Gate-Level Minimization

By:

Ratnakar Dash

August 17, 2022

Table of Contents

Gate-Level Minimization

- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit

The Map Method

- The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented.
- The map method presented here 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 Map Method (contd.)

- The complexity of the digital logic gates
  - the complexity of the algebraic expression
- Logic minimization
  - algebraic approaches: lack specific rules
  - the Karnaugh map (or K-map)
    - a simple straight forward procedure
    - a pictorial form of a truth table
    - applicable if the number of variables < 7
- A diagram made up of squares
  - each square represents one minterm.

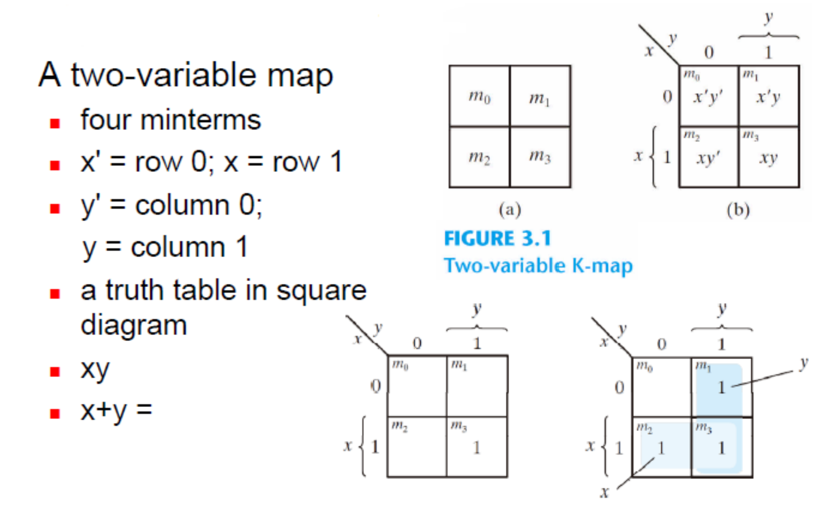
Boolean function

Boolean function

- sum of minterms
- sum of products (or product of sum) in the simplest form
- a minimum number of terms
- a minimum number of literals
- The simplified expression may not be unique

Two-Variable Map

Two-Variable Map



A two-variable map

- four minterms
- x' = row 0; x = row 1
- y' = column 0;

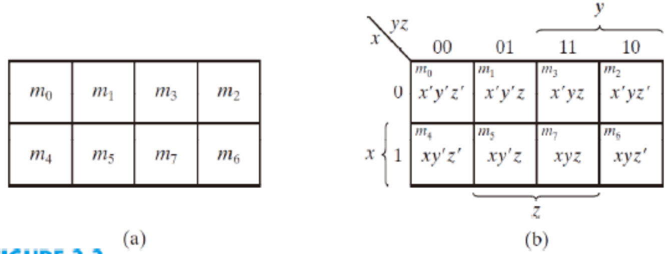
  y = column 1
- a truth table in square diagram
- xy
- x+y =

FIGURE 3.1
Two-variable K-map

❶
②
❸
❹
❺

Three-Variable Map

- eight minterms
- the Gray code sequence
- any two adjacent squares in the map differ by only on variable
  - primed in one square and unprimed in the other
  - e.g. m_5 and m_7 can be simplified
  - $m_5 + m_7 = xy'z + xyz = xz\,(y'+y) = xz$

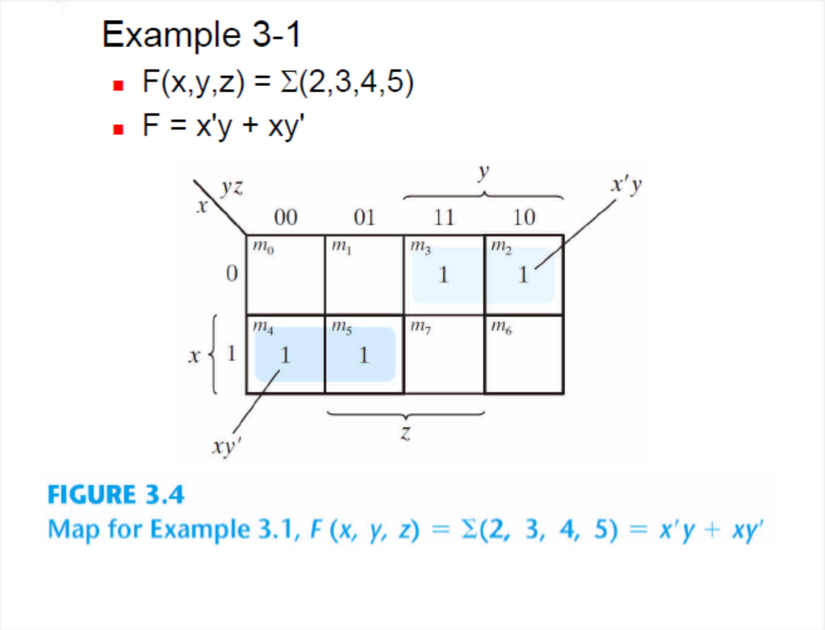


(a)

(b)

FIGURE 3.3
Three-variable K-map

Example



Example 3-1

- F(x,y,z) = Σ(2,3,4,5)
- F = x'y + xy'

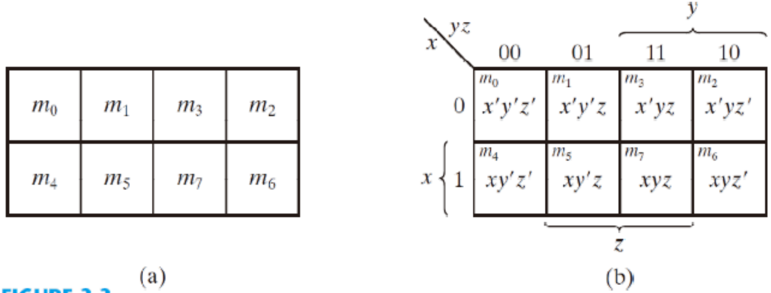
x \ yz	00	01	11	10
0	m_0	m_1	m_3 1	m_2 1
1	m_4 1	m_5 1	m_7	m_6

FIGURE 3.4
Map for Example 3.1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

1
2
3
4
5

Example (Cont...)

- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
- $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

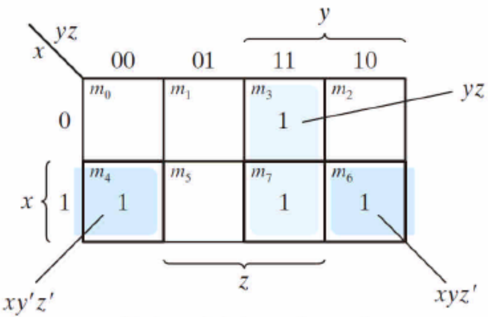
x \ yz	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

FIGURE 3.3
Three-variable K-map

Example (Cont...)

Example 3-2

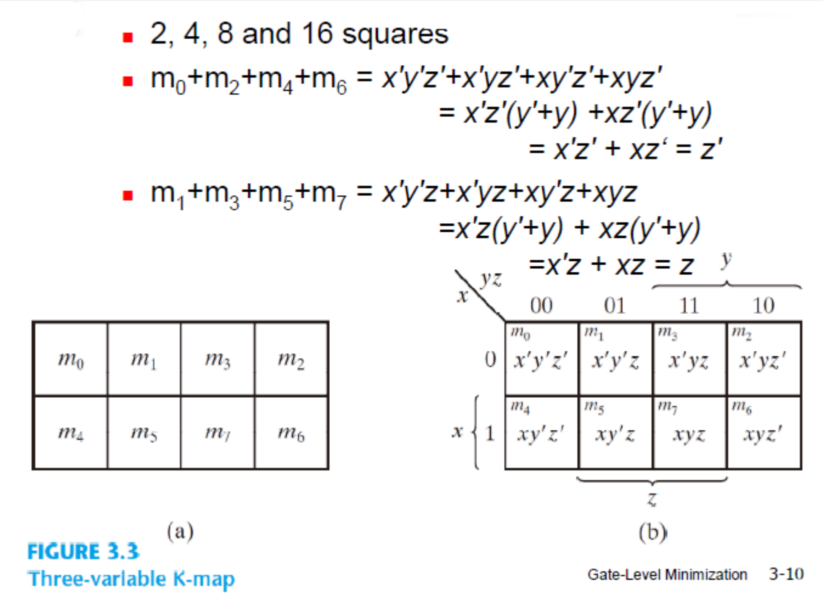
- F(x,y,z) = Σ(3,4,6,7) = *yz*+ *xz'*



Note: xy'z' + xyz' = xz'

