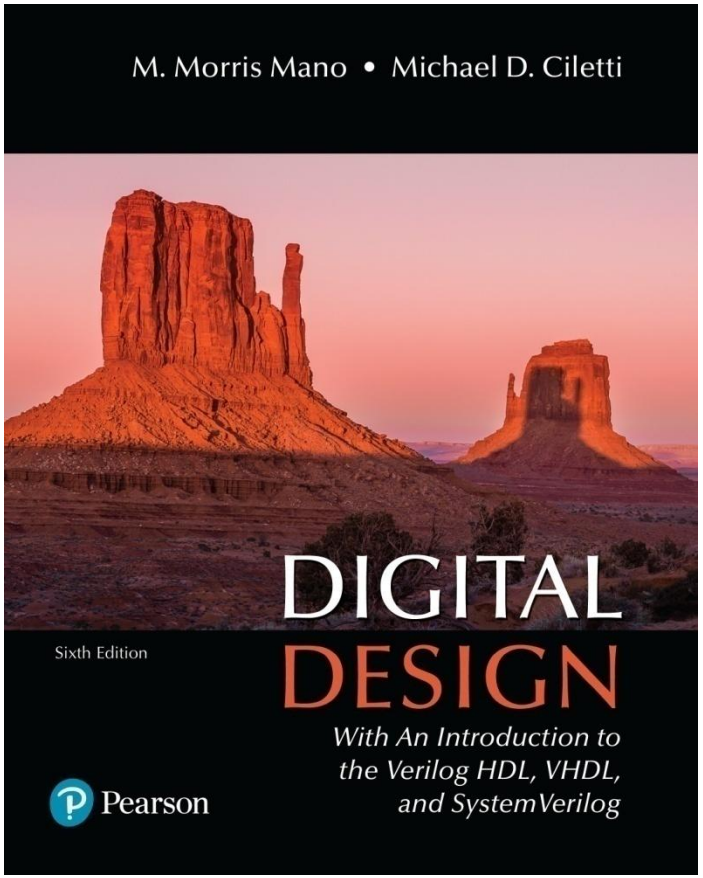
Digital Design

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

6th Edition



Chapter 04

Combinational Logic

Pearson

Figure 4.1
Block diagram of combinational circuit.

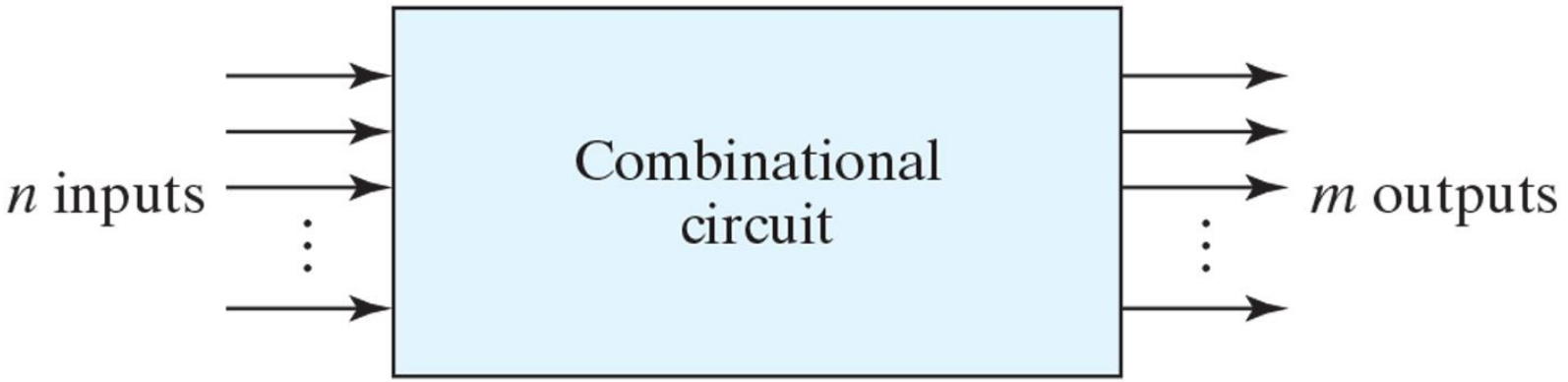


Figure 4.2
Logic diagram for analysis example.

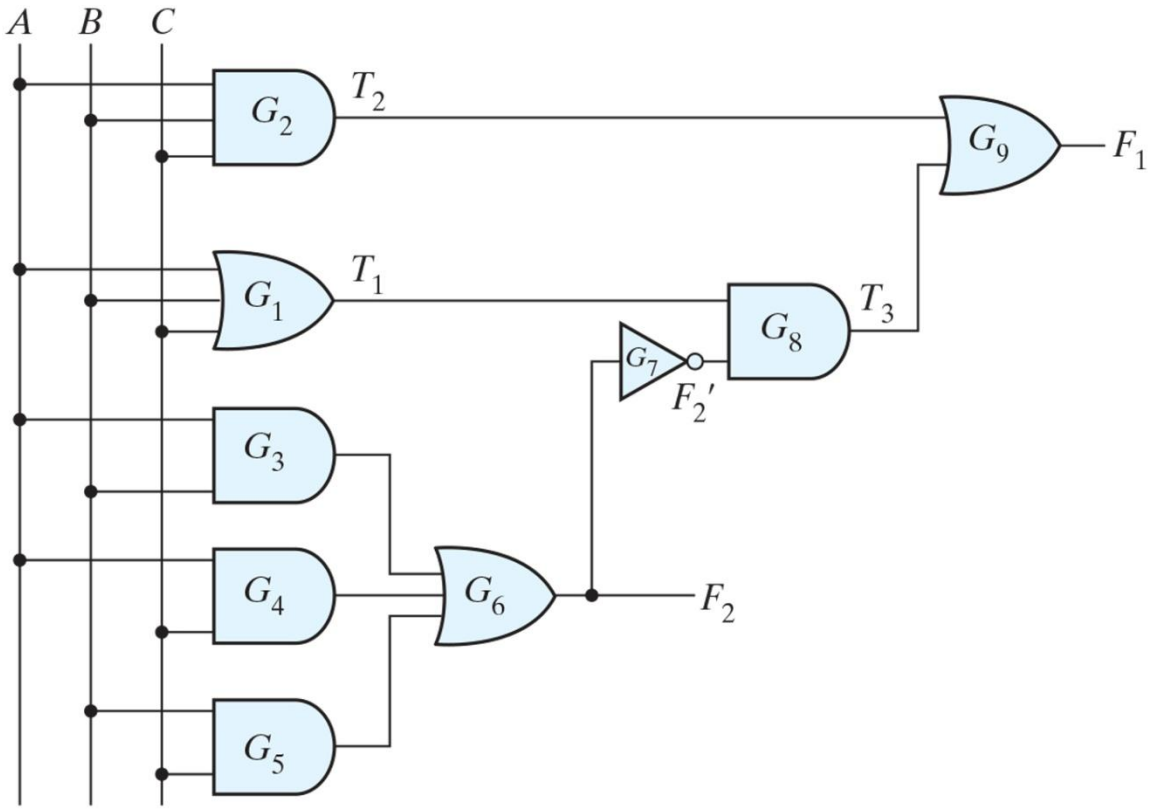


Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2.

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

Figure PE4.1

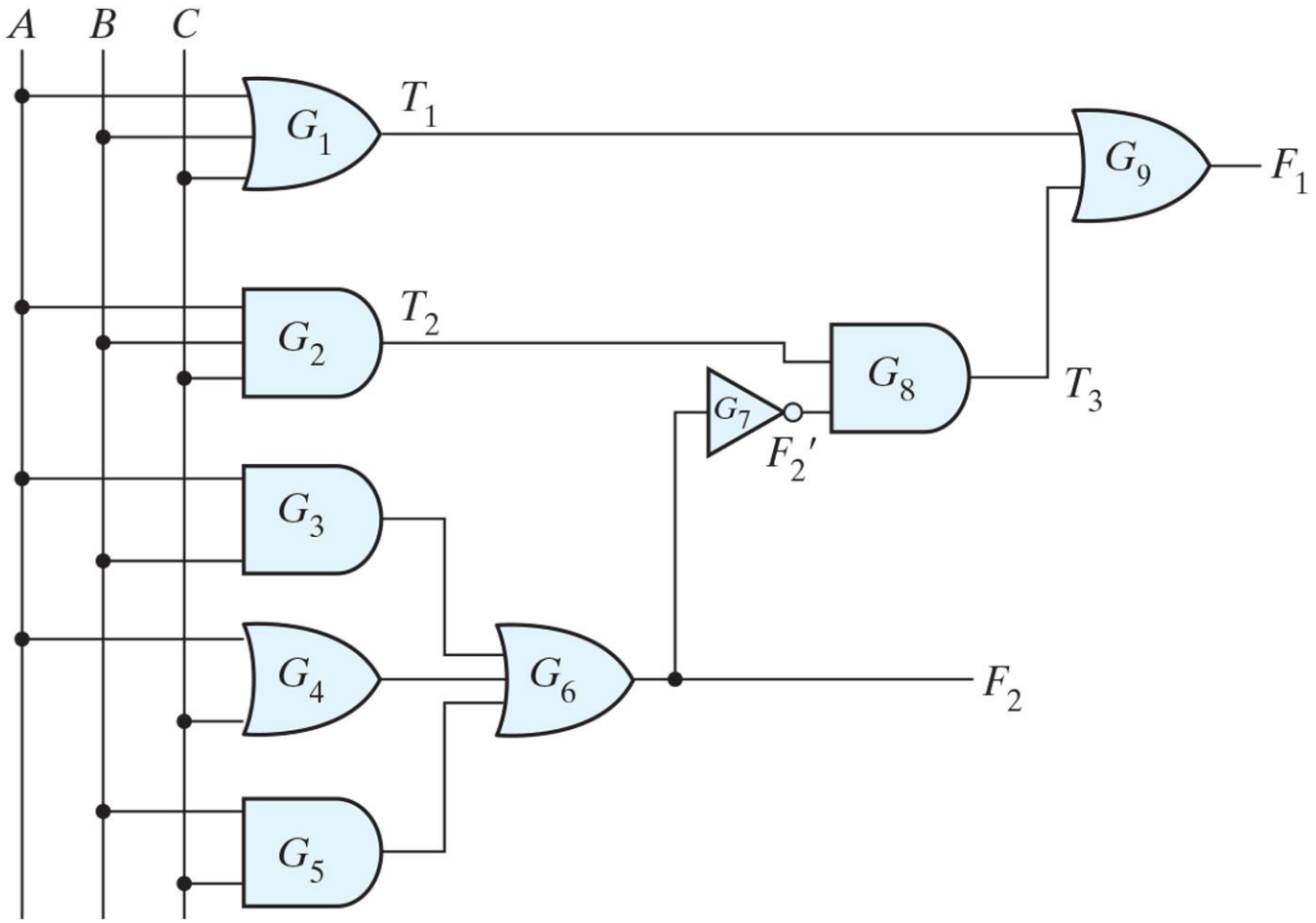


Table 4.2
Truth Table for Code Conversion Example.

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

Figure 4.3
Maps for BCD-to-excess-3 code converter.

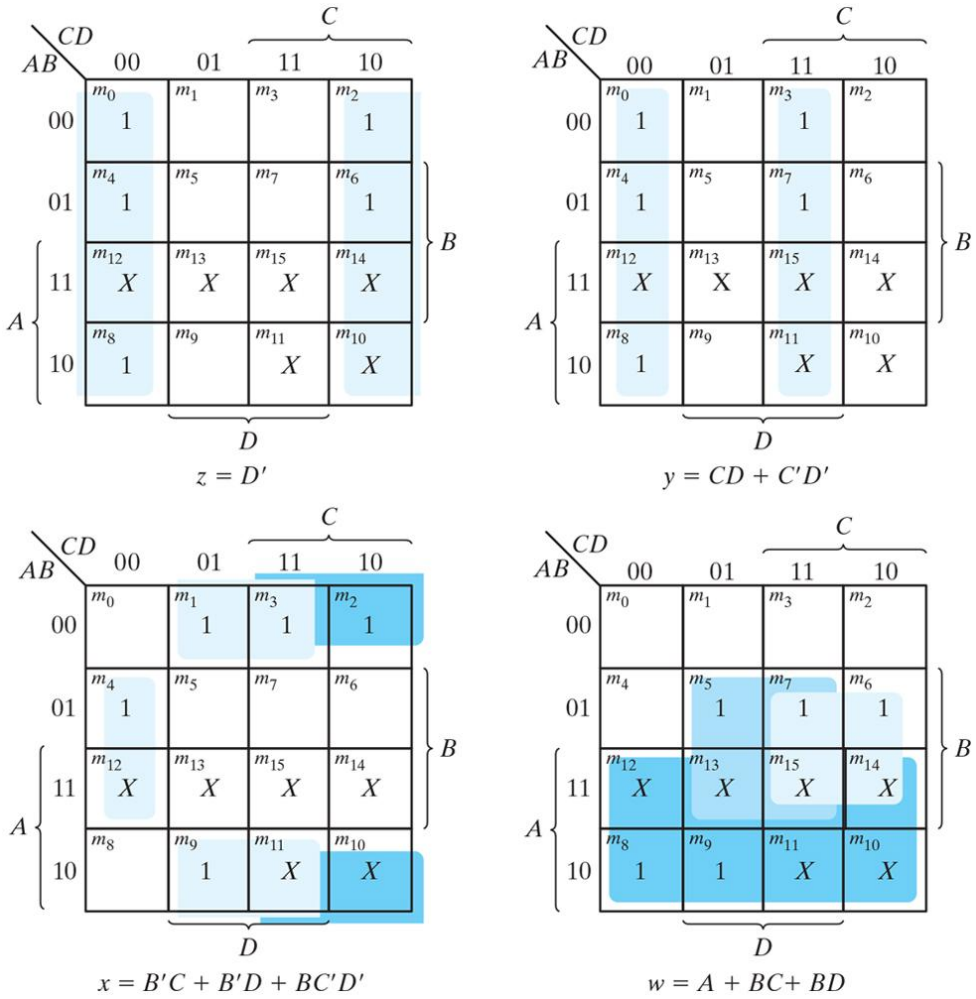


Figure 4.4
Logic diagram for BCD-to-excess-3 code converter.

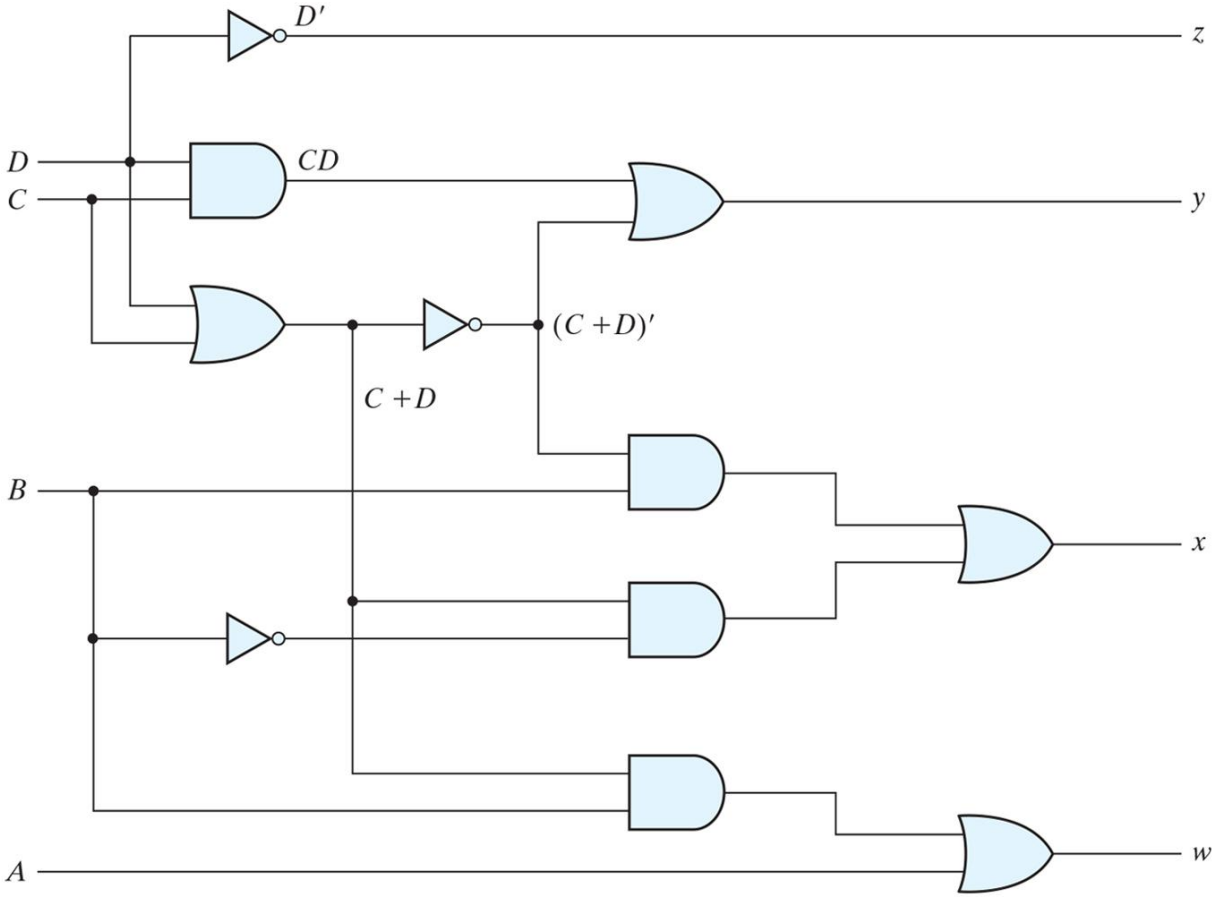
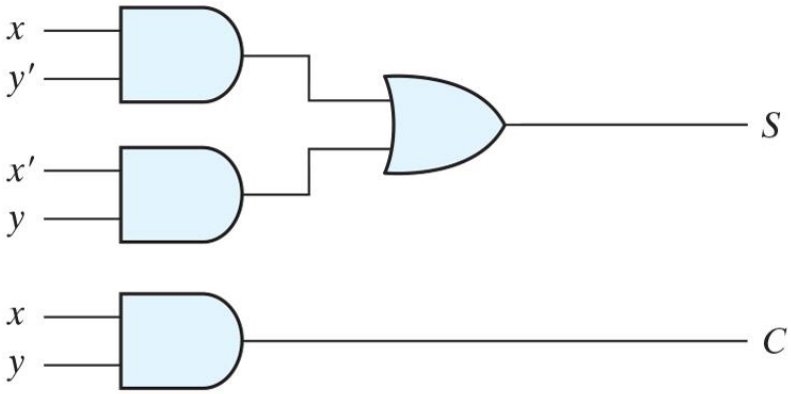


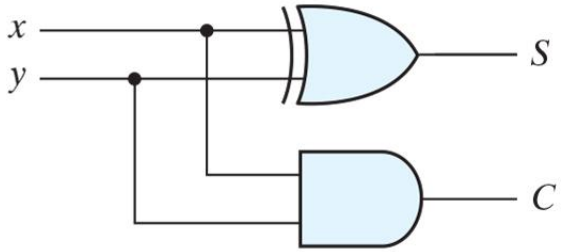
Table 4.3
Half Adder.

x	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 4.5
Implementation of half adder.



