CMPE 260 Laboratory Exercise 1

Introduction to Vivado and Simple ALU

Shubhang Mehrotra

Performed: 26th January 2021
Submitted: 8th February 2021

Lab Section: 6
Instructor: Mr. Richard Cliver
TA: Corey Sheridan,
Jacob Meyerson,
Georgi Thomas

Lecture Section 1
Professor: Mr. Richard Cliver

By submitting this report I attest that I 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 I further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Signature: S.M _____

Abstract

The scope of this activity was to create a custom design from the source, simulate it, synthesize it, implement it, and load it onto a Basys3 FPGA. These objectives were completed in two steps— first by designing a 4-bit ALU supporting basic NOT and Shift operations. Then expanding the design to support additional operations such as Logical Shift, Arithmetic Shift, AND, OR, and XOR within a 32-bit width ALU.

Design Methodology

The first part of the activity required creating a 4 Bit ALU by collecting the source code provided and assembling it into a Xilinx Vivado project so that it can be further analyzed.
Upon creating the required files for the 4 Bit ALU, a VHDL Testbench was created and simulated, as shown in Figure 1.

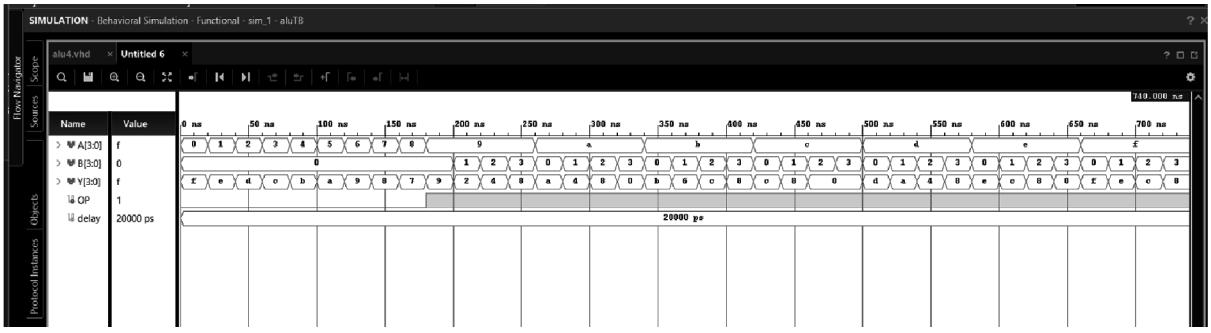

Figure 1. 4 Bit ALU: Behavioral Simulation

Vivado also allows the user to simulate the Synthetic implementation of the VHDL design, hence the 4 Bit ALU was synthesized and then again simulated. Figure 2 shows the post-synthesis Timing simulation, and Figure 3 shows the schematic generated for the 4 Bit ALU.

