Chapter 3

Gate-Level Minimization

J.J. Shann

Contents

3-1 Introduction

補充資料： Cost Criteria

3-2 The Map Method

3-3 Four-Variable Map

補充資料： Five-Variable Map

Sum-of-Products (SoP) Simplification

3-4 **Product-of-Sums (PoS)** Simplification

3-5 Don’t-Care Conditions

SoP/PoS

補充資料：Quine-McCluskey Method (Tabular Method)

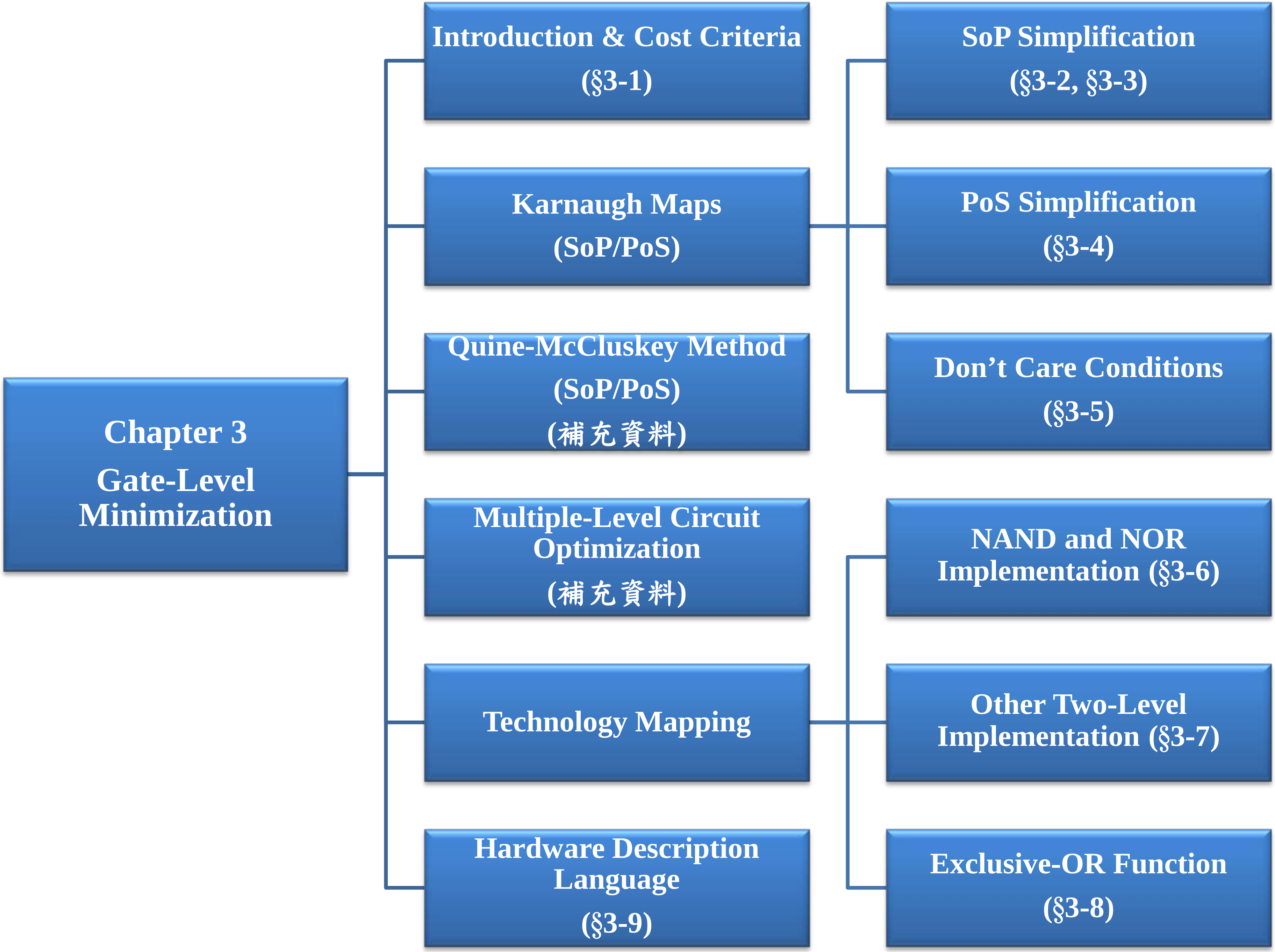
補充資料： Multiple-Level Circuit Optimization

3-6 NAND and NOR Implementation

3-7 Other Two-Level Implementations

3-8 Exclusive-OR Function

3-9 Hardware Description Language (HDL)



Chapter 3

Gate-Level Minimization

Introduction & Cost Criteria (§3-1)

SoP Simplification (§3-2, §3-3)

Karnaugh Maps (SoP/PoS)

PoS Simplification (§3-4)

Quine-McCluskey Method (SoP/PoS) (補充資料)

Don’t Care Conditions (§3-5)

Multiple-Level Circuit Optimization (補充資料)

NAND and NOR Implementation (§3-6)

Technology Mapping

Other Two-Level Implementation (§3-7)

Hardware Description Language (§3-9)

Exclusive-OR Function (§3-8)

3-1 Introduction

- Representation of a Boolean function:
  - Truth table: unique
  - Algebraic expression: many different forms

    ⇒ digital logic circuit
- Minimization of Boolean function:
  - Algebraic manipulation: literal minimization (§2-5)
    - use the rules and laws of ***Boolean algebra***
    - Disadv.: It lacks specific rules to predict each succeeding step in the manipulation process.
  - Map method: gate-level minimization (Ch3)
    - simple straightforward procedure ⇒ Manual design of simple ckts
    - Disadv.: Maps for more than 4 variables are not simple to use.
  - Tabular method: Quine-McCluskey method (補充資料)
    - systematic procedure ⇒ Computer-based logic synthesis tools

補充資料:

Cost Criteria

Cost Criteria

- Two cost criteria:
  1. Literal cost
     - the # of literal appearances in a Boolean expression
  2. Gate input cost (✓)
     - the # of inputs to the gates in the implementation

- Reference:
  - M. Morris Mano & Charles R. Kime, *Logic and Computer Design Fundamentals*, 3rd Edition, 2004, Pearson Prentice Hall. (§2-4)

Literal Cost

- Literal cost:
  - the # of literal appearances in a Boolean expression
  - E.g.: $F = AB + C(D + E)$ → 5 literals

    $F = AB + CD + CE$ → 6 literals
  - Adv.: is very simple to evaluate by counting literal appearances
  - Disadv.: does not represent ckt complexity accurately in all cases
    - E.g.:

$$G = ABCD + \bar{A}\bar{B}\bar{C}\bar{D} \quad \rightarrow 8 \text{ literals}$$

$$G = (\bar{A} + B)(\bar{B} + C)(\bar{C} + D)(\bar{D} + A) \quad \rightarrow 8 \text{ literals}$$

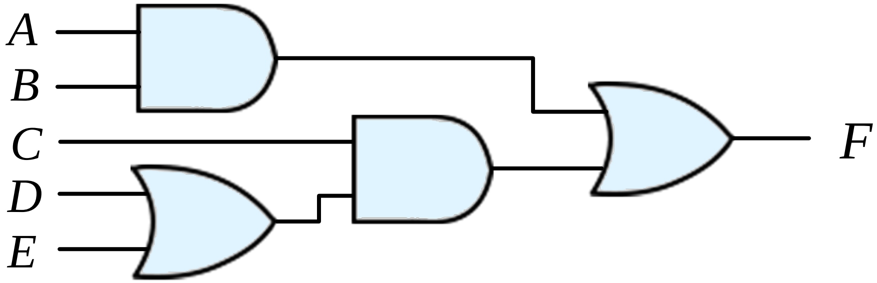
Gate Input Cost

- Gate input cost (GIC):
  - the # of inputs to the gates in the implementation
  - is a good measure for contemporary logic implementation
    - is proportional to the # of transistors and wires used in implementing a logic ckt. (especially for ckt ≥ 2 levels)

- E.g.: nonstandard vs. standard form (p.2-60)

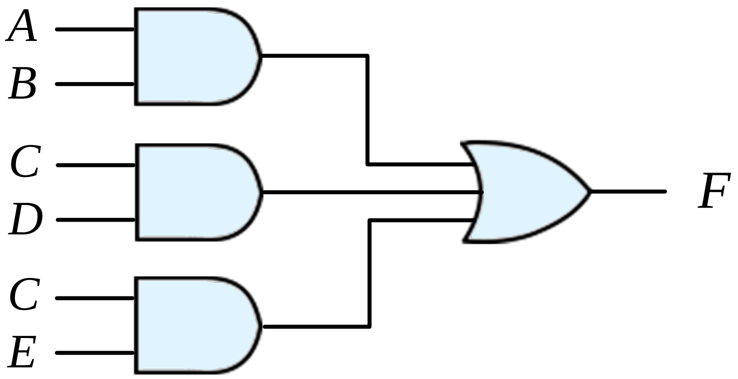
$$F = AB + C\,(D + E) \rightarrow \text{3-level}$$

$$= AB + CD + DE \rightarrow \text{2-level}$$



$AB + C(D + E)$

GIC = 8



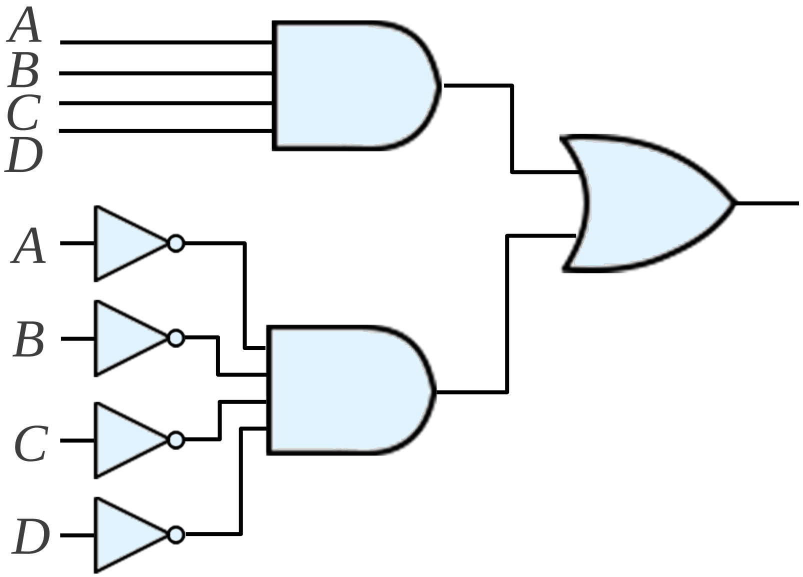
$AB + CD + CE$

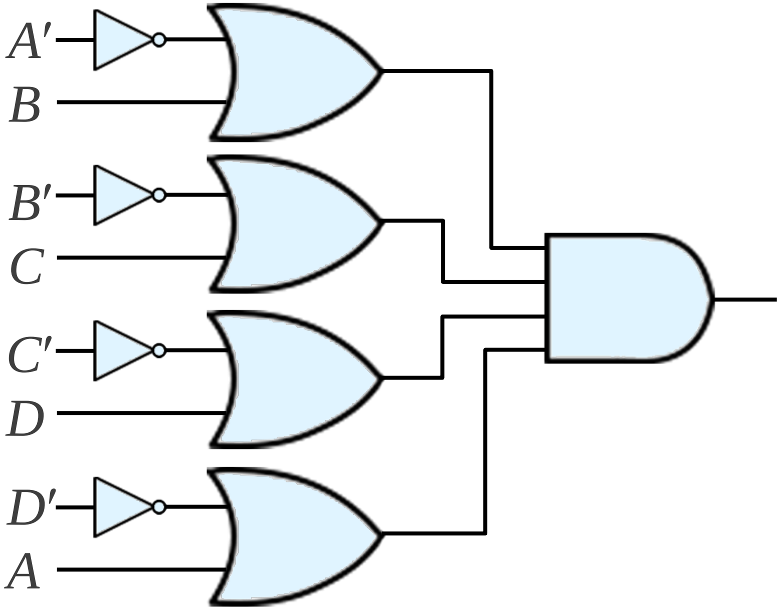
GIC = 9

- E.g.:

$$G = ABCD + \bar{A}\bar{B}\bar{C}\bar{D}$$ → GIC = ?

$$G = (\bar{A} + B)(\bar{B} + C)(\bar{C} + D)(\bar{D} + A)$$ → GIC = ?





GIC = 4 + 2× 4 + 2 = 14

GIC = 4 + 4 × 2 + 4 = 16

- For SoP or PoS eqs, GIC = the sum of
  - all literal appearances
  - the # of terms excluding terms that consist only of a single literal
  - the # of distinct complemented single literals (optional)
  - E.g.: p.3-10

$$G = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \qquad \rightarrow \mathrm{GIC} = 8 + 2\,(+\,4)$$

$$G = (\overline{A} + B)(\overline{B} + C)(\overline{C} + D)(\overline{D} + A) \quad \rightarrow \mathrm{GIC} = 8 + 4\,(+\,4)$$

3-2 The Map Method

$XY + XY' = X$

- Map method: Karnaugh map simplification
  - a simple straightforward procedure
  - K-map: a pictorial form of a truth table
    - a diagram made up of squares **_n_ input variables → 2^n squares**
    - Each square represents one minterm of the function.
    - Any adjacent squares in the map differ by only one variable.
  - The simplified expressions produced by the map are always in one of the two ***standard forms***:
    - SOP (sum of products) or POS (product of sums)
- The simplest algebraic expression: not unique
  - one w/ a minimum # *of terms* and

    w/ the fewest possible # *of literals* per term.

  ⇒ a ckt diagram w/ a minimum # *of gates* and

  the minimum # *of inputs* to the gate.

A. Two-Variable Map

- Two-variable map: 2 variables → 4 minterms
  - 4 squares, one for each minterm.
  - A function of 2 variables can be represented in the map by marking the squares that correspond to the minterms of the function.

***F*(*X*,*Y*)**

X \ Y	0	1
0	0 X′Y′	1 X′Y
1	2 XY′	3 XY

Y \ X	0	1
0	0 X′Y′	2 XY′
1	1 X′Y	3 XY

* Any adjacent squares in the map differ by only one variable.

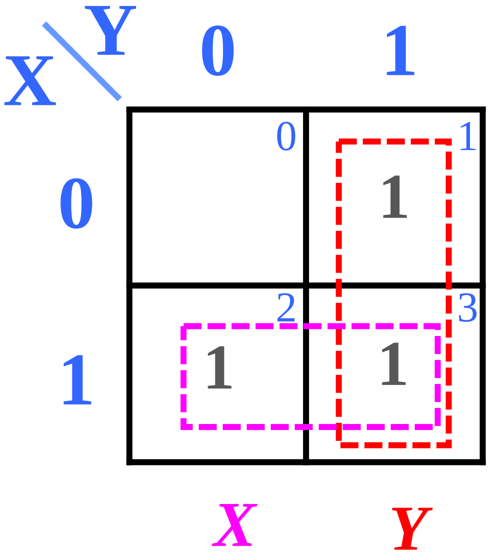
⇒ 2 adjacent minterms may be combined into a term with 1 literal removed.

$(xy + xy' = x)$

Example: 2-Variable K-Map

X\Y	0	1
0	0 X′Y′	1 X′Y
1	2 XY′	3 XY

- $F(X,Y) = m_1 + m_2 + m_3$



$\Rightarrow F = X + Y$

$$\begin{aligned}
F_2 &= m_1 + m_2 + m_3 \\
&= \overline{X}Y + X\overline{Y} + XY \\
&= \overline{X}Y + X(\overline{Y} + Y) \\
&= \overline{X}Y + X \\
&= (\overline{X} + X)(Y + X) \\
&= X + Y
\end{aligned}$$

by combining squares in the map

by applying Boolean algebra

B. Three-Variable Map

- Three-variable map: 8 minterms ⇒ 8 squares

F(X,Y,Z)

X \ YZ	**00**	**01**	**11**	**10**
0	0 $X'Y'Z'$	1 $X'Y'Z$	3 $X'YZ$	2 $X'YZ'$
1	4 $XY'Z'$	5 $XY'Z$	7 XYZ	6 XYZ'

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

- Only one bit changes in value from one adjacent column to the next
  - Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other.
  - E.g.: m_5 & m_7
- Note: Each square has 3 adjacent squares.
  - The right & left edges touch each other to form adjacent squares.
  - E.g.: $m_4 \rightarrow m_0, m_5, m_6$

- Alternatives of 3-variable map:

F(*X*,*Y*,*Z*)

X \ YZ	00	01	11	10
0	0 X′Y′Z′	1 X′Y′Z	3 X′YZ	2 X′YZ′
1	4 XY′Z′	5 XY′Z	7 XYZ	6 XYZ′

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

Z \ XY	00	01	11	10
0	0	2	6	4
1	1	3	7	5

m_0	m_2	m_6	m_4
m_1	m_3	m_7	m_5

- Alternatives of 3-variable map: (cont’d)

$F(X,Y,Z)$

XY \ Z	0	1
00	0	1
01	2	3
11	6	7
10	4	5

YZ \ X	0	1
00	0	4
01	1	5
11	3	7
10	2	6

Map Minimization of SOP Expression

- Basic property of adjacent squares:

X \ YZ	00	01	11	10
0	0 X′Y′Z′	1 X′Y′Z	3 X′YZ	2 X′YZ′
1	4 XY′Z′	5 XY′Z	7 XYZ	6 XYZ′

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

– Any two adjacent squares in the map differ by only one variable: primed in one square and unprimed in the other

  - E.g.: $m_5 = X\overline{Y}Z, \quad m_7 = XYZ$

⇒ Any two minterms in adjacent squares that are ORed together can be simplified to a single AND term w/ a removal of the different variable.

  - E.g.: $m_5 + m_7 = X\overline{Y}Z + XYZ = XZ(\overline{Y} + Y) = XZ$

- Procedure of map minimization of SOP expression:

  i. A 1 is marked in each minterm that represents the function.

  - Two ways:

    (1) Convert each minterm to a binary number and then mark a 1 in the corresponding square.

    (2) Obtain the coincidence of the variables in each term.

  ii. Find possible adjacent 2^k squares:

  - 2 adjacent squares (i.e., minterms) → remove 1 literal
  - 4 adjacent squares (i.e., minterms) → remove 2 literal
  - 2^k adjacent squares (i.e., minterms) → remove k literal

  ⇒ The larger the # of squares combined, the less the # of literals in the product (AND) term.

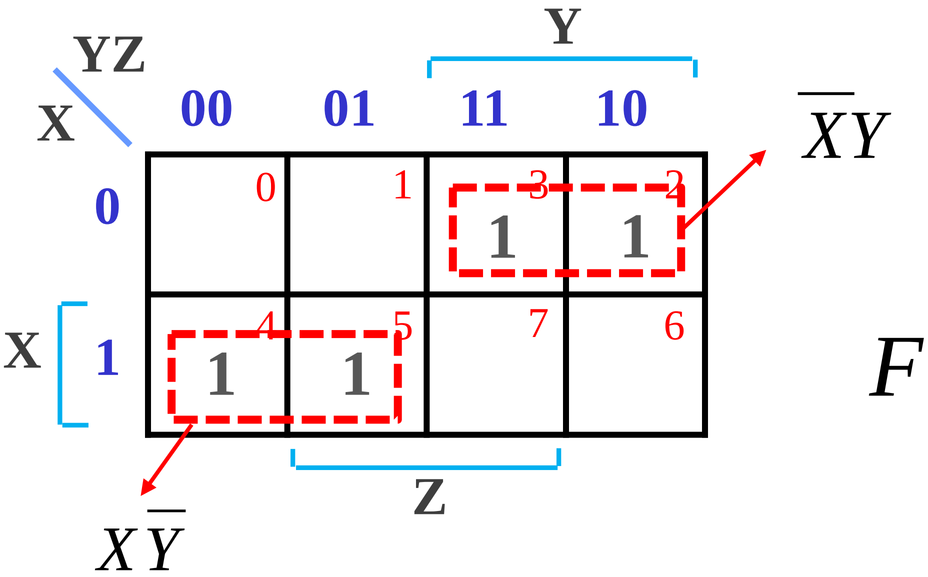
* It is possible to use the same square more than once.

Example 3.1

- Simplify the Boolean function

$$F(X, Y, Z) = \Sigma m(2,3,4,5)$$

<Ans.>

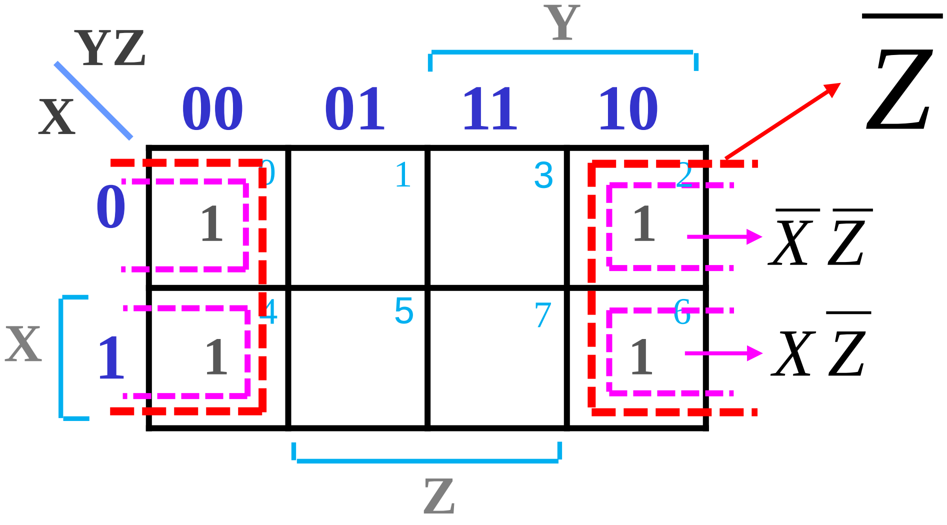


$$F = \overline{X}Y + X\overline{Y}$$

Example: 4-minterm Product Terms

- Product terms using 4 minterms
- F(*X*, *Y*, *Z*) = Σ*m*(0,2,4,6)

* The right & left edges touch each other to form adjacent squares.



