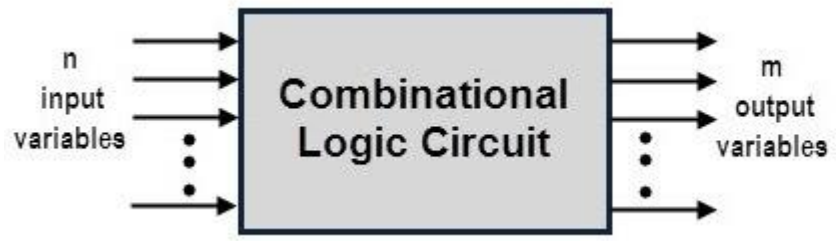
Module 3

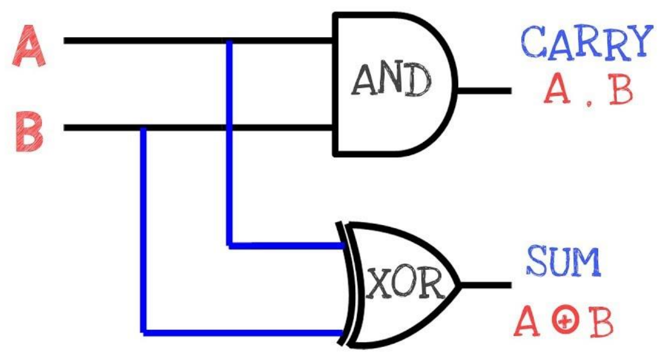
Dr.N.Subhashini
Associate Professor, SENSE
VIT University

Combinational Circuits

- Outputs at any time is a function of only present inputs



Half Adder



Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
assign sum=a^b;
assign carry=a&b;
```

```
module ha_beh(sum,carry, a,b);
input a,b;
output reg sum,carry;
always@(a or b)
begin
sum = a ^ b;
carry = a&b;
end
endmodule
```

```
module ha_beh1(sum,carry,a,b);
input a,b;
output reg sum,carry;
always@(a or b)
 case ({a, b})
     2'b00: begin sum = 0; carry = 0; end
     2'b01: begin sum = 1; carry = 0; end
     2'b10: begin sum = 1; carry = 0; end
     2'b11: begin sum = 0; carry = 1; end

   endcase
endmodule
```

```
module ha_beh2( A, B,  S, C);
 input wire A, B;
output reg S, C;

always @(A or B )
begin
 if(A==0 && B==0)
  begin
   S=0;
   C=0;
  end
  else if(A==0 && B==1)
  begin
   S=1;
   C=0;
  end
  else if(A==1 && B==0 )
  begin
   S=1;
   C=0;
  end
 else if(A==1 && B==1)
  begin
   S=0;
   C=1;
  end

end
endmodule
```

****

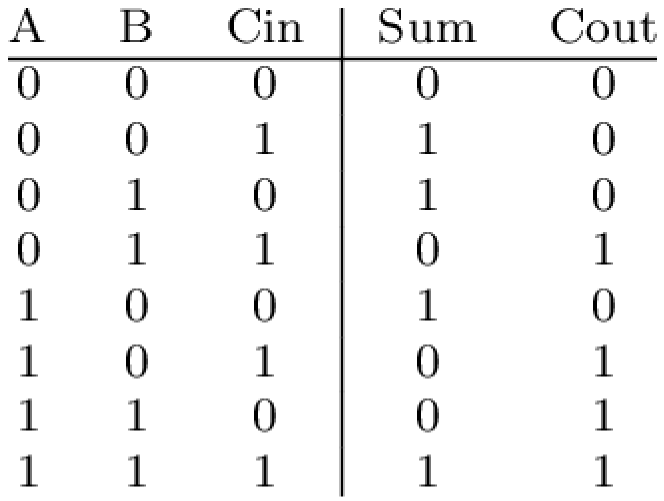
Test Bench- Half adder

****

```
module ha_tb;
reg a,b;
wire s,c;
halfadder_gate dut(.sum(s),.carry(c),.a(a),.b(b));
initial begin
$monitor(s,c,a,b);
a=1'b0;
b=1'b0;
#100;
a=1'b0;
b=1'b1;
#100;
a=1'b1;
b=1'b0;
#100;
a=1'b1;
b=1'b1;
#100;
end
endmodule
```

$monitor- We use this function to monitor the value of signals in our testbench and display a message whenever one of these signals changes state.

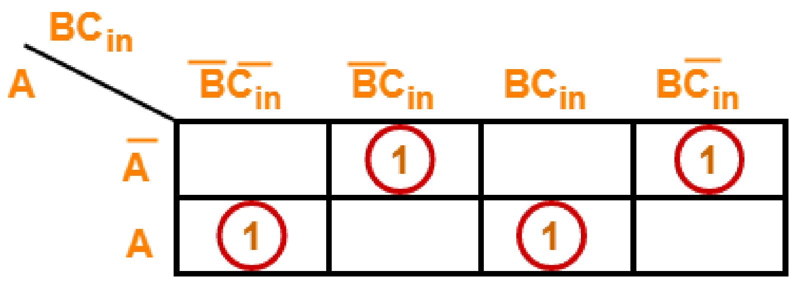
Full adder



A	B	Cin	Sum	Cout
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

```
assign sum=a^b^c;
assign carry=a&b|b&c|c&a;
```

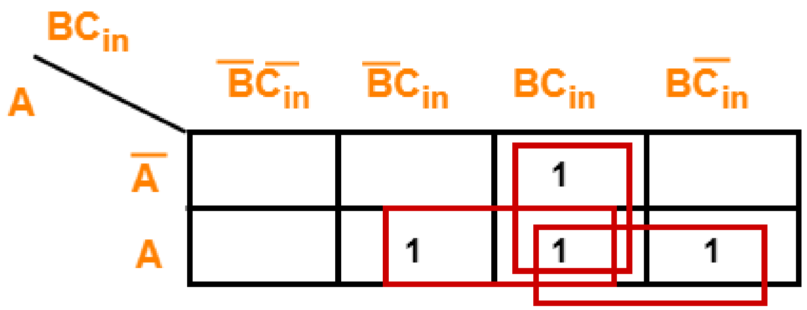
For S:



$A \backslash BC_{in}$	$\overline{B}\,\overline{C_{in}}$	$\overline{B}C_{in}$	BC_{in}	$B\overline{C_{in}}$
$\overline{A}$		1		1
A	1		1	

$$\boxed{S = A \oplus B \oplus C_{in}}$$

For C_{in}:



$A \backslash BC_{in}$	$\overline{B}\,\overline{C_{in}}$	$\overline{B}C_{in}$	BC_{in}	$B\overline{C_{in}}$
$\overline{A}$			1	
A		1	1	1

$$\boxed{C_{out} = AB + BC_{in} + C_{in}A}$$

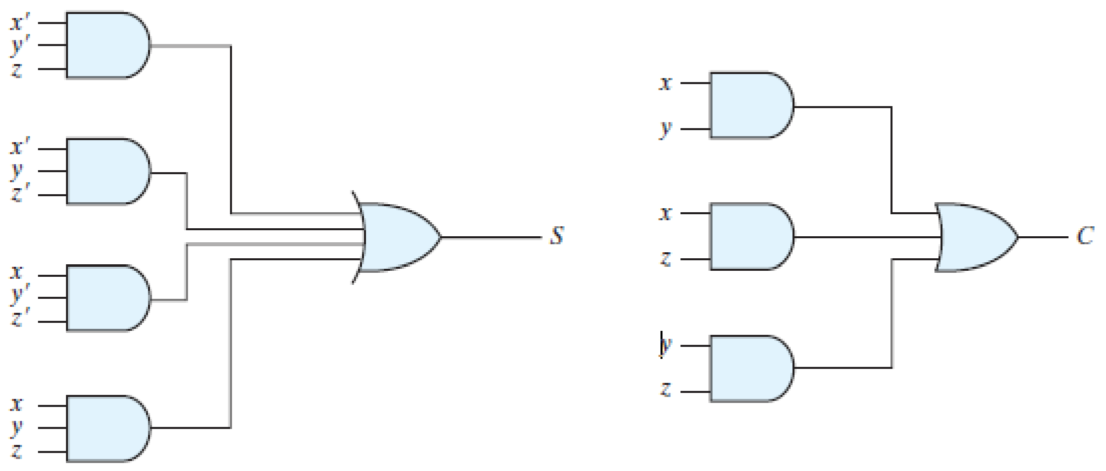
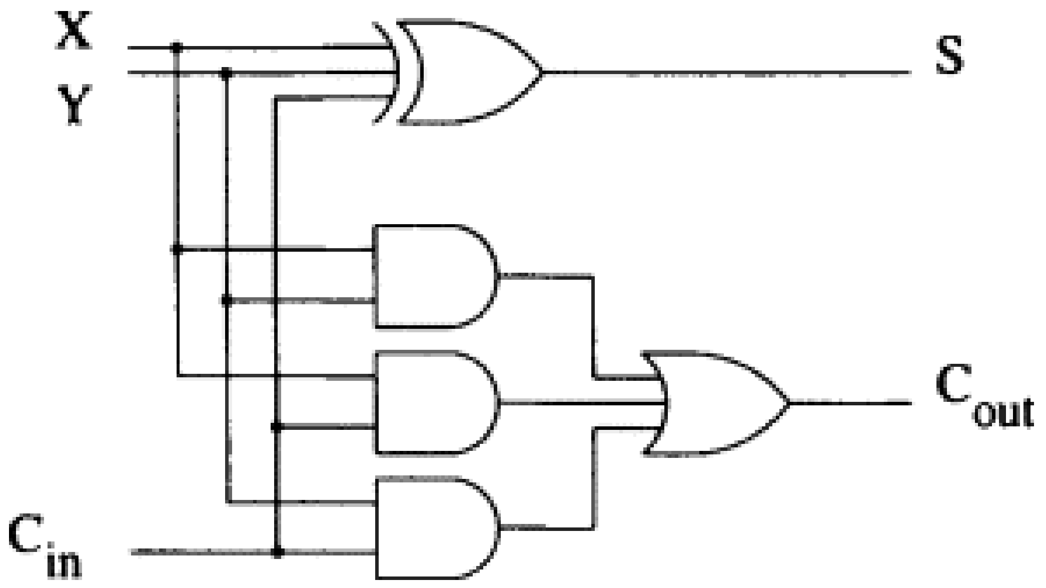


FIGURE 4.7
Implementation of full adder in sum-of-products form



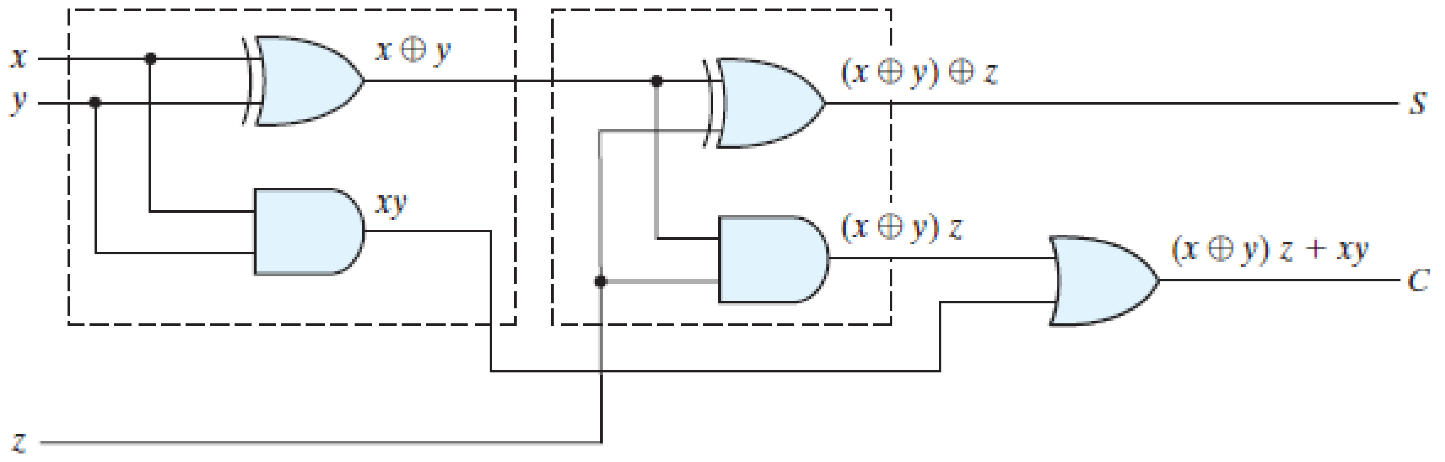
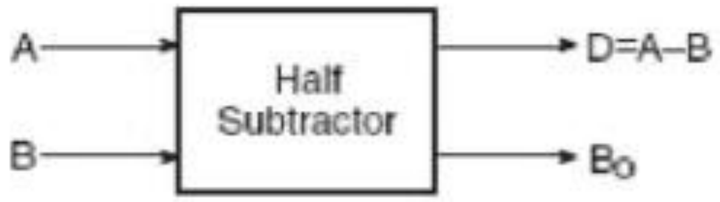


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

$$D = \overline{A}.B + A.\overline{B}$$

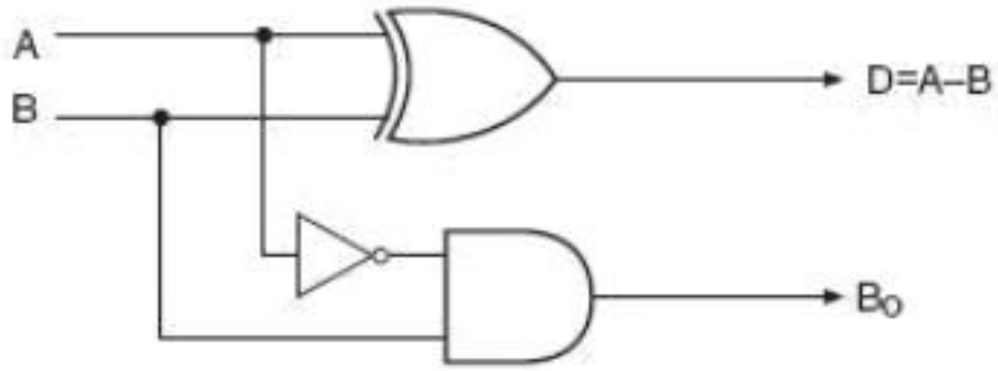
$$B_o = \overline{A}.B$$



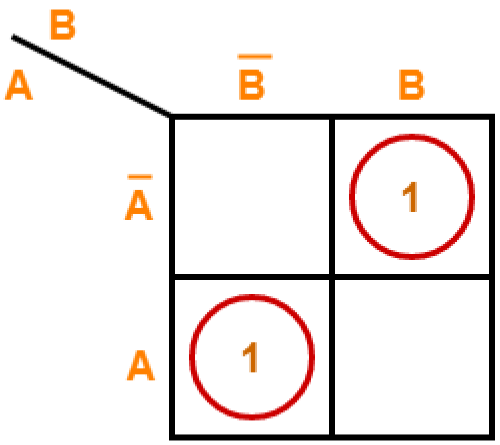
A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor

```
assign diff=a^b;
assign borrow=(~a)&b;
```

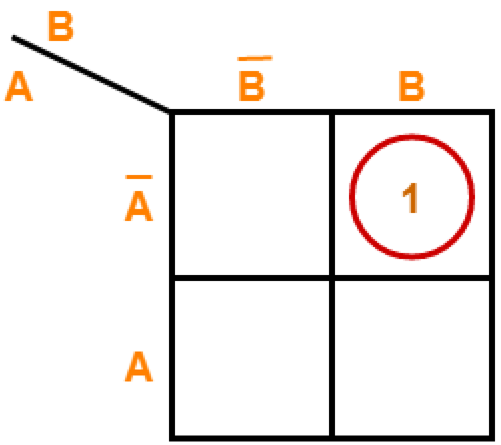


For D:



$$\boxed{D = A \oplus B}$$

For b:



$$\boxed{b = \overline{A}.B}$$

K Maps

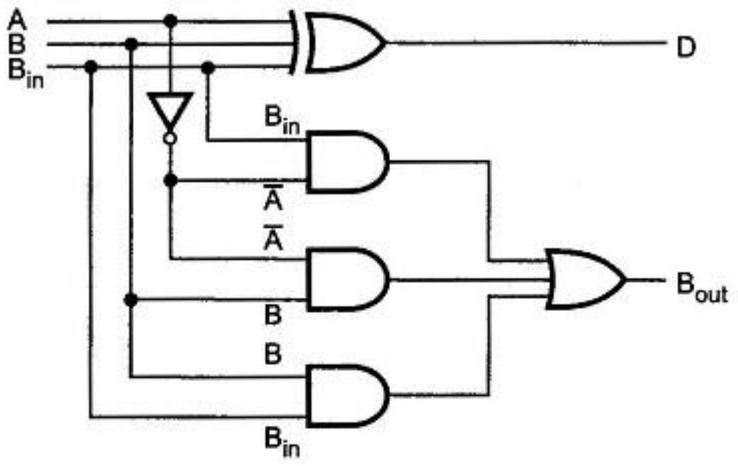
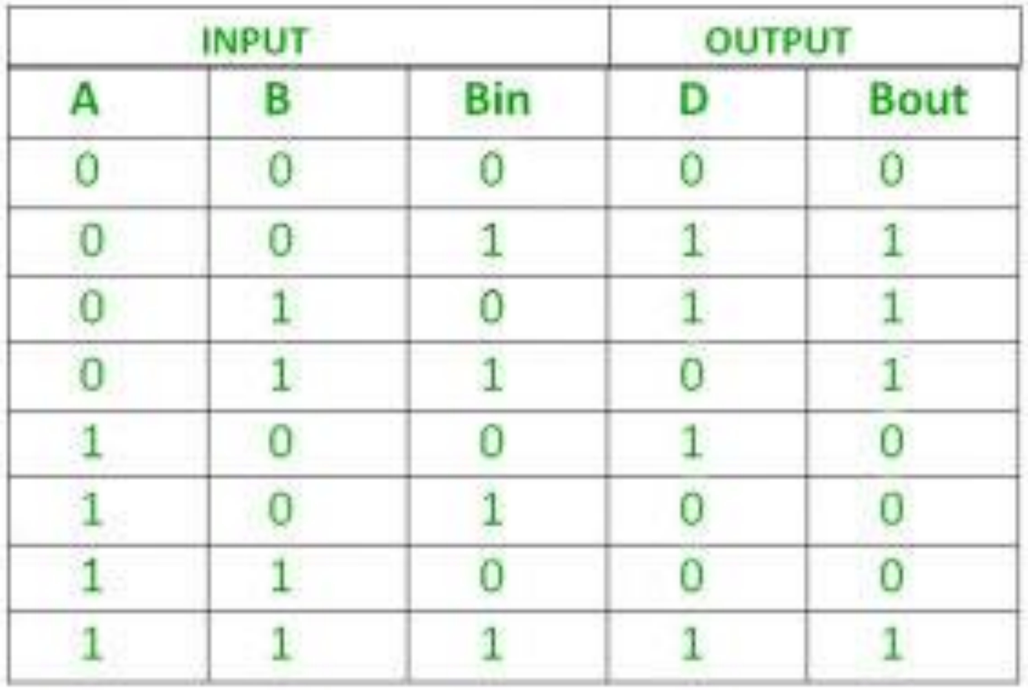


Fig. 3.23 Implementation of full-subtractor

