Chapter 2
Boolean Algebra and Logic Gates

Definitions

Theorems

Functions

Canonical and Standard Forms

Operations

Gates

Integrated Circuits

Mathematical Systems

- Mathematical systems **can be defined with:**
  - **A set of** elements **containing a set of objects with common properties.**
  - **A set of** operators **defined on a set "S" of elements are rules those assign to each pair of elements an element from "S".**
  - **A set of** axioms **or** postulates **forming a basis from which we can deduce rules, theorems and properties of the system.**

Set Notations

- **The following notations are being used in this class:**
  - $x \in S$ **indicates that x is an element of the set S.**
  - $y \notin S$ **indicates that y is not an element of the set S.**
  - **A = {1, 2, 3, 4} indicates that set A exists with a finite number of elements (1, 2, 3, 4).**

Basic Postulates

- **The basic postulates of a mathematical system are:**
  - Closure**. A set S is closed w.r.t a binary operator if this operation only produces results that are within the set of elements defined by the system.**
  - Associative Law**. A binary operator is said to be associative when:**
    - » **(x * y) * z = x * (y * z)**
  - Commutative Law**. A binary operator is said to be commutative when:**
    - » **x * y = y * x**
  - Identity Element**. A set is said to have an identity element with respect to a binary operation if there exists an element, e, that is a member of the set with the property:**
    - » **e * x = x * e = x for every element of the set**
      - **Additive identity is 0 and multiplicative identity is 1**
  - **Note: * + and . are binary operators**

Basic Postulates

- Inverse. **For a set with an identity element with respect to a binary operation, the set is said to have an inverse if for every element of the set the following property holds:**
  - » **x * y = e**
    - **The additive inverse of element a is –a and it defines subtraction, since a + (–a) = 0. Multiplicative inverse of a is 1/a and defines division, since a.1/a = 1**
- Distributive Law. *** is said to be distributive over . when**
  - » **x * (y · z) = (x * y) · (x * z)**
- **Note: * + and . are binary operators. Binary operator + defines addition and binary operator . defines multiplication**

Two-Valued Boolean Algebra

- **Two-value Boolean algebra is defined by the:**
  - **The set of two elements B = {0, 1}**
  - **The operators of AND (·) and OR (+)**
  - Huntington Postulates **are satisfied**

Huntington Postulates

- **Boolean algebra has the following postulates:**
  1. Closure.
     a) **with respect to the binary operation OR (+)**
     b) **with respect to the binary operation AND (·)**
  2. Identity.
     a) **with respect to OR (+) is 0:**
        - **x + 0 = 0 + x = x, for x = 1 or x = 0**
     b) **with respect to AND (·) is 1:**
        - **x · 1 = 1 · x = x, for x = 1 or x =0**
  3. Commutative Law.
     a) **With respect to OR (+):**
        - **x + y = y + x**
     b) **With respect to AND (·):**
        - **x · y = y · x**

Huntington Postulates Continued…

- **Boolean algebra has the following postulates:**
  4. Distributive Law.
     a) **with respect to the binary operation OR (+):**
        - ❖ **$x + (y \cdot z) = (x + y) \cdot (x + z)$ + is distributive over .**
     b) **with respect to the binary operation AND (·):**
        - ❖ **$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$. is distributive over +**
  5. Complement.
     a) **$x + x' = 1$, for $x = 1$ or $x = 0$**
     b) **$x \cdot x' = 0$, for $x = 1$ or $x = 0$**
  6. Membership.
     - ❖ **There exists at least two elements, x and y, of the set such that $x \neq y$.**
       - ❖ **$0 \neq 1$**

Notes on Huntington Postulates

- **The associative law is not listed but it can be derived from the existing postulates for both operations.**
- **The distributive law of + over . i.e.,**

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

is valid for Boolean algebra but not for ordinary algebra.

- **Boolean algebra doesn't have inverses (additive or multiplicative) therefore there are no operations related to subtraction or division.**
- **Boolean algebra deals with only two elements, 0 and 1**

Operator Tables

- **A two-valued Boolean algebra is defined on a set of two elements B={0,1}, with rules for the two binary operators + and . as shown in the following operator tables:**

- **AND Operation**

x	y	x·y
0	0	0
0	1	0
1	0	0
1	1	1

- **OR Operation**

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

- **NOT Operation**

x	x'
0	1
1	0

Proving the Distributive Law

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

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

Duality

- **The duality principle states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.**
  - **The Huntington postulates have been listed in pairs and designed as part (a) and part (b).**
  - **If the dual of an algebraic equation is required, we interchange the OR and AND operators and replace 1's by 0's and 0's by 1's.**
  - **Example:**

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

Duality (More Examples)

P2	x+0 =x	x 1 = x
p5	x+x‘ = 1	x x‘ = 0
T1	x + x = x	x x = x
T2	x + 1 = 1	x 0 = 0
T3, involution	(x‘)’ = x	
p3	x+y = y+x	xy = yx
T4	x+(y+z)=(x+y)+z	x(yz) =(xy)z
P4	x(y+z)=xy+xz	x+yz=(x+y)(x+z)
T5, DeMorgan	(x+y)‘ =x’y‘	(xy)‘=x’+y‘
T6, absorption	x+xy = x	x(x+y) =x

Basic Theorems

- **The following six (6) theorems can be deduced from the Huntington postulates:**
  - **Theorem 1(a):** $x + x = x$
  - **Theorem 1(b):** $x \cdot x = x$
  - **Theorem 2(a):** $x + 1 = 1$
  - **Theorem 2(b):** $x \cdot 0 = 0$
  - **Theorem 3 (involution):** $(x')' = x$
  - **Theorem 4(a) (associative):** $x + (y + z) = (x + y) + z$
  - **Theorem 4(b) (associative):** $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
  - **Theorem 5(a) (DeMorgan):** $(x + y)' = (x' \cdot y')$
  - **Theorem 5(b) (DeMorgan):** $(x \cdot y)' = (x' + y')$
  - **Theorem 6(a) (absorption):** $x + x \cdot y = x$
  - **Theorem 6(b) (absorption):** $x \cdot (x + y) = x$

Proving Theorem 1(a)

$x + x = x$

$x + x = (x + x) \cdot 1$ **By postulate:** **2(b)**

$= (x + x) \cdot (x + x')$ **5(a)**

$= x + x \cdot x'$ **4(b)**

$= x + 0$ **5(b)**

$= x$ **2(a)**

Proving Theorem 1(b)

x . x = x		
x . x = x x + 0	By postulate:	2(a)
= x x + xx‘		5(b)
= x (x + x')		4(a)
= x . 1		5(a)
= x		2(b)

Proving Theorem 2(a)

$x + 1 = 1$		
$x + 1 = 1 \cdot (x + 1)$	By postulate:	2(b)
$= (x + x')(x + 1)$		5(a)
$= x + x' 1$		*4(b)*
$= x + x'$		2(b)
$= 1$		5(a)

- Theorem 2(b) can be proved by duality:

$x \cdot 0 = 0$

Proving Theorem 6(a)

$x + x \cdot y = x$		
$= x \cdot 1 + x \cdot y$	by postulate:	2(b)
$= x \cdot (1 + y)$		4(a)
$= x \cdot (y + 1)$		3(a)
$= x \cdot 1$	by theorem:	2(a)
$= x$	by postulate:	2(b)

Proving Theorem 6(b)

$x \cdot (x + y) =$		
$= (x + 0) \cdot (x + y)$	by postulate:	2(a)
$= x + 0 \cdot y$		4(b)
$= x + y \cdot 0$		3(b)
$= x + 0$	by theorem:	2(b)
$= x$	by postulate:	2(a)

