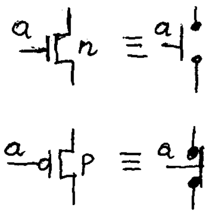
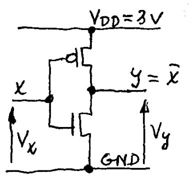
3 Logic Gates and Boolean Algebra

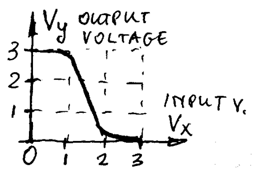
3.1 CMOS Technology

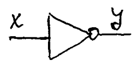
- Digital devises are predominantly manufactured in the Complementary-Metal-Oxide-Semiconductor (CMOS) technology.
- Two types of switches, as discussed in sec. 1.1, are implemented as a pair of complementary MOS Fields-Effect-Transitors FETs:
  - nMOS transistor — "normally open" switch,
  - pMOS transistor — "normally closed" switch,

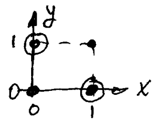


- Such pair of transistors is used to build an **inverter** and consequently all digital devices
- When the input voltage V_x is **low**, that is, the logic signal $x = 0$, the pMOS transistor is closed and nMOS transistor is open. Consequently the output voltage V_y is **high**, that is, the logic signal $y = 1$.
- The inverse situation occurs when the input voltage is **high**.









3.2 Boolean Algebra and Logic Gates

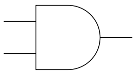
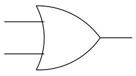
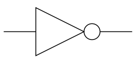
Operations performed by logic gates can be conveniently described in Boolean algebra.
The (two-valued) Boolean algebra is defined on

- a set of two elements, $\mathcal{B} = \{0, 1\}$,
- two binary operators, OR (+) and AND ($\cdot$),
- one unary operator, NOT ($'$), ($\bar{\ }$)

Two Boolean values 0 and 1 correspond to

- two values, "false" and "true" used in mathematical logic, and to
- two voltage levels, "LOW" and "HIGH" used in switching circuits.

Three basic Boolean (logic) operations

name:	AND	OR	NOT
symbol:			
operation:	$y = x_1 \cdot x_0$	$y = x_1 + x_0$	$y = x_1'$

x_1	x_0	y	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

y
1
0

A 2-variable **truth table** lists values of the output signal $y \in \{0, 1\}$ (results of logic operations) for all possible combinations of input signals, $x_1, x_0 \in \{0, 1\}$ (operands).

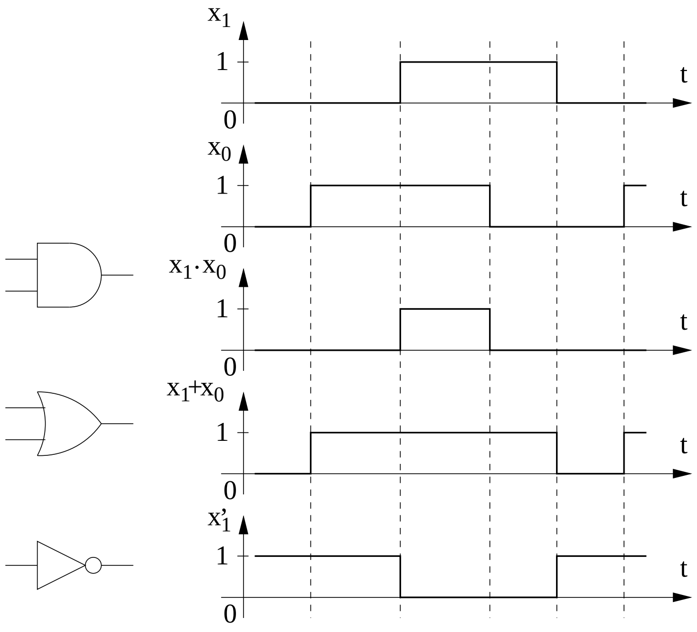
- The result of the **AND operation** is 1 if and only if both operands are 1.

  We also say that the output of the AND gate is HIGH (asserted) if both input signals are HIGH (asserted).
- The AND operation or **logic multiplication** is identical with arithmetic multiplication.
- The result of the **OR operation** is 1 if at least one operand is 1.

