



# Digital Logic

Professor: Yang Peng

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

# Topic

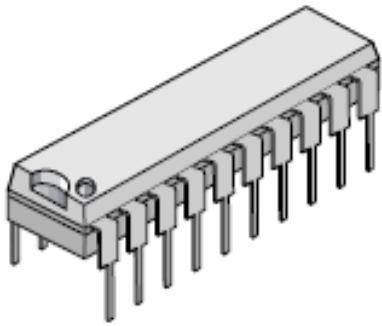
Chapter 1, 2, 3 by Berger

Chapter 3 by Null

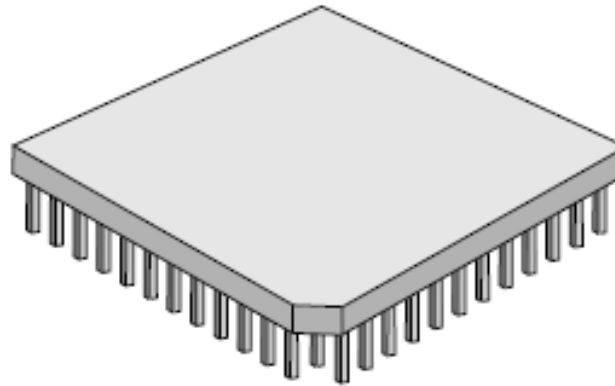
- Digital Logic and Combinational Circuit
  - Logic and Gates
  - K-maps and Boolean equation
  - Combinational Circuit Design

# Integrated Circuits

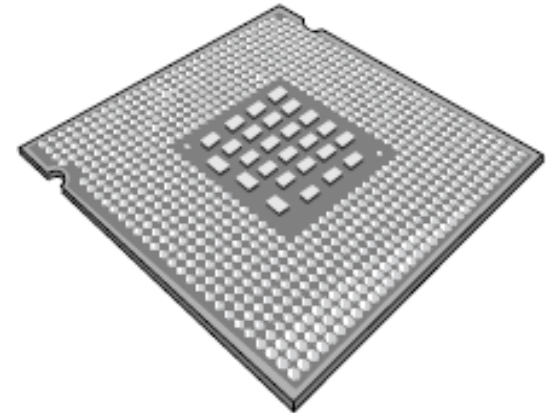
- Integrated Circuits (IC) consist of **gates**



(a)



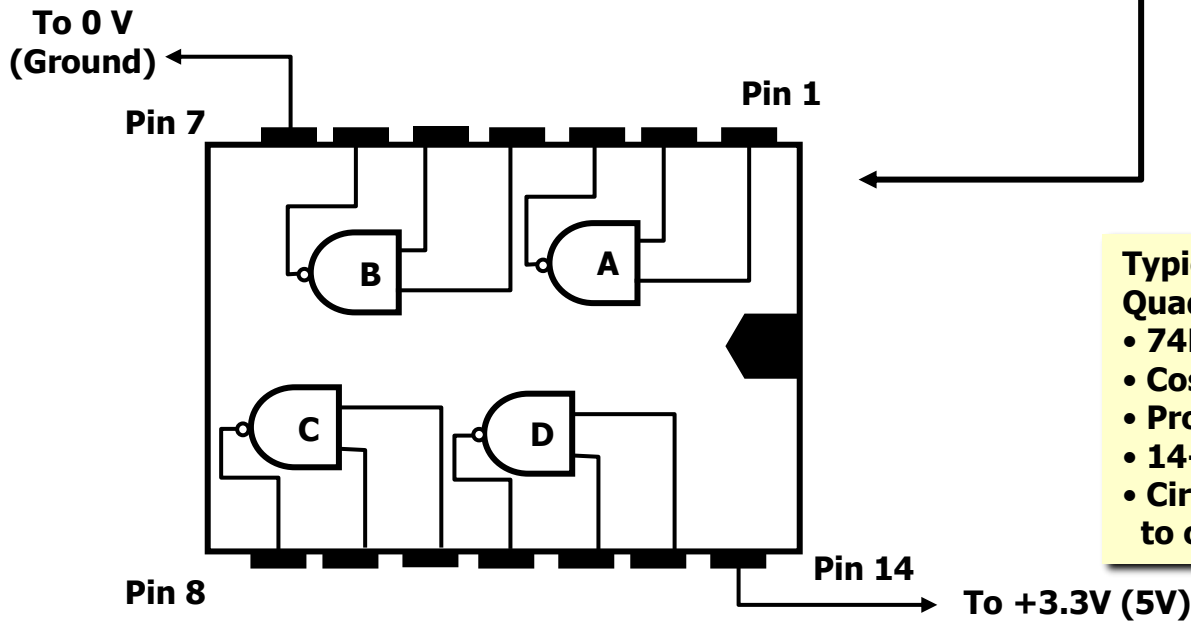
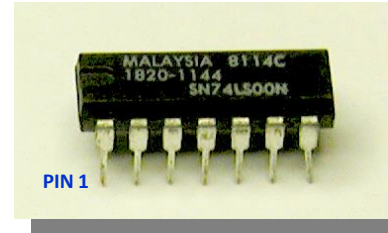
(b)



(c)

