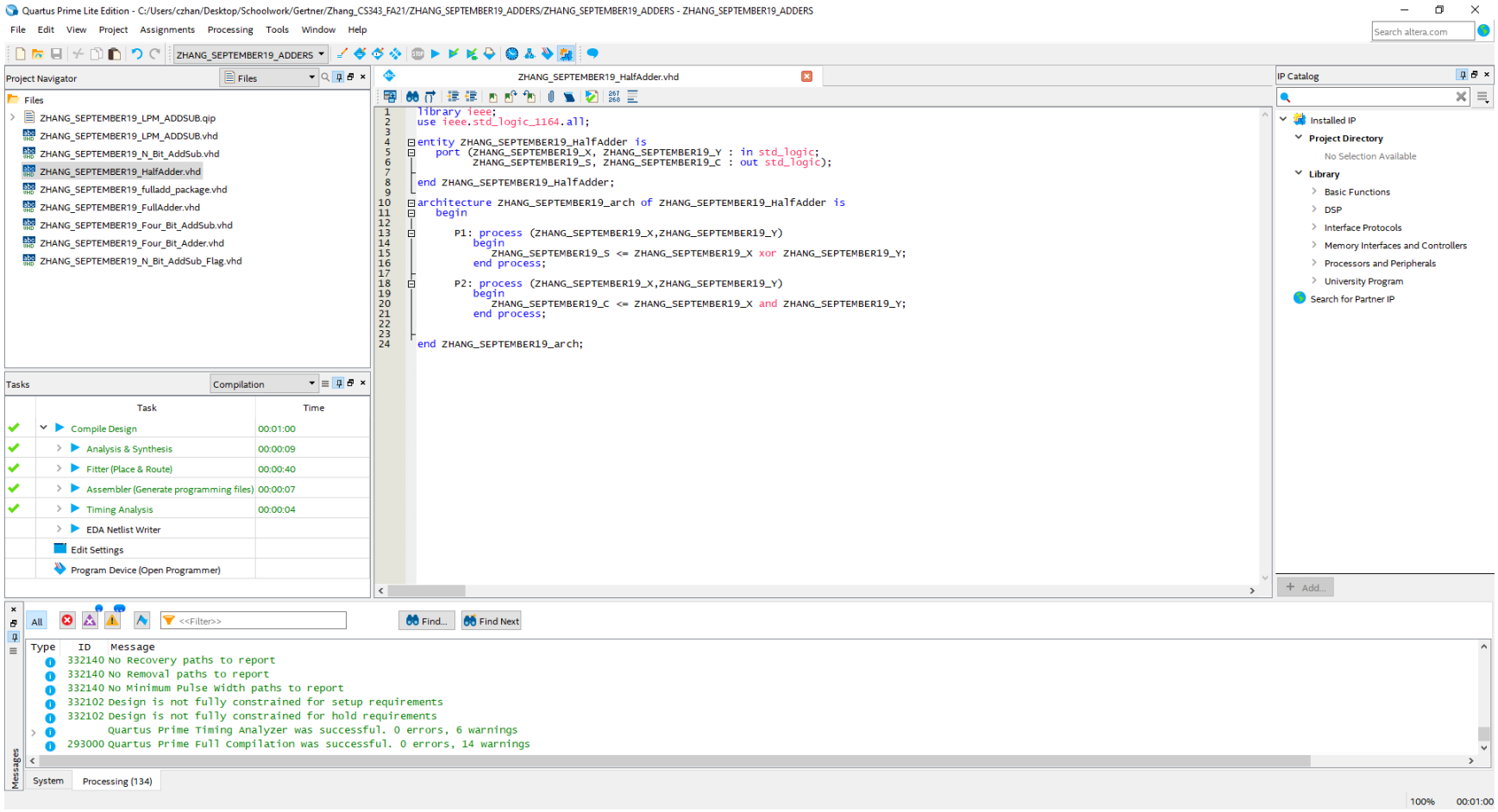
Review Lab : N-Bit Adder

Chue Zhang

Csc343 Fall 2021

Professor Gertner

Table of contents



Half-Adder coded in VHDL using two processes, we XOR inputs X and Y because ‘0’ XOR ‘1’ = 1 which is essentially the sum of 0 and 1. Furthermore, ‘1’ XOR ‘1’ gives 0 which is processed properly in P2 where we do ‘1’ AND ‘1’ which determines the sum as a value with a carry out or excess value.

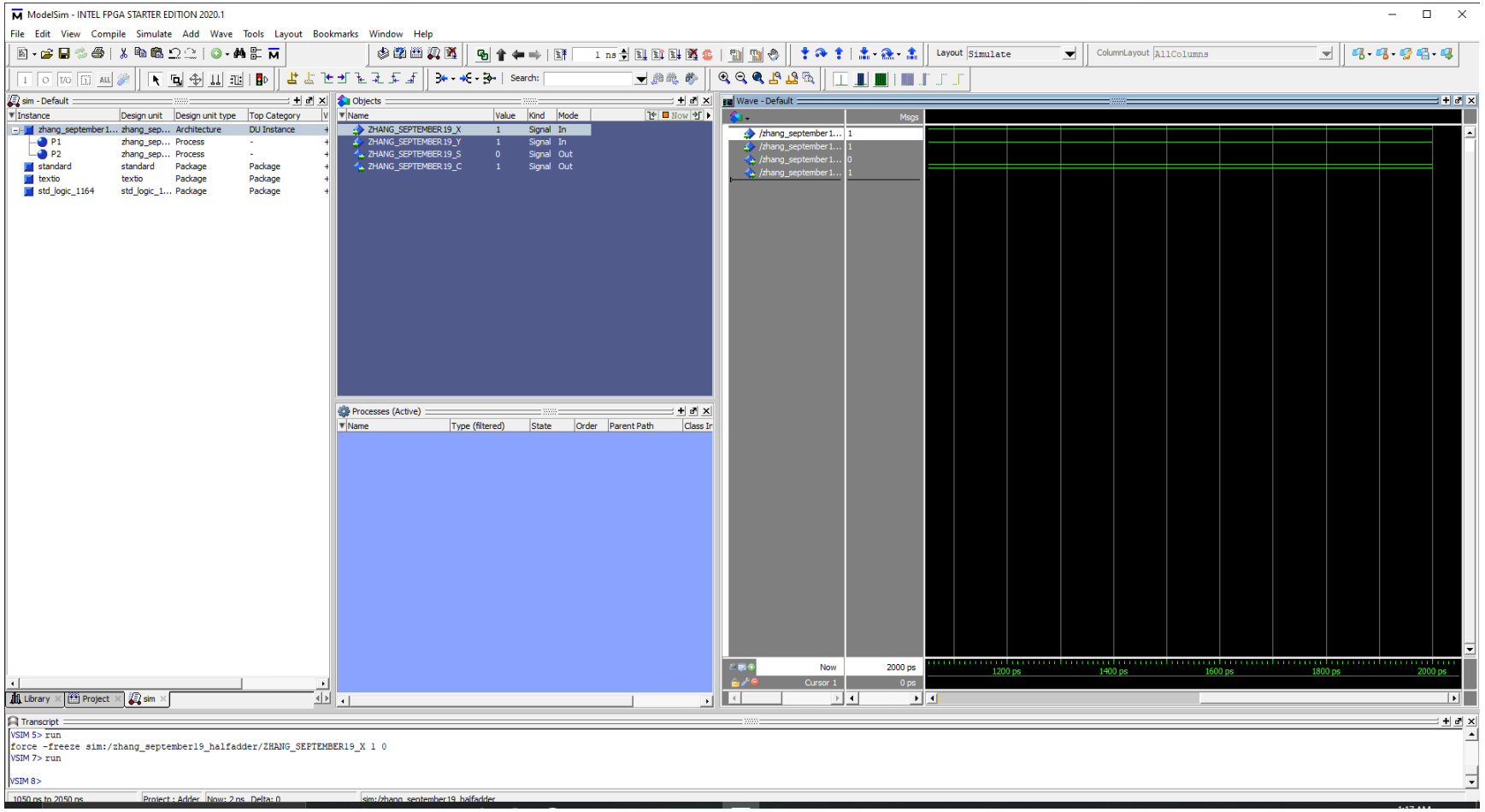
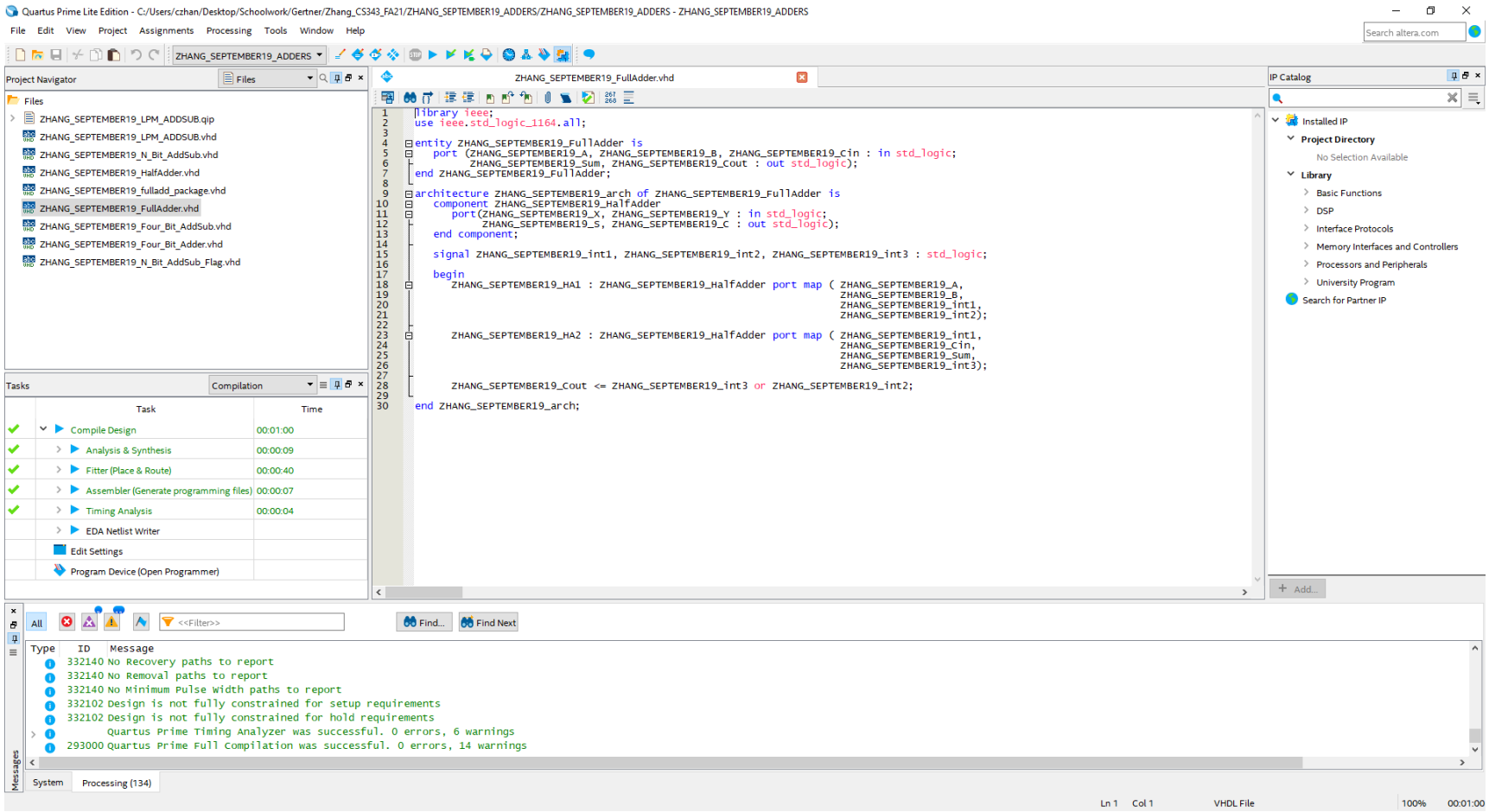
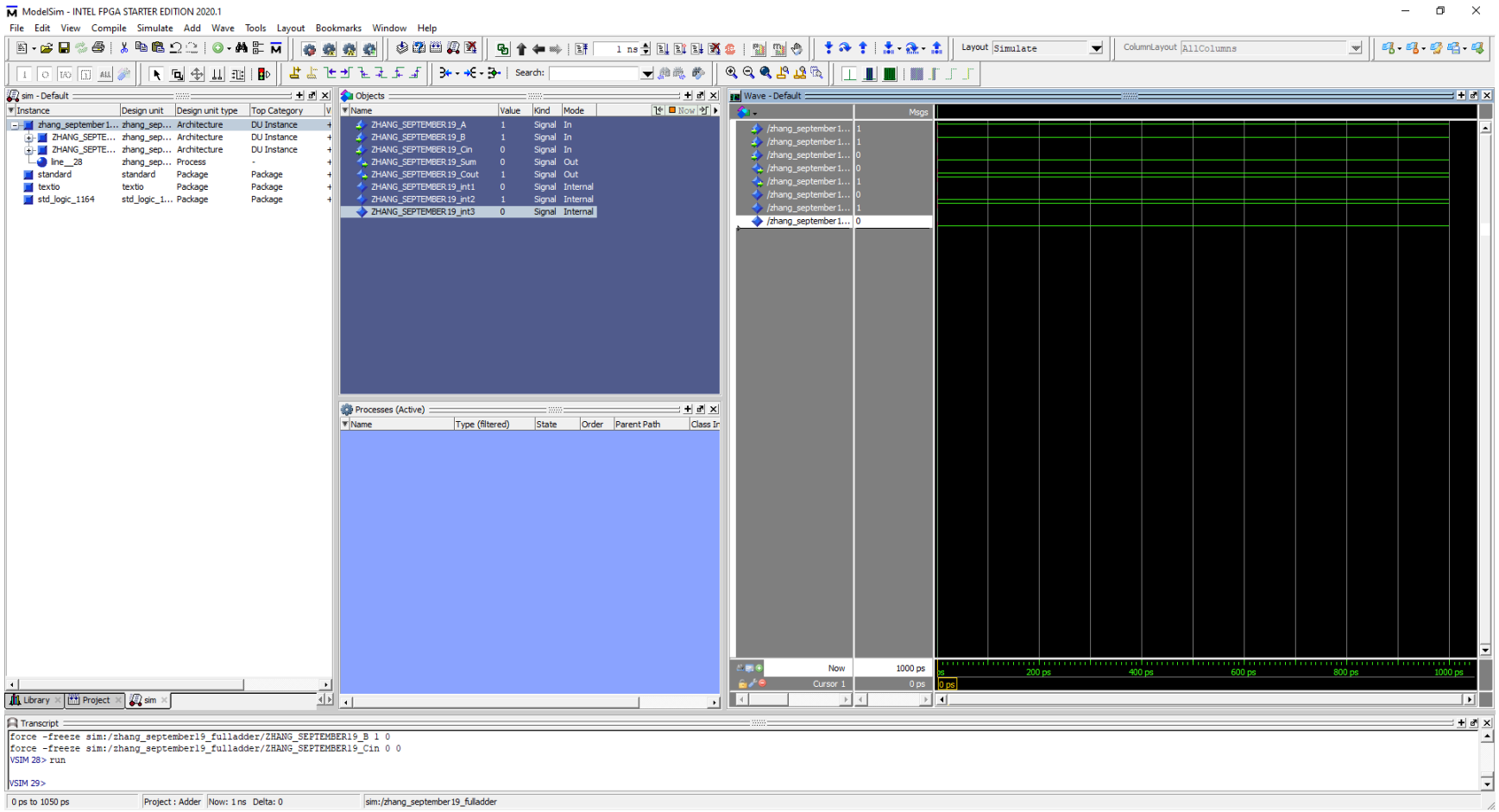


Figure above showcases input values of 1 and 1 which gives a sum value of ‘0’ and a carry out value of ‘1’

Chue Zhang , Task 2: Full Adder



Full 1-bit Adder VHDL code using two half adders in a structural model



Full Adder simulated with input values of ‘1’ and ‘1’ and a carry in value of ‘0’. Sum is ‘0’ with a carry out of ‘1’ which is correct

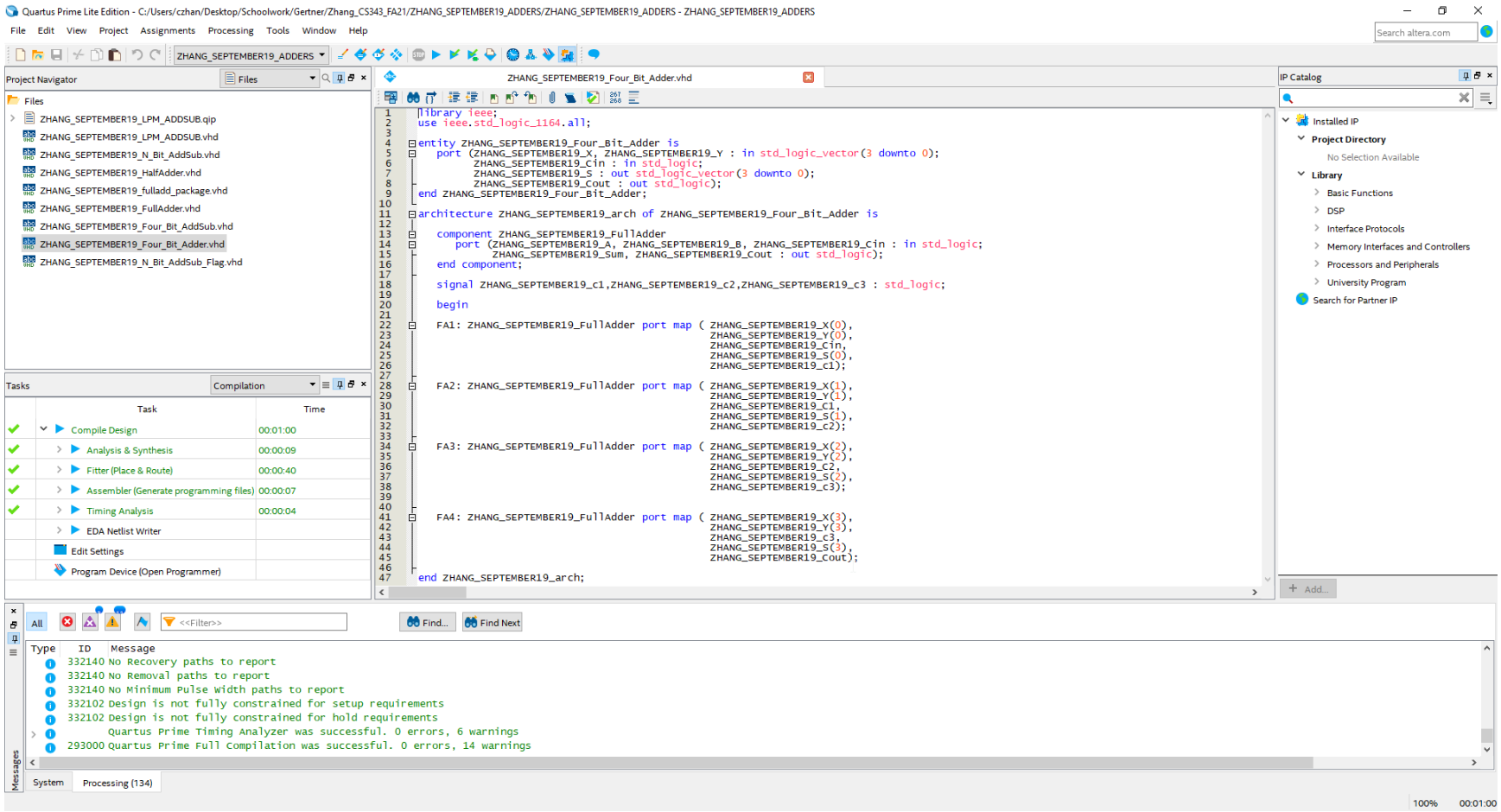


Figure above showcases a Full 4-bit Adder coded in VHDL. The process goes as follows, the first 1- bit Full Adder processes the initial values which then has a carry out value and a sum which is inputted to the next Full Adder and is then processed.

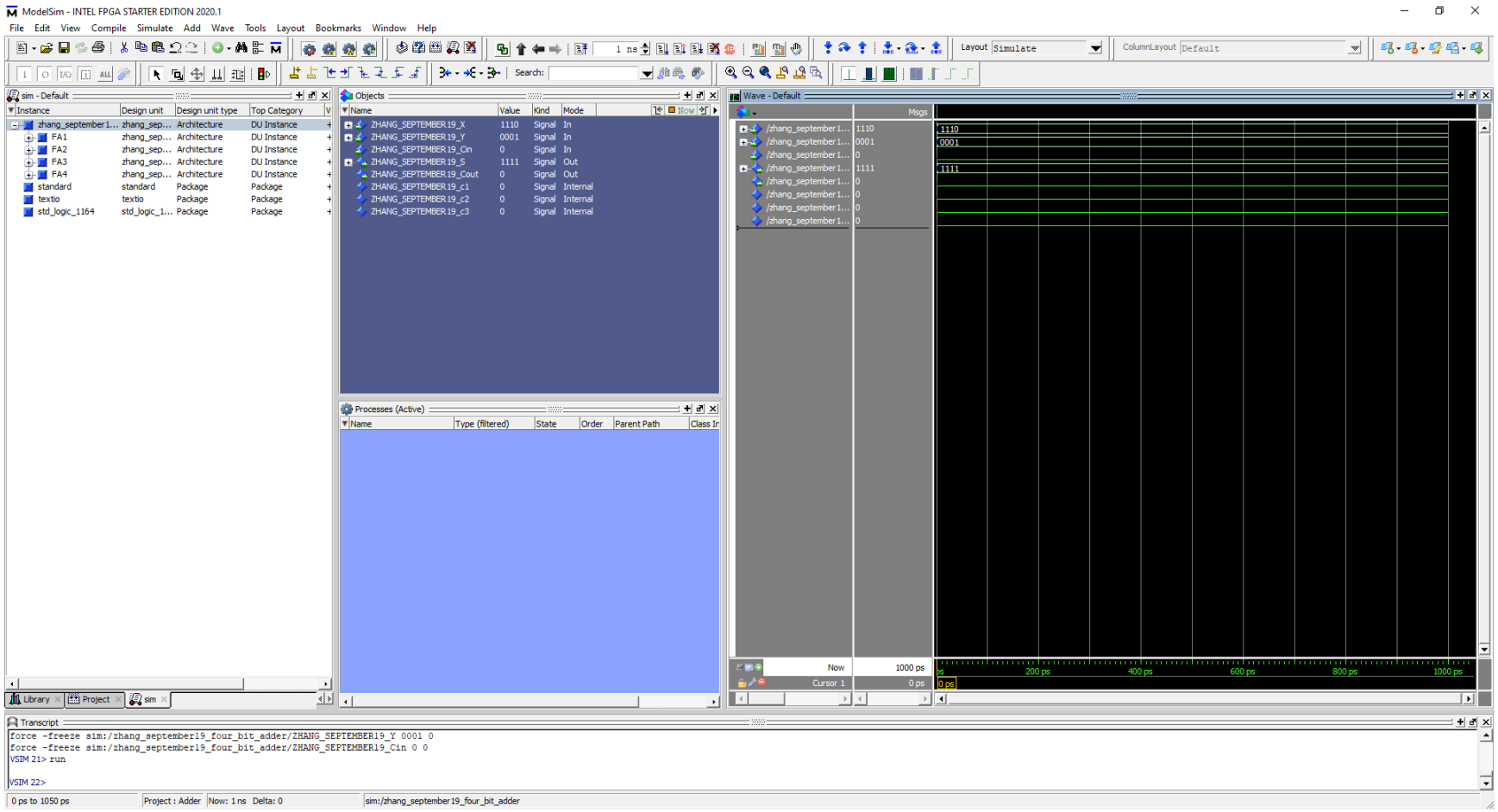


Figure above showcases Full 4-bit Adder simulation using ModelSim with input values of ‘1101’ and ‘0001’ with ‘0’ as cin. The sum output for these two input values are 1111 with a carry out of 0 which is correct.

