Lecture 9 Multiplexer, Decoder, and PLD

- **SSI** (small-scale integration)
  - NAND, NOR, NOT, flip flop etc
    - Gate count < 10.

- **MSI** (medium-scale integration)
  - Adders, multiplexers, decoders, registers, counters
    - Gate count < 100

- **LSI**

- **VLSI** (very large-scale integration)
  - Memory chips, microprocessors
Multiplexers

• Multiplexers are selectors.

\[ Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3. \] (4-to-1 MUX)
Logic Diagram for MUX

- 8-to-1 MUX
Application of MUX

- Select data.
Application of MUX

Crossbar switch
• As s=0, x1 to y1, x2 to y2.
• As s=1, x1 to y2, x2 to y1.

(a) A 2x2 crossbar switch

(b) Implementation using multiplexers
Using Mux for logic function, XOR

- $f = w_1 \text{ xor } w_2$

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$f$</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>0</td>
</tr>
</tbody>
</table>

![MUX Circuit Diagram]
Better

- Modify the truth table.
- When \( w_1 = 0 \), \( f = w_2 \). Otherwise, \( f = \neg w_2 \)

\[
\begin{array}{ccc}
  \text{\( w_1 \)} & \text{\( w_2 \)} & \text{\( f \)} \\
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cc}
  \text{\( w_1 \)} & \text{\( f \)} \\
  0 & \text{\( w_2 \)} \\
  1 & \neg \text{\( w_2 \)} \\
\end{array}
\]

```circuit
\text{\( w_1 \)}  \text{\( w_2 \)}  \text{\( f \)}
```

C-H 7
Using Mux to Implement a barrel shifter

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_0$</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$w_3$</td>
<td>$w_2$</td>
<td>$w_1$</td>
<td>$w_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$w_0$</td>
<td>$w_3$</td>
<td>$w_2$</td>
<td>$w_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$w_1$</td>
<td>$w_0$</td>
<td>$w_3$</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$w_2$</td>
<td>$w_1$</td>
<td>$w_0$</td>
<td>$w_3$</td>
</tr>
</tbody>
</table>

(a) Truth table

(b) Circuit
Buffers/Tri-state Buffer

- Driving capability
- High impedance
Tri-state Buffer Application

- Data transfer between registers
Tri-state Buffer Application (cont.)

- Source selection
Decoder

- 3-to-8 line decoder
  - An n-to-2^n decoder generates all 2^n minterms of the n input variables.
More Decoder

• 2-to-4 decoder: minterm generator

Figure 3.5.3  Decoder: (a) gate-level circuit diagram, (b) logic symbol.
Decoder (cont.)

- 4-to-10 line decoder
ROM

- Read-only memory: stored data cannot be changed under normal operating conditions.

![Diagram of ROM](image)

(a) block diagram (b) truth table for ROM

<table>
<thead>
<tr>
<th>A B C</th>
<th>F₀ F₁ F₂ F₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 1 0 1</td>
</tr>
</tbody>
</table>

typical data stored in ROM

(2³ words of 4 bits each)
ROM (cont.)

- ROM size = $2^n \times m$ bits.
ROM Basic Structure

- 8-word x 4 bit ROM
  - \( F_0 = m_0 + m_1 + m_4 + m_6 \)
Multiple-Output Network Using ROM

- BCD to Excess-3 code
Types of ROM

• Mask-programmable ROM
  – Contents are stored during fabrication.

• Field-programmable ROM
  – PROM: programmable ROM
    • Fusible link (PROM programmer)
  – EPROM: erasable PROM
    • Use ultraviolet light for erasure.
  – EEPROM: electrically erasable PROM.
    • Flash memory
Programmable Logic Devices

- PLA (programmable logic array)
  - Realize m functions of n variables.
Implementation procedure for PLA

• Prepare the truth table based on your system
• Write the Boolean expression in SOP (sum of product) form.
• Obtain the minimum SOP form to reduce the number of product terms to a minimum.
• Decide the input connection of the AND matrix for generating the required product term.
• Then decide the input connections of OR matrix to generate the sum terms.
• Decide the connections of invert matrix.
• Program the PLA.
Programmable Symbology

- **No fuse**
  - Fixed connection at factory

- **Intact fuse**
  - Programmable connection

- **Intact fuse**
  - Simplified representation

- **Blown fuse**
  - Connection broken (after programming)