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

# NAND Gate Example

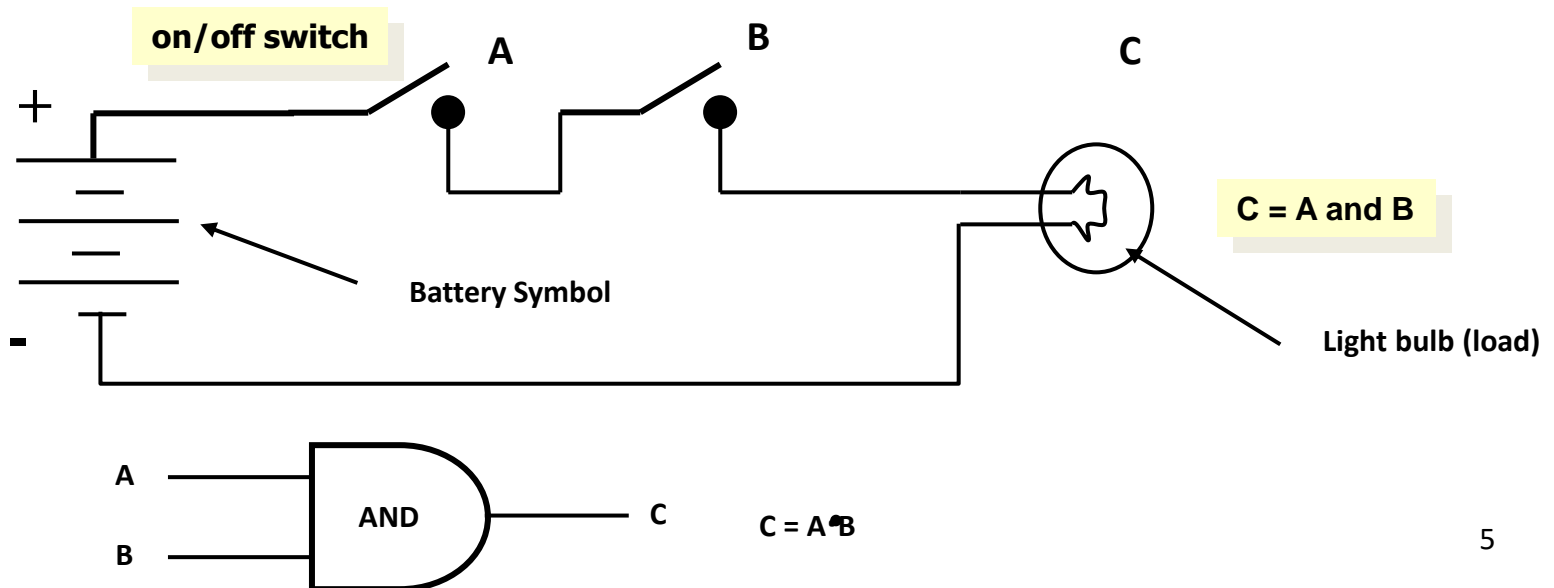


**Typical NAND gate circuit**  
Quad, 2-input NAND gate

- 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

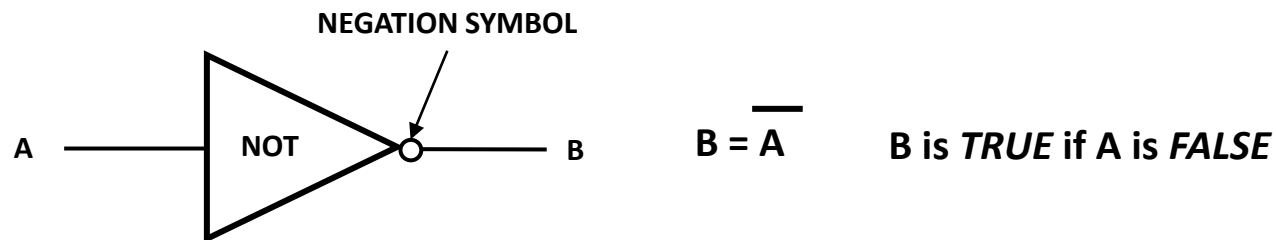
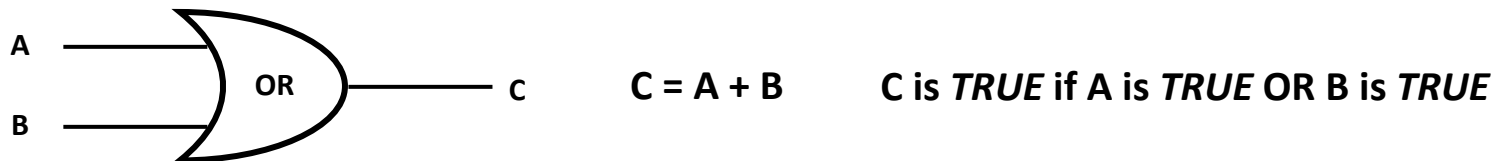
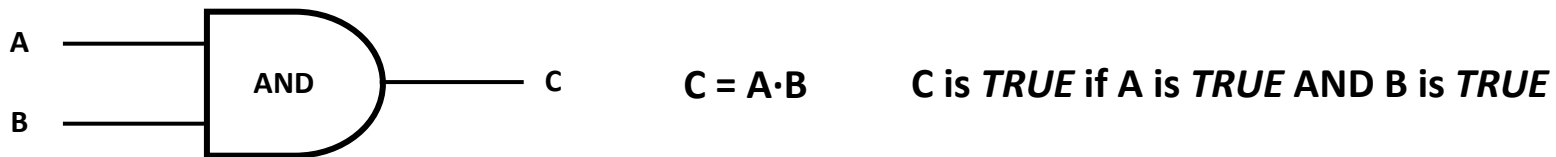
# Gate and Logic

- 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

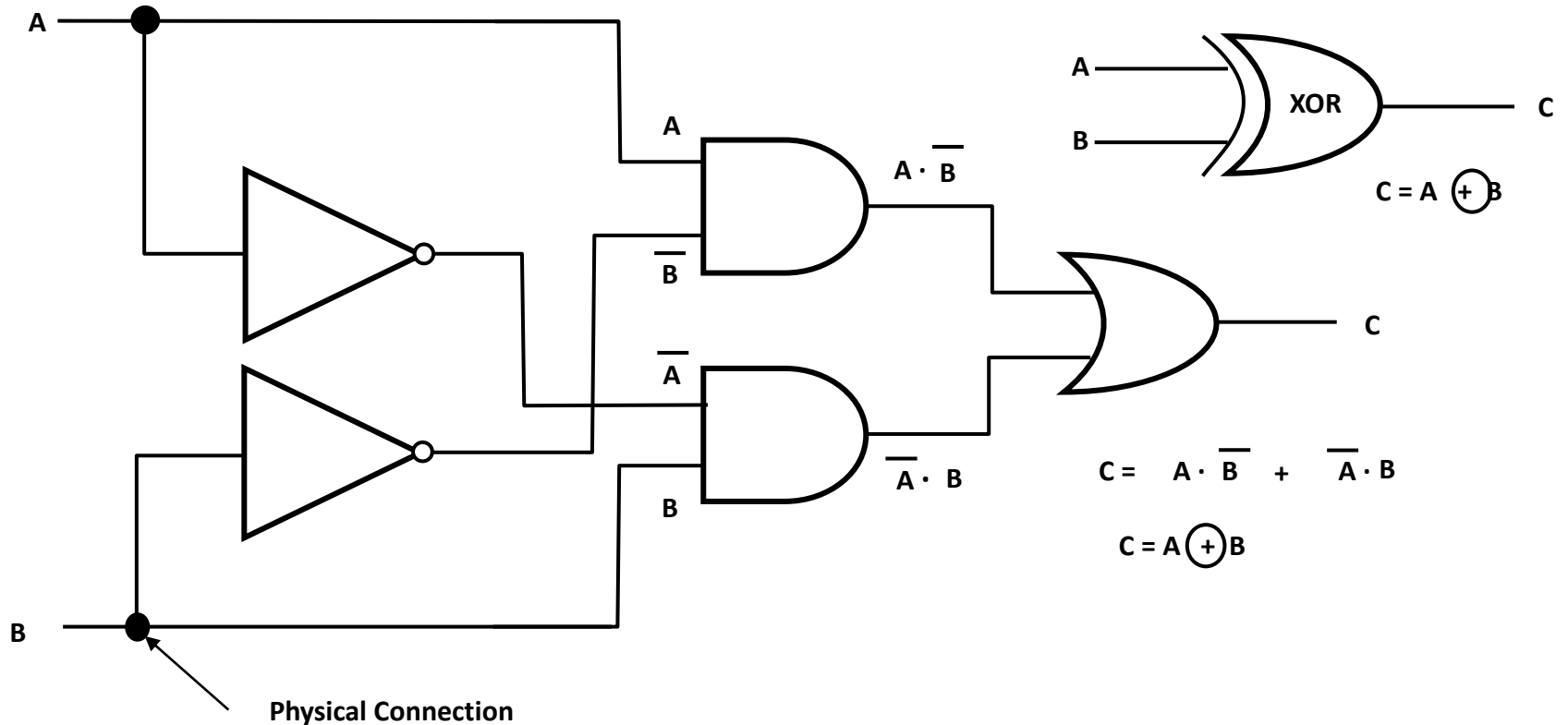


# Logical Gates

- The **Gate** is the basic element of all digital systems
  - Three types of gates form the “**atomic**” elements


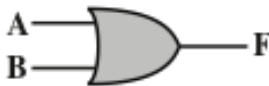


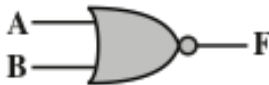



# Exclusive OR ( XOR ) Gate



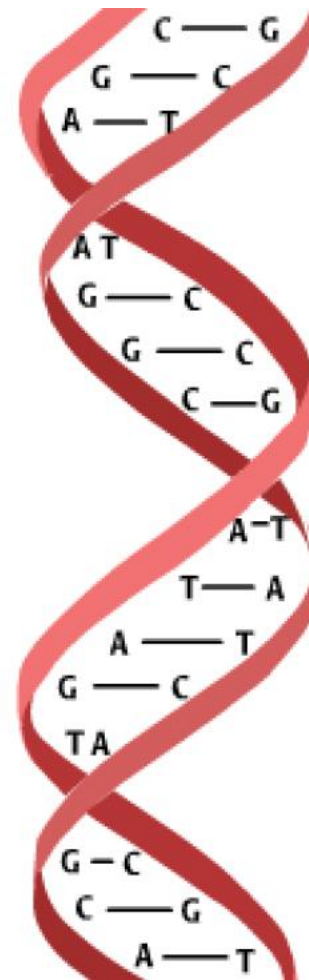
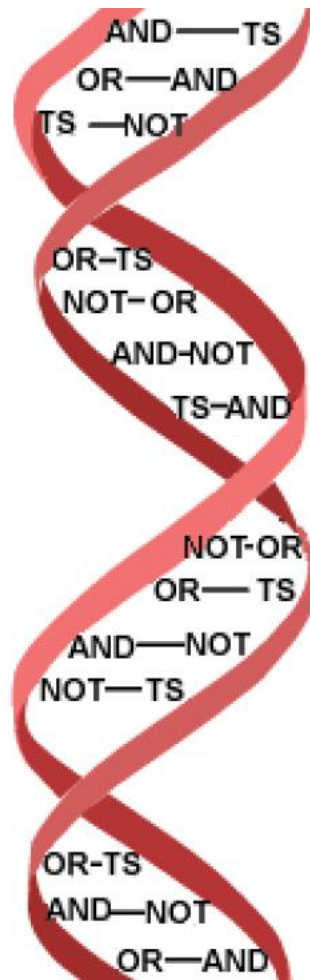
***C is FALSE if  $A = B$***

