Lecture 4
Optimized Implementation of Logic Functions

吳文中

Karnaugh Map

- $f = m_0 + m_2 + m_4 + m_5 + m_6$
- $= \overline{x_1 x_2 x_3} + \overline{x_1} x_2 \overline{x_3} + x_1 \overline{x_2 x_3} + x_1 \overline{x_2} x_3 + x_1 x_2 \overline{x_3}$
- $= \overline{x_1 x_2 x_3} + \overline{x_1} x_2 \overline{x_3} + x_1 \overline{x_2 x_3} + x_1 \overline{x_2} x_3 + x_1 x_2 \overline{x_3} + x_1 \overline{x_2 x_3}$
- $= \overline{x_1}(\overline{x_2} + x_2)\overline{x_3} + x_1(\overline{x_2} + x_2)\overline{x_3} + x_1 \overline{x_2}(x_3 + \overline{x_3})$
- $= \overline{x_1 x_3} + x_1 \overline{x_3} + x_1 \overline{x_2}$
- $= \overline{x_3} + x_1 \overline{x_2}$

Row number	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Karnaugh Map

	x_1	x_2	x_3
m_0	0	0	0
m_2	0	1	0
m_4	1	0	0
m_6	1	1	0

$\overline{x_1 x_3}$

$x_1 \overline{x_3}$

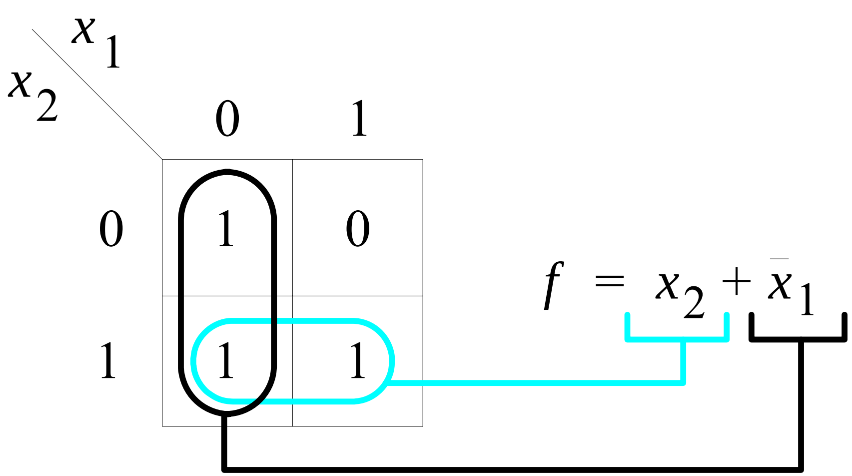
	x_1	x_2	x_3
m_4	1	0	0
m_5	1	0	1

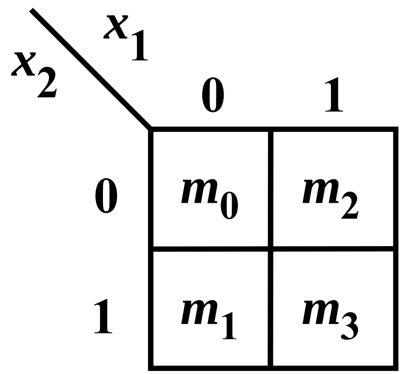
$x_1 \overline{x_2}$

Two-variable Karnaugh Map

- $m_2 + m_3 = x_1\overline{x_2} + x_1x_2$
- $= x_1(\overline{x_2} + x_2)$
- $= x_1$

x_1	x_2	
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3





Three-Variable Karnaugh Map

x_1	x_2	x_3	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

x_3 \ $x_1 x_2$	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

Three-Variable Example

$x_3 \backslash x_1x_2$	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$f = \underbrace{x_1\bar{x}_3} + \underbrace{\bar{x}_2 x_3}$$

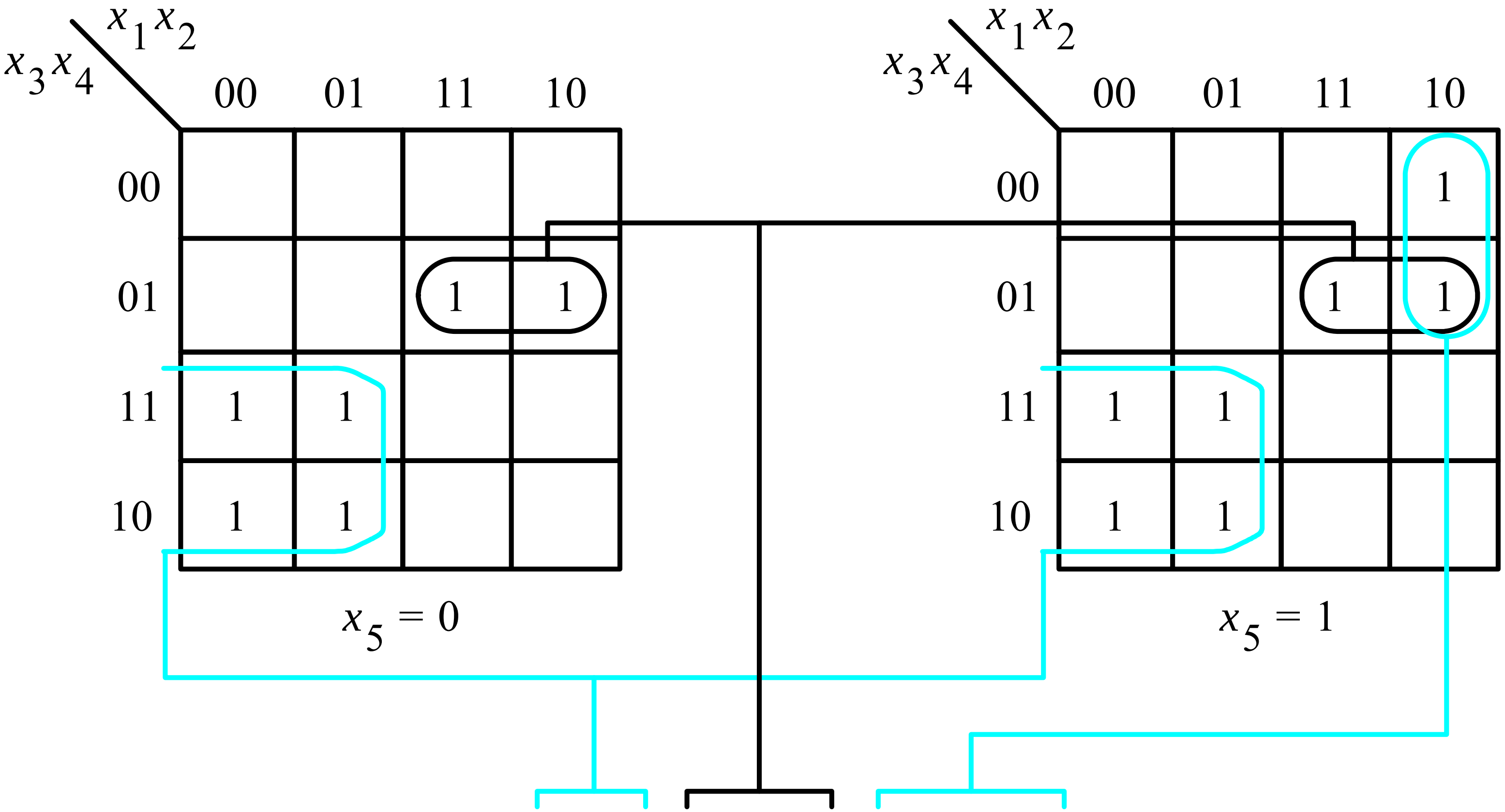
$x_3 \backslash x_1x_2$	00	01	11	10
0	1	1	1	1
1	0	0	0	1

$$f = \underbrace{\bar{x}_3} + \underbrace{x_1\bar{x}_2}$$

Four Variable Karnaugh Map

x_3x_4 \ x_1x_2	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

Five-Variable Map



$$f_1 = \bar{x}_1x_3 + x_1\bar{x}_3x_4 + x_1\bar{x}_2\bar{x}_3x_5$$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	0	0	1
10	1	0	0	1

$$f_1 = \bar{x}_2 x_3 + x_1 \bar{x}_3 x_4$$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	1	1	1	1

$$f_2 = x_3 + x_1 x_4$$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	1	1	0
10	1	1	0	1