  We also say that the output of the OR gate is HIGH (asserted) if at least one input signal is HIGH (asserted).
- The OR operation or **logic addition** differs from arithmetic addition, because $1 + 1 = 1$ not 2!
- The **NOT operator** or logic complement can be arithmetically interpreted by the following expression: $x' = 1 - x$
- The NOT gate or INVERTER complements input signals: $0' = 1,\ \ 1' = 0$.

3.3 Timing diagrams

Operations performed by logic gates can also be described by means of timing diagrams.



In the example the pair of input signals x_1, x_0 goes through all possible combinations in the following way:

$$x = (x_1x_0)_2 \in \{0, 1, 3, 2\}$$

Such a code is known as the Gray code

3.4 Boolean Expressions and Logic Diagrams

Boolean expressions are formed from:

- two Boolean constants, $(0, 1)$,
- three basic logic operations, $(\cdot, +, ')$, and
- parentheses ( ).

The order of evaluation is:

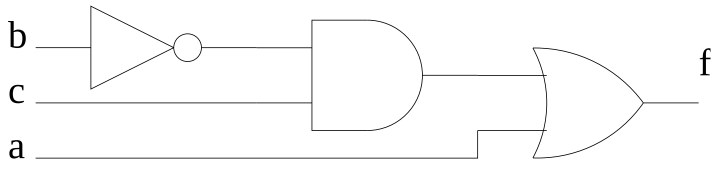
- expressions inside parentheses,
- complement (NOT),
- logic multiplication (AND),
- Logic addition (OR).

Consider a Boolean (logic) expression

$$f = a + b' \cdot c = a + \bar{b} \cdot c$$

where a, b, c, f and Boolean variables.

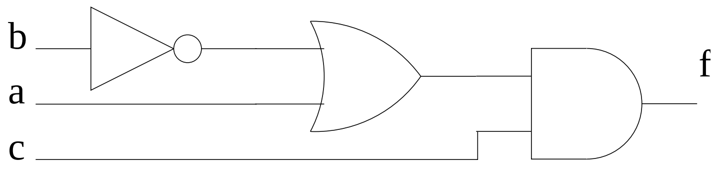
The equivalent logic diagram:



In order for the logic addition to be performed before multiplication we have to add parentheses:

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

The equivalent logic diagram:



The AND operator (the multiplication sign) may be omitted and we can write

$$f = a + b'c \text{ or } f = (a + b')c$$

Identify:

- input-output ports, or signals (wires) connected to the outside world
- gates and
- signal or nets (wires), that is. signals interconnecting gates inputs and outputs.

3.5 VHDL Hardware Description Language — example 1

- Digital devices/circuits can be described/modelled using a hrdware description language, VHDL.
- The description consiste of two main parts:
- **Input-output ports** are specified by the **ENTITY**
- The **circuit structure or function** is specified by an **ARCHITECTURE**

Example:

- The contents of the **log_circ_1.vhd** file:

```
 -- this is a comment
 -- VHDL is NOT case sensitive
 -- To emphasize, the key words are capitalized
 -- this is a black-box a logic circuit

ENTITY  log_circ_1  IS
   PORT (a, b, c : IN  BIT ;
         f       : out BIT ) ;
END [ENTITY]  log_circ_1 ;

-- an architecture for log_circ_1

ARCHITECTURE arch_1  OF log_circ_1 IS

BEGIN
  f <= ( a OR NOT b) AND c ; -- a SIGNAL assignment
END [ARCHITECTURE] arch_1 ;
```

3.6 Truth tables and Karnaugh Maps

The behaviour of a logic circuit, that is, the values of the output signals for all combinations of input signals can be equivalently described by:

- a Boolean expression,
- the truth table,
- the Karnaugh map,

Consider the following Boolean (logic Function

$$f = (a' + b)c$$

The **truth table** lists the values of the function f for all $2^3 = 8$ combinations of three input variables

$(cba)_2$	c	b	a	a'	$a' + b$	f
0	0	0	0	1	1	0
1	0	0	1	0	0	0
2	0	1	0	1	1	0
3	0	1	1	0	1	0
4	1	0	0	1	1	1
5	1	0	1	0	0	0
6	1	1	0	1	1	1
7	1	1	1	0	1	1

3.7 A 3-variable Karnaugh map

- Karnaugh maps are representations of Boolean hyper-cubes.

