Grade 12 A/L ICT

Lesson 4: Fundamentals of Digital Circuits

- 4.1 Logic Gates and Truth Tables: NOT, AND, OR, XOR, NAND, NOR, XNOR
- 4.2 Boolean Algebra and Karnaugh Maps: Laws, SOP, POS and Simplification
- 4.3 Designing Digital Circuits: From Everyday Needs to Logic Gates
- 4.4 Half Adder, Full Adder and Flip-Flops: CPU Blocks and Memory Cells

© 2026 schoolict.net (Ruwan Mangala Suraweera)
All rights reserved.

Grade 12 A/L ICT · Lesson 4

4.1 Logic Gates and Truth Tables: NOT, AND, OR, XOR, NAND, NOR, XNOR

Syllabus 4.1. A **logic gate** is a small circuit that takes one or more **binary inputs** (0 or 1) and gives one **binary output**. In this part you learn the seven basic gates – **NOT, AND, OR, XOR** and the combinational gates **NAND, NOR, XNOR** – how to **draw their symbols**, how to write their **truth tables**, and why **NAND and NOR are universal gates** that can build any other gate.

1. What is a logic gate?

Inside every CPU and every digital chip, decisions are made with **logic gates**. Each gate follows a simple rule: for every possible mix of 0s and 1s at the inputs, the output is either 0 or 1. We write that rule as a **truth table**. A **0** means LOW / false / OFF. A **1** means HIGH / true / ON.

> 💡 A truth table lists **every** input combination. With n inputs there are **2^n rows**. One input → 2 rows. Two inputs → 4 rows. Three inputs → 8 rows.

Seven basic logic gates



The seven basic logic gates

2. Basic gates: NOT, AND, OR, XOR

NOT (inverter) has **one** input. It **flips** the bit: F = A′ (also written ¬A or Ā). If A is 0, F is 1; if A is 1, F is 0.

A	F = NOT
0	1
1	0

AND outputs 1 only when **every** input is 1. Write F = A · B (or A ∧ B). Think of it as "A **and** B must both be true".

A	B	F = AND
0	0	0
0	1	0

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

1	0	0
1	1	1

OR outputs 1 when **at least one** input is 1. Write F = A + B (or A ∨ B). "A **or** B (or both)".

A	B	F = OR
0	0	0
0	1	1
1	0	1
1	1	1

XOR (exclusive OR) outputs 1 when the inputs are **different**. Write F = A ⊕ B. Same inputs → 0; different → 1. It is the gate that makes the **sum bit** of a half adder (Lesson 4.4).

A	B	F = XOR
0	0	0
0	1	1
1	0	1
1	1	0

🧱 **Worked example: Complete the truth table for F = A · B. What is F when A = 1 and B = 1?**
AND is 1 only when both inputs are 1, so F = **1**. The four rows are 00→0, 01→0, 10→0, 11→**1**.

3. Combinational gates: NAND, NOR, XNOR

A **combinational gate** is a basic gate with a **NOT** stuck on its output. **NAND** = NOT-AND: F = (A · B)′. It is 0 only when both inputs are 1. **NOR** = NOT-OR: F = (A + B)′. It is 1 only when both inputs are 0. **XNOR** = NOT-XOR: F = (A ⊕ B)′. It is 1 when the inputs are the **same** (equality gate).

A	B	F = NAND
0	0	1
0	1	1
1	0	1
1	1	0

A	B	F = NOR
0	0	1

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

0	1	0
1	0	0
1	1	0

A	B	F = XNOR
0	0	1
0	1	0
1	0	0
1	1	1

💡 A small **circle** on a gate symbol means NOT (inversion). NAND looks like AND with a circle; NOR looks like OR with a circle; XNOR looks like XOR with a circle.

🧱 **Worked example: For inputs A = 1, B = 0 find the outputs of AND, OR, XOR, NAND, NOR and XNOR.**
AND = 0, OR = 1, XOR = 1, NAND = 1, NOR = 0, XNOR = 0.

4. Truth tables for expressions (up to three inputs)

For an expression such as **F = A · B + A′ · C** you make an 8-row table (A, B, C from 000 to 111). Work out each product term, then OR them.

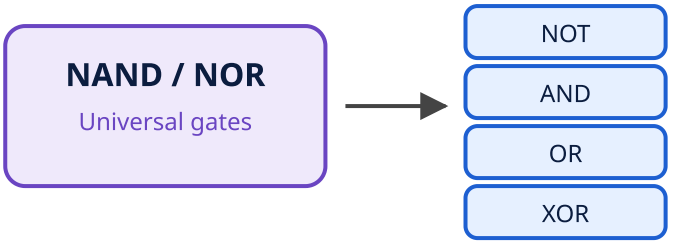
🧱 **Worked example: Build the truth table for F = A · B + A′ · C.**
F is 1 for the rows ABC = 001, 011, 110, 111 (decimal 1, 3, 6, 7). F is 0 for 000, 010, 100, 101.

A	B	C	A·B	A′·C	F
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

5. Universal gates: NAND and NOR

A gate is **universal** if you can build **every** other gate from copies of it alone. **NAND** and **NOR** are universal. That matters in factories: a chip designer can stock one type of gate and still make any circuit.



NAND and NOR can build any other gate

NOT from NAND
Tie both inputs together: A NAND A = A′.

AND from NAND
NAND once, then invert the output with another NAND: (A NAND B)′ = A · B.

OR from NAND
Invert each input, then NAND: A′ NAND B′ = A + B (De Morgan).

🧱 **Worked example: Show that A NAND A equals NOT A, for A = 0 and A = 1.**
A = 0: 0 NAND 0 = 1 = NOT 0. A = 1: 1 NAND 1 = 0 = NOT 1. So NAND with tied inputs is an inverter.

💡 You can also build everything from NOR alone: A NOR A = A′, and so on. In exams, be ready to draw a NOT, AND or OR using only NAND (or only NOR) gates.

🚀 **Beyond the syllabus (2026 update)** *Not required for the exam – read for interest.* In 2026, CPUs still use CMOS transistors that behave as tiny NAND and NOR gates. A modern phone SoC contains tens of billions of such gates. Quantum computers use different “gates” (Hadamard, CNOT) that work on qubits, not bits – but the idea of building big circuits from a small set of universal gates is the same.

