**Lab 3 – Designing an Arithmetic Logic Unit (ALU)**

CECS 341 – Computer Architecture & Organization

Kenry Yu, 028210726

Olena Bilinska, 028897191

Garrett Towner, 028303091

Professor: Mandy He



California State University, Long Beach

College of Engineering

1250 Bellflower Blvd, Long Beach, CA 90840

February 24, 2022

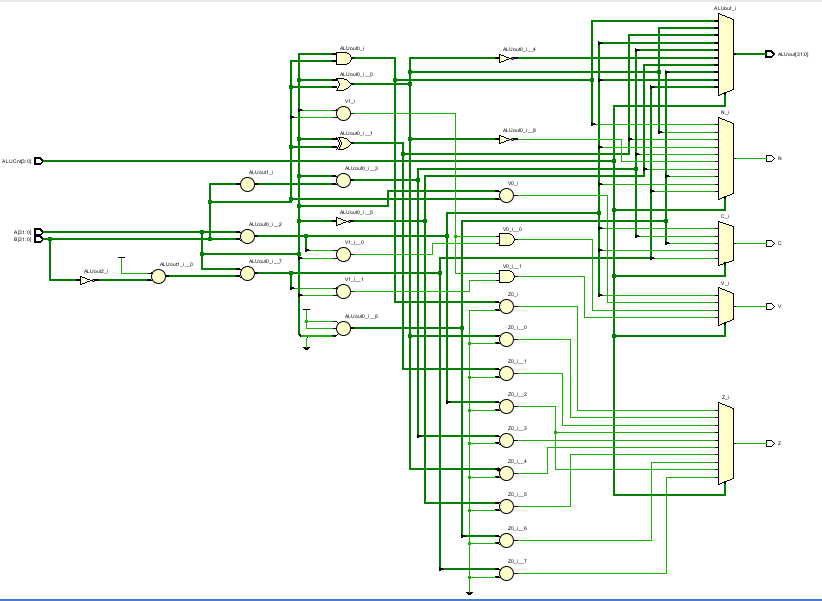
**Goal/Objective:**

The goal of this lab is to design an Arithmetic Logic Unit (ALU) using Verilog. In particular, the ALU is defined using behavioral modeling.

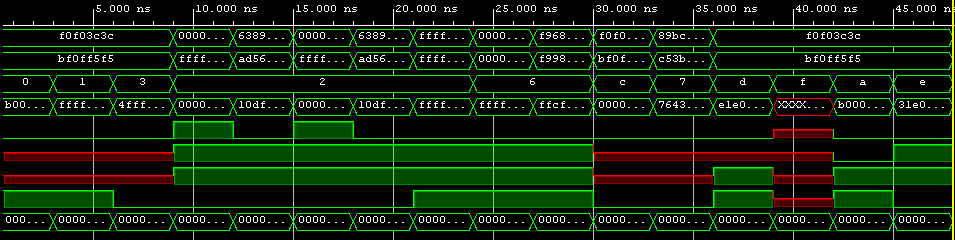
**Technical Description/Steps:**

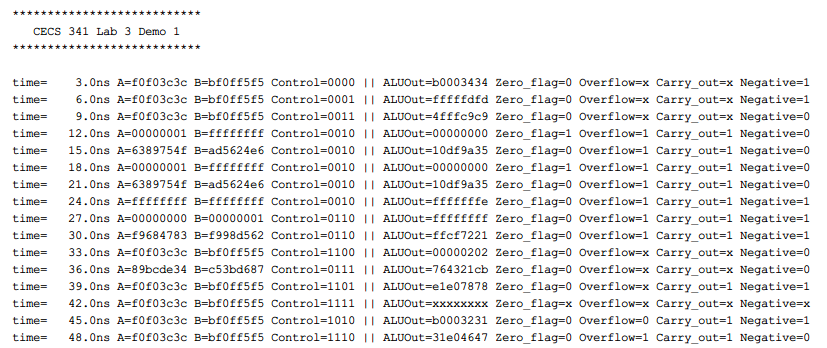
Describe the technical details of this project: what are you doing, what technology are you using, what steps did you take, etc.

