CSE 331 HOMEWORK 2 REPORT

The system consists of 4 big modules. These are alu32, mult32, datapath, and control modules.

1. alu32

alu32 module gets four inputs. These are a, b, ALUop and clock. a and b keep 32 bits binary numbers. ALUop keeps 3 bits binary number which set the operation such as addition, substraction, multiplication, etc. In alu32 module, each operation is applied on given binary numbers. After that, with the help of the multiplexer, the right result is shown to the user.

2. mult32

mult32 module gets seven inputs. These are multiplier, multiplicand, clock, input, mult, add, sub, slt. multiplier and multiplicand keep 32 bits binary numbers. mult, add, sub, and sub inputs are for operation type. Firstly, a temp register is created by first 32 bits are for multiplier, and rest of the bits are empty. In the second iteration, things to do differ according to the operation. For example, if the operation is one of the add, sub or slt, the number is sent to the datapath, and result is shown. On the other hand, if the operation is mult, the temp register is sent to datapath with the information about last bit is zero or not and shift right one. After that, the contents of the result of the datapath are sent to the temp register, and this iteration continues until all 32 bits are checked.

3. datapath

datapath module gets seven inputs. These are multiplier, multiplicand, clock, input, mult, add, sub, slt. multiplier and multiplicand keep 32 bits binary numbers. mult, add, sub, and sub inputs are for operation type. datapath module is responsible of adding two numbers and shifting the output one bit right.

4. control

control module gets two inputs. These are lastbit and clock. Every posedge of clock, lastbit is checked. If the lastbit is 0, returns 0, otherwise returns 1.

5. adder

adder module calls fullAdder16bit module twice.

6. fullAdder16bit

fullAdder16bit calls fullAdder4bit module four times.

7. fullAdder4bit

fullAdder4bit calls fullAdder1bit module four times.

8. fullAdder1bit

fullAdder1bit adds two bits binary number and returns the result as sum and cout. In the case of operation is sub, inverts the second number before adding.

9. and32bit

and32bit module ands two 32 bits binary number by calling and8bit module 8 times.

10. or32bit

or32bit module ors two 32 bits binary number by calling or8bit module 8 times.

11. xor32bit

xor32bit module xors two 32 bits binary number by calling xor8bit module 8 times.

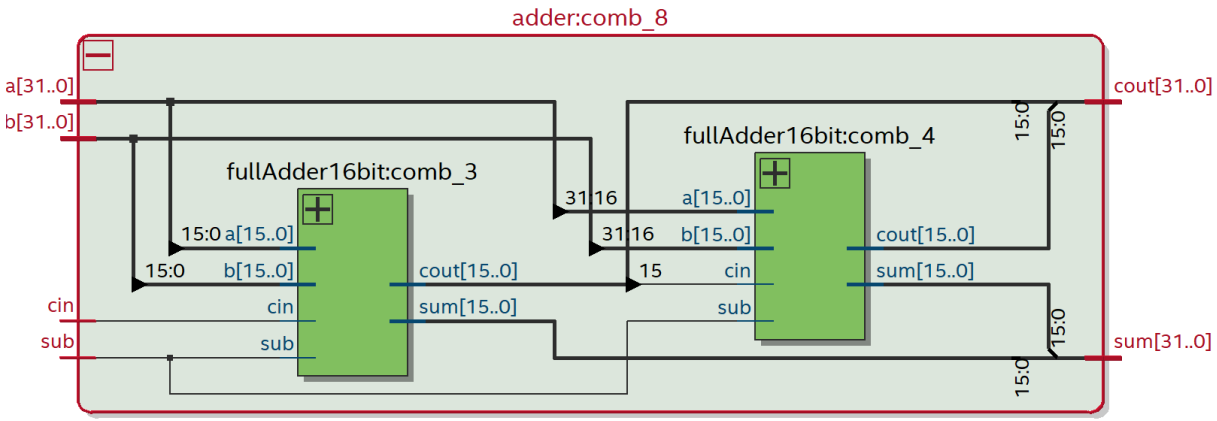
12. nor32bit

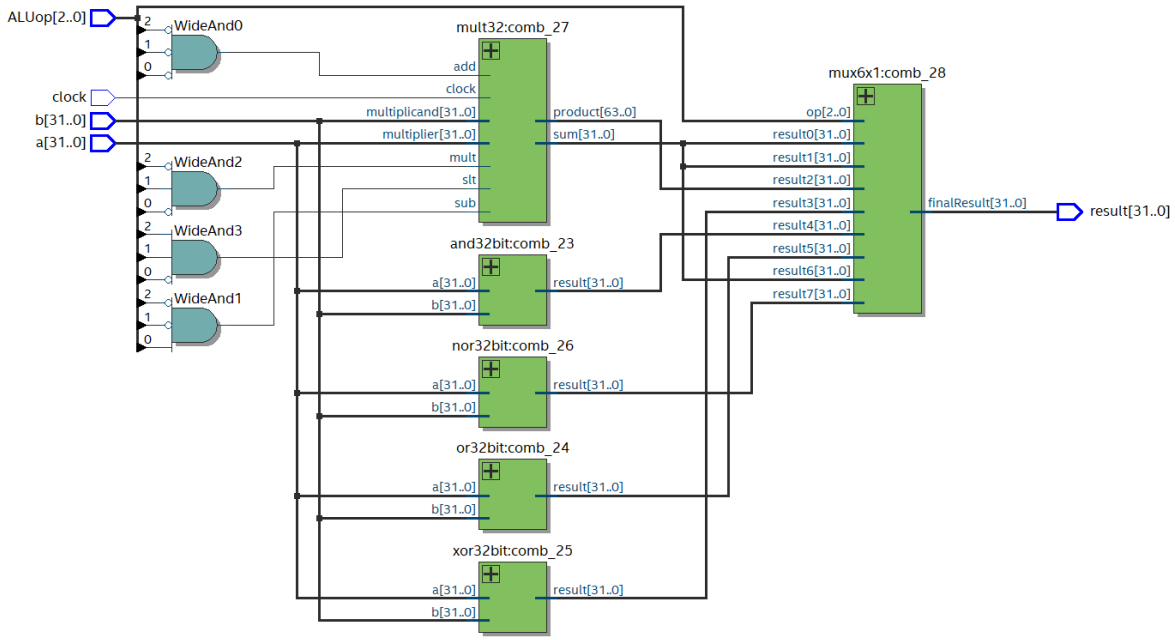
xor32bit module nors two 32 bits binary number by calling nor8bit module 8 times.

