COMP3222/9223 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) = Σ 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

$f = m_0 + m_2 + m_4 + m_5 + m_6$

$= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3$

$= \bar{x}_1\bar{x}_3 + x_1\bar{x}_3 + x_1\bar{x}_2$

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

The VHDL code for the function in L02/S3

```
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 ;
```

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

Should synthesize 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*

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

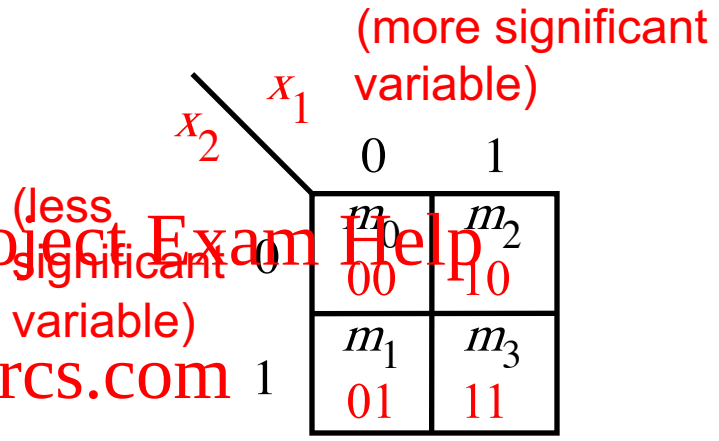
as judiciously as possible.

- 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



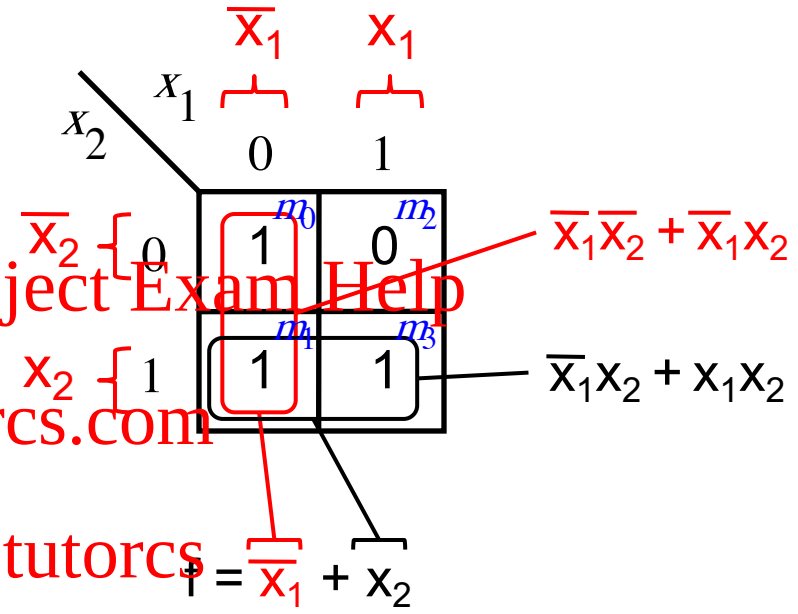
(b) Karnaugh map

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

This facilitates their combination.

The function of L01/S26

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

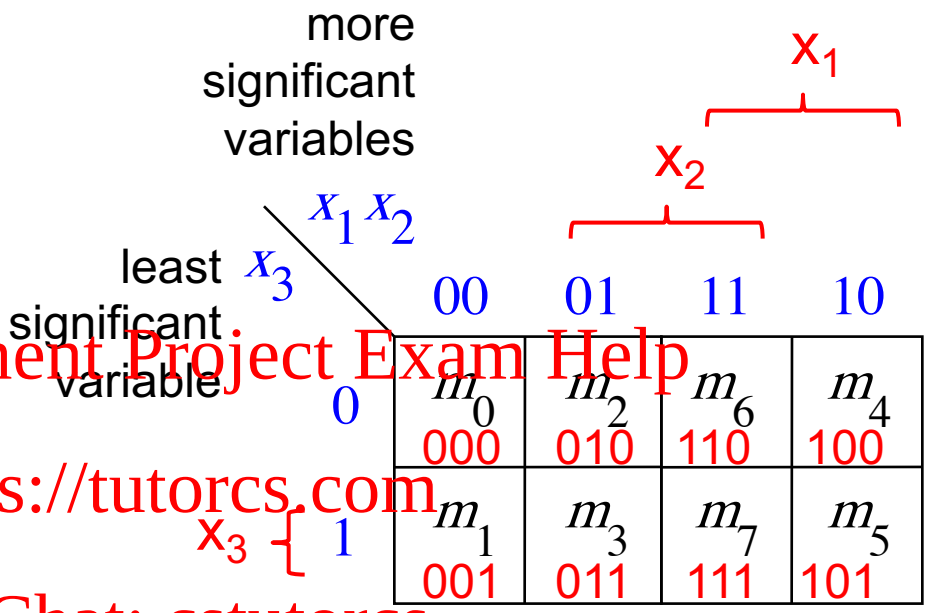


Note that adjacent cells in a column or row of the Karnaugh map can be combined in order to eliminate the variable that is present in both uncomplemented and complemented form

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

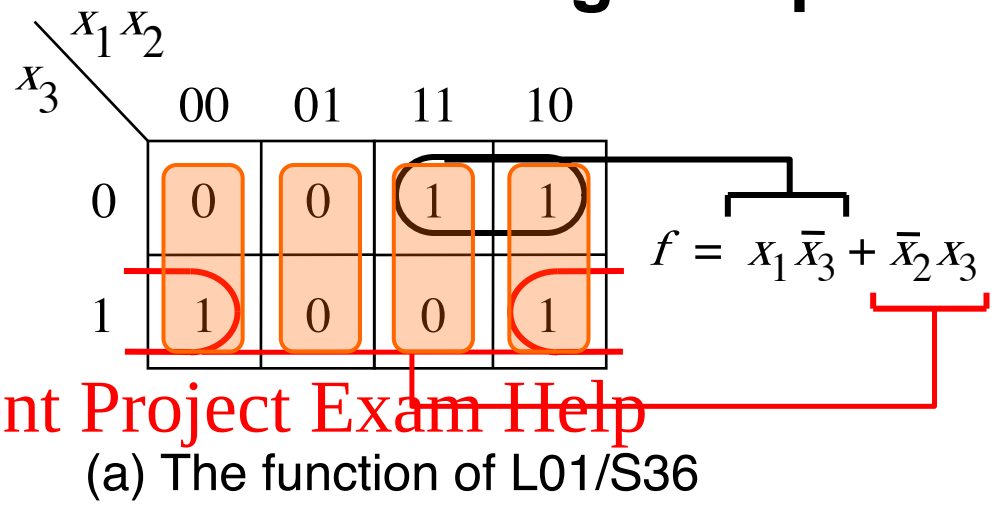


(b) Karnaugh map

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

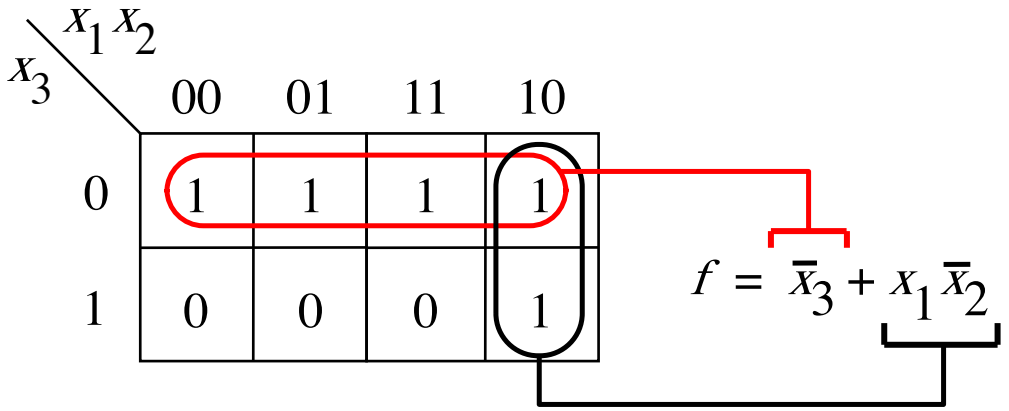
Examples of three-variable Karnaugh maps

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



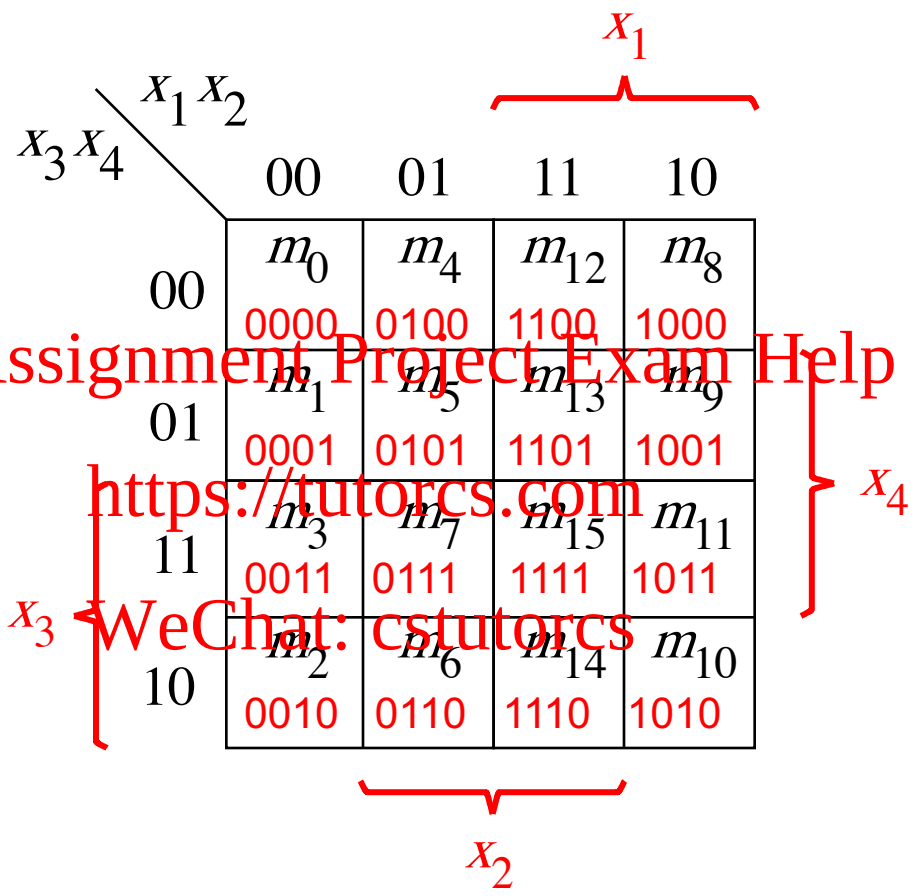
(a) The function of L01/S36

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



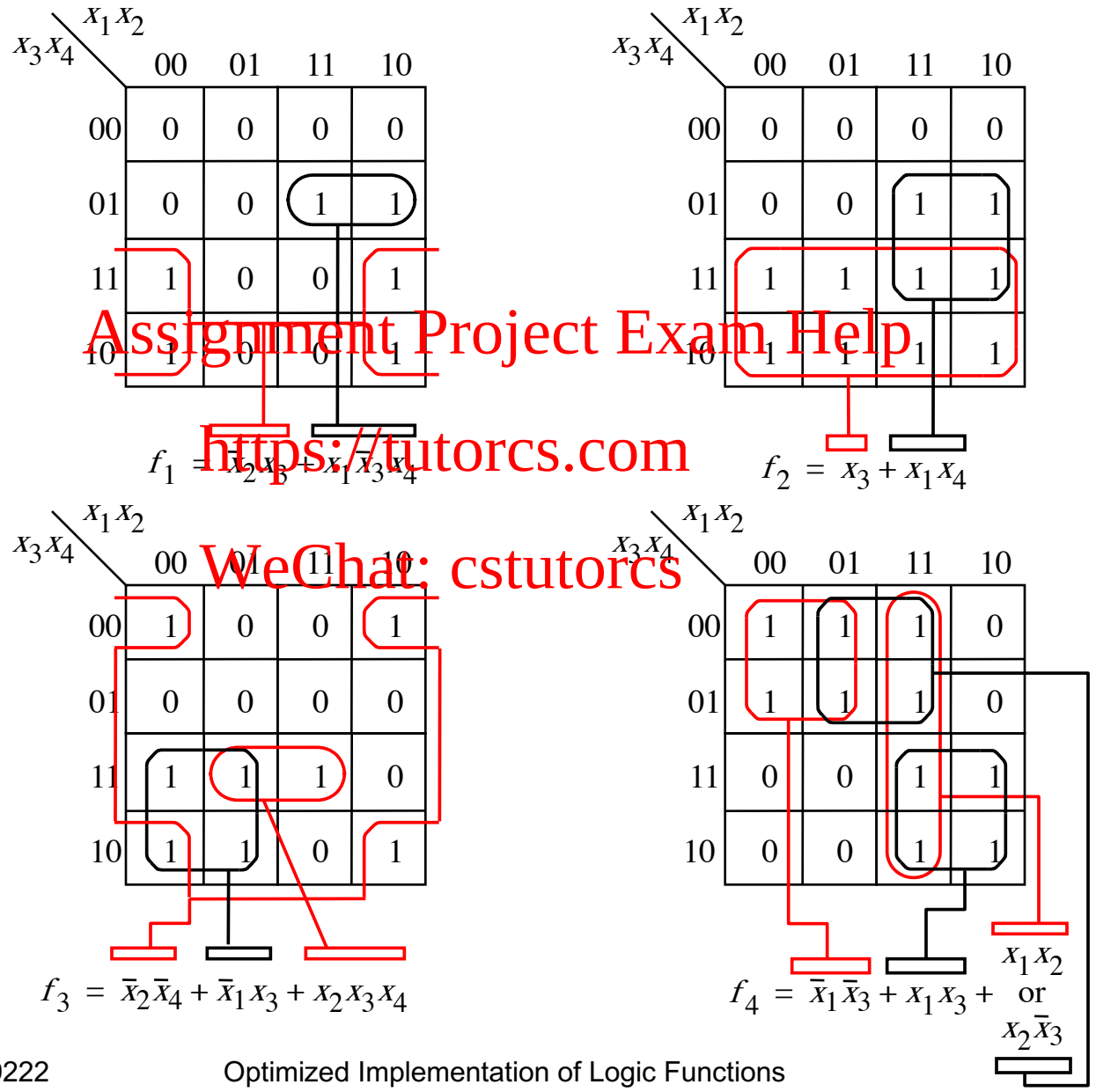
(b) The function of L02/S3

A four-variable Karnaugh map

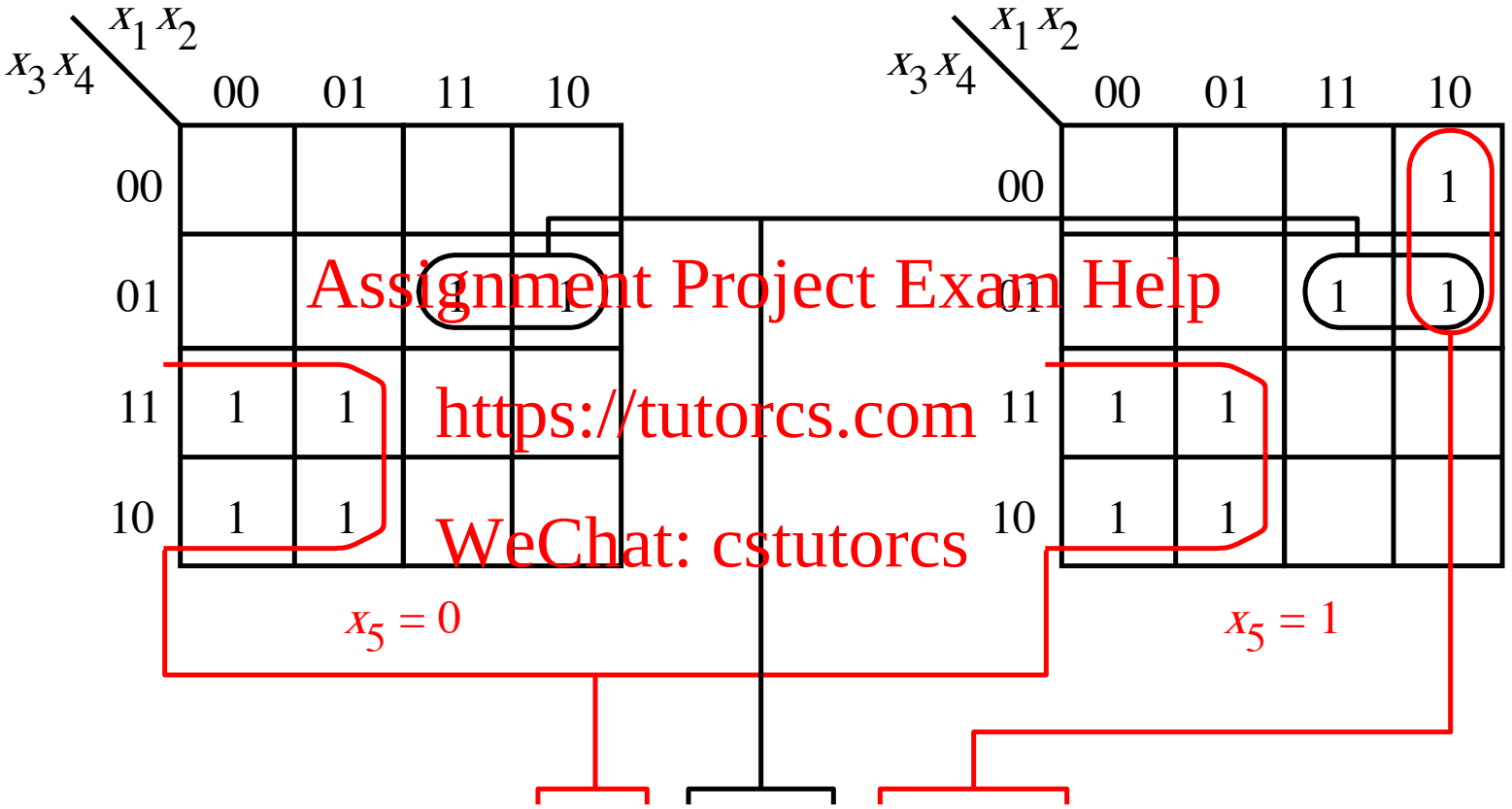


- Note that both row and col addresses are Gray coded
- Note that opposite edges of the map are considered adjacent

