Boolean Algebra

Logic and Digital System Design - CS 303

Erkay Savaş
Sabanci University

Boolean Algebra

- Algebra: A deductive mathematical system
  1. set of elements
  2. set of operators
  3. Number of unproved axioms (postulates)
- Set of elements
  - a set with countable number of elements
  - $S = \{a, b, c, d\}$
  - $x \in S$, and $y \notin S$
- Operators
  - binary operators: x and $y \in S \rightarrow z \in S$
  - unary operators: $x \in S \rightarrow z \in S$

Axioms of Algebra

1. Closure:
   - A set is closed w.r.t. a binary operation *
   - Example: natural numbers are closed w.r.t. (+)
2. Associative law:
   - $(x * y) * z = x * (y * z)$ for all $x, y, z \in S$
3. Commutative law:
   - $x * y = y * x$ for all $x, y \in S$
4. Identity element:
   - S is said to have a identity element e if $\exists\ e \in S$ such that $e * x = x * e = x$ for any $x \in S$
   - Set of integers: e = 1 w.r.t. $\times$ and e = 0 w.r.t. +

Axioms of Algebra (cont.)

5. Inverse
   - S having an identity element e w.r.t. * is said to have an inverse for every $x \in S$, whenever there exists an element $y \in S$ such that
     $x * y = e$
   - Example: set of integers
6. Distributive law
   - If * and • are two binary operators on S, * is said to be distributed over • whenever
   - $x * (y \bullet z) = (x * y) \bullet (x * z)$

An Example of Algebraic Structure

- S = {0, 1, 2} with $\times$ and + forms a algebraic structure

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

  - additive identity 0
  - multiplicative identity 1
  - we have additive and multiplicative inverses
  - $\times$ is distributed over + (not vice versa)

Boolean Algebra - 1

- A set of elements B
  - There exist at least two elements $x, y \in B$ s. t. $x \neq y$
- Binary operators: + and ·
  - closure w.r.t both + and ·
  - additive identity 0
  - multiplicative identity 1
  - commutative w.r.t. both + and ·
  - Associative w.r.t. both + and ·
- Distributive law:
  - · is distributive over +
  - + is distributive over ·
  - We do not have both in ordinary algebra

Boolean Algebra - 2

- Complement
  - For every element $x \in B$, there exist an element $x' \in B$ s. t.
    a. $x + x' = 1$ and
    b. $x \cdot x' = 0$.
  - Not available in ordinary algebra
- Differences btw ordinary and Boolean algebra
  - Ordinary algebra with real numbers
  - Boolean algebra with elements of set B
  - Complement
  - Distributive law
  - Do not substitute laws from one to another where they are not applicable

Two-Valued Boolean Algebra - 1

- To define a Boolean algebra
  - The elements of B
  - Rules for two binary operations
  - The elements of B and rules should conform to our axioms
- Two-valued Boolean algebra
  - B = {0, 1}

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

X	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

x	x′
0	1
1	0

Two-Valued Boolean Algebra - 2

- Check the axioms
  - Two distinct elements, $0 \neq 1$
  - Closure, associative, commutative, identity elements
  - Complement
    - $x + x' = 1$ and $x \cdot x' = 0$
  - Distributive law

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

y·z	x+(y·z)
0	0
0	0
0	0
1	1
0	1
0	1
0	1
1	1

x + y	x + z	(x + y) · (x + z)
0	0	0
0	1	0
1	0	0
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

Two-Valued Boolean Algebra - 2

- Two-valued Boolean algebra is actually equivalent to the binary logic defined heuristically before
  - Operations:
    - · → AND
    - + → OR
    - Complement → NOT
- Binary logic is application of Boolean algebra to the gate-type circuits
  - Two-valued Boolean algebra is developed in a formal mathematical manner
  - This formalism is necessary to develop theorems and properties of Boolean algebra

Duality Principle

- An important principle
  - every algebraic expression deducible from the axioms of Boolean algebra remains valid if the operators and identity elements are interchanged
- Example:
  - $x + x = x$.
  - $x + x = (x + x) \cdot 1$ (identity element)
    $= (x + x)(x + x')$ (complement)
    $= x + xx'$ (+ over ·)
    $= x$ (complement)
  - duality principle
    $x + x = x \quad \Rightarrow \quad x \cdot x = x$

Duality Principle & Theorems

- Theorem a:
  - $x + 1 = 1$
  - $x + 1 = 1 \cdot (x + 1)$
    $= (x + x')(x + 1)$
    $= x + x' \cdot 1$
    $= x + x'$
    $= 1$
- Theorem b: (using duality)
  - $x \cdot 0 = 0$
- Absorption Theorem
  a. $x + xy = x$
  b. $x \cdot (x+y) = x$

Involution & DeMorgan's Theorems

- Involution Theorem:
  - $(x')' = x$
  - $x + x' = 1$ and $x \cdot x' = 0$
  - Complement of x' is $(x')'$
  - Complement is unique
- DeMorgan's Theorem:
  - $(x + y)' = x' \cdot y'$
  - $(x \cdot y)' = x' + y'$

Truth Tables for DeMorgan's Theorem

$-(x + y)' = x' \cdot y'$

$-(x \cdot y)' = x' + y'$

x	y	x+y	(x+y)'	x · y	(x · y)'
0	0	0	1	0	1
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	0	1	0

x'	y'	x' · y'	x' + y'
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Operator Precedence

1. Parentheses
2. NOT
3. AND
4. OR

- Example:
  - $(x + y)'$
  - $x' \cdot y'$
  - $x + x \cdot y'$

Boolean Functions

- Consists of
  - binary variables (normal or complement form)
  - the constants, 0 and 1
  - logic operation symbols, "+" and "·"
- Example:
  - $F_1(x, y, z) = x + y'z$
  - $F_2(x, y, z) = x'y'z + x'yz + xy'$

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Logic Circuit Diagram of F_1

$F_1(x, y, z) = x + y'z$

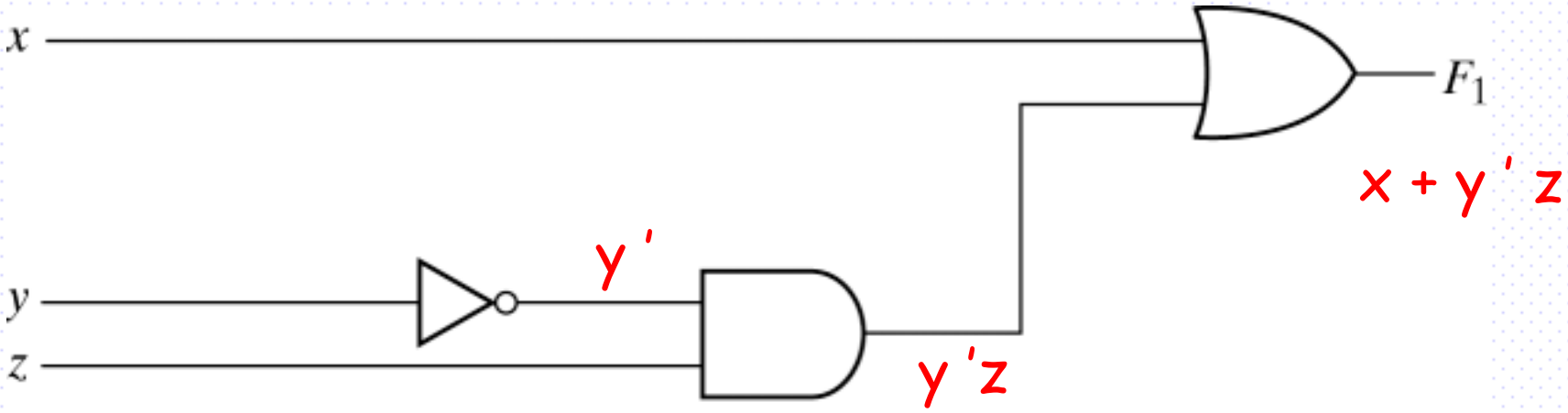
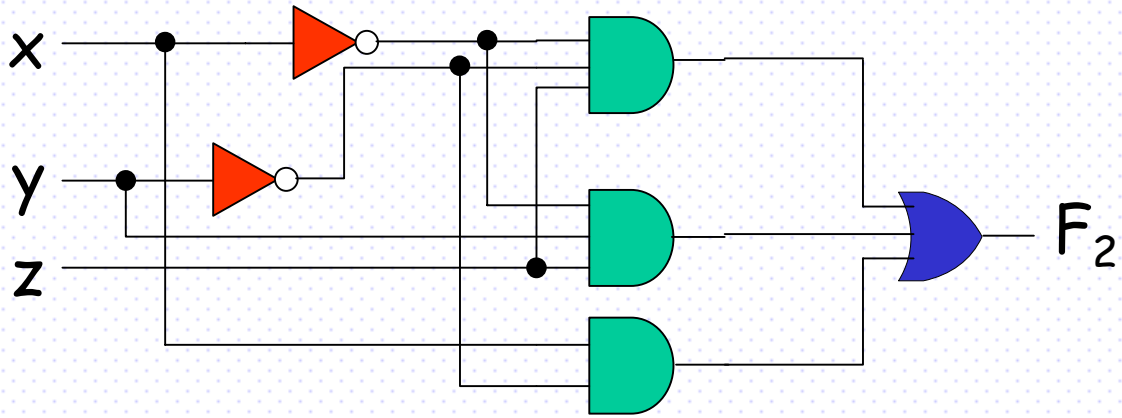