DeMorgan's Theorem

- **With truth table the validity of first DeMorgan's Theorem can be shown**
- **(x + y)' = x'y'**

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

Operator Precedence

- **The operator precedence for evaluating Boolean expressions is:**
  1. **Parentheses**
  2. **NOT**
  3. **AND**
  4. **OR**

Boolean Functions

- **A Boolean function is an expression described by:**
  - **binary variables**
  - **constants 0 and 1**
  - **logic operation symbols**
- **For a given value of the binary variables the result of the function can either be 0 or 1.**
- **An example function:**
  - **$F_1 = x + y'z$**
  - **F_1 is equal to 1 if x is equal to 1 or if both y' and z equal to 1. F_1 is equal to 0 otherwise**

Function as a Truth Table

- **A Boolean function can be represented in a truth table.**
  - **A truth table is a list of combinations of 1's and 0's assigned to the binary variables and and a column that shows the value of the function for each binary combination**

x	y	z		$F_1 = x + y'z$
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

Function as a Gate Implementation

- **A Boolean function can be transformed from an algebraic expression into circuit diagram composed of logic gates.**
  - **$F_1 = x + y'z$**
  - **The logic-circuit diagram for this function is shown below:**

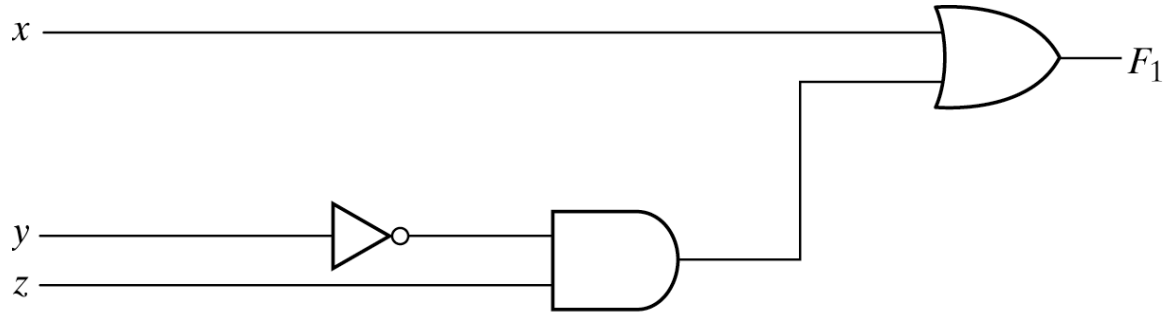
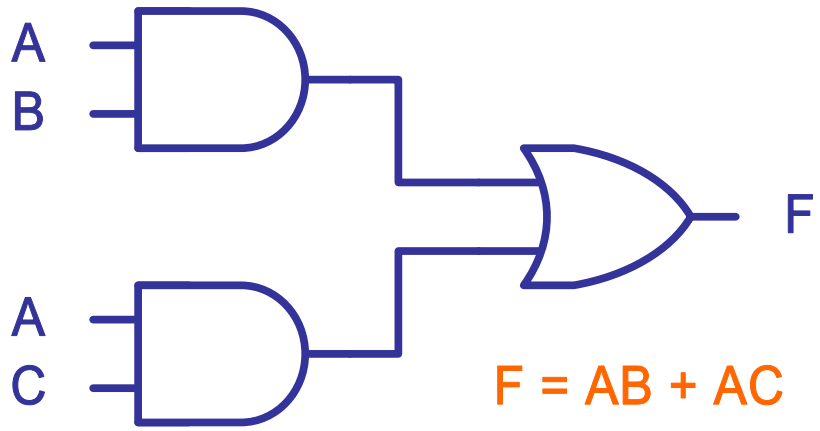
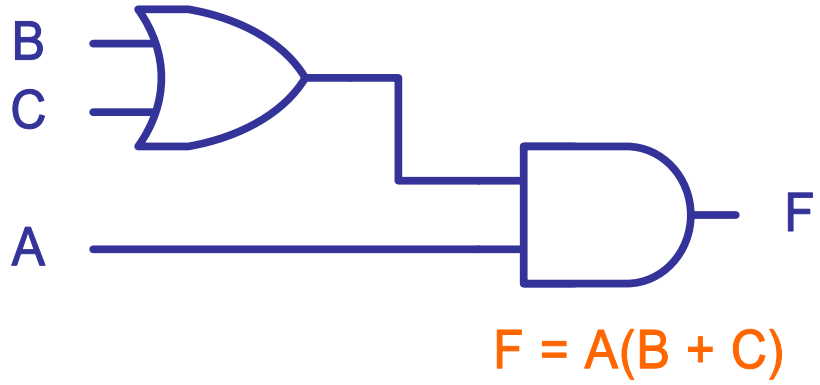


Fig. 2-1 Gate implementation of $F_1 = x + y'z$

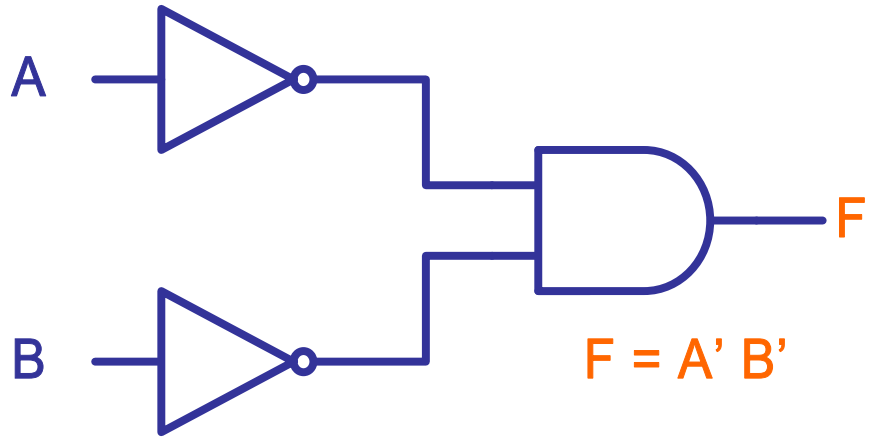
Gate Implementation (Examples)

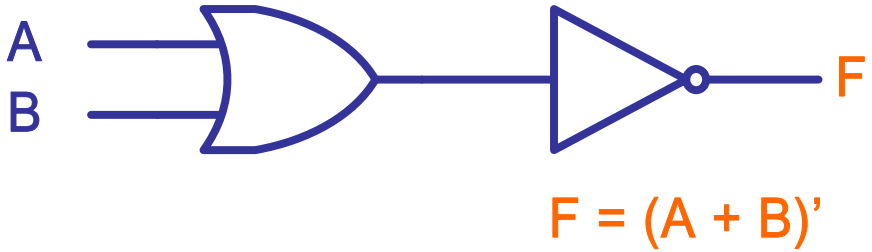




A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Gate Implementation (Examples)





