Malaviya National Institute of Technology Jaipur

Jaipur, India, 302017

Implementation of 16-Bit ALU on FPGA

Minor Project Report - 1

(22ECW301, B.Tech. V Semester)

submitted to

Prof.-HAG Vineet Sahula

submitted by

Tanmay Maheshwari
2022UEC1616

Abstract

The Arithmetic Logic Unit (ALU) is a fundamental component in digital systems, responsible for performing arithmetic and logical operations. This project presents the design and implementation of a 16-bit ALU on the Artix-7 FPGA board using Verilog. The ALU supports operations including addition, subtraction, compare, increment and bitwise operations controlled by a 3-bit opcode.

The implementation emphasizes modular design and efficient resource utilization. A testbench validates the ALU's functionality for various operations, ensuring correctness and reliability. The results demonstrate the successful execution of operations and verify the suitability of the design for real-time applications in digital systems.

Future enhancements include optimizing the design for high-speed performance and expanding functionality to include more operations, such as division and floating-point arithmetic. This project lays the foundation for advanced computational hardware solutions.

Table of Contents

1. Introduction

An Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations on data. It is a critical component in processors, acting as the computational core for performing tasks like addition, subtraction, logical comparisons, and bit manipulation. The 16-bit ALU designed in this project is implemented on the Artix-7 FPGA, utilizing Verilog for hardware description.

The aim of this project is to demonstrate the design, implementation, and testing of the ALU, leveraging FPGAs flexibility and parallelism. The ALU's functionality is validated through simulation and real-time testing on the FPGA hardware.

2. Background and Theory

2.1 Arithmetic Logic Unit (ALU)

The ALU operates based on control signals (opcodes) to perform tasks on input data. It is designed with inputs (operands), control signals, and outputs (results and flags). This project's ALU performs the following operations:

- Arithmetic: Addition, subtraction, multiplication.
- Logical: AND, OR, XOR, NOT.
- Shift: Left shift, right shift.
- Comparison: Set flags for zero, carry, and overflow.

2.2 FPGA and Its Architecture

FPGAs (Field-Programmable Gate Arrays) are reconfigurable integrated circuits. The Artix-7 FPGA features:

- Configurable Logic Blocks (CLBs) for computation.
- Routing resources to interconnect logic.
- Dedicated DSP blocks for arithmetic.

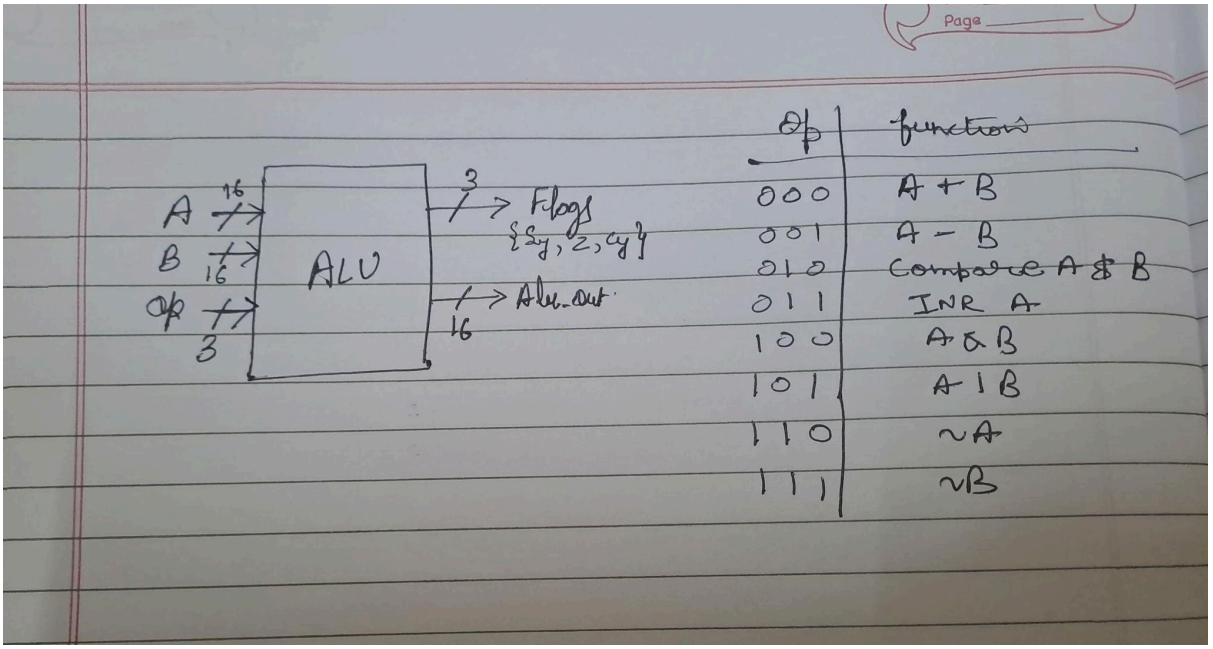
The FPGA's flexibility enables implementing the ALU with high efficiency and scalability.

