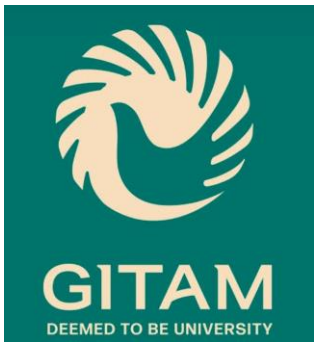


REPORT

Verilog & VHDL code for Arithmetic Logic Unit (ALU)

B. NAGA ROHITHA
2024108382
MTECH (VLSI)

CONTENTS:

INTRODUCTION:

The Main objective of this project is to design and verify the different operations of Arithmetic and logical unit. Here we have designed an 8-bit ALU and the code responding to the operations which it has to perform from the user. It is responsible for executing arithmetic operations like addition and subtraction, multiplication, division as well as logical operations like AND, OR, and XOR, NOR, NOT. Arithmetic Logic Unit (ALU) is one of the central components of digital computing systems and acts as the backbone of processing units like CPUs, microcontrollers, and digital signal processors (DSPs). It is a combinational circuit that performs a wide range of arithmetic and logical operations on binary data, which are essential for executing instructions in a digital system. The ALU performs the desired operations and generated the results accordingly. This report focuses on the design and implementation of an ALU that supports 14 operations using Verilog and VHDL, two widely used hardware description languages. The ALU's design emphasizes flexibility, modularity, and ease of integration into larger digital systems. By supporting a variety of arithmetic, logical, and comparison operations.

OBJECTIVE:

The primary objective of this project is to design and implement a arithmetic Logic Unit (ALU) capable of performing a wide variety of arithmetic and logical operations essential for digital computing systems. This includes both fundamental operations like addition and subtraction and more advanced operations like multiplication, division, and bitwise manipulations. By providing support for 14 operations, the ALU is designed to be versatile and adaptable for a range of applications in digital systems, such as embedded processors, microcontrollers, and FPGAs.

The specific goals of the project include:

1. **Functional:** Develop an ALU that supports 14 unique operations, encompassing arithmetic, logical, and comparison tasks.
2. **Behavioral Modeling**: Use behavioral modeling to describe the ALU's functionality for clarity and ease of understanding.
3. **Modularity**: The design is modular, allowing for easy integration into larger digital systems and simplifying future expansions or modifications.
4. **Scalability**: Design the ALU to be easily adaptable for different word sizes (e.g., 16-bit, 32-bit).
5. **Flag Integration**: Include essential flags, such as Zero and Carry, to provide additional control and monitoring capabilities for specific operations.
6. **Hardware Efficiency**: Optimize the ALU for synthesis on hardware platforms like FPGAs or ASICs, ensuring efficient resource utilization and performance.
7. **Testing and Verification**: Create comprehensive testbenches to verify the correctness and reliability of the ALU across a wide input scenario.

   FEATURES OF THE ALU

   - **Arithmetic Operations**:
   - Addition
   - Subtraction
   - Multiplication
   - Division
   - Modulus

- **Logical Operations**:
    - Bitwise AND
    - Bitwise OR
    - Bitwise XOR
    - Bitwise NOT
    - Bitwise NOR
- **Shift Operations**:
    - Logical Left Shift
    - Logical Right Shift
- **Comparison Operations**:
    - Less Than
    - Equality

DESIGN APPROCH FOR ALU

Behavioral Modeling

Behavioral modeling is used to implement the ALU. This approach employs high-level constructs like the case statement in Verilog and case in VHDL to define the functionality of the ALU. Behavioral modeling simplifies the design process by focusing on the desired operation rather than the low-level gate implementation.

Architecture of the ALU

1. **Entity Declaration**:
    - **Inputs**:
        - A: 4-bit input operand.
        - B: 4-bit input operand.
        - ALUCtrl: 3-bit control signal to select the desired operation.

