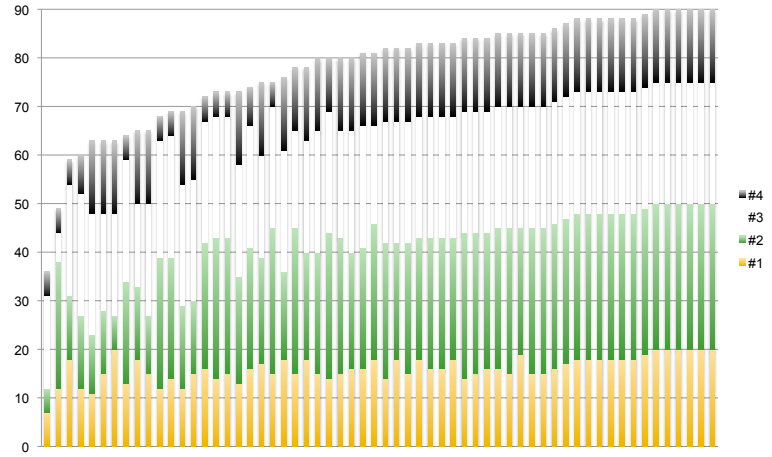
Exam 1 Results

- Average = 78; Median = 81; StDev = 11.1 (68% within 67 and 89)

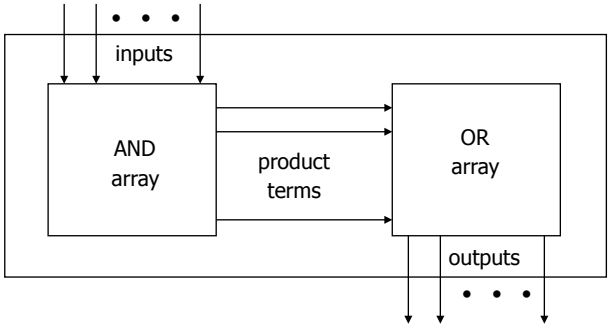


Implementation Technologies

- Standard gates (pretty much done)
  - gate packages
  - cell libraries
- Regular logic (we've been here)
  - multiplexers
  - decoders
- Two-level programmable logic (we are now here)
  - PALs, PLAs, PLDs
  - ROMs
  - FPGAs

Programmable logic arrays

- Pre-fabricated building block of many AND/OR gates
  - actually NOR or NAND
  - "personalized" by making/breaking connections among the gates
  - programmable array block diagram for sum of products form



Enabling concept

- Shared product terms among outputs

example:

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

personality matrix

product term	inputs A	B	C	outputs 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 of terms

input side:

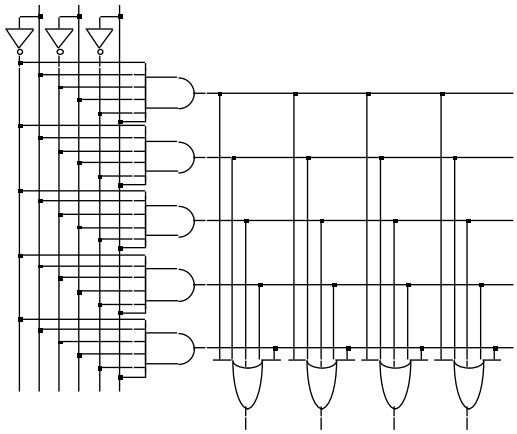
1 = uncomplemented in term
0 = complemented in term
– = does not participate

output side:

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

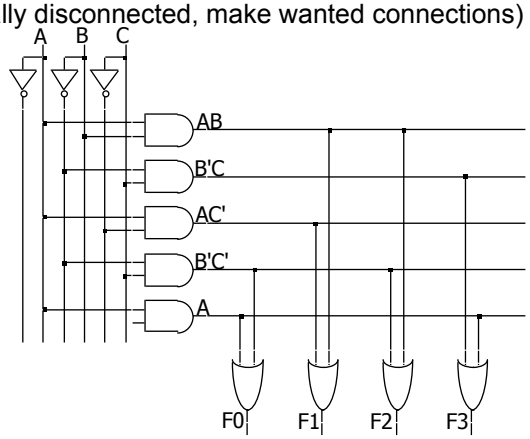
Before programming

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



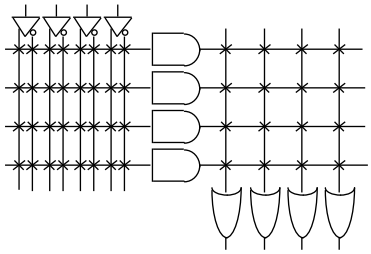
After programming

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

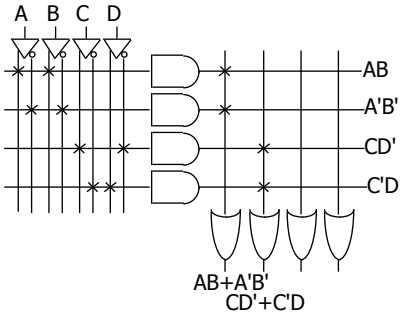


Alternate representation for high fan-in structures

- Short-hand notation so we don't have to draw all the wires
  - × signifies a connection is present and perpendicular signal is an input to gate



