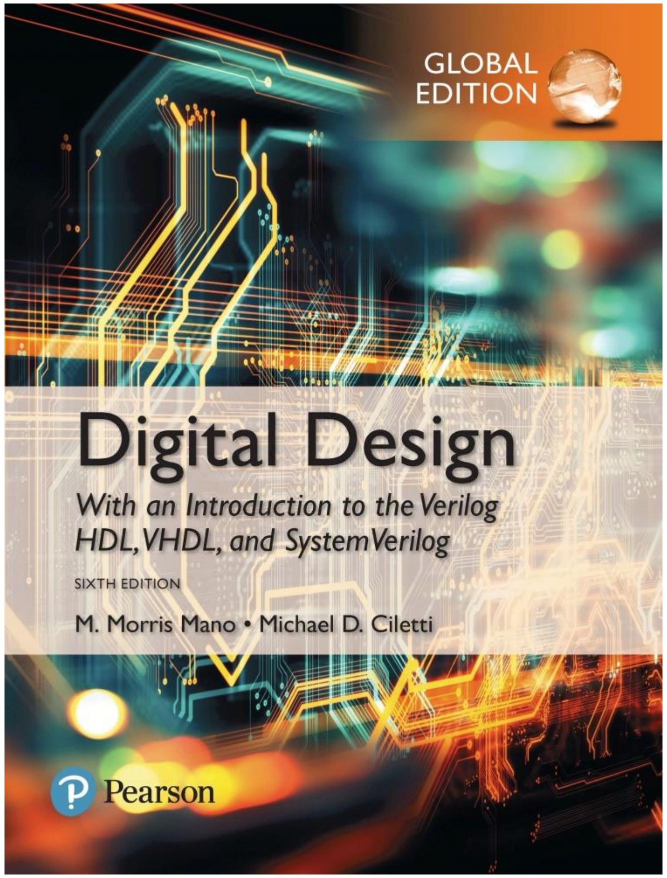
Digital Design

With an Introduction to the Verilog HDL, VHDL, and SystemVerilog

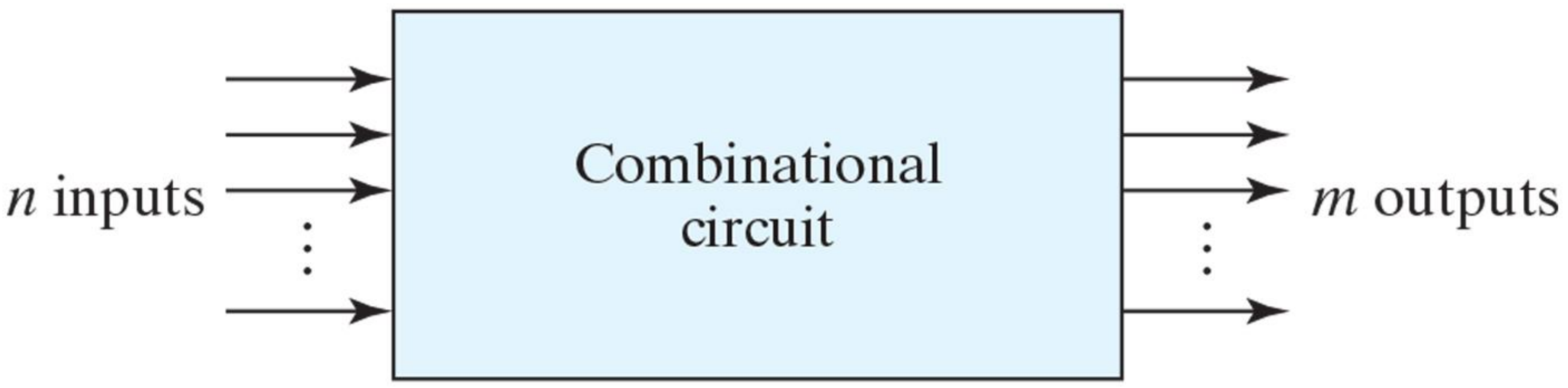
6th Edition, Global Edition


GLOBAL EDITION
Digital Design
With an Introduction to the Verilog HDL, VHDL, and SystemVerilog
SIXTH EDITION
M. Morris Mano • Michael D. Ciletti
Pearson

Chapter 04

Combinational Logic

Block diagram of combinational circuit



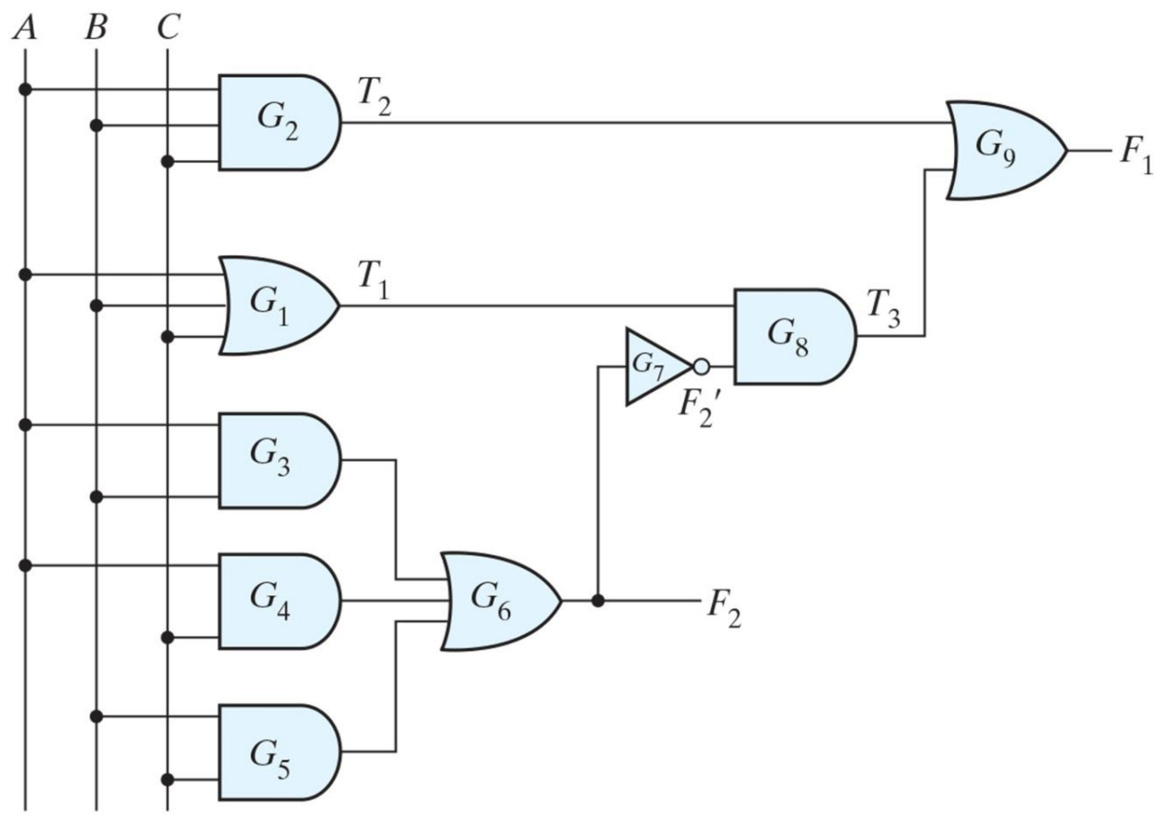
Outputs are determined by the present inputs only

Analysis Procedure

- To determine the function of logic
- Analysis procedure
  - Make sure the logic is combinational or sequential
  - Obtain the output Boolean functions or the truth table

- Boolean function
  - Label all gate outputs
  - Make output functions at each level
  - Substitute final outputs to input variables
- Truth table
  - Put the input variables to binary numbers
  - Determine the output value at each gate
  - Obtain truth table

Logic diagram for analysis example



F1= ?

Truth Table for the Logic Diagram of Fig. 4.2 (Previous slide)

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

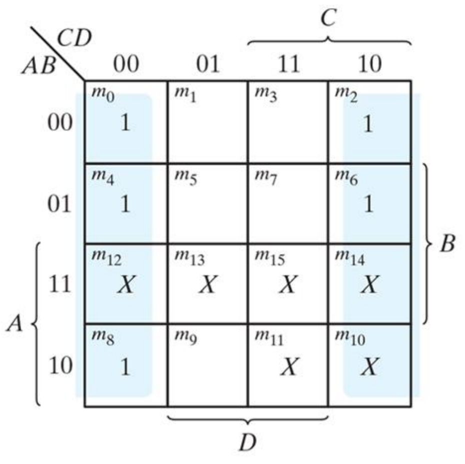
Design Procedure

- Procedure to design
  - Determine the required number of input and output from specification
  - Assign a letter symbol to each input/output
  - Derive the truth table
  - Obtain the simplified Boolean functions
  - Draw the logic diagram and verify design correctness

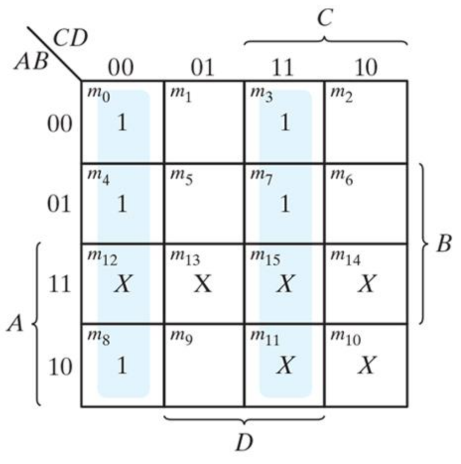
Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

