COMP3222/9222 Digital Circuits & Systems
2. Optimized Implementation of Logic Functions

Lecture 2 Objectives

To learn more about:

- Synthesis & analysis of logic functions

- Graphical representation of logic functions in the form of Karnaugh maps

- Techniques for deriving minimum-cost implementations of logic functions

- Use of CAD tools and VHDL to implement logic functions

Consider simplifying $f(x1, x2, x3) = \Sigma\ m(0, 2, 4, 5, 6)$

- What is a minimum cost implementation?
- How is it obtained?

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

$f = \overline{x_1}\overline{x_2}\overline{x_3} + \overline{x_1}x_2\overline{x_3} + x_1\overline{x_2}\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3}$

$= \overline{x_1}\overline{x_3} + x_1\overline{x_3} + x_1\overline{x_2}$

$= \overline{x_3} + x_1\overline{x_2}$

The VHDL code for the function in L02/S3

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

ENTITY func1 IS
    PORT (x1, x2, x3   : IN   BIT ;
            f   : OUT BIT) ;
END func1 ;

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
    f <= (NOT x1 AND NOT x2 AND NOT x3) OR
        (NOT x1 AND   x2 AND NOT x3) OR
        (x1 AND NOT x2 AND NOT x3) OR
        (x1 AND NOT x2 AND   x3) OR
        (x1 AND   x2 AND NOT x3) ;
END LogicFunc ;

Necessarily synthesizes to
$f = \overline{x_3} + x_1\overline{x_2}$

Karnaugh map

- The key to finding a minimum-cost expression for a given logic function is to reduce the number of product (or sum) terms needed in the expression by applying the *combining/uniting property*

$$x \cdot y + x \cdot y' = x \qquad \text{and} \qquad (x + y) \cdot (x + y') = x$$

as judiciously as possible.

- The Karnaugh map approach provides a systematic way of performing this optimization

Layout of two-variable minterms in a two-variable Karnaugh map

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

(a) Truth table

x_1 (more significant variable)

x_2

(less significant variable)

	0	1
0	m_0 00	m_2 10
1	m_1 01	m_3 11

(b) Karnaugh map

Note that the addresses of horizontally and vertically adjacent cells in the Karnaugh map differ in a single bit

The function of L01/S25

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

$\overline{x}_1\overline{x}_2 + \overline{x}_1 x_2$

$\overline{x}_1 x_2 + x_1 x_2$

$f = \overline{x}_1 + x_2$
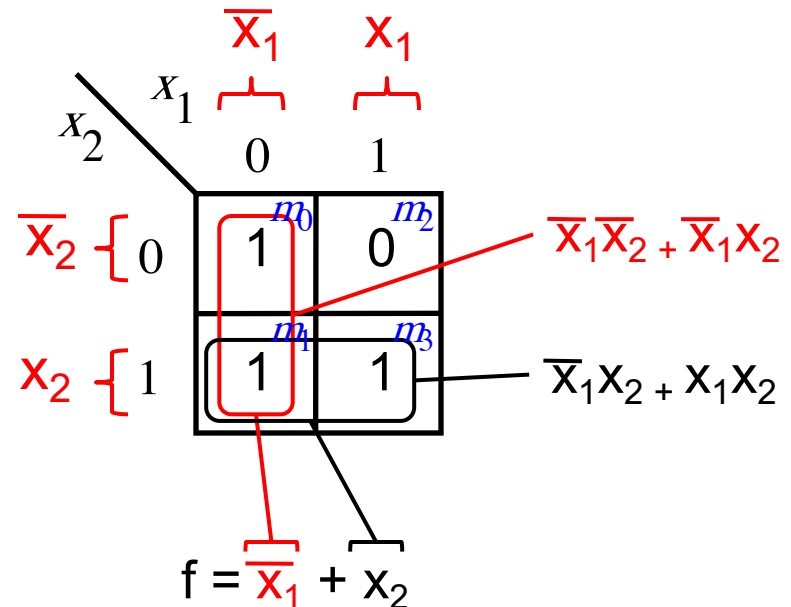
Note that adjacent cells in a column or row of the Karnaugh map can be combined in order to eliminate the variable that is present in both uncomplemented and complemented form

Karnaugh map layout of three-variable minterms

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

(a) Truth table

more significant variables

x_1

x_2

$x_1 x_2$

less significant variables

x_3

$x_1 x_2$	00	01	11	10
0	m_0 000	m_2 010	m_6 110	m_4 100
x_3 1	m_1 001	m_3 011	m_7 111	m_5 101

(b) Karnaugh map

Note: columns (rows) of the table are labeled using a Gray encoding – successive entries differ in just one bit.

Examples of three-variable Karnaugh maps

x_1	x_2	x_3	(a)	(b)
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

$x_1 x_2$ / x_3

	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$f = x_1 \bar{x}_3 + \bar{x}_2 x_3$
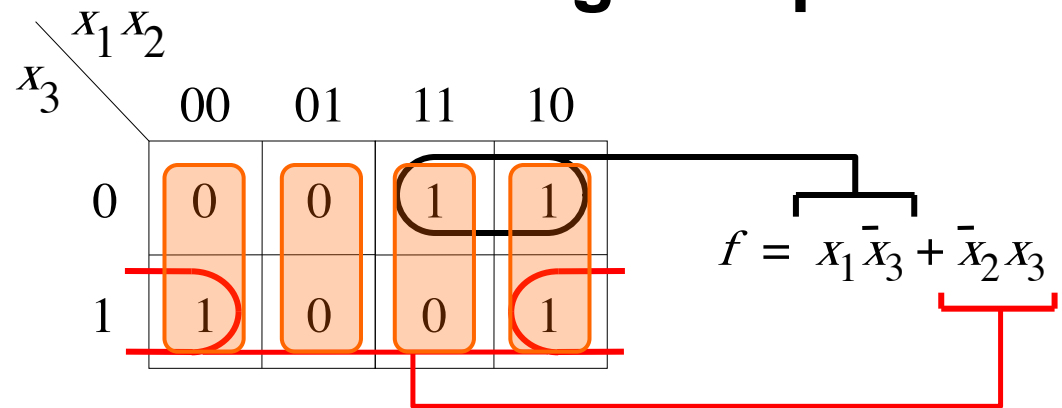
(a) The function of L01/S35

Note that the map wraps around to form a torus – adjacency can span the leftmost and rightmost columns

$x_1 x_2$ / x_3

	00	01	11	10
0	1	1	1	1
1	0	0	0	1

$f = \bar{x}_3 + x_1 \bar{x}_2$
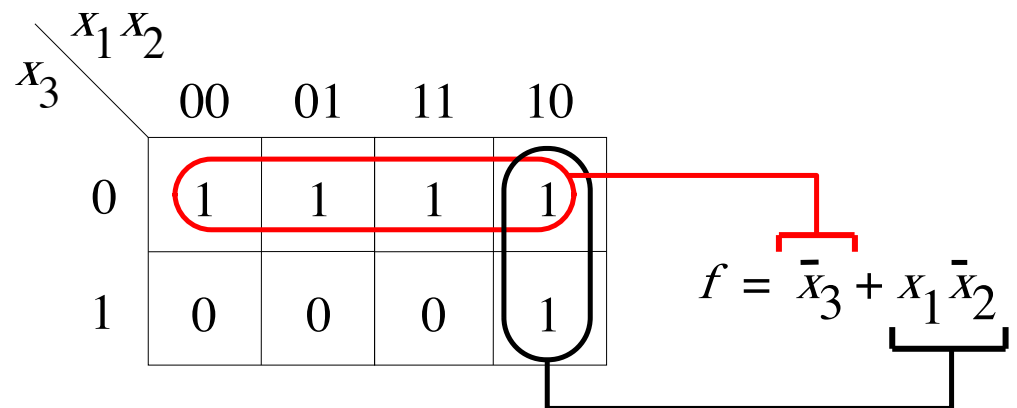
(b) The function of L02/S3

A four-variable Karnaugh map

- Note that opposite edges are considered adjacent
- The map forms a "torus" or "donut-shape"

Examples of four-variable Karnaugh maps

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

$$f_2 = x_3 + x_1 x_4$$

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

$$f_4 = \bar{x}_1 \bar{x}_3 + x_1 x_3 + \begin{array}{c} x_1 x_2 \\ \text{or} \\ x_2 \bar{x}_3 \end{array}$$
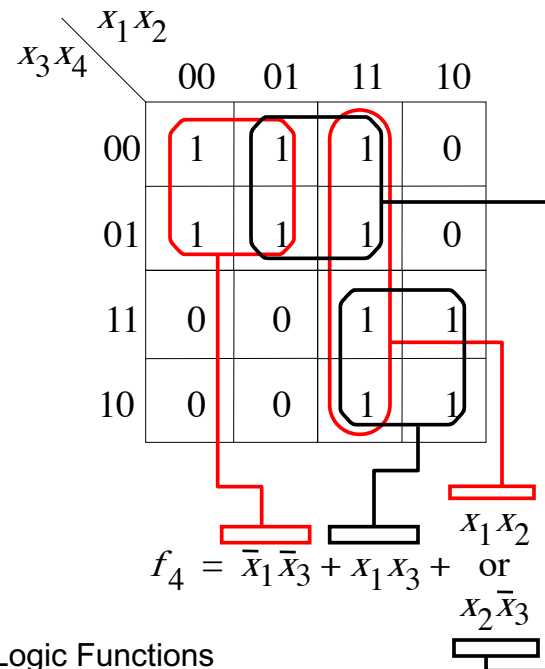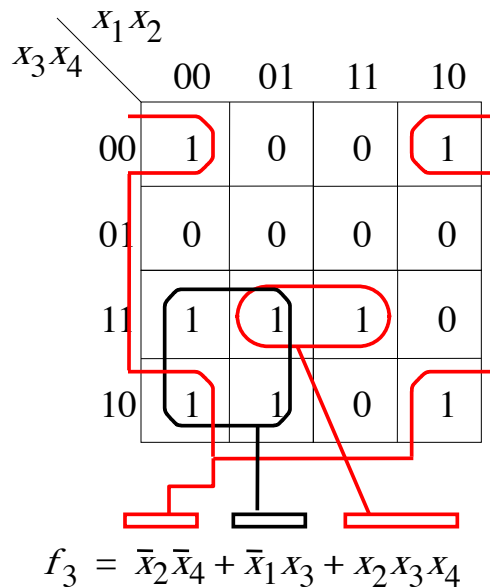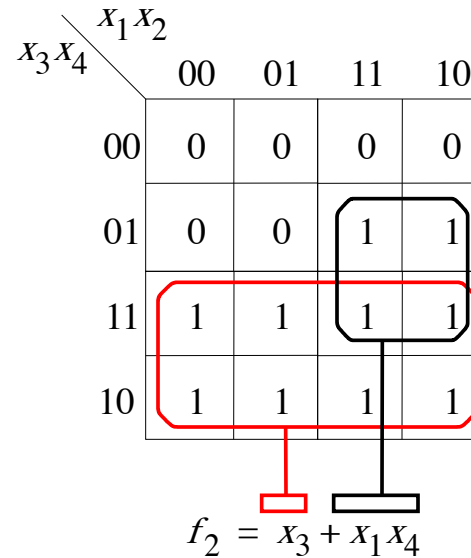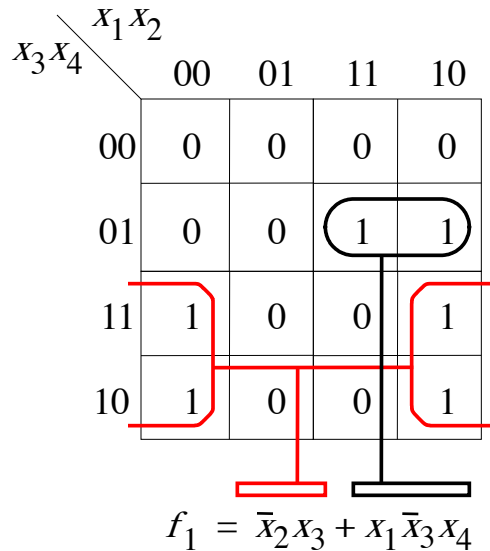
The VHDL code for f_1 in L02/S11

ENTITY func2 IS
    PORT (x1, x2, x3, x4   : IN   BIT ;
               f   : OUT BIT) ;
END func2 ;

ARCHITECTURE LogicFunc OF func2 IS
BEGIN
    f <= (NOT x1 AND NOT x2 AND x3 AND NOT x4) OR
          (NOT x1 AND NOT x2 AND x3 AND x4) OR
          (x1 AND NOT x2 AND NOT x3 AND x4) OR
          (x1 AND NOT x2 AND x3 AND NOT x4) OR
          (x1 AND NOT x2 AND x3 AND x4) OR
          (x1 AND x2 AND NOT x3 AND x4) ;
END LogicFunc ;

Synthesizes as
$f = \overline{x_2}x_3 + x_1\overline{x_3}x_4$

A five-variable Karnaugh map

$$f_1 = \bar{x}_1 x_3 + x_1 \bar{x}_3 x_4 + x_1 \bar{x}_2 \bar{x}_3 x_5$$
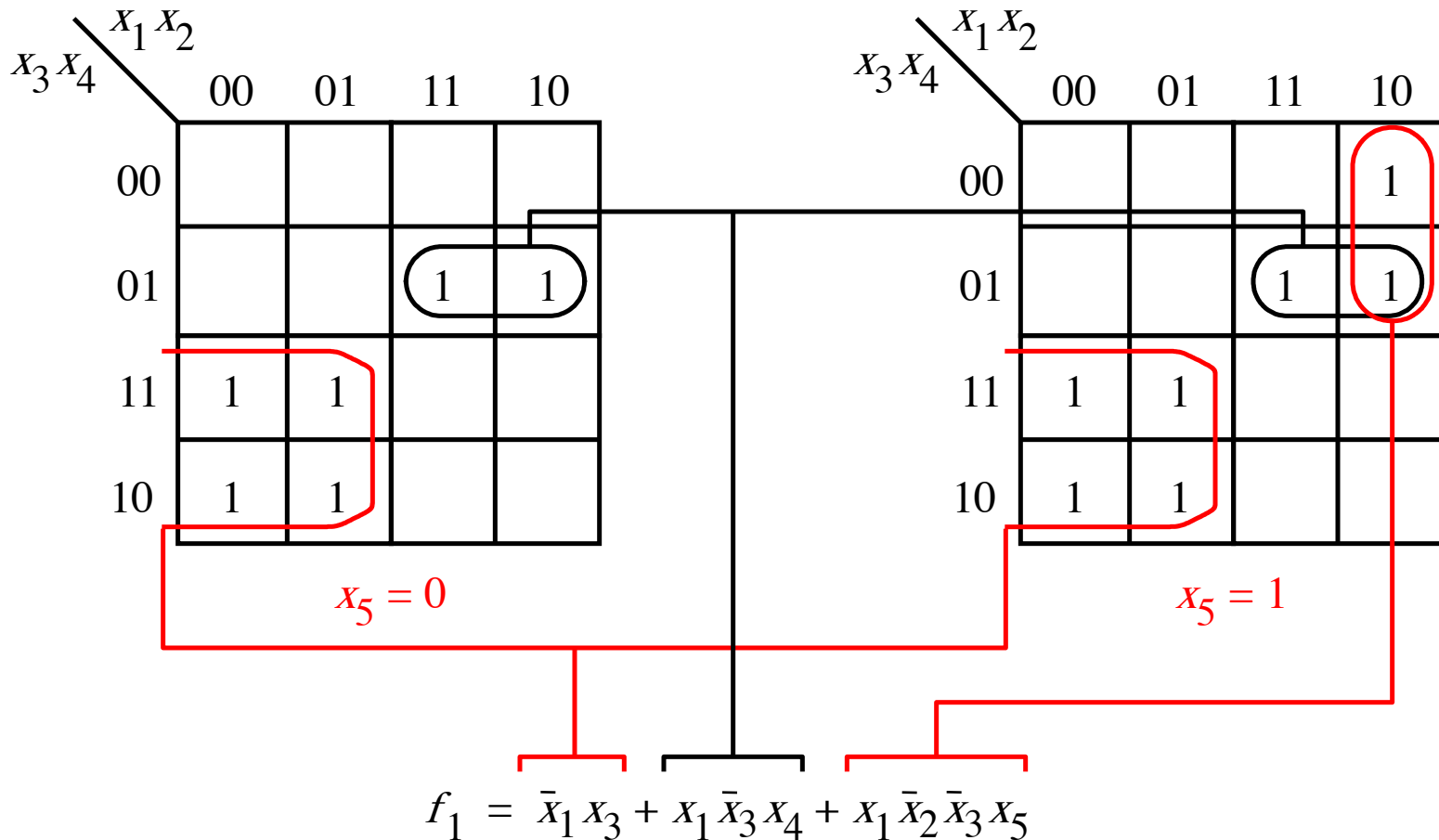
Minimization strategy

Terminology

- Literal
  - Each appearance of a variable in a product term
- Implicant
  - A product term that indicates a set of valuations for which a given function is equal to 1
- Prime implicant
  - Cannot be combined into another implicant that has fewer literals
- Cover
  - A collection of implicants that account for all valuations for which a given function is equal to 1
  - A cover consisting only of prime implicants leads to a minimal cost implementation

Implicants are expanded by square/rectangular powers of 2 to form PIs
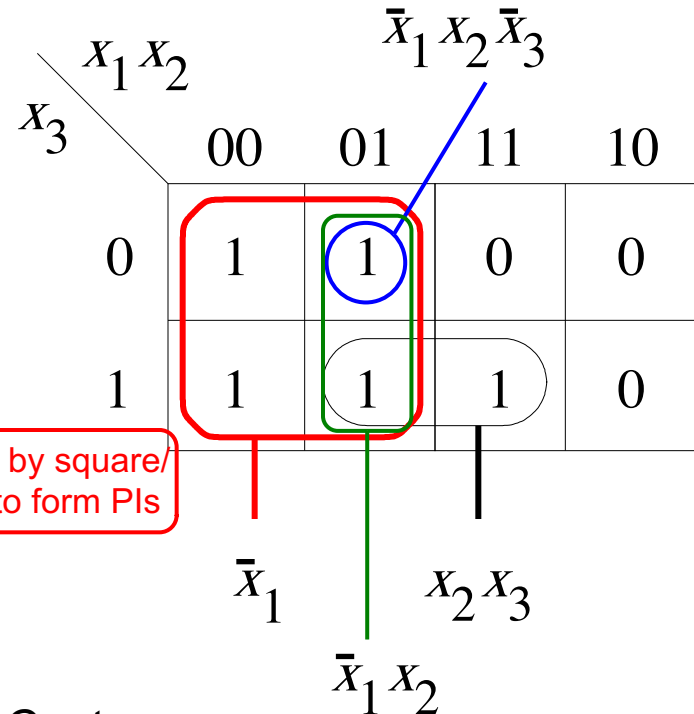
- Cost
  - We use the number of gates plus the total number of inputs to gates, but we assume that the primary inputs are available in both true and complemented form at zero cost – we don't count input inverters

Essential prime implicants

- Where there is choice as to which prime implicants to use in a cover, how do we find the minimum cost subset of prime implicants?

