Unit 3. Combinational circuits

Digital Electronic Circuits
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan-Chico <jjchico@dte.us.es> 2010-2020

Contents

- **Logic functions**
- Boolean algebra
- Combinational design
- Functional analysis
- Timing analysis

Learning outcomes

- Use boolean expressions and their canonical and normalized forms.
- Represent any combinational circuit with truth tables, normalized boolean expressions and equivalent circuits
- Recognize combinational problems and be able to define a function to solve them.
- Obtain optimized boolean expressions using Karnaugh maps.
- Implement circuits for normalized boolean expressions using logic gates.
- Extract the logic function from a combinational circuit schematic.
- Represent the evolution in time of the signals in a combinational circuit using simple delay models.
- Understand the nature and risks of circuits presenting hazards.

Bibliography

- Recommended
  - LaMeres, chapter 4.
- Extra exercises from the course's collection 2 (in Spanish)
  - Function minimization from truth table: 1
  - Function minimization from min(max)terms: 4
  - Function minimization with don't cares: 5 (a, b, h)
  - Representations: 16
  - Design exercises: 7, 10, 19, 23, 24, 25, 26, 27
  - Non-standard design: 20
  - Functional analysis: 2
  - Timing analysis: 14

Logic functions. Example

- Many problems can be described with digital (0,1) variables:
  - on/off, active/inactive, true/false, run/stop, etc.

Verbal description

Example 1

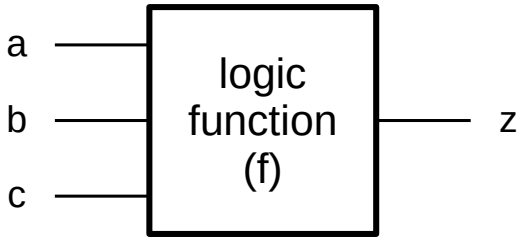
A digital alarm system may be on or off and has a presence sensor and a contact sensor at the main door. When the system is on, the alarm will be activated if presence or a door open is detected. When the system is off the alarm is activated only when presence is detected and the door is open (to prevent leaving the door open when at home).

a (on/off switch): 0-on, 1-off
b (presence sensor): 0-no presence, 1-presence.
c (door sensor): 0-door closed, 1-door open
z (alarm): 0-no activated, 1-activated

Logic functions. Example

- In some problems, the result can be expressed as a function of the variables.
- Function of digital variables are called “logic functions”.
- Truth tables are a formal way to specify the value of a logic function.
- Digital combinational circuits implement logic functions.

Formal description

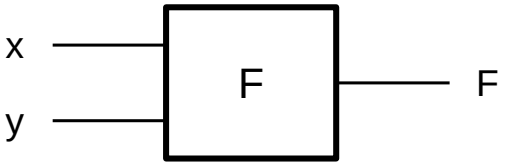


$$z = f(a,b,c)$$

a b c	z
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

a (on/off switch): 0-on, 1-off
b (presence sensor): 0-no presence, 1-presence.
c (door sensor): 0-door closed, 1-door open
z (alarm): 0-no activated, 1-activated

Two variables logic functions



x y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		AND					XOR	OR	NOR	XNOR					NAND	

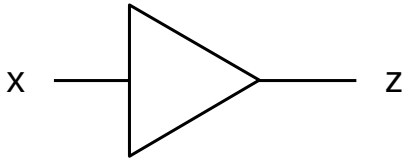
In general, there are $2^{(2^n)}$ logic functions of n variables

Basic logic operators and logic gates

IDENTITY

x	z
0	0
1	1

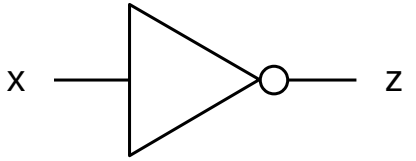
$z = x$



NOT

x	z
0	1
1	0

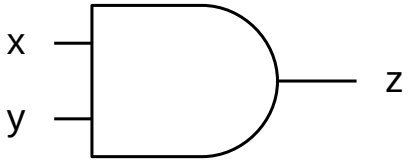
$z = \overline{x}$



AND

x y	z
0 0	0
0 1	0
1 0	0
1 1	1

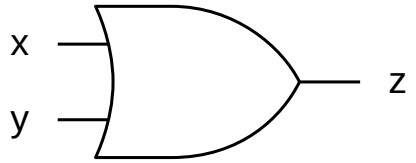
$z = x \cdot y$



OR

x y	z
0 0	0
0 1	1
1 0	1
1 1	1

$z = x + y$

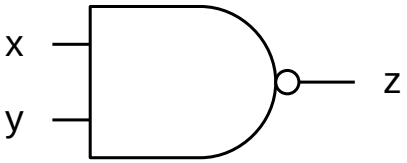


Basic logic operators and logic gates

NAND

x y	z
0 0	1
0 1	1
1 0	1
1 1	0

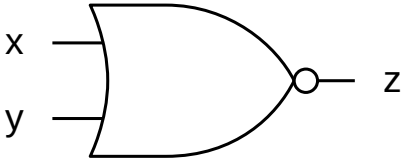
$$z = \overline{x \cdot y}$$



NOR

x y	z
0 0	1
0 1	0
1 0	0
1 1	0

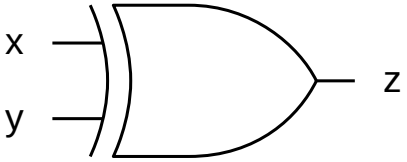
$$z = \overline{x + y}$$



XOR

x y	z
0 0	0
0 1	1
1 0	1
1 1	0

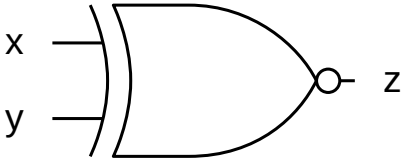
$$z = x \oplus y = \bar{x}y + x\bar{y}$$



