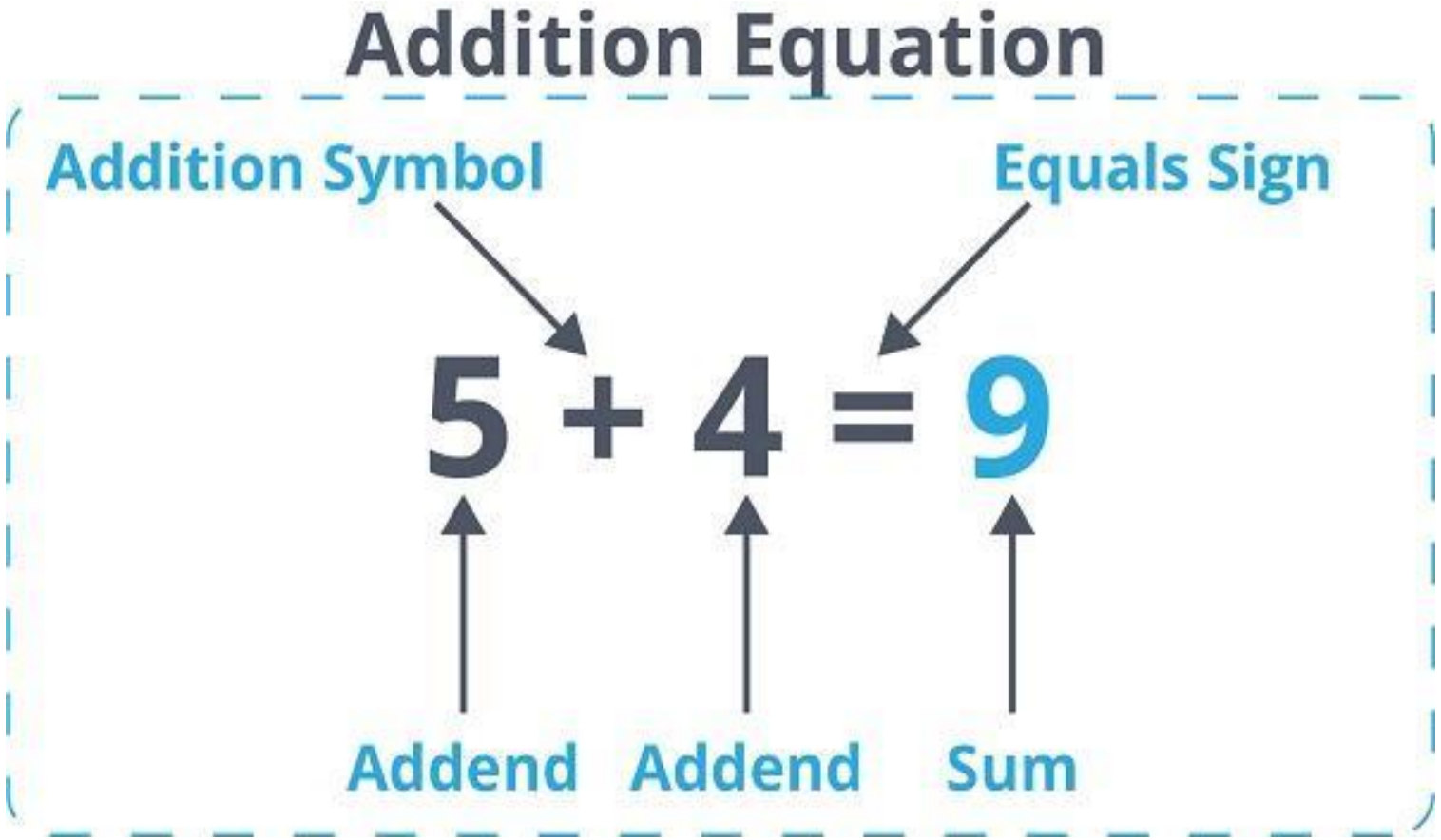
Chapter 6
Functions of Combinational Logic

Half, Full Adder & Parallel Adder

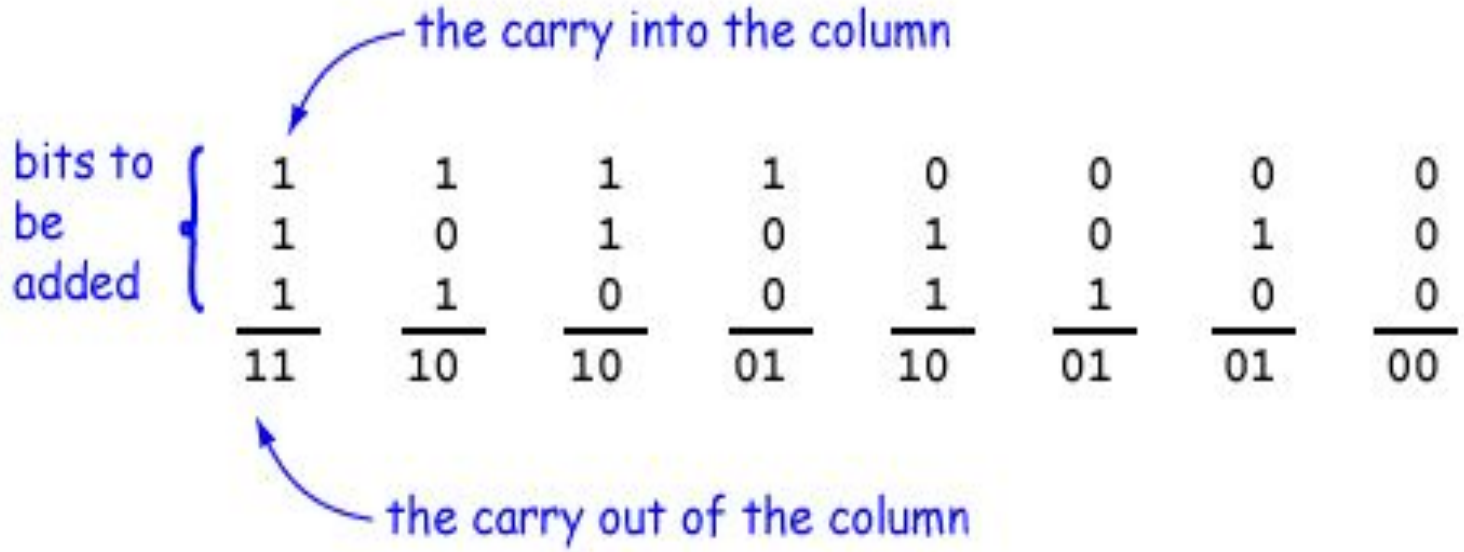
Simple Addition



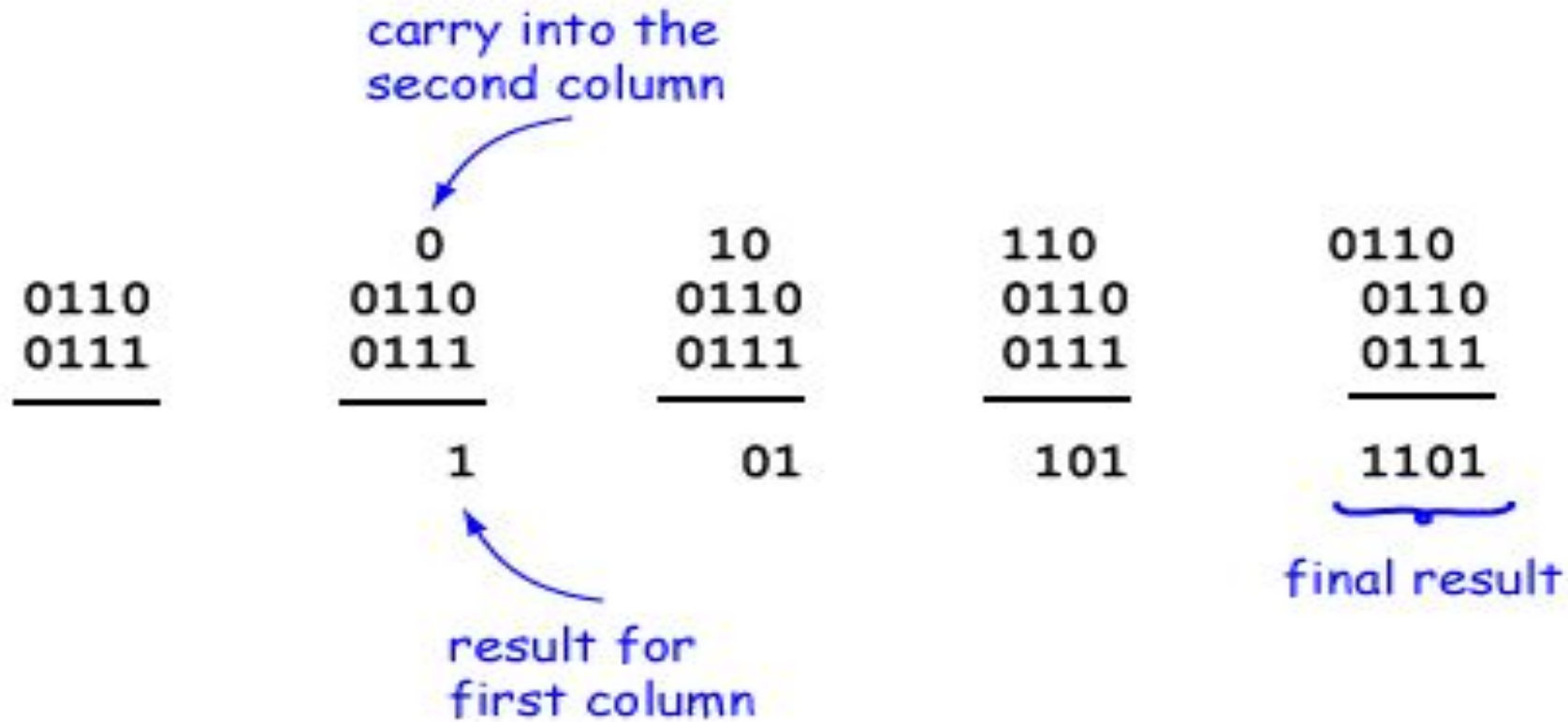
Binary Addition

$$
\begin{array}{r}
21 + 19 = x \\
\hline
\ 1 \\
\ 2\ 1 \\
+\ 1\ 9 \\
\hline
\ 4\ 0
\end{array}
\qquad
\begin{array}{r}
7 + 2 = 9 \\
\hline
\ 1\ 1 \\
\ 0\ 1\ 1\ 1 \\
+\ 0\ 0\ 1\ 0 \\
\hline
\ 1\ 0\ 0\ 1
\end{array}
$$

Rules for Binary Addition



Binary Addition Example



Adder Introduction

- An adder is a digital logic circuit in electronics that implements addition of numbers. In many computers and other types of processors, adders are used to calculate addresses, similar operations and table indices in the ALU and also in other parts of the processors. These can be built for many numerical representations like excess-3 or binary coded decimal.

Adders Are Classified Into Two Types

- Half Adder and Full Adder.
- The half adder circuit has two inputs: A and B.
- It add two input digits and generate a carry and sum.
- The full adder circuit has three inputs: A, B and C.
- It add the three input numbers and generate a carry and sum

Half Adder

- Using Half Adder, you can design simple addition with the help of logic gates.
- Let’s see an addition of single bits.

  0+0 = 0

  0+1 = 1

  1+0 = 1

  1+1 = 10

Half Adder

- These are the least possible single-bit combinations. But the result for 1+1 is 10, the sum result must be re-written as a 2-bit output. Thus, the equations can be written as 0+0 = 00

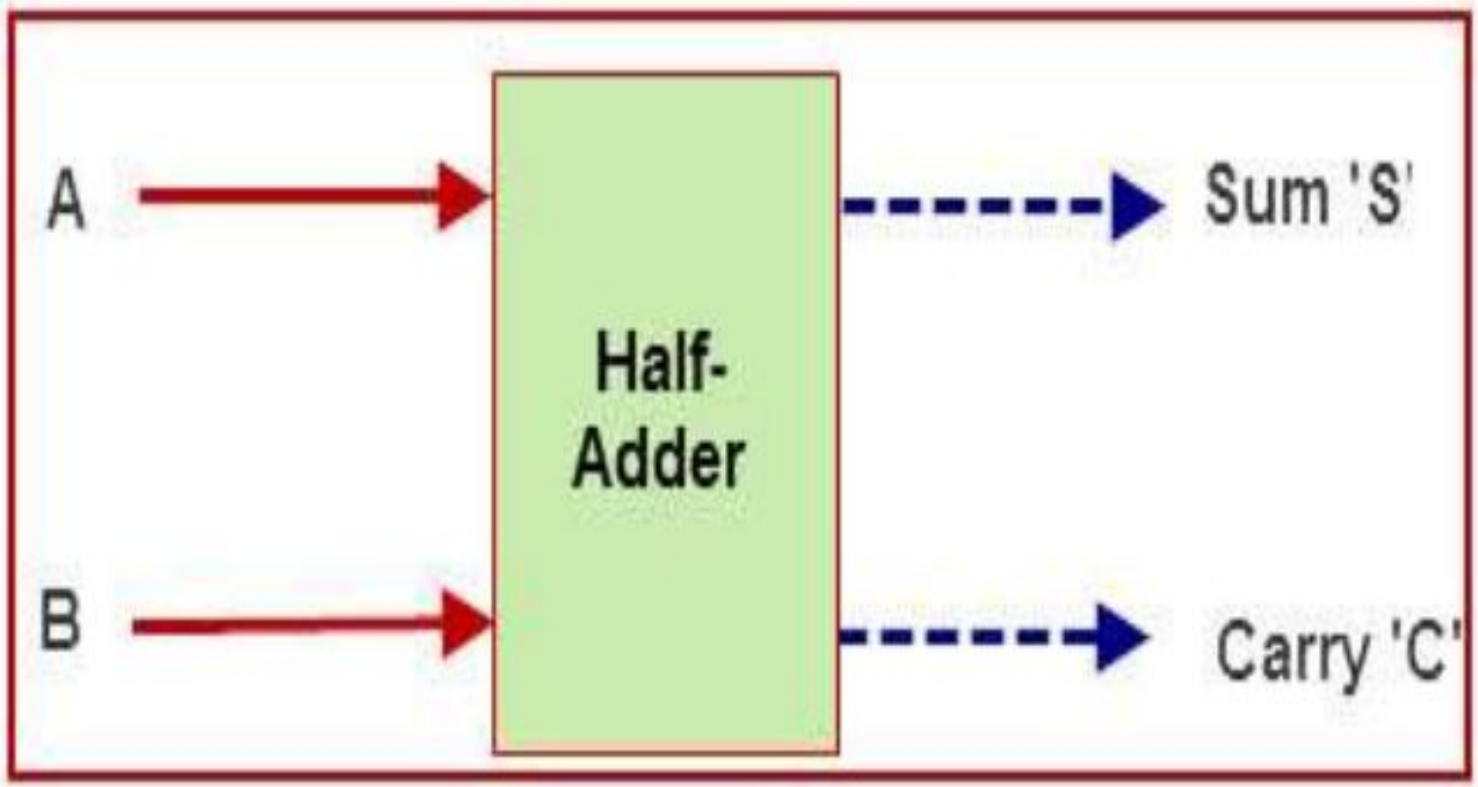
  0+1 = 01

  1+0 = 01

  1+1 = 10

- The output ‘1’of ‘10’ is carry-out. ‘SUM’ is the normal output and ‘CARRY’ is the carry-out.

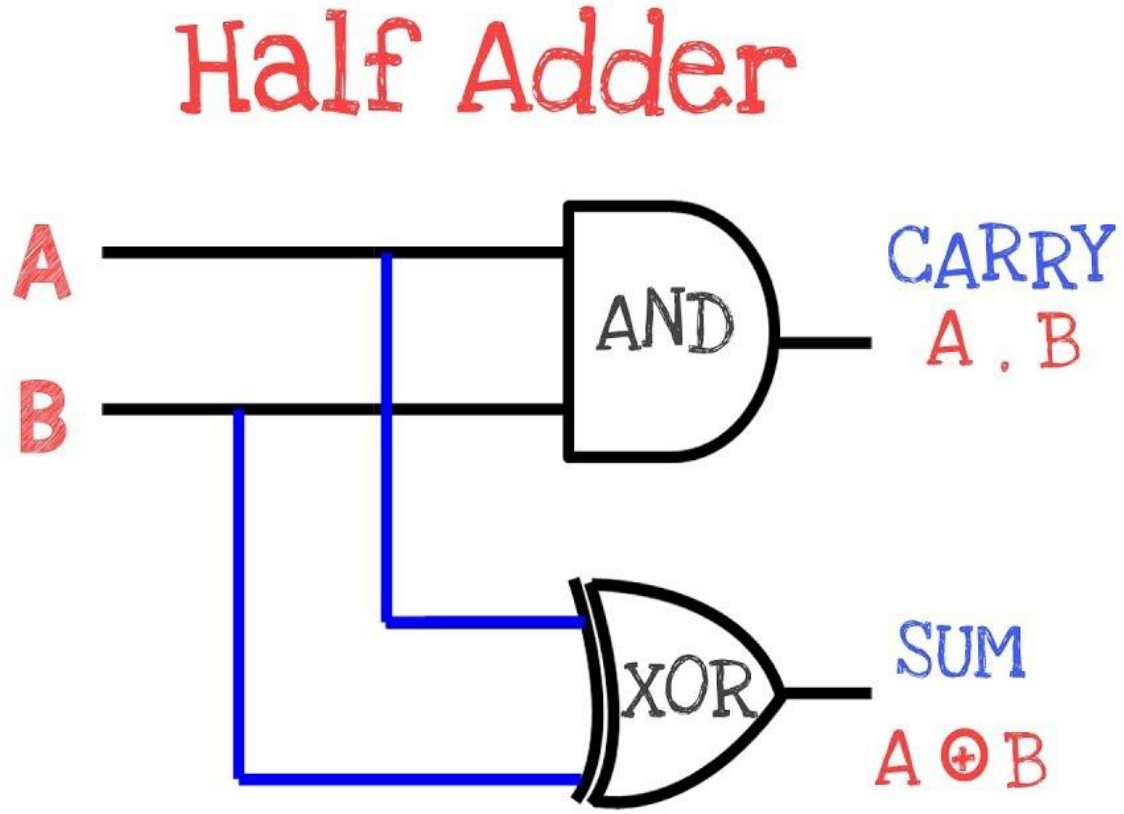
Half Adder



Half Adder Truth Table

INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder Expression



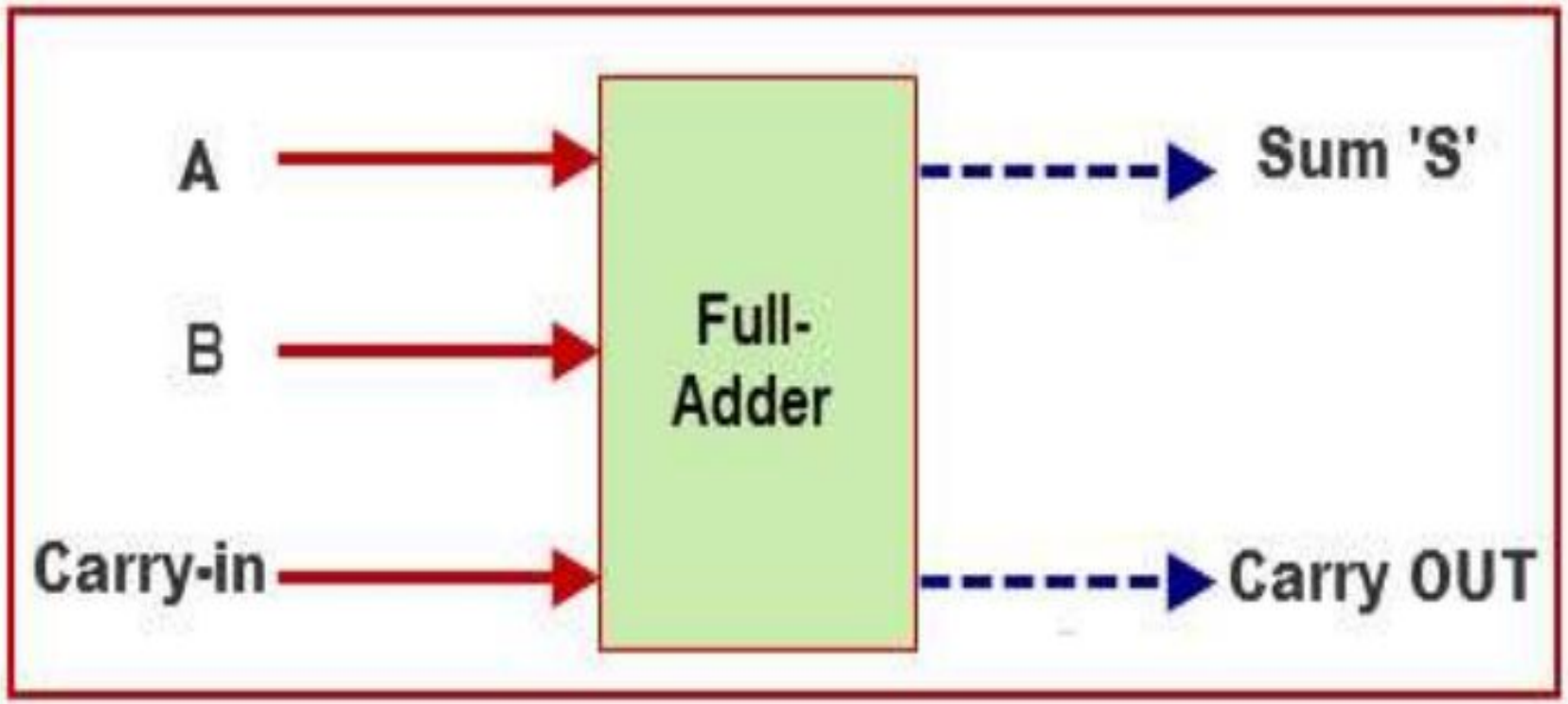
What is Half Adder Circuit

- The half adder adds two binary digits called as augend and addend.
- Half adder produces two outputs as sum and carry.
- XOR is applied to both inputs to produce sum.
- AND gate is applied to both inputs to produce carry.

Full Adder

- The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs.
- Whereas half adder has only two inputs and two outputs.
- The first two inputs are A and B and the third input is an input carry as CIN.
- When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

Full Adder



Full Adder Truth Table

INPUTS			OUTPUT	
A	B	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

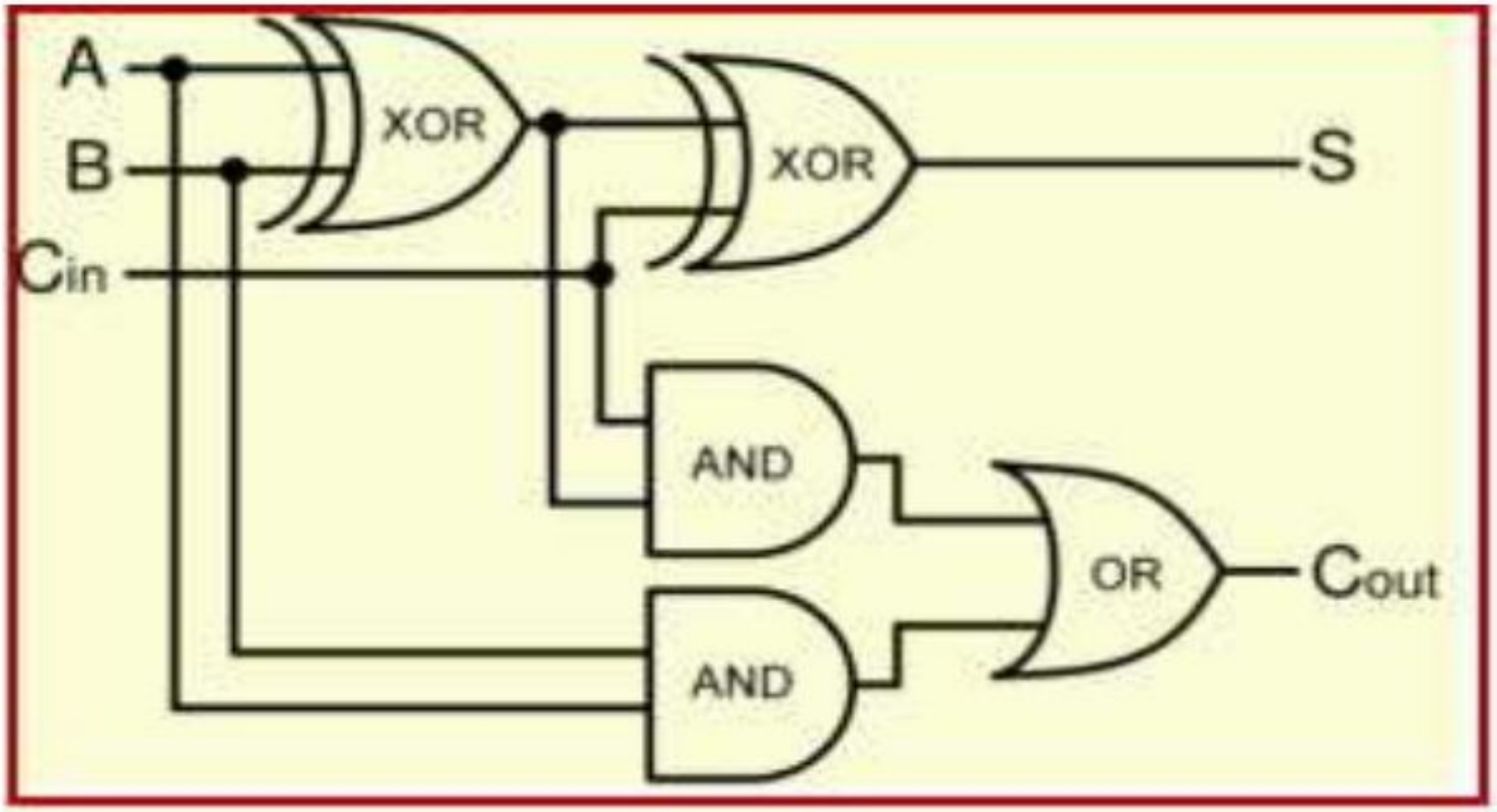
$$\text{Sum} = A \oplus B \oplus Cin$$

$$C\ out = (A \oplus B)\ Cin + (A \cdot B)$$

Full Adder Truth Table

- The full adder logic can be implemented with the truth table.
- The output S is an XOR between the input A and the half-adder, SUM output with B and C-IN inputs.
- Take C-OUT will only be true if any of the two inputs out of the three are HIGH.