Chue Zhang , Task 4 : Full 4-bit Adder/Sub

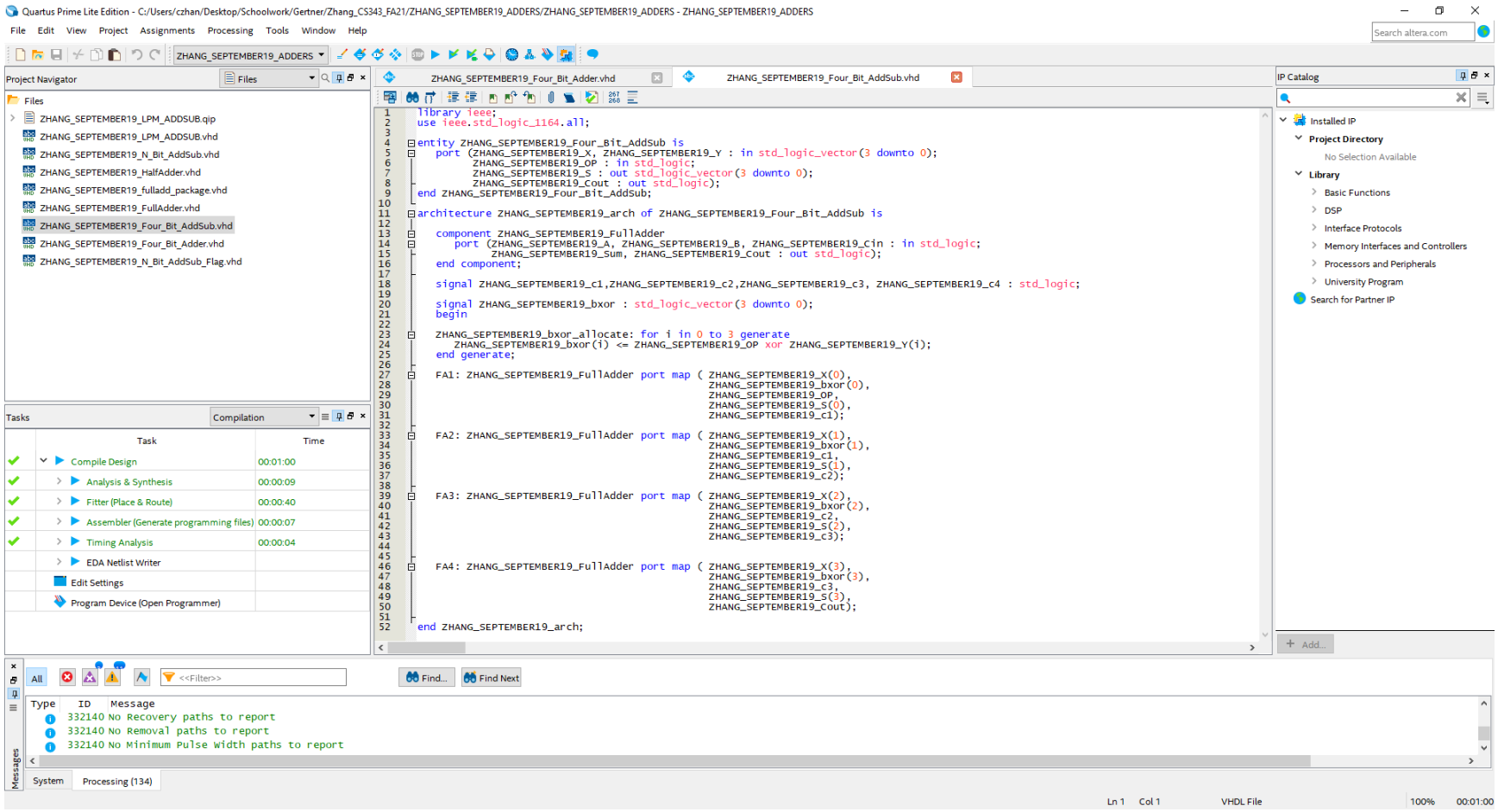


Figure above showcases the VHDL code of a Full 4-bit Adder/Subtractor which is similar to the 4-bit Adder but has an extra input variable that helps decide whether the machine should add or subtract. If OP = ‘1’ then we subtract if OP = ‘0’ then we add.

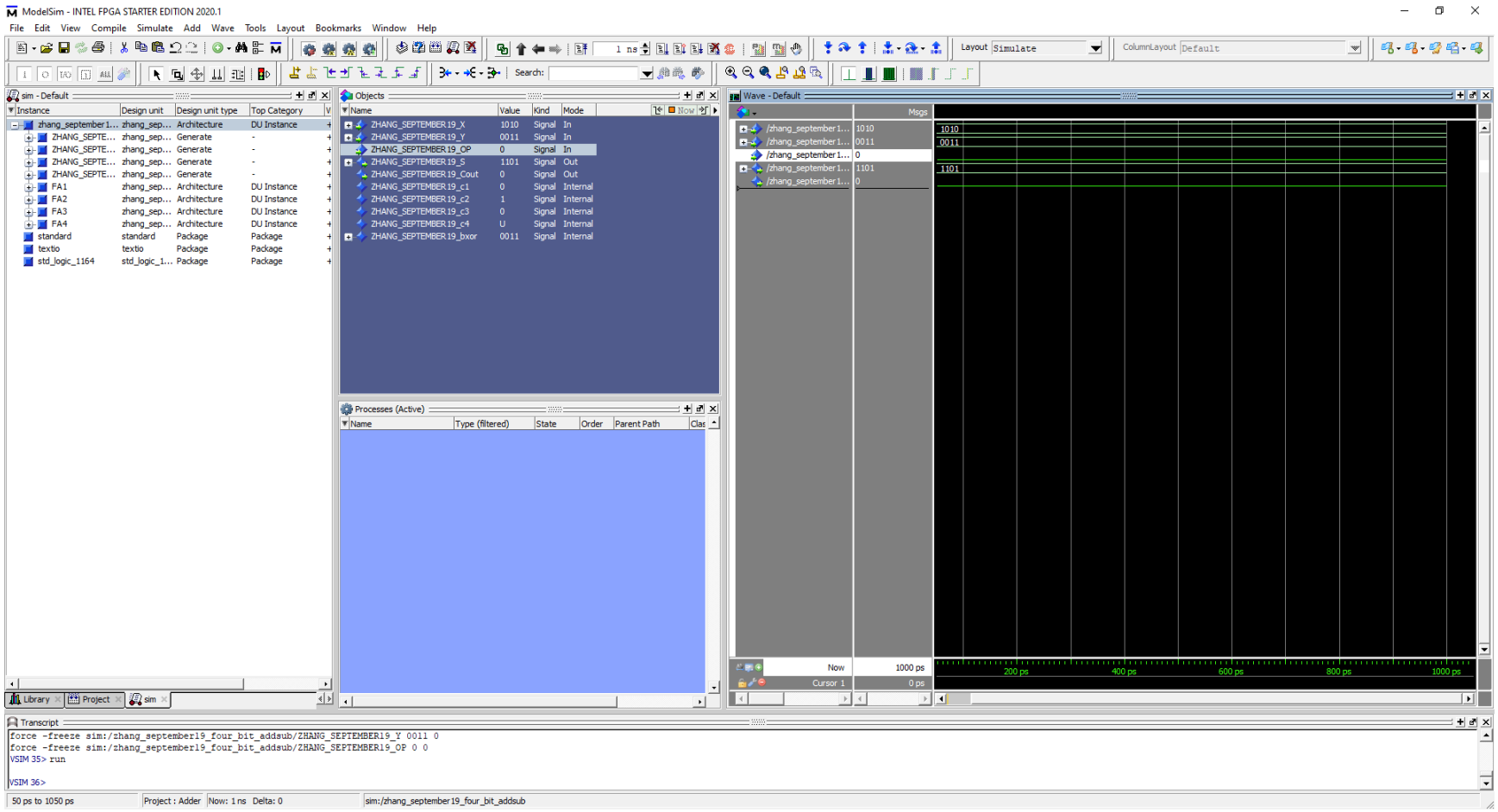


Figure above showcases OP = ‘0’ which means we add the two input values of ‘1010’ and ‘0011’ which gives a sum output of 1101 with a carry out of 0, which is correct.

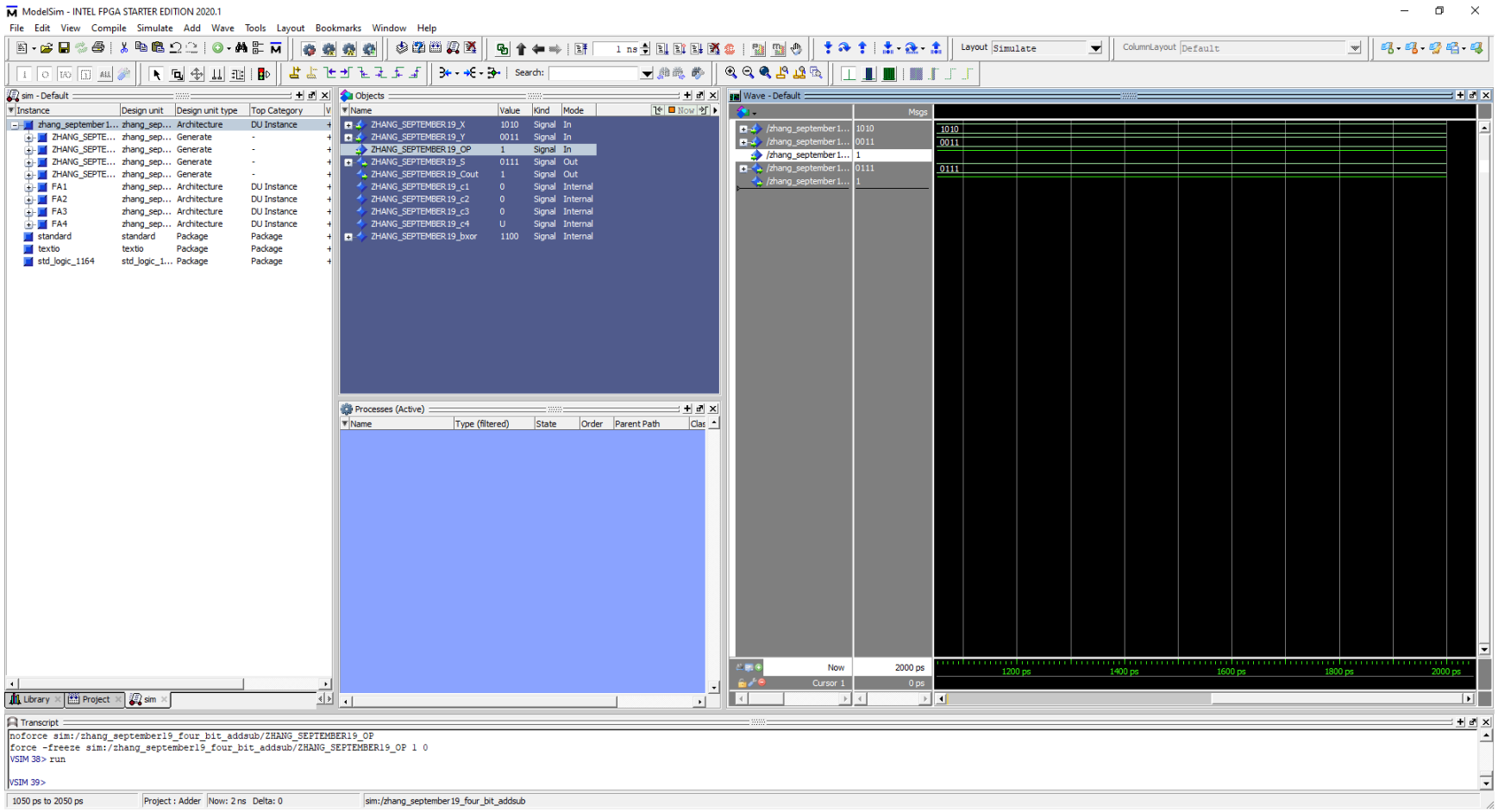
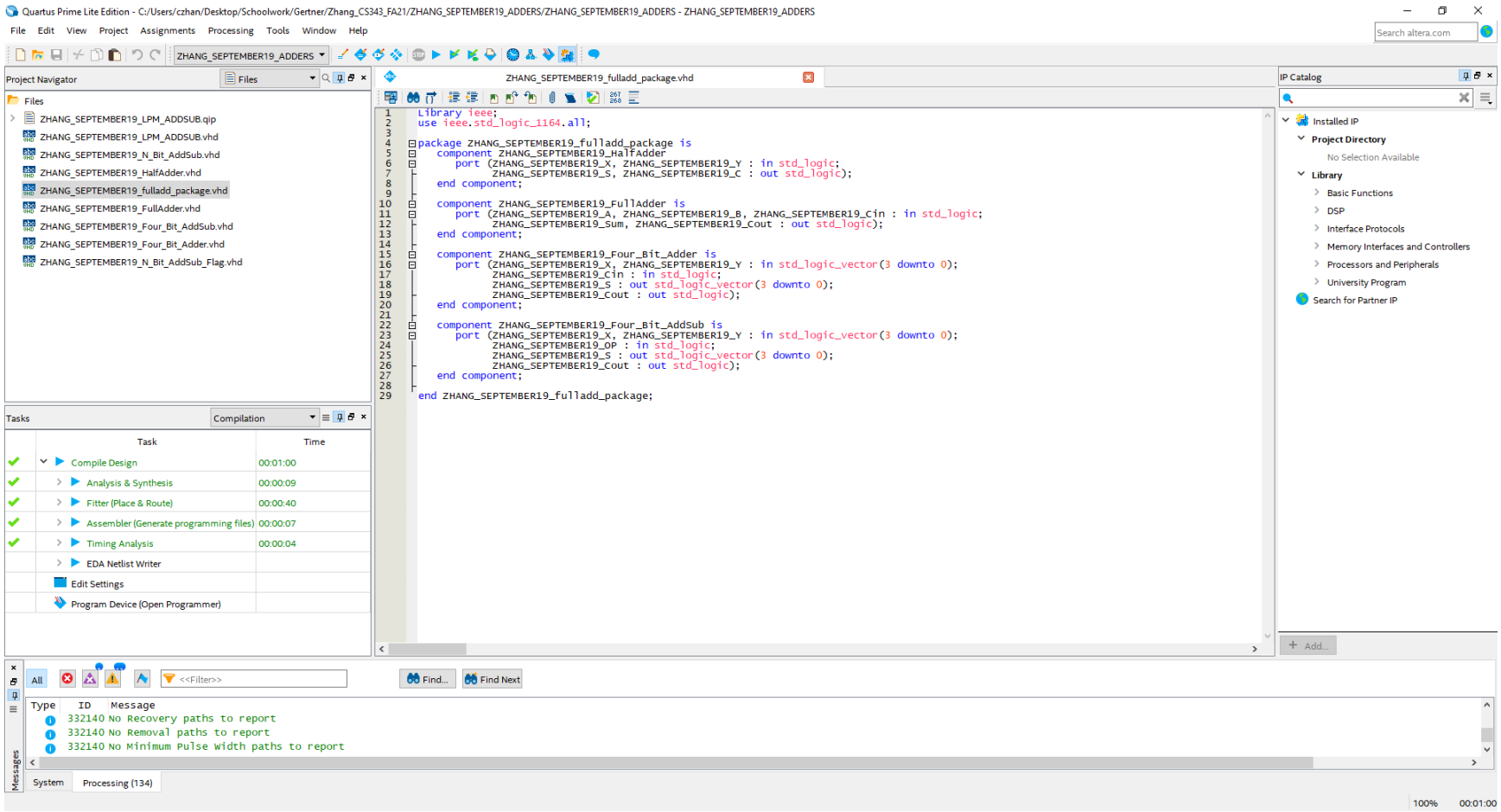


Figure above showcases OP = ‘1’ which means we subtract the two values ‘1010’ and ‘0011’ which gives a sum output of ‘0111’ which is correct

Chue Zhang , Task 5 : Full 4-bit Adder Package



Full Adder Package which houses the 4-bit Adder/Subtractor, 4-bit Adder, 1-bit Full Adder and the Half Adder. Further note is that package CANNOT be compiled as it has no entity but can still perform its function as long as it exists within the files.

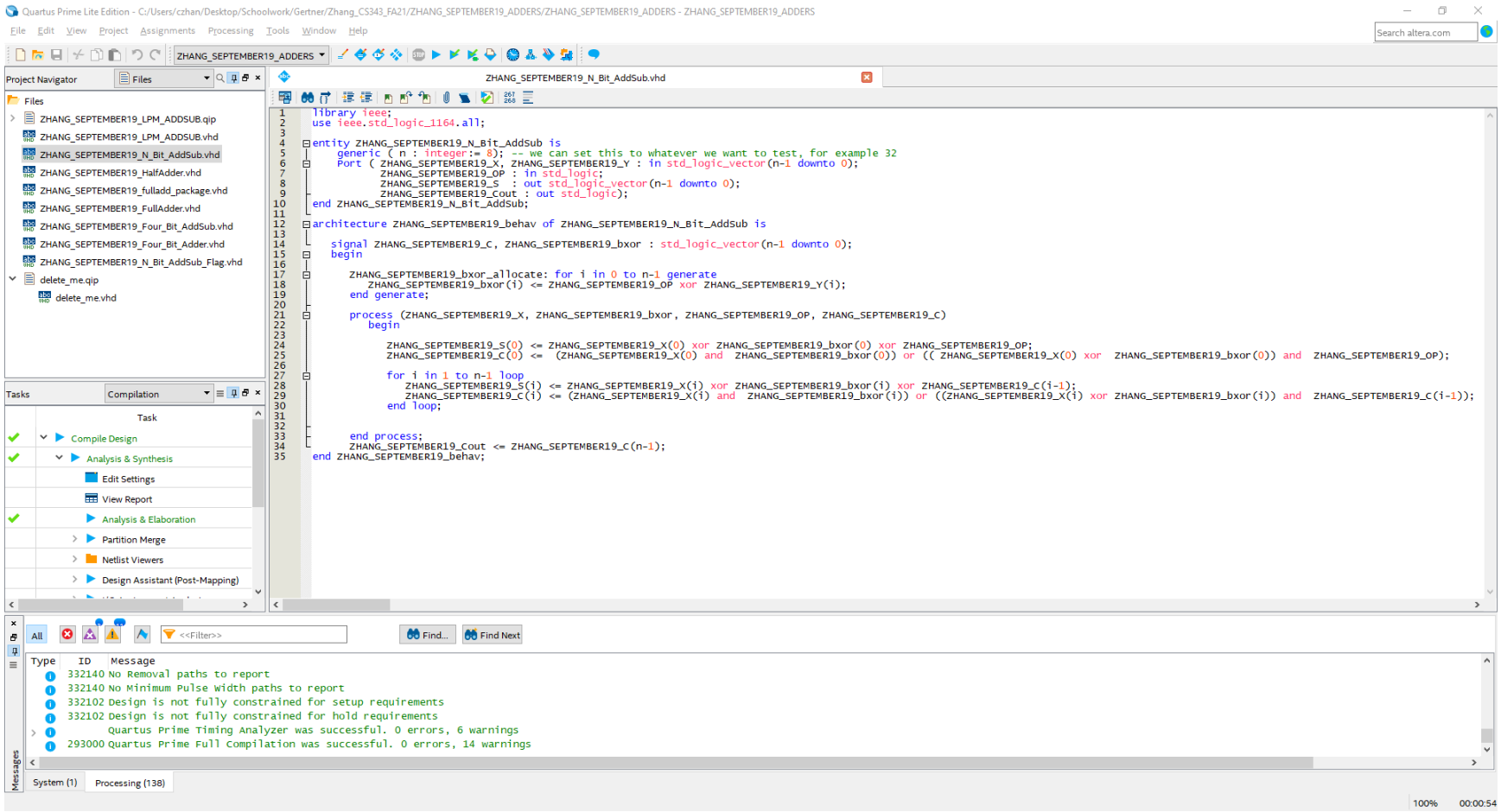
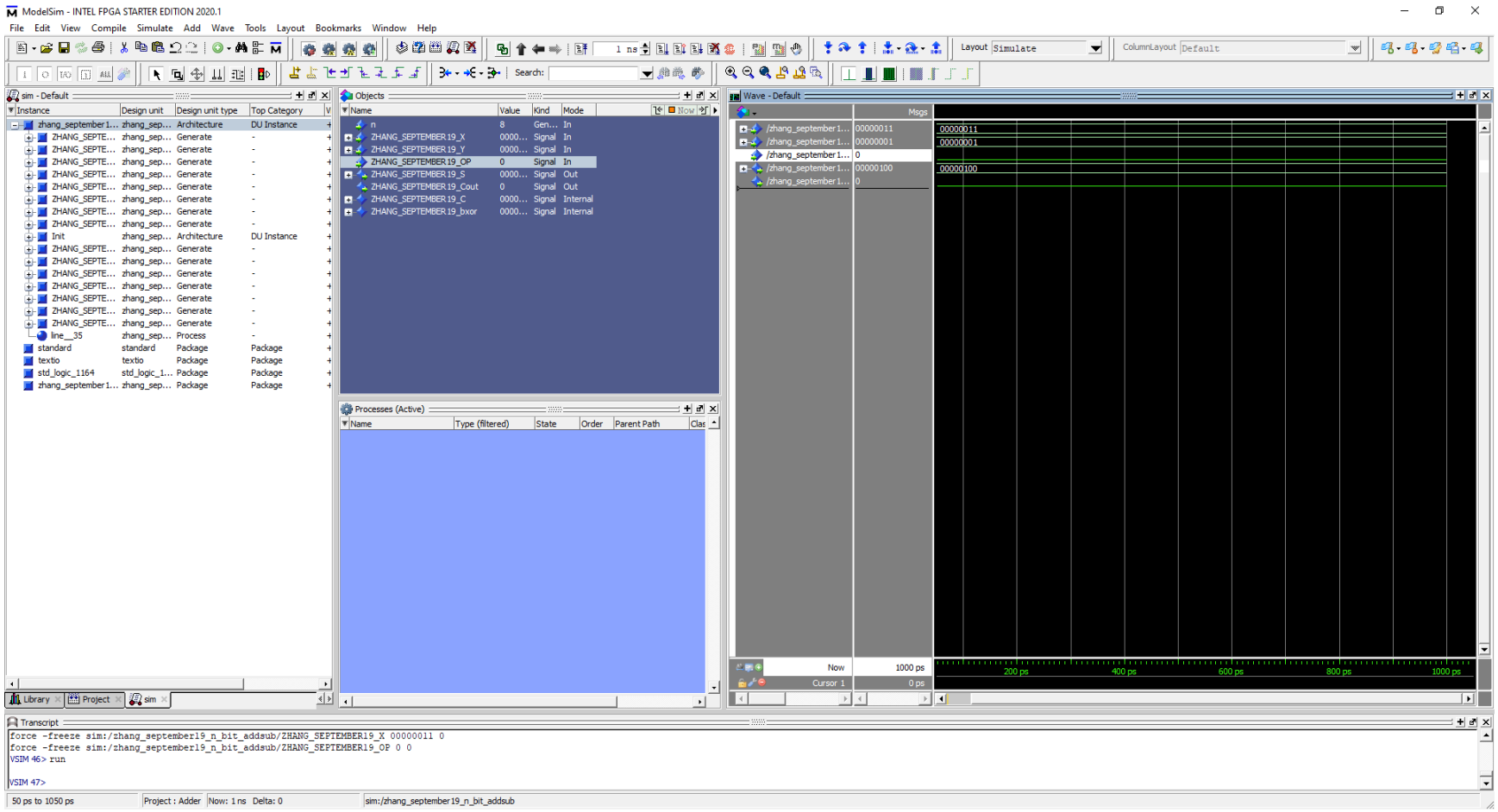
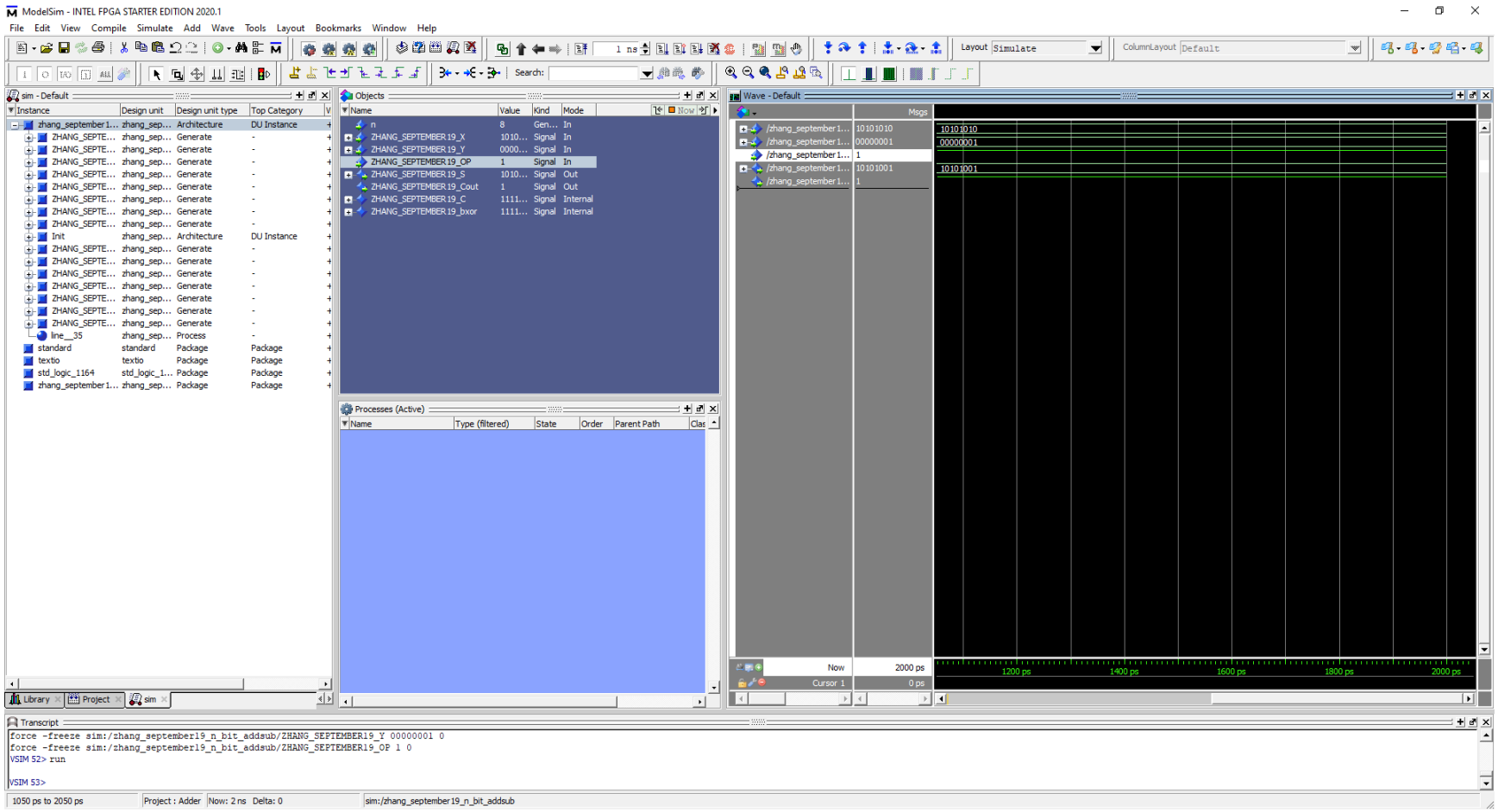