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

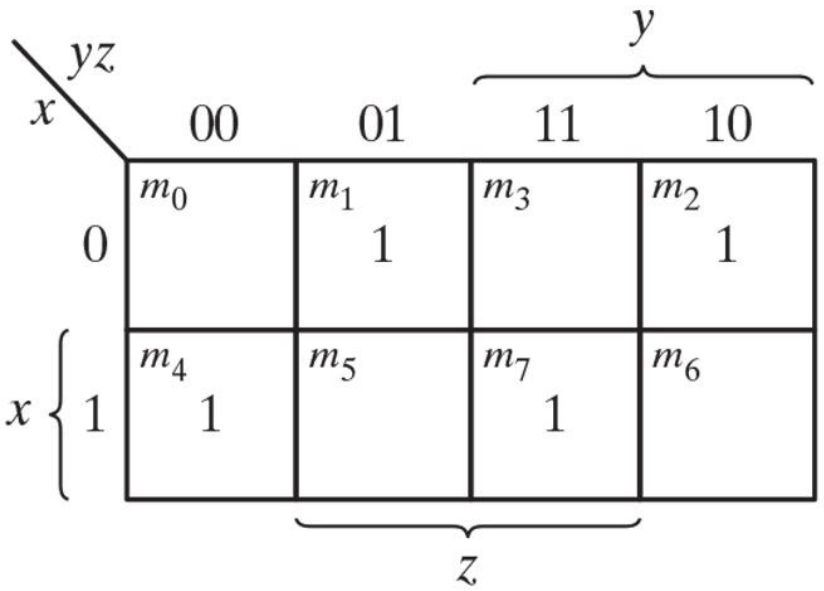


(b) $S = x \oplus y$
$C = xy$

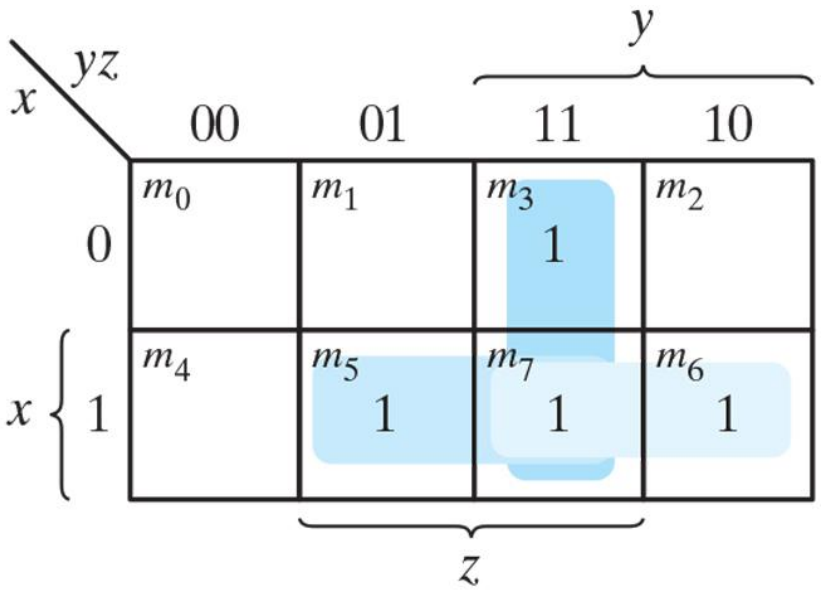
Table 4.4
Full Adder.

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

Figure 4.6
K-Maps for full adder.

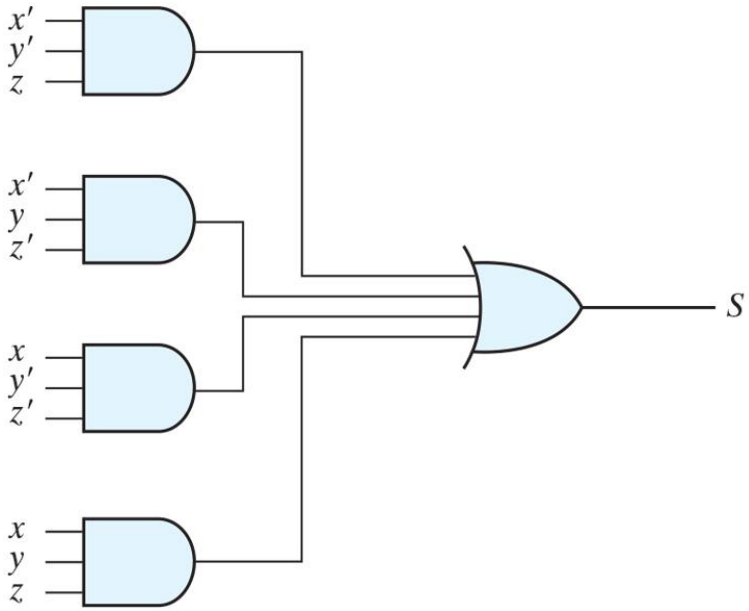


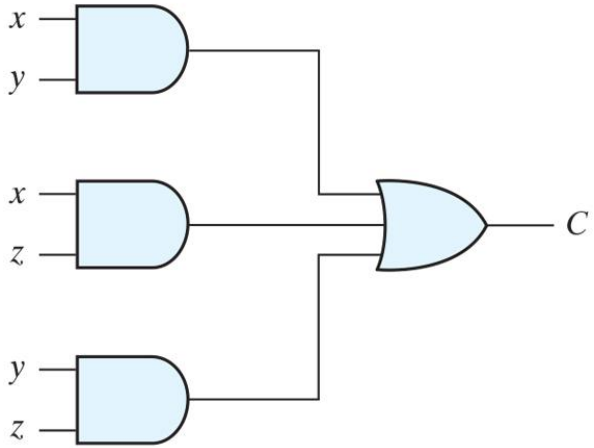
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



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

Figure 4.7
Implementation of full adder in sum-of-products form.



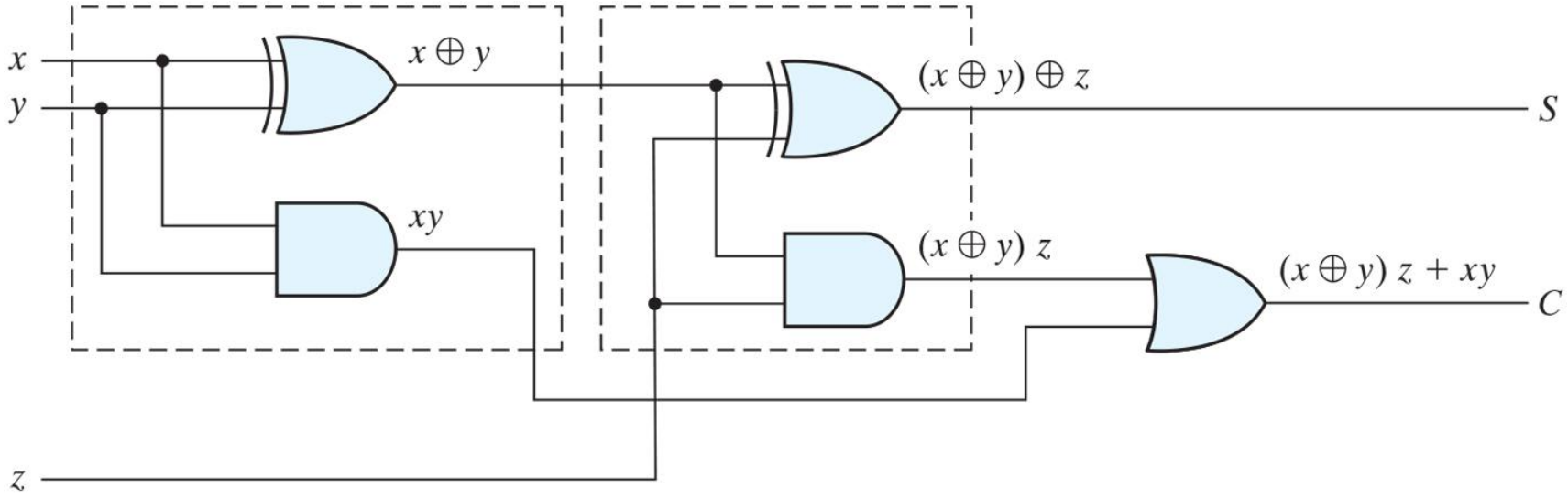


S=x'y'z+x'yz'+xy'z'+xyz
=(x'y'z+xyz)+x'yz'+xy'z'
=(x'y'+xy)z+(x'y+xy')z
=(x xor y)'z + (x xor y)z
=x xor y xor z

C=xy+yz+xz=xy+xyz+x'yz+xz
= xy+xyz+x'yz+xy'z
= xy+xyz+(x XOR y)z
= xy + (x XOR y)z

Pearson

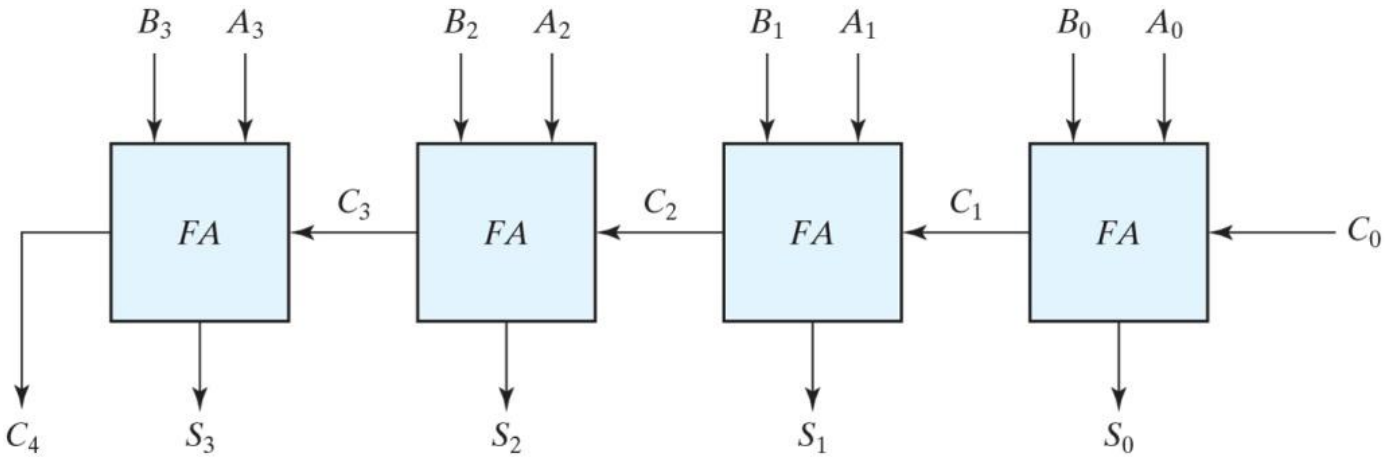
Figure 4.8
Implementation of full adder with two half adders and an OR gate.



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

C=xy+yz+xz=xy+xyz+x'yz+xz
= xy+x'yz+xyz+xy'z
= xy+xyz+(x XOR y)z
= xy + (x XOR y)z

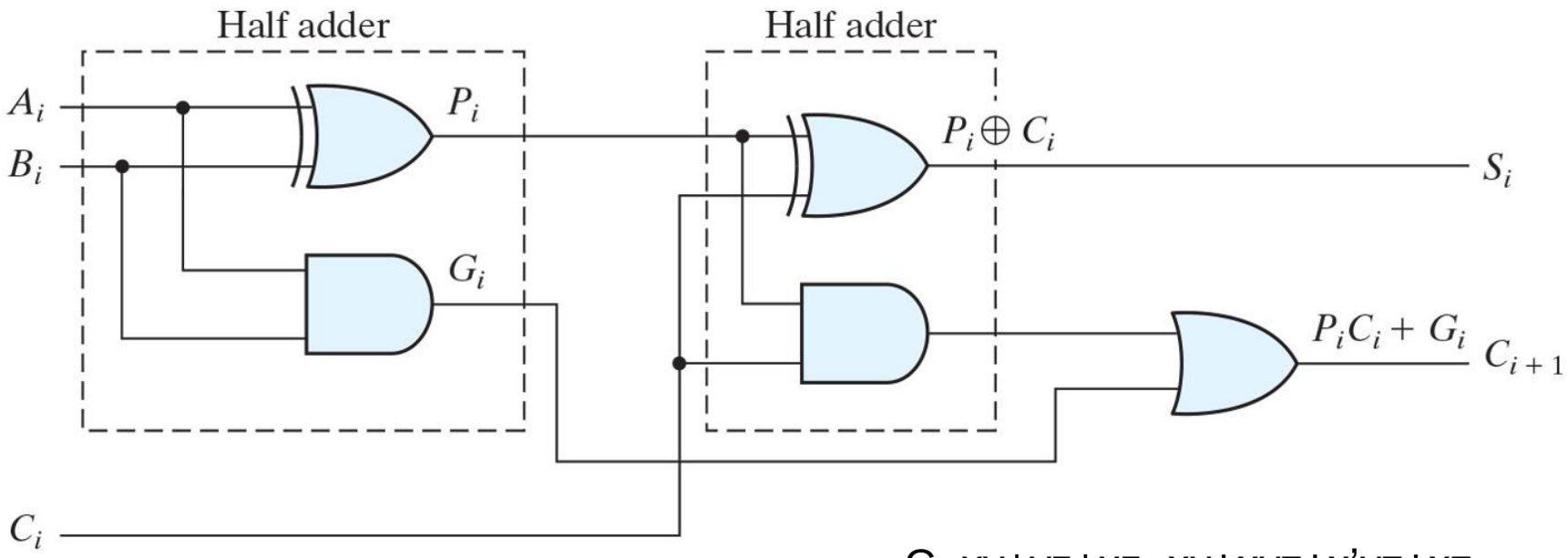
Figure 4.9
Four-bit adder.



