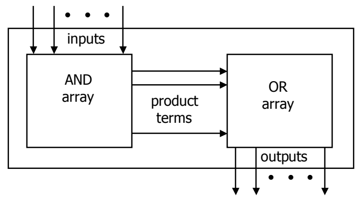
Overview

- Last lecture
  - "Switching-network" logic blocks
    - Multiplexers/selectors
    - Demultiplexers/decoders
  - Programmable logic devices (PLDs)
    - Regular structures for 2-level logic
- Today
  - PLDs
    - PLAs
    - PALs
  - ROMs
  - Tristates
  - Design examples

Programmable logic (PLAs & PALs)

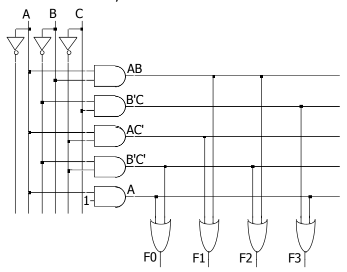
- Concept: Large array of uncommitted AND/OR gates
  - Actually NAND/NOR gates
  - You program the array by making or breaking connections
    - Programmable block for sum-of-products logic



Programming the wire connections

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

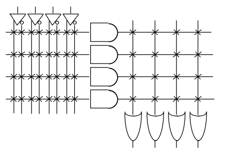
F0 = A + B'C'
F1 = AC' + AB
F2 = B'C' + AB
F3 = B'C + A



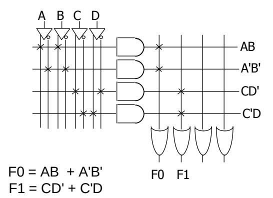
Short-hand notation

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

Before Programming



After Programming

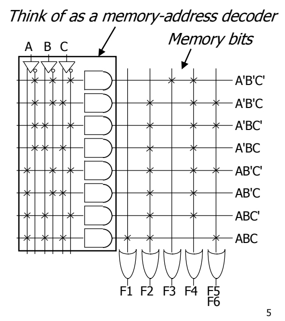


F0 = AB + A'B'
F1 = CD' + C'D

PLA example

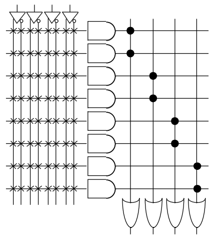
F1 = ABC
F2 = A + B + C
F3 = A' B' C'
F4 = A' + B' + C'
F5 = A xor B xor C
F6 = A xnor B xnor C

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



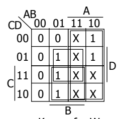
PLAs versus PALs

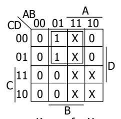
- We've been looking at PLAs
  - Fully programmable AND / OR arrays
    - Can share AND terms
- Programmable array logic (PAL)
  - Programmable AND array
  - OR array is prewired
    - No sharing ANDs
    - Cheaper and faster than PLAs

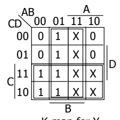


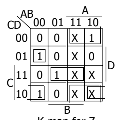
Example: BCD to Gray code converter

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


K-map for W

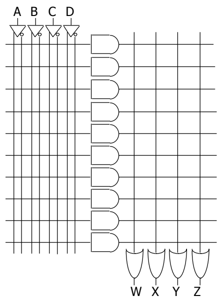

K-map for X


K-map for Y


K-map for Z

Example (con't): Wire a PLA

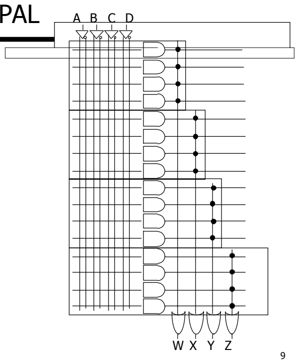
Minimized functions:
W = A + BC + BD
X = BC'
Y = B + C
Z = A'B'C'D + BCD
+ AD' + B'CD'



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?

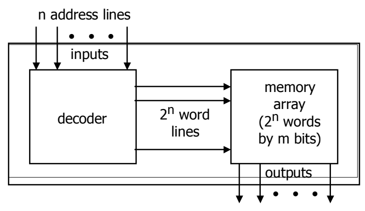


Compare implementations

- 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

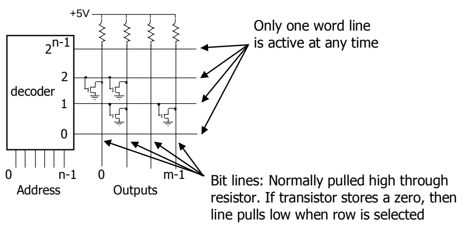
Read-only memories (ROMs)