Full Adder Logic Circuit



What Is Full Adder Circuit

- The full adder adds 3 one bit numbers.
- Where two can be referred to as operands.
- One can be referred to as bit carried in.
- It produces 2-bit output, and these can be referred to as output carry and sum.

Full Adder Logic Circuit

- We can implement a full adder circuit with the help of two half adder circuits.
- First, half adder will be used to add A and B to produce a partial Sum.
- A second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.
- If any of the half adder logic produces a carry, there will be an output carry.

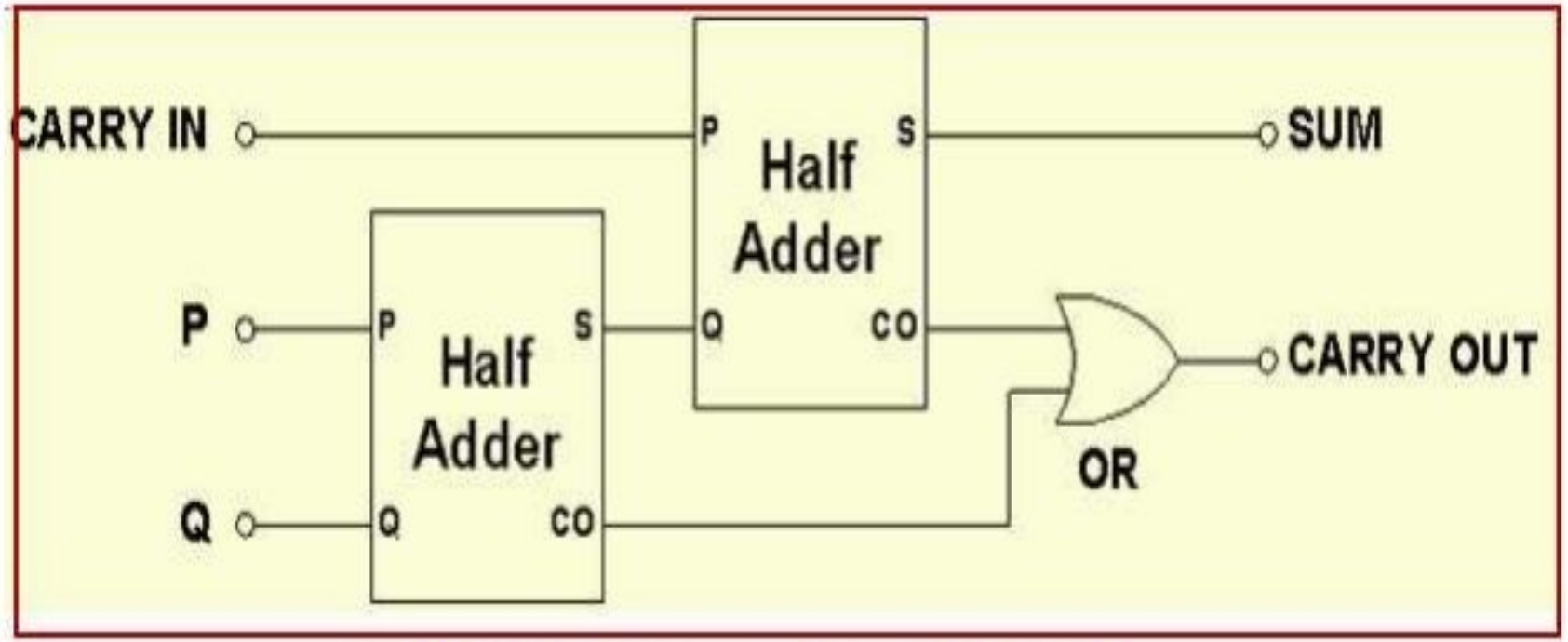
Full Adder Logic Circuit

- We can implement a full adder circuit with the help of two half adder circuits.
- First, half adder will be used to add A and B to produce a partial Sum.
- A second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.
- If any of the half adder logic produces a carry, there will be an output carry.

Full Adder Logic Circuit

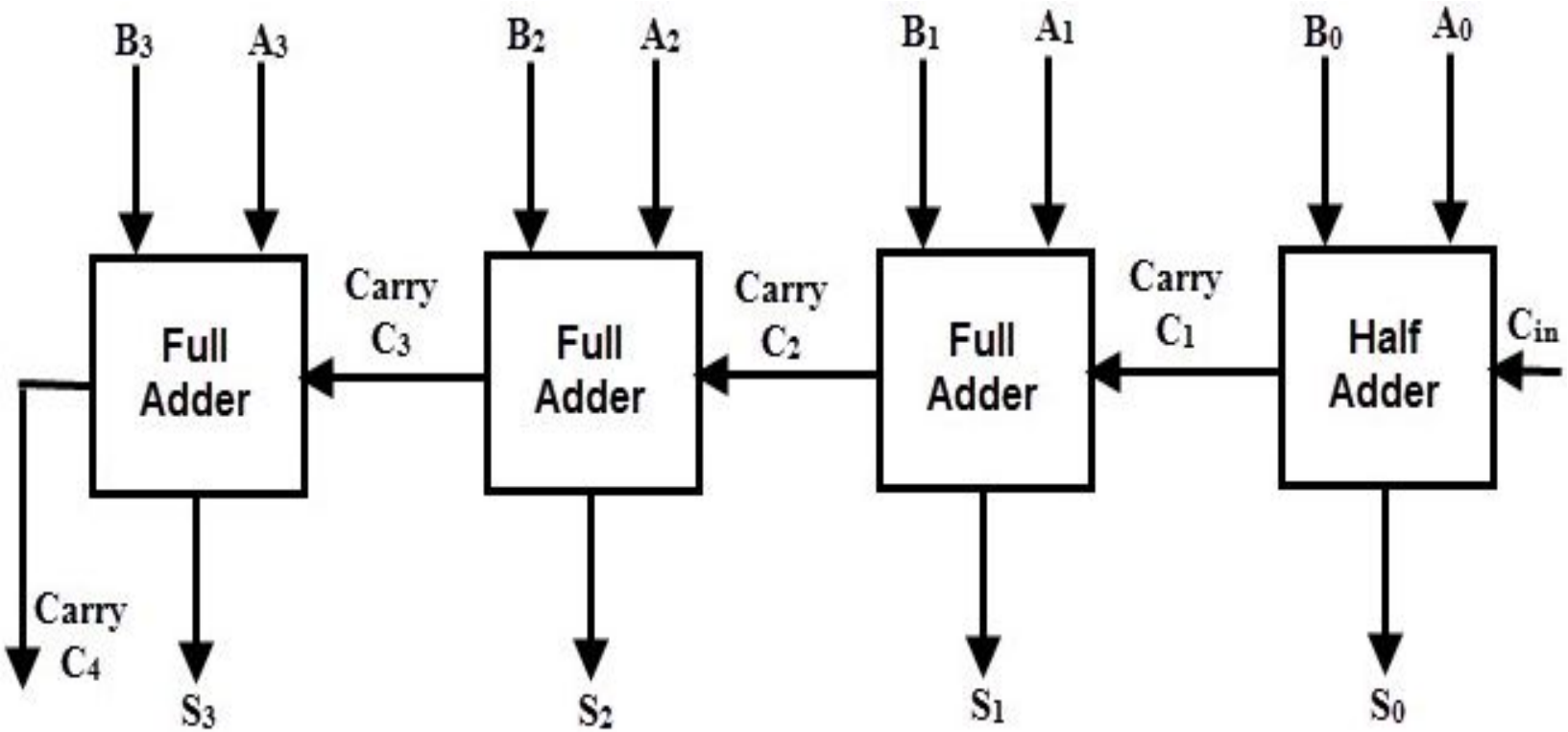
- A C-OUT will be an OR function of the half-adder Carry outputs.
- The implementation of larger logic diagrams is possible with full adder logic.

Making Full Adder Using Half Adder



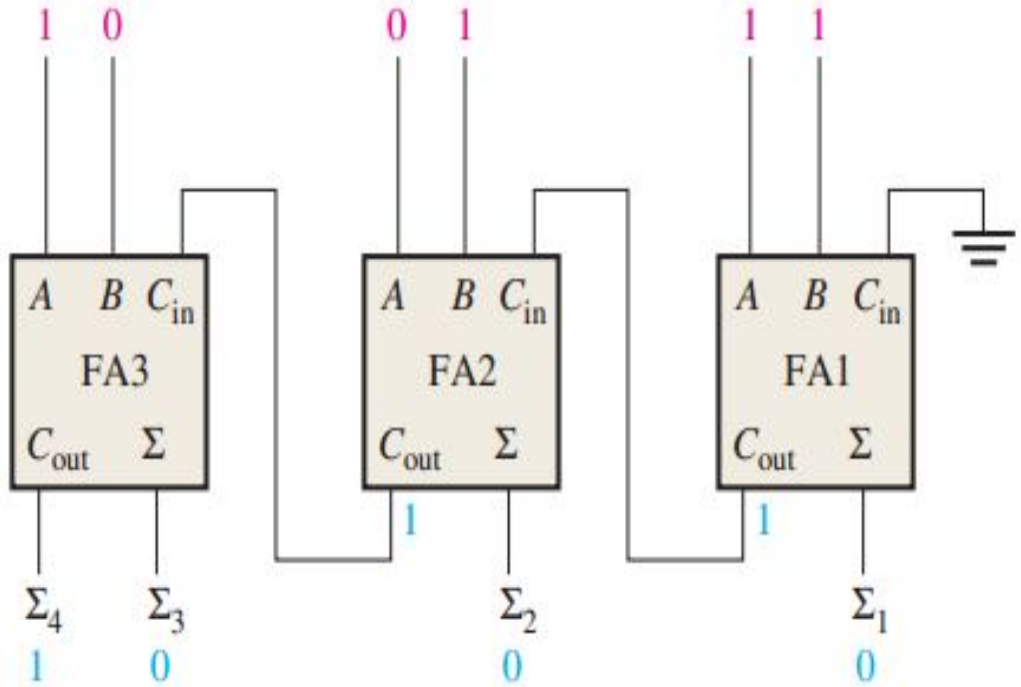
Full Adder Design Using Half Adders

Parallel Adder

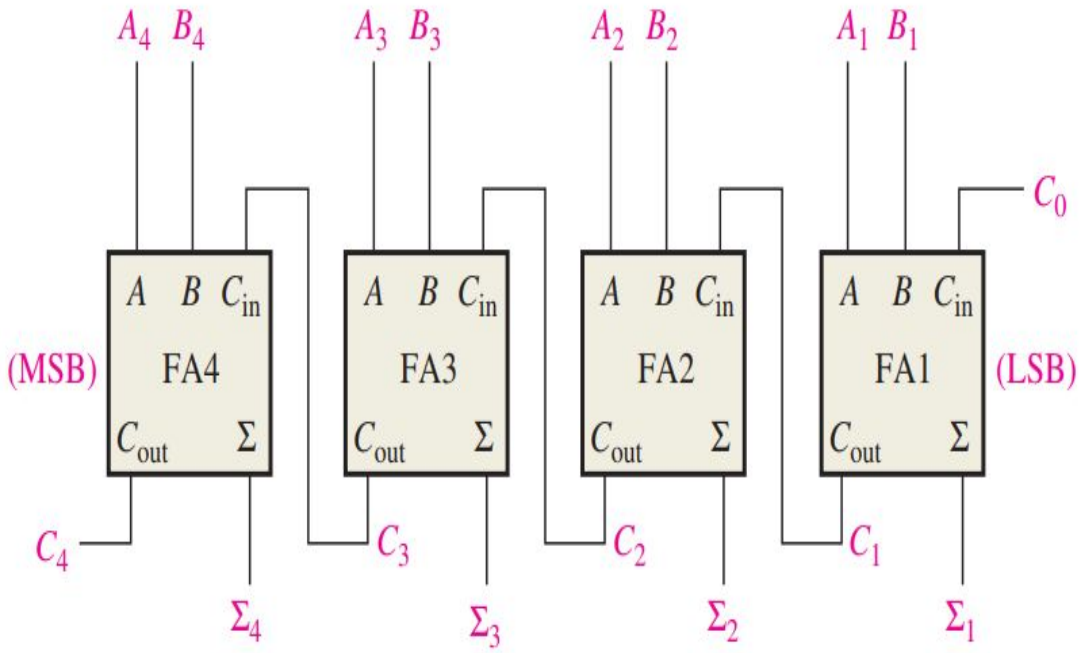


EXAMPLE 6–2

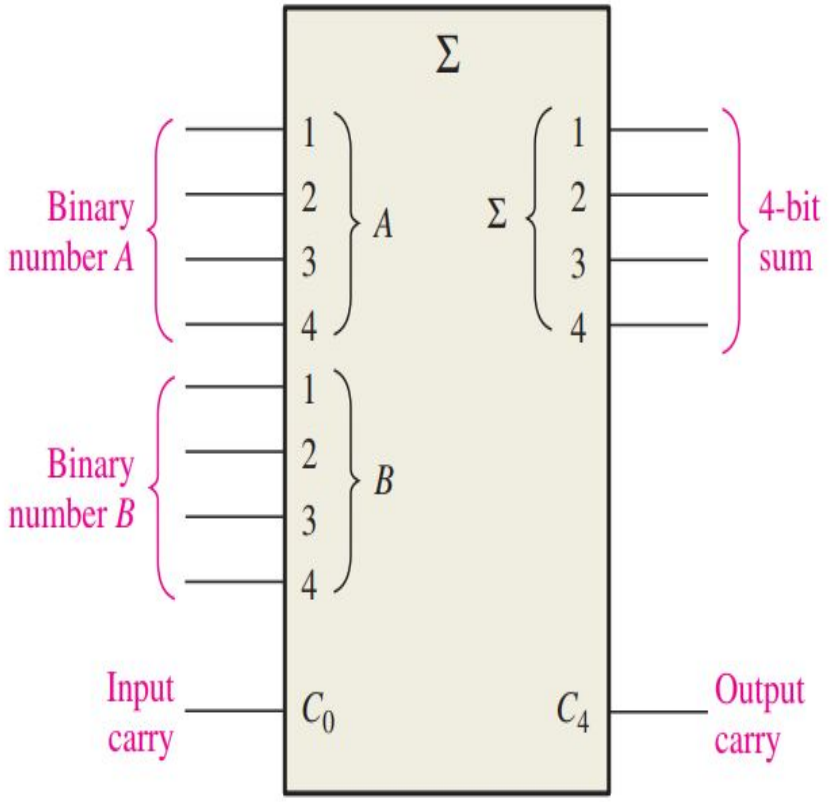
Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.



Four-Bit Parallel Adders



(a) Block diagram



(b) Logic symbol

FIGURE 6–9 A 4-bit parallel adder.

Adder Expansion: Constructing 8 bit Adder using 4 bit Adder IC

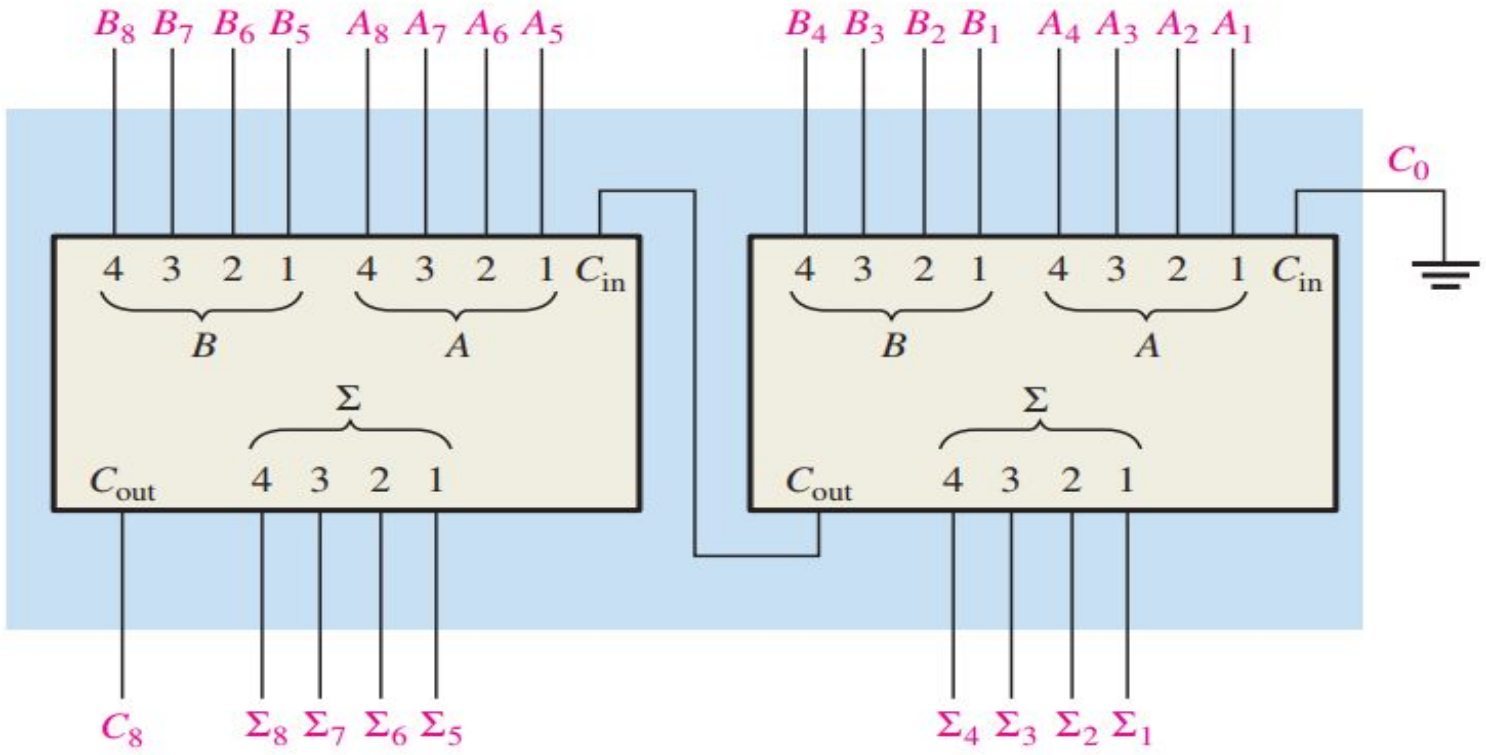


FIGURE 6–11 Cascading of two 4-bit adders to form an 8-bit adder.

EXAMPLE 6–4

Show how two 74HC283 adders can be connected to form an 8-bit parallel adder. Show output bits for the following 8-bit input numbers:

$$A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001 \quad \text{and} \quad B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$$

Solution

Two 74HC283 4-bit parallel adders are used to implement the 8-bit adder. The only connection between the two 74HC283s is the carry output (pin 9) of the low-order adder to the carry input (pin 7) of the high-order adder, as shown in Figure 6–12. Pin 7 of the low-order adder is grounded (no carry input).

The sum of the two 8-bit numbers is

$$\Sigma_9\Sigma_8\Sigma_7\Sigma_6\Sigma_5\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 101010111$$

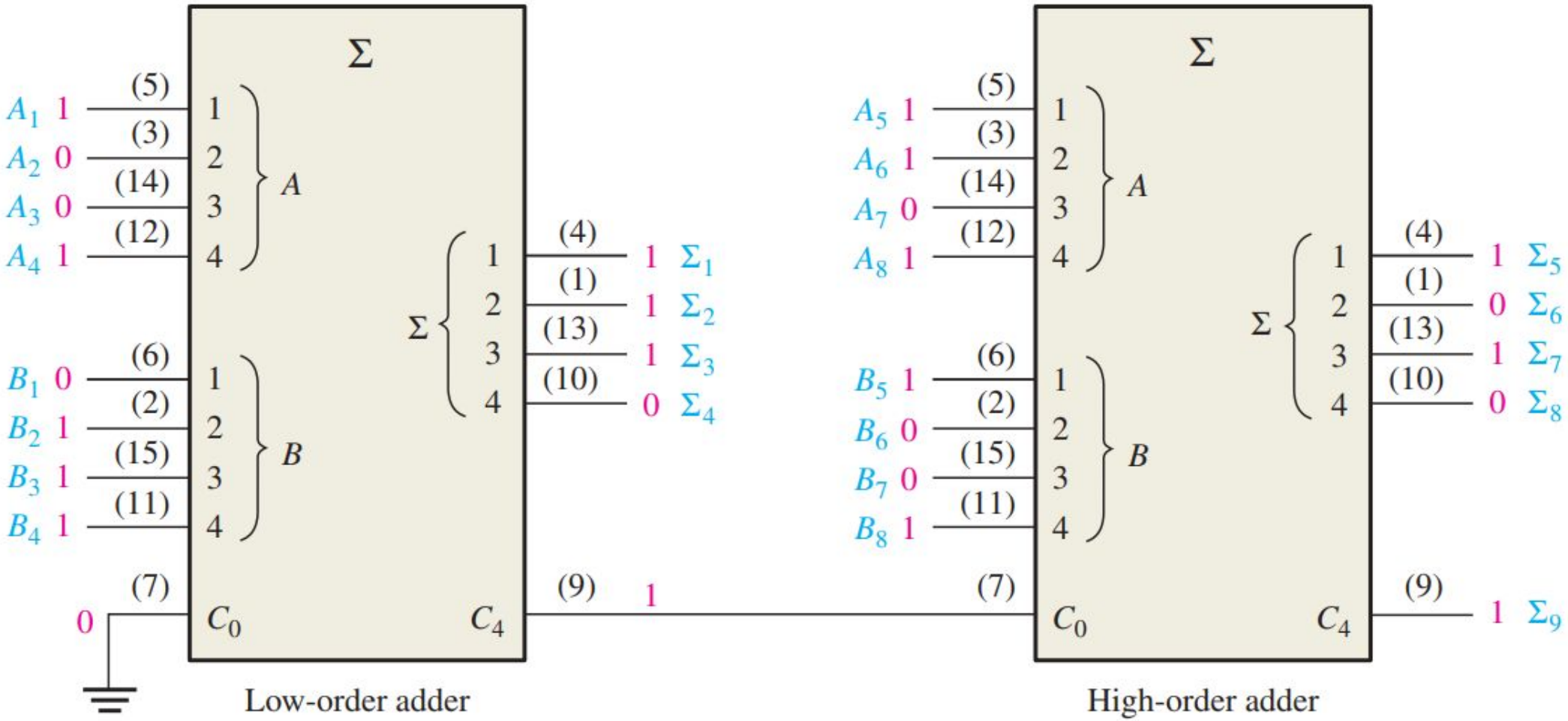


FIGURE 6–12 Two 74HC283 adders connected as an 8-bit parallel adder (pin numbers are in parentheses).

