

3. Gate-Level Minimization

3.1 Introduction

- The complexity of the digital logic gates that implement a Boolean function is implemented.
- Truth Table -> K-map

3.2 The Map Method

● Two-Variable Map

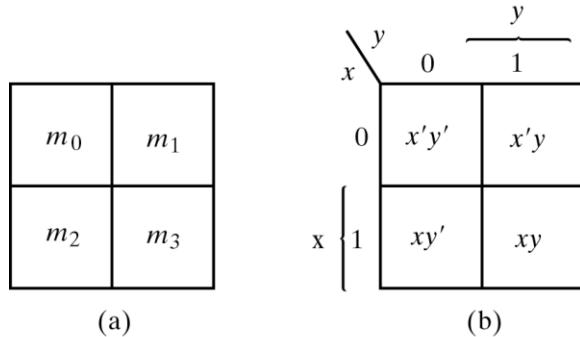


Fig. 3-1 Two-variable Map

● $m_1 + m_2 + m_3 = x'y' + xy' + xy = x + y$

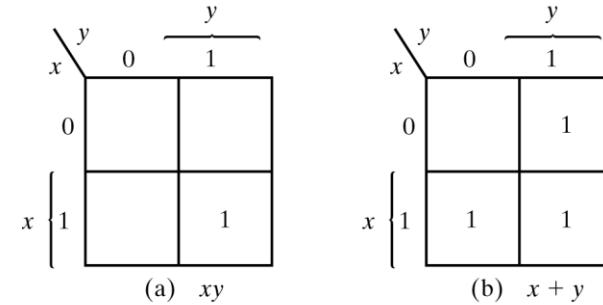


Fig. 3-2 Representation of Functions in the Map

● Three-Variable Map

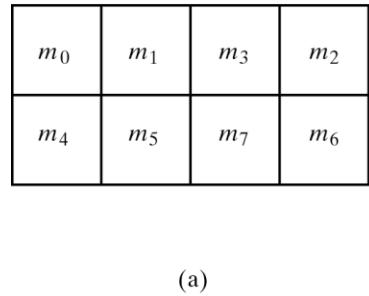


Fig. 3-3 Three-variable Map

3.2 The Map Method

- Ex 3-1) Simplify the Boolean function, $F(x, y, z) = \Sigma(2, 3, 4, 5)$

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

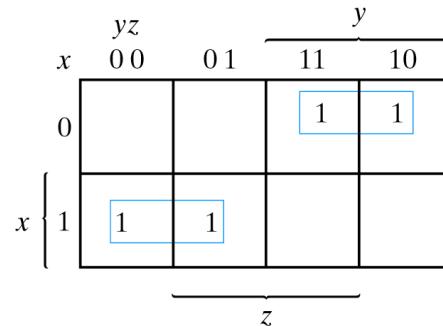


Fig. 3-4 Map for Example 3-1; $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

- Ex 3-4) Given Boolean function, $F = A'C + A'B + AB'C + BC$

a) express it in sum of minterms

$$F(x, y, z) = \Sigma(1, 2, 3, 5, 7)$$

b) find the minimal sum of products

$$F = C + A'B$$

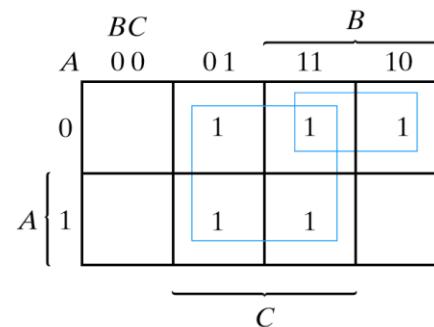
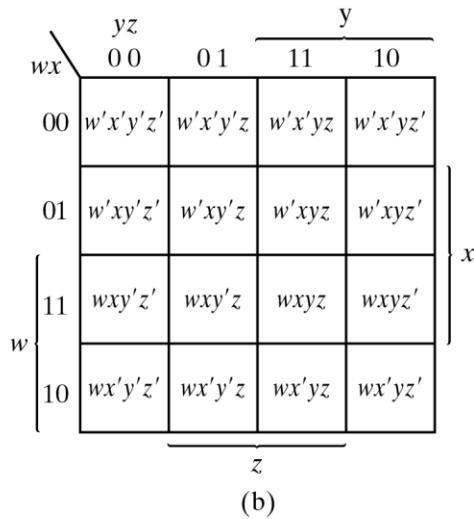


Fig. 3-7 Map for Example 3-4; $A'C + A'B + AB'C + BC = C + A'B$

3.3 Four-Variable Map

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)



(b)