- **Outputs**:
    - Result: 4-bit result of the operation.
    - Zero: 1-bit flag indicating whether the Result is zero.

Purpose: The entity defines the interface of the ALU.

2. **Architecture Behavior**:
    - **Signal/Process Block**:
        - A process monitors changes in the inputs (A, B, ALUCtrl) and computes the Result and Zero flag accordingly.
        - The behavior is implemented using a **case statement**, which selects the operation based on the value of ALUCtrl.
3. **Operations Supported**:
    - ALUCtrl = 000: AND operation (Result = A AND B)
    - ALUCtrl = 001: OR operation (Result = A OR B)
    - ALUCtrl = 010: ADD operation (Result = A + B)
    - ALUCtrl = 011: XOR operation (Result = A XOR B)
    - ALUCtrl = 100: SUBTRACT operation (Result = A - B)
    - ALUCtrl = 101: NAND operation (Result = NOT (A AND B))
    - ALUCtrl = 110: NOR operation (Result = NOT (A OR B))
    - ALUCtrl = 111: XNOR operation (Result = NOT (A XOR B))
4. **Zero Flag Computation**:
    - After the result is computed, the Zero flag is updated.
    - If Result = 0000, the Zero flag is set to 1. Otherwise, it is set to 0.
5. **Default Behavior**:
    - If ALUCtrl does not match any valid operation (unexpected control input), Result defaults to 0000.

VERILOG CODE:

```
module ALU (
    input [3:0] A,        // 4-bit input A
    input [3:0] B,        // 4-bit input B
    input [2:0] ALUCtrl,  // 3-bit control signal for selecting operation
    output reg [3:0] Result,  // 4-bit result output
    output reg Zero          // Zero flag
);
    always @(*) begin
        case (ALUCtrl)
            3'b000: Result = A & B;   // AND operation
            3'b001: Result = A | B;   // OR operation
            3'b010: Result = A + B;   // ADD operation
            3'b011: Result = A ^ B;   // XOR operation
            3'b100: Result = A - B;   // SUBTRACT operation
            3'b101: Result = ~(A & B); // NAND operation
            3'b110: Result = ~(A | B); // NOR operation
            3'b111: Result = ~(A ^ B); // XNOR operation
            default: Result = 4'b0000; // Default case
        endcase
        // Set Zero flag (1 if Result is zero, 0 otherwise)
        Zero = (Result == 4'b0000) ? 1'b1 : 1'b0;
    end
endmodule
```

TEST BENCH:

```
module ALU_tb;
    // Testbench signals
```

```
reg [3:0] A;
reg [3:0] B;
reg [2:0] ALUCtrl;
wire [3:0] Result;
wire Zero;
// Instantiate the ALU module
ALU uut (
    .A(A),
    .B(B),
    .ALUCtrl(ALUCtrl),
    .Result(Result),
    .Zero(Zero)
);
// Stimulus process
initial begin
    // Test case 1: AND operation
    A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b000; // A & B
    #10;
    // Test case 2: OR operation
    A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b001; // A | B
    #10;
    // Test case 3: ADD operation
    A = 4'b0101; B = 4'b0011; ALUCtrl = 3'b010; // A + B
    #10;

    // Test case 4: XOR operation
    A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b011; // A ^ B
    #10;
```

```
        // Test case 5: SUB operation
        A = 4'b0101; B = 4'b0011; ALUCtrl = 3'b100; // A - B
        #10;
        // Test case 6: NAND operation
        A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b101; // ~(A & B)
        #10;
        // Test case 7: NOR operation
        A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b110; // ~(A | B)
        #10;
        // Test case 8: XNOR operation
        A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b111; // ~(A ^ B)
        #10;
        // Test case 9: Check Zero flag (Result = 0)
        A = 4'b0000; B = 4'b0000; ALUCtrl = 3'b010; // A + B = 0
        #10;
        $finish;  // End the simulation
    end
    // Monitor outputs
    initial begin
        $monitor("Time=%0t, A=%b, B=%b, ALUCtrl=%b, Result=%b, Zero=%b",
                 $time, A, B, ALUCtrl, Result, Zero);
    end
endmodule
```

VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ALU is
    Port (
        A       : in  STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input A
        B       : in  STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input B
        ALUCtrl : in  STD_LOGIC_VECTOR(2 downto 0); -- 3-bit control signal
        Result  : out STD_LOGIC_VECTOR(3 downto 0); -- 4-bit result output
        Zero    : out STD_LOGIC                 -- Zero flag
    );
end ALU;
architecture Behavioral of ALU is
begin
    process (A, B, ALUCtrl)
    begin
        case ALUCtrl is
            when "000" => Result <= A and B;        -- AND operation
            when "001" => Result <= A or B;         -- OR operation
            when "010" => Result <= A + B;          -- ADD operation
            when "011" => Result <= A xor B;        -- XOR operation
            when "100" => Result <= A - B;          -- SUBTRACT operation
            when "101" => Result <= not (A and B);  -- NAND operation
            when "110" => Result <= not (A or B);   -- NOR operation
            when "111" => Result <= not (A xor B);  -- XNOR operation
            when others => Result <= "0000";        -- Default case
        end case;
        -- Set Zero flag (1 if Result is zero, 0 otherwise)
        if Result = "0000" then
```

```
            Zero <= '1';
        else
            Zero <= '0';
        end if;
    end process;
end Behavioral;
```

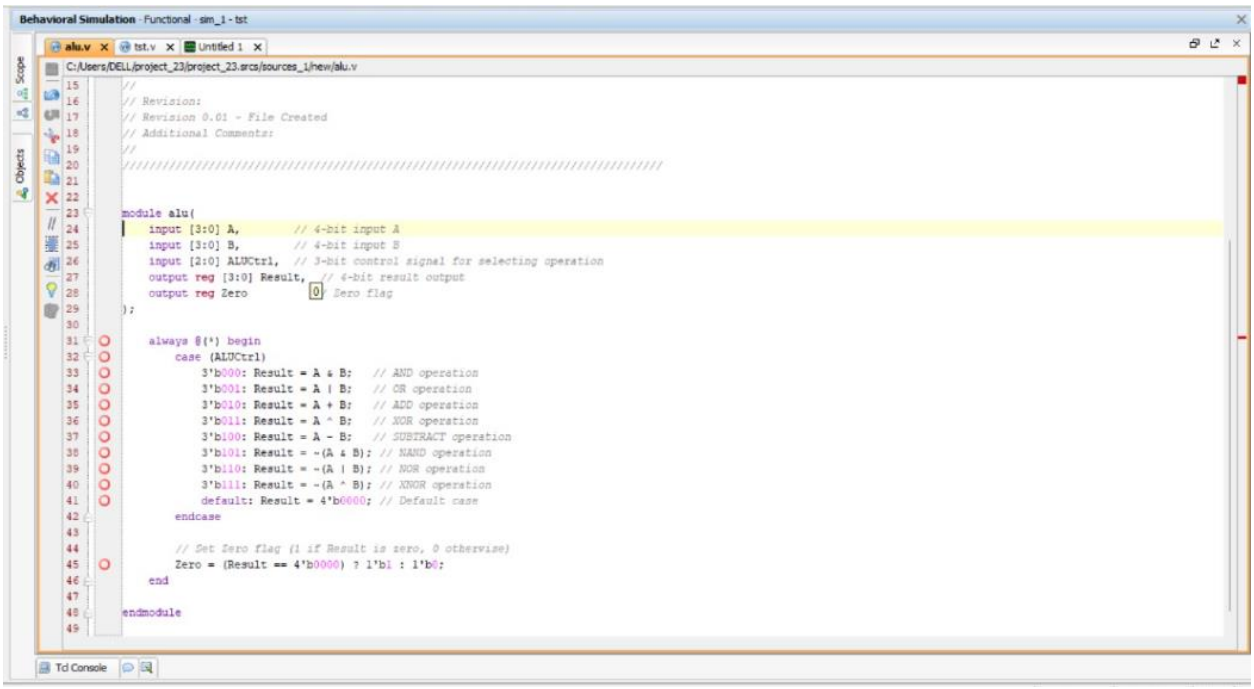
TEST BENCH:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ALU_tb is
    -- No ports for testbench
end ALU_tb;
architecture Behavioral of ALU_tb is
    -- Signals for ALU inputs and outputs
    signal A       : STD_LOGIC_VECTOR(3 downto 0);
    signal B       : STD_LOGIC_VECTOR(3 downto 0);
    signal ALUCtrl : STD_LOGIC_VECTOR(2 downto 0);
    signal Result  : STD_LOGIC_VECTOR(3 downto 0);
    signal Zero    : STD_LOGIC;

begin
    -- Instantiate the ALU
    uut: entity work.ALU
        port map (
            A      => A,
            B      => B,
```

