Topic 2: Boolean logic system

(布尔逻辑系统)

Problem

- How to represent binary number in hardware
- How to design and implement a binary system

Main Content

- Boolean logic
- Boolean logic representation
- Logic gate (digital circuit)
- Combinational logic gate design
- K-map

1. Boolean Logic

- **Binary Logic it has two opposite states in Boolean logic system**
- **True or false, Yes or No， 1 or 0**

2. Logic operation representation

2.1 Mathmatical Expression

- AND: • or ∧ (product)
- OR: + or ∨ (sum)
- NOT: **’ (prime)**or ¯ *(bar)*
- XOR: ⊕

Combinational logic operation

组合逻辑

- F=AB+A’B’+A’B+AB’

- F=ABC+AB’C+A’BC

- A logic system can be represented by logic operations of inputs

  F(A,B,C)

Background

Boolean algebra law

Law	AND form	OR form
Identity 1(同一律)	A = A’’	A = A’’
Identity 2（同一律）	1A = A	0 + A = A
Null	0A = 0	1 + A = 1
Idempotence(幂等性)	AA = A	A + A = A
Complementarity(互补)	AA’ = 0	A + A’ = 1
Commutativity(交换)	AB = BA	A + B = B + A
Associativity(结合)	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributivity(分配)	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption(吸收)	A(A + B) = A	A + AB = A
de Morgan’s law	(AB)’ = A’ + B’	(A + B)’ = A’B’

proven on next slide

familiar from School!

de Morgan’s Law(德摩根)

- (AB)’ = A’ + B’   (A + B)’ = A’B’   *Very useful*!
- Whenever we see an expression whose sub-expressions are all ANDed together, or all ORed together, we can re-state by
    1. negating the overall expression
    2. negating the sub-expressions
    3. flipping the operators from OR to AND, or vice versa

Examples: Simplify the logic expression

$F=A(A+B)$

$F = AB + A'B + AB'$

$F = ABC + AB'C + ABC' + AB'C'$

$F=(A' + B')'$

$F=A$

$F = A+B$

$F = A$

$F=AB$

“2.2 Truth tables”(真值表)

- Use table to express the logic expression

For any pair of binary digits (bits) A and B…

AND		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

TRUE if, and only if, all inputs are TRUE

F = AB

OR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

TRUE if any input is TRUE

F = AB + A’B + AB’ (A+B)

NOT	
A	Q
0	1
1	0

TRUE if, and only if, the single input is FALSE

F = A’

XOR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

(eXclusive OR)

True if an odd number of inputs are TRUE, otherwise FALSE

F = A’B + AB’

1 is interpreted as meaning TRUE; and 0 as FALSE

NAND and NOR as truth tables…

AND		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

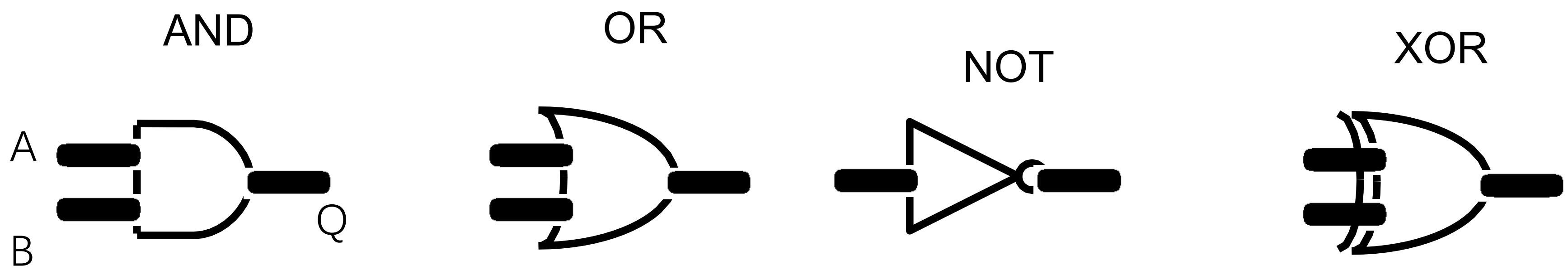
OR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

NAND		
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NOR		
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

