

CMPE 260 Laboratory Exercise 1

Introduction to Vivado & Simple ALU

Mohammed Fareed
Performed: February 8, 2024
Submitted: February 11, 2024

Lab Section: 4
Instructor: Prof. Richard Cliver
TA: Aubrey Tarmu
Henry Bang
William Tom

Lecture Section: 2
Professor: Prof. Marcin Lukowiak

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: _____

Abstract

In this lab exercise, the functionalities of the Xilinx Vivado Design Suite were explored by designing, implementing, and testing a 4-bit, and later a 32-bit, Arithmetic Logic Unit (ALU) on a Basys3 FPGA. Initially, the project setup included the ALU definition with 'not' and 'shift left logical' operations for a 4-bit design, utilizing a provided testbench for behavioral simulation, and observing synthesis and implementation results. Post-synthesis and post-implementation designs were generated, as well as timing simulations conducted. The project was then scaled to a 32-bit ALU, incorporating additional operations and employing a modified testbench for testing with behavioral timing simulations. The results of the simulations were observed to ensure the ALU's correct operation and timing. The exercise demonstrated the process of designing, implementing, and testing an ALU with the Xilinx Vivado Design Suite.

Design Methodology

The design started with a tutorial-guided setup in Vivado, involving project creation, ALU definition, constraint setup, and testbench application. Initially, the ALU supported only 'not' and 'shift left logical' operations for a 4-bit input.

The design was then scaled to a 32-bit ALU, with additional operations including 'and', 'or', 'xor', 'shift right logical', and 'shift right arithmetic'. Figure 1 shows the block diagram of the designed ALU.

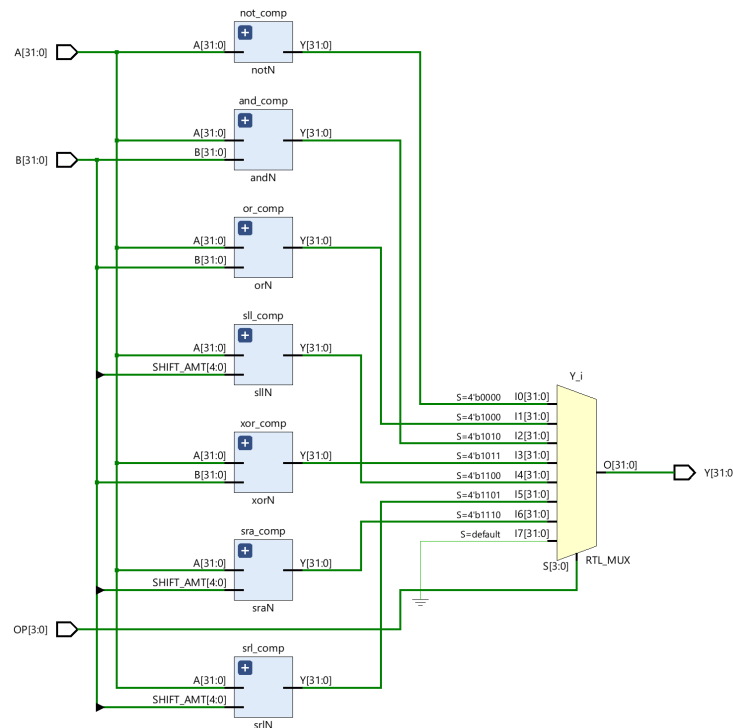


Figure 1: ALU Block Diagram.

The figure above shows the ALU block diagram, with the inputs and outputs labeled. The diagram shows the ALU's signals and operations, including the 32-bit input, 4-bit operation selection, and 32-bit output. The ALU was implemented using VHDL, with global constants defining the ALU input and shift inputs width. The figure is also applicable to the 4-bit ALU, with the only difference being the input and output widths and the shift input width. Table 1 shows the operations supported by the ALU.

OP Code	Operation
0000	NOT
1010	AND
1000	OR
1011	XOR
1100	Shift Left Logical
1101	Shift Right Logical
1110	Shift Right Arithmetic

Table 1: ALU Operations.

