Digital Logic Design

4190.201

2014 Spring Semester

5. Combinational Logic Technologies

Naehyuck Chang
Dept. of CSE
Seoul National University
naehyuck@snu.ac.kr

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
  - Buffers survive and still evolve
  - Single gates are newly born
- 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

History

- Standard parts (1960)
  - SSI and MSI
  - 7400 and 4000 series
- ROMs and PALs
  - ROMs, EPROMs and EEPROMs
  - PALs and GALs
  - PLDs
- ASICs
  - Gate arrays
  - Standard cells
- FPGAs

Technology Metrics

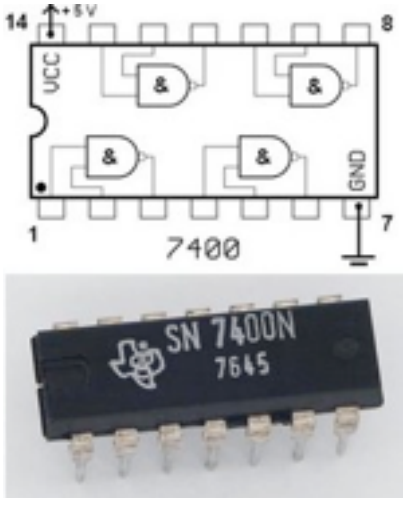
- Gate delay
- Degree of integration
- Power dissipation
- Noise margin
- Component cost
- Fan-out
- Driving capability
- Configurability

Random Logic

- Too hard to figure out exactly what gates to use
  - Map from logic to NAND/NOR networks
  - Determine minimum number of packages
    - Slight changes to logic function could decrease cost
- Changes to difficult to realize
  - Need to rewire parts
  - May need new parts
  - Design with spares (few extra inverters and gates on every board)

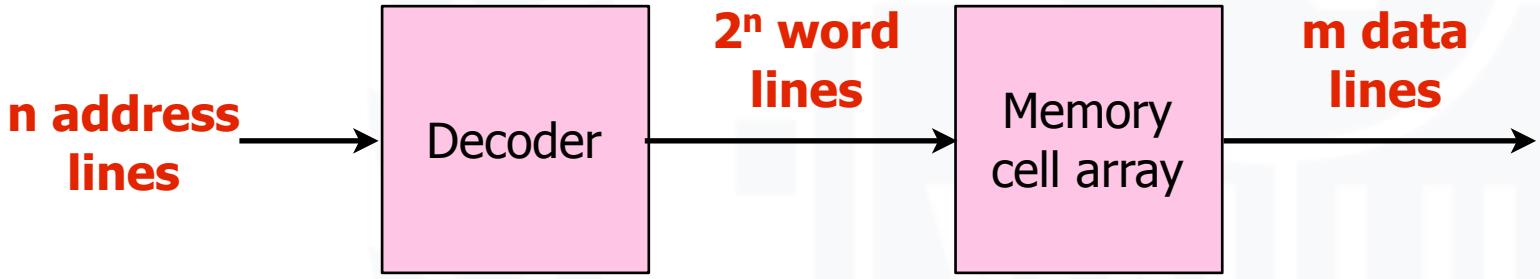
Fixed Logic

- 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



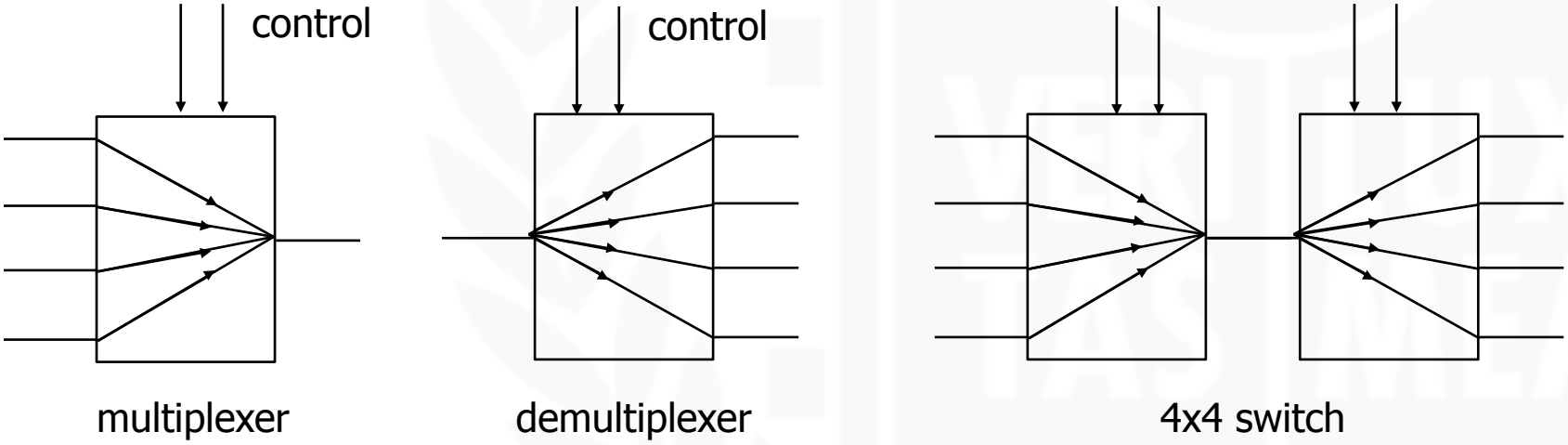
Look-up Table

- Read-only memory (ROM)



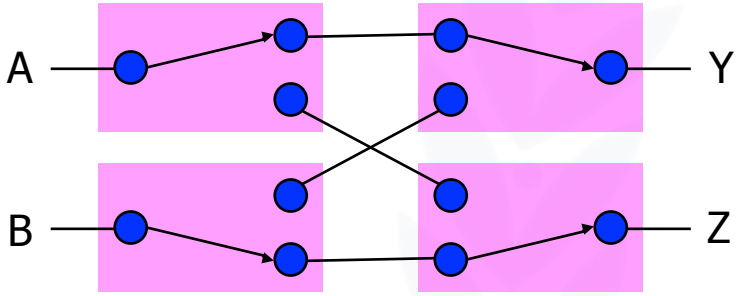
Making Connections

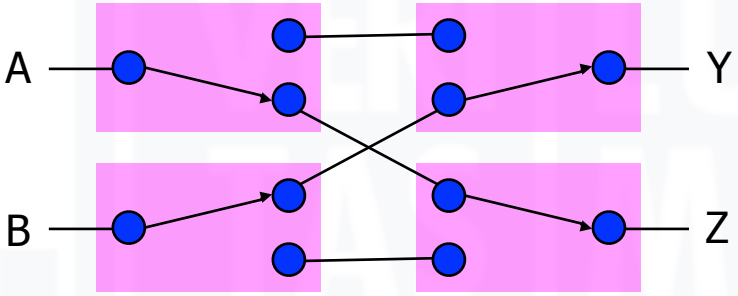
- Direct point-to-point connections between gates
  - Wires we've seen so far
- Route one of many inputs to a single output --- multiplexer
- Route a single input to one of many outputs --- demultiplexer



Mux and Demux

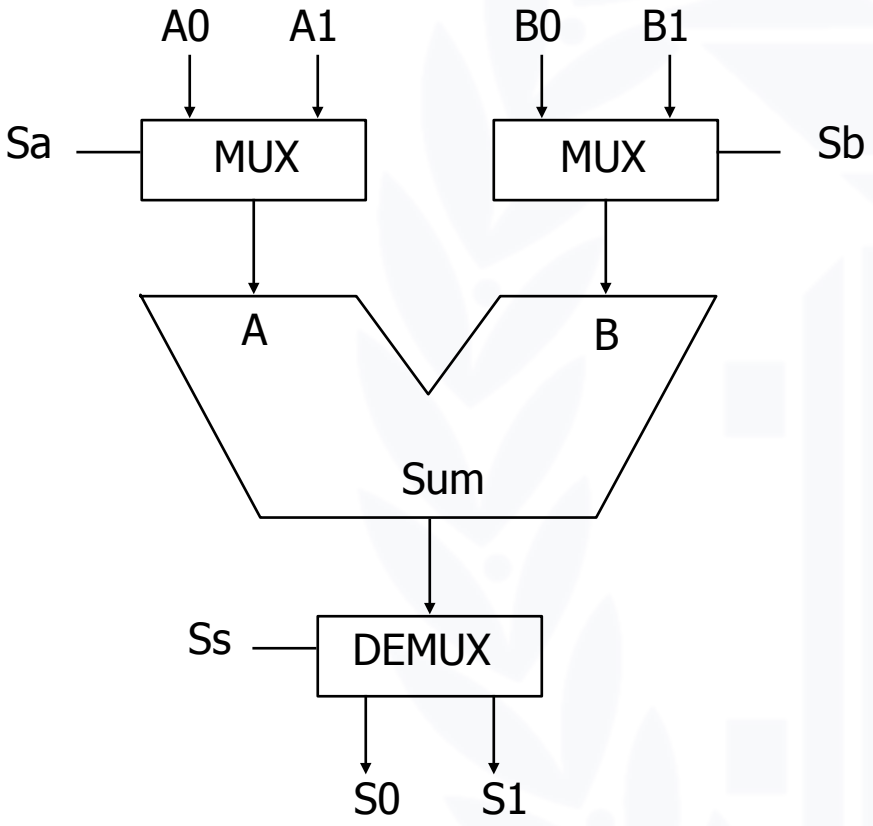
- Switch implementation of multiplexers and demultiplexers
  - Can be composed to make arbitrary size switching networks
  - Used to implement multiple-source/multiple-destination interconnections





Mux and Demux (cont'd)

- Uses of multiplexers/demultiplexers in multi-point connections



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

Functional form

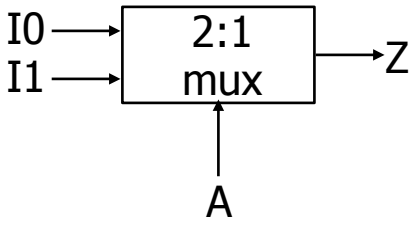
Logical form

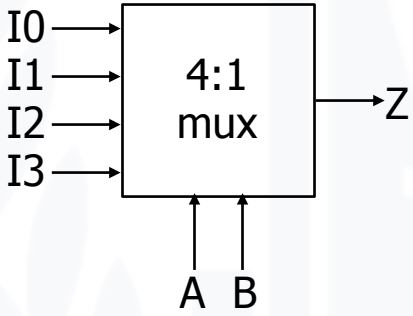
I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

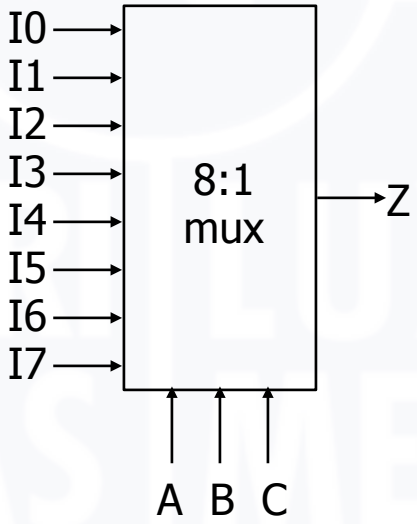
Two alternative forms
for a 2:1 Mux truth table

Multiplexers/Selectors (cont'd)

- 2:1 mux: $Z = A'I0 + AI_1$
- 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: $Z = \sum_{k=0}^{2^n-1}(m_k I_k)$
  - In minterm shorthand form for a 2^n:1 mux