Key Words

Logic gate – A circuit with binary inputs that produces one binary output.

Truth table – A table that lists every input combination and the matching output.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

NOT – Inverter: output is the opposite of the single input. F = A′.	**AND** – Output is 1 only if every input is 1. F = A · B.
OR – Output is 1 if at least one input is 1. F = A + B.	**XOR** – Exclusive OR: output is 1 when inputs differ. F = A ⊕ B.
NAND – NOT-AND: F = (A · B)′. A universal gate.	**NOR** – NOT-OR: F = (A + B)′. A universal gate.
XNOR – NOT-XOR: output is 1 when inputs are equal.	**Universal gate** – A gate that can be used alone to build any other gate (NAND, NOR).
Inversion bubble – The small circle on a gate symbol that means NOT.	**Combinational gate** – A gate made by adding NOT to a basic gate (NAND, NOR, XNOR).

Sample Q&A

1. What is a logic gate?

> **Answer:** A small digital circuit that takes one or more binary inputs and produces one binary output according to a fixed rule.

2. Write the truth table for a 2-input AND gate.

> **Answer:** AB: 00 → 0, 01 → 0, 10 → 0, 11 → 1.

3. How does XOR differ from OR?

> **Answer:** OR is 1 when any input is 1 (including both). XOR is 1 only when the inputs are different – both 1 gives 0.

4. Why are NAND and NOR called universal gates?

> **Answer:** Because any other gate (NOT, AND, OR, XOR, …) can be built using only NAND gates, or only NOR gates.

5. How do you make a NOT gate from a NAND gate?

> **Answer:** Connect both NAND inputs to the same signal A. Then A NAND A = A′.

6. How many rows does a truth table for three inputs have?

> **Answer:** 2^3 = 8 rows (000 through 111).

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

MCQ

1. Which gate outputs 1 only when every input is 1?

A. XOR
B. NOR
C. AND
D. OR
E. NOT

2. The output of a NOT gate when the input is 1 is

A. unchanged
B. 2
C. 1
D. undefined
E. 0

3. Which gate is also called the exclusive OR?

A. XOR
B. OR
C. NAND
D. AND
E. XNOR

4. NAND is the same as

A. NOR followed by NOT
B. OR followed by NOT
C. XOR followed by NOT
D. AND followed by NOT
E. NOT followed by AND

5. Which two gates are universal?

A. OR and XOR
B. AND and OR
C. NAND and NOR
D. XOR and XNOR
E. NOT and AND

6. For A = 1, B = 1 the XOR output is

A. B
B. 0
C. 1
D. A
E. 2

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

7. For A = 0, B = 0 the NOR output is

A. 1
B. undefined
C. A
D. B
E. 0

8. A small circle on a gate symbol means

A. AND
B. power supply
C. inversion (NOT)
D. ground
E. clock

9. How many rows are in a truth table with 3 inputs?

A. 9
B. 3
C. 6
D. 4
E. 8

10. XNOR output is 1 when the inputs are

A. both 0 only
B. equal
C. undefined
D. both 1 only
E. different

11. A NAND B equals A′ when

A. B = 1
B. A = 1
C. B = 0
D. A = 0
E. B is tied to A (B = A)

12. Which expression is OR?

A. A ⊕ B
B. (A · B)′
C. A · B
D. A + B
E. A′

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

13. Which term means: “A circuit with binary inputs that produces one binary output”?

A. NOR
B. Inversion bubble
C. XOR
D. Logic gate
E. NOT

14. Which term means: “A table that lists every input combination and the matching output”?

A. Truth table
B. NAND
C. AND
D. XNOR
E. Combinational gate

15. Which term means: “Inverter: output is the opposite of the single input. F = A′”?

A. NOT
B. Logic gate
C. Universal gate
D. OR
E. NOR

16. Which term means: “Output is 1 only if every input is 1. F = A · B”?

A. Truth table
B. Inversion bubble
C. XOR
D. XNOR
E. AND

17. Which term means: “Output is 1 if at least one input is 1. F = A + B”?

A. NAND
B. OR
C. Universal gate
D. Combinational gate
E. NOT

18. Which term means: “Exclusive OR: output is 1 when inputs differ. F = A ⊕ B”?

A. Inversion bubble
B. AND
C. XOR
D. NOR
E. Logic gate

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

19. Which term means: "NOT-AND: F = (A · B)′. A universal gate"?

A. XNOR
B. NAND
C. Truth table
D. OR
E. Combinational gate

20. Which term means: "NOT-OR: F = (A + B)′. A universal gate"?

A. XOR
B. Logic gate
C. Universal gate
D. NOR
E. NOT

Structured Questions

1. Draw the truth table for a 2-input OR gate and state when the output is 0. *(2 marks)*

> Rows: 00 → 0, 01 → 1, 10 → 1, 11 → 1 (1 each); output is 0 only when both inputs are 0 (1).

2. Explain the difference between AND and NAND using their truth tables. *(3 marks)*

> AND: 11 → 1, other rows 0 (1); NAND is the opposite of AND: 11 → 0, other rows 1 (1); NAND = (A·B)′ (1).

3. Why are universal gates useful? Name the two universal gates. *(2 marks)*

> One gate type can build every other gate / simplifies chip design (1); NAND and NOR (1).

4. Show how to obtain NOT and AND using only NAND gates. *(2 marks)*

> NOT: A NAND A = A′ (1); AND: (A NAND B) then NAND the result with itself to invert → A·B (2).

5. For F = A · B + A′ · C, find F when ABC = 011 and when ABC = 101. *(2 marks)*

> 011: A=0,B=1,C=1 → A·B=0, A′·C=1 → F=1 (1); 101: A=1,B=0,C=1 → A·B=0, A′·C=0 → F=0 (1).

