Chapter 3: Digital Systems and Logic Design

LONG Question from the Chapter "Digital Systems and Logic Design":

EXERCISE LONG Question with Answers

---

1. Explain the usage of Boolean functions in computers.

Answer (concise + clear):

- A **Boolean function** maps Boolean input variables (0/1 or FALSE/TRUE) to a Boolean output. In computers, Boolean functions are used wherever decisions or binary choices are needed.
- **Hardware**: Logic gates implement Boolean functions physically — processors, ALUs, control units, memory address decoders and multiplexers all use Boolean logic.
- **Control & decision making**: Example — a condition `if (A AND B) then ...` corresponds to the Boolean function `A·B`.
- **Arithmetic circuits**: Adders and comparators implement Boolean functions (sum and carry are Boolean expressions of input bits).
- **Software & algorithms**: Conditional statements, bit masks, and logical operators in programming languages are Boolean expressions used for flow control and checks.
- **Summary**: Boolean functions form the foundation of digital design; they convert logical specifications into realizable gate circuits and software conditions.

---

2. Describe how to construct a truth table for a Boolean expression with an example.

Steps (method):

1. **List variables**: Identify the input variables (for example A, B, C).
2. **Rows**: Create a table with 2n2^n rows (n = number of variables) that lists all possible combinations of inputs in binary order (e.g., 000, 001, 010,...).
3. **Evaluate sub-expressions** (optional): If the expression is complex, compute intermediate parts column-by-column.
4. **Compute final output** for each row using the Boolean operators.
5. **Write result** in the output column.

Example: Construct truth table for $F(A, B) = A \cdot \overline{B} + \overline{A} \cdot B$ (which is XOR).

A	B	$\overline{B}$	$A \cdot \overline{B}$	$\overline{A}$	$\overline{A} \cdot B$	F
0	0	1	0	1	0	0
0	1	0	0	1	1	1
1	0	1	1	0	0	1
1	1	0	0	0	0	0

So $F = 1$ only when A and B differ — this is XOR.

---

3. Describe the concept of duality in Boolean algebra and provide an example to illustrate it.

Principle (short):
The **principle of duality** states: *If you take any Boolean identity/expression and swap every AND ($\cdot$) with OR ($+$) and every 0 with 1 (and vice versa), the result is also a valid Boolean identity.*
Complement and variables remain unchanged.

Example:

- Start with identity: A·0=0A \cdot 0 = 0.
- Dual: replace $\cdot \rightarrow +$, and $0 \rightarrow 1$: A+1=1A + 1 = 1.
  Both are true identities.

Another example:

- Identity: A+A=AA + A = A. Dual is A·A=AA \cdot A = A — both true.

Use: Duality helps generate new valid identities and is often used to check work when simplifying expressions.

---

4. Compare and contrast half-adders and full-adders, including their truth tables, Boolean expressions, and circuit diagrams.

Half-Adder

- **Purpose:** Adds two single bits (A and B) and produces **Sum** and **Carry**. There is no input carry.
- **Inputs:** A, B
- **Outputs:** Sum (S), Carry (C)

Truth table:

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

Boolean expressions:

- $S=A \oplus B=AB^{-}+A^{-}BS$ = A \oplus B = A\overline{B} + \overline{A}B
- $C=A \cdot BC$ = A\cdot B

Circuit: XOR gate produces Sum; AND gate produces Carry. (Diagram: A and B to XOR → S; A and B to AND → C.)

Full-Adder

- **Purpose:** Adds three bits — two operands (A, B) and a carry-in (Cin). Produces **Sum** and **Carry-out (Cout)**. Used in cascaded adders.
- **Inputs:** A, B, Cin
- **Outputs:** Sum (S), Cout

Truth table (8 rows):

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

A	B	Cin	Sum (S)	Cout
1	1	0	0	1
1	1	1	1	1

Boolean expressions:

- S=A⊕B⊕CinS = A \oplus B \oplus Cin
  (equivalently S=(A⊕B)⊕CinS = (A\oplus B)\oplus Cin)
- Cout=(A·B)+(Cin·(A⊕B))Cout = (A\cdot B) + (Cin\cdot(A\oplus B))
  (equivalently Cout=A·B+A·Cin+B·CinCout = A\cdot B + A\cdot Cin + B\cdot Cin)

Circuit (construction):

- A typical implementation uses **two half-adders and an OR gate**:
  1. First half-adder: inputs A, B → produces S1 and C1 (S1 = A⊕B, C1 = A·B).
  2. Second half-adder: inputs S1, Cin → produces Sum S and C2 (S = S1⊕Cin, C2 = S1·Cin).
  3. Cout = C1 + C2 (OR gate).
- This modular design makes it easy to cascade full adders for multi-bit addition.

