DESIGN DOCUMENT-ALU PROJECT

Introduction

The Arithmetic Logic Unit (ALU) serves as a core component within the data path of processors, digital signal processors, and microcontrollers. It is responsible for executing a wide range of arithmetic and logical operations that are essential for processing data. This project focuses on designing a parameterized ALU in Verilog that performs both arithmetic and logical operations on 8-bit operands, along with proper flag generation to reflect operation status such as overflow, carry, equality, and error.

The ALU supports operations such as addition, subtraction, increment, decrement, signed arithmetic, bitwise logic (AND, OR, XOR, etc.), and shift/rotate functions. Additionally, the module is designed to operate with a one-clock-cycle delay between input application and result availability, facilitating pipeline integration and synchronization. The design is modular, synthesizable, and verified through extensive simulation.

Objectives

The primary objectives of this project are as follows:

- To implement an ALU module in Verilog that supports a comprehensive set of arithmetic and logical operations with both unsigned and signed operand support.
- To incorporate internal flag mechanisms for carry-out (COUT), overflow (OFLOW), equal (E), less-than (L), greater-than (G), and error (ERR) signals.
- To register the inputs and delay the output by one clock cycle, which is a typical requirement in pipelined CPU architectures.
- To ensure the ALU can handle invalid input scenarios and generate error flags accordingly.

- To verify the ALU functionality through simulations, using testbenches and waveform analysis.
- To prepare the design for synthesis and potential hardware deployment on FPGAs or ASICs.

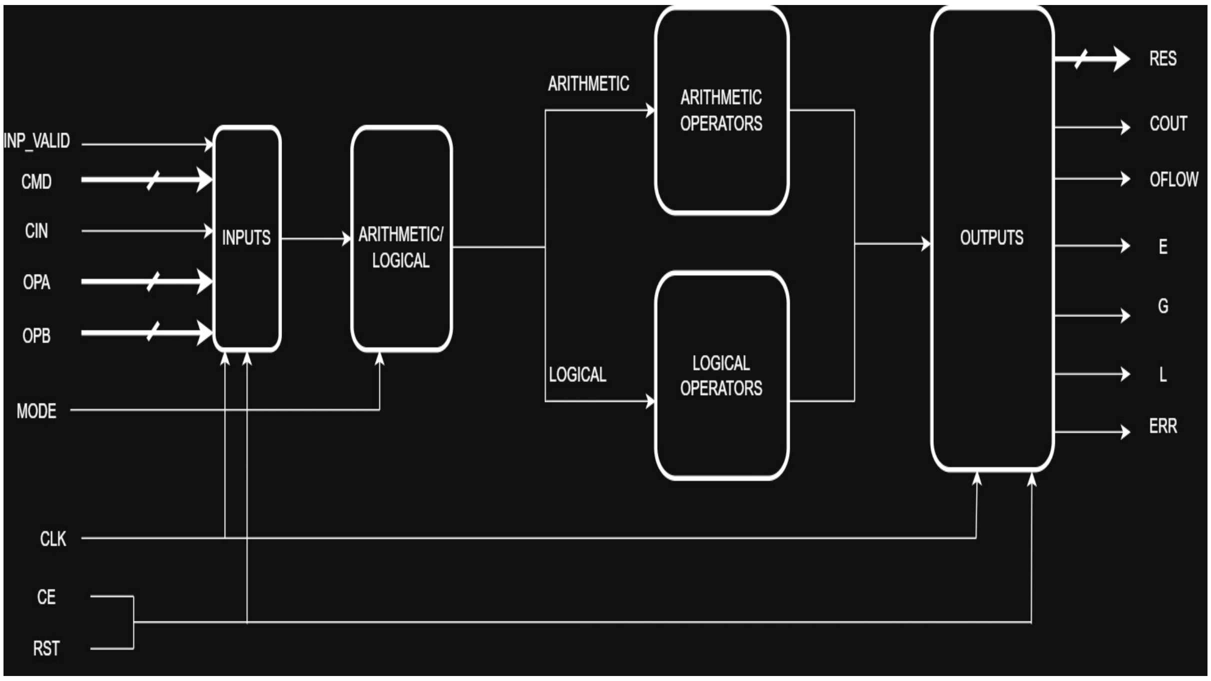
Architecture

The architecture of the ALU module is composed of several functional blocks:

- **Input Stage**: Includes two main operand inputs (OPA, OPB), a command signal (CMD) representing the operation to be performed, a MODE signal to distinguish between arithmetic and logical operations, and additional control signals such as CLK, RST, CE, CIN, and INP_VALID.
- **Input Latching**: All the input signals are latched into temporary registers on the rising edge of the clock. This ensures that the actual computation uses stable, one-cycle-old data — creating a clean separation between input sampling and output processing.
- **Operation Control Logic**: Based on the CMD value and the MODE signal, the ALU routes the operands through different functional paths:

    - Arithmetic operations: Addition, subtraction, increment, decrement, signed addition/subtraction, and custom multiply-like functions.
    - Logical operations: Bitwise operations, NOT, shifts, and rotations.

- **Combinational Processing Block**: This block performs the core logic using a large case statement. It determines the output (RES) and sets the appropriate flags based on the inputs and the operation.
- **Output Stage**: The final output values (RES, COUT, OFLOW, etc.) are updated on the clock edge, introducing the required one-cycle delay. Intermediate results are held in temporary registers like temp_RES, temp_COUT, and others.
- **Parameterization**: The ALU is written using Verilog parameters to support variable operand width (width) and command width (CMD_WIDTH), making the design scalable.



Working

The ALU operates in a sequential-combinational hybrid fashion:

1. **Input Latching**: On each rising clock edge (if CE is high), input operands, command, mode, and flags are captured into temporary registers (temp_OPA, temp_OPB, etc.).
2. **Operation Execution**:

   - If the MODE is high, it selects the arithmetic block. Commands like CMD_ADD, CMD_SUB, CMD_INC_A, etc., are decoded and executed.
   - If the MODE is low, logical operations are performed (e.g., CMD_AND, CMD_OR, CMD_NOT_A, CMD_SHL1_B, etc.).
   - For signed operations (CMD_SIGN_ADD, CMD_SIGN_SUB), the inputs are cast to signed operands before processing.
   - Special operations like compare (CMD_CMP) set the G, E, and L flags without altering the result.

3. **Flag Generation**:

    - COUT is set for carry-out in addition.
    - OFLOW detects arithmetic overflow using sign bit analysis.
    - G, E, and L reflect signed/unsigned comparison results.
    - ERR is flagged for invalid input combinations (e.g., missing valid operand flags).

4. **Output Update**: On the next clock edge, the results and flags are copied from temporary registers to output ports, fulfilling the one-cycle delay requirement.

This design ensures stable timing, controlled data flow, and correct operation regardless of combinational hazards.

