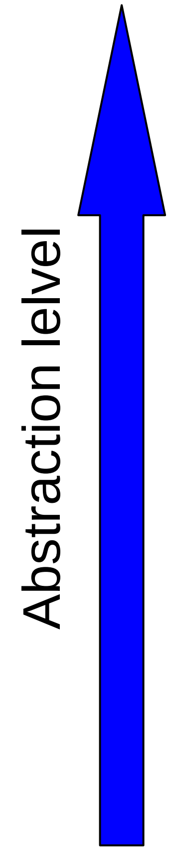
Enrico Nardelli
Logic Circuits and Computer Architecture

Appendix A
Digital Logic Circuits

Part 1: Combinational Circuits and Minimization

Structured organization

- Problem-oriented language level
- Assembly language level
- Operating system machine level
- Instruction set architecture level
- Microarchitecture level
- Digital logic level



Digital Logic level

- Digital circuits
  - Only two logical levels present (i.e., binary)
  - low/high voltage
- Basic gates
  - AND, OR, NOT
- Basic circuits
  - Combinational (without memory, stateless)
  - Sequential (with memory, state dependent behaviour)

Boolean Algebra

- Variables: A, B, …
- Domain of variables: 2 values
  - 1 or 0; Y or N; true or false; …
- Fundamental Operations
  - AND, OR, NOT
- Intended meaning (for humans - *Laws of Thought*)
  - AND: both inputs are true
  - OR: at least one input is true
  - NOT: negate the input
- Named from George Boole

George Boole (1815-1864)

An Investigation of the Laws of Thought, on Which are founded the Mathematical Theories of Logic and Probabilities (1854)

Formal definition of functions (1)

- By means of "truth tables"
  - Explicit representation of the output for all possible inputs

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT
0	1
1	0

Boolean functions

- Conventions
  - NOT (negation): NOT(A) = A′ = $\overline{A}$
  - AND (conjunction): AND(A,B) = AB = A.B
  - OR (disjunction): OR(A,B) = A+B

  - NAND(A,B) = NOT(AND(A,B)) = (AB)′
  - NOR(A,B) = NOT(OR(A,B)) = (A+B)′

Formal definition of functions (2)

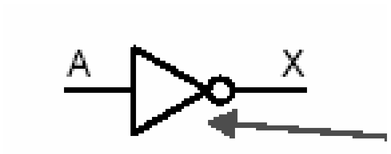
- By means of “boolean equation”
- Functional description of result

M = OR(AND(NOT(A),NOT(B)), AND(A,B))

$M = A'B' + AB$

NOT gate - the simplest one

- NOT gate - inverts the signal



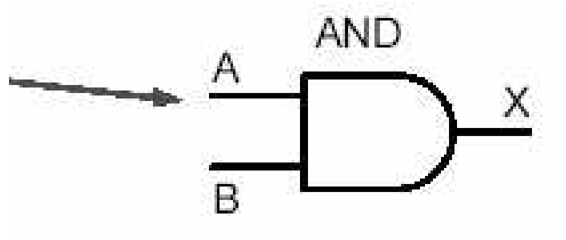
If A is 0, X is 1

If A is 1, X is 0

- A NOT gate is also called an **inverter**

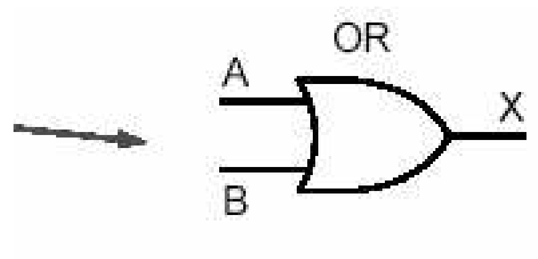
AND gate

- Output is 1 if all inputs are 1
  - In general, if the AND gate has N inputs, both input 1 AND input 2 AND … AND input N must be 1 for the output to be 1
- 2-input AND gate



OR gate

- Output is 1 if at least one input is 1
  - In general, if the OR gate has N inputs, input 1 OR input 2 OR … OR input N must be 1 for the output to be 1
- 2-input OR gate



A more complex example

- 2-input “equivalence” circuit
- The output is 1 if the inputs are the same
  - (i.e., both 0 or both 1)
- Boolean function:

  $M = A'B' + AB$

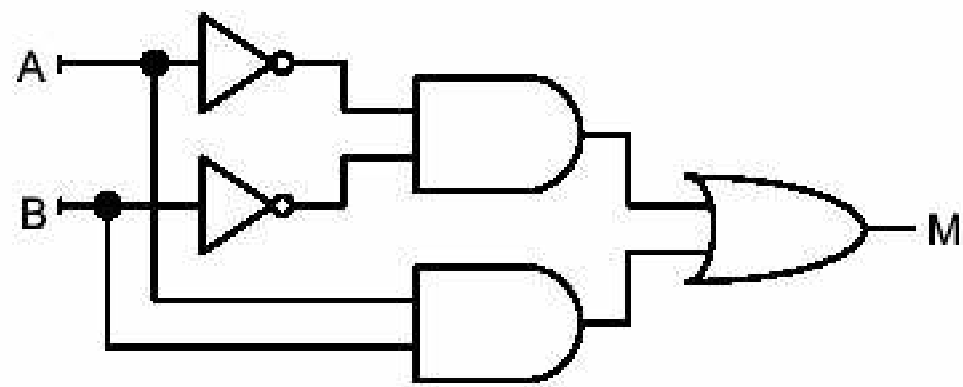
Truth table

A	B	M
0	0	1
0	1	0
1	0	0
1	1	1

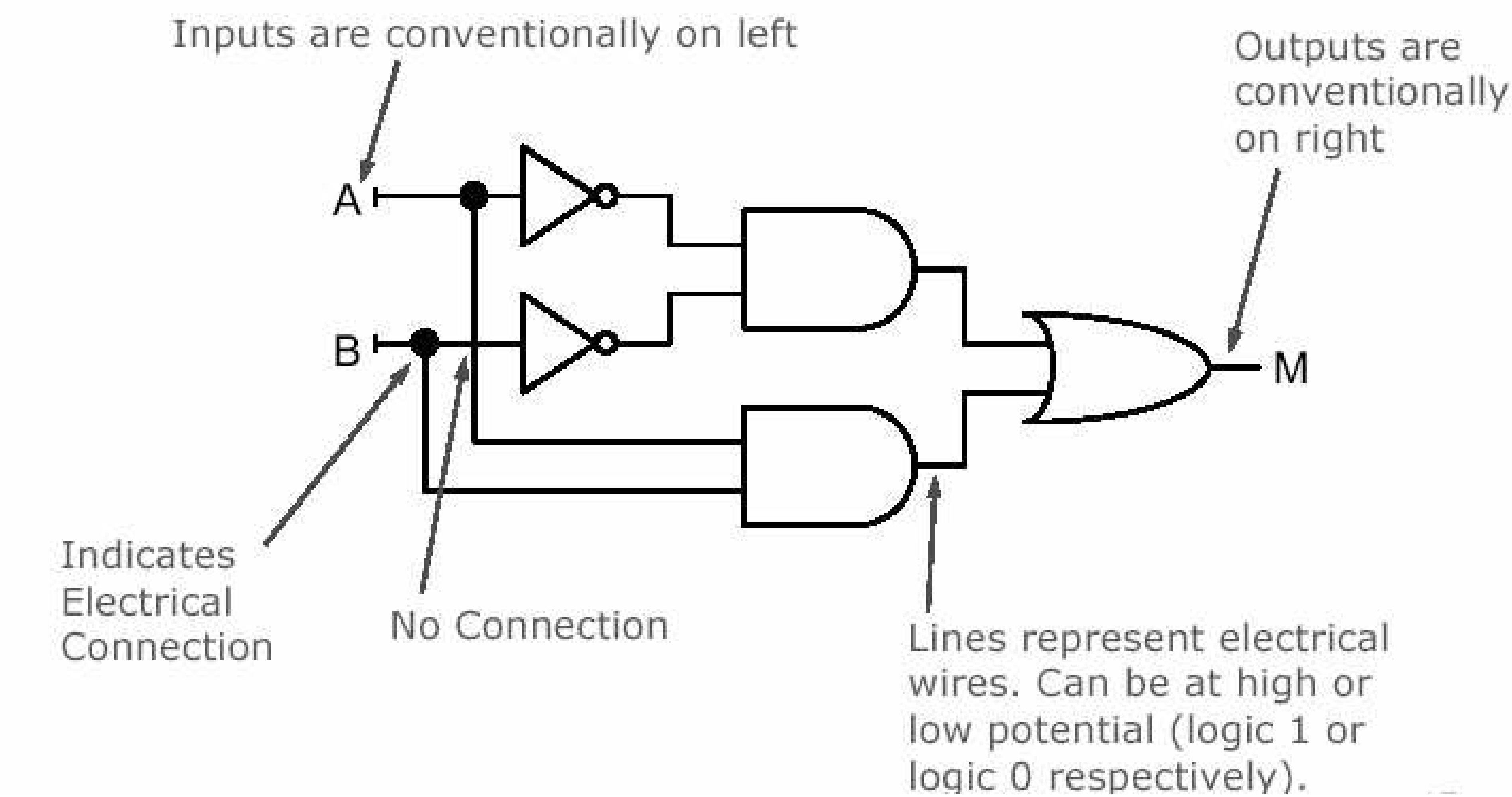
Formal definition of functions (3)

- By means of **logic circuits**
  - Combination of logic gates joined by wires

A	B	M
0	0	1
0	1	0
1	0	0
1	1	1



Conventions for logic circuits



Exercise (1)

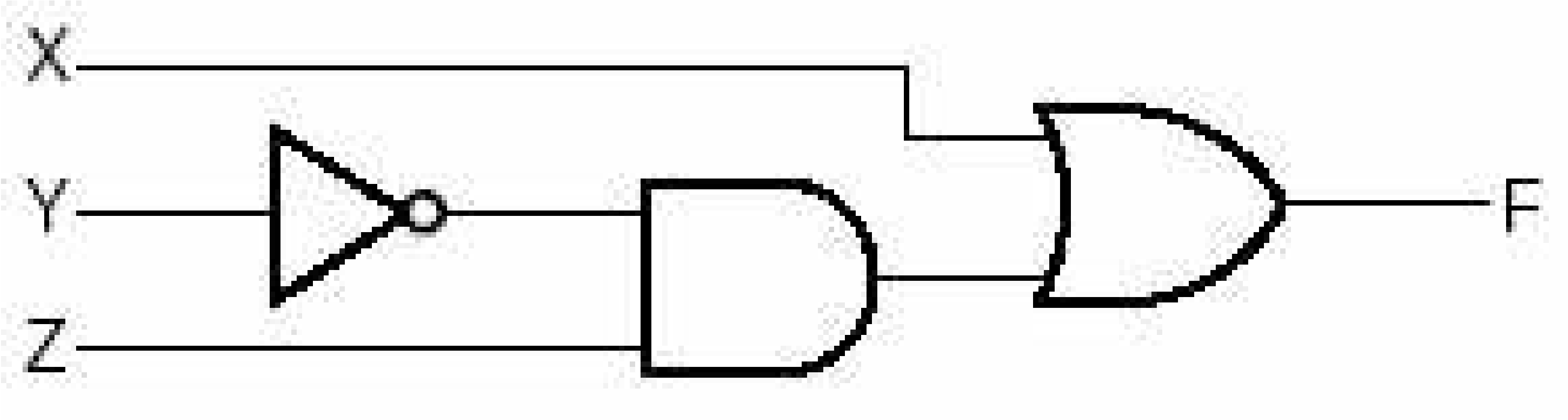
- Write the truth table and the logic circuit for

$F = X + Y'Z$

Truth table

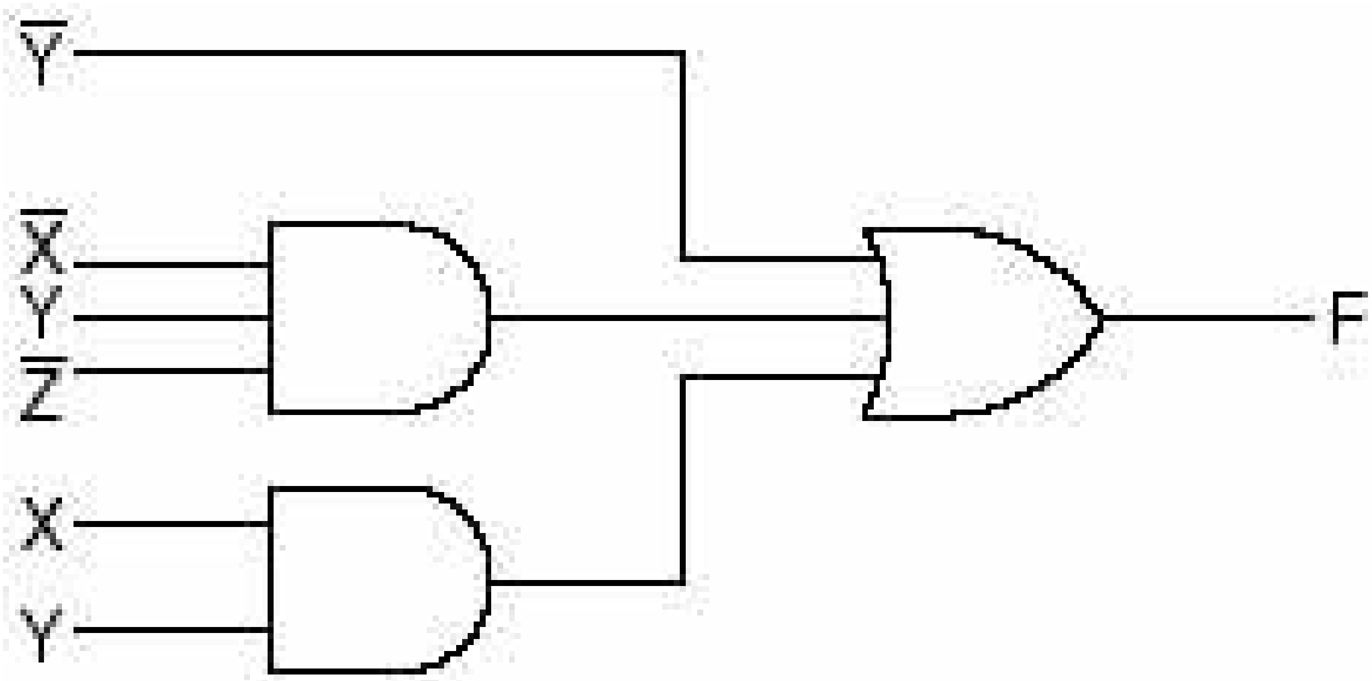
X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Logic Circuit



Exercise (2)