Fig. 2-1 Gate implementation of $F_1 = x + y'z$

Logic Circuit Diagram of F_2

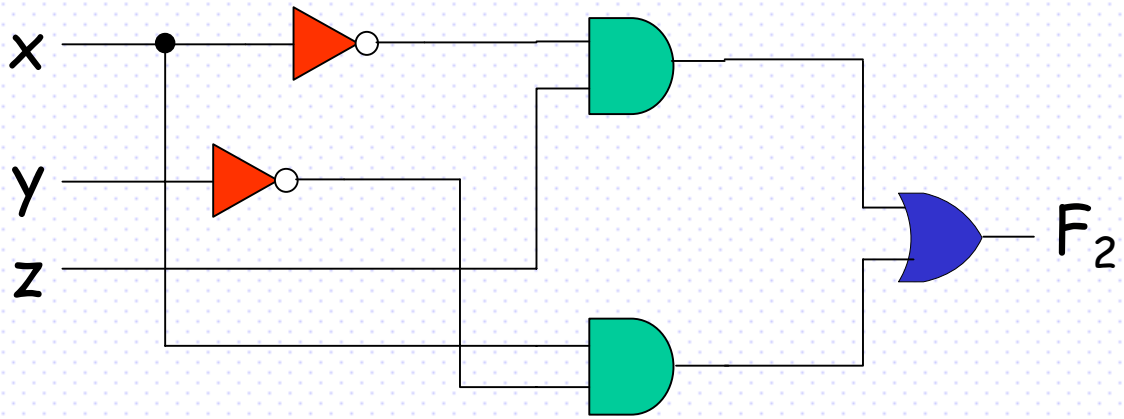
$F_2 = x' y' z + x' y z + x y'$



- Algebraic manipulation
- $F_2 = x' y' z + x' y z + x y'$
  $= x' z (y' + y) + x y'$
  $= x' z + x y'$

Alternative Implementation of F_2

$F_2 = x' z + x y'$



OTHER LOGIC OPERATORS - 1

- AND, OR, NOT are logic operators
  - defined by Boolean functions
  - three of the 16 possible Boolean functions

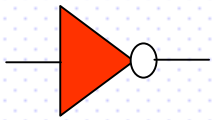
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

x	y	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

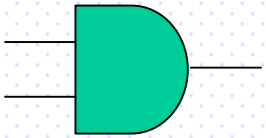
OTHER LOGIC OPERATORS - 2

- Some of the Boolean functions with two variables
  - Constant functions: $F_0 = 0$ and $F_{15} = 1$
  - AND function: $F_1 = xy$
  - OR function: $F_7 = x + y$
  - XOR function:
    - $F_6 = x' y + xy' = x \oplus y$ (x or y, but not both)
  - XNOR (Equivalence) function:
    - $F_9 = xy + x' y' = (x \oplus y)'$ (x equals y)
  - NOR function:
    - $F_8 = (x + y)' = (x \downarrow y)$ (Not-OR)
  - NAND function:
    - $F_{14} = (x y)' = (x \uparrow y)$ (Not-AND)

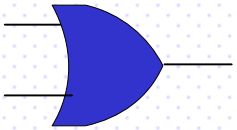
Logic Gate Symbols



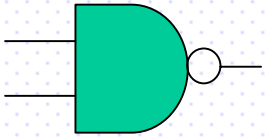
NOT



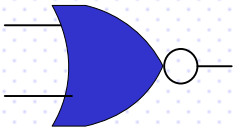
AND



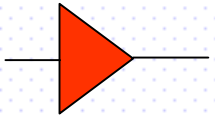
OR



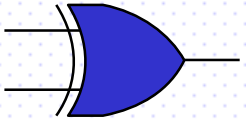
NAND



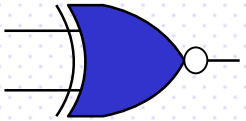
NOR



TRANSFER



XOR



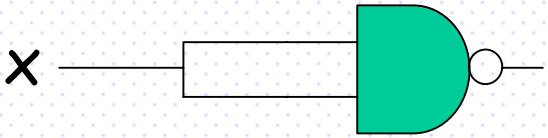
XNOR

Universal Gates

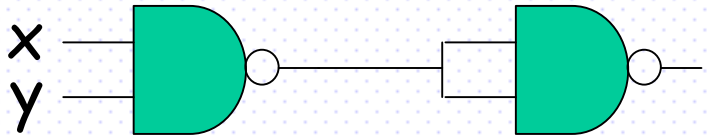
- NAND and NOR gates are universal
- We know any Boolean function can be written in terms of three logic operations:
  - AND, OR, NOT
- In return, NAND and NOR gate can implement these three logic gates

x	y	(xy)'	x'	y '	(x' y')'
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	1