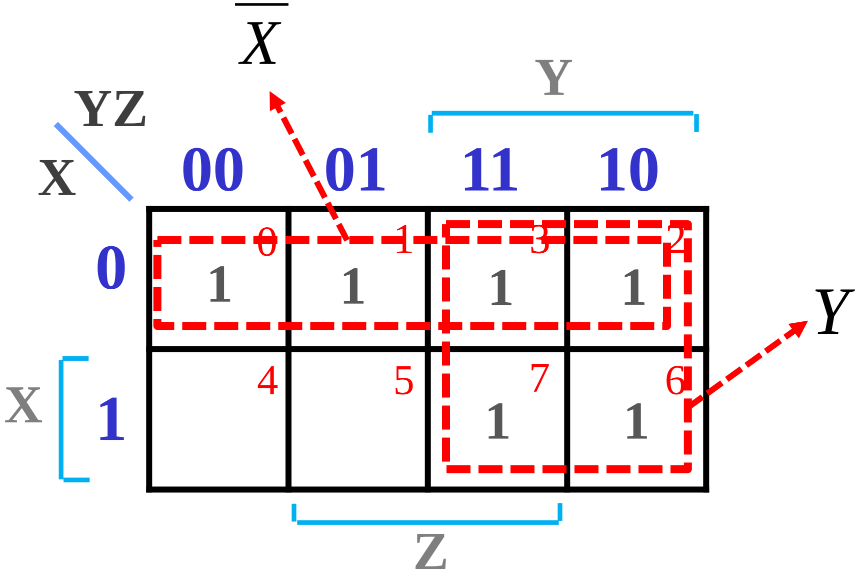
$$\underline{m_0+m_2}+\underline{m_4+m_6} = \underline{\bar{X}\bar{Y}\bar{Z}+\bar{X}Y\bar{Z}}+\underline{X\bar{Y}\bar{Z}+XY\bar{Z}}$$

$$=\underline{\bar{X}\bar{Z}(\bar{Y}+Y)}+\underline{X\bar{Z}(\bar{Y}+Y)}$$

$$=\underline{\underline{\bar{X}\bar{Z}}+\underline{X\bar{Z}}}=\underline{\bar{Z}(\bar{X}+X)}=\underline{\bar{Z}}$$

Example: 4-minterm Product Terms

- F(*X*, *Y*, *Z*) = Σ*m*(0,1,2,3,6,7)



X \ YZ	00	01	11	10
0	0 1	1 1	3 1	2 1
1	4	5	7 1	6 1

$$F = \overline{X} + Y$$

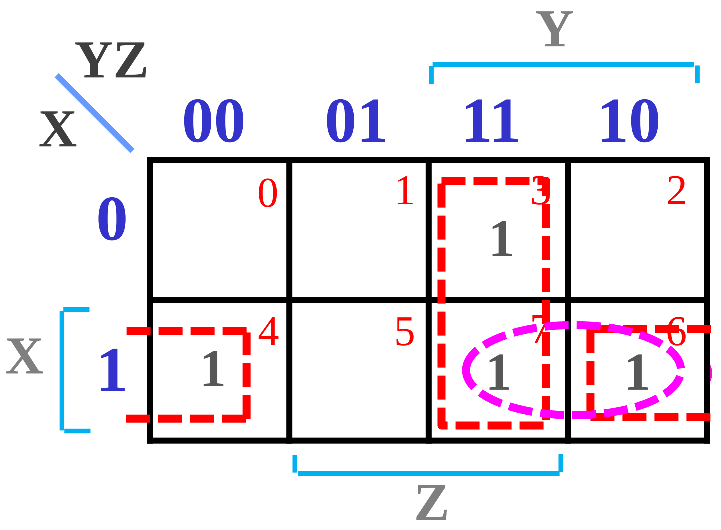
* It is possible to use the same square more than once.

Example 3.2: Redundant Terms

- Simplify the Boolean function

$$F(X,Y,Z)=\sum m(3,4,6,7)$$

<Ans.>



X \ YZ	00	01	11	10
0			1	
1	1		1	1

XY → redundant (×)

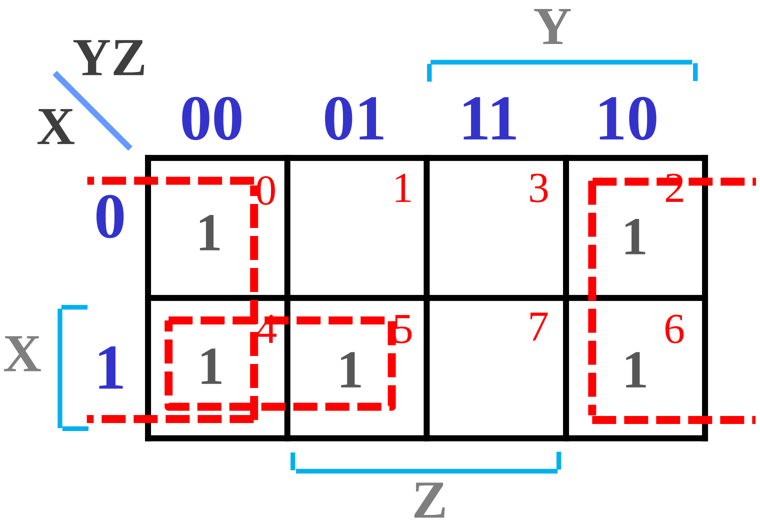
$$F(X,Y,Z)=YZ+X\overline{Z}$$

Example 3.3

- Simplify the Boolean function

$$F(X,Y,Z)=\sum m(0,2,4,5,6)$$

<Ans.>



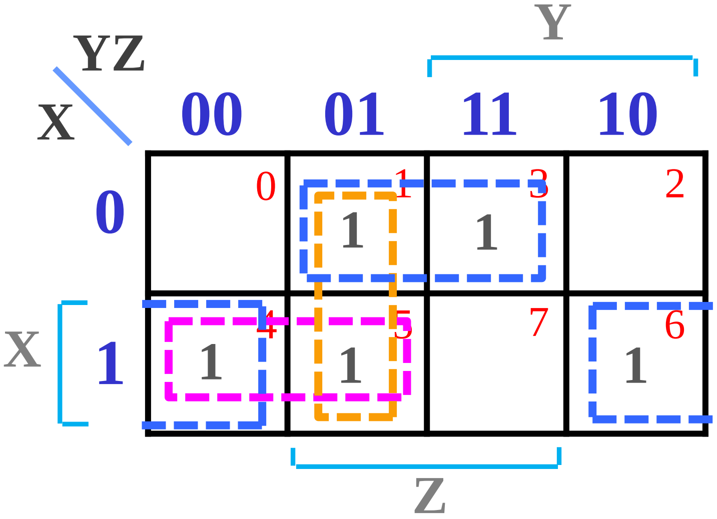
X \ YZ	00	01	11	10
0	0 1	1	3	2 1
1	4 1	5 1	7	6 1

$$F(X,Y,Z)=\overline{Z}+X\overline{Y}$$

Non-unique Optimized Expressions

- There may be alternative ways of combining squares to product equally optimized expressions:

- E.g.: $F(X,Y,Z)=\sum m(1,3,4,5,6)$



X \ YZ	00	01	11	10
0		1	1	
1	1	1		1

$$F(X,Y,Z)=\sum m(1,3,4,5,6)$$
$$=\overline{X}Z+X\overline{Z}+X\overline{Y}$$
$$=\overline{X}Z+X\overline{Z}+\overline{Y}Z$$

Simplifying Functions not Expressed as Sum-of-minterms Form

- If a function is not expressed as a sum of minterms:
  - use the map to obtain the minterms of the function & then simplify the function
- Example 3.4: Given the Boolean function

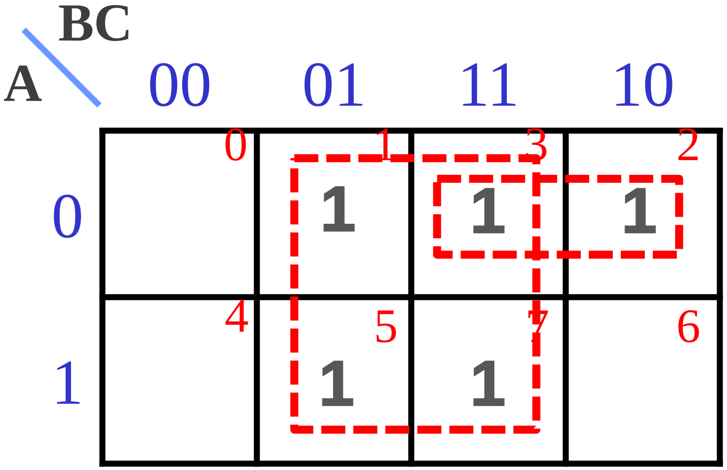
$$F = A'C + A'B + AB'C + BC$$

<Ans.>

$$\begin{array}{lcccc} F = & A'C & + A'B & + AB'C & + BC \\ & 0\text{–}1 & 01\text{–} & 101 & \text{–}11 \\ & 1, 3 & 2, 3 & 5 & 3, 7 \end{array}$$

$$= \Sigma m(1, 2, 3, 5, 7)$$

$$= C + A'B$$



A \ BC	00	01	11	10
0	0	1 1	3 1	2 1
1	4	5 1	7 1	6

3-3 Four-Variable Map

- Four-variable map: 16 minterms ⇒ 16 squares

F(*W*, *X*, *Y*, *Z*)

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

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}

– Note: Each square has 4 adjacent squares.

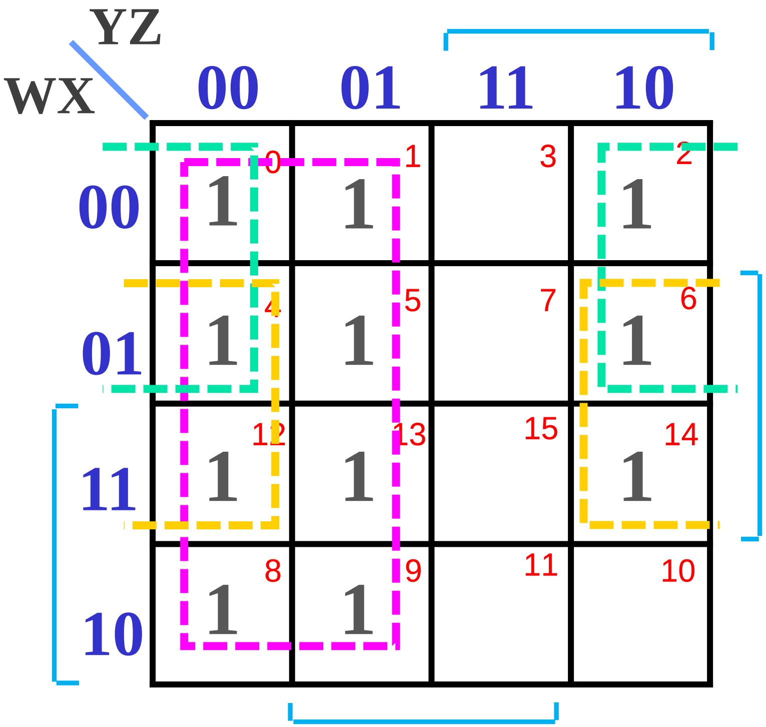
  - The map is considered to lie on a surface w/ the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.
  - E.g.: $m_8 \rightarrow m_0, m_9, m_{10}, m_{12}$

Example 3.5

- Simplify the Boolean function

$$F(W, X, Y, Z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$$

<Ans.>



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

$$F = \overline{Y} + \overline{W}\,\overline{Z} + X\overline{Z}$$

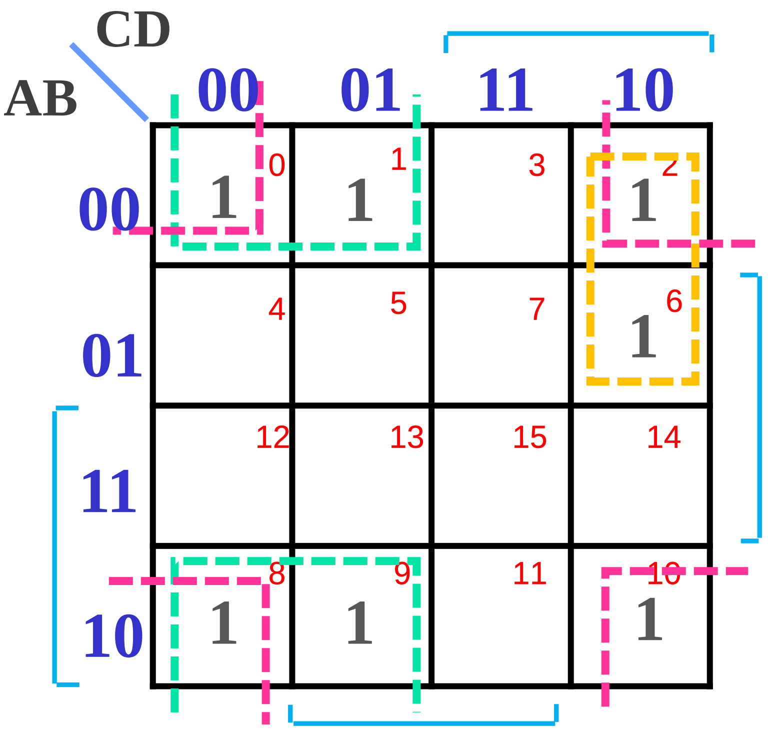
Example 3.6

- Simplify the Boolean function

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

000- -010 0110 100-

<Ans.>



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

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

Map Manipulation

- When choosing adjacent squares in a map:
  - Ensure that all the minterms of the function are covered when combining the squares.
  - Minimize the # of terms in the expression.
    - avoid any redundant terms whose minterms are already covered by other terms

Prime Implicants

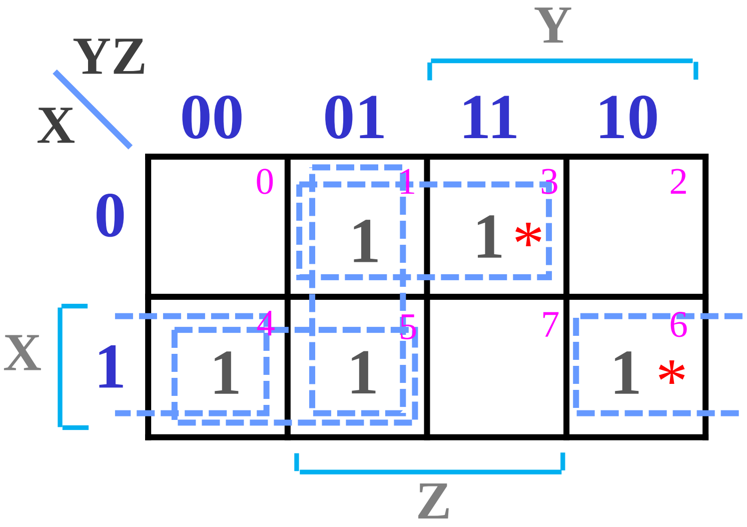
- *Implicant:*
  - A product term is an implicant of a function if the function has the value 1 for all minterms of the product term.
- *Prime implicant*: PI
  - a product term obtained by combining the max. possible # of adjacent squares in the map
- *Essential prime implicant*: EPI, must be included
  - If a minterm in a square is covered by only one PI, that PI is said to be essential.
    - Look at each square marked w/ a 1 and check the # of PIs that cover it.

Example

- Find the PIs and EPIs of the Boolean function

$$F(X, Y, Z) = \Sigma m(1,3,4,5,6)$$

<Ans.>



X \ YZ	00	01	11	10
0		1	1 *	
1	1	1		1 *

$$4\ \text{PIs}:\quad \bar{X}Z,\ \ \bar{Y}Z,\ \ X\bar{Z},\ \ X\bar{Y}$$

$$2\ \text{EPIs}:\quad \bar{X}Z\ (\underset{*}{m_3}),\ \ X\bar{Z}\ (\underset{*}{m_6})$$

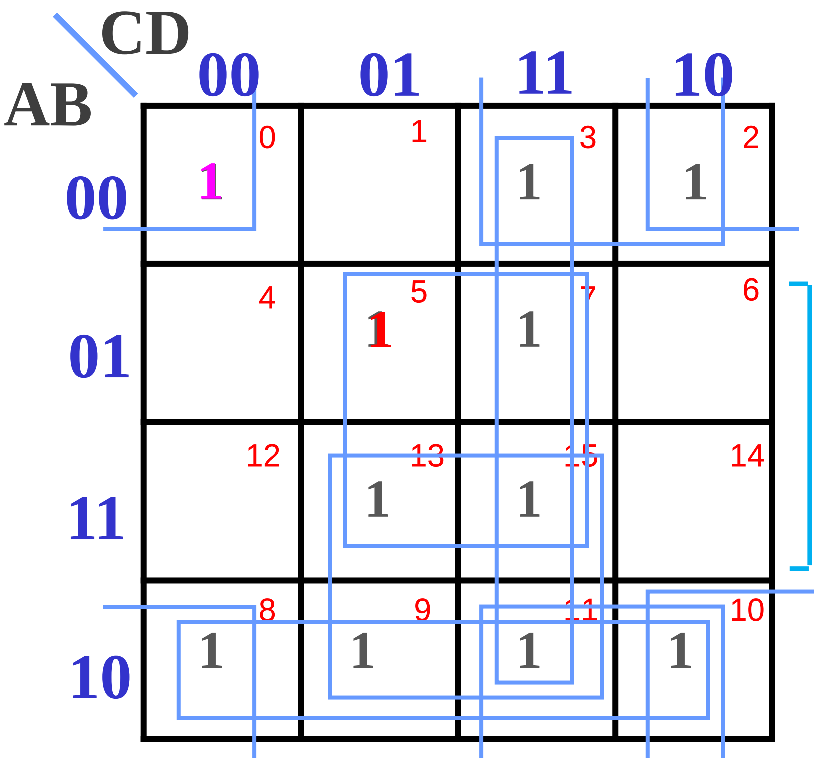
$$\begin{aligned} F(X,Y,Z) &= \sum m(1,3,4,5,6) \\ &= \bar{X}Z + X\bar{Z} + \boxed{X\bar{Y}} \\ &= \bar{X}Z + X\bar{Z} + \boxed{\bar{Y}Z} \end{aligned}$$

Example

- Find the PIs and EPIs of the Boolean function

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

<Ans.>

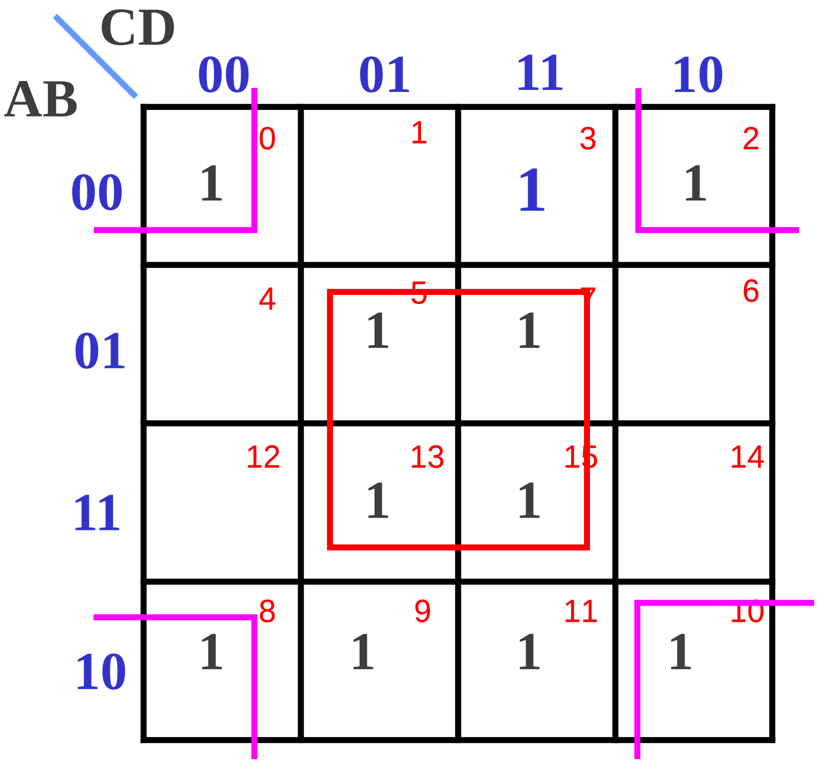


AB \ CD	00	01	11	10
00	0 1	1	3 1	2 1
01	4	5 1	7 1	6
11	12	13 1	15 1	14
10	8 1	9 1	11 1	10 1

PIs:	EPIs:
CD	B′D′ (m_0)
BD	BD (m_5)
AD	
AB′	
B′C	
B′D′	

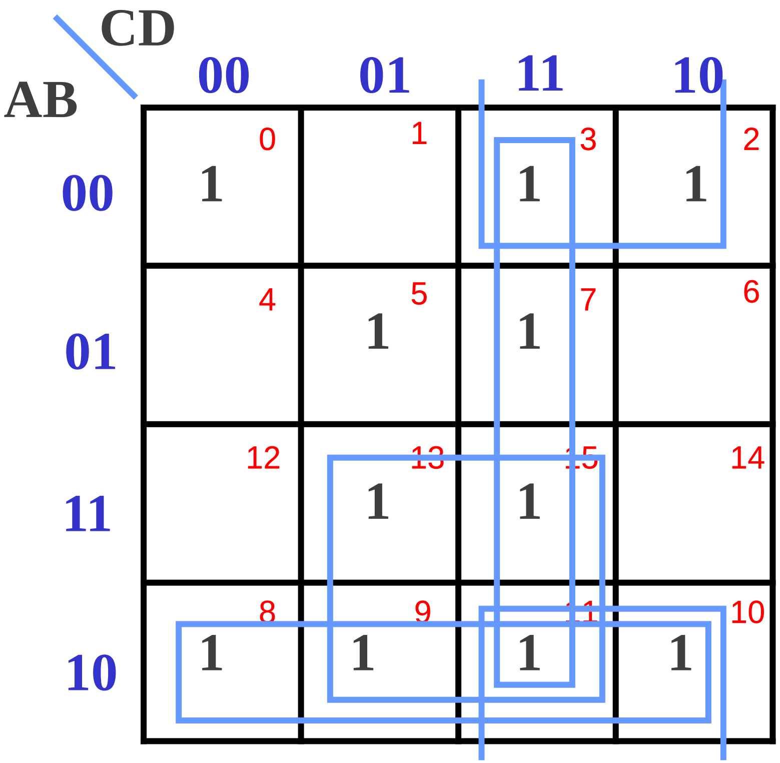
Example (Cont’)

6 PIs: BD, B'D', CD, B'C, AD, AB



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

2 EPIs: BD (m_5), B′D′ (m_0)



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

The remaining 4 PIs:

CD, B′C, AD, AB′

Finding the Simplified Expression

- Procedure for finding the simplified expression from the map: (SoP form)

  i. Determine all PIs.

  ii. The simplified expression is obtained from the logical sum of all the EPIs plus other PIs that may be needed to cover any remaining minterms not covered by the EPIs.

