4-Bit ALU Design Project

Project Overview

This project involves designing a 4-bit Arithmetic Logic Unit (ALU) in Verilog. The ALU can perform various operations such as addition, subtraction, AND, OR, XOR, and NOT. It's a key building block in digital system design and demonstrates knowledge in combinational logic, modular Verilog design, and simulation.

ALU Operations

- 000: Addition
- 001: Subtraction
- 010: Bitwise AND
- 011: Bitwise OR
- 100: Bitwise XOR
- 101: Bitwise NOT (A only)
- 110: Increment A
- 111: Decrement A

Block Diagram

Inputs: A[3:0], B[3:0], Sel[2:0]

Output: Result[3:0], Carry/Overflow (optional)

Verilog Code Snippet

```
module ALU_4bit(input [3:0] A, B,
                input [2:0] Sel,
                output reg [3:0] Result);

always @(*) begin
   case(Sel)
      3'b000: 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
end
endmodule
```

Testbench Sample

```
module test_ALU;
reg [3:0] A, B;
reg [2:0] Sel;
wire [3:0] Result;

ALU_4bit uut(.A(A), .B(B), .Sel(Sel), .Result(Result));

initial begin
    A = 4'b0101; B = 4'b0011; Sel = 3'b000; // A + B
    #10 Sel = 3'b001; // A - B
    #10 Sel = 3'b010; // A & B
end
endmodule
```

Simulation Output

Run the design using tools like ModelSim, Xilinx ISE, or online platforms like EDAPlayground. Observe the waveform to verify each operation.

GitHub Upload Tips

- Create folders: /src (Verilog code), /testbench, /screenshots (simulation)

4-Bit ALU Design Project

- Write a README.md file with description, tools used, and simulation snapshots.

Future Enhancements

- Add 8-bit operation support
- Implement pipelining
- Add overflow/carry flag support
- GUI-based user input with FPGA or simulation interface