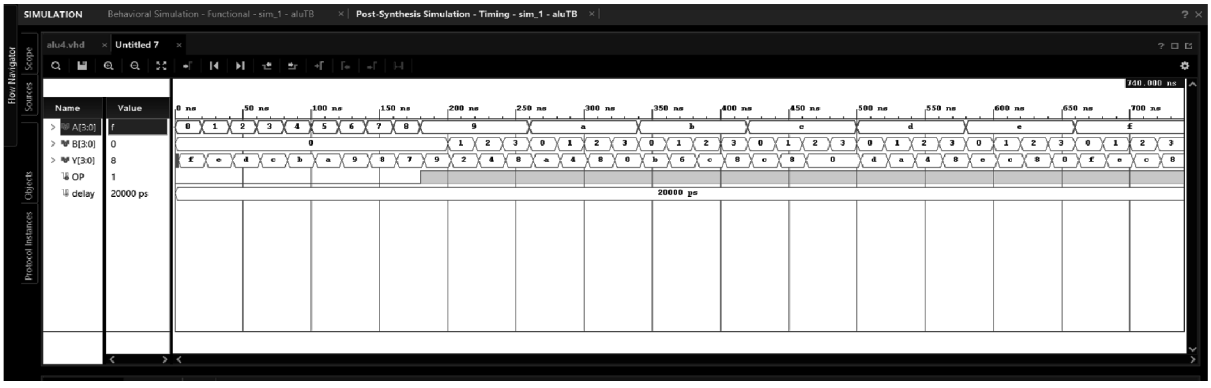

Figure 2. 4 Bit ALU: Post-Synthesis Timing Simulation

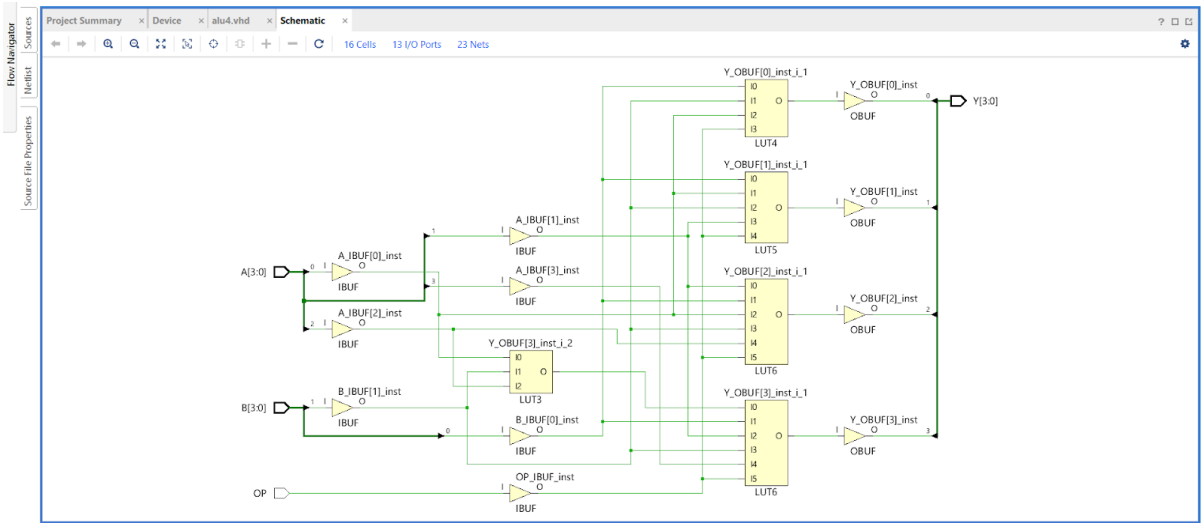

Figure 3. 4 Bit ALU: Synthesis Schematic

A user can also access the Synthetic Utilisation report through Vivado. A Synthetic Utilization report for the 4 Bit ALU is shown in Figures 4.1 and 4.2.

