CS 303
Logic & Digital System Design

Ömer Ceylan

Sabancı Üniversitesi

Boolean Algebra

Boolean Algebra 1/2

- A set of elements B
  - There exist at least two elements x, y ∈ B s. t. x ≠ y
- Binary operators: + and ·
  - <u>closure</u> w.r.t. both + and ·
  - additive identity ?
  - multiplicative identity ?
  - 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 2/2

- Complement
  - $\forall$ x $\in$ B, there exist an element x' $\in$ B $\ni$
    a. x + x' = 1 (multiplicative identity) and
    b. x · x' = 0 (additive identity)
  - 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 1/3

- To define a Boolean algebra
  - The set 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 2/3

- 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)

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

Two-Valued Boolean Algebra 3/3

- Two-valued Boolean algebra is actually equivalent to the binary logic defined heuristically before
  - Operations:
    - · → AND
    - + → OR
    - Complement → NOT
- <u>Binary logic is the application of Boolean algebra to the gate-type circuits</u>
  - 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)
    $= (x+x) \cdot (x+x')$ (complement)
    $= x+x \cdot x'$ (+ over ·)
    $= x$ (complement)
  - duality principle
    $x \cdot x = x \rightarrow$?

Duality Principle & Theorems

- Theorem a:
  - $x + 1 = 1$
  - $x + 1 = 1 \cdot (x+1)$
    $= (x+x')\cdot(x+1)$
    $= x+x'\cdot 1$
    $= x+x'$
    $= 1$
- Theorem b: (using duality)
  - ?

Absorption Theorem

a. $x + xy = x$

b. ?

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:

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

Truth Tables for DeMorgan’s Theorem

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

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

x’	y ’	x’ · y ’	x’ + y’

Operator Precedence

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

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

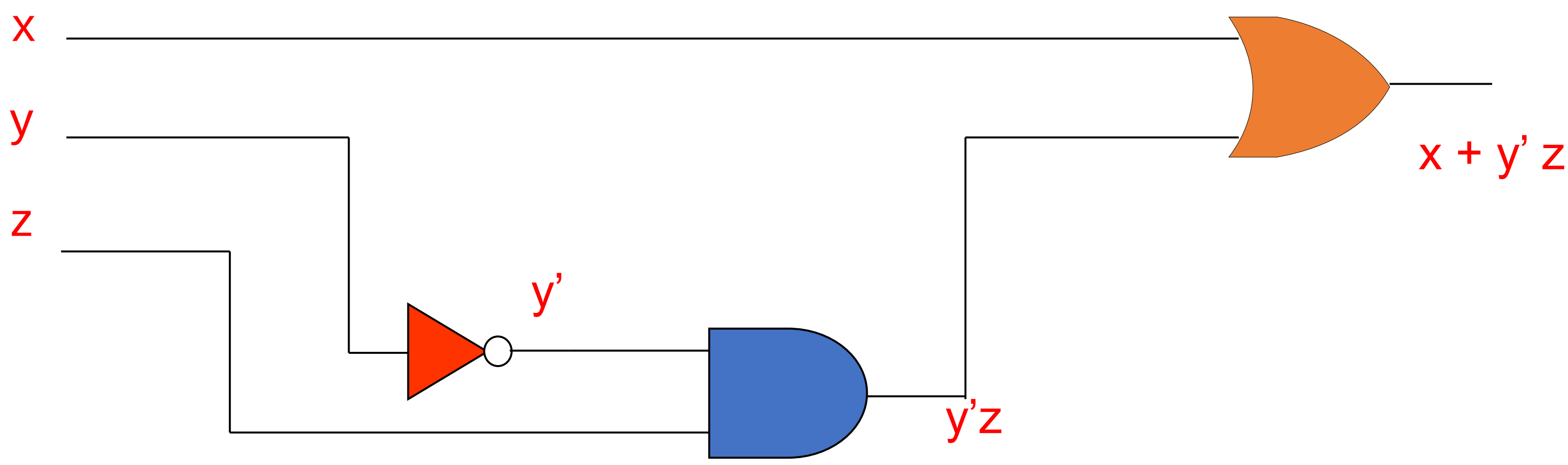
Boolean Functions

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

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

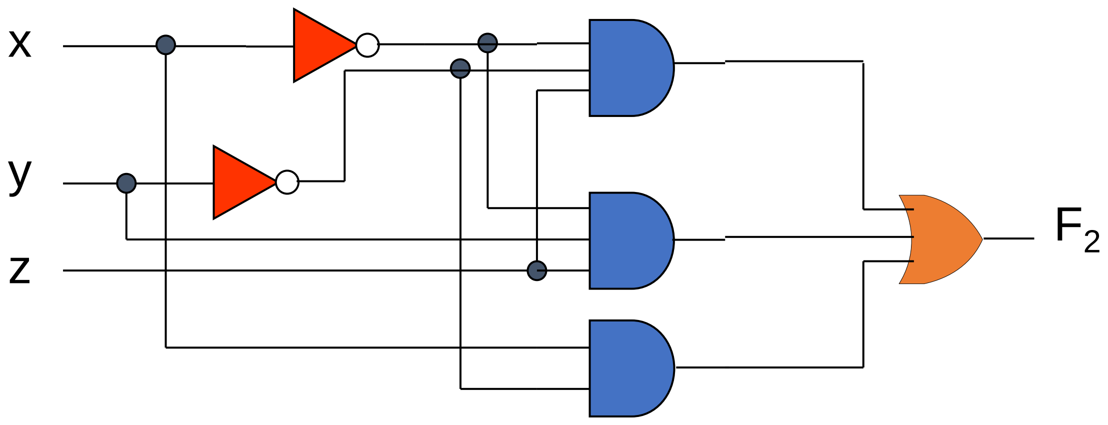
$$F_1(x, y, z) = x + y' z$$



Gate Implementation of $F_1 = x + y' z$

Logic Circuit Diagram of F_2

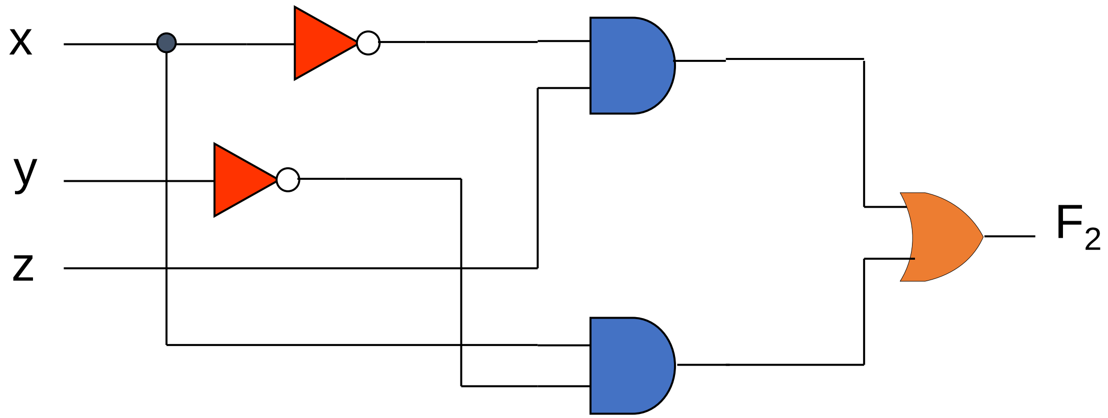
$F_2 = x' y' z + x' y z + xy'$



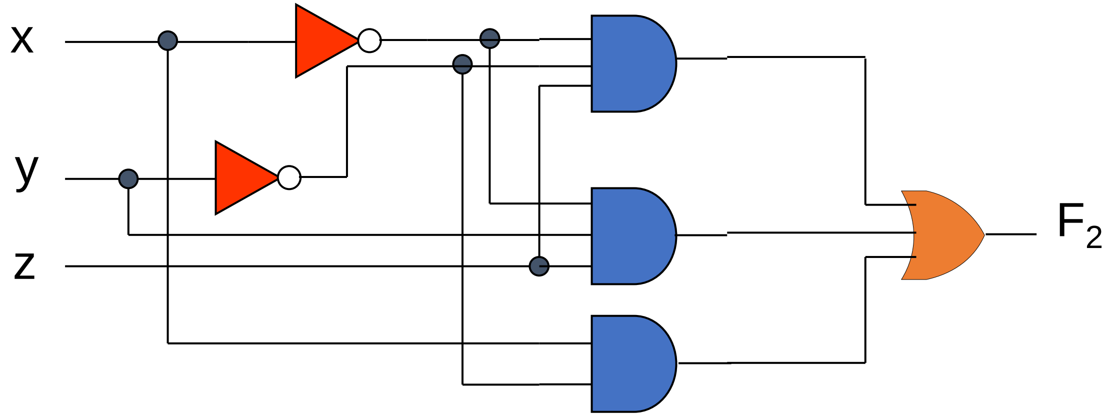
- Algebraic manipulation
- $F_2 = x' y' z + x' y z + xy'$
  $= ?$

Alternative Implementation of F_2

$F_2 = x' z + xy'$



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



Complement of a Function

- F' is complement of F
  - We can obtain F', by interchanging of 0's and 1's in the truth table

x	y	z	F	F'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

F = x'yz' + xy'z' + xy'z

F' = ?

