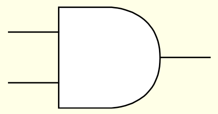
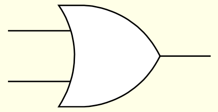
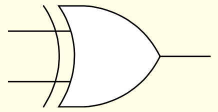
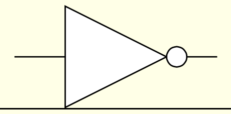
Boolean Logic KO

Define problems using Boolean logic

AND		∧	A ∧ B A AND B
OR		∨	A ∨ B A OR B
XOR		⊻	A ⊻ B A XOR B
NOT		¬	¬A NOT A
The same as		≡	A ≡ B A is the same as B

Truth tables:

AND ∧		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR ∨		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT ¬	
A	Output
0	1
1	0

The XOR gate produces a 1 output if either, but not both of the inputs are 1.

XOR ⊻		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

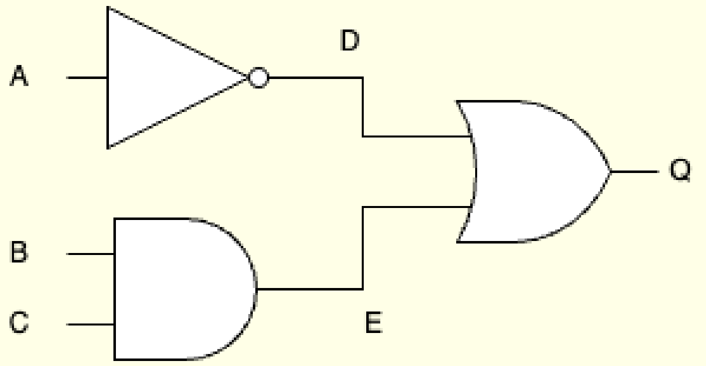
NAND		
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Multiple logic gates can be connected to produce an output based on multiple inputs.

This circuit can be represented by the expression

Q = ¬A ∨ (B ∧ C)

or alternatively as Q = (NOT A) OR (B AND C)

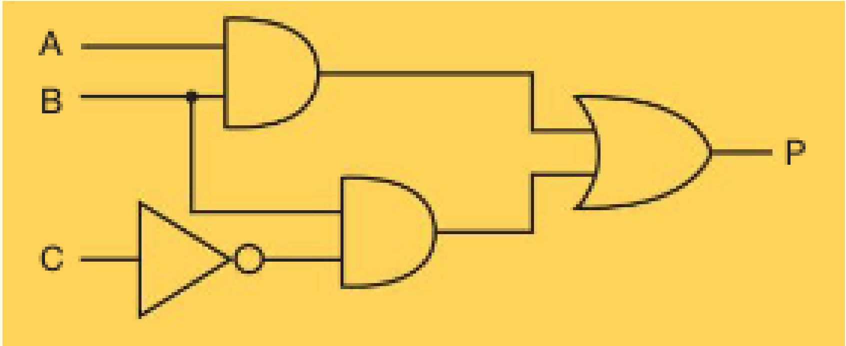


Evaluate the brackets first

Input A	Input B	Input C	D = ¬ A	E = B ∧ C	Output Q = D ∨ E
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

How to write Boolean expression represented in a logic diagram:

Write the Boolean expression represented by the logic diagram below, using AND, OR and NOT instead of symbols. Then write the same expression using symbols.



First write (A AND B).

Then write (B AND NOT C)

These are the inputs to the OR gate

so the expression is:

P = (A AND B) OR (B AND NOT C)

P = (A ∧ B) ∨ (B ∧ ¬ C)

Defining problems with Boolean logic

A boiler has two sensors, a pressure sensor and a temperature sensor. If either the temperature (T) or the pressure (P) is too high, a valve (V) will close.

This can be expressed as V = T v P or alternatively as V = T OR P

The table representing these conditions could be drawn as follows:

Input	Binary value	Condition
T	1	Temperature too high
	0	Temperature not too high
P	1	Pressure too high
	0	Pressure not too high

Worked Example

A chemical process has a sensor to detect a dangerous situation, in which case it sounds an alarm (A). The alarm is sounded if:

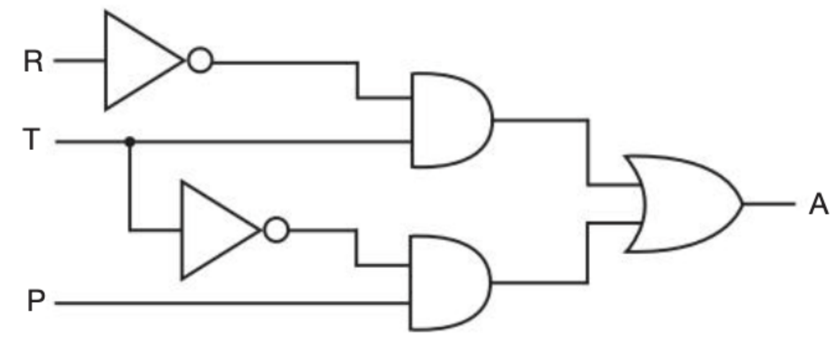
either temperature >= 100°C AND rotator is OFF

or

PH > 6 AND temperature < 100°C

A table can be drawn to represent these conditions as Boolean values.

Input	Binary value	Condition
T	1	Temperature >= 100°C
	0	Temperature < 100°C
R	1	Rotator ON
	0	Rotator OFF
P	1	PH > 6
	0	PH <= 6



The conditions can be written as

$A = (T \wedge \neg R) \vee (P \wedge \neg T)$ or alternatively as A = (T AND NOT R) OR (P AND NOT T)

Input R	Input T	Input P	$X = T \wedge \neg R$	$Y = P \wedge \neg T$	$A = X \vee Y$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	0	0	0

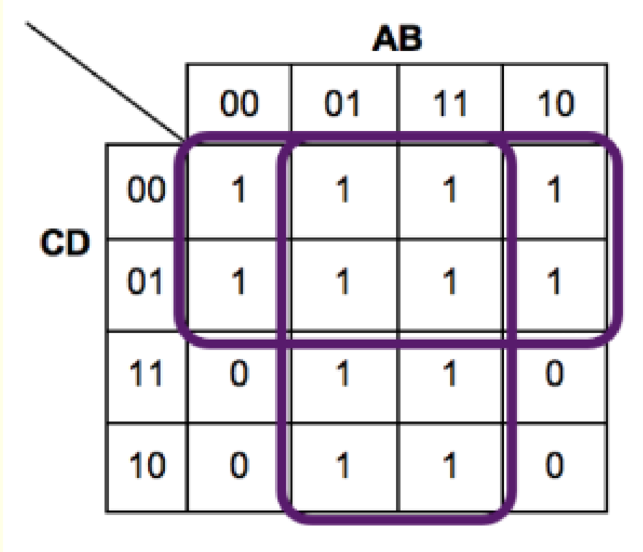
Karnaugh maps (K Maps)

The four-variable problem

- A K Map is a tool that is used for **simplifying Boolean algebra expressions**
- A **visual method** of **grouping together expressions** with **common factors**
- The format of the map makes it **easy to identify** and **eliminate redundant terms**
- They are used in digital logic design, such as simplifying the logic of digital circuits

1. **Grouping the 1's**
   - Make rectangular groups
   - Make groups that are as large as possible
   - Make groups that contain either 8,4,2 or 1 ones
   - Groups can overlap (i.e. some ones can be in multiple groups)
   - The Karnaugh map 'grid' wraps round in all directions so the groups can wrap round
2. **Write a term for each group** by finding the variables that **stay the same** in that group. If a variable is 0 in the group → use **NOT** of it.
3. **OR the group terms together** to form the simplified expression. (Use v)

Example: Give a simplified version of the expression using the Karnaugh map.



You must show your working [3]

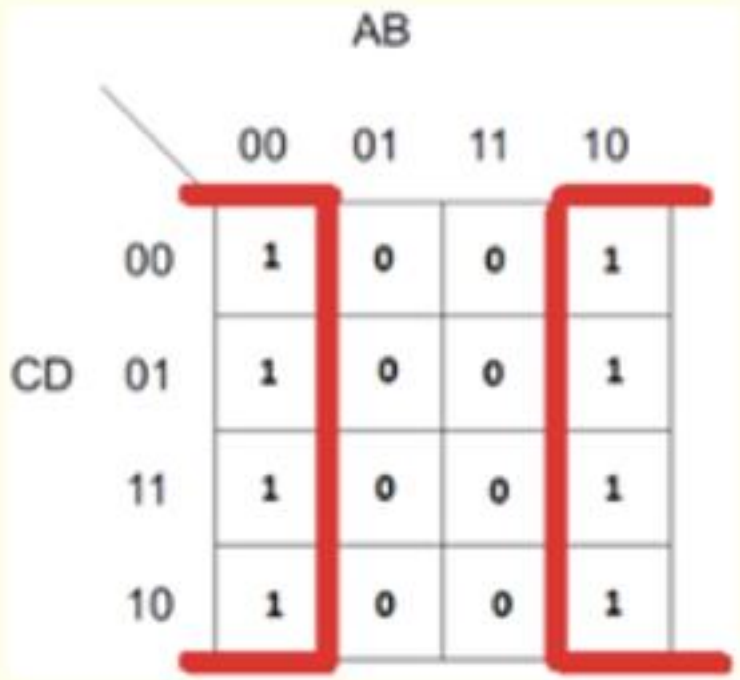
Looking at the group of 8 in the middle of the map, for all the cells in the group the variable B is always a 1

