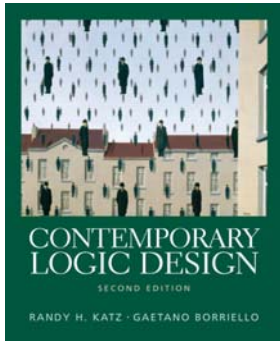


CSE 370 Spring 2006

Introduction to Digital Design

Lecture 11: PLAs and PALs



Last Lecture

- Mux/Demux

Today

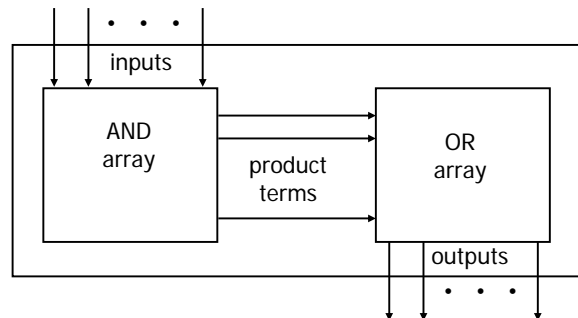
- PLAs and PALs

Administrivia

- HW 4 due Friday

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



Programming the wire connections

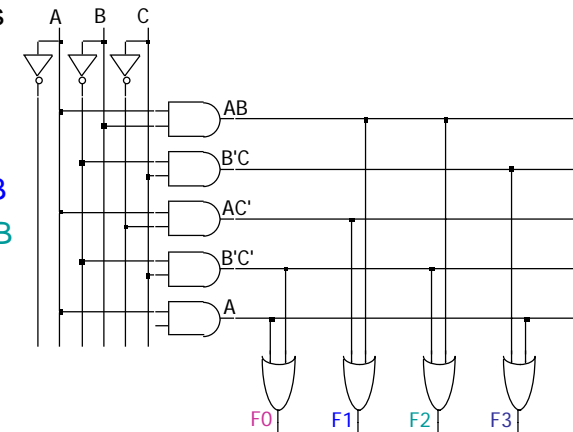
- Fuse: Comes connected; break unwanted connections
- Anti-fuse: Comes disconnected; make wanted connections

$$F0 = A + B'C'$$

$$F1 = AC' + AB$$

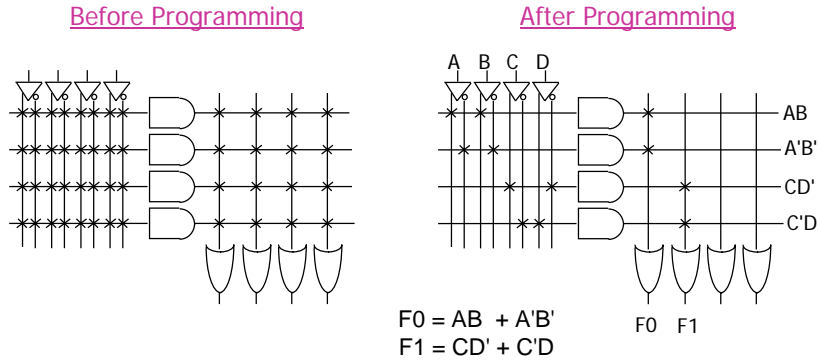
$$F2 = B'C' + AB$$

$$F3 = B'C + A$$



Short-hand notation

- Draw multiple wires as a single wire or bus
- x signifies a connection



Sharing product terms

- Example: $F0 = A + B'C'$
 $F1 = AC' + AB$
 $F2 = B'C' + AB$
 $F3 = B'C + A$

inputs
 1 = asserted in term
 0 = negated in term
 - = does not participate

- Personality matrix:

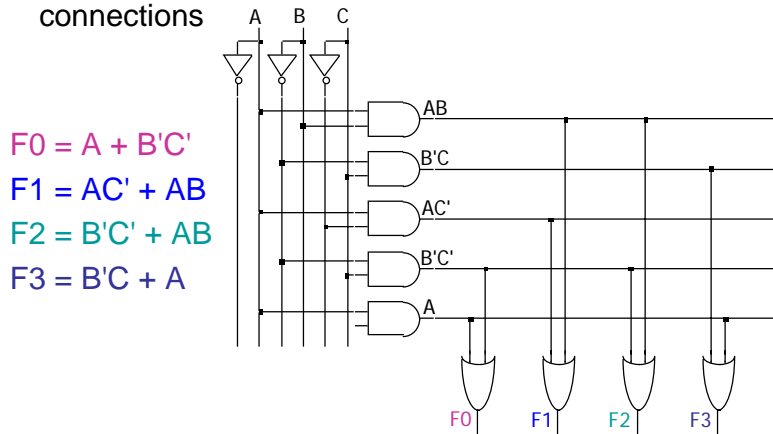
outputs
 1 = term connected to output
 0 = no connection to output

product term	inputs			outputs			
	A	B	C	F0	F1	F2	F3
AB	1	1	-	0	1	1	0
B'C	-	0	1	0	0	0	1
AC'	1	-	0	0	1	0	0
B'C'	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse terms

