4-bit ALU in Verilog - Project Report

1. Introduction

An Arithmetic Logic Unit (ALU) is the heart of a Central Processing Unit (CPU). It performs fundamental arithmetic and logical operations that drive the execution of instructions in digital systems. This project implements a 4-bit ALU using Verilog HDL. The aim is to simulate real-world processor functionality through basic yet essential operations.

This report outlines the design, development, simulation, and verification of the ALU, which supports 8 distinct operations, simulates behavior using ModelSim, and can be synthesized using Intel Quartus Prime Lite.

---

2. Objectives

- Design a compact 4-bit ALU using Verilog HDL.
- Implement 8 fundamental arithmetic and logic operations.
- Simulate the functionality and observe waveforms using ModelSim.
- Understand synthesis, simulation, and testbench construction.

---

3. Tools Used

✅ Intel Quartus Prime Lite Edition

- Used for Verilog code development.
- Compilation and synthesis of hardware modules.
- Preparing design for FPGA (if needed).

✅ ModelSim – Intel FPGA Edition

- Industry-standard simulator for HDL designs.
- Used to verify logic through waveform and console simulation.

---

4. Design Overview

◇ ALU Inputs and Outputs

- **Inputs:**
    - `A[3:0]`: First 4-bit operand

  - `B[3:0]`: Second 4-bit operand
  - `sel[2:0]`: Operation selector (3-bit)
- **Outputs:**
  - `result[3:0]`: 4-bit output result of the operation
  - `carry_out`: Carry flag, indicates overflow for addition
  - `zero`: Status flag, high if result is zero

Functional Table

sel	Operation	Expression	Notes
000	ADD	A + B	Sets carry_out
001	SUB	A – B	No borrow detection
010	AND	A & B	Bitwise
011	OR	A | B	Bitwise
100	XOR	A ^ B	Bitwise
101	NOT A	~A	Ignores B
110	INCREMENT	A + 1	Carry may trigger
111	DECREMENT	A – 1	No underflow detection

---

5. Verilog Code Snippets

ALU Logic (`alu.v`):

```
module ALU (
    input [3:0] A,
    input [3:0] B,
    input [2:0] sel,
    output reg [3:0] result,
    output reg carry_out,
    output reg zero
);

always @(*) begin
    carry_out = 0;
    case (sel)
        3'b000: {carry_out, result} = A + B;
        3'b001: result = A - B;
        3'b010: result = A & B;
        3'b011: result = A | B;
        3'b100: result = A ^ B;
        3'b101: result = ~A;
        3'b110: result = A + 1;
        3'b111: result = A - 1;
        default: result = 4'b0000;
    endcase
    zero = (result == 4'b0000) ? 1 : 0;
end

endmodule
```

Testbench (`alu_tb.v`):

```
`timescale 1ns / 1ps

module ALU_TB;

reg [3:0] A, B;
reg [2:0] sel;
wire [3:0] result;
wire carry_out, zero;

ALU uut (
    .A(A),
    .B(B),
    .sel(sel),
    .result(result),
    .carry_out(carry_out),
    .zero(zero)
);

initial begin
    A = 4'b0101;
    B = 4'b0011;

    $display("TIME\tSEL\tRESULT\tCARRY\tZERO");

    for (sel = 3'b000; sel <= 3'b111; sel = sel + 1) begin
        #50;
        $display("%0t\t%b\t%b\t%b\t%b", $time, sel, result, carry_out, zero);
    end

    $stop;
end

endmodule
```

---

6. Simulation Results

The simulation was conducted using ModelSim. Below are the observed results for inputs A = 0101 (5) and B = 0011 (3):

sel	Operation	Output	carry_out	zero
000	ADD	1000	0	0
001	SUB	0010	0	0
010	AND	0001	0	0
011	OR	0111	0	0
100	XOR	0110	0	0
101	NOT A	1010	0	0
110	INCREMENT	0110	0	0
111	DECREMENT	0100	0	0

Waveform

Waveform confirmed transitions as per the select signal. Each operation executed with correct timing and output transitions.

---

7. Directory Structure

```
4bit_ALU_Project/
├── src/
│   ├── alu.v            # ALU module
│   └── alu_tb.v         # Testbench
├── sim/
│   └── output.txt       # Transcript or simulation output
├── doc/
│   ├── README.md        # GitHub Readme
│   ├── report.pdf       # This report
│   └── block_diagram.png # Architecture diagram
│
```

---

8. Conclusion

This project successfully demonstrates a compact and functioning 4-bit ALU implemented in Verilog. Simulation via ModelSim verified correct logical operations, status flags, and output behavior. It reinforces understanding of testbench creation, ALU architecture, and waveform interpretation.

---

9. Future Enhancements

- Upgrade to 8-bit or 16-bit ALU for more complex operations.
- Add multiplication, division, shift left/right, rotate.
- Support for signed numbers and overflow detection.

- Interface with control unit for complete datapath simulation.

---

10. References

- Intel Quartus Prime Documentation
- ModelSim User Guide
- Digital Design Textbooks
- GitHub: archita-2005

Author: Archita Roy
Profile: github.com/archita-2005