Name: Abdullah Khan

Roll No. : 111801001

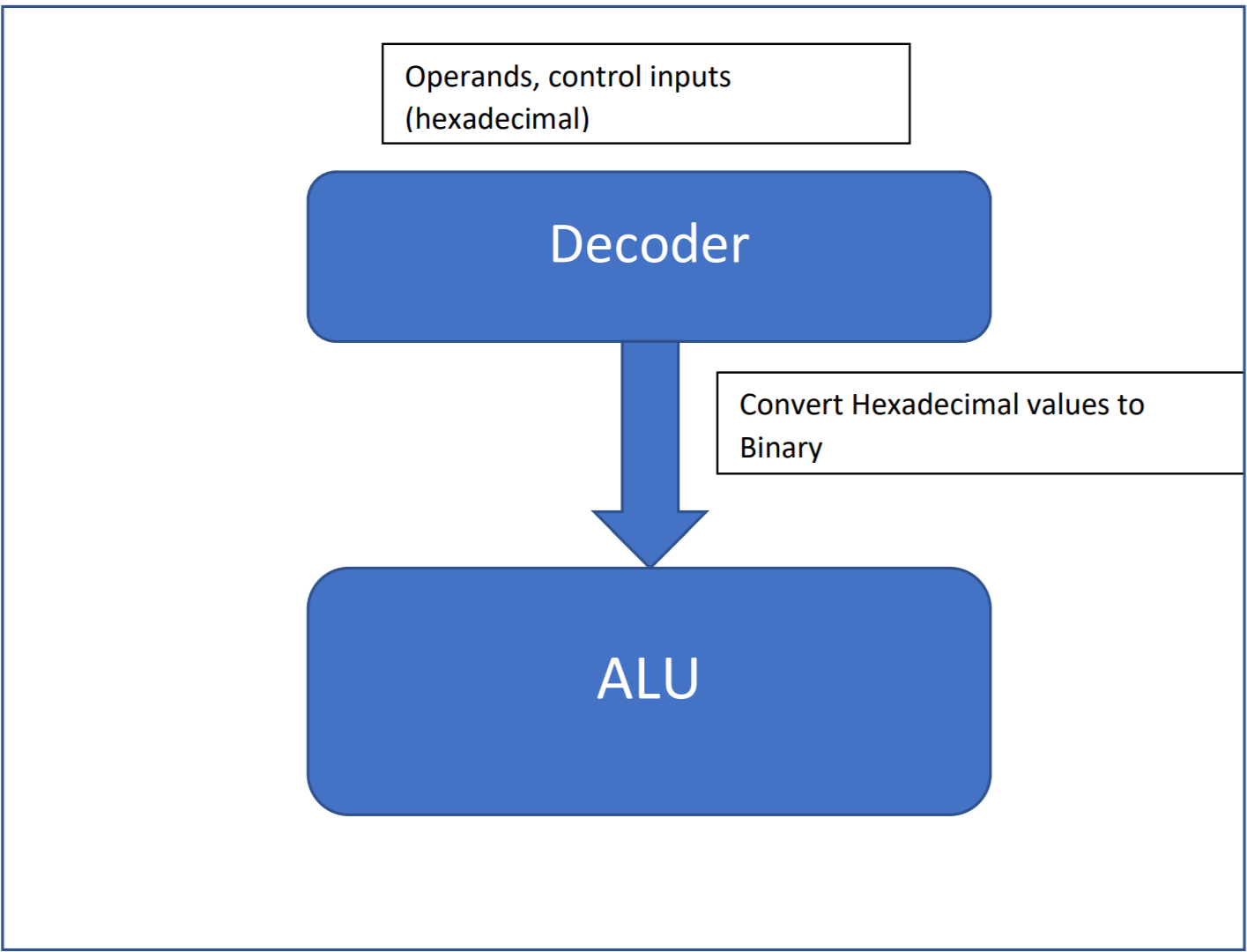
Course Name: CS2160 - Computer Organization Lab

**POST LAB REPORT - 1**horizontal line

# **Exp 1: 64-bit ALU** Completed on: 14 Jan’20

**Problem Statement**

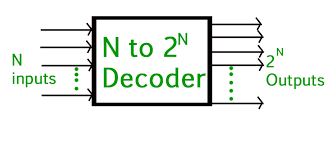
Design a 64 bit ALU for the RISC-V processor. The ALU consists of ADDER, Subtractor, Multiplier and perform logical operations AND, OR, XOR, NAND and NOT. You will be given a text file consisting of instructions in hexadecimal. You are required to design a decoder to convert hexadecimal numbers to binary and then pass it as input to your ALU.

