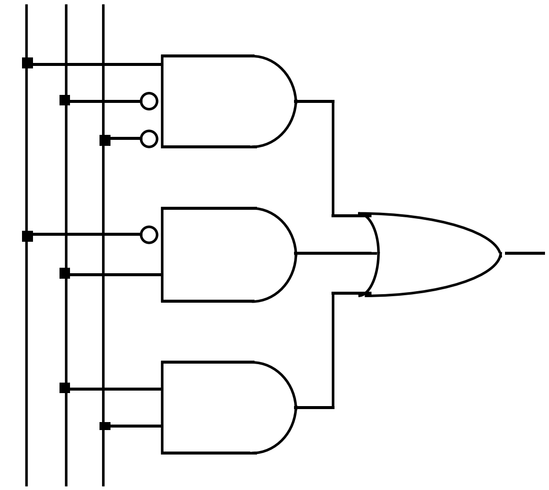
Combinational logic implementation

- **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**
  - multiplexors
  - decoders
  - PAL/PLAs
  - ROMs

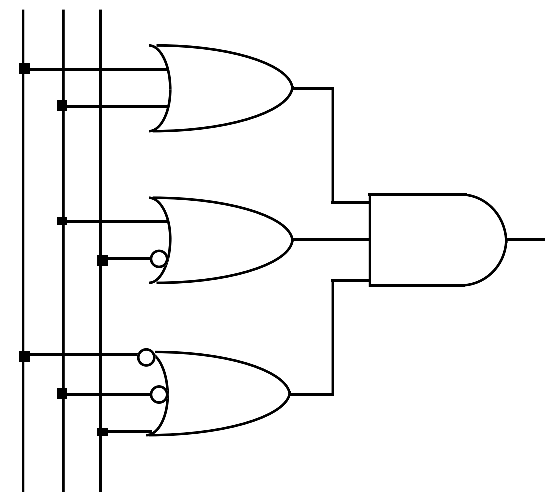
© 1996 Gaetano Borriello

Implementations of two-level logic

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



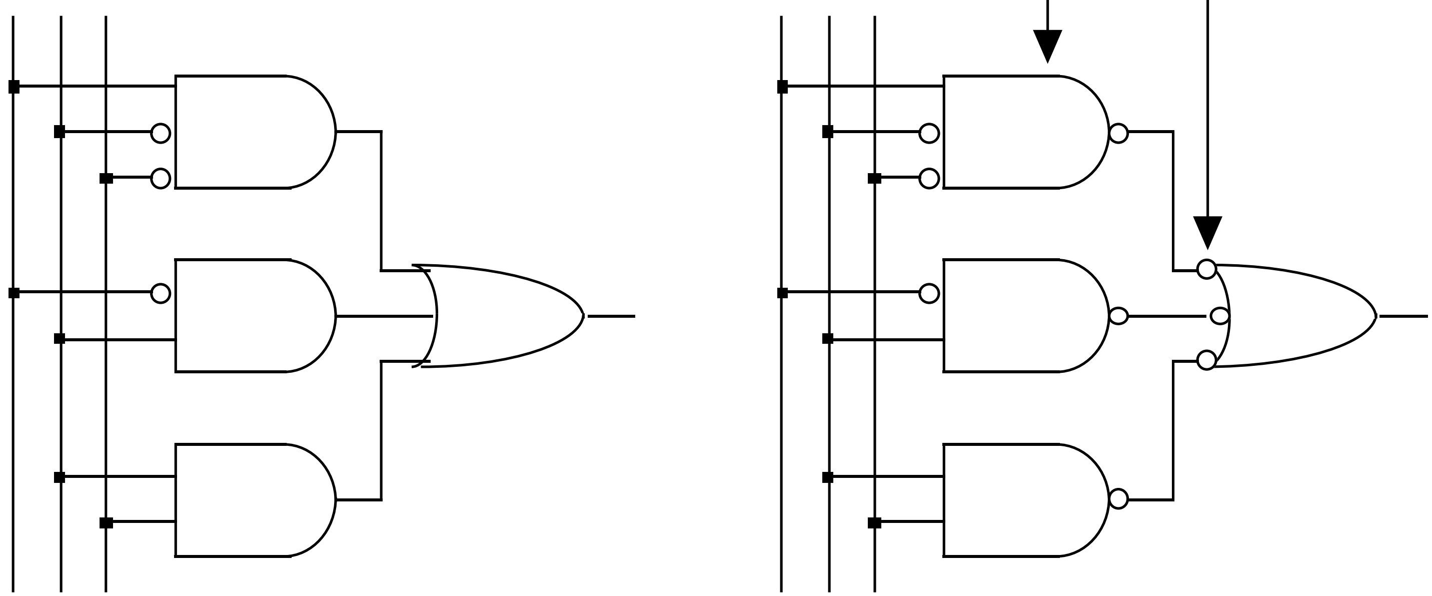
- **Product-of-sums**
  - OR gates to form sum terms (maxterms)
  - AND gates to form product



© 1996 Gaetano Borriello

Two-level logic using NAND gates

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

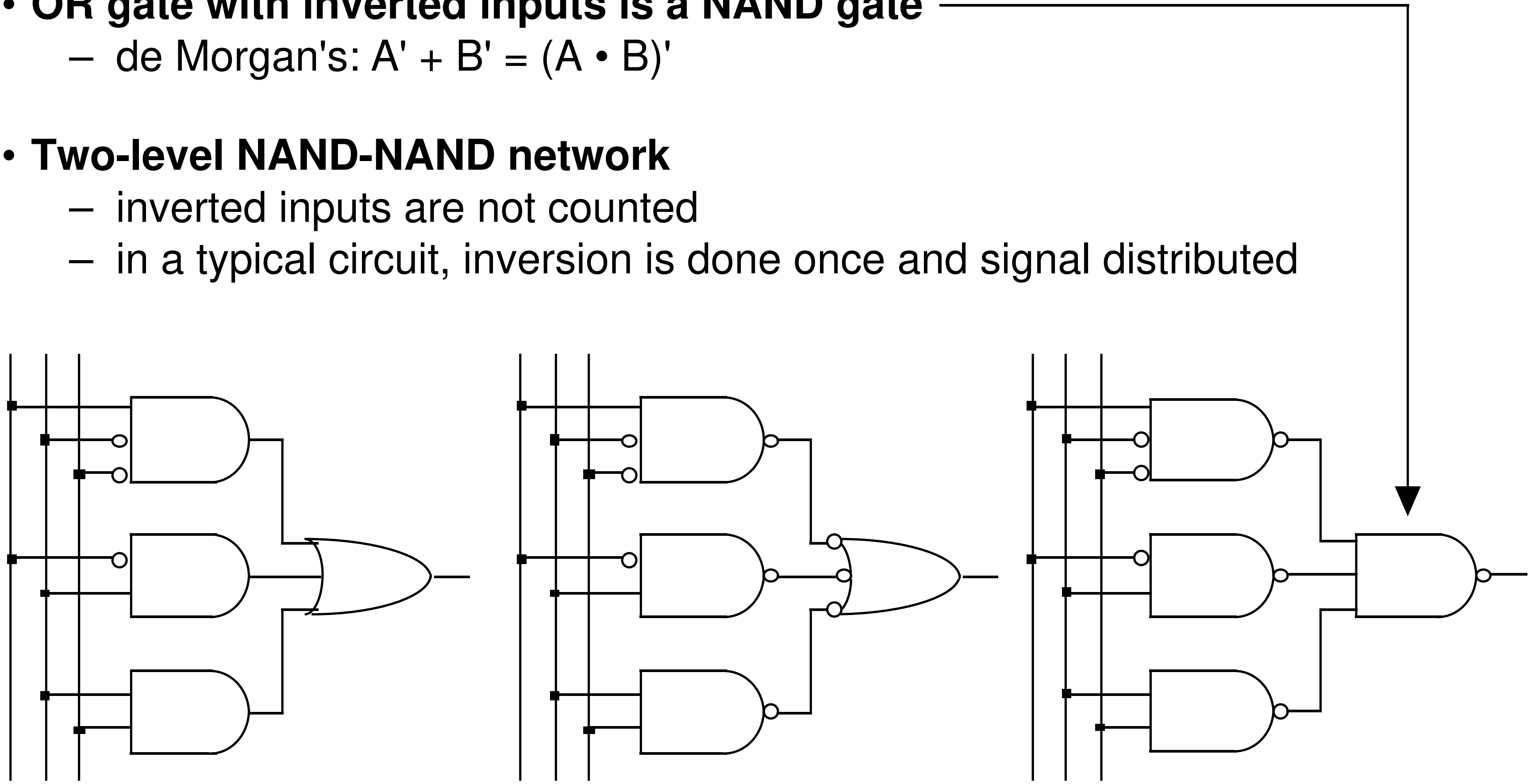


© 1996 Gaetano Borriello

Two-level logic using NAND gates (cont'd)

- **OR gate with inverted inputs is a NAND gate**
  - de Morgan's: A' + B' = (A • B)'

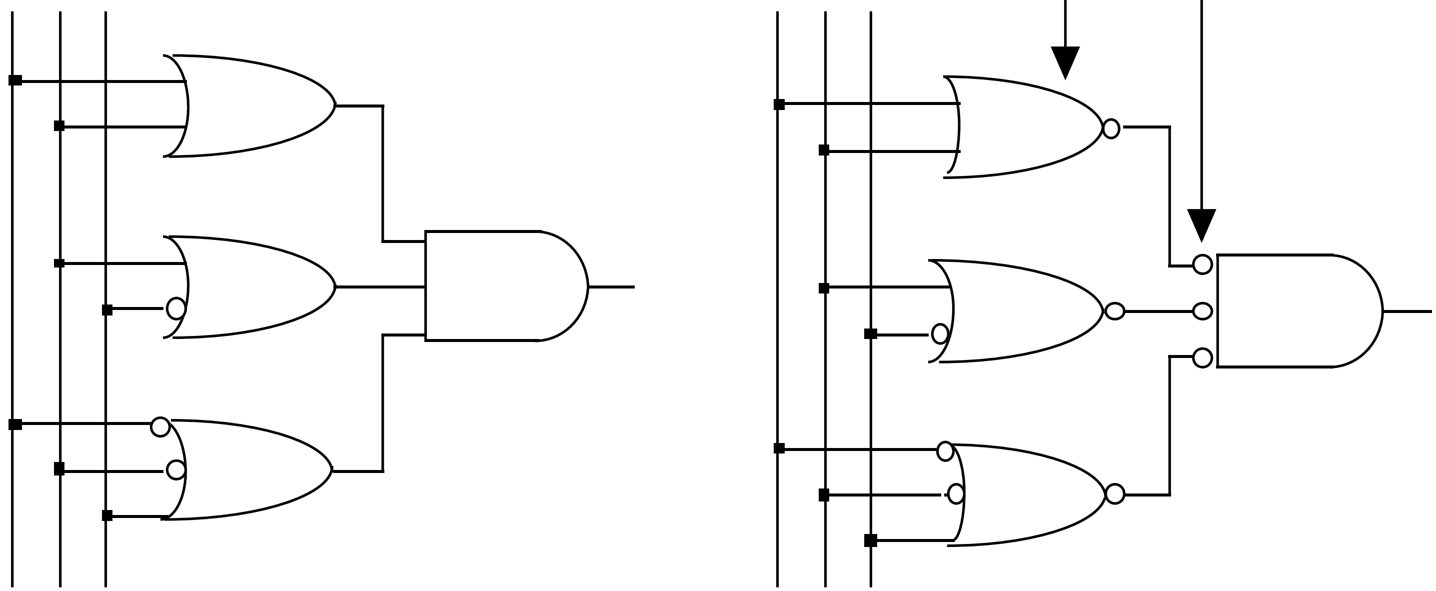
- **Two-level NAND-NAND network**
  - inverted inputs are not counted
  - in a typical circuit, inversion is done once and signal distributed



© 1996 Gaetano Borriello

Two-level logic using NOR gates

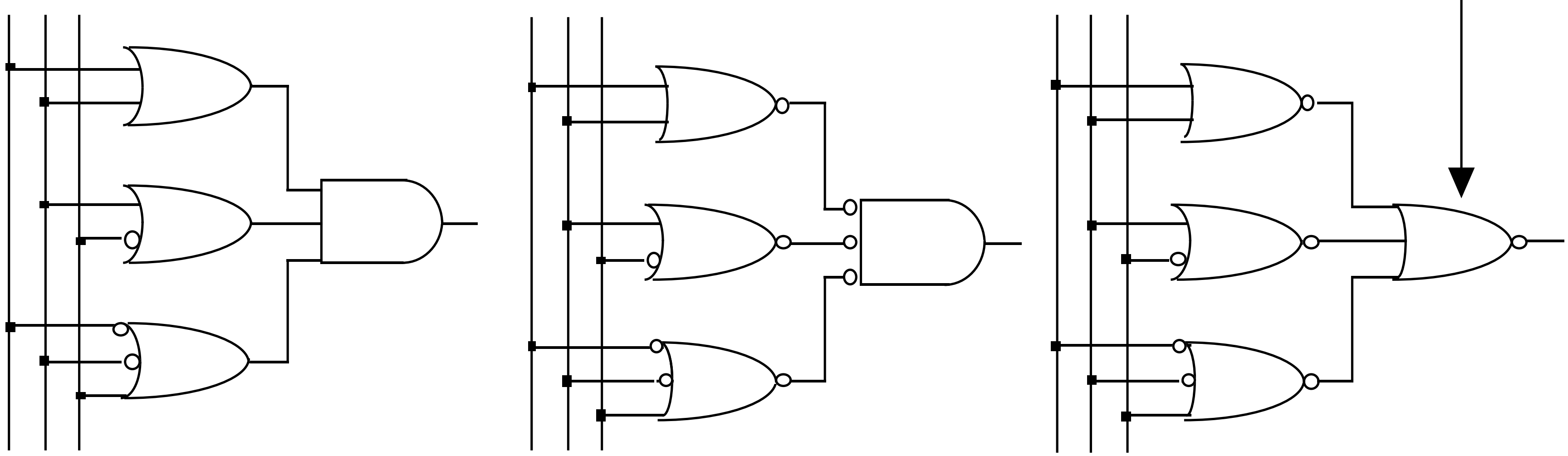
- Replace maxterm OR gates with NOR gates
- Place compensating inversion at inputs of AND gate



© 1996 Gaetano Borriello

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

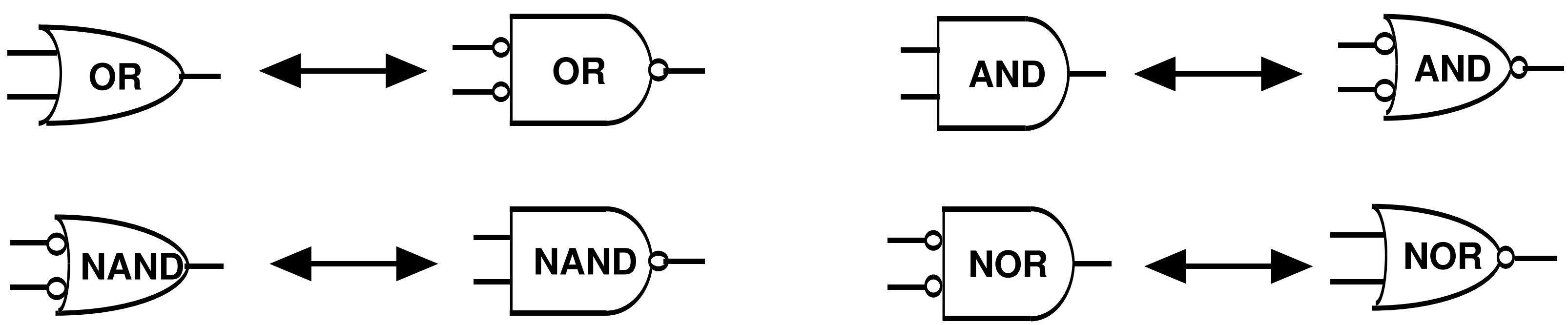


© 1996 Gaetano Borriello

Two-level logic using NAND and NOR gates

- **NAND-NAND and NOR-NOR networks**
  - de Morgan's law: $(A + B)' = A' \bullet B'$ $(A \bullet B)' = A' + B'$
  - written differently: $A + B = (A' \bullet B')'$ $(A \bullet B) = (A' + B')'$