- If a prime implicant includes a minterm for which *f = 1* that is not included in any other prime implicant, <u>then it must be included in the cover</u> and is referred to as an *essential prime implicant*

- 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 included to form a minimum-cost cover

- In this example, $\overline{x}_1 x_3$ is preferred over $\overline{x}_1 x_2 x_4$ for covering $\overline{x}_1 x_2 x_3 x_4$ since it has lower cost

What is the cost of this function?

$x_1 x_2$

$x_3 x_4$

	00	01	11	10
00				
01		1	1	
11	1	1		1
10	1	1	1	1

$\overline{x}_1 x_2 x_4$

$x_2 \overline{x}_3 x_4$

$x_3 \overline{x}_4$

$\overline{x}_1 x_3$

$\overline{x}_2 x_3$
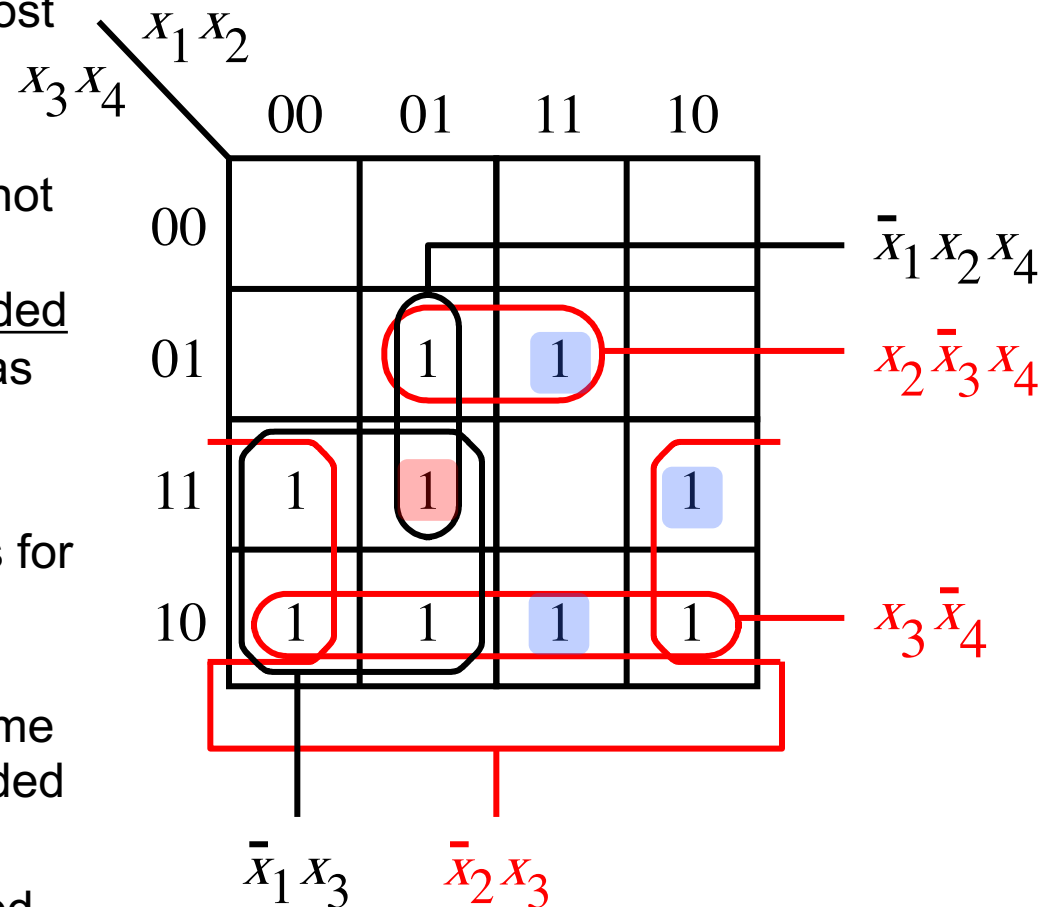
Heuristic choice of nonessential prime implicants

- The choice of nonessential prime implicants to be included in the cover is governed by cost considerations.

- The choice is often not obvious and, for functions of a large number of variables, may involve assessing many possibilities.

- A *heuristic* approach, which considers only a subset of possibilities but gives good results most of the time, is used:

  - *Arbitrarily select one nonessential prime implicant and determine the cost of the resulting cover;*

  - *Next, determine the cost of a cover assuming that the chosen prime implicant <u>is not included</u>; and*

  - *Select for implementation the cover which costs least.*

The function
$f\ (x1,...,x4) = \Sigma\ m(0,\ 4,\ 8,\ 10,\ 11,\ 12,\ 13,\ 15)$
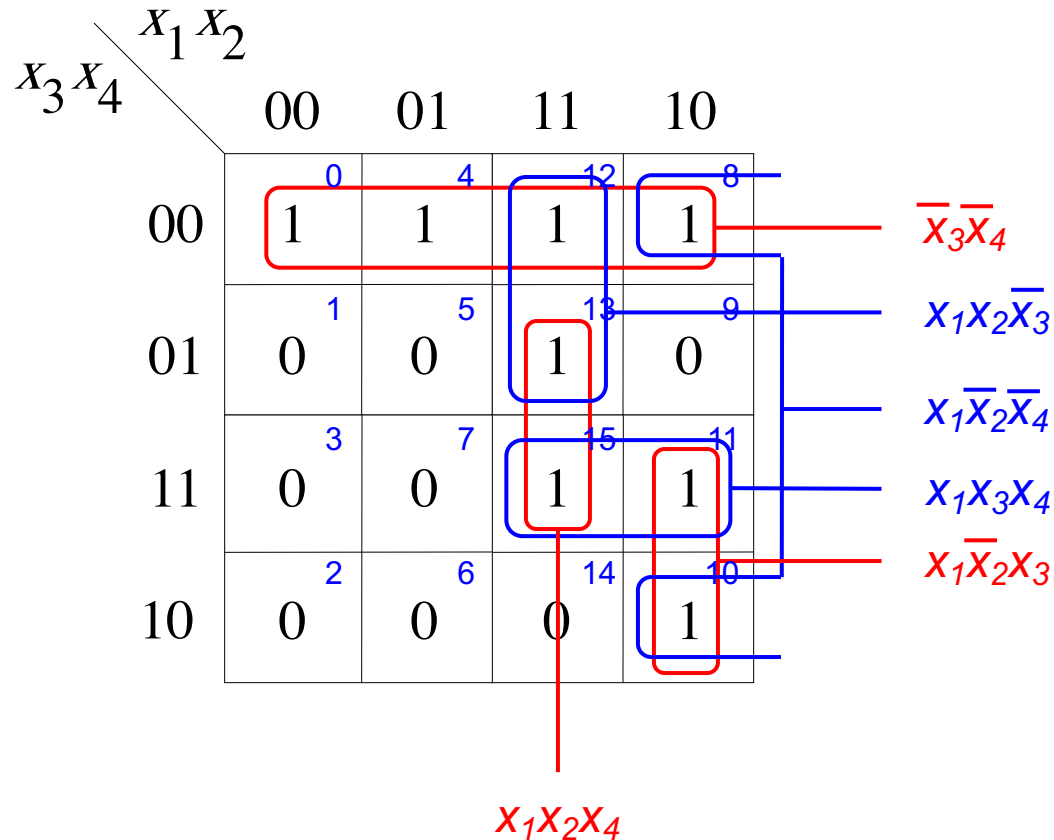
The function
$f(x1,…, x4) = \Sigma\, m(0, 2, 4, 5, 10, 11, 13, 15)$
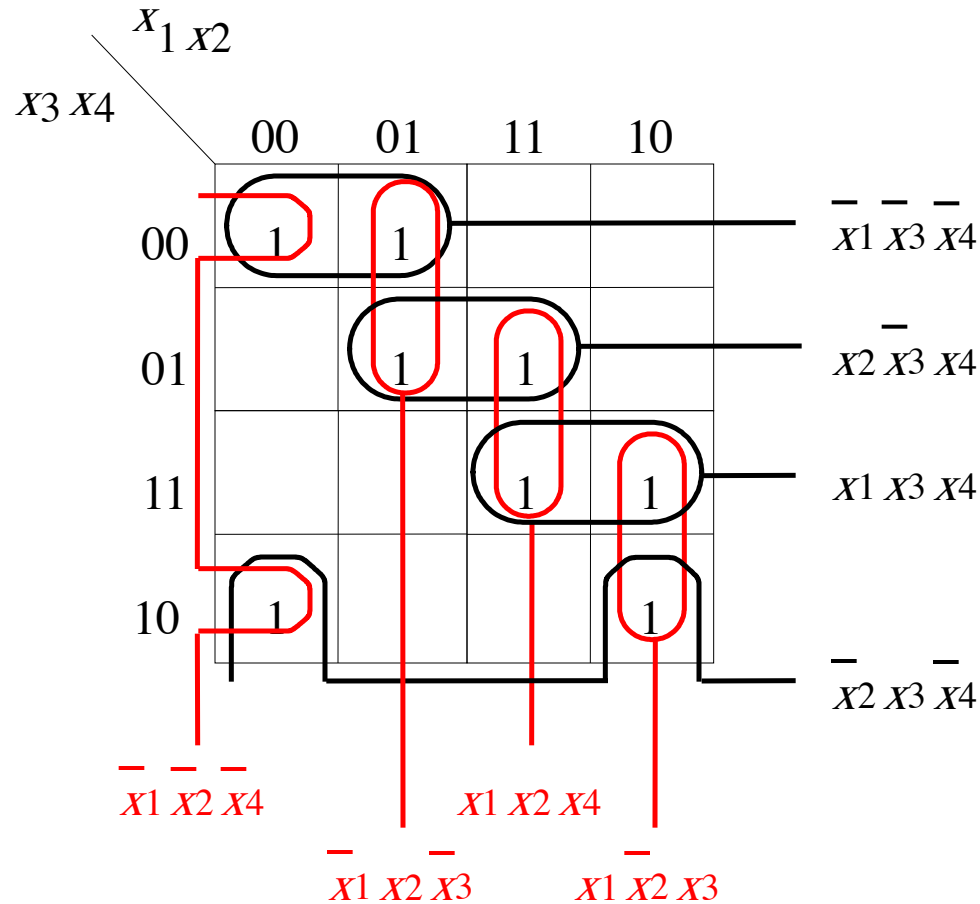
POS minimization of $f(x_1, x_2, x_3) = \Pi M(4, 5, 6)$

- Minimal SOP (sum of product) and POS (product of sum) implementations need to be compared to determine the least cost realization
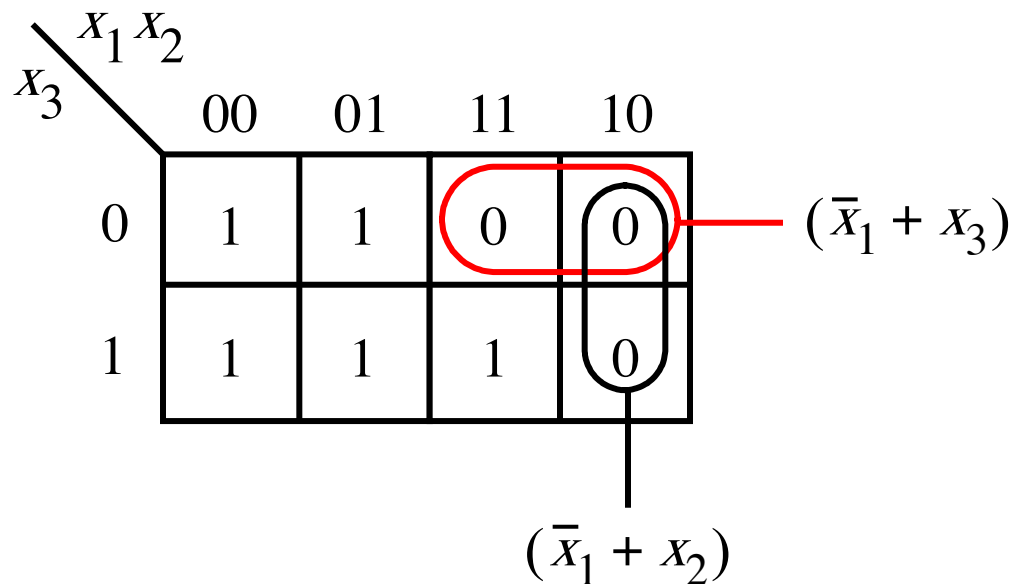
What is the cost of this function?
Which costs less: SOP or POS?

POS minimization of
$f(x1,…, x4) = \Pi\ M(0, 1, 4, 8, 9, 12, 15)$
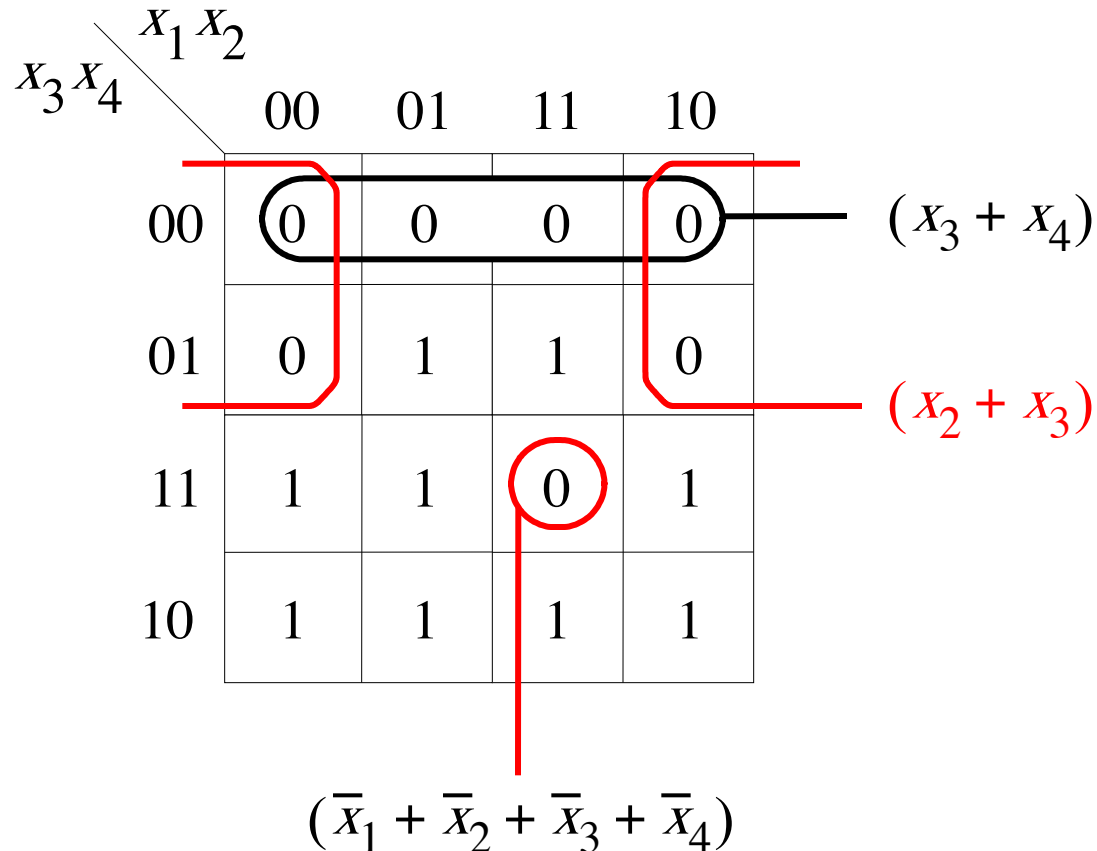
- Which implementation is cheaper: POS or SOP?

