Lab 3: Designing Arithmetic and Logic Unit (ALU)

Twan Tran
Brian Thi
Jae Bum Jang
CECS 341 - Computer Architect Organization
December 11, 2022

Lab objective:

The objective of this lab is to design an Arithmetic and Logic Unit (ALU) using VHDL. The ALU will be defined using the behavioral VHDL model and then it will be converted into a block design using the IP integrator tool. This will allow us to test the functionality of the ALU and ensure that it performs the necessary arithmetic and logical operations.

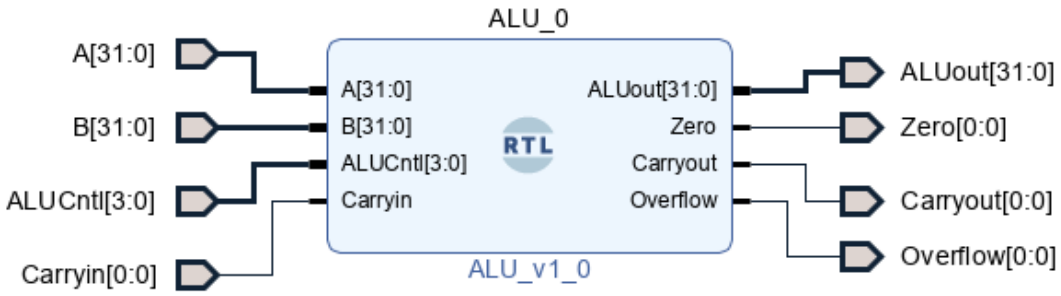
Design Approaches:

This our approach to this VHDL code design:

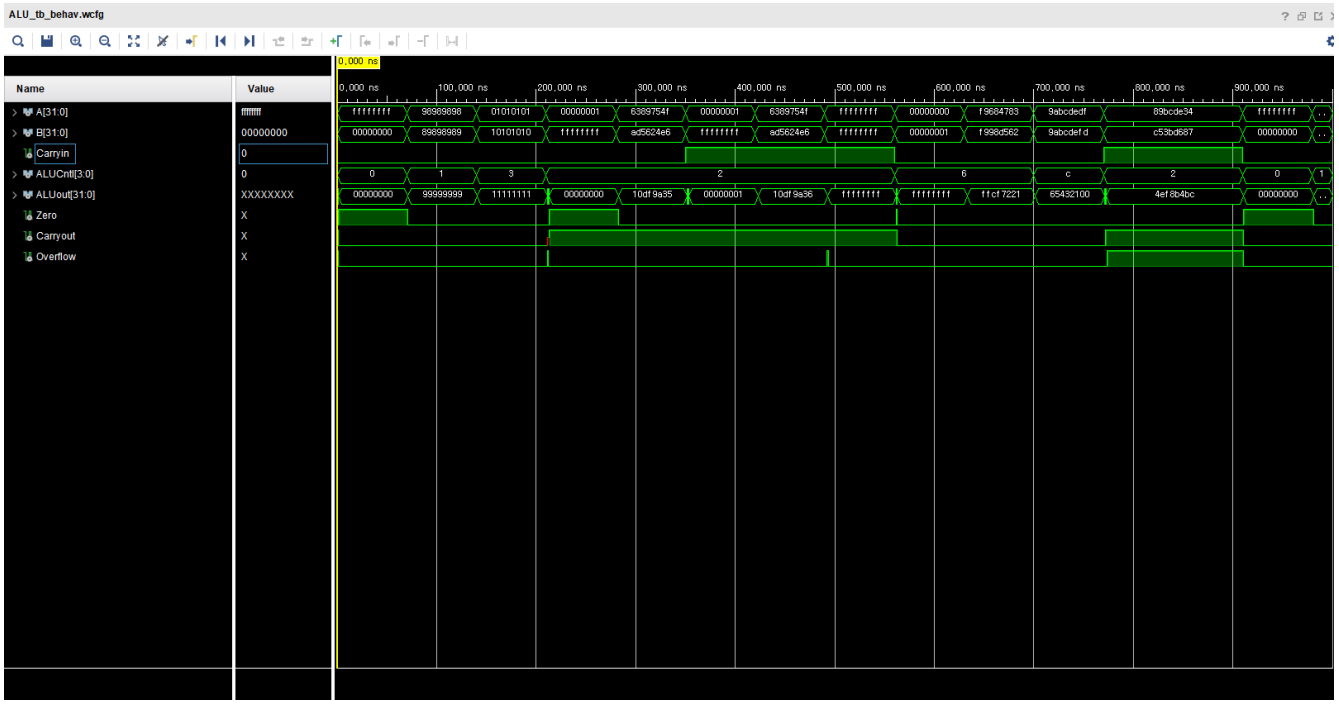
1. First, we will define the ALU component in VHDL using the behavioral model. This will include defining the input and output ports, as well as the internal signals and registers that the ALU will use to perform its operations.
2. Next, we will implement the operations specified in Table 1 using a combination of conditional statements and arithmetic/logical operations. This will allow the ALU to perform the necessary calculations based on the given control signals and input data.
3. We will then test the functionality of the ALU by providing a set of test vectors and verifying that the output is correct for each operation. This will help us ensure that the ALU is performing the operations correctly and producing the expected results.
4. Finally, we will convert the ALU into a block design using the IP integrator tool. This will allow us to easily incorporate the ALU into larger designs and test its performance in a more realistic environment.

