A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

15.2.1 The full adder, flip-flops drawn and traced, and four-variable Karnaugh maps

Name: _____ Class: _____ Date: _____ **Total: 30 marks**

KEY WORDS boolean 布尔 • karnaugh map 卡诺图 • boolean algebra 布尔代数 • flip-flop 触发器 • de morgan's laws 德摩根定律

Objective

Build the skills to answer the **Boolean algebra** 布尔代数 questions at the depth Paper 3 sets them. Sheet 15.2 touches all four statements: it simplifies a two-variable expression, reads a **K-map** group, completes a **half adder** and asks for one difference between the flip-flops. This sheet is the harder half of the same four: the **full adder** 全加器, **drawing** an **SR** and a **JK flip-flop** 触发器 and deriving its truth table, and a **four-variable** K-map where the groups **wrap round the edges**.

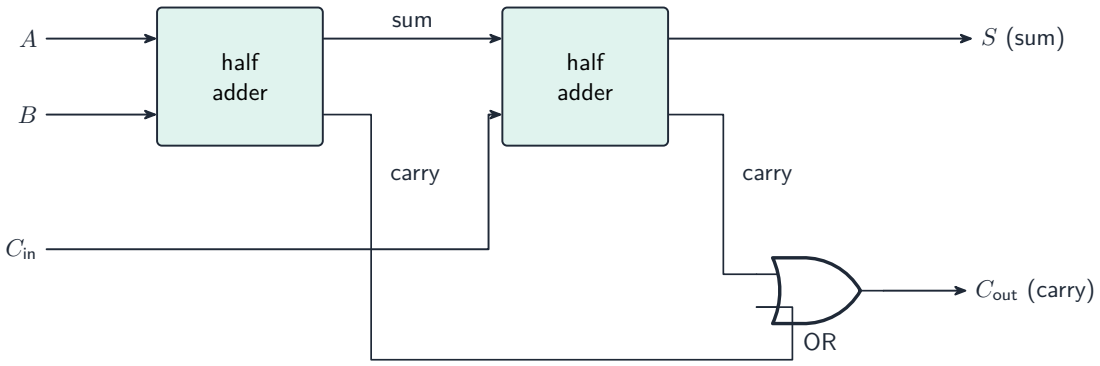
You must be able to:

- produce truth tables for logic circuits including half adders and **full adders**, with gates of more than two inputs
- **draw a logic circuit** for an SR and a JK flip-flop and **derive its truth table**, and explain the role of a flip-flop as a data storage element
- perform Boolean algebra using **De Morgan's laws**, and simplify an expression or a circuit
- solve logic problems using **Karnaugh maps**, and state the benefits of using them

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ The full adder



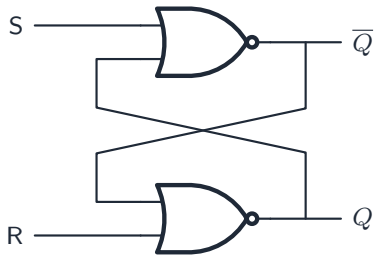
A **half adder** adds two bits and has no way to take a carry **in**, so it can only ever add the rightmost column. A **full adder** takes three inputs – A, B and C_{in} – and is what every other column needs.

$$\text{Sum} = A \oplus B \oplus C_{\text{in}} \qquad C_{\text{out}} = A \cdot B + C_{\text{in}} \cdot (A \oplus B)$$

A	B	C_{in}	**Sum**	C_{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

- Read the Sum column: it is 1 on the rows with an **odd** number of 1s, which is what a chain of XORs does. The carry out is 1 whenever **at least two** inputs are 1.
- A full adder is built from **two half adders and an OR gate**. Saying so earns the structure mark when the question asks how one is constructed.