Practical Reality 1:
Incompletely specified functions

- Don't care conditions/valuations (incompletely specified functions) offer additional opportunities for simplification

- A don't care can be assigned the value 0 or 1 as desired

- Two implementations of the function
  $f (x1,\ldots, x4) = \Sigma m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$

(a) SOP implementation

(b) POS implementation

Practical Reality 2: Multiple-output synthesis

(a) Function f_1

(b) Function f_2
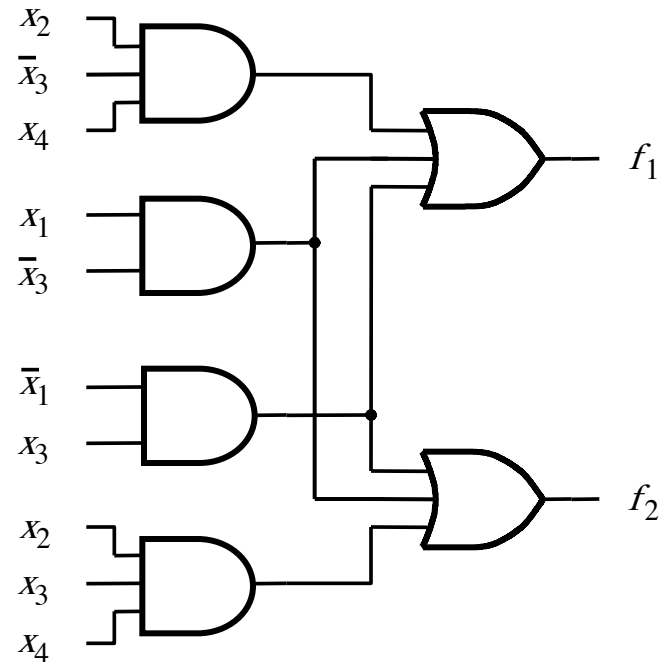
(c) Combined circuit for f_1 and f_2

Another example of multiple-output synthesis
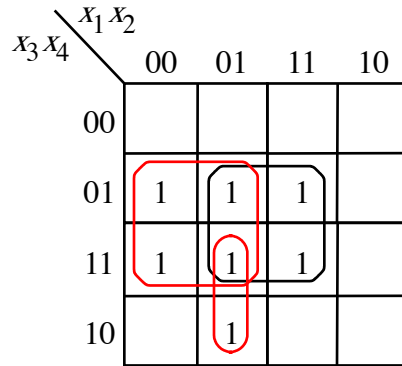
(a) Optimal realization of f_3
Cost = 2x2-in+2x3-in = 14
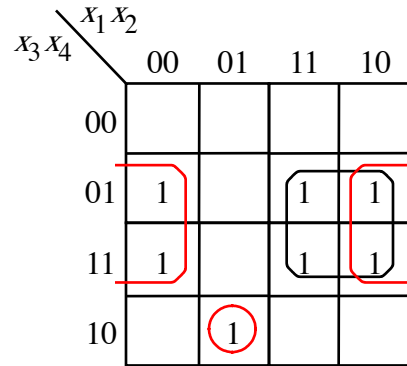
(b) Optimal realization of f_4
Cost = 2x2-in+1x4-in+1x3-in
= 15

Combined cost = 14 + 15 = 29
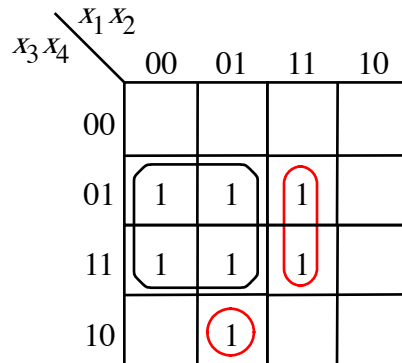
(c) Optimal realization of f_3 and f_4 together
Combined cost = 2x2-in+3x3-in+1x4-in = 23
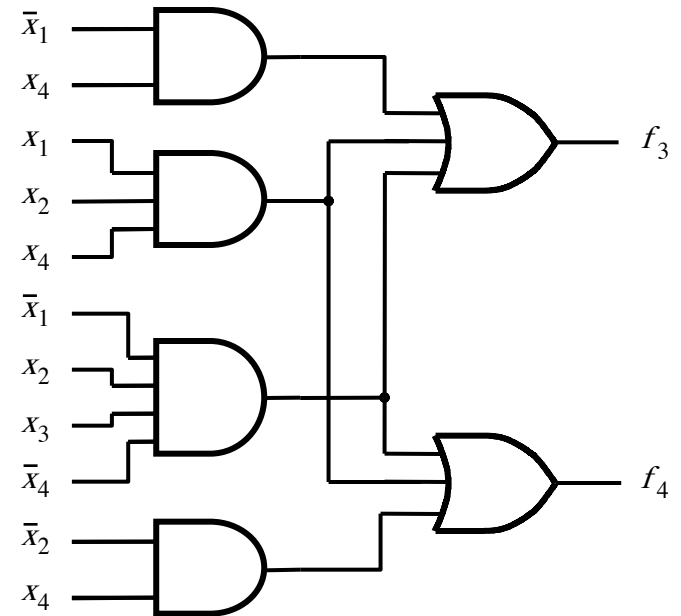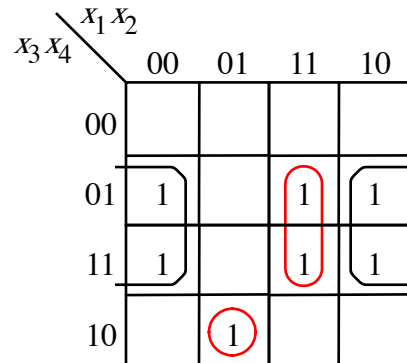
(d) Combined circuit for f_3 and f_4

Note that the realizations of (c) are sub-optimal if considered individually

Practical Reality 3: Factoring for multilevel implementation

$$f(x_1,...,x_7) = x_1x_3\overline{x_6} + x_1x_4x_5\overline{x_6} + x_2x_3x_7 + x_2x_4x_5x_7$$
$$= x_1\overline{x_6}(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5) \text{ [Distributive law]}$$
$$= (x_1\overline{x_6} + x_2x_7)(x_3 + x_4x_5)$$

- Factoring is necessary when the implementation resources (logic gates) haven't got the capacity to implement a function

- Factoring allows a function to be implemented from simpler sub-functions
  - Here, 4-input AND and OR gates are needed to implement *f* naively, but suppose the implementation technology we use only provides 2- or 3-input gates

Implementation technologies

- Digital circuits are implemented in various solid state/silicon device technologies

- Their functions can be classified as either fixed or programmable

- Devices that are manufactured to perform fixed functions range from small-scale integrated devices that contain a few specific logic gates to custom VLSI circuits that comprise millions of gates

- Programmable devices can be customized after fabrication to implement specific circuits

  – Examples include Programmable Logic Arrays, Complex Programmable Logic Devices and FPGAs

Q: Is an ARM or Intel processor fixed or programmable?
Q: How does the programmability of a processor differ from that of an FPGA?

A Field-Programmable Gate Array (FPGA)

Logic block Interconnection switches

I/O block

I/O block

I/O block

I/O block
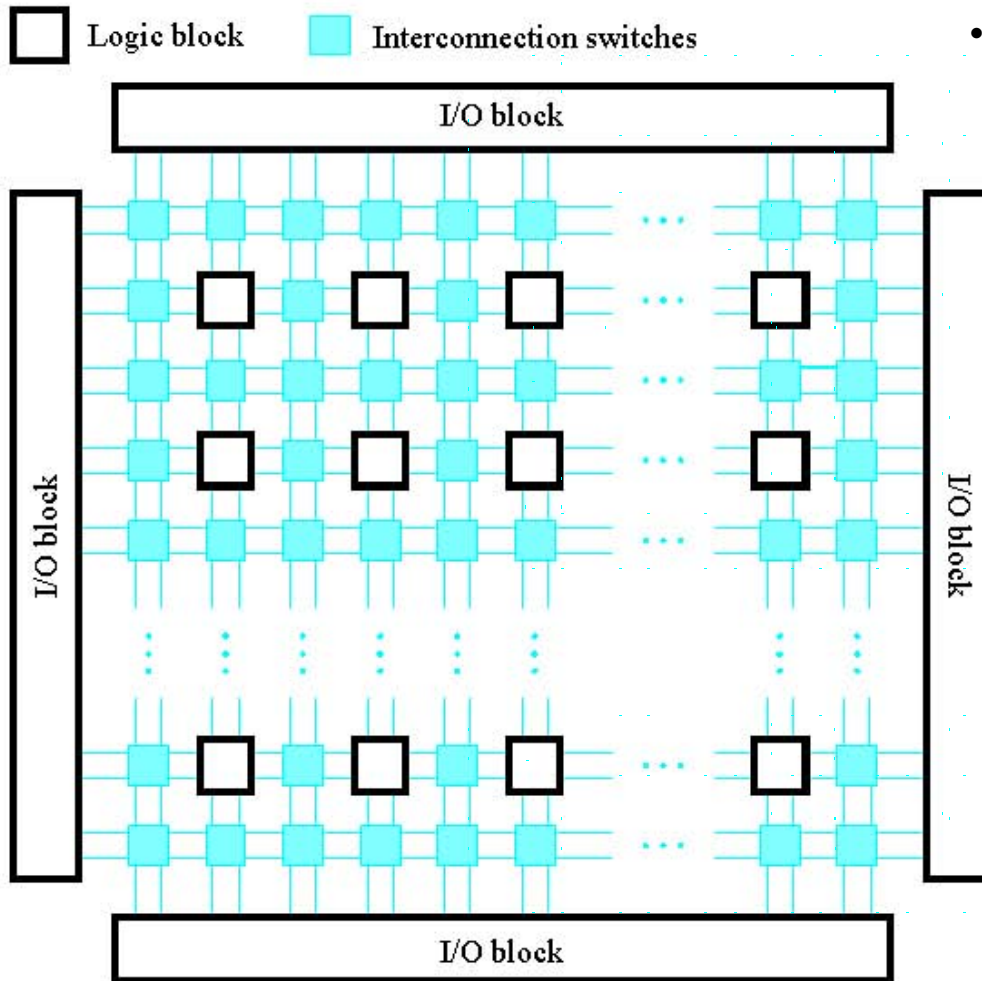
(a) General structure of an FPGA

(b) Pin grid array (PGA) package (bottom view)

- For cost and performance reasons, as few chips as possible are used to implement a design
  - 7400-series chips implement the equivalent of just a few two-input NAND gates (the prevalent metric for chip size)
  - An SPLD or CPLD macrocell represents about 20 equivalent gates; thus a PAL with 8 macrocells can accommodate a circuit that needs up to about 160 gates and a large CPLD with 500 macrocells can implement circuits of up to 10,000 equivalent gates
  - Modern FPGAs can be used to implement circuits of millions of equivalent gates in size

B&V3, Figure 3.35

A two-input lookup table (LUT) as an FPGA logic cell/block
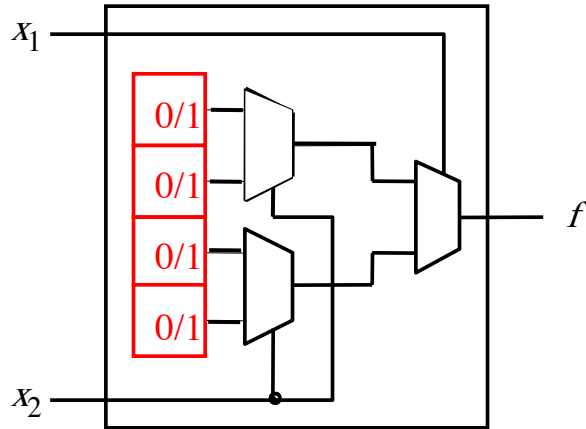
(a) Circuit for a two-input LUT

x_1	x_2	f_1
0	0	1
0	1	0
1	0	1
1	1	1

(b) $f_1 = \bar{x}_1 \bar{x}_2 + x_1$
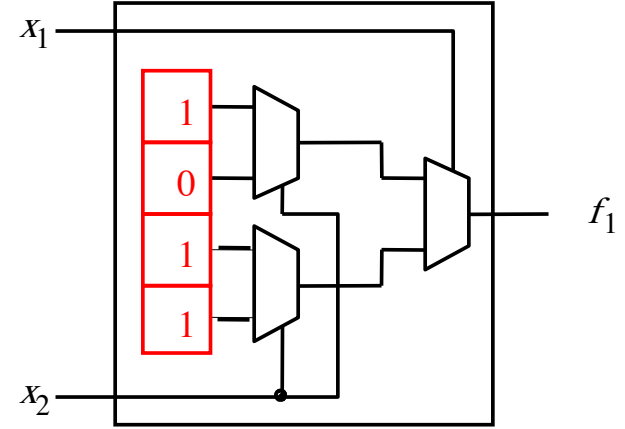
(c) Storage cell contents in the LUT

An *n*-input LUT serves as a small 2^n address memory to implement an arbitrary Boolean function of *n* variables

s	f
0	x_0
1	x_1
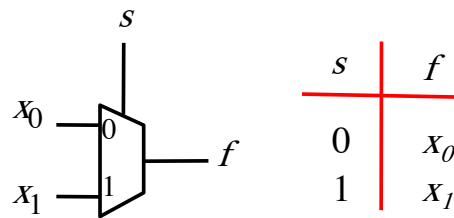
A two-input multiplexer (MUX) $f = \bar{s}.x_0 + s.x_1$ selects one of its two inputs to be routed to the output
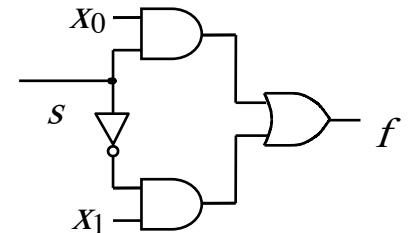
Two-input multiplexer circuit (2-MUX)

