
  - m Inputs: $I_0, I_1, I_2, \ldots\ldots\ldots\ldots I_{(m-1)}$
  - one Output: Y
  - n Control inputs: $S_0, S_1, S_2, \ldots\ldots S_{(n-1)}$
  - One (or more) Enable input(s)

such that Y may be equal to one of the inputs, depending upon the control inputs.

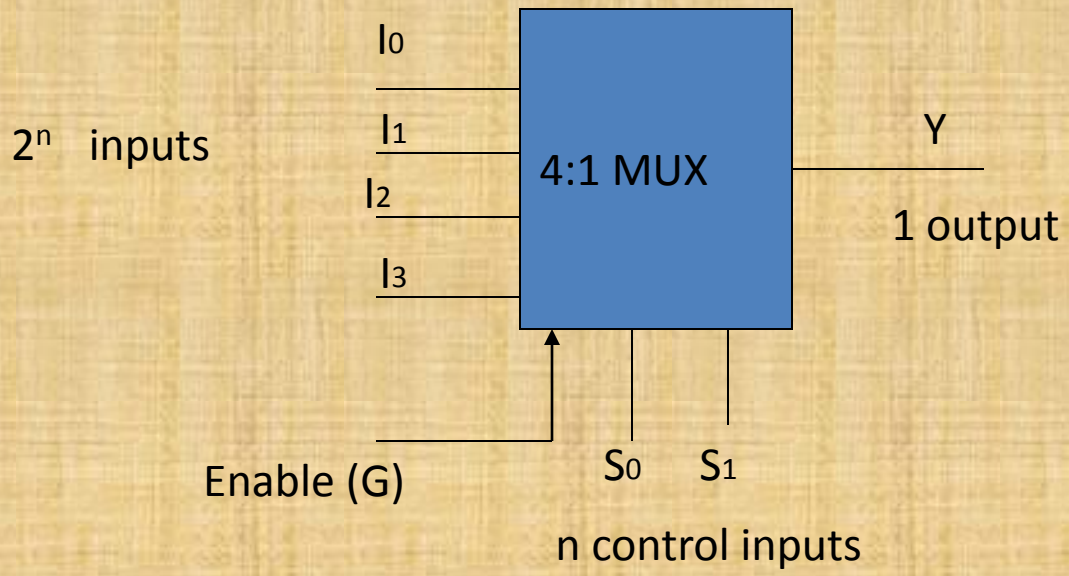
BASIC TWO-INPUT MULTIPLEXER



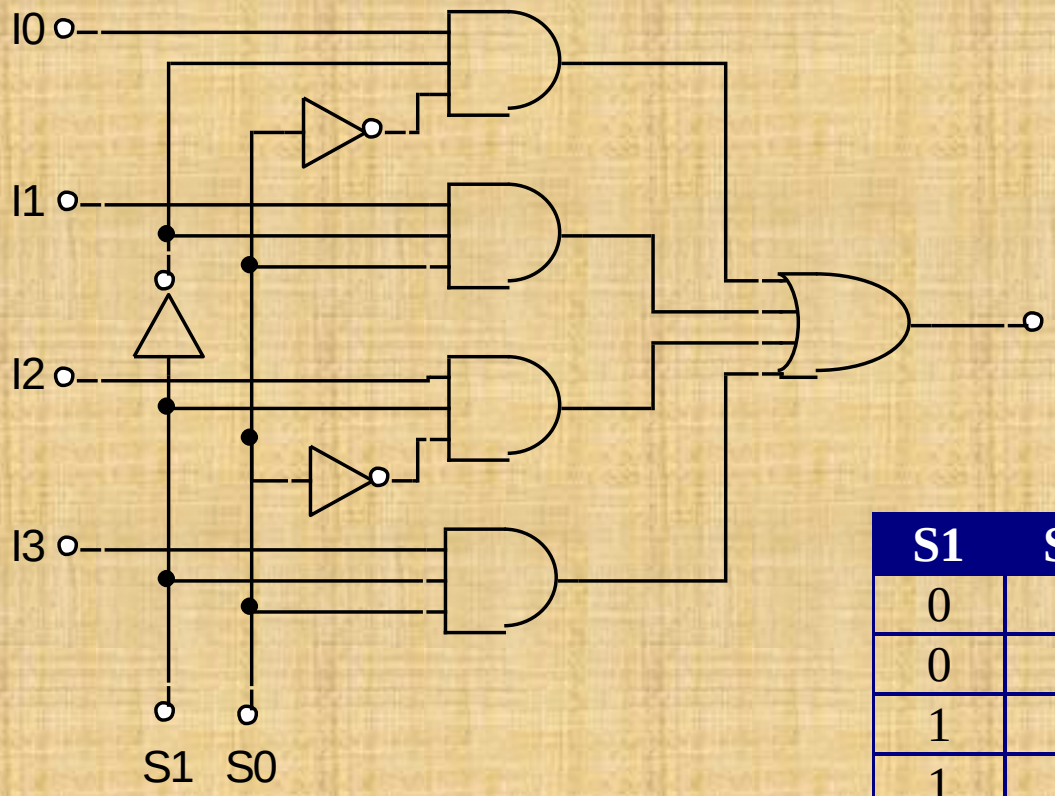
S	OUTPUT
0	Z=I0
1	Z=I1

Example: A 4-to-1 Multiplexer

A 4-to-1 Multiplexer:



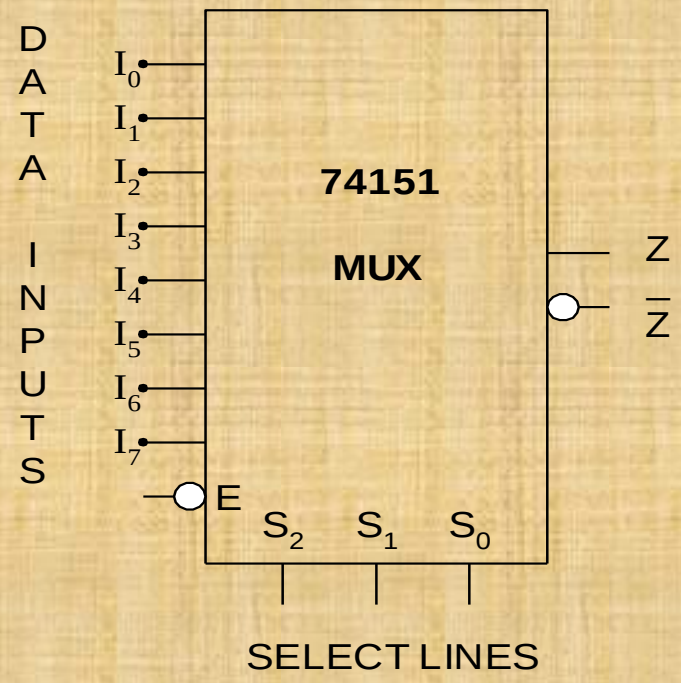
FOUR-INPUT MULTIPLEXER



S1	S0	OUTPUT
0	0	Z=I0
0	1	Z=I1
1	0	Z=I2
1	1	Z=I3

MULTIPLEXER LOGIC DIAGRAM

- Takes one of many inputs and funnels it to an output Z.
- Take the selector lines convert to a decimal number and this is the input funneled to the output.
- Strobe is active low enable

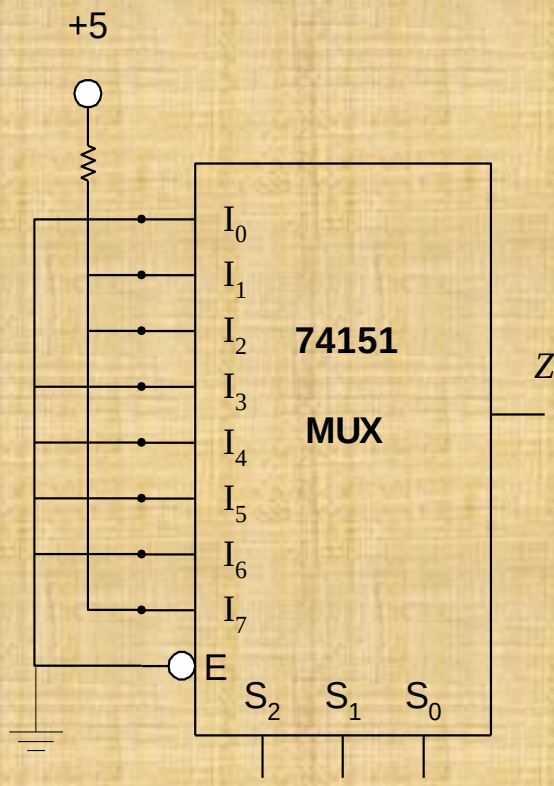


S2	S1	S0	E	Z
0	0	0	0	I0
0	0	1	0	I1
0	1	0	0	I2
0	1	1	0	I3
1	0	0	0	I4
1	0	1	0	I5
1	1	0	0	I6
1	1	1	0	I7

MULTIPLEXER APPLICATIONS

- DATA ROUTING
- PARALLEL-TO-SERIAL CONVERSION
- OPERATION SEQUENCING
- IMPLEMENT LOGIC FUNCTION OF A TRUTH TABLE

LOGIC FUNCTION GENERATION



$$Z = A\overline{B}\overline{C} + \overline{A}B\overline{C} + ABC$$

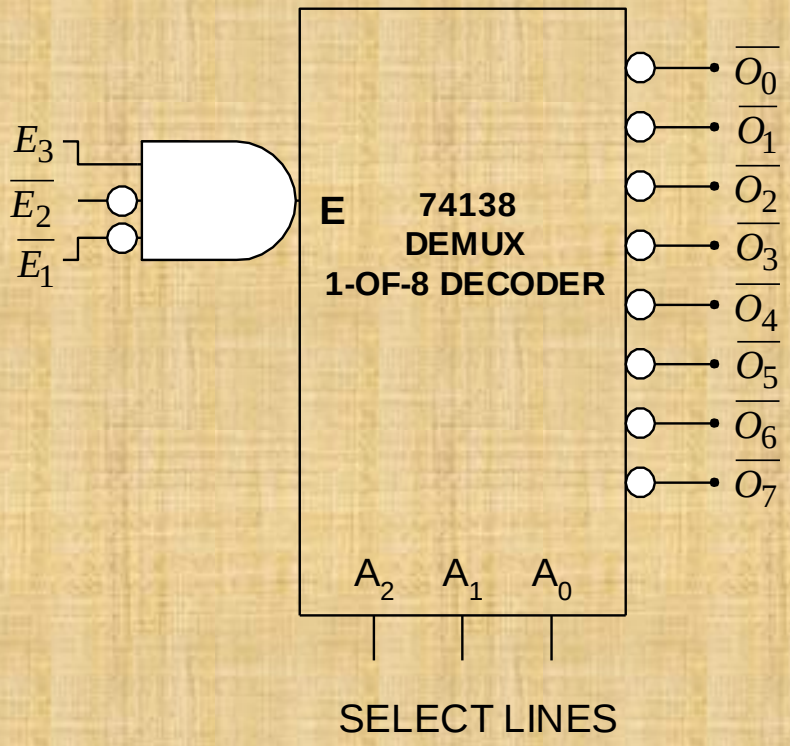
C	B	A	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

DEMULTIPLEXER

DEMULTIPLEXER LOGIC DIAGRAM

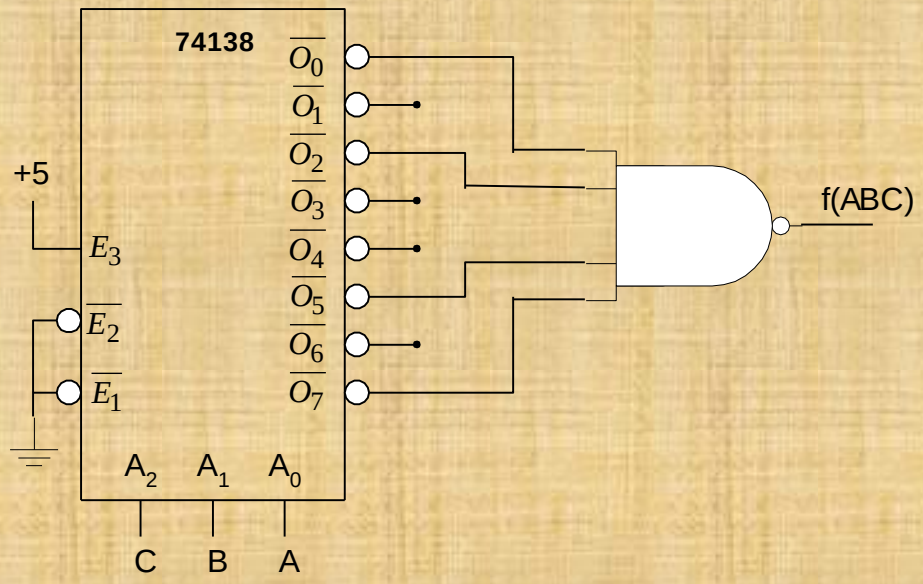
- Logic circuit that depending on the status of its select inputs will funnel its data input to one of several data outputs.

- Separate enable inputs (useful for cascading decoders) into AND gate which must be high to enable the decoder outputs.



$\overline{E_1}$	$\overline{E_2}$	E_3	*OUTPUTS*
0	0	1	RESPOND TO INPUT CODE A2A1A0
1	X	X	DISABLED –ALL HIGH
X	1	X	DISABLED –ALL HIGH
X	X	0	DISABLED –ALL HIGH

LOGIC FUNCTION GENERATION



A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

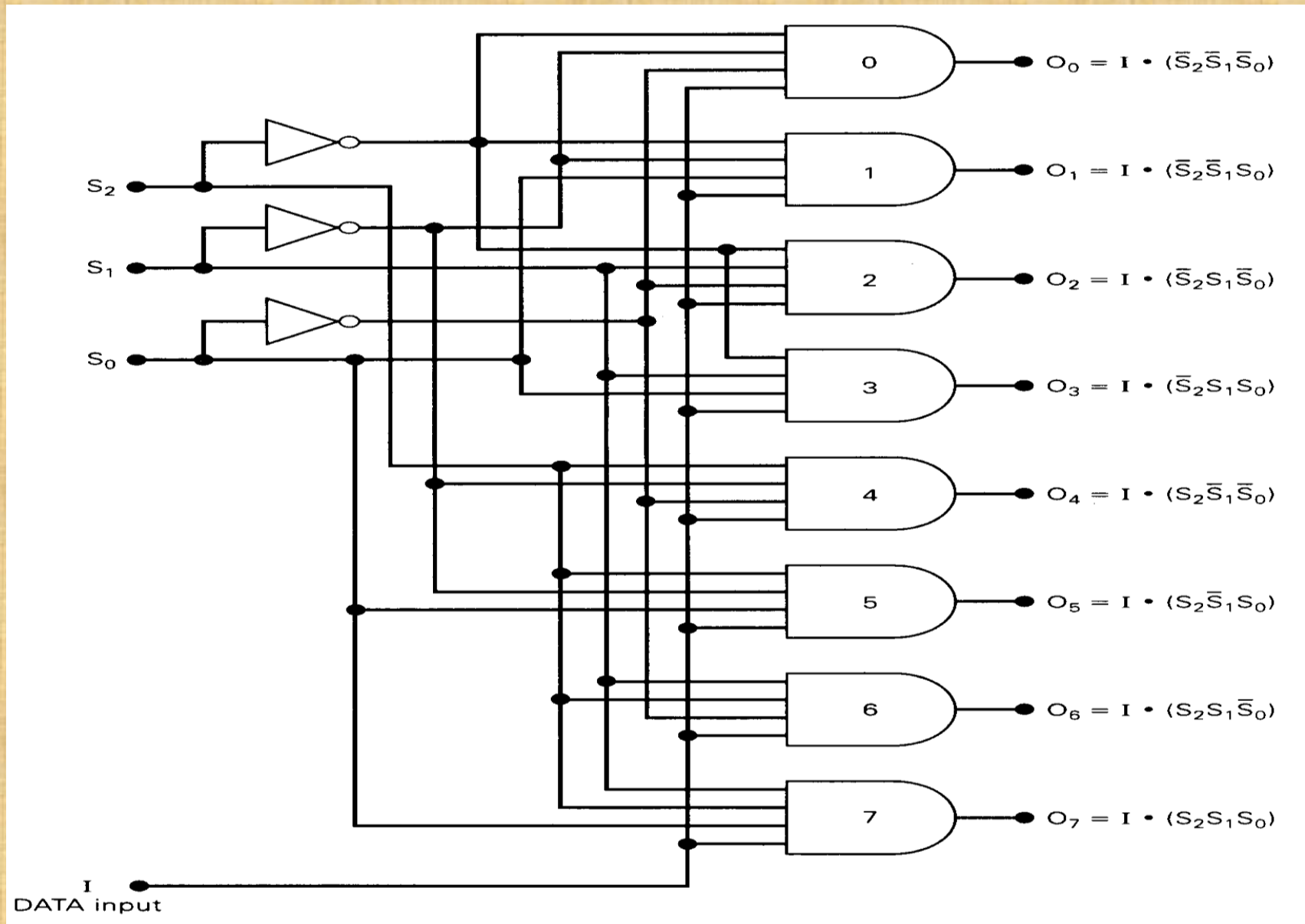
C	B	A	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

NAND- any low in gives a high out

DEMULTIPLEXER

SELECT code			OUTPUTS							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

DEMULTIPLEXER



OTHER COMBINATIONAL LOGIC CIRCUITS

DECODERS

DECODER

- A decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to the input number.