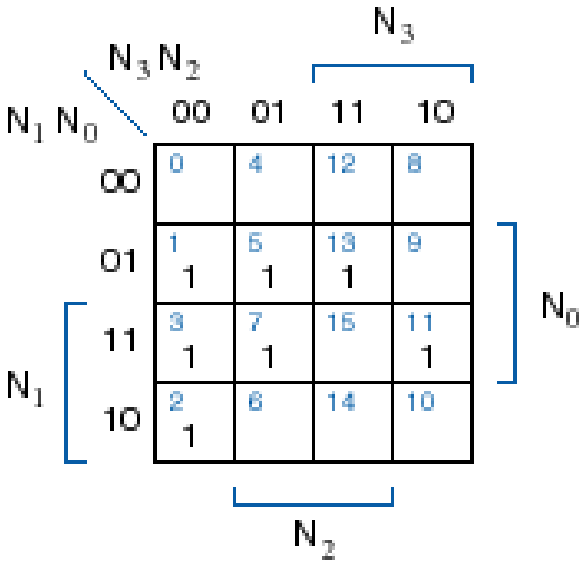
$$f_3 = \bar{x}_2 \bar{x}_4 + \bar{x}_1 x_3 + x_2 x_3 x_4$$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	0	0	1	1
10	0	0	1	1

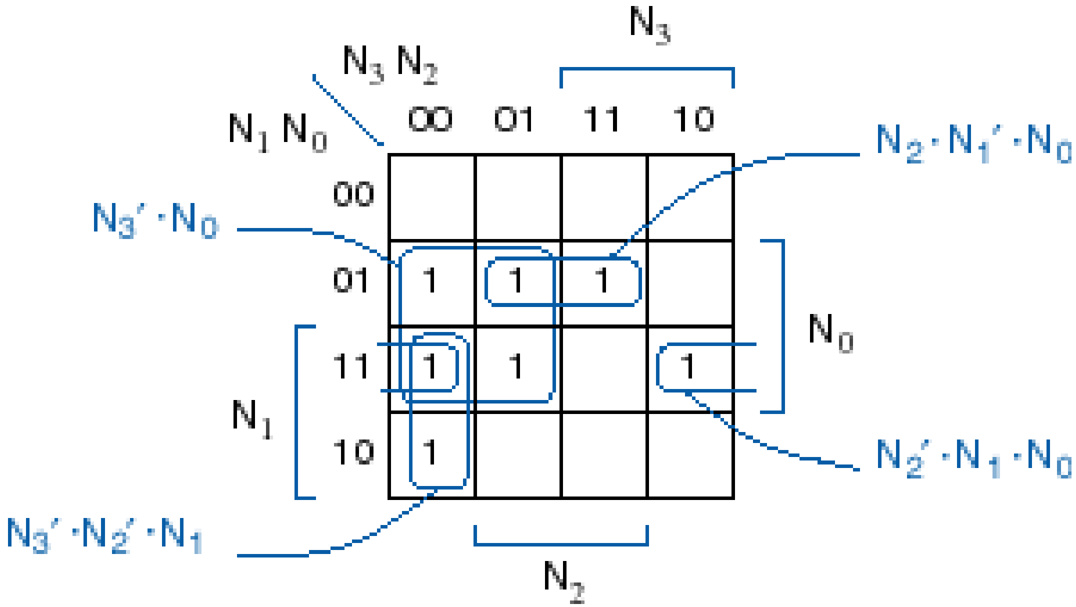
$$f_4 = \bar{x}_1 \bar{x}_3 + x_1 x_3 + \begin{matrix} x_1 x_2 \\ \text{or} \\ x_2 \bar{x}_3 \end{matrix}$$

Examples of Four-Variable Map

Prime-Number Detector (again)



$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11,13)$



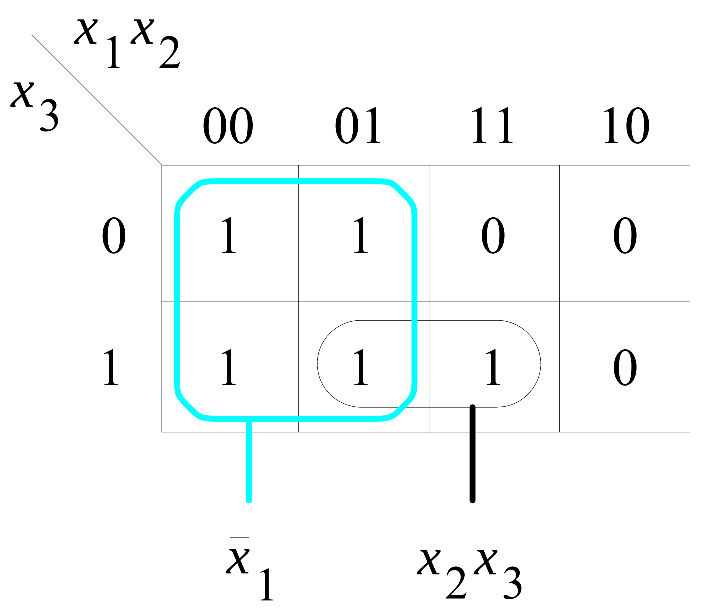
$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$

Terminology

- Literal : A given product term consists of some number of variables, each of which may appear either in uncomplemented or complemented form. e.g. $x_1\overline{x_2}x_3$ has three literals
- Implicant: A product term that indicates the input valuations(s) for which is given function is equal to 1 is called an implicant of the function. The most basic implicants are the minterms, and the terms combining pairs of minterms with consensus theorem are also implicants.

Terminology

- Prime Implicant: An implicant is called a prime implicant if it cannot be combined into another implicant that has fewer literals.



Terminology

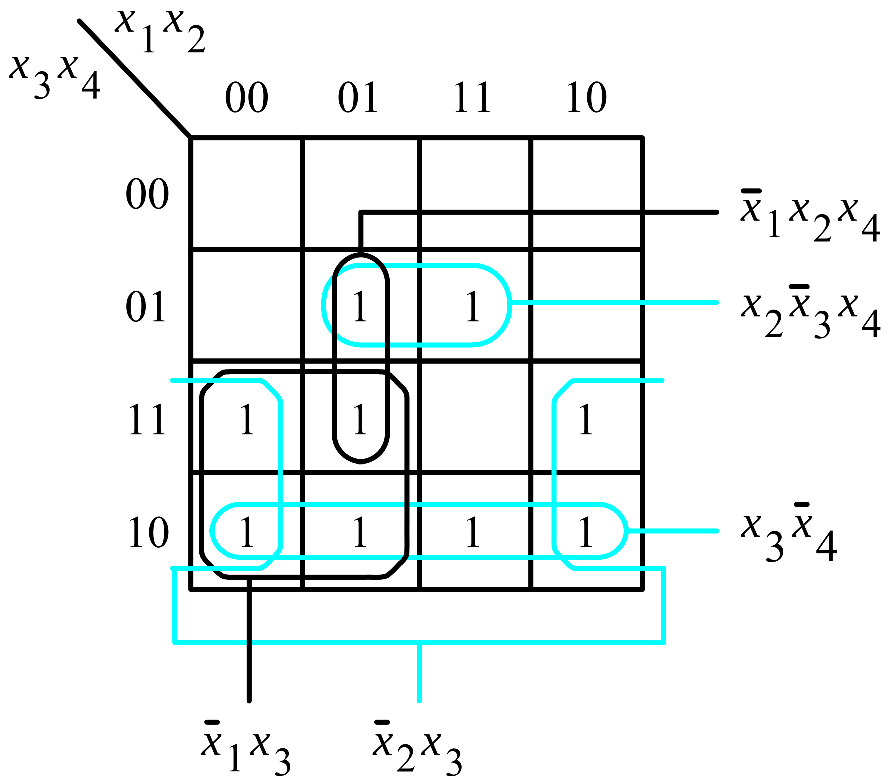
- Cover: A collection of implicants that account for which a given function is equal to 1 is called a cover of that function, e.g.
- $f = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}x_2\overline{x_3} + x_1\overline{x_2}\,\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3}$
- $= \overline{x_1}\,\overline{x_3} + x_1\overline{x_3} + x_1\overline{x_2}$
- $= \overline{x_3} + x_1\overline{x_2}$

Terminology

- Cost: The cost of a logic circuit is the number of gates plus the total number of inputs to all gates in the circuit.
- $f = x_1\overline{x_2} + x_3\overline{x_4}$ has a cost of 9 because it can implanted using two AND gates and one OR gate with six inputs to the AND and OR gates.
- If an inversion is needed inside a circuit, then the corresponding NOT gate and its input are included in the cost. Ex. $\boldsymbol{g} = \overline{\boldsymbol{x_1}\overline{\boldsymbol{x_2}} + \boldsymbol{x_3}}\,(\overline{\boldsymbol{x_4}} + \boldsymbol{x_5})$ has cost of 14.

