L4: Karnaugh diagrams, two-, and multi-level minimization

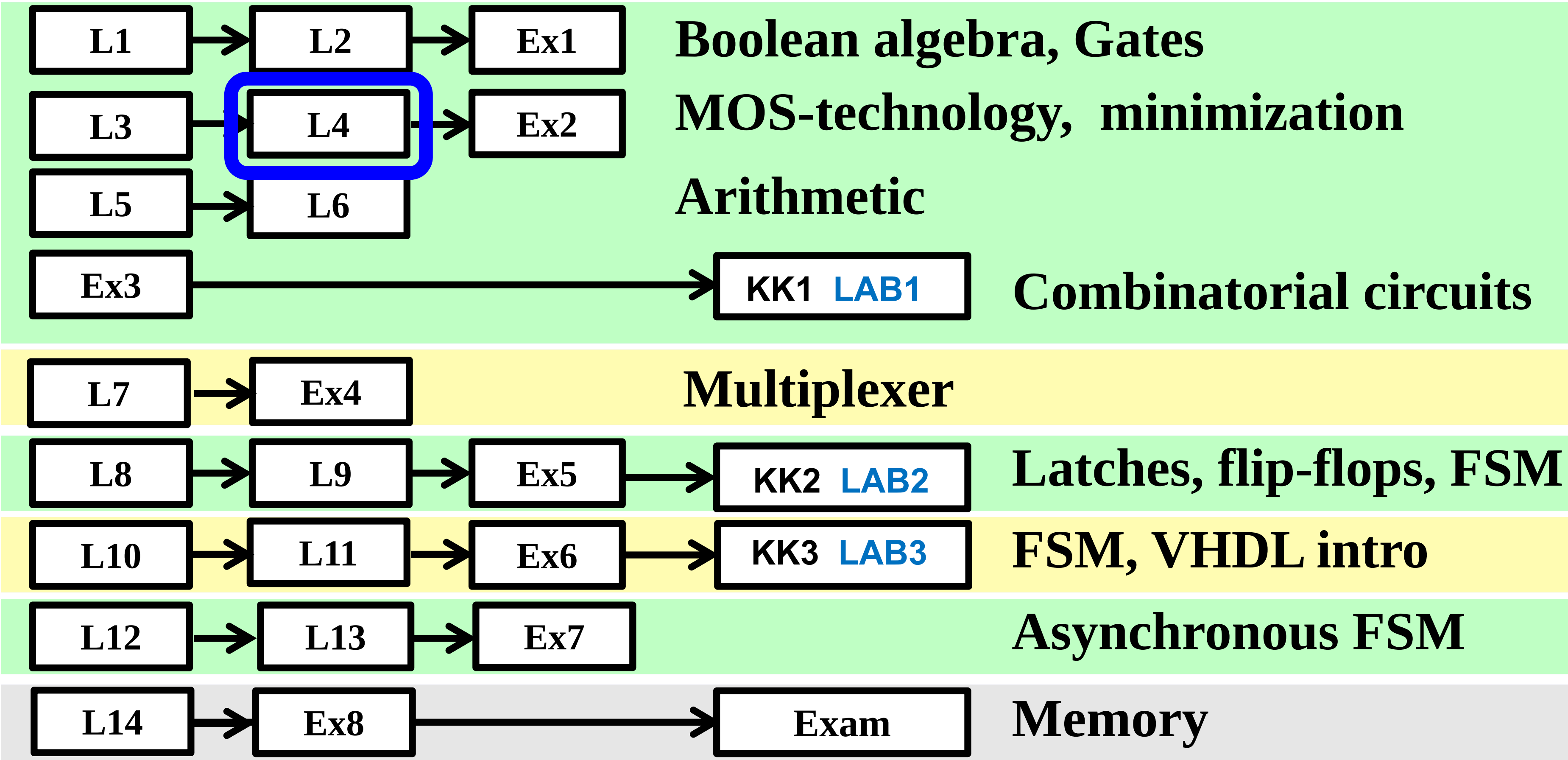
KTH Informations- och kommunikationsteknik

Masoumeh (Azin) Ebrahimi

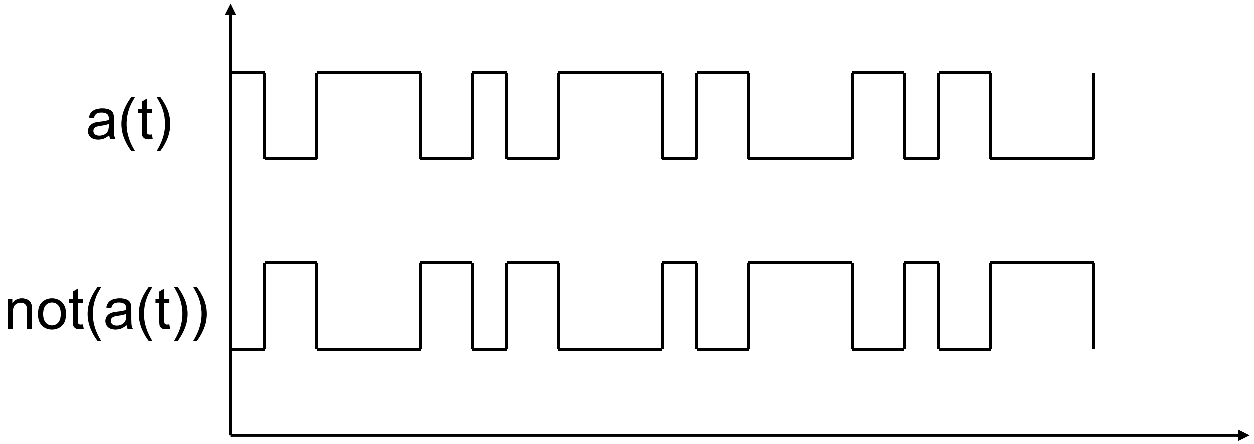
KTH/ICT

mebr@kth.se

IE1204 Digital Design



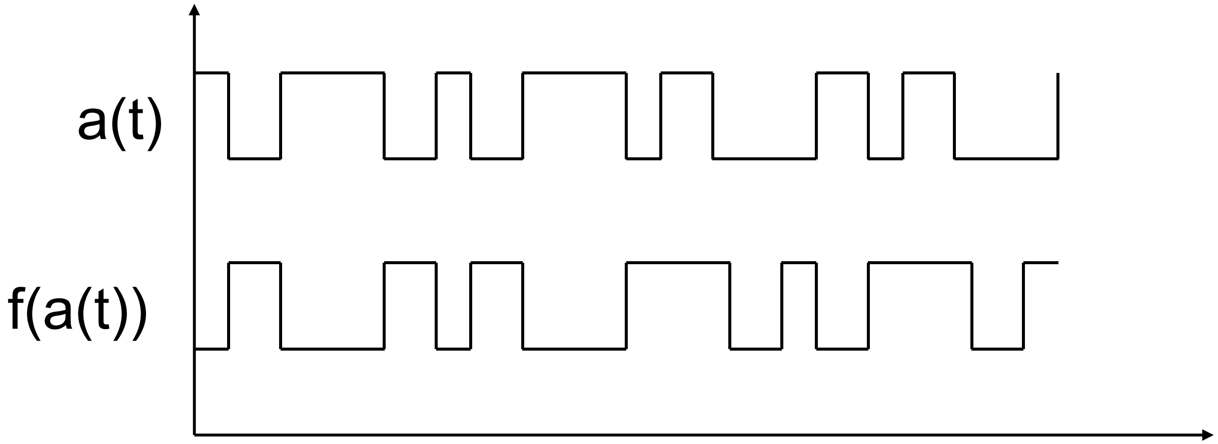
Combinatorial system



A combinatorial system has no memory - its output depends therefore **ONLY** on the **PRESENT** value of the input signal

Lecture 4 - Lecture 7

Sequential system



A sequential system has a built-in memory - its output depends therefore **BOTH** on the **PRESENT** and **PREVIOUS** value(s) of the input signal

Lecture 8 - Lecture 13

This lecture covers …

- BV pp. 168-211

Minterms

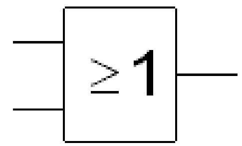
- A minterm MUST contain all variables, otherwise it is not a minterm
- A minterm represents the combination of values of function's variables for which the function evaluates to 1

Example:

$$f = \sum m(1,2,3) = \bar{x}_1 x_0 + x_1 \bar{x}_0 + x_1 x_0$$

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

OR



Minimization with Boolean algebra

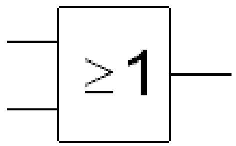
Using Boolean algebra, we can shorten the expression to:

$$
\begin{aligned}
f \quad &= \bar{x}_1 x_0 + x_1 \bar{x}_0 + x_1 x_0 = \bar{x}_1 x_0 + x_1(\bar{x}_0 + x_0) \\
&= \bar{x}_1 x_0 + x_1 \,(\mathbf{1}) = \bar{x}_1 x_0 + x_1 \,(1 + x_0) = \\
&= \bar{x}_1 x_0 + x_1 + x_1 x_0 = x_0 \,(\bar{x}_1 + x_1) + x_1 = x_0 \,(\mathbf{1}) + x_1 \\
&= \boxed{x_1 + x_0}
\end{aligned}
$$

As expected!

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

OR



Maxterm

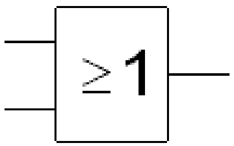
- A maxterm MUST contain all variables, otherwise it is not a maxterm
- A minterm represents the combination of values of function's variables for which the function evaluates to 0

Example:

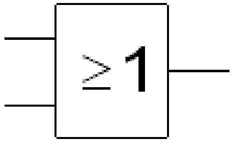
$$f = \prod M(0) = x_0 + x_1$$

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

OR



Graphical minimization method

OR 

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

$x_1 \backslash x_0$	0	1
0	0	1
1	1	1

$x_1 \backslash x_0$	0	1
0	m_0	m_1
1	m_2	m_3

$$\boxed{m_2 + m_3} = x_1\bar{x}_0 + x_1x_0 =$$

$$= x_1(\bar{x}_0 + x_0) = x_1$$

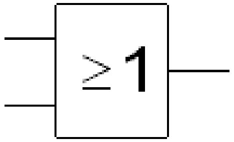
$$\boxed{m_1 + m_3} = \bar{x}_1x_0 + x_1x_0 =$$

$$= x_0(\bar{x}_1 + x_1) = x_0$$

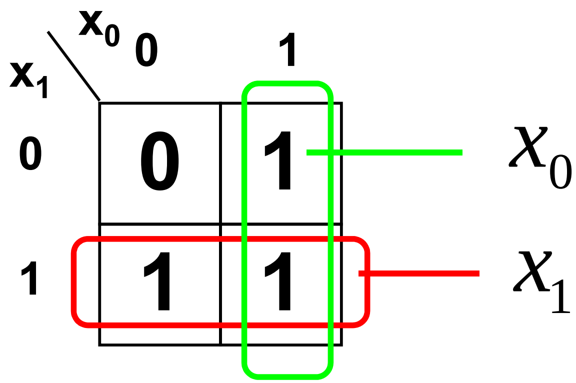
$$f = x_1 + x_0$$

Graphical minimization method

OR



	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1



$$f = x_1 + x_0$$

”Group” two ones that are "neighbors" (vertically or horizontally).

