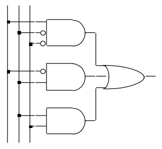
Combinational Logic Implementation

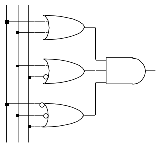
- Two-level logic
  - Implementations of two-level logic
  - NAND/NOR
- Multi-level logic
  - Factored forms
  - And-or-invert gates
- Time behavior
  - Gate delays
  - Hazards
- Regular logic
  - Multiplexers
  - Decoders
  - PAL/PLAs
  - ROMs

Implementations of Two-level Logic

- Sum-of-products
  - AND gates to form product terms (minterms)
  - OR gate to form sum

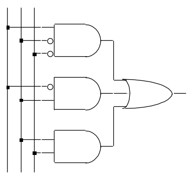
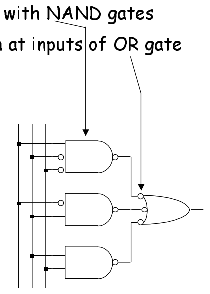


- Product-of-sums
  - OR gates to form sum terms (maxterms)
  - AND gates to form product



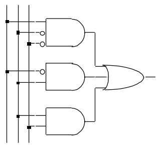
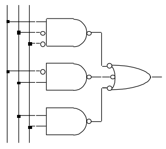
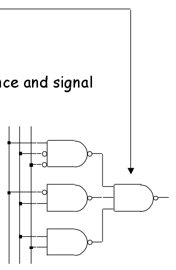
Two-level Logic using NAND Gates

- Replace minterm AND gates with NAND gates
- Place compensating inversion at inputs of OR gate

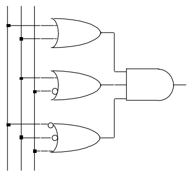
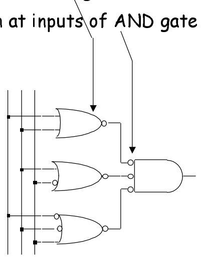
Two-level Logic using NAND Gates (cont'd)

- OR gate with inverted inputs is a NAND gate
  - de Morgan's: A' + B' = (A • B)'
- Two-level NAND-NAND network
  - Inverted inputs are not counted
  - In a typical circuit, inversion is done once and signal distributed

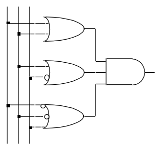
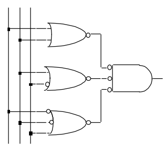
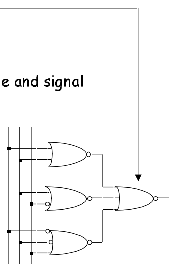
Two-level Logic using NOR Gates

- Replace maxterm OR gates with NOR gates
- Place compensating inversion at inputs of AND gate

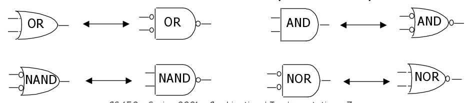
Two-level Logic using NOR Gates (cont'd)

- AND gate with inverted inputs is a NOR gate
  - de Morgan's: A' • B' = (A + B)'
- Two-level NOR-NOR network
  - Inverted inputs are not counted
  - In a typical circuit, inversion is done once and signal distributed

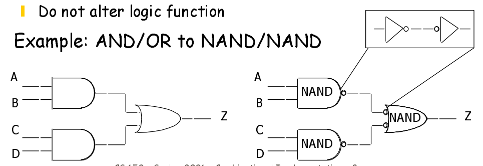
Two-level Logic using NAND and NOR Gates

- NAND-NAND and NOR-NOR networks
  - de Morgan's law: $(A + B)' = A' \cdot B'$
    $(A \cdot B)' = A' + B'$
  - written differently: $A + B = (A' \cdot B')'$
    $(A \cdot B) = (A' + B')'$
- In other words --
  - OR is the same as NAND with complemented inputs
  - AND is the same as NOR with complemented inputs
  - NAND is the same as OR with complemented inputs
  - NOR is the same as AND with complemented inputs