```
assign diff=a^b^c;
assign borrow=(~a)&b|(~a)&c|b&c;
```



INPUT			OUTPUT	
A	B	Bin	D	Bout
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

For D:

$A \backslash BB_{in}$	$\bar{B}\bar{B}_{in}$	$\bar{B}B_{in}$	BB_{in}	$B\bar{B}_{in}$
$\bar{A}$		1		1
A	1		1	

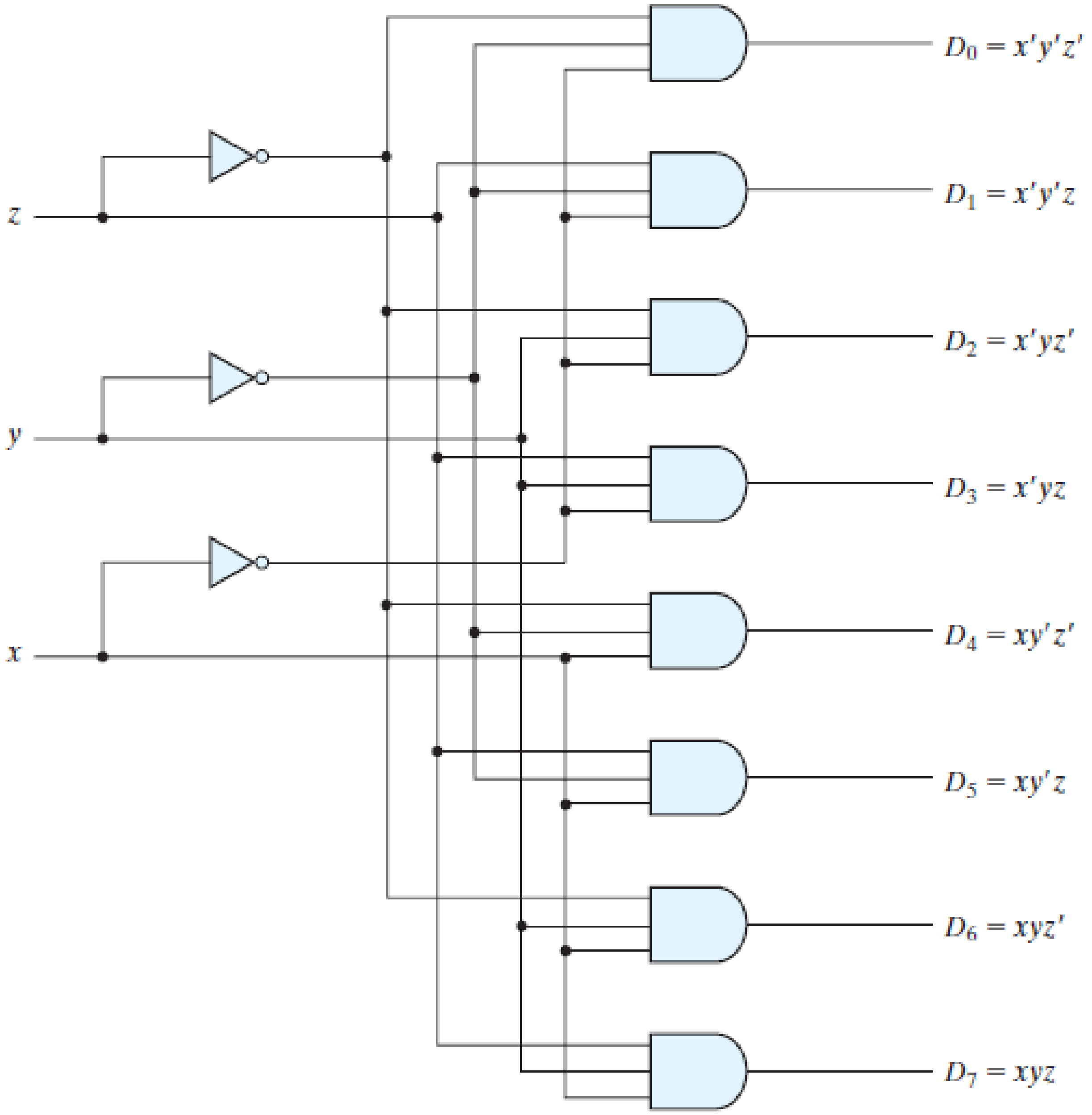
$$\boxed{D = A \oplus B \oplus B_{in}}$$

For B_{in}:

$A \backslash BB_{in}$	$\bar{B}\bar{B}_{in}$	$\bar{B}B_{in}$	BB_{in}	$B\bar{B}_{in}$
$\bar{A}$		1	1	1
A			1	

$$\boxed{B_{out} = \bar{A}B + (\bar{A} + B)B_{in}}$$

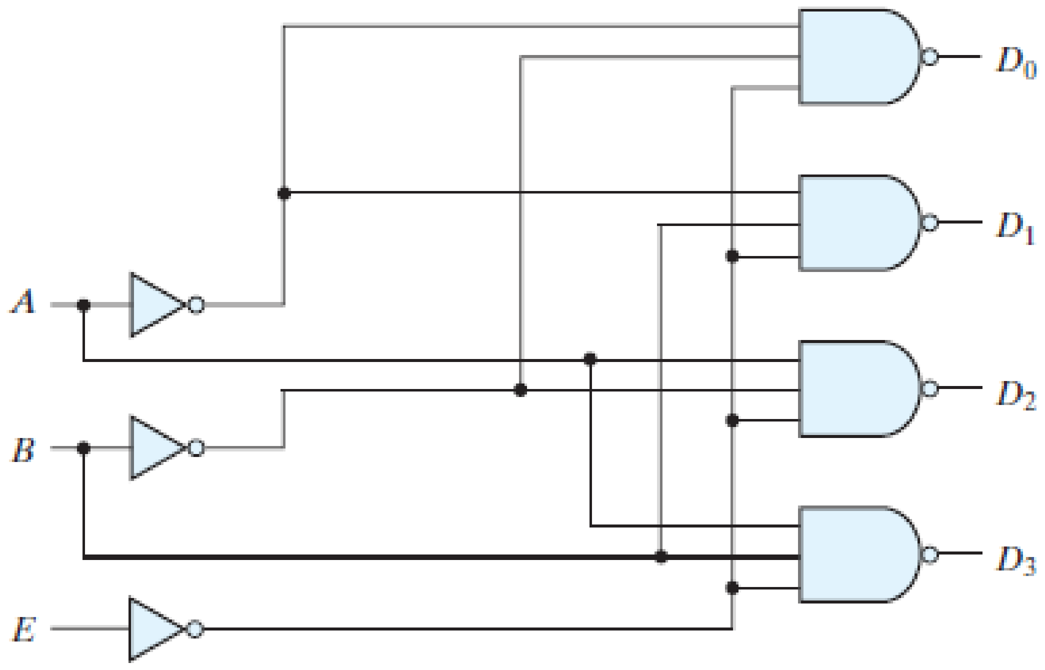
- *decoder* is a combinational circuit that converts binary information from *n* input lines to a maximum of 2^n unique output lines.
- The decoders presented here are called *n* -to-*m* -line decoders, where $m \leq 2^n$. Their
- BCD-to-seven-segment decoder.



IGURE 4.18

Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

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



(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	1

(b) Truth table

A decoder with enable input can function as a *demultiplexer*

- a circuit that receives information from a single line and directs it to one of 2^n possible output lines.