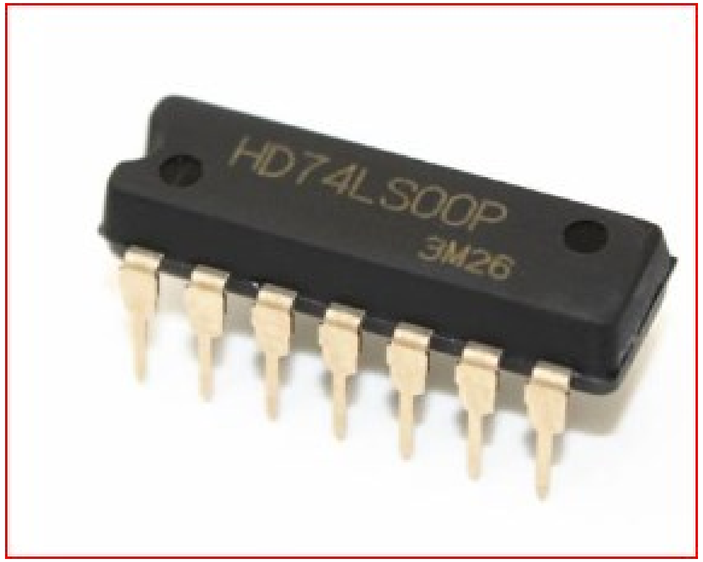
3. Logic gate

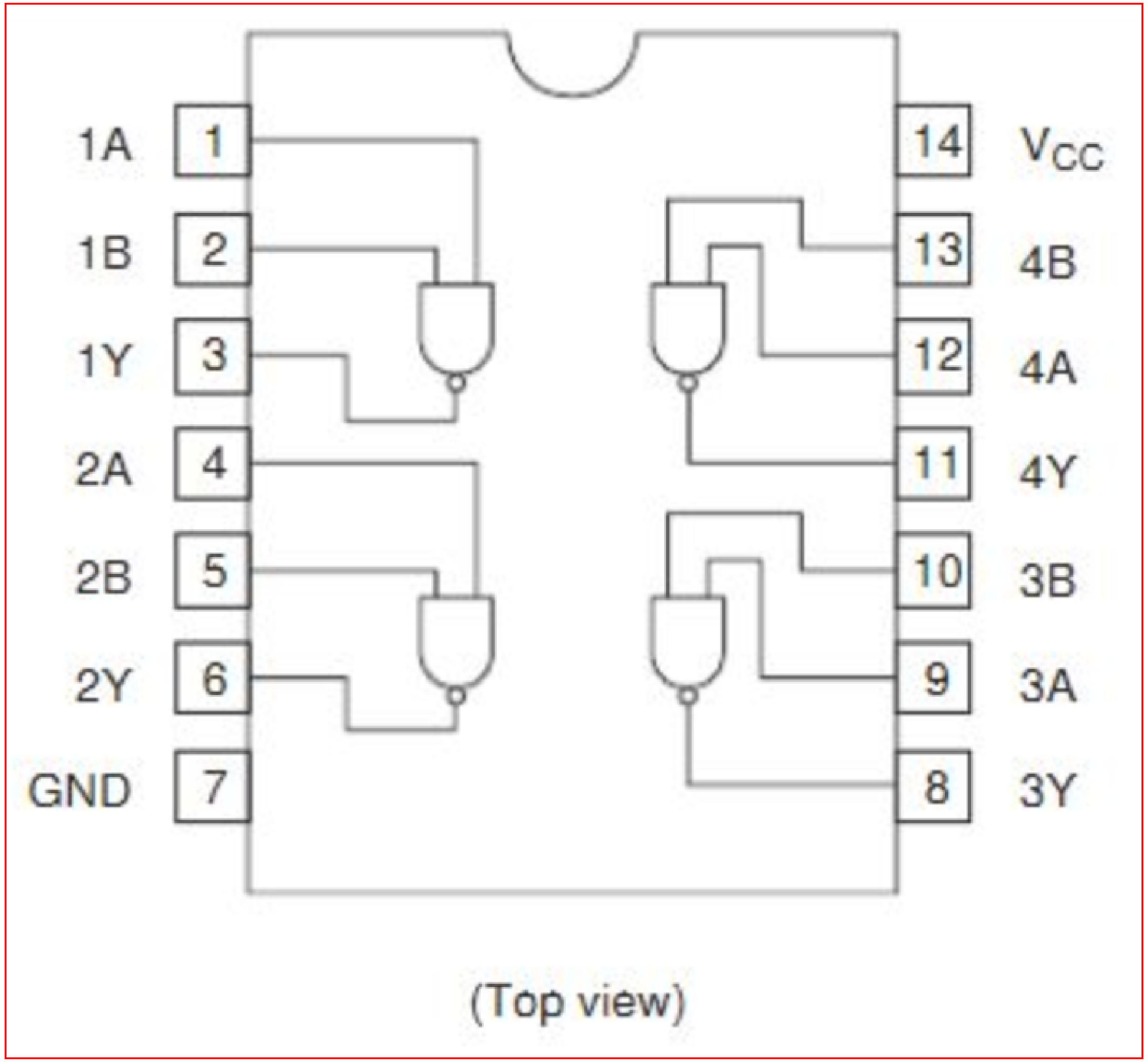
- Gate: switch , a logic unit or component circuit(硬件逻辑单元)



logic gates → circuit symbol

In practice, NANDs and NORs are very commonly used instead of AND/OR/NOT/XOR gates


HD74LS00P
3M26


1A
1B
1Y
2A
2B
2Y
GND
1
2
3
4
5
6
7
14
13
12
11
10
9
8
Vcc
4B
4A
4Y
3B
3A
3Y
(Top view)

What can we use to build computer logic?

Transistors

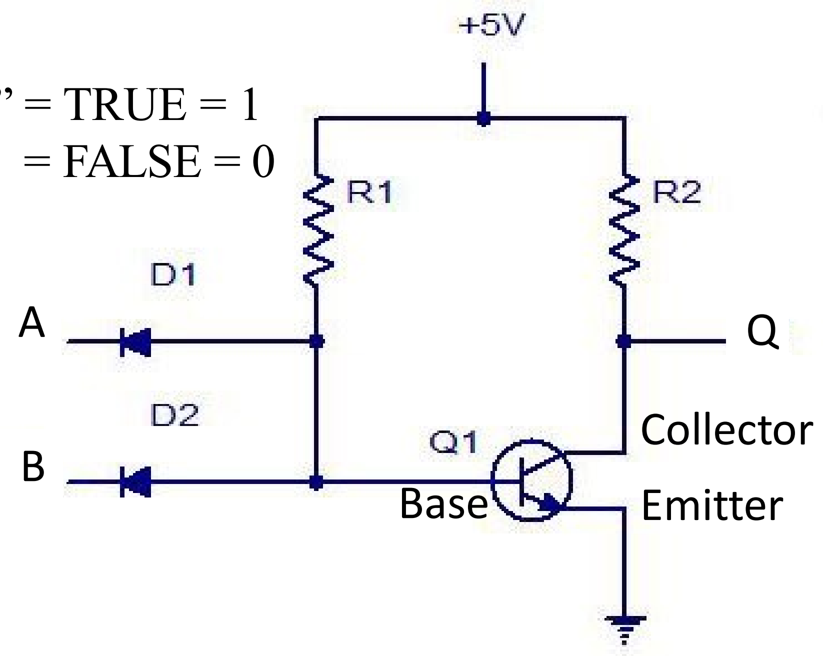
Vacuum Tubes

...anything we can build a switch from

Example: building a ? gate from a transistor

- Applying +5v to inputs A and B "opens" the transistor that so current can flow from the collector to the emitter, taking Q down to 0
  - (analog electronics   can the transistor work in amplifying state)