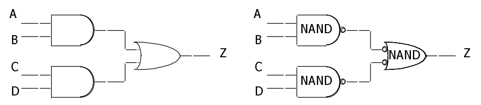
Conversion Between Forms

- Convert from networks of ANDs and ORs to networks of NANDs and NORs
  - Introduce appropriate inversions ("bubbles")
- Each introduced "bubble" must be matched by a corresponding "bubble"
  - Conservation of inversions
  - Do not alter logic function
- Example: AND/OR to NAND/NAND



Conversion Between Forms (cont'd)

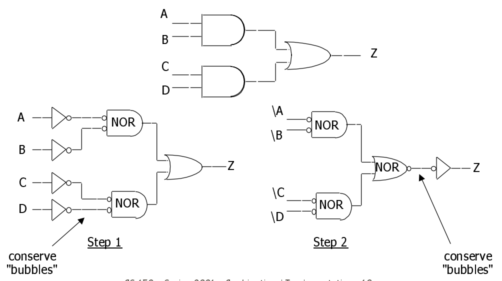
- Example: verify equivalence of two forms



$$\begin{aligned} Z &= [\,(A \cdot B)' \cdot (C \cdot D)'\,]' \\ &= [\,(A' + B') \cdot (C' + D')\,]' \\ &= [\,(A' + B')' + (C' + D')'\,] \\ &= (A \cdot B) + (C \cdot D) \checkmark \end{aligned}$$

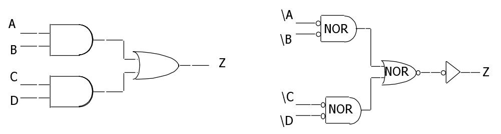
Conversion Between Forms (cont'd)

- Example: map AND/OR network to NOR/NOR network



Conversion Between Forms (cont'd)

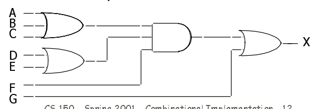
- Example: verify equivalence of two forms



$$\begin{aligned} Z &= \{\,[\,(A' + B')' + (C' + D')'\,]'\,\}' \\ &= \{\,(A' + B') \cdot (C' + D')\,\}' \\ &= (A' + B')' + (C' + D')' \\ &= (A \cdot B) + (C \cdot D) \checkmark \end{aligned}$$

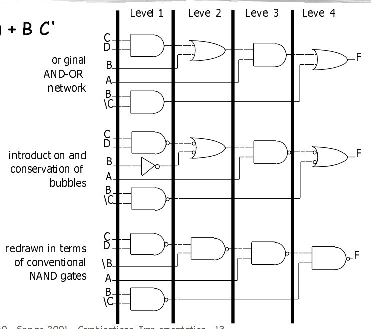
Multi-level Logic

- x = A D F + A E F + B D F + B E F + C D F + C E F + G
  - Reduced sum-of-products form – already simplified
  - 6 x 3-input AND gates + 1 x 7-input OR gate (may not exist!)
  - 25 wires (19 literals plus 6 internal wires)
- x = (A + B + C) (D + E) F + G
  - Factored form – not written as two-level S-o-P
  - 1 x 3-input OR gate, 2 x 2-input OR gates, 1 x 3-input AND gate
  - 10 wires (7 literals plus 3 internal wires)



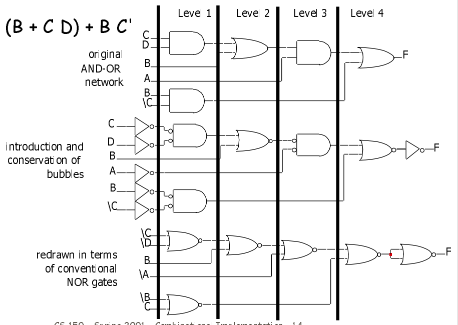
Conversion of Multi-level Logic to NAND Gates

- F = A (B + C D) + B C'


Level 1
Level 2
Level 3
Level 4
original AND-OR network
introduction and conservation of bubbles
redrawn in terms of conventional NAND gates
C
D
B
A
B
\C
\B
F