- In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the one output that corresponds to that number ; all other outputs remain inactive

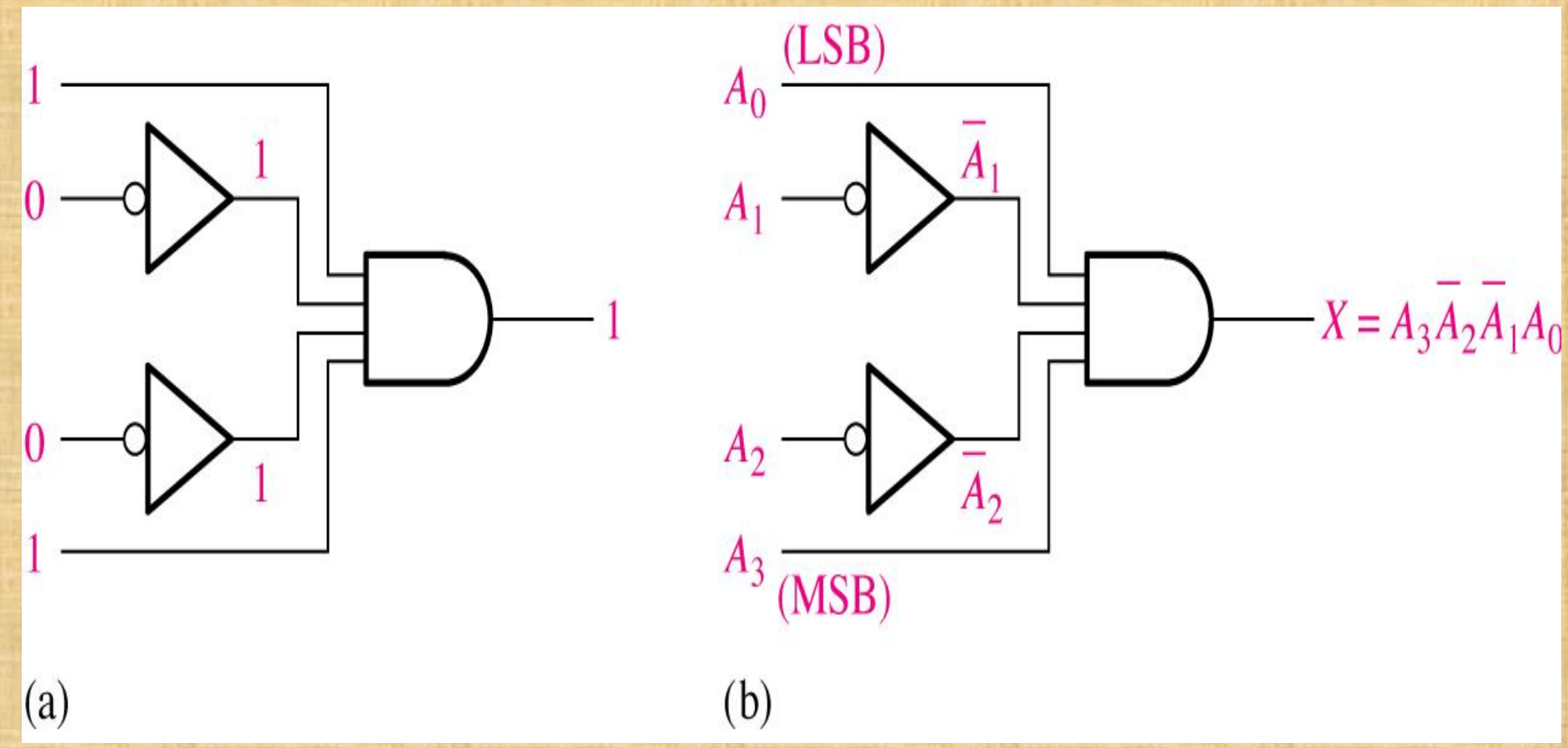
In its general form, a decoder has N input lines to handle N bits and form one to 2^N output lines to indicate the presence of one or more N-bit combinations.

The basic binary function

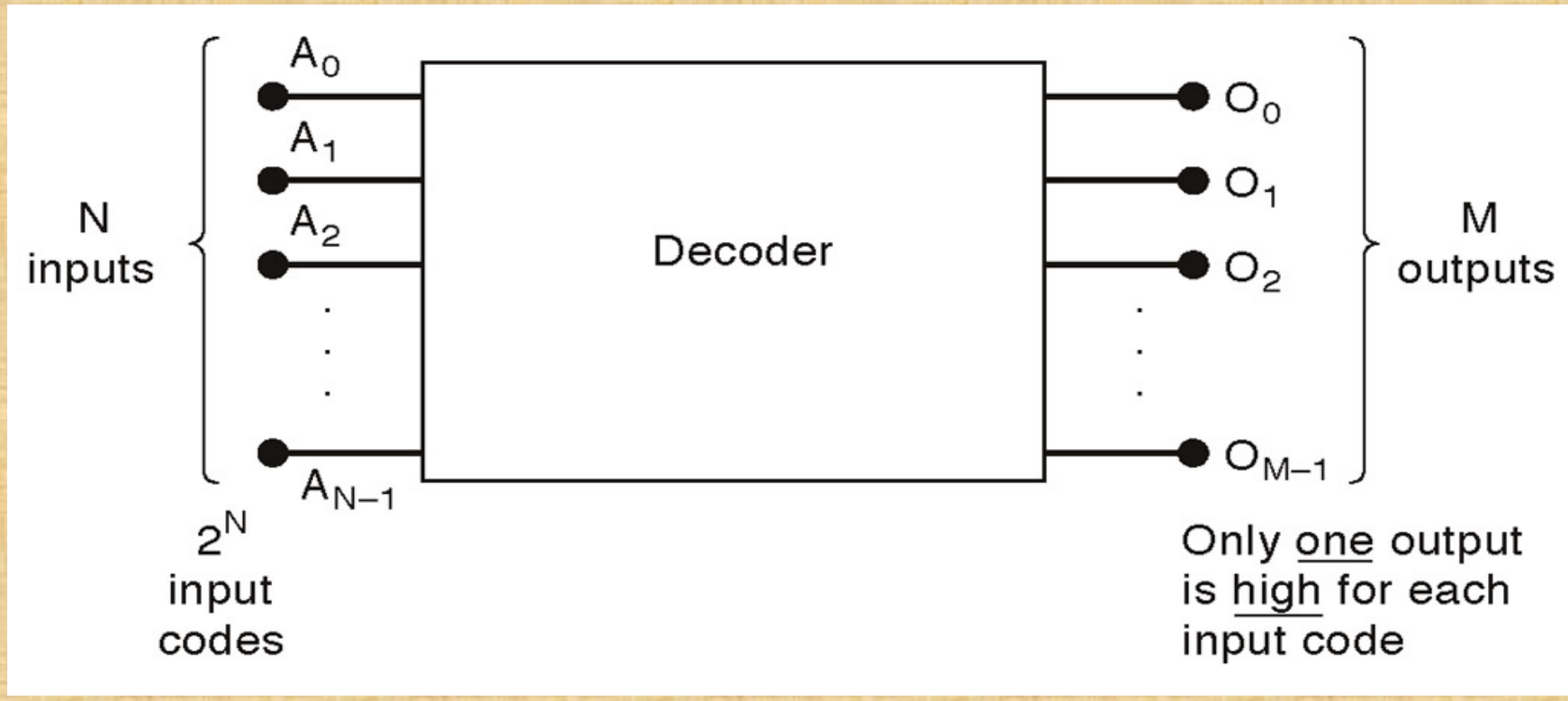
- An AND gate can be used as the basic decoding element because it produces a HIGH output only when all inputs are HIGH

Refer next slide for example

Decoding logic for the binary code 1001 with an active-HIGH output.



General decoder diagram



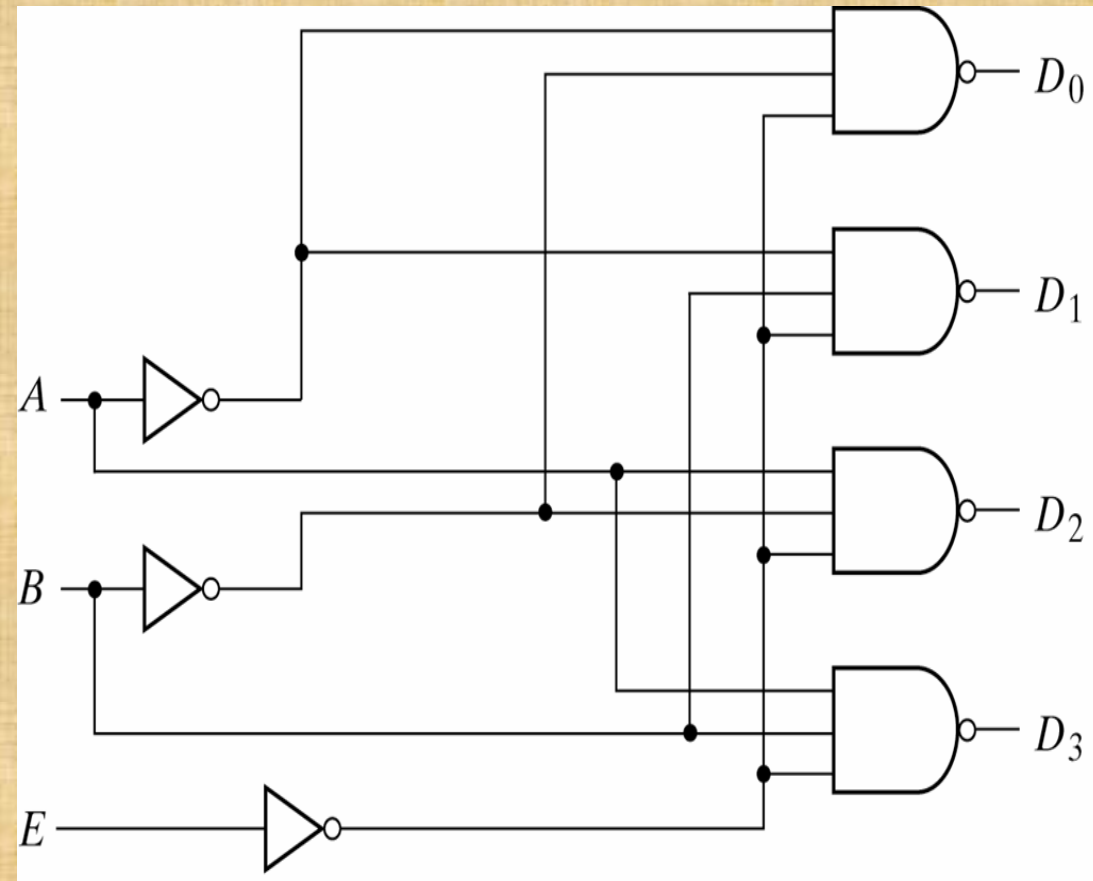
There are 2^N possible input combinations, from A_0 to A_{N-1}.

For each of these input combinations only one of the *M* outputs will be active *HIGH* (1), all the other outputs are *LOW* (0).

- If an active-LOW output (74138, one of the output will low and the rest will be high) is required for each decoded number, the entire decoder can be implemented with
    1. NAND gates
    2. Inverters

- If an active-HIGH output (74139, one of the output will high and the rest will be low) is required for each decoded number, the entire decoder can be implemented with
    - AND gates
    - Inverters

2-to-4-Line Decoder
(with Enable input)-Active LOW output (1)...



(a) Logic diagram

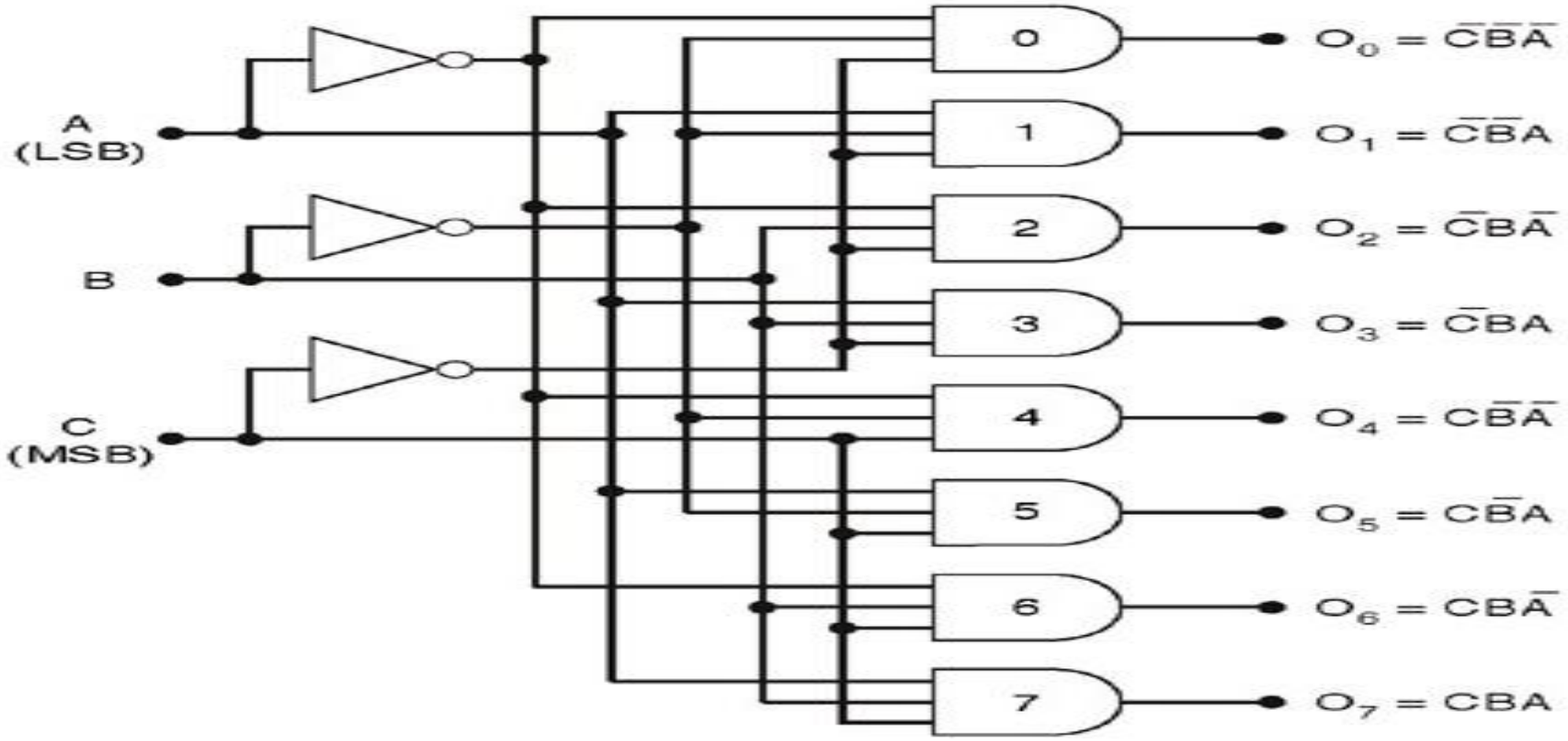
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

2-to-4-Line Decoder (with Enable input)-Active LOW output (2)

- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0.
- Only one output can be equal to 0 at any given time, all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs A and B
- The circuit is disabled when E is equal to 1.