- Write the boolean function and its truth table for the following logic circuit



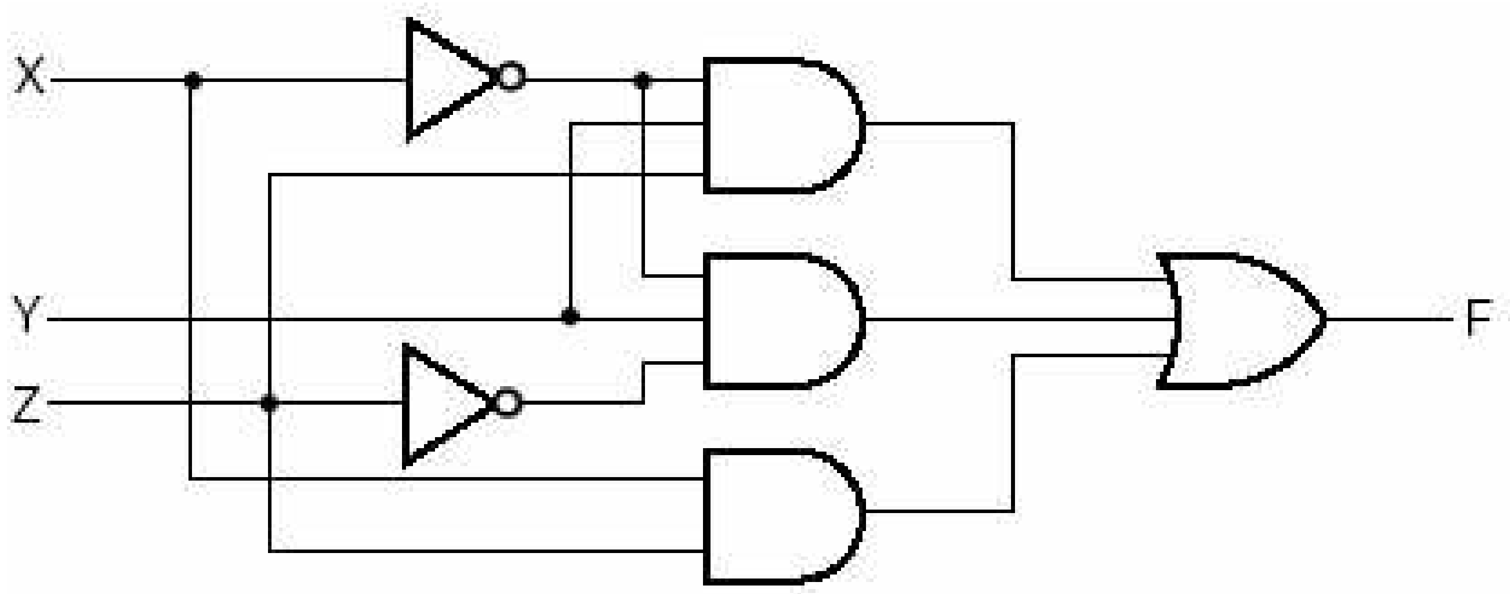
Function and Truth Table

- F = Y′ + X′YZ′ + XY

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Exercise (3)

- Write the boolean function and its truth table for the following logic circuit



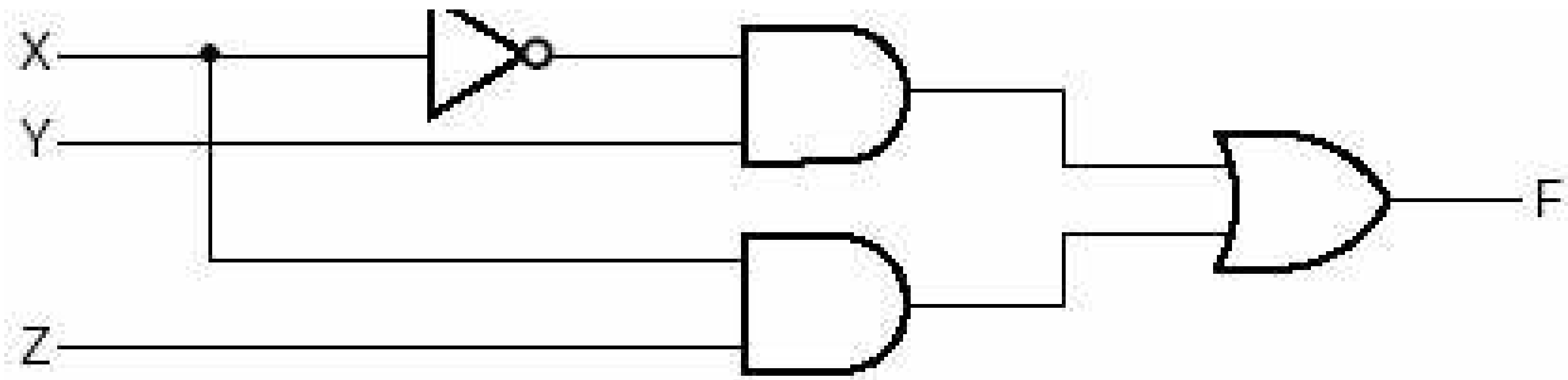
Function and Truth Table

- $F = X'YZ + X'YZ' + XZ$

X	Y	Z	F
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	0
1	1	1	1

A simpler equivalent circuit

- $F = X'Y + XZ$



Conversion between represent.

- Circuit ->
  - -> Boolean formula (left-to-right inspection)
  - -> Truth table (explicit case-by-case computation)
- Boolean formula ->
  - -> Circuit (to “normal” form, then inspection)
  - -> Truth table (explicit case-by-case computation)
- Truth table ->
  - -> Circuit (through boolean formula)
  - -> Boolean formula (explicit case-by-case)

Boolean Identities

$1A = A$	$0+A = A$	Identity
$0A = 0$	$1+A = 1$	Null
$AA = A$	$A+A = A$	Idempotent
$AA' = 0$	$A+A' = 1$	Inverse
$AB = BA$	$A+B = B+A$	Commutative
$(AB)C = A(BC)$	$(A+B)+C = A+(B+C)$	Associative
$A+BC = (A+B)(A+C)$	$A(B+C) = AB+AC$	Distributive
$A(A+B) = A$	$A+AB = A$	Absorption
$(AB)' = A'+B'$	$(A+B)' = A'B'$	De Morgan

Truth tables to verify De Morgan's theorem

A)

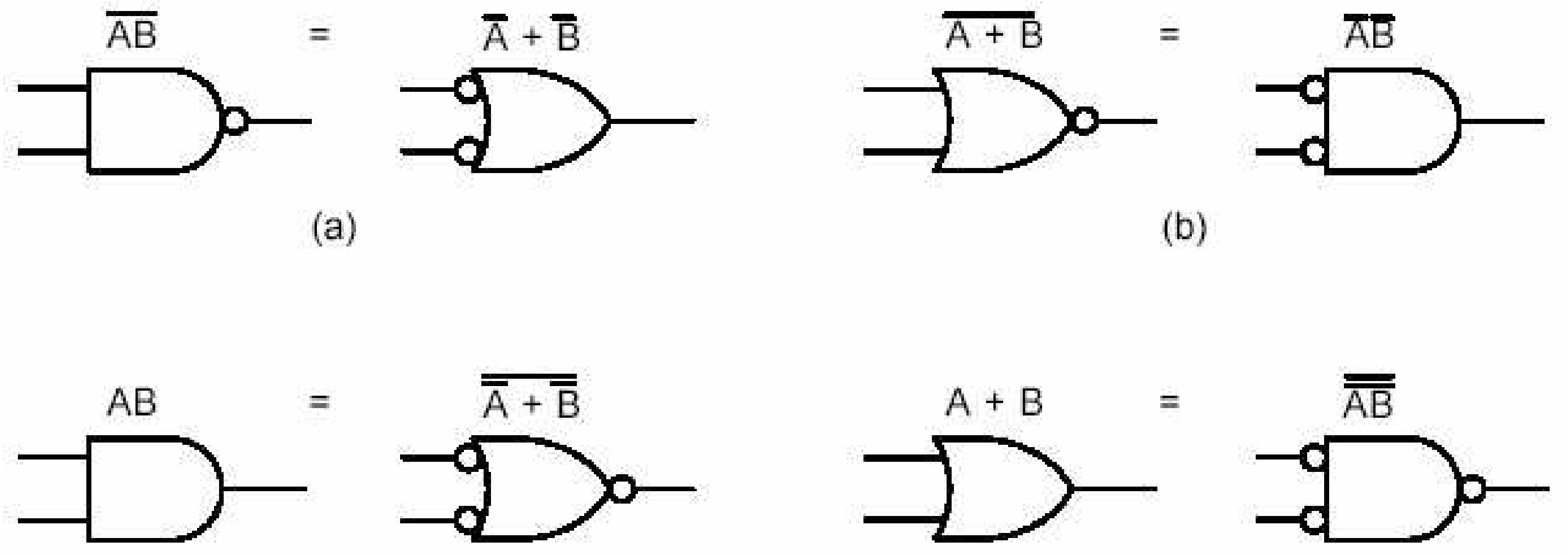
X	Y	$X + Y$	$\overline{X+Y}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

B)

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

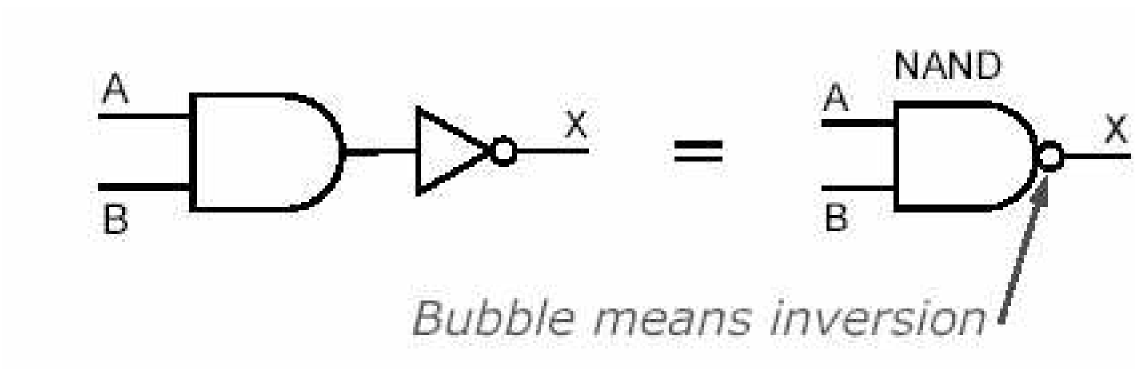
De Morgan circuit equivalents

- AND/OR can be interchanged if you invert the inputs and outputs



NAND gate - the negation of AND

- The opposite of the AND gate is the NAND gate (output is 0 if all inputs are 1)
- Logic diagram



Truth table

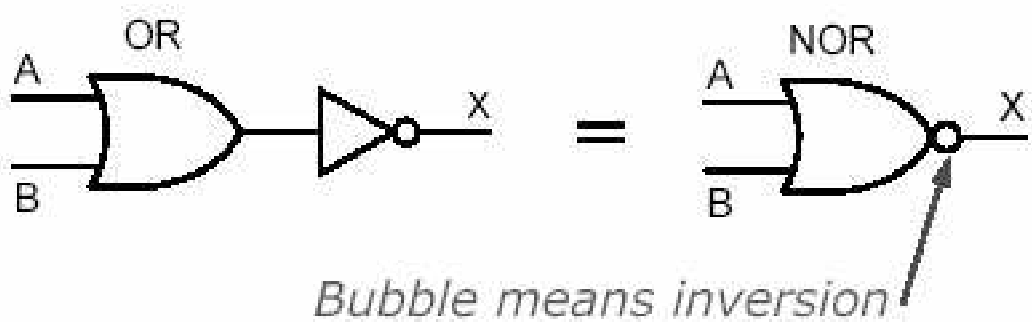
A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate - the negation of OR

- The opposite of the OR gate is the NOR gate (output is 0 if any input is 1)

Truth table

- Logic diagram



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

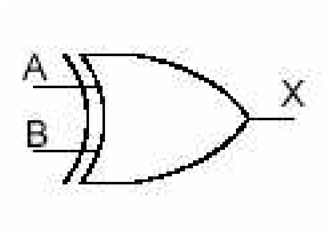
Exercise

- Write the truth table for:
  - a 3 input NAND gate
  - a 3 input NOR gate

XOR gate - the exclusive OR

- For a 2-input gate
  - Output is 1 if exactly one of the inputs is 1

- Logic diagram



- For > 2 inputs: output is 1 if an odd number of inputs is 1

Truth table

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Universal Gates

- How many logical functions there are?
- With *n* inputs there are $2^{(2^n)}$ possible logical functions

A	B	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*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

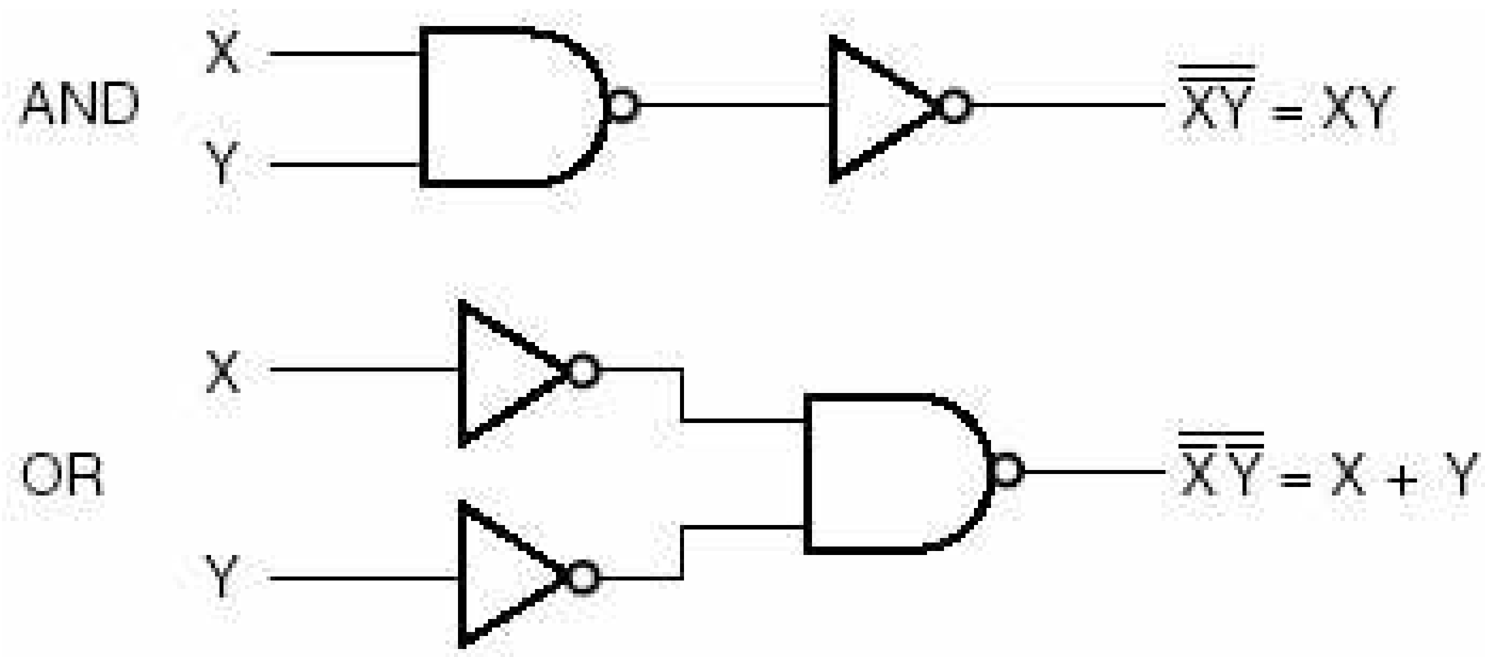
Universal Gates (2)

- AND, OR, NOT can generate all possible boolean functions (**boolean algebra**)
- Is it possible to use fewer basic operations?

Universal Gates (3)

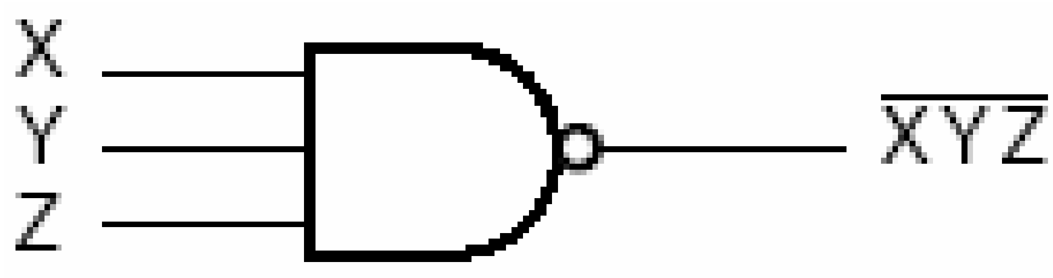
- AND, NOT are enough !
- OR, NOT are enough !
- Even NAND alone or NOR alone are enough !