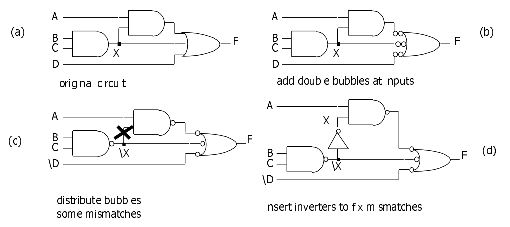
Conversion of Multi-level Logic to NORs

- F = A (B + C D) + B C'


Level 1
Level 2
Level 3
Level 4
original AND-OR network
introduction and conservation of bubbles
redrawn in terms of conventional NOR gates
C
D
B
A
\C
\D
\A
\B
F

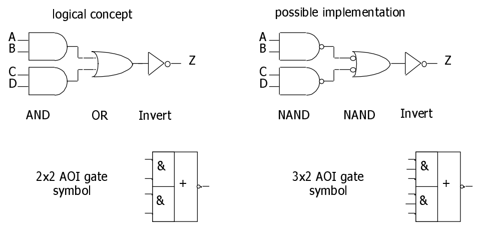
Conversion Between Forms

- Example


(a)
A
B
C
D
X
F
original circuit
(b)
add double bubbles at inputs
(c)
\X
\D
distribute bubbles some mismatches
(d)
insert inverters to fix mismatches

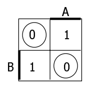
AND-OR-Invert Gates

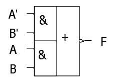
- AOI function: three stages of logic—AND, OR, Invert
  - Multiple gates "packaged" as a single circuit block



Conversion to AOI Forms

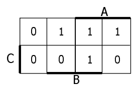
- General procedure to place in AOI form
  - Compute complement of the function in sum-of-products form
  - By grouping the 0s in the Karnaugh map
- Example: XOR implementation--A xor B = A' B + A B'
  - AOI form: F = (A' B' + A B)'



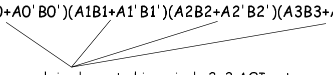


Examples of using AOI gates

- Example:
  - F = B C' + A C' + A B
  - F' = A' B' + A' C + B' C
  - Implemented by 2-input 3-stack AOI gate



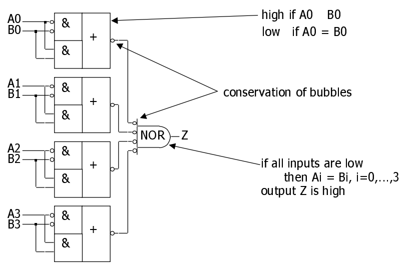
  - F = (A + B) (A + C') (B + C')
  - F' = (B' + C) (A' + C) (A' + B')
  - Implemented by 2-input 3-stack OAI gate
- Example: 4-bit equality function
  - Z = (A0B0+A0'B0')(A1B1+A1'B1')(A2B2+A2'B2')(A3B3+A3'B3')



each implemented in a single 2x2 AOI gate

Examples of Using AOI Gates (cont'd)

- Example: AOI implementation of 4-bit equality function



Summary for Multi-level Logic

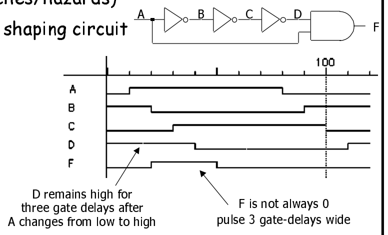
- Advantages
  - Circuits may be smaller
  - Gates have smaller fan-in
  - Circuits may be faster
- Disadvantages
  - More difficult to design
  - Tools for optimization are not as good as for two-level
  - Analysis is more complex

Time Behavior of Combinational Networks

- Waveforms
  - Visualization of values carried on signal wires over time
  - Useful in explaining sequences of events (changes in value)
- Simulation tools are used to create these waveforms
  - Input to the simulator includes gates and their connections
  - Input stimulus, that is, input signal waveforms
- Some terms
  - Gate delay—time for change at input to cause change at output
    - Min delay–typical/nominal delay–max delay
    - Careful designers design for the worst case
  - Rise time—time for output to transition from low to high voltage
  - Fall time—time for output to transition from high to low voltage
  - Pulse width—time an output stays high or low between changes