Examples of four-variable Karnaugh maps

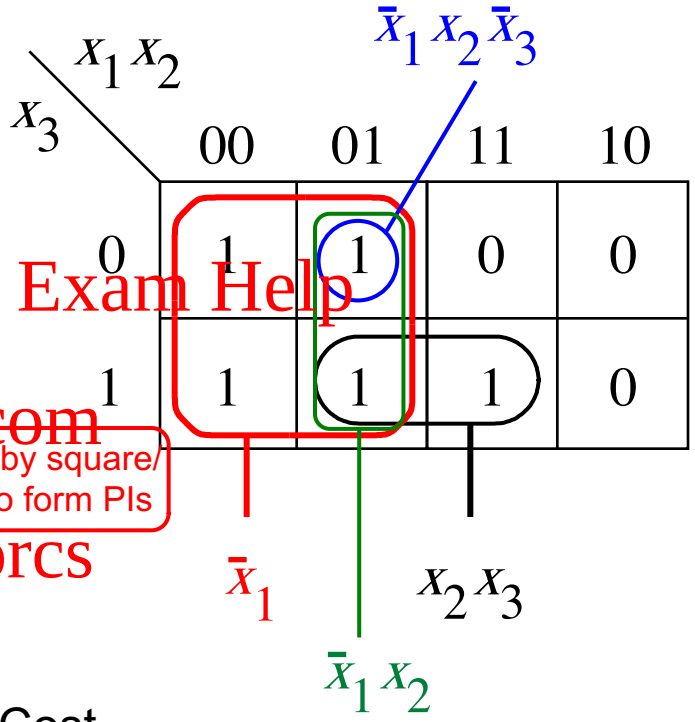


A five-variable Karnaugh map



Minimization strategy

What is the minimum cost of this function?



Implicants are expanded by square/ rectangular powers of 2 to form PIs

Terminology

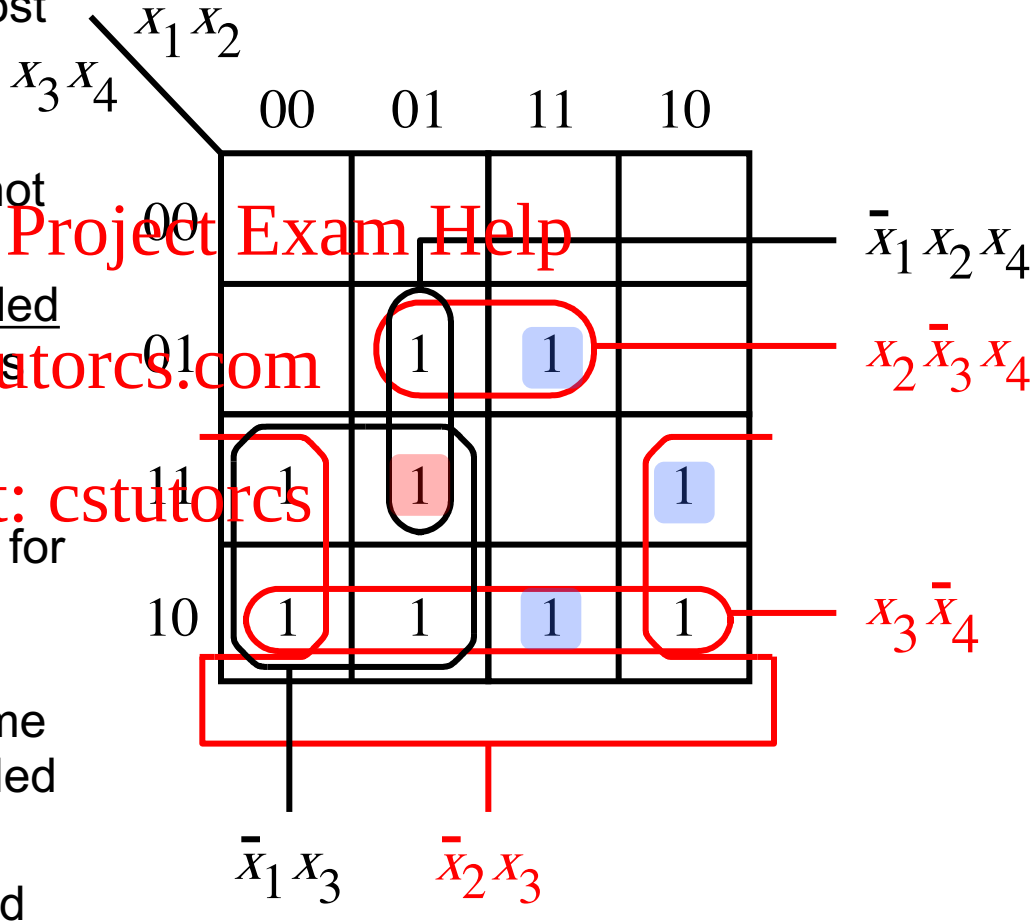
- Literal
  - Each appearance of a variable in a product term
- Implicant
  - A product term that indicates a set of valuations for which a given function is equal to 1
- Prime implicant
  - Cannot be combined into another implicant that has fewer literals
- Cover
  - A collection of implicants that 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

- 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

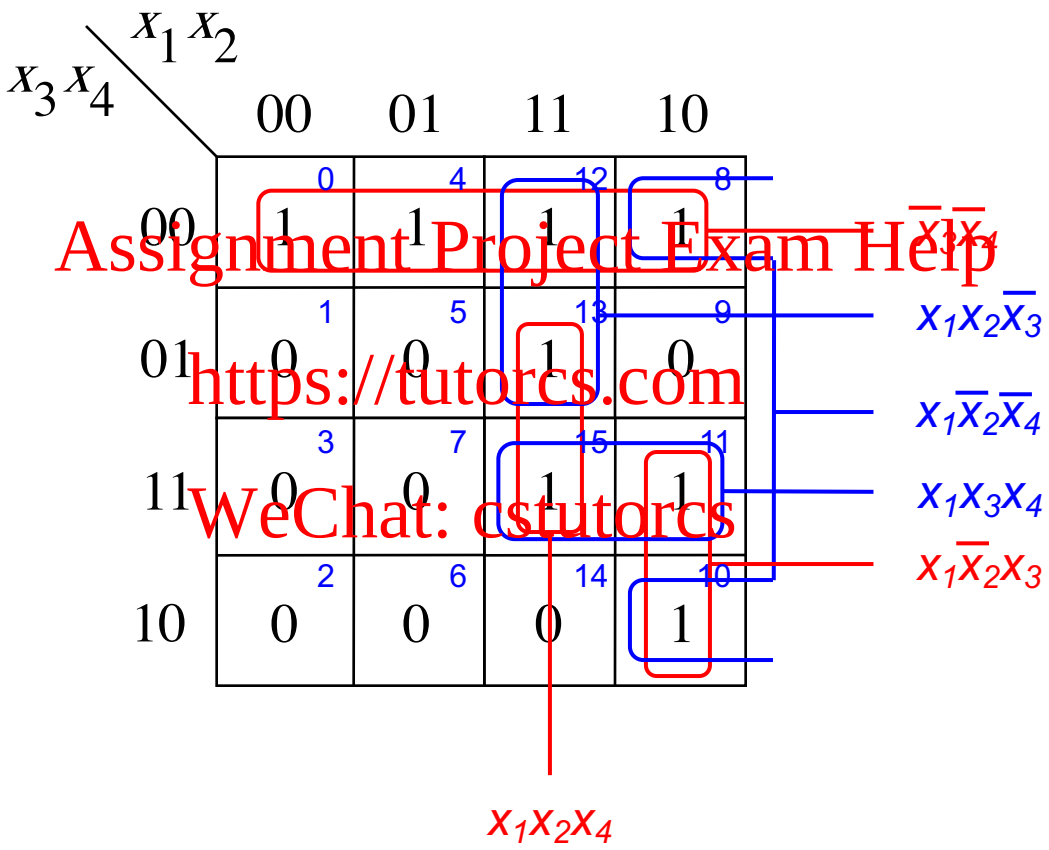
- Where there is choice as to which prime implicants to use in a cover, how do we find the minimum cost subset of prime implicants?
- If a prime implicant includes a minterm for which *f = 1* that is not included in any other prime implicant, then it must be included in the cover and is referred to as 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, $\overline{x}_1x_3$ is preferred over $\overline{x}_1x_2x_4$ for covering $\overline{x}_1x_2x_3x_4$ since it has lower cost

What is the minimum cost of this function?



When there is a choice of covers….

$f(x1,\ldots,x4) = \Sigma\, m(0, 4, 8, 10, 11, 12, 13, 15)$

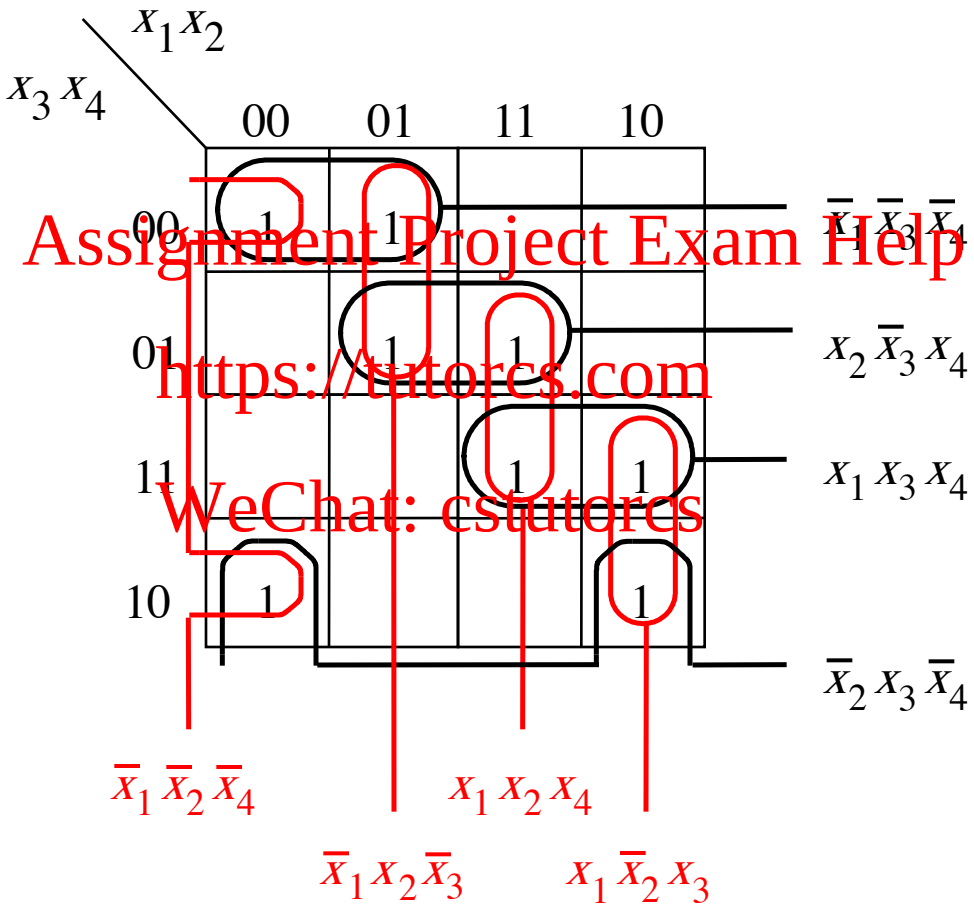


Heuristic choice of nonessential prime implicants

- 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;*
  - *Then, determine the cost of a cover that does not include the chosen prime implicant; and*
  - *Select for implementation the cover which costs least.*

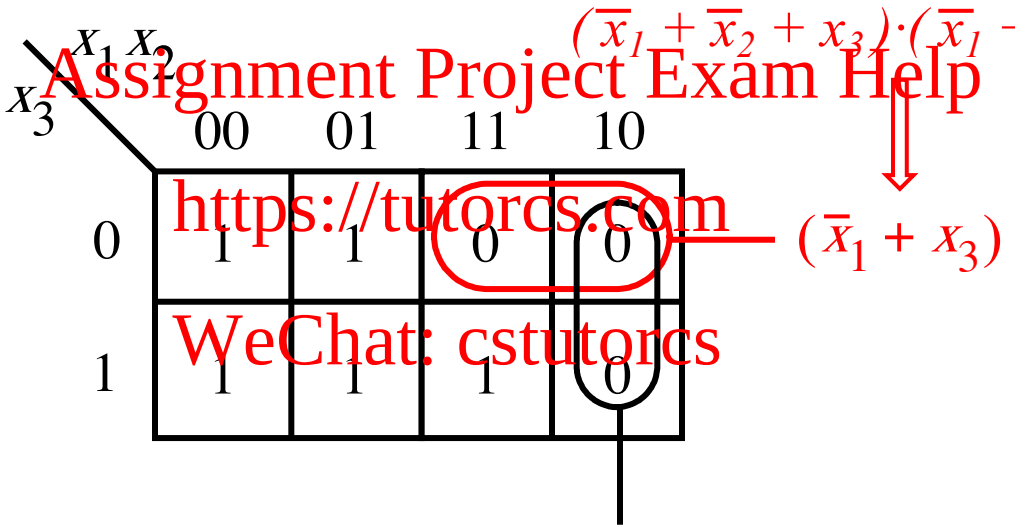
Sometimes alternative covers cost the same…