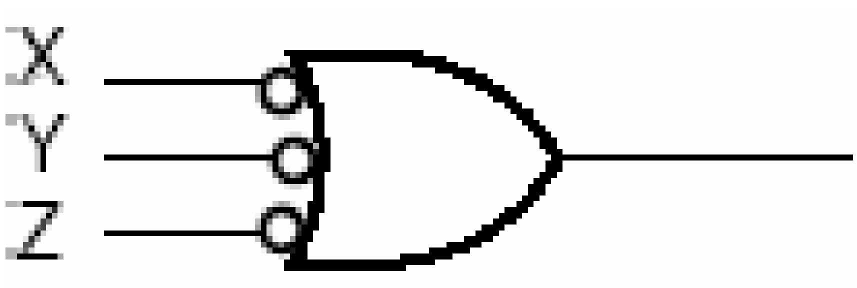
How NAND simulates AND, OR



- Simulation of NOT ???

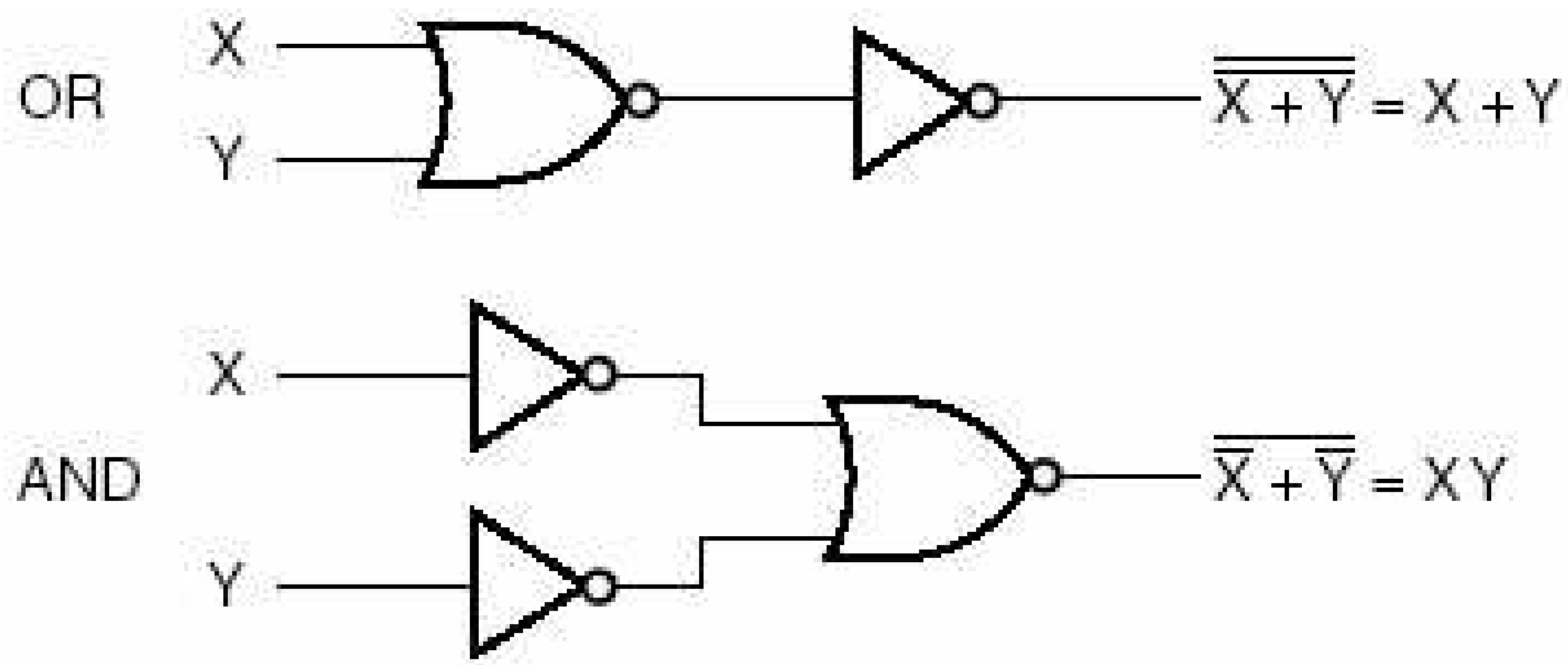
Alternative NAND representations





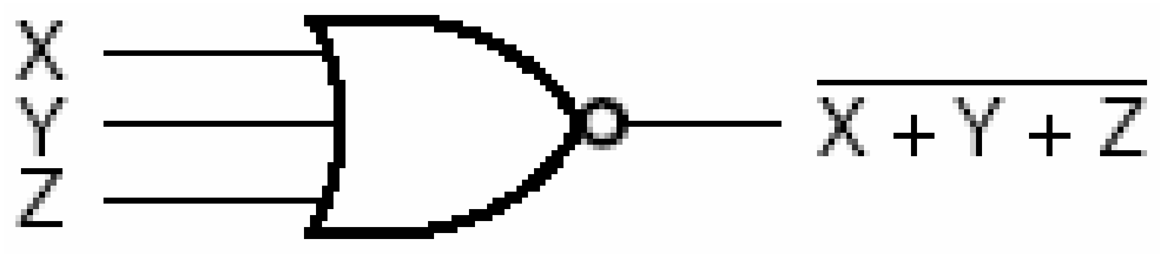
$\overline{X} + \overline{Y} + \overline{Z} = \overline{XYZ}$

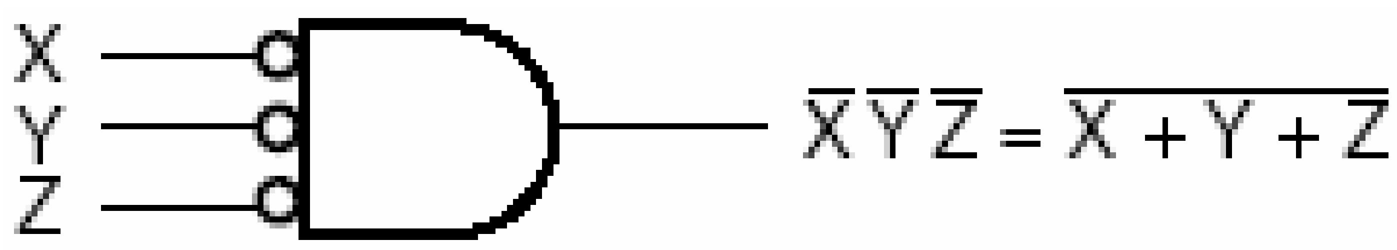
How NOR simulates AND, OR



- Simulation of NOT ???

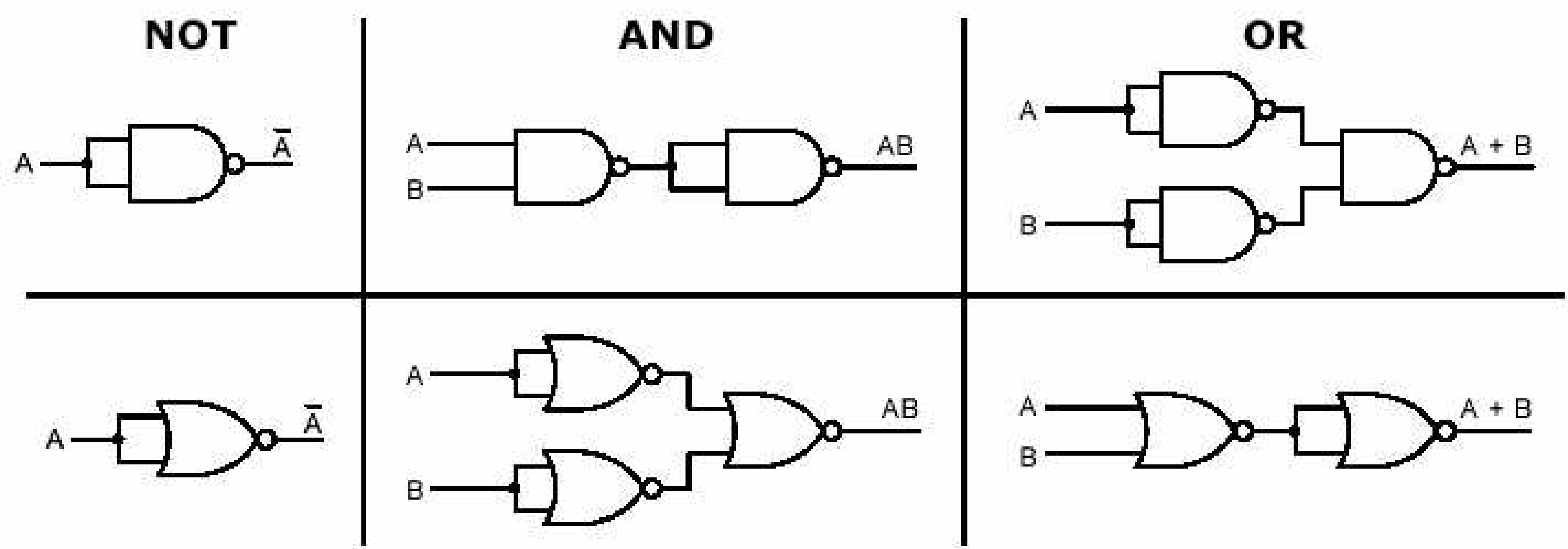
Alternative NOR representations





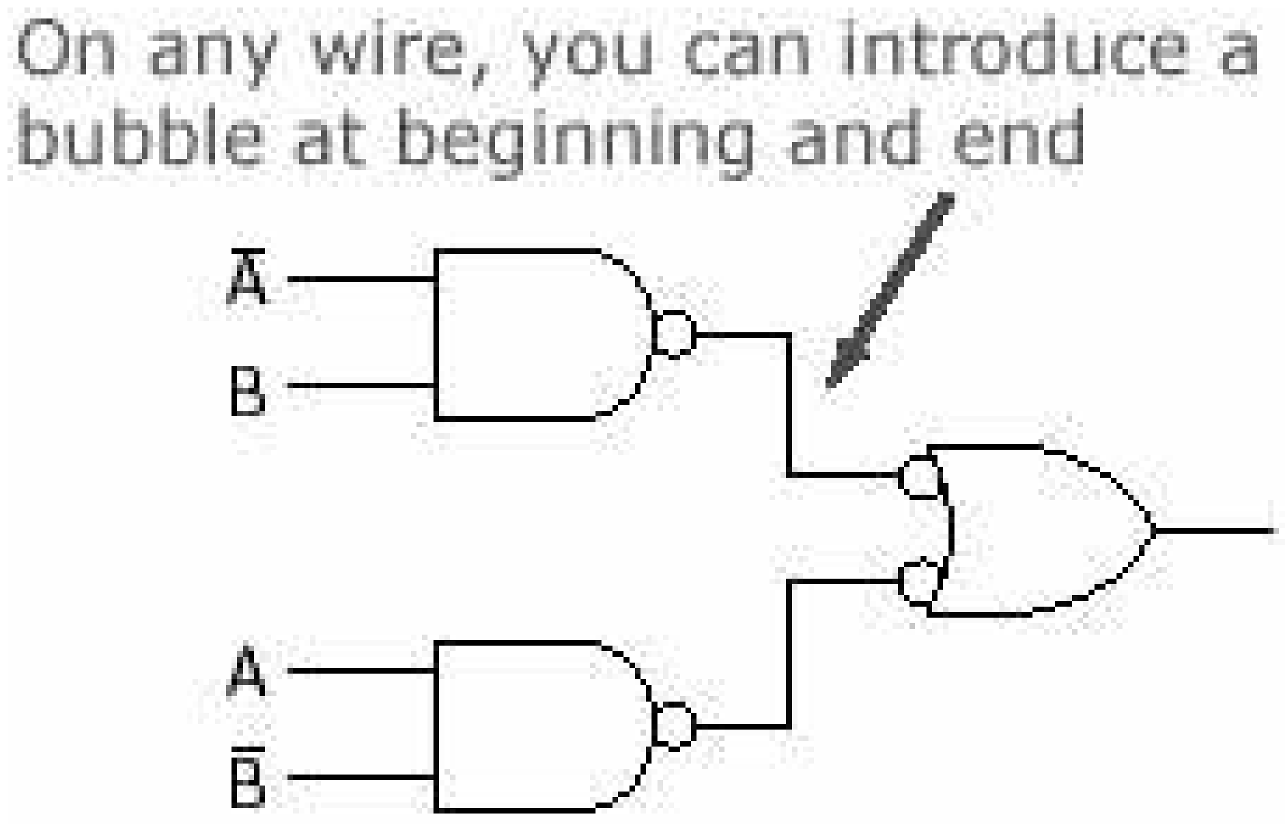
Gate equivalence

- Any AND, OR, NOT gate can be obtained using just NAND gates or just NOR gates



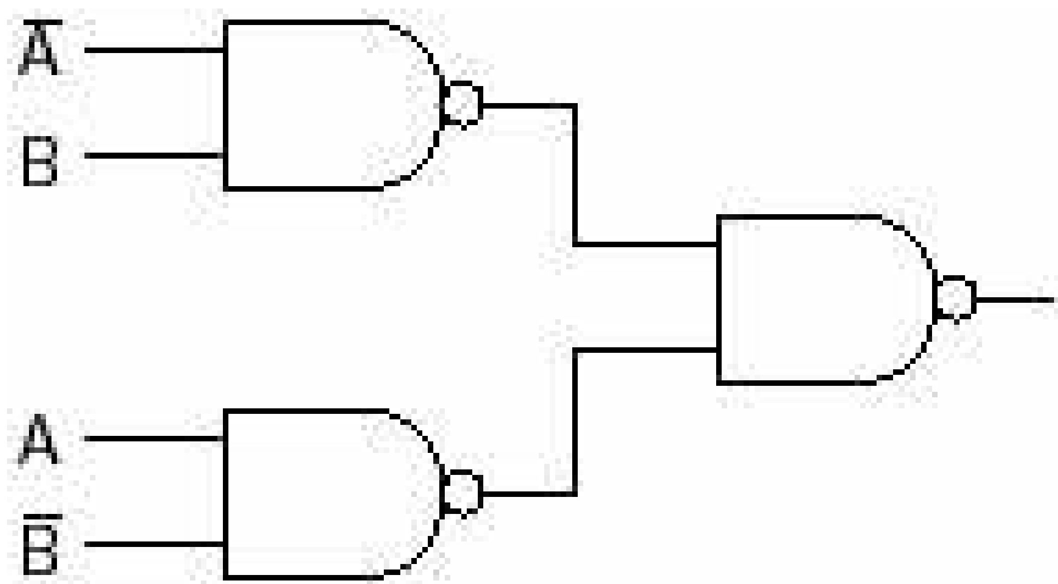
- Consequence: any circuit can be constructed using just NAND gates or just NOR gates (easier to build)

Equivalence modifications (1)