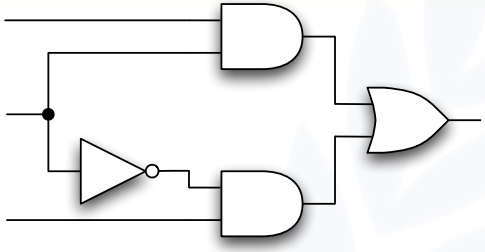


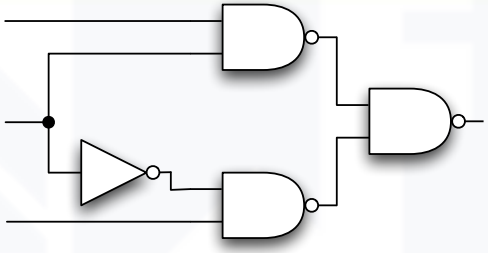




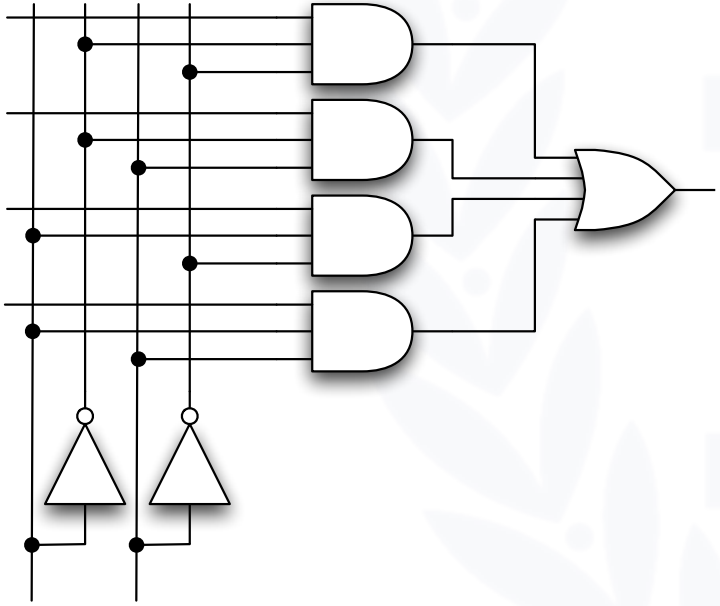
Gate Level Implementation of Muxes

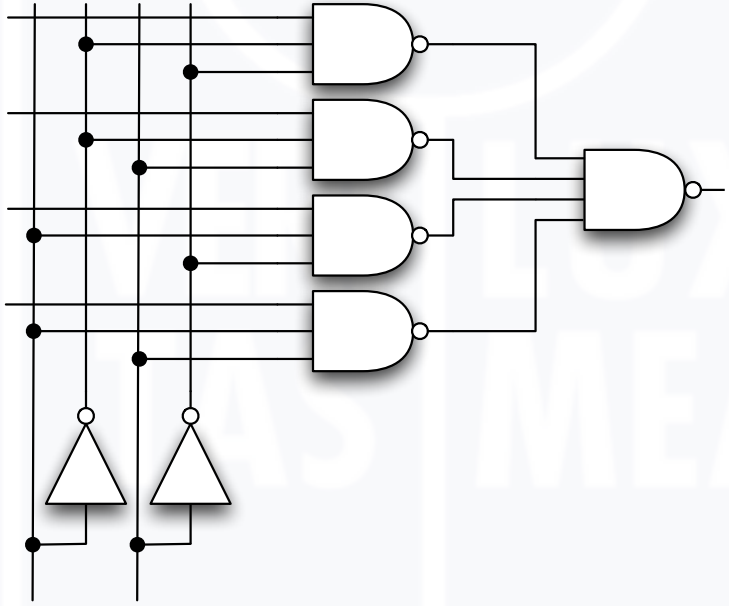
- 2:1 mux





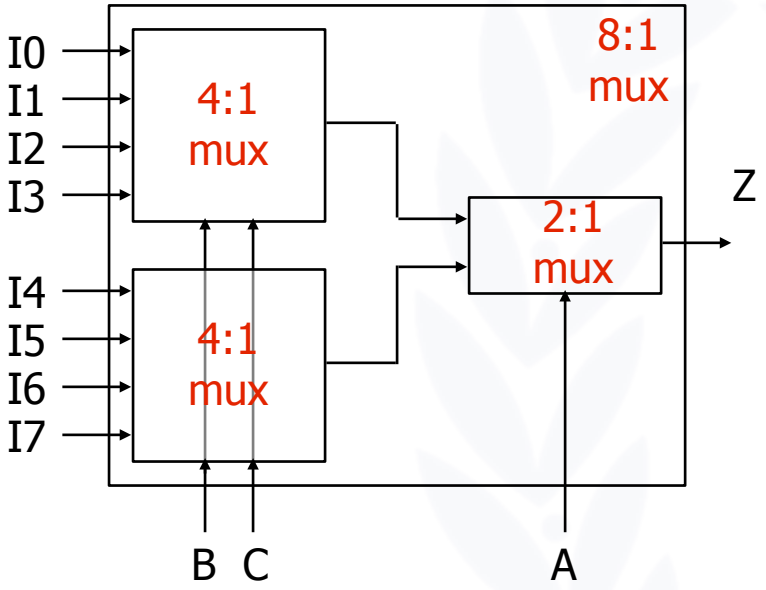
- 4:1 mux





Cascading Multiplexers

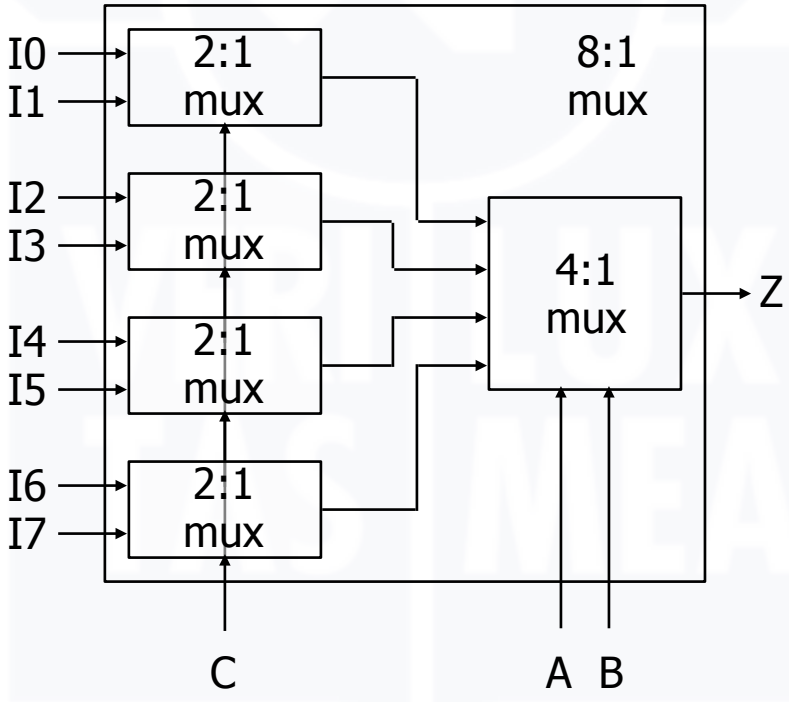
- Large multiplexers can be made by cascading smaller ones



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

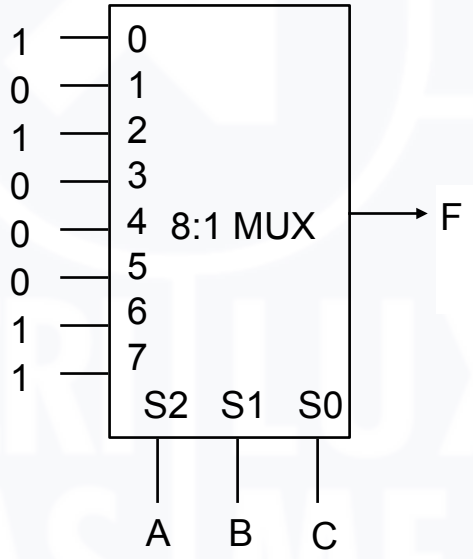
Alternative implementation



Multiplexers as General-Purpose Logic

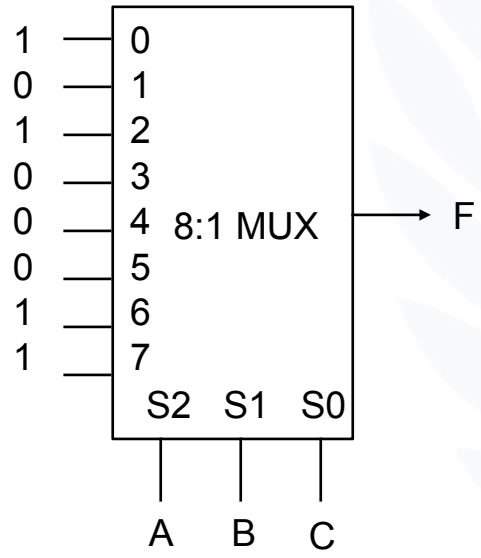
- A 2^n:1 multiplexer can implement any function of n variables
  - With the variables used as control inputs and
  - The data inputs tied to 0 or 1
  - In essence, a lookup table
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'C'(1) + A'B'C(0)
    + A'BC'(1) +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

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

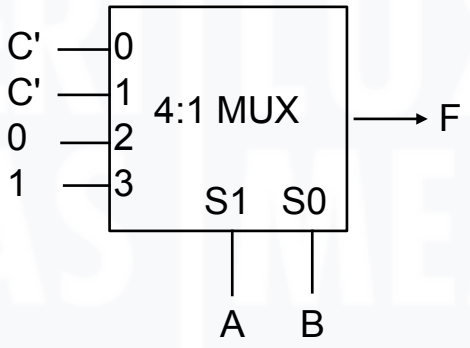


Multiplexers as General-Purpose Logic (cont'd)

- A 2n-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)

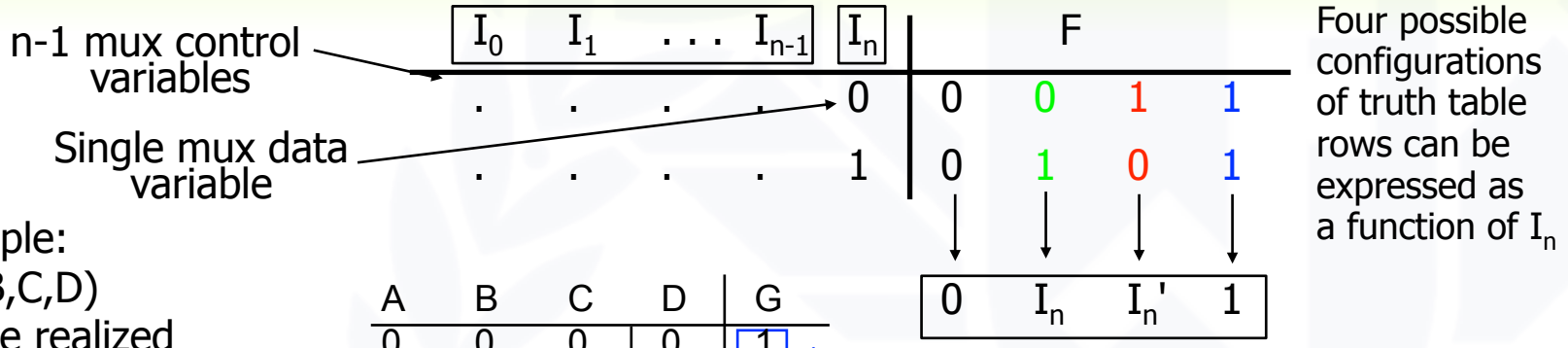


A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



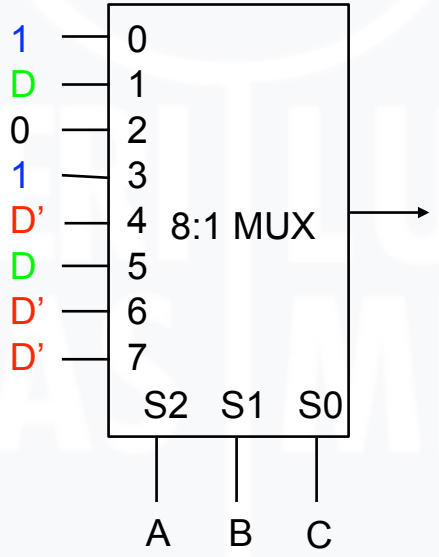
Multiplexers as General-Purpose Logic (cont'd)

- Generalization



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

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, 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

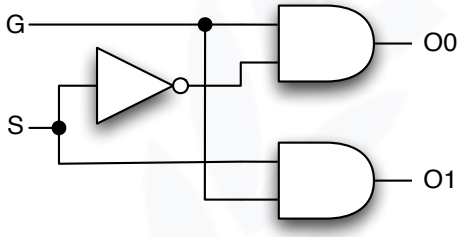
3:8 Decoder:

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

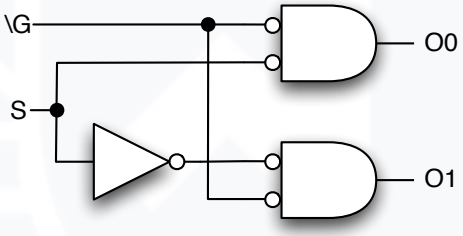
Gate Level Implementation of Demultiplexers

