CSS 422 Hardware and Computer Organization

Digital Logic

Professor: Yang Peng

The slides are re-produced by the courtesy of
Dr. Arnie Berger and Dr. Wooyoung Kim
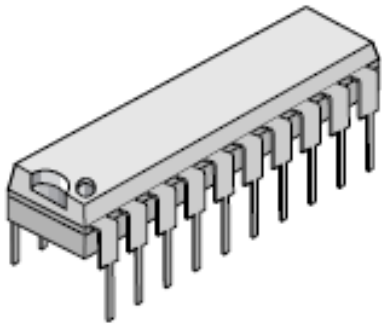
Topic

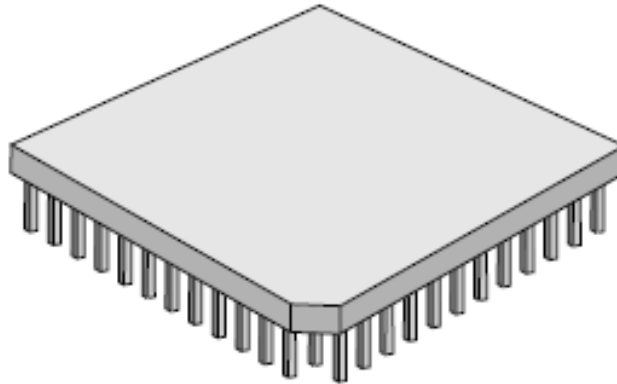Chapter 1, 2, 3 by Berger
Chapter 3 by Null

- Digital Logic and Combinational Circuit
    - Logic and Gates
    - K-maps and Boolean equation
    - Combinational Circuit Design

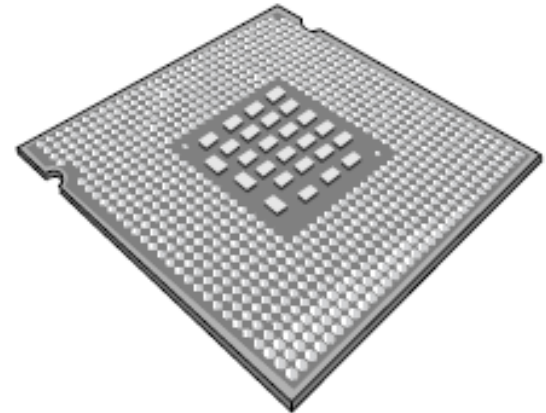# Integrated Circuits

- Integrated Circuits (IC) consist of **gates**
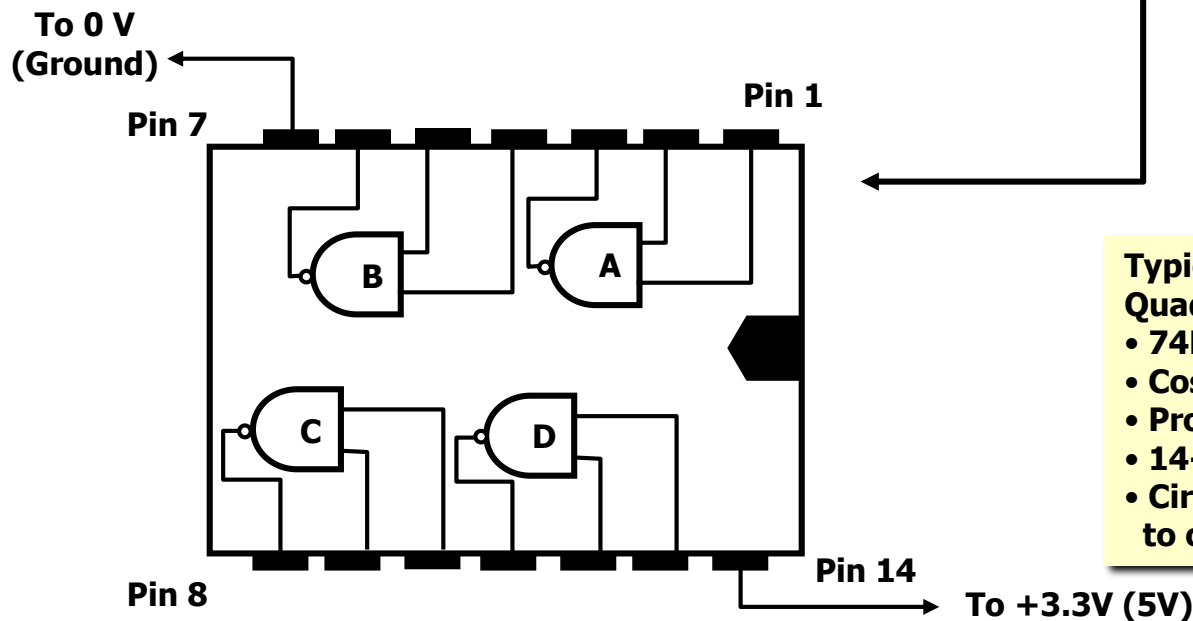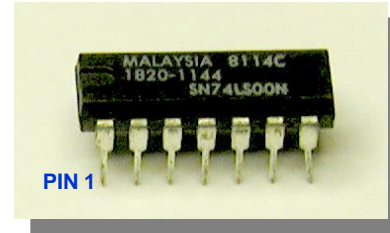

(a)　　　　　　(b)　　　　　　(c)

- Gate is an electronic circuit: the fundamental building blocks of digital systems
- Simple gates are combined to create more complex functions

3

NAND Gate Example

To 0 V (Ground)

Pin 7

Pin 1

B   A

C   D

Pin 8

Pin 14

To +3.3V (5V)

Typical NAND gate circuit
Quad, 2-input NAND gate
- 74LS00
- Cost: ~ $0.10
- Propagation delay ~ 5 nsec
- 14-pin Dual in-line package (DIP)
- Circuits vary from *8* pin DIP packages to over *600* pin high-density packages

Gate and Logic

- Digital computers use the binary system
  - ALL digital computers are **collections of switches made from transistors**
  - A switch is **ON** or **OFF**
- **Principles of Logic** (a branch of Philosophy) are useful to describe the digital circuits in computers
  - True/False, 1/0, On/OFF, High/Low all describe the same possible states of a digital system

on/off switch

+

A   B   C

Battery Symbol

C = A and B

-   Light bulb (load)

A —

AND   — C   C = A•B

B —

5
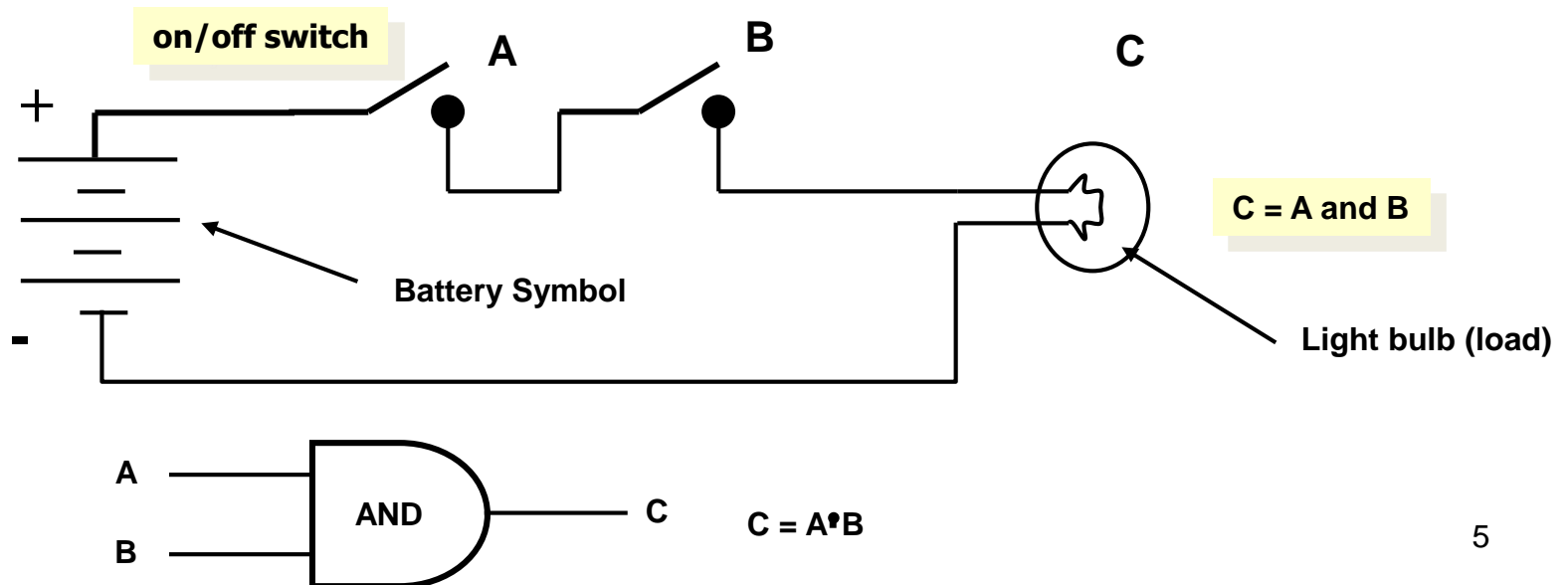
Logical Gates

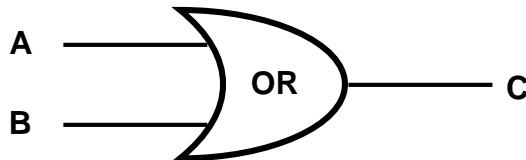- The **_Gate_** is the basic element of all digital systems
  - Three types of gates form the "**atomic**" elements

A
B
AND
C

$C = A \bullet B$
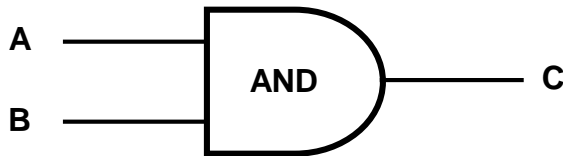
C is _TRUE_ if A is _TRUE_ AND B is _TRUE_