Binary Adder

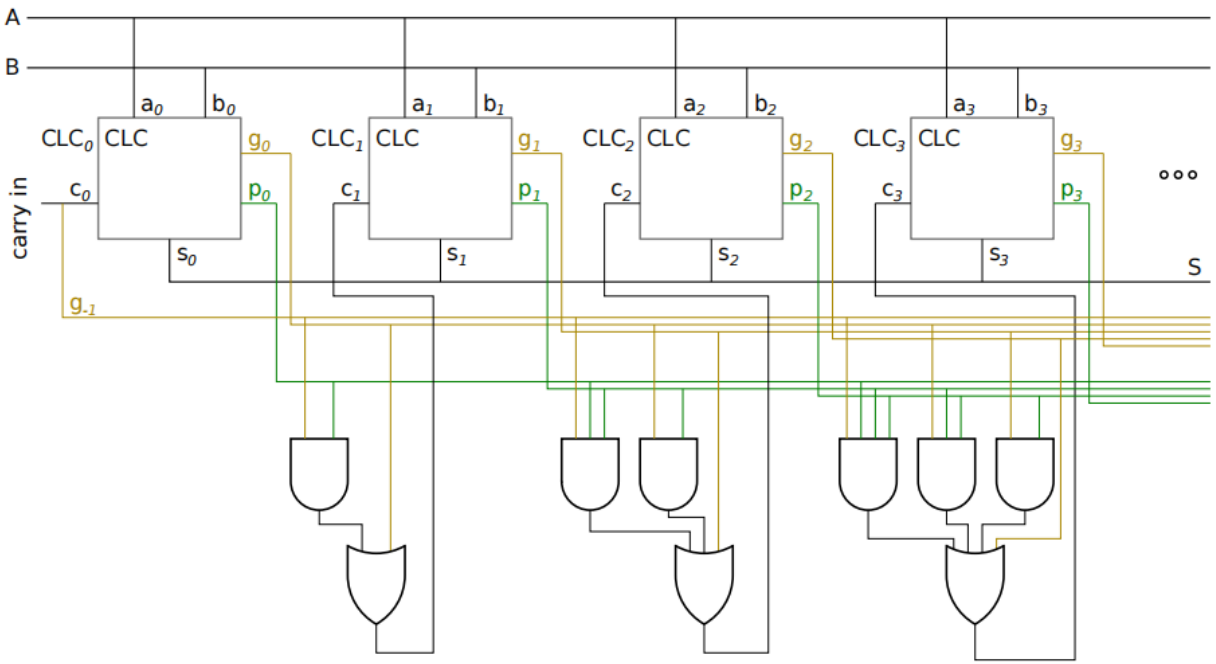
Subscript *i*:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Figure 4.10
Full adder with *P* and *G* shown.



C=xy+yz+xz=xy+xyz+x'yz+xz
= xy+x'yz+xyz+xy'z
= xy+xyz+(x XOR y)z
= xy + (x XOR y)z

Figure 4.12
Four-bit adder with carry lookahead.



Boolean expressions for carry signals c_0 to c_3:

$$c_0 = g_{-1}.$$

$$c_1 = g_{-1}p_0 + g_0.$$

$$c_2 = g_{-1}p_0p_1 + g_0p_1 + g_1.$$

$$c_3 = g_{-1}p_0p_1p_2 + g_0p_1p_2 + g_1p_2 + g_2.$$

Figure 4.11
Logic diagram of carry lookahead generator.

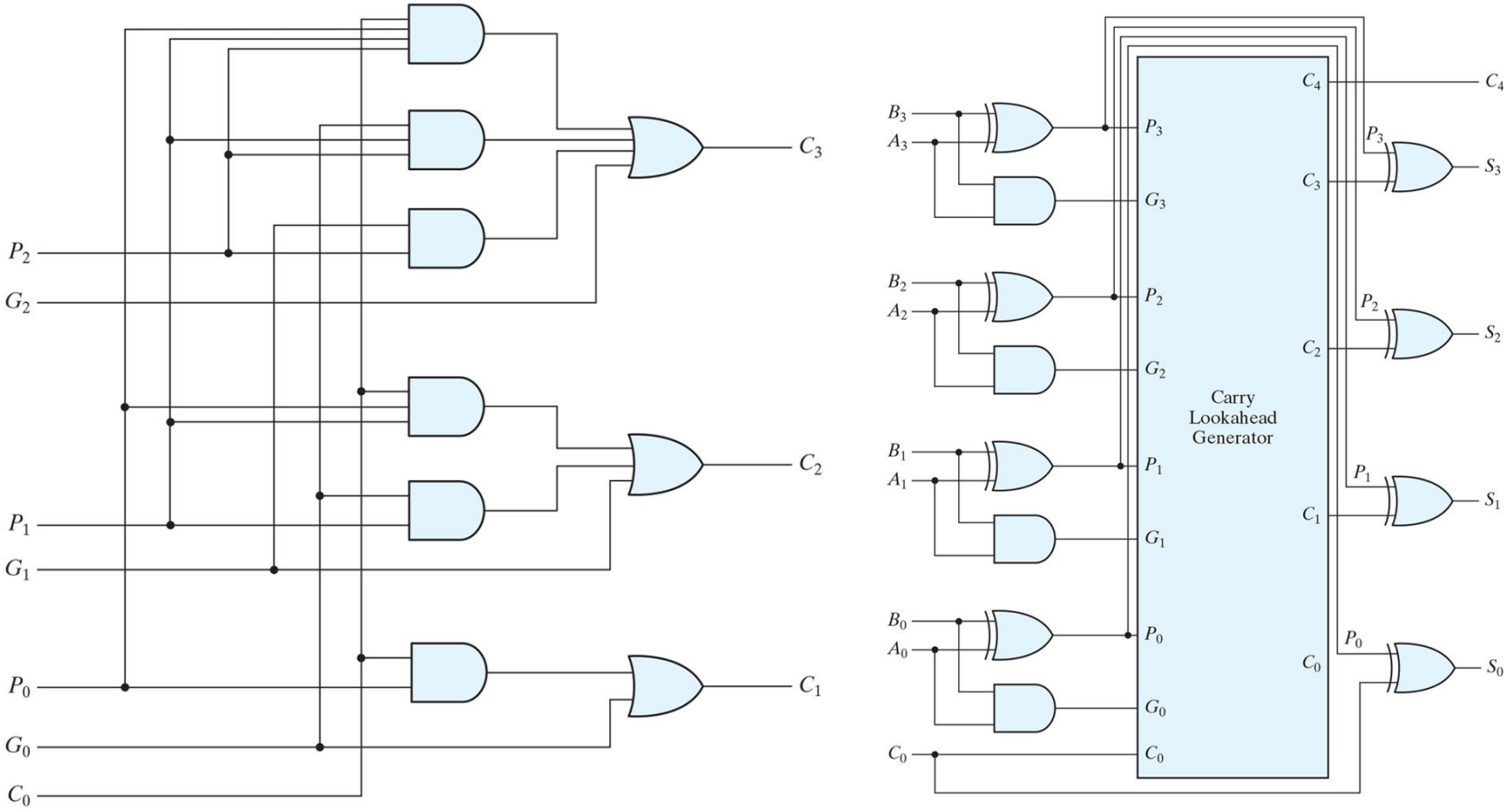


Figure 4.12
Four-bit adder with carry lookahead.

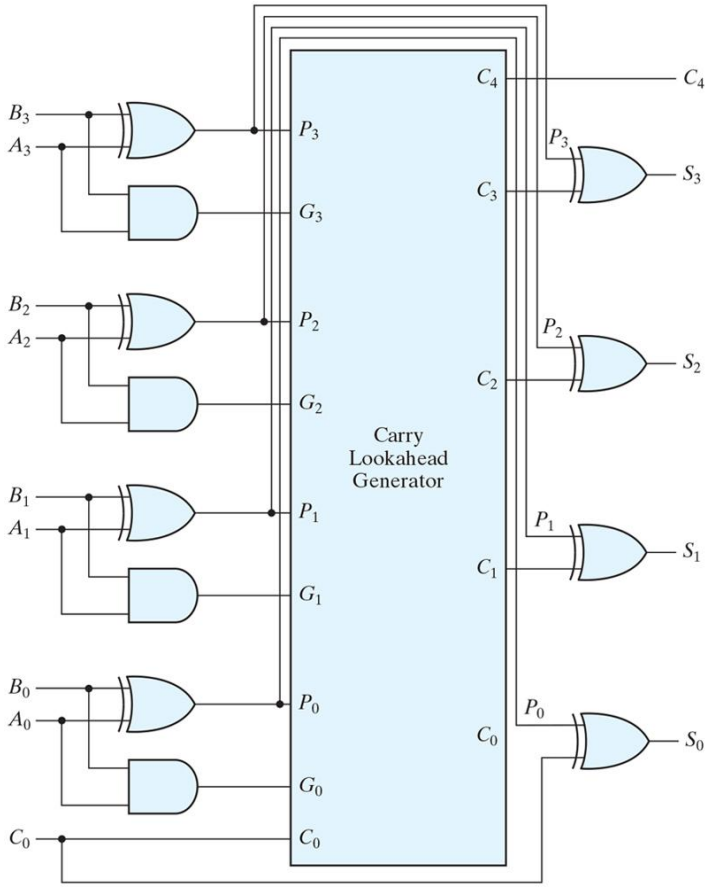


Figure 4.13
Four-bit adder–subtractor (with overflow detection).

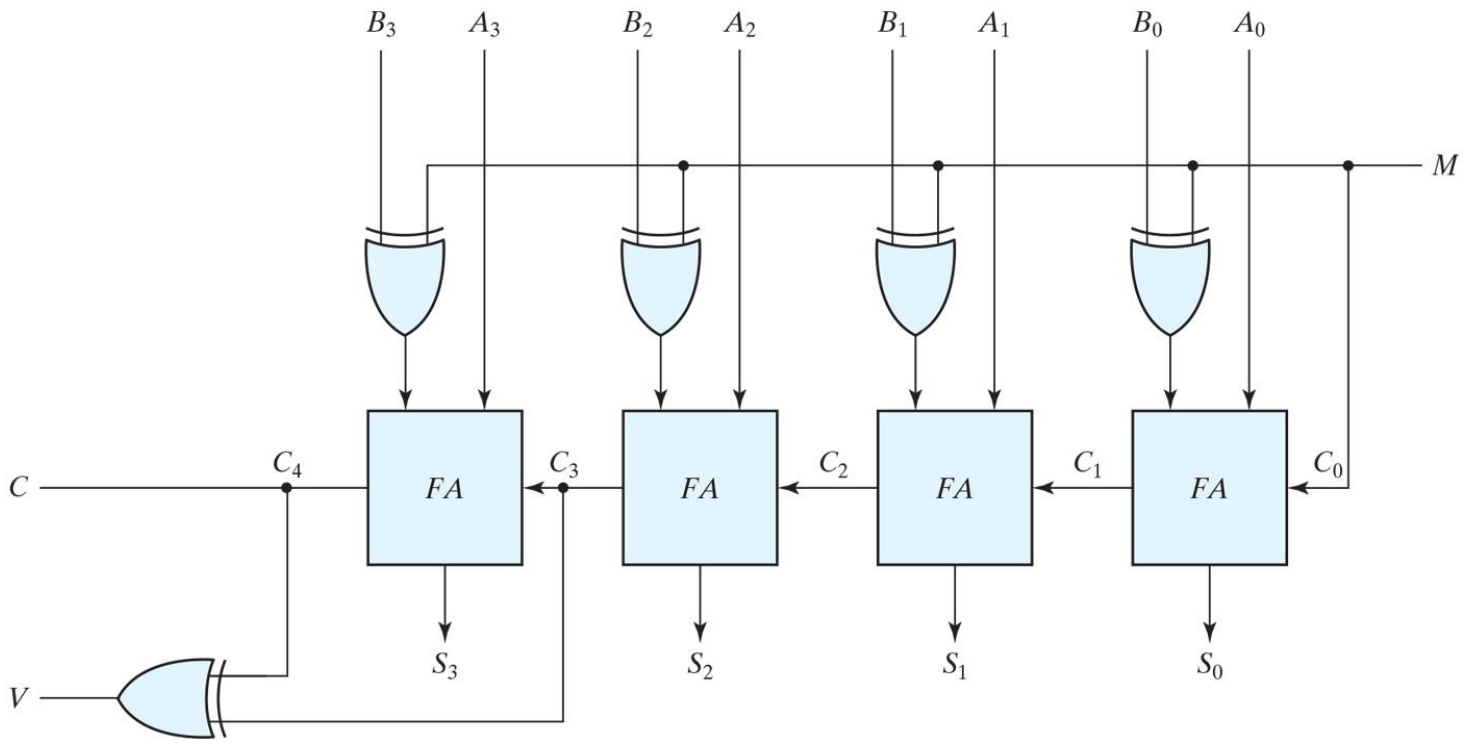


Table 4.5
Derivation of BCD Adder.

	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
+6	0	1	0	1	0	1	0	0	0	0	10
+6	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
+6	1	0	0	1	1	1	1	0	0	1	19

Figure 4.14
Block diagram of a BCD adder.

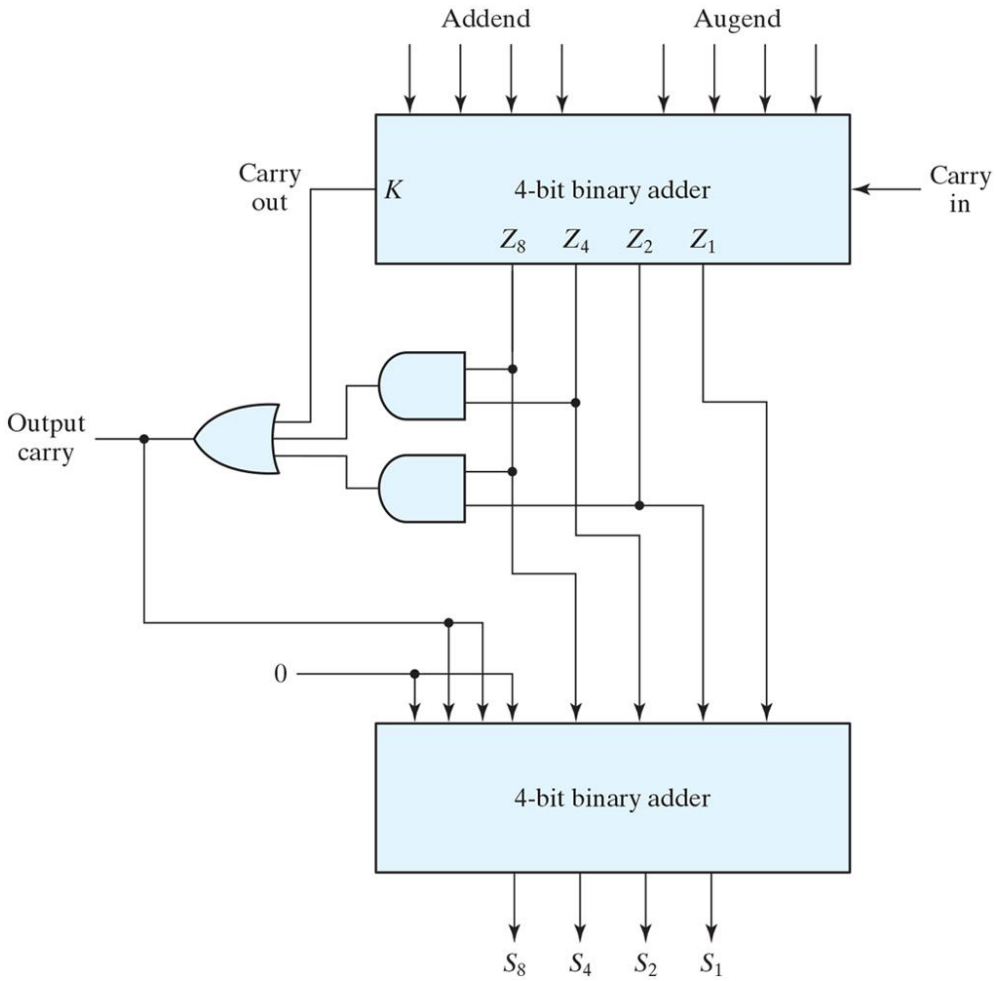


Figure 4.15
Two-bit by two-bit binary multiplier.

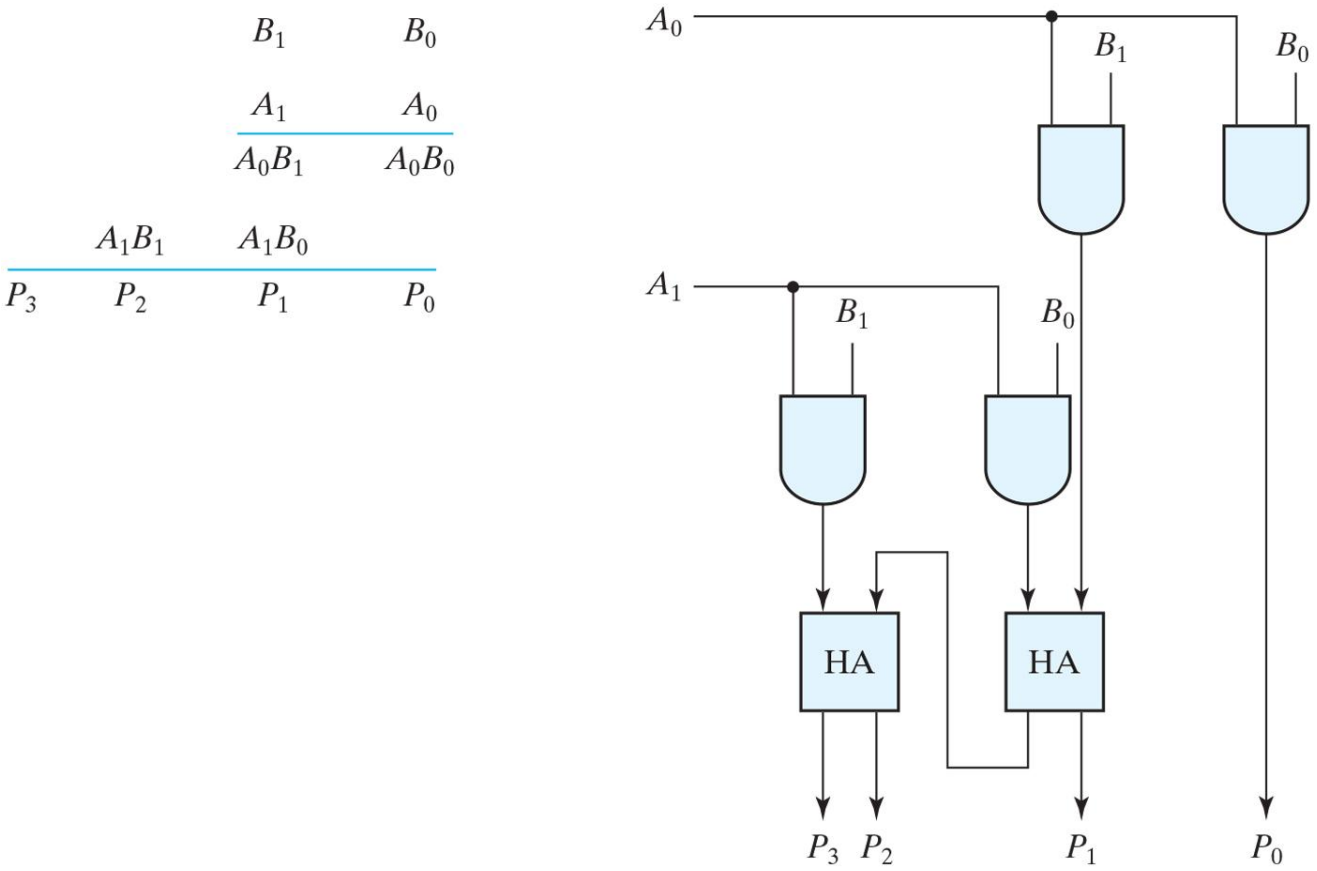


Figure 4.16
Four-bit by three-bit binary multiplier.