XNOR

x y	z
0 0	1
0 1	0
1 0	0
1 1	1

$$z = x \odot y = \overline{x \oplus y} = \bar{x}\,\bar{y} + xy$$



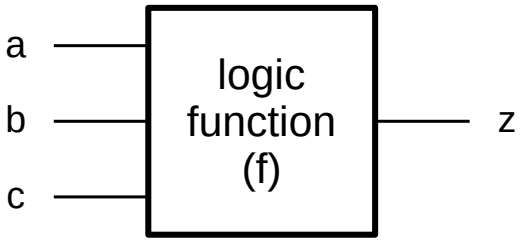
Logic expressions

- Expression involving logic operators, mostly:
  - AND (·), OR (+) y NOT (¯ ')
- Precedence of · over +
  - x + (y·z) = x + y·z
- "·" can be ommited
  - x + y·z = x + yz
- A way of specifying logic functions
  - Ex: calculate f(0,1,1), f(1,0,0), f(1,1,1)

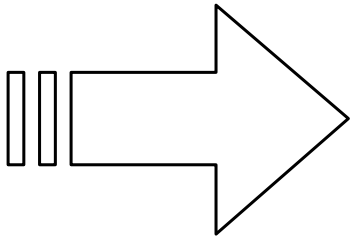
$$f(a,b,c) = (a+b+c)\,(a+\overline{b}\,c) + cd\,\overline{(a+c)}$$

Logic functions summary

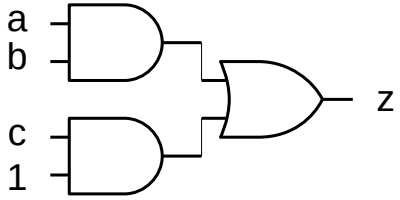
- Because we have basic circuits that do basic logic operations (INV, AND, OR, etc.), we can convert any logic function into a digital circuit (through its logic expression).
- The objective of combinational design is to obtain a logic function that solves a problem and build the corresponding logic circuit.



a b c	z
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1



Combinational circuit

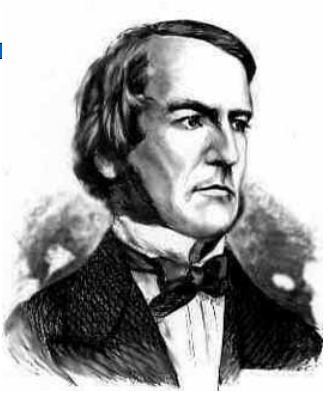


Contents

- Logic functions
- **Boolean algebra**
- Combinational design
- Functional analysis
- Timing analysis

Formalization: Boolean algebra

- B={0, 1}
- {B, NOT, AND, OR} form a Boolean Algebra
  - A particular case: switching algebra.
- Any boolean algebra fulfills this axioms (by definition)



George Boole (1815-1864)

Identity	$x+0 = x$	$x \cdot 1 = x$
Commutativity	$x+y = y+x$	$x \cdot y = y \cdot x$
Distributivity	$x \cdot (y+z) = (x \cdot y)+(x \cdot z)$	$x+(y \cdot z) = (x+y) \cdot (x+z)$
Complement	$x+\overline{x} = 1$	$x \cdot \overline{x} = 0$

Duality: if an expression holds, the expression that results from interchanging + with · and 0 with 1 also holds.

Boolean algebra theorems

Idempotence	$x+x = x$	$x \cdot x = x$
Complement uniqueness	x is unique	
Annihilation	$x+1 = 1$	$x \cdot 0 = 0$
Double complement	$\overline{(\overline{x})} = x$	
Absorption	$x+xy = x$	$x \cdot (x+y) = x$
Consensus	$x+xy = x+y$	$x \cdot (x+y) = x \cdot y$
Associativity	$x+(y+z) = (x+y)+z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
De Morgan	$x \cdot y = x+y$	$x+y = x \cdot y$
Reduction	$xy+xy = x$	$(x+y)(x+y) = x$

Normalized forms

- Sum of products (SOP)
  - Sum of product terms
  - Product term: product of “literals” (single variable, may be complemented) without repetitions.
- Product of sums (POS)
  - Product of sum terms
  - Sum term: sum of “literals” without repetitions.
- How to obtain
  - By applying distributivity, De Morgan and basic simplification theorems.
  - SOP: distributivity of the product (AND) with respect to the sum (OR).
  - POS: distributivity of the sum (OR) with respect to the product (AND).

Normalized forms. Example

Conversion to SOP

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

$$aa+a\bar{b}c+ba+b\bar{b}c+ca+c\bar{b}c+c\,d\,\bar{a}\,\bar{c}=$$

$$a+a\bar{b}c+ab+ac+\bar{b}c$$

Conversion to POS

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

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

$$(a+b+c)(a+\bar{b})(a+c)$$

Canonical forms

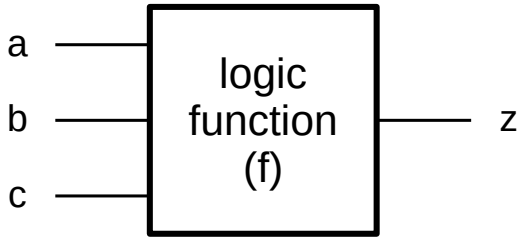
- Given a set of n variables.
- Minterm
  - Product term that contains all the variables.
- Maxterm
  - Sum term that contains all the variables.
- Canonical form of minterms (sum of minterms)
  - SOP where all the product terms are minterms
- Canonical form of maxterms (product of maxterms)
  - POS where all the sum terms are maxterms.

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

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

$$(a+b+c)(a+b+\bar{c})(a+\bar{b}+c)(\bar{a}+b+c)$$

From truth table to sum of minterms



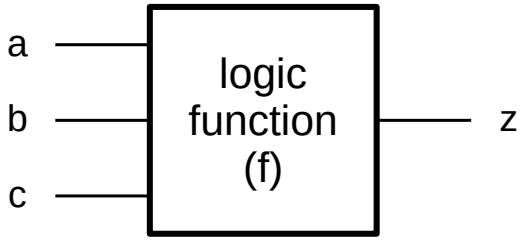
An expression for a function can always be obtained by combining the NOT, AND and OR operators.
Method:
1. For every "1" of the function, build a product term that is "1" for that input combination only.
2. Sum (OR) all the terms.

The resulting expression is a sum of minterms.

a b c	z
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

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

From truth table to product of maxterms



An expression for a function can always be obtained by combining the NOT, AND and OR operators.
Method:
1. For every "0" of the function, build a sum term that is "0" for that input combination only.
2. Multiply (AND) all the terms.