****

**Theory**

* **Decoder:**

The name “Decoder” means to translate or decode coded information from one format into another, so a binary decoder transforms “n” binary input signals into an equivalent code using 2n outputs.



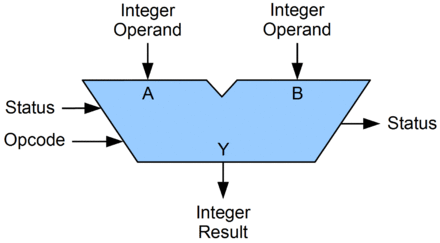
A binary decoder is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of 2n unique outputs.

There are two types of decoders:

1. A **1-of-n binary decoder:**  has n output bits. This type of decoder asserts exactly one of its n output bits, or none of them, for every integer input value. The "address" (bit number) of the activated output is specified by the integer input value. For example, output bit number 0 is selected when the integer value 0 is applied to the inputs.
2. **Code translator:** Code translators differ from 1-of-n decoders in that multiple output bits may be active at the same time. An example of this is a seven-segment decoder, which converts an integer into the combination of segment control signals needed to display the integer's value on a [seven-segment display](https://en.wikipedia.org/wiki/Seven-segment_display) digit.

* **ALU (Arithmetic Logic Unit):**

ALU is the fundamental unit of a microprocessor which performs all the basic operations based on the control input selection. There are separate units which work independently of the main ALU for performing secondary operations such as address computation. The ALU performs arithmetic functions such as addition, subtraction etc and logic functions including, logical AND, logical OR, and logical XOR etc. These various functions of the ALU are implemented using a set of functional units each implementing a function. It may also be done using sharing of the same hardware with the use of certain additional units like multiplexers.



|  |  |  |
| --- | --- | --- |
| **Function** | **Select Line** | **Operation** |
| NOT | 0000 | ~A |
| Subtraction | 0001 | A - B |
| Multiplication | 0010 | A \* B |
| AND | 0011 | A & B |
| OR | 0100 | A | B |
| XOR | 0101 | A ^ B |
| NAND | 0110 | ~ ( A & B ) |
| Addition | 0111 | A + B |

Above table shows various functions performed by proposed ALU. These various functions of the ALU are implemented using a behavioural modelling in Verilog.

* **Top Module:**

**Input**: The input consists of a sequence of 33 digits composed of three hexadecimal numbers. First one is the select line for operation selection followed by two 16 digit numbers representing two 64-bit inputs in hexadecimal form.

**Decoder:** takes the input and separates them into their respective entries along with converting the hexadecimal values to binary.