$f(x1,\ldots,x4) = \Sigma\, m(0, 2, 4, 5, 10, 11, 13, 15)$



POS minimization of $f(x1, x2, x3) = \Pi M(4, 5, 6)$

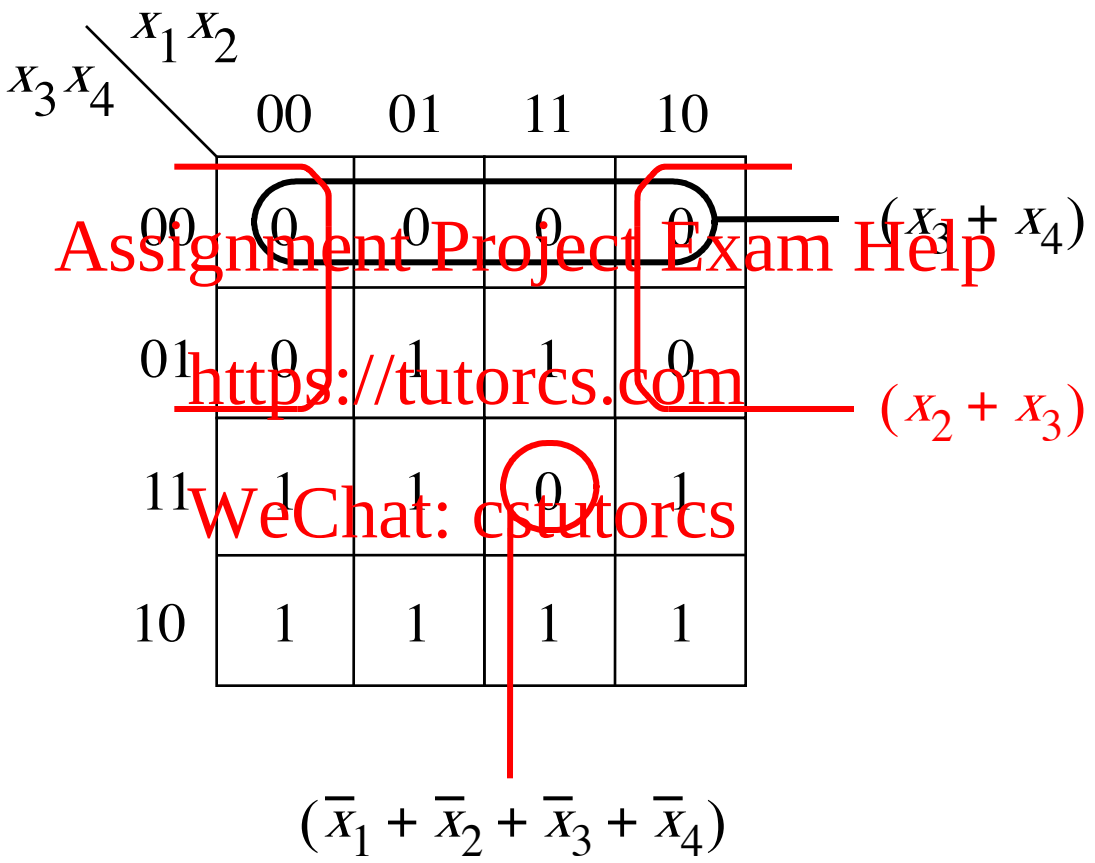
- Minimal SOP (sum of product) and POS (product of sum) implementations need to be compared to determine the least cost realization



$$(\overline{x}_1 + x_2 + x_3)\cdot(\overline{x}_1 + x_2 + \overline{x}_3) \Rightarrow (\overline{x}_1 + x_2)$$

What is the cost of this function?
Which costs less: SOP or POS?

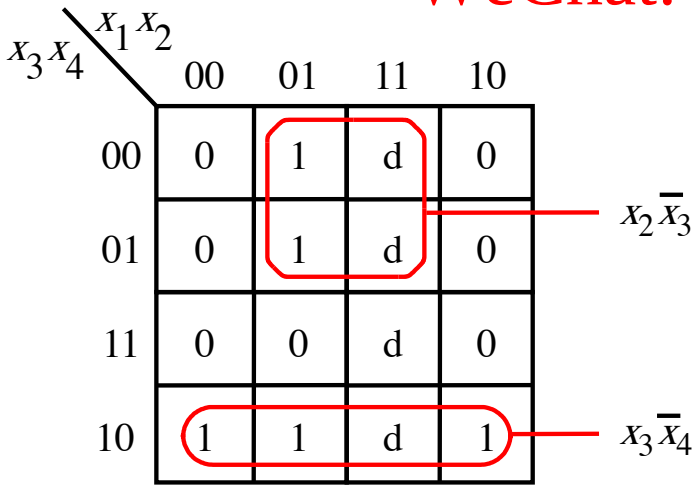
POS minimization of
$f(x1,\ldots,x4) = \Pi M(0, 1, 4, 8, 9, 12, 15)$



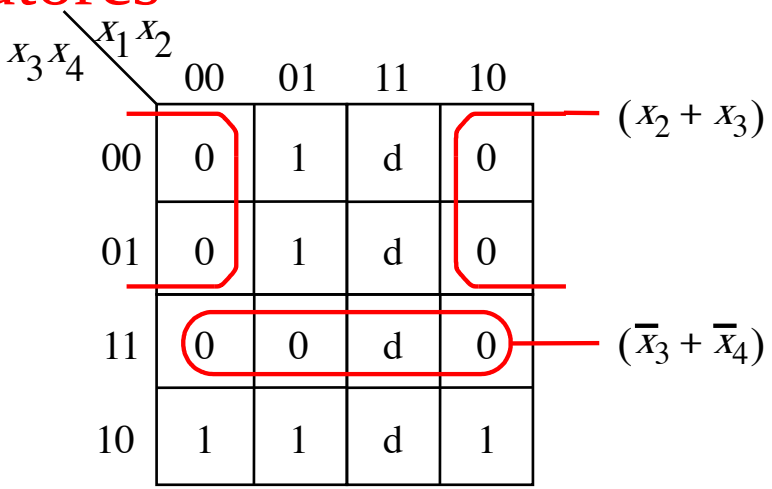
- Which implementation is cheaper: POS or SOP?

Practical Reality 1: Incompletely specified functions

- Often functions are not completely specified
  - The outputs under certain input conditions are not given
    ⇒ we can treat them as "don't care" values
- Don't care conditions offer additional opportunities for simplification
  - A don't care can be assigned the value 0 or 1, as desired
  - E.g., there are two possible implementations of the function
    f (x1,…, x4) = Σ m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)

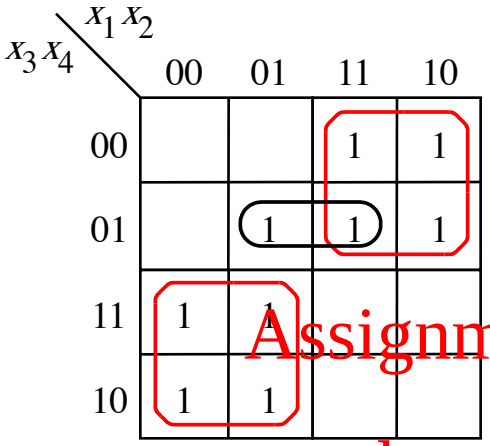


(a) SOP implementation

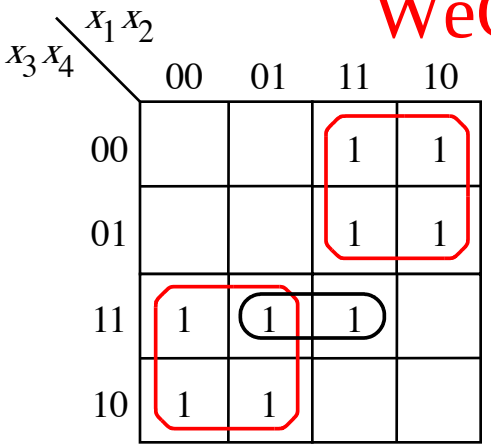


(b) POS implementation

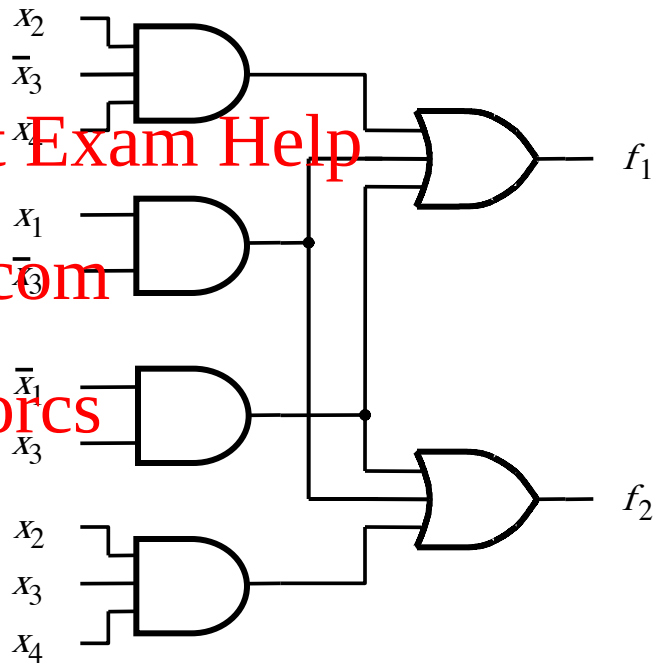
Practical Reality 2: Multiple-output synthesis



(a) Function f_1

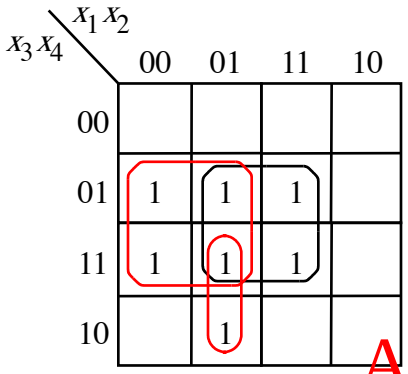


(b) Function f_2

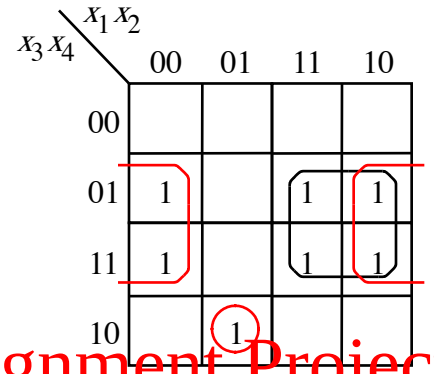


(c) Combined circuit for f_1 and f_2

Another example of multiple-output synthesis

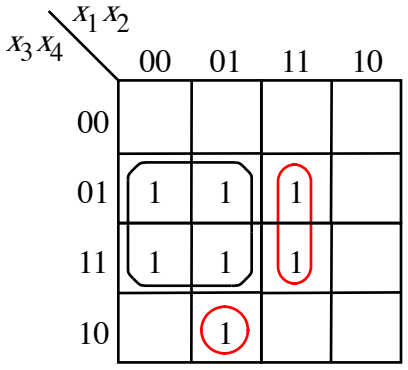


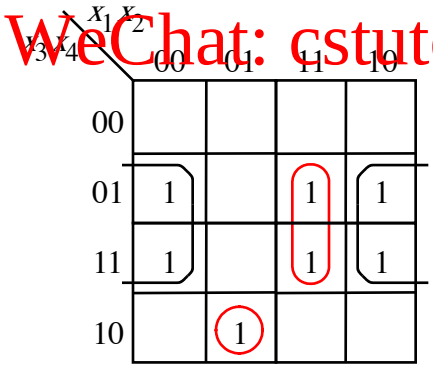
(a) Optimal realization of f_3
Cost = 2x2-in + 2x3-in = 14



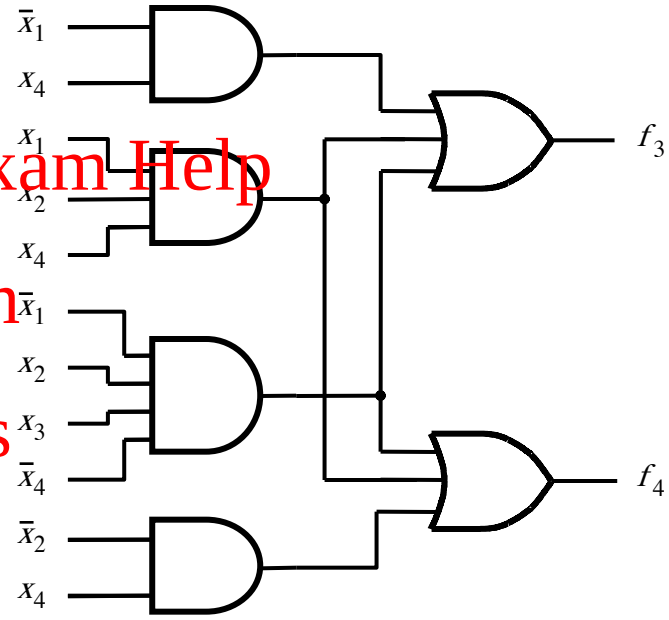
(b) Optimal realization of f_4
Cost = 2x2-in + 1x4-in + 1x3-in = 15

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

$f(x_1,\ldots,x_7) = x_1x_3\overline{x}_6 + x_1x_4x_5\overline{x}_6 + x_2x_3x_7 + x_2x_4x_5x_7$

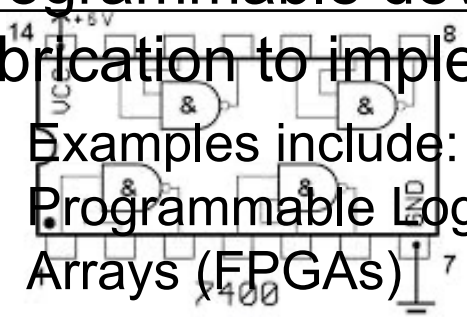
$= x_1\overline{x}_6(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5)$

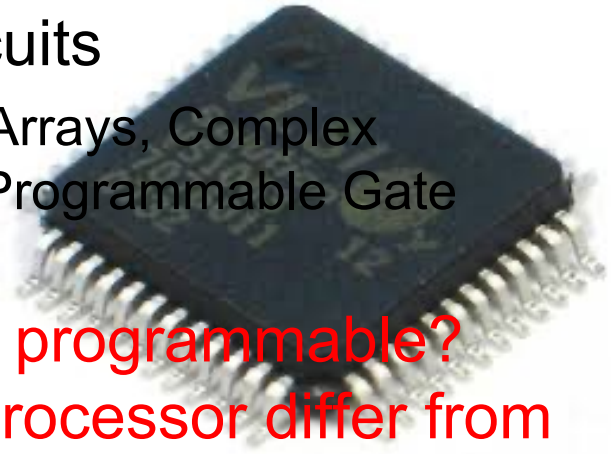
$= (x_1\overline{x}_6 + x_2x_7)(x_3 + x_4x_5)$ … *by the distributive law*

- Electrical properties limit the number of inputs a logic gate can have before performance is compromised
- Similarly, manufacturing constraints may impose an upper limit on the number of inputs to a logic resource
  - Factoring a circuit or function 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 is restricted to 2-input gates?
  - The result is a multilevel (more than two levels of gates) function/circuit implementation

Implementation technologies: A quick introduction to FPGAs

- Digital circuits are implemented in various solid state/silicon device technologies
- Devices are 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 requisite circuits
  - Examples include: Programmable Logic Arrays, Complex Programmable Logic Devices and Field-Programmable Gate Arrays (FPGAs)





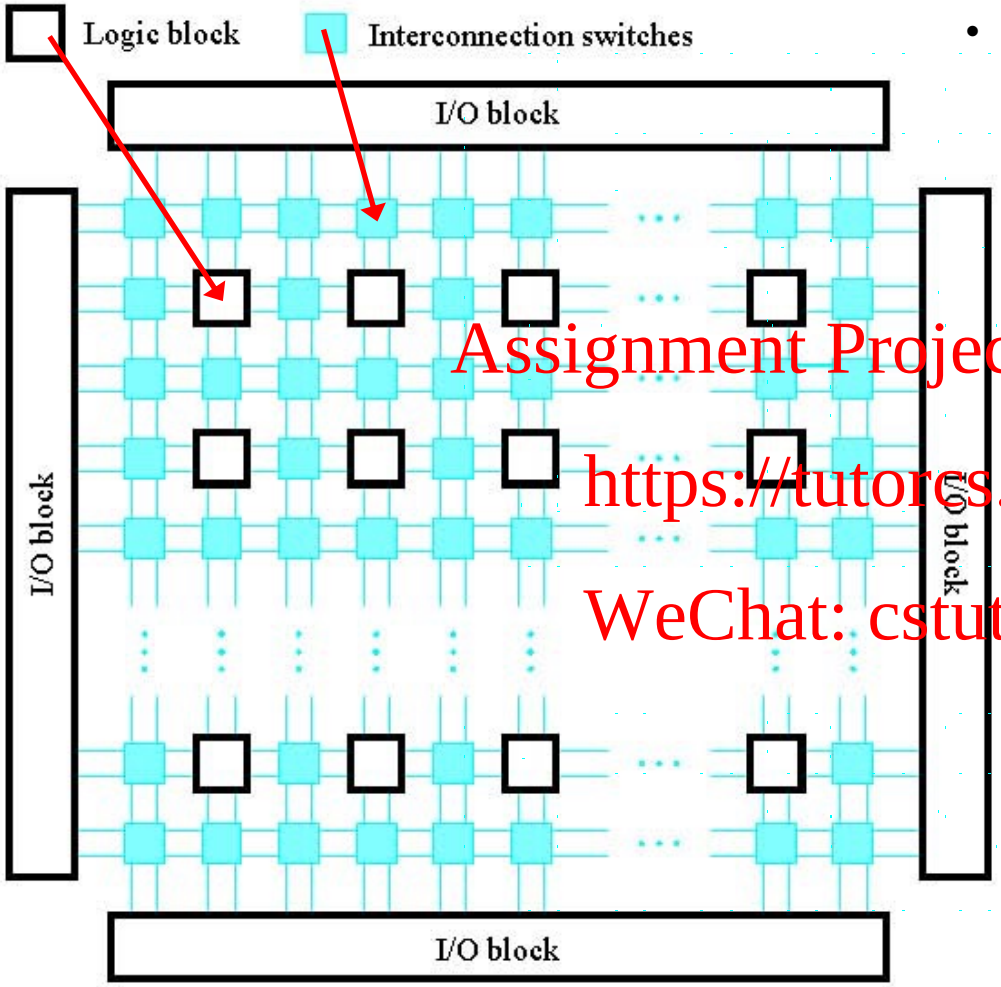
Q: Is an ARM or Intel processor fixed or programmable?