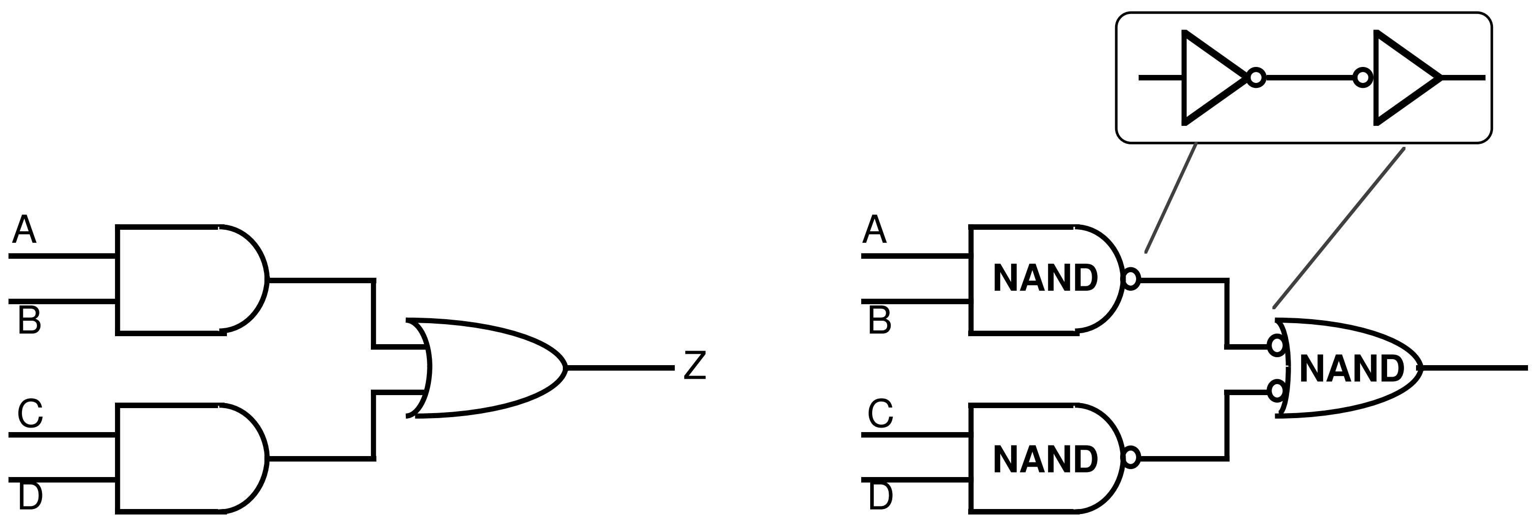
- **In other words —**
  - OR is the same as NAND with complemented inputs
  - AND is the same as NOR with complemented inputs
  - NAND is the same as OR with complemented inputs
  - NOR is the same as AND with complemented inputs



© 1996 Gaetano Borriello

Conversion between forms

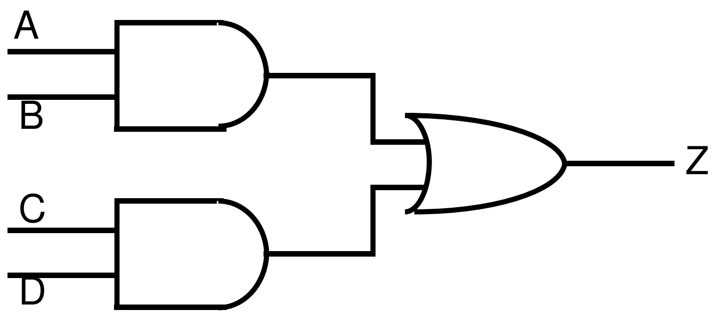
- **Convert from networks of ANDs and ORs to networks of NANDs and NORs**
  – introduce appropriate inversions ("bubbles")
- **Each introduced "bubble" must be matched by a corresponding "bubble"**
  – conservation of inversions
  – do not alter logic function
- **Example: AND/OR to NAND/NAND**

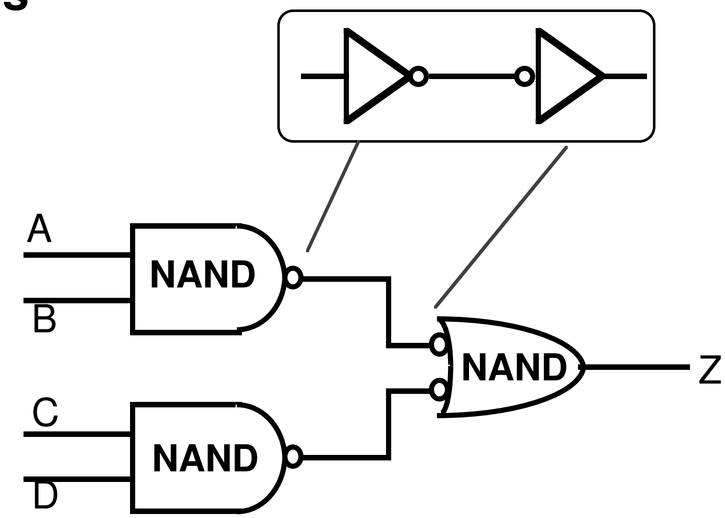


© 1996 Gaetano Borriello

Conversion between forms (cont'd)

- **Example: verify equivalence of two forms**



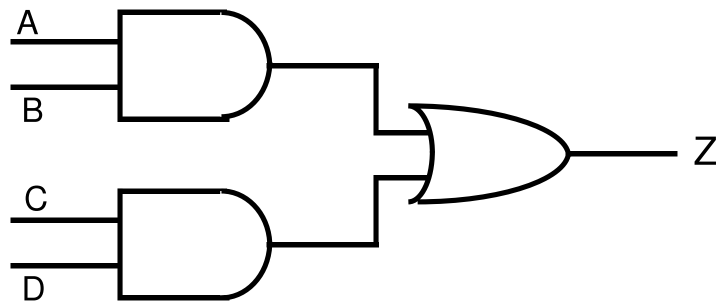


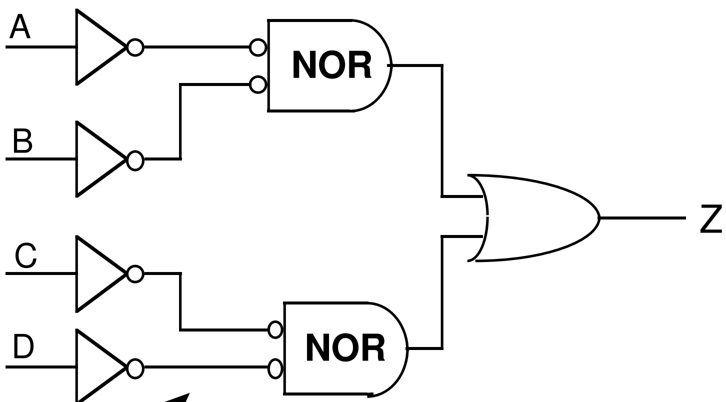
$$
\begin{aligned}
Z &= [\,(A \cdot B)' \cdot (C \cdot D)'\,]' \\
&= [\,(A' + B') \cdot (C' + D')\,]' \\
&= [\,(A' + B')' + (C' + D')'\,] \\
&= (A \cdot B) + (C \cdot D) \ \surd
\end{aligned}
$$

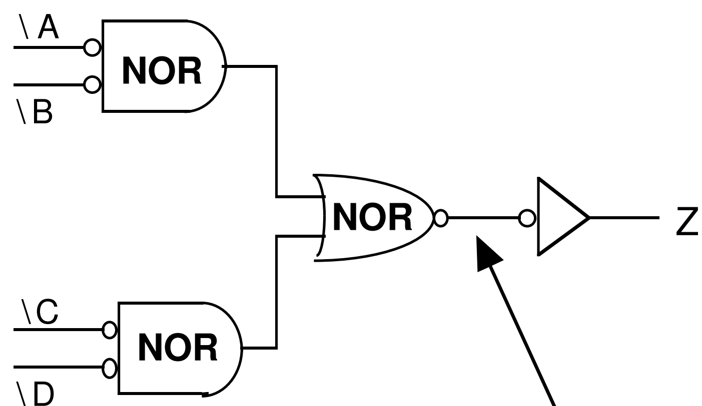
© 1996 Gaetano Borriello

Conversion between forms (cont'd)

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







Step 1

Step 2

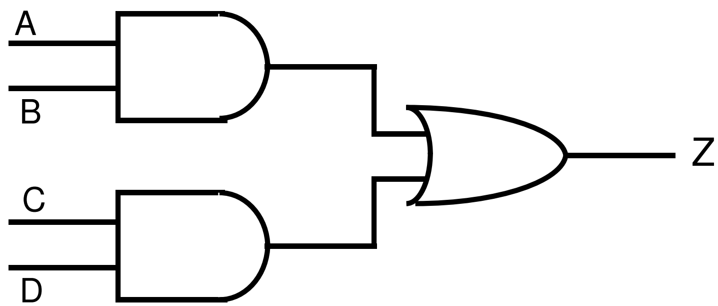
conserve "bubbles"

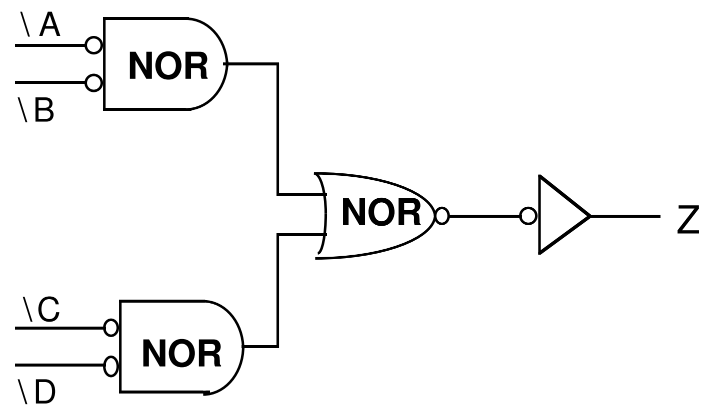
conserve "bubbles"

© 1996 Gaetano Borriello

Conversion between forms (cont'd)

- **Example: verify equivalence of two forms**





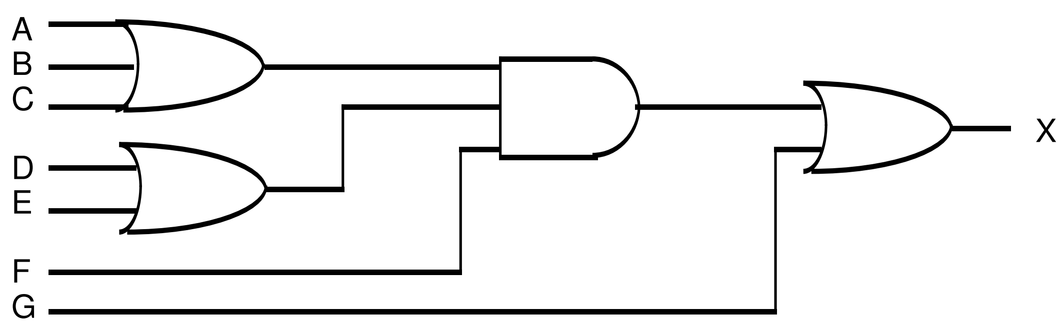
$$
\begin{aligned}
Z &= \{ \; [\, (A' + B')' + (C' + D')' \,]' \; \}' \\
&= \{ \; (A' + B') \cdot (C' + D') \; \}' \\
&= (A' + B')' + (C' + D')' \\
&= (A \cdot B) + (C \cdot D) \quad \surd
\end{aligned}
$$

© 1996 Gaetano Borriello

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 (that may not even 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)



© 1996 Gaetano Borriello

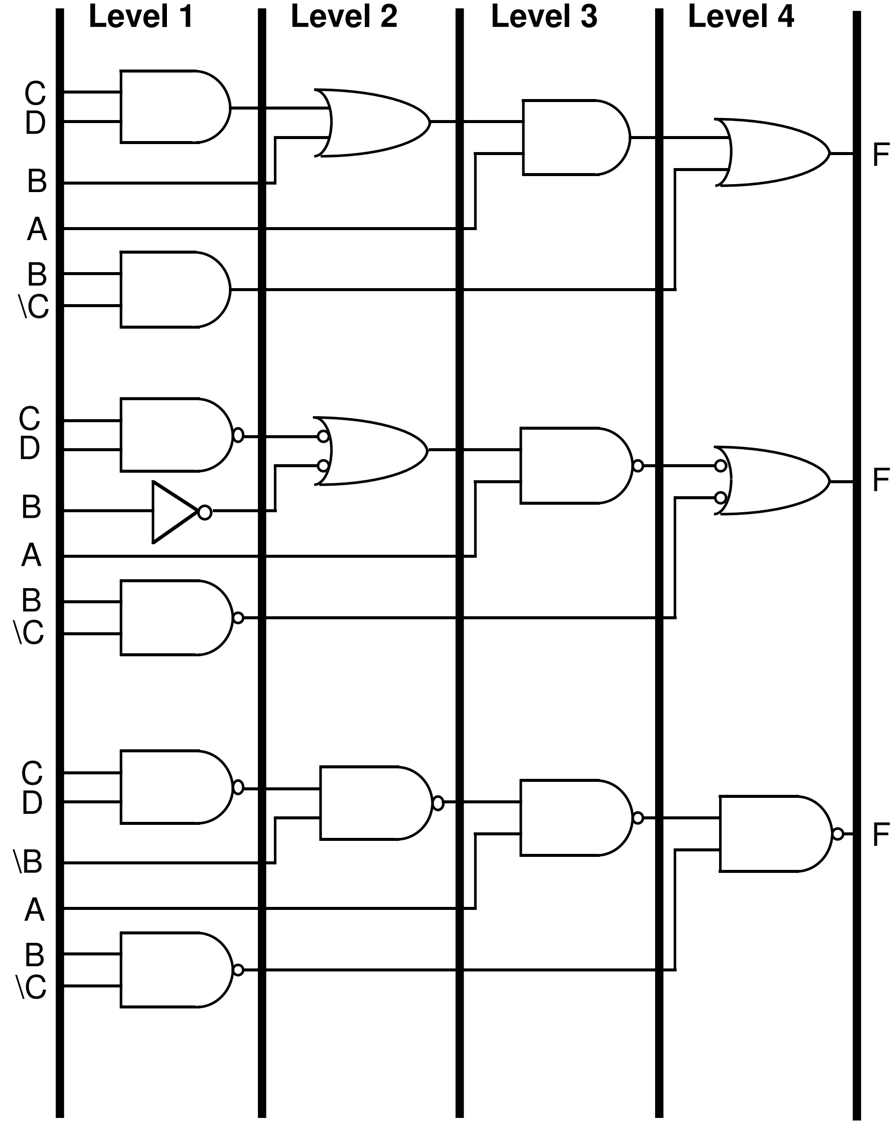
Conversion of multi-level logic to NANDs

