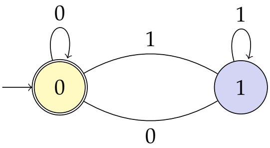
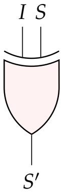
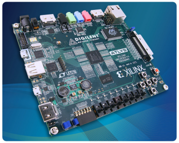
CS 226: Digital Logic Design

Lecture 4: Introduction to Logic Circuits

Ashutosh Trivedi







Department of Computer Science and **Engineering**,
Indian Institute of Technology Bombay.

Objectives

In this lecture we will introduce:

1. Logic functions and circuits,
2. Boolean algebra for manipulating with logic functions,
3. Logic gates, and
4. Synthesis of simple logic circuits.

Objectives

Logic functions and circuits

Boolean Algebra

Synthesis of Simple Circuits

Introduction to CAD tools

Variables and Functions

- Logic circuits form the foundation of digital systems
- Binary logic circuits perform operations of binary signals

Variables and Functions

- Logic circuits form the foundation of digital systems
- Binary logic circuits perform operations of binary signals
- Let's consider the simplest element logic circuit: switch.
  - A switch can be in two states: "open" and "close".
  - Graphical symbol for a switch
  - A simple application using a switch : "lightbulb"

Variables and Functions

- Logic circuits form the foundation of digital systems
- Binary logic circuits perform operations of binary signals
- Let's consider the simplest element logic circuit: switch.
  - A switch can be in two states: "open" and "close".
  - Graphical symbol for a switch
  - A simple application using a switch : "lightbulb"
  - State of the switch can be given as a binary variable x s.t.:
    - $x = 0$ when switch is open, and
    - $x = 1$ when switch is closed.
  - The function of a switch:
    - $F = 0$ when $x = 0$ and
    - $F = 1$ when $x = 1$.
  - In other words $F(x) = x$.

Variables and Functions

- Logic circuits form the foundation of digital systems
- Binary logic circuits perform operations of binary signals
- Let's consider the simplest element logic circuit: switch.
  - A switch can be in two states: "open" and "close".
  - Graphical symbol for a switch
  - A simple application using a switch : "lightbulb"
  - State of the switch can be given as a binary variable x s.t.:
    - $x = 0$ when switch is open, and
    - $x = 1$ when switch is closed.
  - The function of a switch:
    - $F = 0$ when $x = 0$ and
    - $F = 1$ when $x = 1$.
  - In other words $F(x) = x$.
- Keywords: Logic expression defining output, logic function, and input variable.

Variables and Functions

- Logic circuits form the foundation of digital systems
- Binary logic circuits perform operations of binary signals
- Let's consider the simplest element logic circuit: switch.
  - A switch can be in two states: "open" and "close".
  - Graphical symbol for a switch
  - A simple application using a switch : "lightbulb"
  - State of the switch can be given as a binary variable x s.t.:
    - $x = 0$ when switch is open, and
    - $x = 1$ when switch is closed.
  - The function of a switch:
    - $F = 0$ when $x = 0$ and
    - $F = 1$ when $x = 1$.
  - In other words $F(x) = x$.
- Keywords: Logic expression defining output, logic function, and input variable.
- Let's use two switches to control the lightbulb by connecting them in series and parallel.

Introducing AND nd OR

- Logical AND operation (series connection)
  - We write the expression

$$L(x,y) = x \cdot y$$

to say $L(x,y) = x$ AND y when

$$L = \begin{cases} 1 \text{ if } x = 1 \text{ and } y = 1. \\ 0 \text{ otherwise.} \end{cases}$$

Introducing AND nd OR

- Logical AND operation (series connection)
  - We write the expression

$$L(x, y) = x \cdot y$$

to say $L(x, y) = x$ AND y when

$$L = \begin{cases} 1 \text{ if } x = 1 \text{ and } y = 1. \\ 0 \text{ otherwise.} \end{cases}$$

- Logical OR operation (parallel connection)
  - We write the expression

$$L(x, y) = x + y$$

to say $L(x, y) = x$ OR y when

$$L = \begin{cases} 0 \text{ if } x = 0 \text{ and } y = 0. \\ 0 \text{ otherwise.} \end{cases}$$

Introducing AND nd OR

- Logical AND operation (series connection)
  - We write the expression

$$L(x, y) = x \cdot y$$

to say $L(x, y) = x$ AND y when

$$L = \begin{cases} 1 \text{ if } x = 1 \text{ and } y = 1. \\ 0 \text{ otherwise.} \end{cases}$$

