Combinational Circuit

Logical Gates and Karnaugh Map

Ver. 3.1

Professor: Yang Peng

Topics

- Logic and Gates
  - From circuit to logic
  - From logic to circuit
- Canonical Forms
  - Sum of products
  - Product of sums
- Karnaugh Maps and Boolean Equation

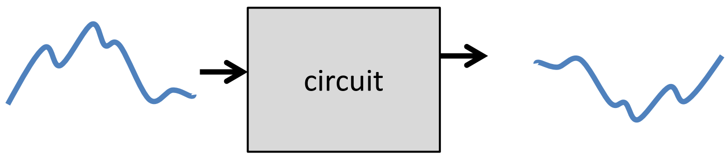
Berger: Ch 1, 2, and 3

Null: Ch 3

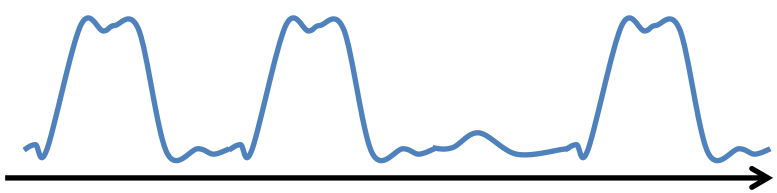
Analog vs. Digital Circuits

Analog

- Values: Any level probably between 0 and 5 volts

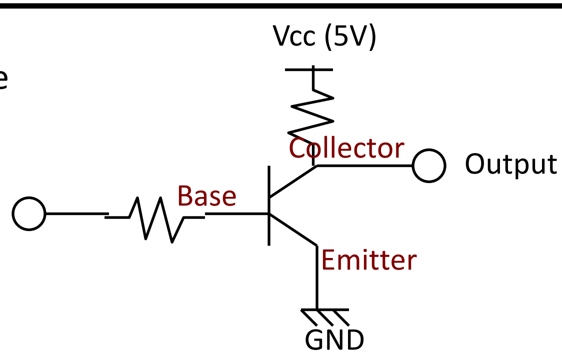


- Signal Transmission



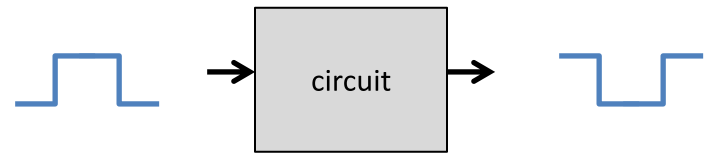
- Example

Input
High = 2.0 ~ 5V
Low = 0 ~ 0.7V

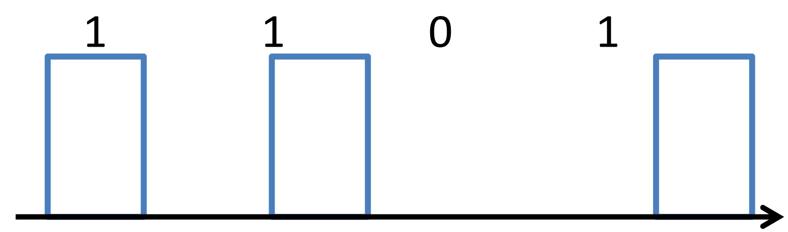


Digital

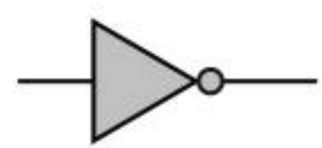
- Values: Two states
  5V = H = 1 = On, 0V = L = Off



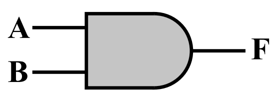
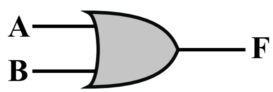
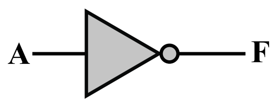
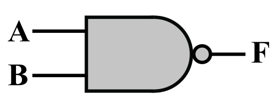
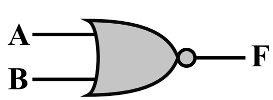
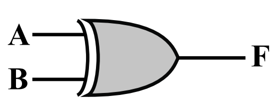
- Signal Transmission



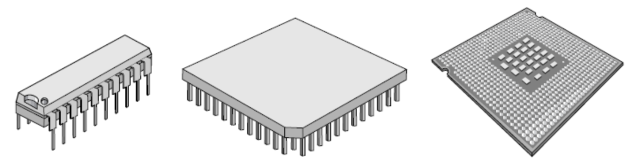
- Example



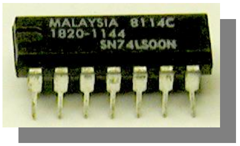
Basic Logical Gates

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

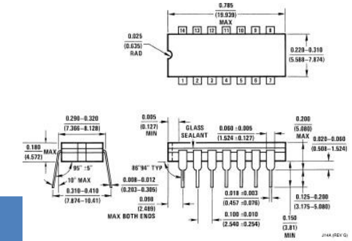
In general, integrated circuits consist of these basic logical gates, in a tiny format.