Adder Application (Voting Machine)

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes. This type of system can be used where a group of people are assembled and there is a need for immediately determining opinions (for or against), making decisions, or voting on certain issues or other matters.

In its simplest form, the system includes a switch for "yes" or "no" selection at each position in the assembly and a digital display for the number of yes votes and one for the number of no votes. The basic system is shown in Figure 6–13 for a 6-position setup, but it can be expanded to any number of positions with additional 6-position modules and additional parallel adder and display circuits.

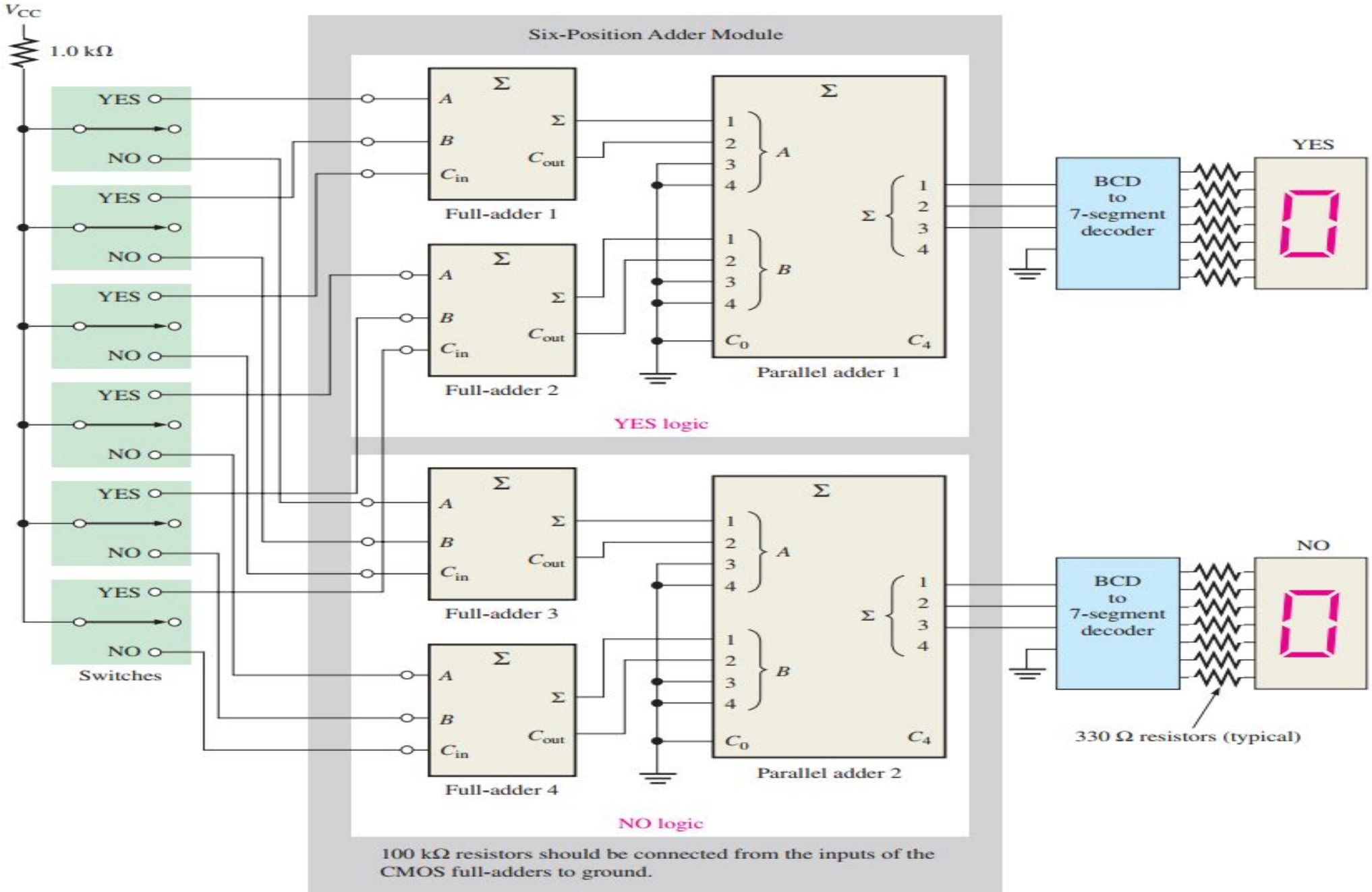
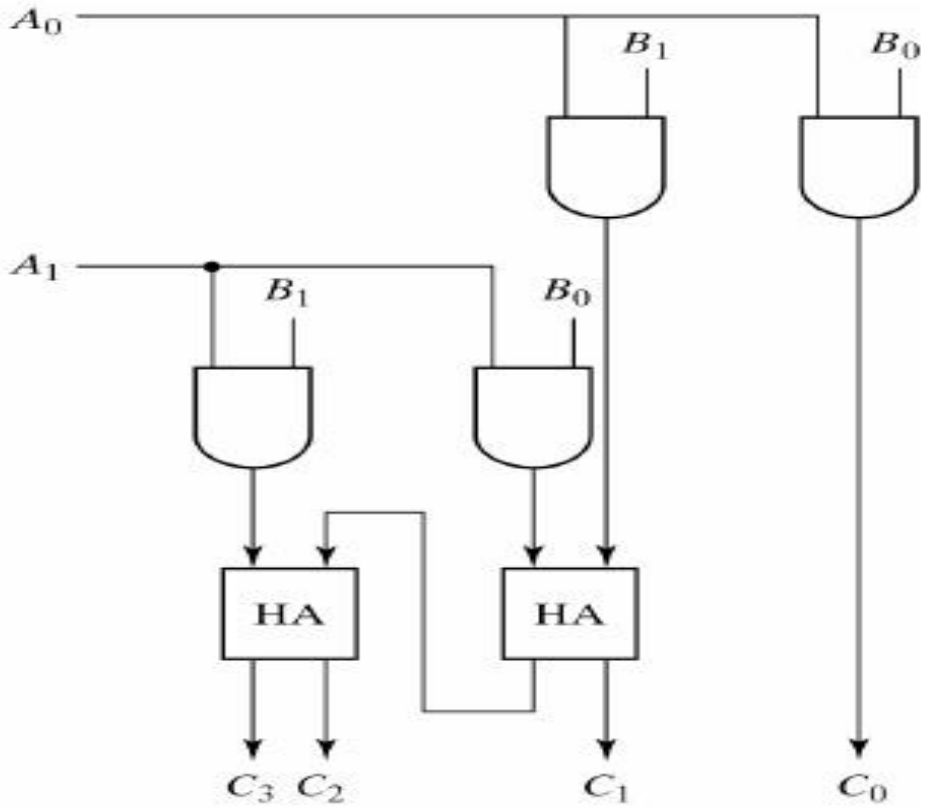


FIGURE 6–13 A voting system using full-adders and parallel binary adders.

2 Bit Multiplier

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



Design 3 Bit & 4 Bit Multiplier

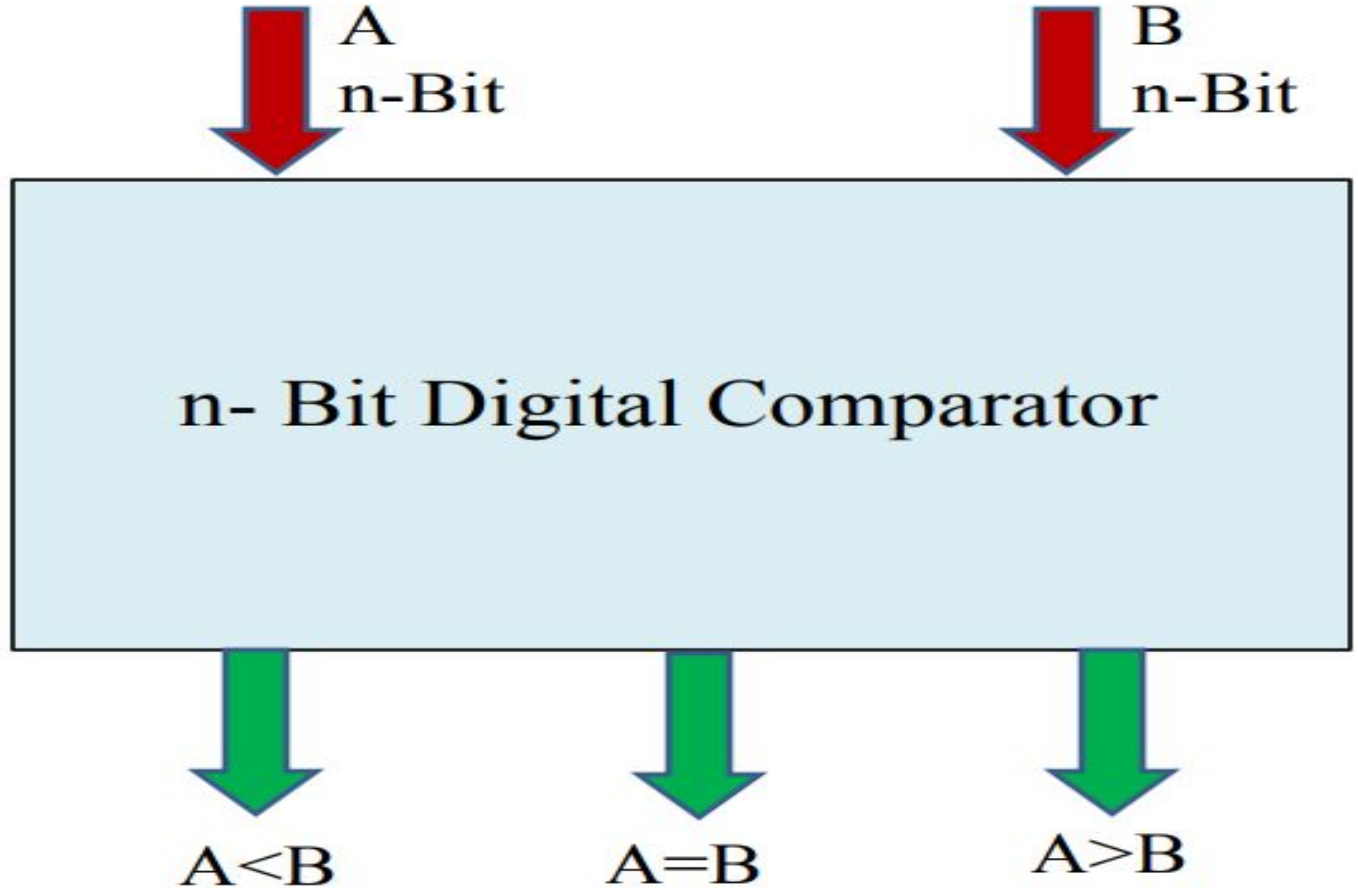
Comparator

MAGNITUDE COMPARATOR: DIGITAL COMPARATOR

- It is a combinational logic circuit.
- Digital Comparator is used to compare the value of two binary digits.
- There are two types of digital comparator
- (i) Identity Comparator
- (ii) Magnitude Comparator.

- IDENTITY COMPARATOR: This comparator has only one output terminal for when A=B, either A=B=1 (High) or A=B=0 (Low)
- MAGNITUDE COMPARATOR: This Comparator has three output terminals namely A>B, A=B, A<B. Depending on the result of comparison, one of the output will be High (1).
- Block Diagram of Magnitude Comparator is shown in Fig. 1

BLOCK DIAGRAM OF MAGNITUDE COMPARATOR



1- Bit Magnitude Comparator

- This magnitude comparator has two inputs A and B and three outputs AB.
- This magnitude comparator compares the two numbers of single bits.
- Truth Table of 1-Bit Comparator

INPUTS		OUTPUTS		
A	B	Y_1 (A<B)	Y_2 (A=B)	Y_3 (A>B)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

K-Maps For All Three Outputs

A \ B	$\overline{B}$ 0	B 1
$\overline{A}$ 0	0	(1)
A 1	0	0

K-Map for Y_1 : A<B

$Y_1 = \overline{A}B$

A \ B	$\overline{B}$ 0	B 1
$\overline{A}$ 0	(1)	0
A 1	0	(1)

K-Map for Y_2 : A=B

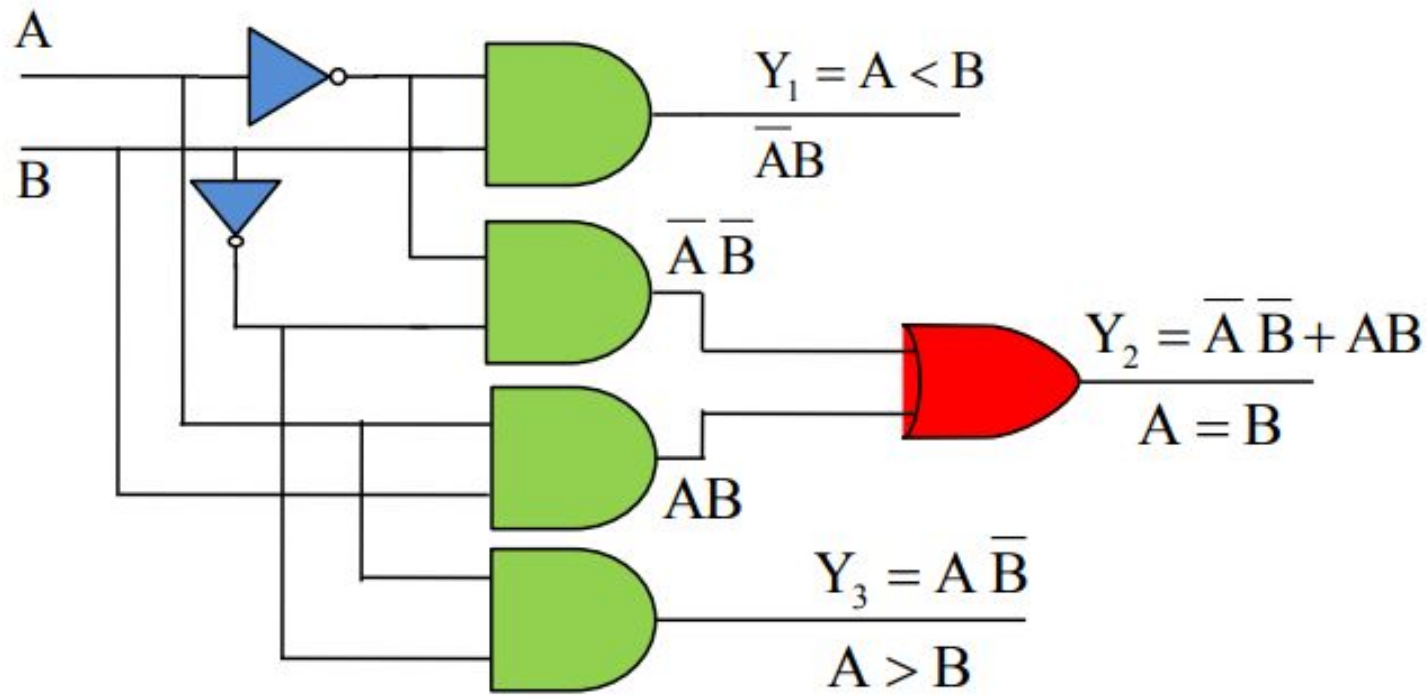
$Y_2 = \overline{A}\,\overline{B} + AB$

A \ B	$\overline{B}$ 0	B 1
$\overline{A}$ 0	0	0
A 1	(1)	0

K-Map for Y_2 : A>B

$Y_3 = A\overline{B}$

Realization of One Bit Comparator

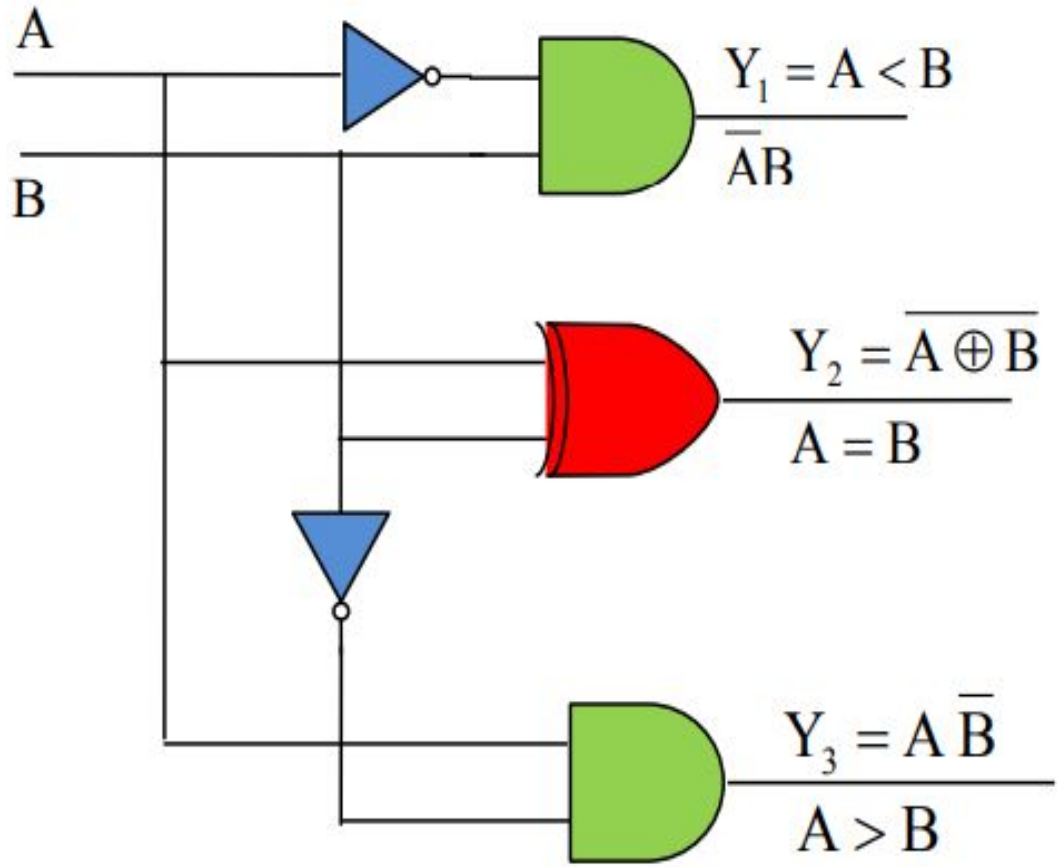


$$Y_1 = \overline{A}B$$

$$Y_2 = \overline{A}\,\overline{B} + AB$$

$$Y_3 = A\overline{B}$$

Realization of by Using AND , EX-NOR gates



2-Bit Comparator:

- A comparator which is used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator.
- Fig. 2 shows the block diagram of 2-Bit magnitude comparator.
- It has four inputs and three outputs.
- Inputs are A_0 ,A_1,B_0 and B_1 and Outputs are Y_1, Y_2 and Y_3

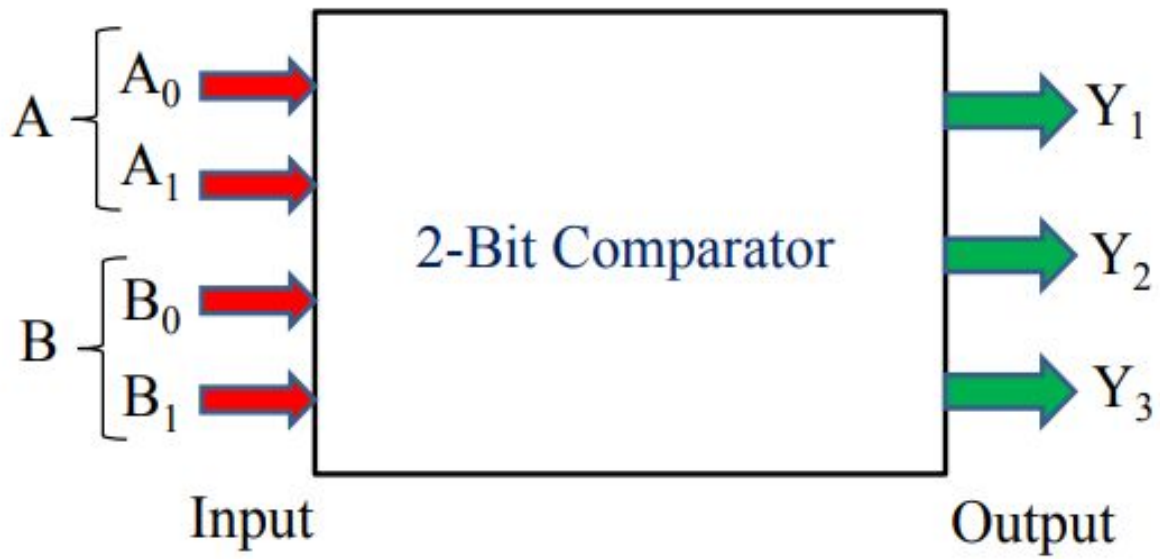
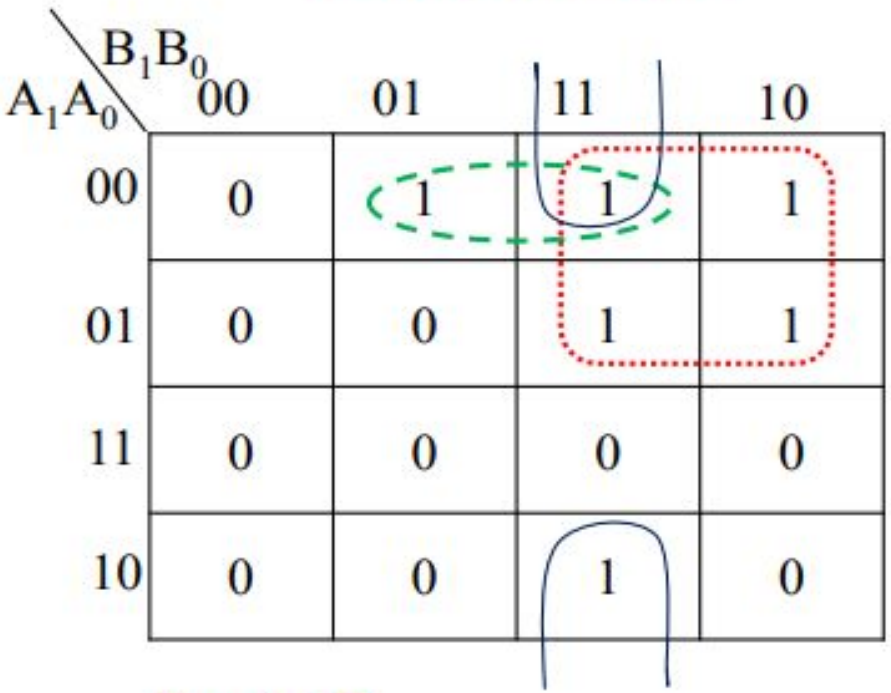


Fig. 2

TRUTH TABLE

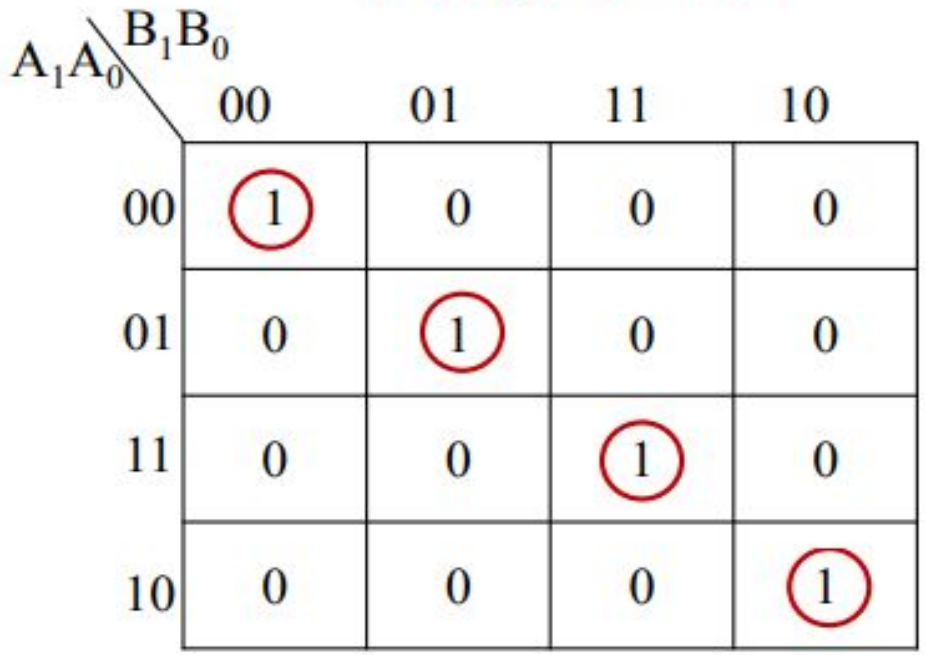
INPUT				OUTPUT		
A_1	A_0	B_1	B_0	Y_1=A<B	Y_2=(A=B)	Y_3=A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