  – There may be more than one expression that satisfied the simplification criteria.

Example: p.3-33

- Simplify the Boolean function

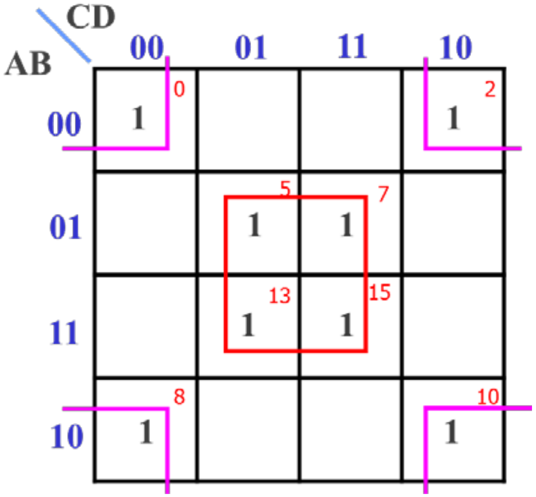
$$F(A,B,C,D) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$$

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

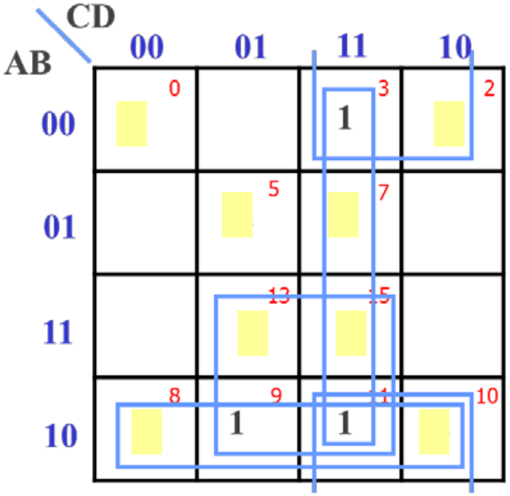
<Ans.>

6 PIs: BD, B'D', CD, B'C, AD, AB'

2 EPIs: BD, B'D'



2 EPIs: BD (m_5), B′D′ (m_0)



4 PIs: CD, B′C, AD, AB′

EPIs: BD, B'D' → $m_0, m_2, m_5, m_7, m_8, m_{10}, m_{13}, m_{15}$

⇒ Combine PIs that contains m_3, m_9, m_{11} (CD, B'C, AD, AB')

m_3, m_{11} m_9, m_{11}

⇒ F = BD + B'D' + CD + AD

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

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

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

補充資料： Five-Variable Map

- Five-variable map: 32 minterms ⇒ 32 squares

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

AB \ CDE	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01								
11								
10								

– Each square has 5 adjacent squares.

Alternatives: Five-variable map

A = 0

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

A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
10	28	29	31	30
11	24	25	27	26

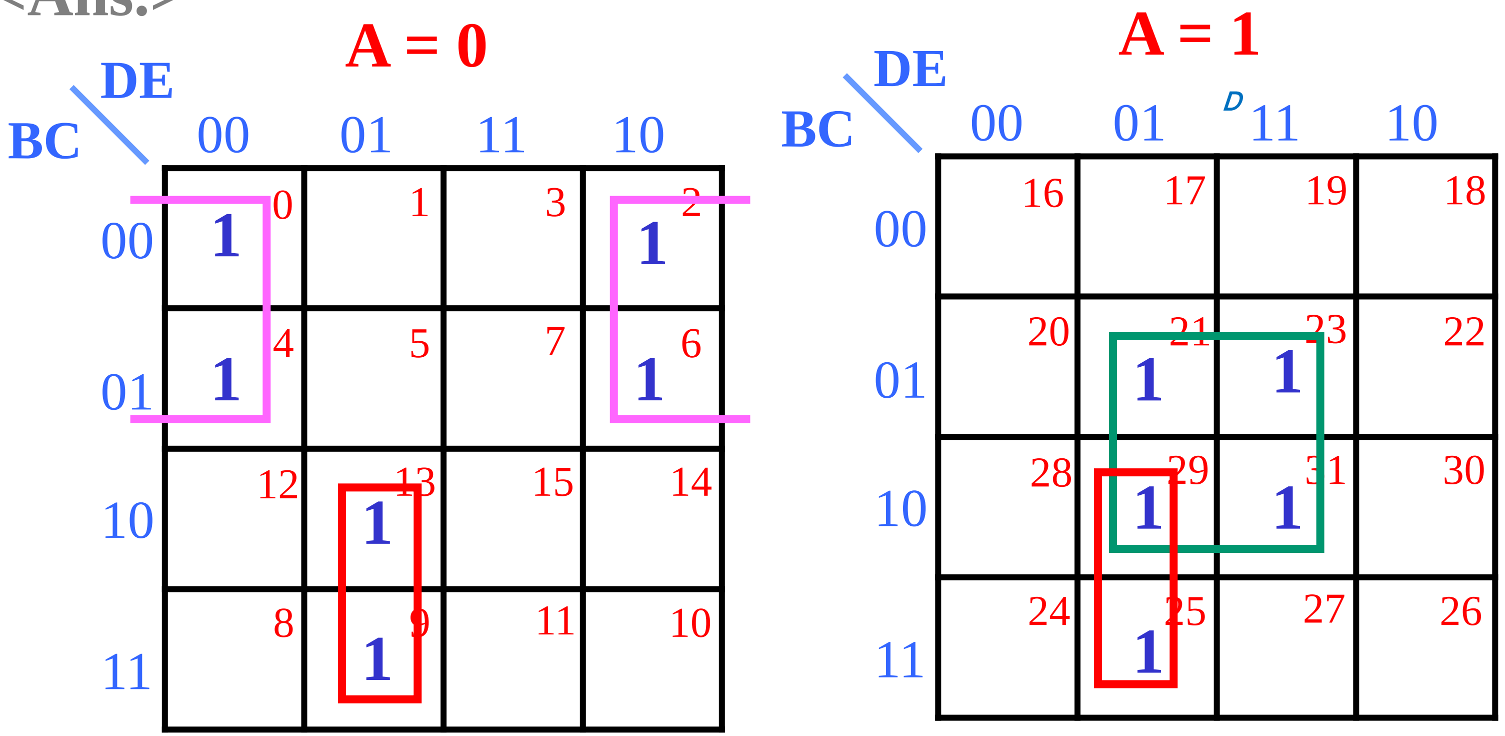
- Each square has 5 adjacent squares.
  - Consider the two half maps as being one on top of the other.
  - Any 2 squares that fall one over the other are considered adjacent.
  - E.g.: $m_8 \rightarrow m_0, m_9, m_{10}, m_{12}, m_{24}$

Example

- Simplify the Boolean function

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

<Ans.>



A = 0

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

A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21 1	23 1	22
10	28	29 1	31 1	30
11	24	25 1	27	26

F = A′B′E′ + ACE + BD′E

Summary

- Five-variable map: 32 minterms ⇒ 32 squares
- Six-variable map: 64 minterms ⇒ 64 squares
- Maps for more than 4 variables are not as simple to use:
  - Employ computer programs specifically written to facilitate the simplification of Boolean functions w/ a large # of variables.

⇒ 補充 Quine-McCluskey Method (p.3-51)

Reference:

- Randy H. Katz & Gaetano Borriello, *Contemporary Logic Design*, Prentice Hall.

3-4 Product of Sums Simplification

- Approach 1: POS of *F*
  - Simplified *F'* in the form of sum of products
  - Apply DeMorgan's theorem $F = (F')'$

    F': sum of products $\Rightarrow$ $F = (F')'$: product of sums

  - E.g.: Simplify the Boolean function in POS:

$$F = \Sigma m(0, 4, 6)$$

$$\Rightarrow F' = \Sigma m(1, 2, 3, 5, 7)$$
$$= C + A'B$$

$$\Rightarrow F = (C + A'B)'$$
$$= C' (A + B')$$

F′

A \ BC	00	01	11	10
0	0	1 1	3 1	2 1
1	4	5 1	7 1	6

Approach 2 (duality): POS of *F*

- combinations of maxterms (it was minterms for SoP)
  - i. A 0 is marked in each maxterm that represents the function.
  - ii. Find possible adjacent 2^k squares and realize each set as a sum (OR) term, w/ variables being complemented.
- E.g.: for 4 variables

AB \ CD	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}

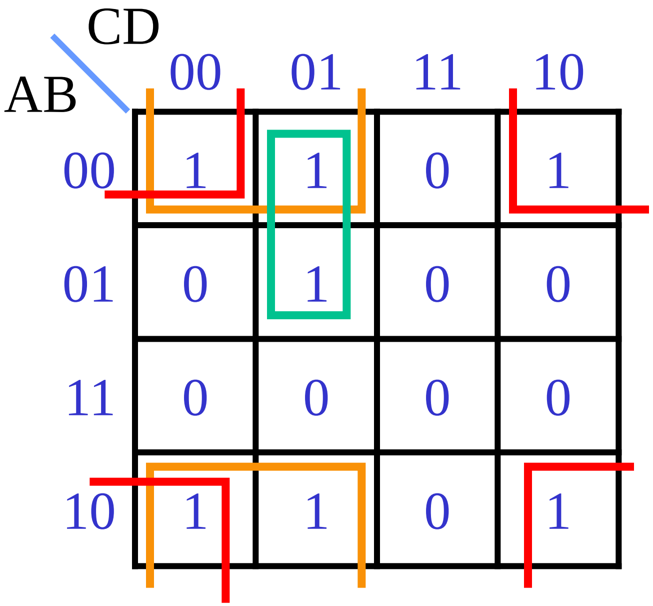
$$
\begin{aligned}
M_0M_1 &= (A+B+C+D)(A+B+C+D') \\
&= (A+B+C) + (DD') \\
&= A + B + C
\end{aligned}
$$

Example 3.7

- Simplify the Boolean function in (a) SOP and (b) POS: F(A,B,C,D) = Σ(0,1,2,5,8,9,10)

<Ans.>

(a) SOP of F(A,B,C,D) = Σ(0,1,2,5,8,9,10)



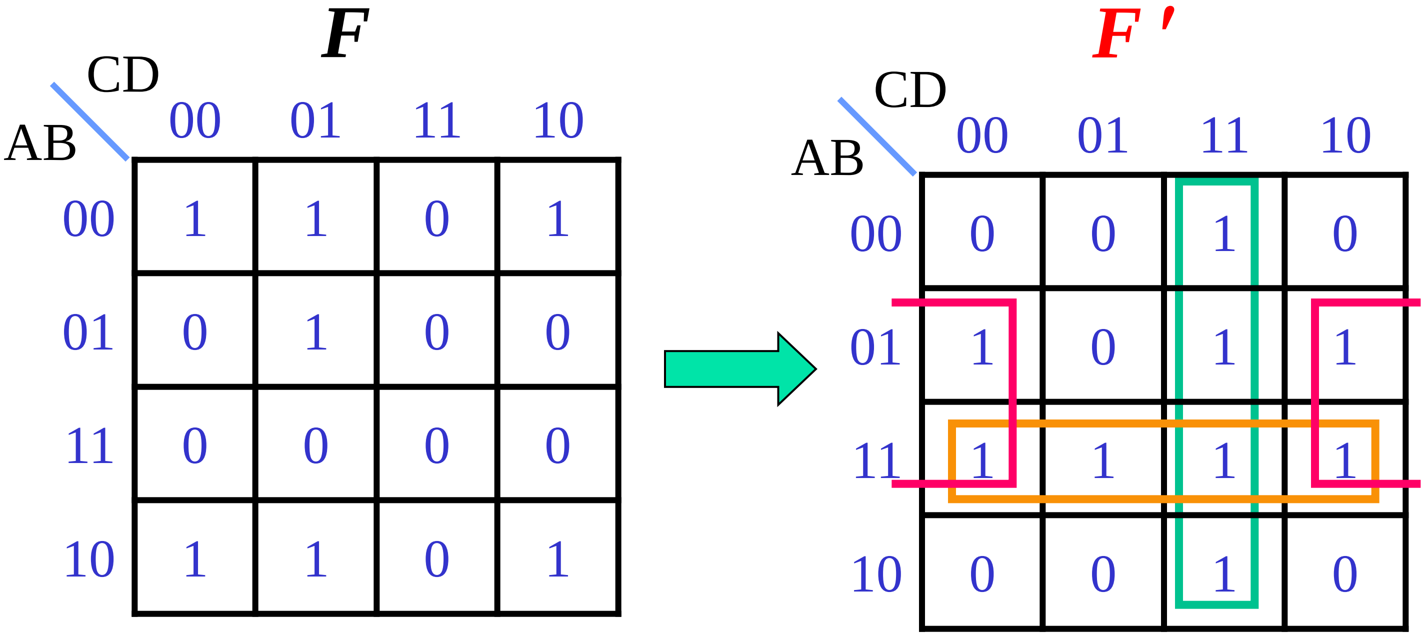
AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

F = B′D′ + B′C′ + A′C′D

<Ans.>

(b) POS of F(A,B,C,D) = Σ(0,1,2,5,8,9,10)

Approach 1:



F

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

F ′

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

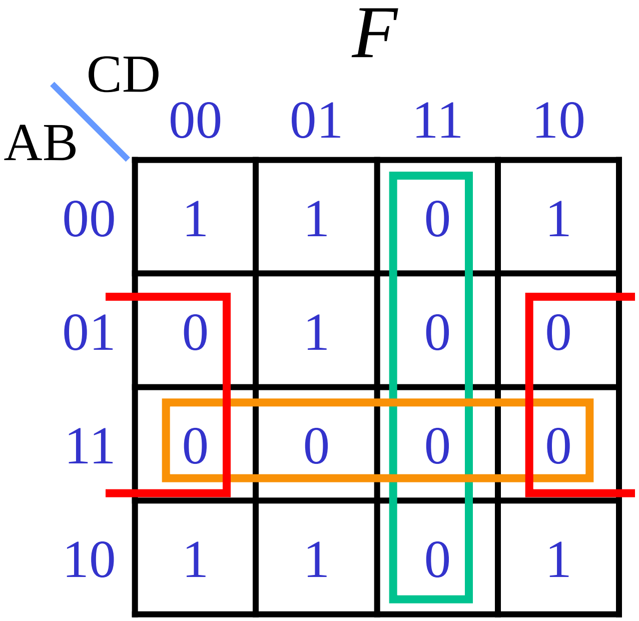
F′ = AB + CD + BD′

F = (F′)′ = (AB + CD + BD′)′ = (A′+B′)(C′+D′)(B′+D)

<Ans.>

(b) POS of F(A,B,C,D) = Σ(0,1,2,5,8,9,10)

Approach 2: think in terms of maxterms



F

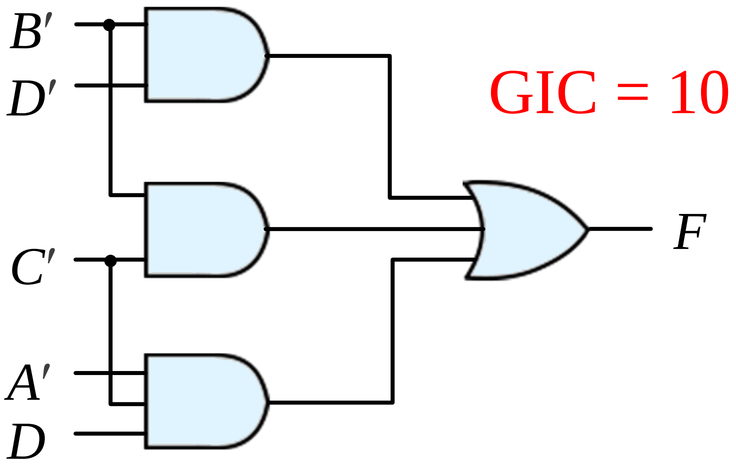
AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

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

- Gate implementation:

  Assumption: The input variables are directly available in their complement ⇒ Inverters are not needed.

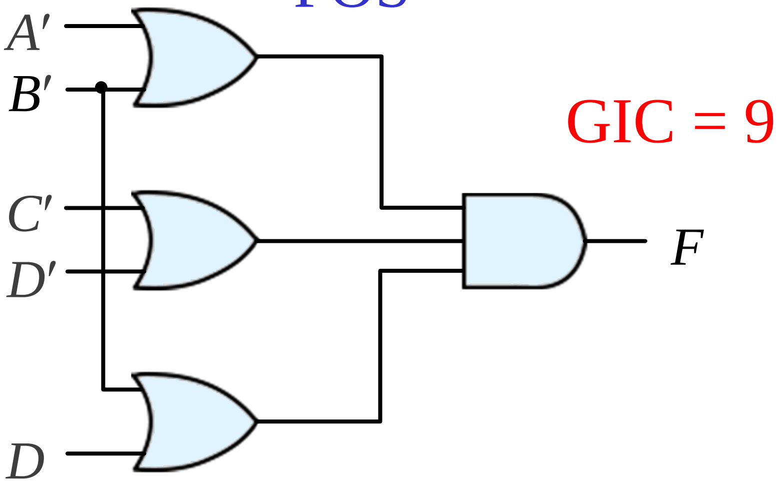
SOP



GIC = 10

F = B′D′ + B′C′ + A′C′D

POS



GIC = 9

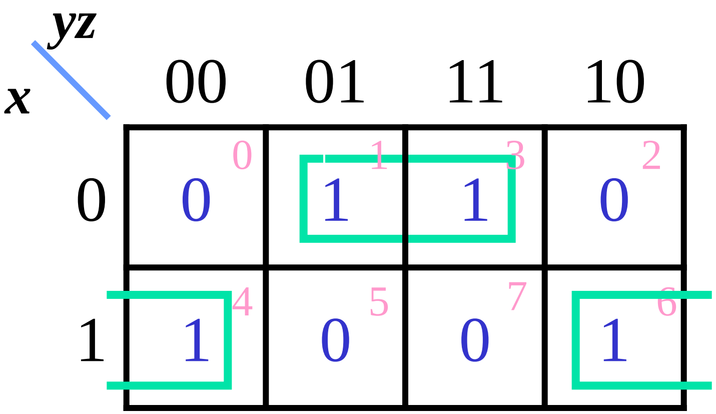
F = (A′ + B′) (C′ + D′) (B ′ + D)

- The implementation of a function in a standard form is said to be a two-level implementation.
- Determine which form will be best for a function.

