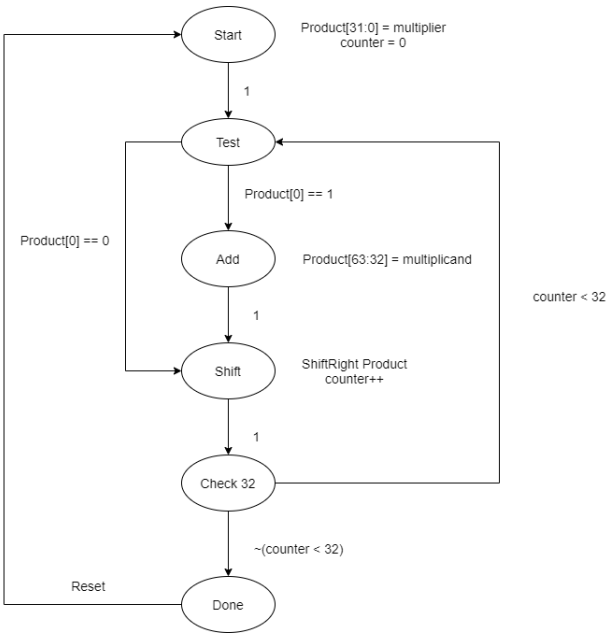
CSE 331/503 Computer Organization Homework 3 Report– ALU with Multiplication Design

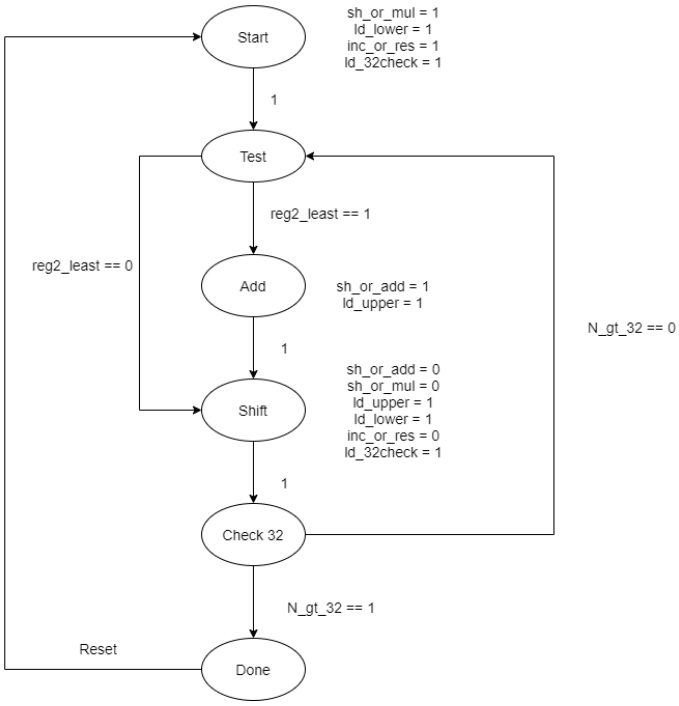
Hasan Mutlu – 1801042673

1. mult32.v Module

High Level State Machine:



Finite State Machine:



State Table:

State	Definition	R[2]	R[1]	R[0]
Start	Initialize multiplier to second register Initialize the counter	0	0	0
Test	Check least significant bit in product	0	0	1
Add	Add multiplicand to first register	0	1	0
Shift	Shift the product and save it Increment counter	0	1	1
check32	Check if counter reached 32	1	0	0
Done	Result is ready	1	0	1

States	R[2]	R[1]	R[0]	input	N[2]	N[1]	N[0]	Next State
Start	0	0	0	1	0	0	1	Test
Test	0	0	1	reg2_least	0	1	0	Add
	0	0	1	~reg2_least	0	1	1	Shift
Add	0	1	0	1	0	1	1	Shift
Shift	0	1	1	1	1	0	0	check32
check32	1	0	0	~N_gt_32	0	0	1	Test
	1	0	0	N_gt_32	1	0	1	Done
Done	1	0	1	Reset	0	0	0	Start
	1	0	1	~Reset	1	0	1	Done

N2 = R2'.R1.R0 + R2.R1'.R0'. N_gt_32 + R2.R1'.R0.Reset'

N1 = R2'.R1'.R0 + R2'.R1.R0'

N0 = R2'.R1'.R0' + R2'.R1'.R0. reg2_least' + R2'.R1.R0' + R2.R1'.R0' + R2.R1'.R0.Reset'

Sh_or_mul = R2'.R1'.R0'

Ld_lower = R2'.R1'.R0' + R2'.R1.R0

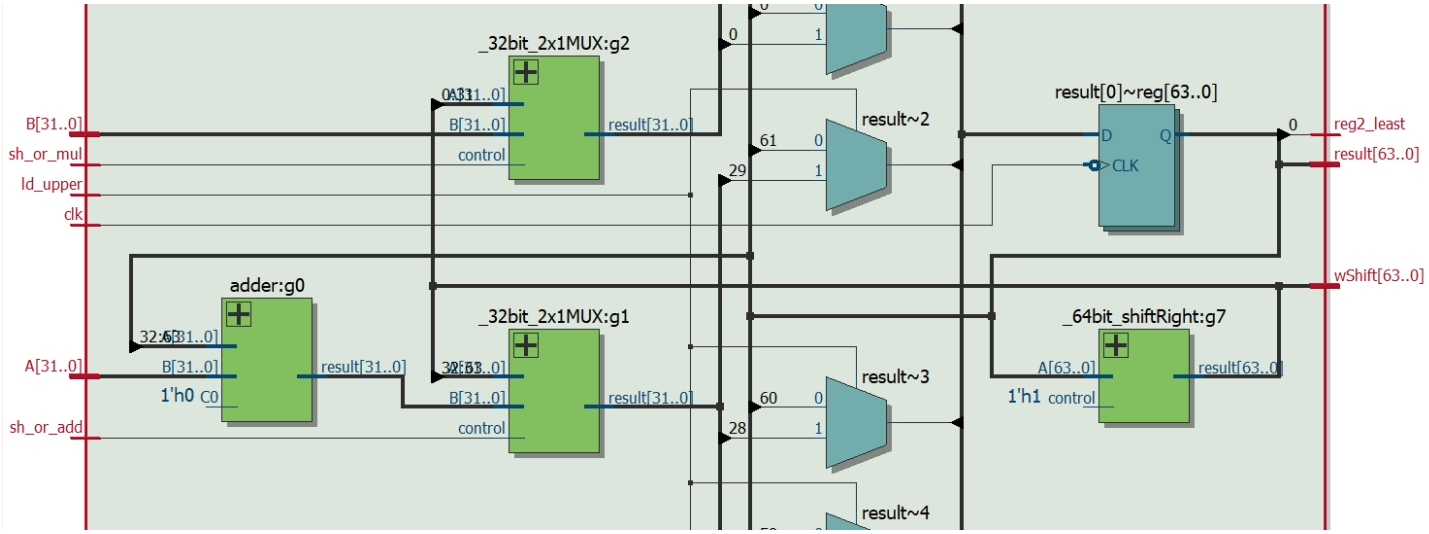
Ld_32check = R2'.R1'.R0' + R2'.R1.R0

Inc_or_res = R2'.R1'.R0'