- Logical OR operation (parallel connection)
  - We write the expression

$$L(x, y) = x + y$$

to say $L(x, y) = x$ OR y when

$$L = \begin{cases} 0 \text{ if } x = 0 \text{ and } y = 0. \\ 0 \text{ otherwise.} \end{cases}$$

- A series-parallel connection

Inversion

- Can we use a switch to implement an "inverted switch"?
- We would like to implement the following function:

$$L(x) = \overline{x}$$

to say $L(x) =$ NOT x or "complement" of x when

$$L \quad = \quad \begin{cases} 0 \text{ if } x = 1. \\ 1 \text{ if } x = 0 \end{cases}$$

Inversion

- Can we use a switch to implement an "inverted switch"?
- We would like to implement the following function:

$$L(x) = \overline{x}$$

to say $L(x) =$ NOT x or "complement" of x when

$$L = \begin{cases} 0 \text{ if } x = 1. \\ 1 \text{ if } x = 0 \end{cases}$$

- The inverting circuit

Inversion

- Can we use a switch to implement an “inverted switch”?
- We would like to implement the following function:

$$L(x) = \overline{x}$$

to say $L(x) =$ NOT x or “complement” of x when

$$L = \begin{cases} 0 \text{ if } x = 1. \\ 1 \text{ if } x = 0 \end{cases}$$

- The inverting circuit
- We can complement not only variables but also expressions, s.t.

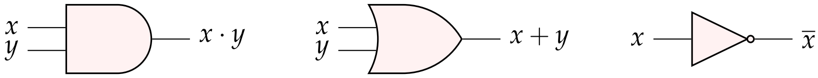
$$f(x, y) = \overline{x + y}.$$

Specification of a logical function: Truth tables

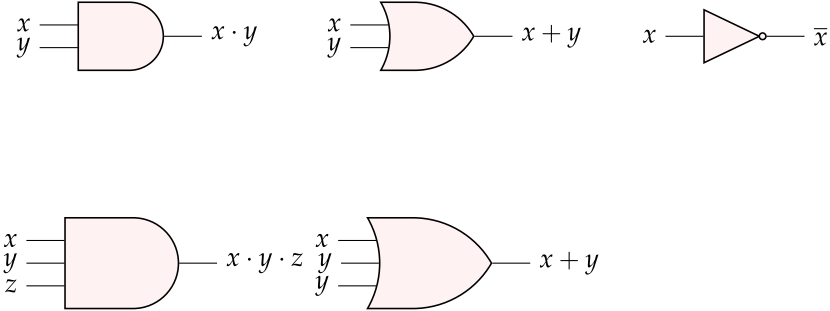
- The specification of a logical function can be enumerate as a truth-table
- Since input variables are finite, there are only finitely many possible combinations for a finite set of inputs.
- Example of truth table for $x \cdot y$, $x + y$ and $\overline{x}$.
- Truth-table for n-input AND, OR, or NOT operations

Logic Gates

- A “switch” can be implemented using a transistor.
- Also other Boolean operations can be implemented using various combinations of switches.
- AND gate, OR gate, NOT gate represent encapsulation of transistor circuits implementing Boolean function
-



Logic Gates

- A “switch” can be implemented using a transistor.
- Also other Boolean operations can be implemented using various combinations of switches.
- AND gate, OR gate, NOT gate represent encapsulation of transistor circuits implementing Boolean function
- 

Analysis and Synthesis of a Logic Network

– Tasks of a designer of a digital system: analysis and synthesis

Analysis and Synthesis of a Logic Network

- Tasks of a designer of a digital system: analysis and synthesis
- Let's analyze the logic network $f(x_1, x_2, x_3) = \overline{x_1} + x_2 \cdot x_3$.
- Construct its Truth-Table

Analysis and Synthesis of a Logic Network

- Tasks of a designer of a digital system: analysis and synthesis
- Let's analyze the logic network $f(x_1, x_2, x_3) = \overline{x_1} + x_2 \cdot x_3$.
- Construct its Truth-Table
- Timing diagram of a circuit
- Functionally equivalent circuits: consider the network that implement Boolean function $g(x_1, x_2) = \overline{x_1} + x_2$. and compare with the function $f(x_1, x_2)$.