The resulting expression is a product of maxterms.

a b c	z
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

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

Canonical form notation

a b c	minterm	m-notation
0 0 0	$\bar{a}\,\bar{b}\,\bar{c}$	m0
0 0 1	$\bar{a}\,\bar{b}\,c$	m1
0 1 0	$\bar{a}\,b\,\bar{c}$	m2
0 1 1	$\bar{a}\,b\,c$	m3
1 0 0	$a\,\bar{b}\,\bar{c}$	m4
1 0 1	$a\,\bar{b}\,c$	m5
1 1 0	$a\,b\,\bar{c}$	m6
1 1 1	$a\,b\,c$	m7

$z = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$
$z = m3 + m5 + m6 + m7$
$z = \Sigma(3,5,6,7)$

minterm:
input combination for which the function is "1"

a b c	maxterm	M-notation
0 0 0	a+b+c	M0
0 0 1	$a+b+\bar{c}$	M1
0 1 0	$a+\bar{b}+c$	M2
0 1 1	$a+\bar{b}+\bar{c}$	M3
1 0 0	$\bar{a}+b+c$	M4
1 0 1	$\bar{a}+b+\bar{c}$	M5
1 1 0	$\bar{a}+\bar{b}+c$	M6
1 1 1	$\bar{a}+\bar{b}+\bar{c}$	M7

$z = (a+b+c)(a+b+\bar{c})(a+\bar{b}+c)(\bar{a}+b+c)$
$z = M0\ M1\ M2\ M4$
$z = \Pi(0,1,2,4)$

maxterm:
input combination for which the function is "0"

Conversion between function representations

- Canonical form to/from truth table
  - Minterms: each '1' is a minterm
  - Maxterms: each '0' is a mexterms
- SOP/POS to canonical form
  - Expand each term to include all the variables.
  - SOP: multiply by '1' like (a+a)
  - POS: sum '0' like (aa)
  - Simplify identical terms.
- SOP/POS to truth table
  - Method 1: obtain a canonical form first.
  - Method 2: identify each term with the 1's or 0's in the truth table
    - Product term: 1's.
    - Sum term: 0's.

SOP/POS to canonical form Examples

$$a + a\,\bar{b}\,c + ac + \bar{b}c =$$

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

$$a\,b\,c+a\,b\,\bar{c}+a\,\bar{b}\,c+a\,\bar{b}\,\bar{c}+a\,\bar{b}\,c+a\,\bar{b}\,c+\bar{a}\,\bar{b}\,c =$$

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

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

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

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

$$(a+\bar{b}+c)(a+b+c)(a+b+\bar{c})(a+\bar{b}+\bar{c})$$

SOP/POS to truth table Examples

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

- z = 1 if and only if:
  - a=1, or
  - a=1 and b=0 and c=1, of
  - a=1 and c=1, or
  - b=0 and c=1

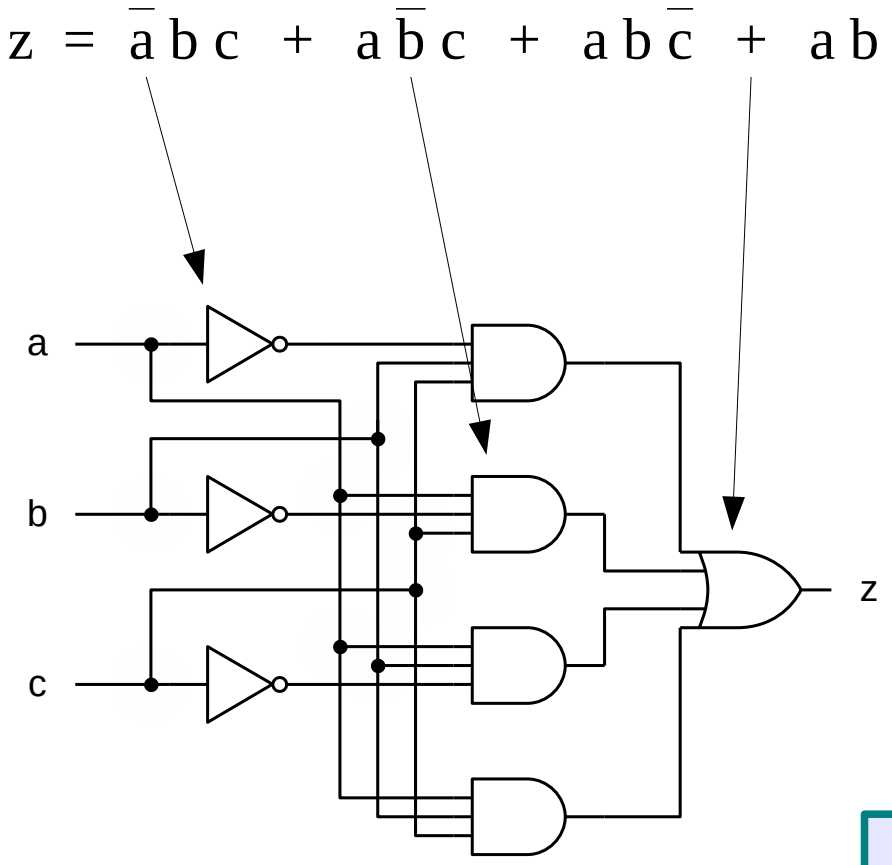
$z(a,b,c) = (a+b+c)(a+\bar{b})(a+c)$

- z = 0 if and only if:
  - a=0 and b=0 and c=0, or
  - a=0 and b=1, or
  - a=0 and c=0

Contents

- Logic functions
- Boolean algebra
- **Combinational design**
- Functional analysis
- Timing analysis

Approximation to logic design



Any SOP (or POS) can be directly translated into a digital circuit by combining basic logic gates:

- Inverters (complement)
- AND (products)
- OR (sums)

Can it be done with a more simple circuit?