- 1:2 decoders

Active-high enable

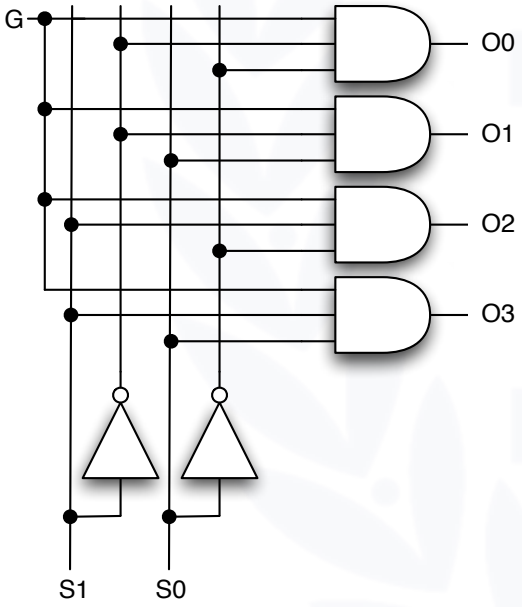


Active-low enable

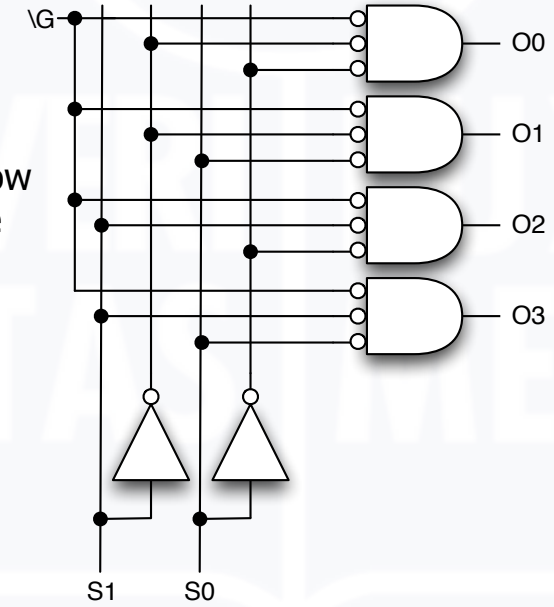


- 2:4 decoders

Active-high enable

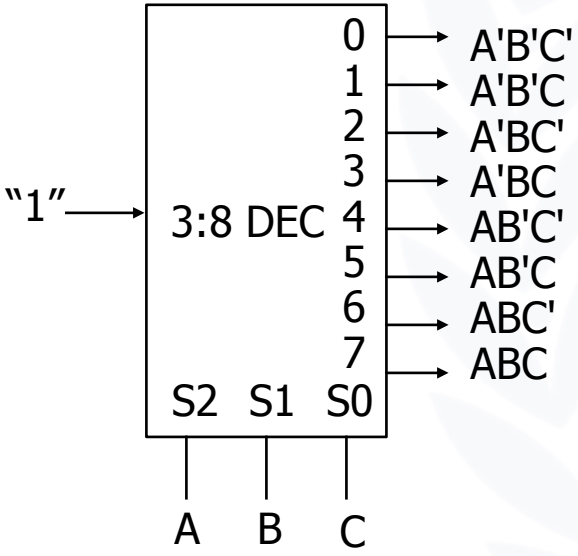


Active-low enable



Demultiplexers as General-Purpose Logic

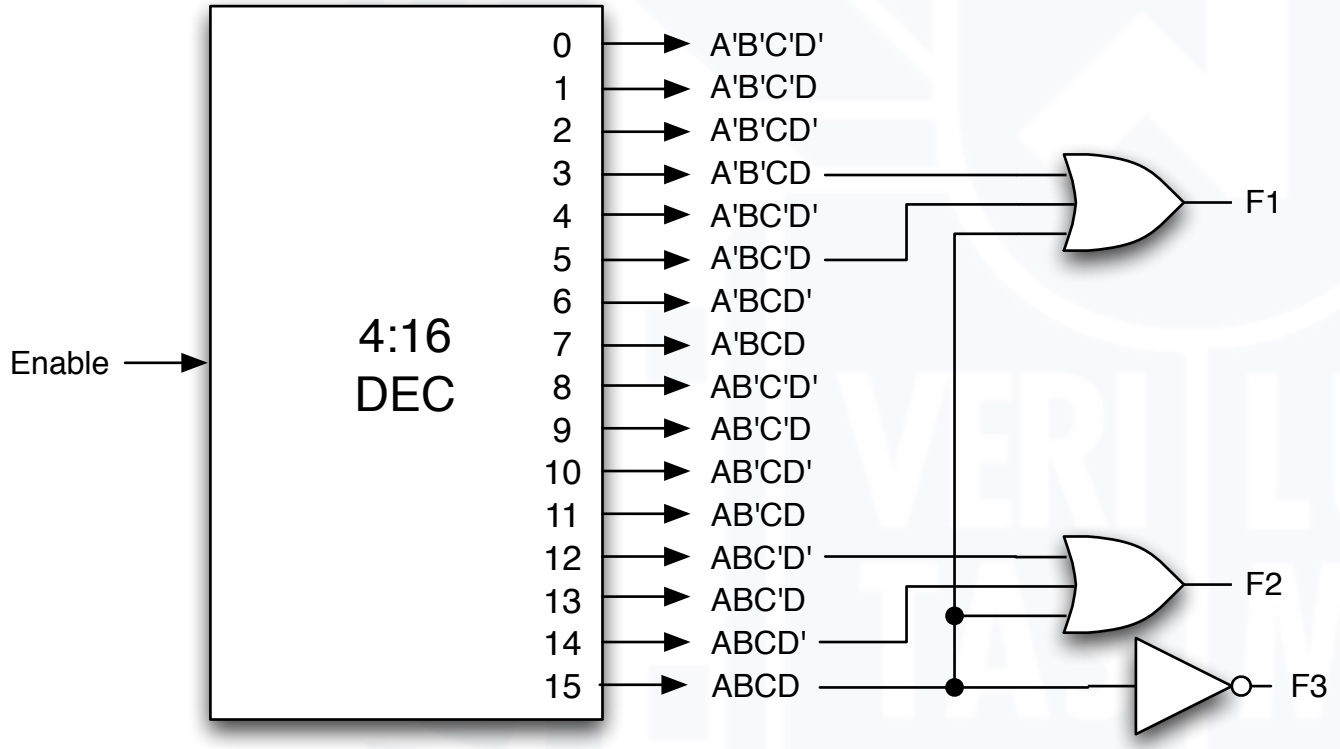
- A n:2^n decoder can implement any function of n variables
  - With the variables used as control inputs
  - The enable inputs tied to 1 and
  - The appropriate minterms summed to form the function



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

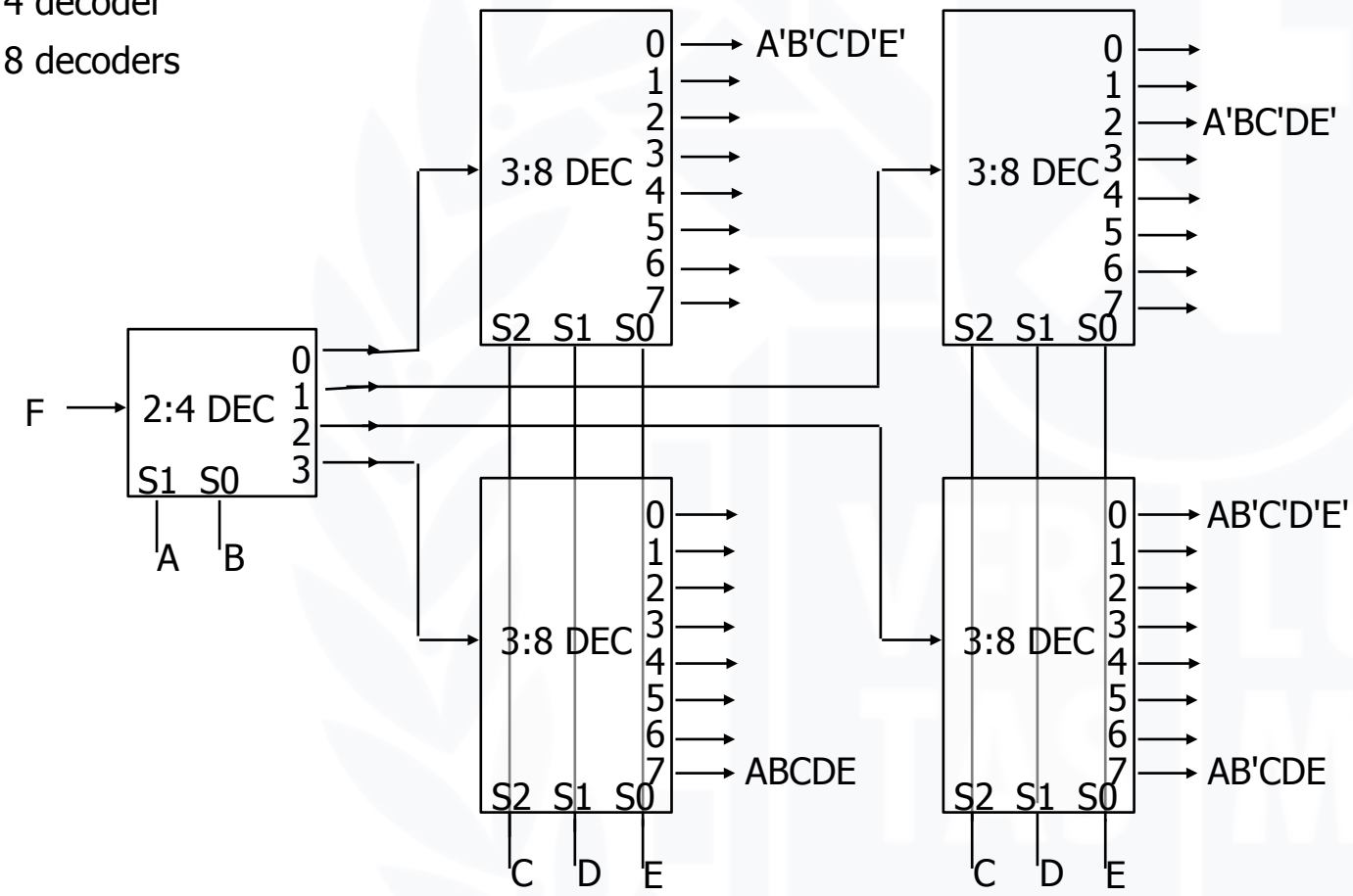
Demultiplexers as General-Purpose Logic (cont'd)

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



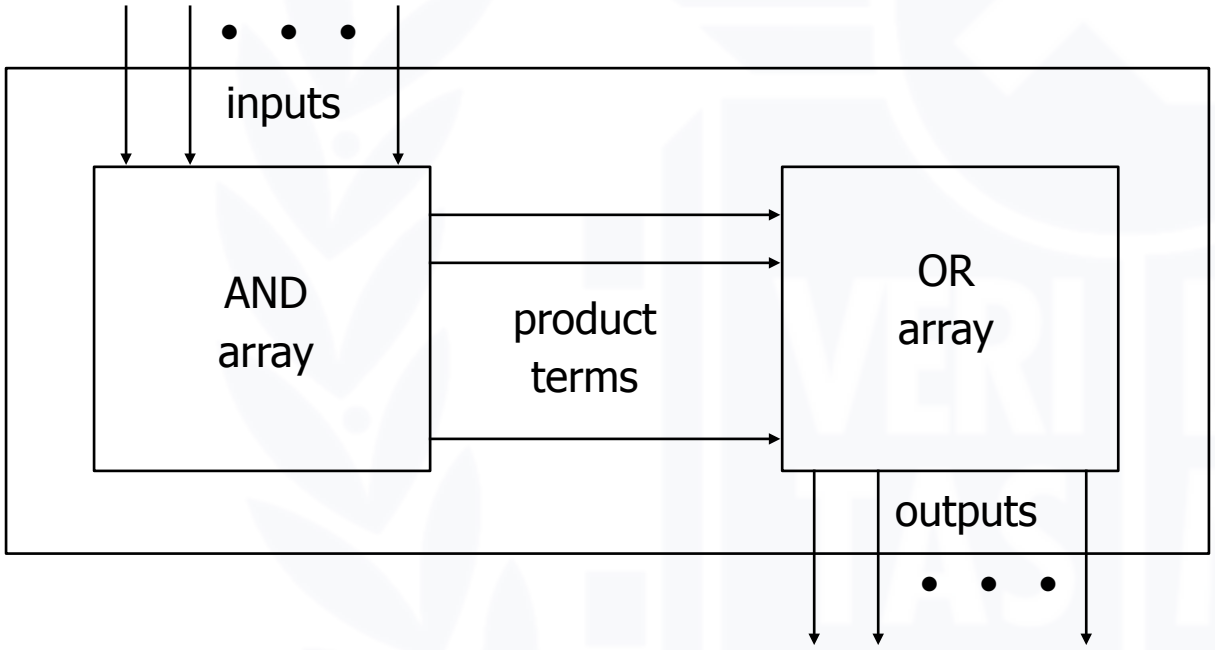
Cascading Decoders

- 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

input side:

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

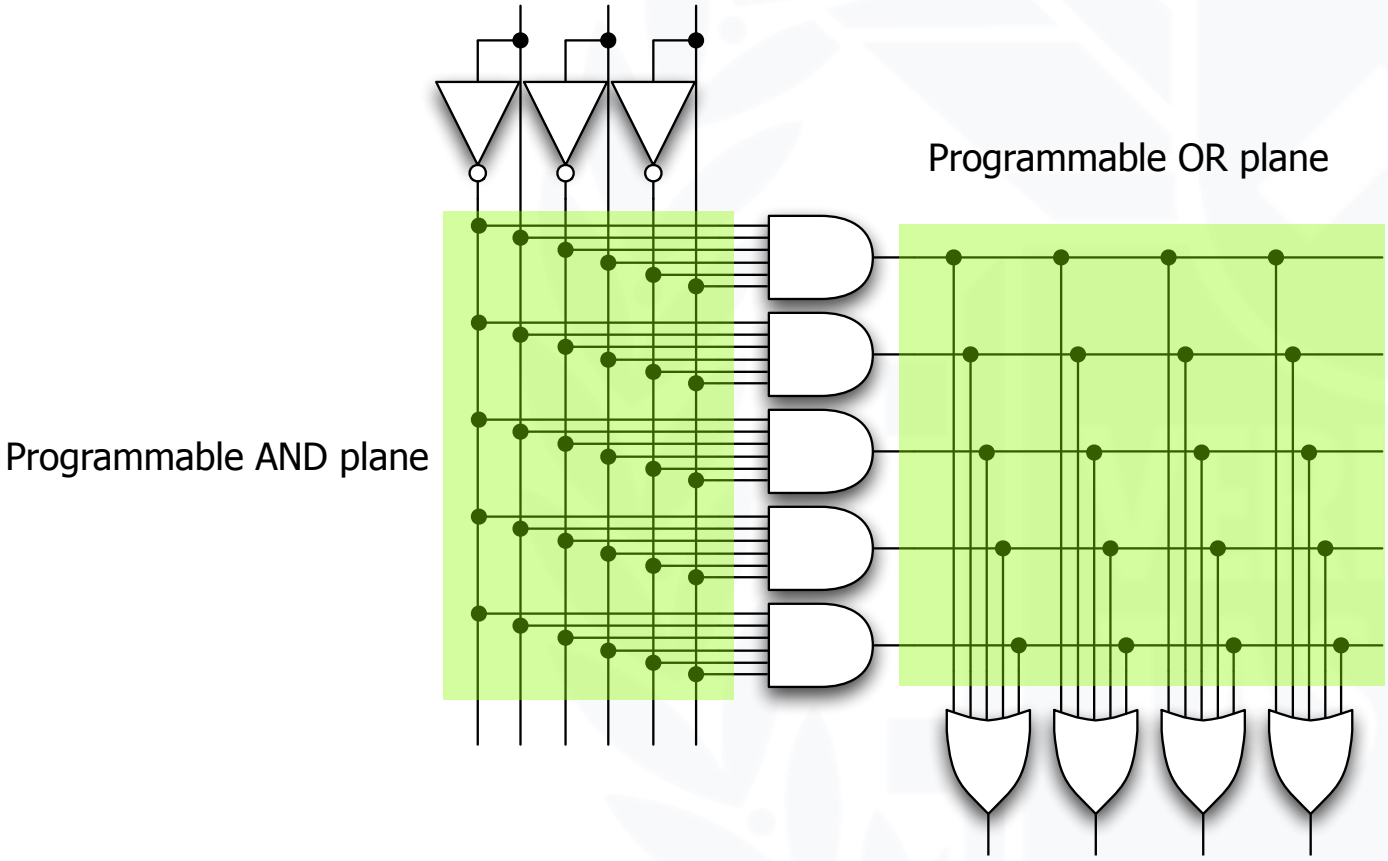
output side:

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

reuse of terms

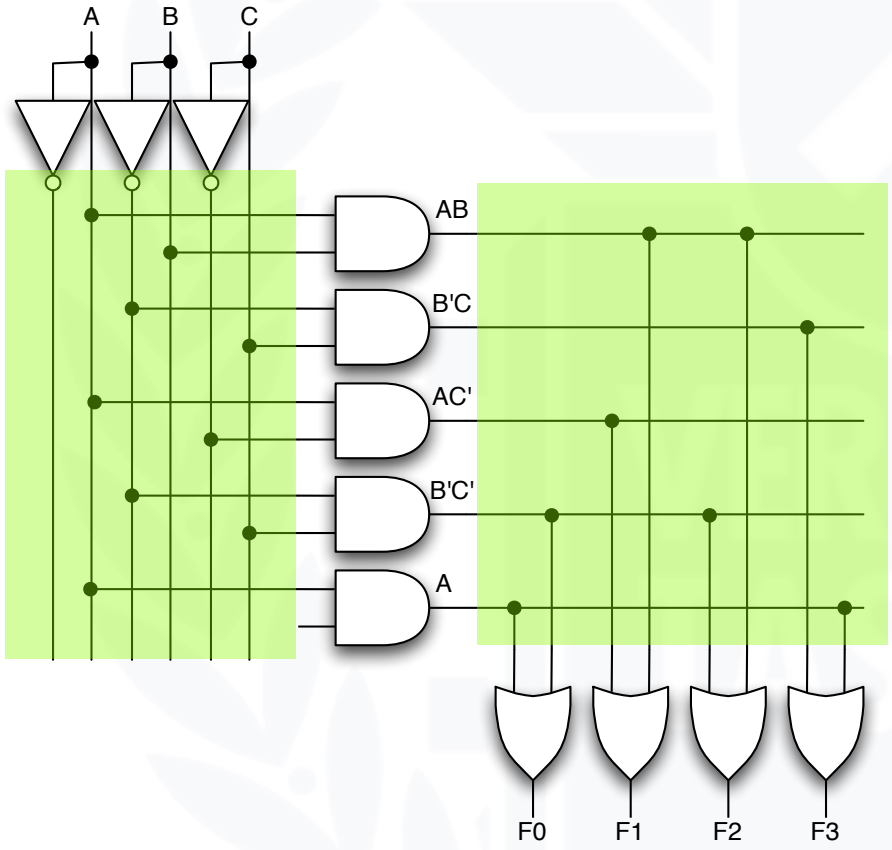
Before Programming

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



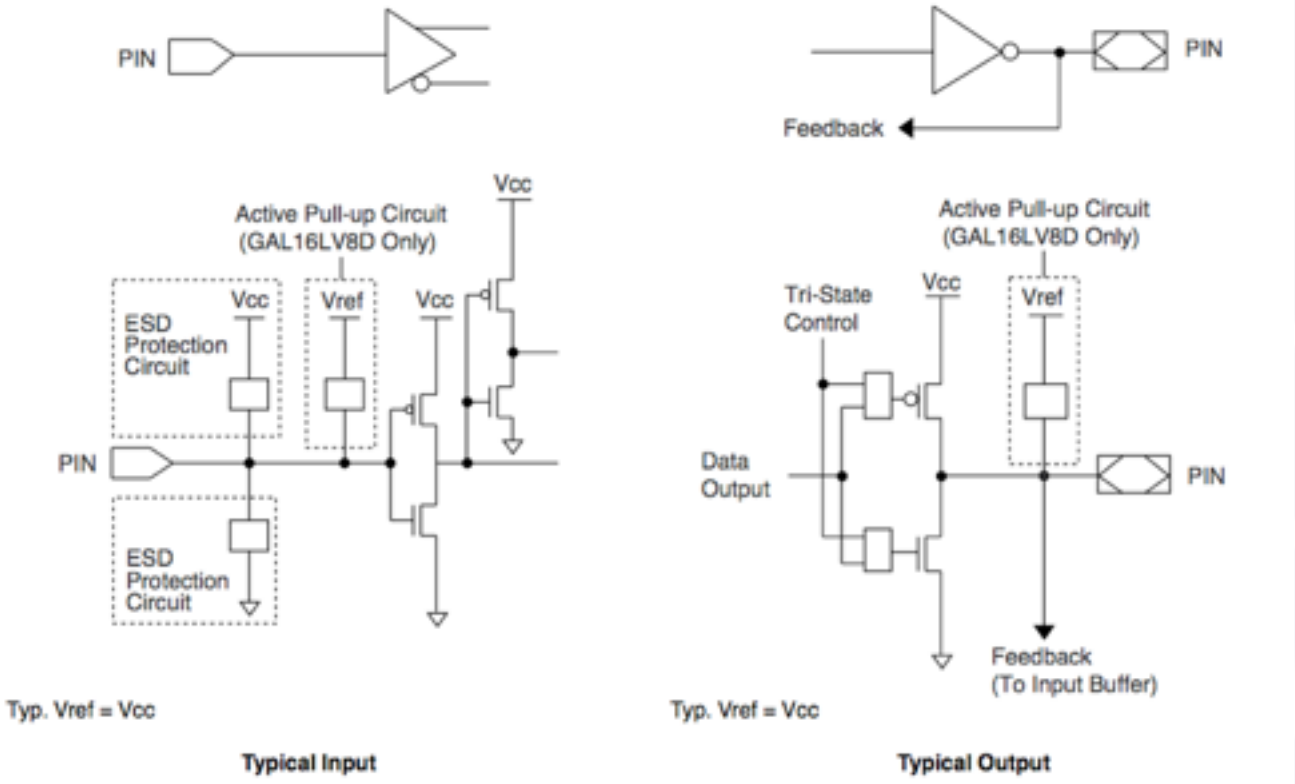
After Programming

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



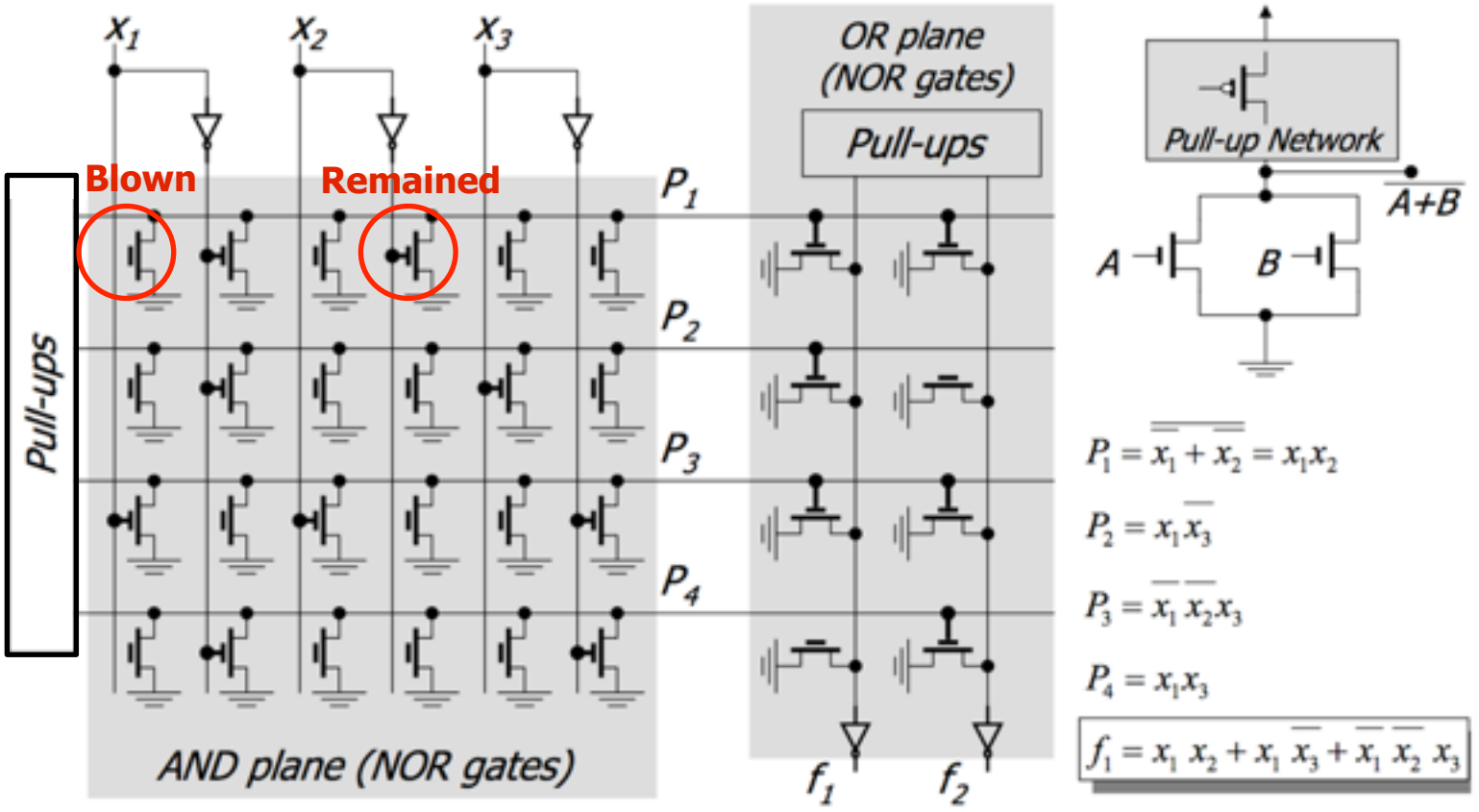
Input/Output Buffers

- I/O buffer structures



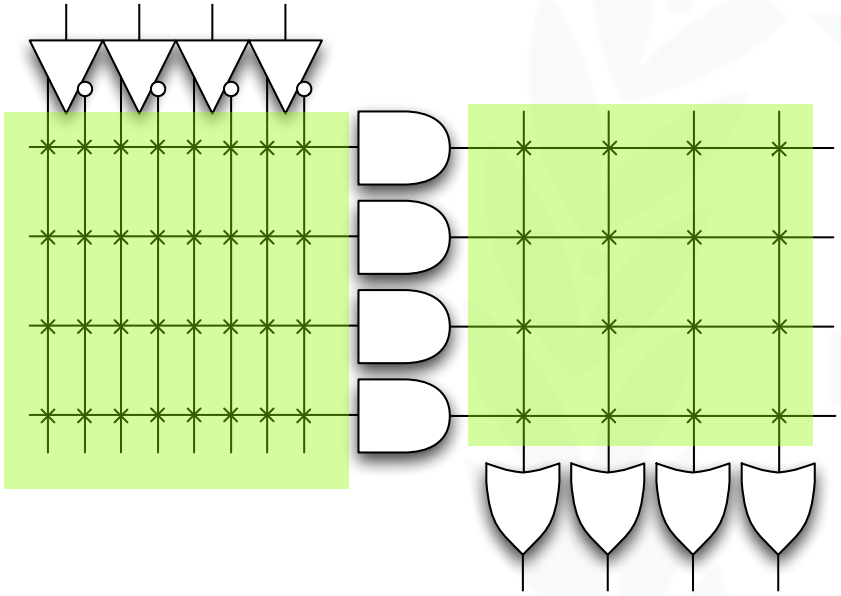
Internal Structure

- Internal AND-array and OR-array structures



Alternate Representation for High Fan-in Structures

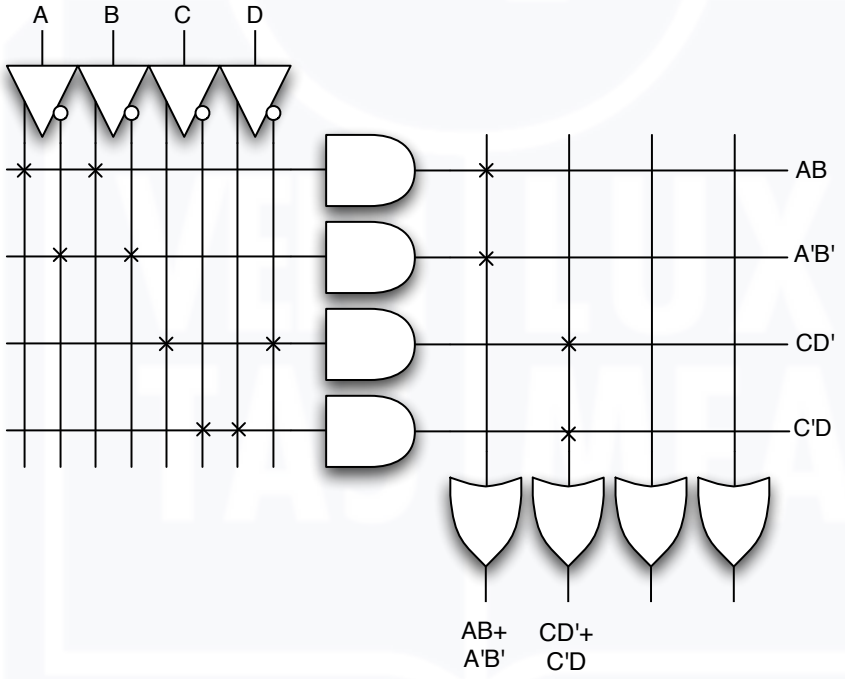
- Short-hand notation so we don't have to draw all the wires
  - Signifies a connection is present and perpendicular signal is an input to gate



Notation for implementing

F0 = A B + A' B'