Comparison summary:

- Half-adder: 2 inputs, no input carry; Full-adder: 3 inputs including carry-in.
- Full-adder handles carry propagation, so it is used in multi-bit adders (ripple-carry adders).

---

5. How do Karnaugh maps simplify Boolean expressions? Provide a detailed example with steps.

Idea (short):

- **Karnaugh map (K-map)** is a visual method for minimizing Boolean expressions by grouping adjacent 1s (or 0s) in a grid that represents all minterms. Groups must be powers of two (1,2,4,8...). Grouping yields simplified product terms by eliminating variables that change within the group.

Example: Simplify F(A,B,C)=Σm(1,3,4,5,7)F(A,B,C) = \Sigma m(1,3,4,5,7). That is, F = 1 for minterms 1,3,4,5,7.

Steps:

1. **Draw 3-variable K-map** (rows and columns in Gray code):

```
        BC
      00  01  11  10
A=0   m0  m1  m3  m2
A=1   m4  m5  m7  m6
```

2. **Mark 1s** at minterms: m1, m3, m4, m5, m7

K-map with 1s:

- (A=0,BC=01) m1 = 1
- (A=0,BC=11) m3 = 1
- (A=1,BC=00) m4 = 1
- (A=1,BC=01) m5 = 1
- (A=1,BC=11) m7 = 1

3. **Group adjacent 1s (largest groups of size power of two):**

- Group1: m3 (A0,BC11) and m7 (A1,BC11) → vertical pair; they differ only in A → group simplifies to BC = 11 → term = $B \cdot C$.
- Group2: m4 (A1,BC00) and m5 (A1,BC01) → horizontal pair; they differ in C only → group simplifies to A·$\overline{B}$ (since BC are 00 & 01 → B=0 constant, A=1 constant) → term = $A \cdot \overline{B}$.
- Group3: m1 (A0,BC01) and m5 (A1,BC01) → vertical pair; they differ only in A → simplifies to $\overline{B} \cdot C$? Wait: BC=01 so B=0, C=1 → term = $\overline{B} \cdot C$. But note m5 already used; overlapping groups OK to get largest simplification.

We can often cover all 1s with groups: m1+m5 pair gives $\overline{B} \cdot C$. Combined with previous groups, cover all minterms.

4. **Write simplified expression:** Combine group terms:

$$F = B \cdot C + A \cdot \overline{B} + \overline{B} \cdot C.$$

5. **Further simplification:** Factor $\overline{B}$ from last two terms:

$$F = BC + \overline{B}(A + C).$$

If desired, check if any term is redundant; often further simplification possible, but this is already simpler than sum of five minterms.

Result: K-map produces a minimized expression faster than algebraic manipulation.

---

6. Design a 4-bit binary adder using both half-adders and full-adders. Explain each step with truth tables, Boolean expressions, and circuit diagrams.

Design approach: Use **ripple-carry adder** with four full-adders chained; the least-significant bit (LSB) can be implemented with a **half-adder** if there is no input carry, but usually we use full adders for uniformity (Cin for first bit may be 0).

Steps:

1. **Bit positions:** Let inputs be A3 A2 A1 A0 and B3 B2 B1 B0 (A0 and B0 are LSB). Let Cin0 be initial carry-in (often 0). Outputs are Sum3..Sum0 and final carry Cout4.
2. **LSB stage (bit 0):**
   - If no external carry-in, you can use a **half-adder**:
     Sum0 = A0 ⊕ B0
     Carry0 = A0·B0
   - If a carry-in exists, use a full-adder: Sum0 = A0 ⊕ B0 ⊕ Cin0; Carry0 = (A0·B0) + (Cin0·(A0⊕B0)).
3. **Middle stages (bit 1..2):** Use full-adder for each bit i:
   - Sum_i = A_i ⊕ B_i ⊕ Carry_{i}
   - Carry_{i+1} = (A_i·B_i) + (Carry_i·(A_i⊕B_i))
4. **MSB (bit 3):** Full-adder same formula produces Sum3 and final carry Cout4.