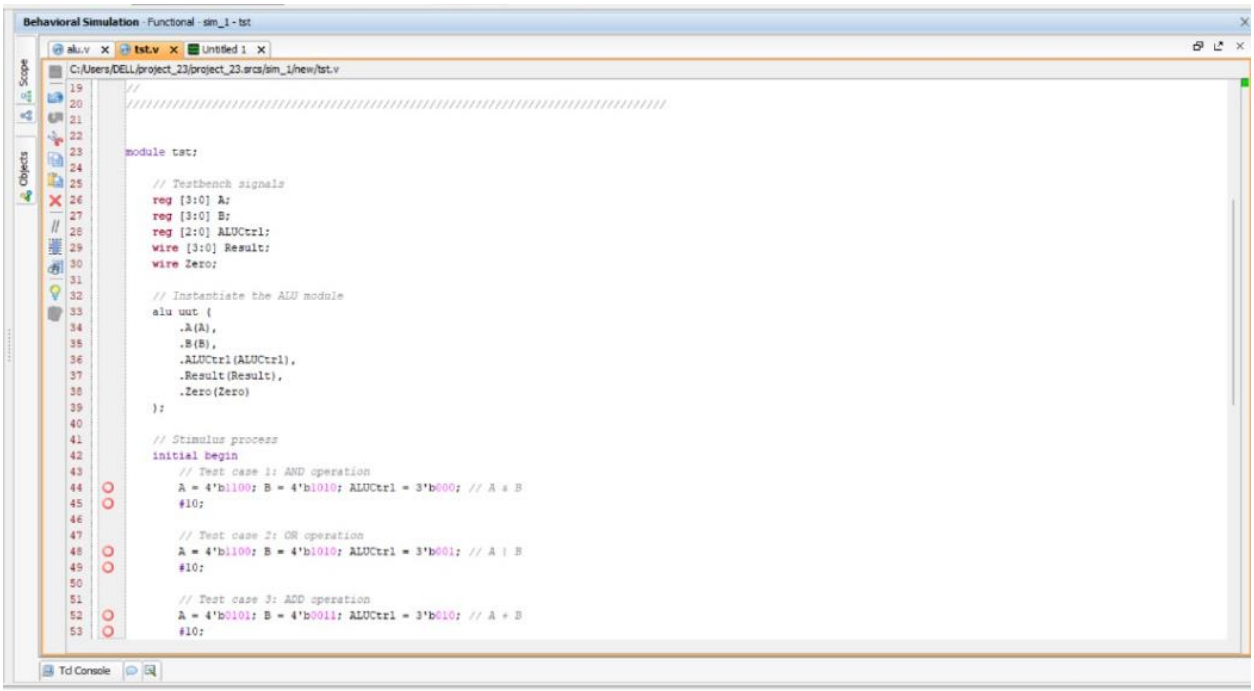
```
        ALUCtrl => ALUCtrl,
        Result  => Result,
        Zero    => Zero
    );
  process
  begin
    -- Test Case 1: A = 0101, B = 0011, ALUCtrl = 000 (AND operation)
    A <= "0101"; B <= "0011"; ALUCtrl <= "000";
    wait for 10 ns;
    -- Test Case 2: A = 0101, B = 0011, ALUCtrl = 001 (OR operation)
    A <= "0101"; B <= "0011"; ALUCtrl <= "001";
    wait for 10 ns;
    -- Test Case 3: A = 0101, B = 0011, ALUCtrl = 010 (ADD operation)
    A <= "0101"; B <= "0011"; ALUCtrl <= "010";
    wait for 10 ns;
    -- Test Case 4: A = 1001, B = 0110, ALUCtrl = 100 (SUB operation)
    A <= "1001"; B <= "0110"; ALUCtrl <= "100";
    wait for 10 ns;
    -- Test Case 5: A = 0000, B = 0000, ALUCtrl = 111 (XNOR operation, zero flag test)
    A <= "0000"; B <= "0000"; ALUCtrl <= "111";
    wait for 10 ns;
    -- Stop simulation
    wait;
  end process;
end Behavioral;
```