Sh_or_add = R2'.R1.R0'

Ld_upper = R2'.R1.R0'+ R2'.R1.R0

Datapath:

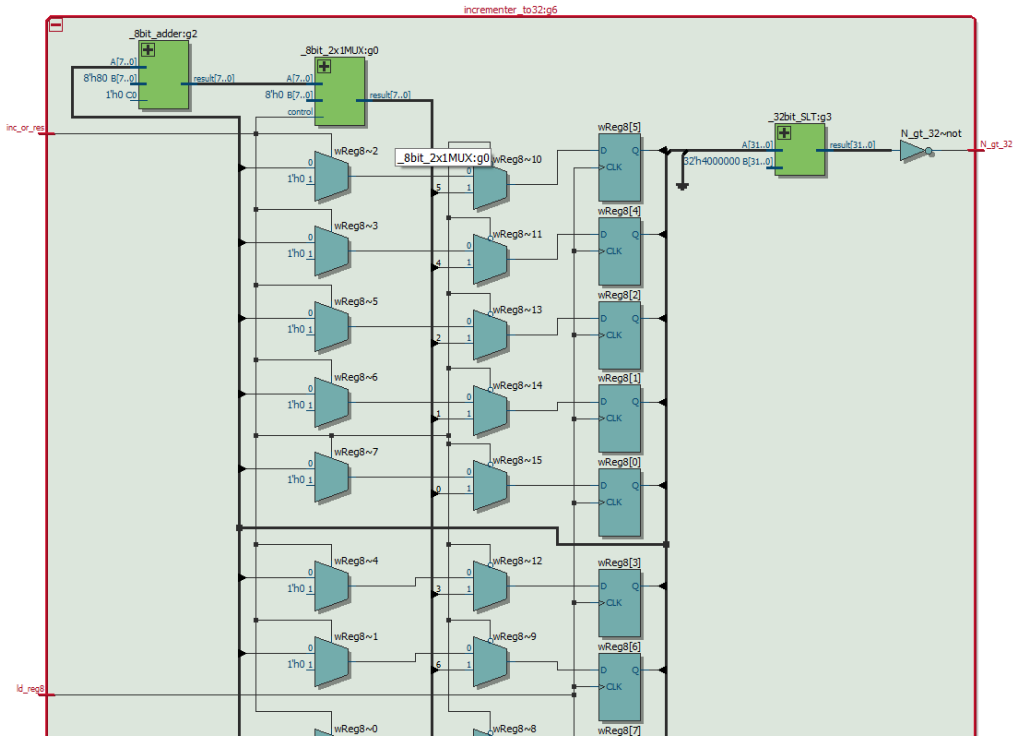


***The muxes on the middle are load input or keep state selection of 64bit register.**

G2 Mux gets shift[31:0] and multiplier as data, sh_or_mul as control.

G1 Mux gets shift[63:32] and addition result as data, sh_or_add as control.

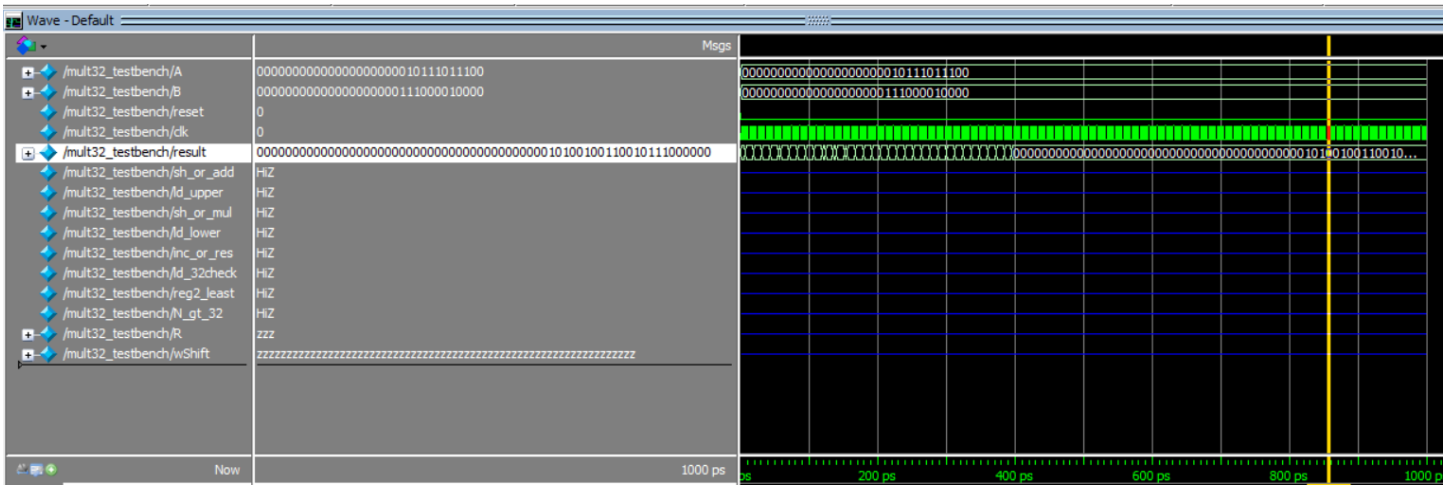
ShiftRight's control is always 1.



Datapath also incudes incrementer_to32 component. It is a counter up to 32.

It takes inc_or_res to increment or reset the counter. And ld_reg8 to load 8 bit register.

Testbench Result of mult32:



Convert Binary Value to Decimal Value

Result

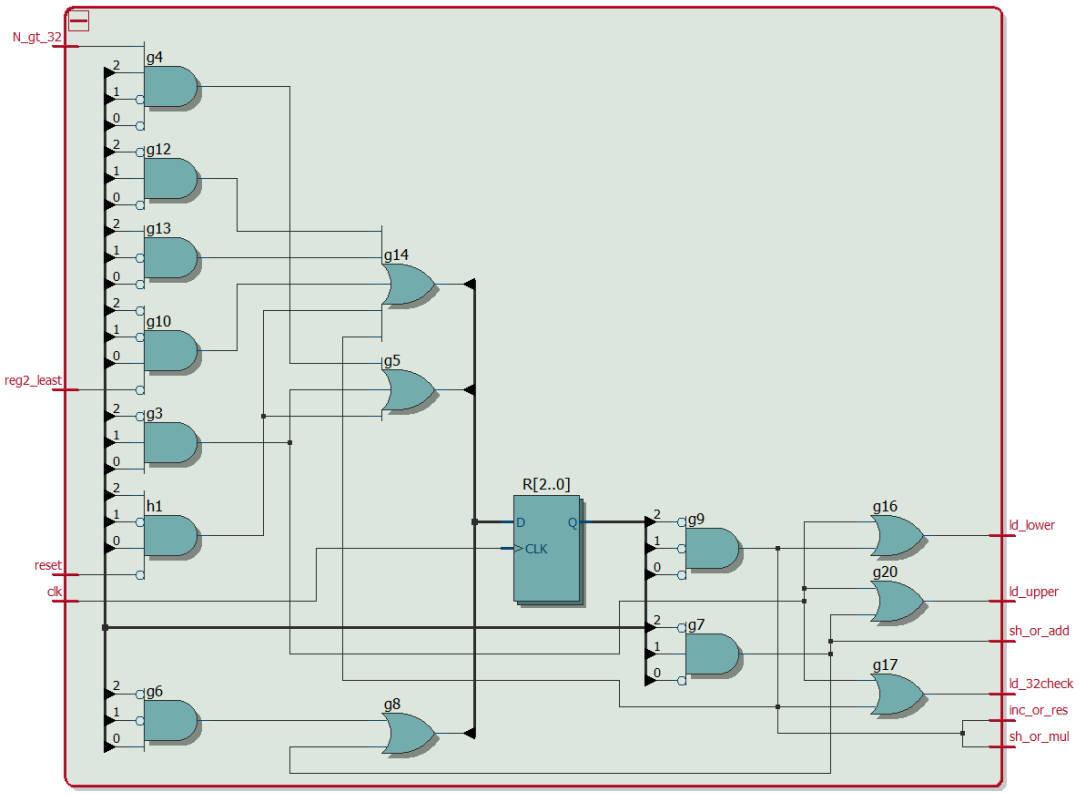
Decimal value: **5400000**

Binary Value: 10100100110010111000000 = ?

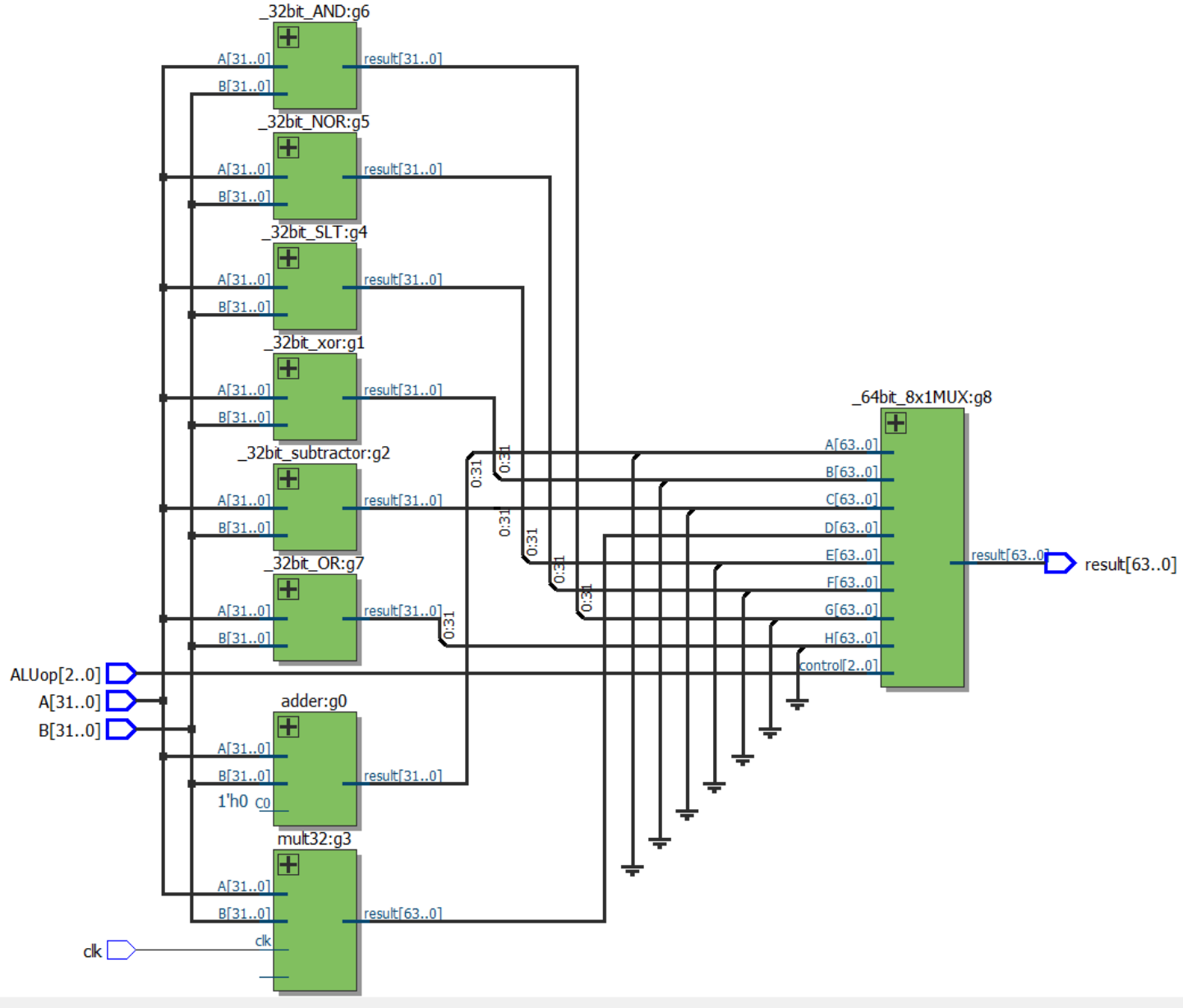
Calculate Clear

1500*3600 = 5400000

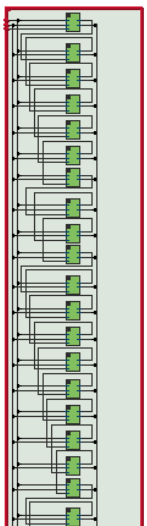
Controller:



1. alu32.v Module

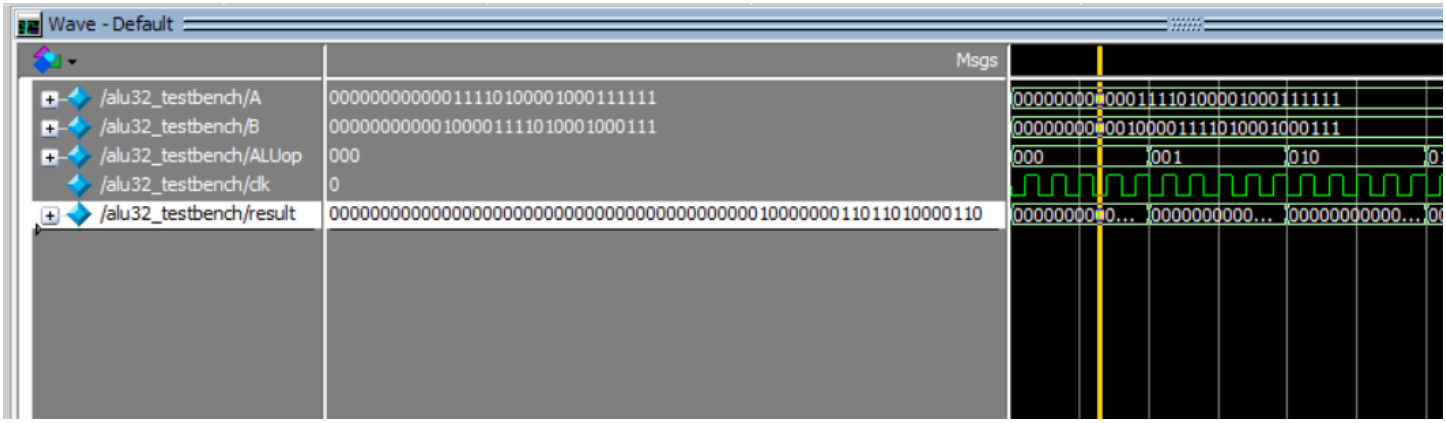


000.) Add Component: adder



It consists of 32 full adders.