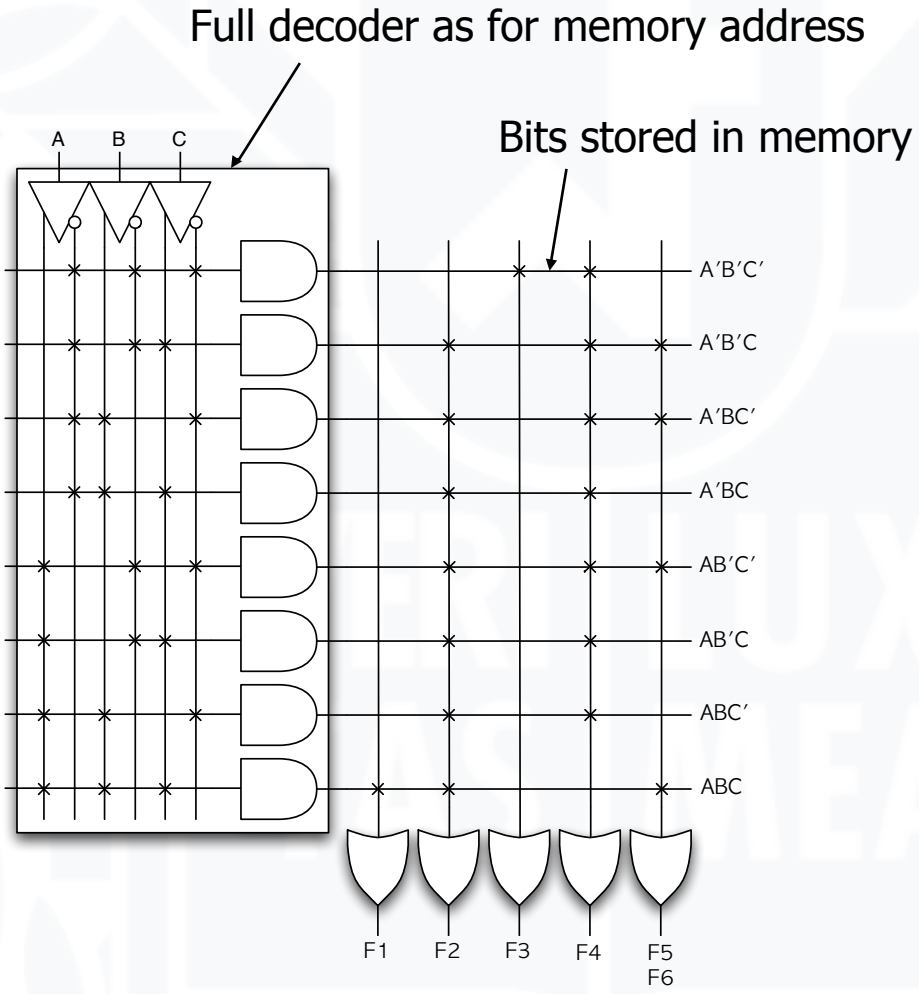
F1 = C D' + C' D



Programmable Logic Array Example

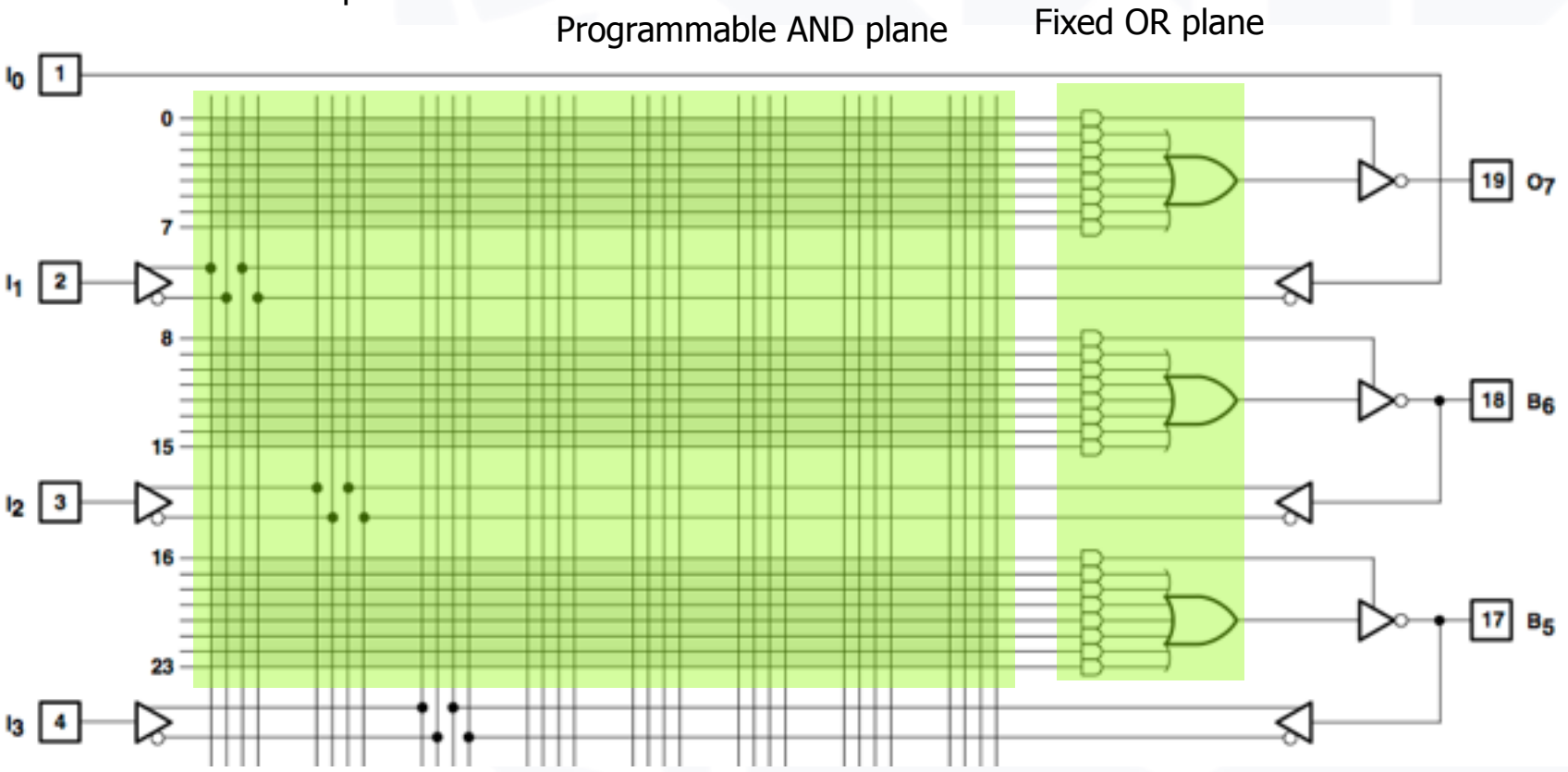
- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A' B' C'
  - F4 = A' + B' + C'
  - F5 = A xor B xor C
  - F6 = A xnor B xnor C

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



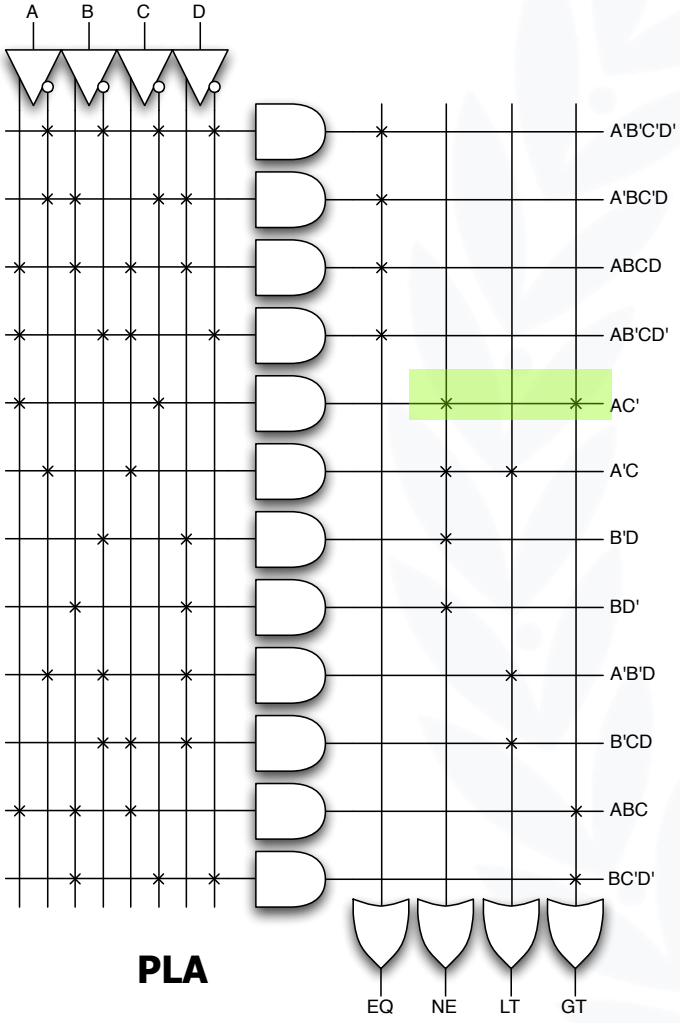
PALs and PLAs

- Programmable array logic (PAL)
  - Programmable AND plane
  - **Fixed OR plane - not more than xx product terms**
  - Innovation by Monolithic Memories
  - Faster and smaller OR plane

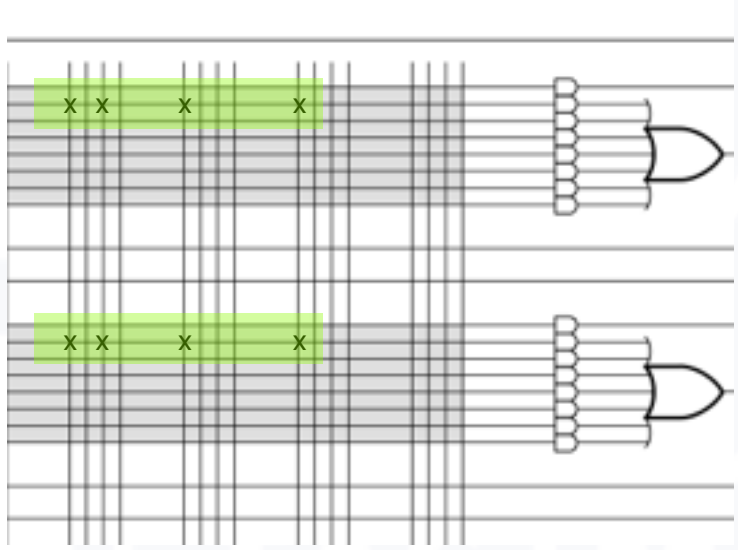


PLA and PAL

- Product term sharing



PLA



PAL

PALs and PLAs: Design Example

- BCD to Gray code converter

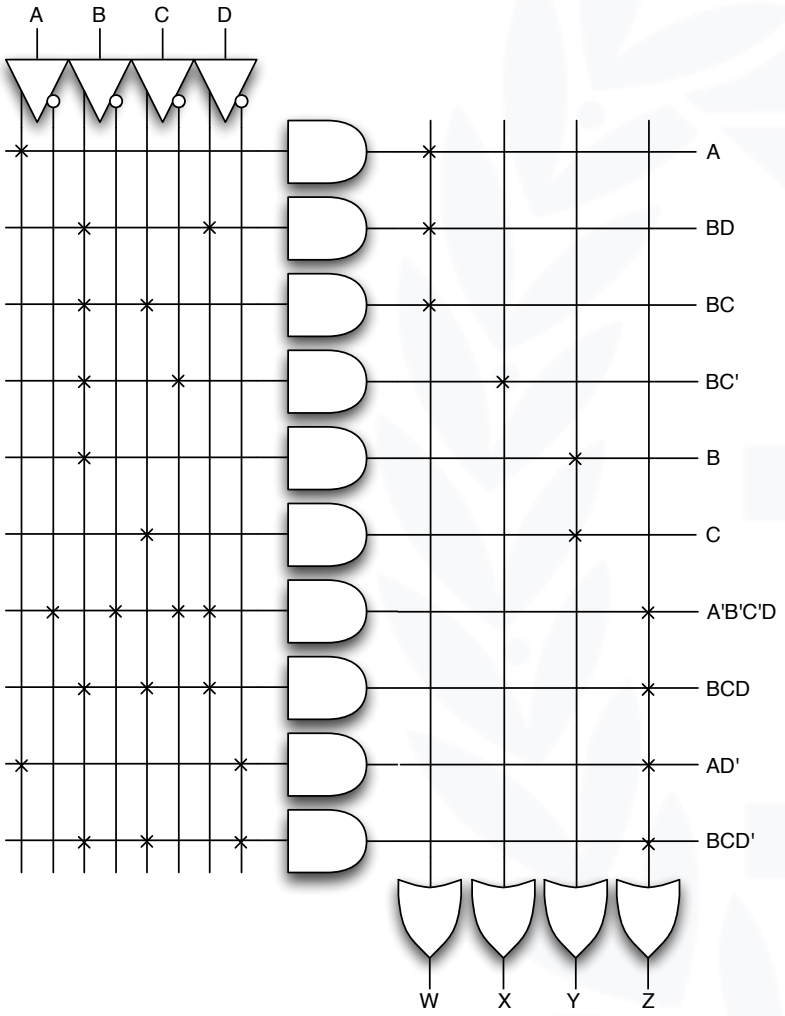
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

Minimized functions:

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

PALs and PLAs: Design Example (cont'd)

- Code converter: programmed PLA



Minimized functions:

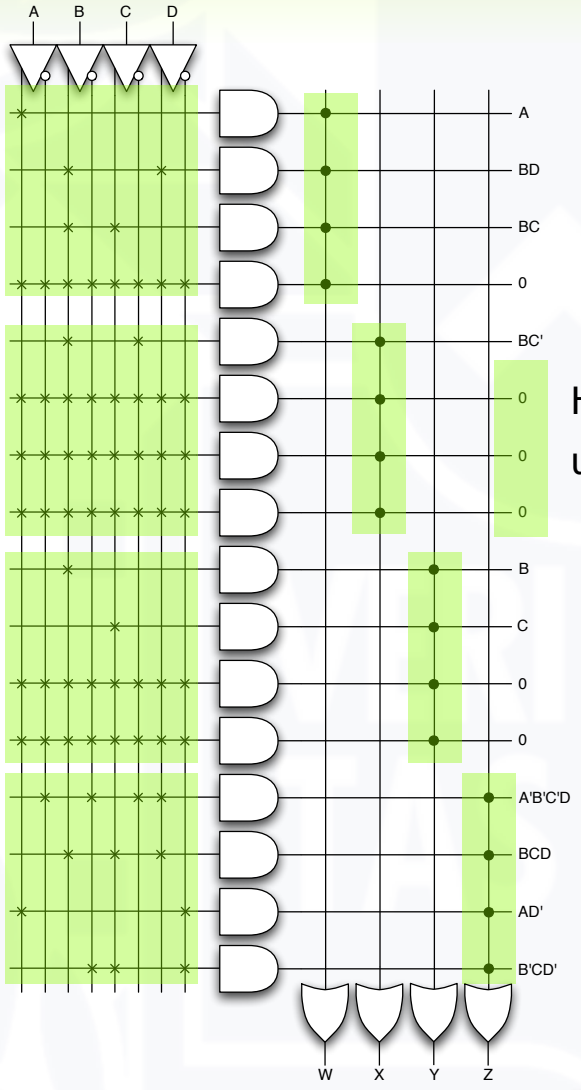
W = A + BD + BC
X = B C'
Y = B + C
Z = A'B'C'D + BCD + AD' + B'CD'

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

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

PALs and PLAs: Design Example (cont'd)