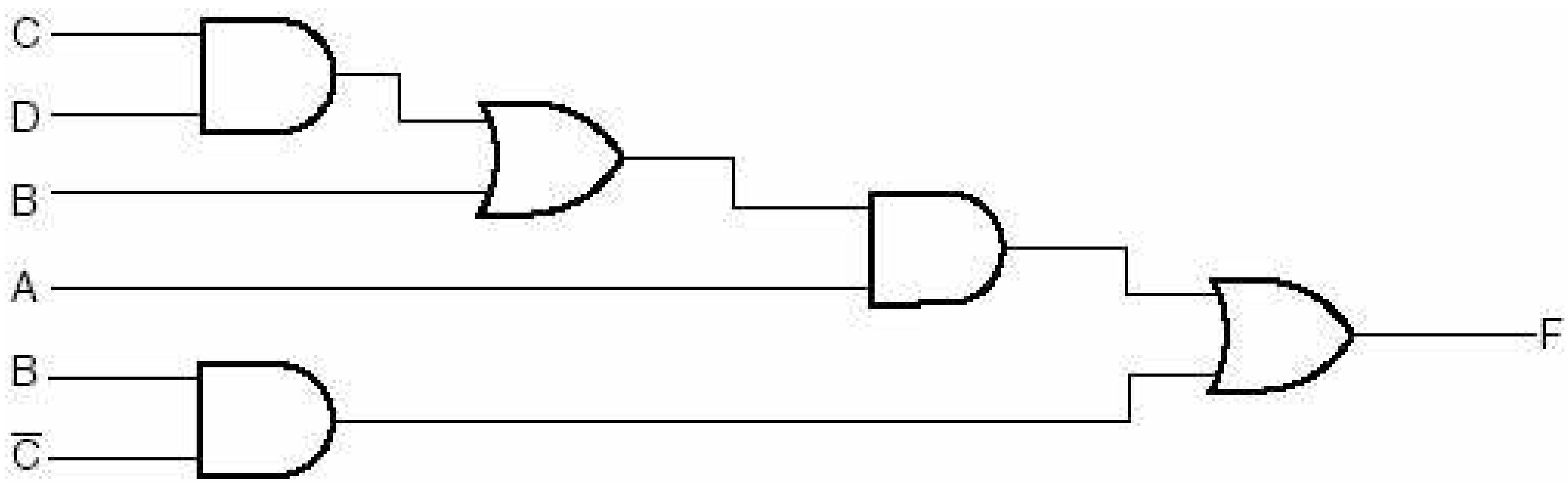
Equivalence modifications (2)

- Substitute equivalent gates

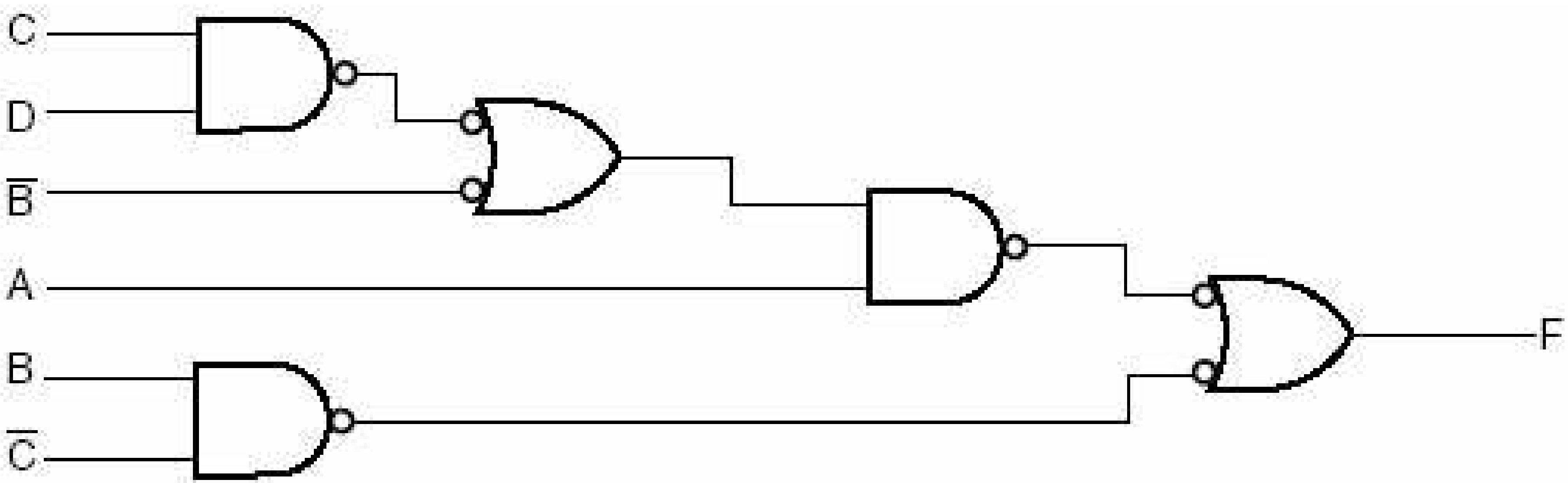


Transforming OR, AND to NAND

- Transform the following circuit



Solution

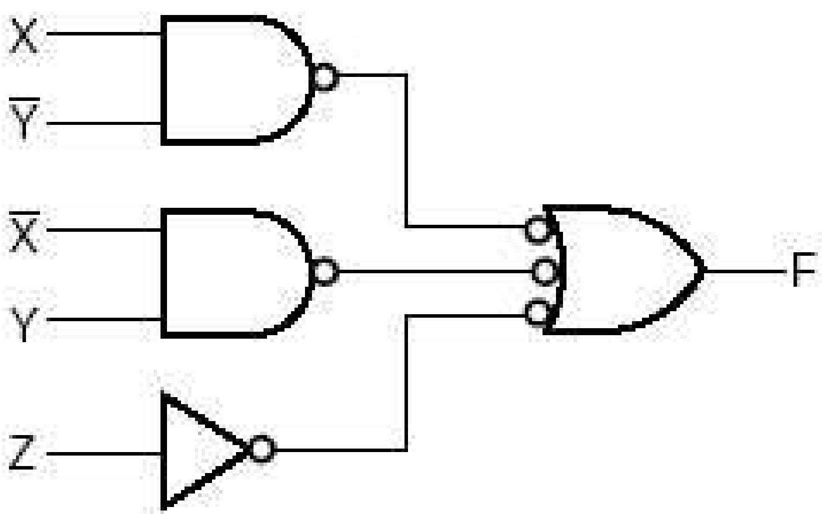


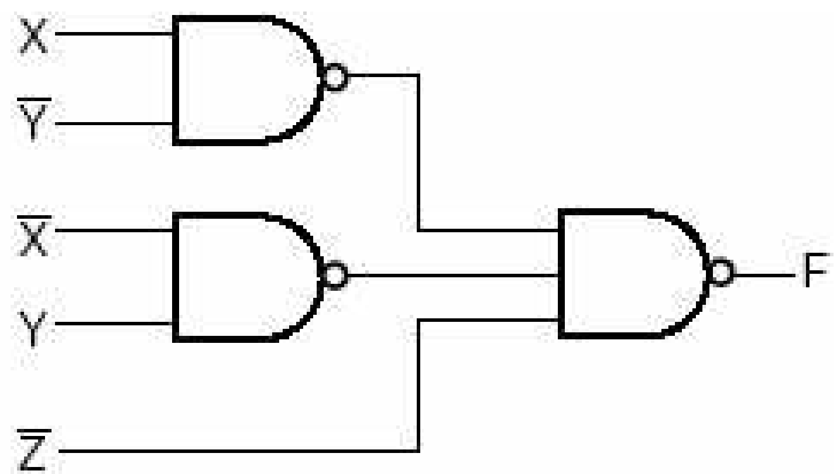
Exercise

- Write a NAND only logic circuit for

  $F = XY' + X'Y + Z$

Solution





Exercise

- Write a NAND only logic circuit for the exclusive OR function (XOR)

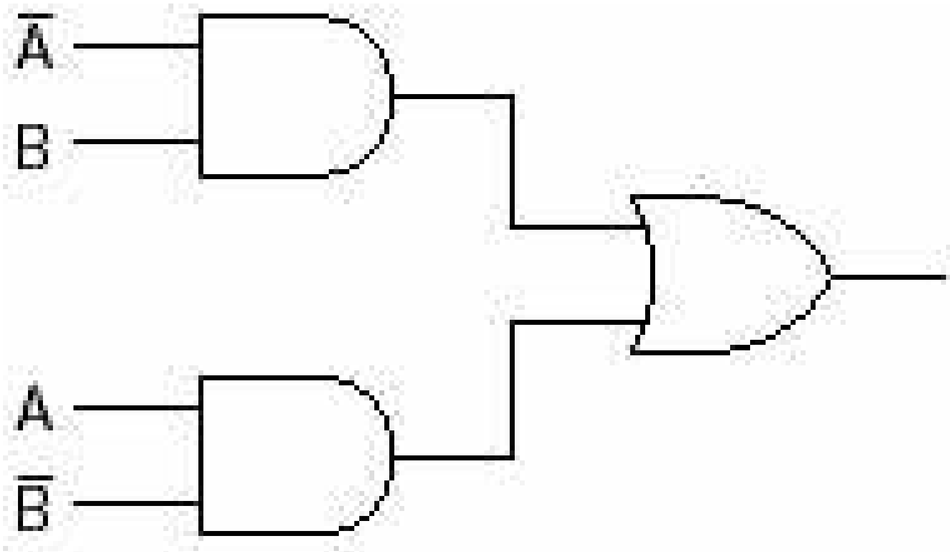
  XOR(A,B) = A'B + AB'

Solution (1)

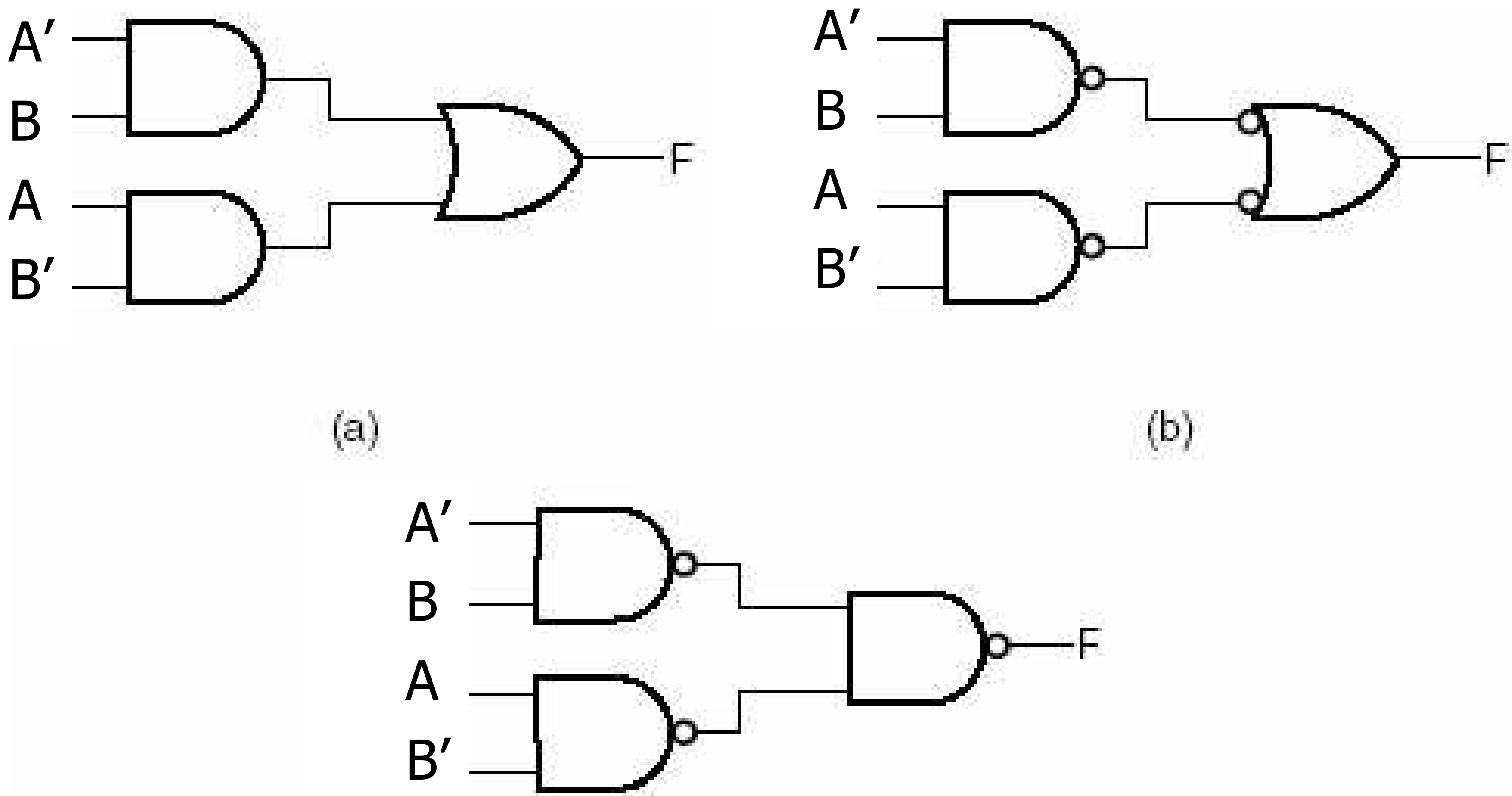
- Truth table

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

initial circuit



Solution (2): equivalence transform.