# Logic and Gate

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																



# Logical DNA



# Basic Identities of Boolean Algebra

## Basic Postulates

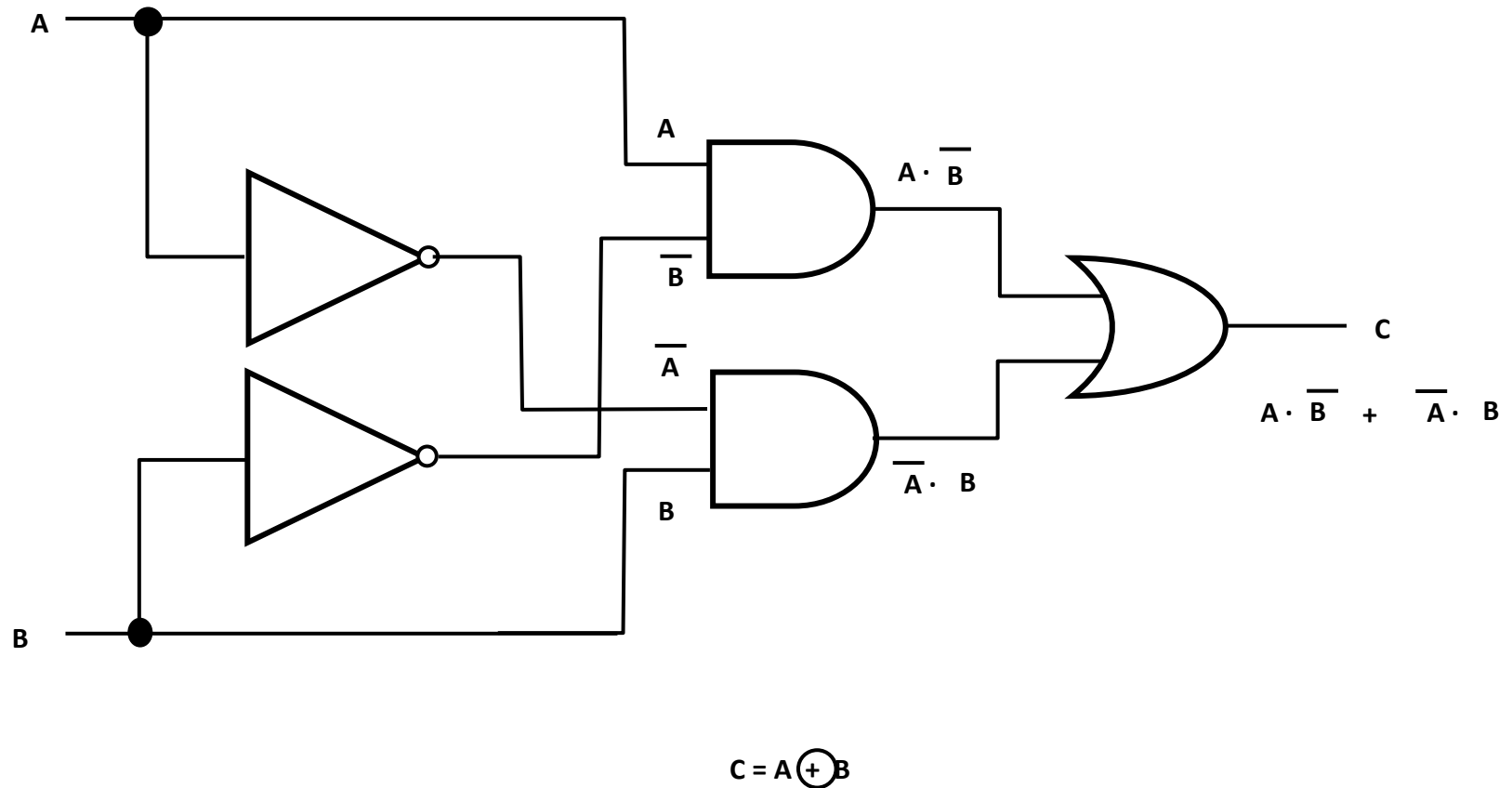
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \bar{A} = 0$	$A + \bar{A} = 1$	Inverse Elements

## 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
$\overline{A \cdot B} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A} \cdot \bar{B}$	DeMorgan's Theorem

# From Circuit to Boolean Equation

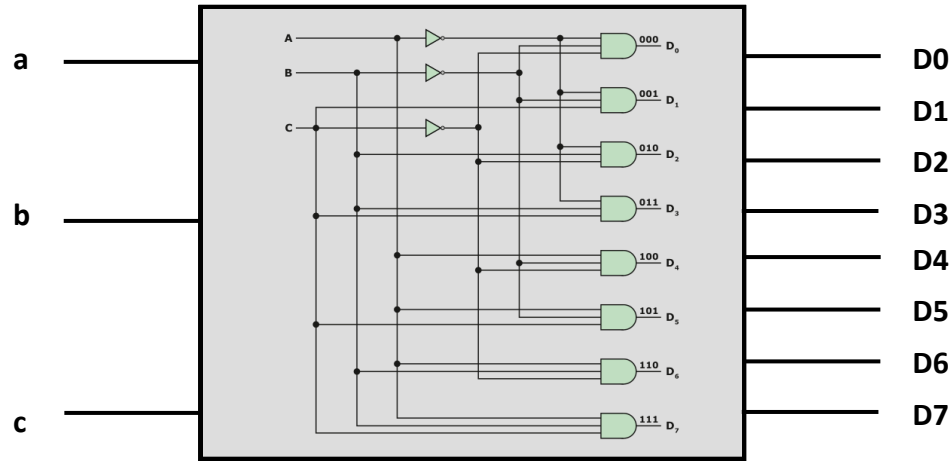
- We can easily interpret the function of a given circuit



**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)

$abc = 000 \rightarrow D0 = 1$

$abc = 001 \rightarrow D1 = 1$

$abc = 010 \rightarrow D2 = 1$

$abc = 011 \rightarrow D3 = 1$

$abc = 100 \rightarrow D4 = 1$

$abc = 101 \rightarrow D5 = 1$

$abc = 110 \rightarrow D6 = 1$

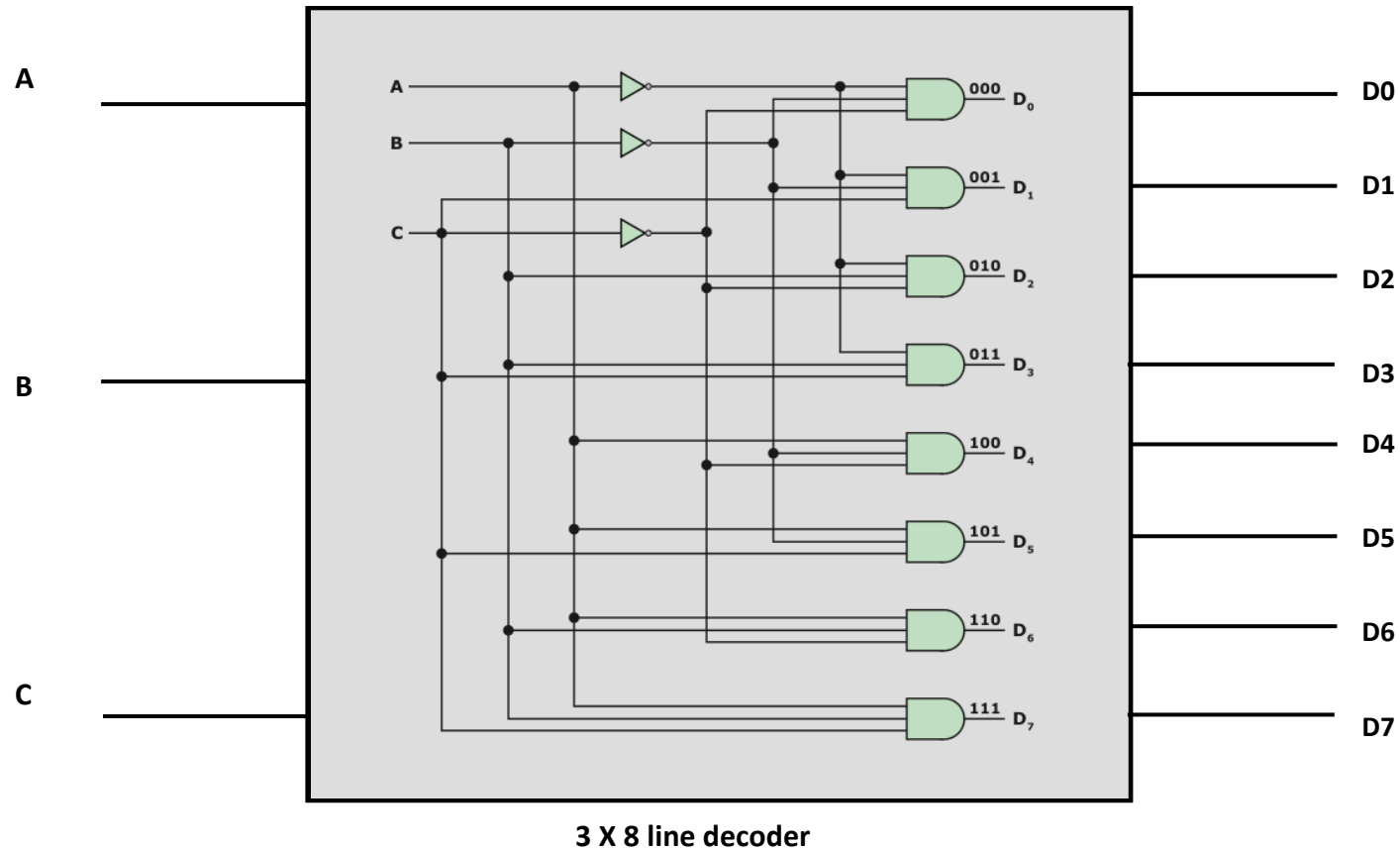
$abc = 111 \rightarrow D7 = 1$

# 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

# 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



# Design a 3 x 8 Line Decoder

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

$abc = 000 \rightarrow D0 = 1$

$abc = 001 \rightarrow D1 = 1$

$abc = 010 \rightarrow D2 = 1$

$abc = 011 \rightarrow D3 = 1$

$abc = 100 \rightarrow D4 = 1$

$abc = 101 \rightarrow D5 = 1$

$abc = 110 \rightarrow D6 = 1$

$abc = 111 \rightarrow D7 = 1$

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

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

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

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

$$D3 = \sim a b c$$

$$D4 = a \sim b \sim c$$

$$D5 = a \sim b c$$

$$D6 = a b \sim c$$

$$D7 = a b c$$

(I will use  $\sim$  instead of  $-$ , easier to type...)