Overall, our goal is to create a robust and efficient ALU that is capable of performing the operations specified in Table 1 accurately and reliably.

Block Design:



Simulation Waveform:



Testing the Design:

#	Ahex	Bhex	Carryin	ALUCntl	ALUouthex	Zero	Overflow	Carryout
1	FFFFFFFF	00000000	-	0000(&)	00000000	1	0	0
2	98989898	89898989	-	0001(/)	99999999	0	0	0
3	01010101	10101010	-	0011(x/)	11111111	0	0	0
4	00000001	FFFFFFFF	0	0010(+)	00000000	1	0	1
5	6389754F	AD5624E6	0	0010	10DF9A35	0	0	1
6	0000001	FFFFFFFF	1	0010	00000001	0	0	1
7	6389754F	AD5624E6	1	0010	10DF9A36	0	0	1
8	FFFFFFFF	FFFFFFFF	1	0010	FFFFFFFF	0	0	1
9	00000000	00000001	-	0110(-)	FFFFFFFF	0	0	0
10	F9684783	F998D562	-	0110	FFCF7221	0	0	0
11	9ABCDEDF	9ABCDEFD	-	1100(N/)	65432100	0	0	0
12	89BCDE34	C53BD687	1	0010	4EF8B4BC	0	1	1

Table 2: Test Vectors for testing the design

Calculation of Table 2:

Test Case 1: ALUCntl → A and B

A*hex*	**B*hex***	**Carryin**	**ALUCntl**	**ALUout*hex***	**Zero**	**Overflow**	**Carryout**
FFFFFFFF	00000000	-	0000	00000000	1	0	0

Convert hex to bin and then perform AND operation.
ALUCntl = 0000 that is AND operation from the Table -1
A: FFFFFFFF
A: 1111 1111 1111 1111 1111 1111 1111 1111
B: 00000000
B: 0000 0000 0000 0000 0000 0000 0000 0000

ALUout: 0000 0000 0000 0000 0000 0000 0000 0000 Or 0x00000000
Zero: 1 because ALUout is 0.

Carryout: 0 because it's a logical operation. So, it’s always 0.
Overflow: 0 because it’s a logical operation. So, it’s always 0. No arithmetic performed.

Test Case 2: ALUCntl → A or B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
98989898	89898989	-	0001	99999999	0	0	0

Convert hex to bin and then perform OR operation.
A: 98989898
A: 1001 1000 1001 1000 1001 1000 1001 1000
B: 89898989
B: 1000 1001 1000 1001 1000 1001 1000 1001

ALUout: 1001 1001 1001 1001 1001 1001 1001 1001 Or 0x99999999
Zero: 0 because ALUout is not 0.
Carryout: 0 because it's a logical operation. So, it’s always 0.
Overflow: 0 because it’s a logical operation. So, it’s always 0.

Test Case 3: ALUCntl -> A xor B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
01010101	10101010	0	0011	11111111	0	0	0

Convert hex to bin and then perform XOR operation.
A: 01010101
A: 0000 0001 0000 0001 0000 0001 0000 0001
B: 10101010
B: 0001 0000 0001 0000 0001 0000 0001 0000

ALUout: 0001 0001 0001 0001 0001 0001 0001 0001 Or 0x11111111
Zero: 0 because ALUout is not 0.
Carryout: 0 because it's a logical operation. So, it’s always 0.
Overflow: 0 because it’s a logical operation. So, it’s always 0.

Test Case 4: ALUCntl -> A + B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
00000001	FFFFFFFF	0	0010	00000000	1	0	1

Convert hex to bin and then perform addition operation.
A: 00000001
A: 0000 0000 0000 0000 0000 0000 0000 0001
B: FFFFFFFF
B: 1111 1111 1111 1111 1111 1111 1111 1111

ALUout: 0000 0000 0000 0000 0000 0000 0000 Or 0x00000000
Zero: 1
Carryout: 1
Overflow: 0

Test Case 5: ALUCntl -> A + B

A*hex*	**B*hex***	**Carryin**	**ALUCntl**	**ALUout*hex***	**Zero**	**Overflow**	**Carryout**
6389754F	AD5624E6	0	0010	10DF9A35	0	0	1