FIGURE 3.5
Map for Example 3.2, $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Four adjacent squares



- 2, 4, 8 and 16 squares
- $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz'$
  $= x'z'(y'+y) +xz'(y'+y)$
  $= x'z' + xz' = z'$
- $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz$
  $=x'z(y'+y) + xz(y'+y)$
  $=x'z + xz = z$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

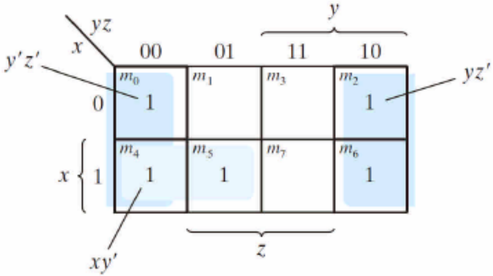
$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

(b)

FIGURE 3.3
Three-variable K-map

Example 3-3

- $F(x,y,z) = \Sigma(0,2,4,5,6)$
- $F = z' + xy'$

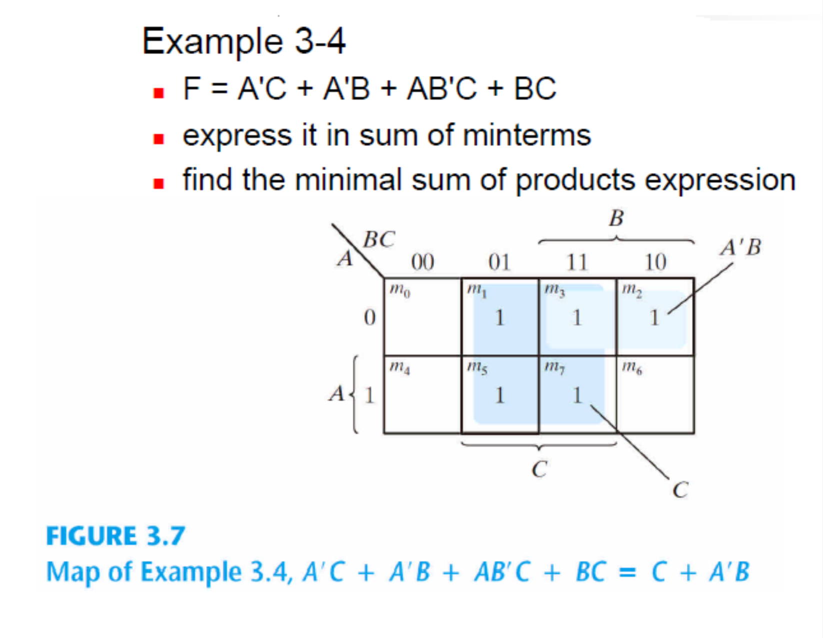


Note: $y'z' + yz' = z'$

FIGURE 3.6
Map for Example 3.3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

1
2
3
4
5

Example



Example 3-4

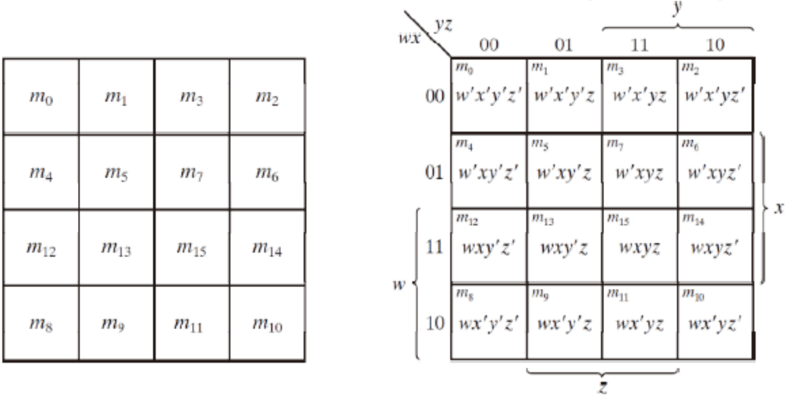
- F = A'C + A'B + AB'C + BC
- express it in sum of minterms
- find the minimal sum of products expression

$A \backslash BC$	00	01	11	10
0	m_0	m_1 1	m_3 1	m_2 1
1	m_4	m_5 1	m_7 1	m_6

FIGURE 3.7
Map of Example 3.4, $A'C + A'B + AB'C + BC = C + A'B$

Four-Variable Map

- 16 minterms
- combinations of 2, 4, 8, and 16 adjacent squares



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}

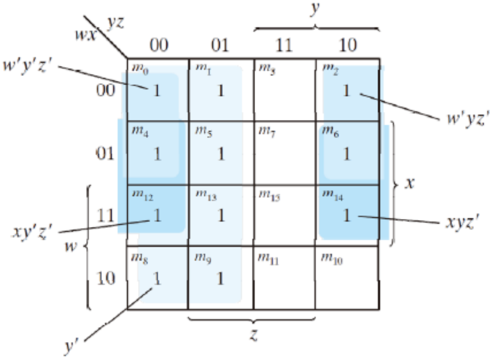
(a)

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} $wx'yz'$

(b)

Example

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



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

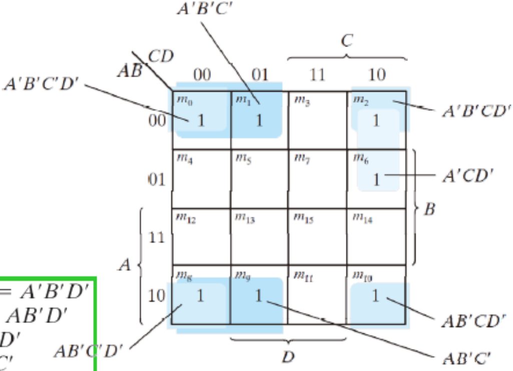
Note: $w'y'z' + w'yz' = w'z'$
$xy'z' + xyz' = xz'$

- $F = y'+w'z'+xz'$

FIGURE 3.9
Map for Example 3.5, $F(w, x, y, z) = \Sigma(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

Simplify the Boolean function

$$F = A'B'C' + B'CD' + A'B'CD' + AB'C'$$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

FIGURE 3.10
Map for Example 3.6, $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

❶
②
❸
❹
❺

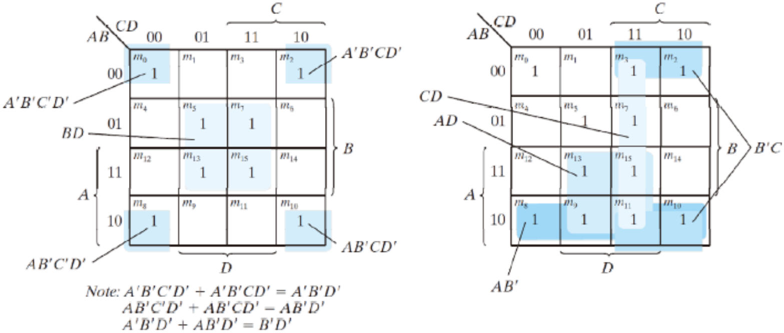
Prime Implicants

- all the minterms are covered
- minimize the number of terms
- a prime implicant: a product term obtained by combining the maximum possible number of adjacent squares (combining all possible maximum numbers of squares)
- essential: a minterm is covered by only one prime implicant
- the essential P.I. must be included

Example

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

- the simplified expression may not be unique
- F = BD+B'D'+CD+AD = BD+B'D'+CD+AB'
  = BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'



(a) Essential prime implicants BD and B'D'

(b) Prime implicants CD, B'C, AD, and AB'