Generalizing Demorgan's Theorem

- We can also utilize DeMorgan's Theorem
  - $(x + y)' = x' y'$
  - $(A + B + C)' = A'B'C'$
- 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. $(x_1 \cdot x_2 \cdot \ldots \cdot x_N)' = x_1' + x_2' + \ldots + x_N'$

Example: Complement of a Function

- Example:
  - $F_1 = x'yz' + x'y'z$
  - $F_1' = (x'yz' + x'y'z)'$

    $= ?$

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

    $= ?$

    $= x' + (y + z)(y' + z')$
- Easy Way to Complement: take the dual of the function and complement each literal

Canonical & Standard Forms

- Minterms
  - <u>A product term</u>: all variables appear (either in its normal, x, or its complement form, x')
  - How many different terms we can get with x and y?
    - x'y' → 00 → m_0
    - x'y → 01 → m_1
    - xy' → 10 → m_2
    - xy → 11 → m_3
  - m_0, m_1, m_2, m_3 (minterms or AND terms, standard product)
  - n variables can be combined to form 2^n minterms

Canonical & Standard Forms

- Maxterms (OR terms, standard sums)
  - $M_0 = x + y \rightarrow 00$
  - $M_1 = x + y' \rightarrow 01$
  - $M_2 = x' + y \rightarrow 10$
  - $M_3 = x' + y' \rightarrow 11$
  - n variables can be combined to form 2^n maxterms
  - $m_0' = M_0$
  - $m_1' = M_1$
  - $m_2' = M_2$
  - $m_3' = M_3$

Example

xyz	m_i	M_i	F
000	$m_0=x'y'z'$	$M_0=x+y+z$	0
001	$m_1=x'y'z$	$M_1=x+y+z'$	1
010	$m_2=x'yz'$	$M_2=x+y'+z$	1
011	$m_3=x'yz$	$M_3=x+y'+z'$	0
100	$m_4=xy'z'$	$M_4=x'+y+z$	0
101	$m_5=xy'z$	$M_5=x'+y+z'$	0
110	$m_6=xyz'$	$M_6=x'+y'+z$	1
111	$m_7=xyz$	$M_7=x'+y'+z'$	0

$F(x, y, z) = x'y'z + x'yz' + xyz' =$

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

Another Example

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

- $F_1(x, y, z) =$
- $F_2(x, y, z) =$

Important Properties

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

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: $F = x + y'z \rightarrow$ canonical form?
  - No
  - But we can put it in canonical form.
  - $x \rightarrow x(y+y')(z+z') = (xy+xy')(z+z') = xyz + xyz' + xy'z + xy'z'$
  - $y'z \rightarrow$
  - $x + y'z = xyz + xyz' + xy'z + xy'z' + xy'z + x'y'z$
  - $F = x + y'z = \sum (7, 6, 5, 4, 1)$
- 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 ·
  - $F = xy + x' \cdot z$

$$= (xy + x') \cdot (xy + z)$$

$$= \Pi\ (4, 5, 0, 2)$$

Conversion Between Canonical Forms

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

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

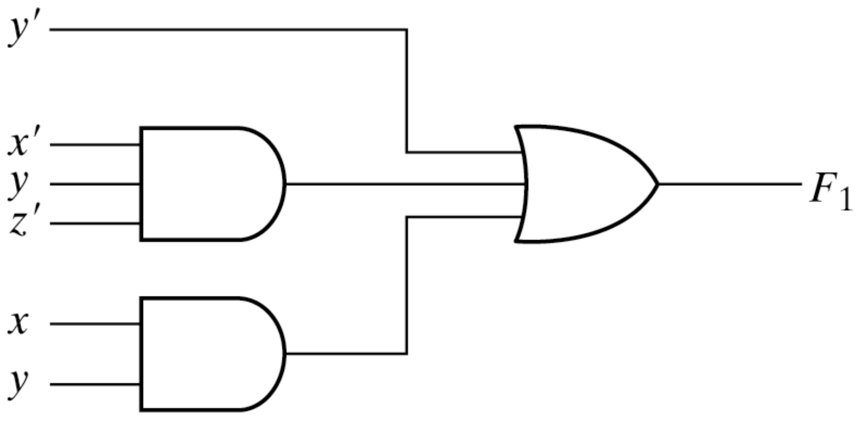
  - $F = \Sigma(1, 3, 6, 7) \rightarrow F = \Pi(?, ?, ?, ?)$

Standard Forms

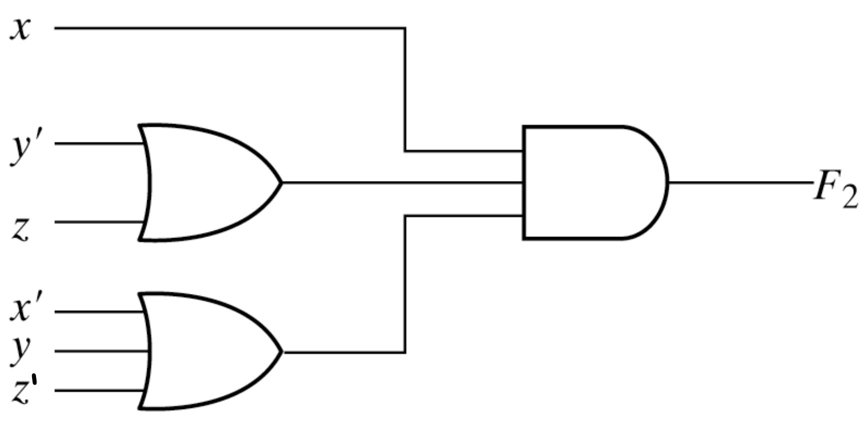
- Fact:
  - Canonical forms are very seldom the ones with the least number of literals
- Alternative representation:
  - Standard form
    - a term may contain any number of literals
  - Two types
    1. the sum of products
    2. the product of sums
  - Examples:
    - $F_1 = y' + xy + x'yz'$
    -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

Example: Standard Forms

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



(a) Sum of Products

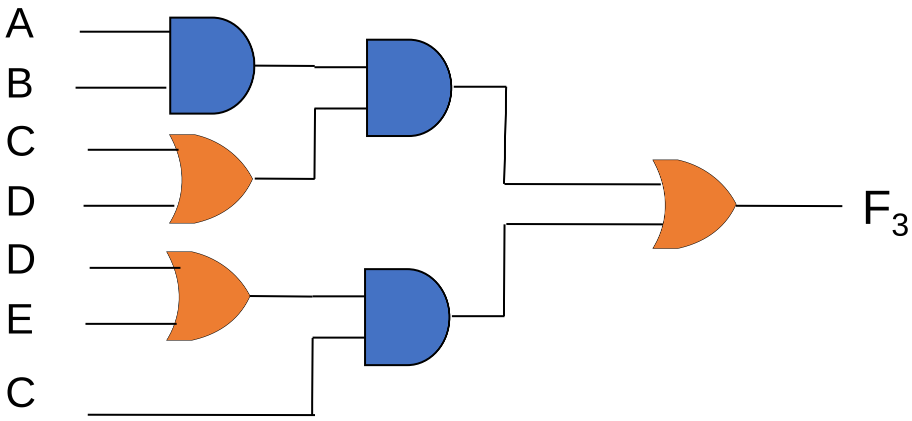


(b) Product of Sums

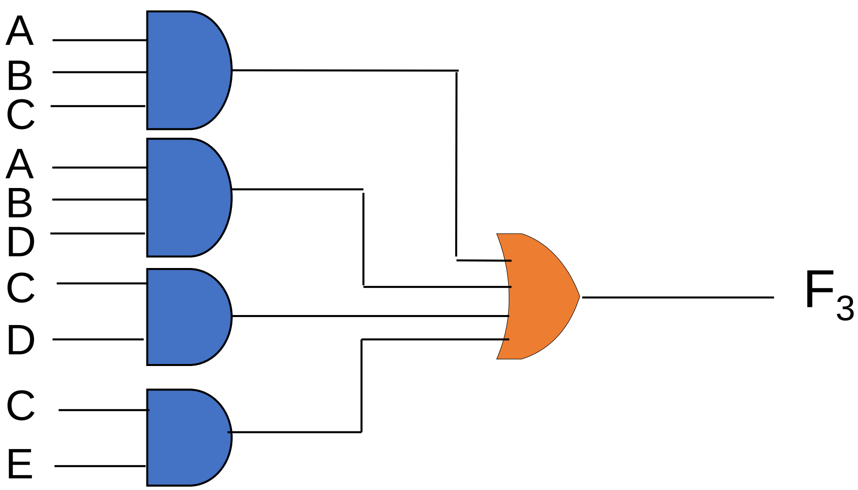
Fig. 2-3 Two-level implementation

Nonstandard Forms

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



– The standard form: $F_3 = ABC + ABD + CD + CE$



OTHER LOGIC OPERATORS - 1

- AND, OR, NOT are logic operators
  - Boolean functions with two variables
  - three of the 16 possible two-variable Boolean functions