Fig. 3-8 Four-variable Map

Ex 3-5) Simplify the Boolean function,

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

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

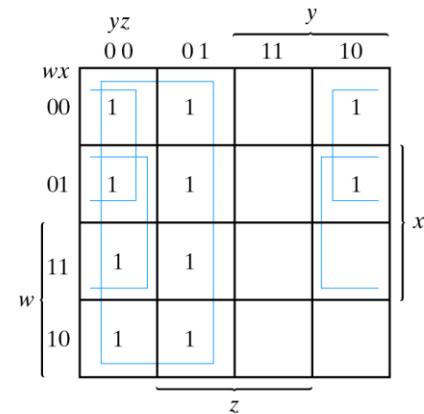
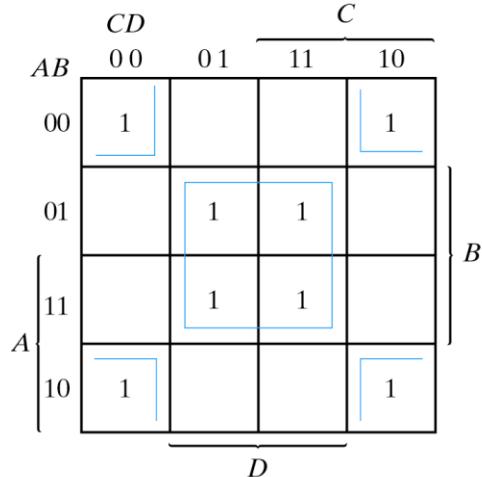


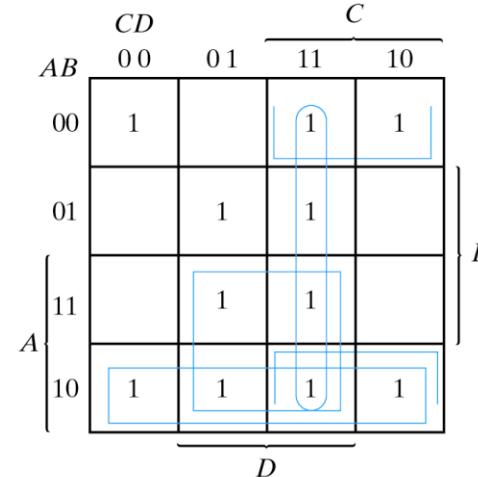
Fig. 3-9 Map for Example 3-5; $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

3.3 Four-Variable Map – Prime Implicants

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



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



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

Fig. 3-11 Simplification Using Prime Implicants

$$\begin{aligned}F &= BD + B'D' + CD + AD \\&= BD + B'D' + CD + AB' \\&= BD + B'D' + B'C + AD \\&= BD + B'D' + CD + AB'\end{aligned}$$

3.4 Five-Variable Map

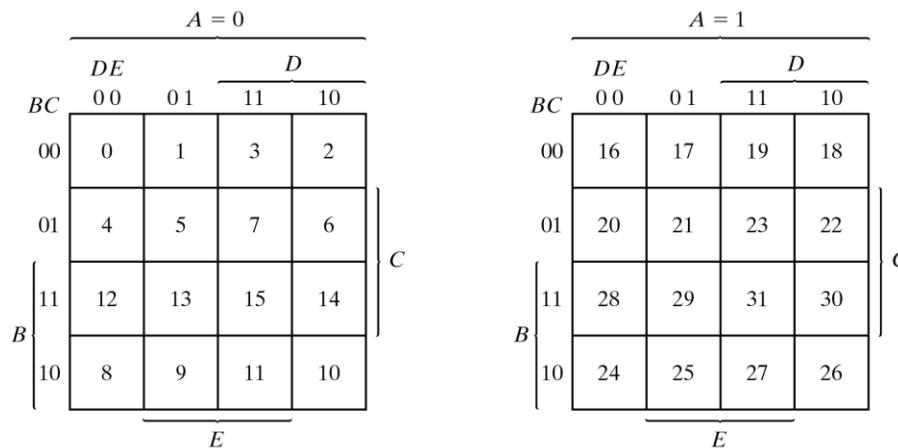


Fig. 3-12 Five-variable Map

Table 3-1
*The Relationship Between the Number of Adjacent Squares
and the Number of Literals In the Term*

K	2^k	Number of Literals in a Term in an n -variable Map			
		$n = 2$	$n = 3$	$n = 4$	$n = 5$
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