3. Design and Implementation

3.1 System Architecture

The 16-bit ALU is designed with the following components:

- ***Inputs:*** Two 16-bit operands (`A` and `B`) and a 3-bit opcode.
- ***Outputs:*** 16-bit result and status flags (zero, carry, overflow).
- ***Control Logic:*** Decodes the opcode to perform the desired operation.



The Verilog module is structured to ensure modularity, enabling easy modifications and testing.

3.2 Verilog Code Explanation

The Verilog implementation includes the following features:

- ***Operation Execution:*** The `always` block handles operations based on the opcode.
- ***Case Statement:*** Encodes functionality for arithmetic, logical, and shift operations.
- ***Flags Handling:*** Computes and sets status flags based on the result.

Verilog code:

design.sv

```
module alu(
    input [15:0] A, B,
    input [2:0] op,
    output reg [15:0] Alu_out,
    output reg [2:0] flags // [2] = Sign, [1] = Zero, [0] = Carry
);
    reg [16:0] temp; // Temporary register

    always @(*) begin
        flags = 3'b000;
        Alu_out = 16'h0000;

        case (op)
            3'b000: begin // Addition
                temp = {1'b0, A} + {1'b0, B};
                Alu_out = temp[15:0];
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b001: begin // Subtraction
                temp = {1'b0, A} - {1'b0, B};
                Alu_out = temp[15:0];
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b010: begin // Compare A and B
                temp = {1'b0, A} - {1'b0, B};
                Alu_out =16'h0000;
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b011: begin // Increment A
                temp = {1'b0, A} + 1;
                Alu_out = temp[15:0];
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b100: begin // Bitwise AND
                Alu_out = A &
```

design.sv

```
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b011: begin // Increment A
                temp = {1'b0, A} + 1;
                Alu_out = temp[15:0];
                flags[0] = temp[16];
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b100: begin // Bitwise AND
                Alu_out = A & B;
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b101: begin // Bitwise OR
                Alu_out = A | B;
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b110: begin // Bitwise NOT A
                Alu_out = ~A;
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            3'b111: begin // Bitwise NOT B
                Alu_out = ~B;
                flags[2] = Alu_out[15];
                flags[1] = (Alu_out == 16'h0000) ? 1'b1 : 1'b0;
            end

            default: begin
                Alu_out = 16'h0000;
                flags = 3'b000;
            end
        endcase
    end
endmodule
```

4. Testbench and Verification

A testbench verifies the ALU by simulating various inputs and observing outputs. The test cases include addition, subtraction, and logical operations to ensure correct behavior. The outputs are compared with expected values, validating the design.