Programming the wire connections

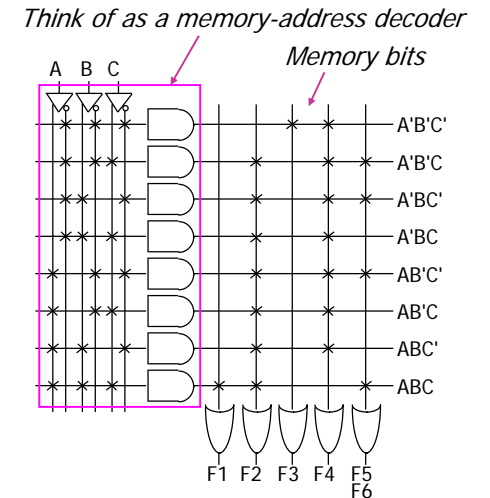
- Fuse: Comes connected; break unwanted connections
- Anti-fuse: Comes disconnected; make wanted connections



PLA example

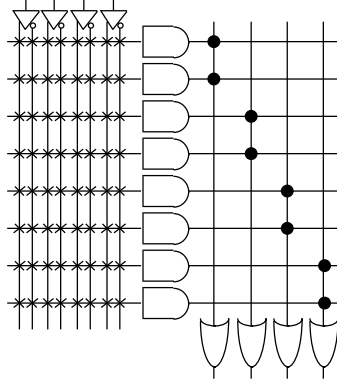
- $F1 = ABC$
- $F2 = A + B + C$
- $F3 = A' B' C'$
- $F4 = A' + B' + C'$
- $F5 = A \text{ xor } B \text{ xor } C$
- $F6 = A \text{ xnor } B \text{ xnor } C$

A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



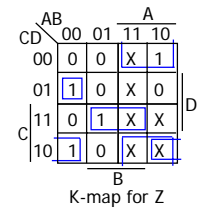
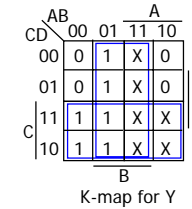
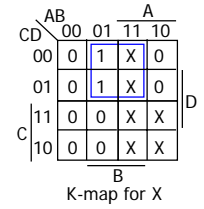
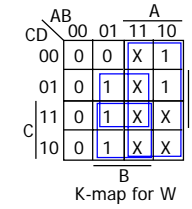
PLAs versus PALs

- We've been looking at PLAs
 - Fully programmable AND / OR arrays
 - Can share AND terms
- Programmable array logic (PAL)
 - Programmable AND array
 - OR array is prewired
 - No sharing ANDs
 - Cheaper and faster than PLAs



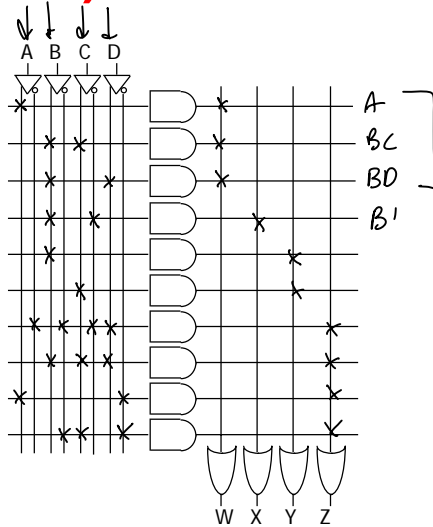
Example: BCD to Gray code converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



Example (con't): Wire a PLA

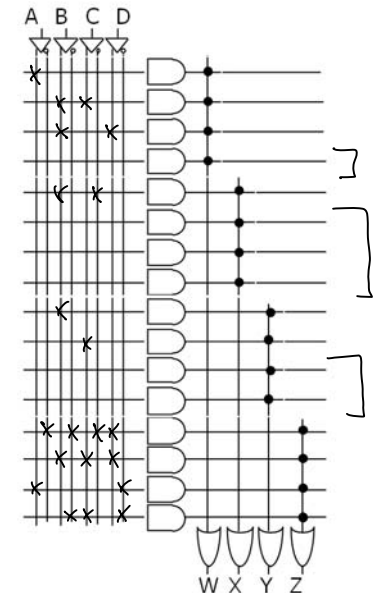
- Minimized functions:
- $W = A + BC + BD$
 - $X = BC'$
 - $Y = B + C$
 - $Z = A'B'C'D + BCD + AD' + B'CD'$



Example: Wire a PAL

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

What do we do with the unused AND gates?