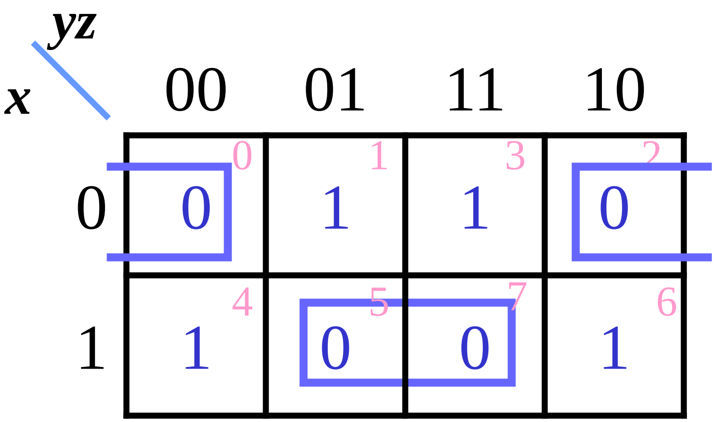
Example

- Given the truth table of a function, simplify the function in (a) SOP and (b) POS.

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



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



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

<Ans.>

F(x, y, z) = Σ(1, 3, 4, 6) = Π(0, 2, 5, 7)

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

(b) i. F′ = $x'z' + xz$

F = (F′)′ = $(x' + z')(x + z)$

ii. Or think in terms of maxterms

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

3-5 Don't-Care Conditions

- ***Don't care condition*:**
  - the unspecified minterms of a function
  - is represented by an ×
  - E.g.: A 4-bit decimal code has 6 combinations which are not used.
- ***Incompletely specified function*:**
  - has unspecified outputs for some input combinations

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

* 1010 ~ 1111 are not used and have no meaning.

Simplification of an Incompletely Specified Function

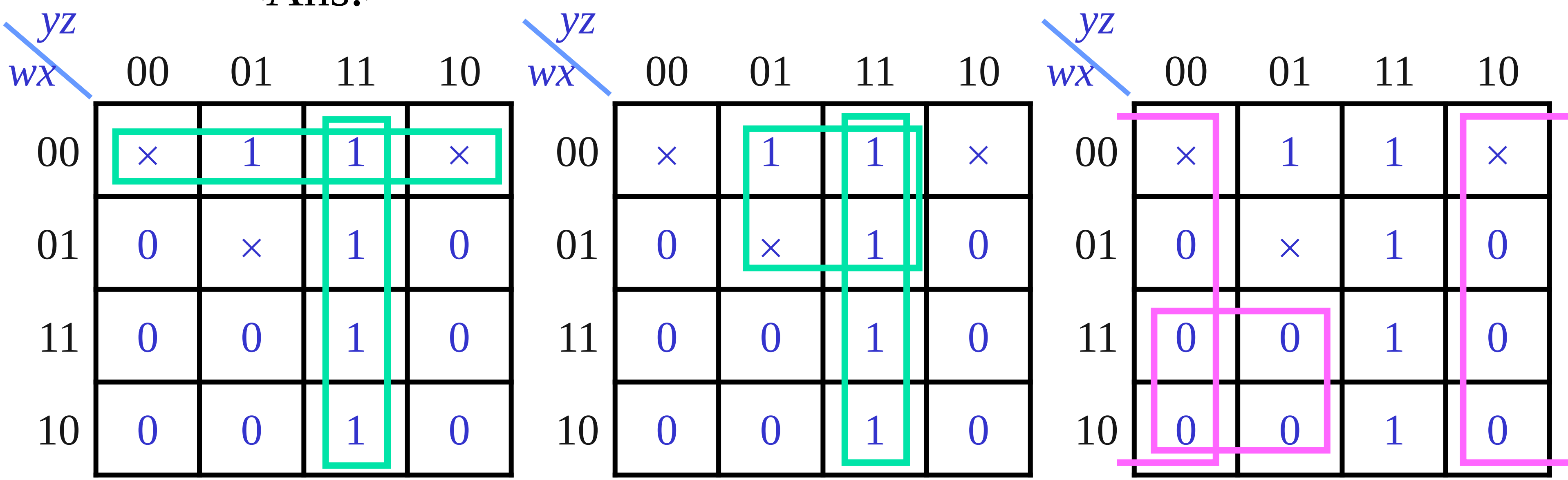
- Simplification of an incompletely specified function:
  - When choosing adjacent squares to simplify the function in the map, the ×'s may be assumed to be either 0 or 1, whichever gives the simplest expression.
  - An × need not be used at all if it does not contribute to covering a larger area.

Example 3.8

* The outputs in a particular implementation of the function are only 0's and 1's.

- Simplify the Boolean function $F(w,x,y,z) = \Sigma m(1,3,7,11,15)$ which has the ***don't-care*** conditions $\mathbf{d}(w,x,y,z) = \Sigma m(0,2,5)$.

<Ans.>



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

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

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

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

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

(b) **POS**: $F(w,x,y,z) = z(w' + y) = \Sigma m(1,3,5,7,11,15)$

補充資料:

Quine-McCluskey Method
&
CAD Tools for Simplification

A. Quine-McCluskey Method (for SOP)

- Tabular method to systematically find all PIs and a minimum cover of PIs for a function

Implication Table

1. Find all prime implicants ($xy + xy' = x$)

(a) Fill Column 1 with minterm and don’t-care condition indices.

Group by number of 1's.

(b) Apply theorem $xy + xy' = x$

Compare elements of adjacent groups.

Differ by one bit ⇒ Combine (eliminate a variable) and place in next column.

Mark the combined elements with a check (✓).

Repeat until no further combinations may be made.

Mark the uncombined elements with a star (*) ⇒ PI.

2. Find the minimum cover of PIs ⇒ simplified SOP

PI Chart

- Tabular method to systematically find all PIs and a minimum cover of PIs for a function

1. Find all prime implicants (*xy* + *xy*′= *x*)

2. Find the minimum cover of PIs

⇒ simplified SOP

PI Chart

(a) Construct the Prime Implicant Chart

(b) Find the EPIs

(* EPIs must be in the set)

(c) Select PIs to cover the remaining minterms if necessary

(* Form the minimum set)

Finding All Prime Implicants (1/4)

0100 0110 1001 1101 0000 0111 1111

F(A,B,C,D) = Σm(4, 5, 6, 8, 9, 10, 13) + Σd(0, 7, 15)

0101 1000 1010

1. Find all prime implicants: Implication Table

(a) Fill Column 1 with *minterm* and *don't-care condition* indices.

* Group by number of 1's.

Implication Table

	Column 1		
0 1's	0	0000	
1 1's	4	0100	
	8	1000	
2 1's	5	0101	
	6	0110	
	9	1001	
	10	1010	
3 1's	7	0111	
	13	1101	
4 1's	15	1111	

Finding All Prime Implicants (2/4)

(b) Apply theorem *xy* + *xy'* = *x* :

- Compare elements of adjacent groups.
- Differ by one bit
- ⇒ Combine (eliminate a variable) and place in next column.
- Mark the combined elements with a check (✓).
- Repeat until no further combinations may be made.
- Mark the uncombined elements with a star (*) ⇒ PI.

Implication Table

Group	Column 1	Column 2	
0	0 0000 ✓	0,4 0-00 0,8 -000	
1	4 0100 ✓ 8 1000 ✓	4, 5 010- 4, 6 01-0 8, 9 100- 8, 10 10-0	
2	5 0101 ✓ 6 0110 ✓ 9 1001 ✓ 10 1010 ✓	5,7 01-1 5,13 -101 6,7 011- 9,13 1-01	
3	7 0111 ✓ 13 1101 ✓	7,15 -111 13,15 11-1	
4	15 1111 ✓		

Finding All Prime Implicants (3/4)

(b) Apply theorem *xy* + *xy′*= *x* :

Compare elements of adjacent groups.

Differ by one bit

⇒ Combine (eliminate a variable) and place in next column.

Mark the combined elements with a check (✓).

Repeat until no further combinations may be made.

Mark the uncombined elements with a star (*) ⇒ PI.

Implication Table

Group	Column 1	Column 2	Column 3
0	0 0000 ✓	0,4 0-00 * 0,8 -000 *	4,5,6,7 01- - *
1	4 0100 ✓ 8 1000 ✓	4, 5 010- ✓ 4, 6 01-0 ✓ 8, 9 100- * 8, 10 10-0 *	5,7,13,15 -1-1 *
2	5 0101 ✓ 6 0110 ✓ 9 1001 ✓ 10 1010 ✓	5,7 01-1 ✓ 5,13 -101 ✓ 6,7 011- ✓ 9,13 1-01 *	
3	7 0111 ✓ 13 1101 ✓	7,15 -111 ✓ 13,15 11-1 ✓	
4	15 1111 ✓		

Finding All Prime Implicants (4/4)

Implication Table

Column 1	Column 2	Column 3
0 0000 ✓	0,4 0-00 *	4,5,6,7 01- - *
	0,8 -000 *	
4 0100 ✓		5,7,13,15 -1-1 *
8 1000 ✓	4, 5 010- ✓	
	4, 6 01-0 ✓	
5 0101 ✓	8, 9 100- *	
6 0110 ✓	8, 10 10-0 *	
9 1001 ✓		
10 1010 ✓	5,7 01-1 ✓	
	5,13 -101 ✓	
7 0111 ✓	6,7 011- ✓	
13 1101 ✓	9,13 1-01 *	
	7,15 -111 ✓	
15 1111 ✓	13,15 11-1 ✓	

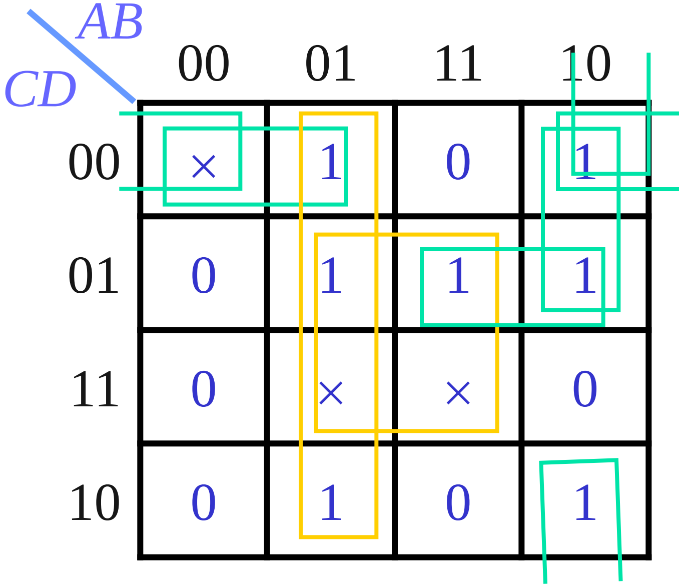
Prime Implicants:

0,4	0-00	→ A′ C′ D′
0,8	-000	→ B′ C′ D′
8,9	100-	→ A B′ C′
8,10	10-0	→ A B′ D′
9,13	1-01	→ A C′ D
4,5,6,7	01- -	→ A′ B
5,7,13,15	-1-1	→ B D

(For Verification Only)

Find all PIs by using K-map: (for verification only)

F(A,B,C,D) = Σm(4,5,6,8,9,10,13) + Σd(0,7,15)



CD / AB	00	01	11	10
00	×	1	0	1
01	0	1	1	1
11	0	×	×	0
10	0	1	0	1

Prime Implicants:

0-00 → A′ C′ D′

-000 → B′ C′ D′

100- → A B′ C′

10-0 → A B′ D′

1-01 → A C′ D

01- - → A′ B

-1-1 → B D

Finding the Minimum Cover (1/4)

2. **Find the smallest set of PIs that cover all the minterms:**

   Prime Implicant Chart

   (a) Construct the Prime Implicant Chart

   (b) Find the EPIs (* EPIs must be in the set)

   (c) Select PIs to cover the remaining minterms if necessary (* Form the minimum set)

Finding the Minimum Cover (2/4)

(a) Construct the PI chart:

Rows = prime implicants

Columns = minterms only

(excludes don’t-care conditions)

Place an “X” if minterm is covered by the PI.

F(A,B,C,D) = Σm(4,5,6,8,9,10,13)
+ Σd(0,7,15)

Prime Implicants:

0-00 = A′ C′ D′

-000 = B′ C′ D′

100- = A B′ C′

10-0 = A B′ D′

1-01 = A C′ D

01- - = A′ B

-1-1 = B D

Prime Implicant Chart

	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
8,10 (10–0)				X		X	
9,13 (1–01)					X		X
4,5,6,7 (01––)	X	X	X				
5,7,13,15 (–1–1)		X					X

Finding the Minimum Cover (3/4)

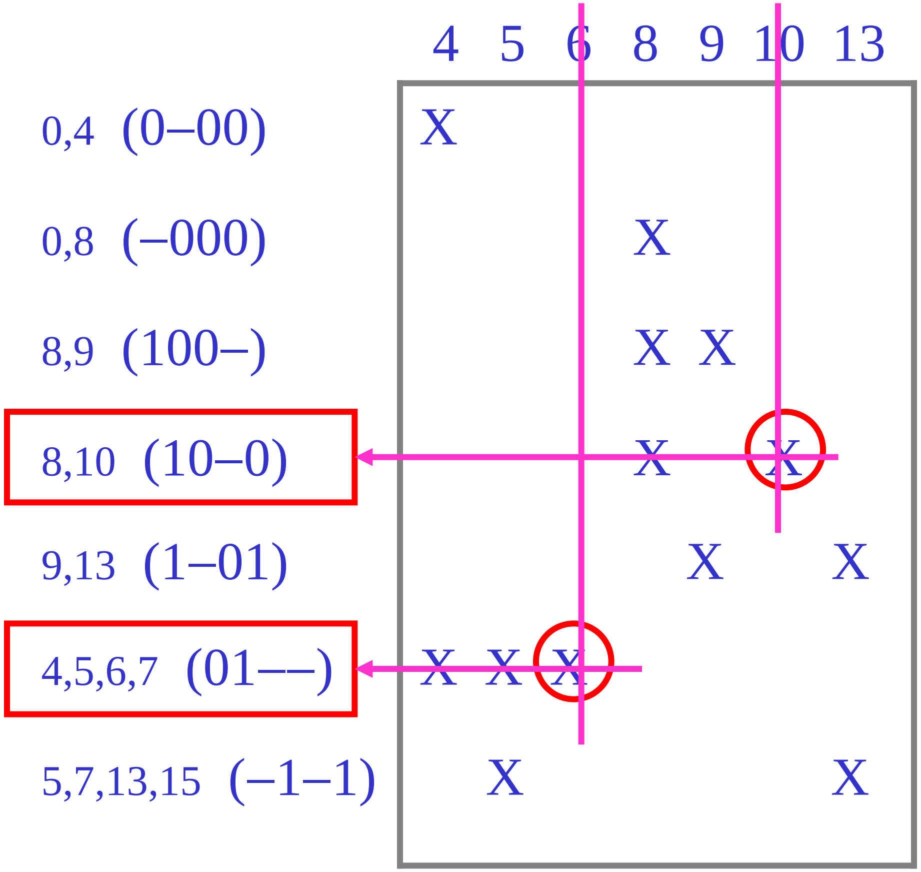
(b) Find the EPIs

(* EPIs must be in the set)

If a column has a single X,
then the PI associated w/ the row
is essential. (EPI)
It must appear in minimum cover.

2 EPIs: A B′ D′ , A′ B

F = A B′ D′ + A′ B + …



	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
8,10 (10–0)				X		X	
9,13 (1–01)					X		X
4,5,6,7 (01––)	X	X	X				
5,7,13,15 (–1–1)		X					X

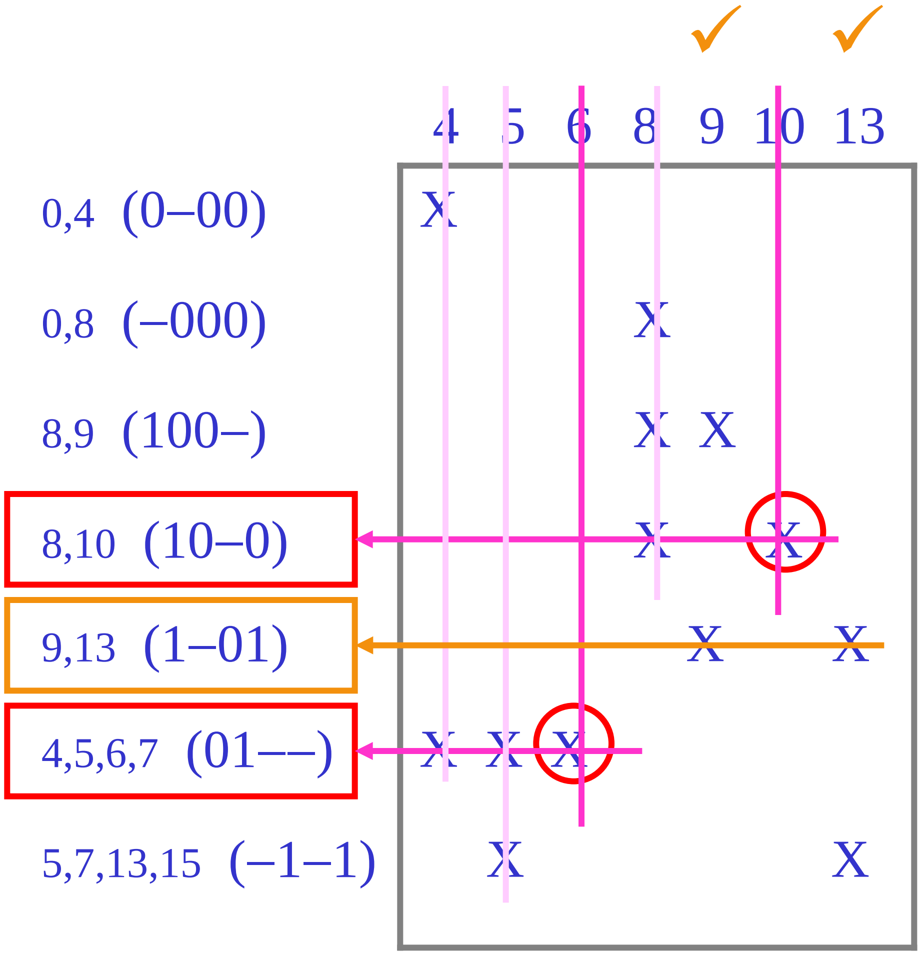
Finding the Minimum Cover (4/4)

(c) Select PIs to cover the remaining minterms if necessary

(* Form the minimum set)

Eliminate all columns covered by EPI.

Find minimum set of rows that cover the remaining columns.



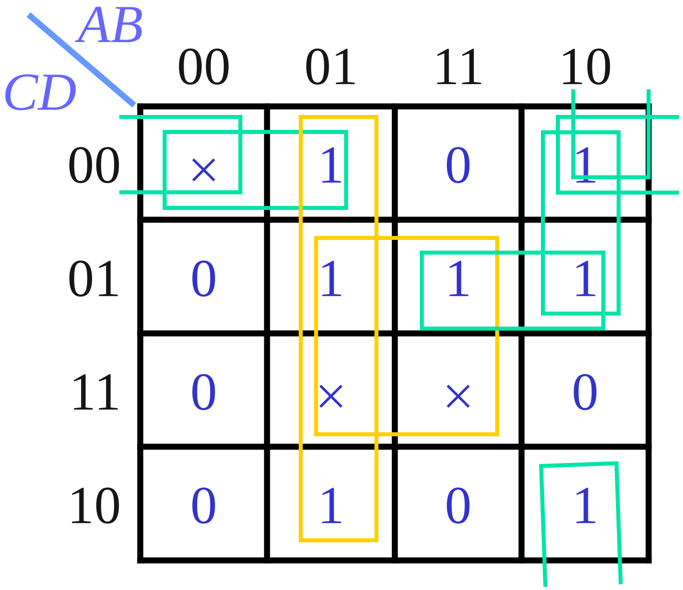
					✓		✓
	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
8,10 (10–0)				X		X	
9,13 (1–01)					X		X
4,5,6,7 (01––)	X	X	X				
5,7,13,15 (–1–1)		X					X

F = A B′ D′ + A′ B + A C′ D

(For Verification Only)

Find the smallest set of PIs that cover all the minterms: (for verification only)

$$F(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$$



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

Prime Implicants:

0-00 → A′ C′ D′
-000 → B′ C′ D′
100- → A B′ C′
10-0 → A B′ D′
1-01 → A C′ D
01- - → A′ B
-1-1 → B D

Essential Prime Implicants:

10-0 → A B′ D′　**01- - → A′ B**

F = A B′ D′ + A′ B + A C′ D

B. CAD Tools for Simplification

- Problems of Quine-McCluskey Method:
  - The # of PIs grows very quickly as the # of inputs increases.
  - Finding a min set cover is a very difficult problem.
    - an NP-complete problem
    - There are not likely to be any efficient algorithms for solving it.
- Espresso:
  - a program for 2-level Boolean function minimization
  - combines many of the best heuristic techniques developed
    - don’t generate all PIs
    - judiciously select a subset of primes that still covers the minterms

Espresso Inputs and Outputs

$$F(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) \; + d(0,7,15)$$

Espresso Input

.i 4	# inputs
.o 1	# outputs
.ilb a b c d	input names
.ob f	output name
.p 10	# product terms
0100 1	A′BC′D′
0101 1	A′BC′D
0110 1	A′BCD′
1000 1	AB′C′D′
1001 1	AB′C′D
1010 1	AB′CD′
1101 1	ABC′D
0000 -	A′B′C′D′ don’t care
0111 -	A′BCD don’t care
1111 -	ABCD don’t care
.e	end of list

Espresso Output

```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01   1
10-0   1
01--   1
.e
```

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

補充資料:

Multiple-Level Circuit Optimization

J.J. Shann

Multiple-Level Circuit Optimization

- 2-level ckt optimization: simplified SoP, PoS
  - can reduce the cost of combinational logic ckts
  - 2-level ckt: minimal propagation delay
- Multi-level ckts:
  - ckts w/ more than 2 levels
  - There are often additional cost saving available
- Reference:
  - M. Morris Mano & Charles R. Kime, *Logic and Computer Design Fundamentals*, 3rd Edition, 2004, Pearson Prentice Hall. (§2-6)

Transformations for Multiple-level Optimization

- Multiple-level ckt optimization (simplification):
  - is based on the use of a set of transformations that are applied in conjunction w/ cost evaluation to find a good, but not necessarily optimum solution.