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

original AND-OR network

introduction and conservation of bubbles

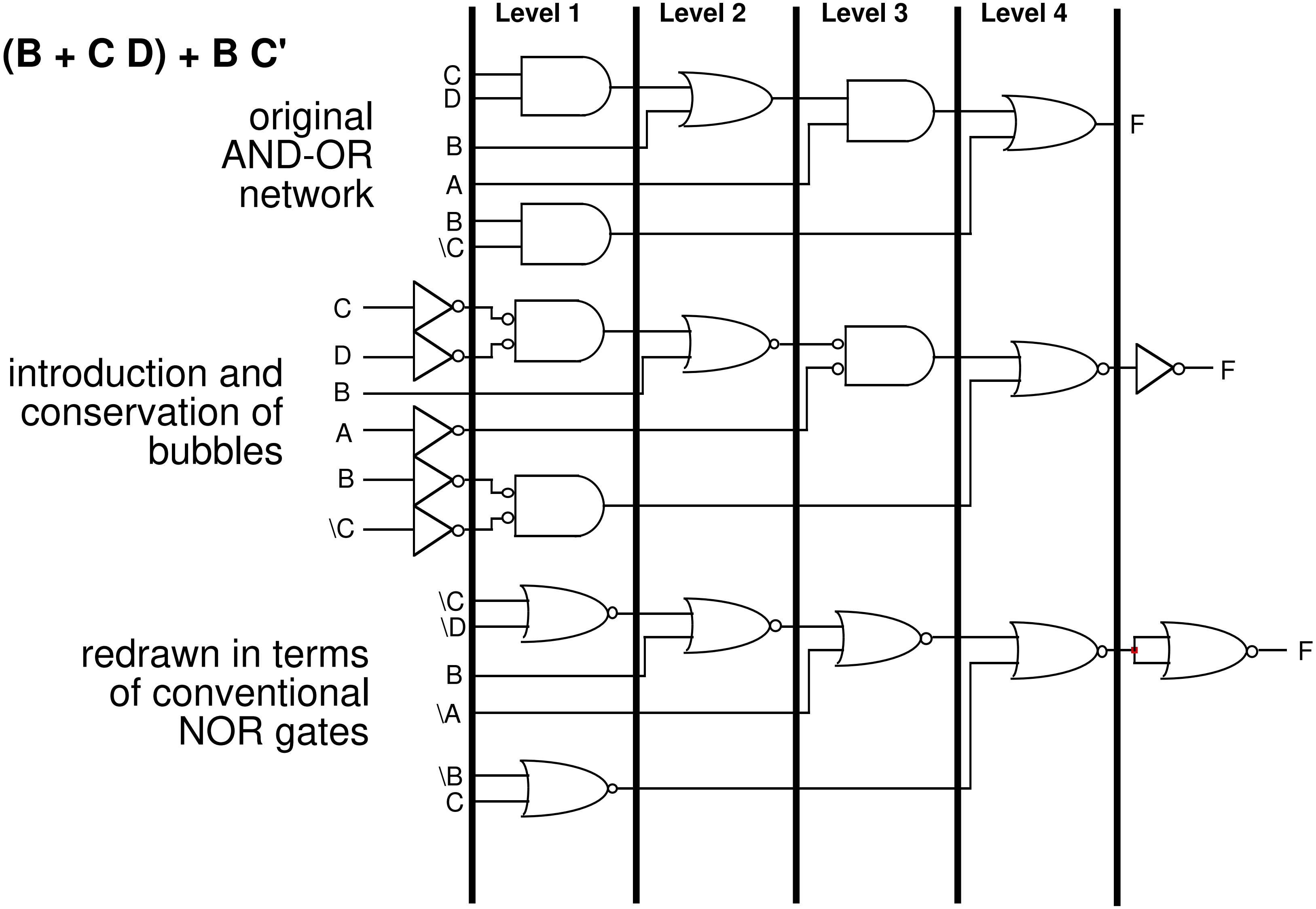
redrawn in terms of conventional NAND gates



© 1996 Gaetano Borriello

Conversion of multi-level logic to NORs

$$F = A\,(B + C\,D) + B\,C'$$

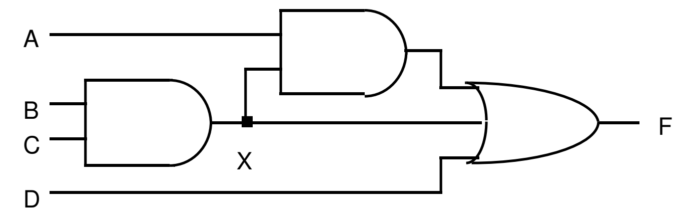


© 1996 Gaetano Borriello

Conversion between forms

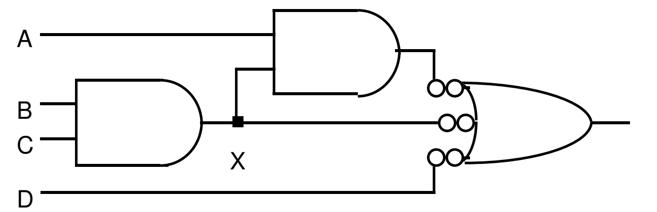
- **Example**

(a)



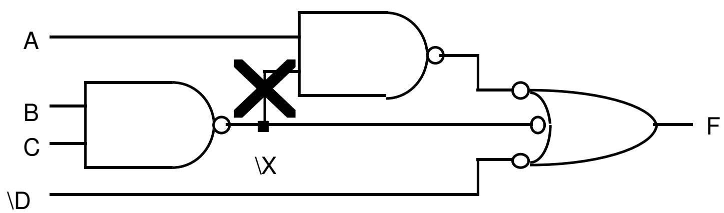
original circuit

(b)



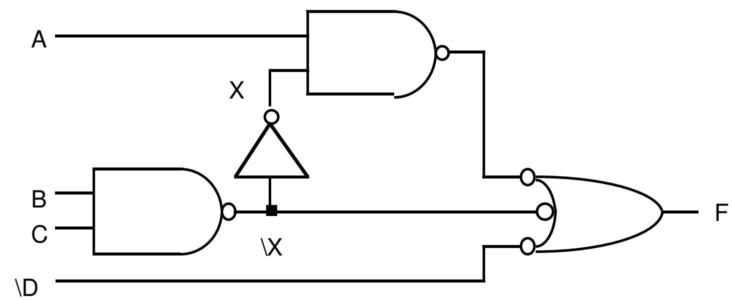
add double bubbles at inputs

(c)



distribute bubbles
some mismatches

(d)

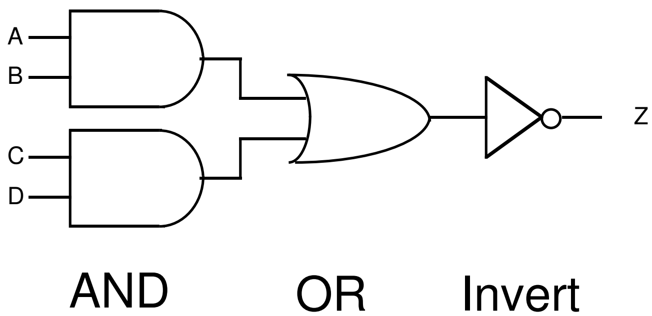


insert inverters to fix mismatches

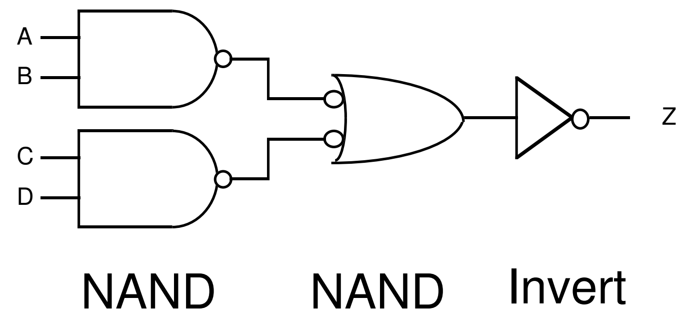
© 1996 Gaetano Borriello

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

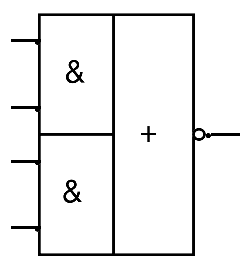
logical concept



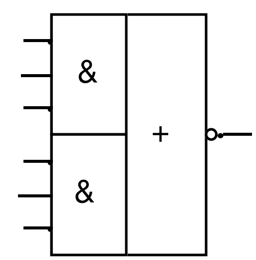
possible implementation



2x2 AOI gate symbol



3x2 AOI gate symbol

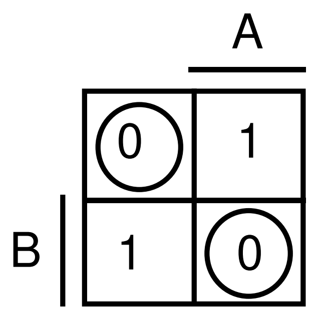


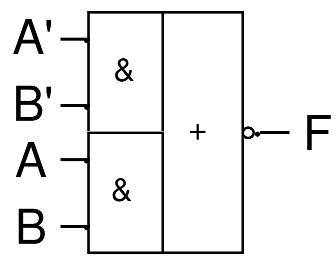
© 1996 Gaetano Borriello

Conversion to AOI forms

- **General procedure to place in AOI form**
  - compute the 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)'



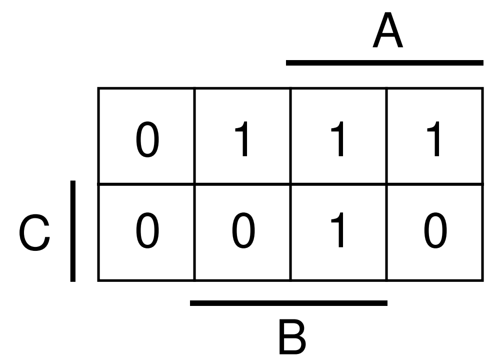


© 1996 Gaetano Borriello

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



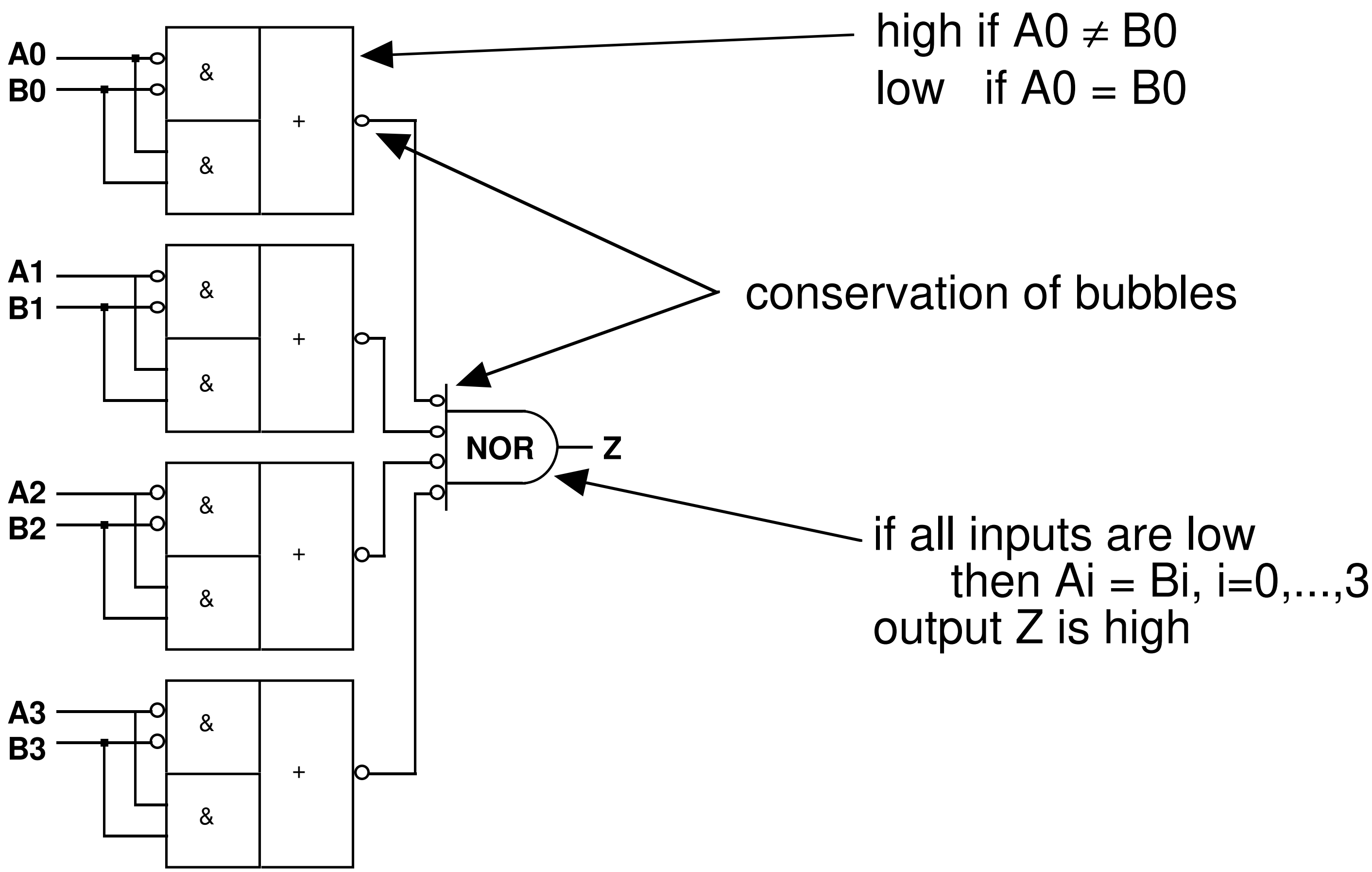
- **Example: 4-bit equality function**
  - $Z = (A0\,B0 + A0'\,B0')\,(A1\,B1 + A1'\,B1')\,(A2\,B2 + A2'\,B2')\,(A3\,B3 + A3'\,B3')$

Each implemented in a single 2x2 AOI gate

© 1996 Gaetano Borriello

Examples of using AOI gates (cont'd)

- **Example: AOI implementation of 4-bit equality function**



© 1996 Gaetano Borriello

Sumary 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

© 1996 Gaetano Borriello

Time behavior of combinational networks

- **Waveforms**
  - visualization of values carried on signal wires over time
  - useful in explaining sequences of events (changes in value)

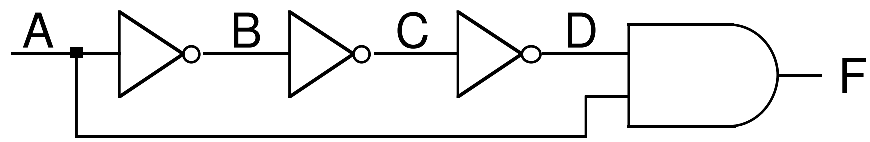
- **Simulation tools are used to create these waveforms**
  - input to the simulator includes gates and their connections
  - input stimulus, that is, input signal waveforms

- **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 that an output stays high or stays low between changes

© 1996 Gaetano Borriello

Momentary changes in outputs

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



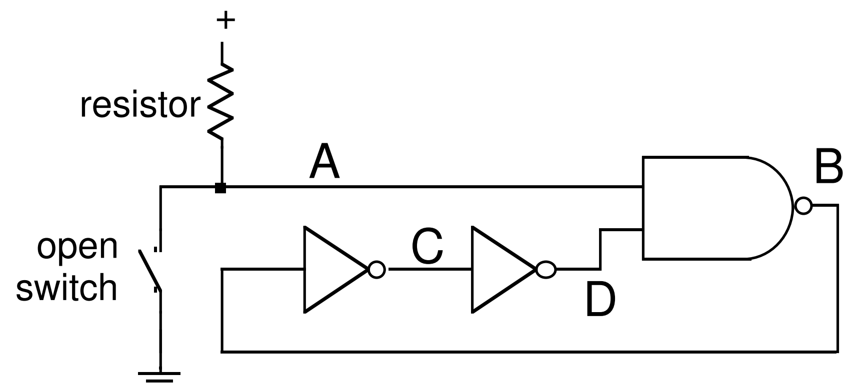
D remains high for
three gate delays after
A changes from low to high

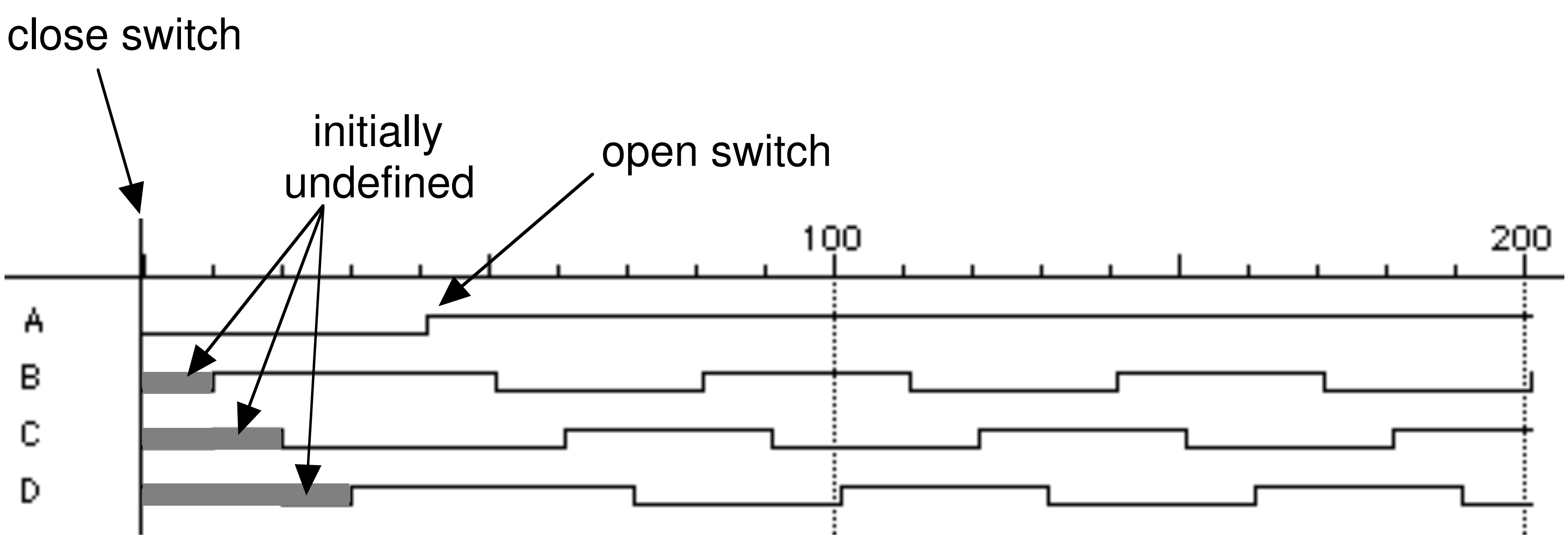
F is not always 0
pulse 3 gate-delays wide

