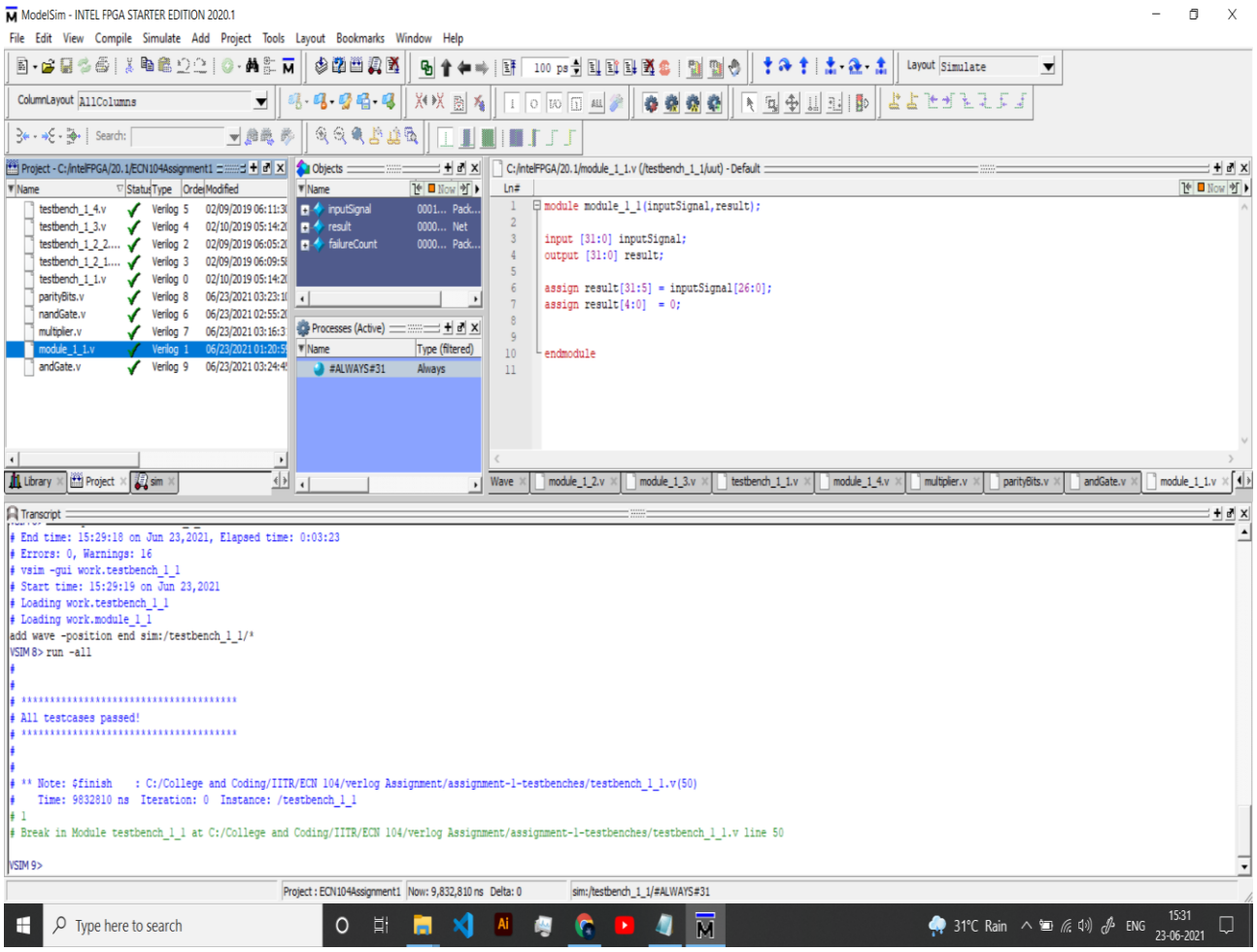
VERILOG ASSIGNMENT -1

RAHUL KURKURE
20114079 CSE O3

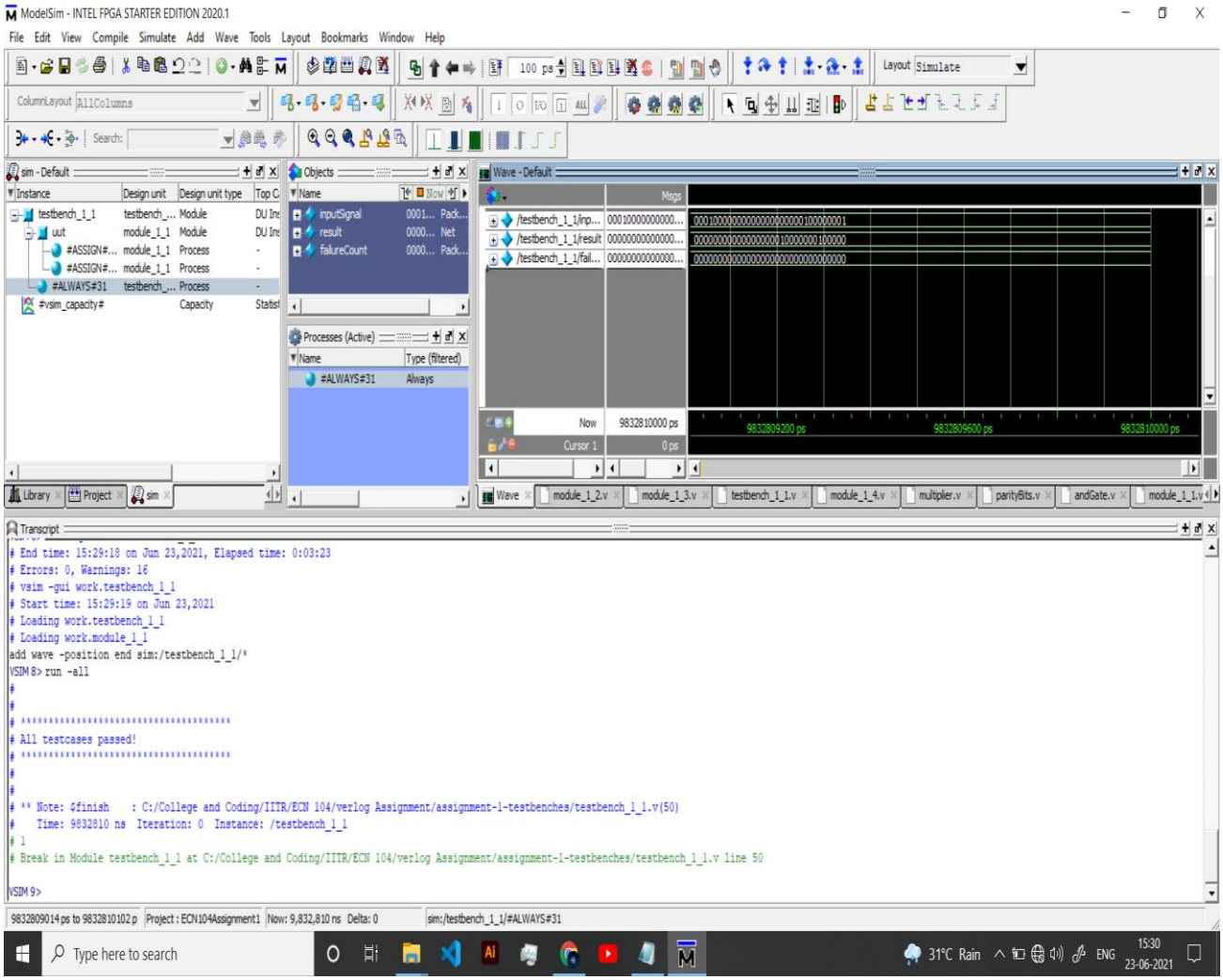
Q1. Write a Verilog module named module_1_1 which uses part select, takes a 32bit vector as input named inputSignal and outputs a 32bit vector result which is simply inputSignal shifted by 5 towards left. Use testbench_1_1.v to verify the output. (Hint: Similar to part select, Verilog supports part assign which allows assigning values to a part of a vector).

Explaination : we are going to use part select method in this question , we will set indices number 26 to 0 from index of input vector as the bits of indices 31 to 5 of result vector and for the bits number 4 to 0 we will set them as 0.

