Exercise walk-through

15.2.1

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

A-Level Computer Science

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 The method

Method — 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.

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

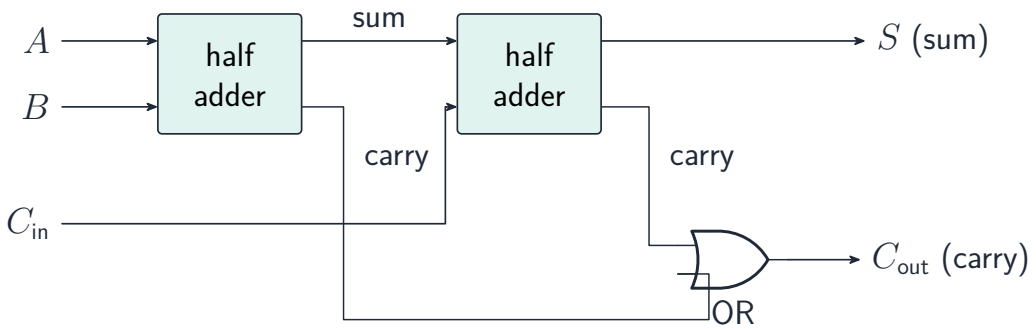
A	B	C_{in}	**Sum**	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Method — The full adder

- 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.

Method — 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.



Method — Flip-flops: drawing them and deriving the table

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

Method — Flip-flops: drawing them and deriving the table

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

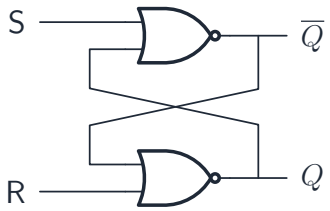
J	K	$Q_{\textbf{next}}$
0	0	Q (hold)
0	1	0 (reset)
1	0	1 (set)
1	1	$\overline{Q}$ (**toggle**)

Method — Flip-flops: drawing them and deriving the table

- 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.

Method — Flip-flops: drawing them and deriving the table

center The JK flip-flop exists to remove that invalid state.

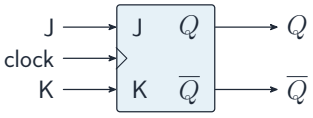


two NOR gates, each output fed back to the other's input

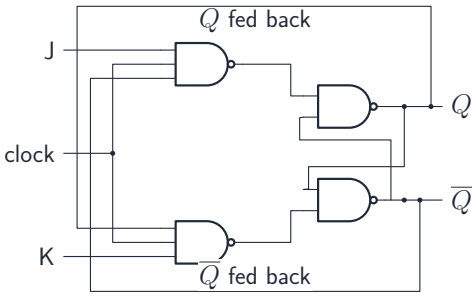
S	**R**	Q	$\overline{Q}$	**effect**
0	0	Q	$\overline{Q}$	hold: no change (memory)
1	0	1	0	set Q to 1
0	1	0	1	reset Q to 0
1	1	0	0	invalid: both outputs 0

Method — Flip-flops: drawing them and deriving the table

center The JK flip-flop exists to remove that invalid state.



block symbol



build from NAND gates

A JK flip-flop, with J, K and a clock input, whose $J = K = 1$ row toggles the output

Method — 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

Method — De Morgan's laws, and simplifying

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 $A \cdot \overline{B}$.

Method — A four-variable Karnaugh map

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.

Method — A four-variable Karnaugh map

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.

Method — A four-variable Karnaugh map

- 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.

Method — A four-variable Karnaugh map

Two rules make a K-map work, and both are about the order : To use one: mark a 1 in every cell where the...

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 $\overline{B}\,\overline{D}$

2 Practice

Practice 2.1 ▪ 2 marks

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.