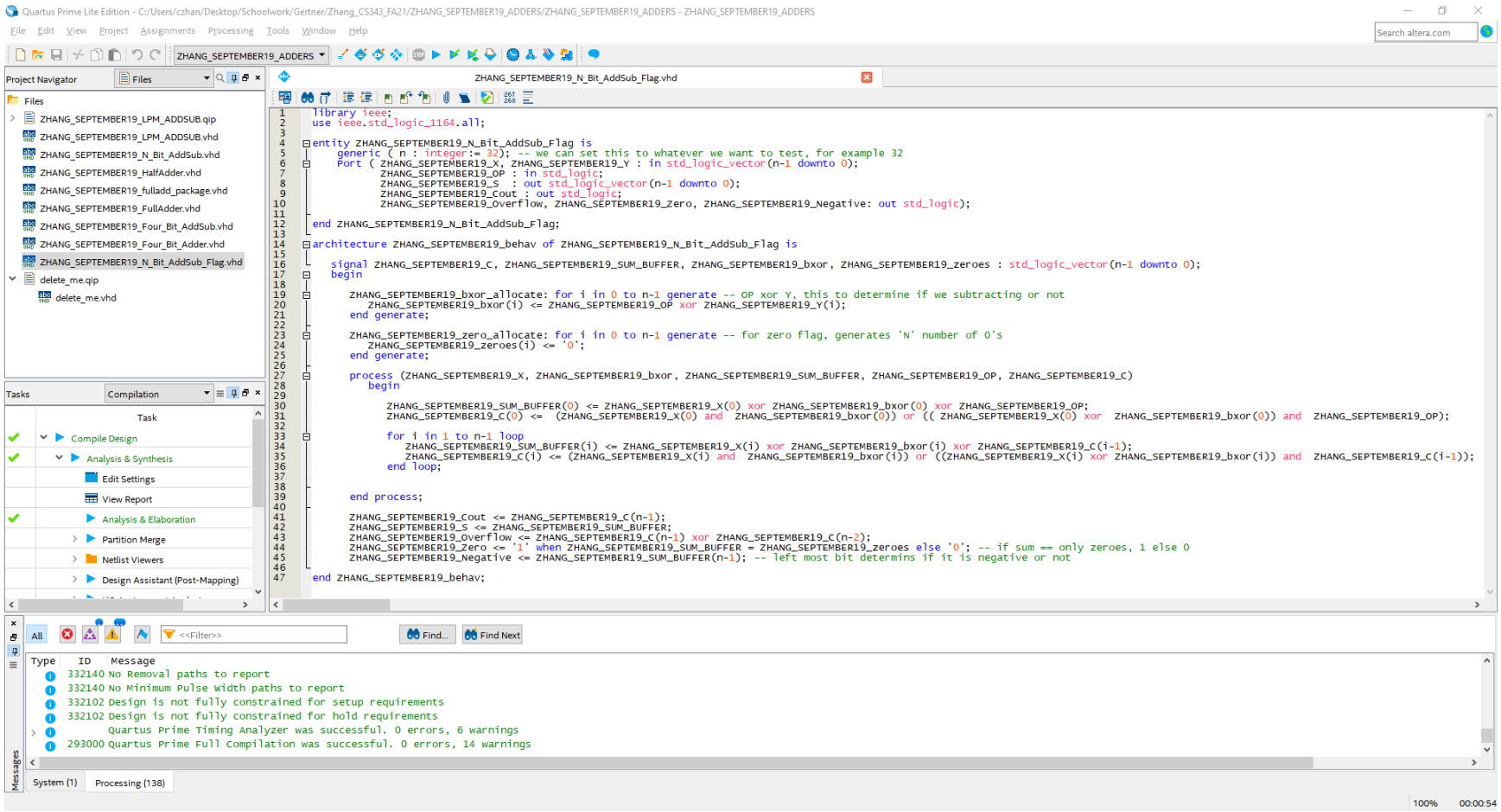
Figure above showcases the VHDL code for N-bit Adder/Subtractor which uses a generic input ‘n’ which we can set to any value. What happens next is the n value helps set the bit count of appropriate variables. Here we are using n = 8.



The figure above showcases the N-bit Adder/Subtractor with an N value of 8. We use input values of ‘00000011’ and ‘00000001’ with an OP value of ‘0’ so we add the two inputs together. The sum output is ‘00000100’ with no carry out which is correct.



The figure above showcases the N-bit Adder/Subtractor with an N value of 8. We use input values of ‘10101010’ and ‘00000001’ with an OP value of ‘1’ so we subtract the two inputs together. The sum output is ‘10101001’ with no carry out which is correct.



VHDL code for an N-bit Adder/Subtractor with Overflow, Negative and Zero flags.

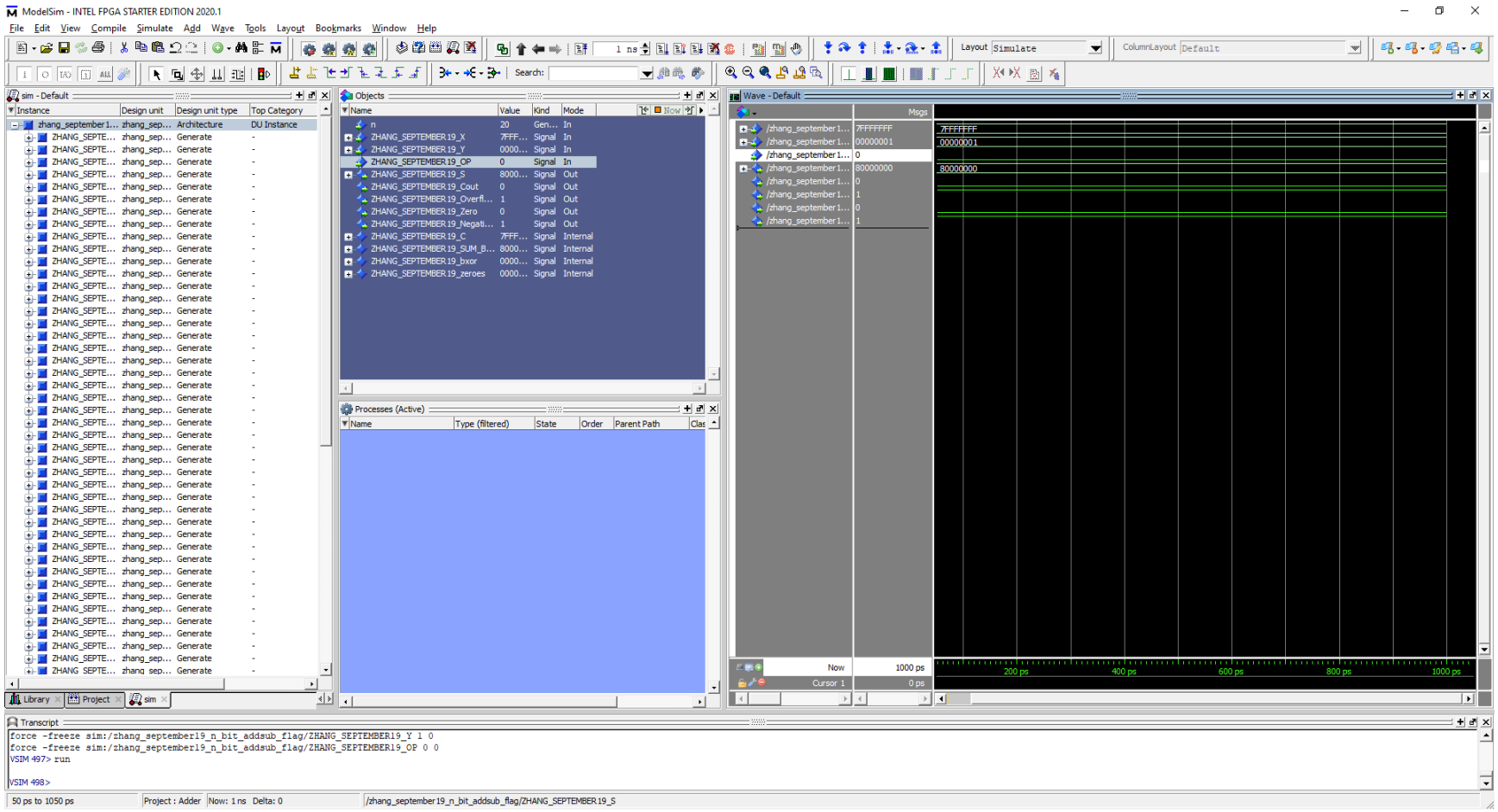


Figure above demonstrates the overflow flag and the negative flag triggering.

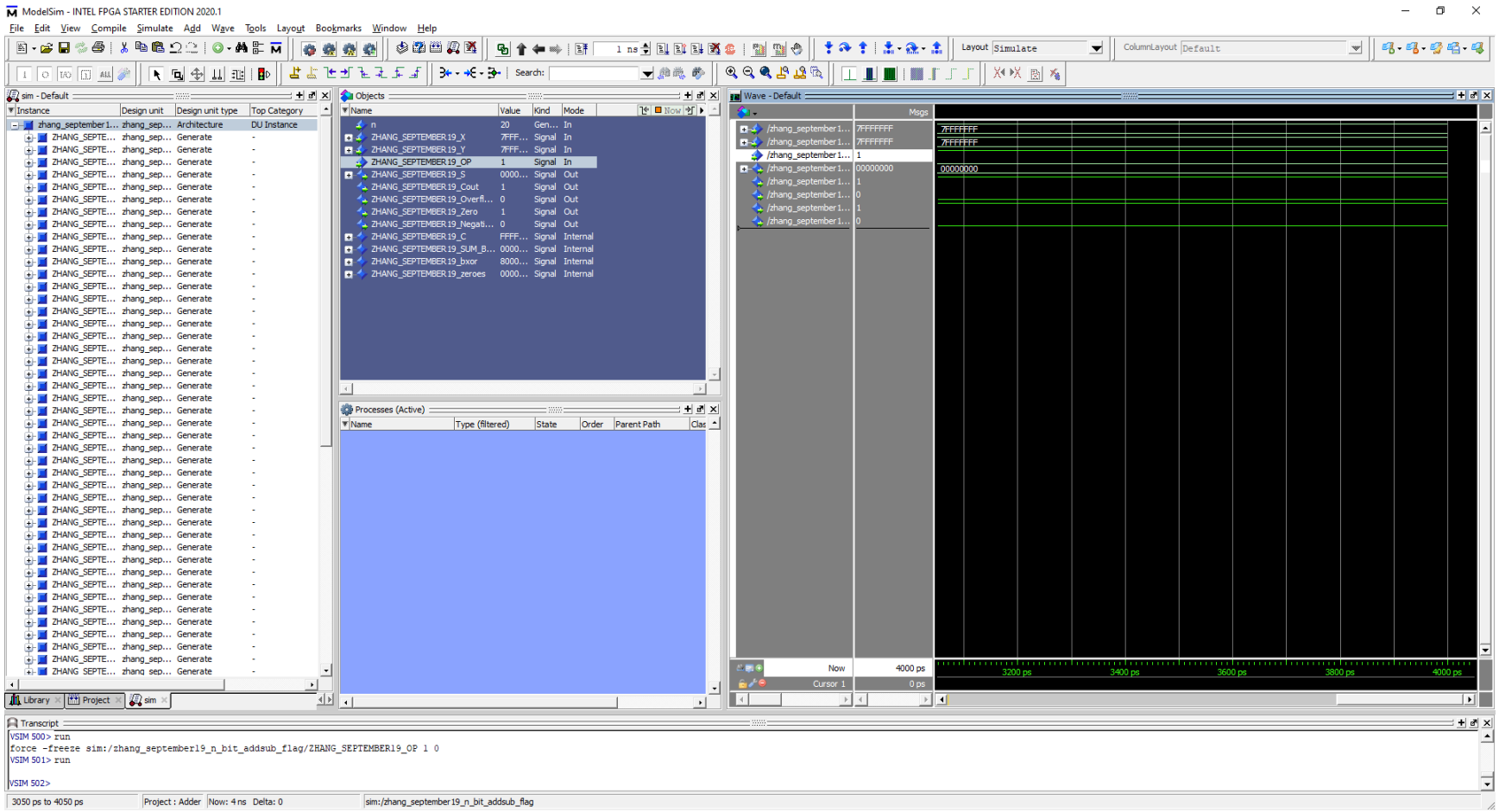
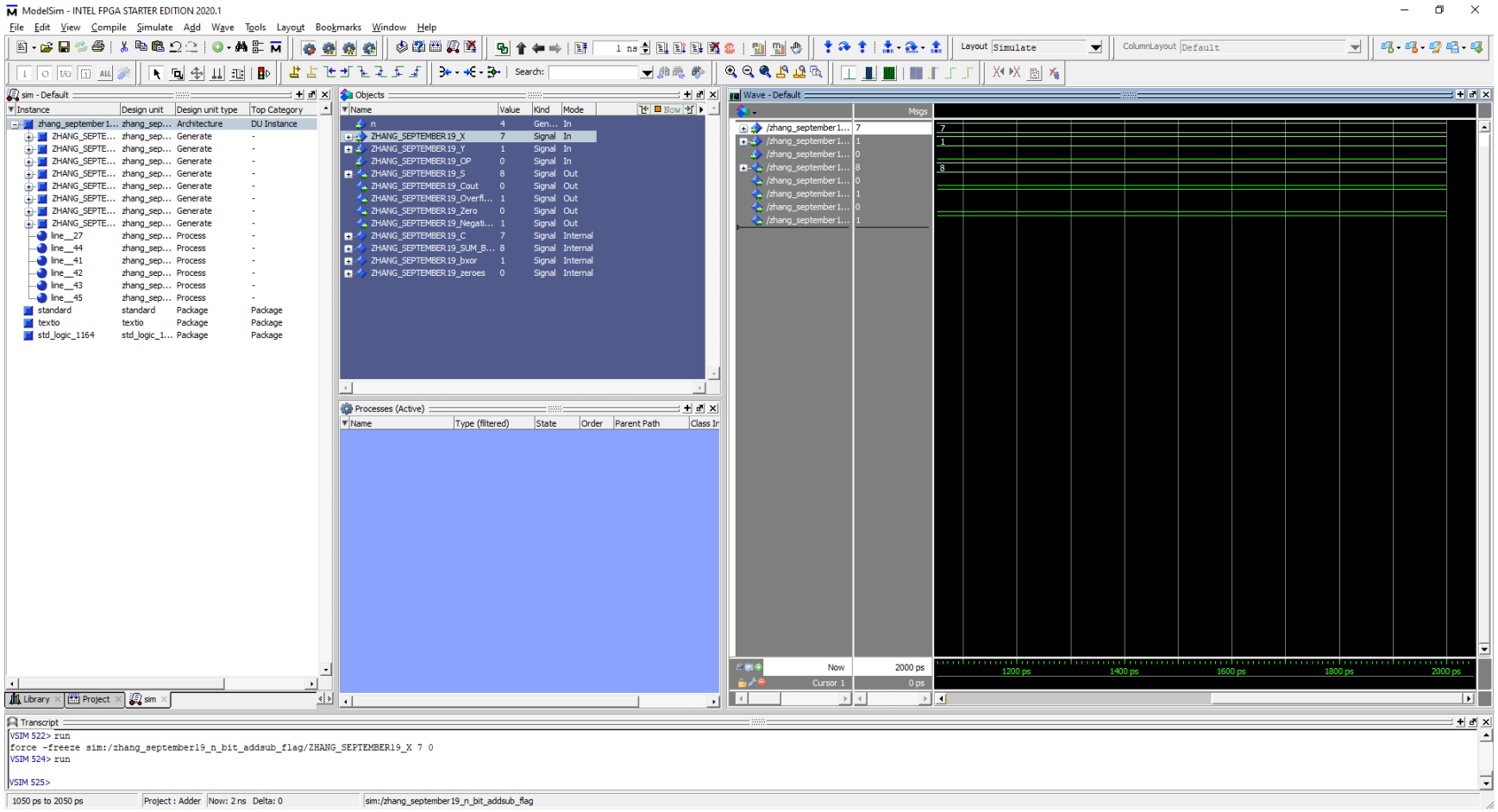


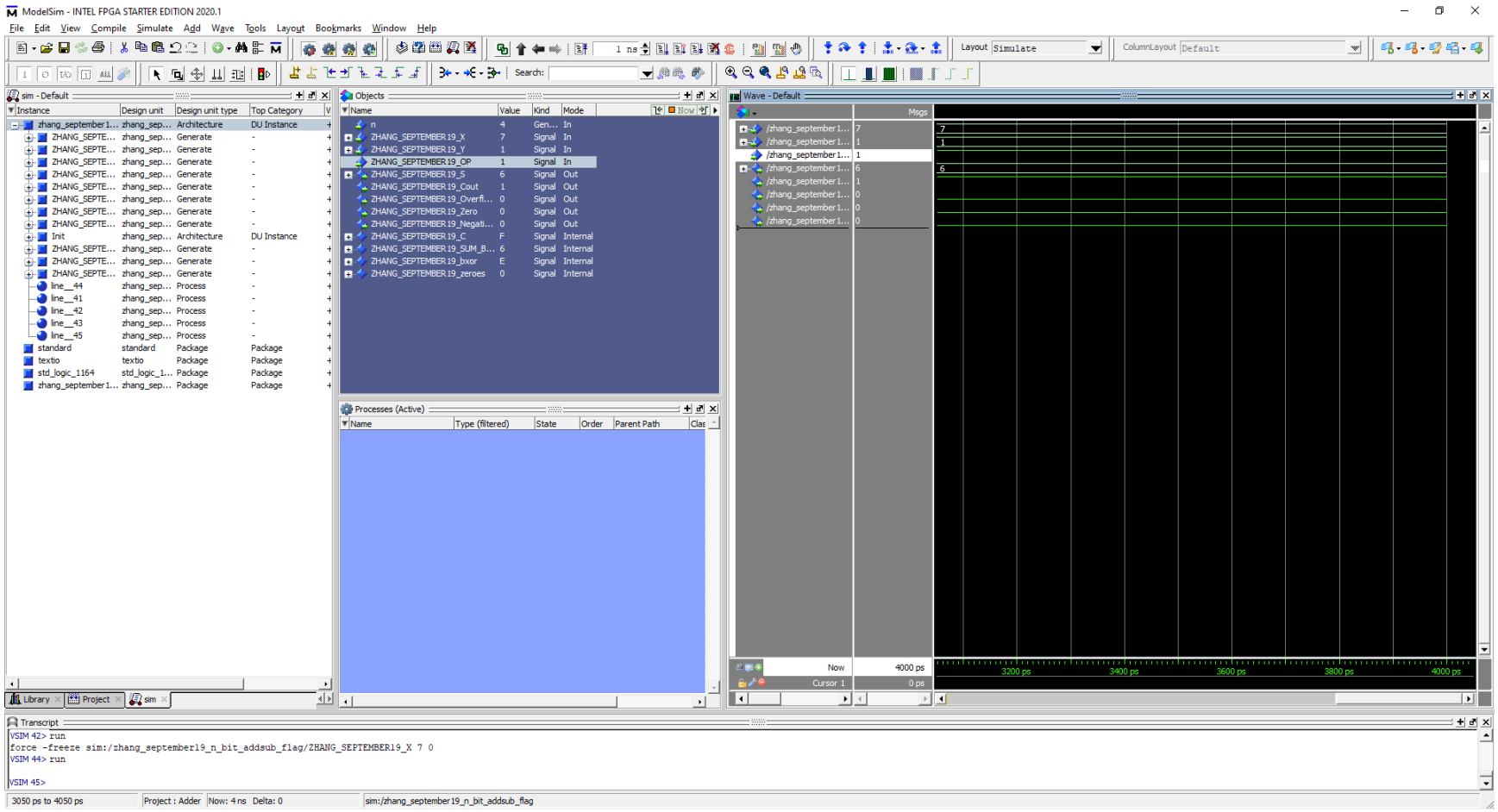
Figure above demonstrates the zero flag being triggered when two values of equal size cancel each other out through subtraction



Task 8a. Most positive N bit + 1, N = 4

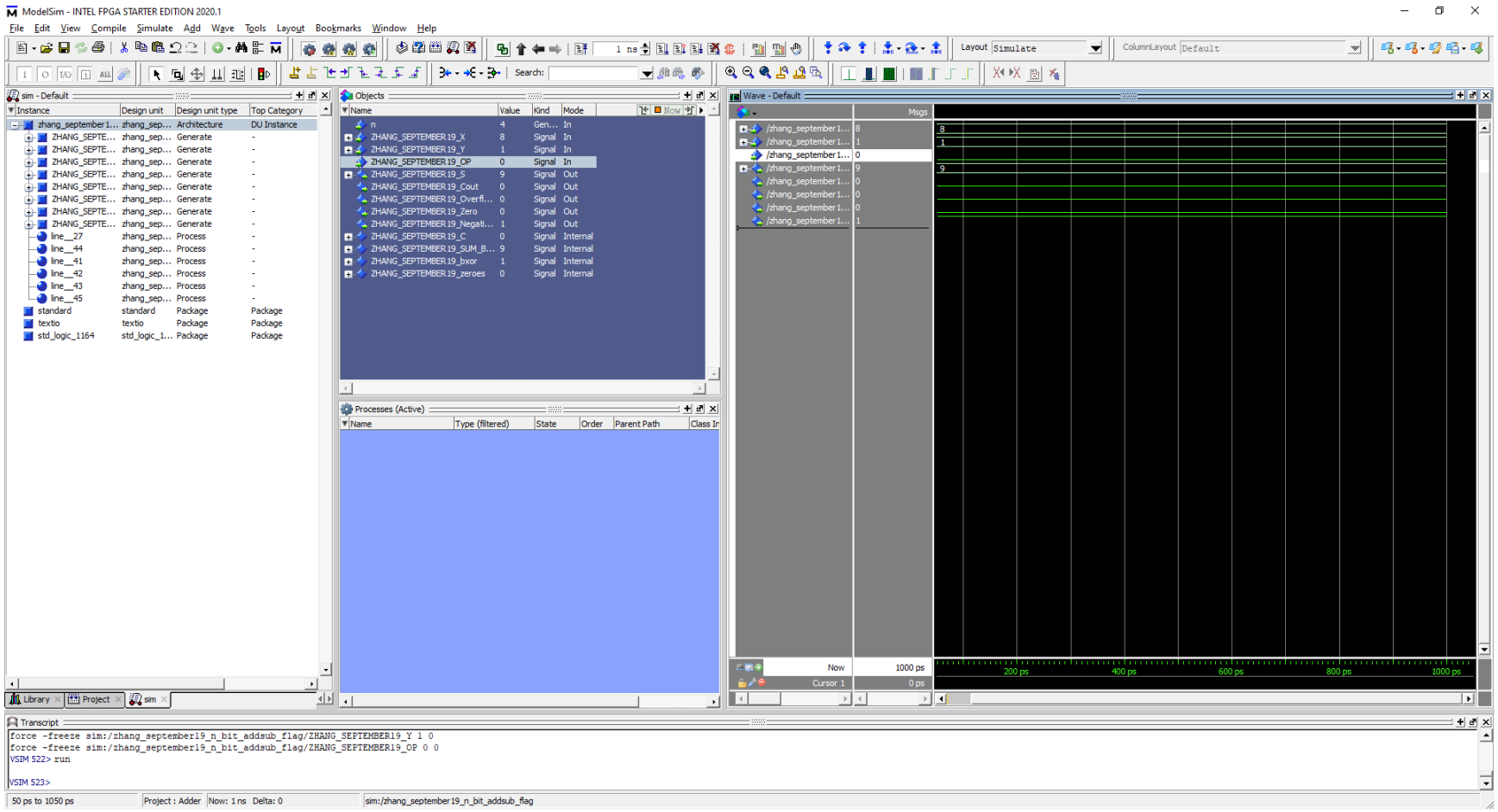
Most positive N bit = 7, 7 + 1 = 8 which ends up becoming a negative value because the most significant bit is now ‘1’. Overflow has also been detected due to positive + positive = negative