(a) (b) (c)

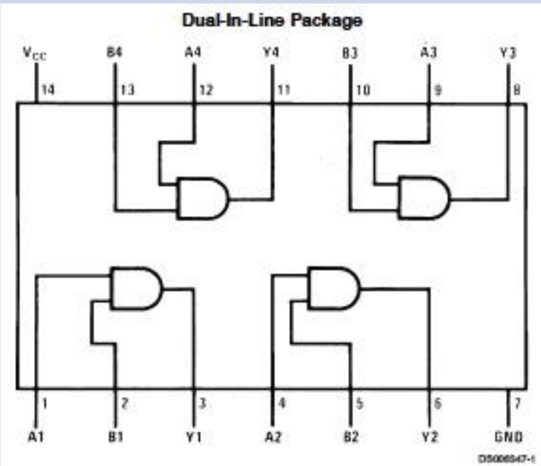


More Logical Gates



AND

DM74LS08
Quad 2-Input AND Gates



Function Table

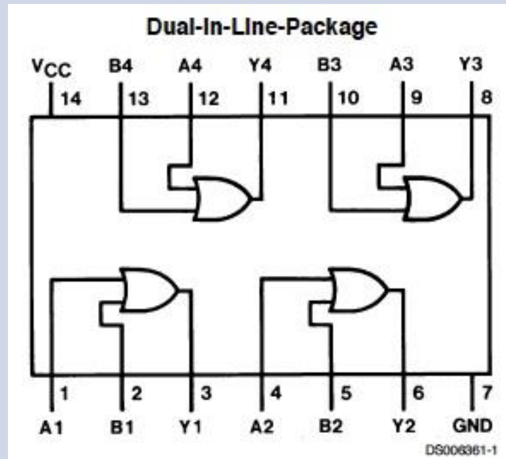
Y = AB

Inputs		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

H = High Logic Level
L = Low Logic Level

OR

DM74LS32
Quad 2-Input OR Gates



Function Table

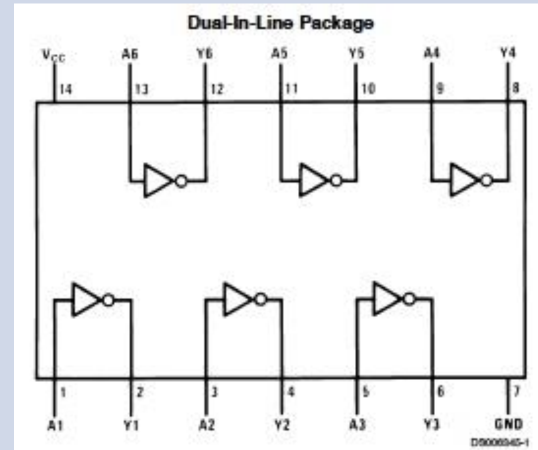
Y = A + B

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

H = High Logic Level
L = Low Logic Level

NOT

DM74LS04
Hex Inverting Gates



Function Table

$Y = \overline{A}$

Input	Output
A	Y
L	H
H	L

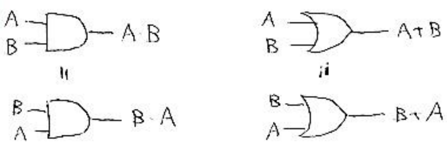
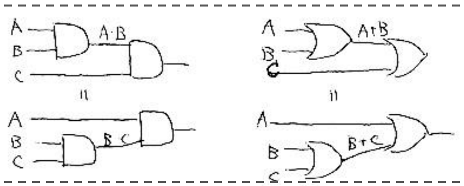
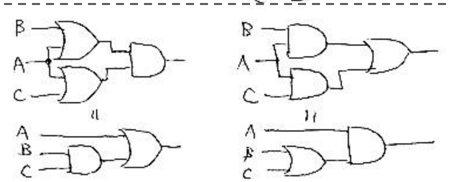
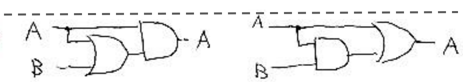
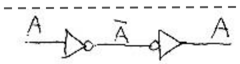
H = High Logic Level
L = Low Logic Level

Boolean Algebra and Logical Gates

Axiom

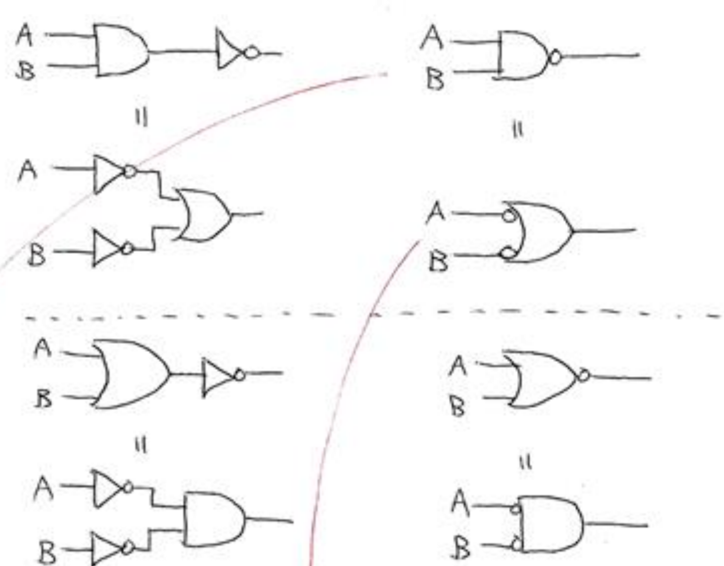
A1	if $A \neq 0$, $A = 1$ if $A \neq 1$, $A = 0$	
A2	$0 \cdot 0 = 0$ $1 + 1 = 1$	
A3	$1 \cdot 1 = 1$ $0 + 0 = 0$	
A4	$0 \cdot 1 = 1 \cdot 0 = 0$ $1 + 0 = 0 + 1 = 1$	
A5	$\bar{1} = 0$ $\bar{0} = 1$	

