Rawisara Chairat

U65053623

Lab Report

Introduction

The goal of this lab was to design and implement a parameterized Arithmetic Logic Unit (ALU) using Verilog, which could perform a range of arithmetic and logical operations. By utilizing modular design and parameterization, the ALU was made flexible to support various word sizes.

Hierarchical Design

The ALU was designed using a modular and hierarchical structure. The top-level module, ALU_Nbit, was parameterized to support any word size by passing the bit-width as a parameter and included a register to store the result of the operations. The ALU operation itself was divided into individual 1-bit operations, implemented in the ALU_1bit module. By breaking the design down into these smaller, reusable components, the 1-bit module could be used repetitively for each bit in the full-width ALU, making the design both efficient and easy to maintain.

Main Modules

1. **ALU_Nbit.v:** This is the top-level module that instantiates N ALU_1bit modules using a generate statement. It supports ALU operations, including MOV, NOT, ADD, SUB, OR, AND, and SLT. It also incorporates a N-bit register to restore the output by passing N as a parameter for the Nbit_Register.
2. **ALU_1bit.v:** This module handles the operations for a single bit, taking in two inputs and a carry in, and producing the results for that bit along with carry out.

   2.1. NOT.v: Implements a NOT gate for bitwise inversion.

   2.2. Nbit_Adder.v: A parameterized 1-bit adder module that uses n=1 to handle addition operations.

      2.2.1. FA_str.v: A full-adder module used within the adder to compute the sum and carry out.

   2.3. Nbit_Subtracter.v: A parameterized 1-bit subtractor module, using n=1 for subtraction operations.

      2.3.1. FS_str.v: A full-subtractor module used within the subtractor to compute the difference and borrow-out.

   2.4. OR_T_2.v: A module that implements an OR gate for the two inputs.

   2.5. AND_T_2.v: This module implements an AND gate for two inputs and also handles the less-than comparison for the SLT operation.
3. **Nbit_Register.v:** A register module that stores the final output of the ALU, updating on the positive edge of the clock signal.

ALU Functions

The ALU performs the following operations, controlled by the ALUOp signal:

ALUOp	Operation
3’b000	R1 = R2 (MOV)
3’b001	R1 = ~R2 (NOT)
3’b010	R1 = R2 + R3 (ADD)
3’b011	R1 = R2 – R3 (SUB)
3’b100	R1 = R2 | R3 (OR)
3’b101	R1 = R2 & R3 (AND)
3’b110	R1 = 1 if R2 < R3 (SLT)

Rawisara Chairat

U65053623

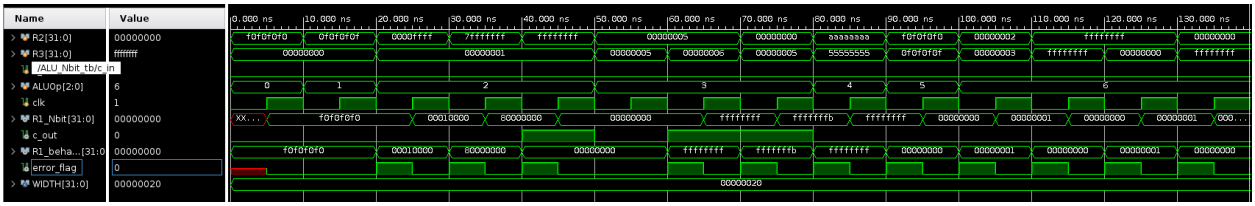
Implementation

Steps in the design process:

1. 1-bit ALU Slice Implementation: The first task was to create the 1-bit ALU slice, which included a multiplexer to handle the selection of operations. The slice was then tested for all functions.
2. Parameterized N-bit ALU Implementation: Once the 1-bit slice is functioning accurately, we combined N slices of 1-bit ALU to creates the N-bit ALU. The module was also tested to ensure that it could works correctly across all N bits.
3. Register Implementation: The parameterized N-bit register was added to store the results coming out of ALU. It was also being tested for correct storage of ALU outputs.
4. SLT Implementation: The SLT operation was added last. We ensured that this operation could worked with both positive and negative integers by adding more statements into the N-bit ALU module. Lastly, we tested all the corner cases for this operation to ensure the accurate comparison.

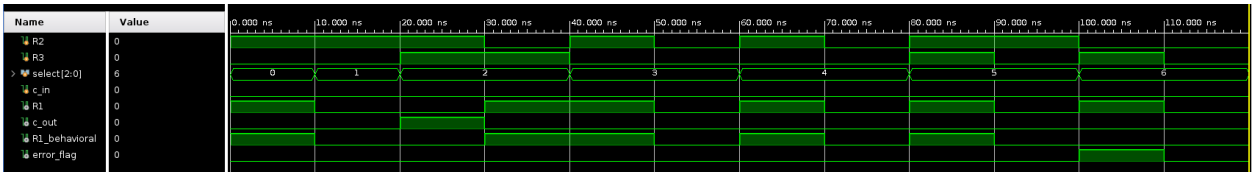
Simulation and Testbench

The testbench was designed to verify the ALU functionality across all operations. A variety of test cases for each operation were selected to cover corner cases, including positive, negative, zero, large input values. Simulation waveforms confirmed that the ALU performed all operations correctly. Notably, the error flag appeared as expected before the rising edge of the clock, since the register must wait for the clock signal to store the value. After the clock edge, the error flag cleared, indicating successful operation and correct output storage in the register, with matches the output from the ALU's behavioral module.



Waveform of an N bit Arithmetic Logic Unit

Since the 1-bit ALU module does not include a register, the testbench for the 1-bit ALU does not incorporate clock-driven storage. However, validation can be done by comparing the output from the 1-bit ALU directly with the parameterized behavioral ALU module set to 1 bit. The simulation results show that the outputs match perfectly, with no error flag present, confirming that the 1-bit ALU operates as expected for all test cases.



Waveform of a 1-bit Arithmetic Logic Unit

Conclusion

This lab successfully implemented a parameterized, hierarchical ALU capable of performing various operations, including bitwise, arithmetic and comparison functions. The modular design ensured flexibility for different bit-widths, making the ALU scalable and reusable. The ALU was thoroughly tested using a testbench, confirming correct functionality.