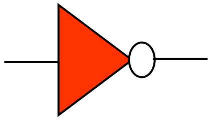
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

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

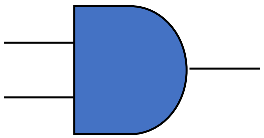
OTHER LOGIC OPERATORS - 2

- 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)

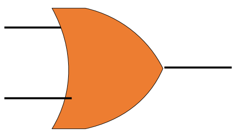
Logic Gate Symbols



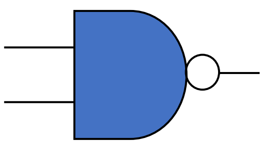
NOT



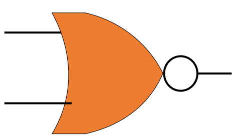
AND



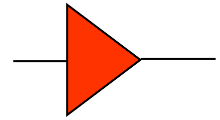
OR



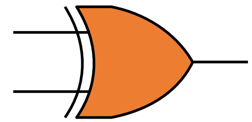
NAND



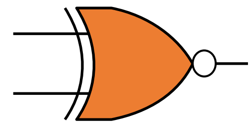
NOR



TRANSFER



XOR



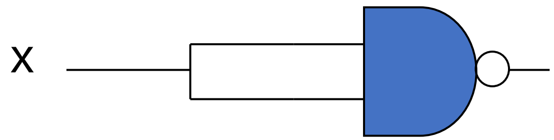
XNOR

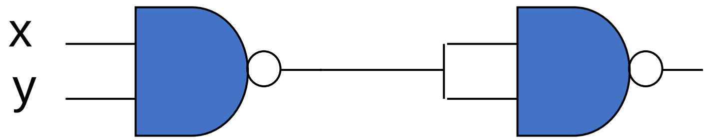
Universal Gates

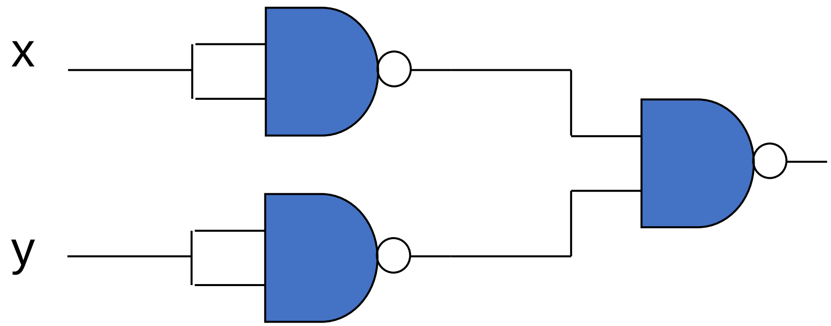
- NAND and NOR gates are universal
- We know any Boolean function can be written in terms of three logic operations:
  - AND, OR, NOT
- In return, NAND gate can implement these three logic gates by itself
  - So can NOR gate