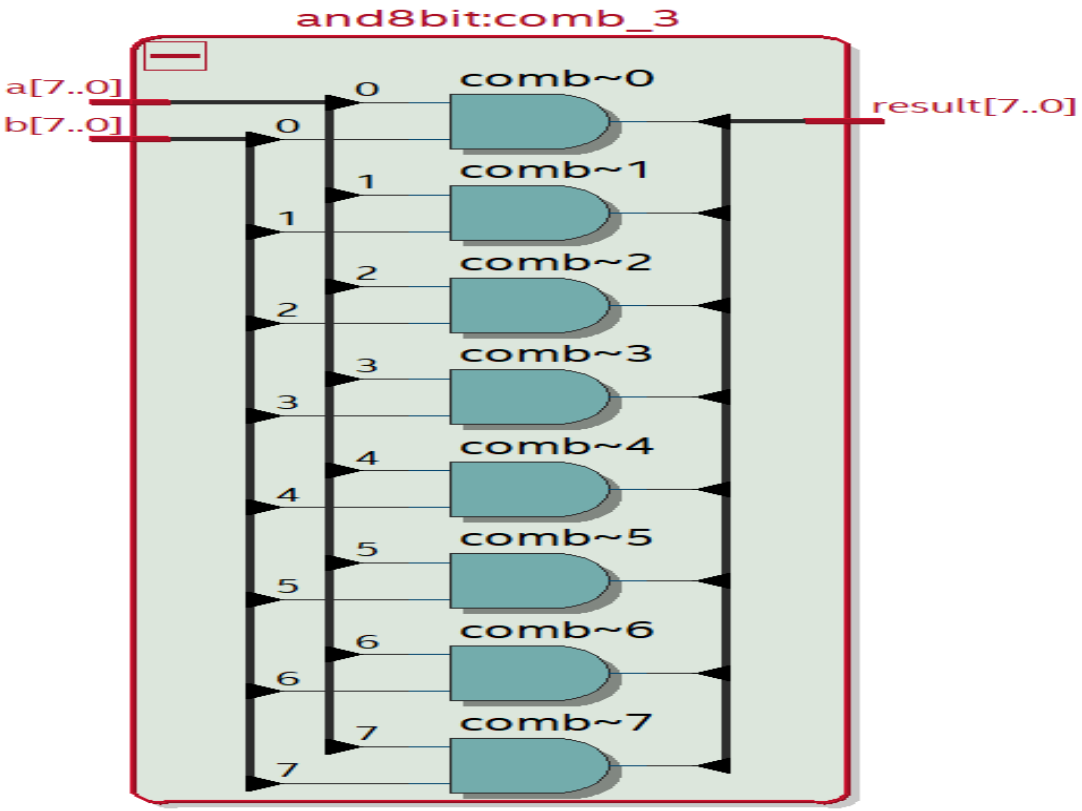
13. **mux6x1**

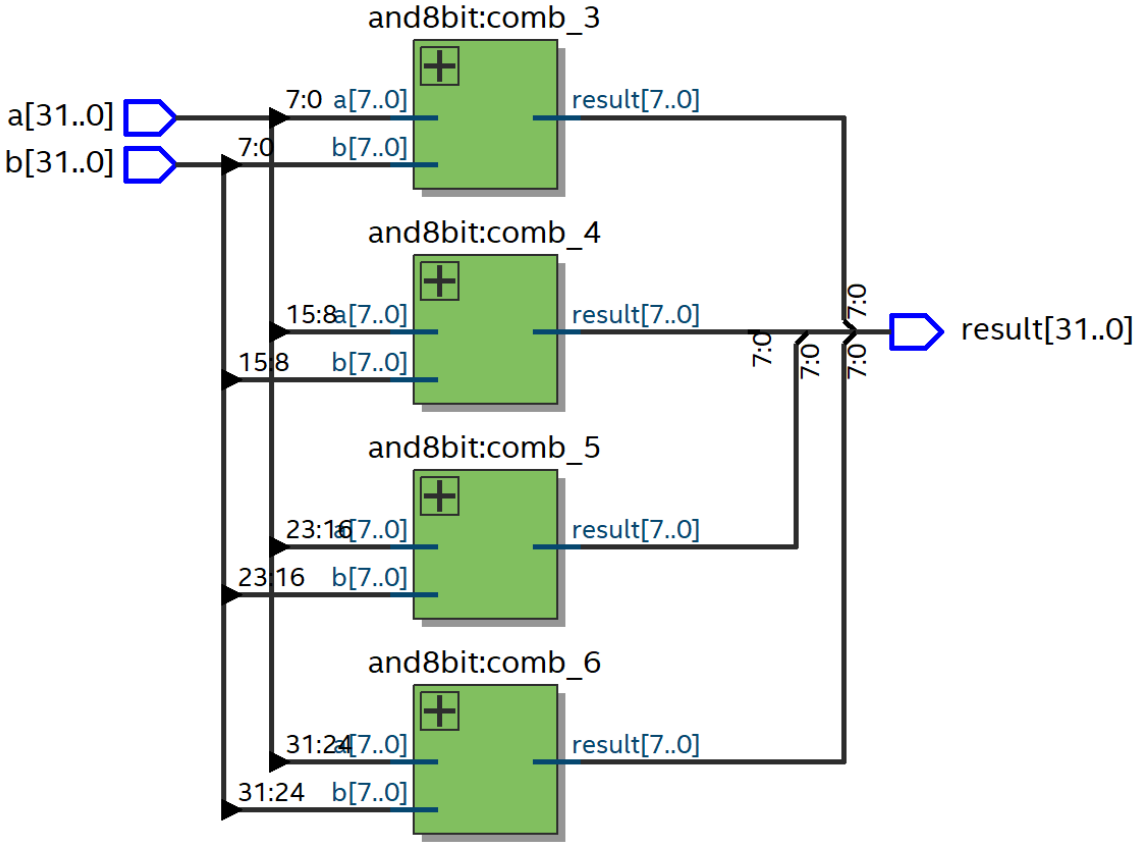
mux6x1 module gets 8 results and returns one of them according to ALUop. This module used in alu32 module.

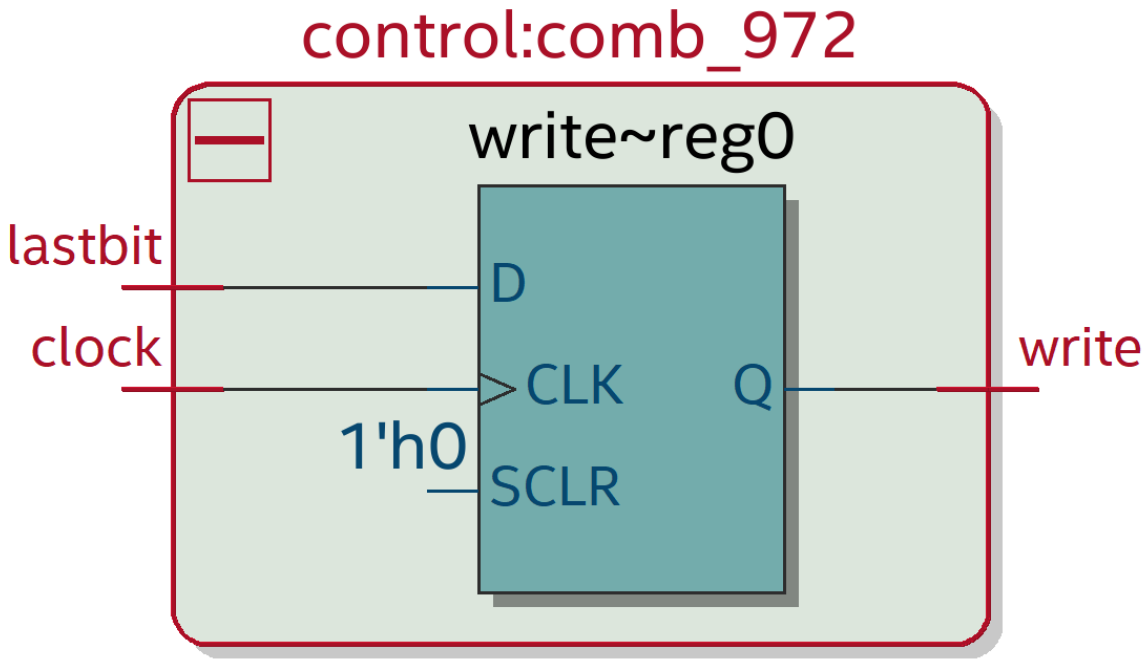
PS: slt operator does not work and shematics are also uploaded in shematics folder with their names.