# Design a 3 x 8 Line Decoder

4. Connect inputs to out using logic gates

$$D_0 = \sim a \sim b \sim c$$

$$D_1 = \sim a \sim b c$$

$$D_2 = \sim a b \sim c$$

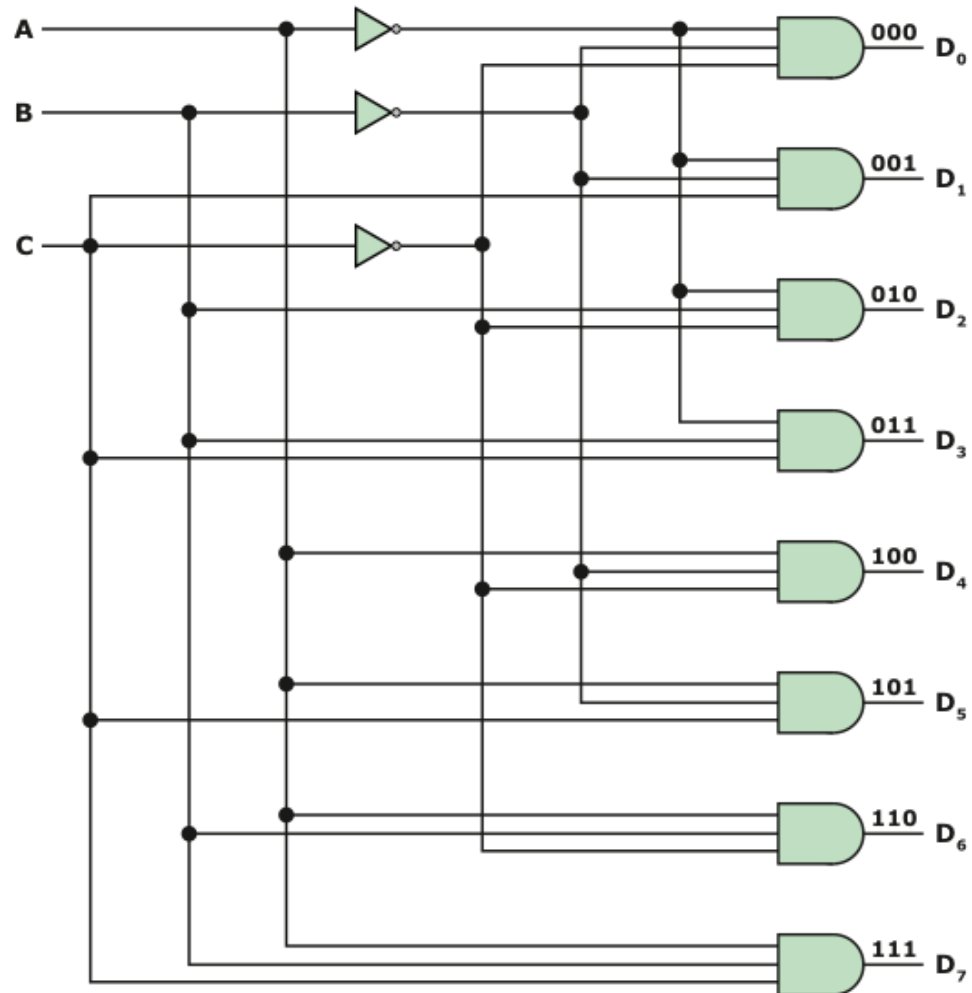
$$D_3 = \sim a b c$$

$$D_4 = a \sim b \sim c$$

$$D_5 = a \sim b c$$

$$D_6 = a b \sim c$$

$$D_7 = a b c$$



# Design a 3 x 8 Line Decoder

## 5. Test! (Use [Logisim](#))

abc = 000 → D0 = 1

abc = 001 → D1 = 1

abc = 010 → D2 = 1

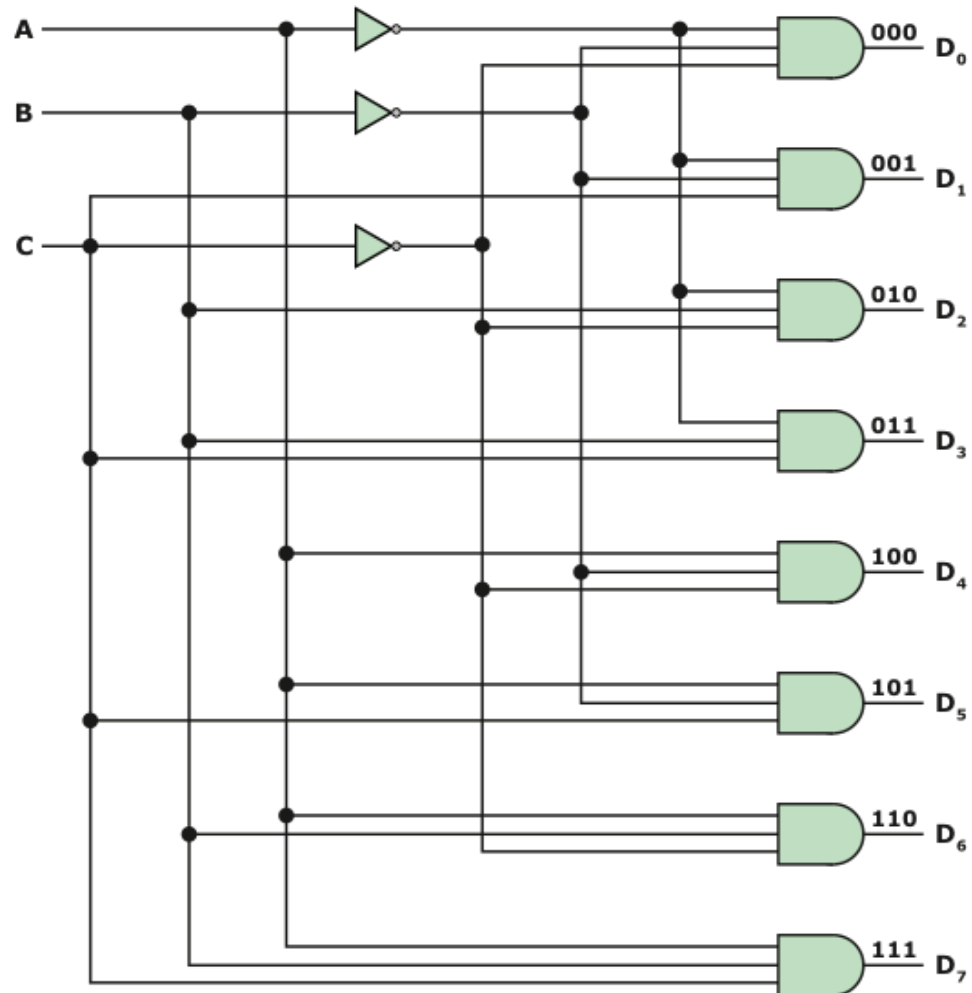
abc = 011 → D3 = 1

abc = 100 → D4 = 1

abc = 101 → D5 = 1

abc = 110 → D6 = 1

abc = 111 → D7 = 1



# 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

$$X = \sim A \sim B \sim C + A \sim B \sim C + AB \sim C + A \sim BC$$

$$X = \sim B \sim C + A(B \sim C + \sim BC)$$

$$X = (\sim A \sim B \sim C + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C} + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C} + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C}) + A(B \sim C + \sim BC)$$

$$X = (\sim A \sim B \sim C + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C}) + (AB \sim C + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C}) + (A \sim BC + \textcolor{red}{A} \sim \textcolor{red}{B} \sim \textcolor{red}{C})$$

$$x = \sim \textcolor{blue}{B} \sim \textcolor{blue}{C} + \textcolor{violet}{A} \sim \textcolor{violet}{C} + \textcolor{red}{A} \sim \textcolor{red}{B}$$

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  $AB + A\sim B = A$** 
  - Why?
    - First Distributive Law:  $A ( B + C ) = (AB) + (AC)$
    - Third Law of Complementation:  $B + \sim B = 1$
    - Finally:  $A \cdot 1 = A$
- Therefore
$$AB + A\sim B = A ( B + \sim B ) = A \cdot 1 = A$$
- <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

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C$				
$C$				

K-Map for 4 input variables

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C \sim D$				
$\sim CD$				
$CD$				
$C \sim D$				

K-Map for 5 input variables

	$\sim A \sim B \sim C$	$\sim A \sim BC$	$\sim ABC$	$\sim AB \sim C$	$AB \sim C$	$ABC$	$A \sim BC$	$A \sim B \sim C$
$\sim D \sim E$								
$\sim DE$								
$DE$								
$D \sim E$								