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

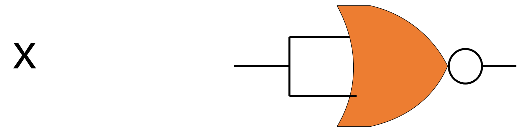
NAND Gate

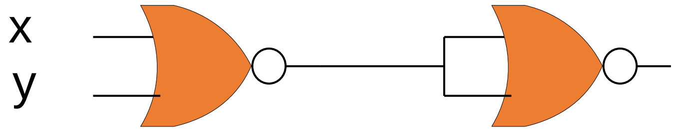


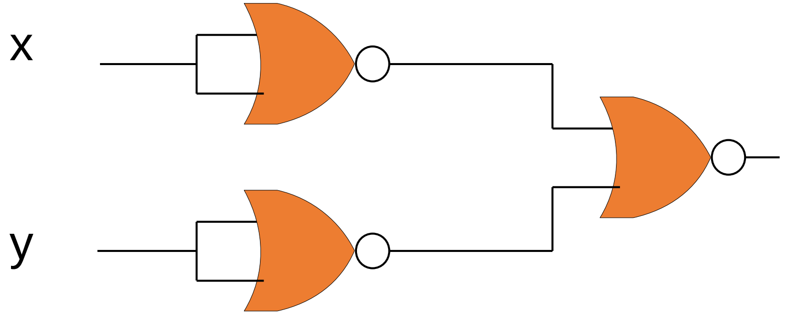




NOR Gate

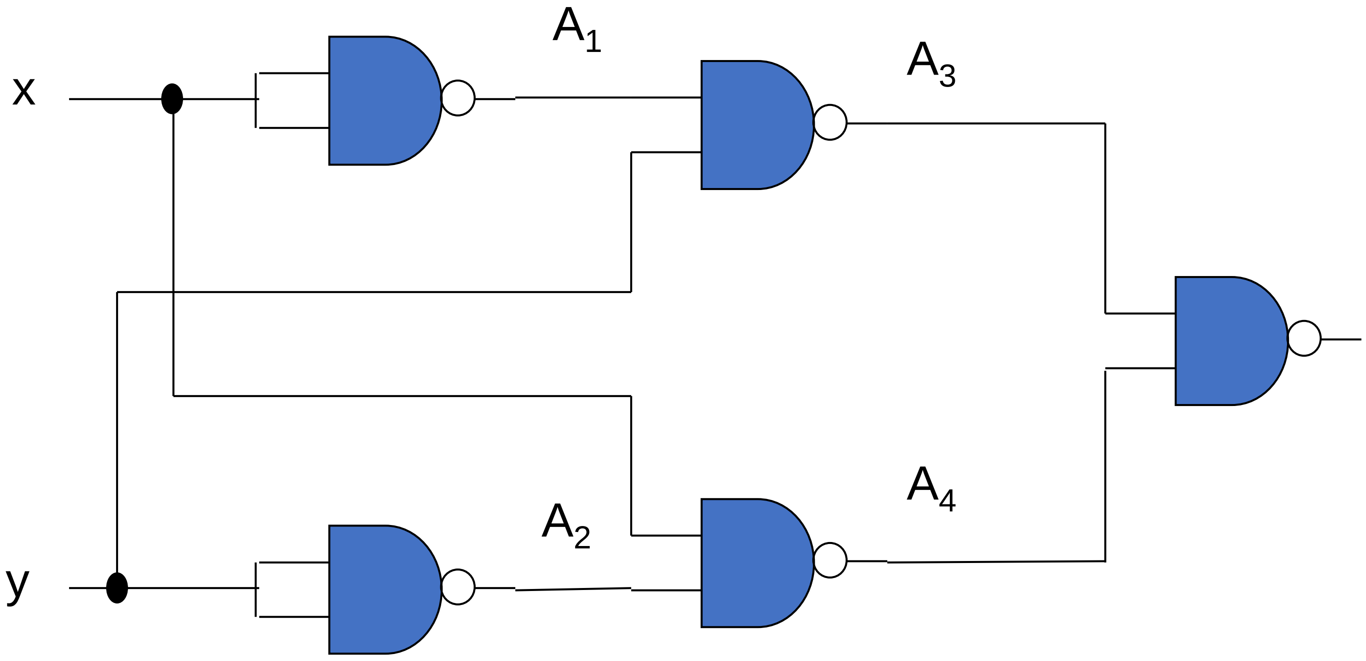






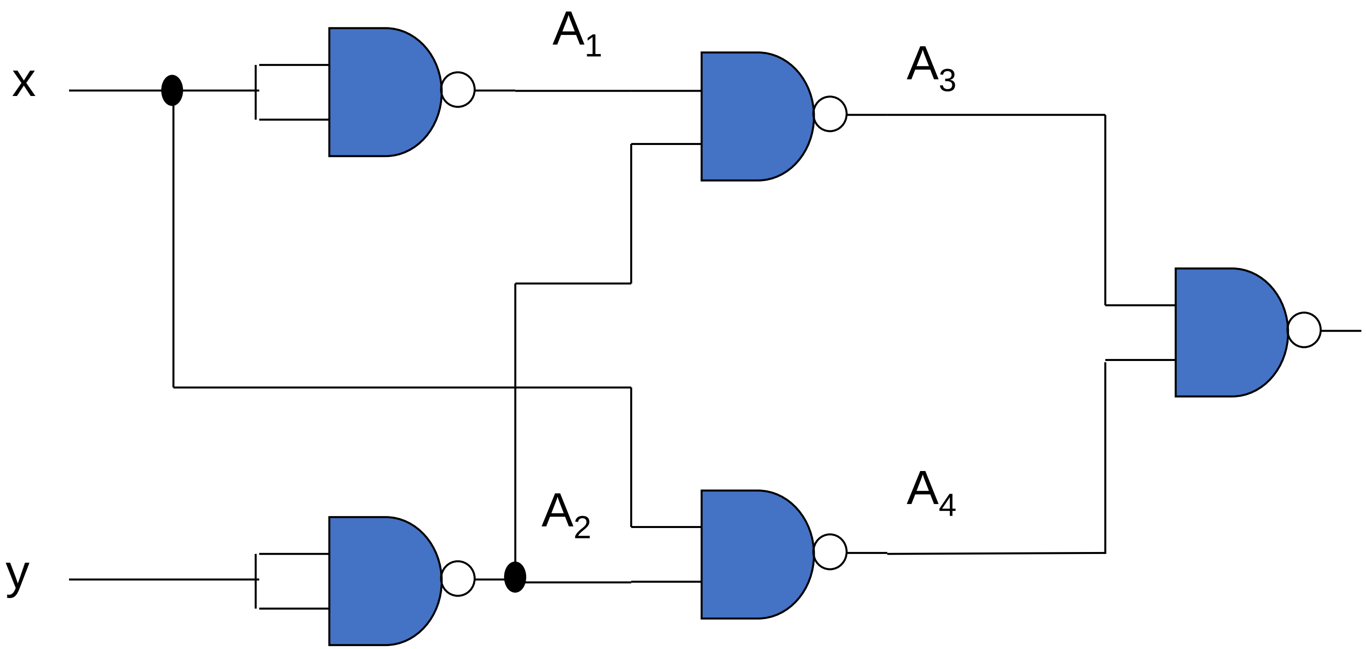
Designs with NAND gates Example 1/2

- A function:
  - $F_1 = x' y + xy'$



Example 2/2

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



Multiple Input Gates

- AND and OR operations:
  - They are both commutative and associative
  - No problem with extending the number of inputs
- NAND and NOR operations:
  - they are both commutative but <u>not associative</u>
  - Extending the number of inputs is not obvious
- <u>Example</u>: NAND gates
  - $((xy)'z)' \neq (x(yz)')'$
  - $((xy)'z)'$ = ?
  - $(x(yz)')'$ = ?

Nonassociativity of NOR operation

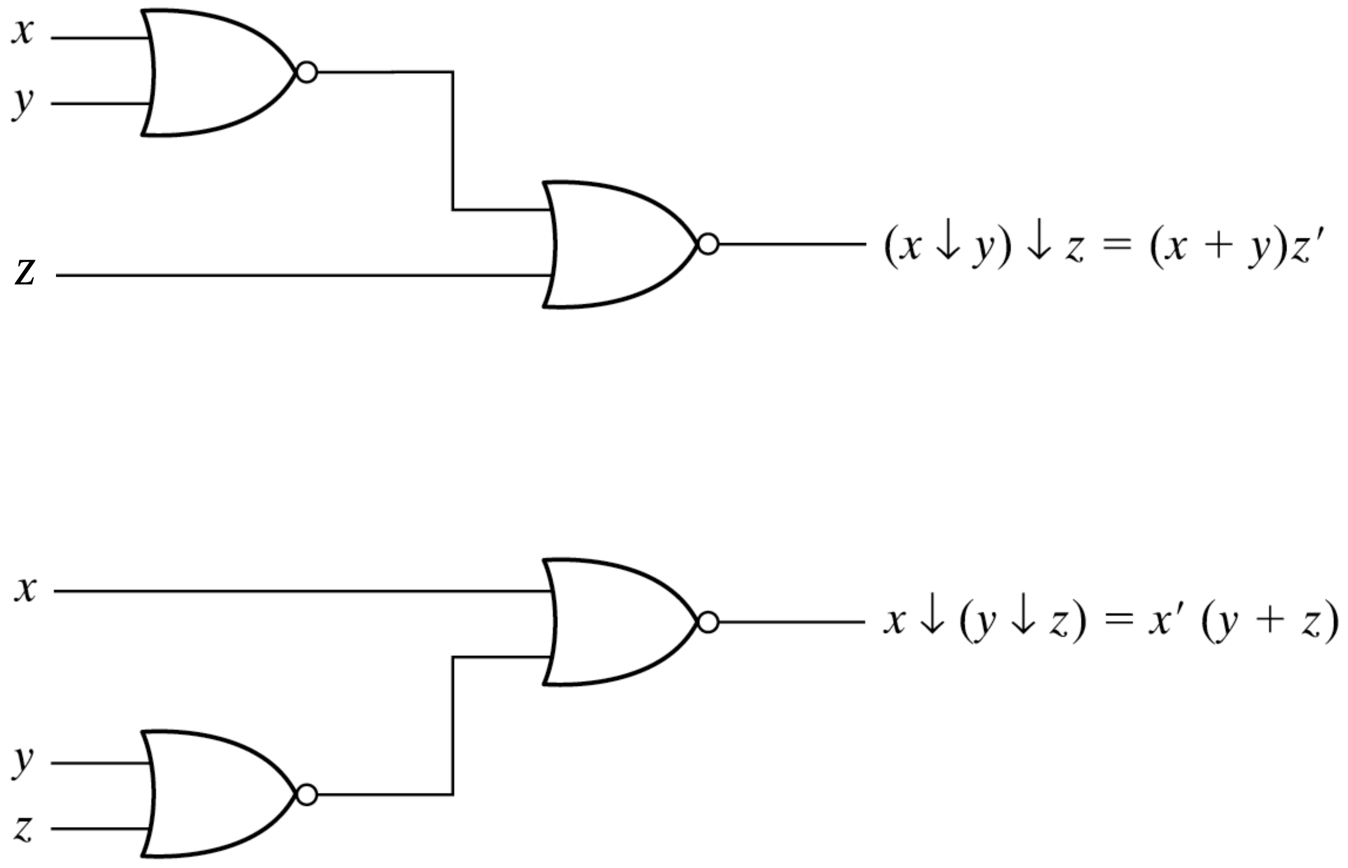
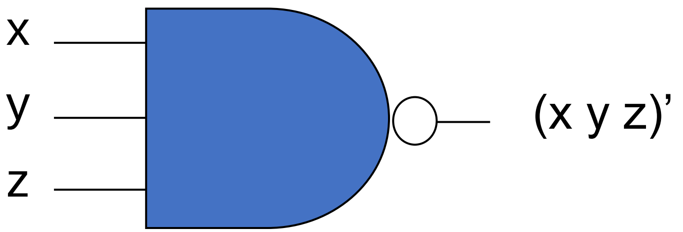


Fig. 2-6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

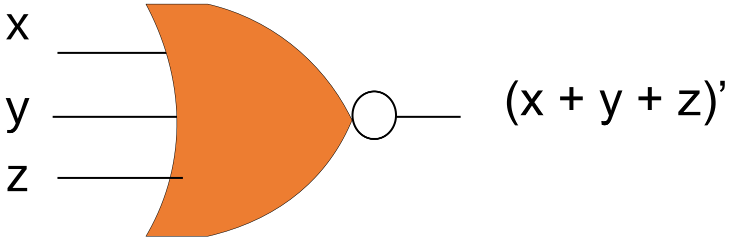
Multiple Input Universal Gates

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

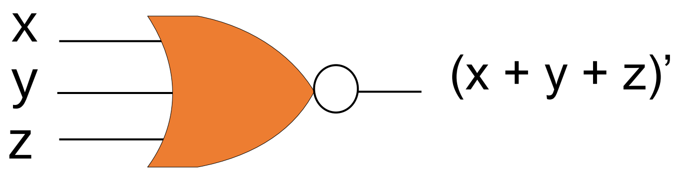
Three input NAND gate: (x y z)'



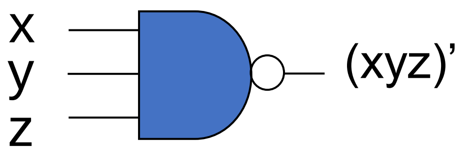
Three input NOR gate: (x + y + z)'