Theorem

T1	$A \cdot B = B \cdot A$ $A + B = B + A$ Commutative Law	
T2	$(AB)C = A(BC)$ $(A+B)+C = A+(B+C)$ Associative Law	
T3	$(A+B)(A+C) = A+BC$ $AB + AC = A(B+C)$ Distribution Law	
T4	$A \cdot 0 = 0$ $A + 1 = 1$	
T5	$A \cdot 1 = A$ $A + 0 = A$	
T6	$A \cdot \bar{A} = 0$ $A + \bar{A} = 1$ Complement Law	
T7	$A \cdot A = A$ $A + A = A$ Square Law	
T8	$A(A+B) = A$ $A + AB = A$ Absortion Law	
T9	$\overline{(\bar{A})} = A$ Double Negation	

De Morgan’s Law

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



The Corresponding Truth Table

B	A	Y	X
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

=

B	A	$\overline{B}$	$\overline{A}$	X
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

In general

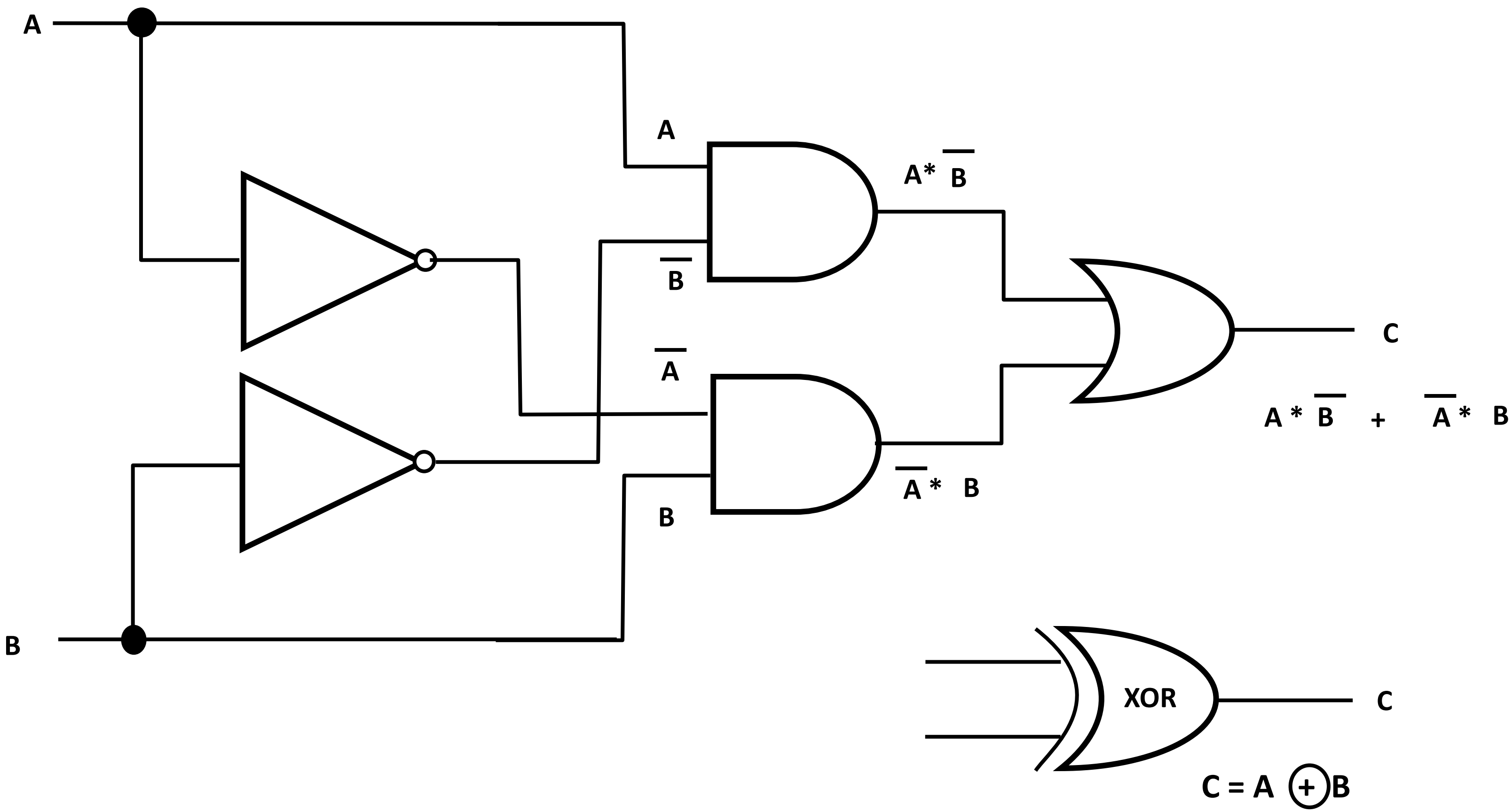
$$\overline{(A \cdot B \cdot C \cdot \cdots\cdots N)} = \overline{A} + \overline{B} + \overline{C} \cdots\cdots + \overline{N}$$
$$\overline{(A + B + C + \cdots + N)} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \cdots\cdots \cdot \overline{N}$$

Please Review Boolean Algebra!