3-8 line decoder (active-HIGH)

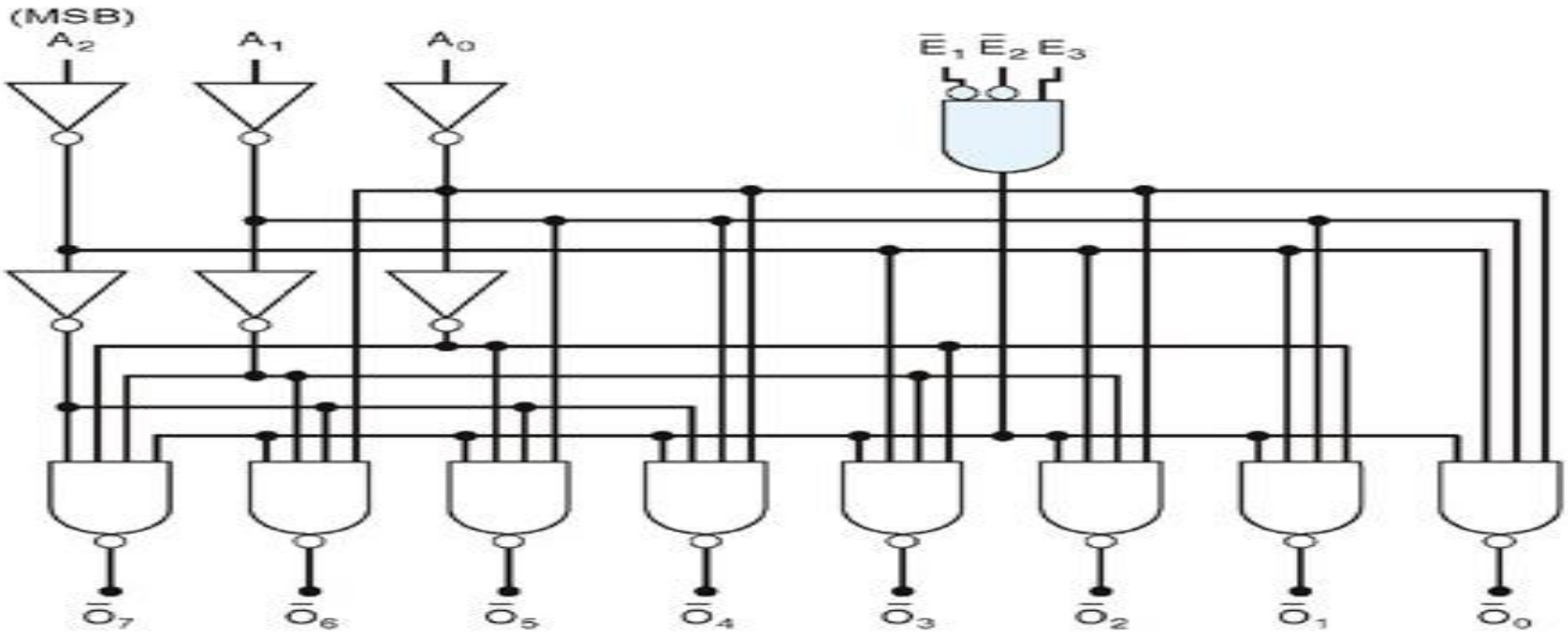


C	B	A	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- This decoder can be referred to in several ways. It can be called a 3-line-to- 8-line decoder, because it has three input lines and eight output lines.

- It could also be called a binary-octal decoder or converters because it takes a three bit binary input code and activates the one of the eight outputs corresponding to that code. It is also referred to as a 1-of-8 decoder, because only 1 of the 8 outputs is activated at one time.

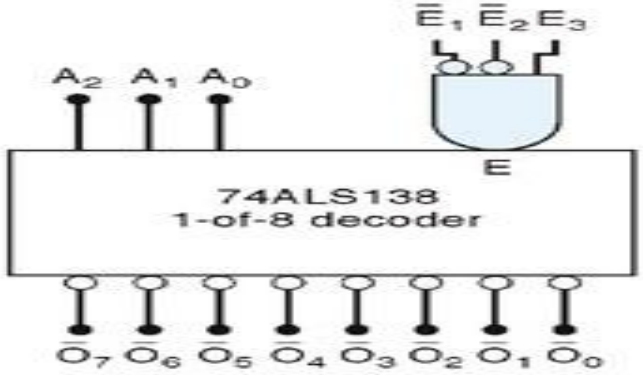
Logic diagram of 74138 (Example of a 3–Bit Decoder)



(a)

$\bar{E}_1$	$\bar{E}_2$	E_3	Outputs
0	0	1	Respond to input code $A_2A_1A_0$
1	X	X	Disabled — all HIGH
X	1	X	Disabled — all HIGH
X	X	0	Disabled — all HIGH

(b)



(c)

Truth table of 74138 (Example of a 3– 8 Bit Decoder) active-LOW

Inputs						Outputs							
Enables			2^2	2^1	2^0	Active–*LOW*							
E_3	$\overline{E}_1$	$\overline{E}_2$	A_2	A_1	A_0	$\overline{O}_7$	$\overline{O}_6$	$\overline{O}_5$	$\overline{O}_4$	$\overline{O}_3$	$\overline{O}_2$	$\overline{O}_1$	$\overline{O}_0$
X	X	H	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H	**L**
H	L	L	L	L	H	H	H	H	H	H	H	**L**	H
H	L	L	L	H	L	H	H	H	H	H	**L**	H	H
H	L	L	L	H	H	H	H	H	H	**L**	H	H	H
H	L	L	H	L	L	H	H	H	**L**	H	H	H	H
H	L	L	H	L	H	H	H	**L**	H	H	H	H	H
H	L	L	H	H	L	H	**L**	H	H	H	H	H	H
H	L	L	H	H	H	**L**	H	H	H	H	H	H	H

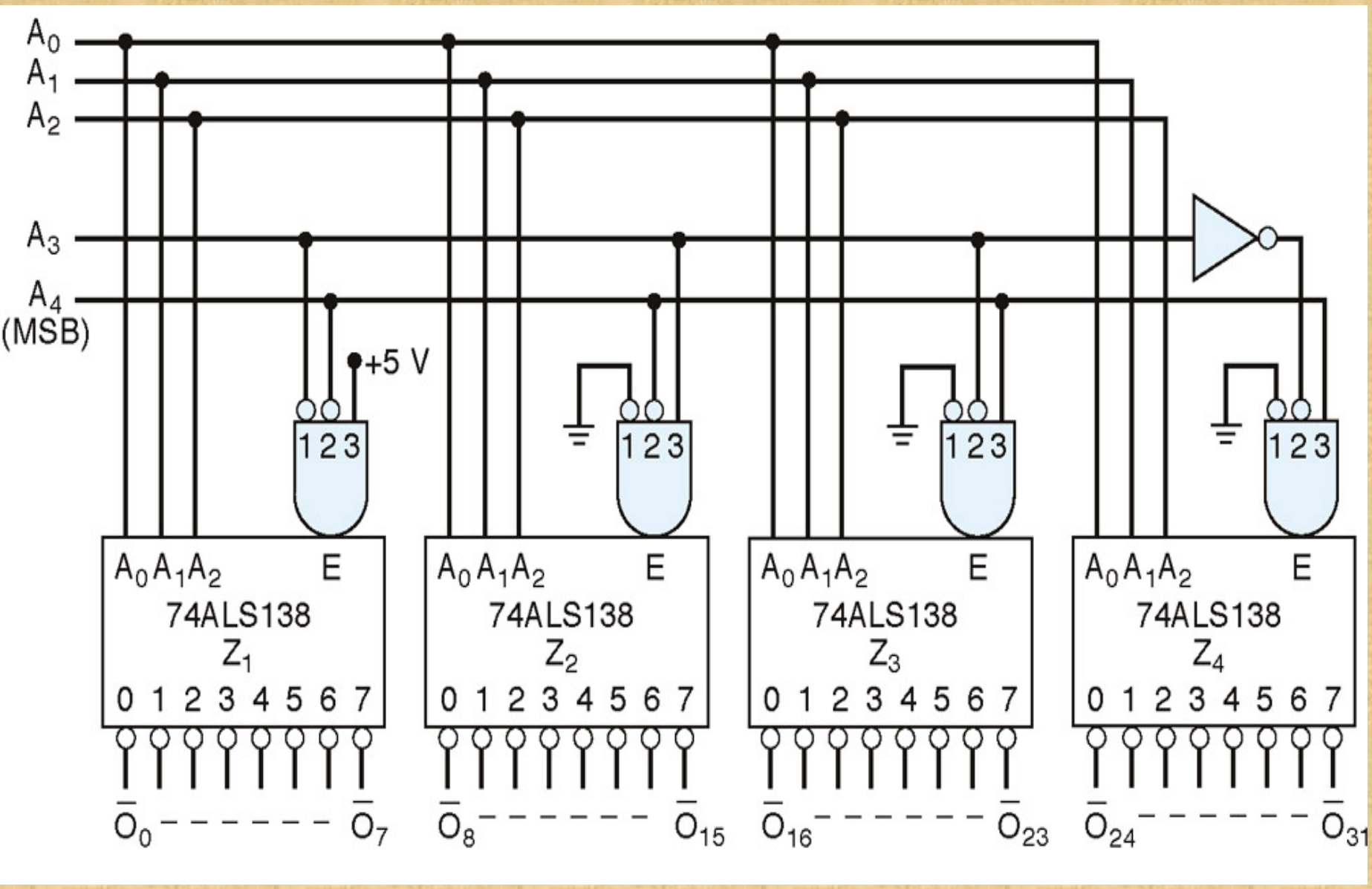
74138 (Example of a 3– 8 Bit Decoder)

- There is an enable function on this device, a *LOW* level on each input E'_1, and E'_2, and a *HIGH* level on input E_3, is required in order to make the enable gate output *HIGH*.

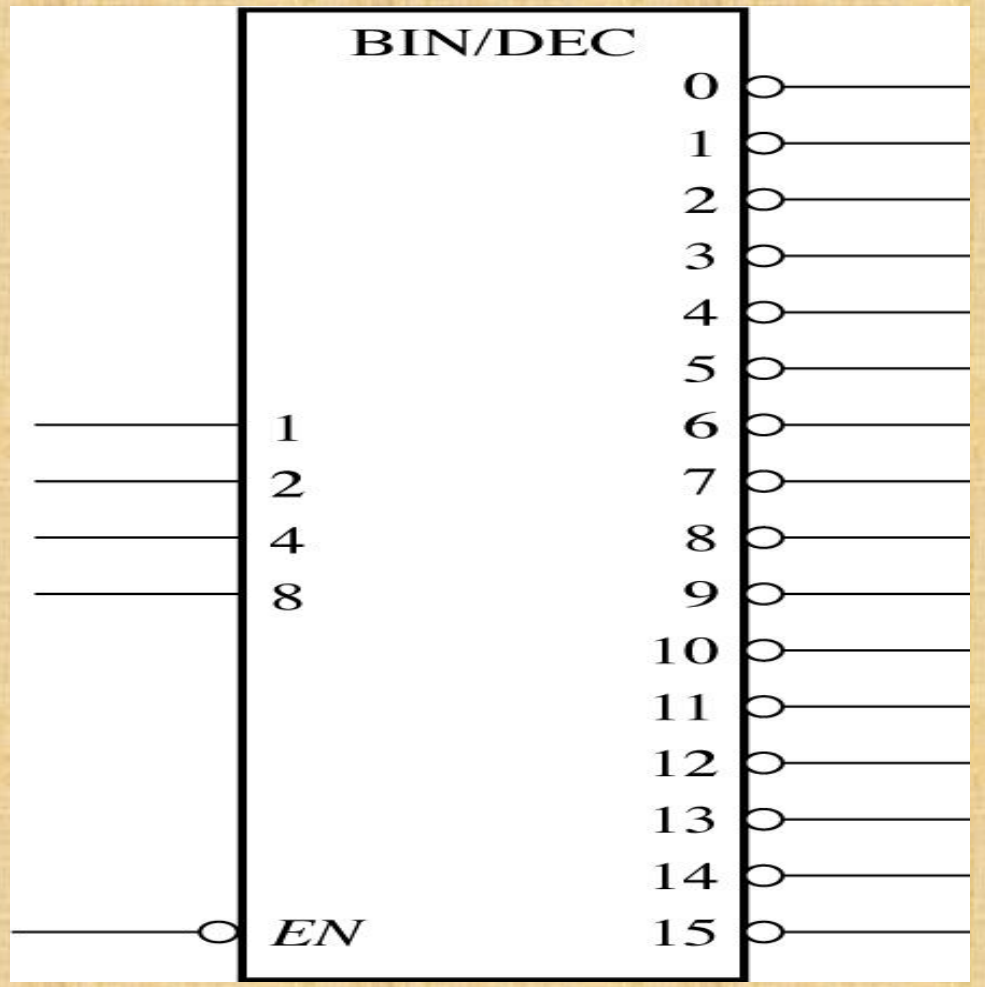
- The enable is connected to an input of each NAND gate in the decoder, so it must be *HIGH* for the NAND gate to be enabled.

- If the enable gate is not activated then all eight decoder outputs will be *HIGH* regardless of the states of the three input variables A_0, A_1, and A_2 .

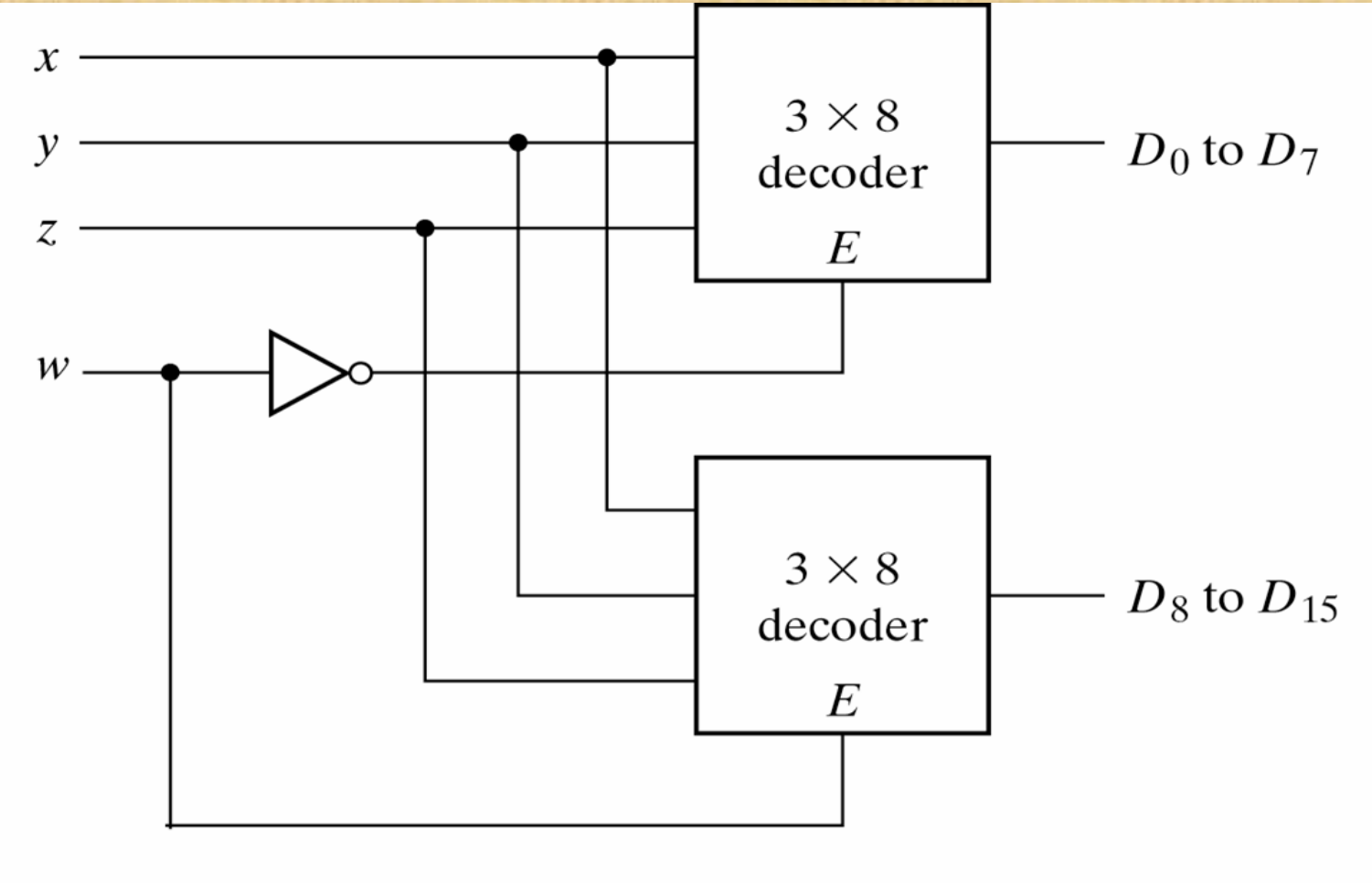
Example of a 5 to 32 Bit Decoder



Logic symbol for a 4-line-to-16-line (1-of-16) decoder . 74HC154



4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders (1)...