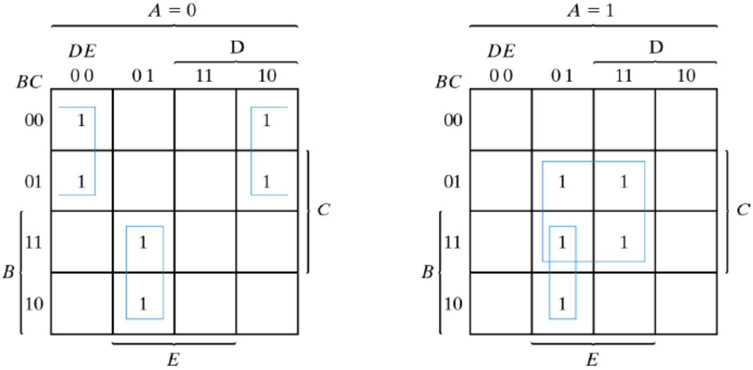
Five-Variable Map

Inverse

- Map for more than four variables becomes complicated
- five-variable map: two four-variable map (one on the top of the other)

Five-Variable Map

- $F = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$



A = 0

BC \ DE	00	01	11	10
00	1			1
01	1			1
11		1		
10		1		

A = 1

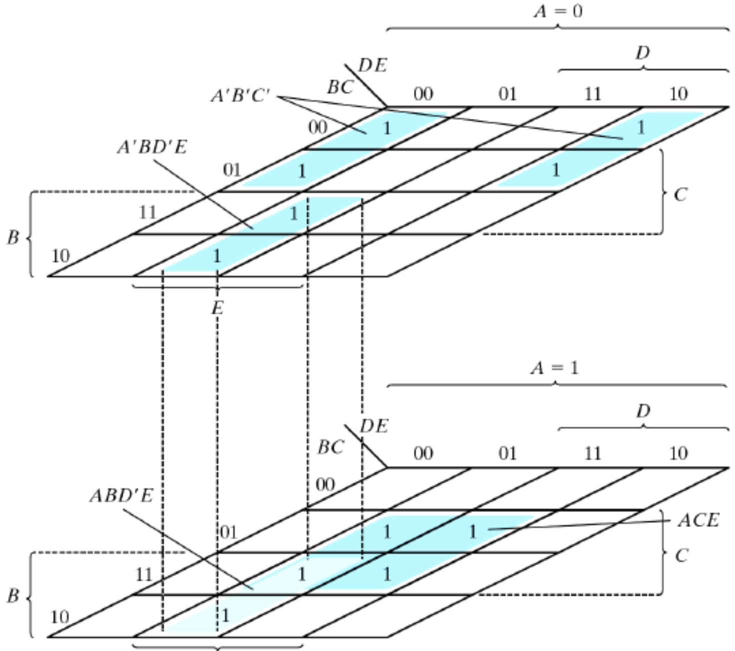
BC \ DE	00	01	11	10
00				
01		1	1	
11		1	1	
10		1		

Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

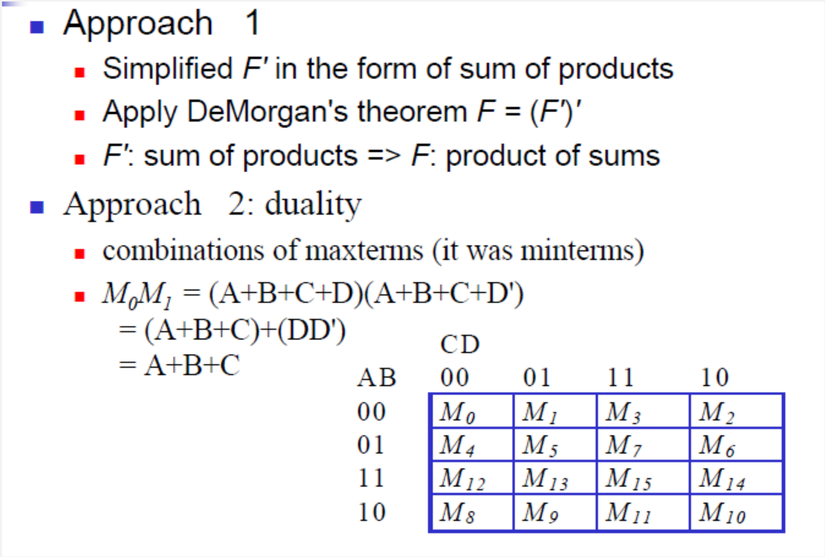
$F = A'B'E' + BD'E + ACE$

Example

❶
②
❸
❹
❺



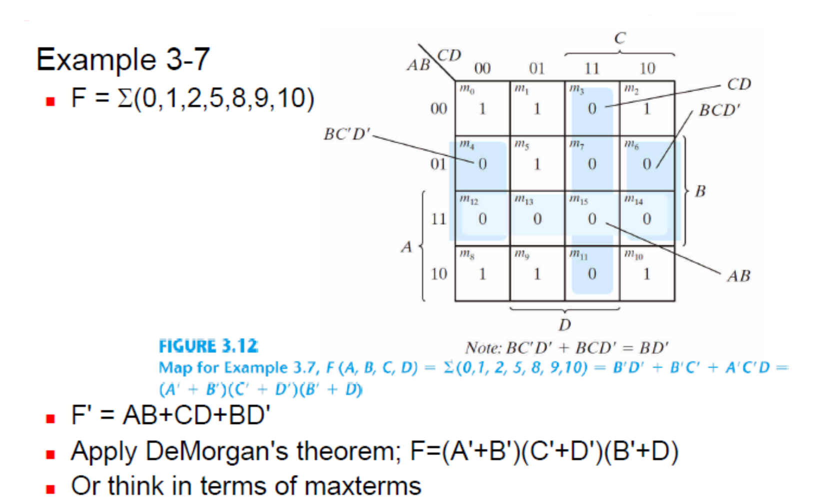
Product of Sums Simplification



- Approach 1
  - Simplified F' in the form of sum of products
  - Apply DeMorgan's theorem $F = (F')'$
  - F': sum of products => F: product of sums
- Approach 2: duality
  - combinations of maxterms (it was minterms)
  - $M_0M_1 = (A+B+C+D)(A+B+C+D')$
    $= (A+B+C)+(DD')$
    $= A+B+C$

	CD			
AB	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

Example



Example 3-7

- F = Σ(0,1,2,5,8,9,10)

AB\CD	00	01	11	10
00	m_0 1	m_1 1	m_3 0	m_2 1
01	m_4 0	m_5 1	m_7 0	m_6 0
11	m_{12} 0	m_{13} 0	m_{15} 0	m_{14} 0
10	m_8 1	m_9 1	m_{11} 0	m_{10} 1

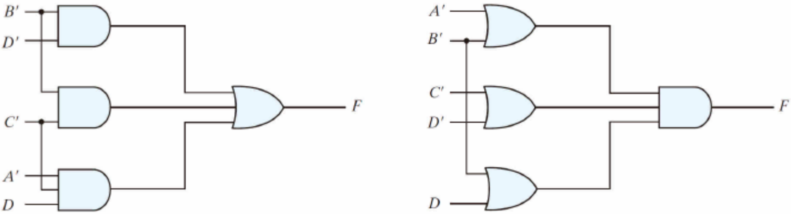
Note: $BC'D' + BCD' = BD'$

FIGURE 3.12
Map for Example 3.7, $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$

- F' = AB+CD+BD'
- Apply DeMorgan's theorem; F=(A'+B')(C'+D')(B'+D)
- Or think in terms of maxterms

Example

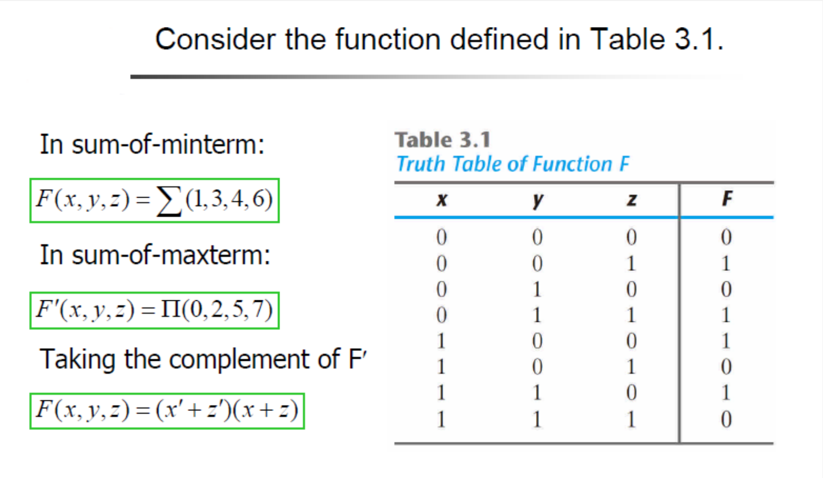
- Gate implementation of the function of Example 3-7



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

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

Example



Consider the function defined in Table 3.1.