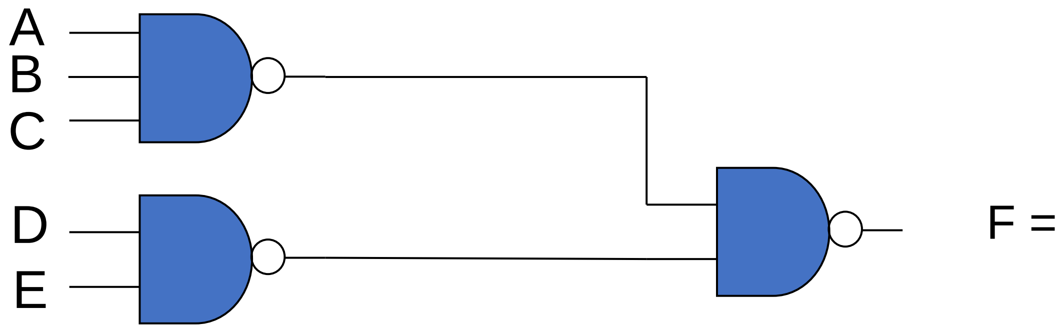
Multiple Input Universal Gates



3-input NOR gate



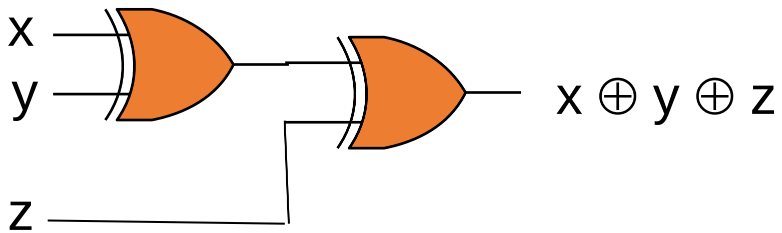
3-input NAND gate



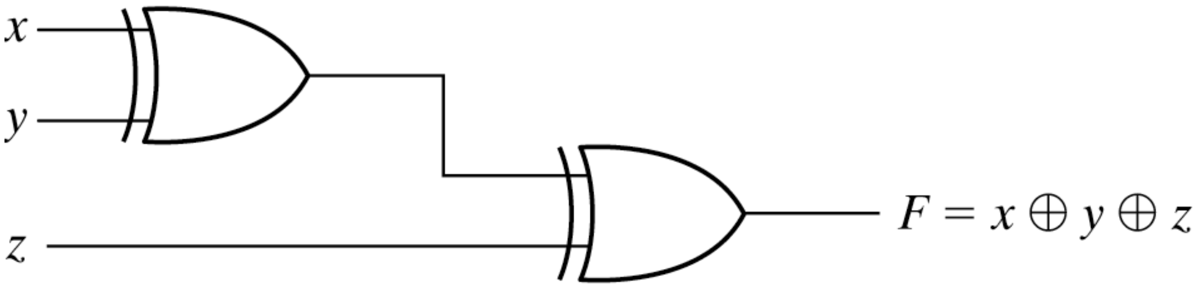
Cascaded NAND gates

XOR and XNOR Gates

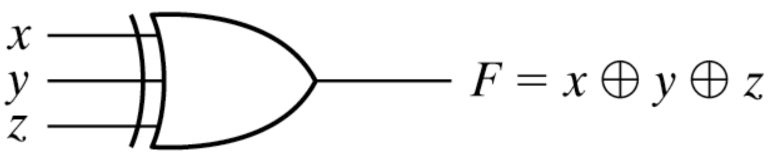
- XOR and XNOR operations are both commutative and associative.
- No problem manufacturing multiple input XOR and XNOR gates
- They are more costly from hardware point of view than AND, OR NAND and NOR gates.



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

Positive & Negative Logic

- In digital circuits, we have two digital signal levels:
  - H – (higher signal level; e.g. 3 ~ 5 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

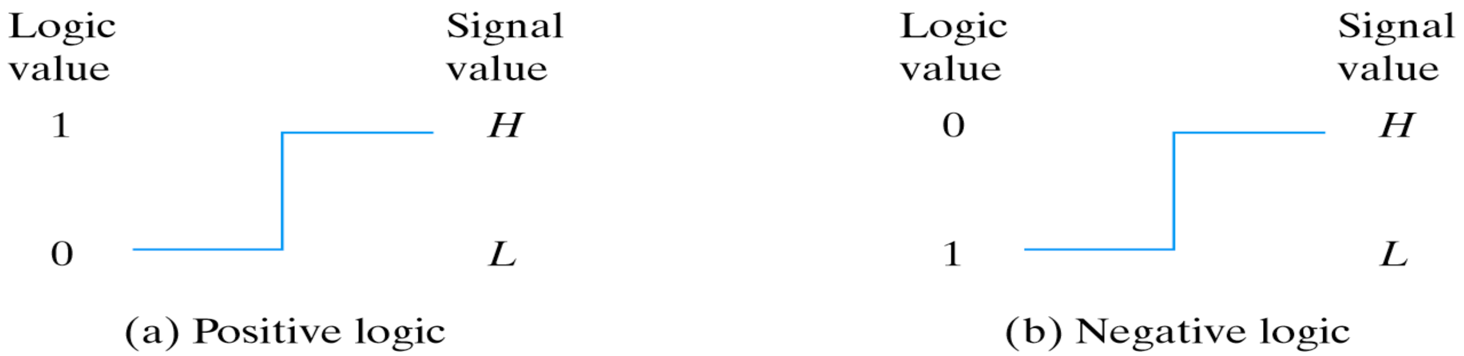
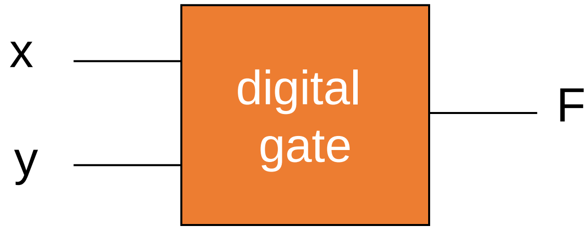


Fig. 2-9 signal assignment and logic polarity

Signal Designation - 1

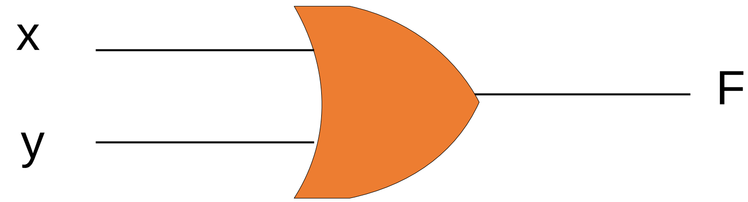


x	y	F
L	L	L
L	H	H
H	L	H
H	H	H

- What kind of logic function does it implement?

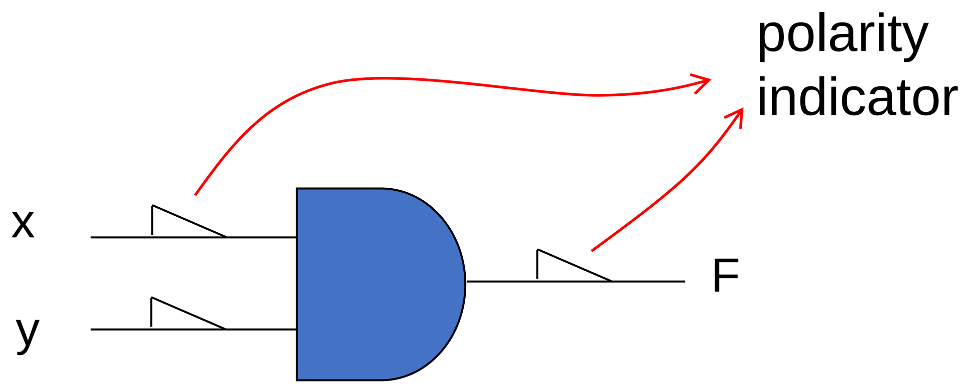
Signal Designation - 2

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

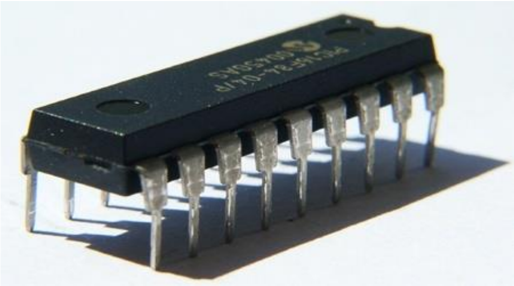
Another Example

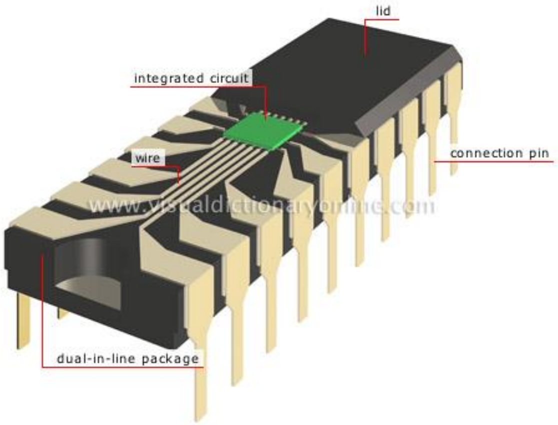
x	y	F
L	L	H
L	H	H
H	L	H
H	H	L

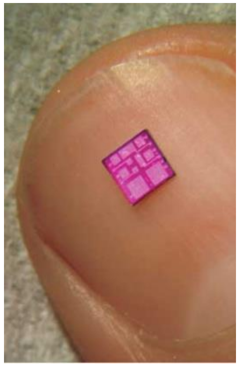
74LS00

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 hundreds)

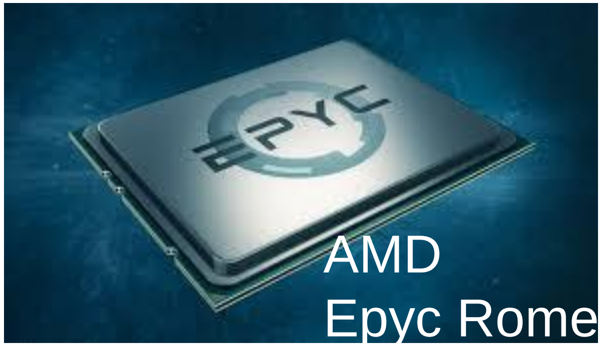






Levels of Integration

- SSI (small-scale integration):
  - Up to 10 gates per chip
- MSI (medium-scale integration):
  - From 10 to 1,000 gates per chip
- LSI (large-scale integration):
  - thousands of gates per chip
- VLSI (very large-scale integration):
  - hundreds of thousands of gates per chip
- ULSI (ultra large-scale integration):
  - Over a million gates per chip



