CH-4: Combinational Logic Technologies

Contemporary Logic Design

YONSEI UNIVERSITY

Fall 2016

Combinational Logic Technologies

- Standard gates
  - gate packages
  - cell libraries
- Regular logic
  - multiplexers
  - decoders
- Two-level programmable logic
  - PALs
  - PLAs
  - ROMs

Random Logic

- Transistors quickly integrated into logic gates (1960s)
- Catalog of common gates (1970s)
  - Texas Instruments Logic Data Book – the yellow bible
  - all common packages listed and characterized (delays, power)
  - typical packages:
    - in 14-pin IC: 6-inverters, 4 NAND gates, 4 XOR gates
- Today, very few parts are still in use
- However, parts libraries exist for chip design
  - designers reuse already characterized logic gates on chips
  - same reasons as before
  - difference is that the parts don't exist in physical inventory – created as needed

Random Logic

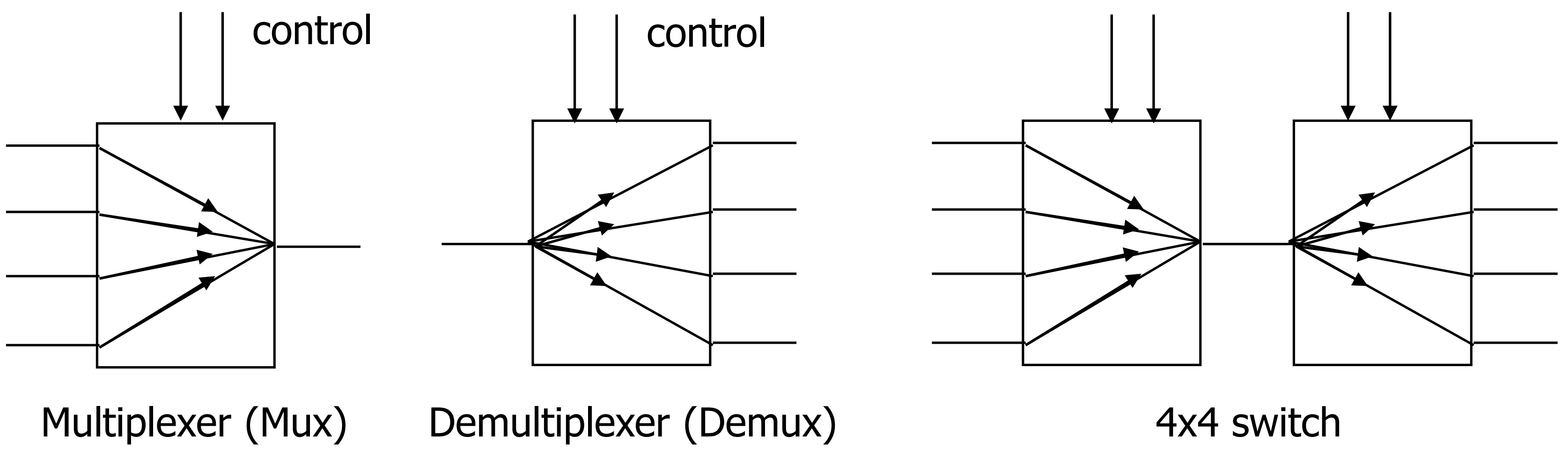
- Too hard to figure out exactly what gates to use
  - map from logic to NAND/NOR networks
  - determine <u>minimum number of packages</u>
- Changes are difficult to realize
  - need to rewire parts
  - may need new parts
  - design with spares (few extra inverters and gates on every board)

Regular Logic

- Need to make design faster
- Need to make engineering changes easier to make
- Simpler for designers to understand and map to functionality
  - harder to think in terms of specific gates
  - better to think in terms of a large multi-purpose block

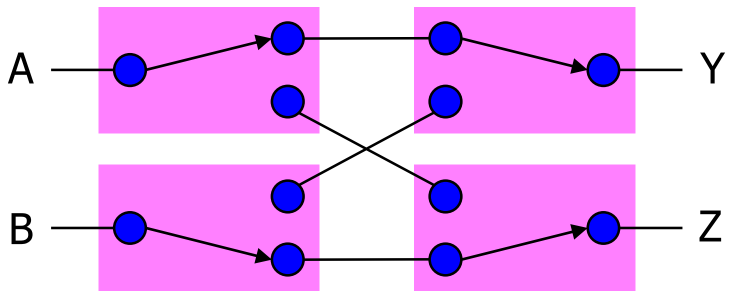
Making Connections

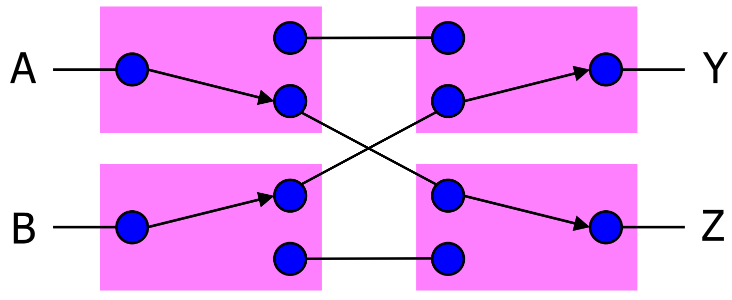
- Direct point-to-point connections between gates
  - wires we've seen so far
- Route one of many inputs to a single output --- *multiplexer*
- Route a single input to one of many outputs --- *demultiplexer*



Mux and Demux

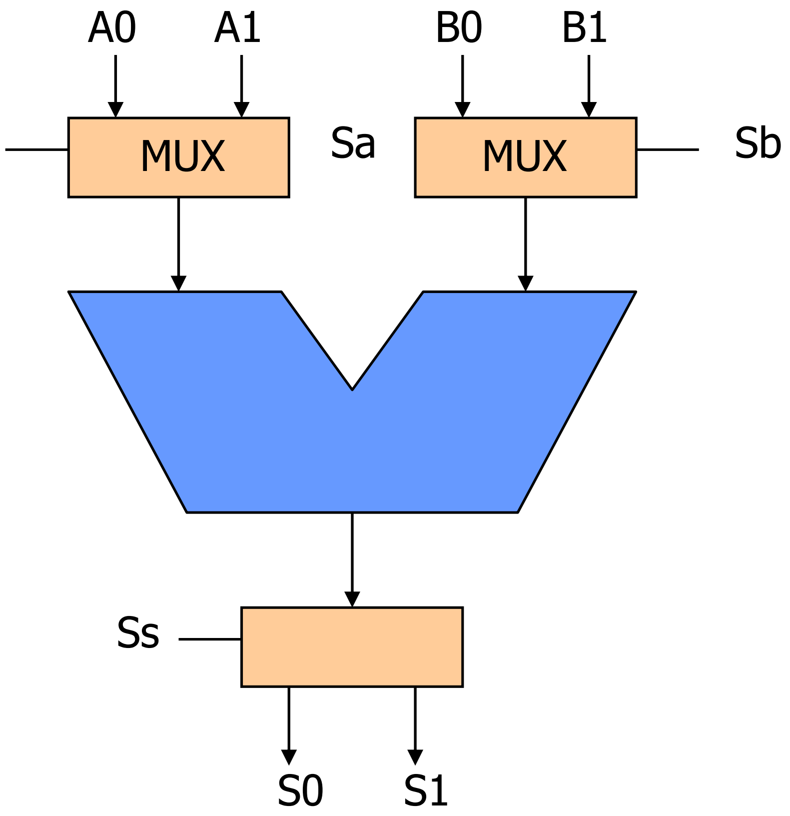
- Switch implementation of multiplexers and demultiplexers
  - can be composed to *make arbitrary size* switching networks
  - used to implement multiple-source/multiple-destination interconnections





Mux and Demux

- Uses of multiplexers/demultiplexers in multi-point connections



multiple input sources

multiple output destinations

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

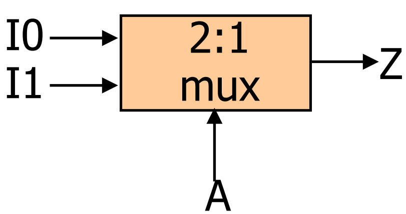
logical form

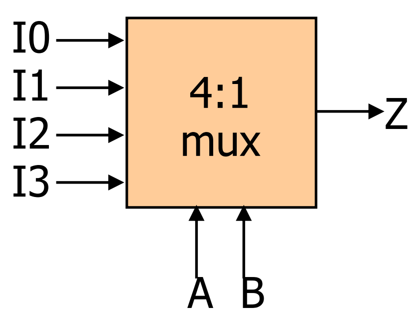
two alternative forms
for a 2:1 Mux truth table

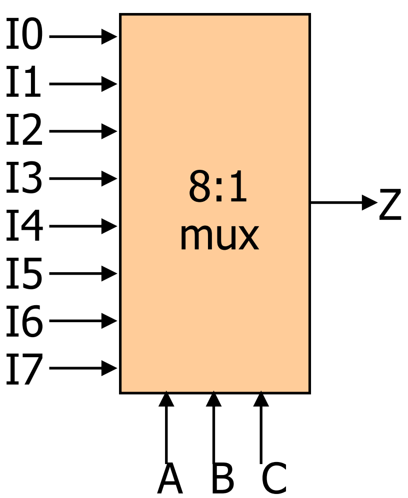
Multiplexers/Selectors

- 2:1 mux: $Z = A'I_0 + AI_1$
- 4:1 mux: $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$
- 8:1 mux: $Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$

- In general: $Z = \Sigma_{k=0}^{2^n-1}(m_k I_k)$
  - in minterm shorthand form for a 2^n:1 Mux