+5v = "high" = TRUE = 1
0v = "low" = FALSE = 0



NAND		
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

- Automatic switch

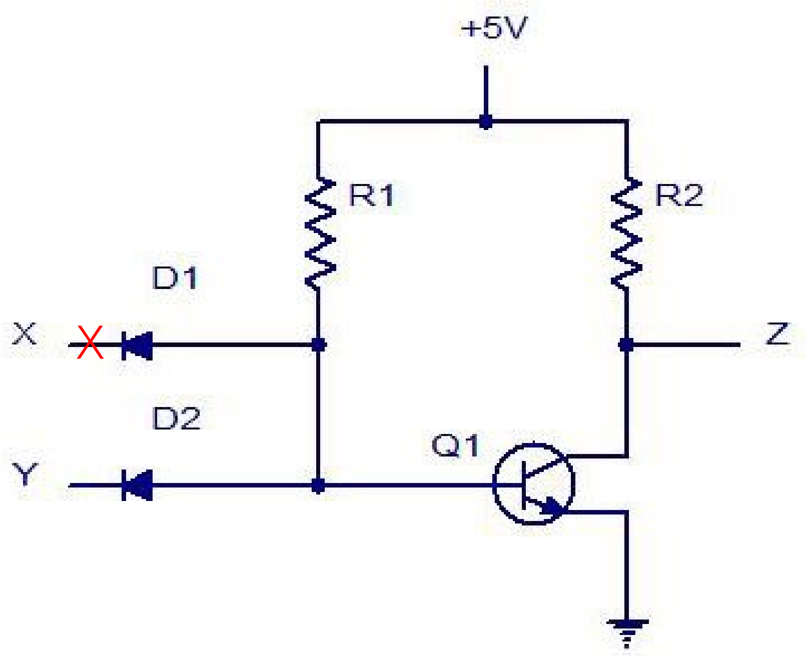
- Controlled by input electric signal

- Signal is high, output is low
- Signal is low, output is high

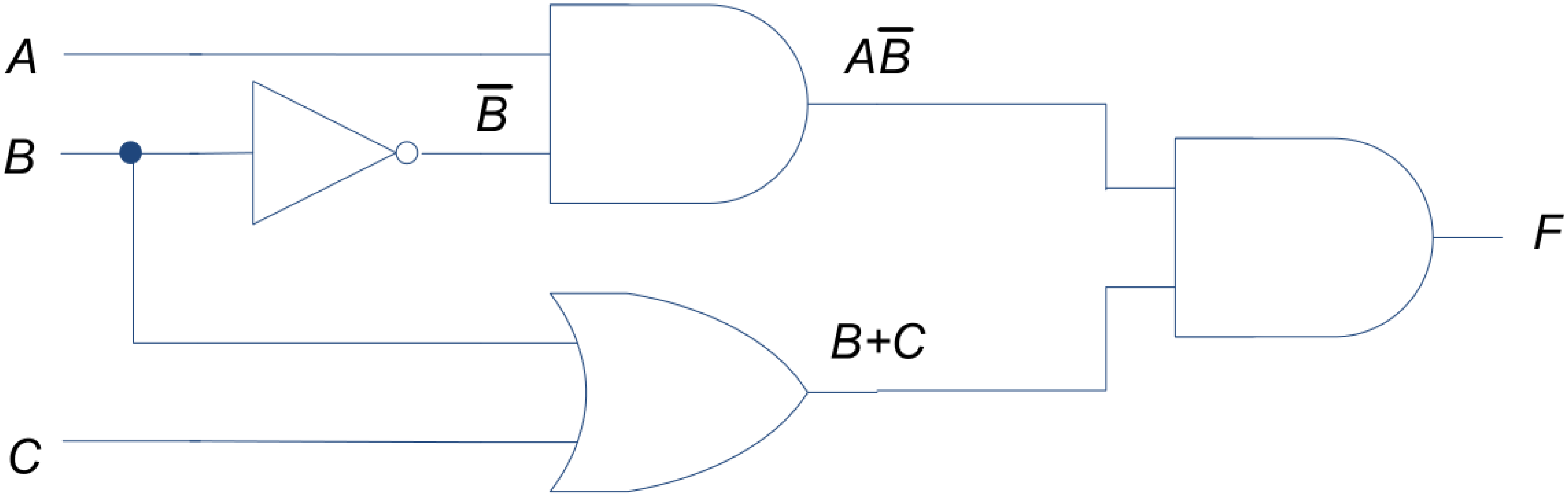
- Not and gate → nand

Example: building a NOT gate from a transistor

Discrete NAND Gate



4. Combinational boolean logic gates design
(组合逻辑电路)



Function of combinational logic gates

- All the arithmetic operations are done by logic gates
  - Addition, subtraction, multiplication, division
- Logic operation: And, Or, Not
- Compare and jump

The approach for design of logic circuits

1. Describe or state your logical function

2. Write out a truth table for the desired logical function

3. Derive a Boolean expression by ORing together all the rows whose "output column" is 1 (only consider truth),
4. input will be negated if input is zero; input will be itself if input is 1
   - ***sum-of-products*** form

5. Translate the Boolean expression to logic gates
   - May need to use Boolean algebra or "Karnaugh maps" (see later) to obtain the simplest mapping to our target types of logic gate

"Truth tables"(真值表)

AND		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
A	Q
0	1
1	0

F = A'

XOR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

F = AB

F = AB + A'B + AB'　(A+B)

F = A'B + AB'