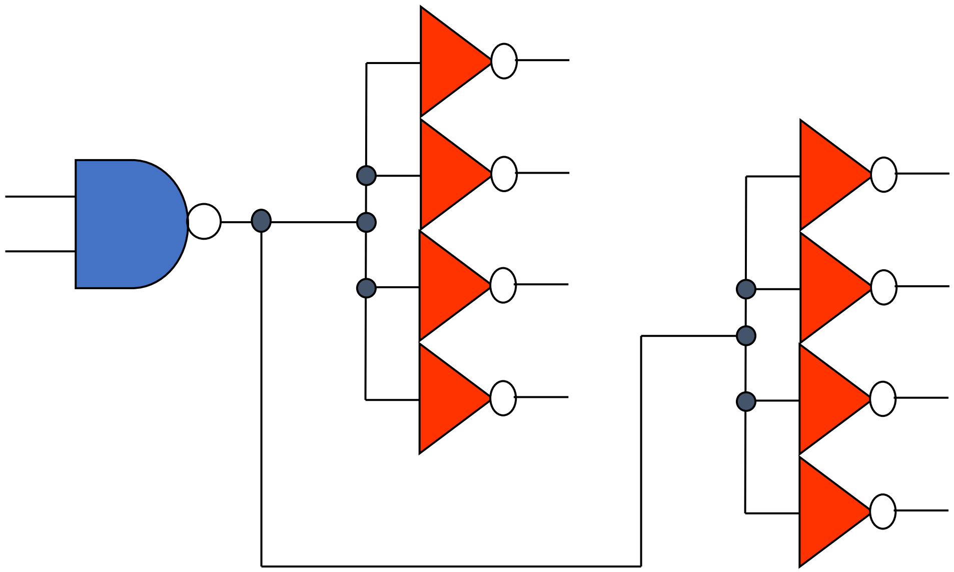
32 billion transistors

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

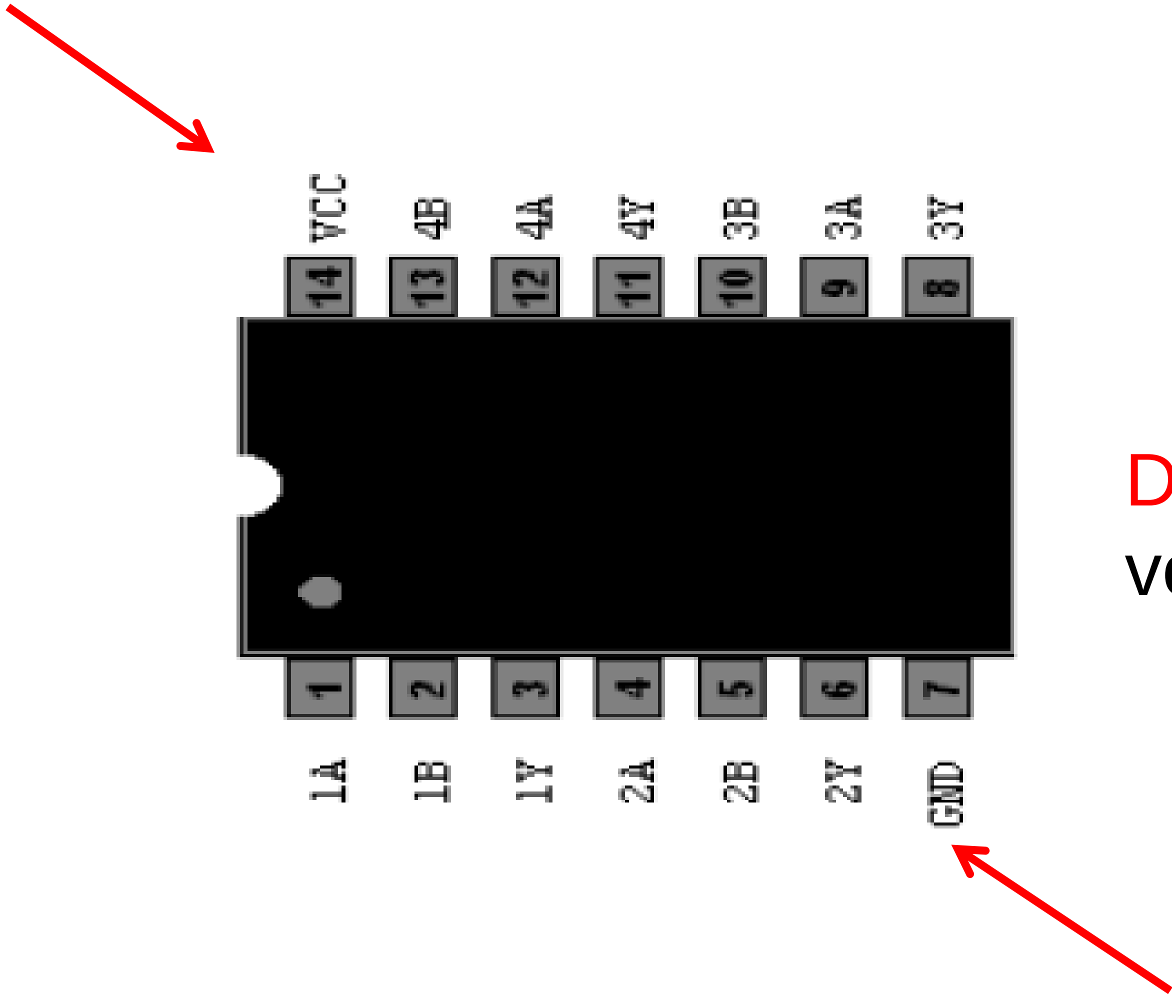


- If a, say NAND, gate drives four such inverters, then the fan-out is equal to 4.0 standard loads.

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 may have some limits
  - 4 for NOR gates
  - 6 for NAND gates
- Power dissipation
  - power needed by the gate that must be available from the power supply

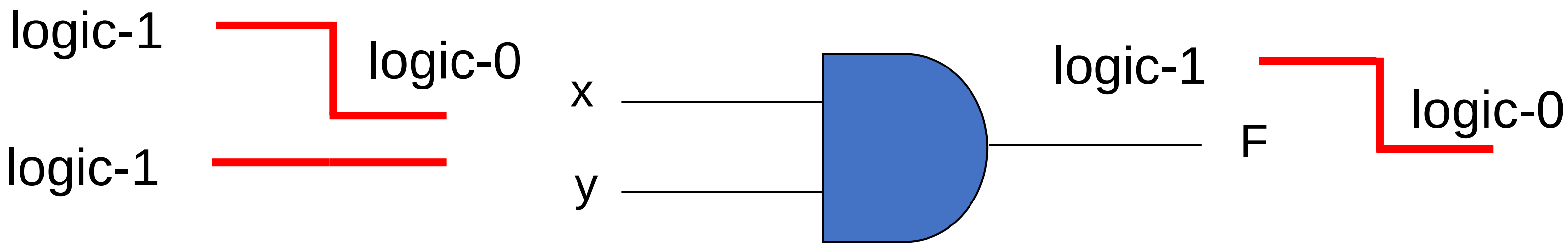
Power Dissipation

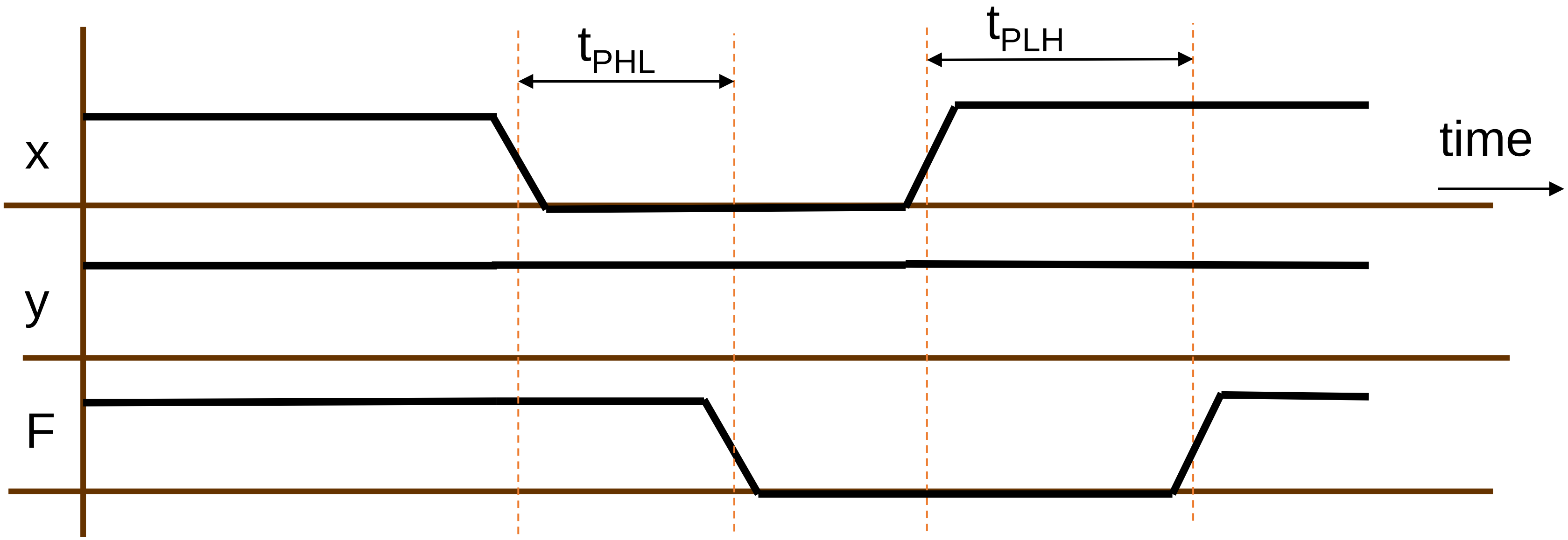


Difference in voltage level !