Q: How does the programmability of a processor differ from that of an FPGA?

7400 Quad 2-NAND (wikipedia.com)

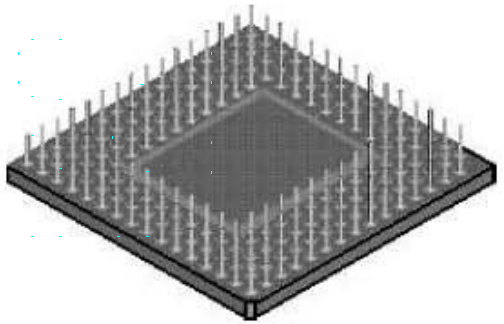
MP3 encoder (eenewseurope.com)

A Field-Programmable Gate Array (FPGA)



(a) General structure of an FPGA

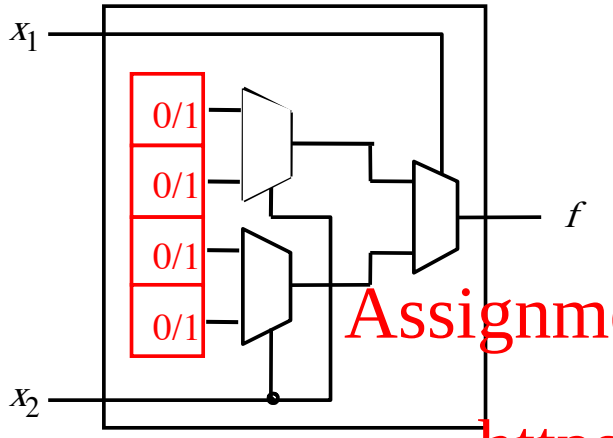
- For cost and performance reasons, as few chips as possible are used to implement a design
  - 7400-series chips implement the equivalent of just a few two-input NAND gates (the prevalent metric for chip size)
  - An SPLD or CPLD macrocell represents about 20 equivalent gates; thus a PAL with 8 macrocells 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) Pin grid array (PGA) package (bottom view)

B&V3, Figure 3.35

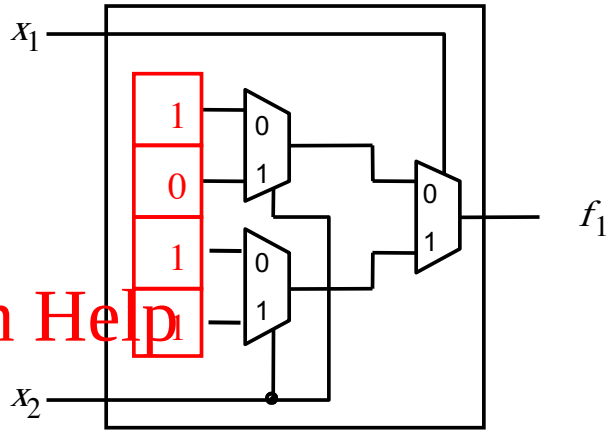
A two-input lookup table (LUT) as an FPGA logic cell/block



(a) Circuit for a two-input LUT

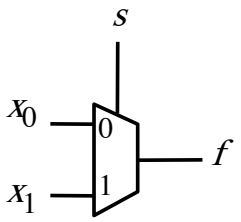
x_1	x_2	f_1
0	0	1
0	1	0
1	0	1
1	1	1

(b) $f_1 = x_1 + \overline{x_2}$



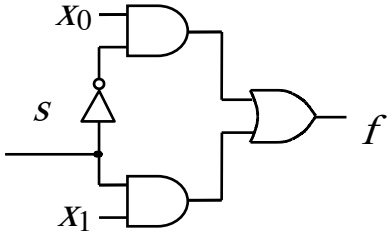
(c) Storage cell contents in the LUT

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

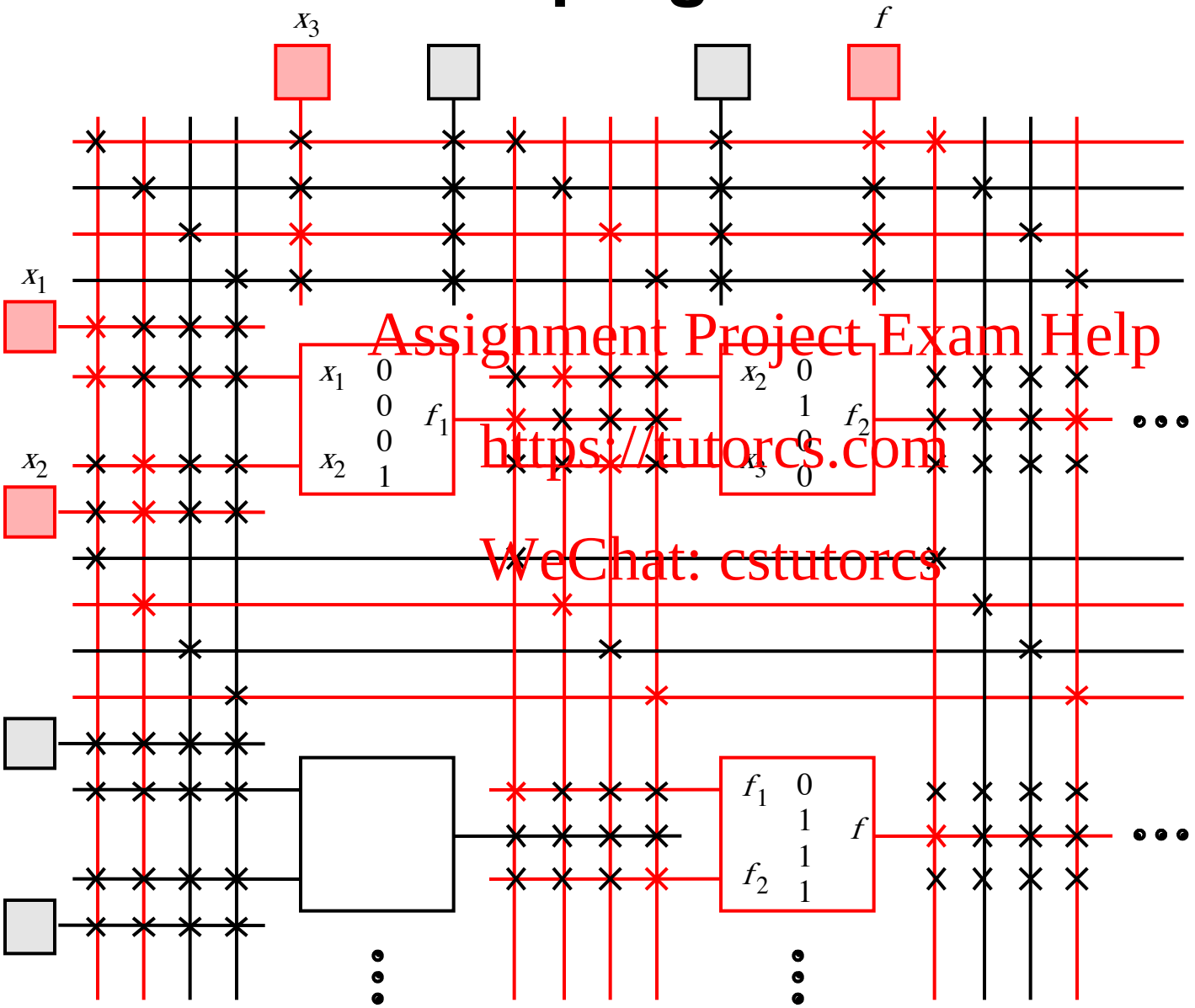
A two-input multiplexer (MUX) $f = \overline{s}.x_0 + s.x_1$ selects one of its two inputs to be routed to the output



Two-input multiplexer circuit (2-MUX)

B&V3, Figure 3.36

A section of a programmed FPGA

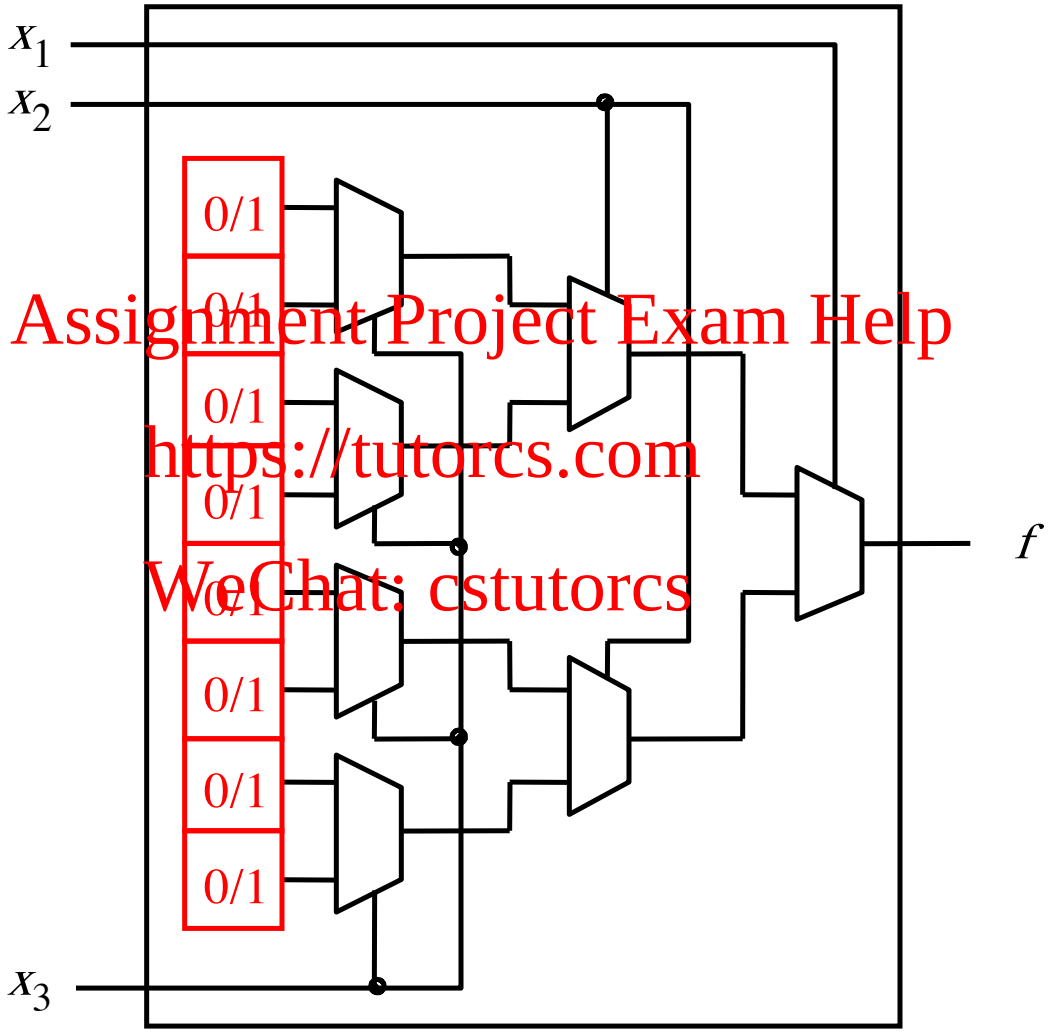


$f_1 = x_1x_2$
$f_2 = \overline{x}_2x_3$
$f = f_1 + f_2$
$= x_1x_2 + \overline{x}_2x_3$

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

B&V3, Figure 3.39

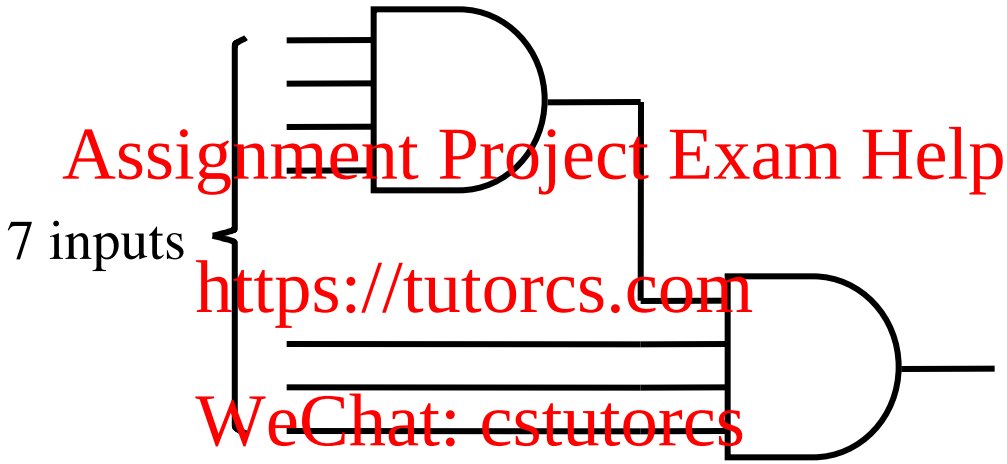
A three-input LUT



B&V3, Figure 3.37

Realizing a seven-input product term

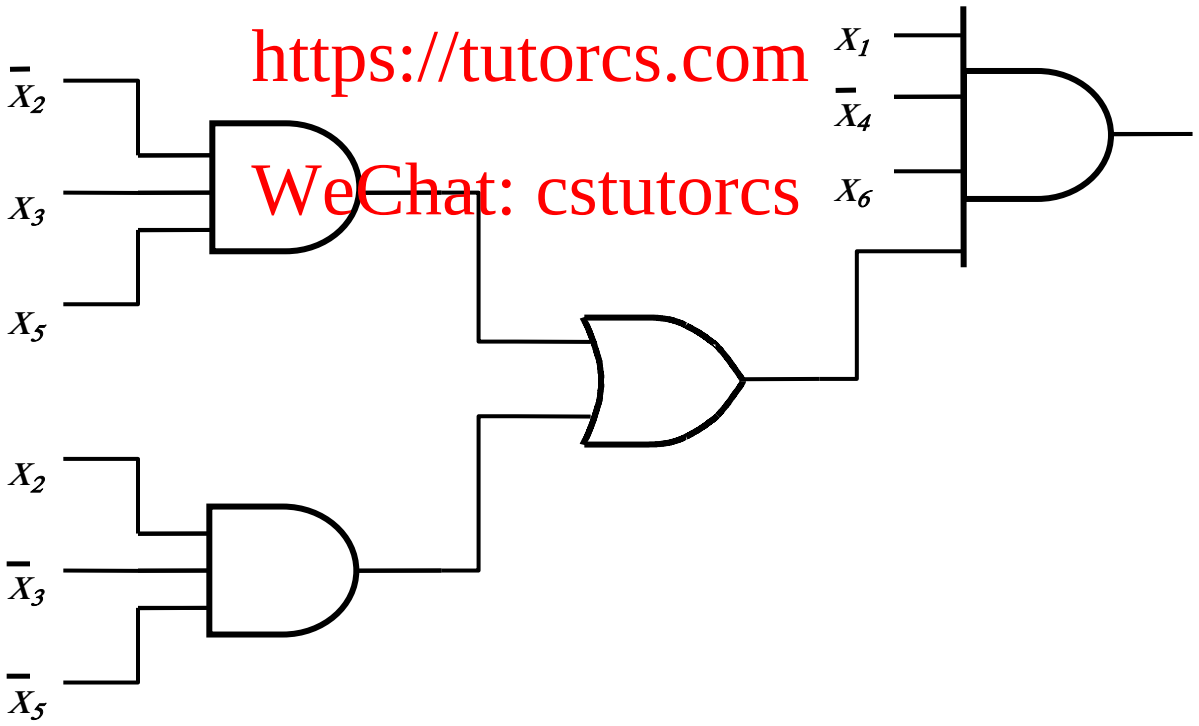
- Implementing wide functions when gate fan-in is limited