Optimum logic circuit design (using logic gates)

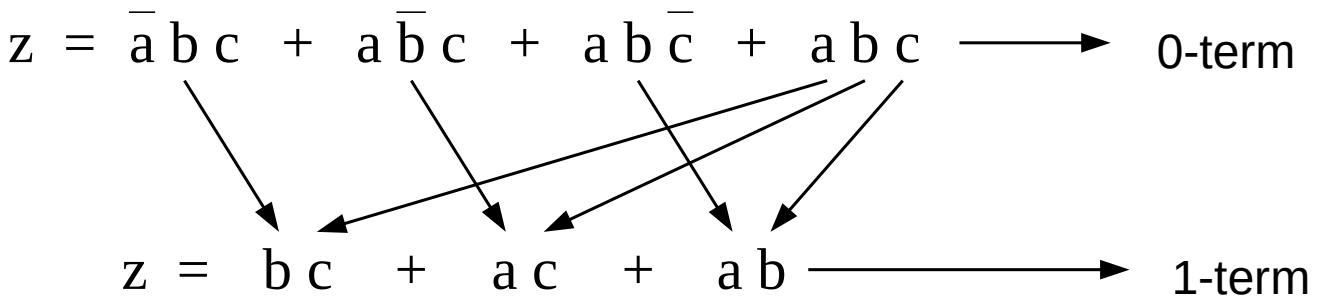
- Any practical circuit must meet some “design constraints”:
  - Maximum size (use small and fewer devices)
  - Maximum power consumption
  - Maximum delay (it has to be fast enough)
- Normalized forms (SOP and POS) are a good starting point
  - SOP and POS can be directly translated to a circuit made of simple gates.
  - Any function can be done in 3 level of gates maximum (delay control).
  - Systematic methods exist to simplify SOP's and POS's.
- General criteria: a simpler expression will yield a simpler circuit.
  - Reduce the number of terms: no. terms = no. gates
  - Reduce the number of literals in each term: no. literals = no. gate inputs

Derive your circuit from a “minimum” normalized expression (SOP or POS)

Logic expression minimization

Reduction theorem: $xy + x\bar{y} = x$ — "x" can be any expression

Idempotence: $x + x = x$ — Each term may be used more than once



Implicant of a function

- Product term that can be part of a S-O-P expression of a function
- Product term that "covers" some minterms of a function

Logic expression minimization
Quine-McCluskey algorithm (partial)

- Start with the function's list of minterms.
- Find all first order implicants (1-term).
  - By applying the reduction theorem to adjacent minterms
- Eliminate covered minterms
  - No need to worry about covered minterms because they are considered in higher order implicants (more simple terms)
- Find all second order implicants (2-term)
  - By applying the reduction theorem to adjacent 1-term's
- Eliminate first order covered implicantes
- Repeat until implicants are as big as possible (prime implicants)
  - Prime implicants are the simplest terms possible to cover all the minterms of the function.
- Select all the essential implicants (with minterms not covered by other prime implicants) and a minimum number of aditional implicants to cover all minterms.

Logic expression minimization
Q-M algorithm example

$$F(a,b,c,d) = \Sigma(0,1,4,9,11,13,15)$$

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

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

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

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

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

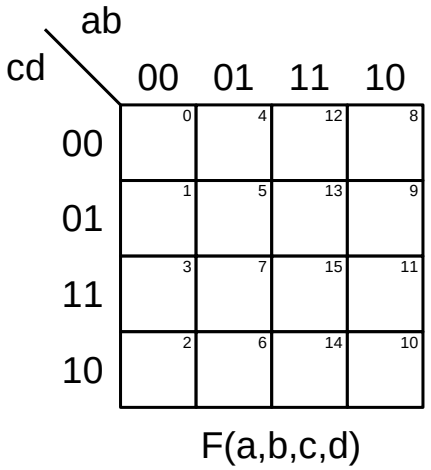
$a\,\bar{b}\,c\,d$

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

$a\,b\,c\,d$

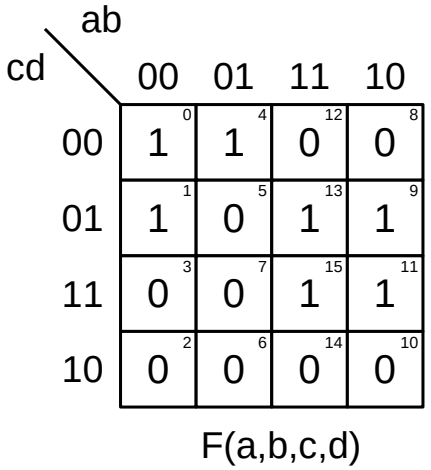
Logic expression minimization
Karnaugh maps (K-maps)

- K-mapas:
  - Bi-dimensional truth tables.
  - Gray code-like variable value ordering.
  - Each cell corresponds to a minterm (maxterm).
  - Easy to localize minterms and n-terms.
  - K-maps are cyclic!
  - Easy finding of prime implicants.
  - Easier expression minimization than written Q-M method.
  - The K-map method is the Q-M method applied graphically.



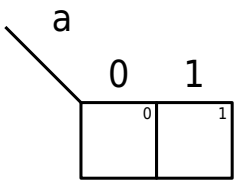
F(a,b,c,d)

$$F(a,b,c,d) = \Sigma(0,1,4,9,11,13,15)$$

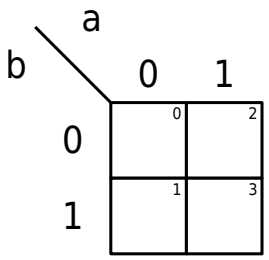


F(a,b,c,d)

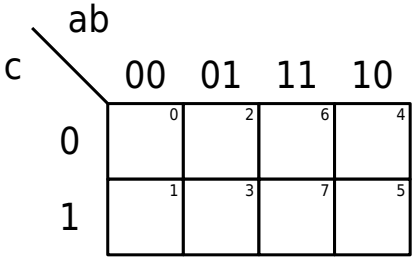
K-maps (1 to 6 variables)



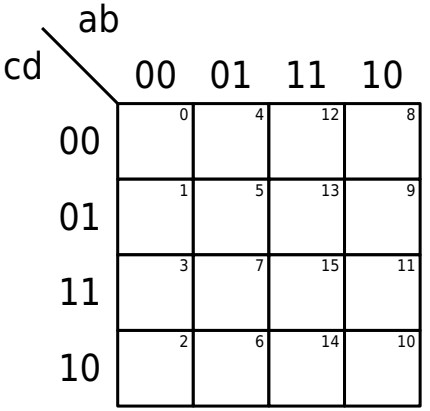
F(a)