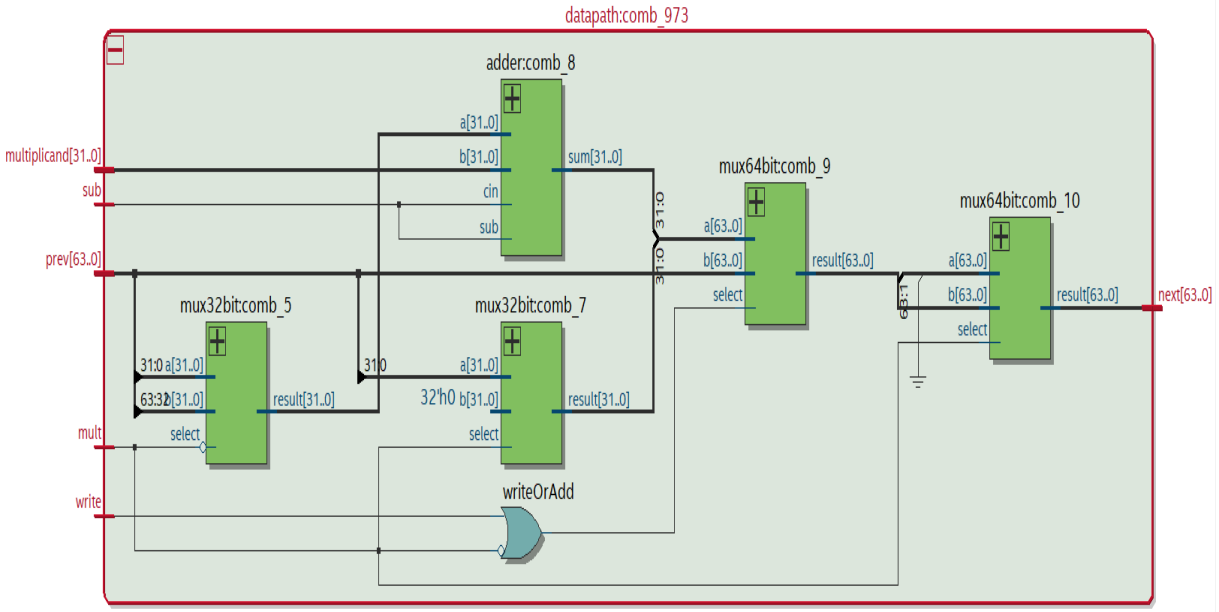


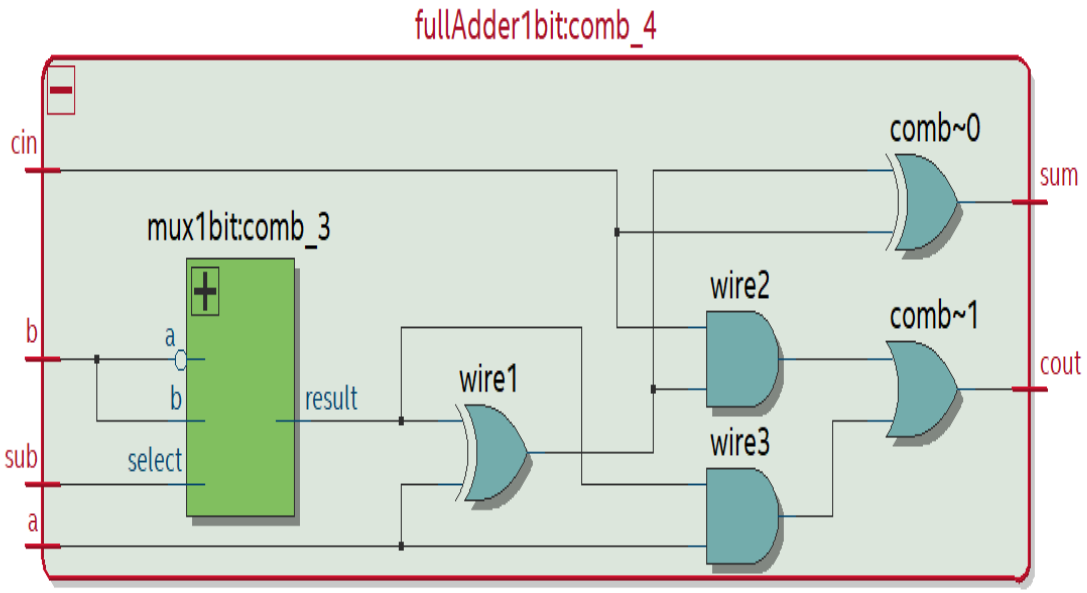


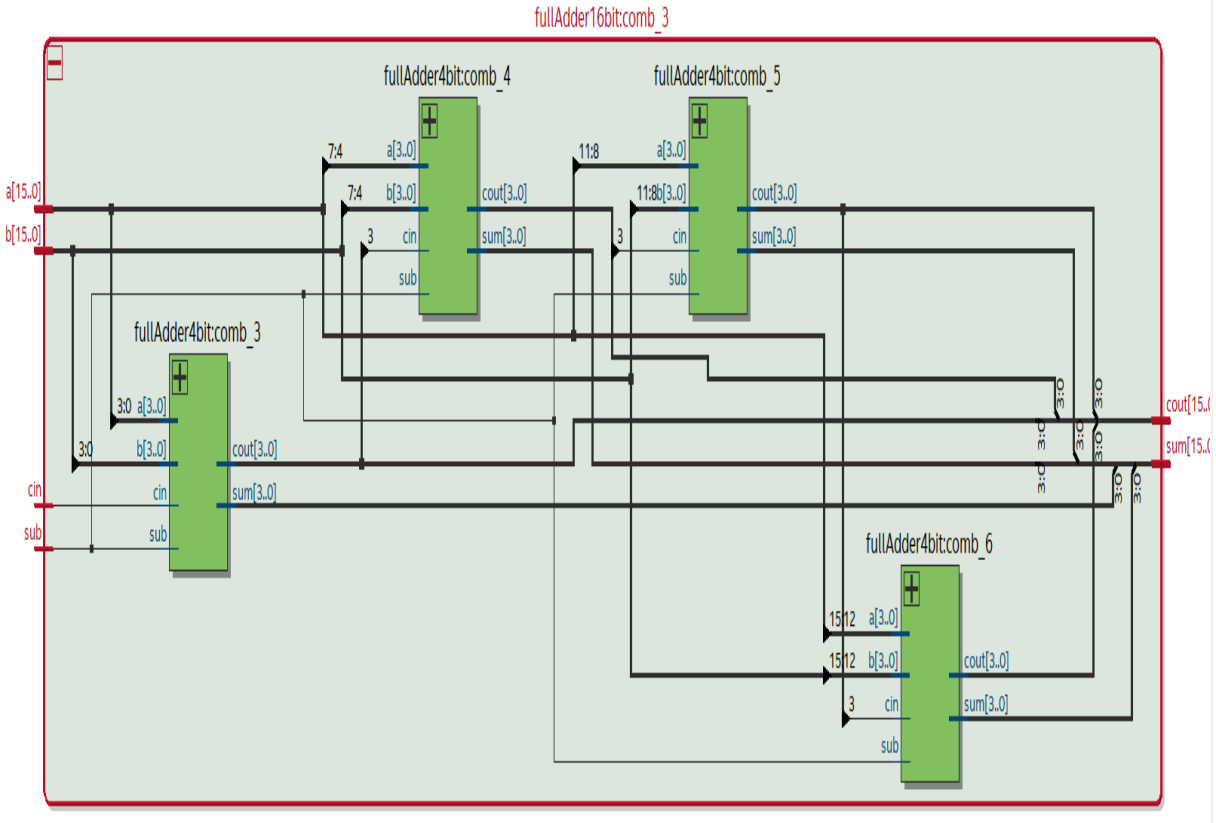

and8bit:comb_3
a[7..0]
b[7..0]
result[7..0]
comb~0
comb~1
comb~2
comb~3
comb~4
comb~5
comb~6
comb~7