In sum-of-minterm:

$$\boxed{F(x,y,z)=\sum(1,3,4,6)}$$

In sum-of-maxterm:

$$\boxed{F'(x,y,z)=\Pi(0,2,5,7)}$$

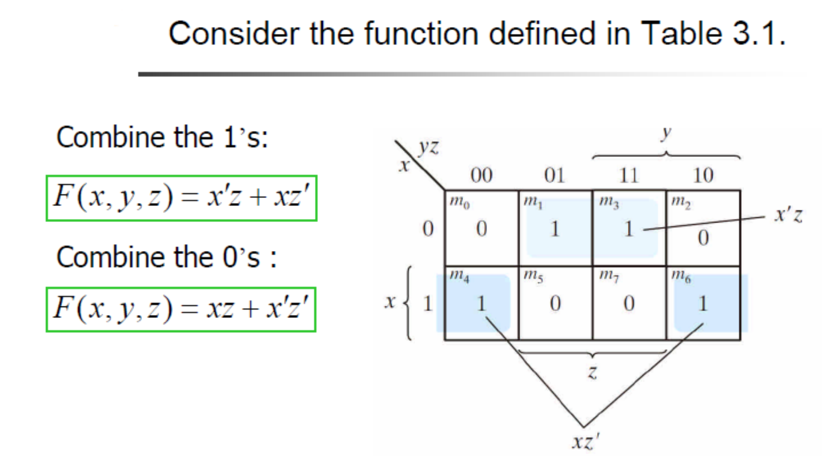
Taking the complement of F′

$$\boxed{F(x,y,z)=(x'+z')(x+z)}$$

Table 3.1
Truth Table of Function F

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Example



Consider the function defined in Table 3.1.

Combine the 1's:

$$\boxed{F(x,y,z) = x'z + xz'}$$

Combine the 0's :

$$\boxed{F(x,y,z) = xz + x'z'}$$

$x \backslash yz$	00	01	11	10
0	m_0 0	m_1 1	m_3 1	m_2 0
1	m_4 1	m_5 0	m_7 0	m_6 1

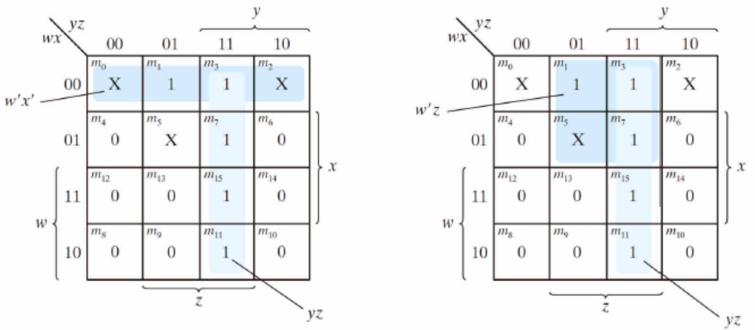
Five-Variable Map

Inverse

- The value of a function is not specified for certain combinations of variables
- BCD; 1010-1111: don't care
- The don't care conditions can be utilized in logic minimization (can be implemented as 0 or 1)
- Example 3-8
  - $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$
  - $d(w, x, y, z) = \sum(0, 2, 5)$

Example

- Figure 3.15(a) : F = yz + w'x'
- Figure 3.15(b) : F = yz + w'z
- $F = \Sigma(0,1,2,3,7,11,15)$; $F = \Sigma(1,3,5,7,11,15)$
- either expression is acceptable

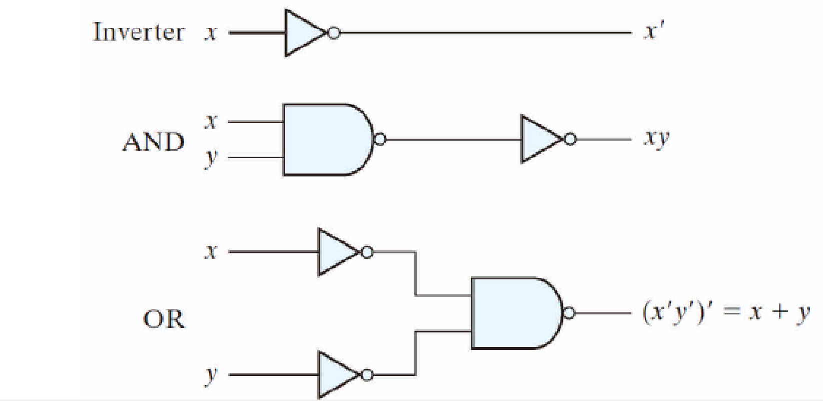


(a) $F = yz + w'x'$

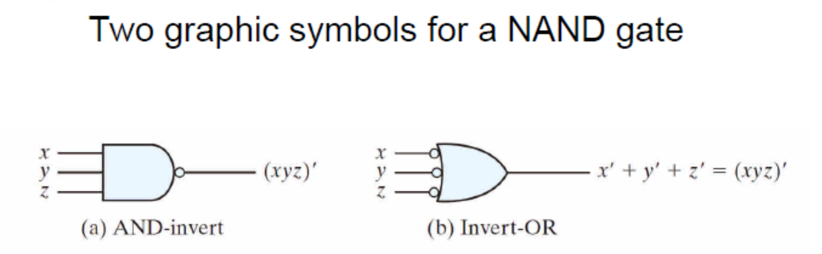
(b) $F = yz + w'z$

NAND and NOR Implementation

- NAND gate is a universal gate
- can implement any digital system



Example

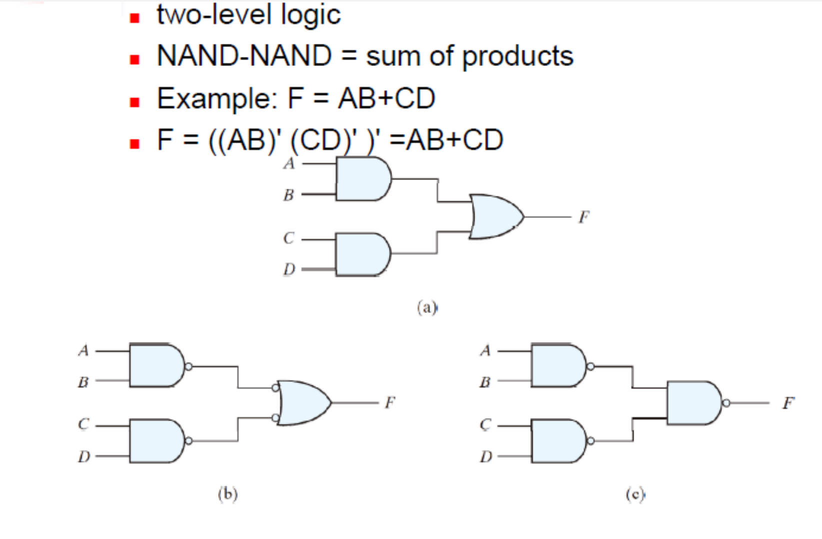


Two graphic symbols for a NAND gate

(a) AND-invert

(b) Invert-OR

Two-level Implementation



- two-level logic
- NAND-NAND = sum of products
- Example: F = AB+CD
- F = ((AB)' (CD)')' =AB+CD

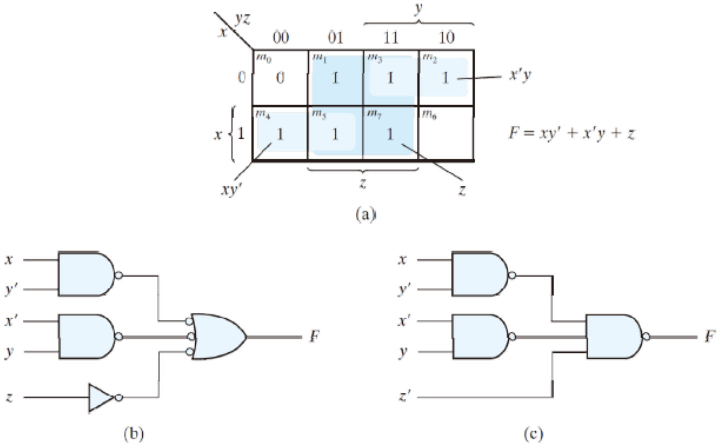
(a)

(b)

(c)

Example

$$F(x,y,z) = \sum(1,2,3,4,5,7) \qquad F(x,y,z) = xy' + x'y + z$$



(a)

(b)

(c)

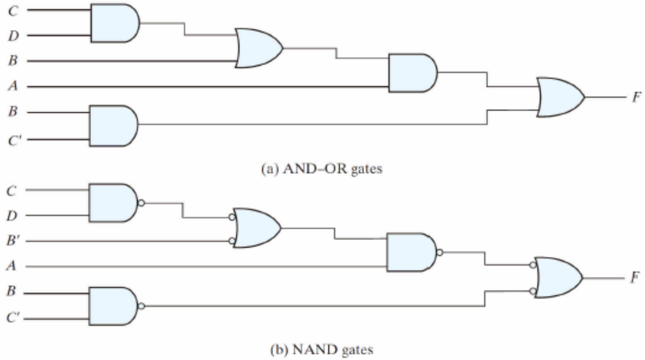
The procedure

- simplified in the form of sum of products
- a NAND gate for each product term; the inputs to each NAND gate are the literals of the term
- a single NAND gate for the second sum term
- A single literal requires an inverter in the first level

Multilevel NAND Circuits

- Boolean function implementation
- AND-OR logic $\Longrightarrow$ NAND-NAND logic
- AND $\Longrightarrow$ NAND + inverter
- OR: inverter + OR = NAND

Example

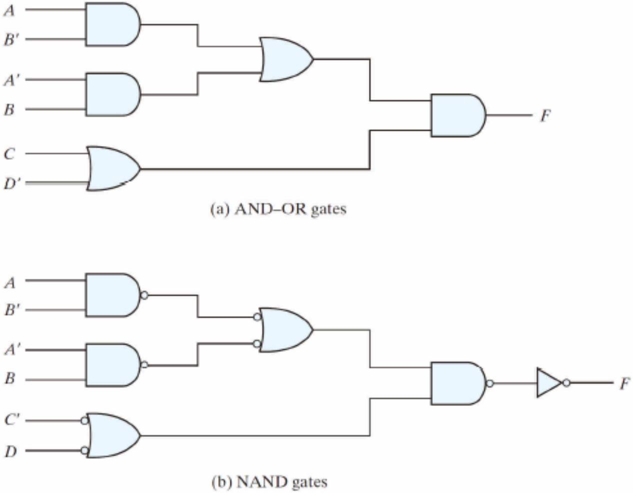


(a) AND–OR gates

(b) NAND gates

FIGURE 3.20
Implementing $F = A(CD + B) + BC'$

Example



(a) AND–OR gates

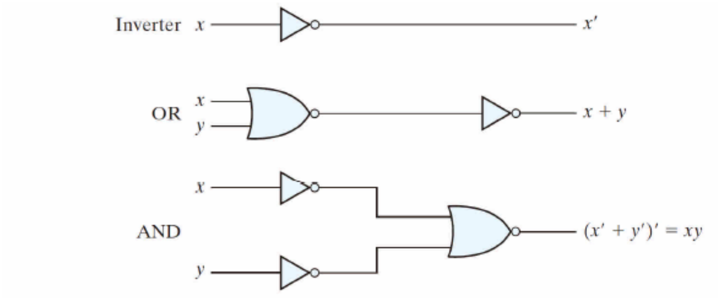
(b) NAND gates

FIGURE 3.21
Implementing $F = (AB' + A'B)\ (C + D')$

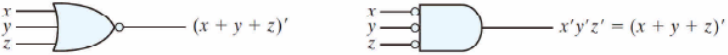
NOR Implementation

- NOR function is the dual of NAND function
- The NOR gate is also universal

Example



Two graphic symbols for a NOR gate



(a) OR-invert

(b) Invert-AND

FIGURE 3.23

Two graphic symbols for the NOR gate

Example: F = (A + B)(C + D)E

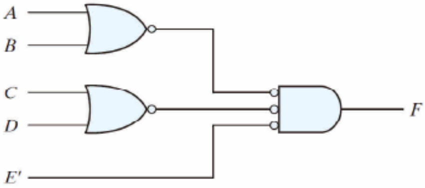


FIGURE 3.24

Implementing $F = (A \mid B)(C \mid D)E$

Two graphic symbols for a NOR gate

Example: F = (AB′ +A′B)(C + D′)

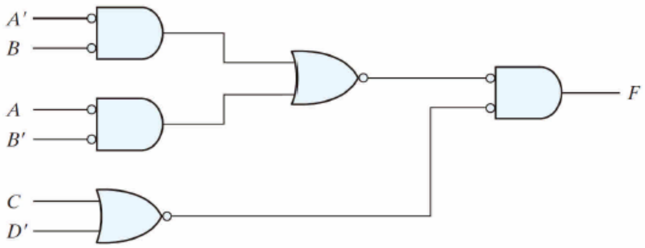


FIGURE 3.25

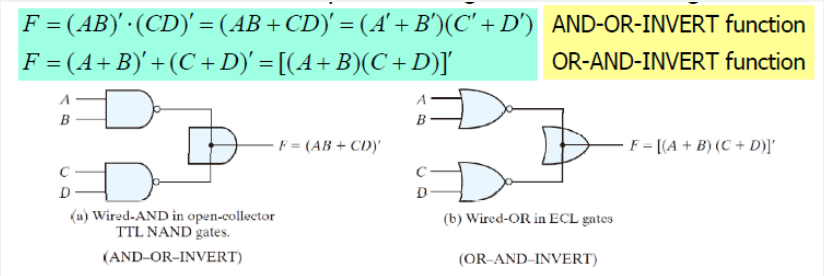
Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

Other Two-level Implementations

Wired logic

- a wire connection between the outputs of two gates
- open-collector TTL NAND gates: wired-AND logic
- the NOR output of ECL gates: wired-OR logic

Other Two-level Implementations



$F = (AB)' \cdot (CD)' = (AB + CD)' = (A' + B')(C' + D')$ AND-OR-INVERT function

$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$ OR-AND-INVERT function

(a) Wired-AND in open-collector TTL NAND gates.

(AND–OR–INVERT)

(b) Wired-OR in ECL gates

(OR–AND–INVERT)

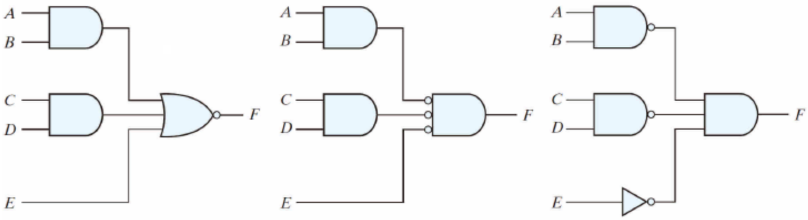
Nondegenerate Forms

- 16 possible combinations of two-level forms
- The eight nondegenerate forms
  - AND-OR, OR-AND, NAND-NAND, NOR-NOR, NOR-OR, NAND-AND, OR-AND, AND-OR
  - AND-OR and NAND-NAND = sum of products
  - OR-AND and NOR-NOR = product of sums
  - NOR-OR, NAND-AND, OR-AND, AND-OR = ?

AND-OR-Invert Implementation

- AND-OR-INVERT (AOI) Implementation
  - NAND-AND = AND-NOR = AOI
  - $F = (AB+CD+E)'$
  - $F' = AB+CD+E$(sum of products)

AND-OR-Invert Implementation



(a) AND–NOR

(b) AND–NOR

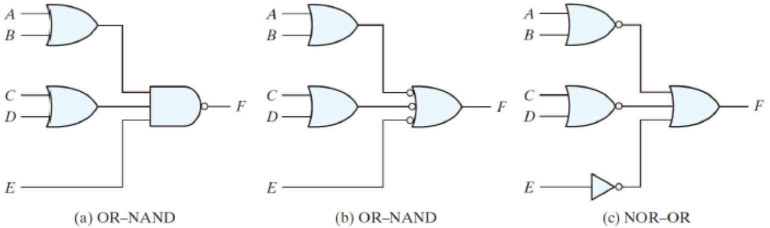
(c) NAND–AND

FIGURE 3.27
AND–OR–INVERT circuits, $F = (AB + CD + E)'$

OR-AND-INVERT (OAI) Implementation

- OR-AND-INVERT (OAI) Implementation
  - OR-NAND = NOR-OR = OAI
  - $F = ((A+B)(C+D)E)'$
  - $F' = (A+B)(C+D)E$ (product of sums)

OR-AND-INVERT (OAI) Implementation



(a) OR–NAND

(b) OR–NAND

(c) NOR–OR

FIGURE 3.28
OR–AND–INVERT circuits, $F = [(A + B)(C + D)E]'$

Tabular Summary and Examples

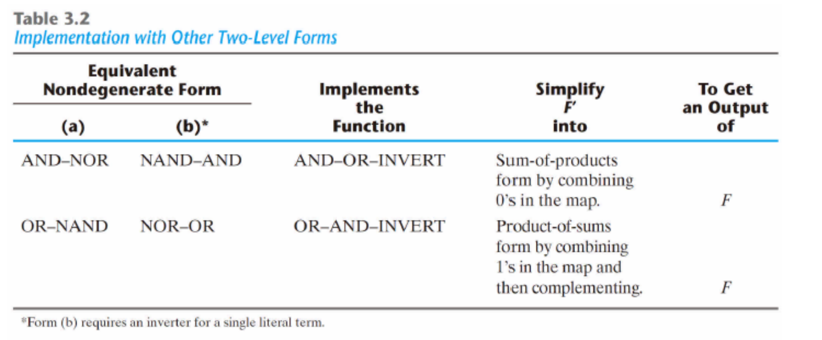
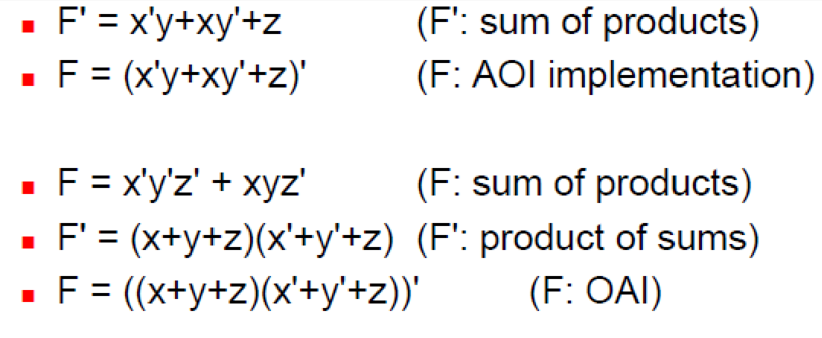


Table 3.2
Implementation with Other Two-Level Forms

Equivalent Nondegenerate Form		Implements the Function	Simplify F' into	To Get an Output of
(a)	**(b)***			
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.	F

*Form (b) requires an inverter for a single literal term.

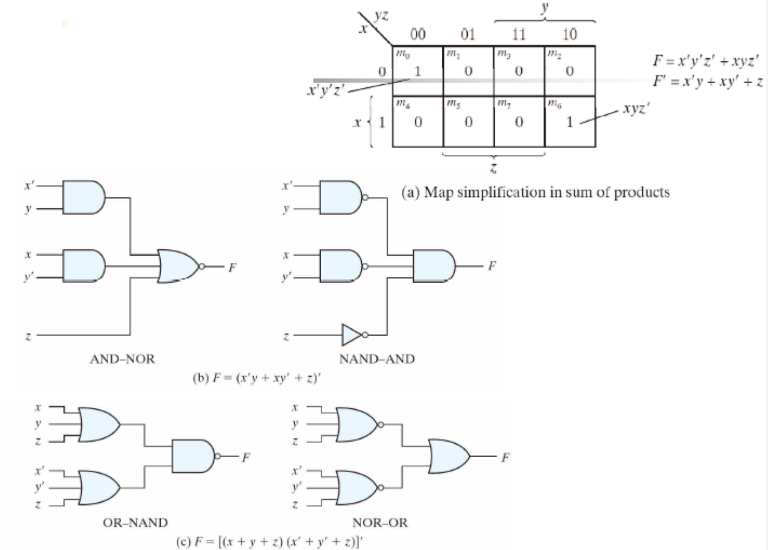
Example



- $F' = x'y+xy'+z$ (F': sum of products)
- $F = (x'y+xy'+z)'$ (F: AOI implementation)

- $F = x'y'z' + xyz'$ (F: sum of products)
- $F' = (x+y+z)(x'+y'+z)$ (F': product of sums)
- $F = ((x+y+z)(x'+y'+z))'$ (F: OAI)

Example



$x \backslash yz$	00	01	11	10
0	m_0 1	m_1 0	m_3 0	m_2 0
1	m_4 0	m_5 0	m_7 0	m_6 1

$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

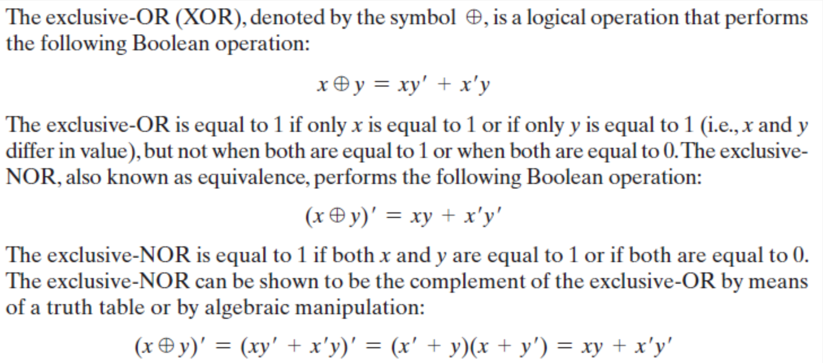
(a) Map simplification in sum of products

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

(c) $F = [(x + y + z)(x' + y' + z)]'$

FIGURE 3.29
Other two-level implementations

EXCLUSIVE-OR FUNCTION



The exclusive-OR (XOR), denoted by the symbol $\oplus$, is a logical operation that performs the following Boolean operation:

$$x \oplus y = xy' + x'y$$

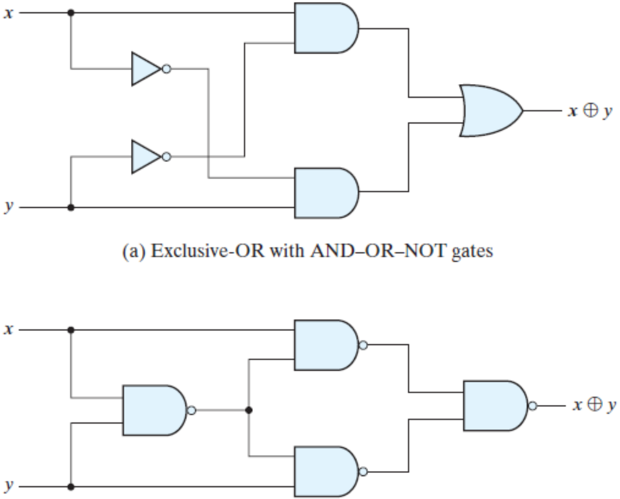
The exclusive-OR is equal to 1 if only x is equal to 1 or if only y is equal to 1 (i.e., x and y differ in value), but not when both are equal to 1 or when both are equal to 0. The exclusive-NOR, also known as equivalence, performs the following Boolean operation:

$$(x \oplus y)' = xy + x'y'$$

The exclusive-NOR is equal to 1 if both x and y are equal to 1 or if both are equal to 0. The exclusive-NOR can be shown to be the complement of the exclusive-OR by means of a truth table or by algebraic manipulation:

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

Example



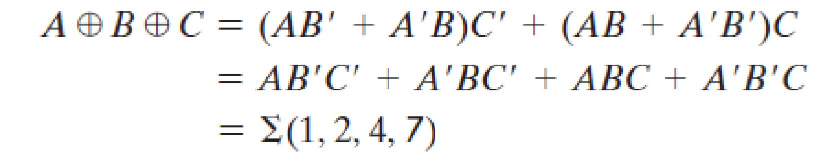
(a) Exclusive-OR with AND–OR–NOT gates

(b) Exclusive-OR with NAND gates

FIGURE 3.30
Exclusive-OR implementations

Odd function

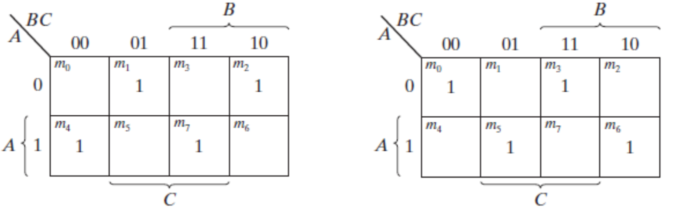
- The exclusive-OR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ symbol with its equivalent Boolean expression.



$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$

Example

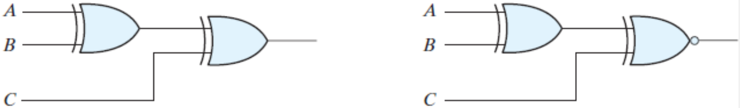
$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D)$$
$$= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'$$
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$



(a) Odd function $F = A \oplus B \oplus C$

(b) Even function $F = (A \oplus B \oplus C)'$

FIGURE 3.31
Map for a three-variable exclusive-OR function

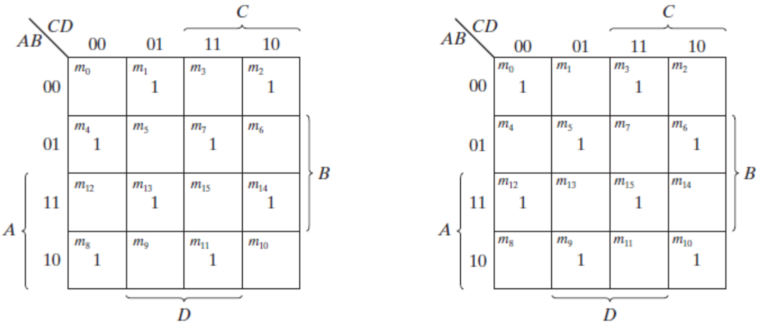


(a) 3-input odd function

(b) 3-input even function

FIGURE 3.32
Logic diagram of odd and even functions

Example



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

(a) Odd function $F = A \oplus B \oplus C \oplus D$

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

(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

FIGURE 3.33
Map for a four-variable exclusive-OR function

Parity Generation and Checking

- The exclusive-OR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ symbol with its equivalent Boolean expression.

Example

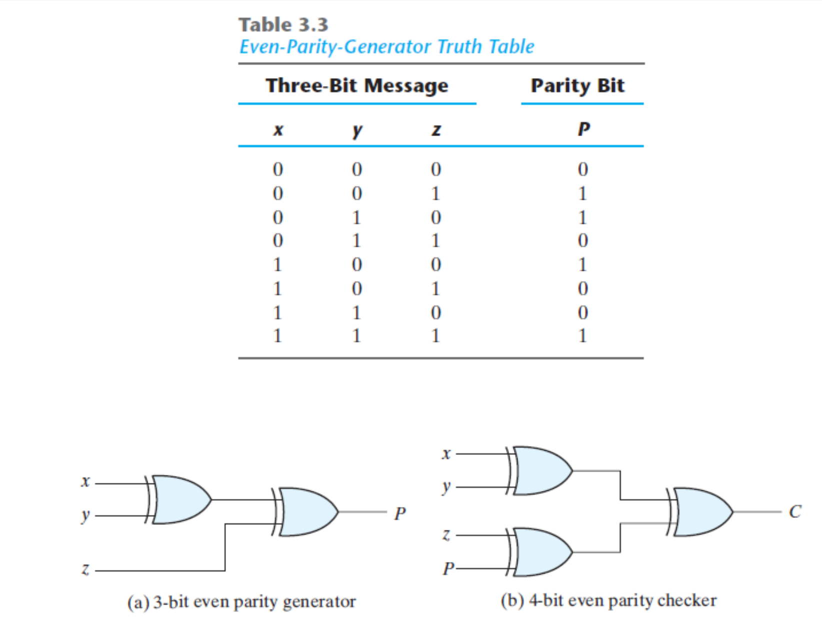


Table 3.3
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	***y***	***z***	***P***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) 3-bit even parity generator

(b) 4-bit even parity checker

1
2
3
4
5

HARDWARE DESCRIPTION LANGUAGE

- A hardware description language (HDL) is a computer-based language that describes the hardware of digital systems in a textual form.
- It resembles an ordinary computer programming language, such as C, but is specifically oriented to describing hardware structures and the behavior of logic circuits.
- It can be used to represent logic diagrams,truth tables, Boolean expressions, and complex abstractions of the behavior of a digital system.

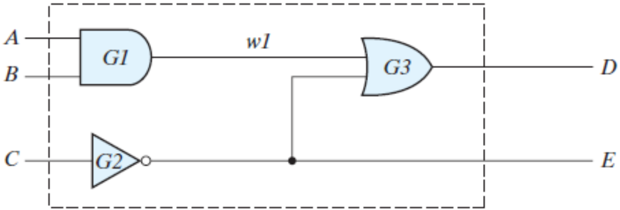
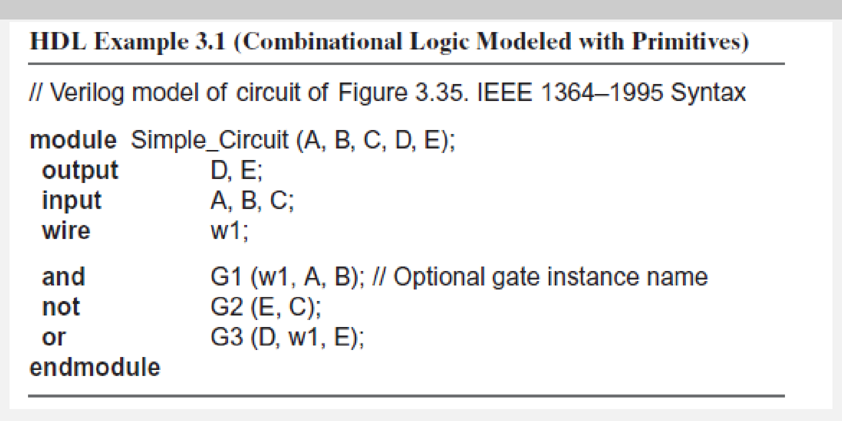


FIGURE 3.35
Circuit to demonstrate an HDL



HDL Example 3.1 (Combinational Logic Modeled with Primitives)

```
// Verilog model of circuit of Figure 3.35. IEEE 1364–1995 Syntax

module  Simple_Circuit (A, B, C, D, E);
  output          D, E;
  input           A, B, C;
  wire            w1;

  and             G1 (w1, A, B); // Optional gate instance name
  not             G2 (E, C);
  or              G3 (D, w1, E);
endmodule
```

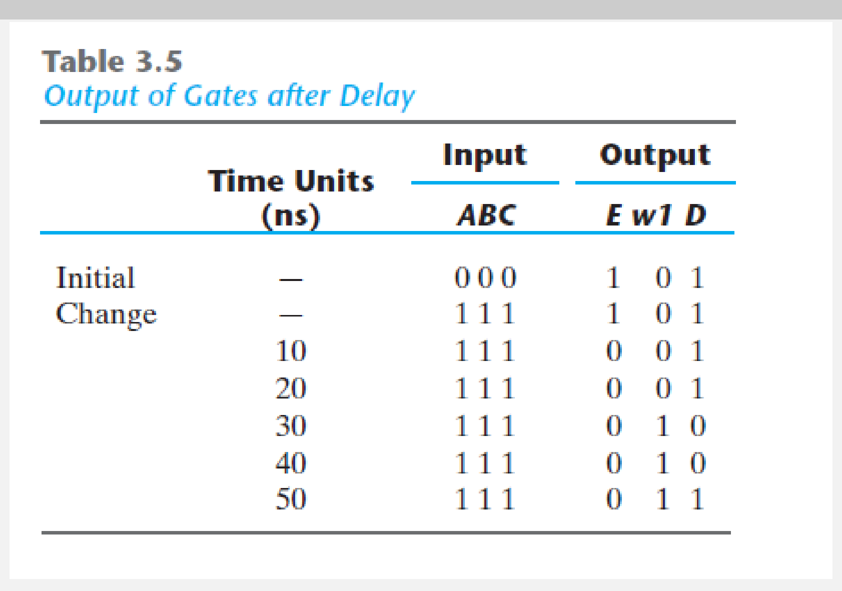
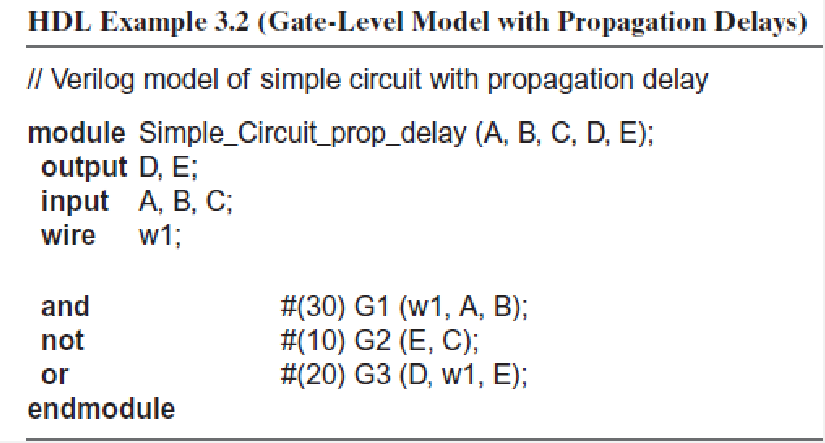


Table 3.5
Output of Gates after Delay

	Time Units (ns)	Input	Output		
		ABC	*E*	*w1*	*D*
Initial	—	0 0 0	1	0	1
Change	—	1 1 1	1	0	1
	10	1 1 1	0	0	1
	20	1 1 1	0	0	1
	30	1 1 1	0	1	0
	40	1 1 1	0	1	0
	50	1 1 1	0	1	1

Gate Delays



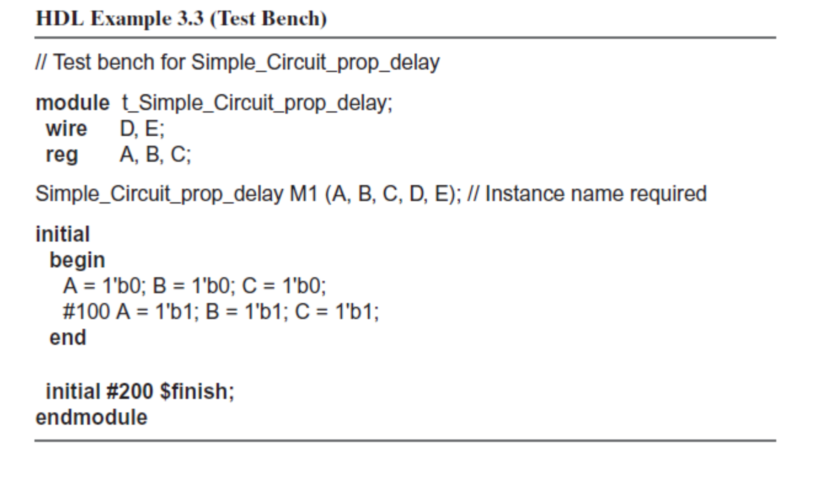
HDL Example 3.2 (Gate-Level Model with Propagation Delays)

```
// Verilog model of simple circuit with propagation delay

module Simple_Circuit_prop_delay (A, B, C, D, E);
 output D, E;
 input  A, B, C;
 wire   w1;

 and               #(30) G1 (w1, A, B);
 not               #(10) G2 (E, C);
 or                #(20) G3 (D, w1, E);
endmodule
```

Gate Delays



HDL Example 3.3 (Test Bench)

```
// Test bench for Simple_Circuit_prop_delay

module  t_Simple_Circuit_prop_delay;
 wire     D, E;
 reg      A, B, C;

Simple_Circuit_prop_delay M1 (A, B, C, D, E); // Instance name required

initial
  begin
    A = 1'b0; B = 1'b0; C = 1'b0;
    #100 A = 1'b1; B = 1'b1; C = 1'b1;
  end

 initial #200 $finish;
endmodule
```

Gate Delays

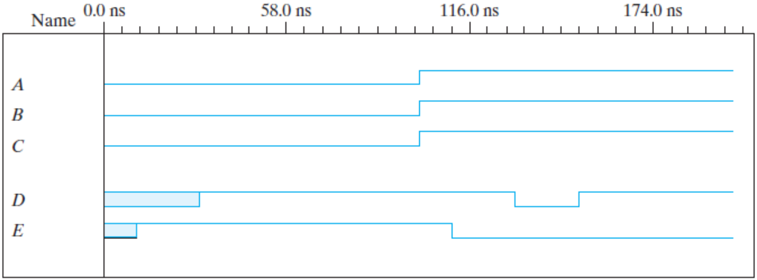
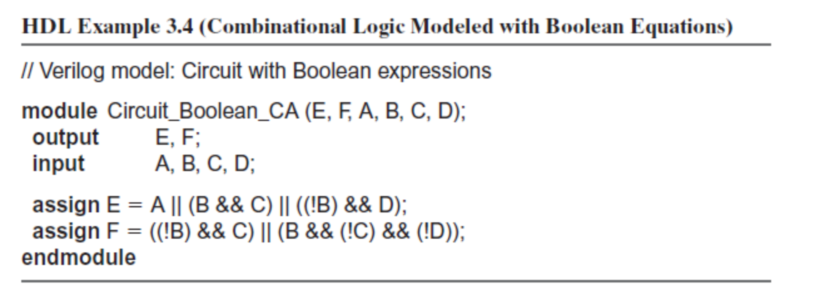


FIGURE 3.36
Simulation output of HDL Example 3.3

Gate Delays



HDL Example 3.4 (Combinational Logic Modeled with Boolean Equations)

```
// Verilog model: Circuit with Boolean expressions

module Circuit_Boolean_CA (E, F, A, B, C, D);
  output        E, F;
  input         A, B, C, D;

  assign E = A || (B && C) || ((!B) && D);
  assign F = ((!B) && C) || (B && (!C) && (!D));
endmodule
```

User-Defined Primitives

- The logic gates used in Verilog descriptions with keywords and, or, etc., are defined by the system and are referred to as system primitives.
- The user can create additional primitives by defining them in tabular form. These types of circuits are referred to as user-defined primitives (UDPs).

User-Defined Primitives

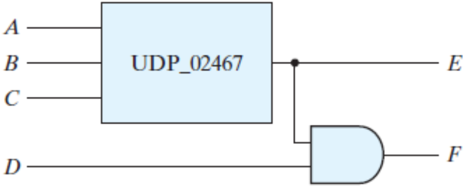


FIGURE 3.37
Schematic for *Circuit with_UDP_02467*

References I

[1] M. M. Mano and M. D. Ciletti, *Digital Design (4th Edition)*. USA: Prentice-Hall, Inc., 2006.

Thank you!