Still Practical Reality 3: A factored circuit

- Factoring can be used to overcome fan-in constraints
- For example, $f = x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6$

  $= x_1\overline{x}_4x_6(\overline{x}_2x_3x_5 + x_2\overline{x}_3\overline{x}_5)$ for FO4 (fan-in of 4) gates



Practical Tradeoff: Functional decomposition

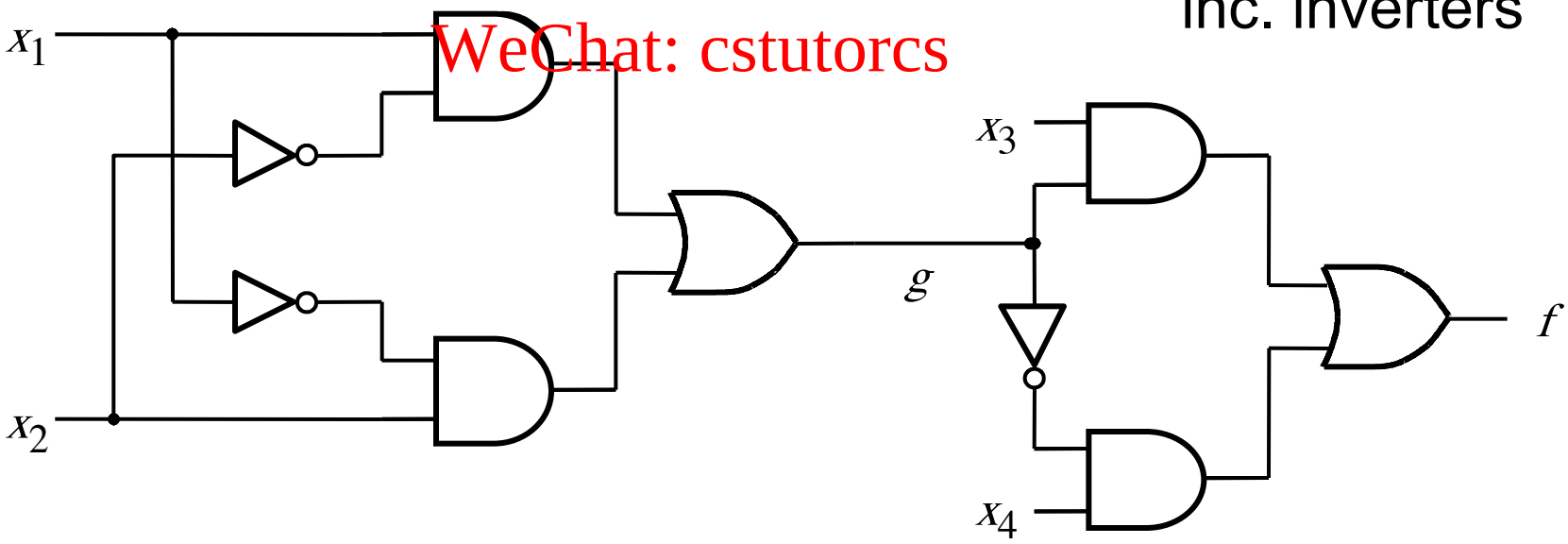
- Multilevel realizations can reduce the cost of a circuit with increased propagation delay penalties
- Consider the minimum-cost SOP expression

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

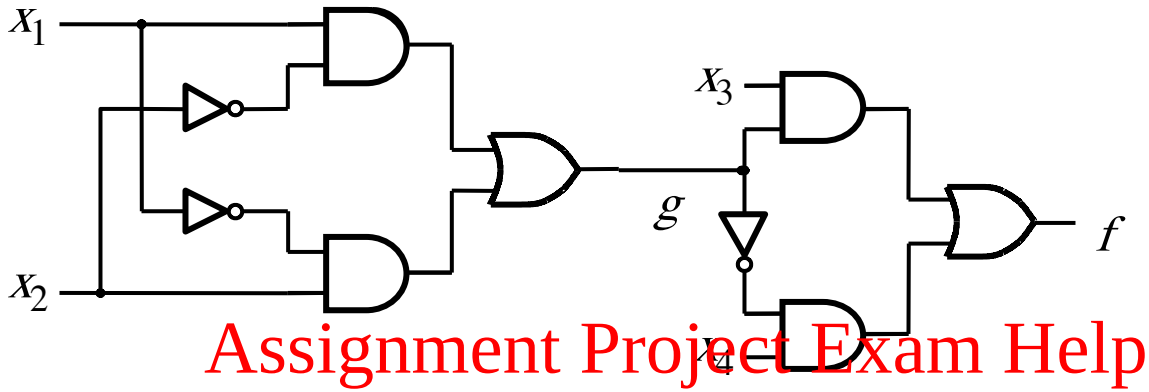
$= (\bar{x}_1 x_2 + x_1 \bar{x}_2) x_3 + (x_1 x_2 + \bar{x}_1 \bar{x}_2) x_4$

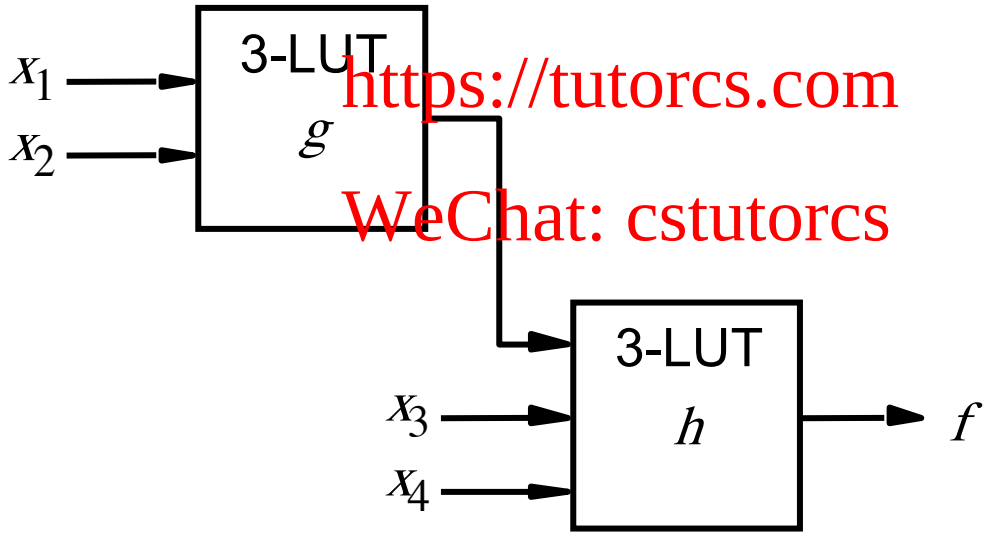
$= g x_3 + \bar{g} x_4$ with $g = \bar{x}_1 x_2 + x_1 \bar{x}_2$ [6 FO2 gates+15 inputs]*

* inc. inverters



The structure of decomposition in L02/S31

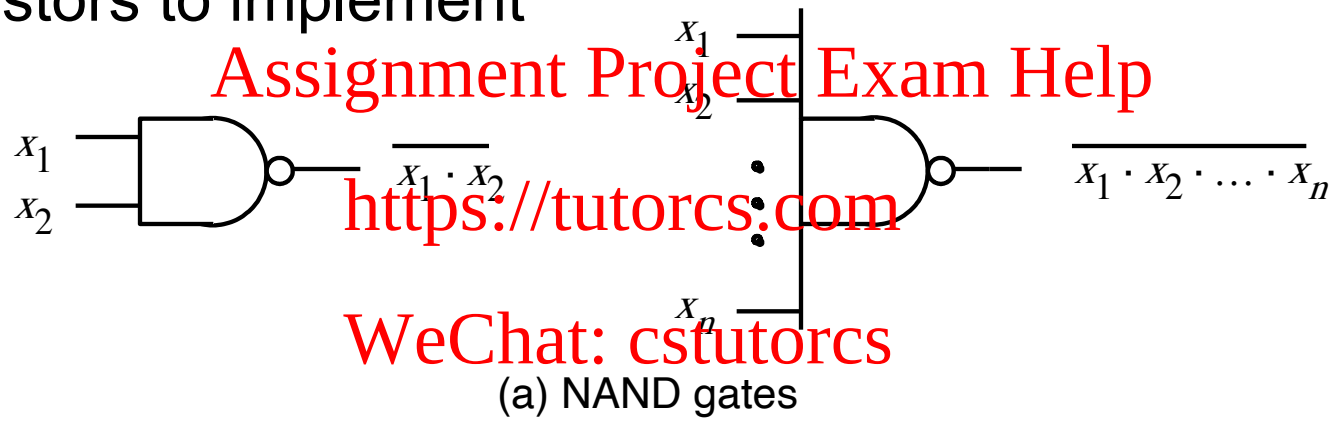




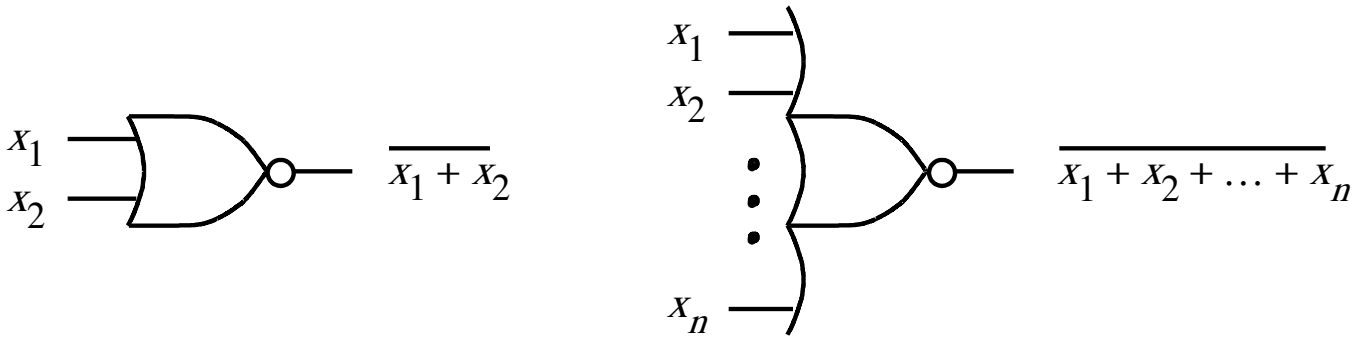
Q: How many 2-LUTs are needed to implement this function?

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



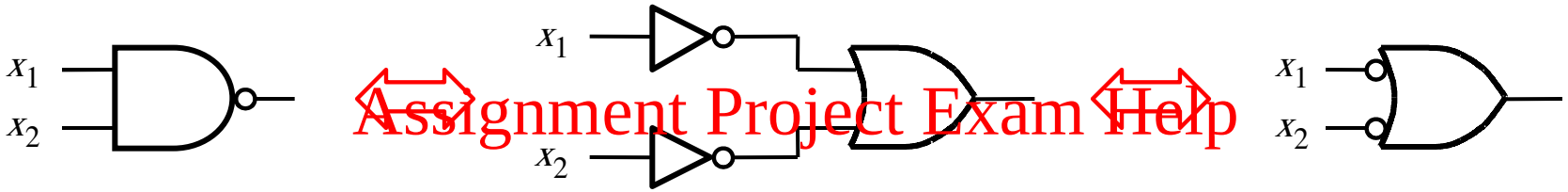
(a) NAND gates



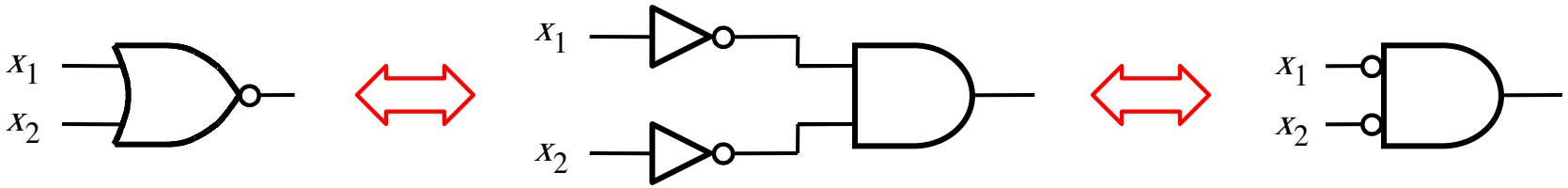
(b) NOR gates

DeMorgan's theorem in terms of logic gates

Allows us to transform AND-OR networks into NAND-only or NOR-only equivalent networks

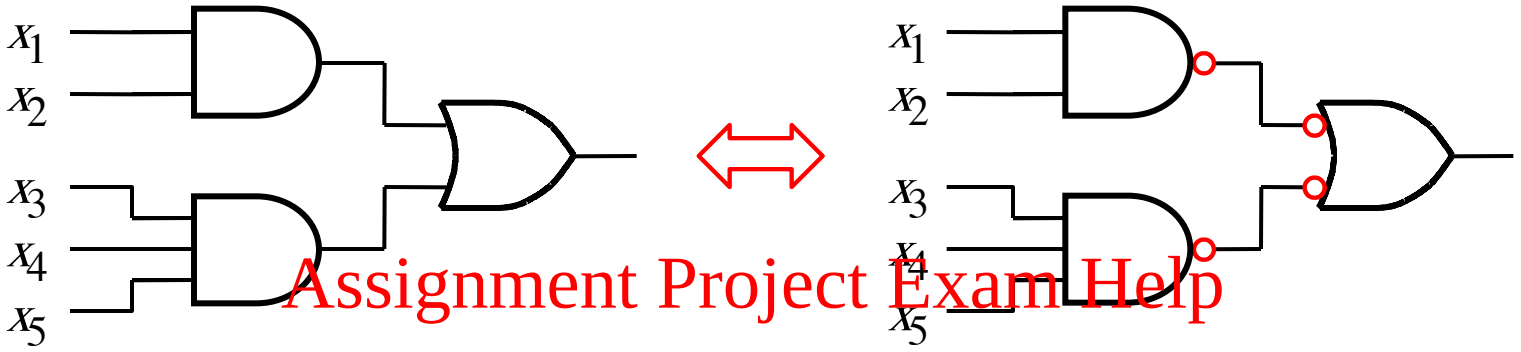


(a) $\overline{x_1 x_2} = \overline{x}_1 + \overline{x}_2$



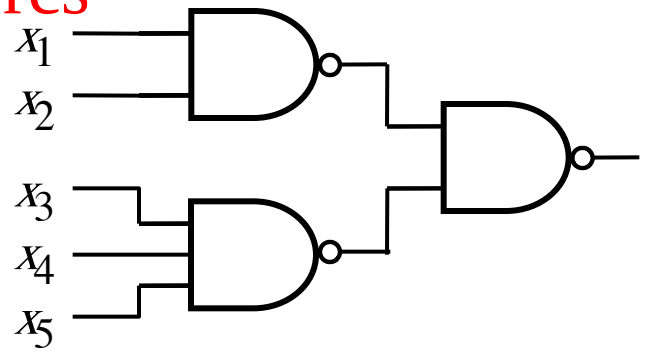
(b) $\overline{x_1 + x_2} = \overline{x}_1 \overline{x}_2$