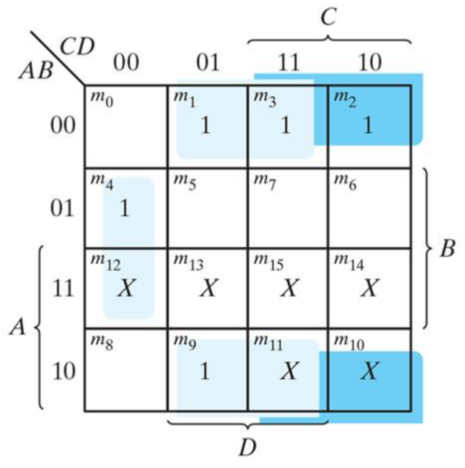
Maps for BCD-to-excess-3 code converter



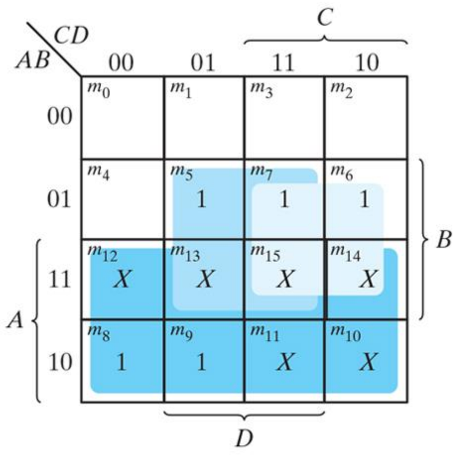
$z = D'$



$y = CD + C'D'$

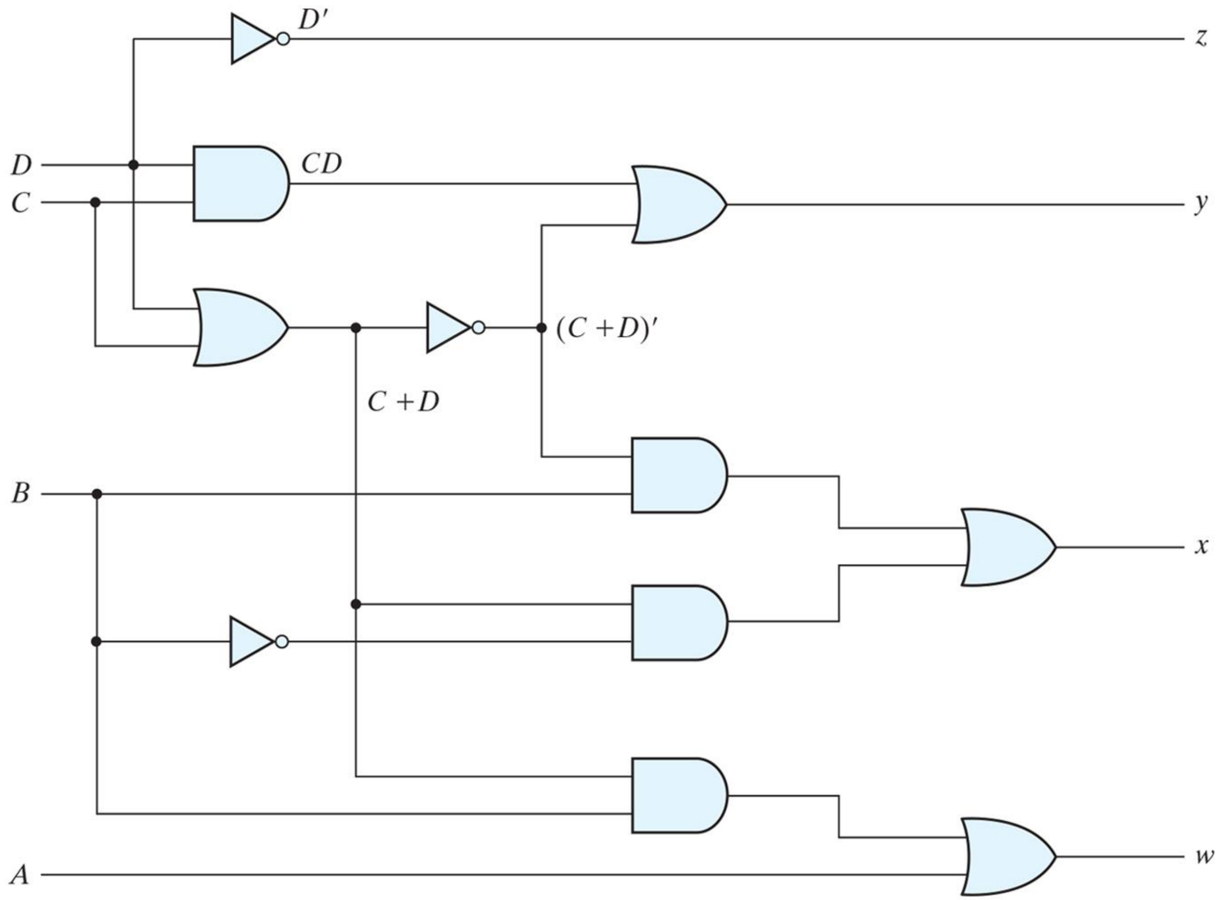


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

Logic diagram for BCD-to-excess-3 code converter



Half Adder

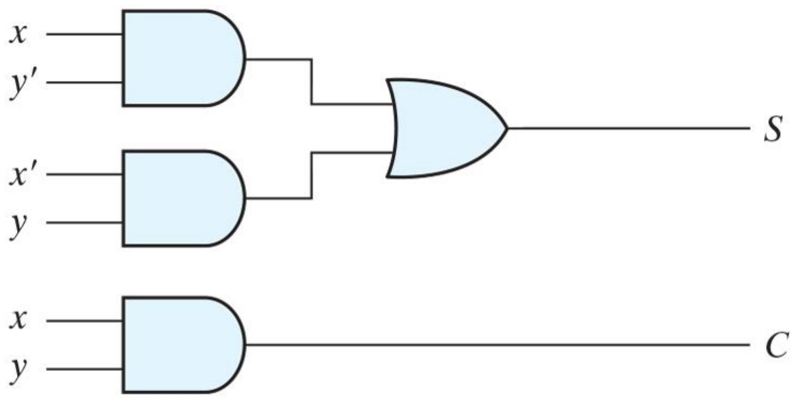
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Sum of 2 binary inputs
- Input : X(augend), Y(addend)
- Output : S(sum), C(carry)

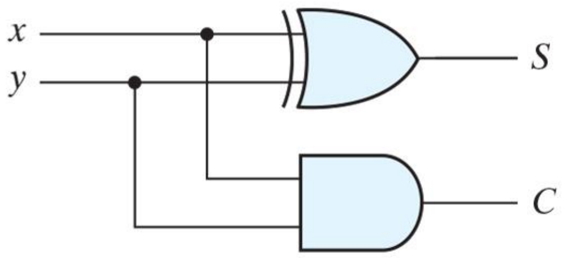
$$S=xy'+x'y$$

$$C=xy$$

Implementation of half adder



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



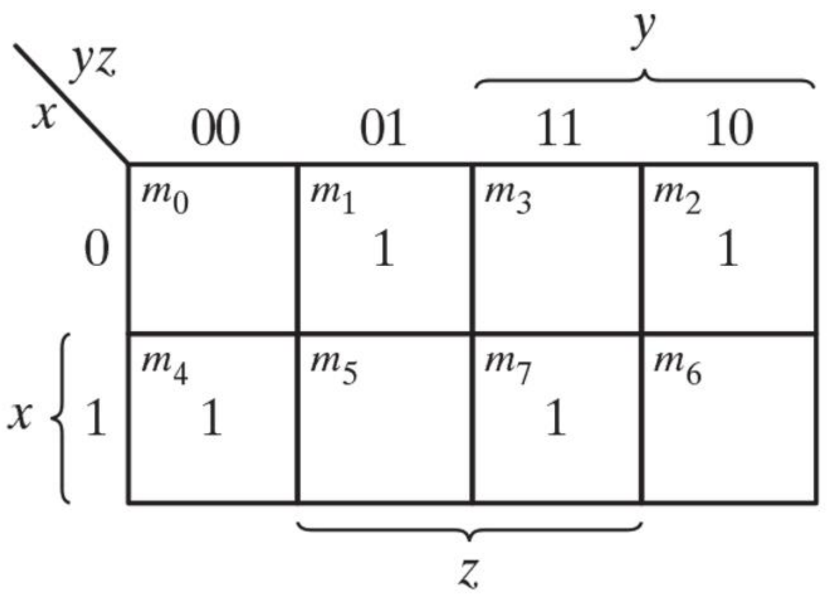
(b) $S = x \oplus y$
$C = xy$

Full Adder

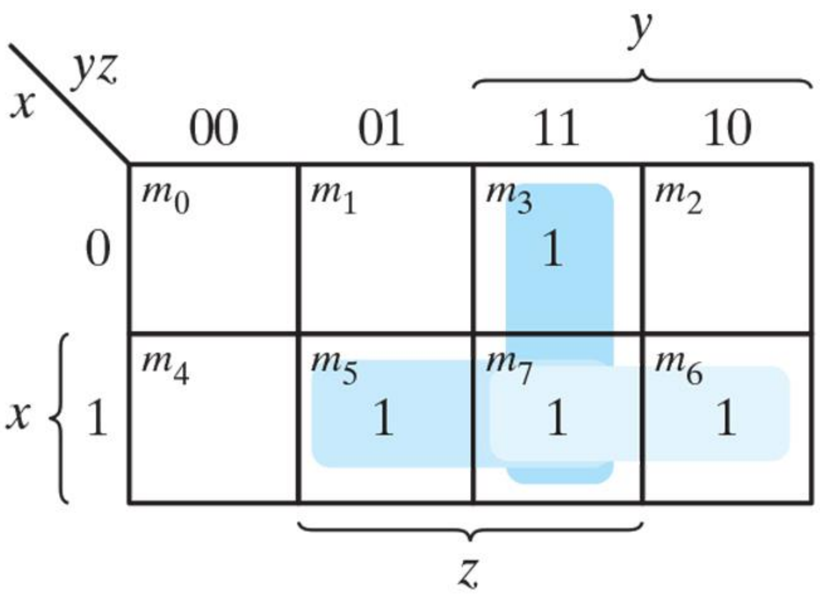
x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Sum of 3 binary inputs
- Input : X,Y(2 significant bits),Z(1 carry bit)
- Output : S(sum),C(carry)