Essential Prime Implicants

- If a prime implicant includes a minterm for which *f*=1 that is not included in any other prime implicant, then it must be included in the cover and is called an essential prime implicants.

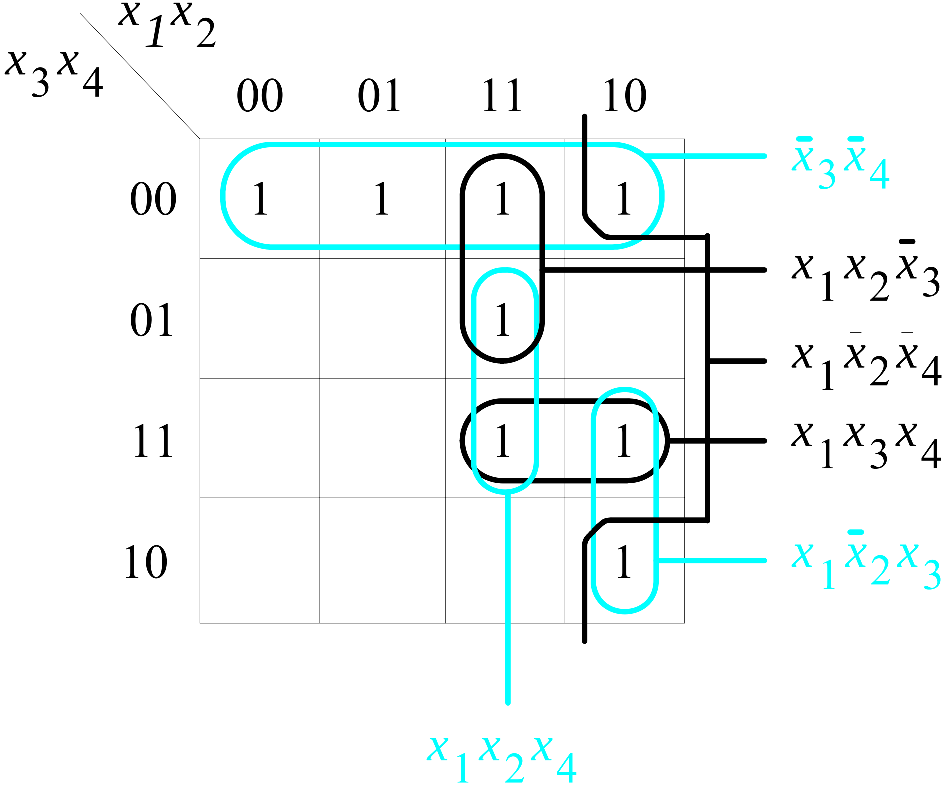


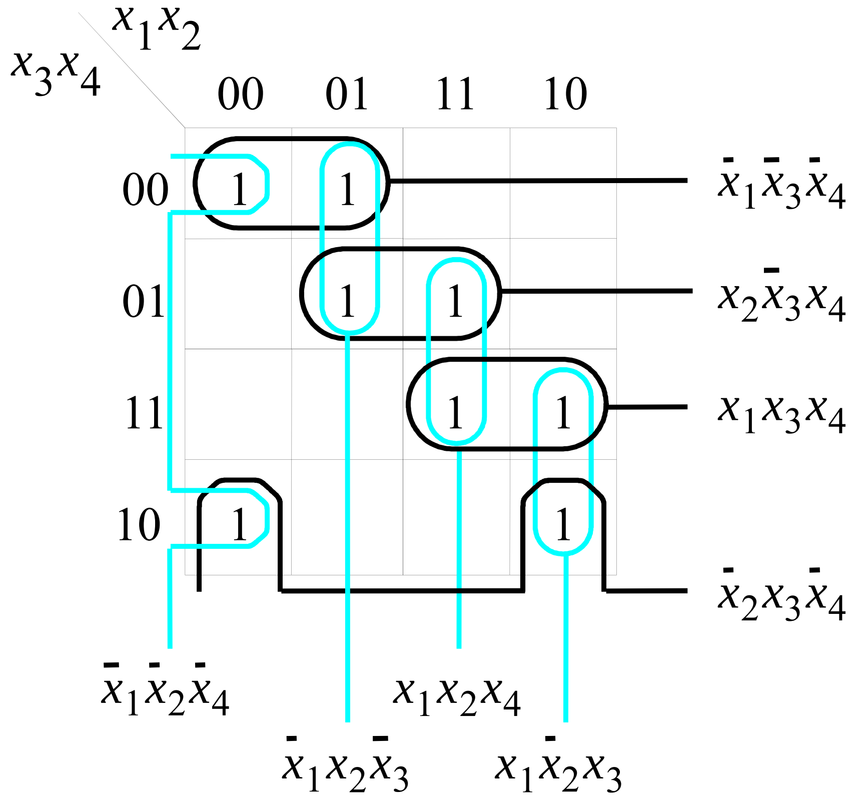
NTU ESOE

Minimized Procedure for Minimum Cost

- 1. Generate all prime implicants for the given function *f*.
- 2. Find the set of essential prime implicants.
- 3 If the set of essential prime implicants covers all valuations for which *f*=1, then this set is the desired cover of *f.* Otherwise, determine the nonessential prime implicants that should be added to form a complete minimum-cost cover.