- function as a one-to-four-line demultiplexer when *E* is taken as a data input line and *A* and *B* are taken as the selection inputs.

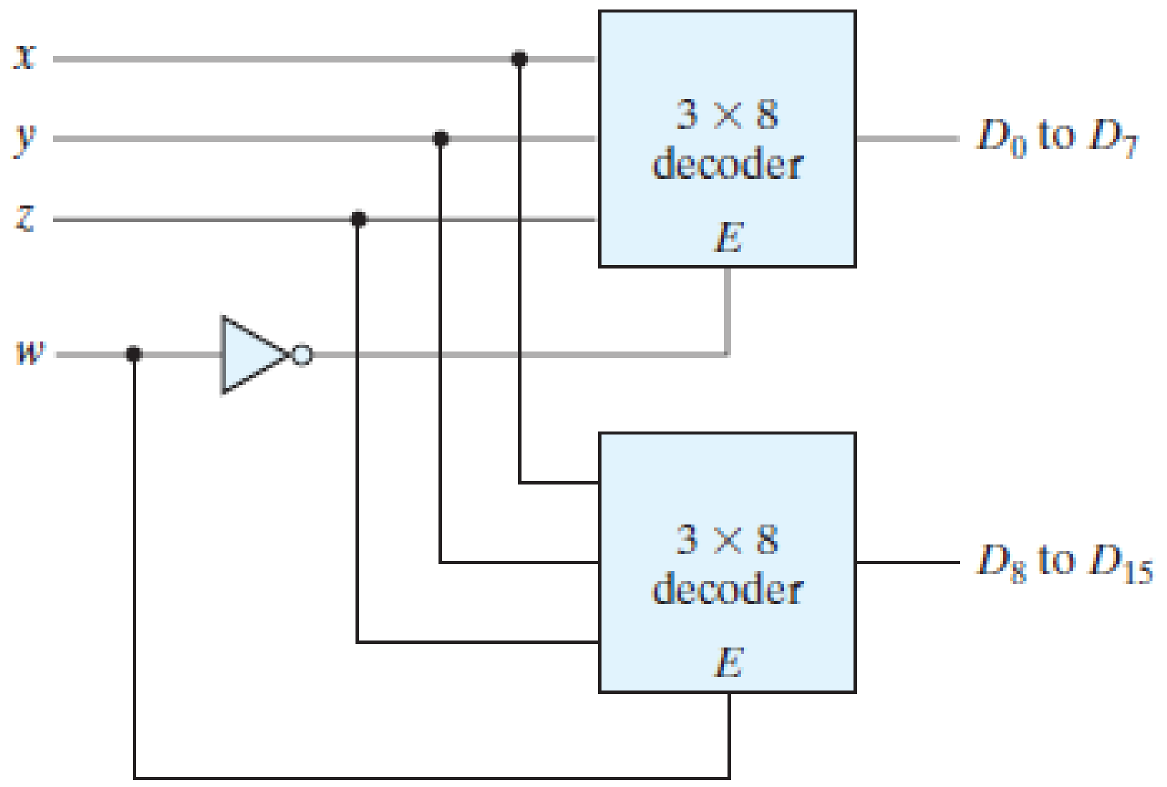


FIGURE 4.20
4×16 decoder constructed with two 3×8 decoders

Combinational Logic Implementation

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

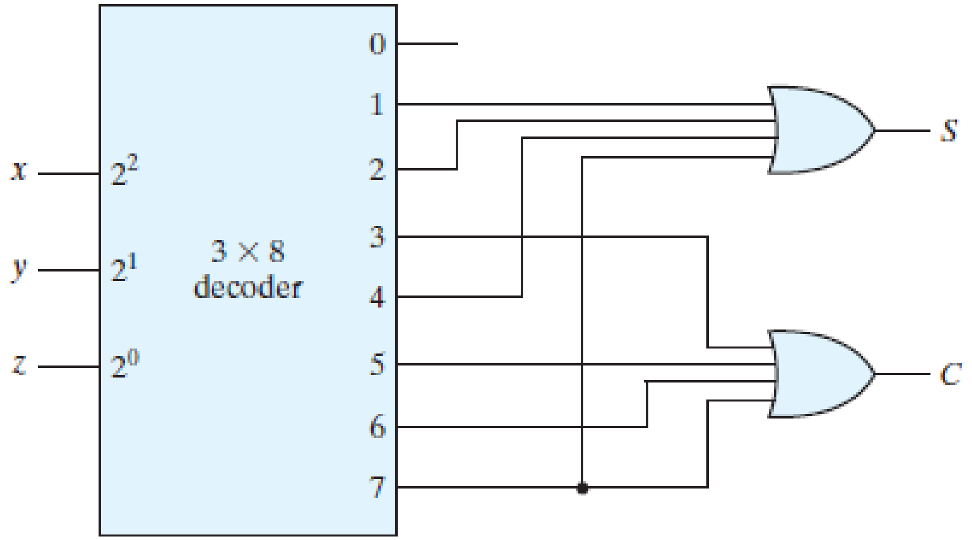
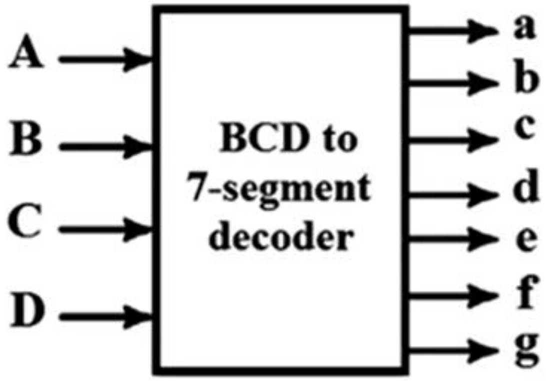


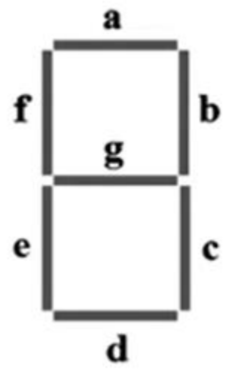
FIGURE 4.21
Implementation of a full adder with a decoder

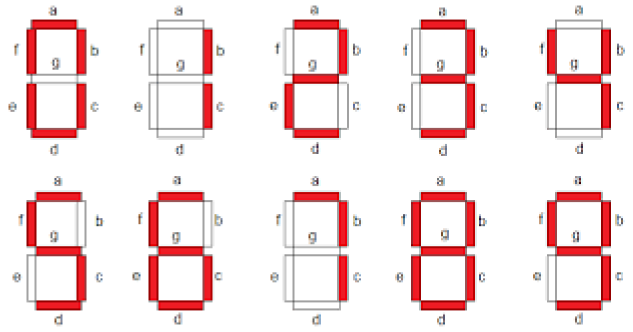
Application

- BCD to 7 segment decoder
- Address Decoder in Computer memory
- Instruction decoder in Control Unit

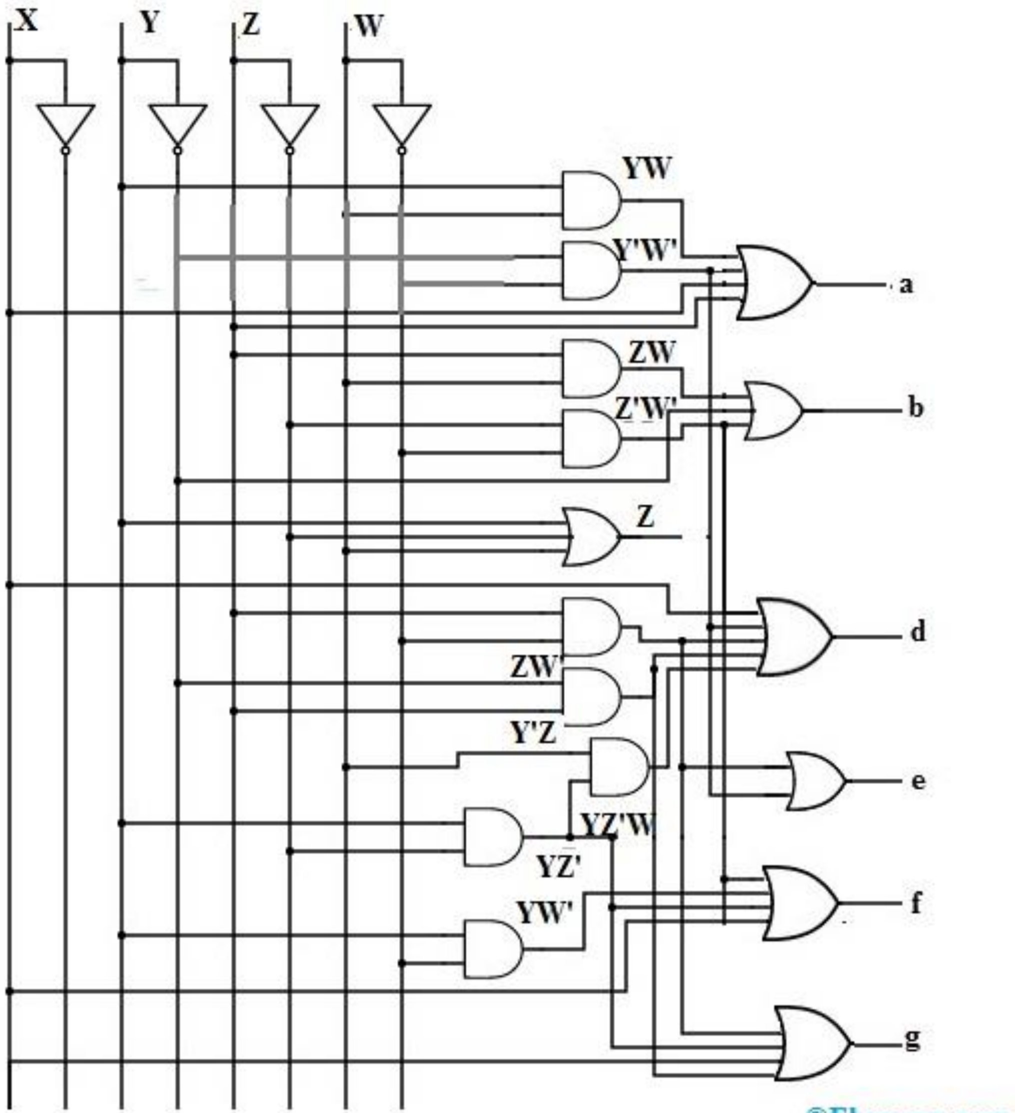
BCD to 7-segment decoder







A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1


X
Y
Z
W
YW
Y'W'
a
ZW
Z'W'
b
Z
d
ZW'
Y'Z
e
YZ'W
YZ'
f
YW'
g

Encoders

Table 4.7
Truth Table of an Octal-to-Binary Encoder

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

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

- if *D*3 and *D*6 are 1- What happens?
- What happens when all ips are zeroes?

Priority Encoder

