Boolean Algebra and logic gates

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid

Outline

- Postulates and fundamental properties of Boolean Algebra
- Boolean functions and expressions
- Logic gates. Digital technologies. Implementation of logic functions
- Minimization of logic functions

Boolean Algebra

- Mathematical foundations of digital circuits
- Called Boolean Algebra after its inventor, George Boole
  - "*An Investigation of the Laws of Thought*" (1854)
- An algebra is defined by a set of elements with some operations. In our case:
  - B = {0, 1}
  - Φ = {+, •}

Postulates of Boolean Algebra

- Internal composition law
  - $\forall\ a, b \in B \Rightarrow a + b \in B,\ a \bullet b \in B$
- Neutral elements
  - $\forall\ a \in B \Rightarrow \exists$ neutral elements (0 y 1 respectively)

    $a + 0 = a$

    $a \bullet 1 = a$
- Commutative property
  - $\forall\ a, b \in B \Rightarrow$ $a + b = b + a$

    $a \bullet b = b \bullet a$
- Distributive property
  - $\forall\ a, b, c \in B \Rightarrow$ $a + b \bullet c = (a + b) \bullet (a + c)$

    $a \bullet (b + c) = a \bullet b + a \bullet c$

Postulates of Boolean Algebra

- Inverse or complementary element
  - $\forall a \in B \Rightarrow \exists\ \bar{a} \in B$

$$a + \bar{a} = 1$$

$$a \bullet \bar{a} = 0$$

Fundamental properties of Boolean Algebra

- Duality: Every valid law has its dual, which is obtained by replacing $0 \leftrightarrow 1$ and $+ \leftrightarrow \bullet$
- Idempotence
  - $\forall\, a \in B \Rightarrow$ $a + a = a$
    $a \bullet a = a$
  - Proof:

$$a = a + 0 = a + a\bar{a} = (a + a)(a + \bar{a}) = (a + a) \bullet 1 = a + a$$

- Anihilation
  - $\forall\, a \in B \Rightarrow$ $a + 1 = 1$
    $a \bullet 0 = 0$

Fundamental properties of Boolean Algebra

- The basic operations can be defined from the previous properties

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

a	$\bar{a}$
0	1
1	0

- Truth table: provides the value of the function for all possible combinations of input values

Fundamental properties of Boolean Algebra

- Involution
  - $\forall\, a \in B \Rightarrow \quad \overline{\overline{a}} = a$
- Absorption
  - $\forall\, a, b \in B \Rightarrow \quad a + ab = a$
    $a\,(a+b) = a$
  - Proof:

$$a + ab = a \bullet 1 + ab = a(1 + b) = a \bullet 1 = a$$

- Associative property
  - $\forall\, a, b, c \in B \Rightarrow \quad (a + b) + c = a + (b + c)$
    $(a \bullet b) \bullet c = a \bullet (b \bullet c)$

Fundamental properties of Boolean Algebra

- De Morgan laws:
  - ∀ a, b∈ B ⇒ $\overline{a+b} = \bar{a}\,\bar{b}$

    $\overline{a \bullet b} = \bar{a} + \bar{b}$
  - Proof:

    $(a+b) + \bar{a}\,\bar{b} = (a+b+\bar{a})(a+b+\bar{b}) = 1 \bullet 1$

    therefore (a+b) is the inverse of $\bar{a}\,\bar{b}$

Boolean Functions and Expressions

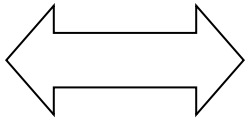
- Definitions:
  - A logic or boolean variable is any element $x \in B = \{0, 1\}$
  - A literal is a variable or its inverse
  - Logic or Boolean Function:

$$f : B^n \rightarrow B$$
$$(x_1, x_2, \ldots, x_n) \rightarrow y$$

Representation of logic functions

- Expression

$$f(a, b) = a + b$$



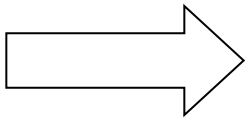
- Truth table

a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	1

Deriving the truth table from a expression

- Evaluate the expression for every combination of input values

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

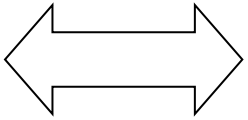


a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Minterm function

- Expression: a product of all variables, either inverted or not
- Truth table: has a 1 for one combination and 0 elsewhere
- Example:

$$f(a,b,c) = \bar{a}b\bar{c} = m_2$$



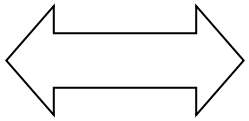
a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

- Rule to obtain the expresion:
  - 0 → inverted variable
  - 1 → non-inverted variable

Maxterm function

- Expression: a sum of all variables, either inverted or not
- Truth table: has a 0 for one combination and 1 elsewhere
- Example:

$$f(a,b,c) = (a + \bar{b} + c) = M_2$$



a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Rule to obtain the expresion:
  - 0 → inverted variable
  - 1 → non-inverted variable

ATTENTION: minterms use the opposite rule!

Shannon's Expansion Theorem

- Any boolean function can be decomposed in any of the following forms

$$f(x_1, x_2, \ldots, x_n) = \bar{x}_i \, f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i \, f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

$$f(x_1, x_2, \ldots, x_n) = \left[\bar{x}_i + f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)\right]\left[x_i + f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)\right]$$

- Proof:

$$x_i = 0 \Rightarrow f(x_1, x_2, \ldots, x_n) = 1 \bullet f(x_1, \ldots, 0, \ldots, x_n) + 0 \bullet f(x_1, \ldots, 1, \ldots, x_n) =$$
$$= f(x_1, \ldots, 0, \ldots, x_n)$$

$$x_i = 1 \Rightarrow f(x_1, x_2, \ldots, x_n) = 0 \bullet f(x_1, \ldots, 0, \ldots, x_n) + 1 \bullet f(x_1, \ldots, 1, \ldots, x_n) =$$
$$= f(x_1, \ldots, 1, \ldots, x_n)$$

  - The proof of the other form is given by duality

Corollary of Shannon's Expansion Theorem

- Apply the Theorem recursively:

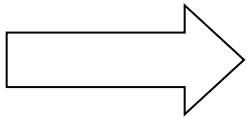
$$
\begin{aligned}
f(a,b,c) &= \bar{a}\,f(0,b,c) + a\,f(1,b,c) = \\
&= \bar{a}(\bar{b}\,f(0,0,c) + b\,f(0,1,c)) + a(\bar{b}\,f(1,0,c) + b\,f(0,1,c)) = \\
&= \bar{a}\bar{b}\,f(0,0,c) + \bar{a}b\,f(0,1,c)) + a\bar{b}\,f(1,0,c) + ab\,f(0,1,c) = \\
&= \bar{a}\bar{b}\bar{c}\,f(0,0,0) + \bar{a}\bar{b}c\,f(0,0,1) + \bar{a}b\bar{c}\,f(0,1,0) + \bar{a}bc\,f(0,1,1) + \\
&+ a\bar{b}\bar{c}\,f(1,0,0) + a\bar{b}c\,f(1,0,1) + ab\bar{c}\,f(1,1,0) + abc\,f(1,1,1) = \\
&= \sum_{3} m_i k_i
\end{aligned}
$$

- A function can be expressed as the sum of all minterms (m_i) multiplied by a coefficient (k_i) equal to the value of the function when each variable is substituted by 0, if the variable appears as inverted in the minterm, or 1 otherwise.