© 1996 Gaetano Borriello

Oscillatory behavior

- **Another pulse shaping circuit**





© 1996 Gaetano Borriello

Hazards/glitches

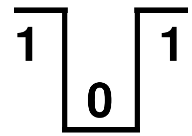
- **Hazards/glitches: unwanted switching at the outputs**
  - occur when different paths through circuit have different propagation delays
    - as in pulse shaping circuits we just analyzed
  - dangerous if logic makes a decision or causes an action while output is unstable
    - may need to guarantee 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)

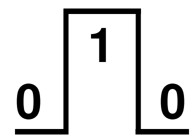
© 1996 Gaetano Borriello

Types of hazards

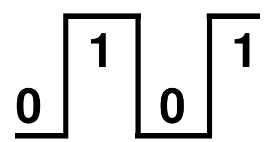
- **Static 1-hazard**
  - input change causes output to go from 1 to 0 to 1

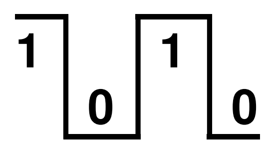


- **Static 0-hazard**
  - input change causes output to go from 0 to 1 to 0



- **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

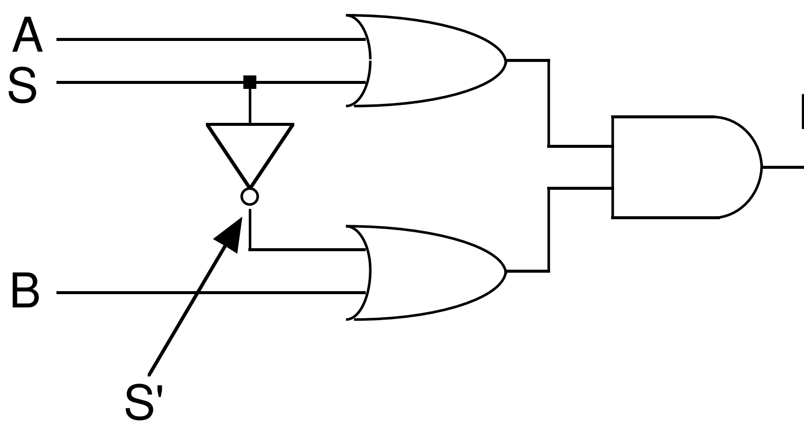


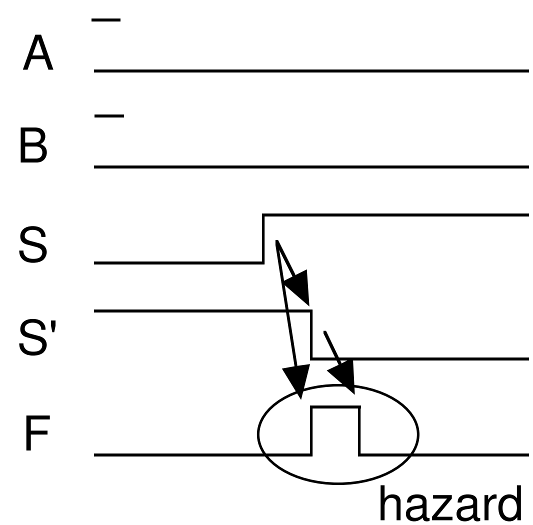


© 1996 Gaetano Borriello

Static hazards

- **Due to a literal and its complement momentarily taking on the same value**
  - through 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: multiplexer**



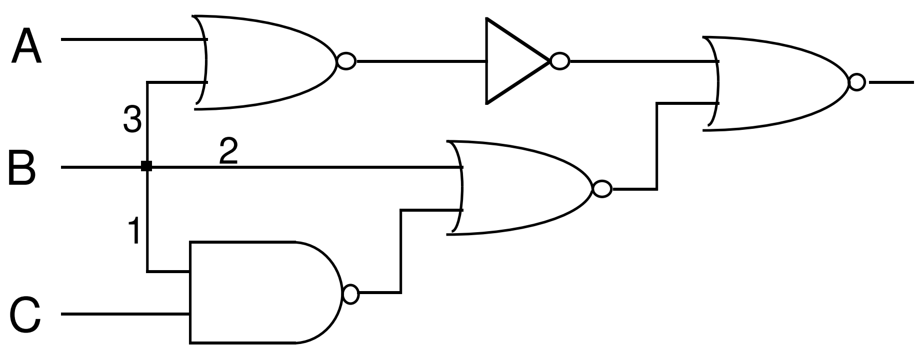


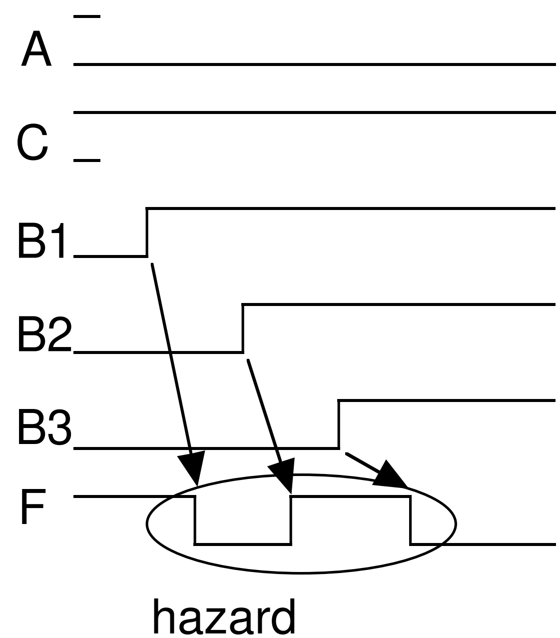
static-0 hazard static-1 hazard

© 1996 Gaetano Borriello

Dynamic hazards

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



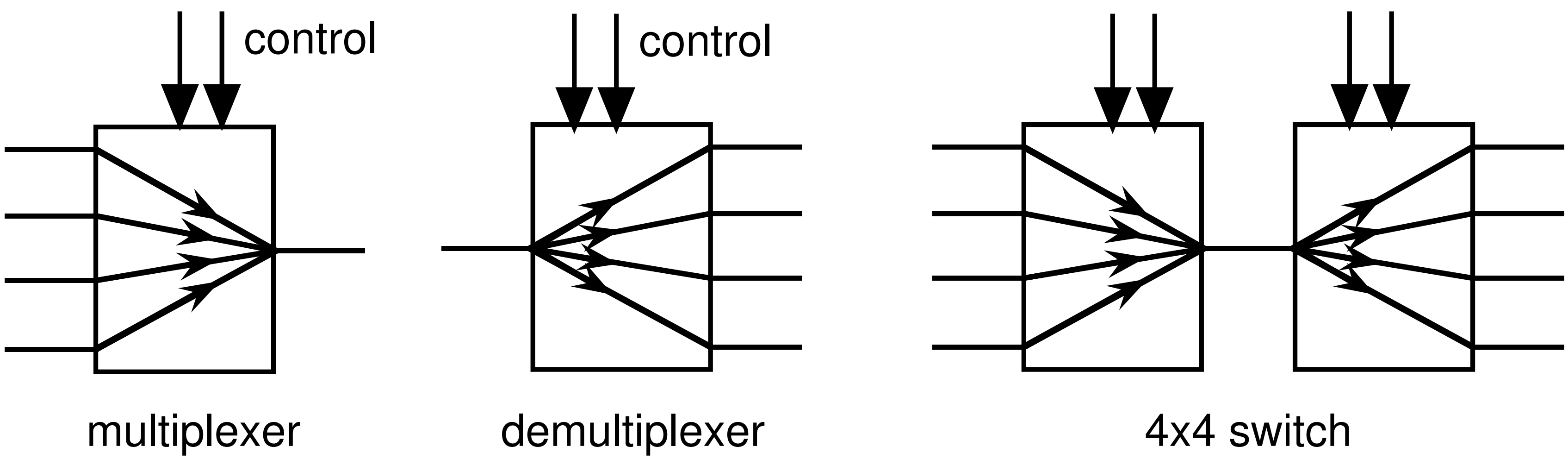


dynamic hazards

© 1996 Gaetano Borriello

Making connections

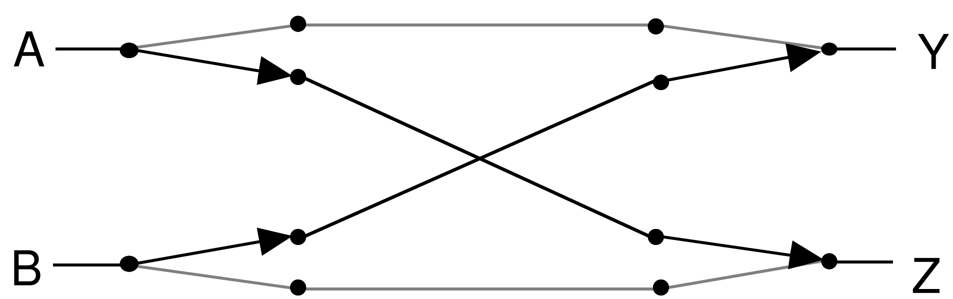
- **Direct point-to-point connections between gates**
  - wires we've seen so far
- **Route one of many inputs to a single output**
  - multiplexer
- **Route a single input to one of many outputs**
  - demultiplexer

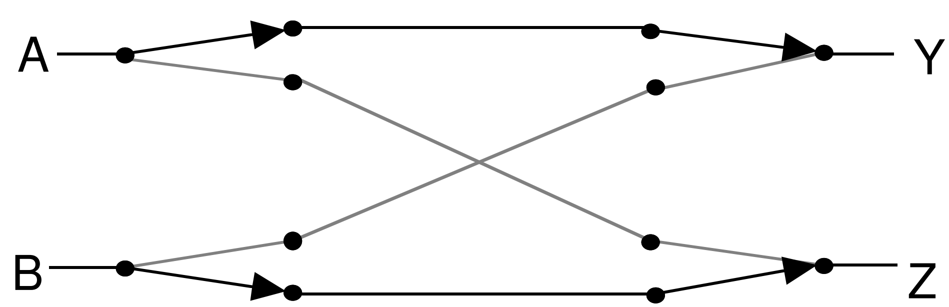


© 1996 Gaetano Borriello

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

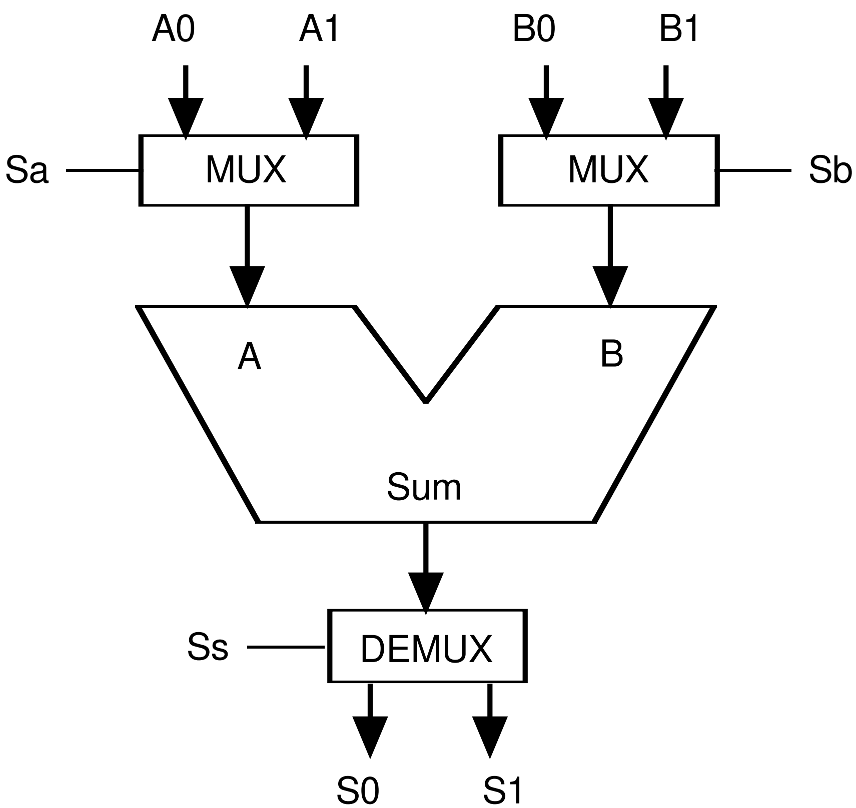




© 1996 Gaetano Borriello

Mux and demux (cont'd)

- **Uses of multiplexers/demultiplexers in multi-point connections**



multiple input sources

multiple output destinations

© 1996 Gaetano Borriello

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

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

functional form

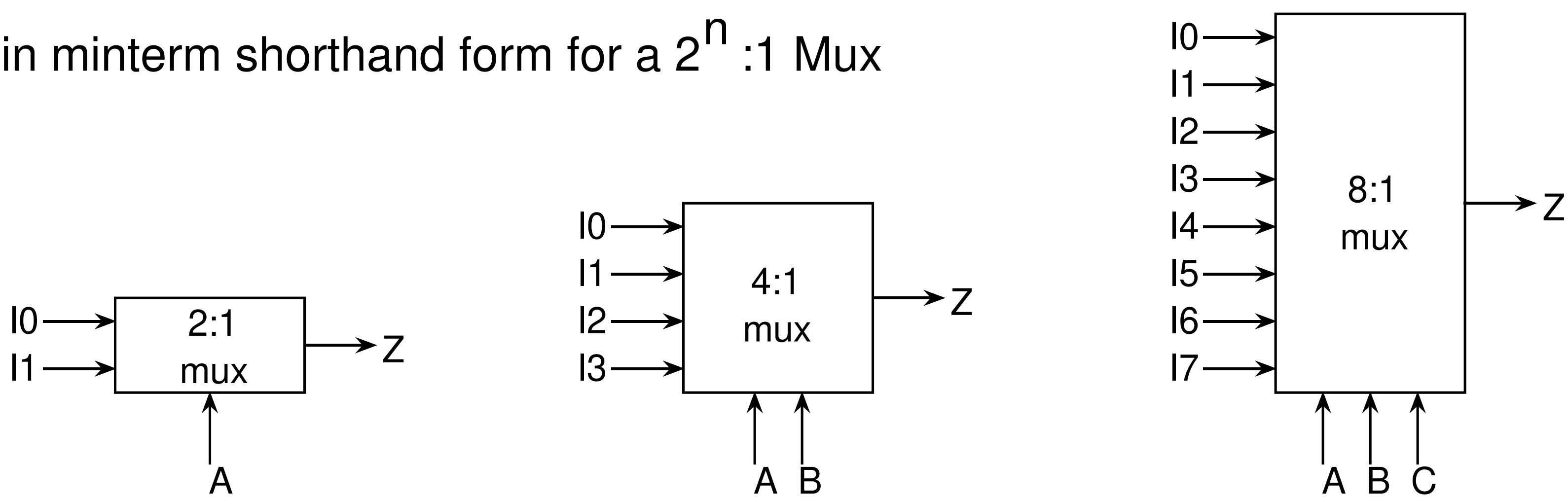
logical form

two alternative forms
for a 2:1 Mux truth table

© 1996 Gaetano Borriello

Multiplexers/selectors (cont'd)