A
B
OR
C

$C = A + B$

C is _TRUE_ if A is _TRUE_ OR B is _TRUE_

NEGATION SYMBOL

A
NOT
B
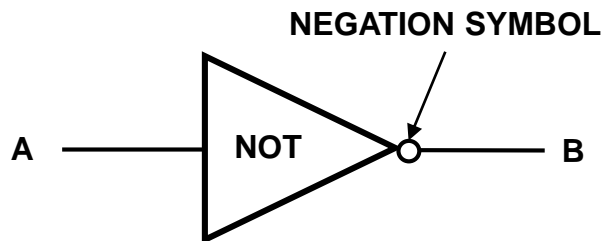
$B = \overline{A}$
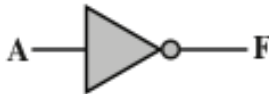
B is _TRUE_ if A is _FALSE_

Exclusive OR (XOR) Gate

A · B̄

Ā · B

C = A · B̄ + Ā · B

C = A ⊕ B

A
B
XOR
C

C = A ⊕ B

Physical Connection
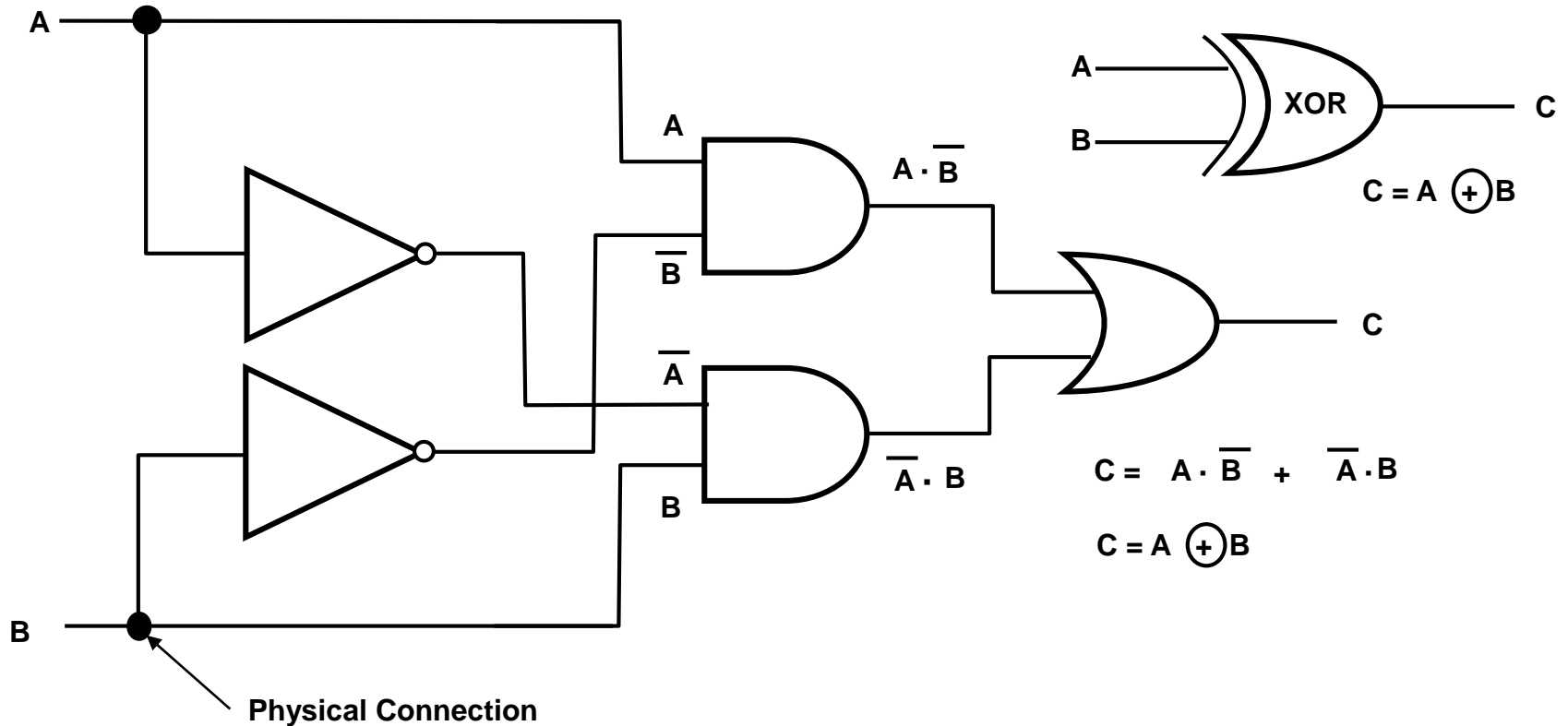
C is FALSE if A = B

Logic and Gate

Name	Graphical Symbol	Algebraic Function	Truth Table
AND	A, B → F (AND gate)	$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR	A, B → F (OR gate)	$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT	A → F (NOT gate)	$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND	A, B → F (NAND gate)	$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	A, B → F (NOR gate)	$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR	A, B → F (XOR gate)	$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

8

Logical DNA

Basic Identities of Boolean Algebra

Basic Postulates

$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements

Other Identities

$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

From Circuit to Boolean Equation

- We can easily interpret the function of a given circuit

A

A

$A \cdot \overline{B}$

$\overline{B}$

$\overline{A}$

C

$A \cdot \overline{B} \ + \ \overline{A} \cdot B$

$\overline{A} \cdot B$

B

B
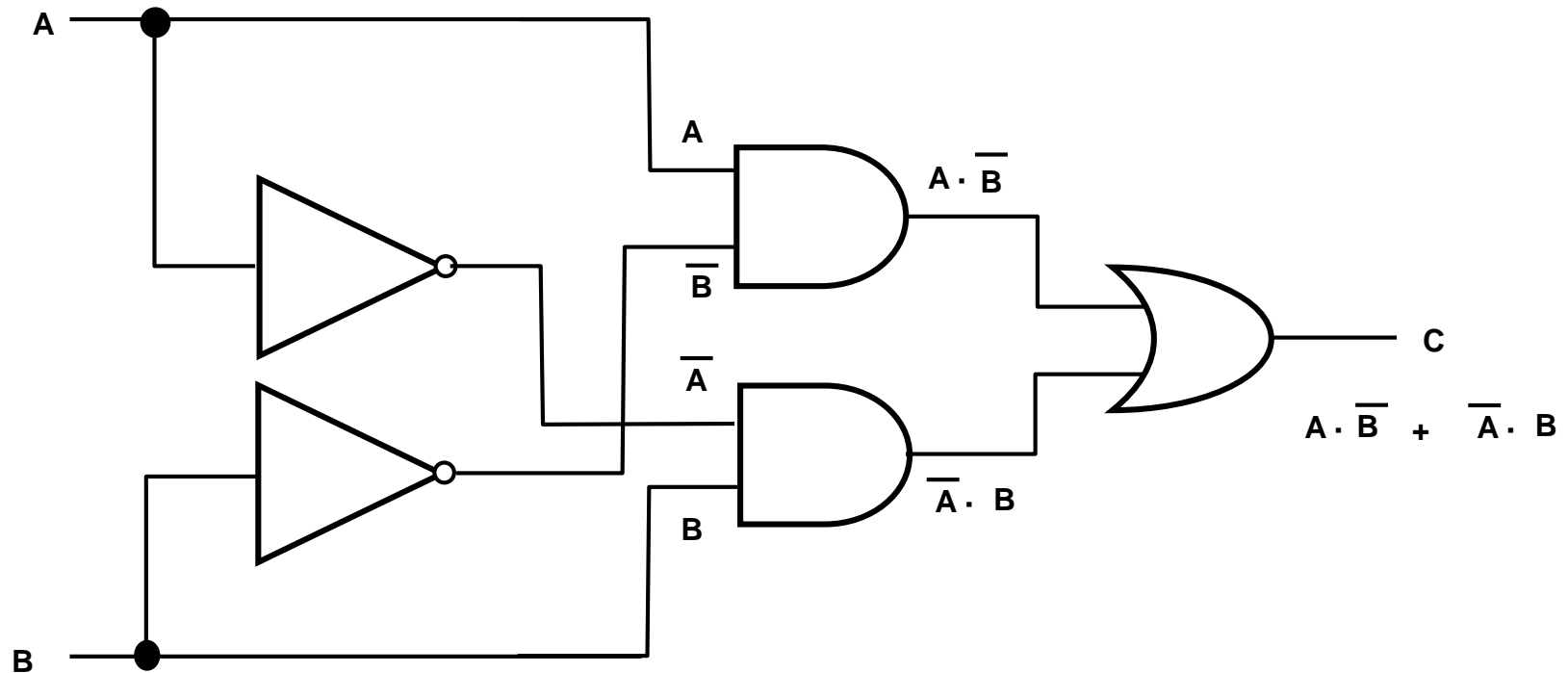
$C = A \oplus B$

How to build a circuit with some given logic?