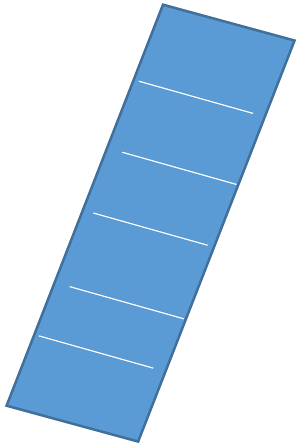
- Representation
  - Description
  - Truth table
  - Boolean logic expression
  - logic gate
- Simplify Using Boolean algebra law and K-map
- Design logic gate, Implement the logic function by NAND

Example

- Two switches control a light
  - We want to switch the light on at the bottom, and off again at the top—and vice versa

    So, the light is on if **S** is *up* and **H** is *down*, **or** if **S** is *down* and **H** is *up, Light is off if S and H are down or up*

Top S



Bottom H

	S	H	Light
	0	0	0
S'H	0	1	1
SH'	1	0	1
	1	1	0

Logic expression

- So, L = S'H + SH'   [*in* ***sum-of-products*** *form*]

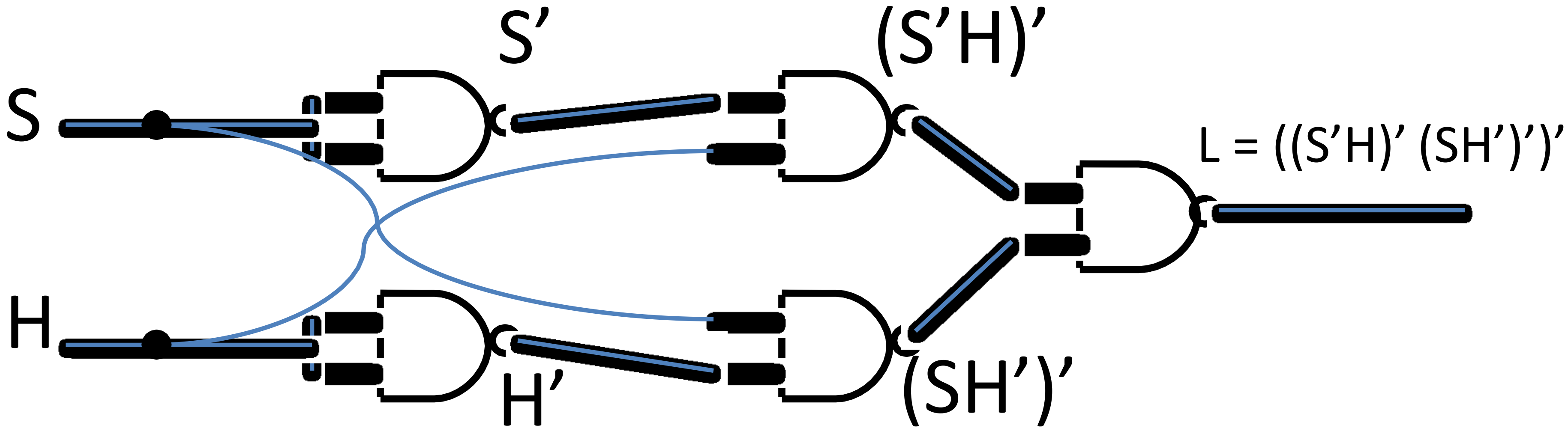
- Let's implement this using NAND gates…

- Apply de Morgan's law
  - By de Morgan's law, X + Y = (X' Y')'
  - Expand to ((S'H)' (SH')')'   So L = ((S'H)' (SH')')'
  - This is now in the required "inverted AND" form…

BEIJING JIAOTONG UNIVERSITY

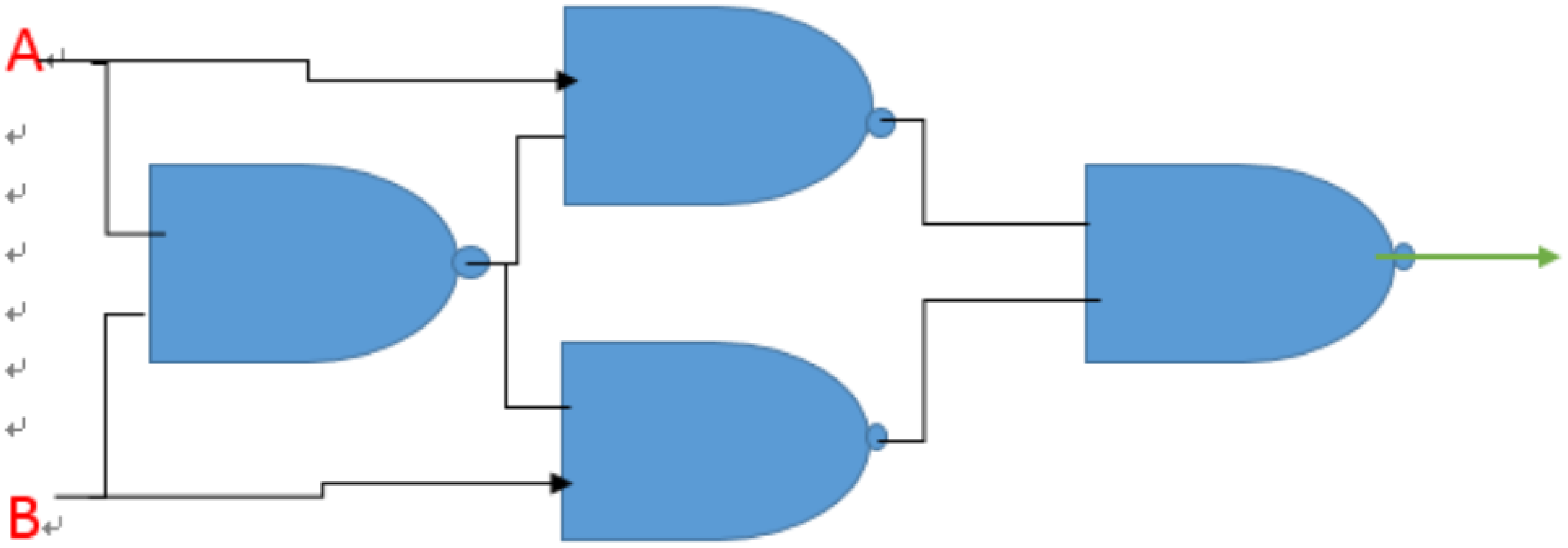
Lancaster University

Logic gates



L = ((S’H)’ (SH’)’)’

