Programmable Logic

- Regular logic
  - Programmable Logic Arrays
  - Multiplexers/Decoders
  - ROMs
- Field Programmable Gate Arrays
  - Xilinx Vertex

Programmable Logic Arrays (PLAs)

- Pre-fabricated building block of many AND/OR gates
  - Actually NOR or NAND
  - "Personalized" by making or breaking connections among gates

Programmable array block diagram for sum of products form

```
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>F0</td>
</tr>
<tr>
<td></td>
<td>F1</td>
</tr>
<tr>
<td></td>
<td>F2</td>
</tr>
<tr>
<td></td>
<td>F3</td>
</tr>
</tbody>
</table>
```

Enabling Concept

- Shared product terms among outputs

```
F0 = A + B'C
F1 = A'C + A'B
F2 = B'C + A'B
F3 = B'C' + A
```

Before Programming

- All possible connections available before "programming"
  - In reality, all AND and OR gates are NANDs

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

Alternate Representation for High Fan-in Structures

- Short-hand notation--don’t have to draw all the wires
  - Signifies a connection is present and perpendicular signal is an input to gate

```
notation for implementing
F0 = A B + A'B'
F1 = C'D + C'D
```

```
A B C D
F0 F1 F2 F3
```

After Programming

- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)

```
A B C D
F0 F1 F2 F3
```

```
A B C D
F0 F1 F2 F3
```

```
A B C D
F0 F1 F2 F3
```

```
A B C D
F0 F1 F2 F3
```

```
A B C D
F0 F1 F2 F3
```

```
A B C D
F0 F1 F2 F3
```
Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A'B'C
  - F2 = A • B • C
  - F3 = A' • B' • C'
  - F4 = A + B + C
  - F5 = A • B • C
  - F6 = (A xor B xor C)

<table>
<thead>
<tr>
<th>Function</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Full decoder as for memory address
Bits stored in memory

PLA Design Example

- BCD to Gray code converter

<table>
<thead>
<tr>
<th>Function</th>
<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>F1</td>
<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>F1</td>
<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>F1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>1</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>F1</td>
<td>1</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>F1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map for W
K-map for X
K-map for Y
K-map for Z

PLA Design Example (cont'd)

- Code converter: programmed PLA

<table>
<thead>
<tr>
<th>Function</th>
<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>F1</td>
<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>F1</td>
<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>F1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>1</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>F1</td>
<td>1</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>F1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map for W
K-map for X
K-map for Y
K-map for Z

Minimized functions:
- W = A • B + B • C
- X = B • C
- Y = B + C
- Z = A'B'C + B'CD + A'D + B'C'D

PLA Design Example #1

- BCD to Gray code converter

<table>
<thead>
<tr>
<th>Function</th>
<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>F1</td>
<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>F1</td>
<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>F1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>1</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>F1</td>
<td>1</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>F1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map for W
K-map for X
K-map for Y
K-map for Z

Minimized functions:
- W = A • B + B • C
- X = B • C
- Y = B + C
- Z = A'B'C + B'CD + A'D + B'C'D

PLA Design Example #2

- Magnitude comparator

<table>
<thead>
<tr>
<th>Function</th>
<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>F1</td>
<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>F1</td>
<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>F1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>1</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>F1</td>
<td>1</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>F1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map for W
K-map for X
K-map for Y
K-map for Z
Connections

- Direct point-to-point connections between gates
- Multiplexer: route one of many inputs to a single output
- Demultiplexer: route single input to one of many outputs

Example:

- 4:1 mux: Direct point-to-point connections between gates
- In general, \( Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \)
- In essence, \( Z = \Sigma_{i=0}^{7} m_I_i \)
- \( m_I_i \) in minterm shorthand form for a 2:1 Mux

Multiplexers/Selectors (cont'd)

- 2:1 mux: \( Z = A'I_0 + A'I_1 \)
- 4:1 mux: \( Z = A'B'I_0 + A'B'I_1 + A'BI_2 + AB'I_3 \)
- 8:1 mux: \( Z = A'B'C'I_0 + A'B'C'I_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \)
- In general, \( Z = \sum_{i=0}^{7} m_{I_i} \)

Multiplexers as Lookup Tables (LUTs)

- 2^n:1 multiplexers implement any function of \( n \) variables
- With the variables used as control inputs and
- Data inputs tied to 0 or 1
- In essence, a lookup table

Example:

- \( F(A,B,C) = m_0 + m_2 + m_6 + m_7 \)
- \( F(A,B,C) = A'B'C' + A'B'C + ABC + ABC \)
- \( F(A,B,C) = A'B'(C) + A'B(C) + AB'(0) + AB(1) \)

Multiplexers as LUTs (cont'd)

- 2^n:1 mux can implement any function of \( n \) variables
- With \( n \) variables used as control inputs and
- Data inputs tied to the last variable or its complement

Example:

- \( F(A,B,C) = m_0 + m_2 + m_6 + m_7 \)
- \( F(A,B,C) = A'B'C' + A'B'C + ABC + ABC \)
- \( F(A,B,C) = A'B'(C) + A'B(C) + AB'(0) + AB(1) \)
Multiplexers as LUTs (cont’d)

- **Generalization**
  - Multiplexer with control inputs: $x_1, x_2, \ldots, x_{n-1}$
  - Single data input: $x_0$
  - Output as a function of the control inputs: $f(x_1, x_2, \ldots, x_{n-1})$

