ENGG2020 Digital Logic and Systems

Chapter 4: Combinational Logic Circuits

The Chinese University of Hong Kong

Digital Circuit Categories

Combinational Logic Circuits

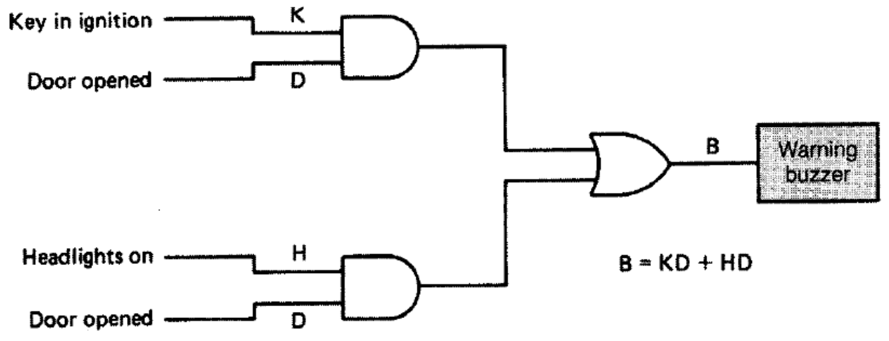
The outputs at any instant of time depend upon the inputs present at that time instant, i.e. no memory in these circuits.

Sequential Logic Circuits

The outputs at any instant of time depend upon the present inputs as well as past inputs/outputs, i.e. there are elements used to store past information.

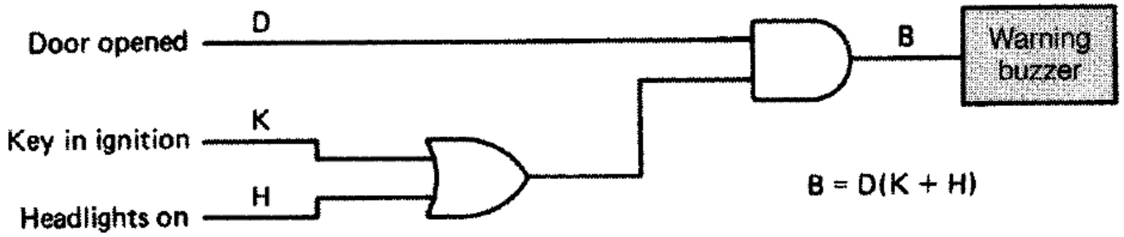
Combinational Logic

Combinational logic is a function that maps inputs to outputs such that the outputs only depend on inputs.



Courtesy of [1]

B = KD + HD can be simplified into B = D(K+H)



Courtesy of [1]

Design Methodology

Whenever possible, decompose a complex design into common, reusable functional blocks

These blocks are tested and well documented.

Top-down design

Proceeds from abstract, high-level specification to a more and more detailed design by decomposition and successive refinement.

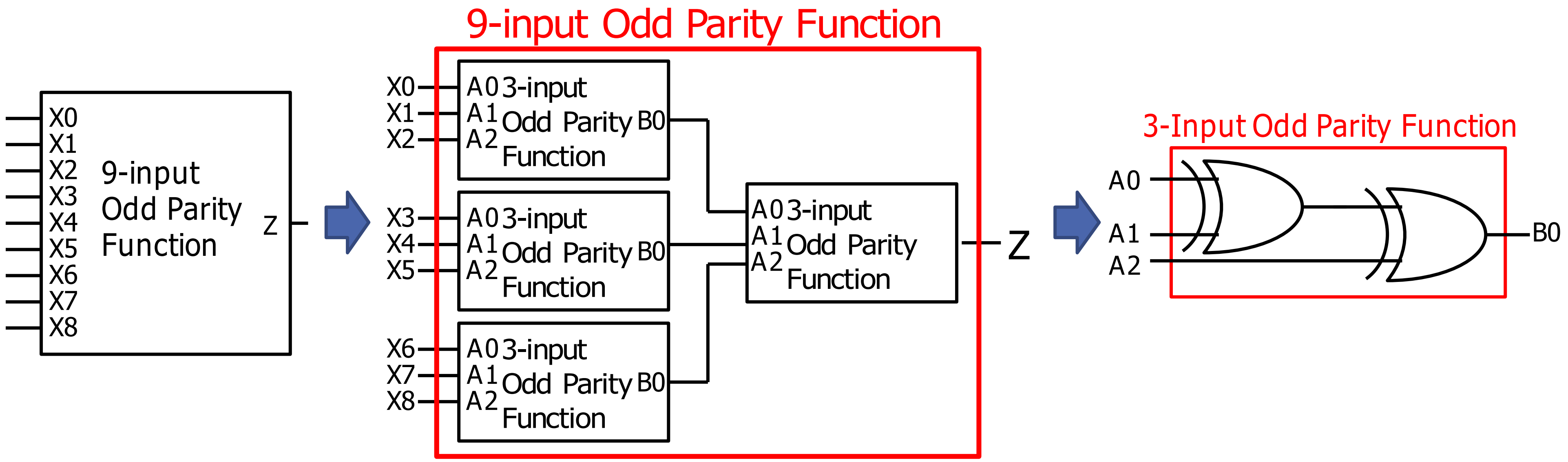
Bottom-up design

Starts with detailed primitive elements and combines them into larger and larger functions.

Designs usually **proceed from both directions** simultaneously.

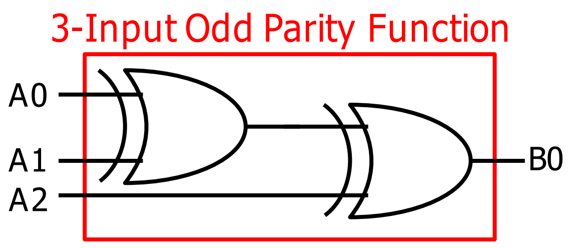
Design Methodology

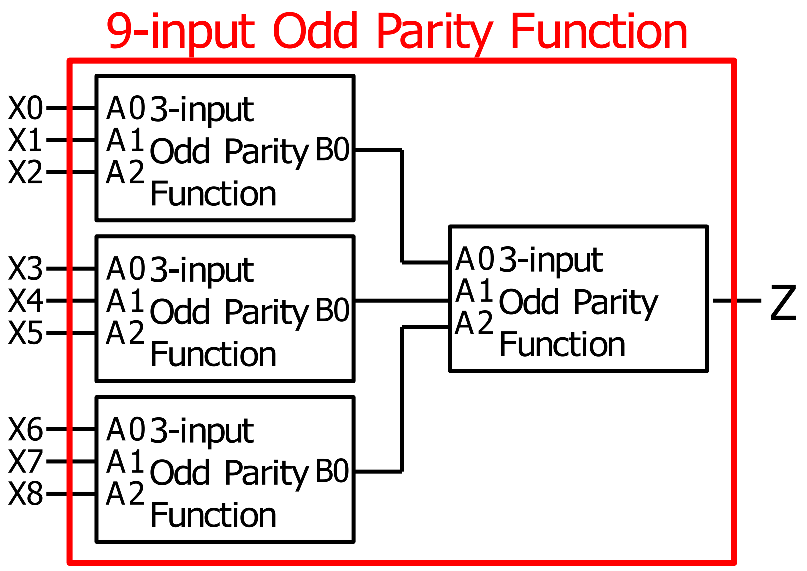
Top-down design answers “What are we building?” and controls complexity.

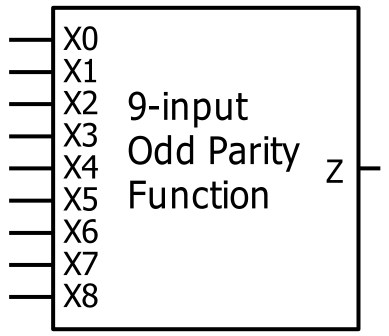

X0
X1
X2
X3
X4
X5
X6
X7
X8
9-input Odd Parity Function
Z
9-input Odd Parity Function
A0
A1
A2
3-input Odd Parity Function
B0
Z
3-Input Odd Parity Function
A0
A1
A2
B0

Design Methodology

Bottom-up design answers “How do we build it?” and handles details.







Analysis

From a design to a specification of the behavior

From a logic diagram to Boolean equations or function table

Logic Simulation is a fast, accurate method of analyzing a circuit to see its logic waveforms or its truth table is correct.

Synthesis

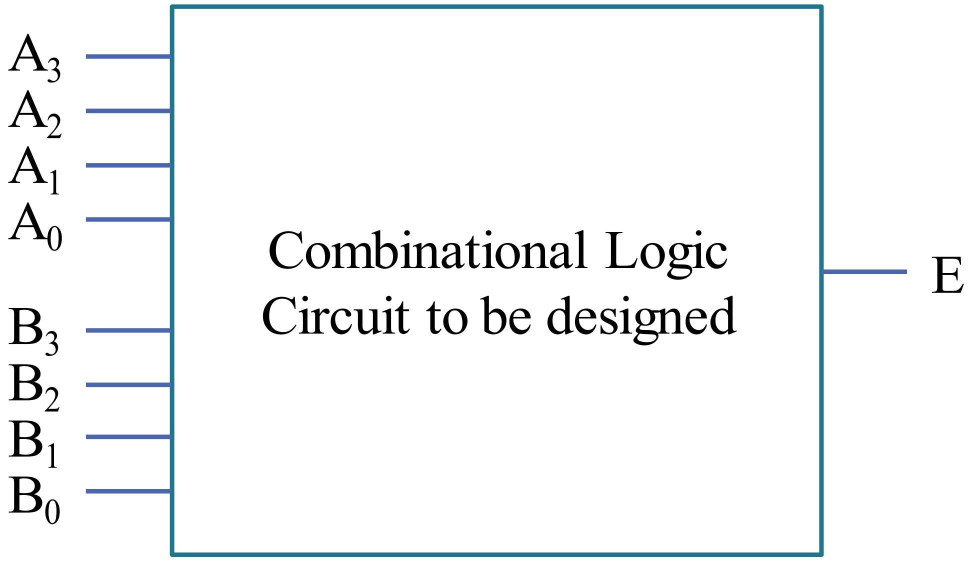
From specification to design implementation

Procedure

- Define the problem (specification)
- Identify the input(s) and output(s)
- Derive function table(s) or equation(s)
- Minimize the Boolean function
- Implement the circuit
- Verify the circuit