Summary

- A logic gate maps binary inputs to one binary output; a truth table lists every case.
- NOT flips; AND needs all 1s; OR needs any 1; XOR needs different inputs.
- NAND, NOR and XNOR are the negated forms of AND, OR and XOR.
- NAND and NOR are universal – each alone can build every other gate.
- With n inputs a truth table has 2^n rows; three inputs → 8 rows.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

MCQ answer key – 4.1

1. **C** – AND is 1 only for the all-1s row.
2. **E** – NOT flips 1 to 0.
3. **A** – XOR = exclusive OR.
4. **D** – NAND = $(A \cdot B)'$.
5. **C** – Only NAND and NOR can build every other gate alone.
6. **B** – Same inputs → XOR = 0.
7. **A** – NOR is 1 only when both inputs are 0.
8. **C** – The bubble is inversion.
9. **E** – $2^3 = 8$.
10. **B** – XNOR is the equality gate.
11. **E** – A NAND A = A'.
12. **D** – + denotes OR in Boolean algebra.
13. **D** – Logic gate – A circuit with binary inputs that produces one binary output.
14. **A** – Truth table – A table that lists every input combination and the matching output.
15. **A** – NOT – Inverter: output is the opposite of the single input. F = A'.
16. **E** – AND – Output is 1 only if every input is 1. F = $A \cdot B$.
17. **B** – OR – Output is 1 if at least one input is 1. F = A + B.
18. **C** – XOR – Exclusive OR: output is 1 when inputs differ. F = $A \oplus B$.
19. **B** – NAND – NOT-AND: F = $(A \cdot B)'$. A universal gate.
20. **D** – NOR – NOT-OR: F = $(A + B)'$. A universal gate.

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.

Grade 12 A/L ICT · Lesson 4

4.2 Boolean Algebra and Karnaugh Maps: Laws, SOP, POS and Simplification

Syllabus 4.2. Digital circuits follow **Boolean algebra** – the maths of 0 and 1. In this part you learn the **postulates and laws** (commutative, associative, distributive, identity, redundancy, De Morgan), how to write expressions in **sum-of-products (SOP)** and **product-of-sums (POS)** form, how to **simplify** with theorems, and how a **Karnaugh map (K-map)** groups 1s so you can read a simpler expression by eye.

1. Why simplify Boolean expressions?

A simpler expression needs **fewer gates**, so the circuit is **cheaper, smaller, cooler and faster**. The exam often asks you to simplify with laws or with a K-map.

2. Postulates and laws

Boolean algebra uses only the values **0 and 1**. The basic operations are **AND (·)**, **OR (+)** and **NOT (′)**. Useful laws:



Boolean algebra laws

Law	AND form	OR form
Commutative	A · B = B · A	A + B = B + A
Associative	(A · B) · C = A · (B · C)	(A + B) + C = A + (B + C)
Distributive	A · (B + C) = A·B + A·C	A + (B · C) = (A+B)·(A+C)
Identity	A · 1 = A	A + 0 = A
Null / Annulment	A · 0 = 0	A + 1 = 1
Idempotent	A · A = A	A + A = A
Complement	A · A′ = 0	A + A′ = 1
Redundancy	A + A·B = A	A · (A + B) = A
De Morgan	(A · B)′ = A′ + B′	(A + B)′ = A′ · B′

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

💡 **De Morgan in words:** break the bar and change the sign – an AND bar becomes OR of the complements; an OR bar becomes AND of the complements.

🧱 **Worked example: Check De Morgan for A = 1, B = 0: is (A · B)′ equal to A′ + B′?**
(1 · 0)′ = 0′ = 1. A′ + B′ = 0 + 1 = 1. Equal – De Morgan holds.

3. Standard forms: SOP and POS

Sum of Products (SOP) is an OR of AND terms, for example A·B + A′·C. Each AND term is a **minterm** when it contains every variable (or its complement) once. **Product of Sums (POS)** is an AND of OR terms, for example (A+B)·(A′+C). Each OR term that has every variable is a **maxterm**.

From a truth table: write one minterm for every row where F = 1, then OR them → **SOP**. Write one maxterm for every row where F = 0, then AND them → **POS**. SOP and POS of the same function are equal.

🧱 **Worked example: F(A,B,C) is 1 for minterms 1, 3, 5, 7. Write the SOP.**
m1=A′B′C, m3=A′BC, m5=AB′C, m7=ABC. SOP = A′B′C + A′BC + AB′C + ABC. This simplifies to **C** (every term has C, and the A/B parts cover all).

4. Simplifying with Boolean theorems

🧱 **Worked example: Simplify F = A · B + A · B′.**
Factor A: F = A · (B + B′) = A · 1 = **A**. (Identity and complement laws.)

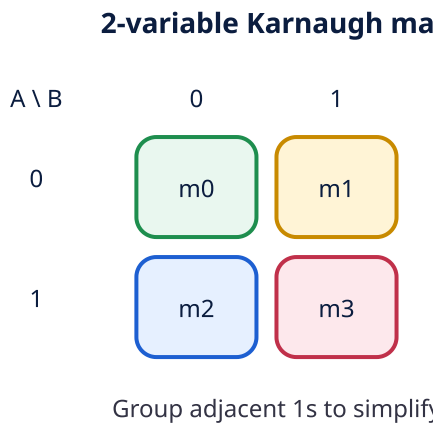
🧱 **Worked example: Simplify F = A + A′ · B.**
Absorption / redundancy: A + A′·B = (A + A′) · (A + B) wait – better: A + A′·B = A·1 + A′·B = A(B+B′) + A′·B … classic result: **A + B**.

🧱 **Worked example: Simplify F = (A + B) · (A + B′).**
Distributive: A + (B · B′) = A + 0 = **A**.

5. Karnaugh maps

A **Karnaugh map** is a truth table redrawn so that **adjacent cells differ by only one bit**. You circle groups of 1, 2, 4 or 8 ones (powers of two). Each group becomes one product term with the variables that stay constant. Larger groups → fewer literals → simpler circuit.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved



A 2-variable Karnaugh map

2-variable K-map: rows A=0,1; columns B=0,1. Cells are minterms m0..m3. **3-variable K-map:** usually rows A, columns BC in Gray code order 00, 01, 11, 10 so that neighbours still differ by one bit. Edges wrap around.

🧱 **Worked example: Simplify F(A,B) = Σm(1,2,3) with a K-map.**
Cells m1,m2,m3 are 1. Group m1+m3 (B=1, A varies) → B. Group m2+m3 (A=1, B varies) → A. Result: **A + B**.

