DIGITAL ELECTRONICS UNIT-2

Dr. RIBHU ABHUSAN PANDA

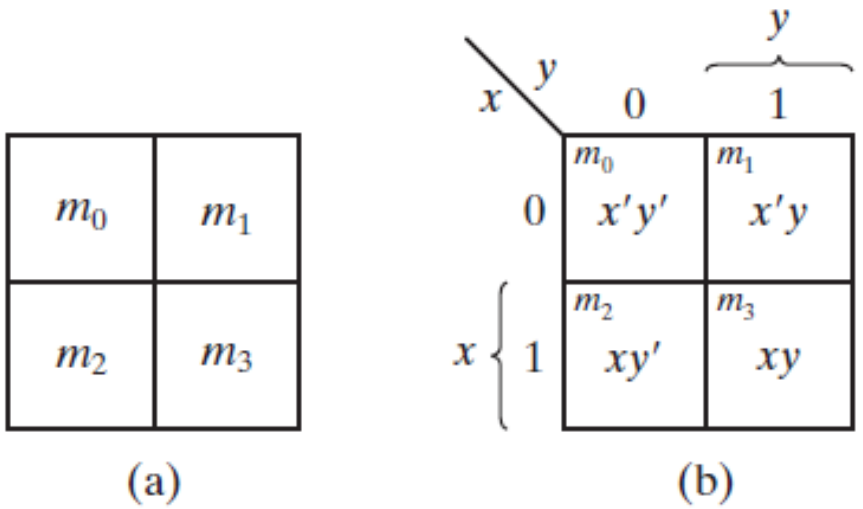
K-maps - Two, Three, and Four Variable K-maps, Don’t-Care Conditions

THE MAP METHOD

- The map method provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table.
- The map method is also known as the *Karnaugh map* or *K-map*.
- The simplified expressions produced by the map are always in one of the two standard forms: sum of products or product of sums.

Two-Variable K-Map

- The two-variable K-map is shown in Fig.
- There are four minterms for two variables; hence, the map consists of four squares, one for each minterm.



- The map is redrawn in (b) to show the relationship between the squares and the two variables *x* and *y*.
- The 0 and 1 marked in each row and column designate the values of variables.
- Variable *x* appears primed in row 0 and unprimed in row 1.
- Similarly, *y* appears primed in column 0 and unprimed in column 1.

The rules of K-map simplification are:

- Groupings can contain only 1s; no 0s.
- Groups can be formed only at right angles (horizontal or vertical); diagonal groups are not allowed.
- The number of 1s in a group must be a power of 2 – even if it contains a single 1.
- The groups must be made as large as possible.

- Each cell containing a one must be in at least one group.
- Groups can overlap.
- Groups can wrap around the sides of the *K*-map.
  - The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.

Example: Simplify the Boolean functions

$F(x, y) = \sum(0, 1)$ $F(x, y) = \sum(2, 3)$ $F(x, y) = \sum(0, 2)$ $F(x, y) = \sum(1, 3)$

x \ y	0	1
0	1	1
1	0	0

x \ y	0	1
0	0	0
1	1	1

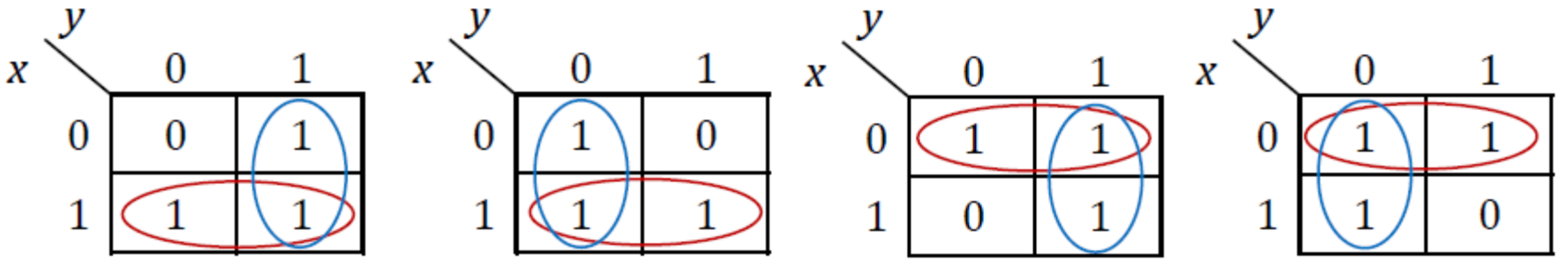
x \ y	0	1
0	1	0
1	1	0

x \ y	0	1
0	0	1
1	0	1

$F(x, y) = x'$ $F(x, y) = x$ $F(x, y) = y'$ $F(x, y) = y$

Example: Simplify the Boolean functions

$F(x, y) = \sum(1, 2, 3)$ $F(x, y) = \sum(0, 2, 3)$ $F(x, y) = \sum(0, 1, 3)$ $F(x, y) = \sum(0, 1, 2)$



$F(x, y) = x + y$ $F(x, y) = x + y'$ $F(x, y) = x' + y$ $F(x, y) = x' + y'$

Three-Variable K-Map

- A three-variable K-map is shown in Fig.
- There are eight minterms for three binary variables; therefore, the map consists of eight squares.
- Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code.
- The characteristic of this sequence is that only one bit changes in value from one adjacent column to the next.
- The map drawn in part (b) is marked with numbers in each row and each column to show the relationship between the squares and the three variables.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

x \ yz	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

y (11, 10) z (01, 11) x (1)

(b)

Example: Simplify the Boolean functions

***F*(*A*, *B*, *C*) = ∑(0, 4)**

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**0**
1	**1**	**0**	**0**	**0**

$$F(A,B,C) = \overline{B}\,\overline{C}$$

***F*(*A*, *B*, *C*) = ∑(1, 3)**

A \ BC	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**

$$F(A,B,C) = \overline{A}\,C$$

***F*(*A*, *B*, *C*) = ∑(4, 5)**

A \ BC	00	01	11	10
0	**0**	**0**	**0**	**0**
1	**1**	**1**	**0**	**0**

$$F(A,B,C) = A\,\overline{B}$$

Example: Simplify the Boolean functions

$F(A, B, C) = \sum(0, 2)$

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**

$$F(A, B, C) = \overline{A}\,\overline{C}$$

$F(A, B, C) = \sum(4, 6)$

A \ BC	00	01	11	10
0	**0**	**0**	**0**	**0**
1	**1**	**0**	**0**	**1**

$$F(A, B, C) = A\,\overline{C}$$

Example: Simplify the Boolean functions

***F*(*A*, *B*, *C*) = ∑(0, 1, 2, 3)**

A \ BC	00	01	11	10
0	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**

$$F(A,B,C) = \overline{A}$$

***F*(*A*, *B*, *C*) = ∑(0, 1, 4, 5)**

A \ BC	00	01	11	10
0	**1**	**1**	**0**	**0**
1	**1**	**1**	**0**	**0**

$$F(A,B,C) = \overline{B}$$

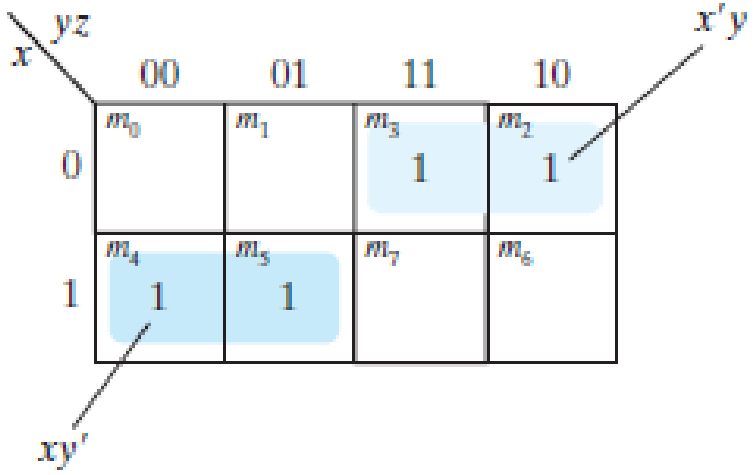
***F*(*A*, *B*, *C*) = ∑(0, 2, 4, 6)**

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**1**
1	**1**	**0**	**0**	**1**

$$F(A,B,C) = \overline{C}$$

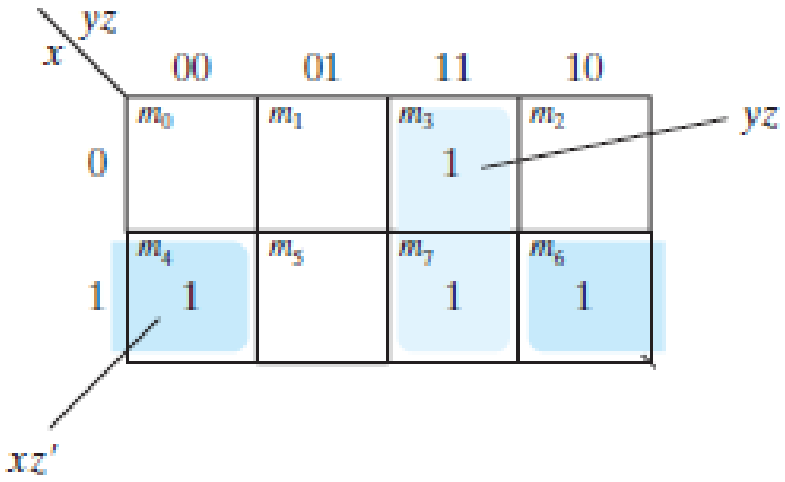
Example: Simplify the Boolean functions

$F(x, y, z) = \sum(2, 3, 4, 5)$



$F(x, y, z) = x'y + xy'$

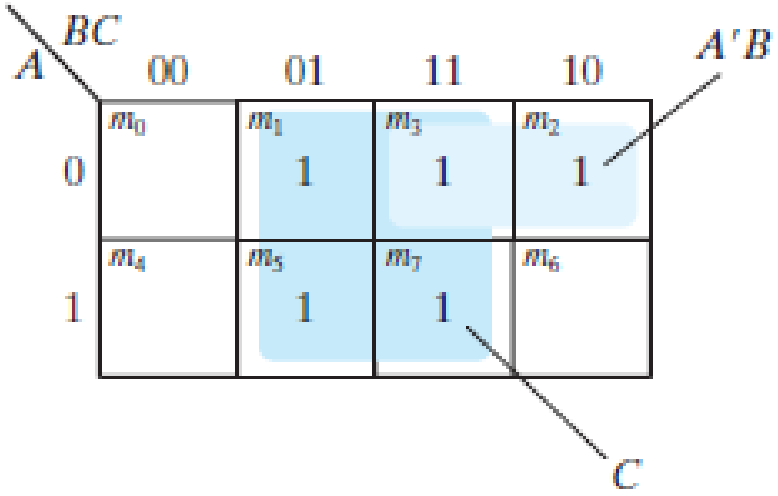
$F(x, y, z) = \sum(3, 4, 6, 7)$



$F(x, y, z) = xz' + yz$

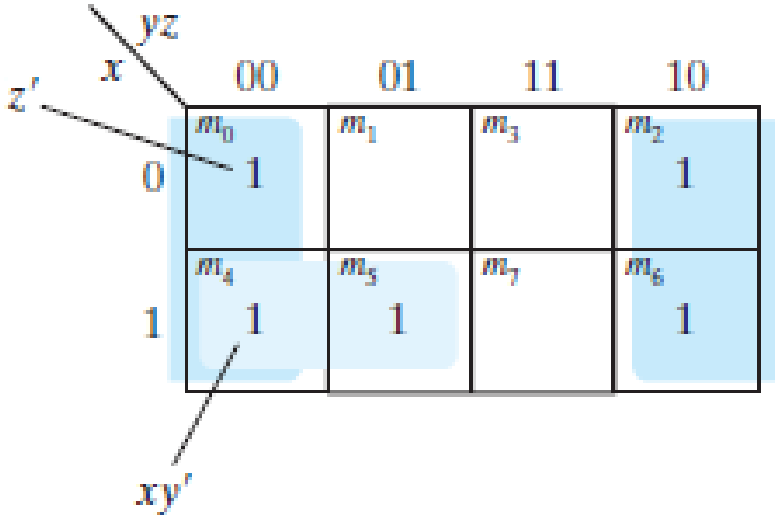
Example: Simplify the Boolean functions

$F(A, B, C) = \sum(1, 2, 3, 5, 7)$



$F(A, B, C) = C + A'B$

$F(x, y, z) = \sum(0, 2, 4, 5, 6)$



$F(x, y, z) = z' + xy'$

Four-Variable K-Map

- The map for Boolean functions of four binary variables (*w*, *x*, *y*, *z*) is shown in Fig.
- In Fig. (a) are listed the 16 minterms and the squares assigned to each.
- In Fig. (b), the map is redrawn to show the relationship between the squares and the four variables.
- The rows and columns are numbered in a Gray code sequence, with only one digit changing value between two adjacent rows or columns.
- The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

wx \ yz	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

y (00, 01, 11, 10) · x (01, 11) · w (11, 10) · z (01, 11)

(b)

The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:

- One square represents one minterm, giving a term with four literals.
- Two adjacent squares represent a term with three literals.
- Four adjacent squares represent a term with two literals.
- Eight adjacent squares represent a term with one literal.
- Sixteen adjacent squares produce a function that is always equal to 1.

Example: Simplify the Boolean functions

***F*(*A*, *B*, *C*, *D*) = ∑(13, 15)**

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	0	0	0

$$F(A,B,C,D) = ABD$$

***F*(*A*, *B*, *C*, *D*) = ∑(5, 13)**

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	0	0	0

$$F(A,B,C,D) = B\overline{C}D$$

Example: Simplify the Boolean functions

***F*(*A*, *B*, *C*, *D*) = ∑(4, 6)**

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	0	0	0	0
10	0	0	0	0

$$F(A,B,C,D) = \overline{A}B\overline{D}$$

***F*(*A*, *B*, *C*, *D*) = ∑(0, 8)**

AB \ CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	0

$$F(A,B,C,D) = \overline{B}\,\overline{C}\,\overline{D}$$

Example: Simplify the Boolean functions

$F(A, B, C, D) = \sum(4, 5, 6, 7)$

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

$$F(A, B, C, D) = \overline{A}B$$

$F(A, B, C, D) = \sum(3, 7, 11, 15)$

AB \ CD	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

$$F(A, B, C, D) = CD$$

Example: Simplify the Boolean functions

$F(A, B, C, D) = \sum(2, 3, 6, 7)$

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	0	0

$$F(A, B, C, D) = \overline{A}C$$

$F(A, B, C, D) = \sum(4, 6, 12, 14)$

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

$$F(A, B, C, D) = B\overline{D}$$

Example: Simplify the Boolean functions

$F(A, B, C, D) = \sum(2, 3, 10, 11)$

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	1	1

$$F(A, B, C, D) = \overline{B}C$$

$F(A, B, C, D) = \sum(0, 2, 8, 10)$

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$$F(A, B, C, D) = \overline{B}\,\overline{D}$$

Example: Simplify the Boolean functions

$F(A, B, C, D) = \sum(4, 5, 6, 7, 12, 13, 14, 15)$ $F(A, B, C, D) = \sum(0, 1, 2, 3, 8, 9, 10, 11)$

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$$F(A, B, C, D) = B$$

AB \ CD	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

$$F(A, B, C, D) = \overline{B}$$

Example: Simplify the Boolean functions

$F(A, B, C, D) = \sum(1, 3, 5, 7, 9, 11, 13, 15)$

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$$F(A, B, C, D) = D$$

$F(A, B, C, D) = \sum(0, 2, 4, 6, 8, 10, 12, 14)$

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$$F(A, B, C, D) = \overline{D}$$

Example: Simplify the following Boolean functions, using four-variable maps:

$F(A, B, C, D) = \sum (0, 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 14)$

$F(A, B, C, D) = \sum (1, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15)$

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	1	0	0	1
10	1	1	1	1

$F(A, B, C, D) = B' + D'$

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	1
10	1	1	0	0

$F(A, B, C, D) = B + A'D + AC'$

Example: Simplify the following Boolean functions, using four-variable maps:

$F(A, B, C, D) = \sum (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$

AB \ CD	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	1	1	0	0
10	1	1	0	1

$F(A, B, C, D) = B'D' + A'D + AC'$

$F(P, Q, R, S) = \sum (0, 1, 3, 4, 5, 6, 7, 13, 15)$

PQ \ RS	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	1	1	0
10	0	0	0	0

$F(P, Q, R, S) = P'R' + P'S + P'Q + QS$

Example: Simplify the following Boolean functions, using four-variable maps:

$F(w, x, y, z) = \sum (1, 4, 5, 6, 12, 14, 15)$ $F(w, x, y, z) = \sum (0, 2, 4, 6, 9, 10, 11, 12, 14)$

wx \ yz	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	0	1	1
10	0	0	0	0

wx \ yz	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	0	1	1	1

$F(w, x, y, z) = xz' + w'y'z + wxy$ $F(w, x, y, z) = w'z' + xz' + yz' + wx'z$

Example: Simplify the following Boolean functions, using four-variable maps:

$F(A, B, C, D) = \sum (3, 4, 5, 7, 9, 13, 14, 15)$ $F(A, B, C, D) = \sum (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

AB \ CD	00	01	11	10
00	0	0	1	0
01	1	1	1	0
11	0	1	1	1
10	0	1	0	0

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	0	1	1	0
10	1	0	0	1

$F(A, B, C, D) = BD + A'CD + AC'D + A'BC' + ABC$ $F(A, B, C, D) = B'D' + BD + A'B$

Here BD is a **Redundant group**.

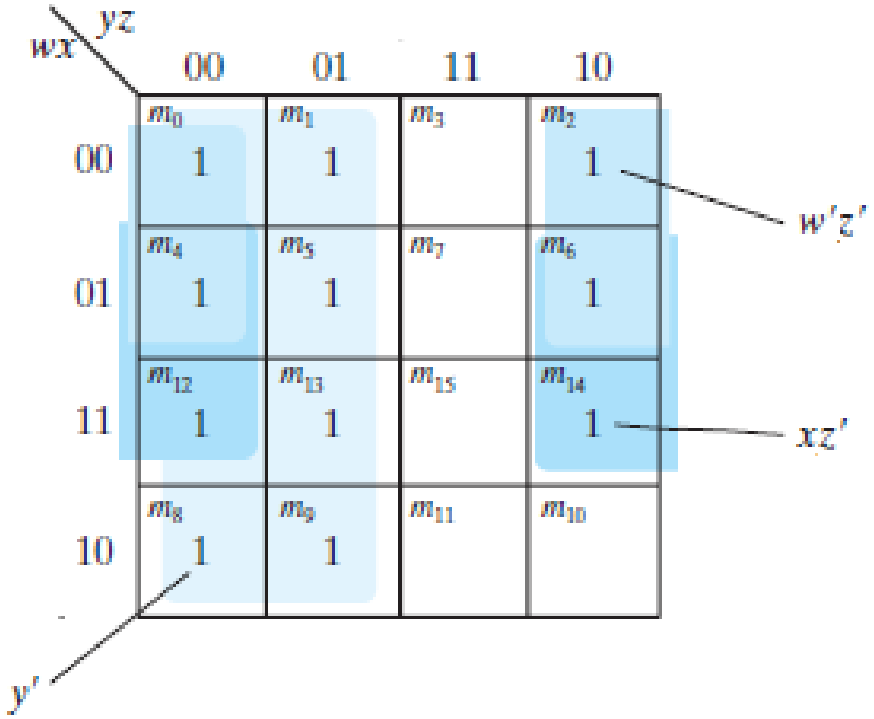
or

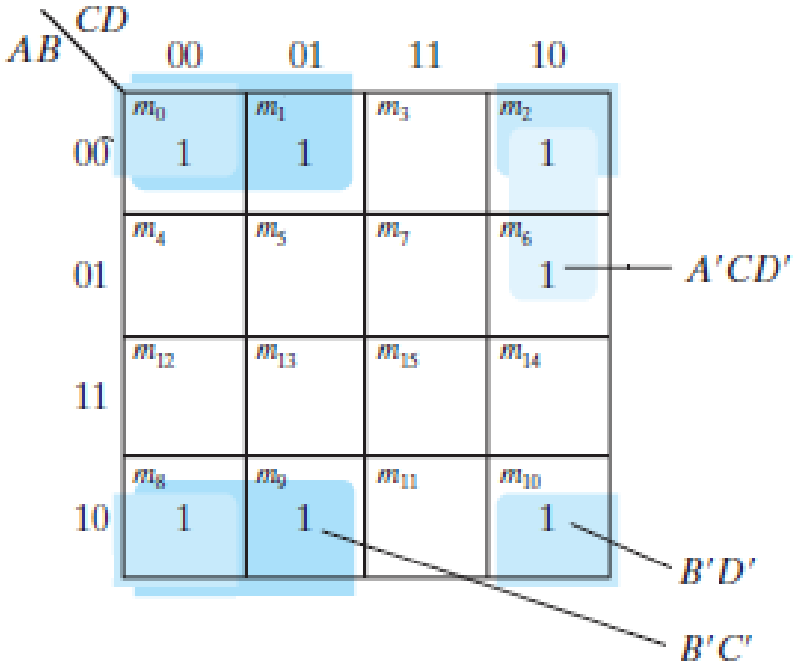
$F(A, B, C, D) = A'CD + AC'D + A'BC' + ABC$ $F(A, B, C, D) = B'D' + BD + A'D'$

Example: Simplify the Boolean functions

$F(w, x, y, z) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$F(A, B, C, D) = \sum (0, 1, 2, 6, 8, 9, 10)$





$F(w, x, y, z) = y' + w'z' + xz'$

$F(A, B, C, D) = B'D' + B'C' + A'CD'$

Prime Implicants

- In choosing adjacent squares in a map, we must ensure that

1. all the minterms of the function are covered when we combine the squares,
2. the number of terms in the expression is minimized, and
3. there are no redundant terms (i.e., minterms already covered by other terms).

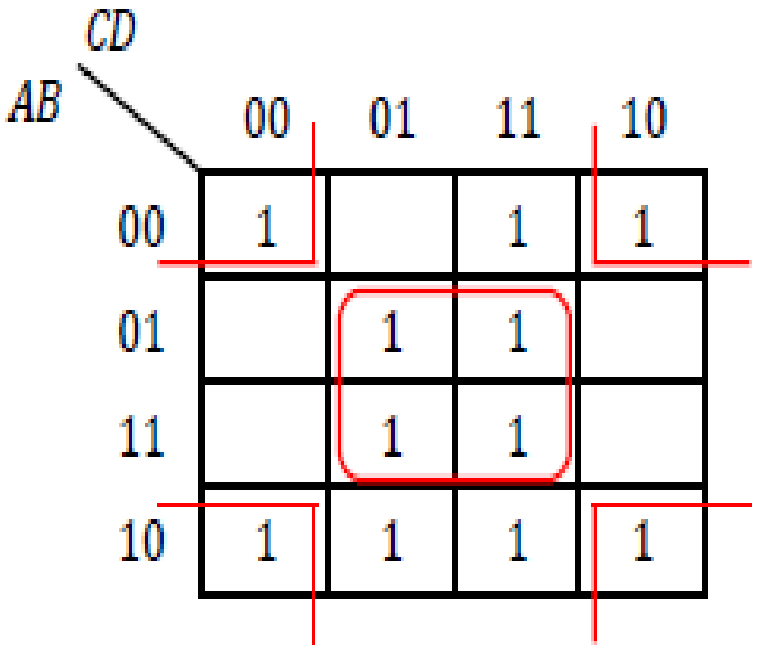
- Sometimes there may be two or more expressions that satisfy the simplification criteria.

- The procedure for combining squares in the map may be made more systematic if we understand the meaning of two special types of terms.
- A *prime implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map.
- If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential*.

Consider the following four-variable Boolean function:

$F(A, B, C, D) = \sum (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

- There is only one way to include minterm m_0 within four adjacent squares. These four squares define the term *B'D'*.
- Similarly, there is only one way that minterm m_5 can be combined with four adjacent squares, and this gives the second term *BD*.
- The two essential prime implicants cover eight minterms.



Essential prime implicants *B'D'* and *BD*

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

$F(A, B, C, D) = B'D' + BD + CD + AB'$

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

$F(A, B, C, D) = B'D' + BD + CD + AD$

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

$F(A, B, C, D) = B'D' + BD + B'C + AB'$

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

$F(A, B, C, D) = B'D' + BD + B'C + AD$

- Figure shows all possible ways that the three minterms (m_3, m_9, and m_{11}) can be covered with prime implicants.
- Minterm m_3 can be covered with either prime implicant *CD* or prime implicant *B'C*.
- Minterm m_9 can be covered with either *AB'* or *AD*.
- Minterm m_{11} is covered with any one of the four prime implicants.
- The simplified expression is obtained from the logical sum of the two essential prime implicants and any two prime implicants that cover minterms m_3, m_9, and m_{11}.
- There are four possible ways that the function can be expressed with four product terms of two literals each:

$$F = B'D' + BD + CD + AB'$$

$$= B'D' + BD + CD + AD$$

$$= B'D' + BD + B'C + AB'$$

$$= B'D' + BD + B'C + AD$$

Find all the prime implicants for the following Boolean functions, and determine which are essential:

$F(A, B, C, D) = \sum (0, 4, 5, 10, 11, 13, 15)$

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	1	1	0
10	0	0	1	1

F = A'C'D' + AB'C + BC'D + ABD

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	1	1	0
10	0	0	1	1

F = A'C'D' + AB'C + BC'D + ACD

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	1	1	0
10	0	0	1	1

F = A'C'D' + AB'C + A'BC' + ABD

- Essential prime implicants *A'C'D'* and *AB'C*
- Prime implicants *BC'D*, *ABD*, *ACD* and *A'BC'*

DON’T-CARE CONDITIONS

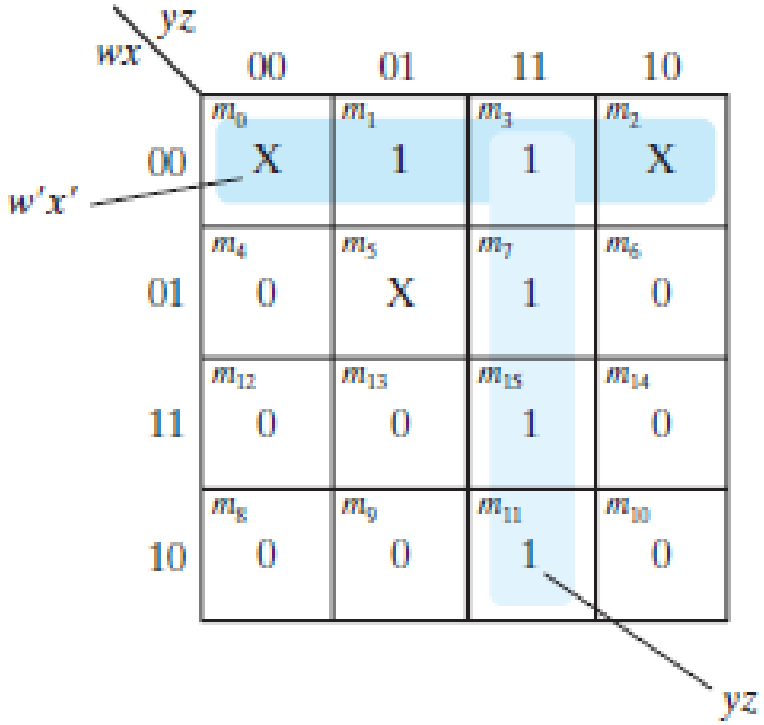
- In some applications the function is not specified for certain combinations of the variables.
- As an example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered to be unspecified.