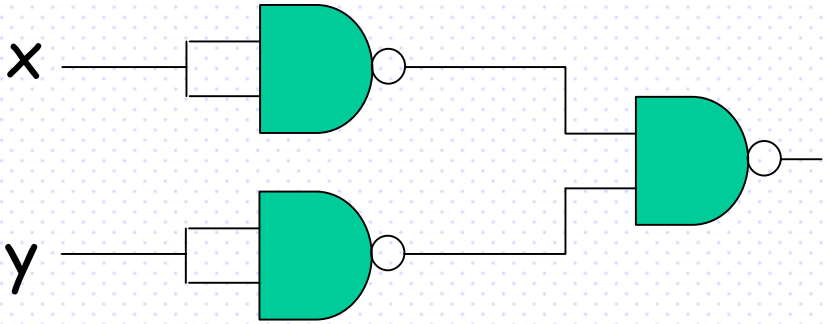
NAND Gate



$(x\ x)' = x'$ ➔ NOT

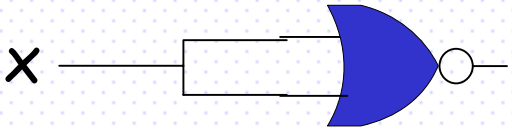


$[\ (x\ y)'\ ]' = x\ y$ ➔ AND

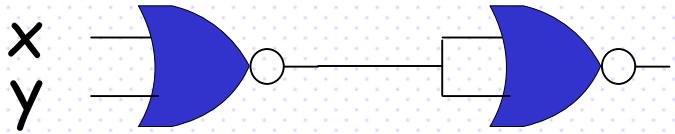


$(x'\ y'\ )' = x + y$ ➔ OR

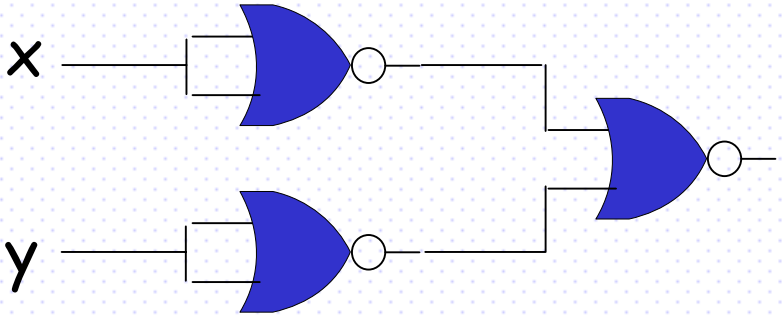
NOR Gate



$(x + x)' = x' \rightarrow$ NOT



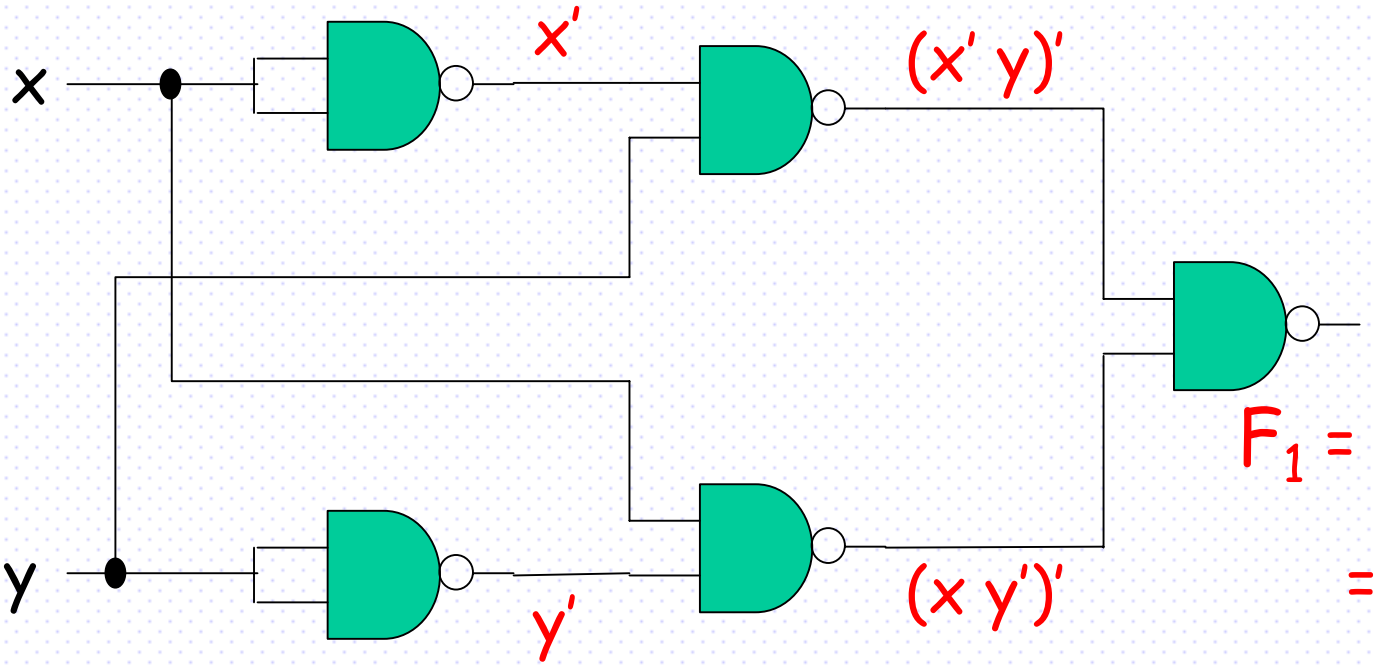
$[\,(x + y)'\,]' = x + y \rightarrow$ OR



$(x' + y')' = x \cdot y \rightarrow$ AND

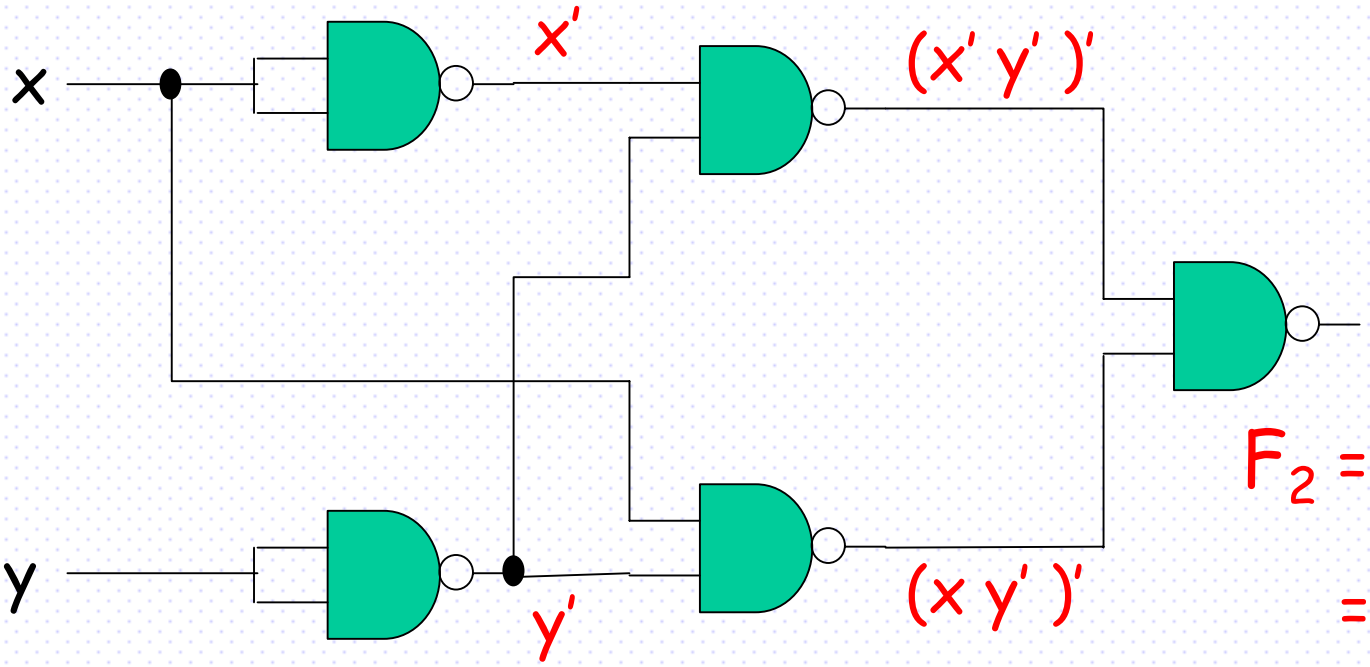
Example - 1

- Two functions:
  - $F_1 = x' y + xy'$ and $F_2 = x' y ' + xy'$



Example - 2

- $F_2 = x' y' + xy'$



$F_2 = [(x' y')' (x y')']'$

$= x' y' + x y'$

Multiple Input Gates

- AND and OR gates:
  - they are both commutative and associative
  - No problem with extending the number of inputs
- NAND and NOR gates
  - they are both commutative but not associative
  - Extending the number of inputs is not obvious
- Example:
  - $(x \uparrow y) \uparrow z \neq x \uparrow (y \uparrow z)$
  - $(x \uparrow y) \uparrow z = [\,(x\,y)'\,z\,]' = (x\,y) + z' = x\,y + z'$
  - $x \uparrow (y \uparrow z) = [\,x\,(y\,z)'\,]' = x' + y\,z$