## 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

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

Note:  $\sim(ab)$  is not equal to  $\sim a \sim b$

$$\sim(ab) = \sim a + \sim b$$

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

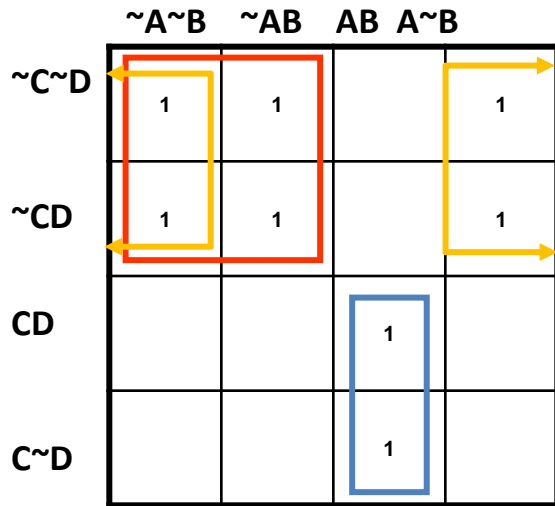
K-Map for 3 input variables

	$\sim a \sim b$	$a \sim b$	$ab$	$\sim ab$
$\sim c$				
$c$	1			

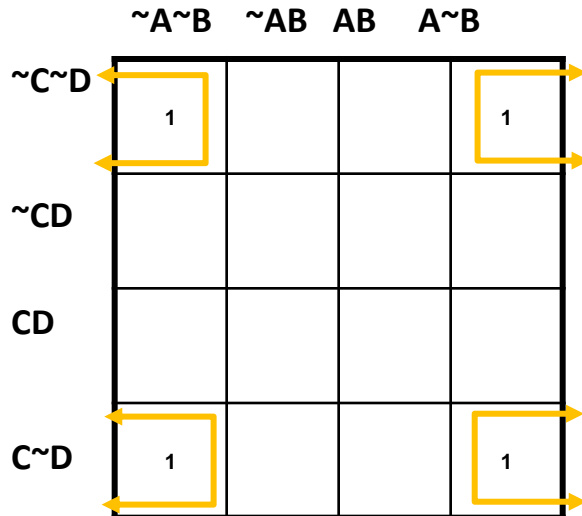
$$D1 = \sim a \sim b c$$

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



This K-Map has three loops. Notice that the yellow loop is actually adjacent.  
The equation is  $X = \sim A \sim C + \sim B \sim C + ABC$



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  $X = \sim B \sim D$

# K-Map Examples – 2

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C \sim D$	1	1	1	1
$\sim CD$	1	1	1	1
$CD$				
$C \sim D$				

This K-Map can really be simplified.

$$X = \sim C$$

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C \sim D$	1	1	1	1
$\sim CD$		1		
$CD$		1		
$C \sim D$	1	1	1	1

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

$$X = \sim D + \sim AB$$

# 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



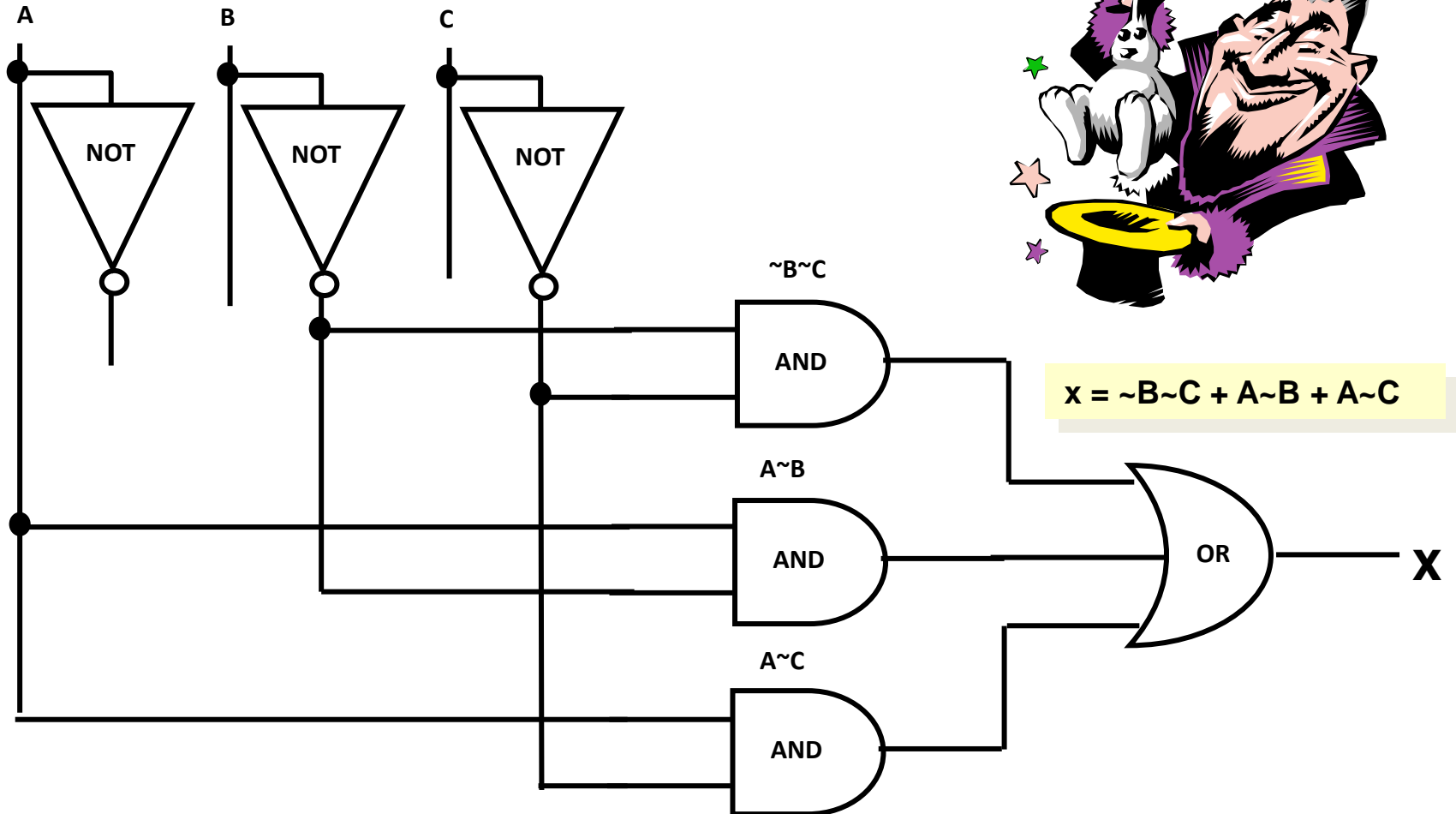
	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C$	1		1	1
$C$				1



$$x = \sim B \sim C + A \sim B + A \sim C$$

Simplified equation

# Designing the Hardware





# Prove Algebraically

Step 1:  $x = \sim A \sim B \sim C + A \sim B \sim C + A \sim BC + AB \sim C$

//From Truth Table

Step 2:  $x = \sim A \sim B \sim C + AB \sim C + A \sim B(\sim C + C)$

// First Distributive Law

Step 3:  $x = \sim A \sim B \sim C + AB \sim C + A \sim B$

// Law of Complementation

Step 4:  $x = \sim A \sim B \sim C + A(B \sim C + \sim B)$

// First Distributive Law

Step 5:  $x = \sim A \sim B \sim C + A[(\sim B + \sim C)(\sim B + B)]$

// Second Distributive Law

Step 6:  $x = \sim A \sim B \sim C + A(\sim B + \sim C)$

// Law of Complementation

Step 7:  $x = \sim A \sim B \sim C + A \sim B + A \sim C$

// First Distributive Law

Step 8:  $x = \sim B(\sim A \sim C + A) + A \sim C$

// First Distributive Law

Step 9:  $x = \sim B[(\sim A + A)(\sim C + A)] + A \sim C$

// Second Distributive Law

Step 10:  $x = \sim B(\sim C + A) + A \sim C$

// First Distributive Law

Step 11:  $x = \sim B \sim C + \sim BA + A \sim C$

// First Distributive Law

# 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.  $X = AB\sim C\sim D + A\sim BC\sim D + ABC\sim D + \sim A\sim B\sim CD + \sim AB\sim CD + \sim A\sim BCD + \sim ABCD$
2.  $X = AB\sim D( \sim C + C ) + A\sim BC\sim D + \sim A\sim CD ( \sim B + B ) + \sim ACD( B + \sim B )$
3.  $X = AB\sim D + A\sim BC\sim D + \sim AD( C + \sim C )$
4.  $X = AB\sim D + A\sim BC\sim D + \sim AD$
5.  $X = A\sim D(B + \sim BC) + \sim AD$
6.  $X = A\sim D(C + B) + \sim AD$  // Second Distributive Law

	$\sim A\sim B$	$\sim AB$	$AB$	$A\sim B$
$\sim C\sim D$			1	
$\sim CD$	1	1		
$CD$	1	1		
$C\sim D$			1	1

$$X = \sim AD + AB\sim D + AC\sim D$$

# 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 such systems with K-maps?

# K-Map with “don’t care” Condition

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C \sim D$			x	
$\sim CD$		1	x	
$CD$		x	1	
$C \sim D$	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

	$\sim A \sim B$	$\sim AB$	$AB$	$A \sim B$
$\sim C \sim D$			1	
$\sim CD$		1	1	
$CD$		1	1	
$C \sim D$	0		1	