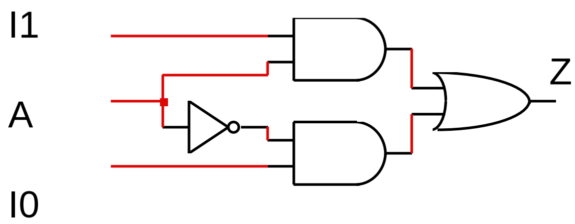


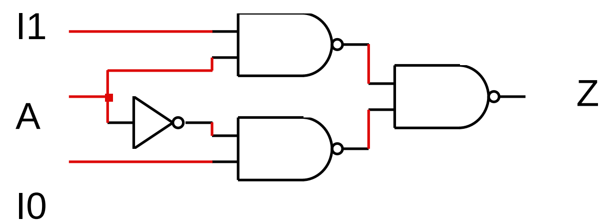




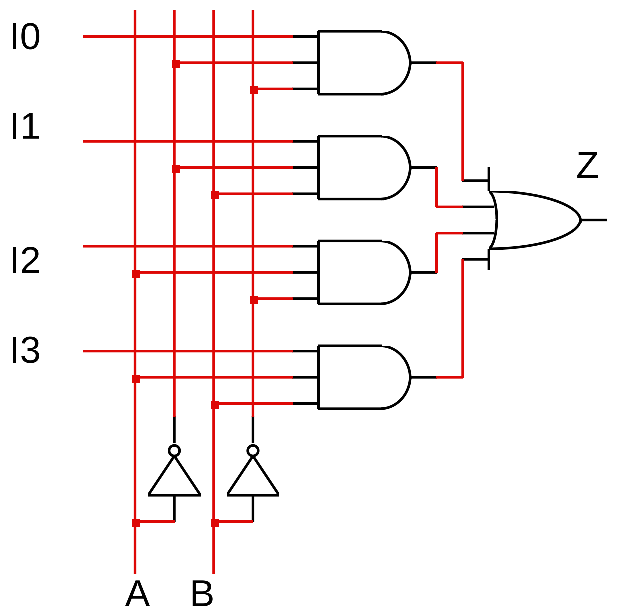
Gate Level Implementation of Muxes

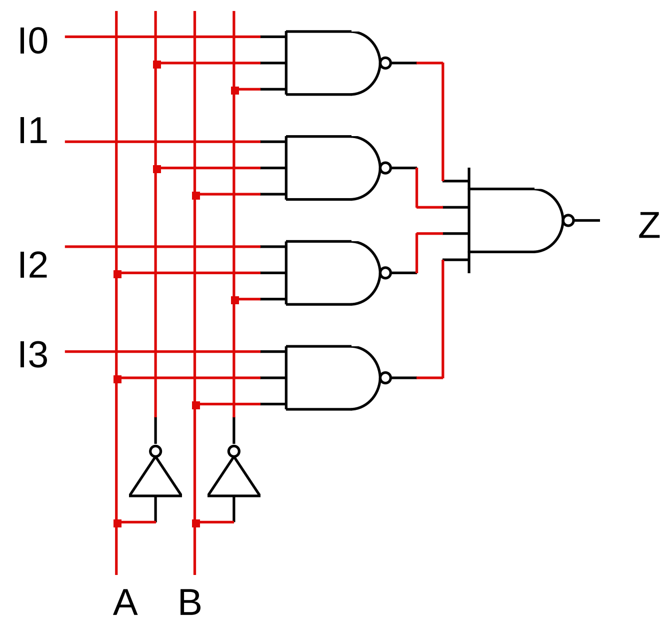
- 2:1 mux





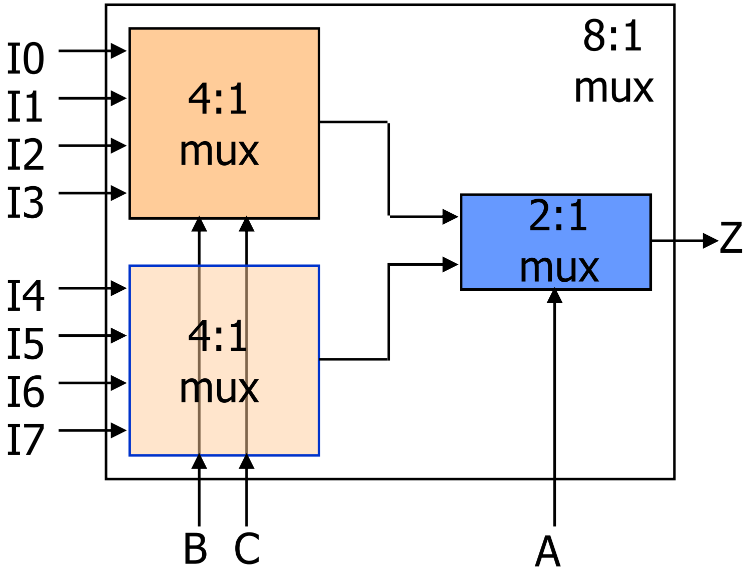
- 4:1 mux





Cascading Multiplexers

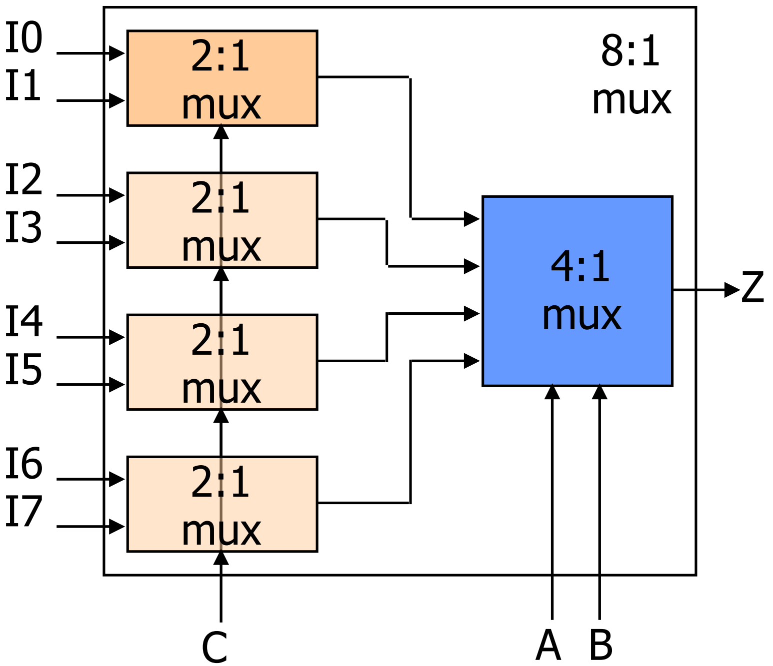
- Large multiplexers can be made *by cascading smaller ones*



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

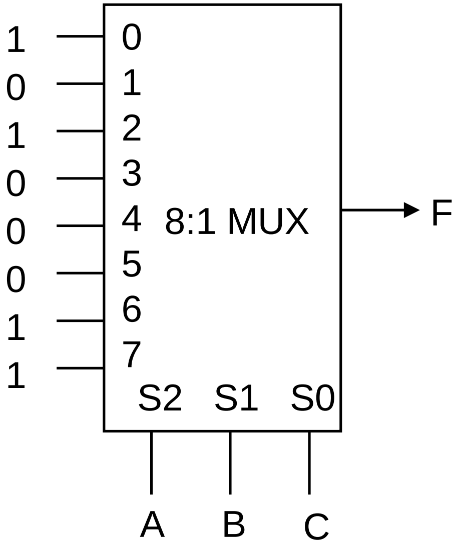
control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation



Multiplexers as General-Purpose Logic

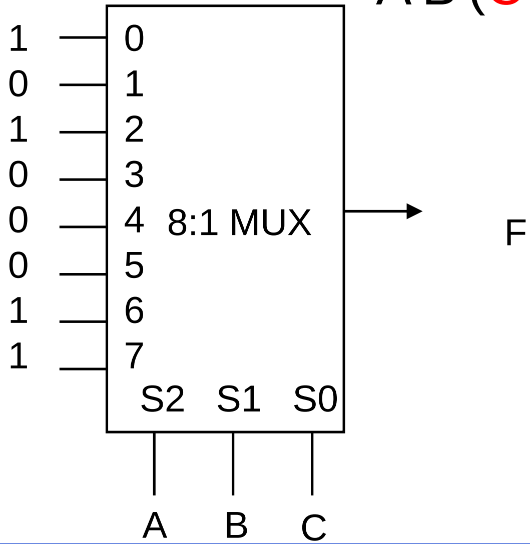
- A 2^n:1 multiplexer can implement *any function of n variables*
  - with the *variables* used as control inputs and
  - the data inputs tied to 0 or 1
  - in essence, a lookup table
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'C'(1) + A'B'C(0)
    + A'BC'(1)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



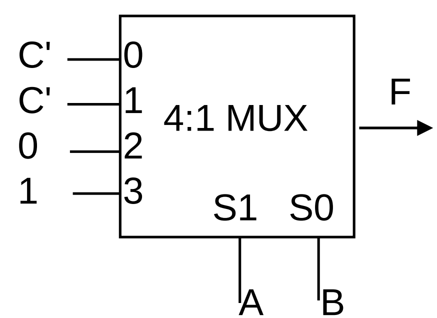
$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$$

Multiplexers as General-Purpose Logic

- 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)

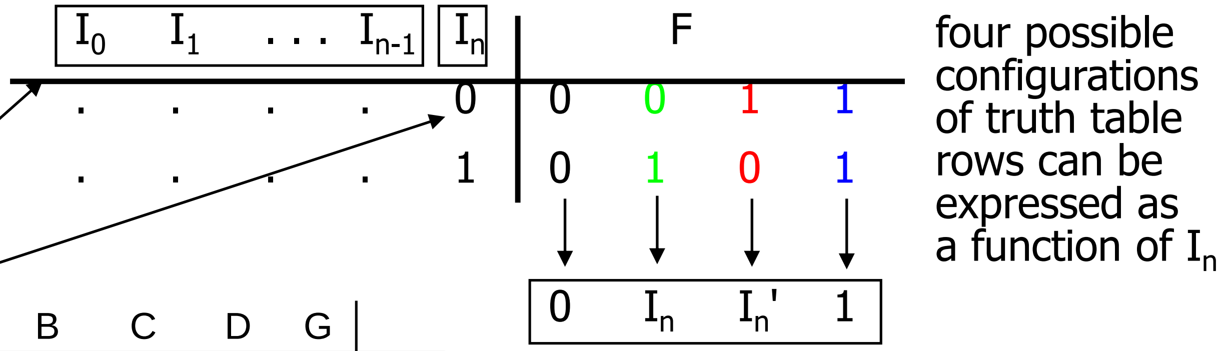


A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



Multiplexers as General-Purpose Logic

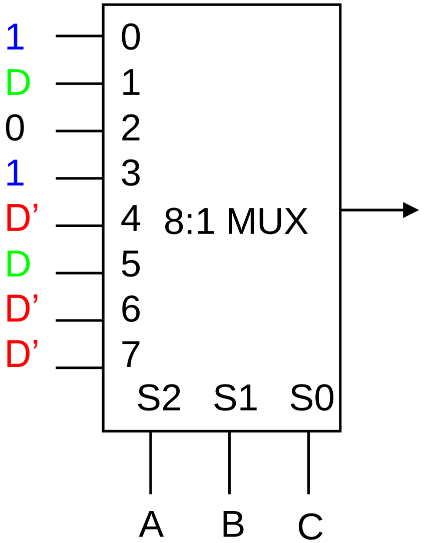
- *Generalization*



- Example: G(A,B,C,D) can be realized by an <u>8:1 MUX</u>

A	B	C	D	G	
0	0	0	0	1	1
0	0	0	1	1	
0	0	1	0	0	D
0	0	1	1	1	
0	1	0	0	0	0
0	1	0	1	0	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	1	D'
1	0	0	1	0	
1	0	1	0	0	D
1	0	1	1	1	
1	1	0	0	1	D’
1	1	0	1	0	
1	1	1	0	1	D’
1	1	1	1	0	

choose A,B,C as control variables



Activity

- Realize F = B’CD’ + ABC’ with a 4:1 multiplexer and a minimum of other gates:

Demultiplexers/Decoders

- Decoders/demultiplexers: general concept
  - *single data input*, *n control* inputs, *2ⁿ 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:

