Lec 2: Gates and Logic

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

• Class newsgroup created
  – Posted on web-page

• Use it for partner finding

• First assignment is to find partners
  – Due this Friday

• Sections start this week

© Kavita Bala, Computer Science, Cornell University
A switch

• A switch is a simple device that can act as a conductor or isolator

• Can be used for amazing things…

Transistors

• Solid-state switch
  – The most amazing invention of the 1900s

• PNP and NPN
P and N Transistors

- **PNP Transistor**
  - Connect E to C when base = 0

- **NPN Transistor**
  - Connect E to C when base = 1

Inverter

- Function: NOT
- Called an inverter
- Symbol: \[ 
\begin{array}{c}
\text{in} \\
\downarrow \\
\text{out}
\end{array} \]
- Useful for taking the inverse of an input
  - CMOS: complementary-symmetry metal–oxide–semiconductor

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth table
NAND Gate

- Function: NAND
- Symbol:

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

NOR Gate

- Function: NOR
- Symbol:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Building Functions

- **NOT:**

- **AND:**

- **OR:**

- **NAND and NOR are universal**
  - Can implement any function with NAND or just NOR gates
  - useful for manufacturing
Logic Equations

- **AND**
  - $\text{out} = a \land b = a \& b = a \land b$

- **OR**
  - $\text{out} = a + b = a \lor b = a \lor b$

- **NOT**
  - $\text{out} = \overline{a} = \neg a = \neg a$

Identities

- Identities useful for manipulating logic equations
  - For optimization & ease of implementation
  - $a + \overline{a} = 1$
  - $a + 0 = a$
  - $a + 1 = 1$
  - $a \overline{a} = 0$
  - $a 0 = 0$
  - $a 1 = a$
  - $a(b+c) = \overline{a} + bc$
  - $(a + b) = \overline{a} \overline{b}$
  - $(\overline{a} b) = \overline{a} + \overline{b}$
  - $a + a b = a$
Logic Manipulation

- Can specify functions by describing gates, truth tables or logic equations
- Can manipulate logic equations algebraically
- Can also use a truth table to prove equivalence
- Example: \((a+b)(a+c) = a + bc\)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a+b</th>
<th>a+c</th>
<th>LHS</th>
<th>bc</th>
<th>RHS</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</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>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Logic Minimization

- A common problem is how to implement a desired function most efficiently
- One can derive the equation from the truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(abc)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(abc)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(\overline{abc})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(\overline{abc})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(abc)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(abc)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(\overline{abc})</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(\overline{abc})</td>
</tr>
</tbody>
</table>

for all outputs that are 1, take the corresponding minterm
Obtain the result in “sum of products” form

- How does one find the most efficient equation?
  - Manipulate algebraically until satisfied
  - Use Karnaugh maps (or K maps)
Multiplexer

- A multiplexer selects between multiple inputs
  - out = a, if d = 0
  - out = b, if d = 1

- Build truth table
- Minimize diagram
- Derive logic diagram

Multiplexer Implementation

- Build a truth table
  \[ out = \overline{a}bd + \overline{b}d + a\overline{b}\overline{d} \]
  \[ = \overline{a}d + bd \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>d</th>
<th>out</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>0</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>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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Multiplexer Implementation

- Draw the circuit

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

\[ \text{out} = ad + bd \]

Logic Gates

- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost \~$1 per chip, mostly for packaging and testing

- Cumbersome, but possible to build devices using gates put together manually
Integrated Circuits

• Or one can manufacture a complete design using a custom mask
• Intel Pentium has approximately 125 million transistors

Voting machine

• Build something interesting

• A voting machine
  – Elections are coming up!

• Assume:
  – A vote is recorded on a piece of paper,
  – by punching out a hole,
  – there are at most 7 choices
  – we will not worry about “hanging chads” or “invalids”
Voting machine

• For now, let’s just display the numerical identifier to the ballot supervisor
  – we won’t do counting yet, just decoding
  – we can use four photo-sensitive transistors to find out which hole is punched out

  • A photo-sensitive transistor detects the presence of light
  • Photo-sensitive material triggers the gate

Ballot Reading

– Input: paper with a hole in it
– Out: number the ballot supervisor can record

© Kavita Bala, Computer Science, Cornell University

Ballots

The 3410 vote recording machine

© Kavita Bala, Computer Science, Cornell University
Encoders

- N sensors in a row
- Want to distinguish which of the N sensors has fired
- Want to represent the firing sensor number in compact form
  - N might be large
  - Only one wire is on at any time
  - Silly to route N wires everywhere, better to encode in log N wires

A 3-bit encoder
(7-to-3)
(5 inputs shown)

Number Representations

- Decimal numbers are written in base 10
  - $3 \times 10^1 + 7 \times 10^0 = 37$
- Just as easily use other bases
  - Base 2 - “Binary”
  - Base 8 - “Octal”
  - Base 16 – “Hexadecimal”
Number Representations

- Base conversion via repetitive division
  - Divide by base, write remainder, move left with quotient
  - Sanity check with 37 and 10

\[
\begin{array}{c|c|c}
37 & \text{binary representation} & \text{base 2 representation} \\
\hline
101 & 1 & 101 \\
\end{array}
\]

Binary Representation

- \(37 = 32 + 4 + 1\)

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
\hline
2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]
Hexadecimal Representation

- 37 decimal = \( (25)_{16} \)
- Convention
  - Base 16 is written with a leading 0x
  - 37 = 0x25
- Need extra digits!
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Binary to hexadecimal is easy
  - Divide into groups of 4, translate groupwise into hex digits

Encoder Truth Table

- \( o_2 = \overline{abcd} \)
- \( o_1 = \overline{abcd} + \overline{abcd} \)
- \( o_0 = \overline{abcd} + \overline{abcd} \)
Ballot Reading

- Ok, we built first half of the machine
- Need to display the result

Ballots

The 3410 voting machine

7-Segment LED Decoder

- 4 inputs encoded in binary
- 8 outputs, each driving an independent, rectangular LED
- Can display numbers
- Just a simple logic circuit
- Write the truth table
7-Segment LED Decoder

- 4 inputs encoded in binary
- 8 outputs, each driving an independent, rectangular LED
- Can display numbers

© Kavita Bala, Computer Science, Cornell University
### 7-Segment Decoder Truth Table

<table>
<thead>
<tr>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>$O_0$</th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
<th>$O_4$</th>
<th>$O_5$</th>
<th>$O_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
<td>1</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>0</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Exercise:** Find the error(s) in this truth table.
Ballot Reading

• Done!

Ballots  The 3410 voting machine

Summary

• We can now implement any logic circuit
  – Can do it efficiently, using Karnaugh maps to find the minimal terms required
  – Can use either NAND or NOR gates to implement the logic circuit
  – Can use P- and N-transistors to implement NAND or NOR gates