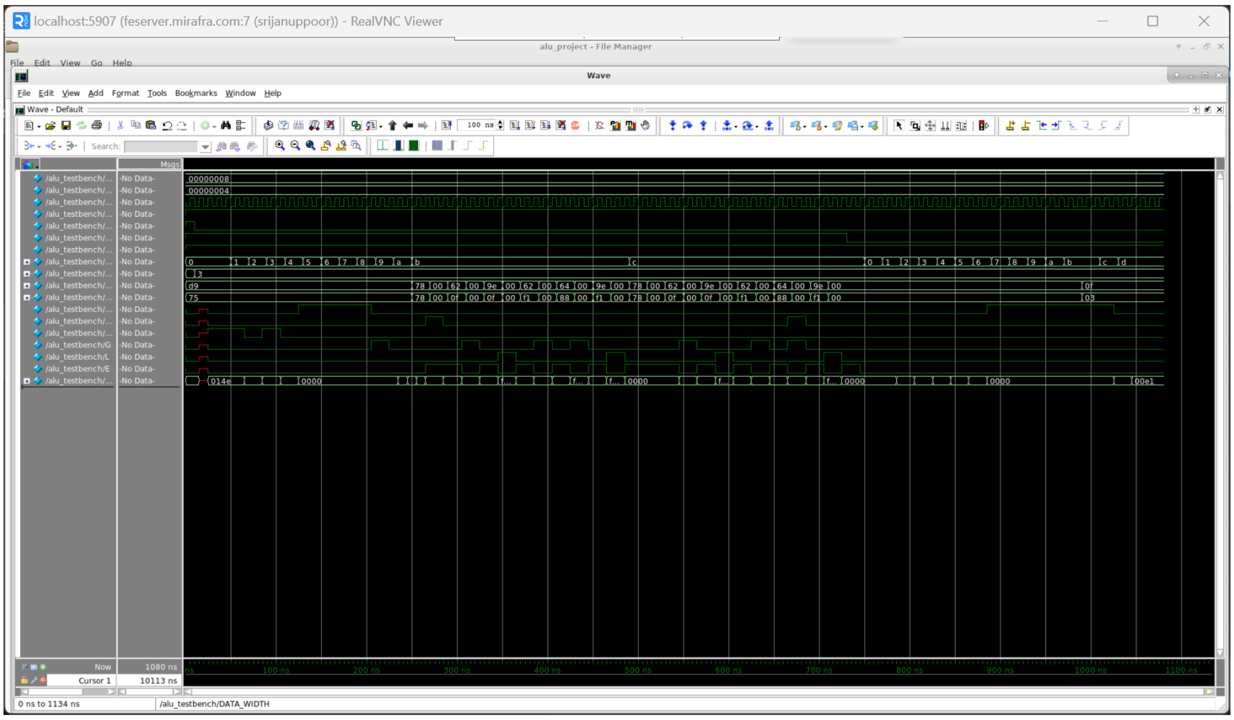
Result

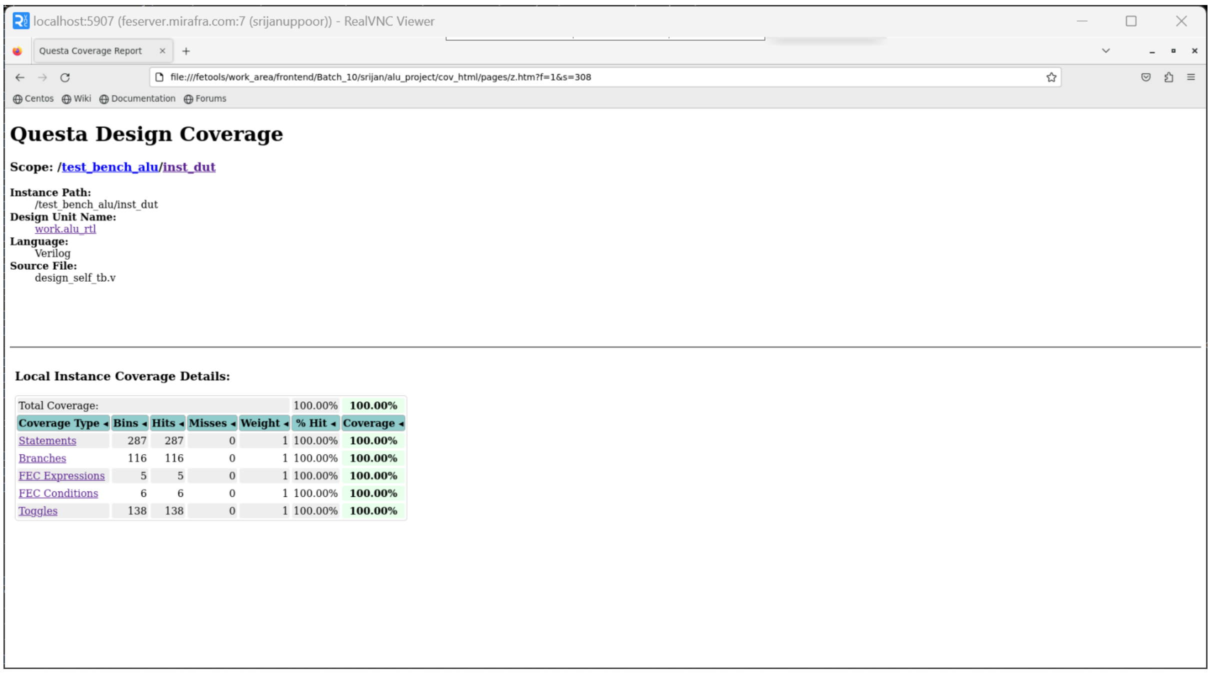
To validate the ALU functionality, a self-testing Verilog testbench was developed. The testbench automatically applies a wide range of stimulus inputs to the ALU, covering all defined arithmetic and logical operations. Each test case includes expected outputs and asserts correctness by comparing actual results against these expectations. This approach eliminates manual checking and ensures repeatable, automated verification.

