Basics of Logic Design: Boolean Algebra, Logic Gates

CPS 104

Today’s Lecture

Outline
• Building the building blocks…
• Logic Design
  ➢ Truth tables, Boolean functions, Gates and Circuits

Reading
  Appendix B
The Big Picture

- The Five Classic Components of a Computer

What We’ve Done, Where We’re Going
Digital Design

- Logic Design, Switching Circuits, Digital Logic

Recall: Everything is built from transistors
- A transistor is a switch
- It is either on or off
- On or off can represent True or False

Given a bunch of bits (0 or 1)...
- Is this instruction a lw or a beq?
- What register do I read?
- How do I add two numbers?
- Need a method to reason about complex expressions

Boolean Algebra

- Boolean functions have arguments that take two values (\{T,F\} or \{1,0\}) and they return a single or a set of (\{T,F\} or \{1,0\}) value(s).
- Boolean functions can always be represented by a table called a “Truth Table”
- Example: \( F: \{0,1\}^3 \rightarrow \{0,1\}^2 \)

\[
\begin{array}{ccc|cc}
 a & b & c & f_1 & f_2 \\
 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 1 & 1 \\
 0 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 \\
 1 & 0 & 0 & 1 & 0 \\
 1 & 1 & 0 & 0 & 1 \\
 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
Boolean Functions

- **Example** Boolean Functions: NOT, AND, OR, XOR, ... 

<table>
<thead>
<tr>
<th>a</th>
<th>NOT (a)</th>
<th>b</th>
<th>AND (a, b)</th>
<th>OR (a, b)</th>
<th>XOR (a, b)</th>
<th>XNOR (a, b)</th>
<th>NOR (a, b)</th>
</tr>
</thead>
<tbody>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<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>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Boolean Functions and Expressions

- **Boolean algebra notation**: Use * for AND, + for OR, ~ for NOT.
  - NOT is also written as A’ and \( \overline{A} \)
- Using the above notation we can write Boolean expressions for functions

\[
F(A, B, C) = (A * B) + (\overline{A} * C)
\]

- We can evaluate the Boolean expression with all possible argument values to construct a truth table.

- **What is truth table for F?**
Boolean Functions and Expressions

\[ F(A, B, C) = (A \cdot B) + (\neg A \cdot C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Function Simplification

- Boolean expressions can be simplified by using the following rules:
  - \( A \cdot A = A \)
  - \( A \cdot 0 = 0 \)
  - \( A \cdot 1 = A \)
  - \( A \cdot \neg A = 0 \)
  - \( A + A = A \)
  - \( A + 0 = A \)
  - \( A + 1 = 1 \)
  - \( A + \neg A = 1 \)
  - \( A \cdot B = B \cdot A \)
  - \( A \cdot (B + C) = (B + C) \cdot A = A \cdot B + A \cdot C \)
### Boolean Function Simplification

#### Table: Inputs and Outputs

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</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>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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>

**f₁ = \neg a \neg b \neg c + \neg a \neg b \neg c + a \neg b \neg c + a \neg b \neg c**

**f₂ = \neg a \neg b \neg c + \neg a \neg b \neg c + a \neg b \neg c + a \neg b \neg c**

Simplify these functions...

### Boolean Function Simplification

**f₁ = \neg a \neg b \neg c + \neg a \neg b \neg c + a \neg b \neg c + a \neg b \neg c**

= \neg a (\neg b \neg c + b \neg c) + a (\neg b \neg c + b \neg c)

= \neg a \neg c (\neg b + b) + a \neg c (\neg b + b)

= \neg a \neg c + a \neg c

= \neg c (\neg a + a)

= c

**f₂ = \neg a \neg b \neg c + \neg a \neg b \neg c + a \neg b \neg c + a \neg b \neg c**

= \neg a (\neg b \neg c + \neg b \neg c) + a (\neg b \neg c + b \neg c)

= \neg a \neg b (c + \neg c) + a \neg b (\neg c + c)

= \neg a \neg b + a \neg b
Boolean Functions and Expressions

- The Fundamental Theorem of Boolean Algebra:
Every Boolean function can be written in disjunctive normal form as an OR of ANDs (Sum-of products) of it's arguments or their complements.

“Proof:” Write the truth table, construct sum-of-product from the table.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>XNOR (a, b)</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>1</td>
</tr>
</tbody>
</table>

XNOR = (¬a * ¬b) + (a * b)

Boolean Functions and Expressions

- Example-2:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f₁, f₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

f₁ = ¬a*¬b*c + ¬a*b*¬c + a*¬b*¬c + a*b*c
f₂ = ¬a*¬b*¬c + ¬a*¬b*c + a*b*¬c + a*b*c
DeMorgan’s Laws

- \( \sim(A+B) = \sim A \sim B \)
- \( \sim(A\ast B) = \sim A + \sim B \)

Example:

- \( \sim C \sim A \sim B + \sim C \ast A \sim B + \sim A \ast B + C \ast \sim A \sim B \)
- Use only XOR to represent this function

Applying the Theory

- Lots of good theory
- Can reason about complex boolean expressions
- Now we have to make it real...
Boolean Gates

- Gates are electronic devices that implement simple Boolean functions

Examples

- AND(a, b)
- OR(a, b)
- NOT(a)
- XOR(a, b)
- NAND(a, b)
- NOR(a, b)
- XNOR(a, b)

Reality Check

- Basic 1 or 2 Input Boolean Gate 1-4 Transistors

Pentium III
- Processor Core 9.5 Million Transistors
- Total: 28 Million Transistors

Pentium 4
- Total: 42 Million Transistors
Digital Design Examples

Input: 2 bits representing an unsigned number (n)
Output: \(n^2\) as 4-bit unsigned binary number

Input: 2 bits representing an unsigned number (n)
Output: 3-n as unsigned binary number
Design Example

- Consider machine with 4 registers
- Given 2-bit input (register specifier, \(I_1, I_0\))
- Want one of 4 output bits (\(O_3-O_0\)) to be 1
  - E.g., allows a single register to be accessed
- What is the circuit for this?

Circuit Example: Decoder

```
<table>
<thead>
<tr>
<th>I_1</th>
<th>I_0</th>
<th>Q_0</th>
<th>Q_1</th>
<th>Q_2</th>
<th>Q_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

© Alvin R. Lebeck
Circuit Example: 2x1 MUX

Multiplexor (MUX) selects from one of many inputs

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

Example 4x1 MUX
Arithmetic and Logical Operations in ISA

• What operations are there?
• How do we implement them?
  ➢ Consider a 1-bit Adder

Summary

• Boolean Algebra & functions
• Logic gates (AND, OR, NOT, etc)
• Multiplexors

Reading
• Appendix B