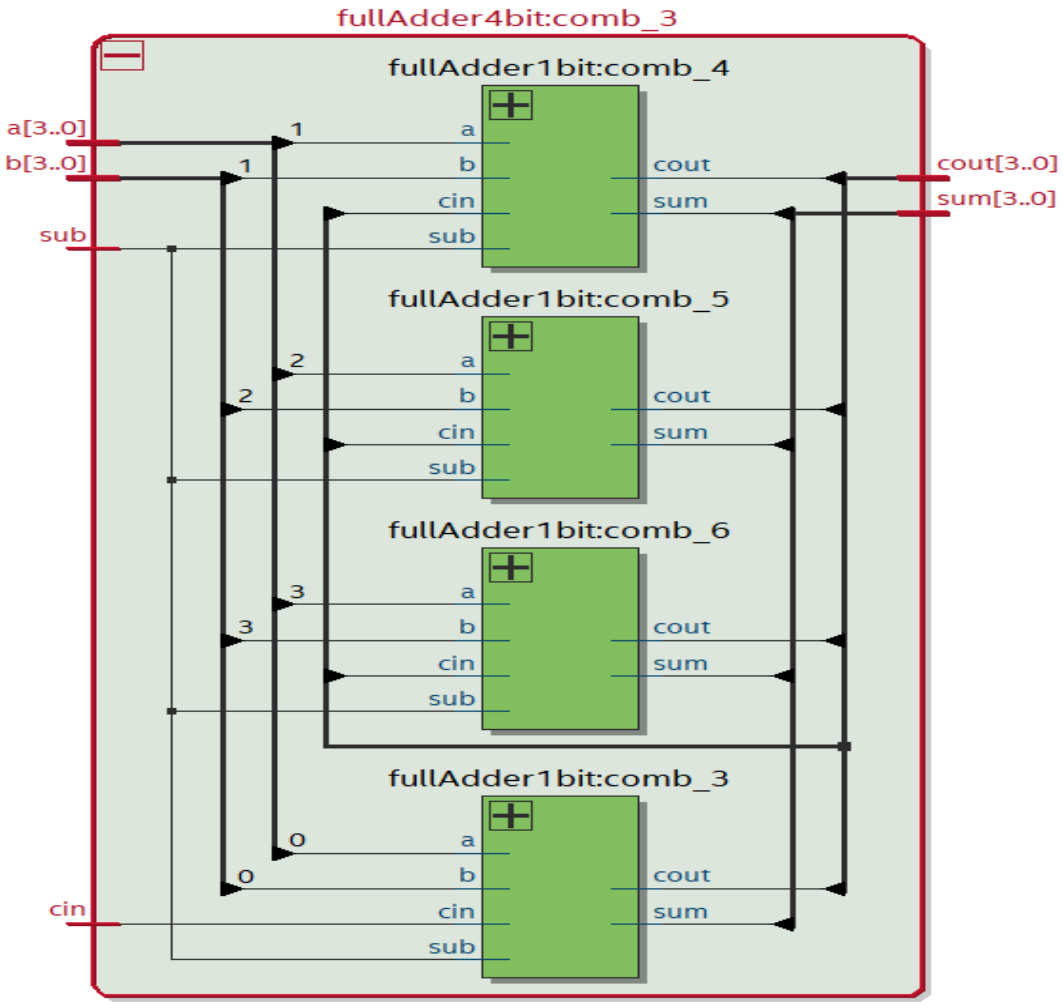

and8bit:comb_3
a[31..0]
b[31..0]
7:0 a[7..0]
b[7..0]
result[7..0]
and8bit:comb_4
15:8 a[7..0]
15:8 b[7..0]
result[7..0]
and8bit:comb_5
23:16 a[7..0]
23:16 b[7..0]
result[7..0]
and8bit:comb_6
31:24 a[7..0]
31:24 b[7..0]
result[7..0]
7:0
result[31..0]

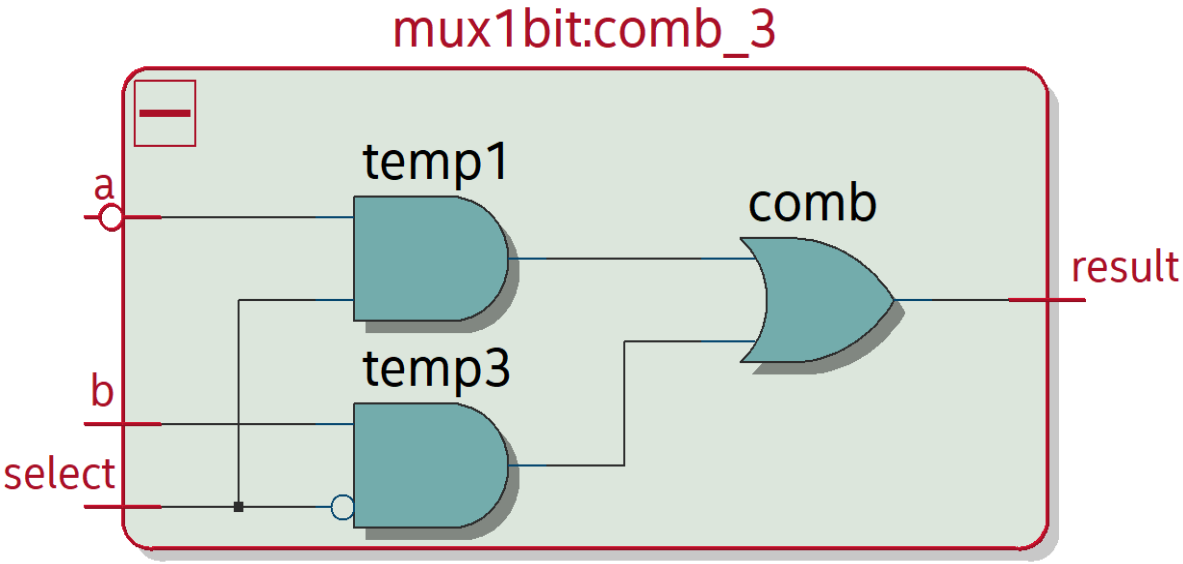

control:comb_972
write~reg0
lastbit
clock
1'h0
D
CLK
SCLR
Q
write

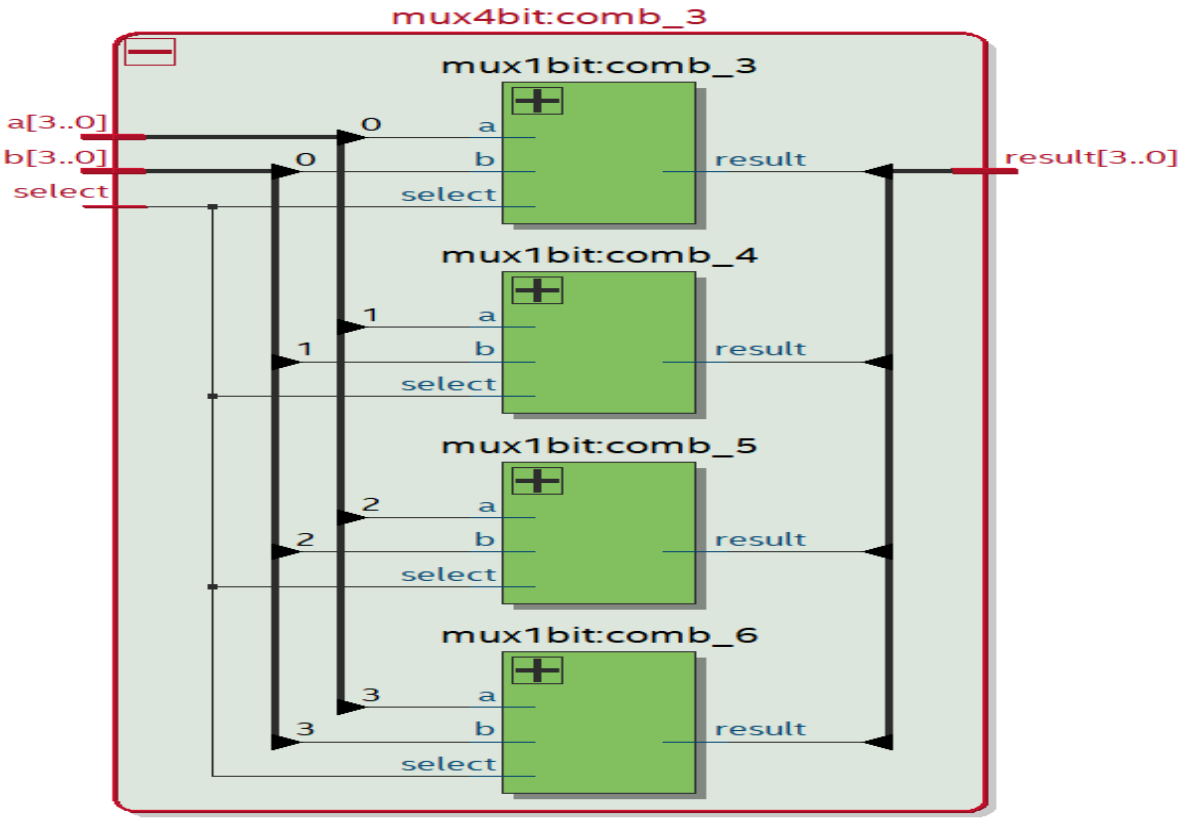

datapath:comb_973
adder:comb_8
a[31..0]
b[31..0]
cin
sub
sum[31..0]
multiplicand[31..0]
sub
prev[63..0]
mux64bit:comb_9
a[63..0]
b[63..0]
select
result[63..0]
31:0
mux64bit:comb_10
a[63..0]
b[63..0]
select
result[63..0]
next[63..0]
63:1
mux32bit:comb_5
31:0 a[31..0]
63:32 b[31..0]
select
result[31..0]
mux32bit:comb_7
a[31..0]
32'h0 b[31..0]
select
result[31..0]
31:0
mult
write
writeOrAdd