Source code :



Simulation :



Q2. Hierarchical design is often helpful in verifying large project easily by verifying each individual module separately. Make a module for 2 input NAND gate named nandGate. Now, make a 2 input AND gate named andGate using multiple of these NAND gates. Use testbench_1_2_1.v for verifying the output of NAND gate and testbench_1_2_2.v for verifying the output of the AND gate.

Explaination :

Here we are first going to make a nand gate by calculating the and of A and B using & operator ,then inverting the result will give us NAND of A and B .

NAND : ~ (A & B)

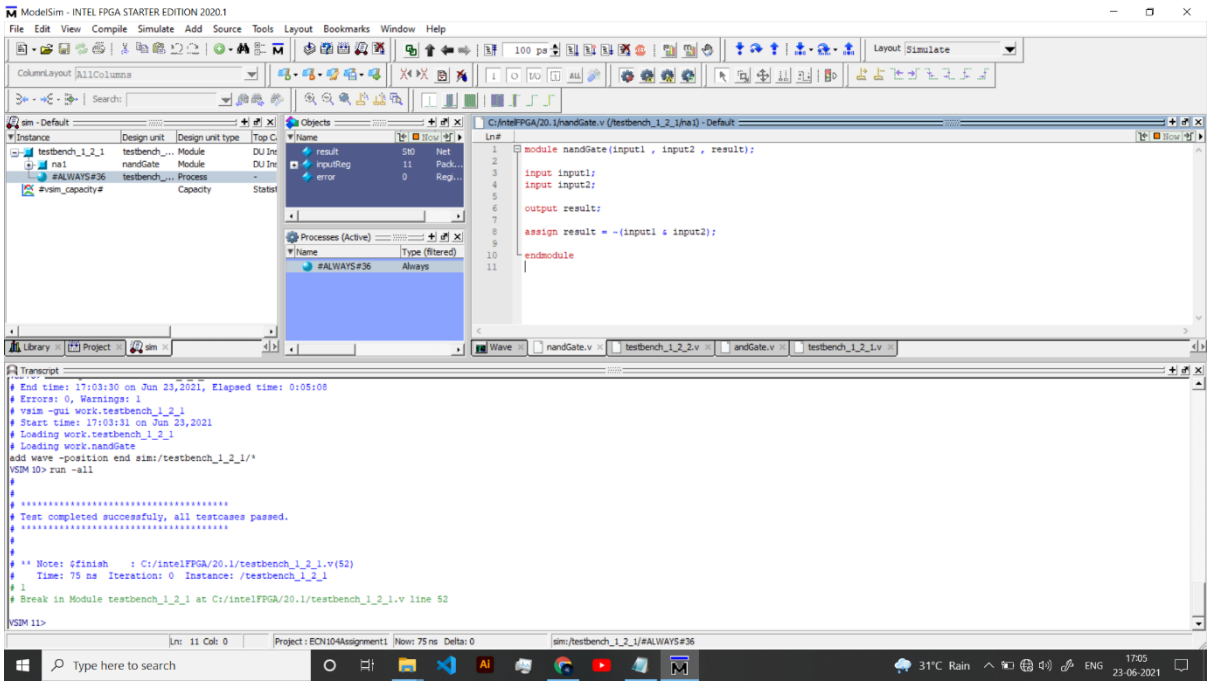
According to question we have to use NAND gates to make AND gates so first we will take nand of both inputs then we will take another NAND to invert our result .

Step 1 : A NAND B : (AB)' → A' + B'