Null's Chapter 3
Berger's Chapter 3

From Circuit to Boolean Logic

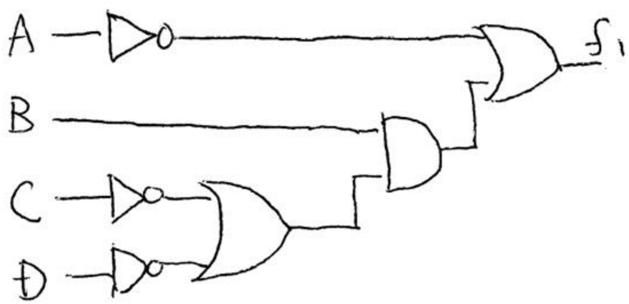
- We can easily interpret the function (logic) of a given circuit.

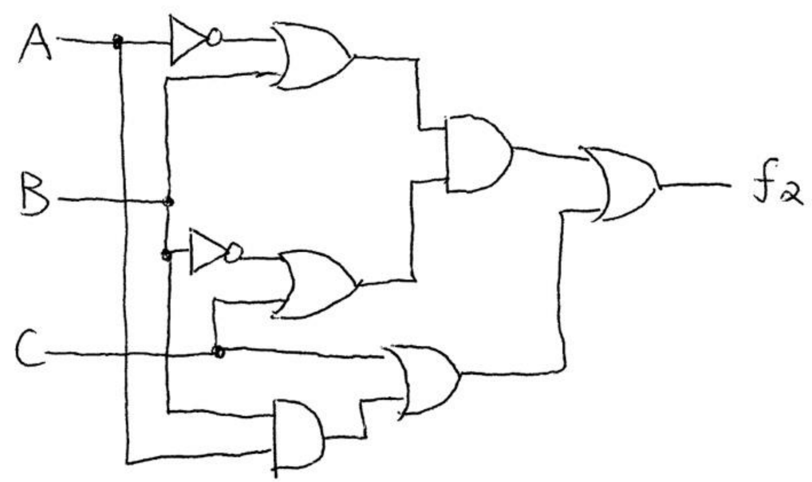


HOW TO BUILD A CIRCUIT WITH SOME GIVEN LOGIC?

From Boolean Equation to Circuit

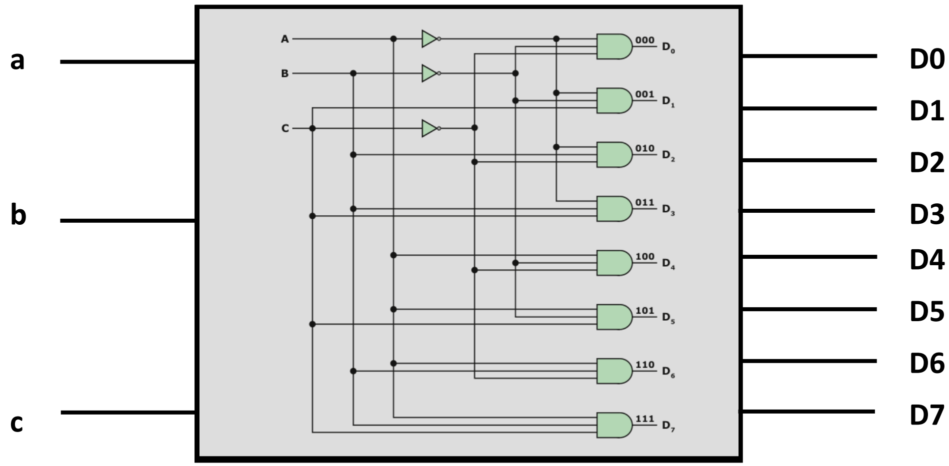
$f_1 = \bar{A} + B(\bar{C} + \bar{D})$ $\quad$ $f_2 = (\bar{A} + B)(\bar{B} + C) + (AB + C)$





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 → D0 =1	abc = 100 → D4 =1
abc = 001 → D1 =1	abc = 101 → D5 =1
abc = 010 → D2= 1	abc = 110 → D6= 1
abc = 011 → D3= 1	abc = 111 → D7= 1

