

**CECS 530 - Lab 1**

**“Multiplexer”**

**Due date: 08/31/21**

Student Name: Jeremy Escamilla

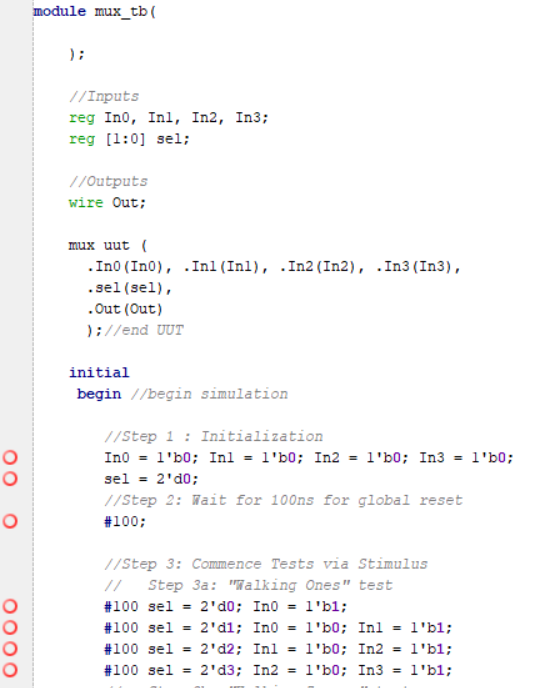
Student ID: **014236545**

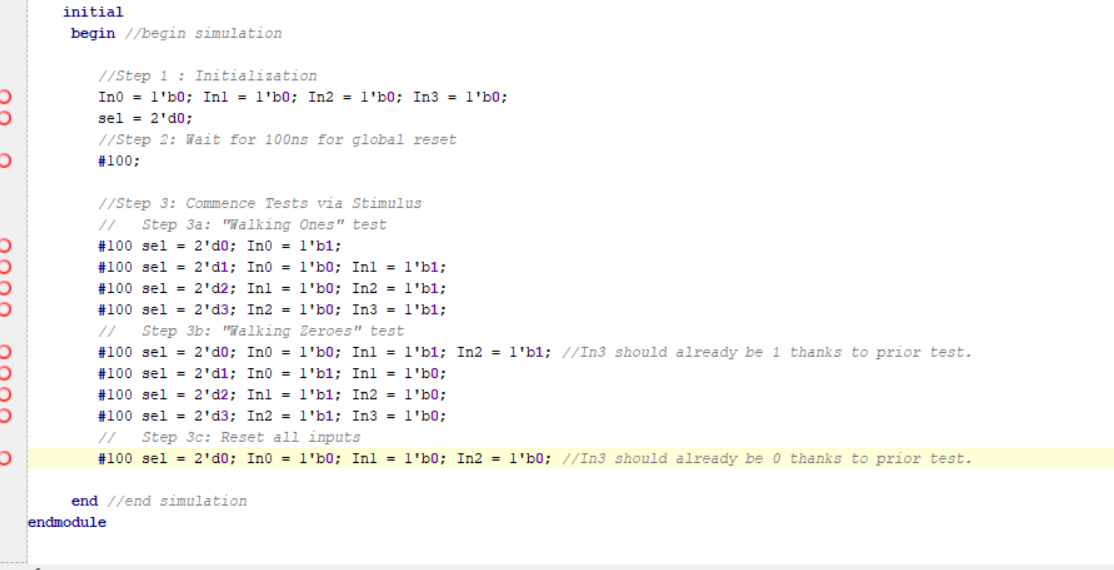
I certify that this submission is my original work