B&V3, Figure 3.36

A three-input LUT

Optimized Implementation of Logic Functions
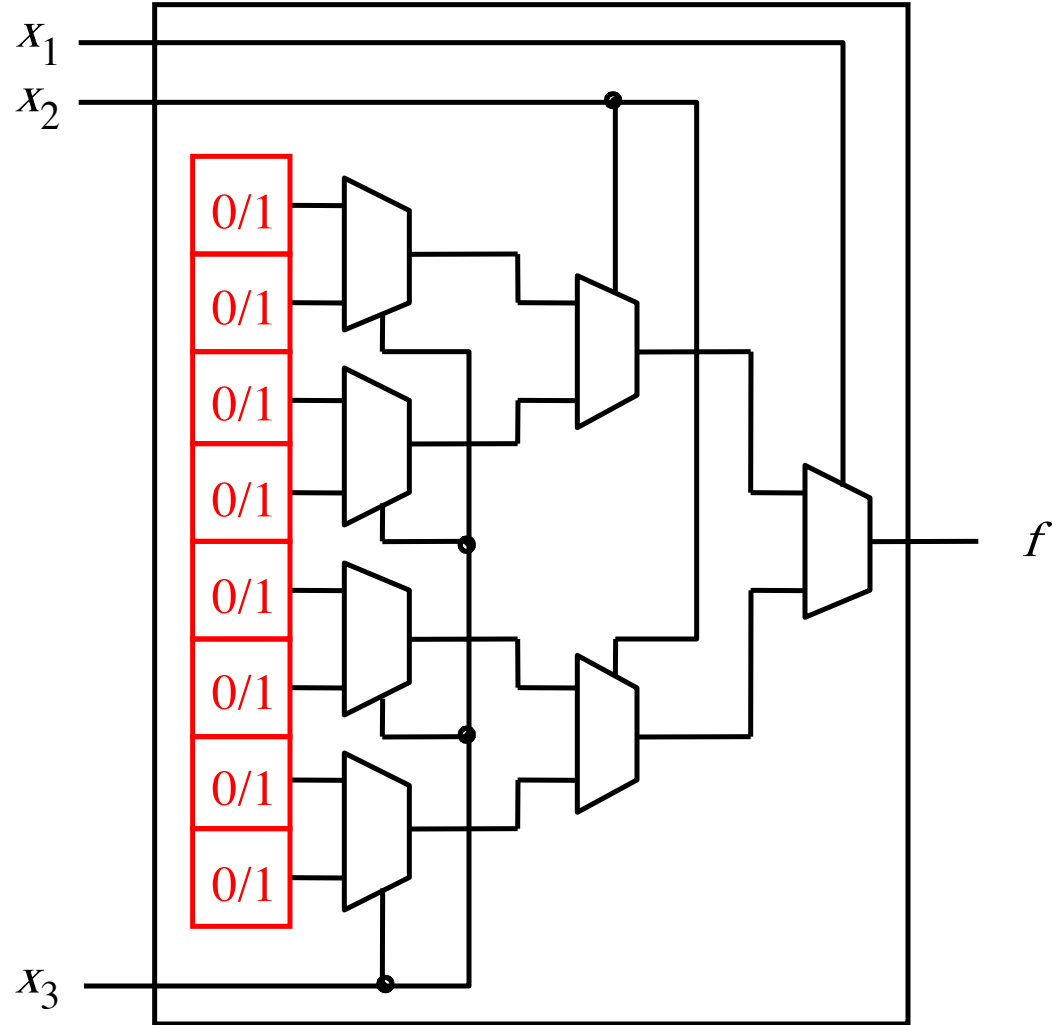
A section of a programmed FPGA
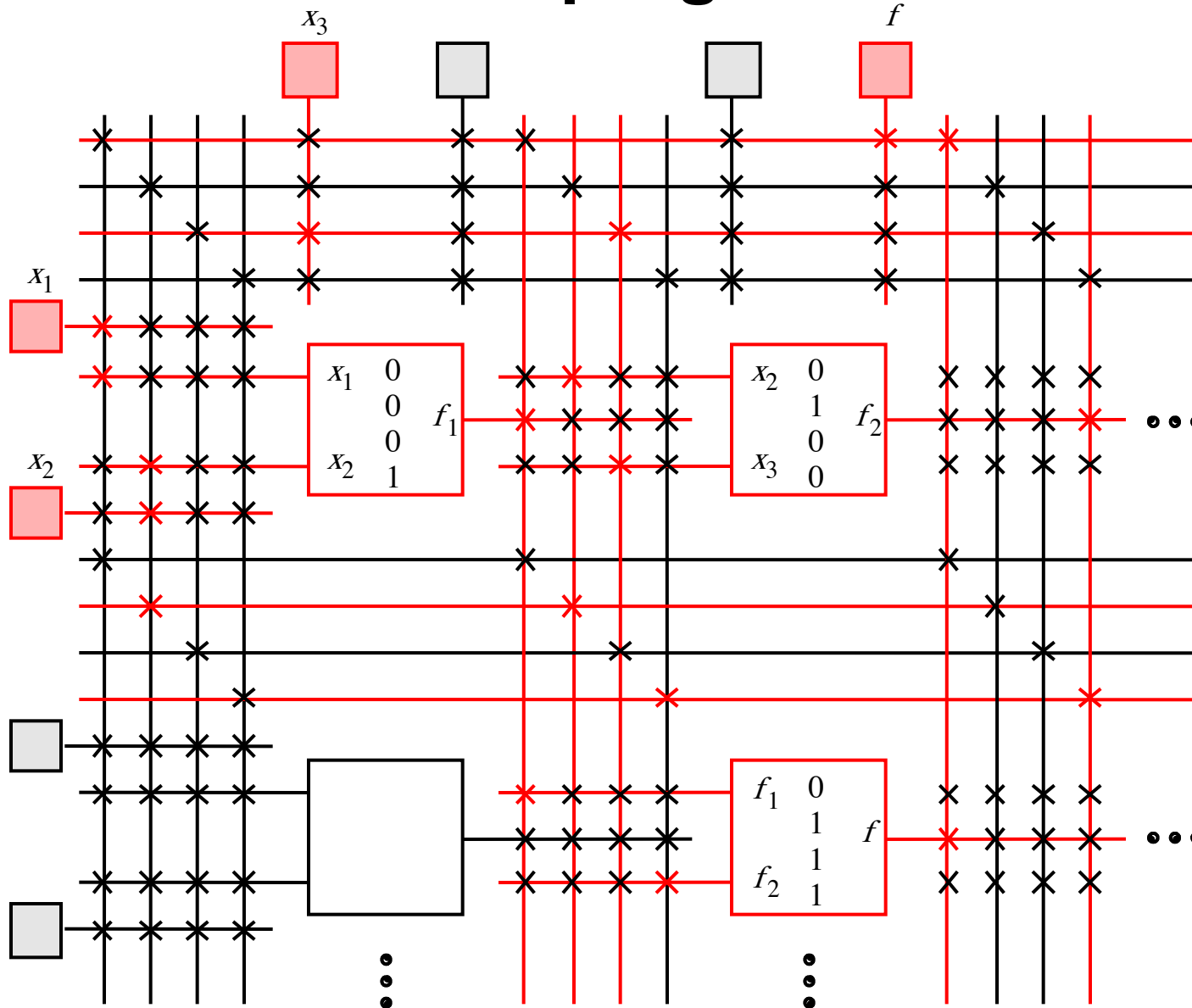
$f_1 = x_1 x_2$

$f_2 = \overline{x}_2 x_3$

$f = f_1 + f_2$

$\quad = x_1 x_2 + \overline{x}_2 x_3$

In an FPGA, the LUT contents, routing switches and connection boxes are most commonly configured via memory cells

B&V3, Figure 3.39

Practical Reality 3: Factoring for multilevel implementation

$$f(x_1, \ldots, x_7) = x_1 x_3 \overline{x_6} + x_1 x_4 x_5 \overline{x_6} + x_2 x_3 x_7 + x_2 x_4 x_5 x_7$$

$$= x_1 \overline{x_6}(x_3 + x_4 x_5) + x_2 x_7 (x_3 + x_4 x_5) \text{ [Distributive law]}$$

$$= (x_1 \overline{x_6} + x_2 x_7)(x_3 + x_4 x_5)$$
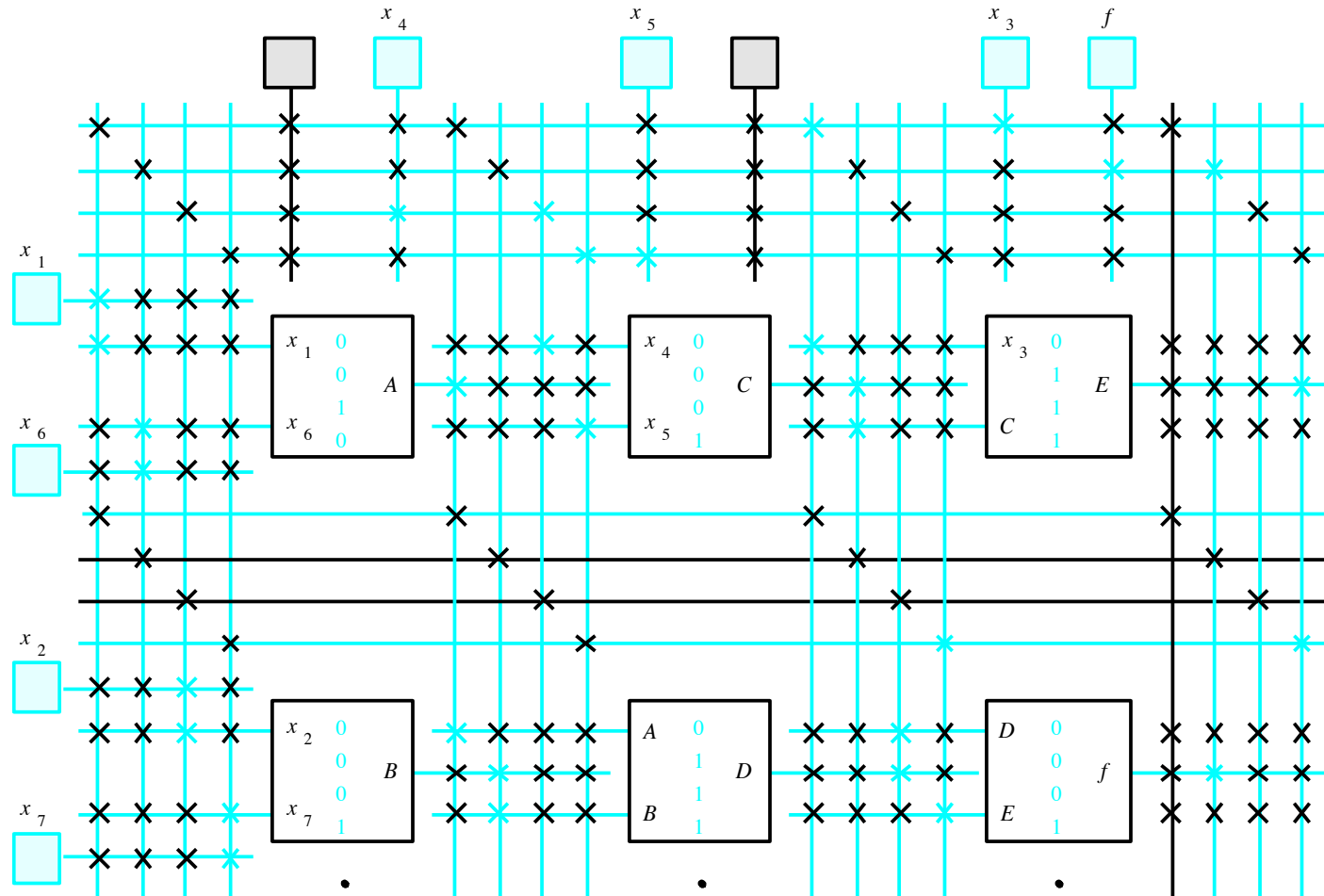
The VHDL code for the function of L02/S24

```
ENTITY func3 IS
     PORT ( x1, x2, x3, x4, x5, x6, x7     : IN     BIT;
              f                             : OUT  BIT);
END func3;

ARCHITECTURE LogicFunc OF func3 IS
BEGIN
     f <= (x1 AND x3 AND NOT x6) OR
          (x1 AND x4 AND x5 AND NOT x6) OR
          (x2 AND x3 AND x7) OR
          (x2 AND x4 AND x5 AND x7);
END LogicFunc;
```

Synthesizes as
$f = (x_1\overline{x}_6 + x_2x_7)(x_3 + x_4x_5)$

Realizing a seven-input product term

- Implementing wide functions when gate fan-in is limited

7 inputs
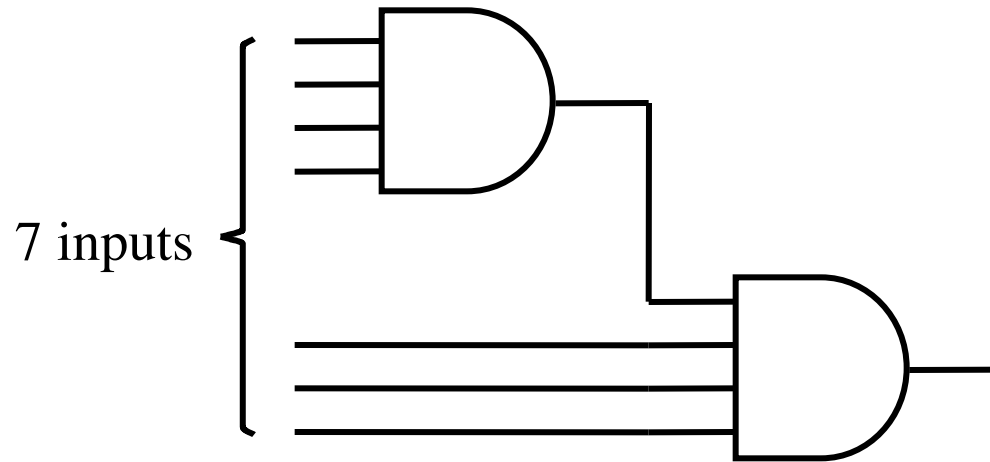
Still Practical Reality 3: A factored circuit

- Factoring can be used to overcome fan-in constraints

- For example, $f = x_1\overline{x_2}x_3\overline{x_4}x_5x_6 + x_1x_2\overline{x_3}\overline{x_4}\overline{x_5}x_6$
$= x_1\overline{x_4}x_6(\overline{x_2}x_3x_5 + x_2\overline{x_3}\overline{x_5})$ for FO4 (fan-in of 4) gates

$\overline{X_2}$
X_3
X_5

X_1
$\overline{X_4}$
X_6

X_2
$\overline{X_3}$
$\overline{X_5}$
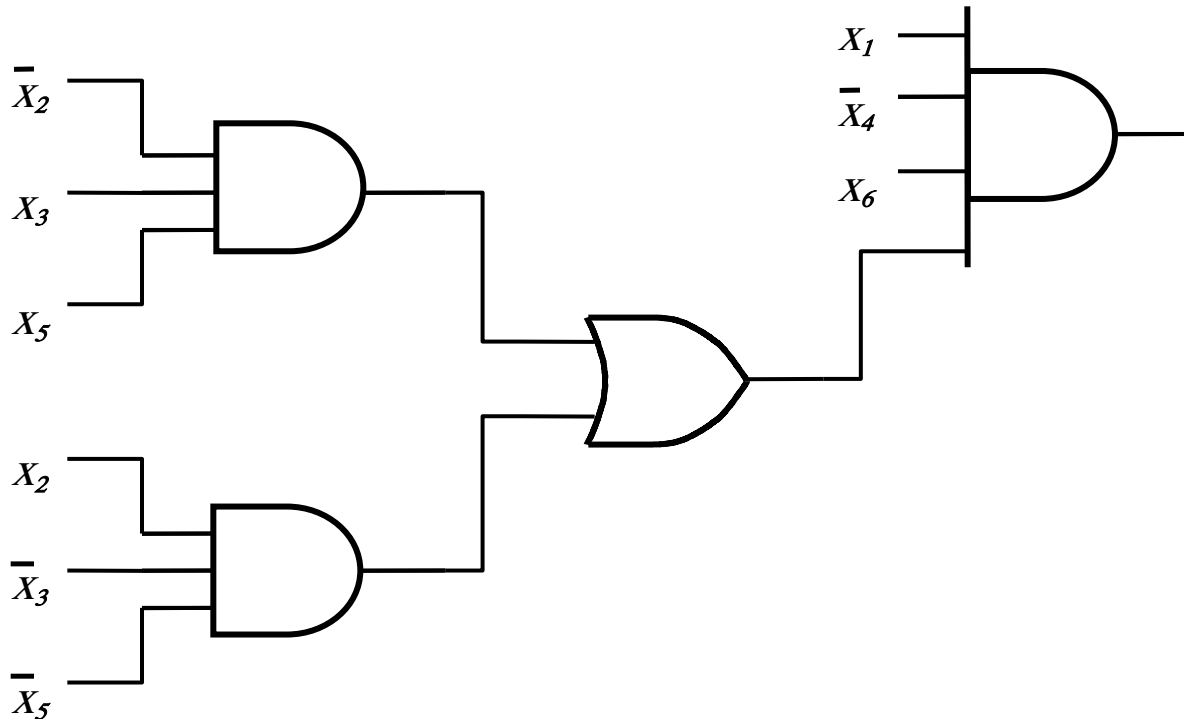
Practical Tradeoff:
Functional decomposition

- Multilevel realizations can reduce the cost of a circuit with increased propagation delay penalties

- Consider the minimum-cost SOP expression

$f = \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 x_4 + \overline{x}_1 \overline{x}_2 x_4$ [7 gates + 18 inputs]*

$= (\overline{x}_1 x_2 + x_1 \overline{x}_2) x_3 + (x_1 x_2 + \overline{x}_1 \overline{x}_2) x_4$

$= g x_3 + \overline{g} x_4$ with $g = \overline{x}_1 x_2 + x_1 \overline{x}_2$ [9 FO2 gates+15 inputs]*