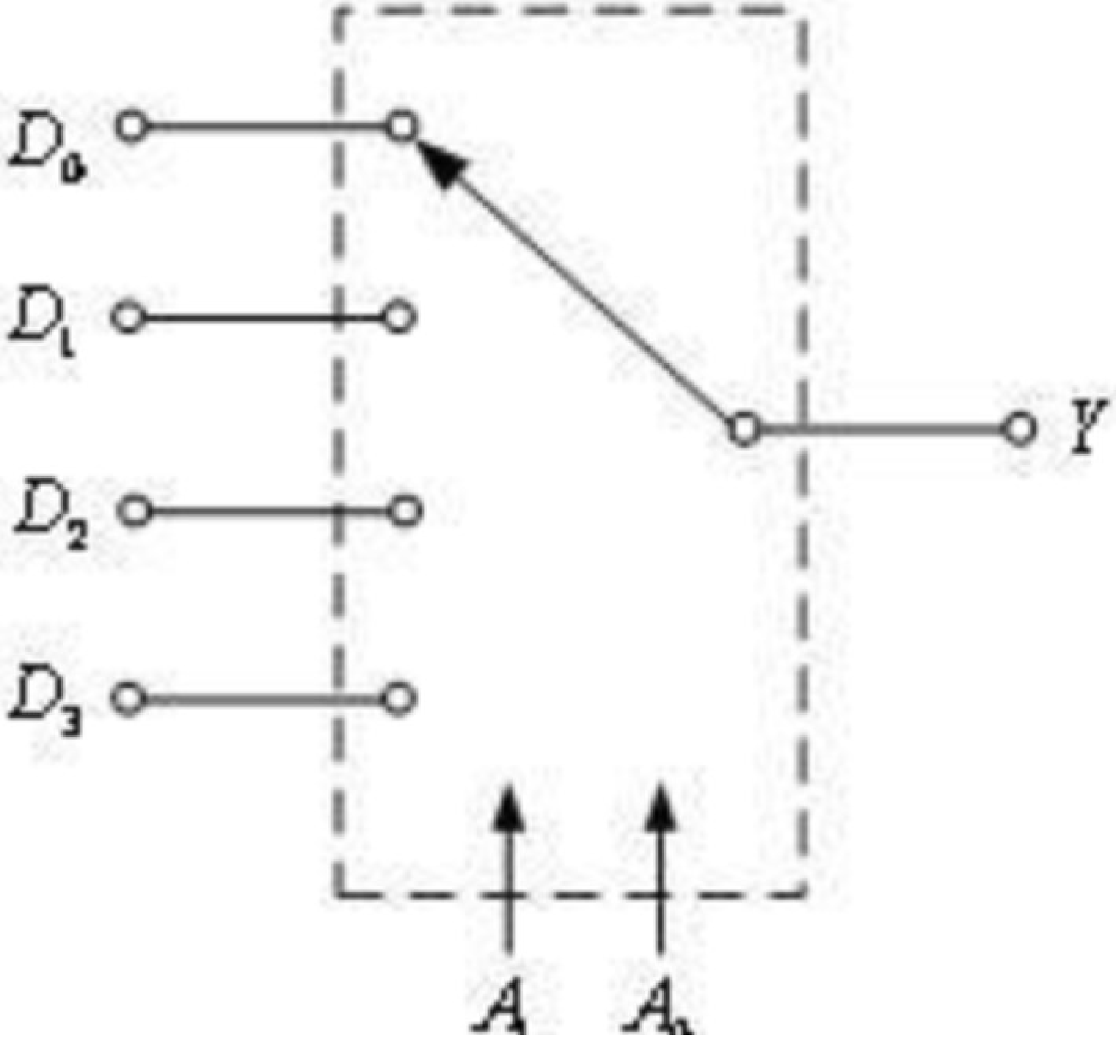
Exercise
Write out following logic function



Y= ((AB)'A)'((AB)'B)')' =((AB')'(A'B)')'

=AB'+A'B

(多路开关)



Multiplexer design

- Write out the truth table and derive the sum-of-products form:

$X = (AC_1'C_2') + (BC_1'C_2) + (CC_1C_2') + (DC_1C_2)$

(each term is taken from an "X=1" row)

an X in a truth table means either 0 or 1 – i.e. "don't care"; this is useful in reducing the number of rows we have to consider!

C_1	C_2	A	B	C	D	X
0	0	1	X	X	X	1
0	0	0	X	X	X	0
0	1	X	1	X	X	1
0	1	X	0	X	X	0
1	0	X	X	1	X	1
1	0	X	X	0	X	0
1	1	X	X	X	1	1
1	1	X	X	X	0	0