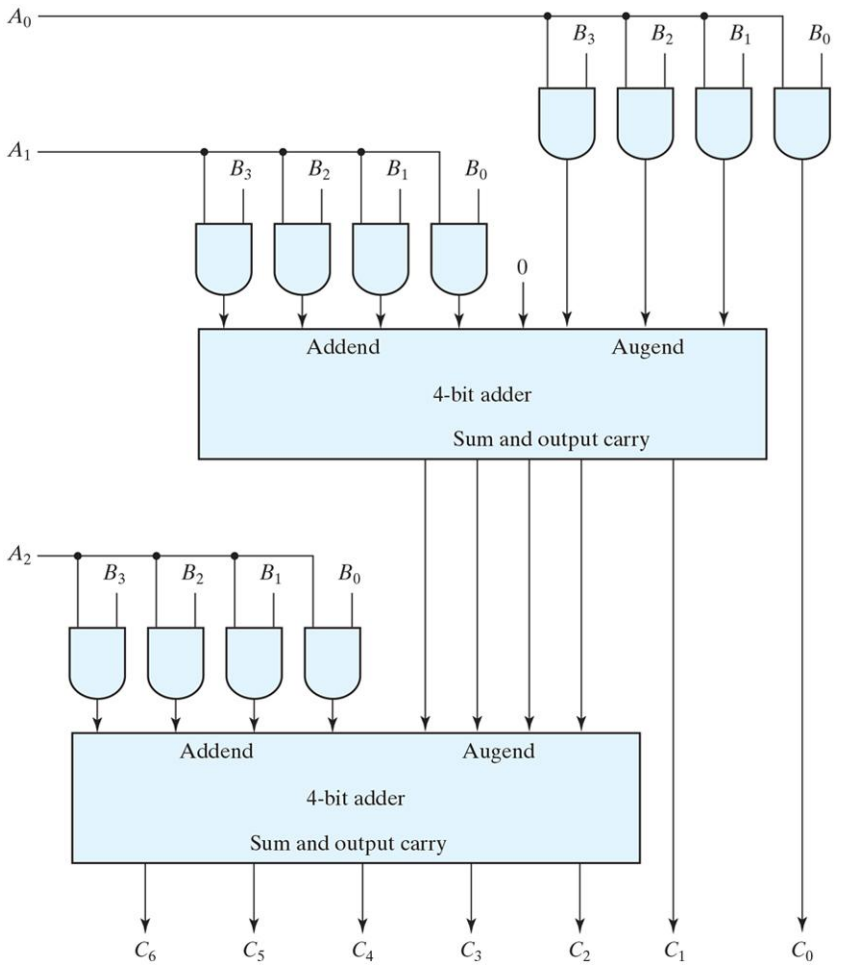


Figure 4.17
Four-bit magnitude comparator.

幅值比较

If the first n bits are equal, then compare the n-1 bit …

A=0, B=1: A<B

A=1, B=0: A>B

A=0, B=1: A<B

A=1, B=0: A>B

A=0, B=1: A<B

A=1, B=0: A>B

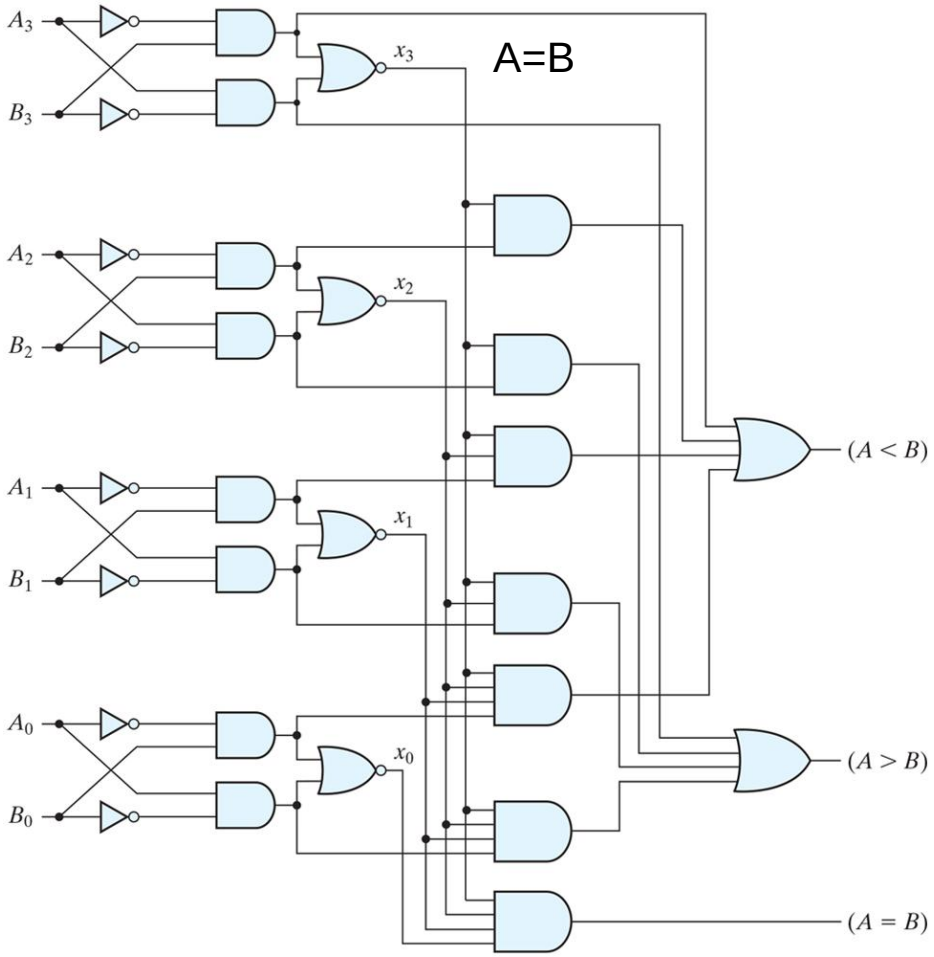


Figure 4.18
Three-to-eight-line decoder.

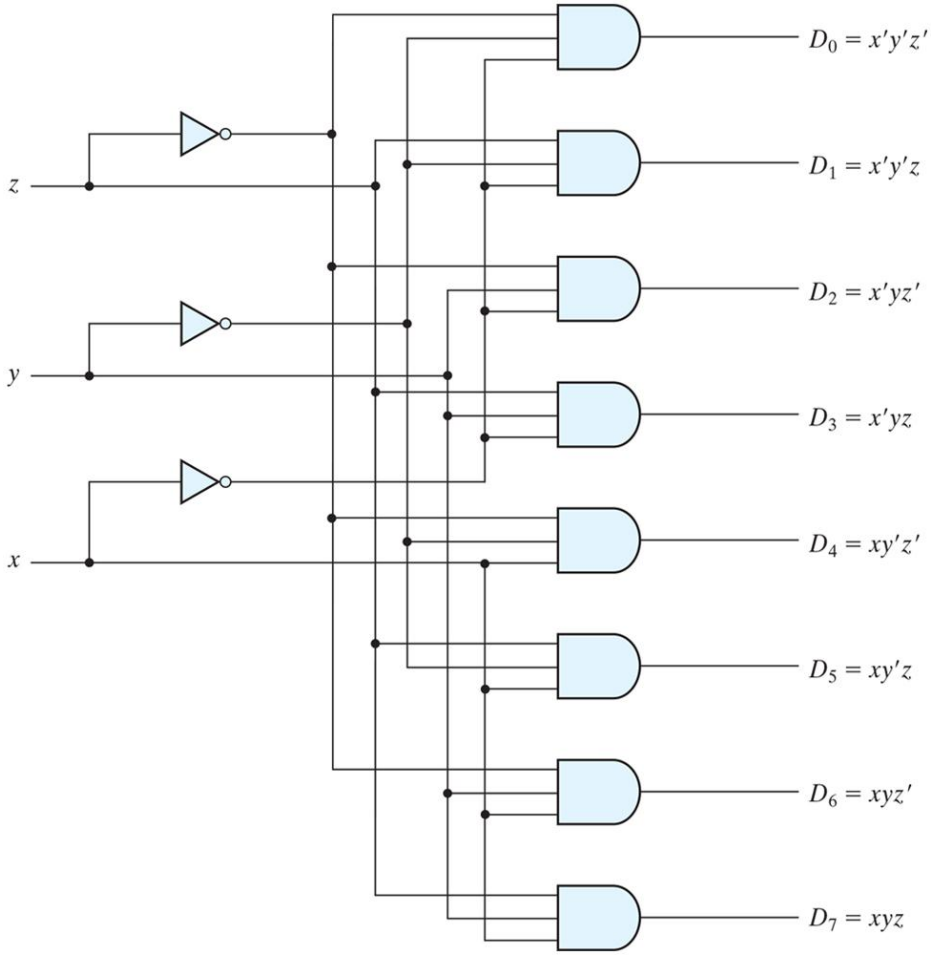


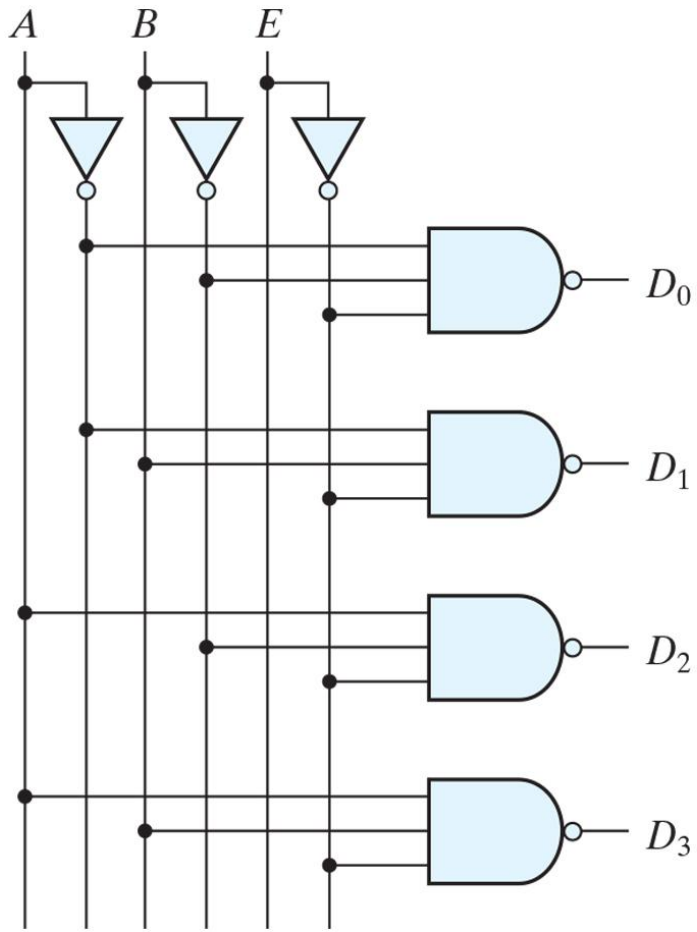
Table 4.6
Truth Table of a Three-to-Eight-Line Decoder.

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

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

A full adder can be built with 3-8 decoder

Figure 4.19
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

Figure 4.20
4 x 16 decoder constructed with two 3 x 8 decoders.