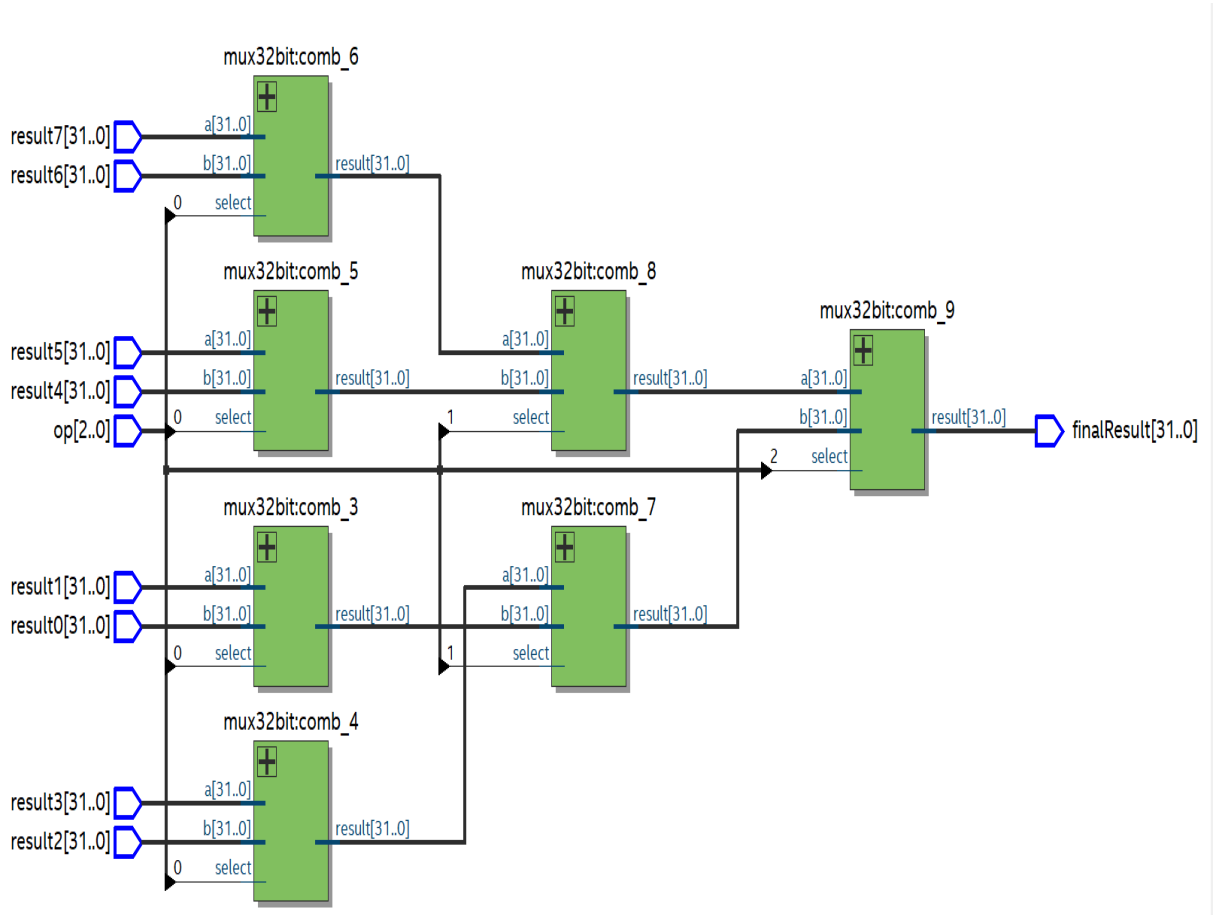

fullAdder1bit:comb_4
cin
b
sub
a
mux1bit:comb_3
a
b
select
result
wire1
wire2
wire3
comb~0
comb~1
sum
cout

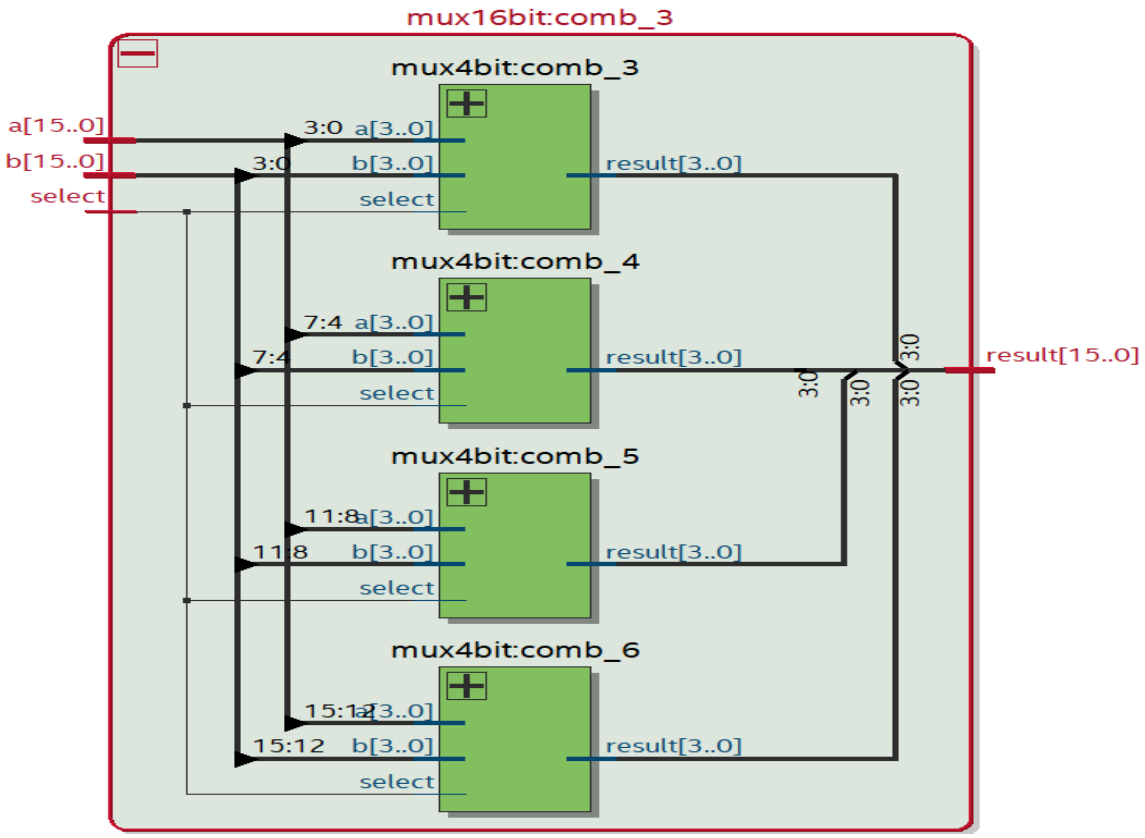

fullAdder16bit:comb_3
fullAdder4bit:comb_4
fullAdder4bit:comb_5
fullAdder4bit:comb_3
fullAdder4bit:comb_6
a[15..0]
b[15..0]
cin
sub
a[3..0]
b[3..0]
cout[3..0]
sum[3..0]
cout[15..
sum[15..