Nonassociativity of NOR operation

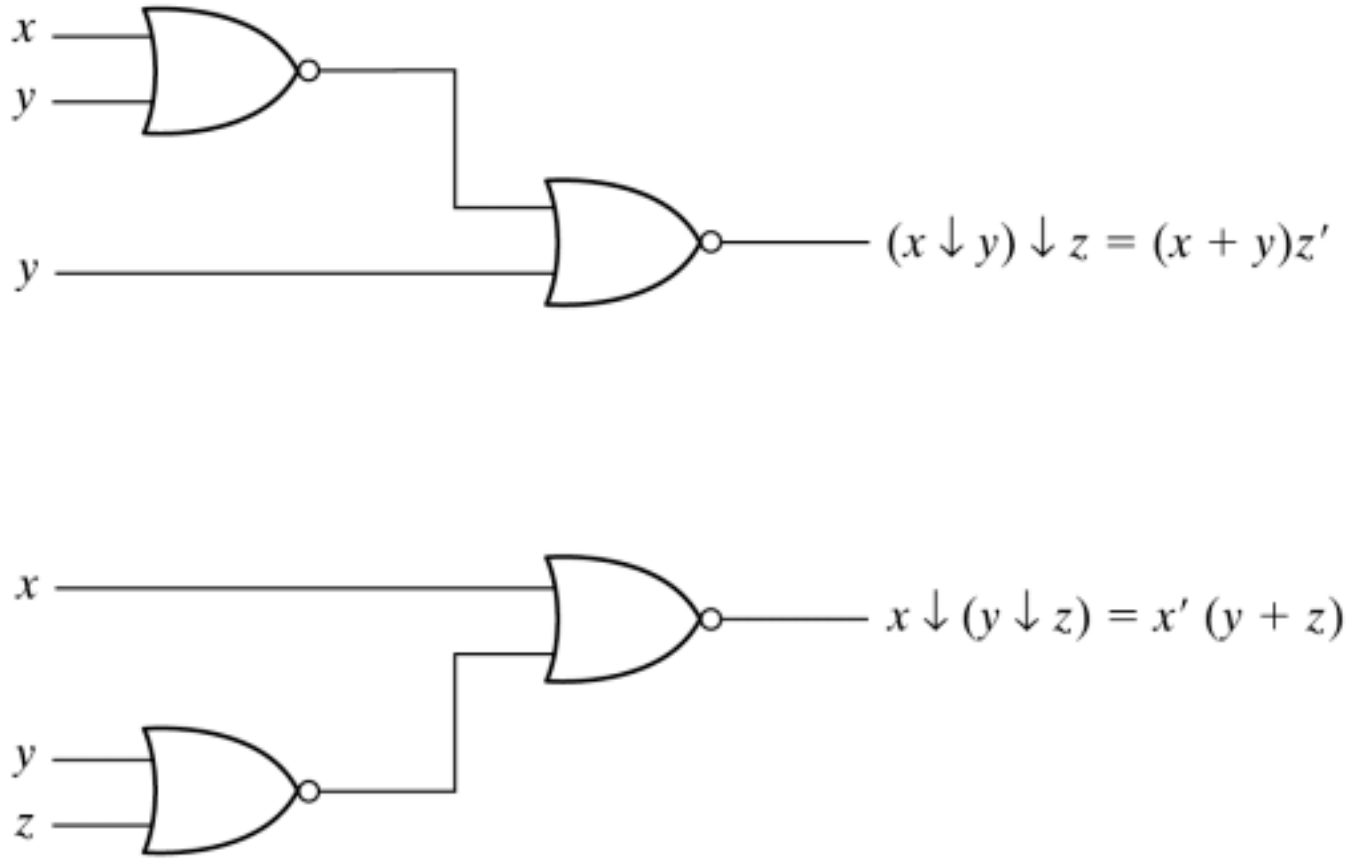
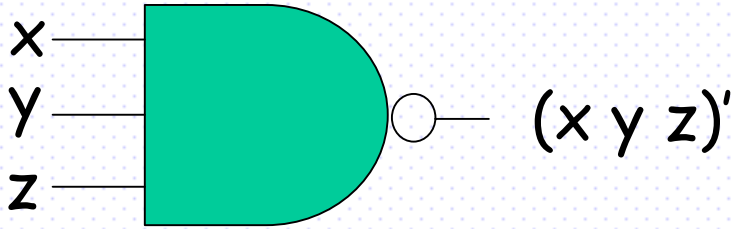


Fig. 2-6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

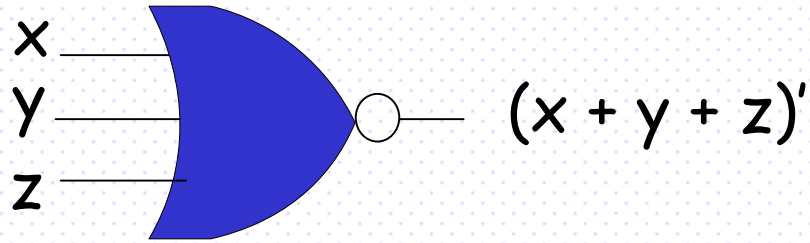
Multiple Input Universal Gates

- To overcome this difficulty, we define multiple-input NAND and NOR gates in slightly different manner

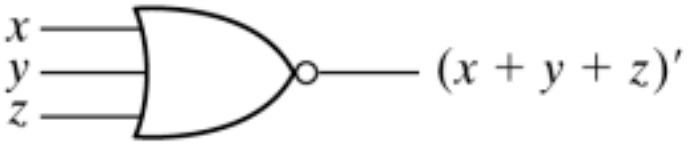
$x \uparrow y \uparrow z = (x\ y\ z)'$



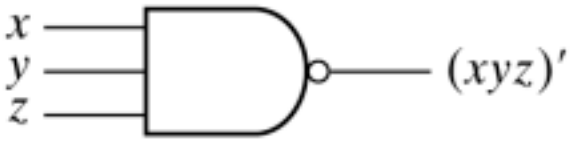
$x \downarrow y \downarrow z = (x + y + z)'$