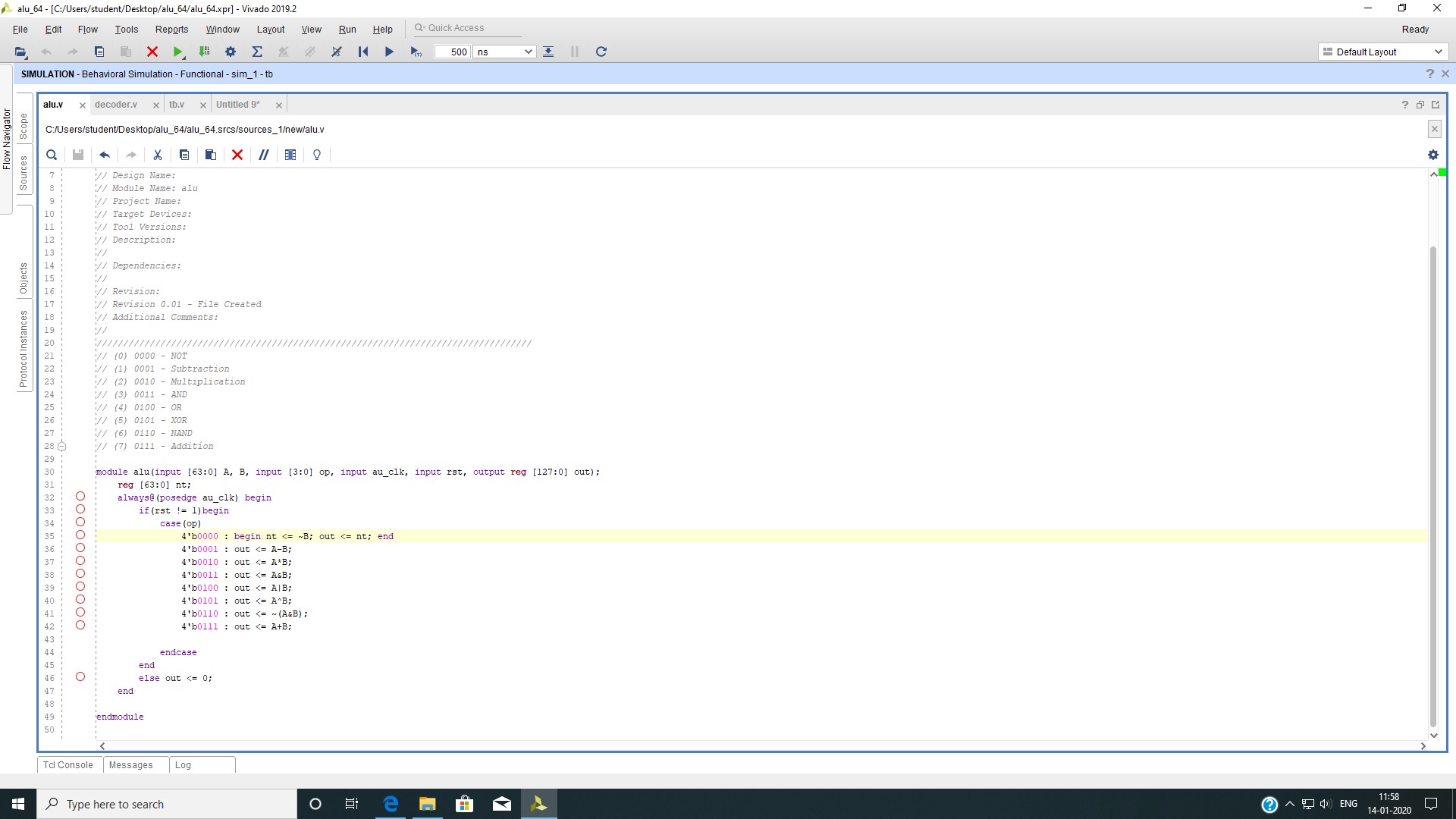
**ALU:** has 3 set of input signals and one output signal. Operands A and B are both 64 bits each. These are fed to the functional units along with select lines which decide the operation to be performed. Each combination of the select lines corresponds to one particular function.

## **Procedure**

* Open Vivado and create a new project, then select zynq7000-clg400 board from the drop-down list.
* Start writing the code in a modular fashion, with separate modules for both the ALU and Decoder.
* Create a test-bench for ease of testing and debugging.
* Simulate the project and verify the result under the simulation tab.