Minterms could then be reduced to "what they have in common".

Commonly used functions in two-dimensional table-form

AND

x_1 \ x_0	0	1
0	0	0
1	0	1

$$x_1 \cdot x_0$$

OR

x_1 \ x_0	0	1
0	0	1
1	1	1

$$x_1 + x_0$$

XOR

x_1 \ x_0	0	1
0	0	1
1	1	0

$$\overline{x_1} x_0 + x_1 \overline{x_0}$$

NOT

x_0	0	1
	1	0

$$\overline{x_0}$$

NAND

x_1 \ x_0	0	1
0	1	1
1	1	0

$$\overline{x_1} + \overline{x_0} = \overline{\overline{\overline{x_1} + \overline{x_0}}} = \overline{x_1 \cdot x_0}$$

NOR

x_1 \ x_0	0	1
0	1	0
1	0	0

$$\overline{x_1} \cdot \overline{x_0} = \overline{\overline{\overline{x_1} \cdot \overline{x_0}}} = \overline{x_1 + x_0}$$

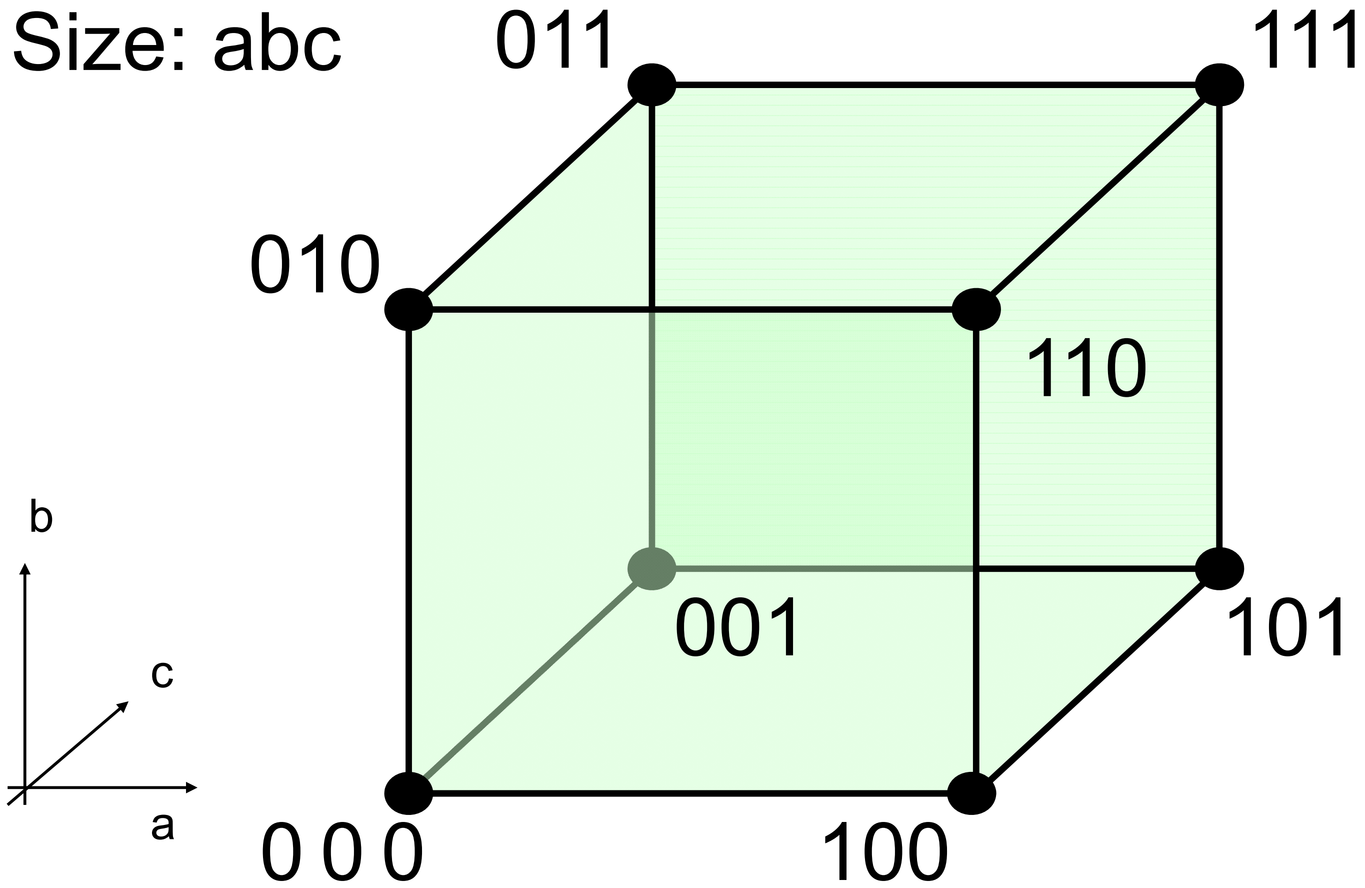
XNOR

x_1 \ x_0	0	1
0	1	0
1	0	1

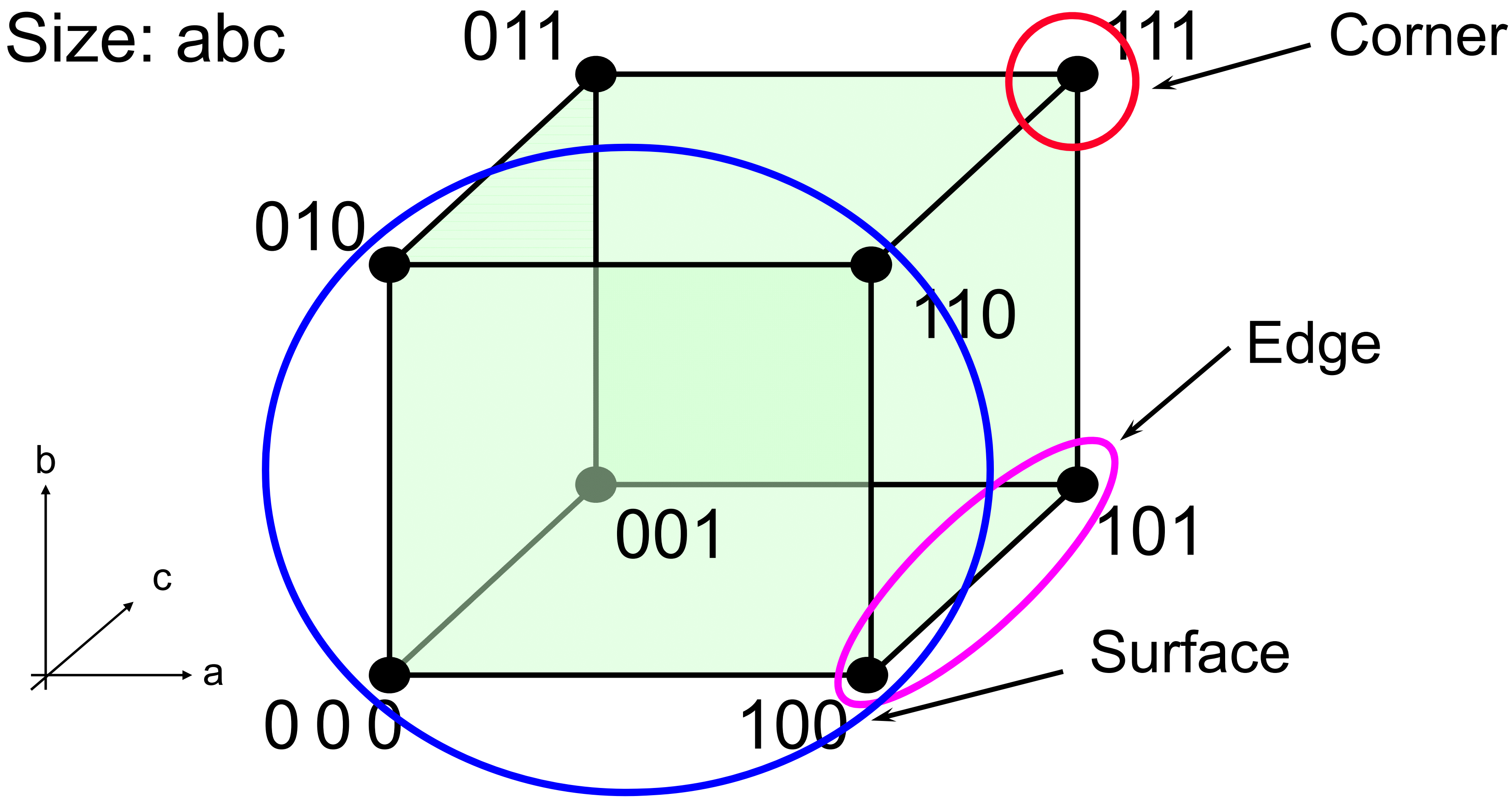
$$\overline{x_1}\,\overline{x_0} + x_1 x_0$$