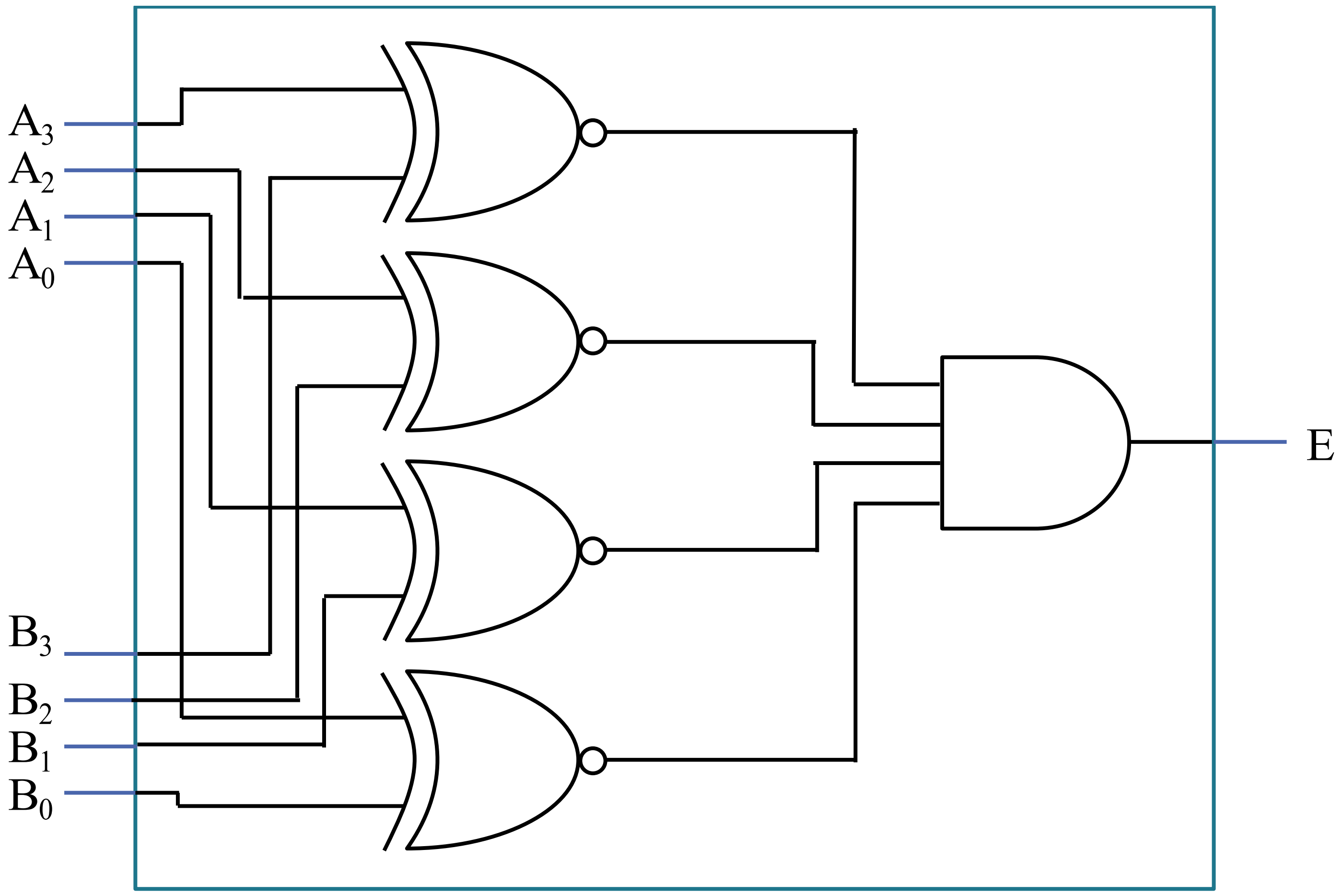
Design of a 4-bit Equality Comparator

Output E is equal to 1 if input A and B are equal and equal to 0 if input A and B are unequal



$$E = (A_3B_3+A_3'B_3')(A_2B_2+A_2'B_2')(A_1B_1+A_1'B_1')(A_0B_0+A_0'B_0')$$

Design of a 4-bit Equality Comparator



Code Converter

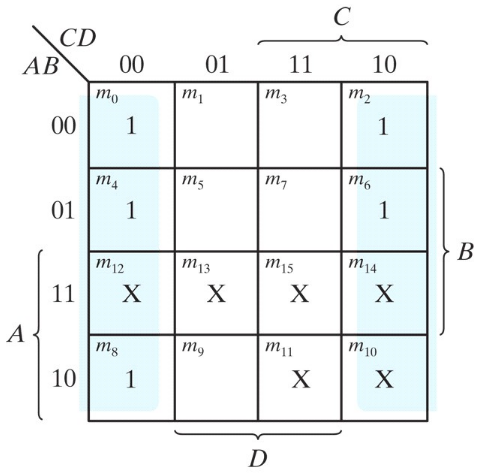
Construct a code converter, which converts BCD code into excess-3 code

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

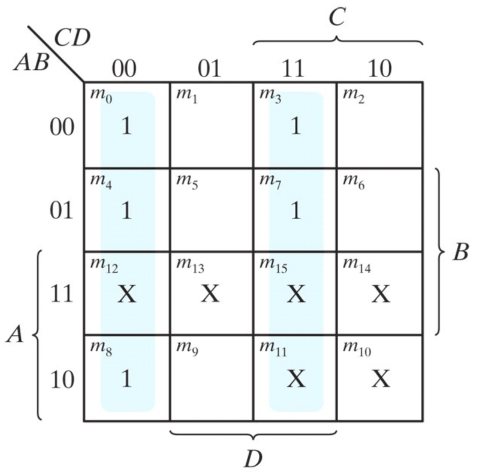
Code Converter

Maps for BCD-to-excess-3 code converter



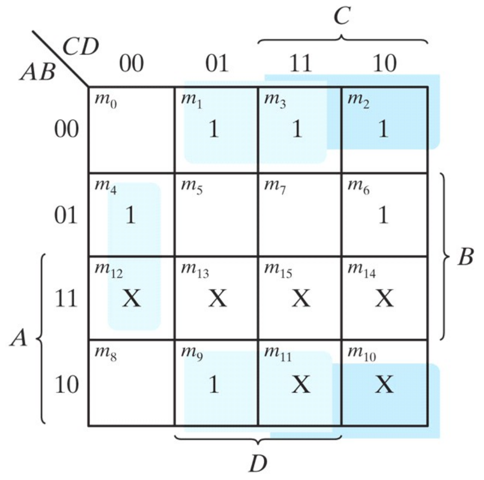
AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3	m_2 1
01	m_4 1	m_5	m_7	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9	m_{11} X	m_{10} X

$z = D'$



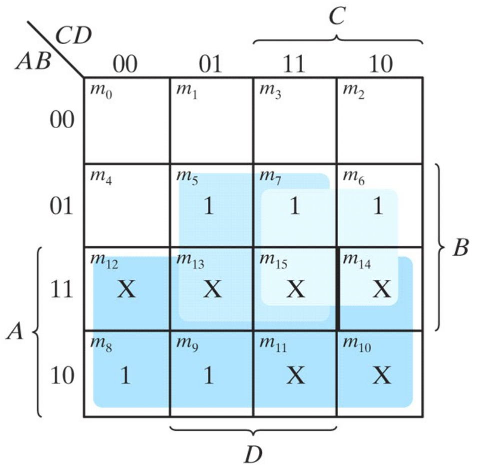
AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3 1	m_2
01	m_4 1	m_5	m_7 1	m_6
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9	m_{11} X	m_{10} X

$y = CD + C'D'$



AB \ CD	00	01	11	10
00	m_0	m_1 1	m_3 1	m_2 1
01	m_4 1	m_5	m_7	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8	m_9 1	m_{11} X	m_{10} X

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

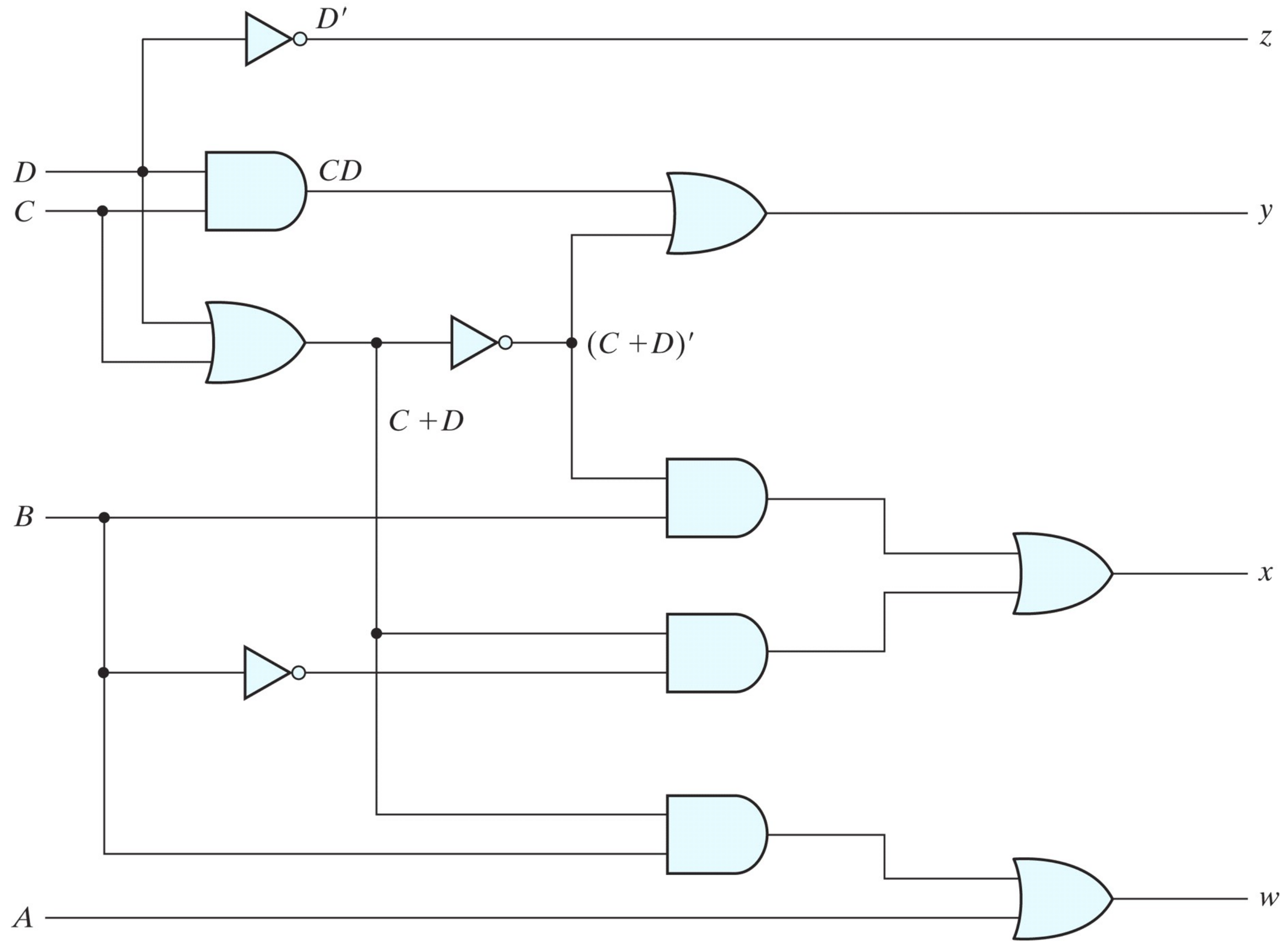


AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5 1	m_7 1	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9 1	m_{11} X	m_{10} X

$w = A + BC + BD$

Code Converter

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



Decoder

Decoding is the conversion of an n-bit input code to an m-bit output code with $n \leq m \leq 2^n$, such that each valid input code word produces a unique output code.

Decoder may have unused bit combinations on its inputs for which no corresponding m-bit code appears at the outputs.

Use a decoder and an OR gate to form SOP directly.