Using NAND gates to implement a SOP network

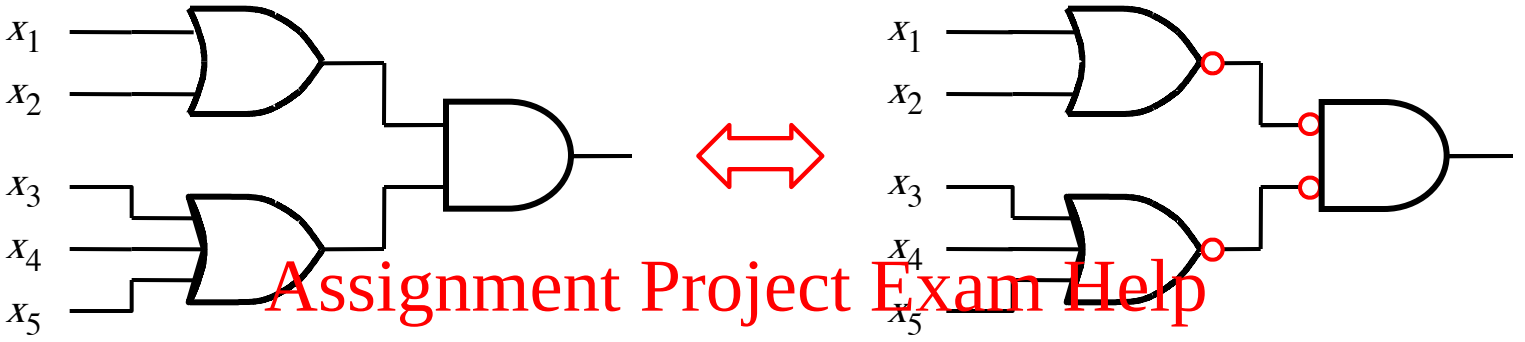


To transform network into NAND-only type:

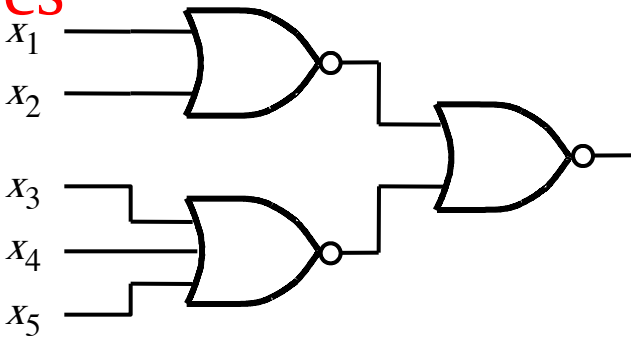
1. Replace AND & OR gates with equivalent NAND gates
2. Ensure the logical value of no wire is changed due to step 1. Insert additional bubbles into wires where only one bubble was added during step 1.



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 $f = m_2 + m_3 + m_4 + m_6 + m_7$

  $= \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3$

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

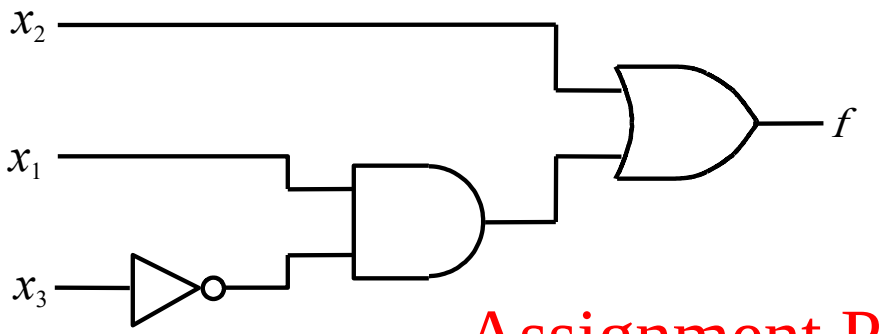
  $f = \overline{x}_1 x_2 (\overline{x}_3 + x_3) + x_1 (\overline{x}_2 + x_2) \overline{x}_3 + x_1 x_2 (\overline{x}_3 + x_3)$

  $= \overline{x}_1 x_2 + x_1 \overline{x}_3 + x_1 x_2$

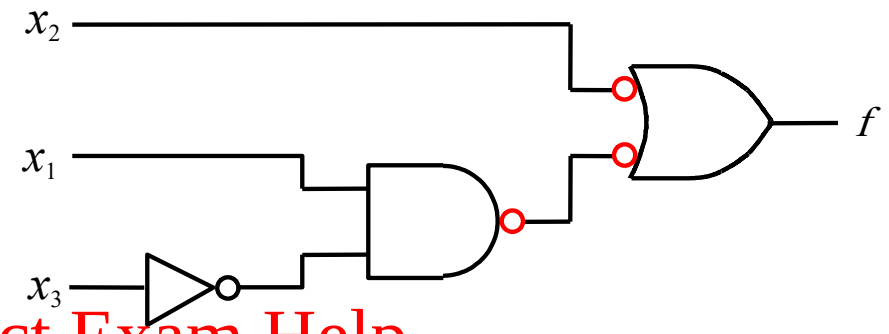
  $= (\overline{x}_1 + x_1) x_2 + x_1 \overline{x}_3$

  $= x_2 + x_1 \overline{x}_3$

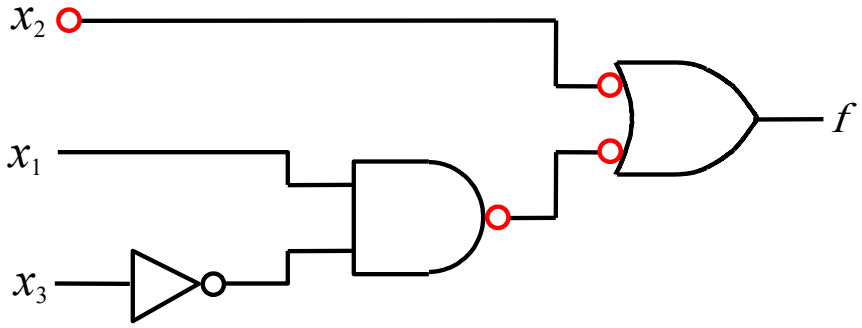
NAND-gate realization of the function in Example 2.3



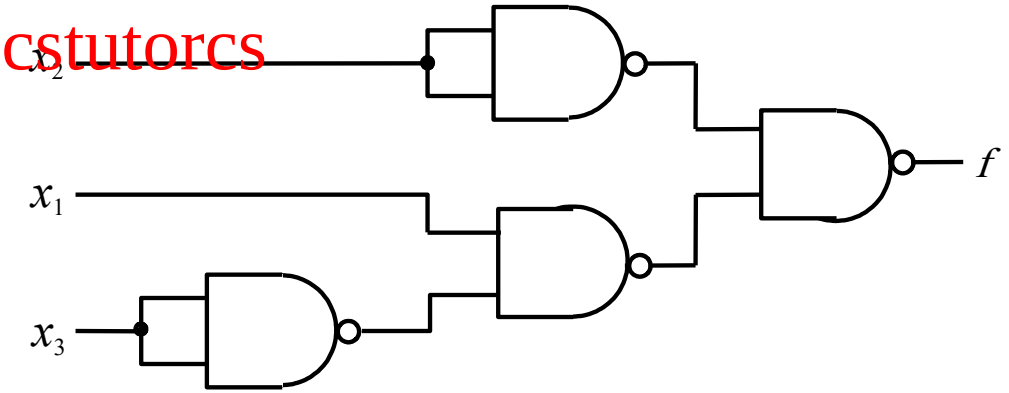
(a) SOP implementation



(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

  $f = M_0 \cdot M_1 \cdot M_5$

  $= (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_3)$

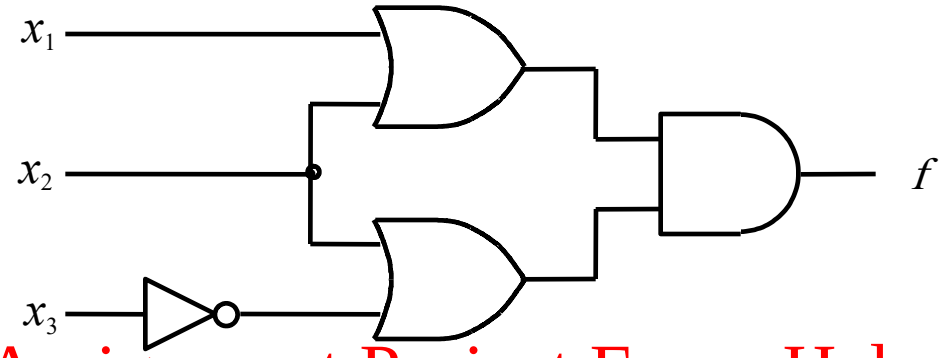
- A simplifeied POS expression can be derived as

  $f = ((x_1 + x_2) + x_3)((x_1 + x_2) + \overline{x}_3)(x_1 + (x_2 + \overline{x}_3))(\overline{x}_1 + (x_2 + \overline{x}_3))$

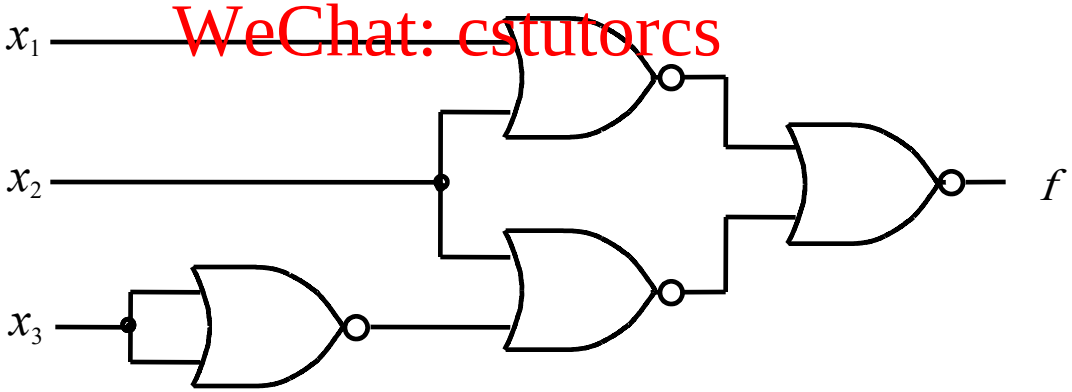
  $= ((x_1 + x_2) + x_3\overline{x}_3)(x_1\overline{x}_1 + (x_2 + \overline{x}_3))$

  $= (x_1 + x_2)(x_2 + \overline{x}_3)$

NOR-gate realization of the function in Example 2.4

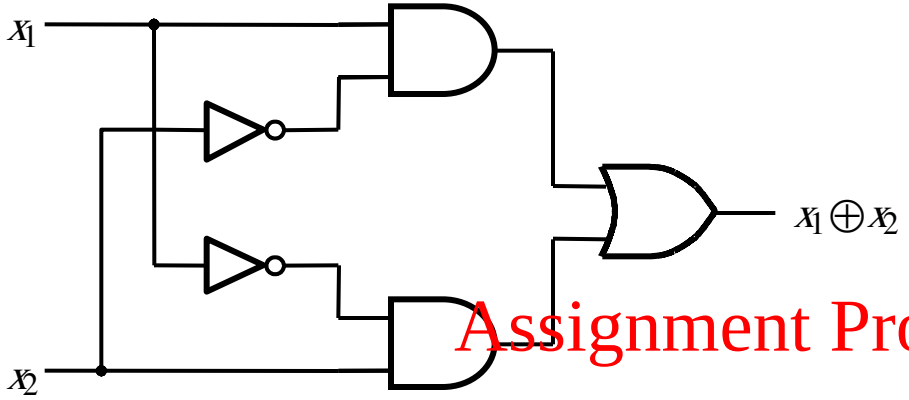


(a) POS implementation

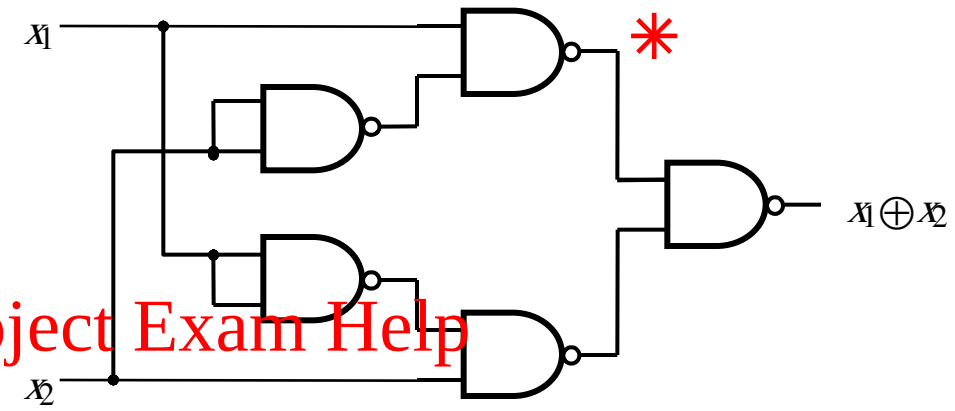


(b) NOR implementation

Implementation of XOR



(a) Sum-of-products implementation



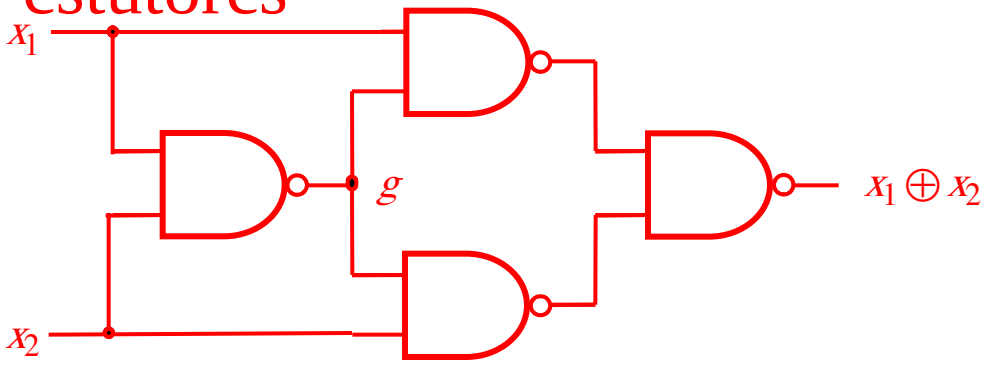
(b) NAND gate implementation

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

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

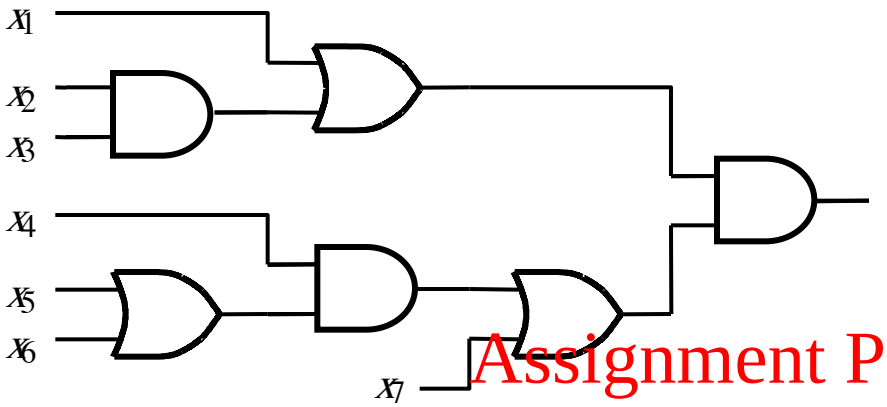
$= \overline{(x_1 \cdot (\overline{x_1} + \overline{x_2}))}$