🧱 **Worked example: Simplify F(A,B,C) = Σm(0,1,2,3,6) with a K-map.**
The four cells with A=0 (m0–m3) form a group of 4 → **A′**. Cell m6 (ABC=110) groups with m2 (010) → **B · C′**. Result: **A′ + B·C′**.

💡 Never make a group of 3 or 5. Only 1, 2, 4, 8, … Cover every 1 at least once; overlapping is allowed; don’t circle 0s.

🚀 **Beyond the syllabus (2026 update)** *Not required for the exam – read for interest.* Logic synthesis tools in 2026 (Synopsys, Cadence, open-source Yosys) still use Boolean algebra and Espresso-style minimisation behind the scenes, then map the result onto millions of standard-cell gates. Learning K-maps by hand teaches you what those tools are doing.

Key Words

Boolean algebra – Algebra of the values 0 and 1 with AND, OR and NOT.

Commutative law – Order does not matter: A·B = B·A and A+B = B+A.

Associative law – Grouping does not matter for AND or for OR.

Distributive law – A·(B+C) = A·B+A·C and A+(B·C) = (A+B)·(A+C).

De Morgan’s laws – (A·B)′ = A′+B′ and (A+B)′ = A′·B′.

SOP – Sum of Products – an OR of AND terms.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

POS – Product of Sums – an AND of OR terms.

Minterm – A product term that includes every variable once (true or complemented).

Maxterm – A sum term that includes every variable once.

Karnaugh map – A grid that groups adjacent 1s to simplify a Boolean function.

Literal – A variable or its complement (A or A′).

Redundancy law – A + A·B = A (and the dual A·(A+B) = A).

Sample Q&A

1. State De Morgan's two laws.

> **Answer:** (A · B)′ = A′ + B′ and (A + B)′ = A′ · B′.

2. What is SOP form?

> **Answer:** Sum of Products: the function is written as an OR of AND terms (minterms).

3. Why do we simplify Boolean expressions?

> **Answer:** Fewer gates → smaller, cheaper, cooler and often faster circuits.

4. Simplify A · B + A · B′.

> **Answer:** A · (B + B′) = A · 1 = A.

5. What size groups are allowed on a K-map?

> **Answer:** Powers of two: 1, 2, 4, 8, … (never 3 or 5).

6. How do you get a POS expression from a truth table?

> **Answer:** Write a maxterm for every row where F = 0, then AND those maxterms.

MCQ

1. De Morgan: (A · B)′ equals

A. A′ + B′
B. A + B
C. A · B
D. A′ · B′
E. (A + B)′

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

2. SOP means

A. Signed Octal Product
B. Sum of Products
C. Product of Sums
D. Sum of Powers
E. Single Output Process

3. POS means

A. Product of Sums
B. Product of Squares
C. Sum of Products
D. Positive Output Signal
E. Parallel Output Sum

4. A · 1 equals

A. 0
B. A · A
C. A
D. 1
E. A′

5. A + A′ equals

A. 0
B. 1
C. A
D. A′
E. A · A′

6. Simplify A · B + A · B′

A. B
B. A · B
C. 0
D. A
E. A + B

7. A K-map group may contain how many 1s?

A. 1, 2, 4 or 8
B. Any odd number
C. Only 4
D. 3 or 5
E. Only 2

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

8. Adjacent K-map cells differ by

A. all bits
B. three bits
C. two bits
D. no bits
E. exactly one bit

9. A + A · B simplifies to

A. A
B. B
C. 1
D. A · B
E. A + B

10. F = Σm(1,2,3) for two variables simplifies to

A. A′
B. A + B
C. B′
D. A · B
E. 1

11. (A + B)′ equals

A. A′
B. A + B
C. A′ + B′
D. A′ · B′
E. A · B

12. A minterm for three variables has how many literals?

A. 1
B. 8
C. 4
D. 2
E. 3

13. Which term means: “Algebra of the values 0 and 1 with AND, OR and NOT”?

A. Minterm
B. Associative law
C. Literal
D. SOP
E. Boolean algebra

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

14. Which term means: “Order does not matter: A·B = B·A and A+B = B+A”?

A. Maxterm
B. Commutative law
C. Redundancy law
D. Distributive law
E. POS

15. Which term means: “Grouping does not matter for AND or for OR”?

A. Boolean algebra
B. De Morgan’s laws
C. Minterm
D. Associative law
E. Karnaugh map

16. Which term means: “A·(B+C) = A·B+A·C and A+(B·C) = (A+B)·(A+C)”?

A. Commutative law
B. Maxterm
C. Distributive law
D. Literal
E. SOP

17. Which term means: “(A·B)′ = A′+B′ and (A+B)′ = A′·B′”?

A. POS
B. Redundancy law
C. Associative law
D. De Morgan’s laws
E. Karnaugh map

18. Which term means: “Sum of Products – an OR of AND terms”?

A. Distributive law
B. Minterm
C. SOP
D. Literal
E. Boolean algebra

19. Which term means: “Product of Sums – an AND of OR terms”?

A. Maxterm
B. Commutative law
C. POS
D. De Morgan’s laws
E. Redundancy law

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

20. Which term means: "A product term that includes every variable once (true or complemented)"?

A. Karnaugh map
B. Boolean algebra
C. Associative law
D. SOP
E. Minterm

Structured Questions

1. State the commutative, associative and distributive laws for AND and OR. *(3 marks)*

> Commutative: A·B=B·A, A+B=B+A (1); Associative: (A·B)·C=A·(B·C), same for + (1); Distributive: A·(B+C)=A·B+A·C and A+(B·C)=(A+B)·(A+C) (1).

2. Explain SOP and POS and how each is obtained from a truth table. *(3 marks)*

> SOP = OR of minterms for F=1 rows (1); POS = AND of maxterms for F=0 rows (1); both represent the same function (1).

3. Simplify F = A + A′ · B using Boolean laws. Show the steps. *(2 marks)*

> A + A′·B = (A + A′)(A + B) is one route, or note redundancy expansion → A + B (2); final answer A + B (1).

4. Using a 2-variable K-map, simplify F = Σm(1,2,3). *(3 marks)*

> Place 1s in m1,m2,m3 (1); group → A and B (1); F = A + B (1).