Decoder

Truth table of a three-to-eight-line Decoder

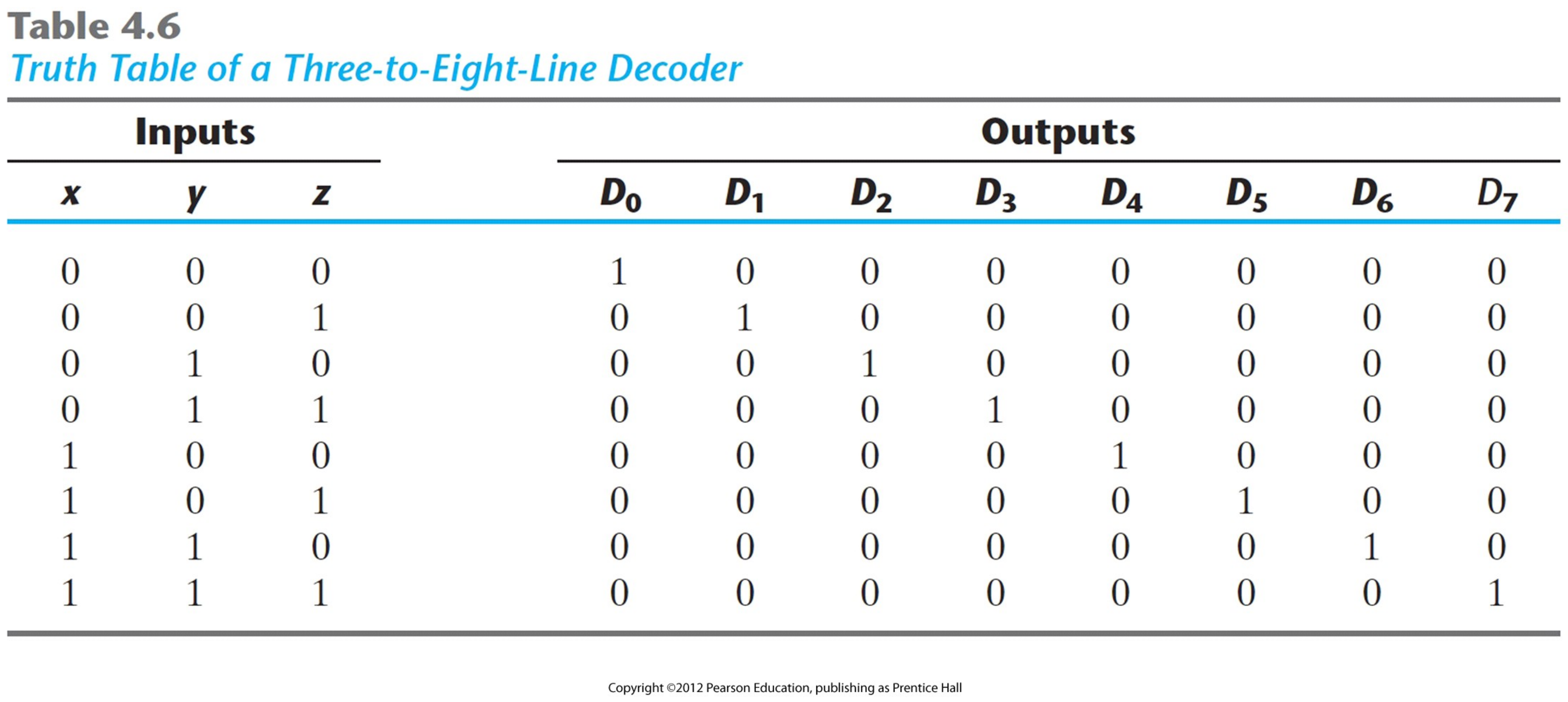
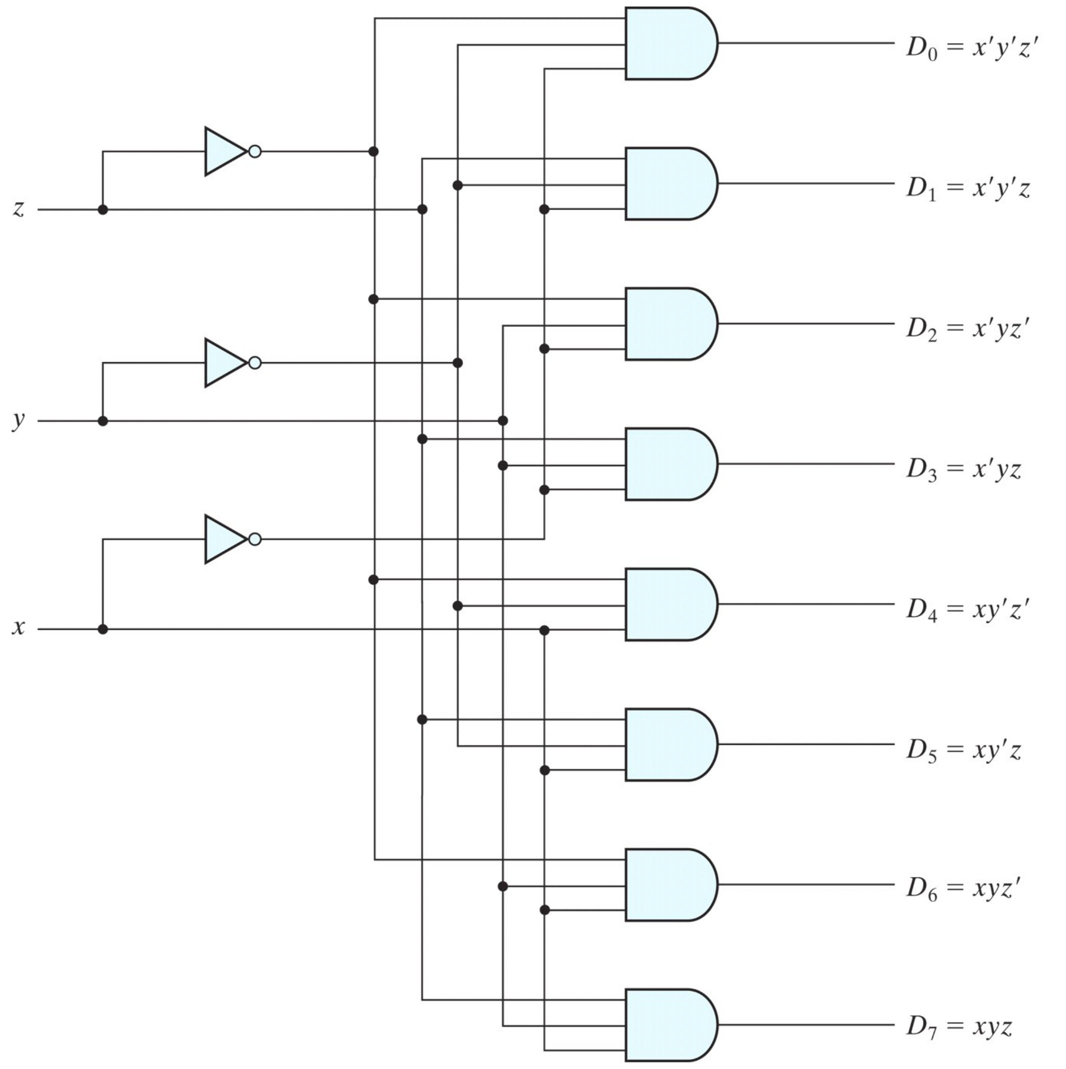


Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

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

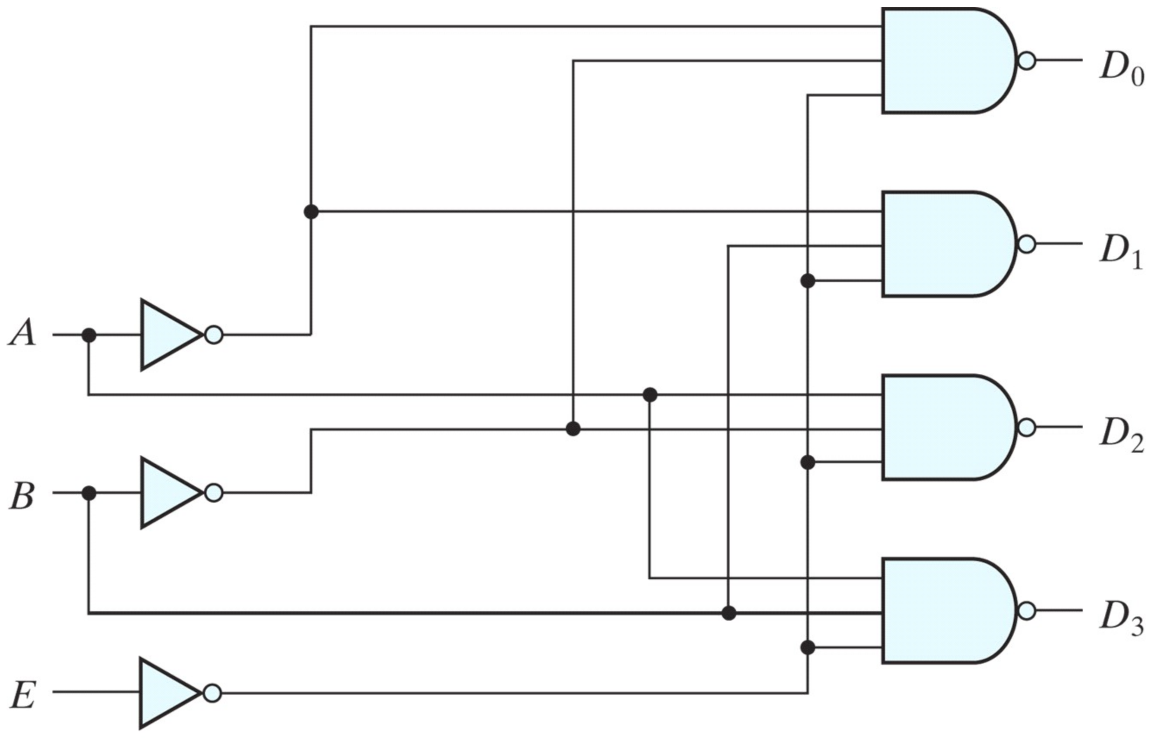
Decoder

Logic diagram of a three-to-eight-line decoder



Decoder

Two-to-four-line decoder with enable input



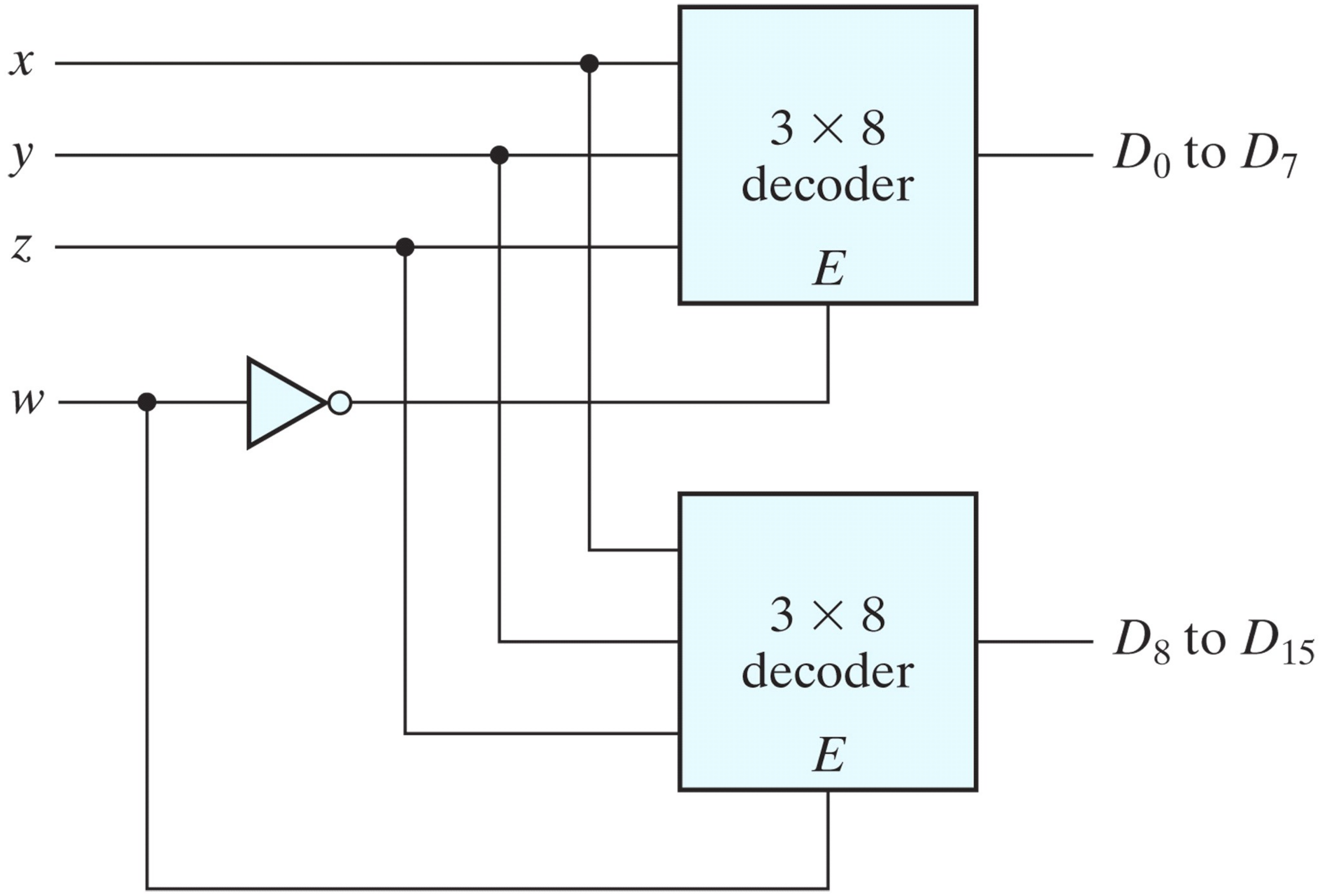
(a) Logic diagram

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

(b) Truth table

Decoder

4 × 16 decoder constructed with two 3 × 8 decoders



Encoder

Encoding is the inverse operation of decoding.