Chue Zhang , Task 8 : Overflow, Zero and Negative Flags verify n=4 and n=32



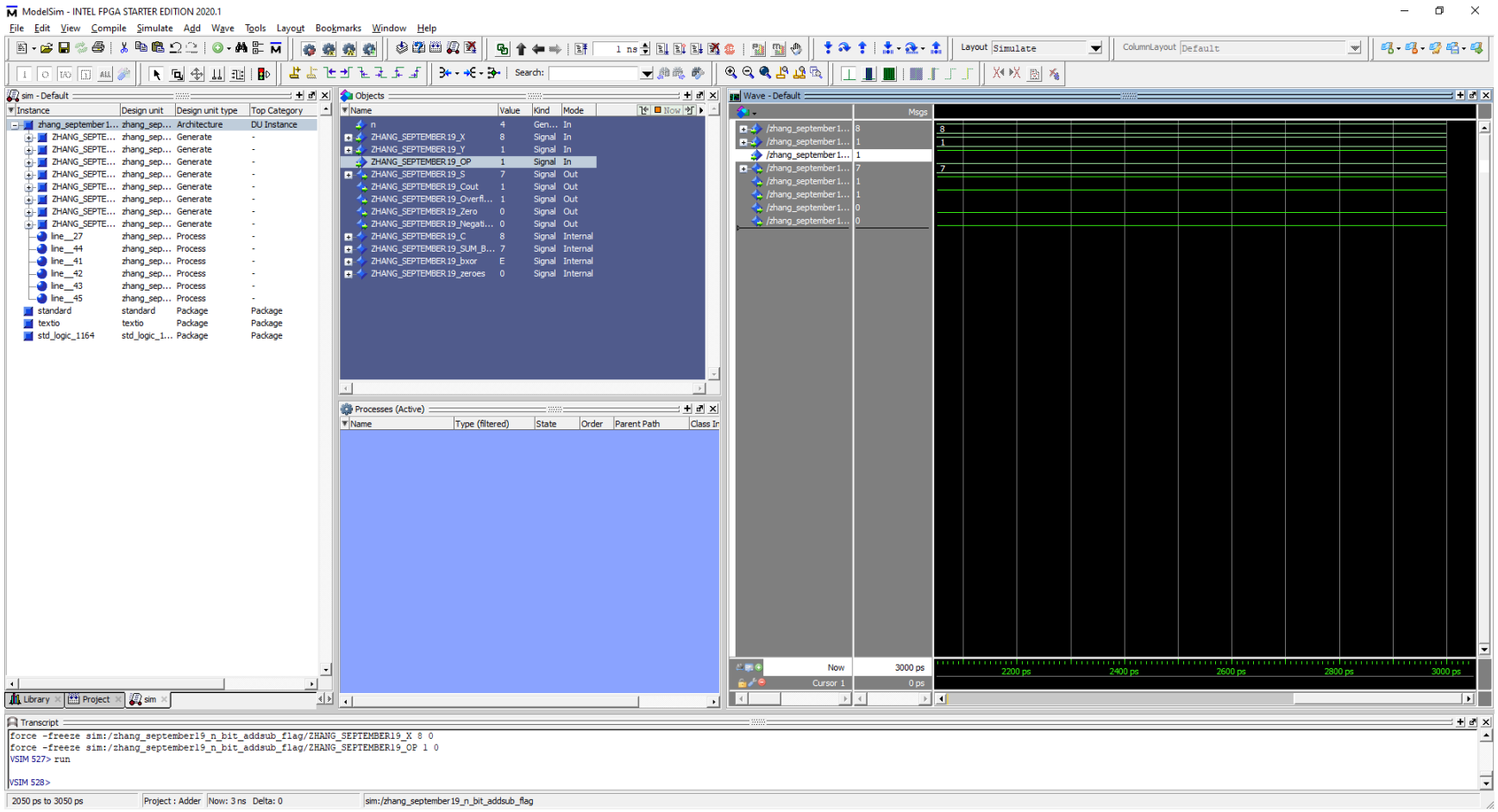
Task 8b. Most positive N bit - 1, N = 4

7 – 1 = 6 which does not trigger any of the flags. 6 is not negative, not zero and does not have overflow because two positives did not become a negative



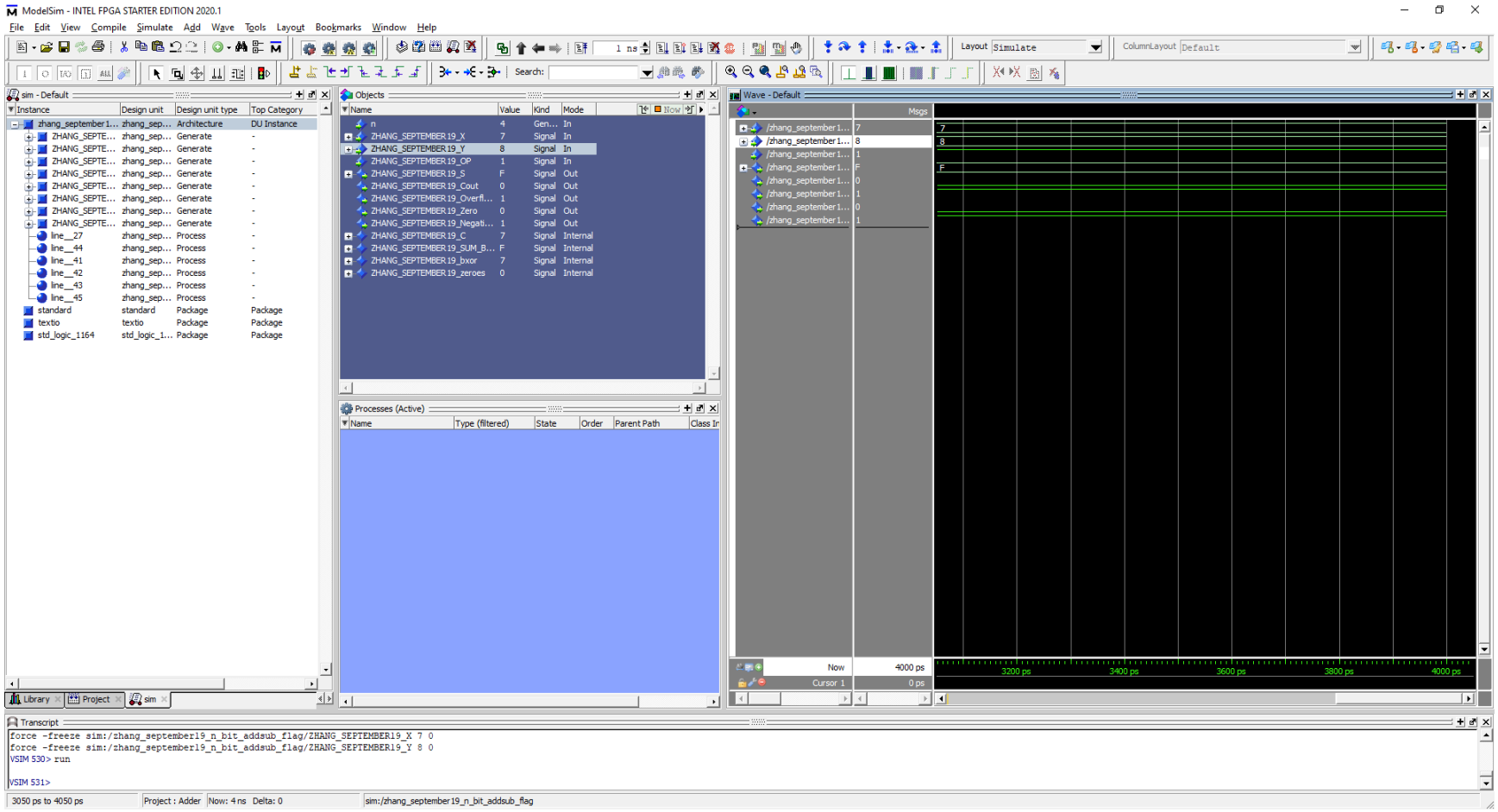
Task 8c. Most Negative N bit + 1, N = 4

Most negative bit + 1 is 9 which triggers only the negative flag and nothing else



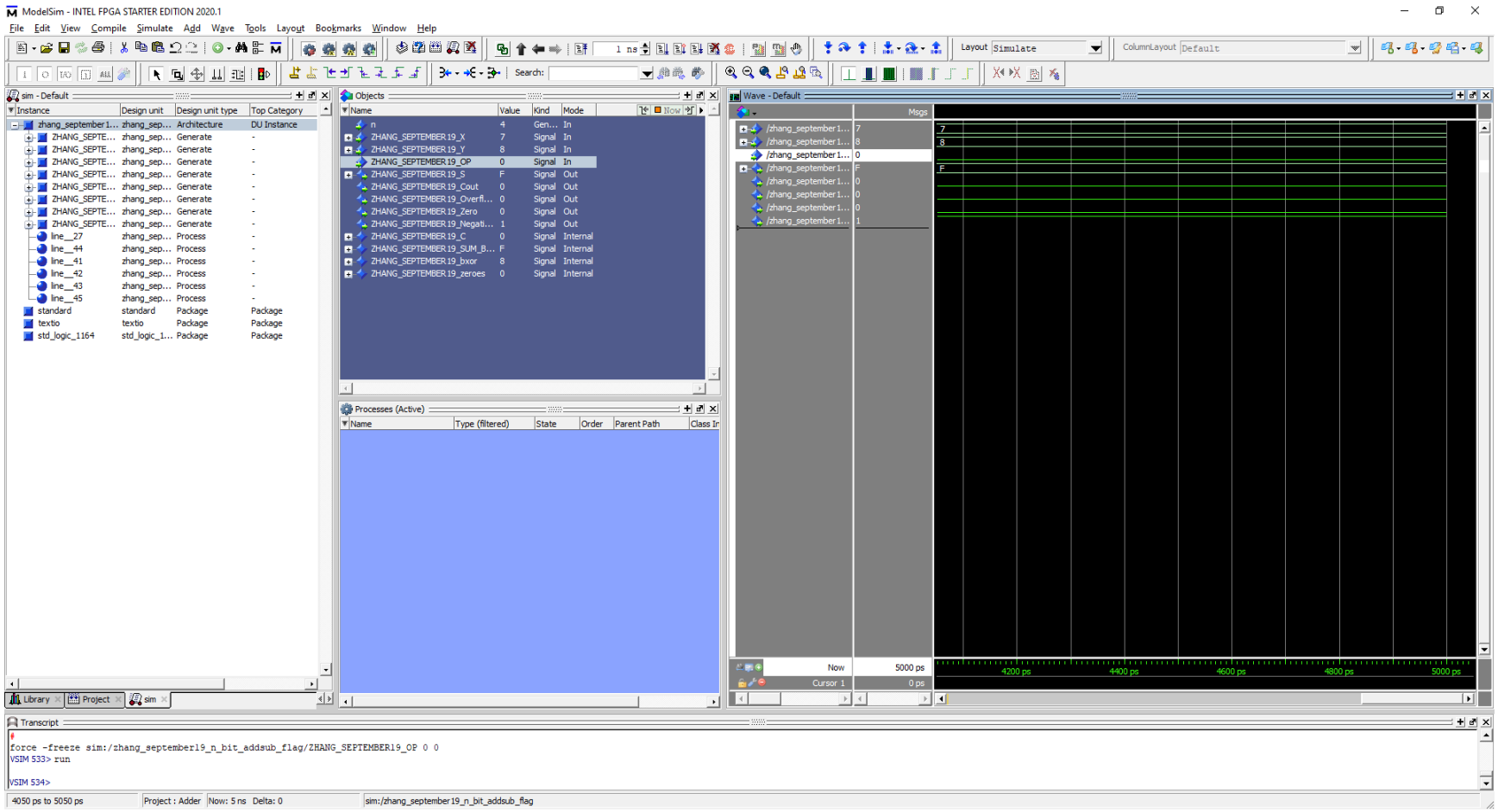
Task 8d. Most Negative N bit - 1, N = 4

8-1 = 7 which is positive but a negative just became positive so the overflow flag triggered



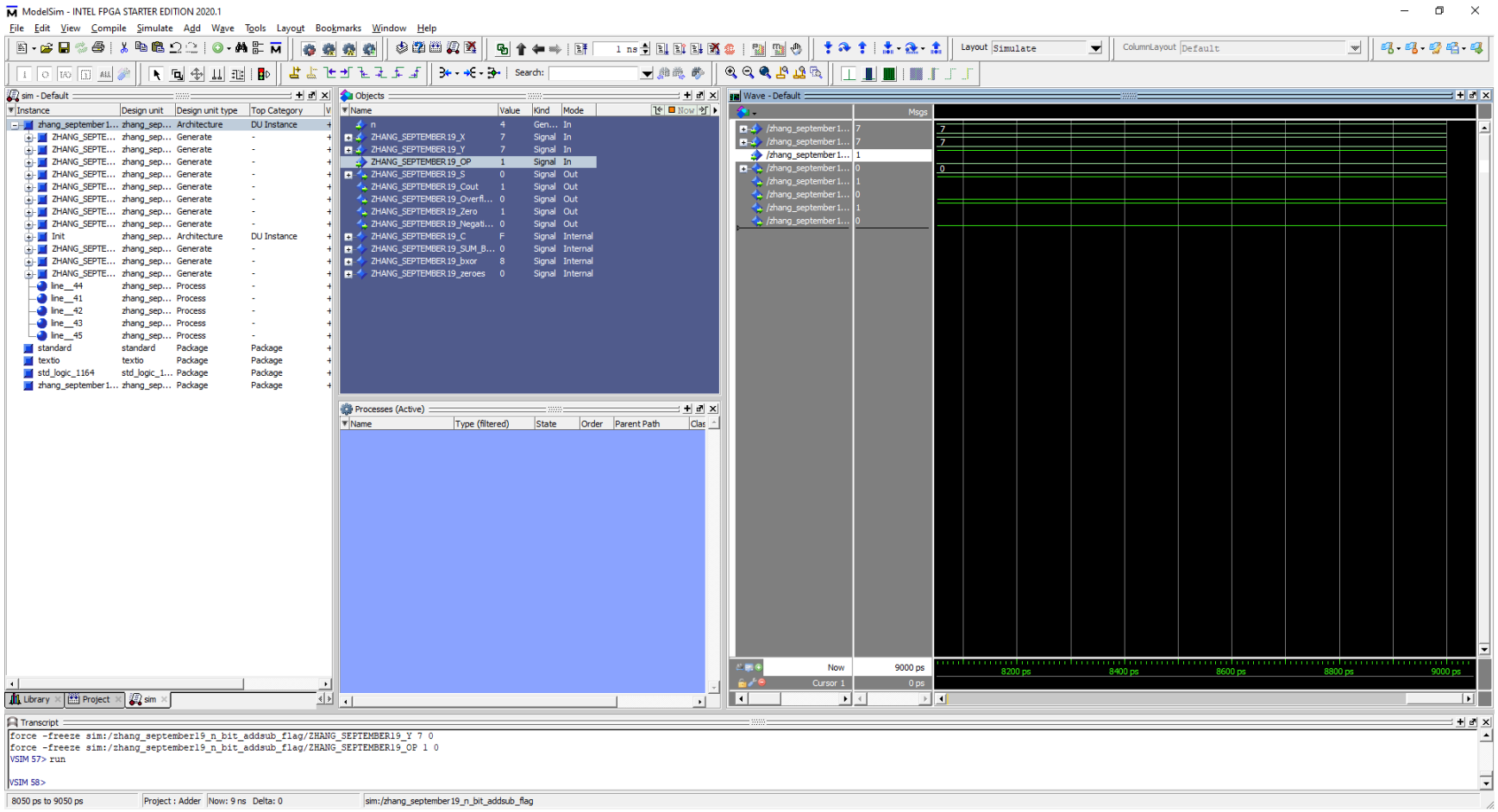
Task 8e. Most Positive N bit - Most Negative N bit, N = 4

7 – 8 = -1 which loops back to F(15). Overflow is triggered because of the state change from negative to positive to negative and negative flag is also triggered because it is negative.



Task 8f. Most Positive N bit + Most Negative N bit, N = 4

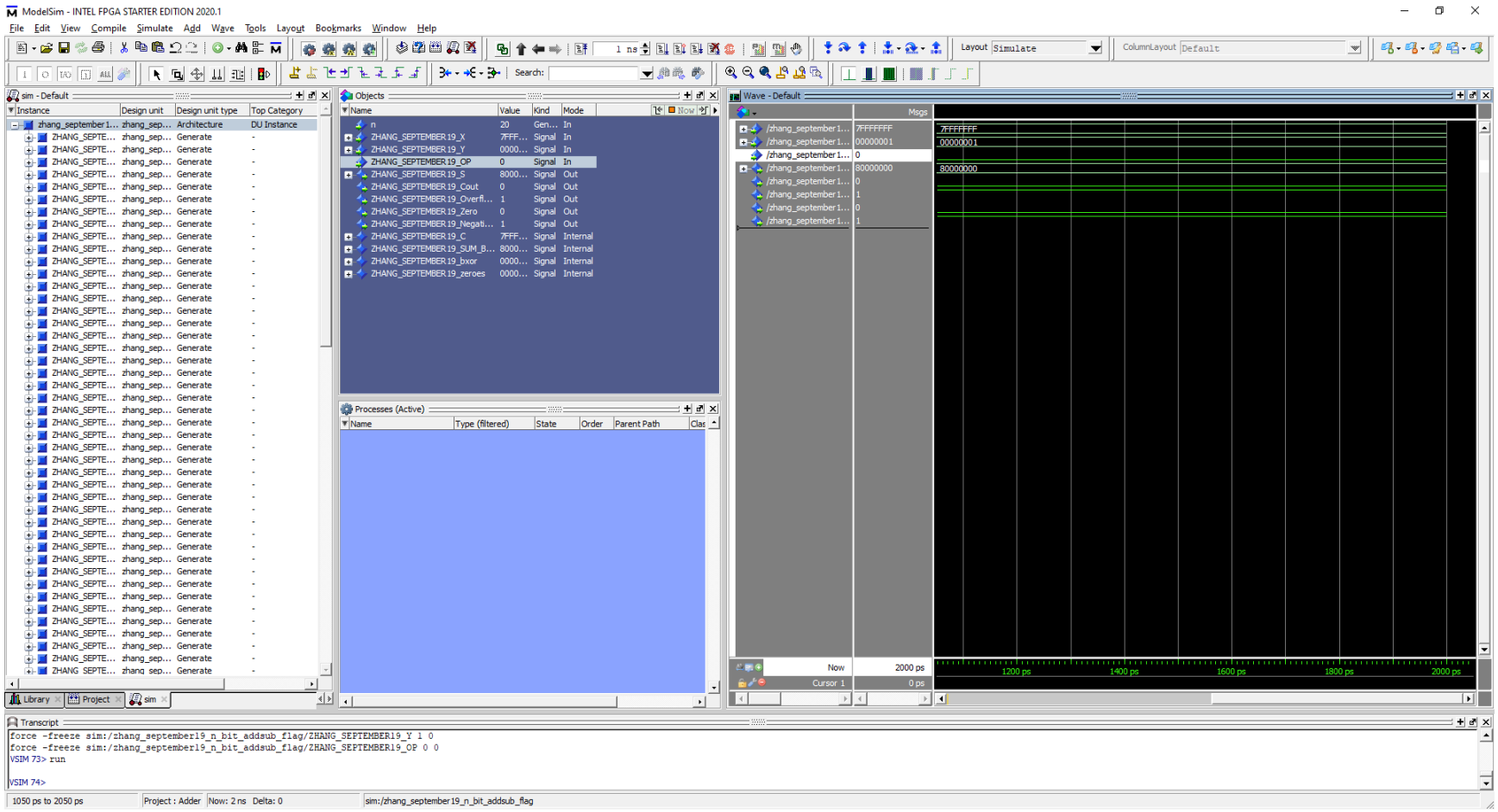
7 + 8 = 15 which goes from negative to negative, therefore no overflow flag is triggered but negative flag is still triggered.



Task 8g. Most Positive N bit - Most Negative N bit, N = 4

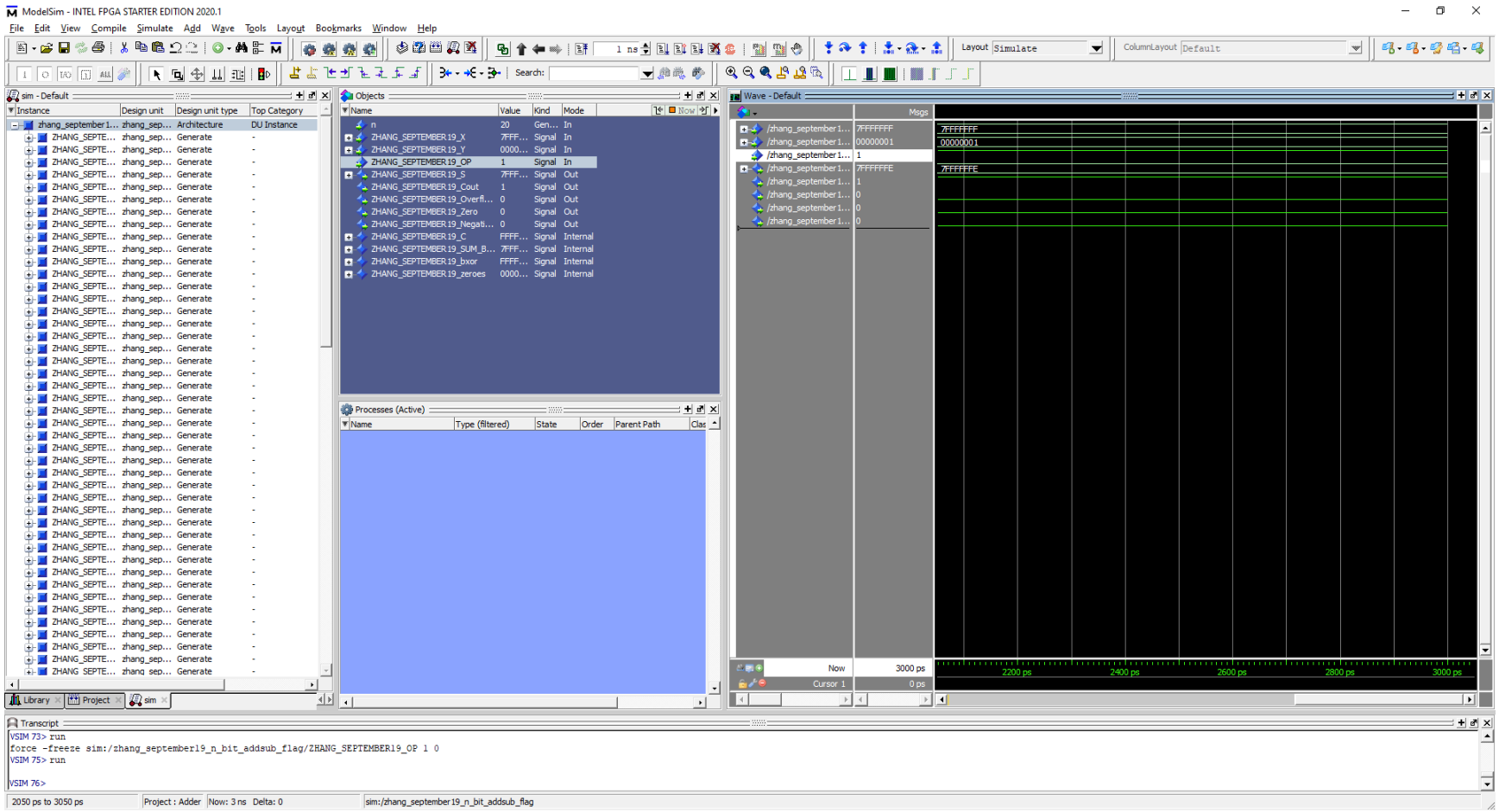
7-7 = 0 which triggers only the zero flag and nothing else.