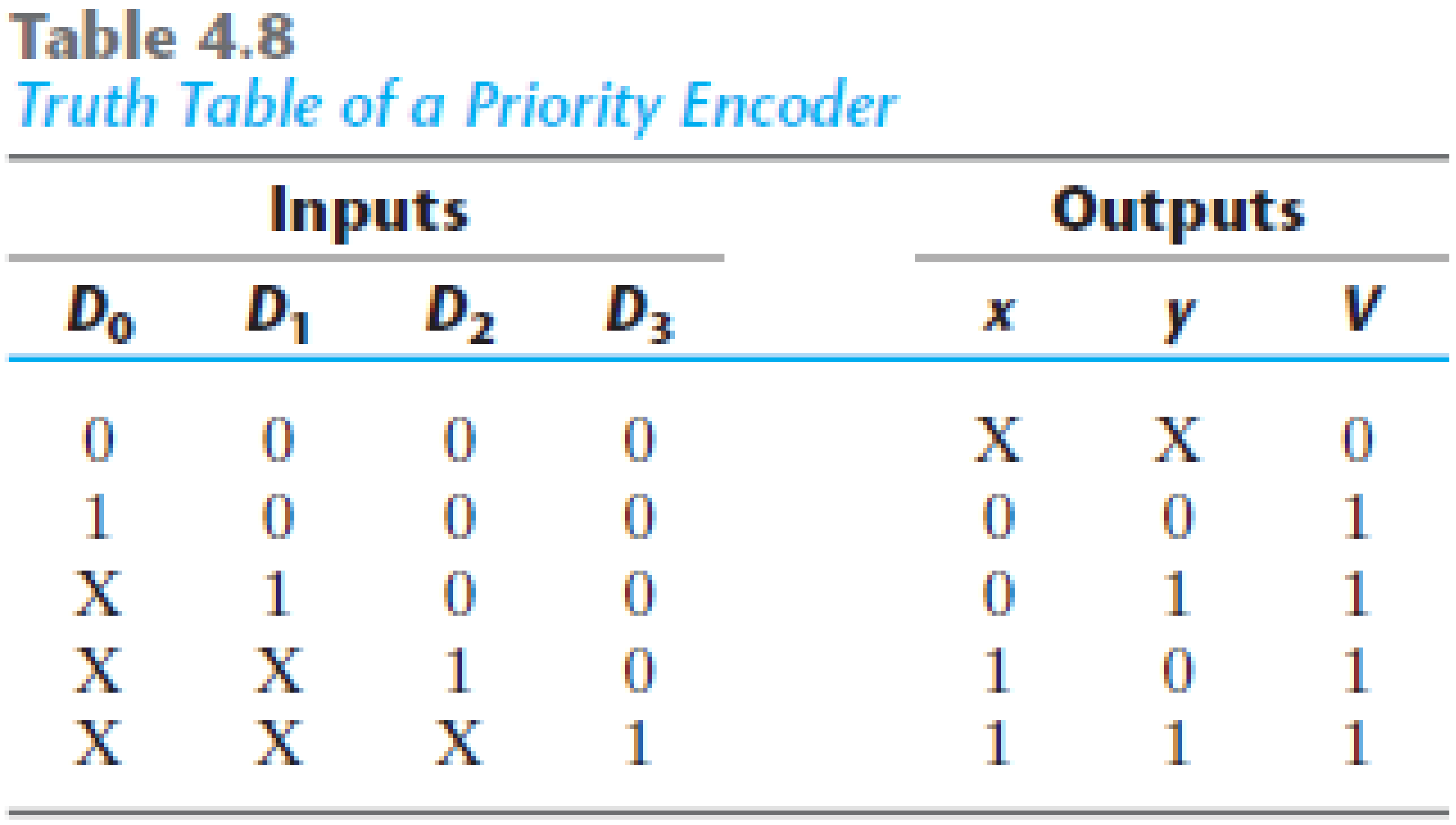
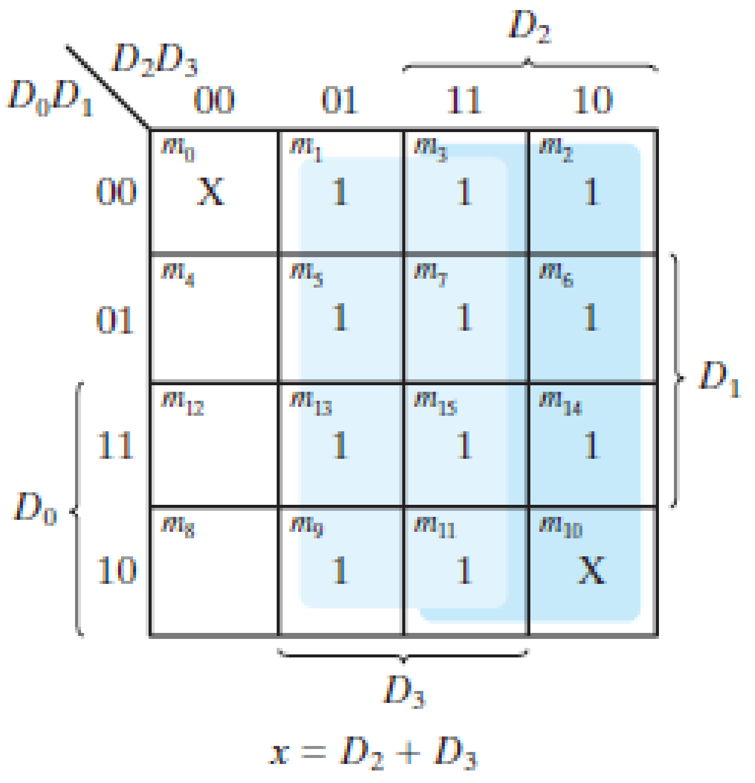


Table 4.8
Truth Table of a Priority Encoder

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



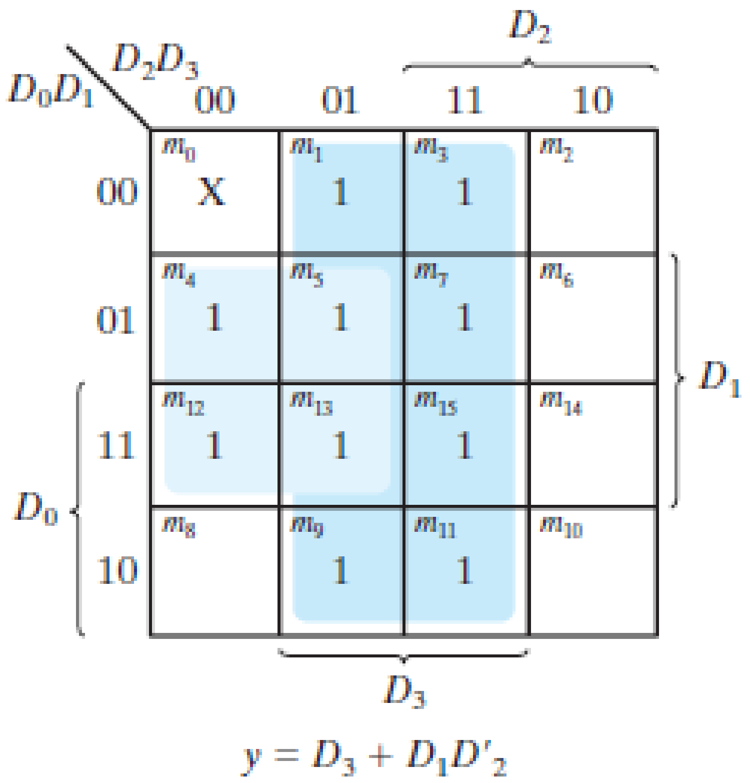


FIGURE 4.22
Maps for a priority encoder

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

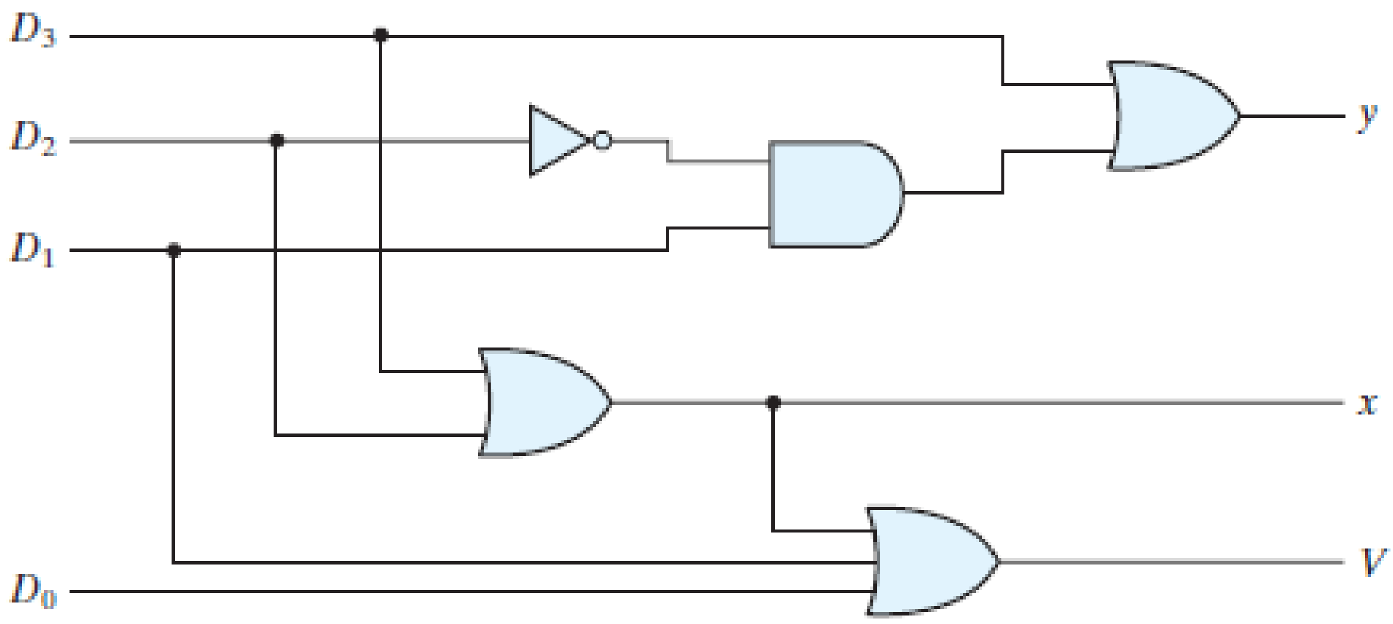
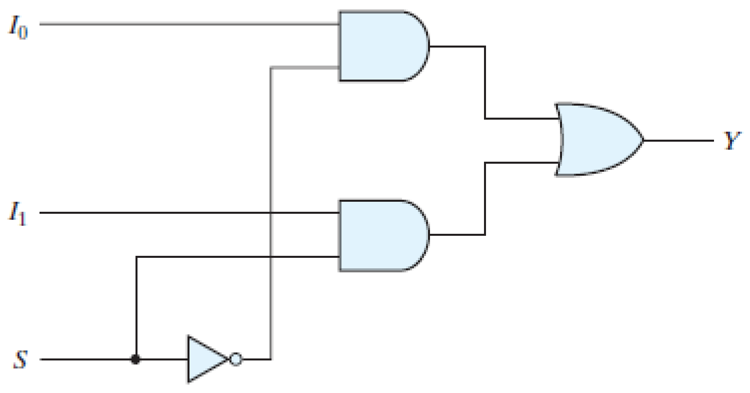


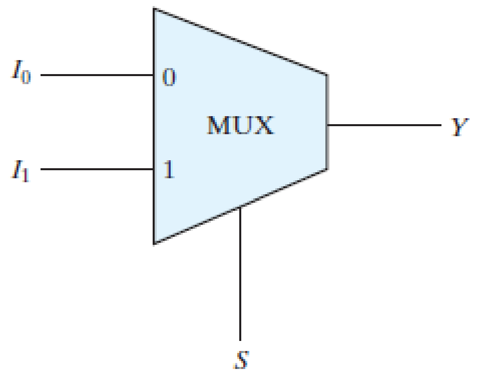
FIGURE 4.23
Four-input priority encoder

Multiplexers

- 2*n* input lines and *n* selection lines



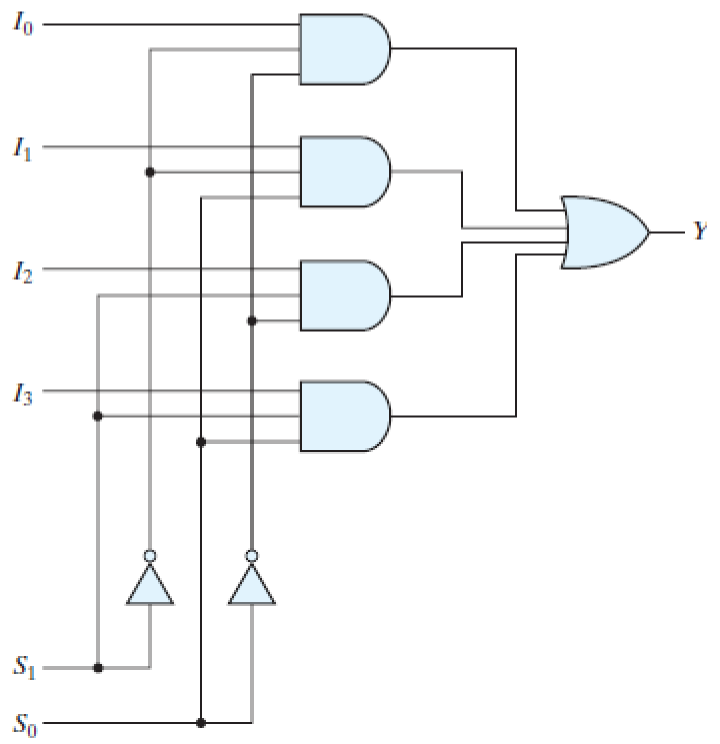
(a) Logic diagram



(b) Block diagram

FIGURE 4.24
Two-to-one-line multiplexer

A multiplexer is also called a *data selector* , since it selects one of many inputs and steers the binary information to the output line



(a) Logic diagram

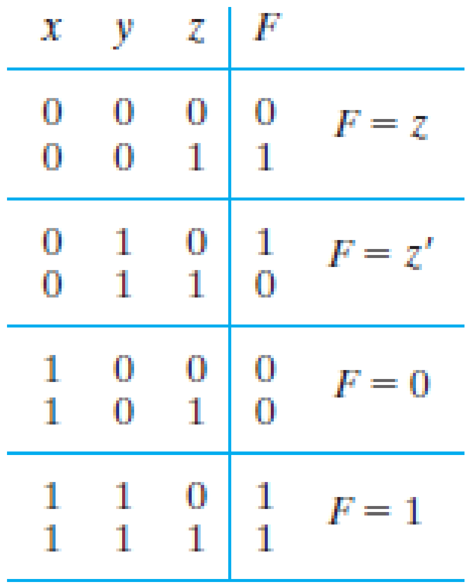
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

FIGURE 4.25
Four-to-one-line multiplexer

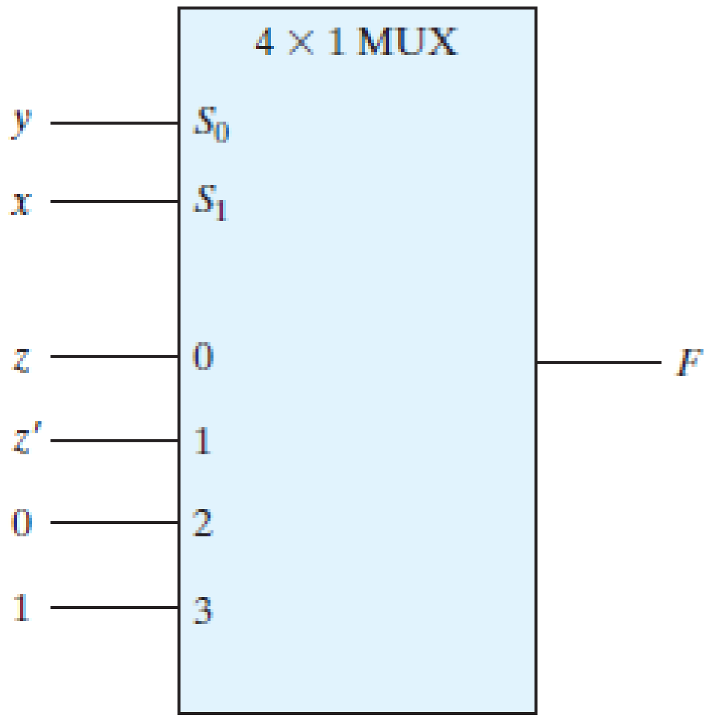
Boolean function Implementation using Mux

- $F(x, y, z) = (1, 2, 6, 7)$



x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