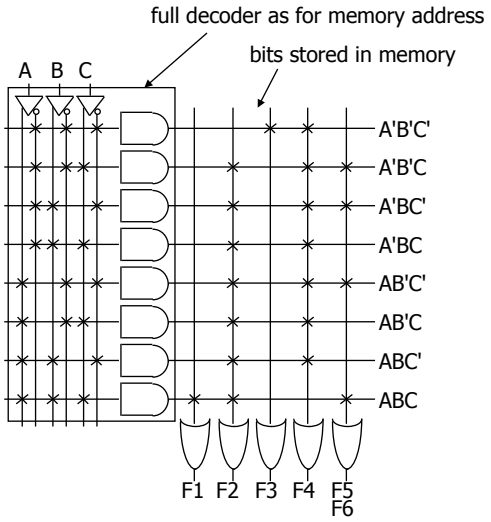
notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D



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 xor B xor C
  - F6 = A xnor B xnor C

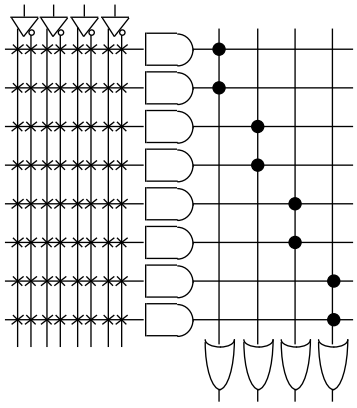
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



PALs and PLAs

- Programmable logic array (PLA)
  - what we've seen so far
  - unconstrained fully-general AND and OR arrays
- Programmable array logic (PAL)
  - constrained topology of the OR array
  - innovation by Monolithic Memories
  - faster and smaller OR plane

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



PALs and PLAs: design example

- BCD to Gray code converter

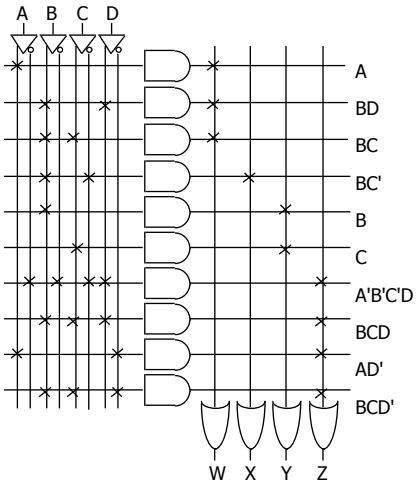
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

minimized functions:

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

PALs and PLAs: design example (cont'd)

- Code converter: programmed PLA



minimized functions:

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

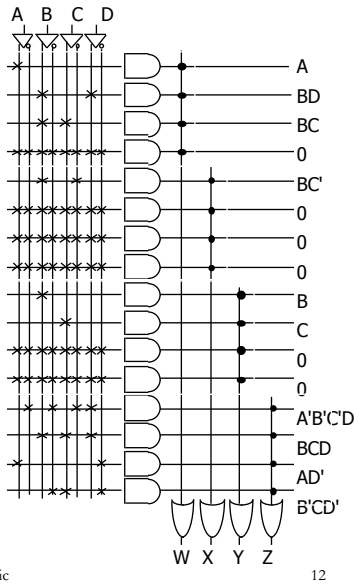
not a particularly good candidate for PAL/PLA implementation since no terms are shared among outputs

however, much more compact and regular implementation when compared with discrete AND and OR gates

PALs and PLAs: design example (cont'd)

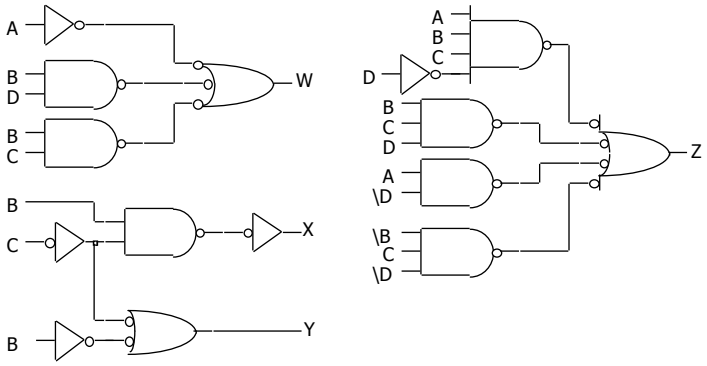
- Code converter: programmed PAL

4 product terms per each OR gate



PALs and PLAs: design example (cont'd)