First canonical form

- A function can be expressed as the sum of all minterms for which the function evaluates to 1

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

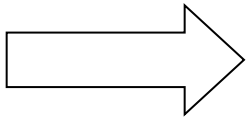


$$f(a,b,c) = \sum_{3}(0,2,5) = \sum_{3} m(0,2,5) =$$
$$= \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c$$

Second canonical form

- A function can be expressed as the product of all maxterms for which the function evaluates to 0

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



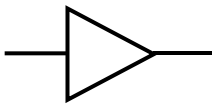
$$f(a,b,c) = \prod_{3}(1,3,4,6,7) = \prod_{3} M(1,3,4,6,7) =$$
$$= (a+b+\bar{c})(a+\bar{b}+\bar{c})(\bar{a}+b+c)$$
$$(\bar{a}+\bar{b}+c)(\bar{a}+\bar{b}+\bar{c})$$

ATTENTION: minterms use the opposite rule!

Logic gates

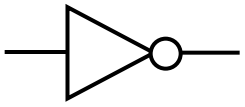
- Logic gates are electronic circuits that implement the basic functions of Boolean Algebra
- There is a symbol for each gate
- Identity
  $z = a$

a	a
0	0
0	1



- NOT gate or inverter
  $z = \bar{a}$

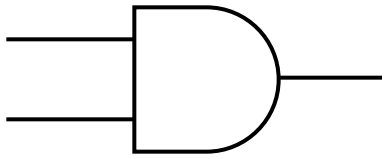
a	$\bar{a}$
0	1
1	0



AND and OR gates

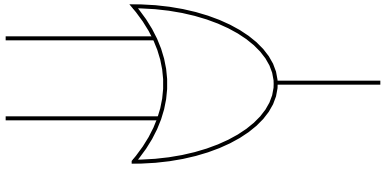
- AND gate
  $z = a \bullet b$

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1



- OR gate
  $z = a + b$

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

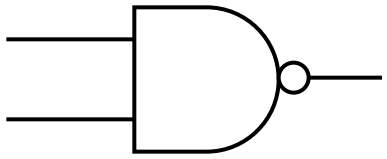


NAND and NOR gates

- NAND gate

$$z = \overline{a \bullet b} = \bar{a} + \bar{b}$$

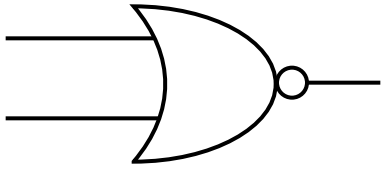
a	b	$\overline{a \bullet b}$
0	0	1
0	1	1
1	0	1
1	1	0



- NOR gate

$$z = \overline{a + b} = \bar{a}\,\bar{b}$$

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0

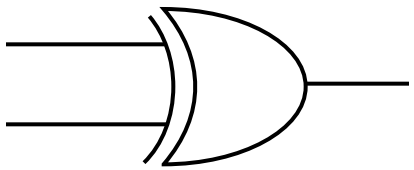


XOR and XNOR gates

- XOR (Exclusive-OR) gate

$$z = a \oplus b = \bar{a}b + a\bar{b} = (\bar{a} + \bar{b})(a + b)$$

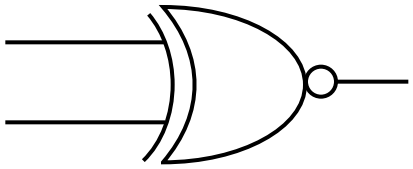
a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0



- XNOR (Exclusive-NOR) gate

$$z = \overline{a \oplus b} = ab + \bar{a}\bar{b} = (\bar{a} + b)(a + \bar{b})$$

a	b	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1

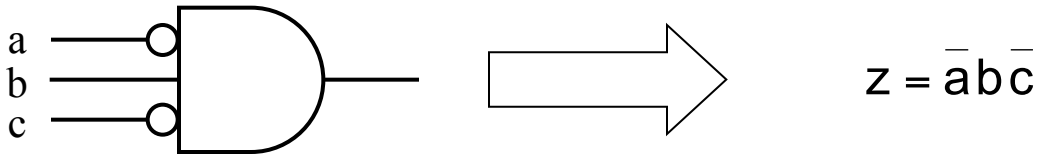


Generalization to n inputs

	Output value	
Gate	0	1
AND	Any input = 0	All inputs = 1
OR	All inputs = 0	Any input = 1
NAND	All inputs = 1	Any input = 0
NOR	Any input = 1	All inputs = 0
XOR	There is an even number of inputs = 1	There is an odd number of inputs = 1
XNOR	There is an odd number of inputs = 1	There is an even number of inputs = 1

Other symbols

- A circle at an input or output indicates inversion



Digital technologies

- Logic gates are electronic circuits
- Logic levels (0 or 1) are represented by means of a voltage level
- "Positive logic" is generally used
  - High voltage (5V, 3.3V, 2.5 V, etc.) → 1
  - Low voltage (0V) → 0
- There are many technolgies, depending on the way logic gates are implemented and the features obtained by every implementation

Logic families

- The set of basic digital components, such as logic gates and others that will be studied along the course, is commonly known as *74 Series of Family*
- There are many subfamilies:
  - According to the operating temperature range:
    - 74 Series : 0º a 70º
    - 54 Series : -55º a 125º
  - According to technology:
    - LS
    - ALS
    - F
    - HC
    - AHC
    - G
    - ....

Logic families

- Component denomination:
  - <Series><Subfamily><Component>
- Example: 74HC00
  - 74 Series: conventional temperature range
  - HC subfamily (High speed CMOS)
  - Component 00: 4 NAND gates, 2 inputs
- Important: members of different subfamilies are not compatible
  - Components of different subfamiles should not be assembled together in a circuit

Datasheets

National Semiconductor

June 1989

54LS00/DM54LS00/DM74LS00
Quad 2-Input NAND Gates

General Description

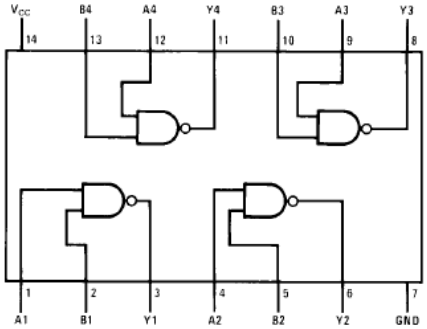
This device contains four independent gates each of which performs the logic NAND function.

Features

- Alternate Military/Aerospace device (54LS00) is available. Contact a National Semiconductor Sales Office/ Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6439–1

Order Number 54LS00DMQB, 54LS00FMQB, 54LS00LMQB, DM54LS00J, DM54LS00W, DM74LS00M or DM74LS00N
See NS Package Number E20A, J14A, M14A, N14A or W14B

