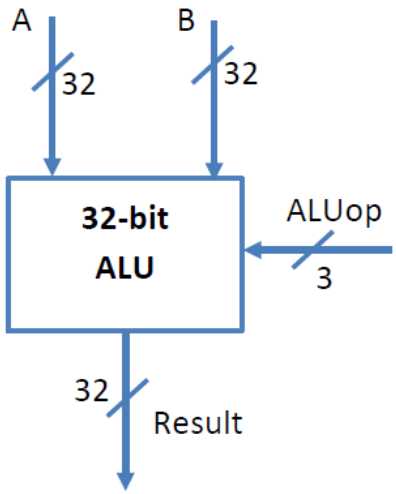
GTU Department of Computer Engineering
CSE 331 - Spring 2022
Homework 2 Report

Emirkan Burak Yılmaz
1901042659

1. ALU Design

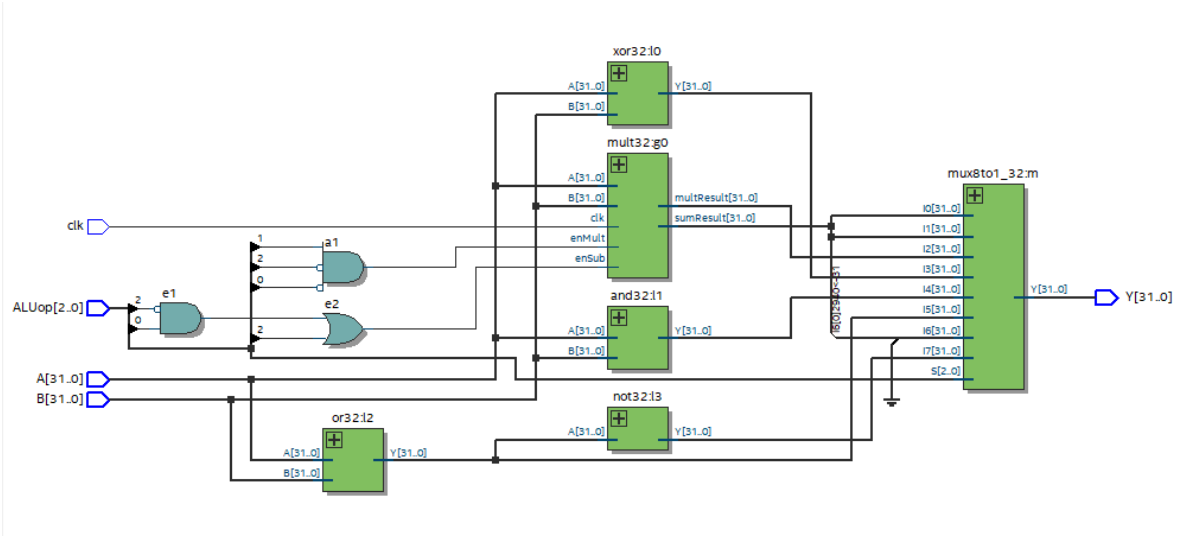


ALUop	Operation	enSUB	enMULT
000	ADD	0	0
001	SUB	1	0
010	MULT	0	1
011	XOR	x	0
100	AND	x	0
101	OR	x	0
110	SLT	1	0
111	NOR	x	0

- This specific ALU has only 1 adder and it's used for addition, subtraction, and multiplication. For that reason, usage of adder depends on the two-input signal **enSUB** and **enMULT.**
- For substraction the **enSUB** signals needs to be set 1 and for the multiplication **enMULT** signal needs to be set 1.