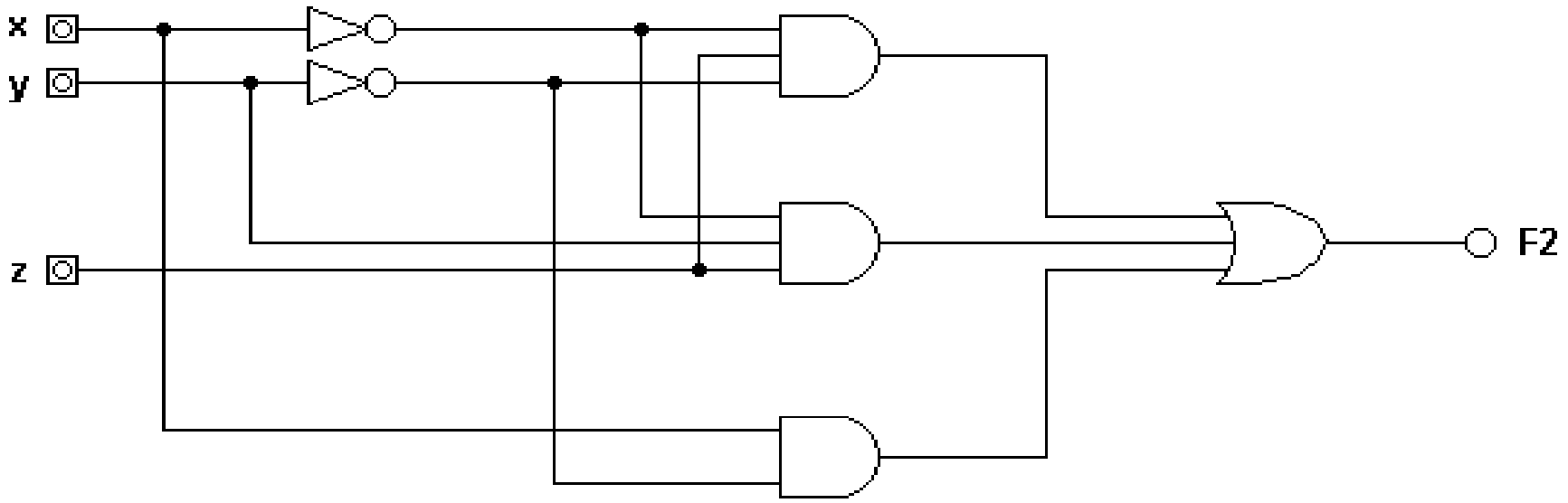
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Minimization

- **Functions in algebraic form can be represented in various ways.**
  - **Remember the postulates and theorems that allows us to represent a function in various ways.**
- **We must keep in mind that the algebraic expression is representative of the gates and circuitry used in a hardware piece.**
  - **We want to be able to minimize circuit design to reduce cost, power consumption, and package count, and to increase speed.**
- **By manipulating a function using the postulates and theorems, we may be able to minimize an expression.**

Non-Minimized Function

- **The following is an example of a non-minimized function:**
  - **$F_2 = x'y'z + x'yz + xy'$**



Minimization of the F_2

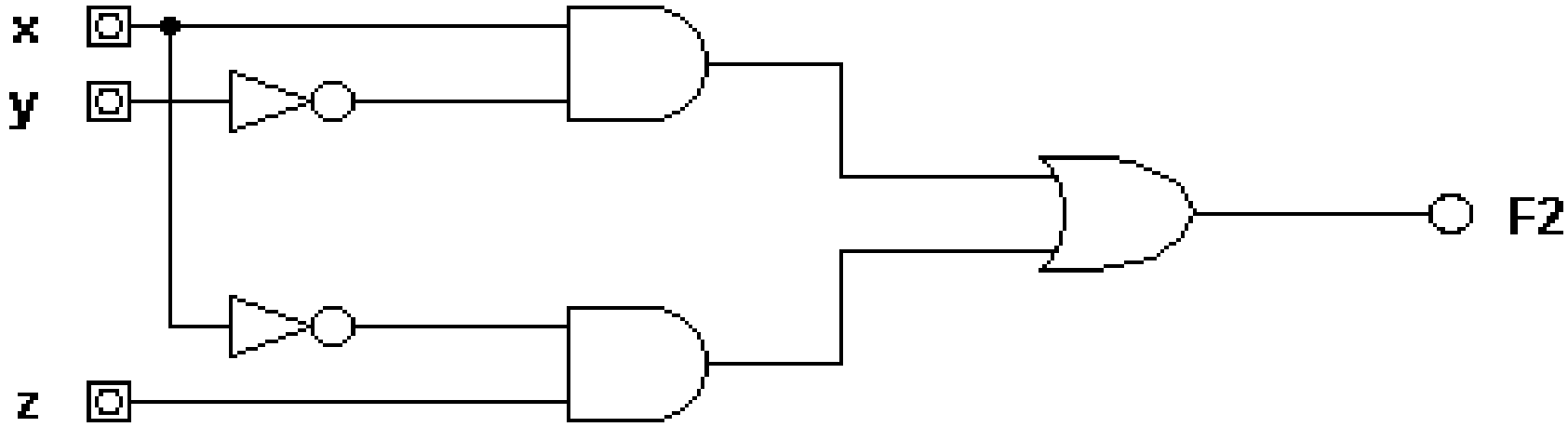
- **The function can be minimized as follows:**

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

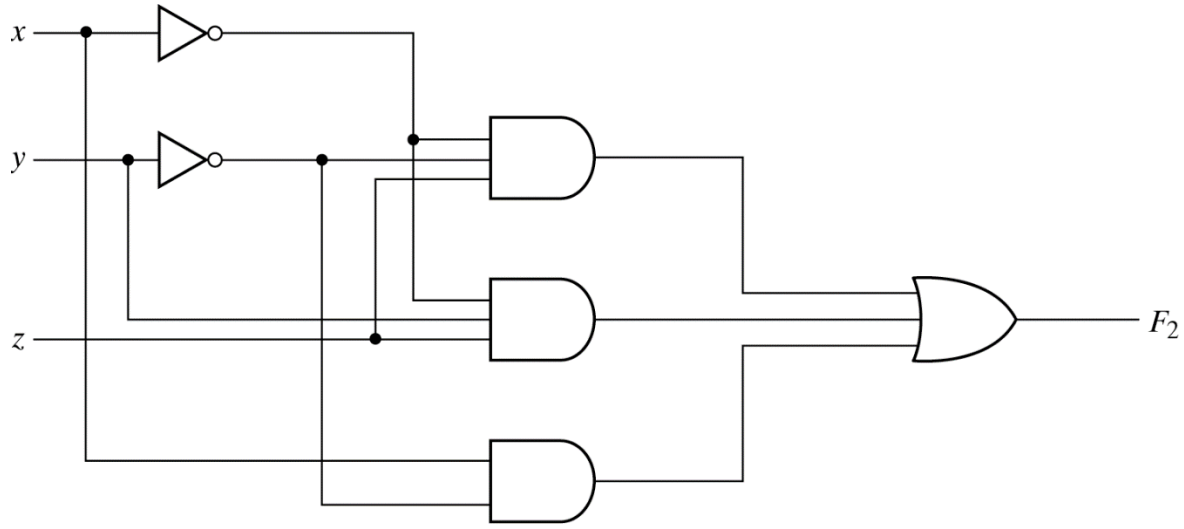
$= x'z \cdot (y' + y) + xy'$ **by postulate: 4(a)**