54LS00/DM54LS00/DM74LS00 Quad 2-Input NAND Gates

Function Table

$Y = \overline{AB}$

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

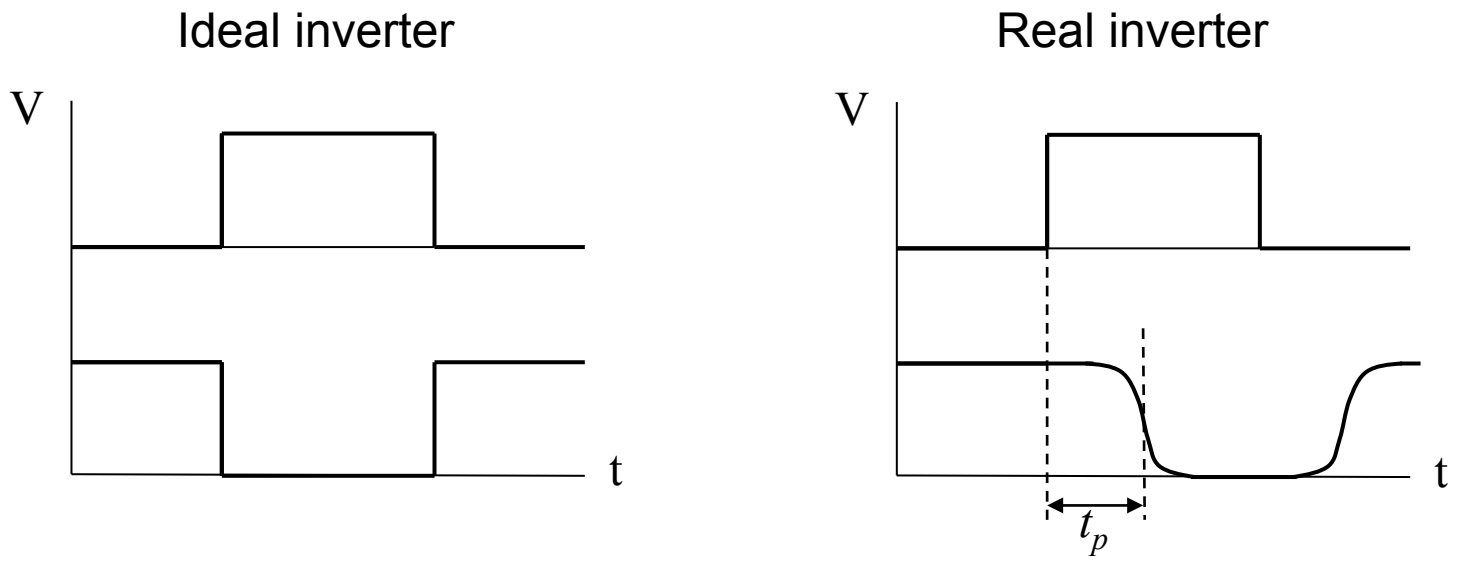
H = High Logic Level
L = Low Logic Level

Features of digital technologies

- Main features:
  - Operating temperature range
  - Voltage supply
  - Noise margin (voltage intervals associated to each logic value)
  - Switching delay
  - Power
  - Others
- Each technology or subfamily shows different values for these features

Delays

- Logic gates do not switch immediately



- Delay limits the operating speed of circuits

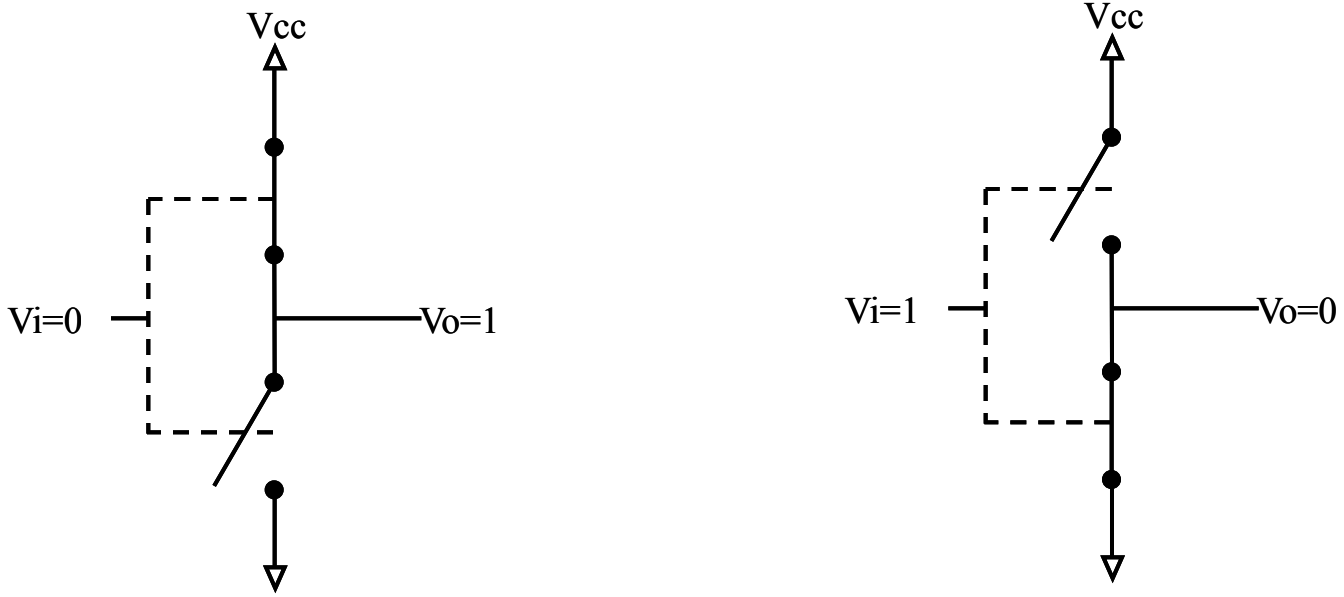
Power

- Logic gates consume energy:
  - Static power: power taken when the circuit is on, without logic activity
  - Dynamic power: power taken when logic gates commute
- In CMOS technology (currently the most widely used), static power consumption is very small. However,
  - Modern circuits can have more than 10^8 logic gates!
  - Dynamic power consumption is proportional to operating frequency
- Power consumption is an important problem:
  - Consumed energy is transformed into heat, which must be dissipated. Dissipation can be difficult if the circuit has a large power consumption
  - In portable devices, battery size and weight are at a premium

CMOS technology

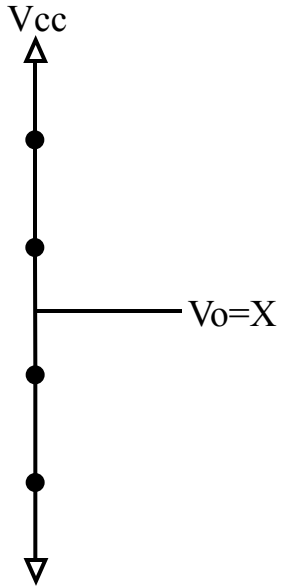
- CMOS (Complementary Metal Oxide Semiconductor) technology is the most widespread technology today
- Based upon:
  - MOS transistors: voltage controlled switches
  - Complementary: every transistor or switch has a complementary one. When a switch is open, its complementary switch is closed and viceversa

CMOS Inverter

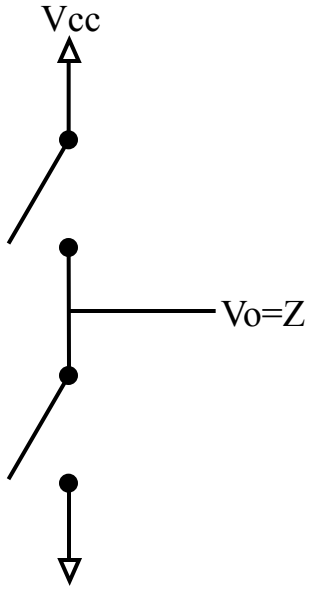


Metalogic values

- There are situations that do not correspond to any logic value
  - Shortcircuit (X)

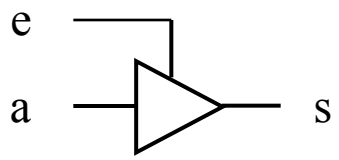


High impedance or tristate (Z)



Tristate buffer

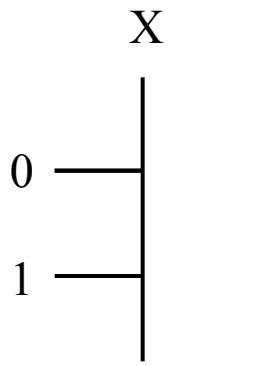
- A special sort of logic gate that can put its output in high impedance



