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 \(\forall x \in S, e * x = x * e = x\).
   • Set of integers: \(e = 1\) w.r.t. \(\times\) and \(e = 0\) w.r.t. +
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 \( x * y = e \)

- Example: set of integers w.r.t. +

6. Distributive law

- If * and \( \bullet \) are two binary operators on S
  \[ x * (y \bullet z) = (x * y) \bullet (x * z) \]
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 ∈ B s. t. x ≠ y

• Binary operators: + and ·
  closure w.r.t. both + and ·
    x, y ∈ B, (x+y) ∈ B, (x ·y) ∈ B
  additive identity?
    0 : x + 0 = 0 + x = x
  multiplicative identity?
    1 : x ·1 = 1 ·x = x
  commutative w.r.t. both + and ·
  associative w.r.t. both + and ·

• Distributive law:
  is · distributive over + ?
    yes : x · (y+z) = (x · y) + (x · z)
  is + distributive over · ?
    yes : x + (y · z) = (x+y) · (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 $x' \in B \ni$
    a. $x + x' = 1$ (multiplicative identity) and
    b. $x \cdot x' = 0$ (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
  $B = \{0, 1\}$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \cdot y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x + y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Two-Valued Boolean Algebra 2/3

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

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$y \cdot z$</th>
<th>$x^+ (y \cdot z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x + y$</th>
<th>$x + z$</th>
<th>$(x + y) \cdot (x + z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ x \cdot (y + z) = (x \cdot y) + (x \cdot z) \]

\[ ((x + y) \cdot (x + z)) \cdot (x + y) = x + y \]

\[ x^+ (x \cdot y) = x^+ (y \cdot x^+) = x^+ \cdot (x^+ \cdot y^+) \]

\[ x + (x^+) \cdot (y + (y^+) \cdot z) = x + (x^+) \cdot (y + (y^+) \cdot z) \]

\[ x + y + z = x + (y + z) = (x + y) + z \]

\[ x \cdot y \cdot z = x \cdot (y \cdot z) = (x \cdot y) \cdot z \]
Two-Valued Boolean Algebra 3/3

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

  Operations:
  - $\cdot \rightarrow \text{AND}$
  - $+ \rightarrow \text{OR}$
  - Complement $\rightarrow \text{NOT}$

• Binary logic is the application of Boolean algebra to gate-type circuits
  - Two-valued Boolean algebra is developed in a formal mathematical manner
  - This formalism is necessary to develop theorems and properties of Boolean algebra
Duality Principle

• An important principle
  – every algebraic expression deducible from the axioms of Boolean algebra remains valid if the operators and identity elements are interchanged 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
    $x + x = x \quad \Rightarrow \quad x \cdot x = x$
Duality Principle & Theorems

- **Theorem a:**
  - $x + 1 = 1$  
  - $x + 1 = 1 \cdot (x + 1)$
  - $= (x + x')(x + 1)$
  - $= x + x' \cdot 1$
  - $= x + x'$
  - $= 1$

- **Theorem b:** (using duality)
  - $x \cdot 0 = 0$
Absorption Theorem

\[ x + xy = x \quad \text{hmmmmm?} \]

\[ = x \cdot 1 + xy \]

\[ = x(1+y) \]

\[ = x(1) \]

\[ = x \]
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

\[(x + y)' = x' \cdot y'\]

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>x+y</th>
<th>(x+y)'</th>
<th>x' \cdot y'</th>
<th>(x \cdot y)'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x'</th>
<th>y'</th>
<th>x' \cdot y'</th>
<th>x' + y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Operator Precedence
(Boolean Operator Priority Order)

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

PiNAOr?
PiNA cOlada?
Boolean Functions

• Consists of
  – binary variables (normal or complement form)
  – the constants, 0 and 1
  – logic operation symbols “+” and “·”

• Example:
  ▪ \( F_1(x, y, z) = x + y' \cdot z \)
  ▪ \( F_2(x, y, z) = x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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 $F_2$

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

\[ f = (x + y \cdot \bar{z}) \cdot (y \cdot z) \]
Example

\[
f = (x + y \cdot \overline{z}) \cdot (y \cdot z)
\]
\[
= (x + y \cdot \overline{z}) \cdot (\overline{y} + \overline{z})
\]
\[
= x \cdot (\overline{y} + \overline{z}) + (y \cdot \overline{z}) \cdot (\overline{y} + \overline{z})
\]
\[
= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{y} + y \cdot \overline{z} \cdot \overline{z}
\]
\[
= x \cdot \overline{y} + x \cdot \overline{z} + 0 \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{z}
\]
\[
= x \cdot \overline{y} + x \cdot \overline{z} + 0 + y \cdot \overline{z} \cdot \overline{z}
\]
\[
= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z}
\]
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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
<th>F₄</th>
<th>F₅</th>
<th>F₆</th>
<th>F₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F₈</th>
<th>F₉</th>
<th>F₁₀</th>
<th>F₁₁</th>
<th>F₁₂</th>
<th>F₁₃</th>
<th>F₁₄</th>
<th>F₁₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
OTHER LOGIC OPERATORS - 2

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

- **NOT**
- **AND**
- **OR**
- **NAND**
- **NOR**
- **TRANSFER**
- **XOR**
- **XNOR**
Universal Gates

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>(xy)'</th>
<th>x'</th>
<th>y'</th>
<th>(x' y')'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
NAND Gate

\[ x \quad \text{NAND Gate} \quad y \]
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)’ ≠ (x(yz)’)’
  ▪ ((xy)’z)’ = xy + z’
  ▪ (x(yz)’)’ = x’ + yz
Nonassociativity of NOR operation

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

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

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

\[(x + y + z)'