5. Apply De Morgan to simplify (A · B′)′. *(2 marks)*

> (A · B′)′ = A′ + (B′)′ (1); (B′)′ = B so A′ + B (1).

Summary

- Boolean laws (commutative, associative, distributive, De Morgan, identity, redundancy) let you rewrite expressions.
- SOP is an OR of products; POS is an AND of sums – both come from the truth table.
- Simplifying cuts the number of gates, so circuits are cheaper and faster.
- A K-map groups adjacent 1s in sizes 1, 2, 4, 8 to read a minimal SOP.
- De Morgan: break the bar and change the operator.

MCQ answer key – 4.2

1. **A** – Break the bar, change · to +.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

2. **B** – OR of AND terms.
3. **A** – AND of OR terms.
4. **C** – Identity law.
5. **B** – Complement law.
6. **D** – A(B+B′)=A.
7. **A** – Powers of two only.
8. **E** – Gray-code adjacency.
9. **A** – Redundancy law.
10. **B** – K-map groups give A + B.
11. **D** – De Morgan for OR.
12. **E** – One literal per variable.
13. **E** – Boolean algebra – Algebra of the values 0 and 1 with AND, OR and NOT.
14. **B** – Commutative law – Order does not matter: A·B = B·A and A+B = B+A.
15. **D** – Associative law – Grouping does not matter for AND or for OR.
16. **C** – Distributive law – A·(B+C) = A·B+A·C and A+(B·C) = (A+B)·(A+C).
17. **D** – De Morgan's laws – (A·B)′ = A′+B′ and (A+B)′ = A′·B′.
18. **C** – SOP – Sum of Products – an OR of AND terms.
19. **C** – POS – Product of Sums – an AND of OR terms.
20. **E** – Minterm – A product term that includes every variable once (true or complemented).

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.

Grade 12 A/L ICT · Lesson 4

4.3 Designing Digital Circuits: From Everyday Needs to Logic Gates

Syllabus 4.3. Now you **design** small circuits yourself. You pick a real situation (a door alarm, a majority vote, a car seatbelt light), write its **truth table** (up to three inputs), turn the table into a **Boolean expression**, simplify it, and finally **draw the gate circuit**. This is how engineers start every digital design.

1. The design steps

Four clear steps take you from a need to a working circuit:



Need → truth table → expression → circuit

1. Identify the need
Name the inputs and the one output in everyday words.

2. Truth table
List every input mix (2^n rows) and the required output.

3. Expression
Write SOP (or POS), then simplify with laws or a K-map.

4. Draw the circuit
Replace each · with AND, each + with OR, each ′ with NOT.

2. Example A – Door alarm at night

A house alarm should sound when the **door is open (D = 1)** AND it is **night (N = 1)**. By day, or when the door is shut, the alarm stays off.

🧱 **Worked example: Write the truth table and the Boolean expression for the door alarm.**
DN: 00→0, 01→0, 10→0, 11→1. So F = **D · N**. The circuit is a single **AND** gate.

D (door)	N (night)	Alarm F
0	0	0
0	1	0
1	0	0

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

1	1	1

3. Example B – Majority of three votes

Three people A, B, C vote yes (1) or no (0). The decision F is yes when **at least two** vote yes – the **majority** function.

> 🧱 **Worked example: Write F for the majority of A, B, C and list when F = 1.**
> F = A·B + B·C + A·C. F = 1 for ABC = 011, 101, 110, 111 (the rows with two or three 1s).

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

> 💡 The majority circuit uses **three AND gates** (one per pair) and **one OR gate** that joins their outputs. It is also the carry logic inside a full adder (4.4).

4. Example C – Seatbelt warning light

A car warning light turns on when the **engine is on (E = 1)** AND the **seatbelt is not fastened (S = 0)**. So F = E · S′.

> 🧱 **Worked example: Draw the truth table for F = E · S′ and name the gates you need.**
> ES: 00→0, 01→0, 10→1, 11→0. Gates: one **NOT** on S, then one **AND** with E.

5. Turning an expression into a circuit

Reading an expression left to right: every ′ becomes a NOT bubble (or a NOT gate), every · becomes an AND gate, every **+** becomes an OR gate. Draw inputs on the left, output on the right. Share NOT gates when the same inverted signal is used twice.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

🧱 **Worked example: For F = A·B + A′·C, list the gates required after simplifying is not needed.**
One NOT (for A′), two AND gates (A·B and A′·C), one OR gate joining them – **4 gates** total.

💡 Always simplify before you draw. F = A·B + A·B′ looks like two ANDs and an OR (3 gates), but it is just A (0 extra gates beyond a wire).

🚀 **Beyond the syllabus (2026 update)** *Not required for the exam – read for interest.* Real products combine these tiny designs into HDL code (Verilog / VHDL). In 2026, students and makers also use low-cost FPGA boards (e.g. Lattice iCE40, Xilinx / AMD Spartan) to download a truth table and watch the LEDs follow the circuit you designed on paper.

Key Words

Digital circuit design – Turning a real need into a truth table, expression and gate circuit.

Input variable – A binary signal that feeds a circuit (door, night, vote, ...).

Output function – The single binary result the circuit must produce.

Majority function – Output is 1 when at least two of three inputs are 1. F = AB+BC+AC.

Alarm circuit – Example: F = D · N (door open AND night).

Gate-level circuit – A drawing that shows each AND, OR and NOT gate.

Simplification before drawing – Reduce the expression so the circuit uses fewer gates.

Three-input design – Syllabus limit: truth tables with at most three inputs (8 rows).

Seatbelt warning – F = E · S′ – engine on and belt not fastened.

Shared inverter – One NOT gate reused when the same inverted signal is needed twice.

Application – A real-life situation that a logic circuit can control.

Boolean to gates – · → AND, + → OR, ′ → NOT.

Sample Q&A

1. List the four steps of digital circuit design.

Answer: Need → truth table → Boolean expression (simplified) → draw gates.

2. What is the Boolean expression for a door-open-at-night alarm?

Answer: F = D · N.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

3. Write the majority function of three inputs.

> **Answer:** F = A·B + B·C + A·C.

4. How many rows does a 3-input design truth table have?

> **Answer:** 8 rows.