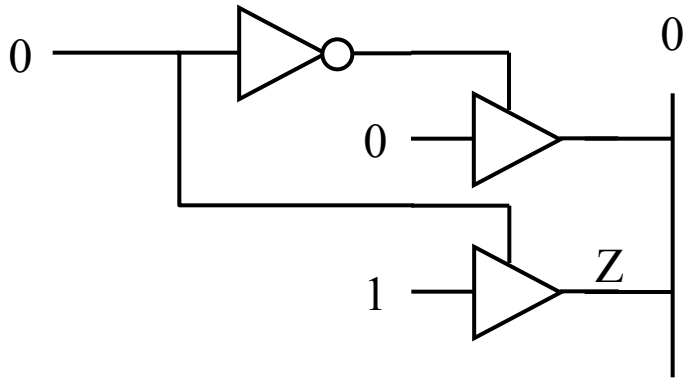
e	a	s
0	0	Z
0	1	Z
1	0	0
1	1	1

Tristate buffer

- Tristate buffers are useful to avoid shortcircuits when connecting multiple signals to a common point



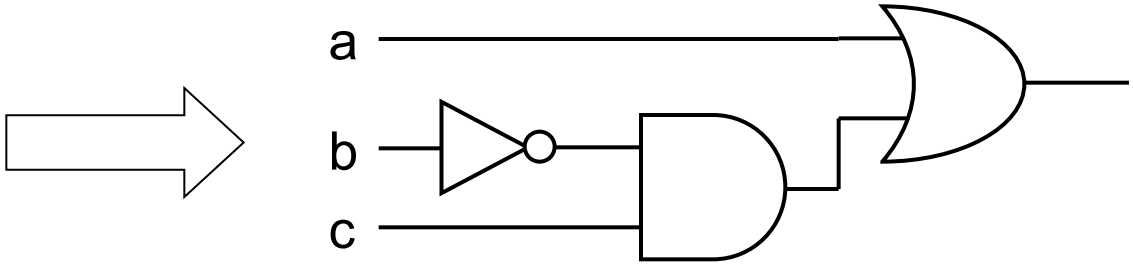
Shortcircuit!



Implementation of a logic function with logic gates

- Take the function expression and substitute logic operations by logic gates
- Example:

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



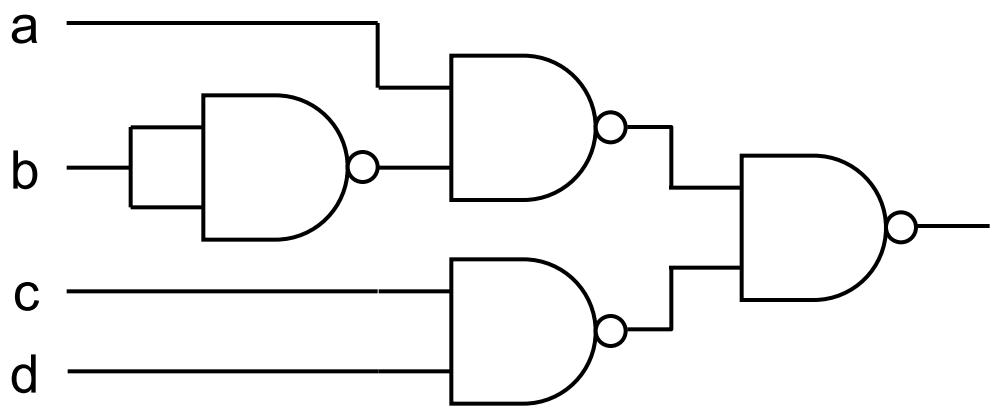
Funtionally complete operations

- A set of functions is said to be functionally complete (or universal) if and only if every logic function can be expressed entirely by means of operations of this set
  - {AND} is ***not*** a complete set
  - {AND, NOT} is a complete set
  - {OR, NOT} is a complete set
  - {NAND} is a complete set
  - {NOR} is a complete set
- {NAND} and {NOR} sets are very useful, since they allow implementing any logic function with a single type of logic gates

Circuit implementation with NAND gates

- Direct application of De Morgan laws
- Example: $f(a,b,c) = a\bar{b} + cd =$

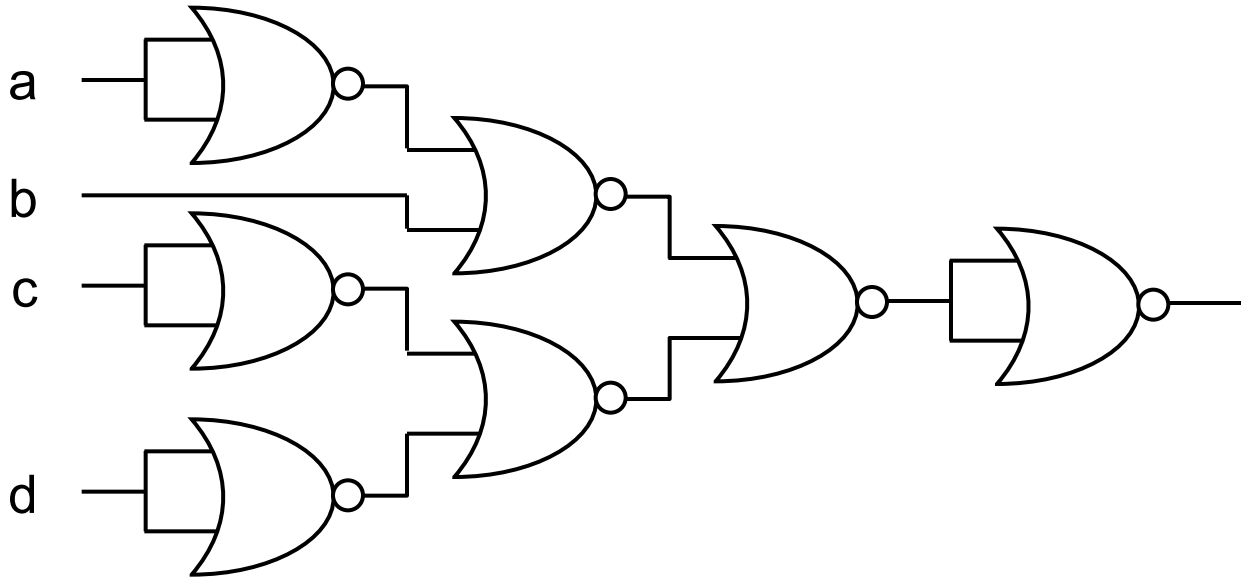
$$= \overline{\overline{a\bar{b} + cd}} = \overline{\overline{a\bar{b}} \bullet \overline{cd}}$$



Circuit implementation with NOR gates

- Direct application of De Morgan laws
- Example: $f(a,b,c) = a\bar{b} + cd =$

$$= \overline{\overline{a\bar{b}} + \overline{cd}} = \overline{\bar{a} + b + \bar{c} + \bar{d}}$$



Minimization of logic functions

- A logic function has many equivalent expressions
  - The simplest expression will lead to a better implementation
- Optimization criteria:
  - Area or size:
    - Smaller logic gate count
    - Logic gates with less inputs
  - Speed or delay:
    - Smaller number of logic gates from input to output
- We will focus on area optimization

Minimization of logic functions

- Optimization methods
  - Manual: direct application of Boolean Algebra laws
    - Very difficult, unsystematic
  - Two level: find an optimal expression as a sum of products or a product of sums
    - There are systematic optimal solutions
    - Feasible in a manual way (for few variables) or with the help of a computer
  - Multilevel
    - Better solution, but much more difficult
    - Feasible only with the help of a computer