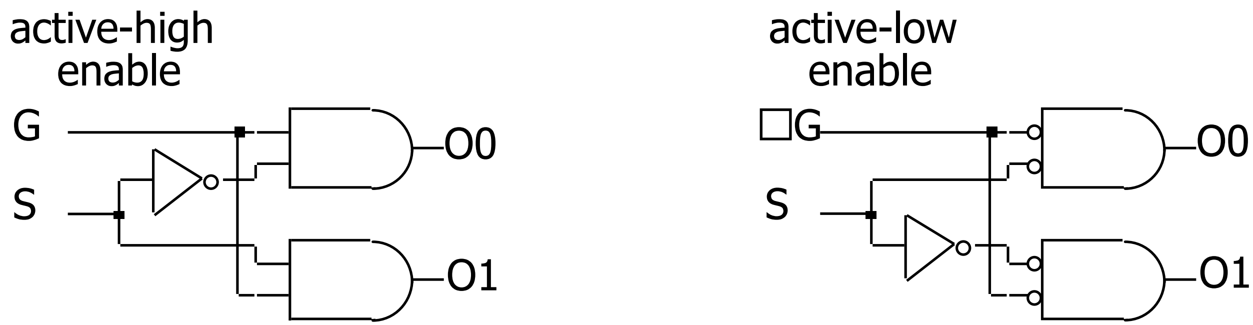
O0 = G • S1′ • S0′
O1 = G • S1′ • S0
O2 = G • S1 • S0′
O3 = G • S1 • S0

3:8 Decoder:

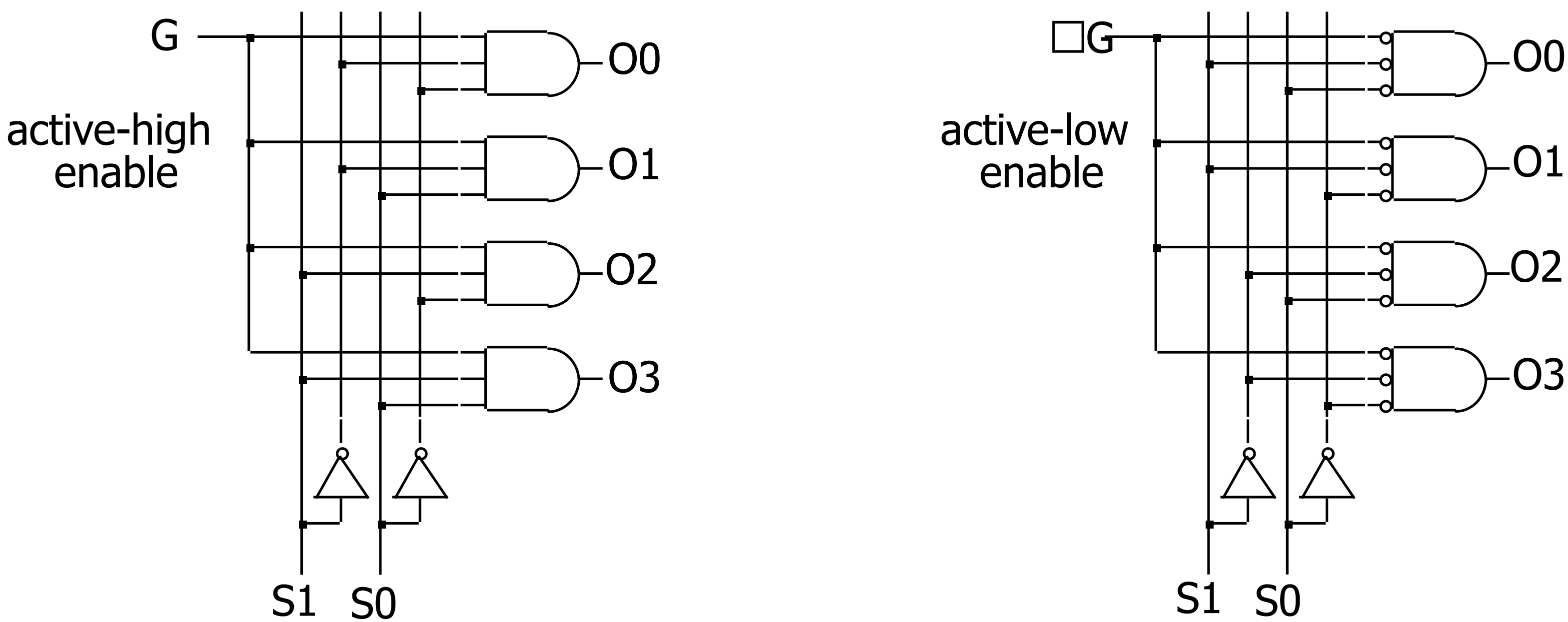
O0 = G • S2′ • S1′ • S0′
O1 = G • S2′ • S1′ • S0
O2 = G • S2′ • S1 • S0′
O3 = G • S2′ • S1 • S0
O4 = G • S2 • S1′ • S0′
O5 = G • S2 • S1′ • S0
O6 = G • S2 • S1 • S0′
O7 = G • S2 • S1 • S0

Gate Level Implementation: Demultiplexers

- 1:2 decoders

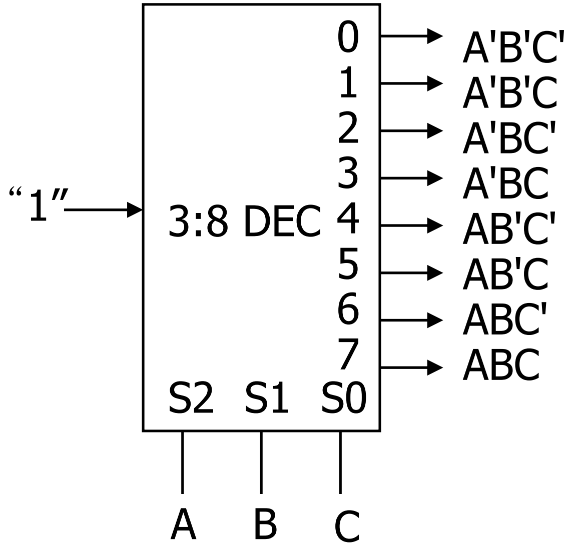


- 2:4 decoders



Demux as General-Purpose Logic

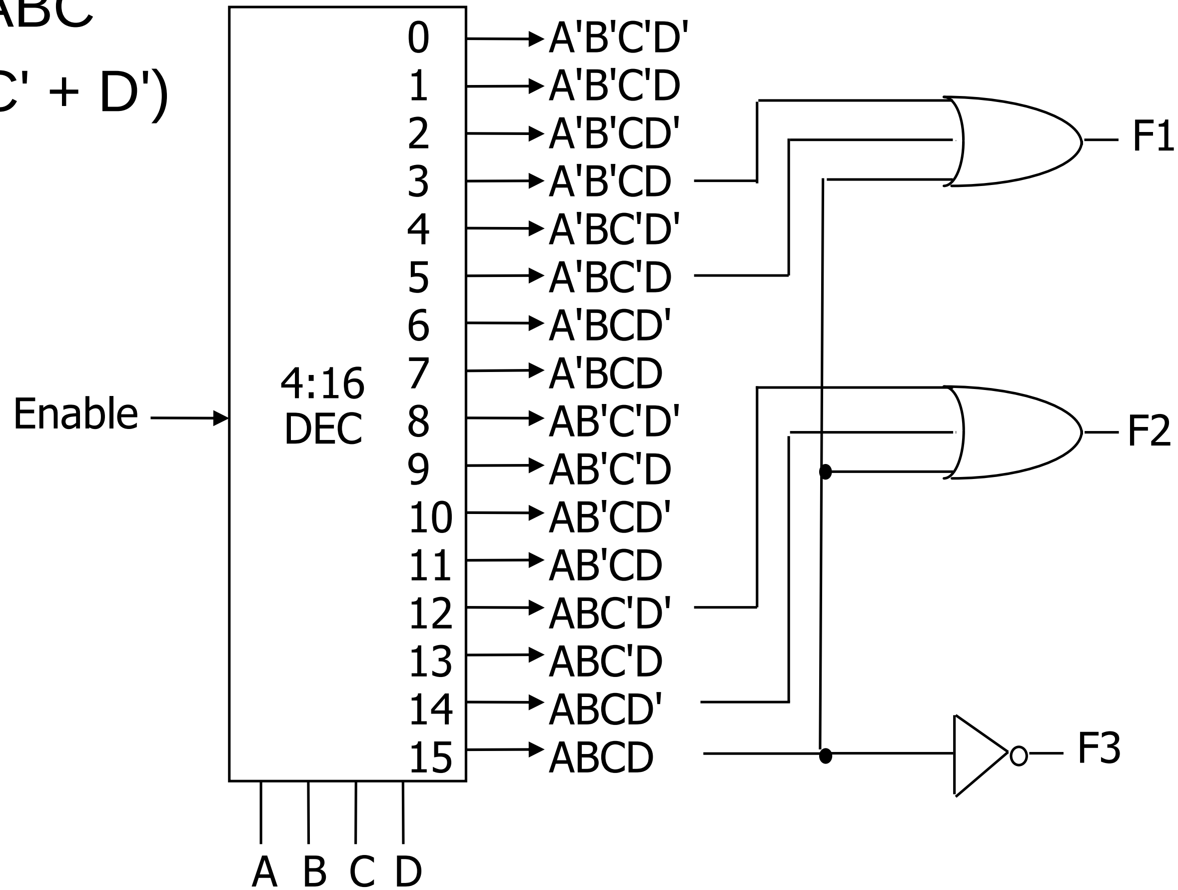
- A *n:2ⁿ decoder* can implement any function of n variables
  - with the variables used as control inputs
  - the enable inputs tied to 1 and
  - the appropriate minterms summed to form the function