Key observations from simulation:

- **Arithmetic operations** (ADD, SUB, INC, DEC, signed ops, etc.) produce correct results, including proper COUT and OFLOW flag behavior.
- **Logical operations** (AND, OR, XOR, NOT, shifts, rotates, etc.) execute accurately, and outputs match expected values.
- **Status flags** (G, E, L, ERR) respond correctly to all comparisons and error conditions, including invalid INP_VALID scenarios.
- **One-cycle delay** between input latching and output update is clearly visible in waveform traces and consistent across all operations.
- The testbench includes corner cases such as operand overflows, zero results, equal operands, and mismatched input validity.

- Pass/fail messages are automatically displayed by the testbench for each operation, providing immediate feedback on test coverage.


localhost:5907 (feserver.mirafra.com:7 (srijanuppoor)) - RealVNC Viewer
alu_project - File Manager
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/alu_testbench/DATA_WIDTH
Now 1080 ns
Cursor 1 10113 ns
0 ns to 1134 ns


localhost:5907 (feserver.mirafra.com:7 (srijanuppoor)) - RealVNC Viewer

Questa Coverage Report

file:///fetools/work_area/frontend/Batch_10/srijan/alu_project/cov_html/pages/z.htm?f=1&s=308

Centos Wiki Documentation Forums

Questa Design Coverage

Scope: /test_bench_alu/inst_dut

Instance Path:
/test_bench_alu/inst_dut
Design Unit Name:
work.alu_rtl
Language:
Verilog
Source File:
design_self_tb.v

Local Instance Coverage Details:

Total Coverage:					100.00%	**100.00%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	287	287	0	1	100.00%	**100.00%**
Branches	116	116	0	1	100.00%	**100.00%**
FEC Expressions	5	5	0	1	100.00%	**100.00%**
FEC Conditions	6	6	0	1	100.00%	**100.00%**
Toggles	138	138	0	1	100.00%	**100.00%**

Waveforms generated during simulation (e.g., in GTKWave) clearly show the input latched on one clock edge and the output appearing on the next, validating the design timing.

Conclusion

The ALU design and implementation in Verilog meets all defined functional requirements. It supports a wide range of operations, handles both signed and unsigned arithmetic, and properly generates operation status flags. The modular structure, use of parameters, and delayed output make the design highly adaptable and suitable for integration into larger pipelined systems. The correctness of the implementation was confirmed through simulation and analysis, demonstrating the robustness of the design.

Future Improvement

To enhance the ALU further and align it with advanced processor design practices, the following improvements are proposed:

- **Pipelining**: Add multiple pipeline stages (e.g., decode, execute, write-back) to improve performance in high-frequency environments.
- **More Operations**: Include additional operations like multiplication, division, modulus, shifts with carry, and floating-point support.
- **Formal Verification**: Employ formal methods (e.g., property specification and equivalence checking) to prove correctness for all cases beyond testbench coverage.
- **Power and Area Optimization**: Modify logic to reduce switching activity and gate count for better synthesis results on hardware.
- **Interface Enhancement**: Integrate with standardized bus interfaces (e.g., AXI or Wishbone) for seamless system-on-chip integration.