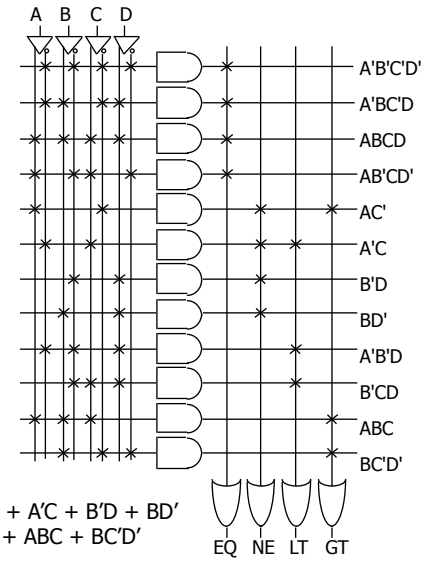
- Code converter: NAND gate implementation
  - loss or regularity, harder to understand
  - harder to make changes



PALs and PLAs: another design example

- Magnitude comparator

A	B	C	D	EQ	NE	LT	GT
0	0	0	0	1	0	0	0
0	0	0	1	0	1	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	0	1
1	0	1	0	1	0	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	0	0	1	0	1
1	1	1	1	1	0	0	0



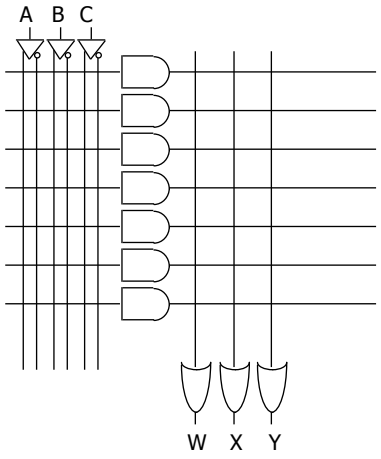
minimized functions:

EQ = A'B'C'D' + A'BC'D + ABCD + AB'CD' NE = AC' + A'C + B'D + BD'

LT = A'C + A'B'D + B'CD GT = AC' + ABC + BC'D'

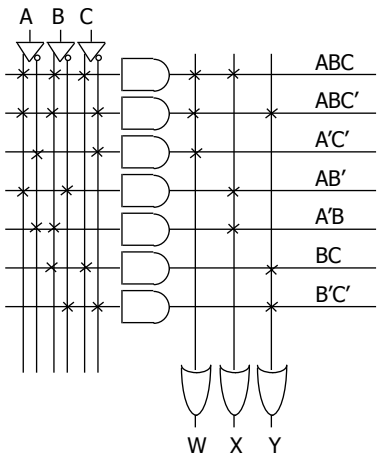
Activity

- Map the following functions to the PLA below:
  - W = AB + A'C' + BC'
  - X = ABC + AB' + A'B
  - Y = ABC' + BC + B'C'



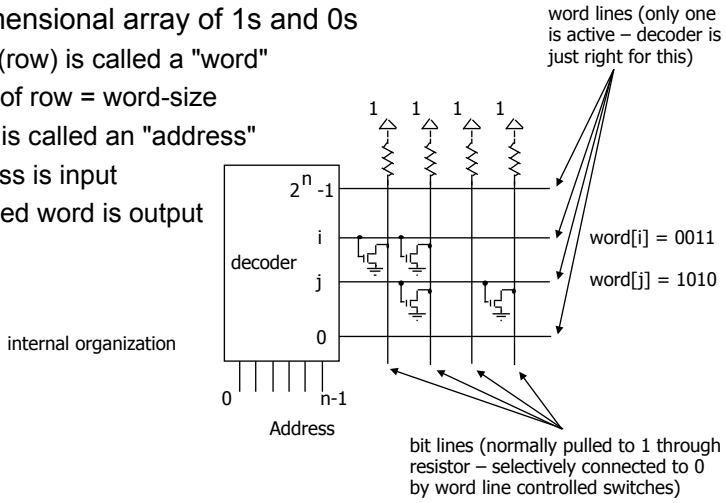
Activity (cont'd)

- 9 terms won't fit in a 7 term PLA
  - can apply concensus theorem to W to simplify to: W = AB + A'C'
- 8 terms wont' fit in a 7 term PLA
  - observe that AB = ABC + ABC'
  - can rewrite W to reuse terms: W = ABC + ABC' + A'C'
- Now it fits
  - W = ABC + ABC' + A'C'
  - X = ABC + AB' + A'B
  - Y = ABC' + BC + B'C'
- This is called technology mapping
  - manipulating logic functions so that they can use available resources