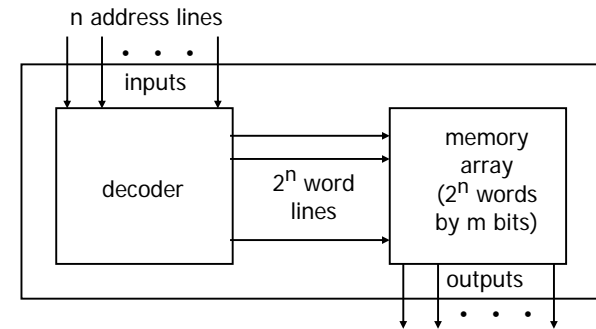


Compare implementations

- 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 poor candidate for PLAs/PALs
 - 10 of 16 possible inputs are decoded
 - No sharing among AND terms
- Better option?
 - Yes — a ROM

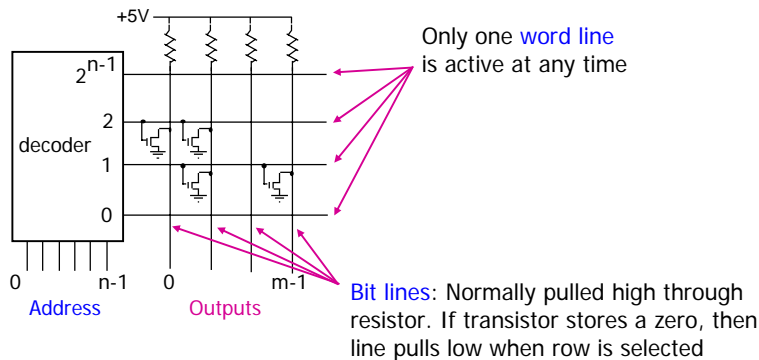
Read-only memories (ROMs)

- 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



ROM details

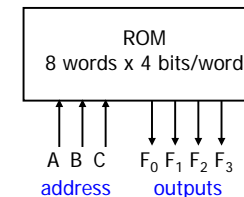
- Similar to a PLA but with a fully decoded AND array
- Completely flexible OR array (unlike a PAL)
- Extremely dense: One transistor per stored bit



Two-level combinational logic using a ROM

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

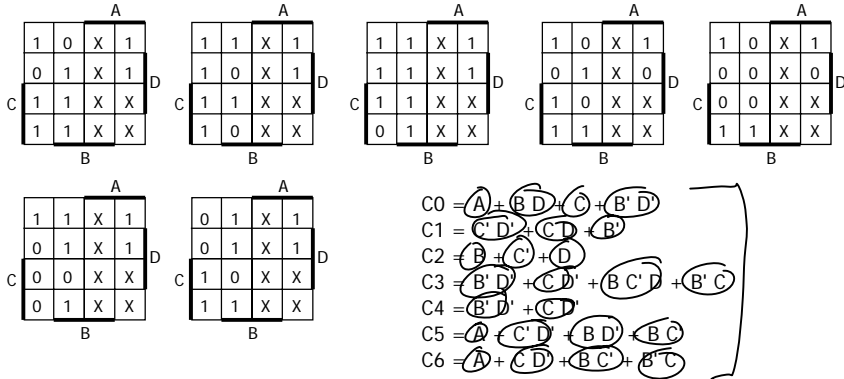
A	B	C	F ₀	F ₁	F ₂	F ₃
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0