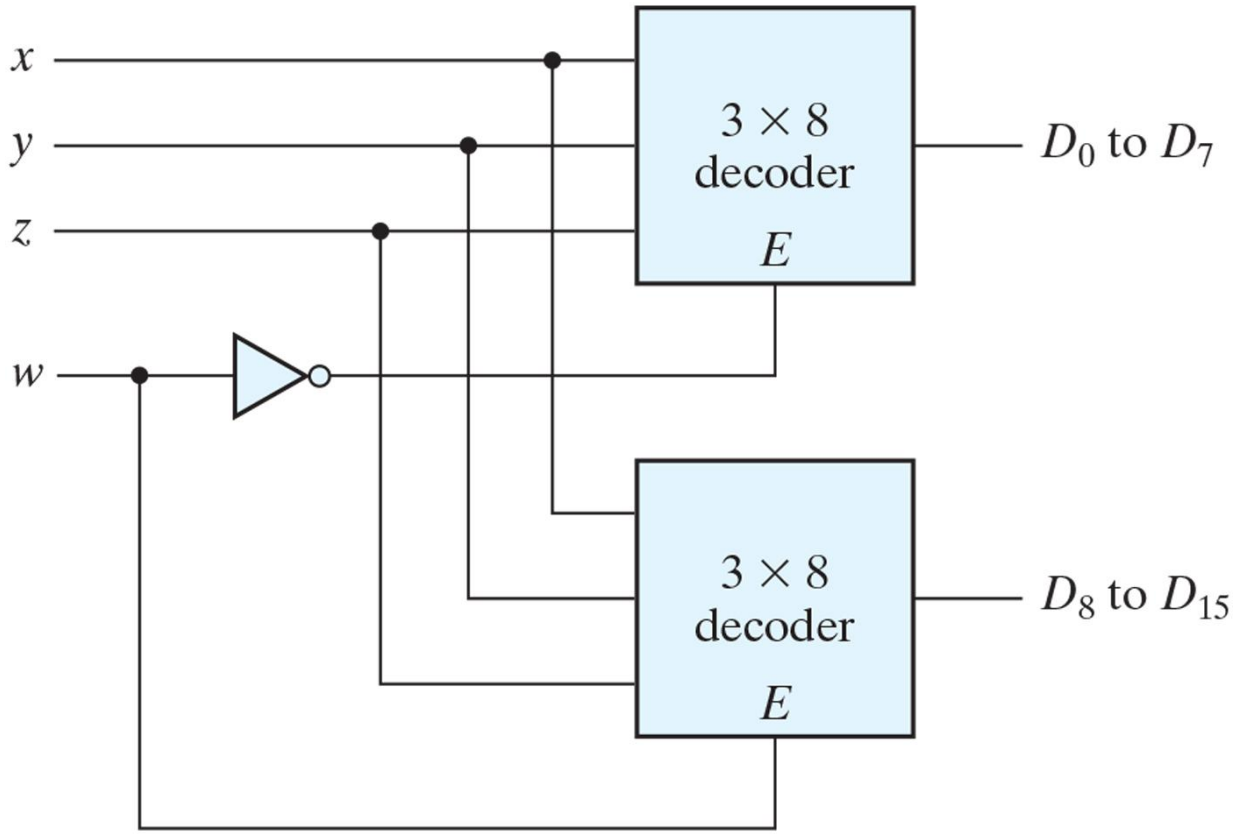
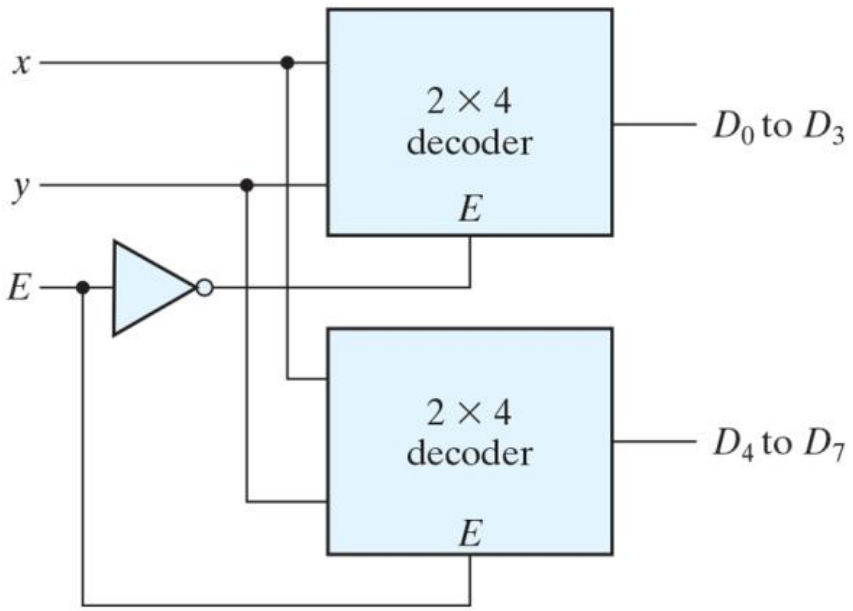
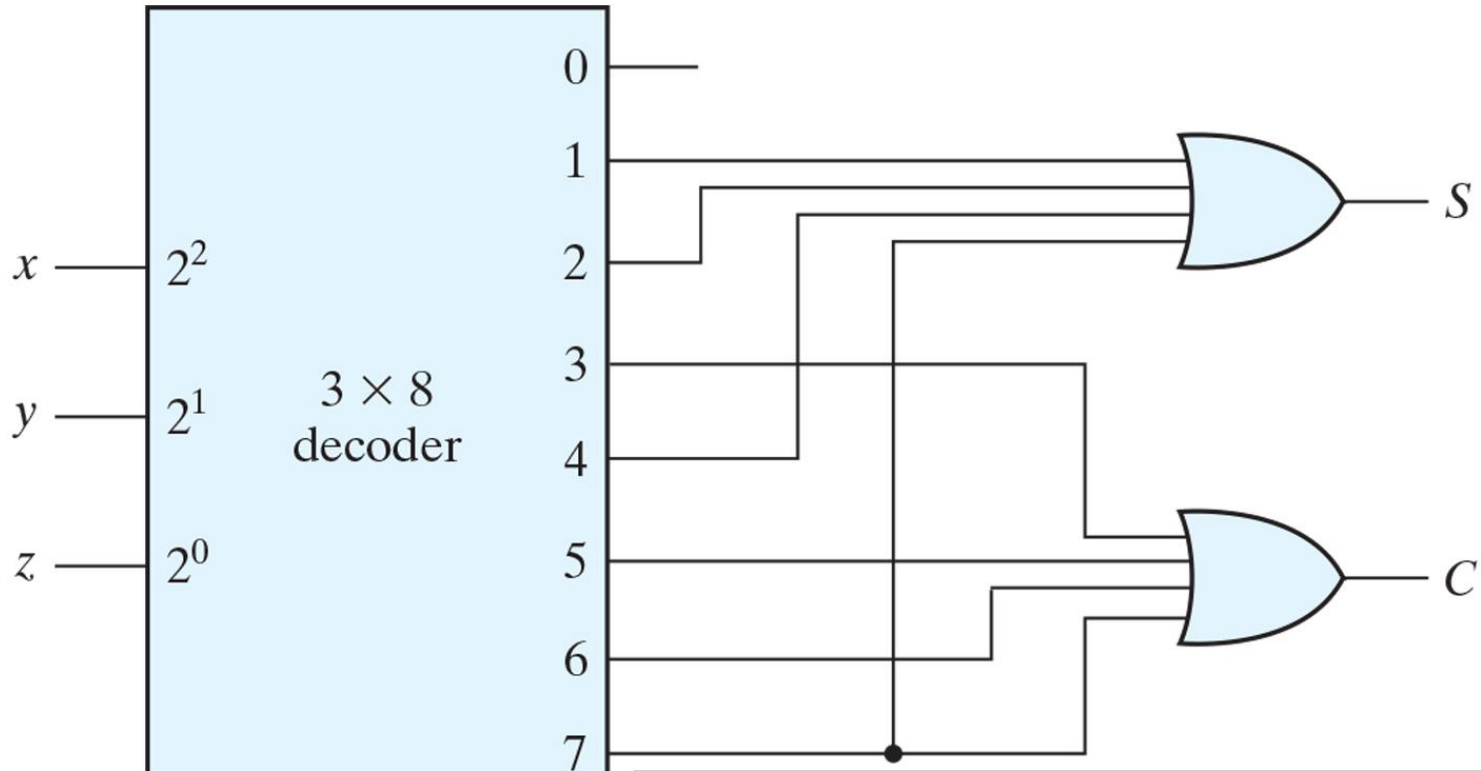


Figure PE4.8



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

Figure 4.21
Implementation of a full adder with a 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

Encoder :Inverse of 3-8 decoder

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

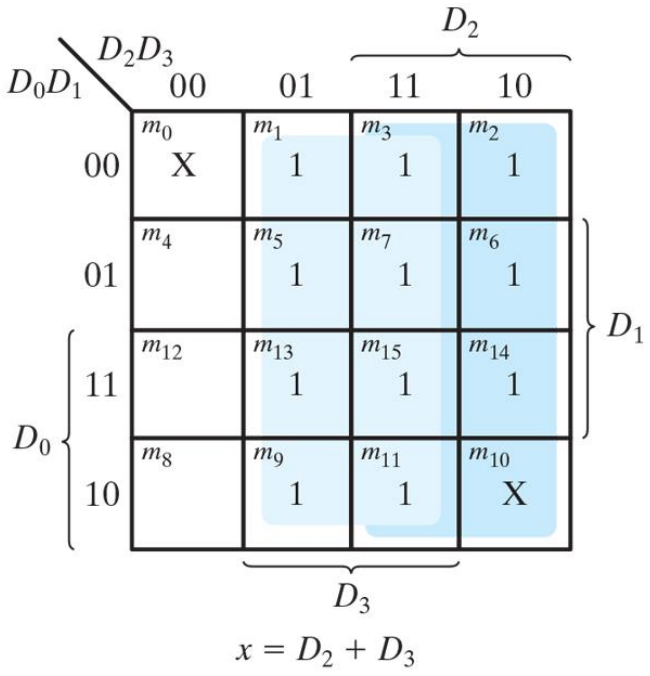
编码器

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

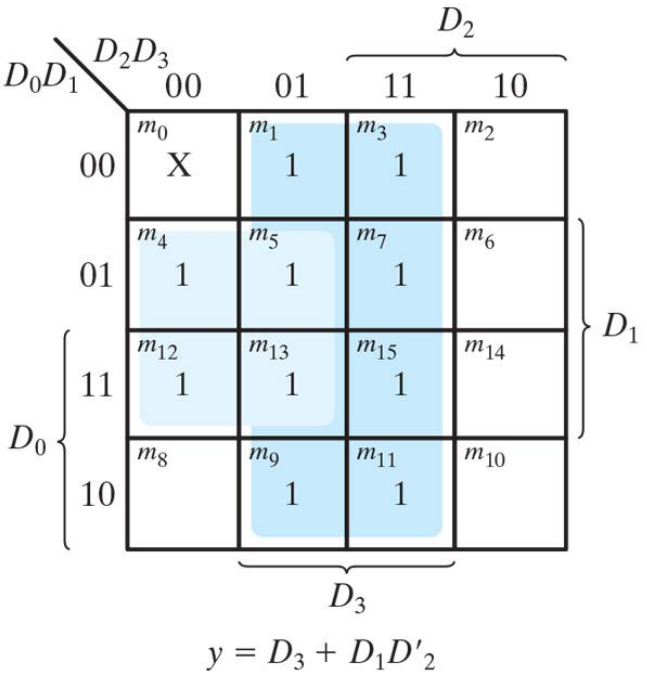
Table 4.8
Truth Table of a Priority Encoder.

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.



$x = D_2 + D_3$



$y = D_3 + D_1D'_2$

Figure 4.23
Four-input priority encoder.

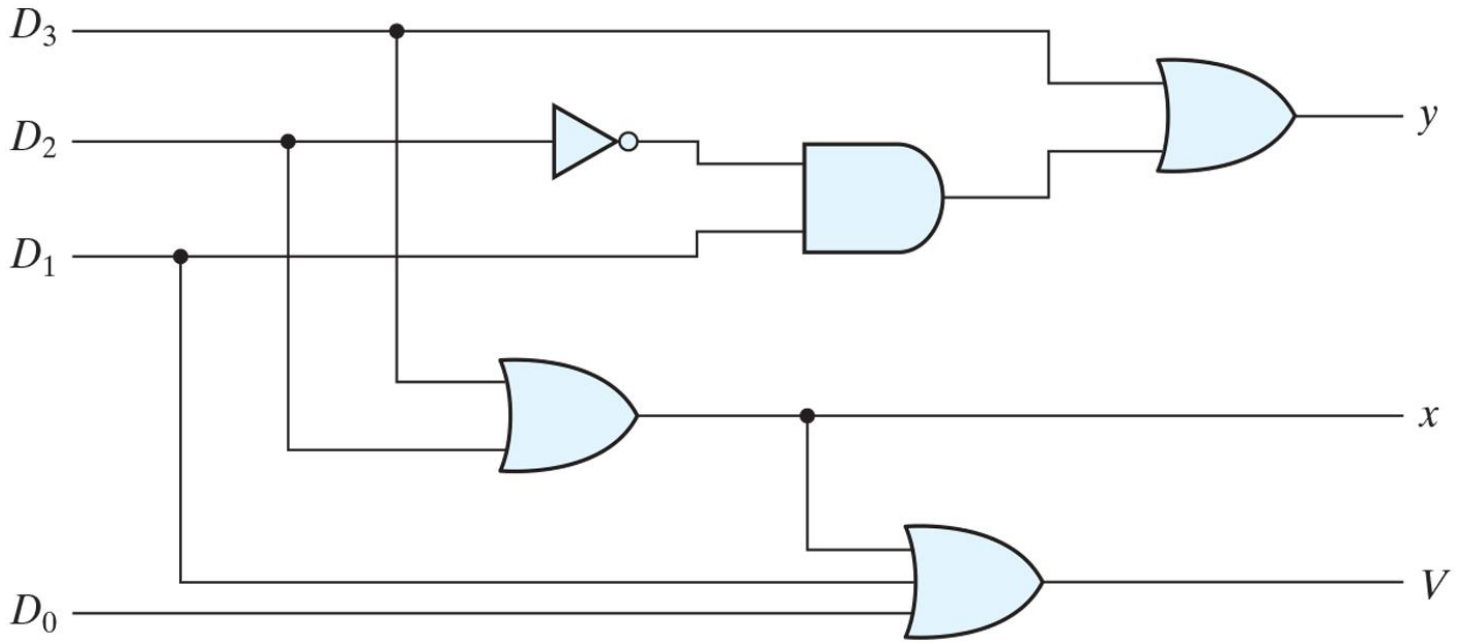
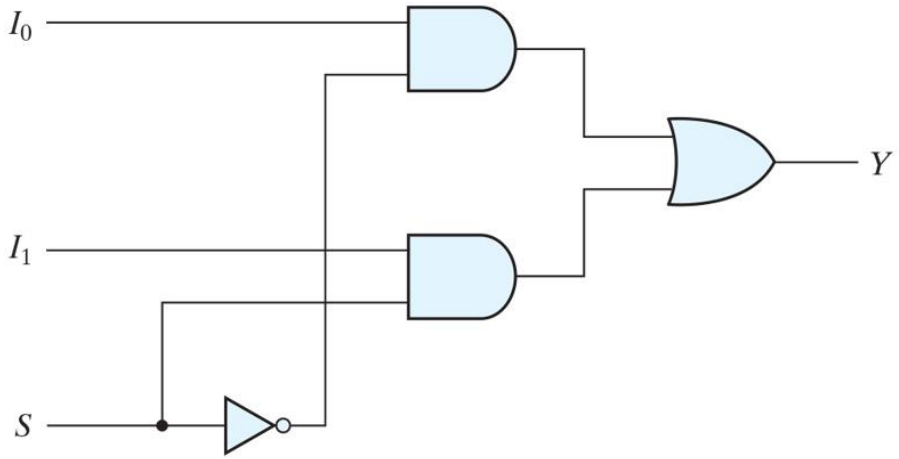
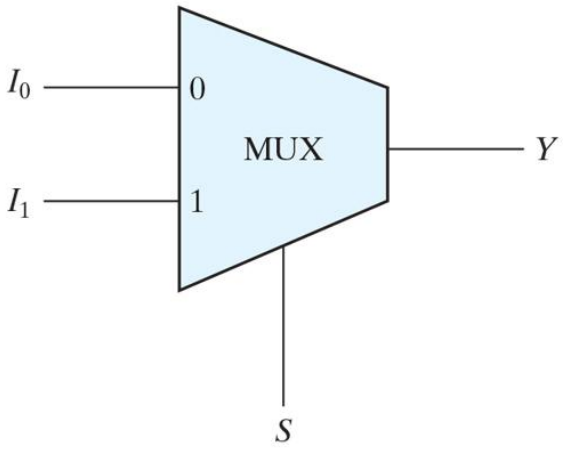


Figure 4.24
Two-to-one-line multiplexer.

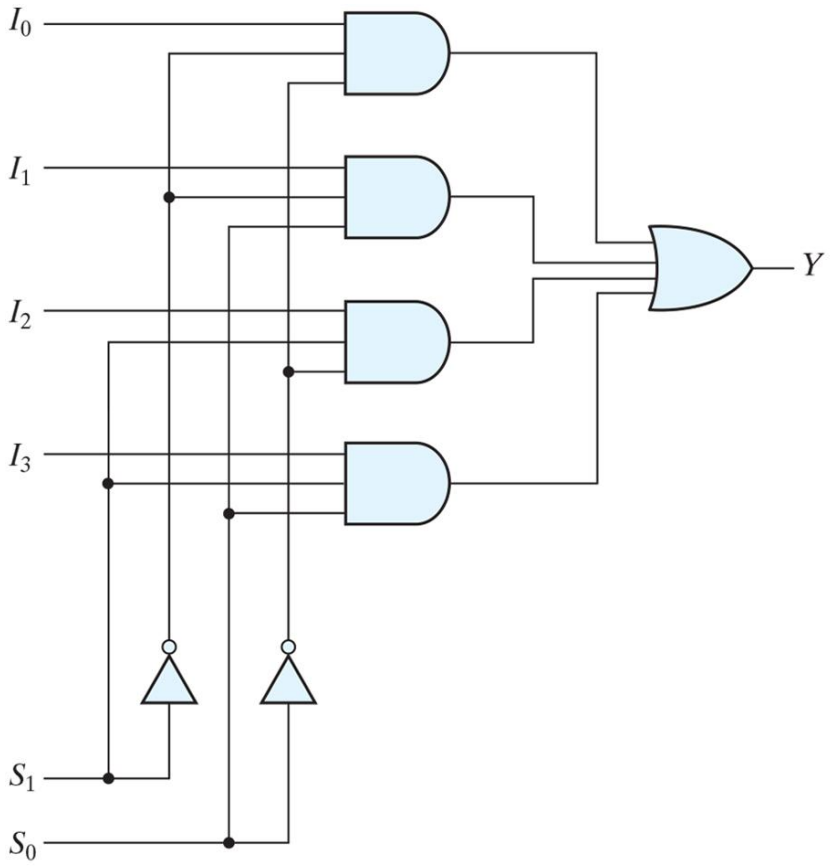


(a) Logic diagram



(b) Block diagram

Figure 4.25
Four-to-one-line multiplexer.



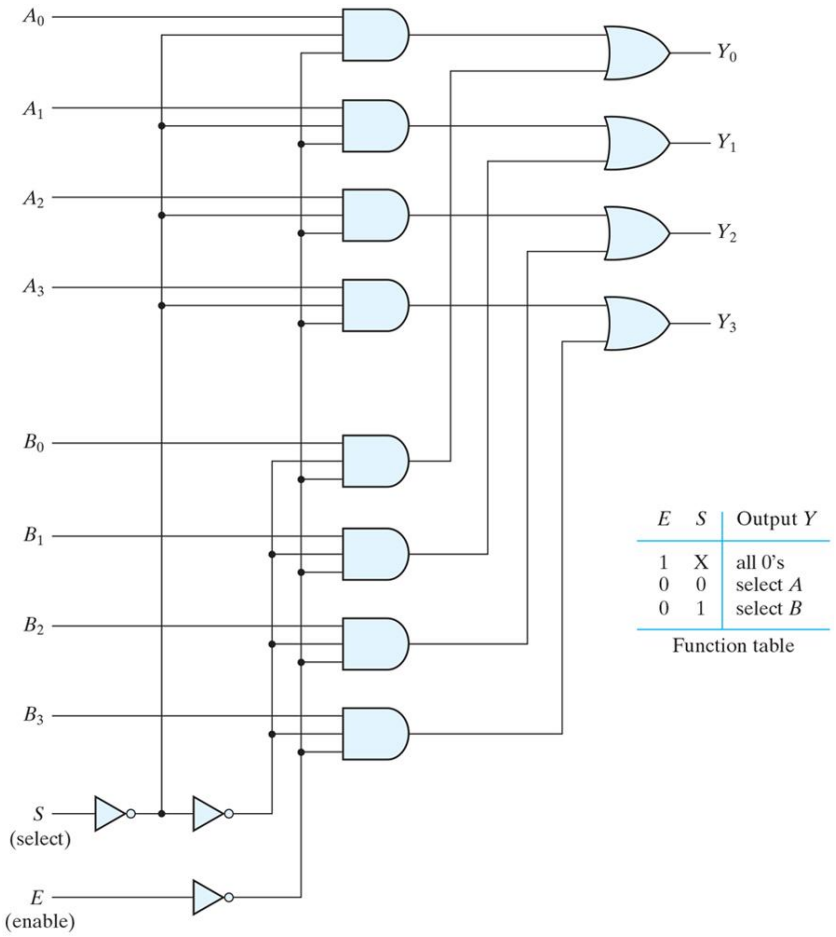
(a) Logic diagram

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

(b) Function table

Figure 4.26
Quadruple two-to-one-line multiplexer.