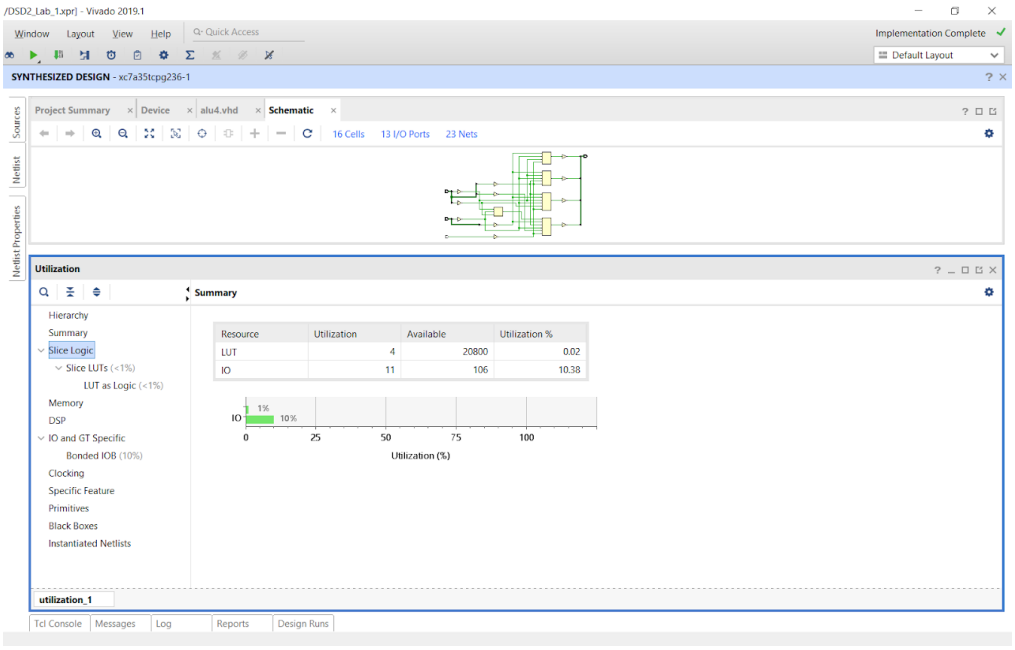

Figure 4.1 4 Bit ALU: Synthesis Utilization Report

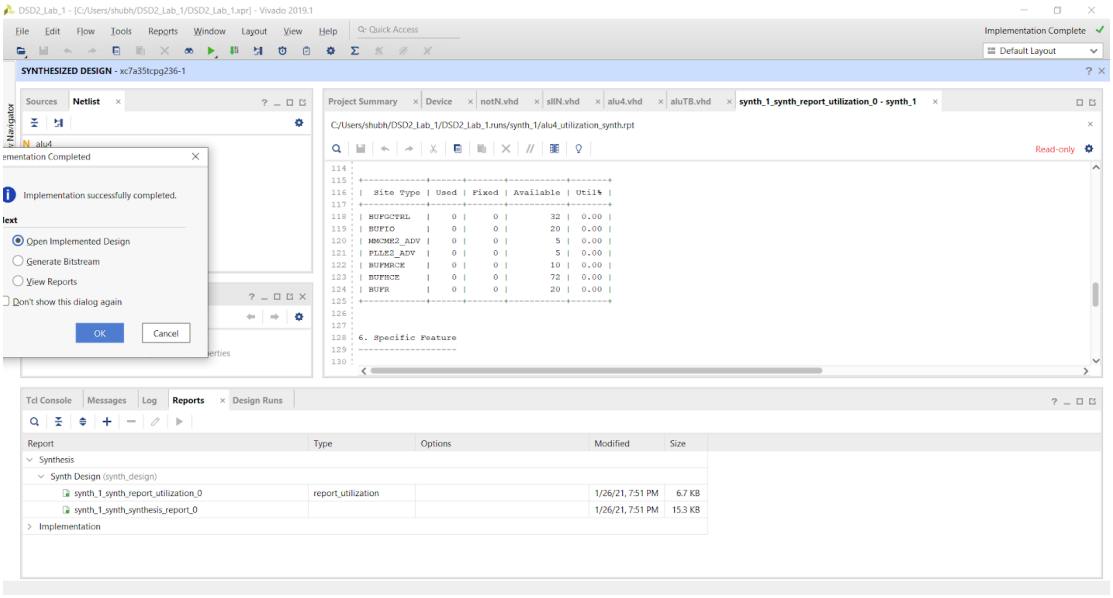

Figure 4.2 4 Bit ALU: Synthesis Utilization Report

Vivado provides the tools for implementation of the design into a model of the FPGA, allowing the user to simulate how the design will behave on the actual hardware. Figure 5 shows the Post-Implementation Timing Simulation diagram for the 4-Bit ALU.