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

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

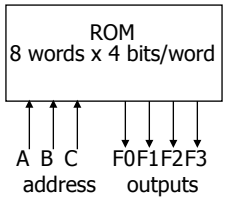
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

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

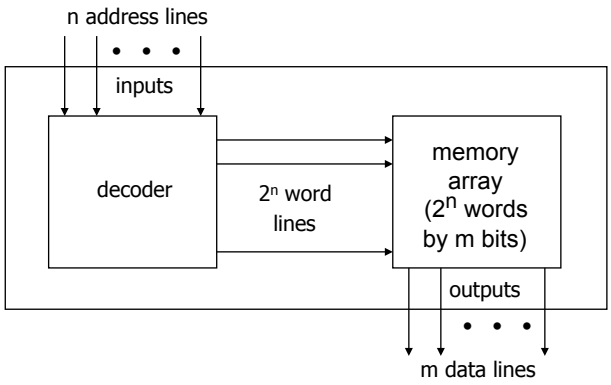
truth table



block diagram

ROM structure

- Similar to a PLA structure but with a fully decoded AND array
  - completely flexible OR array (unlike PAL)



ROM vs. PLA

- ROM approach advantageous when
  - 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
- ROM problems
  - size doubles for each additional input
  - can't exploit don't cares
- PLA approach advantageous when
  - design tools are available for multi-output minimization
  - there are relatively few unique minterm combinations
  - many minterms are shared among the output functions
- PAL problems
  - constrained fan-ins on OR plane

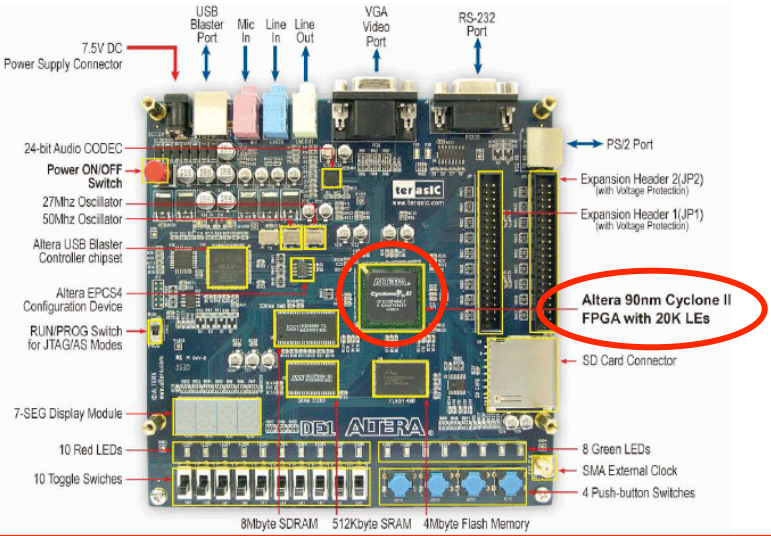
Regular logic structures for two-level logic

- ROM – full AND plane, general OR plane
  - cheap (high-volume component)
  - can implement any function of n inputs
  - medium speed
- PAL – programmable AND plane, fixed OR plane
  - intermediate cost
  - can implement functions limited by number of terms
  - high speed (only one programmable plane that is much smaller than ROM's decoder)
- PLA – programmable AND and OR planes
  - most expensive (most complex in design, need more sophisticated tools)
  - can implement any function up to a product term limit
  - slow (two programmable planes)

Regular logic structures for multi-level logic

- Difficult to devise a regular structure for arbitrary connections between a large set of different types of gates
  - efficiency/speed concerns for such a structure
  - next we'll learn about field programmable gate arrays (FPGAs) that are just such programmable multi-level structures
    - programmable multiplexers for wiring
    - lookup tables for logic functions (programming fills in the table)
    - multi-purpose cells (utilization is the big issue)
    - much more about these in CSE467
- Alternative to FPGAs: use multiple levels of PALs/PLAs/ROMs
  - output intermediate result
  - make it an input to be used in further logic
  - no longer practical approach given prevalence of FPGAs

FPGAs in CSE370



http://www.altera.com/products/devices/cyclone2/overview/cy2-overview.html

Cyclone II architecture

- Logic array blocks (LABs)
  - 4-input lookup tables
  - MUXes for which you specify inputs (function)
- Routing rows and cols to interconnect LABs
  - also composed of MUXes
  - select settings determine wires between LABs and I/O
- Many more parts
  - more later
- You will use synthesis tool (compiler) to determine programming from Verilog