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

# ROM: An example of Combinational Circuit

# 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

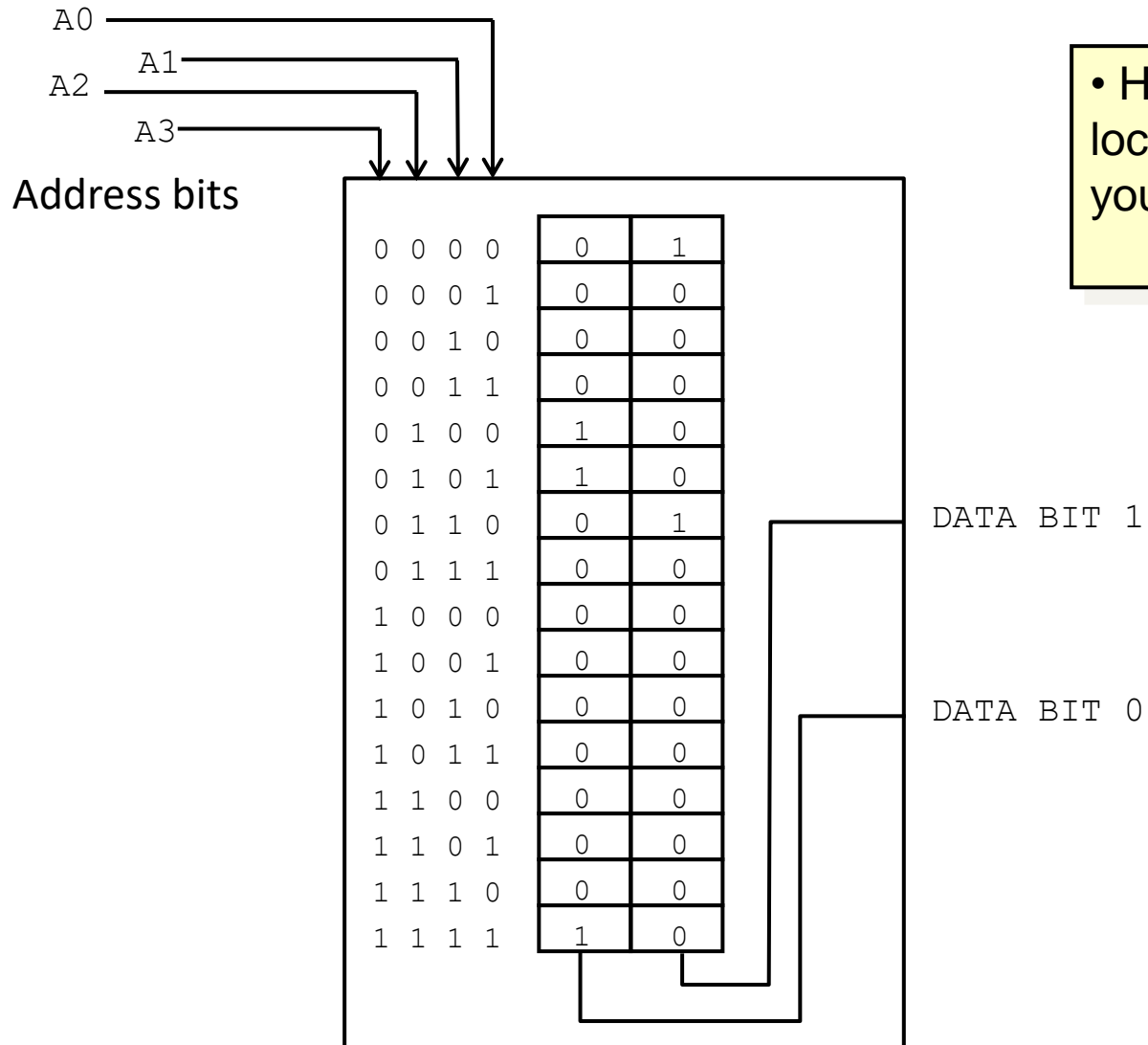
# Read-Only Memory (ROM)

Input				Output			
X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>	Z <sub>4</sub>
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

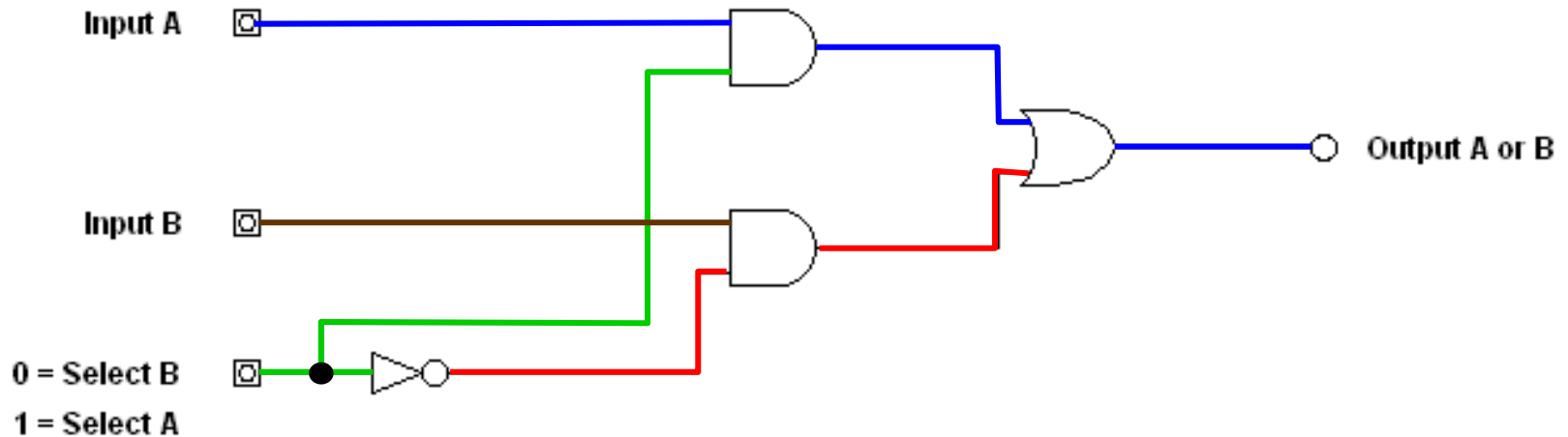


- How many memory locations can you have if you have four input bits?

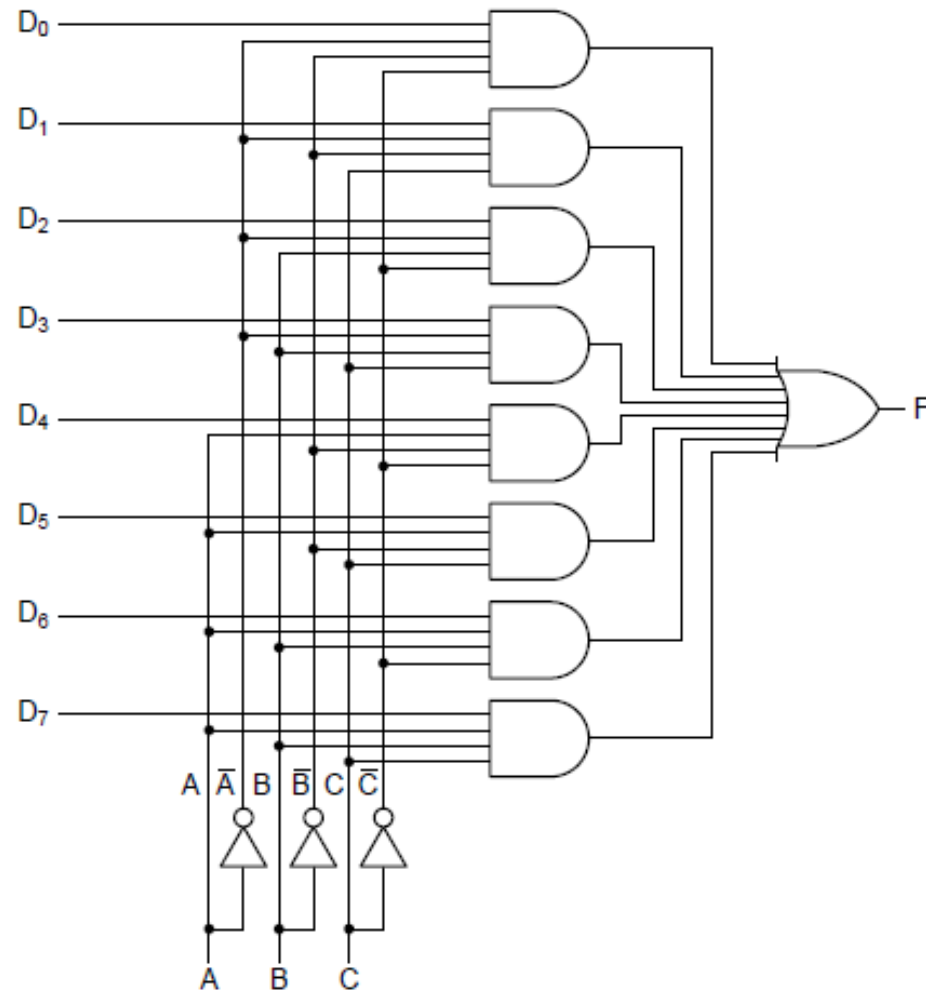
# More Examples of Combinational Circuit