5. Why simplify before drawing the circuit?

> **Answer:** A simpler expression needs fewer gates, so the circuit is cheaper and smaller.

6. For F = E · S′, which gates do you need?

> **Answer:** One NOT on S and one AND with E.

MCQ

1. The first step in designing a digital circuit is

A. write POS only
B. identify the need and name inputs/output
C. buy an FPGA
D. draw random gates
E. skip the truth table

2. Door alarm F = D · N uses which gate?

A. XOR
B. NOR
C. AND
D. NOT
E. OR

3. Majority of A,B,C equals

A. A + B + C
B. A′B′C′
C. A ⊕ B ⊕ C
D. AB + BC + AC
E. A · B · C

4. A 3-input truth table has

A. 4 rows
B. 8 rows
C. 6 rows
D. 3 rows
E. 9 rows

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

5. F = E · S′ needs

A. XOR only
B. OR only
C. NOR only
D. NAND only
E. NOT and AND

6. · in a Boolean expression becomes which gate?

A. OR
B. NOT
C. NOR
D. XOR
E. AND

7. + in a Boolean expression becomes which gate?

A. OR
B. NOT
C. XOR
D. AND
E. NAND

8. For the alarm, F is 1 only when

A. always
B. door shut
C. door open and night
D. never
E. daytime

9. The majority circuit is also used as

A. carry logic of a full adder
B. a clock
C. an antenna
D. a NOT gate
E. a memory cell

10. Simplifying F = A·B + A·B′ before drawing saves

A. the truth table
B. gates (it becomes just A)
C. nothing
D. inputs
E. De Morgan

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

11. Syllabus designs are limited to how many inputs?

A. three
B. five
C. ten
D. eight
E. one

12. Shared inverters are used when

A. you have spare OR gates
B. there is no truth table
C. POS is forbidden
D. F = 1 always
E. the same inverted signal is needed more than once

13. Which term means: “Turning a real need into a truth table, expression and gate circuit”?

A. Output function
B. Application
C. Three-input design
D. Digital circuit design
E. Gate-level circuit

14. Which term means: “A binary signal that feeds a circuit (door, night, vote, …)”?

A. Seatbelt warning
B. Majority function
C. Boolean to gates
D. Input variable
E. Simplification before drawing

15. Which term means: “The single binary result the circuit must produce”?

A. Three-input design
B. Shared inverter
C. Output function
D. Digital circuit design
E. Alarm circuit

16. Which term means: “Output is 1 when at least two of three inputs are 1. F = AB+BC+AC”?

A. Majority function
B. Application
C. Gate-level circuit
D. Seatbelt warning
E. Input variable

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

17. Which term means: “Example: F = D · N (door open AND night)”?

A. Shared inverter
B. Output function
C. Alarm circuit
D. Boolean to gates
E. Simplification before drawing

18. Which term means: “A drawing that shows each AND, OR and NOT gate”?

A. Majority function
B. Three-input design
C. Application
D. Gate-level circuit
E. Digital circuit design

19. Which term means: “Reduce the expression so the circuit uses fewer gates”?

A. Seatbelt warning
B. Simplification before drawing
C. Alarm circuit
D. Input variable
E. Boolean to gates

20. Which term means: “Syllabus limit: truth tables with at most three inputs (8 rows)”?

A. Digital circuit design
B. Gate-level circuit
C. Shared inverter
D. Output function
E. Three-input design

Structured Questions

1. Describe the four design steps with a short example. *(4 marks)*

> Need (e.g. night door alarm) (1); truth table (1); expression F=D·N (1); draw AND gate (1).

2. Build the truth table and expression for a light that is ON when either switch A or switch B is ON (or both). *(3 marks)*

> AB: 00→0, 01→1, 10→1, 11→1 (1); F = A + B (1); circuit = OR gate (1).

3. For the majority function, state F and say for which of ABC = 101 and 100 the output is 1. *(3 marks)*

> F = AB+BC+AC (1); 101 → 1 (two 1s) (1); 100 → 0 (only one 1) (1).

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

4. Design F = E · S′ (seatbelt warning): truth table, expression already given, gates needed. *(2 marks)*

> Table ES: 00→0, 01→0, 10→1, 11→0 (2); gates NOT + AND (1).

5. Explain why you should simplify A·B + A·B′ before drawing, and state the simplified form. *(3 marks)*

> Unsimplified needs 2 AND + 1 OR (1); simplified is A – just a wire (1); saves gates/cost (1).

Summary

- Design flow: need → truth table → simplify expression → draw gates.
- Door alarm: F = D · N (one AND).
- Majority: F = AB + BC + AC (also full-adder carry).
- Seatbelt warning: F = E · S′ (NOT then AND).
- Syllabus designs use at most three inputs (8-row truth tables).

MCQ answer key – 4.3

1. **B** – Start from the real-world need.
2. **C** – AND of door and night.
3. **D** – At least two inputs are 1.
4. **B** – $2^3 = 8$.
5. **E** – Invert S, then AND with E.
6. **E** – · denotes AND.
7. **A** – + denotes OR.
8. **C** – Both D and N must be 1.
9. **A** – Cout = majority(A,B,Cin).
10. **B** – A·B + A·B′ = A.
11. **A** – Up to three inputs.
12. **E** – Reuse one NOT.
13. **D** – Digital circuit design – Turning a real need into a truth table, expression and gate circuit.
14. **D** – Input variable – A binary signal that feeds a circuit (door, night, vote, …).
15. **C** – Output function – The single binary result the circuit must produce.
16. **A** – Majority function – Output is 1 when at least two of three inputs are 1. F = AB+BC+AC.
17. **C** – Alarm circuit – Example: F = D · N (door open AND night).
18. **D** – Gate-level circuit – A drawing that shows each AND, OR and NOT gate.
19. **B** – Simplification before drawing – Reduce the expression so the circuit uses fewer gates.
20. **E** – Three-input design – Syllabus limit: truth tables with at most three inputs (8 rows).

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.