$= \overline{x_1 \cdot (\overline{x_1 \cdot x_2})}$

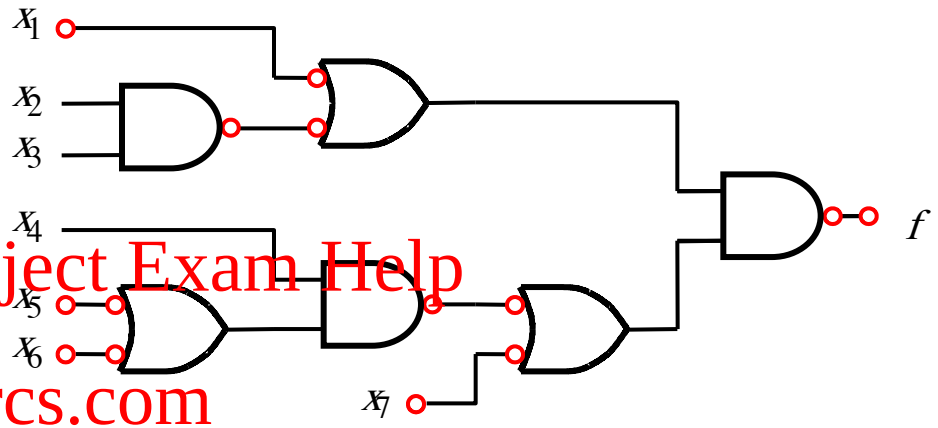


(c) Optimal NAND gate implementation

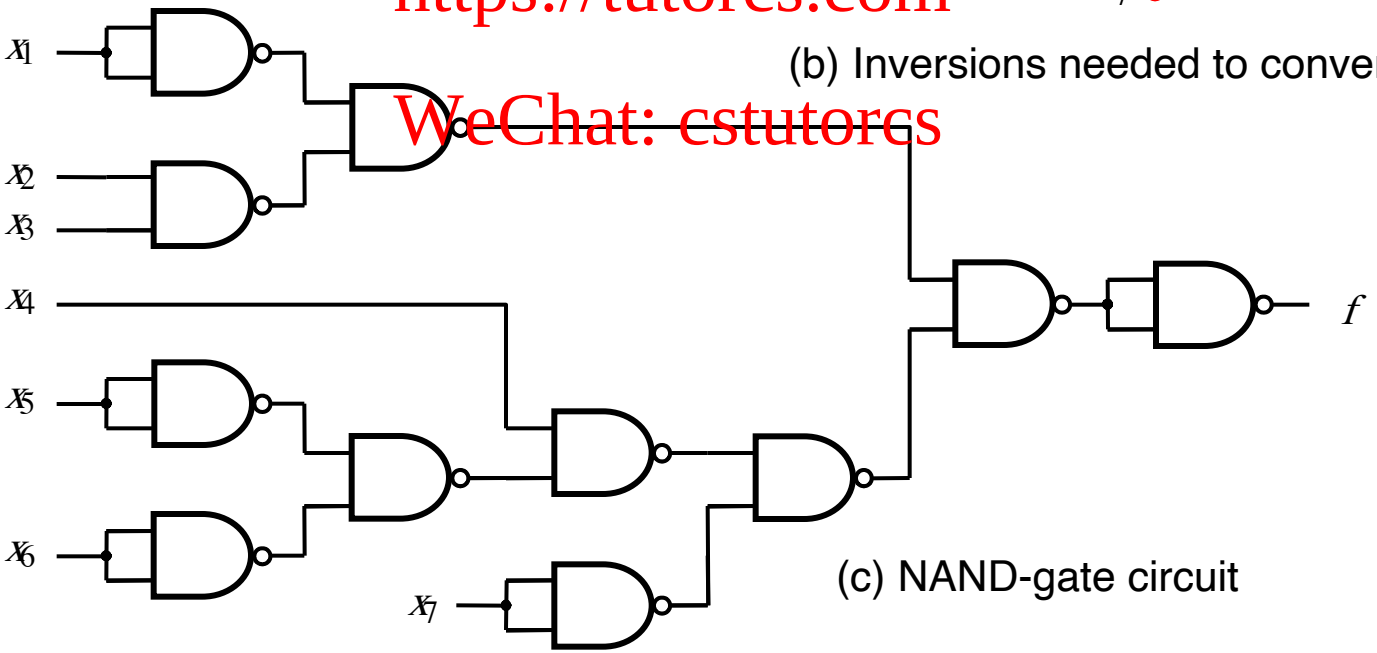
Multilevel NAND-gate circuit



(a) Circuit with AND and OR gates

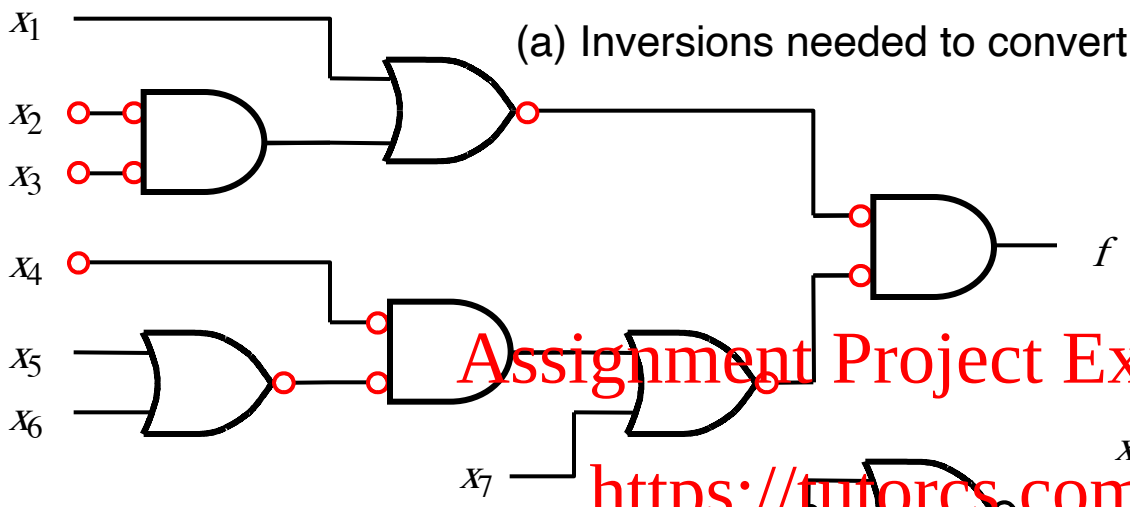


(b) Inversions needed to convert to NANDs

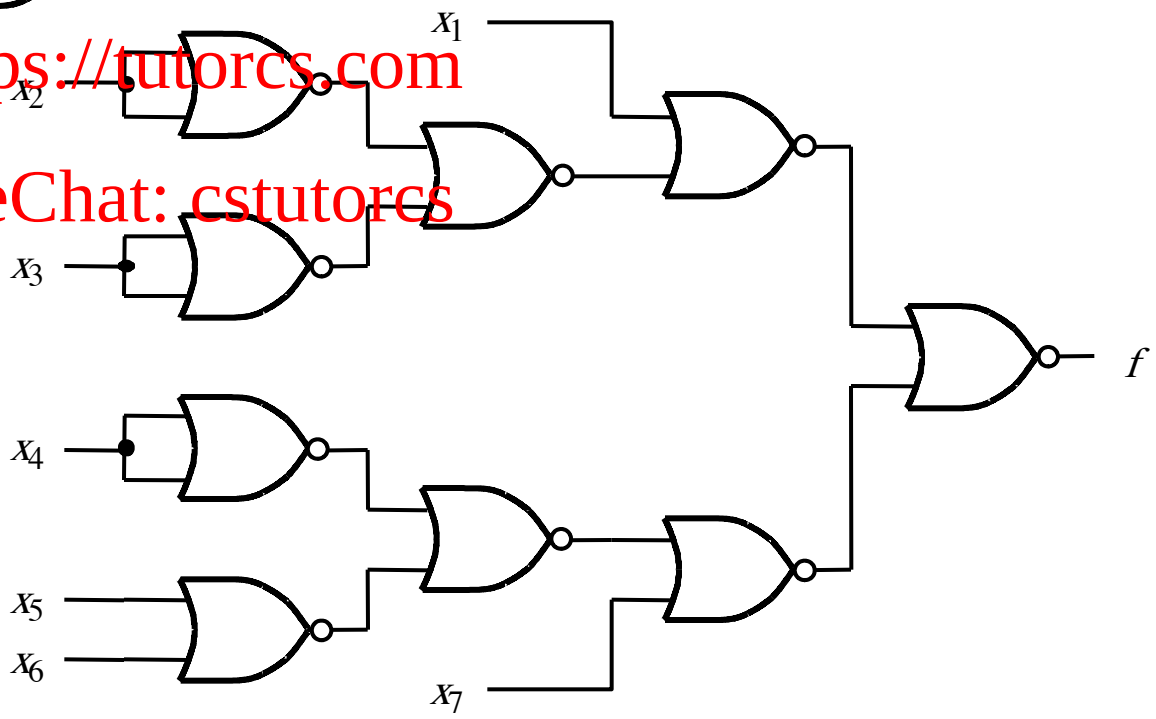


(c) NAND-gate circuit

Equivalent multilevel NOR-gate circuit



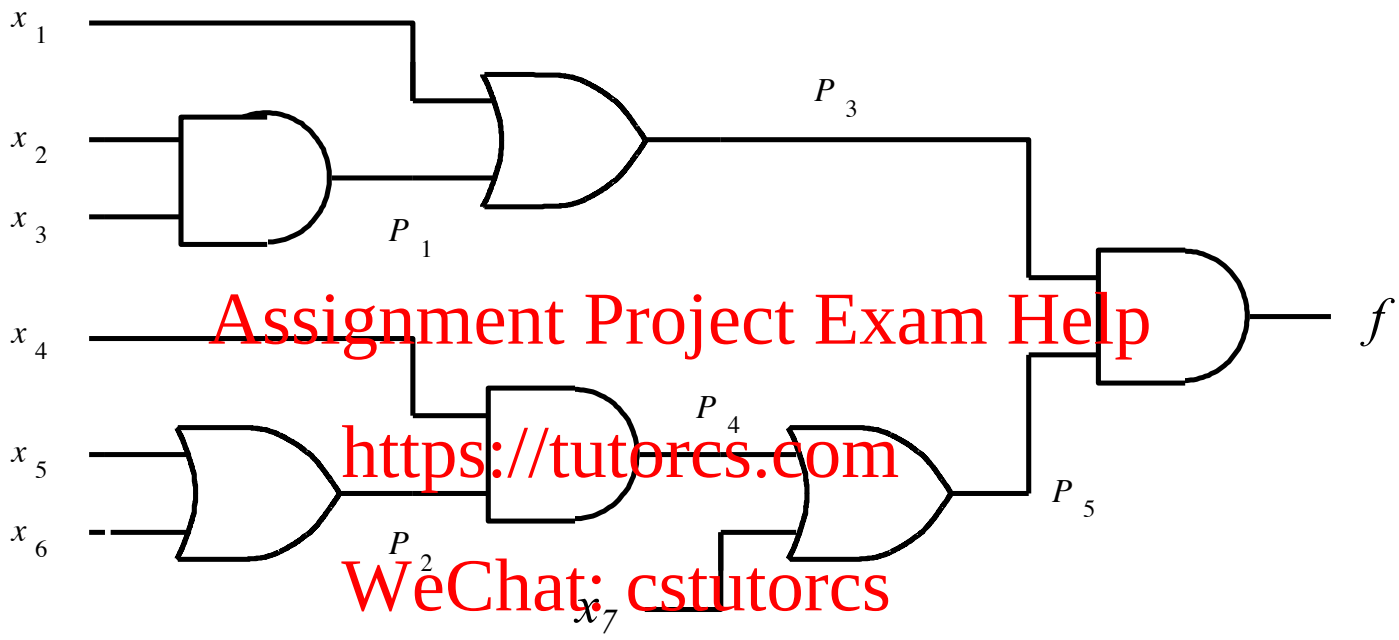
(a) Inversions needed to convert to NORs



(b) NOR-gate circuit

Note that this circuit uses less gates and less levels of gates than the NAND-only circuit of L02/S42(c)

Analysis of multilevel circuit



$P_1 = x_2x_3$

$P_2 = x_5 + x_6$

$P_3 = x_1 + P_1 = x_1 + x_2x_3$

$P_4 = x_4P_2 = x_4(x_5 + x_6)$

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

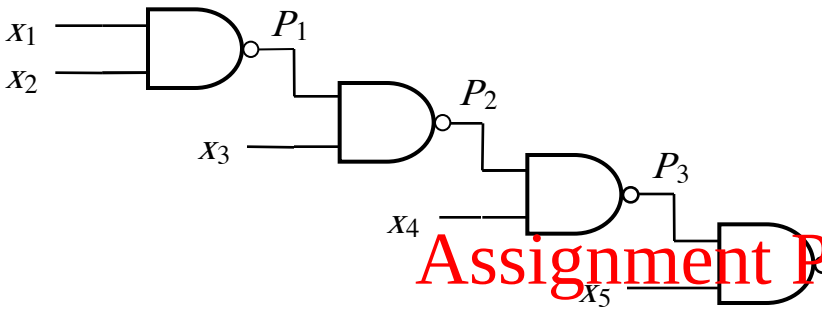
$f = P_3P_5$

$= (x_1 + x_2x_3)(x_4(x_5 + x_6) + x_7)$

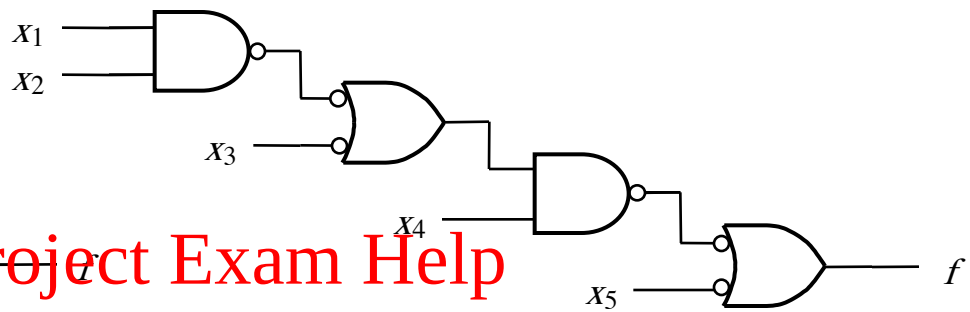
$= x_1x_4x_5 + x_1x_4x_6 + x_1x_7 +$

$x_2x_3x_4x_5 + x_2x_3x_4x_6 + x_2x_3x_7$

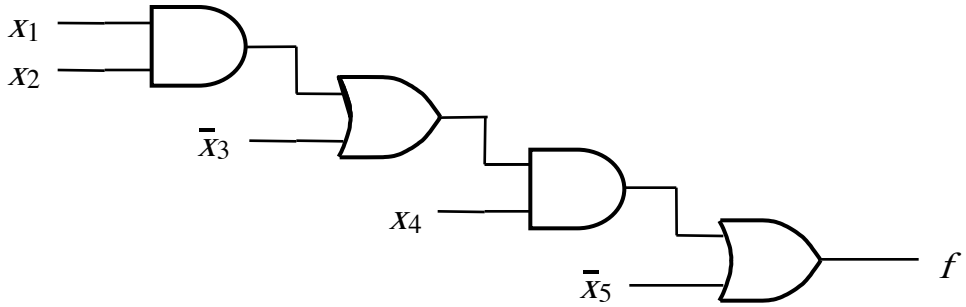
Analyzing multilevel NOR/NAND-only circuits



(a) NAND-gate circuit

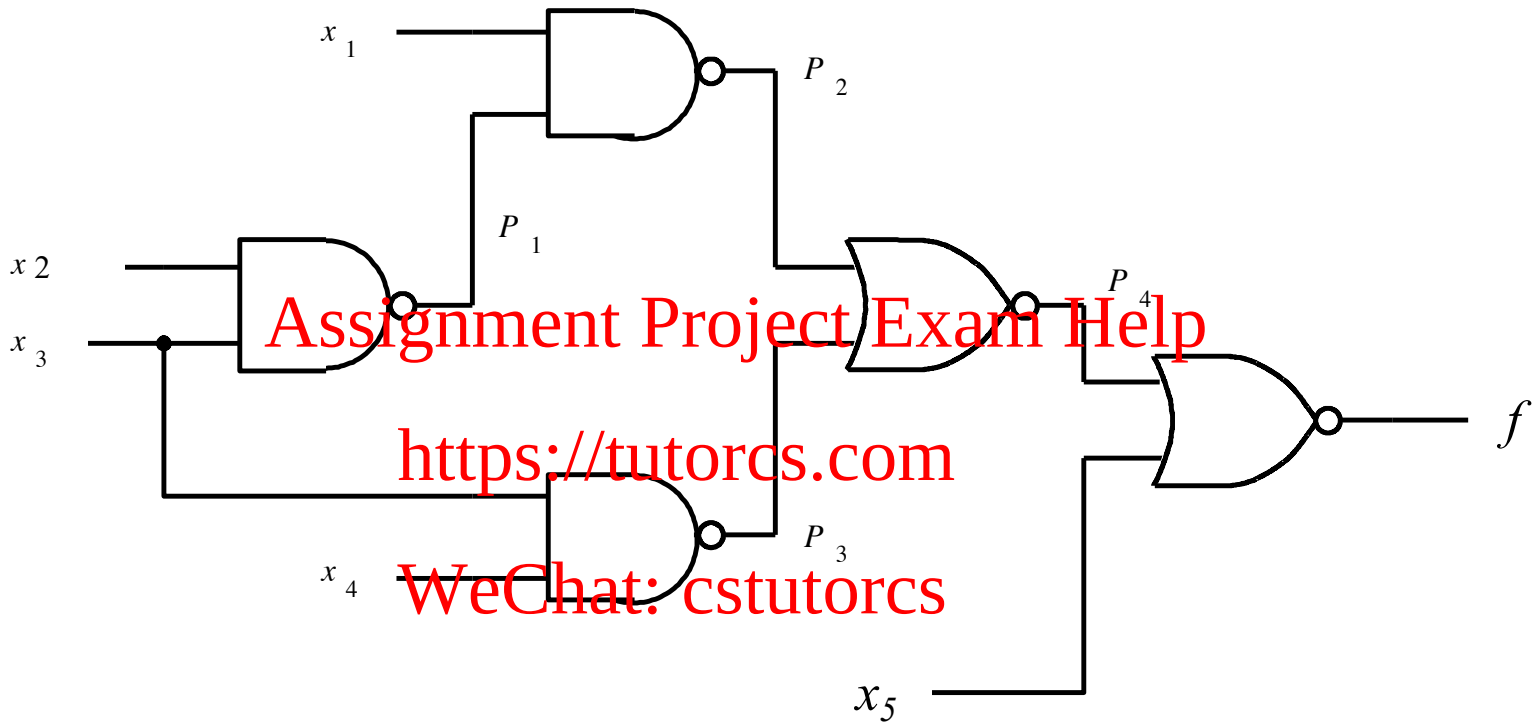


(b) Moving bubbles to convert to ANDs and ORs avoids multiple, tedious inversions



(c) Circuit with AND and OR gates

Exercise: What *f* does this circuit implement?