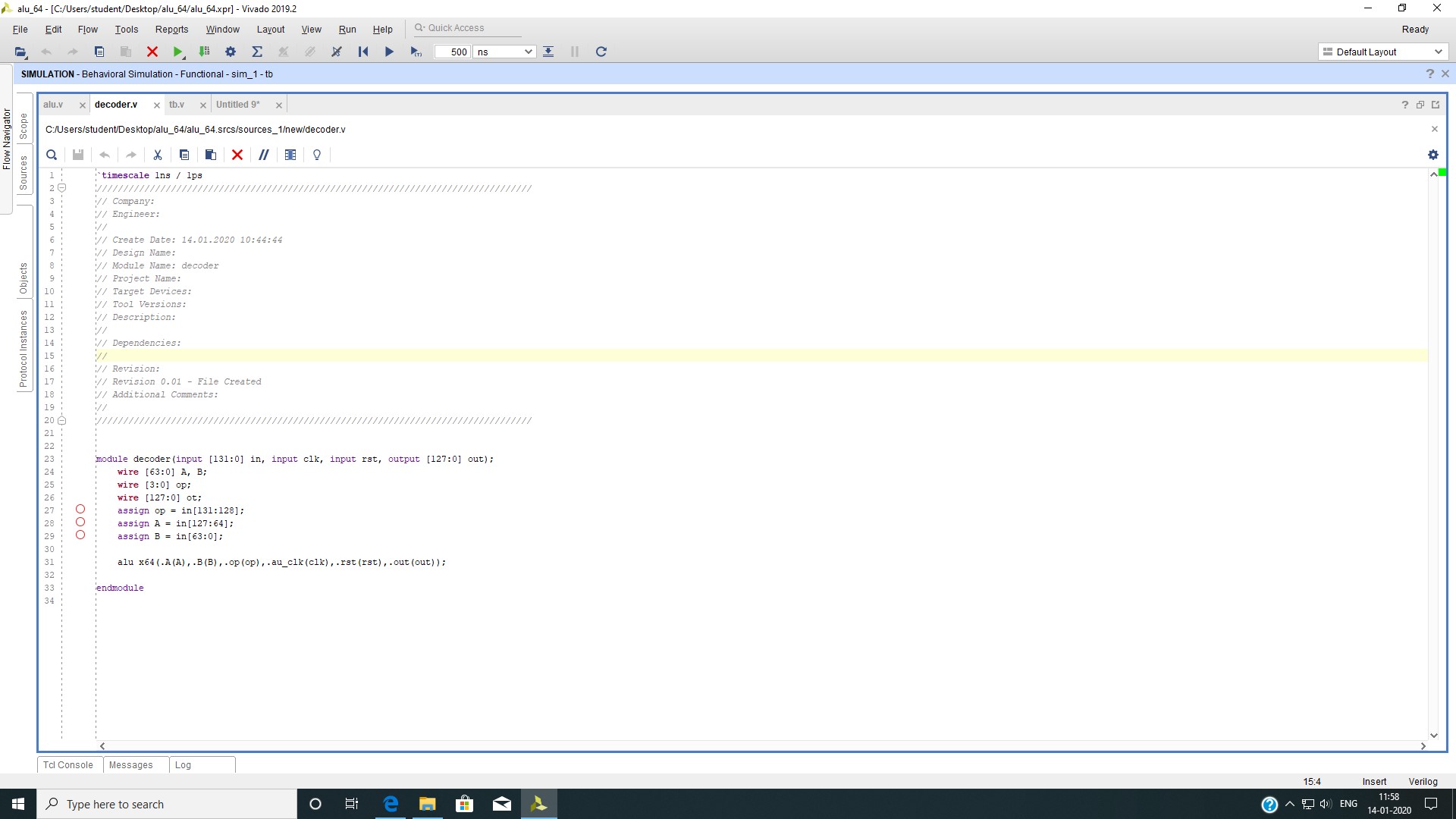
## **Code**

* **ALU**

****

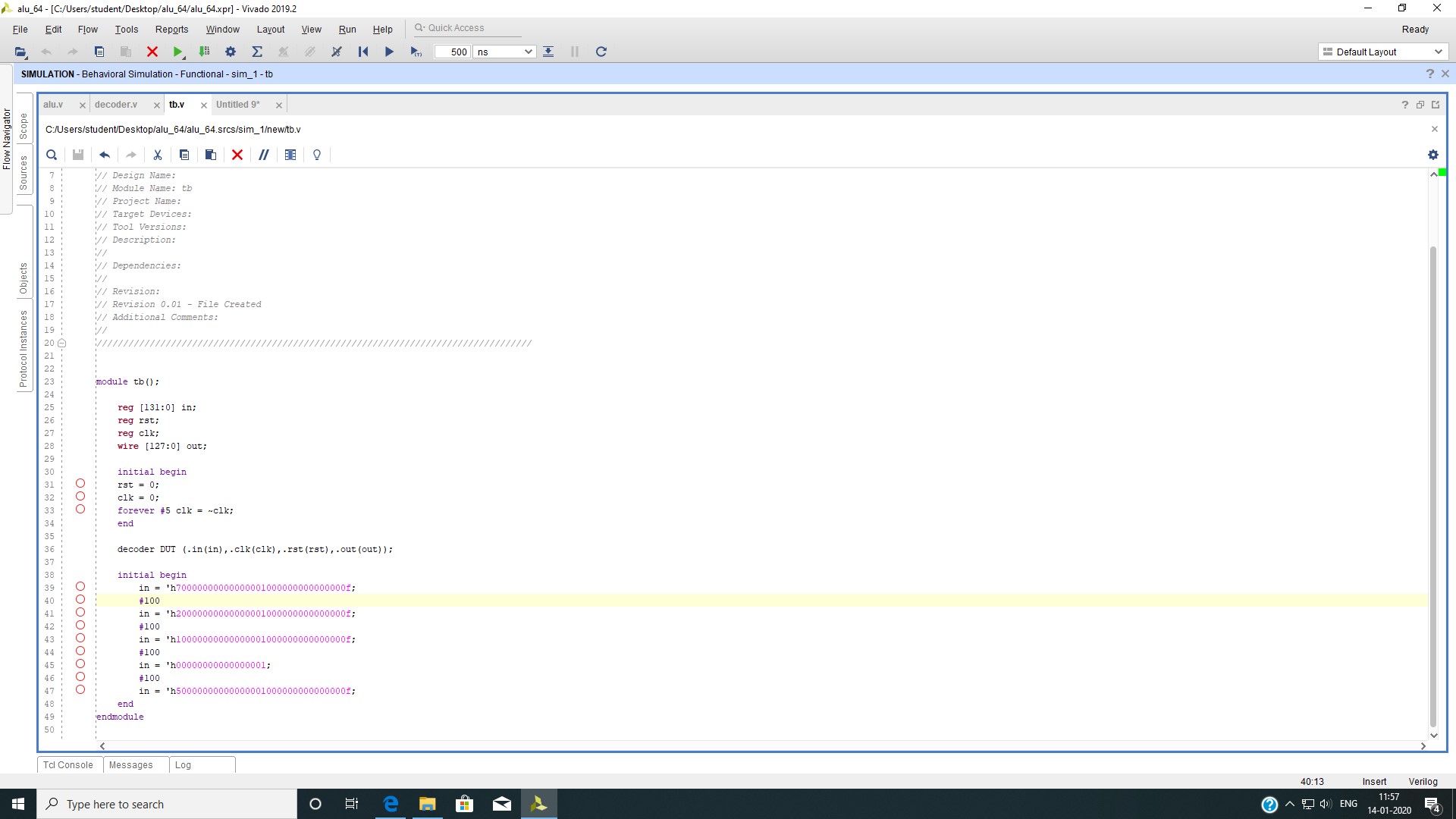
In the ALU module, each functional modules are implemented using high-level behavioural modelling due to the time constraints during the lab. Henceforth the performance of each individual as well as overall performance is not measured.

* **Decoder**



The decoder module simply segregates the single line input given to it into its valid constituents to be passed onto the ALU. As the input values are entered manually in the test-bench code we take advantage of the fact that any value is stored as binary in our system and thus there is no need to convert the values.