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

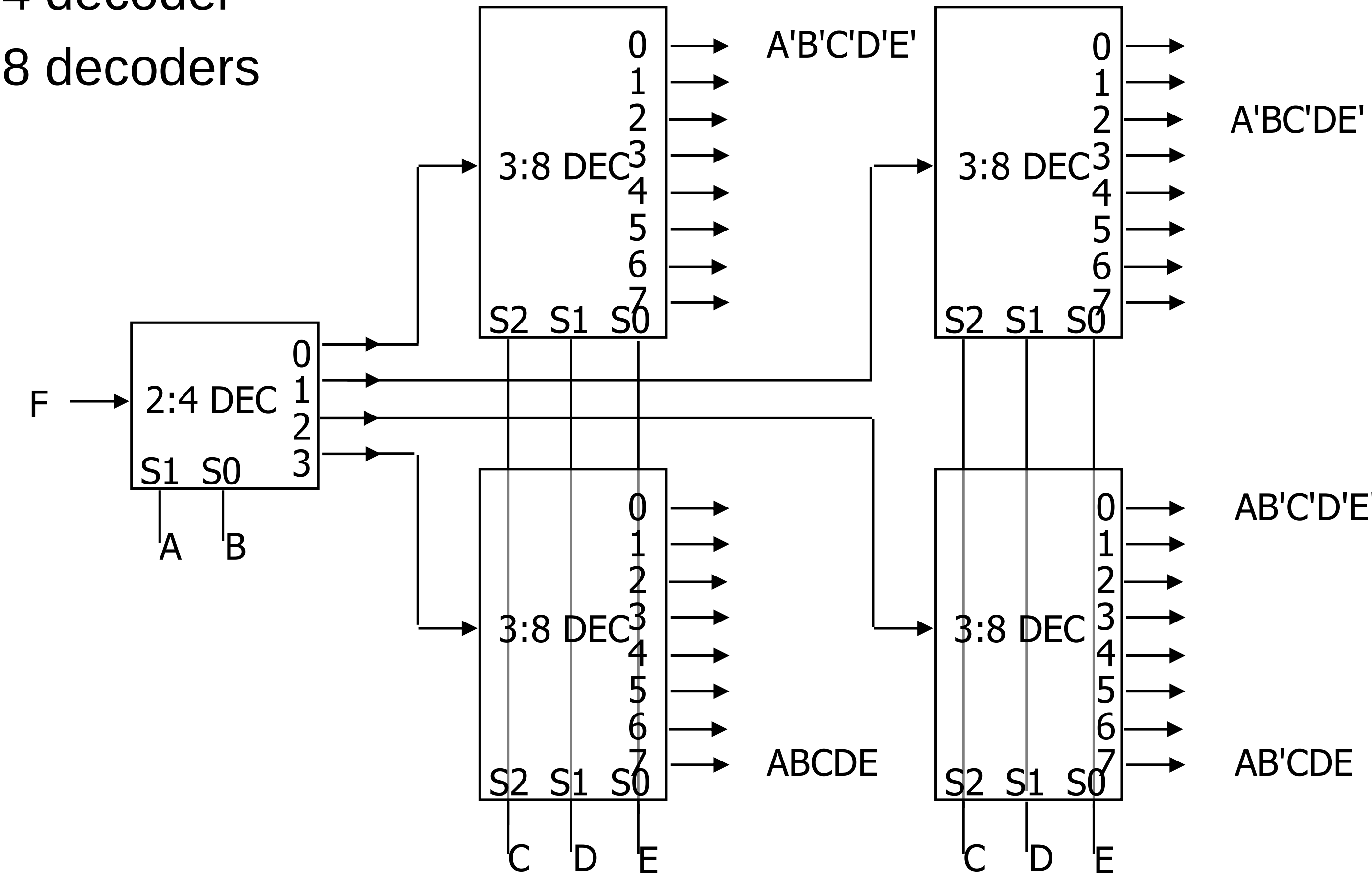
Demux as General-Purpose Logic

- F1 = A'BC'D + A'B'CD + ABCD
- F2 = ABC'D' + ABC
- F3 = (A' + B' + C' + D')



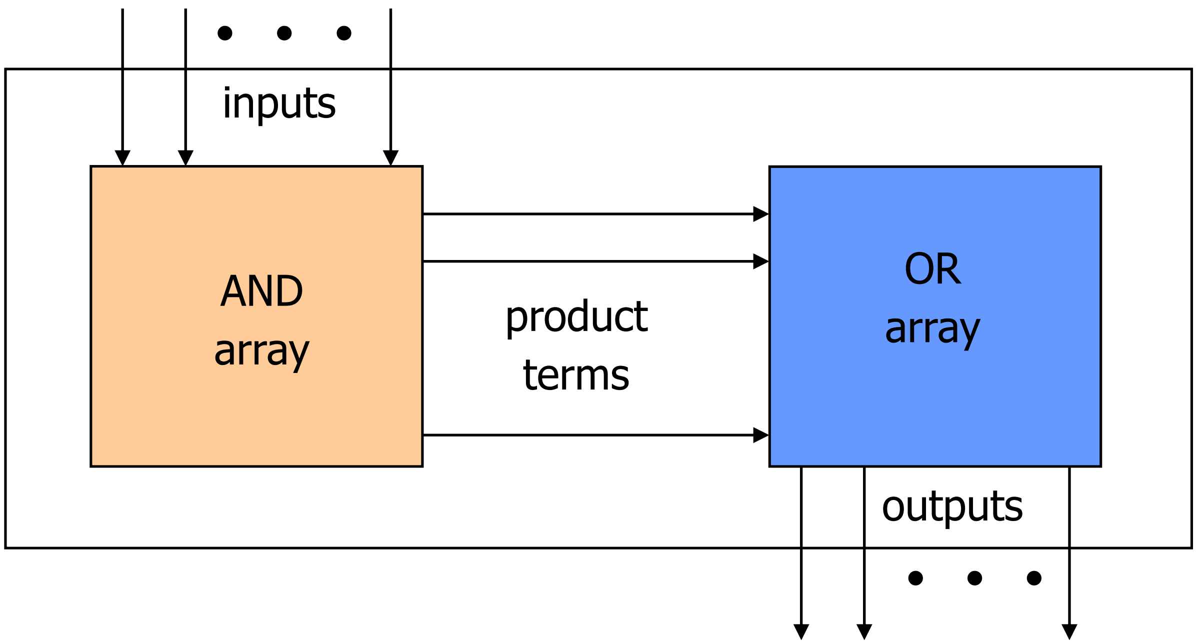
Cascading Decoders

- 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/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:

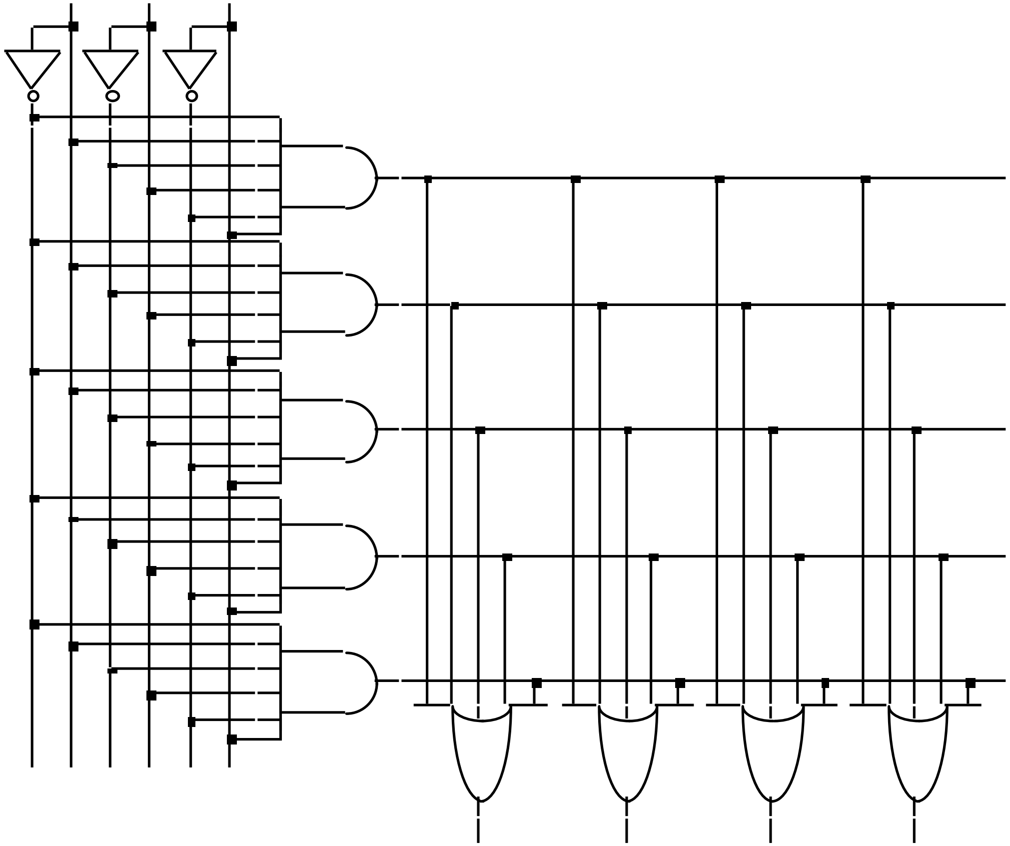
1 = uncomplemented in term
0 = complemented in term
– = does not participate

output side:

