ECE 2300
Digital Logic & Computer Organization
Spring 2022

Combinational Building Blocks
Announcements

• Lab 1 due tomorrow

• Lab 2 will be posted on CMS tonight
Review of CMOS Logic

True or False?

1. NMOS is an active low switch

2. PMOS (pull-up) network passes a poor zero

3. NMOS and PMOS networks can be ON at the same time

4. A CMOS gate consists of an even number of transistors

5. A 2-input NAND requires 2 transistors
Recap: 2-Input NAND Gate in CMOS

![NAND Gate Circuit Diagram]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>L</td>
</tr>
</tbody>
</table>

Lecture 5: 4
Another “Mystery” Gate
2-Input AND Gate in CMOS

- CMOS gates produce inherent inversion
- Need to invert to the output of NAND to implement an AND gate
A More Complicated Circuit

Pull-up: \((A' + B')(C' + D')\)

Pull-down: \((A \cdot B + C \cdot D)'\)
Constructing CMOS Gate from Boolean Expression

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

**Step 1.** Figure out pull-down network that does what you want (e.g., what combination of inputs generates a low output)

**Step 2.** Walk the hierarchy replacing NMOS with PMOS, series subnets with parallel subnets, and parallel subnets with series subnets

**Step 3.** Combine PMOS pull-up network (from Step 2) with NMOS pull-down network (from Step 1) to form fully-complementary CMOS gate
CMOS Sanity Checks

- Equal number of NMOS and PMOS
- NMOS sources tied to ground or to drain of another NMOS
- PMOS sources tied to Vdd or drain of another PMOS
- Inputs tied to pairs of PMOS and NMOS transistors
Sum-of-Products Revisited

AND-OR

NAND-NAND
Product-of-Sums Revisited

OR-AND

NOR-NOR
Multi-Level Logic

- So far we have primarily focused on two-level representations for combinational logic.
- Multi-level logic is typically more compact (i.e., cost-efficient) in practice.

\[ Y = (A + B)(C' + D) + E' \]
Combinational Building Blocks

• More complex functions built from basic gates
  – Comparators
  – Multiplexers
  – Decoders
  – Encoders
  …

• Typically tens to hundreds of transistors
  – Used to be called Medium Scale Integration (MSI)

• Common building blocks for digital systems
XOR Gate

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X⊕Y</th>
<th>(X⊕Y)’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>
</tbody>
</table>

- **XOR:** $F = X \cdot Y' + X' \cdot Y$
  - Similar to OR gate, except when inputs are 1
  - Used for comparisons, error checking, etc.

- **XNOR:** $F = X' \cdot Y' + X \cdot Y$
  - Complemented version of XOR
Equality Comparators Using XOR

1-bit comparator

\[
\begin{array}{cc}
X & \text{different} \\
Y & \\
\end{array}
\]

4-bit comparator

\[
\begin{array}{c}
A0 \\
B0 \\
A1 \\
B1 \\
A2 \\
B2 \\
A3 \\
B3 \\
\end{array}
\]

\[
\begin{array}{c}
\text{different} \\
\end{array}
\]
Multiplexer (“mux”)

• Connects one of $n$ inputs to the output
  – *select* control signals pick one of the $n$ sources
    • $\lceil \log_2 n \rceil$ select bits

• Useful when multiple data sources need to be routed to a single destination
  – Often arises from resource sharing
  – Example: select 1-of-$n$ data inputs to an adder
2-to-1 Mux

- Selects one of two inputs to appear at the output
- \( Y = S' \cdot I0 + S \cdot I1 \)
4-to-1 Mux

• Selects one of four inputs to appear at the output

• \[ Y = S_1' \cdot S_0' \cdot I_0 + S_1' \cdot S_0 \cdot I_1 + S_1 \cdot S_0' \cdot I_2 + S_1 \cdot S_0 \cdot I_3 \]
Cascading Multiplexers

- Large multiplexers can be implemented by cascading smaller ones

![Cascading Multiplexers Diagram]
Logic Functions Using Muxes

- Any function of n variables can be implemented with a $2^n:1$ multiplexer
  - Input variables connected to select inputs
  - Data inputs tied to 0 or 1 according to truth table

<table>
<thead>
<tr>
<th>ABC</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
</tr>
</tbody>
</table>
Getting Away with a Smaller Mux

- Can use \(2^{n-1}:1\) multiplexer and at most one inverter
  - Connect \(n-1\) input variables to select inputs
  - Data inputs tied to 0, 1, \(n^{th}\) variable, or inverted \(n^{th}\) variable

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

8:1 MUX

Lecture 5: 21
Decoder

- **Binary decoders**
  - n inputs, $2^n$ outputs
  - Each output corresponds to a unique input value
  - At most one output asserted at a time

- **Example: A 1-to-2 decoder**

<table>
<thead>
<tr>
<th>A</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$Y_0 = A'$

$Y_1 = A$
2-to-4 Decoder

\[
\begin{array}{c|ccccc}
A_1 & A_0 & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[Y_0 = A_1' \cdot A_0'\]
\[Y_1 = A_1' \cdot A_0\]
\[Y_2 = A_1 \cdot A_0'\]
\[Y_3 = A_1 \cdot A_0\]
Logic Functions Using Decoders

- $n:2^n$ decoder can be used to implement any function of $n$ variables
  - Connect variables to inputs
  - Appropriate minterms summed using extra gates to form the function
Logic Functions Using Decoders

- $F_1 = A'B'C'D' + AB'CD' + ABC'D'$
- $F_2 = A'B'C' + A'B'CD$
- $F_3 = A+B+C+D = (A'B'C'D')'$
Decoder with Enable

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₀</th>
<th>Y₀</th>
<th>Y₁</th>
<th>Y₂</th>
<th>Y₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>A₁</th>
<th>A₀</th>
<th>Y₃</th>
<th>Y₂</th>
<th>Y₁</th>
<th>Y₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>1</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>0</td>
</tr>
</tbody>
</table>

X: don’t care input
Here 0XX covers four input combinations 000, 001, 010, 011
Decoder with Enable (2)

\[ Y_3 = A_1 \cdot A_0 \cdot E \]
\[ Y_2 = A_1 \cdot A_0' \cdot E \]
\[ Y_1 = A_1' \cdot A_0 \cdot E \]
\[ Y_0 = A_1' \cdot A_0' \cdot E \]
Encoders

• Opposite of decoders

• Binary encoders: $2^n$ inputs and $n$ outputs
4-to-2 Encoder

Exactly one input is asserted at any given time

<table>
<thead>
<tr>
<th>I3</th>
<th>I2</th>
<th>I1</th>
<th>I0</th>
<th>Y1</th>
<th>Y0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Priority Encoder

• Highest numbered inputs have priority when multiple inputs are asserted at the same time
• Example: 4-to-2 priority encoder

<table>
<thead>
<tr>
<th>I3</th>
<th>I2</th>
<th>I1</th>
<th>I0</th>
<th>Y1</th>
<th>Y0</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Y1 = I3 + I3'I2 = I3 + I2
Y0 = I3 + I3'I2'I1 = I3 + I2'I1
None = I3'I2'I1'I0'
Before Next Class

• H&H 3.1-3.2

Next Time

Sequential Logic: Clocks, Latches, Flip-Flops