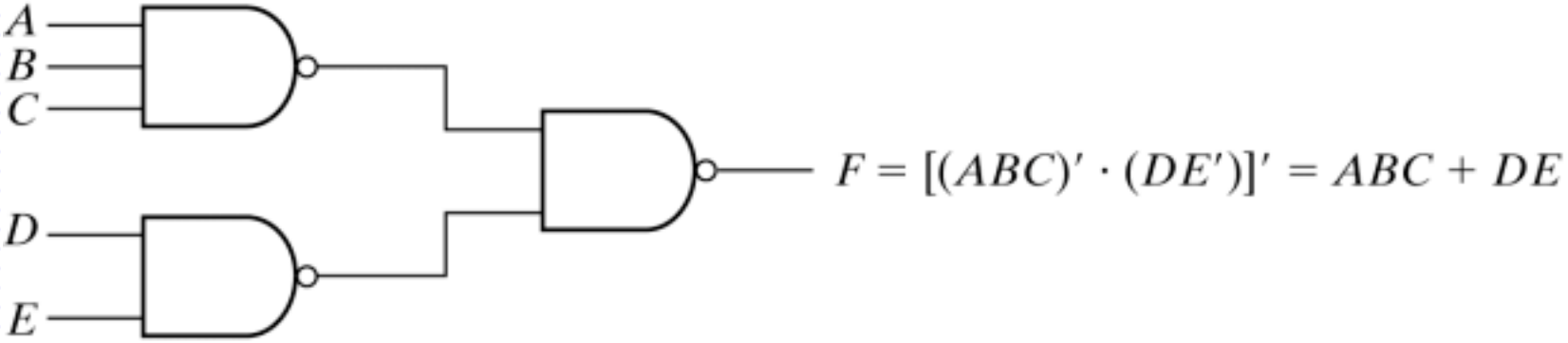
Multiple Input Universal Gates



(a) 3-input NOR gate



(b) 3-input NAND gate

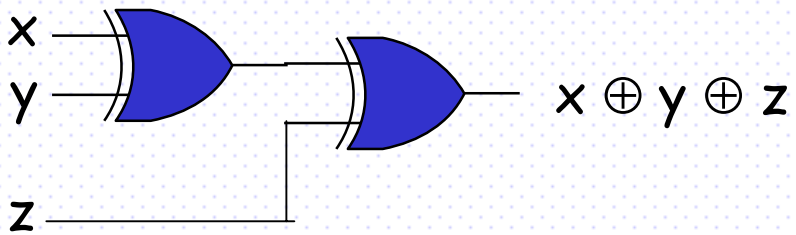


(c) Cascaded NAND gates

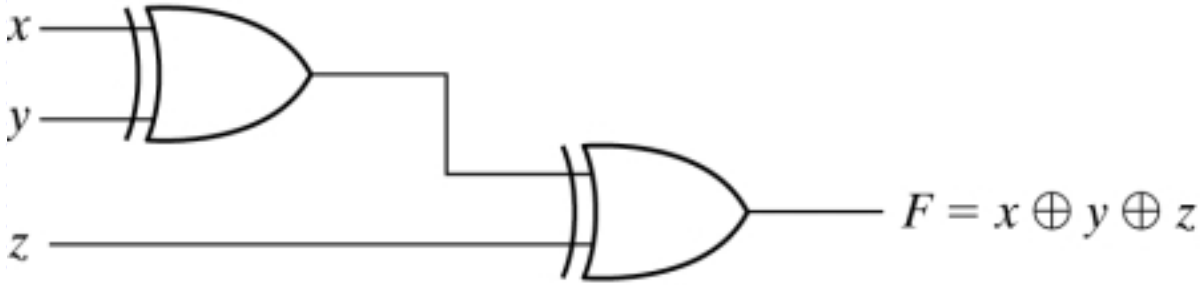
Fig. 2-7 Multiple-input and cascated NOR and NAND gates

XOR and XNOR Gates

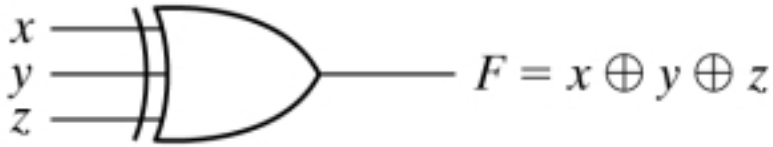
- XOR and XNOR operations are both commutative and associative.
- No problem manufacturing multiple input XOR and XNOR gates
- However, they are more costly from hardware point of view.
- Therefore, we usually have 2 input XOR and XNOR gates



3-input XOR Gates



(a) Using 2-input gates



(b) 3-input gate

x	y	z	F
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

(c) Truth table

Fig. 2-8 3-input exclusive-OR gate

Algebraic Manipulations

- Previous example:
  - $F(x,y,z) = x' y' z + x y z + x y'$
    $= x'z + xy'$
- The goal:
  - The reduce the number of terms
  - The reduce the number of literals in a term
  - to simplify the circuit
- Examples:
  - $x(x'+y) = xy$ and $x+x' y = (x+x')(x+y) = x+y$
  - $(x+y)(x+y') = xx + xy' + xy + yy'$
    $= x + xy' + xy$
    $= x + x (y+y')$
    $= x + x = x$

Consensus Theorem

- $xy + x'z + yz$

$$= xy + x'z + yz(x + x')$$
$$= xy + x'z + xyz + x'yz$$
$$= xy(1 + z) + x'z(1+y)$$
$$= xy + x'z$$

- From duality:

$$(x + y)(x' + z)(y + z) = (x + y)(x' + z)$$

Complement of a Function - 1

- F′ is complement of F
  - We can obtain F′, by interchange of 0's and 1's in the truth table
  - We can also utilize DeMorgan's Theorem
  - $(x+y)' = x'\,y'$
  - $(A+B+C)'$

$$= (A+x)'$$
$$= A'\,x'$$
$$= A'\,(B+C)' = A'B'C'$$

- We can generalize DeMorgan's Theorem

1. $(x_1 + x_2 + \ldots + x_N)' = x_1' \cdot x_2' \cdot \ldots \cdot x_N'$
2. $(x_1 \cdot x_2 \cdot \ldots \cdot x_N)' = x_1' + x_2' + \ldots + x_N'$

Complement of a Function - 2

- Example:
  - $F_1 = x'yz' + x'y'z$
  - $F_1' = (x'yz' + x'y'z)'$
    $= (x'yz')'(x'y'z)'$
    $= (x + y' + z)(x + y + z')$
  - $F_2 = x(y'z' + yz)$
  - $F_2' = (x(y'z' + yz))'$
    $= x' + (y'z' + yz)'$
    $= x' + (y'z')' (yz)'$
    $= x' + (y + z) (y' + z')$
- Important: Somewhat easier method is to take the dual of the function and complement each literal

Canonical & Standard Forms

- Minterms
  - A binary variable may appear in its normal (x) and its complement form (x′)
  - How many different terms we can get with x and y?
  - x′y′, x′y, xy′, and xy → 00, 01, 10, 11
  - m_0, m_1, m_2, m_3 (AND terms or minterms, standard product)
  - n variables can be combined to form 2^n minterms
- Maxterms (OR Terms, standard sums)
  - $M_0 = x + y$
  - $M_1 = x + y'$
  - $M_2 = x' + y$
  - $M_3 = x' + y'$

Min- & Maxterms with n = 3

			Minterms		Maxterms	
x	y	z	term	designation	term	designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Boolean Functions in Standard Form

x	y	z	F_1	F_2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- $F_1(x, y, z) = m_1 + m_4 + m_7 = \Sigma\,(1, 4, 7)$
- $F_2(x, y, z) = m_0 + m_3 + m_5 + m_6 + m_7 = \Sigma\,(0, 3, 5, 6, 7)$

Important Properties

- Any Boolean function can be expressed as a sum of minterms
- Example:
  - $F_1' = \Sigma\,(0, 2, 3, 5, 6)$
    $= x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - How do we find the complement of F_1'?
  - $F_1 = (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z)$
    $= M_0 + M_2 + M_3 + M_5 + M_6$
    $= \Pi\,(0, 2, 3, 5, 6)$
- Any Boolean function can be expressed as a product of maxterms

Canonical Form

- If a Boolean function is expressed as a sum of minterms or product of maxterms the function is said to be in canonical form.
- Example: $F = x + y'z$ → canonical form?
  - No
  - But we can put it in canonical form.
  - $F = x + y'z = \Sigma(7, 6, 5, 4, 1)$
- Alternative way:
  - Obtain the truth table first and then the canonical term.