Chue Zhang , Task 8 : Overflow, Zero and Negative Flags verify n=4 and n=32



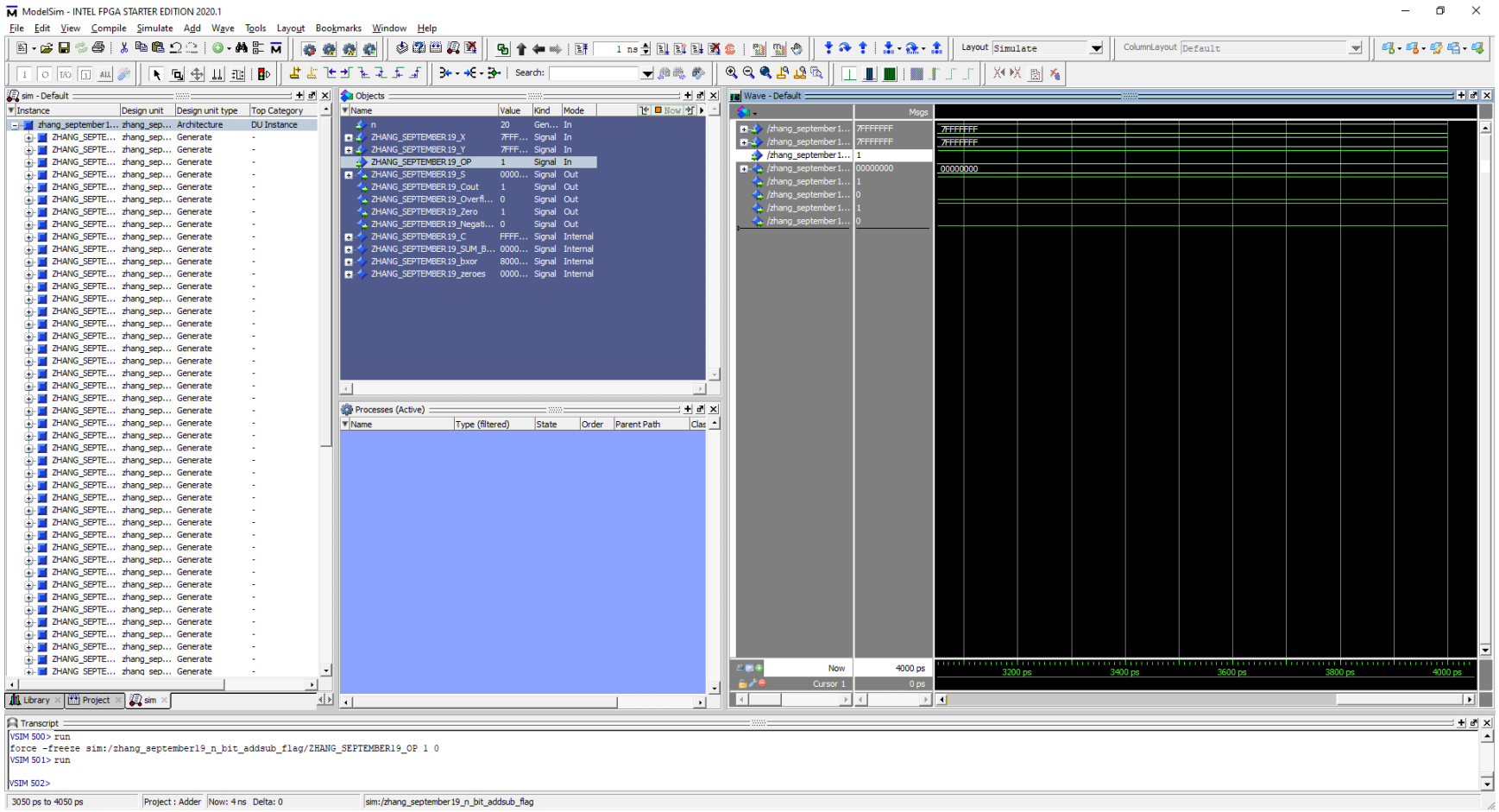
Task 8a. Most Positive N bit + 1, N = 32

The most positive N bit integer add 1 means that the most significant bit now will have a 1 which triggers negative flag and overflow flag



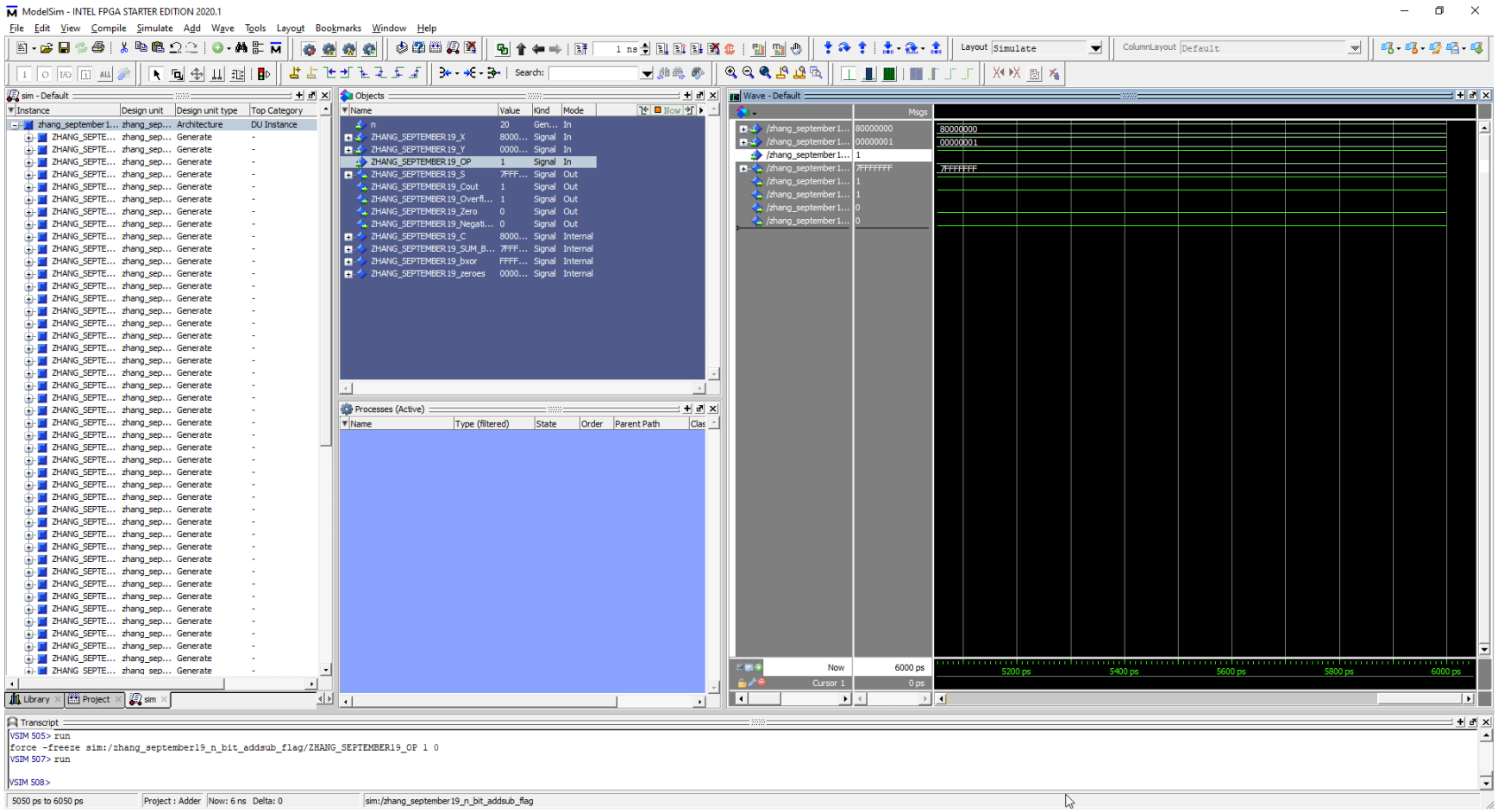
Task 8b. Most Positive N bit - 1, N = 32

No flags triggered as two positives subtracting each other is positive, there are no zeroes and the most positive – 1 is still positive



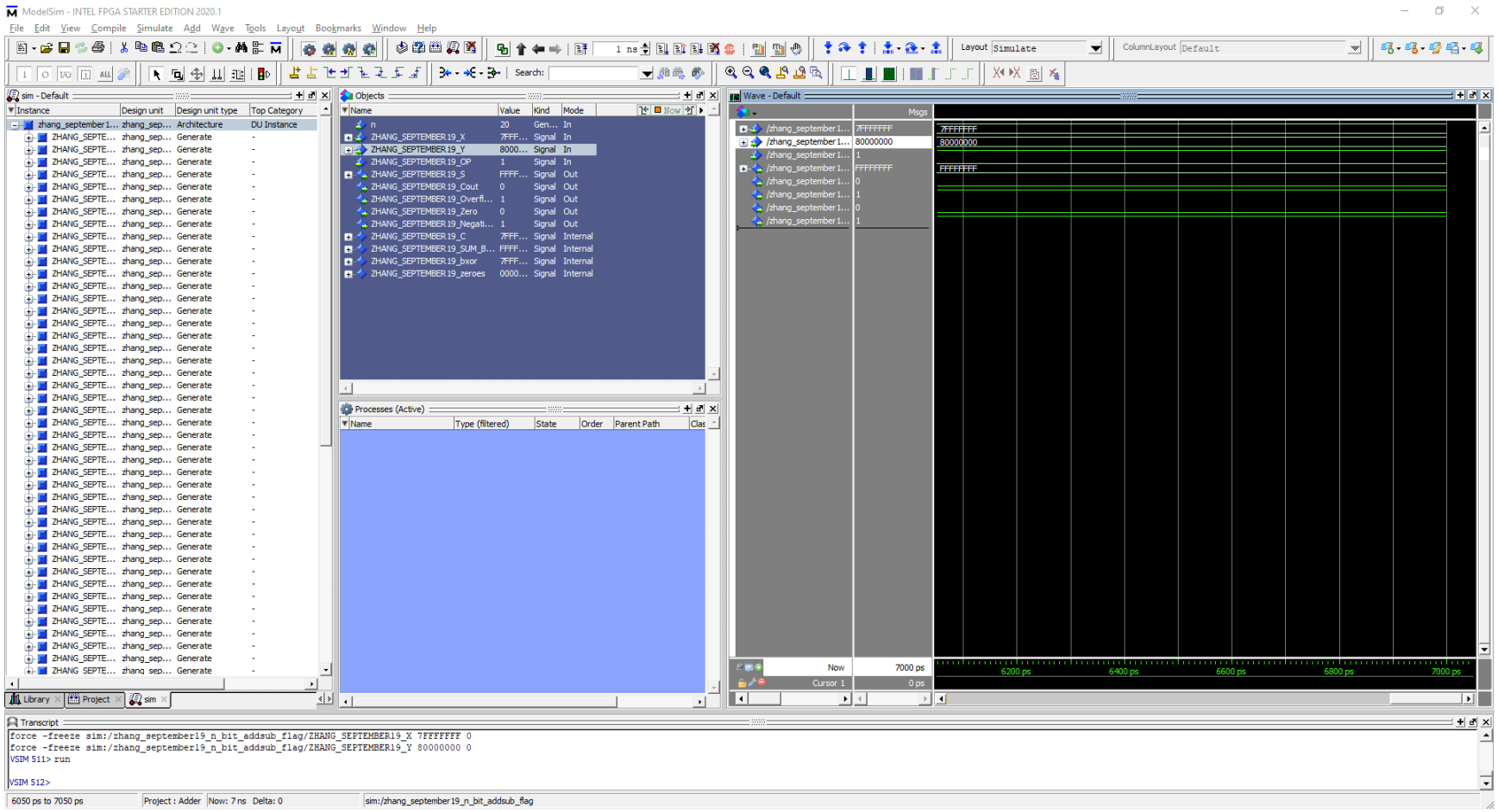
Task 8c. Most Negative N bit + 1, N = 32

The most negative N bit + 1 is 8000 0001 which is negative, which also triggers the negative flag



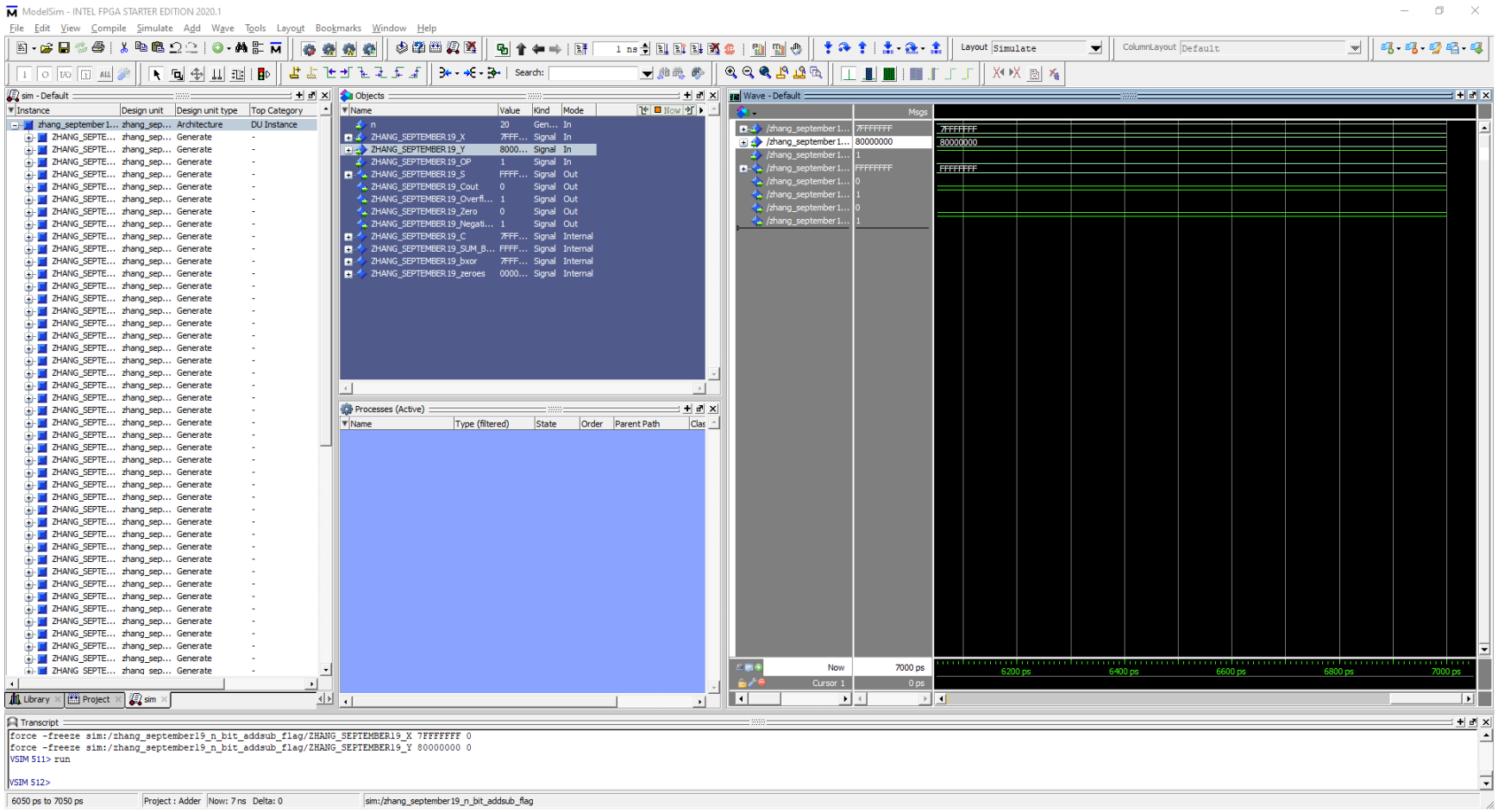
Task 8d. Most Negative N bit - 1, N = 32

Most negative N bit – 1 becomes a positive N bit therefore overflow flag is triggered



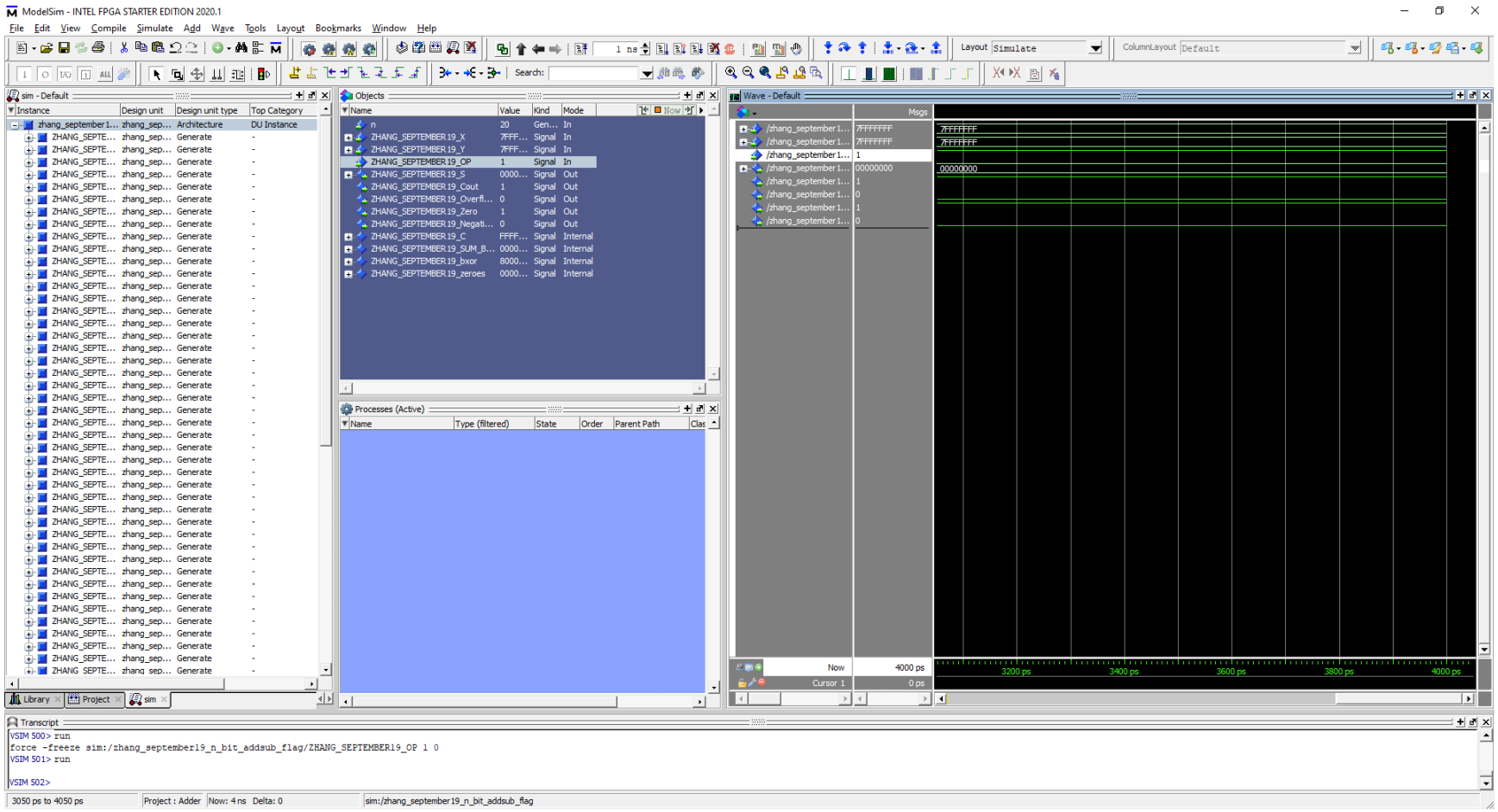
Task 8e. Most Positive N bit - Most Negative N bit, N = 32

The Most positive N bit – Most negative N bit results in FFFFFFFF which is the least negative N bit and also triggers the overflow flag and the negative flag. Overflow is triggered because double negative negates and a positive and a positive just became a negative



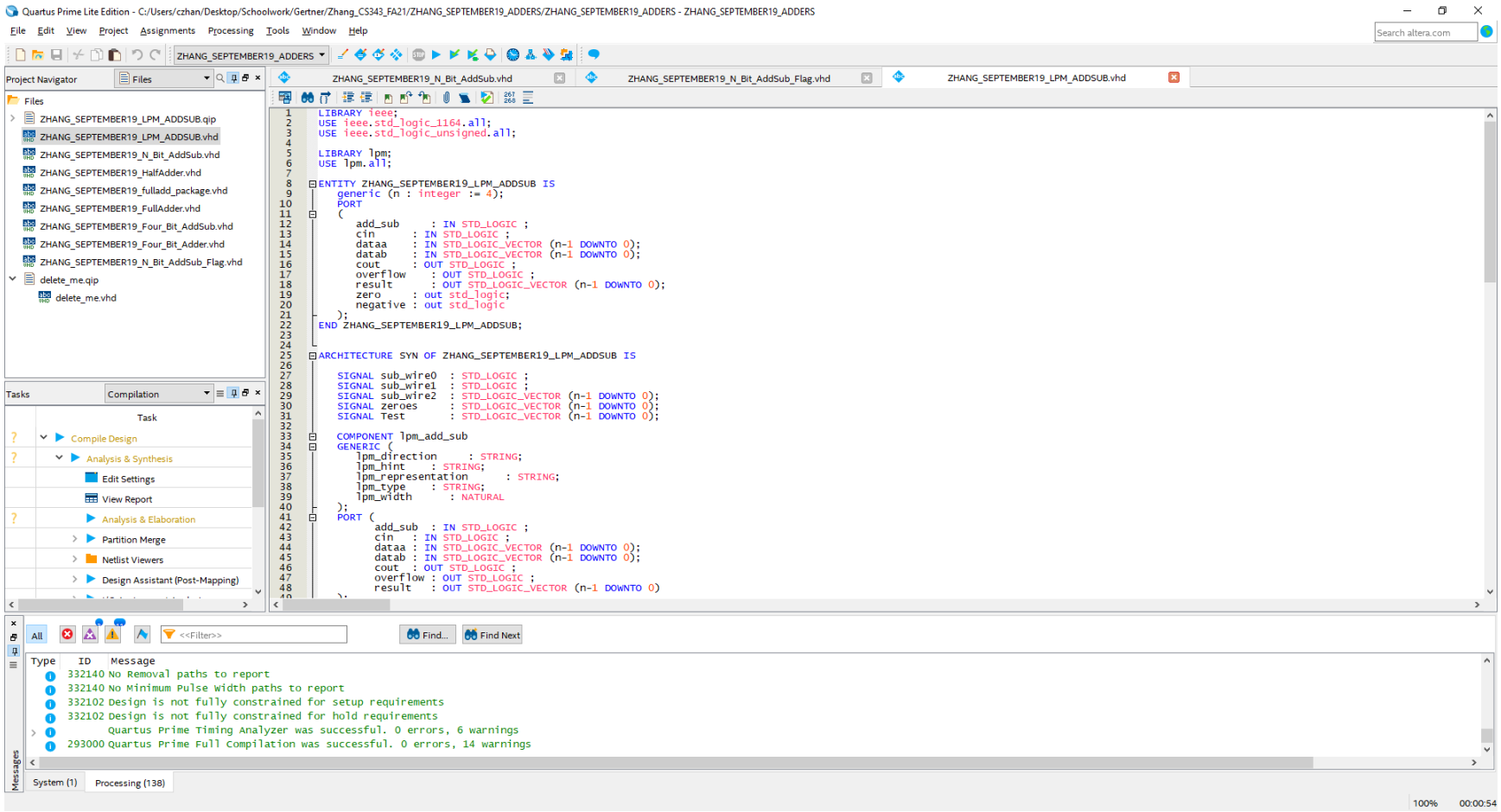
Task 8f. Most Positive N bit + Most Negative N bit, N = 32

The result becomes the same as above except no negative negation therefore no overflow flag but there is a negative flag triggered.

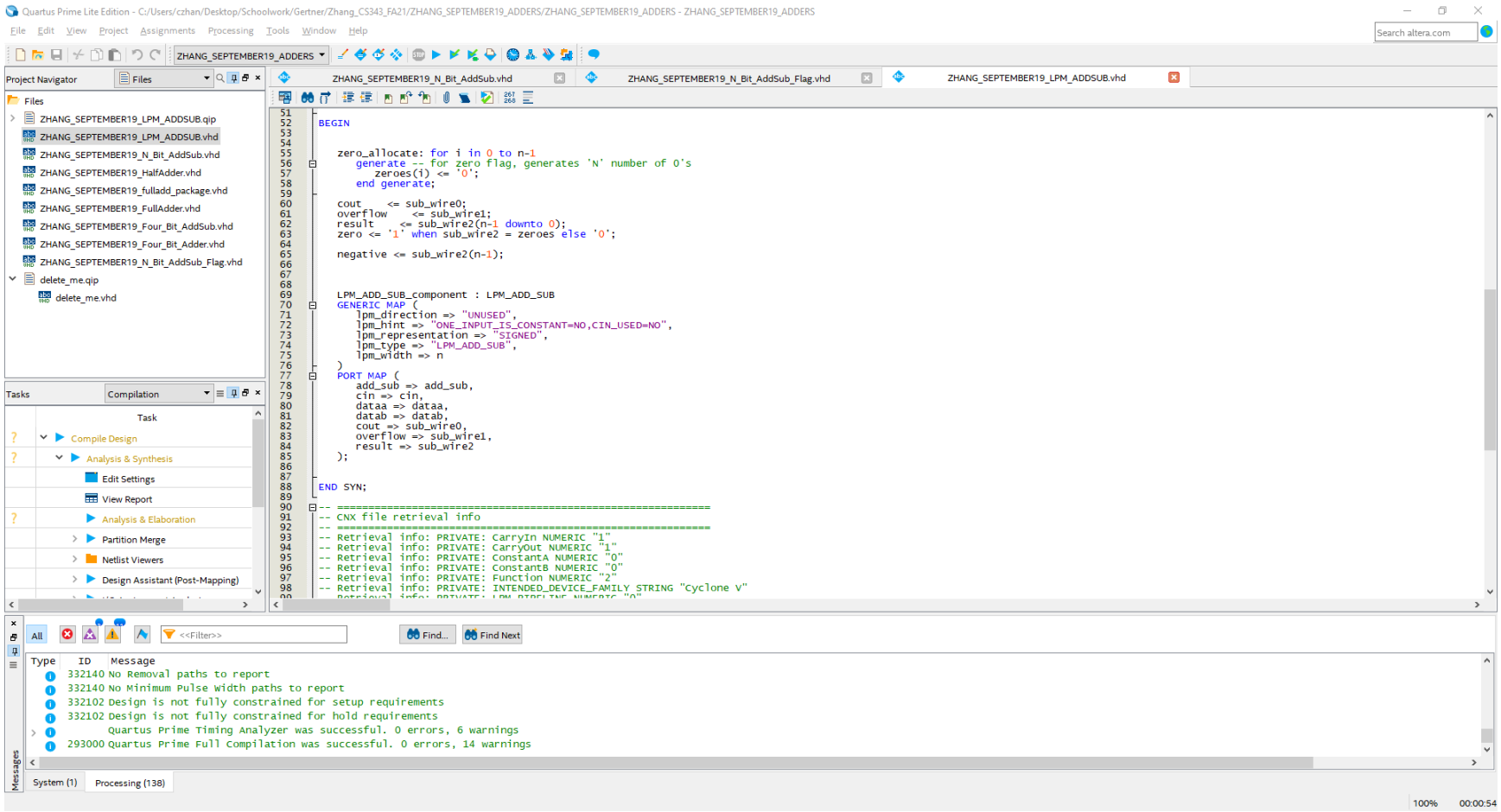


Task 8f. Most Positive N bit - Most Positive N bit, N = 32

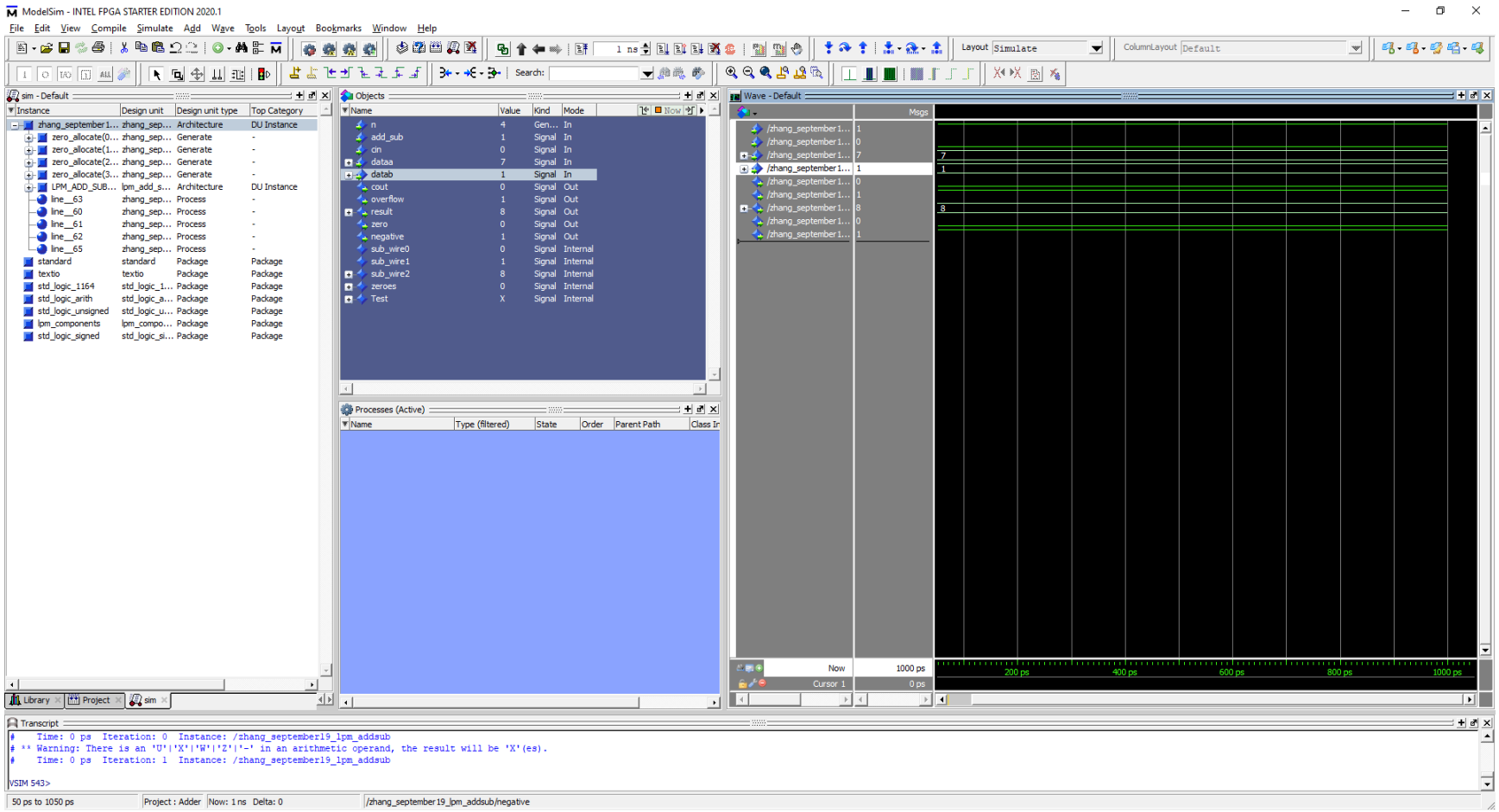
Only Zero flag is triggered as the two most positive N bit integers subtracting each other is 0.