* inc. inverters
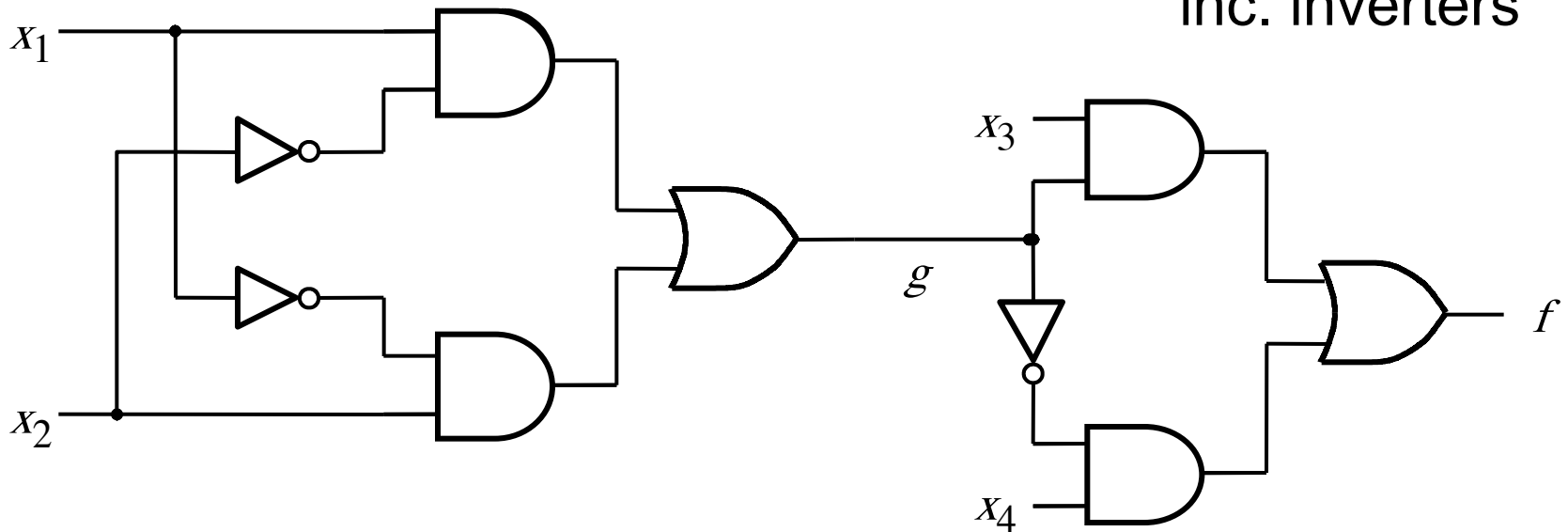
The structure of decomposition in L02/S34
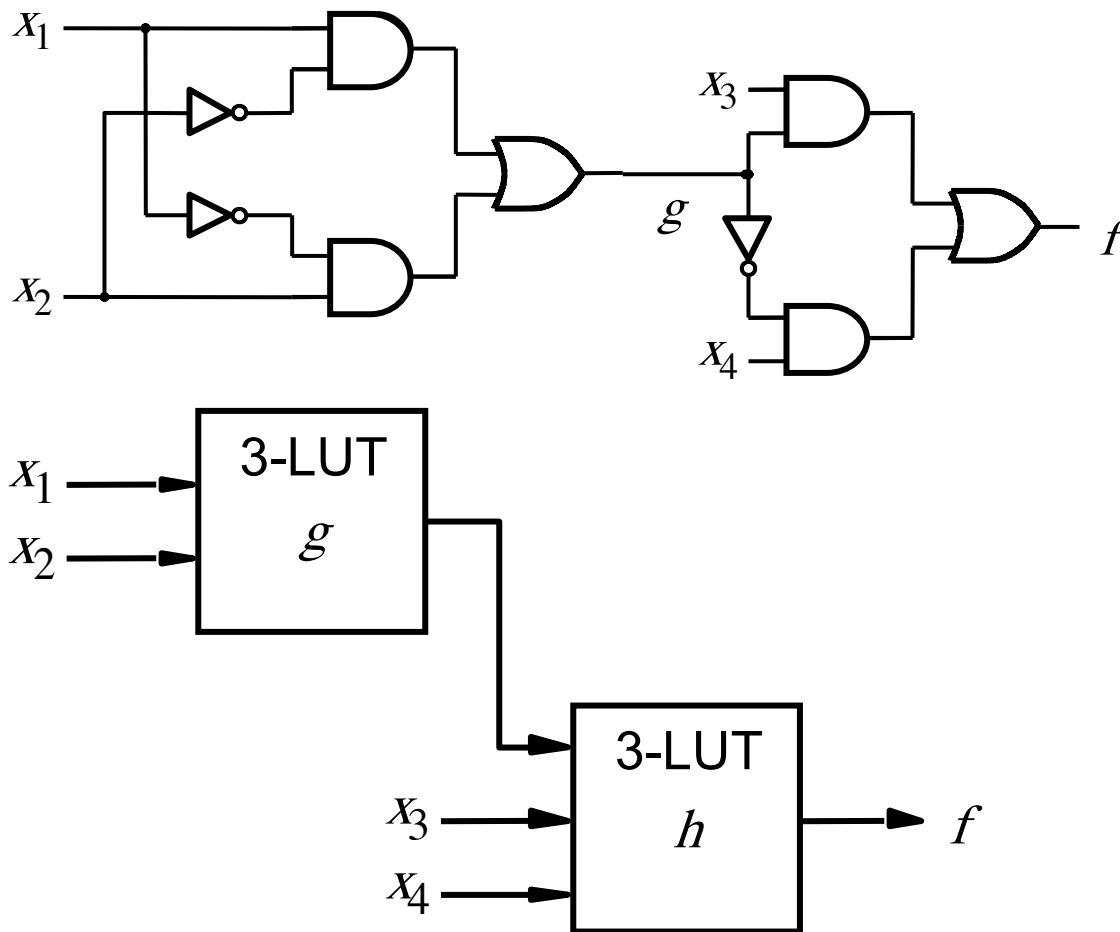
Q: How many 2-LUTs are needed
to implement this function?

Decomposition for Example 4.7

$x_1 x_2$
$x_3 x_4$

	00	01	11	10
00	1			
01		1	1	1
11	1			
10		1	1	1

$x_5 = 0$

$x_1 x_2$
$x_3 x_4$

	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$x_5 = 1$

(a) Karnaugh map for the function f
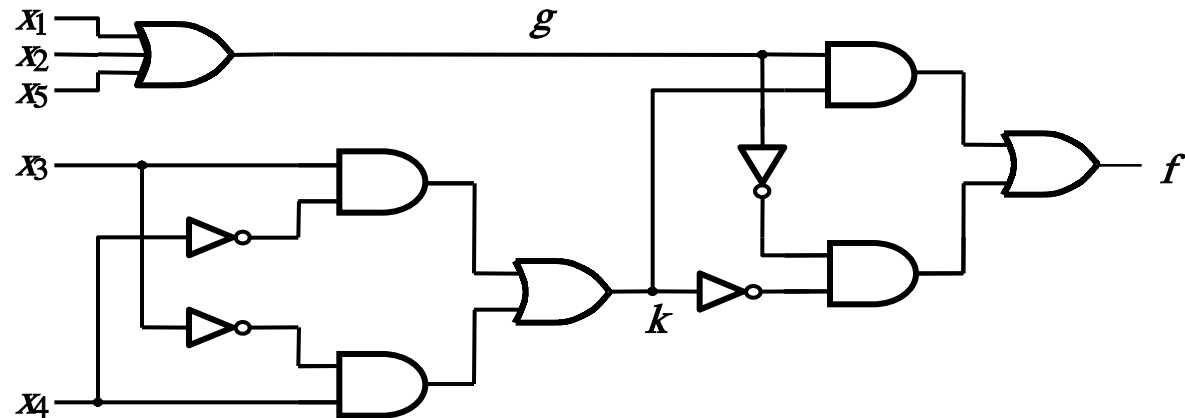
(b) Circuit obtained using decomposition

Q: How many 3-LUTs are needed
to implement this function?

NAND and NOR gates

In custom VLSI implementations, NAND & NOR gate circuit realizations of SOP/POS networks are preferred over AND/OR/NOT gate realizations because they require less transistors to implement

x_1 ——⊐○— $\overline{x_1 \cdot x_2}$
x_2

x_1 ——
x_2 ——
⋮
x_n —— ⊐○— $\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$

(a) NAND gates

x_1 ——⊐○— $\overline{x_1 + x_2}$
x_2

x_1 ——
x_2 ——
⋮
x_n —— ⊐○— $\overline{x_1 + x_2 + \ldots + x_n}$
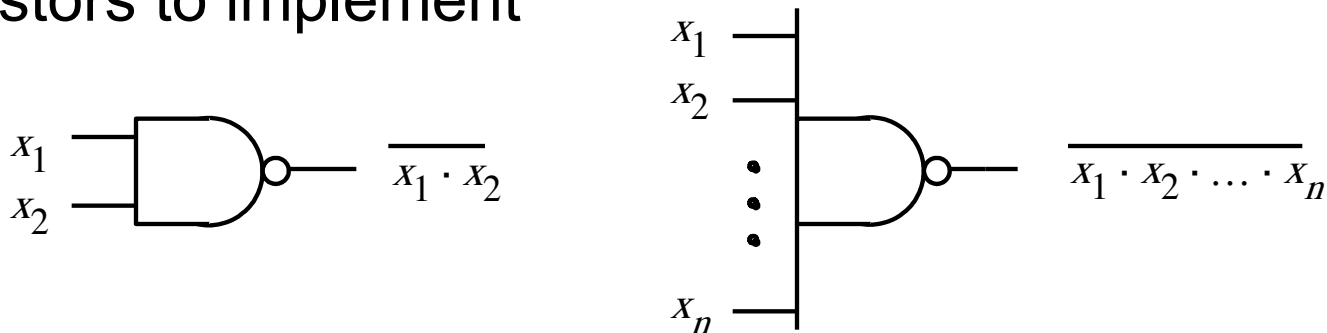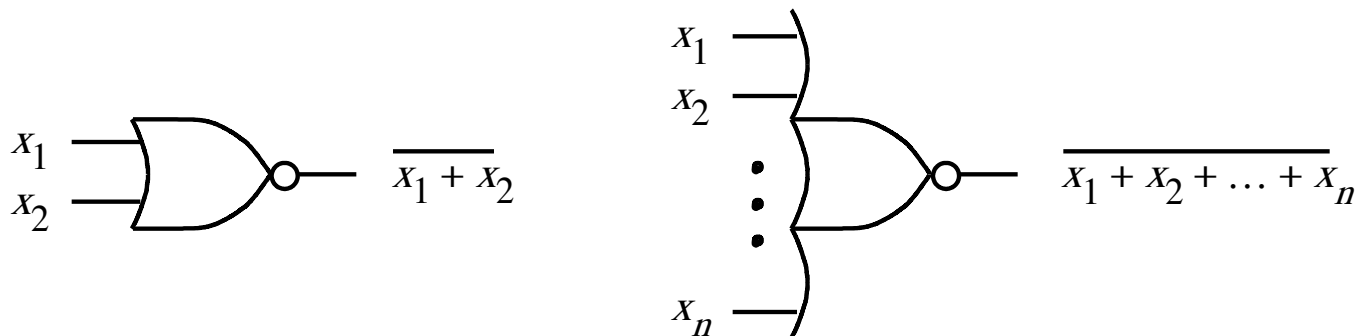
(b) NOR gates

DeMorgan's theorem in terms of logic gates

Allows us to transform AND-OR networks into NAND-only or NOR-only equivalent networks
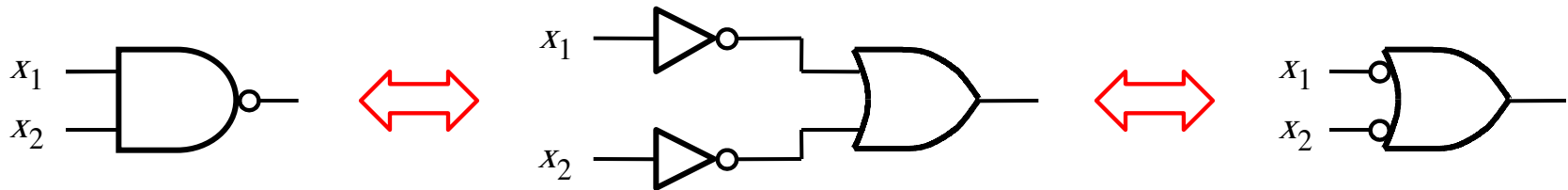
(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$
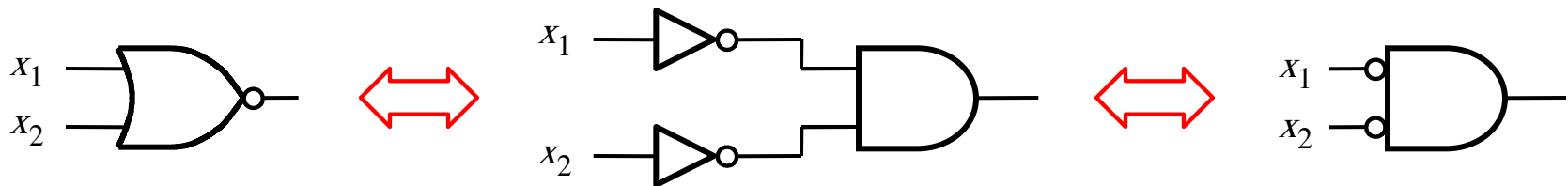
(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

Using NAND gates to implement a SOP network
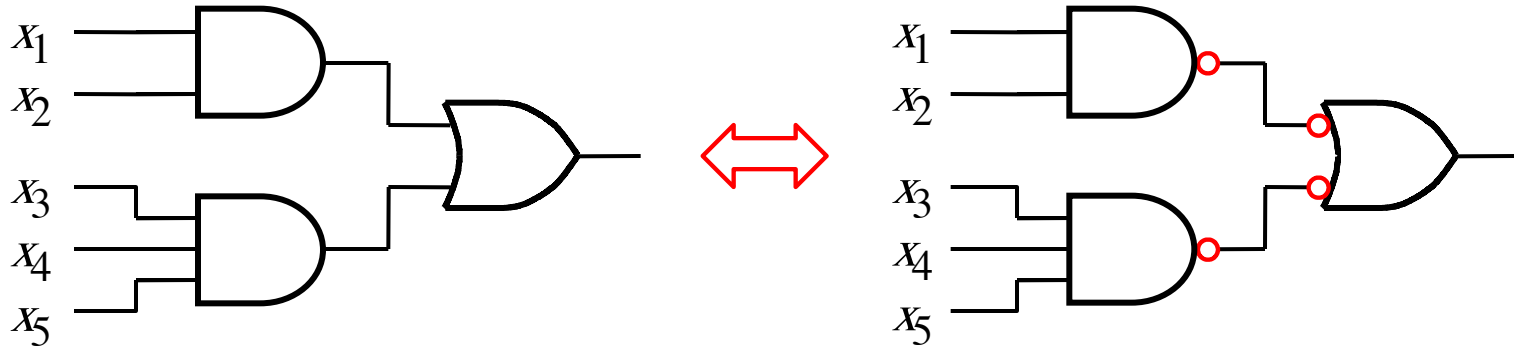
To transform network into NAND-only type:

1. Replace AND & OR gates with equivalent NAND gates

2. Ensure the logical value of no wire is changed due to step 1. Insert additional bubbles into wires where only one bubble was added during step 1.
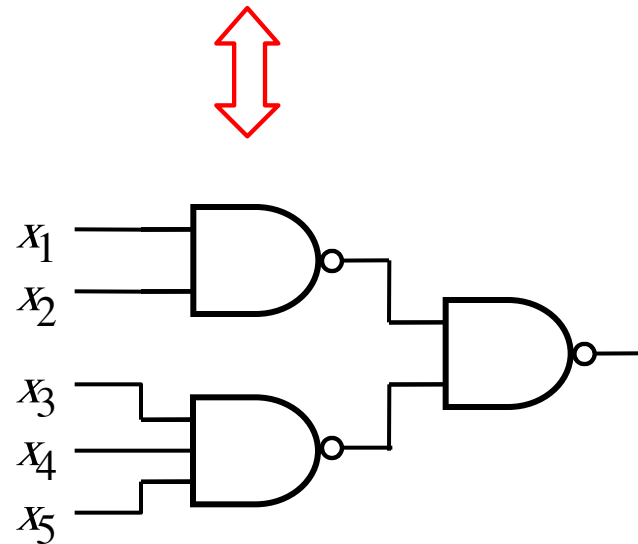
Using NOR gates to implement a POS network
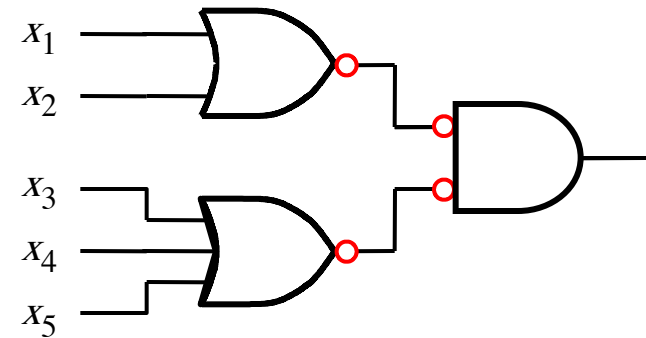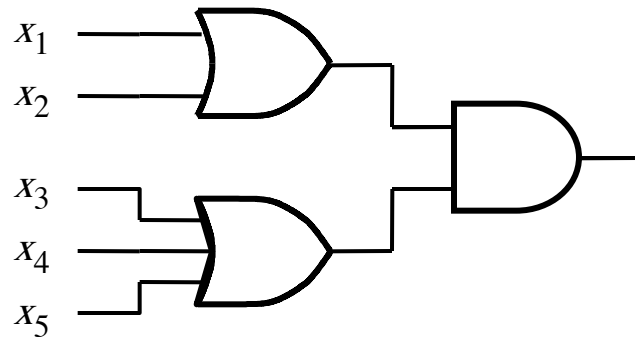
Follow a similar algorithm to transform network into NOR-only type
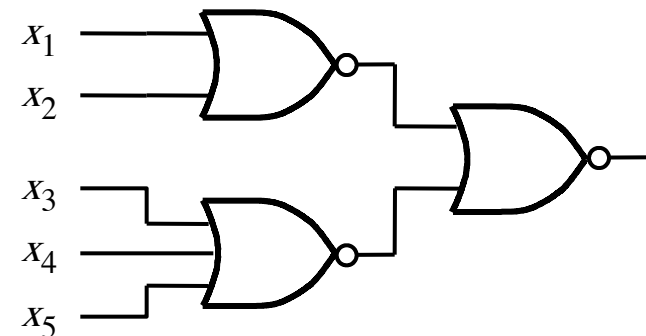
Example 2.3

