Overview

- Last lecture
  - "Switching-network" logic blocks
  - Multiplexers/selectors
  - Demultiplexers/decoders
  - Programmable logic devices (PLDs)
  - Regular structures for 2-level logic

- Today
  - PLDs
  - FPLAs
  - PALs
  - ROMs
  - Tristates
  - Design examples

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

Programming the wire connections

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

Short-hand notation

- Draw multiple wires as a single wire or bus
- x signifies a connection

PLA example

- Think of as a memory-address decoder

Programmable array logic (PAL)

- Fully programmable AND / OR arrays
- Can share AND terms

- Programmable array logic (PAL)
  - Programmable AND array
  - OR array is prewired
  - No sharing ANDs
  - Cheaper and faster than PLAs
Example: BCD to Gray code converter

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

K-map for W

K-map for X

K-map for Y

K-map for Z

Example (con't): Wire a PLA

Minimized functions:

- \( W = A + BC + BD \)
- \( X = BC' \)
- \( Y = B + C \)
- \( Z = AB'C'D + BCD + AD' + B'CD' \)

Compare implementations

- **PLA:**
  - No shared logic terms in this example
  - 10 decoded functions (10 AND gates)
- **PAL:**
  - \( Z \) requires 4 product terms
  - 16 decoded functions (16 AND gates)
  - 8 unused AND gates
- This decoder is a poor candidate for PLAs/PALs
- 10 of 16 possible inputs are decoded
- No sharing among AND terms
- Better option?
  - Yes — a ROM

Read-only memories (ROMs)

- Two dimensional array of stored 1s and 0s
  - Input is an address \( \Rightarrow \) ROM decodes all possible input addresses
  - Stored row entry is called a "word"
  - ROM output is the decoded word

![ROM diagram]

ROM details

- Similar to a PLA but with a fully decoded AND array
- Completely flexible OR array (unlike a PAL)
- Extremely dense: One transistor per stored bit

![ROM detail diagram]
**Two-level combinational logic using a ROM**

- Use a ROM to directly store a truth table
  - No need to minimize logic
  - Example:
    - F0 = A'B'C + A'B'C + AB'C
    - F1 = A'B'C + A'B'C + ABC
    - F2 = A'B'C + A'B'C + ABC
    - F3 = A'B'C + A'B'C + ABC

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F1</th>
<th>F2</th>
<th>F3</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>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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</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>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

You specify whether to store 1 or 0 in each location in the ROM

**ROMs versus PLAs/PALs**

- **ROMs**
  - **Benefits**
    - Quick to design, simple, dense
  - **Limitations**
    - Size doubles for each additional input
    - Can’t exploit don’t cares

- **PLAs/PALs**
  - **Benefits**
    - Logic minimization reduces size
  - **Limitations**
    - PAL OR-plane has hard-wired fan-in

- Another answer: Field programmable gate arrays
  - Learn about in 467

**Example: BCD to 7-segment display controller**

- The problem
  - Input is a 4-bit BCD digit (A, B, C, D)
  - Need signals to drive a display (7 outputs C0 – C6)

**Formalize the problem**

- **Truth table**
  - Many don’t cares

- **Choose implementation target**
  - If ROM, we are done
  - Don’t cares imply PAL/PLA may be good choice

- **Implement design**
  - Minimize the logic
  - Map into PAL/PLA

**Loose end: Tristates**

- Tristate buffers have a control input
  - Enabled: Buffer works normally
  - Disabled: Buffer output is disconnected

**Sum-of-products implementation**

- 15 unique product terms if we minimize individually
Better SOP implementation

- Can do better than 15 product terms
  - Share terms among outputs ⇒ only 9 unique product terms
  - Each term not necessarily minimized

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A xor B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xor B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>always 0</td>
<td></td>
</tr>
</tbody>
</table>

C0 = A + BD + C + BD' |
C1 = CD' + CD + B' |
C2 = B + C + D |
C3 = BD + BD' + BCD + BFC |
C4 = BCD + CD' |
C5 = A + CD' + BD + BC' |
C6 = A + CD + BC + BD' |
C7 = BCD + CD + BD' + BCD' |

Formalize the problem and solve

Implementation choice: multiplexer with discrete gates

PLA implementation

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A xor B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xor B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>always 0</td>
<td></td>
</tr>
</tbody>
</table>

C0 = BCD + CD + BD' + BCD' + A |
C1 = BTD + CD' + CD + B' |
C2 = BTD + BCD + CD' + CD + BCD' |
C3 = BCD + BD + BD' + BCD' |
C4 = BCD + CD' |
C5 = A + CD' + BD + BC' |
C6 = A + CD + BC + BD' |
C7 = BCD + CD + BD' + BCD' |

Pal Feature: Tri-stated outputs

Example: Logical function unit

- Multipurpose functional block
  - 3 control inputs (C) specify function
  - 2 data inputs (operands) A and B
  - 1 output (same bit-width as input operands)

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>always 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td>logical OR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td>logical NAND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A xor B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xor B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td>logical AND</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B')</td>
<td>logical NAND</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>always 0</td>
<td>logical NOR</td>
</tr>
</tbody>
</table>

Pal Feature: Individually Tri-stated outputs