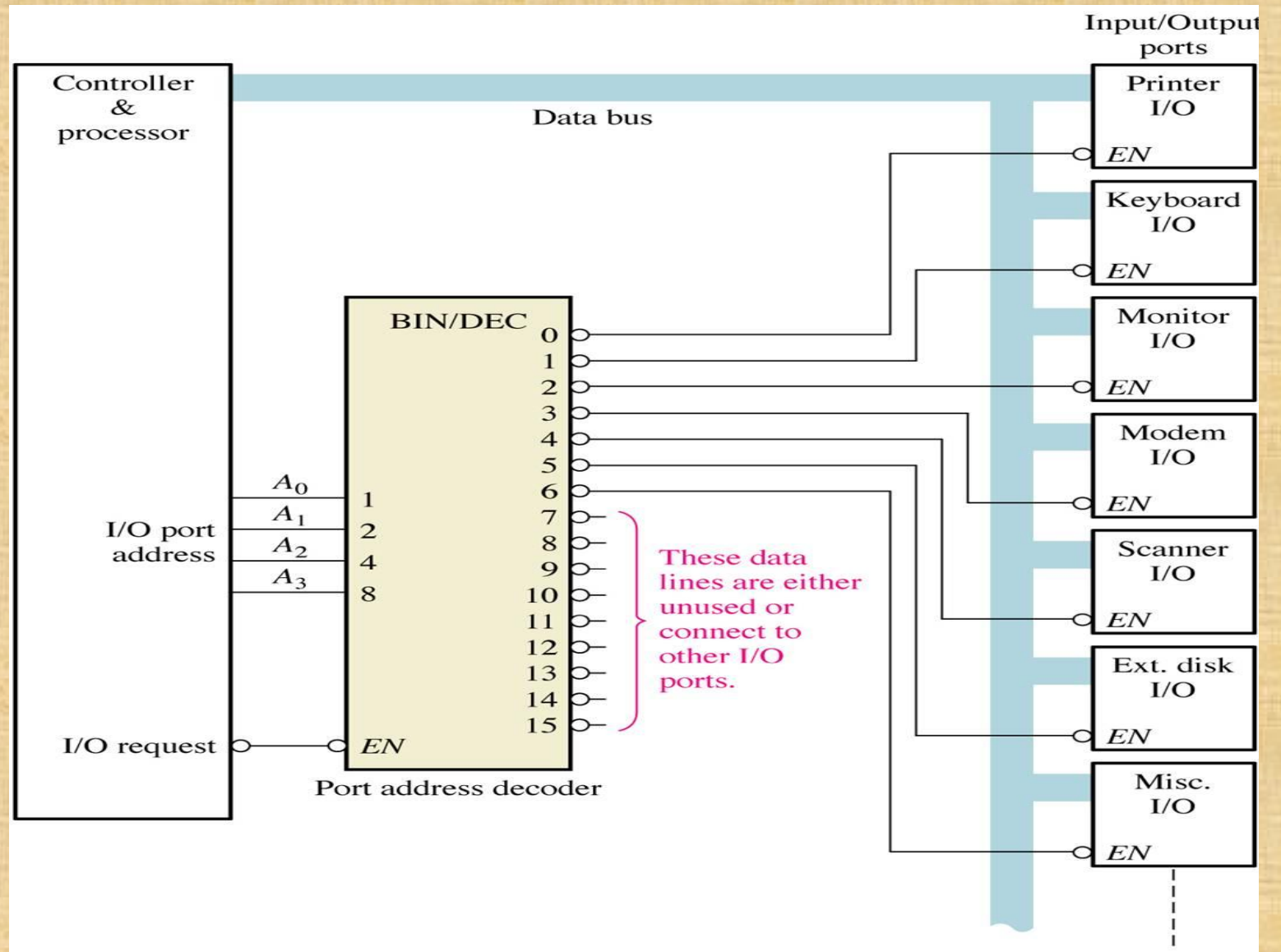


4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders (2)

- When w=0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's , and the top eight outputs generate min-terms 0000 to 0111.
- When w=1, the enable conditions are reversed. The bottom decoder outputs generate min-terms 1000 to 1111, while the outputs of the top decoder are all 0's.

Application example

A simplified computer I/O port system with a port address decoder with only four address lines shown.



- Decoders are used in many types of applications. One example is in computers for I/O selection as in previous slide

- Computer must communicate with a variety of external devices called peripherals by sending and/or receiving data through what is known as input/output (I/O) ports

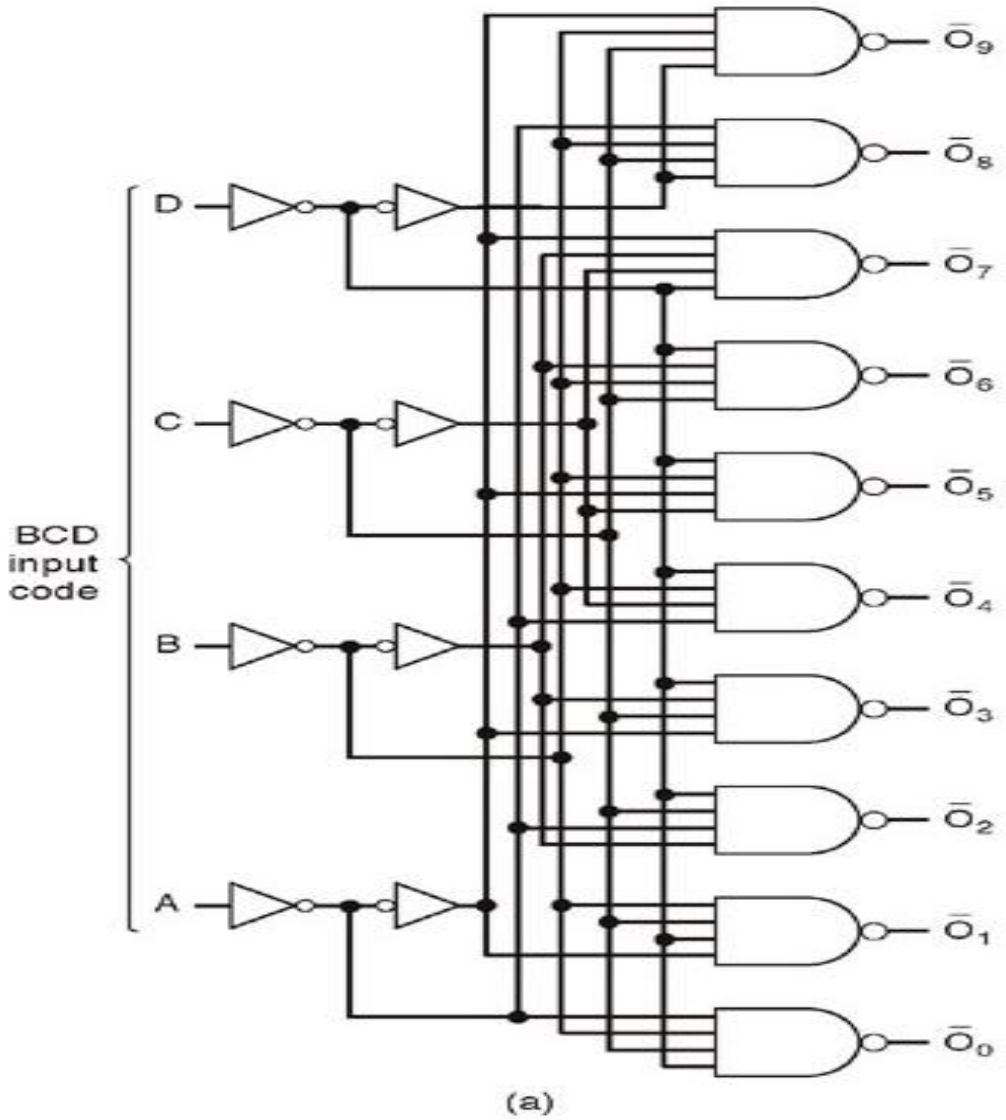
- Each I/O port has a number, called an address, which uniquely identifies it. When the computer wants to communicate with a particular device, it issues the appropriate address code for the I/O port to which that particular device is connected . The binary port address is decoded and appropriate decoder output is activated to enable the I/O port

- Binary data are transferred within the computer on a data bus, which is a set of parallel lines

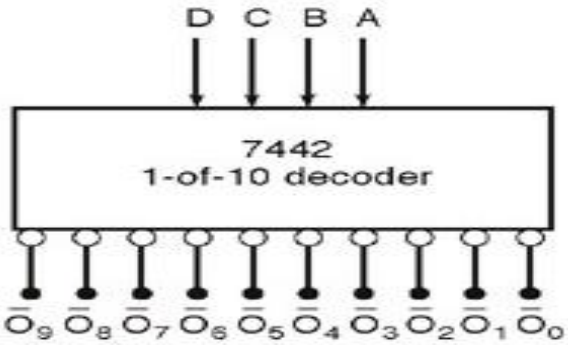
BCD -to- Decimal decoders

- The BCD- to-decimal decoder converts each BCD code into one of Ten Positionable decimal digit indications. It is frequently referred as a 4-line -to- 10 line decoder
- The method of implementation is that only ten decoding gates are required because the BCD code represents only the ten decimal digits 0 through 9.
- Each of these decoding functions is implemented with NAND gates to provide active -LOW outputs. If an active HIGH output is required, AND gates are used for decoding

Logic diagram of BCD - decimal decoder (Active LOW output)



(a)

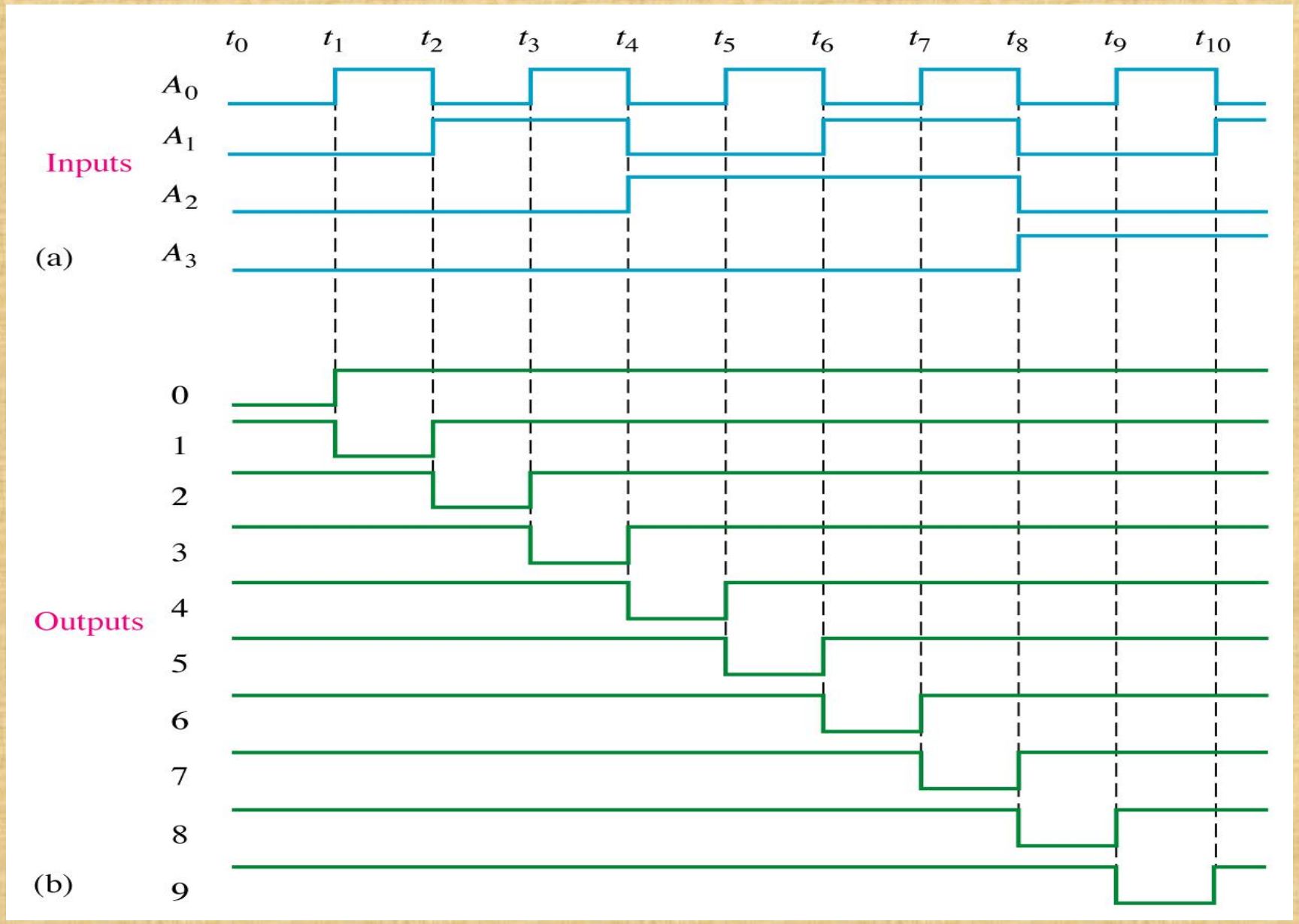


(b)

Inputs				
D	C	B	A	Active Output
L	L	L	L	$\bar{O}_0$
L	L	L	H	$\bar{O}_1$
L	L	H	L	$\bar{O}_2$
L	L	H	H	$\bar{O}_3$
L	H	L	L	$\bar{O}_4$
L	H	L	H	$\bar{O}_5$
L	H	H	L	$\bar{O}_6$
L	H	H	H	$\bar{O}_7$
H	L	L	L	$\bar{O}_8$
H	L	L	H	$\bar{O}_9$
H	L	H	L	None
H	L	H	H	None
H	H	L	L	None
H	H	L	H	None
H	H	H	L	None
H	H	H	H	None