3-input NAND gate

\[(xyz)'

Cascaded NAND gates

\[F = \text{Cascaded NAND gates}\]
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

\[ F = x \oplus y \oplus z \]

(a) Using 2-input gates

(b) 3-input gate

(c) Truth table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</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

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$F$</th>
<th>$F'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

$F =$

$F' =$
Generalizing DeMorgan’s Theorem

• We can also utilize DeMorgan’s Theorem
  ▪ \((x + y)' = x' \cdot y'\)
  ▪ \((A + B + C)' = (A+B)' \cdot C' = A' \cdot B' \cdot C'\)

• 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

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

$F(x, y, z) = ?$ in minterms

$F(x, y, z) = ?$ in maxterms
### Example

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

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

$F(x, y, z) = (x+y+z)(x+y'+z')(x'+y+z)(x'+y+z')(x'+y'+z')$
### Min- & Maxterms with n = 3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th><strong>Minterms</strong></th>
<th></th>
<th></th>
<th><strong>Maxterms</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>term</strong></td>
<td></td>
<td></td>
<td><strong>term</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(x'y'z')</td>
<td></td>
<td></td>
<td>(x + y + z)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(x'y'z)</td>
<td></td>
<td></td>
<td>(x + y + z')</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(x'yz')</td>
<td></td>
<td></td>
<td>(x + y' + z)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(x'yz)</td>
<td></td>
<td></td>
<td>(x + y' + z')</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(xy'z')</td>
<td></td>
<td></td>
<td>(x' + y + z)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(xy'z)</td>
<td></td>
<td></td>
<td>(x' + y + z')</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(xyz')</td>
<td></td>
<td></td>
<td>(x' + y' + z)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(xyz)</td>
<td></td>
<td></td>
<td>(x' + y' + z')</td>
<td></td>
</tr>
</tbody>
</table>
## Formal Expression with Minterms

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

\[
F(x, y, z) = F(0,0,0)m_0 + F(0,0,1)m_1 + F(0,1,0)m_2 + F(0,1,1)m_3 + F(1,0,0)m_4 + F(1,0,1)m_5 + F(1,1,0)m_6 + F(1,1,1)m_7
\]
Formal Expression with Maxterms

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

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• $F₁(x, y, z) =$
• $F₂(x, y, z) =$
Important Properties

• Any Boolean function can be expressed
  ▪ as a sum of minterms
  ▪ as a product of maxterms

• Example:
  ▪ \( F' = \Sigma (0, 2, 3, 5, 6) \)
  
  =
  
  ▪ 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:
  ▪ $F' = \sum (0, 2, 3, 5, 6)$
    \[= 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:
  - \( F' = \sum (0, 2, 3, 5, 6) \)
    \[ = x'y'z' + x'yz' + x'yz + xy'z + xyz' \]
  - 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.

  \[ F = x + y'z = \sum (7, 6, 5, 4, 1) \]

• Alternative way:
  - Obtain the truth table first and then the canonical term.
Example: Product of Maxterms

- $F = xy + x'z$
- Use the distributive law + over $\cdot$
- $F = xy + x'z$
  - $= xy(z + z') + x'z(y + y')$
  - $= xyz + xyz' + x'yz + x'y'z$
  - $= \sum (7, 6, 3, 1)$
Example: Product of Maxterms

- \( F = xy + x'z \)
  - Use the distributive law + over \( \cdot \)
  - \( F = xy + x'z \)
    - \( = xy(z+z') + x'z(y+y') \)
    - \( = xyz + xyz' + x'yz + x'y'z \)
    - \( = \Sigma (7,6,3,1) \)
  - \( = \Pi (4,5,0,2) \)
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
    ▪ $(F')' =$
    ▪ $F = M_0 \cdot M_2 \cdot M_3 = \Pi (0, 2, 3)$
General Rule for Conversion

- **Important relation:**
  - $m_j' = M_j$
  - $M_j' = m_j$

- **The rule:**
  - Interchange symbols $\Pi$ and $\Sigma$, and
  - list those terms missing from the original form

- **Example:** $F = xy + x'z$

$$F = \Sigma(1, 3, 6, 7) \Rightarrow F = \Pi(?, ?, ?, ?)$$
Standard Forms

• Fact:
  – Canonical forms are very seldom the ones with the least number of literals

• Alternative representation:
  – Standard form
    • a term may contain any number of literals
  – Two types
    1. the sum of products
    2. the product of sums
  – Examples:
    • \( F_1 = y’ + xy + x’yz’ \)
    • \( F_2 = x(y’ + z)(x’ + y + z’) \)
Example: Standard Forms

- $F_1 = y' + xy + x'y'z'$
- $F_2 = x(y' + z)(x' + y + z')$
Example: Standard Forms

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

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 $\rightarrow$ logic-1
    • L $\rightarrow$ logic-0

![Diagram showing signal assignment and logic polarity](image-url)
• What kind of logic function does it implement?
Signal Designation - 2

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

positive logic

greater than

negative logic
Another Example

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
</tbody>
</table>

74LS00
Integrated Circuits

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

• SSI (small-scale integration):
  – 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 \(\rightarrow\) transistor-transistor logic
    • has been in use for 50 years and is considered to be standard.
  – ECL \(\rightarrow\) Emitter-coupled logic
    • Fast. Has an advantage in systems requiring high-speed operation.
  – MOS \(\rightarrow\) metal-oxide semiconductor
    • suitable for circuits that need high component density,
  – CMOS \(\rightarrow\) 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
Schematic Editor

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

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

\[
F (x, y, z, t) = xz' + yz't' + xyt'
\]
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
Simulation Results 2/3
Simulation Results 3/3