Jeremy Escamilla

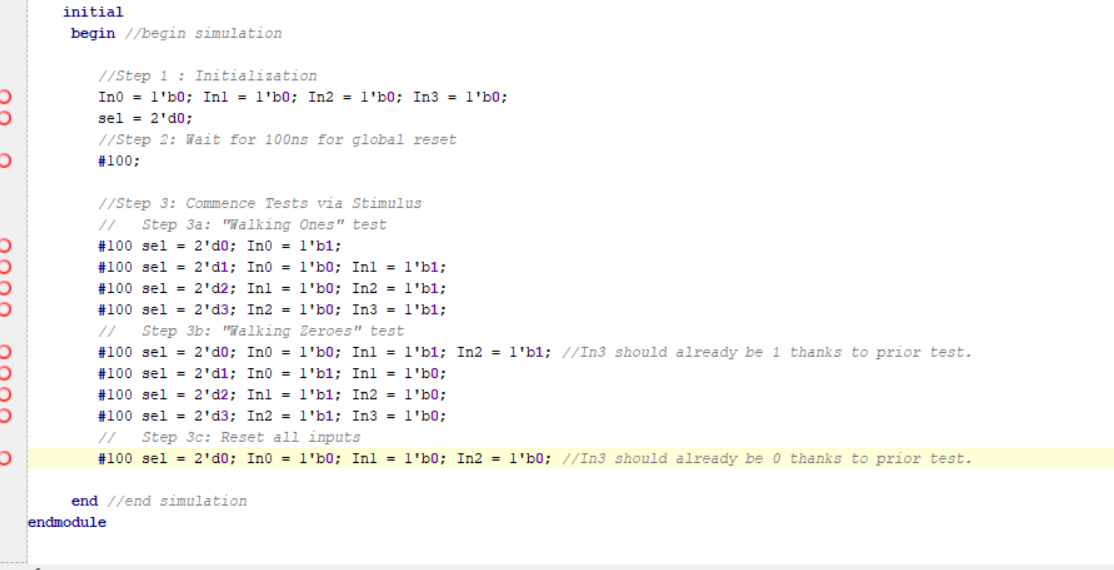
Lab Report: Lab Assignment 1 - “Multiplexer”

1. **Goal:** The End-Goal of this project was to successfully code and simulate a Multiplexer in Vivado.
2. **Steps:**
   1. Step 1: Create Multiplexer module.
   2. Step 2: Create the Multiplexer testbench.
      1. For initialization phase:
         1. Prepare registers (inputs) and wires (outputs), initialize Unit Under Test. Initialize registers, wait ~100 ns for global reset, then begin stimulus phase of test bench.