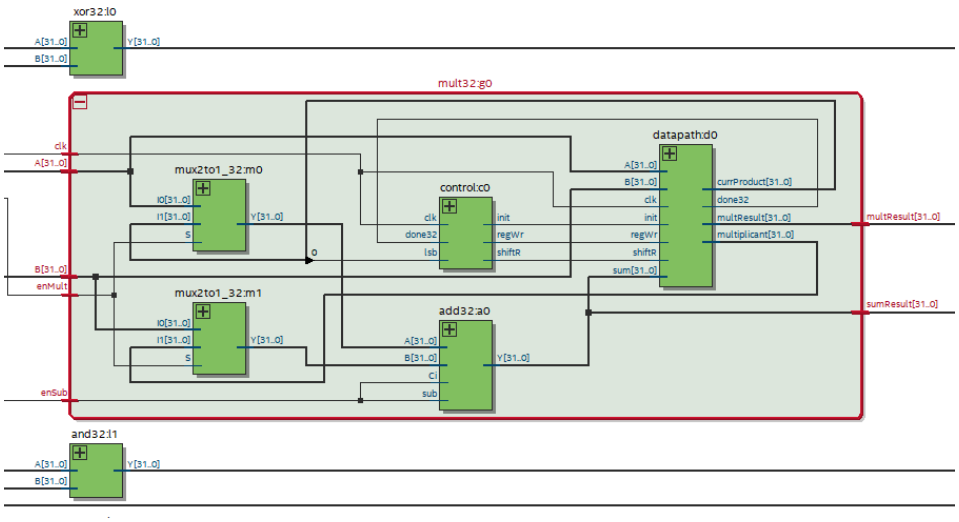
enSUB = $ALUop_2$ **|** (**~**$ALUop_2$ **&** $ALUop_0$)

enMULT = **~**$ALUop_2$ **&** $ALUop_1$ **&** **~**$ALUop_0$



alu32 module

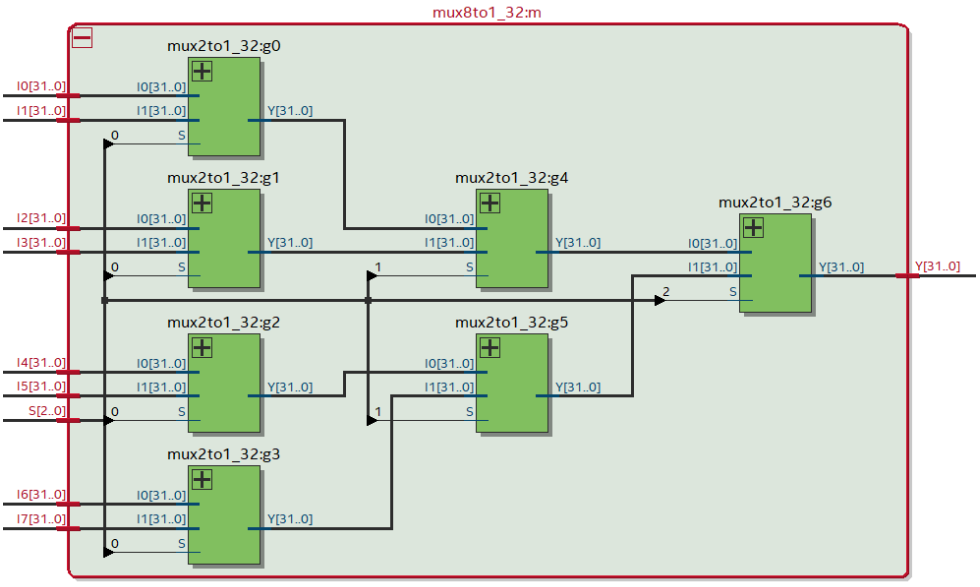
- The **mult32** module contains the **add32** module which is 32-bit full adder that supports subtraction as well as addition. The **add32** module is used for addition, subtraction, and multiplication operation. Those three different operations discriminated with the control signals **enMult** for multiplication and **enSub** for subtraction.
- For SLT operation is add32 used as subtractor and the most significant bit of the result is written to least significant bit of the SLT result.
- The other operations OR, AND XOR, NOR are simply implemented using 32-bit primitive gates and the new modules **and32**, **or32**, **xor32**, **not32** are used in the design.



mult32 module

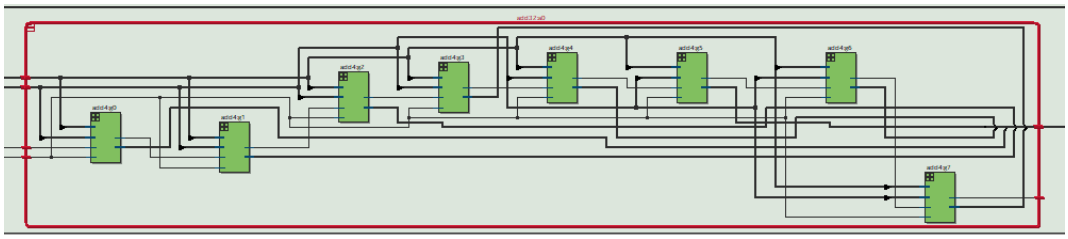
- According to **ALUop,** proper output should be directed as output. For this 32-bit 8x1 multiplexer module **mux8to1_32** is designed and used.