- Consider the function $f(x_1, x_2, x_3) = \Sigma m(2, 3, 4, 6, 7)$

- The canonical SOP expression is derived using minterms $f = m_2 + m_3 + m_4 + m_6 + m_7$

  $= \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3 + x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + x_1x_2x_3$

- The expression can be simplified using algebraic manipulation or a Karnaugh map:

  $f = \overline{x_1}x_2(\overline{x_3} + x_3) + x_1(\overline{x_2} + x_2)\overline{x_3} + x_1x_2(\overline{x_3} + x_3)$

  $= \overline{x_1}x_2 + x_1\overline{x_3} + x_1x_2$

  $= (\overline{x_1} + x_1)x_2 + x_1\overline{x_3}$

  $= x_2 + x_1\overline{x_3}$

NAND-gate realization of the function in Example 2.3
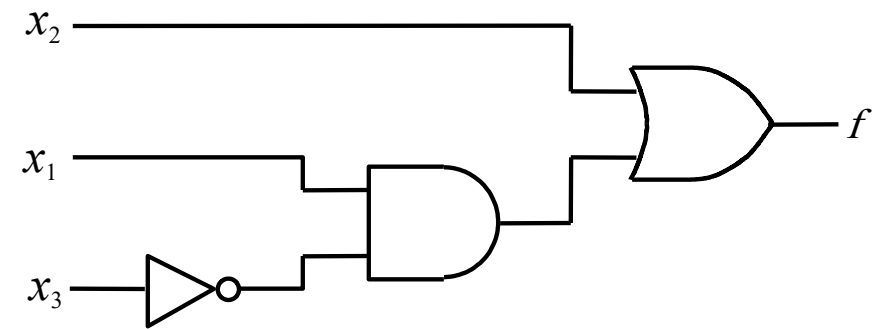
(a) SOP implementation
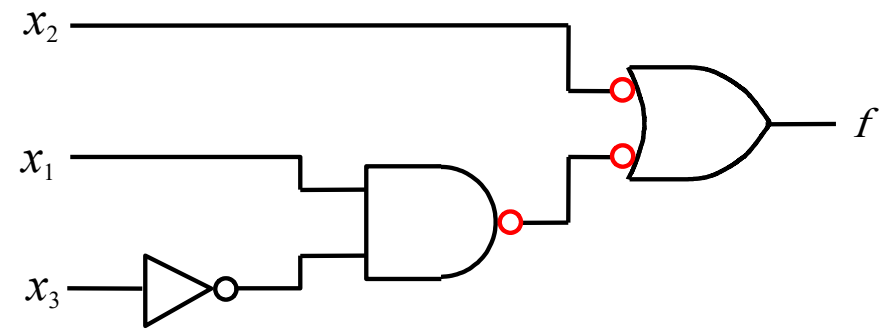
(b) Partial NAND conversion
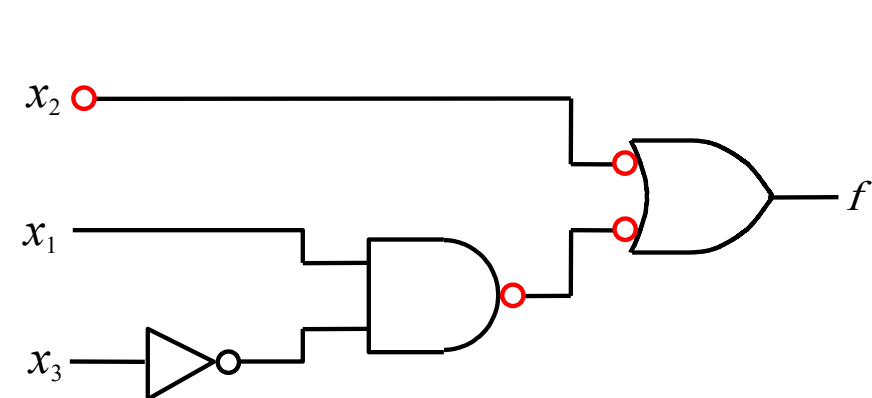
(c) Balancing inserted inversions
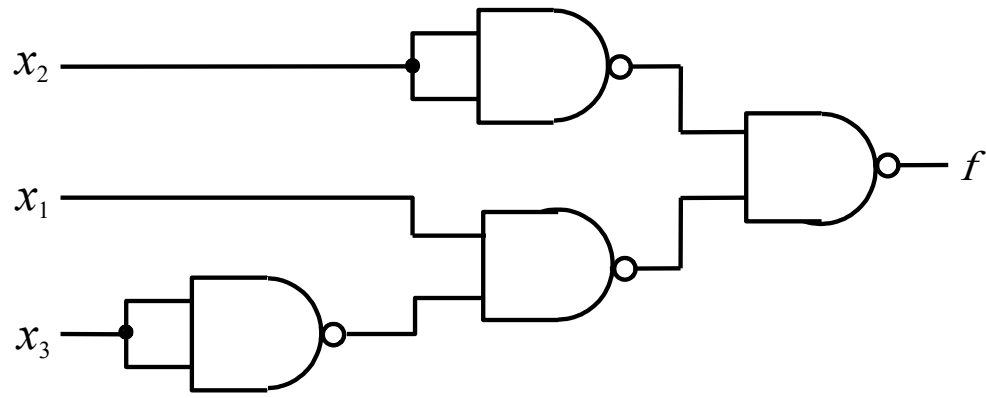
(d) Resulting NAND implementation

Example 2.4

- Consider again the function of Example 2.3. Instead of using the minterms, we can specify this function as a product of maxterms for which $f = 0$, namely,

$f(x_1, x_2, x_3) = \Pi M(0, 1, 5)$

- Then the canonical POS expression is derived as

$f = M_0 \cdot M_1 \cdot M_5$

$= (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x_3})(\overline{x_1} + x_2 + \overline{x_3})$

- A simplifeied POS expression can be derived as

$f = ((x_1 + x_2) + x_3)((x_1 + x_2) + \overline{x_3})(x_1 + (x_2 + \overline{x_3}))(\overline{x_1} + (x_2 + \overline{x_3}))$

$= ((x_1 + x_2) + x_3\overline{x_3})(x_1\overline{x_1} + (x_2 + \overline{x_3}))$

$= (x_1 + x_2)(x_2 + \overline{x_3})$

NOR-gate realization of the function in Example 2.4
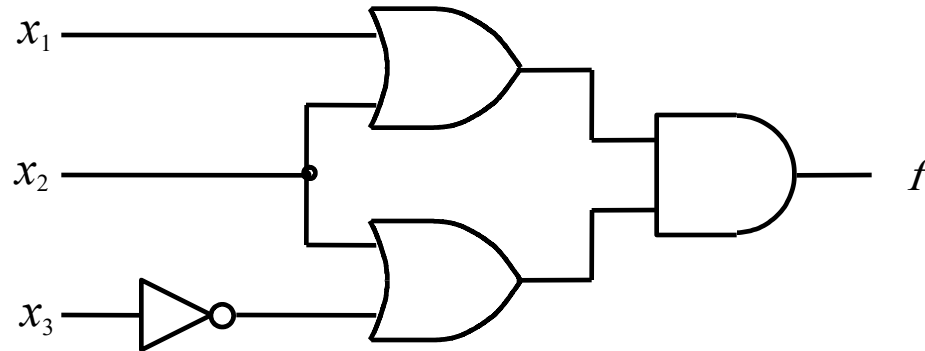
(a) POS implementation
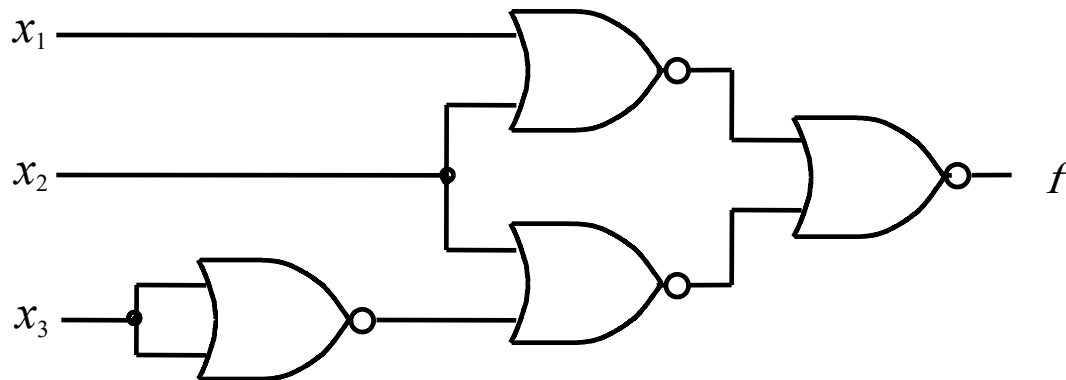
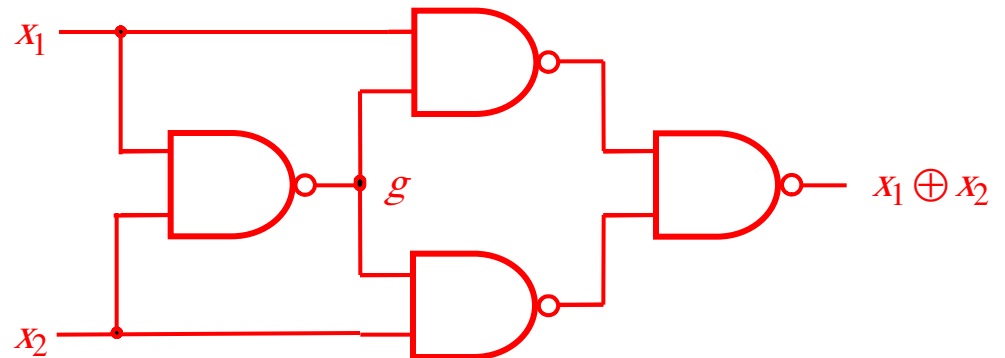(b) NOR implementation

Implementation of XOR
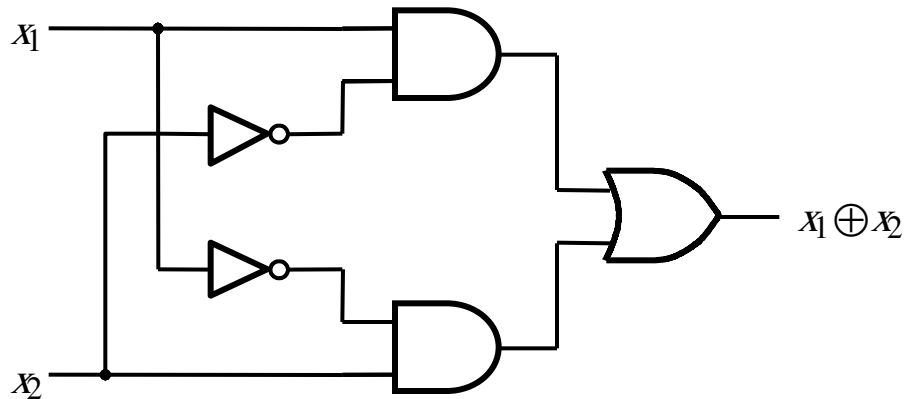
(a) Sum-of-products implementation
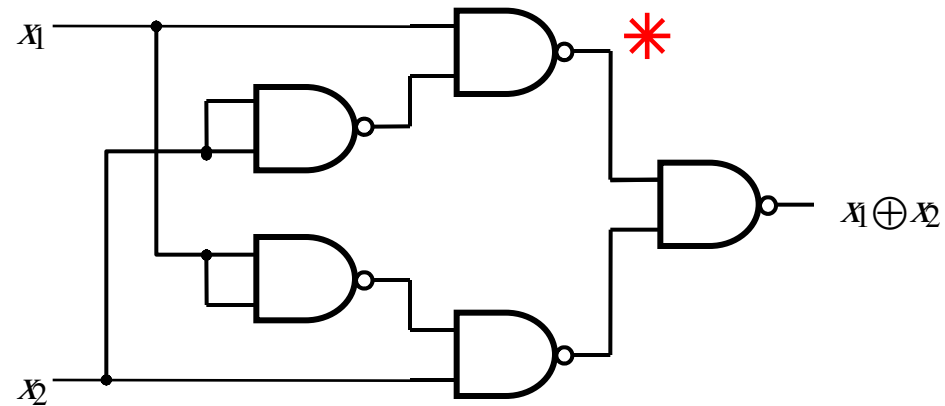
(b) NAND gate implementation

Since $\overline{(x_1 \cdot \overline{x_2})} = \overline{(0 + x_1 \cdot \overline{x_2})}$
$= \overline{(x_1 \cdot \overline{x_1} + x_1 \cdot \overline{x_2})}$
$= \overline{(x_1 \cdot (\overline{x_1} + \overline{x_2}))}$
$= \overline{x_1 \cdot \overline{(x_1 \cdot x_2)}}$