Testbench Results:



Convert Binary Value to Decimal Value

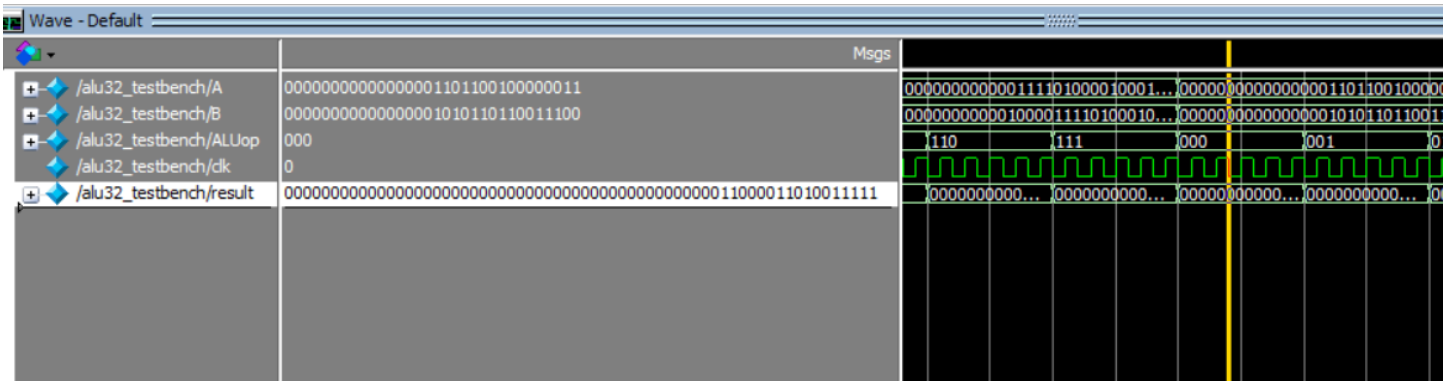
Result

Decimal value: **2111110**

Binary Value: 100000011011010000110 = ?

Calculate Clear

999999 + 1111111 = 2111110



Convert Binary Value to Decimal Value

Result

Decimal value: **99999**

Binary Value: 11000011010011111 = ?

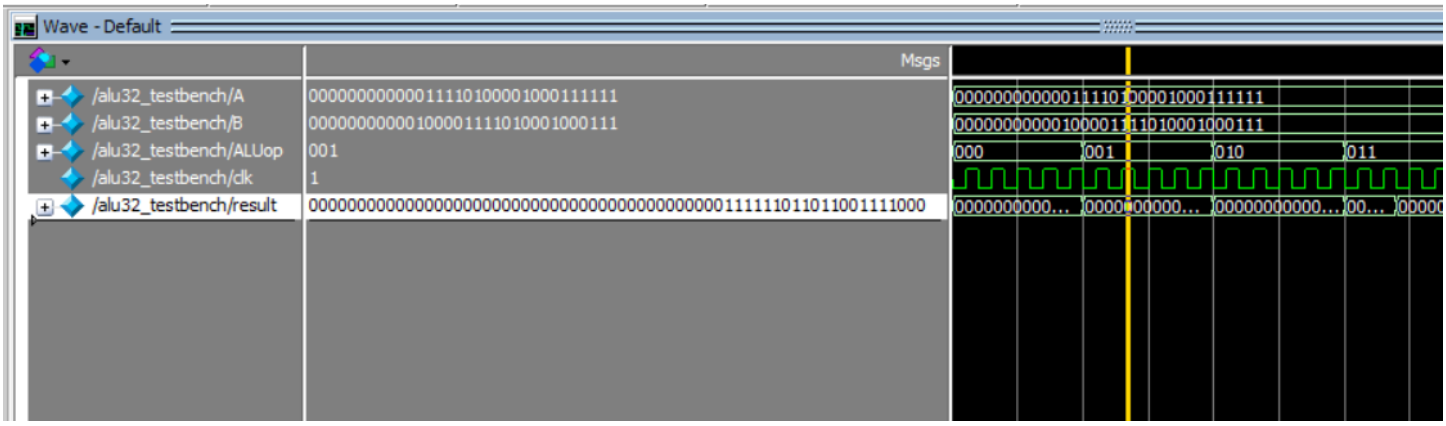
Calculate Clear

55555 + 44444 = 99999

001.) XOR Component: _32bit_XOR

It consists of 32 XOR gates.

Testbench Results:



XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

1111010000100011111

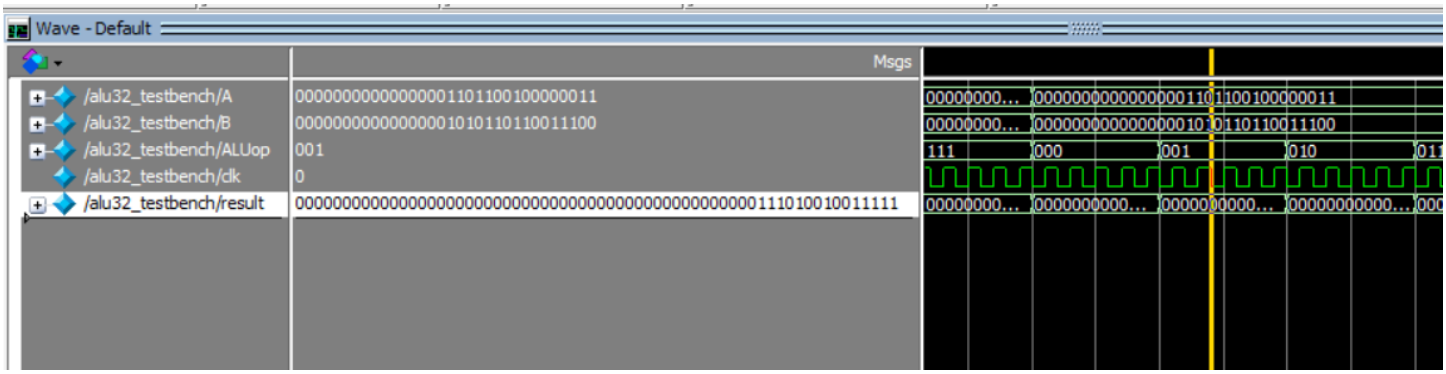
II. Input: hexadecimal (base 16)