$F(A, B, C, D) = (1, 3, 4, 11, 12, 13, 14, 15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

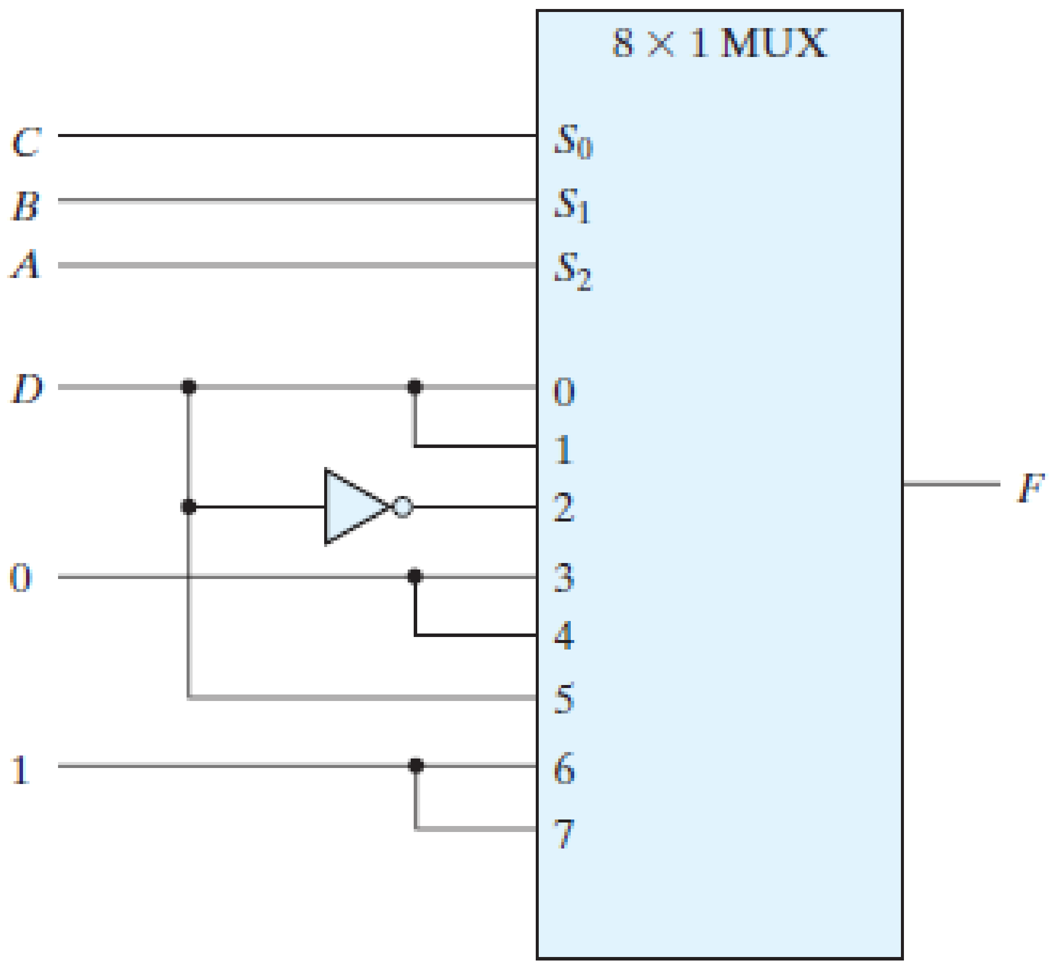
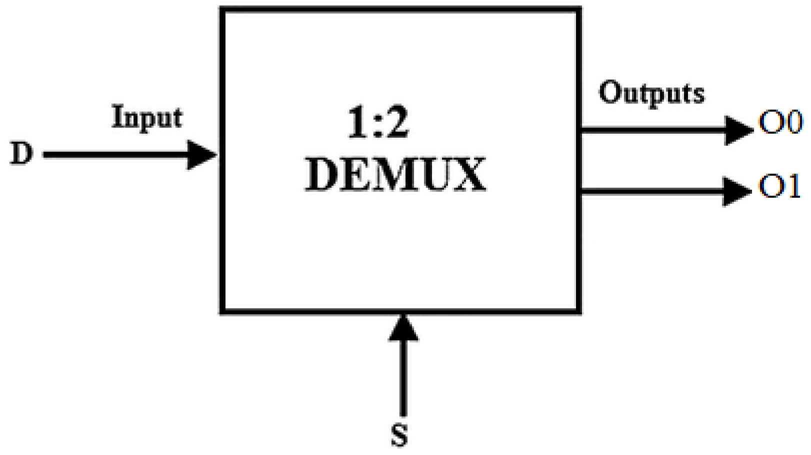
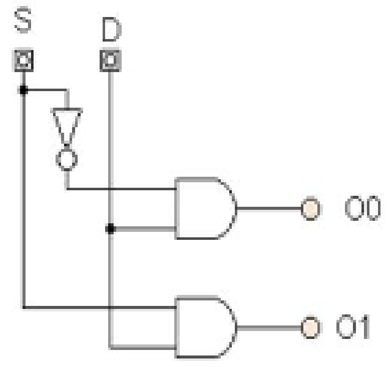


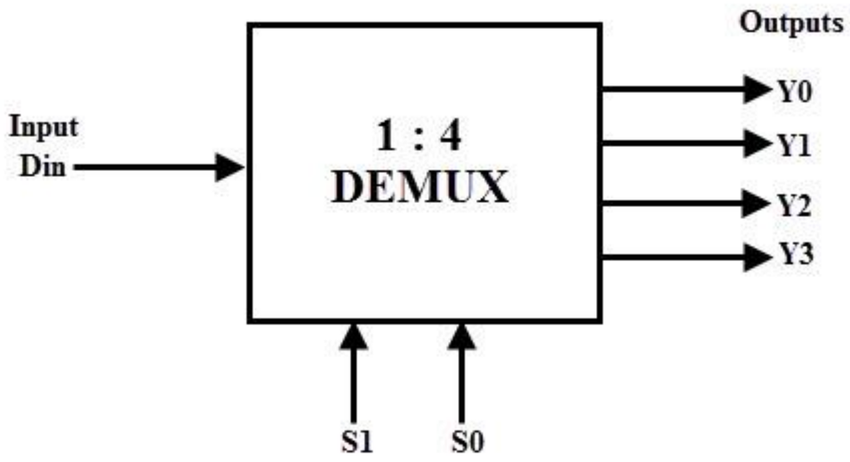
FIGURE 4.28
Implementing a four-input function with a multiplexer

Demultiplexer

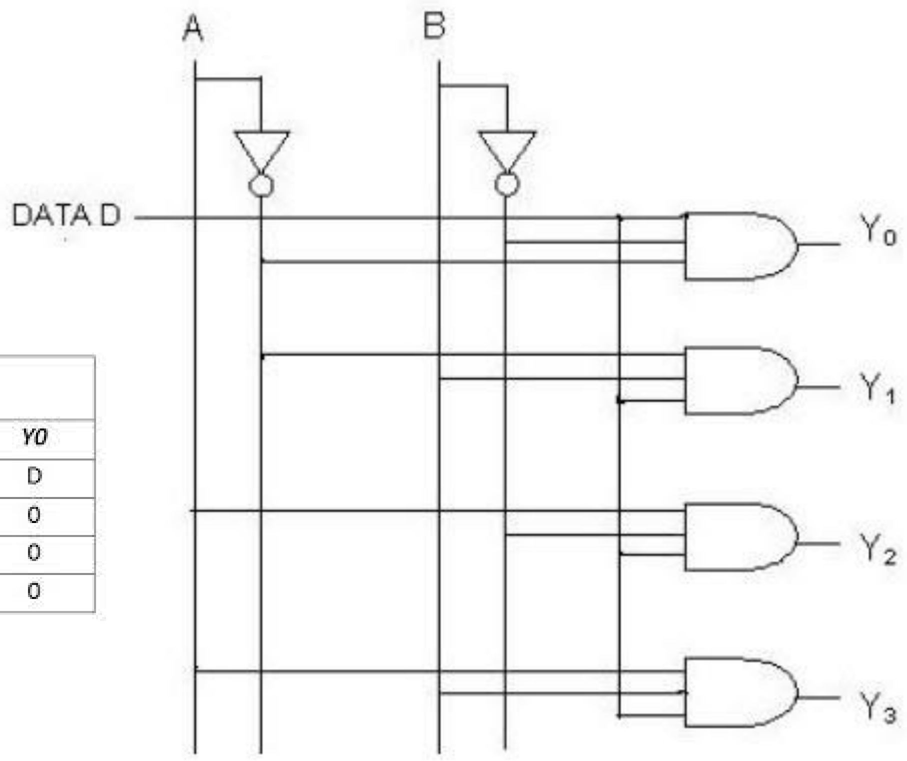


S D	0_1 O_0
0 0	0 0
0 1	0 1
1 0	0 0
1 1	1 0





Data Input	*Select input*		*Outputs*			
D	*S1*	*S0*	*Y3*	*Y2*	*Y1*	*Y0*
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0



DEMULTIPLEXERS

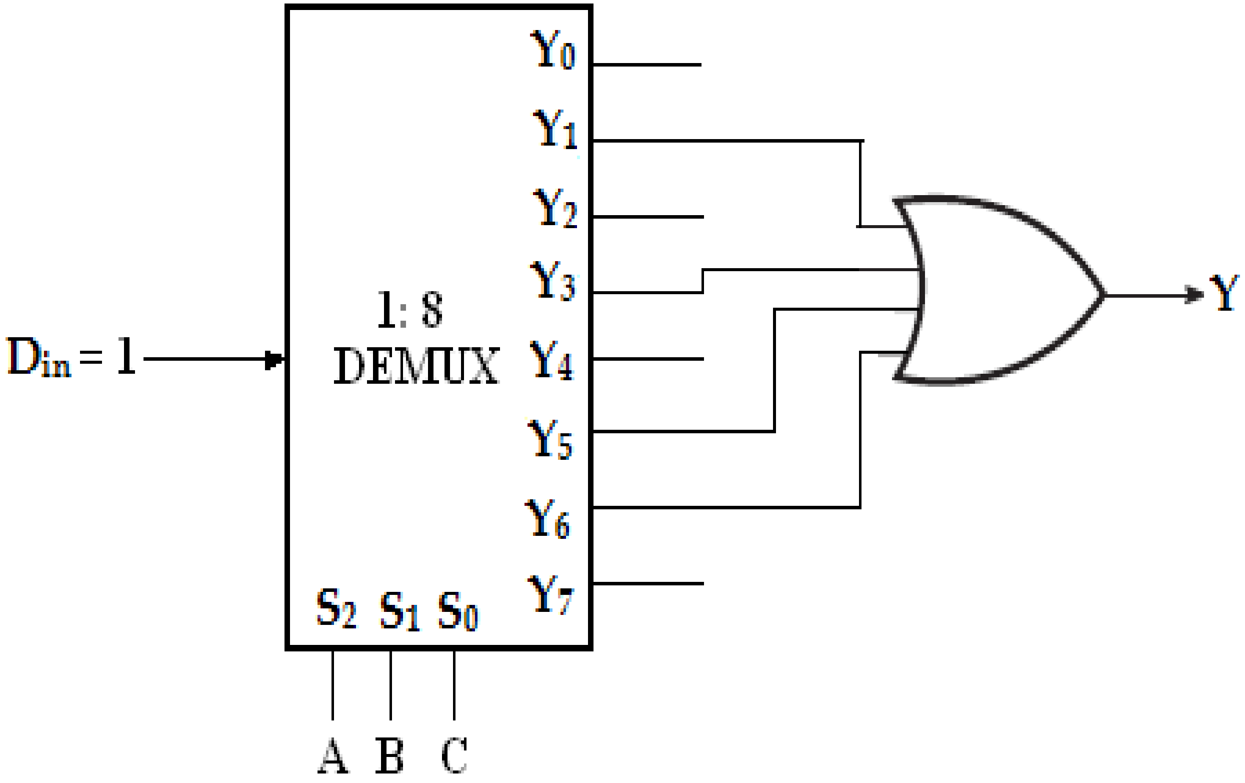
IMPLEMENTATION OF BOOLEAN EXPRESSION USING DEMUX

Example1: **Implement the following boolean function using 1:8 Demux. F (A, B, C) = ∑m (1, 3, 5, 6).**

Inputs			Outputs
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

DEMULTIPLEXERS

IMPLEMENTATION OF BOOLEAN EXPRESSION USING DEMUX



DEMULTIPLEXERS

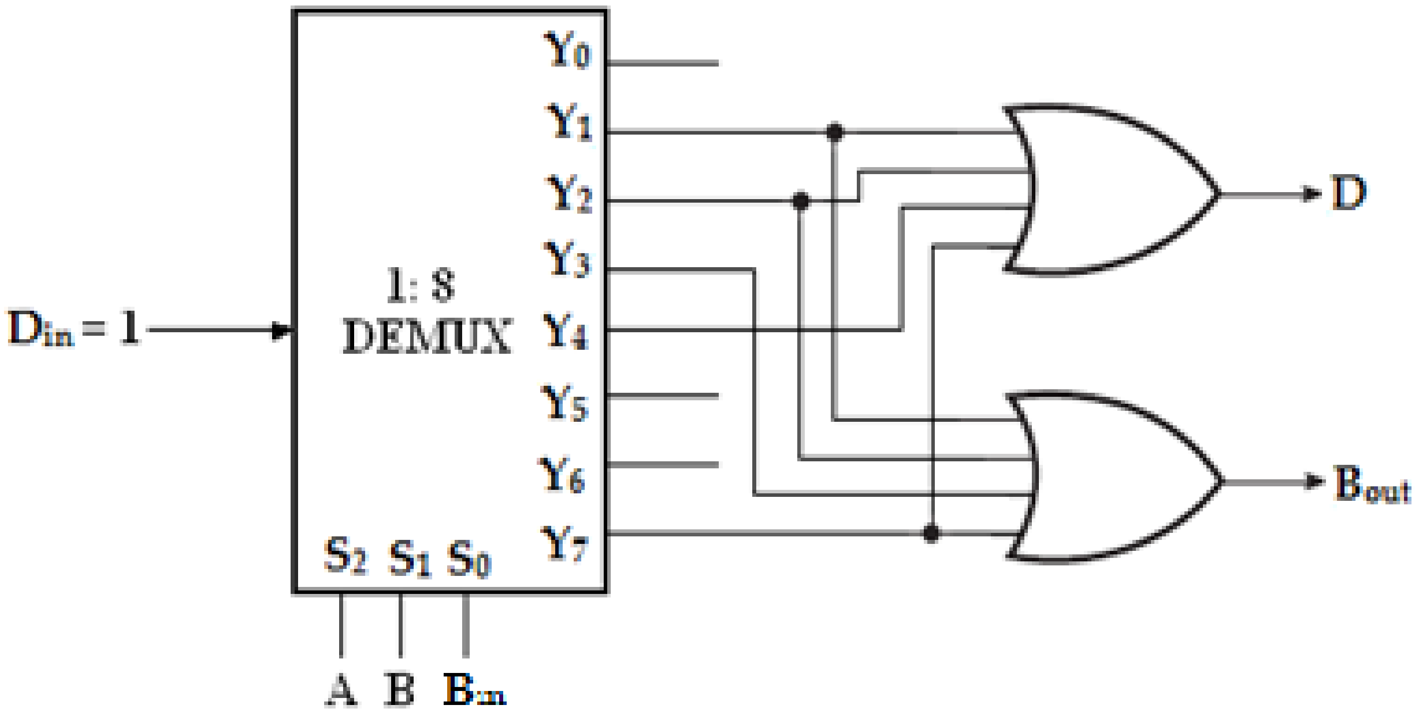
IMPLEMENTATION OF BOOLEAN EXPRESSION USING DEMUX

Example2: Implement full subtractor using demux

Inputs			Outputs	
A	B	B_{in}	Difference(D)	Borrow(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

DEMULTIPLEXERS