Example: Product of Maxterms

- $F = xy + x'z$
  - Use the distributive law + over ·
  - $F = (xy + x') \cdot (xy + z)$
    $= (x + x') \cdot (y + x') \cdot (x + z) \cdot (y + z)$
    $= (y + x') \cdot (x + z) \cdot (y + z)$
    $= (y + x' + zz') \cdot (x + z + yy') \cdot (y + z + xx')$
    $= (x' + y + z) \cdot (x' + y + z') \cdot (x + y + z) \cdot (x + y' + z)$
    $\cdot (x + y + z) \cdot (x' + y + z)$
  - $= \Pi (4, 5, 0, 2)$

Conversion Between Canonical Forms

- Fact:
  - The complement of a function (given in sum of minterms) can be expressed as a sum minterms missing from the original function
- Example:
  - $F(x, y, z) = \Sigma (1, 4, 5, 6, 7)$
  - $F'(x, y, z) = \Sigma (0, 2, 3) = m_0 + m_2 + m_3$.
  - Now take the complement of F' and make use of DeMorgan's theorem
  - $(F')' = (m_0 + m_2 + m_3)' = (m_0' \cdot m_2' \cdot m_3')$
  - $F = M_0 \cdot M_2 \cdot M_3 = \Pi (0, 2, 3)$

General Rule for Conversion

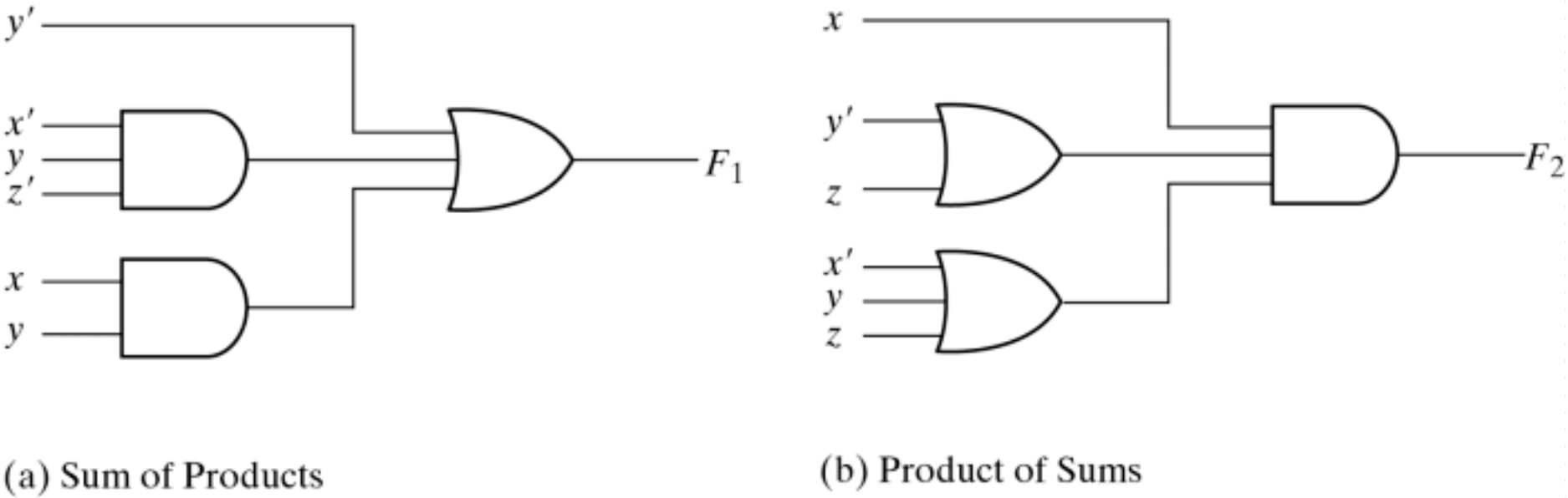
- Important relation:
  - $m_j' = M_j$.
  - $M_j' = m_j$.
- The rule:
  - Interchange symbols Π and Σ, and
  - list those terms missing from the original form
- Example: $F = xy + x'z$
  - derive the truth table
  - Obtain the sum of minterms form
  - $F = \Sigma(1, 3, 6, 7)$
  - $F = \Pi(0, 2, 4, 5)$

Standard Forms

- Fact:
  - Canonical forms are very seldom the ones with the least number of literals
- Alternative representation:
  - Standard form
    - a term may contain any number of literals
  - Two types
    1. the sum of products
    2. the product of sums
  - Examples:
    - $F_1 = y' + xy + x'yz'$
    - $F_2 = x(y' + z)(x' + y + z')$

Example: Standard Forms

- $F_1 = y' + xy + x'yz'$
- $F_2 = x(y' + z)(x' + y + z')$



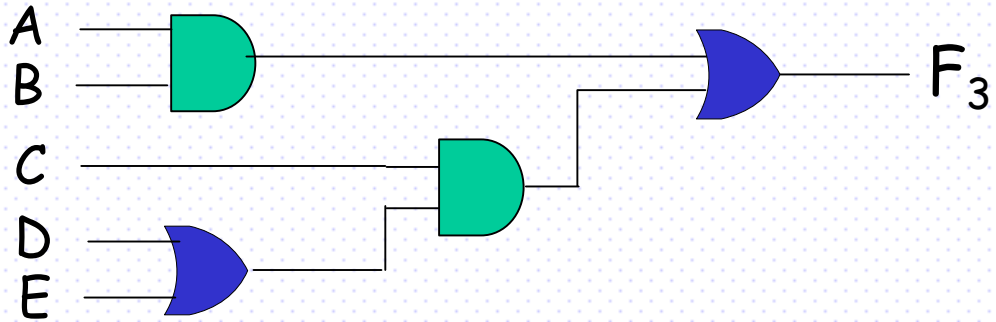
(a) Sum of Products

(b) Product of Sums

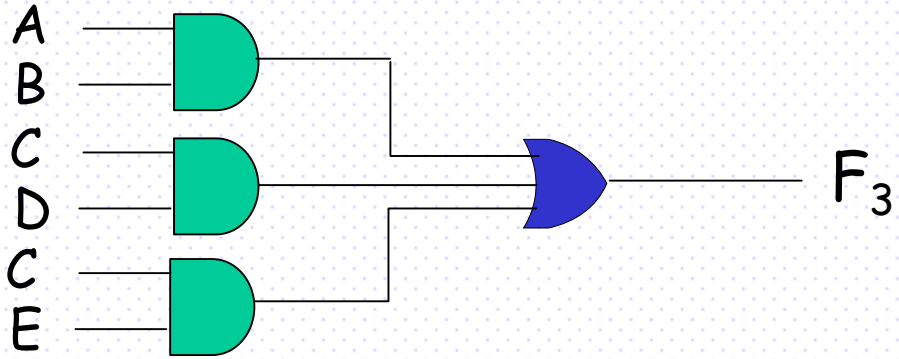
Fig. 2-3 Two-level implementation

Nonstandard Forms

- Example:
  - $F_3 = AB + C(D + E)$
  - This hybrid form yields three-level implementation



  - We can always get the standard form
  - $F_3 = AB + CD + CE$



Positive & Negative Logic

- In digital circuits, we have two digital signal level:
  - H – (higher signal level; e.g. 3 ~ 4 V)
  - L - (lower signal level; e.g. 0 ~ 1 V)
- There is no logic-1 or logic-0 at the circuit level
- We can do any assignment we wish
  - For example:
    - H → logic-1
    - L → logic-0

Fig. 2-9 signal assignment and logic polarity

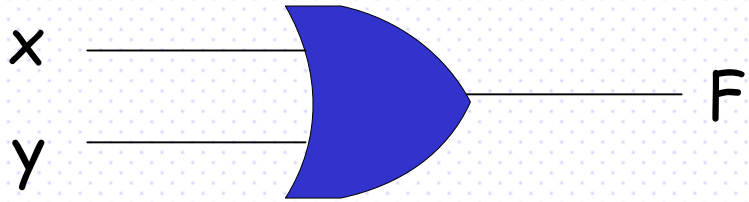
Signal Designation - 1

x	y	F
L	L	L
L	H	H
H	L	H
H	H	H

- What kind of logic function does it implement?