GIC, delay

- Transformations:
  - ***Factoring***: **for *GIC* ↓**
    - is finding a factored form from either a SoP or PoS expression for a function → distributive law
  - ***Elimination***: **for *delay* ↓**
    - function G in an expression for function F is replaced by the expression for G → distributive law

A. Transformation for GIC Reduction (Factoring)

- E.g.: $G = ABC + ABD + E + ACF + ADF$

<Ans.>

2-level implementation: gate-input cost = 17

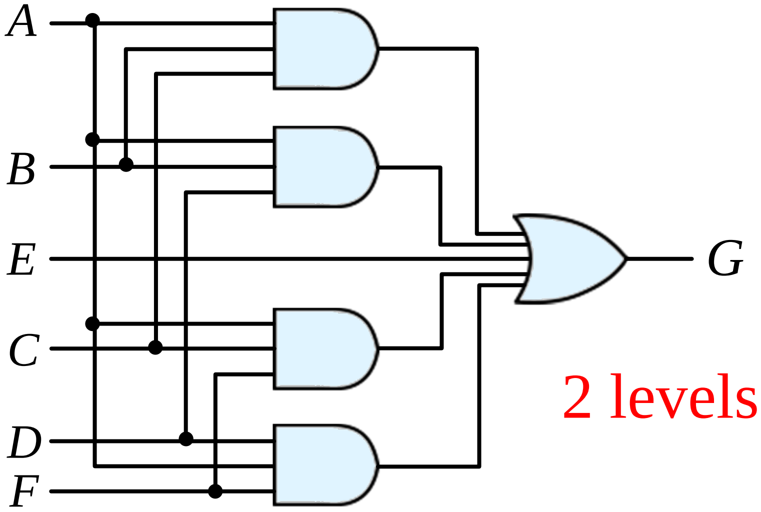
Multi-level implementation: distributive law

$$
\begin{aligned}
G &= \underline{ABC} + \underline{ABD} + E + \underline{ACF} + \underline{ADF} && \text{(a)} \rightarrow 17 \\
&= \underline{\underline{AB(C + D)}} + E + \underline{\underline{AF(C + D)}} && \text{(b)} \rightarrow 13 \\
&= (\underline{\underline{AB} + \underline{AF}) (C + D)} + E && \text{(c)} \rightarrow 12 \\
&= \underline{A(B + F)} (C + D) + E && \text{(d)} \rightarrow 9
\end{aligned}
$$

Gate input count (GIC)

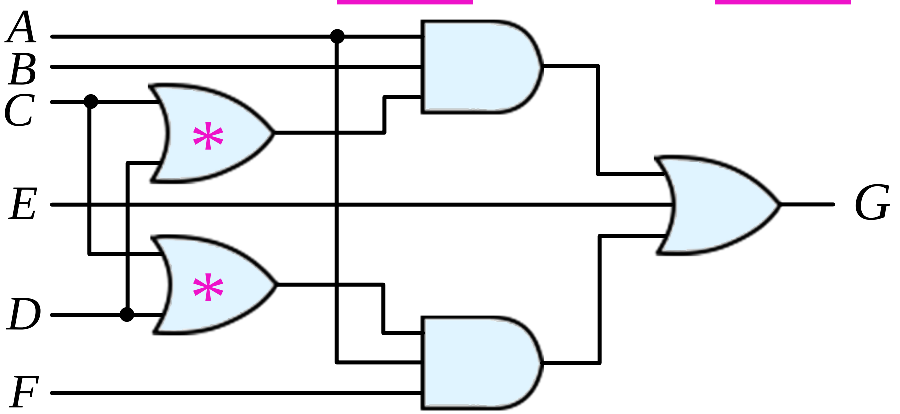
$$\begin{aligned} G &= ABC + ABD + E + ACF + ADF && \text{(a)} \\ &= AB(C+D) + E + AF(C+D) && \text{(b)} \\ &= (AB + AF)(C+D) + E && \text{(c)} \\ &= A(B+F)(C+D) + E && \text{(d)} \end{aligned}$$

(a) $G = ABC + ABD + E + ACF + ADF$

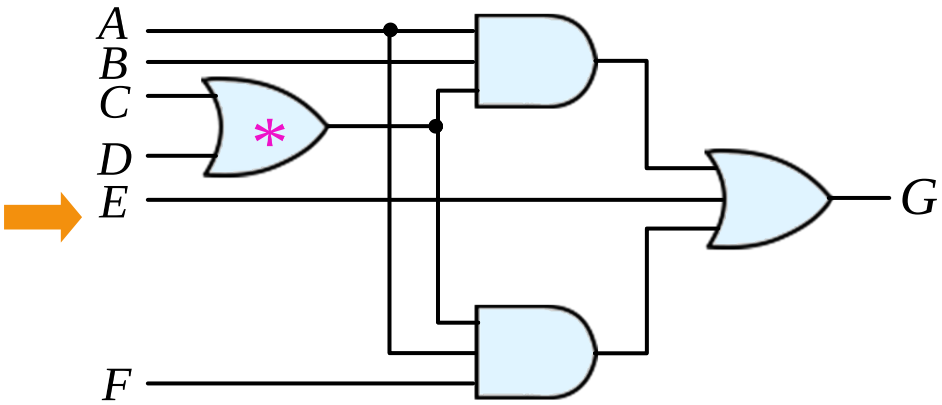


2 levels, GIC = 17

(b) $G = AB(\underline{C+D}) + E + A(\underline{C+D})F$



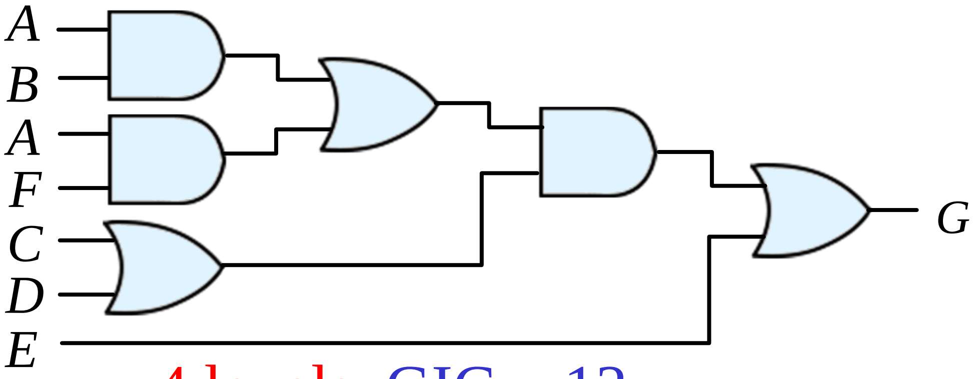
3 levels, GIC = 13



3 levels, GIC = 11

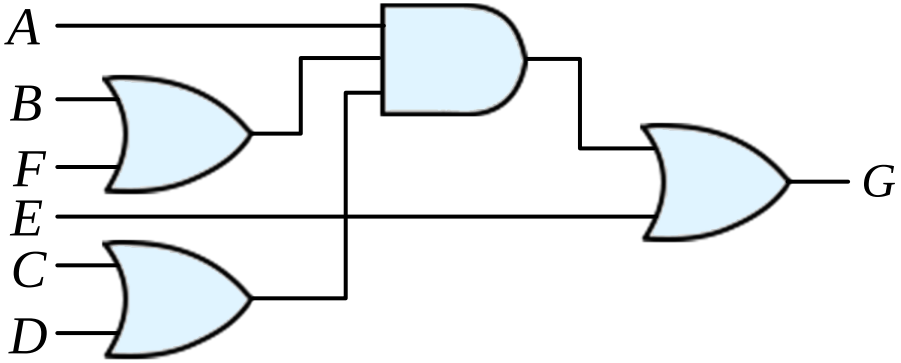
$$
\begin{aligned}
G &= ABC + ABD + E + ACF + ADF && \text{(a)}\\
&= AB(C+D) + E + AF(C+D) && \text{(b)}\\
&= (AB+AF)(C+D) + E && \text{(c)}\\
&= A(B+F)(C+D) + E && \text{(d)}
\end{aligned}
$$

(c) $G = (AB + AF)(C + D) + E$



4 levels, GIC = 12

(d) $G = A(B + F)(C + D) + E$



3 levels, GIC = 9

Ckt	# Levels	GIC
(a)	2	17
(b)	3	13 (11)
(c)	4	12
(d)	3	9

*

Example

- E.g.: Multilevel optimization transformations

$$G = A\bar{C}E + A\bar{C}F + A\bar{D}E + A\bar{D}F + BCD\bar{E}\bar{F}$$

$$H = \bar{A}BCD + ABE + ABF + BCE + BCF$$

<Ans.>

$\underline{\text{GIC}}$

26

$$\begin{aligned}
G &= \underline{A\bar{C}E + A\bar{C}F + A\bar{D}E + A\bar{D}F} + BCD\bar{E}\bar{F} \\
&= \underline{A(\underline{\bar{C}E + \bar{C}F} + \underline{\bar{D}E + \bar{D}F})} + BCD\bar{E}\bar{F} \\
&= A(\underline{\underline{\bar{C}(E+F)} + \underline{\bar{D}(E+F)}}) + BCD\bar{E}\bar{F} \\
&= A\underline{(\bar{C}+\bar{D})(E+F)} + BCD\bar{E}\bar{F} \\
&= A(\bar{C}+\bar{D})X_2 + BX_1\bar{E}\bar{F} \\
&= A\overline{X_1}X_2 + BX_1\bar{X}_2
\end{aligned}$$

18

14

factoring

factoring

factoring

$$\boxed{\begin{aligned} X_1 &= CD \\ X_2 &= E+F \end{aligned}}$$

substitution

$$\boxed{\begin{aligned} &G = A\overline{X}_1 X_2 + BX_1\overline{X}_2 \\ &X_1 = CD \\ &X_2 = E + F \end{aligned}}$$

(Cont'd)

$$H = \underline{\overline{A}BCD + ABE + ABF + BCE + BCF}$$

$$= B(\underline{\overline{A}CD + AE + AF + CE + CF})$$

$$= B(\overline{A}CD + \underline{\underline{A(E+F)} + \underline{C(E+F)}}))$$

$$= B(\overline{A}(CD) + \underline{(A+C)(E+F)}))$$

$$= B(\overline{A}X_1 + (A+C)X_2)$$

factoring

factoring

factoring

$$\boxed{\begin{aligned} &X_1 = CD \\ &X_2 = E + F \end{aligned}}$$

substitution

$$\Rightarrow \boxed{\begin{aligned} &G = A\overline{X}_1 X_2 + BX_1\overline{X}_2 \\ &H = B(\overline{A}X_1 + (A+C)X_2) \\ &X_1 = CD \\ &X_2 = E + F \end{aligned}}$$

$$G = A\overline{C}E + A\overline{C}F + A\overline{D}E + A\overline{D}F + BCD\overline{E}\,\overline{F}$$

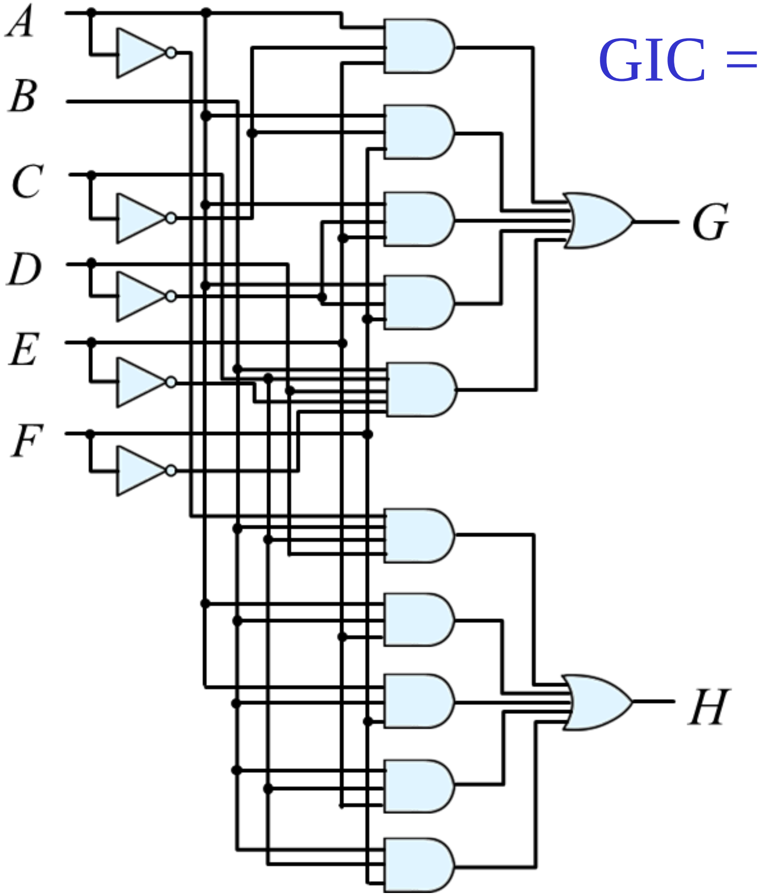
$$H = \overline{A}BCD + ABE + ABF + BCE + BCF$$

$$G = A\overline{X}_1X_2 + BX_1\overline{X}_2$$

$$H = B(\overline{A}X_1 + (A + C)X_2)$$

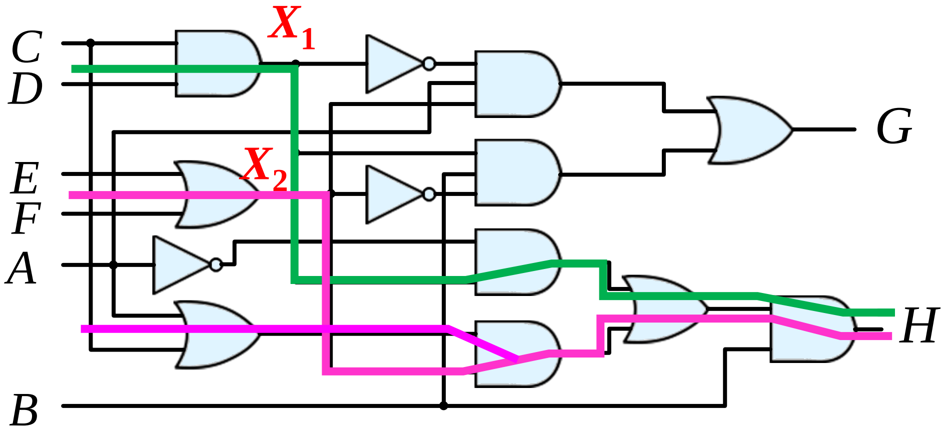
$$X_1 = CD$$

$$X_2 = E + F$$



GIC = 48

GIC = 25 (w/ shared terms)



* 4-level

* 2-level

(ignore the delay of NOT gates)

3-6 NAND & NOR Implementation

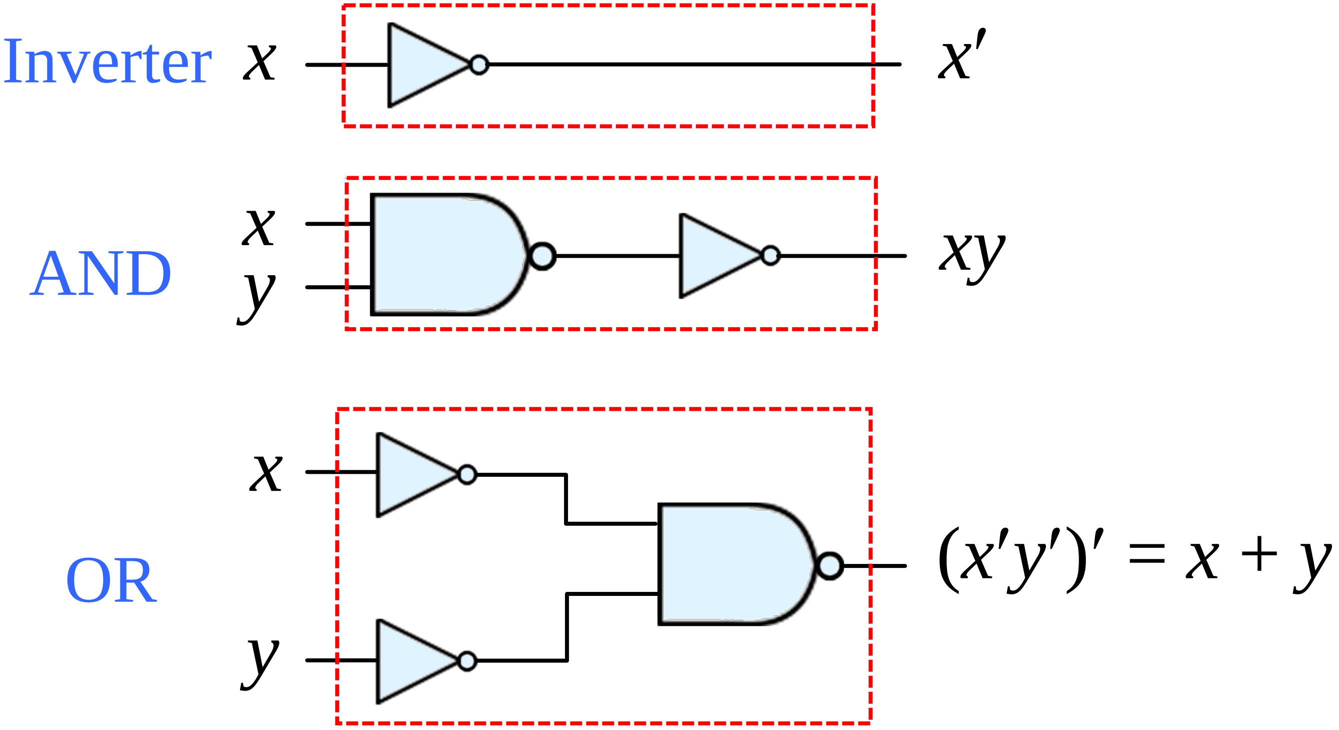
- NAND & NOR gates:
  - are easier to fabricate w/ electronic components.
  - are the basic gates used in all IC digital logic families.
  - have the universal property:
    - Any Boolean function can be implemented w/ NAND (NOR) gates only.

- Boolean function in terms of AND, OR, NOT

  ⇒ equivalent NAND (NOR) logic diagram

A. NAND Circuits

- Universal property of the NAND gate:
  - The logical operations of AND, OR, NOT can be obtained w/ NAND gates only.

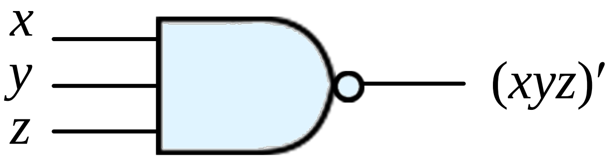


Inverter x → x'

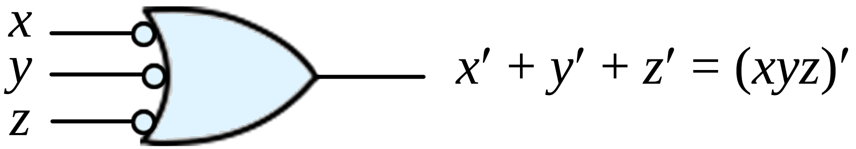
AND x, y → xy

OR x, y → $(x'y')' = x + y$

- Two graphic symbols for NAND gate:



AND-invert



Invert-OR

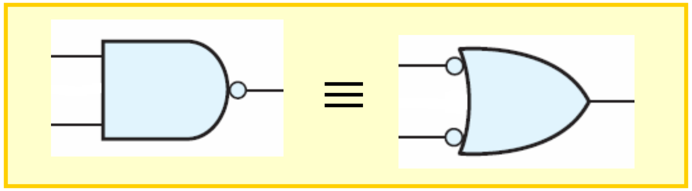
- Implementation of a combinational ckt w/ NAND gates:
  1. Obtain the simplified Boolean functions in terms of AND, OR, NOT.
  2. Convert the function to **NAND** logic.