Convert hex to bin and then perform addition operation.
A: 6389754F
A: 0110 0011 1000 1001 0111 0101 0100 1111
B: AD5624E6
B: 1010 1101 0101 0110 0010 0100 1110 0110

ALUout: 0001 0000 1101 1111 1001 1010 0011 0101 Or 0x10DF9A35
Zero: 0 because ALUout is not 0.
Carryout: 1
Overflow: 0

Test Case 6: ALUCntl -> A + B

A*hex*	**B*hex***	**Carryin**	**ALUCntl**	**ALUout*hex***	**Zero**	**Overflow**	**Carryout**
00000001	FFFFFFFF	1	0010	00000001	0	0	1

Convert hex to bin and then perform addition operation.
A: 00000001
A: 0000 0000 0000 0000 0000 0000 0000 0001
B: FFFFFFFF
B: 1111 1111 1111 1111 1111 1111 1111 1111

ALUout: 0000 0000 0000 0000 0000 0000 0000 0001 Or 0x00000001
Zero: 0 because ALUout is not 0.
Carryout: 1

Overflow: 0

Test Case 7: ALUCntl -> A + B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
6389754F	AD5624E6	1	0010	10DF9A36	0	0	1

Convert hex to bin and then perform addition operation.
A: 6389754F
A: 0110 0011 1000 1001 0111 0101 0100 1111
B: AD5624E6
B: 1010 1101 0101 0110 0010 0100 1110 0110

ALUout: 0001 0000 1101 1111 1001 1010 0011 0110 Or 0x10DF9A36
Zero: 0 because ALUout is not 0.
Carryout: 1
Overflow: 0

Test Case 8: ALUCntl -> A + B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
FFFFFFFF	FFFFFFFF	1	0010	FFFFFFFF	0	0	1

Convert hex to bin and then perform addition operation.
A: FFFFFFFF
A: 1111 1111 1111 1111 1111 1111 1111 1111
B: FFFFFFFF
B: 1111 1111 1111 1111 1111 1111 1111 1111

ALUout: 1111 1111 1111 1111 1111 1111 1111 1111 Or 0xFFFFFFFF
Zero: 0
Carryout: 1
Overflow: 0

Test Case 9: ALUCntl -> A - B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
00000000	00000001	-	0110	FFFFFFFF	0	0	0

Convert hex to bin and then perform subtraction operation.
A: 00000000
A: 0000 0000 0000 0000 0000 0000 0000 0000
B: 00000001

B: 0000 0000 0000 0000 0000 0000 0000 0001

ALUout: 1111 1111 1111 1111 1111 1111 1111 1111 or 0xFFFFFFFF
Zero: 0 because ALUout is not 0.
Carryout: 0
Overflow: 0

Test Case 10: ALUCntl -> A - B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
F9684783	F998D562	-	0110	FFCF7221	0	0	0

Convert hex to bin and then perform subtraction operation.
A: F9684783
A: 1111 1001 0110 1000 0100 0111 1000 0011
B: F998D562
B: 1111 1001 1001 1000 1101 0101 0110 0010

ALUout: 1111 1111 1100 1111 0111 0010 0010 0001 or 0xFFCF7221
Zero: 0 because ALUout is not 0.
Carryout: 0
Overflow: 0

Test Case 11: ALUCntl -> A nor B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
9ABCDEDF	9ABCDEFD	-	1100	65432100	0	0	0

Convert hex to bin and then perform NOR operation.
A: 9ABCDEDF
A: 1001 1010 1011 1100 1101 1110 1101 1111
B: 9ABCDEFD
B: 1001 1010 1011 1100 1101 1110 1111 1101

ALUout: 0110 0101 0100 0011 0010 0001 0000 0000 or 0x65432100
Zero: 0 because ALUout is not 0.
Carryout: 0 because it's a logical operation. So, it’s always 0.
Overflow: 0 because it’s a logical operation. So, it’s always 0. No arithmetic performed.

Test Case 12: ALUCntl -> A + B

A*hex*	B*hex*	Carryin	ALUCntl	ALUout*hex*	Zero	Overflow	Carryout
89BCDE34	C53BD687	1	0010	4EF8B4BC	0	1	1

Convert hex to bin and then perform addition operation.
A: 89BCDE34
A: 1000 1001 1011 1100 1101 1110 0011 0100
B: C53BD687
B: 1100 0101 0011 1011 1101 0110 1000 0111

ALUout: 0100 1110 1111 1000 1011 0100 1011 1100 or 0x4EF8B4BC
Zero: 0
Carryout: 1
Overflow: 1

Conclusion:

To Conclude this LAB, we write a VHDL code and create a block design that shows a waveform that came from our Arithmetic and Logic Unit Design. We had to calculate every test case and compare the Values with our ALU design. We create a testbench that can form out the simulation waveform. We use the waveform to analyze the ALU and the result output. By observing the waveform, we can confirm that our own calculations are correct and that our ALU is working accordingly.