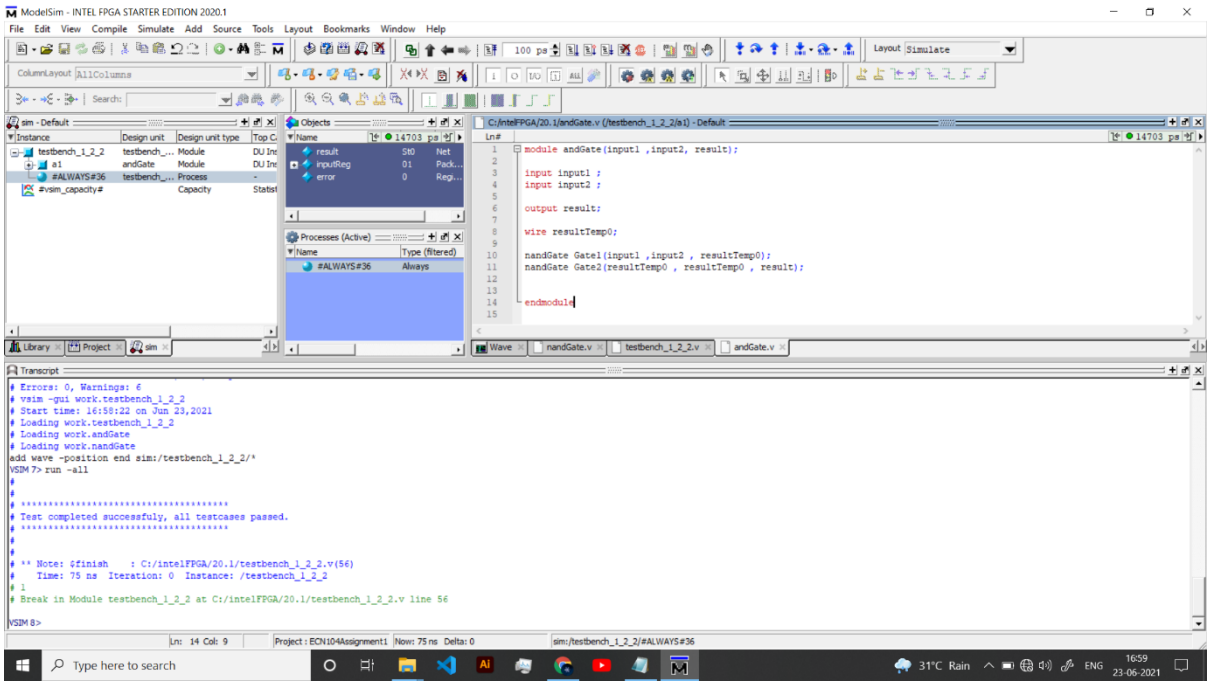
Step 2 : (AB)' NAND (AB)'

Result : '((AB)' (AB)')' → AB

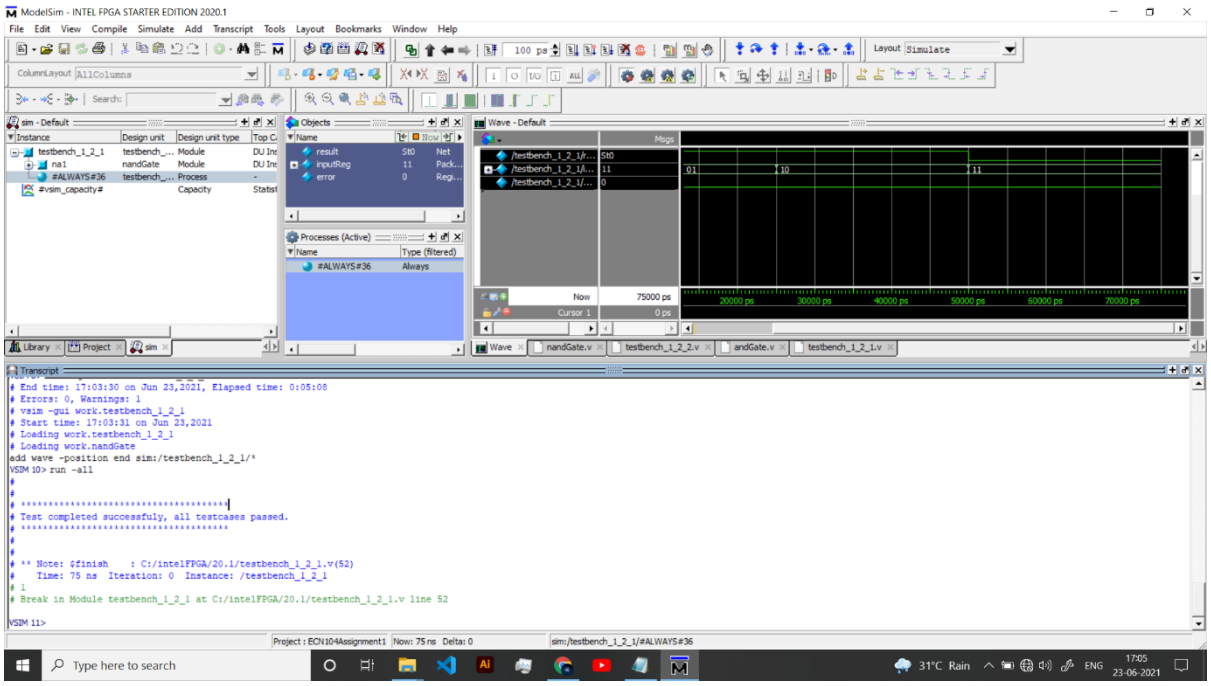
Source Code : NAND Gate :