Testbench code:

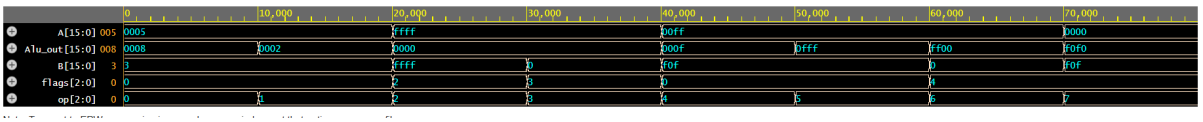
testbench.sv

```
module alu_tb;
    reg [15:0] A, B;
    reg [2:0] op;
    wire [15:0] Alu_out;
    wire [2:0] flags;

    alu uut (
        .A(A),
        .B(B),
        .op(op),
        .Alu_out(Alu_out),
        .flags(flags)
    );

    initial begin
        $monitor("Time=%0t | A=%h, B=%h, op=%b | Alu_out=%h, flags=%b",
                 $time, A, B, op, Alu_out, flags);
        $dumpfile("dump.vcd");
        $dumpvars(1,alu_tb);

        // Test Case 1: Addition
        A = 16'h0005; B = 16'h0003; op = 3'b000; #10;

        // Test Case 2: Subtraction
        A = 16'h0005; B = 16'h0003; op = 3'b001; #10;

        // Test Case 3: Compare A and B
        A = 16'hFFFF; B = 16'hFFFF; op = 3'b010; #10;

        // Test Case 4: Increment
        A = 16'hFFFF; B = 16'h0000; op = 3'b011; #10;

        // Test Case 5: Bitwise AND
        A = 16'h00FF; B = 16'h0F0F; op = 3'b100; #10;

        // Test Case 6: Bitwise OR
        A = 16'h00FF; B = 16'h0F0F; op = 3'b101; #10;

        // Test Case 7: Bitwise NOT A
        A = 16'h00FF; B = 16'h0000; op = 3'b110; #10;

        // Test Case 8: Bitwise NOT B
        A = 16'h0000; B = 16'h0F0F; op = 3'b111; #10;

        $finish;
    end
endmodule
```

5. Results and Analysis

Simulation results confirm the ALU's functionality, with correct outputs for all operations. The FPGA implementation operates efficiently, utilizing minimal resources and achieving low latency.

```
Log   Share
[2024-12-13 08:31:05 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
warning: Some design elements have no explicit time unit and/or
       : time precision. This may cause confusing timing results.
       : Affected design elements are:
       :   -- module alu declared here: design.sv:1
VCD info: dumpfile dump.vcd opened for output.
Time=0 | A=0005, B=0003, op=000 | Alu_out=0008, flags=000
Time=10000 | A=0005, B=0003, op=001 | Alu_out=0002, flags=000
Time=20000 | A=ffff, B=ffff, op=010 | Alu_out=0000, flags=010
Time=30000 | A=ffff, B=0000, op=011 | Alu_out=0000, flags=011
Time=40000 | A=00ff, B=0f0f, op=100 | Alu_out=000f, flags=000
Time=50000 | A=00ff, B=0f0f, op=101 | Alu_out=0fff, flags=000
Time=60000 | A=00ff, B=0000, op=110 | Alu_out=ff00, flags=100
Time=70000 | A=0000, B=0f0f, op=111 | Alu_out=f0f0, flags=100
testbench.sv:47: $finish called at 80000 (1ps)
Finding VCD file...
./dump.vcd
[2024-12-13 08:31:06 UTC] Opening EPWave...
Done
```