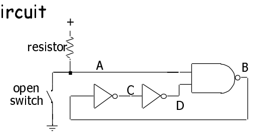
Momentary Changes in Outputs

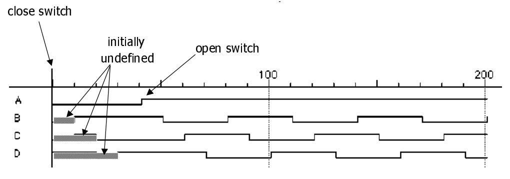
- Can be useful—pulse shaping circuits
- Can be a problem—incorrect circuit operation (glitches/hazards)
- Example: pulse shaping circuit
  - A' • A = 0
  - delays matter in function


A
B
C
D
F
100
D remains high for three gate delays after A changes from low to high
F is not always 0 pulse 3 gate-delays wide

Oscillatory Behavior

- Another pulse shaping circuit



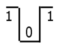


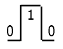
Hazards/Glitches

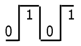
- Hazards/glitches: unwanted switching at the outputs
  - Occur when different paths through circuit have different propagation delays
    - As in pulse shaping circuits we just analyzed
  - Dangerous if logic causes an action while output is unstable
    - May need to guarantee absence of glitches
- Usual solutions
  - 1) Wait until signals are stable (by using a clock): preferable (easiest to design when there is a clock – *synchronous* design)
  - 2) Design hazard-free circuits: sometimes necessary (clock not used – *asynchronous* design)

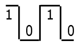
Types of Hazards

- Static 1-hazard
  - Input change causes output to go from 1 to 0 to 1
- Static 0-hazard
  - INput change causes output to go from 0 to 1 to 0
- Dynamic hazards
  - Input change causes a double change from 0 to 1 to 0 to 1 OR from 1 to 0 to 1 to 0



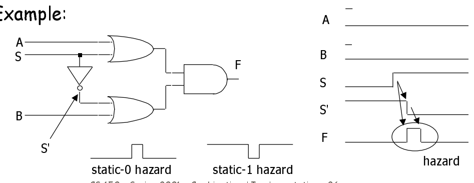






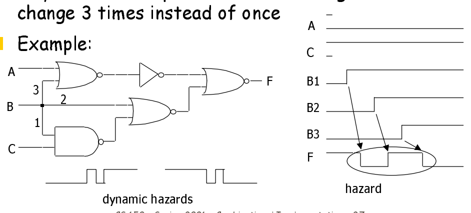
Static Hazards

- Due to a literal and its complement momentarily taking on the same value
  - Thru different paths with different delays and reconverging
- May cause an output that should have stayed at the same value to momentarily take on the wrong value
- Example:



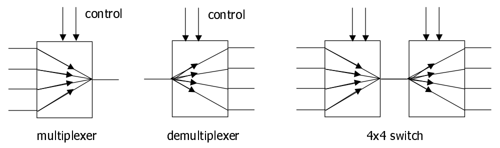
Dynamic Hazards

- Due to the same versions of a literal taking on opposite values
  - Thru different paths with different delays and reconverging
- May cause an output that was to change value to change 3 times instead of once
- Example:



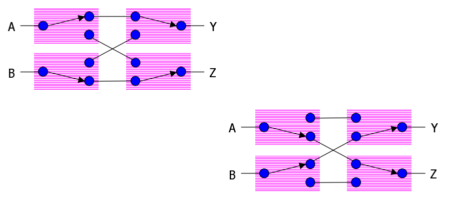
Making Connections

- Direct point-to-point connections between gates
  - Wires we've seen so far
- Route one of many inputs to a single output --- *multiplexer*
- Route a single input to one of many outputs --- *demultiplexer*



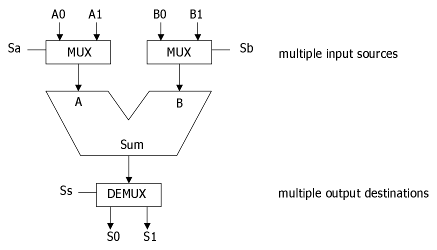
Mux and Demux

- Switch implementation of multiplexers and demultiplexers
  - Can be composed to make arbitrary size switching networks
  - Used to implement multiple-source/multiple-destination interconnections