K-Map for A<B:



A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

K-Map for A=B:



A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

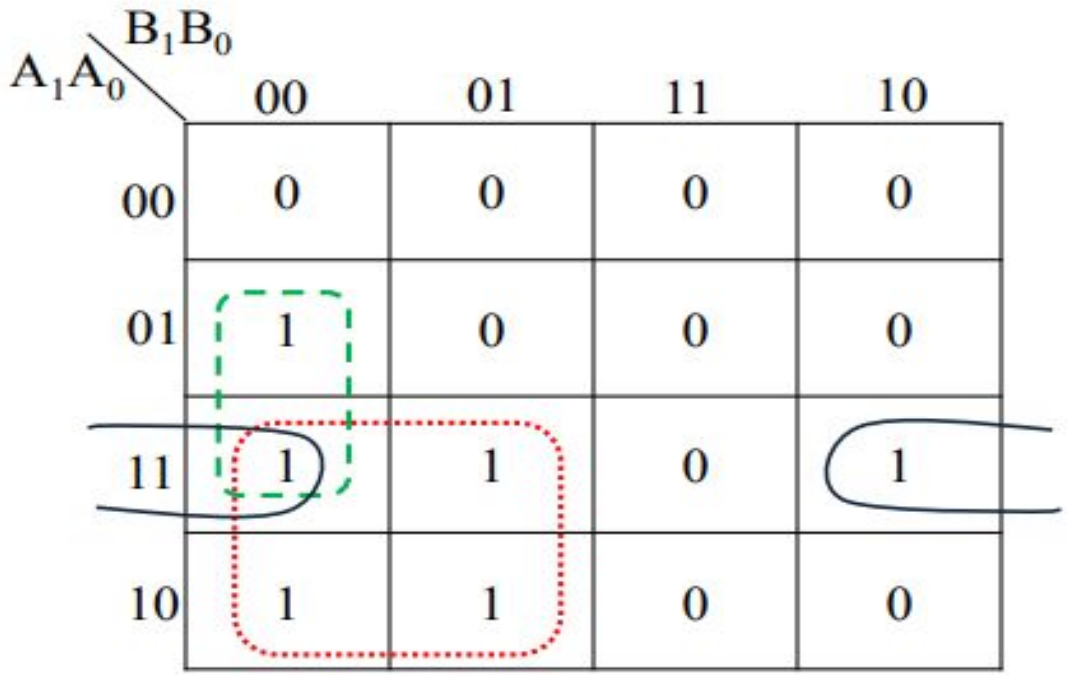
For A<B

$$Y_1 = \overline{A_1}\,\overline{A_0}\,B_0 + \overline{A_1}\,B_1 + \overline{A_0}\,B_1\,B_0$$

For A=B

$$Y_2 = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}\,A_0\,\overline{B_1}\,B_0 + A_1A_0B_1B_0 + A_1\,\overline{A_0}\,B_1\,\overline{B_0}$$

K-Map For A>B



$$Y_3 = A_0 \overline{B_1}\,\overline{B_0} + A_1 \overline{B_1} + A_1 A_0 \overline{B_0}$$

For A=B From K-Map

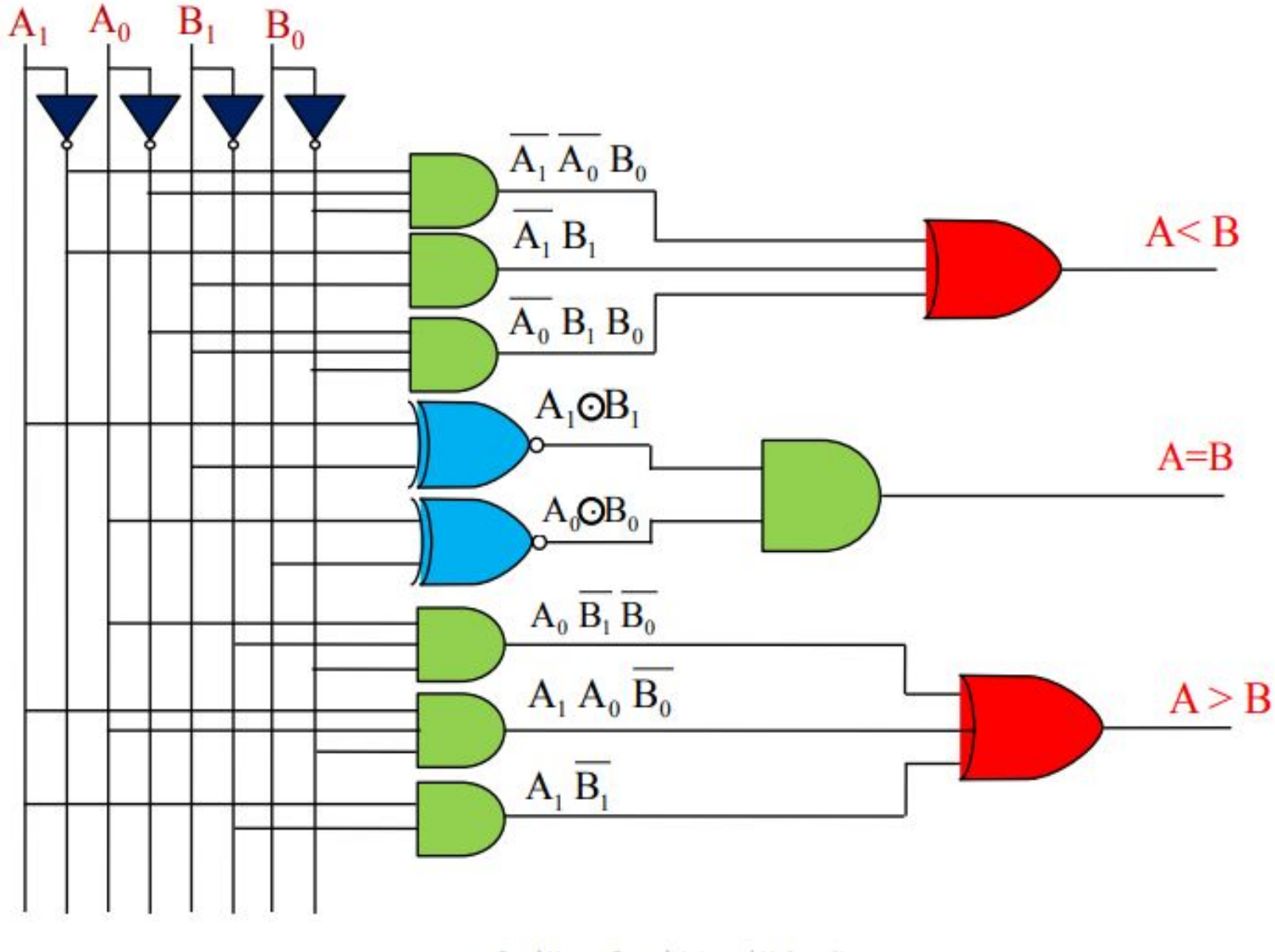
$$Y_2 = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}\,A_0\,\overline{B_1}\,B_0 + A_1A_0B_1B_0 + A_1\,\overline{A_0}\,B_1\,\overline{B_0}$$

$$Y_2 = \overline{A_0}\,\overline{B_0}(\overline{A_1}\,\overline{B_1} + A_1B_1) + A_0B_0(\overline{A_1}\,\overline{B_1} + A_1B_1)$$

$$Y_2 = (\overline{A_1}\,\overline{B_1} + A_1B_1)\ (\overline{A_0}\,\overline{B_0} + A_0B_0)$$

$$Y_2 = (A_1 \odot B_1)\,(A_0 \odot B_0)$$

LOGIC DIAGRAM OF 2-BIT COMPARATOR:



3-bit Comparator

- To design 3-bit comparator.
- Number of inputs 6 A2,A1,A0, B2,B1,B0
- Total number of combination 2^6=64
- Huge truth table then K-map so traditional method is complex.
- We need another approach to design a comparator circuit.

- Designing general equation for comparator.
- In this example we are comparing 3 bits two inputs.
- A2 & B2 are MSB. A0 & B0 are LSB.
- It compares each of these bits in one number with bits in that of other number and produces one of the following outputs as A = B, A < B and A>B. The output logic statements of this converter are

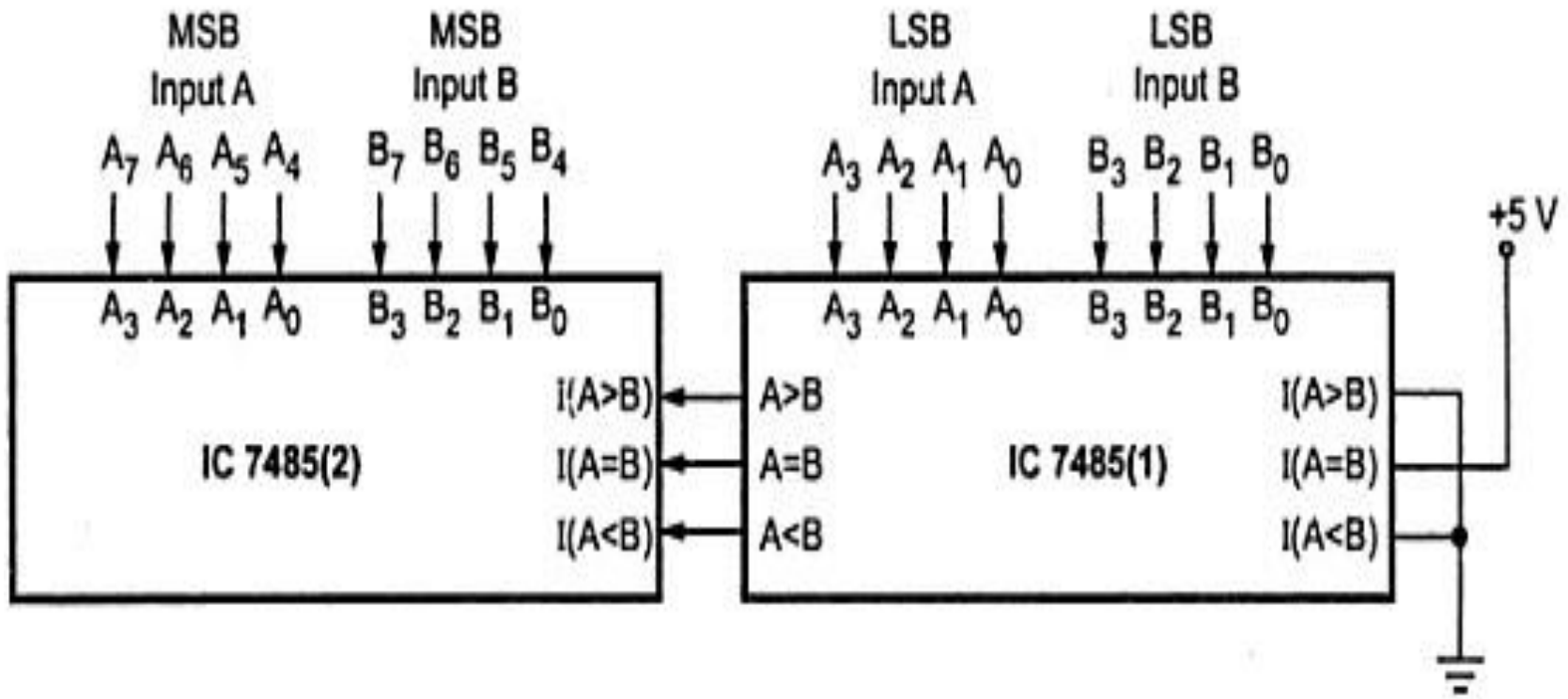
- If A2 = 1 and B2 = 0, then A is greater than B (A>B). Or
- If A1 and B2 are equal, and if A1 = 1 and B1 = 0, then A > B. Or
- If A2 and B2 are equal & A1 and B1 are equal, and if A0 = 1, and B0 = 0, then A>B. Or

Equation

- A>B = A2B2’ + (A2 Ex-Nor B2)A1B1’ + (A2 Ex-Nor B2)(A1 Ex-Nor B1)A0B0’
- A<B = A2’B2 + (A2 Ex-Nor B2)A1’B1 + (A2 Ex-Nor B2)(A1 Ex-Nor B1)A0’B0
- A=B =(A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)

4-Bit Comparator IC

8-Bit Comparator



EXAMPLE 6–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–22.

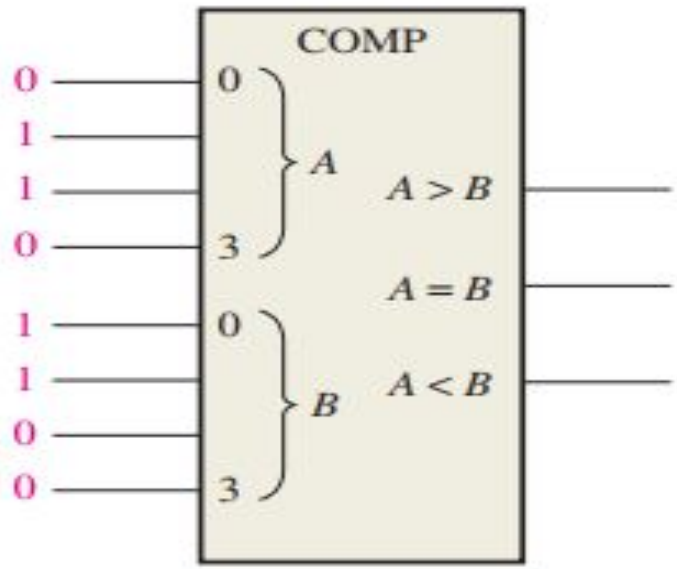


FIGURE 6–22

Solution

The number on the A inputs is 0110 and the number on the B inputs is 0011. The **$A > B$ output is HIGH and the other outputs are LOW.**

Related Problem

What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

Cascading of Comparator

EXAMPLE 6–7

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. Show the comparators with proper interconnections.

Solution

Two 74HC85s are required to compare two 8-bit numbers. They are connected as shown in Figure 6–25 in a cascaded arrangement.

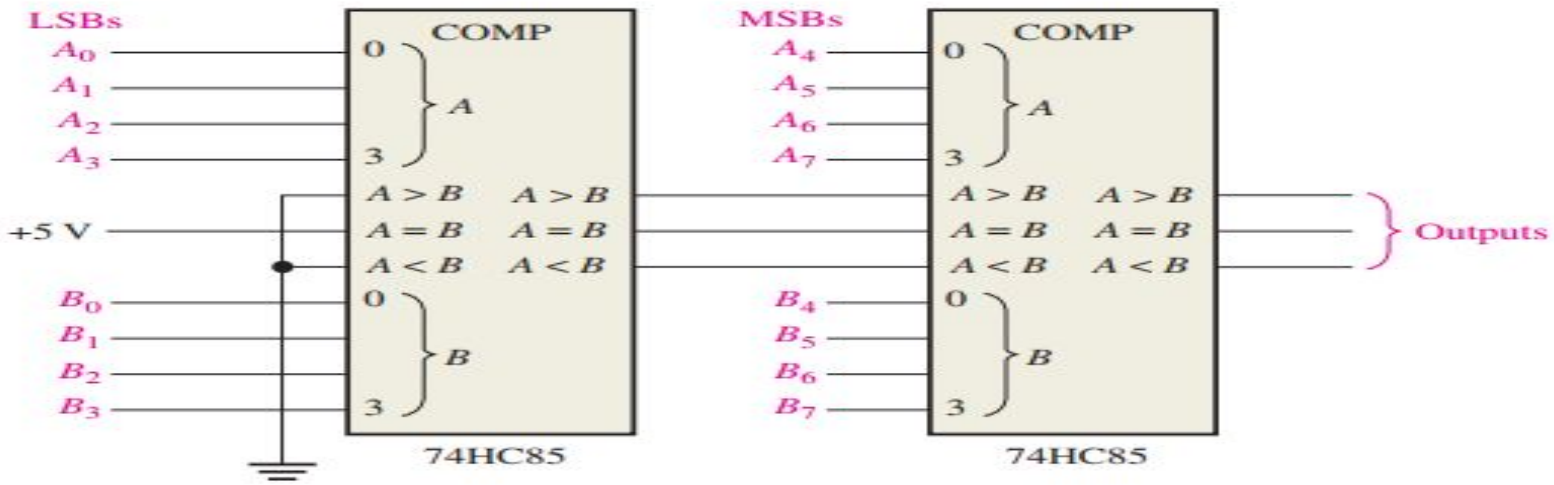


FIGURE 6–25 An 8-bit magnitude comparator using two 74HC85s.

Related Problem

Expand the circuit in Figure 6–25 to a 16-bit comparator.

MUX & DMUX

Multiplexer (MUX)

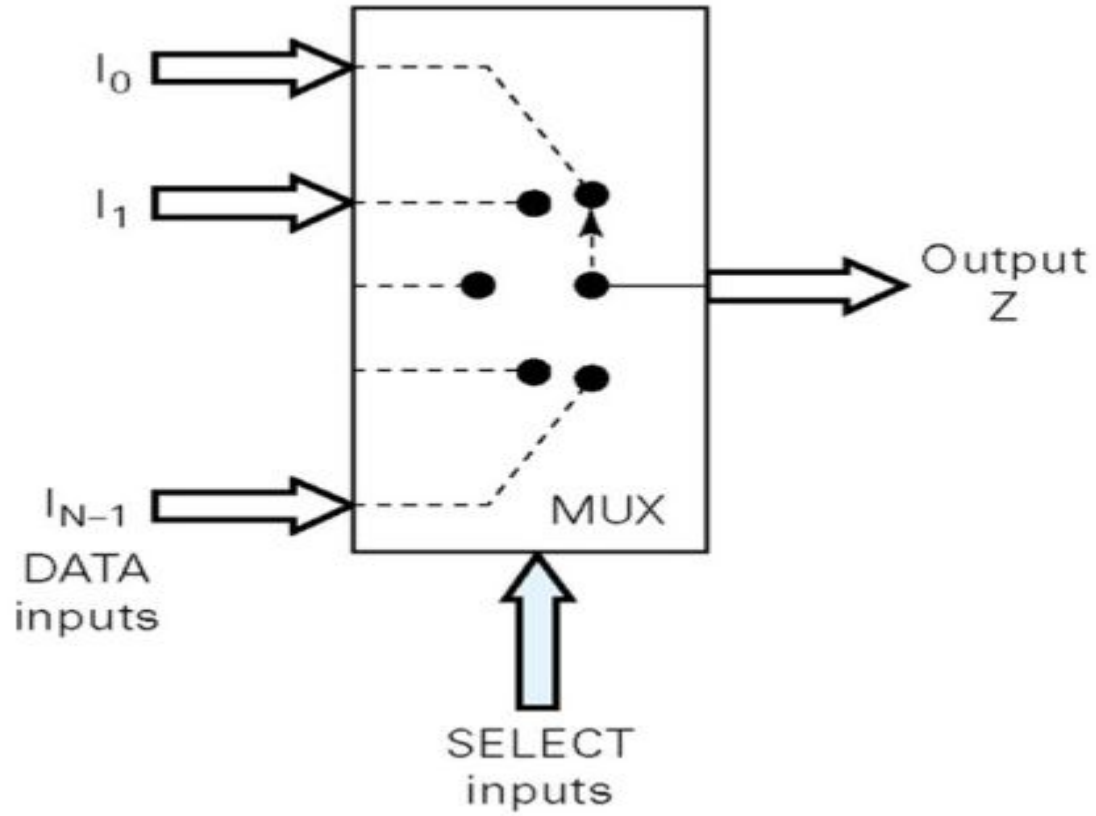
Many to One

Multiplexer (data Selectors)

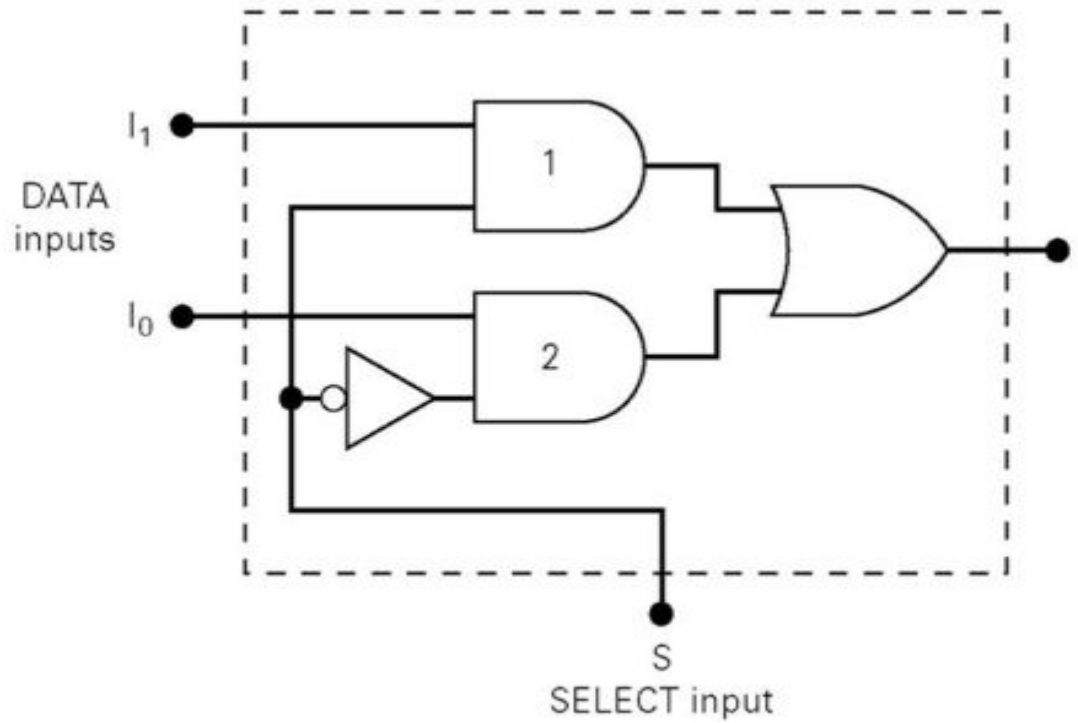
- *Definition :* A multiplexers (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Several data input lines
- Some select line (less than the no. of input lines)
- Single output line
- If there are n data input lines and m select lines, then