- Code converter: programmed PAL



Fixed OR plane
4 input OR

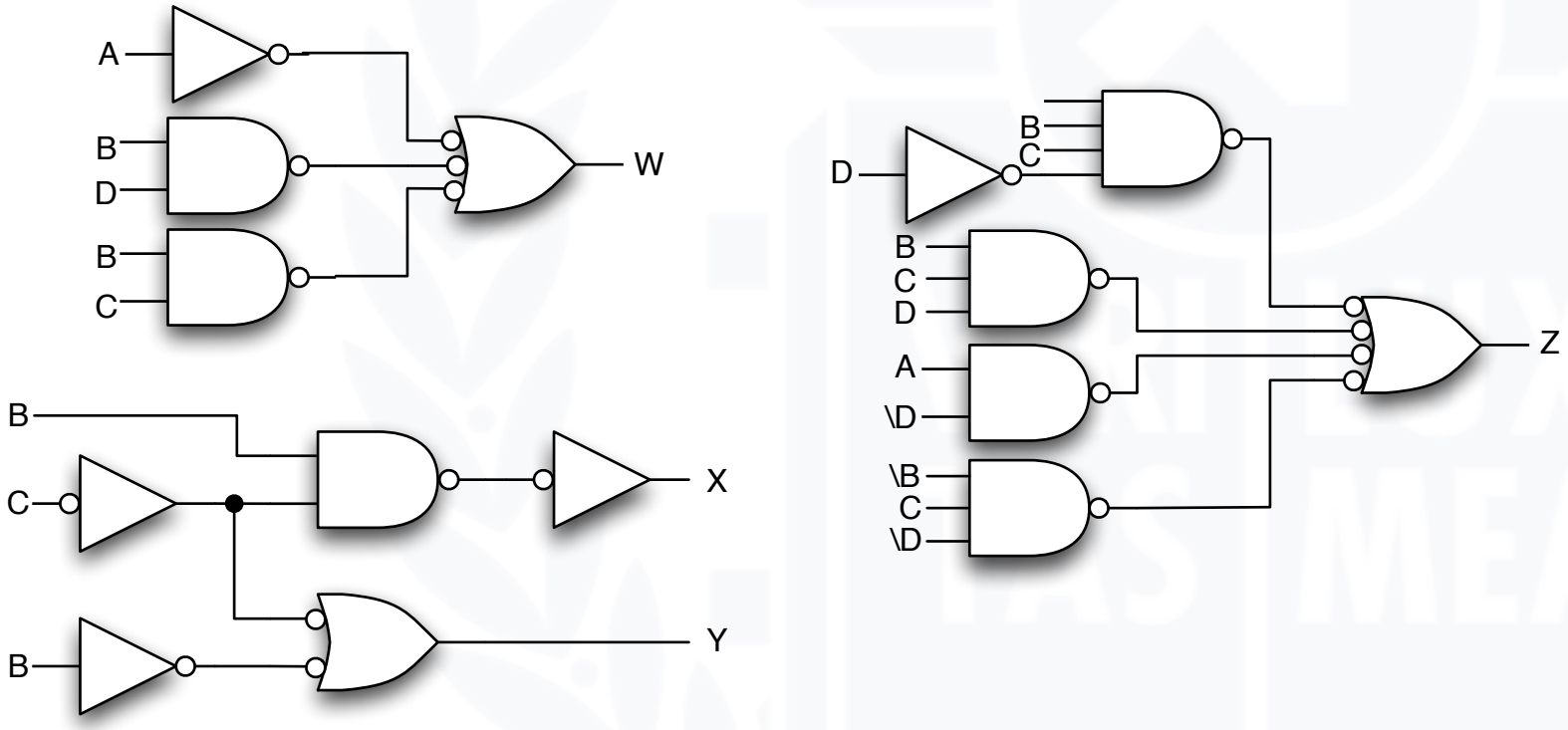
Have to make 0 for unused OR input

4 product terms per each OR gate

Programmable AND plane
Product terms are not shared

PALs and PLAs: Design Example (cont'd)

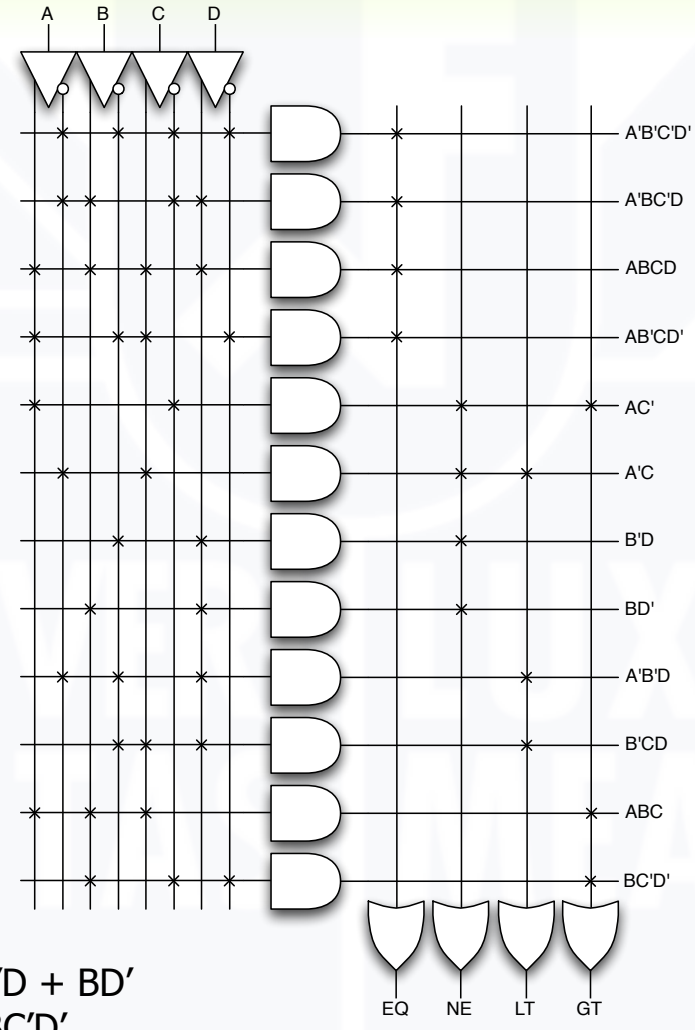
- Code converter: NAND gate implementation
  - Loss or regularity, harder to understand
  - Harder to make changes



PALs and PLAs: Another Design Example

- Magnitude comparator

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



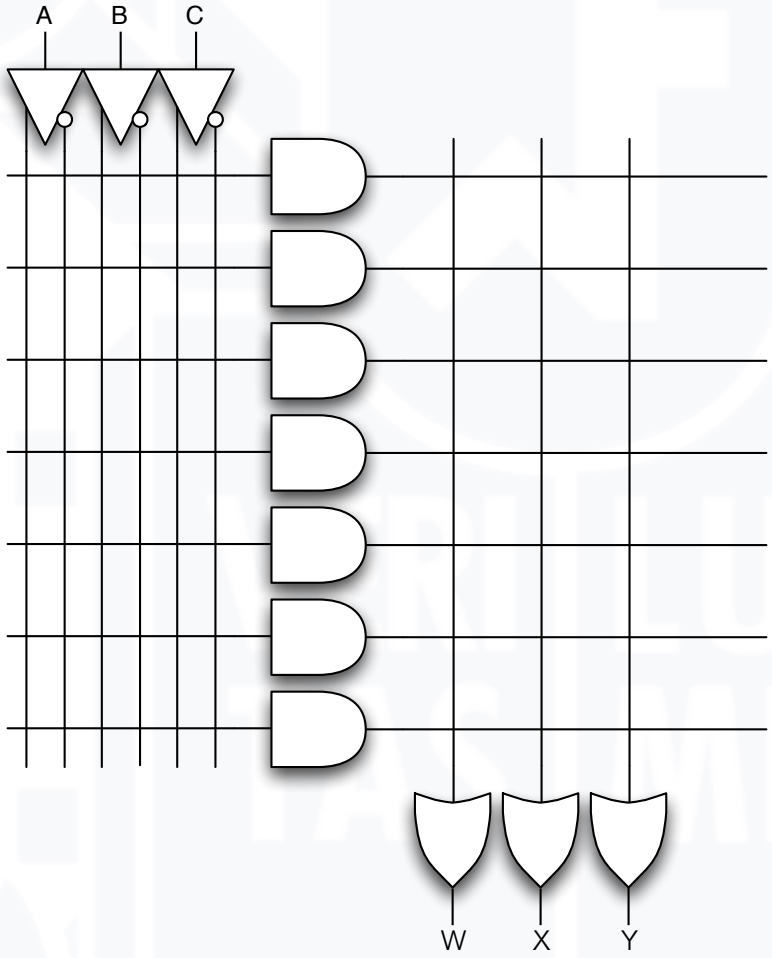
Minimized functions:

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

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

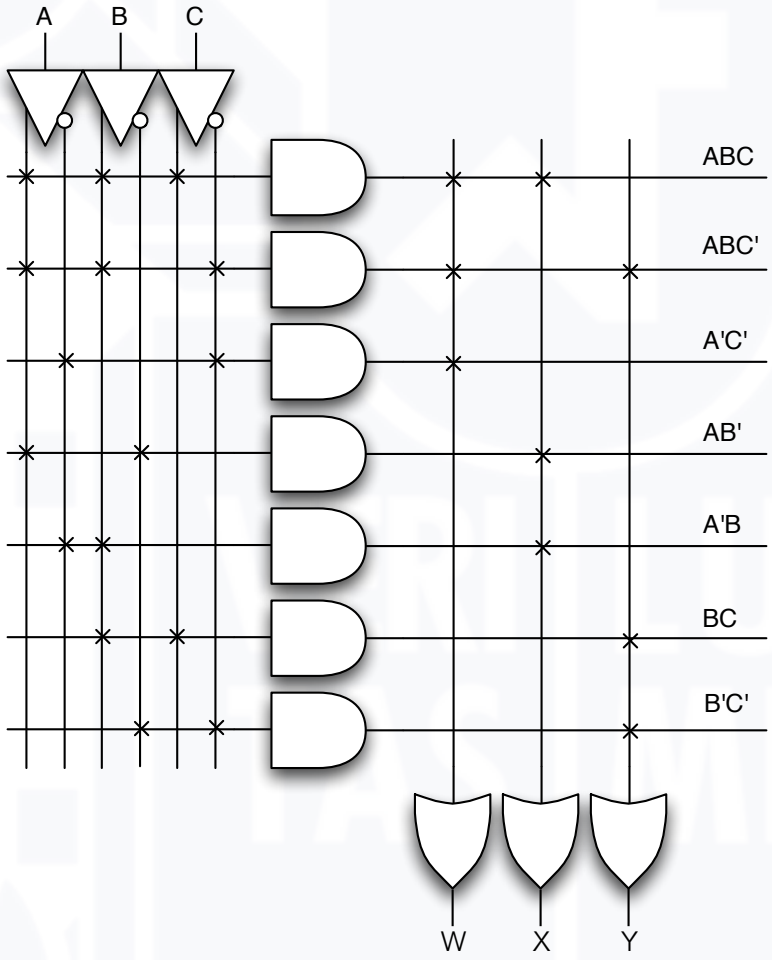
Activity

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



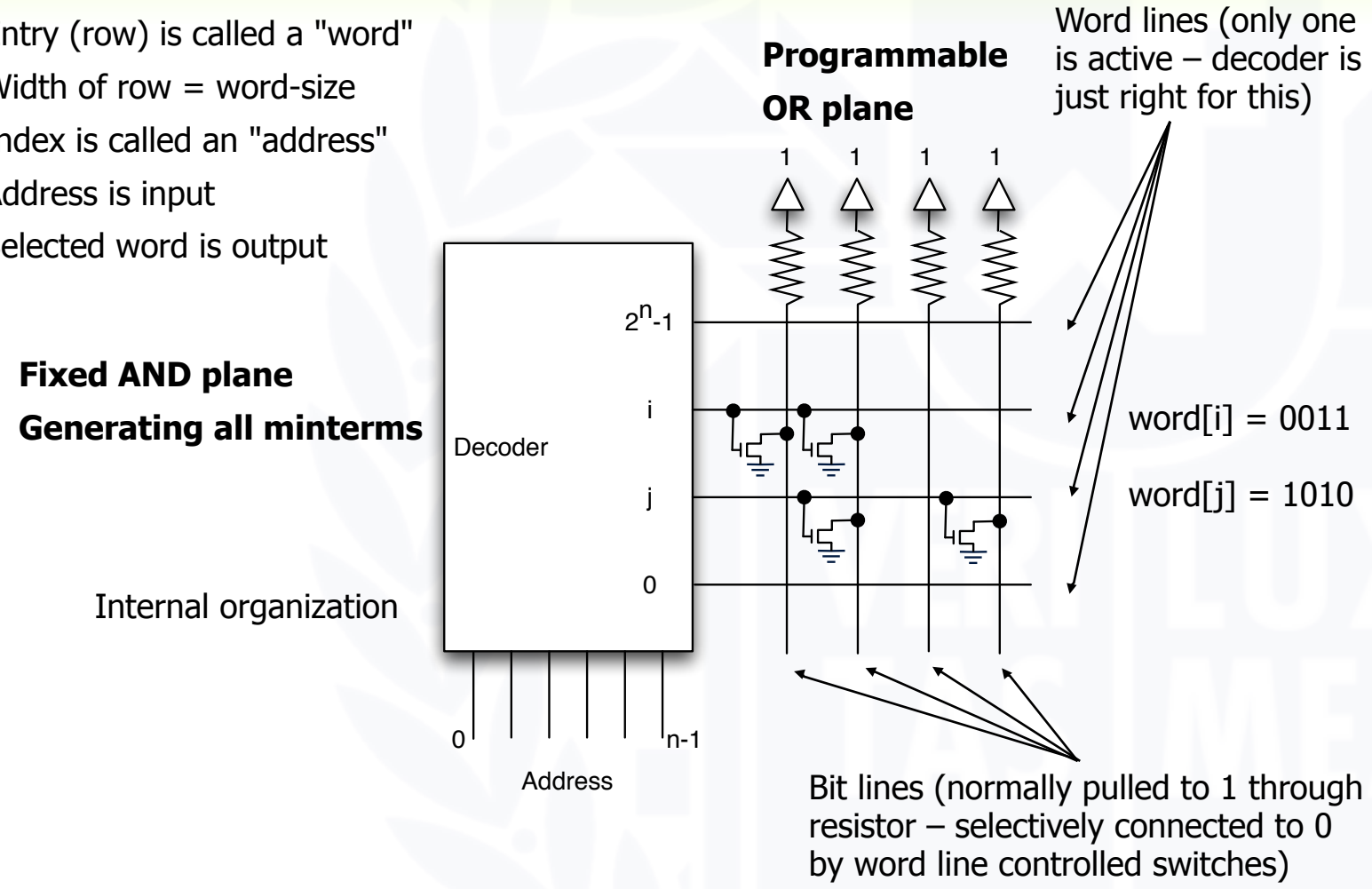
Activity (cont'd)

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



Read-Only Memories

- Two dimensional array of 1s and 0s
  - Entry (row) is called a "word"
  - Width of row = word-size
  - Index is called an "address"
  - Address is input
  - Selected word is output



ROMs and Combinational Logic

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

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

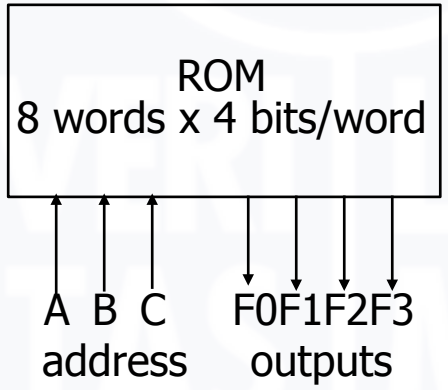
$$F1 = A'\,B'\,C + A'\,B\,C' + A\,B\,C$$