Karnaugh map method

- Two level optimization method
- Can be applied in a manual way up to 6 variables
- Based on the Adjacency Property
  - $\forall E, x \in B \Rightarrow Ex + E\bar{x} = E(x + \bar{x}) = E$

    $(E + x)(E + \bar{x}) = E + (x \bullet \bar{x}) = E$ (dual)

  - Two terms are adjacent if they are identical except for one literal, that appears inverted in one term and non-inverted in the other term
  - The two terms are simplified into a single one by eliminating the differentiating literal

Applying the Adjacency Property

- Example:

$$f(a,b,c) = \sum_3 (0,1,2,3,7) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}bc + abc =$$

$$= \bar{a}\bar{b} + \bar{a}b + bc$$

$$= \bar{a} + bc$$

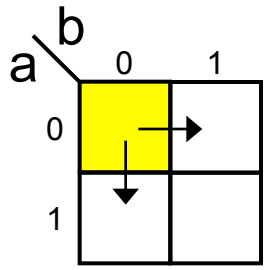
- Realizing adjacencies may be difficult in practice

Karnaugh map

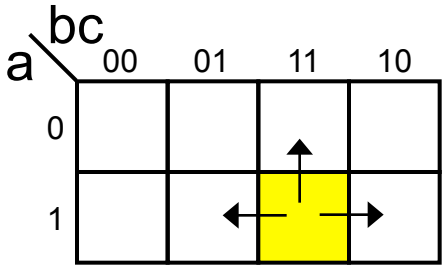
- A map that presents the truth table of a function in such a way that adjacent terms are contiguous :
  - There is a cell for each combination or term
  - Cells are numbered in Gray code
  - In a map for n variables, each cell has n adjacent cells that correspond to the combinations that result from inverting each of the n variables

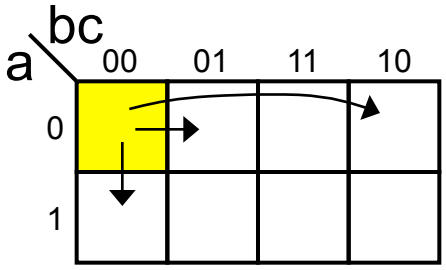
Karnaugh maps: adjacencies

- Two variables



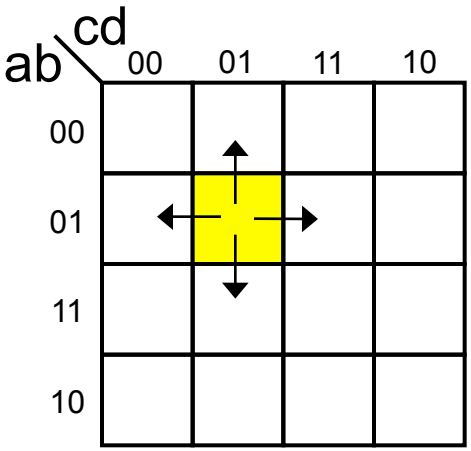
- Three variables

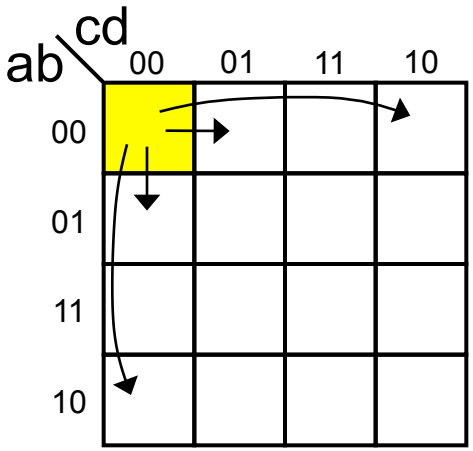




Karnaugh maps: adjacencies

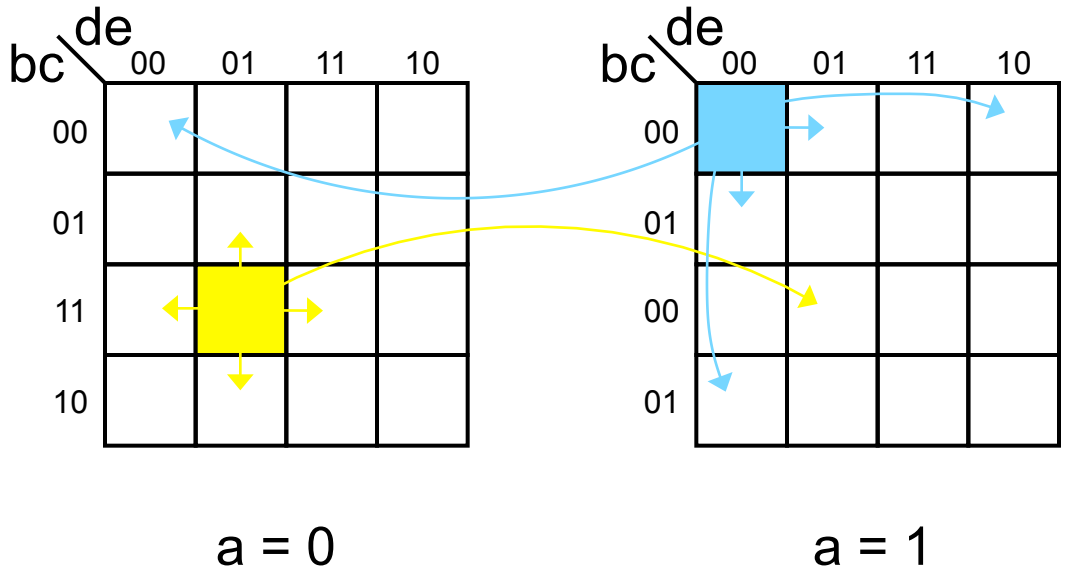
- Four variables





Karnaugh maps: adjacencies

- Five variables



Karnaugh map method: cell numbering

- Two variables

a \ b	0	1
0	0	1
1	2	3

- Three variables

a \ bc	00	01	11	10
0	0	1	3	2
1	4	5	7	6

- Four variables

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

Karnaugh map method: cell numbering

- Five variables

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

a = 0

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

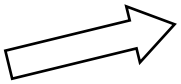
a = 1

Function representation in a Karnaugh map

- Cells corresponding to the minterms or the maxterms of the function are marked by a 1 or a 0, respectively
- Example:

$$f(a,b,c) = \sum_{3}(0,1,2,3,7) =$$

$$= \prod_{3}(4,5,6)$$

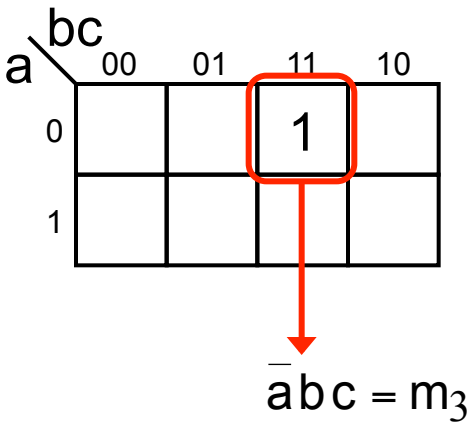


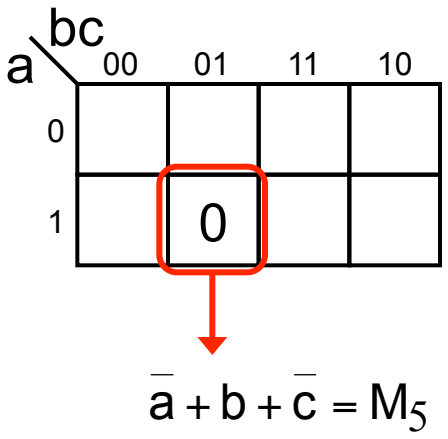
a \ bc	00	01	11	10
0	1	1	1	1
1			1	

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

Deriving an expression from a Karnaugh map

- Follow the rules for minterms and maxterms
  - Minterms rule
    - 0 → inverted variable
    - 1 → non-inverted variable
  - Maxterms rule
    - 0 → non-inverted variable
    - 1 → inverted variable