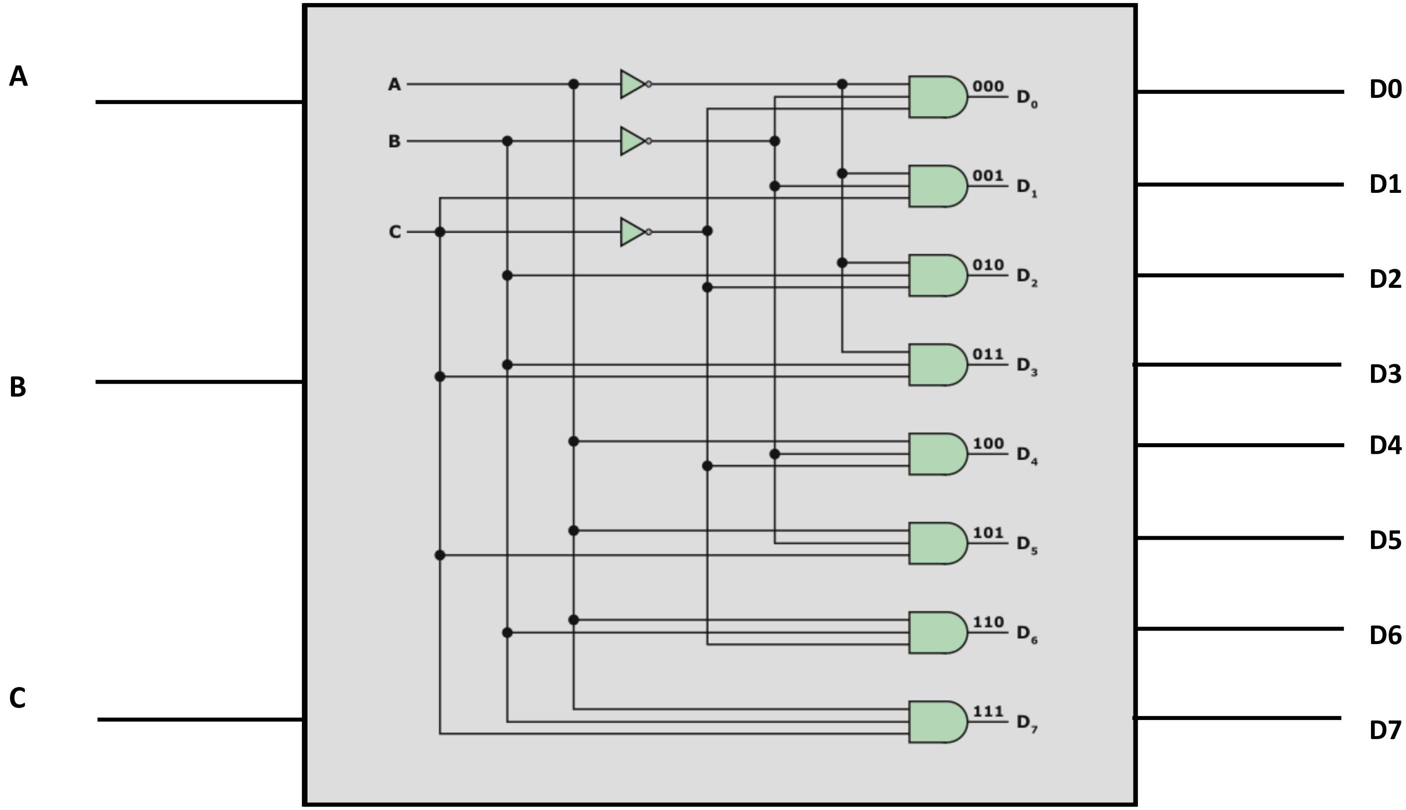
Design a Circuit

- Designing a digital system starts from designing a chip, which consists of a few logic 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. Derive a Boolean equation for each output and connect inputs to out 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)

1. Decide the number of inputs and the number of outputs
   – Three bits for input, eight bits for output



3 X 8 line decoder

Design a 3 x 8 Line Decoder (2)

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)

3. Derive a Boolean equation for each output and connect inputs to out using logic gates

D0 = ~a ~b ~c

D1 = ~a ~b c

D2 = ~a b ~c

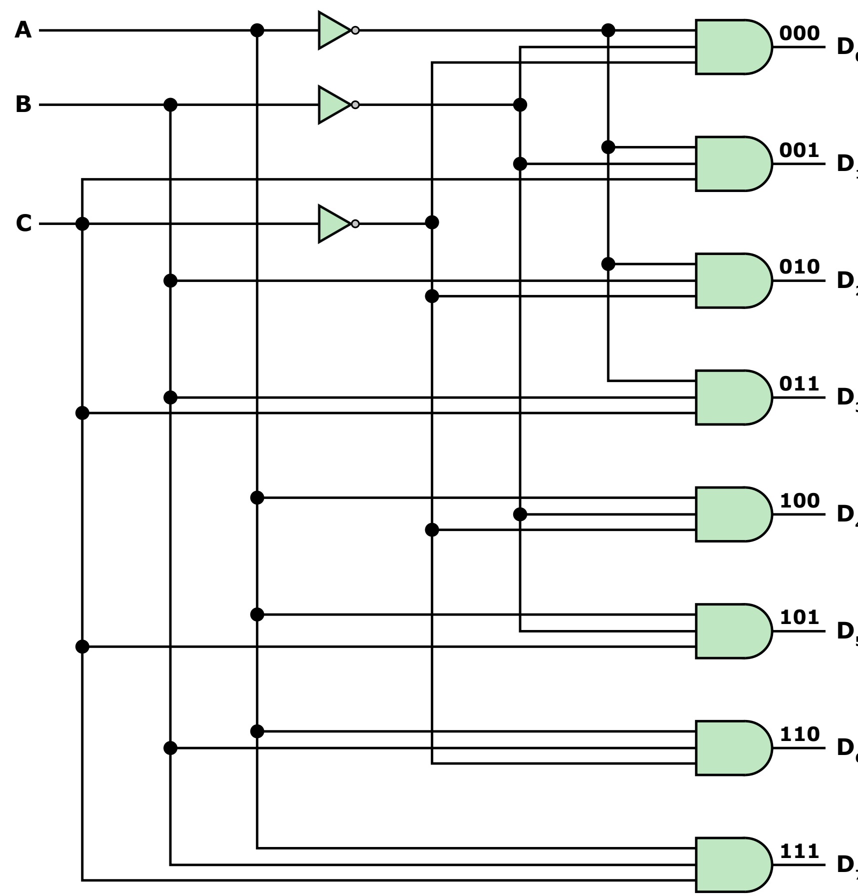
D3 = ~a b c

D4 = a ~b ~c

D5 = a ~b c

D6 = a b ~c

D7 = a b c



Design a 3 x 8 Line Decoder (4)