10000111101000100011

Calculate XOR

III. Output: hexadecimal (base 16)

11111101101100111000



XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

1101100100000011

II. Input: hexadecimal (base 16)

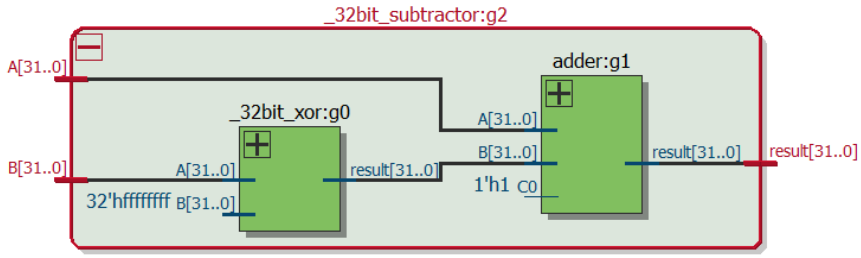
1010110110011100

Calculate XOR

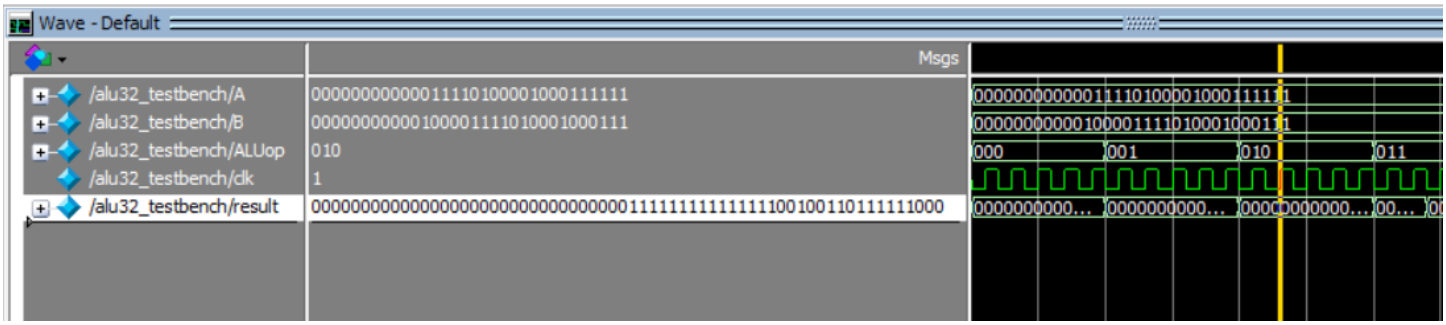
III. Output: hexadecimal (base 16)

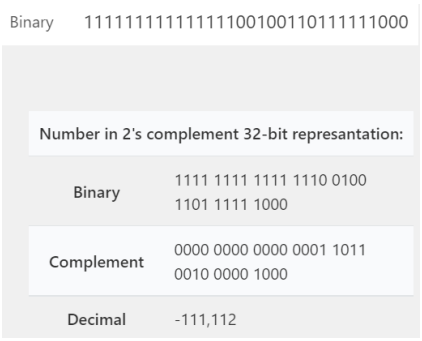
1110100100111111

010) Submission Component: _32bit_subtractor

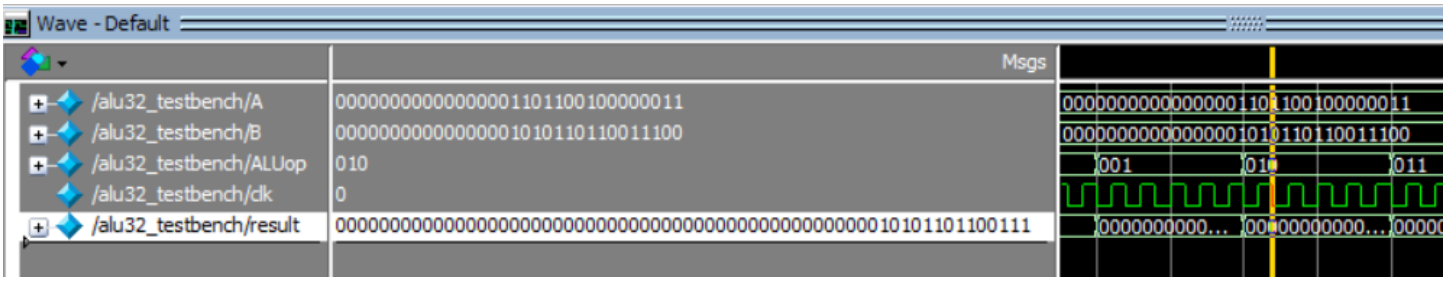


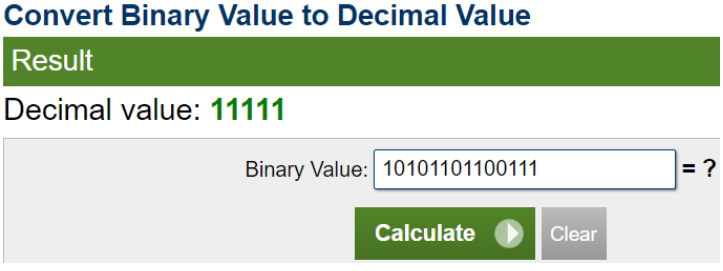
It consists of one 32bit XOR block and one adder.





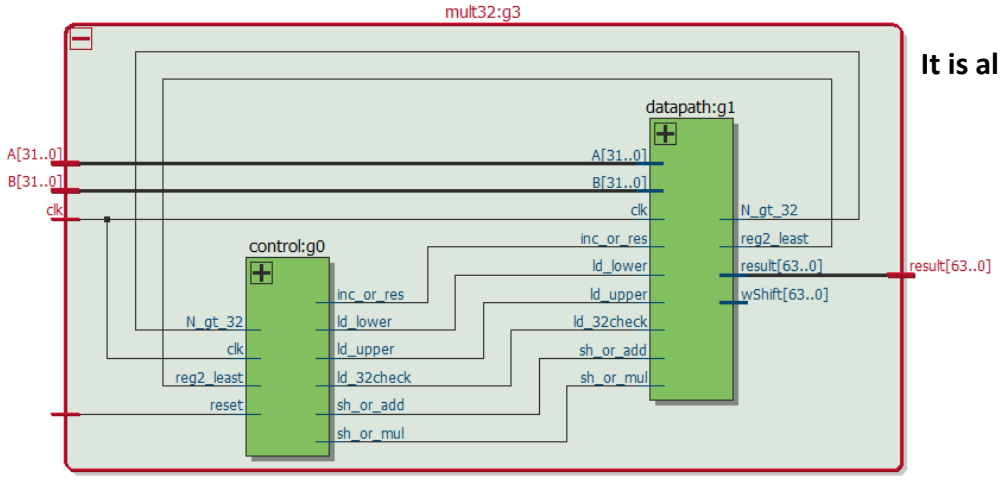
999999 - 1111111 = -111,112



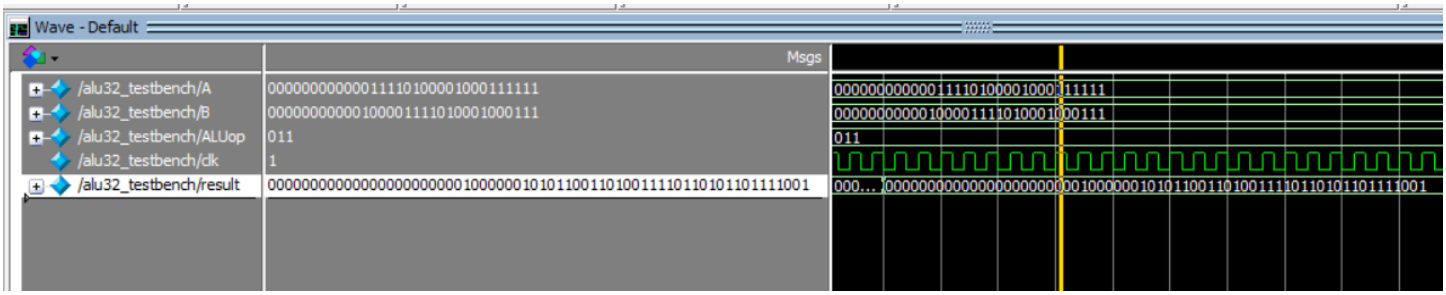


55555 – 44444 = 11111

011) Multiplication Module: mult32



It is already defined in part 1.