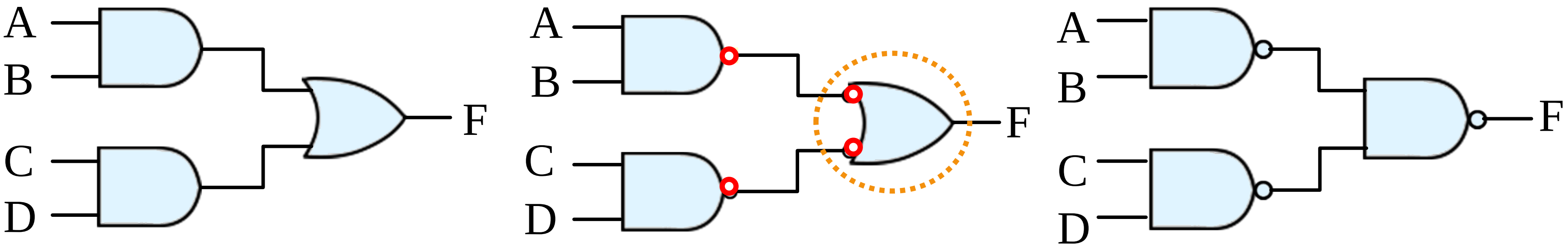
Two-Level NAND Implementation

- Two-level NAND implementation:

  Sum of products (AND-OR) ⇒ NAND-NAND

- Example: F = AB + CD

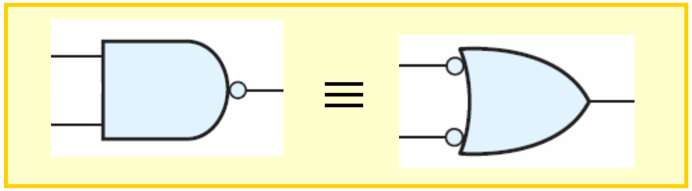




(a) → (c): $F = AB + CD = [(AB + CD)']' = [(AB)' (CD)']'$

(c) → (a): $F = [(AB)' (CD)']' = AB + CD$

Example 3.9



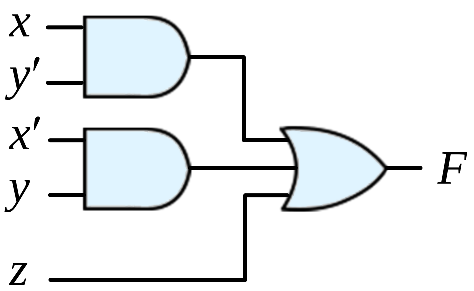
- Implement the following Boolean function w/ NAND gates: F(*x*,*y*,*z*) = Σ(1,2,3,4,5,7)

<Ans.>

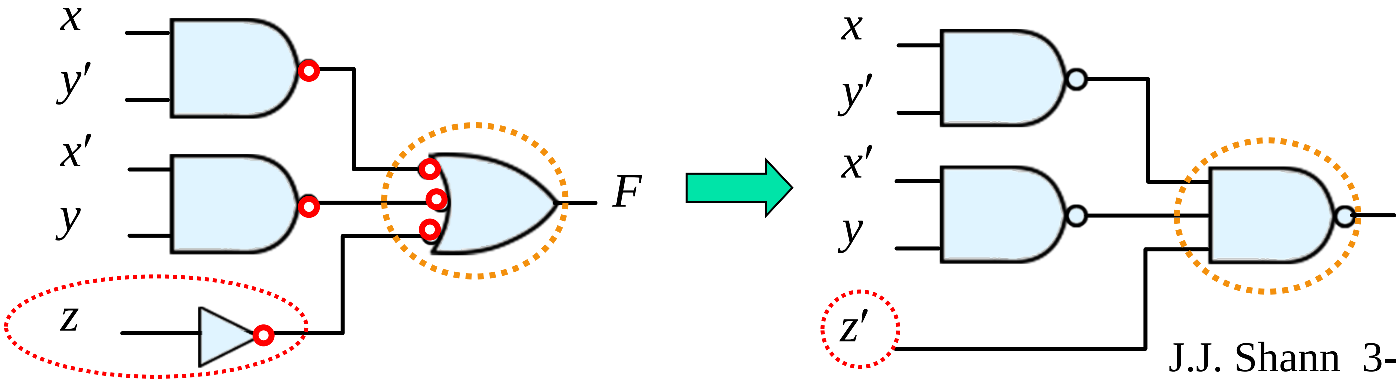
i. Simplify the function in SOP:

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

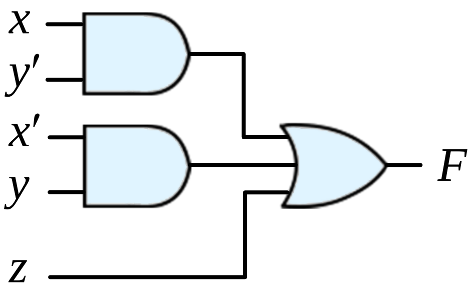
F = $xy' + x'y + z$

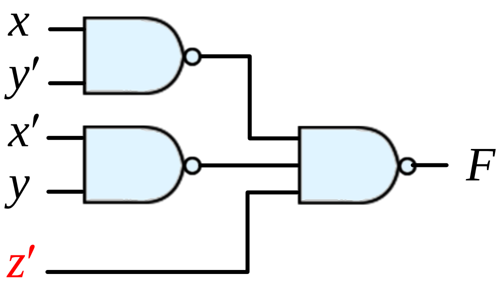


ii. Convert the function to NAND logic:



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



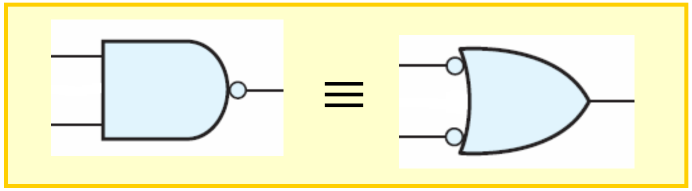


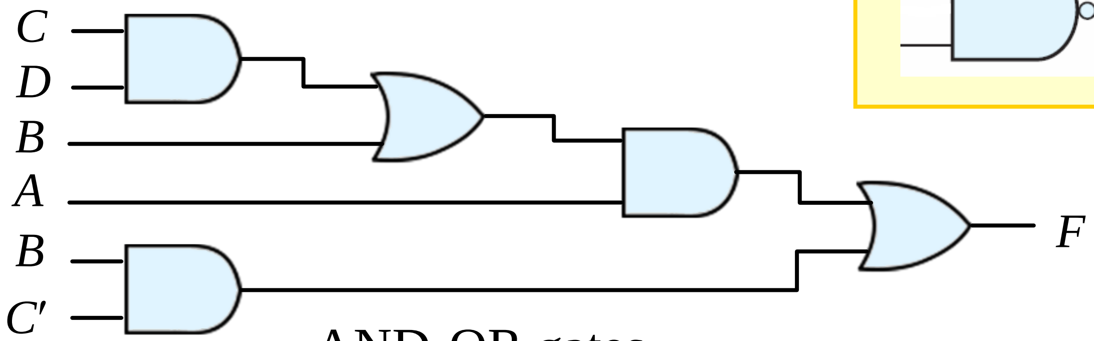
- Procedure of implementing a Boolean function w/ two-level NAND gates:
  1. Simplify the function and express it in sum of products.
  2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term.

     → 1st level
  3. Draw a single NAND gate (using the AND-invert or the invert-OR graphic symbol), w/ inputs coming from outputs of 1st level gates. → 2nd level
  4. A term w/ a single literal requires an inverter in the 1st level or may be complemented and applied as an input of the 2nd-level NAND gate.

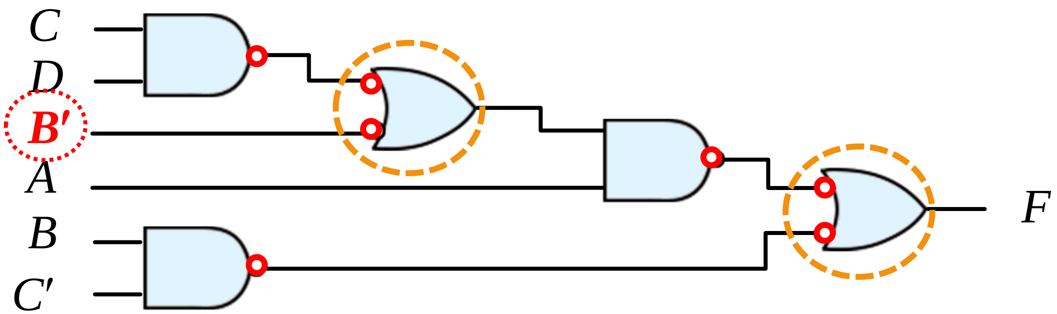
Multilevel NAND Circuits

- Standard form of Boolean function
  $\Rightarrow$ 2-level implementation
- Nonstandard form $\Rightarrow$ Multilevel circuit
  - E.g.: $F = A(CD + B) + BC'$





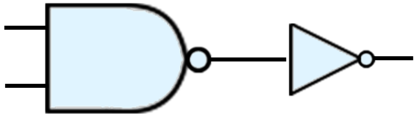
AND-OR gates



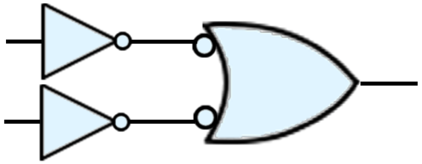
NAND gates

- Procedure for obtaining a multilevel NAND diagram:

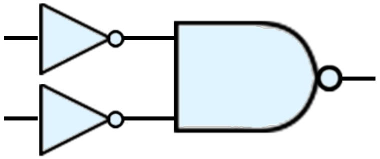
1. From a given Boolean expression, draw the logic diagram w/ AND, OR, and invert gates.
   - Assumption: Both the normal and complement inputs are available.
2. Convert all AND gates to NAND gates w/ AND-invert graphic symbol.



3. Convert all OR gates to NAND gates w/ invert-OR graphic symbols.

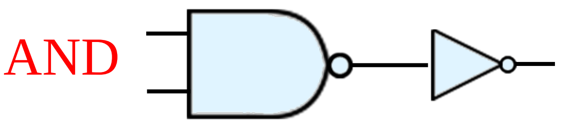


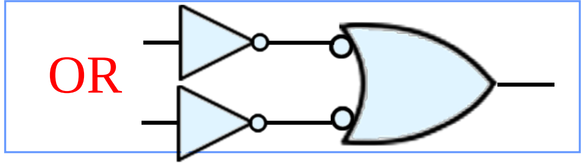
or



4. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (one-input NAND gate) or complement the input literal.

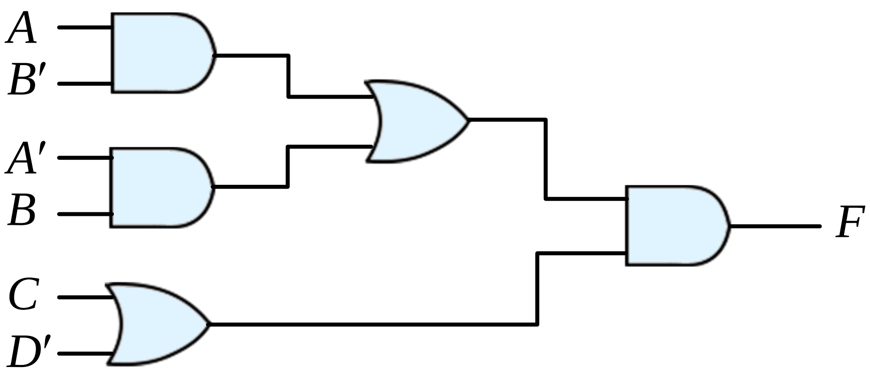
Example



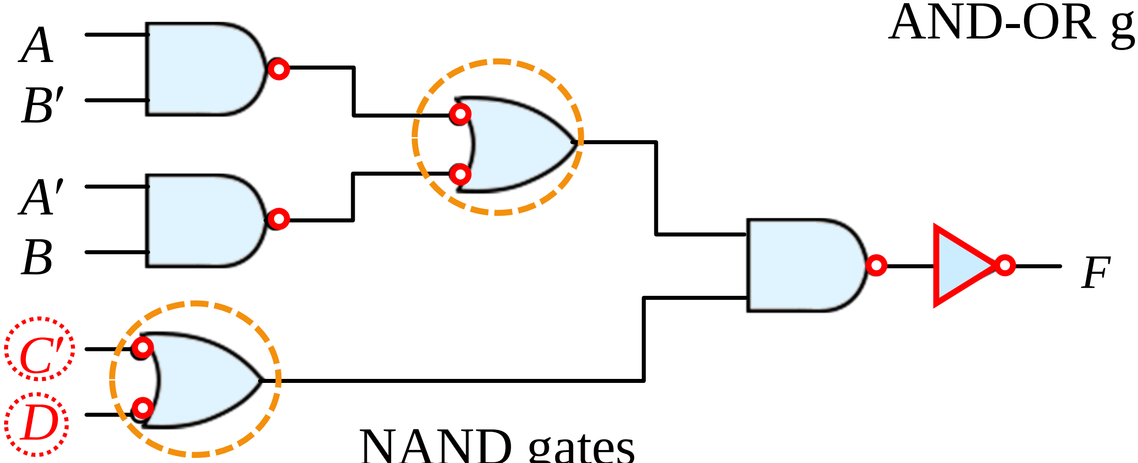


- Implement the multilevel Boolean function using NAND gates: F = (AB′ +A′B) (C + D′)

<Ans.>



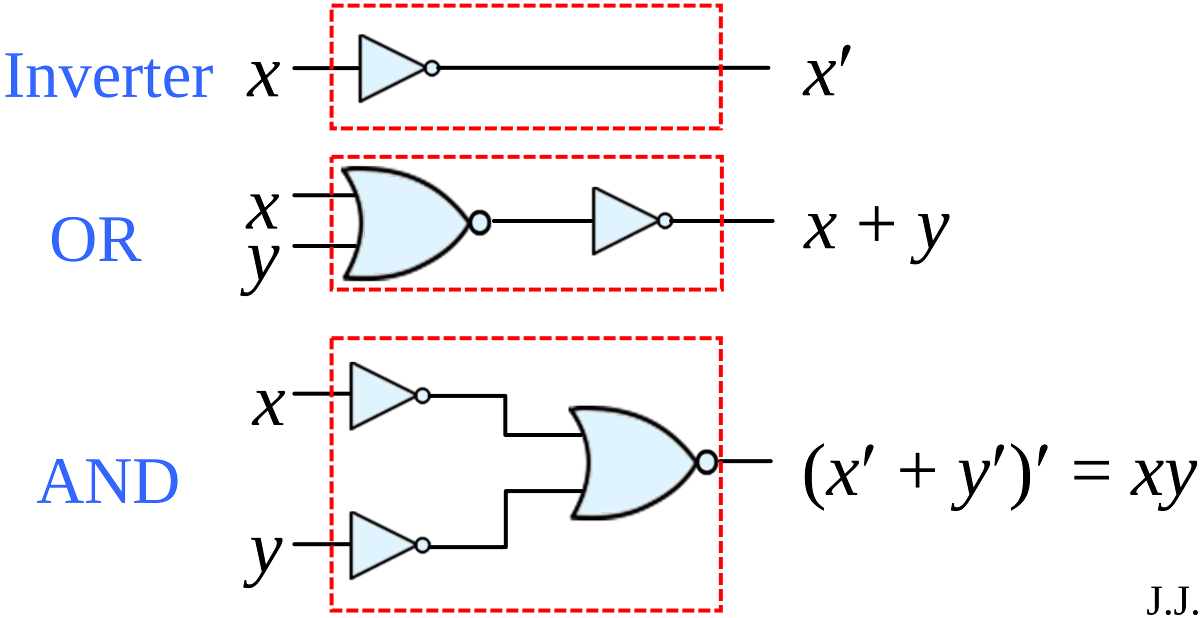
AND-OR gates



NAND gates

B. NOR Implementation

- NOR operation: the dual of NAND operation
  - All procedures and rules for NOR logic are the dual of those developed for NAND logic.
- Universal property of the NOR gate:
  - The logical operations of AND, OR, NOT can be obtained w/ NOR gates only.

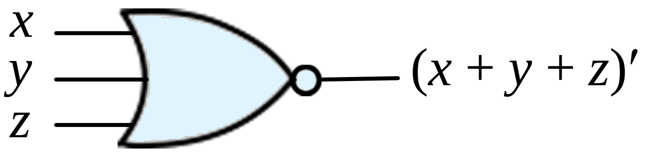


Inverter x → x'

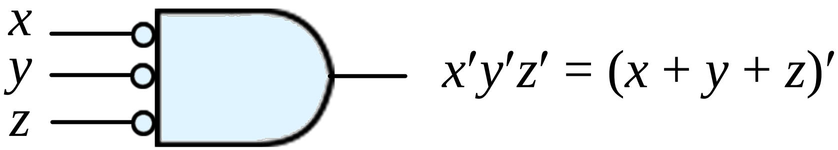
OR x, y → $x + y$

AND x, y → $(x' + y')' = xy$

- Two graphic symbols for NOR gate:



OR-invert



Invert-AND

- Implementation of a combinational ckt w/ NOR gates:
  1. Obtain the simplified Boolean functions in terms of AND, OR, NOT.
  2. Convert the function to **NOR** logic.

Two-Level NOR Implementation

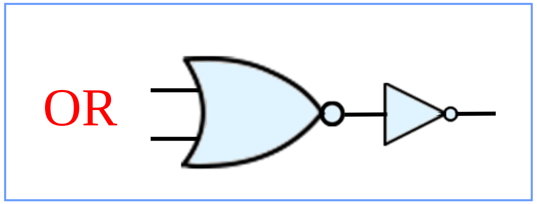
- Two-level NOR implementation:

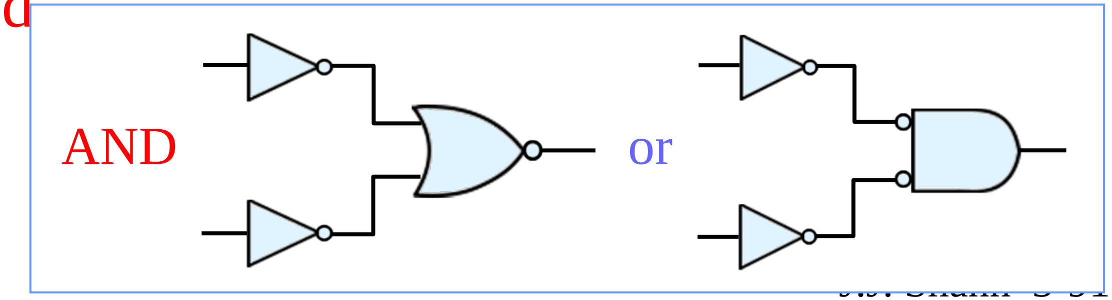
  Product of sums (OR-AND) ⇒ NOR-NOR

- Procedure of implementing a Boolean function w/ two-level NOR gates:

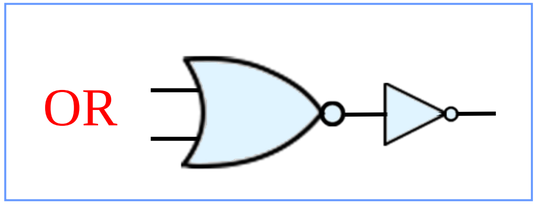
  1. Simplify the function and express it in product of sums.
  2. Draw the OR-AND diagram of the POS expression.
  3. OR gates → NOR gates w/ OR-invert graphic symbols.

