Boolean Algebra

section instructor:
Ufuk Çelikcan

Axioms of Algebra 1/2

1. Closure:
   - A set is closed w.r.t. a binary operation *
   - Is the set of natural numbers 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 an identity element “e” if $\forall\, x \in S$, $e * x = x * e = x$.
   - <u>Set of integers</u>: $e = 1$ w.r.t. $\times$ and $e = 0$ w.r.t. $+$

Axioms of Algebra 2/2

5. Inverse
   - S having an identity element "e" w.r.t. * is said to have an inverse $\forall$ x $\in$ S, whenever there exists an element y $\in$ S such that
     $x * y = e$
   - <u>Example</u>: set of integers w.r.t. +
6. Distributive law

✓If * and • are two binary operators on S
   >> * is said to be distributive over • whenever

✓$x * (y \bullet z) = (x * y) \bullet (x * z)$

Boolean Algebra

- 1854: George Boole:
  - *Boolean Algebra*
- 1904: E. V. Huntington:
  - Formal definition of Boolean Algebra
- 1938: Claude E. Shannon:
  - *Switching Algebra*

Boolean Algebra 1/2

- 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 ·

  $x, y \in B, (x+y) \in B, (x \cdot y) \in B$

  additive identity ?

  $0: \ x + 0 = 0 + x = x$

  multiplicative identity ?

  $1: \ x \cdot 1 = 1 \cdot x = x$

  commutative w.r.t. both + and ·

  associative w.r.t. both + and ·
- Distributive law:

  is · distributive over + ?

  yes : $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$

  is + distributive over · ?

  yes : $x+(y \cdot z) = (x+y) \cdot (x+z)$

  We do not have the second one in ordinary algebra

Boolean Algebra 2/2

- Complement
  - $\forall$ x $\in$ B, there exist an element x’ $\in$ B ∍
    a. x + x’ = 1 (multiplicative identity) and
    b. x · x’ = 0 (additive identity)
  - Not available in ordinary algebra
- No inverses in Boolean Algebra!
- Differences between ordinary and Boolean algebra
  - Ordinary algebra deals with real numbers (infinite)
  - Boolean algebra deals with elements of set B (finite)
  - 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	$x \cdot y$
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

$y \cdot z$	$x+(y \cdot z)$

$x + y$	$x + z$	$(x + y) \cdot (x + z)$

Two-Valued Boolean Algebra 3/3

- Two-valued Boolean algebra is actually equivalent to the binary logic that we defined heuristically before

  Operations:

    - · → AND
    - + → OR
    - Complement → NOT

- <u>Binary logic is the application of Boolean algebra to 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 together
- Example:
  - $x + x = x$
  - $x + x = (x+x) \cdot 1$ (identity element)
    $= (x+x)(x+x')$ (complement)
    $= x+(x \cdot x')$ (+ over ·)
    $= x$ (complement)
  - duality principle
    $x + x = x \Rightarrow x \cdot x = x$

Duality Principle & Theorems

- Theorem a:
  - $x + 1 = 1$ hmmm?
  - $$\begin{aligned} x + 1 &= 1 \cdot (x + 1) \\ &= (x + x')(x + 1) \\ &= x + x' \cdot 1 \\ &= x + x' \\ &= 1 \end{aligned}$$
- Theorem b: (using duality)
  - $x . 0 = 0$

Absorption Theorem

$x + xy = x$ hmmmmm?

$= x.1+xy$

$= x(1+y)$

$= x(1)$

$= x$

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:

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

b. From duality ? **$(x.y)'=x'+y'$**

Truth Tables for DeMorgan's Theorem

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

x	y	$x+y$	$(x+y)'$	$x \cdot y$	$(x \cdot y)'$
0	0				
0	1				
1	0				
1	1				

x'	y'	$x' \cdot y'$	$x' + y'$
1	1		
1	0		
0	1		
0	0		

Operator Precedence (Boolean Operator Priority Order)

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

PiNAOr?

PiNA c**O**lada?

Boolean Functions

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

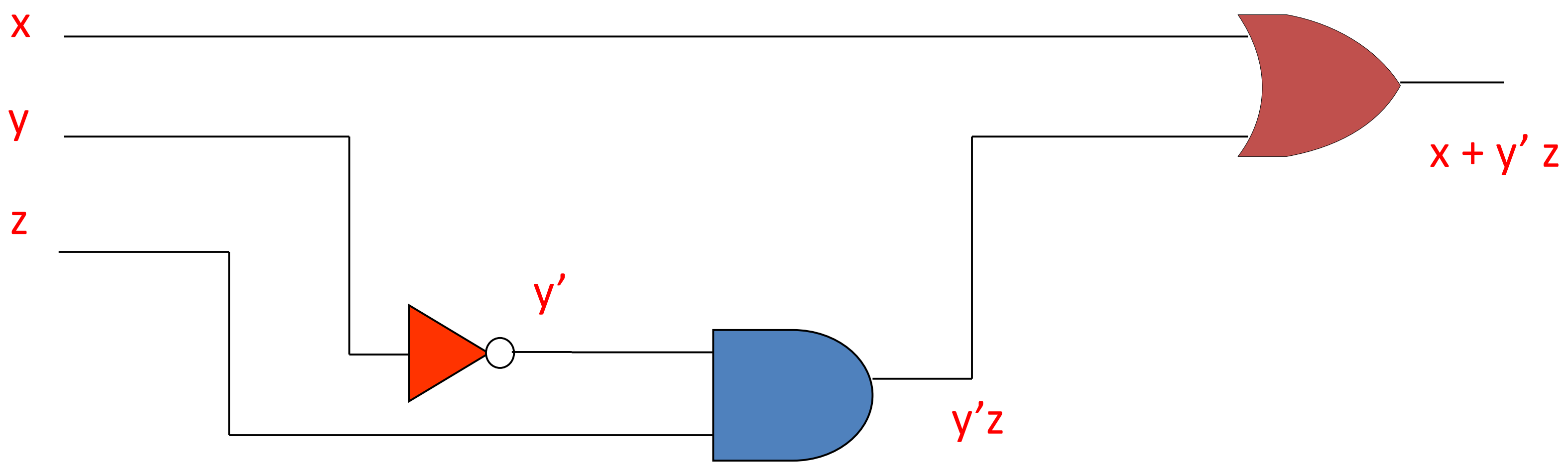
x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Logic Circuit Diagram of F_1

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

Logic Circuit Diagram of F_1

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



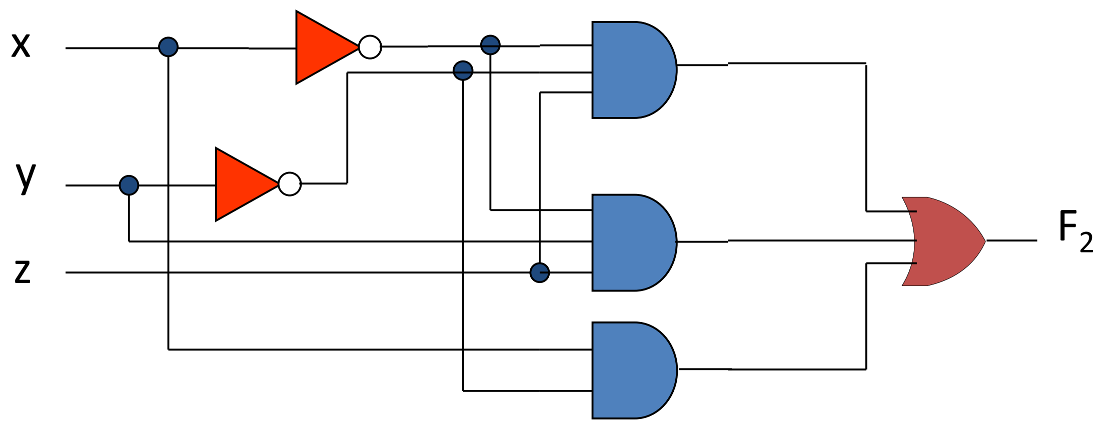
Gate Implementation of $F_1 = x + y' z$

Logic Circuit Diagram of F_2

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

Logic Circuit Diagram of F_2

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



Algebraic manipulation

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

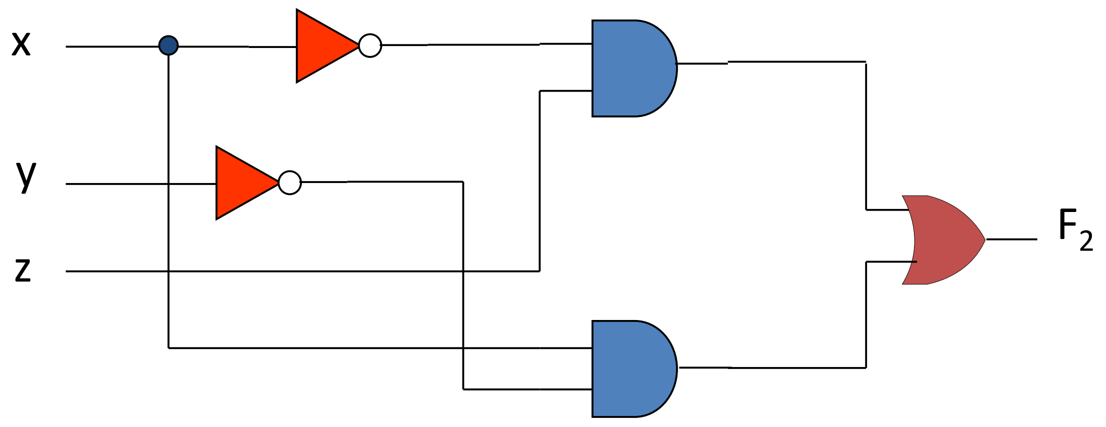
$= x'z(y'+y) + xy'$

Alternative Implementations of F_2

$F_2 = x' z + xy'$

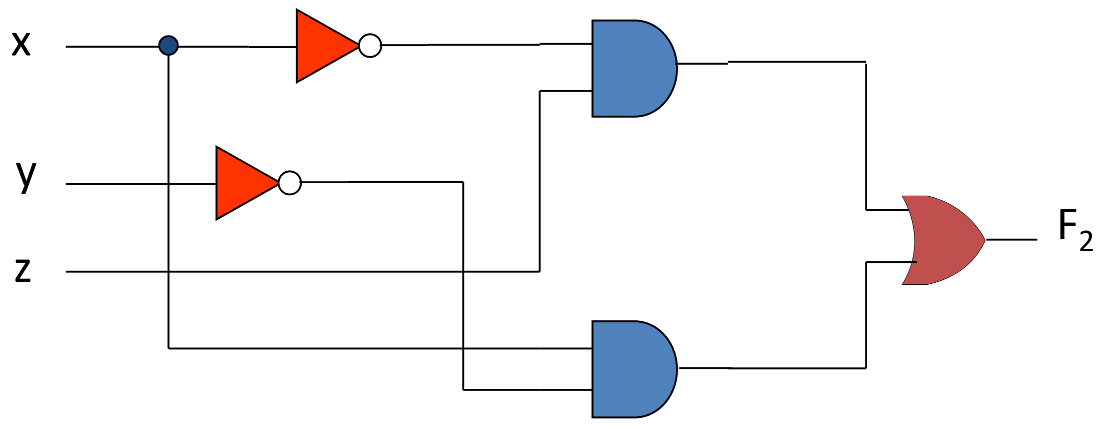
Alternative Implementations of F_2

$F_2 = x' z + xy'$

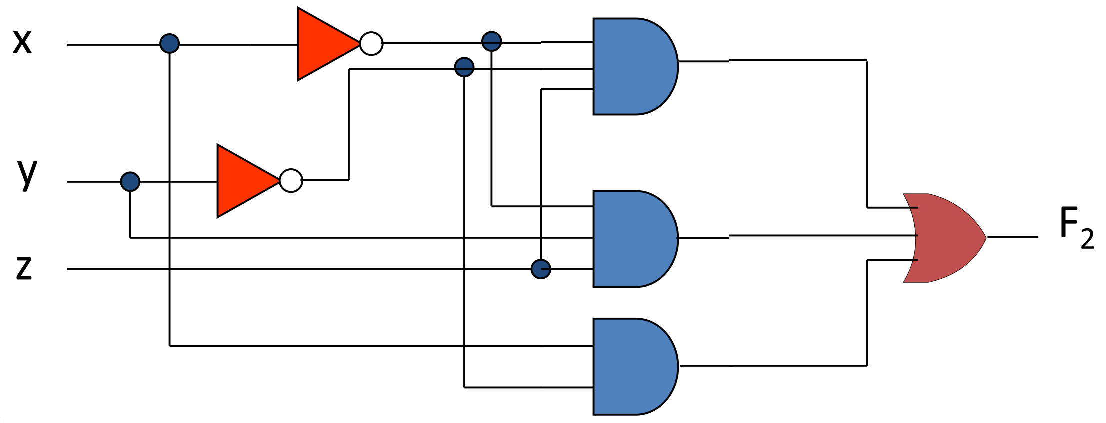


Alternative Implementations of F_2

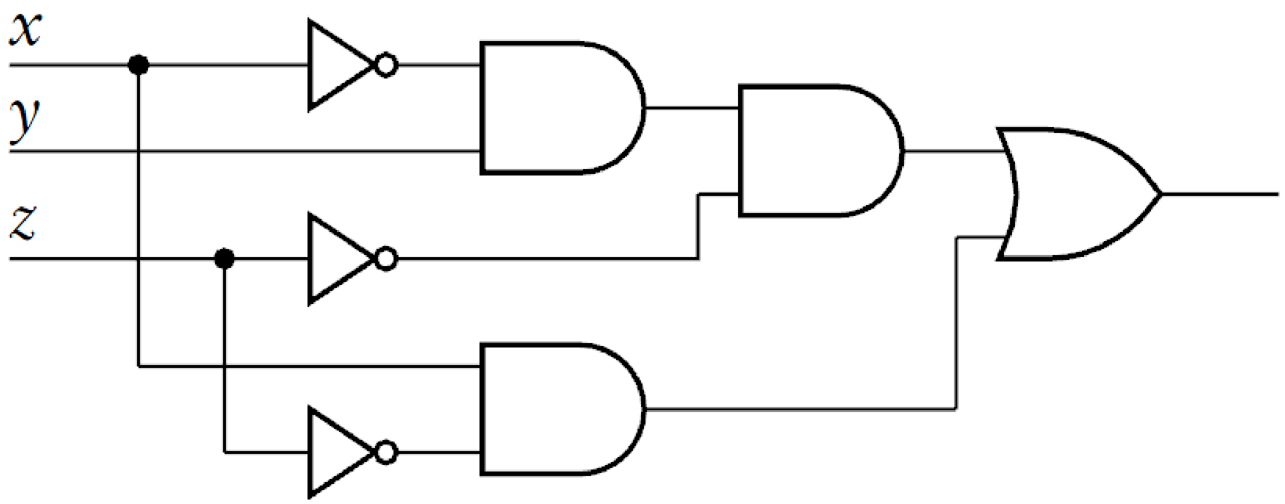
$F_2 = x' z + xy'$



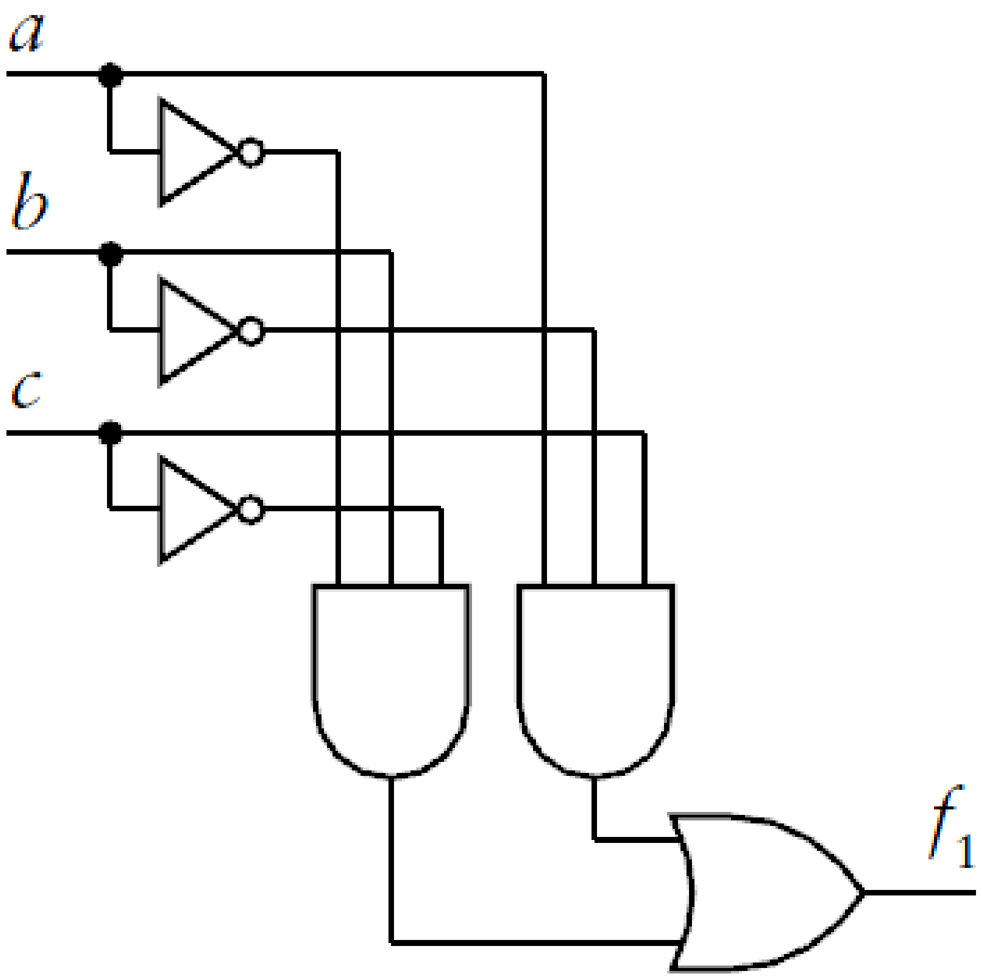
$F_2 = x' y' z + x' y z + xy'$



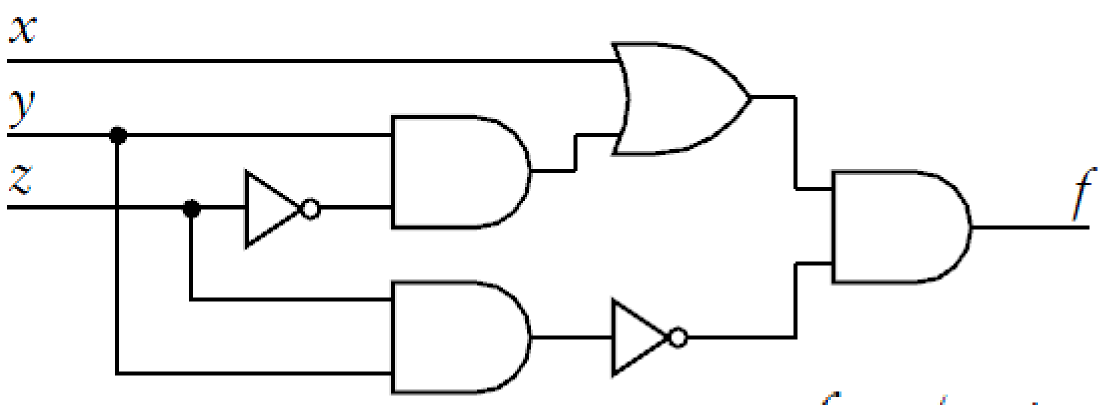
Example



Example

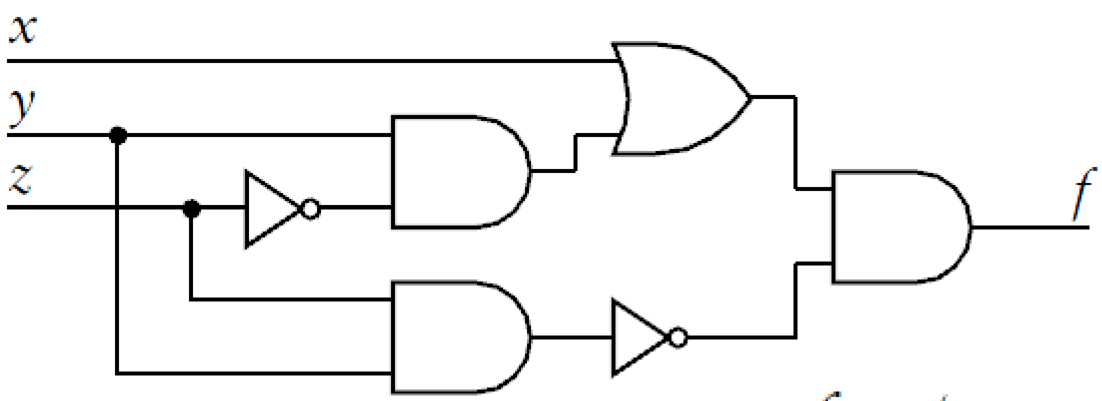


Example



$$f = (x + y \cdot \overline{z}) \cdot \overline{(y \cdot z)}$$

Example



$$
\begin{aligned}
f &= (x + y \cdot \overline{z}) \cdot \overline{(y \cdot z)} \\
&= (x + y \cdot \overline{z}) \cdot (\overline{y} + \overline{z}) \\
&= x \cdot (\overline{y} + \overline{z}) + (y \cdot \overline{z}) \cdot (\overline{y} + \overline{z}) \\
&= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{y} + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 + y \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z}
\end{aligned}
$$

OTHER LOGIC OPERATORS - 1

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

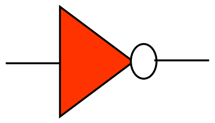
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

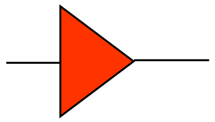
x	y	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

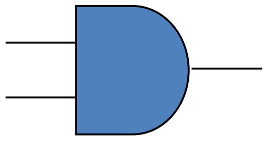
OTHER LOGIC OPERATORS - 2

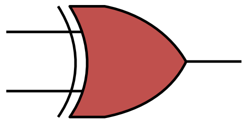
- Some of the Boolean functions with two variables
  - Constant functions: $F_0 = 0$ and $F_{15} = 1$
  - AND function: $F_1 = xy$
  - OR function: $F_7 = x + y$
  - XOR function:
    - $F_6 = x'y + xy' = x \oplus y$ **(x or y, but not both)**
  - XNOR (Equivalence) function:
    - $F_9 = xy + x'y' = (x \oplus y)'$ **(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)

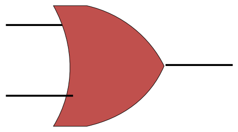
Logic Gate Symbols

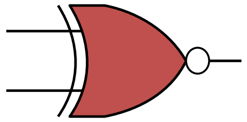

NOT

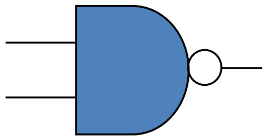

TRANSFER

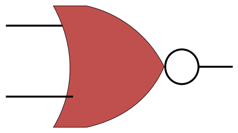

AND


XOR


OR


XNOR


NAND

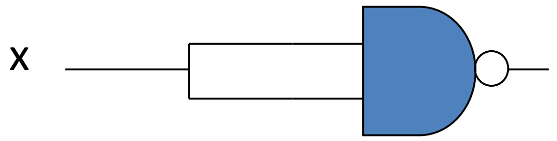

NOR

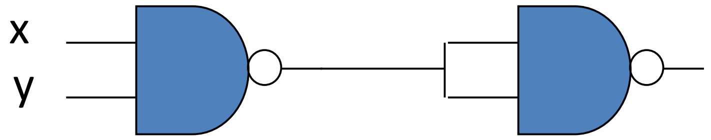
Universal Gates

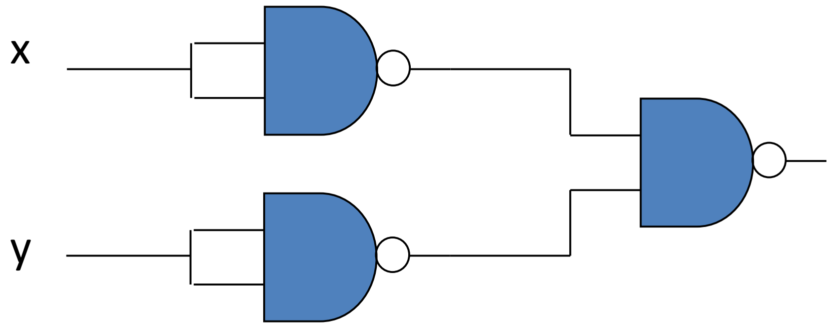
- **NAND and NOR gates are <u>universal</u>**
- We know <u>any</u> Boolean function can be written in terms of three logic operations:
  - AND, OR, NOT
- In return, **NAND** gate can implement these three logic gates by itself
  - **So can NOR gate**

x	y	(xy)’	x’	y ’	(x’ y’)’
0	0	1	1	1	
0	1	1	1	0	
1	0	1	0	1	
1	1	0	0	0	

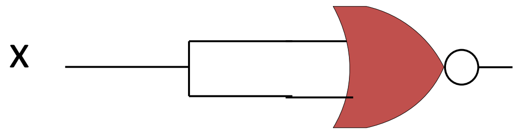
NAND Gate

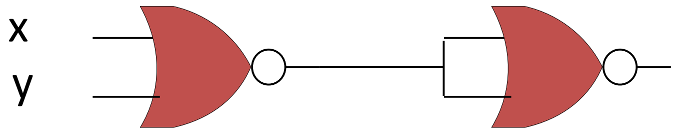


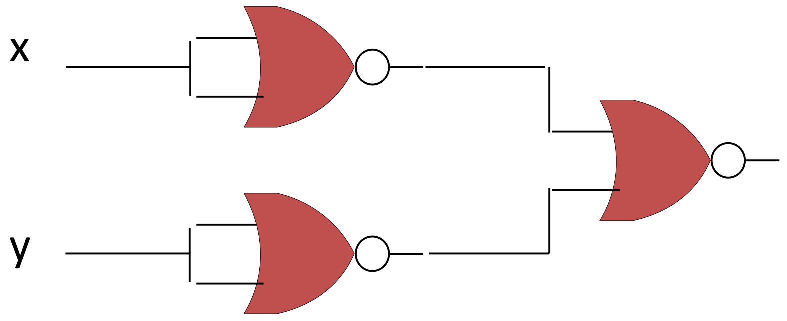




NOR Gate



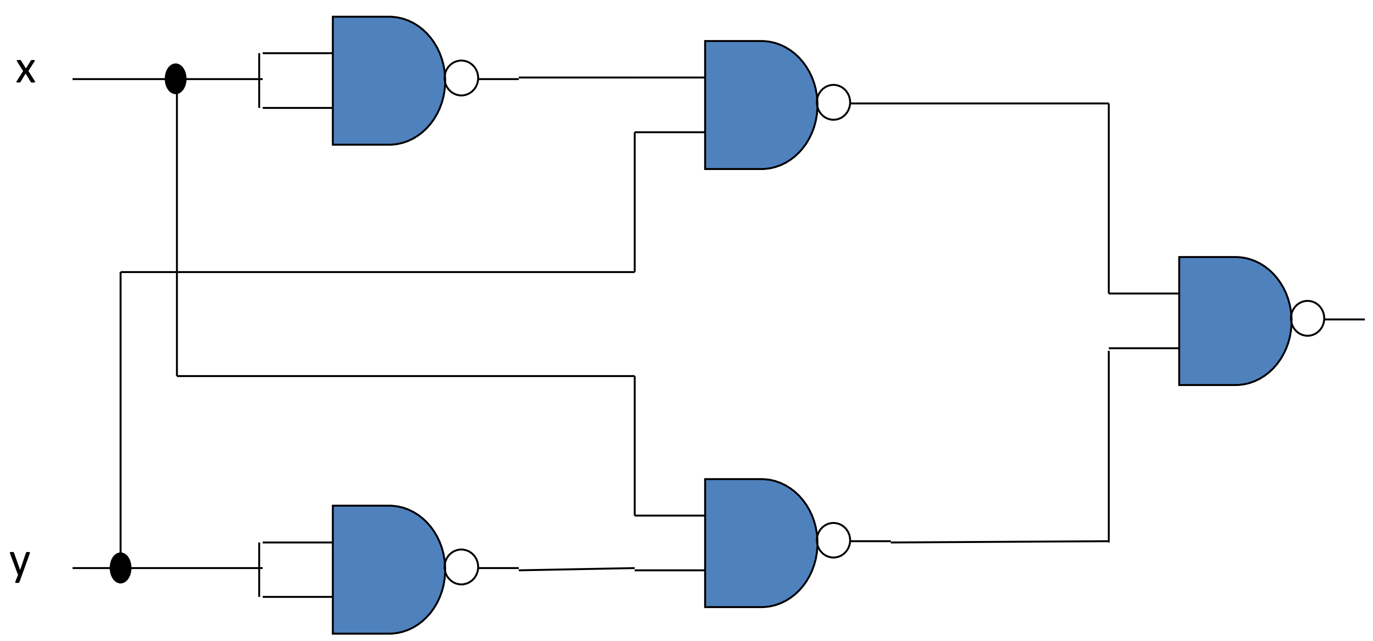




Designs with NAND gates Example 1/2

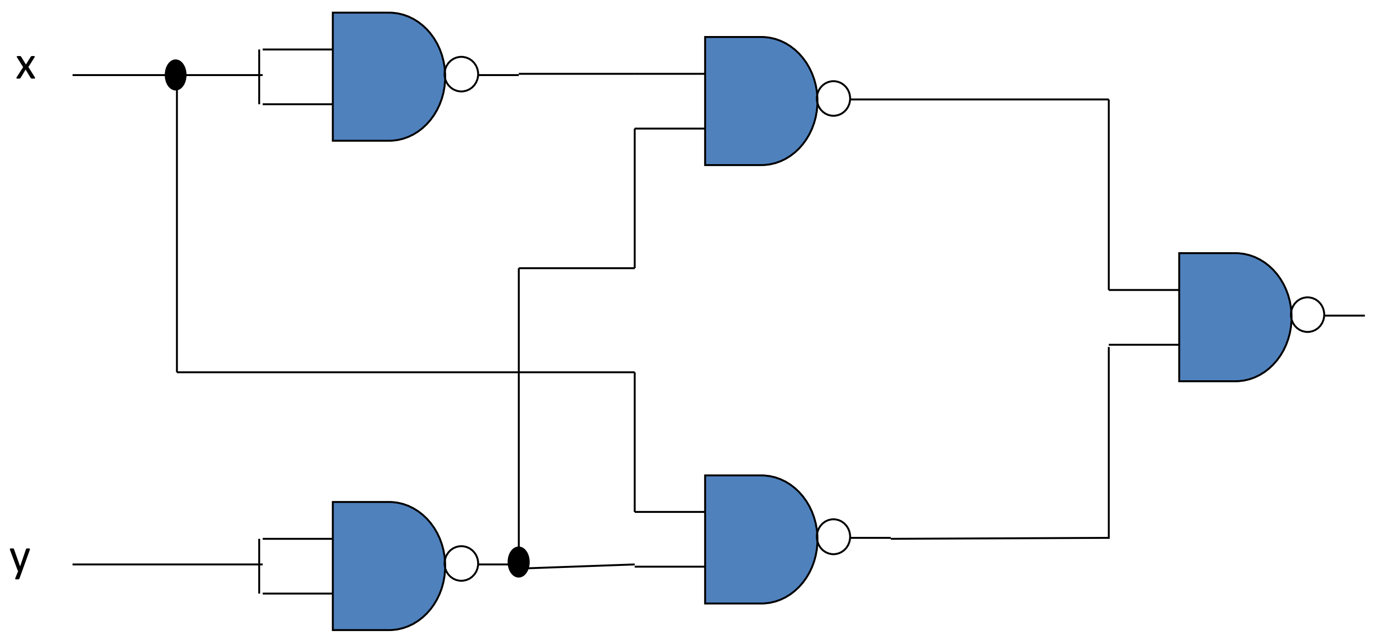
- A function:

  $F_1 = x' y + xy'$



Example 2/2

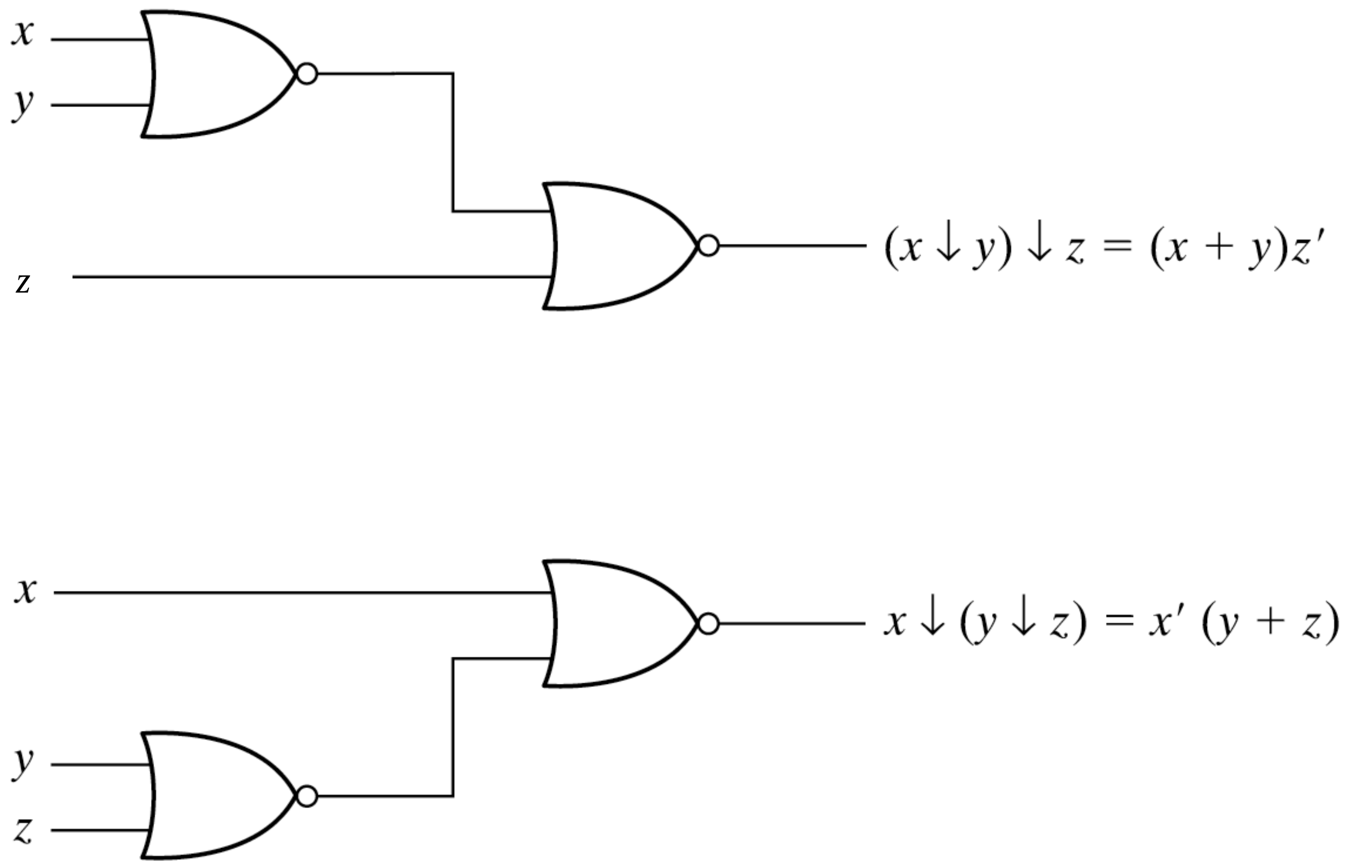
$F_2 = x' y' + xy'$



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 not associative
  - Extending the number of inputs is not obvious
- Example: NAND gates
  - $((xy)'z)' \neq (x(yz)')'$
  - $((xy)'z)' = xy + z'$
  - $(x(yz)')' = x' + yz$

Nonassociativity of NOR operation



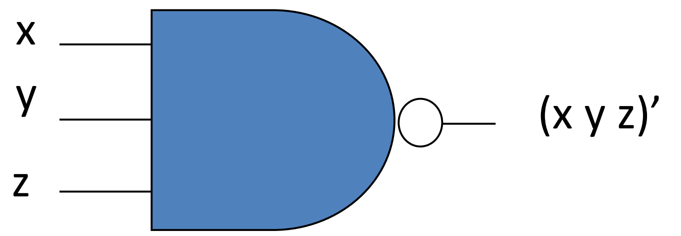
x
y
z
$(x \downarrow y) \downarrow z = (x + y)z'$
x
y
z
$x \downarrow (y \downarrow z) = x' (y + z)$

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

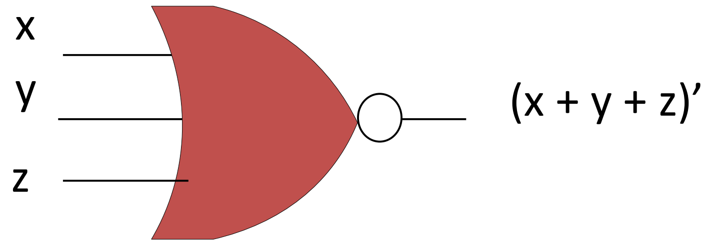
Multiple Input Universal Gates

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

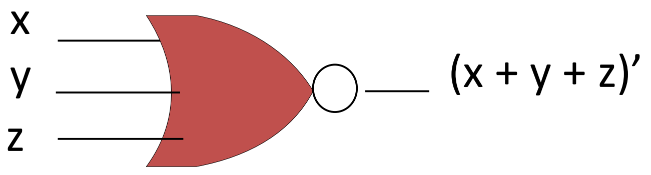
Three input NAND gate: (x y z)'



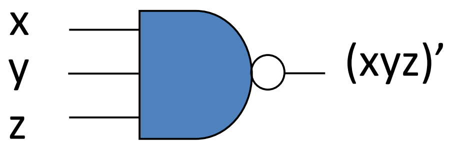
Three input NOR gate:(x + y + z)'



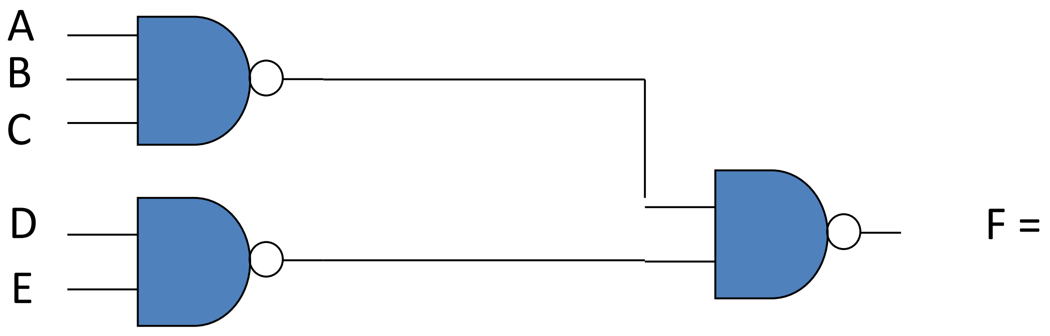
Multiple Input Universal Gates



3-input NOR gate



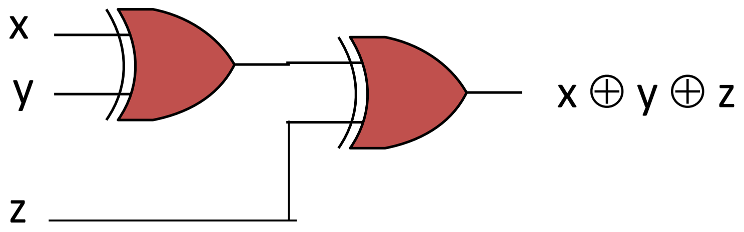
3-input NAND gate



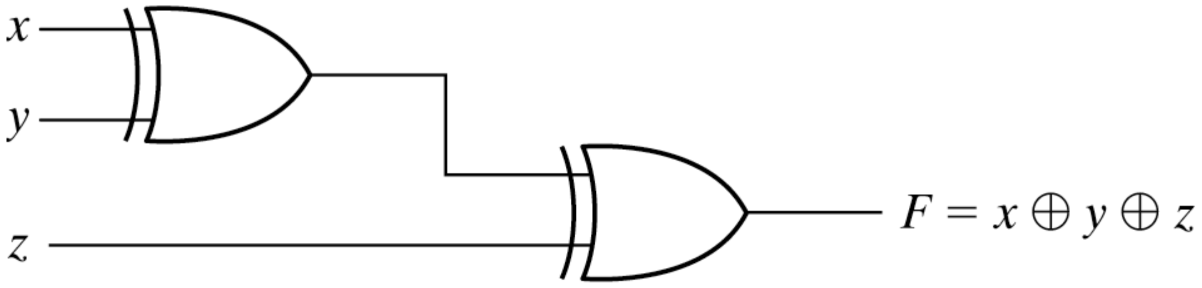
Cascaded NAND gates

XOR and XNOR Gates

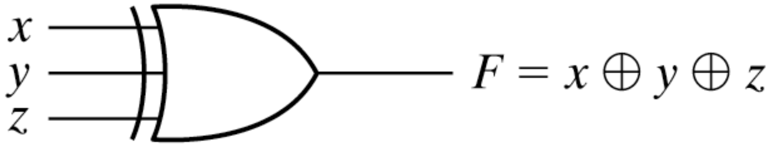
- XOR and XNOR operations are both commutative and associative.
- No problem manufacturing multiple input XOR and XNOR gates
- However, they are more **costly from hardware point of view.**
- Therefore, we usually have 2-input XOR and XNOR gates



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

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	
0	0	1	0	
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	0	
1	1	1	0	

F =

F’ =

Generalizing DeMorgan's Theorem

- We can also utilize DeMorgan's Theorem
  - $(x + y)' = x' y'$
  - $(A + B + C)'$

    $= (A+B)'C'$

    $= A'B'C'$
- **We can generalize DeMorgan's Theorem**
  - $(x_1 + x_2 + \ldots + x_N)' = x_1' \cdot x_2' \cdot \ldots \cdot x_N'$
  - $(x_1 \cdot x_2 \cdot \ldots \cdot x_N)' = x_1' + x_2' + \ldots + x_N'$

Example: Complement of a Function

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

  - $F_2 = x(y'z' + yz)$
  - $F_2' = (x(y'z' + yz))'$
    $= x'+(y'z' + yz)'$
    $= x' + (y + z) (y' + z')$
- Easy Way to Complement: take the dual of the function and complement each literal

Canonical & Standard Forms

- Minterms
  - A product term: all variables appear either in its normal form (x) or its complement form (x')
  - How many different terms we can get with x and y?
    - x'y' $\rightarrow$ 00 $\rightarrow$ m_0
    - x'y $\rightarrow$ 01 $\rightarrow$ m_1
    - xy' $\rightarrow$ 10 $\rightarrow$ m_2
    - xy $\rightarrow$ 11 $\rightarrow$ 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) = ? in minters

F(x, y, z) = ? in maxterms

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

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

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

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

Formal Expression with Minterms

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

$$F(x, y, z) = F(0,0,0)m_0 + F(0,0,1)m_1 + F(0,1,0)m_2 + F(0,1,1)m_3 + F(1,0,0)m_4 + F(1,0,1)m_5 + F(1,1,0)m_6 + F(1,1,1)m_7$$

Formal Expression with Maxterms

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

$$F(x, y, z) = (F(0,0,0)+M_0)\,(F(0,0,1)+M_1)\,(F(0,1,0)+M_2)\,(F(0,1,0)+M_3)$$
$$(F(1,0,0)+M_4)\,(F(1,0,1)+M_5)\,(F(1,1,0)+M_6)\,(F(1,1,1)+M_7)$$

Boolean Functions in Canonical 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

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

Important Properties

- Any Boolean function can be expressed
  - as **<u>a sum of minterms</u>**
  - as **<u>a product of maxterms</u>**
- Example:
  - F’ = Σ (0, 2, 3, 5, 6)

    =
  - How do we find the complement of F’?
  - F =

Important Properties

- Any Boolean function can be expressed
  - as **a sum of minterms**
  - as **a product of maxterms**
- Example:
  - F’ = Σ (0, 2, 3, 5, 6)

    = x’y’z’ + x’yz’ + x’yz + xy’z + xyz’
  - How do we find the complement of F’?
  - F =

Important Properties

- Any Boolean function can be expressed
  - as **a sum of minterms**
  - as **a product of maxterms**
- Example:
  - F’ = Σ (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)

    =

    =

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 = Σ (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 ·
  - $F = xy + x'z$
    $= xy(z+z') + x'z(y+y')$
    $= xyz + xyz' + x'yz + x'y'z$
    $= \Sigma\,(7,6,3,1)$

Example: Product of Maxterms

- $F = xy + x'z$
  - Use the distributive law + over ·
  - F $= xy + x'z$
    $= xy(z+z') + x'z(y+y')$
    $= xyz + xyz' + x'yz + x'y'z$
    $= \Sigma\ (7,6,3,1)$

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

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
  - $(F')' \quad = \qquad\qquad\qquad =$
  - $F \quad = M_0 \cdot M_2 \cdot M_3 = \Pi(0, 2, 3)$

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

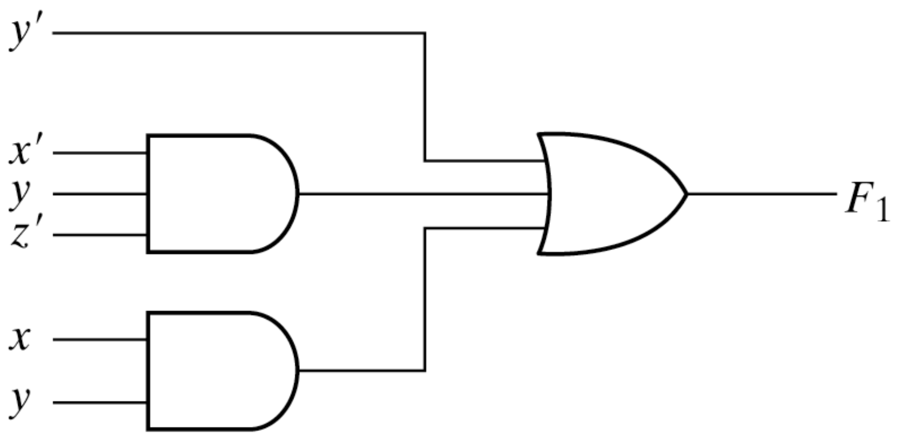
- Fact:
  - Canonical forms are very seldom the ones with the least number of literals
- Alternative representation:
  - Standard form
    - a <u>term</u> may contain any number of literals
  - Two types
    1. the sum of products
    2. the product of sums
  - Examples:
    - $F_1 = y' + xy + x'yz'$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

Example: Standard Forms

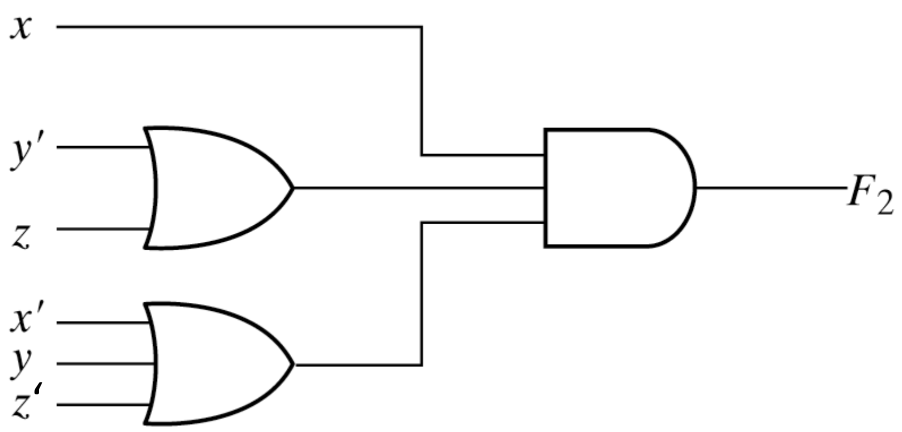
- $F_1 = y' + xy + x'yz'$
- $F_2 = x(y' + z)(x' + y + z')$

Example: Standard Forms

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



(a) Sum of Products

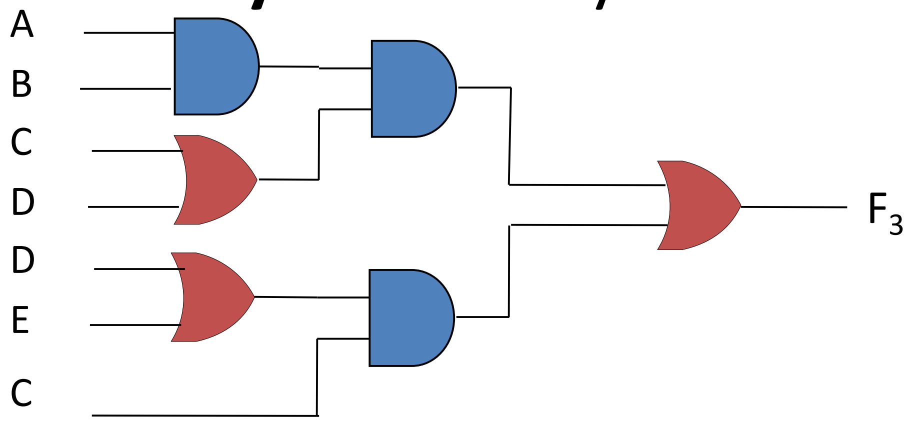


(b) Product of Sums

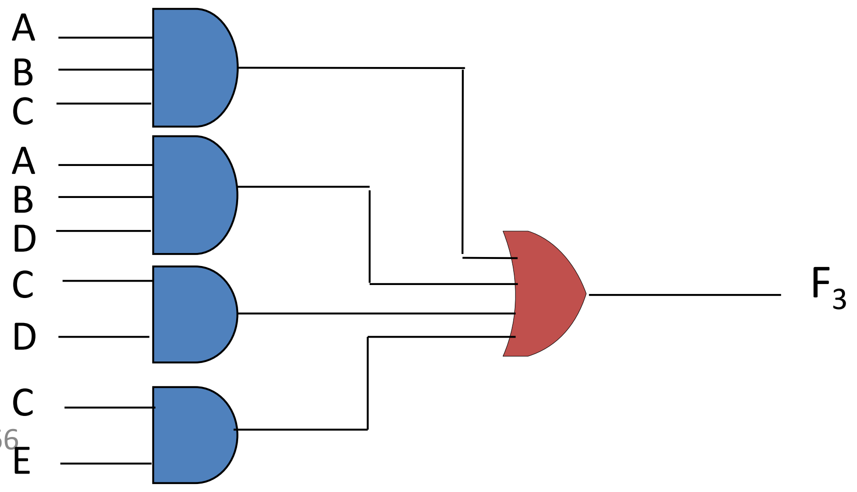
Fig. 2-3 Two-level implementation

Nonstandard Forms

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



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



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

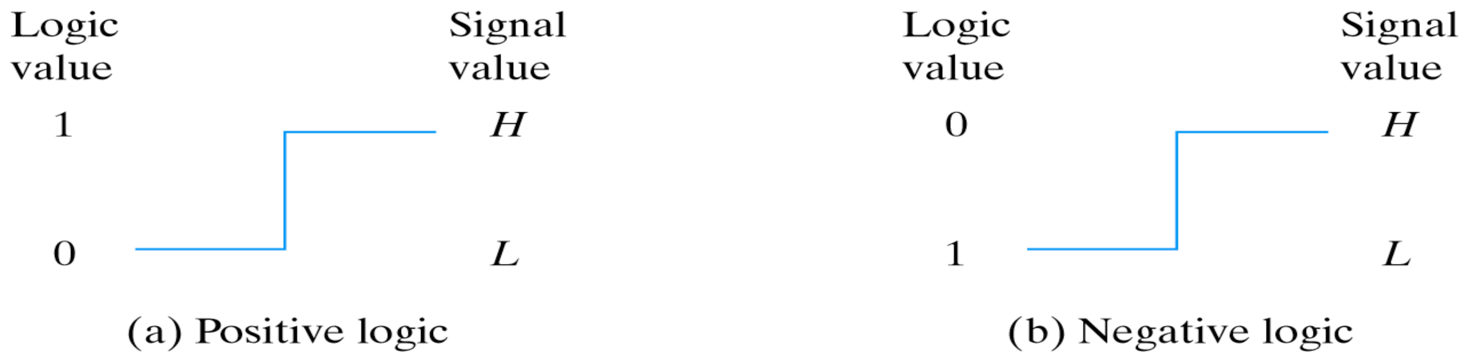
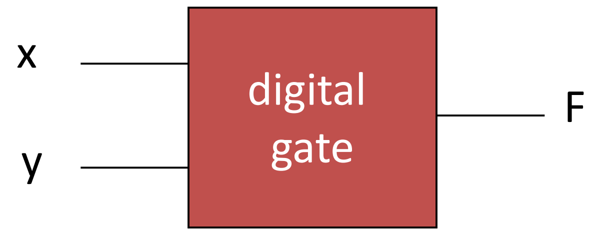


Fig. 2-9 signal assignment and logic polarity

Signal Designation - 1

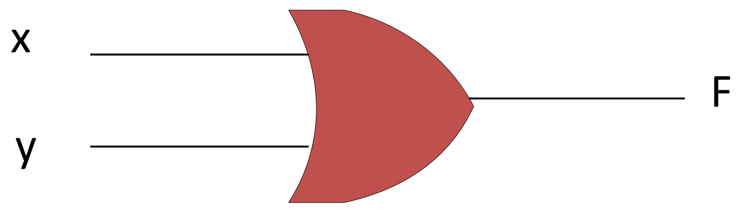


x	y	F
L	L	L
L	H	H
H	L	H
H	H	H

- What kind of logic function does it implement?

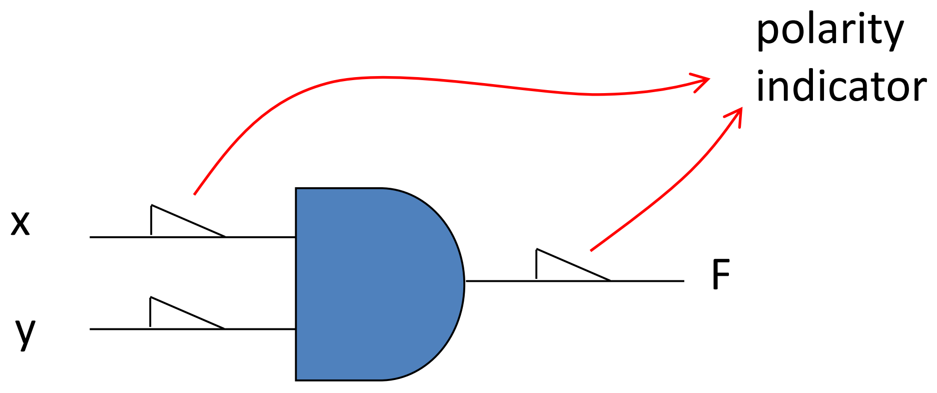
Signal Designation - 2

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

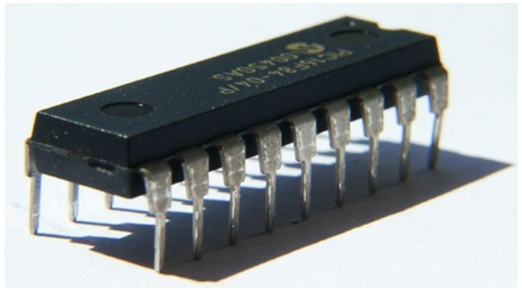
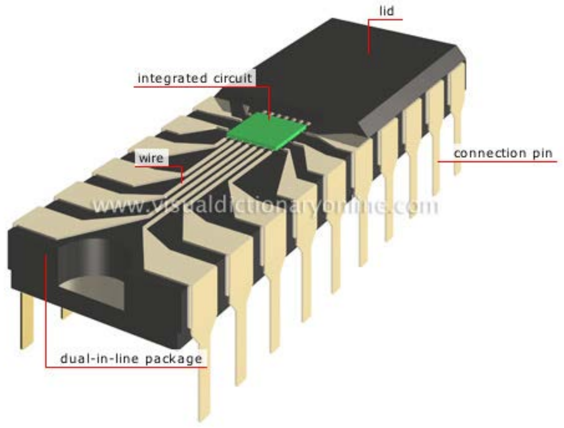
Another Example

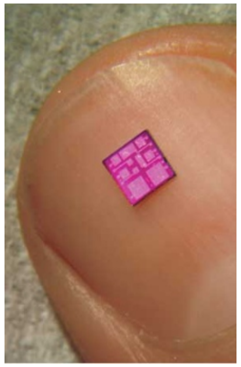
x	y	F
L	L	H
L	H	H
H	L	H
H	H	L

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

904 million transistors

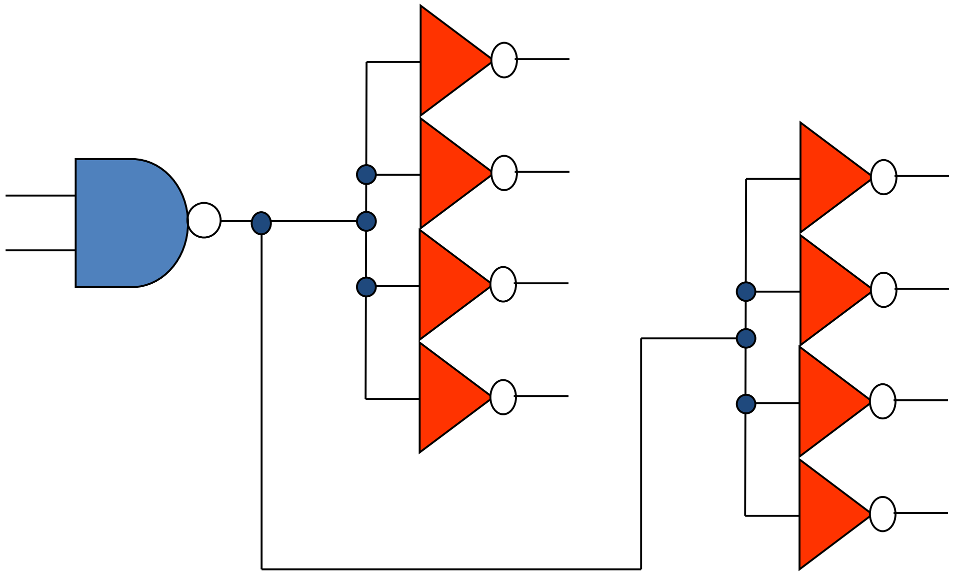
- SSI (small-scale integration):
  - inputs and outputs of the gates are connected directly to the pins in the package.
  - The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.
- MSI (medium-scale integration):
  - From 10 to 1,000 gates per chip
  - usually perform specific elementary digital operations.
  - Usual MSI digital functions: decoders, adders, multiplexers, registers and counters.
- LSI (large-scale integration):
  - 1,000s of gates per chip
  - include digital systems such as processors, memory chips, and programmable logic devices.
- VLSI (very large-scale integration) and ULSI (ultra large-scale integration):
  - devices now contain millions of gates within a single package.
  - Examples are large memory arrays and complex microcomputer chips.
  - Because of their small size and low cost, VLSI devices have revolutionized the computer system design technology, giving the designer the capability to create structures that were previously uneconomical to build.

Digital Logic Families

- Circuit Technologies
  - TTL → transistor-transistor logic
    - has been in use for 50 years and is considered to be standard.
  - ECL → Emitter-coupled logic
    - Fast. Has an advantage in systems requiring **high-speed operation**.
  - MOS → metal-oxide semiconductor
    - suitable for circuits that need **high component density**,
  - CMOS → Complementary MOS
    - preferable in systems requiring **low power consumption**, such as digital cameras, personal media players, and other handheld portable devices.
  - Low power consumption is essential for VLSI design; therefore, CMOS has become the dominant logic family, while TTL and ECL continue to decline in use.

Parameters of Logic Gates - 1

- **Fan-out**
  - Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation.
  - A **standard load** is usually defined as the amount of current needed by an input of another similar gate in the same family.

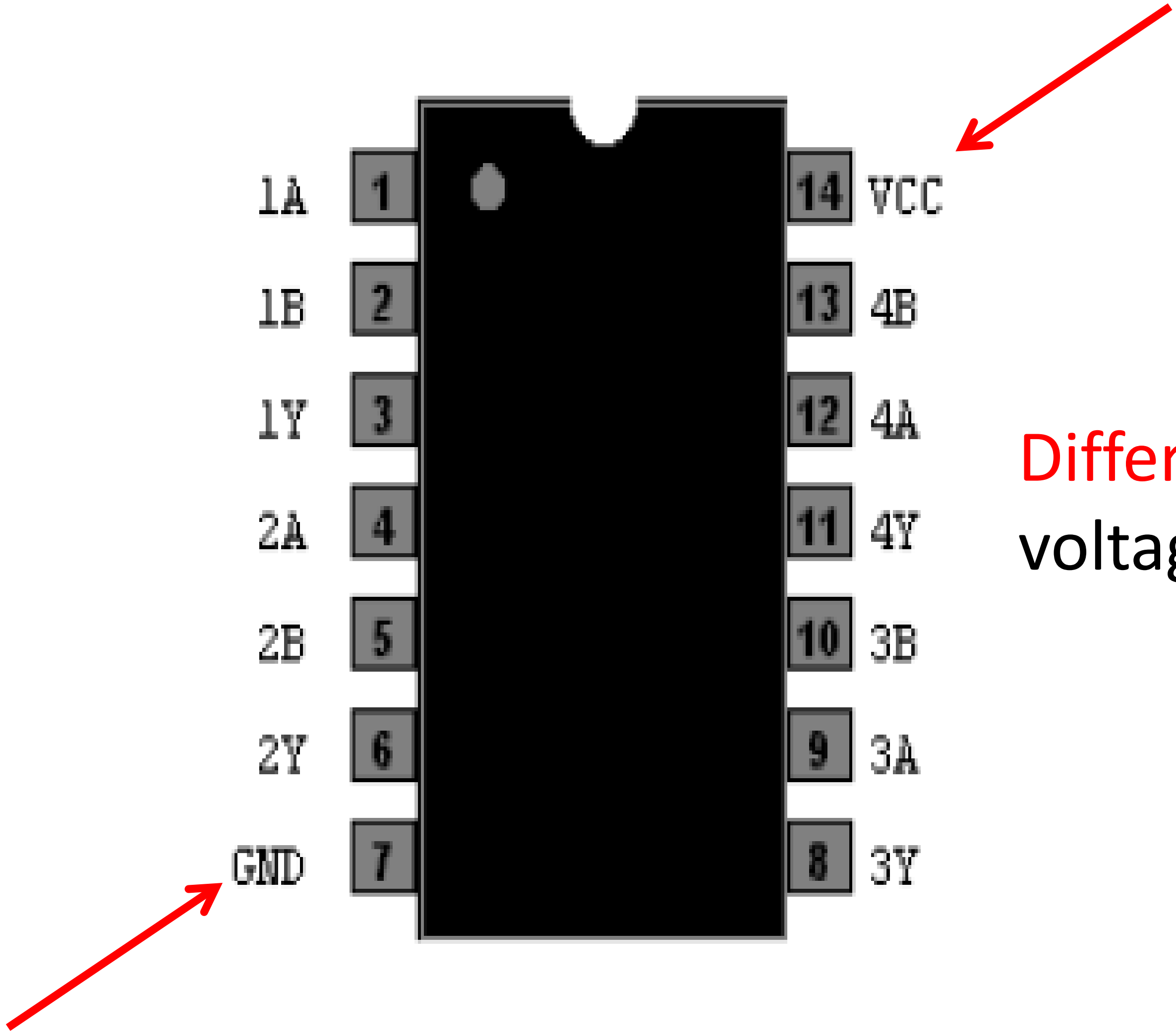


- If a, say NAND, gate drives four such inverters, then the fan-out is equal to 4.0 standard loads.

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 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

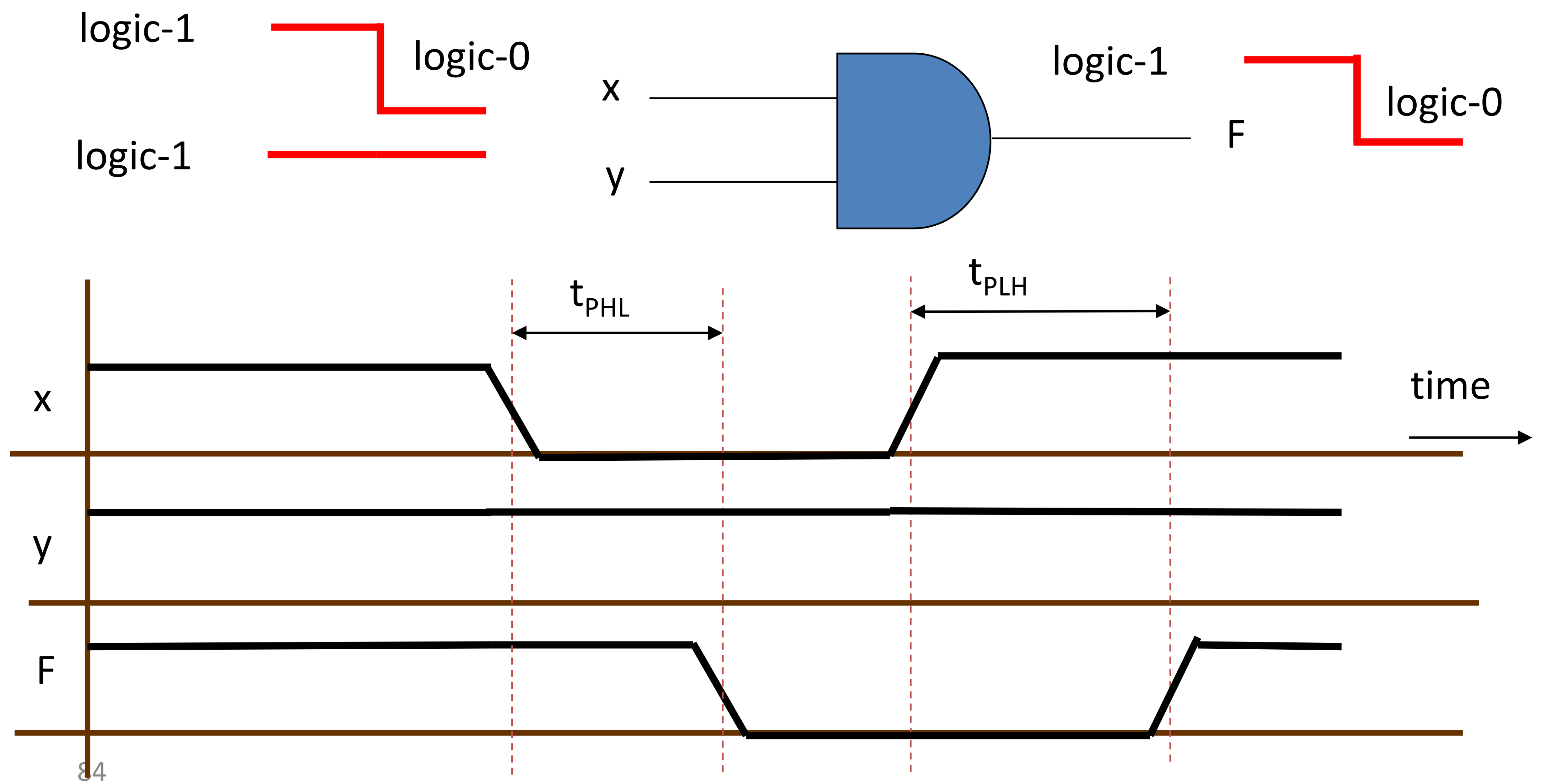
Power Dissipation



Difference in voltage level !

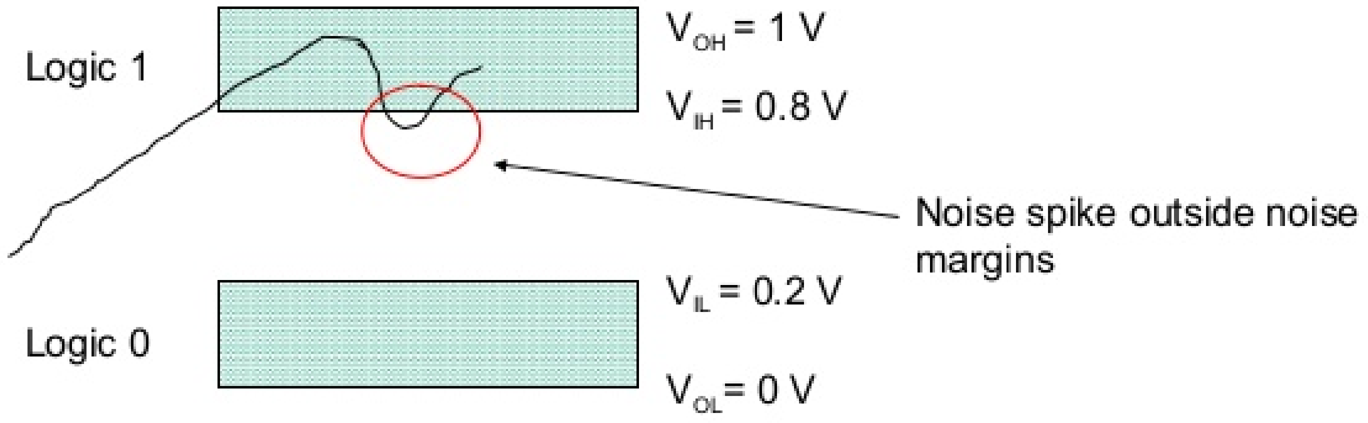
Parameters of Logic Gates - 3

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



Parameters of Logic Gates - 4

- ***Noise margin***
  - the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.



Noise tolerance is the degree to which a gate is impervious to noise spikes at its input.

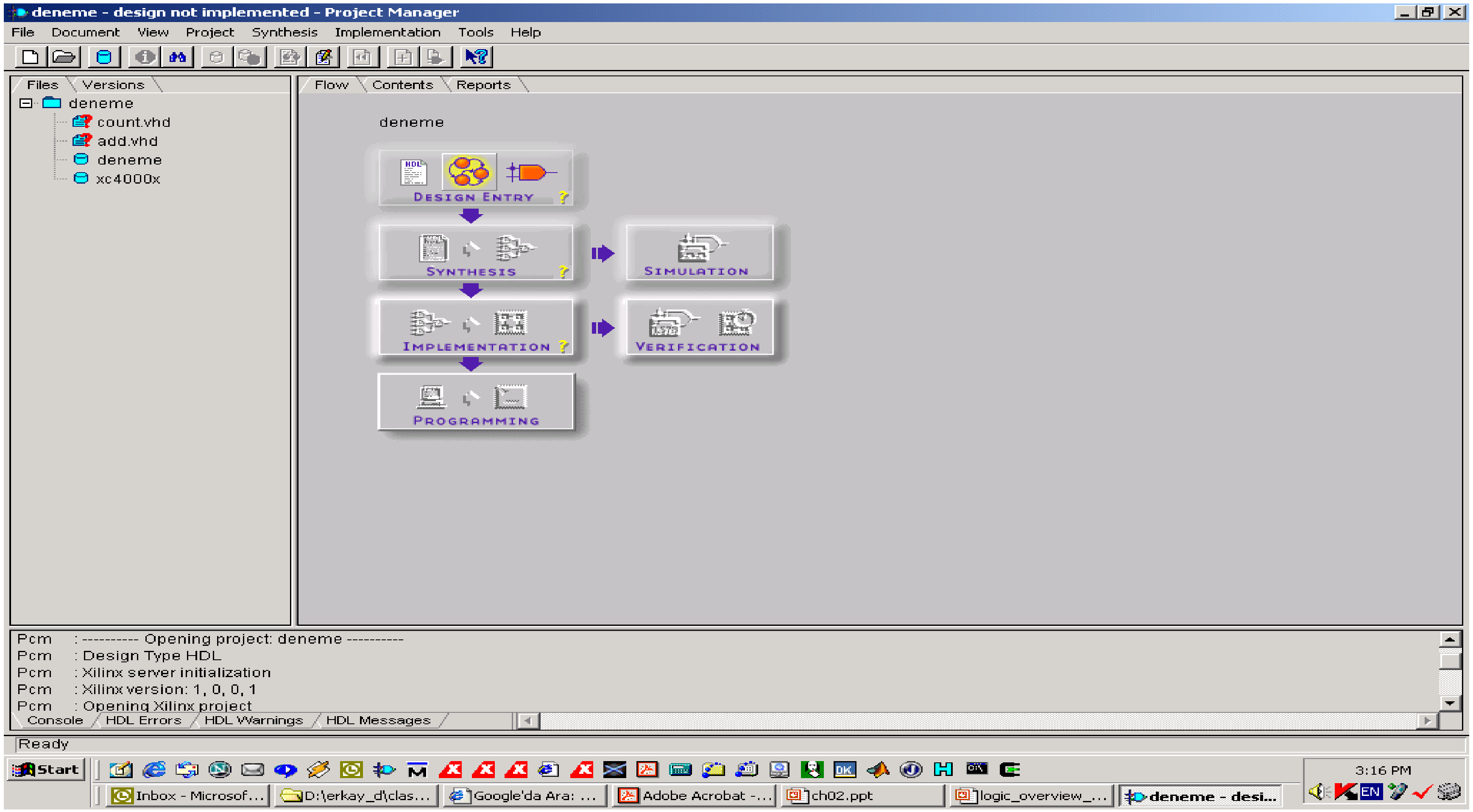
Computer-Aided Design - 1

- CAD
  - Design of digital systems with large circuits containing many 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
  - an application-specific integrated circuit (ASIC),
  - a field-programmable gate array (FPGA),
  - a programmable logic device (PLD),
  - and a full-custom IC.
- For every piece of device we have an array of software tools to facilitate
  - design
  - simulation,
  - testing,
  - and even programming

Xilinx Tools

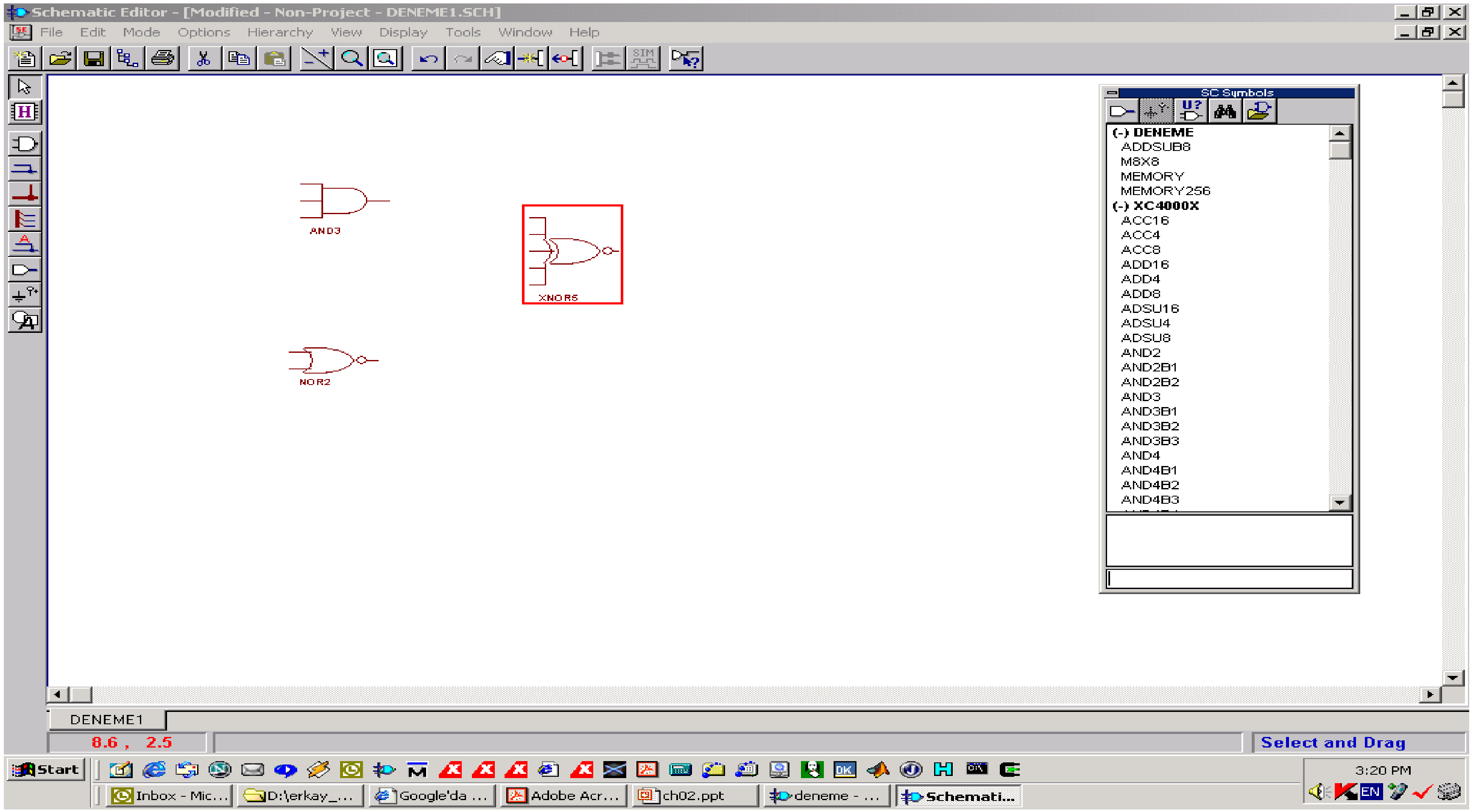


```
Pcm   : ---------- Opening project: deneme ----------
Pcm   : Design Type HDL
Pcm   : Xilinx server initialization
Pcm   : Xilinx version: 1, 0, 0, 1
Pcm   : Opening Xilinx project
```

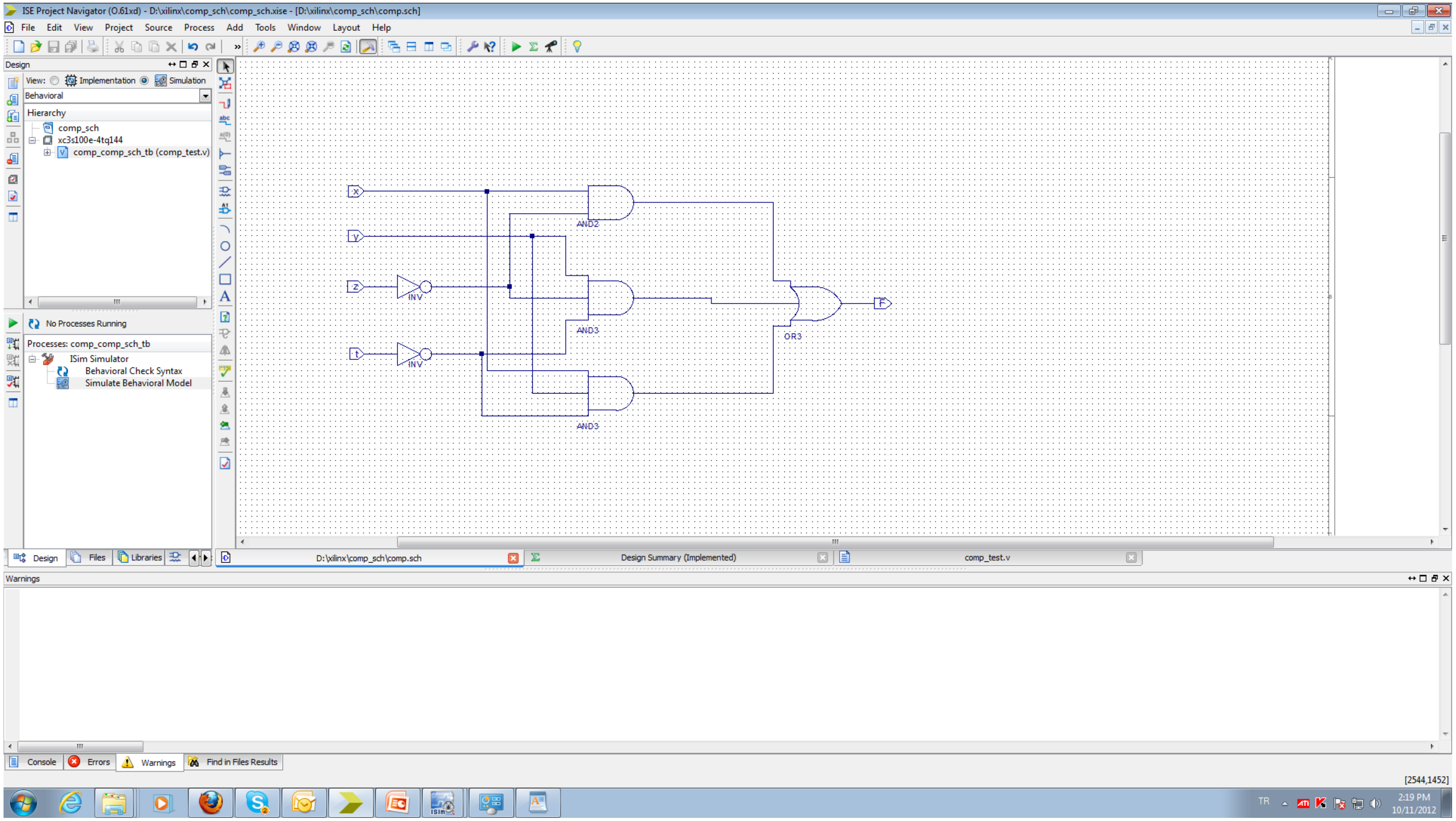
Schematic Editor

- 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



ISE Project Navigator (O.61xd) - D:\xilinx\comp_sch\comp_sch.xise - [D:\xilinx\comp_sch\comp.sch]
File Edit View Project Source Process Add Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
comp_sch
xc3s100e-4tq144
comp_comp_sch_tb (comp_test.v)
No Processes Running
Processes: comp_comp_sch_tb
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
x
y
z
t
INV
INV
AND2
AND3
AND3
OR3
F
Design Files Libraries
D:\xilinx\comp_sch\comp.sch
Design Summary (Implemented)
comp_test.v
Warnings
Console Errors Warnings Find in Files Results
[2544,1452]
TR
2:19 PM
10/11/2012

Hardware Description Languages

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

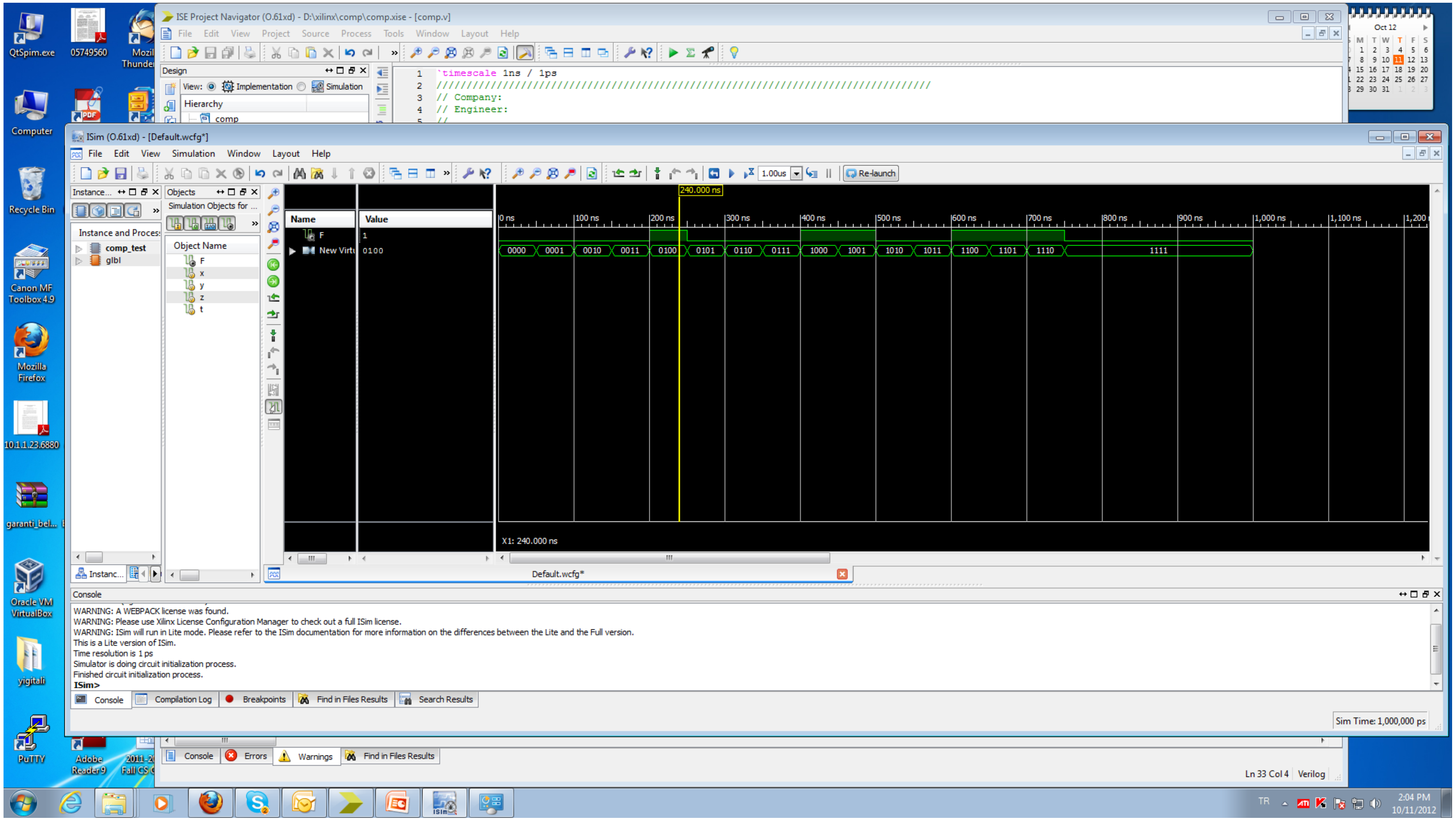
```
module comp(F, x, y, z, t);
      input x, y, z, t;
      output F;
      wire e1, e2, e3;
      and g1(e1, x, ~z);
      and g2(e2, y, ~z, ~t);
      and g3(e3, x, y, ~t);
      or g4(F, e1, e2, e3);
endmodule
```

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

Hardware Description Languages

- HDL
  - It represents logic diagrams and other digital information in textual form to describe the functionality and structure of a circuit.
  - Moreover, the HDL description of a circuit’s functionality can be abstract, without reference to specific hardware,
  - thereby freeing a designer to devote attention to higher level functional detail (e.g., under certain conditions the circuit must detect a particular pattern of 1’s and 0’s in a serial bit stream of data) rather than transistor-level detail.

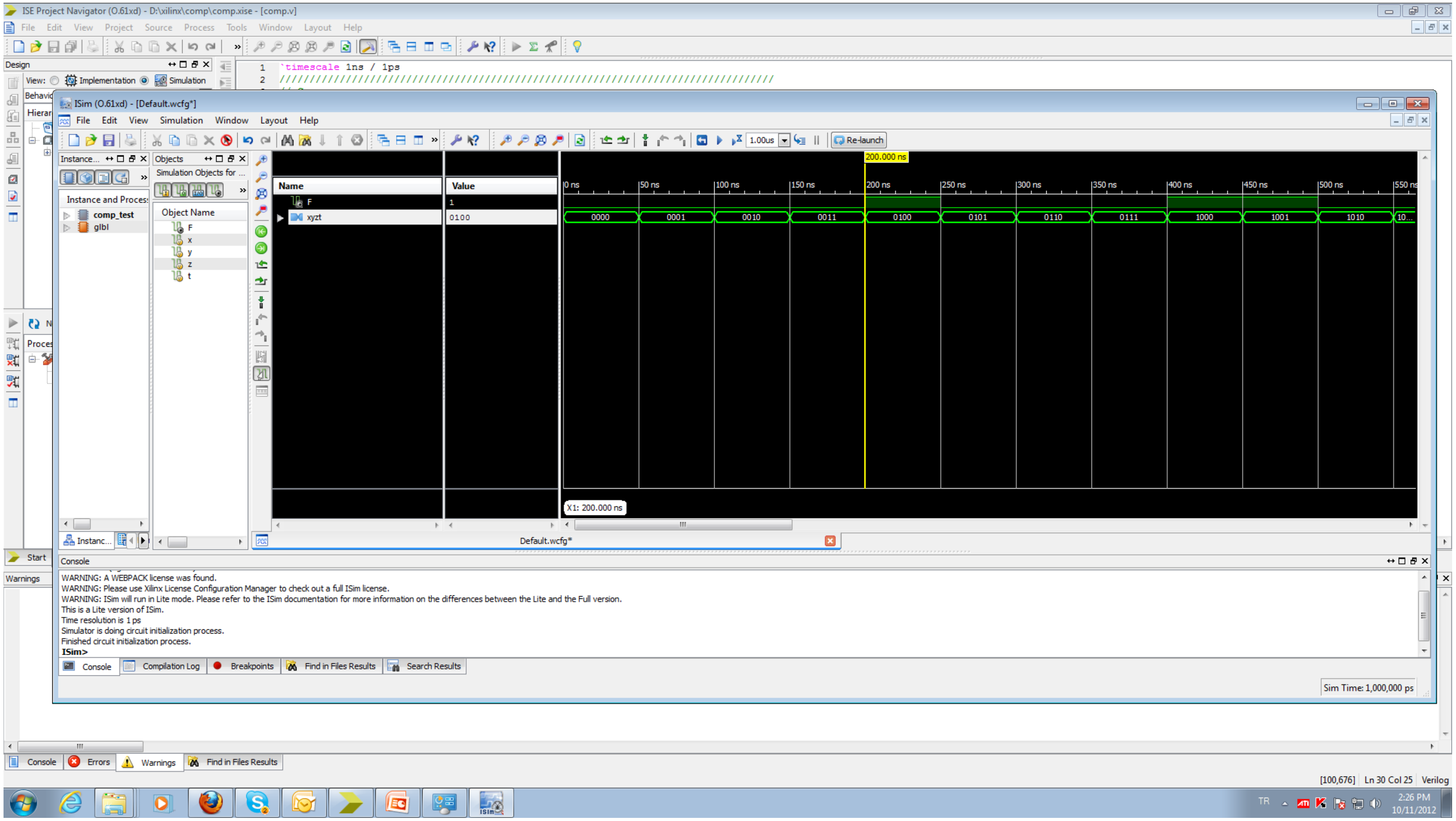
Simulation Results 1/3



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
```

```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

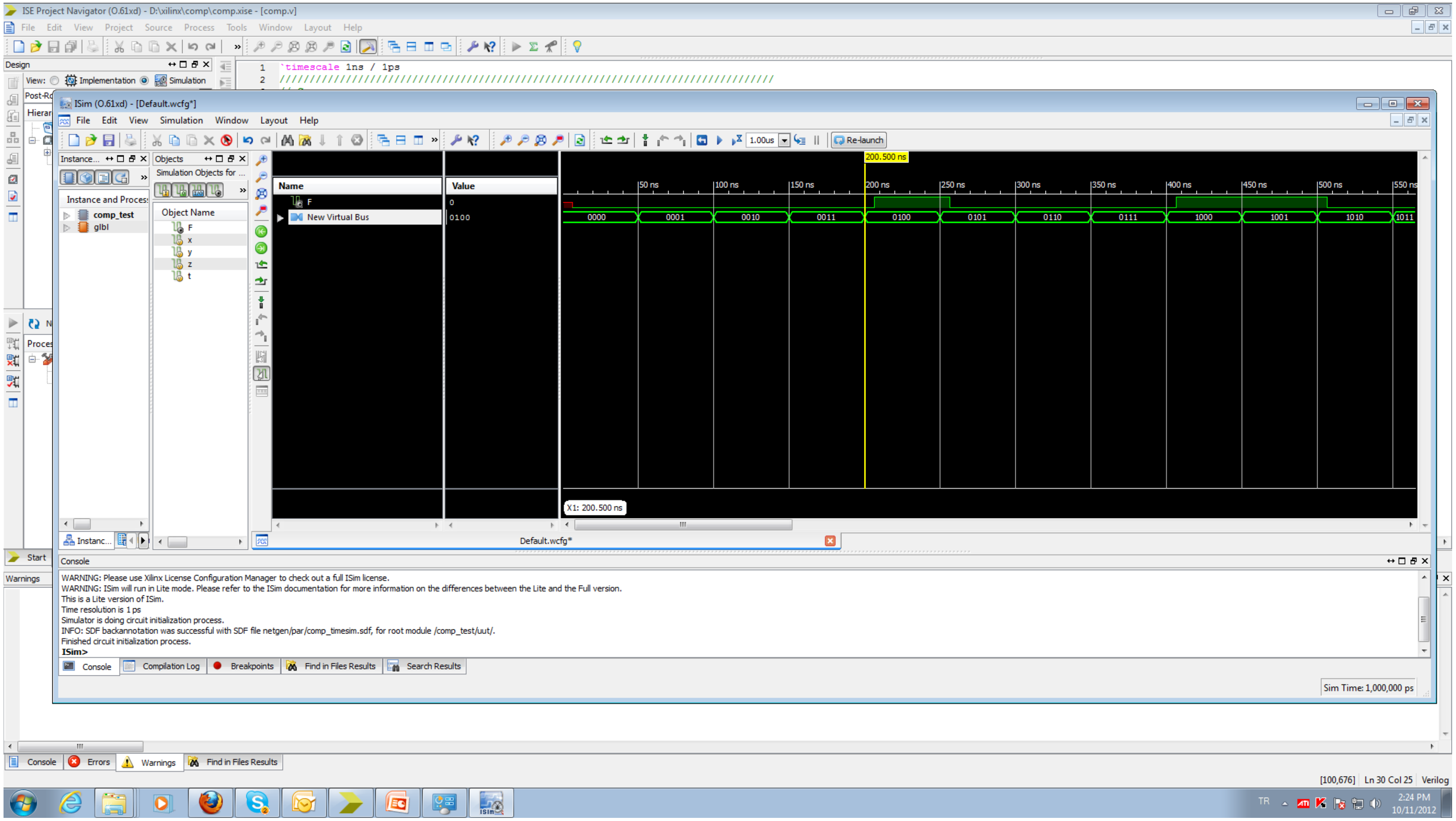
Simulation Results 2/3



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
```

```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Simulation Results 3/3



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
```

```
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
INFO: SDF backannotation was successful with SDF file netgen/par/comp_timesim.sdf, for root module /comp_test/uut/.
Finished circuit initialization process.
ISim>
```