$= x'z \cdot 1 + xy'$ **5(a)**

$= x'z + xy'$ **2(b)**

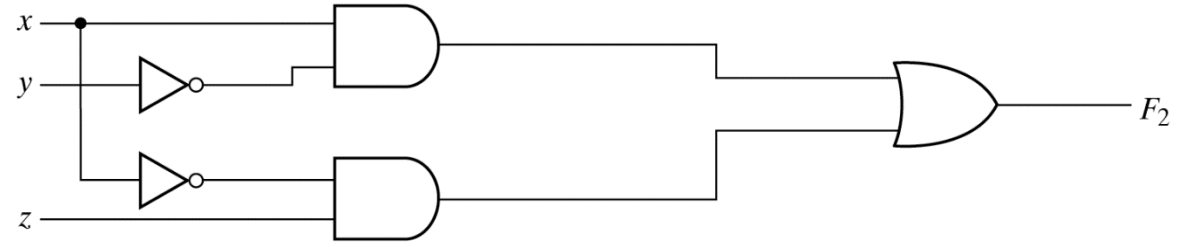


Implementation of Boolean Function



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

- Minimized Function



(b) $F_2 = xy' + x'z$

Fig. 2-2 Implementation of Boolean function F_2 with gates

Algebraic Manipulation

- **By reducing the number of terms, the number of literals (single variable) or both in a Boolean function, it is possible to obtain a simpler circuit, as each term requires a gate and each variable within the term designates an input to the gate .**
  - **F_1 = x'y'z + x'yz + xy' contains 3 terms and 8 literals**
  - **F_2 = x'z + xy' contains 2 terms and 4 literals.**

Example Manipulations

- **The following are some example manipulations:**
  1. **x(x’ + y) = xx’ + xy = 0 + xy = xy**
  2. **x + x’y = (x + x’)(x + y) = 1(x + y) = x + y**
  3. **(x + y)(x + y’) = x + xy + xy’ + yy’ = x(1 + y + y’) = x**
  4. **xy + x’z + yz = xy + x’z + yz(x + x’)**
     = xy + x’z + xyz + x’yz
     = xy(1 + z) + x’z(1 + y)
     = xy + x’z
  5. **(x + y)(x’ + z)(y + z) = (x + y)(x’ + z)** – HOME ASSIGNMENT!

Complement of a Function

- **The complement of a function F is F'.**
  - **It is obtained by interchanging 0's for 1's and 1's for 0's in the value of F.**
- **The complement of a function may be derived algebraically through DeMorgan's theorem.**
  - **Theorem 5(a) (DeMorgan): (x + y)' = (x' · y')**
  - **Theorem 5(b) (DeMorgan): (x · y)' = (x' + y')**
- **Example:**
  - **F_1 = x'yz' + x'y'z**

    F_1' = (x'yz' + x'y'z)'

    = (x + y' + z)(x + y + z') **TT**

Complement of a Function (Example)

- If $F_1 = A+B+C$
- Then F_1'

$=(A+B+C)'$

$= (A+X)'$ let B+C = X

$= A'X'$ by DeMorgan's

$= A'(B+C)'$

$= A'(B'C')$ by DeMorgan's

$= A'B'C'$ associative

Complement of a Function (More Examples)

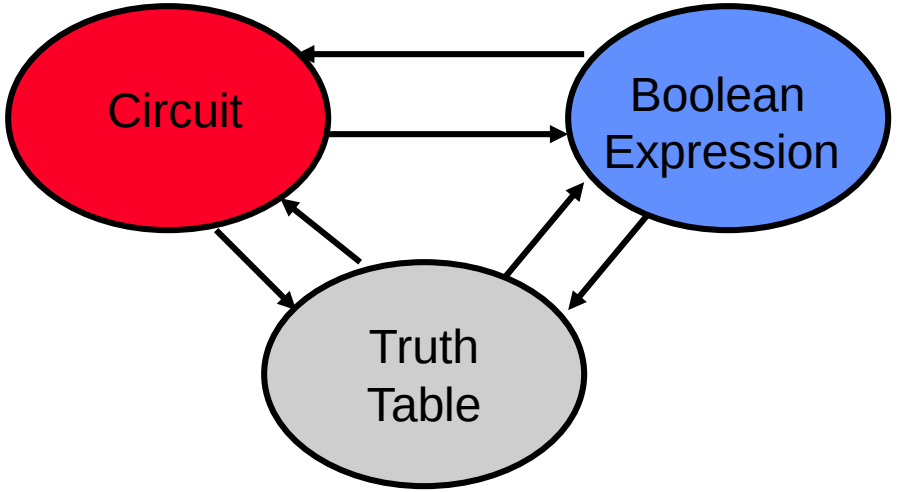
- **(x'yz' + x'y'z)'**
  = (x'yz')' (x‘y'z)'
  = (x+y'+z) (x+y+z')
- **[x(y'z'+yz)]'**
  = x' + (y'z'+yz)'
  = x' + (y'z')' (yz)'
  = x' + (y+z) (y'+z')
- A simpler procedure
  - **take the dual of the function (interchanging AND and OR operators and 1’s and 0’s) and complement each literal. {DeMorgan’s Theorem}**
  - **x'yz' + x'y'z**

    The dual of function is (x'+y+z') (x'+y'+z)

    Complement of each literal: (x+y'+z)(x+y+z')

Representation Conversion

- **Need to transition between boolean expression, truth table, and circuit (symbols).**
- **Converting between truth table and expression is easy.**
- **Converting between expression and circuit is easy.**
- **More difficult to convert to truth table.**



Canonical Forms

- **A canonical form is a standard method for representing Boolean functions.**
- **The two canonical forms that are used are:**
  - **Sum of Minterms**
  - **Product of Maxterms**
- **These forms are sometimes considered the "brute force" method of representing functions as they seldom represent a function in a minimized form.**

Minterms

- **Any given binary variable can be represented in two forms:**
  - **x, its normal form, and**
  - **x', its complement**
- **If we consider two binary variables and the AND operation, there are four combinations of the variables:**
  - **xy**
  - **xy'**
  - **x'y**
  - **x'y'**
- **Each of the above four AND terms is called a minterm or a standard product.**
- **n variables can be combined to form 2^n minterms.**

Minterms Expressed

				Minterms	
x	y	z		Term	Designation
0	0	0		$x'y'z'$	m_0
0	0	1		$x'y'z$	m_1
0	1	0		$x'yz'$	m_2
0	1	1		$x'yz$	m_3
1	0	0		$xy'z'$	m_4
1	0	1		$x'yz'$	m_5
1	1	0		xyz'	m_6
1	1	1		xyz	m_7

Maxterms

- **Any given binary variable can be represented in two forms:**
  - **x, its normal form, and**
  - **x', its complement**
- **If we consider two binary variables and the OR operation, there are four combinations of the variables:**
  - **x + y**
  - **x + y'**
  - **x' + y**
  - **x' + y'**
- **Each of the above four OR terms is called a maxterm or a standard sum.**
- **n variables can be combined to form 2^n maxterms.**
- Each maxterm is the complement of its corresponding minterm and vice-versa.

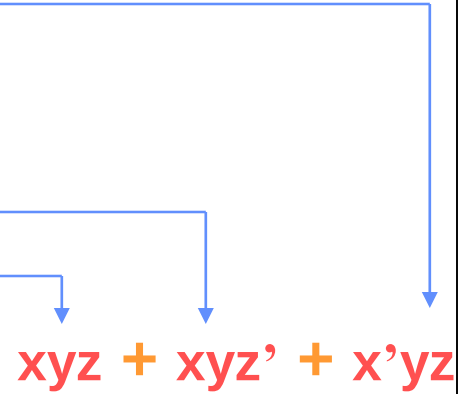
Maxterms Expressed

				Maxterms	
x	y	z		Term	Designation
0	0	0		$x + y + z$	M_0
0	0	1		$x + y + z'$	M_1
0	1	0		$x + y' + z$	M_2
0	1	1		$x + y' + z'$	M_3
1	0	0		$x' + y + z$	M_4
1	0	1		$x' + y + z'$	M_5
1	1	0		$x' + y' + z$	M_6
1	1	1		$x' + y' + z'$	M_7

Truth Table to Expression (Sum of Minterms)