Encoder takes a code from one format and encode the code into another format

An encoder has $2^{\square\square}$(or fewer) input lines and n output lines

The output lines generate the binary code corresponding to the input value

Priority encoder

If two or more inputs are equal to 1 at the same time, the input having the highest priority takes precedence

Encoder

An Octal-to-Binary Encoder

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

Encoder

An Octal-to-Binary Encoder

Boolean output functions:

$x = D_4+D_5+ D_6+ D_7$

$y = D_2+D_3+ D_6+ D_7$

$z = D_1+D_3+ D_5+ D_7$

Encoder

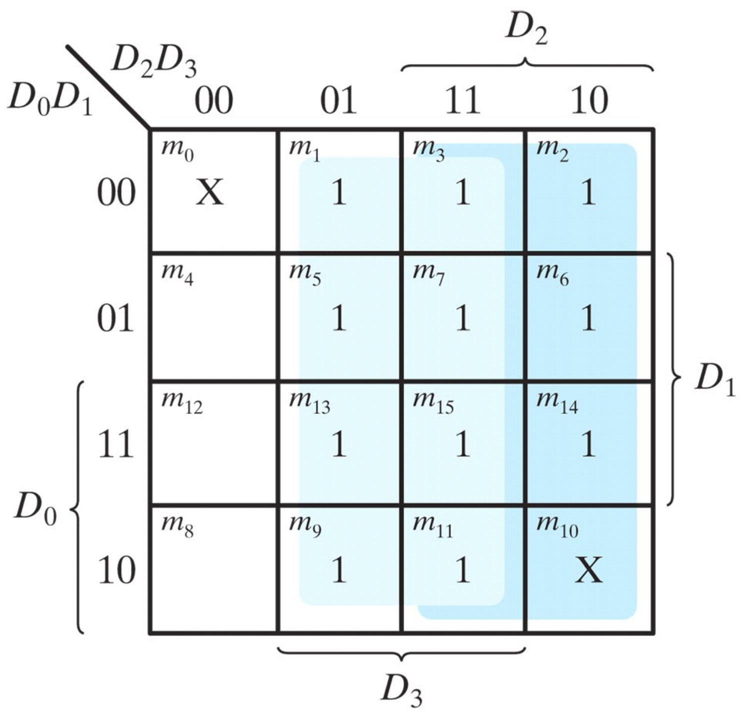
Priority Encoder

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

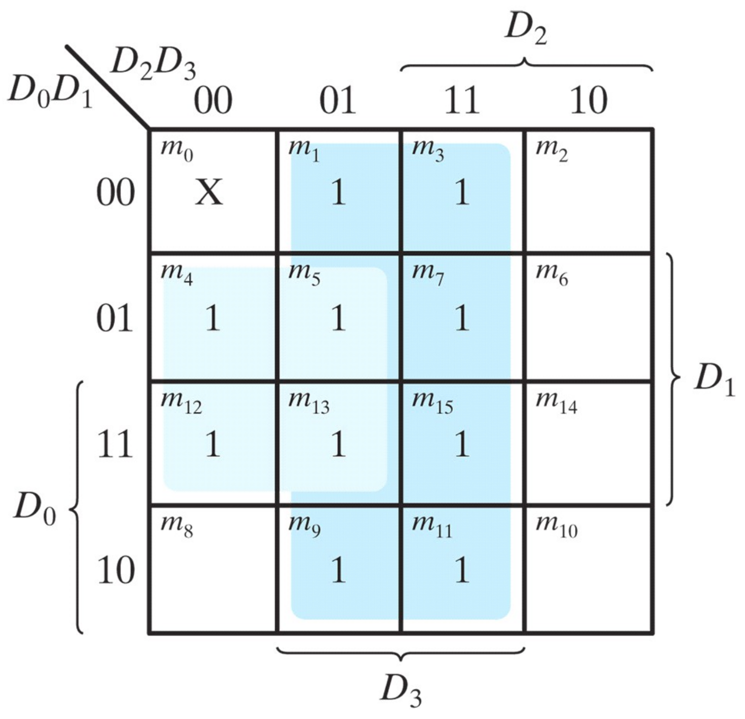
Encoder

Maps for priority encoder



D_2D_3
D_0D_1
00 01 11 10
D_2
D_1
D_0
D_3
m_0 X, m_1 1, m_3 1, m_2 1
m_4, m_5 1, m_7 1, m_6 1
m_{12}, m_{13} 1, m_{15} 1, m_{14} 1
m_8, m_9 1, m_{11} 1, m_{10} X

$x = D_2 + D_3$

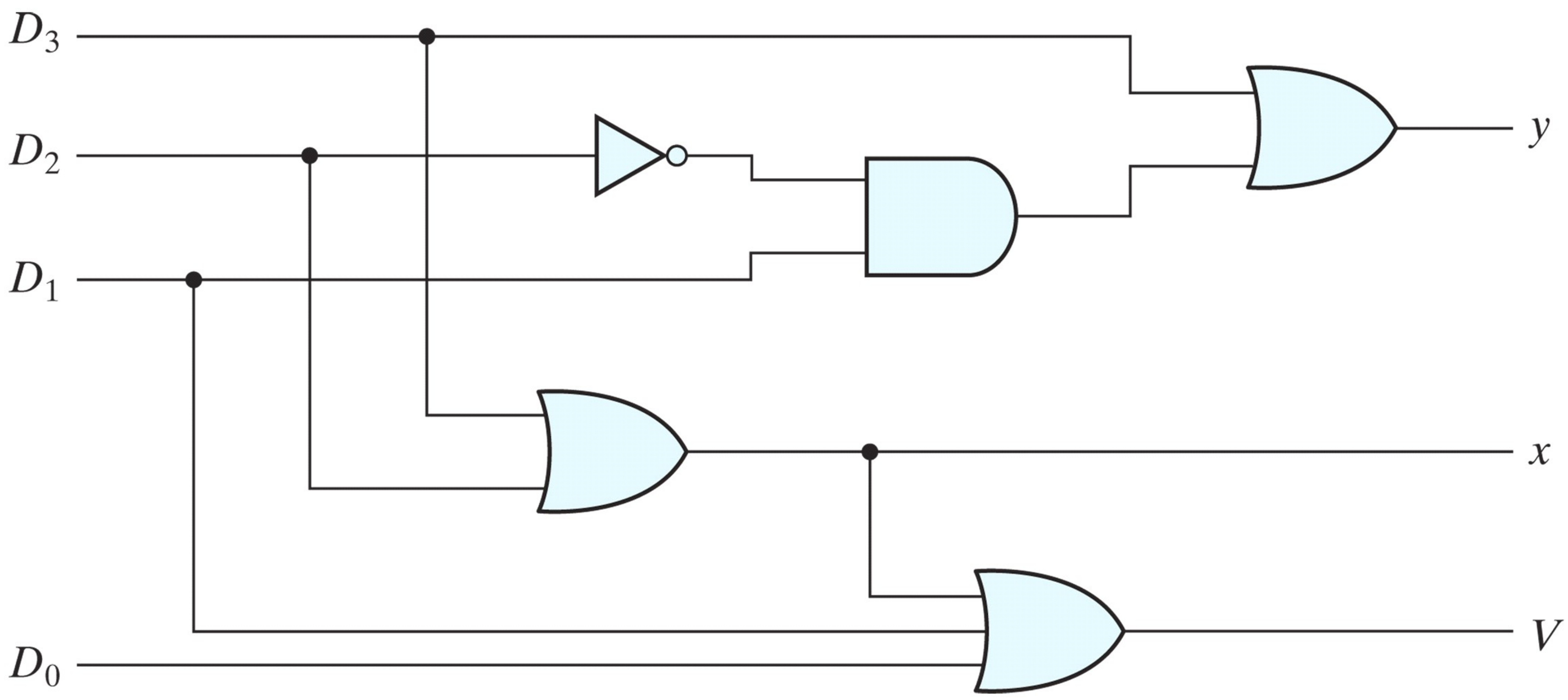


D_2D_3
D_0D_1
00 01 11 10
D_2
D_1
D_0
D_3
m_0 X, m_1 1, m_3 1, m_2
m_4 1, m_5 1, m_7 1, m_6
m_{12} 1, m_{13} 1, m_{15} 1, m_{14}
m_8, m_9 1, m_{11} 1, m_{10}

$y = D_3 + D_1D'_2$

Encoder

Logic diagram for four-input priority encoder

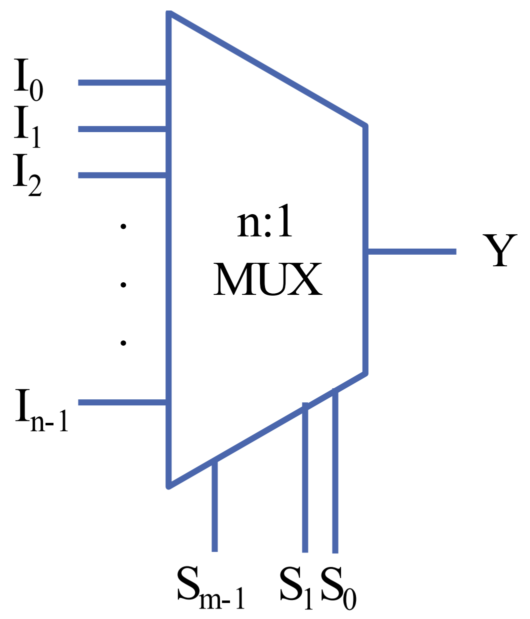


Multiplexer

Also called **Data Selector.**

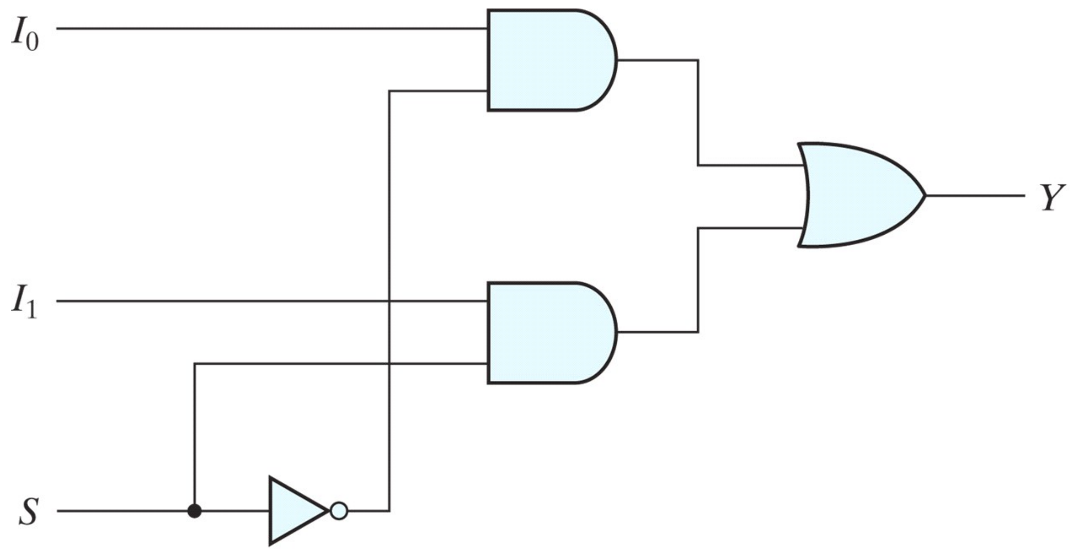
A logic circuit that gates one out of several inputs to a single output.