Analysis and Synthesis of a Logic Network

- Tasks of a designer of a digital system: analysis and synthesis
- Let's analyze the logic network $f(x_1, x_2, x_3) = \overline{x_1} + x_2 \cdot x_3$.
- Construct its Truth-Table
- Timing diagram of a circuit
- Functionally equivalent circuits: consider the network that implement Boolean function $g(x_1, x_2) = \overline{x_1} + x_2$. and compare with the function $f(x_1, x_2)$.
- How do we know if two functions are logically equivalent?

Analysis and Synthesis of a Logic Network

- Tasks of a designer of a digital system: analysis and synthesis
- Let's analyze the logic network $f(x_1, x_2, x_3) = \overline{x_1} + x_2 \cdot x_3$.
- Construct its Truth-Table
- Timing diagram of a circuit
- Functionally equivalent circuits: consider the network that implement Boolean function $g(x_1, x_2) = \overline{x_1} + x_2$. and compare with the function $f(x_1, x_2)$.
- How do we know if two functions are logically equivalent?
- How many different logically equivalent functions over n-variables?

Analysis and Synthesis of a Logic Network

- Tasks of a designer of a digital system: analysis and synthesis
- Let's analyze the logic network $f(x_1, x_2, x_3) = \overline{x_1} + x_2 \cdot x_3$.
- Construct its Truth-Table
- Timing diagram of a circuit
- Functionally equivalent circuits: consider the network that implement Boolean function $g(x_1, x_2) = \overline{x_1} + x_2$. and compare with the function $f(x_1, x_2)$.
- How do we know if two functions are logically equivalent?
- How many different logically equivalent functions over n-variables?
- How to minimize circuit complexity (and cost)?

Objectives

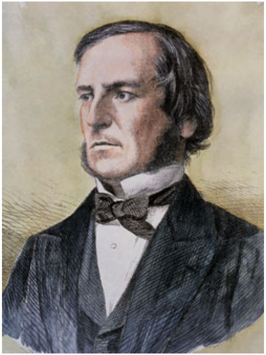
Logic functions and circuits

Boolean Algebra

Synthesis of Simple Circuits

Introduction to CAD tools

Boolean Algebra



- In 1849 George Boole introduced algebra for manipulating processes involving logical thoughts and reasoning
- This scheme, with some refinements, is not know as Boolean algebra.
- Claude Shannon in 1930 showed that Boolean algebra is awesome for describing circuits with switches!
- It is, then, of course awesome to describe logical circuits.
- Let's see how it is a powerful tool to design and analyze logical circuits.

Boolean Algebra: Axioms

1.

$$\begin{aligned} 0 \cdot 0 &= 0 && (1) \\ 1 + 1 &= 1 && (2) \end{aligned}$$

2.

$$\begin{aligned} 1 \cdot 1 &= 1 && (3) \\ 0 + 0 &= 0 && (4) \end{aligned}$$

3.

$$\begin{aligned} 0 \cdot 1 &= 1 \cdot 0 = 0 && (5) \\ 0 + 1 &= 1 + 0 = 1 && (6) \end{aligned}$$

4.

$$\begin{aligned} &\text{If } x = 0 \quad \text{then} \quad \overline{x} = 1 && (7) \\ &\text{If } x = 1 \quad \text{then} \quad \overline{x} = 0 && (8) \end{aligned}$$

Boolean Algebra: Single variable Theorems

1.

$$x \cdot 0 = 0 \tag{9}$$
$$x + 1 = 1 \tag{10}$$

2.

$$x \cdot 1 = x \tag{11}$$
$$x + 0 = x \tag{12}$$

3.

$$x \cdot x = x \tag{13}$$
$$x + x = x \tag{14}$$

4.

$$x \cdot \overline{x} = 0 \tag{15}$$
$$x + \cdot \overline{x} = 1 \tag{16}$$

5.

$$\overline{\overline{x}} = x \tag{17}$$

Boolean Algebra: 2- and 3- Variable Properties

1. Commutativity:

$$
\begin{aligned}
x \cdot y &= y \cdot x \quad (18)\\
x + y &= y + x \quad (19)
\end{aligned}
$$

2. Associativity:

$$
\begin{aligned}
x \cdot (y \cdot z) &= (x \cdot y) \cdot z \quad (20)\\
x + (y + z) &= (x + y) + z \quad (21)
\end{aligned}
$$