$$2^m = n$$

Functional Diagram Of a Multiplexer

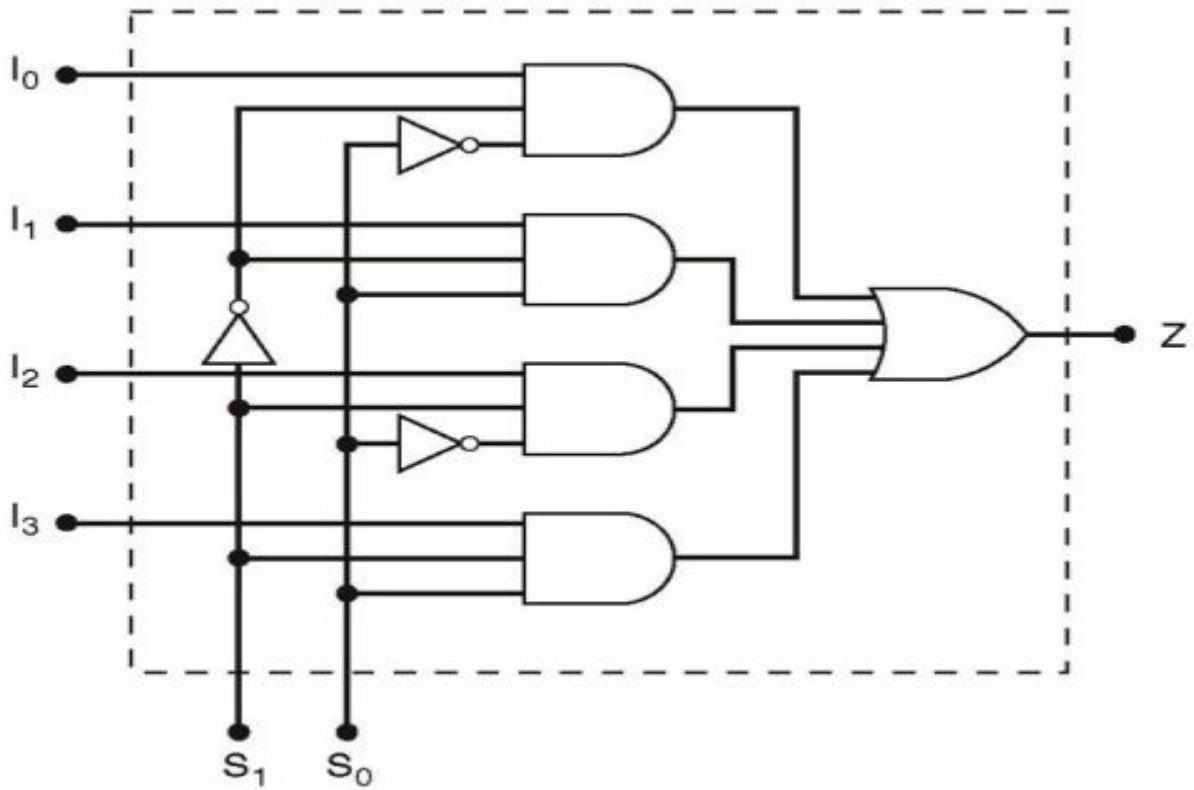


2 : 1 Multiplexer



S	Z
0	I_0
1	I_1

4 : 1 Multiplexer

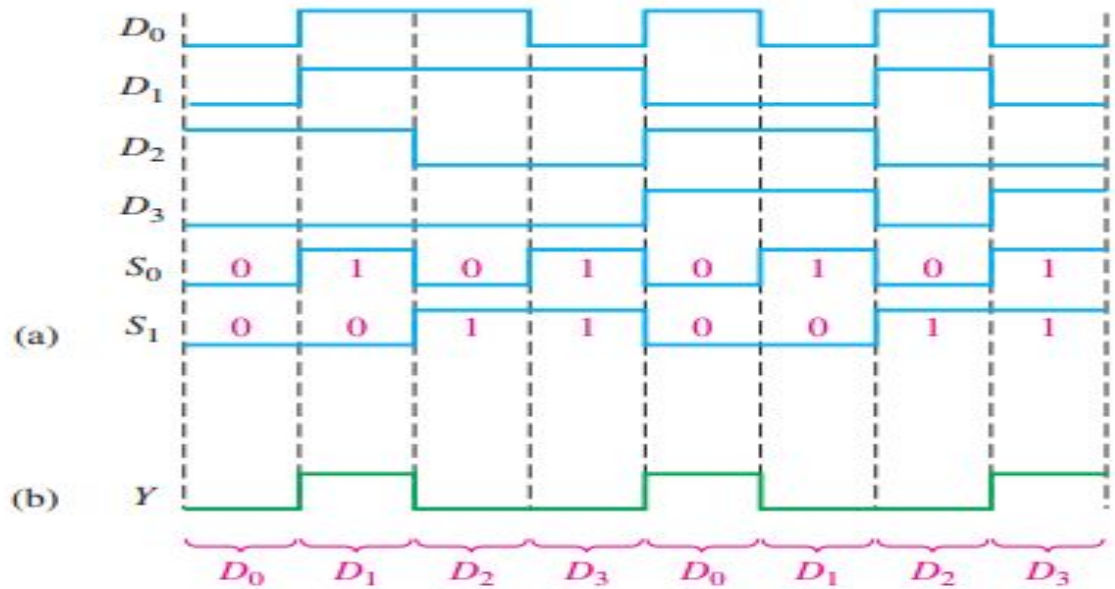


S_0	S_1	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

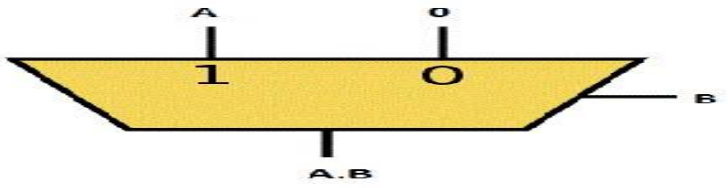
4:1 MUX Timing Diagram

EXAMPLE 6–14

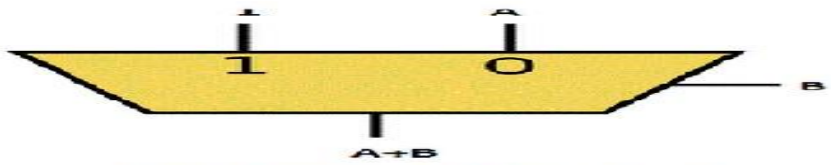
The data-input and data-select waveforms in Figure 6–45(a) are applied to the multiplexer in Figure 6–44. Determine the output waveform in relation to the inputs.



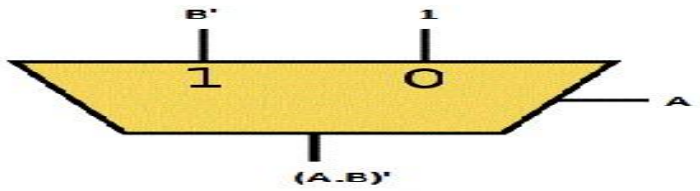
Logic Gates Implementation Using MUX



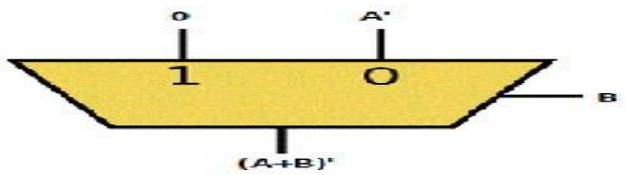
Mux Implementation of AND



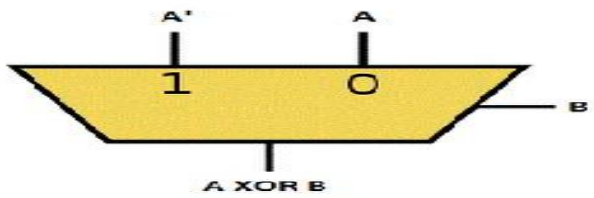
Mux Implementation of OR



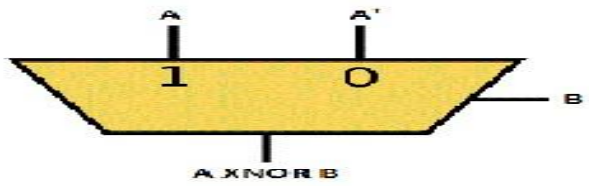
Mux Implementation of NAND



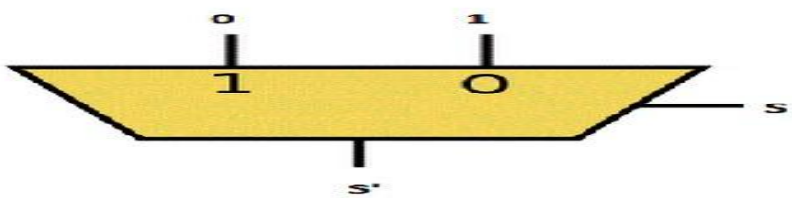
Mux Implementation of NOR



Mux Implementation of XOR



Mux Implementation of XNOR

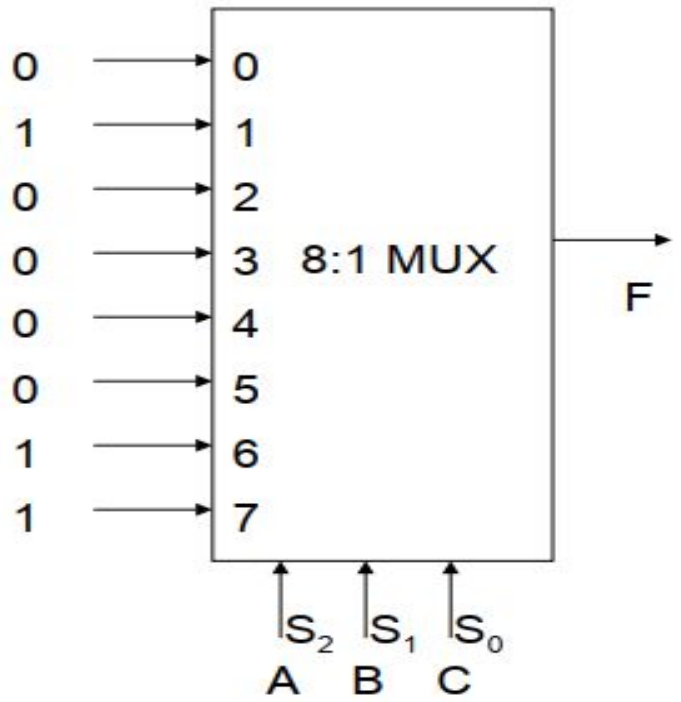


Mux Implementation of Inverter

Implementation Of Logic Functions using Multiplexer

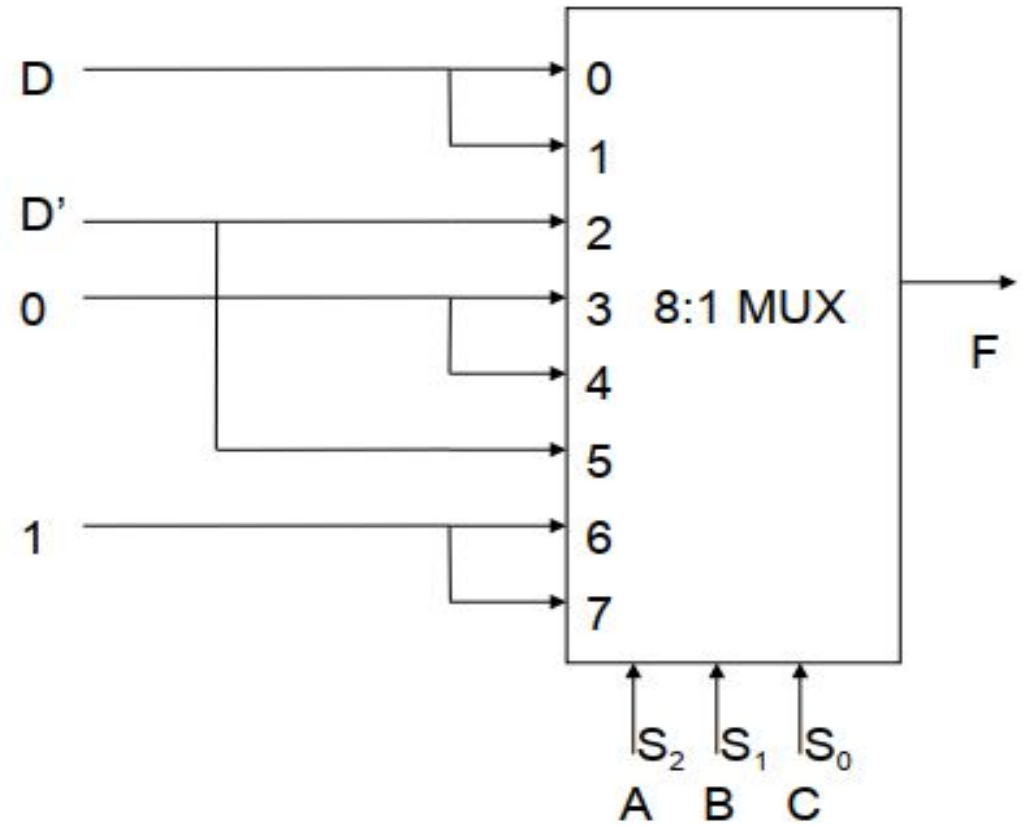
$f(a, b, c) = a'b'c + ab$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

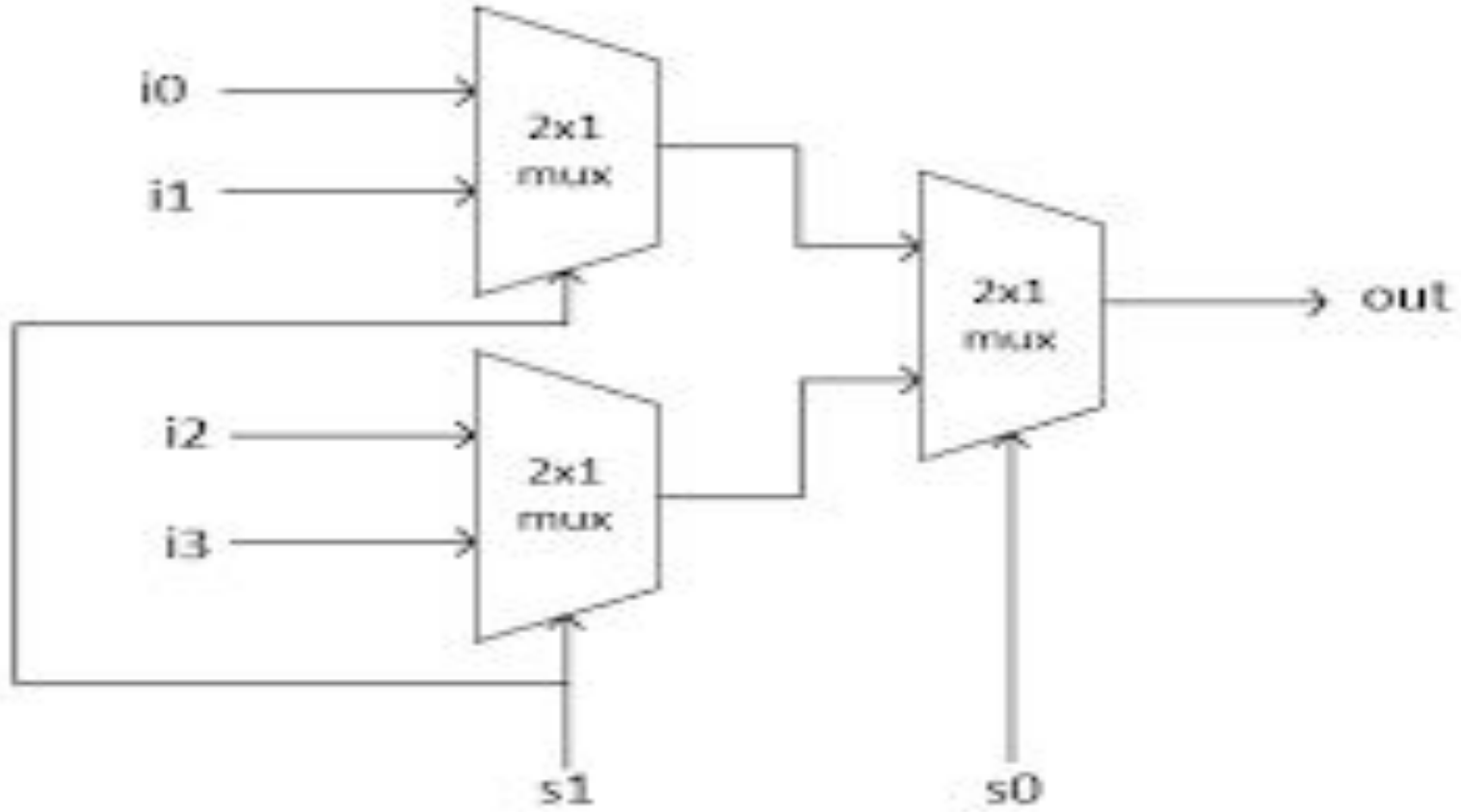


f(a, b, c) = F=A'B'C'D + A'B'CD + A'BC'D' + AB'CD + ABC'D' + ABC'D + ABCD' +ABCD

A	B	C	D	O	F
0	0	0	0	0	D
0	0	0	1	1	
0	0	1	0	0	D
0	0	1	1	1	
0	1	0	0	1	D'
0	1	0	1	0	
0	1	1	0	0	0
0	1	1	1	0	
1	0	0	0	0	0
1	0	0	1	0	
1	0	1	0	0	D'
1	0	1	1	1	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	1	1
1	1	1	1	1	

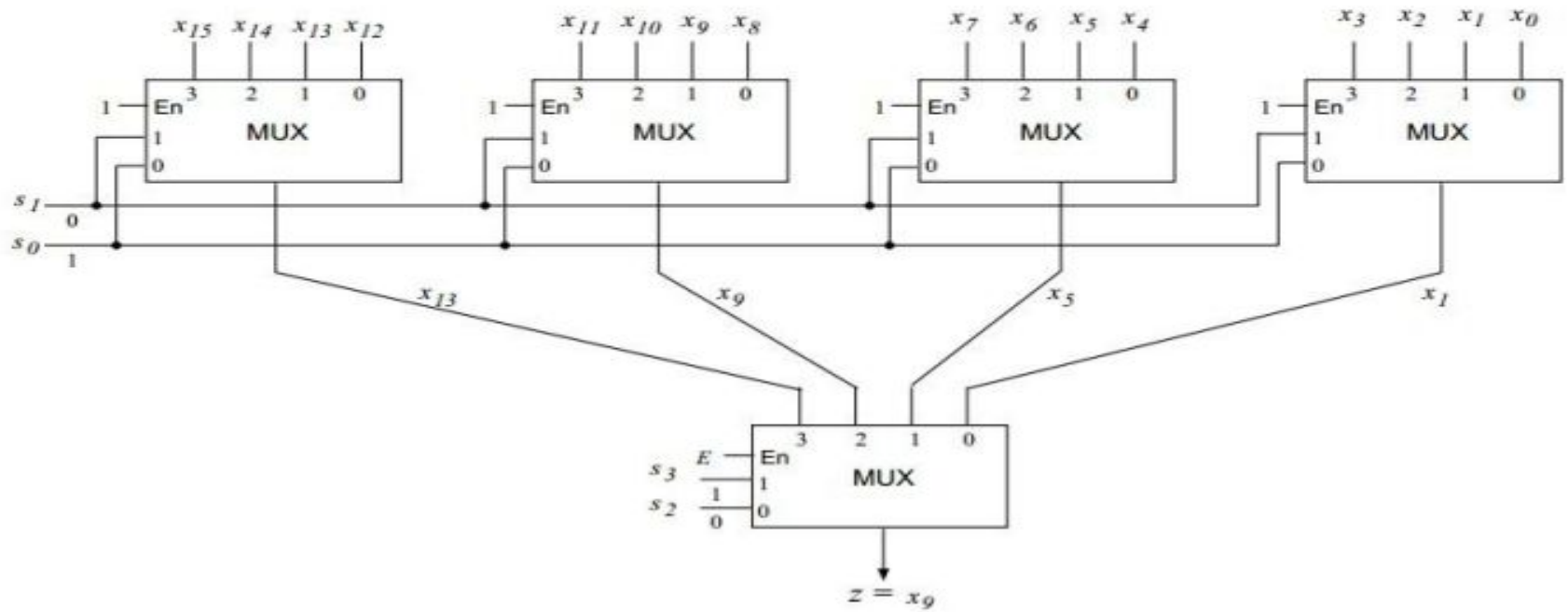


4:1 MUX using 2:1 MUX



Multiplexer Tree

- The Multiplexers with more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs.
- Below is a design of 16:1 MUX using 4 4:1 MUXs :-



Cascading of MUX

EXAMPLE 6–15

Use 74HC151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.

Solution

An expansion of two 74HC151s is shown in Figure 6–48. Four bits are required to select one of 16 data inputs ($2^4 = 16$). In this application the $\overline{Enable}$ input is used as the most significant data-select bit. When the MSB in the data-select code is LOW, the left 74HC151 is enabled, and one of the data inputs (D_0 through D_7) is selected by the other three data-select bits. When the data-select MSB is HIGH, the right 74HC151 is enabled, and one of the data inputs (D_8 through D_{15}) is selected. The selected input data are then passed through to the negative-OR gate and onto the single output line.

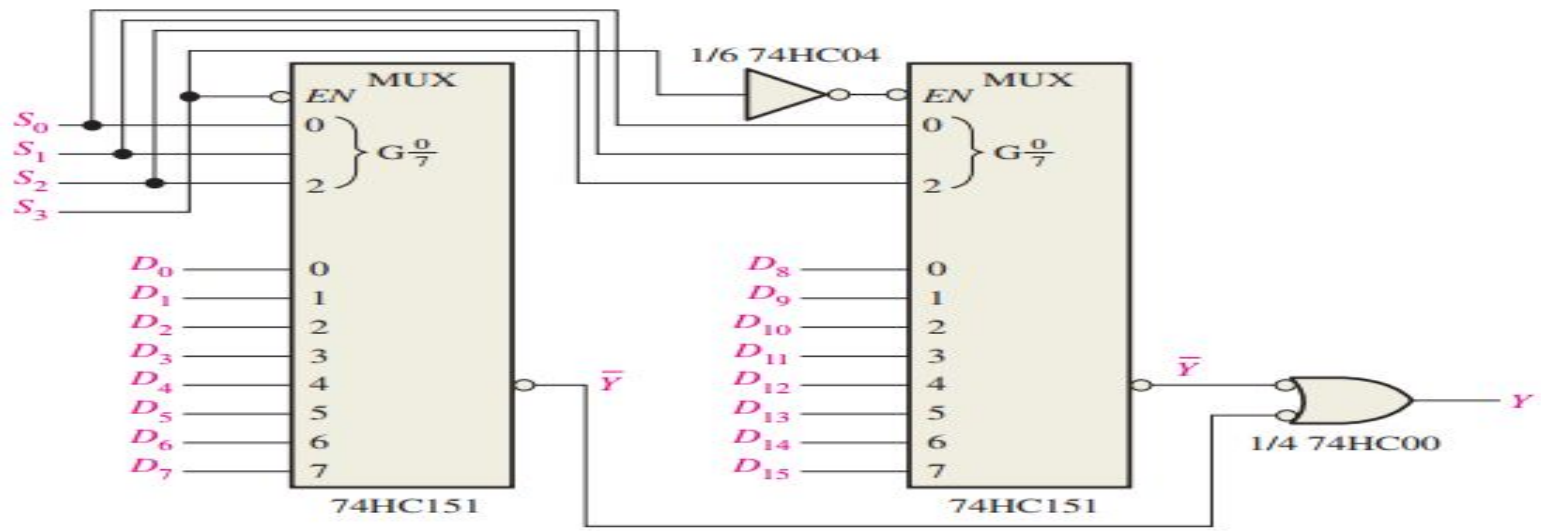


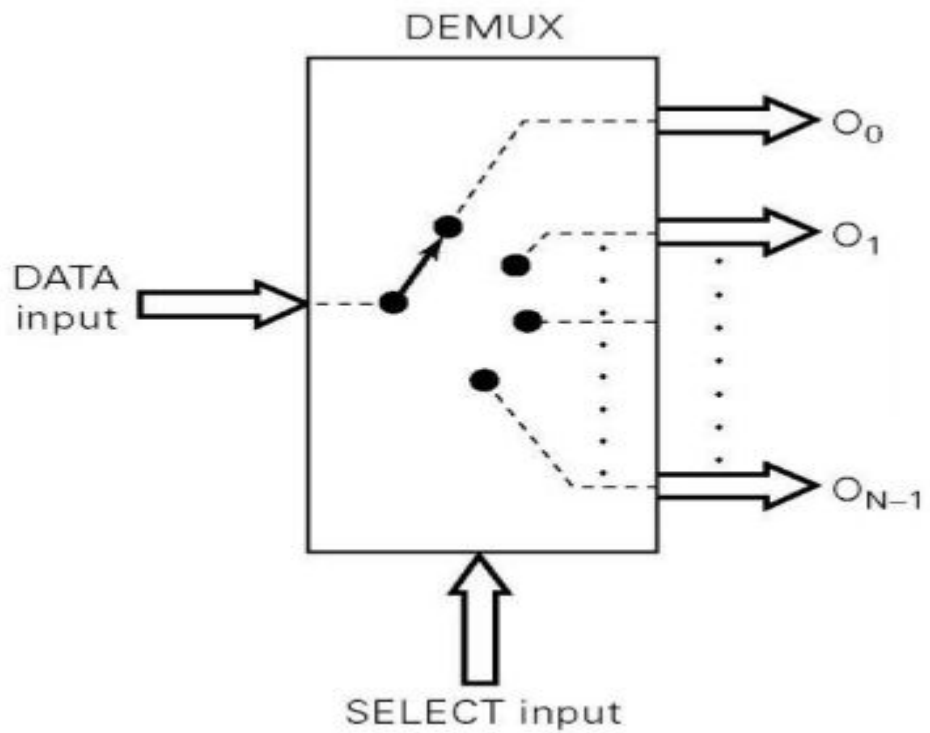
FIGURE 6–48 A 16-input multiplexer.