Circuit diagram (description):

- Four full adders stacked left-to-right (or right-to-left). Carry output of stage i connects to carry input of stage i+1 (ripple). Sum outputs taken from each stage. Optionally, the first stage uses a half-adder if Cin0 = 0.

Truth tables & expressions:

- Each full-adder internally uses the half-adder expressions shown in Q4. The truth table for a full-adder was provided earlier (8 rows). Each stage uses that table for A_i, B_i, Cin to produce Sum_i and Cout_{i+1}.

Example numeric check: Add A = 0101 (5) and B = 0011 (3):

- Stage0: 1+1 = Sum0 0, Carry0 1
- Stage1: 0+1 + Carry0(1) => Sum1 = 0, Carry1 =1
- Stage2: 1+0 + Carry1 => Sum2 =0, Carry2 =1
- Stage3: 0+0 + Carry2 => Sum3 =1, final carry = 0
  Result = 1000 (8) — correct (5+3=8).

Note on performance: Ripple-carry adder is simple but slow for many bits because carries propagate sequentially. For high-performance adders, carry-lookahead or other faster schemes are used.

---

7. Simplify the following Boolean function using Boolean algebra rules:

(interpreting printed expression as)

$$F(A,B) = A \cdot B \; + \; A \cdot \overline{B}$$

Simplification (step-by-step):

1. Factor out A:

$$F = A \cdot (B + \overline{B})$$

2. Use complement law: $B + \overline{B} = 1$.
3. So $F = A \cdot 1 = A$.

Answer: $F(A,B) = A$.

(Thus the function outputs A irrespective of B.)

---

8. Use De Morgan's laws to simplify the following function:

Given (interpreted): $\overline{A + B + AC}$ — i.e. complement of $(A + B + AC)$. I will apply De Morgan to the complement.

Using De Morgan's laws:

$$\overline{X + Y} = \overline{X} \cdot \overline{Y}, \qquad \overline{X \cdot Y} = \overline{X} + \overline{Y}.$$

Apply to $\overline{A + B + AC}$:

$$\overline{A + B + AC} = \overline{A} \cdot \overline{B} \cdot \overline{AC}.$$

Now $\overline{AC} = \overline{A} + \overline{C}$ (by De Morgan).
So:

$$\overline{A + B + AC} = \overline{A} \cdot \overline{B} \cdot (\overline{A} + \overline{C}).$$

Distribute $\overline{A}$:

$$= \overline{A} \cdot \overline{B} \cdot \overline{A} \; + \; \overline{A} \cdot \overline{B} \cdot \overline{C}.$$

But $\overline{A} \cdot \overline{A} = \overline{A}$, so:

$$= \overline{A} \cdot \overline{B} \; + \; \overline{A} \cdot \overline{B} \cdot \overline{C}.$$

Factor $\overline{A} \cdot \overline{B}$:

$$= \overline{A} \cdot \overline{B} \cdot (1 + \overline{C}) = \overline{A} \cdot \overline{B}.$$

(Using $1 + X = 1$.)

Final simplified form: $\boxed{\overline{A} \cdot \overline{B}}$.

---

9. Simplify the following expressions

(a) $\overline{A} + \overline{B} \cdot (A + B)$

Steps:

1. Expand $\overline{B} \cdot (A + B) = \overline{B} \cdot A + \overline{B} \cdot B$.
2. But $\overline{B} \cdot B = 0$. So that part is $\overline{B} \cdot A$.
3. Now expression becomes $\overline{A} + (A \cdot \overline{B})$.
4. Use consensus/covering: $\overline{A} + A \cdot \overline{B} = \overline{A} + \overline{B} \cdot A$. We can apply $X + \overline{X}Y = X + Y$ with $X = \overline{A}$? Better to check another identity:
   Use $X + \overline{X}Y = X + Y$ with $X = \overline{A}$ gives: $\overline{A} + A \cdot \overline{B} = \overline{A} + \overline{B}$.
   Wait that substitution is tricky. Instead observe:
5. Using distributive: $\overline{A} + A \cdot \overline{B} = (\overline{A} + A) \cdot (\overline{A} + \overline{B}) = 1 \cdot (\overline{A} + \overline{B}) = \overline{A} + \overline{B}$.
   (We used $X + YZ = (X + Y)(X + Z)$.)

Simplified result: $\boxed{\overline{A} + \overline{B}}$.

(b) $(A + \overline{B}) \cdot (\overline{A} + B)$

This is a common expression; it equals $A \oplus B$ (XOR) or simplifies to $A\overline{B} + \overline{A}B$.