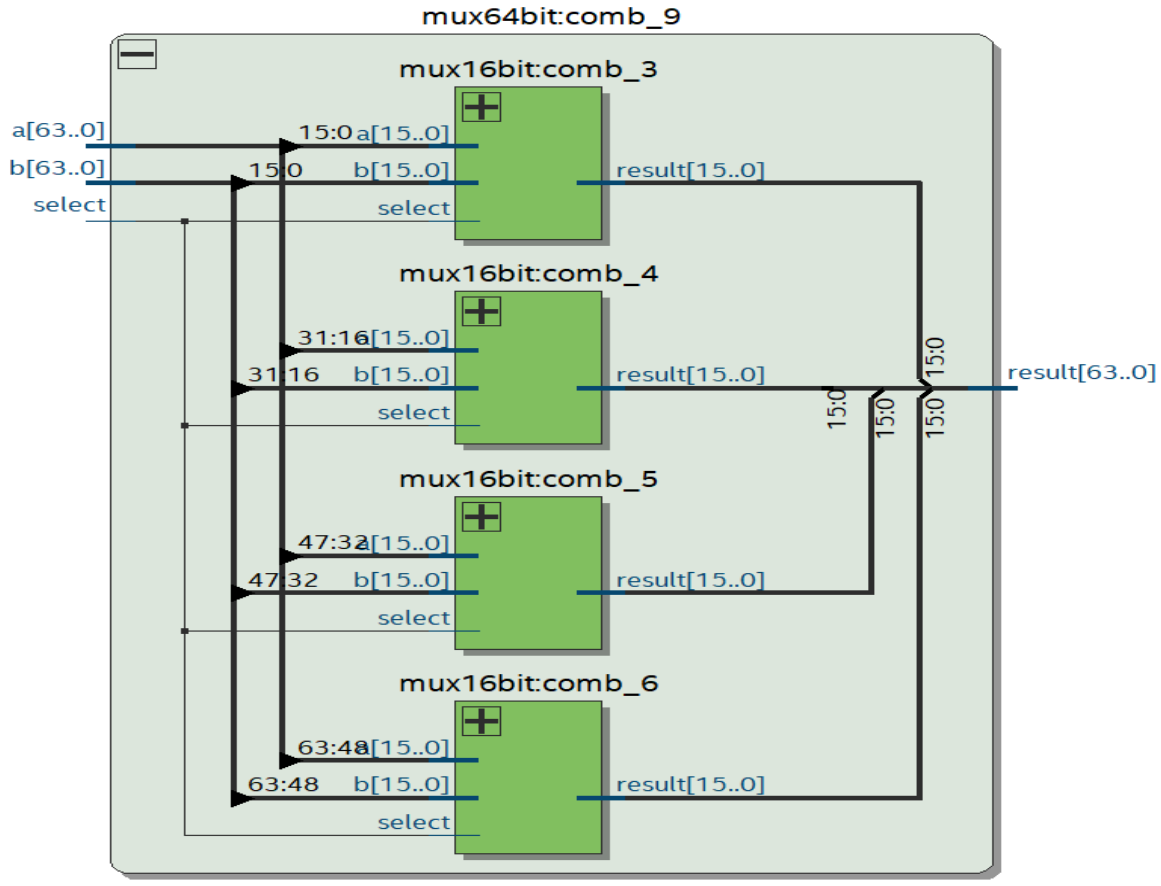

fullAdder4bit:comb_3
fullAdder1bit:comb_4
fullAdder1bit:comb_5
fullAdder1bit:comb_6
fullAdder1bit:comb_3
a[3..0]
b[3..0]
sub
cin
cout[3..0]
sum[3..0]
a
b
cin
sub
cout
sum
1
2
3
0

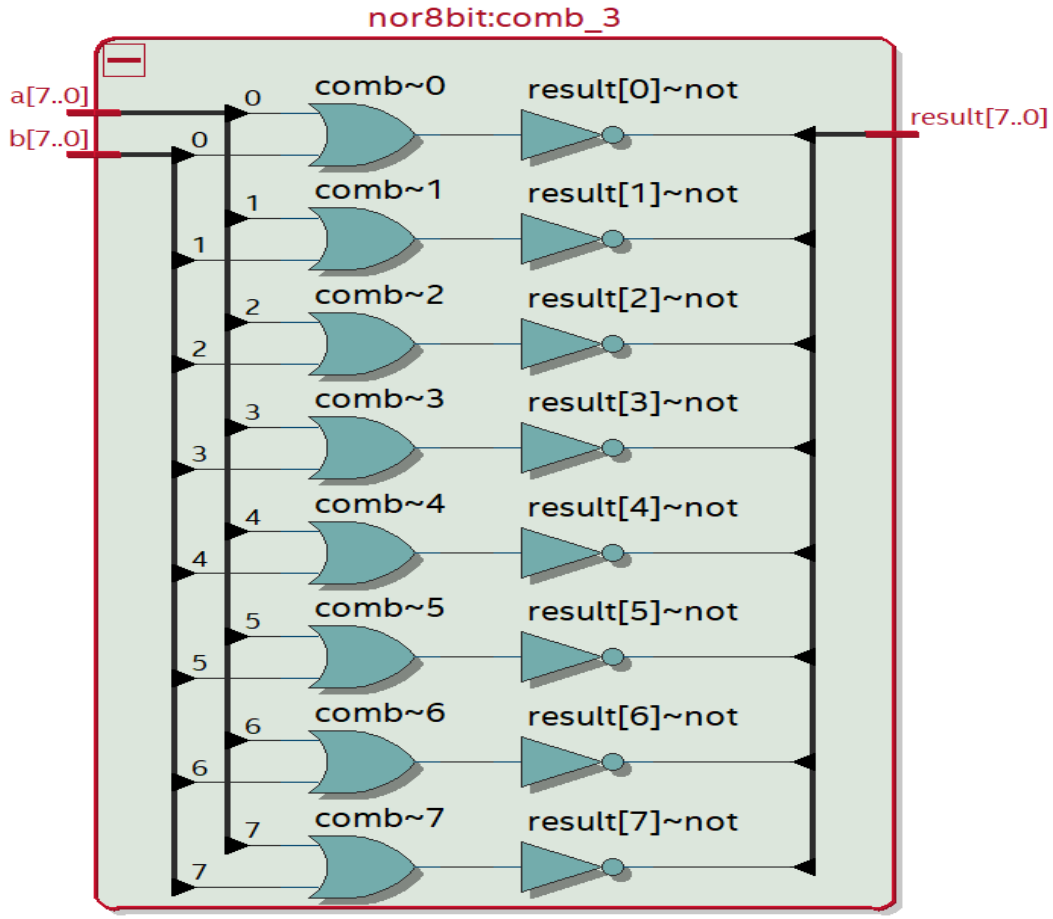

mux1bit:comb_3
a
b
select
temp1
temp3
comb
result

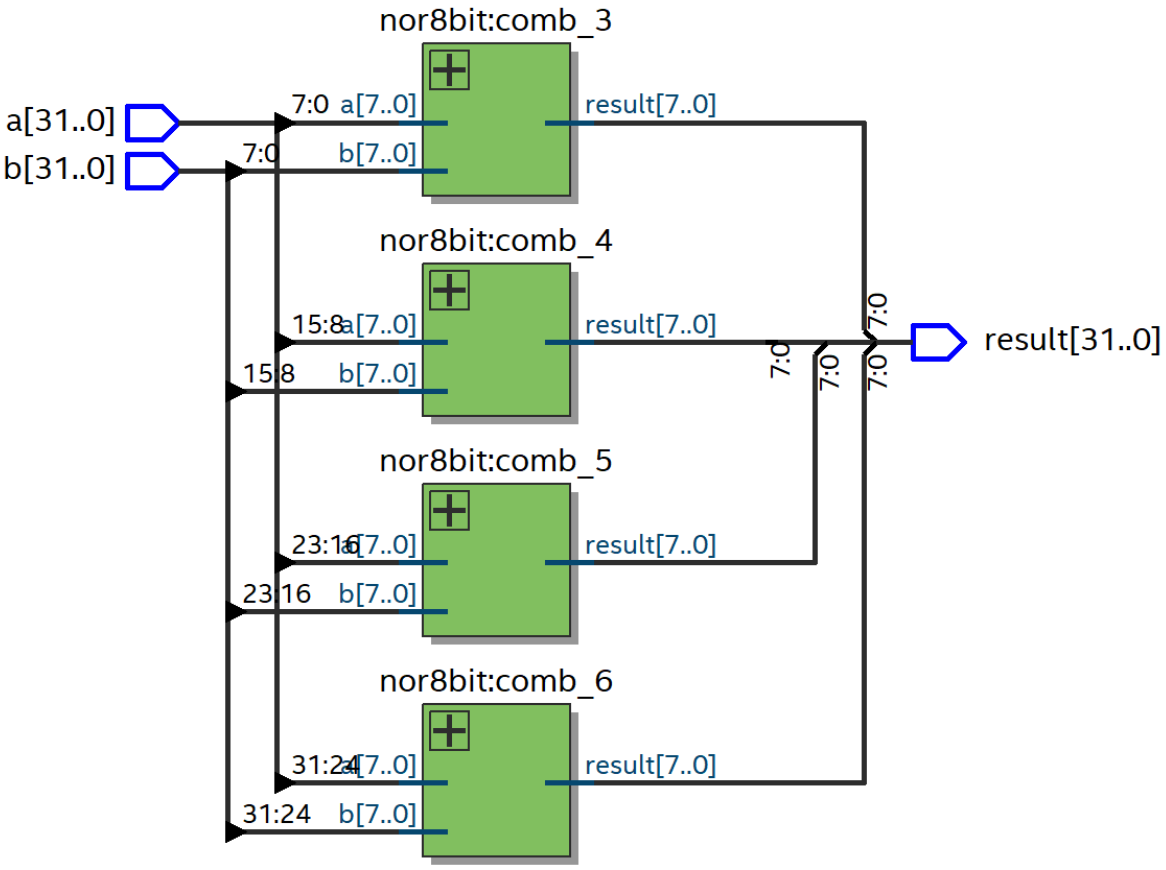

mux4bit:comb_3
mux1bit:comb_3
mux1bit:comb_4
mux1bit:comb_5
mux1bit:comb_6
a[3..0]
b[3..0]
select
result[3..0]
a
b
result
0
1
2
3