3-dimensional Boolean space

Size: abc

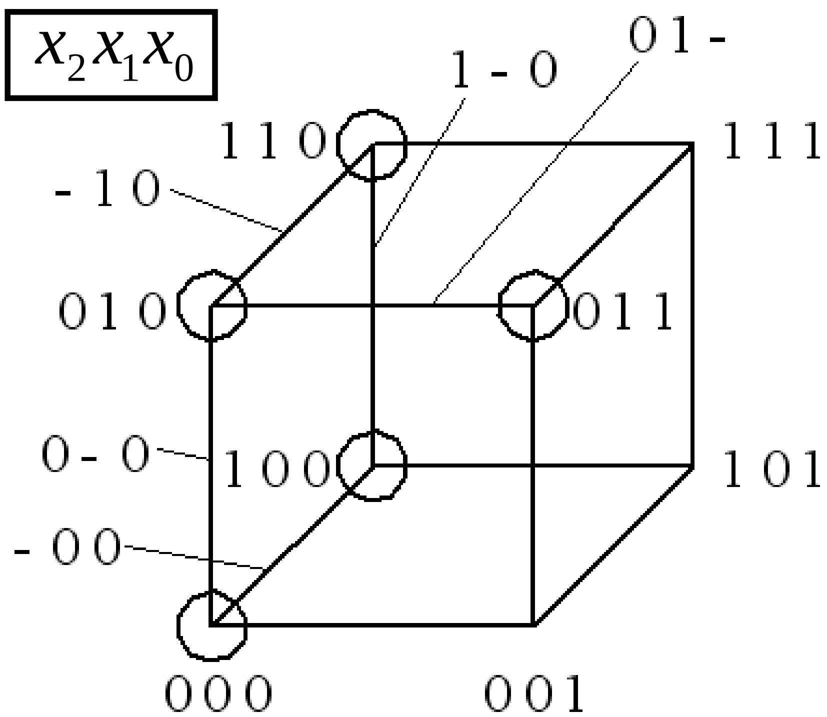


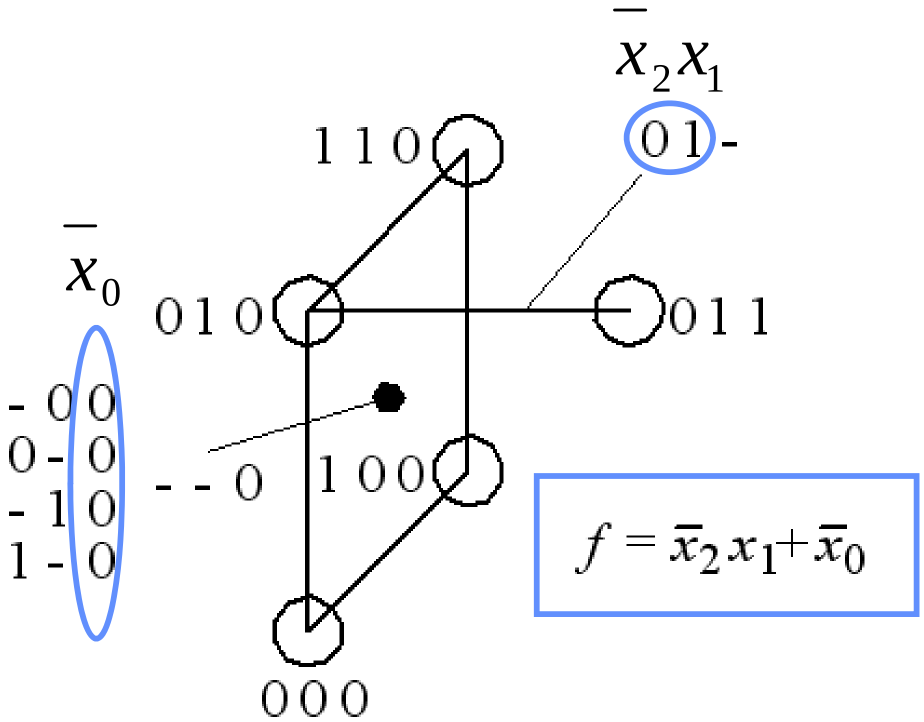
3-dimensional Boolean space



- Cube methods can be generalized to "Hyper Cubes" with any number of variables.

Minimization with cube





- A **surface** is represented by **a variable**.
- An **edge** is represented by **a product term with two variables**.
- A **corner** is represented by **a minterm with three variables**.

Graycode is a mirrored binarycode

00000
00001
00011
00010
00110
00111
00101
00100
01100
01101
01111
01110
01010
01011
01001
01000
11000
11001
11011
11010
11110
...

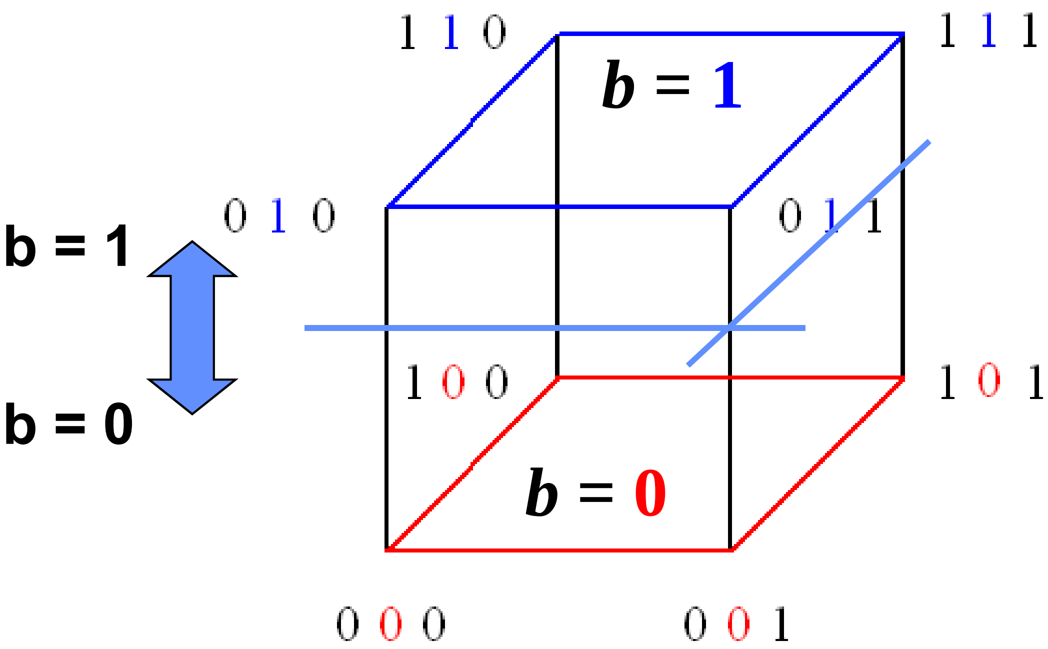
One can easily construct a Gray code with an arbitrary number of bits needed for numbering the "corners" in "hypercubes" with!

Graycode is a mirrored binarycode

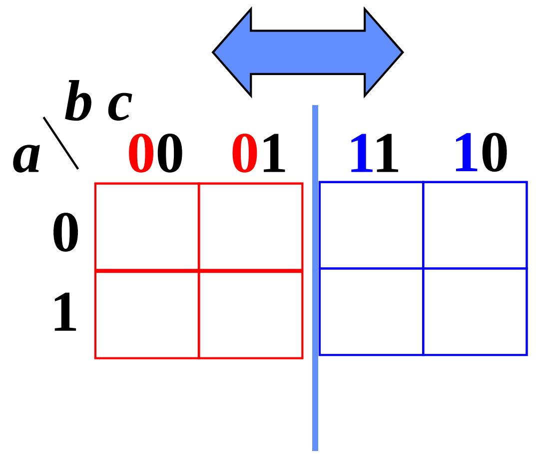
0	0	00	00	000	000	0000	0000	00000
1	1	01	01	001	001	0001	0001	00001
	1	11	11	011	011	0011	0011	00011
	0	10	10	010	010	0010	0010	00010
			10	110	110	0110	0110	00110
			11	111	111	0111	0111	00111
			01	101	101	0101	0101	00101
			00	100	100	0100	0100	00100
					100	1100	1100	01100
					101	1101	1101	01101
					111	1111	1111	01111
					110	1110	1110	01110
					010	1010	1010	01010
					011	1011	1011	01011
					001	1001	1001	01001
					000	1000	1000	01000
							1000	11000
							1001	11001
							1011	11011
							1010	11010
							1110	11110
							...	...

3D-cube $\Rightarrow$ Karnaugh map

a b c



Gray-code → mirror



00
01

11
10

Gray code

A function of four variables a b c d

Truth Table with 11 ”1” and 5 ”0”. The function can be expressed in SoP-form with 11 minterms or in PoS-form with 5 maxterms.

	abcd	f
0	0000	1
1	0001	1
2	0010	1
3	0011	1
4	0100	1
5	0101	1
6	0110	1
7	0111	1
8	1000	1
9	1001	0
10	1010	1
11	1011	0
12	1100	0
13	1101	1
14	1110	0
15	1111	0

$$f(a,b,c,d) = \sum(0,1,2,3,4,5,6,7,8,10,13)$$

$$f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} +$$
$$\bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}c\bar{d} + ab\bar{c}d$$

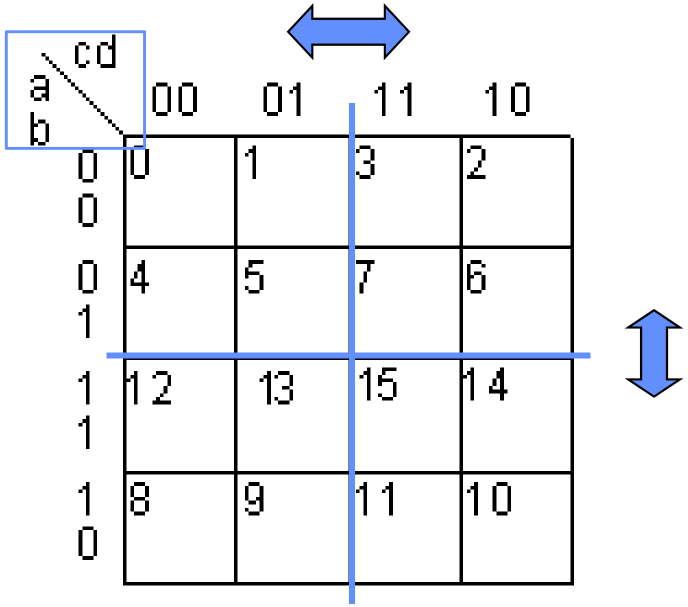
$$f(a,b,c,d) = \prod(9,11,12,14,15)$$

$$f = (\bar{a}+b+c+\bar{d})\cdot(\bar{a}+b+\bar{c}+\bar{d})\cdot(\bar{a}+\bar{b}+c+d)\cdot(\bar{a}+\bar{b}+\bar{c}+d)\cdot(\bar{a}+\bar{b}+\bar{c}+\bar{d})$$

4D-cube ⇒ 2D-map

The Karnaugh map is the truth table lined up in a different way.

	abcd
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111



The frames are ordered in such way that only one bit changes between two vertical frames or horizontal frames. This order is called Gray-code.

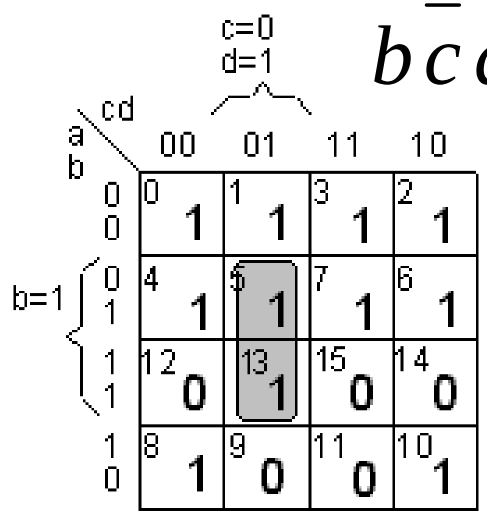
Two "neighbors"

The frames "5" and "13" are "neighbors" in the Karnaugh map (but they are distant from each other in the truth table).

They correspond to *two* minterms with *four* variables. With Boolean algebra, they can be reduced to one term with *three* variables.

	abcd	f	
0	0000	1	
1	0001	1	
2	0010	1	
3	0011	1	
4	0100	1	
5	0101	1	$\bar{a}b\bar{c}d$
6	0110	1	
7	0111	1	
8	1000	1	
9	1001	0	
10	1010	1	
11	1011	0	
12	1100	0	
13	1101	1	$ab\bar{c}d$
14	1110	0	
15	1111	0	

$b\bar{c}d$



$$\bar{a}b\bar{c}d + ab\bar{c}d = b\bar{c}d\underbrace{(\bar{a}+a)}_{=1} = b\bar{c}d$$

What the two frames have in <u>*common*</u> is that b = 1, c = 0 and d = 1; and the reduced term expresses just that.