- The **mult32** module uses **add32** to multiplication. If the operation is multiplication, then the add32 inputs are becomes most significant 32 bit of product register and the multiplier. However, if the operation is addition or subtraction than the given input values A and B should be processed. To support this functionality, inputs of **add32** module are passed through 2 32-bit 2x1 multiplexer.



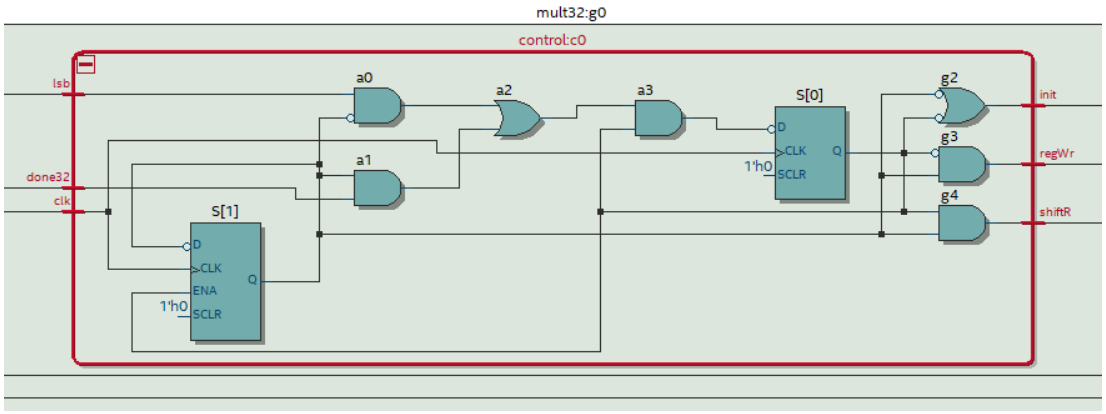
mux_8to1_32 module

- Multiplication can be design as a combinational circuit, but the assignment instructions are requiring a sequential circuit named add-shift multiplier. For that reason, **control** and the **datapath** modules are designed and used inside the mult32 module.

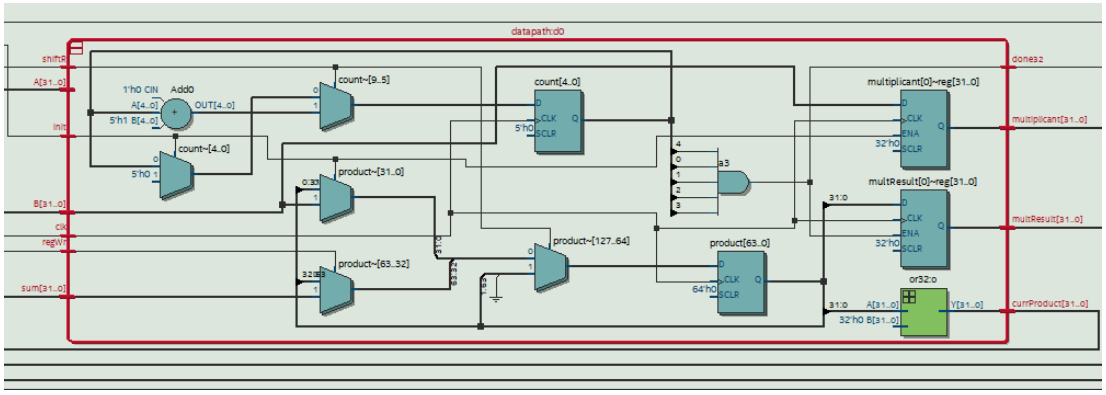


add32 module

- The **add32** module is a carry-ripple adder which uses eight **add4** module. The **add4** module is just the 4-bit version of the add32 module which consist of four **add1** modules. The add1 module is just 1-bit full-adder which consist of designed with two half-adder.

