Logic Design

Chapter 2: Introduction to Logic Circuits

Introduction

• Logic circuits operate on digital signals

• Unlike continuous analog signals that have an infinite number of possible values, digital signals are restricted to a few discrete values

• In particular for binary logic circuits, signals can have only two values: 0 and 1.

Logic Operations

The fundamental logic operations are:

• AND \( F = X \cdot Y \)

• OR \( F = X + Y \)

• NOT \( F = X' \) (complement)

\( X' \) and \( \bar{X} \) are used interchangeably!

Switch networks

(a) Two states of a switch

(b) Symbol for a switch

Switch networks

(a) Simple connection to a battery

(b) Using a ground connection as the return path

Switch networks

(a) The logical AND function (series connection)

(b) The logical OR function (parallel connection)
Switch networks

L(x) = x'

Power supply
R
S
Light

Logic Operations

• Don’t confuse the AND symbol “·” and OR symbol “+” with arithmetic multiplication and addition.

  There are some differences, for example:
  - Arithmetic addition: 1+1=2
  - OR operation: 1+1=1

• Based on the context you should recognize if it is AND/OR or addition/multiplication.

• One more thing: sometimes we drop the “·” symbol.
  
  a·b is the same as ab

Truth table

• The most basic representation of a logic function is a truth table.
  - A truth table lists the output of the circuit for every possible input combination.
  - There are 2^n rows in a truth table for an n-variable function.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>XY</th>
<th>X + Y</th>
<th>X'</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>1</td>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth table:

Logic Gate

• Binary signals are manipulated using logic gates. These are electronic devices whose inputs and outputs are interpreted with only two values, representing logic 0 and logic 1.

• The bubble on the inverter output denotes “inverting” behavior.

Logic Gate

• Combinations of gates form a logic circuit or logic network.

• Analysis: For an existing network determine the function performed by the network.

• Synthesis: Design a network that implements a desired function.

Analysis and Synthesis of a Logic Network
To design logic circuits and describe their operations we use a mathematical tool called Boolean algebra (from English mathematician George Boole in 1800’s) that operates on two-valued (or logic) functions. A logic function can be implemented in many ways with logic circuits, what is and how to find the best implementation?

Boolean Algebra

Axioms of Boolean algebra

The axioms (or postulates) of a mathematical system are a minimal set of basic definitions that we assume to be true. The first three pairs of axioms state the formal definitions of the AND (logical multiplication) and OR (logical addition) operations:

(1a) 0·0 = 0  (1b) 1+1 = 1
(2a) 1·1 = 1  (2b) 0+0 = 0
(3a) 0·1 = 1·0 = 0  (3b) 1+0 = 0+1 = 1

The next axioms embody the complement notation:

(4a) If X=0, then X’=1  (4b) If X=1, then X’=0
Theorems of Boolean algebra

- Theorems are statements, known to be true, that allow us to manipulate algebraic expressions to have simpler analysis or more efficient synthesis of the corresponding circuits.
- Theorems involving a single variable:
  (5a) $X \cdot 0 = 0$ (Null elements)
  (5b) $X + 1 = 1$
  (6a) $X \cdot X = X$ (Identities)
  (6b) $X + 0 = X$
  (7a) $X \cdot X = X$ (Idempotency)
  (7b) $X + X = X$
  (8a) $X \cdot X' = 0$ (Complements)
  (8b) $X + X' = 1$
  (9) $(X')' = X$ (Involution)

- These theorems can be proved to be true. Let us prove 6b:
  $[X=0]: 0+0=0$ (true, according to 2b)
  $[X=1]: 1+0=1$ (true, according to 3b)

- Theorems involving two or three variables:
  (10a) $X \cdot Y = Y \cdot X$ (Commutativity)
  (10b) $X + Y = Y + X$
  (11a) $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$ (Associativity)
  (11b) $(X + Y) + Z = X + (Y + Z)$
  (12a) $X \cdot Y \cdot Z \cdot X = X \cdot (Y \cdot Z) \cdot X$
  (12b) $(X + Y) \cdot (X + Z) = X + Y$
  (13a) $X + X' \cdot Y = X + Y$ (Absorption)
  (13b) $X \cdot (X' + Y) = X$
  (14a) $X \cdot Y \cdot X' = X$
  (14b) $(X + Y) \cdot (X + Y') = X$
  (15a) $(X + Y) + Z = X + (Y + Z)$ (DeMorgan's Theorems)
  (15b) $(X + Y) \cdot Z = X + (Y \cdot Z)$

- Duality

- Theorems were presented in pairs.
- The b version of a theorem is obtained from the a version by swapping "0" and "1", and "·" and "+".
- Principle of Duality: Any theorem or identity in Boolean algebra remains true if 0 and 1 are swapped and · and + are swapped throughout.
- Duality doubles the utilities of everything about Boolean algebra and enriches the manipulation of logic functions.