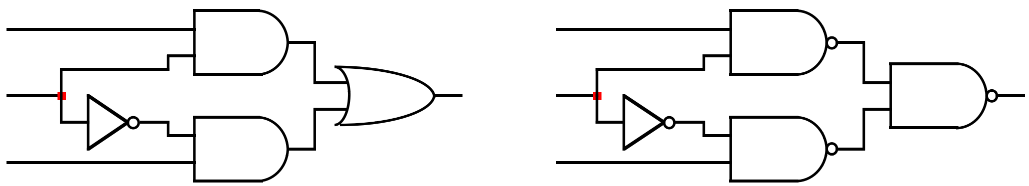
- **2:1 mux:** $Z = A' I0 + A I1$
- **4:1 mux:** $Z = A' B' I0 + A' B I1 + A B' I2 + A B I3$
- **8:1 mux:** $Z = A' B' C' I0 + A' B' C I1 + A' B C' I2 + A' B C I3 + A B' C' I4 + A B' C I5 + A B C' I6 + A B C I7$
- **In general,** $Z = \sum_{k=0}^{2^n - 1} m_k I_k$
  - in minterm shorthand form for a 2^n :1 Mux



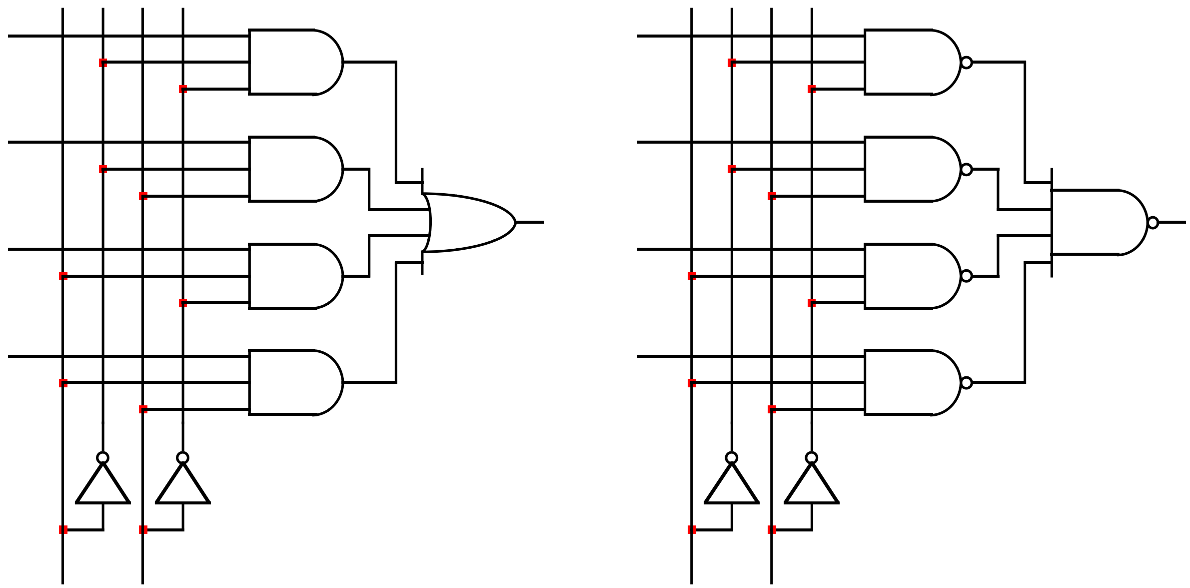
© 1996 Gaetano Borriello

Gate level implementation of muxes

- **2:1 mux**



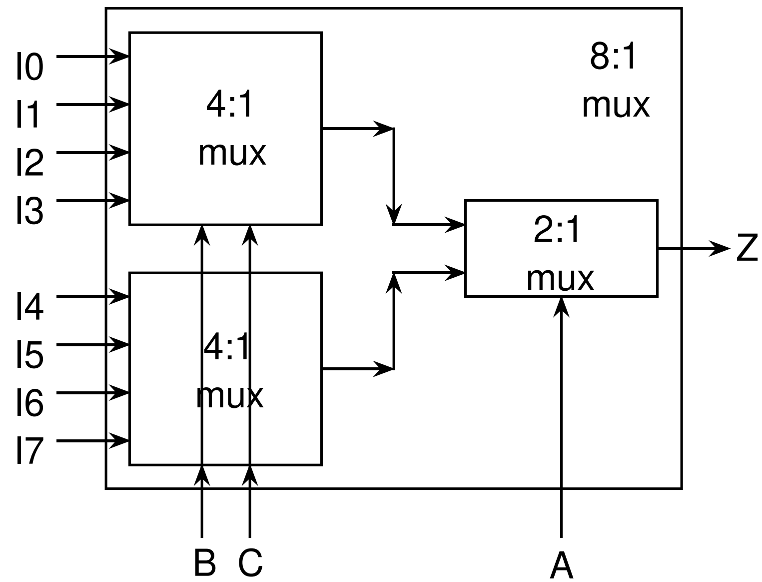
- **4:1 mux**



© 1996 Gaetano Borriello

Cascading multiplexers

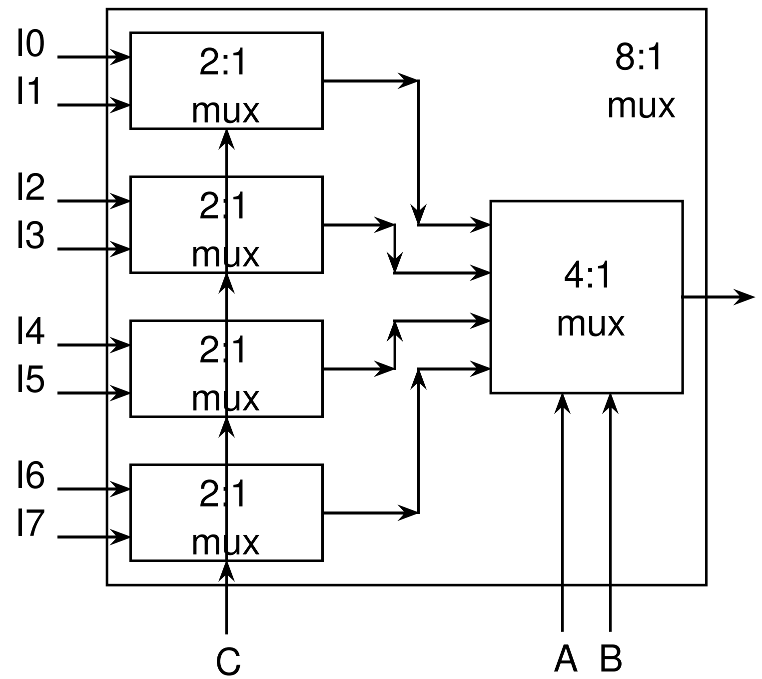
- **Large multiplexers can be implemented by cascading smaller ones**



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

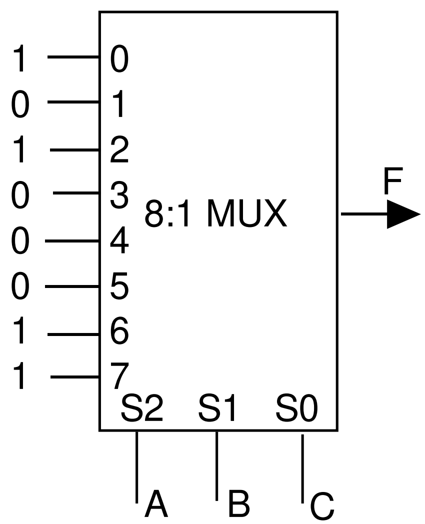
alternative implementation



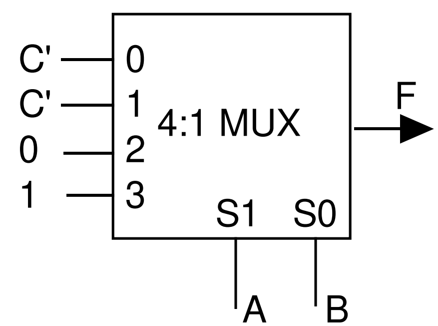
© 1996 Gaetano Borriello

Multiplexers as general-purpose logic

- **A 2^n:1 multiplexer can implement any function of n variables**
  - with the variables used as control inputs and
  - the data inputs tied to 0 or 1
  - in essence, a lookup table
- **A 2^{n-1}:1 multiplexer can implement any function of n variables**
  - with n-1 variables used as control inputs and
  - the 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)



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	



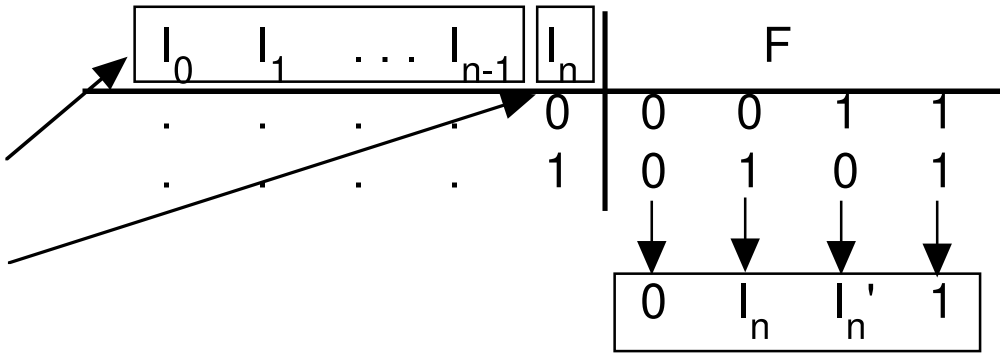
© 1996 Gaetano Borriello

Multiplexers as general-purpose logic (cont'd)

- **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 I_n, 0, 1

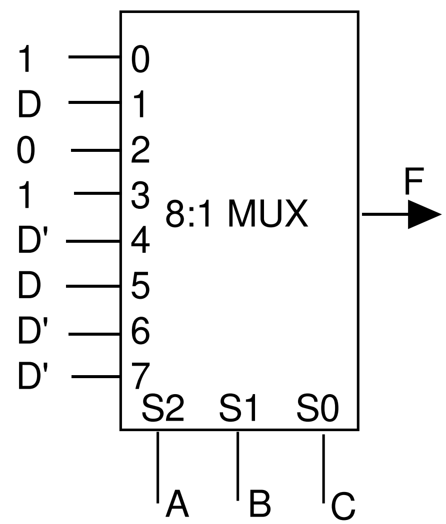
- **Example: F(A,B,C,D) can be implemented by an 8:1 MUX**

CD \ AB	00	01	11	10
00	1	0	1	1
01	1	0	0	0
11	1	1	0	1
10	0	1	1	0

choose A,B,C as control variables

multiplexer implementation

TTL package efficient may be gate inefficient



© 1996 Gaetano Borriello

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)

<u>1:2 Decoder:</u>

O0 = G • S'

O1 = G • S

<u>2:4 Decoder:</u>

O0 = G • S0' • S1'

O1 = G • S0' • S1

O2 = G • S0 • S1'

O3 = G • S0 • S1

<u>3:8 Decoder:</u>

O0 = G • S0' • S1' • S2'

O1 = G • S0' • S1' • S2

O2 = G • S0' • S1 • S2'

O3 = G • S0' • S1 • S2

O4 = G • S0 • S1' • S2'

O5 = G • S0 • S1' • S2

O6 = G • S0 • S1 • S2'

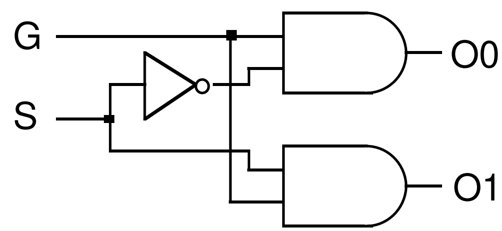
O7 = G • S0 • S1 • S2

© 1996 Gaetano Borriello

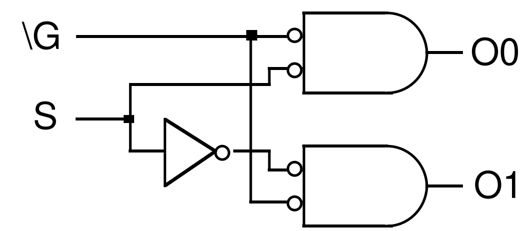
Gate level implementation of demuxes

- **1:2 decoders**

active-high enable

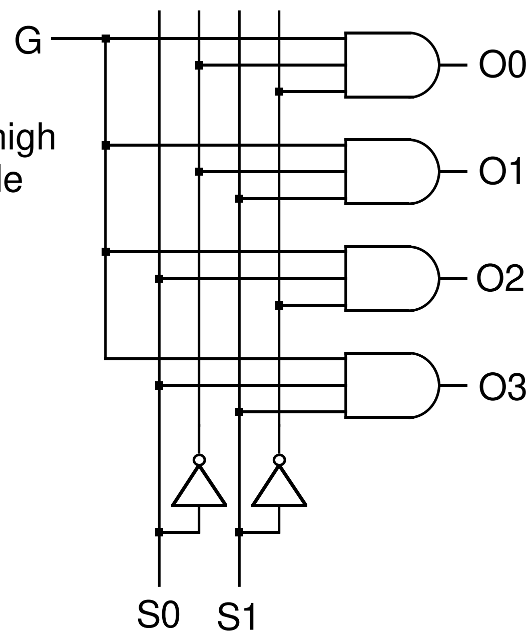


active-low enable

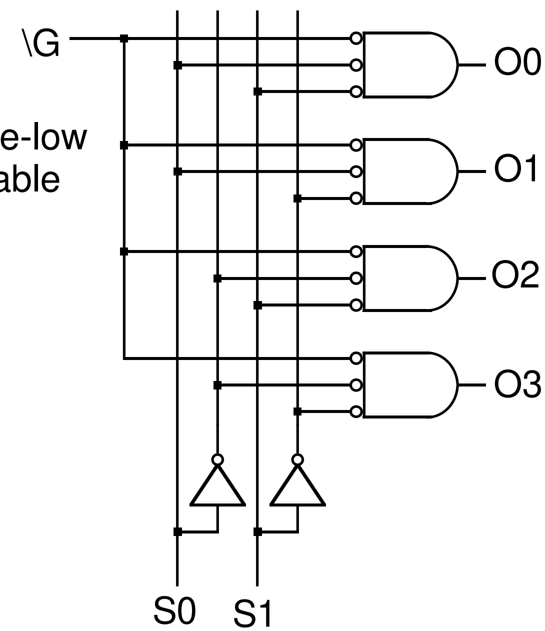


- **2:4 decoders**

active-high enable



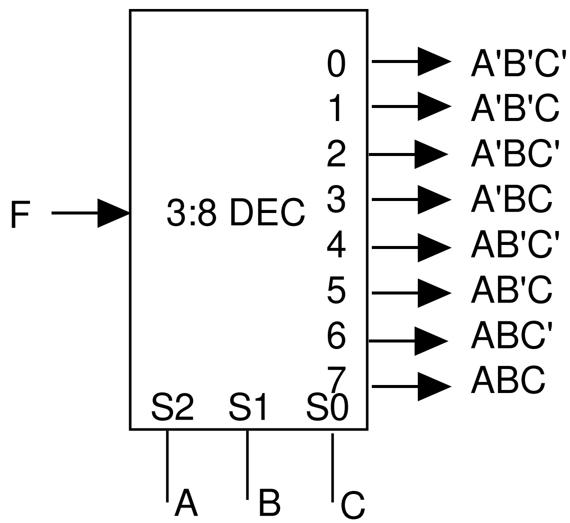
active-low enable



© 1996 Gaetano Borriello

Demultiplexers as general-purpose logic

- **A n:2^n decoder can implement any function of n variables**
  - with the variables used as control inputs
  - the enable inputs tied to 1 and
  - the appropriate minterms summed to form the function