* **Test-Bench**

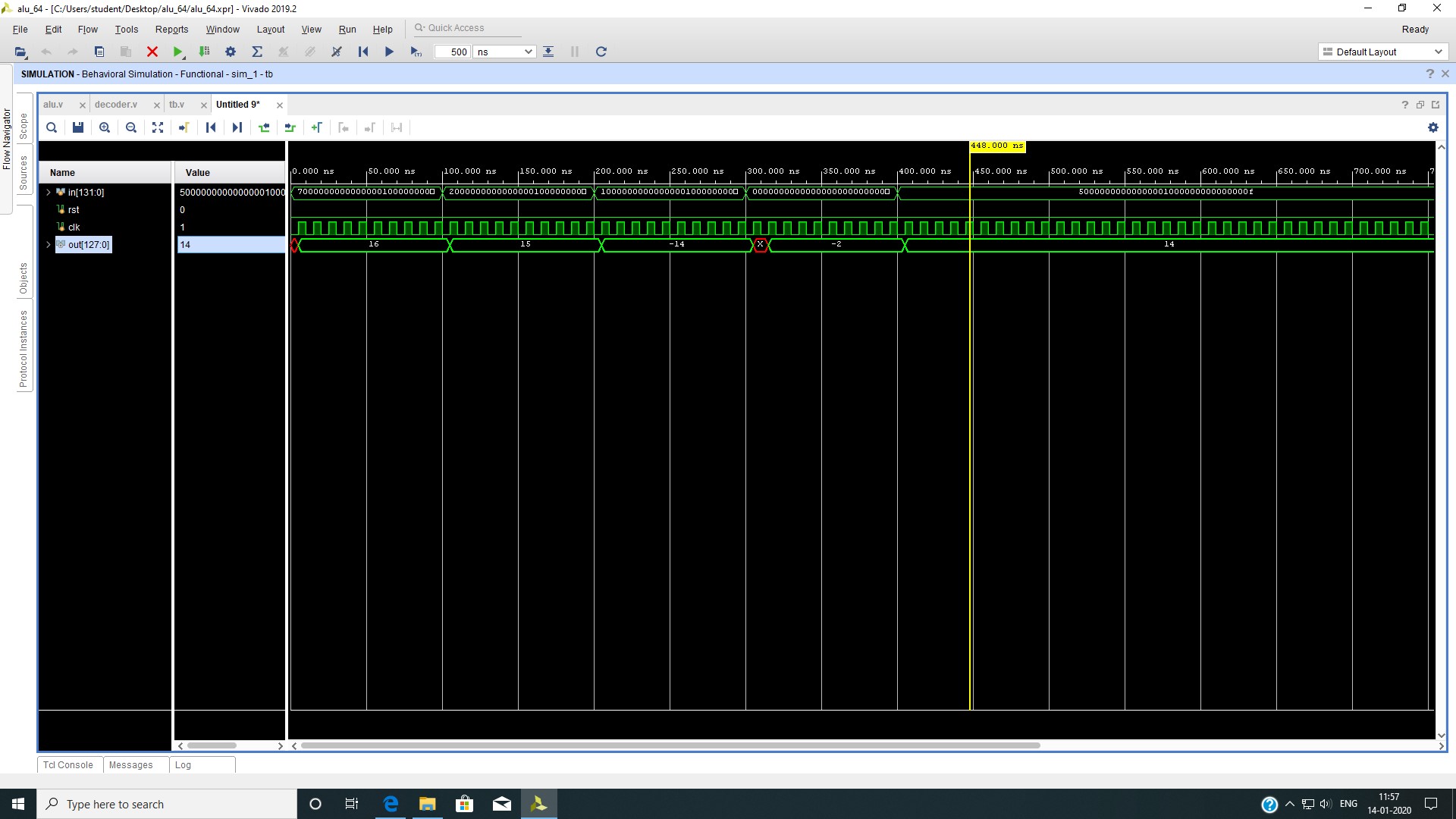


This is a very simple testbench intended only for ease of input formatting and thus requires manual checking of the output through the simulation tab.

## **Observations**

It was observed that the program was working correctly for the all the inputs given through the test-bench and was verified manually through the simulation tab.

Below is a figure that shows the program run for the inputs given in the test-bench (shown above).



The output shown in the figure is in unsigned decimal form.

## **Result**

The 64-bit arithmetic and logic unit is designed using VHDL and synthesized using Vivado Design Suite. Operands are received and fed to the unit corresponding to the operation to be performed and the result is produced by that unit and fed to the output lines of the ALU. We have verified the results obtained from the simulations with the theoretical results for all the operations that were performed and found that they match with the theoretical values.