FPGA Design
EECE 277

Gates and Boolean Algebra

Dr. William H. Robinson
January 17, 2005
Dr. Martin Luther King, Jr. Birthday (observed)

http://eecs.vanderbilt.edu/courses/eece277/

Topics
“Injustice anywhere is a threat to justice everywhere.”
– Dr. Martin Luther King, Jr.

• Administrative stuff
  – Homework #1 Assigned
  – Read Chapter 12 in Brown/Vranesic
  – UP2 Kits
• Boolean Algebra
• Basic logic gates
• DeMorgan’s Theorem

Boolean Algebra

• Variables and functions can only take on values from the set \{0, 1\}

• Sometimes called “switching algebra”

• Logic ‘1’ corresponds to
  – TRUE, HIGH, Vdd

• Logic ‘0’ corresponds to
  – FALSE, LOW, GND

Binary Switch

\[ x = 0 \quad \text{and} \quad x = 1 \]

(a) Two states of a switch

(b) Symbol for a switch
### Light Controlled by a Switch

(a) Simple connection to a battery

(b) Using a ground connection as the return path

### Basic Functions

(a) The logical **AND** function (series connection)

(b) The logical **OR** function (parallel connection)

### Functions in Boolean Algebra

A truth table gives the function output for each input combination

Given a function with \( n \) inputs:
- There are \( 2^n \) possible input combinations
- There are \( 2^{2^n} \) possible functions

Example: 1 input

<table>
<thead>
<tr>
<th>A</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Common Functions and Gates

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>NOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<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>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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Common Functions and Gates

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>NOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Inverter bubbles can be used to complement outputs.

Logic Networks

- **Analysis**
  - Task of determining the function performed by a network

- **Synthesis**
  - Task of designing a network that implements a desired functional behavior

Logic Network Analysis (1)

Logic Network Analysis (2)
Logic Network Analysis (3)

\[
x_1, x_2, f(x_1, x_2)
\]

<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>

Logic Network Analysis (4)

\[
x_1, x_2, f(x_1, x_2)
\]

<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>

Functional Timing Diagram

<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>

Logic Synthesis

How do you realize \(f(x_1, x_2)\)?
Using Product Terms

\[
\begin{array}{c|c|c|c}
 x_1 & x_2 & f(x_1, x_2) \\
--- & --- & --- \\
 0 & 0 & 1 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 1 \\
\end{array}
\]

\[f(x_1, x_2) = \overline{x}_1 \overline{x}_2 + x_1 x_2\]

Using Sum Term

\[
\begin{array}{c|c|c|c}
 x_1 & x_2 & f(x_1, x_2) \\
--- & --- & --- \\
 0 & 0 & 1 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 1 \\
\end{array}
\]

\[f(x_1, x_2) = \overline{x}_1 + x_2\]

Cost of Implementation

- Both are equivalent implementations
- Gate count can be minimized

Minterms and Maxterms

<table>
<thead>
<tr>
<th>Row number</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>Minterm</th>
<th>Maxterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(m_0) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_0) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(m_1) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_1) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>(m_2) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_2) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>(m_3) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_3) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>(m_4) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_4) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>(m_5) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_5) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>(m_6) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_6) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>(m_7) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
<td>(M_7) = \overline{x}_1 \overline{x}_2 \overline{x}_3</td>
</tr>
</tbody>
</table>

Figure 2.17 Three-variable minterms and maxterms.
Synthesizing Functions

- **Sum of Products (SOP)**
  - A function $f$ can be represented by an expression that is the logical sum of the minterms where $f = 1$
  - Canonical SOP can be reduced using properties of Boolean Algebra
  - AND-OR plane

- **Product of Sums (POS)**
  - A function $f$ can be represented by an expression that is the logical product of the maxterms where $f = 0$
  - Canonical POS can be reduced using properties of Boolean Algebra
  - OR-AND plane

Implementation

- Usually implement functions on a chip using similar gates
  - Both NAND and NOR are complete gates which can be used to represent all functions

- By adding pairs of inverter bubbles:
  - Sum of Products (AND-OR) implementation converts to NAND
  - Product of Sums (OR-AND) implementation converts to NOR

NAND and NOR Gates

- (a) NAND gates
- (b) NOR gates

Complete Gates

- (a) NOT gates
- (b) AND gates
- (c) OR gates using only NAND gates or only NOR gates.
DeMorgan’s Theorem

\[
(A \cdot B) = \overline{A} + \overline{B} \\
(A + B) = \overline{A} \cdot \overline{B}
\]

- Complement all variables
- Change OR’s to AND’s
- Change AND’s to OR’s

Using Only NAND Gates

- Add inverter bubbles

Figure 2.21. DeMorgan’s theorem in terms of logic gates.
Using Only NOR Gates

- Add inverter bubbles

Three-Way Light Control

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>C</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>C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Product-of-sums realization

Sum-of-products realization

Summary

- Boolean Algebra uses variables and functions that map to the set of \{0, 1\}
- Basic logic gates can be combined to form larger circuits
- DeMorgan’s theorem provides circuit equivalency for complementing logical functions and variables