The 3 other variables change across the group (i.e. for some cells they are 0 and for others they are 1
So this group is B

Looking at the other group of 8, for all the cells in this group, C is always 0 but the other 3 variables can be 1 or 0 in this group So this group simplifies to ¬C

B v ¬C

Example 2: Wrap around



Simplified ¬B

Simplifying Boolean Expressions

Boolean Simplification Rules

Rule name	Original expression	Simplified form
AND	X AND 1	X
	X AND 0	0
	X AND X	X
	NOT X AND X	0
OR	X OR 0	X
	X OR 1	1
	X OR X	X
	NOT X OR X	1

Rule name	Original expression	Simplified form
De Morgan's Law Flip AND ↔ OR, move NOT inside	¬(A ∧ B)	¬A ∨ ¬B
	¬(A ∨ B)	¬A ∧ ¬B
Distribution Expanding brackets	A ∧ (B ∨ C)	(A ∧ B) ∨ (A ∧ C)
	A ∨ (B ∧ C)	(A ∨ B) ∧ (A ∨ C)
Reverse distribution (factoring) Factor out common term	(A ∧ B) ∨ (A ∧ C)	A ∧ (B ∨ C)
	(A ∨ B) ∧ (A ∨ C)	A ∨ (B ∧ C)
Association Brackets removable	(A ∧ B) ∧ C	A ∧ B ∧ C
	(A ∨ B) ∨ C	A ∨ B ∨ C
Commutation Order doesn't matter	A ∨ B	B ∨ A
	A ∧ B	B ∧ A
Double negation Two NOTs cancel	¬¬A	A
Absorption If A is true, result is true	A ∧ (A ∨ B)	A
	A ∨ (A ∧ B)	A

Worked Example: Boolean Simplification

Step	Explanation	Expression
Original	Given expression	(¬C ∧ ¬D) ∨ (C ∧ ¬D)
Step 1	Identify the repeated term in both brackets	Common term: ¬D
Step 2	Factor out the common term (reverse distribution / factoring)	¬D ∧ (¬C ∨ C)
Step 3	Apply General OR rule: NOT X OR X = 1	¬D ∧ 1
Final	Simplified expression	**¬D**

More simplified versions of the following Boolean expressions and the rule applied:

Original expression	Simplified expression	Rule(s) applied
¬¬A	A	Double negation
¬A ^ ¬B	¬(A ∨ B)	De Morgan's Law
A ∨ (A ^ B) ∨ ¬A	1	Absorption, Complement

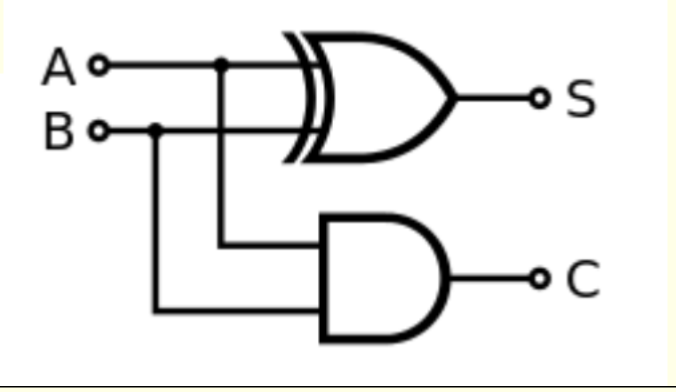
Adders and Flip Flops

Half Adder
A half adder has two inputs, A and B, and two outputs, Sum and Carry. The circuit is formed from just two logic gates: AND and XOR.

