Laboratory Design Assignment

4-bit Arithmetic Logic Unit (ALU) Design

Learning Outcomes:

This assignment will help students:

- Understand **combinational logic design** for an ALU.
- Implement **arithmetic, logical, and shift operations**.
- Develop skills in **hardware description languages (HDL)**.
- Learn **flag management** for overflow, negative, and carry conditions.

Objective: Design, simulate, and implement a **4-bit ALU** capable of performing a range of arithmetic, logical, and shift operations, with proper flag management for overflow, carry, and sign detection. You may extend the designs in Week 5 lecture and Week 6 tutorial to achieve this.

Problem Statement:

You are required to design a **4-bit ALU** that supports various arithmetic, logical, shift, and comparison operations. Your ALU should handle **both signed and unsigned arithmetic** and properly implement **status flags** for error detection. The design should be **optimized for minimal gate usage** while ensuring correctness and efficiency. You must implement a hierarchical design by structuring your circuit into modular, reusable subcomponents. Create clear and well-labelled schematic diagrams to illustrate your design at both the module and system levels. Simulate your circuit using appropriate software, verify its functionality.

Specifications:

1. Inputs:

- **Operand A (4-bit)**
- **Operand B (4-bit)**
- **Opcode (4-bit) – Selects operation**
- **Carry-in (1-bit) – Used for addition with carry**

2. Operations (Based on Opcode):

Opcode	Operation	Description
0000	Addition	A + B (unsigned)
0001	Subtraction	A - B (unsigned)
0010	Signed Addition	A + B (signed)
0011	Signed Subtraction	A - B (signed)
0100	AND	A AND B
0101	OR	A OR B
0110	XOR	$A \oplus B$
0111	XNOR	$A \odot B$
1000	Left Shift	A << 1
1001	Right Shift	A >> 1 (Logical)
1010	Arithmetic Right Shift	A >> 1 (Preserves sign bit)
1011	Rotate Left	Circular left shift (A)
1100	Rotate Right	Circular right shift (A)
1101	Compare	Outputs 1 if A > B, else 0
1110	Bitwise NOT	A'
1111	Transfer	A

3. Outputs:

- **Result (4-bit)** – Computed output
- **Flags (1-bit each):**
    - **Zero Flag (Z)** – Set if the result is 0
    - **Carry Flag (C)** – Set if an arithmetic carry/borrow occurs
    - **Overflow Flag (V)** – Set if a signed overflow occurs
    - **Negative Flag (N)** – Set if the result is negative

Task Breakdown:

1. Design Phase:

- Develop a **truth table and logic expressions** for all operations.
- Implement **multiplexers** to select the correct operation based on the opcode.
- Use **carry look-ahead logic** for faster addition/subtraction.
- Design flag logic for **Zero, Carry, Overflow, and Negative flags**.

2. Implementation Phase:

- Use **Verilog/VHDL** or **Logisim** for simulation and testing.
- Validate functionality using multiple test cases.

3. Report Submission:

- **Introduction** – Overview of the ALU design.
- **ALU Architecture** – Block diagram, equations, and logic implementation.
- **Simulation Results** – Test cases and waveforms.
- **Challenges & Solutions** – Insights from the design process.
- **Conclusion & Future Improvements**.

Bonus Challenges (For Extra 10 marks):

- Implement a **Multiply (A × B) operation** (Modulo 16 result). (6 marks)
- Implement an **8-bit ALU extension** (cascade two 4-bit ALUs). (4 marks)

Rubric

Criteria	Excellent (High Distinction, 85-100%)	Good (Distinction, 70-84%)	Satisfactory (Credit, 55-69%)	Needs Improvement (Fail, <55%)	Marks
Problem Understanding & Approach (20%)	Clearly understands ALU design, well-defined problem statement, structured approach.	Good understanding with minor gaps.	Basic understanding but lacks depth.	Misunderstood problem or vague approach.	/20
ALU Architecture & Design (30%)	Optimized design, well-structured block diagram, minimal gate usage, efficient logic.	Mostly correct, minor inefficiencies or missing minor components.	Functional but lacks optimization or clarity.	Poorly structured, missing key components.	/30
Implementation & Correctness (30%)	Fully functional ALU, all operations work correctly, efficient design.	Mostly functional, minor errors in some operations.	Some operations incorrect or inefficient.	Major errors, incomplete implementation.	/30
Flag Management (20%)	Correct implementation of all flags (Zero, Carry, Overflow, Negative).	Mostly correct, minor flag handling issues.	Some flags missing or incorrectly implemented.	No or incorrect flag implementation.	/20
Bonus (Optional, Extra marks)	Implemented extra complexity (e.g., 8-bit extension, multiplication).	Implemented a minor additional feature.	No additional feature, but complete ALU.	Incomplete ALU, missing core operations.	+ up to 10