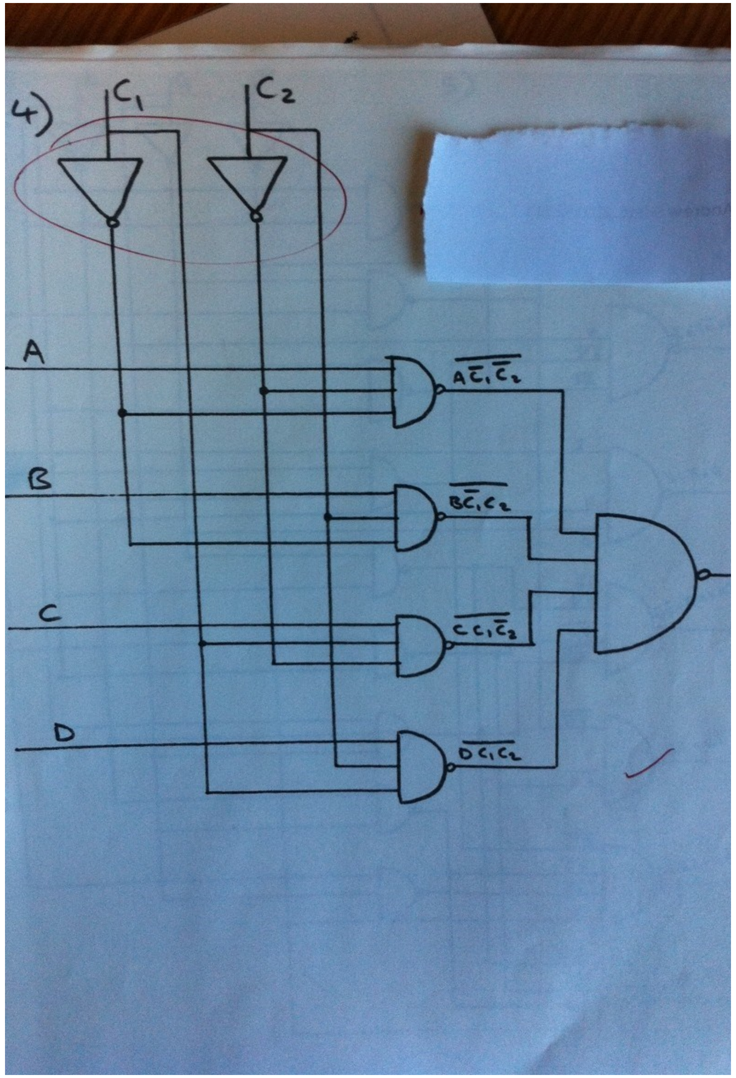
Example *cont.*

- Apply de Morgan’s law to get this into “inverted AND” form

$AC_1'C_2' + BC_1'C_2 + CC_1C_2' + DC_1C_2 =$

$((AC_1'C_2')'(BC_1'C_2)'(CC_1C_2')'(DC_1C_2)')'$

- Now we can map directly to 3-input NANDs and 4-input NAND gate



5 Using Karnaugh maps to minimise Boolean logic functions

卡诺图

What is a Karnaugh map?

- A grid in which each square represents one possible combination of inputs (cf. truth table row)
- USE THE SYMMETRIC WAY TO CREATE A KARNAUGH MAP

	A	A’
B		
B’		

2-input map

	AB	A’B	A’B’	AB’
C				
C’				

3-input map

	AB	A’B	A’B’	AB’
CD				
C’D				
C’D’				
CD’				

4-input map

Using a Karnaugh map

1. Find a places with the required number of inputs, and put a 1 in any square for which we want an output of 1
2. Draw *rectangular groups* of 1s (adjacent 1s is grouped)
   – Groups must contain 2 or 4 or 8 … (2^n) cells
   – Groups may overlap, and may wrap around the edges
   – The *larger* the groups, and the *fewer* the groups, the better

Result: for each group simply list the "unchanged" terms and OR them together ("changed" ones "cancel")

A′BCD + A′B′CD +
A′BC′D + A′B′C′D + AB′C′D +
AB′C′D′ +
ABCD′ + A′BCD′ + A′B′CD′ + AB′CD′

= A′D + AB′C′ + CD′

	AB	A′B	A′B′	AB′
CD		1	1	
C′D		1	1	1
C′D′				1
CD′	1	1	1	1

BEIJING JIAOTONG UNIVERSITY

Lancaster University

Karnaugh map example

- Implement a Decoder function that detects the following inputs: 0, 1, 2, 4 and 5 (assume 3-bit binary)
- Here's the truth table:

Decimal	Binary			Output
	A	B	C	
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

Identify the sum-of-products expression

- $F = A'B'C' + A'B'C + A'BC' + AB'C' + AB'C$

Decimal	Binary			Output	Term
	A	B	C		
0	0	0	0	1	A'B'C'
1	0	0	1	1	A'B'C
2	0	1	0	1	A'BC'
3	0	1	1	0	
4	1	0	0	1	AB'C'
5	1	0	1	1	AB'C
6	1	1	0	0	
7	1	1	1	0	

Now, enter these five terms into a 3- input Karnaugh map template…

- F = A’B’C’ + A’B’C + A’BC’ + AB’C’ + AB’C

	AB	A’B	A’B’	AB’
C	0	0	1	1
C’	0	1	1	1

BEIJING JIAOTONG UNIVERSITY

Lancaster University

Find the groups

- F = A’B’C’ + A’B’C + A’BC’ + AB’C’ + AB’C

	AB	A’B	A’B’	AB’
C			1	1
C’		1	1	1

The **larger** the groups the better
The **fewer** the groups the better…
(doesn’t matter if groups overlap)

BEIJING JIAOTONG UNIVERSITY

Lancaster University

Derive the result…

- Look for the "unchanged" variables in each group

B' is unchanged ("A" cancels: appears as both A and A'; "C" cancels: appears as both C and C')

	AB	A'B	A'B'	AB'
C			1	1
C'		1	1	1

A'C' is unchanged ("B" cancels: appears as both B and B')

Result

- Write down and OR together the“unchanged” variables…
  - **B’** was unchanged
  - **A’C’** was unchanged