- Functions that have unspecified outputs for some input combinations are called *incompletely specified functions*.
- For this reason, it is customary to call the unspecified minterms of a function *don’t-care conditions*.
- These don’t-care conditions can be used on a map to provide further simplification of the Boolean expression.

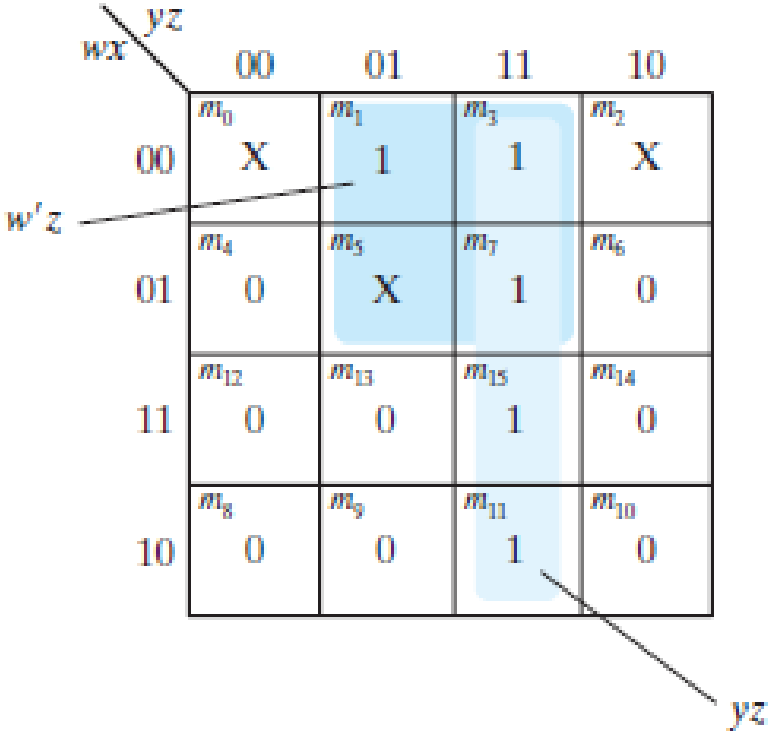
- A don’t-care minterm is a combination of variables whose logical value is not specified.
- To distinguish the don’t-care condition from 1’s and 0’s, an X is used.
- Thus, an X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to *F* for the particular minterm.

- In choosing adjacent squares to simplify the function in a map, the don’t-care minterms may be assumed to be either 0 or 1.
- When simplifying the function, we can choose to include each don’t-care minterm with either the 1’s or the 0’s, depending on which combination gives the simplest expression.

Example: Simplify the Boolean function
$F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$
which has the don't-care conditions
$d(w, x, y, z) = \sum(0, 2, 5)$



$F = yz + w'x'$



$F = yz + w'z$

Example: Simplify the following Boolean function *F*, together with the don't-care conditions *d*.

$F(A, B, C, D) = \sum (1, 5, 6, 12, 13, 14)$

$d(A, B, C, D) = \sum (2, 4)$

AB \ CD	00	01	11	10
00	0	1	0	X
01	X	1	0	1
11	1	1	0	1
10	0	0	0	0

$F(A, B, C, D) = BC' + BD' + A'C'D$

$F(A, B, C, D) = \sum (4, 5, 7, 12, 14, 15)$

$d(A, B, C, D) = \sum (3, 8, 10)$

AB \ CD	00	01	11	10
00	0	0	X	0
01	1	1	1	0
11	1	0	1	1
10	X	0	0	X

$F(A, B, C, D) = AD' + A'BC' + BCD$

Example: Simplify the following Boolean function *F*, together with the don't-care conditions *d*.

$F(A, B, C, D) = \sum (1, 3, 5, 7, 9, 15)$

$d(A, B, C, D) = \sum (4, 6, 12, 13)$

AB \ CD	00	01	11	10
00	0	1	1	0
01	X	1	1	X
11	X	X	1	0
10	0	1	0	0

$F(A, B, C, D) = A'D + BD + C'D$

$F(A, B, C, D) = \sum (1, 3, 4, 5, 8, 10, 11, 15)$

$d(A, B, C, D) = \sum (0, 2, 7, 14)$

AB \ CD	00	01	11	10
00	X	1	1	X
01	1	1	X	0
11	0	0	1	X
10	1	0	1	1

$F(A, B, C, D) = B'D' + A'C' + CD$

PRODUCT-OF-SUMS SIMPLIFICATION

Example: **Simplify the following Boolean functions in POS form.**

$F(x, y, z) = \prod (0, 1, 3, 7)$

x \ yz	00	01	11	10
0	0	0	0	
1			0	

$F'(x, y, z) = x'y' + yz$

$F(x, y, z) = (x + y)(y' + z')$

$F(A, B, C) = \prod (0, 1, 2, 3, 4, 7)$

A \ BC	00	01	11	10
0	0	0	0	0
1	0		0	

$F'(A, B, C) = A' + B'C' + BC$

$F(A, B, C) = A(B + C)(B' + C')$

Example: Simplify the following Boolean functions in POS form.

$F(w, x, y, z) = \prod (0, 4, 6, 7, 8, 12, 13, 14, 15)$

$F(A, B, C, D) = \prod (2, 8, 9, 10, 11, 12, 14)$

wx \ yz	00	01	11	10
00	0			
01	0		0	0
11	0	0	0	0
10	0			

AB \ CD	00	01	11	10
00				0
01				
11	0			0
10	0	0	0	0

$F'(w, x, y, z) = y'z' + xy + wx$

$F(w, x, y, z) = (y + z)(x' + y')(w' + x')$

$F'(A, B, C, D) = AB' + AD' + B'CD'$

$F(A, B, C, D) = (A' + B)(A' + D)(B + C' + D)$

COMBINATIONAL LOGIC

Combinational Circuits;

Analysis Procedure;

Design Procedure;

Adders; Subtractors

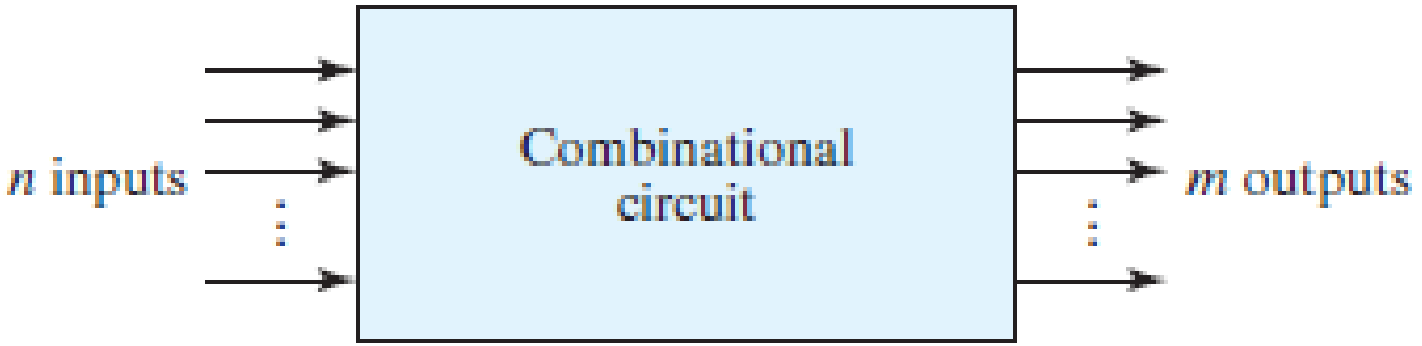
INTRODUCTION

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.

- In contrast, sequential circuits employ storage elements in addition to logic gates.
- Their outputs are a function of the inputs and the state of the storage elements.
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

COMBINATIONAL CIRCUITS

- A combinational circuit consists of an interconnection of logic gates.
- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.
- A block diagram of a combinational circuit is shown in Fig.



- The *n* input binary variables come from an external source; the *m* output variables are produced by the internal combinational logic circuit and go to an external destination.
- Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0.

- For n input variables, there are 2^n possible combinations of the binary inputs.
- For each possible input combination, there is one possible value for each output variable.
- Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.
- A combinational circuit also can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

ANALYSIS PROCEDURE

- The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or possibly, an explanation of the circuit operation.

DESIGN PROCEDURE

- The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

ADDERS

- The most basic arithmetic operation is the addition of two binary digits.
- This simple addition consists of 4 possible operations, namely,

0	0	1	1
+0	+1	+0	+1
			(carry)
0	1	1	1←0

Half Adder

- A combinational circuit that performs the addition of two bits is called a *half adder*.
- A half adder is a combinational circuit with two binary inputs (augend and addend bits) and two binary outputs (sum and carry bits).
- It adds the two inputs (A and B) and produces two outputs, sum (S) and carry (C).

Block diagram

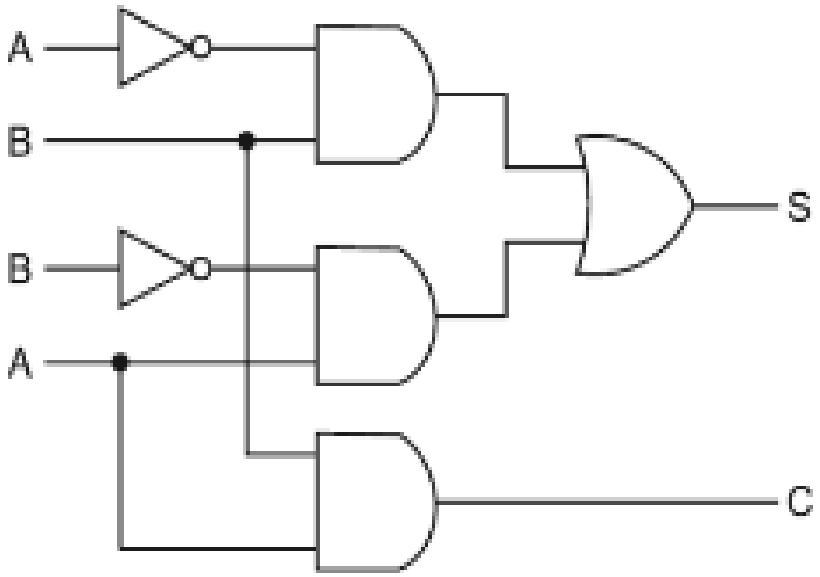
Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

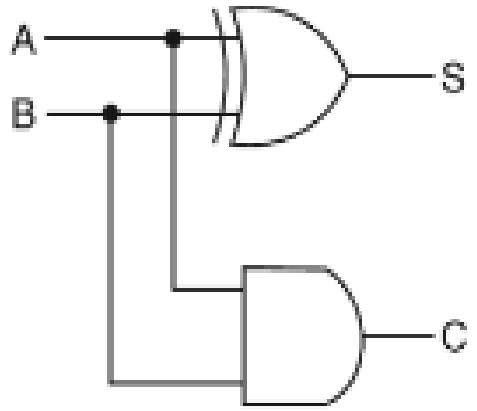
Truth Table

- The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

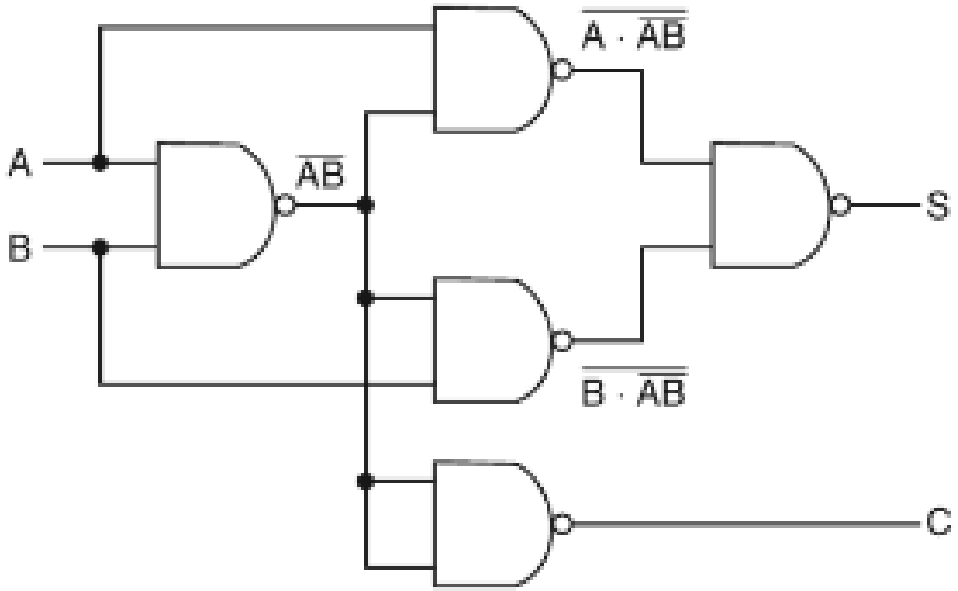




Logic diagrams of half-adder.

Implementation of half adder by using only NAND gates.

$$S = A\bar{B} + \bar{A}B = A\bar{A} + A\bar{B} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{A\bar{B}} + B \cdot \overline{A\bar{B}}$$
$$= \overline{\overline{A \cdot \overline{A\bar{B}}} \cdot \overline{B \cdot \overline{A\bar{B}}}}$$
$$C = AB = \overline{\overline{AB}}$$

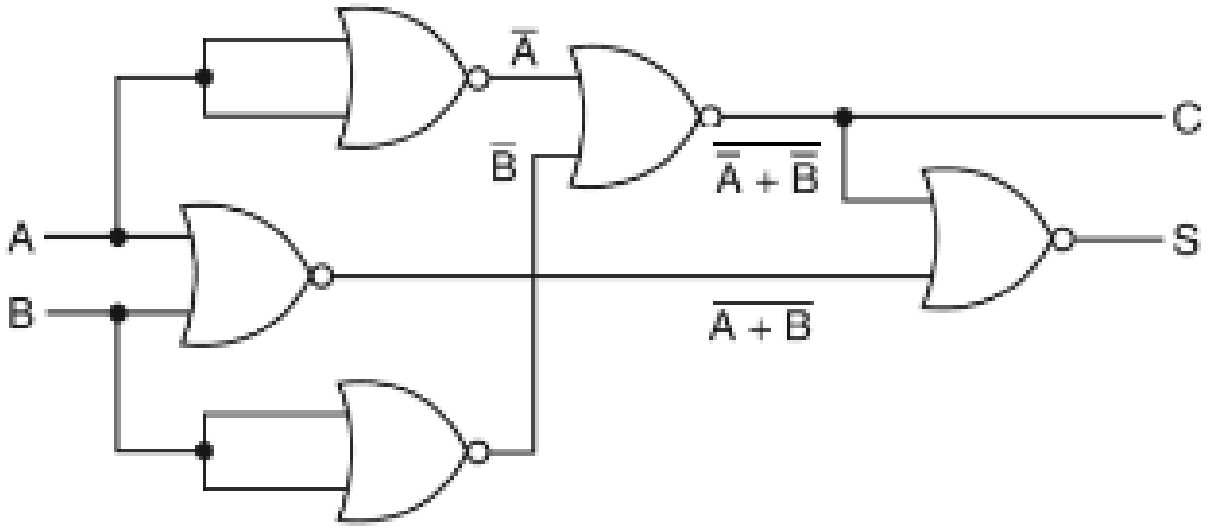


Logic diagram of a half-adder using only 2-input NAND gates.

Implementation of half adder by using only NOR gates.

$$S = A\bar{B} + \bar{A}B = A\bar{A} + A\bar{B} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= (A + B)(\bar{A} + \bar{B})$$
$$= \overline{\overline{A + B} + \overline{\bar{A} + \bar{B}}}$$

$$C = AB = \overline{\overline{AB}} = \overline{\bar{A} + \bar{B}}$$



Logic diagram of a half-adder using only 2-input NOR gates.

Full Adder

- A *full adder* is a combinational circuit that forms the arithmetic sum of three bits (two significant bits and a previous carry).
- It consists of three inputs and two outputs.
- The full adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out}.

Block diagram

Inputs			Outputs	
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

Truth Table

The simplified expressions are

$$S = \sum(1, 2, 4, 7)$$

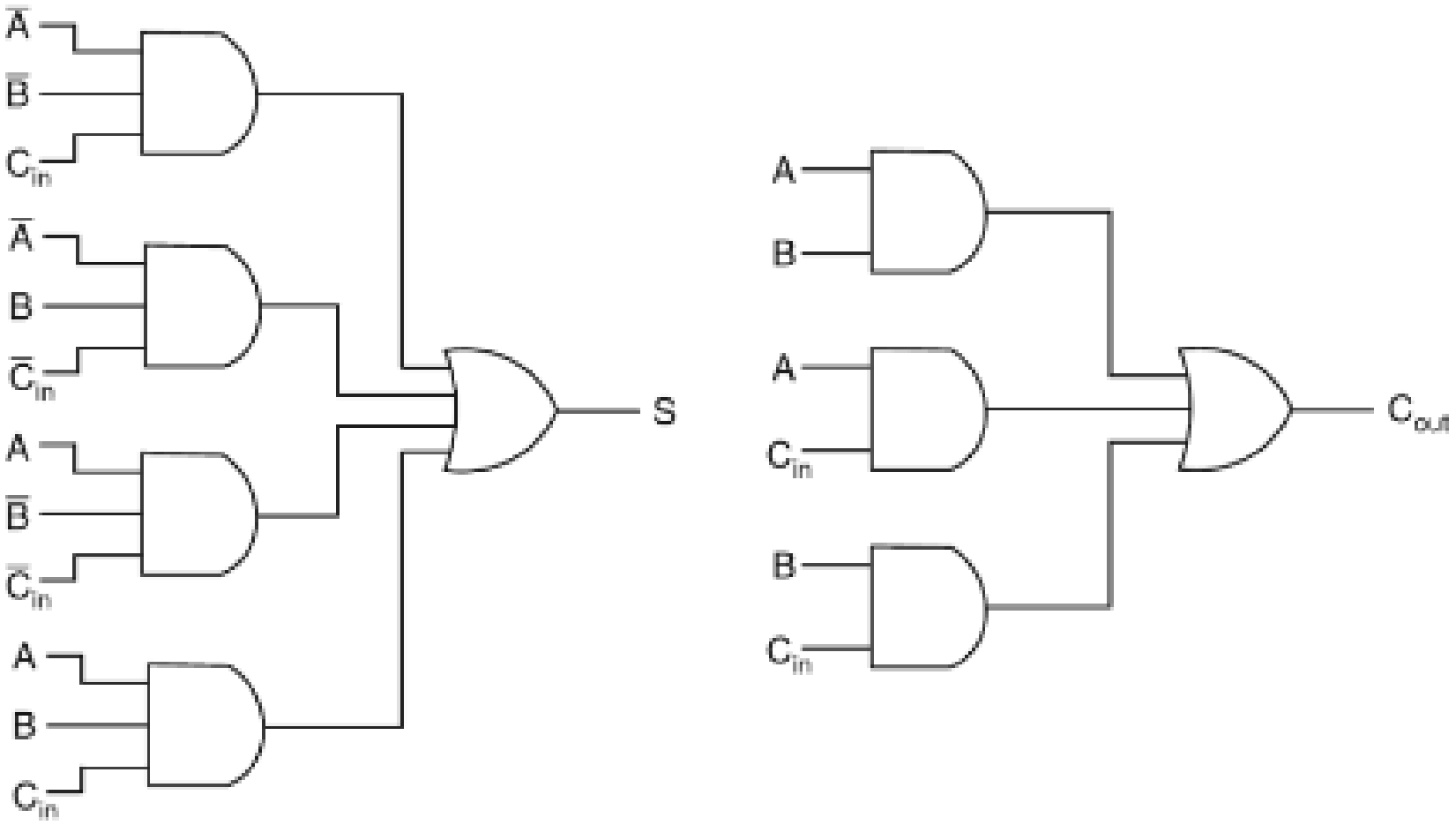
A \ BC_{in}	00	01	11	10
0	m_0	m_1 1	m_3	m_2 1
1	m_4 1	m_5	m_7 1	m_6

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$C_{out} = \sum(3, 5, 6, 7)$$

A \ BC_{in}	00	01	11	10
0	m_0	m_1	m_3 1	m_2
1	m_4	m_5 1	m_7 1	m_6 1

$$C_{out} = AB + AC_{in} + BC_{in}$$



Logic diagrams of full-adder.

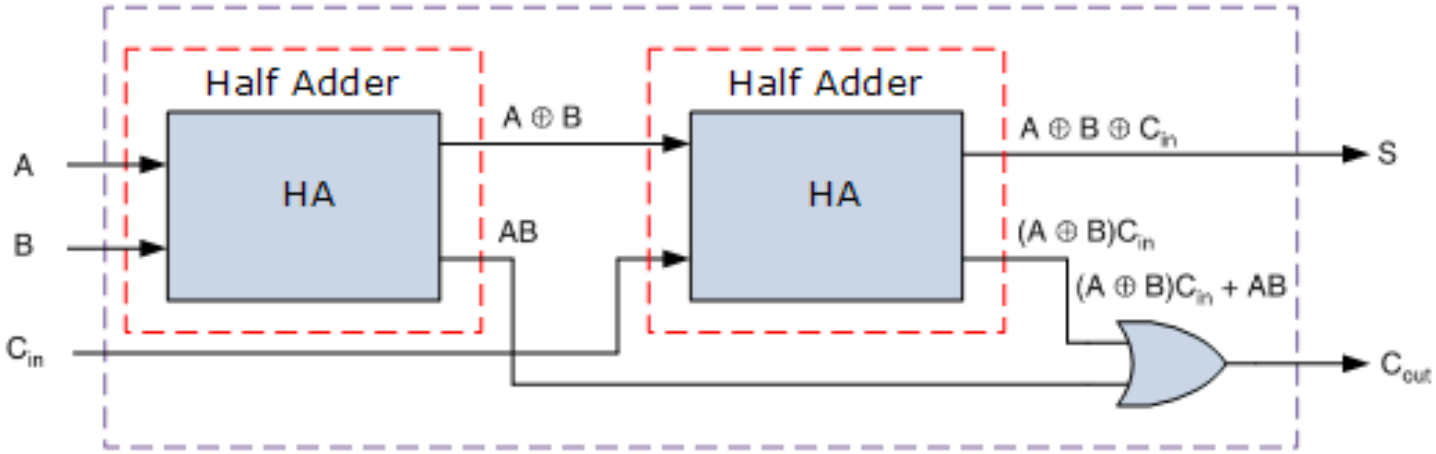
Implementation of full adder with two half adders and an OR gate.

- From the truth table of full adder, the outputs sum and carry are described by

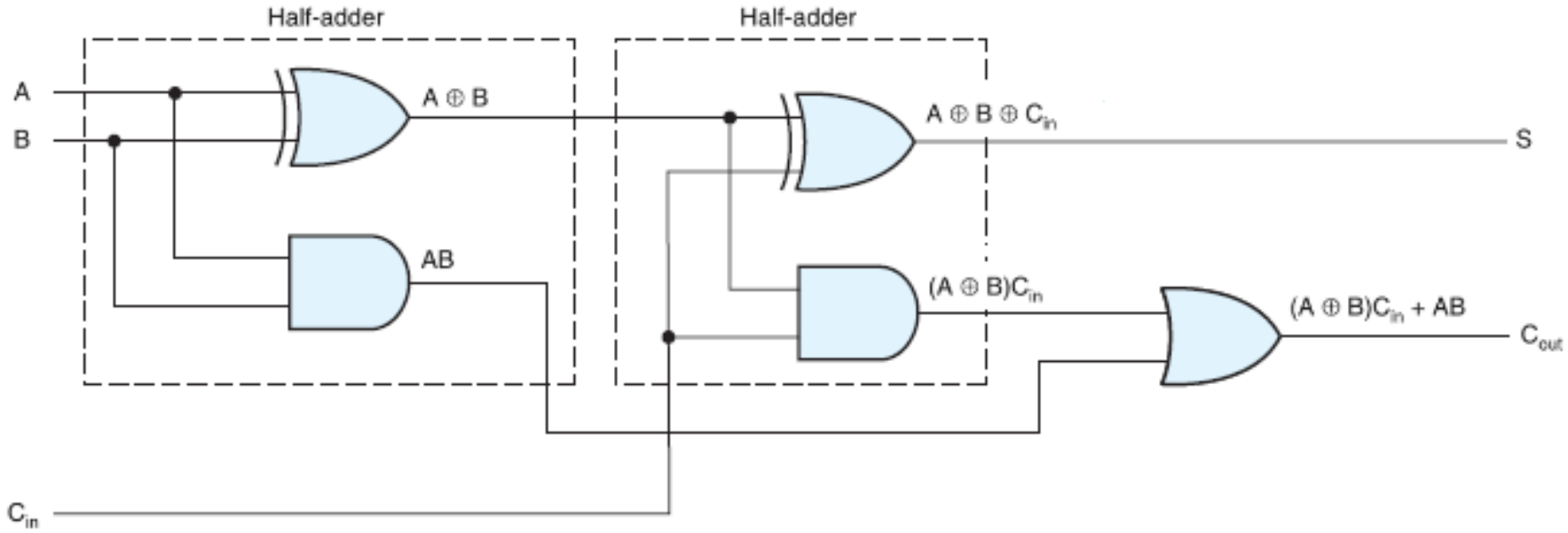
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$
$$= (A\bar{B} + \bar{A}B)\bar{C}_{in} + (AB + \bar{A}\bar{B})C_{in}$$
$$= (A \oplus B)\bar{C}_{in} + \overline{(A \oplus B)}C_{in} = A \oplus B \oplus C_{in}$$

and

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$
$$= (\bar{A}B + A\bar{B})C_{in} + AB(\bar{C}_{in} + C_{in}) = AB + (A \oplus B)C_{in}$$



Block diagram of a full-adder using two half-adders.