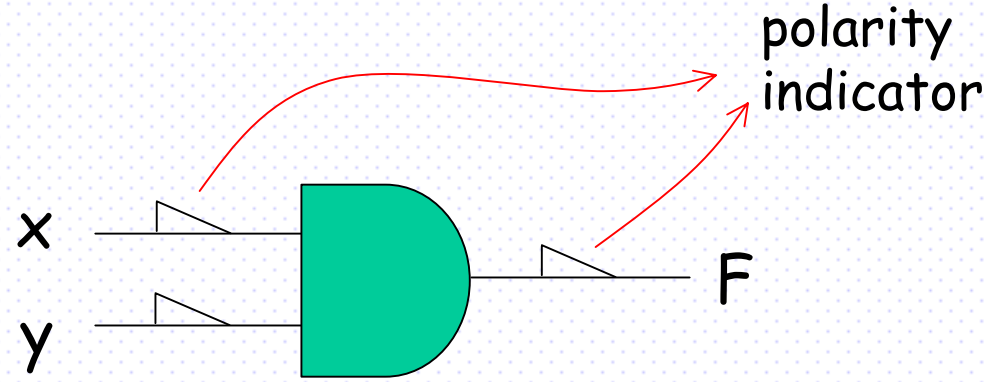
Signal Designation - 2

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1



positive logic

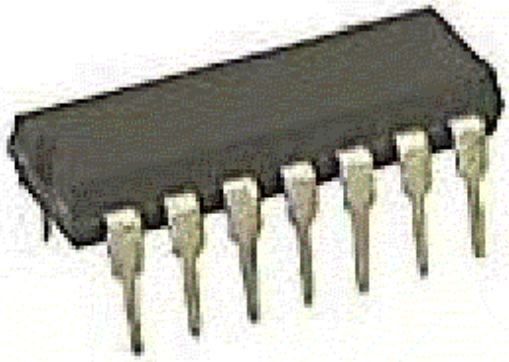
x	y	F
1	1	1
1	0	0
0	1	0
0	0	0



negative logic

Integrated Circuits

- IC – silicon semiconductor crystal ("chip") that contains gates.
  - gates are interconnected inside to implement a Boolean function
  - Chip is mounted in a ceramic or plastic container
  - Inputs & outputs are connected to the external pins of the IC.
  - Many external pins (14 to several thousands)



Levels of Integration

- SSI (small-scale integration):
  - Up to 100 electronic components per chip
- MSI (medium-scale integration):
  - From 100 to 3,000 electronic components per chip
- LSI (large-scale integration):
  - From 3,000 to 100,000 electronic components per chip
- VLSI (very large-scale integration):
  - From 100,000 to 1,000,000 electronic components per chip
- ULSI (ultra large-scale integration):
  - More than 1 million electronic components per chip

Digital Logic Families

- Circuit Technologies
- TTL → transistor-transistor logic
- ECL → Emitter-coupled logic
  - fast
- MOS → metal-oxide semiconductor
  - high density
- CMOS → Complementary MOS
  - low power

Parameters of Logic Gates - 1

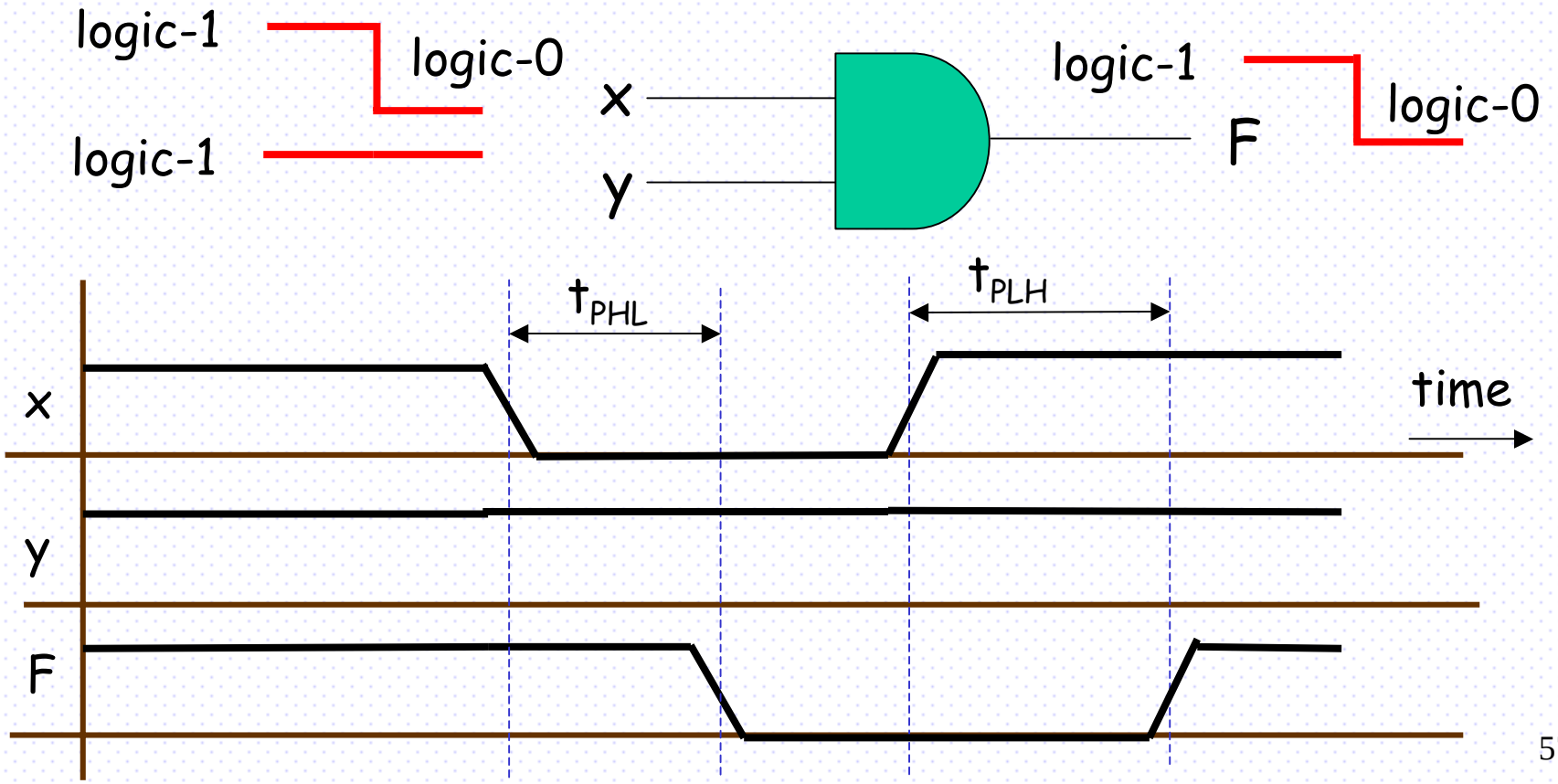
- Fan-out
  - load that the output of a gate drives
  - standard load may be defined as the amount of current needed by the input of a gate of the same family.
  - Maximum fan-out specifies the fan-out that the output can drive without impairing the gate performance
  - Example: the input to an inverter can have load equal to 1.0 standard load.
    - If a, say NAND, gate drives six such inverters, then the fan-out is equal to 6.0 standard loads.
  - The transition time of a gate is affected by the fan-out in CMOS technology
  - Thus, maximum fan-out is the standard load with which the transition time is no greater than the maximum allowable value.

Parameters of Logic Gates - 2

- Fan-in
  - number of inputs that a gate can have in a particular logic family
  - In principle, we can design a CMOS NAND or NOR gate with a very large number of inputs
  - In practice, however, we have some limits
  - 4 for NOR gates
  - 6 for NAND gates
- Power dissipation
  - power consumed by the gate that must be available from the power supply

Parameters of Logic Gates - 3

- Propagation delay:
  - the time required for a change in value of a signal to propagate from input to output.