3. Distributivity:

$$
\begin{aligned}
x \cdot (y \cdot z) &= (x \cdot y) \cdot z \quad (22)\\
x + (y + z) &= (x + y) + z \quad (23)
\end{aligned}
$$

4. Absorption:

$$
\begin{aligned}
x + x \cdot y &= x \quad (24)\\
x \cdot (x + y) &= x \quad (25)
\end{aligned}
$$

Boolean Algebra: 2- and 3- Variable Properties

1. Combining:

$$x \cdot y + x \cdot \overline{y} = x \tag{26}$$

$$(x + y) \cdot (x + \overline{y}) = x \tag{27}$$

2. Consensus:

$$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z \tag{28}$$

$$(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z) \tag{29}$$

Boolean Algebra: DeMorgan's theorem

Augustus De Morgan (27 June 1806 — 18 March 1871)

Boolean Algebra: DeMorgan's theorem

Augustus De Morgan (27 June 1806 — 18 March 1871)

$$\overline{x \cdot y} = \overline{x} + \overline{y} \tag{30}$$

$$\overline{x + y} = \overline{x} \cdot \overline{y} \tag{31}$$

Proof by Perfect Induction (Truth-tables)

Other Remarks

– Venn diagram are quite useful proving theorems in Boolean algebra

Other Remarks

- Venn diagram are quite useful proving theorems in Boolean algebra
- Common symbols to represent OR: $x \vee y$ and $x + y$
- Common symbols to represent AND: $x \wedge y$ and $x \cdot y$
- Common symbols to represent NOT: $\neg x$ and $\overline{x}$

Other Remarks

- Venn diagram are quite useful proving theorems in Boolean algebra
- Common symbols to represent OR: $x \vee y$ and $x + y$
- Common symbols to represent AND: $x \wedge y$ and $x \cdot y$
- Common symbols to represent NOT: $\neg x$ and $\overline{x}$
- Precedence of Operations:

$$NOT > AND > OR$$

Example

- $x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$
- $(x_1 \cdot x_2) + ((\overline{x_1}) \cdot (\overline{x_2}))$
- $x_1 \cdot (x_2 + \overline{x_1}) \cdot \overline{x_2}$

Objectives

Logic functions and circuits

Boolean Algebra

Synthesis of Simple Circuits

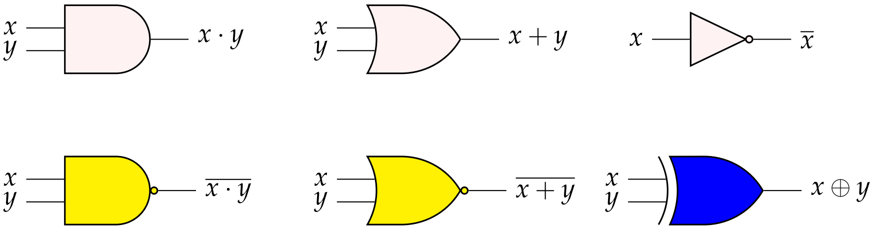
Introduction to CAD tools

Synthesis of simple circuits

Definition (Synthesis)

Given a description of the desired functional behavior, the synthesis is the process to generate a circuit that realizes this behavior.

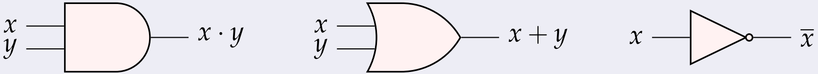
Commonly Used logic Gates:



Synthesis of simple circuits

Theorem

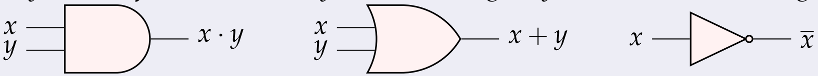
Any Boolean function can be synthesized using only AND, OR, and NOT gates.



Synthesis of simple circuits

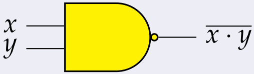
Theorem

Any Boolean function can be synthesized using only AND, OR, and NOT gates.



Theorem

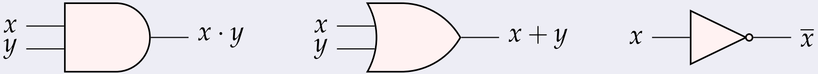
Any Boolean function can be synthesized using only NAND gates.