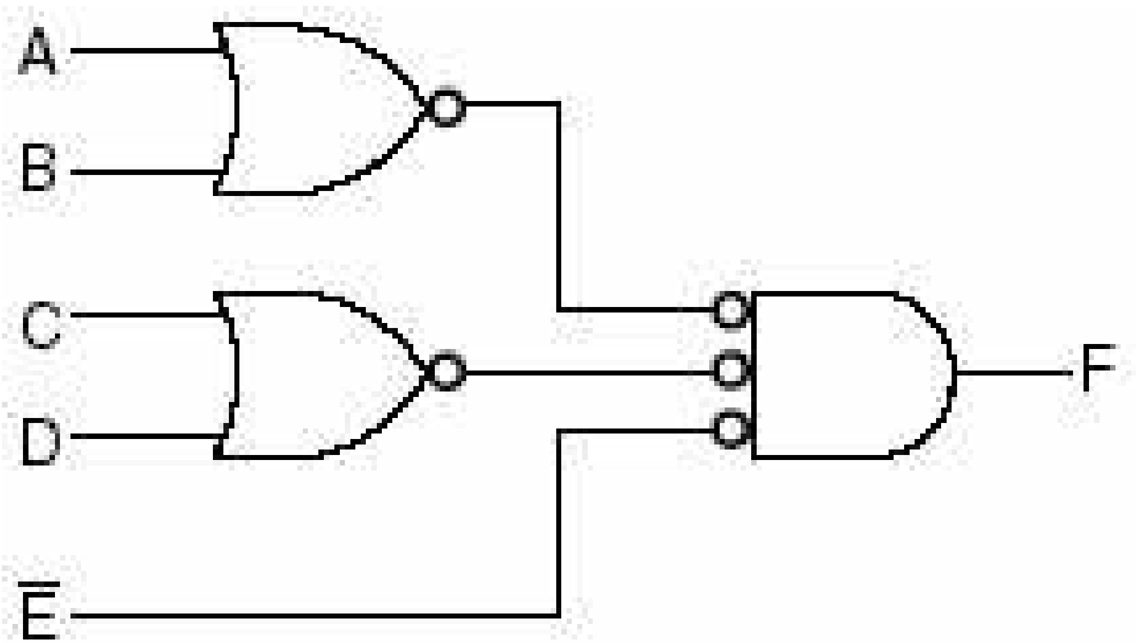
Exercise

- Write a NOR only logic circuit for

  $F = (A+D)(C+D)E$

Solution

- Direct realization

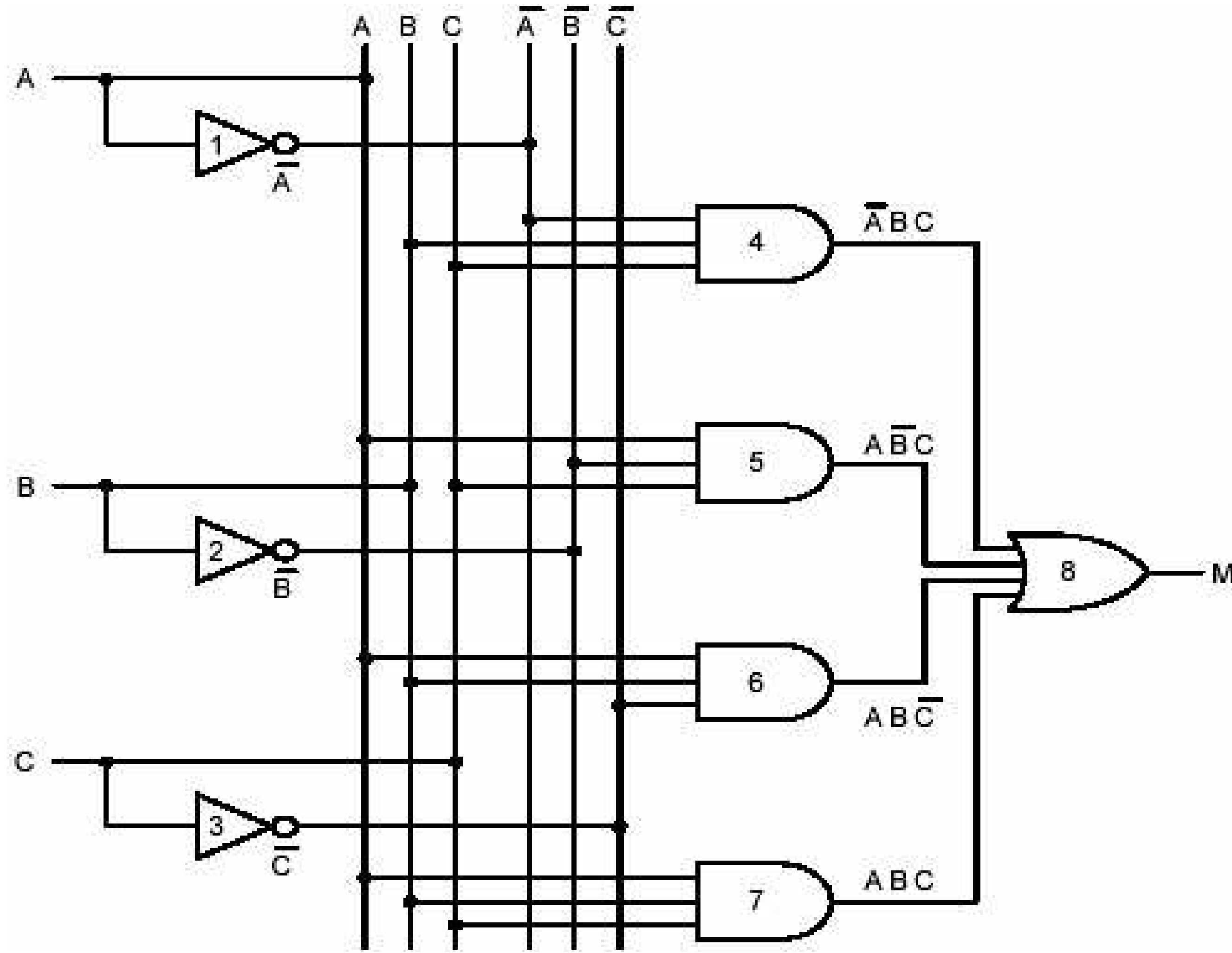


Boolean function implementation

- Any function can be implemented as the OR of the AND combinations of its inputs
  - Called **sum of products** (SOP)
- Start from the truth table
  - For each 1 in the output
  - Write its inputs in AND
  - Write these in OR
- M=A′BC+AB′C+ABC′+ABC

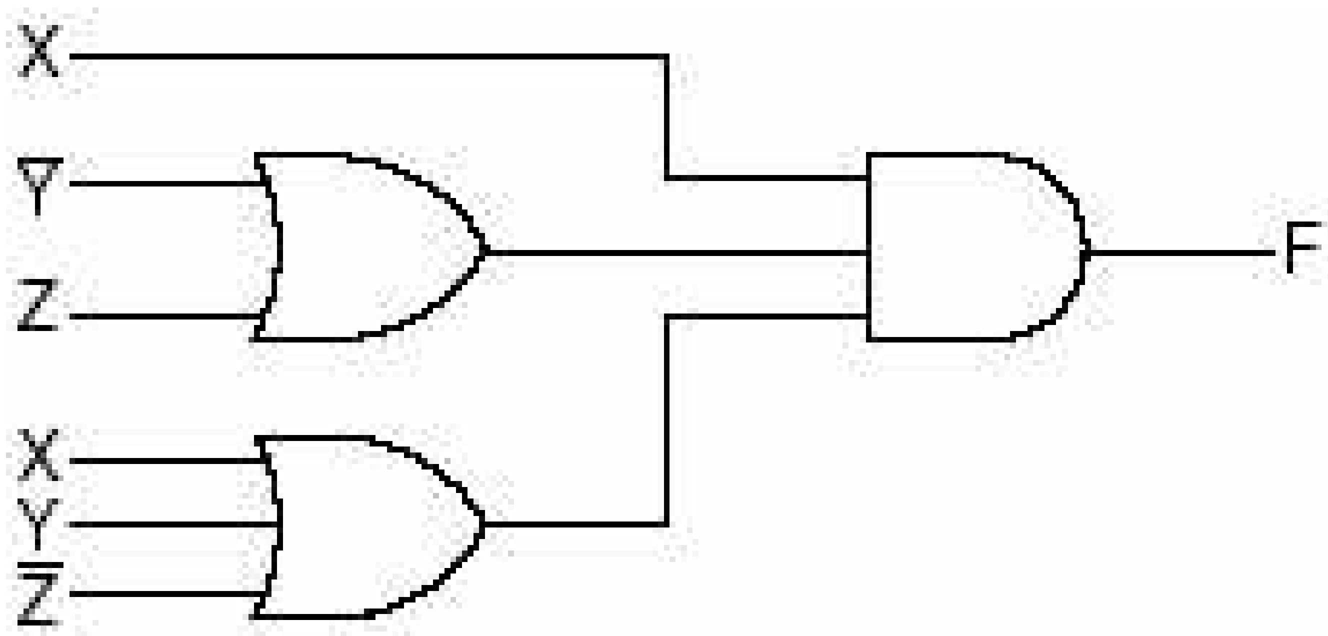
A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	(1)
1	0	0	0
1	0	1	(1)
1	1	0	(1)
1	1	1	(1)

Equivalent Logic Circuit



Exercise

- Write the boolean function and its truth table for the following logic circuit



Solution: truth table

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Solution: boolean formula

$$F = X.(Y'+Z).(X+Y+Z')$$
$$= (XY'+XZ).(X+Y+Z')$$
$$= XY'X+XY'Y+XY'Z'+XZX+XZY+XZZ'$$
$$= XY'+XY'Z'+XZ+XZY$$
$$= X(Y'+Z)+X(Y'Z'+YZ)$$

N.B.: $Y'Z'+YZ$ is not 1 !!!

Exercise

- Express $Z=(A(B+C(A'+B')))'$ as sum of products

Solution

$$Z=(A(B+C(A'+B')))'$$
$$= A' + (B+C(A'+B'))'$$
$$= A' + B'(C(A'+B'))'$$
$$= A' + B'(C'+(A'+B')')$$
$$= A' + B'(C'+AB)$$
$$= A' + B'C' + ABB'$$
$$= A' + B'C' + A0$$
$$= A' + B'C' + 0 \qquad = A' + B'C'$$

Boolean function implem. (2)

- Any function can be implemented as the AND of the OR combinations of its inputs
  - Called **product of sums** (POS)
- Start from the truth table
  - For each 0 in the output
  - Write its inputs in AND
  - Write these **negated** in AND
  - Obtain $F = Z_0' \,.\, Z_1' \,.\, Z_2' \ldots$
  - Finally, apply De Morgan to F

A	B	C	F
0	0	0	(0)
0	0	1	(0)
0	1	0	1
0	1	1	1
1	0	0	(0)
1	0	1	(0)
1	1	0	1
1	1	1	(0)

Boolean function implem. (3)

- De Morgan (general): $(ABC)'=A'+B'+C'$
- $F=(A'B'C')'.(A'B'C)'.(AB'C')'.(AB'C)'.(ABC)'$

$=(A''+B''+C'').(A''+B''+C').(A'+B''+C'').$

$.(A'+B''+C').(A'+B'+C')$

$=(A+B+C).(A+B+C').(A'+B+C).$

$.(A'+B+C').(A'+B'+C')$

Boolean function implem. (4)

- Shortcut procedure for POS form (use only if you know what you are doing!)
  - Complement the table by substituting everywhere a 0 with a 1 and a 1 with a 0
  - Write a SOP form for the complemented table
  - Complement the formula by substituting everywhere and AND with an OR and an OR with an AND
  - Why does it work ???

Canonical Form for boolean functions

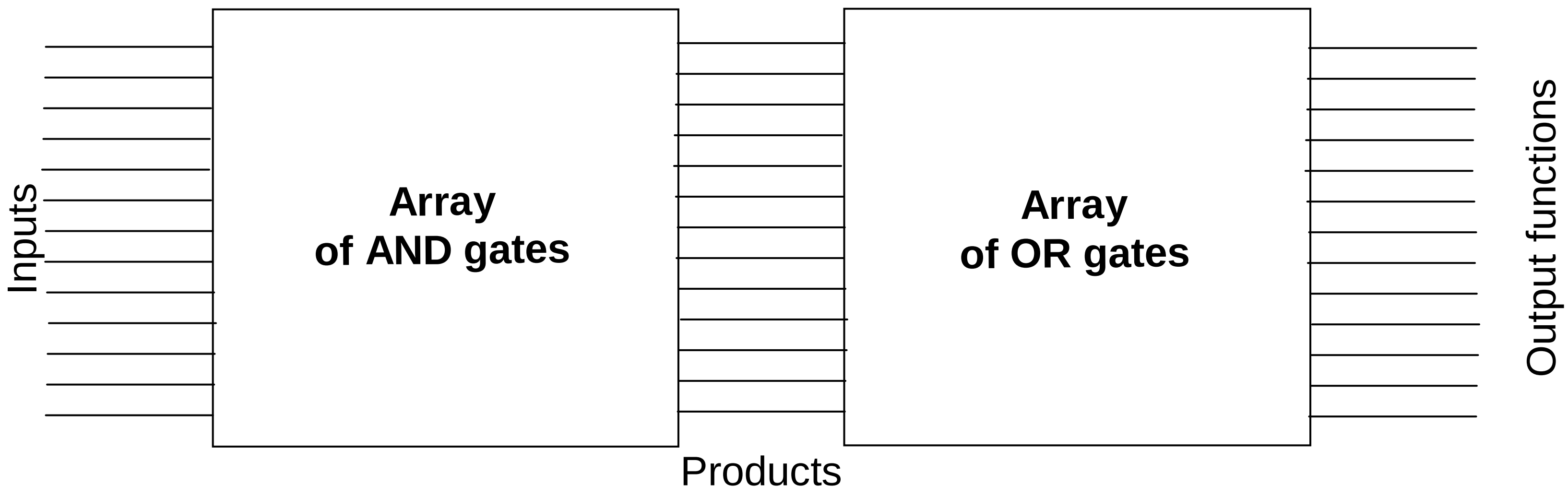
- It is a “standard” way of expressing SOP or POS supporting realization by means of standard electronic components
- It is:
  - a sum of minterms, for SOP
  - a product of maxterms for POS
- A minterm is a product containing all variables, either in the positive form or in the negative form
- A maxterm is a sum containing all variables, either in the positive or in the negative form.
- Examples:

F = (A’+B+C) . (B’+C)	is **<u>not</u>** in a POS canonical form
M = AB + A’BC	is **<u>not</u>** in a SOP canonical form

Standard implementation

- Given a boolean function expressed as sum of products or as product of sums it can be directly implemented in a circuit
- PLA: programmable logic array
- A PLA for sum of products is made by a first module combining inputs to form products, followed by a second module combining products to give the desired function

Schema for a sum-of-products PLA

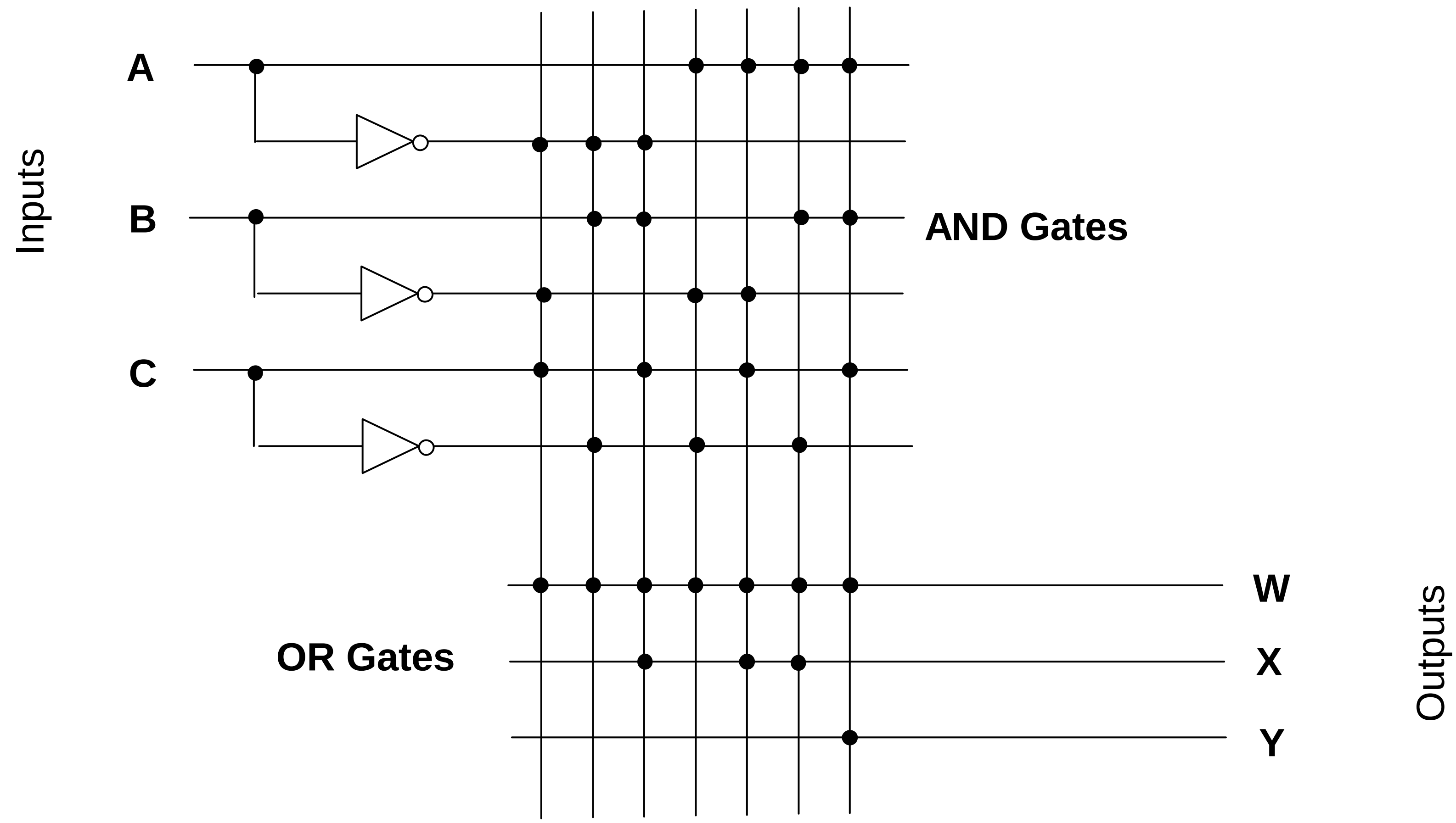


- Inputs are variables and their negation
- Each output line realizes a boolean function