Proof (expand):

$$(A + \overline{B})(\overline{A} + B) = A\overline{A} + AB + \overline{B}\overline{A} + \overline{B}B.$$

But $A\overline{A} = 0$ and $\overline{B}B = 0$. So

$$= AB + \overline{A}\overline{B}? \text{ Wait check terms: } \overline{B}\overline{A} = \overline{A}\overline{B}.$$

So we have $AB + \overline{A}\overline{B}$. That is the XNOR function (equal when bits equal). But earlier I thought XOR — check carefully.

Let's compute with truth table quickly:

- A=0,B=0: (0+1)*(1+0)*=1*1=1
- A=0,B=1: (0+0)*(1+1)*=0*1=0
- A=1,B=0: (1+1)*(0+0)*=1*0=0
- A=1,B=1: (1+0)*(0+1)*=1*1=1

So expression is 1 when A=B; it's **XNOR**: $\overline{A \oplus B}$ or $AB + \overline{A}\,\overline{B}$.

Simplified result: $\boxed{A \cdot B + \overline{A} \cdot \overline{B}}$ (XNOR).

(c) $A + \overline{A} \cdot (\overline{B} + C)$

Steps:

1. Distribute: $A + \overline{A}B + \overline{A}C$.
2. Group $A + \overline{A}B = (A + \overline{A})(A + \overline{B}) = 1 \cdot (A + \overline{B}) = A + \overline{B}$.
   So expression becomes $A + \overline{B} + \overline{A}C$.
3. Now $A + \overline{A}C = A + C$? Use identity $X + \overline{X}Y = X + Y$ with $X = A, Y = C$: yes, $A + \overline{A}C = A + C$.
4. So full becomes $(A + C) + \overline{B}$ = $A + C + \overline{B}$.

You can also reorder: final simplified form is $\boxed{A + \overline{B} + C}$ (since $A + C + \overline{B}$ same).

(d) $\overline{A} \cdot B + A \cdot \overline{B}$

This is the XOR function.

Simplified result: $\boxed{A \oplus B}$ (i.e., 1 when A ≠ B).

(e) $(A \cdot B) + (\overline{A} \cdot \overline{B})$

This is the XNOR function (1 when A = B).

Result: $\boxed{\overline{A \oplus B}}$ or $A \cdot B + \overline{A} \cdot \overline{B}$.

Additional MCQs with Answers

---

1. Analyze the significance of digital logic in modern electronics, focusing on its applications in computers, smartphones, and adder circuits.
2. Discuss the role of Boolean functions in digital systems, including their applications in arithmetic operations, data processing, and control logic.
3. Explain the difference between analog and digital signals, and discuss the importance of ADC and DAC in modern technology with examples.
4. Explain the construction and operation of half-adder and full-adder circuits, including their truth tables, Boolean expressions, and circuit diagrams.
5. How do Karnaugh maps simplify Boolean expressions? Provide a detailed example with steps.
6. Describe how Karnaugh maps are used to simplify Boolean expressions, with a step-by-step example for a three-variable function.
7. Explain the usage of Boolean functions in computers.
8. Simplify the following Boolean function using Boolean algebra rules: $F(A, B) = A \cdot B + A \cdot \bar{B}$
9. Discuss the process of simplifying Boolean expressions using Boolean algebra rules, providing at least three examples of simplification.
10. Use De Morgan's laws to simplify the following function: $F(A, B, C) = \overline{A + B} + \overline{AC}$
11. Describe the primary logic gates (AND, OR, NOT, NAND, XOR) and their functions, including their truth tables and real-world applications.
12. Compare and contrast half-adders and full-adders, including their truth tables, Boolean expressions, and circuit diagrams.
13. Explain the structure and purpose of logic diagrams in representing digital circuits, and provide an example of creating one for a Boolean function.
14. Simplify the following expressions: (a) $\overline{A + B} \cdot (A + B)$ (b) $(A + \bar{B}) \cdot (\bar{A} + B)$ (c) $A + \bar{A} \cdot (\bar{B} + C)$ (d) $\overline{A \cdot B} + A \cdot B$ (e) $(A \cdot B) + (\bar{A} \cdot \bar{B})$
15. Design a 4-bit binary adder using both half-adders and full-adders. Explain each step with truth tables, Boolean expressions, and circuit diagrams.
16. Describe the concept of duality in Boolean algebra and provide an example to illustrate it.
17. Describe how to construct a truth table for a Boolean expression with an example.