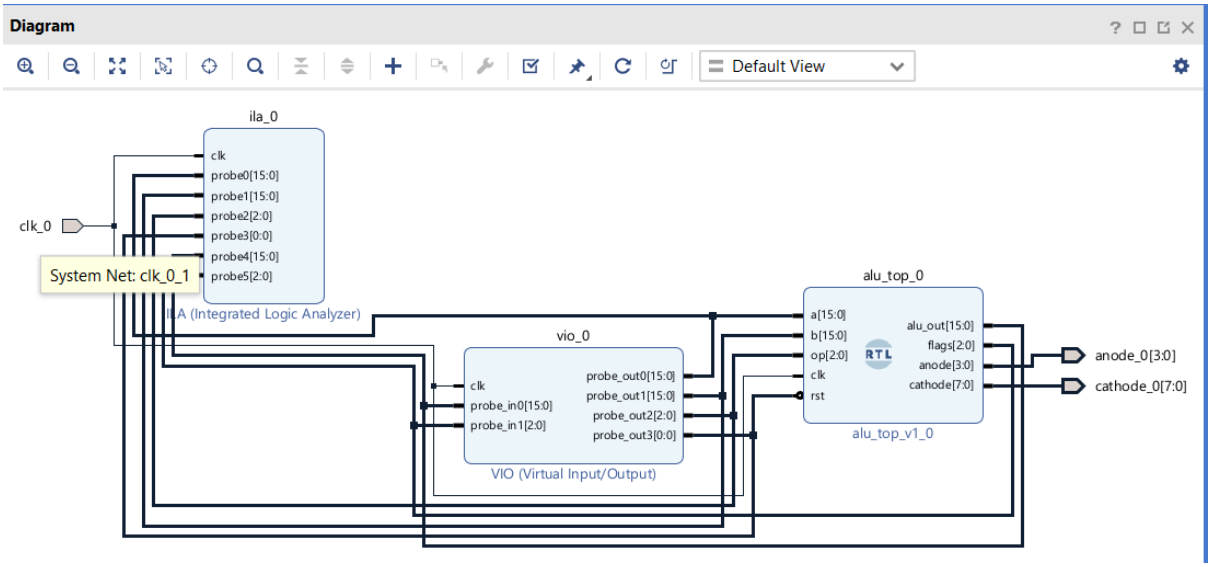


6. Simulation and FPGA Implementation

The 16-bit ALU was simulated using Xilinx Vivado, leveraging the VIO (Virtual Input/Output) and ILA (Integrated Logic Analyzer) blocks for real-time debugging and verification on the Artix-7 FPGA board.