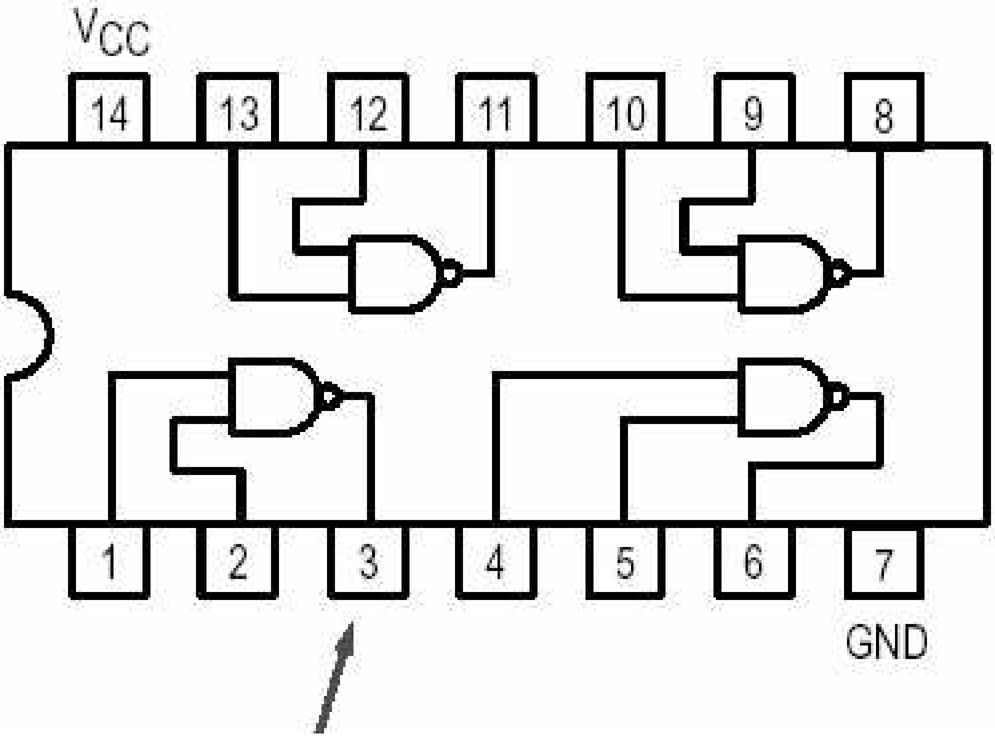
An example

A	B	C	W	X	Y
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

The PLA implementation



Real circuits



Pin spacing is 0.1" x 0.3"; chip is about 15mm long

- 74LS00 - has four 2-input NAND gates
- Small scale integration (SSI)

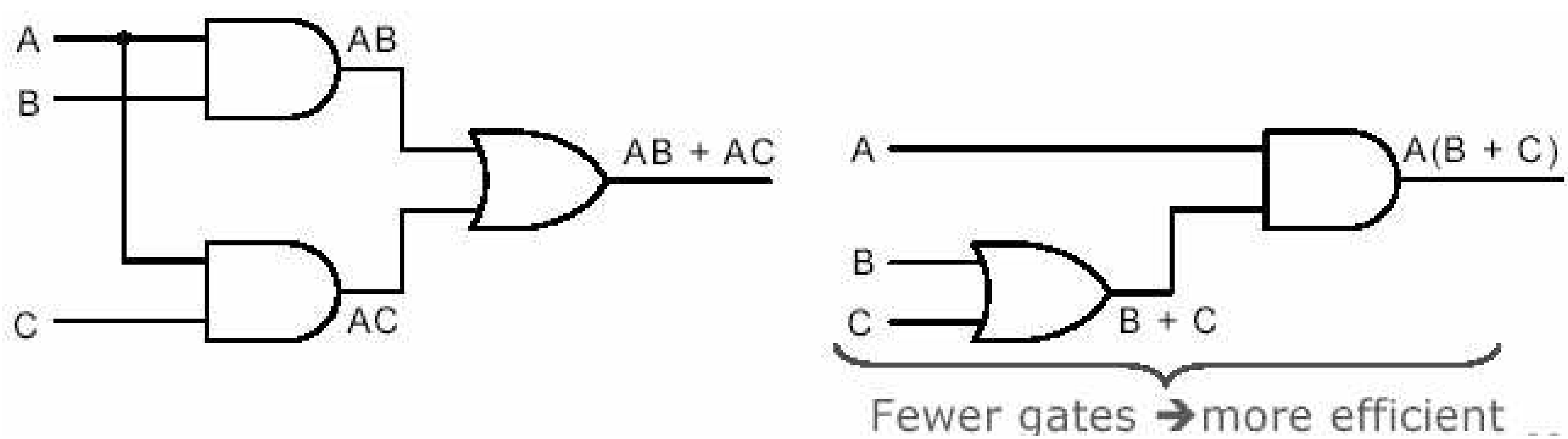
Integrated circuits

- Scales of integration

• (Small)	SSI: 1-10 gates
• (Medium)	MSI: 10-100 gates
• (Large)	LSI: 100-100.000 gates
• (Very Large)	VLSI: > 100.000 gates

Equivalent Functions

- Sum of products (or product of sums) is not necessarily the more efficient form
- Manipulate boolean function to give an equivalent function
- Example: M = AB + AC = A(B+C)



Minimization procedures

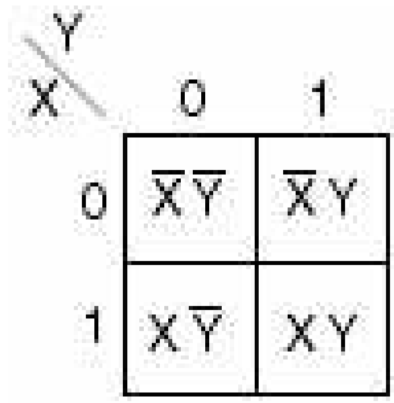
- Karnaugh’s maps (by hand)
- Used to minimize boolean functions of up to 4-5 input variables
- For more variables use the method of Quine-McKluskey (programmable)

Karnaugh's Maps (KM)

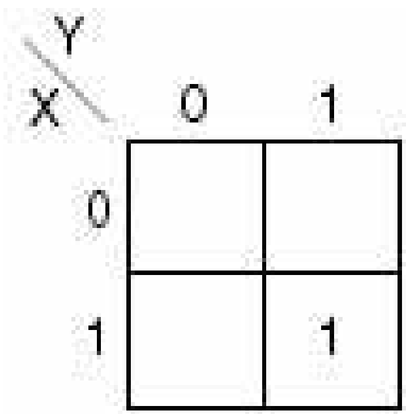
- Grid-like representation for boolean functions
- Minterms with just one variable different occupies adjacent cells
- Consider only 1s in the representation (focusing on a SOP representation)
- IDEA: if 2 adjacent cells have a 1 the function can be simplified

A KM for 2-variable functions

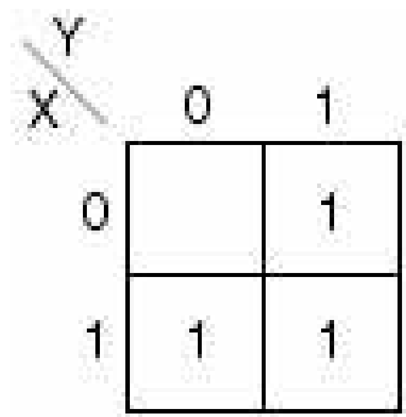
- The generic KM



- Function F = XY



Function F = X + Y

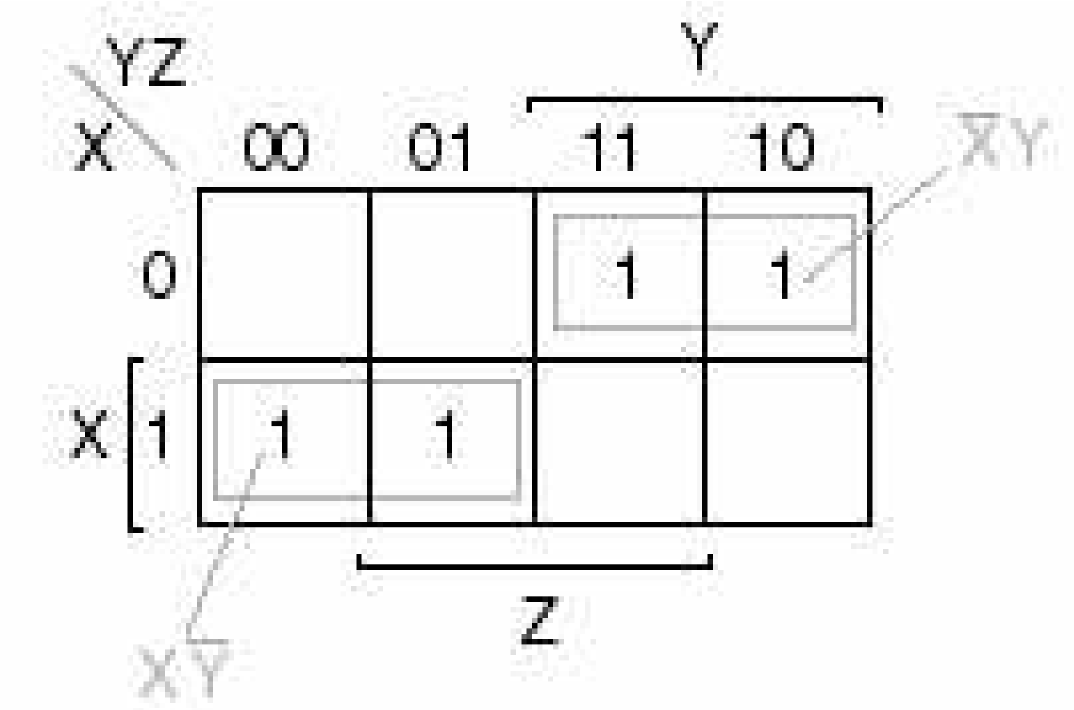


A KM for 3-variable functions

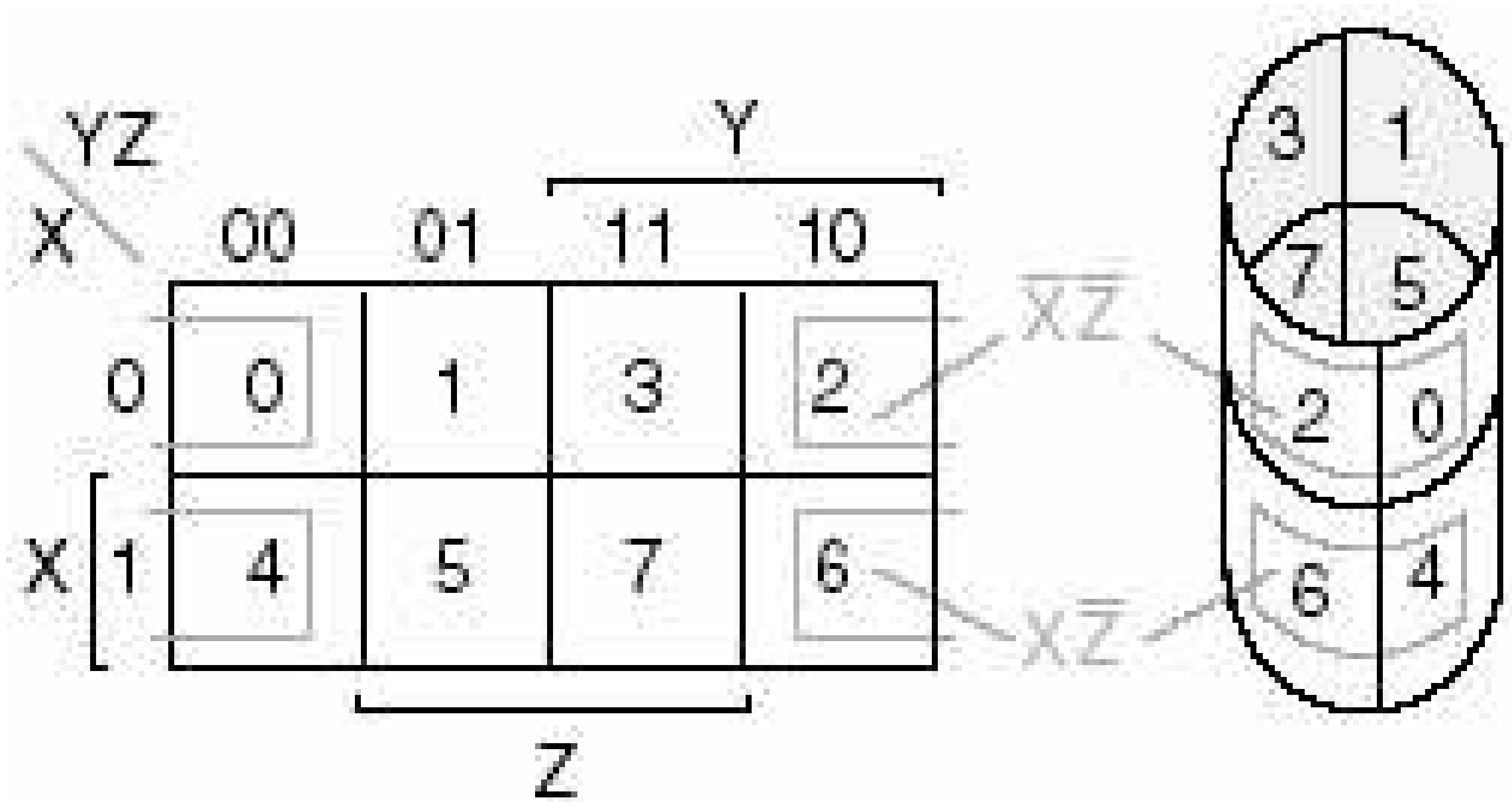
				Y	
	X \ YZ	00	01	11	10
	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	$\overline{X}\,\overline{Y}\,Z$	$\overline{X}\,Y\,Z$	$\overline{X}\,Y\,\overline{Z}$
X	1	$X\,\overline{Y}\,\overline{Z}$	$X\,\overline{Y}\,Z$	$X\,Y\,Z$	$X\,Y\,\overline{Z}$
			Z		