■ Flip-flops: drawing them and deriving the table

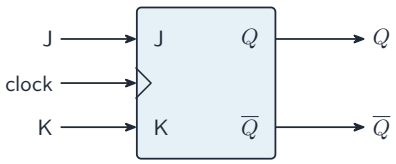


two NOR gates, each output fed back to the other's input

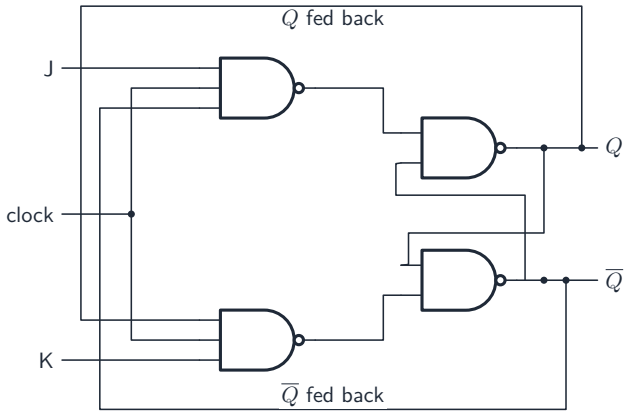
S	R	Q	$\overline{Q}$	effect
0	0	Q	$\overline{Q}$	hold: no change (memory)
1	0	1	0	set Q to 1
0	1	0	1	reset Q to 0
1	1	0	0	invalid: both outputs 0

An **SR flip-flop** is two **NOR** gates **cross-coupled**: the output of each is an input to the other. That feedback is what lets it hold a value with no input applied, which is why a flip-flop is a **data storage element** – one bit of it.

S	R	Q	**What it does**
0	0	Q	**hold** the previous value
0	1	0	**reset**
1	0	1	**set**
1	1	–	**invalid**: both outputs would be 0, so Q and $\overline{Q}$ are no longer opposites



block symbol



build from NAND gates

The **JK flip-flop** exists to remove that invalid state. Its first three rows behave exactly as SR does, and the fourth **toggles**:

J	K	$Q_{\mathbf{next}}$
0	0	Q (hold)
0	1	0 (reset)
1	0	1 (set)
1	1	$\overline{Q}$ (**toggle**)

- The JK is also **clocked**, so it changes only on a clock edge. That makes a chain of them predictable, which is what a register or a counter is built from.
- "State one difference" wants the **invalid state**: SR has one, JK replaces it with a toggle.

De Morgan's laws, and simplifying

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

In words: **break the bar and change the sign**. Both are needed, in both directions.

The laws worth recognising on sight:

Law	
$A + \overline{A}B = A + B$	absorption with a complement
$A(A + B) = A$	absorption
$AB + A\overline{B} = A$	B makes no difference, so it goes
$\overline{\overline{A} + B} = A \cdot \overline{B}$	De Morgan, then double negation

Worked example. *Simplify* $\overline{\overline{A} + B}$.

De Morgan on the whole bar: $\overline{\overline{A}} \cdot \overline{B}$. Then $\overline{\overline{A}} = A$, so the answer is $A \cdot \overline{B}$.

A four-variable Karnaugh map

three variables: the worked example

$C\backslash AB$	00	01	11	10
0	1	1	0	1
1	1	1	0	1

red loop of 4 (columns 00, 01): $A = 0$ throughout, so $\overline{A}$
blue loop of 4 wrapping round (columns 00, 10): $B = 0$, so $\overline{B}$
$Z = \overline{A} + \overline{B}$

four variables: a wrap-around loop

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

the four corners are one loop of 4: $B = 0$ and $D = 0$
in every corner, so the term is $\overline{B}\,\overline{D}$

Two rules make a K-map work, and both are about the **order**:

1. The rows and columns run in **Gray code** – `00`, `01`, `11`, `10` – **not** in binary counting order. Adjacent cells then differ in exactly one variable, which is what lets a group cancel that variable.
2. The map **wraps round**: the left column is adjacent to the right, and the top row to the bottom. So the **four corners** form a legitimate group of four.

To use one: mark a 1 in every cell where the output is 1; make the **largest possible** groups of 1, 2, 4 or 8 adjacent 1s, overlapping where that makes a group bigger; then write each group as the variables that **stay the same** across it, and OR the groups together.

Worked example. *The output is 1 for* $ABCD = 0000$, 0010, 1000 *and* 1010 *only.*

Those are the four **corners**. Across all four, $B = 0$ and $D = 0$, while A and C each take both values and so cancel.

$$F = \overline{B} \cdot \overline{D}$$

- The **benefits** of a K-map, which is a question in its own right: it gives a **simplified** expression directly, without the algebra; the simplified circuit uses **fewer gates**, so it is cheaper, smaller and faster; and grouping is a visual step, so it is **less error-prone** than a chain of algebraic manipulations.
- A group must be a power of two in size, and rectangular. Three adjacent 1s are grouped as a 2 and an overlapping 2, never as a 3.

2 Practice

Now use the methods above.