Designing a Digital System

- We can use the concept of a **Truth Table** as a design tool
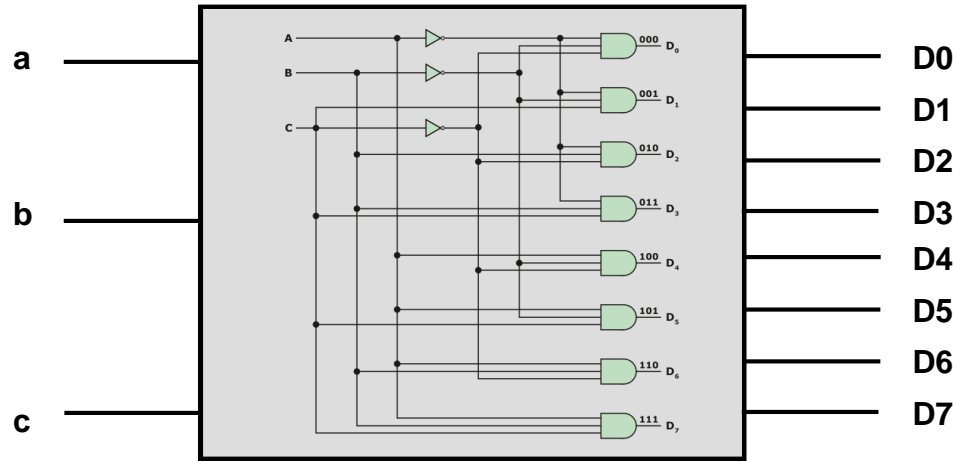
3 X 8 line decoder

- Suppose that we want to design a digital system, such as the above "line decoder" (used for line selection)

$$abc = 000 \rightarrow D0 = 1 \qquad abc = 100 \rightarrow D4 = 1$$
$$abc = 001 \rightarrow D1 = 1 \qquad abc = 101 \rightarrow D5 = 1$$
$$abc = 010 \rightarrow D2 = 1 \qquad abc = 110 \rightarrow D6 = 1$$
$$abc = 011 \rightarrow D3 = 1 \qquad abc = 111 \rightarrow D7 = 1$$

Design a Circuit

- Designing a digital system starts from designing a chip, which consists of a number of gates (simple or combinational)
- Gates are composed of, **Inputs**, **Outputs** and **Logic Gates** (the tree atomic gates, AND, OR, NOT)
- How to start?

1. Based on the purpose of the system, decide the **number of inputs** and the **number of outputs** (inputs and outputs are binary: on / off)
   - Figure out how many bits needed for input and output
2. Draw a **truth table** for each input combination and output combination
   - For example, how many combinations for three bits of input? Derive a Boolean equation for each output
3. **Connect inputs to outputs** using **logic gates**
4. **Test** if your circuit matches with the design – the logic in truth table

14

Design a 3 x 8 Line Decoder

1. Decide the number of inputs and the number of outputs (inputs and outputs are binary)
   - Three bits for input, eight bits for output
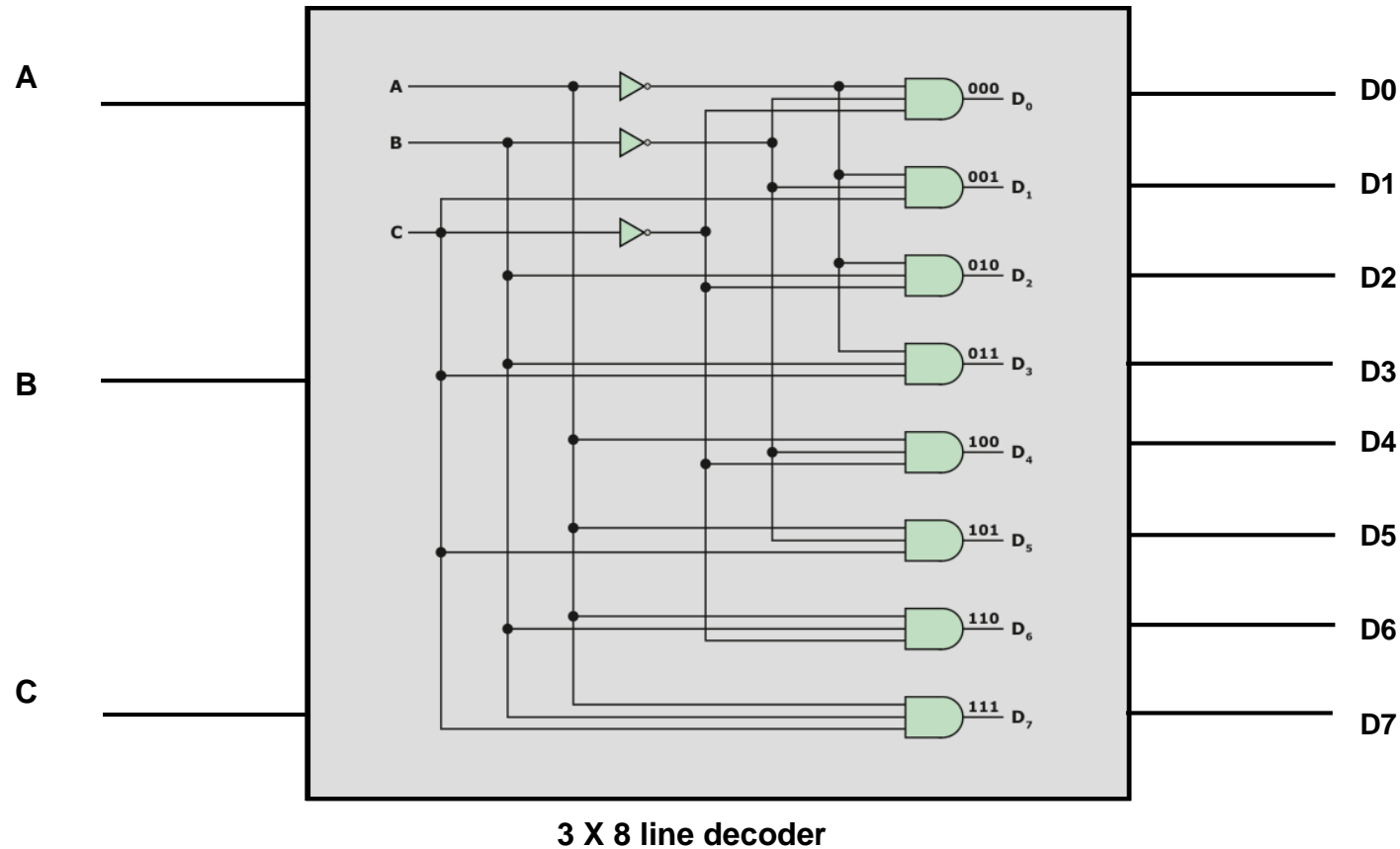
3 X 8 line decoder

Design a 3 x 8 Line Decoder

2. Draw a truth table for each input combination and output combination

abc = 000 → D0 =1
abc = 001 → D1 =1
abc = 010 → D2= 1
abc = 011 → D3= 1

abc = 100 → D4 =1
abc = 101 → D5 =1
abc = 110 → D6= 1
abc = 111 → D7= 1

a	b	c	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Design a 3 x 8 Line Decoder

3. Derive a Boolean equation for each output

D0 =

D1 =

D2=

D3 =

D4 =

D5 =

D6 =

D7 =

a	b	c	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Design a 3 x 8 Line Decoder

- Obtain Boolean equation for each output

D0 = ~a ~b ~c

D1 = ~a ~b c

D2 = ~a b ~c

D3 = ~a b c

D4 = a ~b ~c

D5 = a ~b c

D6 = a b ~c

(I will use ~ instead of – , easier to type…)