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

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

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

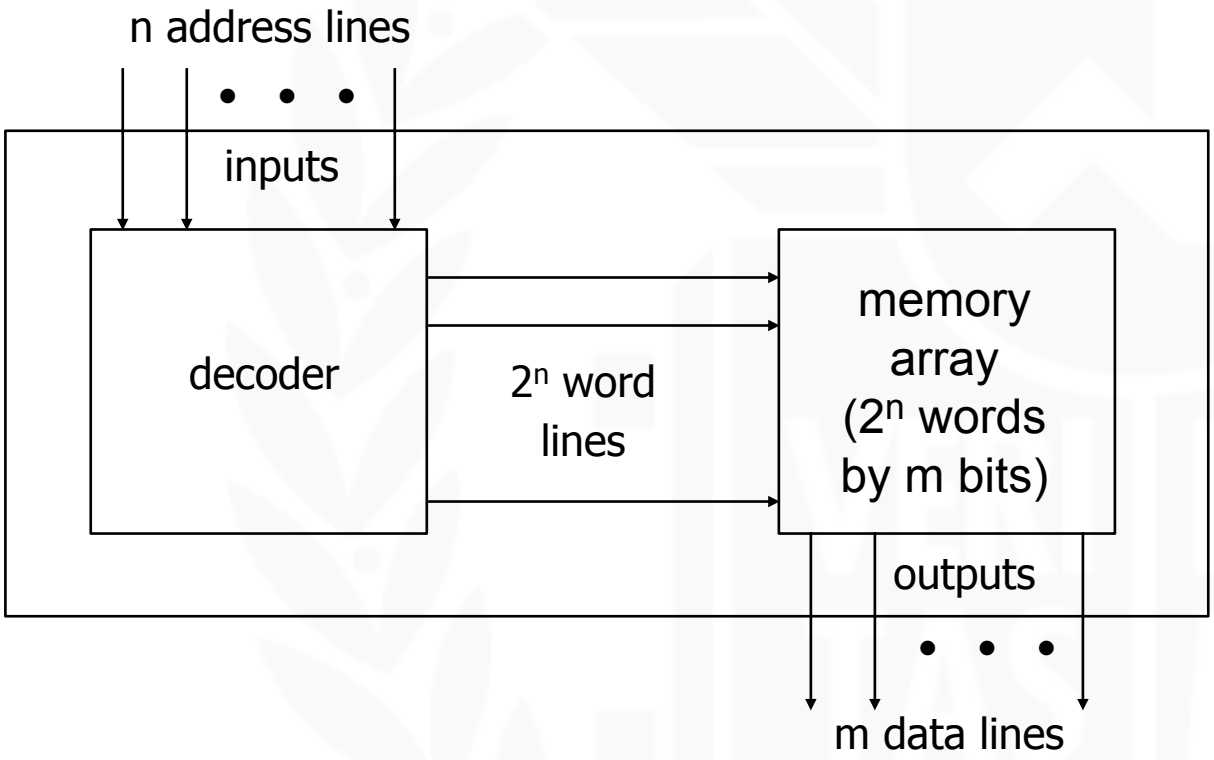
Truth table



Block diagram

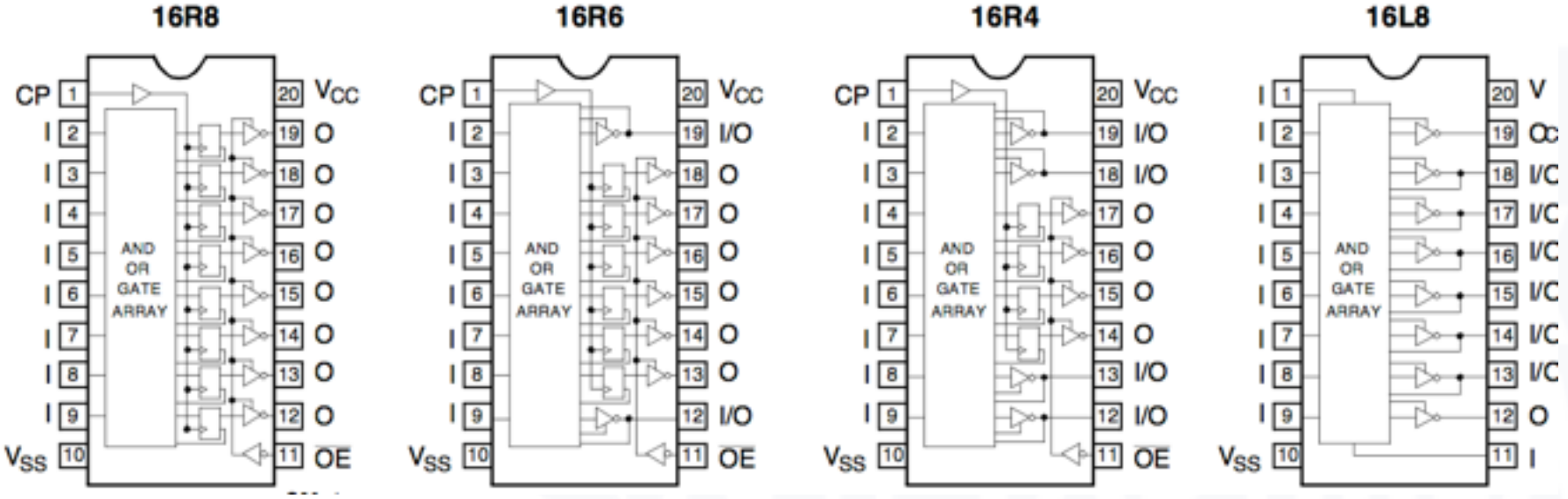
ROM structure

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



Types of PALs

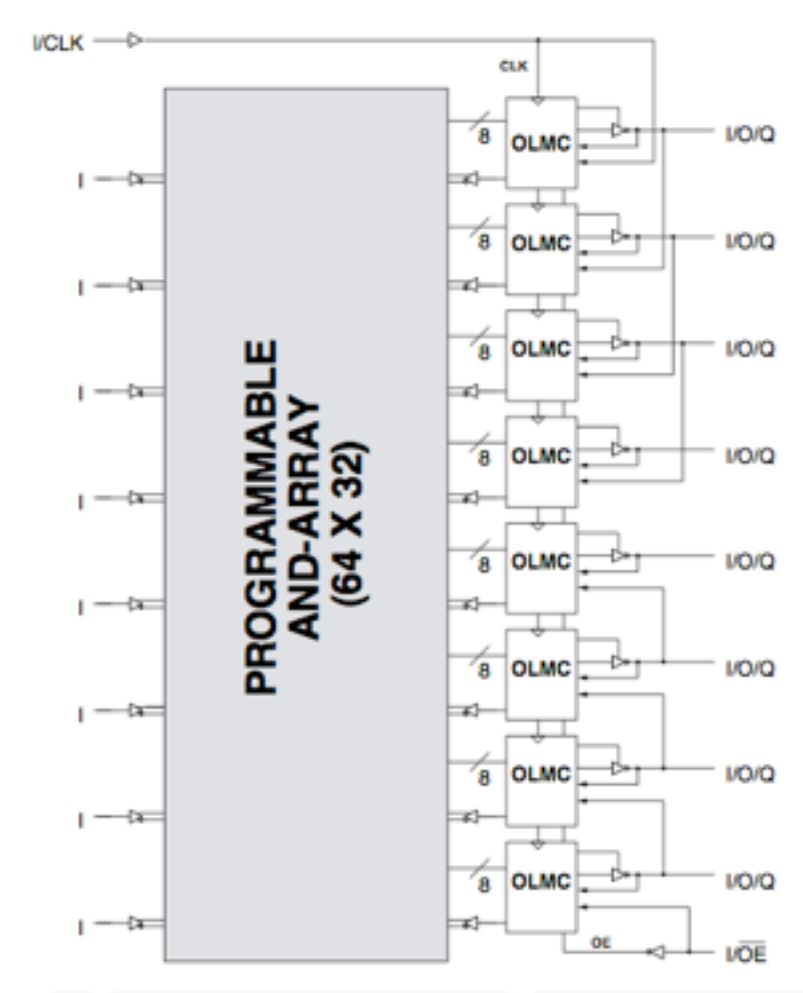
- Typical PAL types
  - Number of inputs
  - Number of OR-plane inputs
  - Number of outputs
  - Registered output



www.cypress.com

GAL

- GAL (Generic Array Logic)
  - High-speed EECMOS
  - 2.5 ns maximum clock input to output data



www.latticesemi.com

Electrical Characteristics

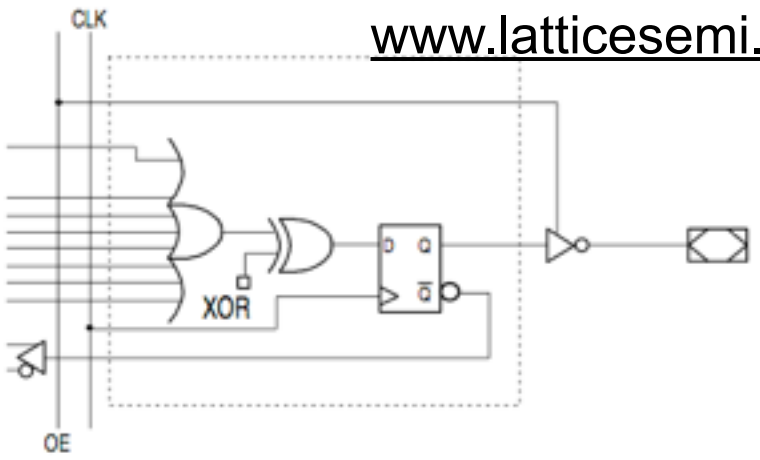
- 3.3 and 5 V compatible
  - 3 to 15 ns input to output delay

SYMBOL	PARAMETER	CONDITION	MIN.	TYP.[3]	MAX.	UNITS
V_{IL}	Input Low Voltage		Vss – 0.5	—	0.8	V
V_{IH}	Input High Voltage		2.0	—	Vcc+1	V
I_{IL}[1]	Input or I/O Low Leakage Current	$0V \le V_{IN} \le V_{IL}$ (MAX.)	—	—	–100	µA
I_{IH}	Input or I/O High Leakage Current	$3.5V \le V_{IN} \le V_{CC}$	—	—	10	µA
V_{OL}	Output Low Voltage	I_{OL} = MAX. Vin = V_{IL} or V_{IH}	—	—	0.5	V
V_{OH}	Output High Voltage	I_{OH} = MAX. Vin = V_{IL} or V_{IH}	2.4	—	—	V
I_{OL}	Low Level Output Current		—	—	12	mA
I_{OH}	High Level Output Current		—	—	–2	mA
I_{OS}[2]	Output Short Circuit Current	V_{CC} = 5V V_{OUT} = 0.5V T_A = 25°C	–30	—	–150	mA
I_{CC}	Operating Power Supply Current	V_{IL} = 0.5V V_{IH} = 3.0V, f_{toggle} = 15MHz Outputs Open, L-7/-10	—	75	130	mA

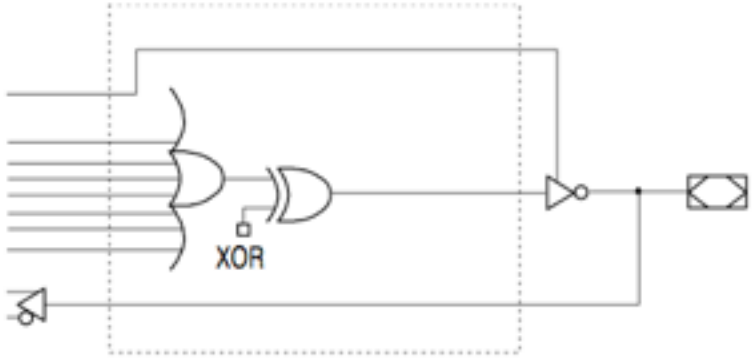
GAL

- Configurable to registered- and combinatorial outputs
  - Cover both L- and R- type PALs

www.latticesemi.com



Registered configuration



Combinatorial configuration

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
  - Generates all the minterms
  - 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)

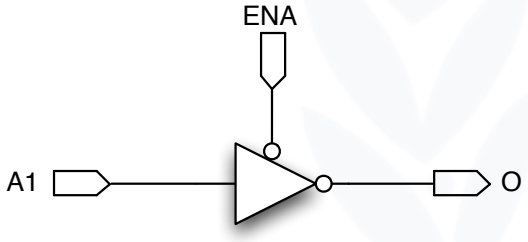
Regular Logic Structures for Multi-Level Logic

- 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
  - 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

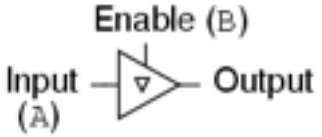
Non-Logic Gates

- Tri-state output
  - Tri-State is a TM, and Three-State is the real name
  - Neither 0 nor 1, high impedance
    - Floating
  - Use of symbol Z

  - In general, output enable is active low



Tristate buffer symbol



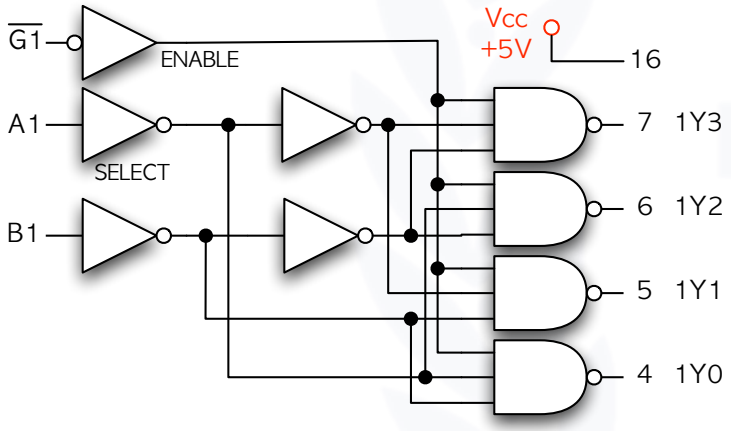
Truth table

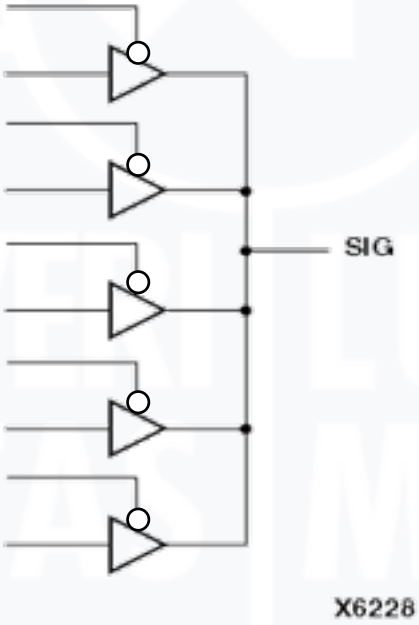
A	B	Output
0	0	High-Z
0	1	0
1	0	High-Z
1	1	1