2.1 State the **two** components a full adder has that a half adder does not, in terms of its inputs and what it is built from. [2]

2.2 Give the Boolean expression for the **Sum** output and for the **carry out** of a full adder. [2]

2.3 Use De Morgan's laws to simplify $\overline{\overline{A} + B}$. [2]

2.4 State what happens to the output of a **JK** flip-flop when $J = 1$ and $K = 1$. [1]

2.5 State why the rows and columns of a Karnaugh map are written in **Gray code** rather than in binary counting order. [2]

3 Exam-style questions

3.1 Complete the truth table for a **full adder** with inputs A, B and C_{in}. [4]

A	B	$C_{\textbf{in}}$	**Sum**	$C_{\textbf{out}}$
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

3.2 (a) Draw the logic circuit for an **SR flip-flop**, labelling the inputs S and R and the outputs Q and $\overline{Q}$. [3]

(b) Complete the truth table for the flip-flop, and state which row is **invalid** and why.[4]

3.3 Explain why a flip-flop can act as a **data storage element**. [2]

3.4 A logic circuit with four inputs A, B, C and D gives an output of 1 only when $ABCD$ is 0000, 0010, 1000 or 1010.

(a) Use a Karnaugh map to obtain the simplified Boolean expression for the output. [3]

(b) State **two** benefits of using a Karnaugh map rather than Boolean algebra. [2]

3.5 Simplify $Z = A + \overline{A}B$, showing each step and naming the law used. [3]

Solutions

Each answer shows the steps, then the **result**.

2.1

- it has a **third input**, the carry in C_{in}, so it can add a column that receives a carry from the column to its right
- it is built from **two half adders and an OR gate**

2.2

- Sum $= A \oplus B \oplus C_{\text{in}}$
- $C_{\text{out}} = A \cdot B + C_{\text{in}} \cdot (A \oplus B)$ – any equivalent form is accepted

2.3

- De Morgan on the whole bar: $\overline{\overline{A}} \cdot \overline{B}$
- double negation gives $\mathbf{A} \cdot \overline{\mathbf{B}}$

2.4

- the output **toggles**: it changes to the opposite of its previous value

2.5

- in Gray code, **adjacent cells differ in exactly one variable**
- so a group of adjacent 1s has that one variable taking both values, which lets it be cancelled; in binary counting order adjacent cells could differ in two variables and grouping them would be invalid

3.1

- Sum column correct: 0, 1, 1, 0, 1, 0, 0, 1
- C_{out} column correct: 0, 0, 0, 1, 0, 1, 1, 1

A	B	C_{in}	**Sum**	C_{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

- the Sum is 1 on the rows with an **odd** number of 1s; the carry out is 1 whenever at least two inputs are 1

3.2

(a) two **NOR** gates

- **cross-coupled**: the output of each is an input to the other
- S and R labelled as the remaining input of one gate each, and the outputs labelled Q and $\overline{Q}$

(b) $S = 0$, $R = 0$: Q **holds** its previous value

- $S = 0$, $R = 1$: $Q = 0$, reset; $S = 1$, $R = 0$: $Q = 1$, set
- $S = 1$, $R = 1$ is the **invalid** row
- because both outputs would be forced to 0, so Q and $\overline{Q}$ would no longer be complements of each other, and what happens next depends on which input changes first

3.3

- the outputs are **fed back** as inputs, so with both inputs at 0 the circuit holds whatever state it was last put into
- that state persists until it is deliberately set or reset, so one flip-flop stores **one bit**; a register is a row of them

3.4

(a) the four cells are the **corners** of the map, and a K-map **wraps round**, so they form one valid group of four

- across all four, $B = 0$ and $D = 0$; A and C each take both values and cancel
- $\mathbf{F = \overline{B} \cdot \overline{D}}$
- four separate groups of one, giving four ANDed terms, earns the first mark only: the question is about grouping

(b) any two: it gives the simplified expression **directly**, without a chain of algebra

- the simplified circuit needs **fewer gates**, so it is cheaper, smaller and faster; and grouping is visual, so it is less error-prone than algebraic manipulation

3.5

- $A + \overline{A}B$: apply the **distributive** law, $A + \overline{A}B = (A + \overline{A})(A + B)$
- $A + \overline{A} = 1$ – the **complement** law
- $1 \cdot (A + B) = \mathbf{A + B}$ – the identity law
- quoting the absorption result $A + \overline{A}B = A + B$ directly earns the answer mark but not the working marks, and the question says to show each step