CSE140: Components and Design Techniques for Digital Systems

Introduction

Prof. Tajana Simunic Rosing
Let's get started!

- What we’ll do today
  - NAND/NOR, AND/OR in CMOS
  - Logic gates, Universal gates
  - Boolean function representations

- What comes next:
  - Boolean algebra

- Upcoming deadlines:
  - HW#1 assigned next Tuesday
  - ZyBook chap 1 due today prior to class time
    - TED has a test version of scores as of Mon pm; script updated as of 11am
    - We have your actual scores in zybook, will upload later today. Let us know if there are still issues.
  - Quiz#1 Thursday – with iClickers
    - Trial participation score will be input into TED today for those who use iClickers in class to make sure that your iClicker is connected to the correct section prior to Quiz
    - Let me know if you do not see your points posted, but attended the correct cse140 section and used iClicker
Transistor Circuit Design

• **nMOS:**
  - turns on when gate is connected to logic 1
  - passes 0’s well, so connect source to GND

• **pMOS:**
  - turns on when gate is connected to logic 0
  - passes 1’s well, so connect source to $V_{DD}$

• **Note:** Vahid’s textbook shows some circuits with pMOS connected to GND and nMOS to Vdd: this is NOT done in practice!
CMOS Gates: Two Input NAND Gate

NAND

\[ Y = \overline{AB} \]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</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>

\[
\text{Res}
\begin{align*}
\text{both on} & \Rightarrow \text{Res} = \frac{R_P}{2} \sim R_N \\
\text{both on} & \Rightarrow \text{Res} = 2R_N
\end{align*}
\]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
Two Input NOR Gate

CMOS gate structure:

Two-input NOR

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
Building a Two Input OR Gate

- **pMOS pull-up network**
- **nMOS pull-down network**

Inputs: A, B

Output: y = A + B

Truth Table:

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

Gates: PA, PB, NA, NB, P_i, N_i

V_DD connected to output:

- A = 0
- B = 0

Output: y = A + B

Diagram shows the circuit implementation of the OR gate.
Which is a Two Input AND gate?

A) pMOS pull-up network

B) nMOS pull-down network

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y = A*B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</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>

C) All of the above

D) None of the above
### Common Logic Gates

**AND**

\[ a \cdot b = a \land b \]

\[
\begin{array}{c|c|c}
 a & b & \text{AND} \\
 0 & 0 & 0 \\
 0 & 1 & 0 \\
 1 & 0 & 0 \\
 1 & 1 & 1 \\
\end{array}
\]

**OR**

\[ a + b = a \lor b \]

\[
\begin{array}{c|c|c}
 a & b & \text{OR} \\
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 1 \\
\end{array}
\]

**NOT**

\[ a' = \neg a \]

\[
\begin{array}{c|c}
 a & \text{NOT} \\
 0 & 1 \\
 1 & 0 \\
\end{array}
\]

**XOR**

\[ ab' + a'b = a \oplus b \]

\[
\begin{array}{c|c|c}
 a & b & \text{XOR} \\
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 0 \\
\end{array}
\]

**NAND**

\[ (a \cdot b)' = \overline{a \cdot b} \]

**NOR**

\[ (a+b)' = \overline{a+b} \]

**BUF**

\[ a = a \]

**CMOS**

\[
\begin{array}{c|c|c}
 a & b & \text{AND} \\
 0 & 0 & 0 \\
 0 & 1 & 0 \\
 1 & 0 & 0 \\
 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 a & b & \text{OR} \\
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 a & b & \text{XOR} \\
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 0 \\
\end{array}
\]
CSE140: Components and Design Techniques for Digital Systems

Universal Gates

Tajana Simunic Rosing
Completeness of Gates: NAND

- Any logic function can be implemented using just NAND gates. Why?
  - Boolean algebra: need AND, OR and NOT

\[
\begin{array}{c}
A \\
B \\
\end{array}
\rightarrow
\begin{array}{c}
x \\
y \\
\end{array}

y = A + B

y = \overline{x \cdot y} = \overline{x + y}

\begin{array}{c|c|c|c}
\text{x} & \text{y} & \text{x \cdot y} & \text{x + y} \\
\hline
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 \\
\end{array}
Implement using only NAND

- \( F = X'Y + Z \)
  - \( W = AY \)
  - \( A = X' \)

\[ \begin{align*}
X & \quad \rightarrow \quad A \\
Y & \quad \rightarrow \quad OR \\
W & \quad \rightarrow \quad OR \\
Z & \quad \rightarrow \quad OR \\
\end{align*} \]

\( F \) trans. 14

\( F \) transist. 2 4 2 4 12
Completeness of Gates: NOR

• Any logic function can be implemented using just NOR gates. Boolean algebra needs AND, OR and NOT.
Implement using only NOR

- \( F = X'Y + Z \)

\[
\begin{align*}
W &= AY \\
A &= X'
\end{align*}
\]

\[
\begin{array}{c}
\text{trans} \quad 2 \quad 4 \quad 4 \quad 2 \quad \text{total} 12 \quad \therefore < 14
\end{array}
\]
CSE140: Components and Design Techniques for Digital Systems

Representation of Boolean Functions

Tajana Simunic Rosing
Representations of Boolean Functions:

Example of Logical AND

Boolean Equation

\[ Y = A \cdot B \]

Truth table

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

Logic Circuit
Converting Text to Boolean Functions

- Convert the following English statement to a function
  - Q1. answer is 1 only if a is 1 and b is 1.
    - Answer: $F = a \cdot b$
  - Q2. answer is 1 only if either of a or b is 1.
    - Answer: $F = a + b$
  - Q3. answer is 1 only if a is 1 and b is 0.
    - A. $F = a \cdot b'$
    - B. $F = a + b'$
    - C. $F = a$
    - D. Both A. and B.
    - E. None of the above
Representations of Boolean Functions:
A More Complex Example

English: F outputs 1 when a is 0 and b is 0, or when a is 0 and b is 1.

Revised description: F outputs 1 when a is 0, regardless of b’s value.

Equation: \( F(a,b) = a'b' + a'b \)

Revised Equation: \( F(a,b) = a' \)

Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>F</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>0</td>
</tr>
</tbody>
</table>
Boolean equation & circuit example

• Given the Boolean expression, we can draw the circuit it represents by cascading gates (and vice versa)

Logic circuit & Boolean Algebra Expression
Convert equation to logic gates

- There is more than one way to map expressions to gates
  e.g., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot (C + D)))$