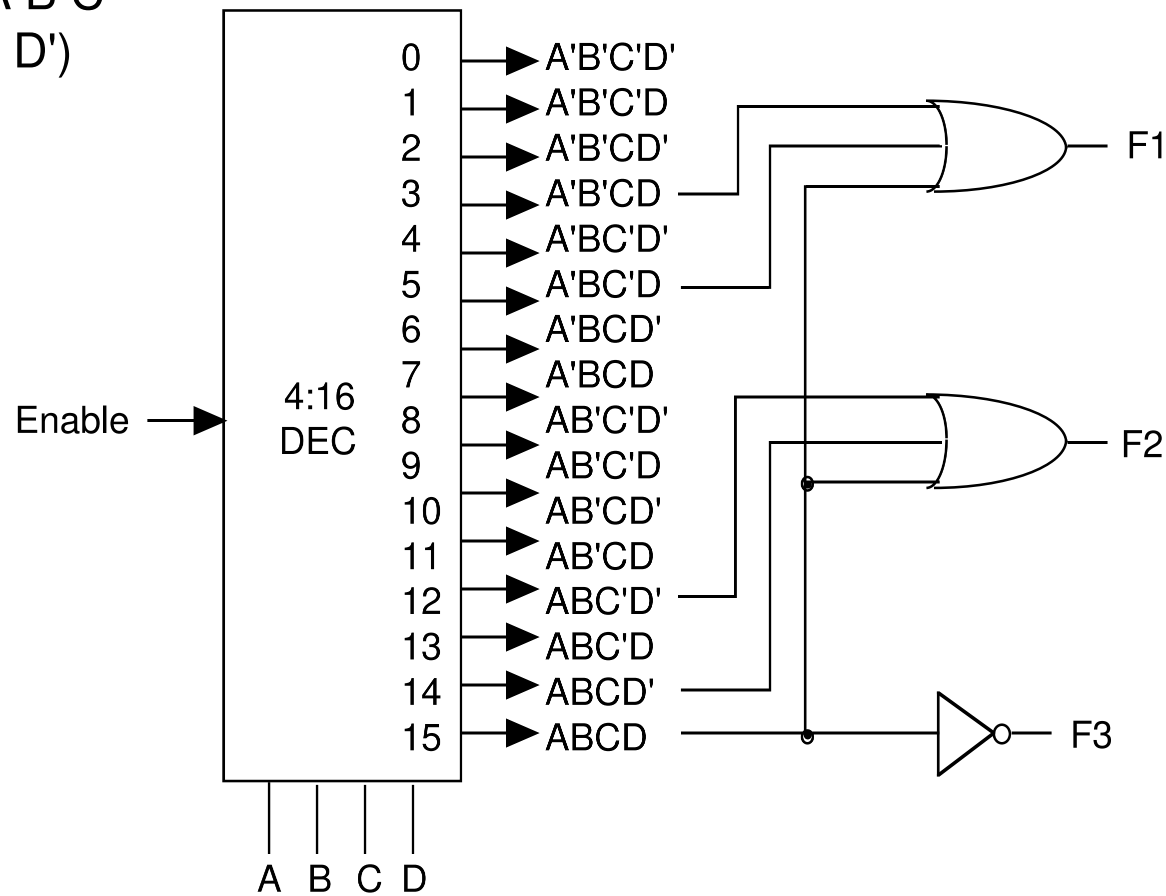


demultiplexer generates appropriate minterm based on control signals (it "decodes" control signals)

© 1996 Gaetano Borriello

Demultiplexers as general-purpose logic (cont'd)

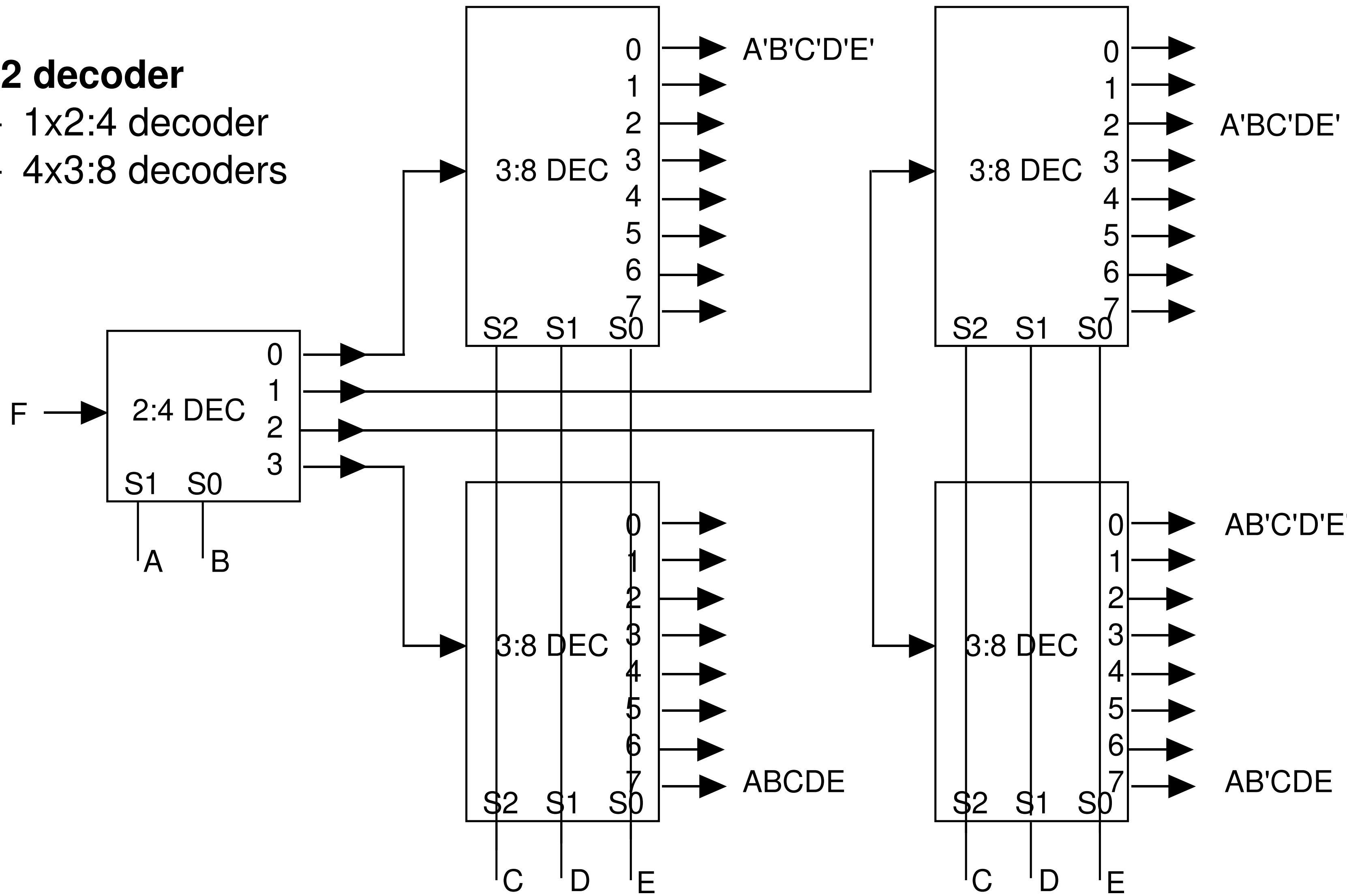
- **Example functions**
  - F1 = A' B C' D + A' B' C D + A B C D
  - F2 = A B C' D' + A B C
  - F3 = (A' + B' + C' + D')



© 1996 Gaetano Borriello

Cascading decoders

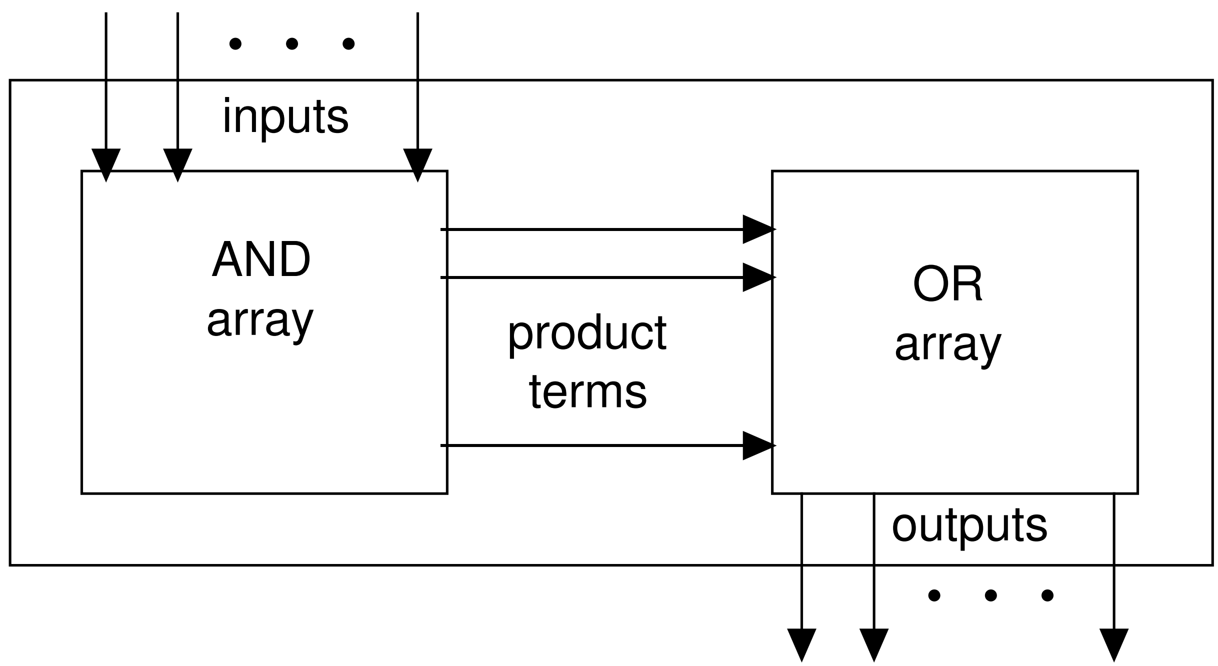
- **5:32 decoder**
  - 1x2:4 decoder
  - 4x3:8 decoders



© 1996 Gaetano Borriello

Programmable logic arrays

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



© 1996 Gaetano Borriello

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 = asserted in term
0 = negated in term
– = does not participate

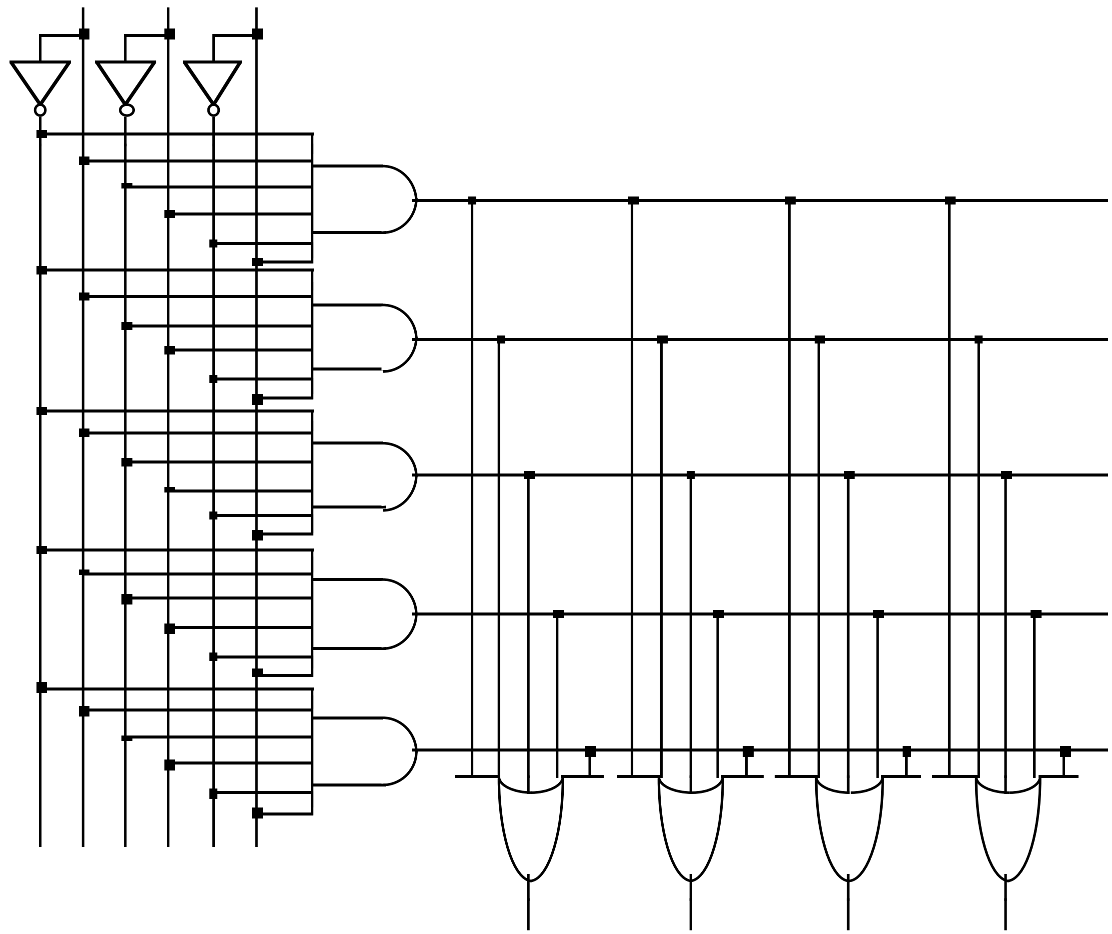
output side:

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

© 1996 Gaetano Borriello

Before programming

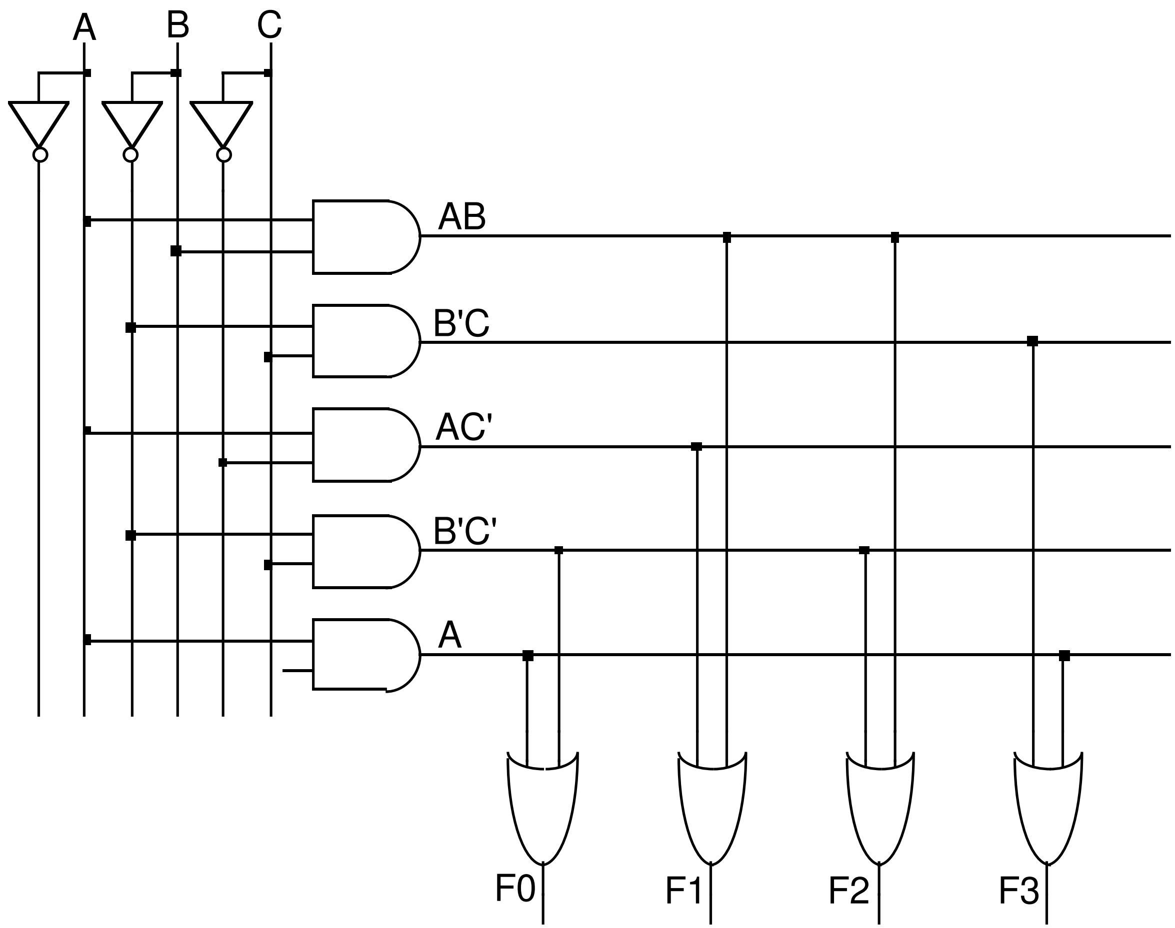
- **All possible connections are available before "programming"**