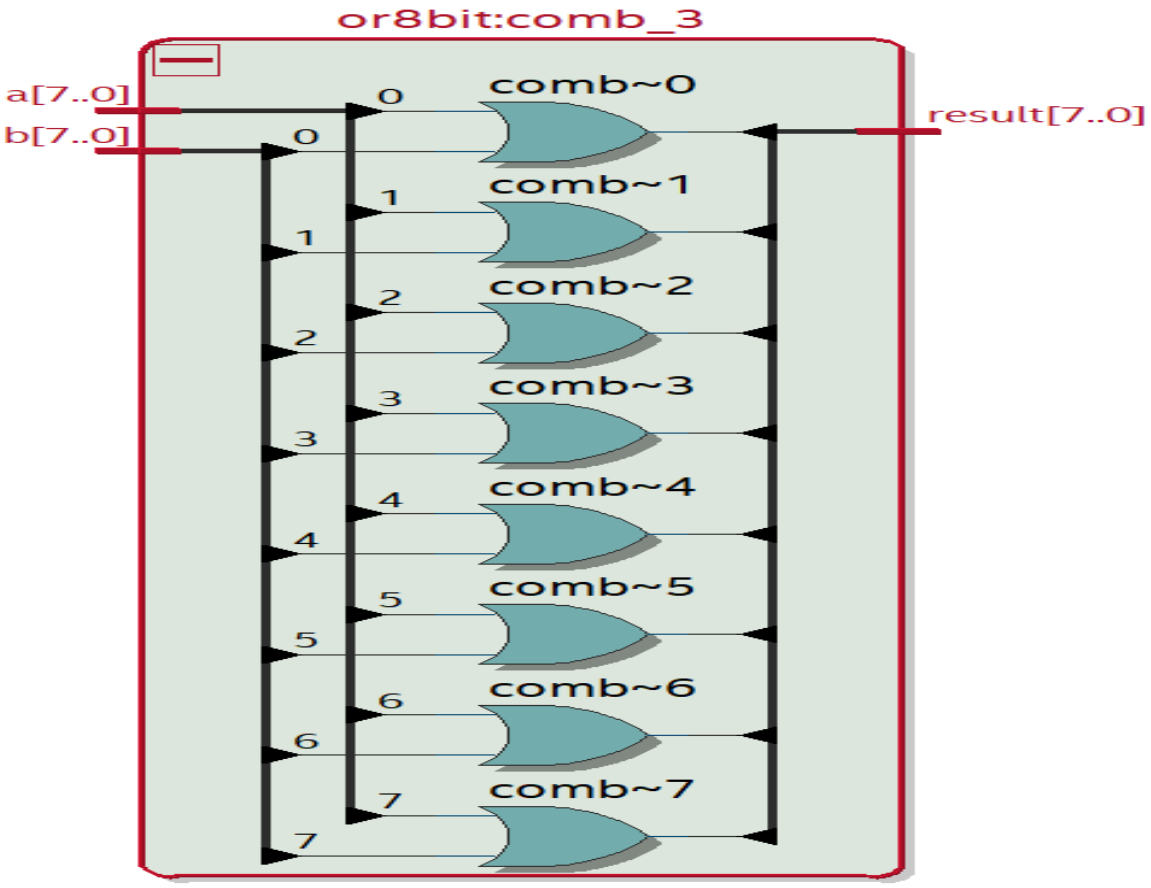

mux32bit:comb_6
mux32bit:comb_5
mux32bit:comb_8
mux32bit:comb_9
mux32bit:comb_3
mux32bit:comb_7
mux32bit:comb_4
result7[31..0]
result6[31..0]
result5[31..0]
result4[31..0]
op[2..0]
result1[31..0]
result0[31..0]
result3[31..0]
result2[31..0]
a[31..0]
b[31..0]
select
result[31..0]
finalResult[31..0]

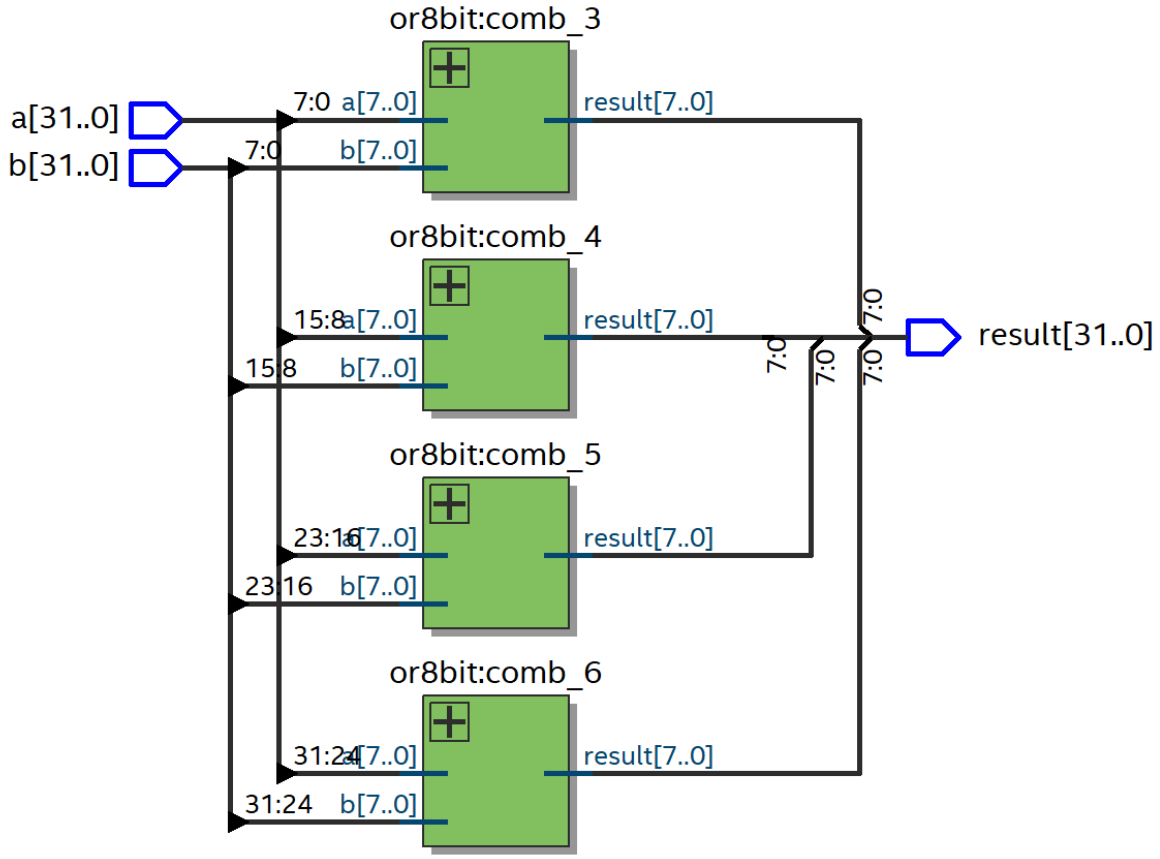

mux16bit:comb_3
mux4bit:comb_3
mux4bit:comb_4
mux4bit:comb_5
mux4bit:comb_6
a[15..0]
b[15..0]
select
a[3..0]
b[3..0]
result[3..0]
result[15..0]
3:0
7:4
11:8
15:12
3:0