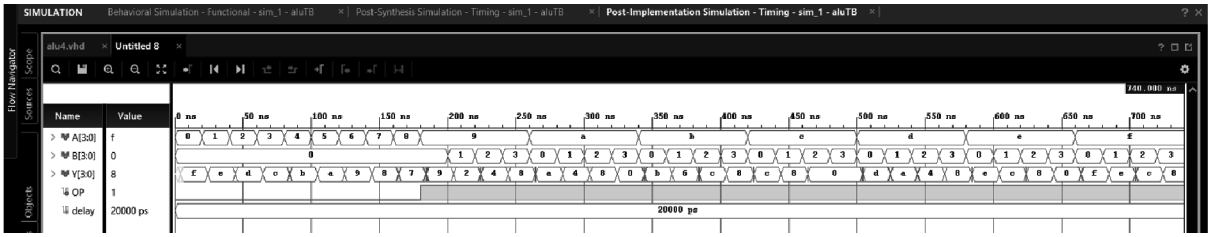

Figure 5. 4 Bit ALU: Post-Implementation Timing Simulation

To add to the functionality of the 4-Bit ALU, a Logical Shift right was also created. Figure 6 shows the RTL Schematic provided by Vivado for the Logical Right Shift component.

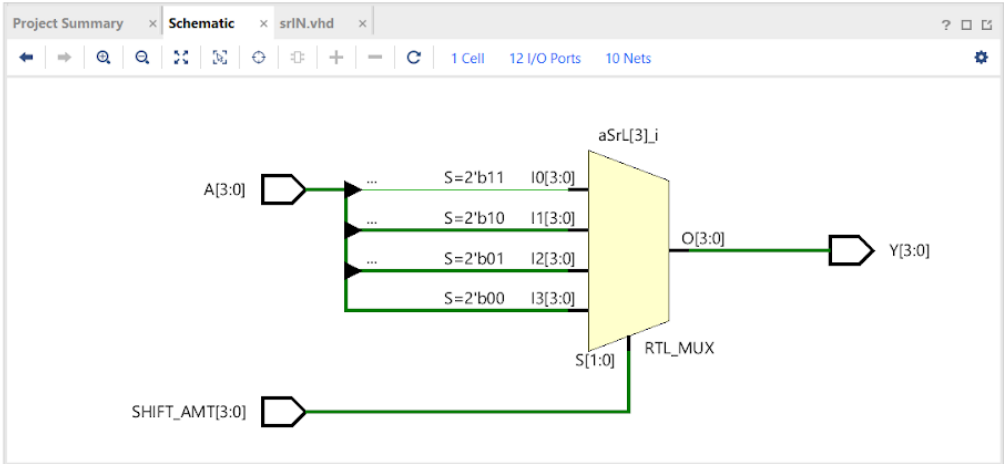

Figure 6. RTL Schematic for srlN shifter

Utilizing the 4-Bit ALU as the starting point, a 32-Bit ALU was then modeled in the second part of the activity using generic N-bit components for added flexibility. The 32 Bit ALU was made capable of performing operations such as Logical AND, Logical OR, Logical XOR, Logical Shift Left, Logical Shift Right, and Arithmetic Shift Right. Figure 7 shows the schematic for the 32-Bit ALU.