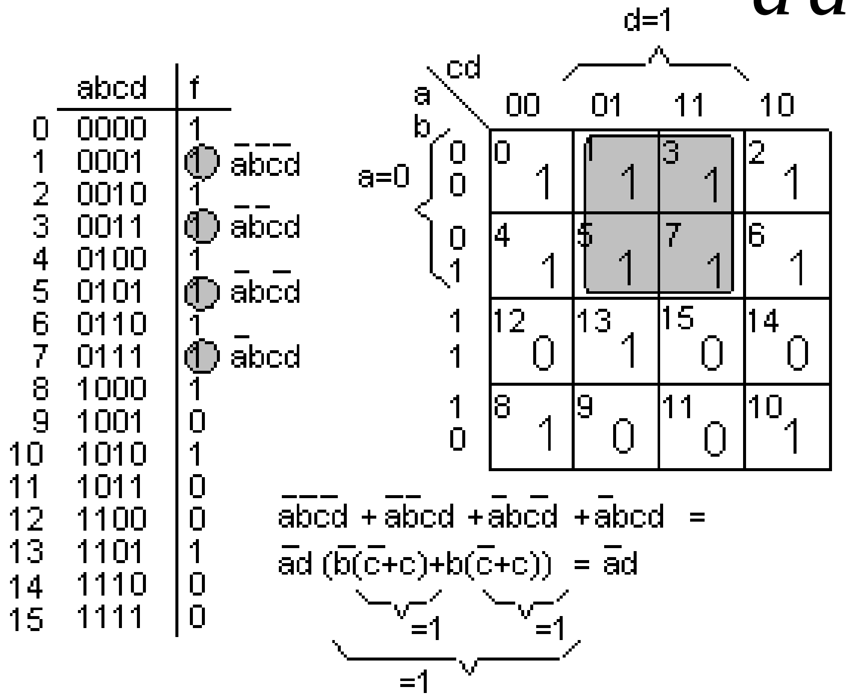
Everywhere in the Karnaugh map where one can find two ones that are "neighbors" (vertically or horizontally) the minterms could be reduced to "what they have in common". This is called a grouping.

Four ”neighbors”

Frames "1" "3" "5" "7" is a group of four frames with "1" that are "neighbors" to each other.

The four minterms could be reduced to a term that expresses what is common for the frames, namely that a = 0 and d = 1.

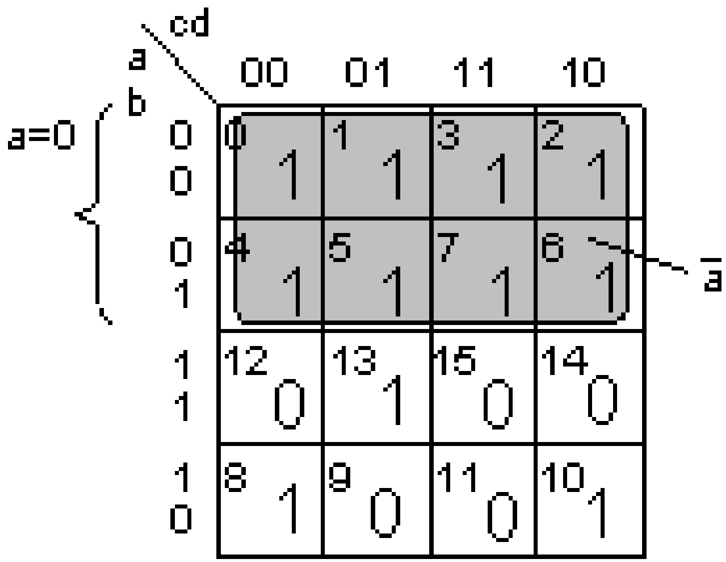
$\bar{a}\,d$



Everywhere in Karnaugh map where one can find such groups of four ones such simplifications can be done, grouping of four.

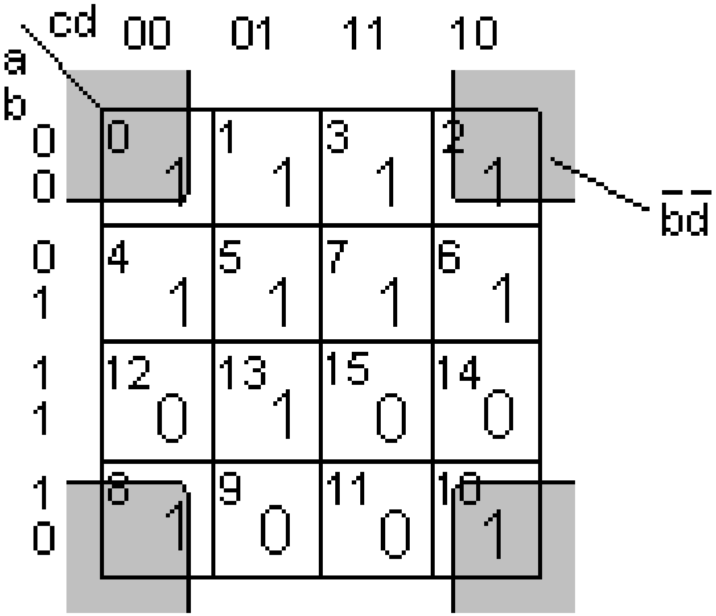
Eight "neighbors"

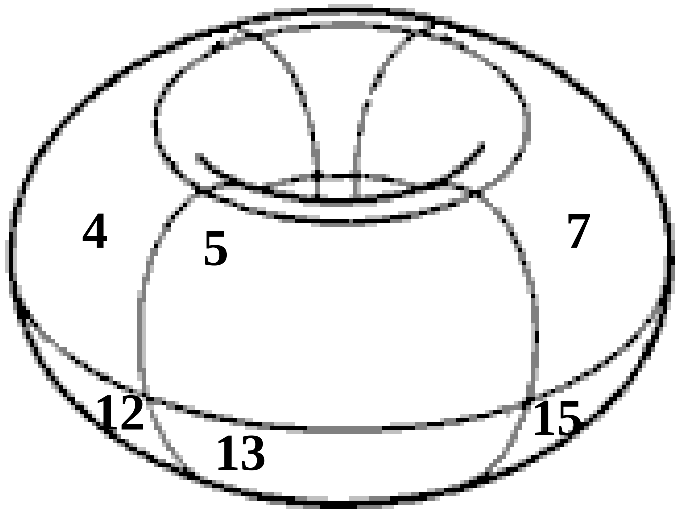
$\overline{a}$



All groups of 2, 4, 8, (... 2 ^{N} ie. powers of 2) frames containing ones can be reduced to a term, with what they have in common, grouping of n.

Karnaugh - torus

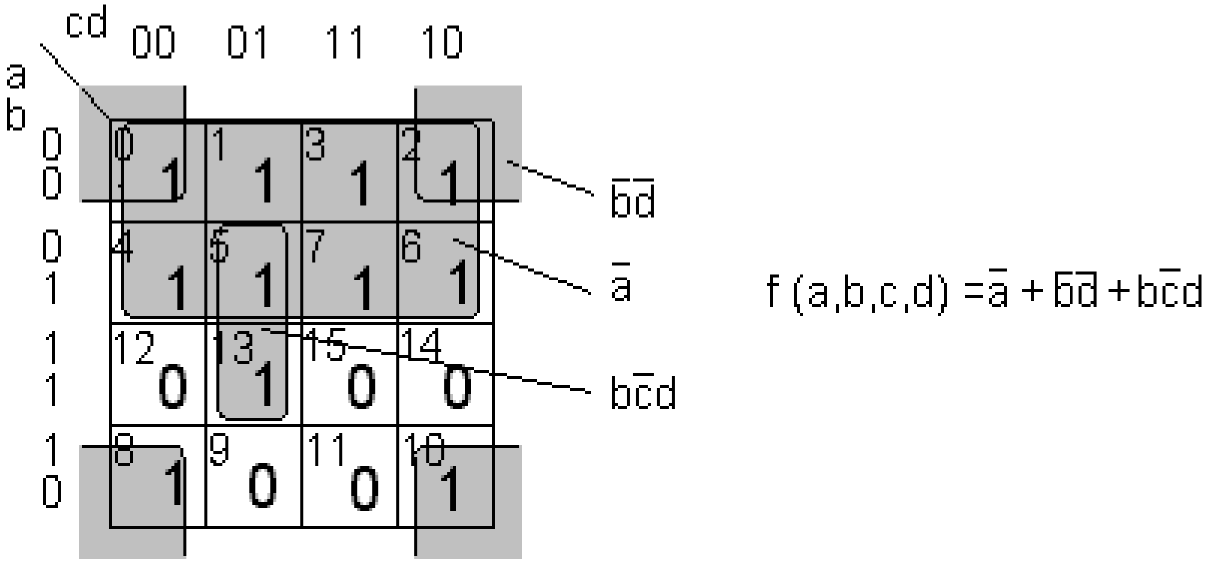




The Karnaugh map should be drawn on a torus (a donut). When we reach an edge, the graph continues from the opposite side! Frame 0 is the "neighbor" with frame 2, but also the "neighbor" with frame 8 which is "neighbor" to frame 10.

The optimal groupings

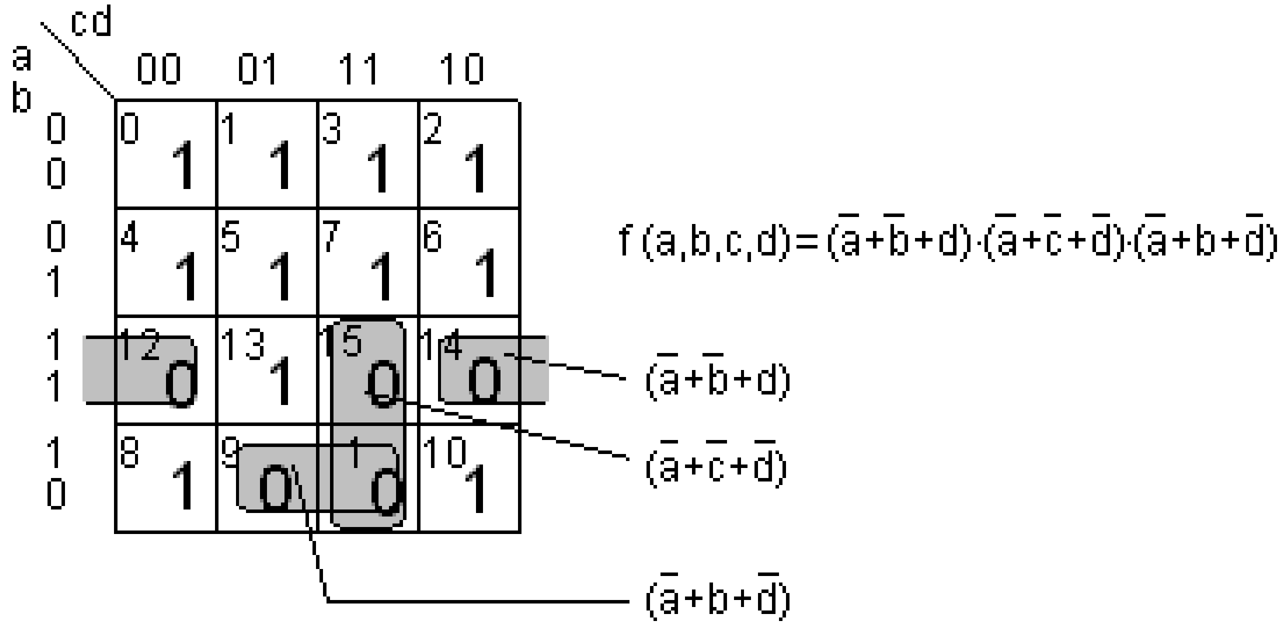
One is looking for the bigest grouping as possible. In the example, there is a grouping with eight ones (frames 0, 1, 3, 2, 4, 5, 7, 6). Corners (0, 2, 8, 10) is a group of four ones.