An example

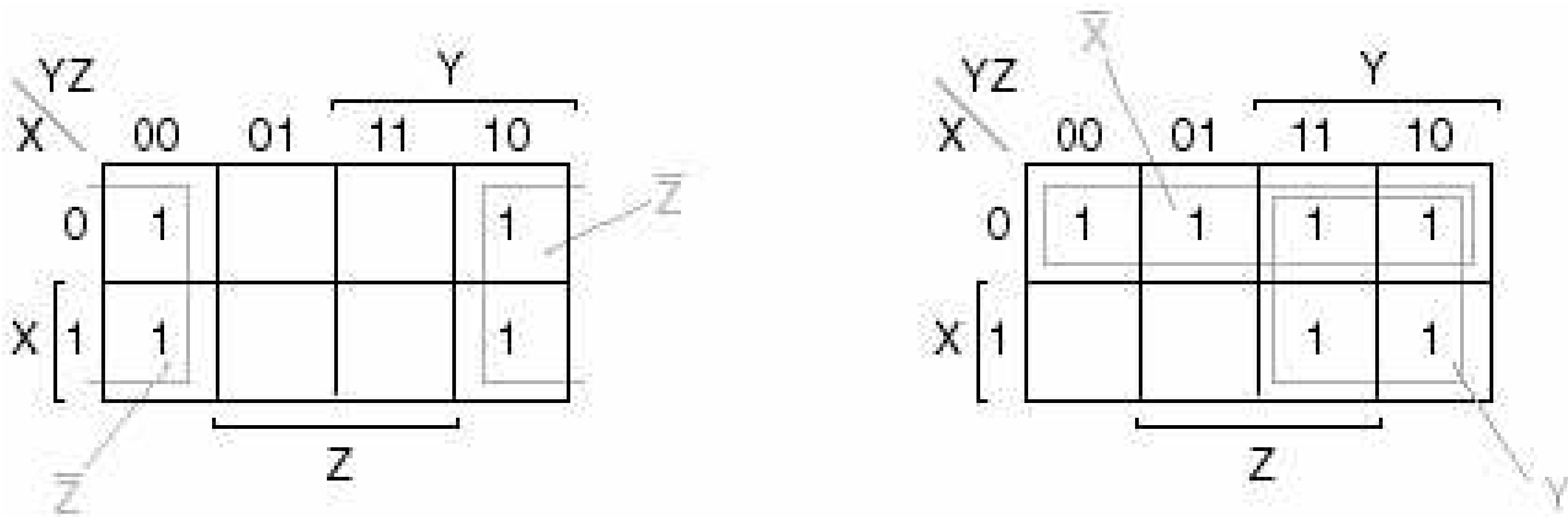
- F = XY′Z + XY′Z′ + X′YZ + X′YZ′



Circular adjacencies for 3 variables

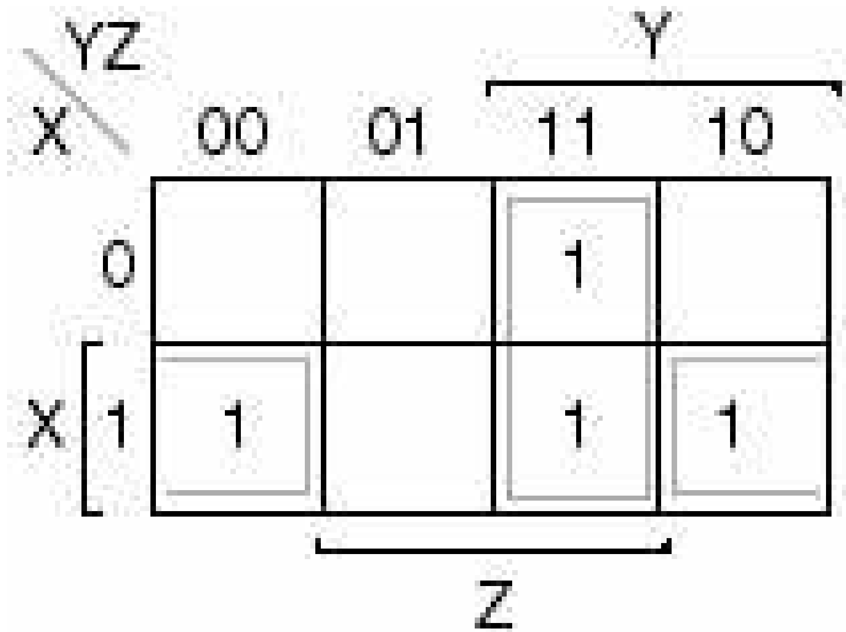


Four 1 adjacents



Exercise (1)

- Which is the minimal function F1 for this KM?

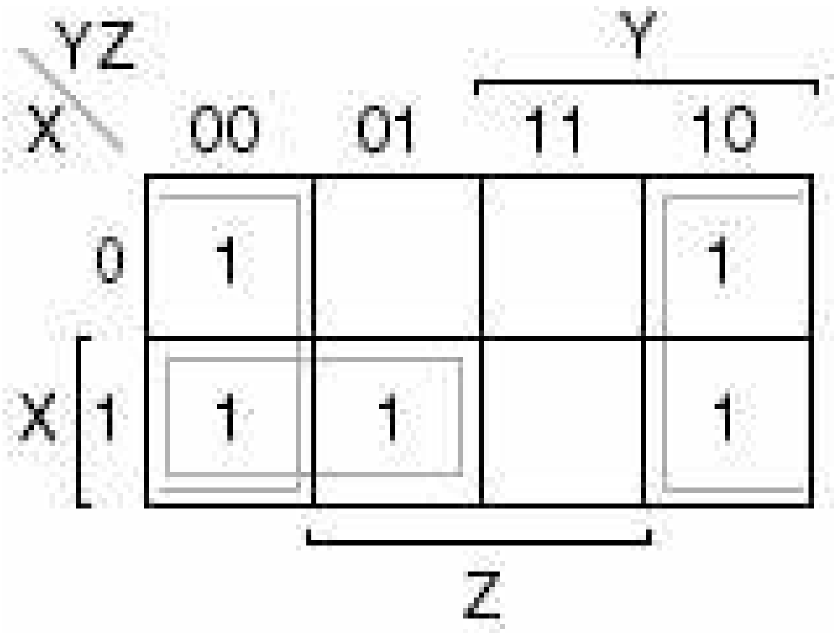


Solution

- $F1 = YZ + XZ'$

Exercise (2)

- Which is the minimal function F2 for this KM?



Solution

- $F2 = Z' + XY'$

k-cube of 1s

- Is a set of 2^k adjacent cells in *k* dimensions, where at most *k* variables change value
- 0-cube, 1 cell, a minterm
- 1-cube, 2 adjacent cells
- 2-cube, 4 adjacent cells
- 3-cube, 8 adjacent cells
- ....

Prime implicants

- $F = P_1 + P_2 + P_3 + \dots$
- A *k*-cube is also called an **implicant**
  - Infact, it is a term P_n which implies the function F,
  - i.e. if P_n is *true* then F is *true*
- A *k*-cube is a **prime implicant for F** if it does not imply any other implicant of F

Minimal representation

- A prime implicant can be chosen by selecting a *k*-cube in the KM which is not contained in a larger *h*-cube (**maximal *k*-cube**)

- $F = P_1 + P_2 + P_3 + \dots$

has a **minimal representation** if:

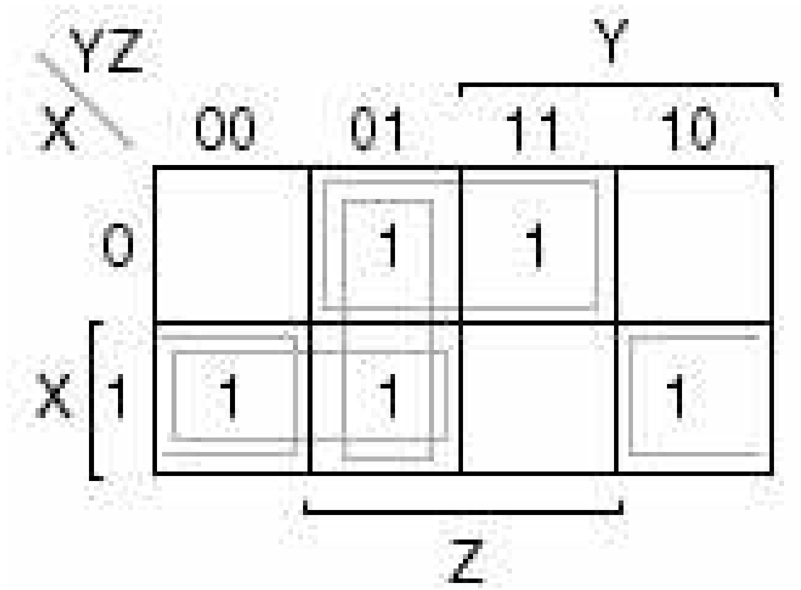
1. Each P_n is a prime implicant
2. There is a minimum number of them

Minimality procedure

1. Find maximal *k*-cubes (prime implicants)
2. If a 1 is covered by only one maximal *k*-cube this has to be chosen (**essential** prime implicants)
3. Select a minimum number of the remaining *k*-cube so to cover all 1s not covered by essential prime implicants

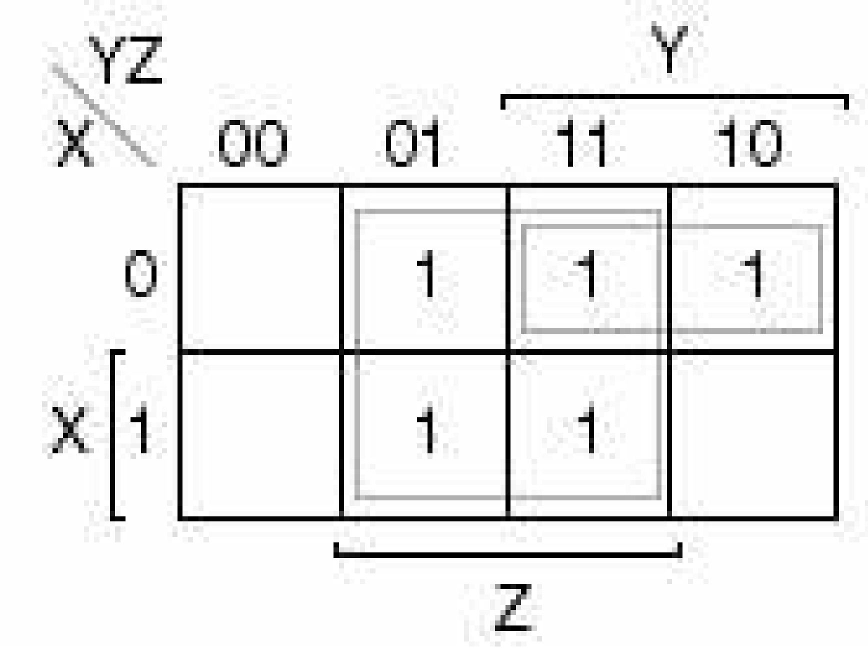
Exercise (3)

- Which is the minimal function F3 for this KM?



Exercise (4)

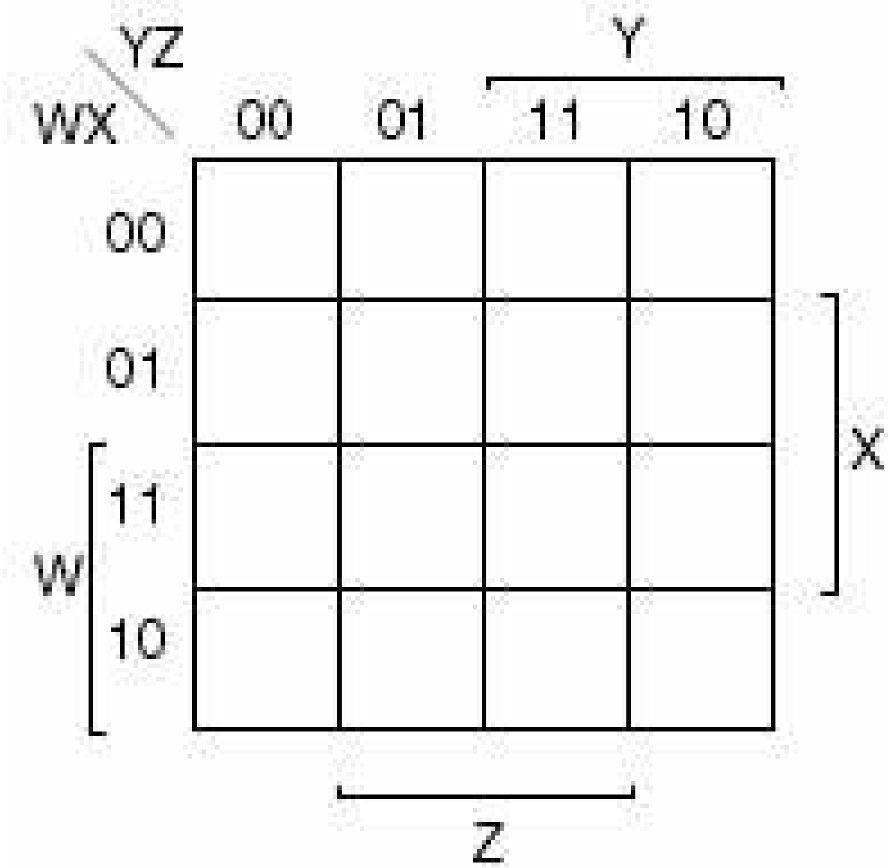
- Which is the minimal function F4 for this KM?