LPM N-B it Adder/Subtractor VHDL code with flags part 1

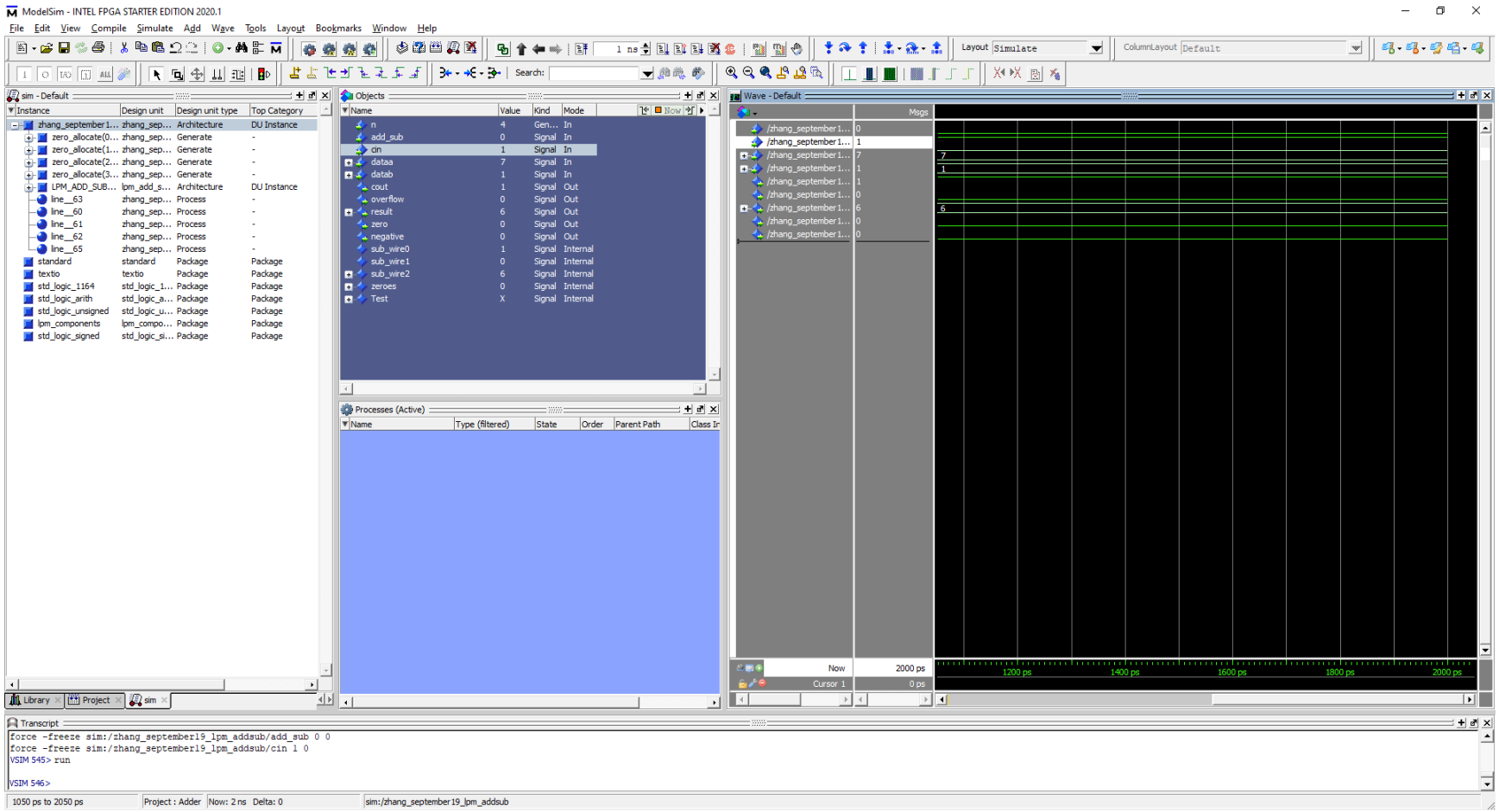


LPM N-B it Adder/Subtractor VHDL code with flags part 2



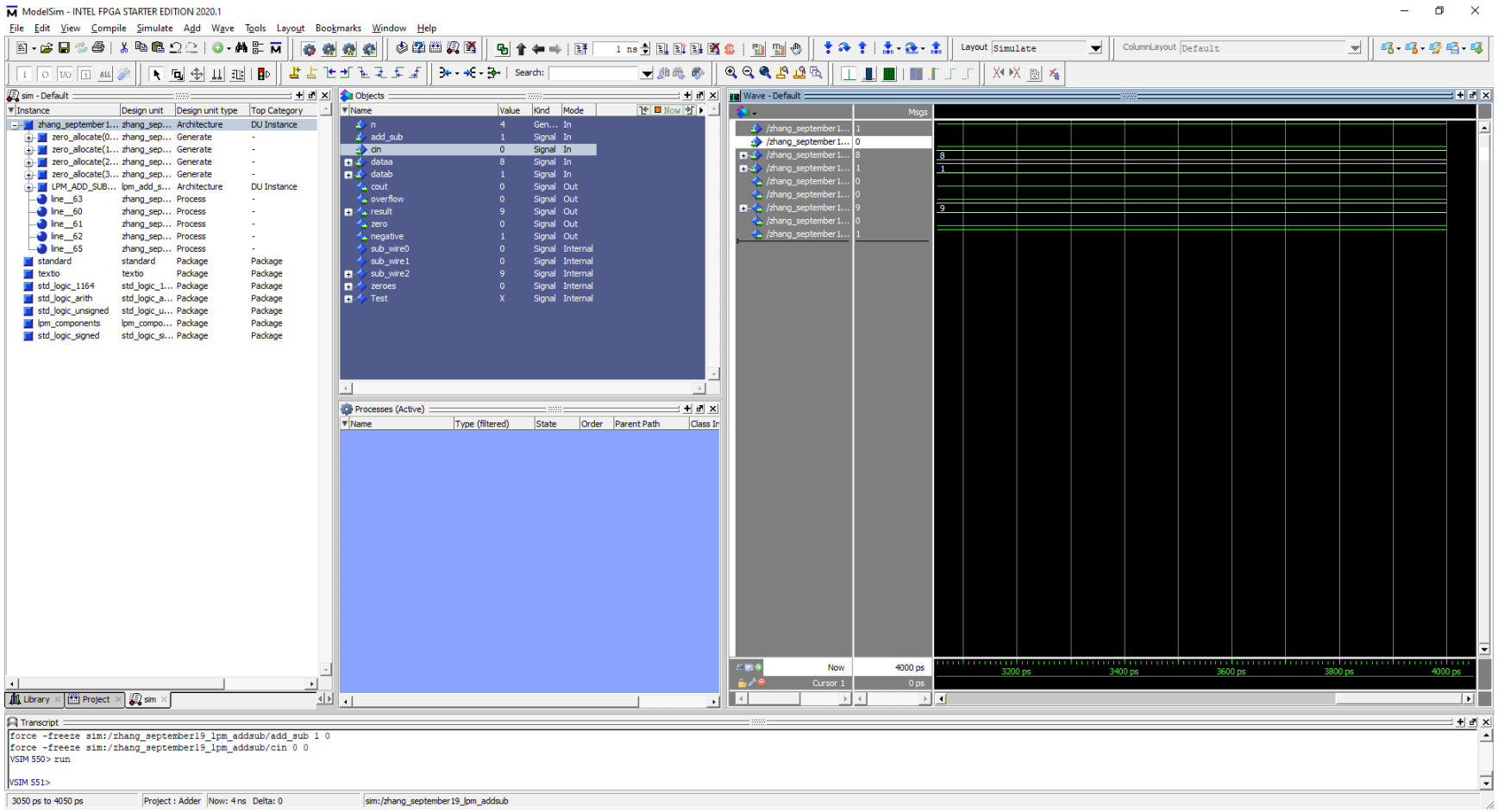
Task 8a. Most Positive N bit + 1, N = 4

Most positive N = 7 which when added with 1 triggers the overflow flag and the negative flag. Two positives became a negative so overflow flag and its negative so negative flag



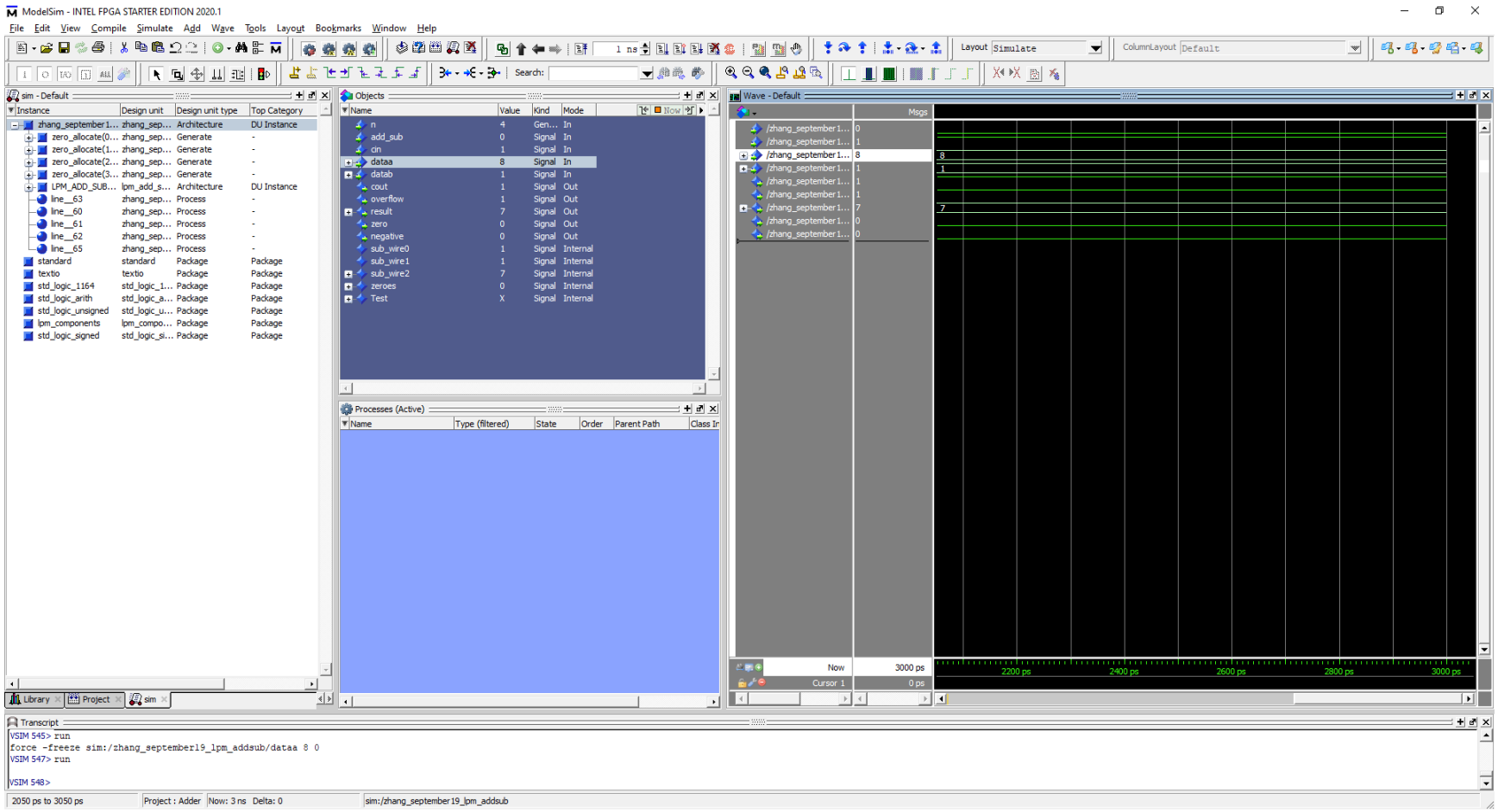
Task 8b. Most Positive N bit - 1, N = 4

7 -1 = 6 which leaves it as a positive number and no other flags trigger



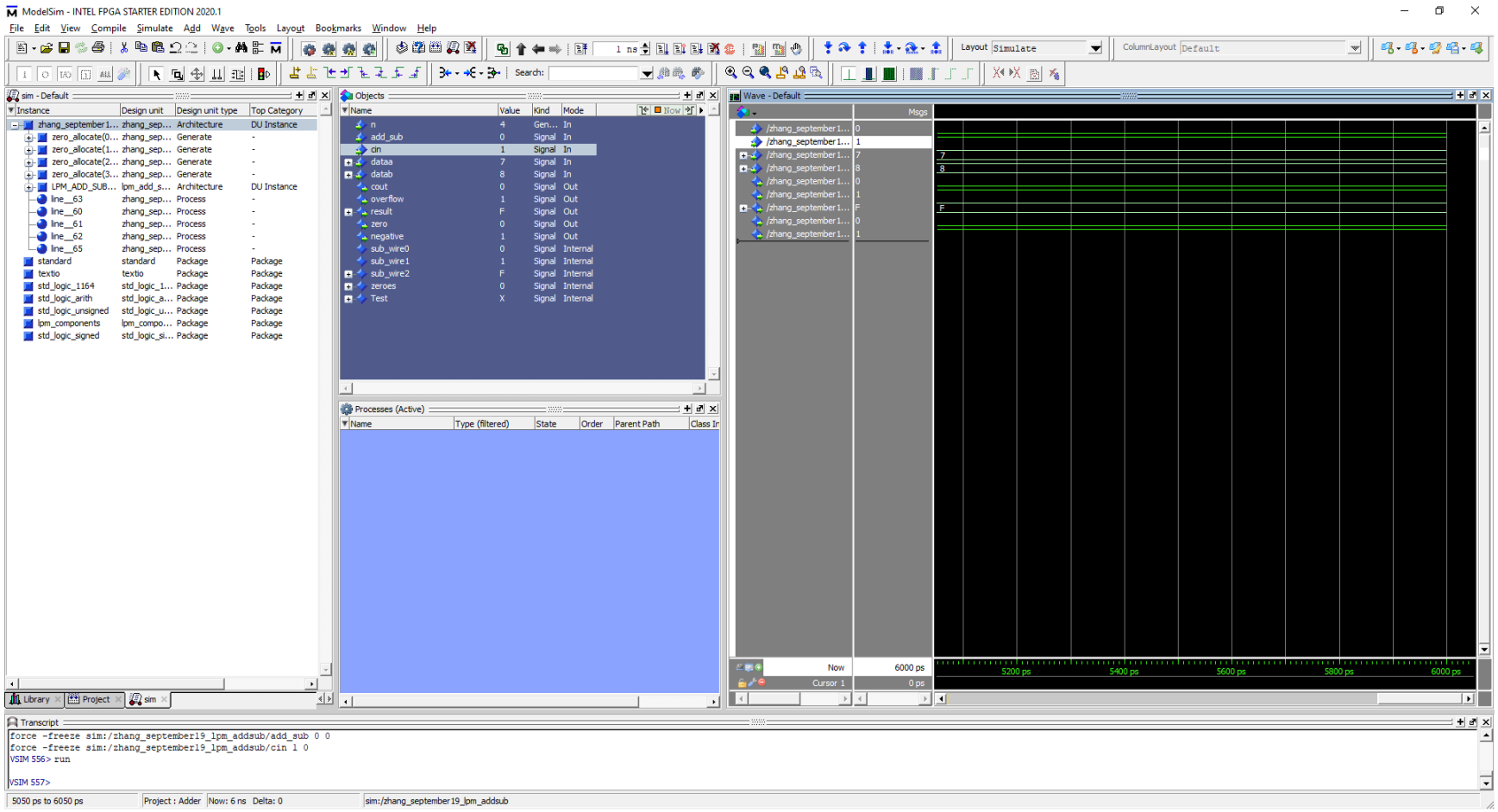
Task 8c. Most Negative N bit + 1, N = 4

8 is the most negative N bit and when added with 1, it becomes 9 which triggers the negative flag only



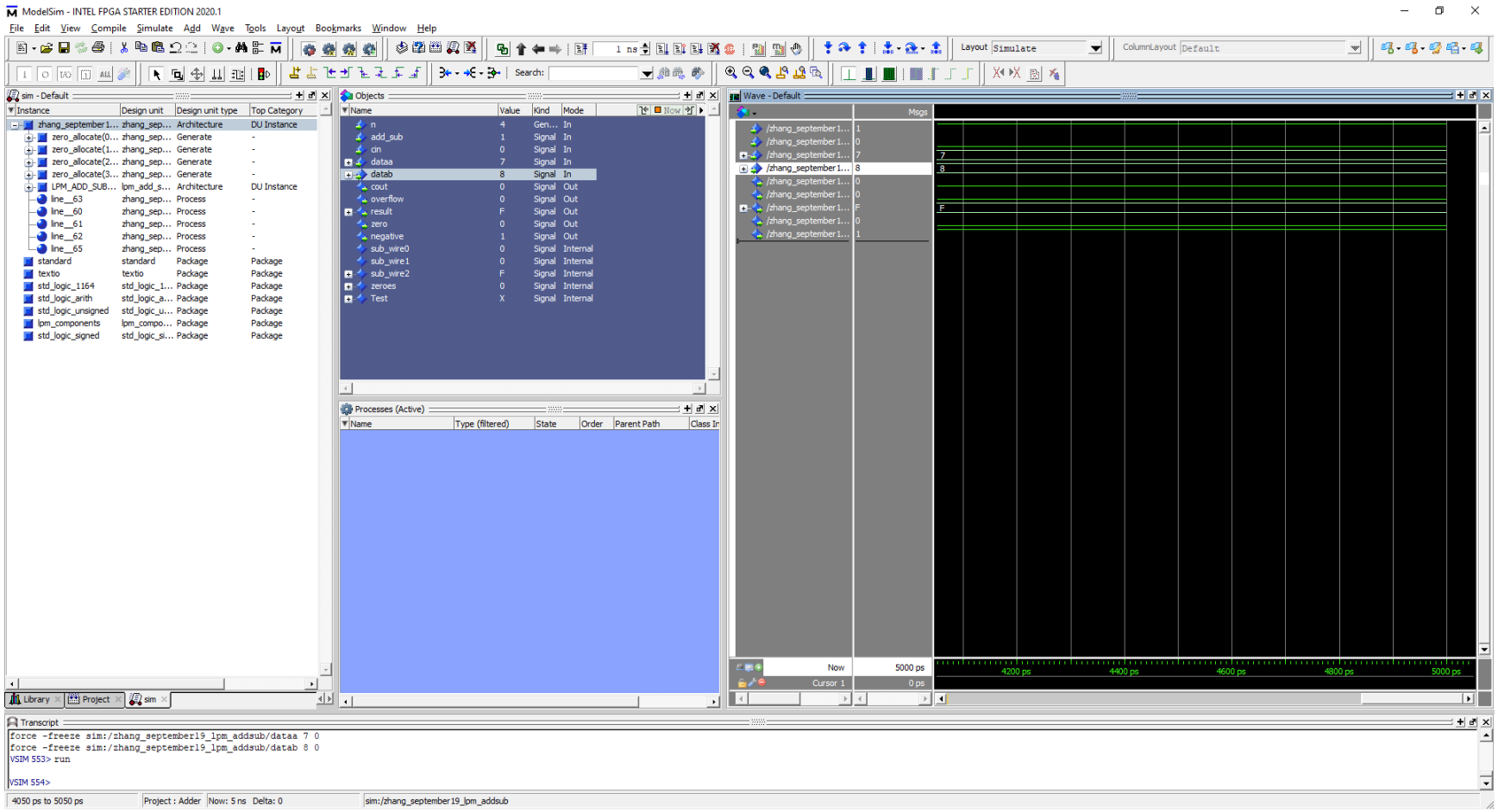
Task 8d. Most Negative N bit - 1, N = 4

8 – 1 = 7 which triggers the overflow flag because of the sign change



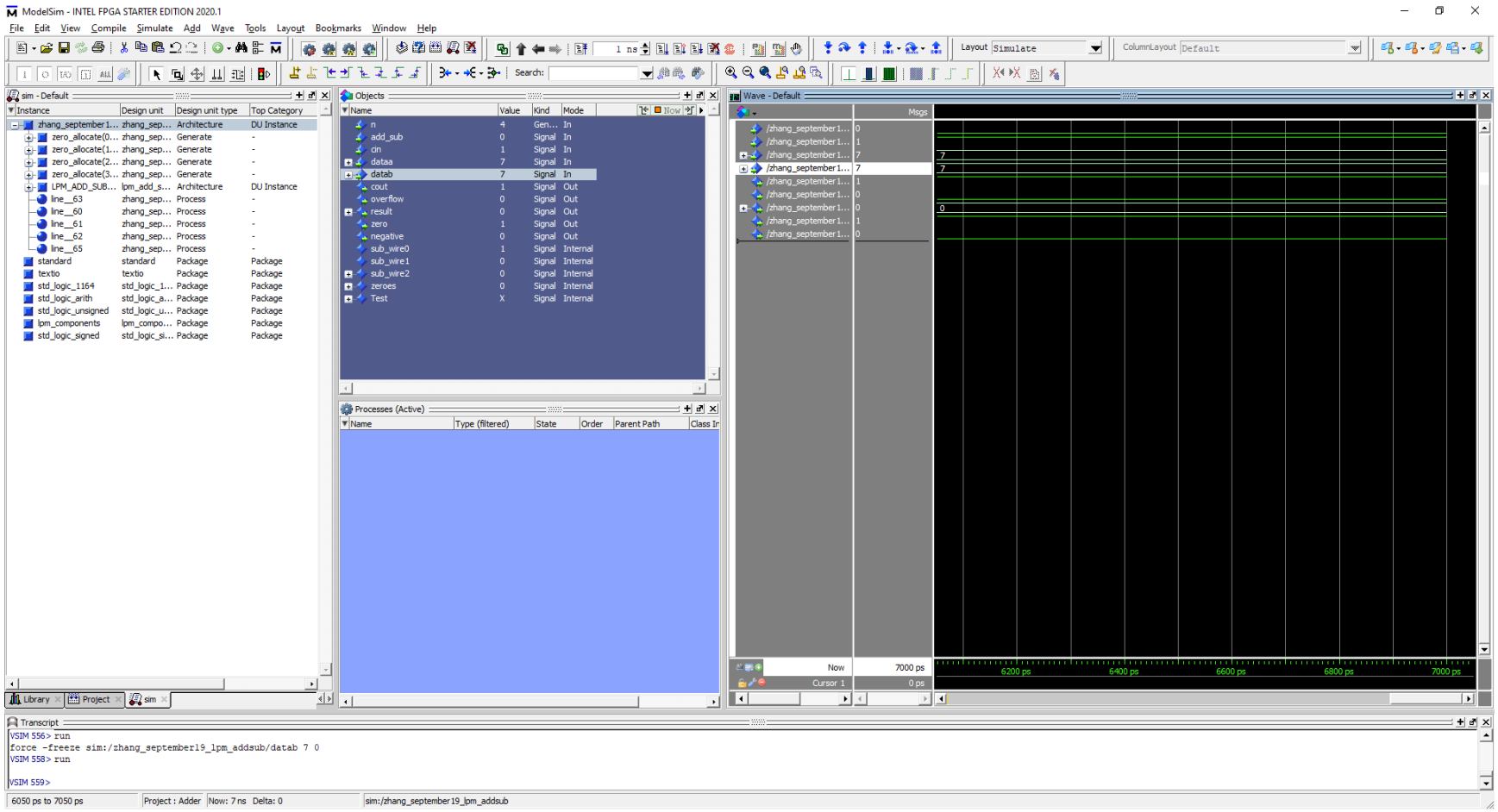
Task 8e. Most Positive N bit - Most Negative N bit , N = 4

7 – 8 = -1 which means it goes from negative to positive to negative which triggers the overflow flag as well as the negative flag