4X 2-input multiplexer



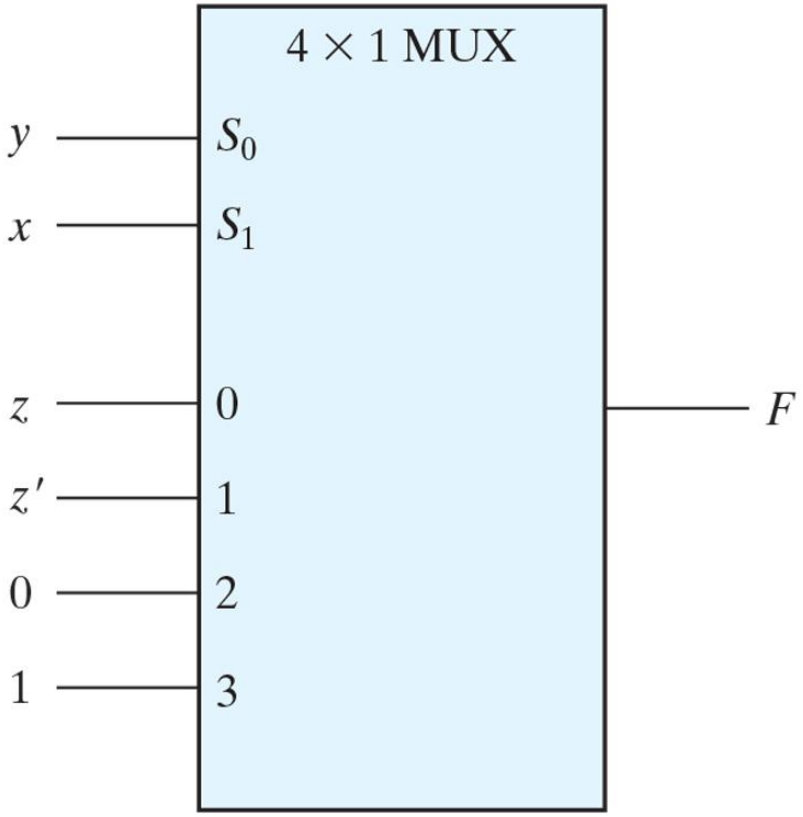
E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

Figure 4.27
Implementing a Boolean function with a multiplexer.

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

Figure 4.28
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	

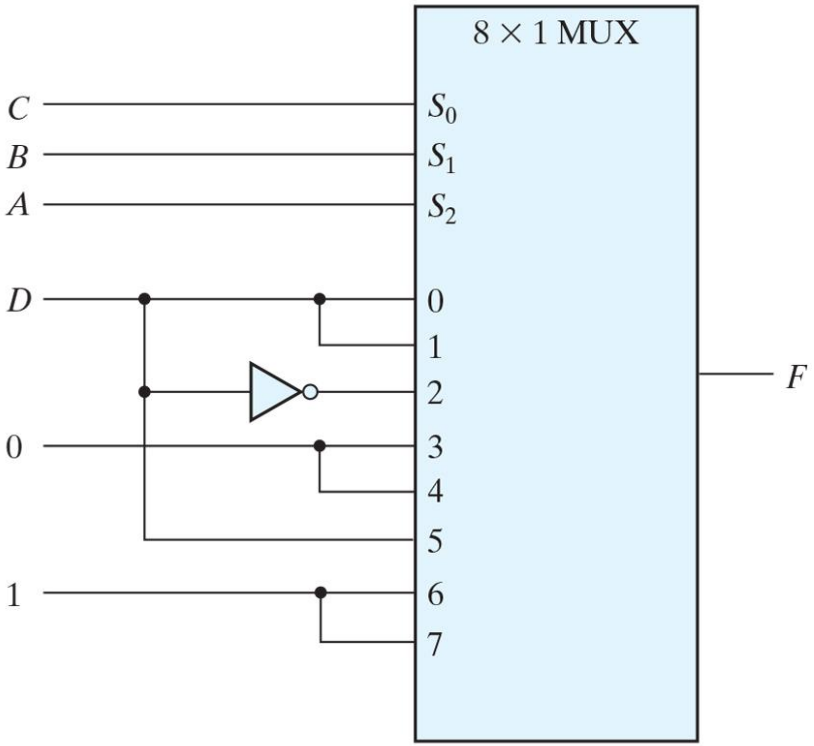


Figure PE4.9

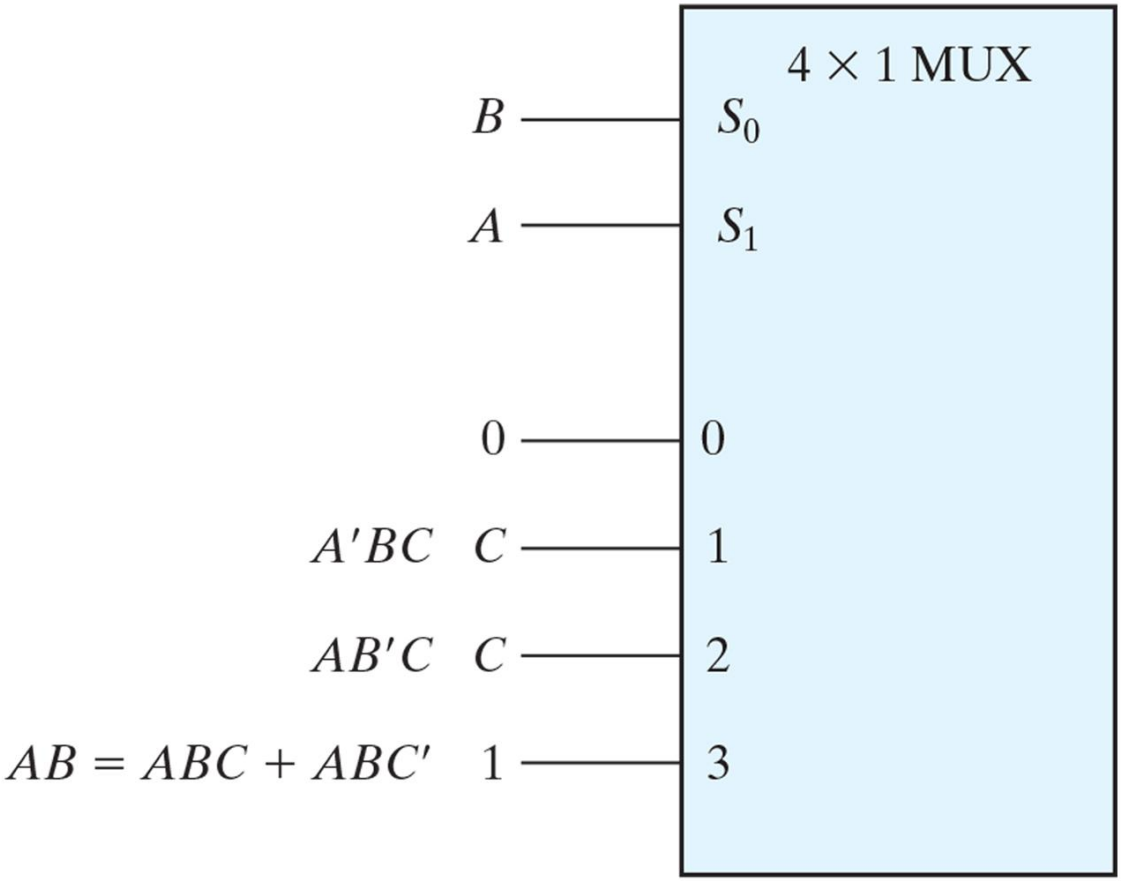


Figure 4.29
Graphic symbol for a three-state buffer.

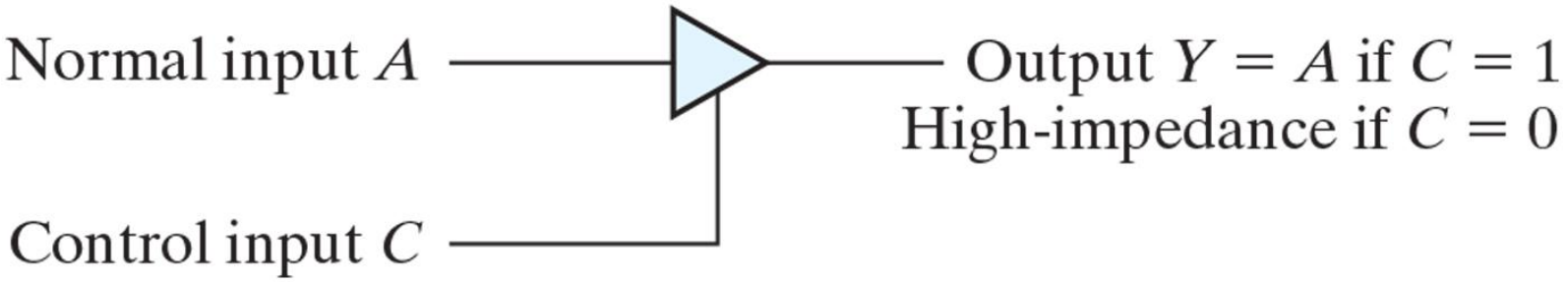


Figure 4.30
Multiplexers with three-state gates.

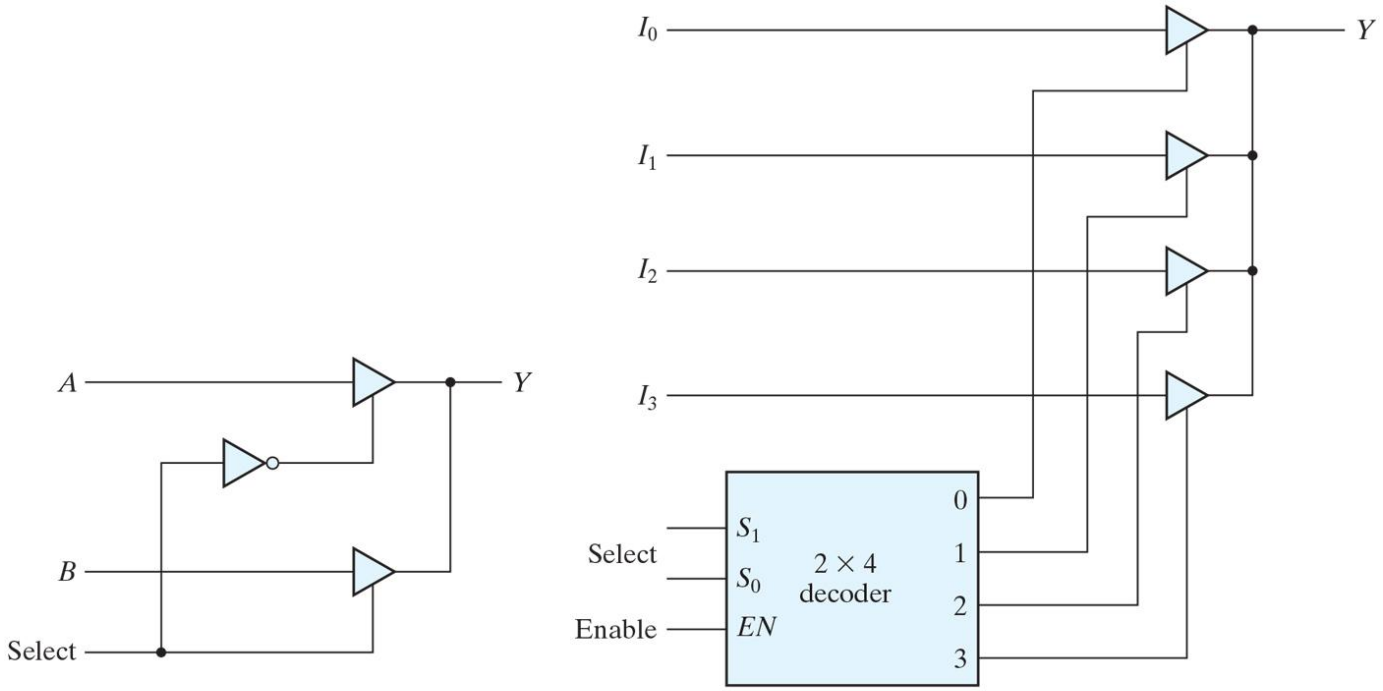


Figure 4.31
Relationship of Verilog constructs to truth tables, Boolean equations, and schematics.

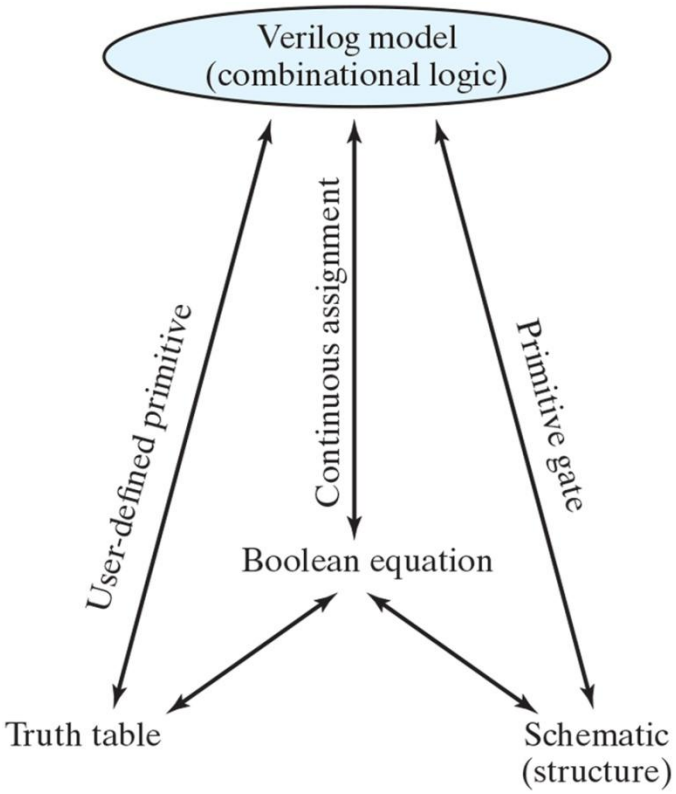


Figure 4.32
Relationship of VHDL constructs to truth tables, Boolean equations, and schematics three-state gates.

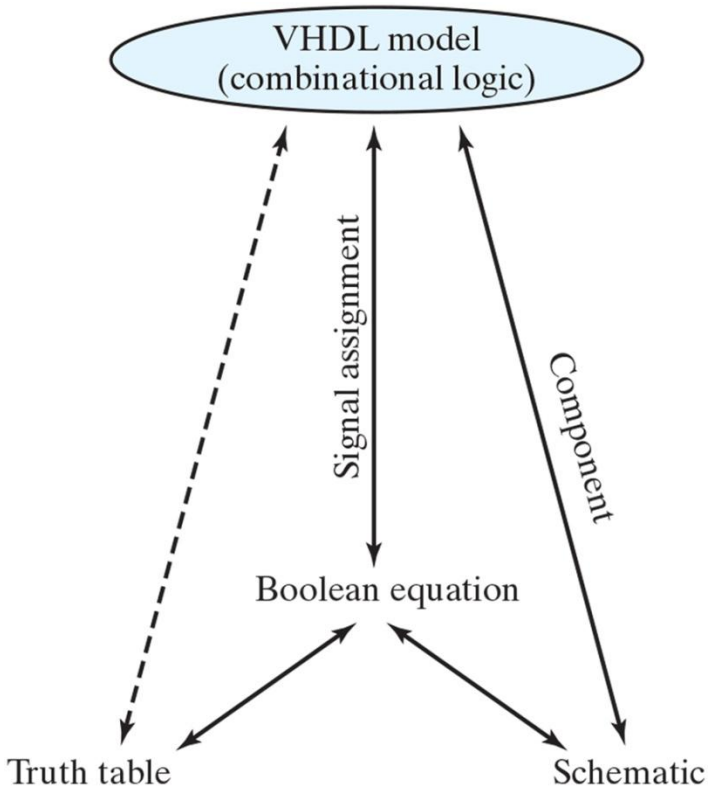


Table 4.9
Truth Table for Predefined Primitive Gates.

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

Figure 4.33
Design hierarchy of an 8-bit ripple-carry adder. For simplicity, some blocks are omitted where they replicate what is already shown.