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

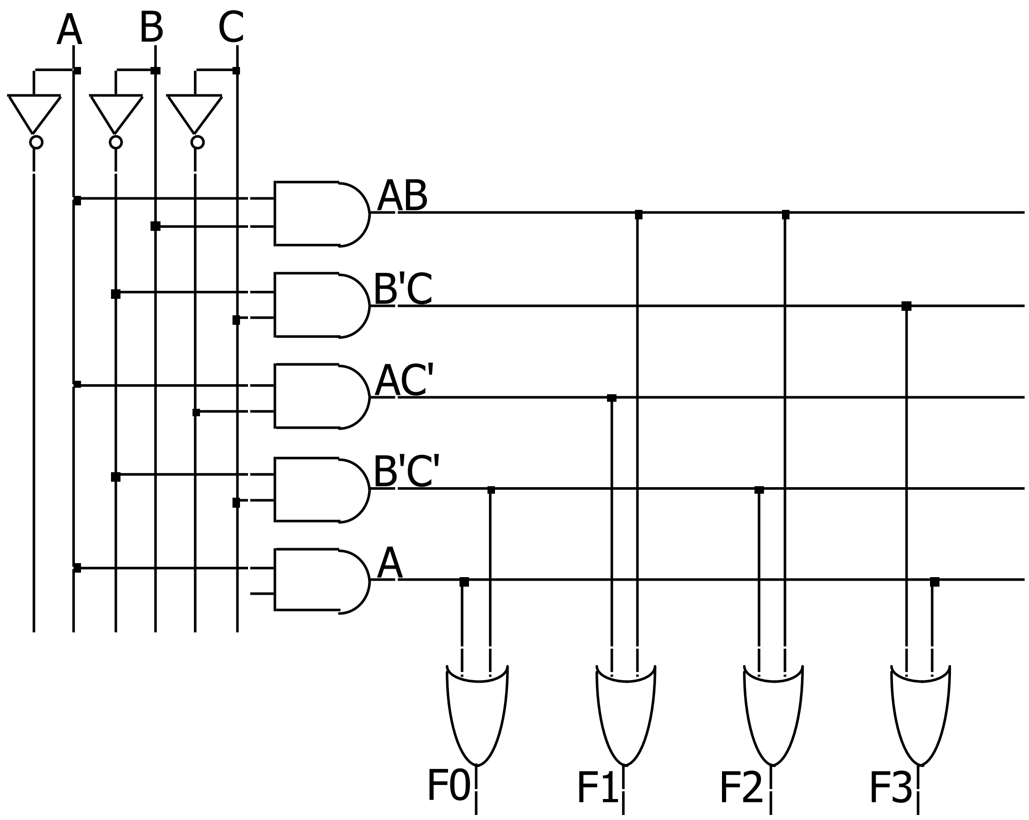
Before Programming

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



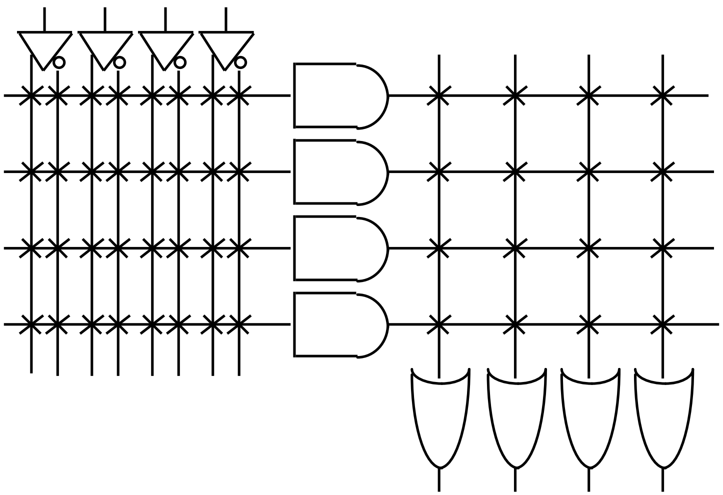
After Programming

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

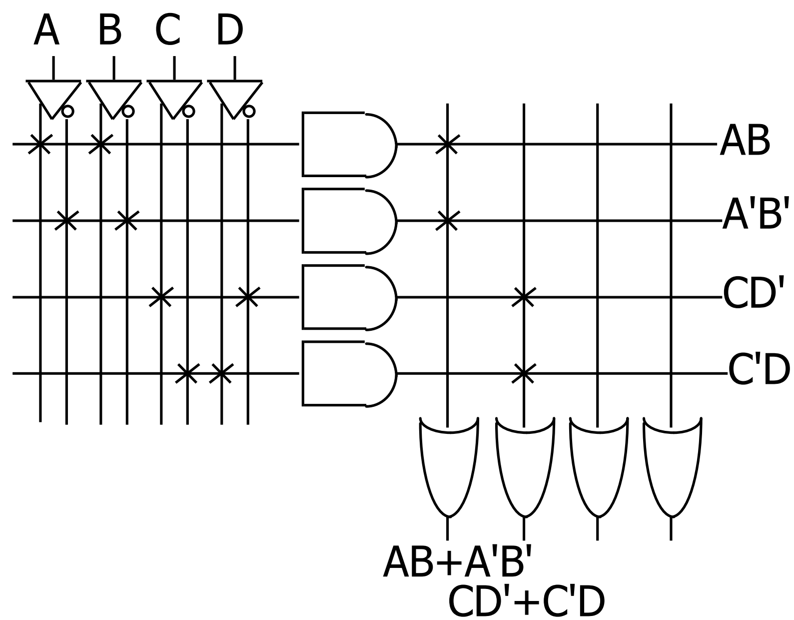


Alternate Representation: High Fan-in Structures

- *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



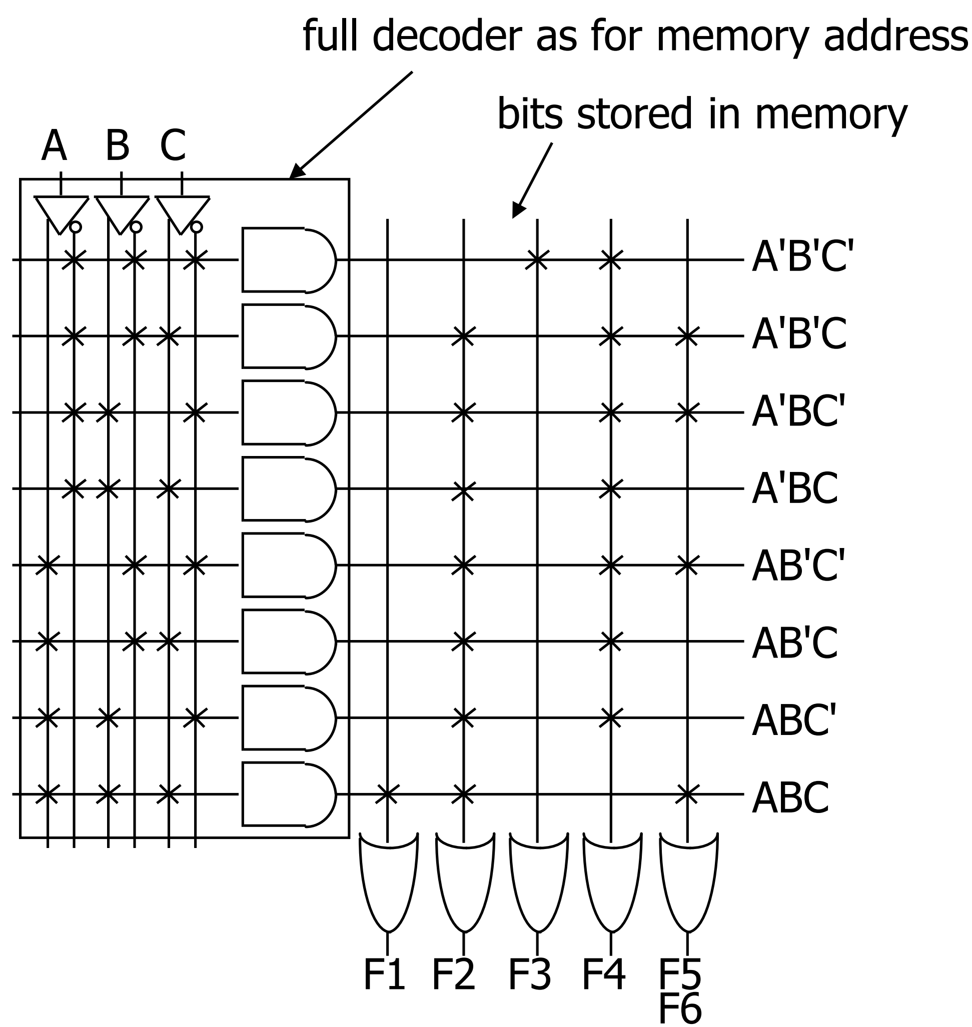
notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D



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

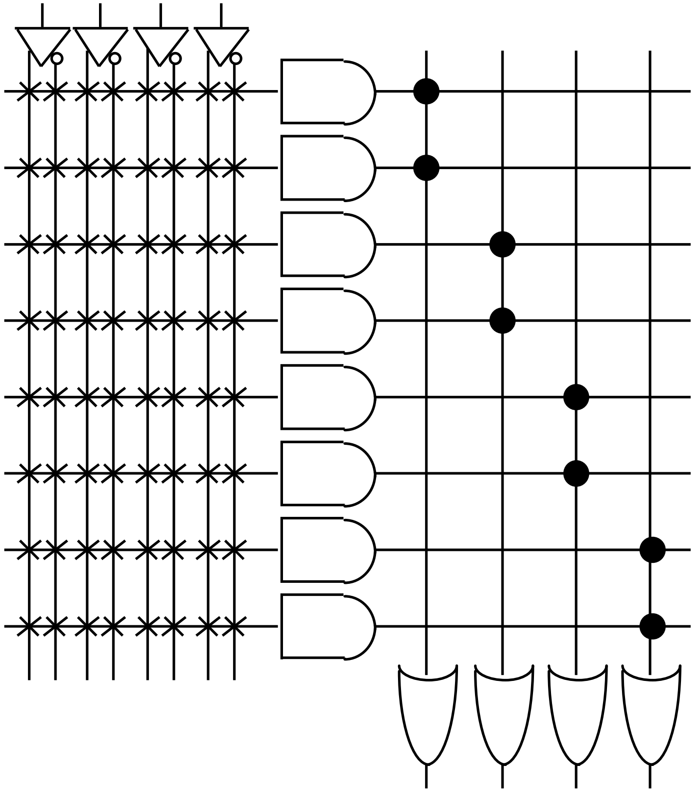
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



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*

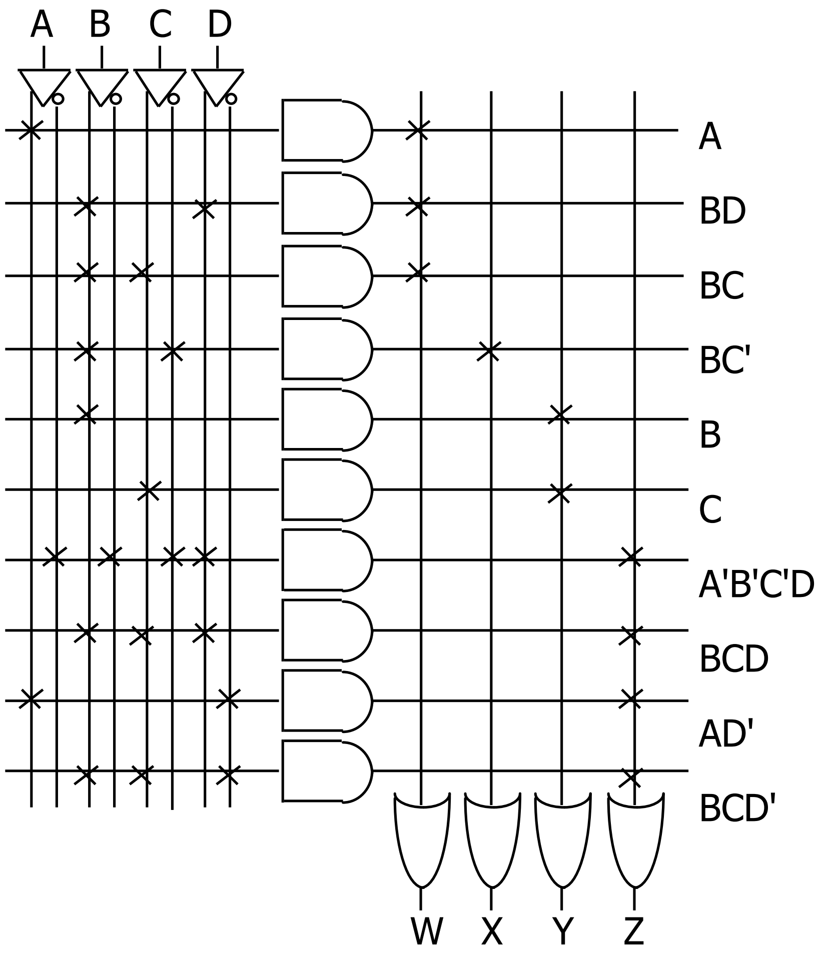
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

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

- *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'

