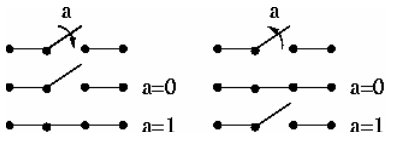
03 Logic networks

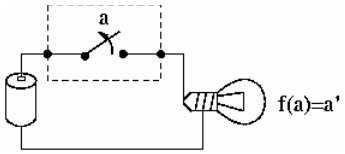
03.02 Switching networks and combinational circuits

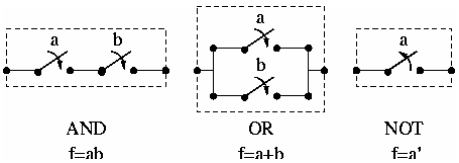
- Switching networks
- Logic gates
- Boolean networks and logic families
- Transistors
- CMOS gates
- Combinational circuits
- Examples

Switching networks

- Switches and bulbs are 2-state components of electrical circuits. Boolean variables may be associated with their states (ON and OFF).
- Switches may be used to control a bulb. The state of the bulb is a Boolean function of the states of the switches.
- Switch networks can implement Boolean operators
- Switch networks may implement any Boolean function







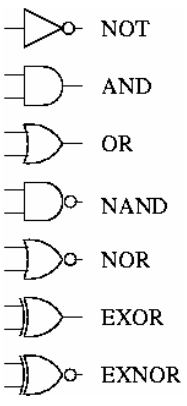
Logic gates

- Logic gates are elementary building blocks of digital circuits
- A logic gate is a component that takes in input one or more logic signals and provides one output signal whose logic value is a function of the configuration of the input signals

- Logic gates are implemented as switch networks

Logic gates

- Logic gates implement elementary Boolean operations:



$f = a' = \bar{a}$

$f = ab$

$f = a + b$

$f = (ab)'$

$f = (a + b)'$

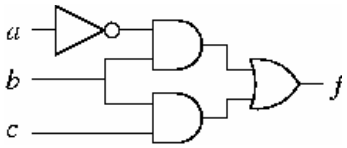
$f = a \oplus b = ab' + a'b$

$f = (a \oplus b)' = ab + a'b'$

Boolean networks

- Logic gates can be composed to form a Boolean network as the corresponding operations can be composed in a Boolean expression
- There is a 1-1 correspondence between Boolean networks and Boolean expressions
- The output of gate *g*1 is connected to the input of gate *g*2 if and only if the result of operation *o*1 is an operand of operation *o*2.

$$f = a'b + bc$$

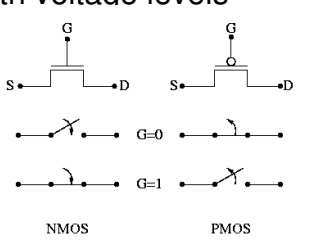


Logic families

- *Logic library:* set of logic gates to be used as building blocks for implementing Boolean networks
- *Functional completeness*: capability of a logic library to implement any Boolean function
- Examples:
  - {AND, OR, NOT}
    Functional completeness demonstrated by SoP canonical forms
  - {NAND}
    Functional completeness demonstrated by showing that and, or and not can be expressed in terms of NANDs
    a' = (*aa*)', *ab* = ((*ab*)'(*ab*)')', *a*+*b* = ((*aa*)'(*bb*)')'
  - {NOR}

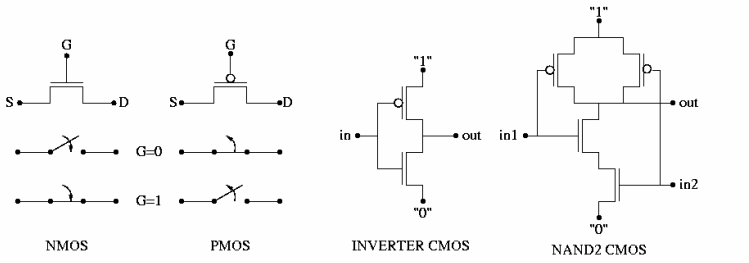
Transistors

- A transistor is a 3-terminal component realizing an electronic switch that uses terminal G (*gate*) to control the connection between terminal S (*source*) and D (*drain*)
- Logic values are associated with voltage levels (0=low, 1=high)
- nMOS and pMOS transistors form complementary switches