Synthesis of simple circuits

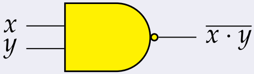
Theorem

Any Boolean function can be synthesized using only AND, OR, and NOT gates.



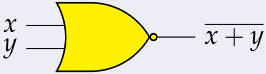
Theorem

Any Boolean function can be synthesized using only NAND gates.



Theorem

Any Boolean function can be synthesized using only NOR gates.



Synthesis using AND, OR, and NOT gates

Given a Boolean function f given in the form of a truth table, the expression that realizes f can be obtained:

- (SUM-OF-PRODUCTS) by considering rows for which $f = 1$, or
- (PRODUCTS-OF-SUMS) by considering rows for which $f = 0$.

Synthesis using AND, OR, and NOT gates

Given a Boolean function f given in the form of a truth table, the expression that realizes f can be obtained:

- (SUM-OF-PRODUCTS) by considering rows for which $f = 1$, or
- (PRODUCTS-OF-SUMS) by considering rows for which $f = 0$.

Example:

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	0
1	1	1

$$
\begin{aligned}
f(x_1, x_2) &= \overline{x_1}x_2 + x_1x_2 = m_1 + m_3 \\
f(x_1, x_2) &= \overline{(\overline{x_1}\overline{x_2} + \overline{x_1}x_2)} = \overline{(\overline{x_1}\overline{x_2})} \cdot \overline{(\overline{x_1}x_2)} = (x_1 + x_2) \cdot (x_1 + \overline{x_2}) = M_0 \cdot M_2
\end{aligned}
$$

Sum-of-Product Canonical Form

- For a function of n variables a minterm is a product term in which each of the variable occur exactly once. For example: $x_0 \cdot x_1 \cdot \overline{x_2}$ and $\overline{x_0} \cdot \overline{x_1} \cdot x_2$.
- A function f can be represented by an expression that is sum of minterms.
- A logical expression that consists of product terms that are summed (ORed) together is called a sum-of-product form.
- If each of the product term is a minterm, then the expression is called canonical sum-of-product expression.
- Notice that two logically equivalent functions will have the same canonical representation.
- For a truth-table of n variables, we represent the minterm corresponding to the i-th row as m_i where $i \in \{0, 2^n - 1\}$.
- A unique representation of a function can be given as an explicit sum of minterms for rows for which function is true.
- For example $f(x_1, x_2) = \overline{x_1}x_2 + x_1x_2 = m_1 + m_3 = \sum(m_1, m_3) = \sum m(1, 3)$.

Sum-of-Product Canonical Form

- For a function of n variables a maxterm is a sum term in which each of the variable occur exactly once. For example

$$(x_0 + x_1 + \overline{x_2}) \text{ and } (\overline{x_0} + \overline{x_1} + x_2).$$

- A function f can be represented by an expression that is product of maxterms.
- A logical expression that consists of sum terms that are factors of logical product (AND) is called a product-of-sum form.
- If each of the sum term is a maxterm, then the expression is called canonical product-of-sum expression.
- Notice that two logically equivalent functions will have the same canonical product-of-sum representation.
- For a truth-table of n variables, we represent the maxterm corresponding to $\overline{\text{complement of the minterm of the } i\text{-th row}}$ as M_i where $i \in \{0, 2^n - 1\}$.
- A unique representation of a function can be given as an explicit product of maxterms for rows for which function is false.
- For example $f(x_1, x_2) = \overline{(\overline{x_1 x_2} + \overline{x_1} x_2)} = \prod(M_0, M_2) = \prod M(0, 2)$.

Questions

- Give SOP form of the function $f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$ and simplify it.
- Give POS for of the function $f(x_1, x_2, x_3) = \prod M(0, 1, 5)$ and simplify it.

NAND and NOR logic networks

- De Morgan's theorem in terms of logic gates
- Using NAND gates to implement a sum-of-products
- Using NOR gates to implement a product-of-sums

Examples

Design the logic circuits for the following problems:

- Three-way light control
- Multiplexer circuit

Objectives

Logic functions and circuits

Boolean Algebra

Synthesis of Simple Circuits

Introduction to CAD tools

Steps in Design process

1. Design Entry
   - Schematic capture
   - Hardware description language
2. Synthesis (or translating/ compiling)
3. Functional Simulation
4. Physical Design
5. Timing Simulation
6. Chip configuration (programming)

Introduction to VHDL