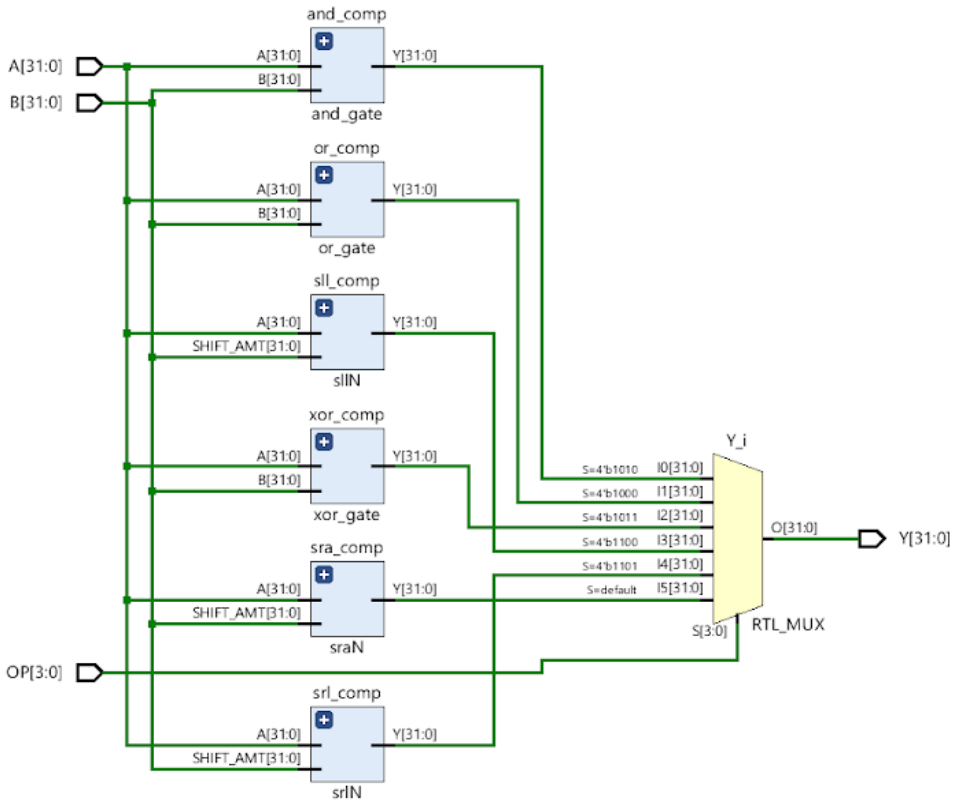

Figure 7. 32 Bit ALU Schematic

The 32 Bit ALU uses the different OP Codes for different operations, the OP Codes are listed in Table 1.

Table. 1 Operations and OP Codes

Opcode	Operation
1000	Logical OR
1010	Logical AND
1011	Logical XOR
1100	Shift Left Logical (SLL)
1101	Shift Right Logical (SRL)
1110	Shift Right Arithmetic (SRA)

To be able to perform more complicated arithmetic, the 32 bit ALU also uses the Shift Right operations, which is implemented in both Logical and Arithmetic form. The Logical Shift Right (SRL), shifts the digits of a byte to the right by the amount inputted. It achieves this by eliminating the digits on the Right Most end (Least Significant Bits), and each preceding digit takes its place, while simultaneously populating the Most Significant Bits with 0s. This

operation successfully works for unsigned integers but is not useful for Signed integers. To rectify that, an Arithmetic Shift Right(SRA) Operation is used to preserve the Most Significant Bits (Sign Bits). SRA works similarly to SRL for the most part, except it populates the MSB with the original MSB, hence retaining the sign of the integer.

Results and Analysis

To verify the functionality of the 32 BIT ALU, a Testbench was created and simulated in a Vivado Project. The TestBench was first simulated behaviorally, and the result obtained is depicted in Figure 8.

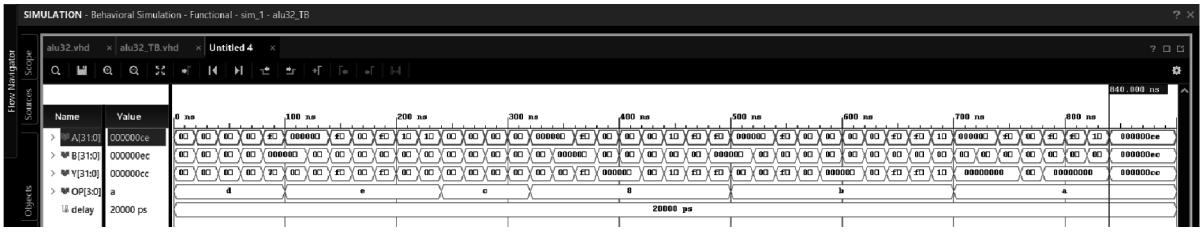

Figure 8. 32 Bit ALU: Behavioral Simulation

Using the tools provided by Vivado, the design was then implemented to depict the real-world usage scenario of the 32 Bit ALU. Figure 9 shows the post-Implementation Timing Simulation.

