

CSE 331 HW3

REPORT

EMRE SEZER

1901042640

PART 1:

Datapath:

Gets inputs from control unit and returns the outputs.

If write input is 1 : it adds multiplicand to the first 32 bits of the product.

If write input is 0 : it adds 0 to the first 32 bits of the product.

For if statements I use mux2_32 module. It works like a 2x1 multiplexer for 32 bit numbers.

With that approach multiplicand addition on the pdf only happens when control unit returns write = 1.

If shift input is 1 : it shifts the product right by 1 bit.

If shift input is 0: it doesn't shift.

For if statements I use mux2_64 module. It works like a 2x1 multiplexer for 64 bit numbers.

With that approach multiplicand shift operation on the pdf only happens when control unit returns shift = 1.

If shift input is 1 : counter increases by 1. It adds 1 to the counter using adder.v .

If shift input is 0: counter stays the same. It adds 0 to the counter using adder.v

For if statements I use mux2_32 module. It works like a 2x1 multiplexer for 64 bit numbers.

Output result: modified product

Output newCounter: modified counter

Output done: it is 0 if counter is less than 32. It is 1 if counter is not less than 32. I used slt.v for checking if counter is less than 32.

Control:

I created control by simply following state and output table I created.

mult32:

I added comments on code for explanation.

PART 2:

alu32/mux32:

mux32 is a 8x1 multiplexer for 32 bit numbers.

alu32 uses mux32 module to decide which operation's result to be returned. mux32 takes ALUop as selection input and result of 8 different operations as inputs. mux32 is constructed from 32 mux1's. mux1 works same as mux32 but 8 inputs are 1 bit numbers. When used 32 mux1's I was able to construct a 8x1 mux for 32 bit numbers.

Operations can be add, and, or, subtract, slt, mult, xor, nor for 32 bits.

adder:

It uses 1 bit full adder and half adders 32 times and does addition for 32 bit numbers.

xor32:

It uses 1 bit xor gates 32 times for each bit of the input 32 bit number and does xor for 2 32 bit numbers.

subtractor:

It takes 2's complement of the second input and adds that number to first input (using adder.v). For taking 2's complement of a number: First it uses not32.v (It uses 1 bit not gates 32 times for each bit of the input) and adds 1 to that number using adder.v .

slt:

It subtracts second input from the first input and returns the most significant bit of the result of that subtraction. It uses subtractor.v.

nor32:

It uses 1 bit or gates 32 times for each bit of the input 32 bit number. Then, it uses not32.v in order to get the not of the 32 bit number. In the end it returns nor of the 2 32 bit numbers.

and32:

It uses 1 bit and gates 32 times for each bit of the input 32 bit number and does and for 2 32 bit numbers.

or32:

It uses 1 bit or gates 32 times for each bit of the input 32 bit number and does or for 2 32 bit numbers.

I added testbenches for each operation I described.

!! I added #1500 delay for mult32 at alu32 testbench. In order to perform.

SCREENSHOTS

PART 1:

Datapath:

Gets inputs from control unit and returns the outputs.

If write input is 1 : it adds multiplicand to the first 32 bits of the product.

If write input is 0 : it adds 0 to the first 32 bits of the product.

For if statements I use mux2_32 module. It works like a 2x1 multiplexer for 32 bit numbers.

With that approach multiplicand addition on the pdf only happens when control unit returns write = 1.

If shift input is 1 : it shifts the product right by 1 bit.

If shift input is 0: it doesn't shift.

For if statements I use mux2_64 module. It works like a 2x1 multiplexer for 64 bit numbers.

With that approach multiplicand shift operation on the pdf only happens when control unit returns shift = 1.

If shift input is 1 : counter increases by 1. It adds 1 to the counter using adder.v .

If shift input is 0: counter stays the same. It adds 0 to the counter using adder.v

For if statements I use mux2_32 module. It works like a 2x1 multiplexer for 64 bit numbers.

Output result: modified product

Output newCounter: modified counter

Output done: it is 0 if counter is less than 32. It is 1 if counter is not less than 32. I used `slt.v` for checking if counter is less than 32.

Control:

I created control by simply following state and output table I created.

mult32:

I added comments on code for explanation.