IMPLEMENTATION OF VERILOG CODE:

The code is implemented in Xilinx vivado. It is a software suits for designing and analyzing hardware description languages (HDL) designs.

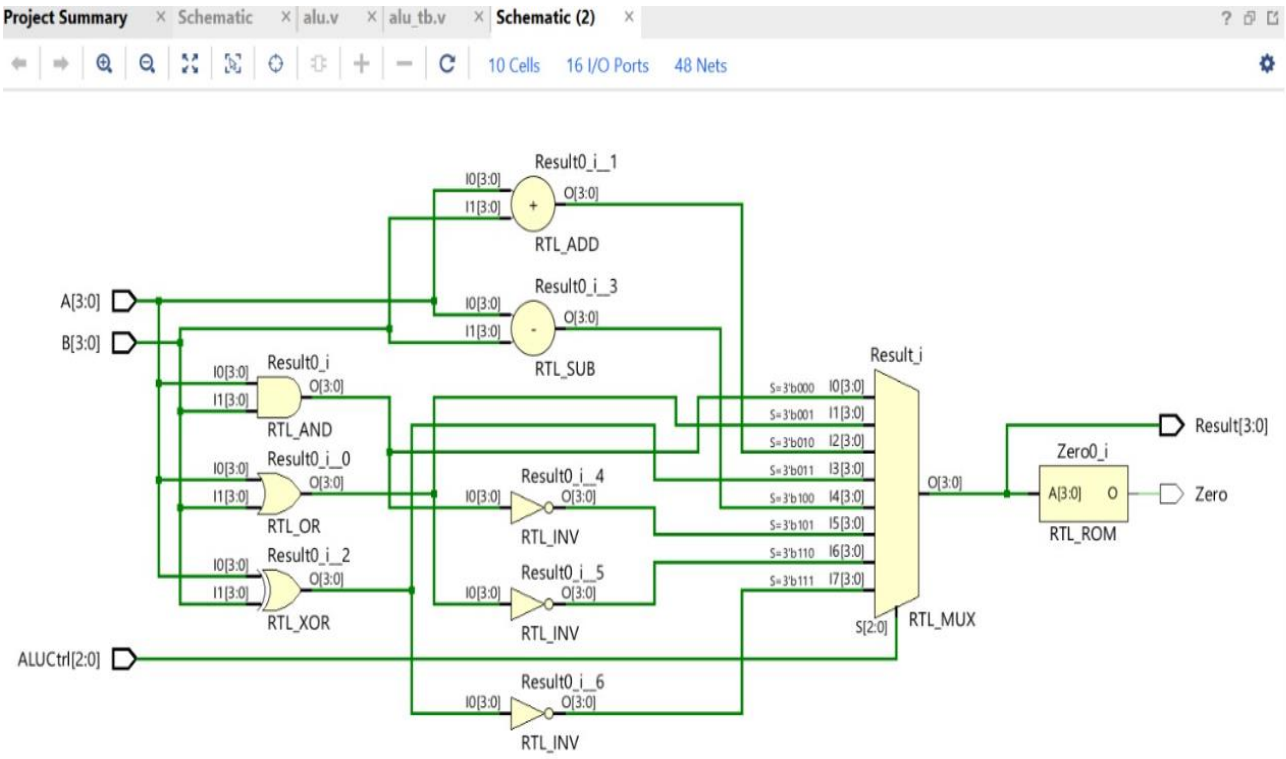



```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module alu(
    input [3:0] A,        // 4-bit input A
    input [3:0] B,        // 4-bit input B
    input [2:0] ALUCtrl,  // 3-bit control signal for selecting operation
    output reg [3:0] Result,  // 4-bit result output
    output reg Zero          0  Zero flag
);

    always @(*) begin
        case (ALUCtrl)
            3'b000: Result = A & B;   // AND operation
            3'b001: Result = A | B;   // OR operation
            3'b010: Result = A + B;   // ADD operation
            3'b011: Result = A ^ B;   // XOR operation
            3'b100: Result = A - B;   // SUBTRACT operation
            3'b101: Result = ~(A & B); // NAND operation
            3'b110: Result = ~(A | B); // NOR operation
            3'b111: Result = ~(A ^ B); // XNOR operation
            default: Result = 4'b0000; // Default case
        endcase

        // Set Zero flag (1 if Result is zero, 0 otherwise)
        Zero = (Result == 4'b0000) ? 1'b1 : 1'b0;
    end

endmodule
```




```
//
//////////////////////////////////////////////////////////////////////////////////


module tst;

    // Testbench signals
    reg [3:0] A;
    reg [3:0] B;
    reg [2:0] ALUCtrl;
    wire [3:0] Result;
    wire Zero;

    // Instantiate the ALU module
    alu uut (
        .A(A),
        .B(B),
        .ALUCtrl(ALUCtrl),
        .Result(Result),
        .Zero(Zero)
    );

    // Stimulus process
    initial begin
        // Test case 1: AND operation
        A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b000; // A & B
        #10;

        // Test case 2: OR operation
        A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b001; // A | B
        #10;

        // Test case 3: ADD operation
        A = 4'b0101; B = 4'b0011; ALUCtrl = 3'b010; // A + B
        #10;
```