Mux and Demux (cont'd)

- Uses of multiplexers/demultiplexers in multi-point connections



Multiplexers/Selectors

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

$Z = A' I_0 + A I_1$

A	Z
0	I_0
1	I_1

functional form

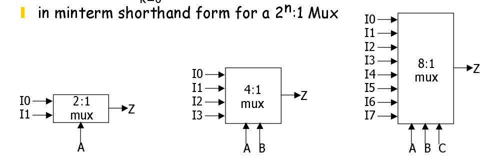
logical form

I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

two alternative forms for a 2:1 Mux truth table

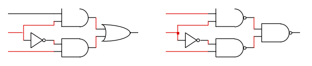
Multiplexers/Selectors (cont'd)

- 2:1 mux: $Z = A' I0 + A I1$
- 4:1 mux: $Z = A' B' I0 + A' B I1 + A B' I2 + A B I3$
- 8:1 mux: $Z = A'B'C'I0 + A'B'CI1 + A'BC'I2 + A'BCI3 + AB'C'I4 + AB'CI5 + ABC'I6 + ABCI7$
- In general, $Z = {}_{k=0}^{2^n-1} (m_k I_k)$
  - in minterm shorthand form for a 2^n:1 Mux

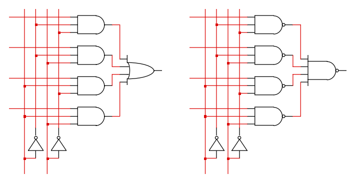


Gate Level Implementation of Muxes

- 2:1 mux

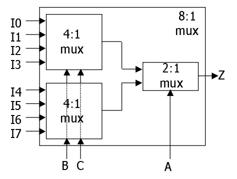


- 4:1 mux



Cascading Multiplexers

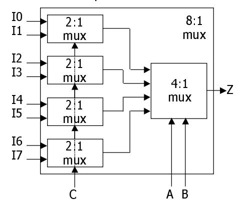
- Large multiplexers implemented by cascading smaller ones



control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

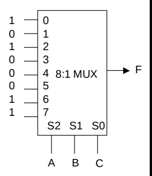
control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation



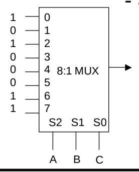
Multiplexers as General-purpose Logic

- 2^n:1 multiplexer implements any function of n variables
  - With the variables used as control inputs and
  - Data inputs tied to 0 or 1
  - In essence, *a lookup table*
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



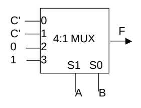
Multiplexers as General-purpose Logic (cont'd)

- 2^{n-1}:1 mux can implement any function of n variables
  - With n-1 variables used as control inputs and
  - Data inputs tied to the last variable or its complement
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



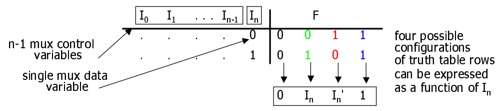
F

A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	

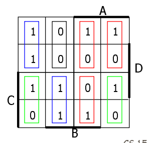


Multiplexers as General-purpose Logic (cont'd)

- Generalization

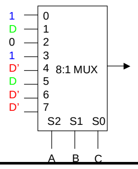

I_0 I_1 . . . I_{n-1} I_n
F
n-1 mux control variables
single mux data variable
0 0 1 1
0 1 0 1
0 I_n I_n' 1
four possible configurations of truth table rows can be expressed as a function of I_n

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



choose A,B,C as control variables

multiplexer implementation



Demultiplexers/Decoders

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

1:2 Decoder:

O0 = G	S'	
O1 = G	S	

2:4 Decoder:

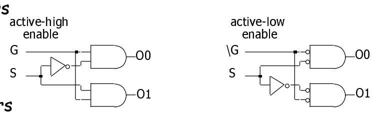
O0 = G	S1'	S0'
O1 = G	S1'	S0
O2 = G	S1	S0'
O3 = G	S1	S0