Standard ICs are available for 2:1, 4:1, 8:1, and 16:1 multiplexers.

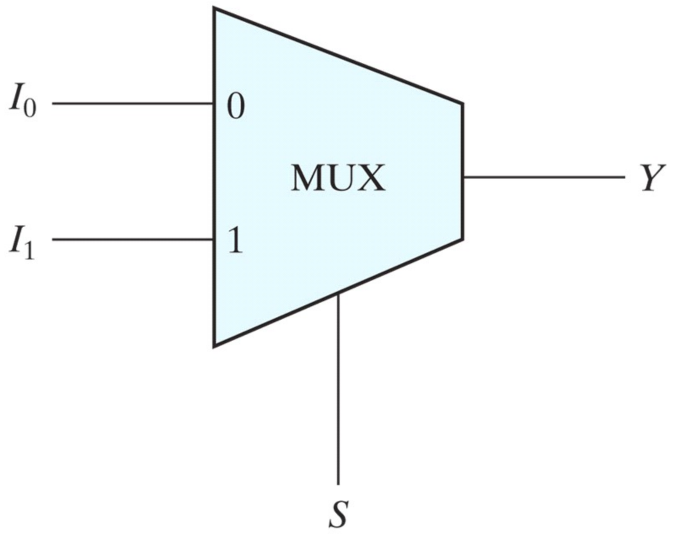


Multiplexer

Two-to-one-line multiplexer



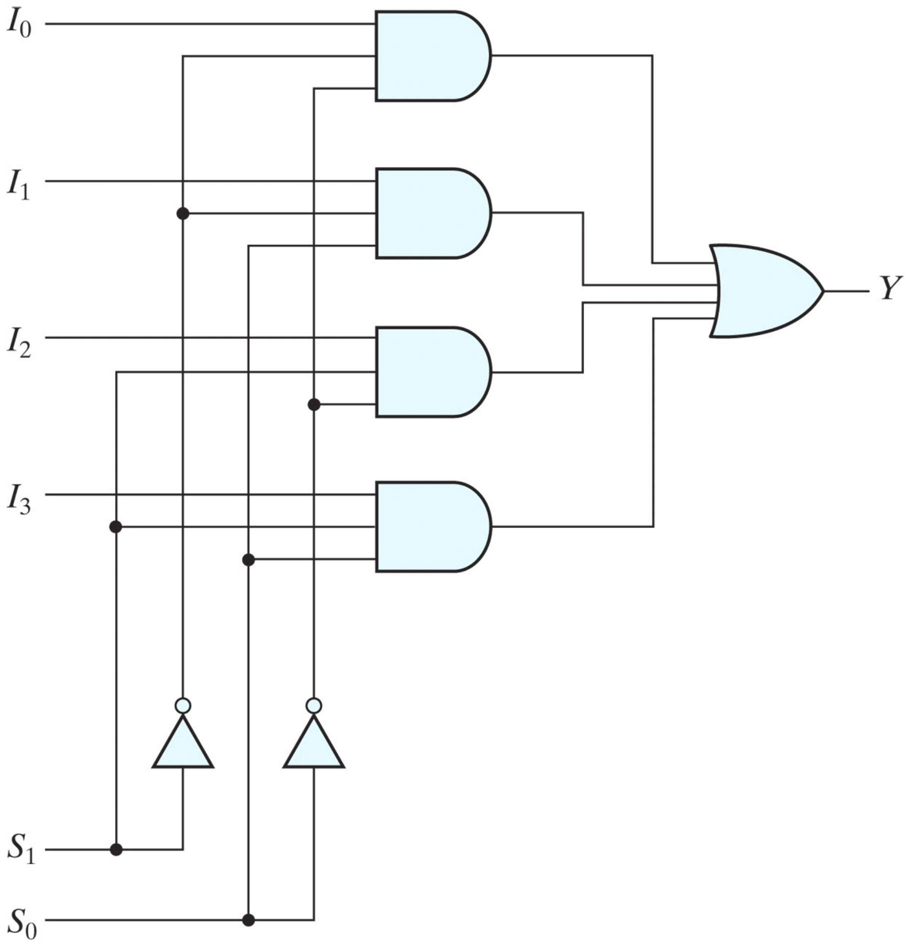
(a) Logic diagram



(b) Block diagram

Multiplexer

Four-to-one-line multiplexer



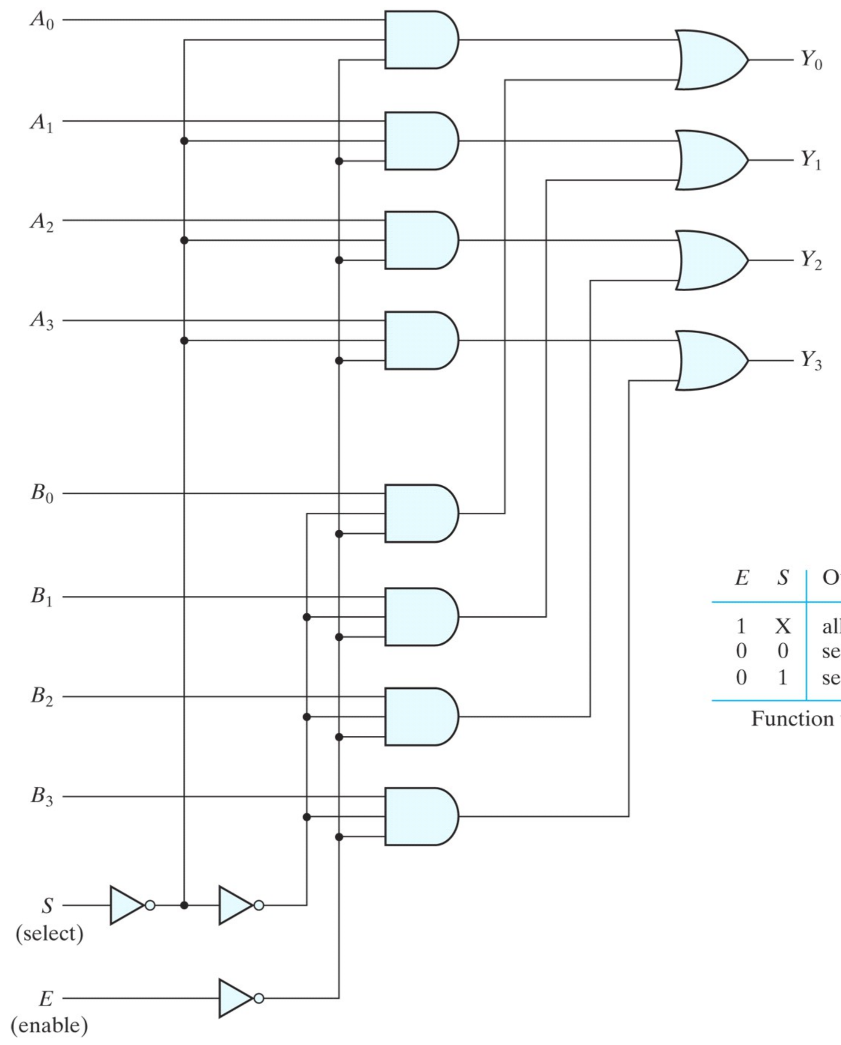
(a) Logic diagram

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

(b) Function table

Multiplexer

Quadruple two-to-one-line multiplexer



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

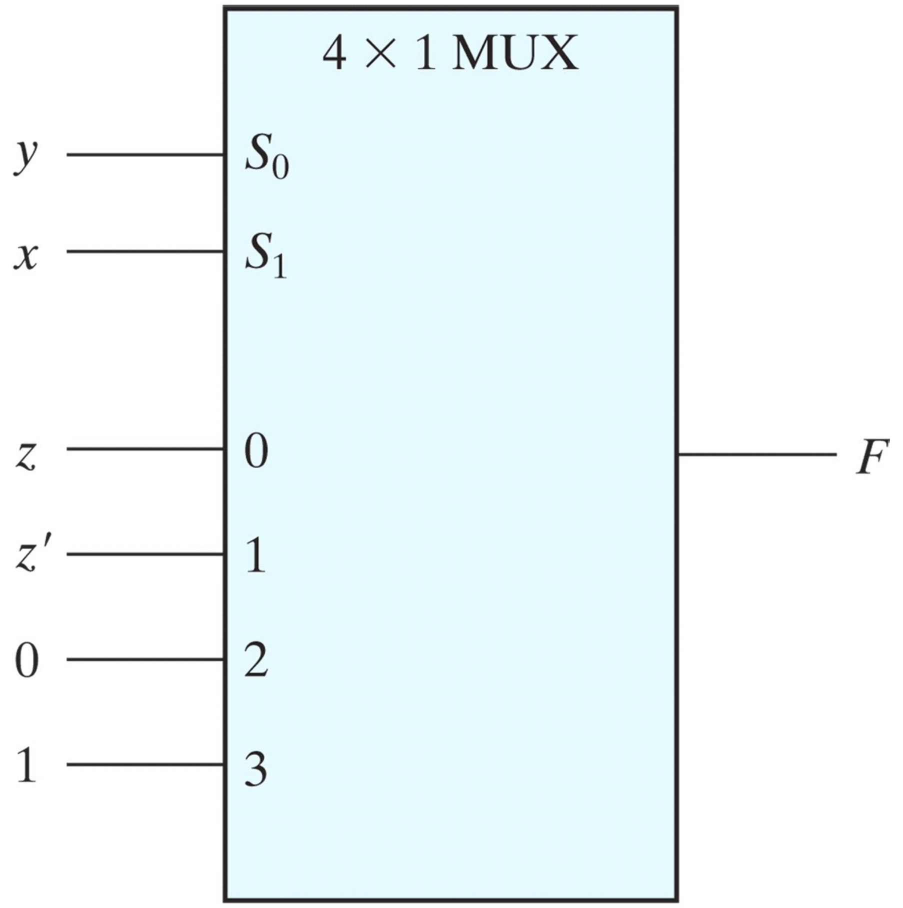
Function table

Multiplexer

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

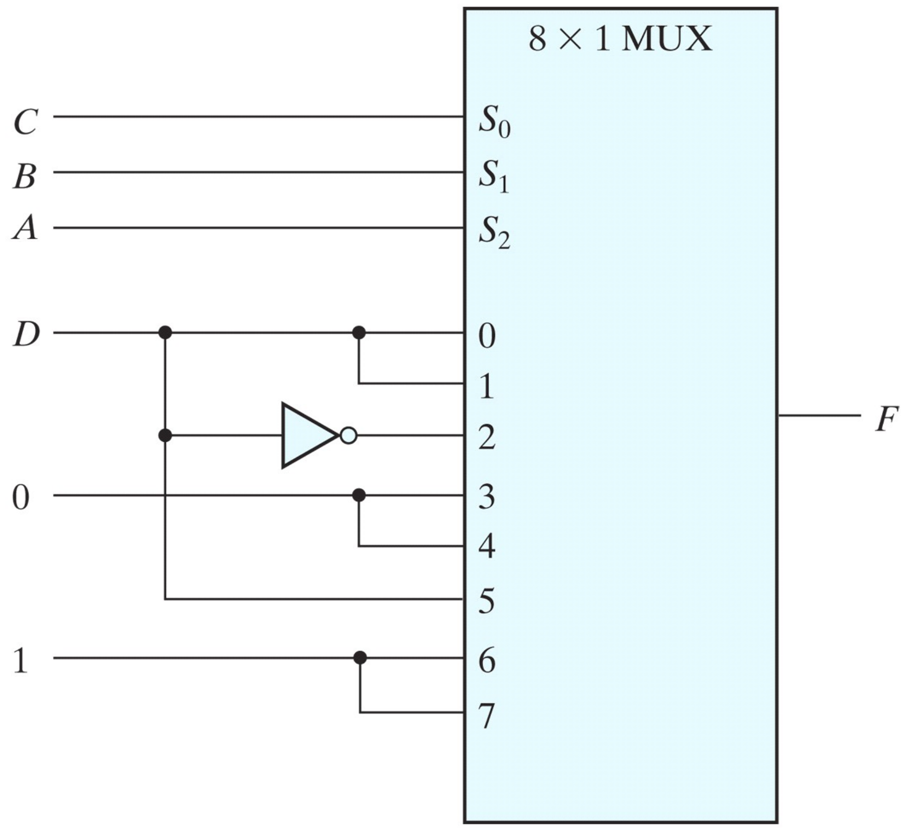


(b) Multiplexer implementation

Multiplexer

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	

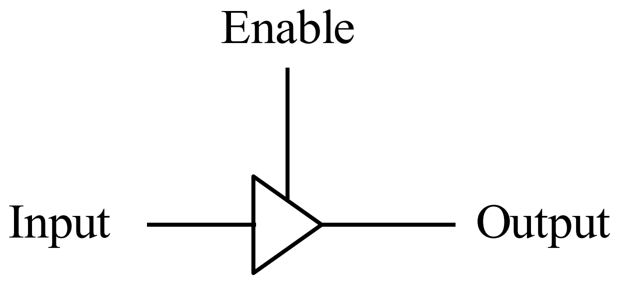


Tri-State Buffer

Tri-State logic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels.

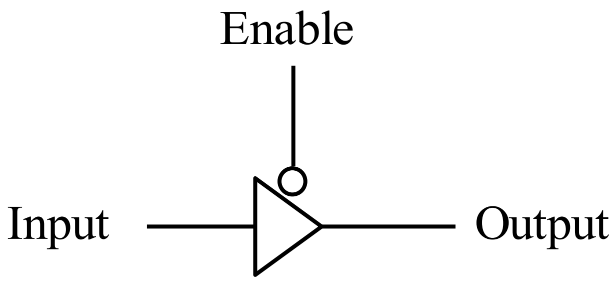
High impedance state effectively remove the device's influence from the rest of the circuit.