PART 2:

alu32/mux32:

mux32 is a 8x1 multiplexer for 32 bit numbers.

alu32 uses mux32 module to decide which operation's result to be returned. mux32 takes ALUop as selection input and result of 8 different operations as inputs. mux32 is constructed from 32 mux1's. mux1 works same as mux32 but 8 inputs are 1 bit numbers. When used 32 mux1's I was able to construct a 8x1 mux for 32 bit numbers.

Operations can be add, and, or, subtract, slt, mult, xor, nor for 32 bits.

adder:

It uses 1 bit full adder and half adders 32 times and does addition for 32 bit numbers.

xor32:

It uses 1 bit xor gates 32 times for each bit of the input 32 bit number and does xor for 2 32 bit numbers.

subtractor:

It takes 2's complement of the second input and adds that number to first input (using adder.v). For taking 2's complement of a number: First it uses not32.v (It uses 1 bit not gates 32 times for each bit of the input) and adds 1 to that number using adder.v .

slt:

It subtracts second input from the first input and returns the most significant bit of the result of that subtraction. It uses subtractor.v.

nor32:

It uses 1 bit or gates 32 times for each bit of the input 32 bit number. Then, it uses not32.v in order to get the not of the 32 bit number. In the end it returns nor of the 2 32 bit numbers.

and32:

It uses 1 bit and gates 32 times for each bit of the input 32 bit number and does and for 2 32 bit numbers.

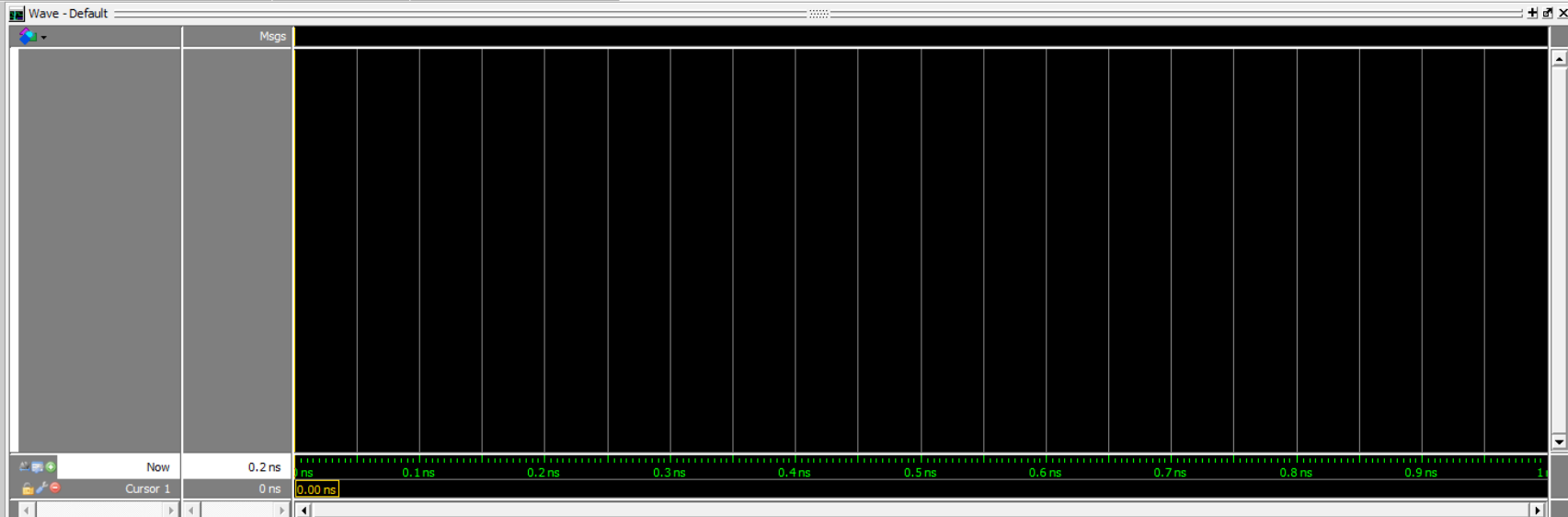
or32:

It uses 1 bit or gates 32 times for each bit of the input 32 bit number and does or for 2 32 bit numbers.

I added testbenches for each operation I described.