  - More than two outputs can share the same signal line in a time-division multiplexing manner

Non-Logic Gate

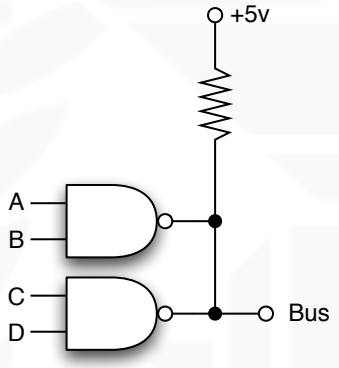
- Tri-state gate multiplexer
  - Combination of a decoder and tri-state gates



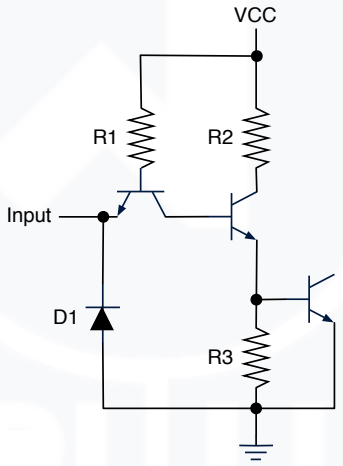


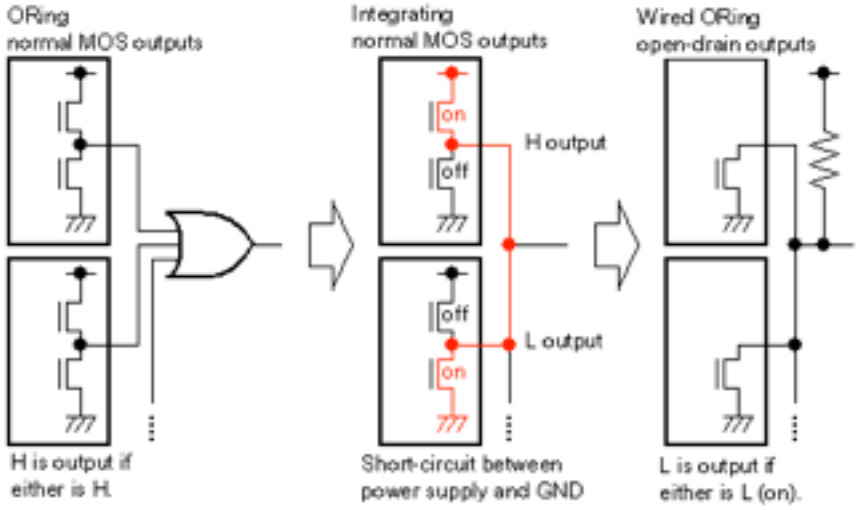
Open-Collector Output

- Bipolar open-collector gate
- CMOS open-drain gate
- No pull-up transistor
  - Use of a pull-up resistor
- Wired logic



Inverter circuit with open-collector output





Combinational 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

Logic Design Practice in a Lab

- Design Entry
  - Truth table
  - Boolean equation
  - Schematic diagram
    - Random logic and fixed logic (built-in library)
    - Programmable logic (GAL): currently, combinational only
- Logic simulation
  - Xilinx ISE 9.2
- Implementation
  - TTL 74 series
  - PALCE16V8 (GAL16V8)
  - Other passive components

Digital Logic Design

4190.201

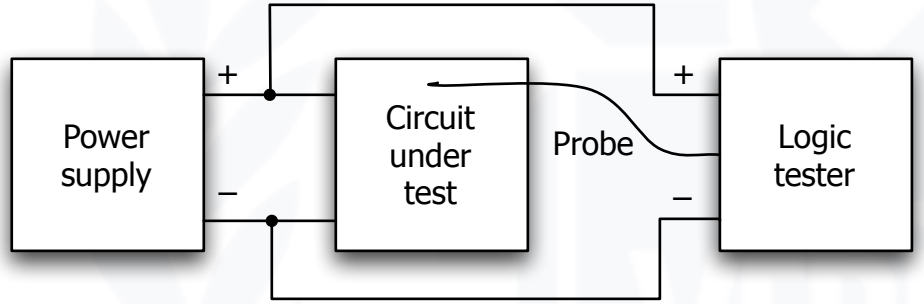
2014 Spring Semester

Supplementary Slide for the lab

Naehyuck Chang
Dept. of CSE
Seoul National University
naehyuck@snu.ac.kr

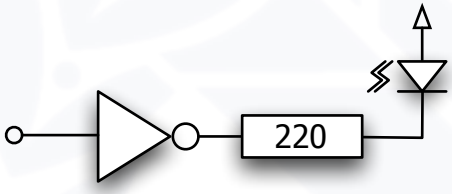
Logic Tester

- Detect high and low
  - Share the same power supply
  - Share the reference voltage, i.e., GND

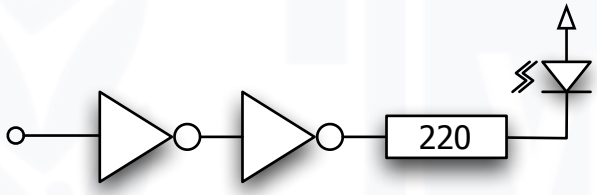


Logic Tester

- Detect high
  - Should have no loading effect

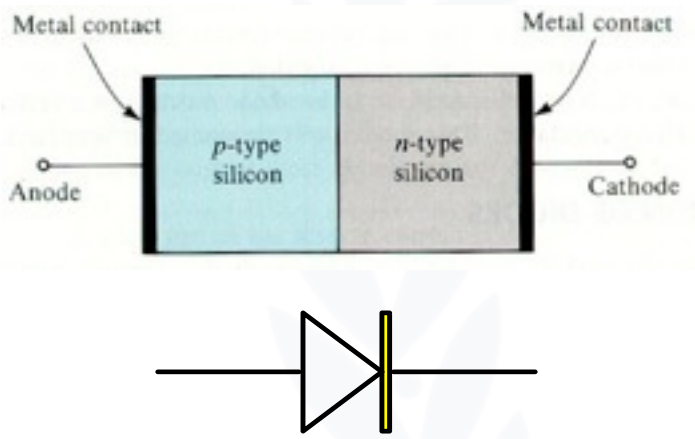


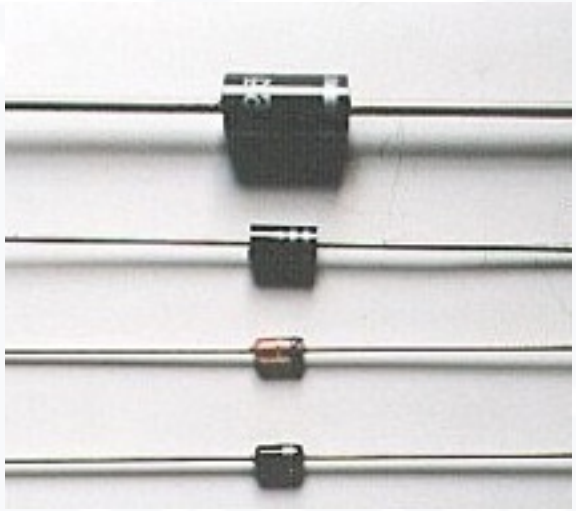
- Detect low
  - Should have no loading effect



Diode (1)

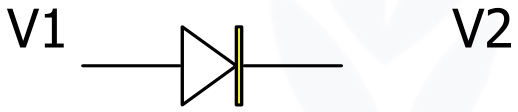
- P-N junction diode
  - 1N4148

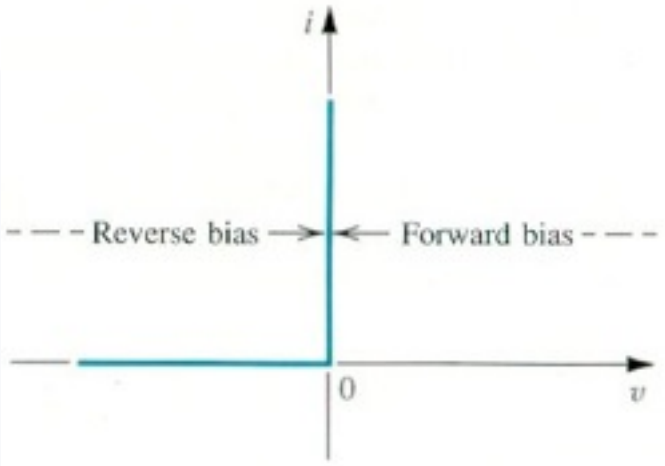




Diode (2)

- Ideal diode
  - Forward bias: V1 > V2
  - Reverse bias: V1 < V2





Diode (4)

- Real diode
  - Break down
- Silicon Diode
- Germanium Diode

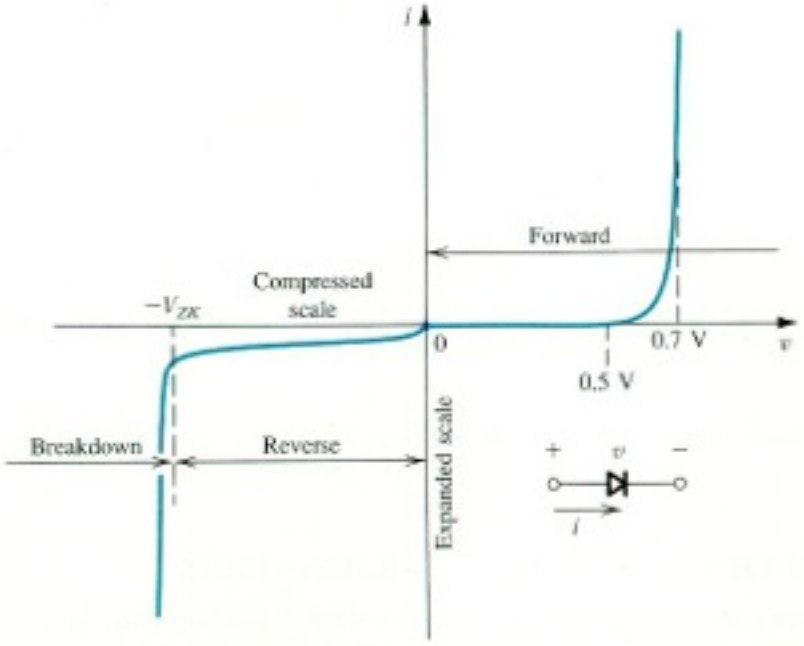
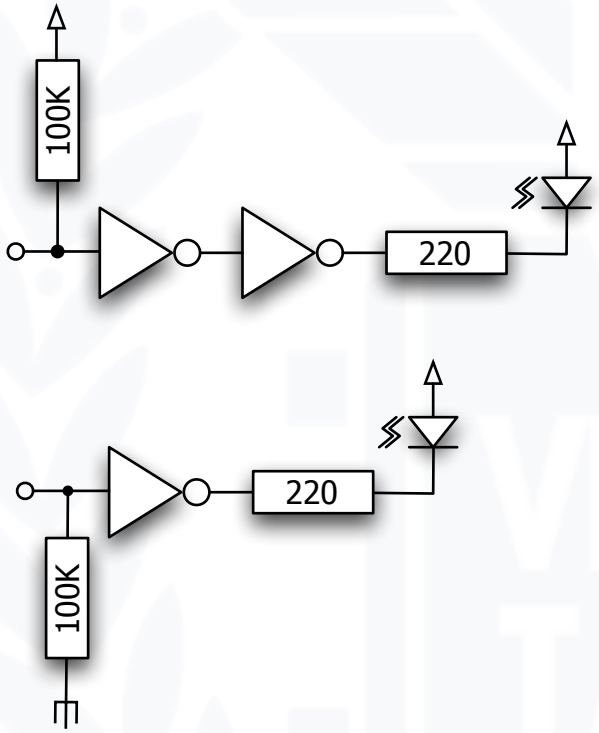


Fig. 3.8 The diode i–v relationship with some scales expanded and others compressed in order to reveal details.

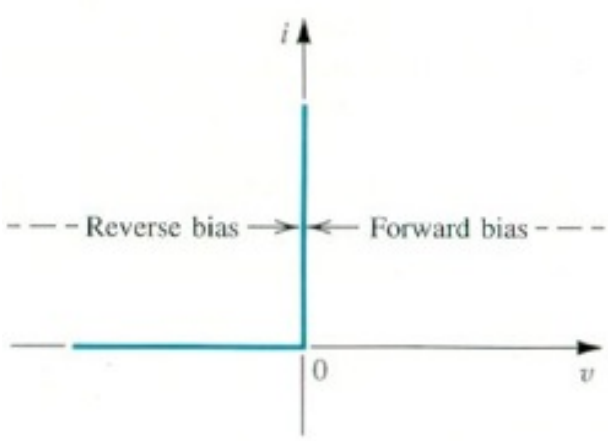
Logic Tester

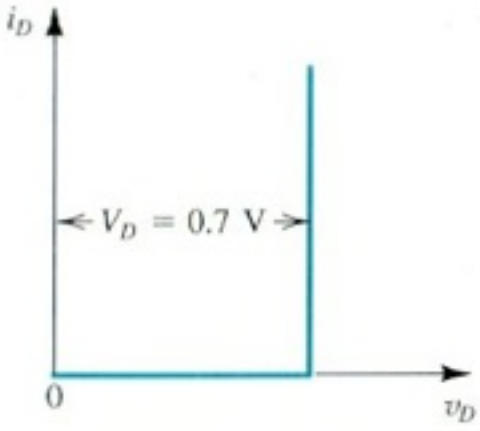
- Prevent from floating
  - Can we combine these together?

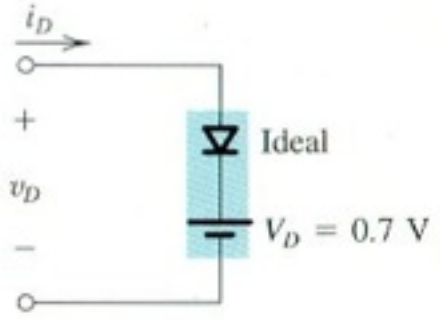


Diode (6)

- Diode in digital circuits
  - Silicon Diode: switching and rectification
  - Germanium Diode: detection







Logic Tester

- Final design