Solution 2.1

Method: The full adder — p.4

Worked steps

- 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 B1
- it is built from **two half adders and an OR gate** B1

Practice 2.2 ▪ 2 marks

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

Solution 2.2

Method: The full adder — p.4

Worked steps

- Sum $= A \oplus B \oplus C_{\text{in}}$ **B1**
- $C_{\text{out}} = A \cdot B + C_{\text{in}} \cdot (A \oplus B)$ – any equivalent form is accepted **B1**

Practice 2.3 ▪ 2 marks

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

Solution 2.3

Method: De Morgan's laws, and simplifying — p.12

Worked steps

- De Morgan on the whole bar: $\overline{\overline{A}} \cdot \overline{B}$ M1
- double negation gives $\mathbf{A} \cdot \overline{\mathbf{B}}$ A1

Practice 2.4 ▪ 1 mark

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

Solution 2.4

Method: Flip-flops: drawing them and deriving the table — p.7

Worked steps

- the output **toggles**: it changes to the opposite of its previous value B1

Practice 2.5 ▪ 2 marks

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

Solution 2.5

Method: A four-variable Karnaugh map — p.14

Worked steps

- in Gray code, **adjacent cells differ in exactly one variable** B1
- 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 B1

3 Exam-style

Exam-style 3.1 ▪ 4 marks

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

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

Solution 3.1

> Method: The full adder — p.4

Worked steps

- Sum column correct: 0, 1, 1, 0, 1, 0, 0, 1 **B1** **B1**
- C_{out} column correct: 0, 0, 0, 1, 0, 1, 1, 1 **B1** **B1**

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

Solution 3.1

- 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

Exam-style 3.2 (a) ▪ 3 marks

Draw the logic circuit for an **SR flip-flop**, labelling the inputs S and R and the outputs Q and $\overline{Q}$.

Solution 3.2 (a)

Method: Flip-flops: drawing them and deriving the table — p.7

Worked steps

two **NOR** gates B1

- **cross-coupled**: the output of each is an input to the other B1
- S and R labelled as the remaining input of one gate each, and the outputs labelled Q and $\overline{Q}$ B1

Exam-style 3.2 (b) ▪ 4 marks

Complete the truth table for the flip-flop, and state which row is **invalid** and why.

Solution 3.2 (b)

Worked steps

$S = 0$, $R = 0$: Q **holds** its previous value B1

- $S = 0$, $R = 1$: $Q = 0$, reset; $S = 1$, $R = 0$: $Q = 1$, set B1
- $S = 1$, $R = 1$ is the **invalid** row B1
- 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 B1

Exam-style 3.3 ▪ 2 marks

Explain why a flip-flop can act as a **data storage element**.

Solution 3.3

Method: Flip-flops: drawing them and deriving the table — p.7

Worked steps

- the outputs are **fed back** as inputs, so with both inputs at 0 the circuit holds whatever state it was last put into B1
- that state persists until it is deliberately set or reset, so one flip-flop stores **one bit**; a register is a row of them B1

Exam-style 3.4 ▪ 5 marks

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.

Exam-style 3.4 (a) ▪ 3 marks

Use a Karnaugh map to obtain the simplified Boolean expression for the output.

Solution 3.4 (a)

Method: A four-variable Karnaugh map — p.14

Worked steps

the four cells are the **corners** of the map, and a K-map **wraps round**, so they form one valid group of four M1

- across all four, $B = 0$ and $D = 0$; A and C each take both values and cancel M1
- $\mathbf{F = \overline{B} \cdot \overline{D}}$ A1
- four separate groups of one, giving four ANDed terms, earns the first mark only: the question is about grouping

Exam-style 3.4 (b) ▪ 2 marks

State **two** benefits of using a Karnaugh map rather than Boolean algebra.

Solution 3.4 (b)

Worked steps

any two: it gives the simplified expression **directly**, without a chain of algebra B1

- 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 B1

Exam-style 3.5 ▪ 3 marks

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

Solution 3.5

Method: A four-variable Karnaugh map — p.14

Worked steps

- $A + \overline{A}B$: apply the **distributive** law, $A + \overline{A}B = (A + \overline{A})(A + B)$ M1
- $A + \overline{A} = 1$ – the **complement** law M1
- $1 \cdot (A + B) = \mathbf{A} + \mathbf{B}$ – the identity law A1
- 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