SCREENSHOTS

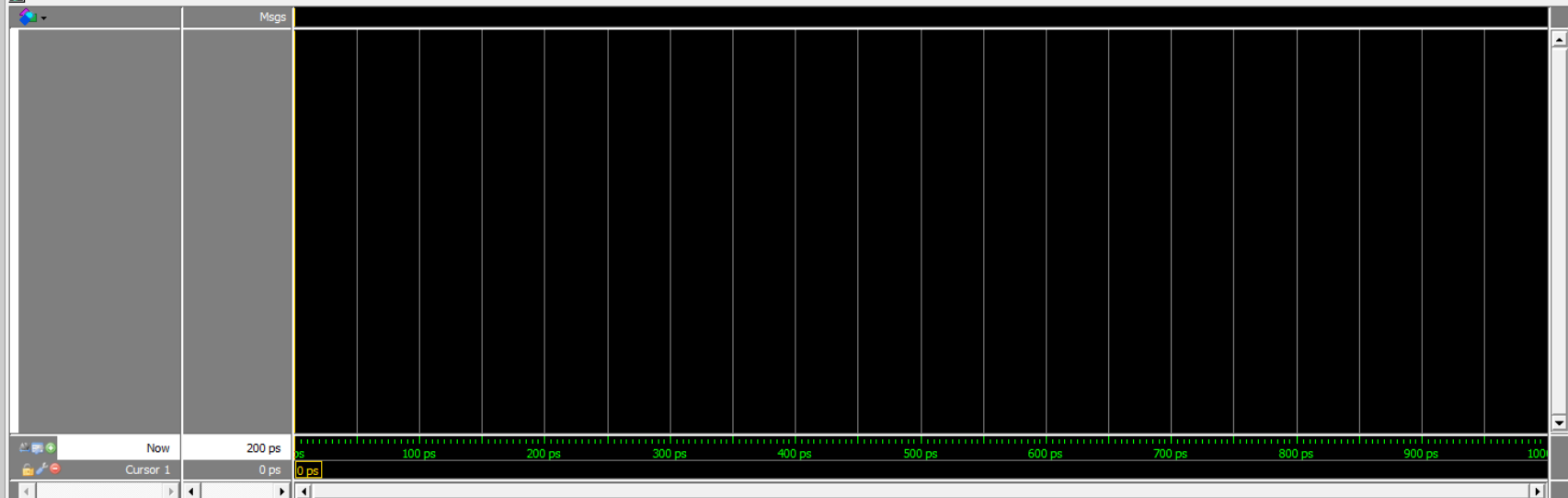
ADDER TESTBENCH

[illegible]


Msgs

[illegible][illegible]