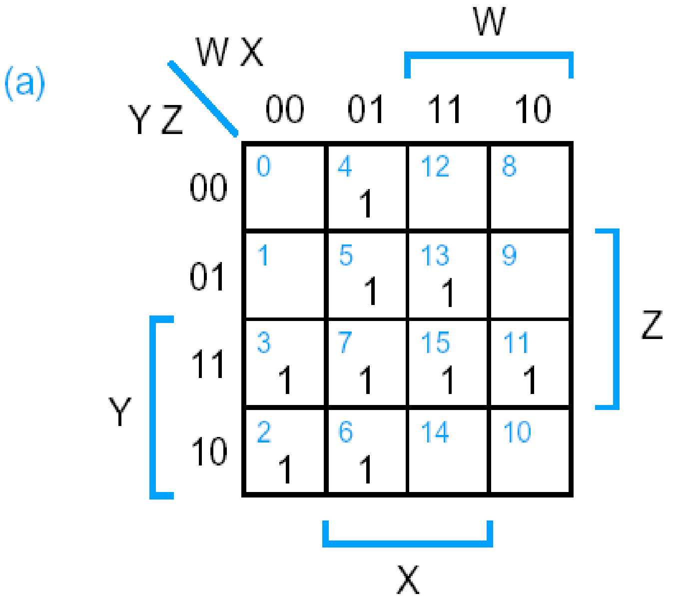
Examples



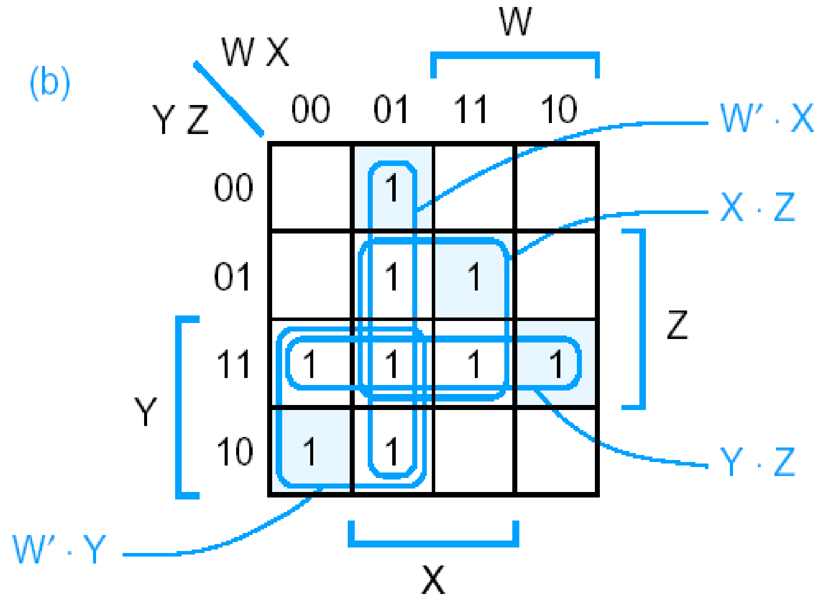


NTU ESOE

All Prime-Implicants are Essential

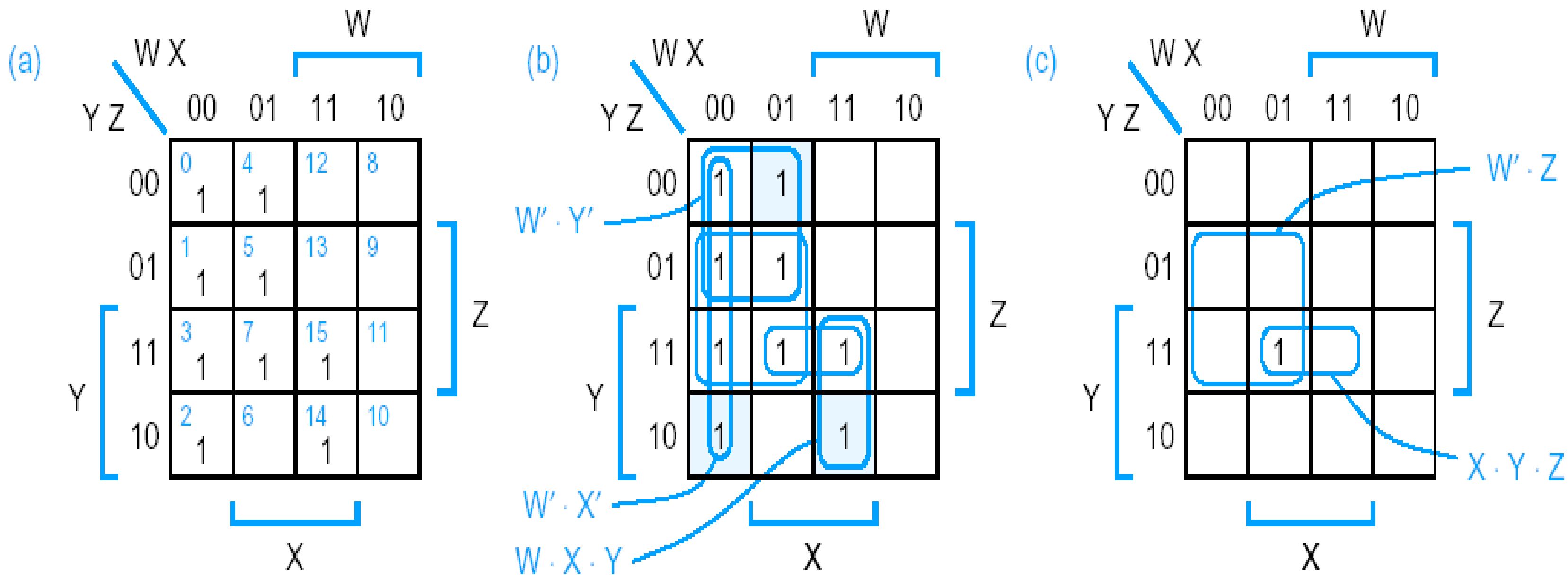


$F = \Sigma_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15)$



F = W′ · Y + W′ · X + X · Z + Y · Z

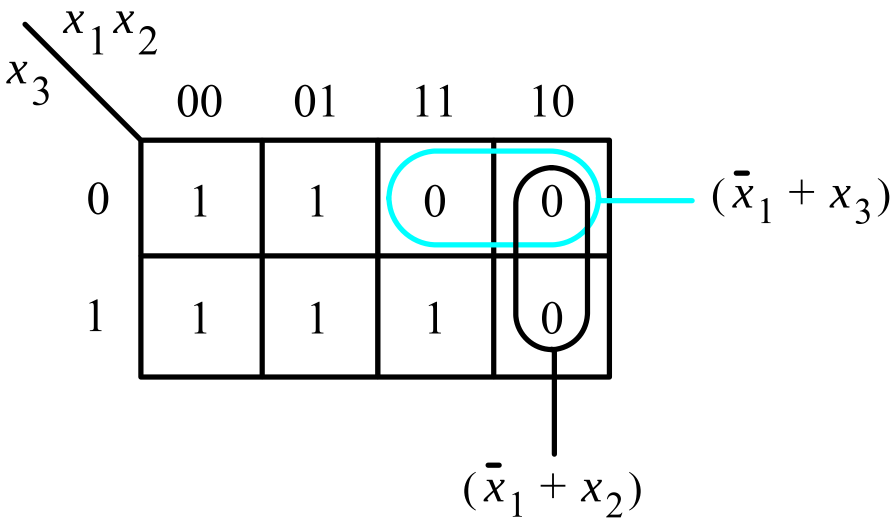
Another Example

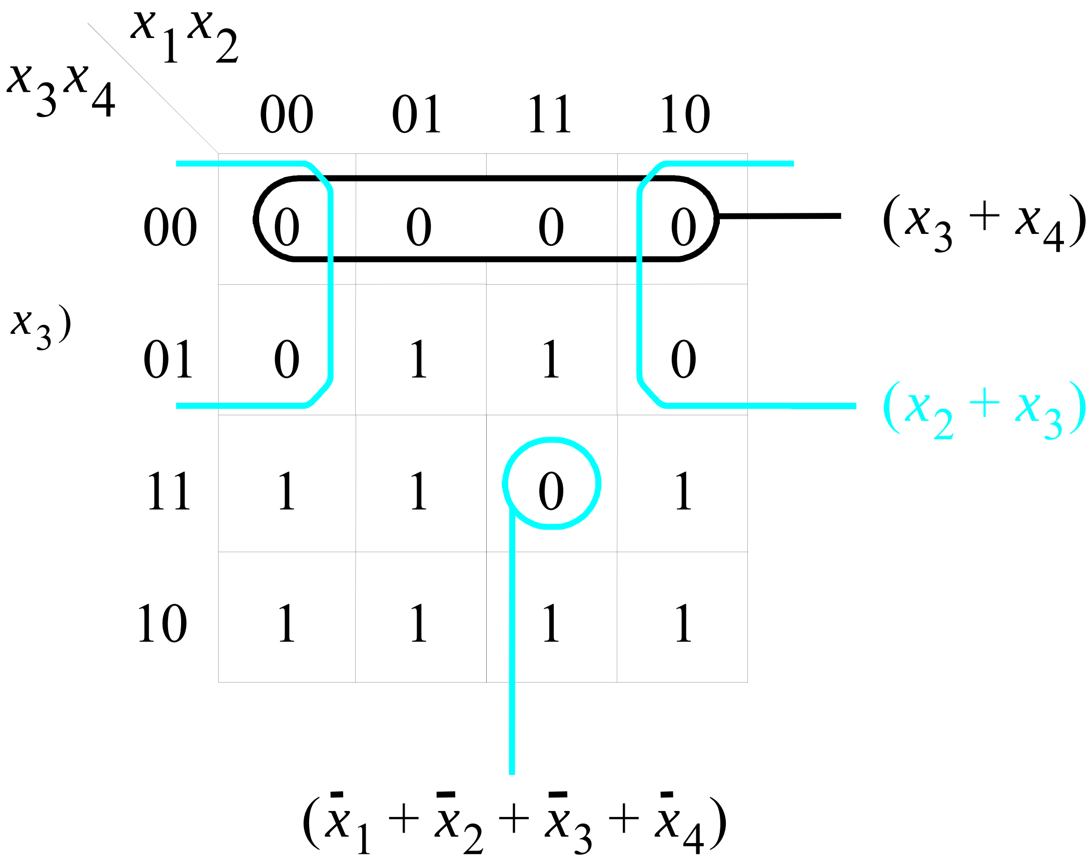


$F = \Sigma_{W,X,Y,Z}(0,1,2,3,4,5,7,14,15)$

$F = W' \cdot Y' + W' \cdot X' + W \cdot X \cdot Y + W' \cdot Z$

Minimization of Product-of-Sums

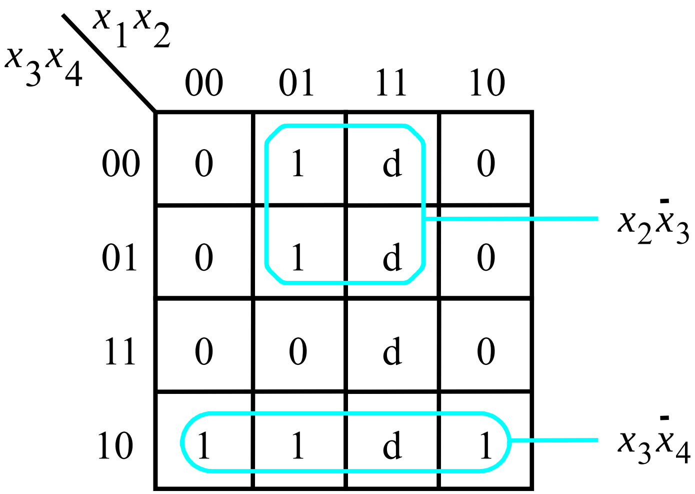




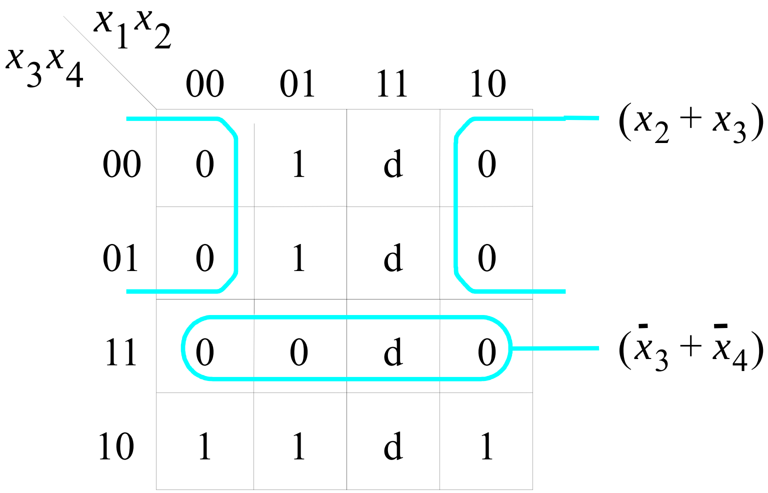
NTU ESOE

Incomplete Specified Functions

- $f(x_1, \ldots, x_4) = \sum m(2,4,5,6,10) + D(12,13,14,15)$