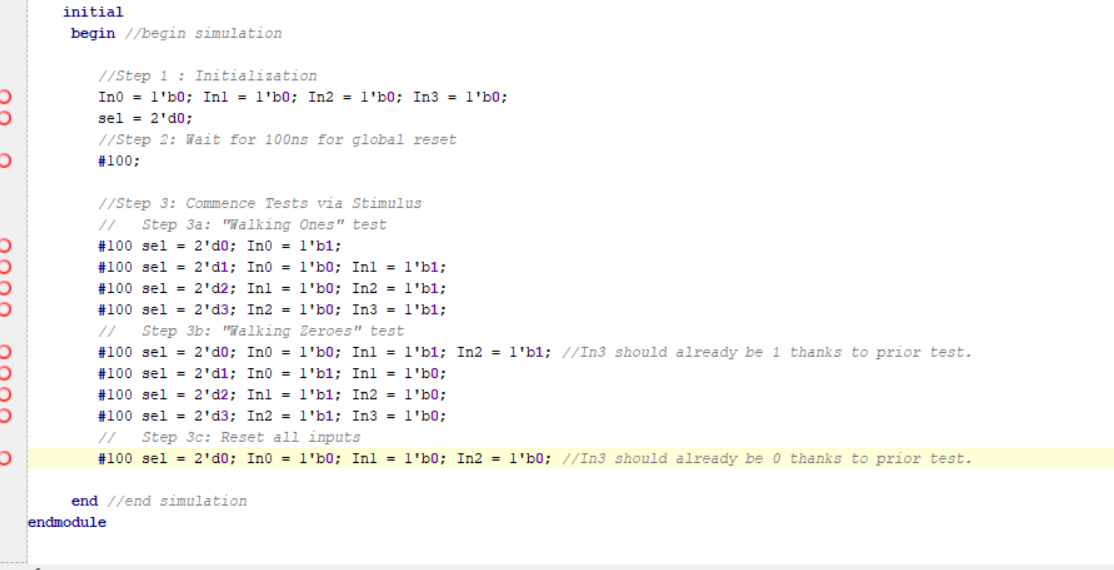




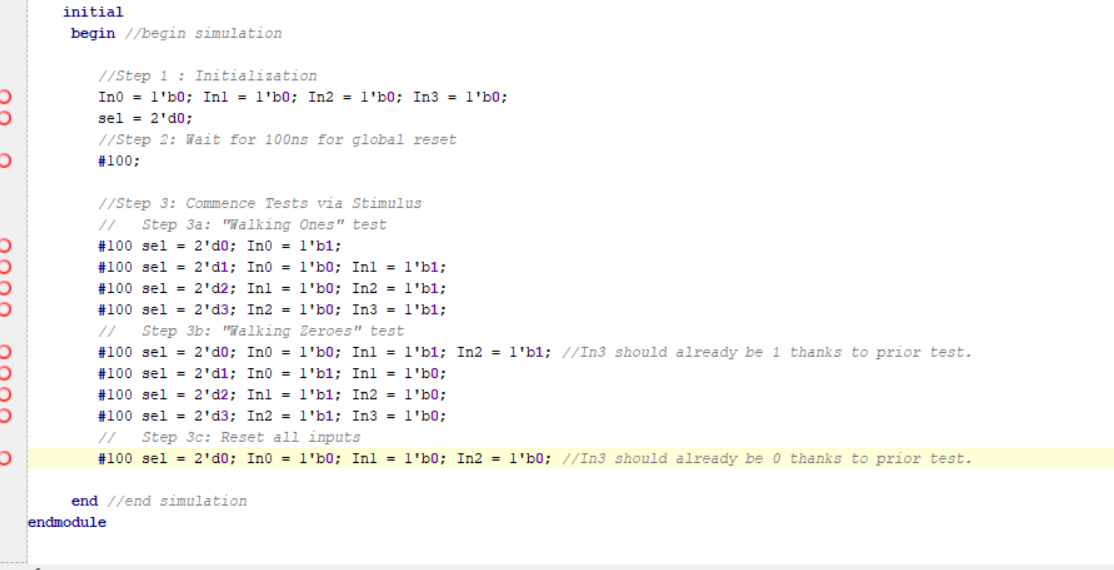
* + 1. For the stimulus phase:
       1. The first test done is the “walking ones” test. This test is done by setting one input out of the four as high (‘1’) while the rest are low (‘0’), and shifting the ‘1’ bit right, while the selector is set to a value that corresponds to the current position of the shifted bit. The expected result is that the output register “Out” has the value of a 1-bit ‘1’, as the output value should “follow” the “walking” bit.



* + - 1. The second test done is the “walking zeroes” test, which is the inverse of the “walking ones” test. The process of shifting a singular contrasting bit and ‘following’ it via the mux selector is the same, however, one input is set to low (‘0’) while the others are high. The expected result is that the output has the value of a 1-bit ‘0’ accordingly instead of ‘1’.

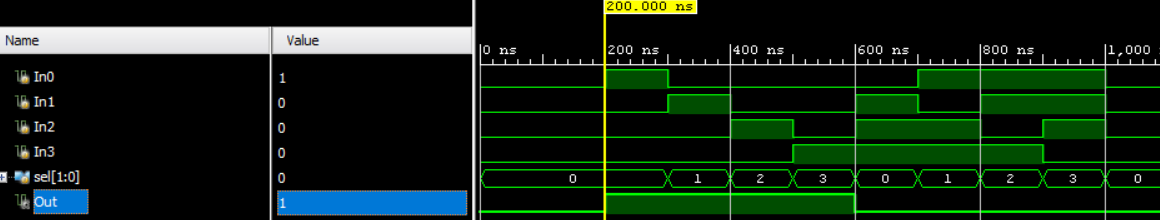


* + 1. For the Closing Reset:
       1. All the inputs are reset to ‘0’, and the output is expected to follow accordingly.