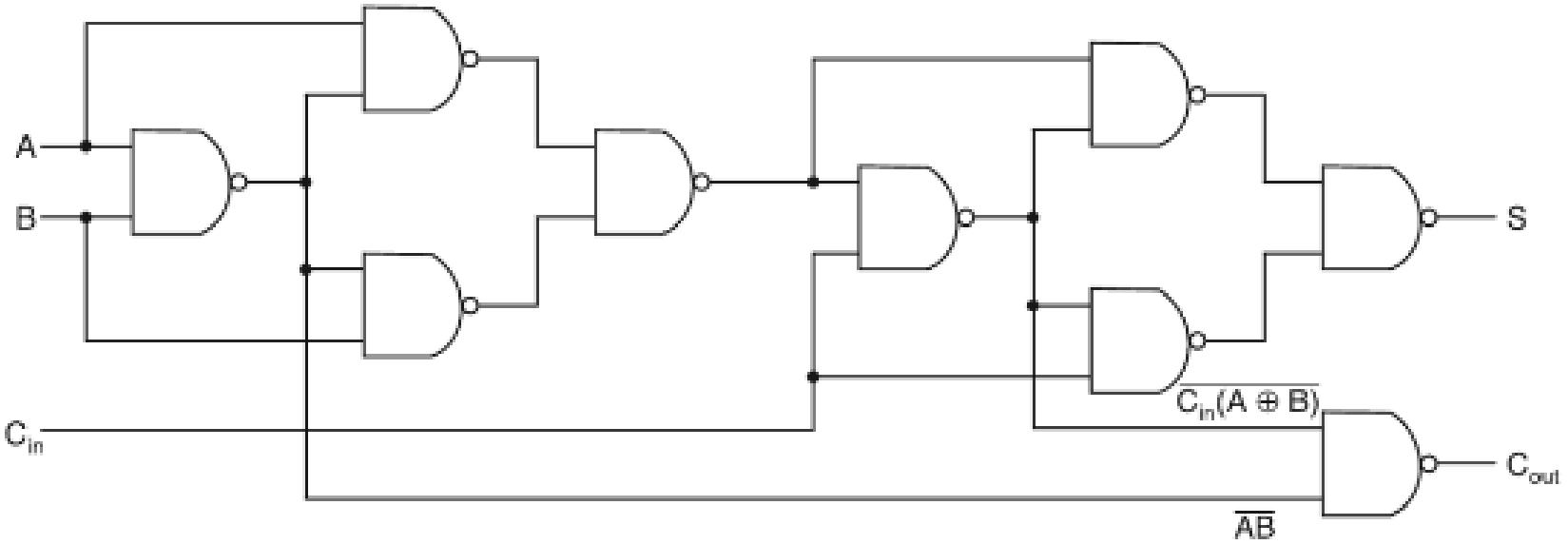
Logic diagram of a full-adder using two half-adders.

Implementation of full adder by using only NAND gates.

We know that $A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$

Then $S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$

$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$



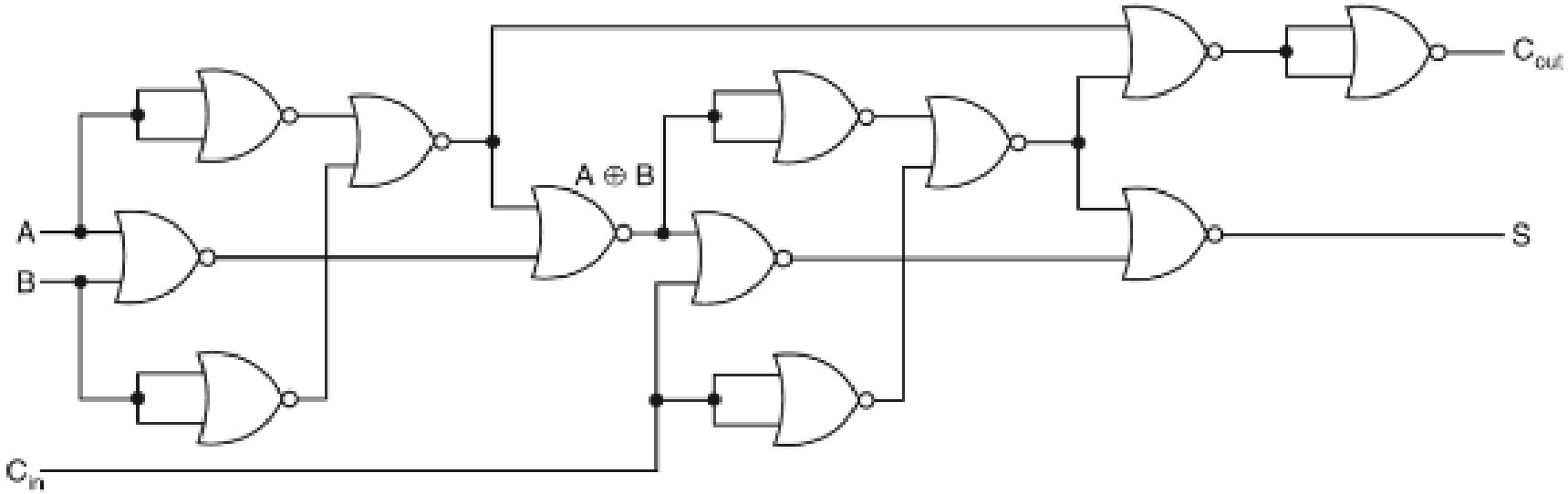
Logic diagram of a full-adder using only 2-input NAND gates.

Implementation of full adder by using only NOR gates.

We know that $A \oplus B = \overline{\overline{(A + B)} + \overline{A} + \overline{B}}$

Then $S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) + C_{in}} + \overline{(A \oplus B) + \overline{C}_{in}}}$

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{A} + \overline{B} + \overline{C_{in} + A \oplus B}} = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{C}_{in} + \overline{A \oplus B}}}$$



Logic diagram of a full-adder using only 2-input NOR gates.

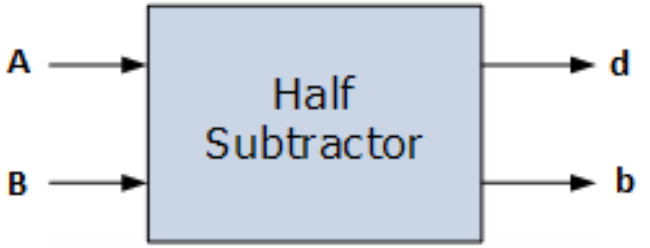
Substractors

- In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit.
- If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position.

			(borrow)
0	1	1	1 → 0
- 0	- 0	- 1	- 1
0	1	0	1

Half Substractor

- A *half subtractor* is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed.
- It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.
- A half subtractor is a combinational circuit with two inputs A and B and two outputs d and b, d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed.



Block diagram

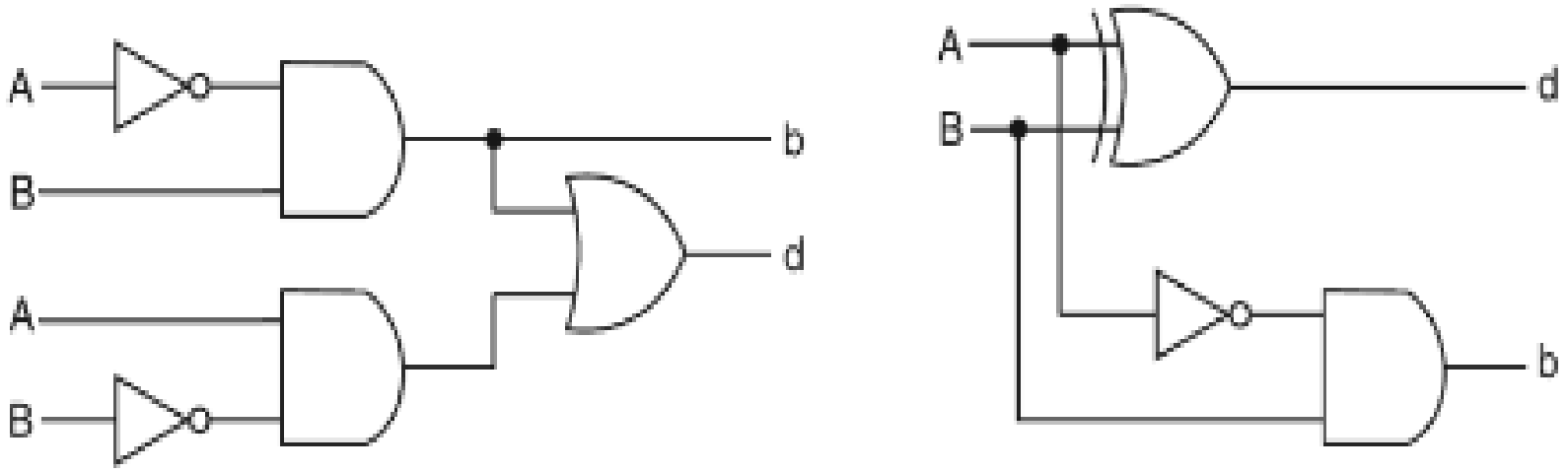
Inputs		Outputs	
A	B	b	d
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Truth Table

- The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

$$d = \overline{A}B + A\overline{B} = A \oplus B$$

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

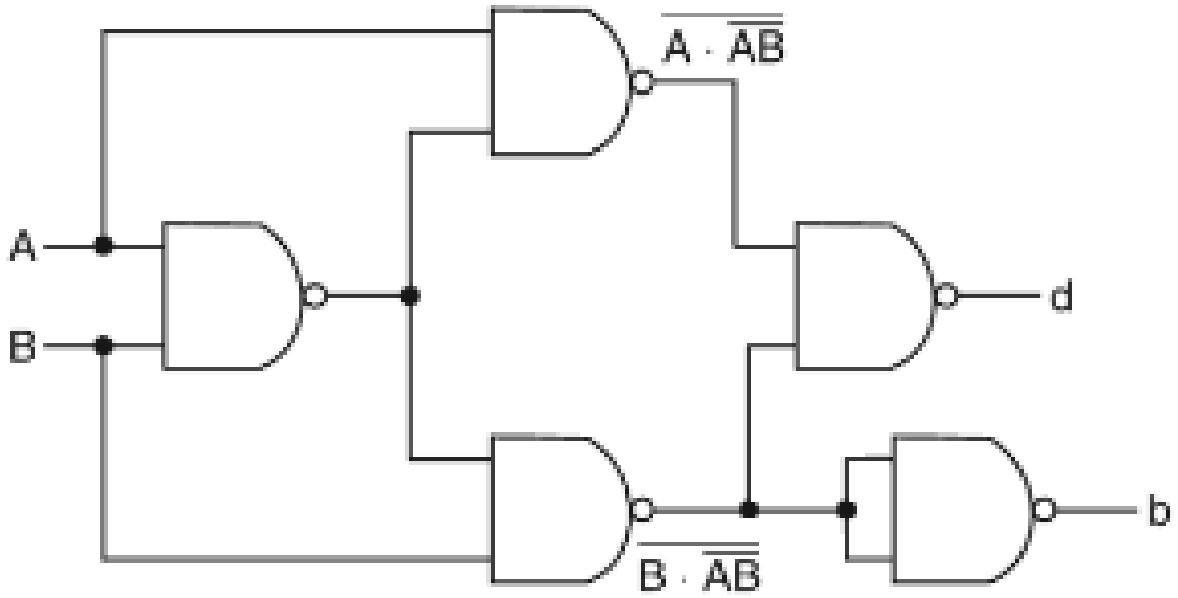


Logic diagrams of a half-subtractor.

Implementation of half substractor by using only NAND gates.

$$d = A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

$$b = \bar{A}B = \bar{A}B + B\bar{B} = B(\bar{A} + \bar{B}) = B(\overline{AB}) = \overline{\overline{B \cdot \overline{AB}}}$$



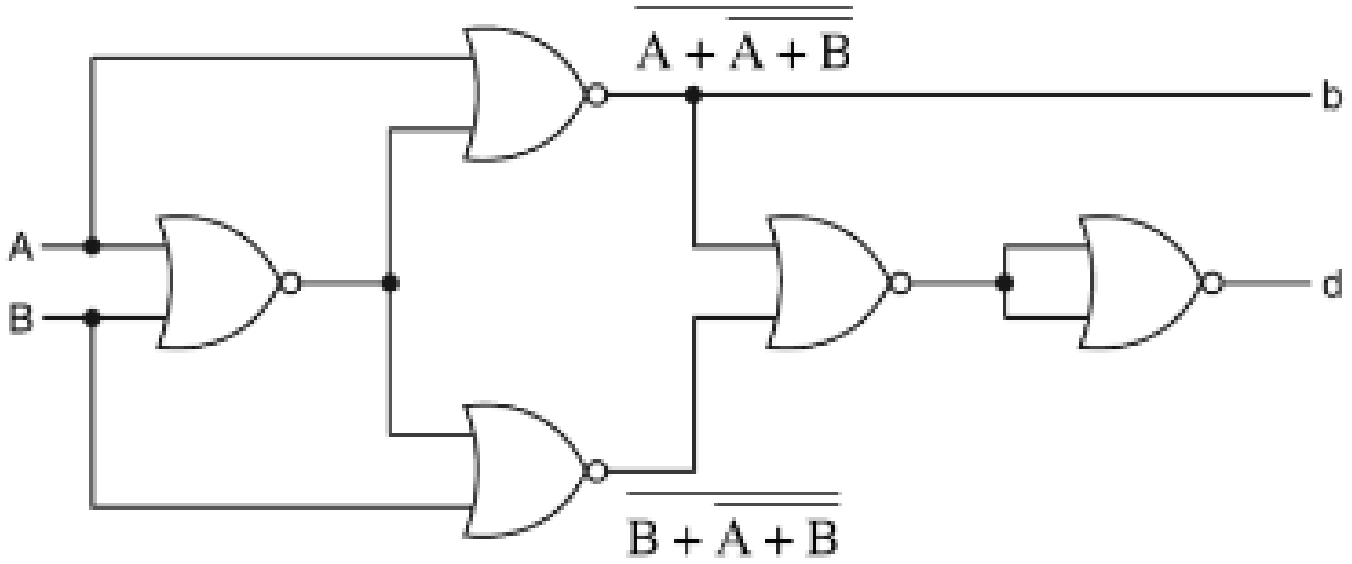
Logic diagram of a half-subtractor using only 2-input NAND gates.

Implementation of half substractor by using only NOR gates.

$$d = A \oplus B = A\bar{B} + \bar{A}B = A\bar{B} + B\bar{B} + A\bar{A} + \bar{A}B$$

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

$$b = \bar{A}B = A\bar{A} + \bar{A}B = \bar{A}(A + B) = \overline{\bar{A}(A + B)} = \overline{A + (\overline{A + B})}$$

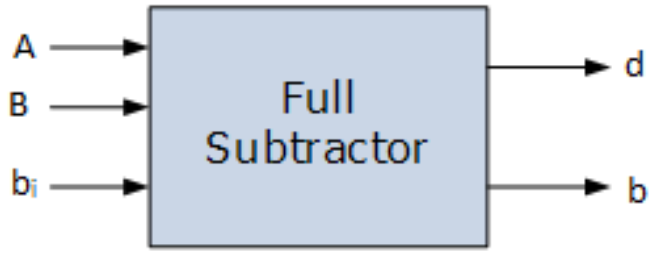


Logic diagram of a half-subtractor using only 2-input NOR gates.

Full Substractor

- The *full substractor* is a combinational circuit which is used to perform subtraction of three input bits.
- It subtracts one bit from another bit, when already there is a borrow from this column for the subtraction in the preceding column, and outputs the difference bit and the borrow bit required from the next column.
- So a full substractor is a combinational circuit with three inputs (A, B, b_i) and two outputs d and b. The two outputs present the difference and output borrow.

The simplified expressions are

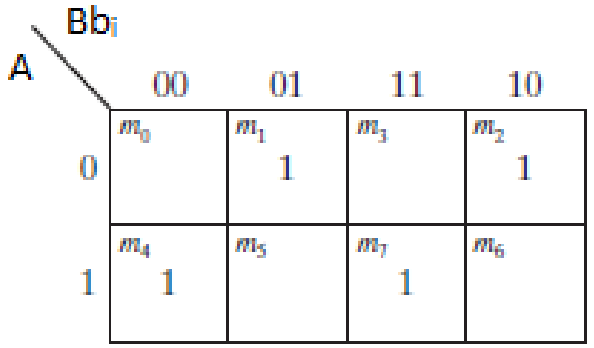


Block diagram

Inputs			Outputs	
A	B	b_i	b	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

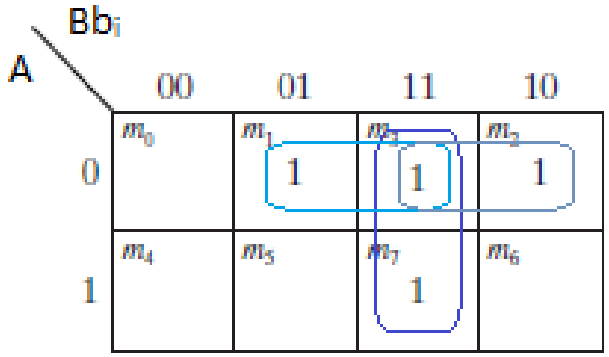
Truth Table

$$d = \sum(1, 2, 4, 7)$$

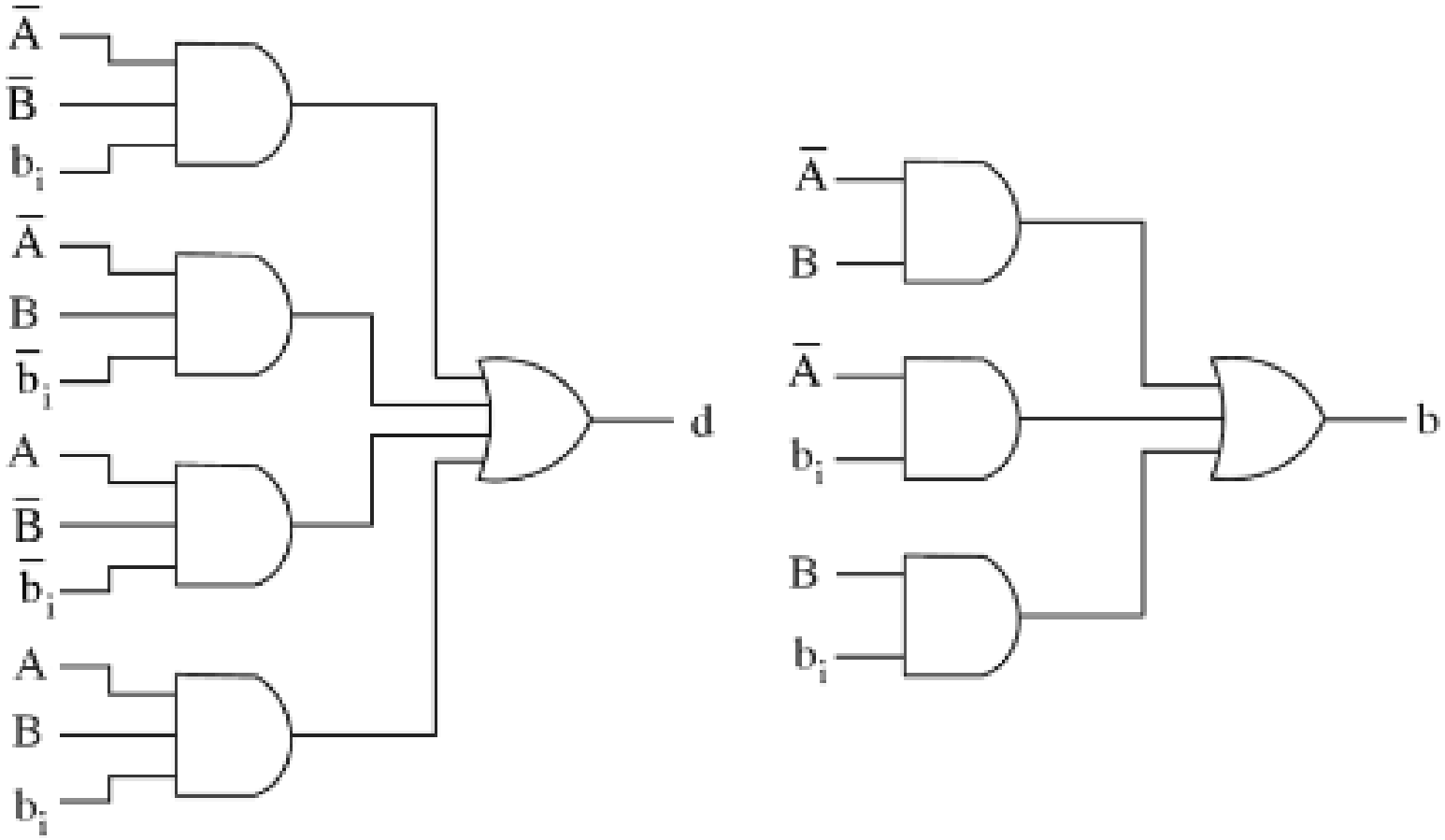


$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$

$$b = \sum(1, 2, 3, 7)$$



$$b = \bar{A}B + \bar{A}b_i + Bb_i$$



Logic diagram of a full-subtractor.

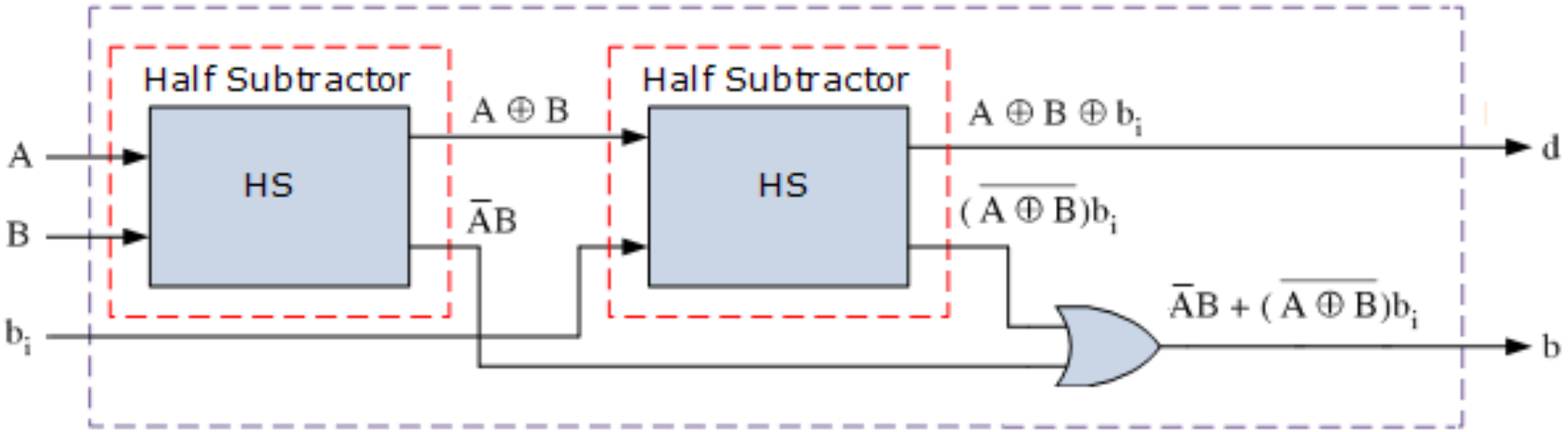
Implementation of full substractor with two substractors and an OR gate.

- From the truth table of full adder, the outputs difference and borrow are described by

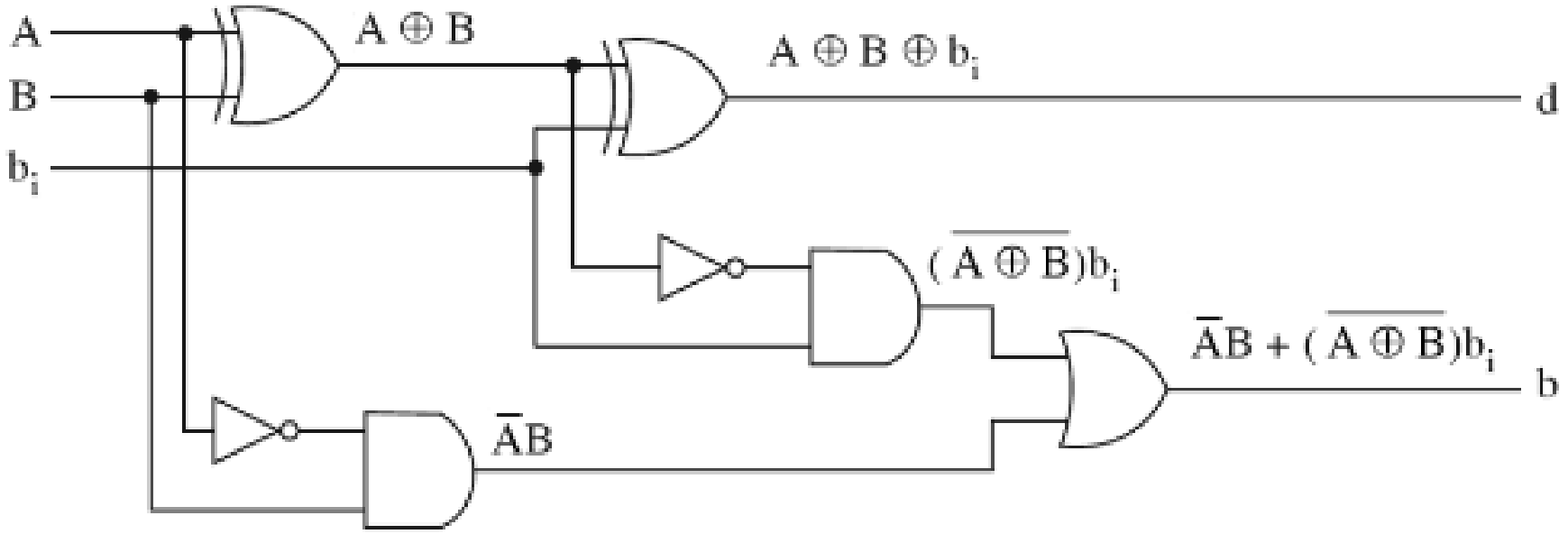
$$\begin{aligned} d &= \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i \\ &= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B) \\ &= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i \end{aligned}$$

and

$$\begin{aligned} b &= \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i \\ &= \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i \\ &= \bar{A}B + (\overline{A \oplus B})b_i \end{aligned}$$



Block diagram of a full-subtractor using two half-subtractor.