Convert Binary Value to Decimal Value

Result

Decimal value: **1111109888889**

Binary Value: 10000001010110011010011111 = ?

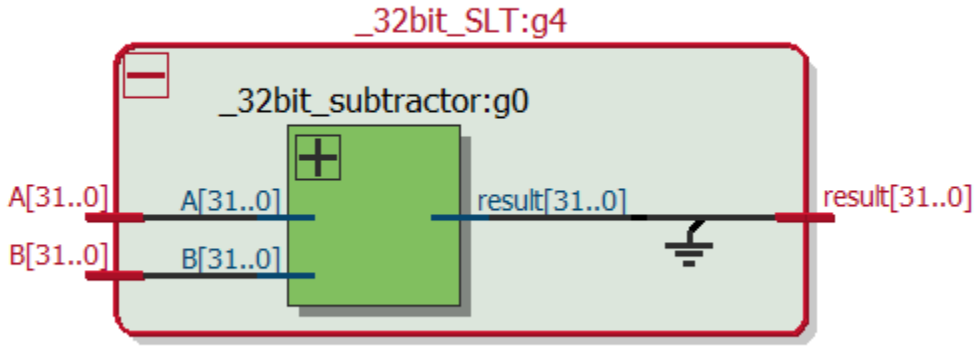
Calculate Clear

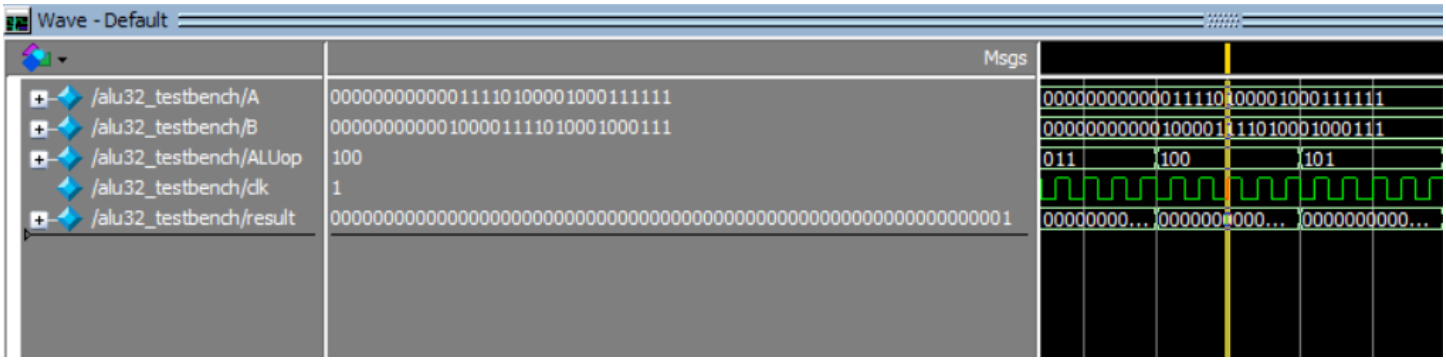
999999 * 1111111 = 1111109888889

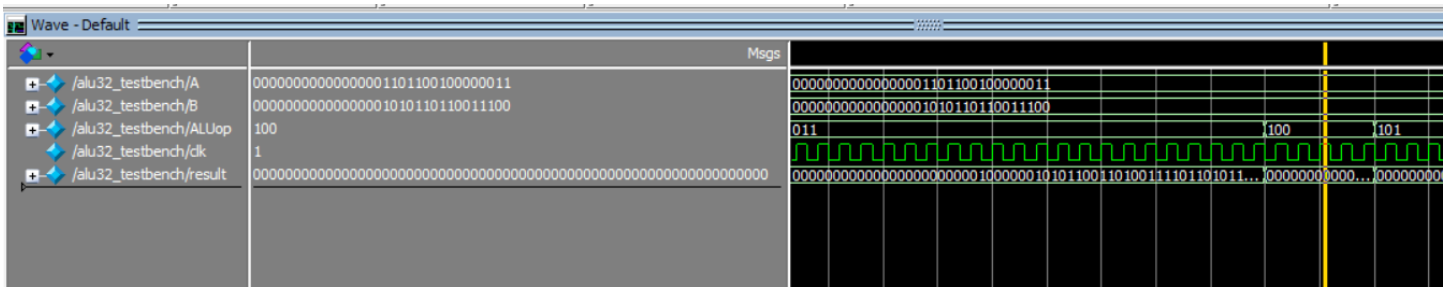
➔ **Unfortunately, I couldn't reset the multiplier after first call, so there isn't a second testbench result in alu32_testbench.**

100) Set on less than Component: _32bit_SLT

It consists of a 32 bit subtractor. Result's most significant bit is returned as result.







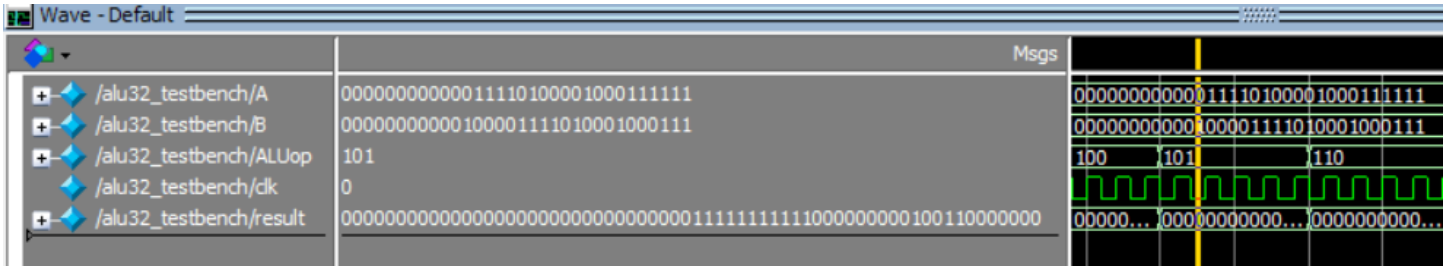
999999 < 1111111 == true

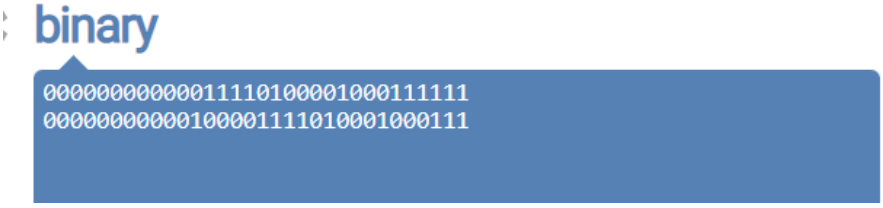
55555 < 44444 == false

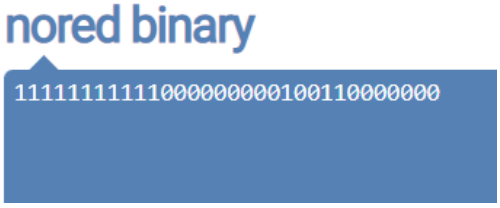
Validity of the results are obvious.