Consensus Theorem:

- $X \cdot Y + X \cdot Z + Y \cdot Z = X \cdot Y + X \cdot Z$
- redundant

Note: $Y$ and $Z$ are associated with $X$ and $\overline{X}$, and appear together in the term that is eliminated.

By duality:

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

Differences between Boolean and ordinary algebra

- Distributive law of + over ·
  $x + (y \cdot z) = (x + y) \cdot (x + z)$ is not valid in ordinary algebra

- Boolean algebra does not have additive or multiplicative inverse so there is no subtraction or division operations
Boolean Algebra

- Boolean algebra is used for manipulating logical functions when designing digital hardware.

- However, today most design is done using Computer-Aided Design (CAD) software that includes schematic capture, logic simplification and simulation.

- Other methods include truth tables, Venn diagrams and Karnaugh Maps.

Venn Diagram

- A graphical tool that can be used for Boolean algebra
- A binary variable s is represented by a contour
- Area within the contour corresponds to s=1
- Area outside the contour corresponds to s=0
- Two variables are represented by two overlapping circles

Venn Diagram

- Variable s
- Area within the contour corresponds to s=1
- Area outside the contour corresponds to s=0
- Two variables are represented by two overlapping circles

Precedence of operations

- In the absence of parentheses, operations in a logic expression must be performed in the order: NOT, AND, OR.

Example:

\[ f = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2} \]

Synthesis using AND, OR and NOT

- One way of designing a logic circuit that implements a truth table is to create a product term that has a value of 1 for each valuation for which the output function has to be 1.

- Then we take the logical sum of these product terms to realize f

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( f(x_1, x_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>
Minterm, Maxterm

- **Minterm**
  A product term in which all variables of a function appear exactly once, uncomplemented or complemented.

- **Maxterm**
  A sum term in which all variables of a function appear exactly once, uncomplemented or complemented.

Minterm, Maxterm

For a Boolean function of \( n \) variables, there are \( 2^n \) minterms:

\[
m_0 \ldots m_{2^n - 1}\]

and \( 2^n \) maxterms:

\[
M_0 \ldots M_{2^n - 1}\]

Note that: \( M_i = \overline{m_i} \)

Minterm, Maxterm

<table>
<thead>
<tr>
<th>Row No.</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Minterm</th>
<th>Maxterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \overline{A} \overline{B} \overline{C} = m_0 )</td>
<td>( \overline{A} + \overline{B} + \overline{C} = M_0 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( \overline{A} B \overline{C} = m_1 )</td>
<td>( \overline{A} + B + \overline{C} = M_1 )</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( \overline{A} B \overline{C} = m_2 )</td>
<td>( \overline{A} + B + \overline{C} = M_2 )</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \overline{A} B C = m_3 )</td>
<td>( \overline{A} + B + C = M_3 )</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( A \overline{B} \overline{C} = m_4 )</td>
<td>( A + \overline{B} + \overline{C} = M_4 )</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>( A \overline{B} C = m_5 )</td>
<td>( A + \overline{B} + C = M_5 )</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( A B \overline{C} = m_6 )</td>
<td>( A + B + \overline{C} = M_6 )</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( A B C = m_7 )</td>
<td>( A + B + C = M_7 )</td>
</tr>
</tbody>
</table>

\( M_i = m_i^\prime \)

Canonical Sum of Products Form

- A Boolean function \( f(x_1, x_2, x_3) \) can be expressed algebraically as a logical sum of minterms:

\[
\begin{array}{c|c|c|c|c}
\hline
\text{Row number} & x_1 & x_2 & x_3 & f(x_1, x_2, x_3) \\
\hline
0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 \\
2 & 0 & 1 & 0 & 0 \\
3 & 0 & 1 & 1 & 0 \\
4 & 1 & 0 & 0 & 1 \\
5 & 1 & 0 & 1 & 1 \\
6 & 1 & 1 & 0 & 1 \\
7 & 1 & 1 & 1 & 0 \\
\hline
\end{array}
\]
Canonical Sum of Products Form

- f can be expressed as sum of product terms (SOP)
  \[ f(x_1, x_2, x_3) = \sum (m1, m4, m5, m6) \]
  \[ f(x_1, x_2, x_3) = \sum m(1,4,5,6) \]

Canonical Product of Sums Form

- The complement of \( f(x_1,x_2,x_3) \) can be formed as the logical sum of all minterms not used in \( f(x_1,x_2,x_3) \):
  \[ \tilde{f}(x_1, x_2, x_3) = m0 + m2 + m3 + m7 \]
  \[ f = m0 \cdot m2 \cdot m3 \cdot m7 \]
  \[ f = M_0 \cdot M_2 \cdot M_3 \cdot M_7 \]
  This is called the product of sum presentation of \( f \)

Conversion Between the Canonical Forms

- It is easy to convert from one canonical form to other one, simply use the DeMorgan’s theorem.