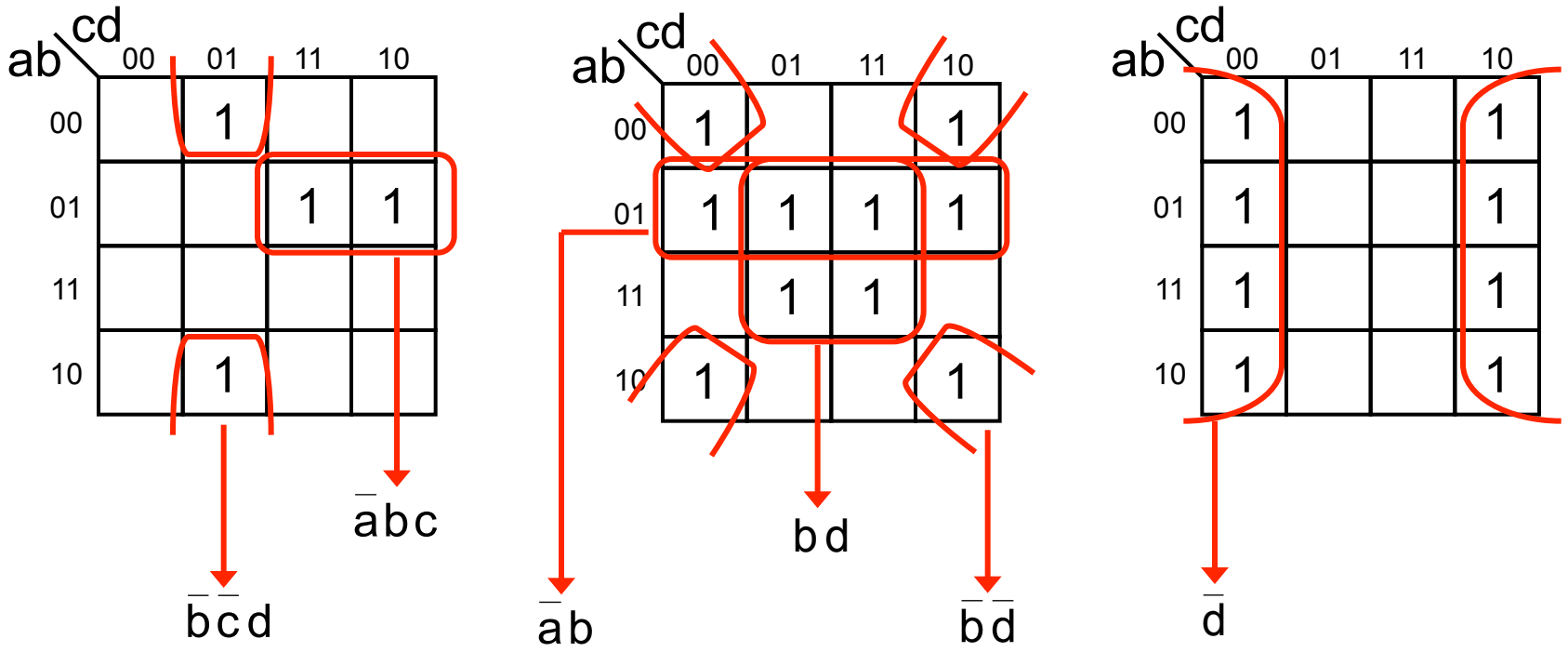




Simplification with Karnaugh maps

- Two options
  - By minterms (ones): result is a sum of products
  - By maxterms (zeros): result is product of sums
- Search for groups of adjacent cells
  - A group of 2 adjacent cells removes 1 variable
  - A group of 4 adjacent cells removes 2 variables
  - A group of 8 adjacent cells removes 3 variables
  - A group of 16 adjacent cells removes 4 variables
  - ….
- Goal: cover all minterms (maxterms) with the largest possible groups and the smaller number of groups
  - Terms can be covered more than once, if needed (absorption property)