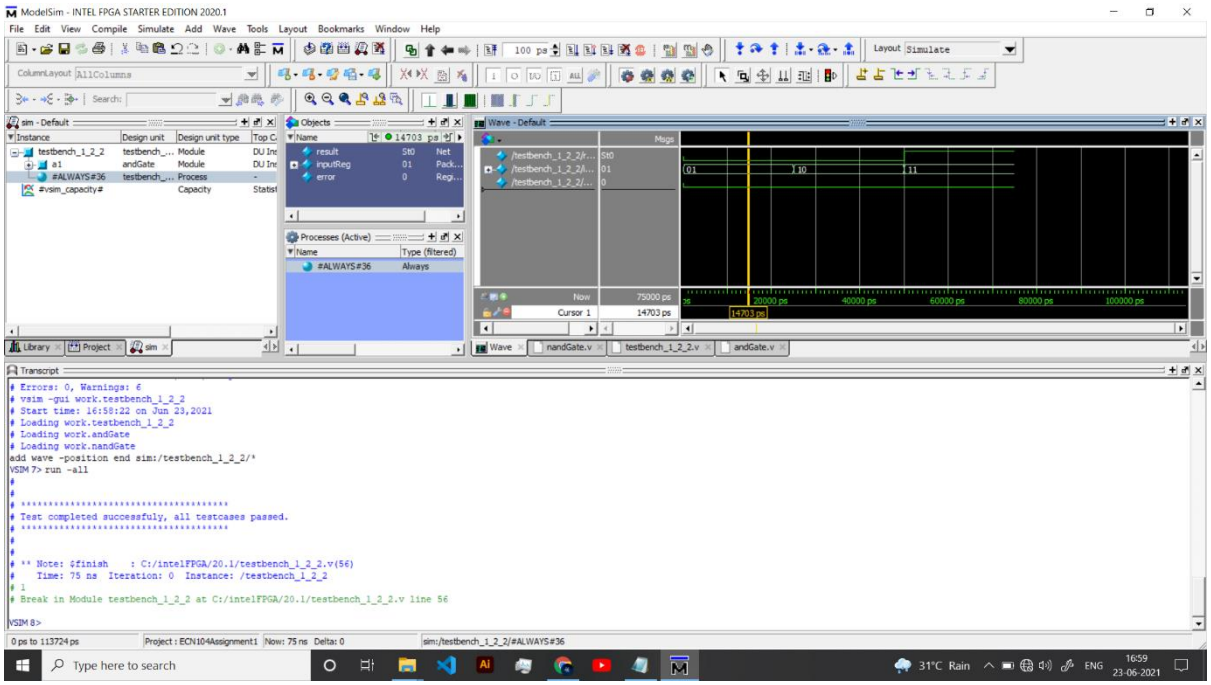


AND Gate :



Simulations :