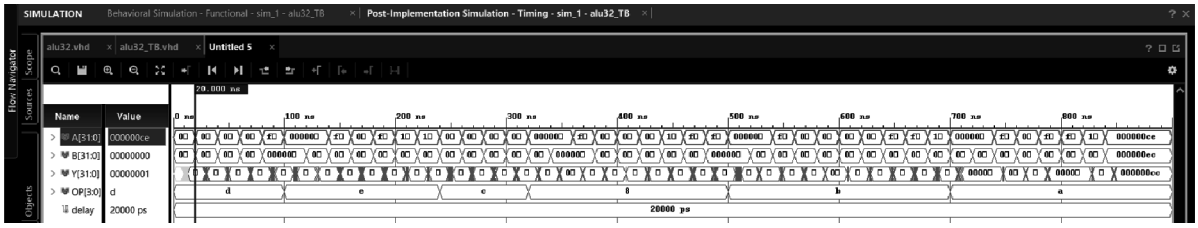

Figure 9. 32 Bit ALU: Post-Implementation Timing Simulation

To test the SRL operation, the Testbench contained the following Generic and Edge cases to ensure maximum functionality. Figure 10 shows the successful test cases for the SRL Operation.

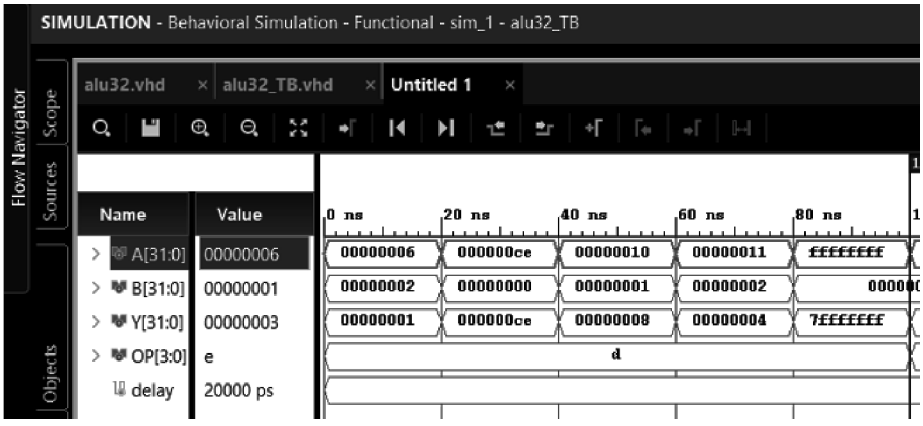

Figure 10. SRL Test Cases

To test the SRA operation, the Testbench contained the following Generic and Edge cases to ensure maximum functionality. Figure 11 shows the successful test cases for the SRL Operation.

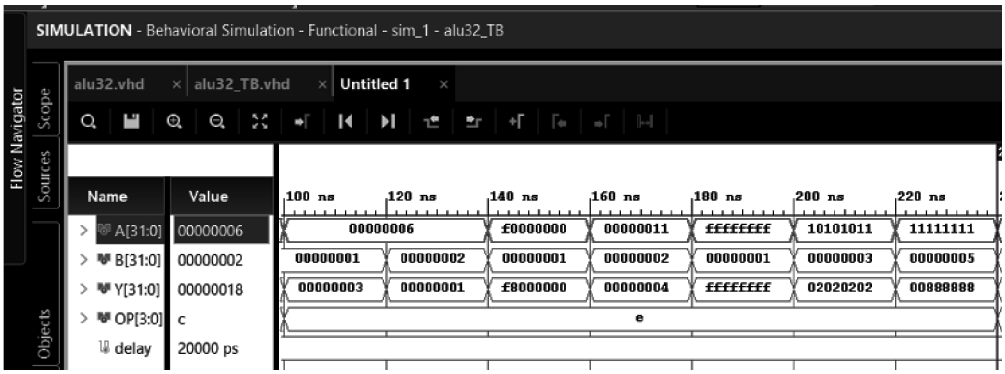

Figure 11. SRA Test cases

To test the SRA operation, the Testbench contained the following Generic cases to ensure maximum functionality. Figure 12 shows the successful test cases for the SRL Operation.