SLT TESTBENCH

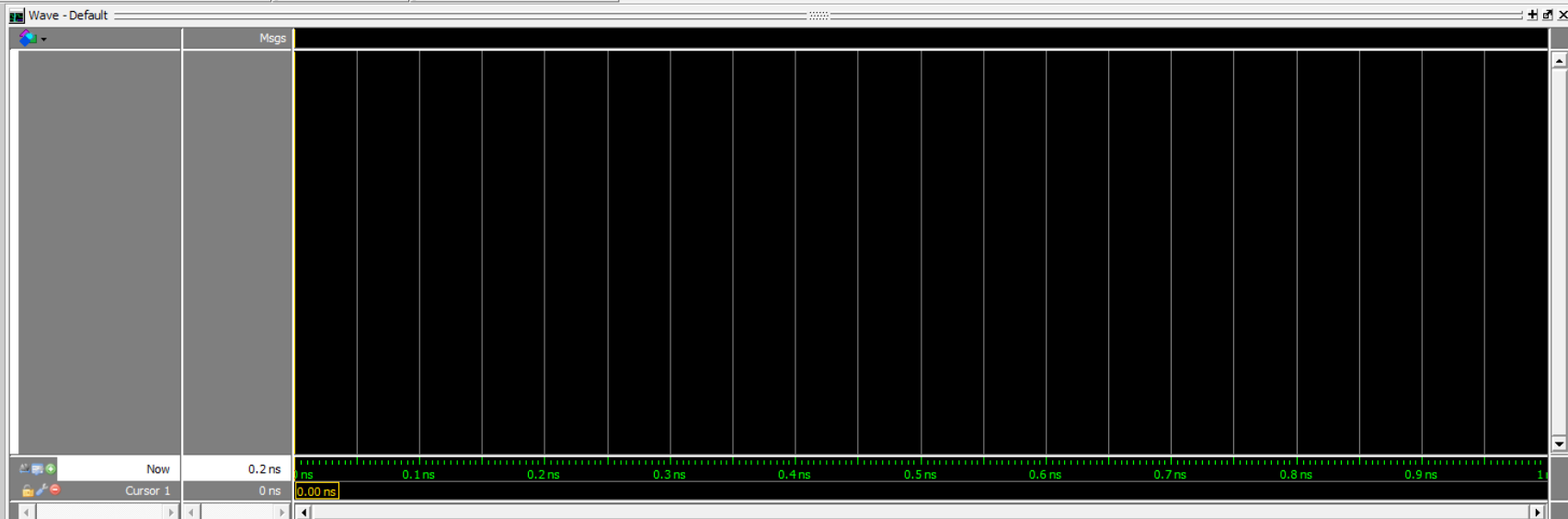


```
VSIM 3> run
```

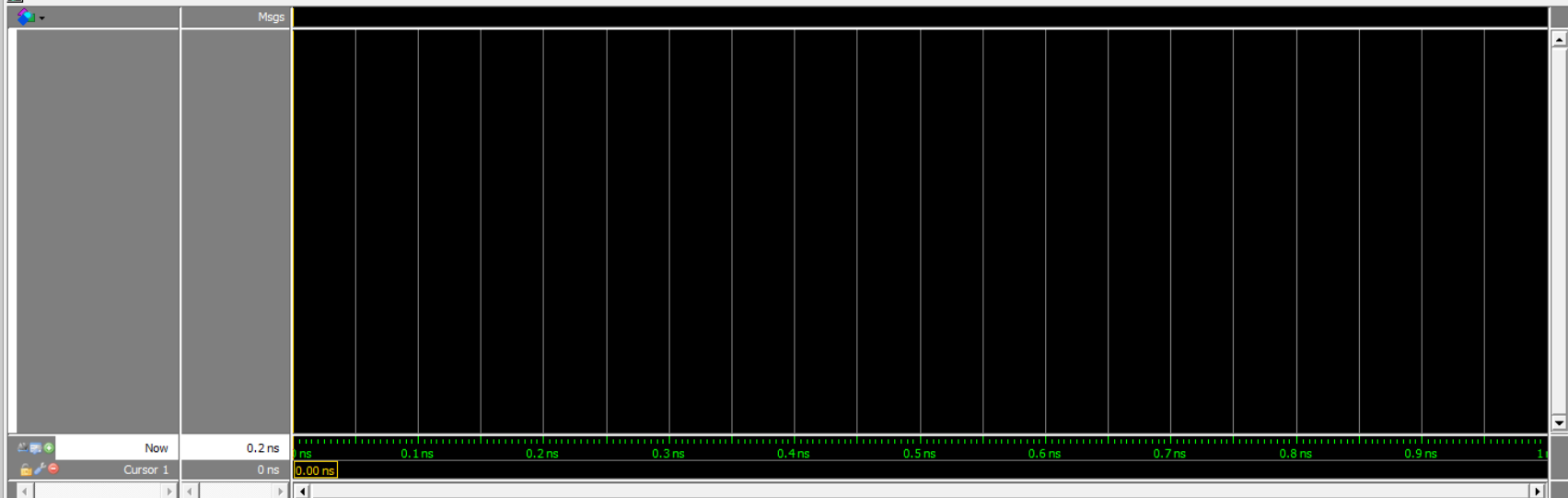
[illegible]