3.4 Five-Variable Map

- Ex) Simplify the Boolean function,

$$F(A,B,C,D,E)=\Sigma(0,2,4,6,9,13,21,23,25,29,31)$$

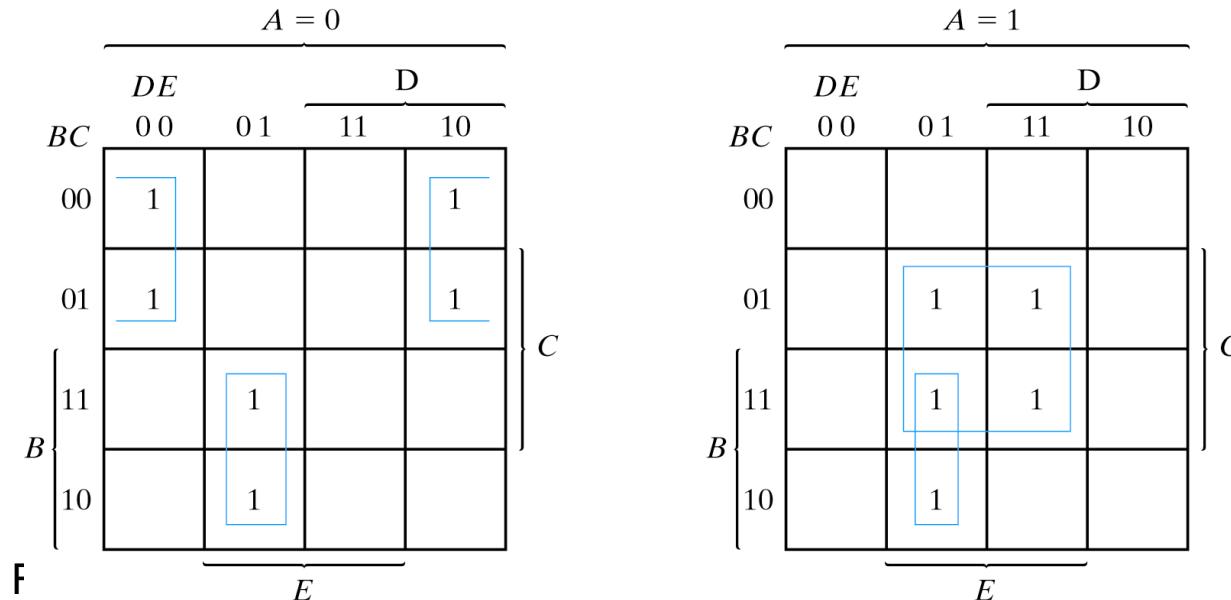


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

3.5 Product of Sums Simplification

- Ex 3-7) Simplify the Boolean function,

$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

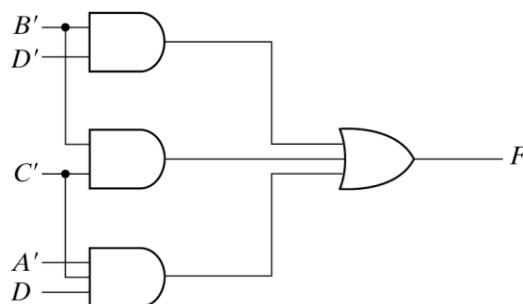
a) sum of products

$$F = B'D' + B'D' + A'C'D'$$

b) product of sum

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

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



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

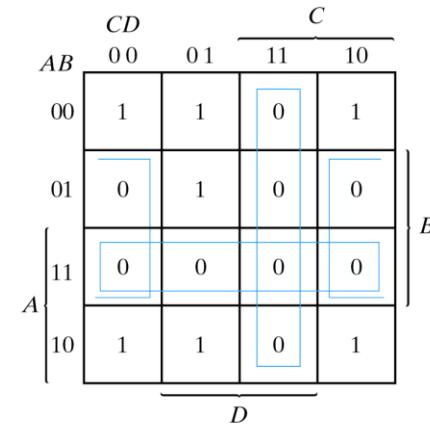
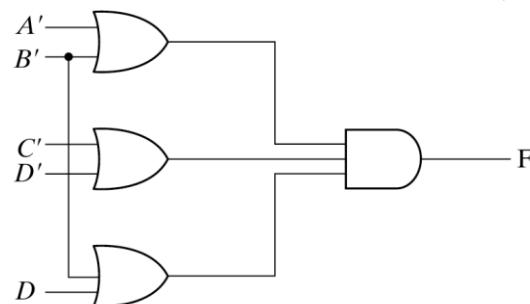


Fig. 3-14 Map for Example 3-8; $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$
= $B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$



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

Fig. 3-15 Gate Implementation of the Function of Example 3-8

3.5 Product of Sums Simplification

Table 3-2
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

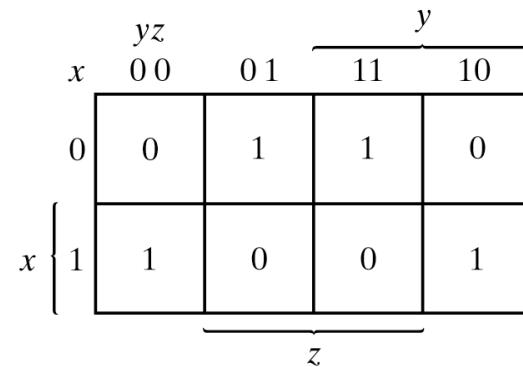


Fig. 3-16 Map for the Function of Table 3-2

● $F(x, y, z) = \Sigma(1, 3, 4, 6) = \Pi(0, 2, 5, 7)$

$$F = x'z + xz'$$

$$F' = xz + x'z'$$

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

3.6 Don't-Care Conditions

- Ex 3-8) Simplify the Boolean function, $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$
Don't-care conditions, $d(w, x, y, z) = \Sigma(0, 2, 5)$

		yz	00	01	11	y 10
		wx	X	1	1	X
wx		00	X	1	1	X
w	01	0	X	1	0	
	11	0	0	1	0	
	10	0	0	1	0	
			<u>z</u>			