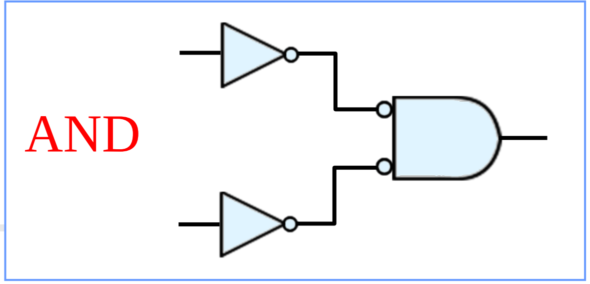
     AND gate → NOR gate w/ invert-AND graphic symbol.
  4. A single literal term going into the 2nd-level gate must be complemented

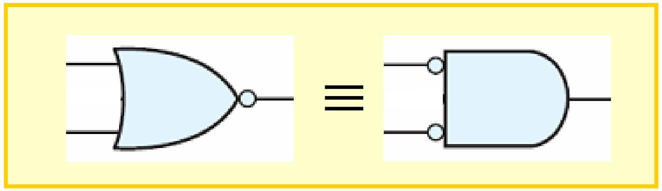


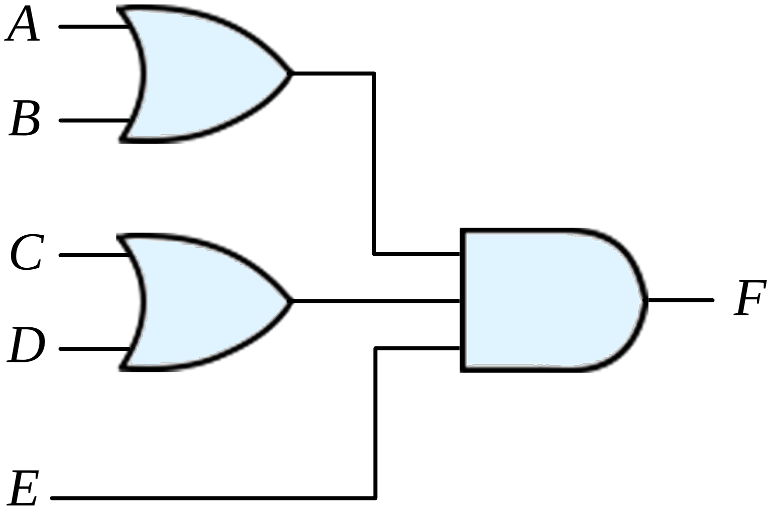


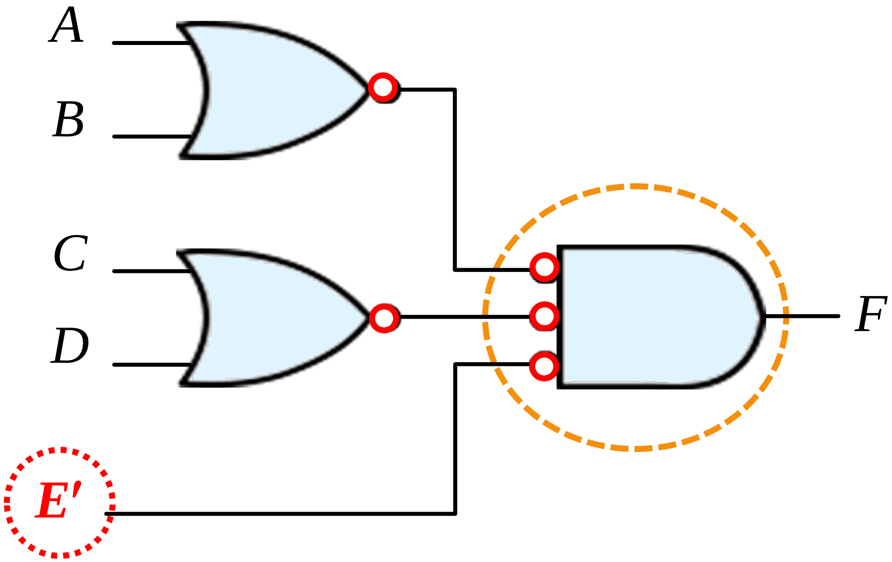
- Example: F = (A + B)(C + D)E





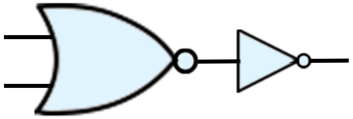




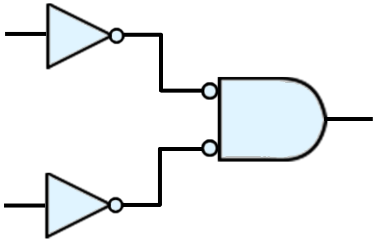


Multilevel NOR Circuits

- Procedure for obtaining a multilevel NOR diagram:
  1. From a given Boolean expression, draw the logic diagram w/ AND, OR, and invert gates.
     - Assumption: Both the normal and complement inputs are available.
  2. Convert each OR gate to a NOR gate w/ OR-invert symbol.

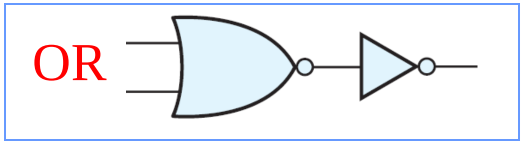


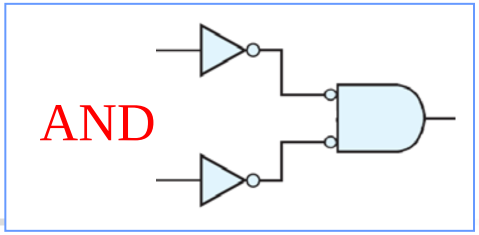
  3. Convert each AND gate to a NOR gates w/ invert-AND symbols.



  4. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (one-input NOR gate) or complement the input literal.

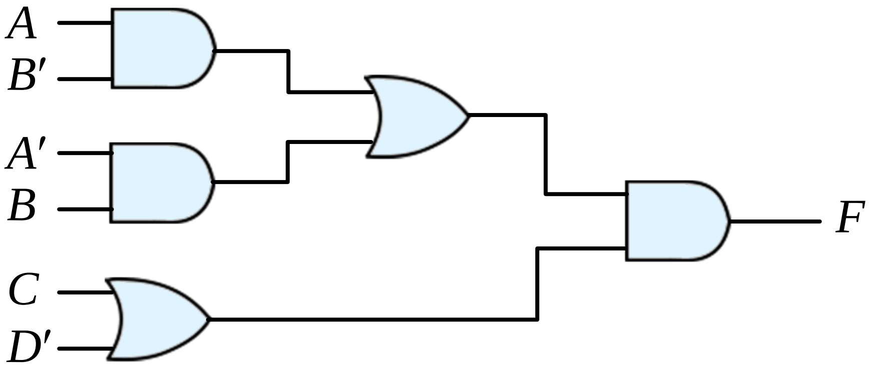
Example



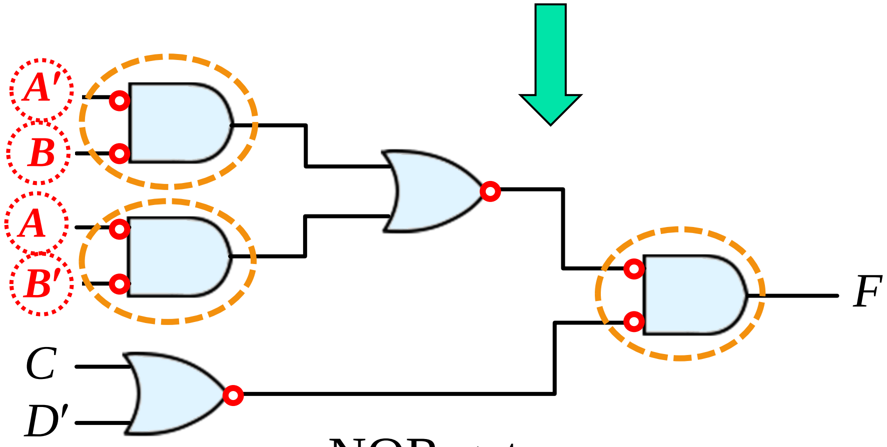


- Implement the multilevel Boolean function using NOR gates: F = (AB′ +A′B) (C + D′)

<Ans.>



AND-OR gates



NOR gates

3.7 Other Two-Level Implementations

- The types of gates most often found in ICs are NAND and NOR.
  - NAND and NOR logic implementations are the most important from a practical point of view.
- Wired logic:
  - Some NAND or NOR gates allow the possibility of a wire connection b/t the outputs of two gates to provide a specific logic function.
  - E.g.: Wired-AND logic

    Wired-OR logic

Wired Logic

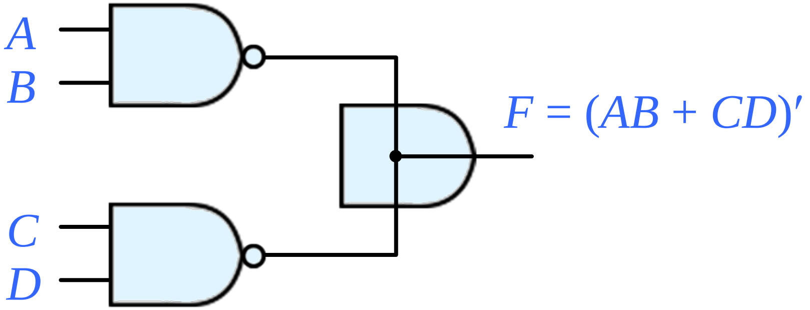
* The wired-logic gate is not a physical gate.

- Wired-AND logic:
  - E.g.: open-collector TTL NAND gates

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

    $= (AB + CD)'$

    ⇒ **AND-OR-INVERT (AOI)** function

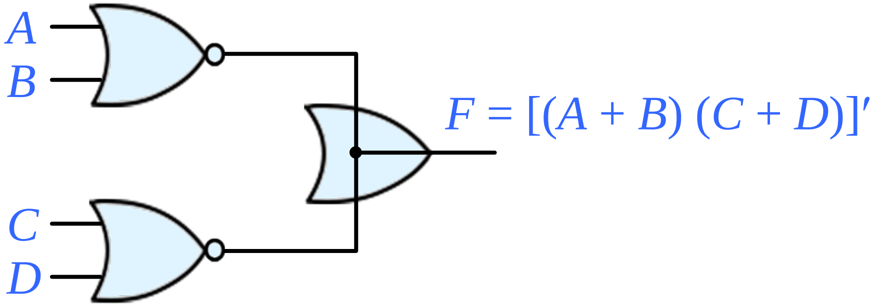


- Wired-OR logic:
  - E.g.: ECL NOR gates

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

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

    ⇒ **OR-AND-INVERT (OAI)** function



2-Level Combinations of Gates

- 2-level combinations of gates:

  AND, OR, NAND, NOR ⇒ 16

  – Degenerate forms: 8

  (degenerate to a single operation; 2-level → 1-level)

AND-AND	OR-OR	NAND-OR	NOR-AND
AND-NAND	OR-NOR	NAND-NOR	NOR-NAND

  – Nondegenerate forms: 8

AND-OR	OR-AND	NAND-NAND	NOR-NOR
NAND-AND	**NOR-OR**	AND-NOR	OR-NAND

  - AND-OR & NAND-NAND ⇔ sum of products (AO)
  - OR-AND & NOR-NOR ⇔ product of sums (OA)
  - **NAND-AND** **NOR-OR** AND-NOR OR-NAND ⇔ ?

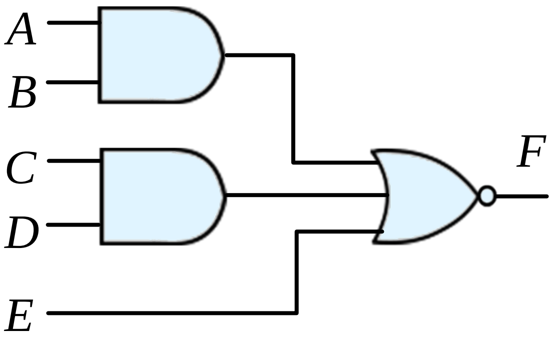
AOI (A.)

OAI (B.)

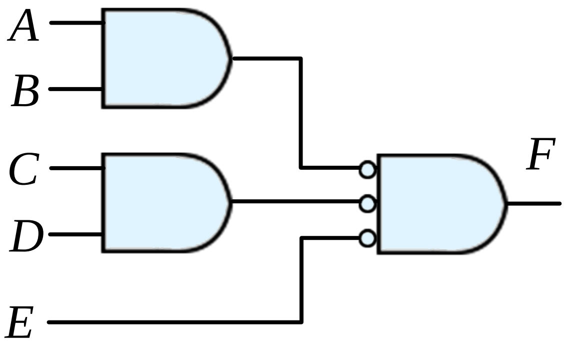
A. AND-OR-Invert (AOI) Implementation (AND-NOR & NAND-AND)

- AND-OR-Invert: inversion of SoP
  - Simplify F′ in SoP
  - E.g.: F′ = AB + CD + E (sum of products)

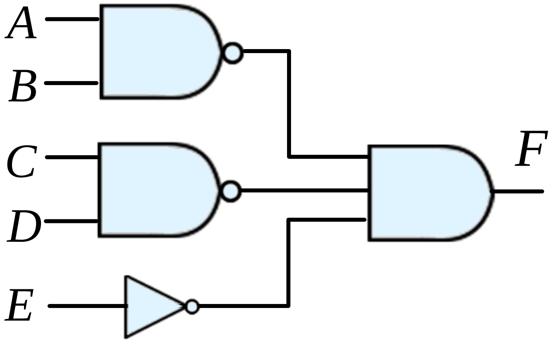
    ⇒ F = (AB + CD + E)′



AND-NOR



AND-NOR



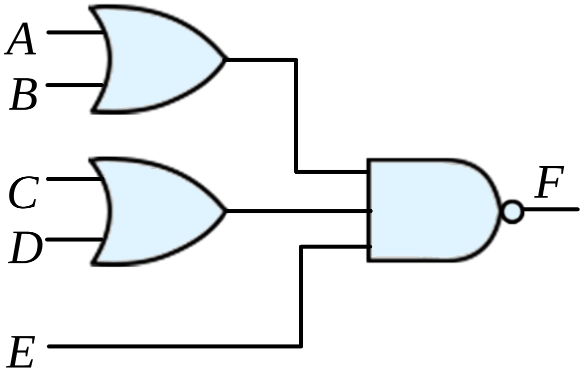
NAND-AND

AND-OR-Invert ⇔ AND-NOR ⇔ NAND-AND

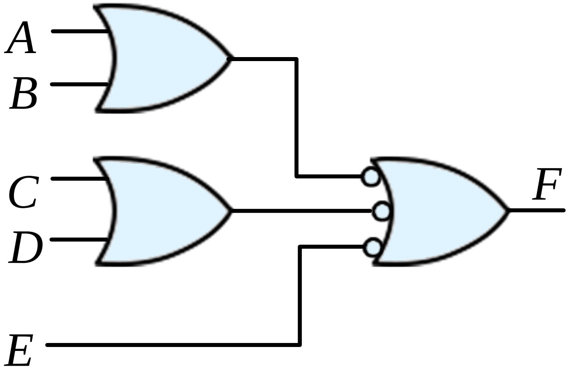
B. OR-AND-Invert (OAI) Implementation (OR-NAND & NOR-OR)

- OR-AND-Invert: inversion of PoS
  - Simplify F′ in PoS
  - E.g.: $F' = (A + B)(C + D)E$ (product of sums)

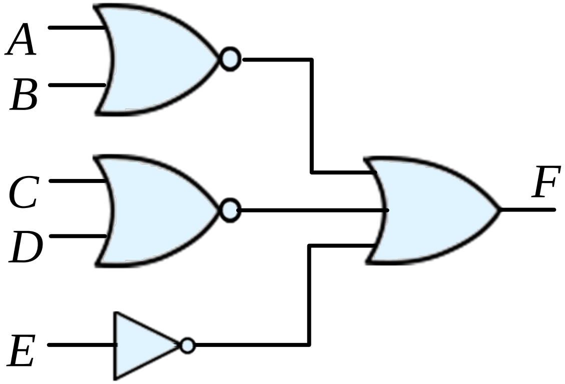
$$\Rightarrow F = [(A + B)(C + D)E]'$$



OR-NAND



OR-NAND



NOR-OR

OR-AND-Invert ⇔ OR-NAND ⇔ NOR-OR

C. Tabular Summary

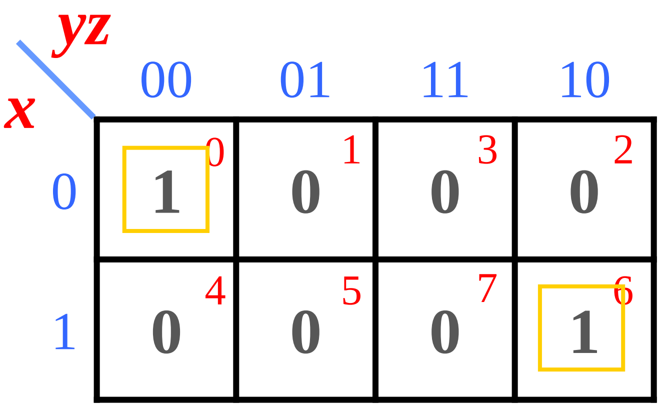
- Implementation w/ other 2-level forms:
  - p.129, Table 3.2

Example 3.10

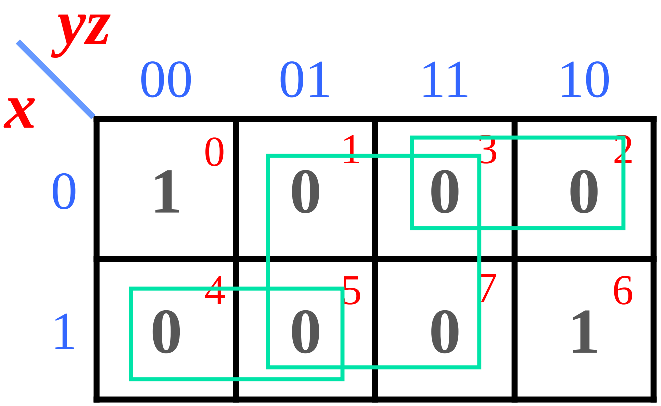
- Implement the following function w/ AND-NOR, NAND-AND, OR-NAND, NOR-OR 2-level forms:

$$F(x, y, z) = \Sigma(0, 6)$$

<Ans.>



x \ yz	00	01	11	10
0	0 1	1 0	3 0	2 0
1	4 0	5 0	7 0	6 1



x \ yz	00	01	11	10
0	0 1	1 0	3 0	2 0
1	4 0	5 0	7 0	6 1

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

PoS of F = $(x + y')(x' + y)z'$

SoP of F′ = $x'y + xy' + z$

PoS of F′ = $(x + y + z)(x' + y' + z)$

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

PoS of F = $(x + y')(x' + y)z'$

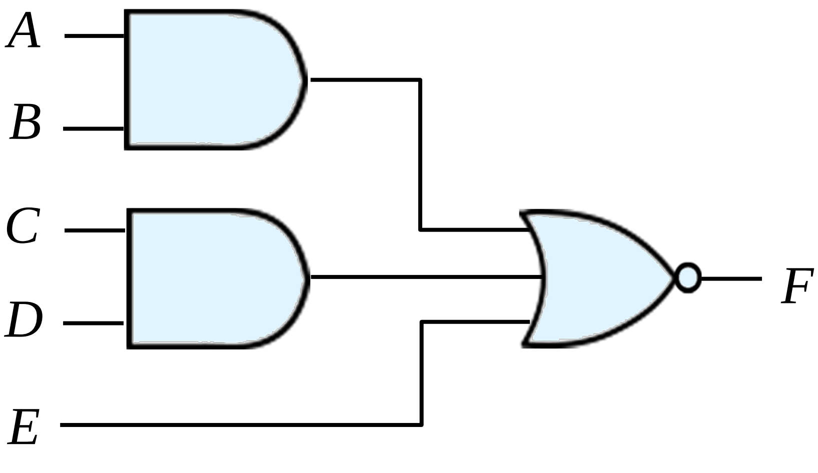
SoP of F′ = $x'y + xy' + z$

PoS of F′ = $(x + y + z)(x' + y' + z)$

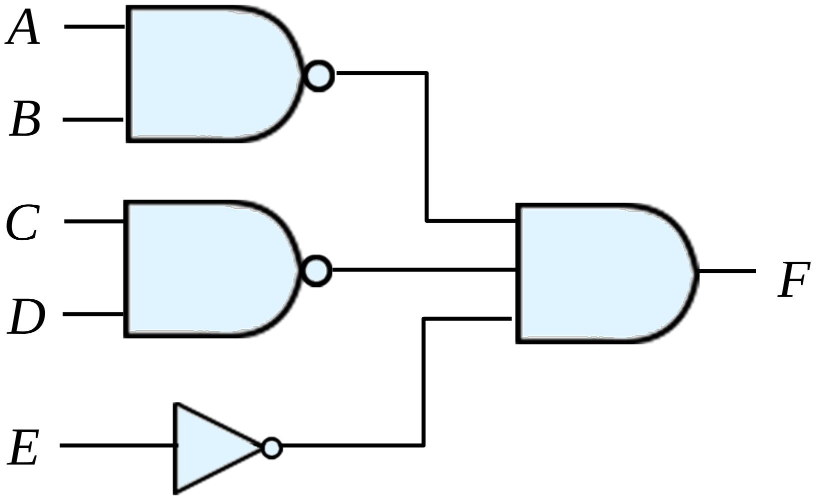
– AND-NOR & NAND-AND implementations: AOI

Simplify F′ in SoP: SoP of F′ = $x'y + xy' + z$

$\Rightarrow$ F = $(x'y + xy' + z)'$ … AOI



AND-NOR



NAND-AND

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

PoS of F = $(x + y')(x' + y)z'$

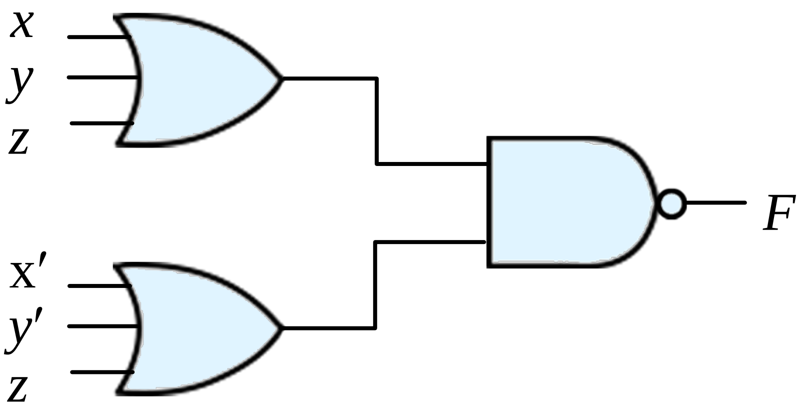
SoP of F′ = $x'y + xy' + z$

PoS of F′ = $(x + y + z)(x' + y' + z)$

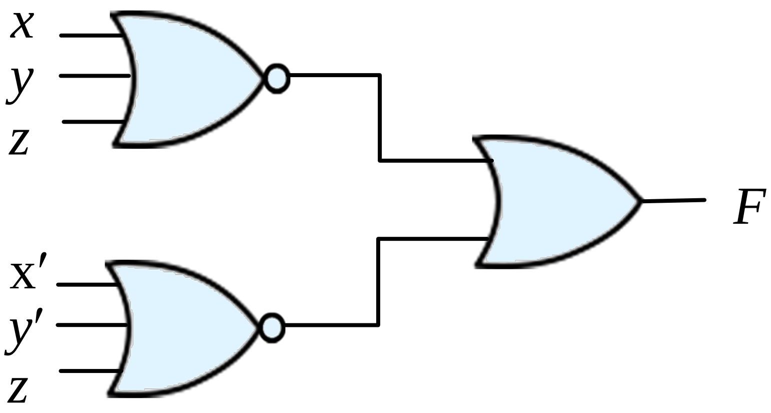
- OR-NAND & NOR-OR implementations: OAI

  Simplify F′ in PoS: PoS of F′ = $(x + y + z)(x' + y' + z)$

  $\Rightarrow$ F = $[(x + y + z)(x' + y' + z)]'$ … OAI



OR-NAND



NOR-OR

3-8 Exclusive-OR Function

- Exclusive-OR: XOR, ⊕

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

  – is equal to 1 if only $x = 1$ or if only $y = 1$, but not both.