- **Any Boolean function can be expressed as a sum of minterms or sum of products (i.e. the ORing of terms).**
  - **You can form the function algebraically by forming a minterm for each combination of the variables that produces a 1 in the function. (Each row with output of 1 becomes a product term)**
  - **Sum (OR) product terms together.**

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



Sum of Minterms Example

x	y	z		Function F_1	Required Minterms
0	0	0		1	x'y'z'
0	0	1		0	
0	1	0		0	
0	1	1		1	x'yz
1	0	0		1	xy'z'
1	0	1		0	
1	1	0		0	
1	1	1		0	

$$F_1 = x'y'z' + x'yz + xy'z'$$

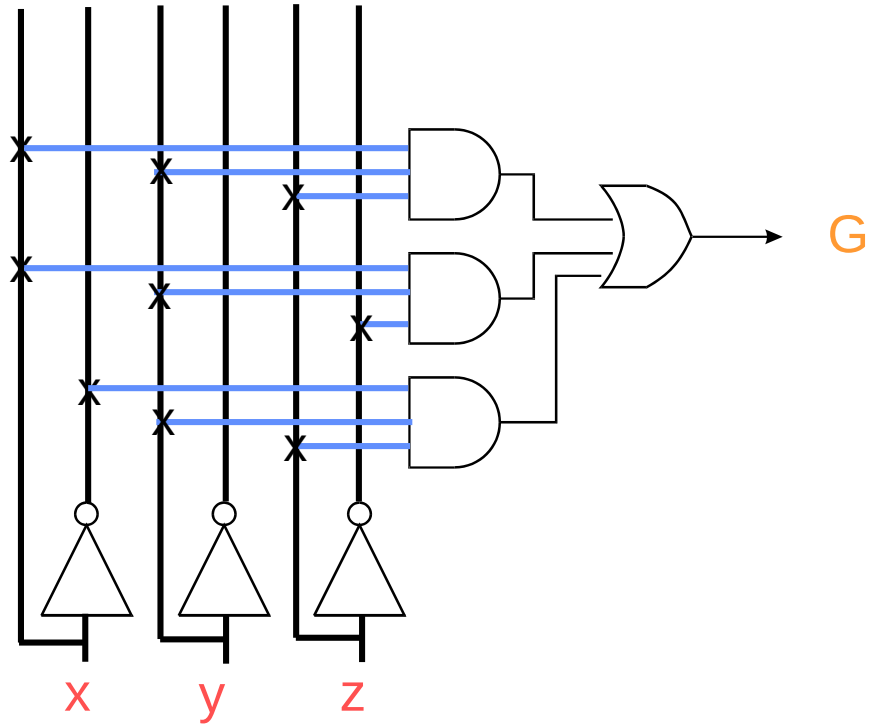
$$= m_0+m_3+m_4$$

$$= \sum(0,3,4)$$

Equivalent Representations of Circuits

- **All three formats are equivalent**
- **Number of 1's in truth table output column equals AND terms for Sum-of-Products (SOP)**

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



G = xyz + xyz' + x'yz

Truth Table to Expression (Product of Maxterms)

- **Any Boolean function can be expressed as a product of maxterms or product of sums (i.e. the ANDing of terms).**
  - **You can form the function algebraically by forming a maxterm for each combination of the variables that produces a 0 in the function. (Each row with output of 0 becomes a standard sums)**
  - **AND these maxterms together.**

Product of Maxterms Example

x	y	z		Function F_1	Required Maxterms
0	0	0		1	
0	0	1		0	$x + y + z'$
0	1	0		0	$x + y' + z$
0	1	1		1	
1	0	0		1	
1	0	1		0	$x' + y + z'$
1	1	0		0	$x' + y' + z$
1	1	1		0	$x' + y' + z'$

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

$$= M_1M_2M_5M_6M_7$$

$$= \prod(1,2,5,6,7)$$

Minterms and Maxterms

- Each variable in a Boolean expression is a literal
- Boolean variables can appear in normal (x) or complement form (x')
- Each AND combination of terms is a minterm
- Each OR combination of terms is a maxterm
- Example:

Minterms

x	y	z	Minterm	
0	0	0	x'y'z'	m_0
0	0	1	x'y'z	m_1
			…	
1	0	0	xy'z'	m_4
			…	
1	1	1	xyz	m_7

Maxterms

x	y	z	Maxterm	
0	0	0	x+y+z	M_0
0	0	1	x+y+z'	M_1
			…	
1	0	0	x'+y+z	M_4
			…	
1	1	1	x'+y'+z'	M_7

Obtaining Sum of Minterms Form

F = A'B + B' + C

A	B	C	F = A'B + B' + C	Required Minterms	Required Designations
0	0	0	1	A'B'C'	m_0
0	0	1	1	A'B'C	m_1
0	1	0	1	A'BC'	m_2
0	1	1	1	A'BC	m_3
1	0	0	1	AB'C'	m_4
1	0	1	1	AB'C	m_5
1	1	0	0		
1	1	1	1	ABC	m_7

F = A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + AB'C+ ABC

= $m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_7$

F(A, B, C) = $\sum(0, 1, 2, 3, 4, 5, 7)$

Obtaining Product of Maxterms

F = A'B + B'C

A	B	C		F = A'B + B'C	Required Maxterms	Required Designations
0	0	0		0	A + B + C	M_0
0	0	1		1		
0	1	0		1		
0	1	1		1		
1	0	0		0	A' + B + C	M_4
1	0	1		1		
1	1	0		0	A' + B' + C	M_6
1	1	1		0	A' + B' + C'	M_7

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

$= M_0 . M_4 . M_6 . M_7$

$F(A, B, C) = \prod(0, 4, 6, 7)$

Canonical Form Conversion

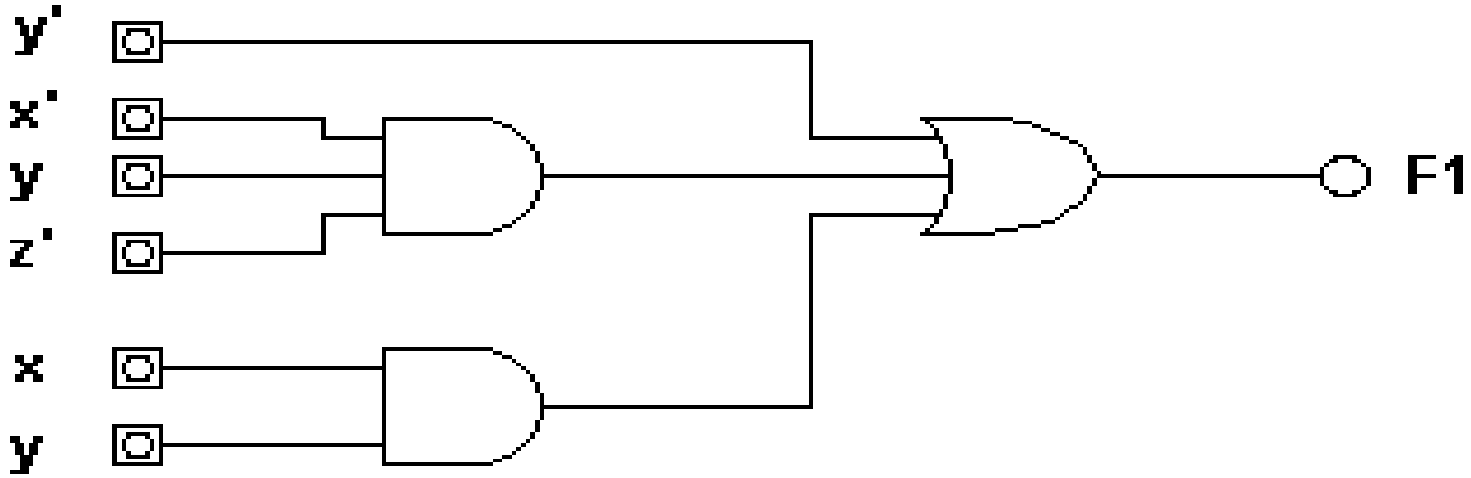
- **A function represented as Sum of minterms can be represented as the Product of maxterms of the remaining terms.**
- **The complement of a function expressed in sum of minterms equals the sum of minterms missing from the original function**
  - **$F(A, B, C) = \sum(0, 3,4) = m_0+m_3+m_4$**
  - **$F'(A, B, C) = \sum(1,2,5,6,7)= m_1+m_2+m_5+m_6+m_7$**
- **Now if we take the complement of F' using DeMorgan's theorem, we obtain F in the product of maxterms form:**
  - **$(F')' = (m_1+m_2+m_5+m_6+m_7)'$**
  - **$F = m_1' . m_2' . m_5' . m_6' . m_7'$ [Complement of minterms]**
  - **$= M_1M_2M_5M_6M_7$ [maxterms]**
  - **$= \prod(1,2, 5, 6, 7)$**
- **This implies the following relation:**
  - **$m'_j = M_j$**
- **So sum of minterms: $\sum(0,3,4)$ = product of maxterms: $\prod(1,2, 5, 6, 7)$**