This allows multiple circuits to share the same output line(s).



Active-High Enable

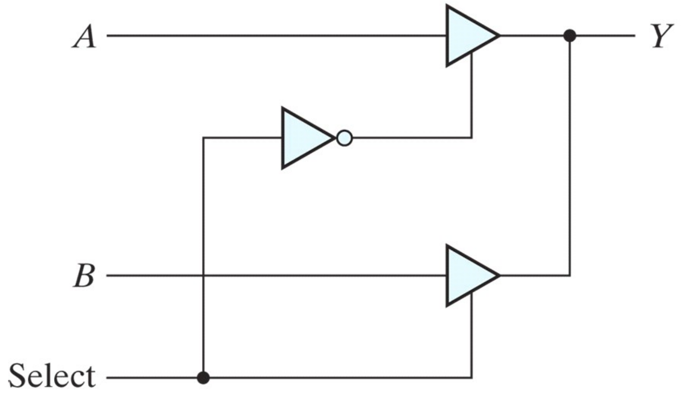
Active-High Enable	Input	Output
0	0	Z
0	1	Z
1	0	0
1	1	1



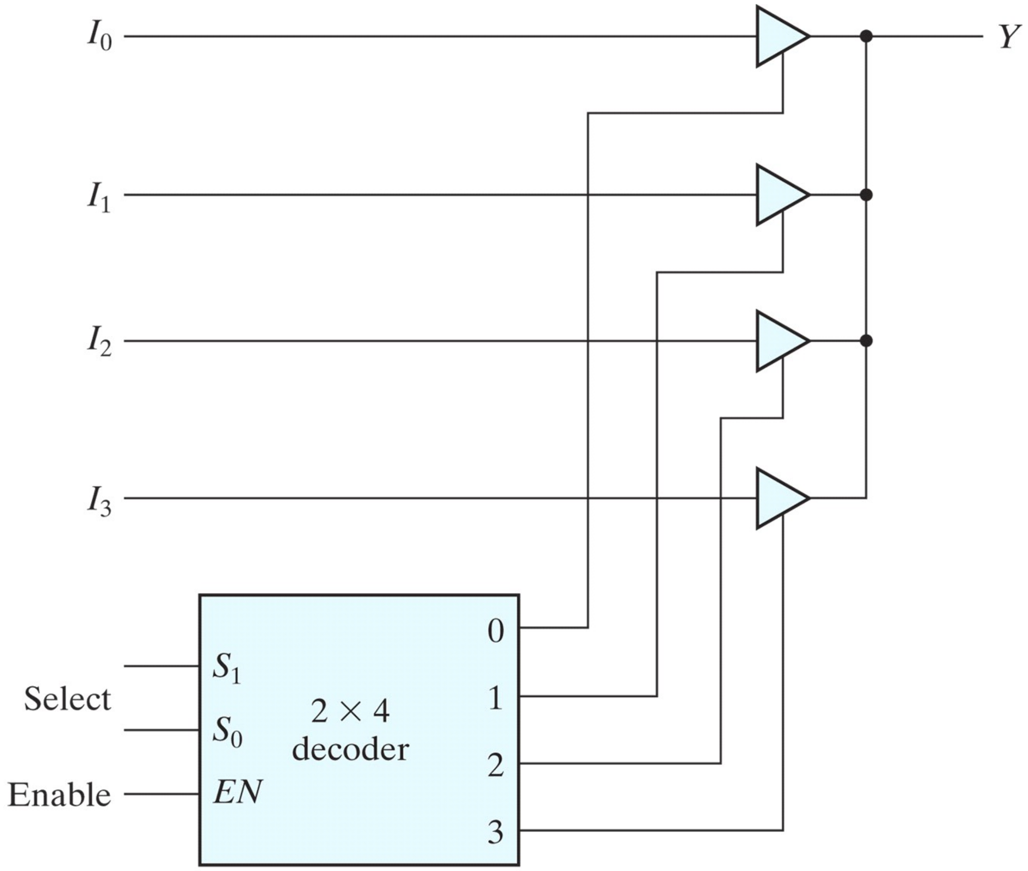
Active-Low Enable

Tri-State Buffer

Multiplexers with three-state gates



(a) 2-to-1-line mux

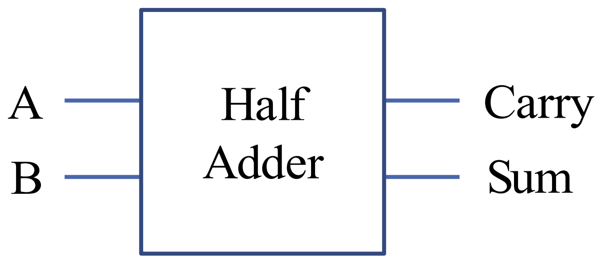


(b) 4-to-1-line mux

Half Adder

A circuit adds two inputs to generate two outputs based on the rules for binary addition.

$$\begin{array}{lrr} & & A \\ + & & B \\ \hline & \text{Carry} & \text{Sum} \end{array}$$



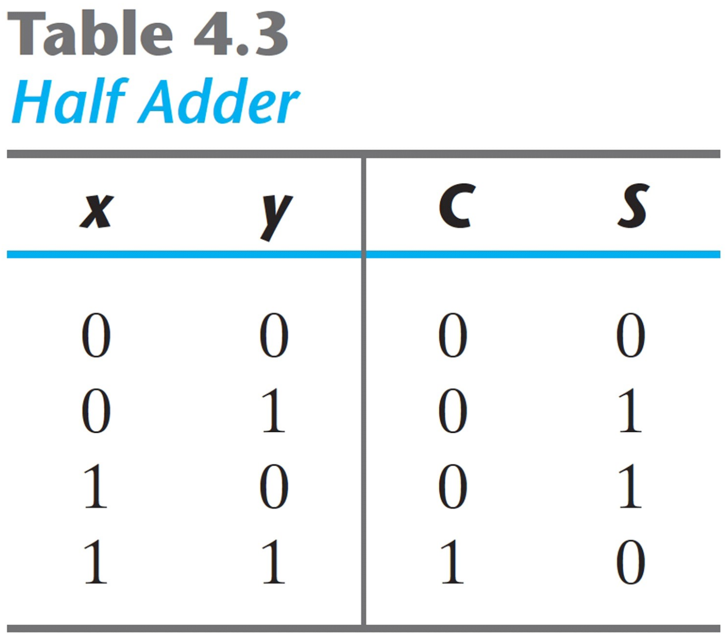
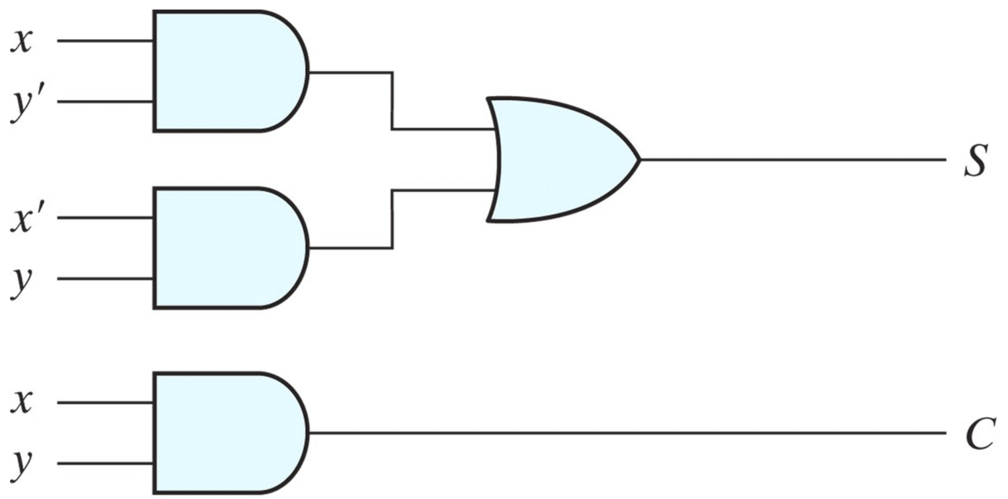


Table 4.3
Half Adder

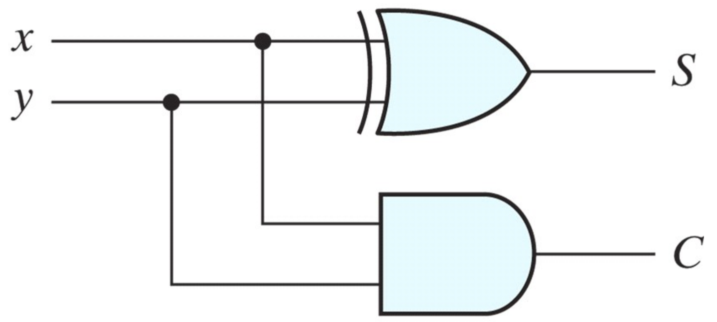
x	*y*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder

Implementation of half adder



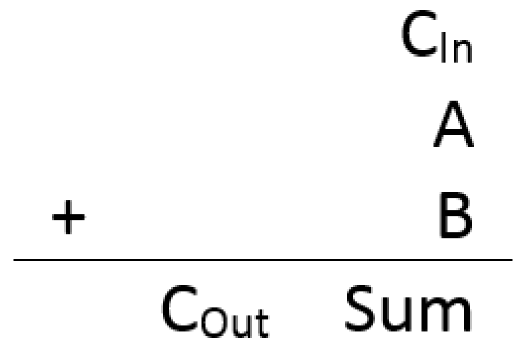
(a) $S = xy' + x'y$
$C = xy$



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

Full Adder

A circuit adds two inputs and a carry from a previous addition together to form sum and carry.