3:8 Decoder:

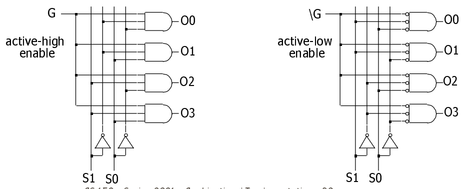
O0 = G	S2'	S1'	S0'
O1 = G	S2'	S1'	S0
O2 = G	S2'	S1	S0'
O3 = G	S2'	S1	S0
O4 = G	S2	S1'	S0'
O5 = G	S2	S1'	S0
O6 = G	S2	S1	S0'
O7 = G	S2	S1	S0

Gate Level iImplementation of Demultiplexers

- 1:2 Decoders

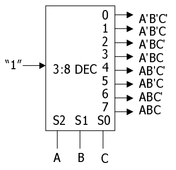


- 2:4 Decoders



Demultiplexers as General-purpose Logic

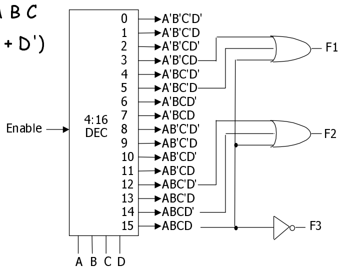
- n:2^n decoder implements any function of n variables
  - With the variables used as control inputs
  - Enable inputs tied to 1 and
  - Appropriate minterms summed to form the function



demultiplexer generates appropriate minterm based on control signals (it "decodes" control signals)

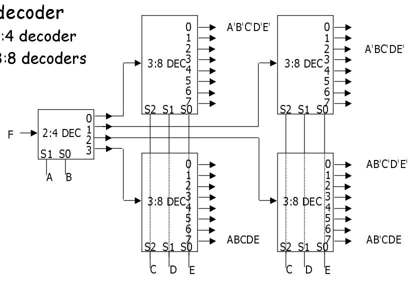
Demultiplexers as General-purpose Logic (cont'd)

- F1 = A' B C' D + A' B' C D + A B C D
- F2 = A B C' D' + A B C
- F3 = (A' + B' + C' + D')



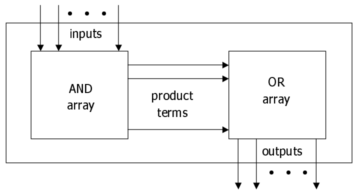
Cascading Decoders

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



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:
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

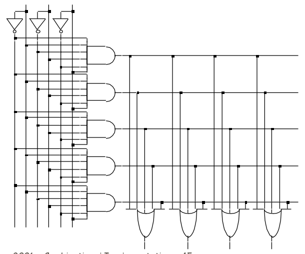
reuse of terms

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

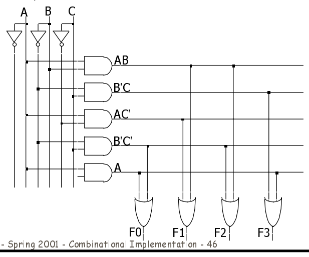
Before Programming

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



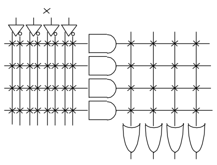
After Programming

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

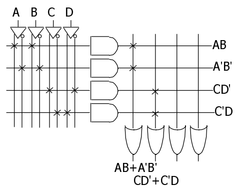


Alternate Representation for High Fan-in Structures

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



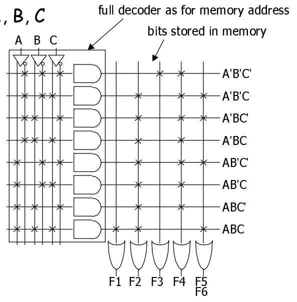
notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D



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

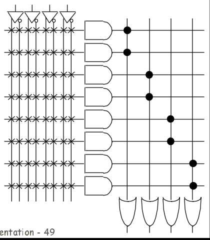
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