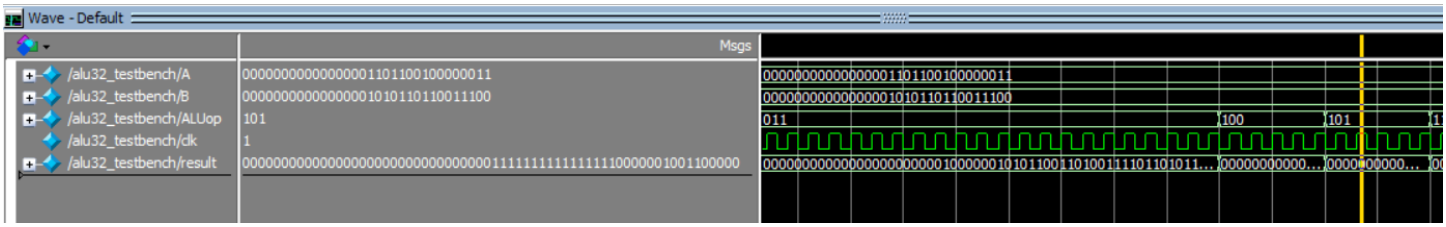
101.) NOR Component: _32bit_NOR

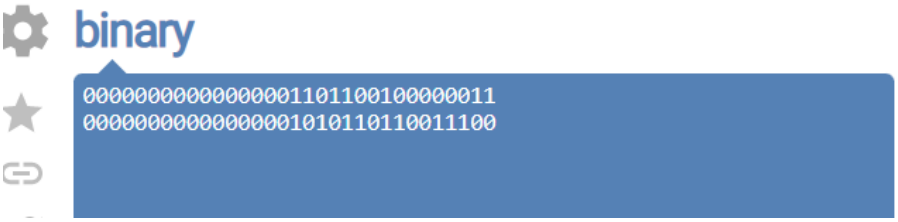
It consists of 32 consecutive NOR gates.





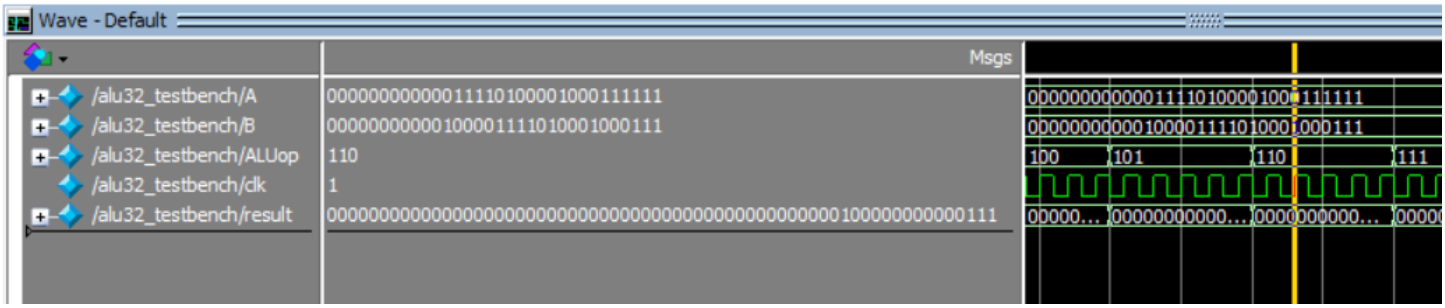


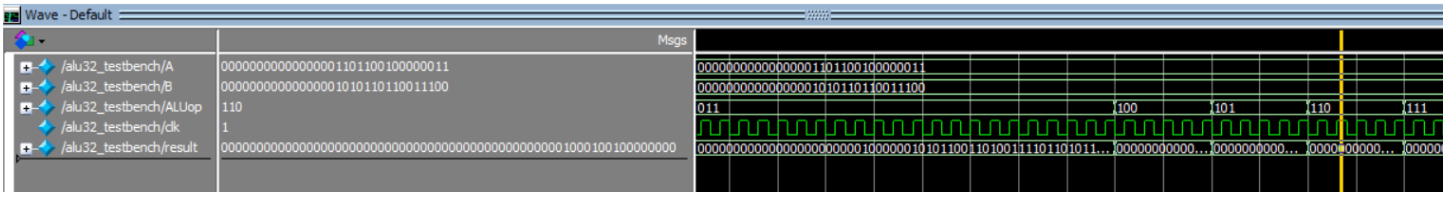


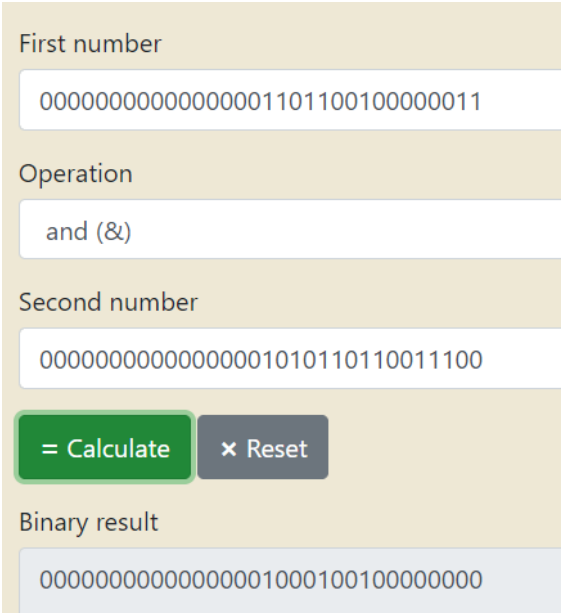


110) AND Component: _32bit_AND

It consists of 32 consecutive and gates.