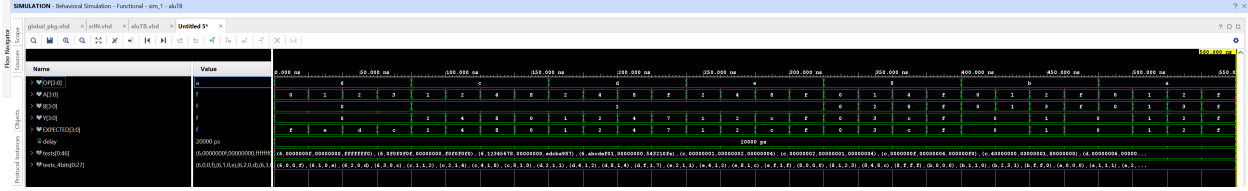
The operations 'and', 'or', 'xor', and 'not' were implemented using native VHDL operators, while the 'shift left logical' was implemented using a generator that creates a new vector holding vectors of the input vector shifted by all possible amounts, with the shift amount being the index of the vector. The output is then set to the vector indexed with the provided shift amount. The 'shift right logical' and 'shift right arithmetic' operations were implemented using the `shit_right` operator in the `ieee.numeric_std` package. The the logical shift used the unsigned value of the input, while the arithmetic shift used the signed value of the input. The maximum shift amount was defined using a global constant, manually calculated to be equal to $\log_2(\text{input_width})$, which is 2 bits for the 4-bit ALU and 5 bits for the 32-bit ALU.

The initial 4-bit ALU was redesigned to support 32-bit inputs and outputs by using the global constants to define the input and output widths and generics to define the operation selection width. Generics allowed the design to be scalable, with the 32-bit ALU supporting the same operations as the 4-bit ALU. Changing the global constants from 4 to 32, and vice versa, results in the design scaling to the new width.

Results and Analysis

The ALU's functionality was verified through behavioral simulation and post-implementation timing simulation. The initial 4-bit design and the enhanced 32-bit design were both tested. The testbench for testing was modified such that a vector of test cases was used to test the ALU's operations, with the expected results being compared to the actual results. Assertions were used to verify the results, reporting an error if the expected and actual results did not match.

4-bit test cases were first used to test the 4-bit ALU. Figure 2 shows the behavioral simulation results for the 4-bit ALU.



The post-synthesis and post-implementation designs were generated for the 32-bit ALU, with the results of the timing simulations shown in Figures 6 and 7.



The testbench was designed such that a record is created that holds the operation, inputs, and expected output. Two vectors were then created that hold the different test cases, one vector with 32-bit inputs and the other with 4-bit inputs. The testbench then iterates through the vectors, performing the operations and comparing the expected and actual results. The testbench was designed to be scalable, with the 32-bit ALU using the same testbench as the 4-bit ALU. Four test cases were used for each operation, with the 32-bit ALU including additional edge cases.

This exercise successfully demonstrated the process of designing, implementing, and testing an ALU with the Xilinx Vivado Design Suite. Starting from a basic 4-bit ALU, the project was scaled to a more complex 32-bit version, with the design verified through extensive testing. The results of the behavioral, synthesis, and implementation simulations indicated that the ALU's operations were functional and reliable, with the 32-bit ALU supporting the same operations as the 4-bit ALU. The exercise provided a comprehensive understanding of the Vivado Design Suite and the process of designing and testing an ALU.

Exercise 1: Introduction to Vivado & Simple ALU

Student's Name: Mohammed Fareed Section: 4

Demo		Point Value	Points Earned	Date
Part 1: 4-bit ALU	Behavioral Simulation	4	4	def 2-8
	Post-Synthesis Timing Simulation	4	4	
	Synthesis Schematic	4	4	
	Synthesis Utilization Report	4	4	
	Post-Implementation Timing Simulation	4	4	
	RTL Schematic for srlN shifter	4	4	
	Hardware Demonstration	4	4	
Part 2: 32-bit ALU	Behavioral Simulation	16	16	
	Post-Implementation Timing Simulation	16	16	

To receive any grading credit students must earn points for both the demonstration and the report.