4. Test! (Use Logisim)

abc = 000 → D0 =1

abc = 001 → D1 =1

abc = 010 → D2= 1

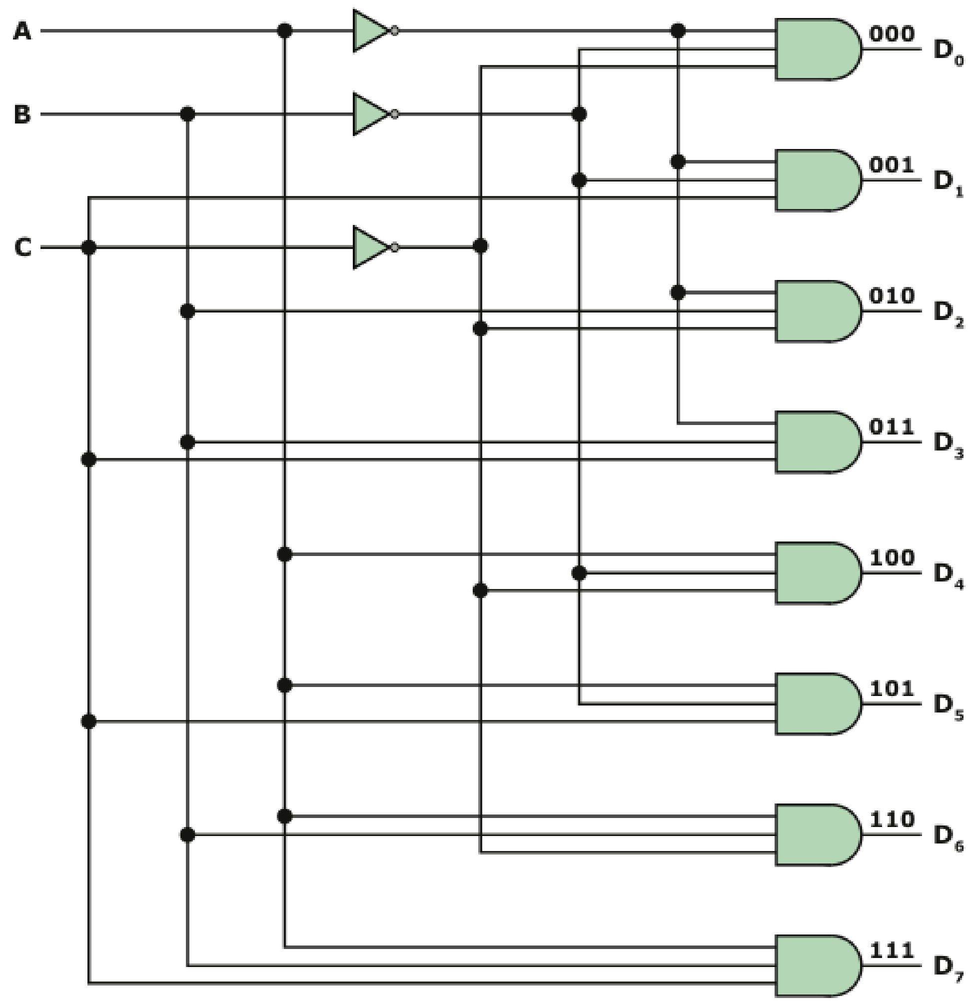
abc = 011 → D3= 1

abc = 100 → D4 =1

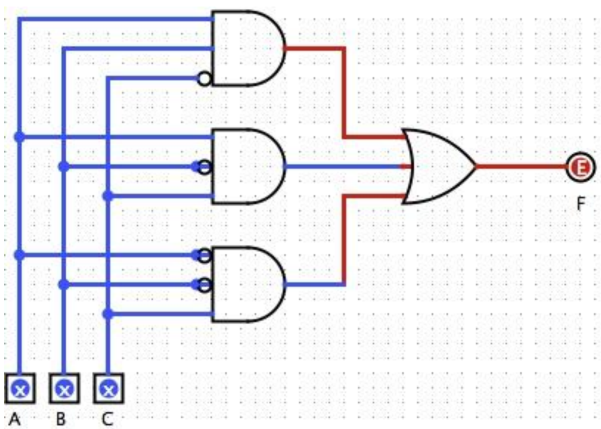
abc = 101 → D5 =1

abc = 110 → D6= 1

abc = 111 → D7= 1



Canonical Form: Sum of Products



product

F = AB~C + A~BC + ~ABC

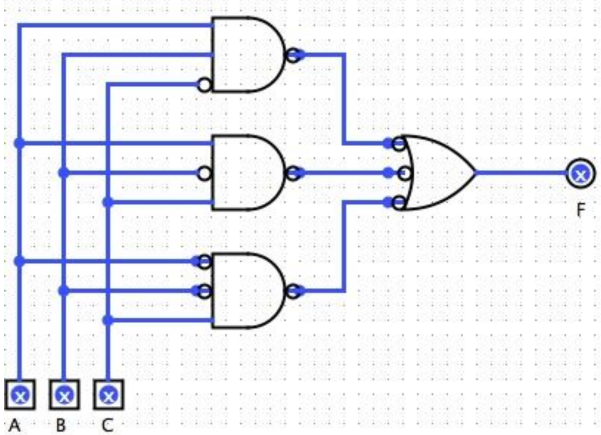
sum

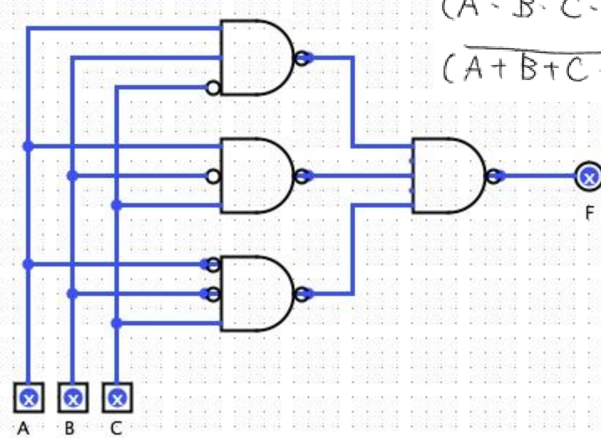
(where ~ = NOT)

Through De Morgan's Law, the circuit can be changed to consist of only NAND gates (with NOT gates)

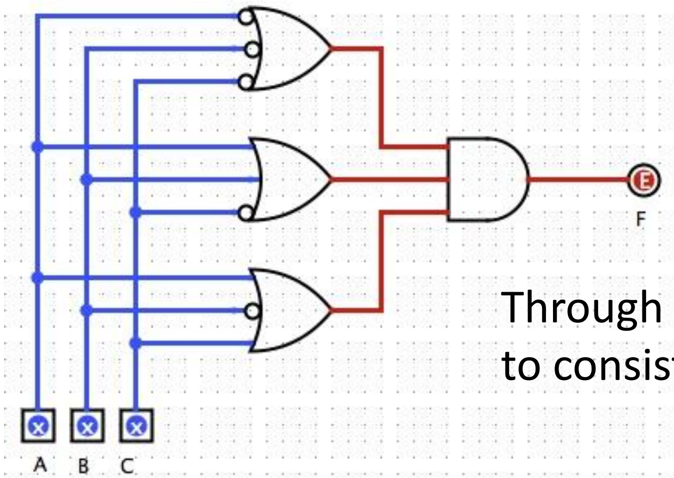