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

Sum-of-products implementation

■ 15 unique product terms if we minimize individually

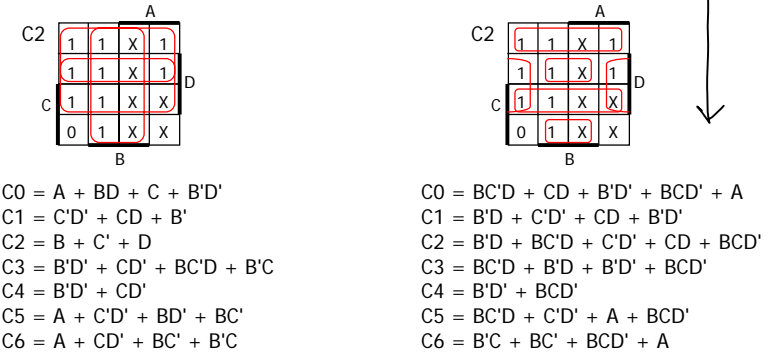


Better SOP implementation

■ Can do better than 15 product terms

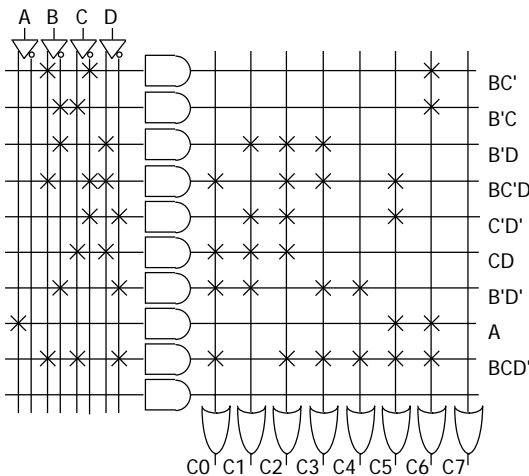
■ Share terms among outputs \Rightarrow only 9 unique product terms

■ Each term not necessarily minimized



PLA implementation

$C0 = BC'D + CD + B'D' + BCD' + A$
 $C1 = B'D + C'D' + CD + B'D'$
 $C2 = B'D + BC'D + C'D' + CD + BCD'$
 $C3 = BC'D + B'D + B'D' + BCD'$
 $C4 = B'D' + BCD'$
 $C5 = BC'D + C'D' + A + BCD'$
 $C6 = B'C + BC' + BCD' + A$



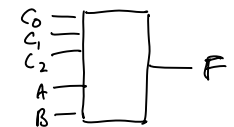
Example: Logical function unit

■ Multipurpose functional block

■ 3 control inputs (C) specify function

■ 2 data inputs (operands) A and B

■ 1 output (same bit-width as input operands)



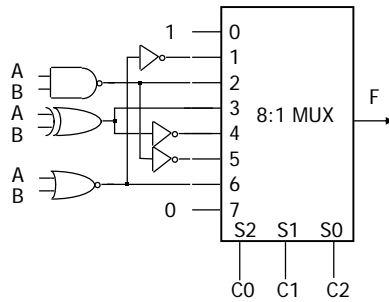
$C0$	$C1$	$C2$	Function	Comments
0	0	0	1	always 1
0	0	1	$A + B$	logical OR
0	1	0	$(A \cdot B)'$	logical NAND
0	1	1	$A \text{ xor } B$	logical xor
1	0	0	$A \text{ xnor } B$	logical xnor
1	0	1	$A \cdot B$	logical AND
1	1	0	$(A + B)'$	logical NOR
1	1	1	0	always 0

3 control inputs: $C0, C1, C2$
 2 data inputs: A, B
 1 output: F

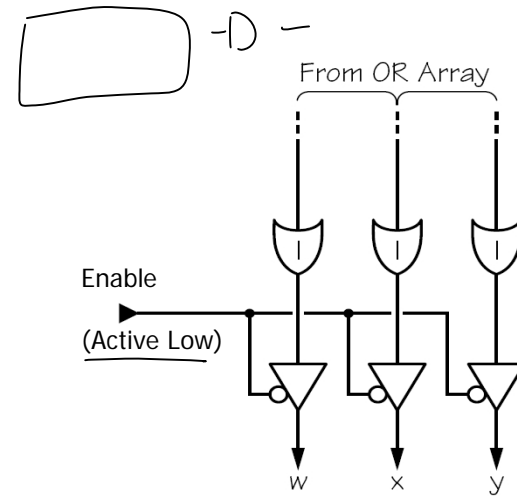
Formalize the problem and solve

C0	C1	C2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

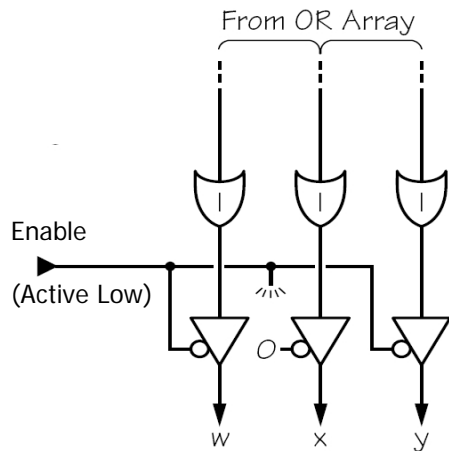
Implementation choice:
multiplexer with discrete gates



PAL Feature: Tri-stated outputs



PAL Feature: Individually Tri-stated outputs



Pal Feature: Feedback terms