The VHDL code for the function in L02/S3

LIBRARY is needed to include package

These two lines need to be included before every design module that uses data objects of type std_logic

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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

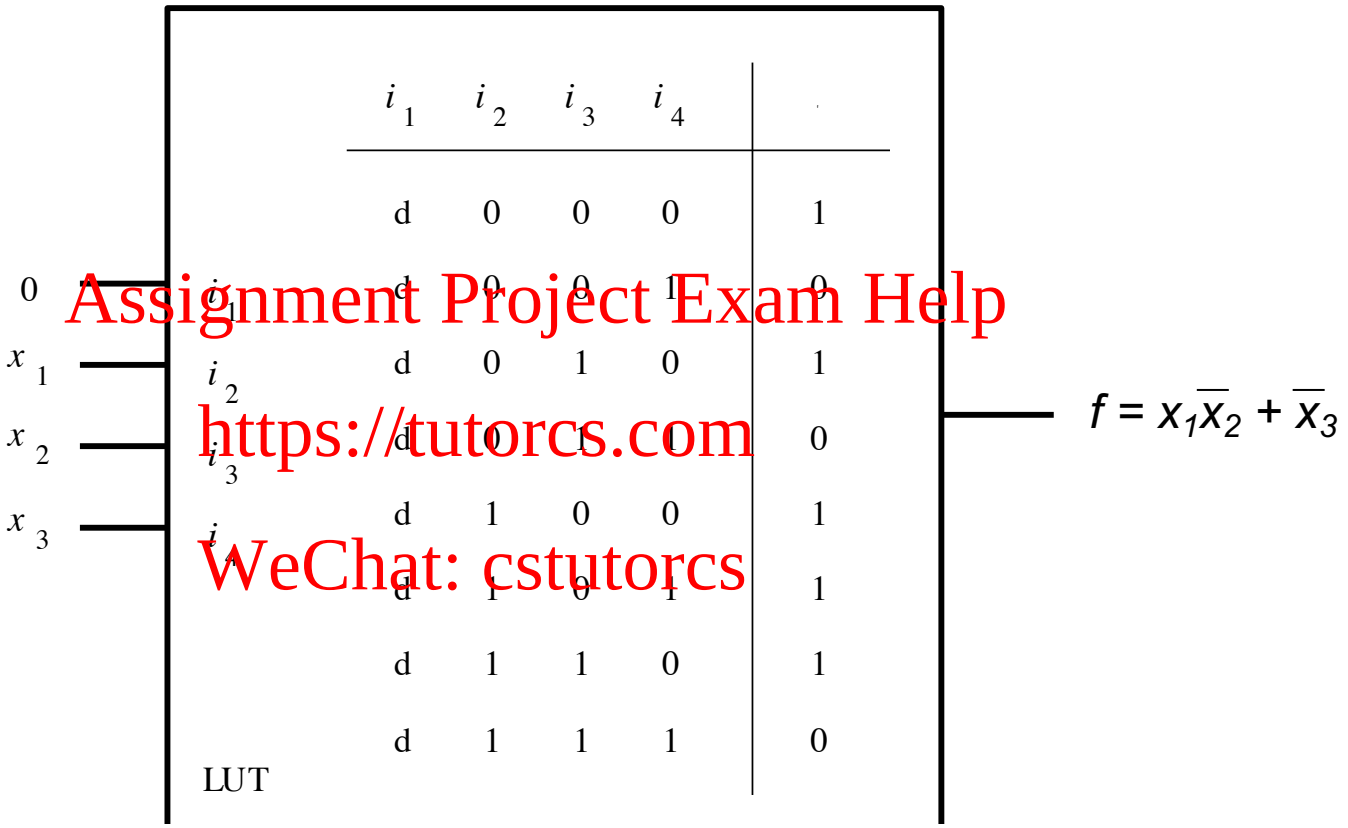
The std_logic package defines legal values and uses for data type; STD_LOGIC values: {0,1, Z, -, U, X, W, L, H}

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

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

Z high impedance
- don't care
U uninitialized
X unknown
W weak signal
L weak tending to 0
H weak tending to 1

The VHDL code in L02/S47 implemented in a 4-LUT



The VHDL code for the function of L02/S23

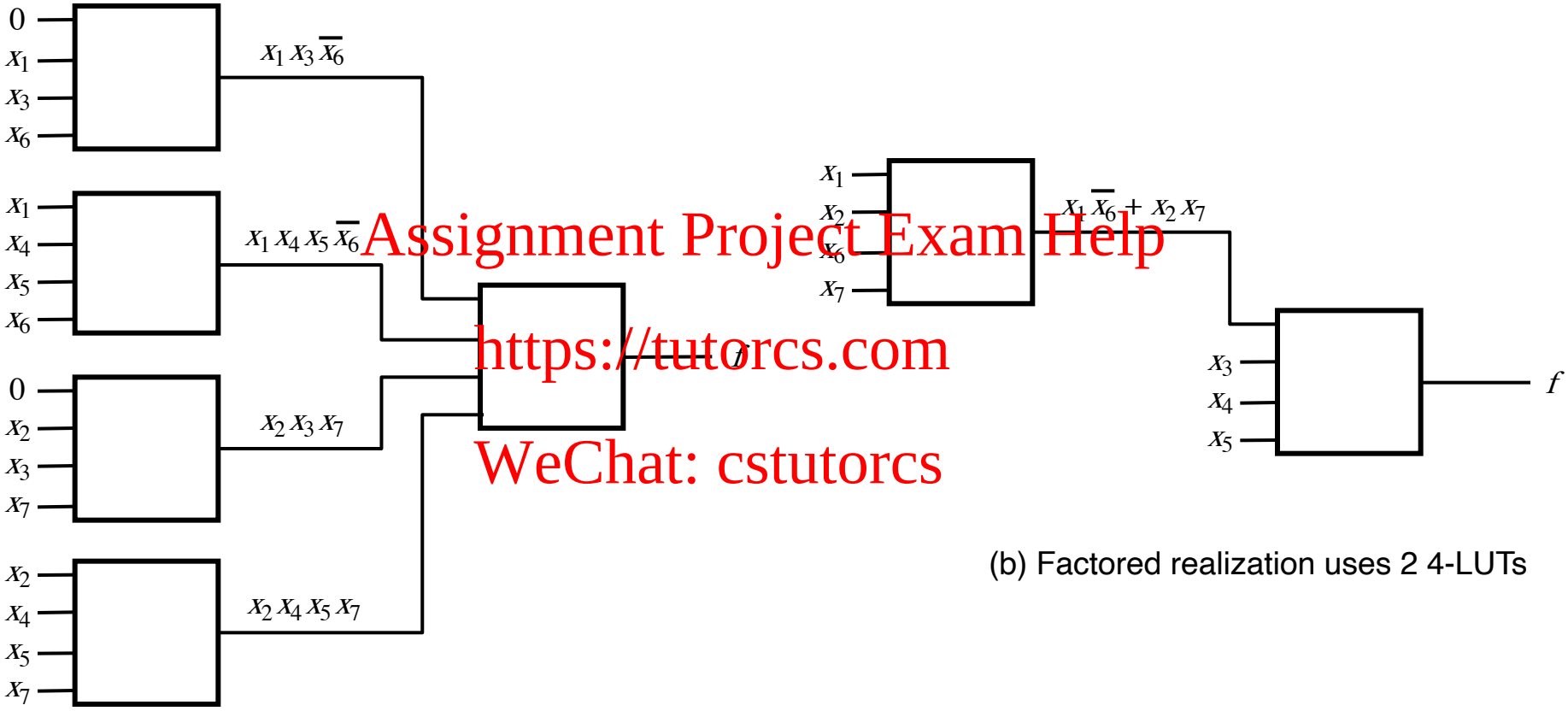
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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/S49

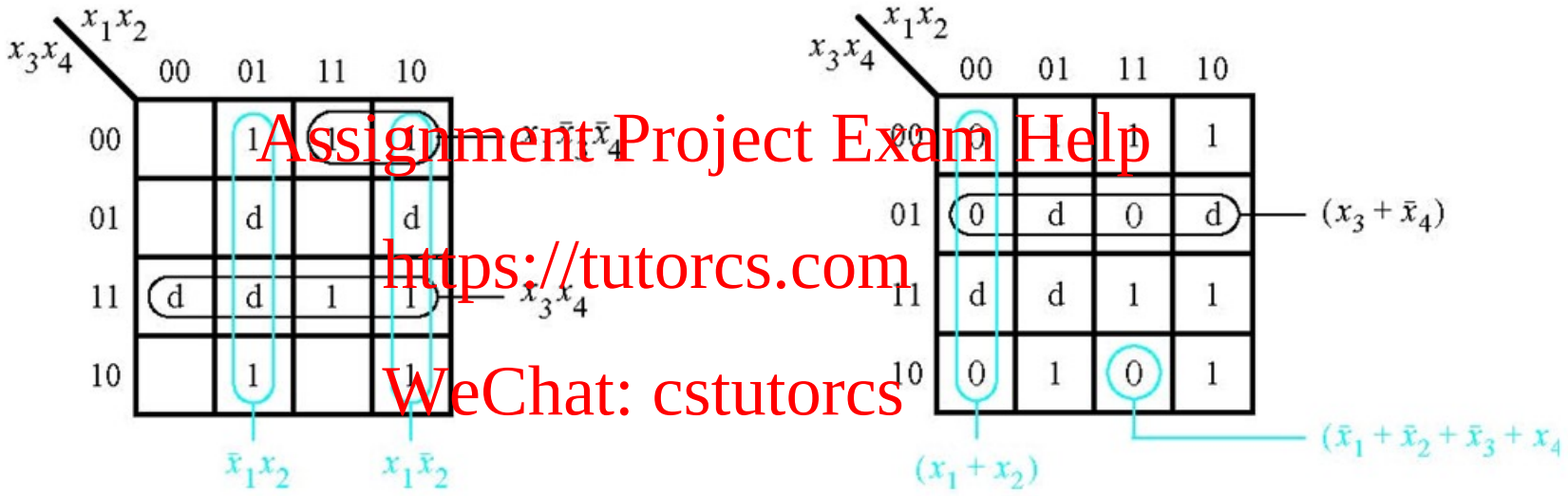


(a) Sum-of-products realization requires 5 4-LUTs

(b) Factored realization uses 2 4-LUTs

Exercise:

- Determine the minimum-cost SOP and POS expressions for the function f(x1,x2,x3,x4) = Σm(4,6,8,10,11,12,15) + D(3,5,7,9)



(a) Determination of the SOP expression

(b) Determination of the POS expression

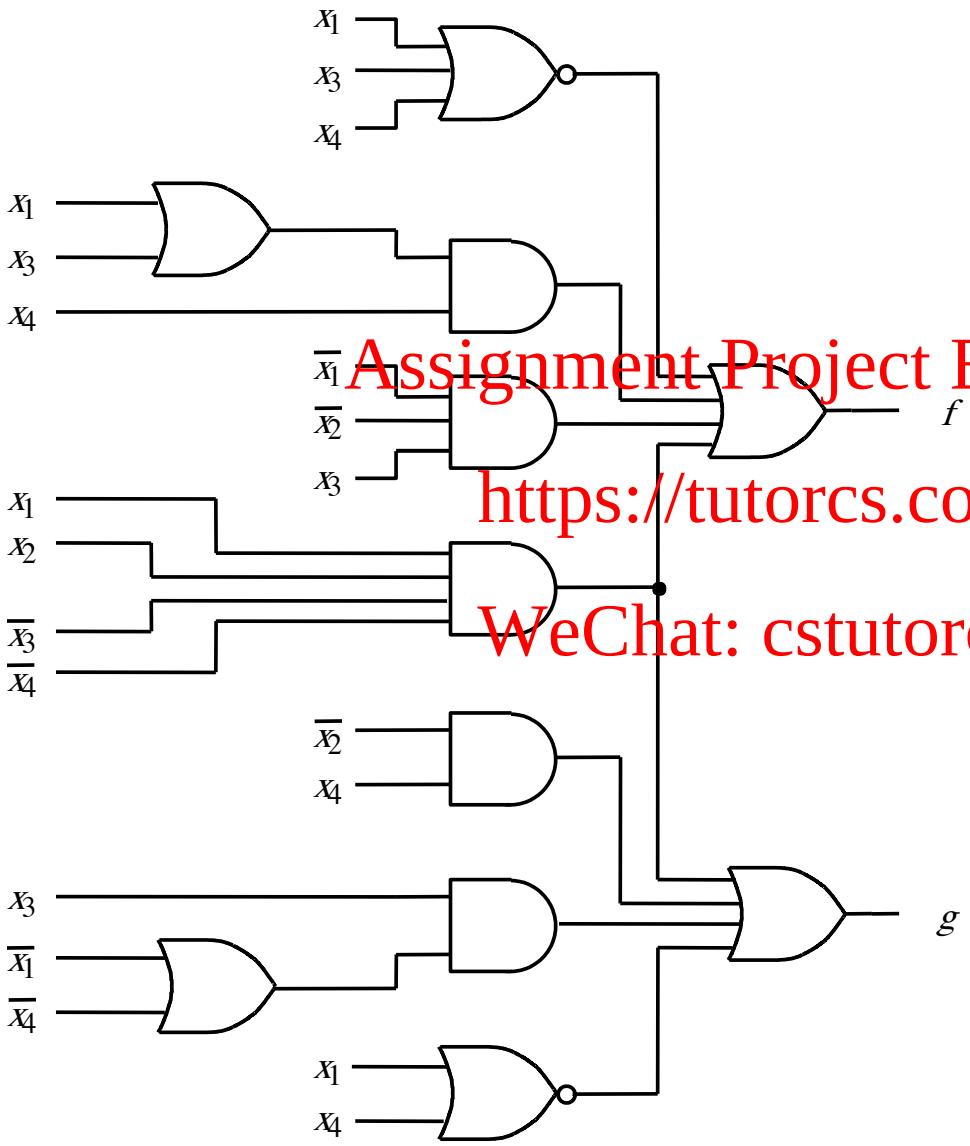
Exercise:

- 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 $D = \Sigma(9,12,14)$

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?)