© 1996 Gaetano Borriello

After programming

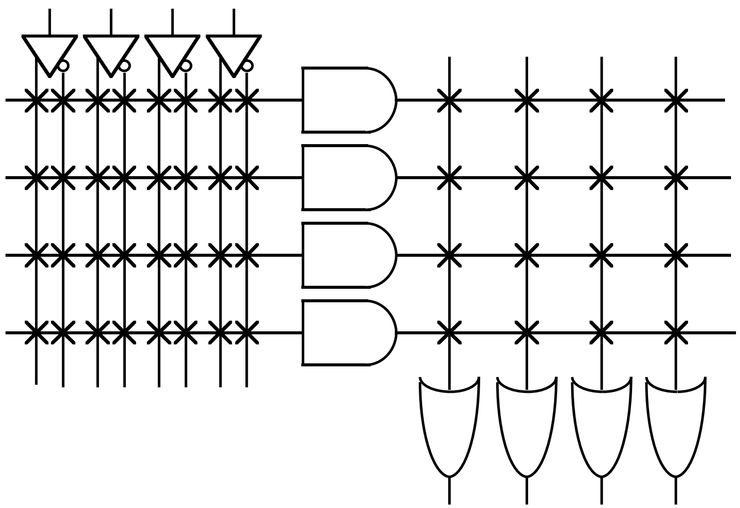
- **Unwanted connections are "blown"**
  - fuse (normally connected, break unwanted ones)
  - anti-fuse (normally disconnected, make wanted connections)



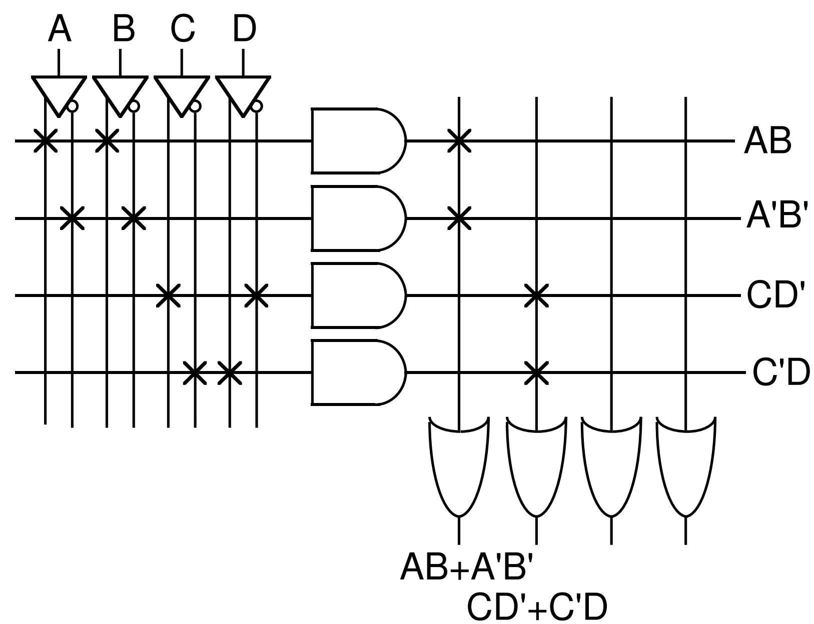
© 1996 Gaetano Borriello

Alternative 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

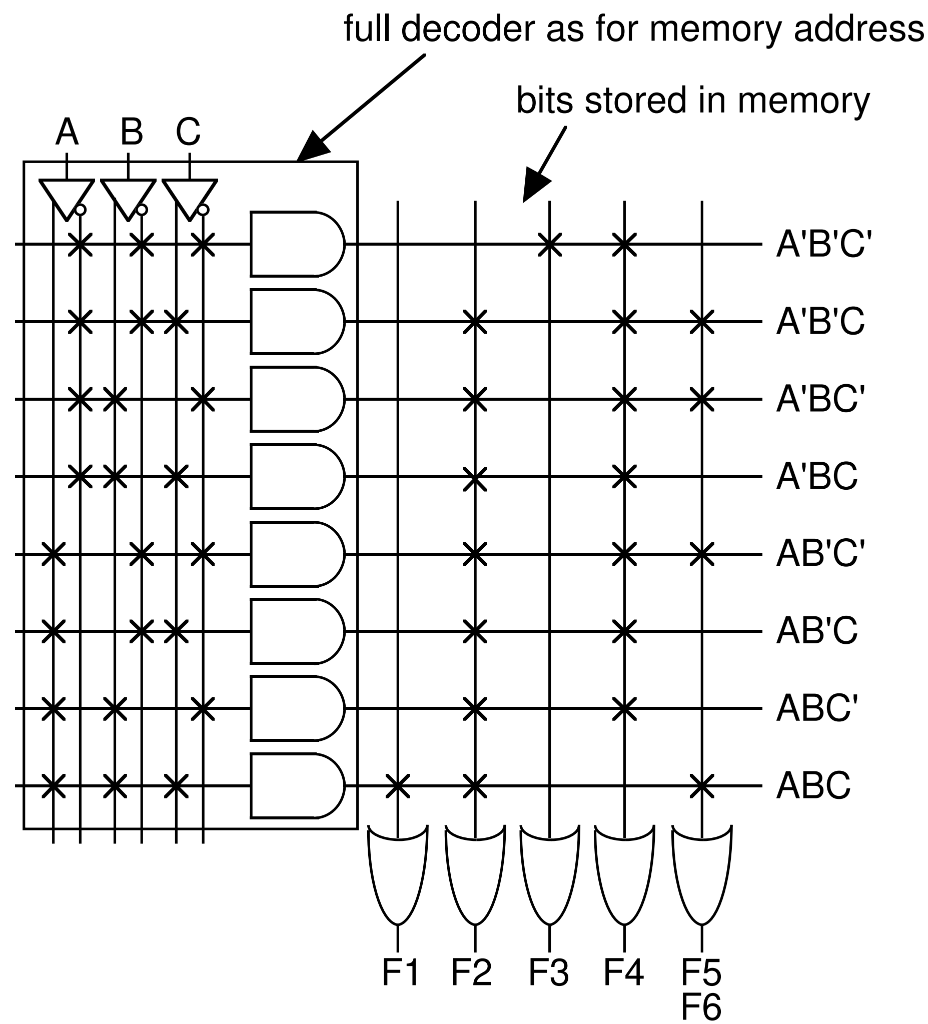


© 1996 Gaetano Borriello

Example of programmable logic array

- **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

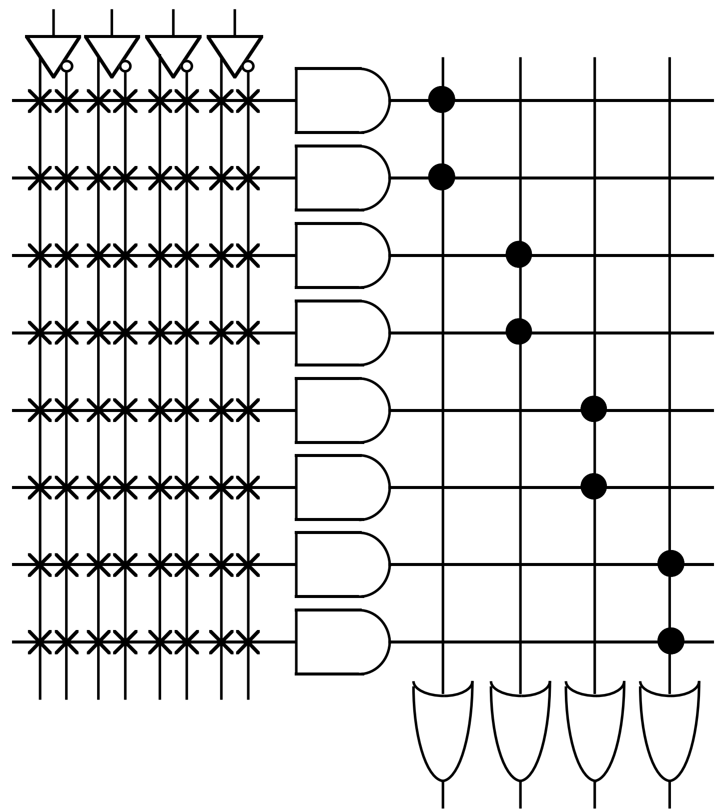
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



© 1996 Gaetano Borriello

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



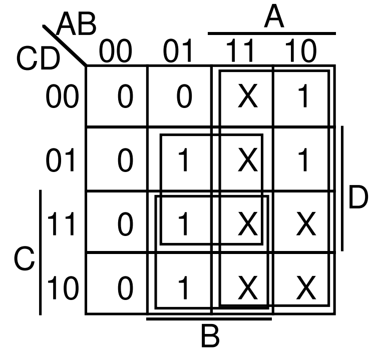
a given column of the OR array
has access to only a subset of
the possible product terms

© 1996 Gaetano Borriello

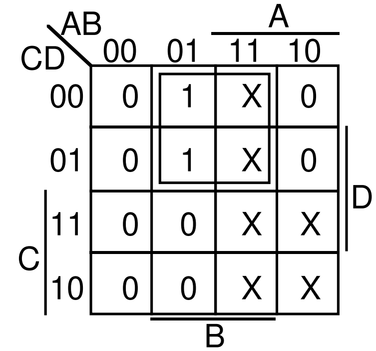
PALs and PLAs: design example

- **BCD to Gray code converter**

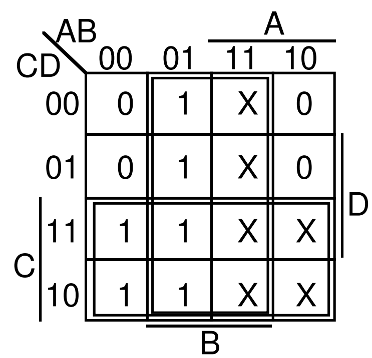
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–



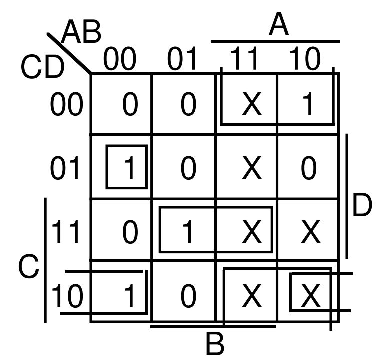
K-map for W



K-map for X



K-map for Y



K-map for Z

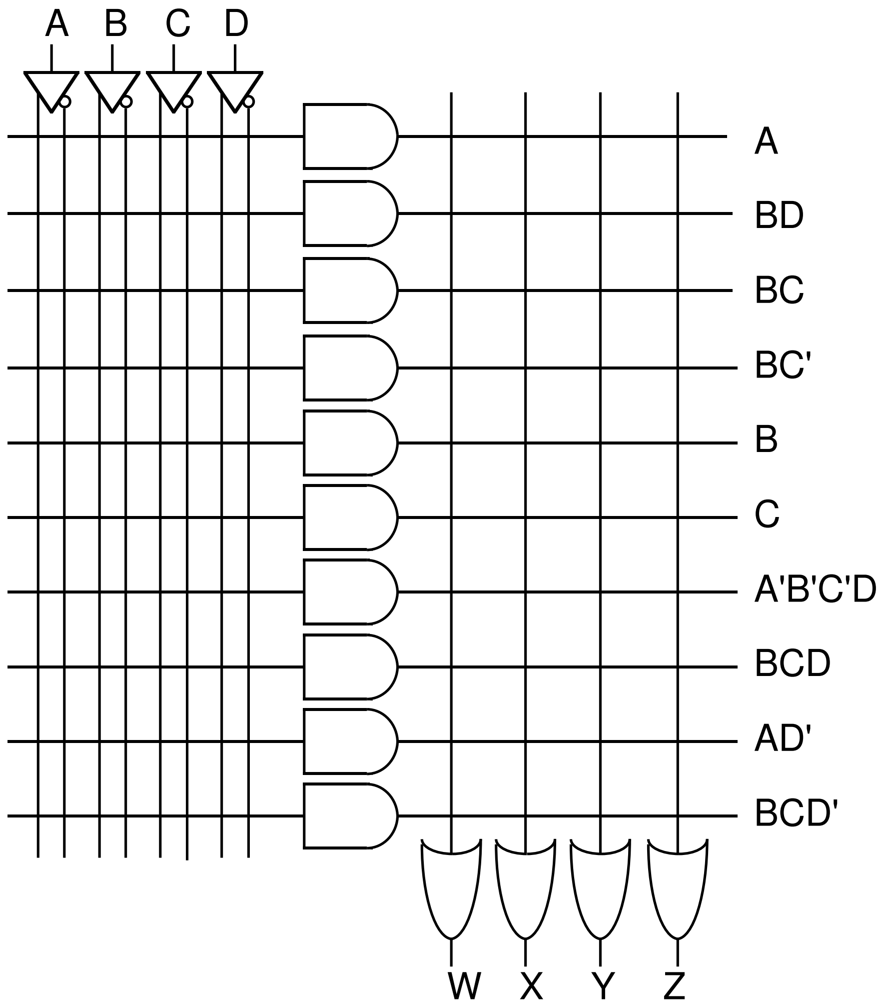
minimized functions:

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'

© 1996 Gaetano Borriello

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

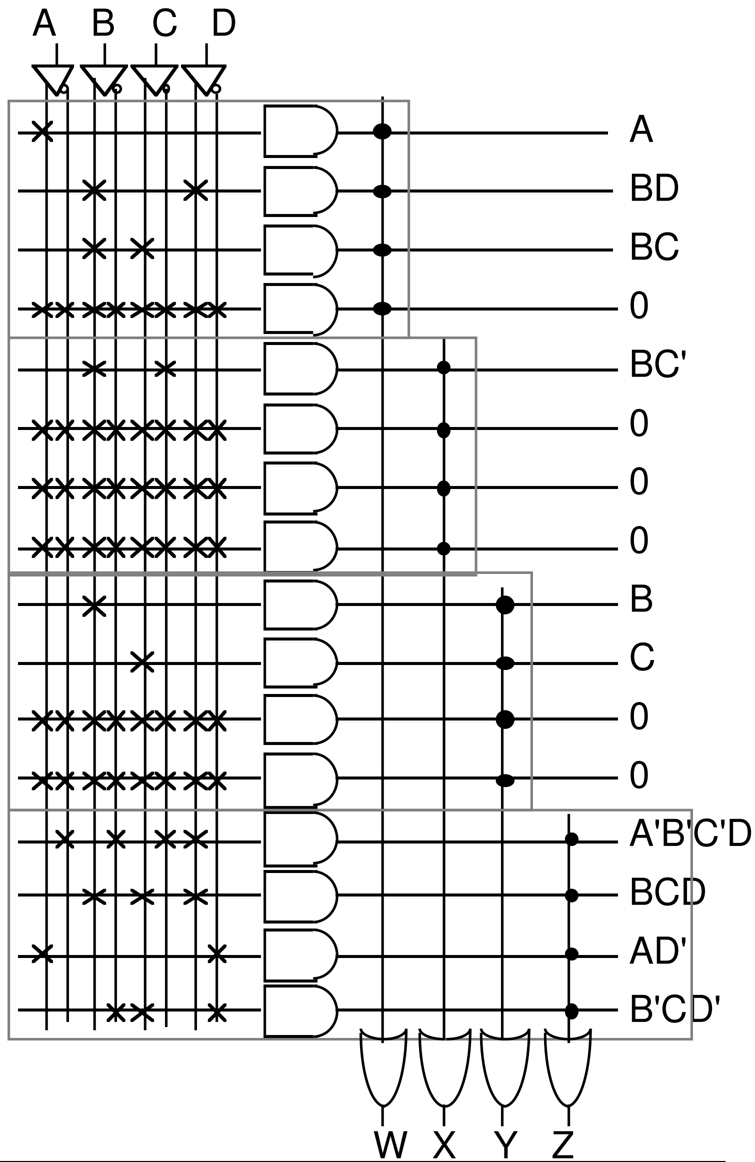
however, much more compact and regular implementation when compared with discrete AND and OR gates

© 1996 Gaetano Borriello

PALs and PLAs: design example (cont'd)