D7 = a b c

Design a 3 x 8 Line Decoder

4. Connect inputs to out using logic gates

$D0 = \sim a \sim b \sim c$

$D1 = \sim a \sim b\ c$

$D2 = \sim a\ b \sim c$

$D3 = \sim a\ b\ c$

$D4 = a \sim b \sim c$

$D5 = a \sim b\ c$

$D6 = a\ b \sim c$

$D7 = a\ b\ c$

19
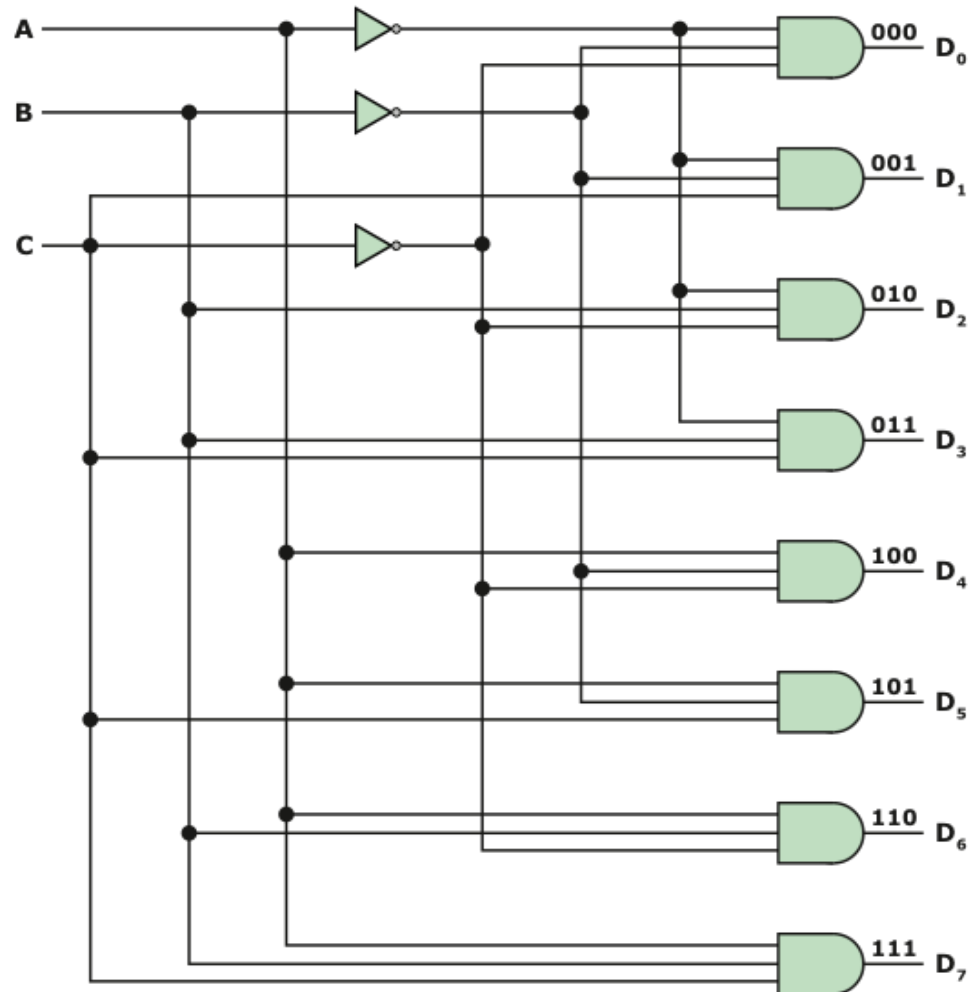
Design a 3 x 8 Line Decoder

5. Test! (Use Logisim)

abc = 000 → D0 =1
abc = 001 → D1 =1
abc = 010 → D2= 1
abc = 011 → D3= 1

abc = 100 → D4 =1
abc = 101 → D5 =1
abc = 110 → D6= 1
abc = 111 → D7= 1

20
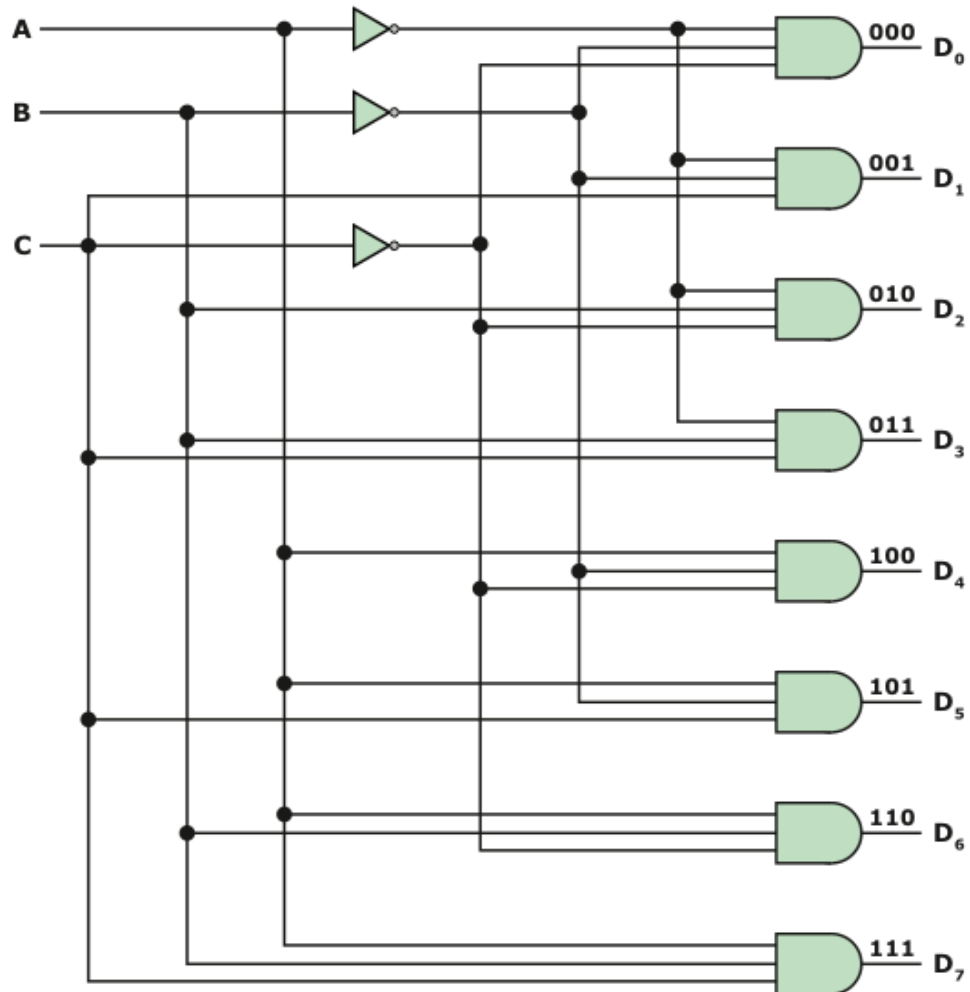
Another Example

A	B	C	x
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- Suppose you have to design a circuit system that produces the output X based on the three inputs: A, B, and C

- Draw the circuit diagram for this system

X = ~A~B~C + A~B~C + AB~C + A~BC

X = ~B~C + A(B~C+~BC)

X = (~A~B~C + A~B~C + A~B~C + A~B~C) + A(B~C+~BC)

X = (~A~B~C + A~B~C) + (AB~C + A~B~C) + (A~BC + A~B~C)

x = $\overline{B}*\overline{C}$ + A*$\overline{B}$ + A*$\overline{C}$

Please Review Boolean Algebra!

Null's Chapter 3.2
Berger's Chapter 3

Karnaugh Maps

- We want to systematically derive a Boolean equation from a truth table
- The Karnaugh Map (pronounced "car know") is a graphical method of simplifying the "Sum of Products" (minterm) truth table
- **Based upon Boolean algebra simplification $A*B + A*\overline{B} = A$**
  - Why?
    - First Distributive Law: $A *(B + C) = (A*B) + (A*C)$
    - Third Law of Complementation: $B + \overline{B} = 1$
    - Finally: $A*1 = A$
- Therefore
  $A*B + A*\overline{B} = A * (B + \overline{B}) = A * 1 = A$
- http://sourceforge.net/projects/k-map/files/k-map/0.4/Kmap-04-setup.exe/download

Karnaugh Maps – 2

- For **each output**, you have to build **one k-map**
- For the 3 x 8 line decoder, we need eight k-maps
- Rules for building a Karnaugh Map of a truth table
  - **Rule 1**: # cells = # of possible combinations of input variables
    - The number of cells = $2^{(\text{NUMBER OF INPUT VARIABLES})}$
    - For example,
      - 3 input variables: A, B, C = 2^3 = 8 cells
      - 4 input variables: A, B, C, D = 2^4 = 16 cells
      - 5 input variables: A, B, C, D, E = 2^5 = 32 cells
  - **Rule 2**: The horizontal and vertical axes are labeled such that **only one variable changes from complemented to un-complemented** (or vice versa) as you go across or down
    - The **first** and **last** cells in a row or in a column are considered to be **adjacent to each other**

Karnaugh Maps – 3

K-Map for 3 input variables

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$				
C				

K-Map for 4 input variables

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\overline{D}$				
$\overline{C}D$				
CD				
$C\overline{D}$				

K-Map for 5 input variables

	$\overline{A}\overline{B}\overline{C}$	$\overline{A}\overline{B}C$	$\overline{A}B\overline{C}$	$\overline{A}BC$	$A\overline{B}\overline{C}$	$A\overline{B}C$	$AB\overline{C}$	ABC
$\overline{D}\overline{E}$								
$\overline{D}E$								
DE								
$D\overline{E}$								

Note:
1- The colored lines are actually adjacent to each other
2- The corner cells are adjacent
3- The variables are organized so that only one variable changes as you go from cell to cell, including the opposite ends
4- Diagonals are not adjacent

Simplification using K-Maps

1. Construct **one Karnaugh Map for each output variable**
2. Place a "1" in every cell that has a 1 in the corresponding row of the truth table

3. Form **the largest possible loops** of cells containing 1, 2, 4, 8,16, etc., *adjacent* "1" terms
4. **Any cell can be involved in any number of loops**, but **each new loop must contain at least one entry that is not contained in any other loop**, in order to avoid a redundant loop
   - **"loop within loop"** is **NOT ALLOWED!**
5. Inspect the map for any loops whose terms are all enclosed in other loops and remove those loops
6. Each loop represents a simplified minterm (sum of product) of the logic equation
   - Simplify the loop by **removing any variable that appears in its complemented and un-complemented form**

Design a 3 x 8 Line Decoder

3. Derive a Boolean equation for each output

K-Map for 3 input variables

	$\bar{a}\,\bar{b}$	$a\,\bar{b}$	ab	$\bar{a}\,b$
$\bar{c}$	1			
c				

Focus on one output (D0) at a time

1. Fill out the cells which have 1 only (others are zero)

2. Read the cell with 1: $D0 = \bar{a}\ \bar{b}\ \bar{c}$

Note: $\overline{ab}$ is not equal to $\bar{a}\,\bar{b}$

$\overline{ab} = \bar{a} + \bar{b}$

a	b	c	D0
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Design a 3 x 8 Line Decoder

- Do it for each output

K-Map for 3 input variables

	$\overline{a}\,\overline{b}$	$a\,\overline{b}$	ab	$\overline{a}\,b$
$\overline{c}$				
c	1			

D1 = ~a ~b c

(I will use ~ instead of —)

a	b	c	D1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

K-Map Examples

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$	1	1		1
$\overline{C}D$	1	1		1
CD			1	
$C\overline{D}$			1	
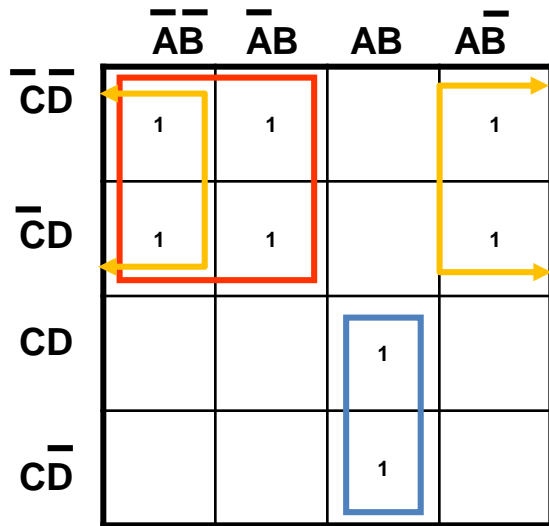
This K-Map has three loops. Notice that the yellow
 loop is actually adjacent.
The equation is **X = ~A*~C + ~B*~C + A*B*C**

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$	1			1
$\overline{C}D$				
CD				
$C\overline{D}$	1			1
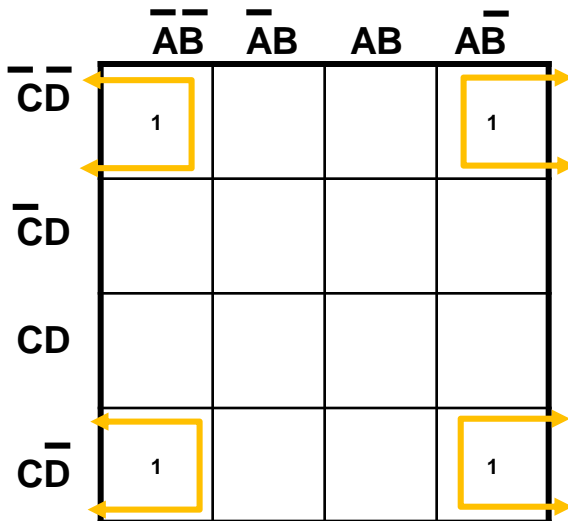
This K-Map first appears to have two loops. The diagonal are not adjacent,
but by first rotating the map around the vertical axis and then around a
horizontal axis, we can cluster the four terms into a single 2 by 2 loop.