Grade 12 A/L ICT · Lesson 4

4.4 Half Adder, Full Adder and Flip-Flops: CPU Blocks and Memory Cells

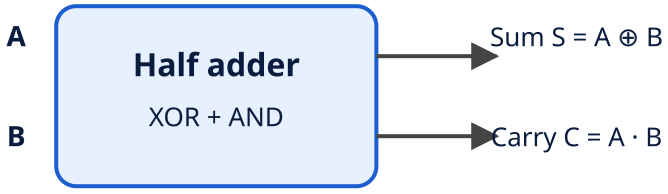
Syllabus 4.4. The CPU adds numbers with **adder** circuits, and memory stores bits with **flip-flops**. In this part you build the truth tables and Boolean expressions for a **half adder** and a **full adder**, see how full adders chain into an ALU, and learn how a **feedback loop** in a **flip-flop** remembers one bit – the building block of registers and RAM.

1. Building blocks of the CPU

Arithmetic in the ALU starts with adding two bits. A **half adder** adds two bits and makes a sum and a carry. A **full adder** also takes a **carry-in** from the previous column, so you can chain eight full adders to add two 8-bit numbers.

2. Half adder

Inputs: A, B. Outputs: **Sum S** and **Carry C**. Looking at binary addition: 0+0=00, 0+1=01, 1+0=01, 1+1=10. So S is 1 when the inputs differ → **S = A ⊕ B**. C is 1 only when both are 1 → **C = A · B**.



Half adder: XOR for Sum, AND for Carry

A	B	Sum S	Carry C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

🧱 **Worked example: For A = 1, B = 1 find S and C of a half adder.**
S = 1 ⊕ 1 = **0**. C = 1 · 1 = **1**. (Binary 1 + 1 = 10_2.)

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

💡 A half adder has **no Cin**, so it can only add the least-significant bit of a multi-bit number. Every higher bit needs a full adder.

3. Full adder

Inputs: A, B, **Cin**. Outputs: **Sum S** and **Cout**. S is 1 when an **odd** number of the three inputs is 1 → **S = A ⊕ B ⊕ Cin**. Cout is 1 when **at least two** inputs are 1 → **Cout = A·B + B·Cin + A·Cin** (the majority function from 4.3).



S = A ⊕ B ⊕ Cin Cout = majority(A,B,Cin)

Full adder with Cin and Cout

A	B	Cin	Sum S	Cout
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

🧱 **Worked example: For A = 1, B = 0, Cin = 1 find S and Cout.**
S = 1 ⊕ 0 ⊕ 1 = **0**. Cout = majority = **1**. (1+0+1 = 10_2.)

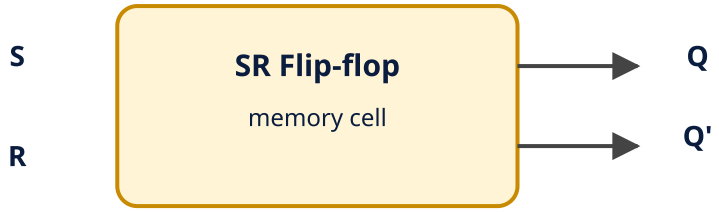
🧱 **Worked example: For A = 1, B = 1, Cin = 1 find S and Cout.**
S = 1 ⊕ 1 ⊕ 1 = **1**. Cout = **1**. (1+1+1 = 11_2.)

You can build a full adder from **two half adders plus an OR**: first HA adds A and B → S1, C1; second HA adds S1 and Cin → S, C2; then Cout = C1 + C2.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

4. Storing bits – feedback and flip-flops

A combinational circuit's output depends **only on the current inputs**. To **remember** a bit you need a **feedback loop**: feed the output back to an input so the circuit can hold its state. A **flip-flop** is the basic memory cell that stores **one bit**.



Feedback loop stores one bit

SR flip-flop stores one bit with feedback

The simple **SR (Set-Reset) flip-flop** has inputs S and R and outputs Q and Q′:

- S = 1, R = 0 → **Set**: Q becomes 1.
- S = 0, R = 1 → **Reset**: Q becomes 0.
- S = 0, R = 0 → **Hold**: Q stays as it was (memory!).
- S = 1, R = 1 → **Forbidden** in a basic SR latch (both outputs fight).

💡 Registers, cache and static RAM are banks of flip-flops. Dynamic RAM uses tiny capacitors instead, but the idea – store one bit per cell – is the same.

🧱 **Worked example: A flip-flop currently holds Q = 0. What are S and R to (a) set it to 1, (b) keep it at 0?**
(a) S = 1, R = 0 (set). (b) S = 0, R = 0 (hold) – or actively reset with S = 0, R = 1.

🚀 **Beyond the syllabus (2026 update)** *Not required for the exam – read for interest.* In 2026, a phone CPU still uses billions of CMOS full-adder cells in its ALU and SIMD units, and SRAM caches built from 6-transistor flip-flop-like cells. Emerging MRAM and ReRAM try to replace some DRAM, but the syllabus idea stays: adders compute, flip-flops remember.

Key Words

Half adder – Adds two bits: S = A ⊕ B, C = A · B. No Cin.

Full adder – Adds A, B and Cin: S = A⊕B⊕Cin, Cout = majority.

Sum bit – The result bit of an adder column (XOR of the inputs).

Carry – The extra bit that goes to the next column when the sum ≥ 2.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

Cin / Cout – Carry in from the previous column; carry out to the next.

Flip-flop – A memory cell that stores one bit using feedback.

Feedback loop – Wiring an output back to an input so the circuit can hold state.

SR flip-flop – Set-Reset latch: S sets Q=1, R resets Q=0, 00 holds.

Set – Make Q = 1 (S = 1, R = 0).

Reset – Make Q = 0 (S = 0, R = 1).

Hold / memory – S = R = 0 keeps the previous Q.

Register – A group of flip-flops storing a multi-bit word in the CPU.

Sample Q&A

1. Write the Boolean expressions for a half adder.

> **Answer:** S = A ⊕ B, C = A · B.

2. Why does a multi-bit adder need full adders, not only half adders?

> **Answer:** Higher bit columns must accept a carry-in from the previous column; a half adder has no Cin.

3. Write S and Cout for a full adder.

> **Answer:** S = A ⊕ B ⊕ Cin; Cout = A·B + B·Cin + A·Cin.

4. What does a flip-flop store?

> **Answer:** One bit, held by a feedback loop.