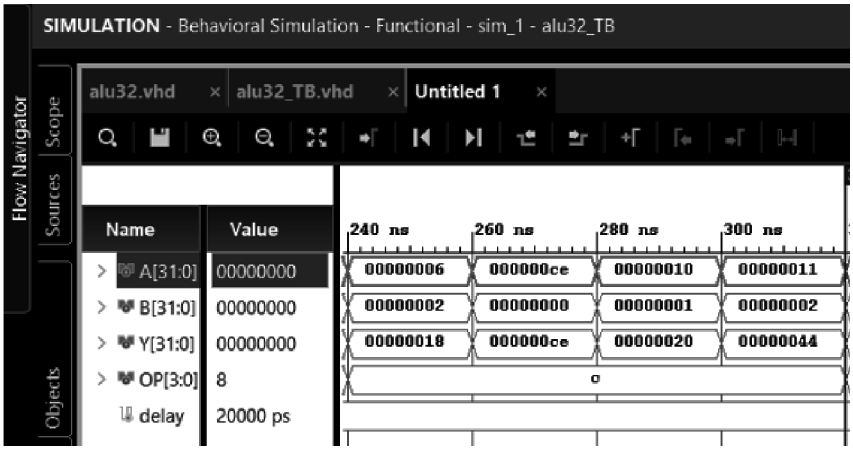

Figure 12. SLL Test cases

To test the OR operation, the Testbench contained the following Generic and Edge cases to ensure maximum functionality. Figure 13 shows the successful test cases for the SRL Operation.

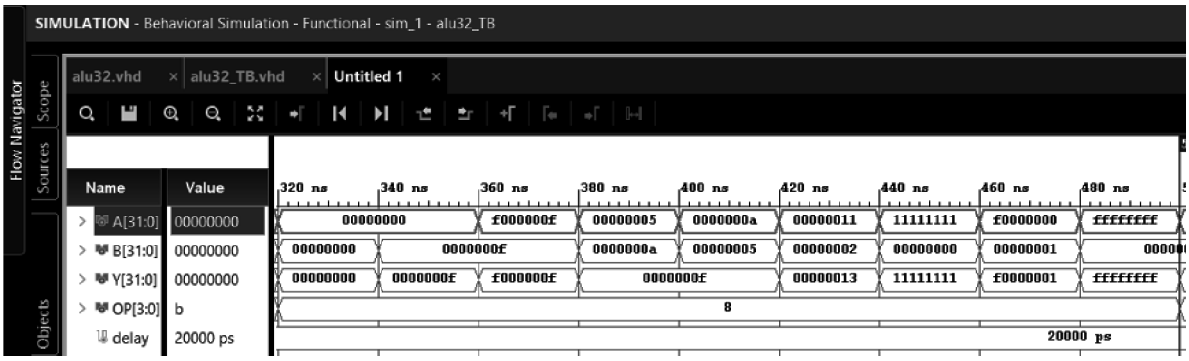

Figure 13. OR Test cases

To test the XOR operation, the Testbench contained the following Generic and Edge cases to ensure maximum functionality. Figure 14 shows the successful test cases for the SRL Operation