Behavioral Simulation - Functional - sim_1 - tst
alu.v
tst.v
Untitled 1
C:/Users/DELL/project_23/project_23.srcs/sim_1/new/tst.v
// Test case 2: OR operation
A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b001; // A | B
#10;
// Test case 3: ADD operation
A = 4'b0101; B = 4'b0011; ALUCtrl = 3'b010; // A + B
#10;
// Test case 4: XOR operation
A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b011; // A ^ B
#10;
// Test case 5: SUB operation
A = 4'b0101; B = 4'b0011; ALUCtrl = 3'b100; // A - B
#10;
// Test case 6: NAND operation
A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b101; // ~(A & B)
#10;
// Test case 7: NOR operation
A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b110; // ~(A | B)
#10;
// Test case 8: XNOR operation
A = 4'b1100; B = 4'b1010; ALUCtrl = 3'b111; // ~(A ^ B)
#10;
// Test case 9: Check Zero flag (Result = 0)
A = 4'b0000; B = 4'b0000; ALUCtrl = 3'b010; // A + B = 0
#10;
$finish; // End the simulation
end

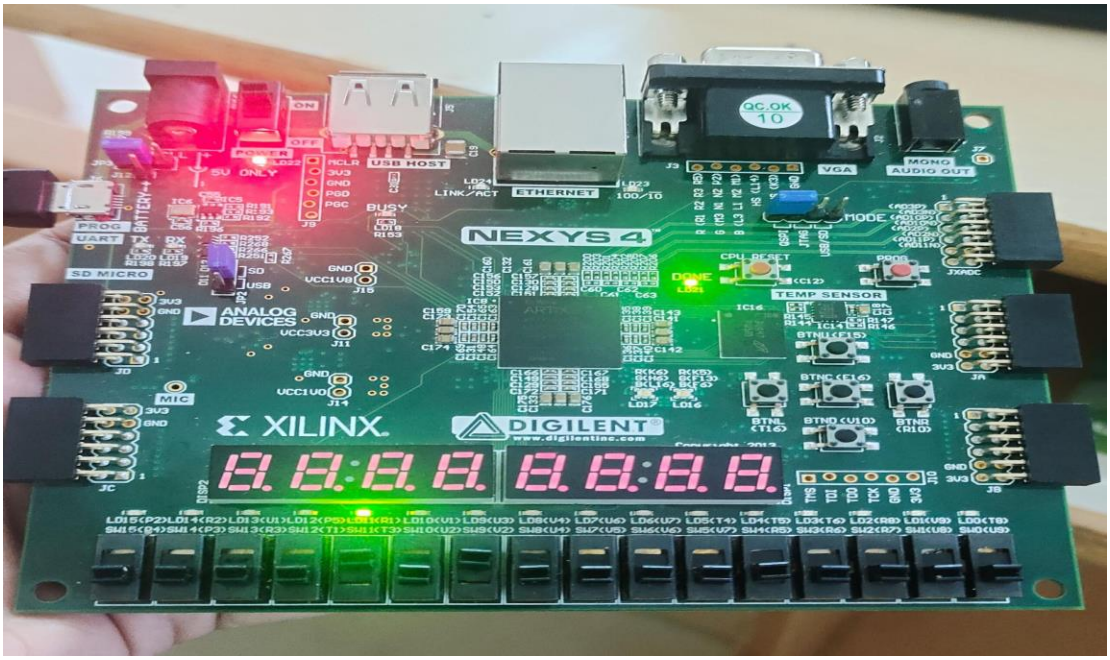

Project Summary
Schematic
alu.v
alu_tb.v
Schematic (2)
10 Cells
16 I/O Ports
48 Nets
A[3:0]
B[3:0]
ALUCtrl[2:0]
Result0_i_1
RTL_ADD
Result0_i_3
RTL_SUB
Result0_i
RTL_AND
Result0_i_0
RTL_OR
Result0_i_2
RTL_XOR
Result0_i_4
RTL_INV
Result0_i_5
RTL_INV
Result0_i_6
RTL_INV
Result_i
RTL_MUX
Zero0_i
RTL_ROM
Result[3:0]
Zero