- **Code converter: programmed PAL**

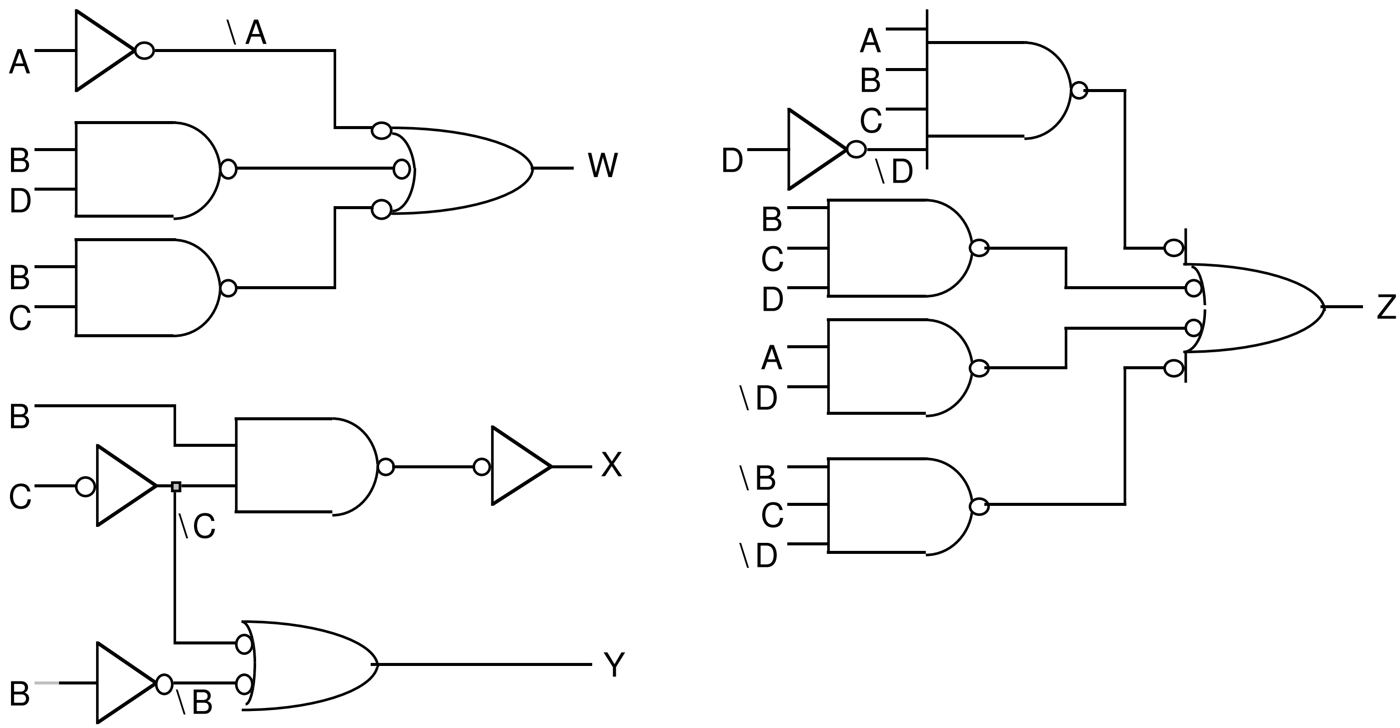
4 product terms
per each OR gate



© 1996 Gaetano Borriello

PALs and PLAs: design example (cont'd)

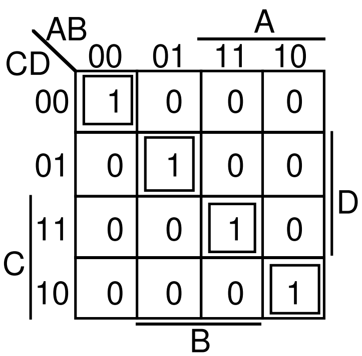
- **Code converter: NAND gate implementation**
  - loss or regularity, harder to understand
  - harder to make changes



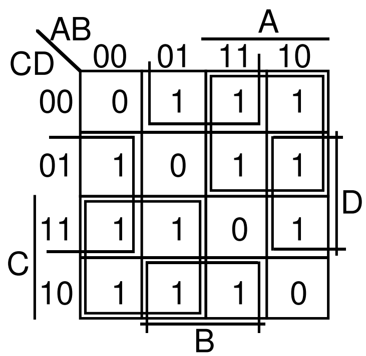
© 1996 Gaetano Borriello

PALs and PLAs: another design example

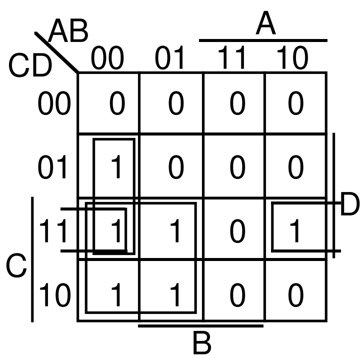
- **Magnitude comparator**



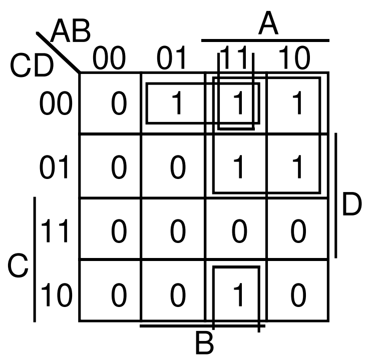
K-map for EQ



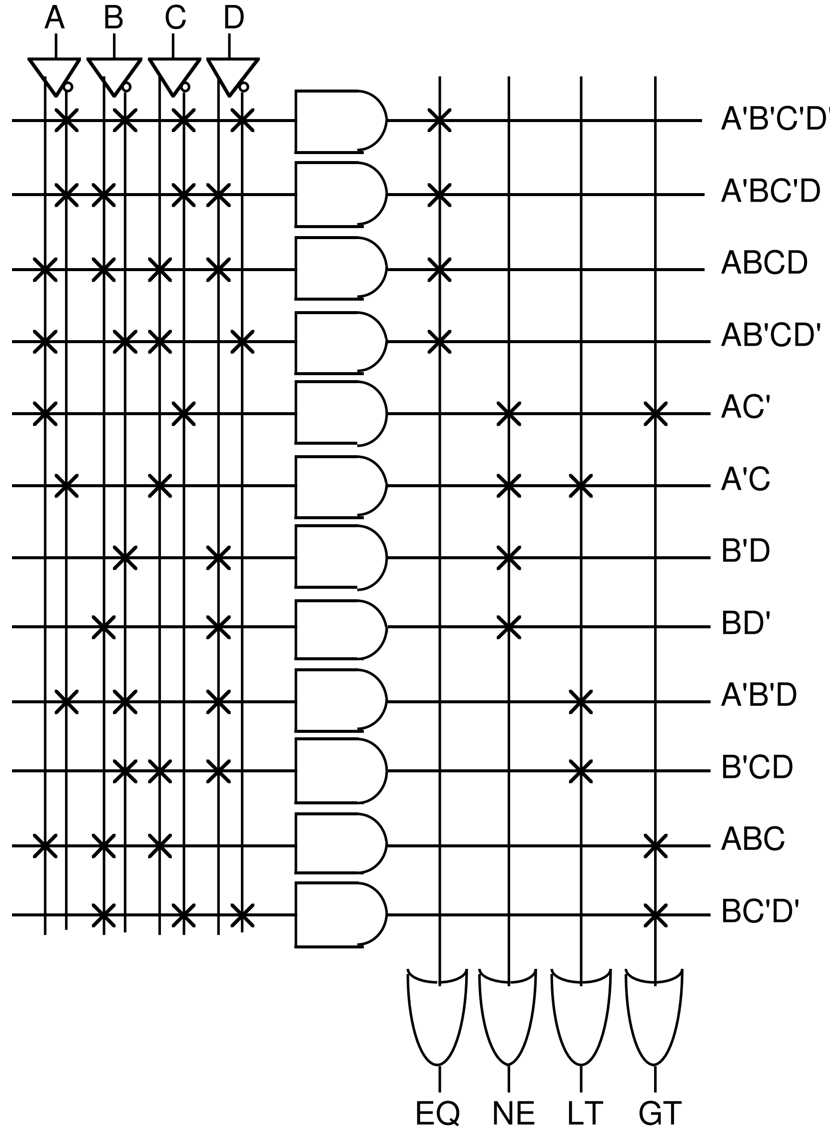
K-map for NE



K-map for LT



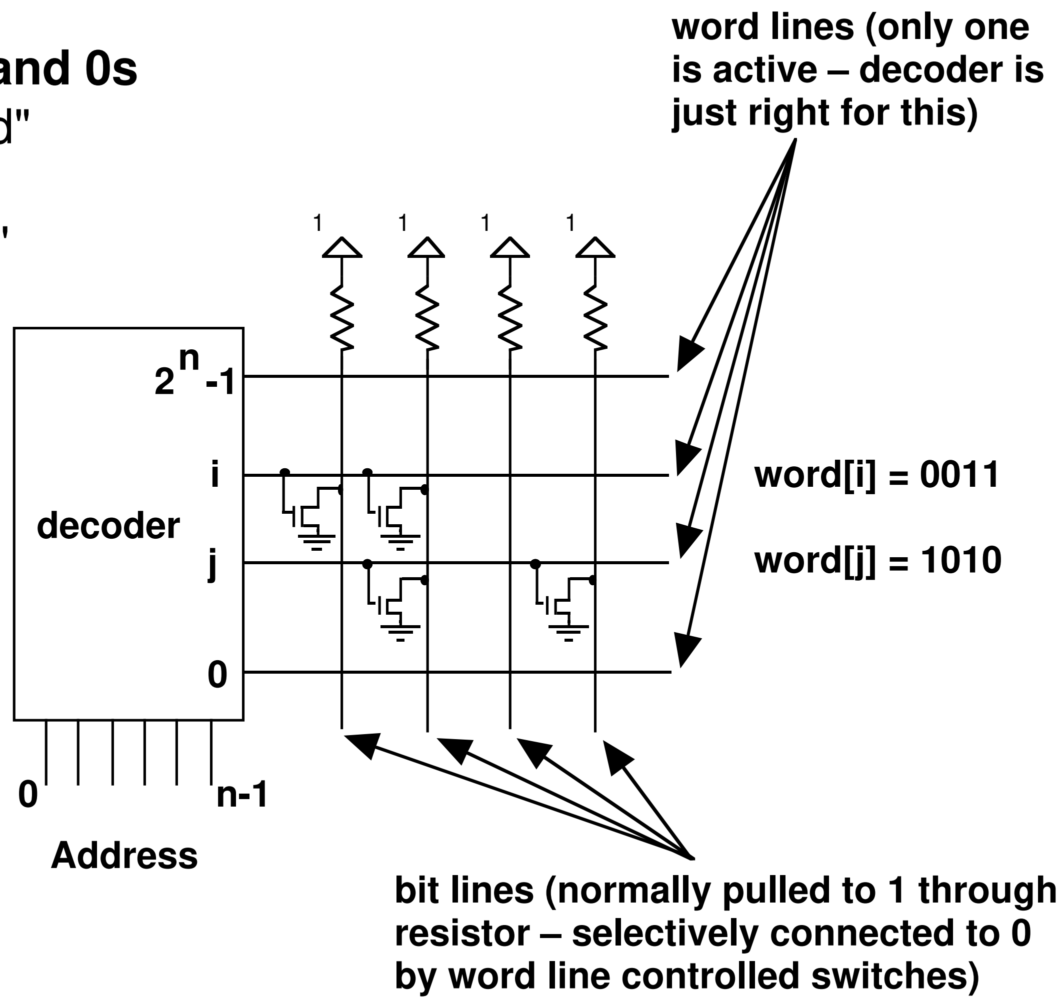
K-map for GT



© 1996 Gaetano Borriello

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



internal organization

© 1996 Gaetano Borriello

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

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

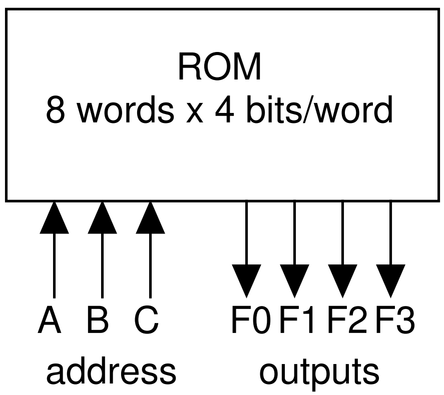
$$F1 = A'\,B'\,C + A'\,B\,C' + A\,B\,C$$

$$F2 = A'\,B'\,C' + A'\,B'\,C + A\,B'\,C'$$

$$F3 = A'\,B\,C + A\,B'\,C' + A\,B\,C'$$

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

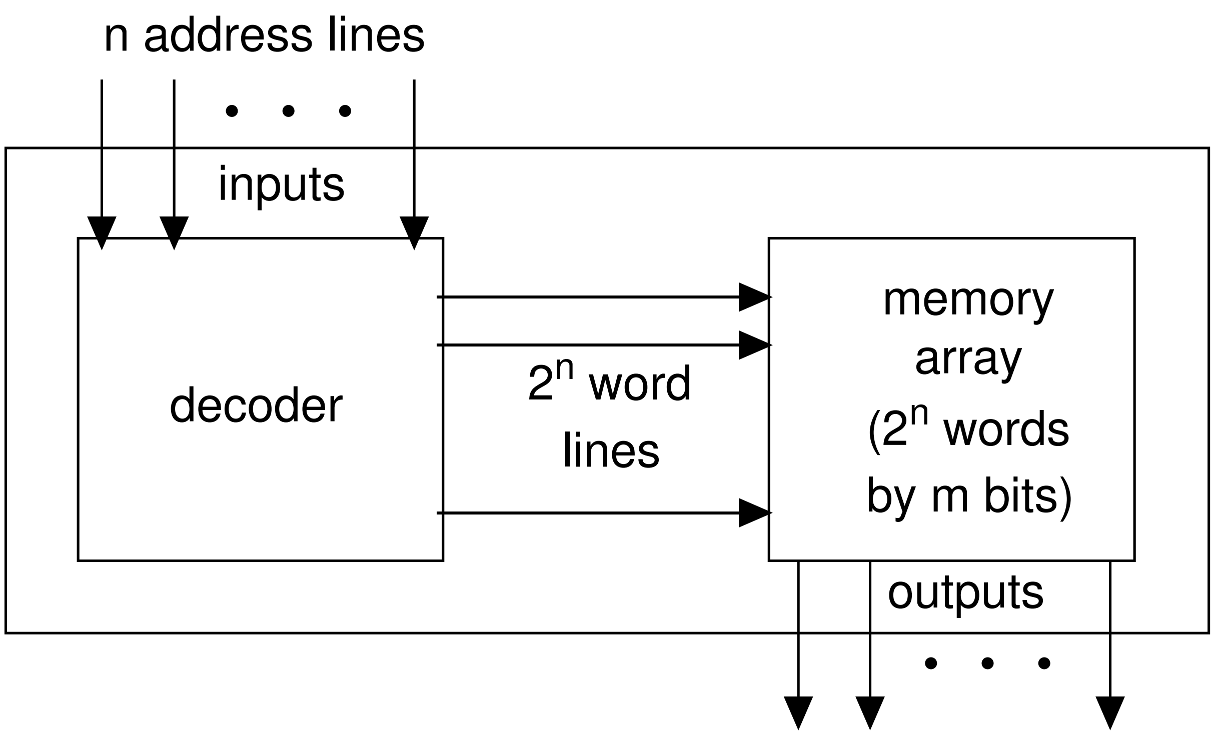


block diagram

© 1996 Gaetano Borriello

ROM structure

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



© 1996 Gaetano Borriello

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 problems: size doubles for each additional input
can't exploit don't cares

- **PLA approach advantageous when**
  - (1) design tools are available for multi-output minimization
  - (2) there are relatively few unique minterm combinations
  - (3) many minterms are shared among the output functions

  - PAL problem: constrained fan-ins on OR plane

© 1996 Gaetano Borriello

Regular logic structures for two-level combinational 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)

© 1996 Gaetano Borriello

Regular logic structures for multi-level combinational 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
  - in 467 you'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)

- **Use multiple levels of PALs/PLAs/ROMs**
  - output intermediate result
  - make it an input to be used in further logic

© 1996 Gaetano Borriello

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

© 1996 Gaetano Borriello