**Design Specification Document**

**Project Name:** Arithmetic Logic Unit (ALU) Design

**Version:** 1.0

**Date:** [03/19/2025]

**Author:** [XYZ]

---

## 1. Introduction

The Arithmetic Logic Unit (ALU) is a combinational circuit responsible for performing arithmetic and logical operations. This design implements a 32-bit ALU supporting five operations: addition, subtraction, bitwise AND, bitwise OR, and bitwise XOR.

## 2. Design Overview

The ALU is designed as a Verilog module with configurable data width and command width parameters. The operation to be performed is determined by a 3-bit command input. The module operates in a purely combinational manner, producing outputs based on the provided inputs without clock dependencies.

## 3. Functional Requirements

The ALU must support the following operations:

- **Addition (`ALU_ADD` - 3'b000):** Computes the sum of two operands.
- **Subtraction (`ALU_SUB` - 3'b001):** Computes the difference between two operands.
- **Bitwise XOR (`ALU_XOR` - 3'b010):** Performs a bitwise XOR operation.
- **Bitwise AND (`ALU_AND` - 3'b011):** Performs a bitwise AND operation.
- **Bitwise OR (`ALU_OR` - 3'b100):** Performs a bitwise OR operation.

## 4. Interface Specifications

### 4.1 Parameters

- **`data_width`** (Default: 32) - Defines the width of the operand inputs and output.
- **`cmd_width`** (Default: 3) - Defines the width of the ALU command input.

### 4.2 Inputs

| Signal Name | Width | Description |
|---|---|---|
| `alu_cmd` | `cmd_width` | Specifies the operation to be performed. |
| `alu_operand0` | `data_width` | First operand input. |
| `alu_operand1` | `data_width` | Second operand input. |

### 4.3 Outputs

| Signal Name | Width | Description |
|---|---|---|
| `alu_out` | `data_width` | Output result of the ALU operation. |

## 5. Functional Description

The ALU implements a case statement inside an `always @(*)` block to evaluate operations based on the `alu_cmd` input. The five operations are defined using `define` macros for readability and ease of modification.

## 6. Test Plan

A testbench will be created to verify the functionality of the ALU. The testbench will perform the following:

1. Apply five random pairs of inputs for each ALU operation.
2. Verify that the output matches expected results for each test case.
3. Check edge cases such as all zeroes, all ones, and carry/borrow conditions.

## 7. Assumptions and Limitations

- The ALU does not handle overflow or carry-out detection.
- The ALU does not support multiplication or division.
- The design assumes all inputs are valid; no error handling is implemented.

## 8. Conclusion

This ALU design provides a simple yet functional arithmetic and logic processing unit, suitable for embedded applications and digital design projects. The implementation ensures ease of verification and flexibility through parameterization.

---

**End of Document**