VSIM 4>

NOR TESTBENCH

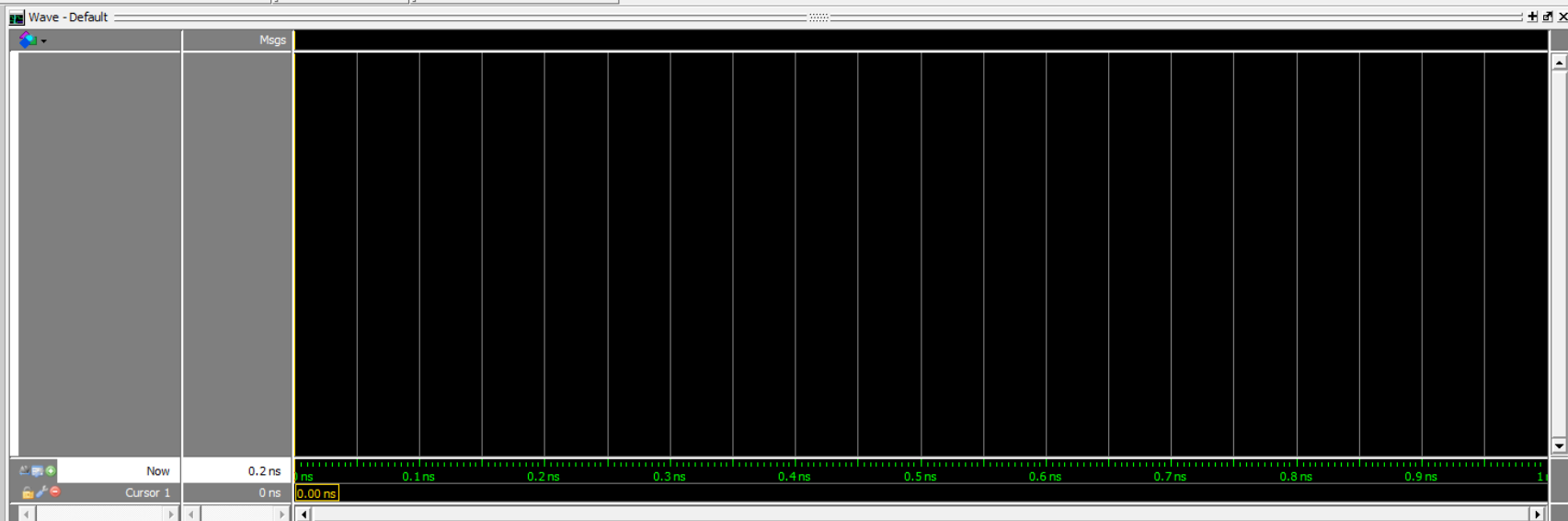
[illegible]






AND TESTBENCH

[illegible]

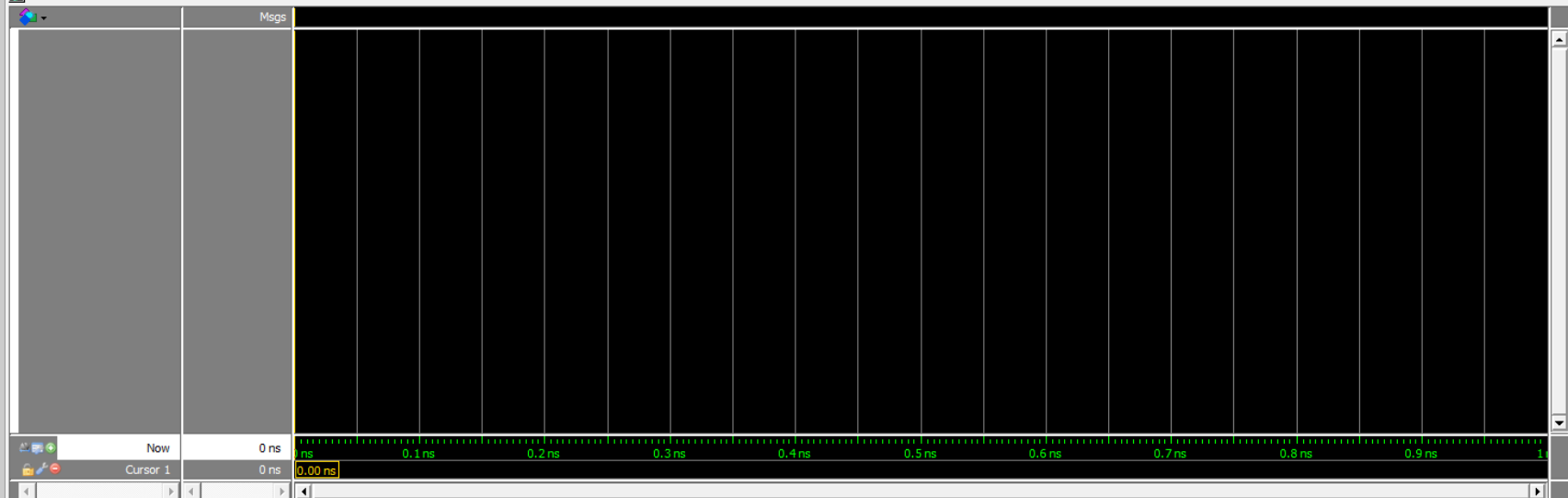
Instance	Design unit	Design unit type	Visibility	Total coverage
or32_testbench	or32_testbe...Module		+acc=<...	
or32_test	or32	Module	+acc=<...	
#vsmim_capacity#		Capacity	+acc=<...	