Standard Forms

- **Standard forms are those forms that allow the terms forming the function to consist of any number of the variables.**
- **There are two standard forms:**
  - **sum of products (SOP)**
  - **product of sums (POS)**

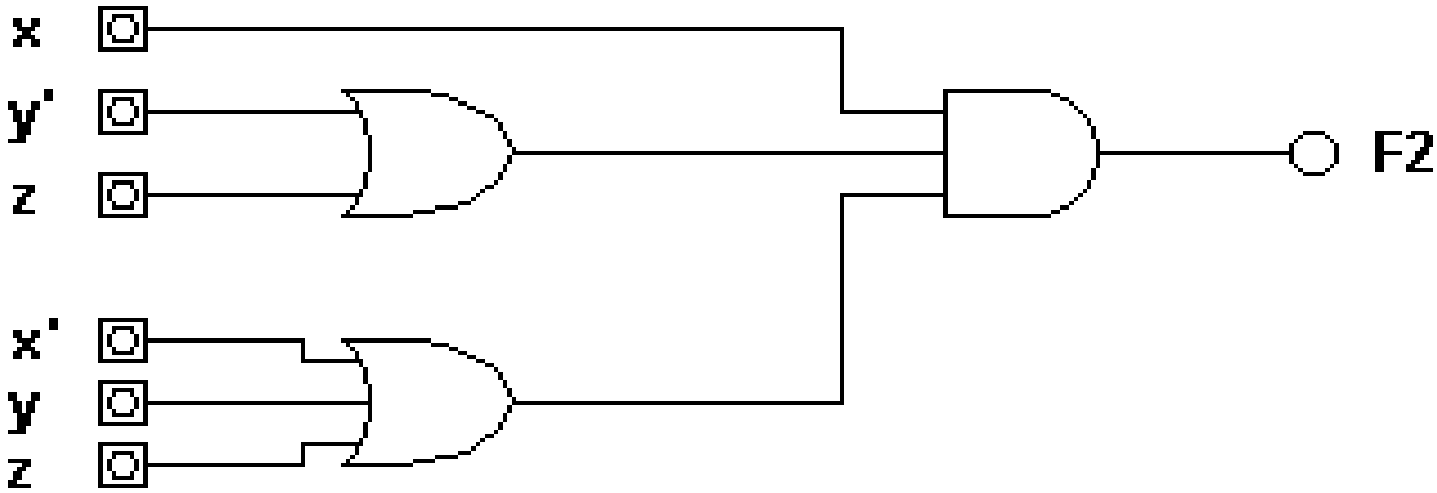
Sum of Products

- **The Sum of Products (SOP) is a Boolean expression containing AND terms, called product terms, of one or more literals each.**
  - **F_1 = y' + xy + x'yz'**



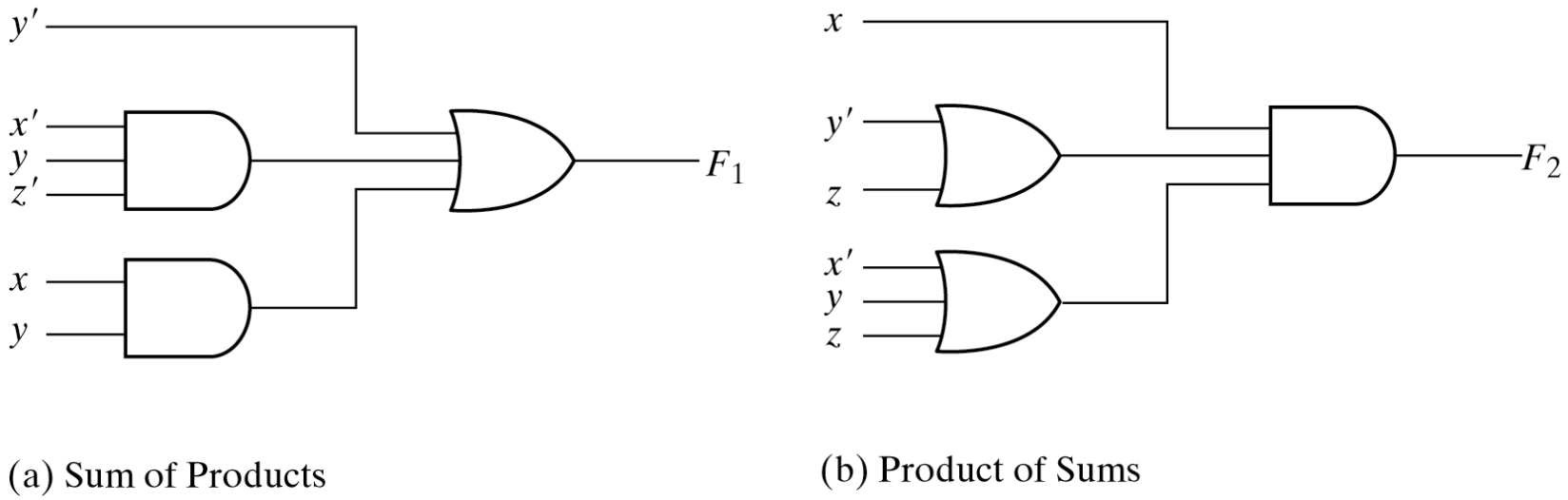
Product of Sums

- **The Product of Sums (POS) is a Boolean expression containing OR terms, called sum terms, of one or more literals each.**
  - $F_2 = x(y' + z)(x' + y + z')$