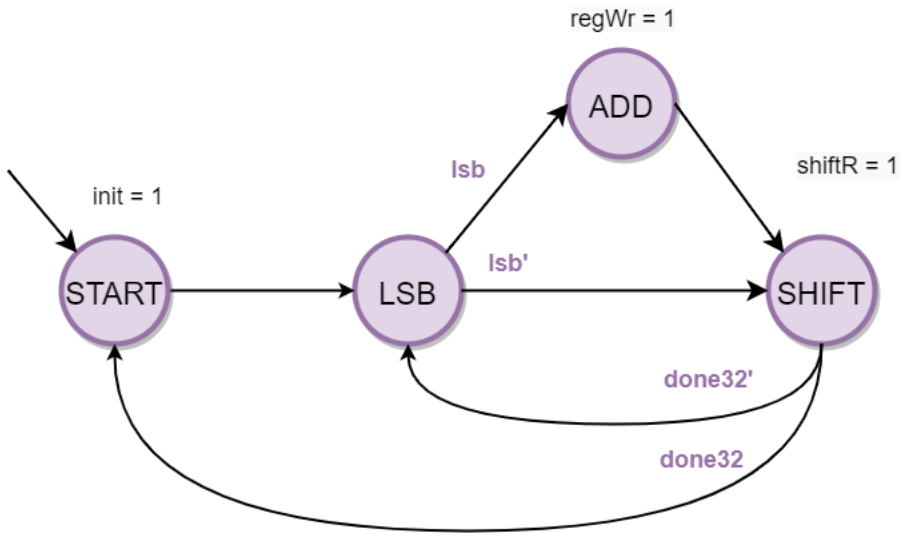


control module



datapath module

2. **Multiplier Design**
   a. **Control Part**
      i. **State Diagram**



Signal	Meaning
lsb	Least significant bit of product register
done32	Is multiplication process done
Init	Initializes the registers content
regWr	Register write
shiftR	One right shift for product register

States

1. **START:** Starting position of multiplication. Initial register assignments are done at this stage. Least 32 bit of product register takes the multiplier and the most significant 32 bit of product takes 0. The Multiplier register takes the multiplier value. And finally, the counter is set to 0 to keep track of the number of cycles that are made on multiplication process.

2. **LSB:** On this stage the least significant bit of product register checked with the input signal lsb. If it is 1 then the next stage becomes ADD. Otherwise, the next stage becomes SHIFT.

3. **ADD:** Summation of the most significant 32 bit of product register and the multiplier is written to the most significant 32 bit of the product register.

4. **SHIFT:** Product register shifts left by 1 and the counter value is incremented by one. If the input signal done32 is 1, which indicates the counter becomes 31, then the multiplication process is finished, and the next stage becomes START to write the result. Otherwise, the next stage becomes LSB and the multiplication process continues.

ii. State Table

State	s_1	s_0	lsb	done32	n_1	n_0	init	regWr	shiftR
start	0	0	x	x	0	1	1	0	0
lsb	0	1	0	x	1	1	0	0	0
	0	1	1	x	1	0	0	0	0
add	1	0	x	x	1	1	0	1	0
shift	1	1	x	0	0	1	0	0	1
	1	1	x	1	0	0	0	0	1

iii. Boolean Expressions

n_1 = (s_1 **&** **~**s_0) **|** (**~**s_1 **&** s_0)