OR
TESTBENCH

[illegible]

Instance	Design unit	Design unit type	Visibility	Total coverage
 mult32_testbench	mult32_test...	Module	+acc=<...	
 c	mult32	Module	+acc=<...	
 #INITIAL#8	mult32_test...	Process	+acc=<...	
 #INITIAL#15	mult32_test...	Process	+acc=<...	
 #vsim_capacity#		Capacity	+acc=<...	

MULT32 TESTBENCH



Transcript

[illegible]

First result is that line. As you can see result is correct. I couldn't understand how to stop and print the result. Before each calculation last line is the result line. After result line multiplier and multiplicands are changed as you can see.

➤ Second result line is that line. As you can see result is correct.

ModelSim ALTERA STARTER EDITION 10.1d

File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help

100 ps

Layout Simulate

ColumnLayout AllColumns

sim - Default

Instance	Design unit	Design unit type	Visibility
alu32_testbench	alu32_testb...	Module	+acc=<...
alu	alu32	Module	+acc=<...
#INITIAL#10	alu32_testb...	Process	+acc=<...
#INITIAL#54	alu32_testb...	Process	+acc=<...
alu_capacity	Capacity		+acc=<...

Wave - Default

Now	832103.45 ns
Cursor 1	0.00 ns

Transcript

```
# time = 20, ALUop = 000, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 000010000100000100010101100
# time = 40, ALUop = 001, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 0000100001110000000110100001100
# time = 60, ALUop = 001, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 111101111011111101101010100010
# time = 80, ALUop = 010, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 11110111100011111111100111101000
# time = 100, ALUop = 010, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 00001000010000010001010101110
# time = 120, ALUop = 011, A = 0000000000000000000000000001010, B = 0000000000000000000000001100100, RESULT = 00000000000000000000000000000000
# time = 1410, ALUop = 011, A = 0000000000000000000000000001010, B = 0000000000000000000000001100100, RESULT = 0000000000000000000000001111010000
# time = 1450, ALUop = 011, A = 0000000000000000000000000001010, B = 0000000000000000000000001100100, RESULT = 000000000000000000000000111101000
# time = 1620, ALUop = 011, A = 0000000000000000000000000001010, B = 000000000000000000000000100000, RESULT = 000000000000000000000000111101000
# time = 1630, ALUop = 011, A = 0000000000000000000000000001010, B = 000000000000000000000000100000, RESULT = 00000000000000000000000000000000
# time = 2930, ALUop = 011, A = 0000000000000000000000000001010, B = 000000000000000000000000100000, RESULT = 00000000000000000000000010101000000
# time = 2970, ALUop = 011, A = 0000000000000000000000000001010, B = 000000000000000000000000100000, RESULT = 00000000000000000000000010101000000
# time = 3120, ALUop = 100, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 000000000000000000000000000000001
# time = 3140, ALUop = 100, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 000000000000000000000000000000000
# time = 3160, ALUop = 101, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 111101111000111011001011100010
# time = 3180, ALUop = 101, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 000000000000000000000000101011000
# time = 3200, ALUop = 110, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 0000000000000001000000000000101
# time = 3220, ALUop = 110, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 0000100001000001000101000000101
# time = 3240, ALUop = 111, A = 0000000000000001000101000000101, B = 0000100001110000100100000011101, RESULT = 0000100001110000100110100001101
# time = 3260, ALUop = 111, A = 0000100001000001000101000000101, B = 1111111111111111111101010011, RESULT = 1111111111111111111101010011
```

ALU32 TESTBENCH

(adder part)

(XOR part)

(Subtractor part)

Result of first multiplication. I explained the situation on the mult32 screenshots

Result of the second multiplication. I explained the situation on the mult32 screenshots

(SLT part)

(NOR part)

(AND part)

(OR part)

Now: 999,819,060 ps Delta: 0

sim:/alu32_testbench

0 ps to 770 ps