Programmable Logic

- Regular logic
  - Programmable Logic Arrays
  - Multiplexers/Decoders
  - ROMs
- Field Programmable Gate Arrays (FPGAs)
  - Xilinx

Programmable Logic Arrays

- 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

Enabling Concept

- Shared product terms among outputs
  - Example:
    - $F_0 = A + B'C'$
    - $F_1 = AC' + AB$
    - $F_2 = B'C' + AB$
    - $F_3 = B'C + A$
  - Personality matrix:
    - Inputs: $A, B, C$
    - Outputs: $F_0, F_1, F_2, F_3$

Before Programming

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

After Programming

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

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

### Diagrams

- Programmable Logic Arrays block diagram
- Personality matrix
- Enabling Concept
- Before and After Programming examples
- Alternate Representation for High Fan-in Structures
Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A \cdot B \cdot C
  - F2 = A + B + C
  - F3 = \overline{A} \cdot \overline{B} \cdot \overline{C}
  - F4 = \overline{A} + \overline{B} + \overline{C}
  - F5 = A \oplus B \oplus C
  - F6 = A \oplus B \oplus C

*Note: All decoders as for memory address bits stored in memory*

Programmable Logic Array Example (cont'd)

- Code converter: programmed PLA
  
  - minimized functions:
    - W = A \cdot B \cdot D + B \cdot C
    - X = B \cdot C'
    - Y = B + C
    - Z = A' \cdot B' \cdot C' \cdot D + B \cdot C \cdot D + A \cdot D' + B' \cdot C \cdot D'

*Note: Not a particularly good candidate for PLA implementation since no terms are shared among outputs. However, much more compact and regular implementation when compared with discrete AND and OR gates*

PLAs Design Example

- BCD to Gray code converter
  
  - minimized functions:
    - W = A \cdot B \cdot D + B \cdot C
    - X = B \cdot C'
    - Y = B + C
    - Z = A' \cdot B' \cdot C' \cdot D + B \cdot C \cdot D + A \cdot D' + B' \cdot C \cdot D'

PLAs Design Example (cont'd)

- Code converter: programmed PLA
  
  - minimized functions:
    - W = A \cdot B \cdot D + B \cdot C
    - X = B \cdot C'
    - Y = B + C
    - Z = A' \cdot B' \cdot C' \cdot D + B \cdot C \cdot D + A \cdot D' + B' \cdot C \cdot D'

*Note: Not a particularly good candidate for PLA implementation since no terms are shared among outputs. However, much more compact and regular implementation when compared with discrete AND and OR gates*

PLA Second Design Example

- Magnitude comparator
  
  - K-maps for EQ, NE, LT, GT

Multiplexers/Selectors

- Multiplexers/Selectors: general concept
  1. 2 data inputs, n control inputs (called 'selects'), 1 output
  2. Used to connect 2^n points to a single point
  3. Control signal pattern forms binary index of input connected to output

- Multiplexer: route one of many inputs to a single output
- Demultiplexer: route single input to one of many outputs

Multiplexers/Selectors: Making 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
Multiplexers/Selectors (cont'd)

- 2:1 mux: \( Z = A'B + A'B' \)
- 4:1 mux: \( Z = A'B' + A'B + A'B' + A'B' \)

In general, \( Z = \sum_{i=0}^{m-1} (m_i I_i) \)
- In minterm shorthand form for a 2^m:1 MUX

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

Generalization
- With the variables used as control inputs and
- Data inputs tied to 0 or 1
- In essence, a lookup table
- Example:
  \( R(A, B, C) = m_0 + m_2 + m_6 + m_7 \)
  - \( A'B'C' + A'B'C + A'B'C + A'B'C \)
  - \( A'B'C + A'B'C + A'B'C + A'B'C \)

Multiplexers as General-purpose Logic (cont’d)

- 2^m: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:
- \( R(A, B, C) = m_0 + m_2 + m_6 + m_7 \)
- \( A'B'C + A'B'C + A'B'C + A'B'C \)