# Simple Multiplexer (MUX)

- The MUX circuit allows one of two (or more) logical signals to be selected

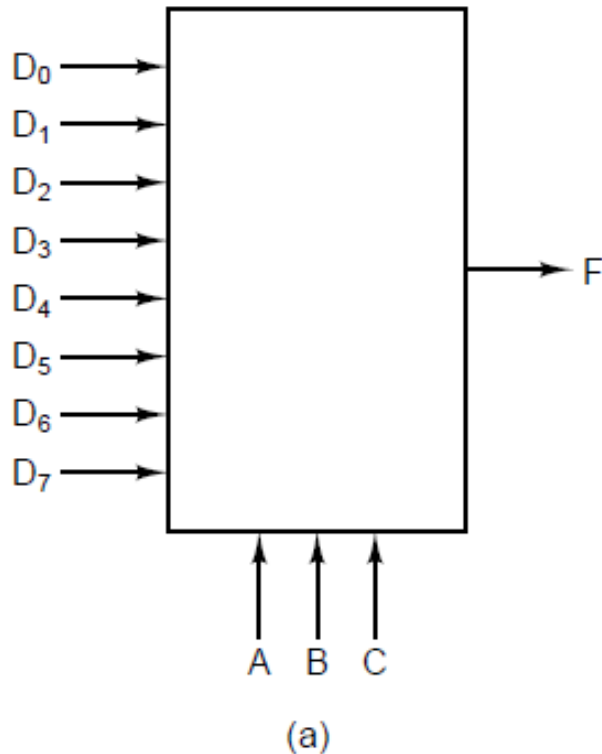


# Multiplexers

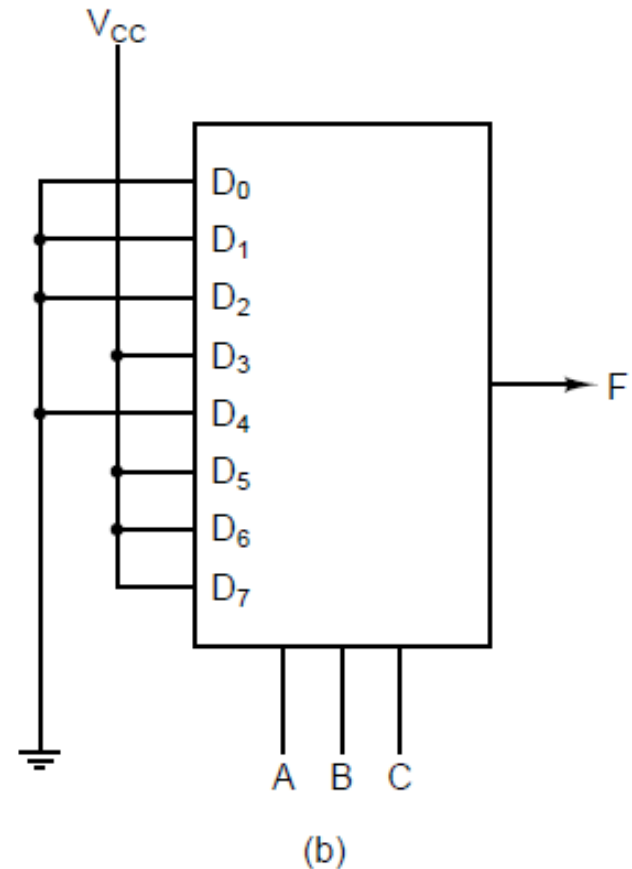


A, B, C are control lines to determine:  
which one of the eight input lines to be gated.

# Multiplexers – cont'd

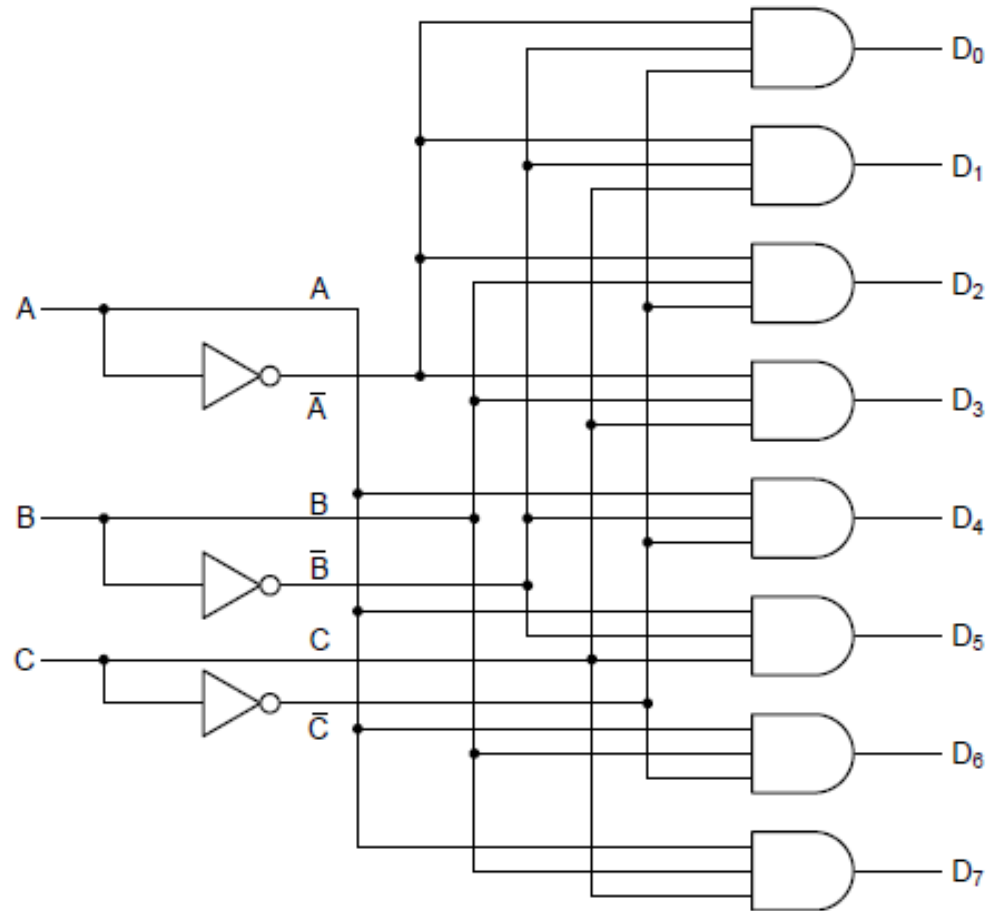


(a) An eight-input multiplexer.



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

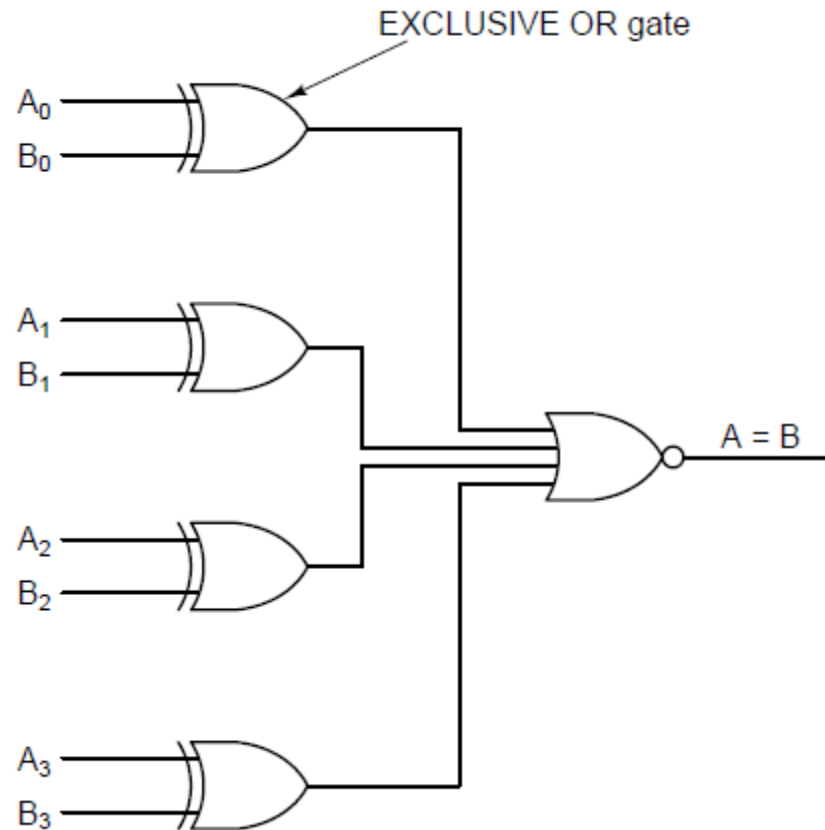
# Decoders



A 3-to-8 decoder circuit:

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

# Comparators

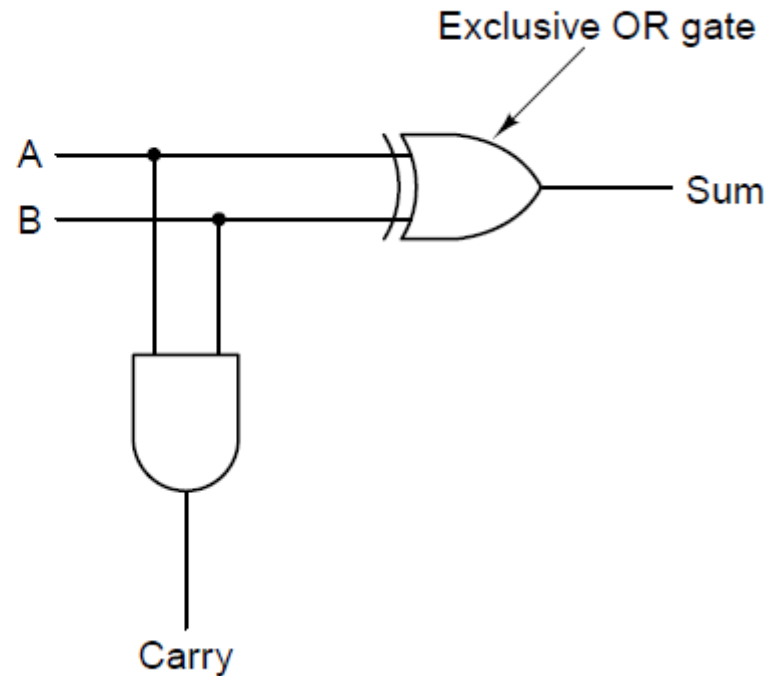


A simple 4-bit comparator.