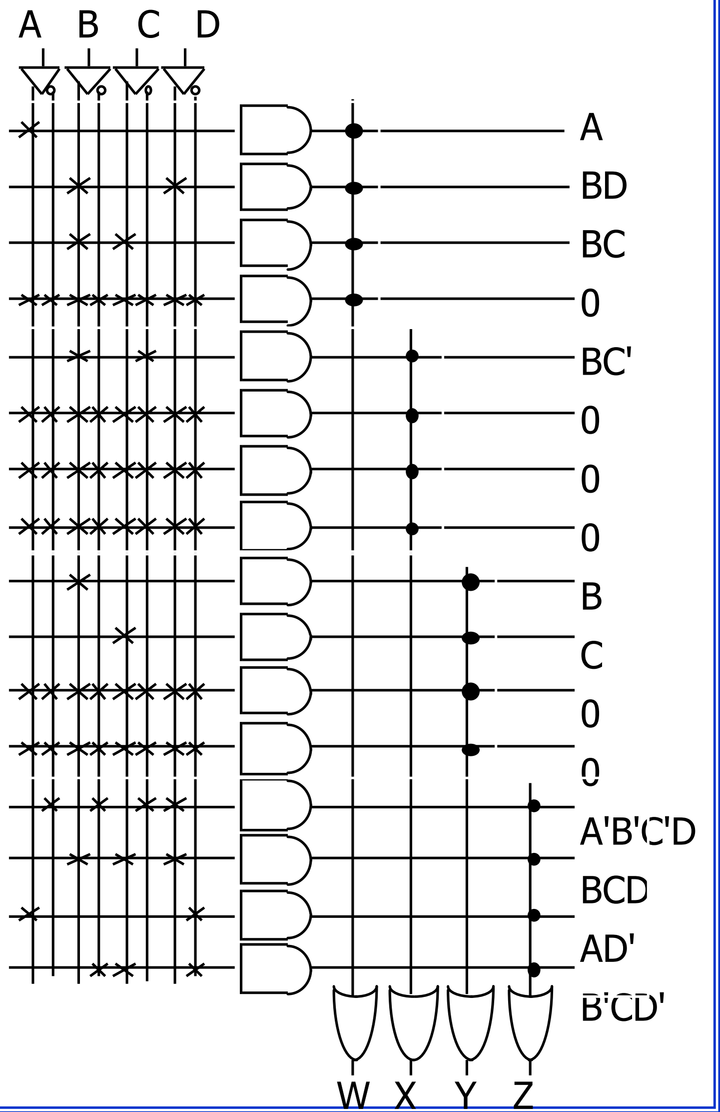
not a particularly good candidate for PAL/PLA implementation since no terms are shared among outputs

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

PALs and PLAs: Design Example

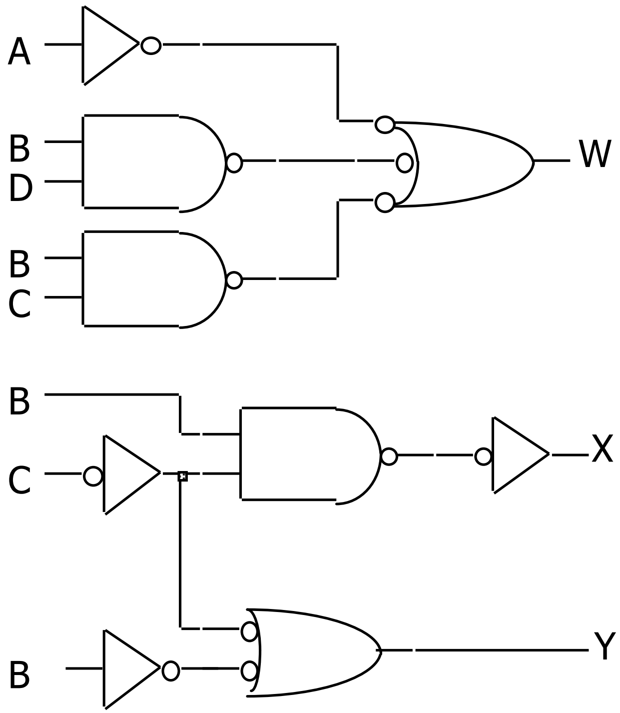
- *Code converter*: programmed PAL

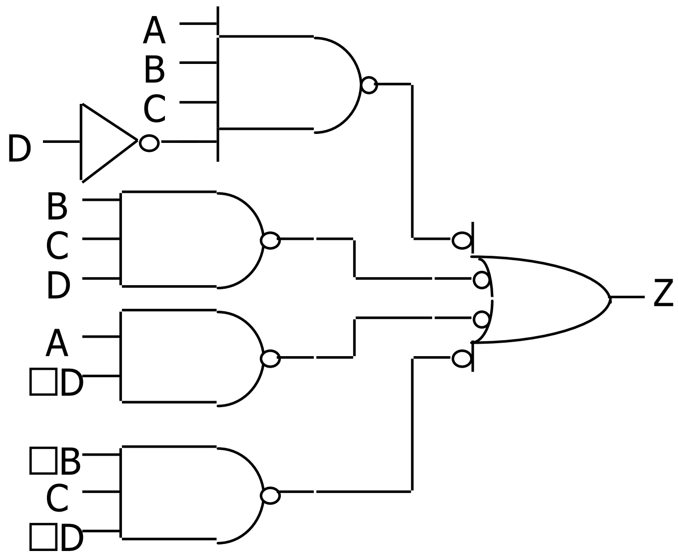
4 product terms
per each OR gate



PALs and PLAs: Design Example

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

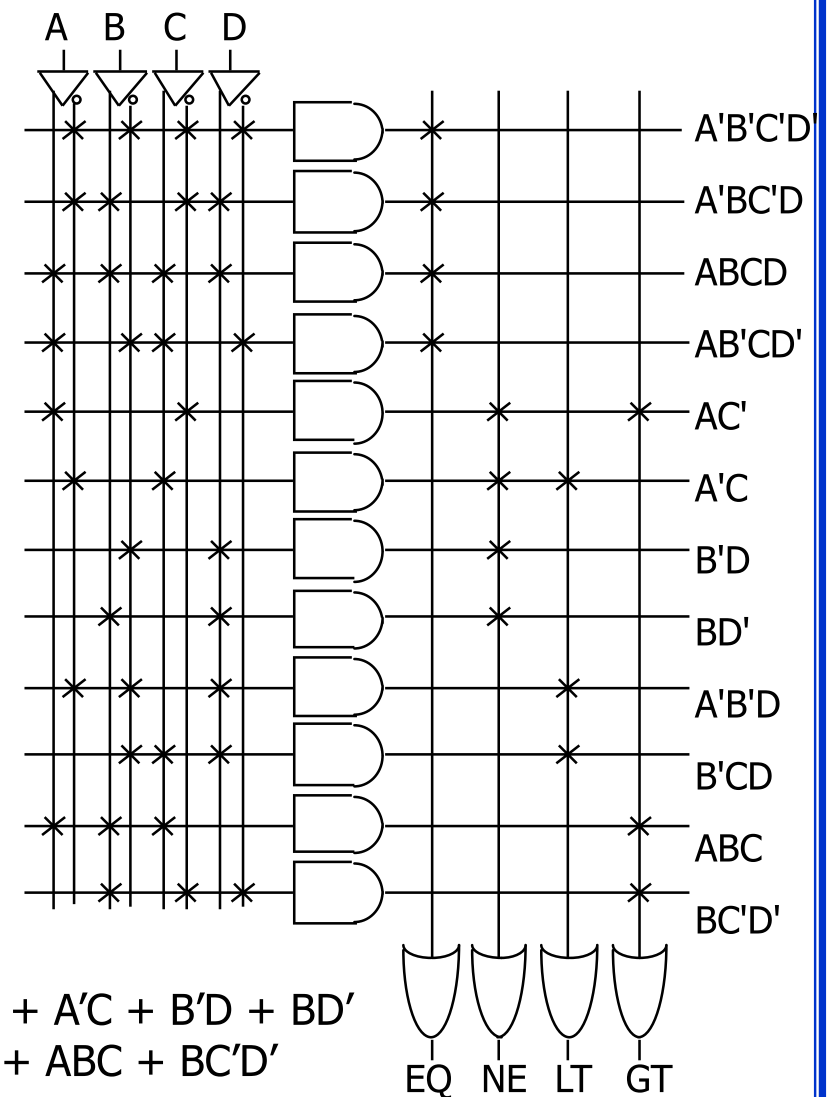




PALs and PLAs: Another Design EX

- *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′

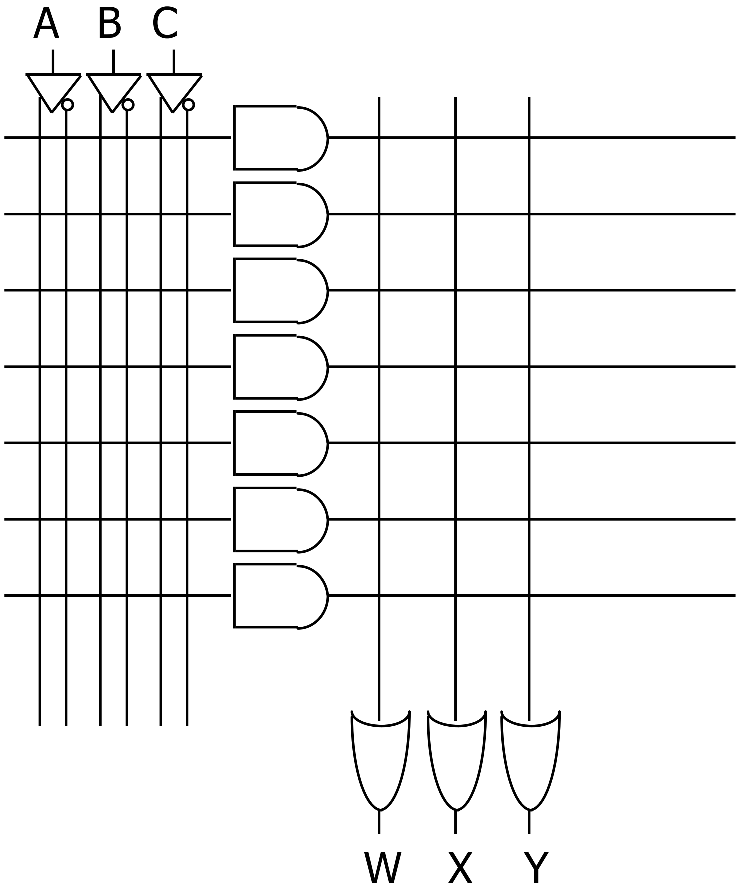
NE = AC′ + A′C + B′D + BD′

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

- 9 terms won’t fit in a 7 term PLA

Tri-State and Open-Collector

Logic States: "0", "1"
Don't Care/Don't Know State: "X" (must be some value in real circuit!)

Third State: "Z" · <u>high impedance</u> · <u>infinite resistance</u>, <u>no connection</u>

Tri-state gates: **output values are "0", "1", and "Z"**
additional input: output enable (OE)

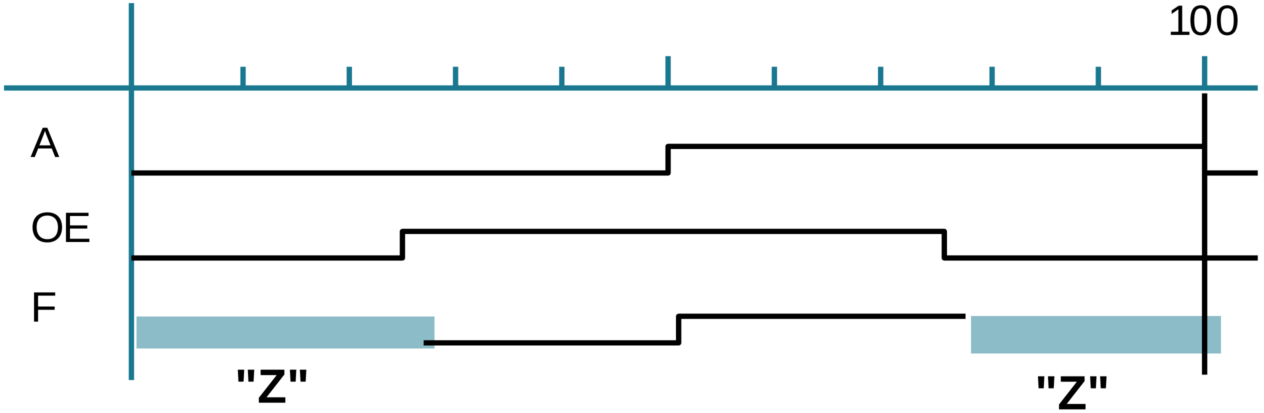
A	OE	F
X	0	Z
0	1	0
1	1	1

When OE is high, this gate is a non-inverting "buffer"

When OE is low, it is as though the gate was *<u>disconnected</u>* <u>from the output</u>!