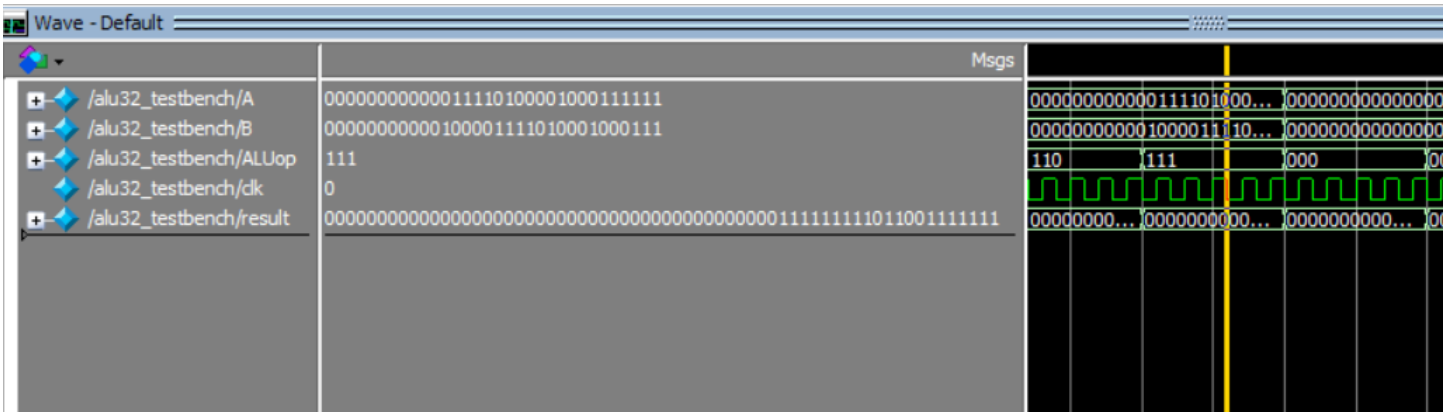


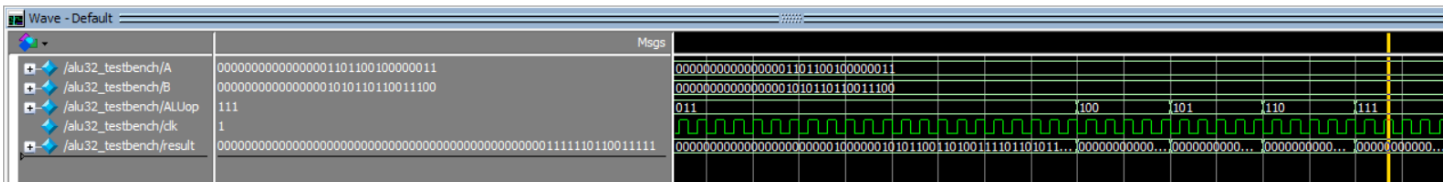




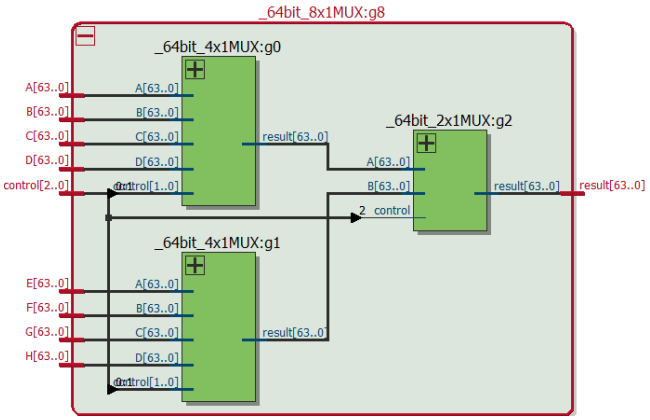
111.) OR Component: _32bit_OR

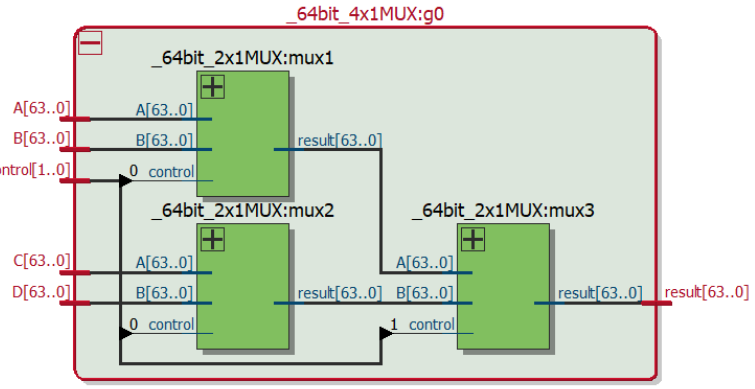
It consists of 32 consecutive or gates.

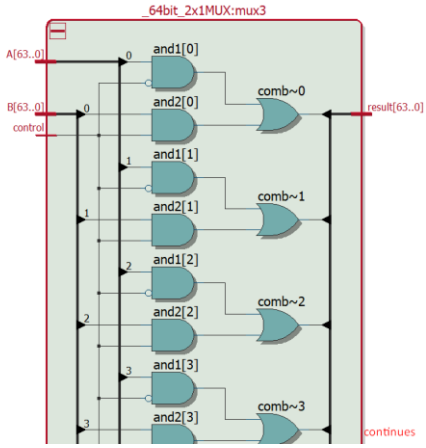


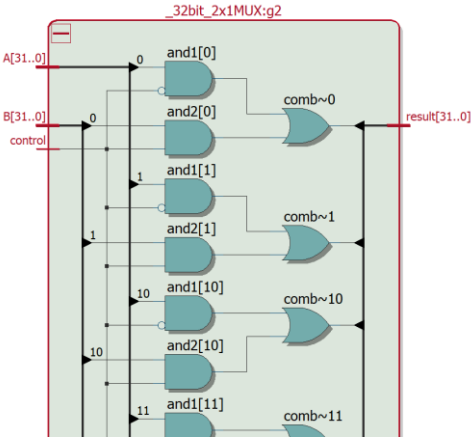


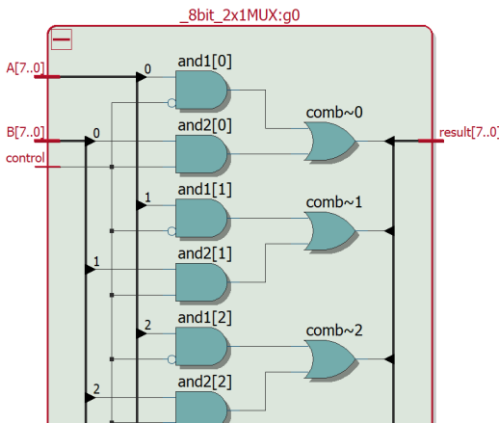
Helpful Components:

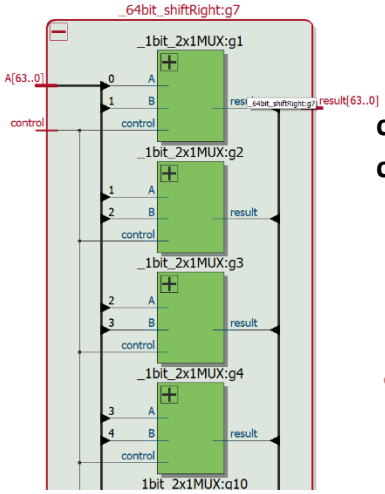












ShiftRight component includes 64 1bit 2x1 Muxes. Consecutive bit and current bit are connected to muxes and there is control. In this homework, this control input was always 1.