Logic diagram of a full-subtractor using two half-subtractor.

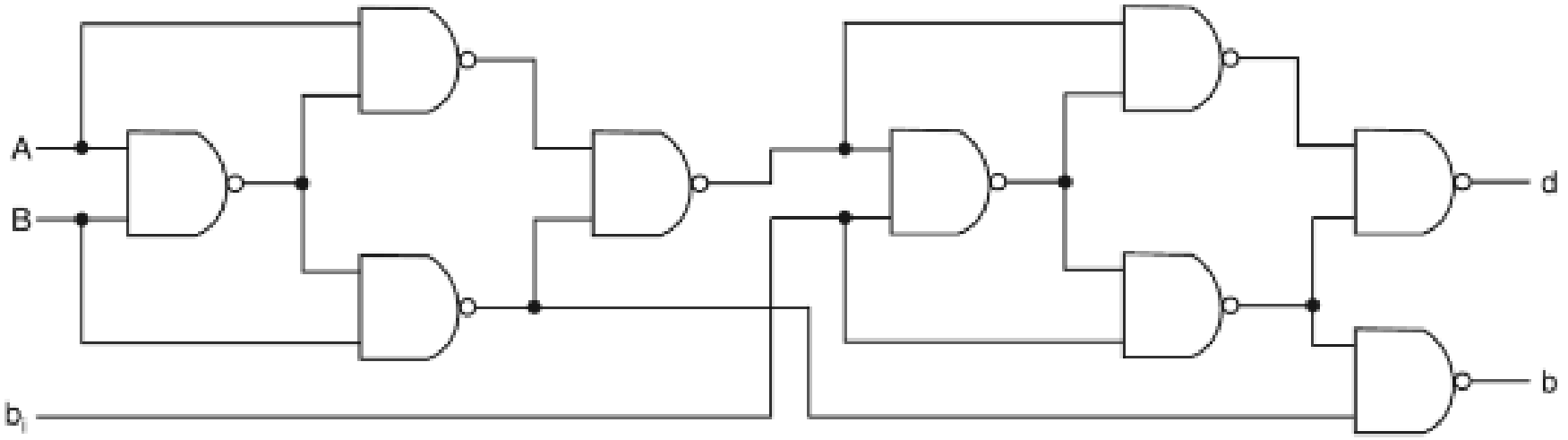
Implementation of full subtractor by using only NAND gates.

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)\overline{(A \oplus B)b_i}} \cdot \overline{b_i(A \oplus B)b_i}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\bar{A}B + b_i(\overline{A \oplus B})}$$

$$= \overline{\bar{A}B \cdot \overline{b_i(A \oplus B)}} = \overline{B(\bar{A} + \bar{B}) \cdot \overline{b_i(\bar{b}_i + (A \oplus B))}}$$

$$= \overline{B \cdot \overline{AB} \cdot \overline{b_i[b_i \cdot (A \oplus B)]}}$$



Logic diagram of a full-subtractor using only 2-input NAND gates.

Implementation of full substractor by using only NOR gates.

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$

$$= \overline{(A \oplus B)b_i + (\overline{A \oplus B})\bar{b}_i}$$

$$= \overline{[(A \oplus B) + (\overline{A \oplus B})\bar{b}_i][b_i + (\overline{A \oplus B})\bar{b}_i]}$$

$$= \overline{(A \oplus B) + \overline{(A \oplus B) + b_i} + b_i + \overline{(A \oplus B) + b_i}}$$

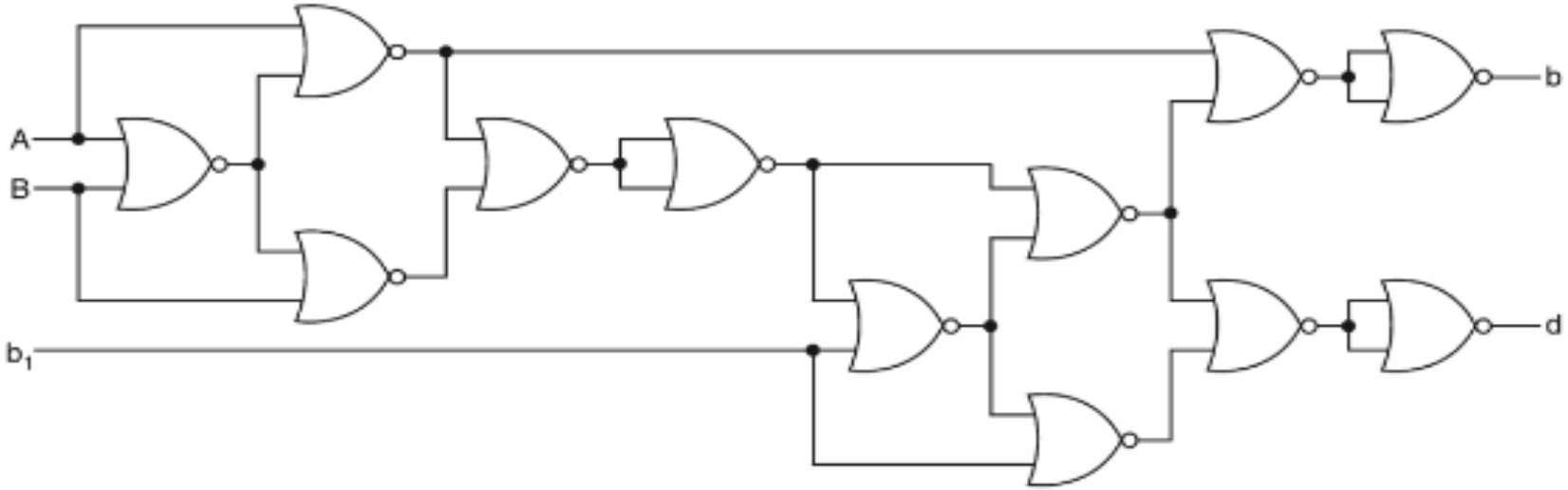
$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

$$= \overline{A + (\overline{A + B})} + \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}}$$

$$= \overline{\overline{A + (\overline{A + B})} + \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}}}$$

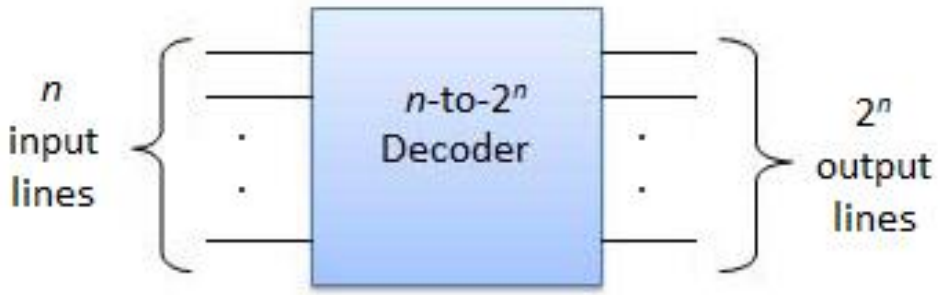


Logic diagram of a full subtractor using only 2-input NOR gates.

Decoders; Encoders; Multiplexers; De-multiplexers

Decoders

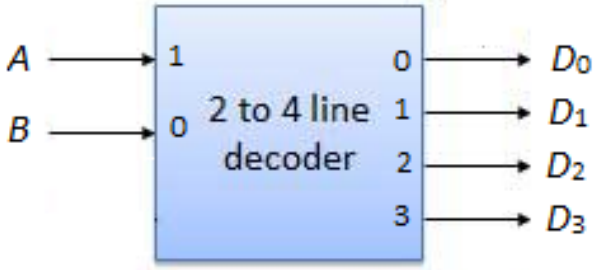
- A *decoder* is a combinational circuit that converts binary information from *n* input lines to a maximum of 2^n unique output lines.
- If the *n*-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.



- The decoders presented here are called *n*-to-*m*-line decoders, where $m \leq 2^n$.
- Their purpose is to generate the 2^n (or fewer) minterms of *n* input variables.
- Each combination of inputs will assert a unique output.
- The name *decoder* is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.

2-to-4-Line Decoder

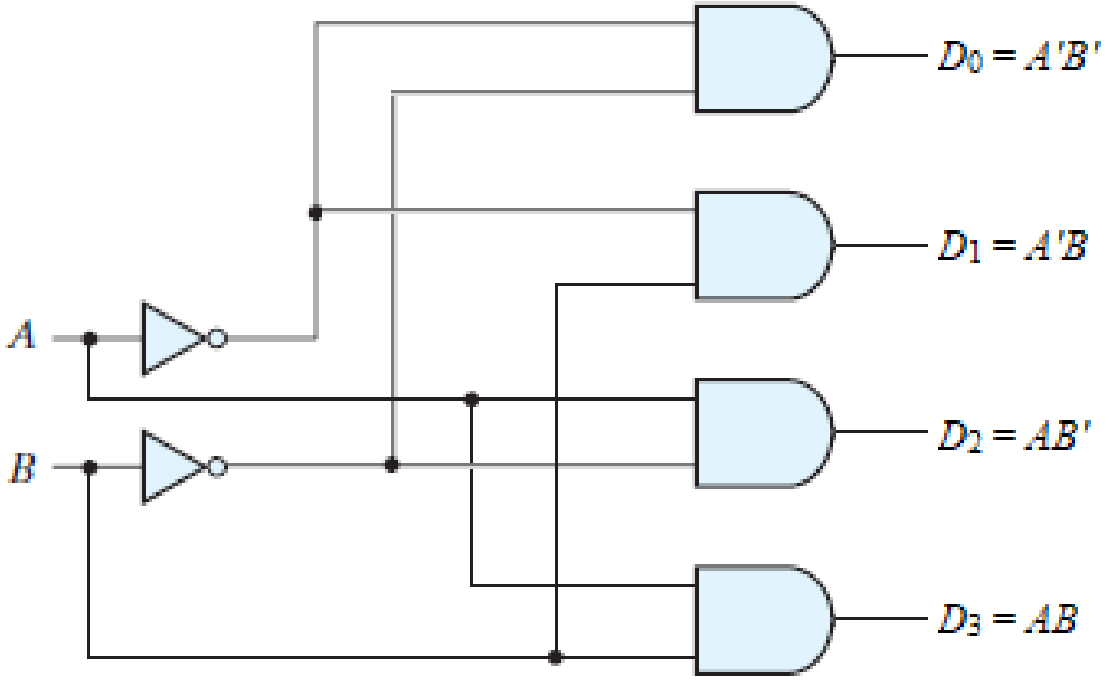
- In 2-to-4-line decoder circuit, the two inputs are decoded into four outputs, each representing one of the minterms of the two input variables.
- The two inputs are represented by A and B and outputs are represented by D_0, D_1, D_2 and D_3.



Block diagram

Inputs		Outputs			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

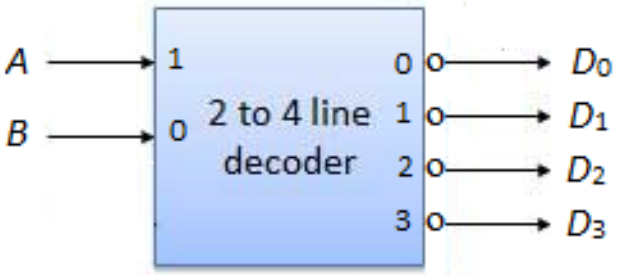
Truth table



Logic diagram

2-to-4-Line Active Low Decoder

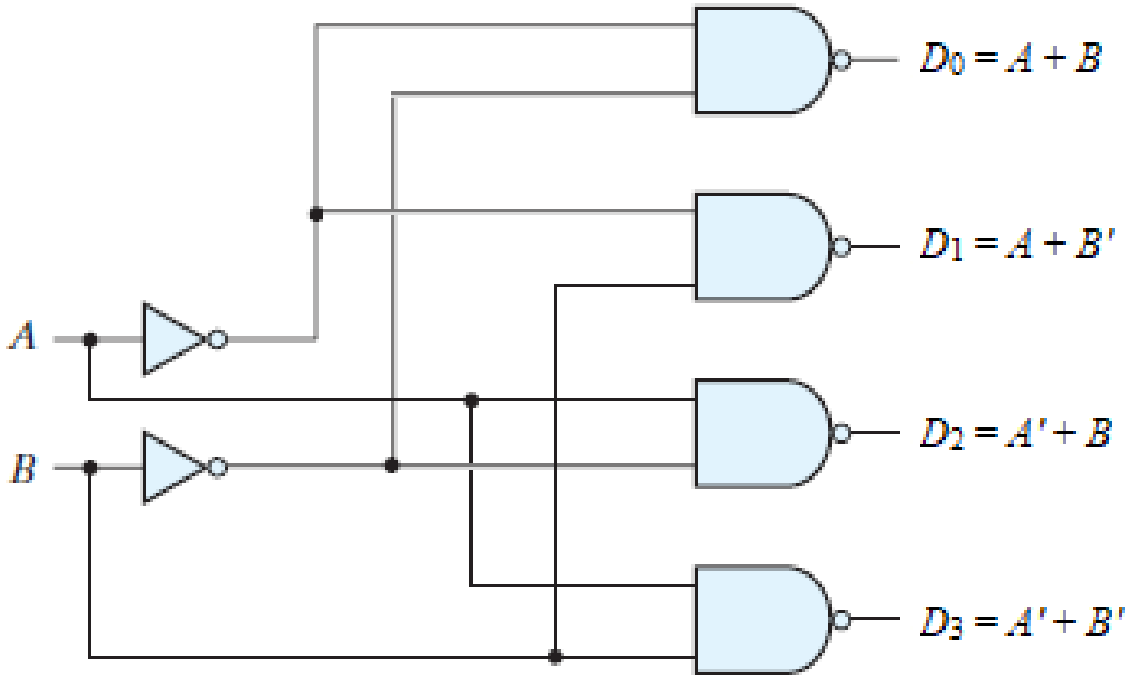
- Some decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form.
- The use of NAND gates as the decoding element, results in an active - “LOW” output while the rest will be “HIGH”.



Block diagram

Inputs		Outputs			
A	B	D_0	D_1	D_2	D_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

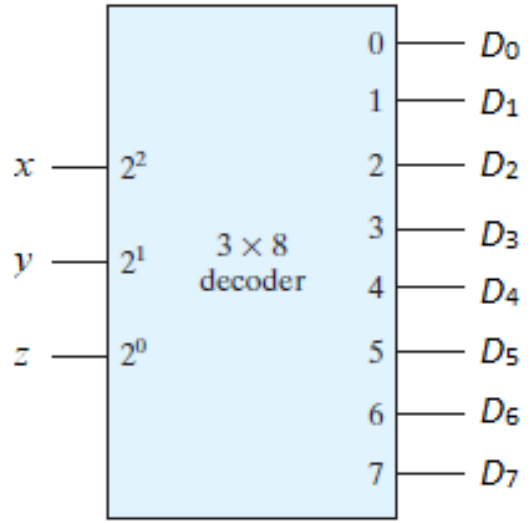
Truth table



Logic diagram

3-to-8-Line Decoder

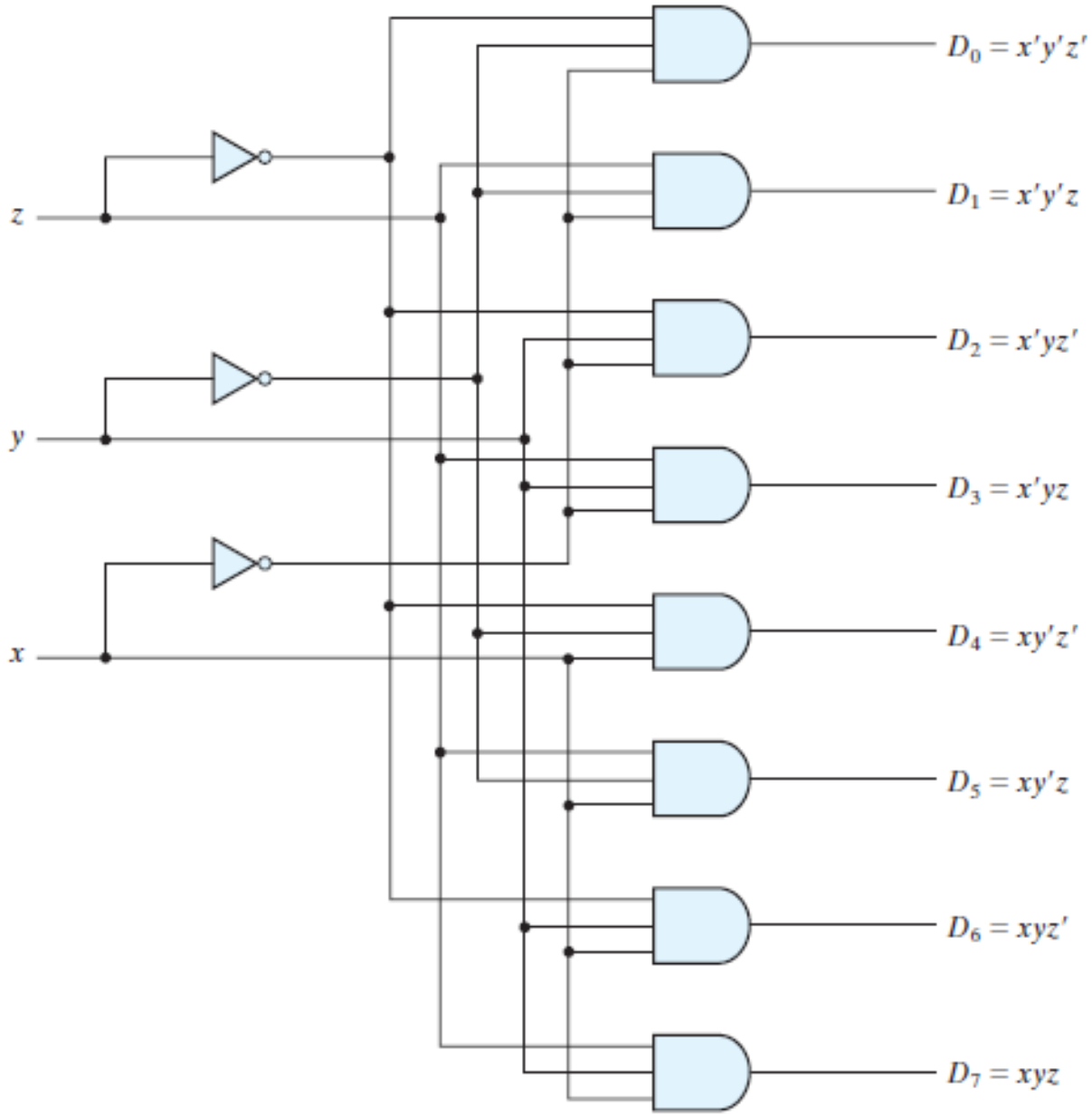
- In 3-to-8-line decoder circuit, the three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.
- A particular application of this decoder is binary-to-octal conversion. The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system.
- However, a 3-to-8-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.



Block diagram

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

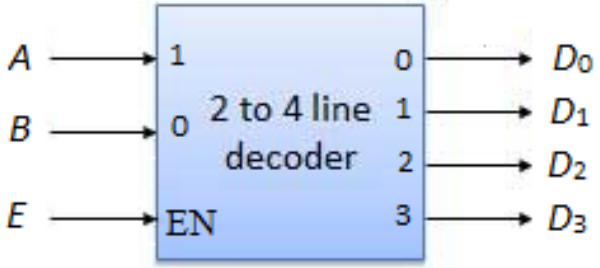


Three-to-eight-line decoder

Decoders with Enable Inputs

- Generally, decoders have the “enable” input.
- The enable input performs no logical operation, but is only responsible for making the decoder active or inactive.
- If the enable “*E*”
  - is zero, then all outputs are zero regardless of the input values.
  - is one, then the decoder performs its normal operation.

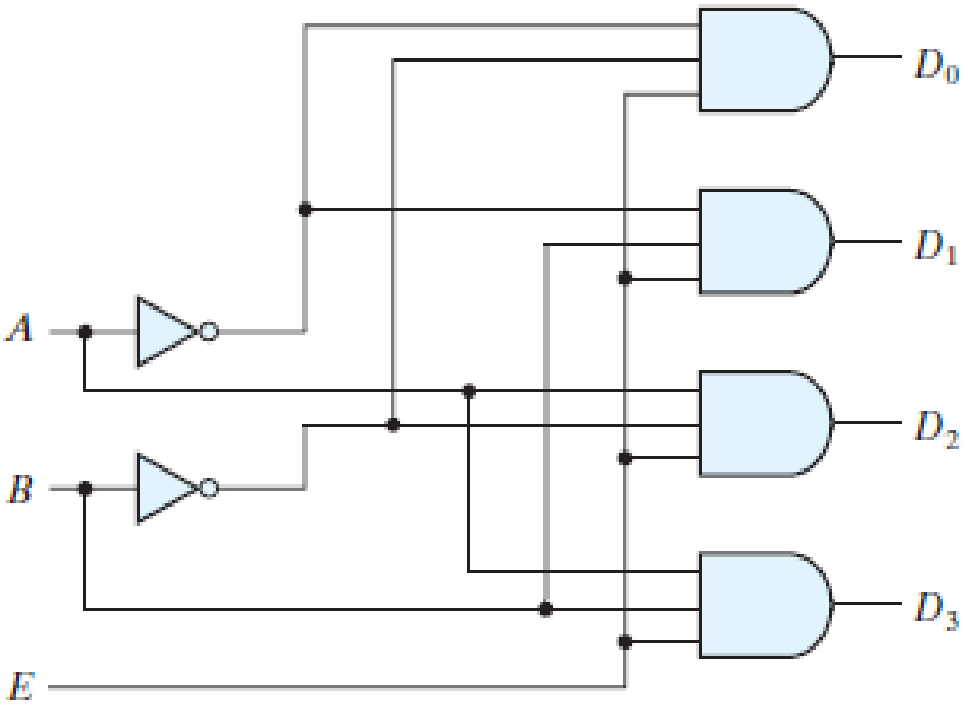
- Decoders with enable inputs can be connected together to form a larger decoder circuit.
- For example, consider the 2-to-4 line decoder with enable input.
- If enable *E* is zero, then all outputs of the decoder will be zeros, regardless of the values of *A* and *B*.
- However, if *E* is one, then the decoder will perform its normal operation, as shown in the truth table.



Block diagram

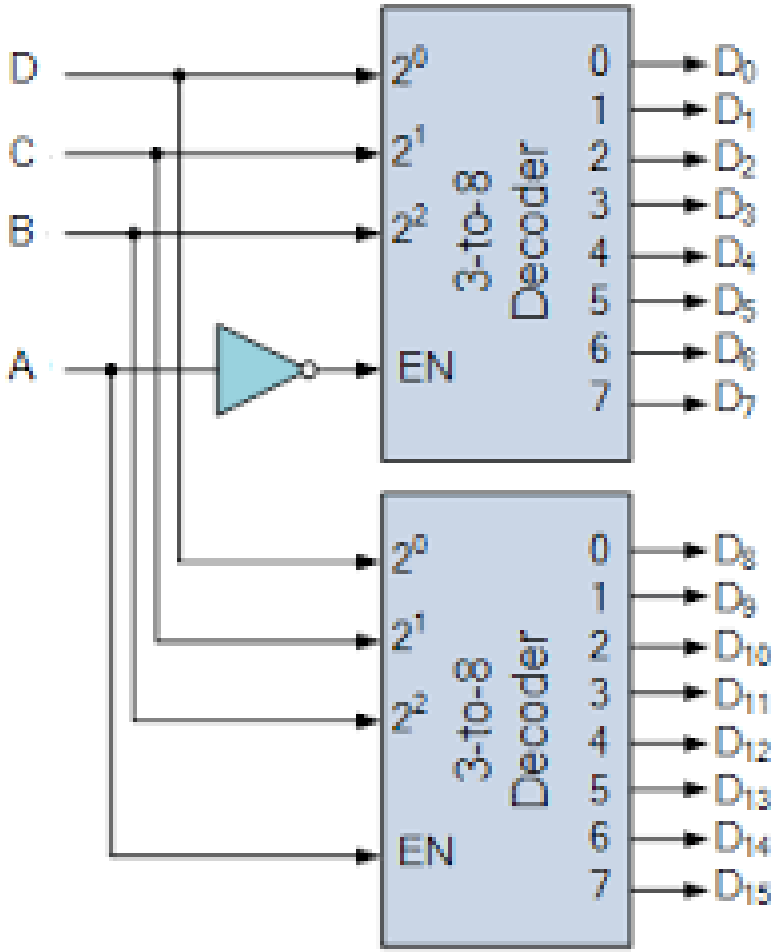
Inputs			Outputs			
E	A	B	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Truth table