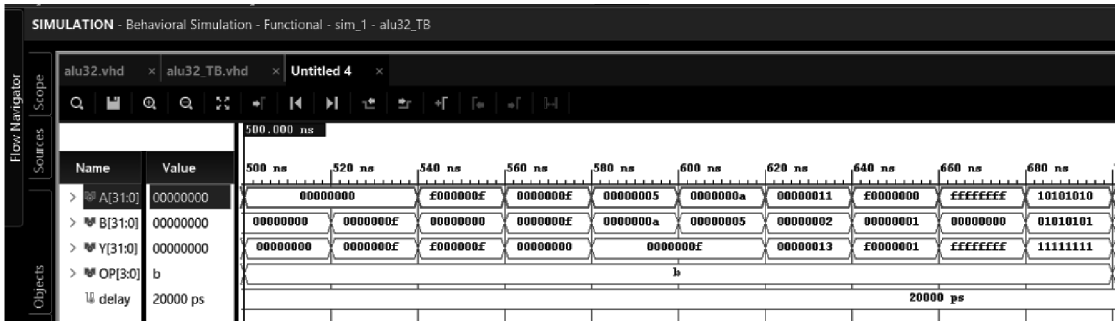

Figure 14. XOR Test cases

To test the AND operation, the Testbench contained the following Generic and Edge cases to ensure maximum functionality. Figure 15 shows the successful test cases for the SRL Operation

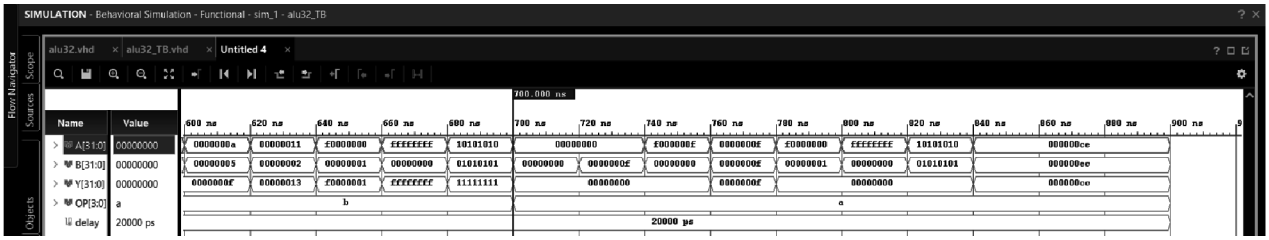

Figure 15. AND Test cases

The Post Implementation Timing Simulation shows a Red Line at the beginning of the output signal, and then each of the output is not perfectly matched with its Inputs. This is not an error and is expected in any hardware. To counteract this, the instructions need to be written in a way that accommodates the time delay between different operations.

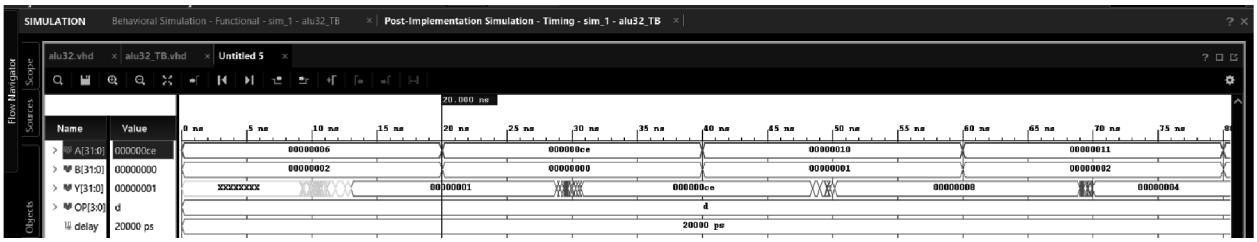

Figure 16. Error in Post Implementation.

Conclusion

The exercise was successful in understanding the Xilinx Vivado environment and the various tools it offers. The tools were taught through the creation of a 4 Bit ALU and then a 32 bit ALU was designed and tested to perform six operations— SLL, SRL, SRA, OR, XOR, AND. The ALU proved to be successful, and the operations display the anticipated results.