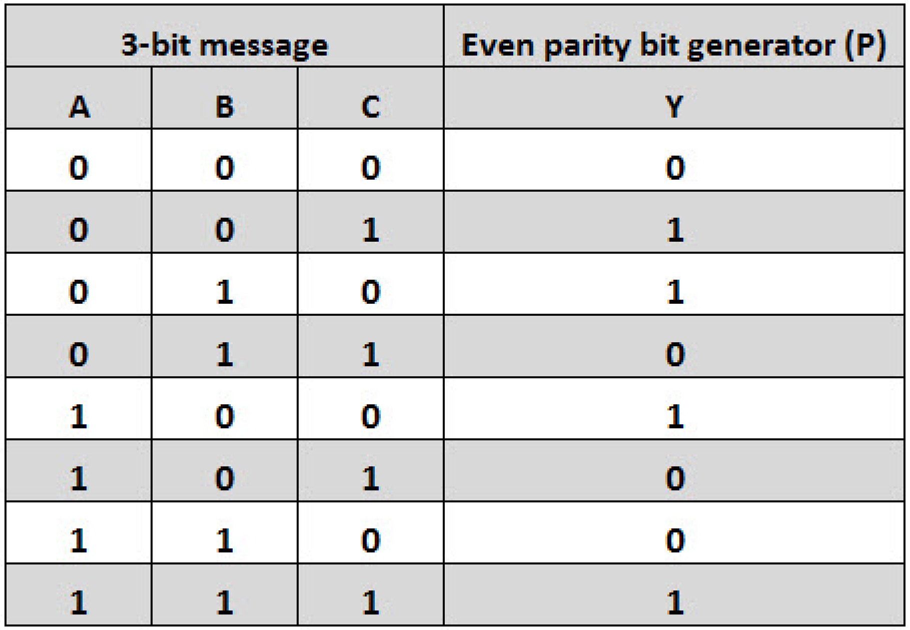
IMPLEMENTATION OF BOOLEAN EXPRESSION USING DEMUX



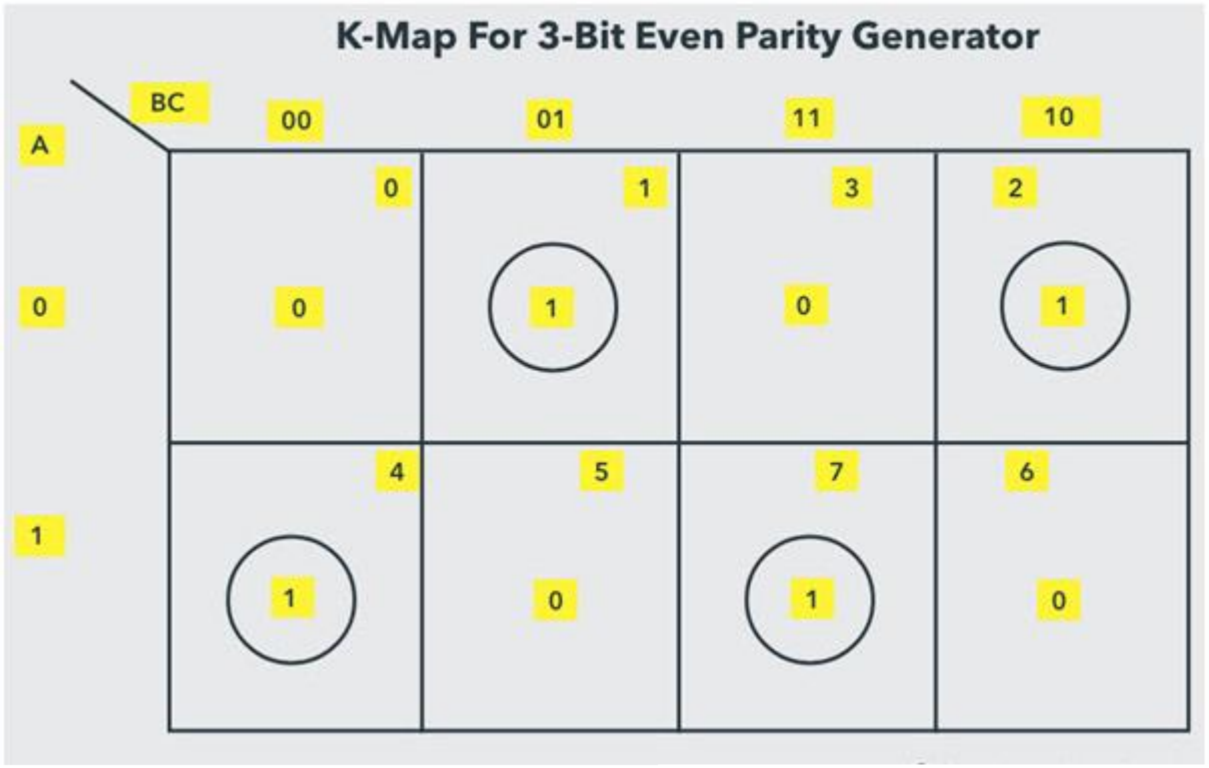
Full Subtractor using 1:8 demux

Parity Generator and Checker

- Even and odd parity
- Even parity generator



3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

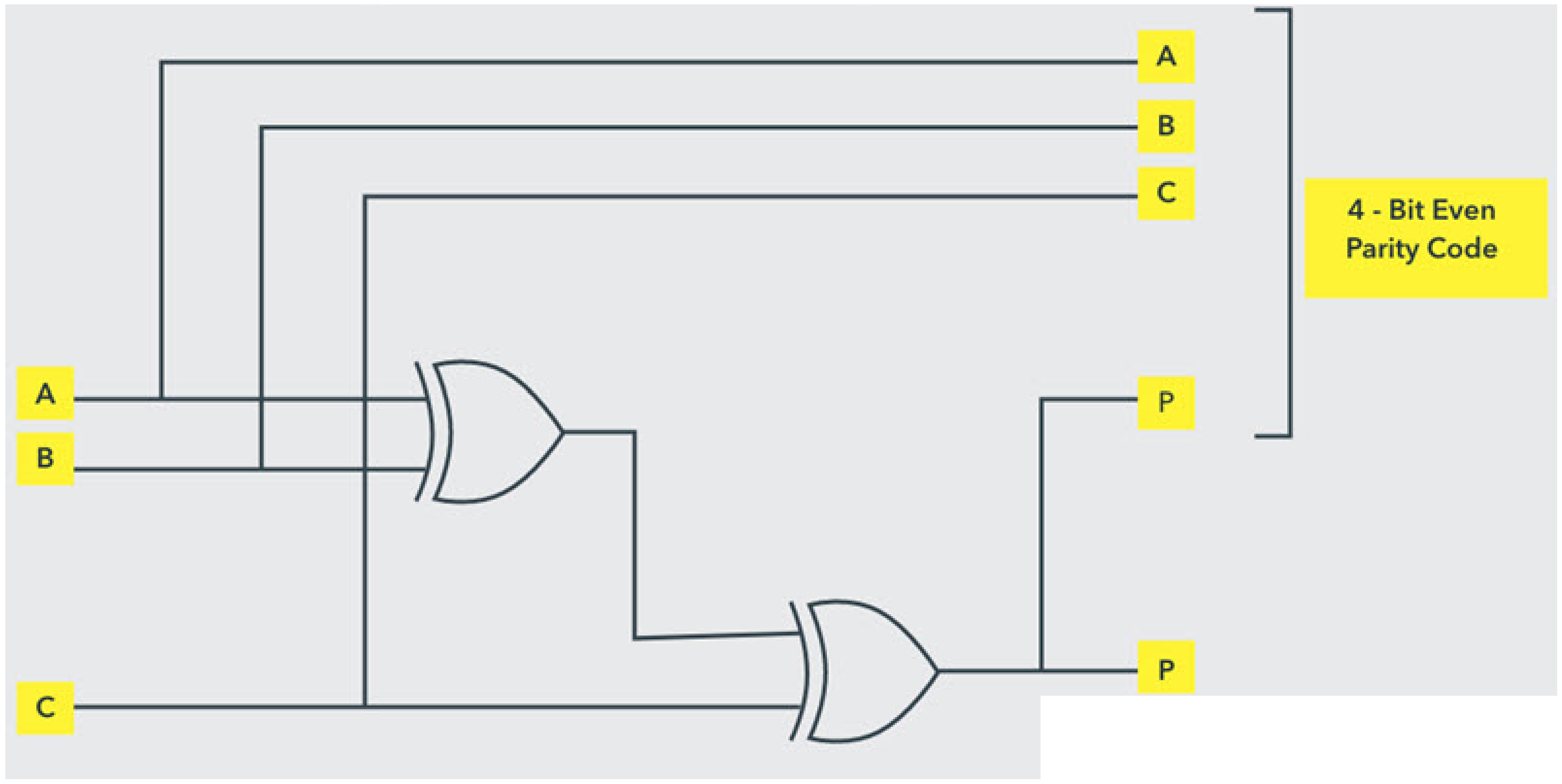


$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

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

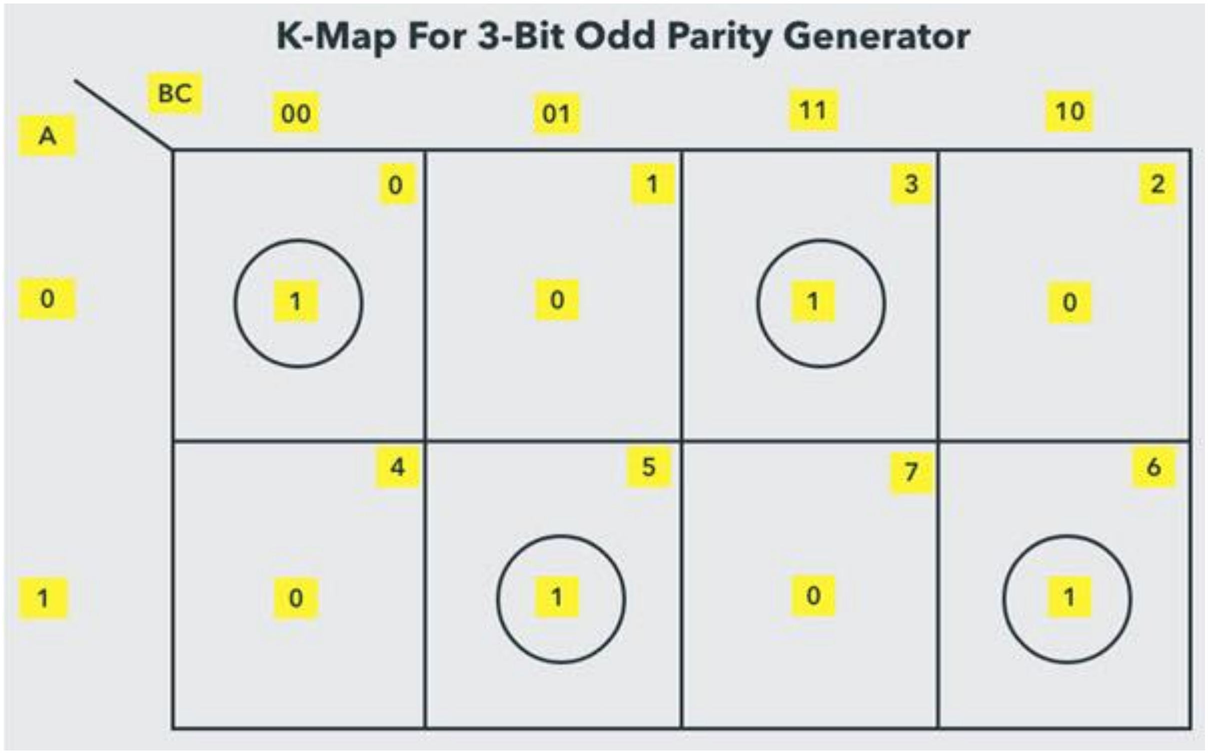
$$= \bar{A}\,(B \oplus C) + A\,(\overline{B \oplus C})$$

$$P = A \oplus B \oplus C$$


A
B
C
4 - Bit Even Parity Code
A
B
P
C
P

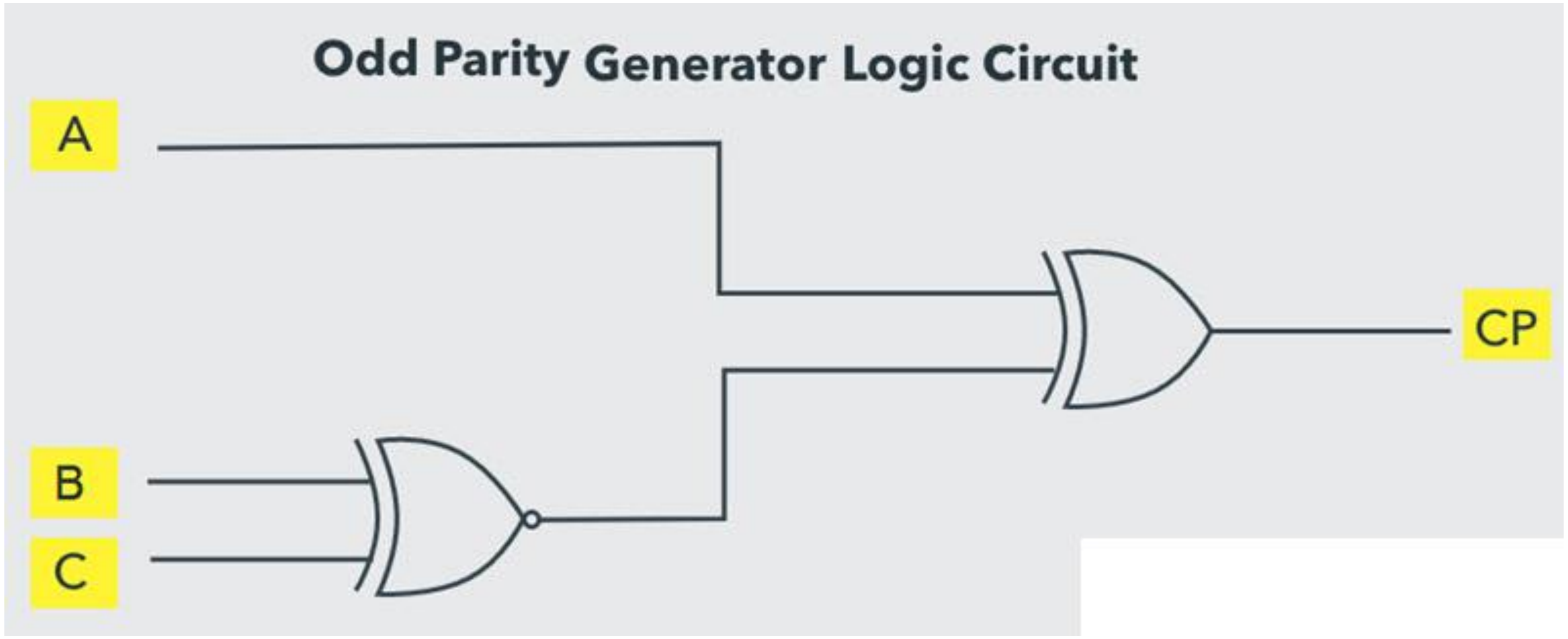
- Odd parity generator

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

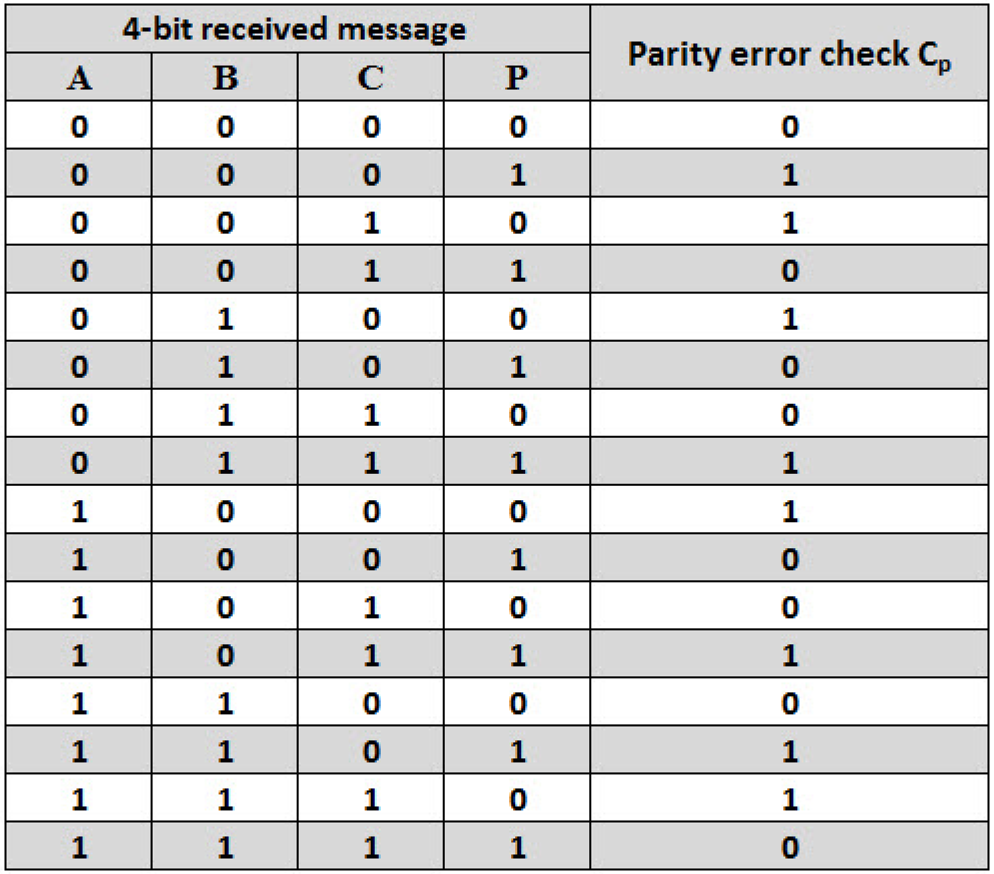

K-Map For 3-Bit Odd Parity Generator

A \ BC	00	01	11	10
0	$\overset{0}{1}$	$\overset{1}{0}$	$\overset{3}{1}$	$\overset{2}{0}$
1	$\overset{4}{0}$	$\overset{5}{1}$	$\overset{7}{0}$	$\overset{6}{1}$

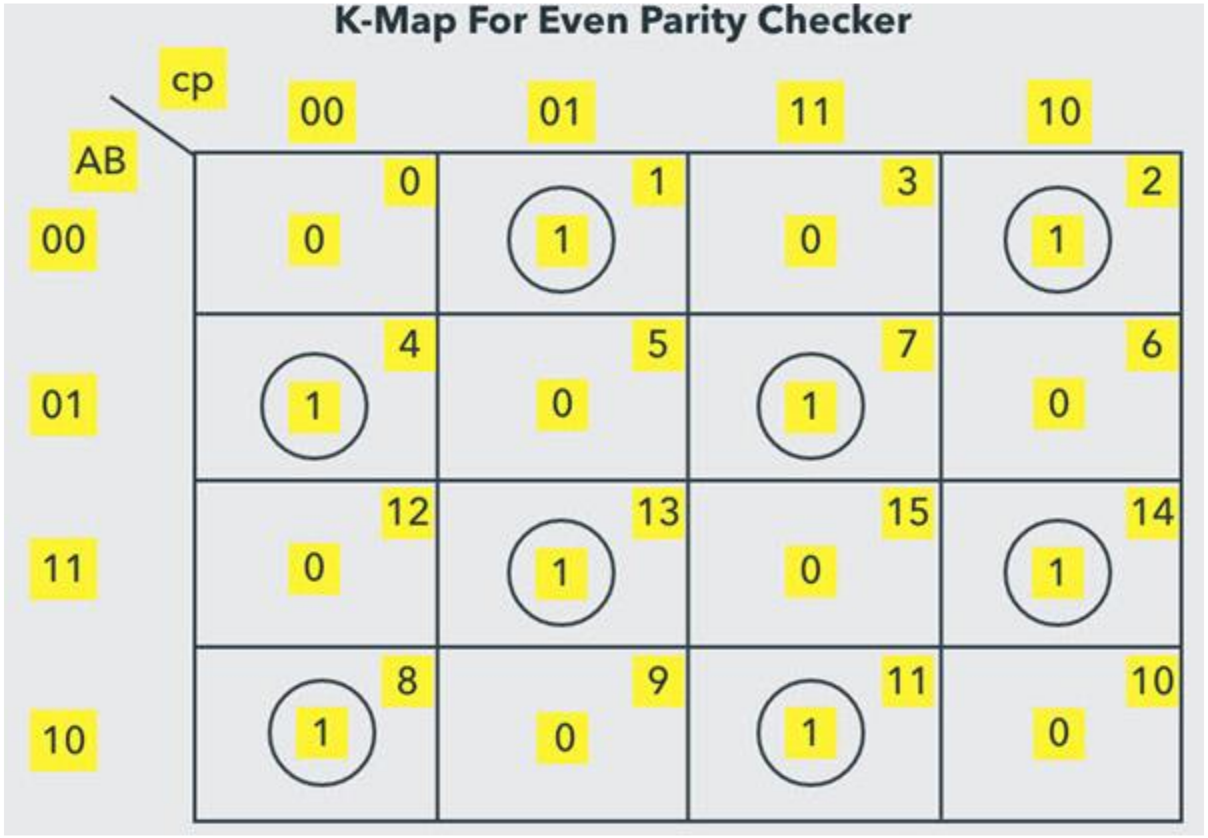
$$P = A \oplus (B \overline{\oplus} C)$$


Odd Parity Generator Logic Circuit
A
B
C
CP

Even parity checker



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