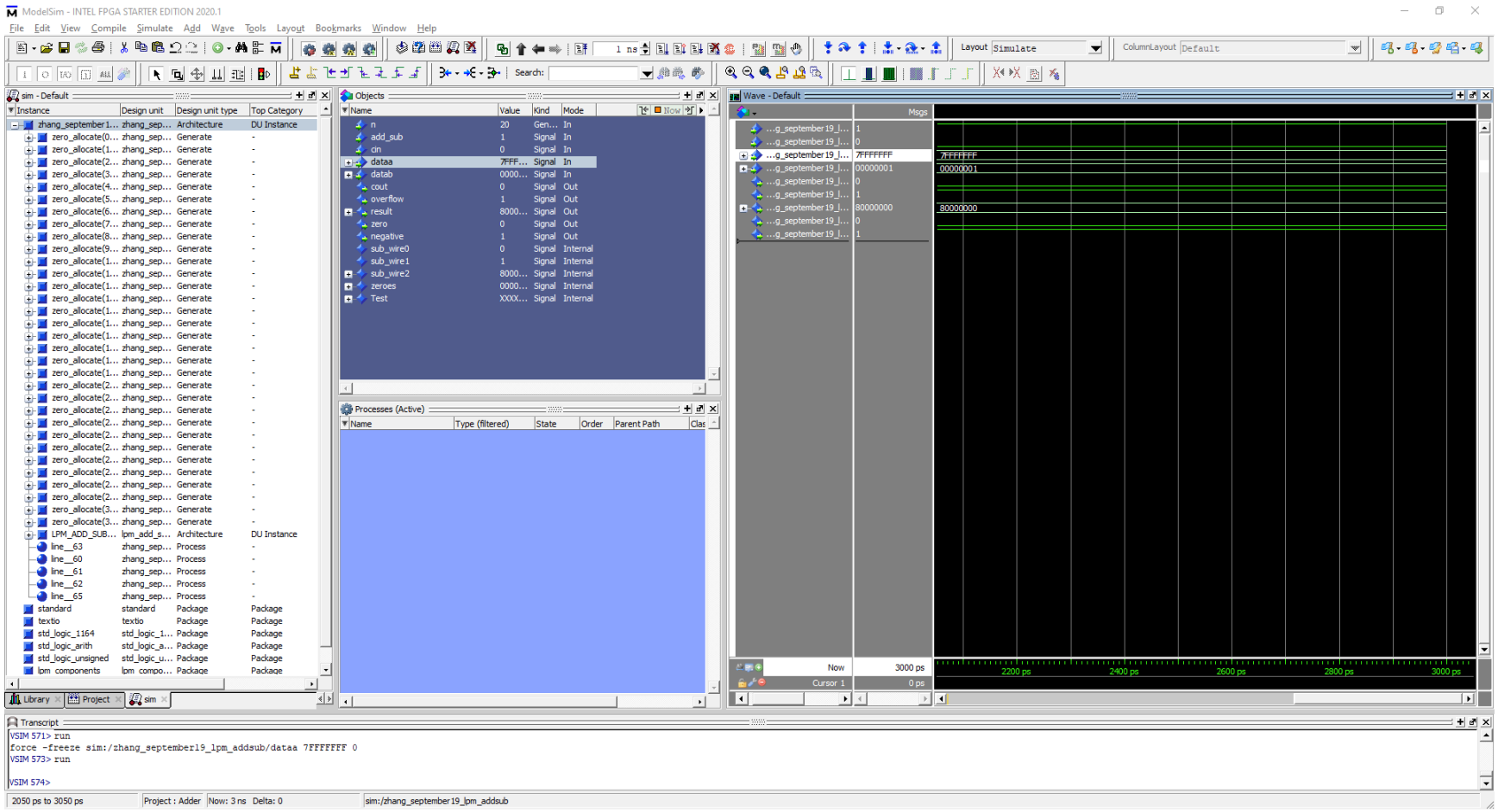
Task 8e. Most Positive N bit + Most Negative N bit , N = 4

7+8 = 15 which is negative so only the negative flag is triggered, no overflow occurs



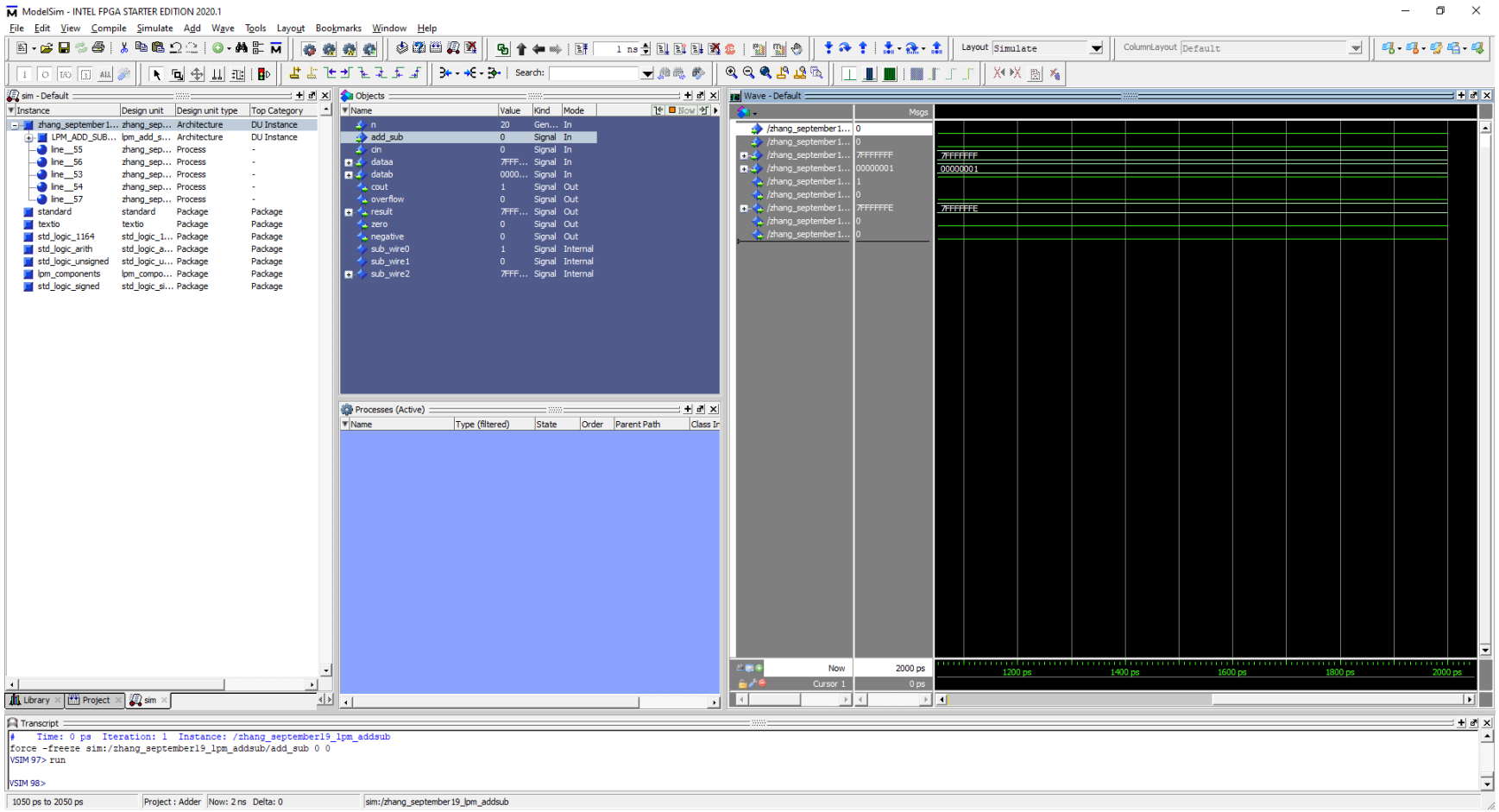
Task 8f. Most Positive N bit - Most Positive N bit , N = 4

7 – 7 = 0 which only triggers the zero flag



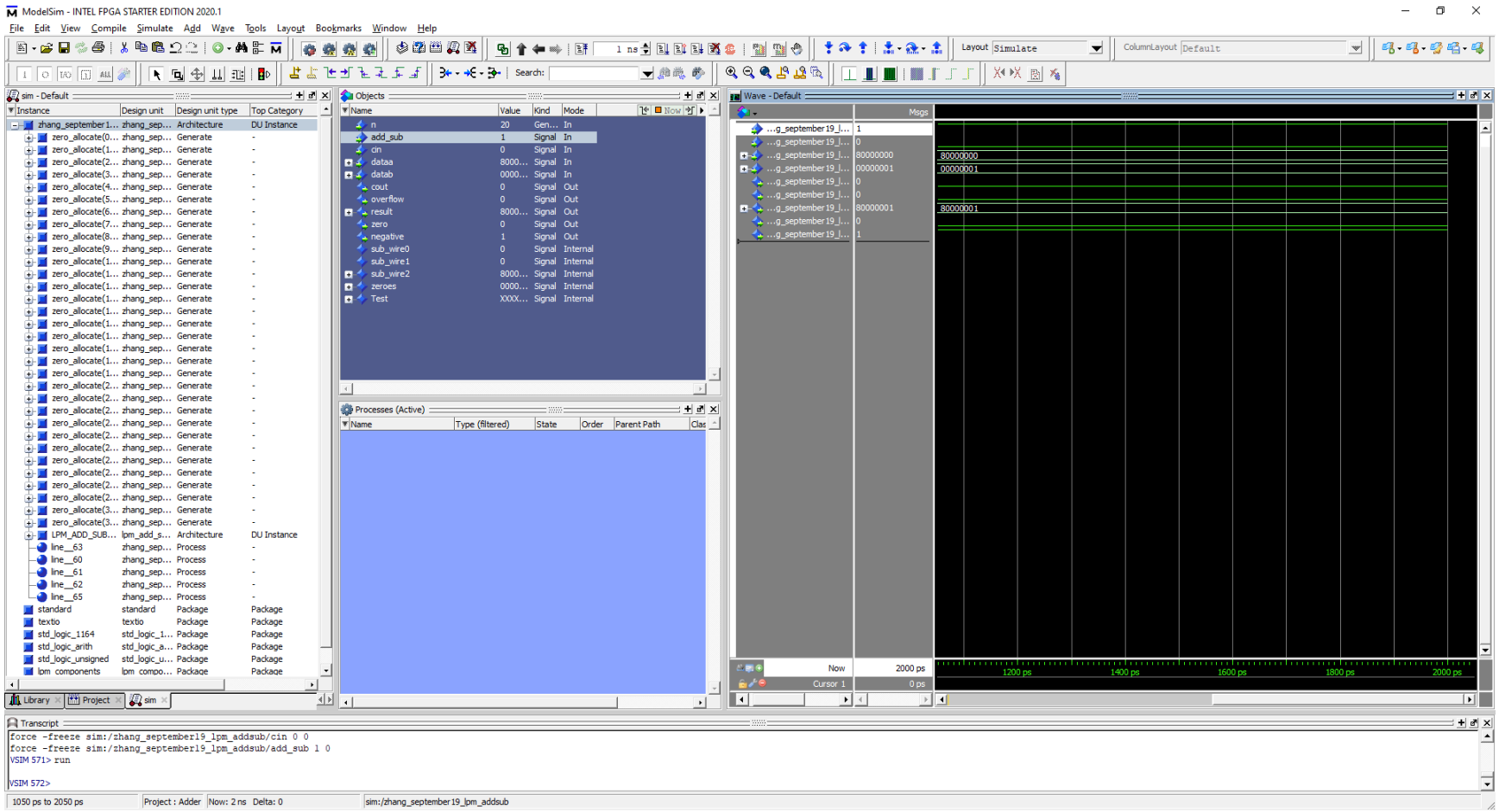
Task 8a. Most Positive N bit + 1, N = 32

Most positive N bit integer + 1 becomes negative afterwards so negative flag triggered alongside overflow



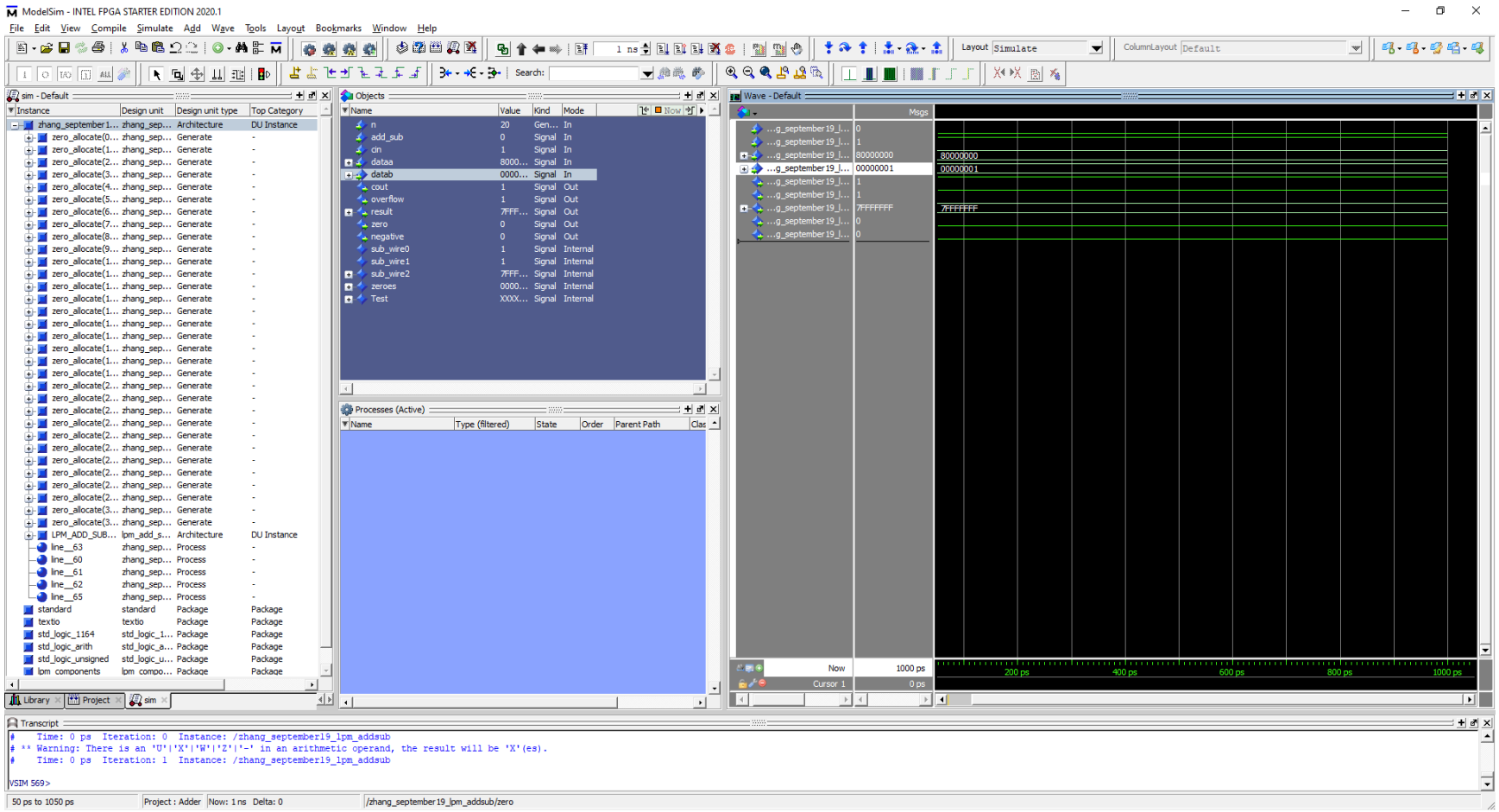
Task 8b. Most Positive N bit - 1, N = 32

Most positive N bit – 1 is still a positive integer value therefore no flags are triggered



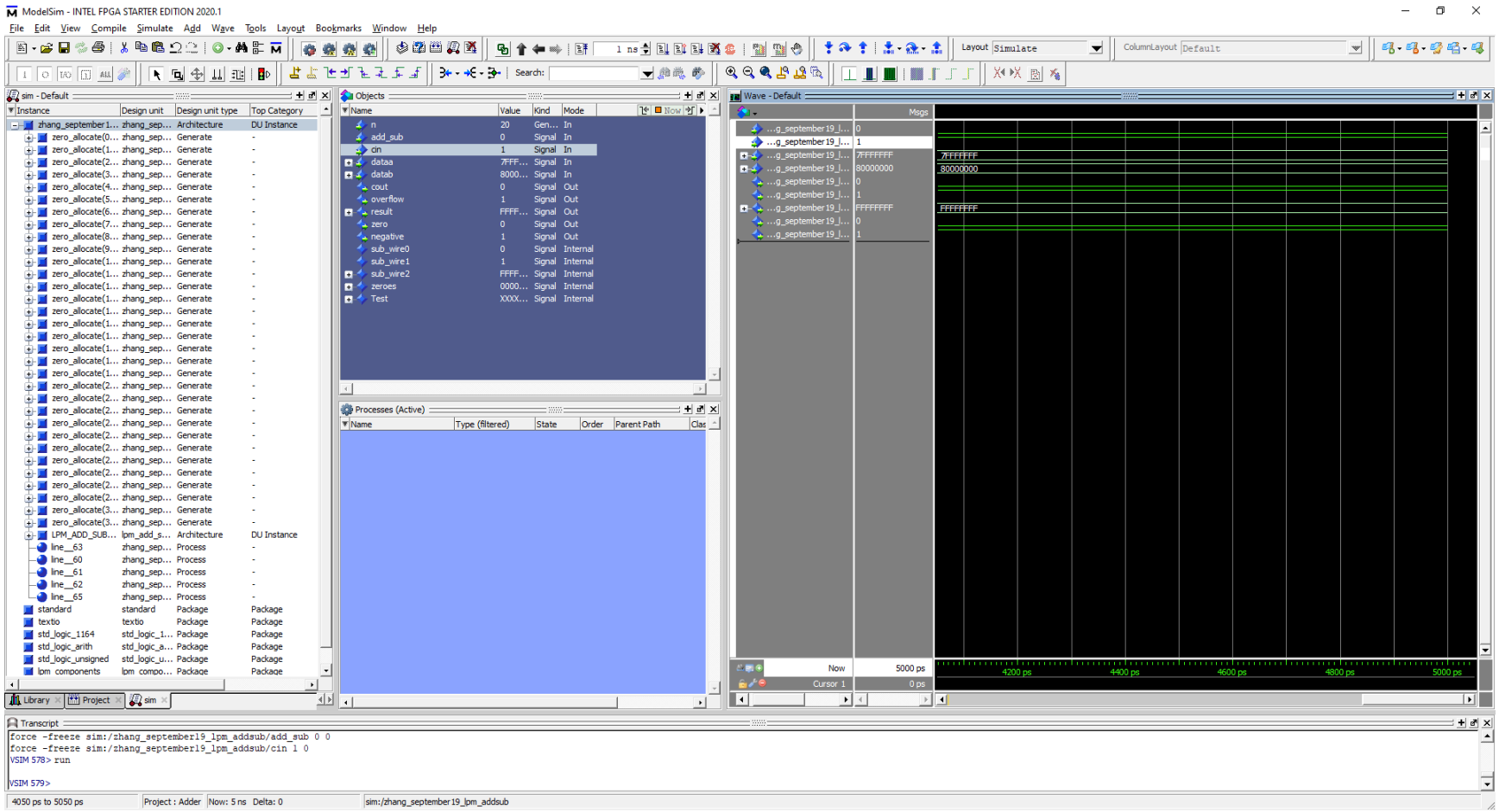
Task 8c. Most Negative N bit + 1, N = 32

Most Negative bit + 1 because ‘80000001’ which is still negative therefore negative flag is triggered



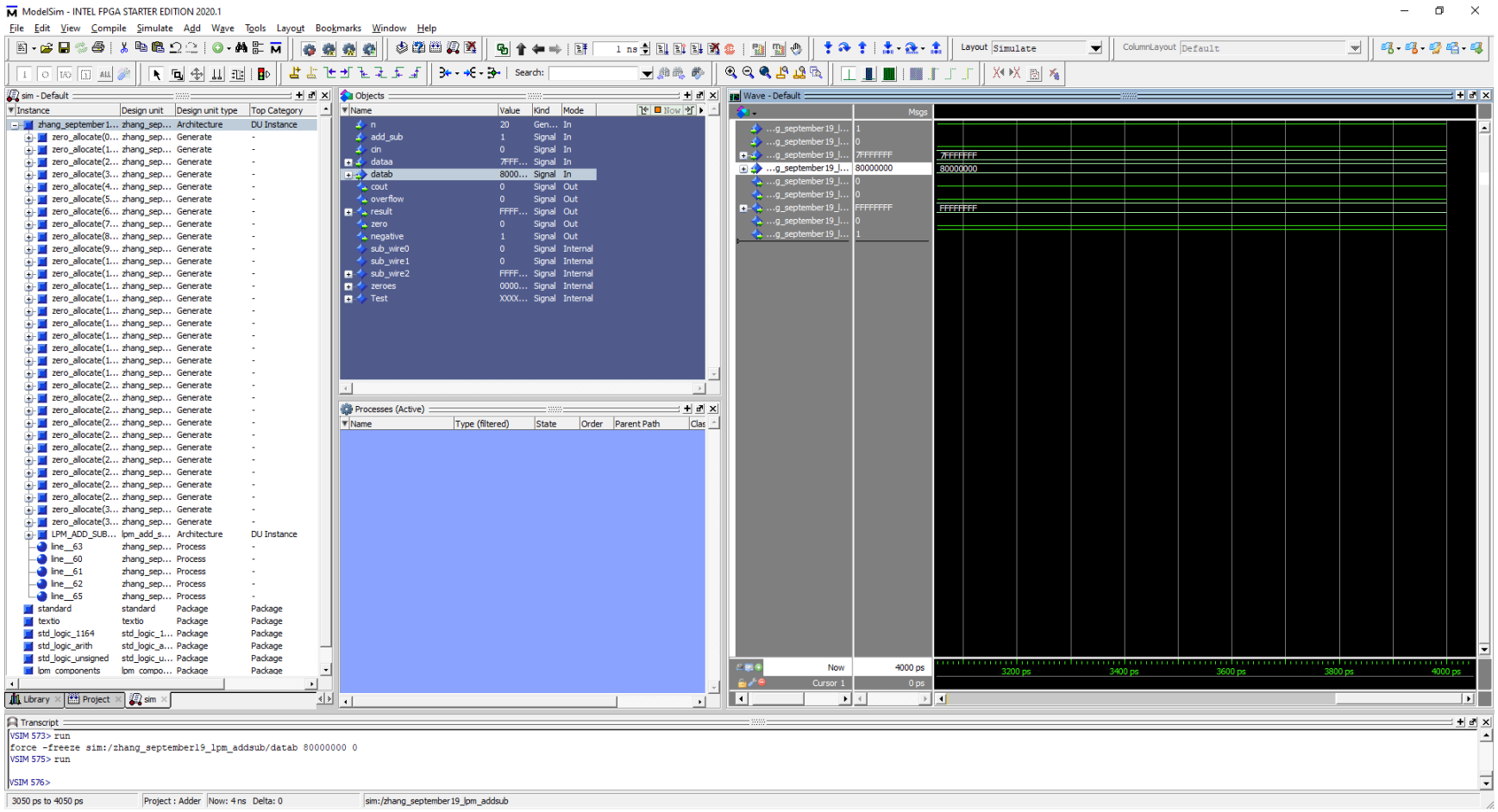
Task 8d. Most Negative N bit - 1, N = 32

Most Negative N bit – 1 is becomes positive therefore overflow flag is triggered only.



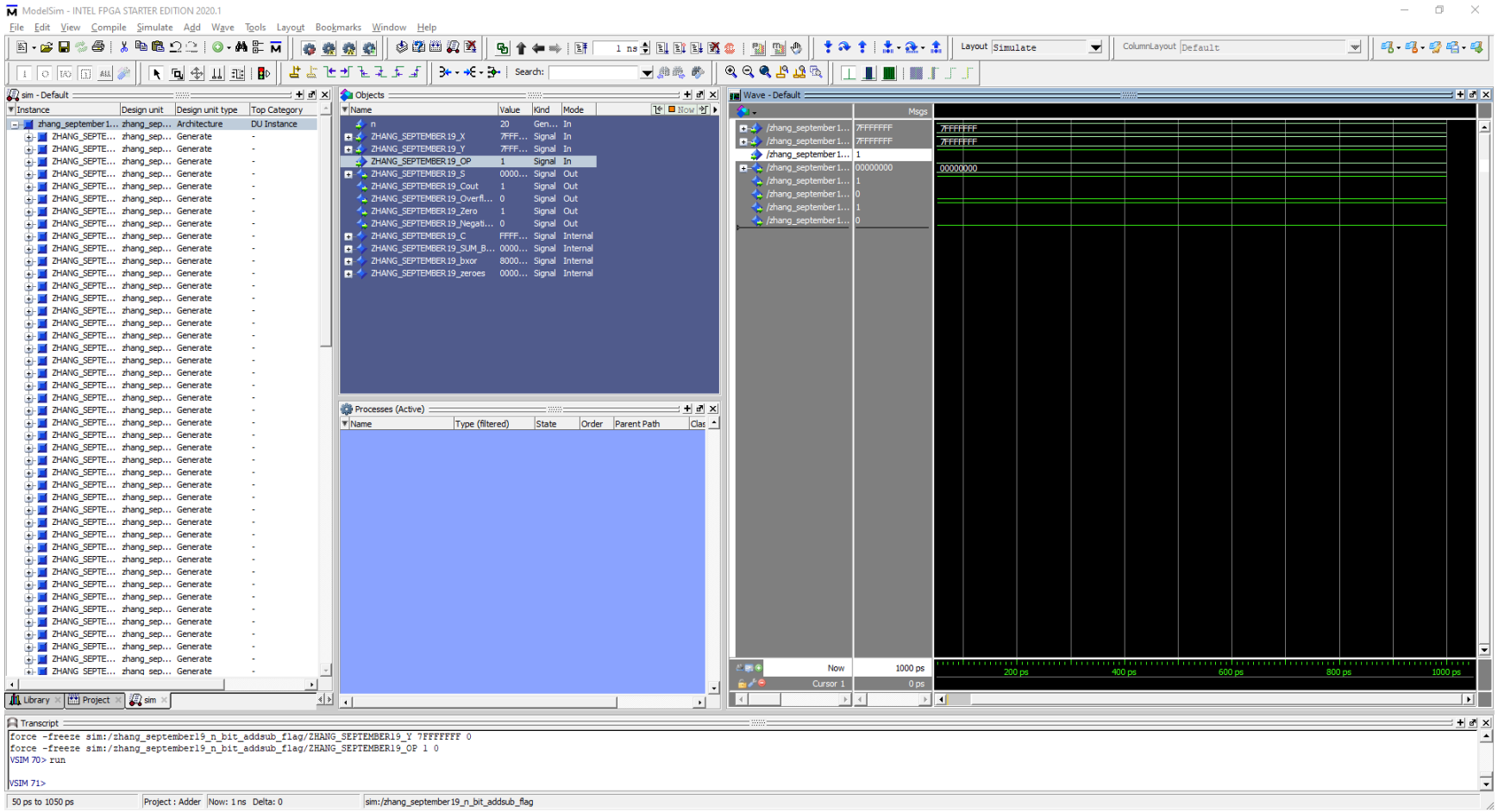
Task 8e. Most Positive N bit - Most Negative N bit , N = 32

The output goes from negative to positive to negative therefore overflow flag is triggered alongside the negative flag. FFFFFFF is the smallest negative bit and is also the limit.