Two of the frames (0.10) has already been included in the first group, but it does not matter if a minterm is included multiple times. All ones in the logic function must either be in a grouping, or be included as a minterm. The "1" in frame 13 may form a group with "1" in frame 5, unfortunately there are no bigger grouping for this "1".

$$f(a,b,c,d) = \bar{a} + \bar{b}\,\bar{d} + b\,\bar{c}\,d$$

Grouping of "0"



The Karnaugh map is also useful for groupings of 0's. The groupings may include the same number of frames as the case of groupings of 1's. In this example, 0: s are grouped together in pairs with their "neighbors". Maxterms are simplified to what is in common for the frames.

$$f(a,b,c,d) = (\overline{a}+\overline{b}+d)(\overline{a}+\overline{c}+\overline{d})(\overline{a}+b+\overline{d})$$

Maps for other number of variables

	c	b	a
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

c \ ba	00	01	11	10
0	0	1	3	2
1	4	5	7	**6**

	b	a
0	0	0
1	0	1
2	1	0
3	1	1

b \ a	0	1
0	0	1
1	2	3

Karnaugh maps with three and two variables are also useful.

The Karnaugh map can conveniently be used for functions of up to four variables, and with a little practice up to six variables.

Maps for different number of variables

	abcd
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

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

Literals and Product-Terms

- A given product-term consists of several variables (like $a\bar{b}\bar{c}$)
  - Each of the variables may appear complemented or uncomplemented
  - These variables are called literals (like a, $\bar{b}$, $\bar{c}$)

$$f(a,b,c) = a\bar{b}\bar{c} + ab + bc$$

Minterms and product-terms

- A **minterm always contains all variables** of a function
  - Examples of minterms for three variable functions:

    abc, $a\bar{b}\bar{c}$

- A **product-term may contain less variables**
  - Examples of product-terms for three variable functions:

    abc, ab, a

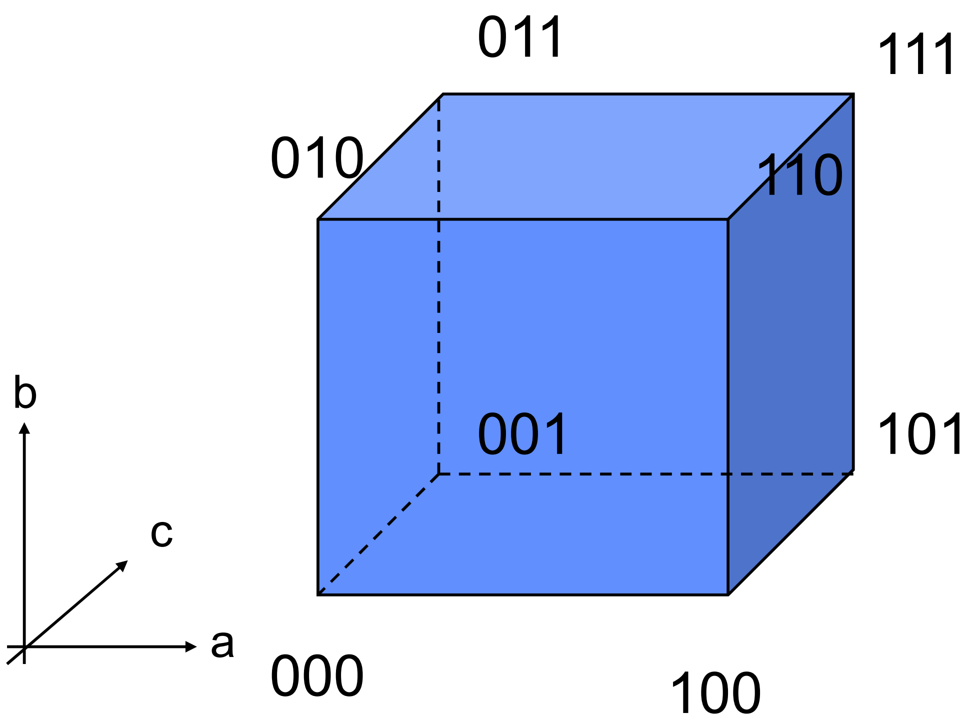
Implicants and Covers

- **Implicant** - A product-term for which the function evaluates to 1
- **Prime implicant** - An implicant which is not contained in any other implicant
  - A prime implicants cannot be expanded into a larger implicant
- **Cover** is a set of implicants which contains all minterms for which the function evaluates to 1

More implicant treminology

- A prime implicant is **essential** if there is a minterm covered by that implicant, but no other prime implicant
  - Essential imlicants will always be included in a cover of a function
- If we can remove an implicant, but all minterms are still covered, then such an implicant is called **redundant**

Example

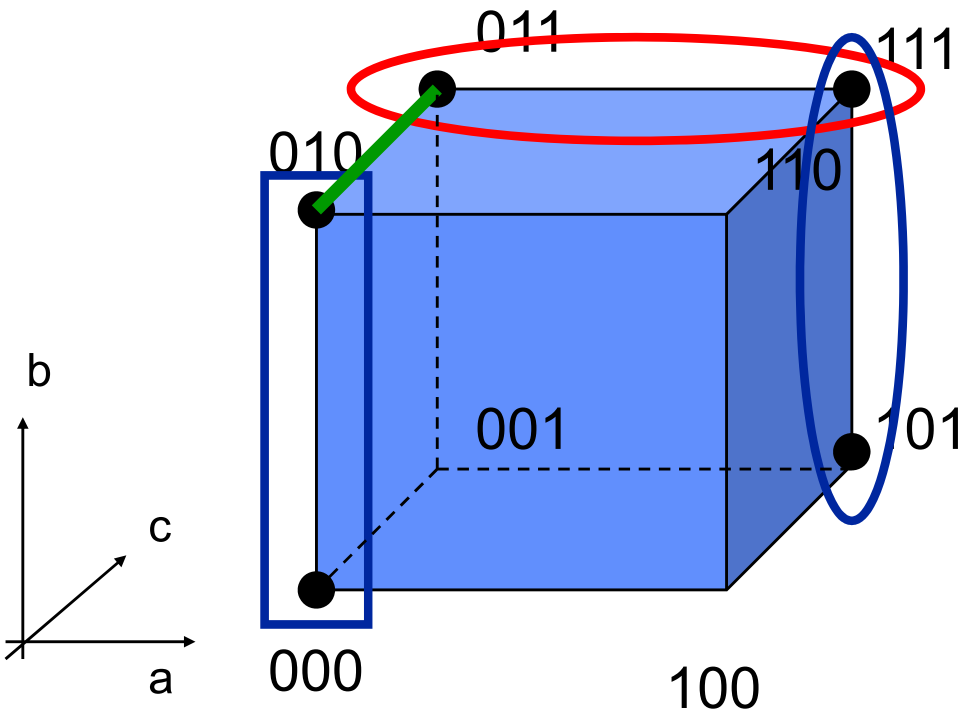


Redundant implicants - both are not necessary to cover the function

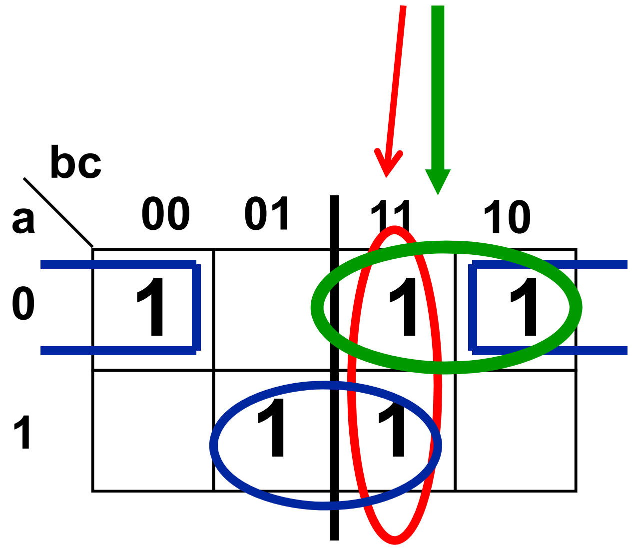
a \ bc	00	01	11	10
0	1		1	1
1		1	1	

$$f\,(a, b, c) = \Sigma m\,(0,2,3,5,7)$$

Example



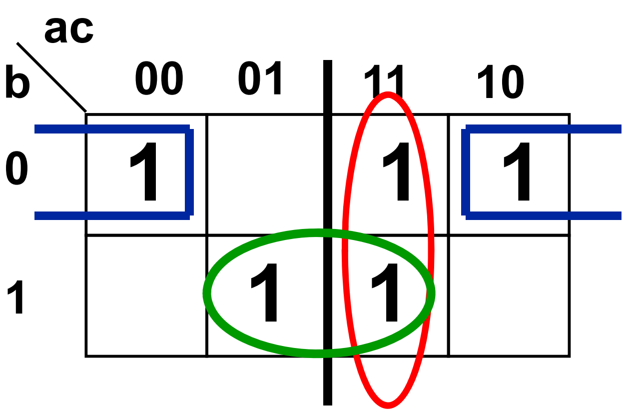
Redundant implicants - both are not necessary to cover the function



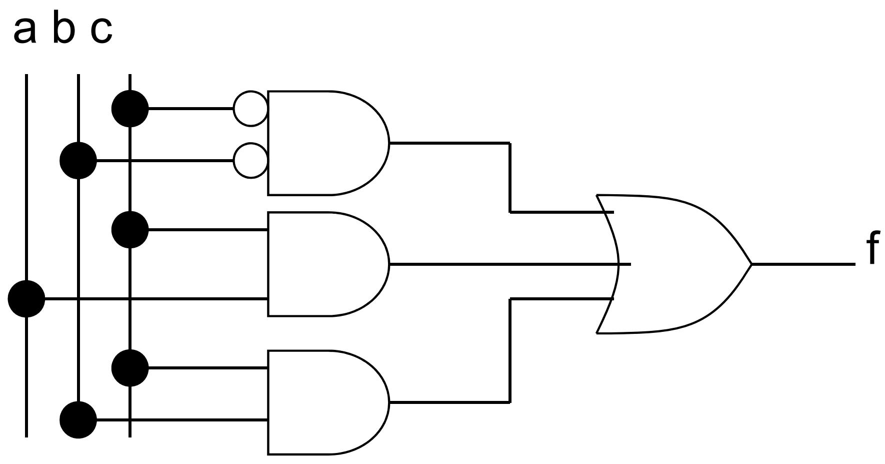
$$f\ (a,\ b,\ c) = \Sigma m\ (0,2,3,5,7)$$