Demultiplexer (Data Distributor)

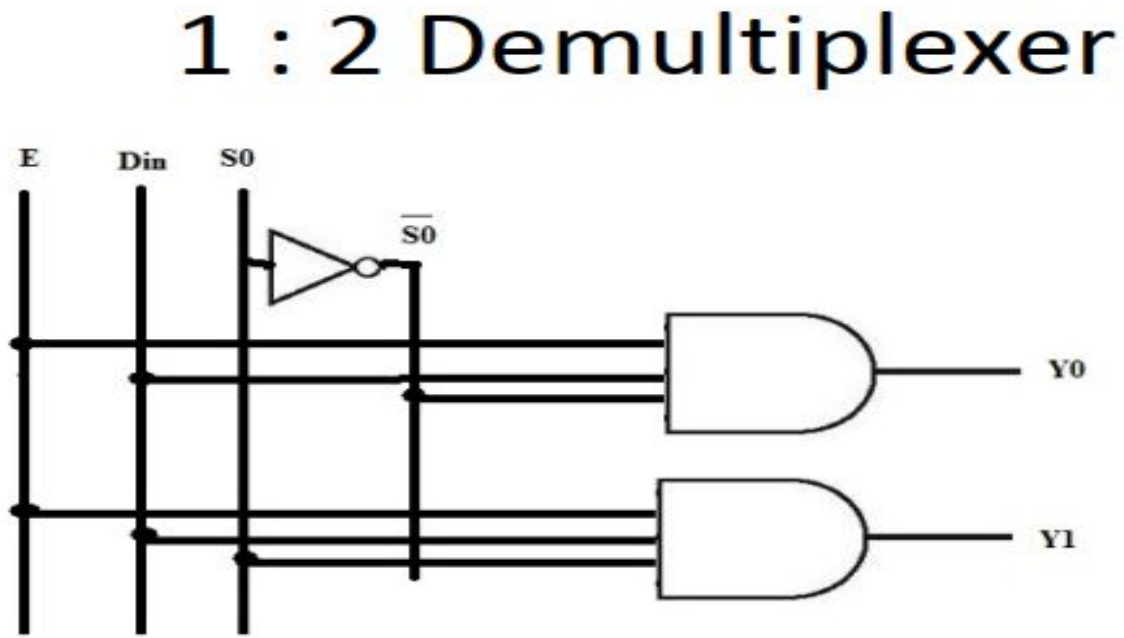
- Definition : A DEMULTIPLEXER (DEMUX) basically reverses the multiplexing function. It takes data from one line and distributes them to a given number of output lines. For this reason, the demultiplexers is also known as a data distributor.
- Single data input line.
- Some select line (less than the no. of output lines)
- Several output line
- If there are n data output lines and m select lines, then

$$2^m = n$$

Functional Diagram Of a Demultiplexer

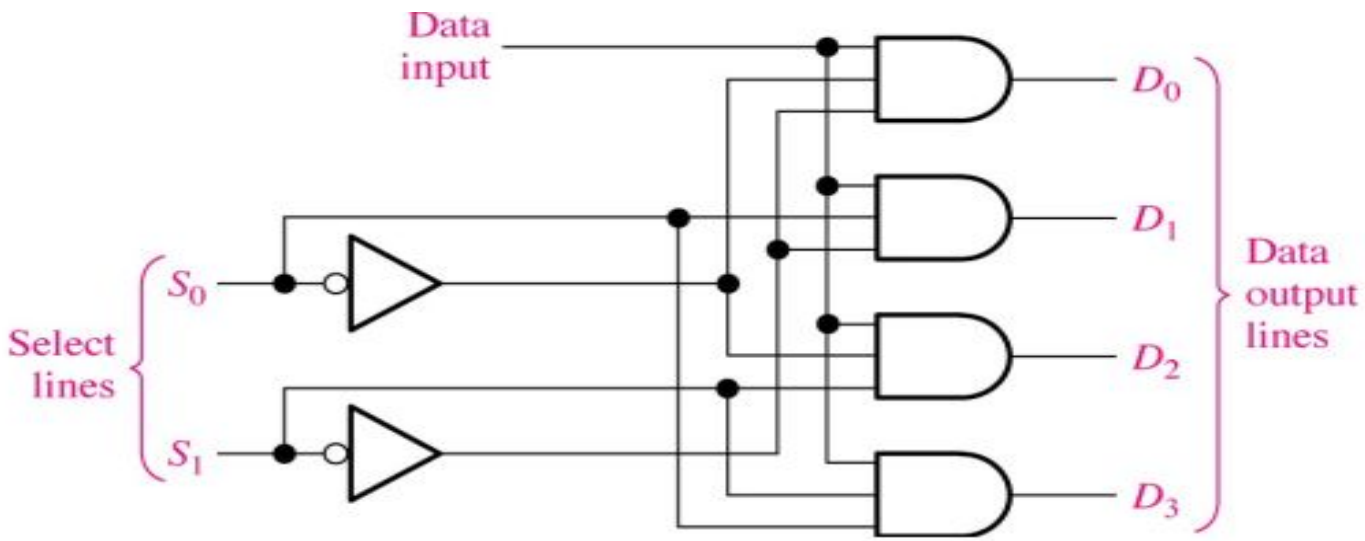


1 : 2 Demultiplexer



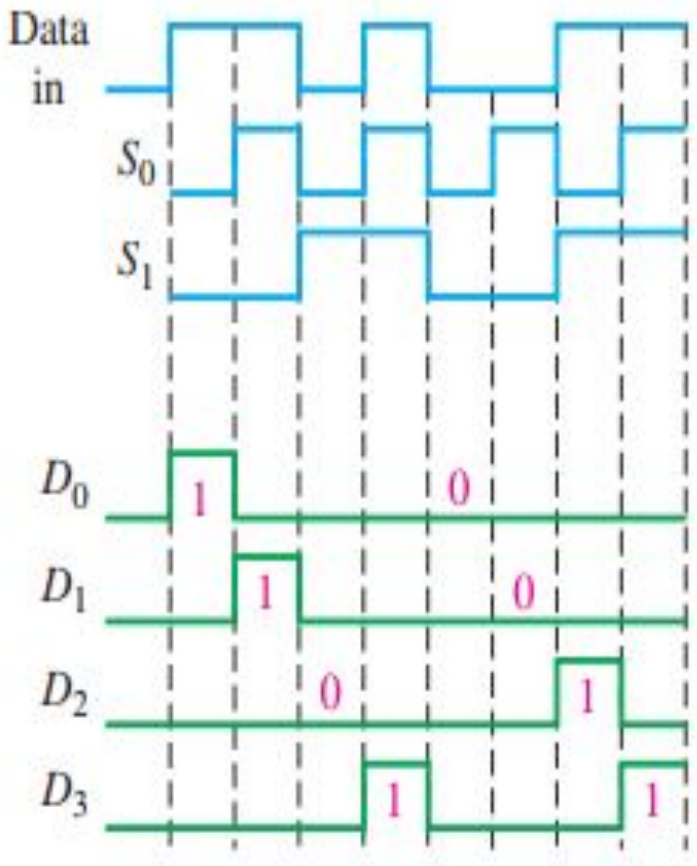
S_0	Y_0	Y_1
0	D	0
1	0	D

1 : 4 Demultiplexer



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

1:4 DMUX Timing Diagram



DECODERS

Definition

- An important part of the system which selects the cells to be read from and written into is the decoder.
  - Accepts a value and decodes it
  - Output corresponds to value of n inputs
- It also is called many-to-one decoder, a decoder matrix or simply a decoder.

The Basic Binary Decoder

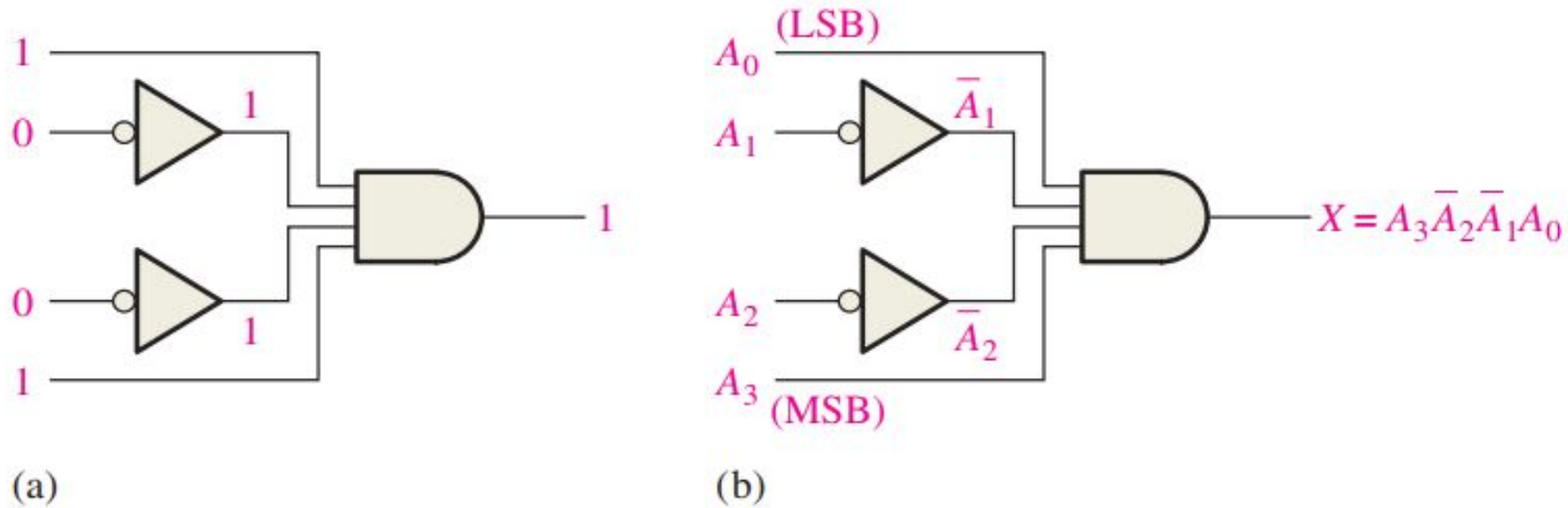
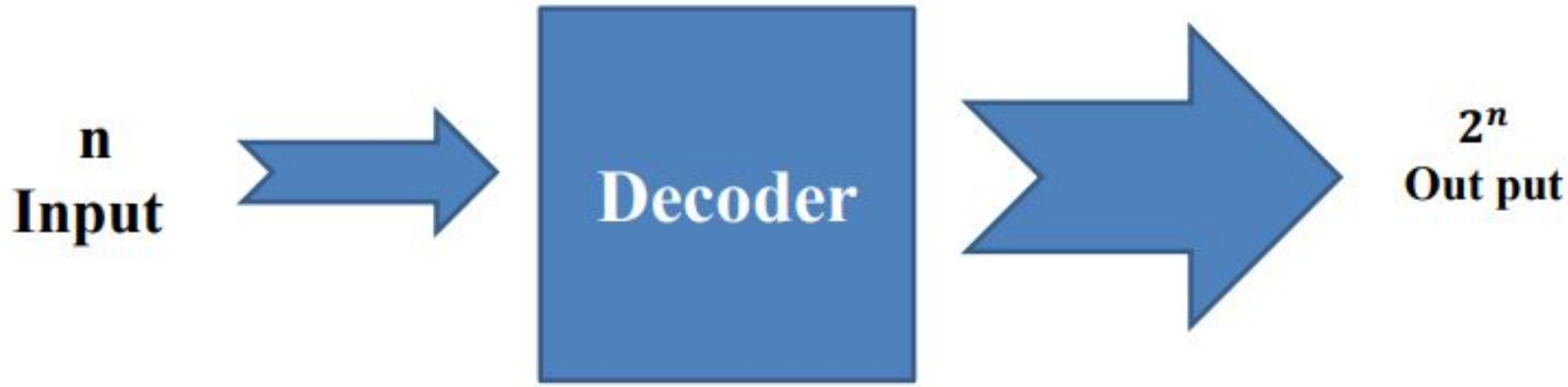


FIGURE 6–26 Decoding logic for the binary code 1001 with an active-HIGH output.

A decoder consists of

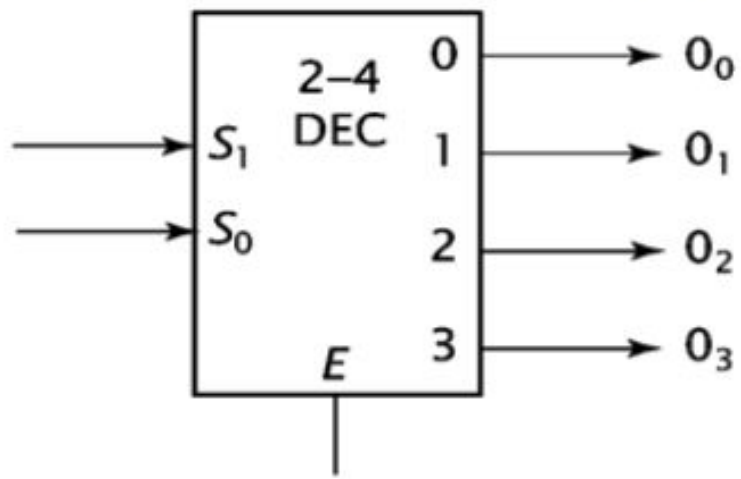
- Inputs (n)
- Outputs (2^n , numbered from 0 $\rightarrow$ 2^n - 1)
- Selectors / Enable (active high or active low)

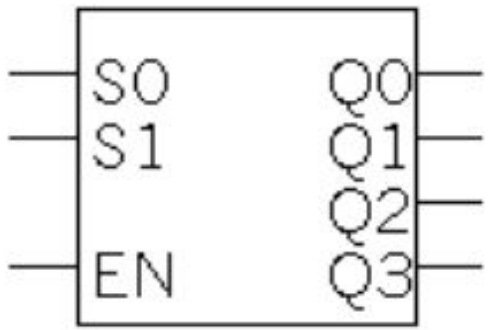
- The decoder has the characteristic that for each of the possible 2^n binary input numbers which can be taken by the n input cells, the matrix will have a unique one of its 2^n output lines selected.



n is the number of input flip-flop being decoded.

Number of AND gates is equal to number of output lines





$$Q0 = \overline{S1}\ \overline{S0}$$
$$Q1 = \overline{S1}\ S0$$
$$Q2 = S1\ \overline{S0}$$
$$Q3 = S1\ S0$$

Input
n=2

Output
$2^2=4$

What a decoder does

- A n-to-2^n decoder takes an n-bit input and produces 2^n outputs. The n inputs represent a binary number that determines which of the 2^n outputs is *uniquely* true.
- A 2-to-4 decoder operates according to the following truth table.
  - The 2-bit input is called S1S0, and the four outputs are Q0-Q3.
  - If the input is the binary number i, then output Qi is uniquely true.

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- For example, if the input S1 S0 = 10, then output Q_2 is true, and Q_0, Q_1, Q_3 are all false.
- This circuit "decodes" a binary number into a "one-of-four" code.

How can you build a 2-to-4 decoder?

- Follow the design procedures from last time! We have a truth table, so we can write equations for each of the four outputs (Q0-Q3), based on the two inputs (S0-S1).

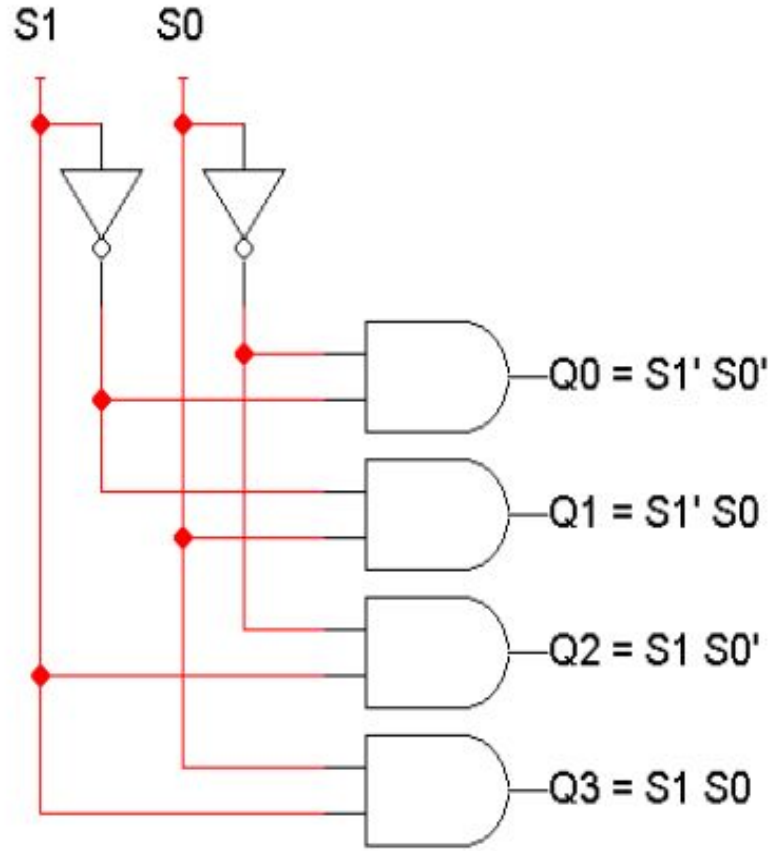
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- Here are the equations:

$$Q0 = \overline{S1}\ \overline{S0}$$
$$Q1 = S1\ \overline{S0}$$
$$Q2 = S1\ S0$$
$$Q3 = S1\ S0$$

A picture of a 2-to-4 decoder

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Enable inputs

- Many devices have an additional enable input, which is used to "activate" or "deactivate" the device.
- For a decoder,
  - EN=1 activates the decoder, so it behaves as specified earlier. Exactly one of the outputs will be 1.
  - EN=0 "deactivates" the decoder. By convention, that means *all* of the decoder's outputs are 0.
- We can include this additional input in the decoder's truth table:

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	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

- In this table, note that whenever EN=0, the outputs are always 0, *regardless* of inputs S1 and S0.

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	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

- We can abbreviate the table by writing x's in the input columns for S1 and S0.

EN	S1	S0	Q0	Q1	Q2	Q3
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

The 4-Bit Decoder

TABLE 6–4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

Decimal Digit	Binary Inputs				Decoding Function	Outputs														
	A_3	A_2	A_1	A_0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

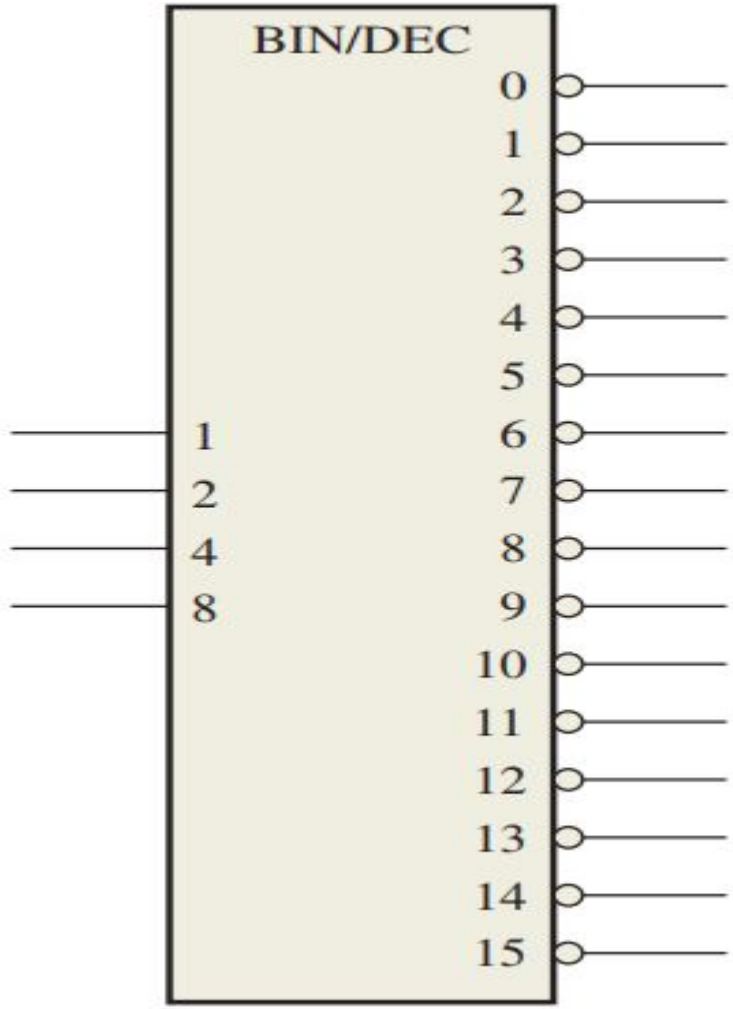


FIGURE 6–28 Logic symbol for a 4-line-to-16-line (1-of-16) decoder. Open file F06-28 to verify operation.