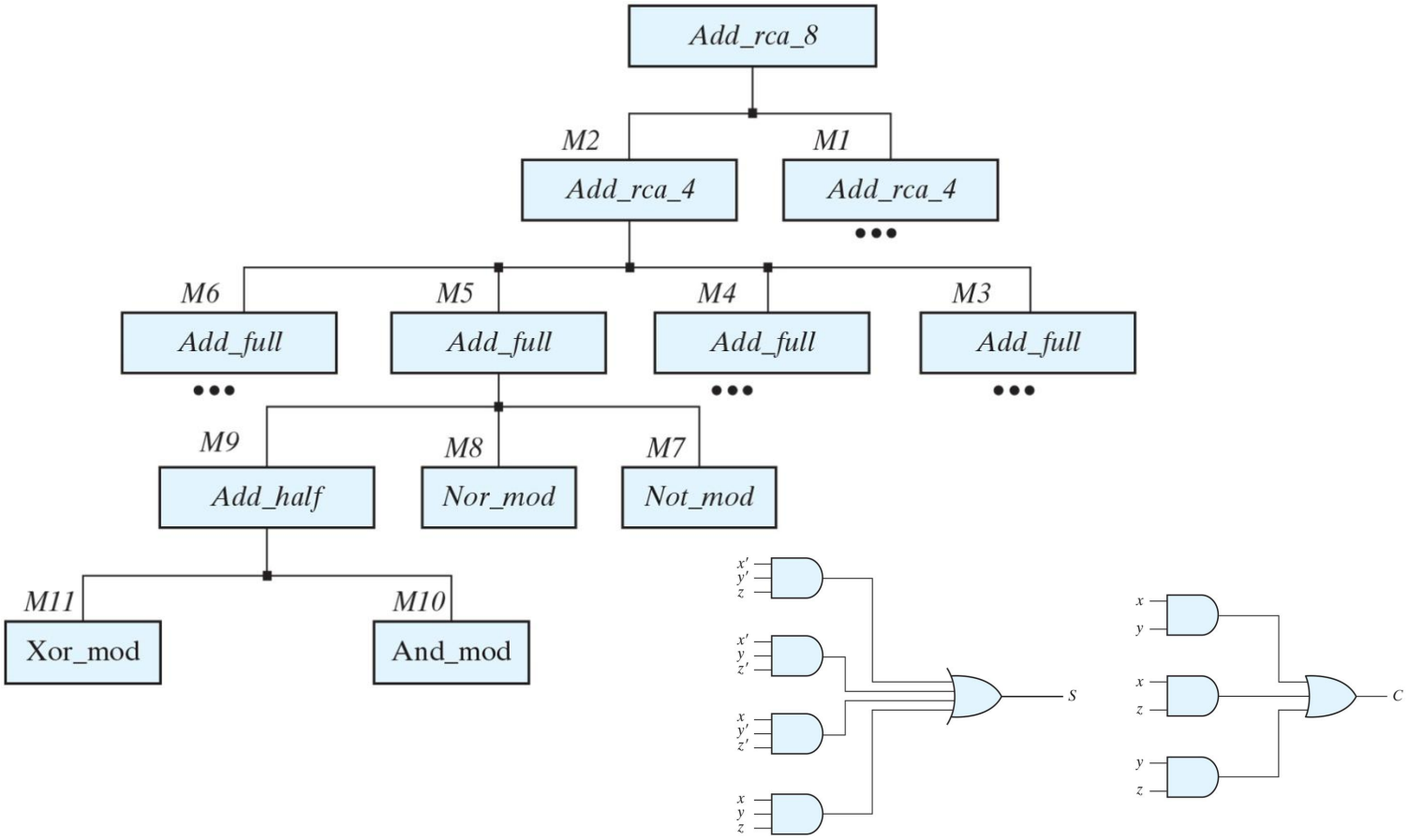


Figure 4.34
Decomposition of an 8-bit ripple carry adder into a chain of two 4-bit adders;[9] each 4-bit adder consists of a chain of four full adders. The full adders are composed of half adders and one OR gate; the half adders are composed of logic gates.

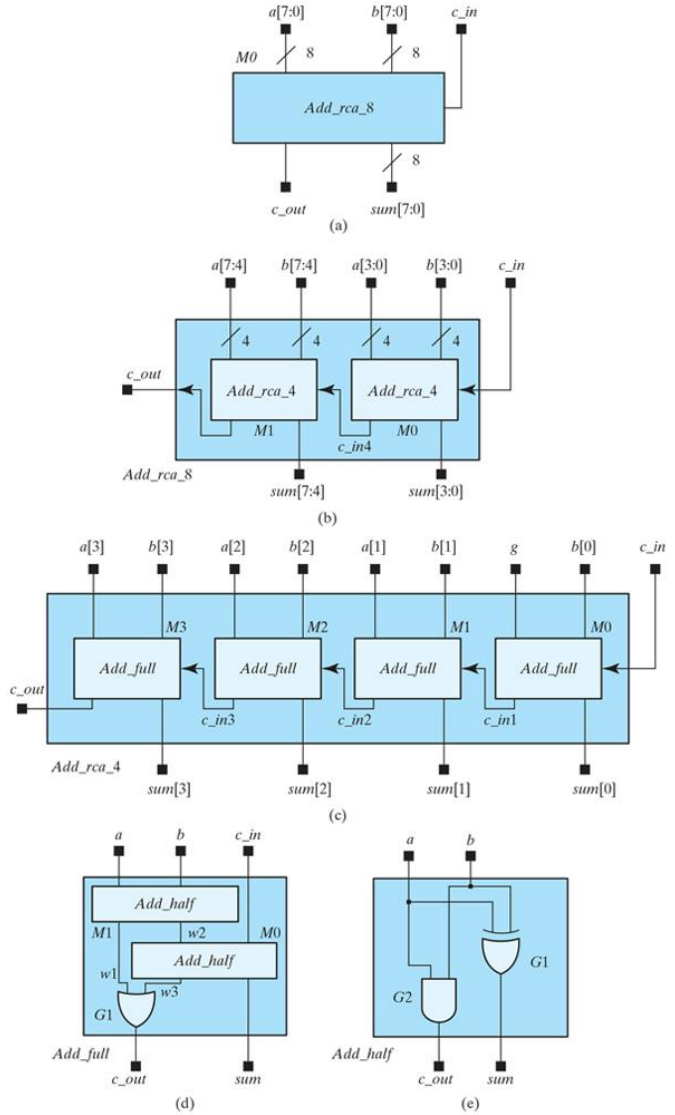


Figure 4.35
Three-state gates.

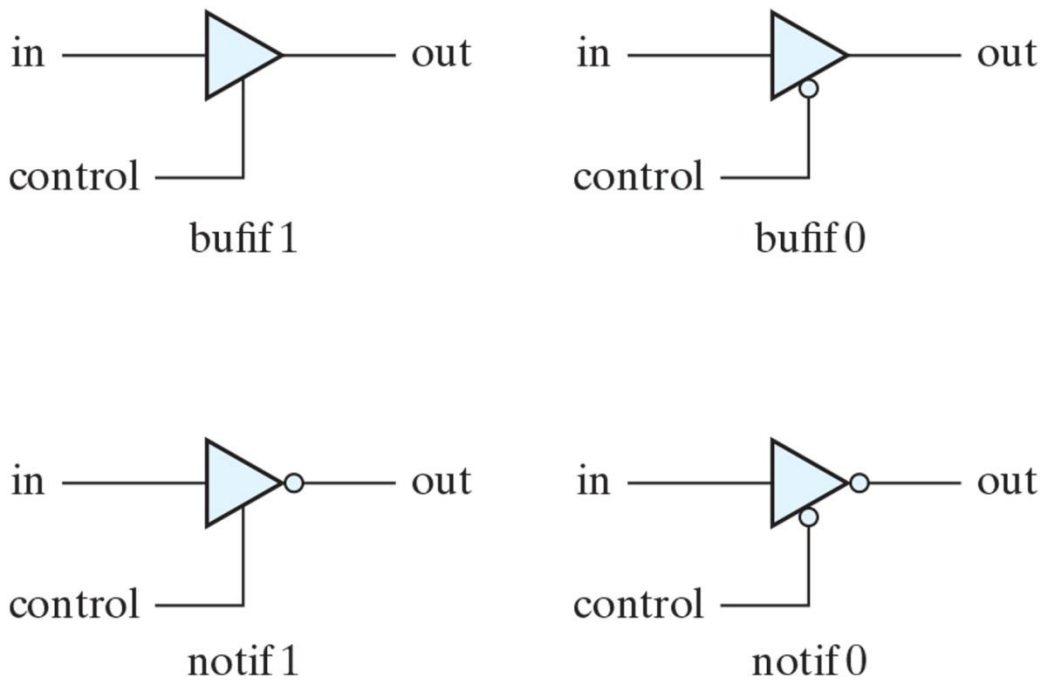


Figure 4.36
Two-to-one-line multiplexer with three-state buffers.

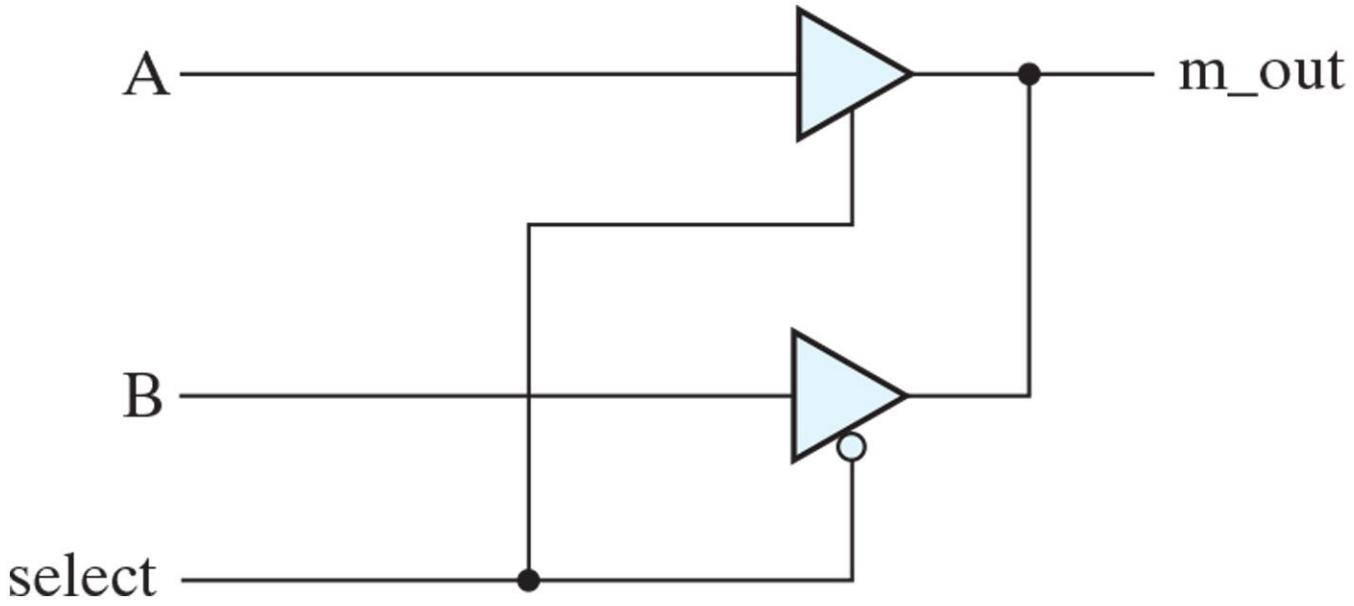


Table 4.10
Some Verilog Operators.

Symbol	Operation	Symbol	Operation
+	binary addition		
−	binary subtraction		
&	bitwise AND	&&	logical AND
|	bitwise OR	||	logical OR
^	bitwise XOR		
~	bitwise NOT	!	logical NOT
==	equality		
>	greater than		
<	less than		
{ }	concatenation		
? :	conditional		

ignore

Table 4.11a
SystemVerilog Assignment Operators[15].

Operator	Description
+=	Add RHS to LHS and assign
−=	Subtract RHS from LHS and assign
*=	Multiply LHS by RHS and assign
/=	Divide LHS by RHS and assign
%=	Divide LHS by RHS and assign remainder
&=	Bitwise AND RHS with LHS and assign
|=	Bitwise OR RHS with LHS and assign
^=	Bitwise exclusive OR RHS with LHS and assign
<<=	Bitwise left-shift the LHS by the number of times indicated by the RHS and assign
>>=	Bitwise right-shift the LHS by the number of times indicated by the RHS and assign
<<<=	Arithmetic-shift the LHS by the number of times indicated by the RHS and assign
>>>=	Arithmetic-shift the LHS by the number of times indicated by the RHS and assign

Table 4.11b
SystemVerilog Increment/Decrement Operators.

Usage	Operation	Description
j = i++;	*Postincrement*	*j gets i, then i is incremented by 1*
j = i––;	*Postdecrement*	*j gets i, then i is decremented by 1*
j = ++i;	*Preincrement*	*i is incremented by 1, then j gets i*
j = ––i;	*Predecrement*	*i is decremented by 1, then j gets i*

Table 4.12
Predefined VHDL Data Types.

VHDL Data Type	Value
bit	'0' or '1'
boolean	FALSE or TRUE
integer	$-(2^{31} - 1) \leq$ INTEGER VALUE $\leq (2^{31} - 1)$
positive	$1 \leq$ INTEGER VALUE $\leq (2^{31} - 1)$
natural	$0 \leq$ INTEGER VALUE $\leq (2^{31} - 1)$
real	$-1.0e38 \leq$ FLOATNG POINT VALUE $\leq \cdot 1.0E38$
character	Alphabetical characters (a . . . z, A . . . Z), digits (0, . . . 9), special characters (e.g., %) each enclosed in single quotes
time	integer with units fs, ps, ns, us, ms, sec, min, or hr

Table 4.13
VHDL Operators.

Operator Type	Symbol	Operand(s)	Result	Precedence
Binary Logical	**and or nand nor xor xnor**	Bit, boolean, boolean_vector, bit_vector,	**Same as operands**	
Relational	= / = < < = > > =	Two expression matched in type and size	**FALSE, TRUE**	
Shift Operators	**sll srl sla sra rol ror**	bit_vector	bit_vector	
Addition Operators	+ −	Integer Real number	Integer Real number	Precedence
Concatenation Operator	**&**	Vectors	Vectors	
Unary Sign Operator	+ −			
Multiplication Operators	**·/mod rem**			
Miscellaneous Operators	**not** **abs** **	Numerical Numerical Integer, Floating Point	Exponentiated by integer	Highest Precedence

Figure 4.37
Interaction between testbench and Verilog design unit.

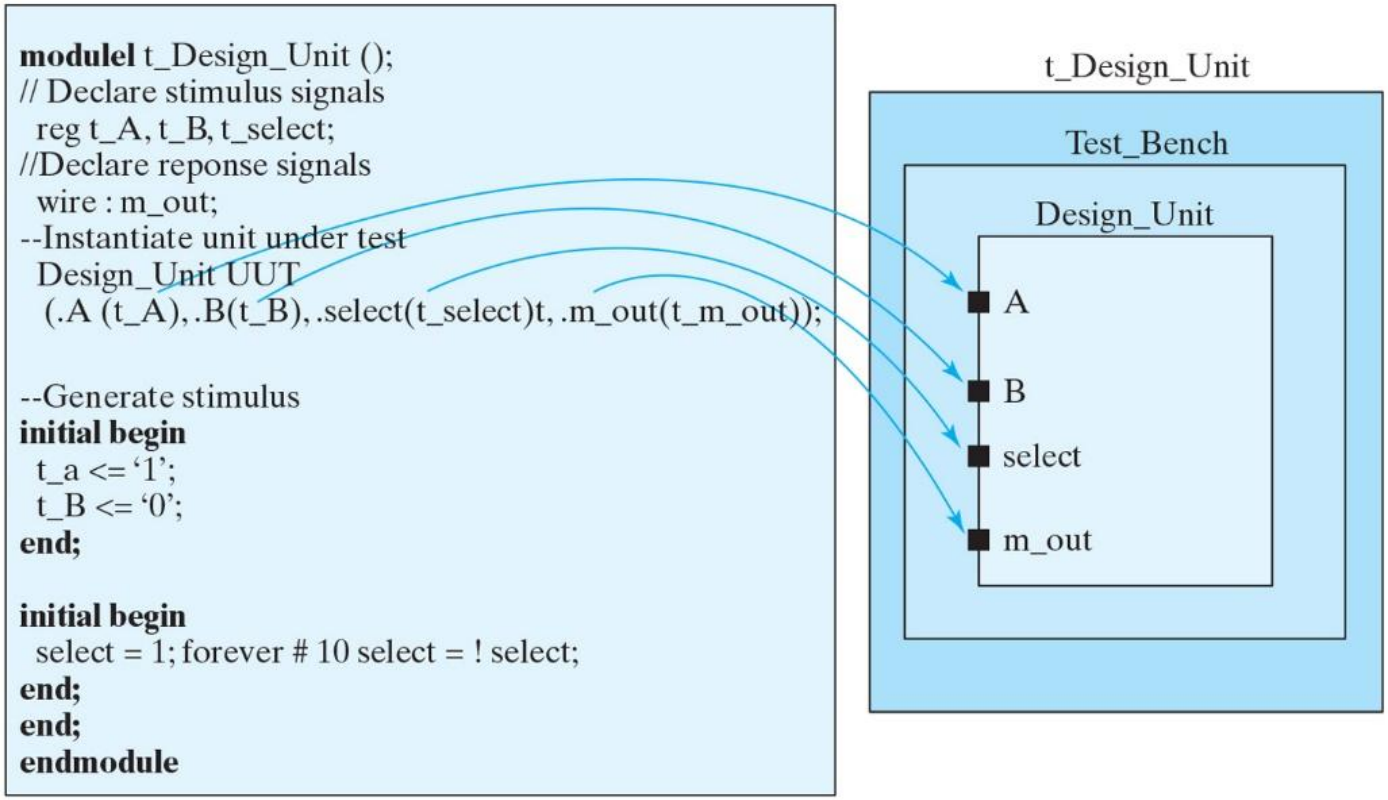


Figure 4.38
Interaction between testbench and VHDL design unit.

