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

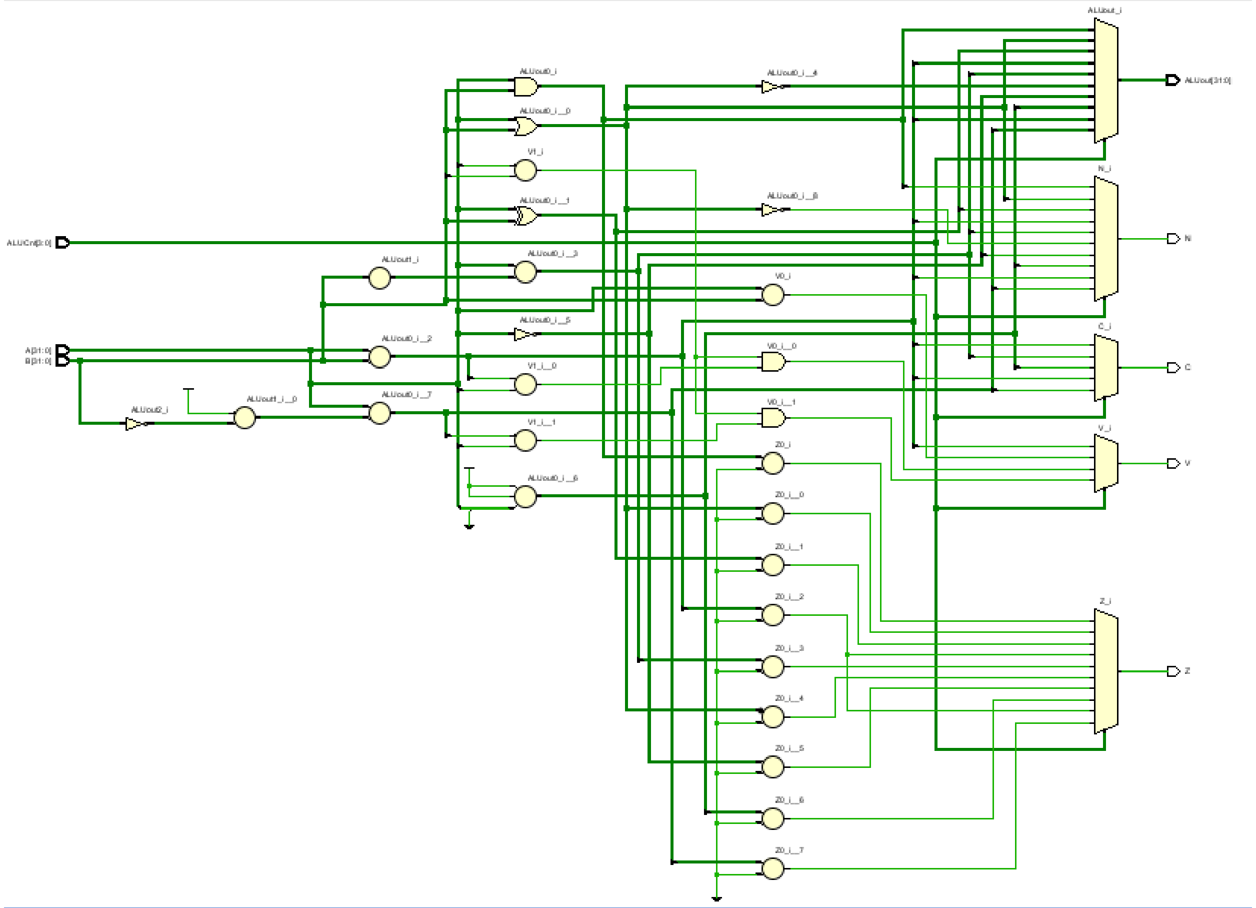
Goal/Objective:

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

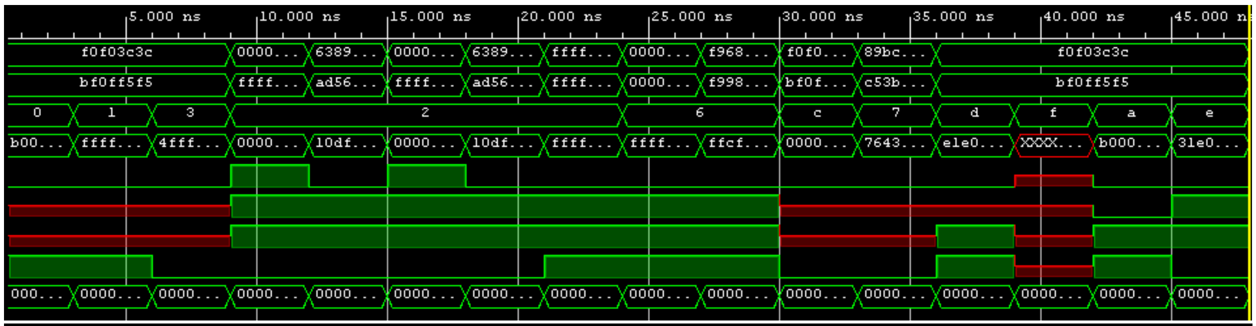
Technical Description/Steps:

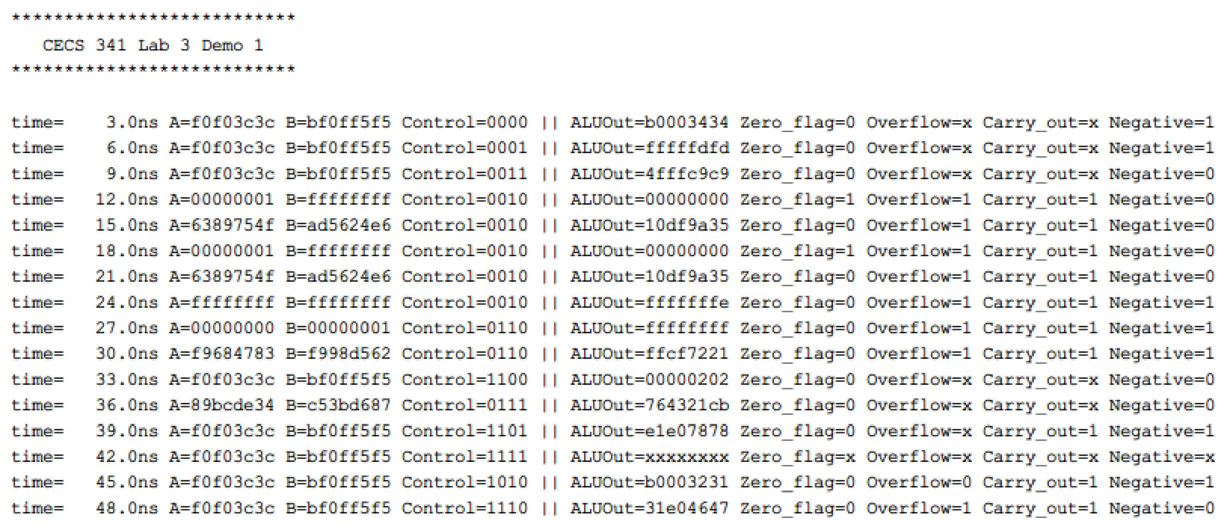
Describe the technical details of this project: what are you doing, what technology are you using, what steps did you take, etc.

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.


ALUout_i
ALUout[31:0]
N_i
N
C_i
C
V_i
V
Z_i
Z
ALUCntl[3:0]
A[31:0]
B[31:0]
ALUout0_i
ALUout1_i
ALUout2_i
V1_i
V0_i
Z0_i

Results:

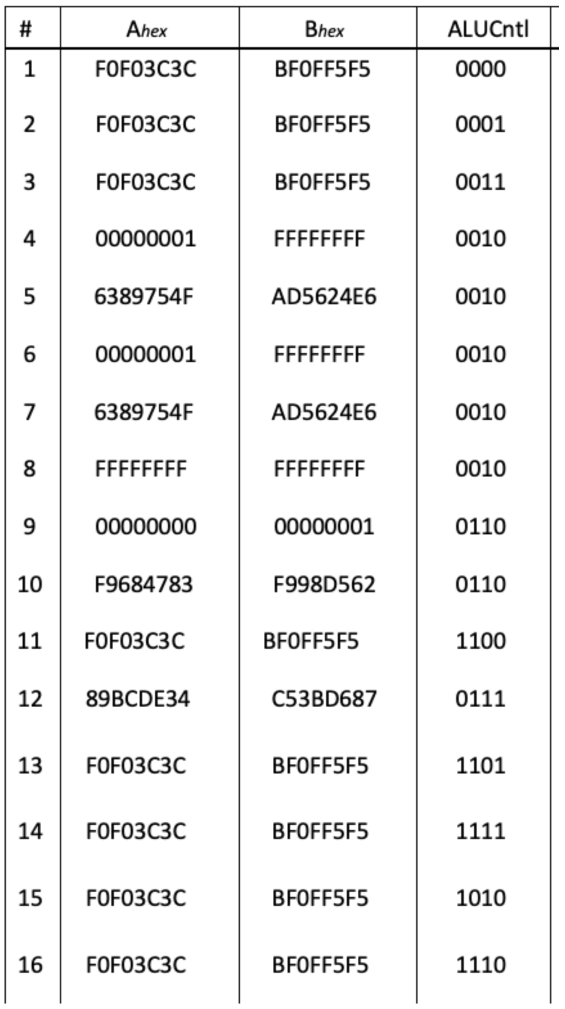

5.000 ns
10.000 ns
15.000 ns
20.000 ns
25.000 ns
30.000 ns
35.000 ns
40.000 ns
45.000 ns
f0f03c3c
0000...
6389...
ffff...
f968...
f0f0...
89bc...
bf0ff5f5
ad56...
f998...
bf0f...
c53b...
0
1
3
2
6
c
7
d
f
a
e
b00...
4fff...
10df...
ffcf...
7643...
e1e0...
XXXX...
b000...
31e0...
000...



```
***************************
   CECS 341 Lab 3 Demo 1
***************************

time=    3.0ns A=f0f03c3c B=bf0ff5f5 Control=0000 || ALUOut=b0003434 Zero_flag=0 Overflow=x Carry_out=x Negative=1
time=    6.0ns A=f0f03c3c B=bf0ff5f5 Control=0001 || ALUOut=fffffdfd Zero_flag=0 Overflow=x Carry_out=x Negative=1
time=    9.0ns A=f0f03c3c B=bf0ff5f5 Control=0011 || ALUOut=4fffc9c9 Zero_flag=0 Overflow=x Carry_out=x Negative=0
time=   12.0ns A=00000001 B=ffffffff Control=0010 || ALUOut=00000000 Zero_flag=1 Overflow=1 Carry_out=1 Negative=0
time=   15.0ns A=6389754f B=ad5624e6 Control=0010 || ALUOut=10df9a35 Zero_flag=0 Overflow=1 Carry_out=1 Negative=0
time=   18.0ns A=00000001 B=ffffffff Control=0010 || ALUOut=00000000 Zero_flag=1 Overflow=1 Carry_out=1 Negative=0
time=   21.0ns A=6389754f B=ad5624e6 Control=0010 || ALUOut=10df9a35 Zero_flag=0 Overflow=1 Carry_out=1 Negative=0
time=   24.0ns A=ffffffff B=ffffffff Control=0010 || ALUOut=fffffffe Zero_flag=0 Overflow=1 Carry_out=1 Negative=1
time=   27.0ns A=00000000 B=00000001 Control=0110 || ALUOut=ffffffff Zero_flag=0 Overflow=1 Carry_out=1 Negative=1
time=   30.0ns A=f9684783 B=f998d562 Control=0110 || ALUOut=ffcf7221 Zero_flag=0 Overflow=1 Carry_out=1 Negative=1
time=   33.0ns A=f0f03c3c B=bf0ff5f5 Control=1100 || ALUOut=00000202 Zero_flag=0 Overflow=x Carry_out=x Negative=0
time=   36.0ns A=89bcde34 B=c53bd687 Control=0111 || ALUOut=764321cb Zero_flag=0 Overflow=x Carry_out=x Negative=0
time=   39.0ns A=f0f03c3c B=bf0ff5f5 Control=1101 || ALUOut=e1e07878 Zero_flag=0 Overflow=x Carry_out=1 Negative=1
time=   42.0ns A=f0f03c3c B=bf0ff5f5 Control=1111 || ALUOut=xxxxxxxx Zero_flag=x Overflow=x Carry_out=x Negative=x
time=   45.0ns A=f0f03c3c B=bf0ff5f5 Control=1010 || ALUOut=b0003231 Zero_flag=0 Overflow=0 Carry_out=1 Negative=1
time=   48.0ns A=f0f03c3c B=bf0ff5f5 Control=1110 || ALUOut=31e04647 Zero_flag=0 Overflow=1 Carry_out=1 Negative=0
```

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.



#	A_{hex}	B_{hex}	ALUCntl
1	F0F03C3C	BF0FF5F5	0000
2	F0F03C3C	BF0FF5F5	0001
3	F0F03C3C	BF0FF5F5	0011
4	00000001	FFFFFFFF	0010
5	6389754F	AD5624E6	0010
6	00000001	FFFFFFFF	0010
7	6389754F	AD5624E6	0010
8	FFFFFFFF	FFFFFFFF	0010
9	00000000	00000001	0110
10	F9684783	F998D562	0110
11	F0F03C3C	BF0FF5F5	1100
12	89BCDE34	C53BD687	0111
13	F0F03C3C	BF0FF5F5	1101
14	F0F03C3C	BF0FF5F5	1111
15	F0F03C3C	BF0FF5F5	1010
16	F0F03C3C	BF0FF5F5	1110

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.