  A concept of adjacent vertices

  A Karnaugh map for

$$f = (a' + b)c$$

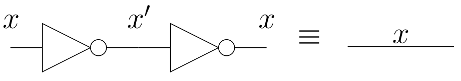
$c \backslash b\,a$	0 0	0 1	1 1	1 0
0	0	0	0	0
1	1	0	1	1

3.8 Theorems of Boolean Algebra and their circuit interpretation

Transformations and simplification of logic circuits are based on a variety of Boolean algebra theorems which can easily be verified by

- Double Complement — Involution property

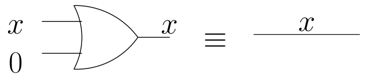
$$x'' = x$$



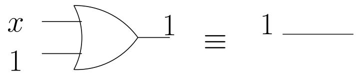
  Double NOT operation can be removed.

- Operations with constants

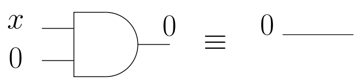
$x + 0 = x$



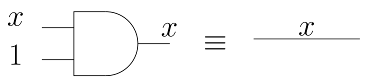
$x + 1 = 1$



$x \cdot 0 = 0$

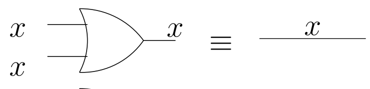


$x \cdot 1 = 1$

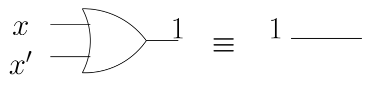


- Operations with repeated arguments

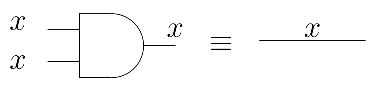
$x + x = x$



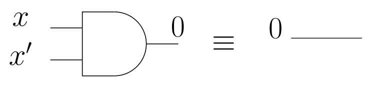
$x + x' = 1$



$x \cdot x = x$



$x \cdot x' = 0$



- OR and AND are commutative operations — all gate inputs are identical:

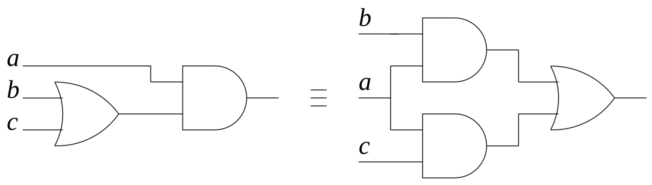
$$\text{OR: } a + b = b + a \quad \text{AND: } a \cdot b = b \cdot a$$

- OR and AND are associative operations — n-input gates exist:

$$\text{OR: } a + (b + c) = (a + b) + c = a + b + c$$

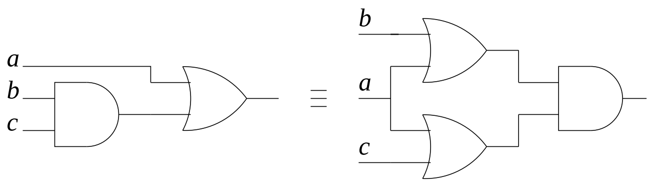
- OR operation (logic addition) is distributive

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



- AND operation (logic multiplication) is **also** distributive!

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



Verification of the distributive law for the AND operation using the truth table method:

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

$(cba)_2$	c	b	a	$b \cdot c$	LHS	$a + b$	$a + c$	RHS
0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	1	1
2	0	1	0	0	0	1	0	0
3	0	1	1	0	1	1	1	1
4	1	0	0	0	0	0	1	0
5	1	0	1	0	1	1	1	1
6	1	1	0	1	1	1	1	1
7	1	1	1	1	1	1	1	1

For all combinations of variables LHS = RHS, therefore, the distributive law for the logic multiplication is valid.