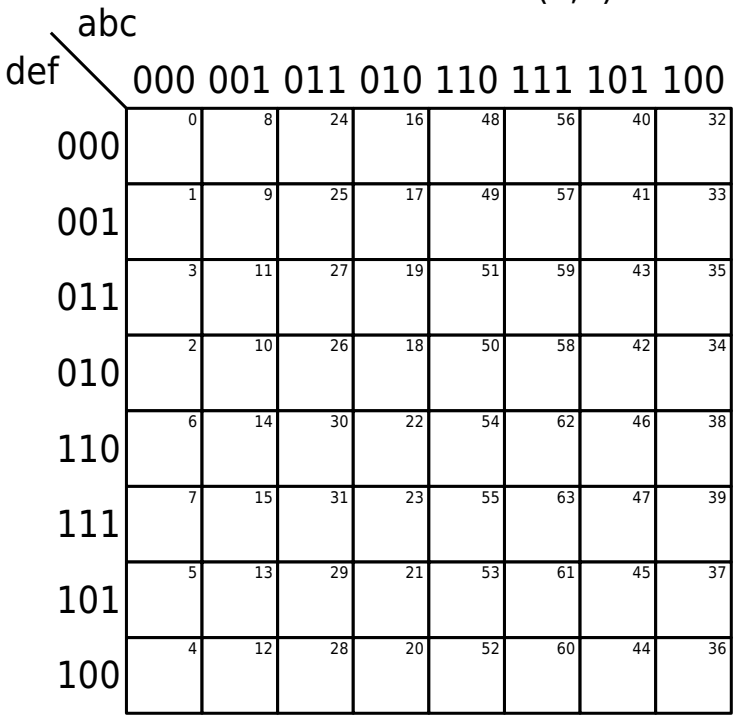
F(a,b)



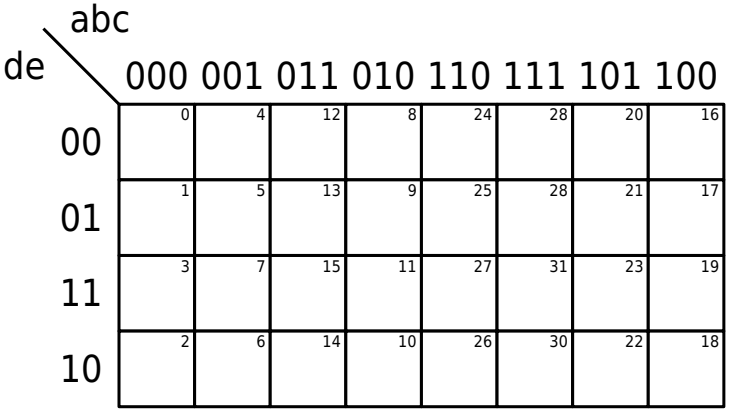
F(a,b,c)



F(a,b,c,d)



F(a,b,c,d,e,f)



F(a,b,c,d,e)

The minterm index depend on how variables are ordered!

K-map minimization. Example

- From a list of minterms/maxterms
  - f(a,b,c) = ∑(1,2,3,4,5)
  - f(a,b,c,d) = ∏(2,3,4,5,6,7,8,10,12,14)
- From a truth table
  - Copy the data in the K-map in the right order!
- From an (possible non-minimal) expression
  - Convert the expression to a SOP or POS
  - Translate the expression to the K-map
  - Find minimum SOP/POS

a b c	z
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

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

Logic expression minimization (summary)

- SOP/POS minimization: systematic.
- Two-level minimum expressions guarantee.
- Small and homogeneous delay.
- Only basic operators/gates: AND, OR, NOT
- Methods
  - Simple circuits (up to 6 inputs): K-map
  - Moderately simple circuits: Quine-McCluskey method (we won't use it in exercises)
    - Can be programmed in a computer
    - Exponential complexity: 32 inputs functions has more than 10^{15} prime implicants!
  - Complex circuits: heuristic logic minimizers (no optimum solution)
  - Even more complex circuits: combinational subsystems (Unit 4)

From minimum SOP/POS to circuit

- Each normalized form corresponds to a type of circuit
  - SOP → two levels AND-OR circuit
    - 1st level: products (AND gates).
    - 2nd level: sum (one OR gate).
  - POS → two levels OR-AND circuit
    - 1st level: sums (OR gates)
    - 2nd level: products (AND gates)
- Single rail vs double rail inputs
  - Single rail inputs: input signals are only available uncomplemented. Use inverters to obtain the complements.
  - Double rail inputs: input signals are available both uncomplemented and complemented. No need to use inverters.

Cost

- Cost
  - Number of resources used to build the circuit (transistors, resistors, etc.)
  - Number of terms → number of gates
  - Number of literals → number of inputs to the gates
  - Number of (unique) complements → number of inverters
- In conventional CMOS technology, every input to a gate “costs” two transistors. In this case, the cost can be estimated like:

$$cost = no.terms + no.literals + no.complements$$

- Note:
  - Single literal terms are not counted (they are already counted as literals).
  - No. of complements is not included if inputs are in double rail.
  - A minimum circuit is derived from the expression with less cost (SOP or POS)

Optimum logic circuit design (examples)

Example 2

Design an optimum two-levels combinational circuit for example 1 (introduction).

Example 3

A modern processor run four processing units at a time: P1, P2, P3 and P4. Each unit sets an output bit 'pi' to one when it is busy. The system is considered busy when any of the following conditions is met:

- P1 and any other unit are busy.
- P2 and P3 are busy.
- P4 is busy and neither P1 nor P2 are busy.

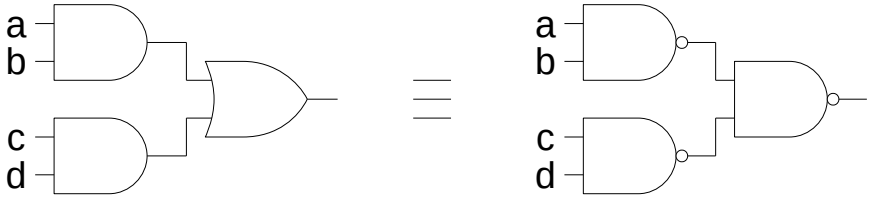
Design a minimum two-level circuit (plus inverters). Inputs are single rail.