The equation is **X = ~B*~D**

29

K-Map Examples – 2

This K-Map can really be simplified.

X = ~C

Here we have two loops. The blue loop wraps around the top and bottom edges, enclosing 8 terms, and the red loop encloses one column of 4 terms. The equation is

X = ~D + ~A*B

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	A$\overline{B}$
$\overline{C}\,\overline{D}$	1	1	1	1
$\overline{C}D$	1	1	1	1
CD				
C$\overline{D}$				

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	A$\overline{B}$
$\overline{C}\,\overline{D}$	1	1	1	1
$\overline{C}D$		1		
CD		1		
C$\overline{D}$	1	1	1	1

Simplification using K-Map

A	B	C	x
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$	1		1	1
C				1

$$x = \overline{B}*\overline{C} + A*\overline{B} + A*\overline{C}$$

Simplified equation

Designing the Hardware

$$x = \overline{B}*\overline{C} + A*\overline{B} + A*\overline{C}$$

X
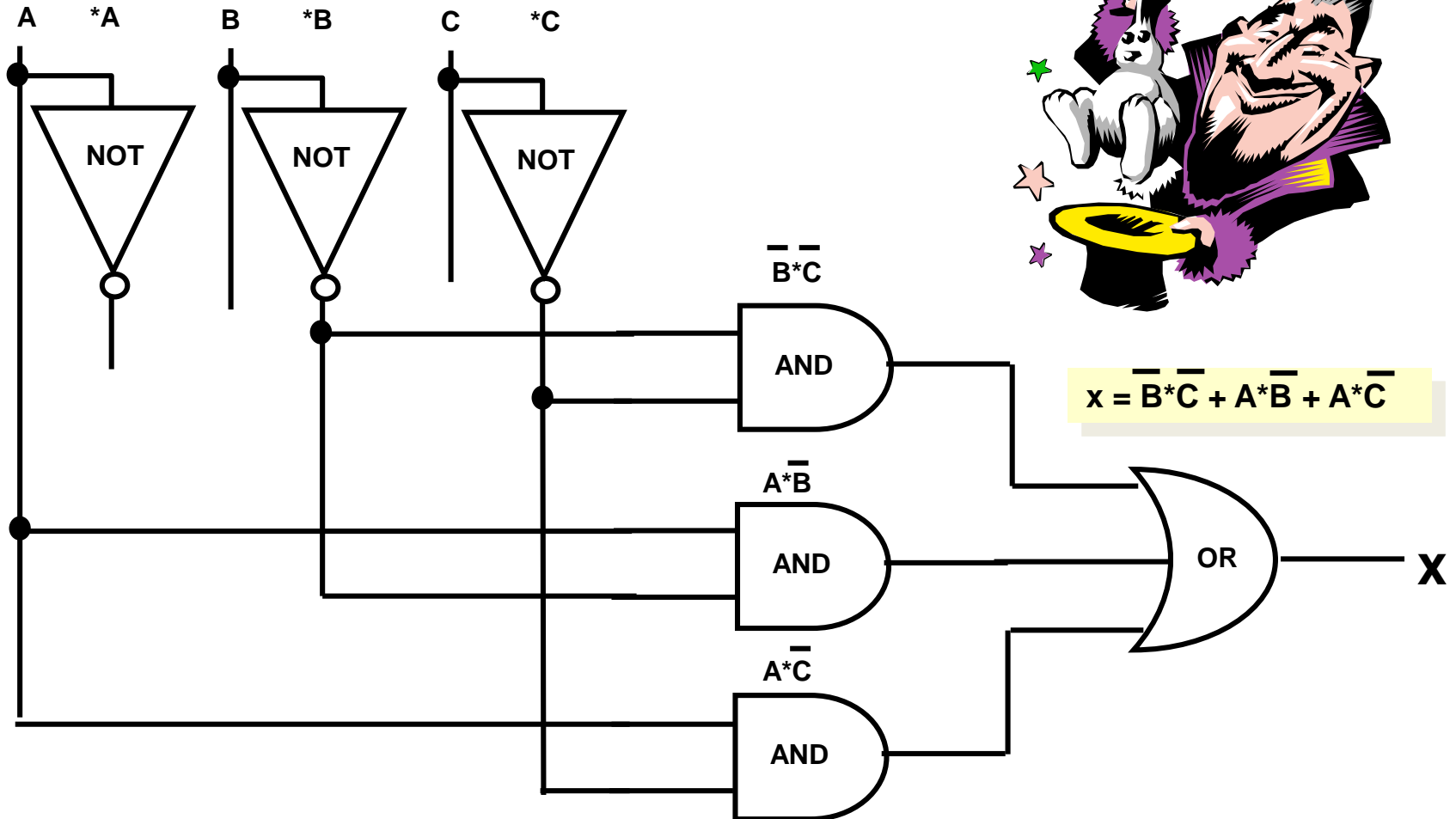
Prove Algebraically

Step 1: $x = \bar{A}*\bar{B}*\bar{C} + A*\bar{B}*\bar{C} + A*B*\bar{C} + A*\bar{B}*C$ //From Truth Table

Step 2: $x = \bar{A}*\bar{B}*\bar{C} + A*B*\bar{C} + A*\bar{B}(C + \bar{C})$ // First Law of Distribution

Step 3: $x = \bar{A}*\bar{B}*\bar{C} + A*B*\bar{C} + A*\bar{B}$ // Law of Complementation

Step 4: $x = \bar{A}*\bar{B}*\bar{C} + A(B*\bar{C} + \bar{B})$ // First Law of Distribution

Step 5: $x = \bar{A}*\bar{B}*\bar{C} + A[(\bar{B}+\bar{C})*(\bar{B} + B)]$ // Second Law of Distribution

Step 6: $x = \bar{A}*\bar{B}*\bar{C} + A(\bar{B}+\bar{C})$ // Law of Complementation

Step 7: $x = \bar{A}*\bar{B}*\bar{C} + A*\bar{B} + A*\bar{C}$ // First Law of Distribution

Step 8: $x = \bar{B}(\bar{A}*\bar{C} + A) + A*\bar{C}$ // First Law of Distribution

Step 9: $x = \bar{B}[(\bar{A}+A)*(\bar{C} + A)] + A*\bar{C}$ // Second Law of Distribution

Step 10: $x = \bar{B}(\bar{C} + A) + A*\bar{C}$ // First Law of Distribution

Step 11: $x = \bar{B}*\bar{C} + \bar{B}*A + A*\bar{C}$ // First Law of Distribution

Another Example

A	B	C	D	X
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	1
0	0	1	0	0
1	0	1	0	1
0	1	1	0	0
1	1	1	0	1
0	0	0	1	1
1	0	0	1	0
0	1	0	1	1
1	1	0	1	0
0	0	1	1	1
1	0	1	1	0
0	1	1	1	1
1	1	1	1	0

1. X = A*B*~C*~D + A*~B*C*~D + A*B*C*~D + ~A*~B*~C*D + ~A*B*~C*D + ~A*~B*C*D + ~A*B*C*D
2. X = A*B*~D*(~C + C) + A*~B*C*~D + ~A*~C*D* (~B+ B) + ~A*C*D*(B + ~B)
3. X = A*B*~D + A*~B*C*~D + ~A*D*(C + ~C)
4. X = A*B*~D + A*~B*C*~D + ~A*D
5. X = A*~D*(B + ~B*C) + ~A*D
6. X = A*~D*(C + B) + ~A*D //Second law of Dist.

X = ~A*D + A*B*~D + A*C*~D

Revisit K-maps

- What if there is a "don't care" condition?

- The **output** could be either 0 or 1, but the value will not affect the system logic. So, we don't care about this output!

- Some system produces an output that we "Do not care!"

How to design this system with K-maps?

K-Map with "don't care" Condition

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\overline{D}$			x	
$\overline{C}D$		1	x	
CD		x	1	
$C\overline{D}$	x		1	

- X can be 0 or 1
- Choose so the expression can be as simple as possible.
- What's the Boolean equation?