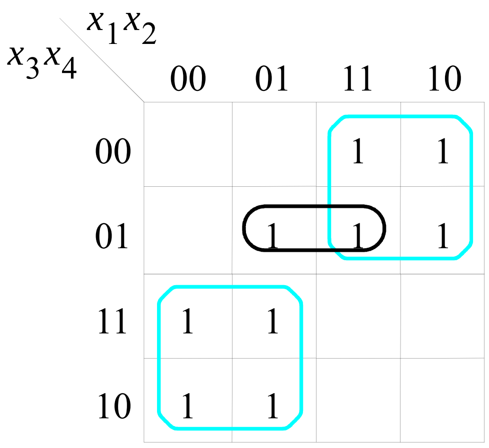


(a) SOP implementation

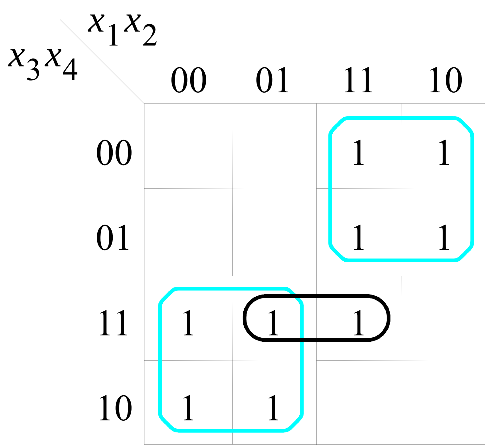


(b) POS implementation

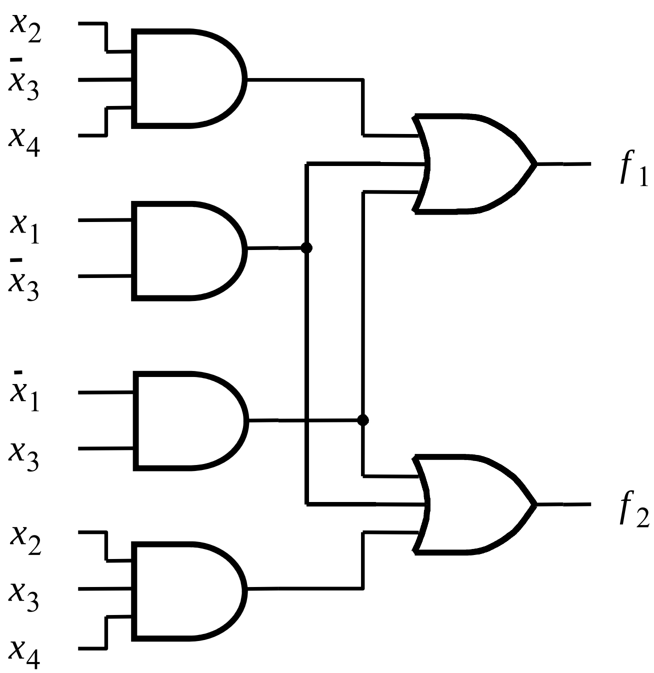
Example 4.1 Gate Sharing



(a) Function f_1

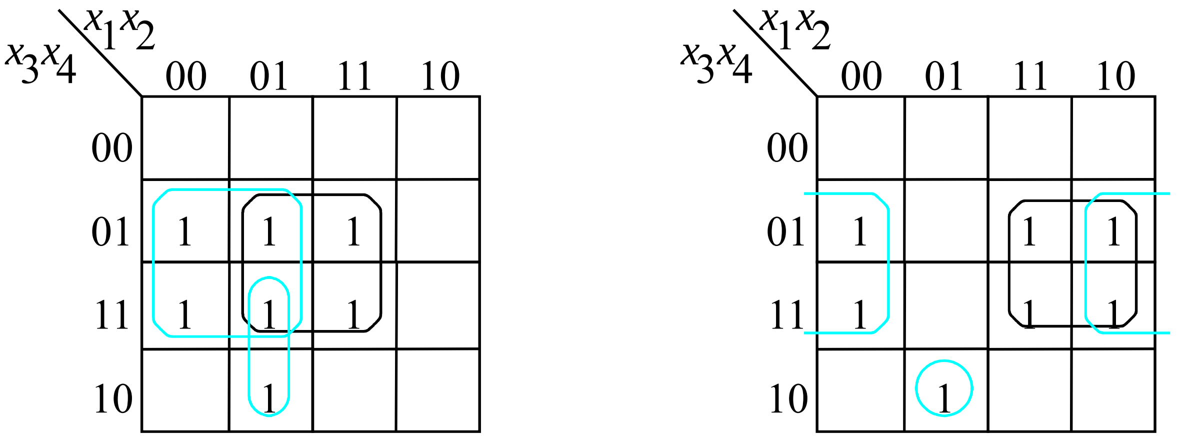


(b) Function f_2



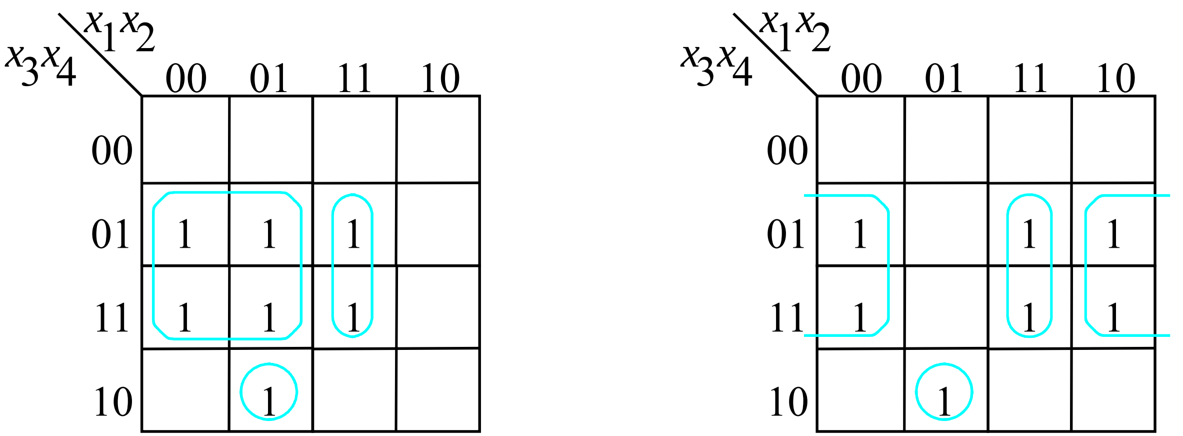
(c) Combined circuit for f_1 and f_2

Example 4.4 Minimized Gate Sharing SOP



(a) Optimal realization of f_3

(b) Optimal realization of f_4



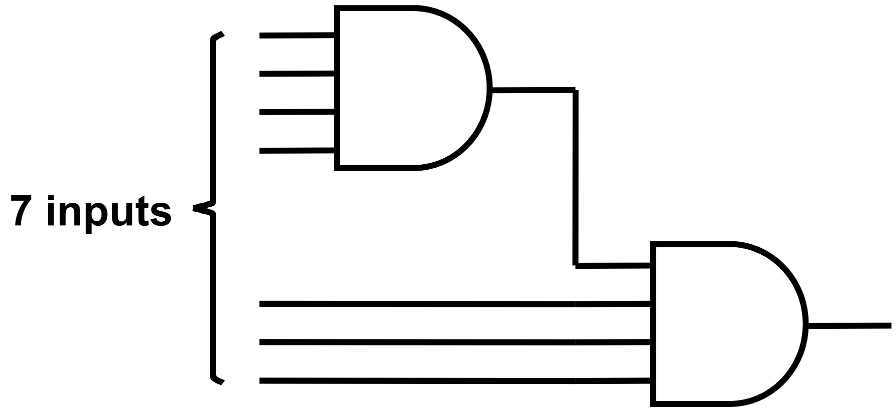
(c) Optimal realization of f_3 and f_4 together

Multilevel Synthesis

- A two-level realization is usually efficient for functions of a few variables.
