Chapter # 4: Programmable and Steering Logic

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley
June 1993

PALs and PLAs
Pre-fabricated building block of many AND/OR gates (or NOR, NAND) "Personalized" by making or breaking connections among the gates

Programmable Array Block Diagram for Sum of Products Form
Key to Success: Shared Product Terms

Example:

\[
\begin{align*}
F_0 &= A + B' C' \\
F_1 &= A C' + A B \\
F_2 &= B' C' + A B \\
F_3 &= B' C + A
\end{align*}
\]

Personality Matrix

<table>
<thead>
<tr>
<th>Product term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>1 1 -</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>B C</td>
<td>- 0 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>A C</td>
<td>1 - 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>B C</td>
<td>- 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>A</td>
<td>1 - -</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

Input Side:
1 = asserted in term
0 = negated in term
- = does not participate

Output Side:
1 = term connected to output
0 = no connection to output

Example Continued

All possible connections are available before programming

Figure 3.18a. PAL Before Programming
Example Continued

Unwanted connections are "blown"

Note: some array structures work by making connections rather than breaking them

Alternative representation for high fan-in structures

Short-hand notation so we don't have to draw all the wires!

Notation for implementing

\[ F_0 = A \overline{B} + A' \overline{B}' \]
\[ F_1 = C \overline{D} + C' D \]
Design 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 xor B xor C

F6 = A xnor B xnor C

What is difference between Programmable Array Logic (PAL) and Programmable Logic Array (PLA)?

PAL concept — implemented by Monolithic Memories
constrained topology of the OR Array

A given column of the OR array has access to only a subset of the possible product terms

PLA concept — generalized topologies in AND and OR planes
Design Example: BCD to Gray Code Converter

\[
W = A + B \cdot D + B \cdot C \\
X = B \cdot C' \\
Y = B + C \\
Z = A' \cdot B' \cdot C' \cdot D + B \cdot C \cdot D + A \cdot D' + B' \cdot C \cdot D'
\]

Programmed PAL:

4 product terms per each OR gate
**Code Converter Discrete Gate Implementation**

1: 7404 hex inverters
2: 7400 quad 2-input NAND
3: 7410 tri 3-input NAND
4: 7420 dual 4-input NAND

4 SSI Packages vs. 1 PLA/PAL Package!

**Another Example: Magnitude Comparator**

K-map for EQ
K-map for NE
K-map for LT
K-map for GT
**Non-Gate Logic**

*Introduction*

AND-OR-Invert
PAL/PLA

Generalized Building Blocks
Beyond Simple Gates

**Kinds of "Non-gate logic":**
- switching circuits built from CMOS transmission gates
- multiplexer/selecter functions
- decoders
- tri-state and open collector gates
- read-only memories

---

**Selection Function/Demultiplexer Function with Transmission Gates**

**Selector:**
- Choose I₀ if S = 0
- Choose I₁ if S = 1

**Demultiplexer:**
- I to Z₀ if S = 0
- I to Z₁ if S = 1
Well-formed Switching Networks

Problem with the Demux implementation:
multiple outputs, but only one connected to the input!

The fix: additional logic to drive every output to a known value

Never allow outputs to “float”

Multiplexers/Selectors

Multi-point connections

Multiple input sources

Multiple output destinations
**General Concept**

$2^n$ data inputs, n control inputs, 1 output

used to connect $2^n$ points to a single point

control signal pattern form binary index of input connected to output

\[ Z = A' I_0 + A I_1 \]

Functional form

Logical form

**Two alternative forms for a 2:1 Mux Truth Table**

\[ Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3 \]

\[ Z = A' B' C' I_0 + A' B' C I_1 + A' B C' I_2 + A' B C I_3 + A B' C' I_4 + A B' C I_5 + A B C' I_6 + A B C I_7 \]

In general, \( Z = \sum_{k=0}^{2^n-1} m_k l_k \)

in minterm shorthand form for a $2^n$ :1 Mux
Alternative Implementations

Gate Level Implementation of 4:1 Mux

 Transmission Gate Implementation of 4:1 Mux

thirty six transistors twenty transistors

Large multiplexers can be implemented by cascaded smaller ones

Control signals B and C simultaneously choose one of $I_0$-$I_3$ and $I_4$-$I_7$

Control signal A chooses which of the upper or lower MUX's output to gate to Z

Alternative 8:1 Mux Implementation
**Multiplexers/selectors as a general purpose logic block**

$2^{n-1}:1$ multiplexer can implement any function of $n$ variables

$n-1$ control variables; remaining variable is a data input to the mux

**Example:**

$$F(A, B, C) = m_0 + m_2 + m_6 + m_7$$

$$= A' B' C' + A' B C' + A B C' + A B C$$

$$= A' B' (C') + A' B (C') + A B' (0) + A B (1)$$

"Lookup Table"

**Generalization**

n-1 Mux control variables

single Mux data variable

Four possible configurations of the truth table rows

Can be expressed as a function of In, 0, 1
Example:

G(A,B,C,D) can be implemented by an 8:1 MUX:

Decoders/Demultiplexers

**Decoder**: single data input, \( n \) control inputs, \( 2^n \) outputs

- control inputs (called **select** \( S \)) represent Binary index of output to which the input is connected
- data input usually called "**enable**" (\( G \))

1:2 Decoder:

\[
\begin{align*}
O_0 &= G \cdot S_0 \cdot S_1 \\
O_1 &= G \cdot S
\end{align*}
\]

2:4 Decoder:

\[
\begin{align*}
O_0 &= G \cdot S_0 \cdot S_1 \\
O_1 &= G \cdot S_0 \cdot S_1 \\
O_2 &= G \cdot S_0 \cdot S_1 \\
O_3 &= G \cdot S_0 \cdot S_1 \\
O_4 &= G \cdot S_0 \cdot S_1 \\
O_5 &= G \cdot S_0 \cdot S_1 \\
O_6 &= G \cdot S_0 \cdot S_1 \\
O_7 &= G \cdot S_0 \cdot S_1
\end{align*}
\]

3:8 Decoder:

\[
\begin{align*}
O_0 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_1 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_2 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_3 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_4 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_5 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_6 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_7 &= G \cdot S_0 \cdot S_1 \cdot S_2
\end{align*}
\]
Alternative Implementations

1:2 Decoder, Active High Enable

1:2 Decoder, Active Low Enable

2:4 Decoder, Active High Enable

2:4 Decoder, Active Low Enable

Switch Logic Implementations

Naive, Incorrect Implementation
All outputs not driven at all times

Correct 1:2 Decoder Implementation
Switch Implementation of 2:4 Decoder

Operation of 2:4 Decoder

S0 = 0, S1 = 0
one straight thru path
three diagonal paths

Decoder as a Logic Building Block

Example Function:

\[ F_1 = A' B C' D + A' B' C D + A B C D \]
\[ F_2 = A B C' D' + A B C \]
\[ F_3 = (A' + B' + C' + D') \]
Decoder/Demultiplexer

**Decoder as a Logic Building Block**

If active low enable, then use NAND gates!

Multiplexers/Decoders

**Alternative Implementations of 32:1 Mux**

Multiplexer Only  Multiplexer + Decoder
Multiplexers/Decoders

5:32 Decoder

Tri-State and Open-Collector

The Third State

Logic States: "0", "1"
Don't Care/Don't Know State: "X" (must be some value in real circuit!)

Third State: "Z" — high impedance — infinite resistance, no connection

Tri-state gates: output values are "0", "1", and "Z"
additional input: output enable (OE)

<table>
<thead>
<tr>
<th>A</th>
<th>OE</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When OE is high, this gate is a non-inverting "buffer"

When OE is low, it is as though the gate was disconnected from the output!

This allows more than one gate to be connected to the same output wire, as long as only one has its output enabled at the same time

Non-inverting buffer's timing waveform
## Tri-state and Open Collector

### Using tri-state gates to implement an economical multiplexer:

When `SelectInput` is asserted high
Input1 is connected to F

When `SelectInput` is driven low
Input0 is connected to F

This is essentially a 2:1 Mux

### Alternative Tri-state Fragment

Active low tri-state enables plus inverting tri-state buffers

Switch Level Implementation of tri-state gate
Tri-State and Open Collector

4:1 Multiplexer, Revisited

Decoder + 4 tri-state Gates

Open Collector

another way to connect multiple gates to the same output wire

gate only has the ability to pull its output low; it cannot actively drive the wire high

this is done by pulling the wire up to a logic 1 voltage through a resistor

OC NAND gates

Wired AND:

If A and B are "1", output is actively pulled low
If C and D are "1", output is actively pulled low
if one gate is low, the other high, then low wins
if both gates are "1", the output floats, pulled high by resistor

Hence, the two NAND functions are AND'd together!
**Tri-State and Open Collector**

*4:1 Multiplexer*

Decoder + 4 Open Collector Gates

---

**Read-Only Memories**

ROM: Two dimensional array of 1's and 0's

Row is called a "word"; index is called an "address"

Width of row is called *bit-width* or *wordsize*

Address is input, selected word is output

Internal Organization
Read-Only Memories

Example: Combination Logic Implementation

\[ F_0 = A' B' C + A B' C' + A B C \]
\[ F_1 = A' B' C + A' B C' + A B C \]
\[ F_2 = A' B' C' + A' B C + A B' C' \]
\[ F_3 = A' B C + A B' C' + A B C' \]

Address outputs

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F_0</th>
<th>F_1</th>
<th>F_2</th>
<th>F_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Read-Only Memories

Not unlike a PLA structure with a fully decoded AND array!