K-Maps for full adder

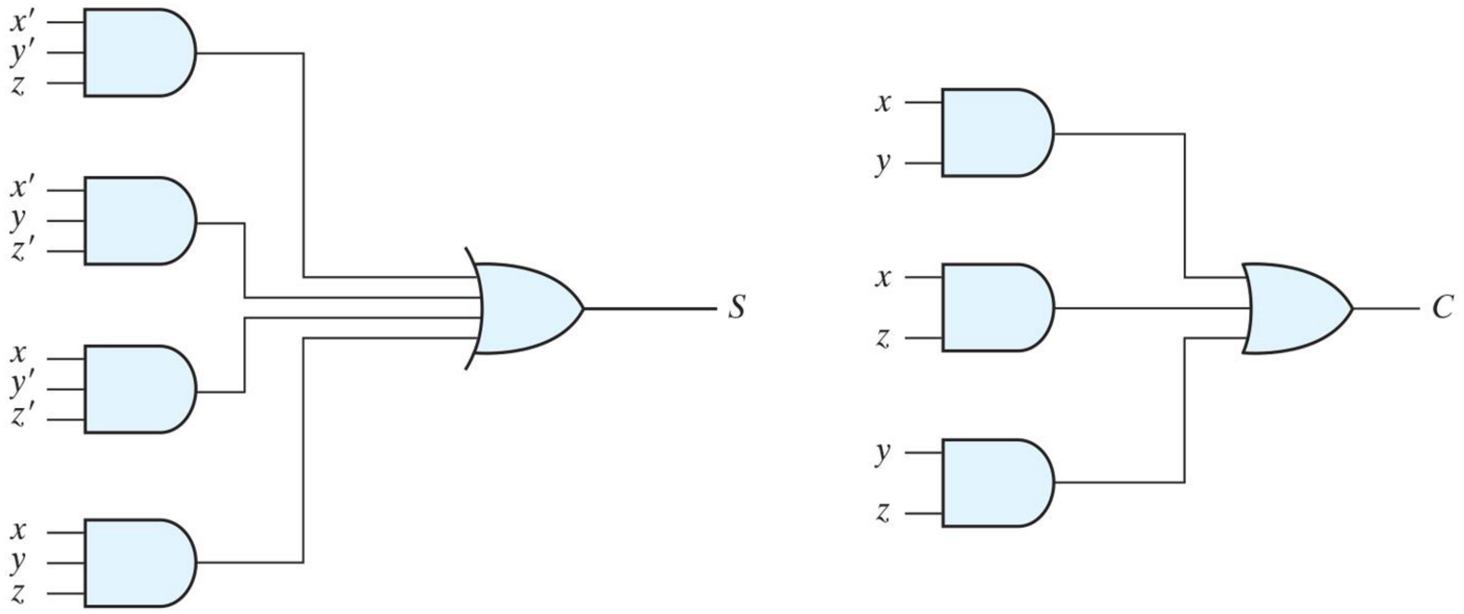


(a) $S = x'y'z + x'yz' + xy'z' + xyz$

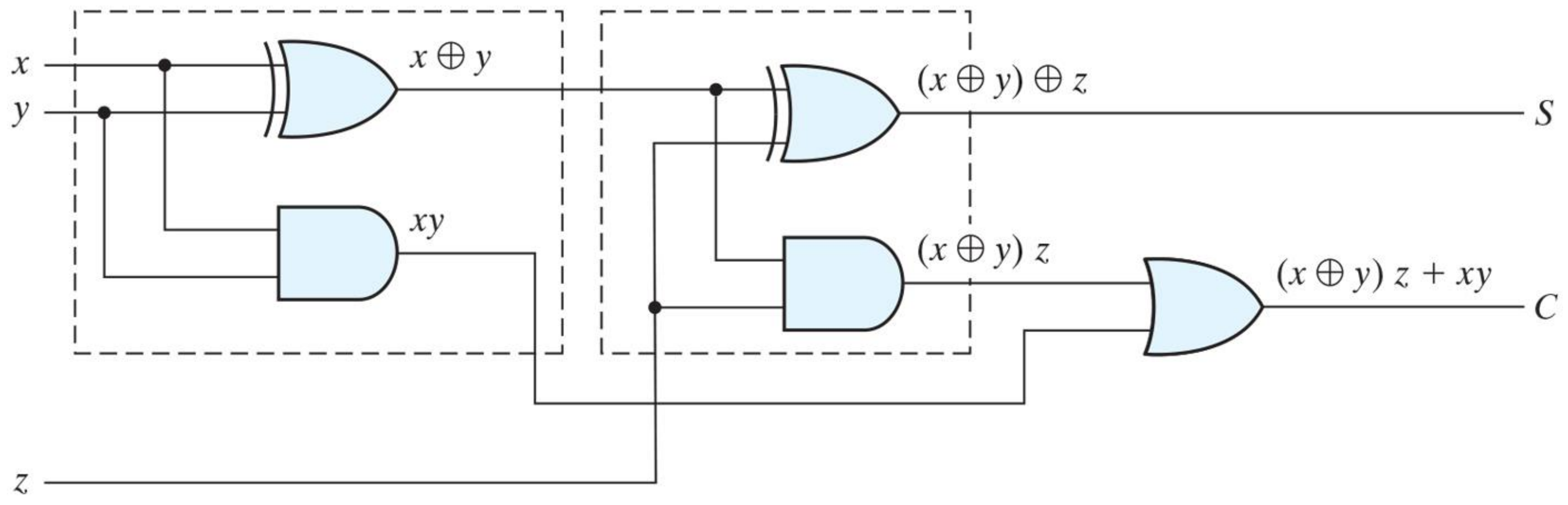


(b) $C = xy + xz + yz$

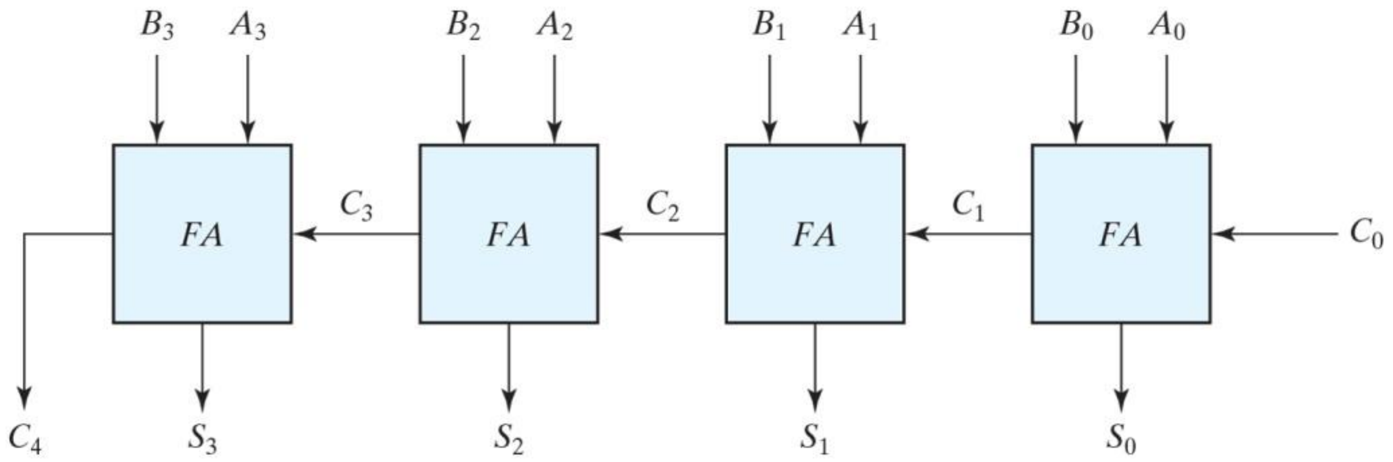
Implementation of full adder in sum-of-products form