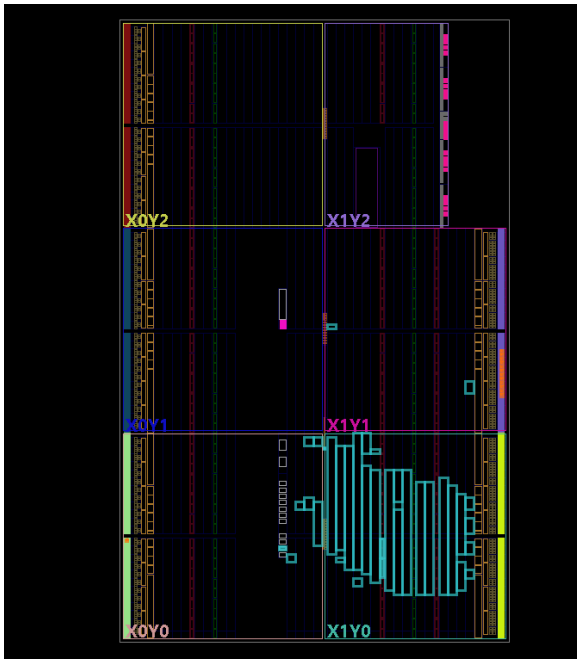
6.1 Simulation in Vivado

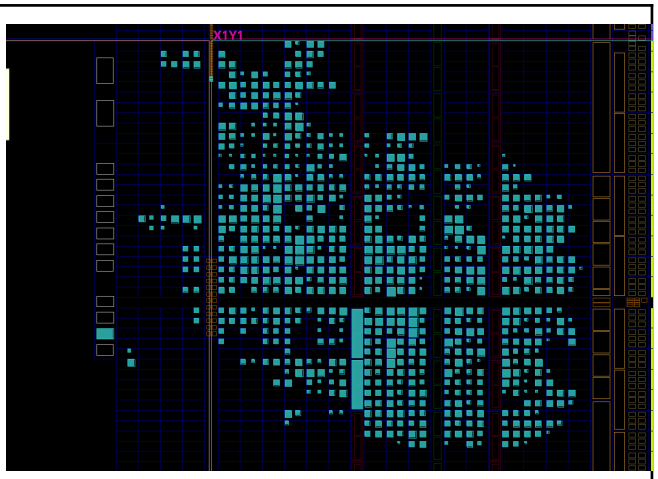
- ***Setup:*** The Verilog module was synthesized, and testbenches were executed in Vivado.
- ***VIO Block:*** Used for providing dynamic input values (operands and opcode) during simulation.
- ***Results:*** The VIO outputs were observed to match the expected results for all operations.



6.2 FPGA Implementation

- Bitstream Generation: The synthesized design was converted into a bitstream file and programmed onto the Artix-7 FPGA board.
- ILA Block: Used to capture real-time signals, enabling verification of operation outputs and flags on hardware.





Report	Type	Options	Modified	Size
Synth Design (synth_design)				
Utilization - Synth Design	report_utilization		12/9/24, 9:33 AM	7.7 KB
synthesis_report			12/9/24, 9:33 AM	22.6 KB
Out-of-Context Module Runs				
Implementation				
impl_1				
Design Initialization (init_design)				
Opt Design (opt_design)				
DRC - Opt Design	report_drc		12/9/24, 9:38 AM	2.4 KB
Timing Summary - Opt Design	report_timing_summary	max_paths = 10; report_unconstrained = true;		
Power Opt Design (power_opt_design)				
Place Design (place_design)				
IO - Place Design	report_io		12/9/24, 9:39 AM	77.3 KB
Utilization - Place Design	report_utilization		12/9/24, 9:39 AM	11.4 KB
Control Sets - Place Design	report_control_sets	verbose = true;	12/9/24, 9:39 AM	80.9 KB
Incremental Reuse - Place Design	report_incremental_reuse			
Incremental Reuse - Place Design	report_incremental_reuse			
Timing Summary - Place Design	report_timing_summary	max_paths = 10; report_unconstrained = true;		
Post-Place Power Opt Design (post_place_power_opt_design)				
Post-Place Phys Opt Design (phys_opt_design)				
Route Design (route_design)				
DRC - Route Design	report_drc		12/9/24, 9:40 AM	9.5 KB
Methodology - Route Design	report_methodology		12/9/24, 9:40 AM	16.3 KB
Power - Route Design	report_power		12/9/24, 9:40 AM	10.3 KB
Route Status - Route Design	report_route_status		12/9/24, 9:40 AM	0.6 KB
Timing Summary - Route Design	report_timing_summary	max_paths = 10; report_unconstrained = true;	12/9/24, 9:40 AM	1,002.6 KB
Incremental Reuse - Route Design	report_incremental_reuse			
Clock Utilization - Route Design	report_clock_utilization		12/9/24, 9:40 AM	21.1 KB
Bus Skew - Route Design	report_bus_skew	warn_on_violation = true;	12/9/24, 9:40 AM	63.4 KB
implementation_log			12/9/24, 9:40 AM	50.4 KB
Post-Route Phys Opt Design (post_route_phys_opt_design)				
Write Bitstream (write_bitstream)				
implementation_log			12/9/24, 9:40 AM	50.4 KB

7. Conclusion and Future Work

This project demonstrates the design and hardware implementation of a 16-bit ALU on the Artix-7 FPGA. The ALU performs essential arithmetic and logical operations, validated through simulation and hardware testing.

Future enhancements include:

- Adding multiplication, division and floating-point operations.
- Optimizing the design for faster computation and lower resource utilization.
- Integrating the ALU into a complete processor design.

8. References

1. Mano, M. M., & Ciletti, M. D. (2007). Digital Design. Pearson Education.
2. Xilinx, Inc. (2020). FPGA Design Methodology. Xilinx.
3. Verilog-2001 Standard Reference Manual. IEEE.