- As the number of inputs increases a two-level circuit may result in fan-in problems (no enough inputs, variables for specific implementations)
- To solve the fain-in problem, *f* must be expressed in a form that has more than two levels of logic operations.

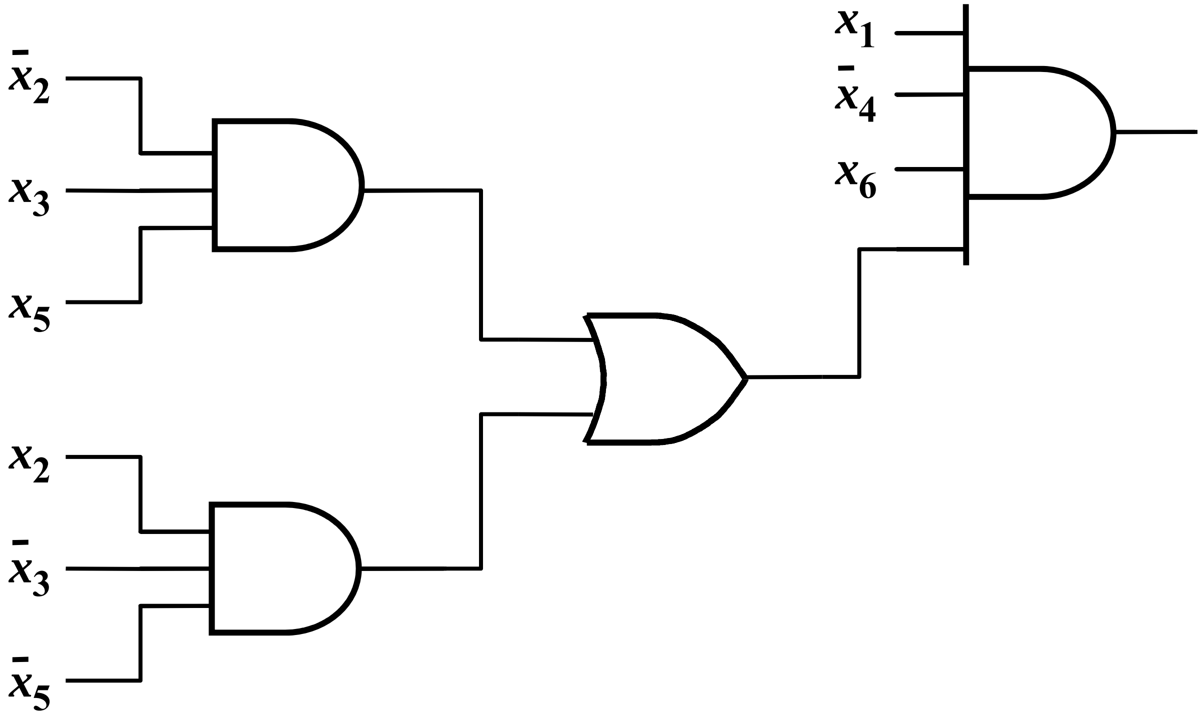
Factoring

- The distributive law allows
$$f = x_1\overline{x_6}(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5)$$
$$= (x_1\overline{x_6} + x_2x_7)(x_3 + x_4x_5)$$
- Fan-in problem: e.g. a 7-input product term are re-factoring to 2 four-input AND gates.



A Factored Circuit

- $f = x_1\overline{x_2}x_3\overline{x_4}x_5x_6 + x_1x_2\overline{x_3x_4x_5}x_6$ f
  $= x_1\overline{x_4}x_6(\overline{x_2}x_3x_5 + x_2\overline{x_3x_5})$