Design alternatives

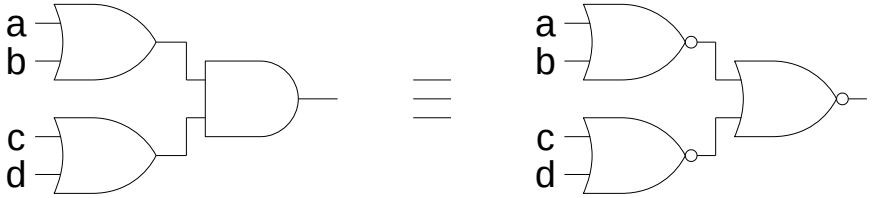
Modern IC design uses CMOS technology. Are NAND and NOR gates equally efficient?

- In general, it is easier to build inverting gates (NAND, NOR) than non-inverting gates (AND, OR).
- AND-OR (SOP) circuits can be converted to NAND-NAND circuits just by replacing the gates.
- OR-AND (POS) circuits can be converted to NOR-NOR circuits just by replacing the gates.

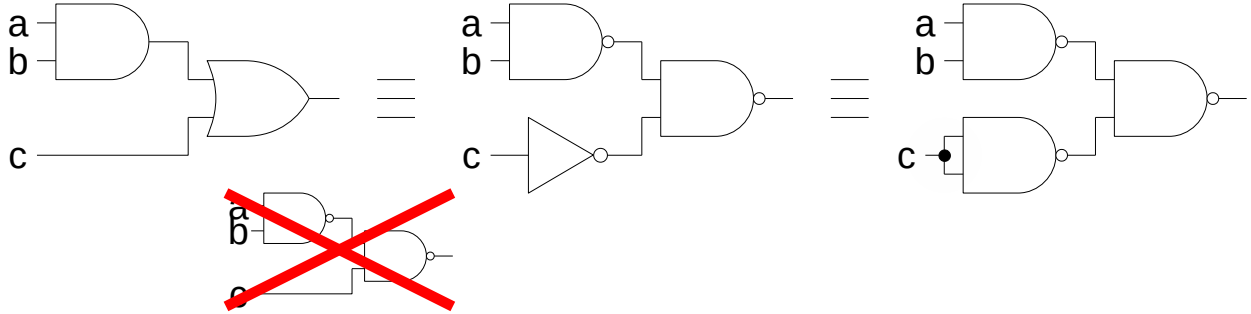
$$ab+cd=\overline{(\overline{ab})(\overline{cd})}$$



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



$$ab+c=\overline{(\overline{ab})(\overline{c})}$$



Optimum simple logic circuit design (summary)

- Understand the verbal description of the problem.
- Clearly define digital inputs and outputs.
- Make a formal description of the problem: truth table, K-map, expression, etc.
- Convert the description to a K-map.
- Simplify the expression as SOP and/or POS depending on conditions.
- Convert the expression into a circuit.
  - See if you have simple or double rail (affects the cost).
  - Consider the cost of SOP and POS.
  - Use the appropriate gates depending on conditions: AND, OR, NAND, NOR, etc.
- Be clean when drawing the circuit!

Design example

Example 4

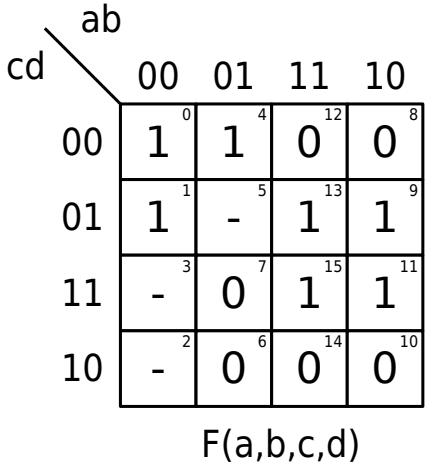
A modern processor run four processing units at a time: P1, P2, P3 and P4. Each unit sets an output bit 'pi' to one when it is busy. The system is considered busy when any of the following conditions is met:

- P1 and any other unit are busy.
- P2 and P3 are busy.
- P4 is busy and neither P1 nor P2 are busy.

Design a minimum two-levels circuit using only NAND gates.

Don't cares

$$F(a,b,c,d) = \Sigma(0,1,4,9,11,13,15)+d(2,3,5)$$



- Don't cares are input values for which the output of the function is not defined.
- Don't cares make function implementation more simple and should be identified when designing a logic function.
- Method (for SOP):
  - Considered '1' when useful to make bigger (more simple) prime implicants.
  - But don't cares do not need to be “covered” when selecting prime implicants.