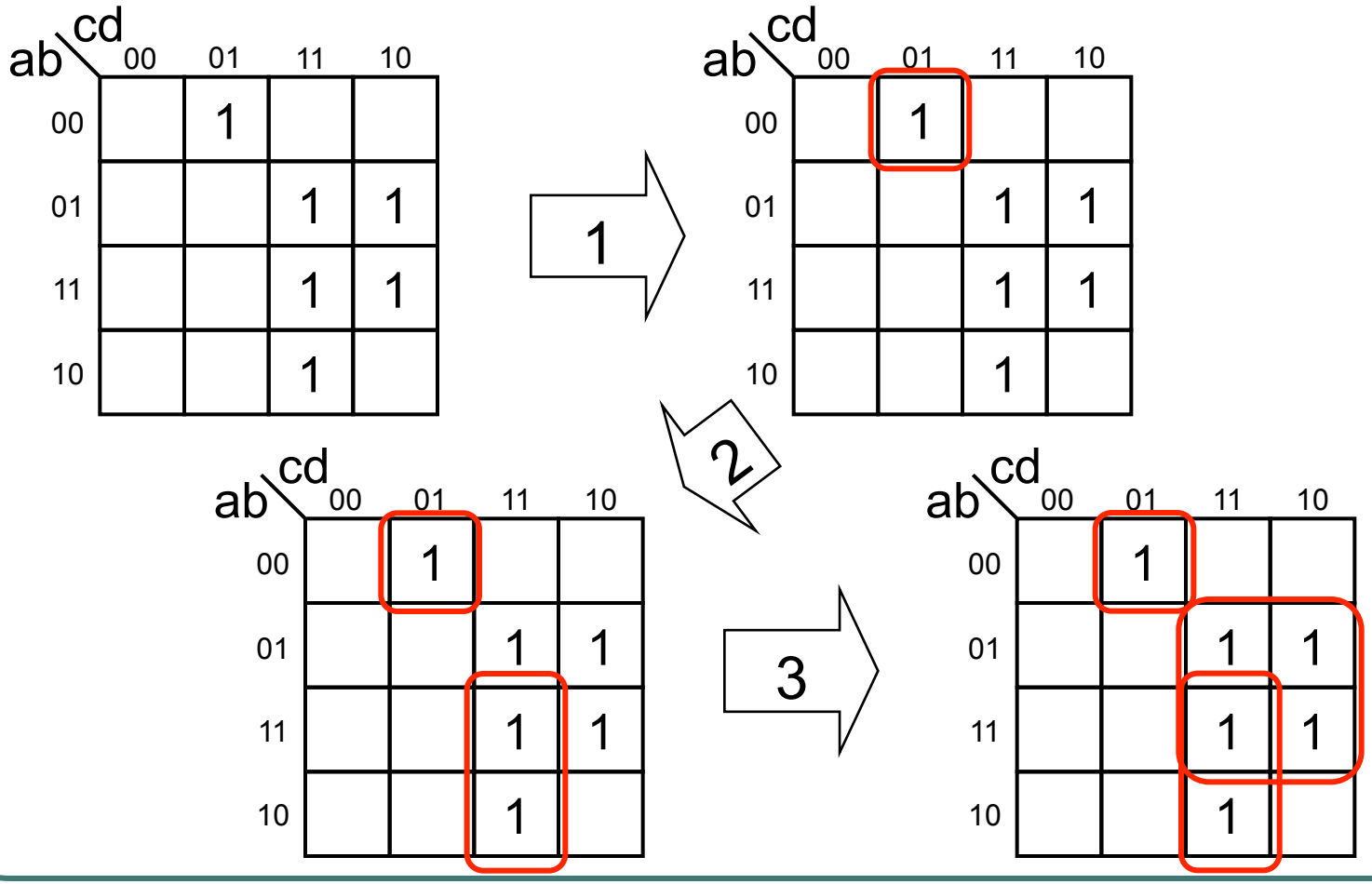
Simplification: groups



Simplification with Karnaugh maps: Algorithm

- Systematic algorithm
  1. Cover cells that cannot form groups of 2 cells
  2. Cover cells that can form groups of 2 cells, but not of 4 cells
  3. Cover cells that can form groups of 4cells, but not of 8 cells
  4. Cover cells that can form groups of 8 cells, but not of 16 cells
  5. …
- If there is more than one choice at any step:
  - First cover cells with less choices

Simplification with Karnaugh maps: Example



Incompletely specified functions

- An incompletely specified (or incomplete) function is a function that is not specified for some combination of input values
- Incomplete functions occur in practice:
  - When inputs come from another circuit that cannot produce some particular output combinations by construction
  - When there are combination for which the function value is meaningless or don't-care
- Notation:
  - A don't-care is represented as 'X' or '-'
  - The don't-care set is typicalled represented by Δ

Incompletely specified functions

- Example: Function to determine if a BCD number is even
  - Codes from 10 to 15 are meaningless in BCD

$$f(b3,b2,b1,b0) = \sum_4 (1,3,5,7,9) + \Delta(10,11,12,13,14,15) =$$

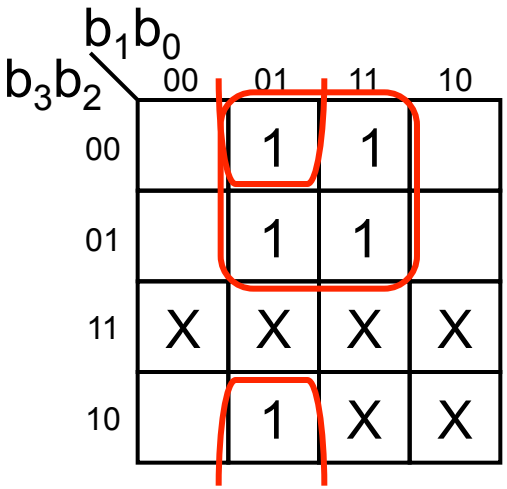
$$= \prod_4 (0,2,4,6,8) + \Delta(10,11,12,13,14,15)$$

b3	b2	b1	b0	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

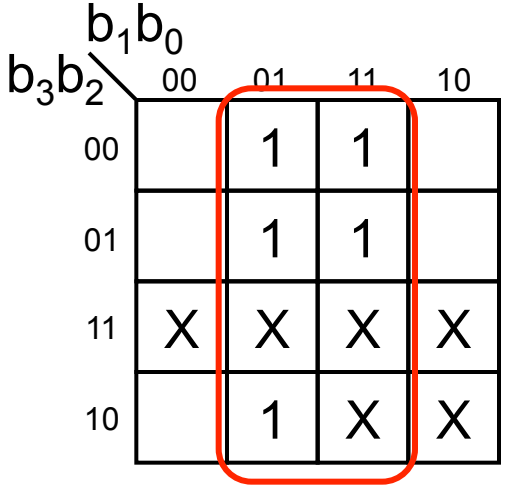
Don't-care terms

Minimization of incompletely specified functions

- Don't-care terms are "wild cards": they can be covered or not, as needed to form larger groups



$$f(b_3,b_2,b_1,b_0) = \overline{b_3}\, b_0 + \overline{b_2}\, \overline{b_1}\, b_0$$



$$f(b_3,b_2,b_1,b_0) = b_0$$

☑ *Correct*

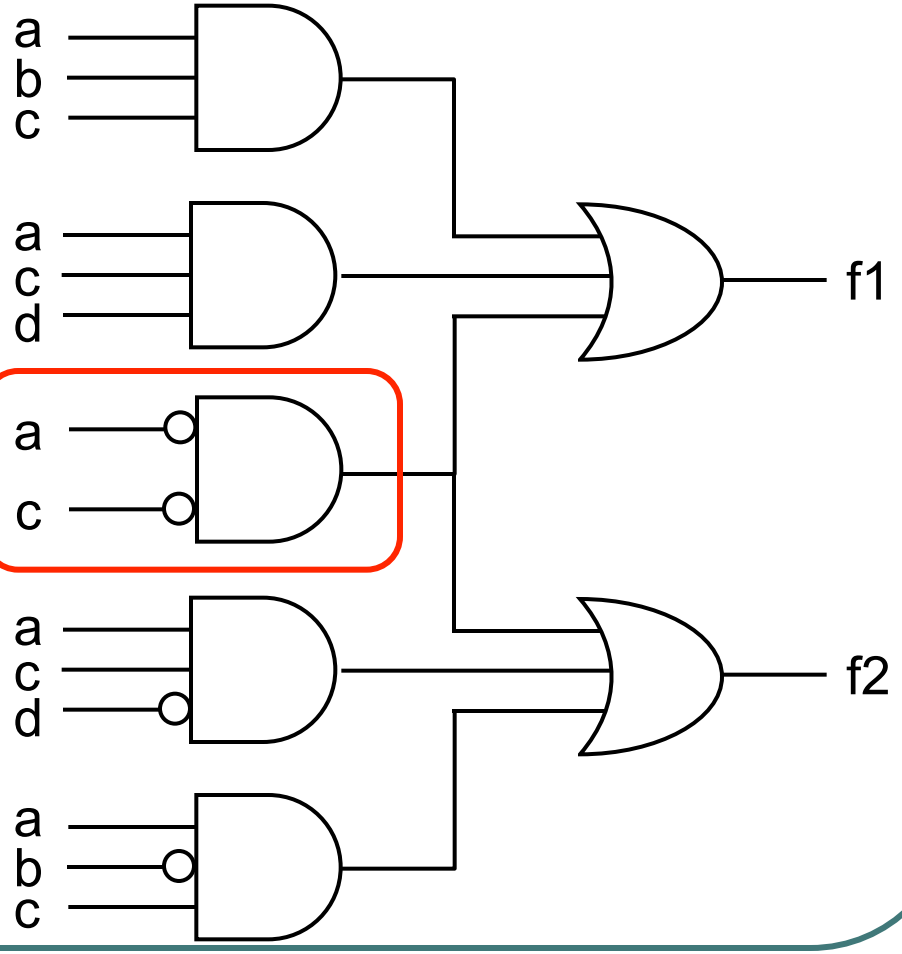
Multiple functions

- Digital circuits generally implement multiple functions: several functions or multiple-output functions
- Multiple function can be better implemented by considering them jointly
  - Common parts or terms can be shared to save logic
- Decomposition of multiple functions in order to maximize common subexpressions is difficult
  - Algorithms are difficult to apply in a manual way
  - We will generally do it by visual inspection