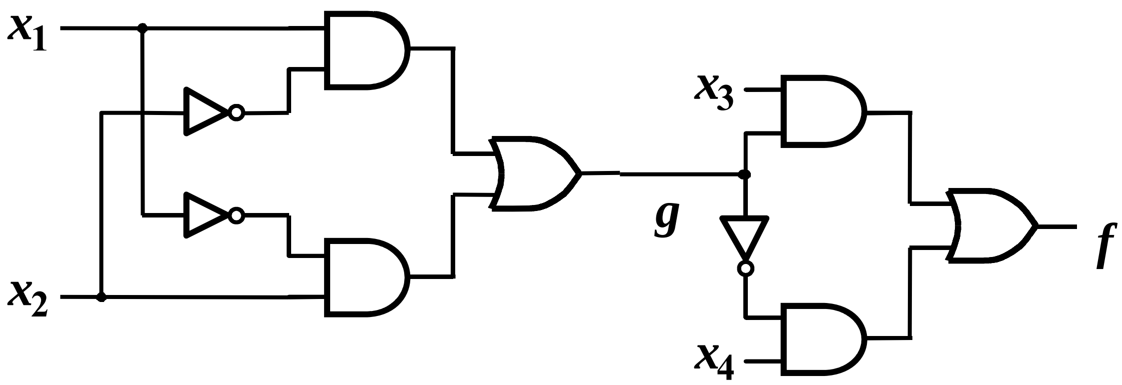
Example 4.6 Function Decomposition

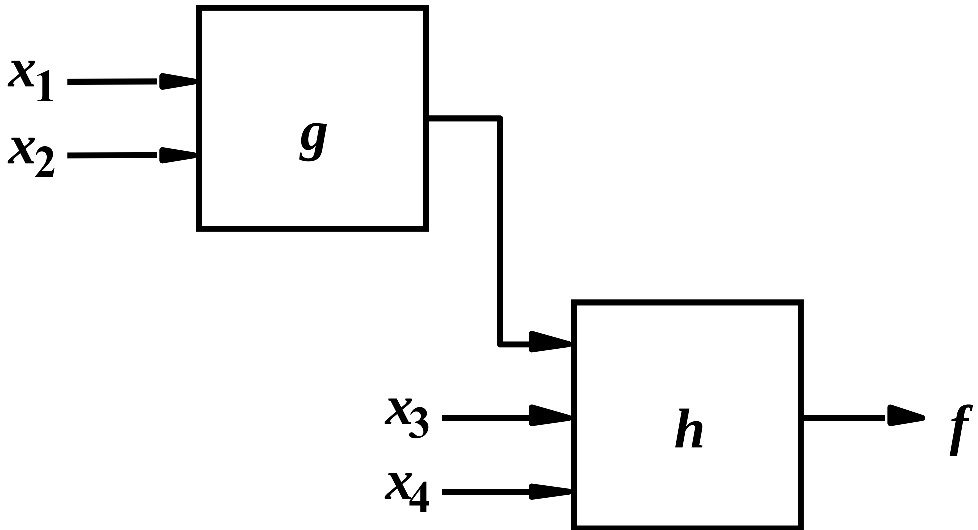
- $f = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2x_4 + \overline{x_1}\overline{x_2}x_4$
  $= (\overline{x_1}x_2 + x_1\overline{x_2})x_3 + (x_1x_2 + \overline{x_1}\overline{x_2})x_4$
- Let $g(x_1, x_2) = \overline{x_1}x_2 + x_1\overline{x_2}$

$$
\begin{aligned}
\bar{g} &= \overline{\overline{x_1}x_2 + x_1\overline{x_2}} \\
&= \overline{\overline{x_1}x_2} \cdot \overline{x_1\overline{x_2}} \\
&= (x_1 + \overline{x_2})(\overline{x_1} + x_2) \\
&= \cancel{x_1\overline{x_1}} + x_1x_2 + \overline{x_1}\overline{x_2} + \cancel{x_2\overline{x_2}} \\
&= x_1x_2 + \overline{x_1}\overline{x_2}
\end{aligned}
$$

- $f = gx_3 + \bar{g}x_4 = h[g(x_1, x_2), x_3, x_4]$

Example 4.6 Function Decomposition





Example 4.7

x_3x_4 \ x_1x_2	00	01	11	10
00	1			
01		1	1	1
11	1			
10		1	1	1

$x_5 = 0$

x_3x_4 \ x_1x_2	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

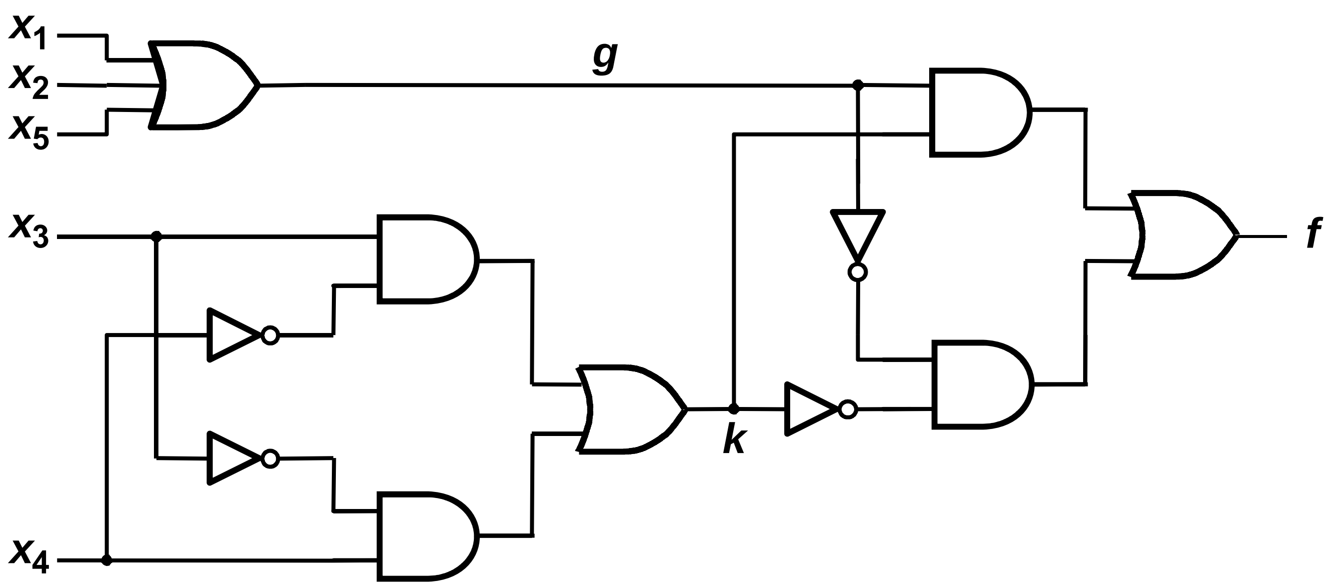
$x_5 = 1$

- The patterns, "rows" defines $k = x_3'x_4 + x_3x_4'$
- Within the patterns (blue stripes), "columns" of 1s defines $g = x_1 + x_2 + x_5$ (columns 2,3,4 in x_5=0 plane and all columns in x_5= 1 plane)
- From Ex 4.6 $k' = x_3'x_4' + x_3x_4$

Example 4.7

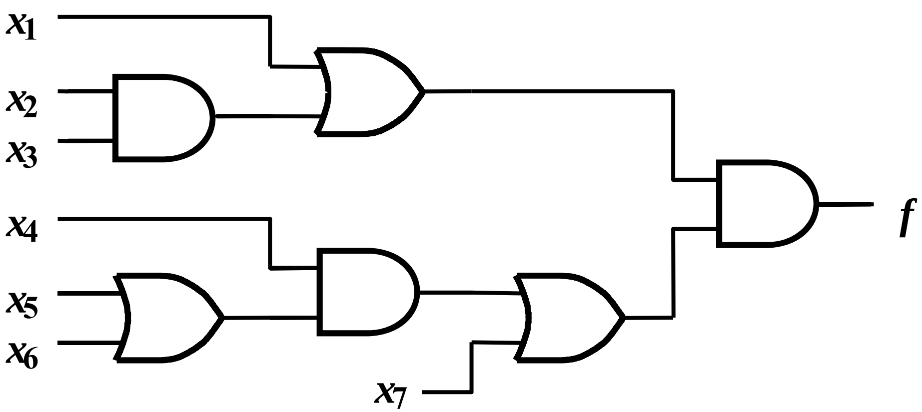
- Thus decomposition of

$f =$ kg (rows 2,4 & columns 2,3,4 in x_5=0 plane and all columns in x_5= 1 plane)
+ $k'g'$ (rows 1,3 & columns 1 in x_5=0 plane)