K-Map with "don't care" Condition

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$			**1**	
$\overline{C}D$		1	**1**	
CD		**1**	1	
$C\overline{D}$	**0**		1	

- X can be 0 or 1
- Choose so the expression can be as simple as possible.
- What's the Boolean equation?
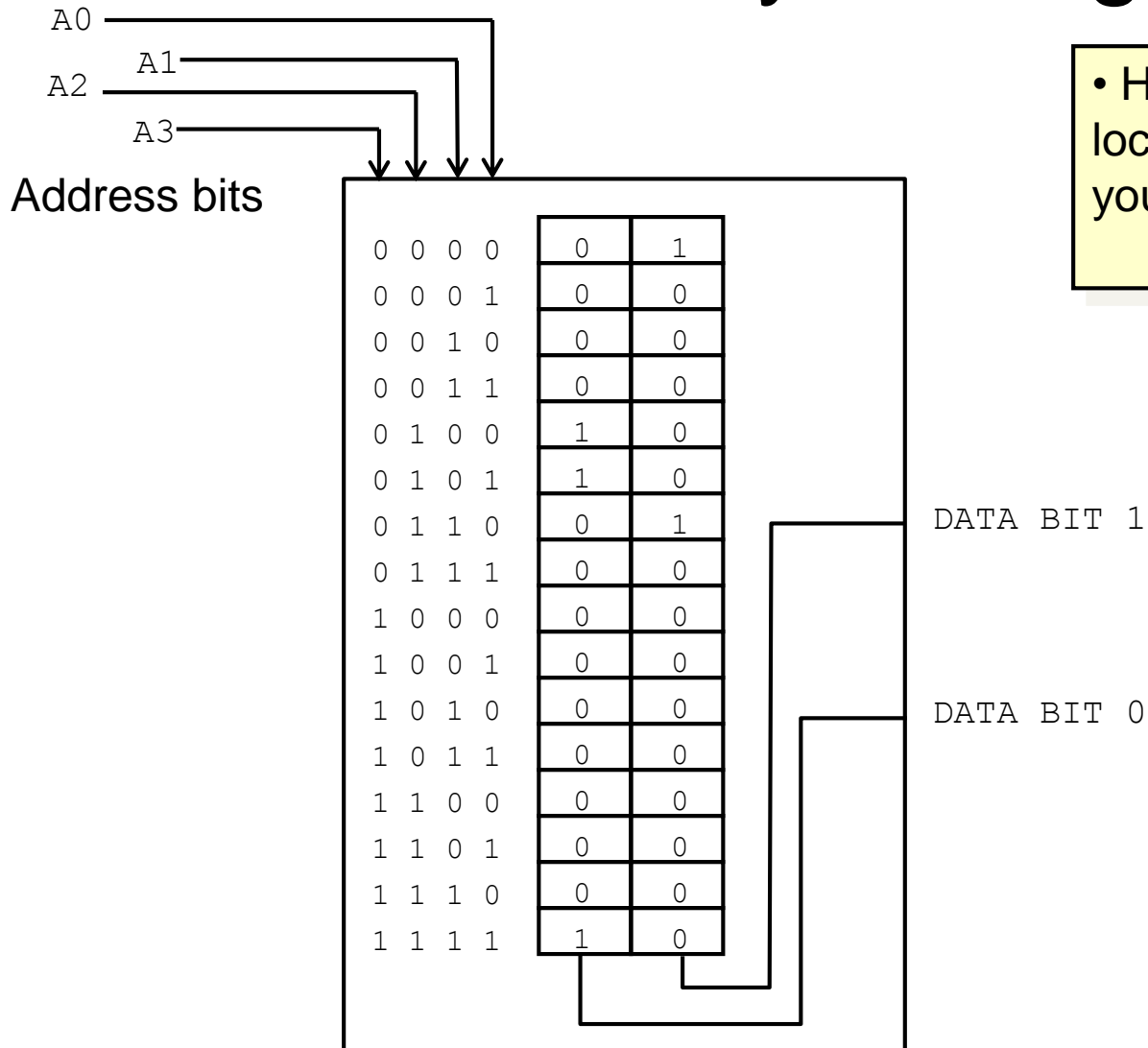
37

Read-Only Memory (ROM)

- Memory that is implemented with combinational circuits
  - Combinational circuits are often referred to as "memoryless" circuits because their output depends only on their current input and no history of prior inputs is retained

- Memory unit that performs only the read operation
  - Binary information stored in a ROM is permanent and is created during the fabrication process
  - A given input to the ROM (address lines) always produces the same output (data lines)
  - Because the outputs are a function only of the present inputs, ROM is a combinational circuit

Read-Only Memory (ROM)

Input					Output			
X_1	X_2	X_3	X_4		Z_1	Z_2	Z_3	Z_4
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		0	1	1	1
0	1	1	0		0	1	0	1
0	1	1	1		0	1	0	0
1	0	0	0		1	1	0	0
1	0	0	1		1	1	0	1
1	0	1	0		1	1	1	1
1	0	1	1		1	1	1	0
1	1	0	0		1	0	1	0
1	1	0	1		1	0	1	1
1	1	1	0		1	0	0	1
1	1	1	1		1	0	0	0

Truth Table for a ROM

Memory as Logic

A0

A1

A2

A3

Address bits

• How many memory locations can you have if you have four input bits?

0 0 0 0	0	1
0 0 0 1	0	0
0 0 1 0	0	0
0 0 1 1	0	0
0 1 0 0	1	0
0 1 0 1	1	0
0 1 1 0	0	1
0 1 1 1	0	0
1 0 0 0	0	0
1 0 0 1	0	0
1 0 1 0	0	0
1 0 1 1	0	0
1 1 0 0	0	0
1 1 0 1	0	0
1 1 1 0	0	0
1 1 1 1	1	0

DATA BIT 1

DATA BIT 0

40

More Examples of Combinational Circuit

Simple Multiplexer (MUX)

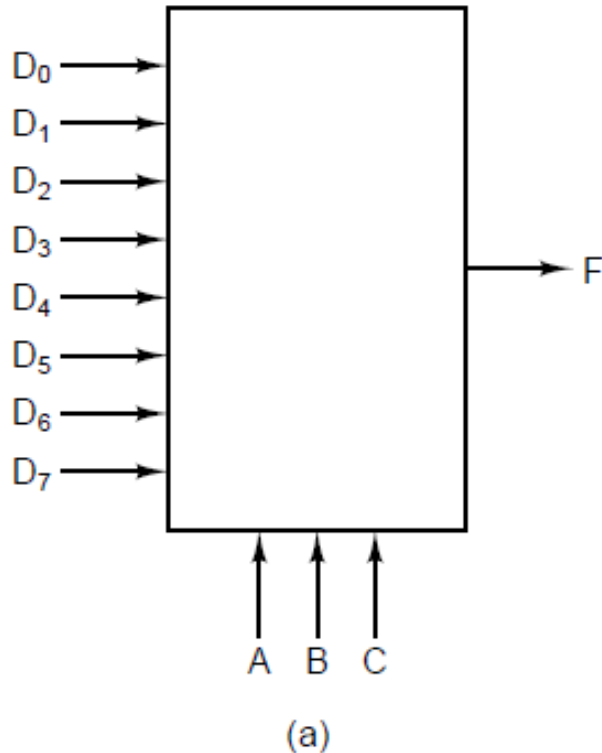- The MUX circuit allows one of two (or more) logical signals to be selected
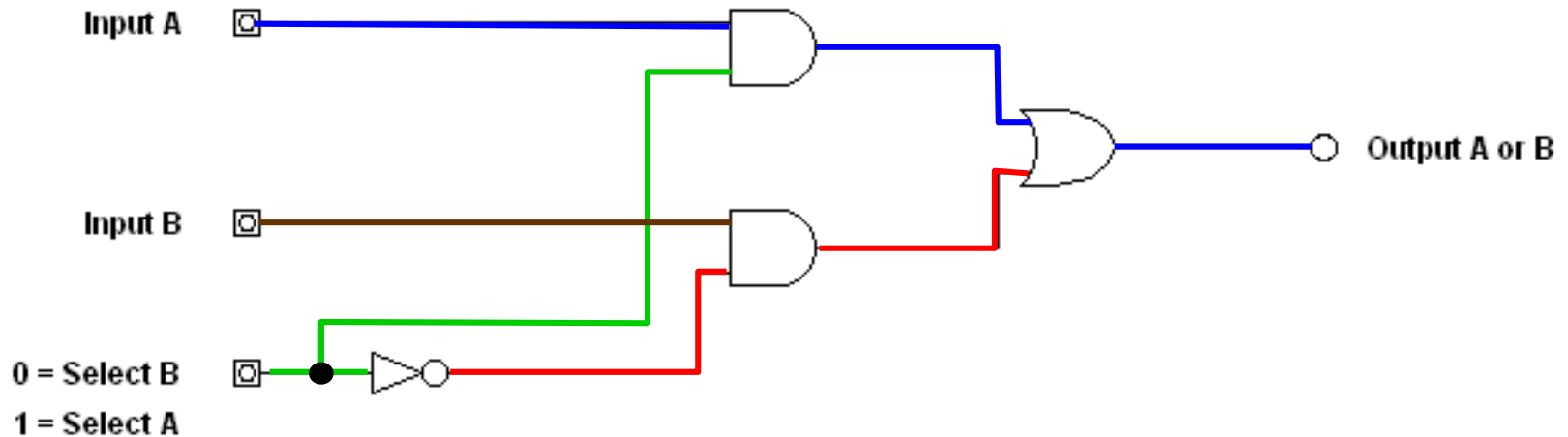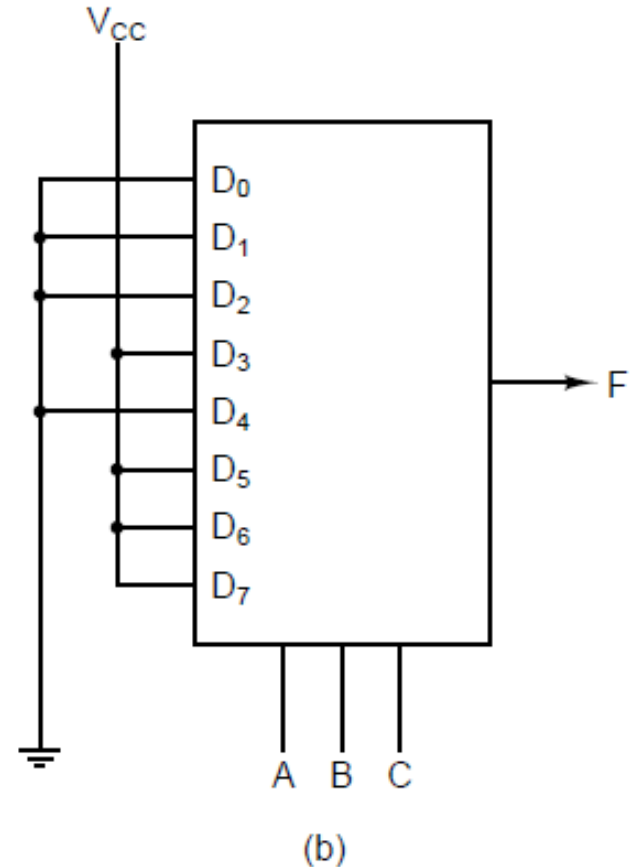
Multiplexers
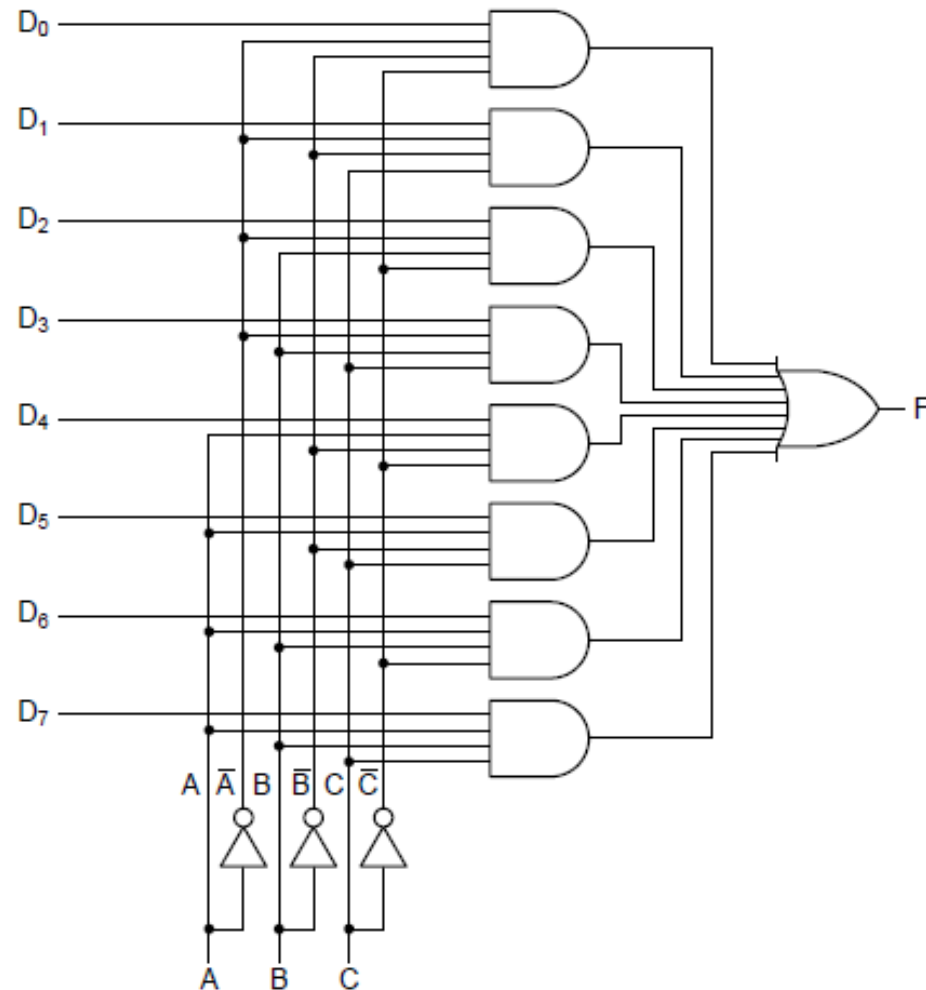
A, B, C are control lines to determine:

which one of the eight input lines to be gated.
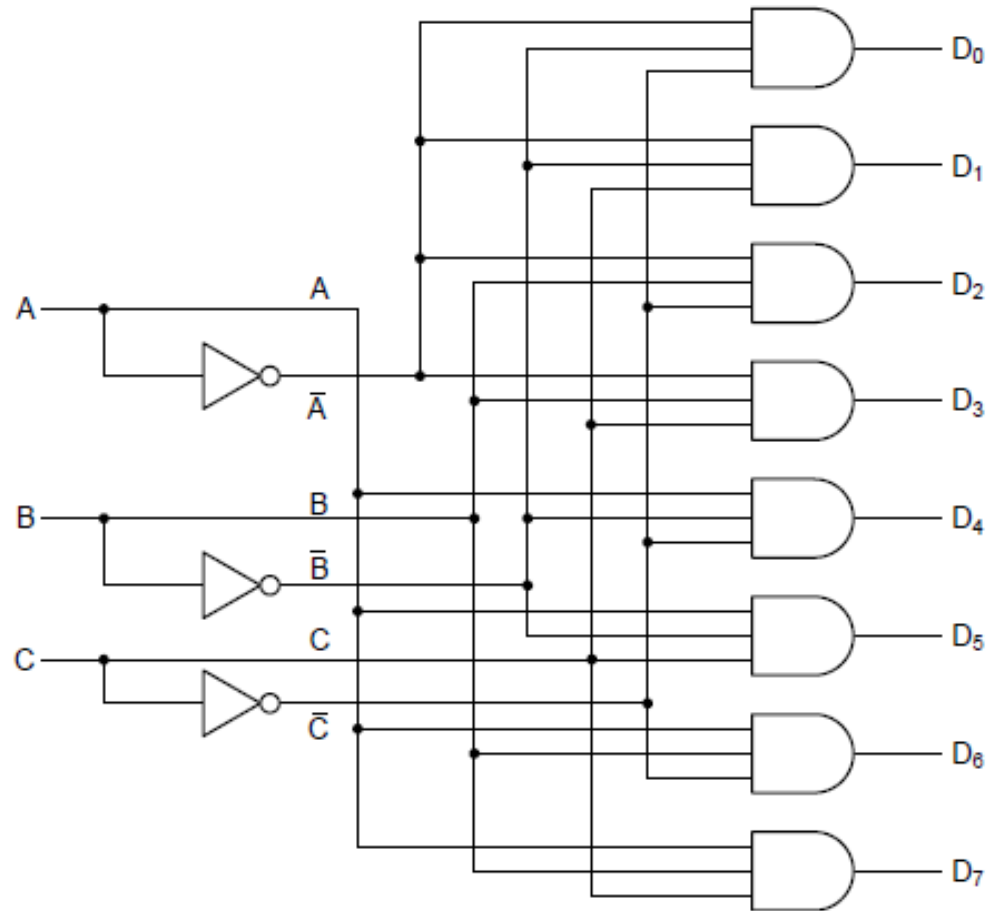
Multiplexers – cont'd

(a) An eight-input multiplexer.

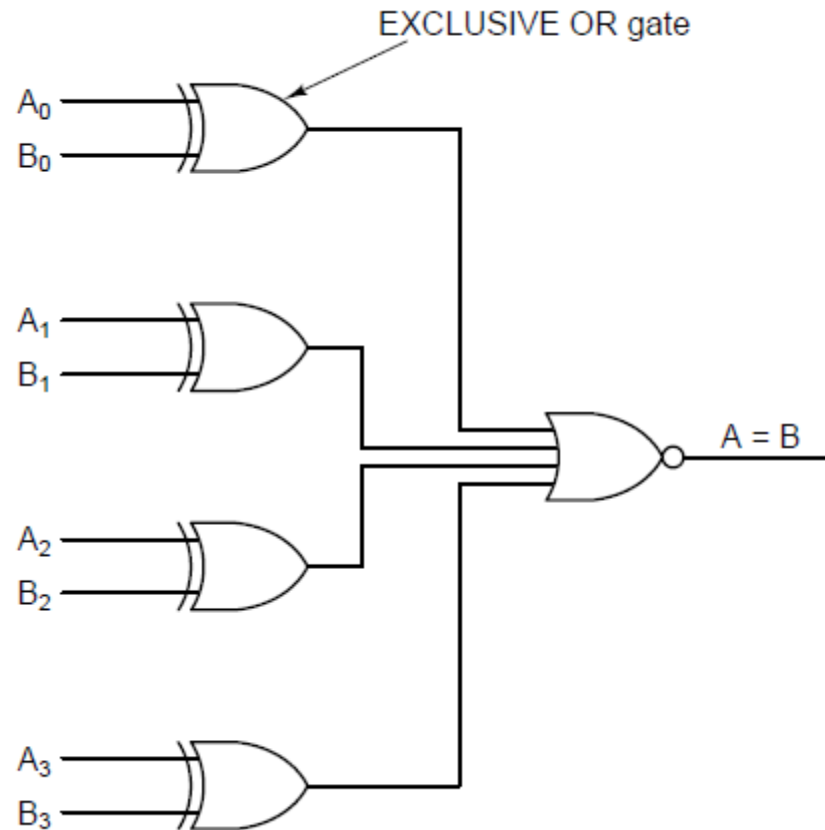(b) The same multiplexer wired to compute the majority function.

Decoders

A 3-to-8 decoder circuit:

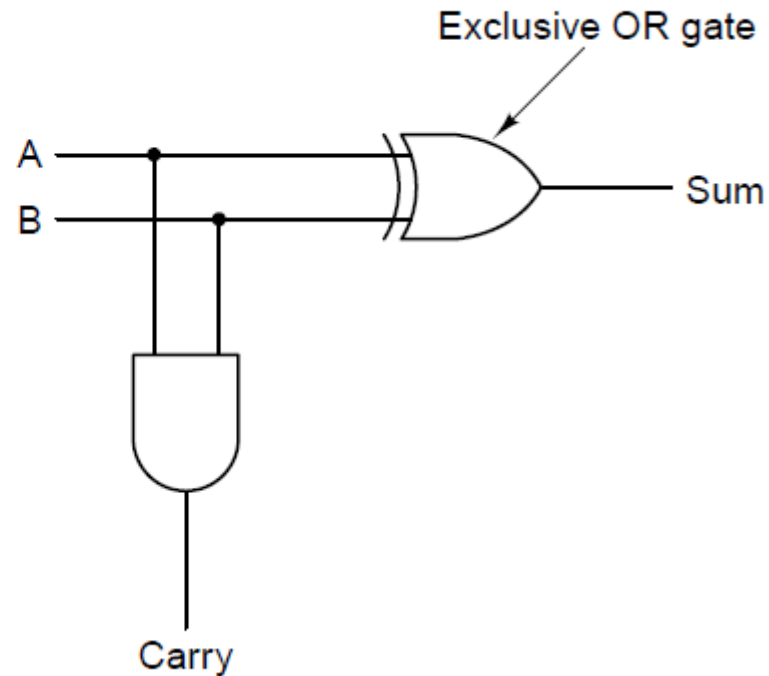Depending on the inputs, only one of eight outputs is 1.