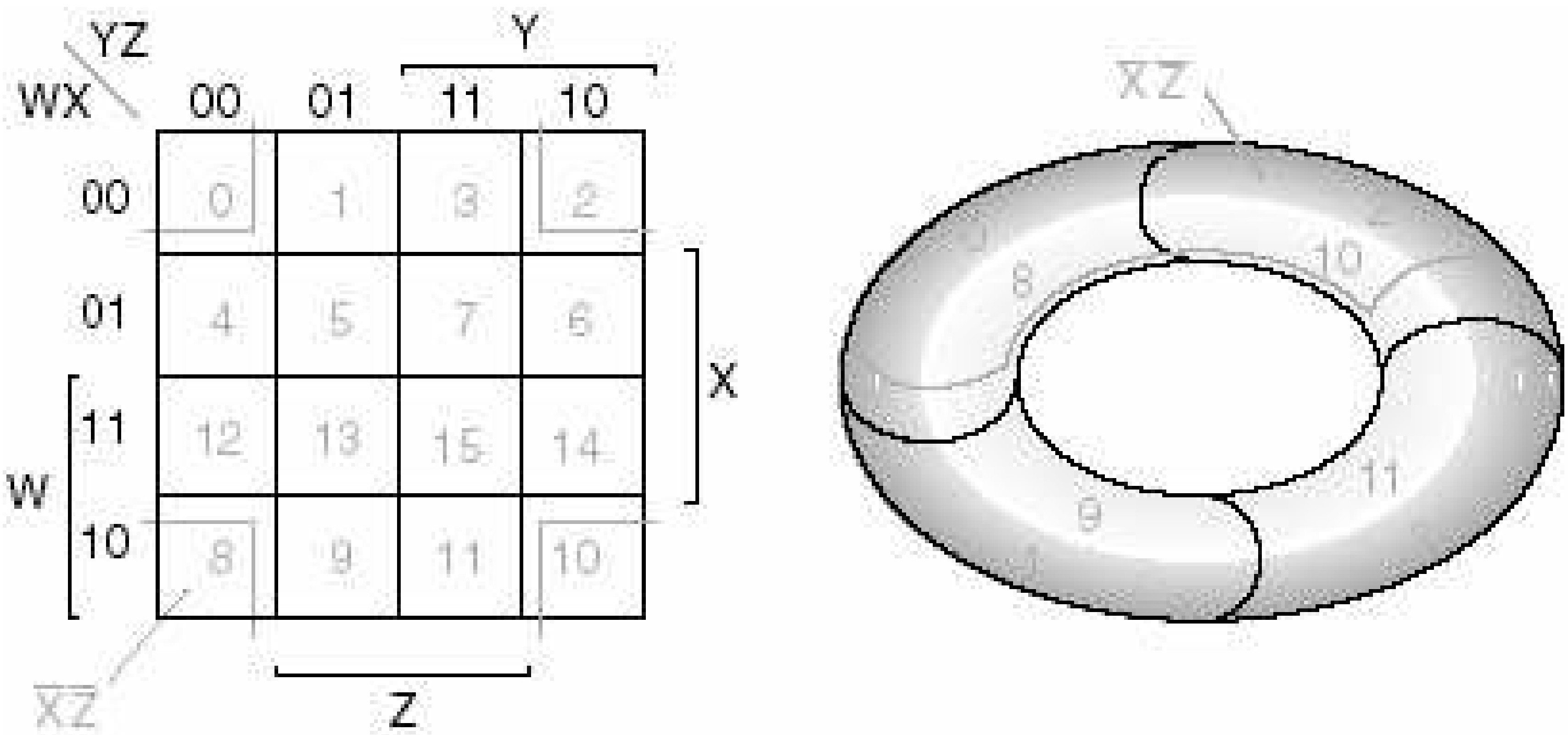
Solutions

- $F3 = X'Z + XY' + XZ' = Y'Z + XZ' + X'YZ$
- $F4 = Z + X'Y$

A KM for 4-variable functions

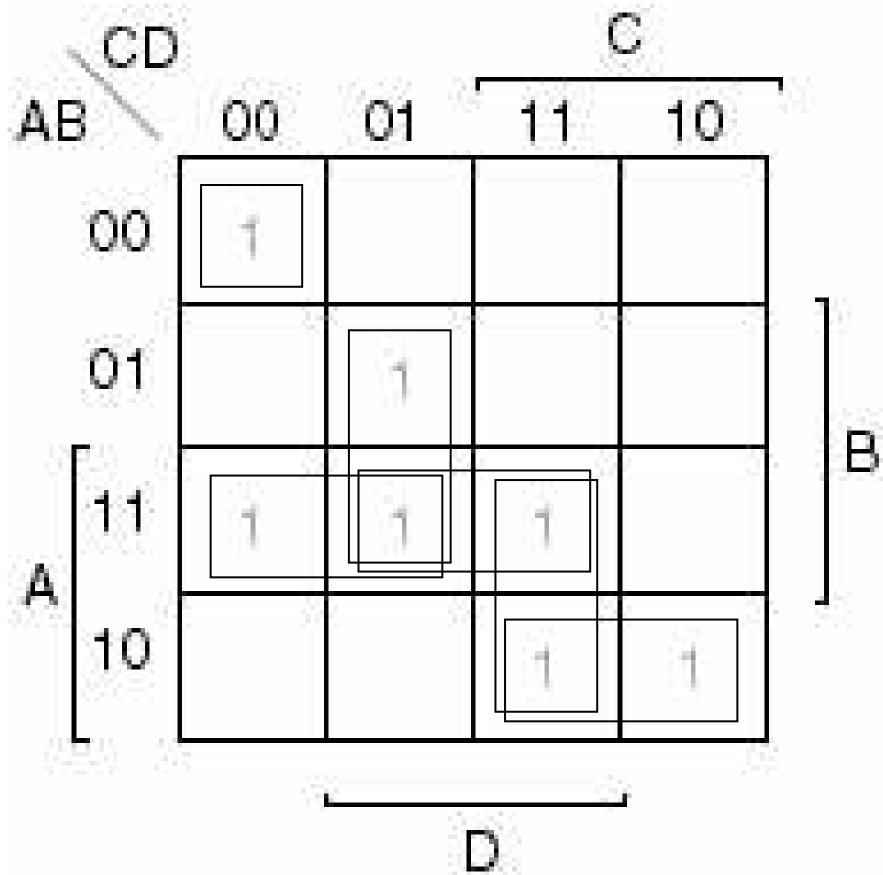


Circular adjacencies for 4 variables

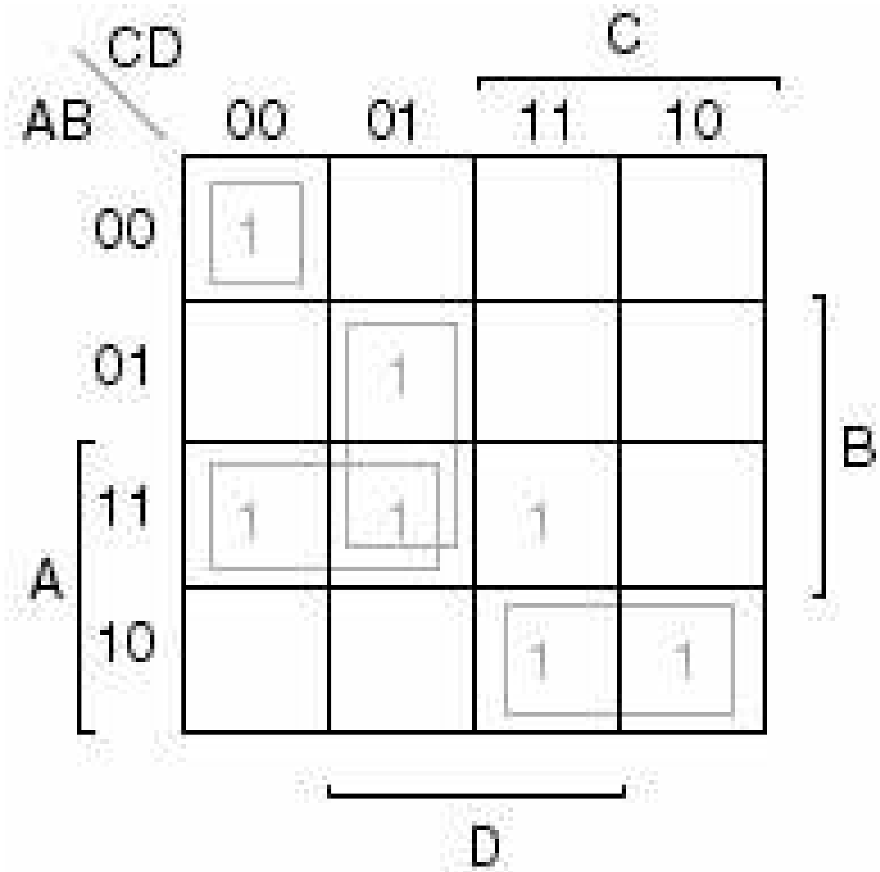


A simple example (1)

- All prime implicants are shown
- Find the essential ones

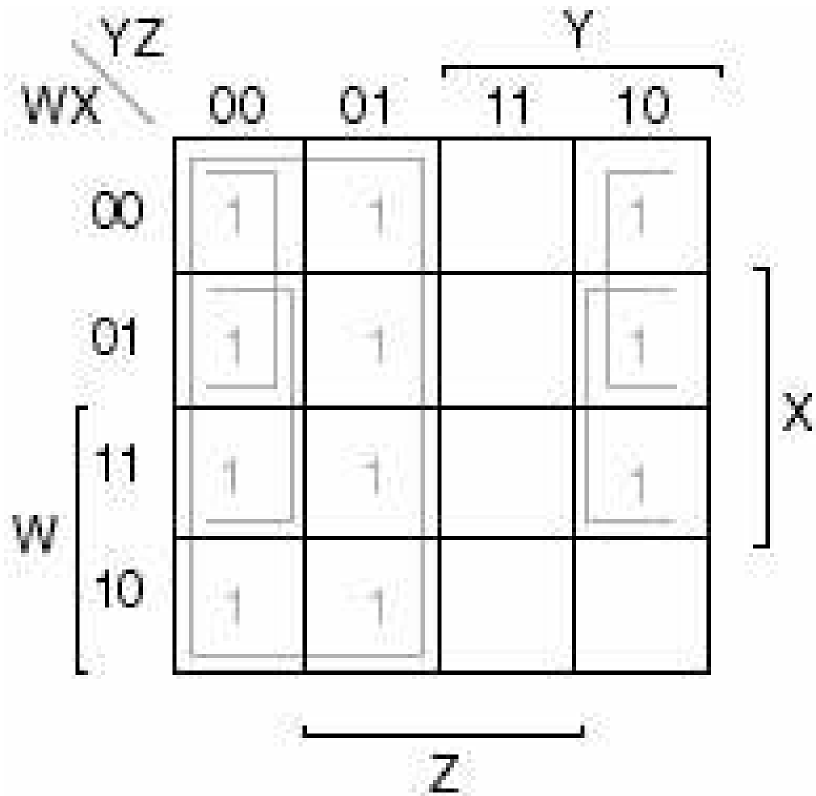


A simple example (2)



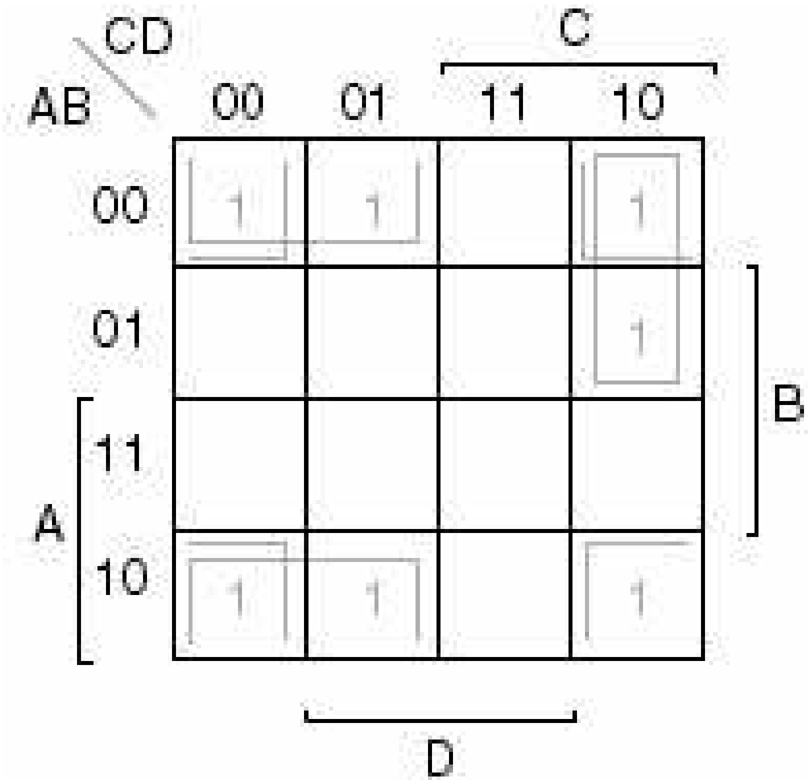
More exercise (1)

- Which is the function G1 for this KM?



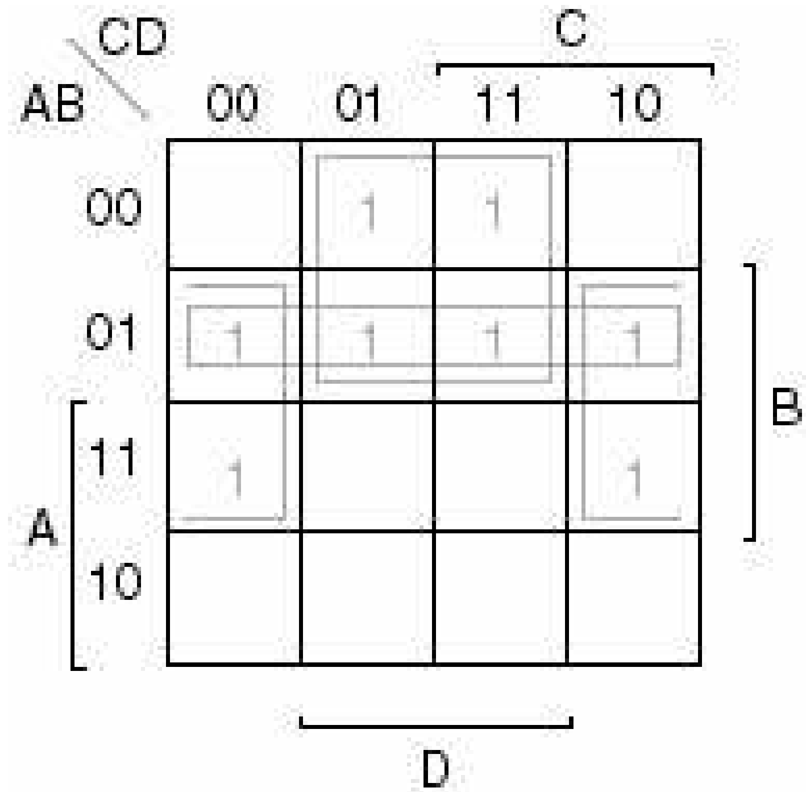
More exercise (2)

- Which is the function G2 for this KM?



More exercise (3)

- Which is the function G3 for this KM?

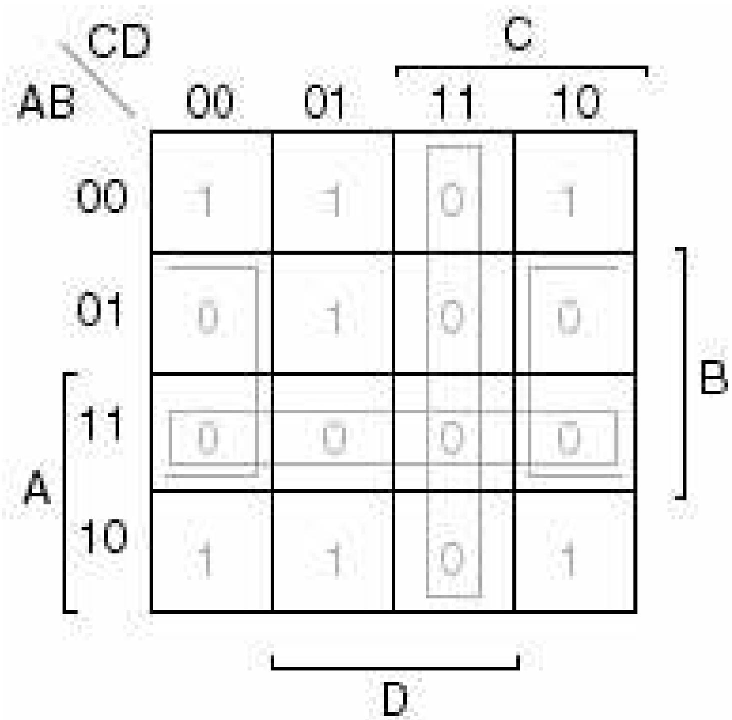


Answers

- $G1 = Y' + W'Z' + XZ'$
- $G2 = B'C' + B'D' + A'CD'$
- $G3 = A'D + A'B + BD'$

The KM method for POS

- Which is the POS expression of function F represented by this KM?



Solution

- Use the same method used for build POS canonical form from truth tables
- F = (CD)′ . (BD′)′ . (AB)′

  = (C′+D′) . (B′+D″) . (A′ + B′)

  = (C′+D′) . (B′+D) . (A′ + B′)

Don't care values

- A *don't care* value in a truth table means indifference for the actual value of the function
  - Truth table on the left may be substitued by anyone on the right

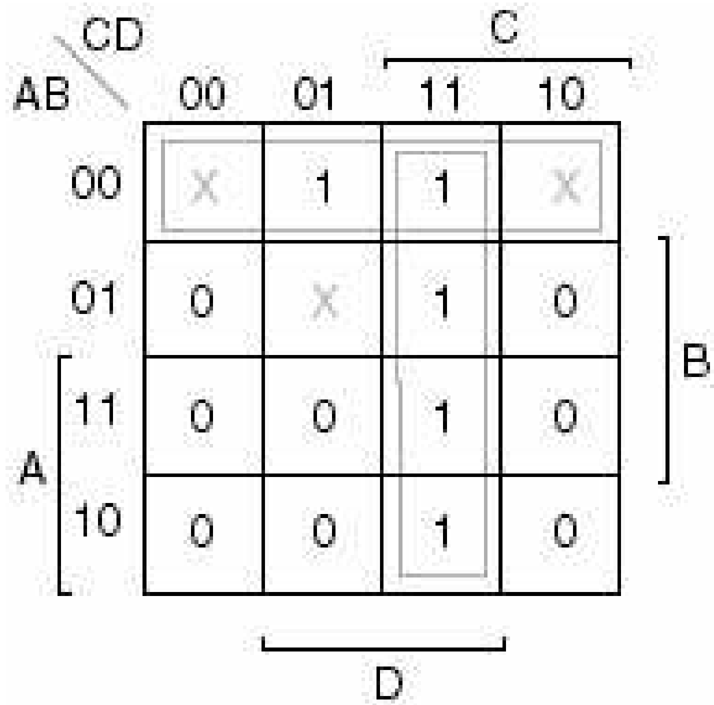
A	B	F
0	0	0
0	1	1
1	0	1
1	1	dc

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Don't care values in KMs

- Simplify the choice, since each of them (X) can be considered a 0 or a 1



An alternative choice

- Previous choice gives F = CD + A'B' but a different choice is possible F = CD + A'D