5. In an SR flip-flop, what do S = 1, R = 0 and S = R = 0 do?

> **Answer:** S=1,R=0 sets Q to 1; S=R=0 holds the previous value.

6. How can you build a full adder from half adders?

> **Answer:** Two half adders plus an OR gate for the two carry outputs.

MCQ

1. Half-adder Sum equals

A. A ⊕ B ⊕ Cin
B. A · B
C. A ⊕ B
D. A + B
E. A′

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

2. Half-adder Carry equals

A. A · B
B. A + B
C. A ⊕ B ⊕ Cin
D. A′ · B′
E. A ⊕ B

3. A full adder has how many inputs?

A. 1
B. 8
C. 4
D. 2
E. 3 (A, B, Cin)

4. Full-adder Sum equals

A. A · B
B. A ⊕ B ⊕ Cin
C. A ⊕ B
D. A + B
E. majority

5. Full-adder Cout equals

A. A + B + Cin only as OR of three
B. A ⊕ B ⊕ Cin
C. majority of A, B, Cin
D. A′
E. A ⊕ B

6. For A=1,B=1 on a half adder, S and C are

A. S=1, C=1
B. S=0, C=0
C. S=1, C=0
D. S=0, C=1
E. S=1, C=undefined

7. A flip-flop stores

A. no data
B. an analog voltage only
C. one word only
D. one bit
E. one byte

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

8. Feedback in a flip-flop is used to

A. add two numbers
B. hold / remember the state
C. invert a bit only
D. create Cin
E. delete memory

9. SR flip-flop: S=1, R=0 means

A. Hold
B. Toggle randomly
C. Forbidden
D. Reset Q = 0
E. Set Q = 1

10. SR flip-flop: S=R=0 means

A. Hold previous Q
B. Reset
C. Set
D. Q becomes 1 always
E. Forbidden

11. Which is forbidden in a basic SR latch?

A. S = 1, R = 0
B. S = 1 and R = 1
C. S = 0, R = 1
D. Q = 1
E. S = 0, R = 0

12. An 8-bit adder needs how many full adders (roughly)?

A. 1
B. 16
C. 8
D. 4
E. 2

13. Which term means: "Adds two bits: S = A ⊕ B, C = A · B. No Cin"?

A. Sum bit
B. Hold / memory
C. Flip-flop
D. SR flip-flop
E. Half adder

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

14. Which term means: “Adds A, B and Cin: S = A⊕B⊕Cin, Cout = majority”?

A. Full adder
B. Register
C. Set
D. Feedback loop
E. Carry

15. Which term means: “The result bit of an adder column (XOR of the inputs)”?

A. Half adder
B. Reset
C. SR flip-flop
D. Sum bit
E. Cin / Cout

16. Which term means: “The extra bit that goes to the next column when the sum ≥ 2”?

A. Full adder
B. Flip-flop
C. Hold / memory
D. Carry
E. Set

17. Which term means: “Carry in from the previous column; carry out to the next”?

A. Feedback loop
B. Register
C. Sum bit
D. Reset
E. Cin / Cout

18. Which term means: “A memory cell that stores one bit using feedback”?

A. SR flip-flop
B. Flip-flop
C. Hold / memory
D. Half adder
E. Carry

19. Which term means: “Wiring an output back to an input so the circuit can hold state”?

A. Register
B. Full adder
C. Feedback loop
D. Set
E. Cin / Cout

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

20. Which term means: “Set-Reset latch: S sets Q=1, R resets Q=0, 00 holds”?

A. SR flip-flop
B. Half adder
C. Reset
D. Sum bit
E. Flip-flop

Structured Questions

1. Draw the truth table of a half adder and write S and C as Boolean expressions. *(2 marks)*

> Table AB→SC: 00→00, 01→10, 10→10, 11→01 (2); S=A⊕B, C=A·B (2).

2. Explain why a full adder is needed for bits above the LSB. *(3 marks)*

> Those columns receive a carry from the right (1); half adder has no Cin (1); full adder accepts Cin and produces Cout (1).

3. For A=1, B=1, Cin=0 find S and Cout of a full adder. Show working. *(3 marks)*

> S=1⊕1⊕0=0 (1); Cout=AB+BCin+ACin=1+0+0=1 (1); check $1+1+0=10_2$ (1).

4. Describe how a flip-flop stores a bit and what S=R=0 means in an SR flip-flop. *(3 marks)*

> Feedback loop holds state (1); stores one bit (1); S=R=0 is hold / memory of previous Q (1).

5. State the four SR input combinations and their effects. *(4 marks)*

> 10 set Q=1 (1); 01 reset Q=0 (1); 00 hold (1); 11 forbidden (1).

Summary

- Half adder: S = A ⊕ B, C = A · B (no Cin).
- Full adder: S = A ⊕ B ⊕ Cin, Cout = majority(A,B,Cin).
- Chain full adders to add multi-bit numbers in the ALU.
- A flip-flop uses feedback to store one bit.
- SR: 10 set, 01 reset, 00 hold, 11 forbidden.

MCQ answer key – 4.4

1. **C** – XOR of the two bits.
2. **A** – AND of the two bits.
3. **E** – A, B and carry-in.
4. **B** – XOR of all three.
5. **C** – AB + BCin + ACin.
6. **D** – $1+1 = 10_2$.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

7. **D** – Basic memory cell = 1 bit.
8. **B** – Output fed back keeps Q.
9. **E** – Set input active.
10. **A** – Memory / hold state.
11. **B** – Both set and reset at once.
12. **C** – One per bit column.
13. **E** – Half adder – Adds two bits: S = A ⊕ B, C = A · B. No Cin.
14. **A** – Full adder – Adds A, B and Cin: S = A⊕B⊕Cin, Cout = majority.
15. **D** – Sum bit – The result bit of an adder column (XOR of the inputs).
16. **D** – Carry – The extra bit that goes to the next column when the sum ≥ 2.
17. **E** – Cin / Cout – Carry in from the previous column; carry out to the next.
18. **B** – Flip-flop – A memory cell that stores one bit using feedback.
19. **C** – Feedback loop – Wiring an output back to an input so the circuit can hold state.
20. **A** – SR flip-flop – Set-Reset latch: S sets Q=1, R resets Q=0, 00 holds.

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.