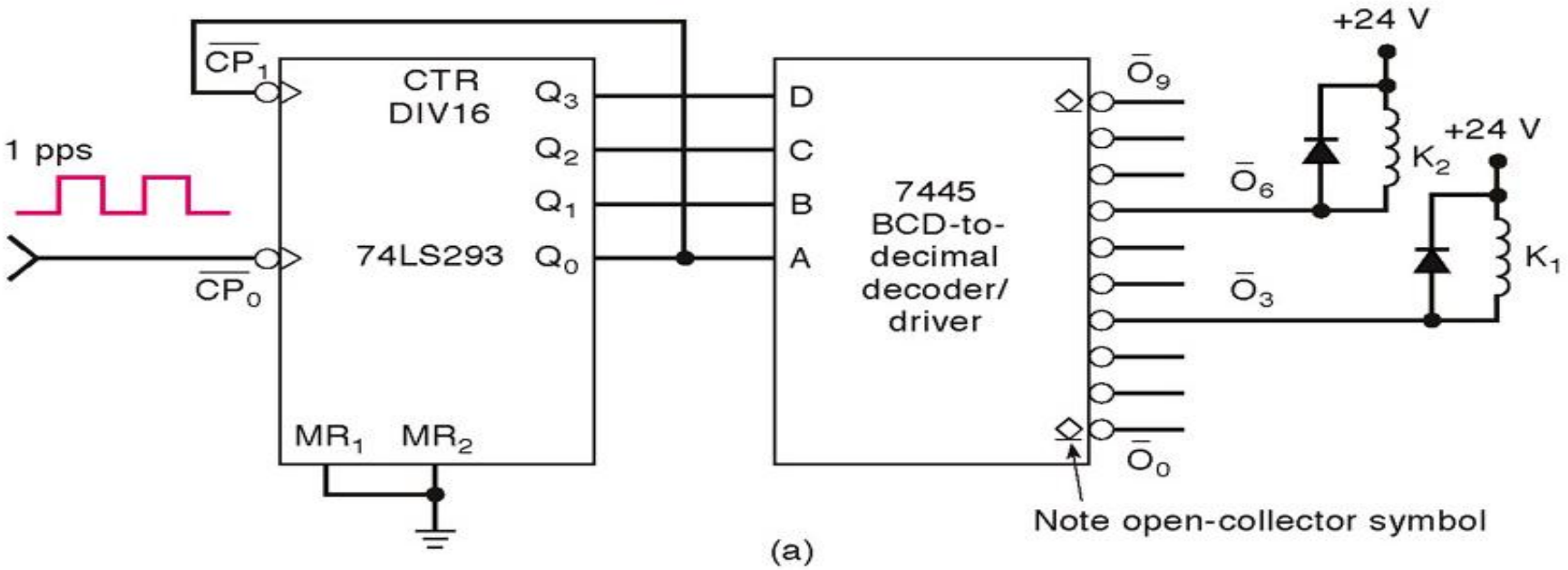
H = HIGH Voltage Level
L = LOW Voltage Level

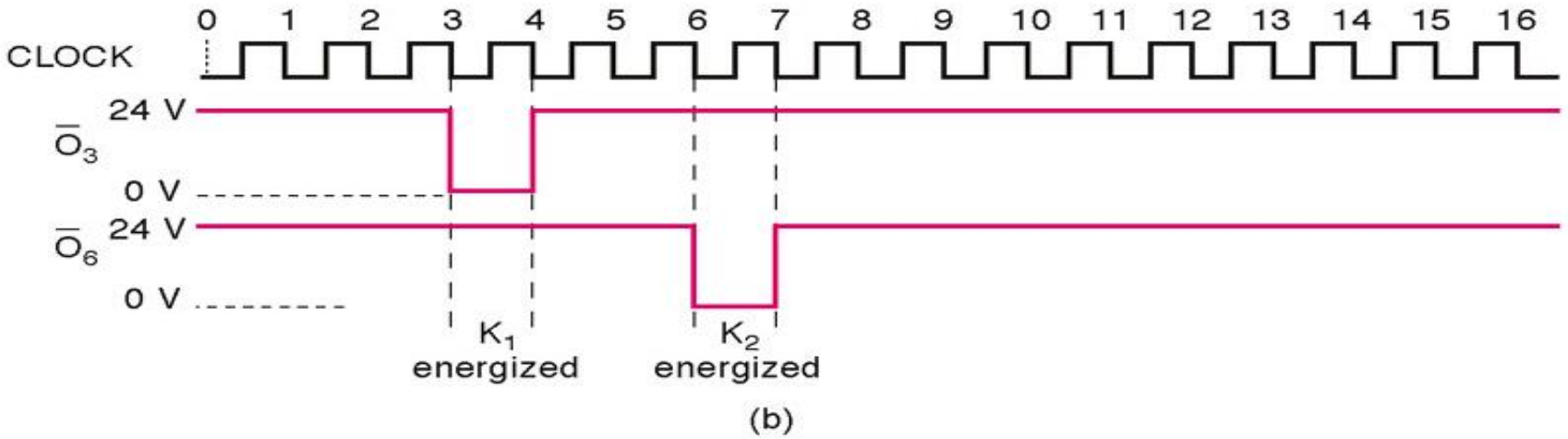
(c)



Output Waveform for BCD Decoder

A Decoder Application - Counter -decoder combination used to provide timing and sequential operations (1)...

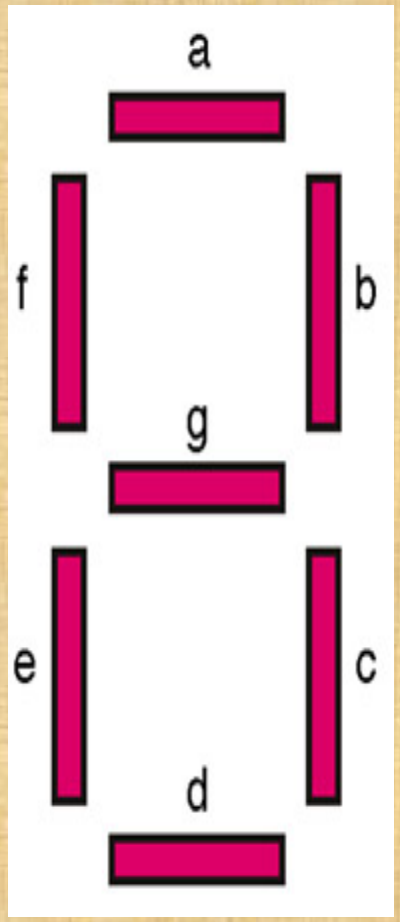




A Decoder Application - Counter -decoder combination used to provide timing and sequential operations (1)...

- Decoders are used whenever an output or a group of outputs is to be activated only on the occurrence of specific combination of input levels. These input levels are often provided by the outputs of a counter or register.
- When the decoder inputs come from a counter that is being continually pulsed, the decoder outputs will be activated sequentially, and there can be used as timing or sequencing signals to turn device on or off at specific times

BCD-7segment decoders/drivers

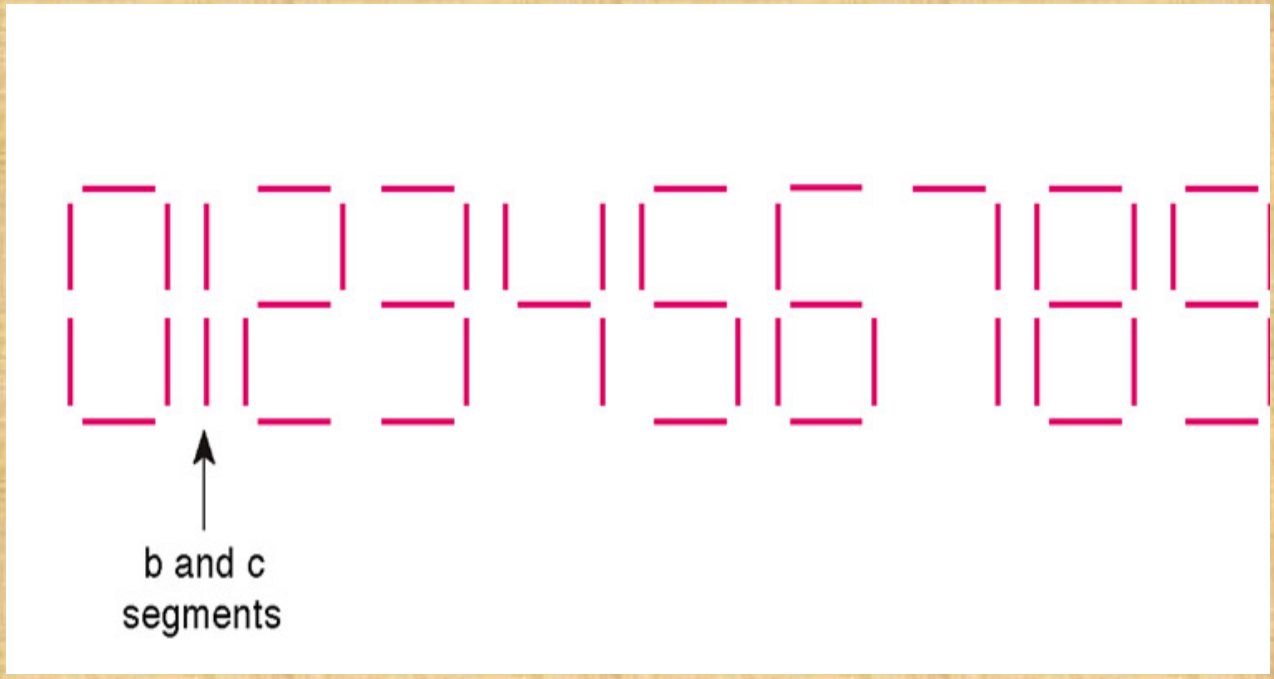


Most digital equipment has some means for displaying information in a form that can be understood by the user. This information is often numerical data but also be alphanumeric.

One of the simplest and most popular methods for displaying numerical digits uses a 7-segment configuration to form digital characters 0 to 9 and some times the hex characters A to F

One common arrangements uses light-emitting diodes (LED's) for each segment. By controlling the current thru each LED, some segments will be light and others will be dark so that desired character pattern will be generated

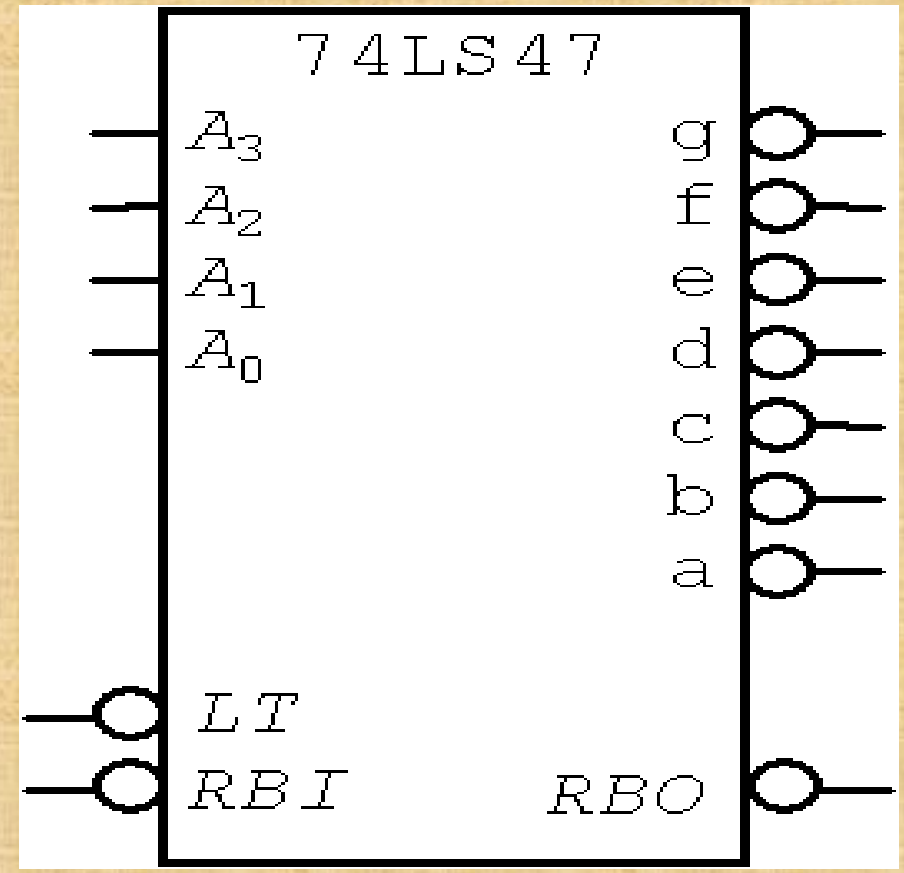
Figure shows the segment pattern that are used to display the various digits. For example, to display a “6” the segments a,c,d,e,f and g are made bright while segment b is dark

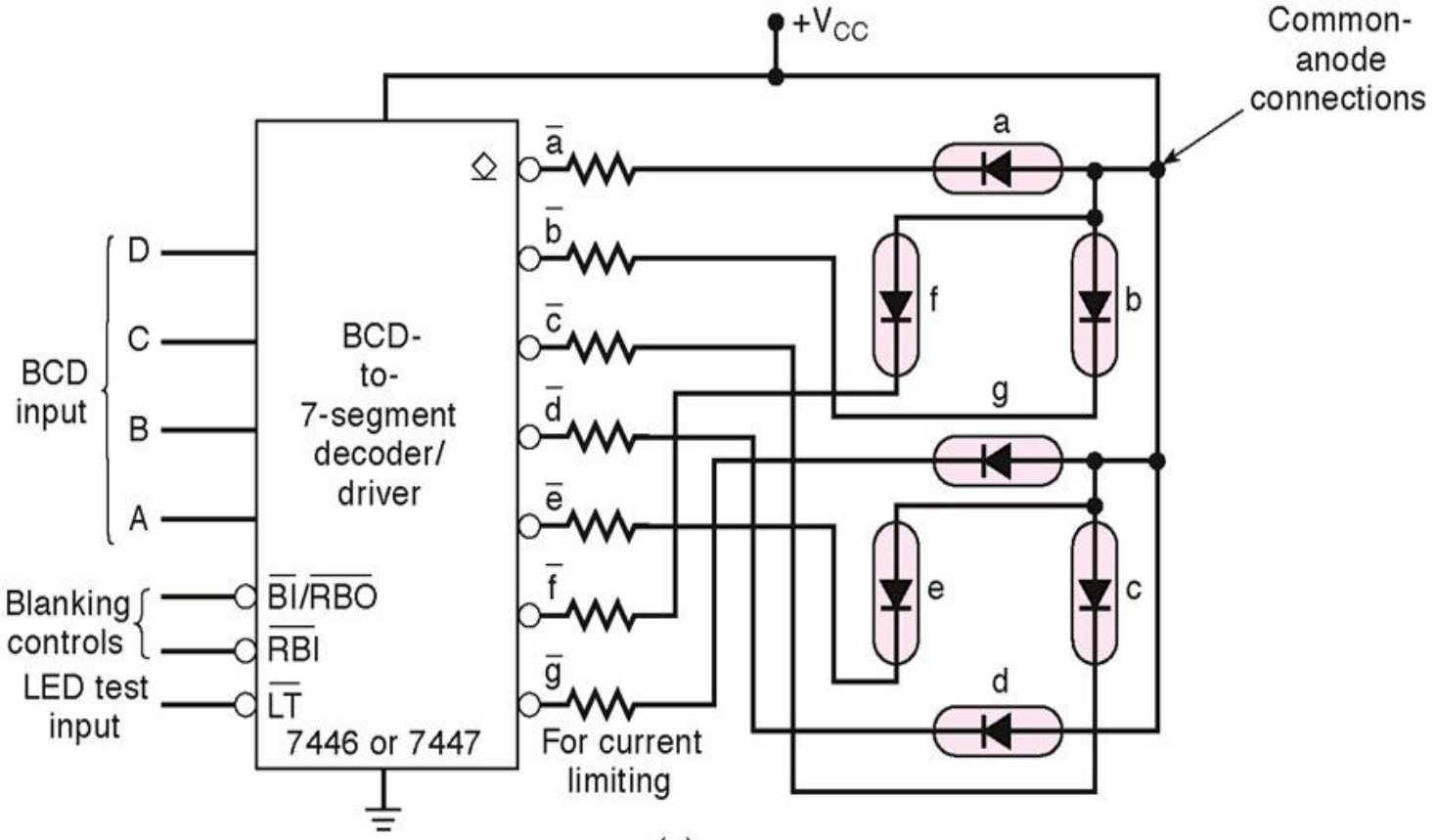


7-segment decoder

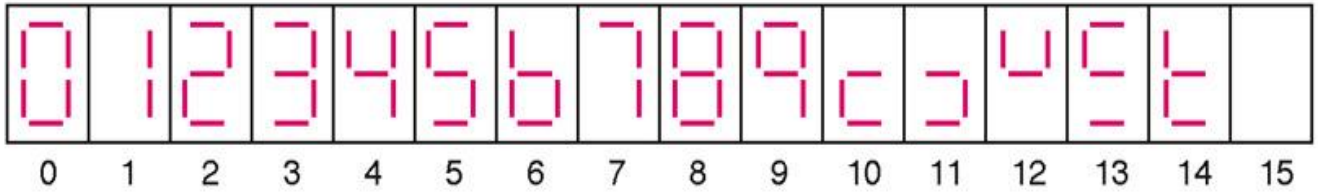
- A BCD-7 segment decoder/driver is used to take four-bit BCD input and provide the outputs that will pass current through the appropriate segments to display the decimal digit.
- The logic for this decoder is more complicated than the logic of decoders of earlier case, because each output is activated for more than one combination of inputs.

