Lecture 11

- **Logistics**
  - HW3 due now
  - Lab4 goes on as normal next week
  - Tuesday review 6pm, place TBD

- **Last lecture**
  - "Switching-network" logic blocks
    - Multiplexers and Demultiplexers

- **Today**
  - PLDs
    - PLAs
    - PALs
    - ROMs

The “WHY” slide

- **Programmable Logic Devices (PLDs)**
  - Often you want to have a look up table of functions stored away somewhere in your device. Rather than having specific circuits build every time, it would be nice to have a "general-purpose" structure that could be “programmed” for a specific usage. PLDs have a generic structure that allows any function to be expressed and stored.

  - And it is nice if it is reprogrammable. Some PLDs are reprogrammable (like your memory sticks).
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

![Diagram of Programmable Logic Array](Diagram.png)

Programming the wire connections

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

F0 = A + B'C'
F1 = AC' + AB
F2 = B'C' + AB
F3 = B'C + A

![Diagram of Wire Connections](Diagram.png)
Short-hand notation

- Draw multiple wires as a single wire or bus
- $\times$ signifies a connection

Before Programming

After Programming

$F_0 = AB + A'B'$

$F_1 = CD' + C'D$

PLA example

$F_1 = ABC$
$F_2 = A + B + C$
$F_3 = A'B' C'$
$F_4 = A' + B' + C'$
$F_5 = A \text{ xor } B \text{ xor } C$
$F_6 = A \text{ xnor } B \text{ xnor } C$

Think of as a memory-address decoder

Memory bits

CSE370, Lecture 11
PLAs versus PALs

- We've been looking at PLAs
  - Fully programmable AND / OR arrays

- Programmable array logic (PAL)
  - Programmable AND array
  - OR array is prewired
    - Cheaper and faster than PLAs

Example: BCD to Gray code converter
Example: BCD to Gray --- Wire a PLA

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

Example: Wire a PAL

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

Fine example for the use of PAL (because no shared AND terms)

Many AND gates wasted, but still faster and cheaper than PLA
Compare implementations for this example

- **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)
    - 6 unused AND gates
  - This decoder is a best candidate for PLAs/PALs
    - 10 of 16 possible inputs are decoded
    - No sharing among AND terms

- **Another 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

![Diagram of a ROM](image)
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

```
2^n-1
+5V
```

Only one word line is active at any time

```
Address
0 n-1
```

```
Outputs
0 m-1
```

Bit lines: Normally pulled high through resistor. If transistor stores a zero, then line pulls low when row is selected

---

Two-level combinational logic using a ROM

- Use a ROM to directly store a truth table
  - No need to minimize logic
  - Example: 
    
    \[
    F_0 = A'B'C + AB'C' + AB'C \\
    F_1 = A'B'C + A'BC' + ABC \\
    F_2 = A'B'C' + A'BC + AB'C' \\
    F_3 = A'BC + AB'C' + ABC'
    \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</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>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>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8 words x 4 bits/word

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

---

CSE370, Lecture 11
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
    - PALs faster/cheaper than PLAs
  - **Limitations**
    - PAL OR-plane has hard-wired fan-in

- Another alternative: Field programmable gate arrays
  - Learn a bit more later in this class

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Sum-of-products implementation

- 15 unique product terms if we minimize individually

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

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

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

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

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

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

C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

4 input, 7 output
PLA: 15 AND gates
PAL: 4 product terms per output (28 AND gates)
If choosing PLA: better SOP implementation

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

\[
\begin{align*}
C_0 &= BC'D + CD + B'D' + BCD' + A \\
C_1 &= B'D + C'D' + CD + B'D' \\
C_2 &= B'D + BC'D + C'D' + CD + BCD' \\
C_3 &= B'D' + CD' + BC'D + B'C \\
C_4 &= B'D' + CD' \\
C_5 &= A + C'D' + BD' + BC' \\
C_6 &= A + CD' + BC' + B'C
\end{align*}
\]

PLA implementation

\[
\begin{align*}
C_0 &= BC'D + CD + B'D' + BCD' + A \\
C_1 &= B'D + C'D' + CD + B'D' \\
C_2 &= B'D + BC'D + C'D' + CD + BCD' \\
C_3 &= BC'D + B'D + B'D' + BCD' \\
C_4 &= B'D' + BCD' \\
C_5 &= BC'D + C'D' + A + BCD' \\
C_6 &= B'C + BC' + BCD' + A
\end{align*}
\]