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

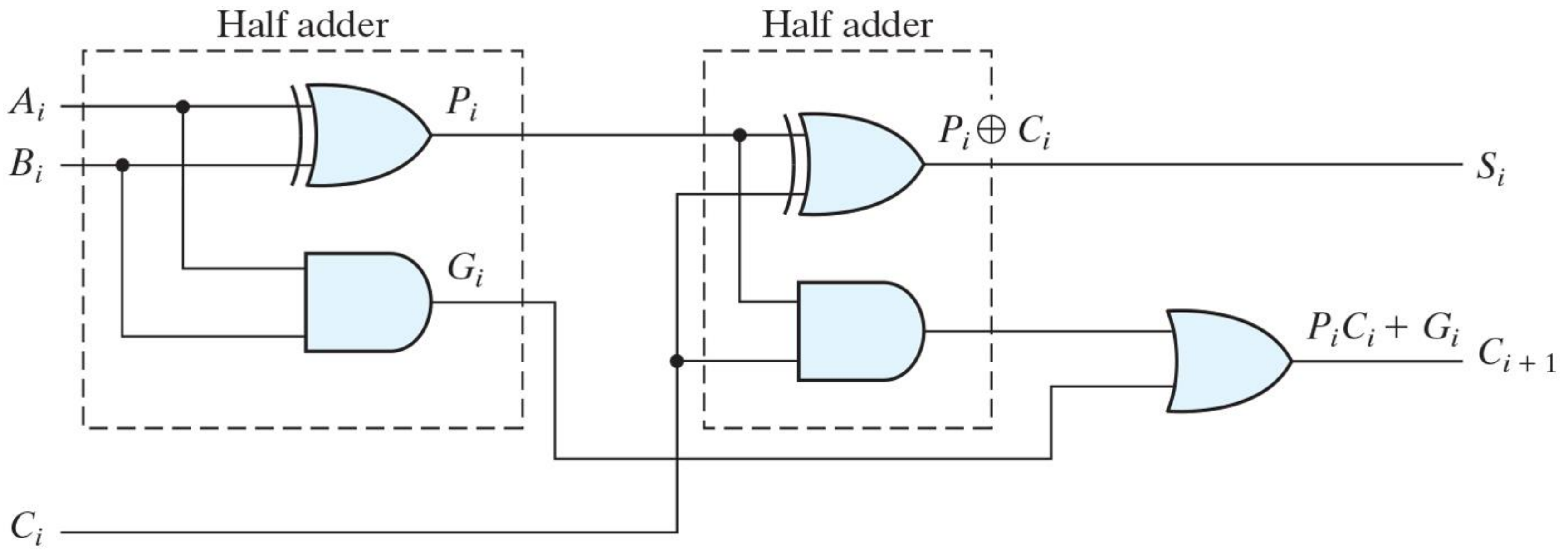


Four-bit adder



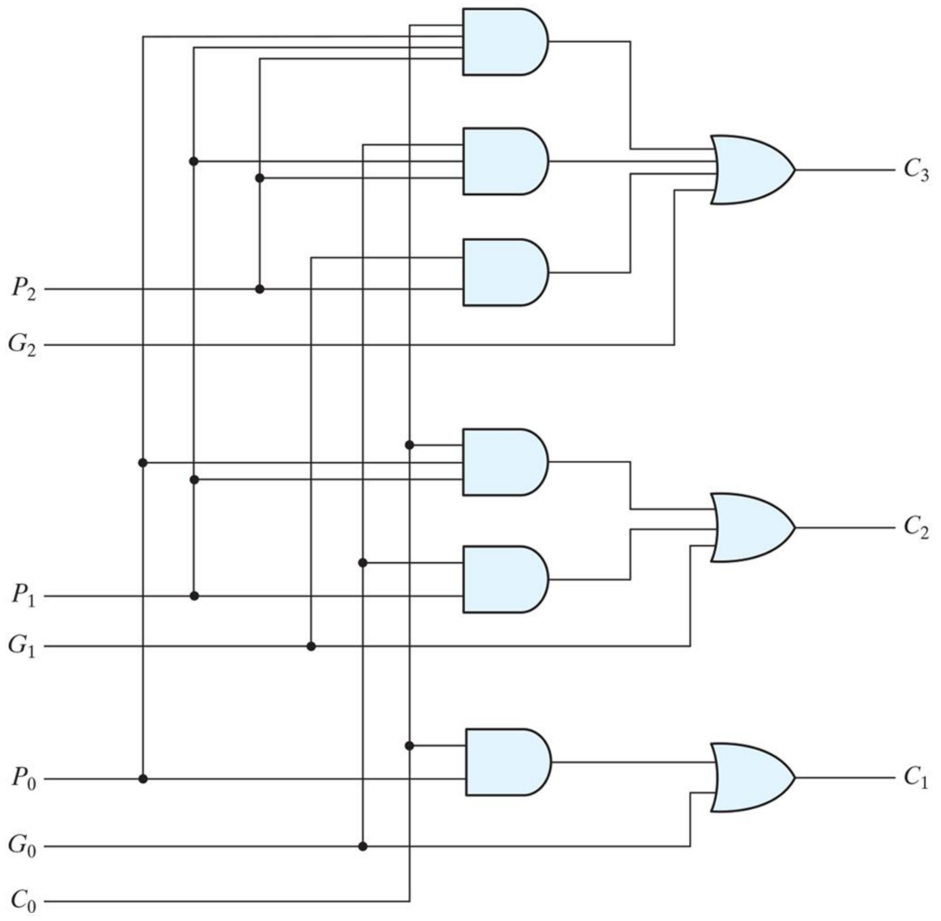
Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Full adder with *P* and *G* shown

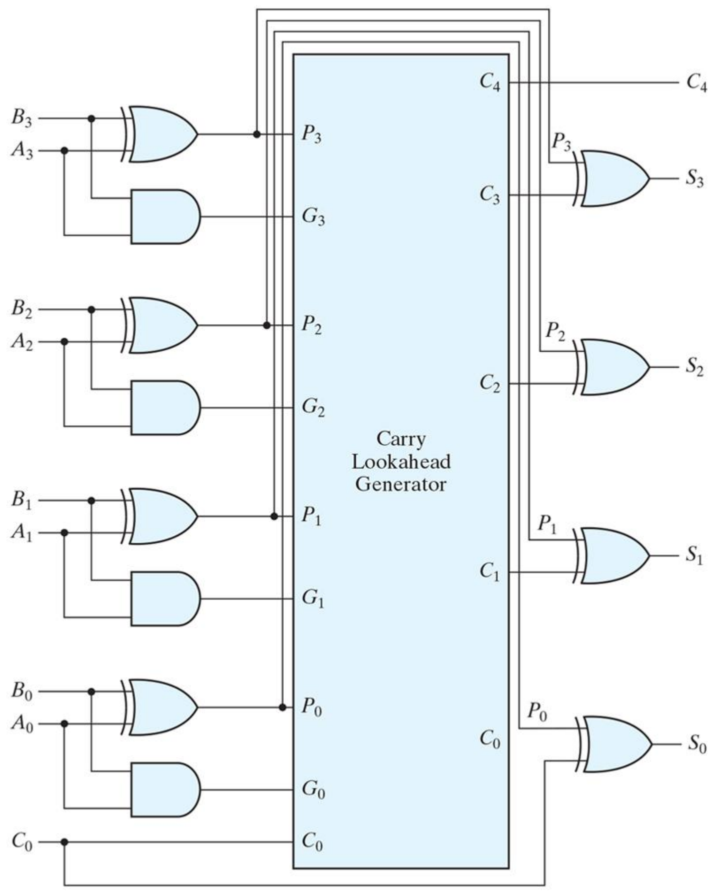


- Delay time increase (carry delay)
- One solution is ***carry lookahead***
- All carry is a function of P_i,G_i and C_0

Logic diagram of carry lookahead generator



Four-bit adder with carry lookahead



C_0 = input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

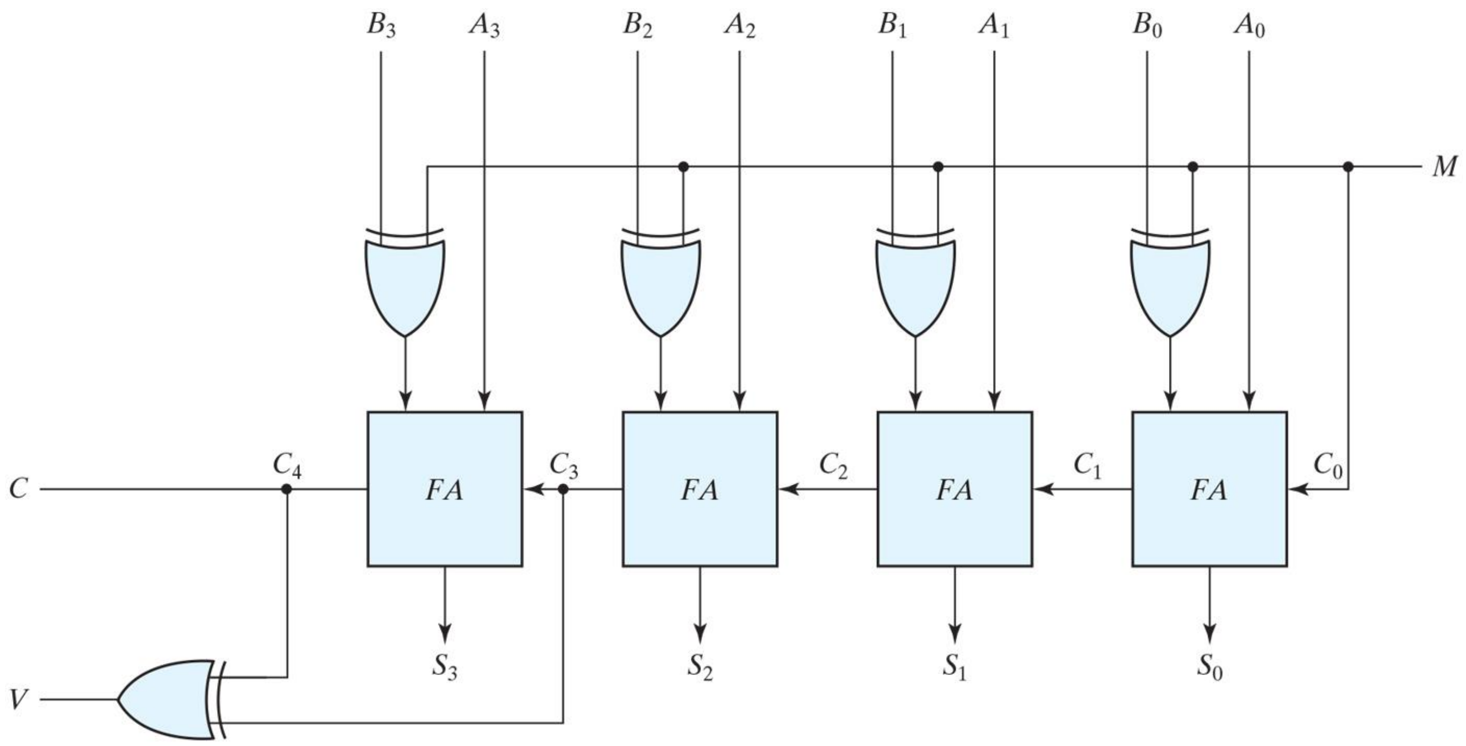
$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

Overflow

- Sum of *n* digit number occupies *n*+1digit
- Occurs when two numbers are same sign
- (examples of overflow)

carries:	0	1		carries:	1	0	
+70		0	1000110	−70		1	0111010
+80		0	1010000	−80		1	0110000
+150		1	0010110	−150		0	1101010

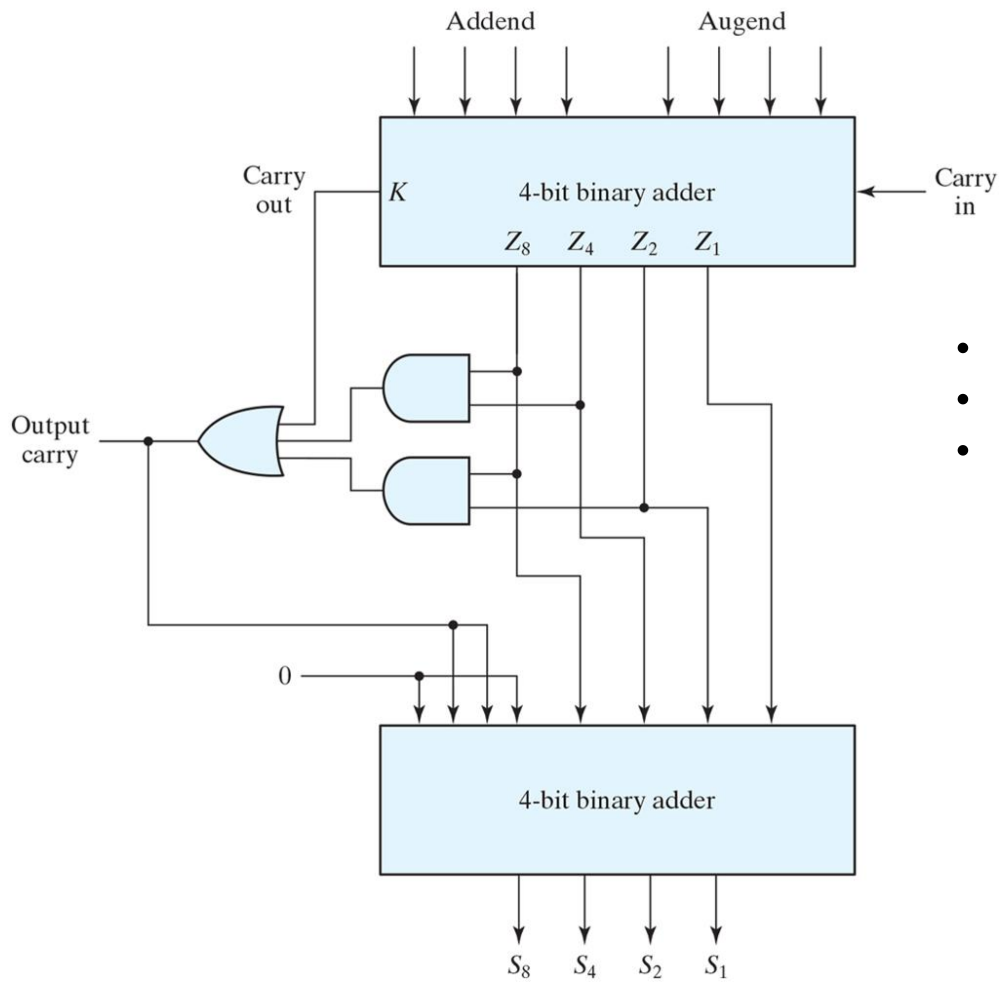
Four-bit adder–subtractor (with overflow detection)



Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Block diagram of a BCD adder

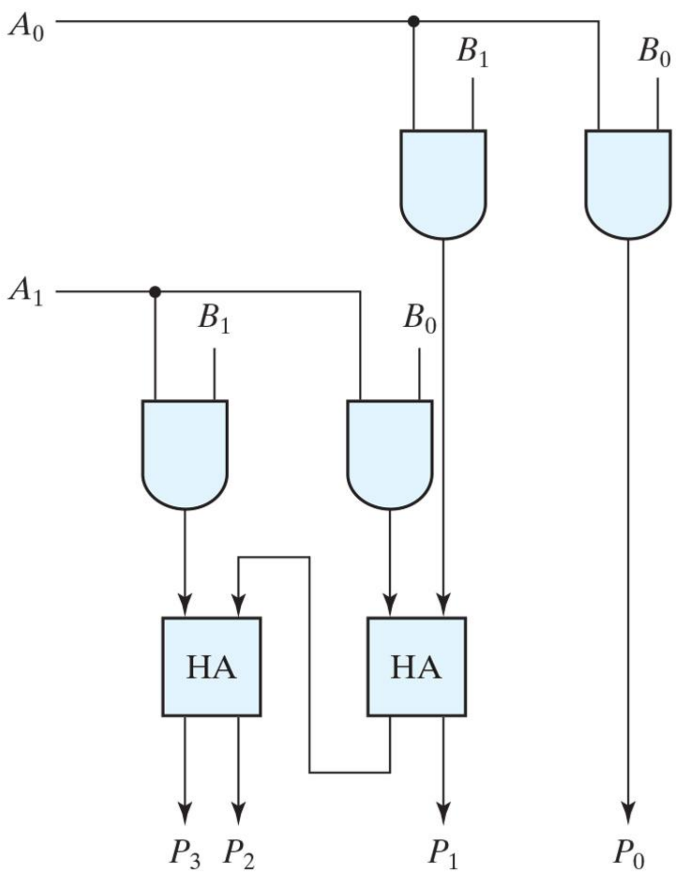


- BCD digit output of 2-BCD digit sum
- Carry occurs if output 1010~1111
- C=K+Z8Z4+Z8Z2

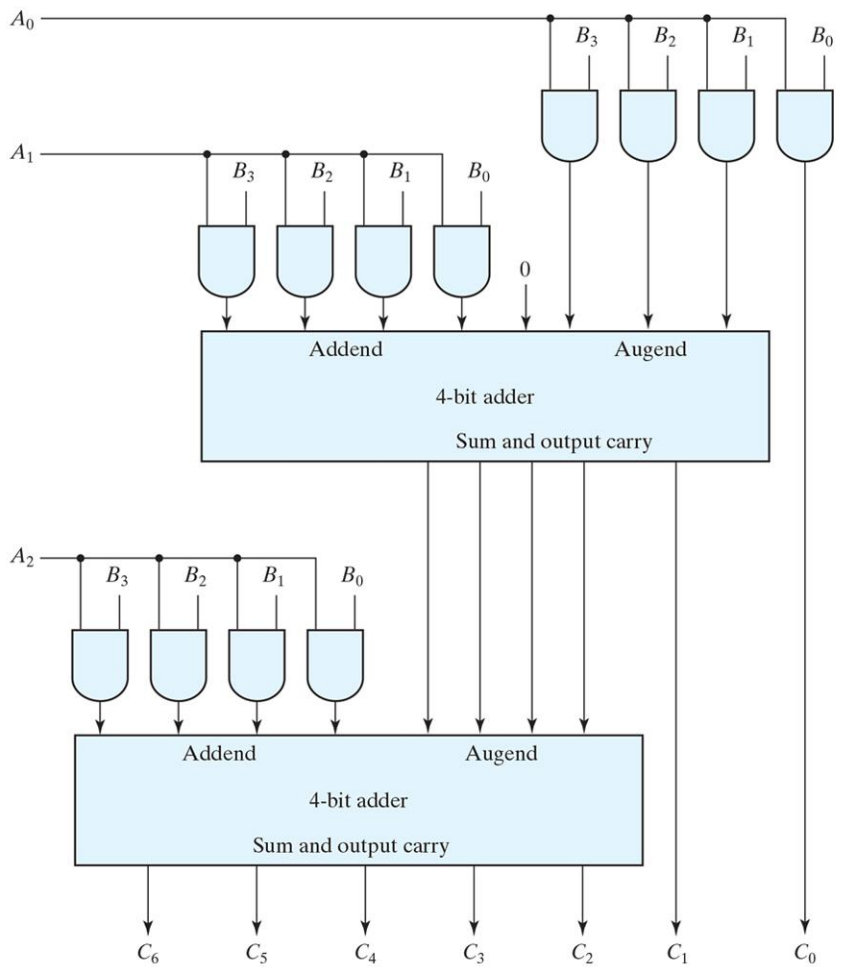
1100	1010
1101	1011
1110	
1111	

Two-bit by two-bit binary multiplier

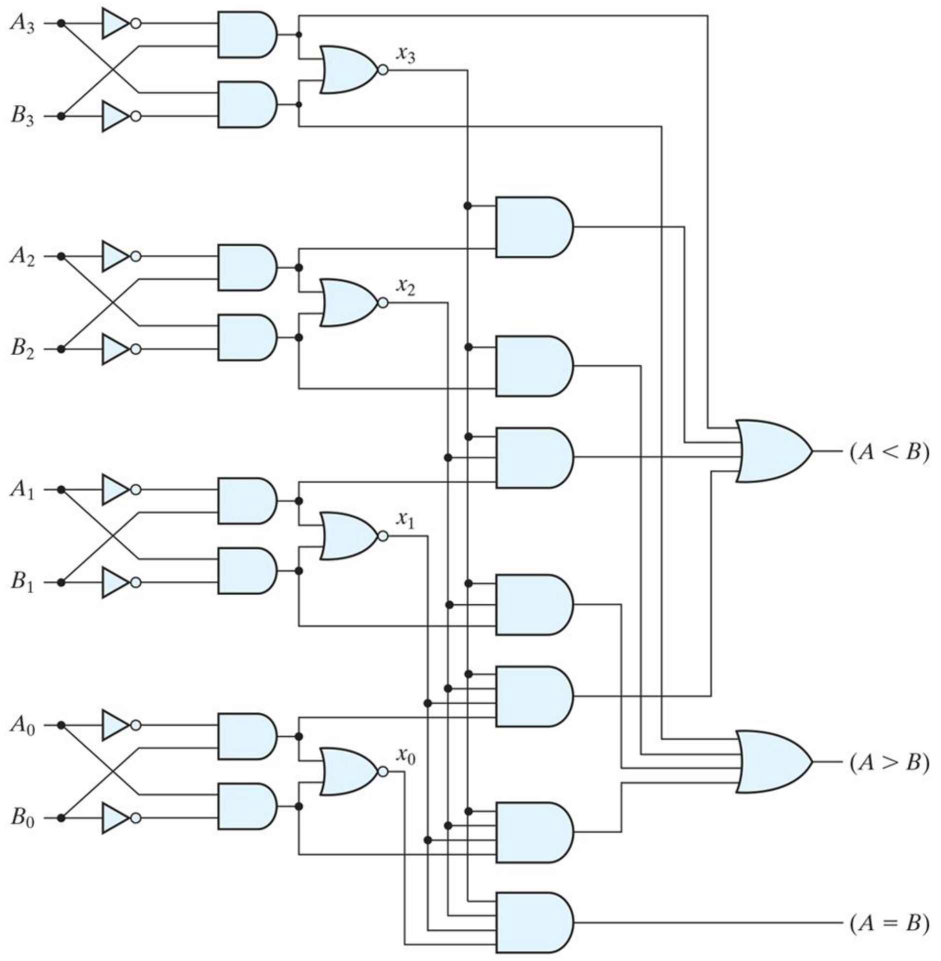
$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\cline{3-4}
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
P_3 & P_2 & P_1 & P_0
\end{array}
$$