Two-level minimization

Minimum sum-of-product implementation

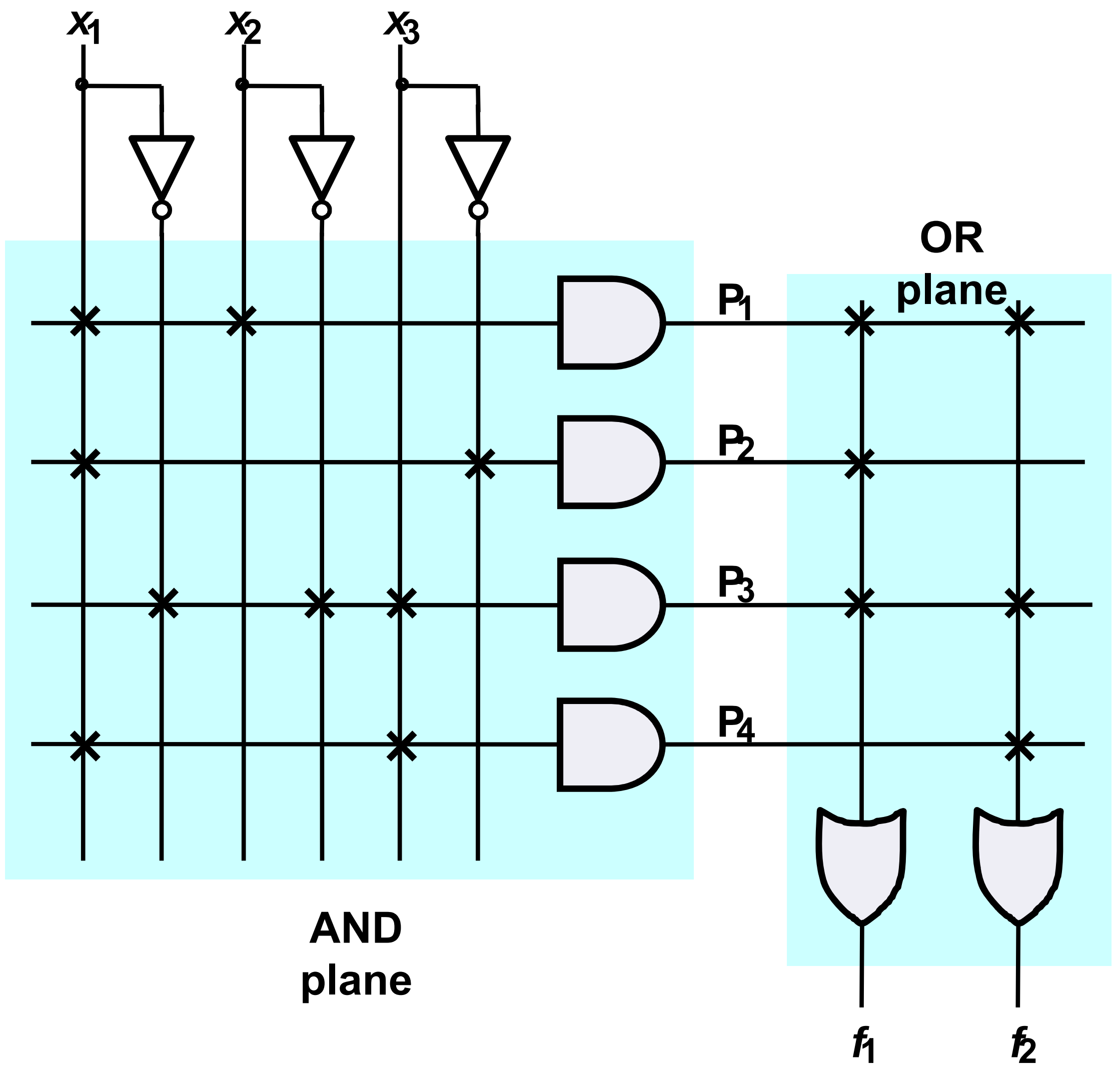


$$f = \bar{b}\bar{c} + ac + bc$$



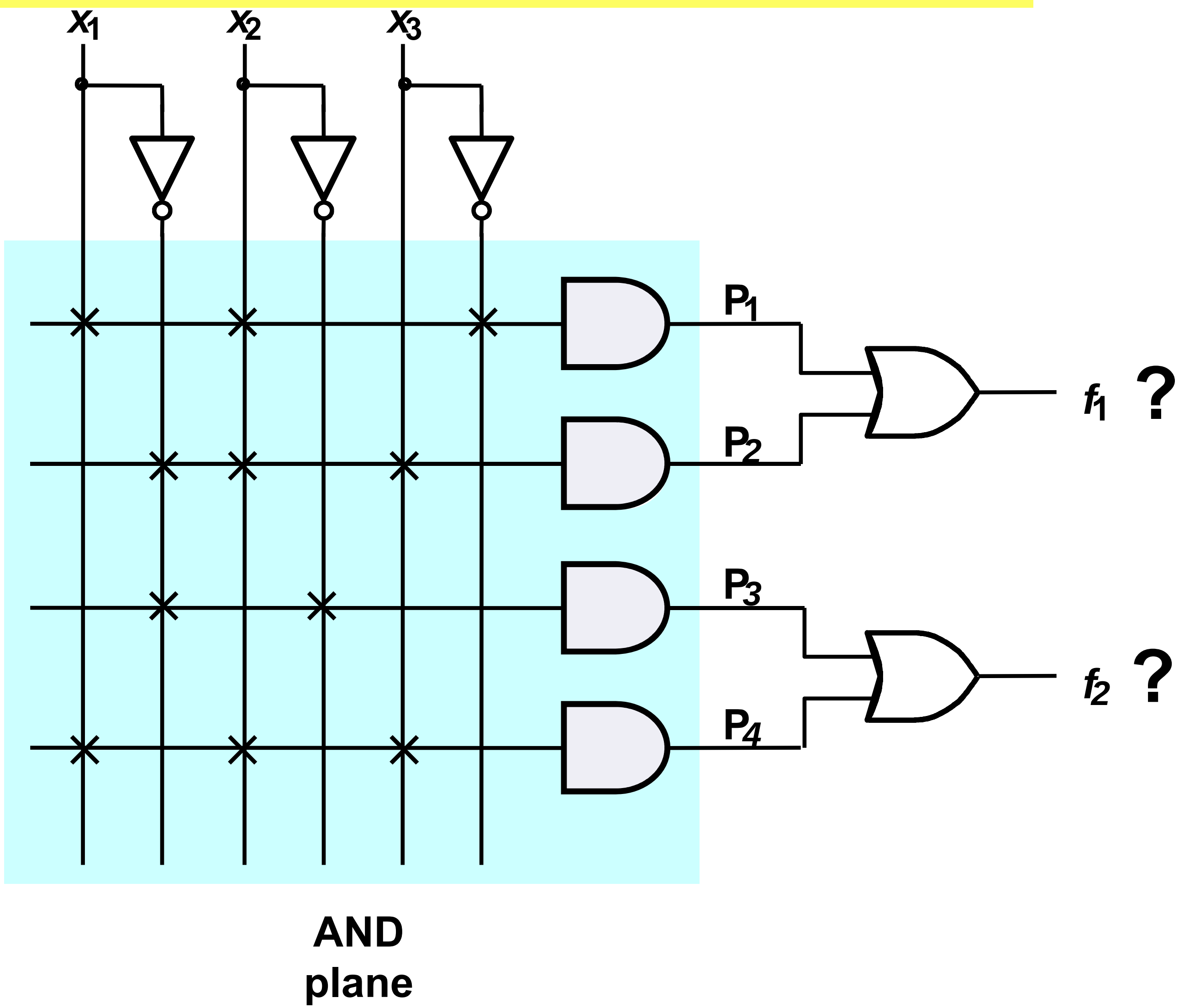
Programmable Logic Array (PLA)

- Both AND and OR arrays are programmable

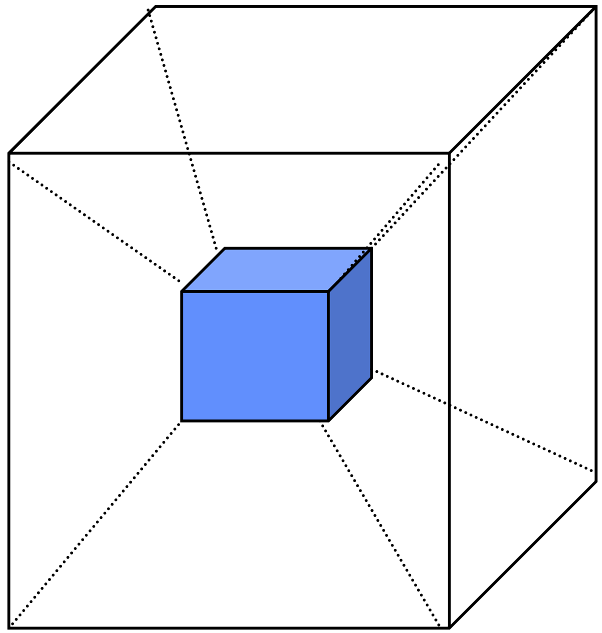


Programmable Array Logic (PAL)

- Only the AND arrays are programmable
- Which functions P_1, P_2, P_3 and P_4 represent ?



Karnaugh map with 4 variables



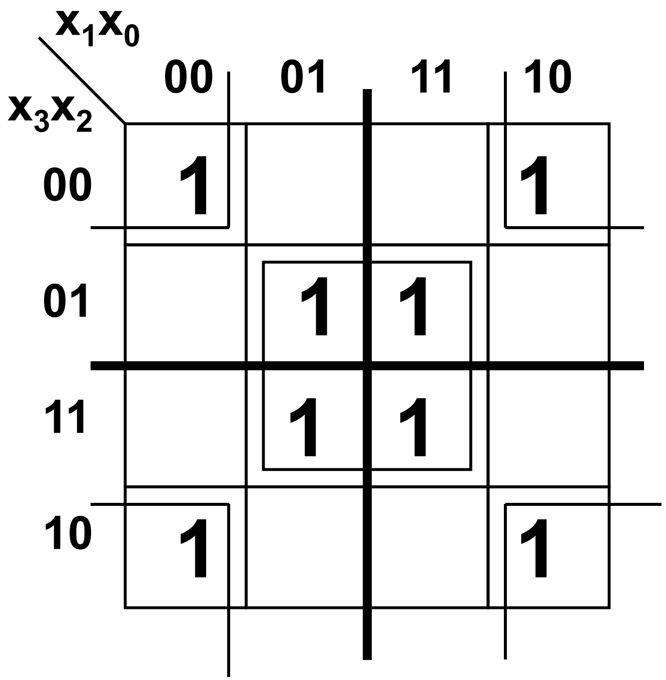
$x_3x_2 \backslash x_1x_0$	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

$$f = x_3$$

We always circle an entire sub-space (as large as possible) !!!

XOR/XNOR function?

