Overview

- Last lecture
  - Timing diagrams
  - Multilevel logic
    - Multilevel NAND/NOR conversion
    - AOI and OAI gates
  - Hazards
- Today
  - "Switching-network" logic blocks
    - Multiplexers/selectors
    - Demultiplexers/decoders
  - Programmable logic devices (PLDs)
    - Regular structures for 2-level logic

Switching-network logic blocks

- Multiplexer
  - Routes one of many inputs to a single output
  - Also called a selector
- Demultiplexer
  - Routes a single input to one of many outputs
  - Also called a decoder

Rationale: Sharing complex logic functions

- Share an adder: Select inputs; route sum

\[
\begin{array}{ccc}
A_0 & A_1 & B_0 \\
\downarrow & \downarrow & \downarrow \\
S_a & MUX & MUX \\
\downarrow & \downarrow & \downarrow \\
A & B & S_b \\
\downarrow & \downarrow & \downarrow \\
Sum & \text{MUX} & \text{MUX} \\
\downarrow & \downarrow & \downarrow \\
S_a & \text{DEMUX} & Z_0 \\
\downarrow & \downarrow & \downarrow \\
S & Z_1 \\
\end{array}
\]

Multiple inputs

Single adder

Multiple output destinations

Multiplexers

- Basic concept
  - \(2^n\) data inputs; \(n\) control inputs ("selects"); 1 output
  - Connects one of \(2^n\) inputs to the output
  - "Selects" decide which input connects to output
  - Two alternative truth-tables: Functional and Logical

Example: A 2:1 Mux

<table>
<thead>
<tr>
<th>(S)</th>
<th>(Z)</th>
<th>(I_n)</th>
<th>(I_n)</th>
<th>(S)</th>
<th>(Z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>0</td>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Functional truth table

Logical truth table

We construct these devices from:
- (1) logic gates
- (2) networks of transistor switches
Logic-gate implementation of multiplexers

2:1 mux

4:1 mux

Multiplexers (con't)

- 2:1 mux: \( Z = S'\text{In}_0 + S\text{In}_1 \)
- 4:1 mux: \( Z = S_0'S_1'\text{In}_0 + S_0'S_1\text{In}_1 + S_0S_1'\text{In}_2 + S_0S_1\text{In}_3 \)
- 8:1 mux: \( Z = S_0'S_1'S_2'\text{In}_0 + S_0'S_1S_2\text{In}_1 + ... \)

Cascading multiplexers

- Can form large multiplexers from smaller ones
- Many implementation options

Multiplexers as general-purpose logic

- A \(2^n:1\) mux can implement any function of \(n\) variables
  - A lookup table
  - A \(2^{n-1}:1\) mux also can implement any function of \(n\) variables
- Example: \(F(A,B,C) = m_0 + m_2 + m_6 + m_7\)
  \(= A'B'C' + A'BC' + ABC' + ABC\)
  \(= A'B'(C') + A'B(C') + AB(0) + AB(1)\)
Multiplexers as general-purpose logic

- Implementing a $2^n:1$ mux as a function of $n-1$ variables
  - $(n-1)$ mux control variables $S_0 - S_{n-1}$
  - One data variable $S_n$
  - Four possible values for each data input: 0, 1, $S_n, S_n'$
  - Example: $F(A,B,C,D)$ implemented using an 8:1 mux

Demultiplexers

- Basic concept
  - Single data input; $n$ control inputs (“selects”); $2^n$ outputs
  - Single input connects to one of $2^n$ outputs
  - “Selects” decide which output is connected to the input
  - When used as a decoder, the input is called an “enable” ($G$)

Demultiplexers as general-purpose logic

- A $n:2^n$ demux can implement any function of $n$ variables
  - Use variables as select inputs
  - Tie enable input to logic 1
  - Sum the appropriate minterms (extra OR gate)

Logic-gate implementation of demultiplexers

1:2 demux

2:4 demux

3:8 demux

Demultiplexer “decodes” appropriate minterms from the control signals
Demultiplexers as general-purpose logic

Example

\[ F_1 = A'B'CD + A'B'CD + ABCD \]
\[ F_2 = ABC'D' + ABC \]
\[ F_3 = (A'+B'+C'+D') \]

Cascading demultiplexers

- 5:32 demux

Programmable logic (PLAs & PALs)

- Concept: Large array of uncommitted AND/OR gates
  - Actually NAND/NOR gates
  - You program the array by making or breaking connections
    - Programmable block for sum-of-products logic

All two-level logic functions are available

- You "program" the wire connections
Sharing product terms

- **Example:**
  - \( F_0 = A + B'C' \)
  - \( F_1 = AC' + AB \)
  - \( F_2 = B'C' + AB \)
  - \( F_3 = B'C + A \)

- **Personality matrix:**

<table>
<thead>
<tr>
<th>( AB )</th>
<th>( B'C )</th>
<th>( AC' )</th>
<th>( B'C' )</th>
<th>( A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

  **inputs**
  - 1 = asserted in term
  - 0 = negated in term
  - – = does not participate

  **outputs**
  - 1 = term connected to output
  - 0 = no connection to output

  Reuse terms

Programming the wire connections

- **Fuse:** Comes connected; break unwanted connections
- **Anti-fuse:** Comes disconnected; make wanted connections

\[
\begin{align*}
F_0 &= A + B'C' \\
F_1 &= AC' + AB \\
F_2 &= B'C' + AB \\
F_3 &= B'C + A
\end{align*}
\]