Four-bit by three-bit binary multiplier



Four-bit magnitude comparator

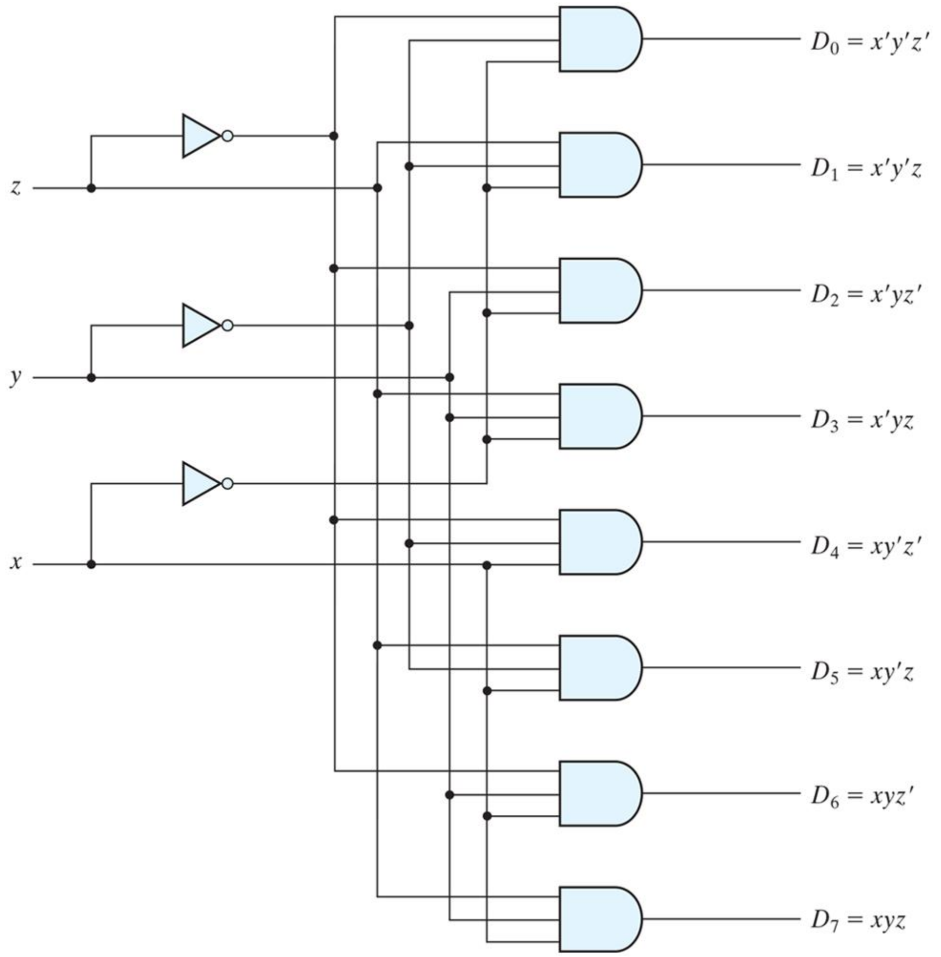


Decoder

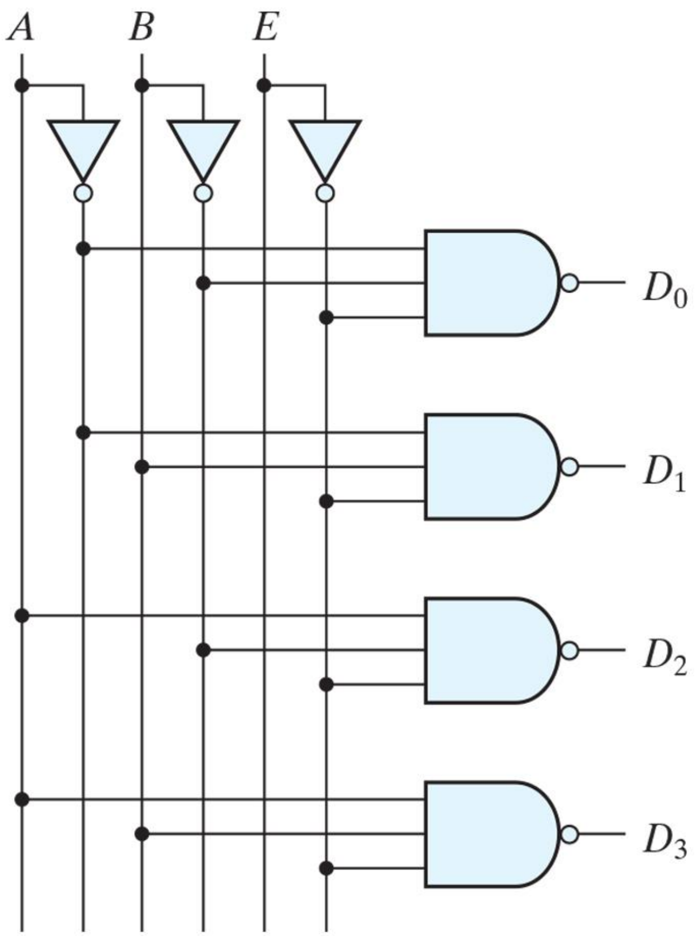
- Generate the 2^n(or less) minterms of n input variables
  - Eg)3 to 8 line decoder

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

Three-to-eight-line decoder



Two-to-four-line decoder with enable input



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

4 x 16 decoder constructed with two 3 x 8 decoders

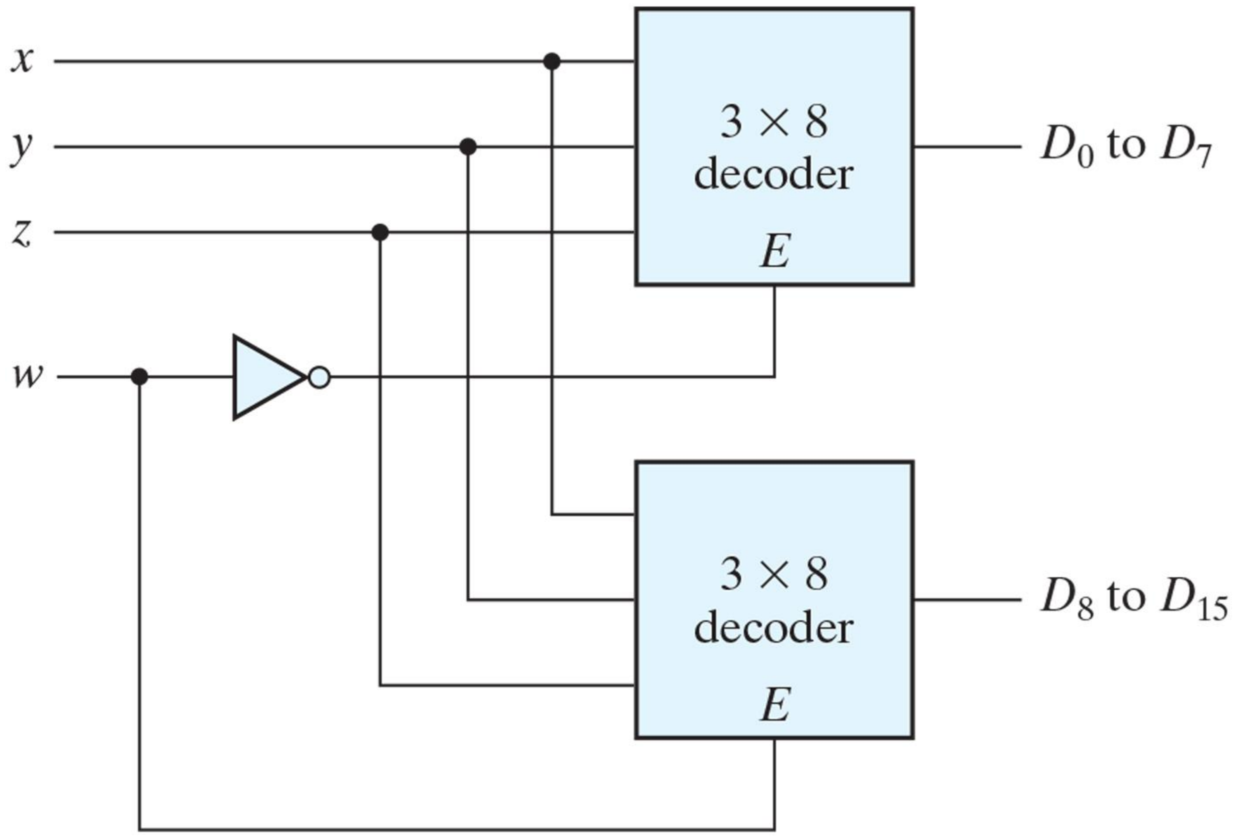
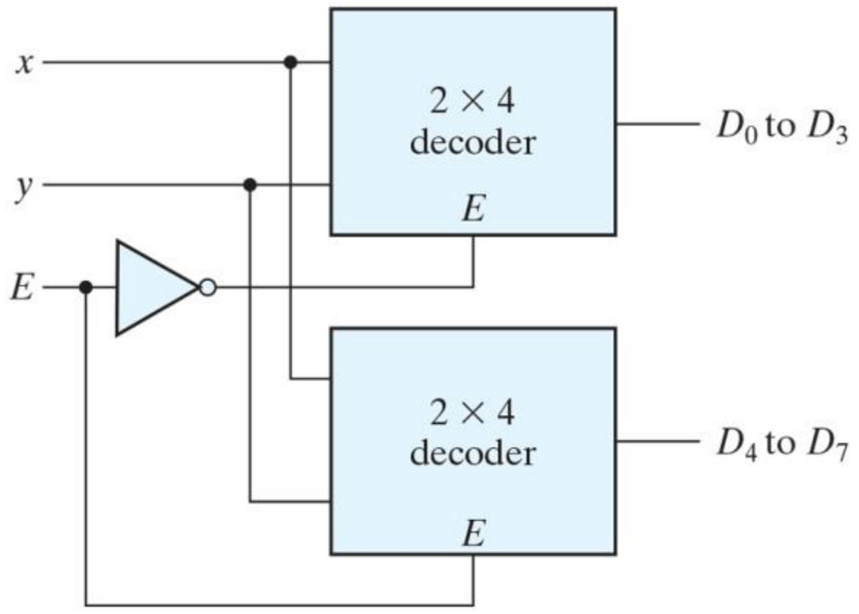


Figure PE4.8

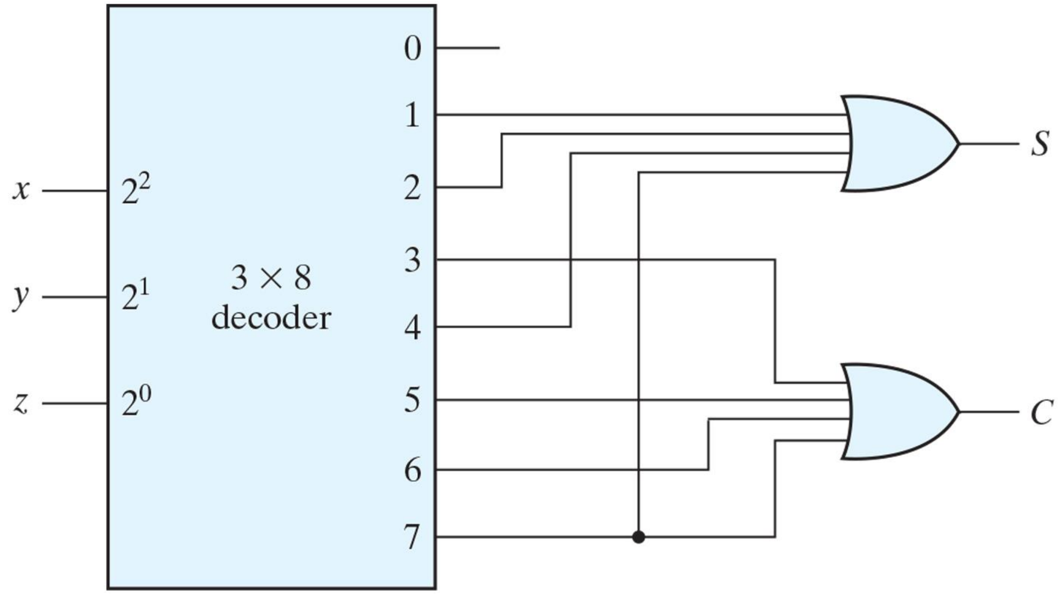


E	x	y	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation of a full adder with a decoder