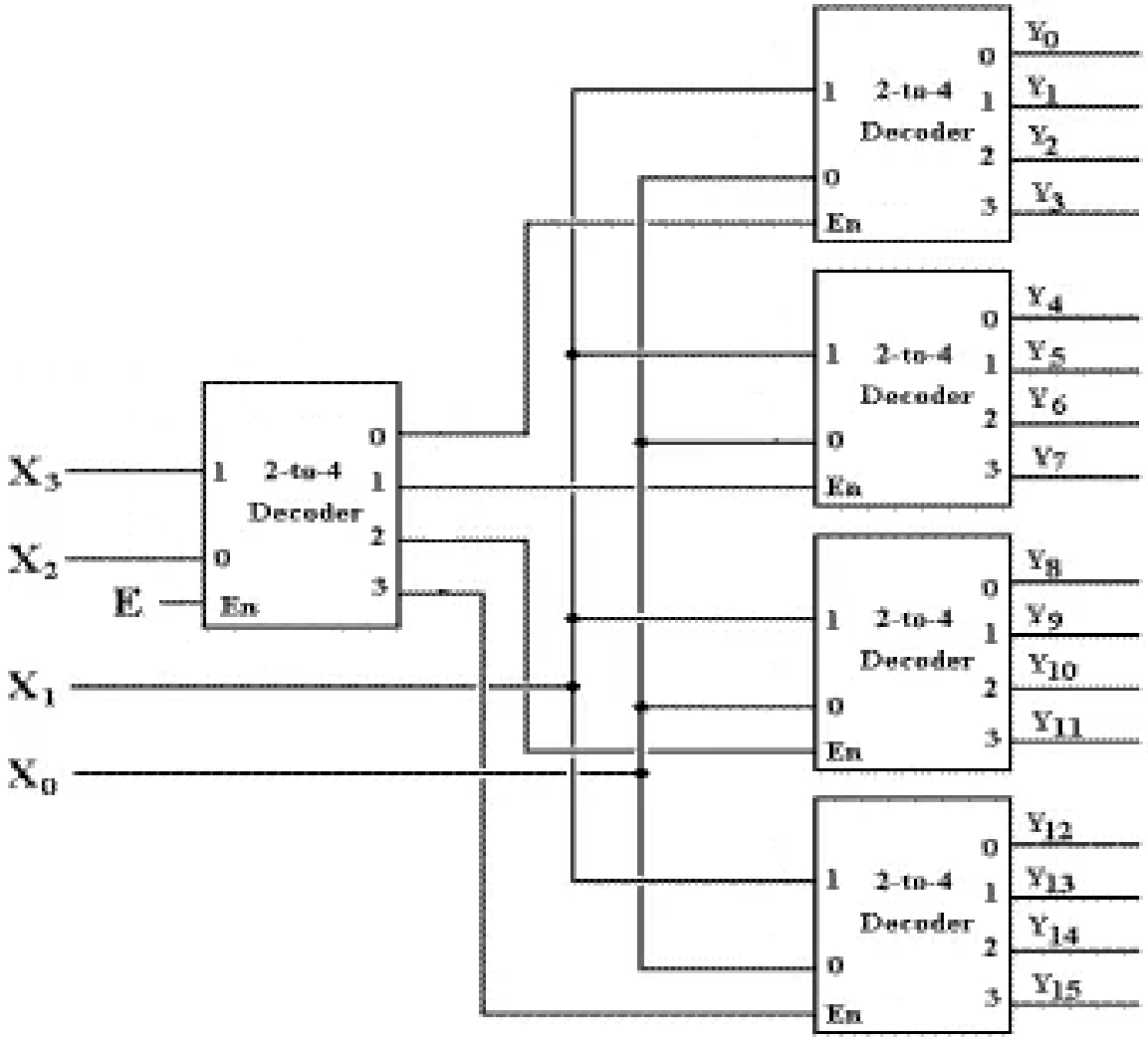


Logic diagram

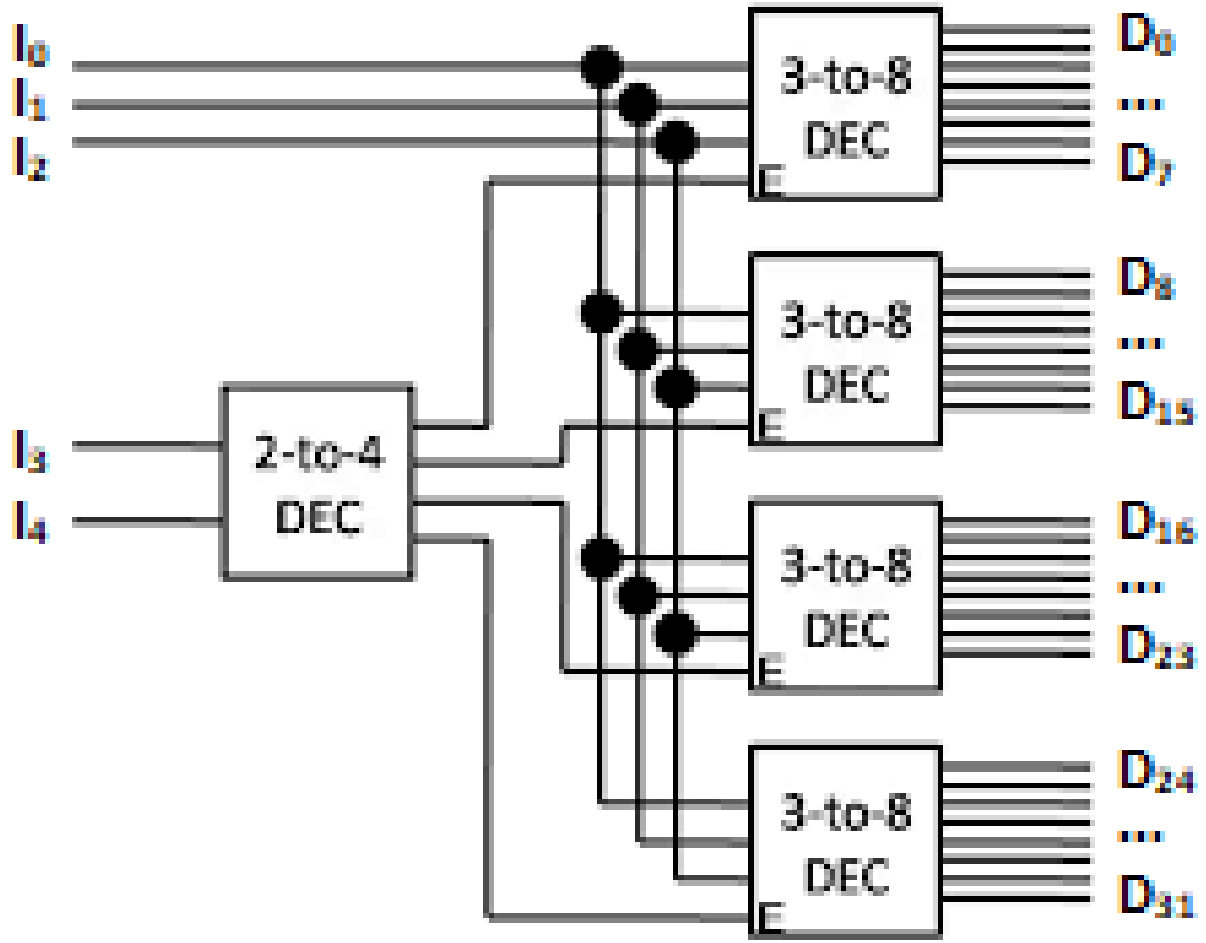
Example: Construct a 4-to-16-line decoder with two 3-to-8-line decoders with enable.



Example: Construct a 4-to-16-line decoder with five 2-to-4-line decoders with enable.



Example: Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and a 2-to-4-line decoder.



Combinational Logic Implementation

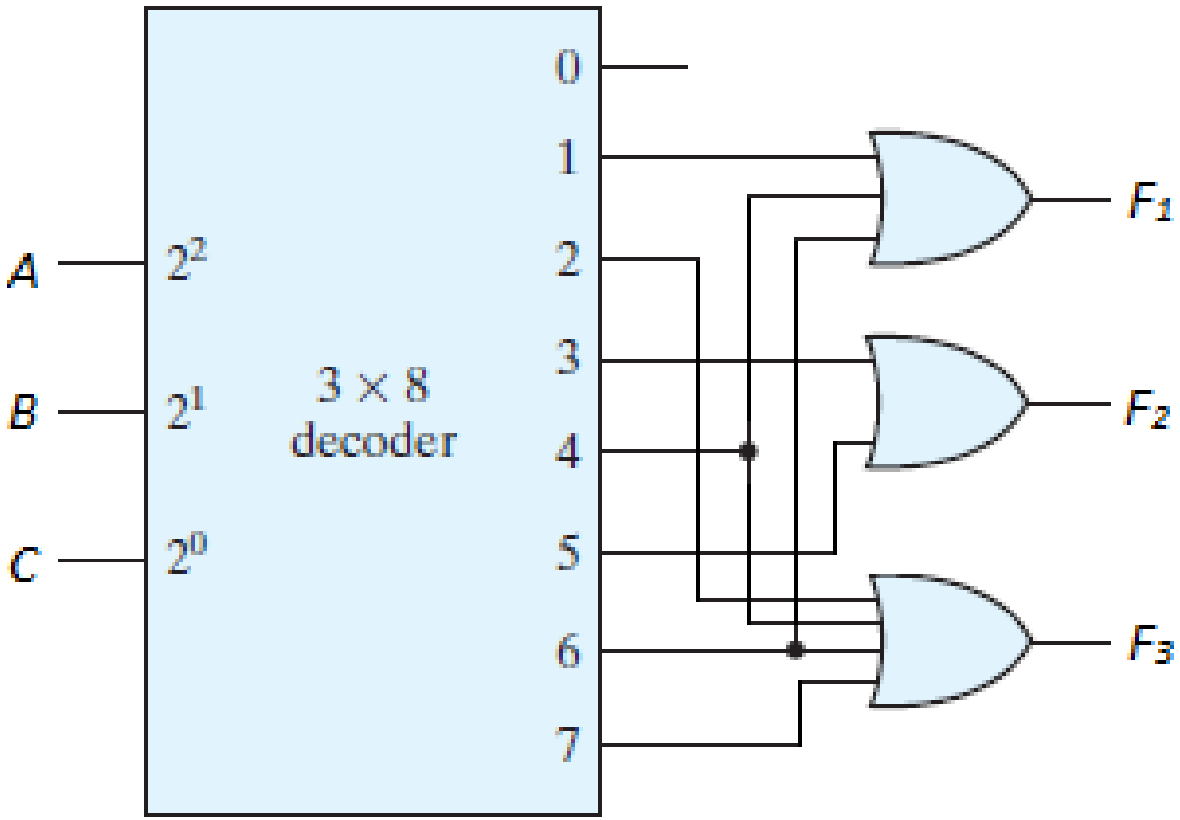
- Example: A combinational circuit is specified by the following three Boolean functions:

  $F_1(A, B, C) = \sum(1, 4, 6)$

  $F_2(A, B, C) = \sum(3, 5)$

  $F_3(A, B, C) = \sum(2, 4, 6, 7)$

  Implement the circuit with a decoder.


A
B
C
2^2
2^1
2^0
3 × 8
decoder
0
1
2
3
4
5
6
7
F_1
F_2
F_3

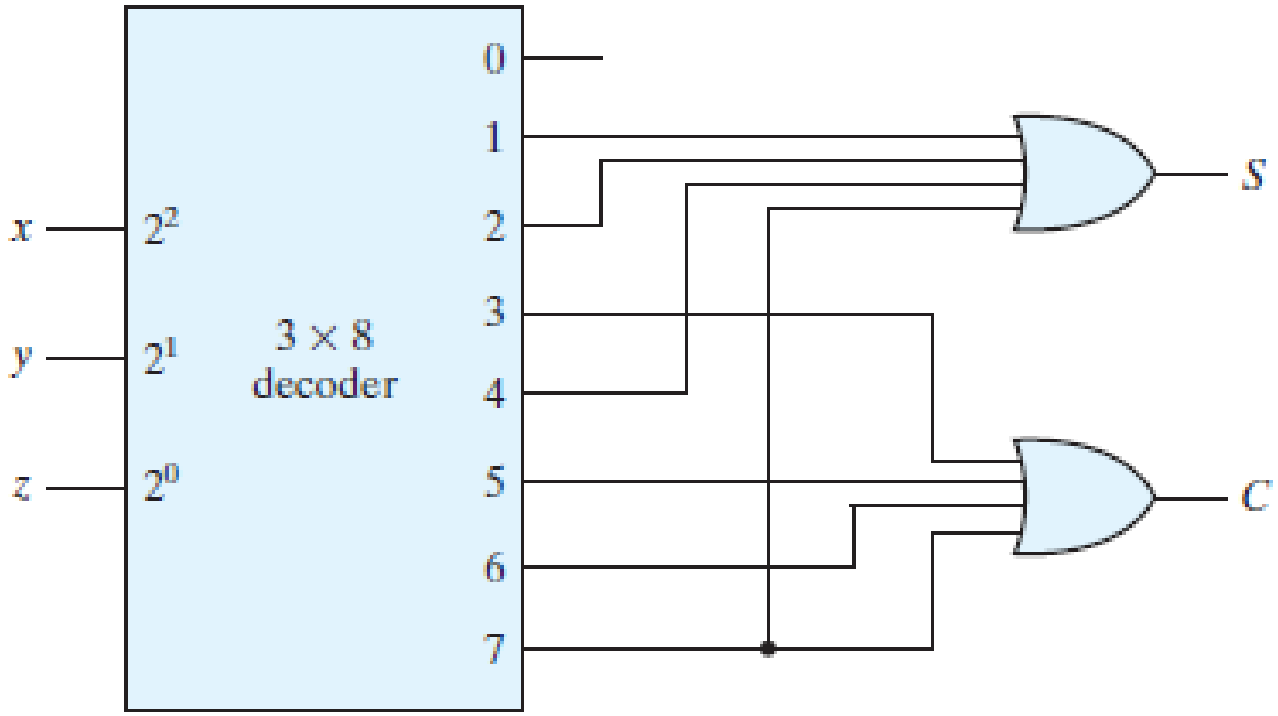
Example: Implement a full adder with a decoder and two OR gates.

- From the truth table of the full adder, we obtain the functions for the combinational circuit in sum-of-minterms form:

$$S(x, y, z) = \sum(1, 2, 4, 7)$$

$$C(x, y, z) = \sum(3, 5, 6, 7)$$

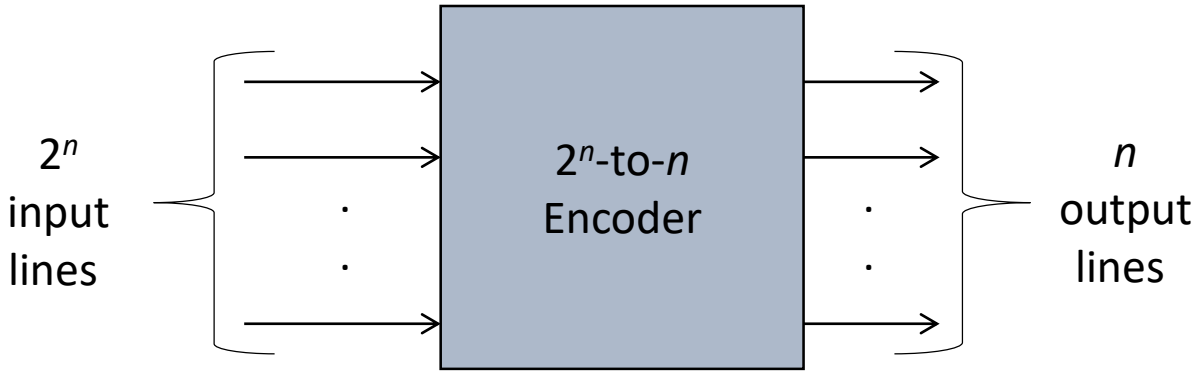
Since there are three inputs and a total of eight minterms, we need a 3-to-8-line decoder.



Implementation of a full adder with a decoder

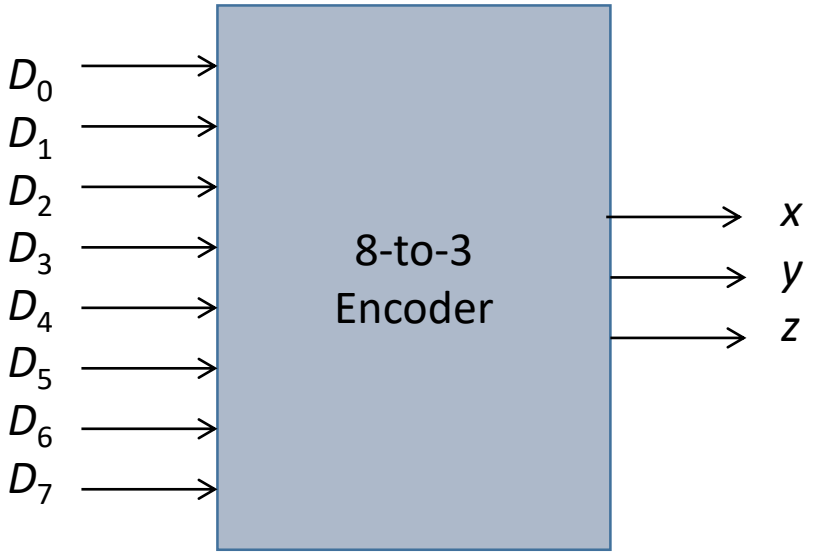
Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and n output lines.
- The output lines, as an aggregate, generate the binary code corresponding to the input value.



Octal-to-Binary Encoder

- It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.
- It is assumed that only one input has a value of 1 at any given time.



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

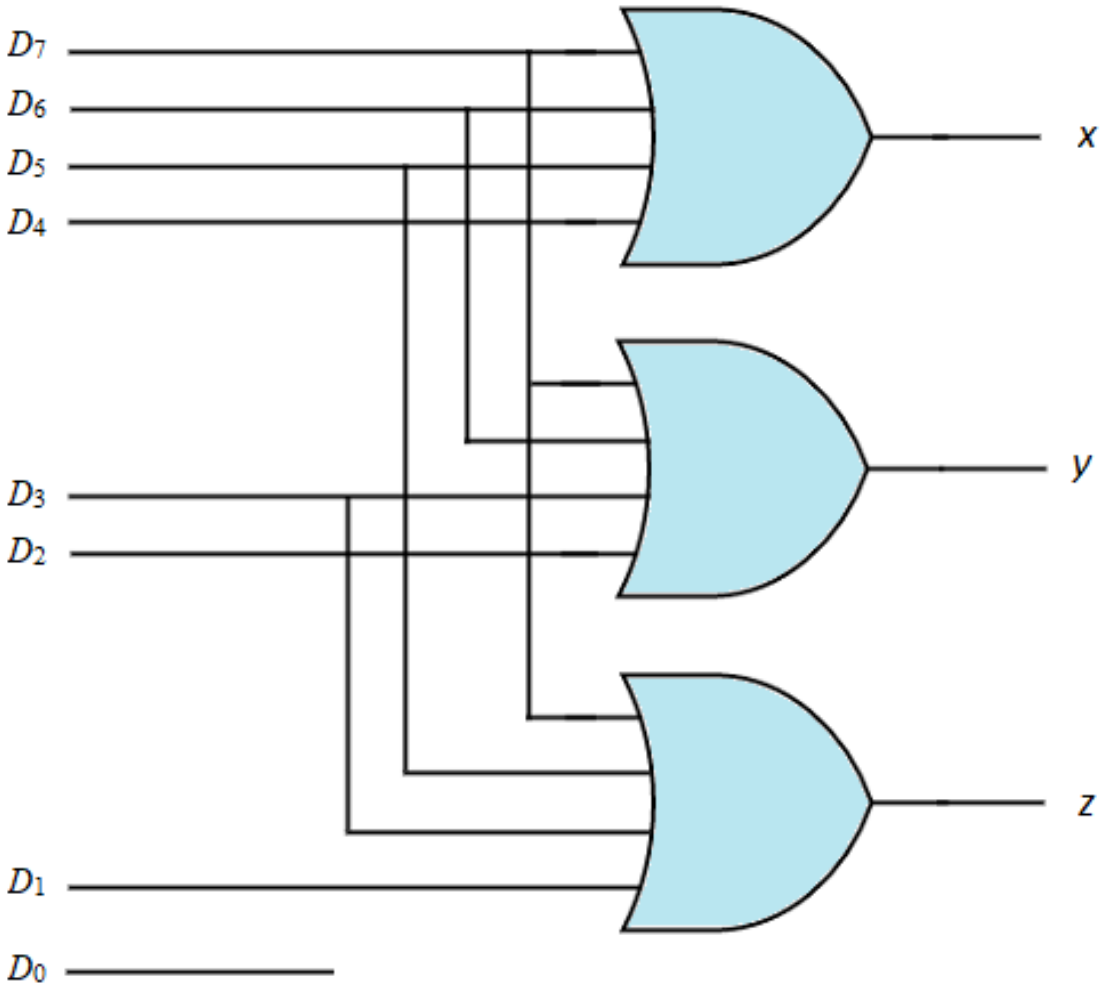
- From the truth table, output *z* is equal to 1 when the input octal digit is 1, 3, 5, or 7. Output *y* is 1 for octal digits 2, 3, 6, or 7, and output *x* is 1 for digits 4, 5, 6, or 7.
- These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

- The encoder can be implemented with three OR gates.


D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
x
y
z

- There are two ambiguities associated with the design of a octal-to-binary encoder:

1. Only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination. For example, if D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary 3 or binary 6.
   - To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded.

2. An output with all 0's is generated when all the inputs are 0; but this output is the same as when D_0 is equal to 1.
   - The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Four-input Priority Encoder

- The truth table of a four-input priority encoder is given in Table.
- In addition to the two outputs *x* and *y*, the circuit has a third output designated by *V*; this is a *valid* bit indicator that is set to 1 when one or more inputs are equal to 1.
- If all inputs are 0, there is no valid input and *V* is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.

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

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

$$x = D_2 + D_3$$

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

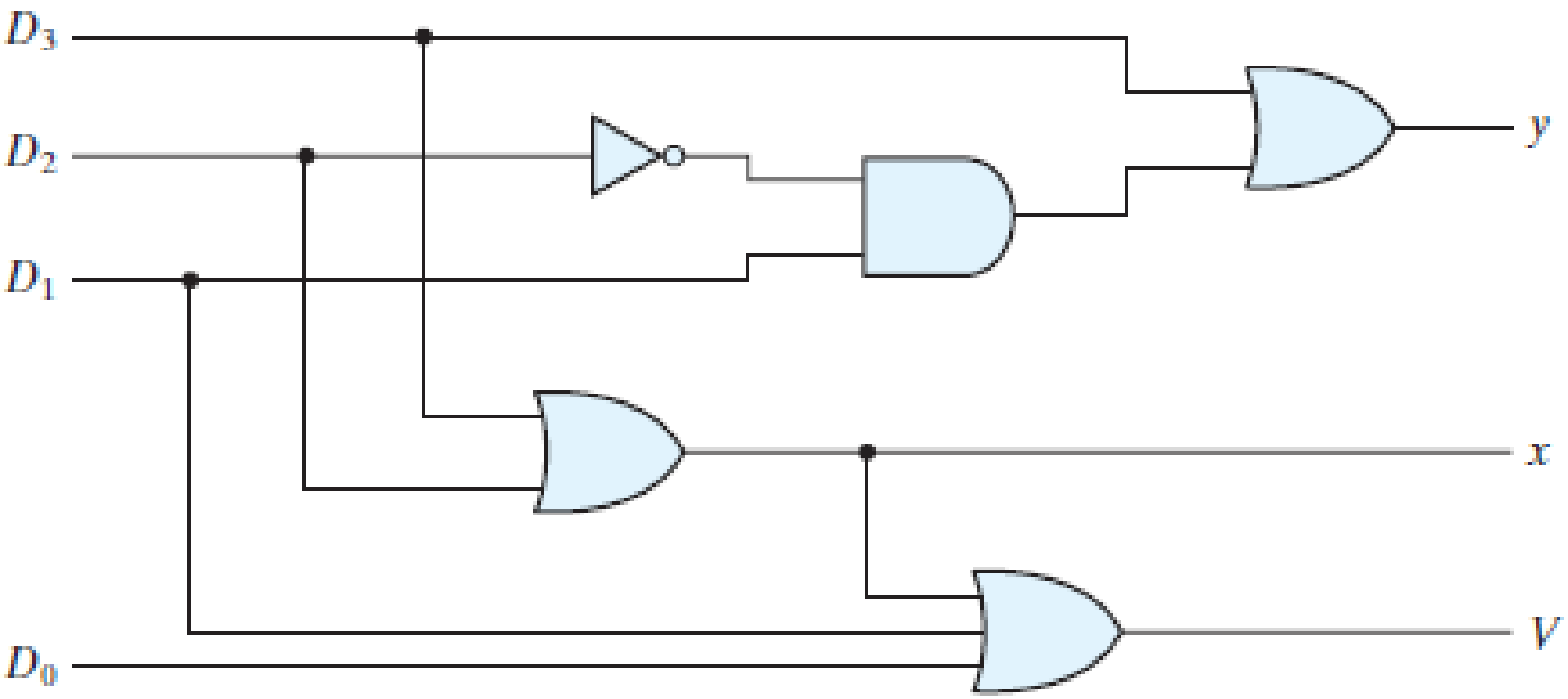
$$y = D_3 + D_1D'_2$$

- The simplified Boolean expressions for the priority encoder are obtained from the maps.
- The condition for output *V* is an OR function of all the input variables.
- The priority encoder is implemented in Fig. according to the following Boolean functions:

$$x = D_2 + D_3$$

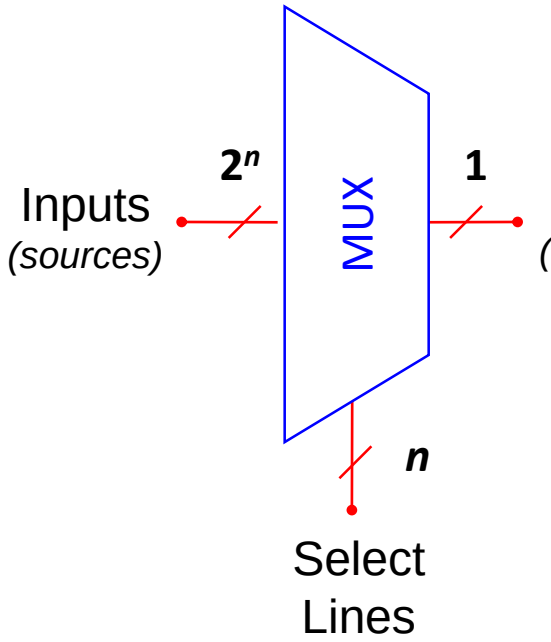
$$y = D_3 + D_1D_2'$$

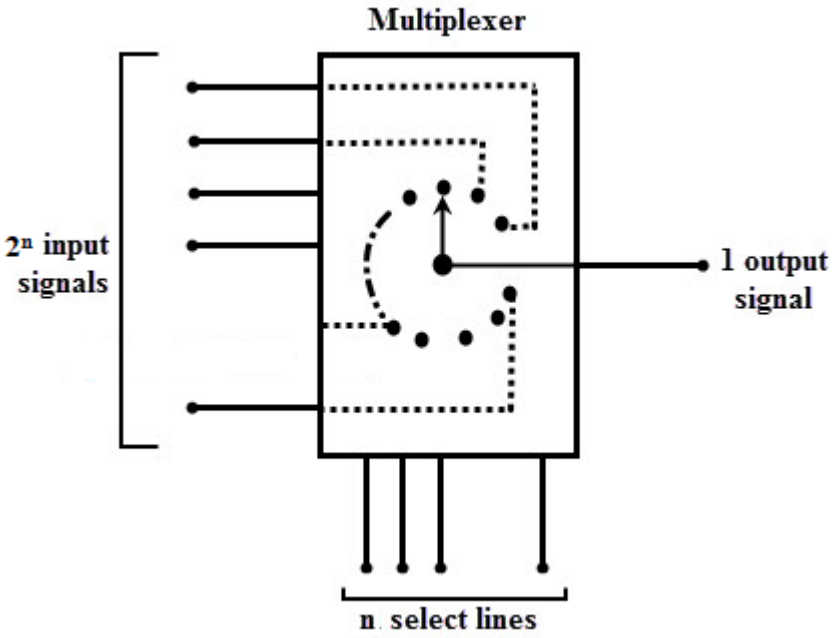
$$V = D_0 + D_1 + D_2 + D_3$$


D_3
D_2
D_1
D_0
y
x
V

Multiplexers

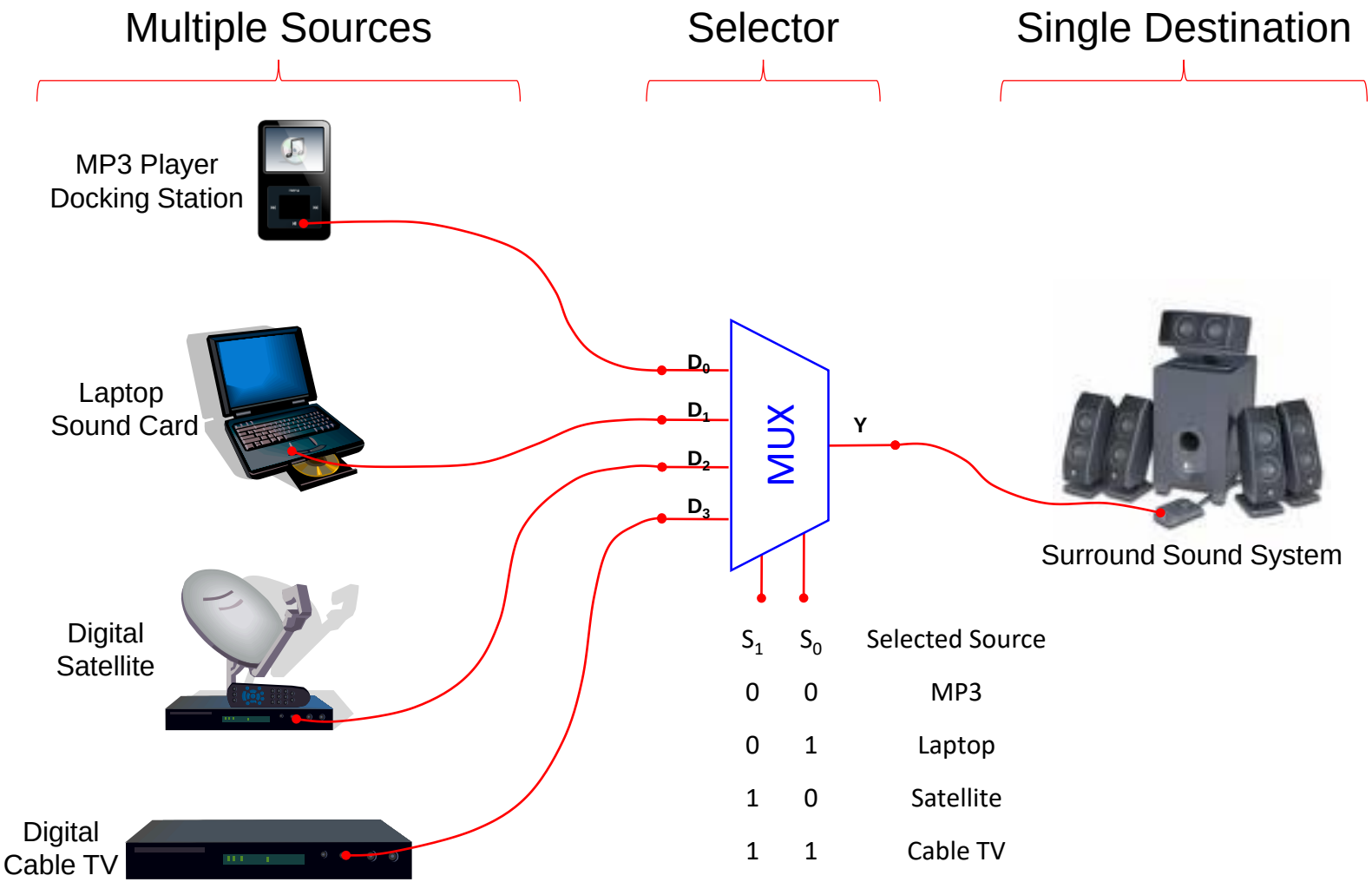
- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.





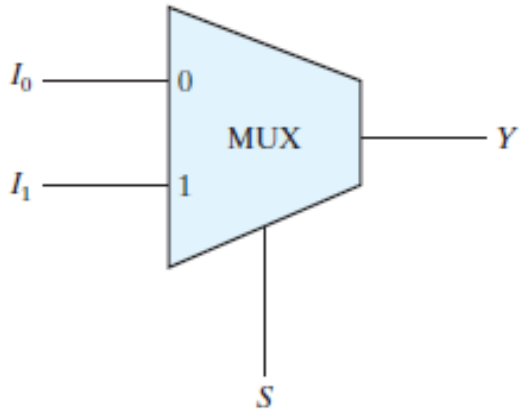
- A multiplexer is also called a *data selector*, since it selects one of many inputs and steers the binary information to the output line.

Typical Application of a MUX



Two-to-One-line Multiplexer

- A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in Fig.
- The circuit has two data input lines, one output line, and one selection line *S*.

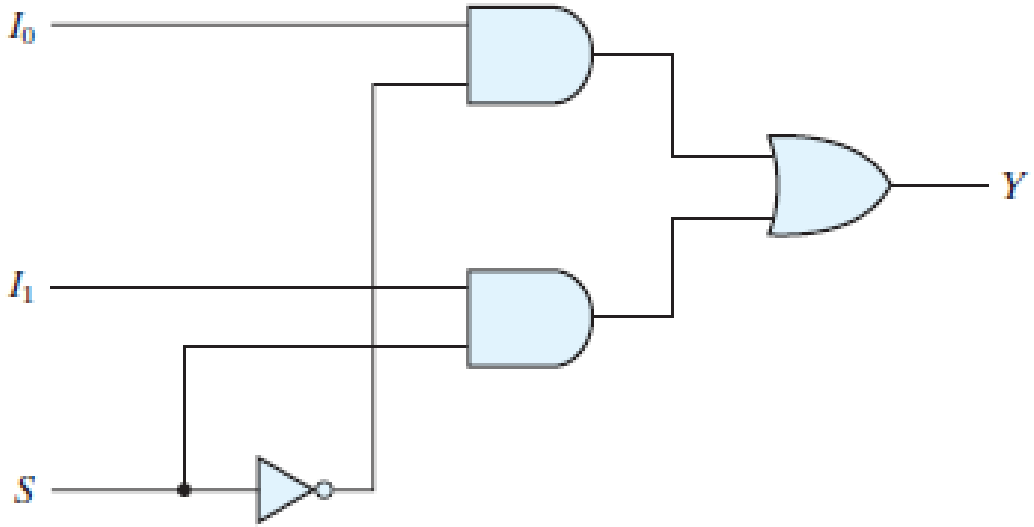


Block diagram

S	Y
0	I_0
1	I_1

Function table

$$I_0\overline{S} + I_1S$$

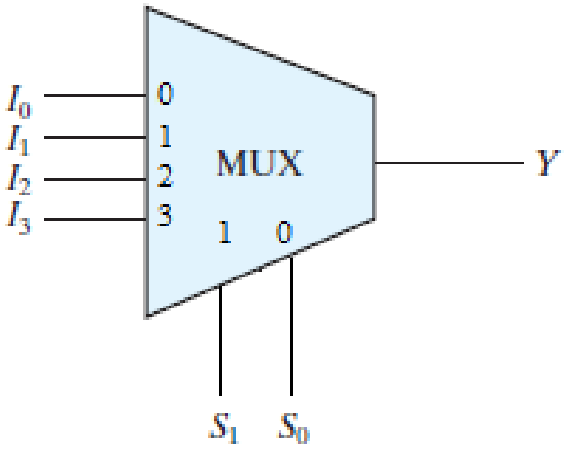


Logic diagram

- When $S = 0$, the upper AND gate is enabled and I_0 has a path to the output.
- When $S = 1$, the lower AND gate is enabled and I_1 has a path to the output.
- The multiplexer acts like an electronic switch that selects one of two sources.

Four-to-One-line Multiplexer

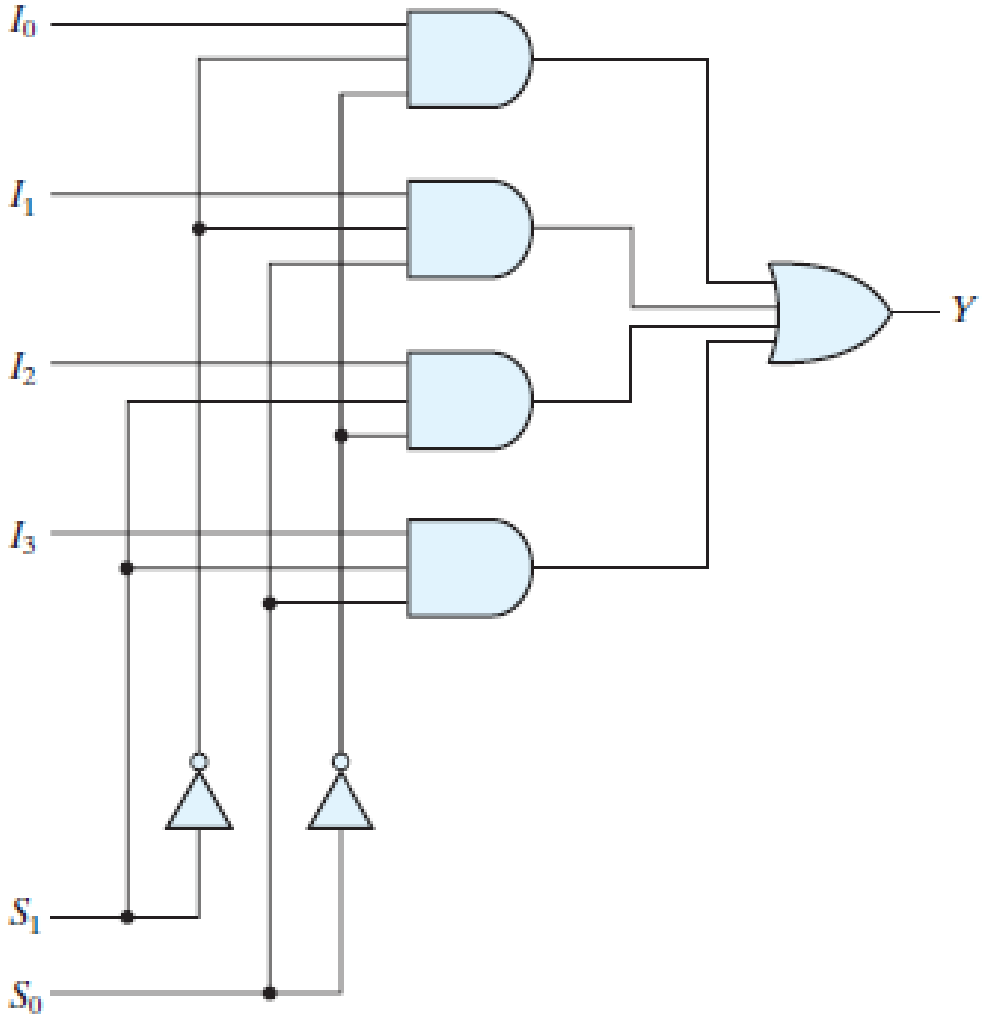
- A four-to-one-line multiplexer is shown in Fig.
- Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate.
- Selection lines S_1 and S_0 are decoded to select a particular AND gate.
- The outputs of the AND gates are applied to a single OR gate that provides the one-line output.
- The function table lists the input that is passed to the output for each combination of the binary selection values.



Block diagram

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

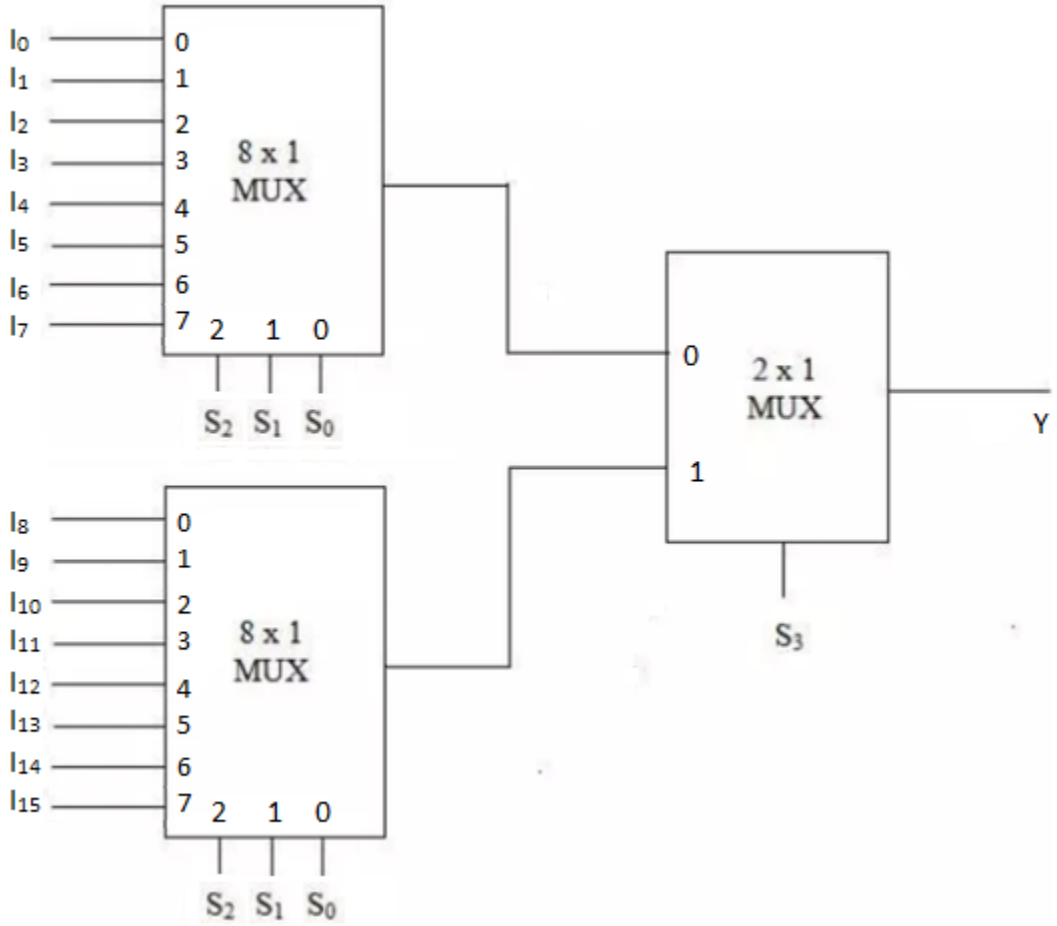
Function table



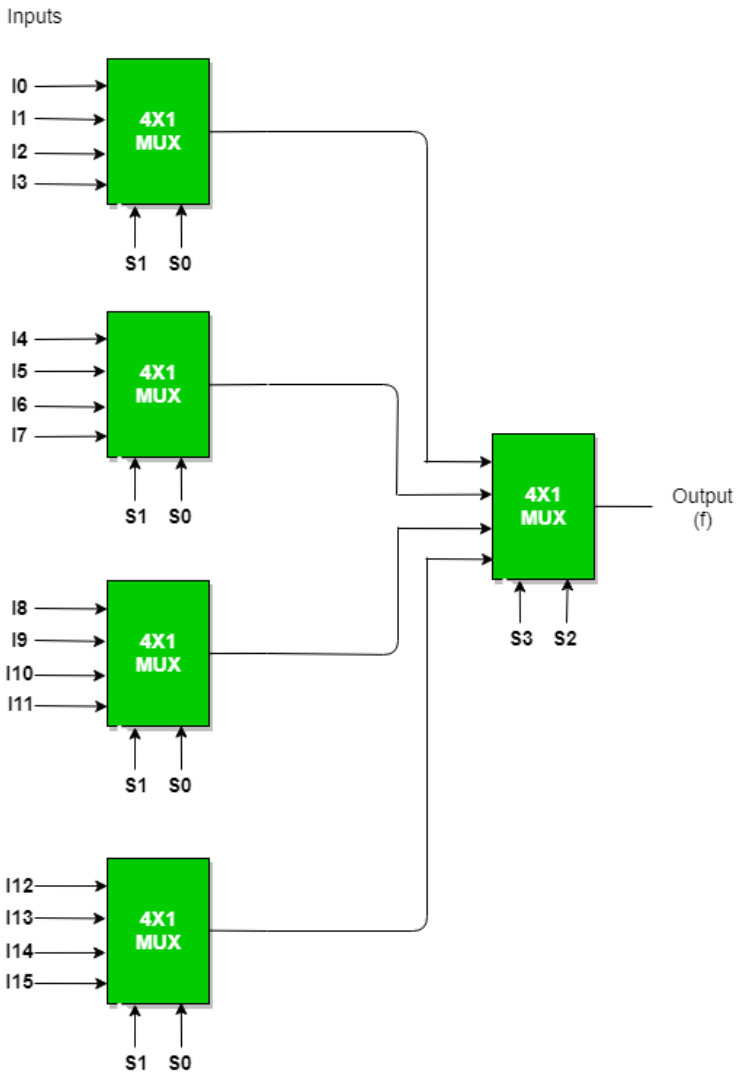
Logic diagram

$$Y = I_0\overline{S}_1\overline{S}_0 + I_1\overline{S}_1S_0 + I_2S_1\overline{S}_0 + I_3S_1S_0$$

Example: Construct a 16 × 1 multiplexer with two 8 × 1 and one 2 × 1 multiplexers.



Example: Construct a 16 × 1 multiplexer with five 4 × 1 multiplexers.

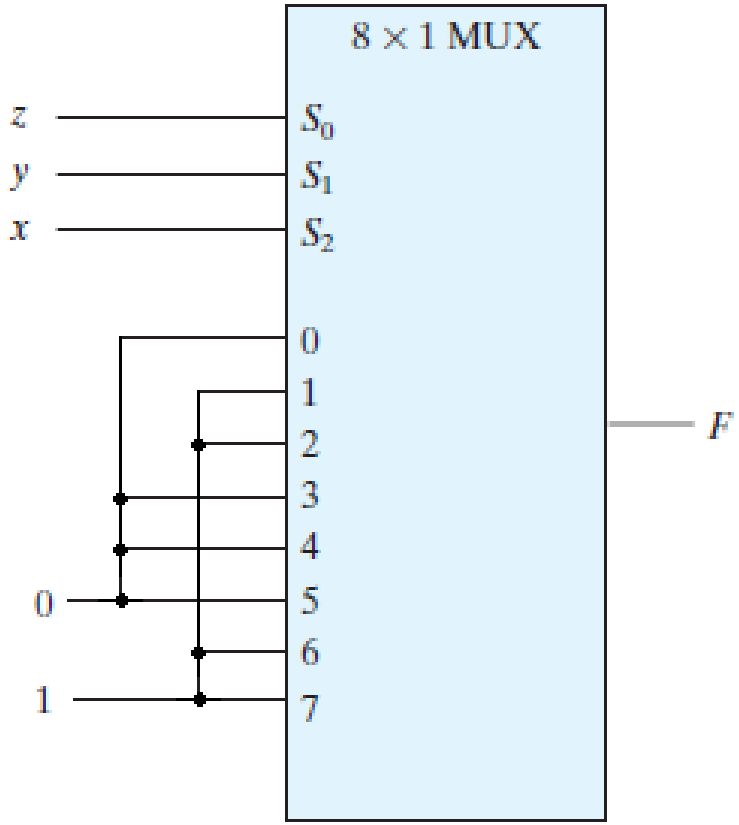


Boolean Function Implementation

- The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs.
- The individual minterms can be selected by the data inputs, there by providing a method of implementing a Boolean function.

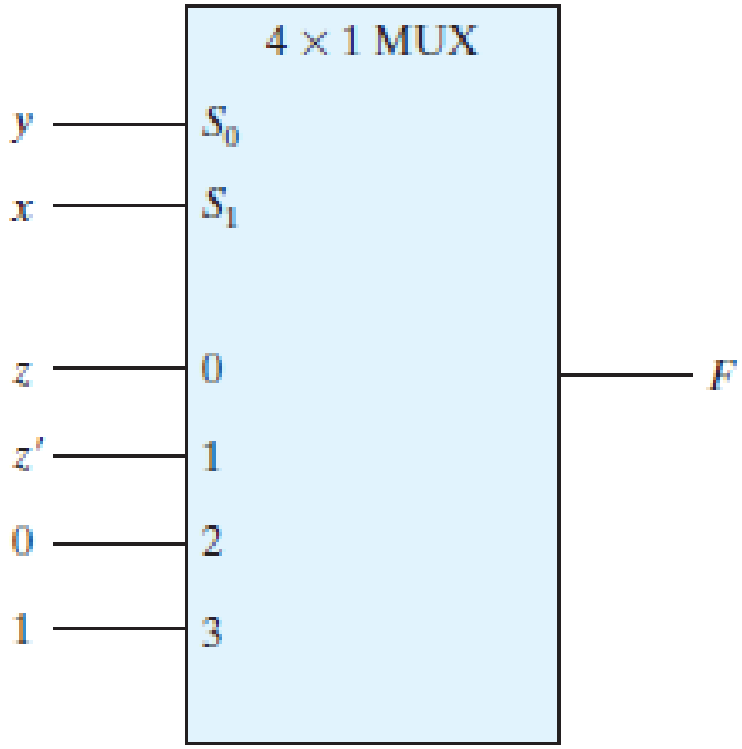
Example: Implement the following Boolean function with a multiplexer.

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



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

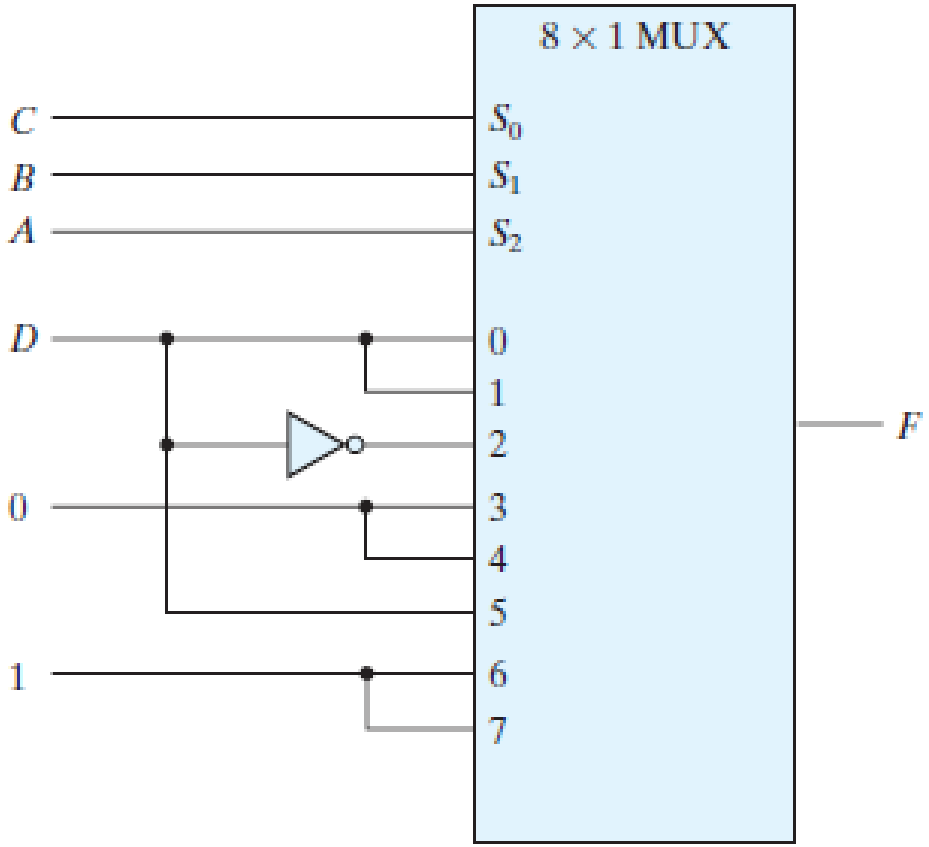
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	



Example: Implement the following Boolean function with a multiplexer.