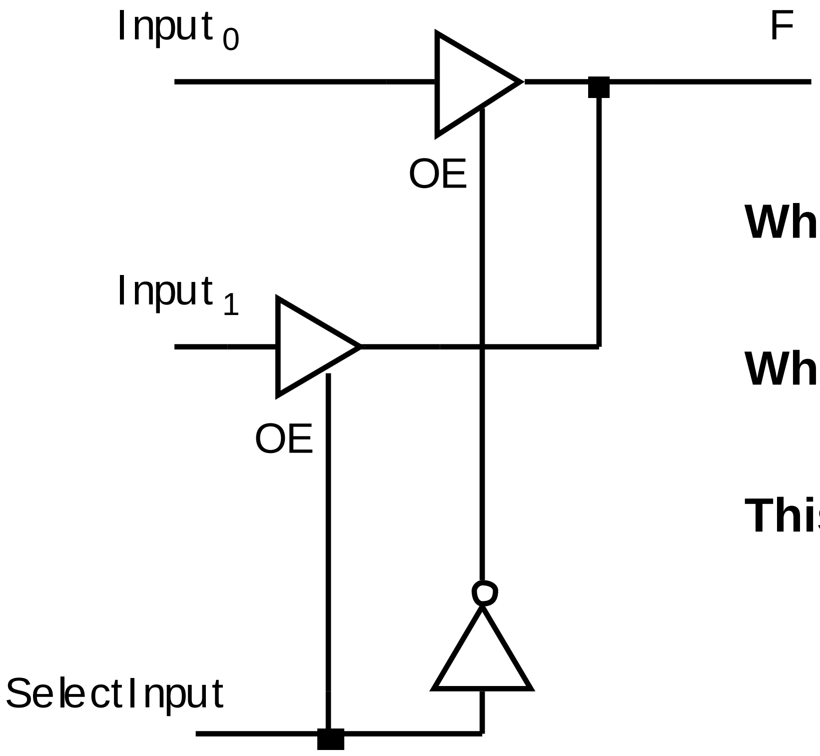
This allows more than one gate to be connected to the same output wire, as long as only one has its output enabled *at the same time*

Non-inverting buffer's timing waveform



Tri-state and Open Collector

Using tri-state gates to implement an economical multiplexer:



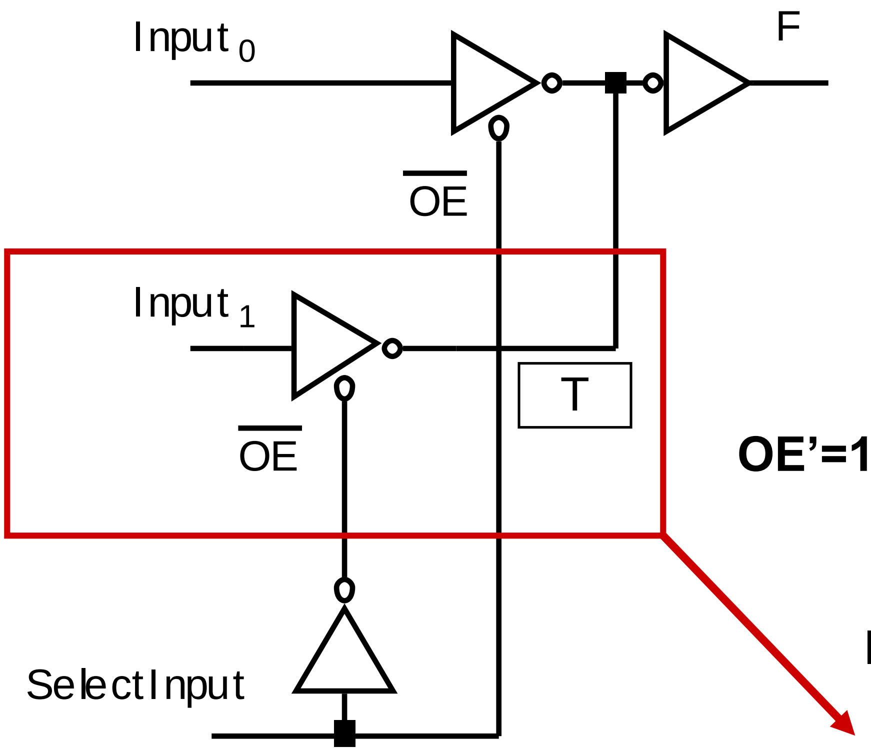
When SelectInput is asserted high
Input1 is connected to F

When SelectInput is driven low
Input0 is connected to F

This is essentially a 2:1 Mux

Tri-state and Open Collector

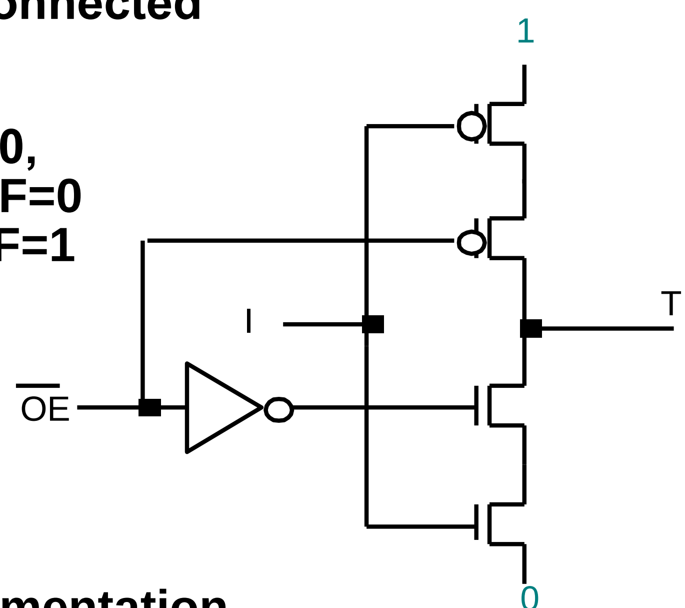
Alternative Tri-state Fragment



<u>Active low tri-state</u> enables plus inverting tri-state buffers

OE'=1, disconnected

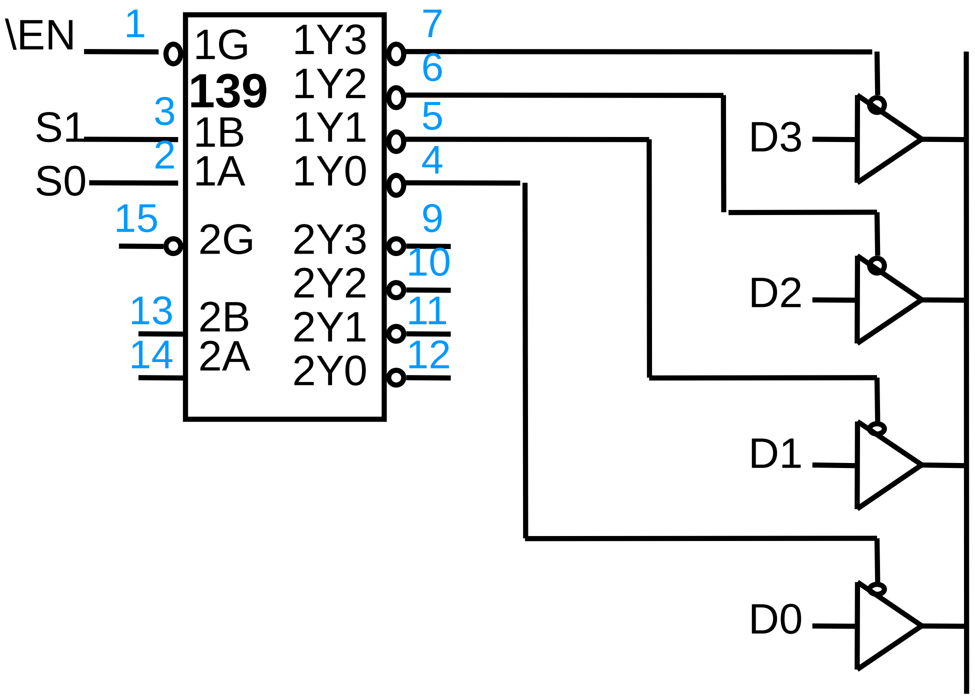
OE'=0,
I=1 → F=0
I=0 → F=1



Switch Level Implementation of inverting tri-state gate

Tri-State and Open Collector

4:1 Multiplexer, Revisited



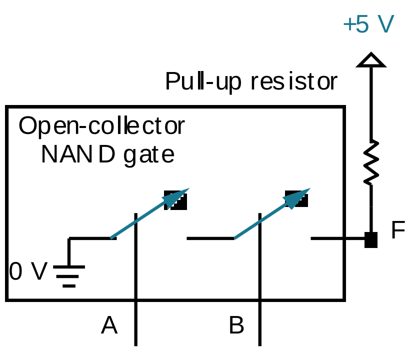
Decoder + 4 tri-state Gates

Open Collector

another way to <u>connect multiple gates to the same output wire</u>

gate <u>only has the ability to pull its output low</u>; it cannot actively drive the wire high

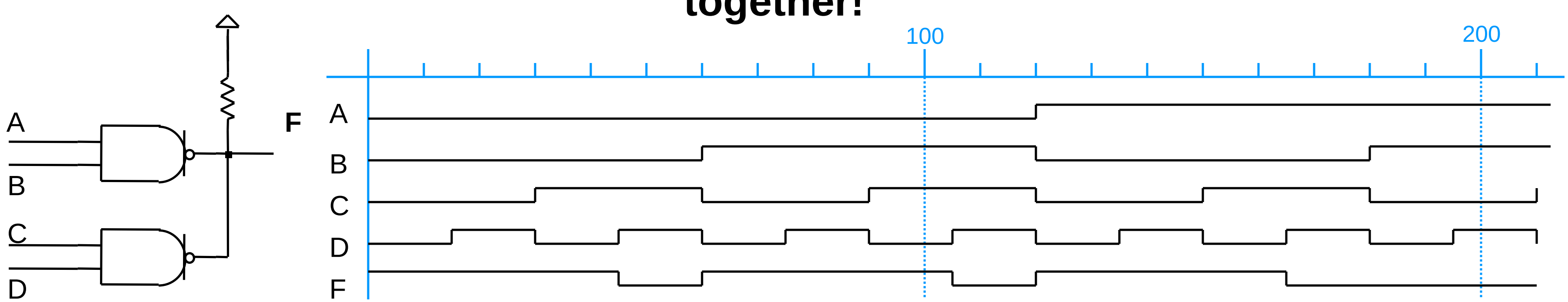
this is done by pulling the wire up to a logic 1 voltage through a resistor