EXAMPLE 6–9

A certain application requires that a 5-bit number be decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

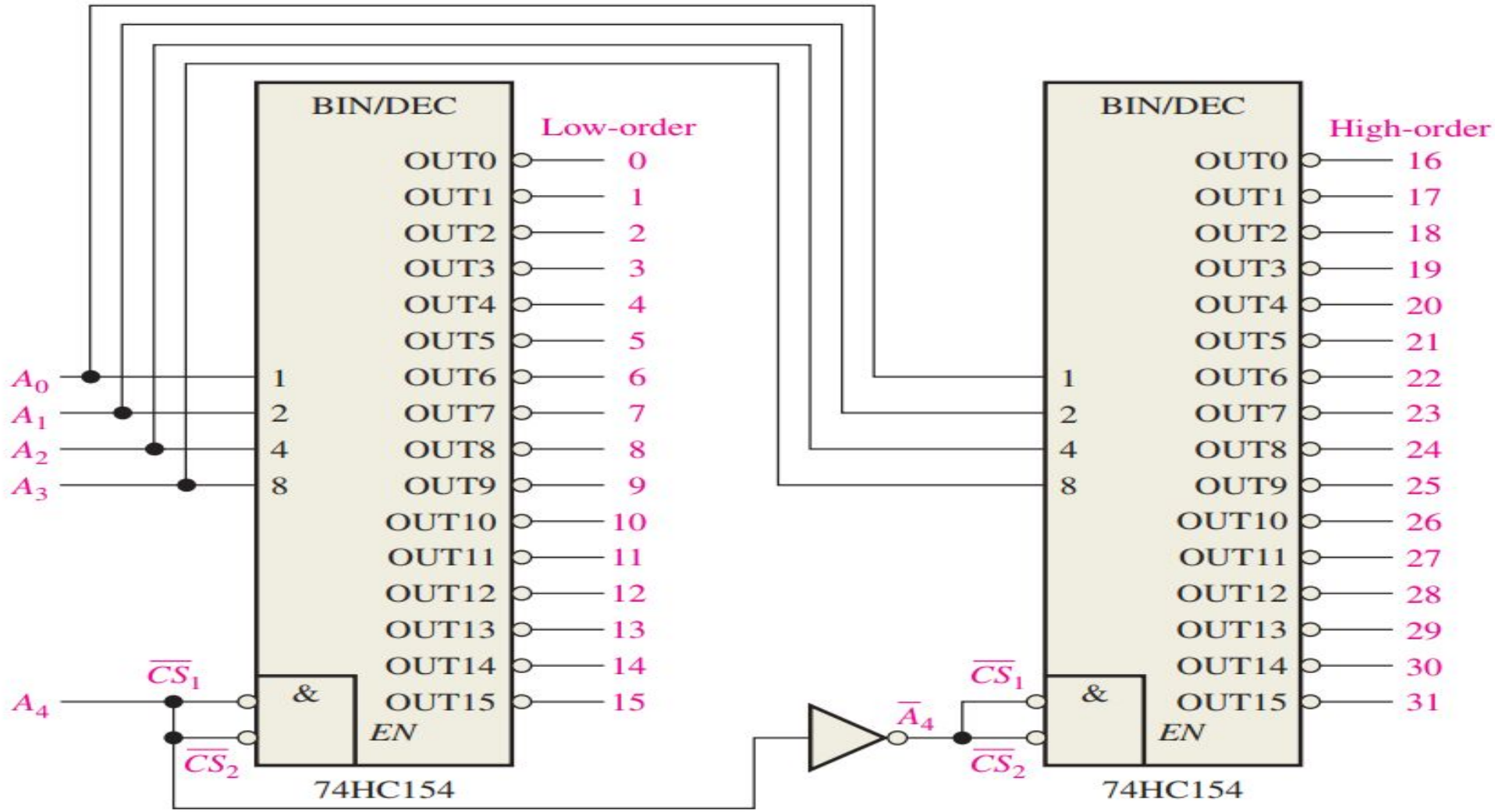
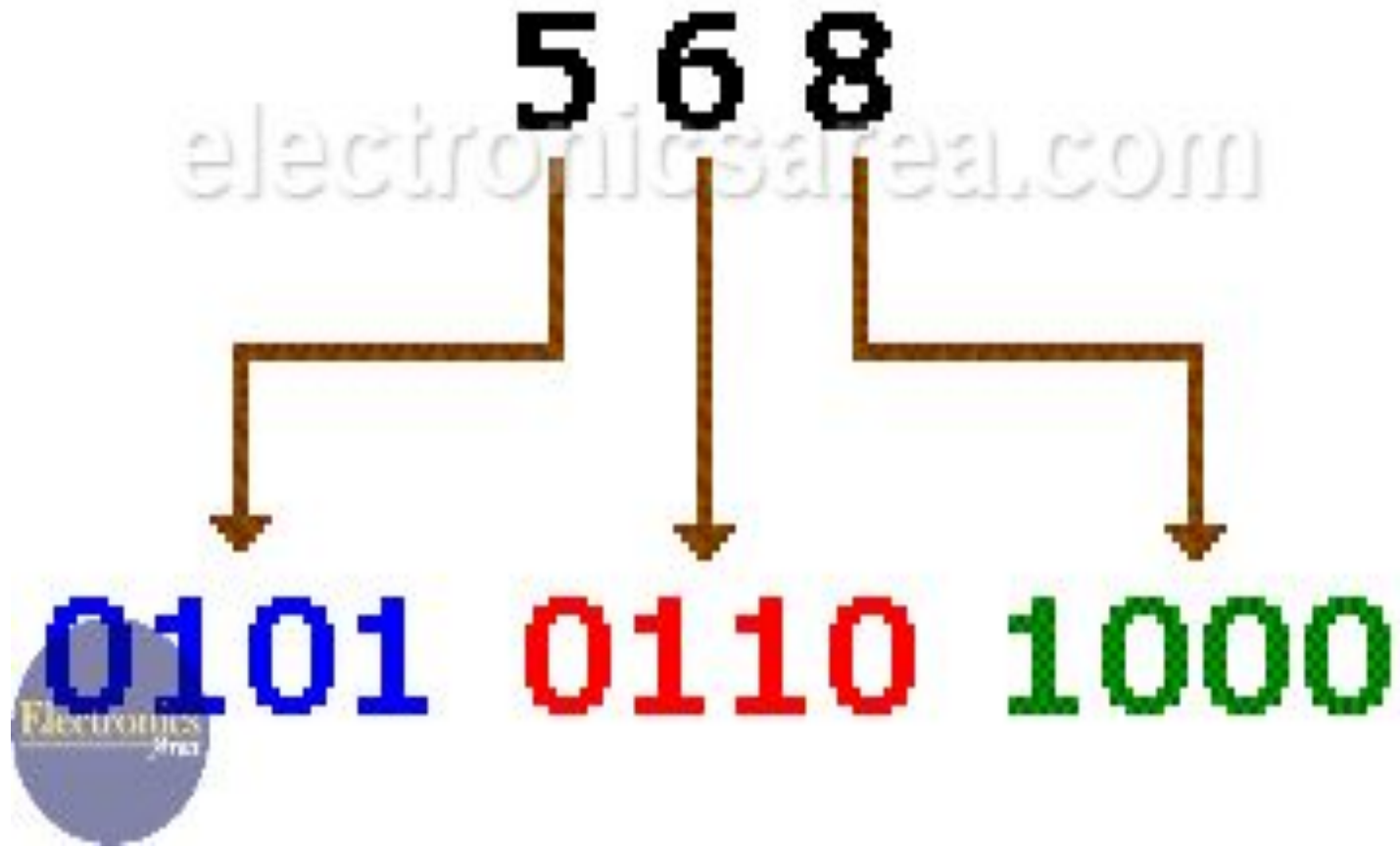


FIGURE 6–30 A 5-bit decoder using 74HC154s.

BCD (Binary Coded Decimal)

- Binary coded decimal (**BCD**) is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base-10) numeral.
- The four-bit **BCD** code for any particular single base-10 digit is its representation in binary notation.
- 4bit can make 16 combination but in BCD we only uses start 10 combinations.


568
0101 0110 1000

Number	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010 – invalid BCD number	0001 0000
11	1011 – invalid BCD number	0001 0001

The BCD-to-Decimal Decoder

TABLE 6–5

BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

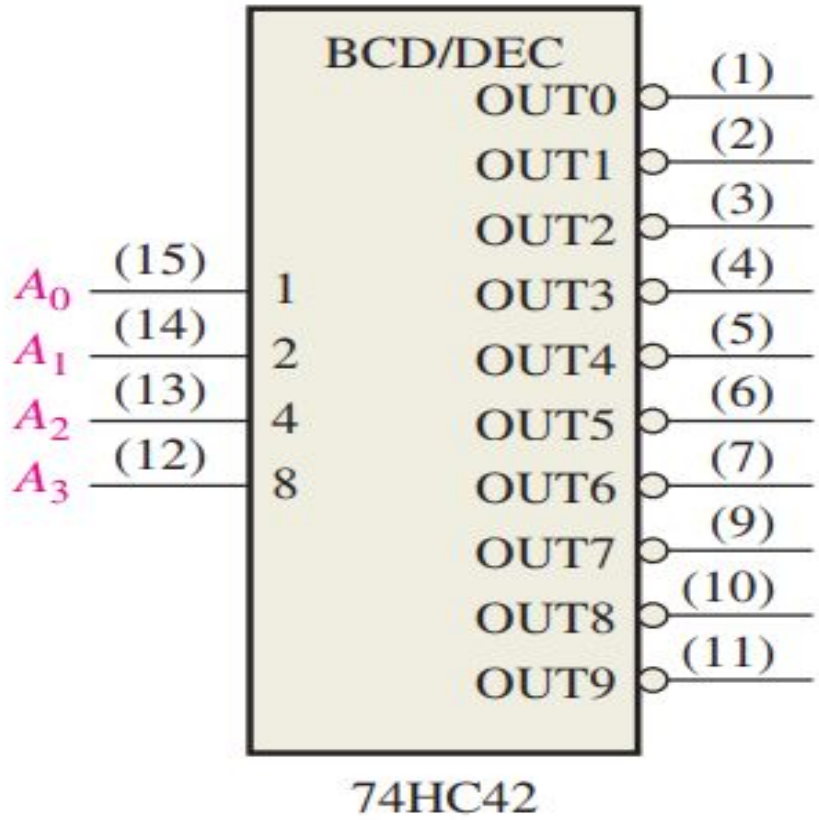
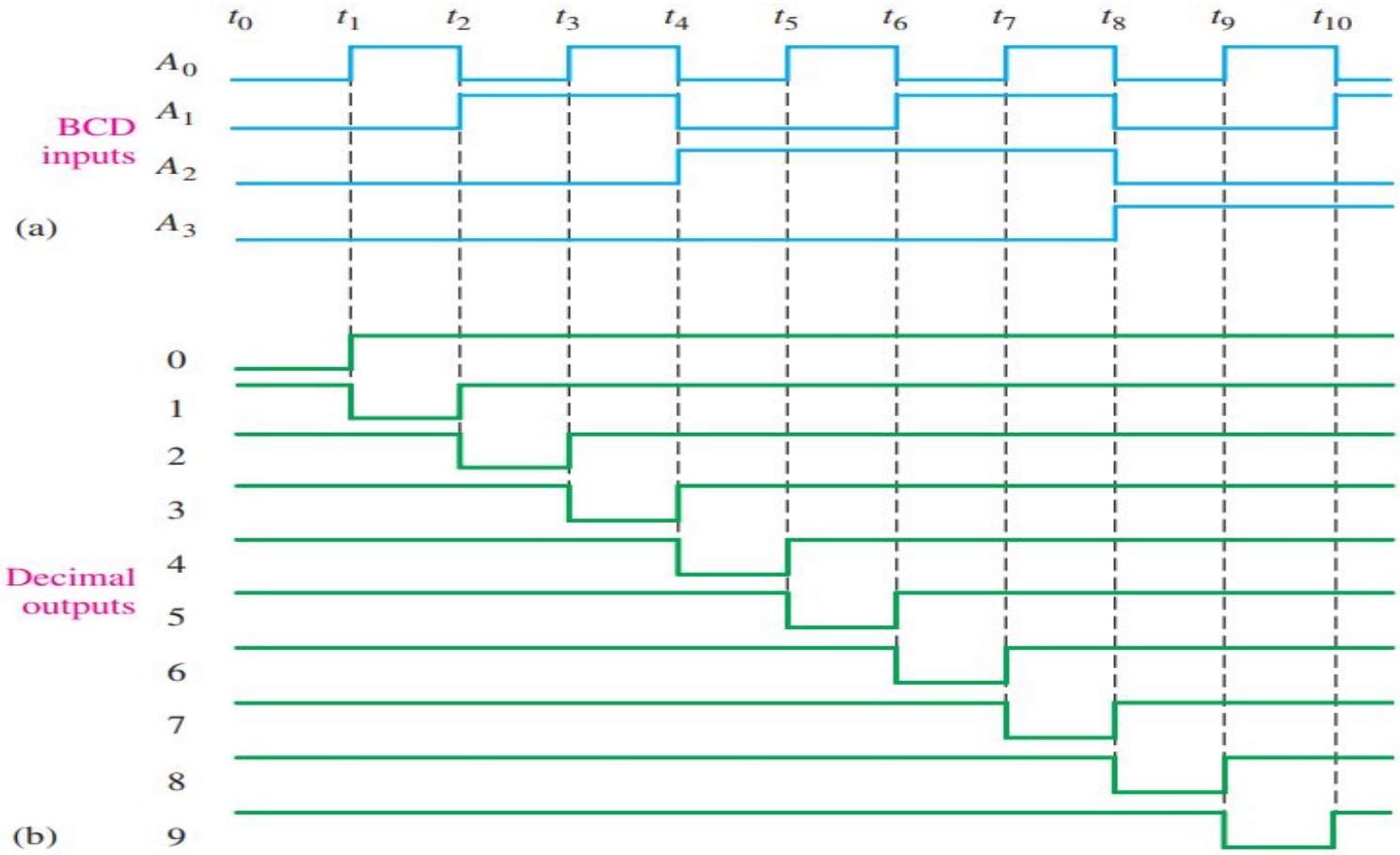


FIGURE 6–31 The 74HC42 BCD-to-decimal decoder.

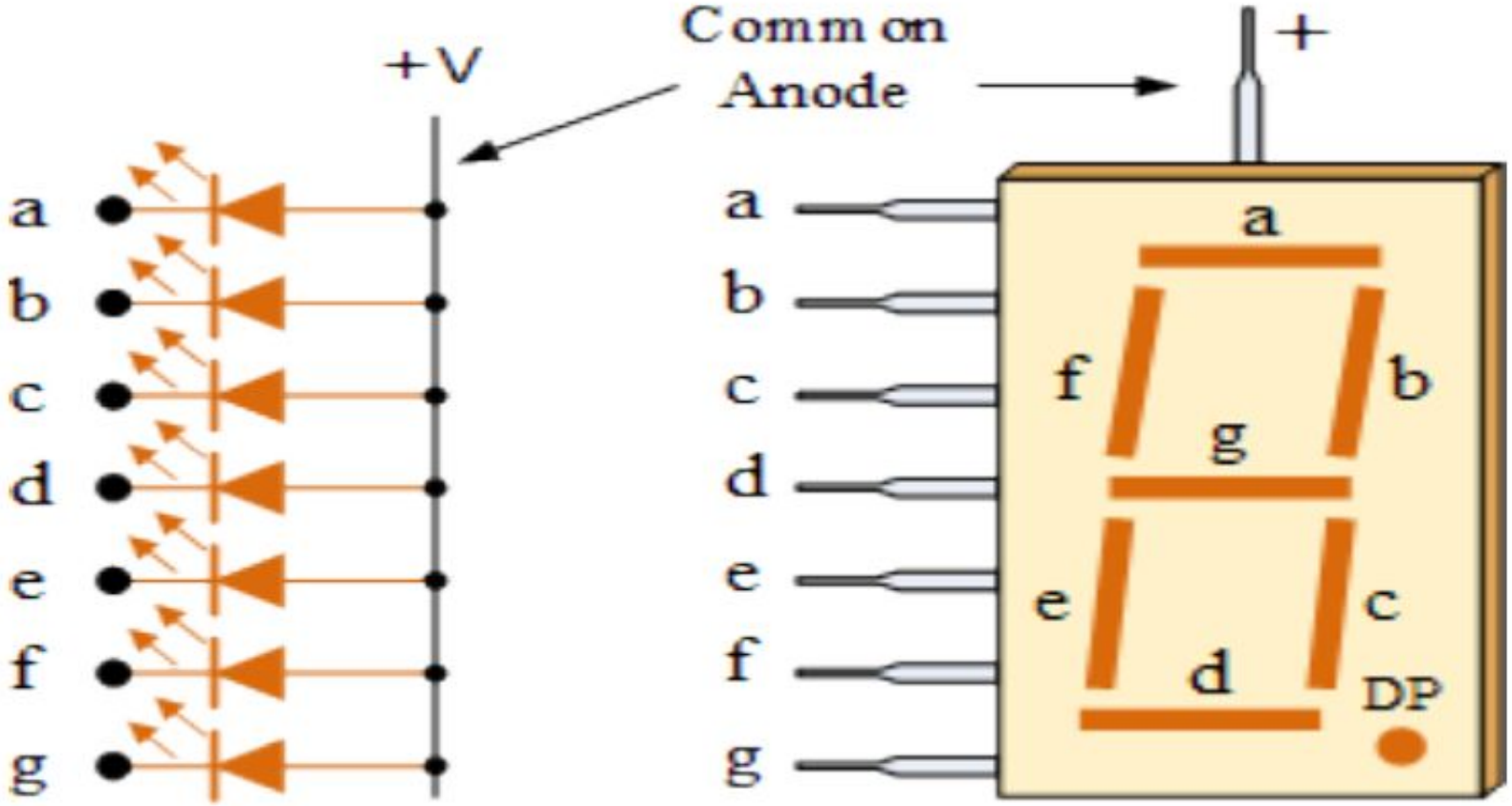
BCD-Decimal Decoder Waveform



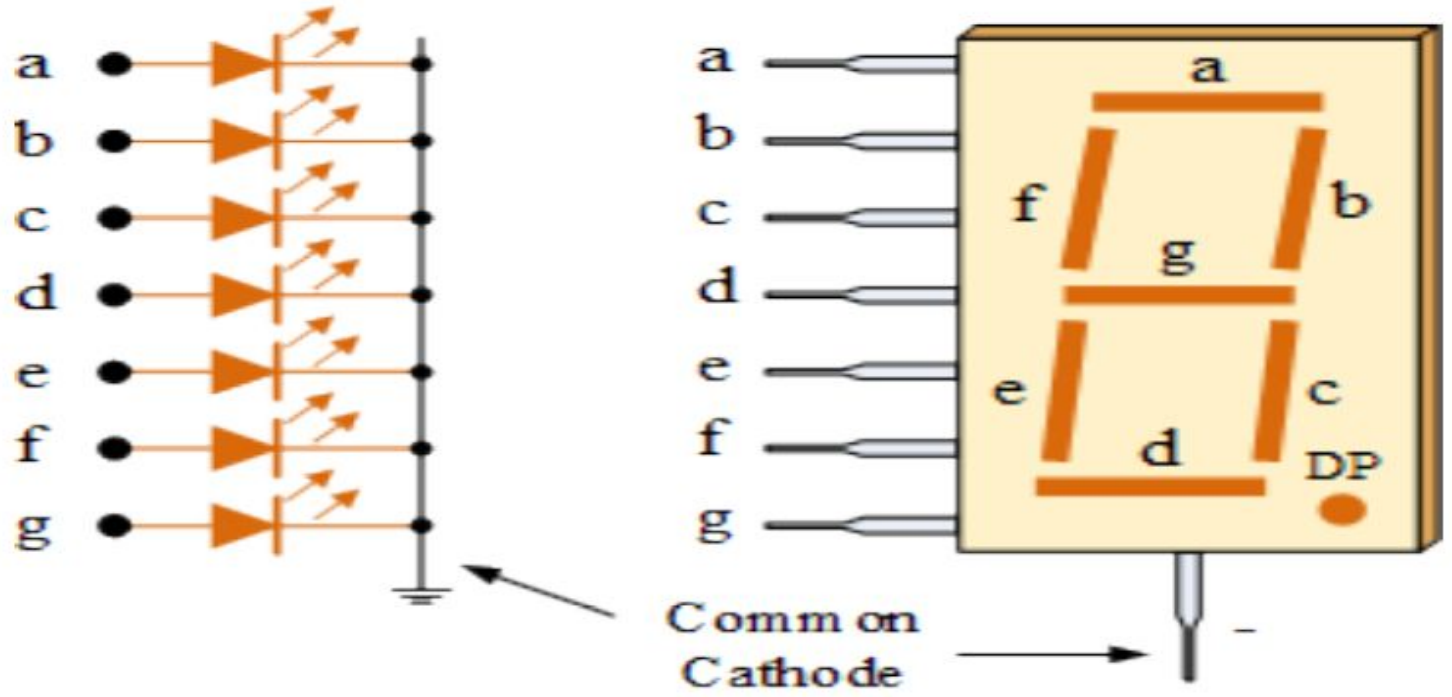
Seven Segment Display

- 7 LEDs (light emitting diodes), each one controlled by an input
- Two types
  - Common Anode
  - Common Cathode

Common Anode (CA)

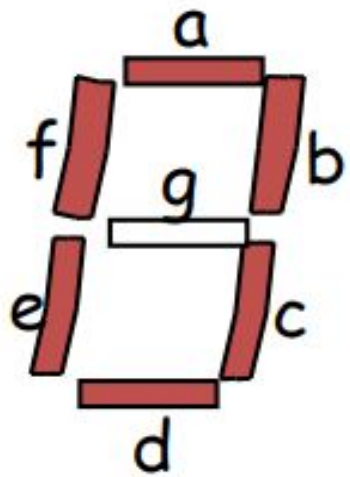


Common Cathode (CC)



Seven Segment Decoder

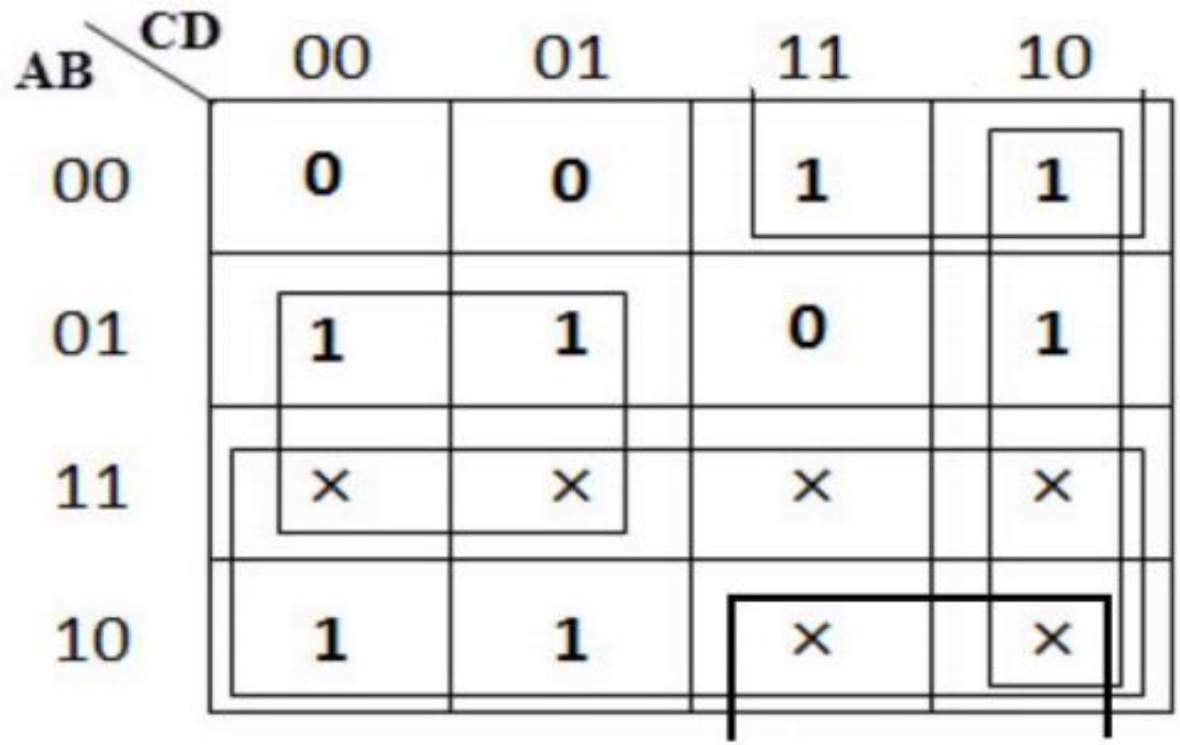
- Input is a 4-bit BCD code → 4 inputs (A, B, C, D)
- Output is a 7-bit code (a,b,c,d,e,f,g) that allows for the decimal equivalent to be displayed
- Example:
  - Input: 0000BCD
  - Output: 1111110 (a=b=c=d=e=f=1, g=0)



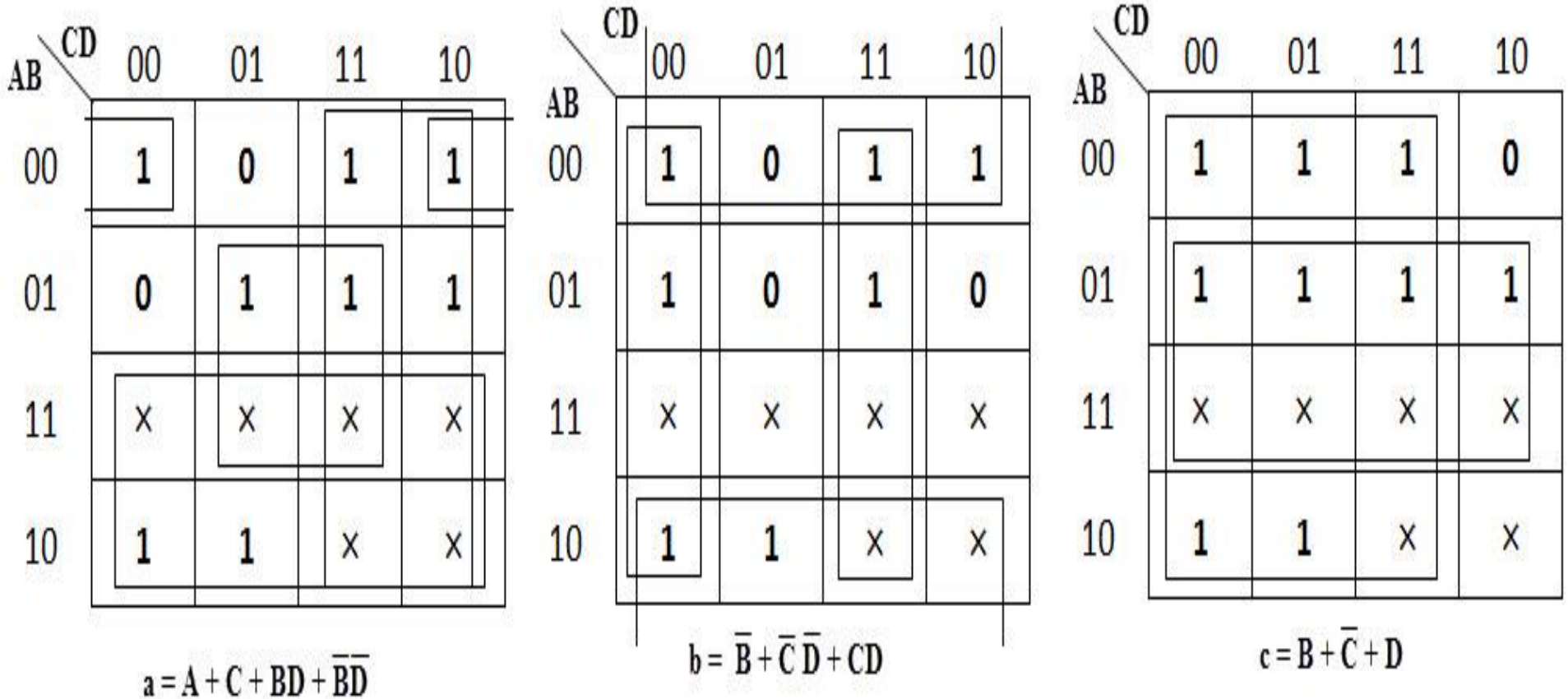
Truth Table for 7 Segment Decoder

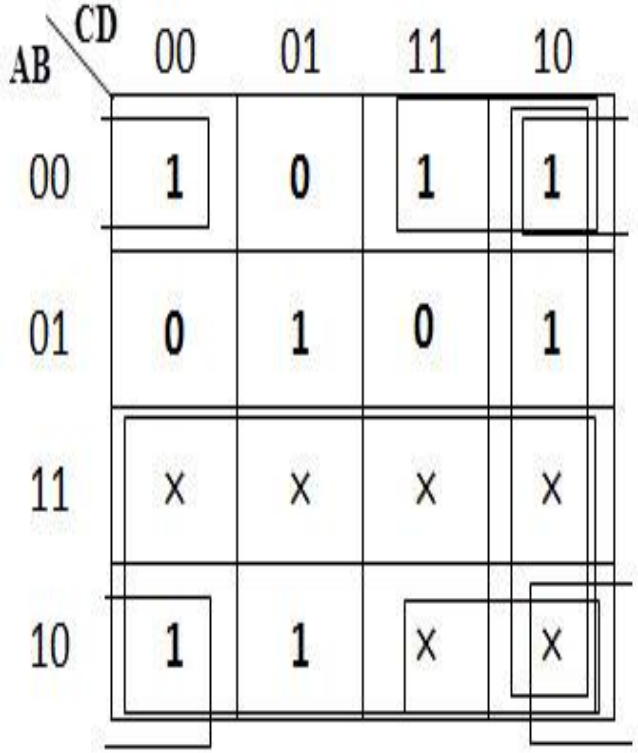
Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	