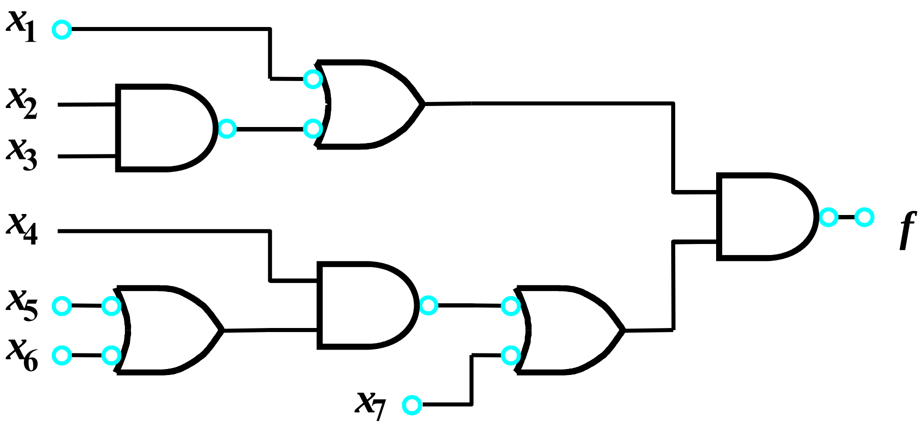


NTU ESOE

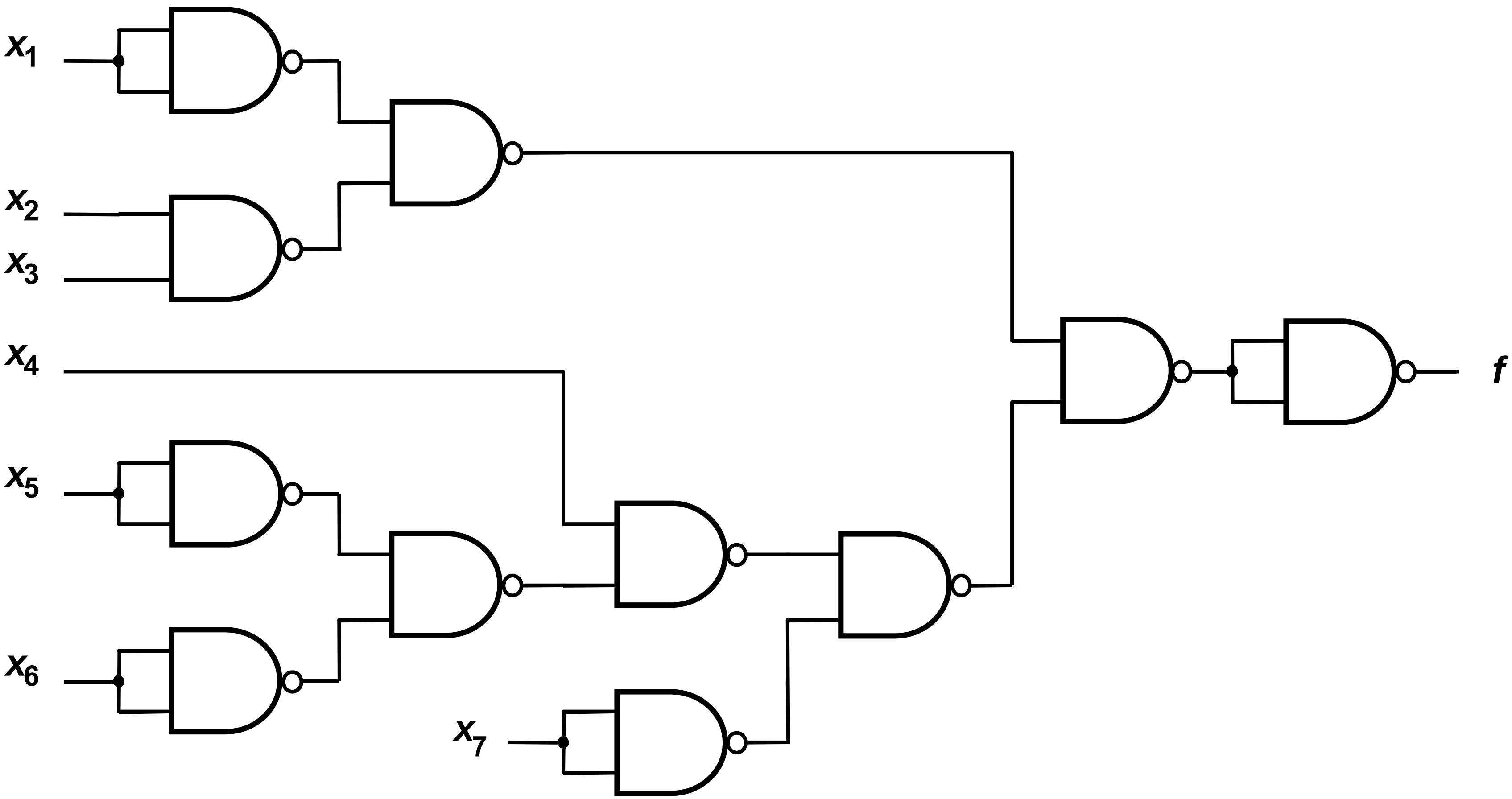
Conversion of NAND Circuit



(a) Circuit with AND and OR gates

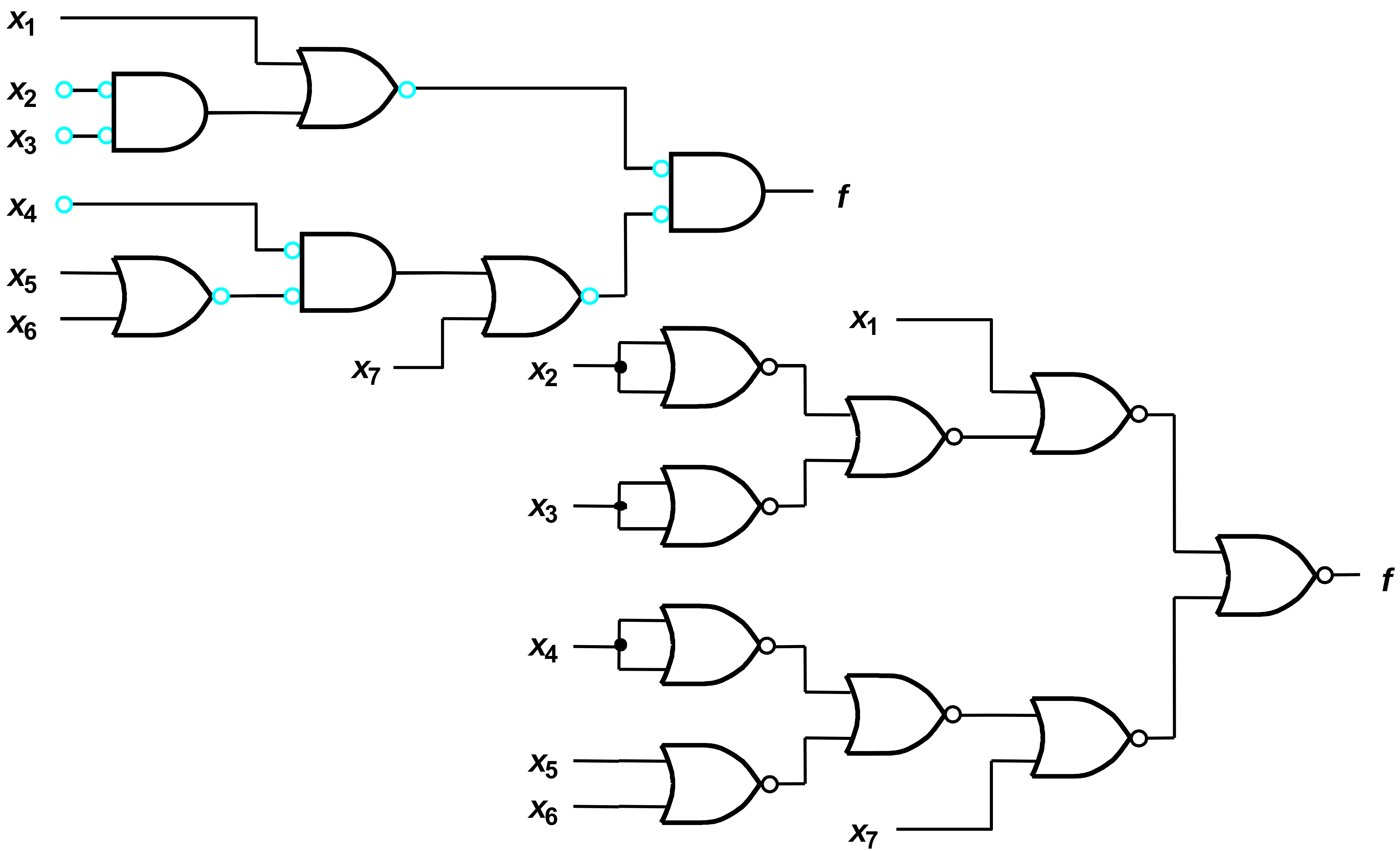


(b) Inversions needed to convert to NANDs



x_1
x_2
x_3
x_4
x_5
x_6
x_7
f

NOR Conversion



Example 4.10 Multilevel Circuits

- $P_1 = x_2x_3$; $P_2 = x_5 + x_6$; $P_3 = x_1 + P_1$
- $P_4 = x_4P_2 = x_4(x_5 + x_6)$
- $P_5 = P_4 + x_7 = x_4(x_5 + x_6) + x_7$
- $f = P_3P_5 = x_1x_4x_5 + x_1x_4x_6 + x_1x_7 + x_2x_3x_4x_5 + x_2x_3x_4x_6 + x_2x_3x_7$ (cost=6A+1O+25i)