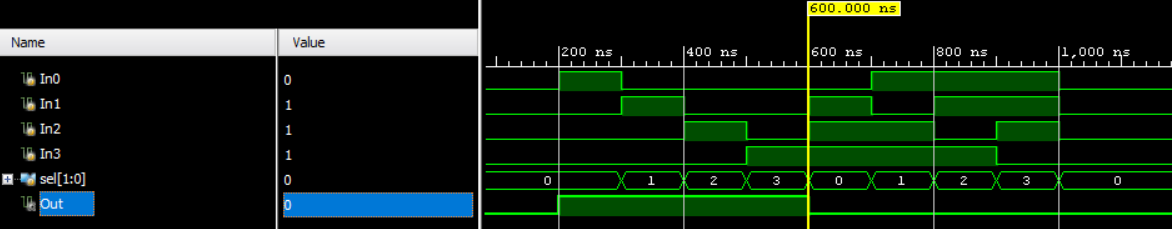
1. **Results:** The results obtained matched expectations. In the initialization phase of waveform, the output on the waveform was ‘0’, as it should be.

\*As seen in 0 – 200ns.



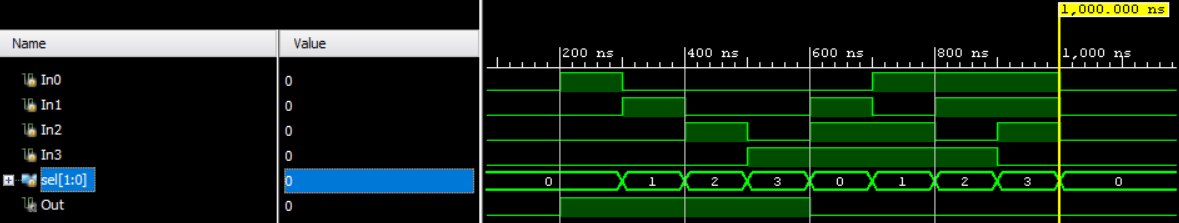
In the stimulus phase, the outputs followed the corresponding walking bit in each given test: for the “walking ones” test, the output only reflected the input holding the walking ‘one’.

\*As seen in 200 – 600 ns



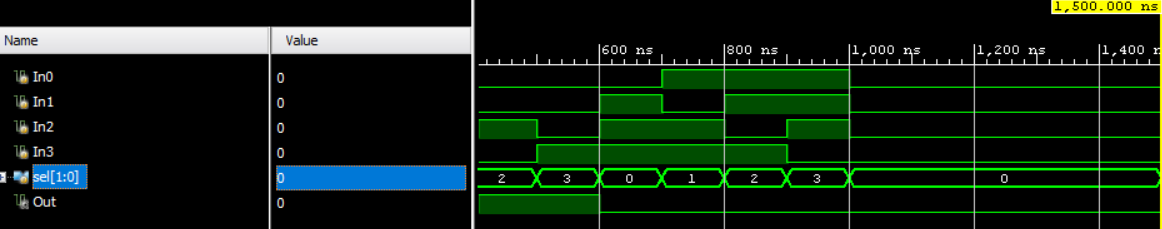
Similarly, in the “walking zeroes” test, the output only reflected the input holding the walking ‘zero’.

\*As seen in 600 – 1000 ns.



Lastly, the output for the closing reset was appropriately ‘0’ as the outputs reflected the first input, which like all other inputs, was set to ‘0’.

\*As seen in 1000+ ns.



1. **Conclusion:** I re-learned how to properly use Vivado, how to create and program a module, and create a functional testbench that properly simulates the prior module. Additionally, I re-learned the basics of implementing a mux via case statement and accounting for the output register. One challenge I faced during this lab was that I had spent over 6 months away from Vivado, Verilog, and programming in general, and as such I’ve been in desperate need of a refresher. It took a moment to properly explore Vivado, and ensure any syntax errors were cleared before simulation. Luckily, there were no logic errors when simulating in this lab.