ROM vs. PLA:

ROM approach advantageous when
1. design time is short (no need to minimize output functions)
2. most input combinations are needed (e.g., code converters)
3. little sharing of product terms among output functions

ROM problem: size doubles for each additional input, can't use don't cares

PLA approach advantageous when
1. design tool like espresso is available
2. there are relatively few unique minterm combinations
3. many minterms are shared among the output functions

PAL problem: constrained fan-ins on OR planes
Read-Only Memories

2764 EPROM
8K x 8

16K x 16 Subsystem

Combinational Logic Word Problems

General Design Procedure

1. Understand the Problem
   what is the circuit supposed to do?
   write down inputs (data, control) and outputs
   draw block diagram or other picture

2. Formulate the Problem in terms of a truth table or other suitable design representation
   truth table or waveform diagram

3. Choose Implementation Target
   ROM, PAL, PLA, Mux, Decoder + OR, Discrete Gates

4. Follow Implementation Procedure
   K-maps, espresso, misII
Combinational Logic Word Problems

Process Line Control Problem

Statement of the Problem

Rods of varying length (+/-10%) travel on conveyor belt
Mechanical arm pushes rods within spec (+/-5%) to one side
Second arm pushes rods too long to other side
Rods too short stay on belt

3 light barriers (light source + photocell) as sensors
Design combinational logic to activate the arms

Understanding the Problem

Inputs are three sensors, outputs are two arm control signals
Assume sensor reads "1" when tripped, "0" otherwise
Call sensors A, B, C

Draw a picture!

Combinational Logic Word Problems

Process Control Problem

Spec

+ 5%

+ 10%

Spec

Spec

ROD Too Long

ROD Within Spec

ROD Too Short

Where to place the light sensors A, B, and C to distinguish among the three cases?

Assume that A detects the leading edge of the rod on the conveyor
### Process Control Problem

#### Specification

- Too Long
- Within Spec
- Too Short

#### A to B distance place apart at specification - 5%

#### A to C distance placed apart at specification +5%

---

#### Truth Table and Logic Implementation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>too short</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>in spec</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>too long</td>
</tr>
</tbody>
</table>

Truth table and logic implementation now straightforward

"too long" = A B C  
(all three sensors tripped)

"in spec" = A B C’  
(first two sensors tripped)
Combinational Logic Word Problems

**BCD to 7 Segment Display Controller**

**Understanding the problem:**
- Input is a 4 bit BCD digit
- Output is the control signals for the display

- 4 inputs: A, B, C, D
- 7 outputs: C0 — C6

**Block Diagram**

Formulate the problem in terms of a truth table

<table>
<thead>
<tr>
<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>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>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>1</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>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>1</td>
<td>0</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>
</tr>
<tr>
<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>
<td>0</td>
<td>0</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>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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<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>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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<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>1</td>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>1</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>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</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>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Choose implementation target:
- If ROM, we are done
- Don't cares imply PAL/PLA may be attractive

Follow implementation procedure:
- Hand reduced K-maps vs.
- espresso
Combinational Logic Word Problems

**BCD to 7 Segment Display Controller**

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

14 Unique Product Terms

**Combinational Logic Word Problems**

**BCD to 7 Segment Display Controller**

16H8PAL Can Implement the function
Combinational Logic Word Problems

14H8PAL
Cannot implement the function

Combinational Logic Word Problems

BCD to 7 Segment Display Controller

PLA Implementation
**Combinational Logic Word Problems**

**BCD to 7 Segment Display Controller**

**Multilevel Implementation**

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

- 52 literals
- 33 gates
- Ineffective use of don't cares

---

**Combinational Logic Word Problems**

**Logical Function Unit**

**Statement of the Problem:**

<table>
<thead>
<tr>
<th>C</th>
<th>C</th>
<th>C2</th>
<th>F</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 and</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A xor B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xnor B</td>
<td>logical xnor</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A B</td>
<td>logical and</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>always 0</td>
</tr>
</tbody>
</table>

3 control inputs: C0, C1, C2
2 data inputs: A, B
1 output: F
Combinational Logic Word Problems

**Logical Function Unit**

Formulate as a truth table

Choose implementation technology

- 5-variable K-map
- espresso
- multiplexor implementation

4 TTL packages:
- 4 x 2-input NAND
- 4 x 2-input NOR
- 2 x 2-input XOR
- 8:1 MUX

Follow implementation procedure

\[ F = C_2' A' B' + C_0' A B' + C_0' A' B + C_1' A B \]

5 gates, 5 inverters

Also four packages:
- 4 x 3-input NAND
- 1 x 4-input NAND

Alternative: 32 x 1-bit ROM
- single package
The Priority Mux connects a device output (OA, OB, OC, OD) to the output line (OUTPUT) based on priorities. Each device has a request line (RA, RB, RC, RD) that is asserted when the system output line is requested by that device. The Mux must return a signal (SA, SB, SC, SD) to each device indicating whether the request was accepted. These signals should be asserted only if a request has been received from the device and the device output has been connected to the system output.

The output of this Priority Encoder represent the binary code of the highest asserted input at any point in time. The Inactive output indicates when no input is asserted.
Chapter Review

• **Non-Simple Gate Logic Building Blocks:**
  - PALs/PLAs
  - Multiplexers/Selecters
  - Decoders
  - ROMs
  - Tri-state, Open Collector

• **Combinational Word Problems:**
  - Understand the Problem
  - Formulate in terms of a Truth Table
  - Choose implementation technology
  - Implement by following the design procedure