$$\overline{(A\cdot B\cdot C\cdot \cdots N)} = \overline{A}+\overline{B}+\overline{C}\cdots+\overline{N}$$

$$\overline{(A+B+C+\cdots+N)} = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdots\cdot\overline{N}$$





Canonical Form: Product of Sums



$$F = \underbrace{(\sim A+\sim B+\sim C)}_{\text{sum}}(A+B+\sim C)(A+\sim B+C)$$

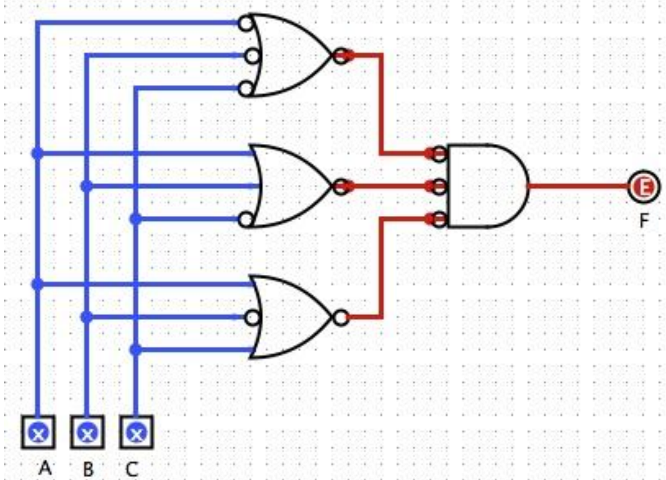
product

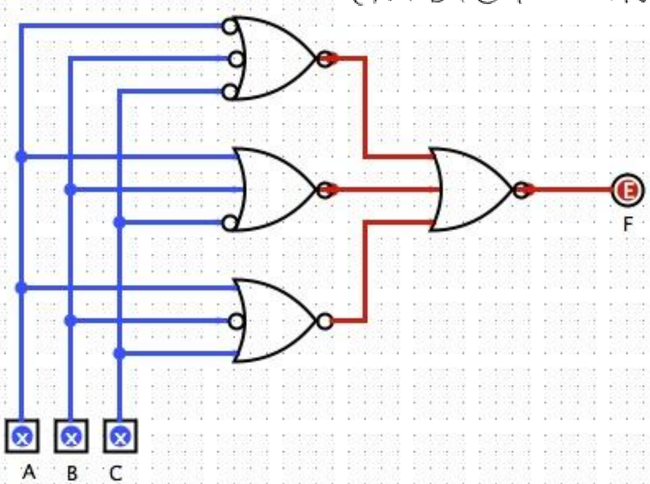
(where ~ = NOT)

Through De Morgan's Law, the circuit can be changed to consist of only NOR gates (with NOT gates).