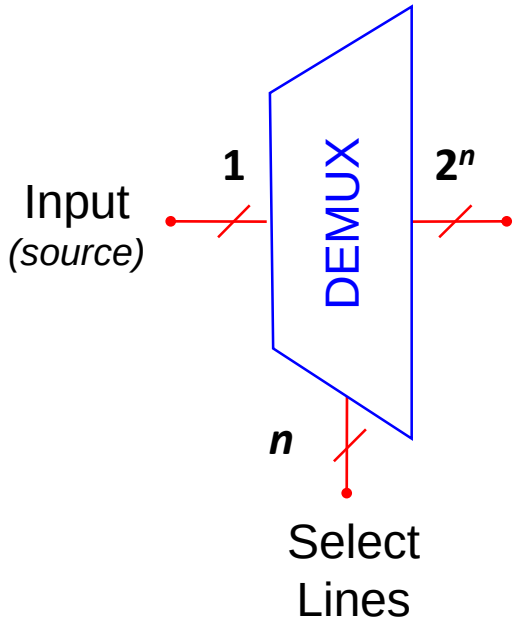
$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

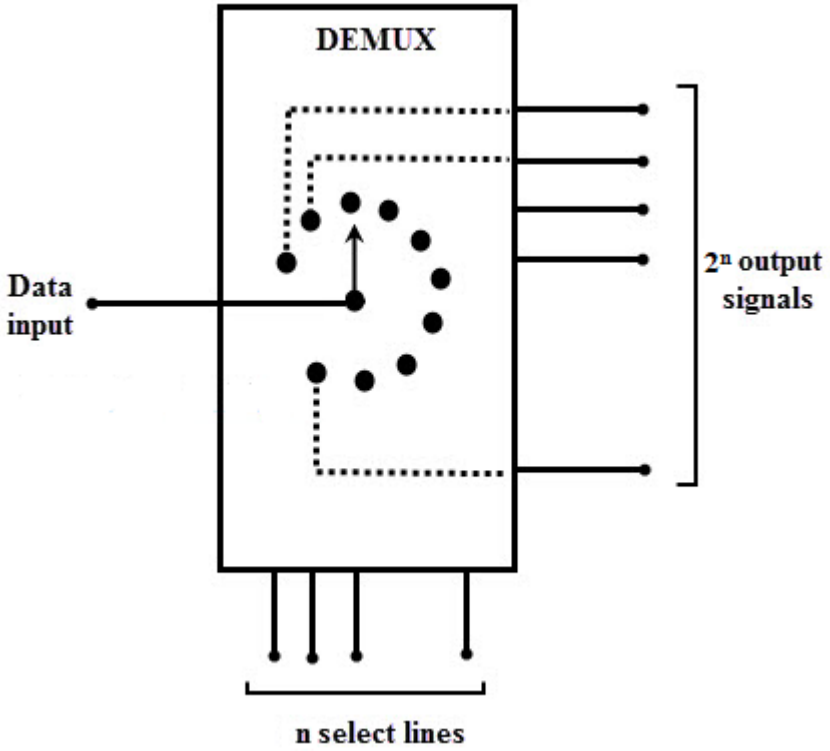
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



De-multiplexers

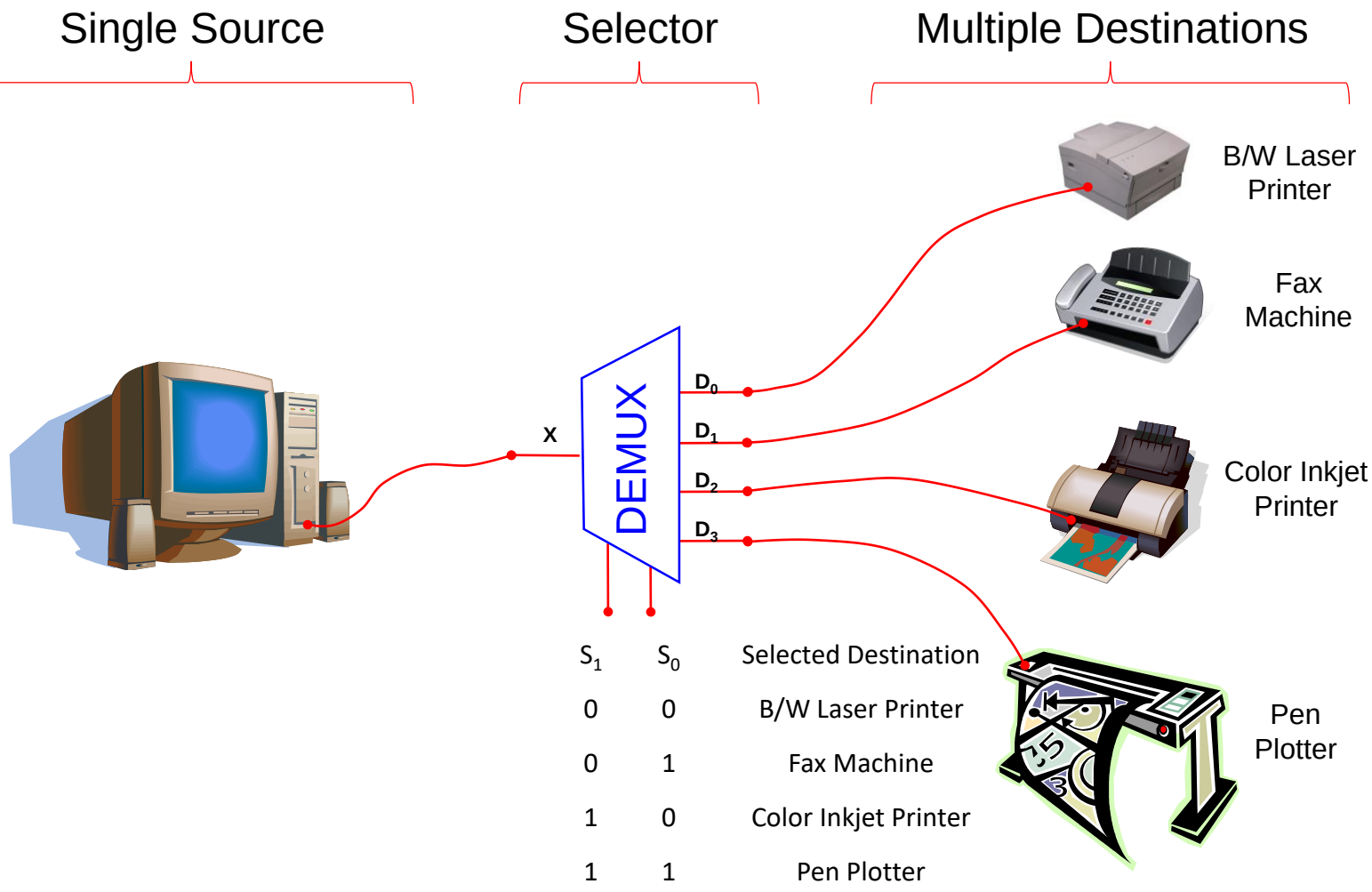
- A de-multiplexer performs the reverse operation of multiplexer; it takes single input and distributes it over several outputs.
- A decoder with enable input can function as a *de-multiplexer* - a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.





- A de-multiplexer is also called a *data distributor*, since it transmits the same data to different destinations.

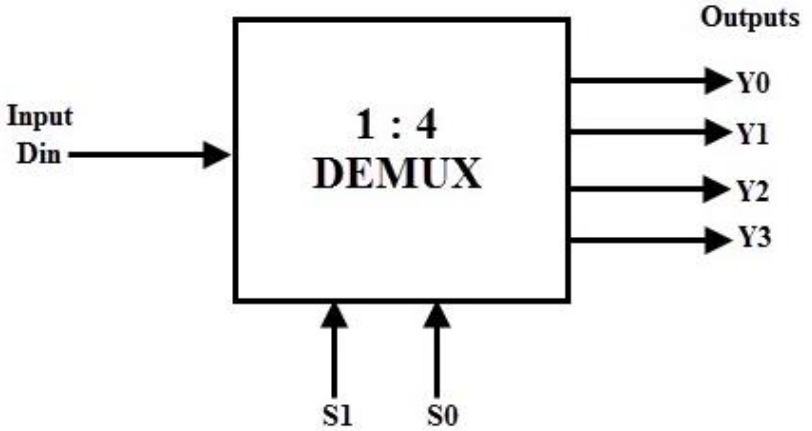
Typical Application of a DEMUX



S_1	S_0	Selected Destination
0	0	B/W Laser Printer
0	1	Fax Machine
1	0	Color Inkjet Printer
1	1	Pen Plotter

1-to-4 Line De-multiplexer

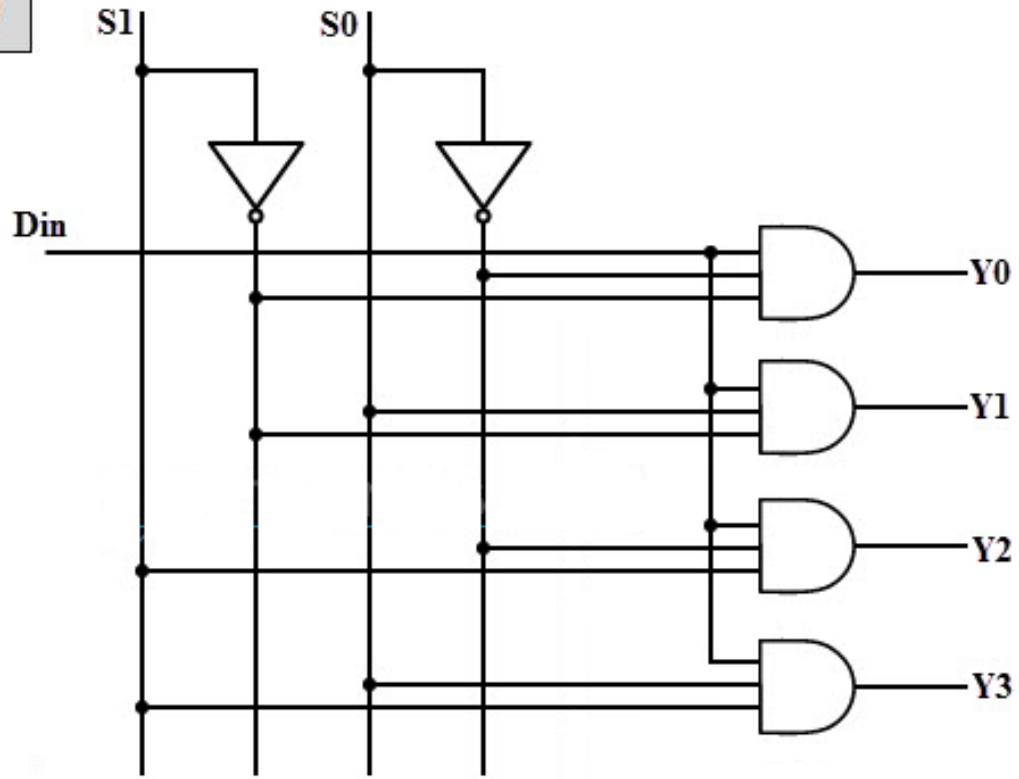
- A 1-to-4 line de-multiplexer has a single input (D), two selection lines (S_1 and S_0) and four outputs (Y_0 to Y_3).



- The input data goes to an t a given time for a particular combination of select lines.

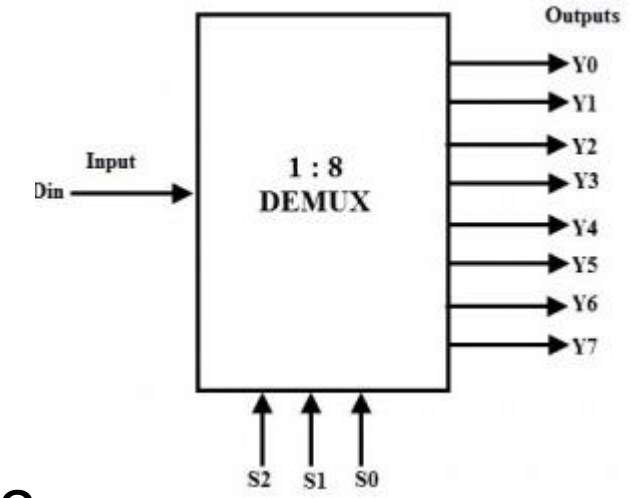
Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

$$Y_0 = \overline{S}_1 \overline{S}_0 D$$
$$Y_1 = \overline{S}_1 S_0 D$$
$$Y_2 = S_1 \overline{S}_0 D$$
$$Y_3 = S_1 S_0 D$$



1-to-8 Line De-multiplexer

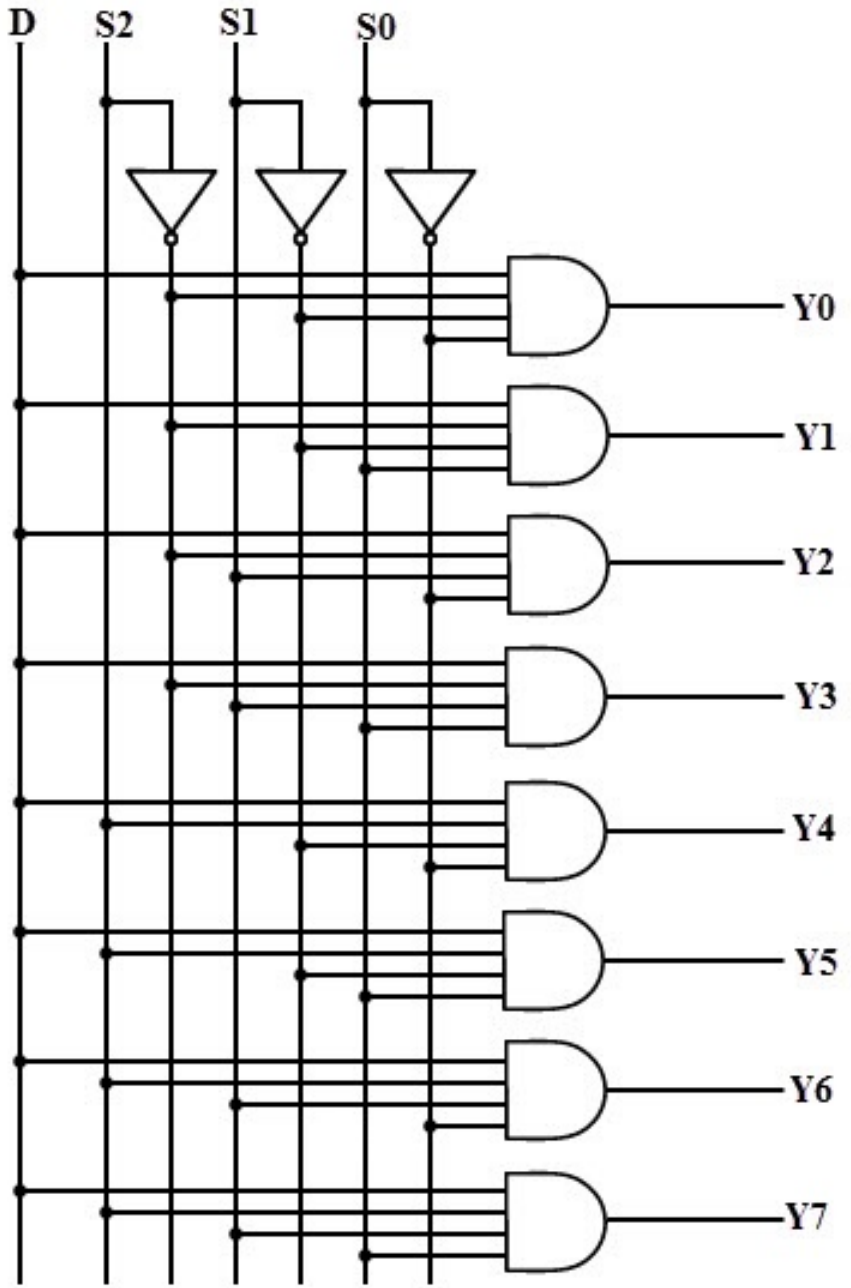
- A 1-to-8 de-multiplexer consists of single input D, three select inputs S_2, S_1 and S_0 and eight outputs from Y_0 to Y_7.



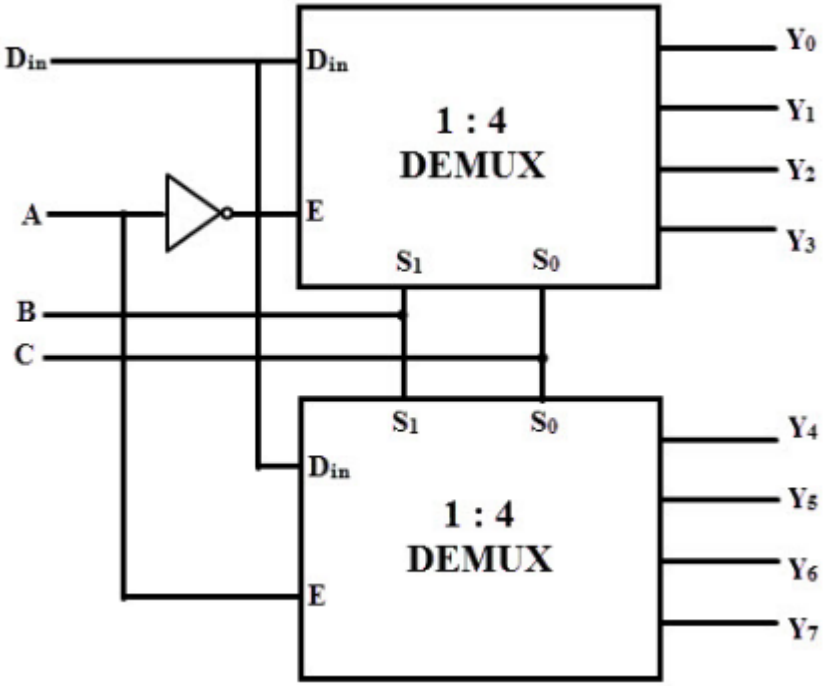
- It distributes one input line to one of 8 output lines depending on the combination of select inputs.

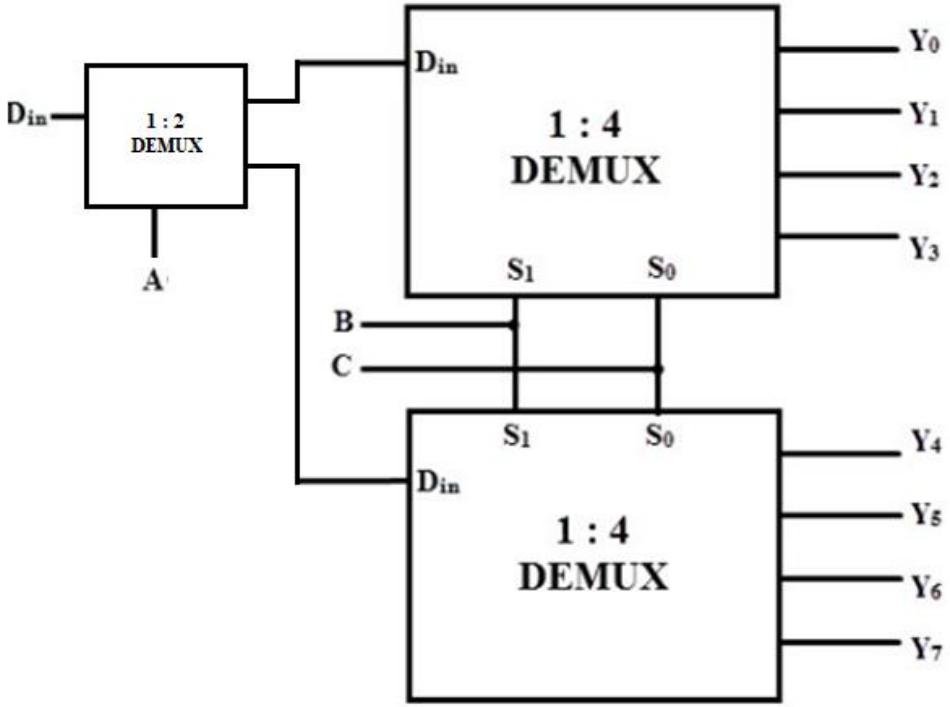
Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

$$Y_0 = \overline{S}_2\overline{S}_1\overline{S}_0D \qquad Y_4 = S_2\overline{S}_1\overline{S}_0D$$
$$Y_1 = \overline{S}_2\overline{S}_1S_0D \qquad Y_5 = S_2\overline{S}_1S_0D$$
$$Y_2 = \overline{S}_2S_1\overline{S}_0D \qquad Y_6 = S_2S_1\overline{S}_0D$$
$$Y_3 = \overline{S}_2S_1S_0D \qquad Y_7 = S_2S_1S_0D$$


D
S2
S1
S0
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7

Example: Construct a 1-to-8 DEMUX using two 1-to-4 De-multiplexers.





Binary Parallel Adder;
Binary Adder-Subtractor;
Binary Multiplier;
Magnitude Comparator;
Code Converters

Binary Parallel Adder

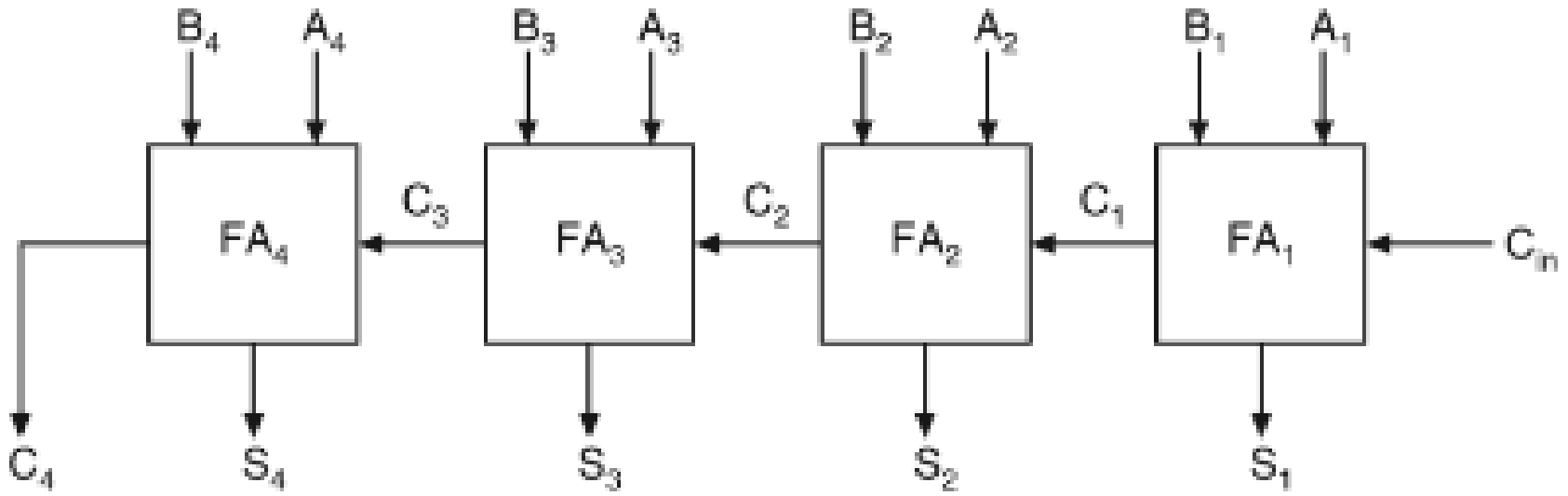
- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

Example:

- Consider the two binary numbers A = 1011 and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:

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 shows the interconnection of four full-adder (FA) circuits to provide a 4-bit binary parallel adder.



Logic diagram of a 4-bit binary parallel adder.

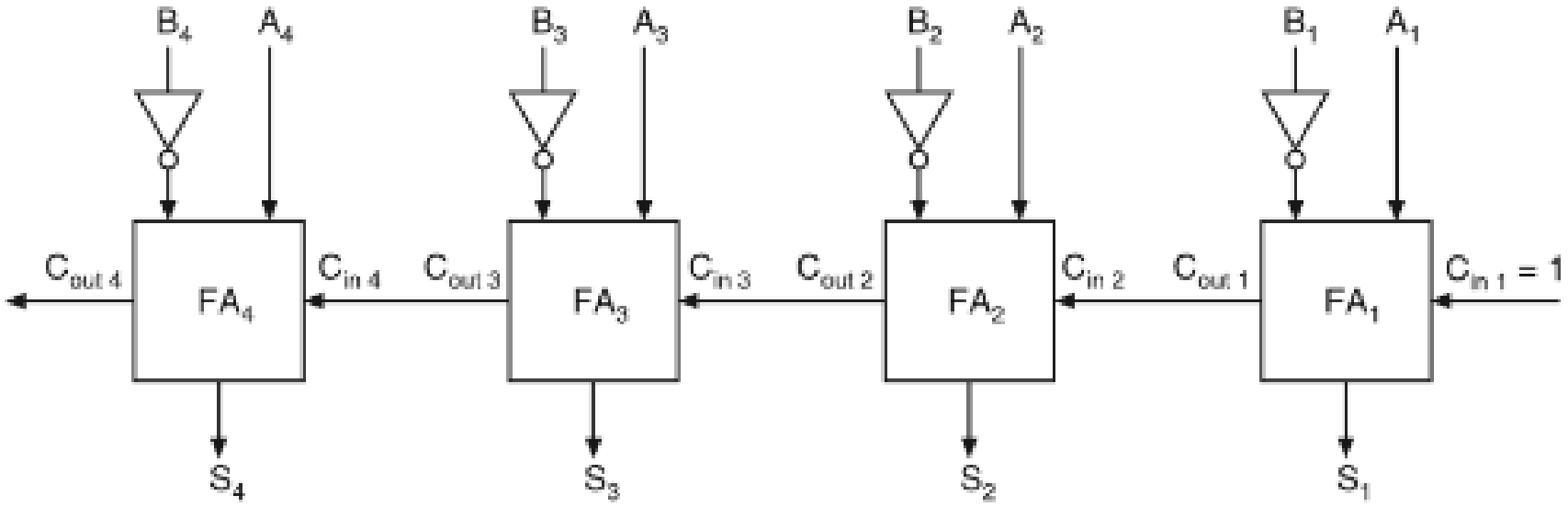
- The augend bits of *A* and the addend bits of *B* are designated by subscript numbers from right to left, with subscript 1 denoting the least significant bit.
- The carries are connected in a chain through the full adders. The input carry to the adder is C_{in}, and it ripples through the full adders to the output carry C_4.
- The *S* outputs generate the required sum bits.

- An *n*-bit adder requires *n* full adders, with each output carry connected to the input carry of the next higher order full adder. The input carry to the least significant position is fixed at 0.
- The parallel adder, in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a *ripple carry adder*.

Binary Parallel Substractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of complements.
- Remember that the subtraction *A* - *B* can be done by taking the 2's complement of *B* and adding it to *A* .
- The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.

- The 1’s complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

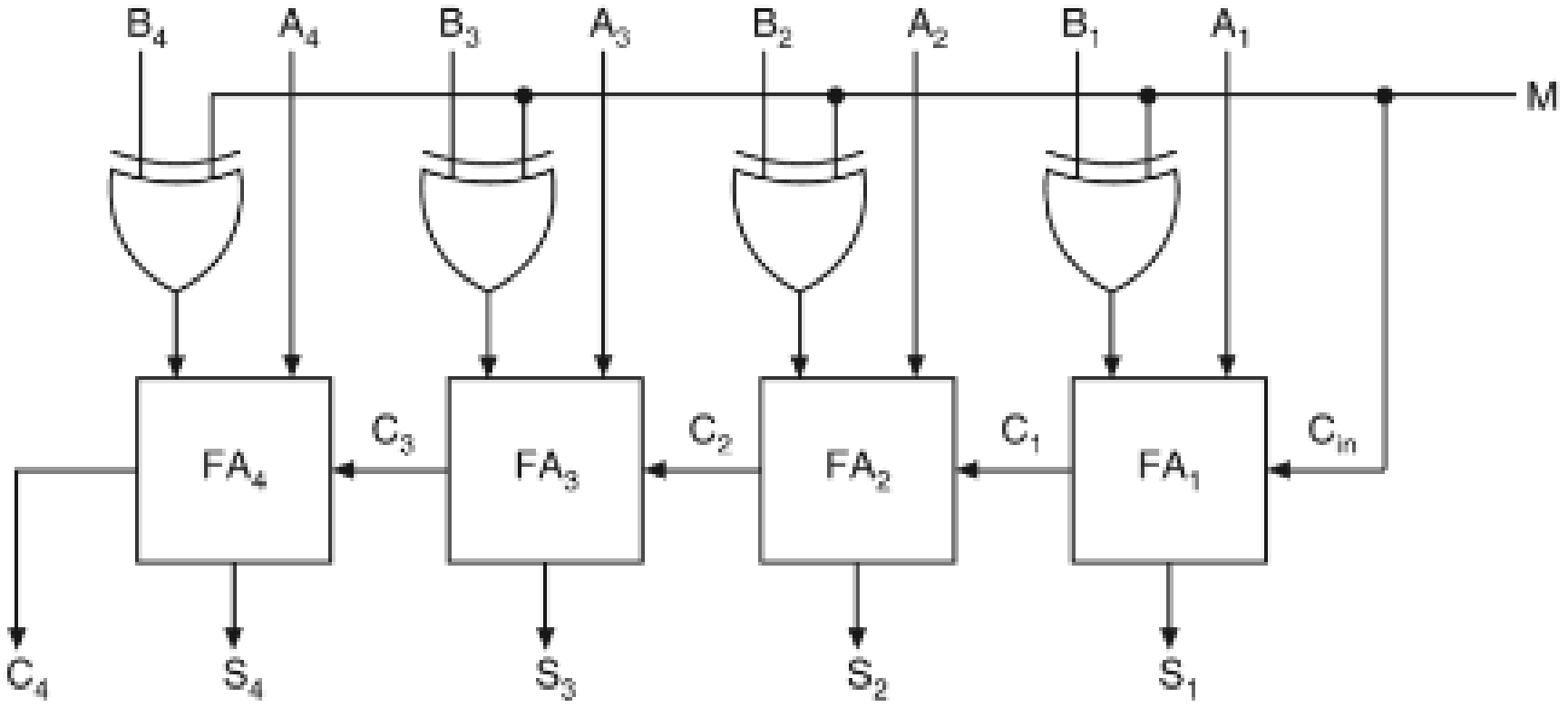


Logic diagram of a 4-bit parallel subtractor.