- **Example:** $F(A, B, C, D)$ implemented by an 8:1 MUX

Demultiplexers/Decoders

- **Decoders/demultiplexers:** general concept
  - Single data input, $n$ control inputs, $2^n$ outputs
  - Control inputs (called "selects") represent binary index of output to which the input is connected
  - Data input usually called "enable" ($G$

Demultiplexers as General-Purpose Logic

- **n2° decoder implements any function of n variables**
  - With the variables used as control inputs
  - Enable inputs tied to 1
  - Appropriate minterms summed to form the function

Demultiplexers as General-Purpose Logic (cont’d)

- $F_1 = A' B' C' D + A' B' C D + A B C D$
- $F_2 = A B C D' + A B C$
- $F_3 = (A' + B' + C' + D')$

Announcements

- We took everyone on the wait list into the class
  - Result is that Tu labs are very crowded!
  - Th night lab is very light — think of switching to get more TA face time!
- Send email to pokai@berkeley.edu to request a lab change
- First HW due Friday at 2 PM ... just before Lab Lecture
  - CS 150 hand-in box outside and just to the right of 125 Cory doors
  - Second HW on class web site
- Use ucb.class.cs150 newsgroup for lab, hw, course questions!
### Read-only Memories
- Two dimensional array of 1s and 0s
  - Entry (row) is called a "word"
  - Width of row = word-size
  - Index is called an "address"
  - Address is input
  - Selected word is output

### ROMs and Combinational Logic
- Combinational logic implementation (two-level canonical form) using a ROM
  \[
  \begin{align*}
  F_0 &= A'B'C + A'B'C + A'B'C \\
  F_1 &= A'B'C + A'B'C + A'B'C \\
  F_2 &= A'B'C + A'B'C + A'B'C \\
  F_3 &= A'B'C + A'B'C + A'B'C
  \end{align*}
  \]

### ROM Structure
- Similar to a PLA structure but with a fully decoded AND array
  - Completely flexible OR array (unlike PAL)

### ROM vs. PLA
- ROM
  - Design time is short (no need to minimize output functions)
  - Most input combinations are needed (e.g., code converters)
  - Little sharing of product terms among output functions
  - Size doubles for each additional input
  - Can't exploit don't cares
  - Cheap (high-volume component)
  - Can implement any function of n inputs
  - Medium speed
- PLA
  - Design tools are available for multi-output minimization
  - There are relatively few unique minterm combinations
  - Many minterms are shared among the output functions
  - Most complex in design, need more sophisticated tools
  - Can implement any function up to a product term limit
  - Slow (two programmable planes)

### Field-Programmable Gate Arrays
- PLAs: 100s of gate equivalents
- FPGAs: 1000-10000s gates
  - Logic blocks
    - Implement combinational and sequential logic
  - Interconnect
  - I/O blocks
    - Special logic blocks at periphery of device for external connections
  - Key questions:
    - How to make logic blocks programmable?
    - How to connect the wires?
    - After the chip has been fabbed

### Tradeoffs in FPGAs
- Logic block: how are functions implemented: fixed functions (manipulate inputs) or programmable?
  - Support complex functions, need fewer blocks, but they are bigger
  - Need more blocks, but they are smaller
  - More on chip
- Interconnect
  - How are logic blocks arranged?
  - How many wires will be needed between them?
  - Are wires evenly distributed across chip?
  - Programmability slows down - are some wires specialized to long distance?
  - How many inputs/outputs must be routed from each logic block?
  - What utilization are we willing to accept? 50% 20% 90%?
Xilinx 4000 Series Programmable Gate Arrays

- CLB - Configurable Logic Block
- 5-input, 1 output function
- or 2 4-input, 1 output functions
- optional register on outputs
- Built-in fast carry logic
- Can be used as memory
- Three types of routing
  - direct
  - general-purpose
  - long lines of various lengths
- RAM-programmable
- can be reconfigured

Two 4-Input Functions, Registered Output

CLB Used as RAM

5-Input Function, Combinational Output

Xilinx 4000 Interconnect
Xilinx FPGA Combinational Logic Examples

- Key: General functions are limited to 5 inputs
  - (4 even better - 1/2 CLB)
  - No limitation on function complexity
- Example
  - 2-bit comparator:
    \[ A \land B = C \land D \text{ and } A \lor B \lor C \lor D \text{ implemented with } 1 \text{ CLB} \]
    \[ \text{EQ} \quad G = A'B'C'D' + A'B'C'D + A'B'C'D + A'B'C'D' \]
- Can implement some functions of > 5 input

Xilinx FPGA Combinational Logic

- Examples
  - N-input majority function: 1 whenever n/2 or more inputs are 1
  - N-input parity functions: 5 input/1 CLB; 2 levels yield 25 inputs!

CLB

Xilinx FPGA Adder Example

- Example
  - 2-bit binary adder - inputs: A1, A0, B1, B0, CIN
    outputs: S0, S1, Cout
  - Full Adder, 4 CLB delays to final carry out
  - 2 x Two-bit Adders (3 CLBs each) yields 2 CLBs to final carry out

Xilinx FPGA Combinational Logic

- 5-input Majority Circuit
- 7-input Majority Circuit
- 9-input Parity Logic

Combinational Logic Implementation Summary

- Regular Logic Structures
  - Programmable Logic Arrays
  - Programmable connections: AND-OR (NOR-NOR) Arrays
  - Multiplexers/decoders
  - Multipoint connections for signal routing
  - Lookup Tables
  - ROMs
  - Truth table in hardware
  - Field Programmable Gate Arrays (FPGAs)
  - Programmable logic (LUTs, Truth Tables) and connections
  - Advantages/disadvantages of each