Task 8f. Most Positive N bit + Most Negative N bit , N = 32

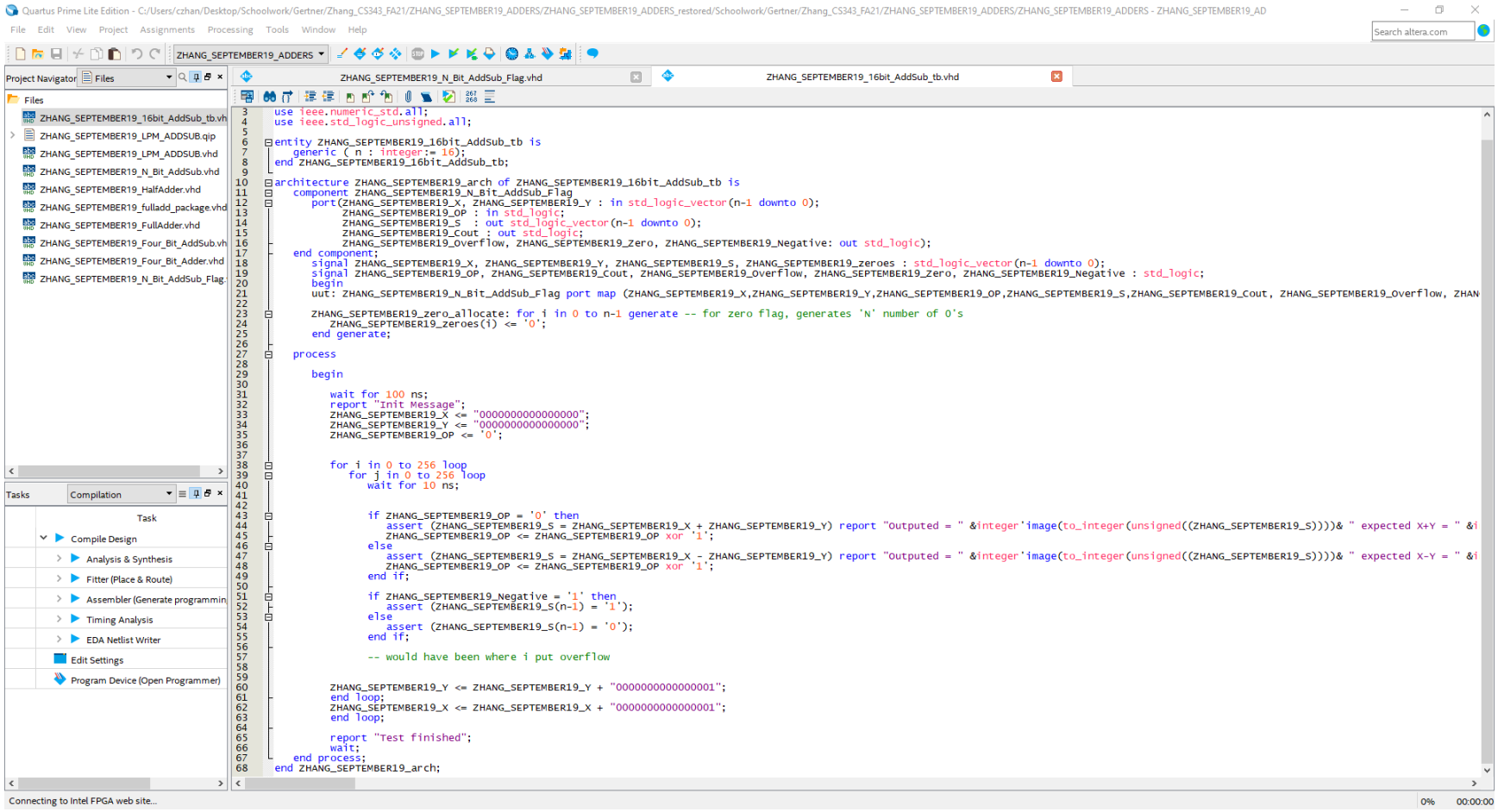
Nothing is triggered except for negative flag and there is no overflow because it goes from negative to negative



Task 8g. Most Positive N bit - Most Positive N bit , N = 32

Two identical values subtracting each other results in zero thus triggering the zero flag

Chue Zhang , Task 10 : Testbench NO FLAGS



```
 3  use ieee.numeric_std.all;
 4  use ieee.std_logic_unsigned.all;
 5
 6  entity ZHANG_SEPTEMBER19_16bit_AddSub_tb is
 7      generic ( n : integer:= 16);
 8  end ZHANG_SEPTEMBER19_16bit_AddSub_tb;
 9
10  architecture ZHANG_SEPTEMBER19_arch of ZHANG_SEPTEMBER19_16bit_AddSub_tb is
11      component ZHANG_SEPTEMBER19_N_Bit_AddSub_Flag
12          port(ZHANG_SEPTEMBER19_X, ZHANG_SEPTEMBER19_Y : in std_logic_vector(n-1 downto 0);
13               ZHANG_SEPTEMBER19_OP : in std_logic;
14               ZHANG_SEPTEMBER19_S  : out std_logic_vector(n-1 downto 0);
15               ZHANG_SEPTEMBER19_Cout : out std_logic;
16               ZHANG_SEPTEMBER19_Overflow, ZHANG_SEPTEMBER19_Zero, ZHANG_SEPTEMBER19_Negative: out std_logic);
17      end component;
18          signal ZHANG_SEPTEMBER19_X, ZHANG_SEPTEMBER19_Y, ZHANG_SEPTEMBER19_S, ZHANG_SEPTEMBER19_zeroes : std_logic_vector(n-1 downto 0);
19          signal ZHANG_SEPTEMBER19_OP, ZHANG_SEPTEMBER19_Cout, ZHANG_SEPTEMBER19_Overflow, ZHANG_SEPTEMBER19_Zero, ZHANG_SEPTEMBER19_Negative : std_logic;
20          begin
21          uut: ZHANG_SEPTEMBER19_N_Bit_AddSub_Flag port map (ZHANG_SEPTEMBER19_X,ZHANG_SEPTEMBER19_Y,ZHANG_SEPTEMBER19_OP,ZHANG_SEPTEMBER19_S,ZHANG_SEPTEMBER19_Cout, ZHANG_SEPTEMBER19_Overflow, ZHAN
22
23          ZHANG_SEPTEMBER19_zero_allocate: for i in 0 to n-1 generate -- for zero flag, generates 'N' number of 0's
24              ZHANG_SEPTEMBER19_zeroes(i) <= '0';
25          end generate;
26
27      process
28
29          begin
30
31              wait for 100 ns;
32              report "Init Message";
33              ZHANG_SEPTEMBER19_X <= "0000000000000000";
34              ZHANG_SEPTEMBER19_Y <= "0000000000000000";
35              ZHANG_SEPTEMBER19_OP <= '0';
36
37
38              for i in 0 to 256 loop
39                  for j in 0 to 256 loop
40                      wait for 10 ns;
41
42
43                      if ZHANG_SEPTEMBER19_OP = '0' then
44                          assert (ZHANG_SEPTEMBER19_S = ZHANG_SEPTEMBER19_X + ZHANG_SEPTEMBER19_Y) report "Outputed = " &integer'image(to_integer(unsigned((ZHANG_SEPTEMBER19_S))))& " expected X+Y = " &i
45                          ZHANG_SEPTEMBER19_OP <= ZHANG_SEPTEMBER19_OP xor '1';
46                      else
47                          assert (ZHANG_SEPTEMBER19_S = ZHANG_SEPTEMBER19_X - ZHANG_SEPTEMBER19_Y) report "Outputed = " &integer'image(to_integer(unsigned((ZHANG_SEPTEMBER19_S))))& " expected X-Y = " &i
48                          ZHANG_SEPTEMBER19_OP <= ZHANG_SEPTEMBER19_OP xor '1';
49                      end if;
50
51                      if ZHANG_SEPTEMBER19_Negative = '1' then
52                          assert (ZHANG_SEPTEMBER19_S(n-1) = '1');
53                      else
54                          assert (ZHANG_SEPTEMBER19_S(n-1) = '0');
55                      end if;
56
57                      -- would have been where i put overflow
58
59
60              ZHANG_SEPTEMBER19_Y <= ZHANG_SEPTEMBER19_Y + "0000000000000001";
61              end loop;
62              ZHANG_SEPTEMBER19_X <= ZHANG_SEPTEMBER19_X + "0000000000000001";
63              end loop;
64
65              report "Test finished";
66              wait;
67      end process;
68  end ZHANG_SEPTEMBER19_arch;
```

Image above is the testbench code

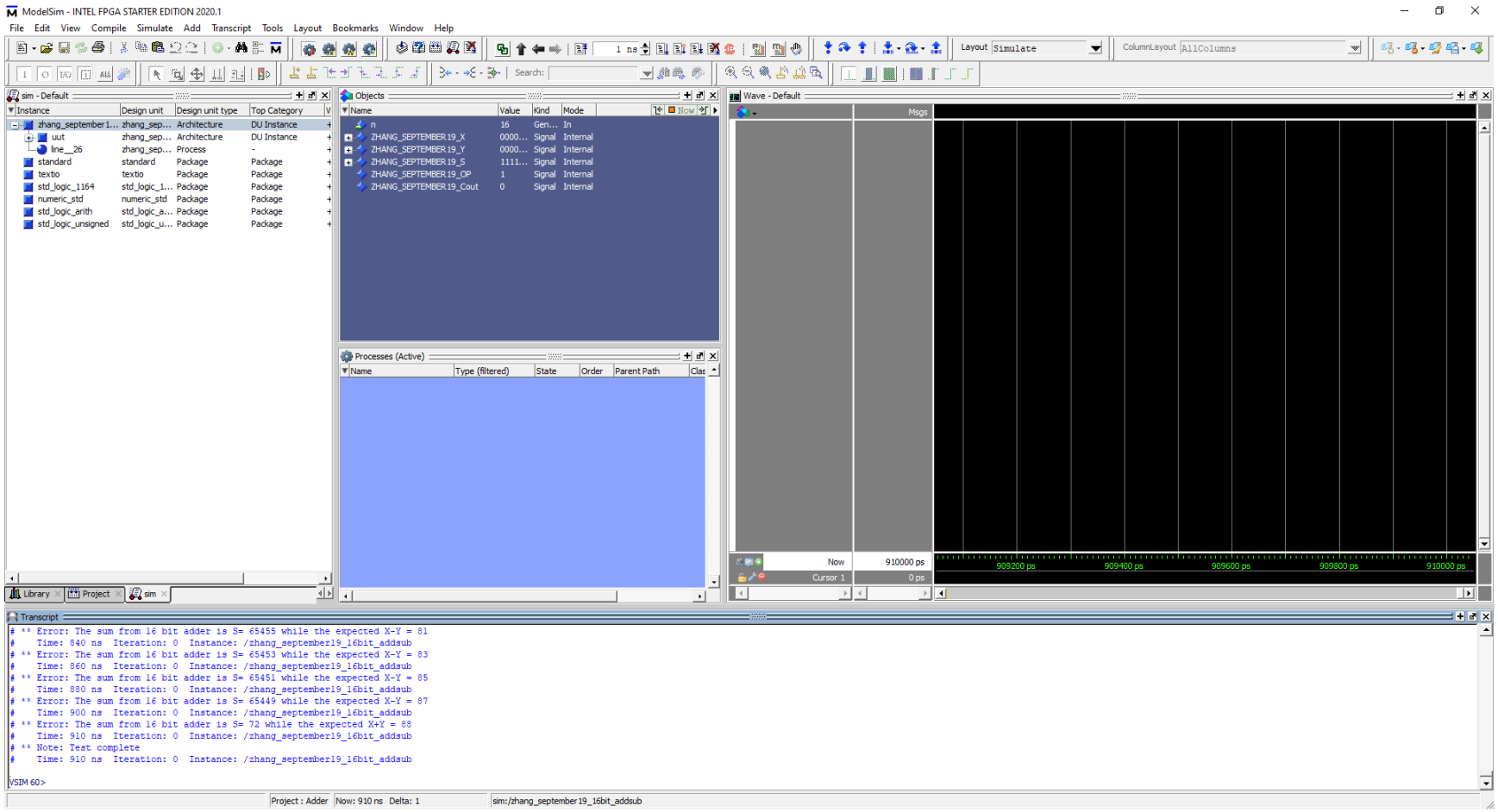
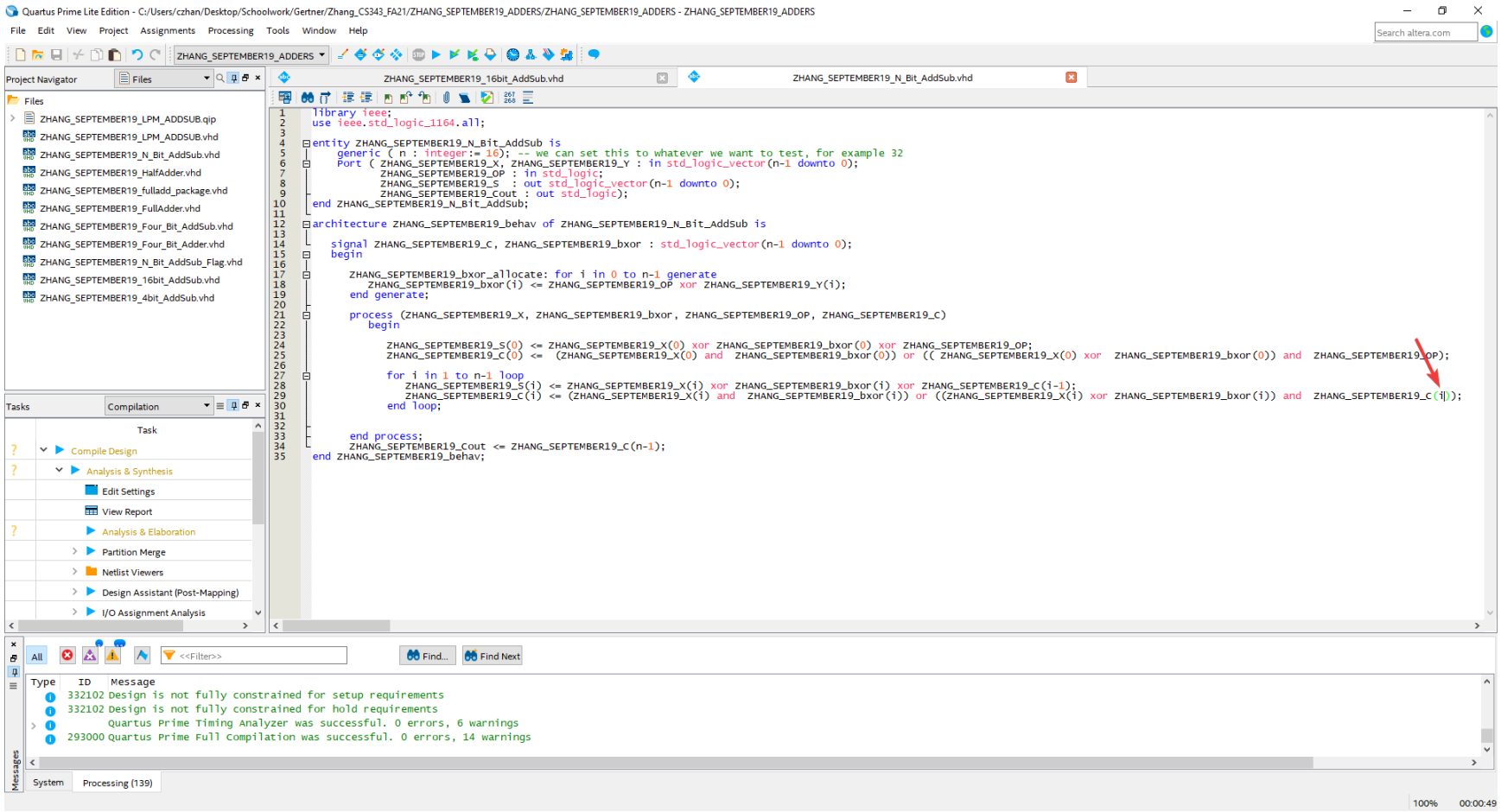


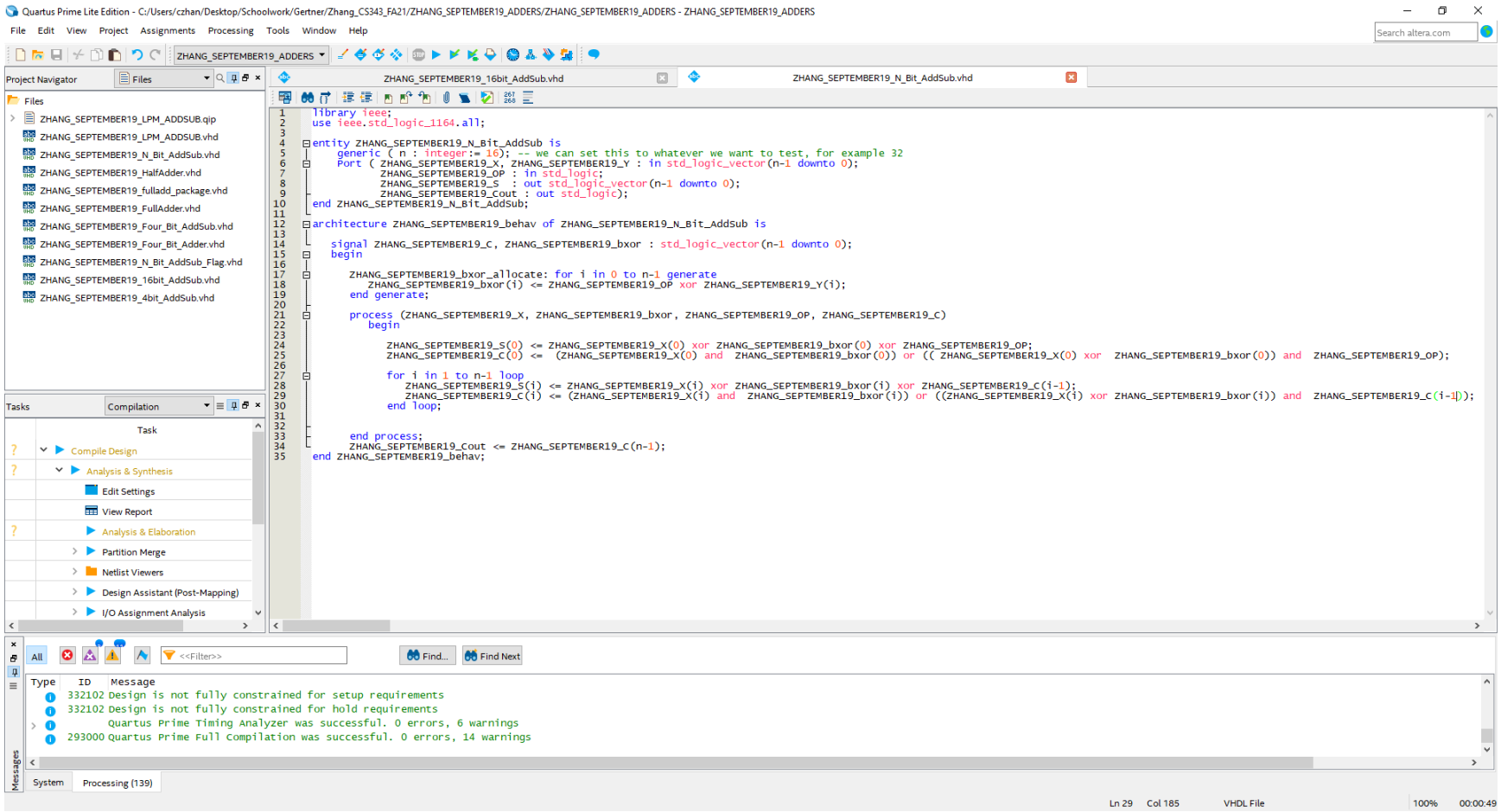
Figure above showcases the testbench and the errors if there is a fault in the code

Chue Zhang , Task 10 : Testbench NO FLAGS

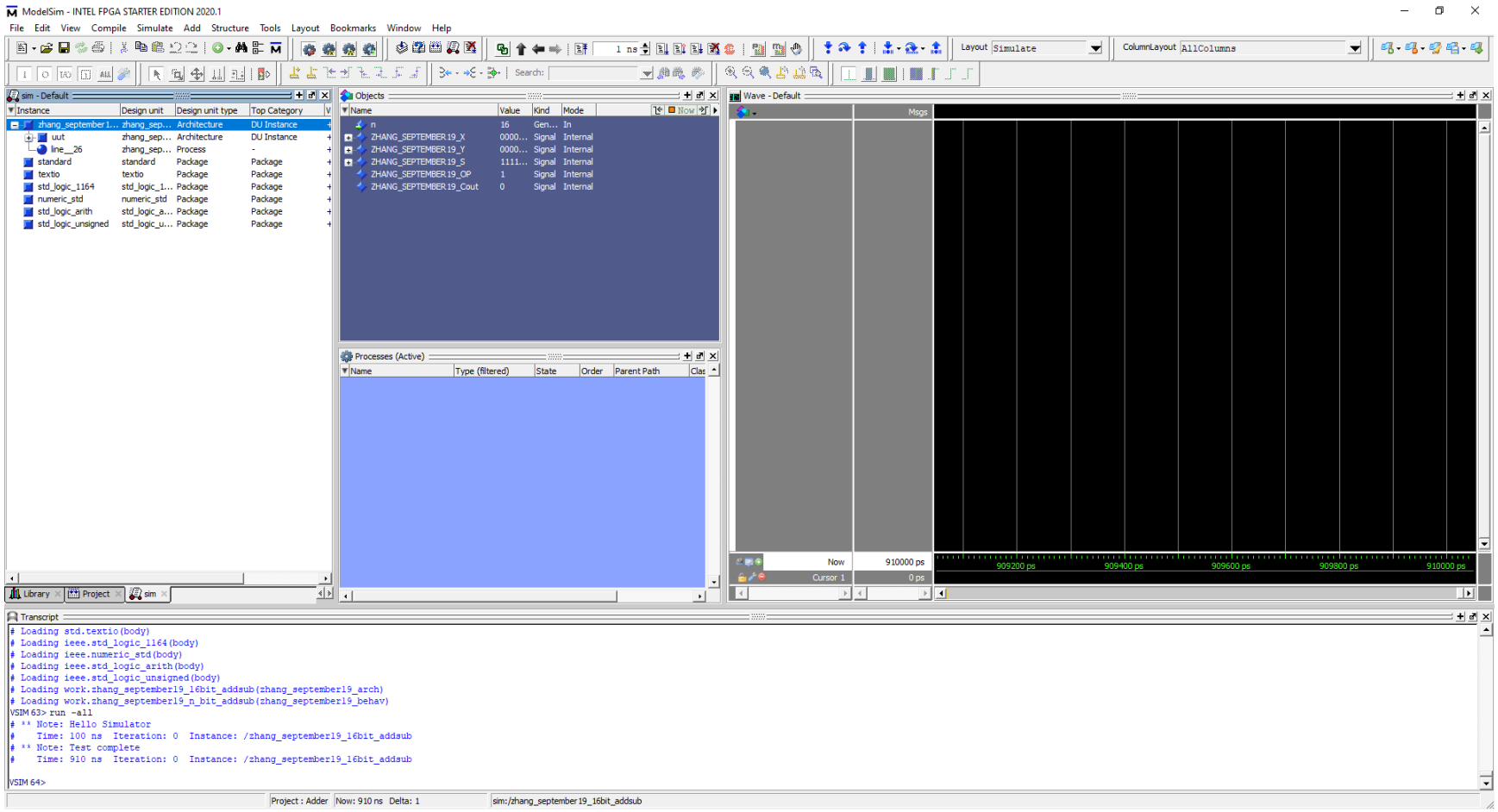


This was the code that we tested on our testbench, the red arrow points to where the fault is so we correct it.

Chue Zhang , Task 10 : Testbench NO FLAGS



Corrected N bit Add/Sub code to be simulated on



New test runs and we see that there are no errors created.