Multiple functions: Example

$f1(a,b,c,d) = \bar{a}\bar{c} + abc + acd$

$f2(a,b,c,d) = \bar{a}\bar{c} + a\bar{b}c + ac\bar{d}$



☑ *Common terms*

Multiple functions: Example

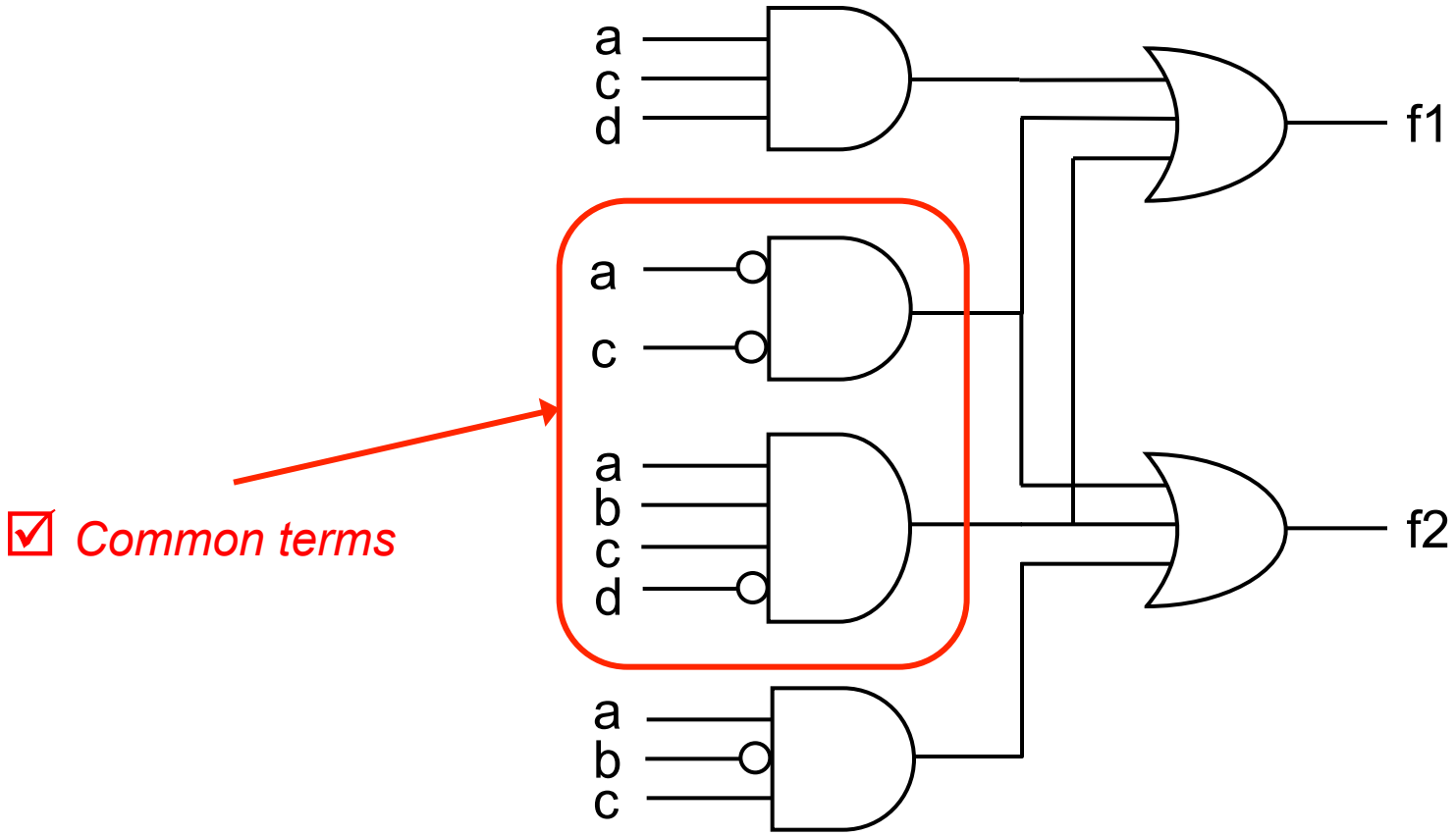
- More common terms may be found

$$f1(a,b,c,d) = \bar{a}\bar{c} + abc + acd = \bar{a}\bar{c} + abc\bar{d} + acd$$

$$f2(a,b,c,d) = \bar{a}\bar{c} + a\bar{b}c + ac\bar{d} = \bar{a}\bar{c} + abc\bar{d} + a\bar{b}c$$

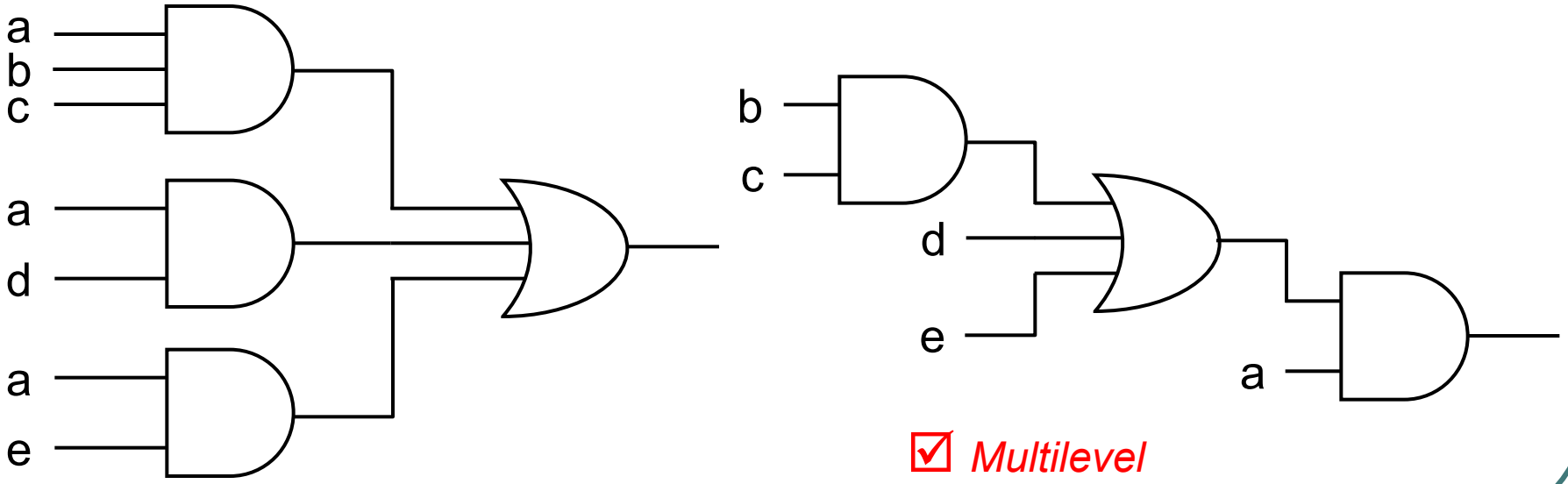
  - Function expressions are not optimal separately, but they are optimal as a group!
  - Design tools include algorithms to minimize multiple functions

Multiple functions: Example



Multilevel synthesis

- Better solutions are possible by removing the two-level constraint
  - Computer-aided heuristic algorithms are used
- Example: $f(a,b,c,d,e) = abc + ad + ae = a(bc + d + e)$



Optimization tools

- Manual methods:
  - Only two-level, few variables
- Software tools
  - Multilevel, multiple functions, many variables
  - Area and delay optimization
  - Generally incorporated into logic synthesis tools
- Logic synthesis tools
  - Work as a compiler, from a design description in schematic form or in a Hardware Description Language
  - Optimize the design and generate the logic gate description for the selected technology

References

- “Introduction to Digital Logic Design”. J. P. Hayes. Ed. Addison-Wesley
- “Circuitos y sistemas digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores