Ollscoil na Gaillimhe
University of Galway

CT101 Computing Systems

Dr. Bharathi Raja Chakravarthi
Lecturer-above-the-bar
Email: bharathi.raja@universityofgalway.ie

University
ofGalway.ie

Ollscoil na Gaillimhe
University of Galway

Canonical & Standard Forms

Minterm/Standard product

- A binary variable may appear either in its normal form (**x**) or in its complement form (**x'**).
- Consider two binary variables x and y combined with an **AND** operation.
- Each variable may appear in either four possible combinations: **x'y', x'y, xy'**, and **xy**.
- Each of these four AND terms is called a **minterm**, or a **standard product**.

Minterm/Standard product

- In a similar manner, **n** variables can be combined to form **2^n** minterms.
- The binary numbers from **0** to **2^n - 1** are listed under the **n** variables.
- Each **minterm** is obtained from an **AND** term of the n variables, with each variable being primed if the corresponding bit of the binary number is a **0** and unprimed if a **1**.
- A symbol for each **minterm** is of the form **m_j**.

Where,
j - the decimal equivalent of the binary number of the minterm designated.

Minterm/Standard product

The **2^n** different minterms determined by a method shown in Table for three variables.

			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

Maxterm/Standard sum

- n variables forming an OR term, with each variable being primed or unprimed, provide **2^n** possible combinations, called **maxterms**, or **standard sums**.

- Note that
  (1) each maxterm is obtained from an OR term of the n variables, with each variable being unprimed if the corresponding bit is a **0** and primed if a **1**, and
  (2) each **maxterm** is the **complement of** its corresponding **minterm** and vice versa.

- A Boolean function can be expressed algebraically from a given truth table by **forming a minterm** for each combination of the variables that **produces a 1** in the function and then **taking the OR of all those terms**.

Maxterm/Standard sum

The **eight(2^n)** maxterms for three variables, together with their symbolic designations, are listed in Table.

			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

Minterms

For example,

- The function f_1 in Table is determined by expressing the combinations **001**, **100**, and **111** as **x'y'z**, **xy'z'**, and **xyz**, respectively.
- Since each one of these minterms results in $f_1 = 1$, we have

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

x	y	z	Function f_1	Function f_2
0	0	0	0	0
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

Minterms

- Similarly, it may be easily verified that

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

- These examples demonstrate an important property of Boolean algebra: Any Boolean function can be expressed as a sum of **minterms** (with "**sum**" meaning the **OR**ing of terms).

Maxterms

- Now consider the complement of a Boolean function.

- It may be read from the truth table by **forming a minterm** for each combination that **produces a 0** in the function and **then ORing** those terms.

- The complement of f1 is read as

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Maxterms

- If we take the complement of $\mathbf{f_1'}$, we obtain the function $\mathbf{f_1}$:

$$\mathbf{f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)}$$
$$\mathbf{= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6}$$

- Similarly, it is possible to read the expression for $\mathbf{f_2}$ from the table:

$$\mathbf{f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)}$$
$$\mathbf{= M_0M_1M_2M_4}$$

Maxterms

- These examples demonstrate a second property of Boolean algebra: Any Boolean function can be expressed as a product of **maxterms** (with "**product**" meaning the **AND**ing of terms).

- Form a maxterm for each combination of the variables that produces a **0** in the function, and then form the **AND** of all those maxterms.

- Boolean functions expressed as a **sum of minterms or product of maxterms** are said to be in **canonical form**

Sum of Minterms

- The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

- Since the function can be either **1** or **0** for each minterm, and since there are $\mathbf{2^n}$ minterms, one can calculate all the functions that can be formed with **n** variables to be $\mathbf{2^{2^n}}$.

- It is sometimes convenient to express a Boolean function in its **sum-of-minterms** form.

Sum of Minterms

- If the function is not in **sum-of-minterms** form, it can be made so by first expanding the expression into a **sum of AND** terms.

- Each term is then inspected to see if it contains all the variables.

- If it misses one or more variables, it is **ANDed** with an expression such as **x + x'**, where x is one of the missing variables.

Sum of Minterms

- Express the Boolean function **F = A + B'C** as a sum of minterms.

- The function has three variables: **A**, **B**, and **C**. The first term **A** is missing two variables; therefore,

$$A = A(B + B') = AB + AB'$$

- This function is still missing one variable, so

$$A = AB(C + C') + AB'(C + C')$$
$$= ABC + ABC' + AB'C + AB'C'$$

- The second term B'C is missing one variable; hence,

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Sum of Minterms

- Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

- But AB'C appears twice, and according to theorem 1 **(x + x = x)**, it is possible to remove one of those occurrences.

- Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C + ABC' + ABC \\ &= m1 + m4 + m5 + m6 + m7 \end{aligned}$$

Sum of Minterms

- When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$\mathbf{F(A, B, C) = \Sigma(1, 4, 5, 6, 7)}$$

  - The summation symbol $\sum$ stands for the ORing of terms
  - the numbers following it are the indices of the minterms of the function.

- The letters in parentheses following **F** form a list of the variables in the order taken when the minterm is converted to an **AND** term.

Sum of Minterms

- An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

- Consider the Boolean function given in Example;

$$F = A + B'C$$

Sum of Minterms

- The truth table shown below can be derived directly from the algebraic expression by listing the eight binary combinations under variables A,B, and C and inserting 1's under F for those combinations for which **A = 1** and **BC = 01**.
- From the truth table, we can then read the **five minterms** of the function to be 1, 4, 5, 6, and 7.

Truth Table for $F = A + B'C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Product of Maxterms

- Each of the 2^{2^n} functions of **n** binary variables can be also expressed as a **product of maxterms**.
- To express a Boolean function as a product of maxterms, it must first be brought into a form of **OR** terms.
- This may be done by using the distributive law, **$x + yz = (x + y)(x + z)$**.
- Then any missing variable **x** in each OR term is **ORed with xx'**. The procedure is clarified in the following example.

Product of Maxterms

- Express the Boolean function **F = xy + x'z** as a product of maxterms. First, convert the function into **OR terms** by using the distributive law:

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

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

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

Product of Maxterms

- The function has three variables: **x**, **y**, and **z**. Each OR term is missing one variable; therefore,

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

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

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

Product of Maxterms

- Combining all the terms and removing those which appear more than once, we finally obtain

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0M_2M_4M_5$$

- A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

- The product symbol, Π, denotes the **ANDing** of maxterms; the numbers are the indices of the maxterms of the function.

Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

- This is because the original function is expressed by those minterms which **make the function equal to 1**, whereas its complement is a 1 for those minterms for which the function is a 0.

- As an example, consider the function

$$F(A, B, C) = \sum (1, 4, 5, 6, 7)$$

Conversion between Canonical Forms

- This function has a complement that can be expressed as

$$F'(A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3$$

- Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m'_0 \cdot m'2 \cdot m'3 = M0M2M3 = \Pi(0, 2, 3)$$

Conversion between Canonical Forms

- The last conversion follows from the definition of minterms and maxterms as shown in Table below.

- From the table, it is clear that the following relation holds:

$$\mathbf{m'_j = M_j}$$

- That is, the maxterm with subscript **j** is a complement of the minterm with the same subscript **j** and vice versa.

			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

Conversion between Canonical Forms

- The last example demonstrates the conversion between a function expressed in sum-of-minterms form and its equivalent in product-of-maxterms form.

- A similar argument will show that the conversion between the product of maxterms and the sum of minterms is similar.

- We now state a general conversion procedure:

  - To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form.
  - In order to find the missing terms, one must realize that the total number of minterms or maxterms is $\mathbf{2^n}$, where n is the number of binary variables in the function.

Conversion between Canonical Forms

- A Boolean function can be converted from an algebraic expression to a product of maxterms by means of a truth table and the canonical conversion procedure.

- Consider, for example, the Boolean expression

$$F = xy + x'z$$

- First, we derive the truth table of the function, as shown in Table.

Truth Table for $F = xy + x'z$

x	y	z	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

Minterms

Maxterms

Conversion between Canonical Forms

- The 1's under F in the table are determined from the combination of the variables for which xy = 11 or xz = 01.

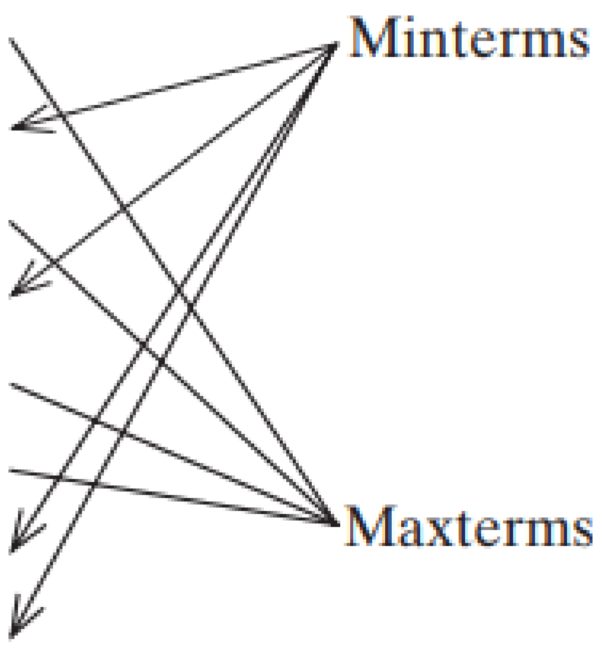
- The minterms of the function are read from the truth table to be 1, 3, 6, and 7.

- The function expressed as a sum of minterms is

$$F(x, y, z) = \sum(1, 3, 6, 7)$$

Truth Table for $F = xy + x'z$

x	y	z	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



Conversion between Canonical Forms

- Since there is a total of eight minterms or maxterms in a function of three variables, we determine the missing terms to be 0, 2, 4, and 5.

- The function expressed as a **product of maxterms** is

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Standard Forms

- The two canonical forms of Boolean algebra are basic forms that one obtains from reading a given function from the truth table.

- These forms are very seldom the ones with the least number of literals, because each minterm or maxterm must contain, by definition, all the variables, either complemented or uncomplemented.

- Another way to express Boolean functions is in **standard form**.

Standard Forms

- In this configuration, the terms that form the function may contain one, two, or any number of literals.

- Two types:
    - sum of products
    - products of sums

- The **sum of products** is a Boolean expression containing **AND terms**, called **product terms**, with one or more literals each.

- The sum denotes the ORing of these terms.

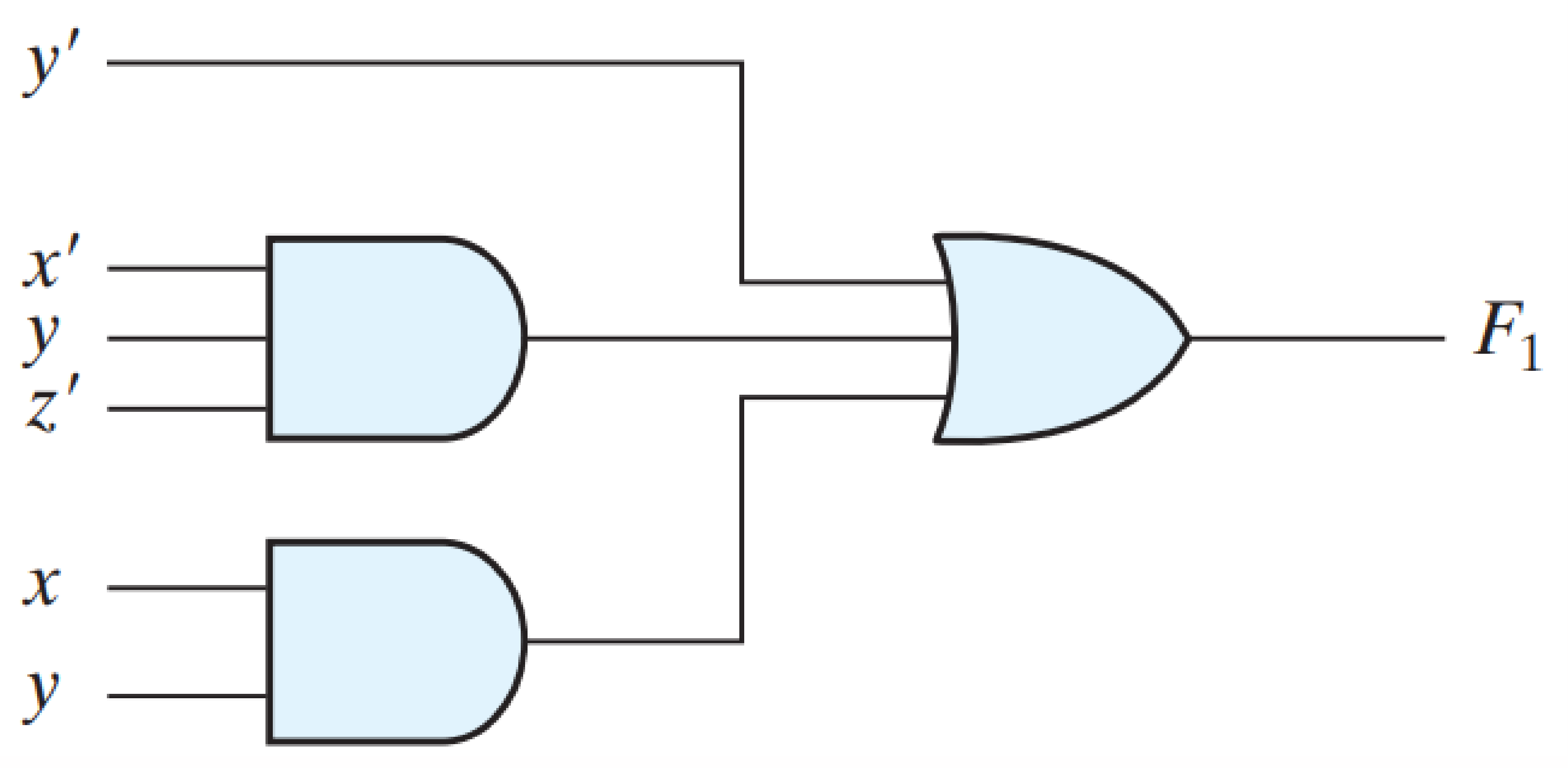
Standard Forms

- An example of a function expressed as a sum of products is

$$F1 = y' + xy + x'yz'$$

- The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an **OR** operation.

- The logic diagram of a sum-of-products expression consists of a group of **AND gates** followed by a **single OR gate**.



Standard Forms

- Each product term requires an AND gate, except for a term with a single literal.
- The logic sum is formed with an OR gate whose inputs are the outputs of the AND gates and the single literal.
- It is assumed that the input variables are directly available in their complements, so inverters are not included in the diagram.
- This circuit configuration is referred to as a **two-level implementation**.

Standard Forms

- A **product of sums** is a Boolean expression containing OR terms, called sum terms.
- Each term may have any number of literals.
- The product denotes the ANDing of these terms.

Standard Forms

- An example of a function expressed as a product of sums is

$$F2 = x(y' + z)(x' + y + z')$$

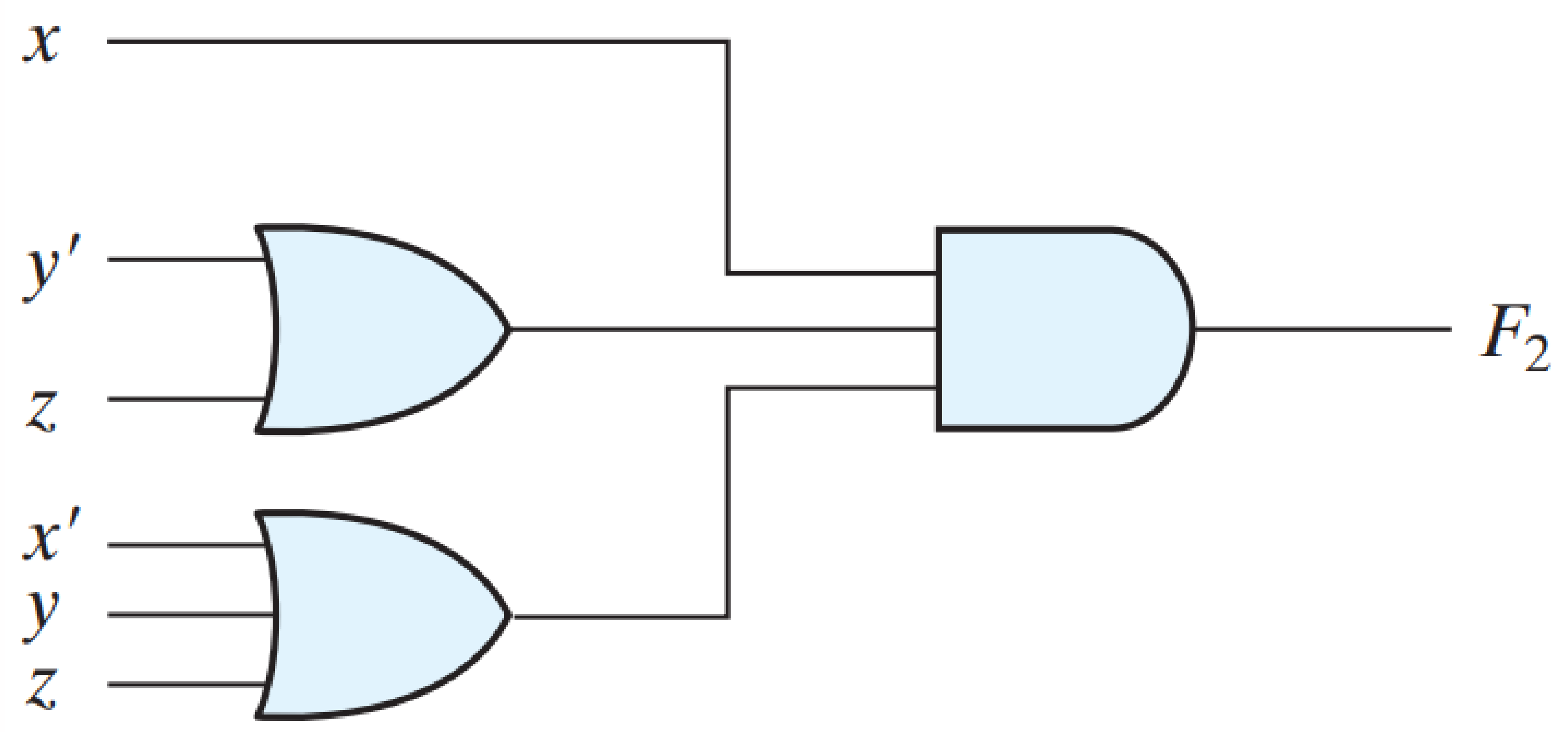
- This expression has three sum terms, with one, two, and three literals.
- The product is an AND operation.
- The use of the words product and sum stems from the similarity of the AND operation to the arithmetic product (multiplication) and the similarity of the OR operation to the arithmetic sum (addition).

Standard Forms

- The gate structure of the product-of-sums expression consists of a **group of OR gates** for the sum terms (except for a single literal), **followed by an AND gate**, as shown in Figure below.

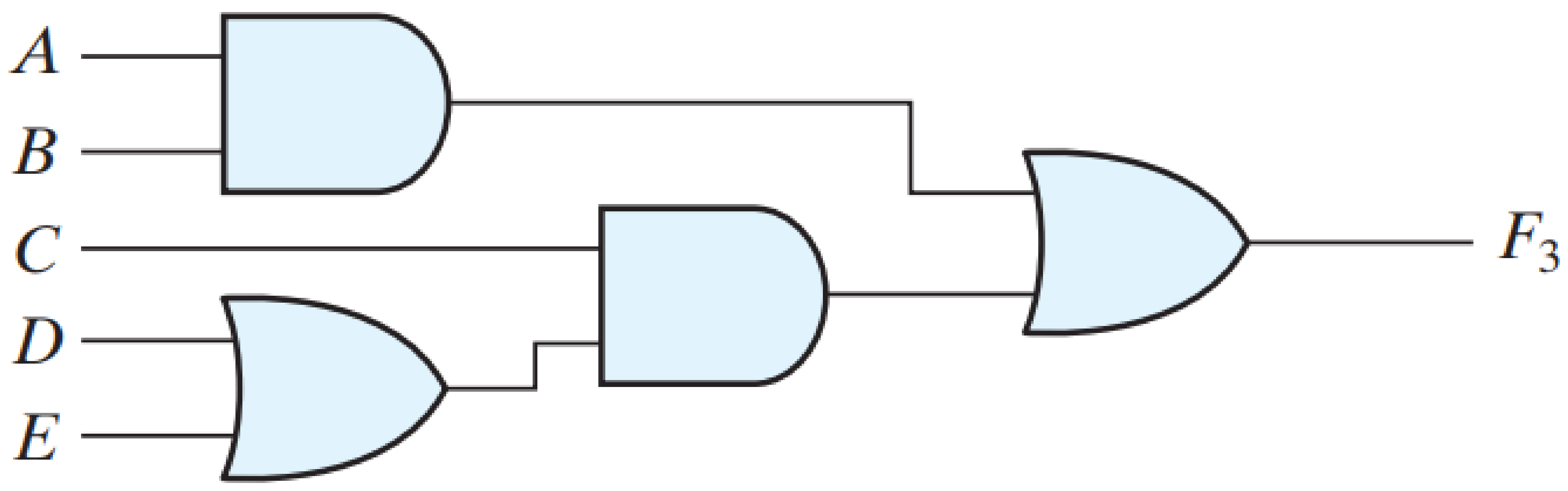
- This standard type of expression results in a **two-level** structure of gates.

- A Boolean function may be expressed in a nonstandard form.



Standard Forms

- For example, the function **F3 = AB + C(D + E)** is neither in sum-of-products nor in product-of-sums form.
- The implementation of this expression is shown below and requires **two AND gates** and **two OR gates**.
- There are three levels of gating in this circuit.



Ollscoil na Gaillimhe
University of Galway

Standard Forms

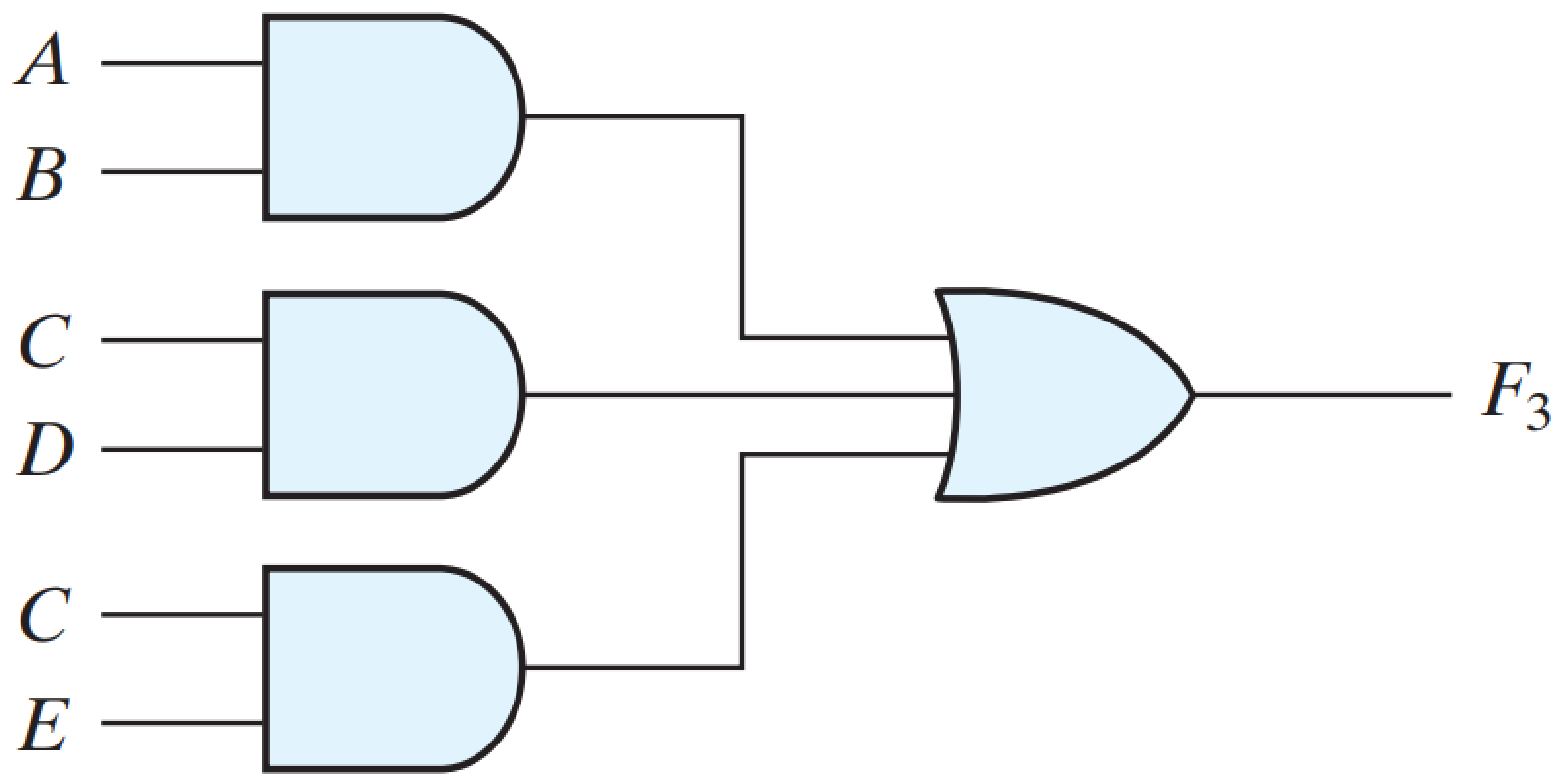
- It can be changed to a standard form by using the distributive law to remove the parentheses:

$$F3 = AB + C(D + E) = AB + CD + CE$$

- The sum-of-products expression is implemented in Figure shown below.

- In general, a **two-level implementation is preferred** because it produces the least amount of delay through the gates when the signal propagates from the inputs to the output.

- However, the number of inputs to a given gate might not be practical.



Other Logic Operations

- When **AND** and **OR** are placed between two variables, x and y form **x · y** and **x + y**.
- 2^{2^n} functions for **n** binary variables.
- For two variables, **n = 2**, and the number of possible Boolean functions is **16**.
- Therefore, the **AND and OR functions are only 2** of a total of 16 possible functions formed with two binary variables.
- It would be instructive to find the other 14 functions and investigate their properties.

Other Logic Operations

Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- The above table shows the truth tables for the 16 functions formed with two binary variables.
- Each of the 16 columns, F0 to F15, represents one possible function for the two variables, x and y.
- Note that the functions are determined from the 16 binary combinations that can be assigned to F.

Other Logic Operations

- 16 functions can be expressed algebraically in Boolean functions, as is shown in the first column of Table in the next slide.

- The Boolean expressions listed are simplified to their minimum number of literals.

Other Logic Operations

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Other Logic Operations

- Although each function can be expressed in terms of the Boolean operators AND, OR, and NOT, there is no reason one cannot assign special operator symbols for expressing the other functions.

- Such operator symbols are listed in the second column of Table.

- However, of all the new symbols shown, only the **exclusive-OR symbol**, $\oplus$, is in common use by digital designers.

- Each of the functions are listed with **name** and a **comment** that explains the function in some way.

Other Logic Operations

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Other Logic Operations

16 functions listed can be subdivided into three categories:

1. Two functions that produce a constant **0** or **1**.
2. Four functions with unary operations: **complement** and **transfer**.
3. Ten functions with binary operators that define eight different operations: **AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication**.

Note:
- ˆ symbol - used to indicate the exclusive or operator, e.g., xˆy.
- AND symbol - sometimes omitted from the product of two variables, e.g., xy.

Other Logic Operations

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Other Logic Operations

- Constants for binary functions - **1 or 0**.
- Complement function produces the complement of each binary variables.
- A function that is equal to an input variable has been given the name **transfer**, because the variable x or y is transferred through the gate that forms the function without changing its value.
- Of the eight binary operators, two (**inhibition and implication**) are used by logicians, but are seldom used in computer logic.
- **AND and OR** operators - conjunction with Boolean algebra.
- Other four functions are used extensively in the design of digital systems.

Other Logic Operations

- **NOR function** - complement of the OR function(not-OR).
- **NAND** - complement of AND(not-AND).
- **XOR (Exclusive-OR)** - similar to OR, but excludes the combination of both x and y being equal to 1; it holds only when x and y differ in value.
- **Equivalence** - a function that is 1 when the two binary variables are equal (i.e., when both are 0 or both are 1).
- The **exclusive-OR** and **equivalence** functions are the complements of each other.

Other Logic Operations

- This can be easily verified by inspecting below table: The truth table for exclusive-OR is F6 and for equivalence is F9, and these two functions are the complements of each other.
- The **equivalence** function is called **XNOR** (exclusive-NOR).

Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Digital Logic Gates

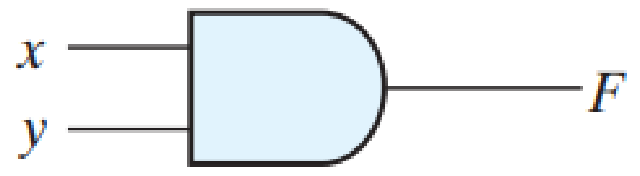
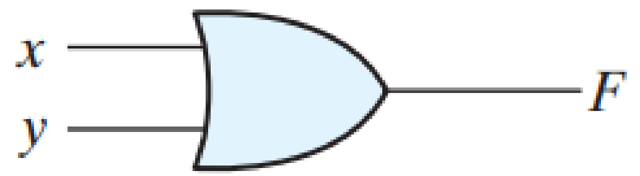
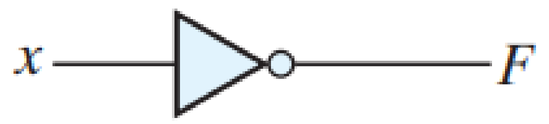
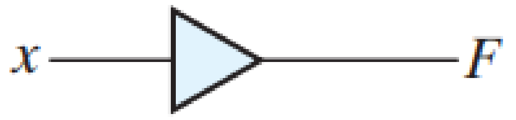
- It is easier to implement a Boolean function with AND, OR, and NOT gates.
- Still, the possibility of constructing gates for the other logic operations is of practical interest.
- Factors to be weighed in considering the construction of other types of logic gates are

    1. the feasibility and economy of producing the gate with physical components,
    2. the possibility of extending the gate to more than two inputs,
    3. the basic properties of the binary operator, such as commutativity and associativity, and
    4. the ability of the gate to implement Boolean functions alone or in conjunction with other gates.

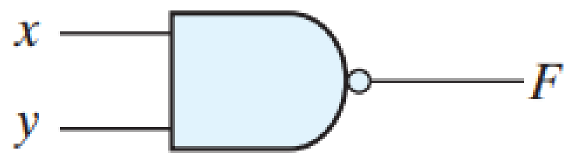
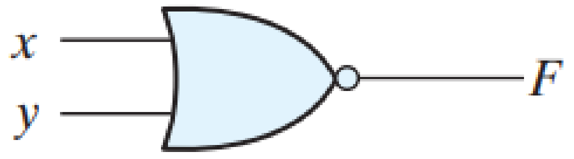
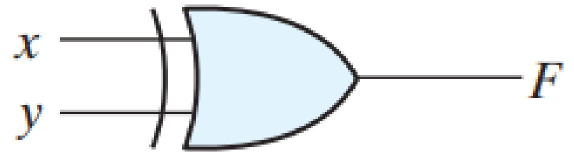
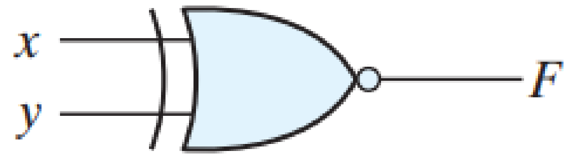
Digital Logic Gates

- Of the 16 functions defined, **two** are equal to a **constant** and **four** are **repeated**.
- There are only **10 functions** left to be considered as candidates for logic gates.
- **Two**—inhibition and implication—are not commutative or associative and thus are impractical to use as standard logic gates.
- The **other eight**—complement, transfer, AND, OR, NAND, NOR, exclusive-OR, and equivalence—are used as standard gates in digital design.

Digital Logic Gates

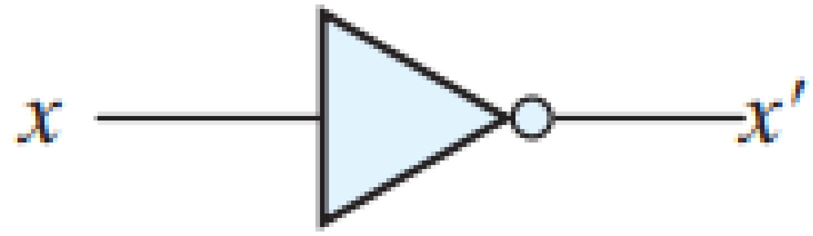
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Digital Logic Gates

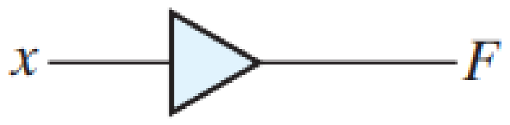
- From the table, Each gate has one or two binary input variables, designated by **x and y**, and one binary output variable, designated by **F**.
- The inverter circuit inverts the logic sense of a binary variable, producing the **NOT, or complement**, function.
- The small circle in the output of the graphic symbol of an inverter (referred to as a bubble) designates the logic complement.
- The triangle symbol by itself designates a buffer circuit.



Digital Logic Gates

- A **buffer** produces the transfer function, but does not produce a logic operation, since the binary value of the **output is equal to the input**.

Buffer



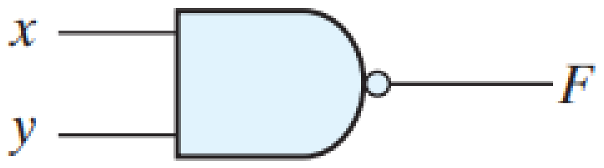
$F = x$

x	F
0	0
1	1

- This circuit is used for power amplification of the signal and is equivalent to two inverters connected in cascade.

- The **NAND function** is the complement of the AND function, as indicated by a graphic symbol that consists of an AND graphic symbol followed by a small circle.

NAND



$F = (xy)'$

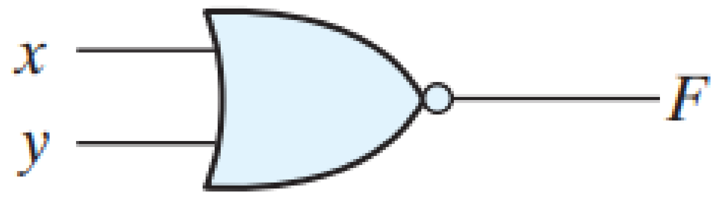
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

Digital Logic Gates

- The **NOR function** is the complement of the OR function and uses an OR graphic symbol followed by a small circle.

- NAND and NOR gates are used extensively as standard logic gates because these gates are easily constructed with transistor circuits and because digital circuits can be easily implemented with them.

NOR



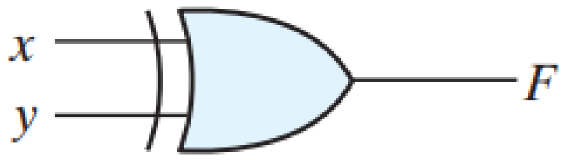
$F = (x + y)'$

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

Digital Logic Gates

- The **exclusive-OR gate** has a graphic symbol similar to that of the OR gate, except for the additional curved line on the input side.

Exclusive-OR (XOR)

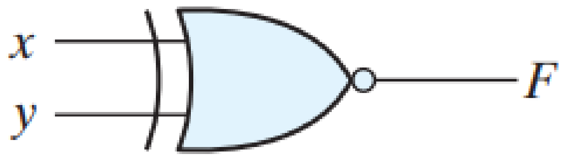


$$F = xy' + x'y = x \oplus y$$

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

- The **equivalence**, or **exclusive-NOR, gate** is the complement of the exclusive-OR, as indicated by the small circle on the output side of the graphic symbol.

Exclusive-NOR or equivalence



$$F = xy + x'y' = (x \oplus y)'$$

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

Extension to Multiple Inputs

- The gates except for the inverter and buffer can be extended to have more than two inputs.
- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.
- The AND and OR operations, defined in Boolean algebra, possess these two properties.
- For the OR function, we have

$$x + y = y + x \text{ (commutative)}$$

and

$$(x + y) + z = x + (y + z) = x + y + z \text{ (associative)}$$

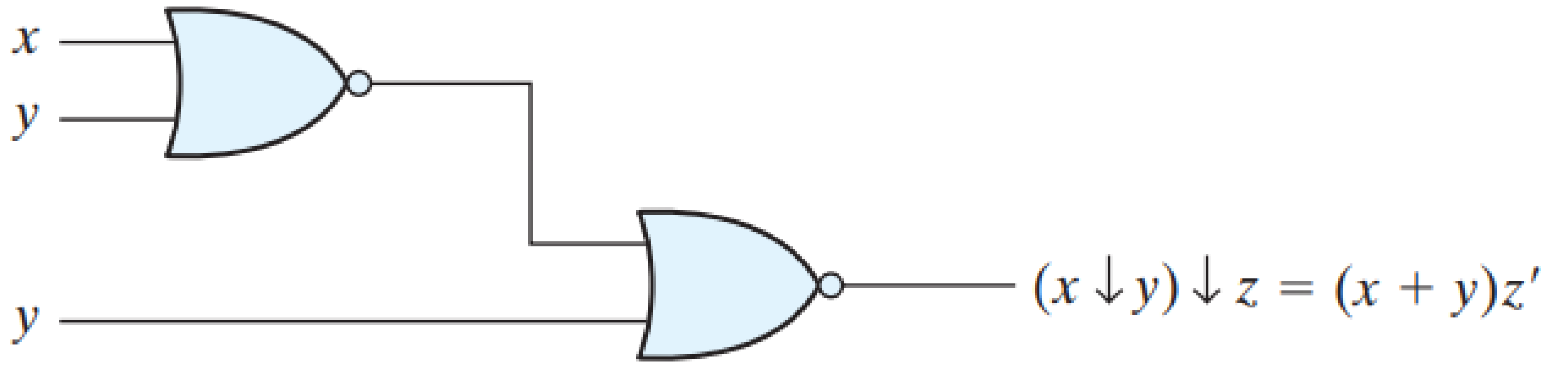
Note:
The above equations indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.

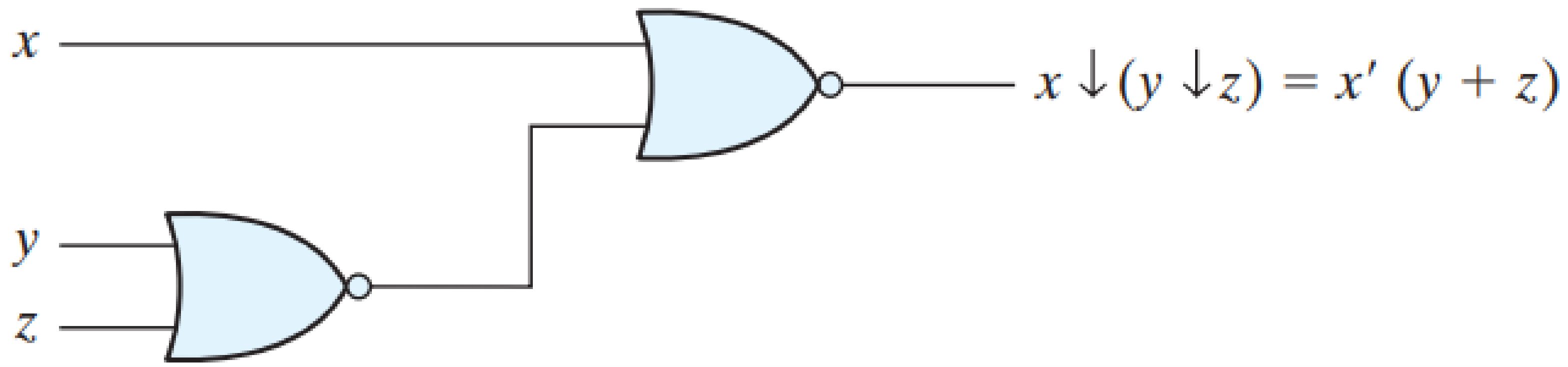
Extension to Multiple Inputs

- The **NAND and NOR functions** are commutative, and their gates can be extended to have more than two inputs, provided that the definition of the operation is modified slightly.

- The difficulty is that the NAND and NOR operators are not associative (i.e., $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$), and the following equations:

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$





Extension to Multiple Inputs

- To overcome this difficulty, the multiple NOR (or NAND) gate as a complemented OR (or AND) gate is defined. Thus, by definition, we have

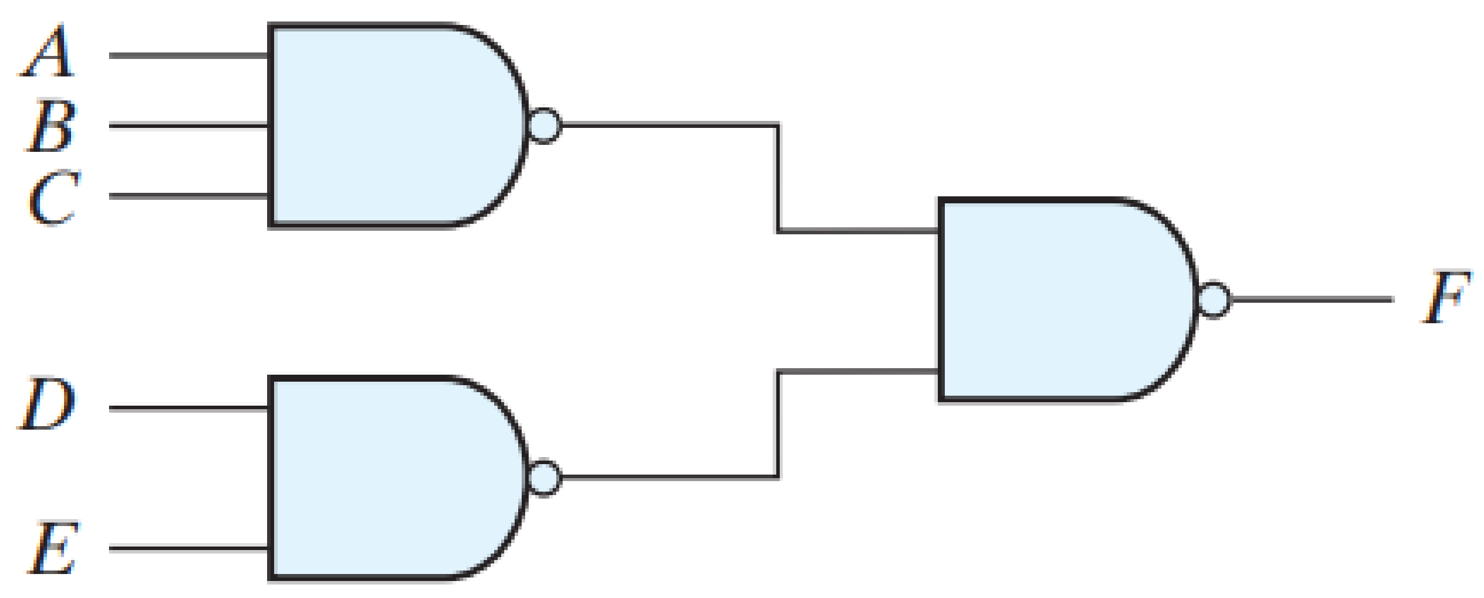
$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

- In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates.

Extension to Multiple Inputs

- To demonstrate this principle, consider the circuit of below Figure.
- The Boolean function for the circuit must be written as

F = [(ABC) '(DE) '] ' = ABC + DE

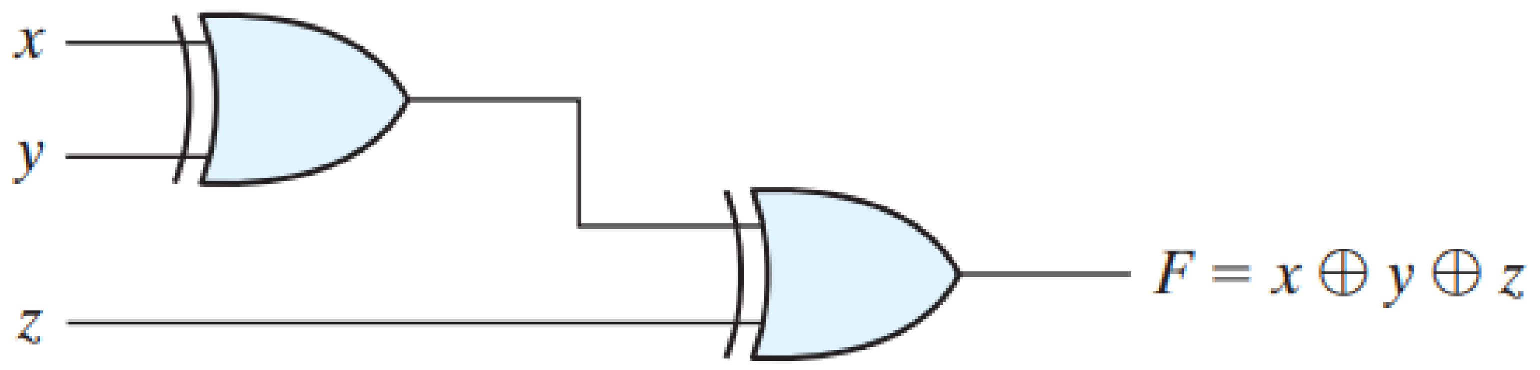


Extension to Multiple Inputs

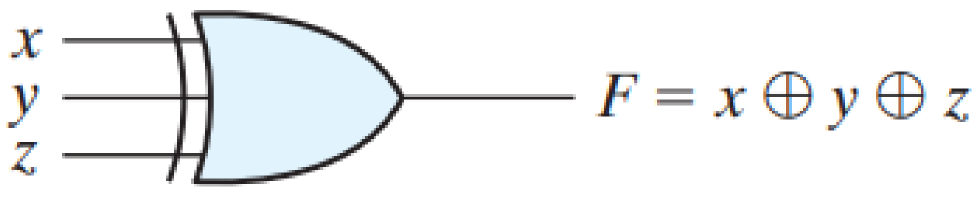
- The second expression is obtained from one of DeMorgan's theorems.
- It also shows that an expression in sum-of-products form can be implemented with NAND gates.
- **The exclusive-OR** and equivalence gates are both commutative and associative and can be extended to more than two inputs.

Extension to Multiple Inputs

- The definition of the function must be modified when extended to more than two variables.
- Exclusive-OR is an odd function (i.e., it is equal to 1 if the input variables have an odd number of 1's).
- The construction of a three-input exclusive-OR function is shown below.



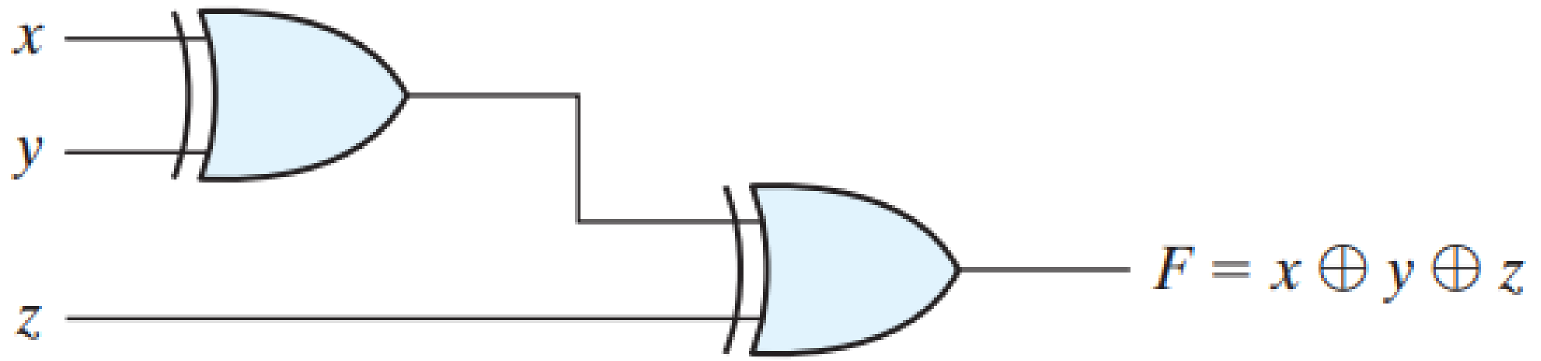
(a) Using 2-input gates



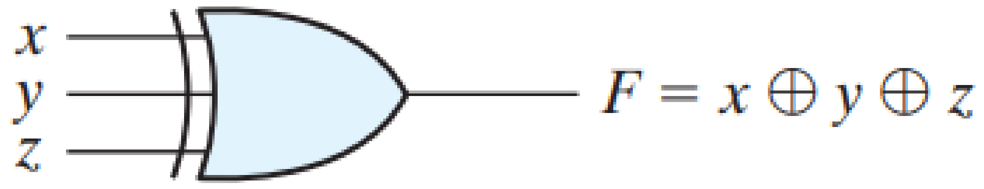
(b) 3-input gate

Extension to Multiple Inputs

- This function is normally implemented by cascading two-input gates, as shown in (a).
- Graphically, it can be represented with a single three-input gate, as shown in (b).
- The truth table in (c) clearly indicates that the output F is equal to 1 if only one input is equal to 1 or if all three inputs are equal to 1 (i.e., when the total number of 1's in the input variables is odd).



(a) Using 2-input gates



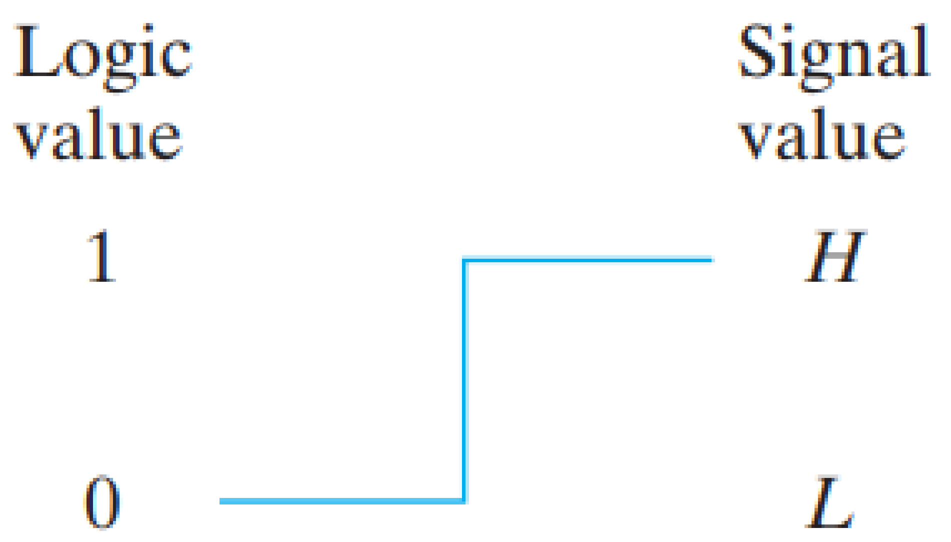
(b) 3-input gate

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

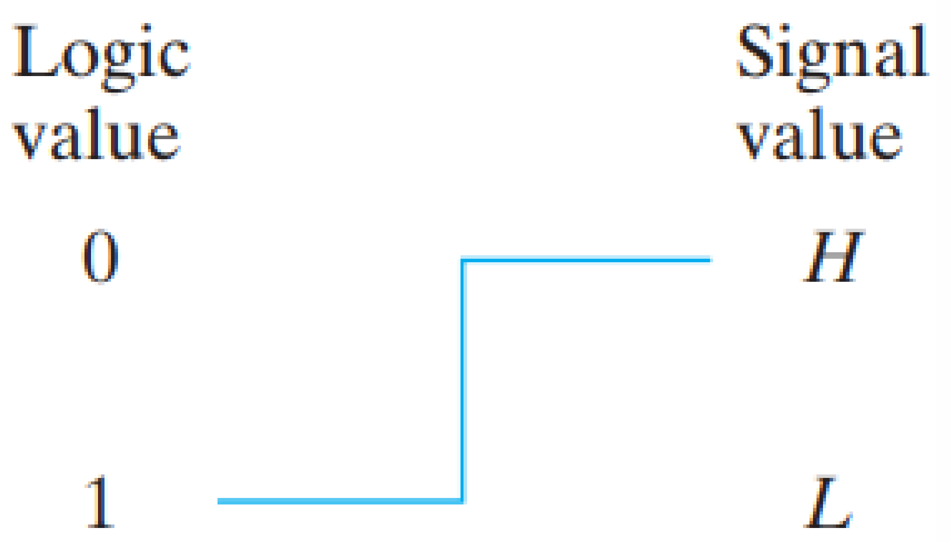
(c) Truth table

Positive and Negative Logic

- The binary signal at the inputs and outputs of any gate **has one of two values**, except during transition.
- One signal value represents **logic 1** and the other **logic 0**.
- Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value, as shown below.



(a) Positive logic



(b) Negative logic

Positive and Negative Logic

- The higher signal level is designated by **H** and the lower signal level by **L**.
- Choosing the high-level H to represent logic 1 defines a **positive logic** system.
- Choosing the low-level L to represent logic 1 defines a **negative logic** system.

(a) Positive logic (b) Negative logic

Positive and Negative Logic

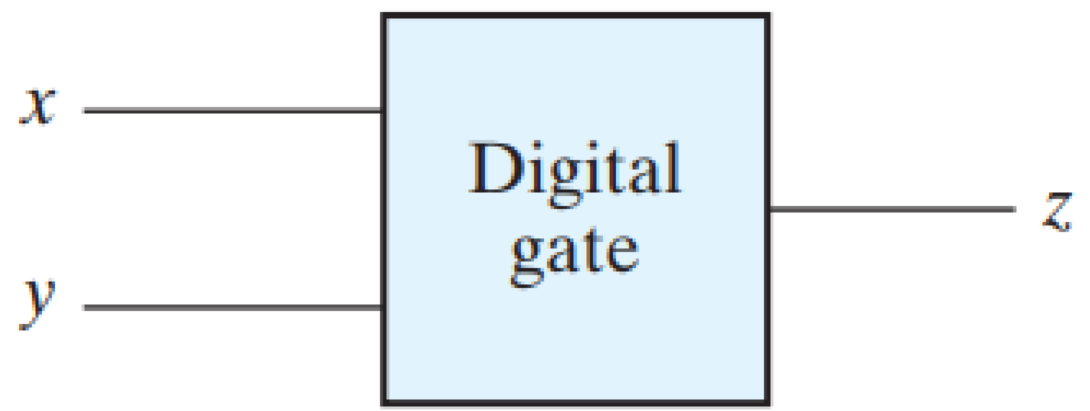
- The terms positive and negative are somewhat misleading, since both signals may be positive or both may be negative.
- It is not the actual values of the signals that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels.

Positive and Negative Logic

- Hardware digital gates are defined in terms of signal values such as **H** and **L**.
- It is up to the user to decide on a positive or negative logic polarity.
- Consider, for example, the electronic gate and the truth table shown below.

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with H and L



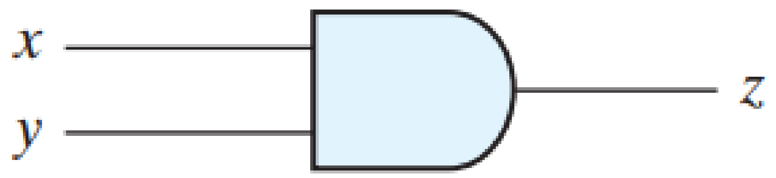
(b) Gate block diagram

Positive and Negative Logic

- It specifies the physical behavior of the gate when H is 3 V and L is 0 V.
- The truth table (c) assumes a positive logic assignment, with H = 1 and L = 0.
- This truth table is the same as the one for the AND operation.
- The graphic symbol for a positive logic AND gate is shown in Fig. 2.10 (d).

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

(c) Truth table for positive logic



(d) Positive logic AND gate

Positive and Negative Logic

- Now consider the negative logic assignment for the same physical gate with L = 1 and H = 0.
- The result is the truth table:

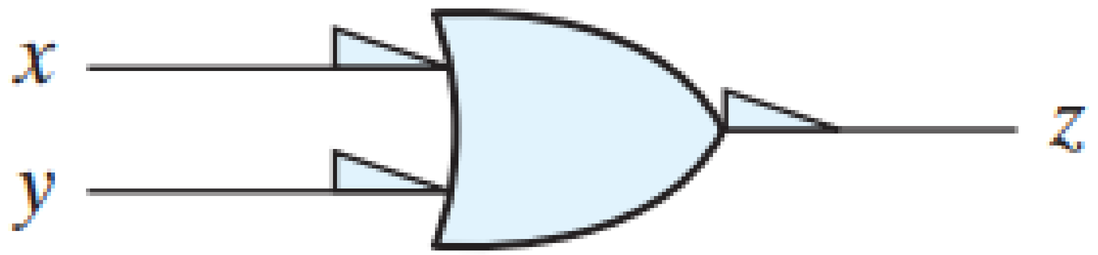
x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic

- This table represents the OR operation, even though the entries are reversed.

Positive and Negative Logic

- The graphic symbol for the negative- logic OR gate is shown in (f).



(f) Negative logic OR gate

- The small triangles in the inputs and output designate a **polarity indicator**, the presence of which along a terminal signifies that negative logic is assumed for the signal.
- Thus, the same physical gate can operate either as a positive-logic AND gate or as a negative-logic OR gate.

References

- Computer Organization and Architecture Designing for Performance Tenth Edition by William Stallings
- Digital Design With an Introduction to the Verilog HDL FIFTH EDITION by M Morris, M. and Michael, D., 2013.

Ollscoil na Gaillimhe
University of Galway

Thank *you*