- **Blown fuse**
  - Simplified representation

**Multiple input AND gate**
- Pull-up resistors not shown

**Input terms**
- $A \overline{A} B \overline{B} \ldots$

**All fuses intact**
- $A \cdot \overline{A} \cdot B \cdot \overline{B} = 0$

**An X placed inside an AND gate also represents all fuses intact**

**Multiply input OR gate**
- Pull-down resistors not shown

**Product terms**
- $p_1 p_2 p_3 p_4 \ldots$

**All fuses intact**
- $p_1 + p_2 + p_3 + p_4 \ldots$

**An X placed inside an OR gate also represents all fuses intact**

**Input terms**
- $A \overline{A} B \overline{B} \ldots$

**All fuses blown**
- 1 (Due to pull-up resisters)

**Product terms**
- $p_1 p_2 p_3 p_4 \ldots$

**All fuses blown**
- 0 (Due to pull-down resisters)

**Figure 3.6.2** Programmable symbology summary for the different types of PLDs.
PLA

- PLA 3 inputs, 5 product terms, 4 outputs.

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs $A$ $B$ $C$</th>
<th>Outputs $F_0$ $F_1$ $F_2$ $F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A'B'$</td>
<td>0 0</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>$AC'$</td>
<td>1 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>$B$</td>
<td>0 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>$BC'$</td>
<td>0 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>$AC$</td>
<td>1 1</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

$F_0 = A'B' + AC'$

$F_1 = AC' + B$

$F_2 = A'B' + BC'$

$F_3 = B + AC$
Figure 3.6.4  PLA circuit: (a) circuit representation using gates, (b) simplified circuit representation using gates.
PLA (cont.)

- PLA 3 inputs, 5 product terms, 4 outputs. AND-OR array.
PLA Application

- $f_1 = a'b'd + abd + ab'c' + b'c$
- $f_2 = c + a'b'd$
- $f_3 = bc + ab'c' + abd$

- If $abcd = 0111$, 1st, 5th, 6th rows are selected. $f_1 = 1 +0+0$, $f_2 = 1+1+0$, etc.
Programmable Array Logic (PAL)

- AND array is programmable. OR array is fixed.
- Use input buffer to drive many ANDs.
Programmable Array Logic (PAL)

- Full Adder using a PAL
Why prefers PLA rather than ROM

• A combinational circuit may occasionally have don't-care conditions. When implemented with a ROM, a don't care condition becomes an address input that will never occur.

• The words at the don't care address need not be programmed and may be left in their original state (all 0's or all 1's).

• The has the result that all the bit patterns available in ROM are used, but for the don’t care addresses you really can do without them.
What Programmable Means

- "programmable" does not indicate that all PLAs are field-programmable.
- Many are mask-programmed during manufacture in the same manner as a mask ROM.
  - This is particularly true of PLAs that are embedded in more complex and numerous integrated circuits such as microprocessors.
- PLAs that can be programmed after manufacture are called FPGA (Field-programmable gate array)
Complex Programming
Logic Devices

- Xilinx XCR3064XL CPLD
  - Function block (16 macrocells) = PLA
  - Macrocell = a flip flop + multiplexers
  - IA routes signals
Function Block and MC

- Signal from PLA -> marcocell -> I/O pin
- Use CAD tool to fit the design into the PLD.
Field Programmable Gate Arrays (FPGA)

- Logic cell: configurable logic blocks (CLBs)
- Input/Output blocks (I/O blocks)
Configurable Logic Block

- Inside a CLB: function generators (LUT), FFs, and MUXs
- LUT: lookup table (truth table) is a reprogrammable ROM (16 1-bit words)

*= Programmable MUX
A Lookup Table (LUT)

- If we want $F = abc$ (one minterm)
  - $1110$ (and $F=1$) + $1111$ (and $F=1$)
- Or if we want $F = a'b'c'd' + a'b'cd' + \ldots abcd$. (15 minterms)
- Require a single function generator. Program the LUT table to get what we want.

<table>
<thead>
<tr>
<th>a b c d</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

To be implemented

This is a 4-variable function generator.

Bit stored in the LUT.
Shannon’s Expansion Theorem

• What if # of variables > 4 variables

\[ f(x_1, x_2, \ldots, x_n) = x_i' f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) = x_i' f_0 + x_i f_1 \]

\[ f(a, b, c, d, e) = a' f(0, b, c, d, e) + a f(1, b, c, d, e) \]

Let \( a = 0 \), what lefts are terms with \( b, c, d, e \)

2-1 MUX \( F = a' I_1 + a I_2 \)