(c) Optimal NAND gate implementation

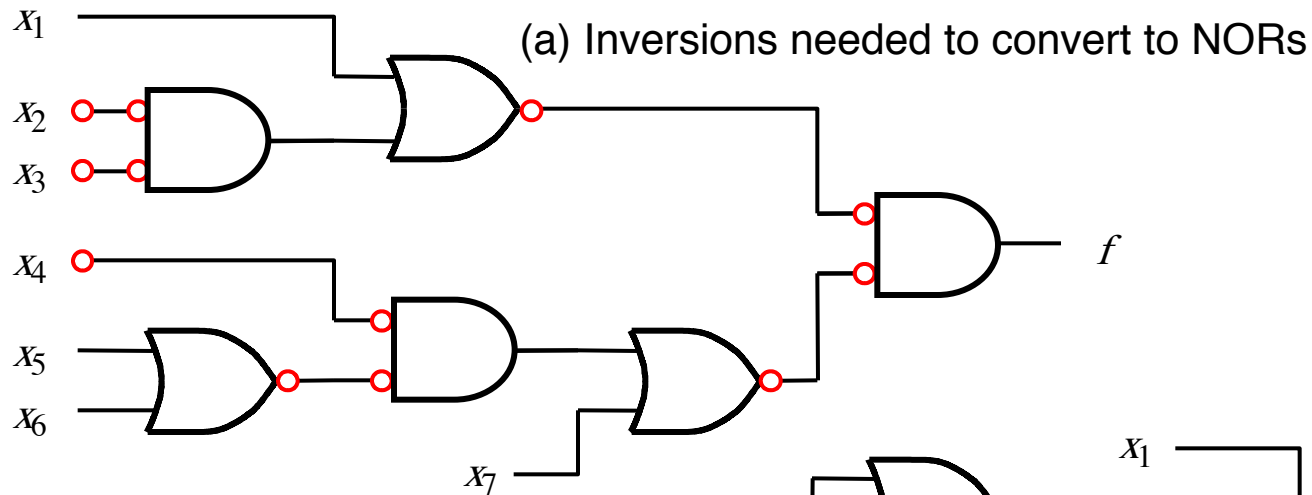# Multilevel NAND-gate circuit
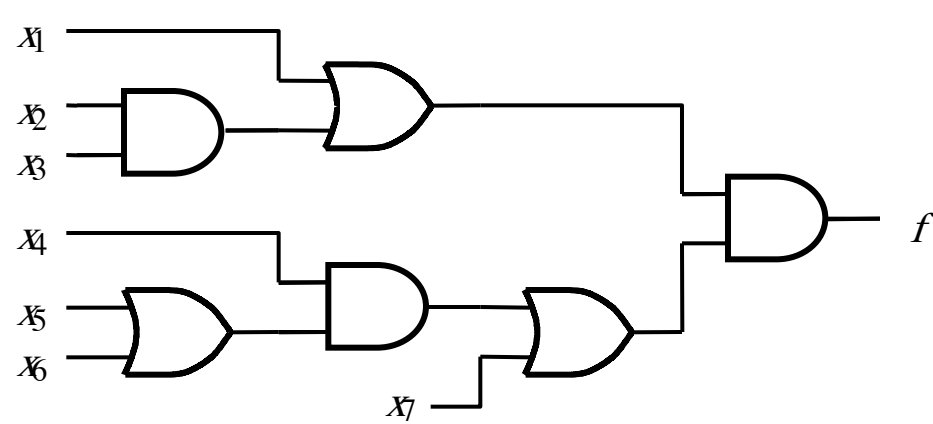
(a) Circuit with AND and OR gates
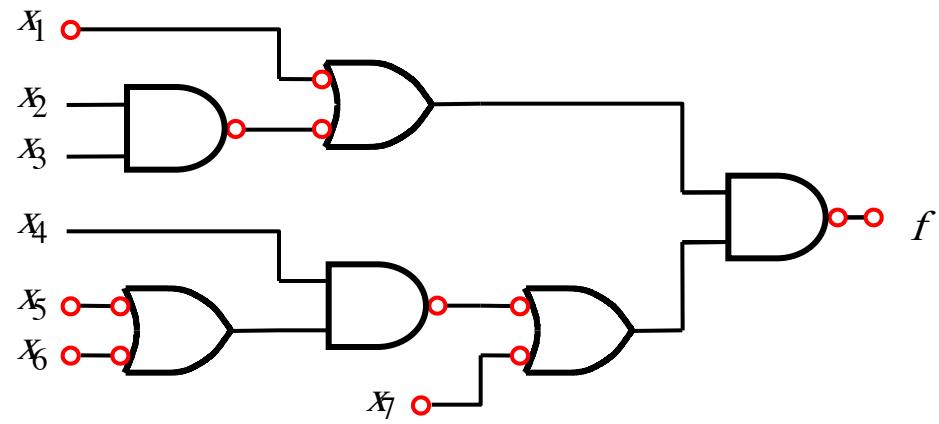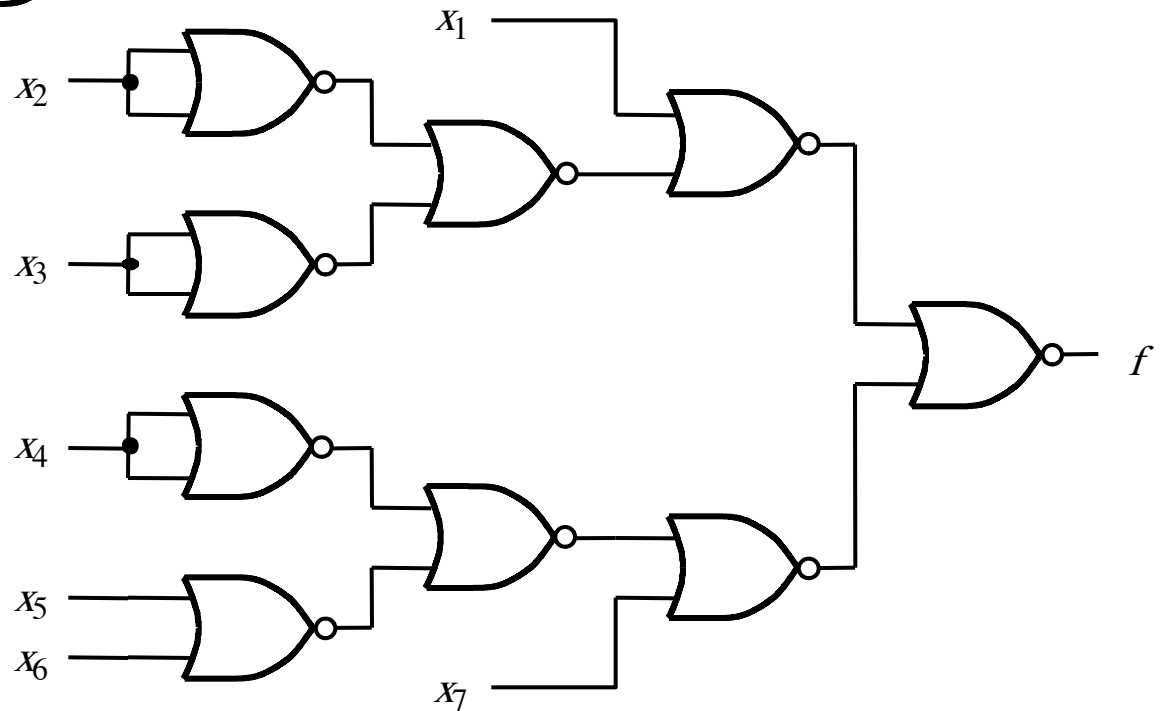
(b) Inversions needed to convert to NANDs
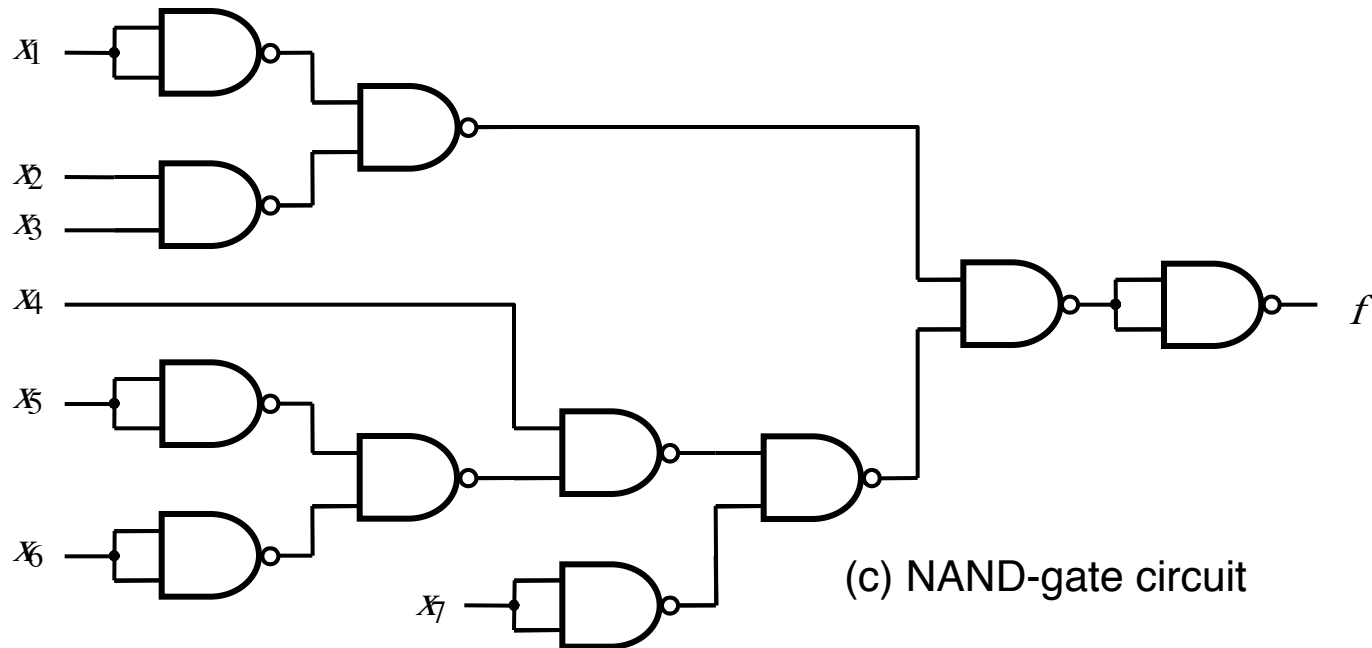
(c) NAND-gate circuit

Equivalent multilevel NOR-gate circuit

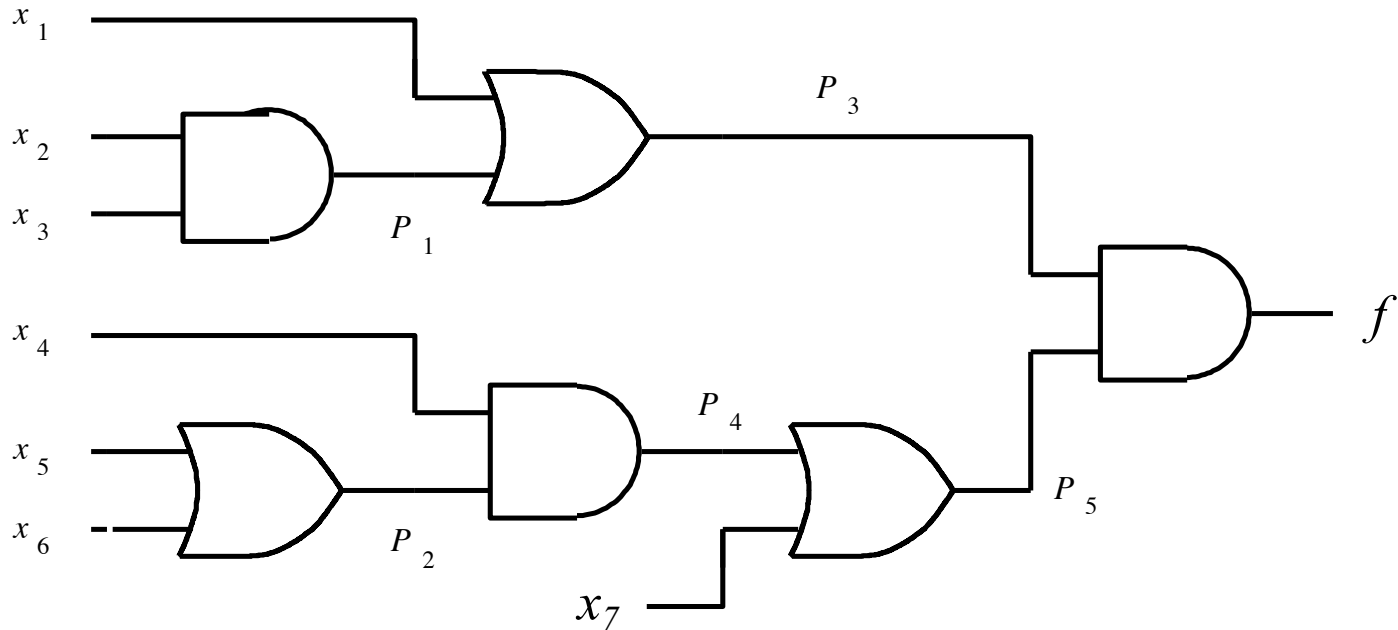(a) Inversions needed to convert to NORs

(b) NOR-gate circuit

Note that this circuit uses less gates and less levels of gates than the NAND-only circuit of L02/S46(c)

Analysis of multilevel circuit

$P_1 = x_2 x_3$

$P_2 = x_5 + x_6$

$P_3 = x_1 + P_1 = x_1 + x_2 x_3$

$P_4 = x_4 P_2 = x_4(x_5 + x_6)$
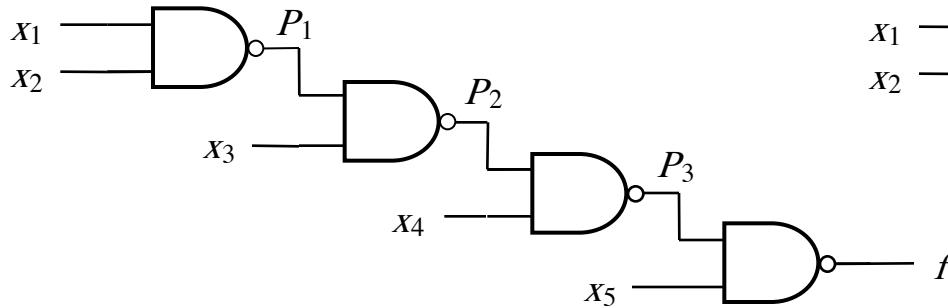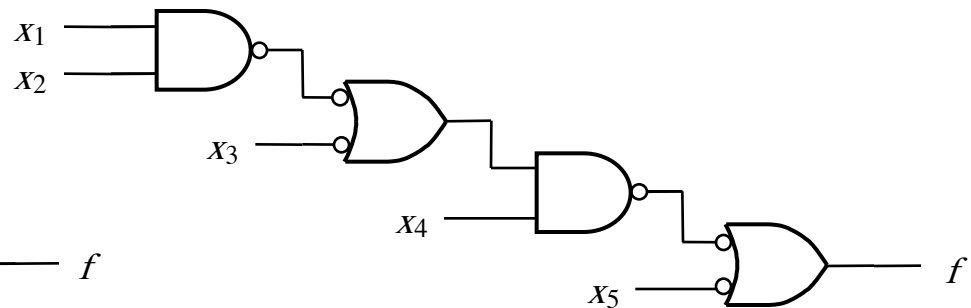
$P_5 = P_4 + x_7 = x_4(x_5 + x_6) + x_7$

$f = P_3 P_5$

$= (x_1 + x_2 x_3)(x_4(x_5 + x_6) + x_7)$

$= x_1 x_4 x_5 + x_1 x_4 x_6 + x_1 x_7 +$

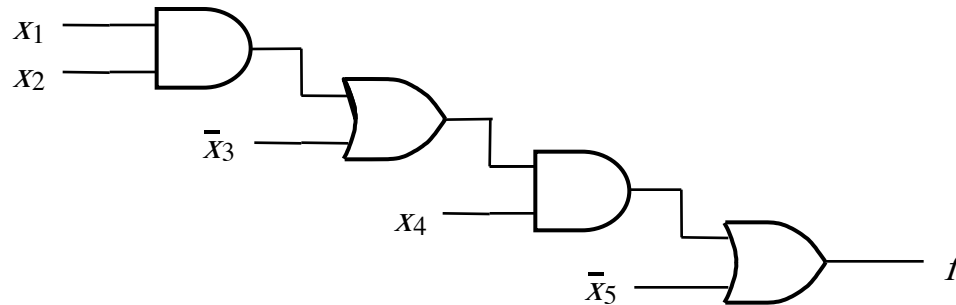$\quad x_2 x_3 x_4 x_5 + x_2 x_3 x_4 x_6 + x_2 x_3 x_7$

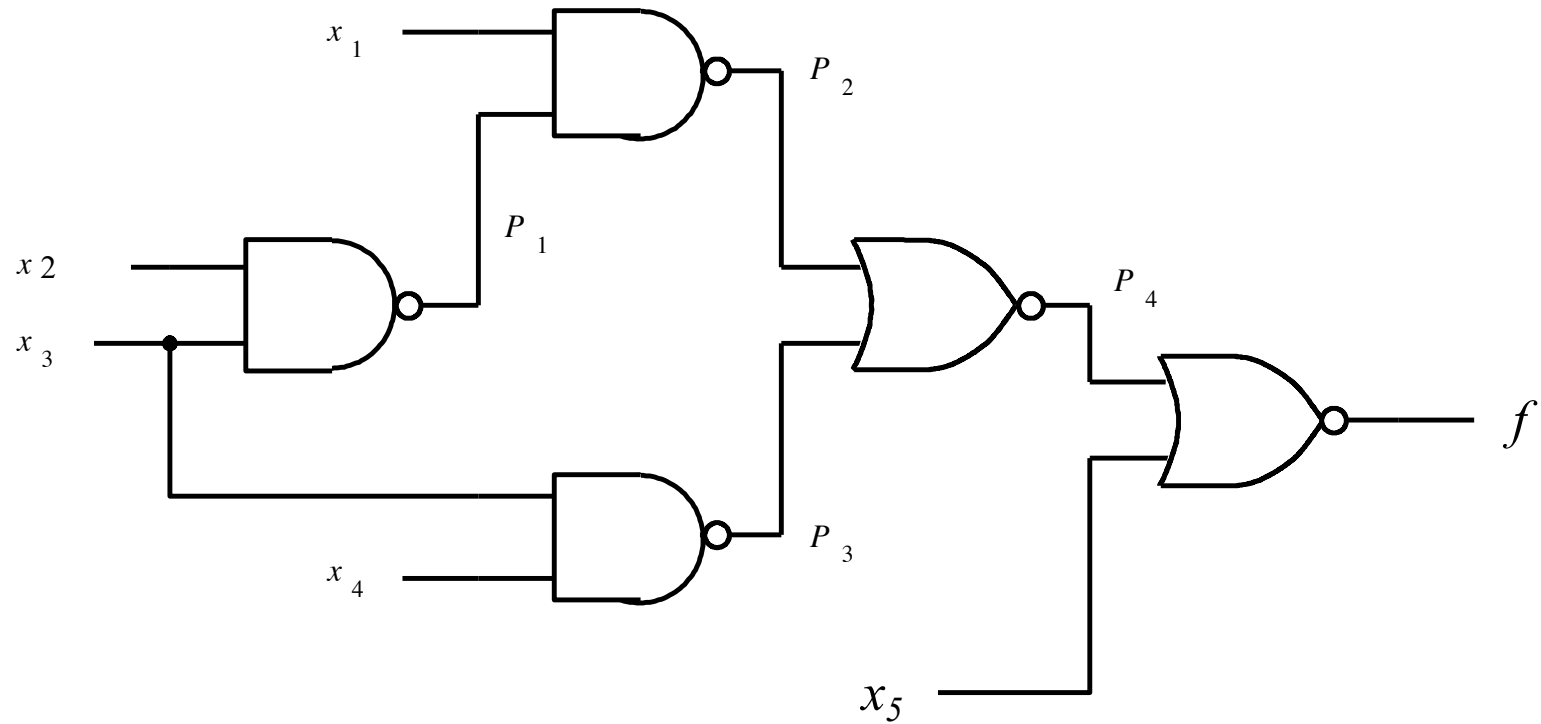# Analyzing multilevel NOR/NAND-only circuits

(a) NAND-gate circuit

(b) Moving bubbles to convert to ANDs and ORs avoids multiple, tedious inversions

(c) Circuit with AND and OR gates

Exercise: What *f* does this circuit implement?