- Two dimensional array of stored 1s and 0s
  - Input is an address ⇒ ROM decodes all possible input addresses
  - Stored row entry is called a "word"
  - ROM output is the decoded word



ROM details

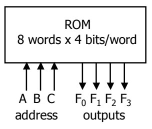
- Similar to a PLA but with a fully decoded AND array
- Completely flexible OR array (unlike a PAL)
- Extremely dense: One transistor per stored bit


+5V
2^{n-1}
2
1
0
decoder
0
n-1
Address
0
m-1
Outputs
Only one word line is active at any time
Bit lines: Normally pulled high through resistor. If transistor stores a zero, then line pulls low when row is selected

Two-level combinational logic using a ROM

- Use a ROM to directly store a truth table
  - No need to minimize logic
  - Example: F0 = A'B'C + AB'C' + AB'C
    F1 = A'B'C + A'BC' + ABC
    F2 = A'B'C' + A'B'C + AB'C'
    F3 = A'BC + AB'C' + ABC'

A	B	C	F0	F1	F2	F3
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

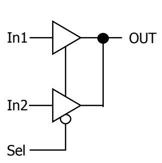
ROMs versus PLAs/PALs

- ROMs
  - Benefits
    - Quick to design, simple, dense
  - Limitations
    - Size doubles for each additional input
    - Can't exploit don't cares
- PLAs/PALs
  - Benefits
    - Logic minimization reduces size
  - Limitations
    - PAL OR-plane has hard-wired fan-in
- Another answer: Field programmable gate arrays
  - Learn about in 467

Loose end: Tristates

- Tristate buffers have a control input
  - Enabled: Buffer works normally
  - Disabled: Buffer output is disconnected

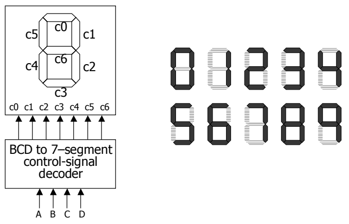
2:1 Tristate Mux



```
module muxtri (In1,In2,Sel,OUT);
  input  In1,In2,Sel;
  output OUT;
  tri    OUT;
  bufif1 (OUT,In1,Sel);
  bufif0 (OUT,In2,Sel);
endmodule
```

Example: BCD to 7-segment display controller

- The problem
  - Input is a 4-bit BCD digit (A, B, C, D)
  - Need signals to drive a display (7 outputs C0 – C6)



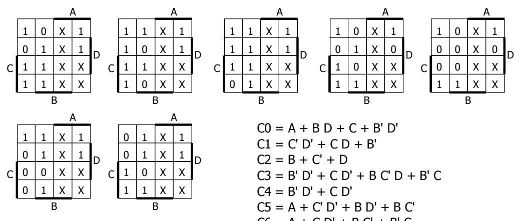
Formalize the problem

- Truth table
  - Many don't cares
- Choose implementation target
  - If ROM, we are done
  - Don't cares imply PAL/PLA may be good choice
- Implement design
  - Minimize the logic
  - Map into PAL/PLA

A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

Sum-of-products implementation

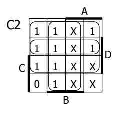
- 15 unique product terms if we minimize individually

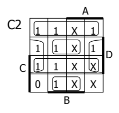


C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

Better SOP implementation

- Can do better than 15 product terms
  - Share terms among outputs ⇒ only 9 unique product terms
    - Each term not necessarily minimized



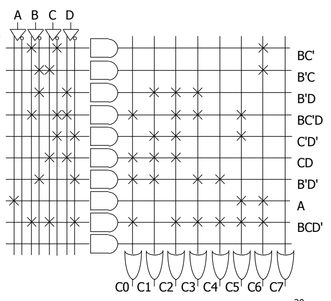


C0 = A + BD + C + B'D'
C1 = C'D' + CD + B'
C2 = B + C' + D
C3 = B'D' + CD' + BC'D + B'C
C4 = B'D' + CD'
C5 = A + C'D' + BD' + BC'
C6 = A + CD' + BC' + B'C

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

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)

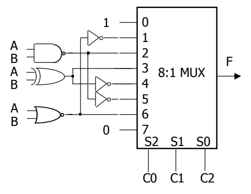
C0	C1	C2	Function	Comments
0	0	0	1	always 1
0	0	1	A + B	logical OR
0	1	0	(A • B)'	logical NAND
0	1	1	A xor B	logical xor
1	0	0	A xnor B	logical xnor
1	0	1	A • B	logical AND
1	1	0	(A + B)'	logical NOR
1	1	1	0	always 0

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	1
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


1
A
B
A
B
A
B
0
0
1
2
3
4
5
6
7
8:1 MUX
F
S2
S1
S0
C0
C1
C2