# Arithmetic Circuits

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

(a) Truth table for a half-adder.



(b) Logic diagram for a half-adder.

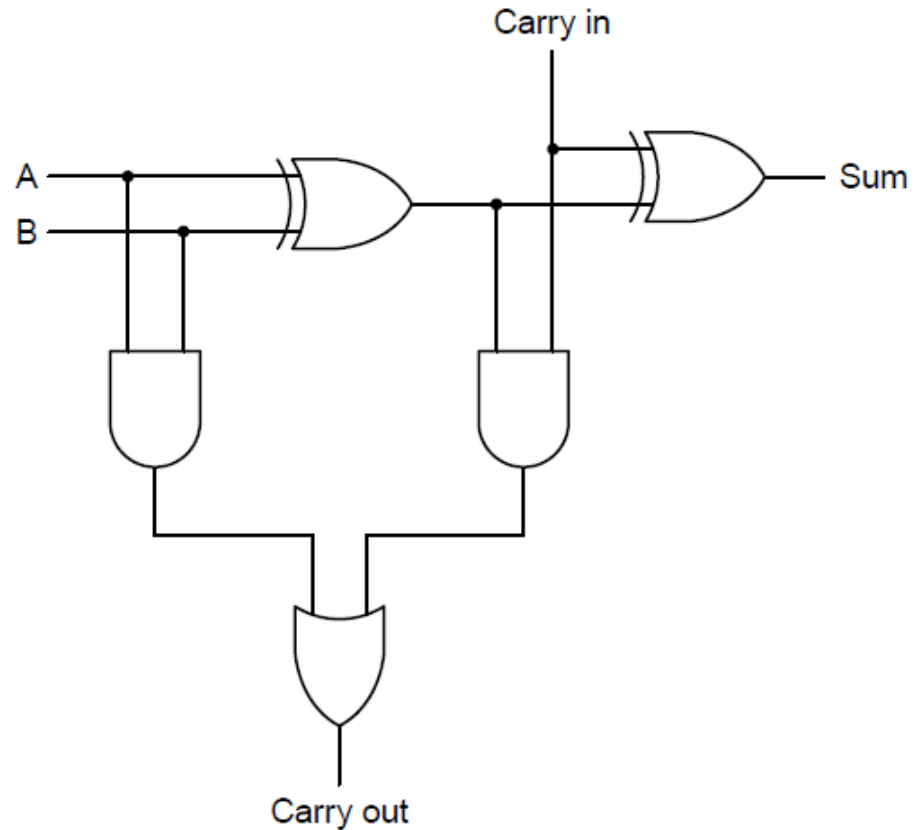


# Arithmetic Circuits – cont'd

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)

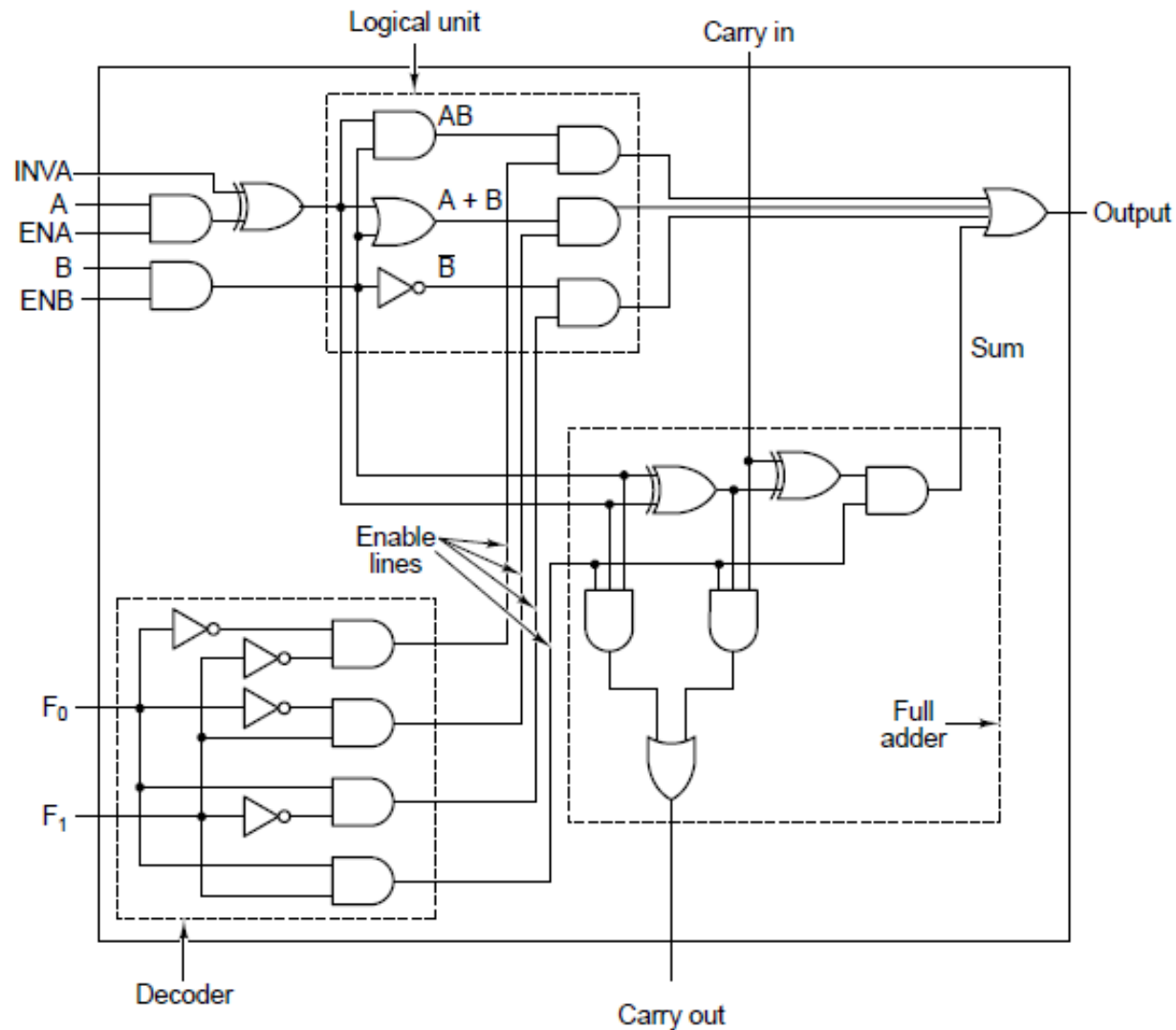
(a) Truth table for a full-adder.



(b)

(b) Logic diagram for a full-adder.

# ALU: Arithmetic Logic Units

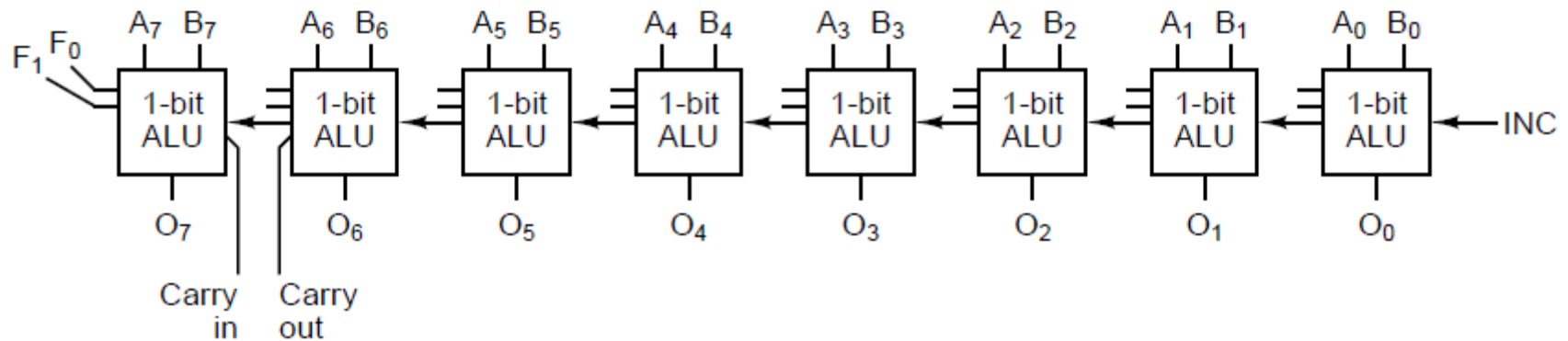


1-bit ALU

# 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