Lecture 4: Logic Gates

Last Lecture
- CMOS
- Basic Boolean Functions
- Boolean Algebra

Today
- Logic Gates
- Different Implementations
- Bubbles

Logic Gates and Truth Tables

- **AND** $X \cdot Y$ $XY$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  \end{array}
  \]

- **OR** $X + Y$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 1 \\
  \end{array}
  \]

- **NOT** $\overline{X}$ $X'$
  \[
  \begin{array}{cc}
  x & y \\
  0 & 1 \\
  1 & 0 \\
  \end{array}
  \]

- **Buffer** $X$
  \[
  \begin{array}{cc}
  x & y \\
  0 & 0 \\
  1 & 1 \\
  \end{array}
  \]

Logic Gates and Truth Tables

- **NAND** $X \cdot \overline{Y}$ $XY$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 1 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  \end{array}
  \]

- **NOR** $\overline{X} + \overline{Y}$ $X + Y$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 1 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  \end{array}
  \]

- **XOR** $X \oplus Y$ $X \oplus Y$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
  \end{array}
  \]

- **XNOR** $\overline{X} \oplus \overline{Y}$ $X \oplus Y$
  \[
  \begin{array}{ccc}
  x & y & z \\
  0 & 0 & 1 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  \end{array}
  \]

Useful Lingo

- **Schematic**: A drawing of interconnected logic gates
- **Net**: wires that are all at the same voltage
- **Netlist**: A list of all the devices and connections in a schematic
- **Fan-in**: The number of inputs to a gate
- **Fan-out**: The number of outputs of a gate
Boolean Functions to Gates

Example: $F = (A \cdot B)' + C \cdot D$

Example: $F = C \cdot (A + B)'$

More than one way to map expressions to gates

$e.g., \quad Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot (C + D)))$

use of 3-input gate

What is the Optimal Implementation?

- We use the axioms and theorems of Boolean algebra to “optimize” our designs
- Design goals vary
  - Reduce the number of inputs?
  - Reduce the number of gates?
  - Reduce number of gate levels?
- How do we explore the tradeoffs?
  - CAD tools
  - Logic minimization: Reduce number of gates and complexity
  - Logic optimization: Maximize speed and/or minimize power

Example: A Binary Full Adder

1-bit binary adder
- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$S = A'B'Cin + A'BCin' + AB'Cin' + ABCin$

$Cout = A'BCin + AB'Cin + ABCin' + ABCin$
**Full Adder: Sum**

**Before Boolean minimization**
Sum = $A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$

**After Boolean minimization**
Sum = $(A \oplus B) \oplus C_{in}$

---

**Full Adder: Carry**

**Before Boolean minimization**
$C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$

**After Boolean minimization**
$C_{out} = BC_{in} + AC_{in} + AB$

---

**Preview: A 2-bit Ripple-Carry Adder**

---

**In Class Challenge**
Implement XOR using NANDs and NORs (challenge: fewest)
Different Realizations

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

- **two-level realization** (we don’t count NOT gates)
  - XOR gate (easier to draw but costlier to build)
- **multi-level realization** (gates with fewer inputs)

Waveform View of Logic Functions

- Just a sideways truth table
  - but note how edges don’t line up exactly
  - it takes time for a gate to switch its output!

Are All Implementations Equivalent?

- Under the same input stimuli, the three alternative implementations have almost the same waveform behavior
  - delays are different
  - glitches (hazards) may arise – these could be bad, it depends
  - variations due to differences in number of gate levels and structure
- The three implementations are functionally equivalent

CMOS is Inverting

- CMOS logic gates are inverting
  - Get NAND, NOR, NOT
  - Don’t get AND, OR, Buffer
**DeMorgan’s Theorem**

- Replace
  - • with +, + with •, 0 with 1, and 1 with 0
- All variables with their complements
- Example 1: $Z = A'B' + A'C'$
  
  $Z' = (A'B' + A'C')' = (A'B)' \cdot (A'C)' = (A+B) \cdot (A+C)$

- Example 2: $Z = A'B'C + A'BC + AB'C + ABC'$
  
  $Z' = (A'B'C + A'BC + AB'C + ABC')' = (A'B'C)' \cdot (A'BC)' \cdot (AB'C)' \cdot (ABC')' = (A+B+C') \cdot (A+B'+C') \cdot (A'+B+C') \cdot (A'+B'+C)$

**DeMorgan’s, NAND, NOR**

- DeMorgan’s Theorem
  - Standard Form: $(A+B)' = A'B'$
  - Inverted Form: $(A+B) = (A'B')'$
- AND with complemented inputs $\equiv$ NOR
- OR with complemented inputs $\equiv$ NAND
- OR $\equiv$ NAND with complemented inputs and outputs
- AND $\equiv$ NOR with complemented inputs and outputs

**Double Bubble**

- Introduce inversions ("bubbles")
- Introduce bubbles in pairs
- Conserve inversions
- Do not alter logic function
- Example
  - AND/OR to NAND/NAND

**Bubble Trouble Continued**

- Example: AND/OR network to NOR/NOR
  
  $Z = AB + CD$
  
  $Z = (A'B')' + (C+D)' = (A'B)(C+D)' = (AB)'(CD)'$

- Conserve "bubbles"
Example: OR/AND to NAND/NAND

Example: OR/AND to NOR/NOR