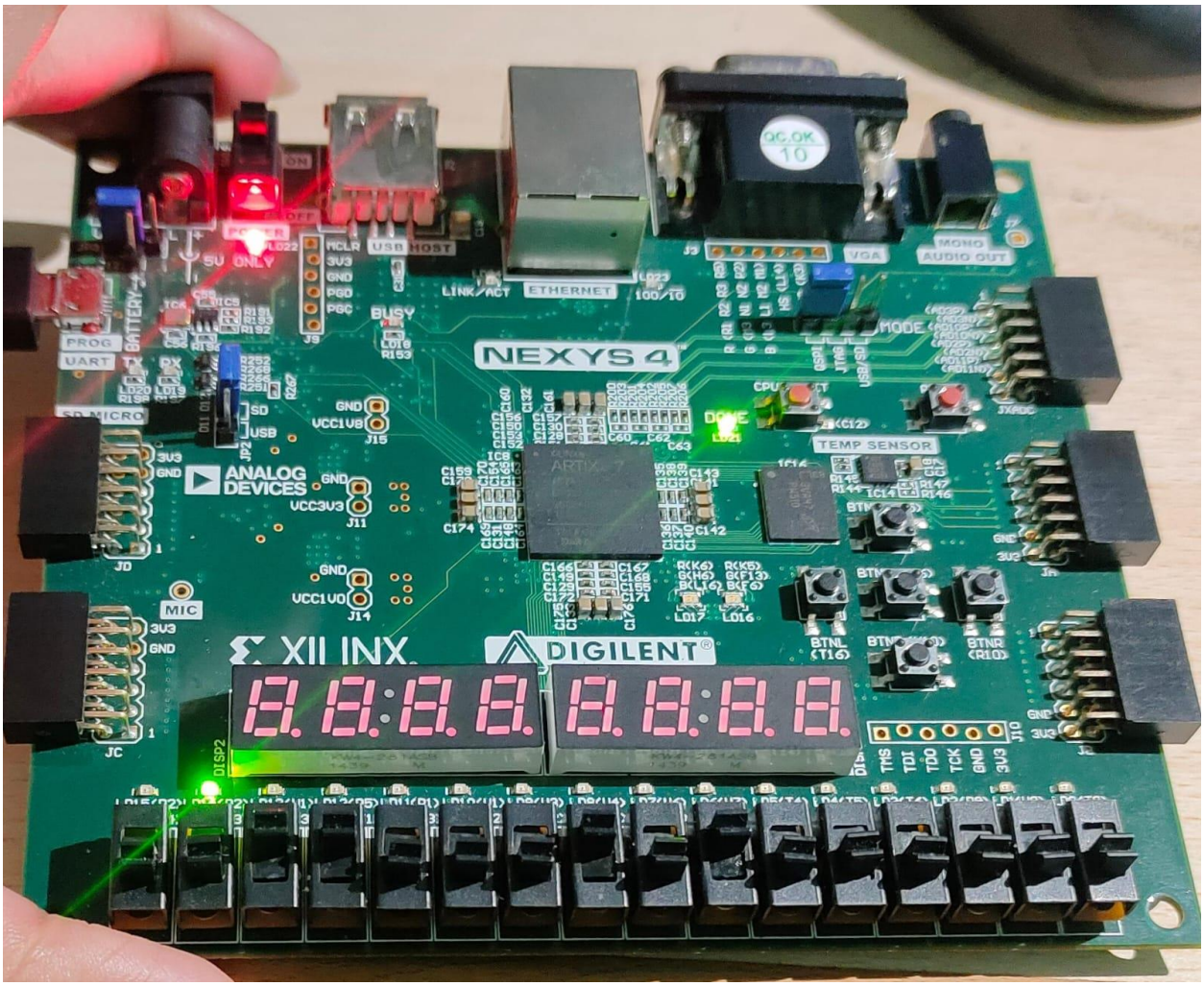
SAMPLE TEST CASE FOR THE TEST BENCH:

Test Case	Inputs	Operation	Expected Result	Expected Zero Flag
1	A = 0101, B = 0011, ALUCtrl = 000	AND	Result = 0001	Zero = 0
2	A = 0101, B = 0011, ALUCtrl = 001	OR	Result = 0111	Zero = 0
3	A = 0101, B = 0011, ALUCtrl = 010	ADD	Result = 1000	Zero = 0
4	A = 1001, B = 0110, ALUCtrl = 100	SUBTRACT	Result = 0011	Zero = 0
5	A = 0000, B = 0000, ALUCtrl = 111	XNOR	Result = 1111	Zero = 0
6	A = 0011, B = 0011, ALUCtrl = 011	XOR	Result = 0000	Zero = 1

SIMULATION RESULT:

FPGA IMPLEMENTATION:





CONCLUSION:

The ALU (Arithmetic Logic Unit) is an essential part of a digital system that performs various arithmetic and logical operations like addition, subtraction, AND, OR, and more. In this project, we successfully designed an ALU that can handle different operations based on input values. By testing the ALU with a testbench, we ensured that it works correctly for each operation.

This ALU design is flexible and can be easily expanded for more operations if needed. Overall, the project shows how an ALU functions and how it is tested to ensure accuracy in a digital system