74LS47 (BCD–to–Seven–Segment Decoder)




+V_{CC}
Common-anode connections
BCD input
D
C
B
A
BCD-to-7-segment decoder/driver
Blanking controls
$\overline{BI}/\overline{RBO}$
$\overline{RBI}$
LED test input
$\overline{LT}$
7446 or 7447
$\overline{a}$
$\overline{b}$
$\overline{c}$
$\overline{d}$
$\overline{e}$
$\overline{f}$
$\overline{g}$
For current limiting
a
b
c
d
e
f
g

(a)


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

(b)

Lamp Test (LT)

- When $\overline{LT}$ = Low, $\overline{BI/RBO}$ = HIGH then all of the 7 segments in display are turned zero, LT is used to verify that no segments are burned out

Zero Suppression ($\overline{BI}$, $\overline{RBI}$, $\overline{RBO}$)

- Zero suppression is a feature used for multi digit displays to blank out unnecessary zeros.

Example:
In a 6-digit display the number 6.4 may be displayed as 006.400 if the zeros are not blanked out

- **Leading Zero Suppression**

Blanking the zeros at the front of a numbers

- **Trailing Zero Suppression**

Blanking the zeros at the back of the number

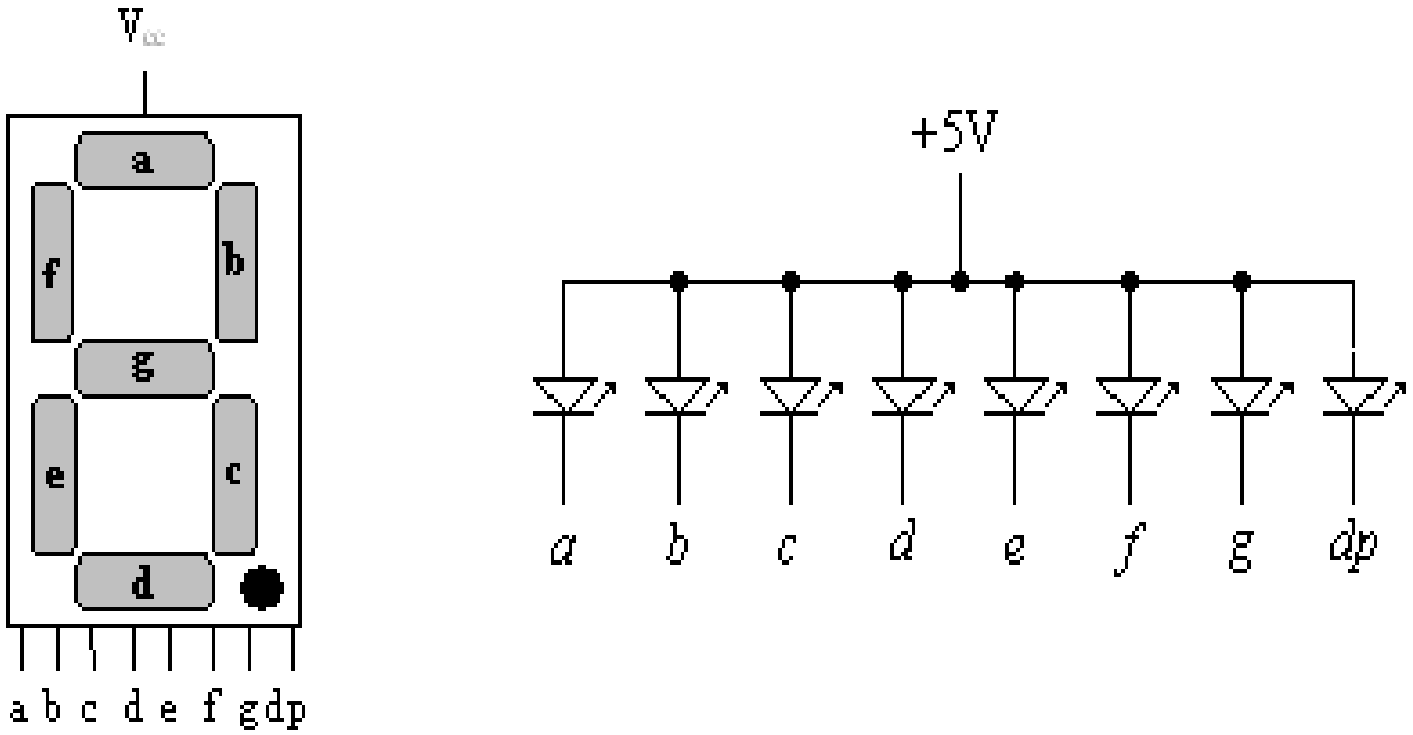
Only nonessential zeros are blanked, the number 030.080 will be displayed as 30.08 (the essential zeros remain)

7-segment display

- There are two types of 7–segment LED displays;
- A) common - anode
- B) common – cathode

Common Anode

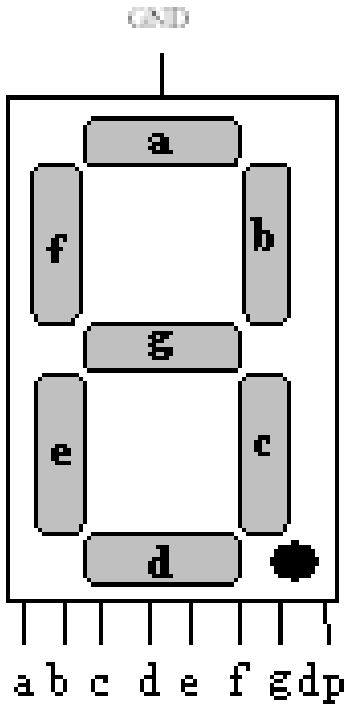
In common–anode, the anode of all of the LEDs are tied together to positive of the power supply (V_{cc}) as shown

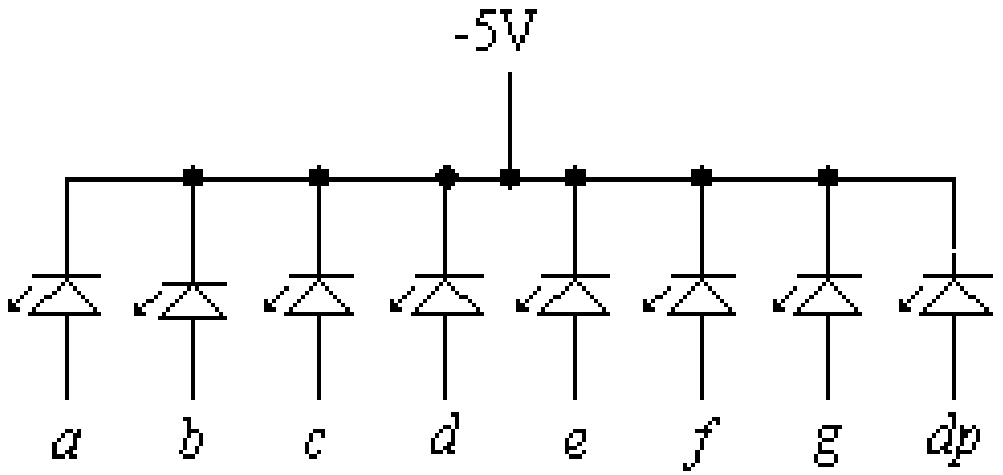


Common–anode 7–segment LED display.

Common Cathode

- In common–cathode, the cathode of all of the LEDs are tied together to ground as shown.





Common–cathode 7–segment LED display.

Combinational Logic Circuit Implementation using a Decoder

- Any combinational logic circuit with n inputs and m outputs can be implemented with an n-to-2n-line decoder and m OR gates.
- Procedure:
  - Express the given Boolean function in sum of min-terms
  - Choose a decoder to generate all the min-terms of the input variables.
  - Select the inputs to each OR gate from the decoder outputs according to the list of min-term for each function.

Combinational Logic Circuit Implementation using a Decoder - An example (1)

- From the truth table of the 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 functions can be expressed in sum of min-terms.

$S(x,y,z) = \Sigma m(1,2,4,7)$

$C(x,y,z) = \Sigma m(3,5,6,7)$

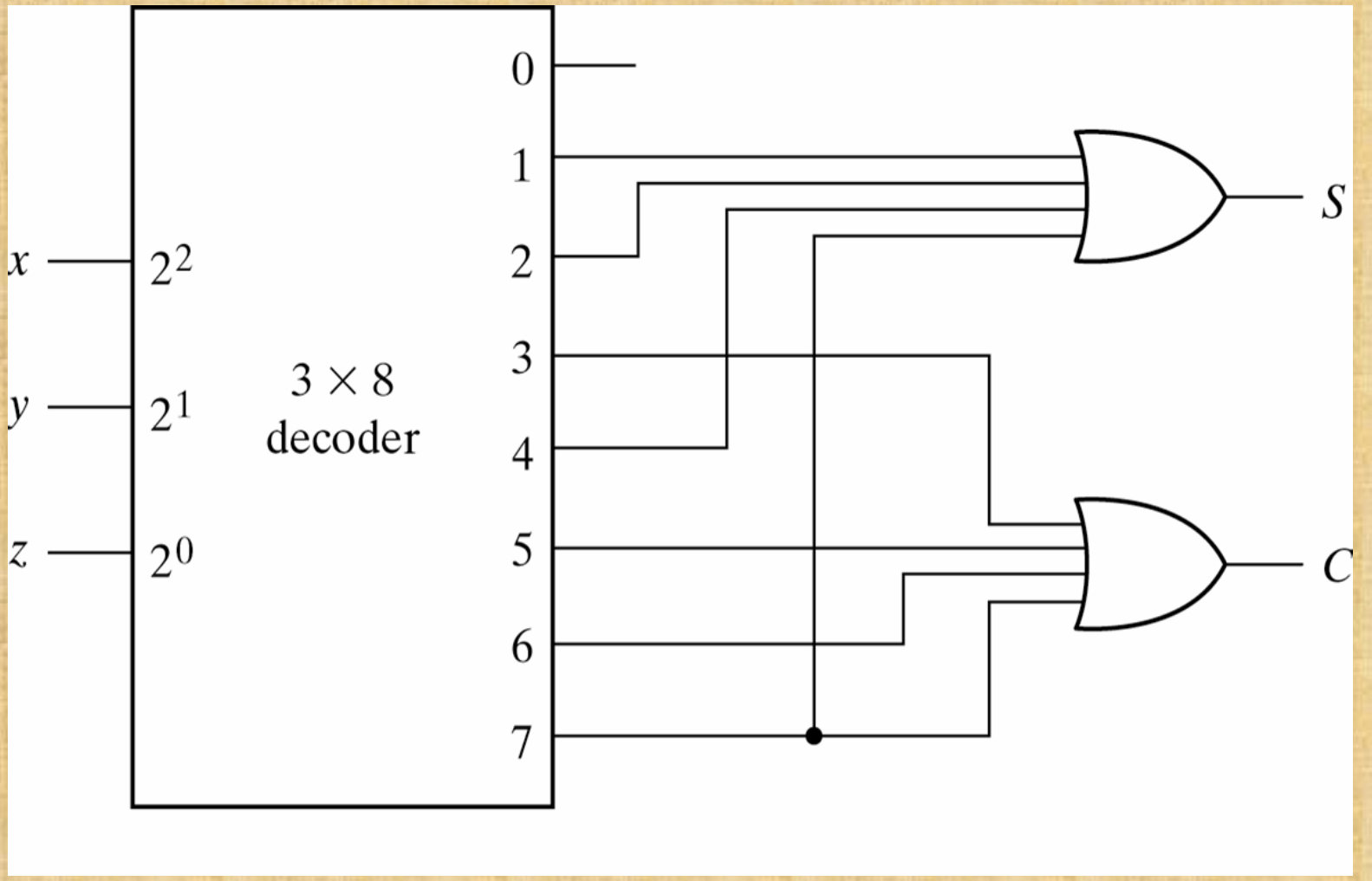
where Σ indicates sum, m indicates min-term and the number in brackets indicate the decimal equivalent

Combinational Logic Circuit Implementation using a Decoder - An example (2)

Since there are three inputs and a total of eight min-terms, we need a 3-to-8 line decoder.

- The decoder generates the eight min-terms for x,y,z
- The OR gate for output S forms the logical sum of min-terms 1,2,4, and 7.
- The OR gates for output C forms the logical sum of min-terms 3,5,6, and 7

Combinational Logic Circuit Implementation using a Decoder - example (3)



Implementation of a Full Adder with a Decoder

Encoders

Encoder

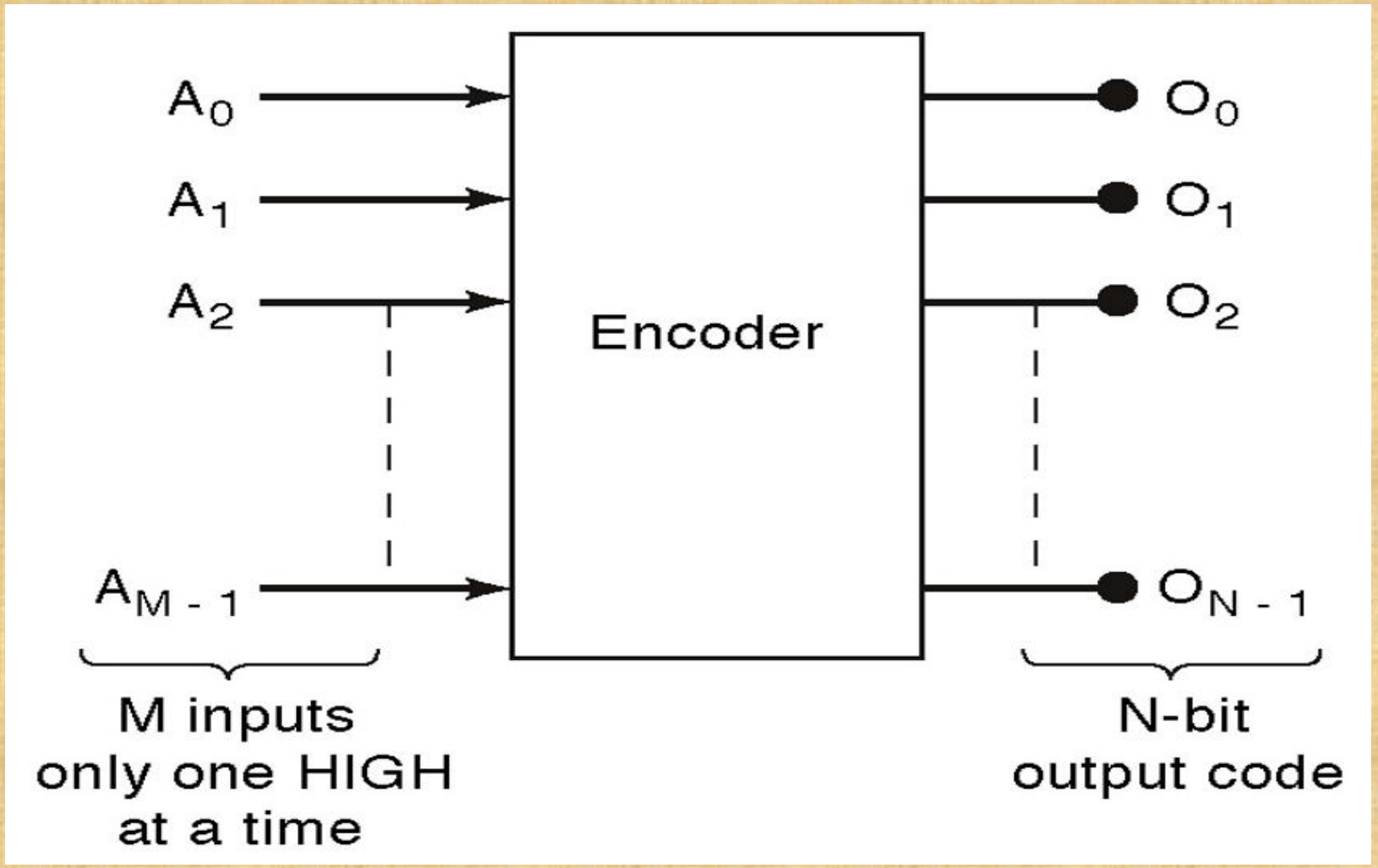
- An encoder is a combinational logic circuit that essentially performs a "reverse" of decoder functions.
- An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting from familiar symbols or numbers to a coded format is called ***encoding***.