OC NAND gates

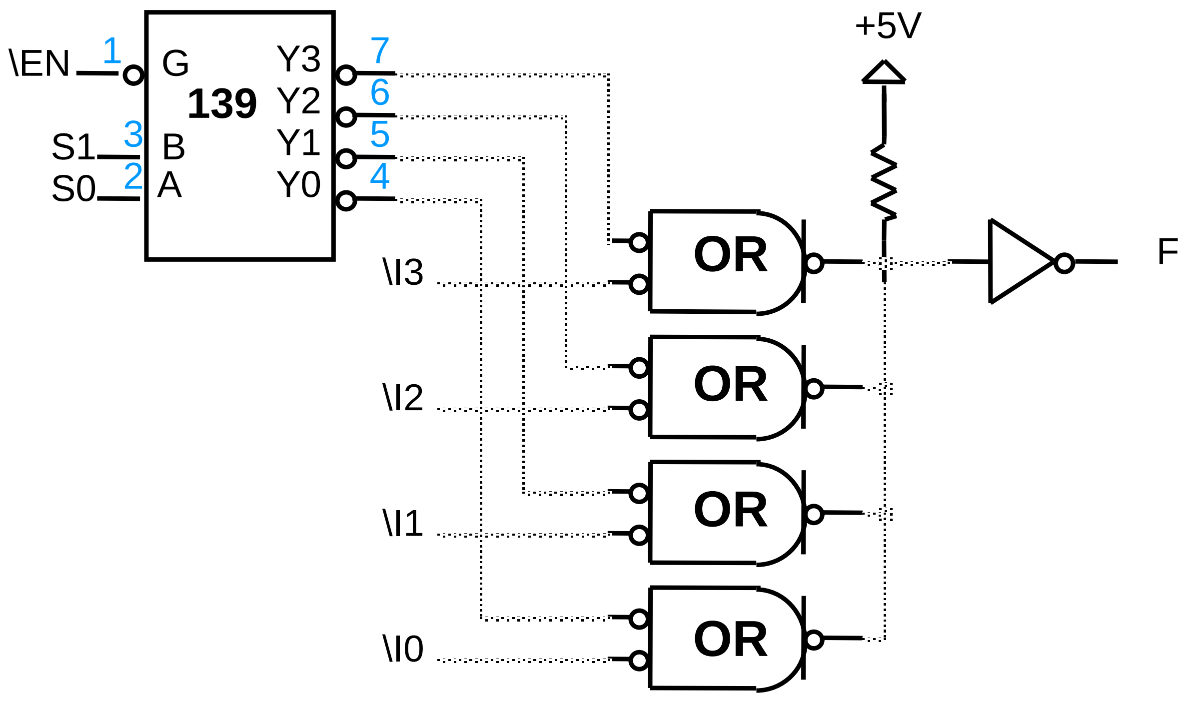
Wired AND:

If A and B are "1", output is actively pulled low
if C and D are "1", output is actively pulled low
if one gate is low, the other high, then low wins
if both gates are "1", the output floats, pulled high by resistor
Hence, the two NAND functions are <u>*AND'd*</u> together!



Open Collector

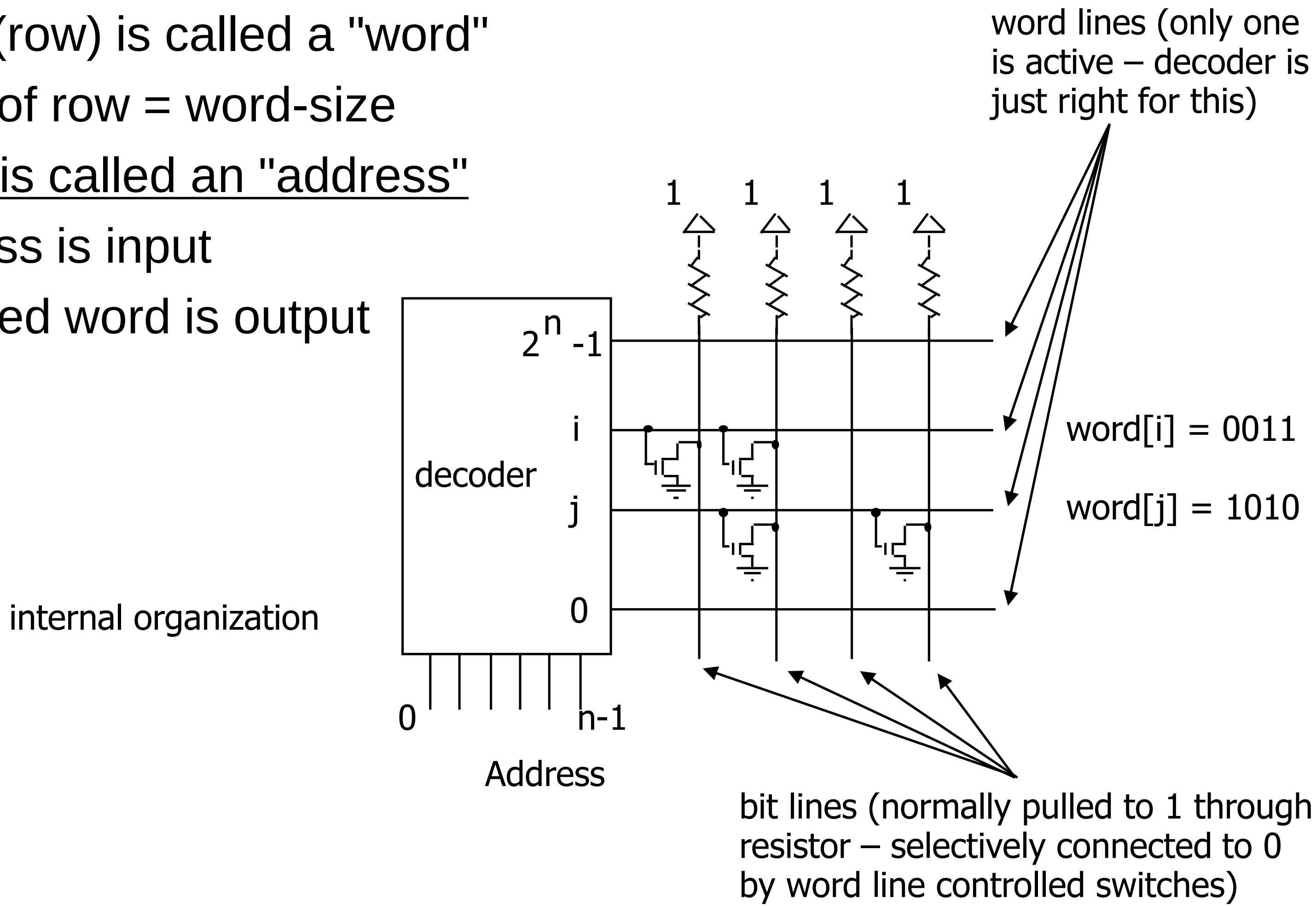
4:1 Multiplexer



Decoder + 4 Open Collector Gates

Read-only Memories

- Two dimensional array of 1s and 0s
  - entry (row) is called a "word"
  - width of row = word-size
  - <u>index is called an "address"</u>
  - address is input
  - selected word is output



ROMs and Combinational Logic

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

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

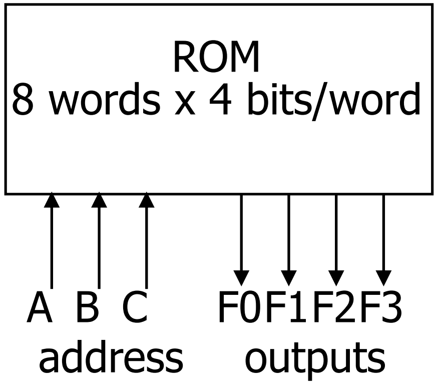
$$F1 = A'\,B'\,C \;+\; A'\,B\,C' \;+\; A\,B\,C$$

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

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

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

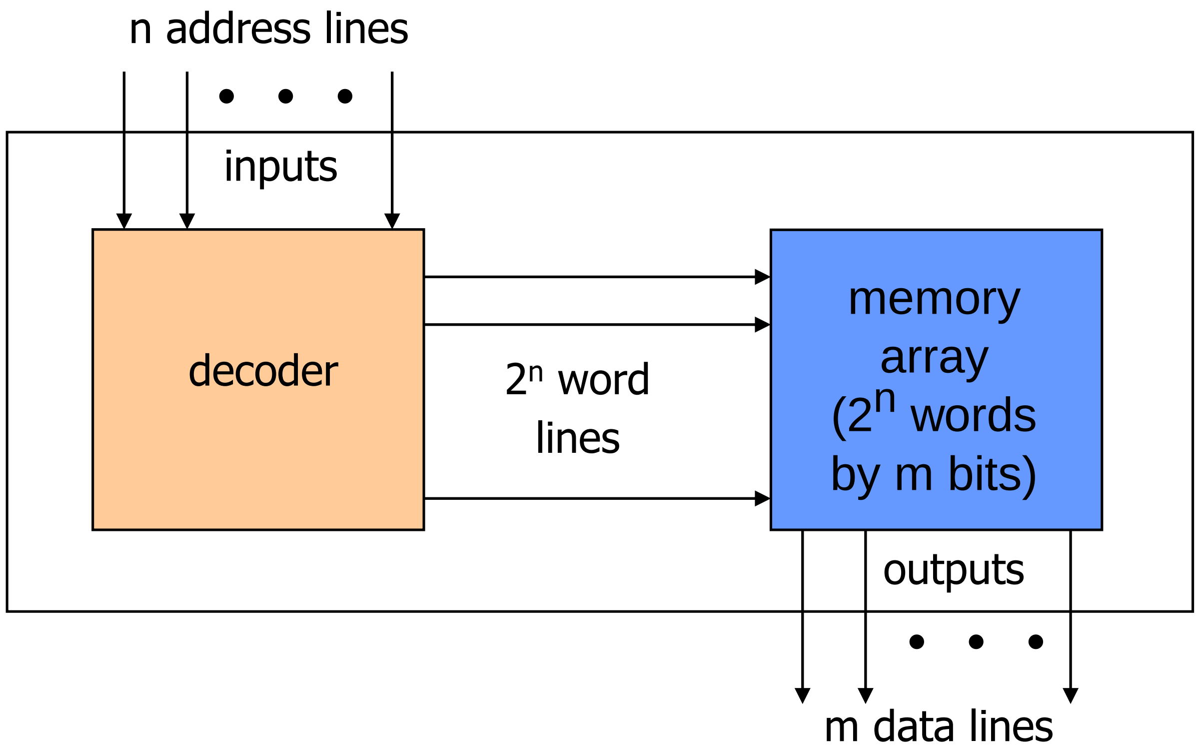
truth table



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: 2-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 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
  - 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

Comb-Logic Technology Summary

- Random logic
  - Single gates or in groups
  - 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