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
  - Configuration file to program FPGA

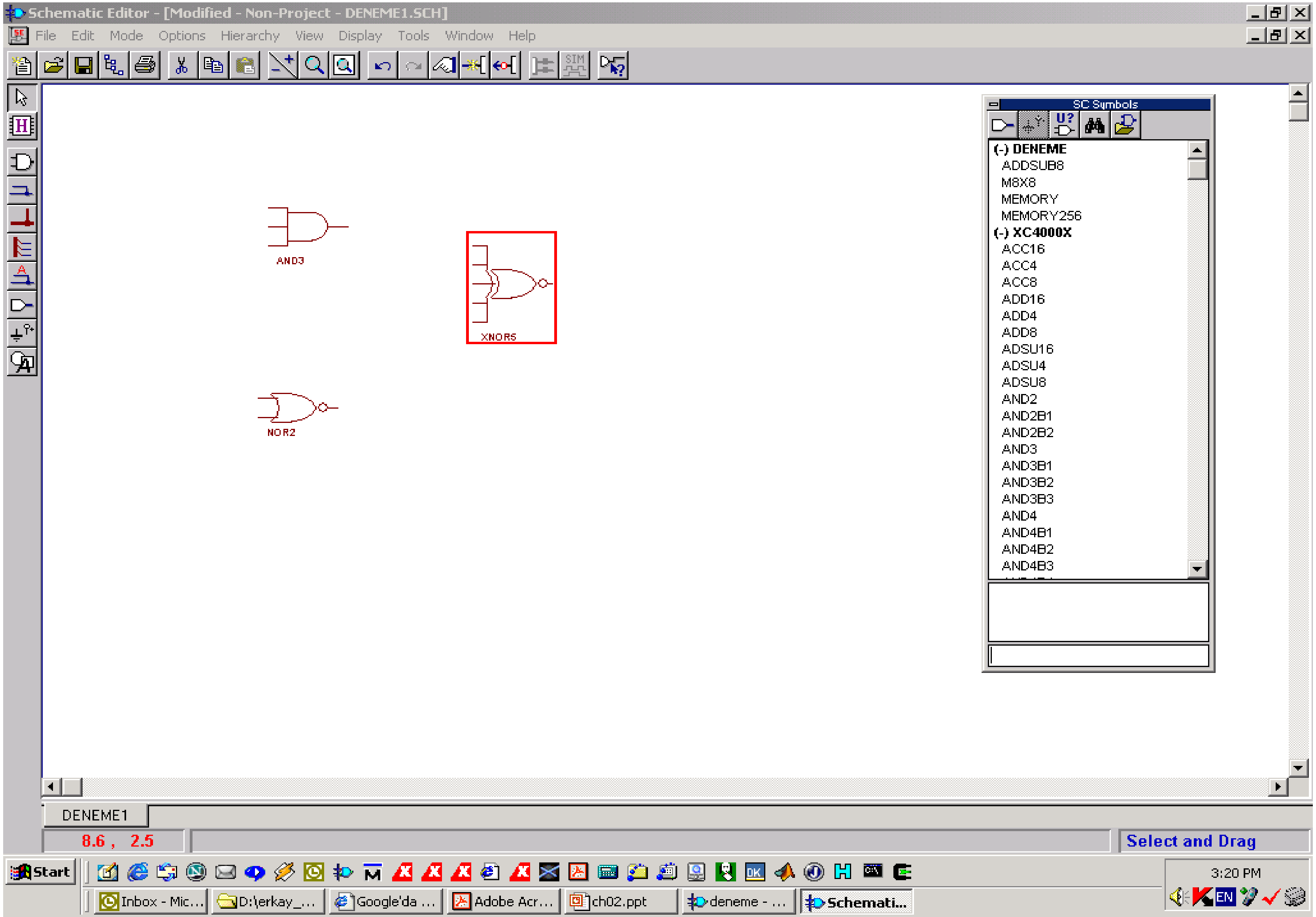
Computer-Aided Design - 2

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

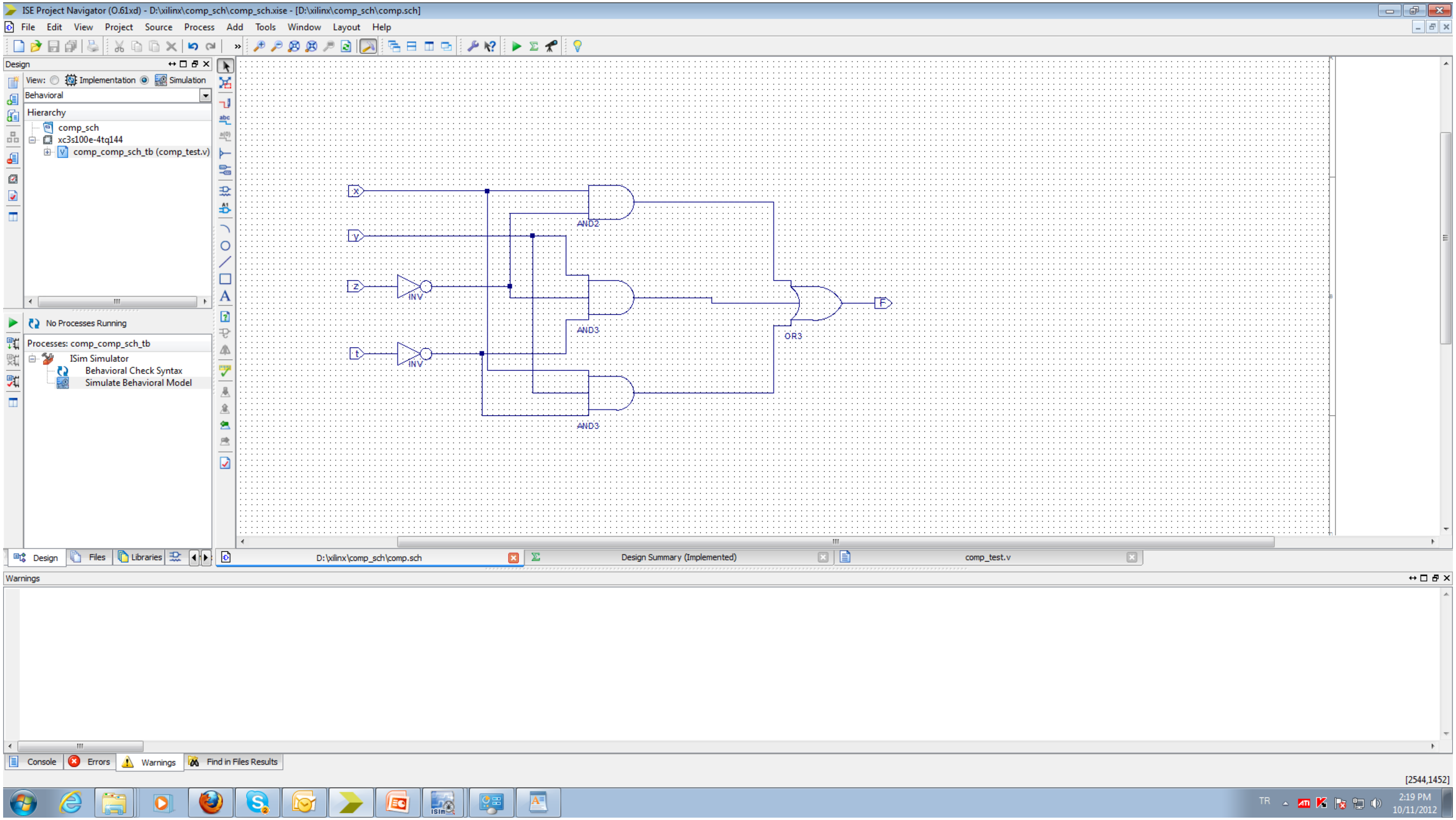
Schematic Editor

- 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

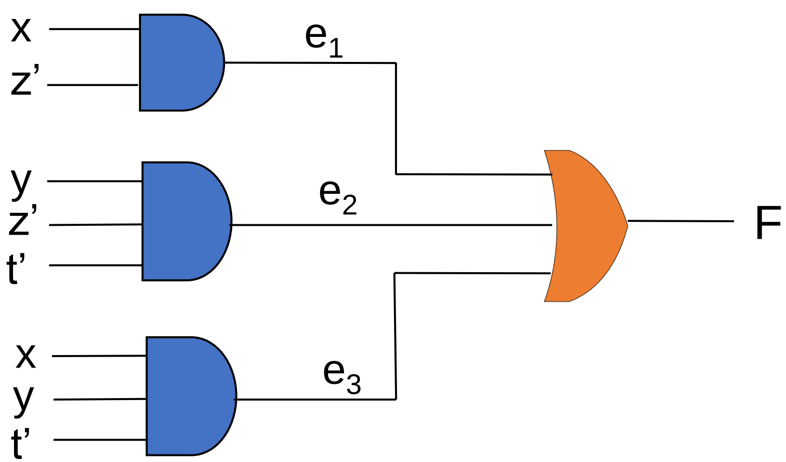


A Schematic Design



Hardware Description Languages

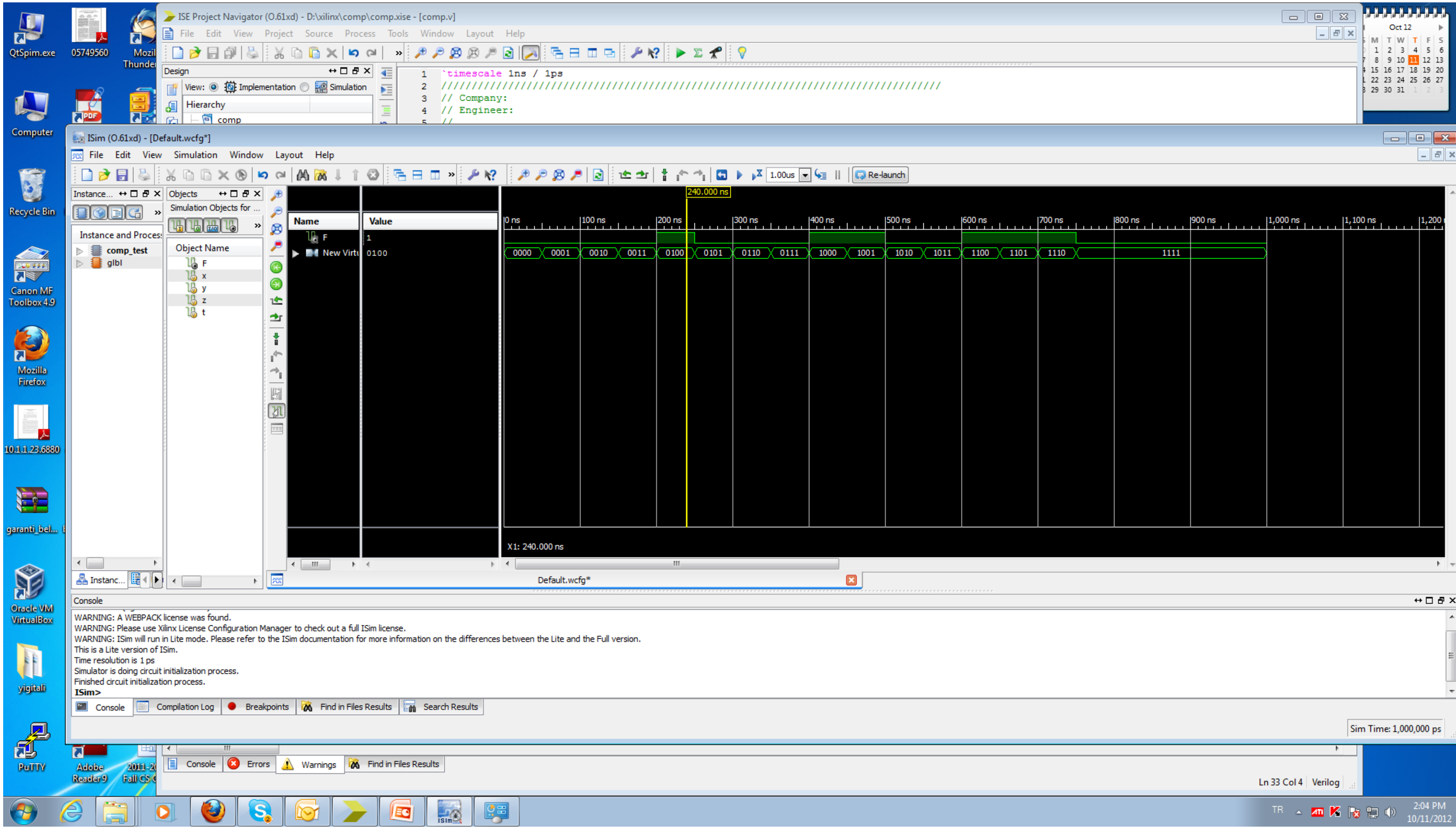
- HDL
  - Verilog, VHDL