•Most decoders accept an input code and produce a HIGH

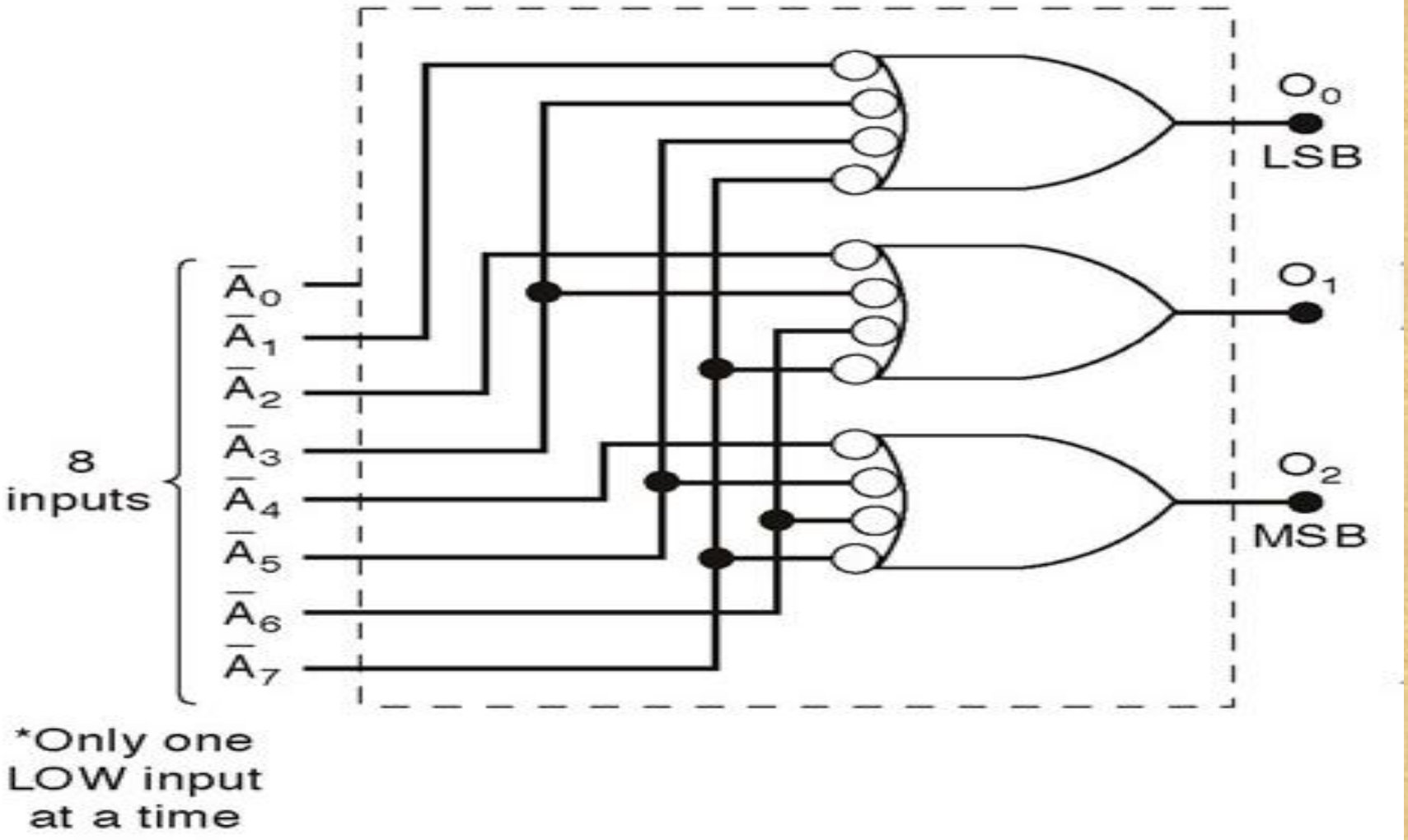
•(or a LOW) at one and only one output line. In otherworlds , a decoder identifies, recognizes, or detects a particular code. The opposite of this decoding process is called encoding and is performed by a logic circuit called an encoder.

•An encoder has a number of input lines, only one of which input is activated at a given time and produces an N-bit output code,depending on which input is activated.

General encoder diagram



Logic circuit for octal-to binary encoder [8-line-3-line]



Truth table for octal-to binary encoder [8-line- 3-line]

Inputs								Outputs		
$\bar{A}_0$	$\bar{A}_1$	$\bar{A}_2$	$\bar{A}_3$	$\bar{A}_4$	$\bar{A}_5$	$\bar{A}_6$	$\bar{A}_7$	O_2	O_1	O_0
X	1	1	1	1	1	1	1	0	0	0
X	0	1	1	1	1	1	1	0	0	1
X	1	0	1	1	1	1	1	0	1	0
X	1	1	0	1	1	1	1	0	1	1
X	1	1	1	0	1	1	1	1	0	0
X	1	1	1	1	0	1	1	1	0	1
X	1	1	1	1	1	0	1	1	1	0
X	1	1	1	1	1	1	0	1	1	1

A low at any single input will produce the output binary code corresponding to that input. For instance , a low at A_3' will produce O_2 =0, O_1=1 and O_0 =1, which is binary code for 3. A_0' is not connected to the logic gates because the encoder outputs always be normally at 0000 when none of the inputs is LOW

Design of 4-input Priority Encoder (4-line-to 2 line priority encoder) (1)…

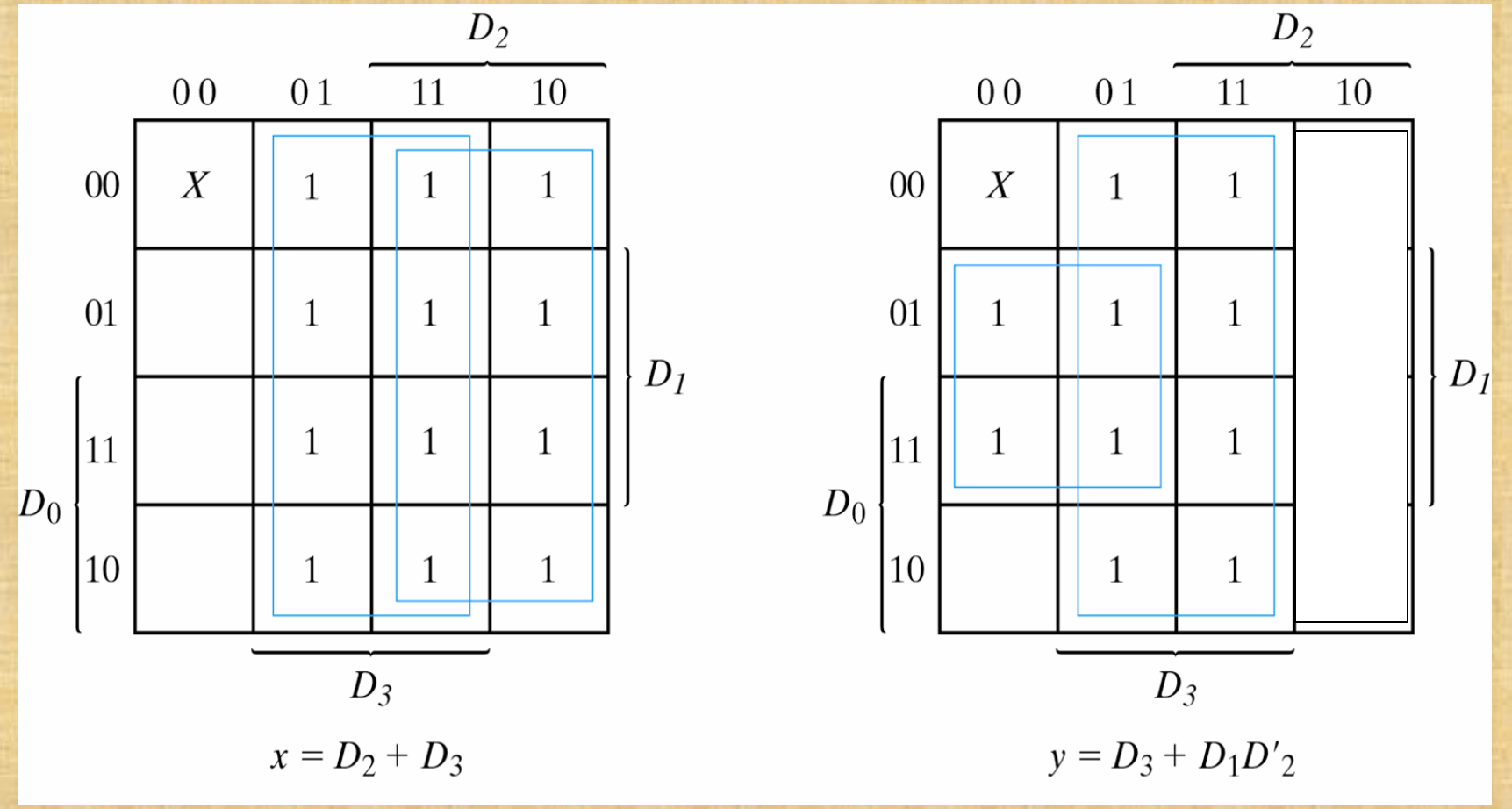
- A priority encoder is an encoder that includes the priority function
- If two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- **Truth Table of a 4-input Priority Encoder:**

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	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

Design of 4-input Priority Encoder (4-line-to 2 line priority encoder) (2)...

- In addition to two outputs x, and y, the truth table has a third output designated by V, which is a valid bit indicator that is set 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.
- X's in the output column indicate don't care conditions, the X's in the input columns are useful for representing a truth table in condensed form.
- The higher the subscript number, the higher the priority of the input. Input D3 has the highest priority, so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3)

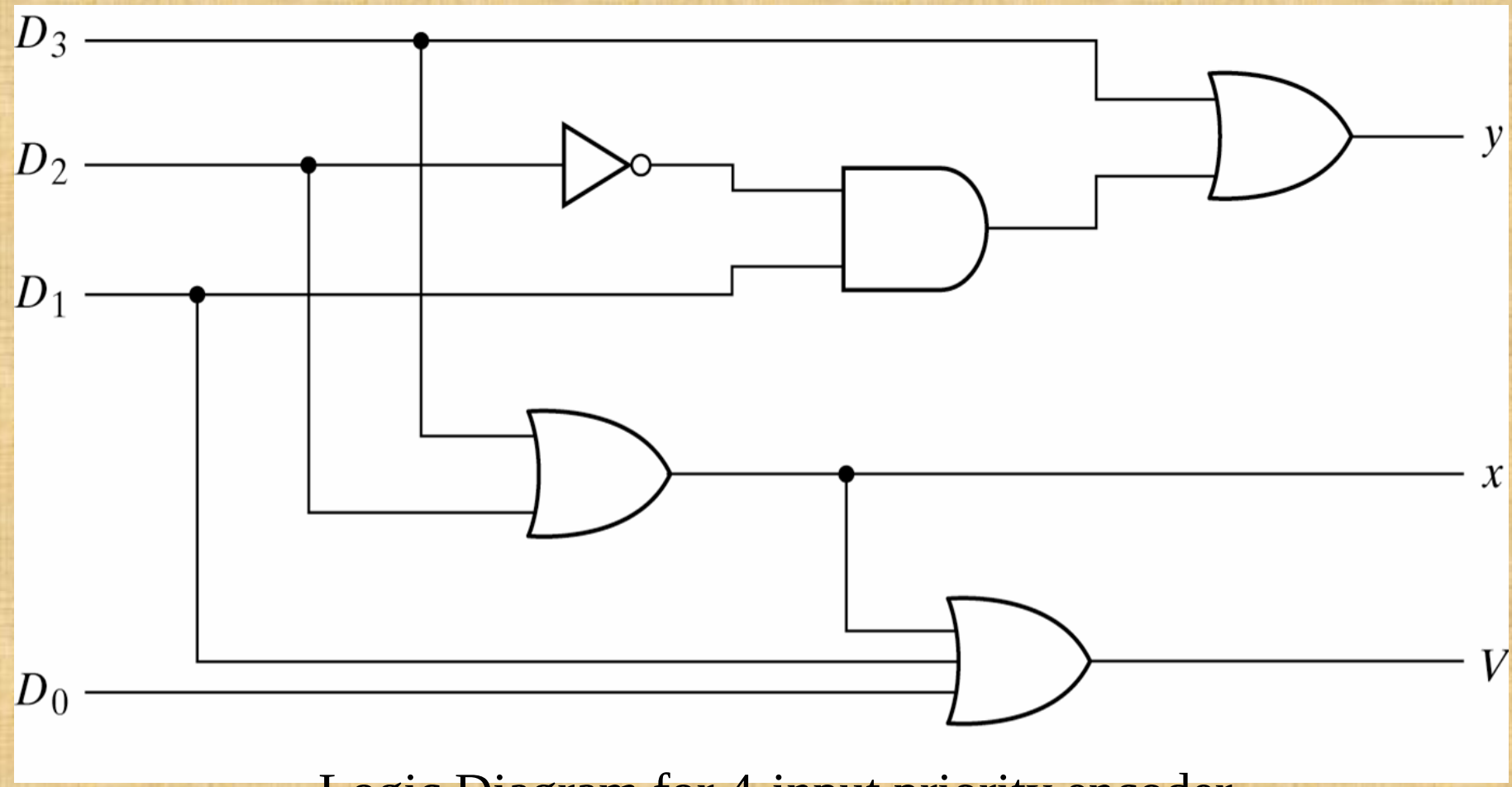
Design of 4-input Priority Encoder (4-line-to 2 line priority encoder) (3)…



$V=D_0+D_1+D_2+D_3$

K-Maps for 4-input Priority Encoder

Design of 4-input Priority Encoder (4-line-to 2 line priority encoder) (4)

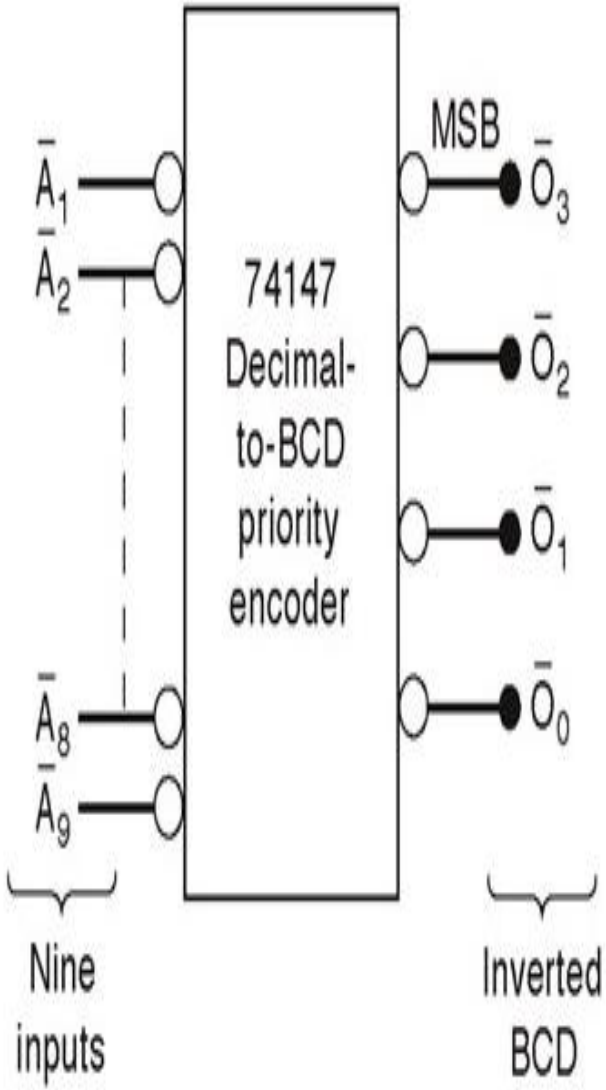


Logic Diagram for 4-input priority encoder

Decimal-BCD priority encoder

- Encoder will produce a BCD output corresponding to the highest-order decimal digit input that is active and will ignore any other lower order active inputs.
- For instance if the input 6 and the 3 are active, the output will be 1001, which is the inverse value of BCD output 0110 (which represents decimal 6)