- The circuit for subtracting *A* - *B* consists of an adder with inverters placed between each data input *B* and the corresponding input of the full adder.
- The input carry C_{in} must be equal to 1 when subtraction is performed.
- The operation thus performed becomes *A*, plus the 1's complement of *B*, plus 1. This is equal to *A* plus the 2's complement of *B*.
- For unsigned numbers, that gives *A* - *B* if $A \geq B$ or the 2's complement of (*B* - *A*) if $A < B$.

Binary Adder-Substractor

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder.
- A 4-bit adder-substractor circuit is shown in Fig.
- The mode input *M* controls the operation.
- When *M* = 0, the circuit is an adder, and when *M* = 1, the circuit becomes a substractor.



Logic diagram of a 4-bit binary adder-subtractor.

- Each exclusive-OR gate receives input *M* and one of the inputs of *B* .
- When $M = 0$, we have $B \oplus 0 = B$. The full adders receive the value of *B* , the input carry is 0, and the circuit performs *A* plus *B*.
- When $M = 1$, we have $B \oplus 1 = B'$ and $C_{in} = 1$. The *B* inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation *A* plus the 2's complement of *B*.

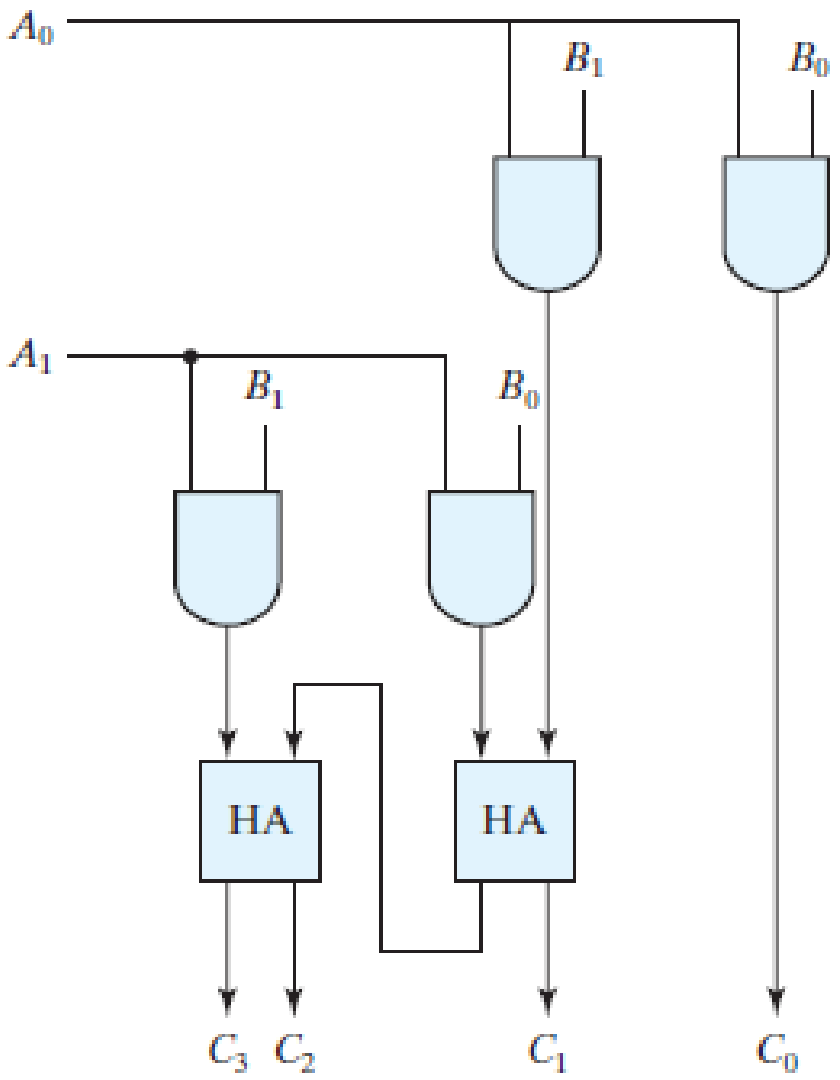
Binary Multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product. Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.

- Consider the multiplication of two 2-bit numbers as shown in Fig.
- The multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is $C_3C_2C_1C_0$.
- The first partial product is formed by multiplying B_1B_0 by A_0.
- The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation. Therefore, the partial product can be implemented with AND gates as shown in the diagram.

- The second partial product is formed by multiplying B_1B_0 by A_1 and shifting one position to the left.
- The two partial products are added with two half-adder (HA) circuits.
- Usually, there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.
- Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



Two-bit by two-bit binary multiplier

- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
- A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product.
- The last level produces the product.
- For J multiplier bits and K multiplicand bits, we need $(J \times K)$ AND gates and $(J - 1)K$-bit adders to produce a product of $(J + K)$ bits.

Comparators

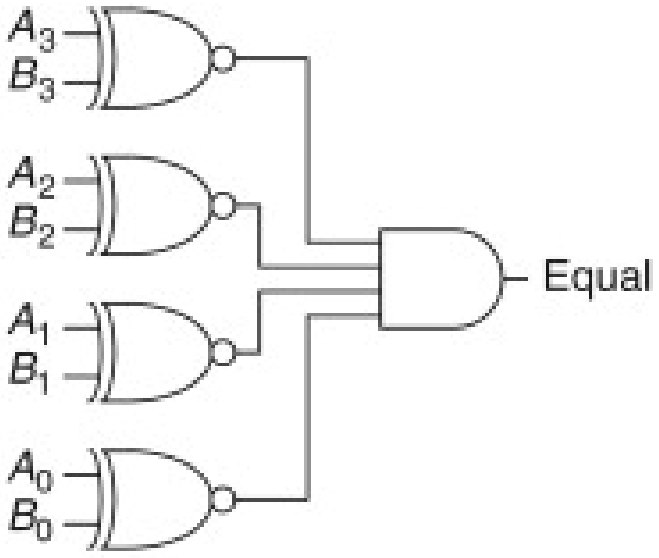
- A comparator is a logic circuit, used to compare the magnitudes of two binary numbers.
- The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number.
- There are two common types of comparators.
  - Equality Comparator
  - Magnitude Comparator

Equality Comparator

- An *equality comparator* produces a single output indicating whether A is equal to B.
- Two binary numbers are equal, if and only if all their corresponding bits coincide.
- For example, two 4-bit binary numbers, $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are equal, if and only if, $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$.

- The equality can be expressed logically with an exclusive-NOR function as

$$EQUALITY = (A_3 \odot B_3)\,(A_2 \odot B_2)\,(A_1 \odot B_1)\,(A_0 \odot B_0)$$



Logic diagram of equality comparator.

Magnitude Comparator

- A *magnitude comparator* is a combinational circuit that compares two numbers *A* and *B* and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

1-bit Magnitude Comparator

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

$$A < B: L = \bar{A}_0 B_0$$

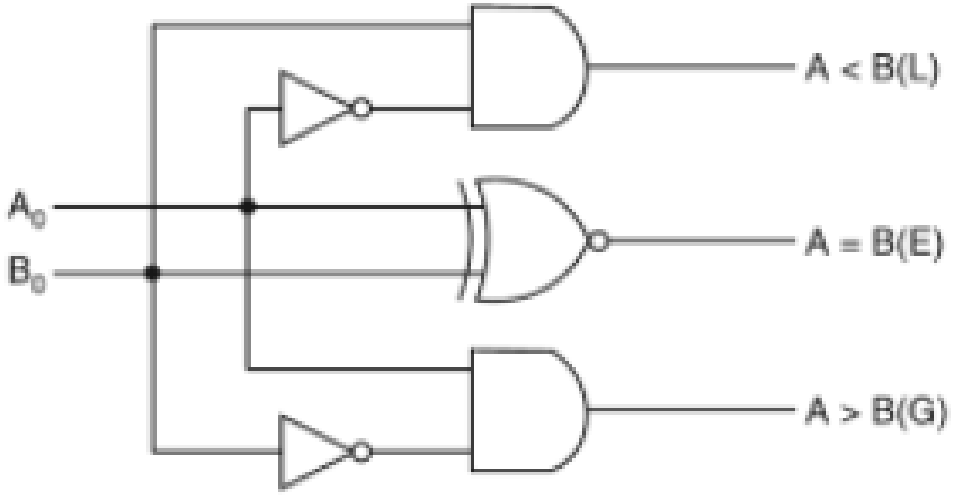
If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

The truth table and the logic diagram for the 1-bit comparator are shown in Figure. The logic expressions for G, L, and E can also be obtained from the truth table.

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit magnitude comparator:

2-bit Magnitude Comparator

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

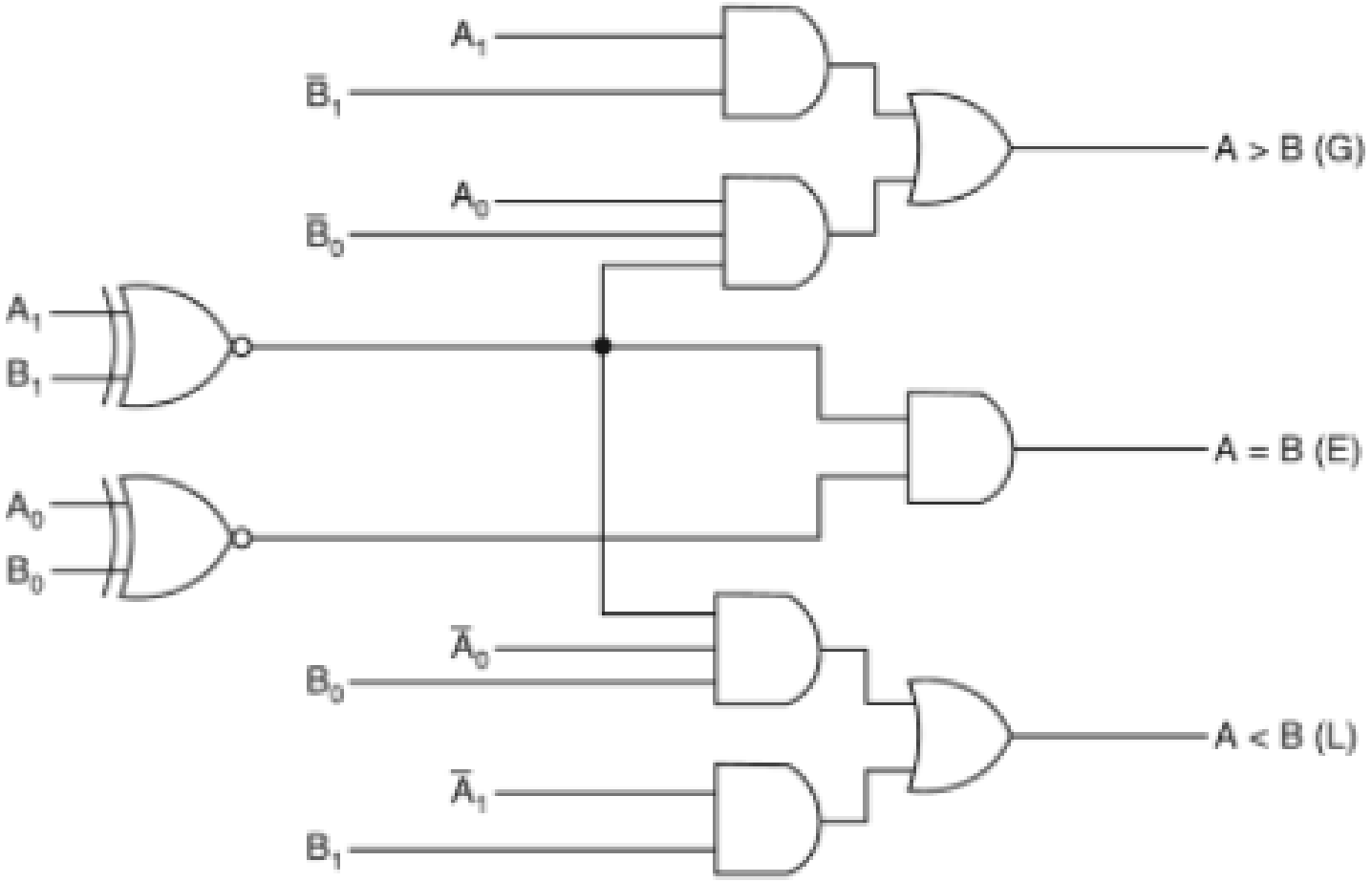
1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \bar{A}_1B_1 + (A_1 \odot B_1)\bar{A}_0B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

The logic diagram for a 2-bit comparator is as shown in Figure.



Logic diagram of a 2-bit magnitude comparator.

4-bit Magnitude Comparator

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) : G = A_3\overline{B}_3 + (A_3 \odot B_3)A_2\overline{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\overline{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\overline{B}_0$$

Similarly, the logic expression for $A < B$ can be written as

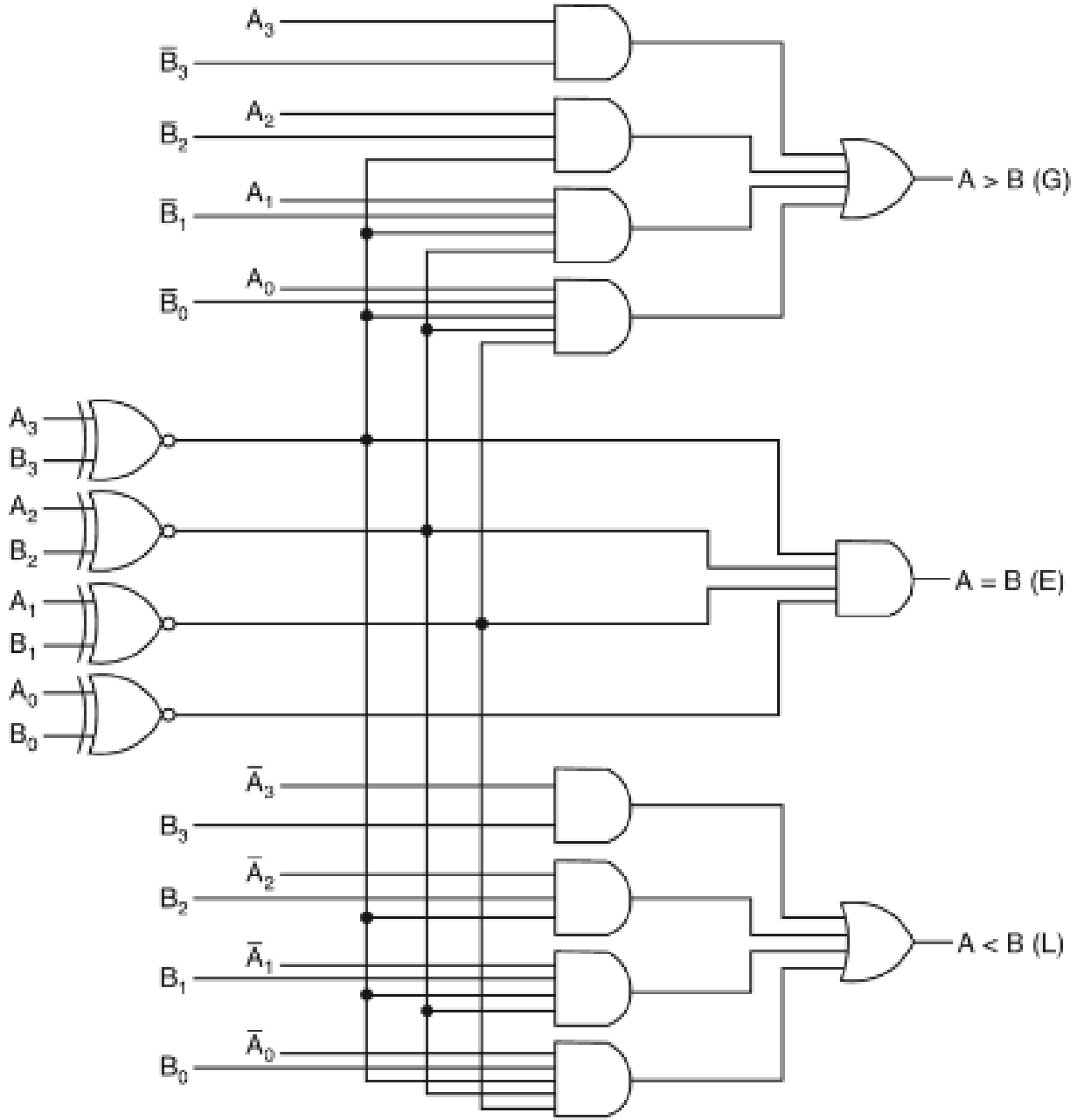
$$(A < B): L = \overline{A}_3B_3 + (A_3 \odot B_3)\overline{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A}_0B_0$$

If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

$$(A = B): E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Figure shows the logic diagram of a comparator that implements the logic we have described. Note that, it provides three active-HIGH outputs: $A > B$, $A < B$, and $A = B$.



Logic diagram of a 4-bit magnitude comparator.

Code Converters

- A code converter is a logic circuit whose inputs are bit patterns representing numbers (or characters) in one code and whose outputs are the corresponding representations in a different code.
- Code converters are usually multiple output circuits.
- To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

Design of a 4-bit Binary-to-Gray Code Converter.

- The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code. There are 16 possible combinations of 4-bit binary input and all of them are valid. Hence no don’t cares.
- The 4-bit binary code and the corresponding output Gray code are shown in the conversion table.

4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Conversion table

4-bit binary-to-Gray code converter

- From the conversion table, we observe that expressions for the outputs G_4, G_3, G_2, and G_1 are as follows:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$$
$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$$

K-map for G_4

B_4B_3 \ B_2B_1	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12 1	13 1	15 1	14 1
10	8 1	9 1	11 1	10 1

$$G_4 = B_4$$

K-map for G_3

B_4B_3 \ B_2B_1	00	01	11	10
00	0	1	3	2
01	4 1	5 1	7 1	6 1
11	12	13	15	14
10	8 1	9 1	11 1	10 1

$$G_3 = \overline{B}_4 B_3 + B_4 \overline{B}_3$$

K-map for G_2

B_4B_3 \ B_2B_1	00	01	11	10
00	0	1	3 1	2 1
01	4 1	5 1	7	6
11	12 1	13 1	15	14
10	8	9	11 1	10 1

$$G_2 = \overline{B}_3 B_2 + B_3 \overline{B}_2$$

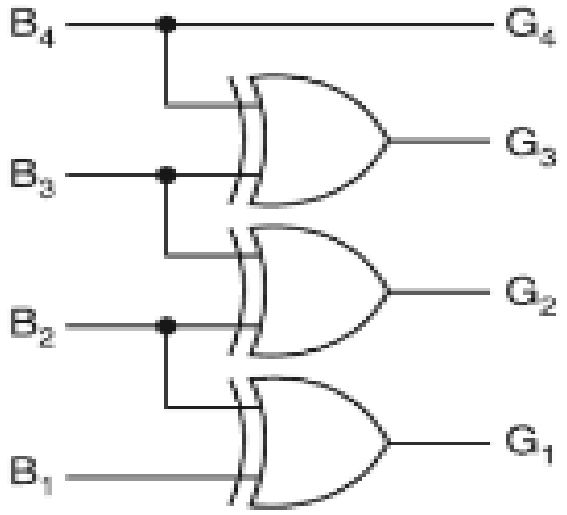
K-map for G_1

B_4B_3 \ B_2B_1	00	01	11	10
00	0	1 1	3	2 1
01	4	5 1	7	6 1
11	12	13 1	15	14 1
10	8	9 1	11	10 1

$$G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1$$

- The minimal expressions for the outputs obtained from the K-map are:

$$G_4 = B_4$$
$$G_3 = \overline{B}_4 B_3 + B_4 \overline{B}_3 = B_4 \oplus B_3$$
$$G_2 = \overline{B}_3 B_2 + B_3 \overline{B}_2 = B_3 \oplus B_2$$
$$G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1 = B_2 \oplus B_1$$



Logic diagram

4-bit binary-to-Gray code converter.

Design of a 4-bit Gray-to-Binary Code Converter.

- The input to the 4-bit Gray-to-binary code converter circuit is a 4-bit Gray and the output is a 4-bit binary code. There are 16 possible combinations of 4-bit Gray input and all of them are valid. Hence no don’t cares.
- The 4-bit Gray code and the corresponding output binary code are shown in the conversion table.

4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Conversion table

4-bit Gray-to-binary code converter.

- From the conversion table, we observe that expressions for the outputs B_4, B_3, B_2, and B_1 are as follows:

$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

K-map for B_4

G_4G_3 \ G_2G_1	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12 [1]	13 [1]	15 [1]	14 [1]
10	8 [1]	9 [1]	11 [1]	10 [1]

$B_4 = G_4$

K-map for B_3

G_4G_3 \ G_2G_1	00	01	11	10
00	0	1	3	2
01	4 [1]	5 [1]	7 [1]	6 [1]
11	12	13	15	14
10	8 [1]	9 [1]	11 [1]	10 [1]

$B_3 = G_4 \oplus G_3$

K-map for B_2

G_4G_3 \ G_2G_1	00	01	11	10
00	0	1	3 [1]	2 [1]
01	4 [1]	5 [1]	7	6
11	12	13	15 [1]	14 [1]
10	8 [1]	9 [1]	11	10

$B_2 = G_4 \oplus G_3 \oplus G_2$

K-map for B_1

G_4G_3 \ G_2G_1	00	01	11	10
00	0	1 [1]	3	2 [1]
01	4 [1]	5	7 [1]	6
11	12	13 [1]	15	14 [1]
10	8 [1]	9	11 [1]	10

$B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$

- The minimal expressions for the outputs obtained from the K-map are:

$$B_4 = G_4$$

$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$

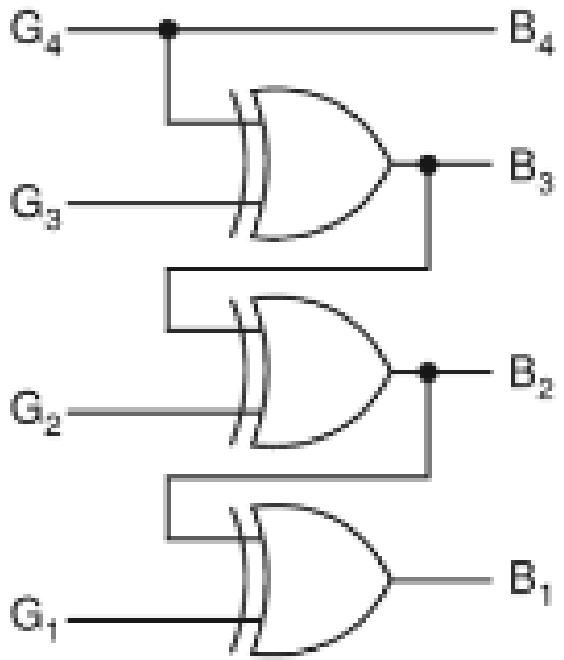
$$B_2 = \bar{G}_4 \bar{G}_3 G_2 + \bar{G}_4 G_3 \bar{G}_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$

$$= \bar{G}_4 (\bar{G}_3 G_2 + G_3 \bar{G}_2) + G_4 (\bar{G}_3 \bar{G}_2 + G_3 G_2)$$

$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$

$$
\begin{aligned}
B_1 &= \bar{G}_4\bar{G}_3\bar{G}_2G_1 + \bar{G}_4\bar{G}_3G_2\bar{G}_1 + \bar{G}_4G_3\bar{G}_2\bar{G}_1 + \bar{G}_4G_3G_2G_1 \\
&\quad + G_4\bar{G}_3\bar{G}_2\bar{G}_1 + G_4\bar{G}_3G_2G_1 + G_4G_3G_2\bar{G}_1 + G_4G_3\bar{G}_2G_1 \\
&= \bar{G}_4\bar{G}_3(\bar{G}_2G_1 + G_2\bar{G}_1) + \bar{G}_4G_3(\bar{G}_2\bar{G}_1 + G_2G_1) \\
&\quad + G_4\bar{G}_3(\bar{G}_2\bar{G}_1 + G_2G_1) + G_4G_3(\bar{G}_2G_1 + G_2\bar{G}_1) \\
&= \bar{G}_4\bar{G}_3(G_2 \oplus G_1) + \bar{G}_4G_3\overline{(G_2 \oplus G_1)} + G_4\bar{G}_3\overline{(G_2 \oplus G_1)} + G_4G_3(G_2 \oplus G_1) \\
&= (\bar{G}_4\bar{G}_3 + G_4G_3)(G_2 \oplus G_1) + (\bar{G}_4G_3 + G_4\bar{G}_3)\overline{(G_2 \oplus G_1)} \\
&= \overline{(G_4 \oplus G_3)}(G_2 \oplus G_1) + (G_4 \oplus G_3)\overline{(G_2 \oplus G_1)} \\
&= G_4 \oplus G_3 \oplus G_2 \oplus G_1 \\
&= B_2 \oplus G_1
\end{aligned}
$$

$$B_4 = G_4$$
$$B_3 = G_4 \oplus G_3$$
$$B_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1$$



Logic diagram

4-bit Gray-to-binary code converter.

THANK YOU