Table 4-4
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



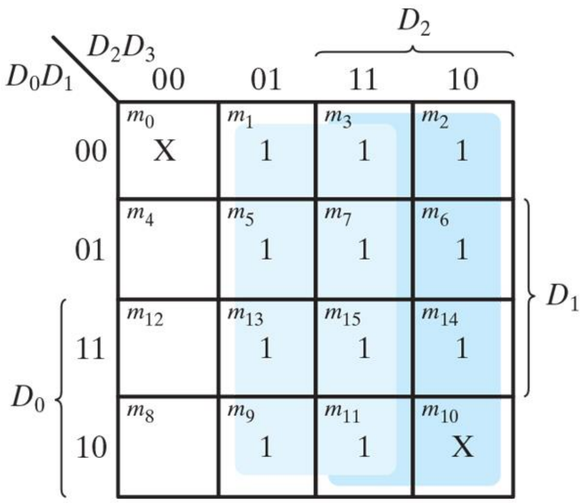
Encoder

- Inverse operation of a decoder
- Generate n outputs of 2^n input values
  - Eg) octal to binary encoder

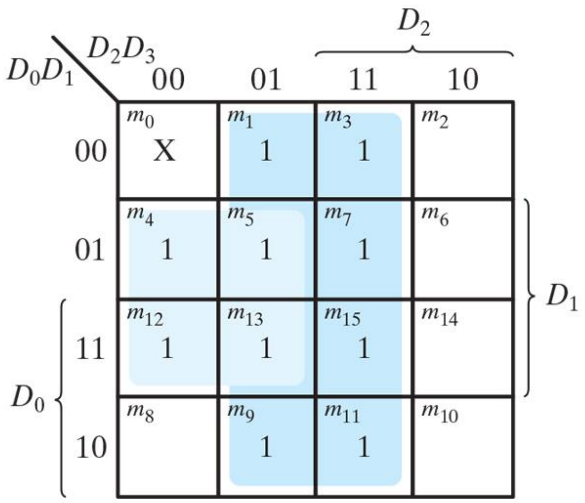
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

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

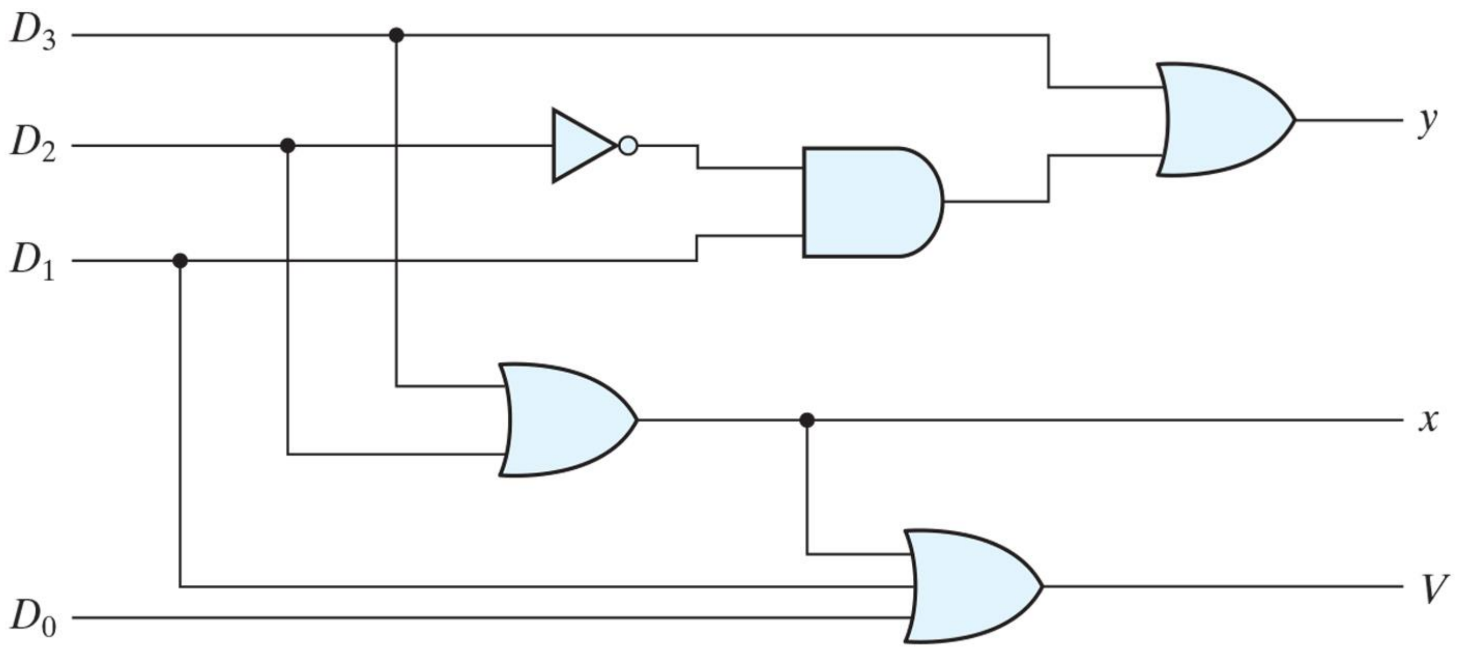


$x = D_2 + D_3$



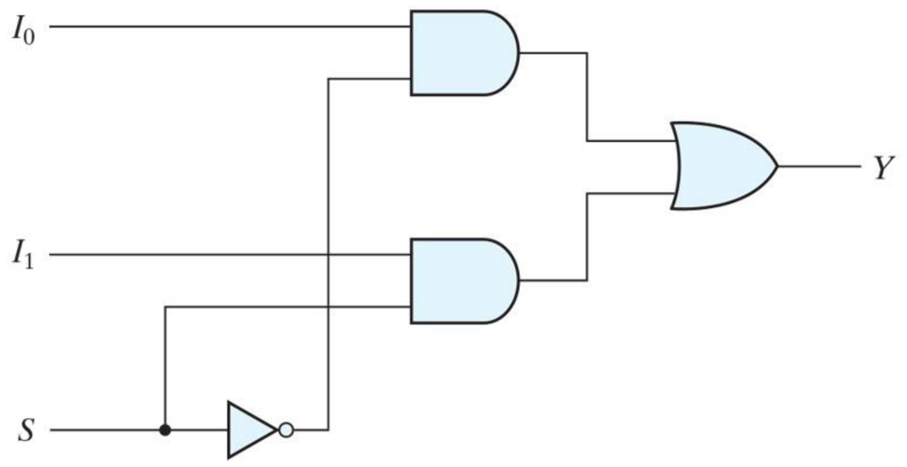
$y = D_3 + D_1D'_2$

Four-input priority encoder

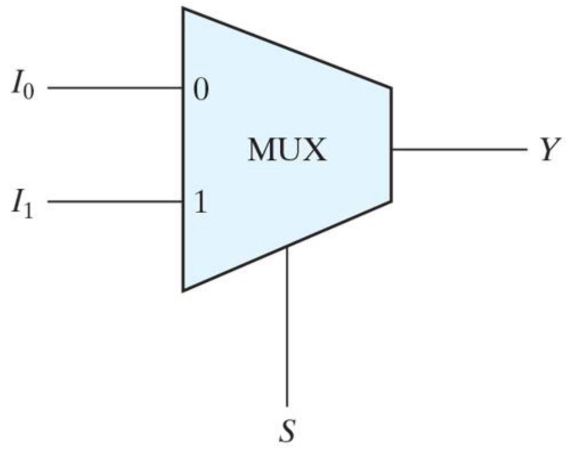


Multiplexer (MUX)

- Select a binary information from many input lines
- Selection is controlled by a set of selection lines
- 2^n input lines have *n* selection lines