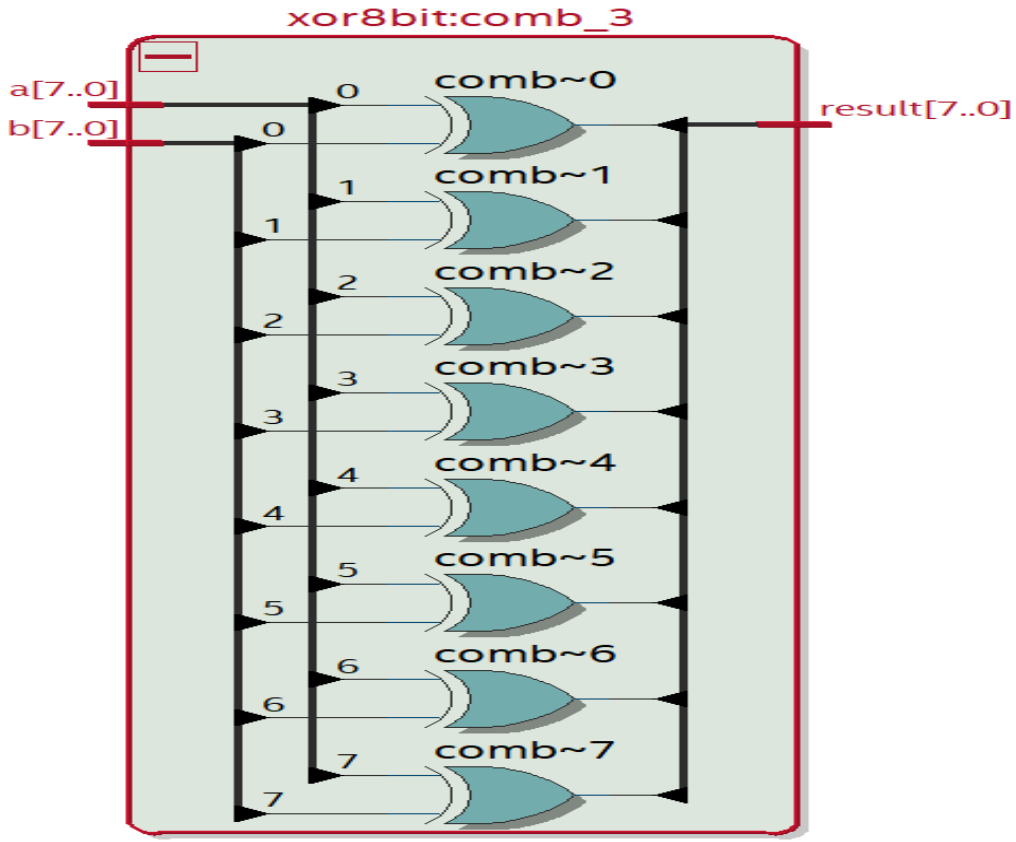

mux64bit:comb_9
mux16bit:comb_3
mux16bit:comb_4
mux16bit:comb_5
mux16bit:comb_6
a[63..0]
b[63..0]
select
a[15..0]
b[15..0]
result[15..0]
result[63..0]
15:0
31:16
47:32
63:48

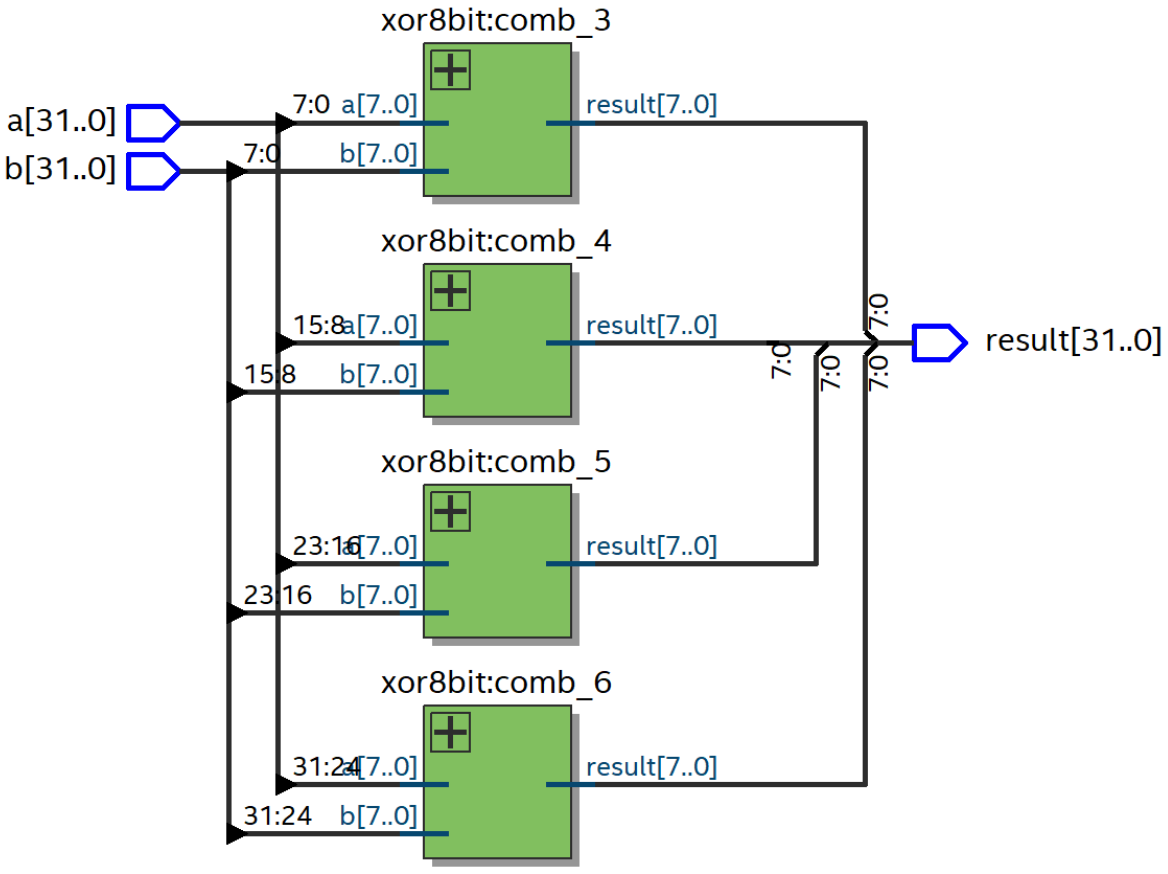

nor8bit:comb_3
a[7..0]
b[7..0]
result[7..0]
comb~0
comb~1
comb~2
comb~3
comb~4
comb~5
comb~6
comb~7
result[0]~not
result[1]~not
result[2]~not
result[3]~not
result[4]~not
result[5]~not
result[6]~not
result[7]~not


nor8bit:comb_3
nor8bit:comb_4
nor8bit:comb_5
nor8bit:comb_6
a[31..0]
b[31..0]
result[31..0]
a[7..0]
b[7..0]
result[7..0]
7:0
15:8
23:16
31:24


or8bit:comb_3
a[7..0]
b[7..0]
result[7..0]
comb~0
comb~1
comb~2
comb~3
comb~4
comb~5
comb~6
comb~7


or8bit:comb_3
a[31..0]
b[31..0]
7:0 a[7..0]
b[7..0]
result[7..0]
or8bit:comb_4
15:8 a[7..0]
15:8 b[7..0]
result[7..0]
or8bit:comb_5
23:16 a[7..0]
23:16 b[7..0]
result[7..0]
or8bit:comb_6
31:24 a[7..0]
31:24 b[7..0]
result[7..0]
result[31..0]


xor8bit:comb_3
a[7..0]
b[7..0]
result[7..0]
comb~0
comb~1
comb~2
comb~3
comb~4
comb~5
comb~6
comb~7


xor8bit:comb_3
a[31..0]
b[31..0]
7:0 a[7..0]
b[7..0]
result[7..0]
xor8bit:comb_4
15:8 a[7..0]
15:8 b[7..0]
result[7..0]
xor8bit:comb_5
23:16 a[7..0]
23:16 b[7..0]
result[7..0]
xor8bit:comb_6
31:24 a[7..0]
31:24 b[7..0]
result[7..0]
7:0
result[31..0]