Two Level Implementation

- **The standard type of expression results in a two-level gating structure**



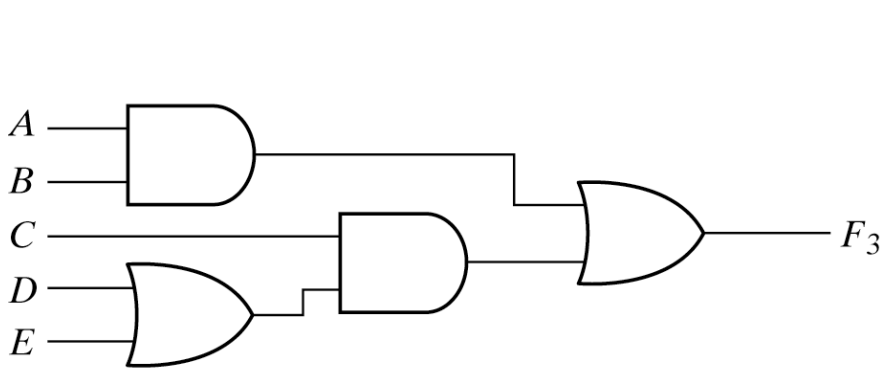
(a) Sum of Products

(b) Product of Sums

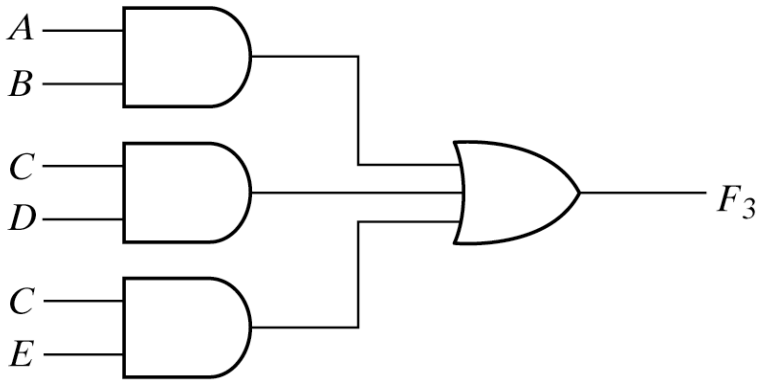
Fig. 2-3 Two-level implementation

Conversion from Nonstandard to Standard Form

- **A Boolean function may be expressed in a nonstandard form (fig 2.4a shows a function that is neither in sum of products nor in product of sums). It has three levels of gating**
- **It can be converted to a standard form (Sum of product) by using distributive law to remove parenthesis**
- **Two-level implementation is preferred as it produces the least amount of delay**



(a) $AB + C(D + E)$



(b) $AB + CD + CE$

Fig. 2-4 Three- and Two-Level implementation

Other Logic Operations

- **Given two Boolean variables:**
  - **When binary operators AND and OR are placed between two variables they form two Boolean functions x . y and x + y**
  - **there are 2^{2x2} = 16 combinations of the two variables as there are 2^{2n} possible functions for n binary variables (we will see the details of these 16 functions in next slides)**
  - **each combination of the variables can result in one of two values, 0 or 1, therefore there are 2^4=16 functions (combinations of 0's and 1's for the four combinations, 00,01,10,11)**
- **AND and OR represent two of the 16 possible functions.**

Function Combinations

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

- **F_1 represents the AND Operation**
- **F_7 represents the OR Operation**
- **There are 14 other functions**

16 Two-Variable Functions

Boolean Function	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 (XNOR)	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

Function Gate Implementations

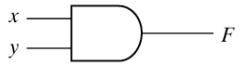
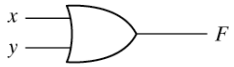
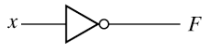
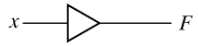
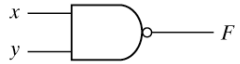
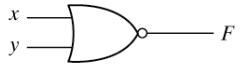
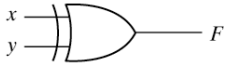
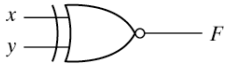
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	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

Fig. 2-5 Digital logic gates

Function Gate Implementations

- **It is easier to implement a Boolean function with these types of gates (as seen on last slide)**
- **Inverter (Complement), Buffer (transfer), AND, OR, NAND, NOR, X-OR, and XNOR (equivalence) are used as standard gates in digital design**
- **NAND and NOR are extensively used logic gates and are more popular than AND and OR gates because these gates are easily constructed with transistor circuits and digital circuits are easily implemented with them**

Multiple Inputs

- **All of the previous defined gates, with the exception of the inverter and the buffer, can have multiple inputs.**
  - **A gate can have multiple inputs provided it is a binary operation that is commutative ($x + y = y + x$ and $xy = yx$) and associative ($x + (y + z) = (x + y) + z$ and $x(yz) = (xy)z$)**
  - **NAND and NOR functions are commutative but not associative**
  - **To overcome this difficulty we define multiple NOR (or NAND) gate as a complemented OR (or AND) gate e.g., as (x+y+z)' or (xyz)'**

Multiple Inputs (Nonassociative NOR operation)

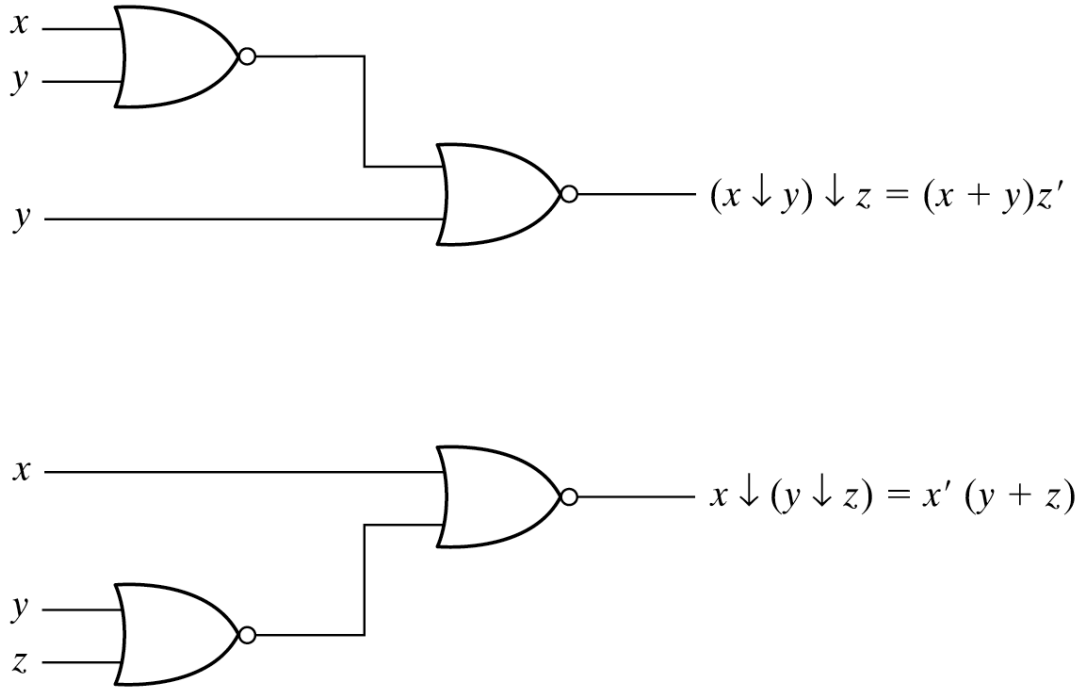
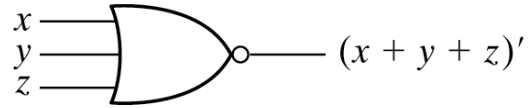
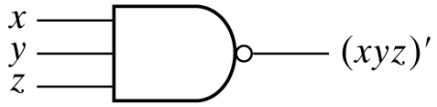