- Result is therefore, **F = B’ + A’C’**

Let’s do the same using Boolean algebra

- F = A’B’C’ + A’B’C + A’BC’ + AB’C’ + AB’C

- F = A’B’C’ + A’B’C + A’BC’ + AB’C’ + AB’C
  - Can use *idempotence* to expand:
    - = A’B’C’ + A’B’C + A’BC’ + A’B’C’ + AB’C’ + AB’C
  - Can then use *distributivity* to combine “similar” pairs of terms:
    - = A’B’(C’ + C) + A’(B + B’)C’ + AB’(C’ + C)
  - Can then use… …ementarity and then *identity-2* to simplify:

    = A’B’ + A… …B’
  - Can then use *commutativity* to rearrange:

    = A’B’ + AB’ + A’C’
  - Can then use *distributivity* (again):

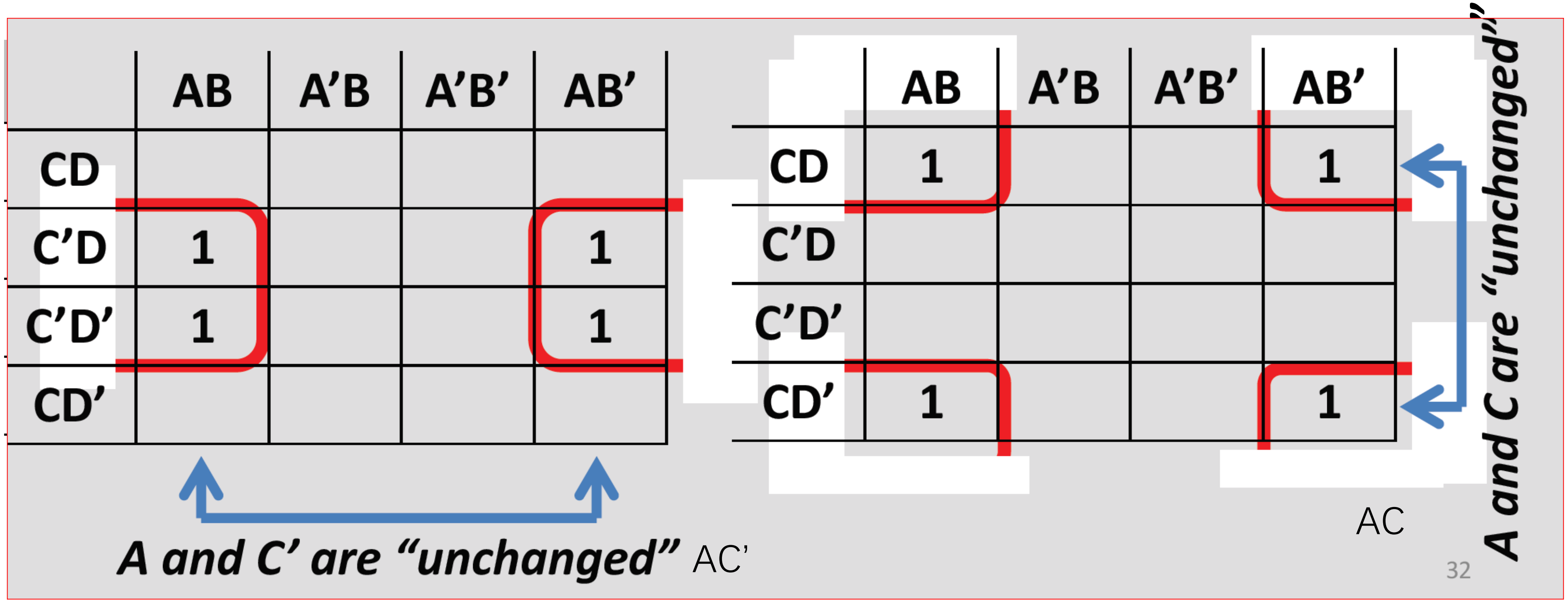
    = **(A’ + A)**B’ + A’C’
  - Can then use *complementarity* and then *identity-2* (again) to simplify:

    = B’ + A’C’

[X+X’=1 and then 1X=X]

Be careful not to miss "wrap around" possibilities

- Remember that groups may "wrap around"
- So, in each of the following examples we have a single group of four cells