Comparators

EXCLUSIVE OR gate

A_0
B_0

A_1
B_1

A_2
B_2

A_3
B_3

$A = B$

A simple 4-bit comparator.

Arithmetic Circuits

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Exclusive OR gate

A

B

Sum

Carry

(a) Truth table for a half-adder.

(b) Logic diagram for a half-adder.

47

Arithmetic Circuits – cont'd

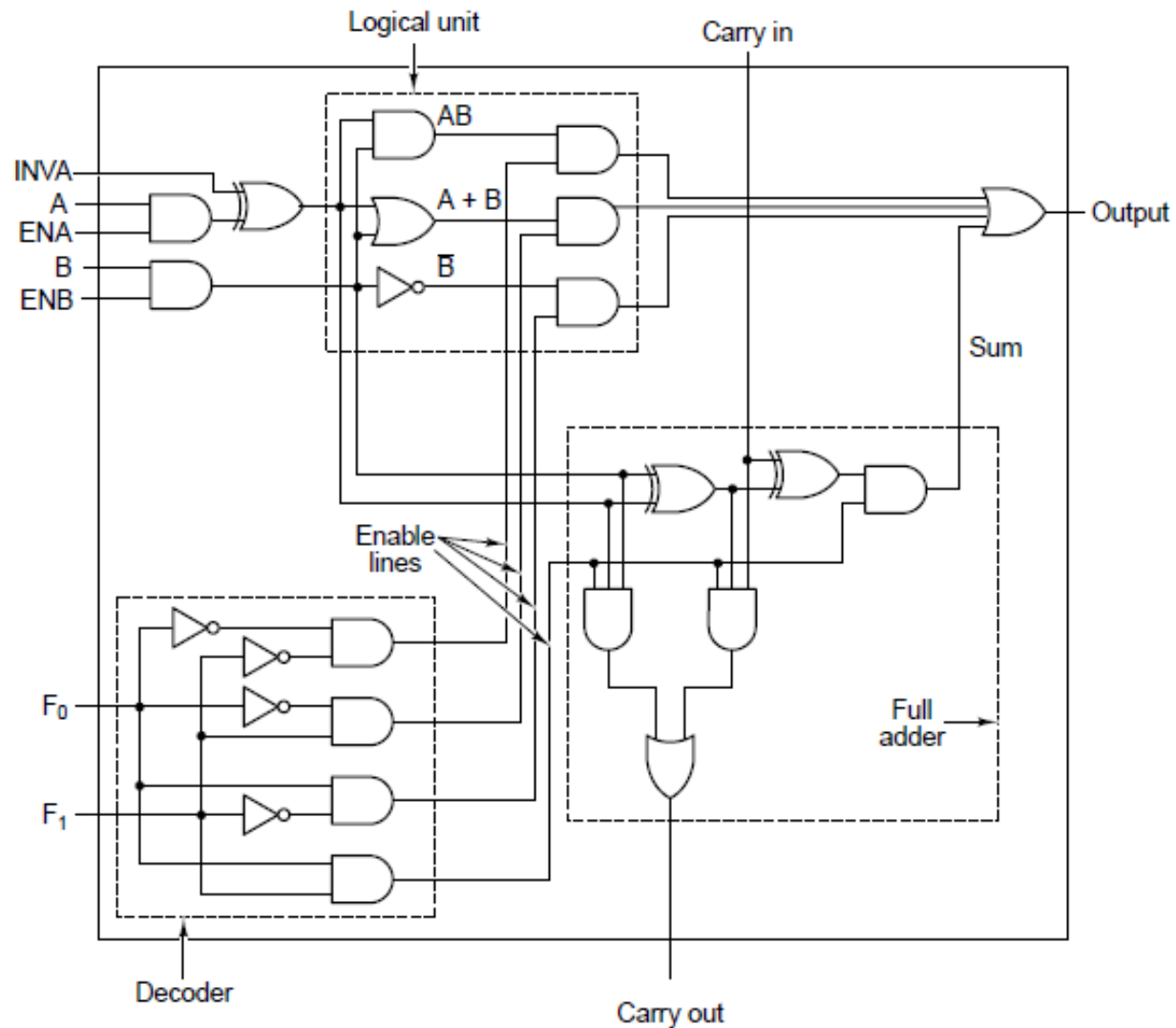| A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

(b)

(a) Truth table for a full-adder.
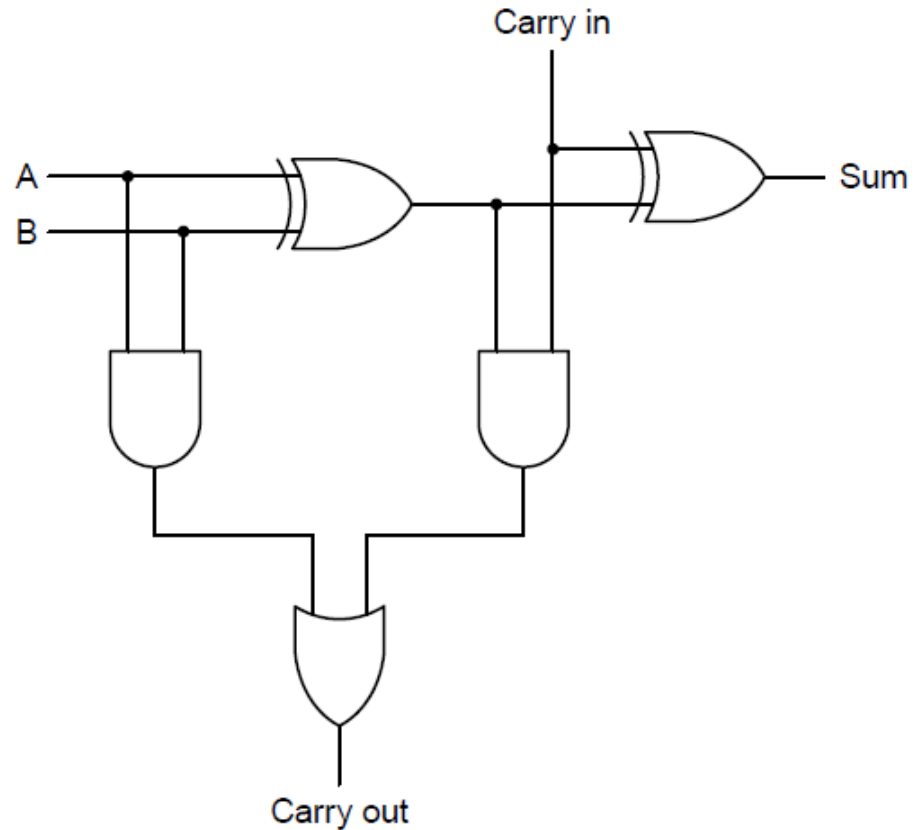
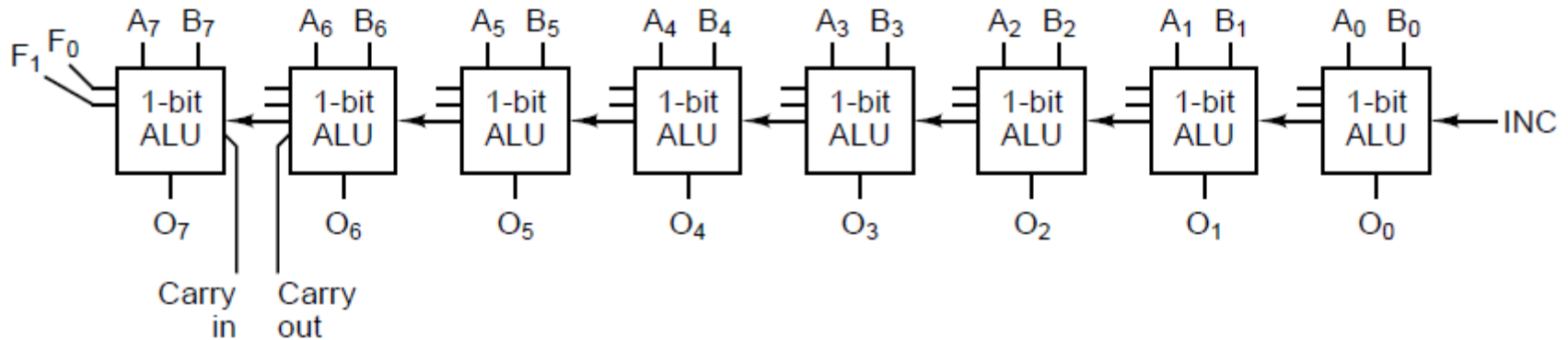(b) Logic diagram for a full-adder.

48

ALU: Arithmetic Logic Units

1-bit ALU

49

From 1-bit ALU to 8-bit ALU

Eight 1-bit ALU slices connected to make an 8-bit ALU.

The enable and invert signals are not shown for simplicity.

Summary

- Gate is a fundamental building block of all digital systems
- Logic gates are expressed with Boolean equation
- K-maps are used to derive Boolean equations from a truth table
- Design a digital system (combinational circuit)
  - Draw truth table
  - Draw K-maps to derive Boolean equations
  - Draw a circuit diagram based on the equations
- Decoder, Multiplexer, ALU, comparators are the examples of digital systems