Fig. 2-6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

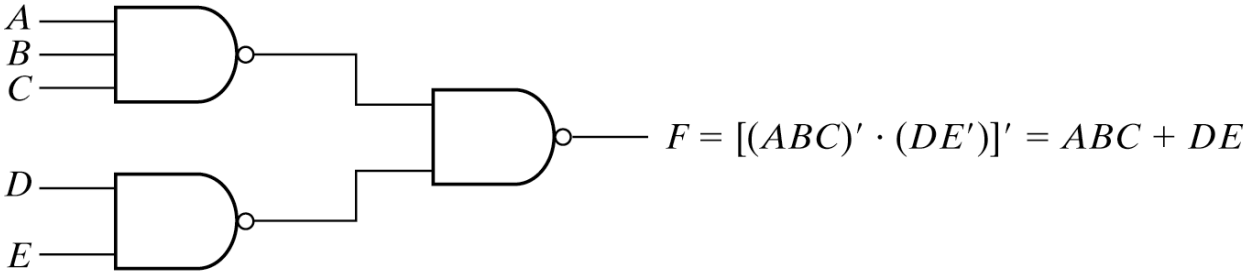
Multiple Inputs NOR and NAND gates



(a) 3-input NOR gate



(b) 3-input NAND gate

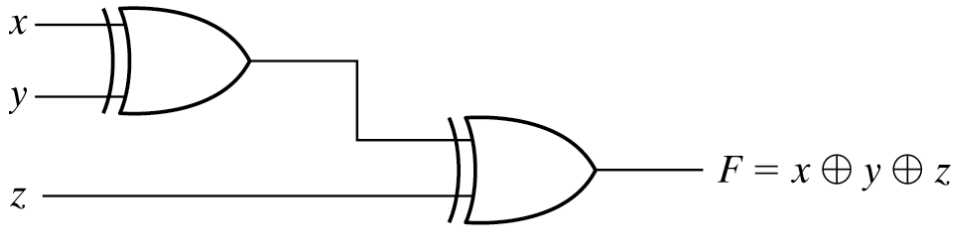


(c) Cascaded NAND gates

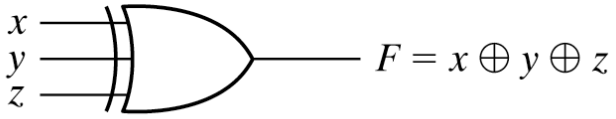
Fig. 2-7 Multiple-input and cascated NOR and NAND gates

Multiple Inputs XOR gate

- **3-input XOR gate is normally implemented by cascading 2-input gates (multiple inputs XOR is uncommon from hardware point)**



(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 and Negative Logic

- **Binary signals in a circuit can have one of two values.**
  - **One signal represents logic-1 and the other logic-0.**
- **A circuit input or output will hold either a high or low signal.**
  - **Choosing the high level, H, to represent logic-1 is called a positive logic system.**
  - **Choosing the low level, L, to represent logic-1 is called a negative logic system**

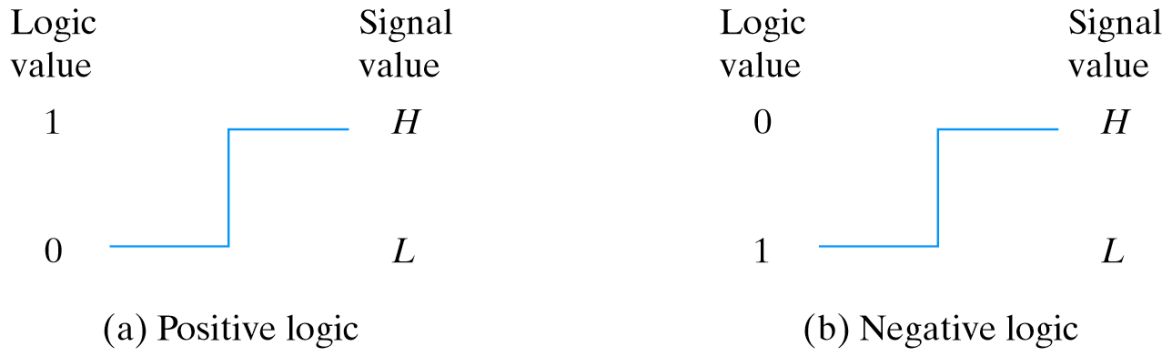
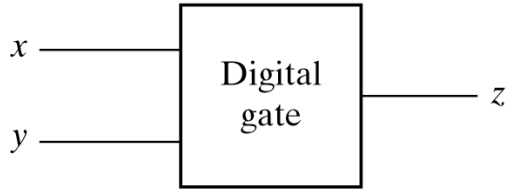


Fig. 2-9 signal assignment and logic polarity

Positive and Negative Logic gates

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

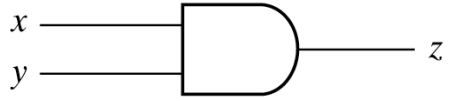
(a) Truth table with H and L



(b) Gate block diagram

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

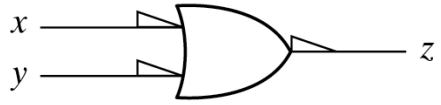
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic



(f) Negative logic OR gate

Fig. 2-10 Demonstration of positive and negative logic

Integrated Circuits

- **An integrated circuit (IC) is a silicon semiconductor crystal, called a chip, containing the electronic components for constructing digital gates.**
  - **Gates are interconnected within the chip to form the required circuit**
  - **The IC is housed inside a ceramic or plastic container with connections welded to external pins**
  - **There can be 14 to several thousand pins on a chip**
  - **Each IC has a numeric designation printed on the surface for identification. The number can be looked up in catalogs (paper and electronic) that contain descriptions and information about the IC**

Levels of Integration

- **ICs are categorized by the number of gates that they contain in them:**
  - **Small-scale integration (SSI) devices contain several (usually less than 10) independent gates in a single package. Early 60's**
  - **Medium-scale integration (MSI) devices include 10 to 1000 gates in a single package, used to perform elementary digital operations. Late 60's**
  - **Large-scale integration (LSI) devices contain thousands of gates in a single package, used in processors, memory chips, and programmable logic devices. Mid 70's**
  - **Very Large-scale integration (VLSI) devices contain hundreds of thousands of gates in a single package, used in large memory arrays and complex microcomputer chips. 80's**
  - **Ultra Large-scale integration (ULSI) devices contain Millions of gates in a single package. 90's and 00's**

Digital Logic Families

- **ICs are also classified by the specific circuit technology (digital logic family) that they belong to:**
  - Transistor-transistor logic (TTL) **is a standard.**
  - Emitter-coupled logic (ECL) **is used in high-speed operation.**
  - Metal-oxide semiconductor (MOS) **is used for high component density.**
  - Complementary metal-oxide semiconductor (CMOS) **is used in low power consumption.**

Logic Family Characteristics

- **Digital logic families are usually compared by the following characteristics:**
  - Fan-out **specifies the number of standard loads that the output of a gate can drive without impairing its normal operation or it specifies the amount of current that an output needs to drive many input pins on other gates.**
  - Fan-in **is the number of inputs available in a gate.**
  - Power dissipation **is the power consumed by the gate.**
  - Propagation delay **is the average delay time for the signal to propagate from input to output.**
  - Noise margin **is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.**
  - Real estate **is the amount of space required to implement the IC.**
  - Reliability **is the long-term success factor of the IC.**

Integrated Circuits Design

- **Why is it better to have more gates on a single chip?**
  - Easier to build systems
  - Less power consumption
  - Higher clock frequencies
- **What are the drawbacks of large circuits?**
  - Complex to design
  - Chips have design constraints
  - Need tools to help develop integrated circuits
- **Need tools to help develop integrated circuits**
  - Computer Aided Design (CAD) tools
  - Automate tedious steps of design process
  - Hardware description language (HDL) describe circuits

End of Chapter 2