Design example

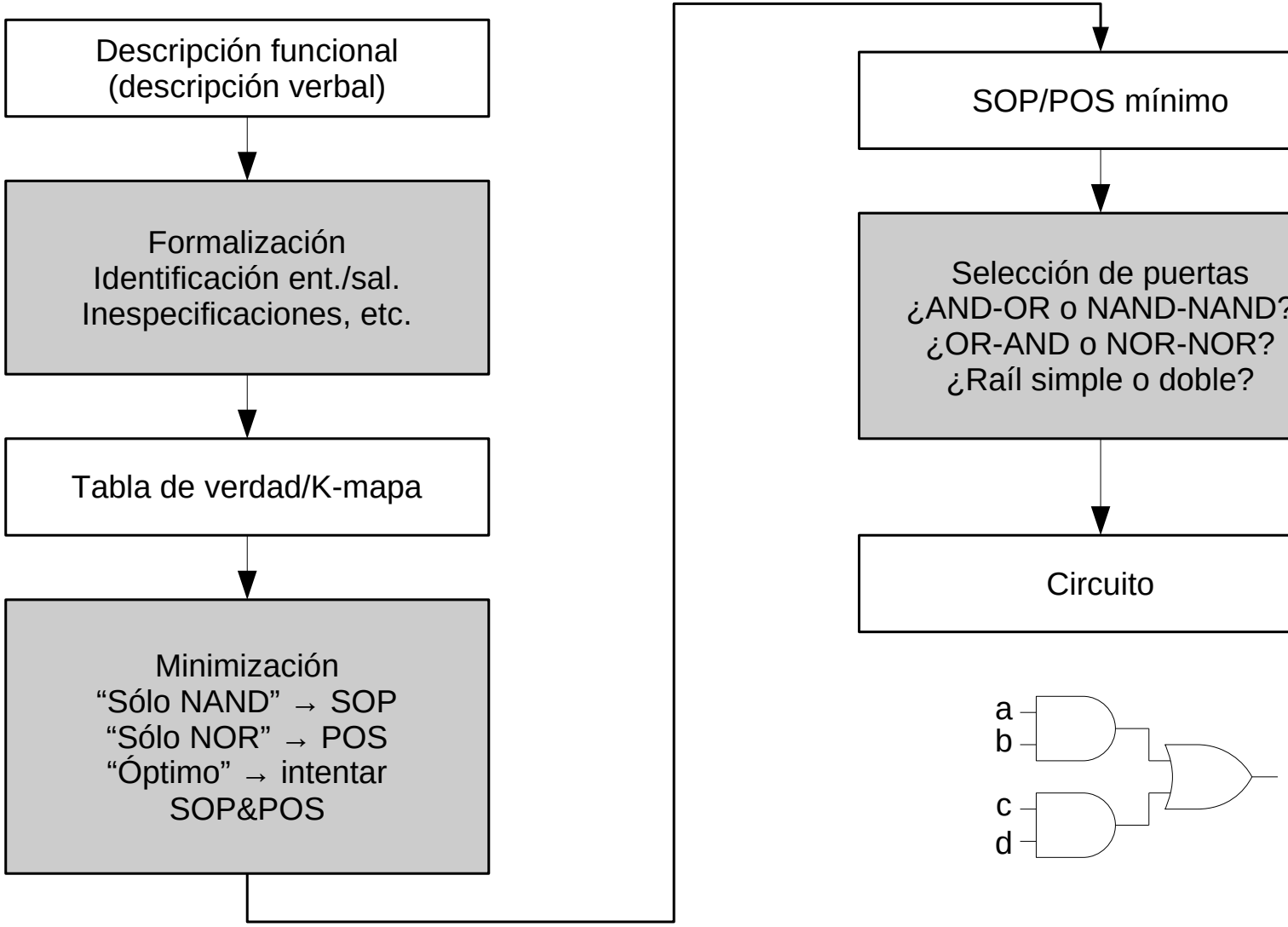
Example 5

Design a combinational circuit with four inputs (x3, x2, x1, x0) that represent the bits of a BCD digit X, and two outputs (q1, q0) that represents the bits of a magnitude Q, where q is the quotient of the division X/3.

E.g. if X=7 → Q=2, that is, (x3,x2,x1,x0)=(0,1,1,1) → (c1,c0)=(1,0)

Design the circuit using a minimum two-level structure of only NAND gates.

K-map logic circuit design (summary)



Contents

- Logic functions
- Boolean algebra
- Combinational design
- **Functional analysis**
- Timing analysis

Functional analysis

- What it is?
  - Obtain the logic function and possibly a description of the operation of a combinational circuit.
- Why is it useful?
  - To repair the circuit (if not working properly).
  - To design a new circuit with the same functionality, maybe with improvements or a different technology.
- Method:
  - Identify inputs and outputs.
  - Starting at primary inputs and for each gate with known inputs, calculate the logic expression of the output.
  - Repeat until all the outputs of the circuit are known.
  - Convert to something easier to analyze: truth table, K-map, etc.
  - Explain what the circuit does with words (if possible).

Functional analysis. Example

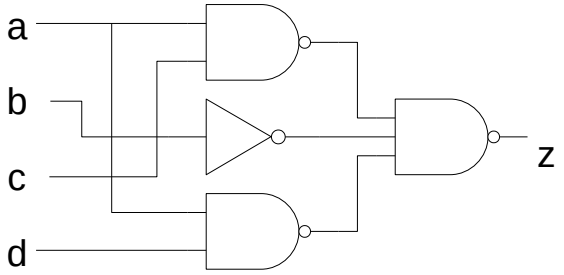
Example 6

The circuit below corresponds to a damaged alarm system with four inputs and one output. Looking at the connections we know that the inputs correspond to:

- a: system activation (0 - off, 1 - on)
- b: fire sensor (0 - no fire, 1 - fire)
- c: front door sensor (0 - close, 1 - open)
- d: presence sensor (0 - no presence, 1 - presence)

When output z is active (z=1) the alarm rings.

- Analyze the circuit and obtain its truth table.
- Describe with words the operation of the alarm: cases that make the alarm to ring, etc.
- Redesign the circuit using only NOR gates.



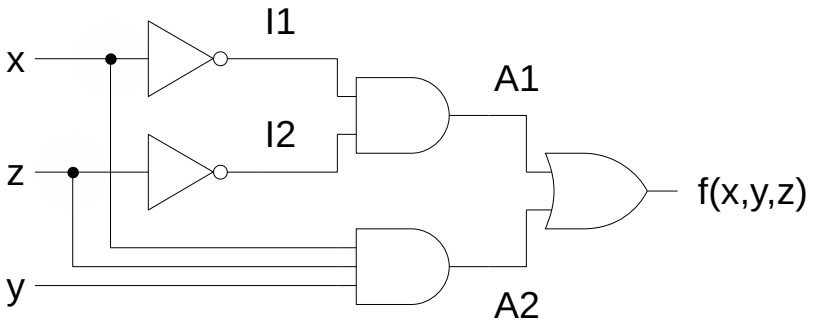
Contents

- Logic functions
- Boolean algebra
- Combinational design
- Functional analysis
- **Timing analysis**

Timing analysis

- What is it?
  - The study of the evolution in time of the internal and output signals of a circuit for a set of known input signals.
  - Waveform: representation of the variation of a signal with time.
- Why is it useful?
  - Calculate the propagation delay of a circuit: the time spent by the circuit to obtain a correct output value after an input change.
  - Analyze possible failures or unexpected behavior of the circuit due to timing factors: excessive delay, transient values (hazards), etc.
- Method
  - For every gate: equations of the output as a function of the inputs.
  - Substitute constant input values (DO NOT SUBSTITUTE ANYTHING ELSE)
  - Draw node waveforms from primary inputs to the outputs. Delay every output transition by the gate's delay time (simple model: same delay for every gate: Δ).