- Exclusive-NOR: XNOR, equivalence

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

  – is equal to 1 if both x and y are equal to 1 or if both are equal to 0.

  * Two-variable XOR & XNOR are the complement to each other.

- They are particularly useful in arithmetic operations and error-detection and correction ckts.

Properties of XOR

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

- Identities:

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

$$x \oplus x = 0$$

$$x \oplus x' = 1$$

$$x \oplus y' = x' \oplus \mathrm{y} = (x \oplus y)'$$

- Commutativity and associativity:

$$\mathrm{A} \oplus \mathrm{B} = \mathrm{B} \oplus \mathrm{A}$$

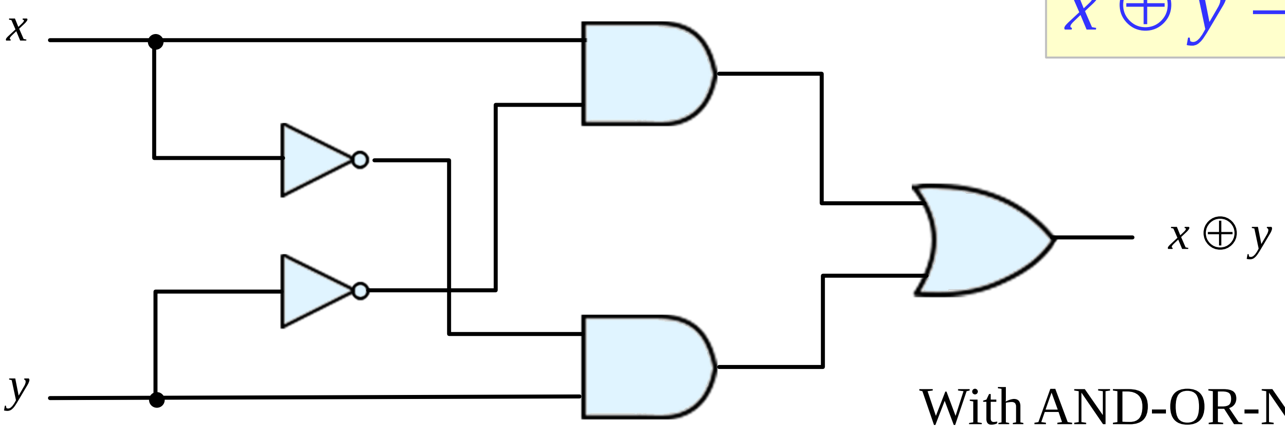
$$(\mathrm{A} \oplus \mathrm{B}) \oplus \mathrm{C} = \mathrm{A} \oplus (\mathrm{B} \oplus \mathrm{C}) = \mathrm{A} \oplus \mathrm{B} \oplus \mathrm{C}$$

⇒ XOR gates w/ three or more inputs

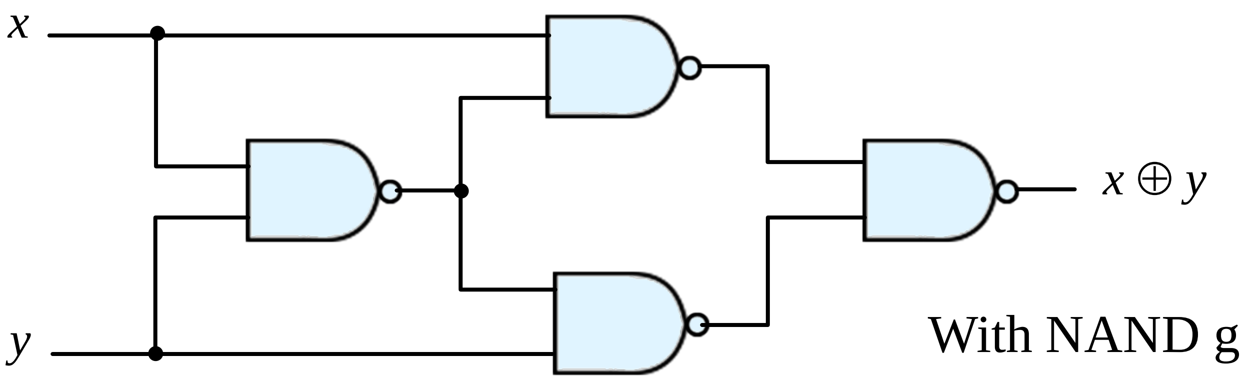
Implementations of XOR function

- XOR function is usually constructed w/ other types of gates:

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



With AND-OR-NOT gates



With NAND gates

Odd Function

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

- Multiple-variable XOR operation: odd function
  - equal to 1 if the input variables have an odd # of 1’s
- E.g.: 3-variable XOR

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

$\Rightarrow$ is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.

A \ BC	00	01	11	10
0	0	1 1	3	2 1
1	4 1	5	7 1	6

- E.g.: 4-variable XOR

$$
\begin{aligned}
& A \oplus B \oplus C \oplus D \\
&= (AB' + A'B) \oplus (CD' + C'D) \\
&= (AB' + A'B)\,(CD' + C'D)' \\
&\quad + (AB' + A'B)'\,(CD' + C'D) \\
&= (AB' + A'B)(CD + C'D') \\
&\quad + (AB + A'B')(CD' + C'D) \\
&= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)
\end{aligned}
$$

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

- An *n*-variable XOR function is defined as the logical sum of the $2^n/2$ minterms whose binary numerical values have an odd # of 1's.

Even Function

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

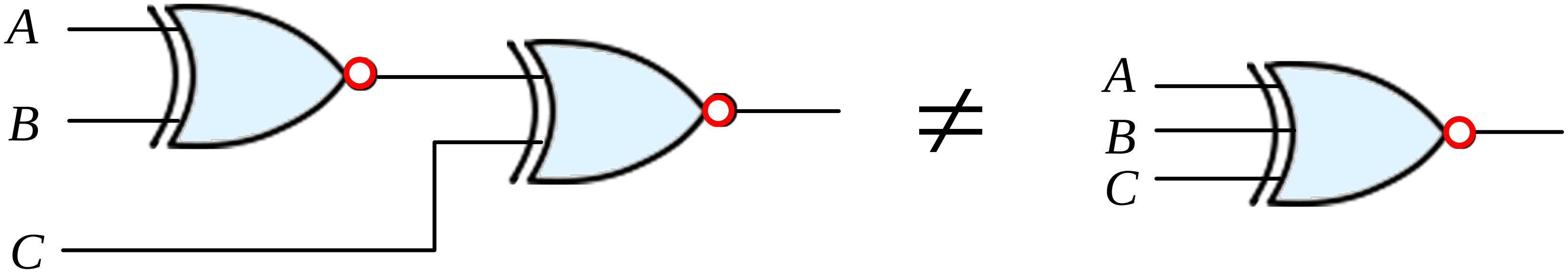
- XNOR is commutative and associative

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

$$[(A \oplus B)' \oplus C]' = [A \oplus (B \oplus C)']'$$

$$= A \oplus B \oplus C$$

$$\neq (A \oplus B \oplus C)'$$



- Modifying the definition of multiple-variable XNOR operation: even function
  - equal to 1 if the input variables have an even # of 1's

- E.g.: 3-variable XNOR

XOR: $A \oplus B \oplus C = \Sigma(1,2,4,7)$

XNOR: $(A \oplus B \oplus C)' = \Sigma(0,3,5,6)$

A \ BC	00	01	11	10
0	0	1 1	3	2 1
1	4 1	5	7 1	6

Odd function $F = A \oplus B \oplus C$

A \ BC	00	01	11	10
0	0 1	1	3 1	2
1	4	5 1	7	6 1

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

- E.g.: 4-variable XNOR

XOR: A⊕B⊕C⊕D $= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$

XNOR: (A⊕B⊕C⊕D)′ $= \Sigma(0, 3, 5, 6, 9, 10, 12, 15)$

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

Odd function F = A⊕B⊕C⊕D

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

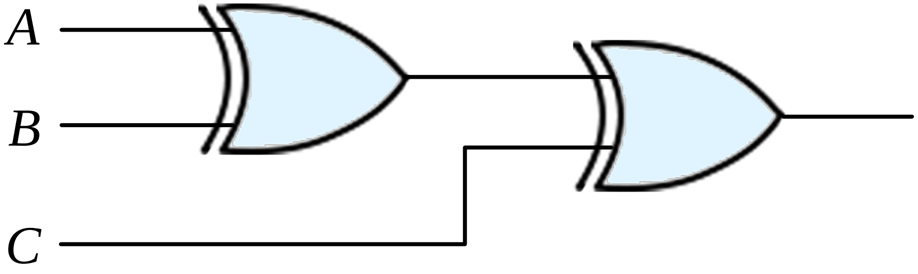
Even function F = (A⊕B⊕C⊕D)′

- An *n*-variable XNOR function is defined as the logical sum of the $2^n/2$ minterms whose binary numerical values have an even # of 1’s.

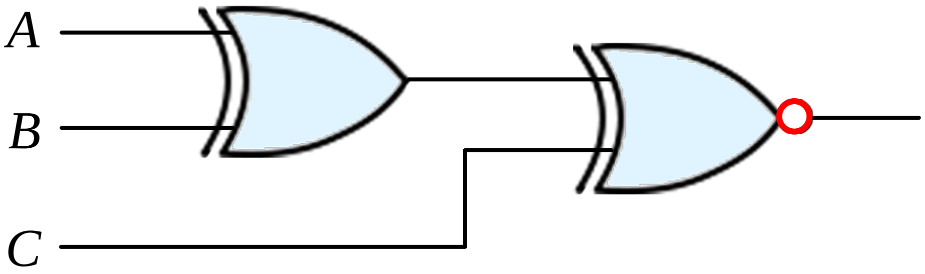
Logic Diagram of Odd & Even Functions

- Logic diagram of odd & even functions:

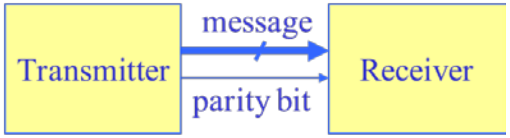
3-input
odd function



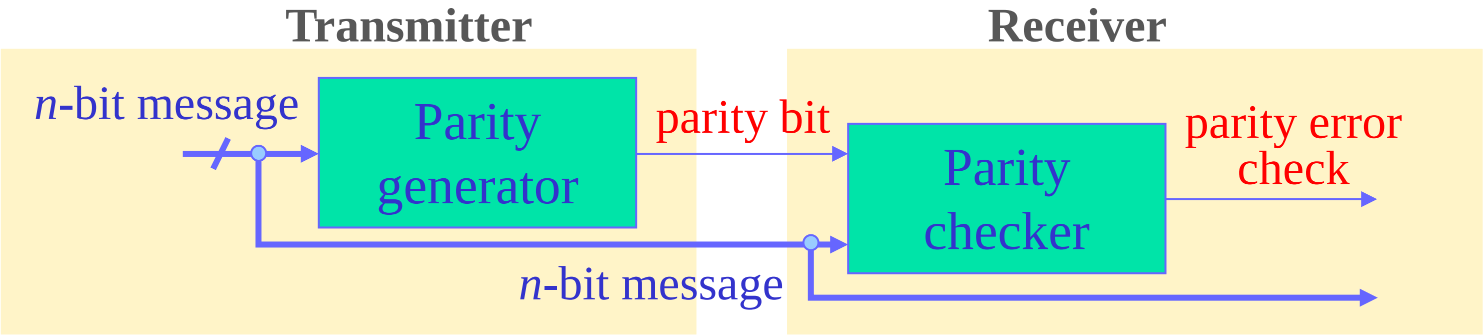
3-input
even function



Parity Generation & Checking

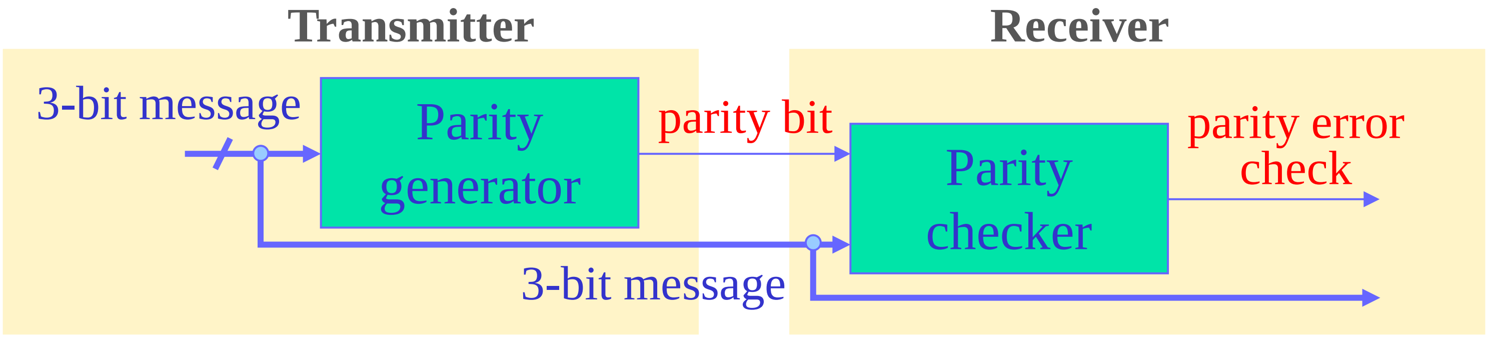


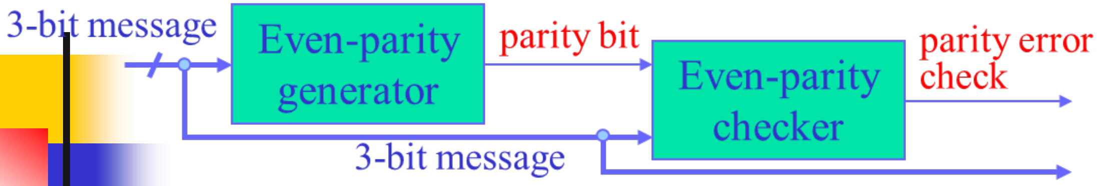
- Error-detection and correction codes: (§1-7, §7-4)
- Parity bit:
  - detecting errors during transmission of binary information.
  - is an extra bit included w/ a binary message to make the # of 1's either odd or even.
  - The message, including the parity bit, is transmitted and then checked at the receiving end for errors.
  - Parity generator: generates the parity bit in the transmitter
  - Parity checker: checks the parity in the receiver



Example

- Consider a 3-bit message to be transmitted together w/ an even parity bit.





Even-parity generator:

3-bit msg *x y z*	*Parity Bit* *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

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

Even-parity checker:

C = 1 if error occurs

4-bit received *x y z P*	*Error check* *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

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

Logic diagram

Even parity generator:

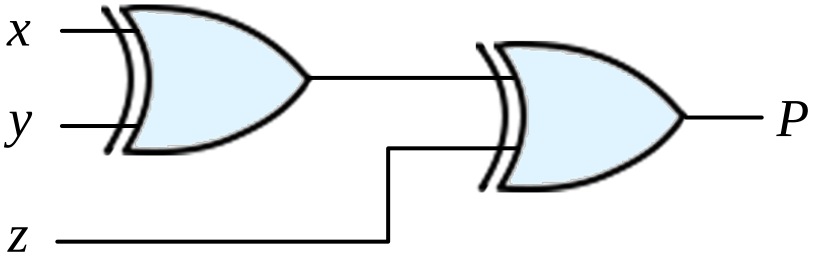
(3 inputs)

$P = x \oplus y \oplus z$

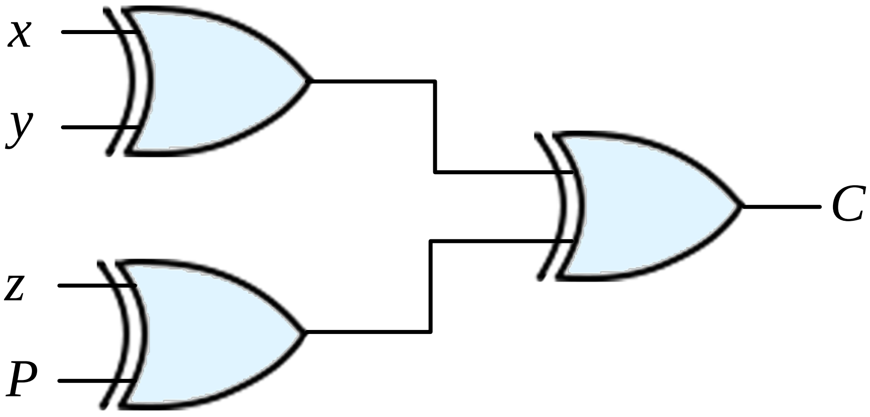
Even parity checker:

(4 inputs)

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



3-bit even parity generator



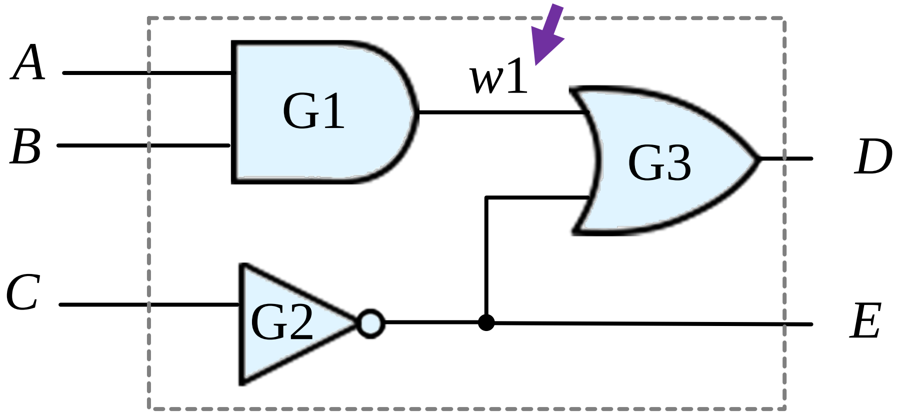
4-bit even parity checker

3-9 Hardware Description Language (HDL)

- HDL:
  - is a language that describes the hardware structure and behavior of digital systems in a textual form.
  - Can be used to represent logic diagrams, Boolean expressions, and other more complex digital ckts.
- Two applications of HDL processing:
  - Logic simulation:
    - the representation of the structure and behavior of a digital logic system through the use of a computer.
  - Logic synthesis:
    - the process of deriving a list of components and their interconnections (*netlist*) from the model of a digital system described in HDL.

Module Declaration

- HDL Example:



//Verilog model of circuit of Figure 3.37

```
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
```

Chapter Summary

- Minimization of Boolean function:
  - Algebraic manipulation: literal minimization (Ch2)
  - Map method: gate-level minimization (§3-2~3-5)
    - SoP simplification & PoS simplification
    - Don’t-care conditions
  - Tabular method: Quine-McCluskey method (補充資料)
- Multiple-Level Circuit Optimization (補充資料)
- NAND and NOR Implementation
- Other two-level implementation
- XOR and XNOR Functions
- Hardware Description Language (HDL)

Problems & Homework (6th ed)

Sections	Exercises	Homework
§3-2	3.1~3.3	3.3(a)
§3-3	3.4~3.12, 3.26	3.5(b), 3.6(b), 3.10(b)
§3-4	3.13, 3.14	3.13(b)
§3-5	3.15	3.15(b)
補充資料		Repeat 3-15(b) by Quine-McCluskey method
§3-6	3.16~3.23	3.16(a), 3.20
§3-7	3.24, 3.25	3.24
§3-8	3.27, 3.28, 3.30	3.28
Ch4	3.29	
HDL	3.31~3.40	