If two groups of four minterms can not form a group of eight, the XOR / XNOR function may be helpful.



This is under the assumption that there exists an efficient implementation of the XOR function.

$$f = \overline{x}_2 \overline{x}_0 + x_2 x_0 = \overline{x_2 \oplus x_0}$$

Order of minterms

$$f(x_3, x_2, x_1, x_0) = \sum m(0,3,7,14)$$

MSB LSB

x_3x_2 \ x_1x_0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Example

- Example: f (a, b, c, d) = ?

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	1	1	1
10	0	0	1	0

Example

- Example: f (a, b, c, d) = abc + bd + cd

ab

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	1	1	1
10	0	0	1	0

c

Example

- Example: f (a, b, c, d) = abc + bd + cd

b

d

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	1	1	1
10	0	0	1	0

Example

- Example: f (a, b, c, d) = abc + bd + cd

-

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	1	0
cd { 11	1	1	1	1
10	0	0	1	0

Example

- Example: f (a, b, c, d) = abc + bd + cd

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	1	1	1
10	0	0	1	0

Example

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	**1**	**1**	**1**	**1**
01	**1**	**0**	**1**	**1**
11	**1**	**1**	**1**	**1**
10	**1**	**1**	**1**	**1**

Alternative: Circle 0s

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	1	1	1
10	1	1	1	1

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	1	1	1
10	1	1	1	1

$$\overline{\bar{x}_3\, x_2\, \bar{x}_1\, x_0} =$$

$$x_3 + \bar{x}_2 + x_1 + \bar{x}_0$$

Circle the zeros as zeros are less than ones !!!

Karnaugh map with 5 variables

x_4

Approach A

$x_4 = 0$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

$x_4 = 1$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

Same in both diagrams, independent of x_4. $\overline{x}_3 x_2 x_1 x_0$

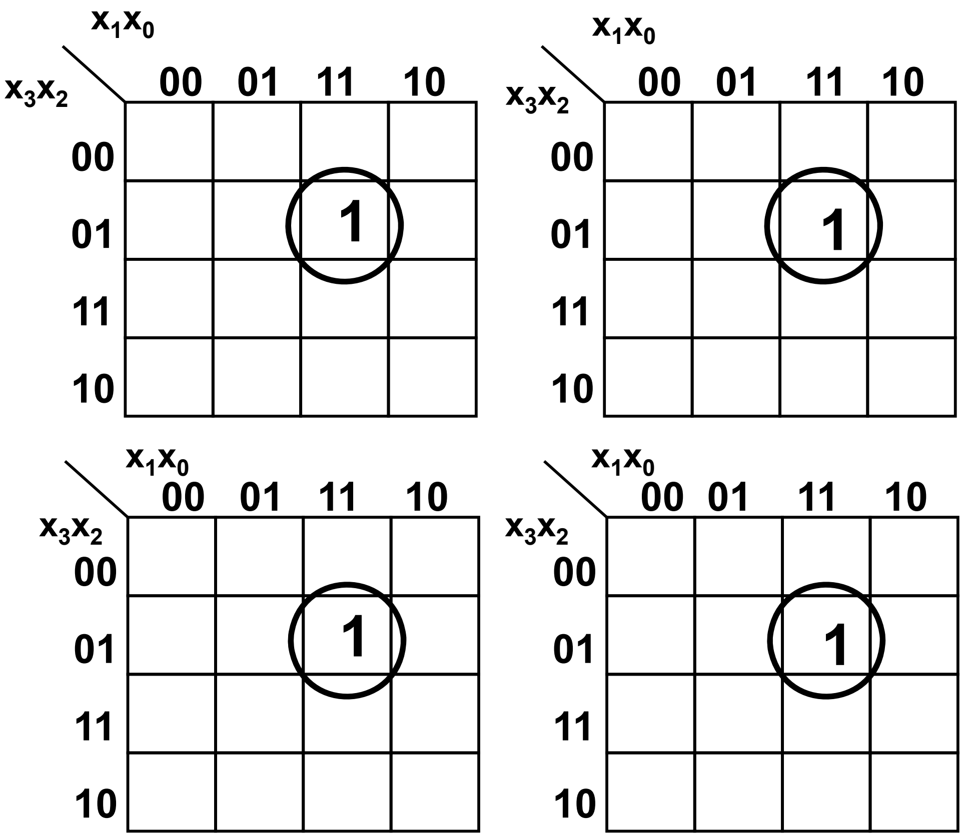
Karnaugh map with 6 variables

$x_4 = 0$ $x_4 = 1$

Approach A

$x_5 = 0$

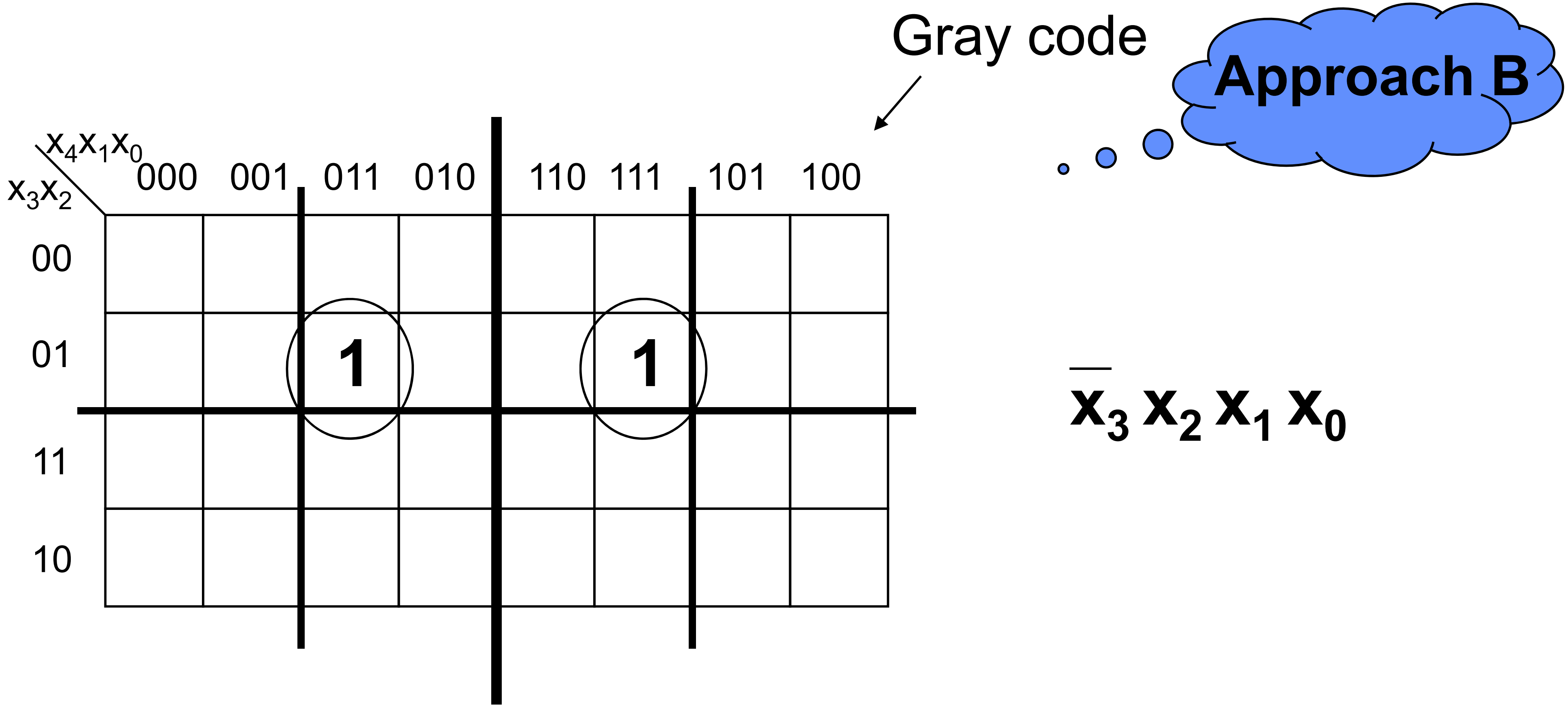
$x_5 = 1$



$\bar{x}_3 x_2 x_1 x_0$

Independent of x_5 and x_4.

Karnaugh map with 5 variables



$$\bar{x}_3\, x_2\, x_1\, x_0$$

Example

Approach B

x_4x_3 \ $x_2x_1x_0$	000	001	011	010	110	111	101	100
00						1		
01								
11								
10						1		

$$\bar{x}_3\, x_2\, x_1\, x_0$$

Karnaugh map with 6 variables

Note!

Approach B

$x_5x_3x_2$ \ $x_4x_1x_0$	000	001	011	010	110	111	101	100
000								
001			1			1		
011								
010								
110								
111								
101			1			1		
100								

$$\overline{x}_3 x_2 x_1 x_0$$

Example

$x_5x_4x_3$ \ $x_2x_1x_0$	000	001	011	010	110	111	101	100
000						1		
001								
011								
010						1		
110						1		
111								
101								
100						1		

Approach B

$$\overline{x}_3 \, x_2 \, x_1 \, x_0$$

Incomplete functions with don’t cares

- Often you can simplify a specification of the logic function knowing that some input combinations never occur
- For these combinations, we use the value "don’t care"
- There are different symbols for "don't care"
  - ’d', 'D', '-', 'ϕ', ’x'