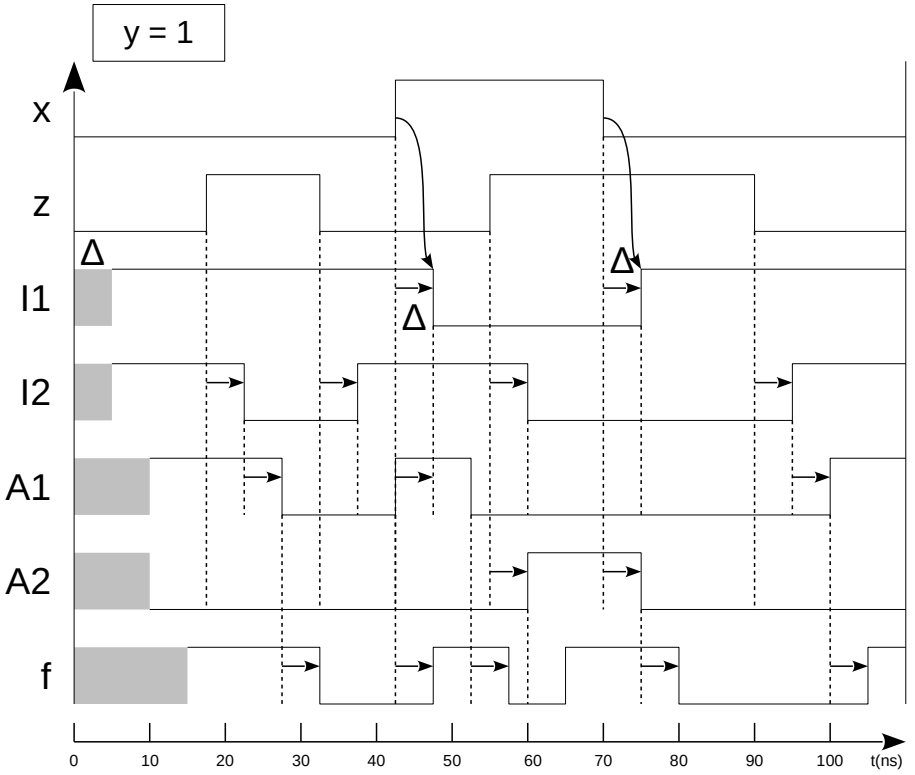
Timing analysis. Example



$I1 = \overline{x}$
$I2 = \overline{z}$
$A1 = I1\ I2$
$A2 = x\ y\ z$
$f = A1 + A2$

y = 1

$I1 = \overline{x}$
$I2 = \overline{z}$
$A1 = I1\ I2$
$A2 = x\ z$
$f = A1 + A2$

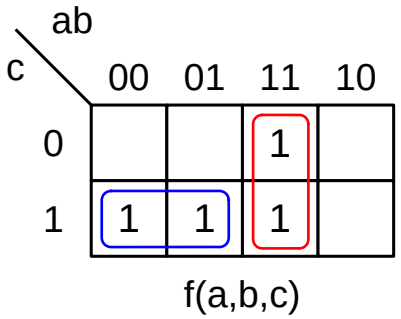


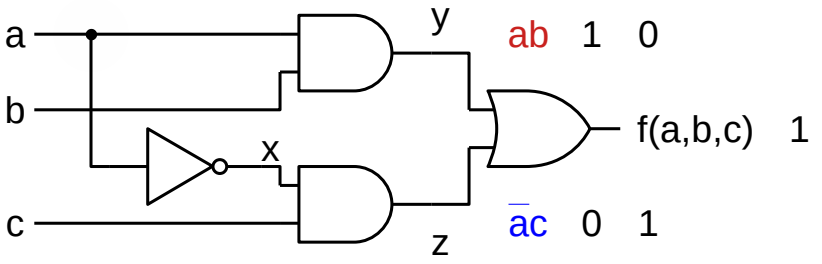
Timing analysis. Hazards

- What is it?
  - A wrong **transitory** value of the circuit output after inputs change to a new value.
- What is the cause?
  - The propagation delay in logic devices (gates) when an input affect the output through more than one path with different propagation delays.
  - Their existence depends on the internal structure of the circuit.
- Is the circuit wrong, then?
  - Hazards are perfectly normal and frequent.
  - Not harmful in general because they are transitory.
  - May cause problems in particular applications.
  - Can be avoided by changing the circuit's design.

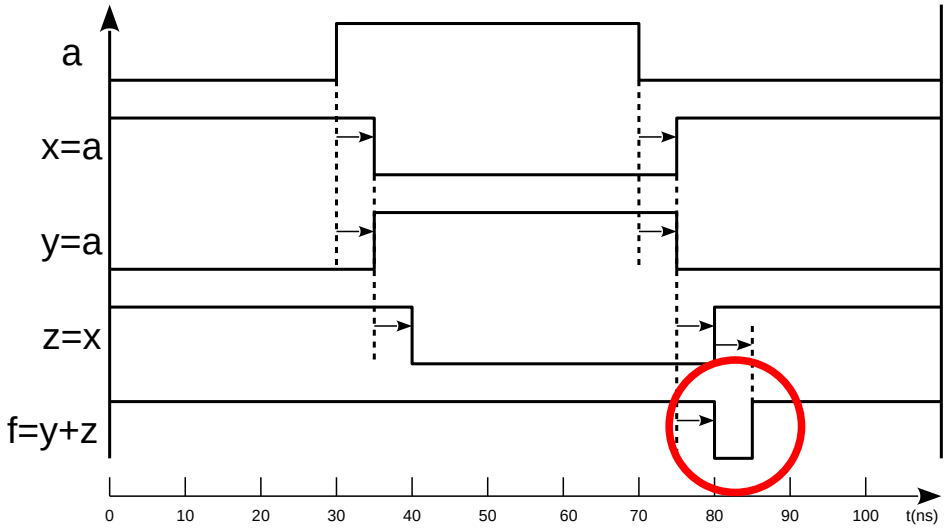
Timing analysis. Hazards

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





b=c=1 → $f(a,b,c) = a + \bar{a} = 1$



Is it a problem? Imagine b and c are alternative control signals and a is a selector.

How to avoid?