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

		yz	00	01	11	y 10
		wx	X	1	1	X
wx		00	X	1	1	X
w	01	0	X	1	0	
	11	0	0	1	0	
	10	0	0	1	0	
			<u>z</u>			

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

Fig. 3-17 Example with don't-care Conditions

$$F(w, x, y, z) = yz + w'x' = \Sigma(0, 1, ,2, 3, 7, 11, 15)$$

$$F(w, x, y, z) = yz + w'z = \Sigma(1, 3, 5, 7, 11, 15)$$

3.7 NAND and NOR Implementation – NAND Circuit

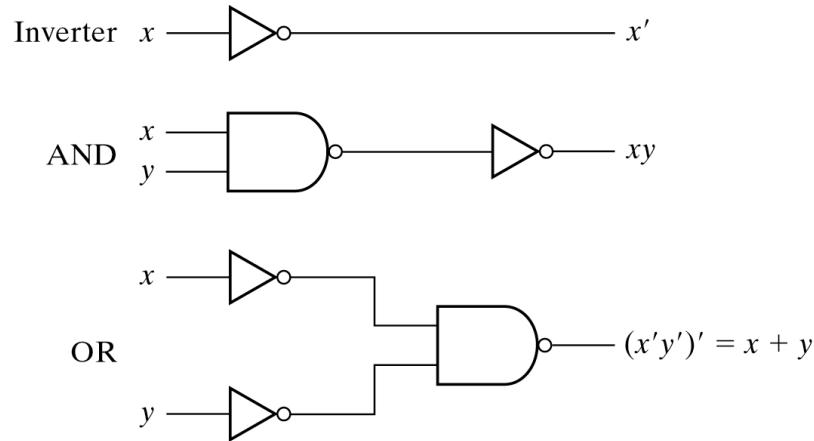


Fig. 3-18 Logic Operations with NAND Gates

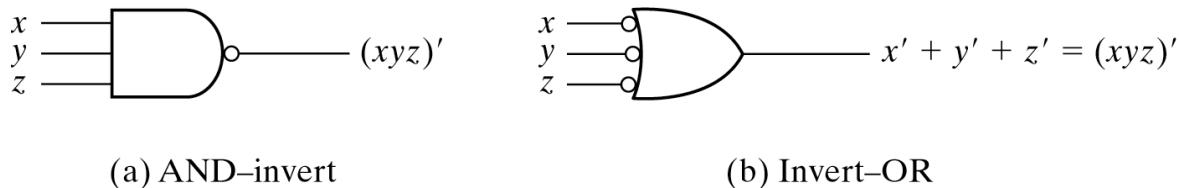
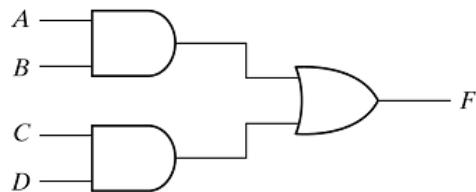


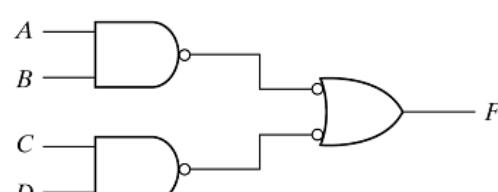
Fig. 3-19 Two Graphic Symbols for NAND Gate

3.7 NAND and NOR Implementation – Two Level Implementation

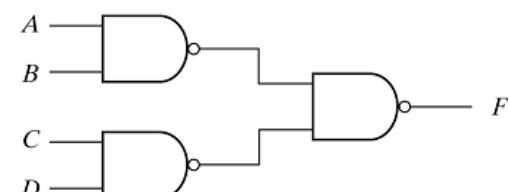
- $F = ((AB)'(CD)')' = AB + CD$



(a)



(b)

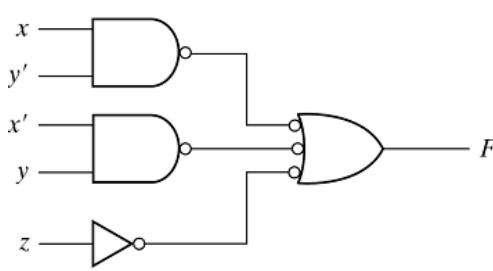
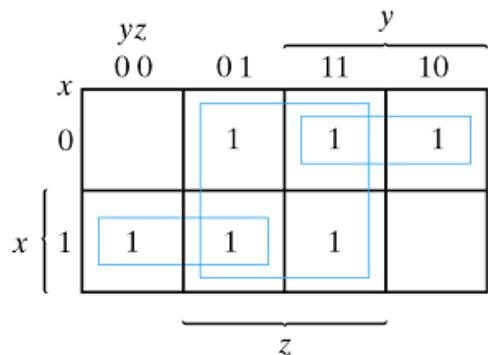


(c)

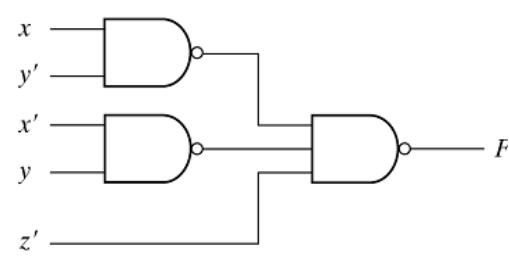
Fig. 3-20 Three Ways to Implement $F = AB + CD$

- Ex 3-10) Implement the following Boolean function with NAND gates:

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



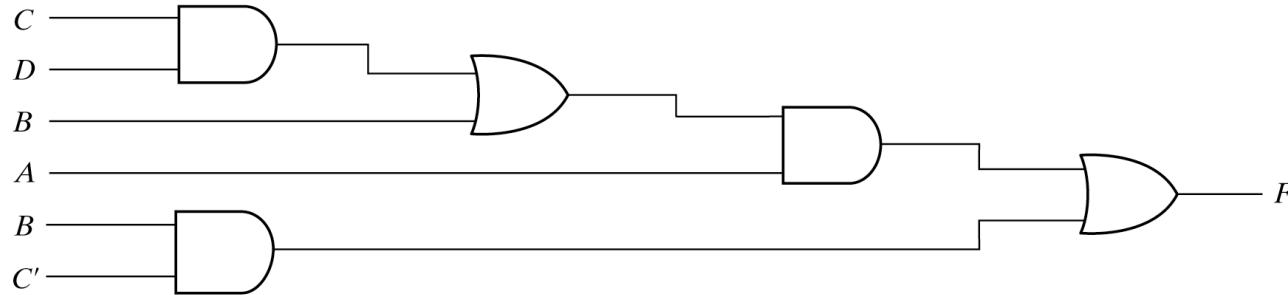
(b)



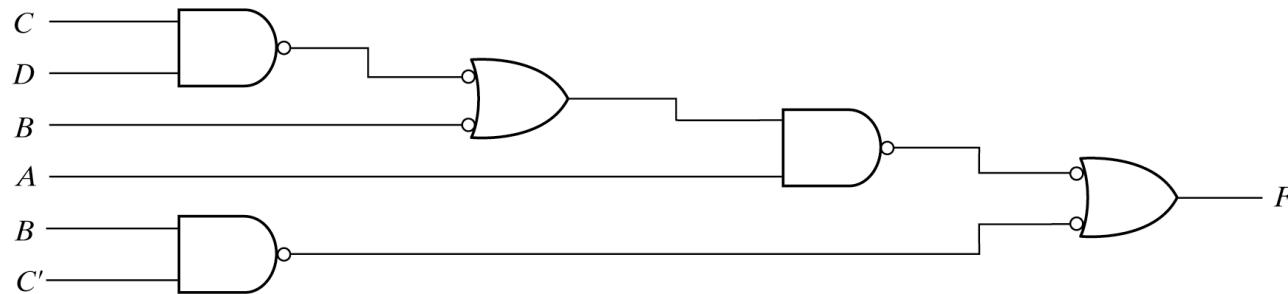
(c)

Fig. 3-21 Solution to Example 3-10

3.7 NAND and NOR Implementation – Multilevel NAND Circuit



(a) AND-OR gates



(a) NAND gates

Fig. 3-22 Implementing $F = A(CD + B) + BC'$

3.7 NAND and NOR Implementation – NOR Implementation

NOR Implementation

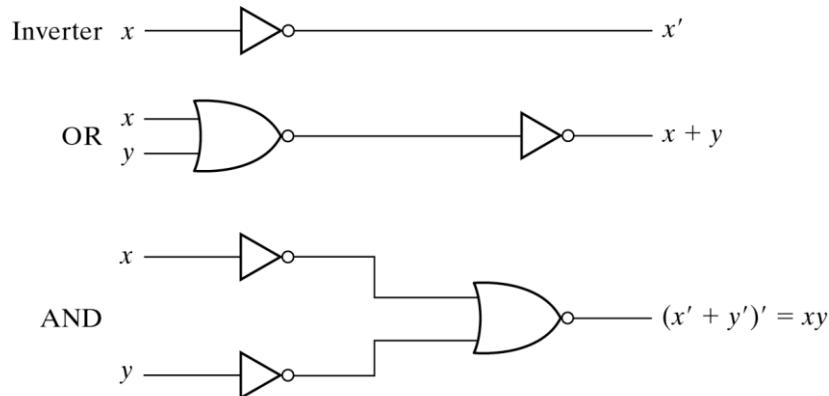
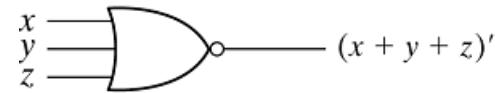
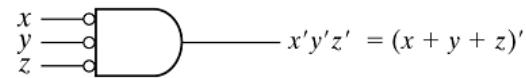


Fig. 3-24 Logic Operations with NOR Gates



(a) OR-invert



(a) Invert-AND

Fig. 3-25 Two Graphic Symbols for NOR Gate

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

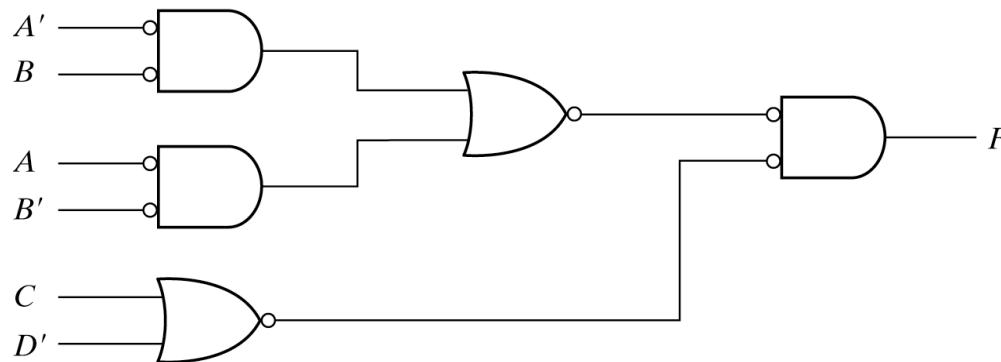
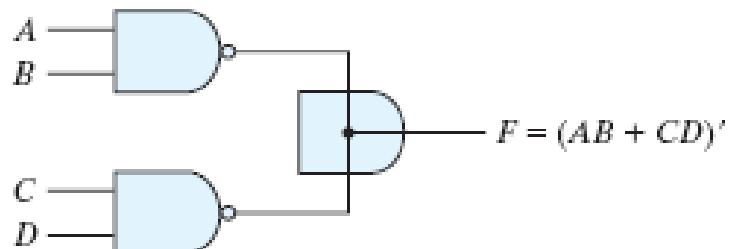


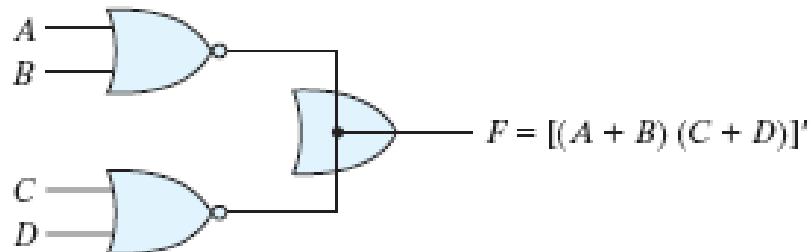
Fig. 3-27 Implementing $F = (AB' + A'B)(C + D')$ with NOR Gates

3.8 Other Two-Level Implementation

- (a) $F = (AB)' \cdot (CD)' = (AB + CD)'$
- (b) $F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$



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

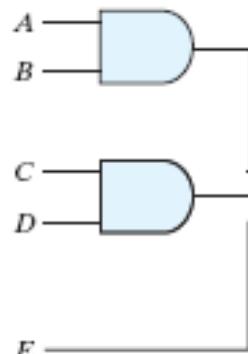


(b) Wired-OR in ECL gates
(OR-AND-INVERT)

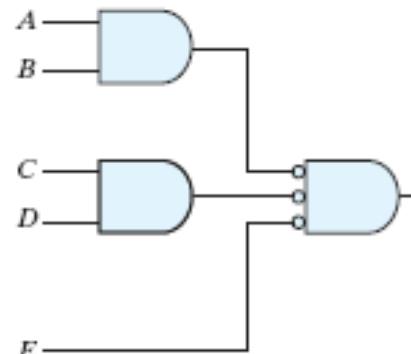
3.8 Other Two-Level Implementation

AND-OR-Invert Circuits

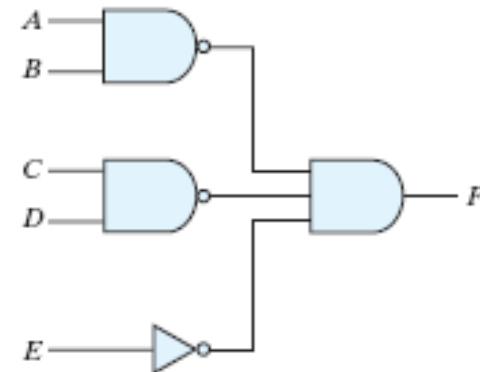
$$F = (AB + CD + E)'$$



(a) AND-NOR



(b) AND-NOR

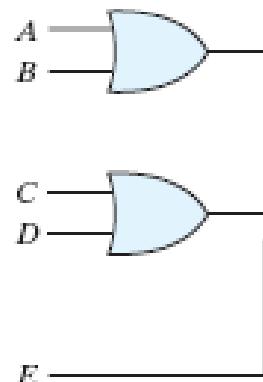


(c) NAND-AND

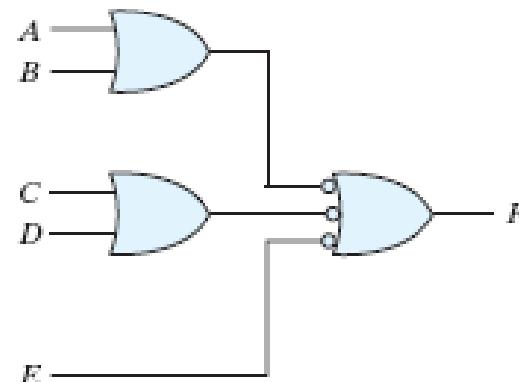
3.8 Other Two-Level Implementation

OR-AND-Invert Circuits

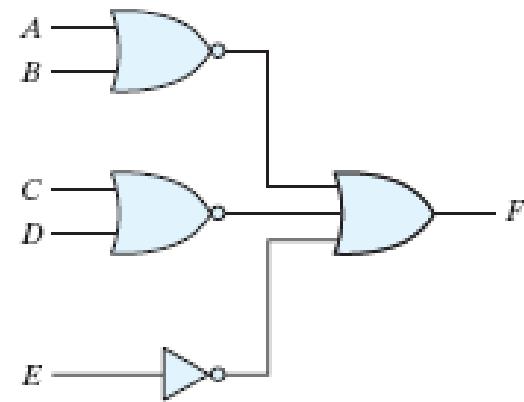
$$F = [(A + B)(C + D)E]'$$



(a) OR-NAND



(b) OR-NAND



(c) NOR-OR

3.8 Other Two-Level Implementation

Table 3.3
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.

3.9 Exclusive-OR Function

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

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

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

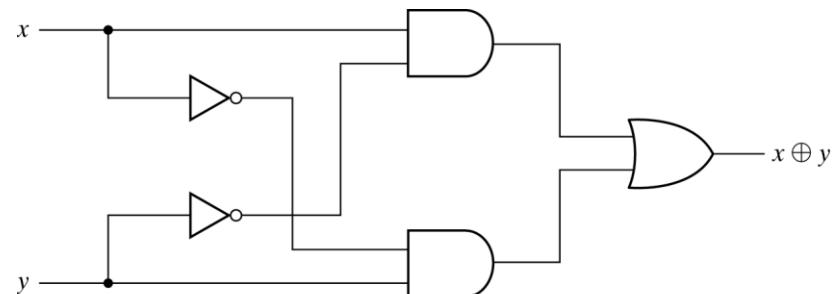
$$x \oplus x = 0$$

$$x \oplus x' = 1$$

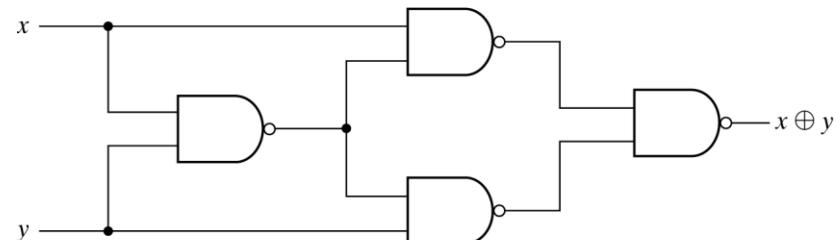
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

- $A \oplus B = B \oplus A$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$



(a) With AND-OR-NOT gates



(b) With NAND gates

Fig. 3-32 Exclusive-OR Implementations

3.9 Exclusive-OR Function

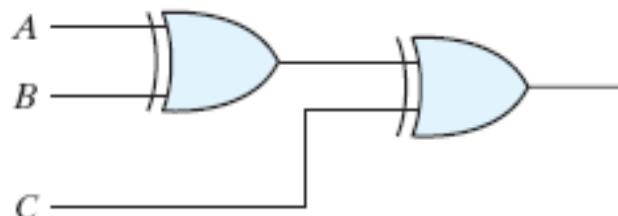
Odd / Even Function

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

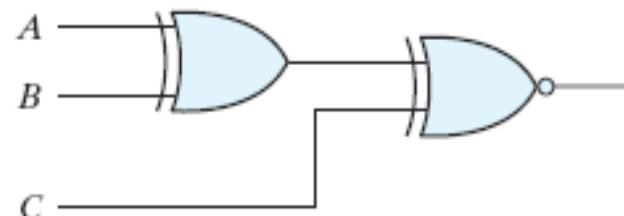
(a) Odd function $F = A \oplus B \oplus C$

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

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



(a) 3-input odd function



(b) 3-input even function

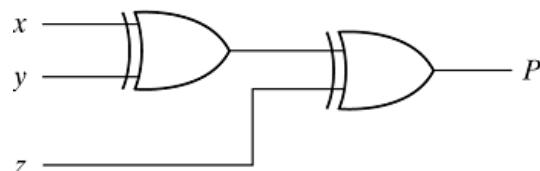
3.9 Exclusive-OR Function - Parity Generation and Checking

Parity Generation and Checking

Table 3-4
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

$$P = x \oplus y \oplus z$$

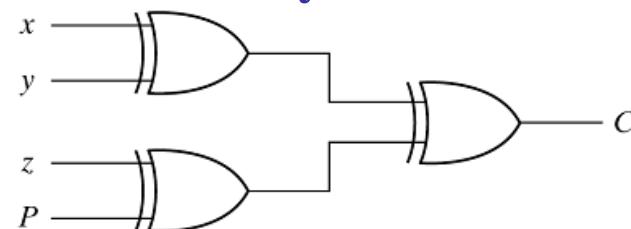


(a) 3-bit even parity generator

Table 3-5
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$



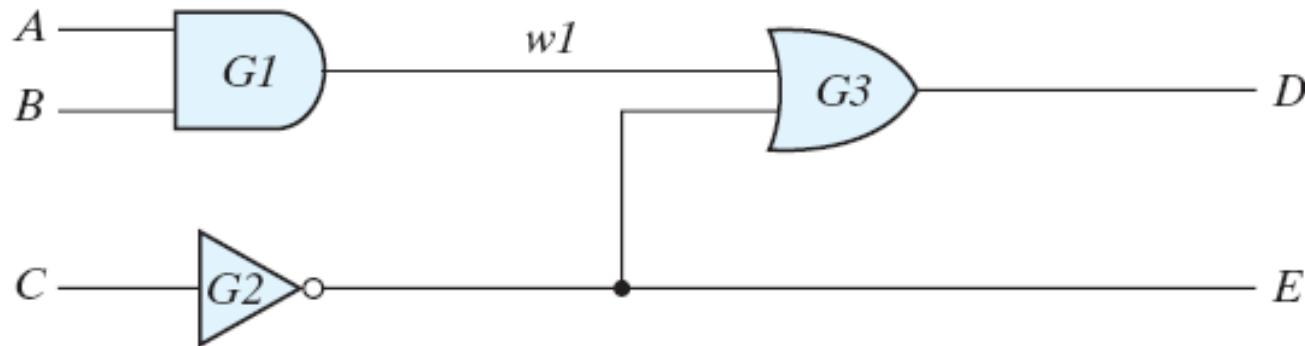
(a) 4-bit even parity checker

Fig. 3-36 Logic Diagram of a Parity Generator and Checker

3.10 HDL(Hardware Description Language)

Example 3-1

```
// Verilog model: Simple_Circuit
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
```



3.10 HDL – Gate delays

- Gate Delays - `timescale 1ns/100ps

```
//HDL Example 3-2
//Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input A,B,C;
    output x,y;
    wire e;
    and #(30) g1(e,A,B);
    or  #(20) g3(x,e,y);
    not #(10) g2(y,C);
endmodule
```

```
//HDL Example 3-3
//Stimulus for simple circuit
module stimcrct;
    reg A,B,C;
    wire x,y;
    circuit_with_delay cwd(A,B,C,x,y);
    initial
        begin
            A = 1'b0; B = 1'b0; C = 1'b0;
            #100
            A = 1'b1; B = 1'b1; C = 1'b1;
            #100 $finish;
        end
    endmodule
```

3.10 HDL

• Boolean Expressions

- AND, OR, NOT => ($\&$), ($|$), (\sim)

```
assign x = (A & B) | ~C);
```

```
//HDL Example 3-4
//Circuit specified with Boolean
equations
module circuit_bln (x,y,A,B,C,D);
    input A,B,C,D;
    output x,y;
    assign x = A | (B & C) | (~B & C);
    assign y = (~B & C) | (B & ~C &
~D);
endmodule
```

3.10 HDL – User-Defined Primitives (UDP)

User – Defined Primitives(UDP)

```
//HDL Example 3-5
//User defined primitive(UDP)
primitive crctp (x,A,B,C);
    output x;
    input A,B,C;
//Truth table for x(A,B,C) = Minterms (0,2,4,6,7)
    table
//    A  B  C : x (Note that this is only a comment)
        0  0  0 : 1;
        0  0  1 : 0;
        0  1  0 : 1;
        0  1  1 : 0;
        1  0  0 : 1;
        1  0  1 : 0;
        1  1  0 : 1;
        1  1  1 : 1;
    endtable
endprimitive
```

```
//Instantiate primitive
module declare_crctp;
    reg x,y,z;
    wire w;
    crctp (w,z,y,z);
endmodule
```