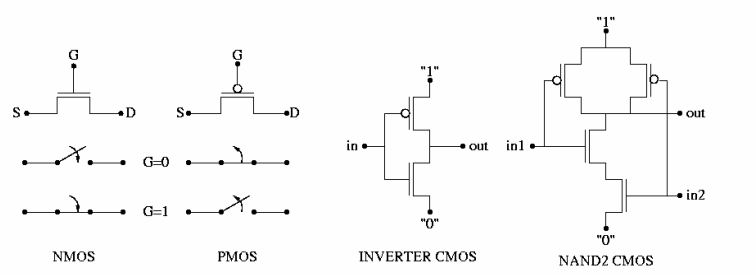
CMOS gates

- Complementary MOS gates
- Use complementary networks of nMOS and pMOS transistors to connect the output node either to the "0" or to the "1" voltage source
- Switch networks connecting to 0 and 1 are called *pull-down* and *pull-up*, respectively



CMOS gates (2)

- Elementary CMOS gates form inverting functions
- A *n*-input CMOS gate is made of 2*n* transistors
- Thanks to De Morgan's law, pull-up and pull-down networks are never simultaneously active



Combinational circuits

- A combinational circuit is a circuit that implements a Boolean function
- The logic value assigned to the output signals is a Boolean function of the current configuration of input signals
- A combinational circuit that implements a Boolean function is called an *implementation* of that Boolean function
- There are infinite possible implementations of the same Boolean function

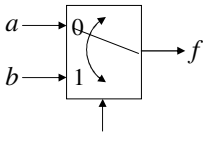
Implementation

1. Starting from a functional specification
2. Find a formal representation of the target Boolean function (e.g., truth table)
3. Find a Boolean expression that represents the Boolean function
4. Possibly optimize the Boolean expression by means of Boolean manipulation, according to given optimization criteria (e.g., minimum number of literals)
5. Realize the Boolean network associated with the optimal Boolean expression

Example: MUX

- Functional specification: take the output value *f* from either input *a* or *b*, depending on the value of control signal *c*

c=0 ➔ *f*=*a*, *c*=1 ➔ *f*=*b*



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

$$f = ab'c' + abc' + a'bc + abc$$
$$= ac'(b'+b) + bc(a'+a)$$
$$= ac' + bc$$

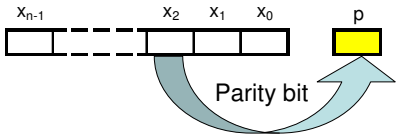
$$f = ((ac')')'+((bc)')' = ((ac')'(bc)')'$$

NAND implementation

Example: Parity bit

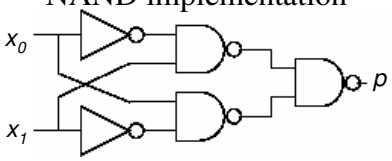
- Functional specification: extra bit to added to a word in order to obtain an even number of 1's

x_1	x_0	p
0	0	0
0	1	1
1	0	1
1	1	0



$$f = x_1'x_0 + x_1x_0' = ((x_1'x_0)'(x_1x_0')')' = x_1 \oplus x_0$$

NAND implementation



EXOR

Example: Parity bit (2)

x_2	x_1	x_0	p
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$f = x_2'x_1'x_0 + x_2'x_1x_0' + x_2x_1'x_0' + x_2x_1x_0 =$$
$$= x_2'(x_1'x_0 + x_1x_0') + x_2(x_1'x_0' + x_1x_0) =$$
$$= x_2'(x_1 \oplus x_0) + x_2(x_1 \oplus x_0)' = x_2 \oplus (x_1 \oplus x_0)$$