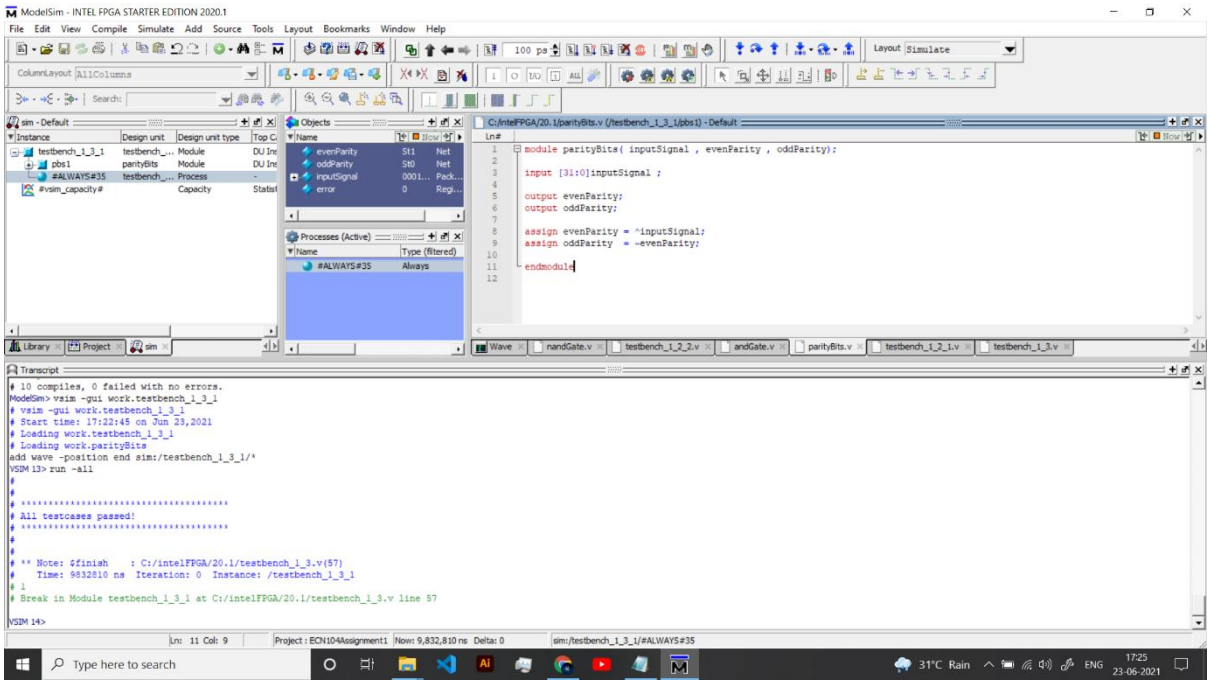


Q3. Write a Verilog module named parityBits which uses reduction operator, takes a 32bit vector as input named signal, two single bit output named parityEven and parityOdd. Use testbench_1_3.v to verify output. (Hint: Calculating XOR of all bits of a signal gives even parity of the signal.)

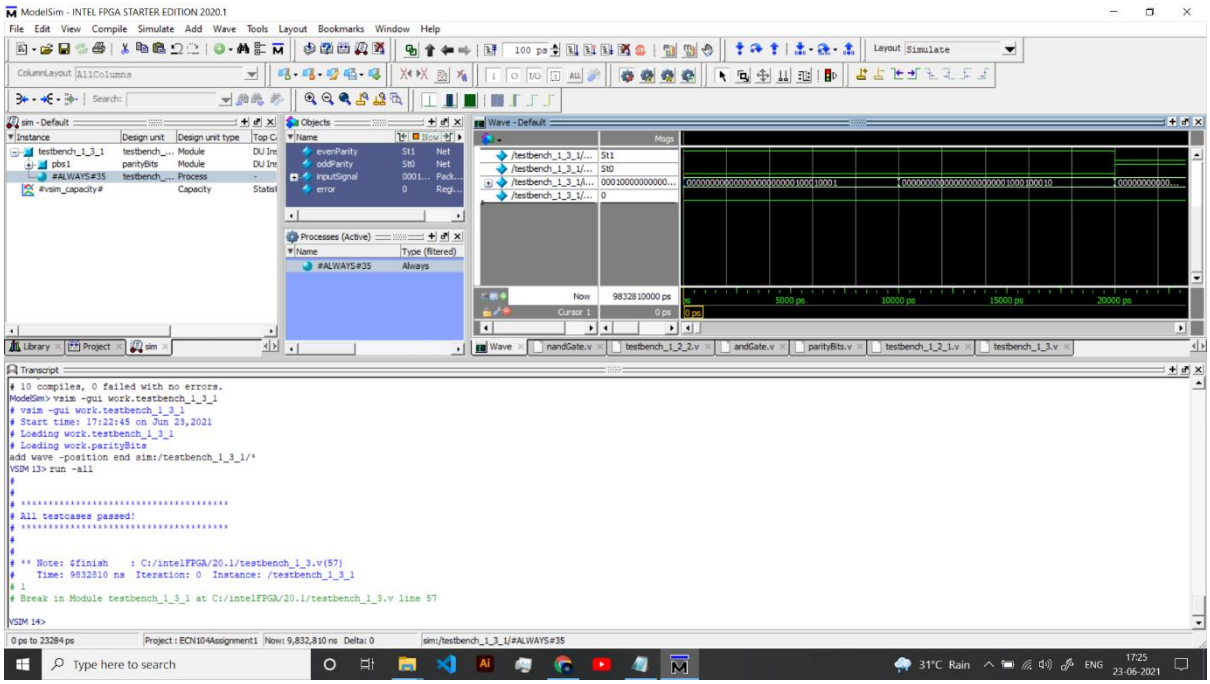
Explaination :