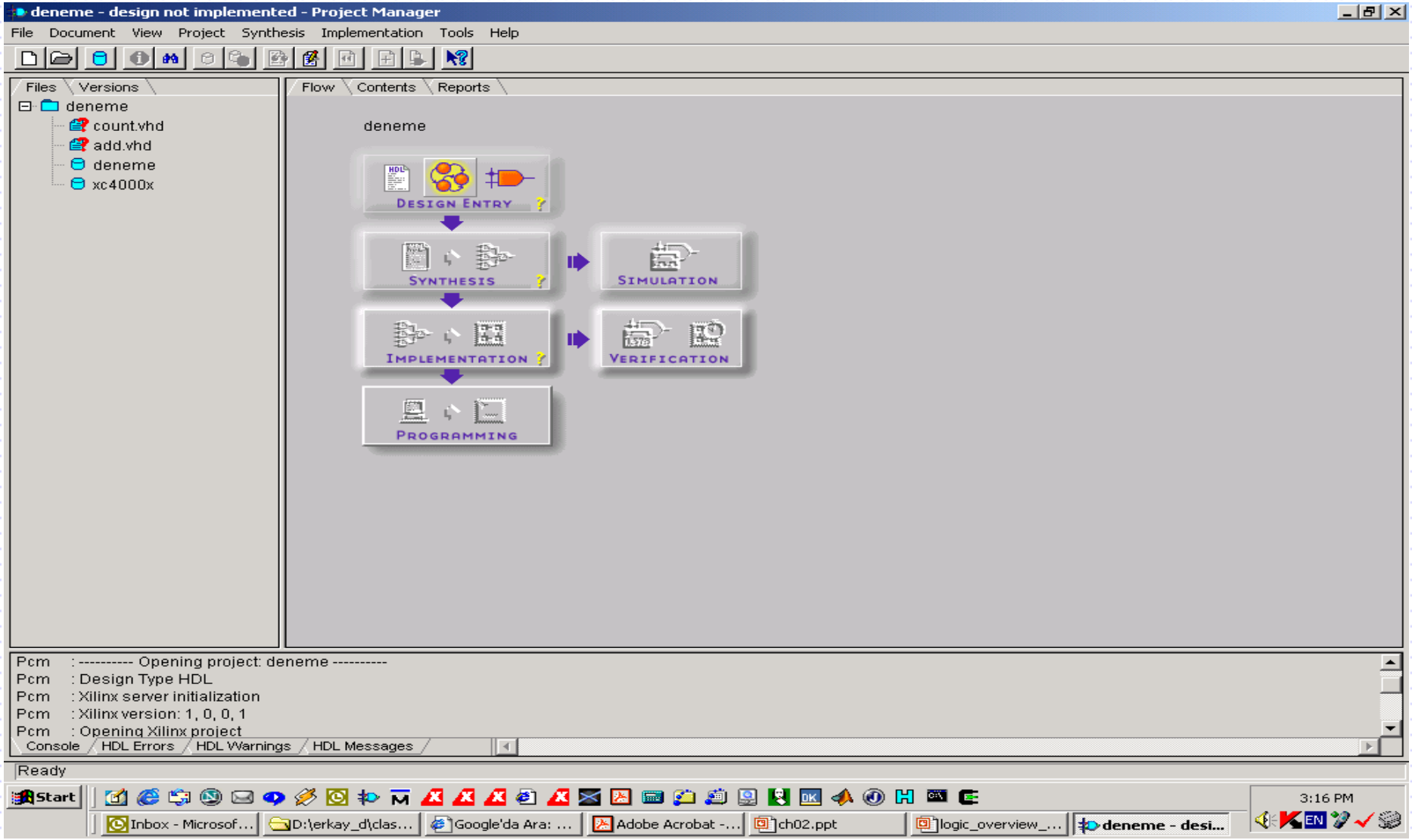
Computer-Aided Design - 1

- CAD
  - Design of digital systems with VLSI circuits containing millions of transistors is not easy and cannot be done manually.
- To develop & verify digital systems we need CAD tools
  - software programs that support computer-based representation of digital circuits.
- Design process
  - design entry
  - …
  - database that contains the photomask used to fabricate the IC

Computer-Aided Design - 2

- Different physical realizations
  - ASIC (application specific integrated circuit)
  - FPGA
  - PLD
- For every piece of device we have an array of software tools to facilitate
  - design,
  - testing,
  - and even programming

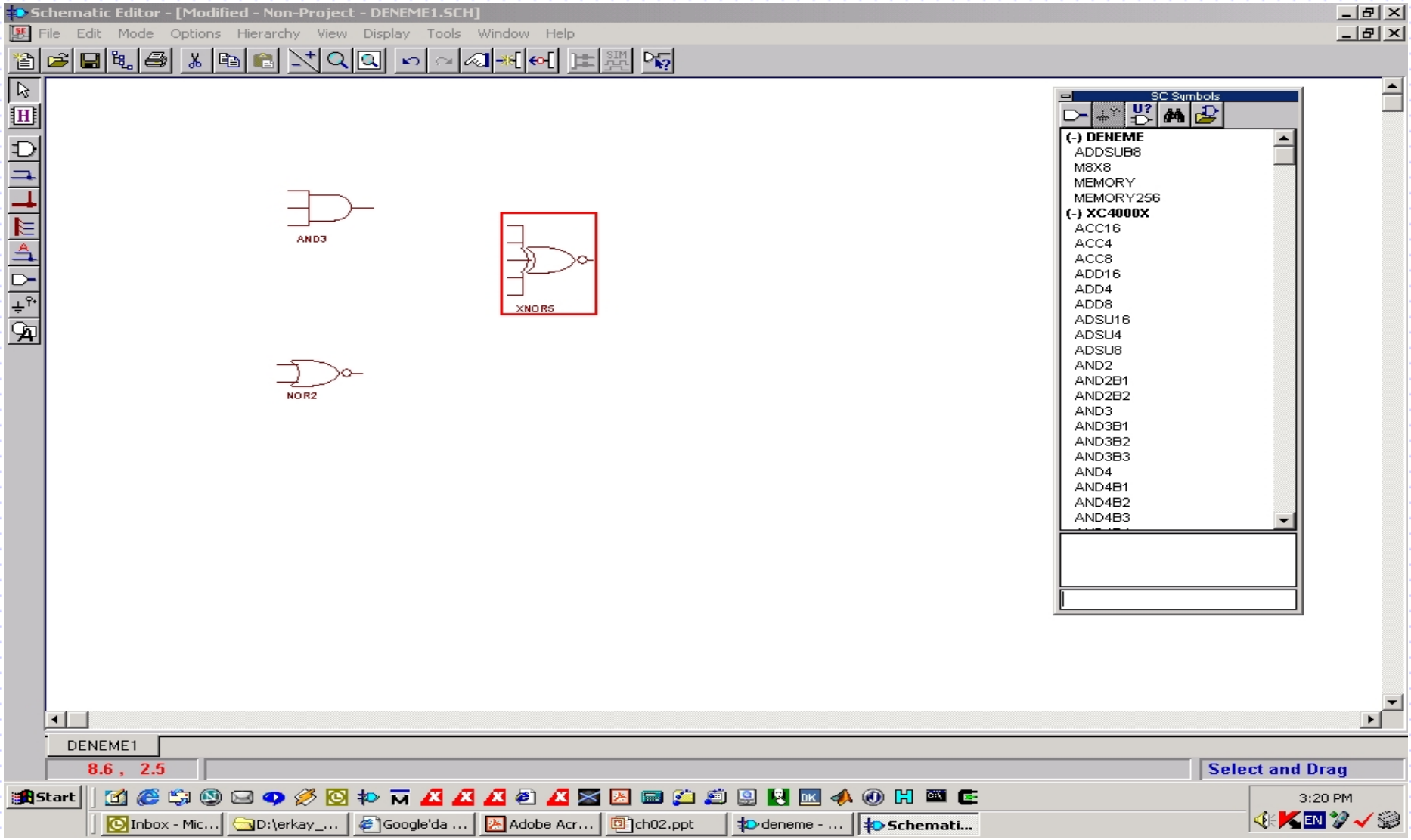
Xilinx Foundation



Schematic Editor

- Editing programs for creating and modifying schematic diagrams on a computer screen
  - schematic capturing or schematic entry
  - you can drag-and-drop digital components from a list in an internal library (gates, decoders, multiplexers, registers, etc.)
  - You can draw interconnect lines from one component to another

Schematic Editor



Hardware Description Languages

- HDL
  - Verilog
  - VHDL
  - resembles a programming language
  - designed to describe digital circuits so that we can develop and test digital circuits