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



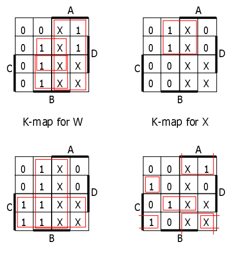
PALs and PLAs: Design Example

- 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	–	–	–	–	–	–

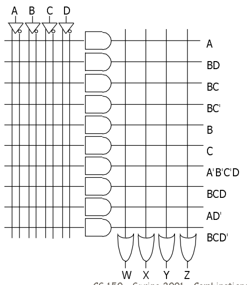
minimized functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A'B'C'D + B C D + A D' + B' C D'



PALs and PLAs: Design Example (cont'd)

- Code converter: programmed PLA



minimized functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A'B'C'D + B C D + A D' + B' C D'

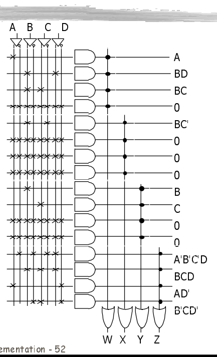
not a particularly good candidate for PAL/PLA implementation since no terms are shared among outputs

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

PALs and PLAs: Design Example (cont'd)

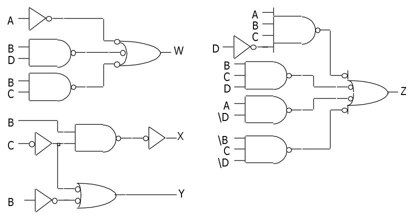
- Code converter: programmed PAL

4 product terms per each OR gate



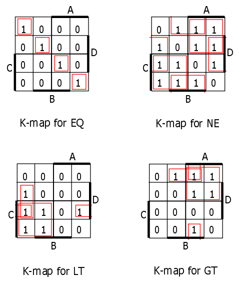
PALs and PLAs: Design Example (cont'd)

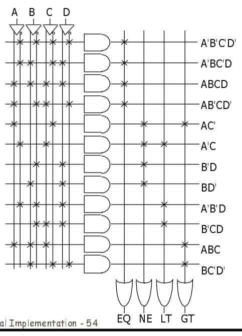
- Code converter: NAND gate implementation
  - Loss of regularity, harder to understand
  - Harder to make changes



PALs and PLAs: Another Design Example

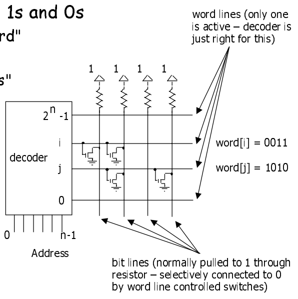
- Magnitude comparator





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



word lines (only one is active – decoder is just right for this)
1
1
1
1
2^n -1
i
decoder
j
0
word[i] = 0011
word[j] = 1010
0
n-1
Address
bit lines (normally pulled to 1 through resistor – selectively connected to 0 by word line controlled switches)

internal organization

ROMs and Combinational Logic

- Combinational logic implementation (two-level canonical form) using a ROM

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

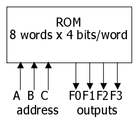
F1 = A' B' C + A' B C' + A B C

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

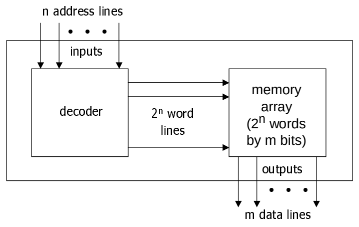
truth table



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
  - Xilinx field programmable gate arrays (FPGAs) 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)
- Use multiple levels of PALs/PLAs/ROMs
  - Output intermediate result
  - Make it an input to be used in further logic

Combinational Logic Implementation Summary

- Multi-level Logic
  - Conversion to NAND-NAND and NOR-NOR networks
  - Transition from simple gates to more complex gate building blocks
  - Reduced gate count, fan-ins, potentially faster
  - More levels, harder to design
- Time Response in Combinational Networks
  - Gate delays and timing waveforms
  - Hazards/glitches (what they are and why they happen)
- Regular Logic
  - Multiplexers/decoders
  - ROMs
  - PLAs/PALs
  - Advantages/disadvantages of each