1. Before beginning to write out the Verilog code in Vivado, we examined the difference and requirements of each different operator. Especially the value of overflow and carry out for the addition and subtraction.
2. After examining each operator, we wrote out the Verilog code and assigned Z, V, C, and N to zero, overflow, carry out, and negative. We also found the optimal way to compute each operator's “ALUOut” value.
3. After finishing the “ALU.v” file in Vivado, we fixed the problem in our signed subtraction operator. Initially, we didn’t implement the 2’s complement method in our code. The operator computed desired output after we fixed it.
4. We built a test bench according to the instruction document to see how the ALU module functions. We used looping and case statements to simplify running multiple test cases in the test bench.

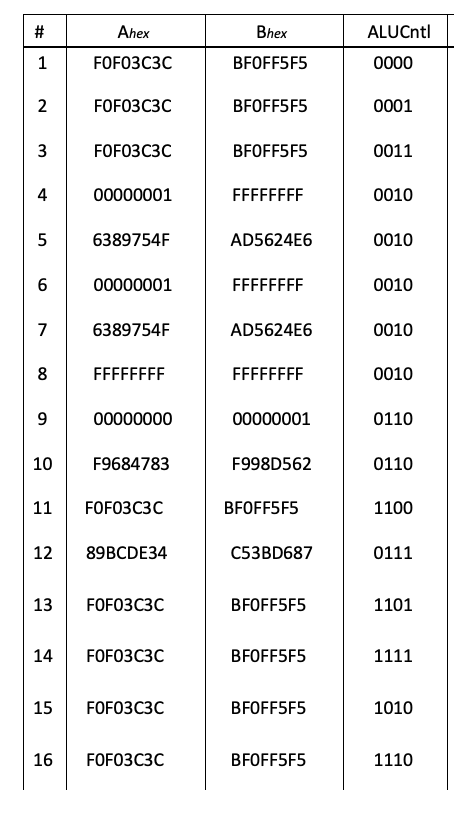


**Results:**

****

****

In the screen capture of the console log, the inputs are shown on the left while all the outputs, namely output, zero, overflow, carry out, and negative, are shown on the right. After implementing the 2’s complement in the signed subtraction, the result looked promising. All of the test cases are outputting the desired results. The overflow and carry out is shown as “x” when addition or subtraction is not the operator in the test cases.



| ALUOut | Zero | Overflow | Carry out | Negative |
| --- | --- | --- | --- | --- |
| b0003434 | 0 | x | x | 1 |
| fffffdfd | 0 | x | x | 1 |
| 4fffc9c9 | 0 | x | x | 0 |
| 00000000 | 1 | 1 | 1 | 0 |
| 10df9a35 | 0 | 1 | 1 | 0 |
| 00000000 | 1 | 1 | 1 | 0 |
| 10df9a35 | 0 | 1 | 1 | 0 |
| fffffffe | 0 | 1 | 1 | 1 |
| ffffffff | 0 | 1 | 1 | 1 |
| ffcf7221 | 0 | 1 | 1 | 1 |
| 00000202 | 0 | x | x | 0 |
| 764321cb | 0 | x | x | 0 |
| ele07878 | 0 | x | 1 | 1 |
| xxxxxxxx | x | x | x | x |
| b0003231 | 0 | 0 | 1 | 1 |
| 31e04647 | 0 | 1 | 1 | 0 |

**Conclusion:**

Our group learned how to write Verilog code using behavioral modeling. We also learned to implement binary addition, subtraction, bitwise AND, OR, NOR, and XOR operations. We used the Xilinx Vivado Design Suite to write the Verilog design and test bench code.

For the most part, the design of this ALU was straightforward since the overall design and format of this circuit had already been set. As for the design of the circuit, we had nothing to decide on our own, but when it came to testing the ALU, it took some time to fully understand how it works and how to make it work. Since there are many inputs in this circuit, the user must know what they do and refer to. Figuring this out is the hardest part of the lab and the most important.

When testing the circuit for the first time, some inputs did not produce the expected results. Although our circuit wiring was correct, our error came from not fully knowing how an ALU worked. This error allowed us to solve it and find out what we did wrong, thus increasing our experience with ALU. It also shows that user errors are possible despite correct circuit wiring.