Duality Principle

Every theorem of the Boolean algebra remains valid if the **operators** and **constants** are interchanged, that is:

$$\begin{array}{rcl} \text{AND} & \Longleftrightarrow & \text{OR} \\ 1 & \Longleftrightarrow & 0 \end{array}$$

Example:

If the following equality

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

is valid, then interchanging '+' with '$\cdot$' we obtain the **dual** equality:

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

which is also valid.

Absorption Rules

1. $a + a \cdot b = a$

   Verification by algebraic manipulation:

   $$a + a \cdot b = a \cdot (1 + b) = a \text{ , because } 1 + b = 1$$

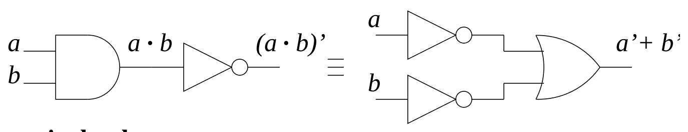
2. $a \cdot (a + b) = a$ — the dual equality
3. $a + a' \cdot b = a + b$ — Important!
4. $a \cdot (a' + b) = a \cdot b$

Absorption rules are important in circuit simplification.

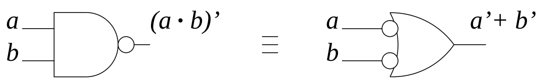
De Morgan's Theorems

1. The complement of a product (AND) is equal to the sum (OR) of the complements:

$$(a \cdot b)' = a' + b'$$



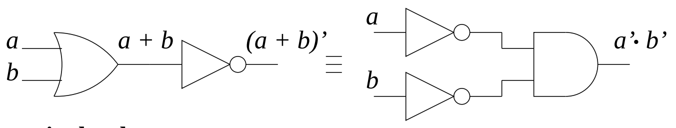
equivalently



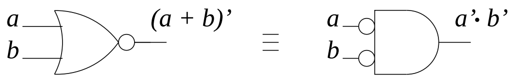
This is the NAND gate (NOT AND)

2. The complement of a sum (OR) is equal to the product (AND) of the complements (the dual theorem):

$$(a + b)' = a' \cdot b'$$



equivalently



This is the NOR gate (NOT OR)

3.9 All two-variable functions $y = F(b, a)$

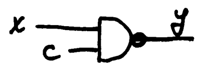
b	0	0	1	1		
a	0	1	0	1		
F_0	0	0	0	0	0	
F_1	0	0	0	1	$b \cdot a$	AND
F_2	0	0	1	0	$b \cdot \bar{a}$	
F_3	0	0	1	1	b	
F_4	0	1	0	0	$\bar{b} \cdot a$	
F_5	0	1	0	1	a	
F_6	0	1	1	0	$\bar{b} \cdot a + b \cdot \bar{a} = b \oplus a$	XOR
F_7	0	1	1	1	$b + a$	OR
F_8	1	0	0	0	$(\overline{b+a}) = \bar{b} \cdot \bar{a}$	NOR
F_9	1	0	0	1	$\bar{b} \cdot \bar{a} + b \cdot a = (\overline{b \oplus a})$	EQ XNOR
F_{10}	1	0	1	0	$\bar{a}$	
F_{11}	1	0	1	1	$b + \bar{a}$	
F_{12}	1	1	0	0	$\bar{b}$	
F_{13}	1	1	0	1	$\bar{b} + a$	
F_{14}	1	1	1	0	$\overline{b \cdot a} = \bar{b} + \bar{a}$	NAND
F_{15}	1	1	1	1	1	

3.10 NAND and NOR gates

the truth tables for the NAND and NOR gates

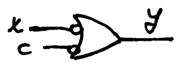
$x_1 x_0$	$\overline{x_1 x_2}$ $=\bar{x}_1+\bar{x}_2$	$\overline{x_1+x_2}$ $=\bar{x}_1\cdot\bar{x}_2$
00	1	1
01	1	0
10	1	0
11	0	0
	NAND	NOR

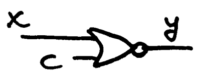
Alternative, functional description of NAND and NOR gates



NAND

	y
0	1
1	$\bar{x}$





NOR

	y
0	$\bar{x}$
1	0