= s_1 **^** s_0

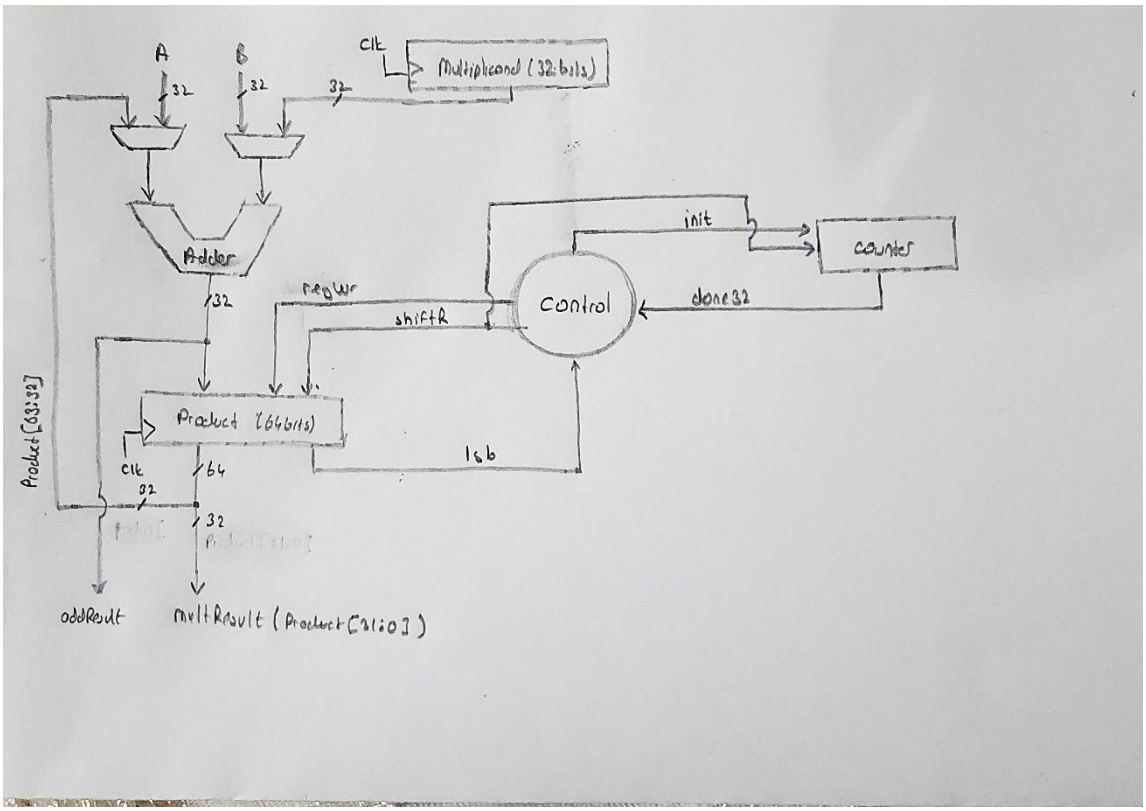
n_0 = **~**(s_0 **&** ((**~**s_1 **&** lsb) **|** (s_1 **&** done32)))

init = **~**s_1 **&** **~**s_0

regWr = s_1 **&** **~**s_0

shiftR = s_1 **&** s_0

b. Datapath Design



3. Tests

alu32_tb

```
VSIM 3> run -all
# time =     0, ALUop = 000, A = 00000000000000000000000000001101, B = 00000000000000000000000000001100, result = 00000000000000000000000000011001
# time =  1250, ALUop = 000, A = 10000000000000000000000000001101, B = 00000000000000000000000000001011, result = 10000000000000000000000000001101001
# time =  2500, ALUop = 001, A = 00000000000000000000000000101101, B = 00000000000000000000000000001111, result = 00000000000000000000000000011110
# time =  3750, ALUop = 001, A = 10000000000000000000000000101101, B = 00000000000000000000000000001111, result = 10000000000000000000000000011110
# time =  5000, ALUop = 010, A = 00000000000000000000000000000011, B = 00000000000000000000000000000010, result = 00000000000000000000000000000000
# time =  6250, ALUop = 010, A = 00000000000000000000000000001111, B = 00000000000000000000000000000101, result = 00000000000000000000000000000000
# time =  7500, ALUop = 011, A = 00000010000000000000000000001101, B = 00000010000000000000000000001100, result = 00000000000000000000000000000001
# time =  8750, ALUop = 100, A = 00000010000000000000000000001101, B = 00000010000000000000000000001100, result = 00000010000000000000000000001100
# time = 10000, ALUop = 101, A = 00000010000000000000000000001101, B = 00000010000000000000000000001100, result = 00000010000000000000000000001101
# time = 11250, ALUop = 110, A = 00000010000000000000000000001111, B = 00000010000000000000000000001100, result = 00000000000000000000000000000000
# time = 12500, ALUop = 110, A = 00000010000000000000000000000101, B = 00000010000000000000000000001100, result = 00000000000000000000000000000001
# time = 13750, ALUop = 111, A = 00000010000000000000000000001101, B = 00000010000000000000000000001100, result = 11111101111111111111111111110010
# ** Note: $finish    : D:/intelFPGA_lite/18.1/workspace/alu32/alu32_tb.v(59)
#    Time: 15 ns  Iteration: 0  Instance: /alu32_tb
```