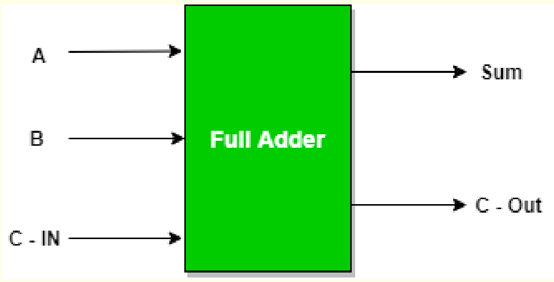
S = SUM
C – Carry

0+0 = 0 SUM 0 No Carry 0
1+0 = 1 SUM 1 No Carry 0
1+1 =1 0 SUM 0 Carry 1
1+1+1=11 SUM 1 Carry 1

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder A full adder is similar to a half adder, but has an additional input, allowing for carry in to be represented.



A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

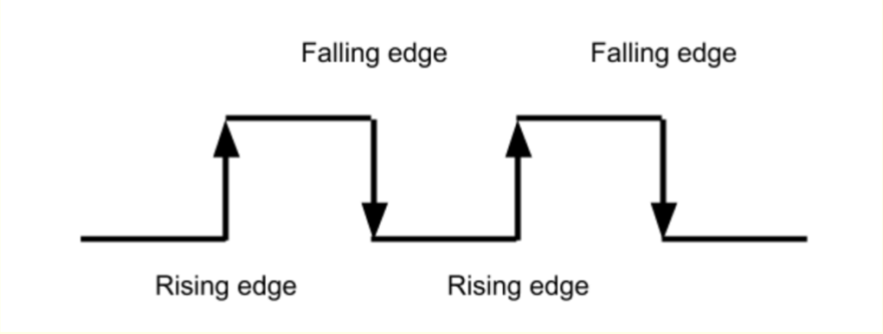
Cover the Cout and SUM columns in the Truth Table and fill in the table yourself

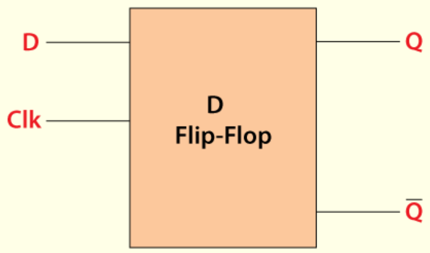
D Type Flip Flops

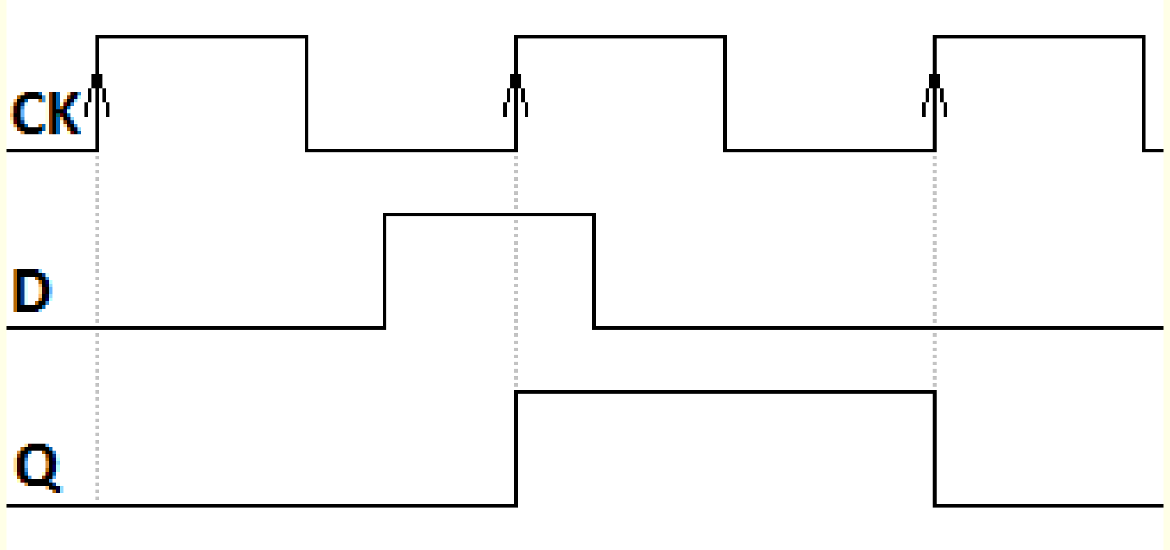
A flop flop is a type of logic circuit which can store the value of one bit.

A flip flop has two inputs, a control signal and a clock input.

Clock Pulse







A D-type flip flop can only change at a rising edge, the start of a clock tick.

D – INPUT
Q – OUTPUT