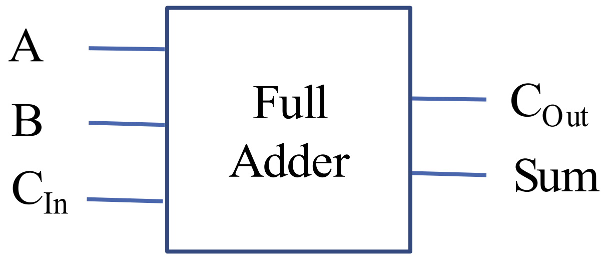
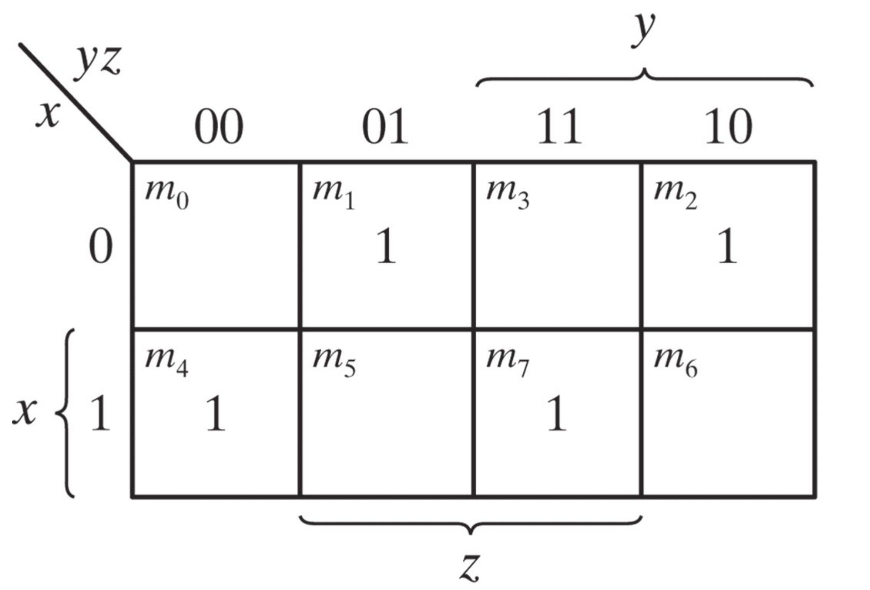


Table 4.4
Full Adder

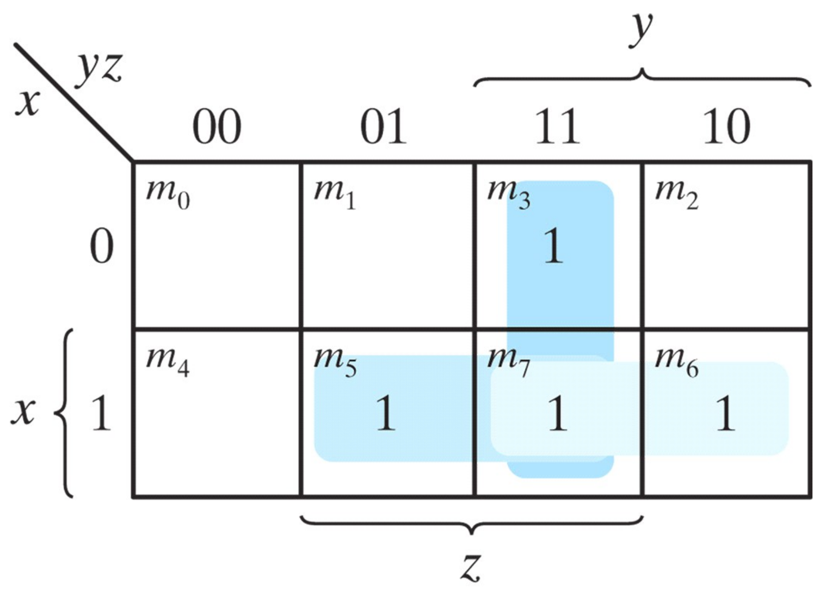
x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder

K-Maps for full adder



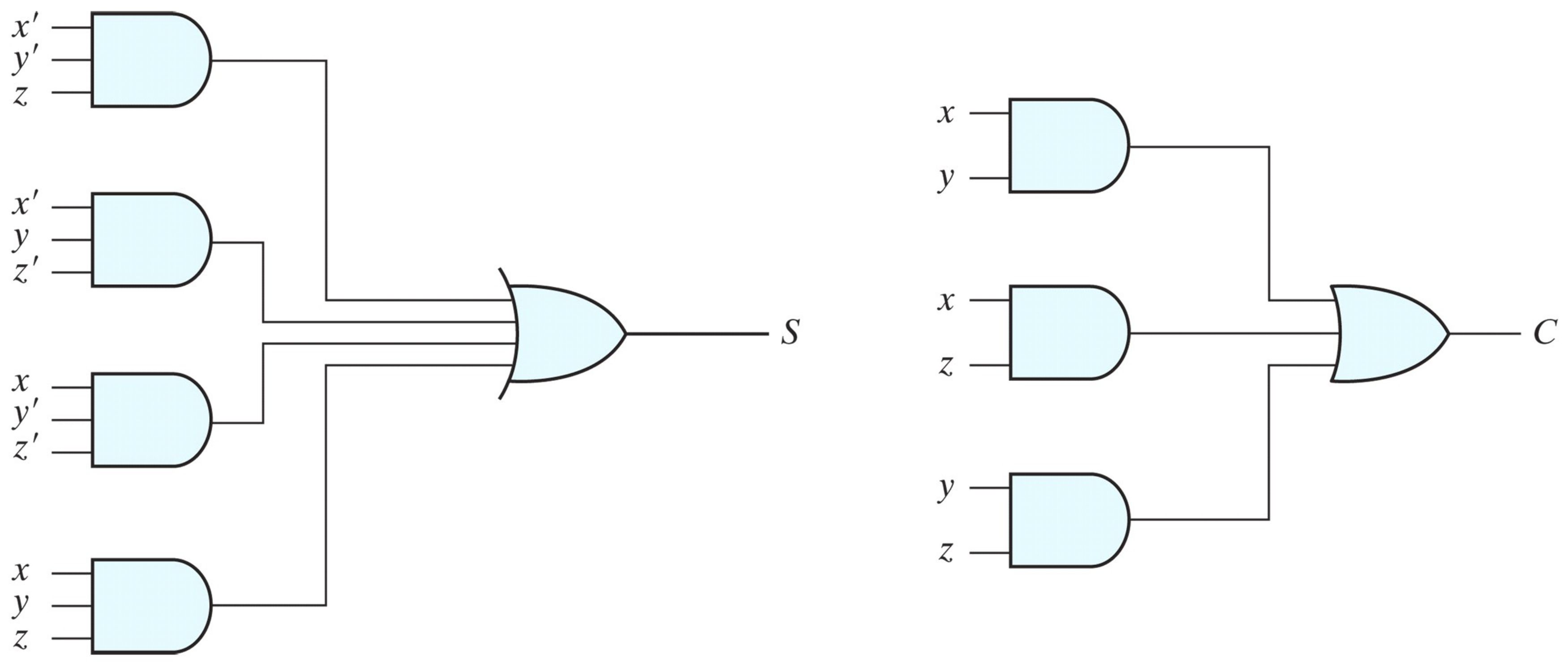
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