add32_tb

```
VSIM 11> run -all
# time =    0, A =   13, B =   12, enSub = 0, carry-out = 0, result =   25
# time = 2500, A =  -15, B =   12, enSub = 0, carry-out = 0, result =   -3
# time = 3750, A =   13, B =   12, enSub = 1, carry-out = 1, result =    1
# time = 5000, A =   -7, B =   12, enSub = 1, carry-out = 1, result =  -19
# time = 6250, A =   13, B =  -31, enSub = 1, carry-out = 0, result =   44
# ** Note: $finish    : D:/intelFPGA_lite/18.1/workspace/alu32/add32_tb.v(29)
#    Time: 7500 ps  Iteration: 0  Instance: /add32_tb
```

full_adder_tb

```
VSIM 27> run -all
# time =  0, A = 0, B = 0, carry-in = 0, sum = 0, carry-out = 0
# time = 20, A = 0, B = 0, carry-in = 1, sum = 1, carry-out = 0
# time = 40, A = 0, B = 1, carry-in = 0, sum = 1, carry-out = 0
# time = 60, A = 0, B = 1, carry-in = 1, sum = 0, carry-out = 1
# time = 80, A = 1, B = 0, carry-in = 0, sum = 1, carry-out = 0
# time = 100, A = 1, B = 0, carry-in = 1, sum = 0, carry-out = 1
# time = 120, A = 1, B = 1, carry-in = 0, sum = 0, carry-out = 1
# time = 140, A = 1, B = 1, carry-in = 1, sum = 1, carry-out = 1
# ** Note: $finish    : D:/intelFPGA_lite/18.1/workspace/alu32/full_adder1_tb.v(27)
#    Time: 160 ps  Iteration: 0  Instance: /full_adder1_tb
```

mult32_tb

```
VSIM 24> run -all
# time =    0, A =    3, B =    2, result =    0
# time = 2500, A =   15, B =    4, result =    0
# time = 5000, A =   25, B =    5, result =    0
# ** Note: $finish    : D:/intelFPGA_lite/18.1/workspace/alu32/mult32_tb.v(28)
#    Time: 7500 ps  Iteration: 0  Instance: /mult32_tb
```

control_tb

```
# time =   62, lsb = 1, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =   66, lsb = 1, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =   70, lsb = 1, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =   74, lsb = 1, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =   78, lsb = 1, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =   82, lsb = 1, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =   86, lsb = 1, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =   90, lsb = 1, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =   94, lsb = 1, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =   98, lsb = 1, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  102, lsb = 1, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  106, lsb = 1, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =  110, lsb = 1, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  114, lsb = 1, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  118, lsb = 1, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =  120, lsb = 0, done32 = 0, regWr = 1, shiftR = 0, init = 1
# time =  122, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  126, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  130, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  134, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  138, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  142, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  146, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  150, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  154, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  158, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  162, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  166, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  170, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  174, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  178, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  182, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  186, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  190, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  194, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  198, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  202, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  206, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  210, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  214, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  218, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  222, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  226, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  230, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# time =  234, lsb = 0, done32 = 0, regWr = 0, shiftR = 1, init = 0
# time =  238, lsb = 0, done32 = 0, regWr = 0, shiftR = 0, init = 1
# ** Note: $finish    : D:/intelFPGA_lite/18.1/workspace/alu32/control_tb.v(26)
#    Time: 240 ps  Iteration: 0  Instance: /control_tb
```

Note: ALU accomplishes all the operations except multiplication. I try a lot to find the bug, however, I couldn't. If I cannot solve this issue after more effort, I am planning to talk with the TA or the professor about the problem.