Examples

	AB	A'B	A'B'	AB'
CD	1			
C'D		1	1	
C'D'	1	1	1	1
CD'			1	1

	AB	A'B	A'B'	AB'
CD	1			
C'D		1	1	
C'D'	1	1	1	1
CD'			1	1

$$Y = ABCD + A'C' + C'D' + B'D'$$

	AB	A'B	A'B'	AB'
CD	1	1	1	1
C'D	1	1	1	1
C'D'			1	1
CD'	1	1		

Y=D+B'C'+BC

	AB	A’B	A’B’	AB’
CD		1		
C’D	1			1
C’D’				
CD’	1	1		

	AB	A'B	A'B'	AB'
CD		1		
C'D	1			1
C'D'				
CD'	1	1		

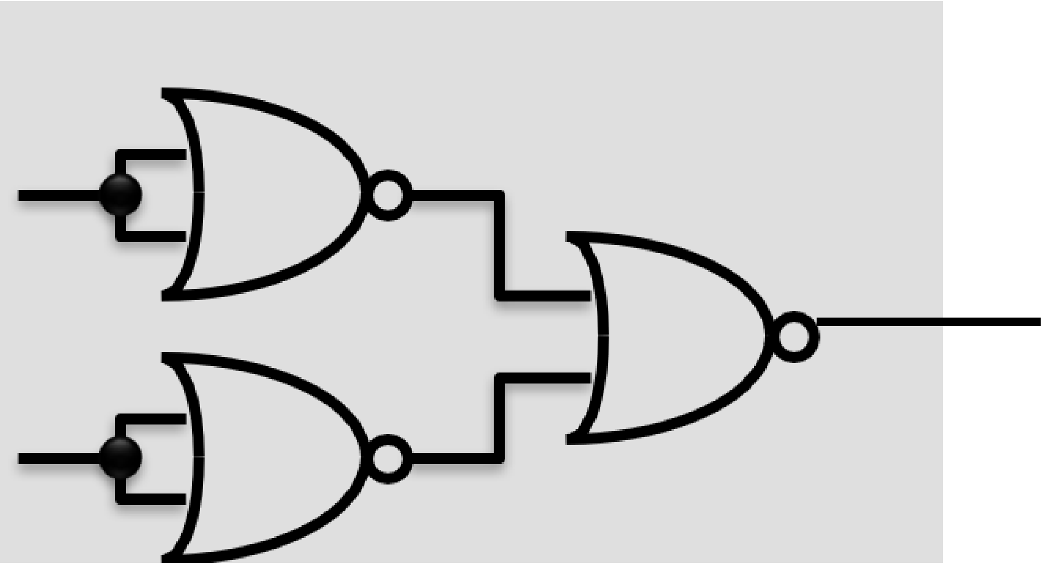
$$AC'D+BCD'+A'BC$$

		00	01	11	10
		$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
0	$\overline{C}$			1	1
1	C	1		1	1

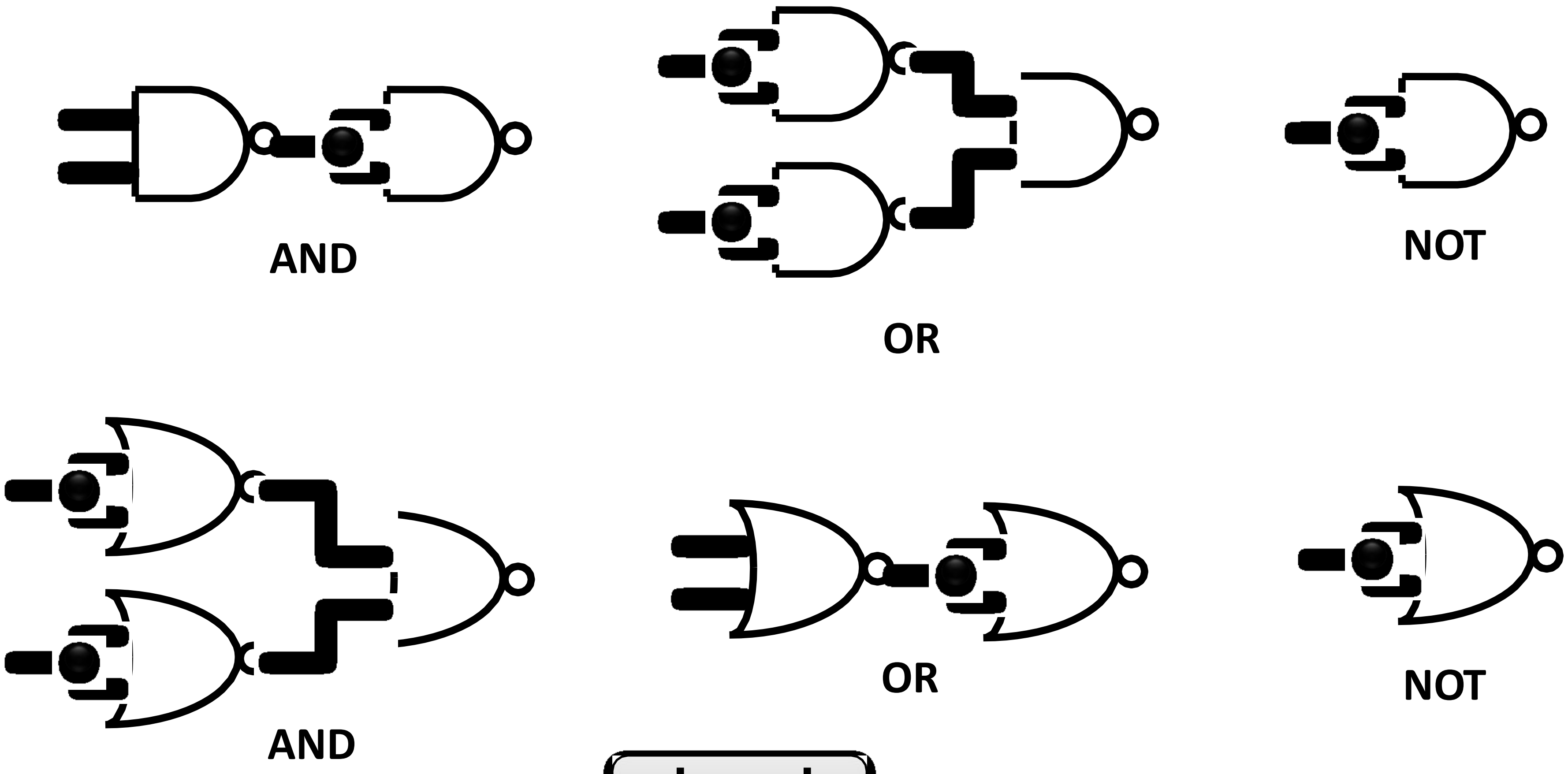
		00	01	11	10
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
00	$\bar{C}\bar{D}$	1	1	1	1
01	$\bar{C}D$	1	1	1	1
11	CD			1	1
10	$C\bar{D}$	1	1		

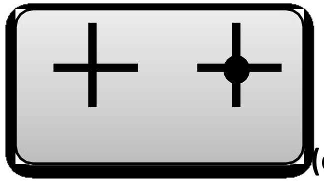
Y=

Write out the logical expression



Building AND/OR/NOT from NANDs/NORs





(we'll see XOR later…)

- How to implement logic operation in program

- A&B
- A|B
- ~A
- A^B
- A,B can be boolean type
- A=1,B=0?

Summary

- Four basic Boolean logic gates
- The laws of Boolean algebra law
- Design logic gate process
  - a logic function specification → a truth table → a "sum of products" logic expression → a logic circuit
- Minimize logic expressions using Karnaugh maps

Design a voter

3 people take a vote

The output is true when the majority agreed