Demultiplexers/Decoders

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

- Truth table for 2 and 3:1 Decoders:
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 + S_1 \cdot S_2 \cdot S_3 \)
  - \( O_0 = G \cdot S_0 + S_1 \cdot S_2 + S_1 \cdot S_2 \)
  - \( O_0 = G \cdot S_0 + S_1 \cdot S_2 \cdot S_3 \)

- Truth table for 2 and 3:8 Decoders:
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \)

- Truth table for 2 and 3:16 Decoders:
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \cdot S_7 \cdot S_8 \cdot S_9 \cdot S_10 \cdot S_11 \cdot S_12 \cdot S_13 \cdot S_14 \cdot S_15 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \cdot S_7 \cdot S_8 \cdot S_9 \cdot S_10 \cdot S_11 \cdot S_12 \cdot S_13 \cdot S_14 \cdot S_15 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \cdot S_7 \cdot S_8 \cdot S_9 \cdot S_10 \cdot S_11 \cdot S_12 \cdot S_13 \cdot S_14 \cdot S_15 \)
  - \( O_0 = G \cdot S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 \cdot S_7 \cdot S_8 \cdot S_9 \cdot S_10 \cdot S_11 \cdot S_12 \cdot S_13 \cdot S_14 \cdot S_15 \)
Demultiplexers as General-purpose Logic

- n:2 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

Demultiplexer generates appropriate minterm based on control signals
(“decodes” control signals)

\[
\begin{align*}
F_1 &= A'B'C'D + A'B'C + A'BC' + ABC' \\
F_2 &= A'B'C' + A'B'C + A'BC + ABC \\
F_3 &= (A' + B' + C' + D')
\end{align*}
\]

Cascading Decoders

- 5:32 decoder
- 1x2:4 decoder
- 4x3:8 decoders

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

\[
\begin{align*}
F_1 &= A'B'C'D + A'B'C + A'BC' + ABC' \\
F_2 &= A'B'C' + A'B'C + A'BC + ABC \\
F_3 &= (A' + B' + C' + D')
\end{align*}
\]

ROM Structure
**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
  - Size (two programmable planes)

---

**Field-Programmable Gate Arrays**

- **PLAs:** 100s of gate equivalents
- **FPGAs:** 1000-10000s gates

- **Logic blocks**
  - Implement combinational and sequential logic
- **Interconnect**
  - Wires to connect inputs and outputs to logic blocks
- **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 fabricated?

---

**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
  - Support simple functions, need more blocks, but they are smaller

- **Interconnect**
  - How are logic blocks arranged?
  - How many wires will be needed between them?
  - Are wires evenly distributed across chip?
  - Programmability of a bus wired down - some wires specialized to long distances?

- **I/O blocks**
  - How many inputs/outputs must be routed to/from each logic block?
  - Can logic be customized to accept 50% 20% 90%?

---

**Xilinx 4000 Series Programmable Gate Arrays**

- **CLB - Configurable Logic Block**
  - 5-input, 1 output function
  - or 24-input, 1 output functions
  - Optional register on outputs
  - Can be used as memory
  - Three types of routing
  - Direct
  - General-purpose
  - Long lines of various lengths
- **RAM-programmable**
  - Can be reconfigured

---

**The Xilinx 4000 CLB**

Two 4-input functions, registered output

---

**Figure 1:** Simplified Block Diagram of XC3000 Series CLB (533 and Carry Logic functions not shown)
5-input function, combinational output

CLB Used as RAM

Xilinx 4000 Interconnect

Xilinx FPGA Combinational Logic Examples

Xilinx FPGA Combinational Logic

Xilinx FPGA Adder Example
Combinational Logic Implementation Summary

- Regular Logic Structures
  - PLA/PALs
  - Multiplexers/decoders
  - ROMs
  - Advantages/disadvantages of each