Expression for segment g

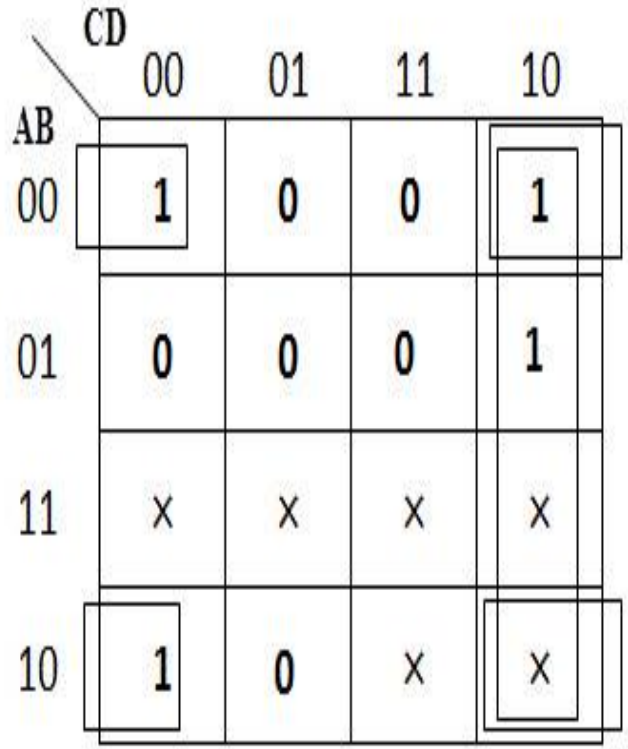


$$g = \overline{B}\,C + C\,\overline{D} + B\,\overline{C} + A$$

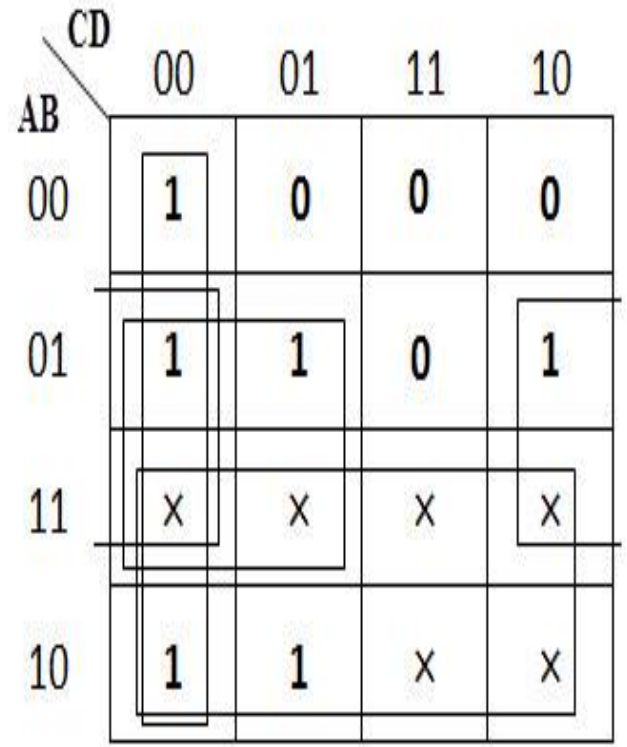

CD
AB
00 01 11 10
00 1 0 1 1
01 0 1 1 1
11 X X X X
10 1 1 X X
a = A + C + BD + $\overline{B}\,\overline{D}$
CD
AB
00 01 11 10
00 1 0 1 1
01 1 0 1 0
11 X X X X
10 1 1 X X
b = $\overline{B}$ + $\overline{C}\,\overline{D}$ + CD
CD
AB
00 01 11 10
00 1 1 1 0
01 1 1 1 1
11 X X X X
10 1 1 X X
c = B + $\overline{C}$ + D



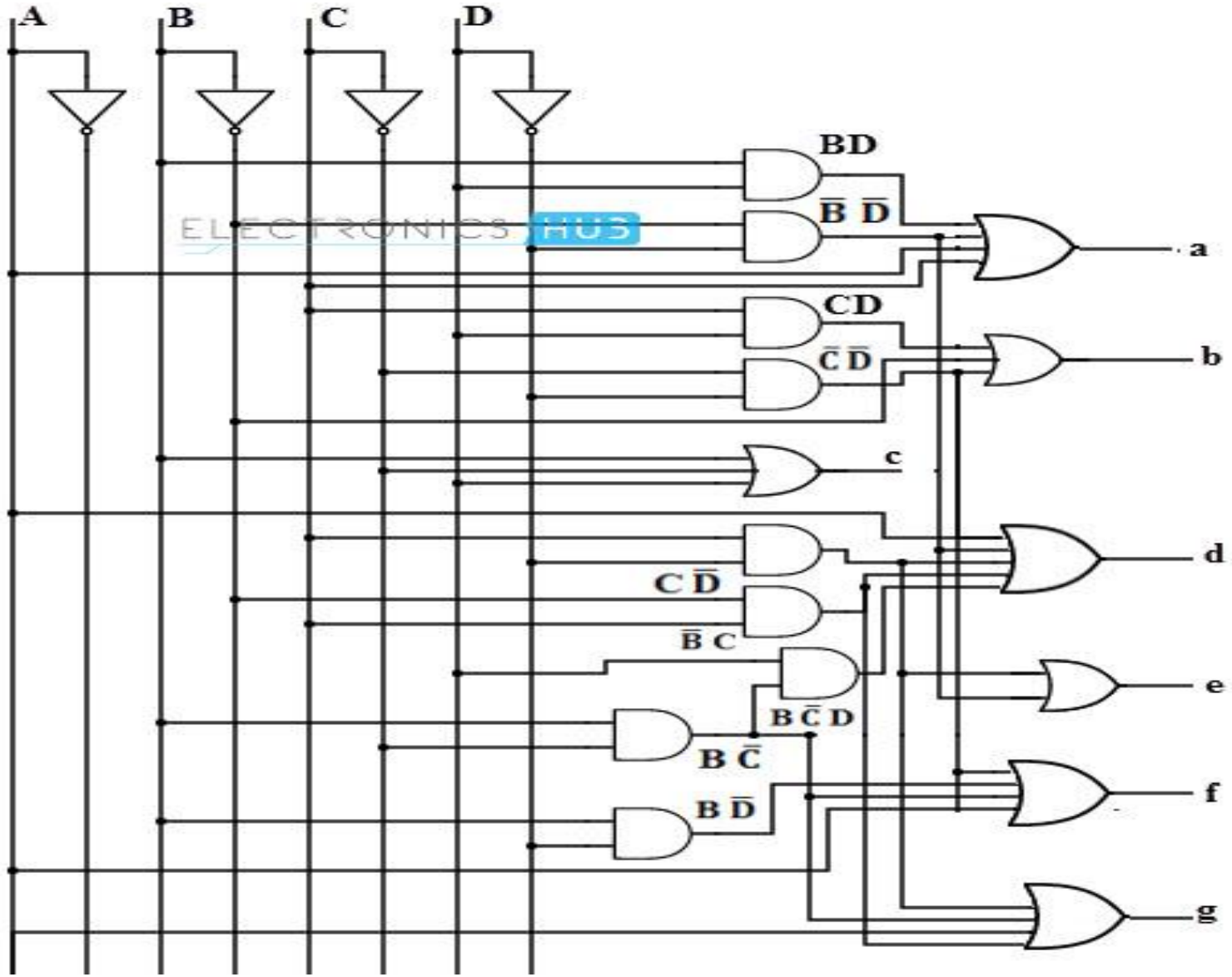
$d = \bar{B}\bar{D} + C\bar{D} + B\bar{C}D + \bar{B}C + A$



$e = \bar{B}\bar{D} + C\bar{D}$



$f = A + \bar{C}\bar{D} + B\bar{C} + B\bar{D}$


A
B
C
D
BD
$\bar{B}\,\bar{D}$
a
CD
$\bar{C}\,\bar{D}$
b
c
d
$C\,\bar{D}$
$\bar{B}\,C$
e
$B\,\bar{C}\,D$
$B\,\bar{C}$
f
$B\,\bar{D}$
g

ENCODERS

Encoders

An **encoder** is a combinational logic circuit that essentially performs a "reverse" decoder function. An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, 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*.

The Decimal-to-BCD Encoder

This type of encoder has ten inputs—one for each decimal digit—and four outputs corresponding to the BCD code, as shown in Figure 6–36. This is a basic 10-line-to-4-line encoder.

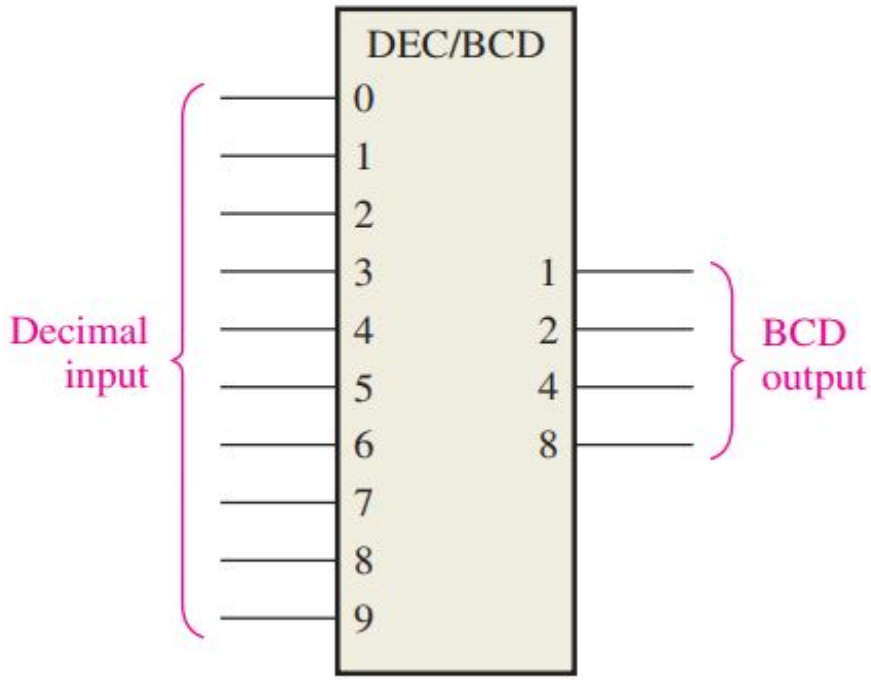


FIGURE 6–36 Logic symbol for a decimal-to-BCD encoder.

TABLE 6–6

Decimal Digit	BCD Code			
	A_3	A_2	A_1	A_0
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
8	1	0	0	0
9	1	0	0	1

The BCD (8421) code is listed in Table 6–6. From this table you can determine the relationship between each BCD bit and the decimal digits in order to analyze the logic. For instance, the most significant bit of the BCD code, A_3, is always a 1 for decimal digit 8 or 9. An OR expression for bit A_3 in terms of the decimal digits can therefore be written as

$$A_3 = 8 + 9$$

Bit A_2 is always a 1 for decimal digit 4, 5, 6 or 7 and can be expressed as an OR function as follows:

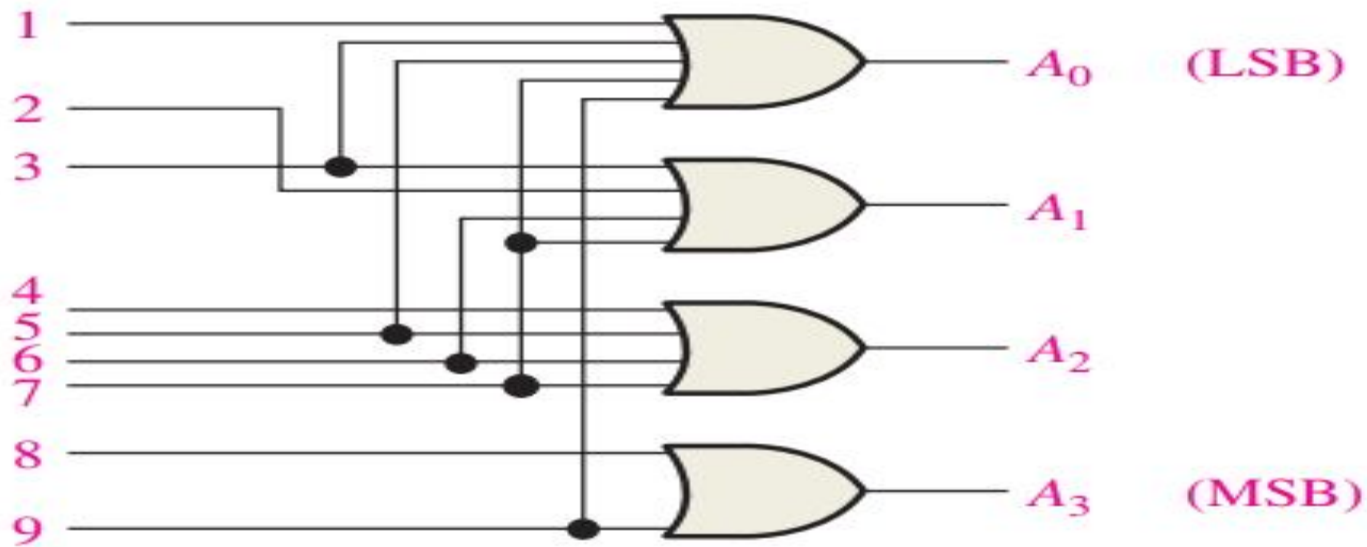
$$A_2 = 4 + 5 + 6 + 7$$

Bit A_1 is always a 1 for decimal digit 2, 3, 6, or 7 and can be expressed as

$$A_1 = 2 + 3 + 6 + 7$$

Finally, A_0 is always a 1 for decimal digit 1, 3, 5, 7, or 9. The expression for A_0 is

$$A_0 = 1 + 3 + 5 + 7 + 9$$



The Decimal-to-BCD Priority Encoder

This type of encoder performs the same basic encoding function as previously discussed. A **priority encoder** also offers additional flexibility in that it can be used in applications that require priority detection. The priority function means that the 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 6 and the 3 inputs are both active, the BCD output is 0110 (which represents decimal 6).

An Application

The ten decimal digits on a numeric keypad must be encoded for processing by the logic circuitry. In this example, when one of the keys is pressed, the decimal digit is encoded to the corresponding BCD code. Figure 6–39 shows a simple keyboard encoder arrangement using a priority encoder. The keys are represented by ten 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.

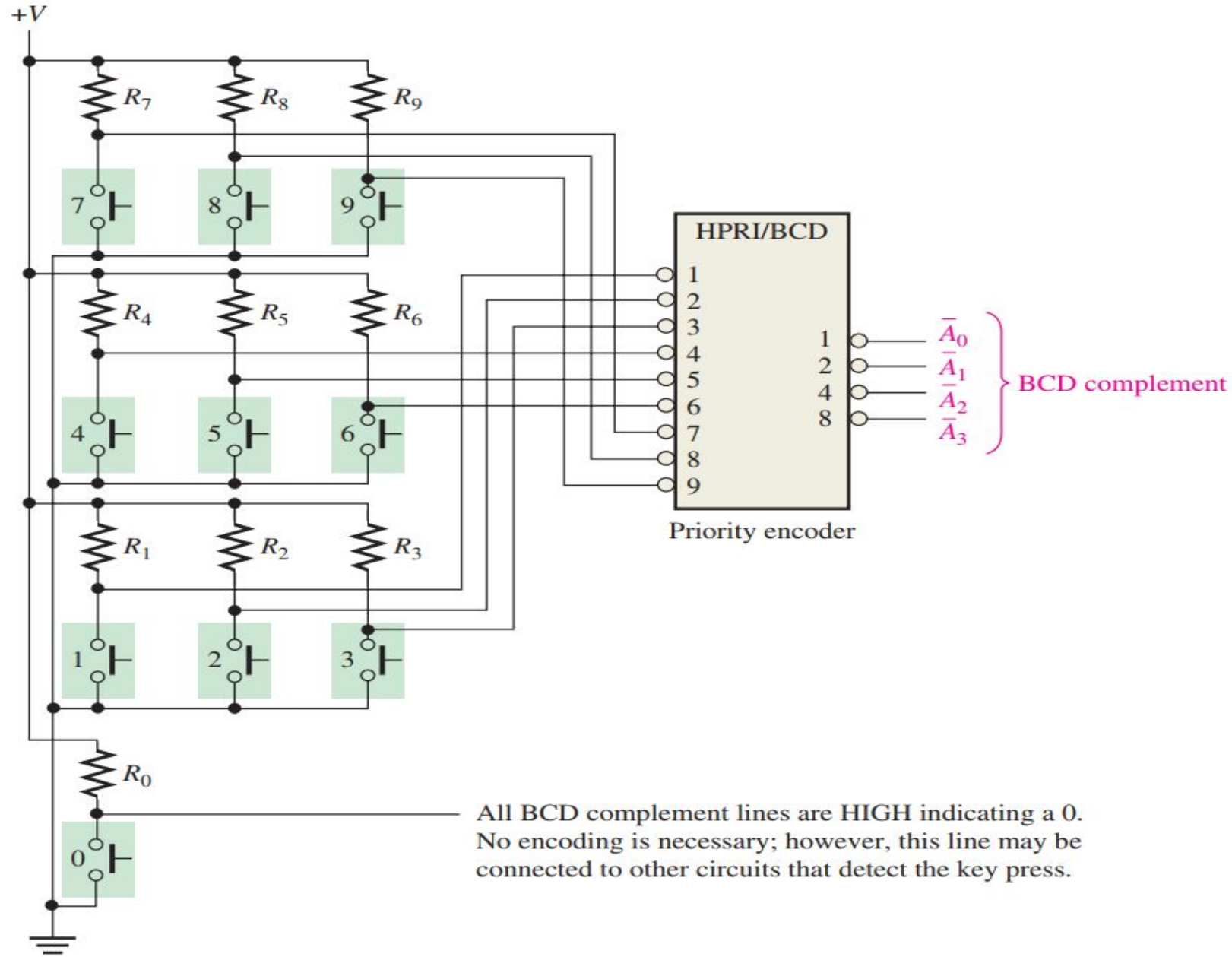


FIGURE 6–39 A simplified keyboard encoder.

Code Converters

In this section, we will examine some methods of using combinational logic circuits to convert from one code to another.

After completing this section, you should be able to

- Explain the process for converting BCD to binary
- Use exclusive-OR gates for conversions between binary and Gray codes

BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8} \quad \underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6–7.

TABLE 6–7

Binary representations of BCD bit weights.

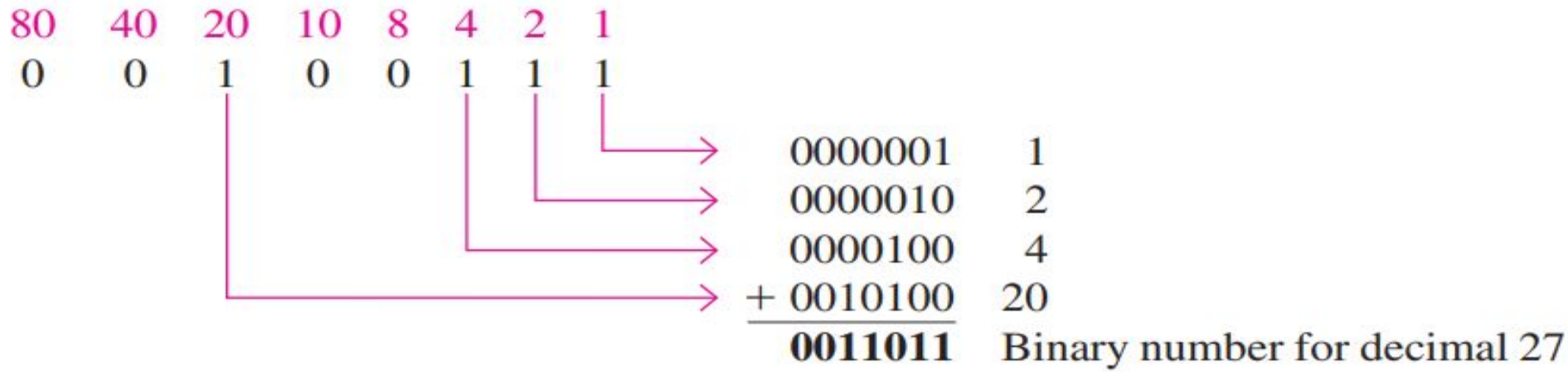
BCD Bit	BCD Weight	(MSB) Binary Representation (LSB)						
		64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

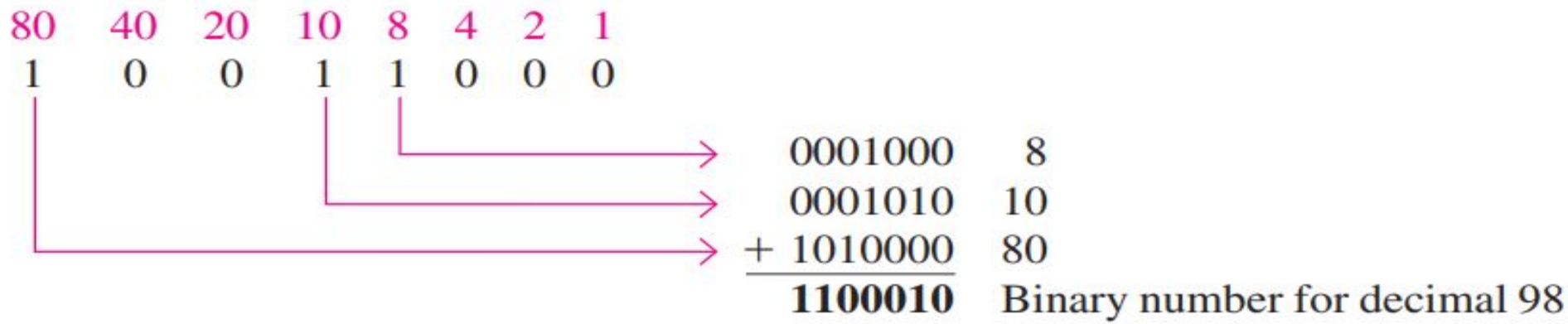
EXAMPLE 6–12

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.





Binary-to-Gray and Gray-to-Binary Conversion

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions. Programmable logic devices (PLDs) can also be programmed for these code conversions. Figure 6–40 shows a 4-bit binary-to-Gray code converter, and Figure 6–41 illustrates a 4-bit Gray-to-binary converter.