$$\overline{(A \cdot B \cdot C \cdot \cdots\cdots N)} = \overline{A} + \overline{B} + \overline{C} \cdots\cdots + \overline{N}$$

$$\overline{(A + B + C + \cdots + N)} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \cdots\cdots \cdot \overline{N}$$





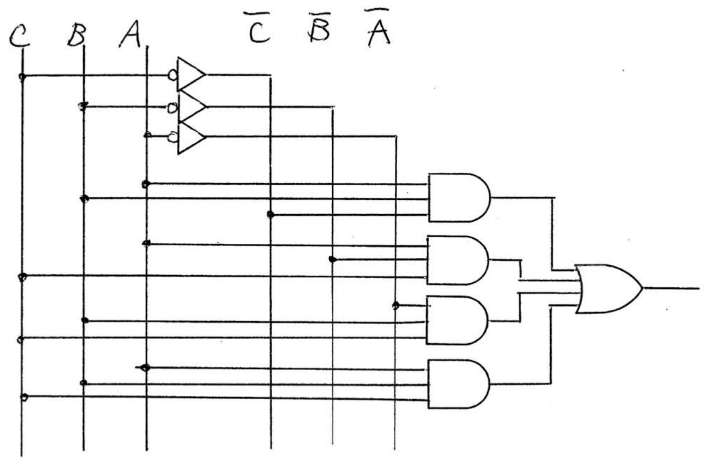
Circuit Design: Sum of Products

- Circuit Design Steps
  - Create a truth table.
  - Identify all products that output 1 or “H”.
  - Create a sum-of-products form.
  - Design the corresponding logical circuit with ANDs and ORs.
- Disadvantages
  - Many ANDs
  - Many inputs to the OR
- **Logic Simplification**
  - **Convert a truth table into a Karnough map.**
  - **Group neighboring products.**

Design a 3-input majority circuit.

C	B	A	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	AB~C
1	0	0	0	
1	0	1	1	A~BC
1	1	0	1	~ABC
1	1	1	1	ABC

F = AB~C + A~BC + ~ABC + ABC



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" truth table
- **Based upon Boolean algebra simplification AB + A~B = A**
  - Why?
    - Third Law of Complementation: **B + ~B = 1**
    - Finally: **A·1 = A**
- Therefore

  AB + A~B = A (B + ~B) = A · 1 = A
- http://sourceforge.net/projects/k-map/files/k-map/0.4/Kmap-04-setup.exe/download

Karnaugh Map – Creation

- Rules for building a Karnaugh Map of a truth table
  - **Rule 1**: For **each output**, you must build **one k-map**
    - E.g., for the 3 x 8 line decoder, we need eight k-maps
  - **Rule 2**: For each k-map, # cells = # of 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 3**: 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 **adjacent to each other**

Example: Karnaugh Map Creation

K-Map for 3 input variables

	~A~B	~AB	AB	A~B
~C				
C				

K-Map for 4 input variables

	~A~B	~AB	AB	A~B
~C~D				
~CD				
CD				
C~D				

K-Map for 5 input variables

	~A~B~C	~A~BC	~ABC	~AB~C	AB~C	ABC	A~BC	A~B~C
~D~E								
~DE								
DE								
D~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

Example: Karnaugh Map Creation 2

- Convert a truth table (w/ 2-4 inputs) into a Karnaugh map

2 inputs

A	B	P
0	0	1
0	1	0
1	0	1
1	1	0

truth table

A \ B	0	1
0	1	0
1	1	0

Kmap

3 inputs

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

truth table

A \ BC	00	01	11	10
0	0	1	0	0
1	1	1	1	1

Kmap

4 inputs

A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

truth table

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

Kmap

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

Design a 3 x 8 Line Decoder

Derive a Boolean equation for each output

K-Map for 3 input variables

	~a~b	**a~b**	**ab**	**~ab**
~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 = ~a ~b ~c

Note: ~(ab) is not equal to ~a~b

~(ab) = ~a + ~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

	~a~b	a~b	ab	~ab
~c				
c	1			

D1 = ~a ~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

Simplification using K-Maps

1. Form **the largest possible loops** of cells containing 1, 2, 4, 8,16, etc., ***adjacent*** "1" terms
2. **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!**
3. Inspect the map for any loops whose terms are all enclosed in other loops and remove those loops
4. Each loop represents a simplified "sum of product" of the logic equation
   - Simplify the loop by **removing any variable that appears in both complemented and un-complemented form**

Based upon Boolean algebra simplification: AB + A~B = A

Third Law of Complementation: **B + ~B = 1**

Finally: **A·1 = A**

Therefore: **AB + A~B = A (B + ~B) = A · 1 = A**

Simplification using K-Maps: Example 1

	~A~B	~AB	AB	A~B
~C~D	1	1		1
~CD	1	1		1
CD			1	
C~D			1	

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

	~A~B	~AB	AB	A~B
~C~D	1			1
~CD				
CD				
C~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**

Simplification using K-Maps: Example 2

	~A~B	~AB	AB	A~B
~C~D	1	1	1	1
~CD	1	1	1	1
CD				
C~D				

This K-Map can really be simplified.

X = ~C

	~A~B	~AB	AB	A~B
~C~D	1	1	1	1
~CD		1		
CD		1		
C~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 = ~D + ~AB

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 a truth table.
  - Draw K-maps to derive Boolean equations.
  - Draw a circuit diagram based on the equations.