Specification of incomplete functions

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	d	d	d	d
10	0	0	0	1

$x_2\overline{x}_1$

$x_1\overline{x}_0$

(A) SOP implementation

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	d	d	d	d
10	0	0	0	1

$(\overline{x}_1 + \overline{x}_0)$

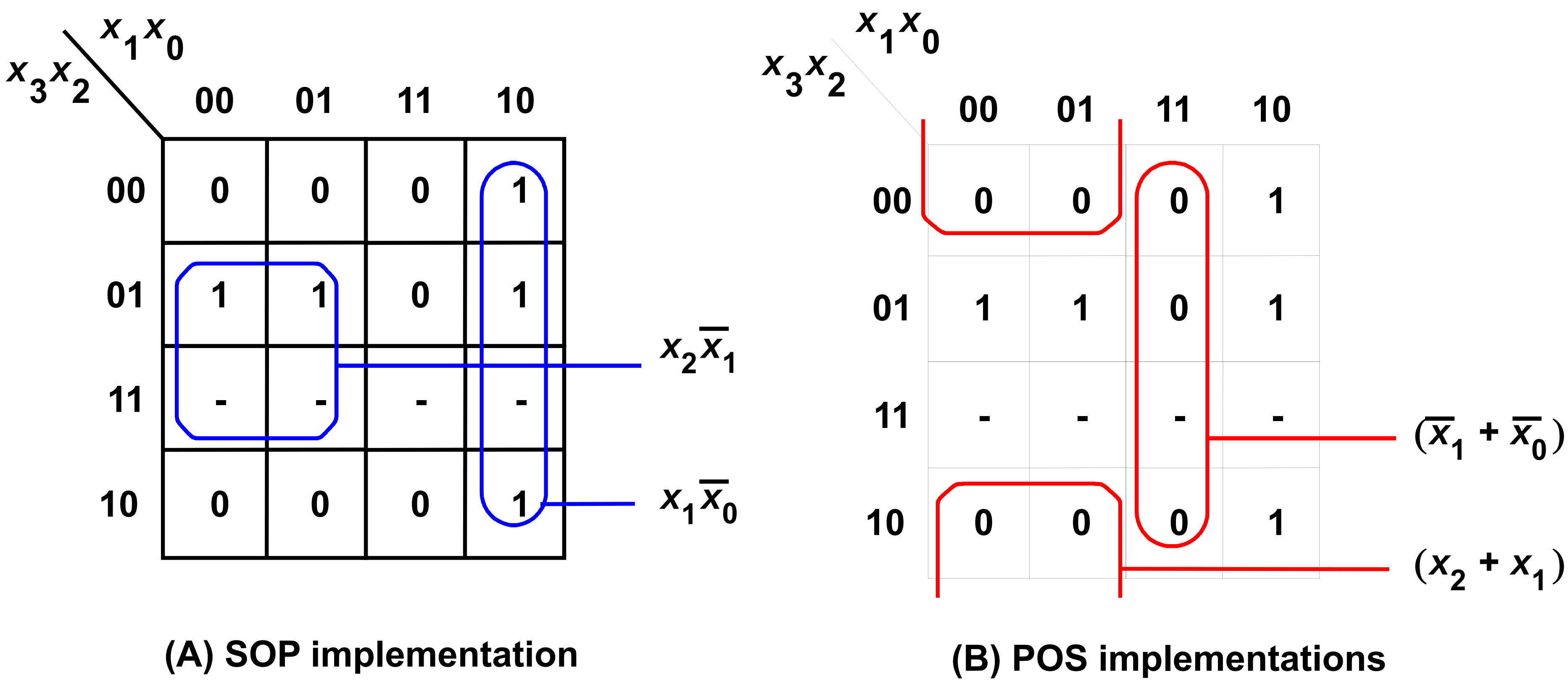
$(x_2 + x_1)$

(B) POS implementations

Two implementations of the function

$f(x_3,\ldots, x_0) = \Sigma m(2, 4, 5, 6, 10) + D(12, 13, 14, 15).$

Alternative notation



Two implementations of the function
$f(x_3,\ldots, x_0) = \Sigma m(2, 4, 5, 6, 10) + D(12, 13, 14, 15).$

Functions with multiple outputs

f_0

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	0	0	0	1
10	1	0	1	1

f_1

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	1	0	0	1
10	1	0	1	1

Different outputs can share implicants!

Functions with multiple outputs

f_0

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	0	0	0	1
10	1	0	1	1

f_1

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	1	0	0	1
10	1	0	1	1

$$f_1 = f_0 + x_3\bar{x}_0$$

Different outputs can share implicants!

Multi-level minimization

Do we need multi-level logic?

- One can realize all the combinational circuits with two-level (AND-OR, OR-AND)
  - The assumption is that all inputs are also availible in the inverted form (as in PAL, PLA)

Why multi-level logic?

- A multi-level implementation may have considerably less gates than a two-level implementation

Two strategies for multi-level logic

1. Factorization
2. Functional Decomposition

Factorization

- Suppose the following function is to be implemented

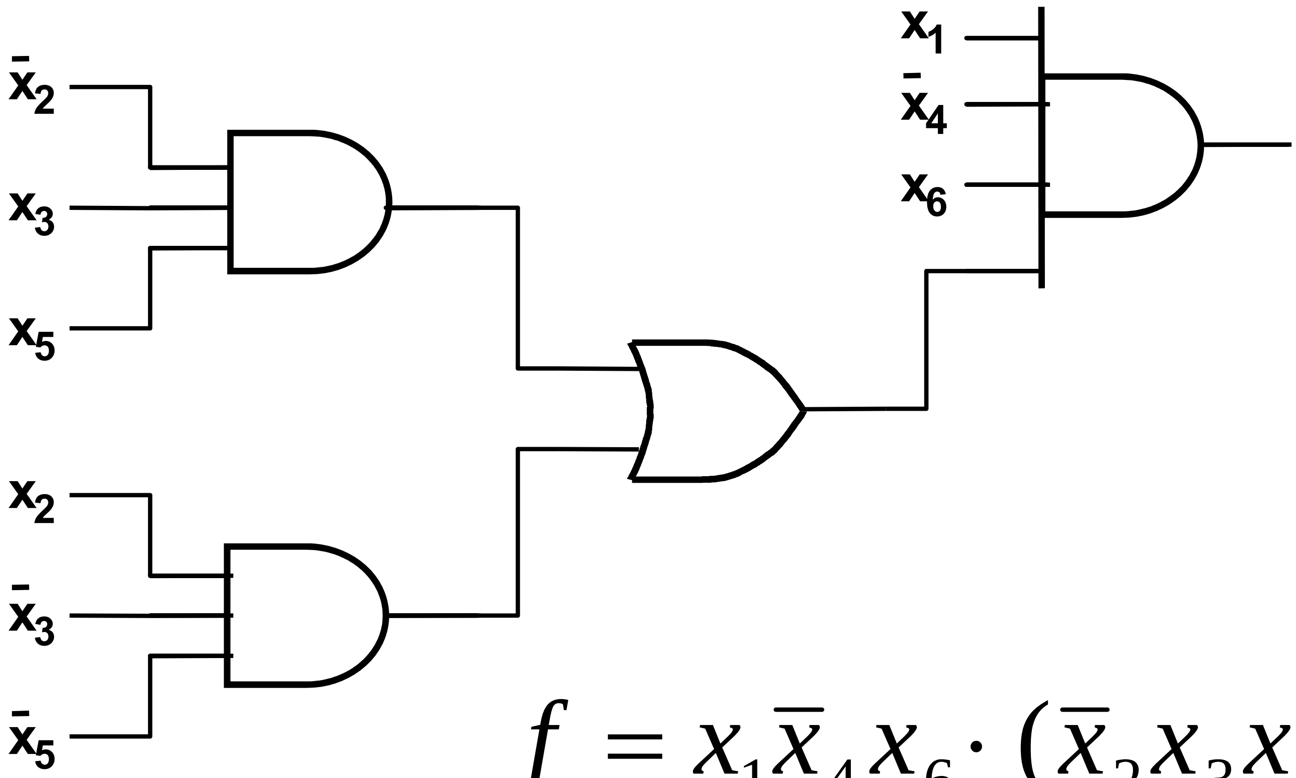
$$f = x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6$$

Factorization

- If we use the distributive rule (L2, s.21, 12a), we can get the following multi-level implementation:

$$f = x_1\bar{x}_2x_3\bar{x}_4x_5x_6 + x_1x_2\bar{x}_3\bar{x}_4\bar{x}_5x_6$$
$$= x_1\bar{x}_4x_6 \cdot (\bar{x}_2x_3x_5 + x_2\bar{x}_3\bar{x}_5)$$

Factorization



$$f = x_1 \bar{x}_4 x_6 \cdot (\bar{x}_2 x_3 x_5 + x_2 \bar{x}_3 \bar{x}_5)$$

Functional decomposition

- One can often reduce the complexity of a logic function by *reusing its sub-functions* several times
- For implementation it means to *share sub-circuits* in its construction

Functional Decomposition

- How can the following function be implemented?

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

Functional Decomposition

- Factorization gives us

$$
\begin{aligned}
f &= \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_4 + \bar{x}_1 \bar{x}_2 x_4 \\
&= (\underbrace{\boxed{\bar{x}_1 x_2 + x_1 \bar{x}_2}}_{\text{XOR}})_{=\,g}\, x_3 + (\underbrace{\boxed{x_1 x_2 + \bar{x}_1 \bar{x}_2}}_{\text{XNOR}})_{=\,\bar{g}}\, x_4
\end{aligned}
$$

- If we define a sub-function *g* as

$$g = \bar{x}_1 x_2 + x_1 \bar{x}_2$$

Functional Decomposition

- So, we get

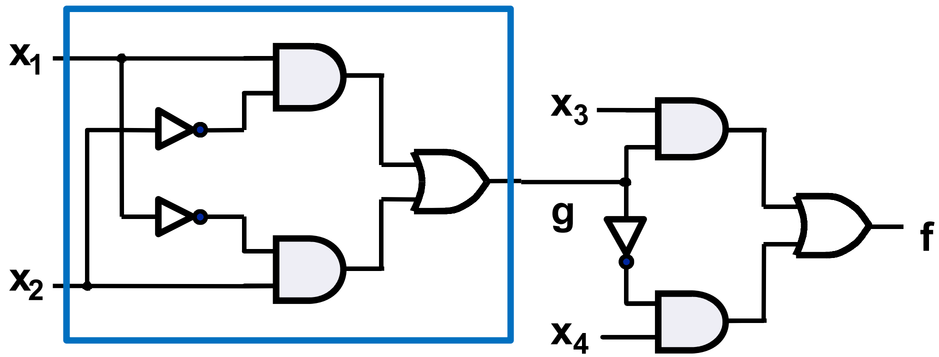
$$f = gx_3 + \overline{g}x_4$$

- where

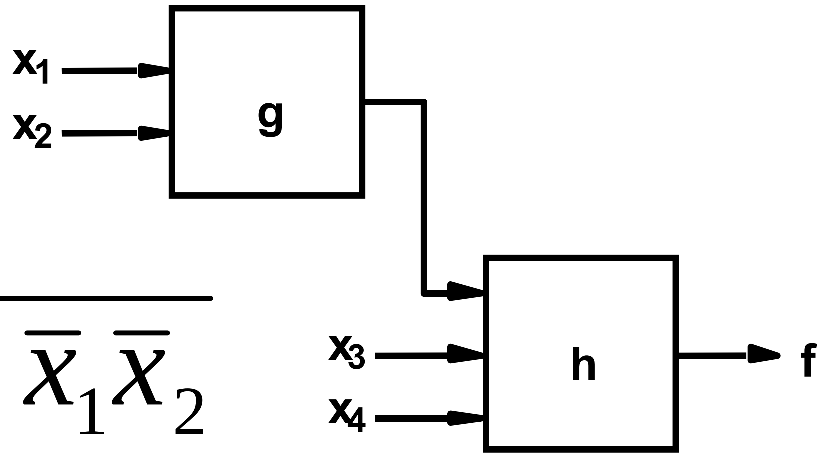
$$g = \overline{x}_1 x_2 + x_1 \overline{x}_2 = \overline{x_1 x_2 + \overline{x}_1 \overline{x}_2}$$

Functional Decomposition

Implementation



$$f = gx_3 + \bar{g}x_4$$



$$g = \bar{x}_1 x_2 + x_1 \bar{x}_2 = \overline{x_1 x_2 + \bar{x}_1 \bar{x}_2}$$

Algorithms for minimization

- Karnaugh map minimization gives a good insight into how to minimize logic functions
- But to minimize the complex functions with the help of computer, there are better algorithms
- Chapter 4.9 and 4.10 in Brown/Vranesic provides an introduction to the minimization algorithms (for the interested student)

Summary

- Karnaugh maps are a good tool for minimizing logic functions with a few variables
- There are algorithms for both two-level and multi-level minimization
- Next lecture: BV pp. 250-280