The VHDL code for the function in L02/S3

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY func1 IS
    PORT (x1, x2, x3 : IN   STD_LOGIC ;
              f   : OUT STD_ LOGIC) ;
END func1 ;

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

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
      f <= (NOT x1 AND NOT x2 AND NOT x3) OR
            (NOT x1 AND x2 AND NOT x3) OR
            (x1 AND NOT x2 AND NOT x3) OR
            (x1 AND NOT x2 AND x3) OR
            (x1 AND x2 AND NOT x3) ;
END LogicFunc ;

Synthesizes to
$f = x_1\overline{x_2} + \overline{x_3}$

The VHDL code in **L02/S51** implemented in a 4-LUT

	i_1	i_2	i_3	i_4	
	d	0	0	0	1
0	d	0	0	1	0
x_1	d	0	1	0	1
x_2	d	0	1	1	0
x_3	d	1	0	0	1
	d	1	0	1	1
	d	1	1	0	1
	d	1	1	1	0

i_1
i_2
i_3
i_4

LUT
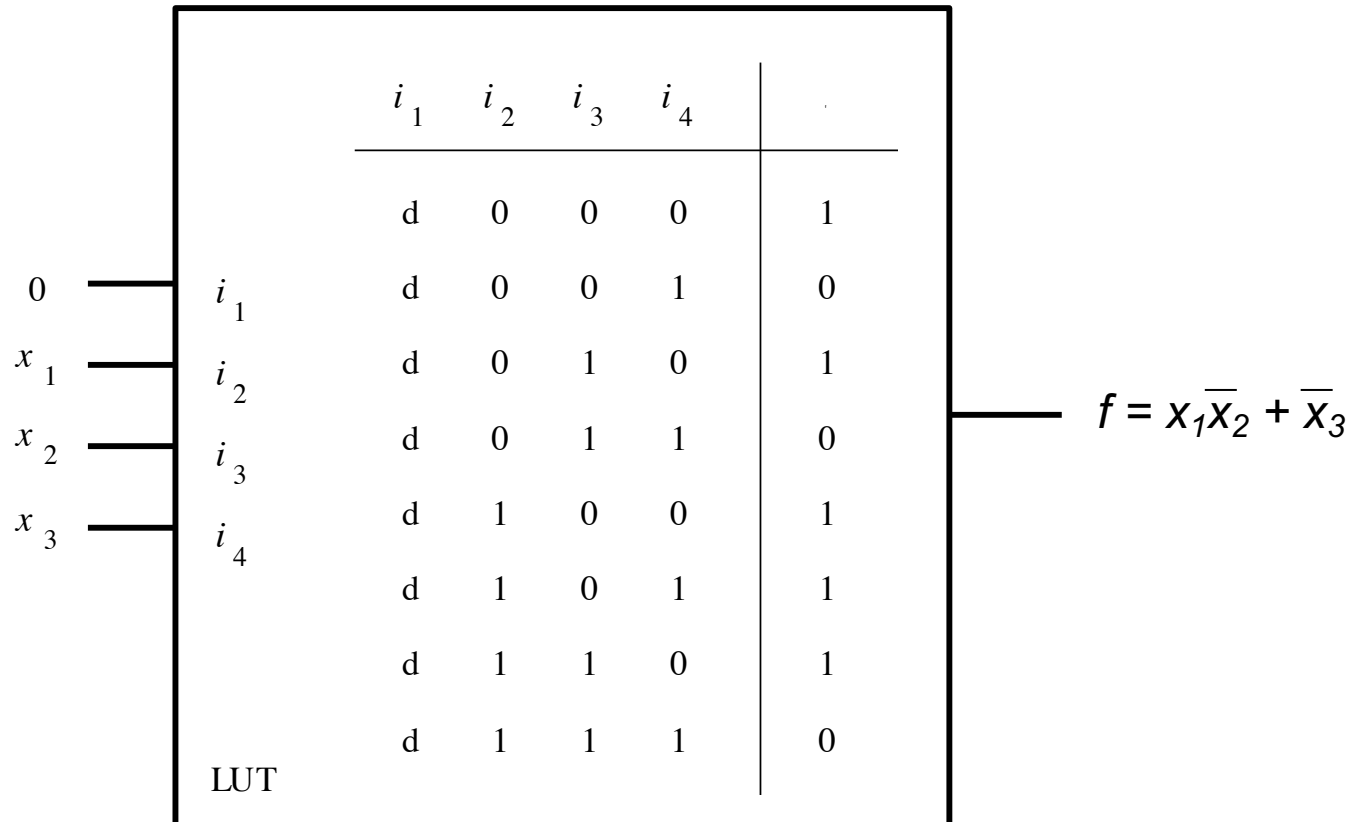
$$f = x_1\overline{x_2} + \overline{x_3}$$

The VHDL code for the function of L02/S24
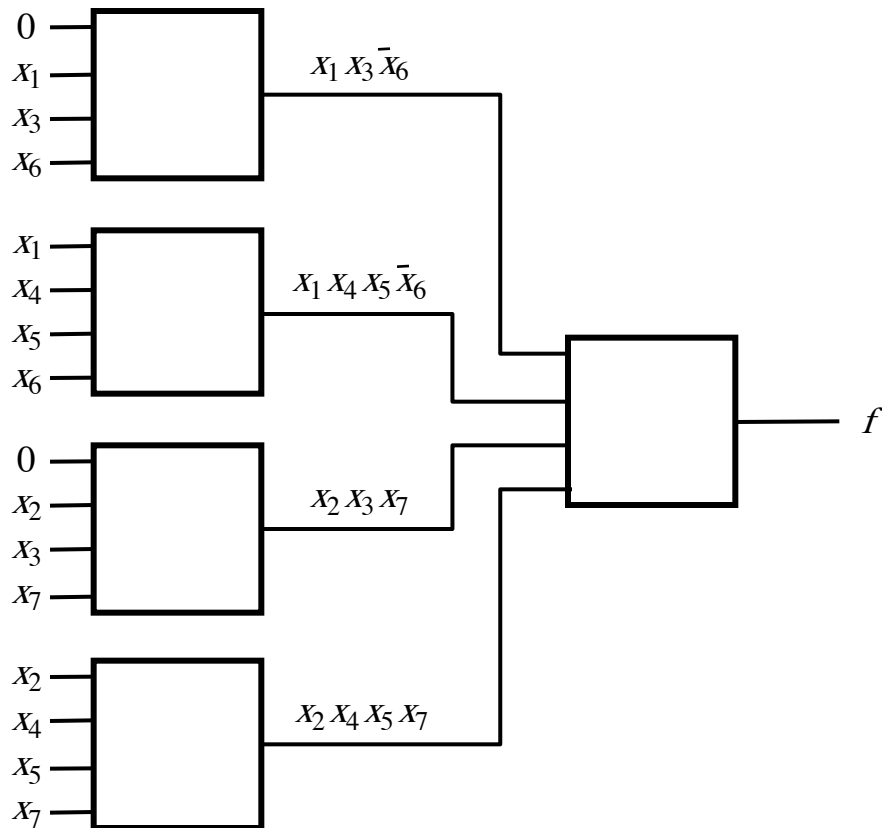
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY func3 IS
    PORT ( x1, x2, x3, x4, x5, x6, x7    : IN    STD_LOGIC;
                 f                               : OUT  STD_LOGIC);
END func3;

ARCHITECTURE LogicFunc OF func3 IS
BEGIN
    f <= (x1 AND x3 AND NOT x6) OR
          (x1 AND x4 AND x5 AND NOT x6) OR
          (x2 AND x3 AND x7) OR
          (x2 AND x4 AND x5 AND x7);
END LogicFunc;
```
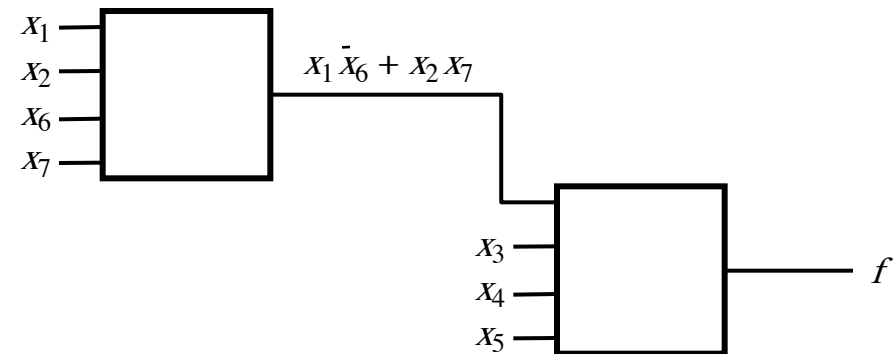
Synthesizes as
$f = (x_1\overline{x}_6 + x_2x_7)(x_3 + x_4x_5)$

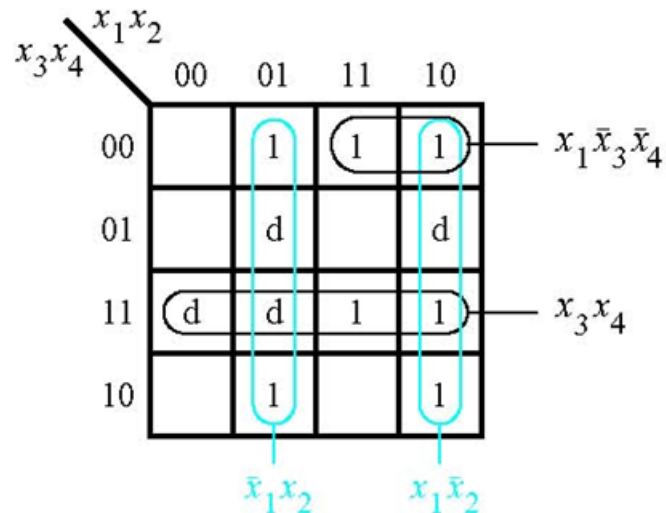# Implementation of the VHDL code in L02/S53
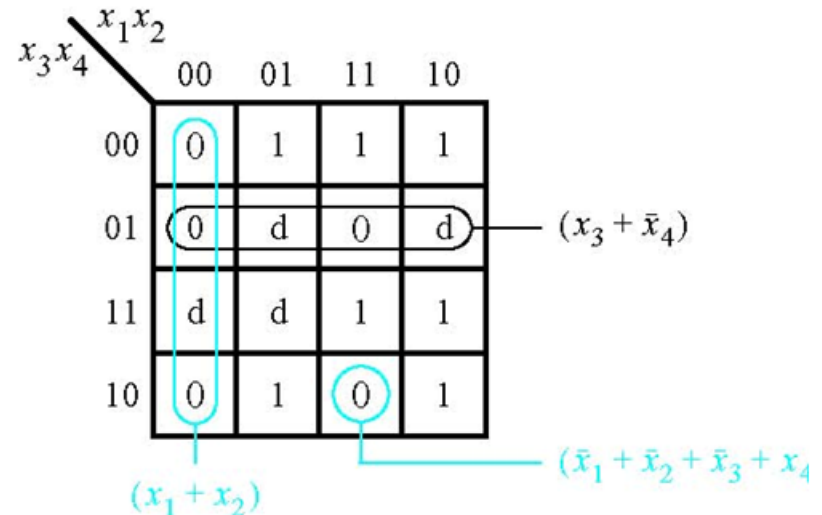

(a) Sum-of-products realization requires 5 4-LUTs

(b) Factored realization uses 2 4-LUTs

Exercise:

- Determine the minimum-cost SOP and POS expressions for the function $f(x_1,x_2,x_3,x_4) = \Sigma m(4,6,8,10,11,12,15) + D(3,5,7,9)$

(a) Determination of the SOP expression
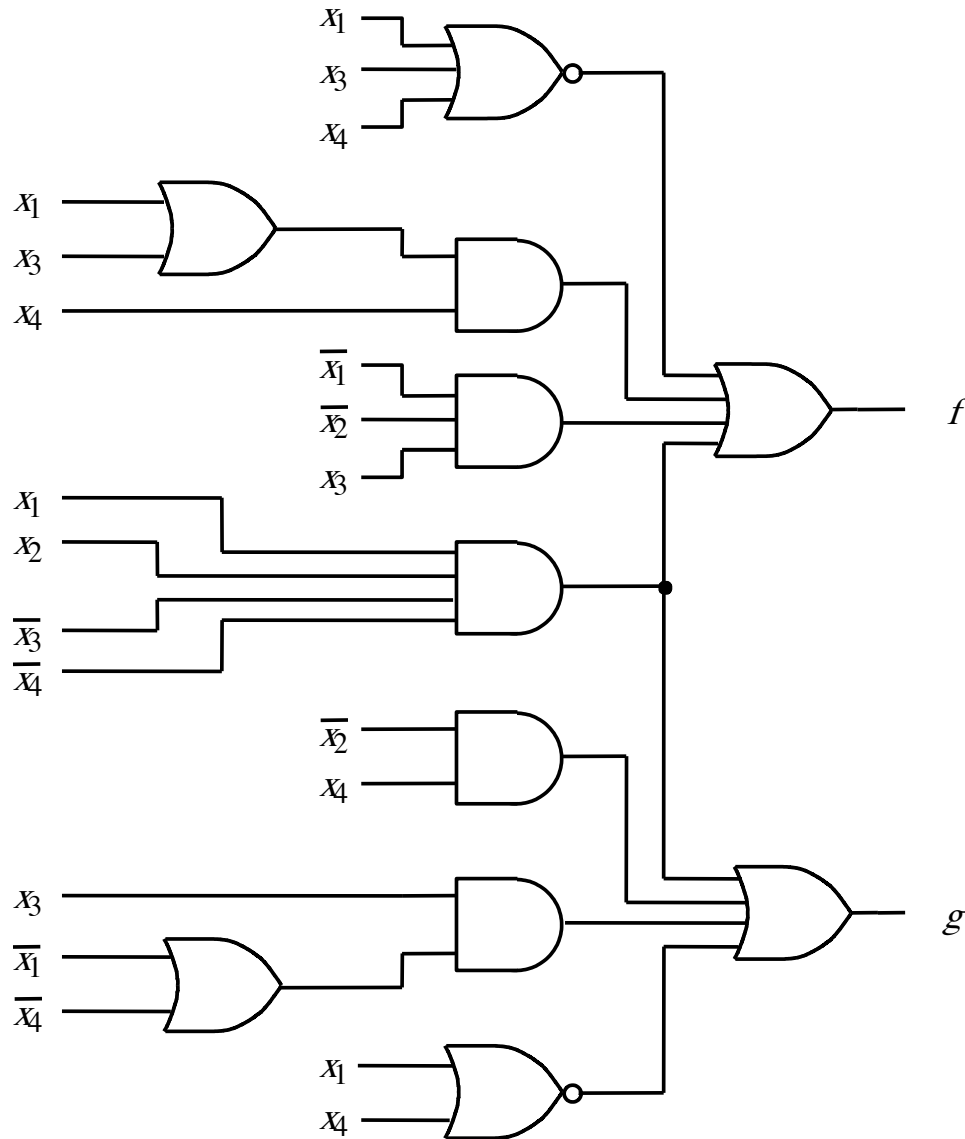
(b) Determination of the POS expression

Exercise:

- Use Karnaugh maps to find the minimum-cost SOP and POS expressions for the function

  $f(x_1,..x_4) = \overline{x}_1\overline{x}_3\overline{x}_4 + x_3x_4 + x_1x_2x_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4$

  assuming don't cares are defined as D = Σ(9,12,14)

Circuit for problem 4.43

What is the cost of this circuit, assuming variables are available in complemented and un-complemented forms?

Re-implement the circuit at the lowest possible cost. (Can you beat 33?)