As hinted in the question we will use an ^ xor operator to get the even parity signal , if we just invert the even parity signal we will get an Odd parity signal .

Source Code :



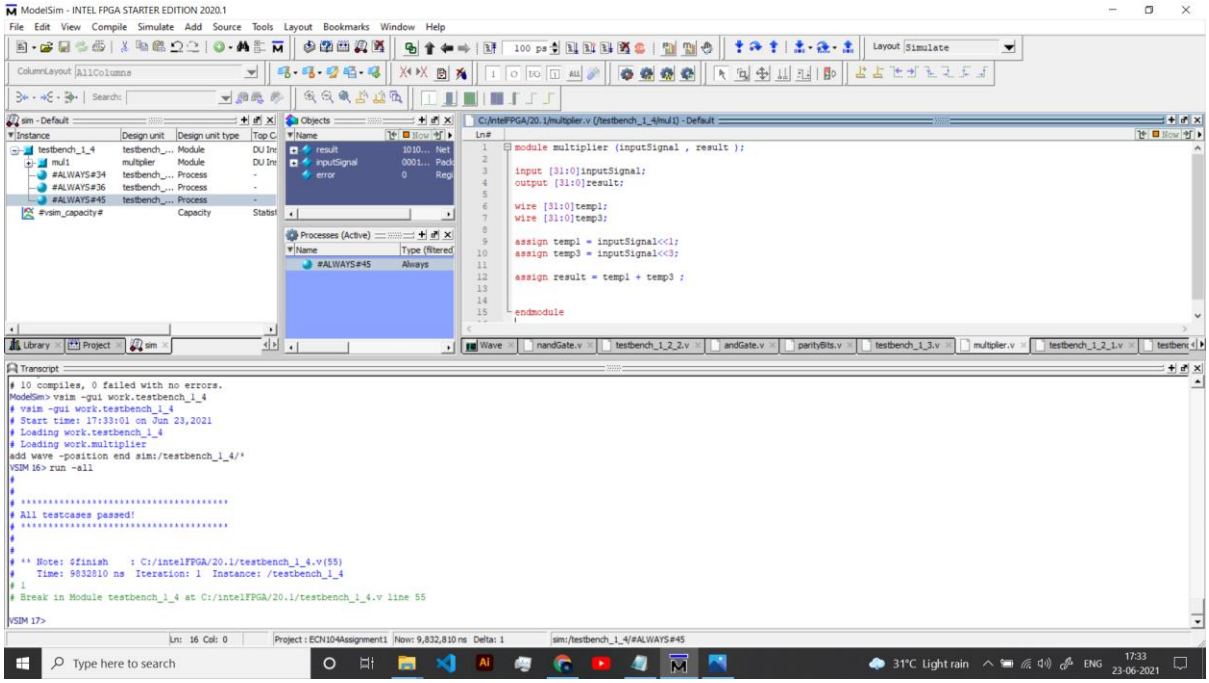
Simulation :



Q4. Write a module named multipiler for multiplying a 32-bit binary number by 10(4'b1010 in binary) without using the multiply operator. Assume the input is such that output doesn't overflow. Use testbench_1_4.v to verify output. RTL schematic of your module should resemble Fig. 4. The input to the module should be named inputSignal and the output should be named result to allow the testcases to run.

Explaination : Adding zeroes to the binary numbers to wont make any difference so we will completely ignore the multiplication by zero , for first one we will shift the bits to the left by 1 and for next time multiplication by 1 we will shift the bits by 3 units to the left and then finnaly add those two numbers.

Source Code :



Simulation :