Example:
  \[ F(A,B,C) = \sum (1,4,5,6,7) \]
  \[ F'(A,B,C) = \sum (0,2,3) \]
  \[ F(A,B,C) = (m0 + m2 + m5) = m_0 \cdot m_2 \cdot m_5 = M_0 \cdot M_2 \cdot M_5 \]
  \[ F(A,B,C) = \prod (0,2,3) \]

Cost of a Logic Circuit

- Cost of a logic circuit: total number of gates plus total number of inputs to all gates in the circuit
- The canonical SOP and POS implementations described before are not necessarily minimum cost
- We can simplify them to obtain minimum-cost SOP and POS circuits

Reducing Cost

How can we simplify a logic function?
- There are systematic approaches for doing this (e.g., Karnaugh map) that we will learn later
- The other way is to use theorems and properties of Boolean algebra and do algebraic manipulations
- Do an example on the board.

Reducing Cost

- The simplified version of SOP is called minimal SOP
- The simplified version of POS is called minimal POS
- We cannot in general predict whether the minimal SOP expression or minimal POS expression will result in the lowest cost.
- It is often useful to check both expressions to see which gives the best result.
Other Logic Operations

- NAND
- NOR
- XOR
- XNOR

NAND

- NAND: a combination of an AND gate followed by an inverter.

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

- Symbol for NAND is ↑
- NAND gates have several interesting properties:
  \[ A \uparrow A = A' \]
  \[ (A \uparrow B)' = AB \]
  \[ (A' \uparrow B') = A + B \]

NAND

- These three properties show that a NAND gate with both of its inputs driven by the same signal is equivalent to a NOT gate.
- A NAND gate whose output is complemented is equivalent to an AND gate, and a NAND gate with complemented inputs acts as an OR gate.
- Therefore, we can use a NAND gate to implement all three of the elementary operators (AND, OR, NOT).
- Therefore, ANY Boolean function can be constructed using only NAND gates.

NAND

- NOR: a combination of an OR gate followed by an inverter.

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

- NOR gates also have several interesting properties:
  \[ A \downarrow A = A' \]
  \[ (A \downarrow B)' = A + B \]
  \[ A' \downarrow B' = AB \]

NAND

- Just like the NAND gate, any logic function can be implemented using just NOR gates.
- Both NAND and NOR gates are very valuable as any design can be realized using either one.
- It is easier to build an IC chip using all NAND or NOR gates than to combine AND, OR, and NOT gates.
- NAND/NOR gates are typically faster at switching and cheaper to produce.
NAND and NOR networks

- NAND and NOR can be implemented by simpler electronic circuits than the AND and OR functions.
- Can these gates be used in synthesis of logic circuits?

Exclusive OR (XOR)

- The eXclusive OR (XOR) function is an important Boolean function used extensively in logic circuits.
- The XOR function maybe:
  - implemented directly as an electronic circuit (truly a gate)
  - or
  - implemented by interconnecting other gate types (used as a convenient representation)
- The XOR function means: X OR Y, but NOT BOTH

XOR

- XOR gates assert their output when exactly one of the inputs is asserted, hence the name.
- The symbol for this operation is ⊕
  \[ Y = A' B + AB' \]
XNOR

- The eXclusive NOR function is the complement of the XOR function
- The symbol for this operation is $\oplus$, i.e. $1 \oplus 1 = 0$ and $1 \oplus 0 = 0$.

\[
Y = A' B' + AB
\]

- Why is the XNOR function also known as the equivalence function?

XOR Implementations

- A SOP implementation

\[
\begin{array}{c|c|c}
A & B & Y \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

- A NAND implementation

XOR and XNOR

- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers

XOR

- XOR identities:
  \[
  X \oplus 0 = X \\
  X \oplus X = 0 \\
  X \oplus Y = Y \oplus X \\
  X \oplus 1 = X' \\
  X \oplus X' = 1
  \]

Gates with more than two inputs

- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.
- AND and OR operations have these two properties
- NAND and NOR are not associative:
  \[
  (A \uparrow B) \uparrow C \neq A \uparrow (B \uparrow C) \\
  (A \uparrow B) \uparrow C \neq A \uparrow (B \uparrow C)
  \]

Gates with more than two inputs

- We define multiple input NAND and NOR gates as follows:
  \[
  A \downarrow B \downarrow C = (A + B + C)' \\
  A \uparrow B \uparrow C = (ABC)'
  \]
Gates with more than two inputs

- XOR and XNOR are both commutative and associative
- Definition of XOR should be modified for more than two inputs
- For more than 2 inputs, XOR is called an odd function: it is equal to 1 if the input variables have an odd number of 1’s
- Similarly, for more than 2 inputs, XNOR is called an even function: it is equal to 1 if the input variables have an even number of 1’s

Learning Objectives

- List the three basic logic operations
- Draw the truth table for the basic logic operations
- Build truth table for an arbitrary number of variables
- Draw schematic for basic logic gates
- Perform analysis on simple logic circuits
- Draw timing diagram for simple logic circuits