74147 decimal-BCD priority encoder

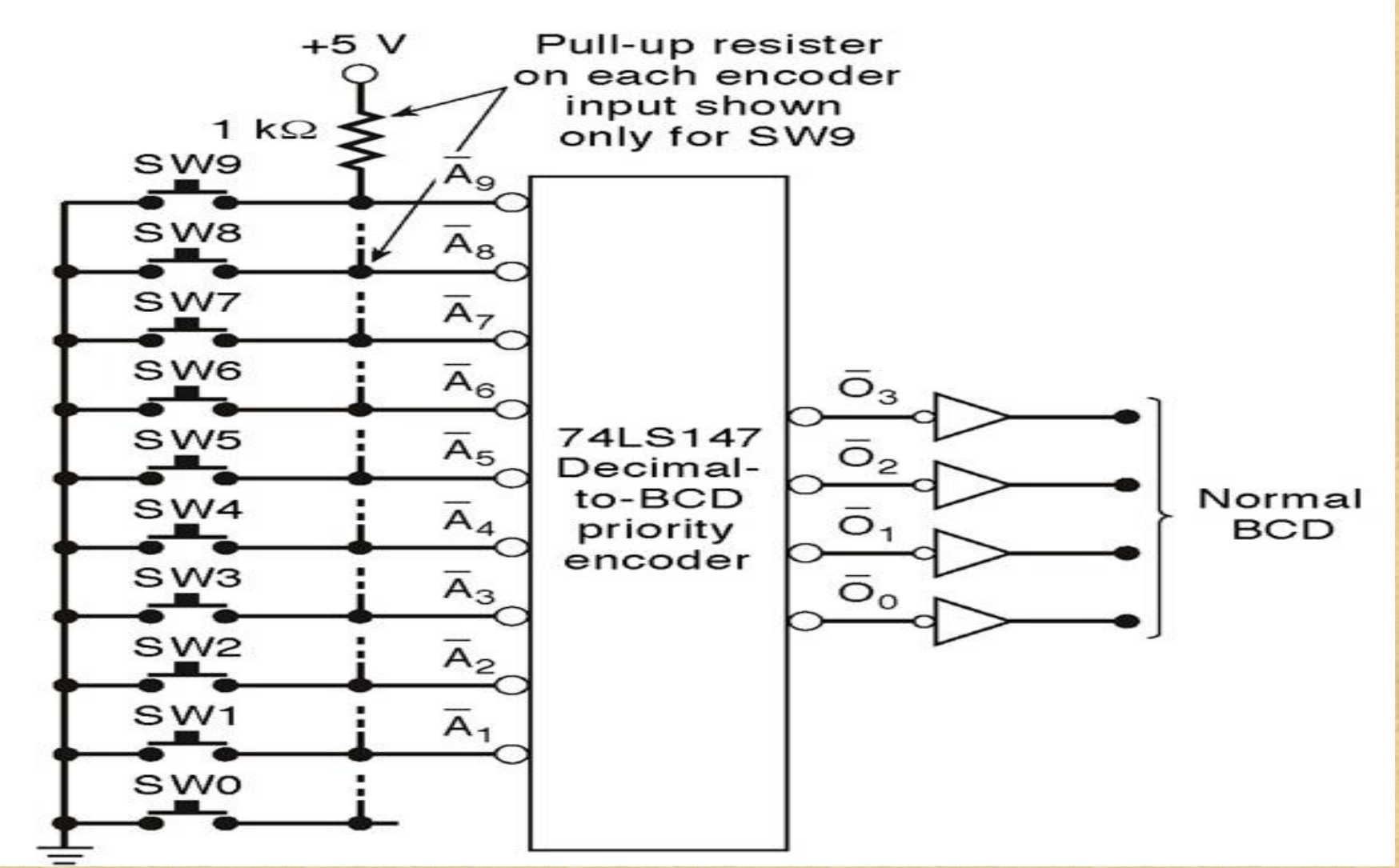


$\bar{A}_1$	$\bar{A}_2$	$\bar{A}_3$	$\bar{A}_4$	$\bar{A}_5$	$\bar{A}_6$	$\bar{A}_7$	$\bar{A}_8$	$\bar{A}_9$	$\bar{O}_3$	$\bar{O}_2$	$\bar{O}_1$	$\bar{O}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

X = either 0 or 1

When A9' is low, the output is 0110, which is inverse of 1001 (eq to 9 in BCD)

Decimal- BCD switch decoder



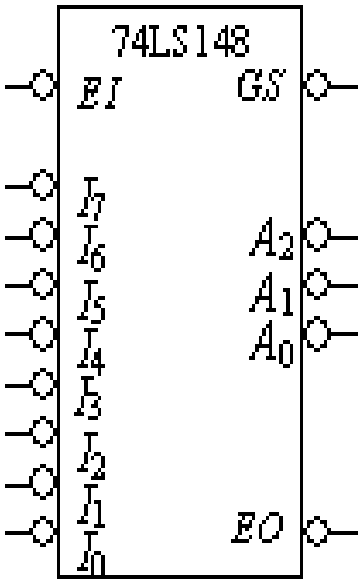
The output of the decoder are inversed to produce the normal BCD value

The Octal–to–Binary Priority Encoder- Example

- The 74LS148 is a priority encoder that has eight active *LOW* inputs and three active–*LOW* binary outputs

- To enable the device, the *EI* (enable input) must be *LOW*. It also has the *EO* (enable output) and *GS* (group signal output) for expansion purposes.

The Octal–to–Binary Encoder



Inputs									Outputs				
$\overline{EI}$	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	$\overline{GS}$	$\overline{A}_0$	$\overline{A}_1$	$\overline{A}_2$	$\overline{O}$
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	**L**	L	L	L	L	H
L	X	X	X	X	X	X	**L**	H	L	H	L	L	H
L	X	X	X	X	X	**L**	H	H	L	L	H	L	H
L	X	X	X	X	**L**	H	H	H	L	H	H	L	H
L	X	X	X	**L**	H	H	H	H	L	L	L	H	H
L	X	X	**L**	H	H	H	H	H	L	H	L	H	H
L	X	**L**	H	H	H	H	H	H	L	L	H	H	H
L	**L**	H	H	H	H	H	H	H	L	H	H	H	H

Logic symbol and truth table for 74LS148 8–line–to–3–line priority encoder.

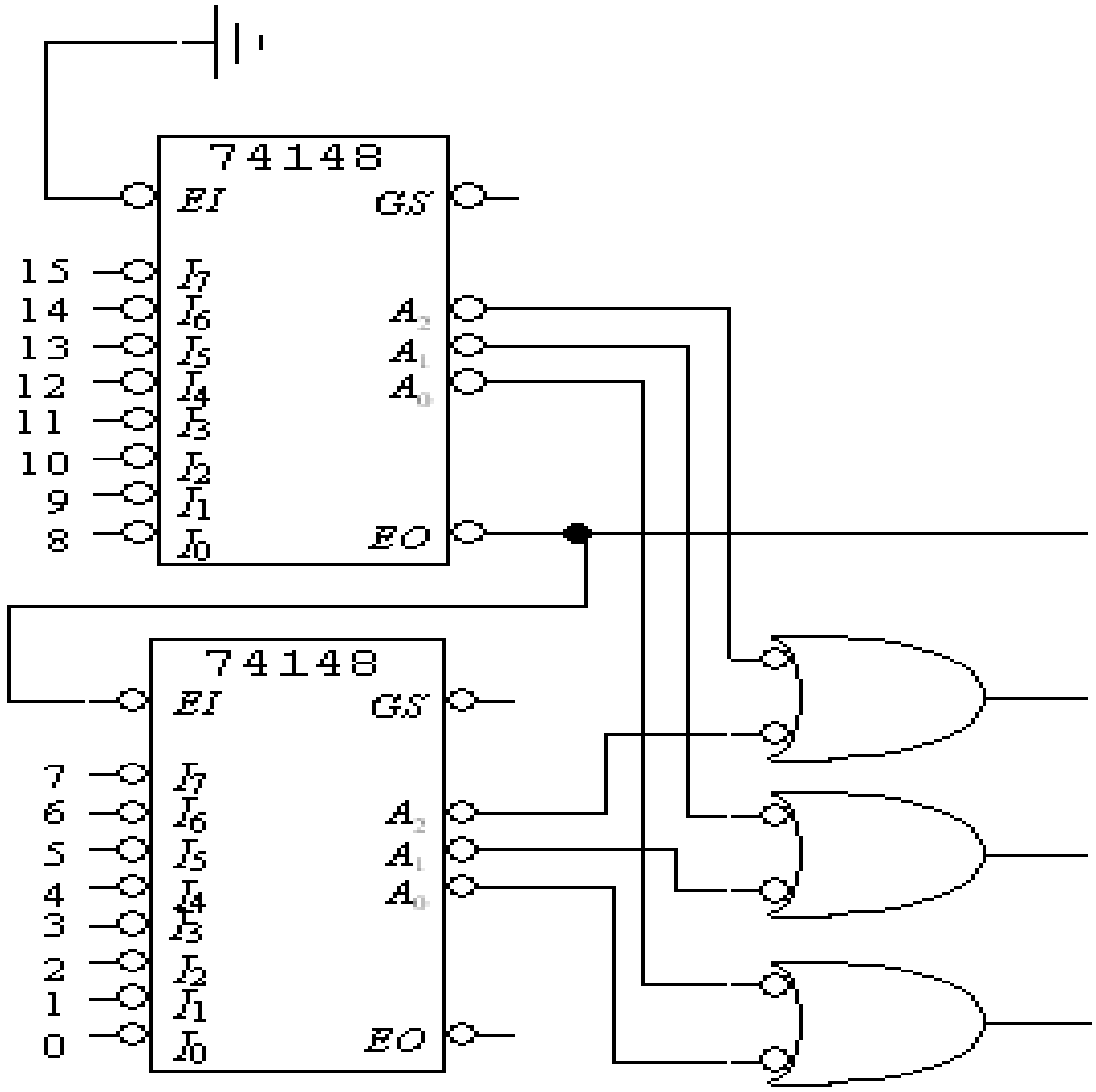
The Octal–to–Binary Encoder

- $\overline{EI}$ Active–*LOW* enable input, a *HIGH* on the input forces all outputs to their inactive state (*HIGH*).
- $\overline{EO}$ Active–*LOW* enable output, the output pin goes *LOW* when all inputs are inactive (*HIGH*) and is *LOW*.
- $\overline{GS}$ Active–*LOW* group signal output, this output pin goes *LOW* whenever any of the inputs are active (*LOW*) and is *LOW*.

The 16 –to–4 Encoder

The 74LS148 can be expanded to a 16–line–to–4–line encoder by connecting the *EO* of the higher–order encoder to the *EI* of the lower–order encoder and negative–ORing the corresponding binary outputs as shown

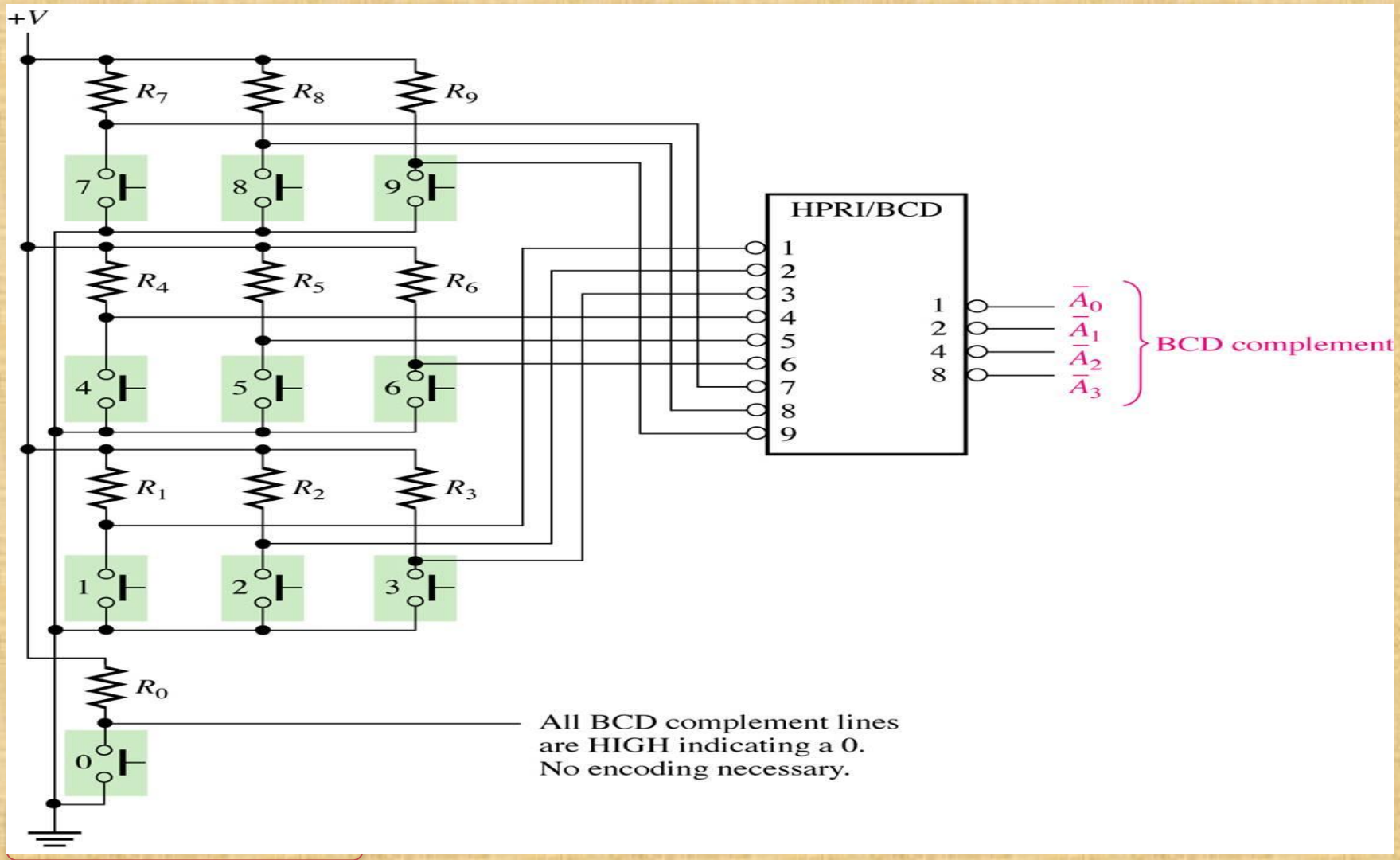
The 16 –to–4 Encoder



A 16–line–to–4–line encoder using 74LS148s and external logic.

Application example

A simplified keyboard encoder.



•When one of the keys is pressed, the decimal digit is encoded to the corresponding BCD code

•The keys are represented by 10 push-button switches, each with a **pull-up resistor** to V+. The pull-up resistor ensures that the line is HIGH when a key is not depressed.

•When a key is depressed, the line is connected to ground, and a LOW is applied to the corresponding encoder input.

•The zero key is not connected because the BCD output represents zero when none of the other keys is depressed

•The BCD complement output of the encoder goes into a storage device, and each successive BCD code is stored until the entire number has been entered

Assignment

Design a single encoder for following functions.

F1 = Σm(1, 3, 7, 15)

f2 = Σm(4,6,8,10)