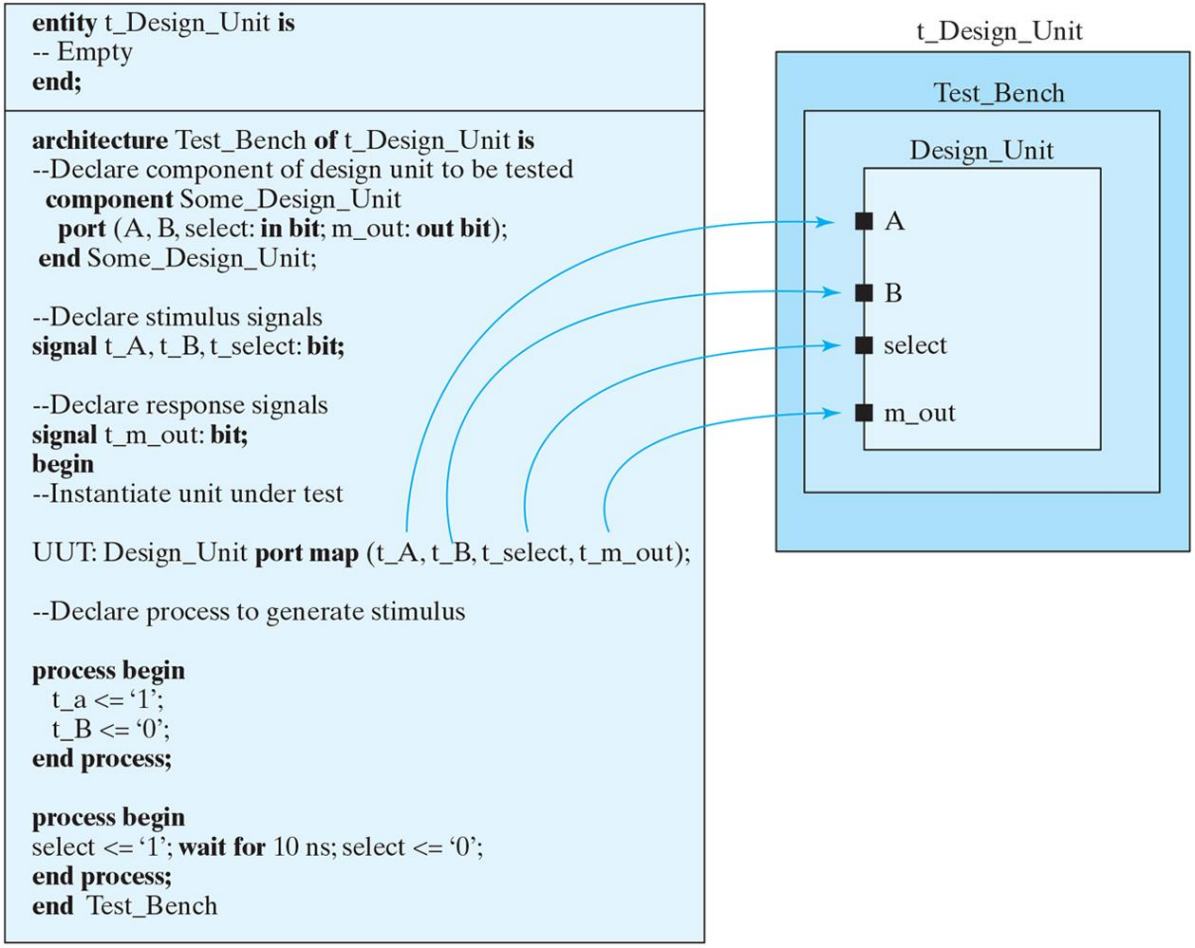


Figure 4.39
Circuit for event-driven simulation.

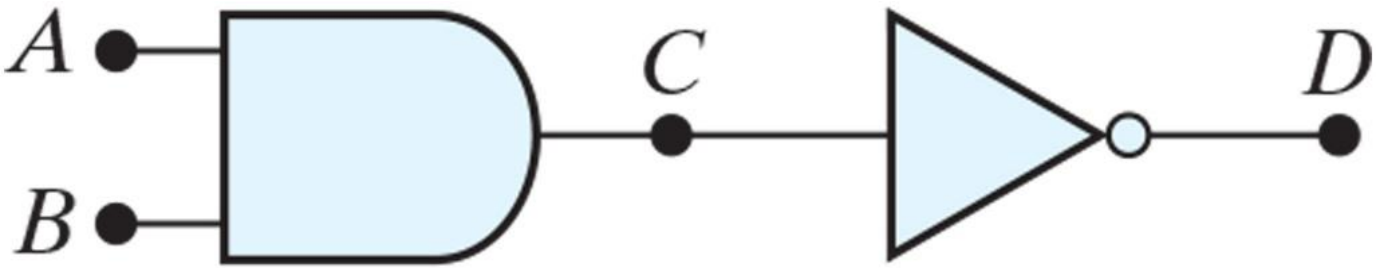


Figure 4.40
Representation of event-driven simulation (with zero delay).

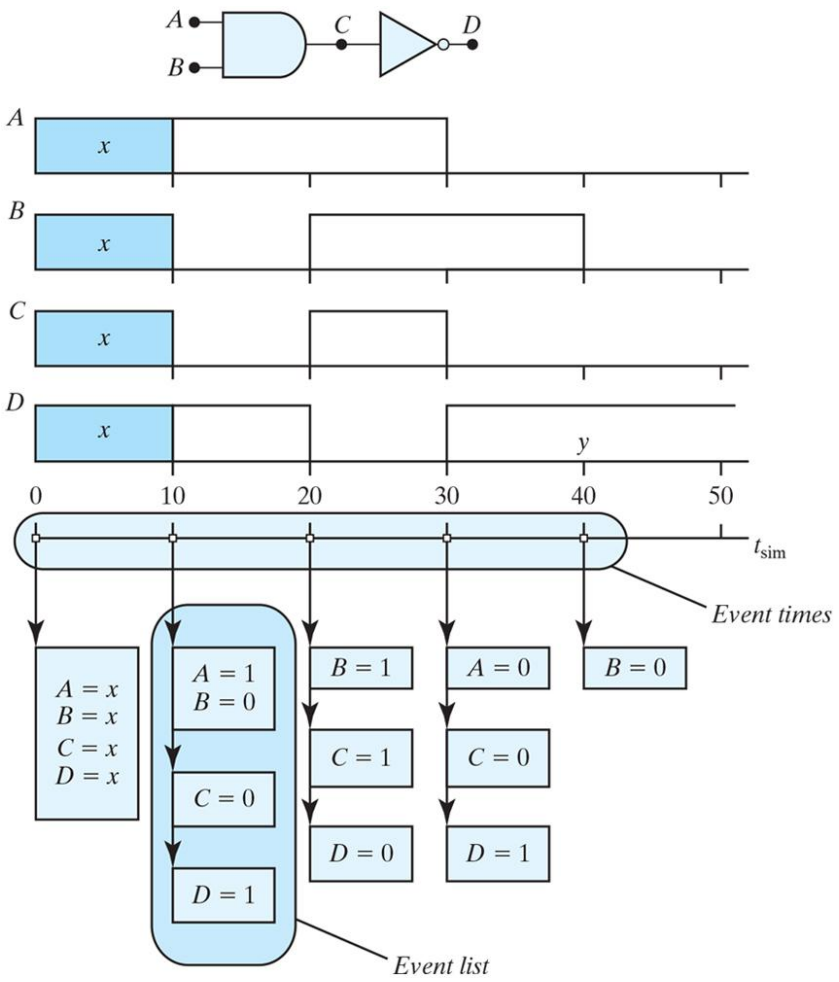


Figure 4.41
Representation of event-driven simulation (with propagation delay).

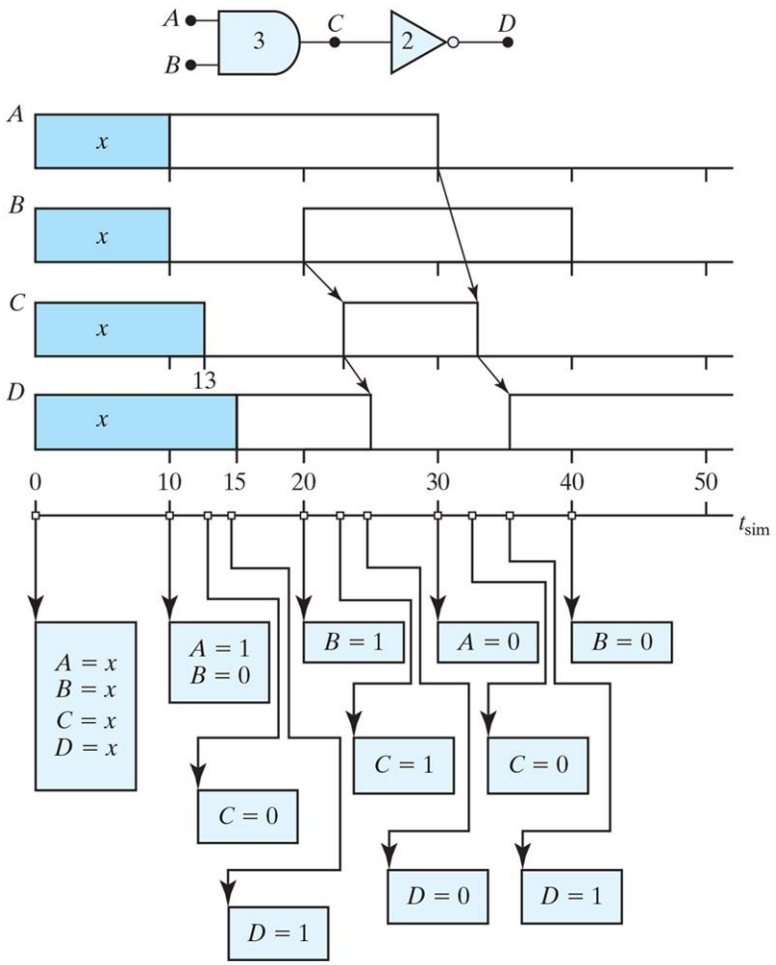


FIGURE P4.1

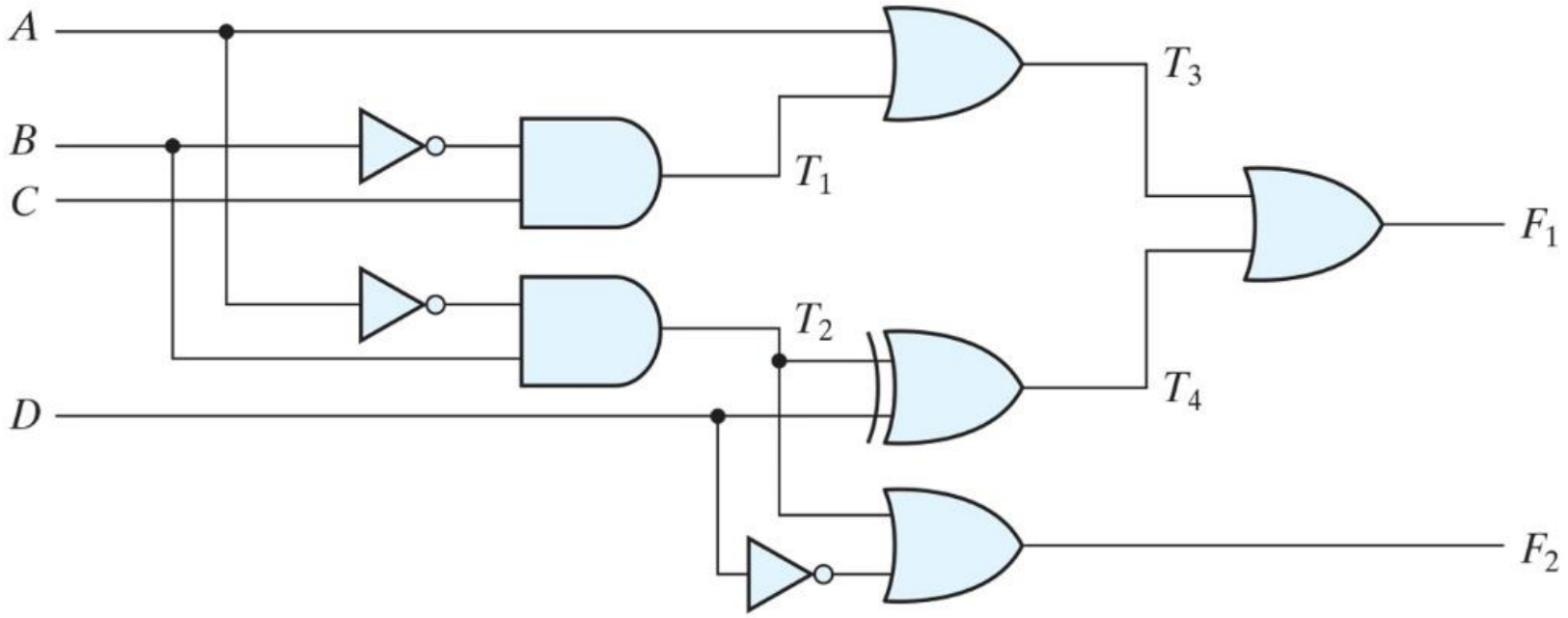


FIGURE P4.2

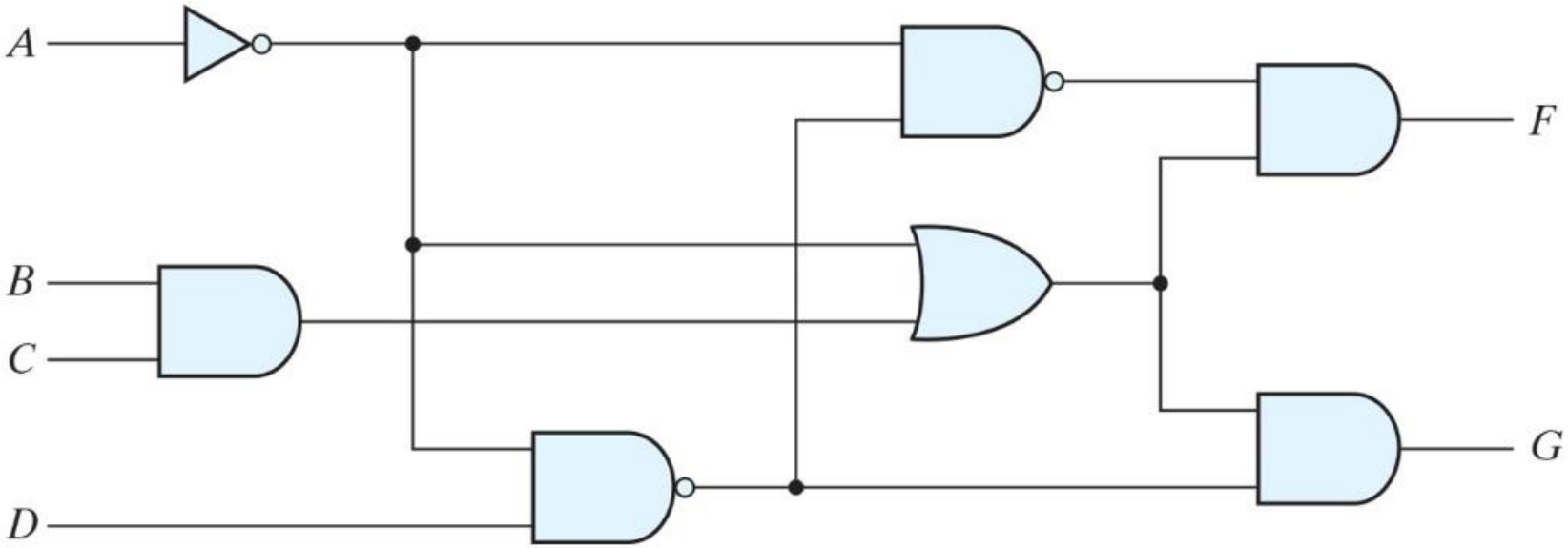


FIGURE P4.9

(a) Segment designation

(b) Numerical designation for display

FIGURE P4.16