K-Map For Even Parity Checker

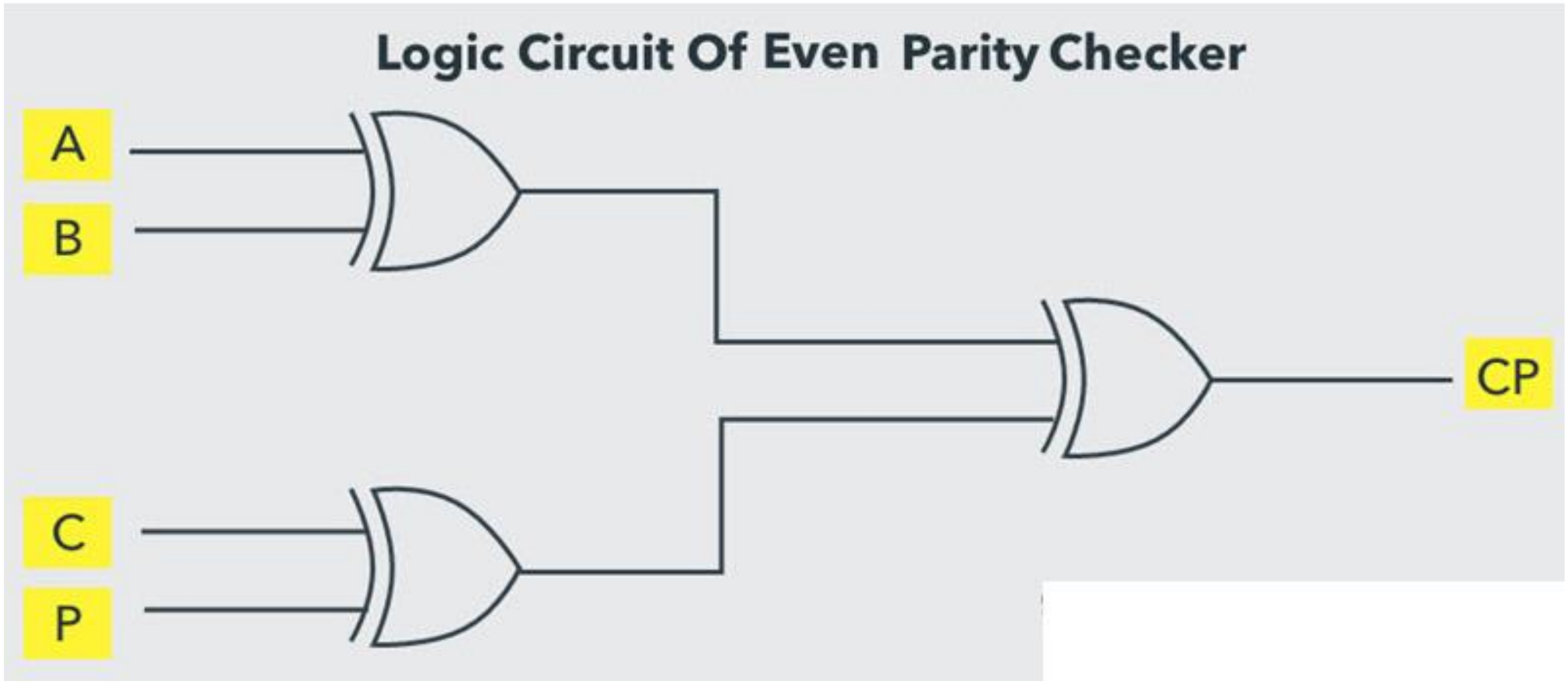
AB \ cp	00	01	11	10
00	0 (0)	1 (1)	3 (0)	2 (1)
01	4 (1)	5 (0)	7 (1)	6 (0)
11	12 (0)	13 (1)	15 (0)	14 (1)
10	8 (1)	9 (0)	11 (1)	10 (0)

$$PEC = \overline{A}\,\overline{B}\,(\overline{C}\,P + C\,\overline{P}) + \overline{A}\,B\,(\overline{C}\,\overline{P} + C\,P) + A\,B\,(\overline{C}\,P + C\,\overline{P}) + A\,\overline{B}\,(\overline{C}\,\overline{P} + C\,P)$$

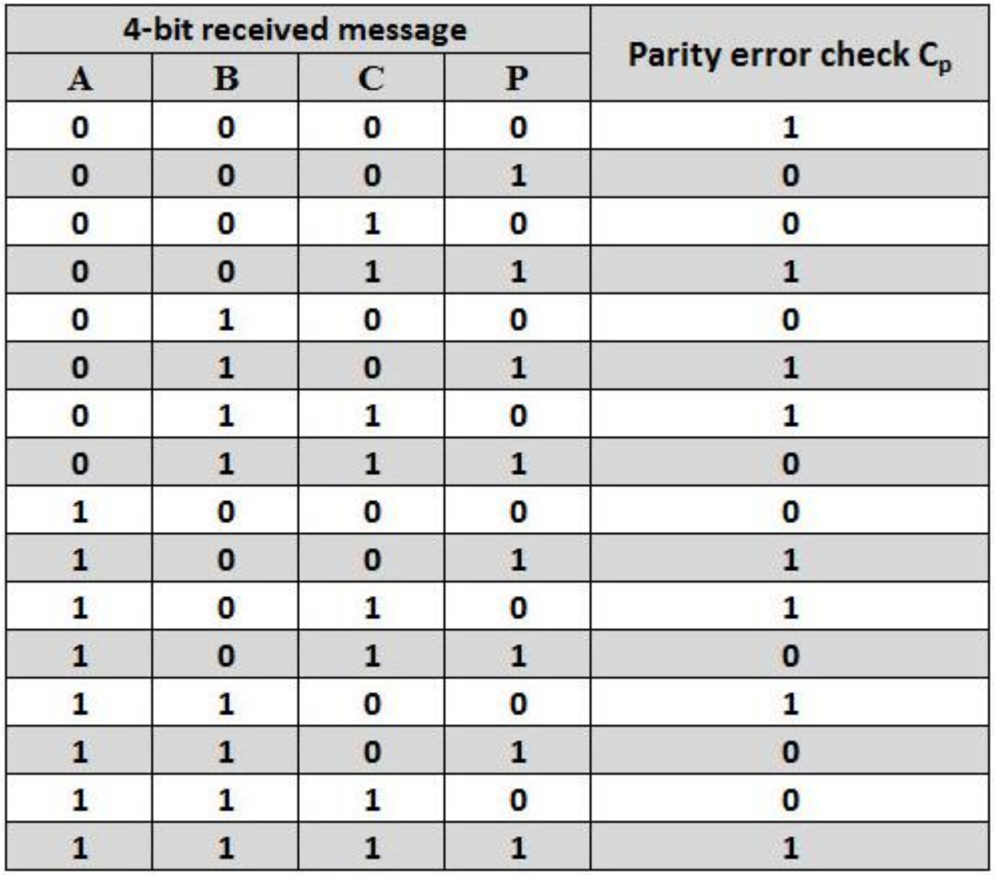
$$= \overline{A}\,\overline{B}\,(C \oplus P) + \overline{A}\,B\,(\overline{C \oplus P}) + A\,B\,(C \oplus P) + A\,\overline{B}\,(\overline{C \oplus P})$$

$$= (\overline{A}\,\overline{B} + A\,B)(C \oplus P) + (\overline{A}\,B + A\,\overline{B})(\overline{C \oplus P})$$

$$= (A \oplus B) \oplus (C \oplus P)$$

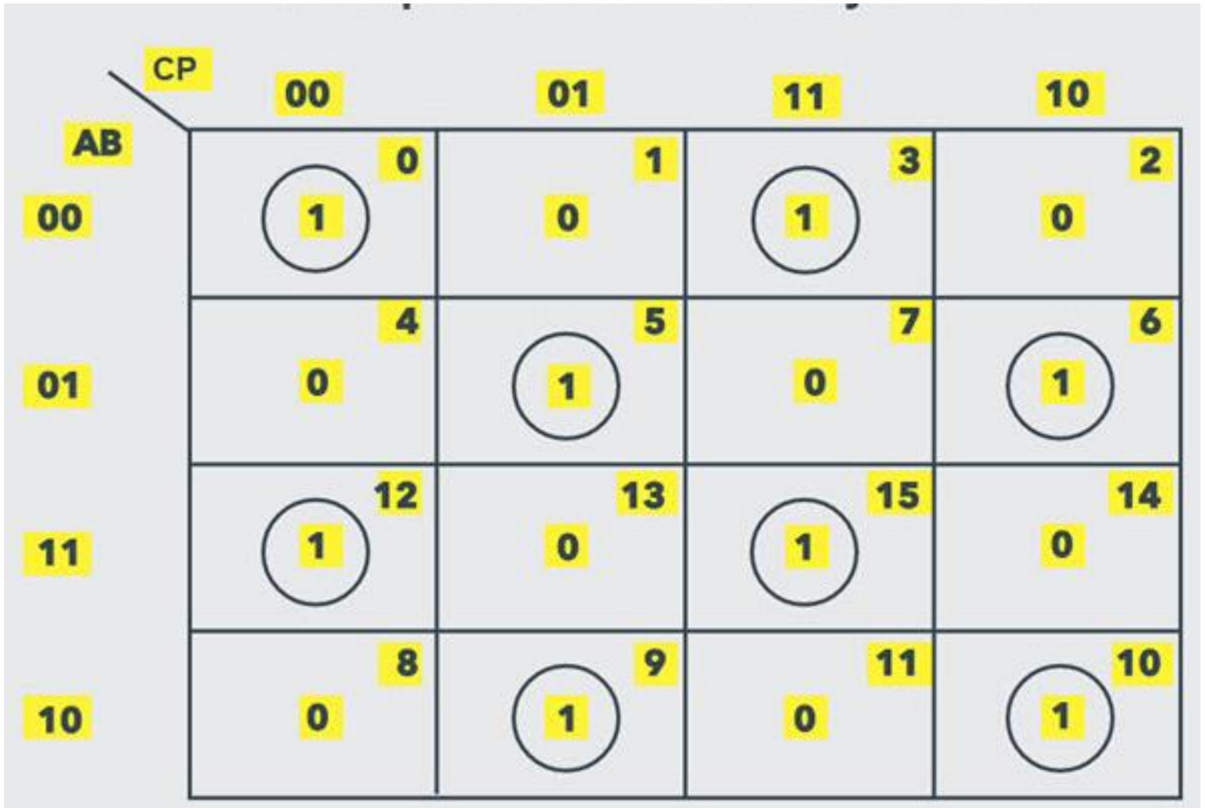


Odd parity checker



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Microsoft Excel 2010



AB \ CP	00	01	11	10
00	1^{0}	0^{1}	1^{3}	0^{2}
01	0^{4}	1^{5}	0^{7}	1^{6}
11	1^{12}	0^{13}	1^{15}	0^{14}
10	0^{8}	1^{9}	0^{11}	1^{10}

PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR P)

Logic Circuit Of Odd Parity Checker