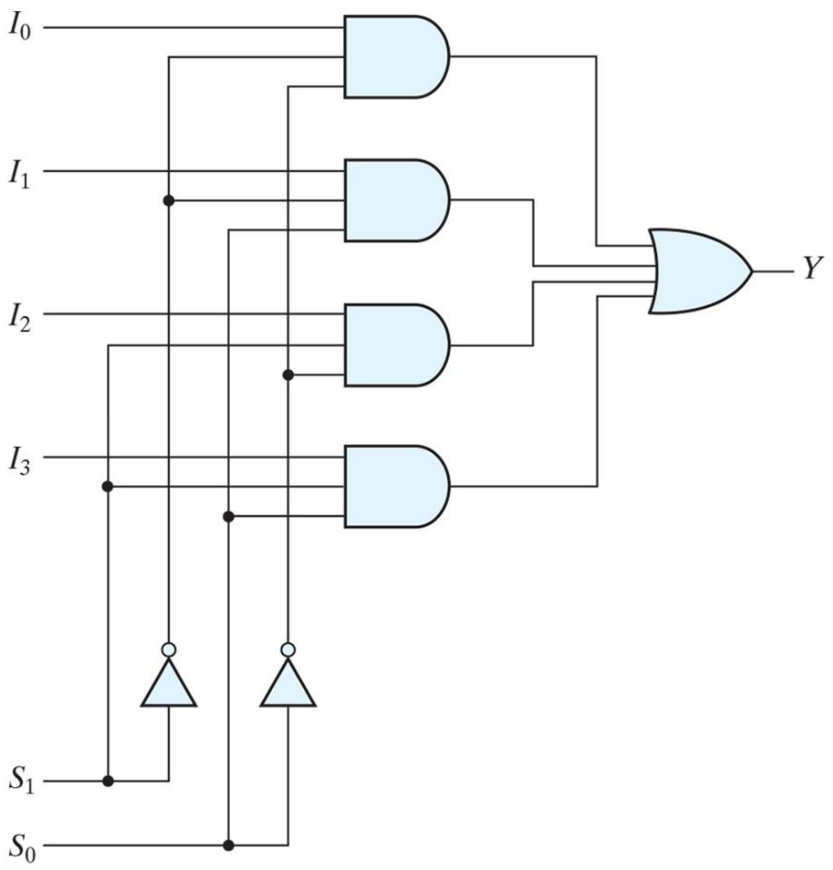


(a) Logic diagram



(b) Block diagram

Four-to-one-line multiplexer

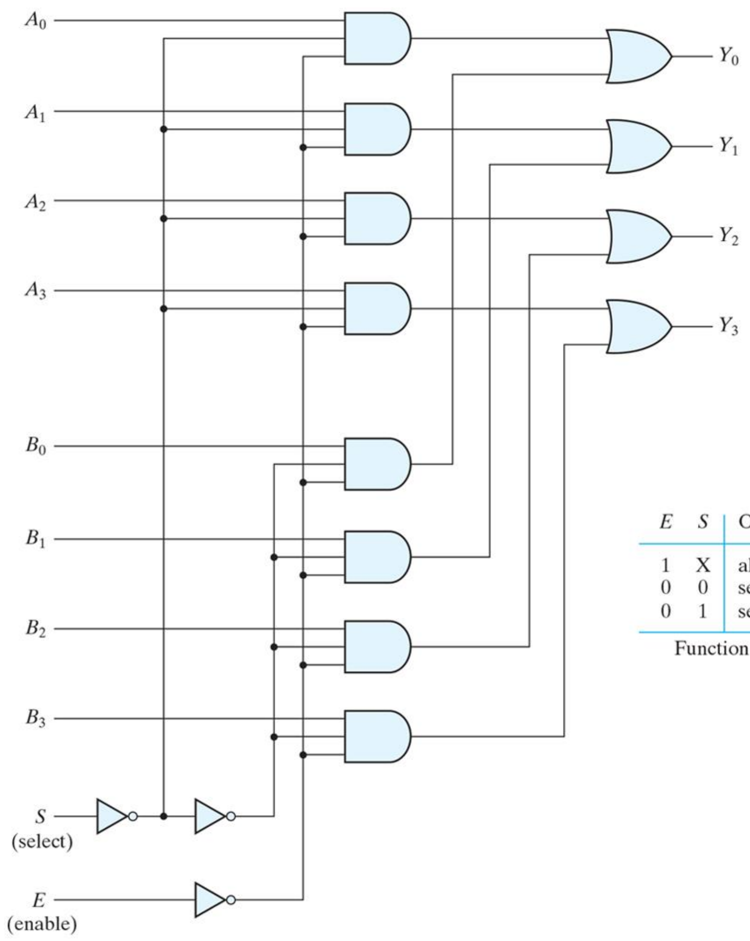


(a) Logic diagram

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

(b) Function table

Quadruple two-to-one-line multiplexer



E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

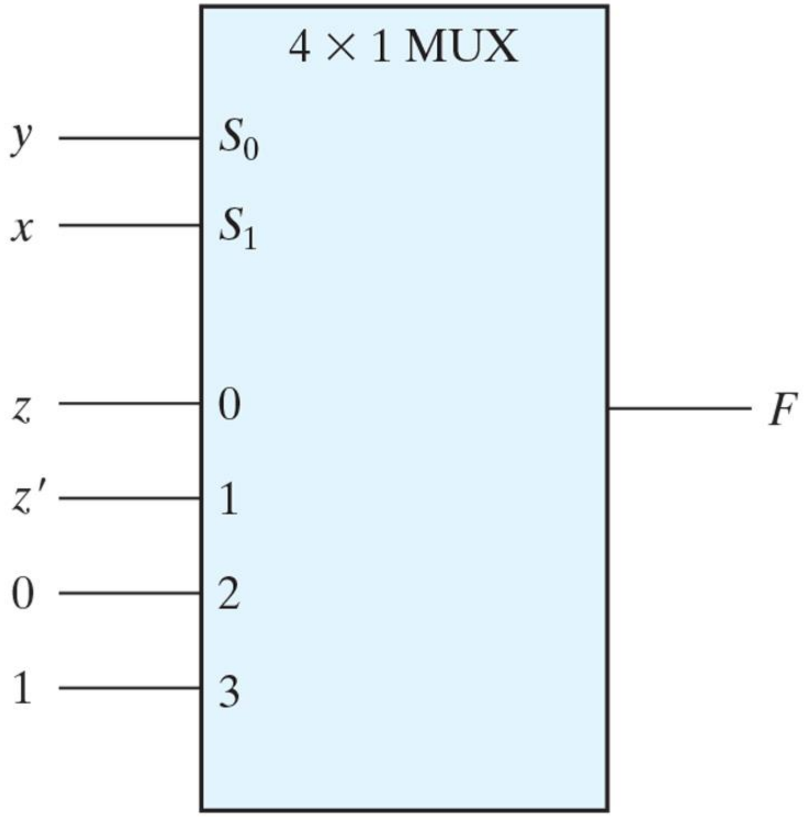
Function table

Implementing a Boolean function with a multiplexer

- Minterms of function are generated in a MUX
- n input variables, n-1 selection input

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	

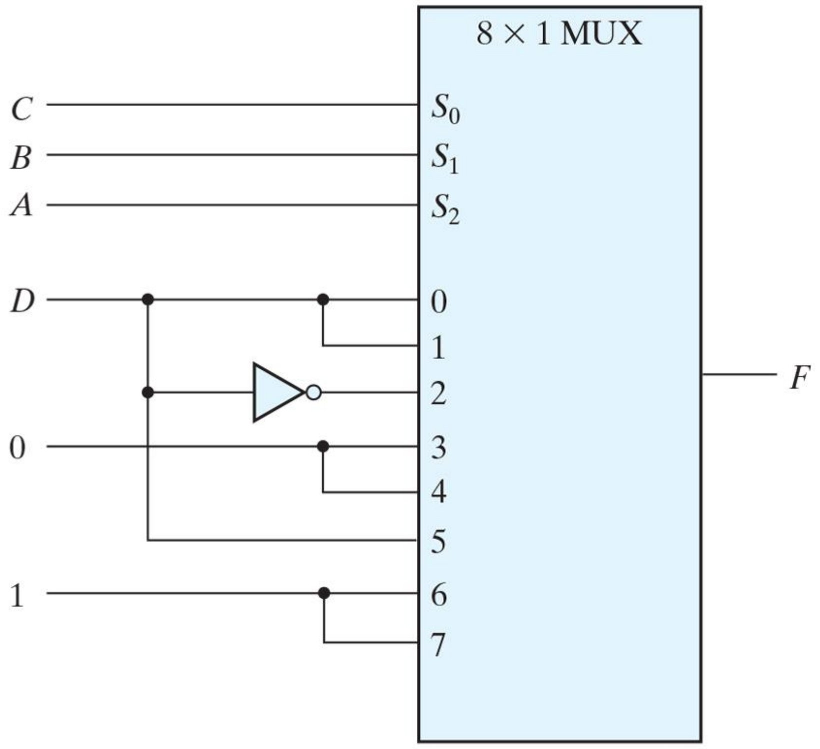
(a) Truth table



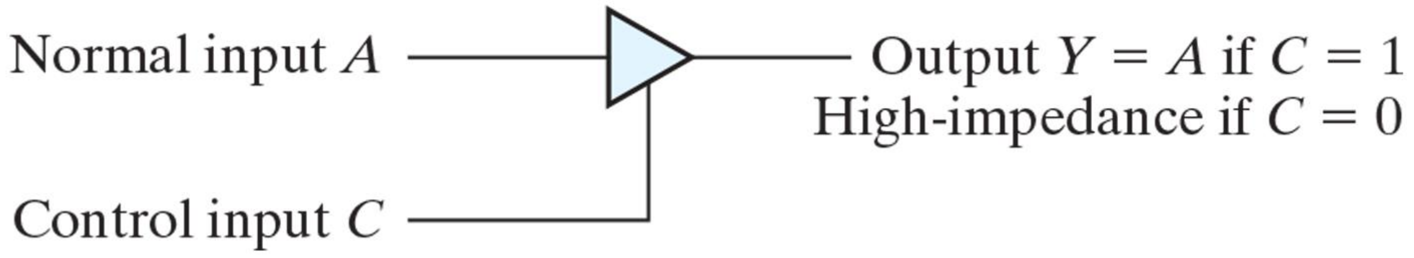
(b) Multiplexer implementation

Implementing a four-input function with a multiplexer

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	

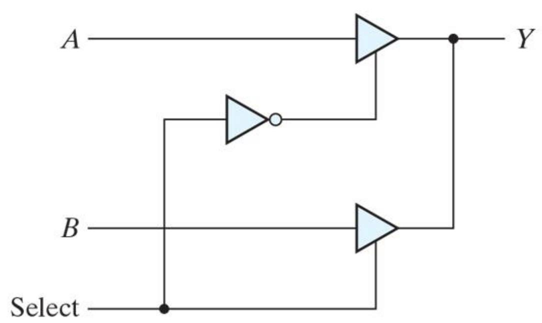


Graphic symbol for a three-state buffer

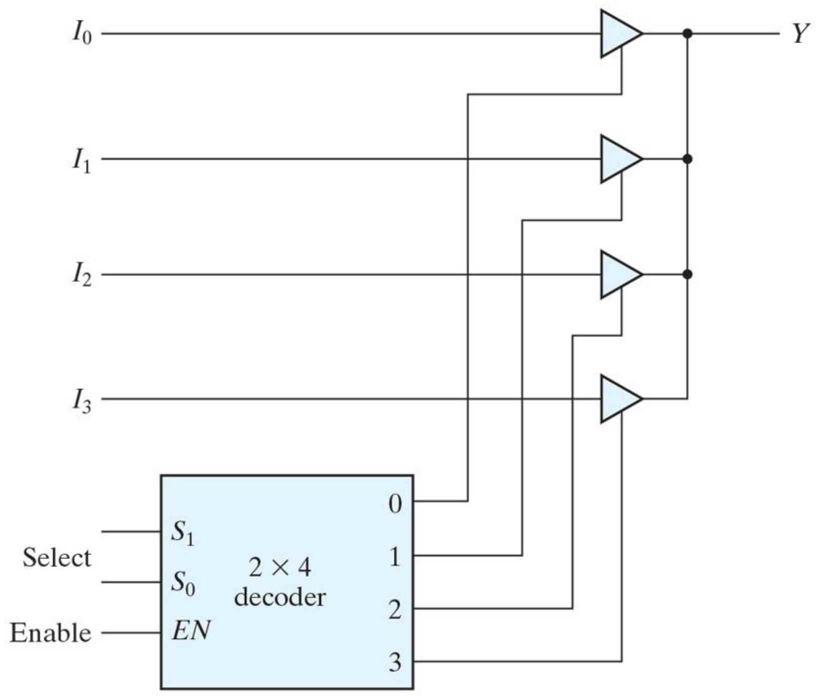


High-impedance behaves like an open circuit

Multiplexers with three-state gates



(a) 2-to-1-line mux



(b) 4-to-1-line mux