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

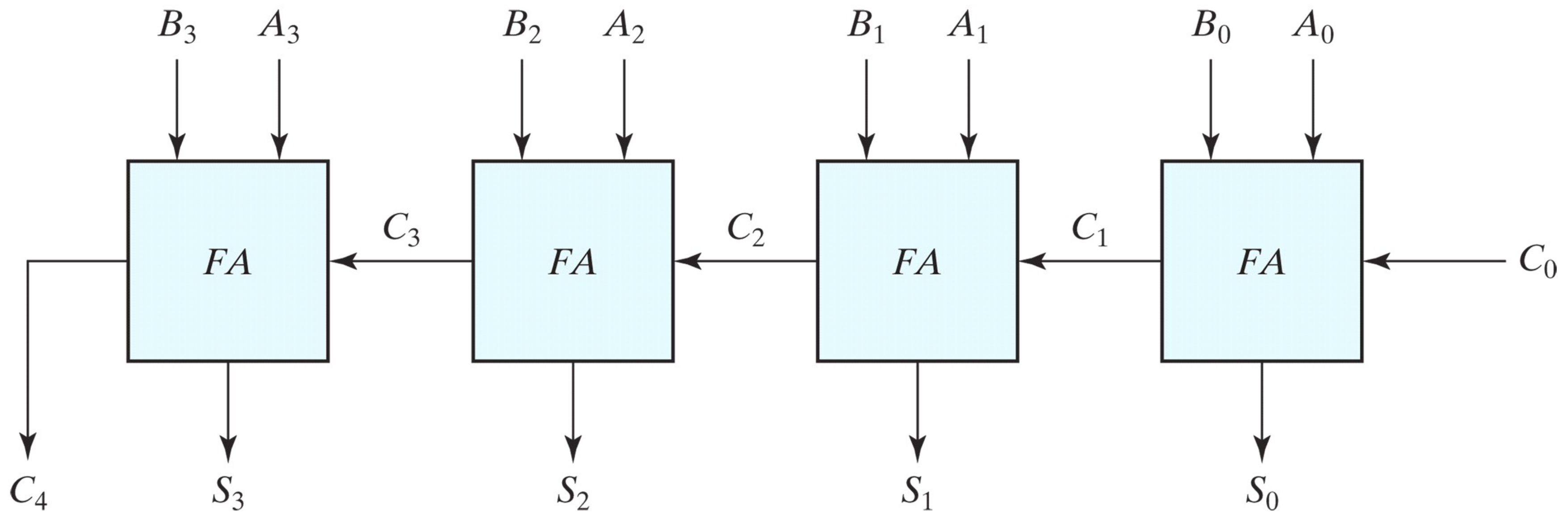
Full Adder

Implementation of full adder in sum-of-products form



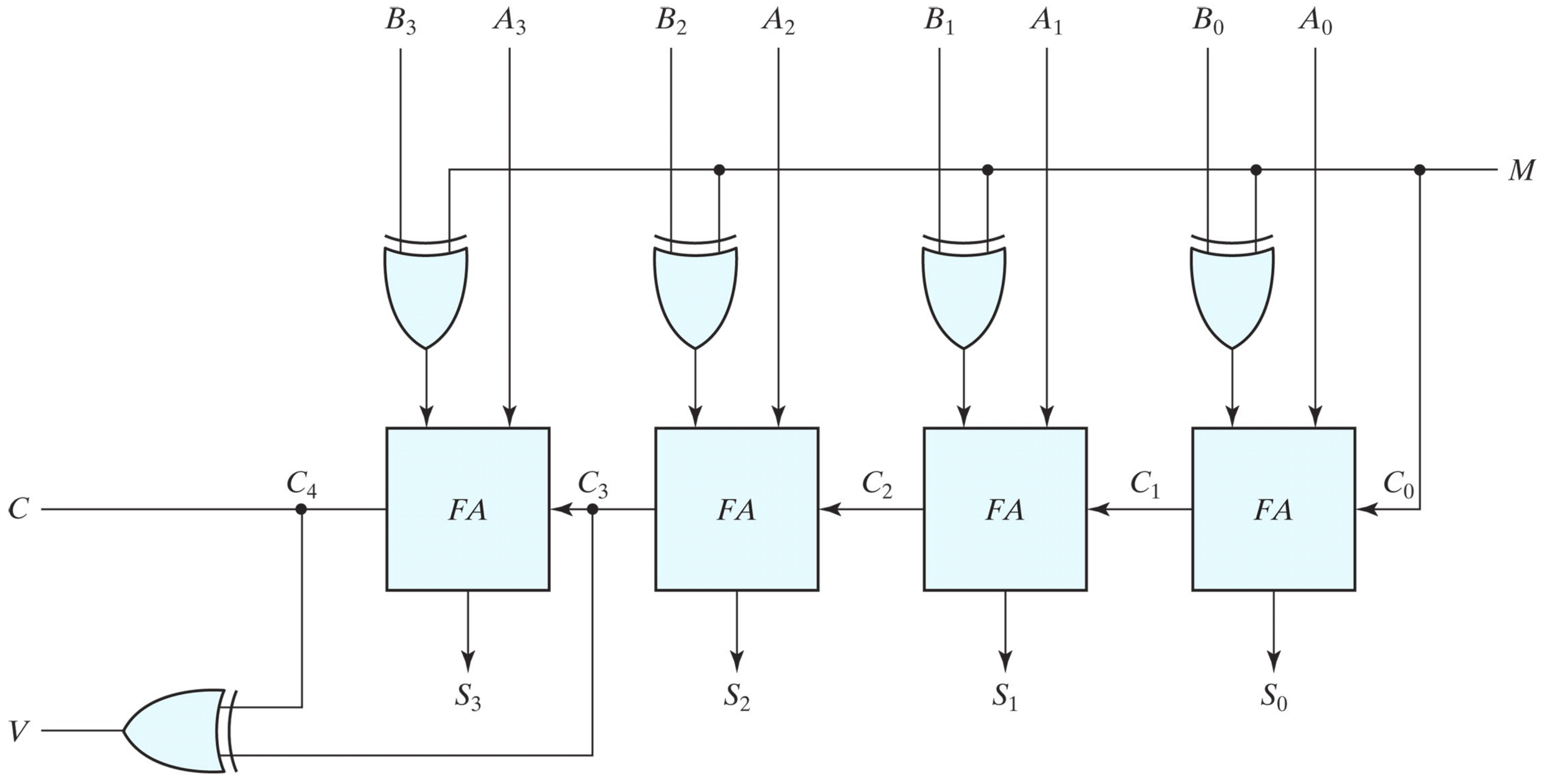
Four-bit adder

Four-bit adder composing of full adders



Four-bit adder–subtractor

Four-bit adder–subtractor (with overflow detection)



BCD Adder

Addition of two decimal digits in BCD

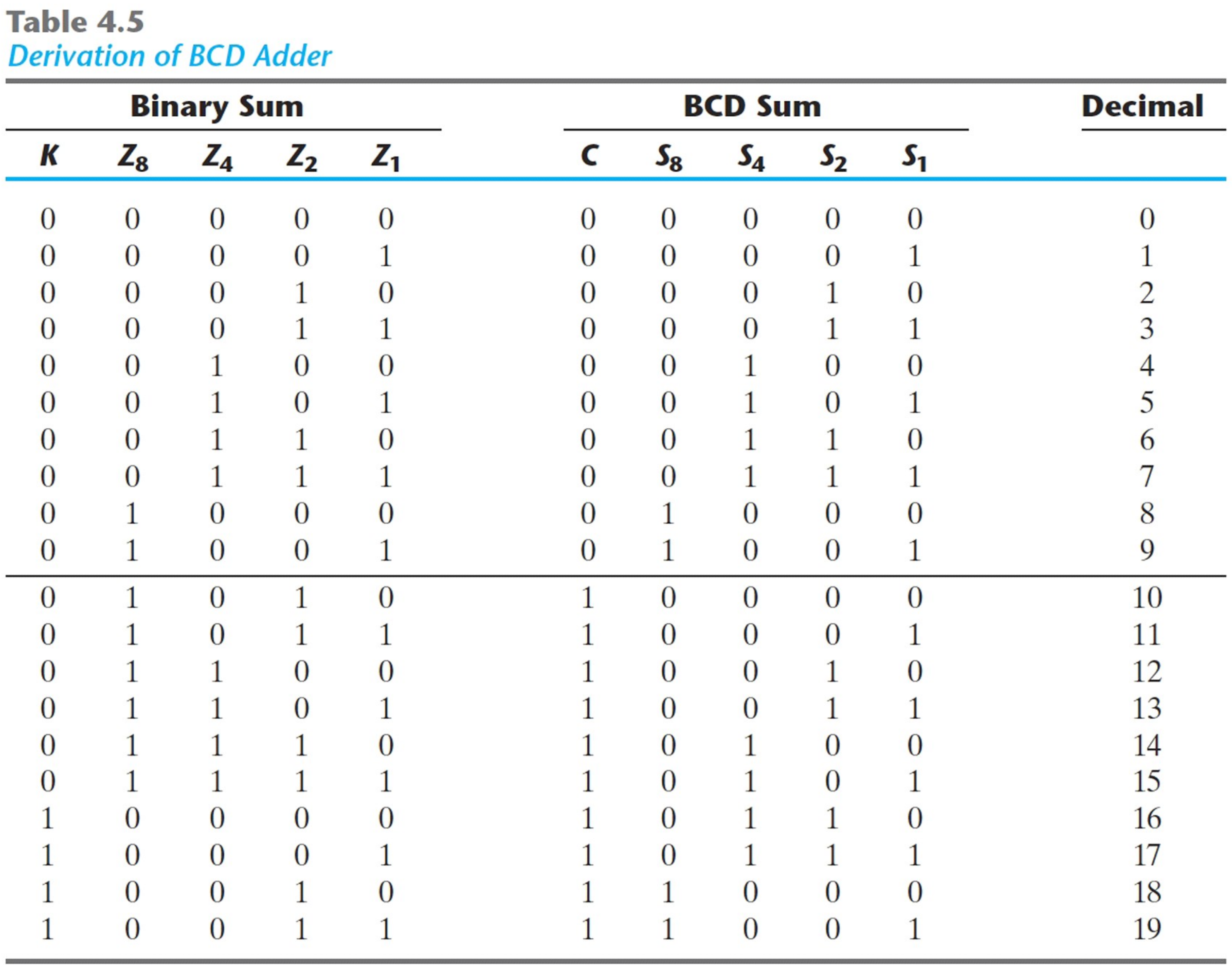


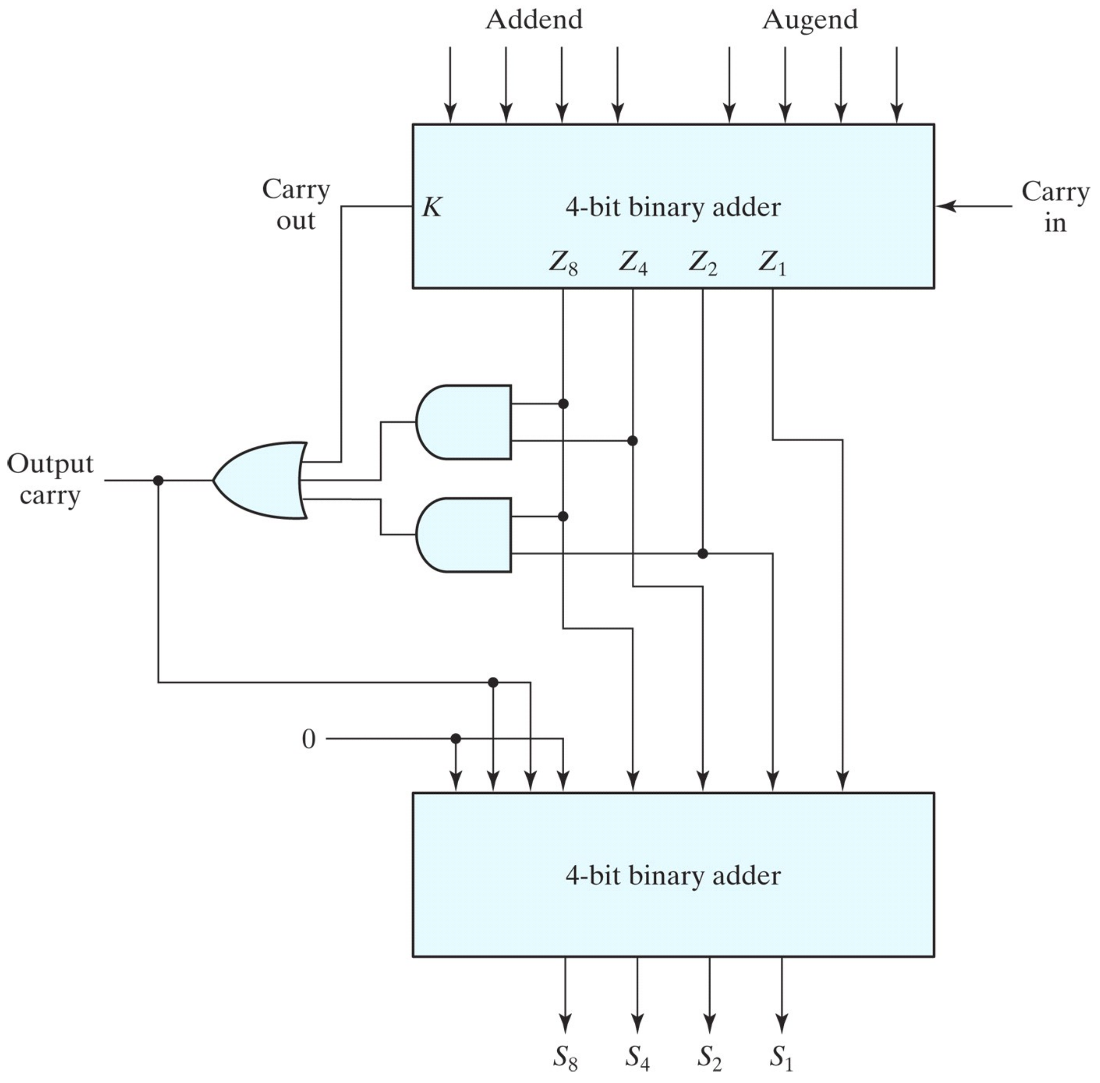
Table 4.5
Derivation of BCD Adder

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
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

BCD Adder

The condition for a correction and an output carry

$C = K + Z_8Z_4 + Z_8Z_2$



Block diagram of a BCD adder