  - resembles a programming language
  - designed to describe digital circuits so that we can develop and test digital circuits



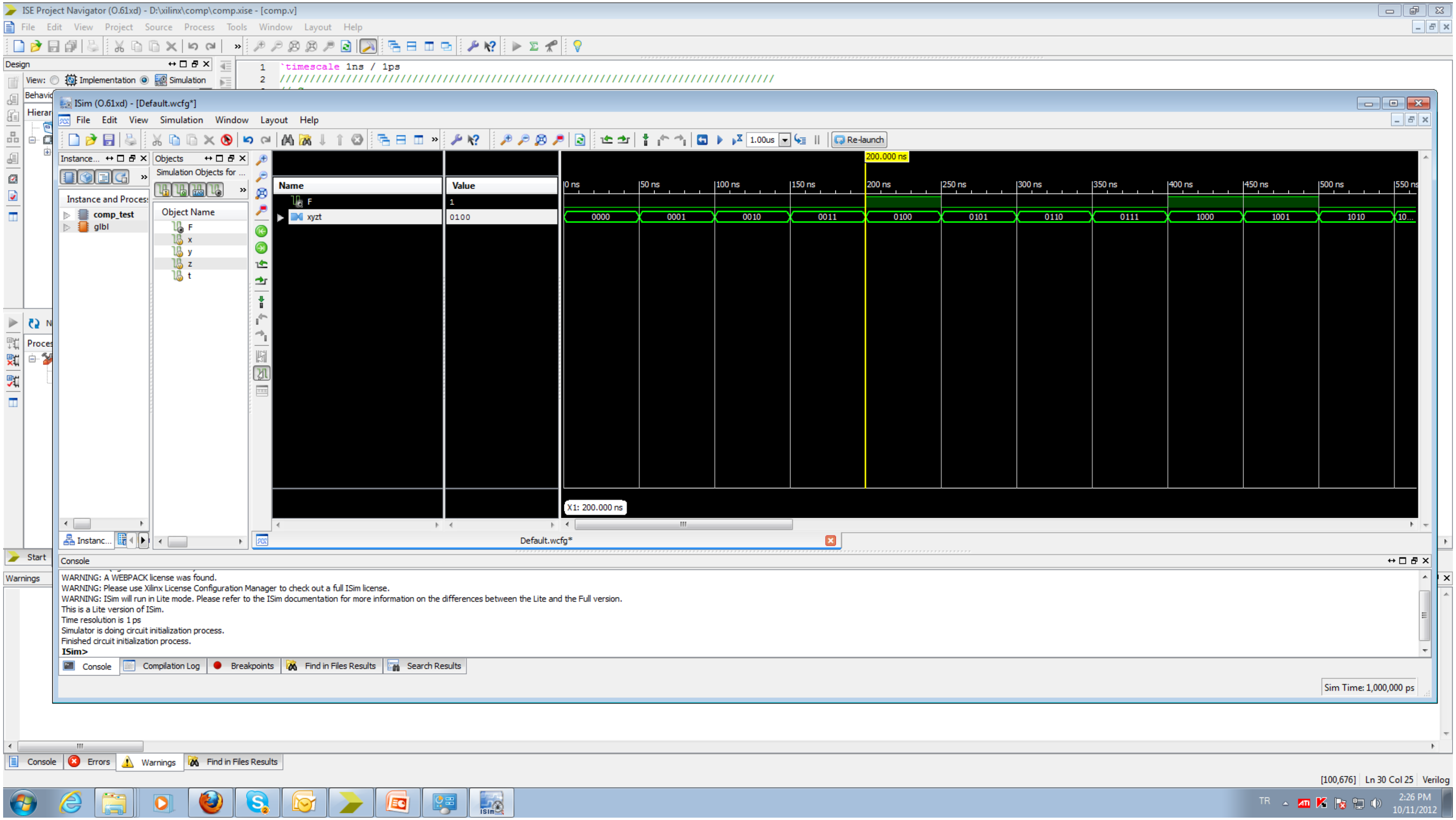

```
module comp(F, x, y, z, t);
    input x, y, z, t;
    output F;
    wire e1, e2, e3;
    and g1(e1, x, ~z);
    and g2(e2, y, ~z, ~t);
    and g3(e3, x, y, ~t);
    or g4(F, e1, e2, e3);
endmodule
```

F (x,y,z,t) = xz' + yz't' + xyt'

Simulation Results 1/3



Simulation Results 2/3



Simulation Results 3/3