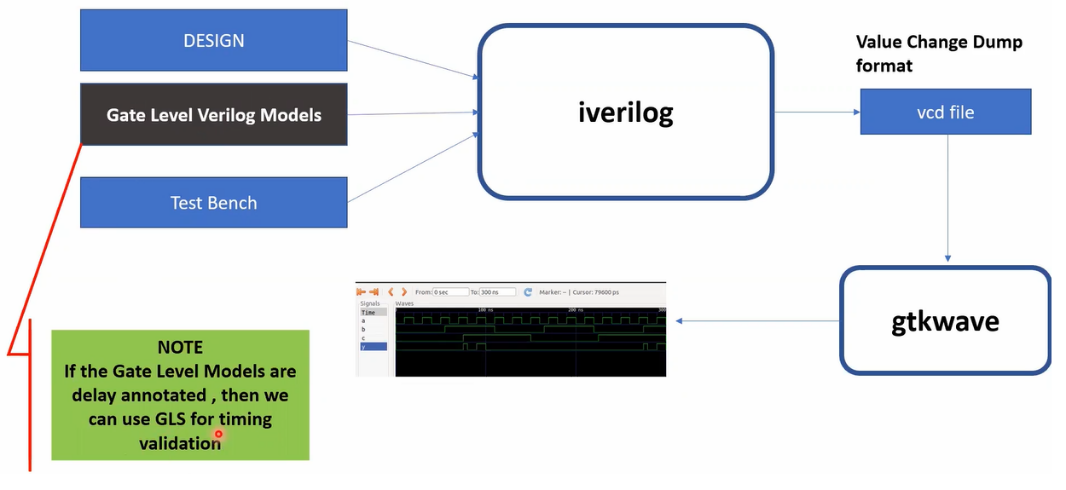
**GLS, Blocking vs Non-Blocking and Synthesis-Simulation Mismatch**

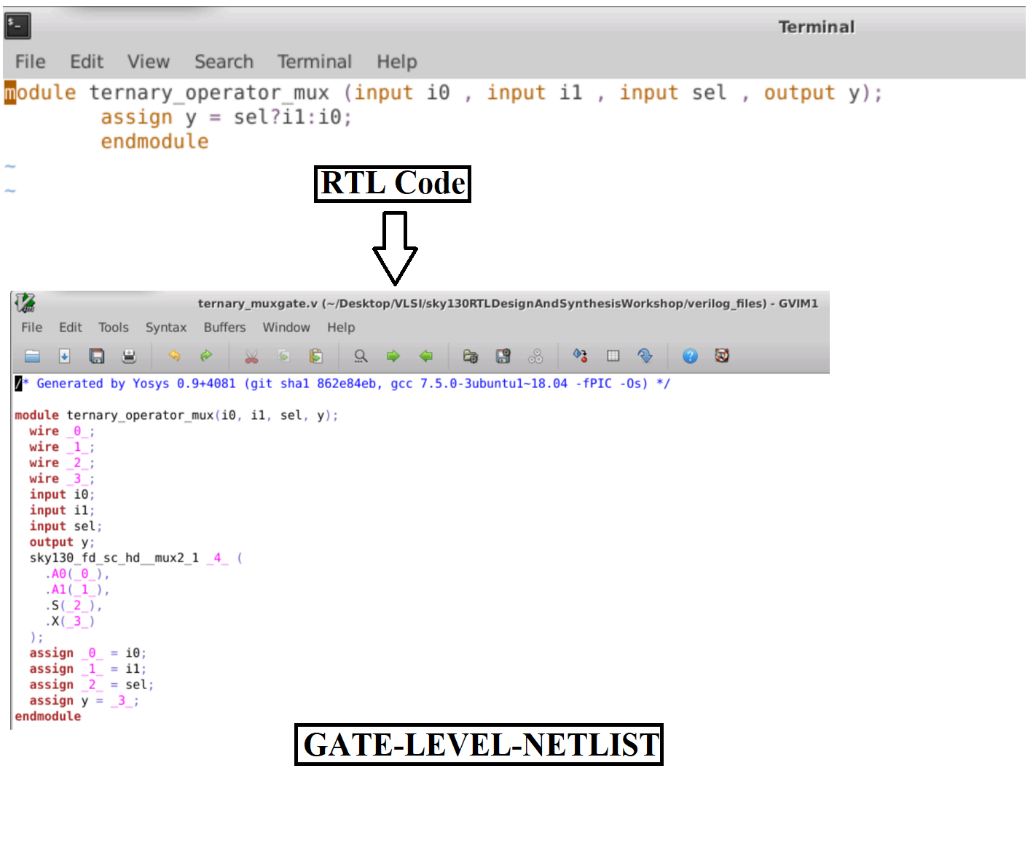
Gate-Level-Simulation (GLS):

* Running the test bench with Netlist as Design Under Test (DUT)
* Netlist is logically same as RTL Code.
  + Same Test Bench will align with the Design.
* GLS verifies the logical correctness of design after synthesis
* Ensuring the timing of the design is met.
  + For this GLS needs to be run with delay annotation



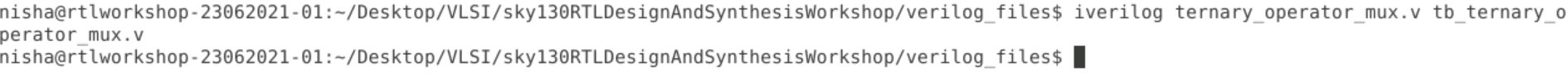
**Fig: GLS input and output**

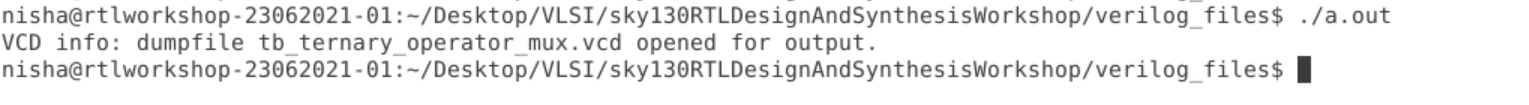
The Yosys tool maps the design in RTL to gate level netlist like in figure above using the gate level Verilog model during synthesis. The gate level Verilog model can be functional and/or timing aware.

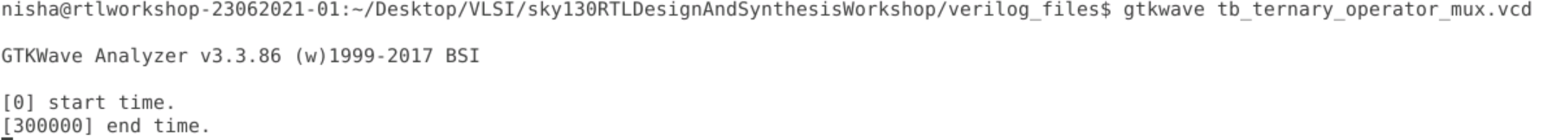


**Fig: Mapping Of RTL Code to Gat-Level-Netlist**

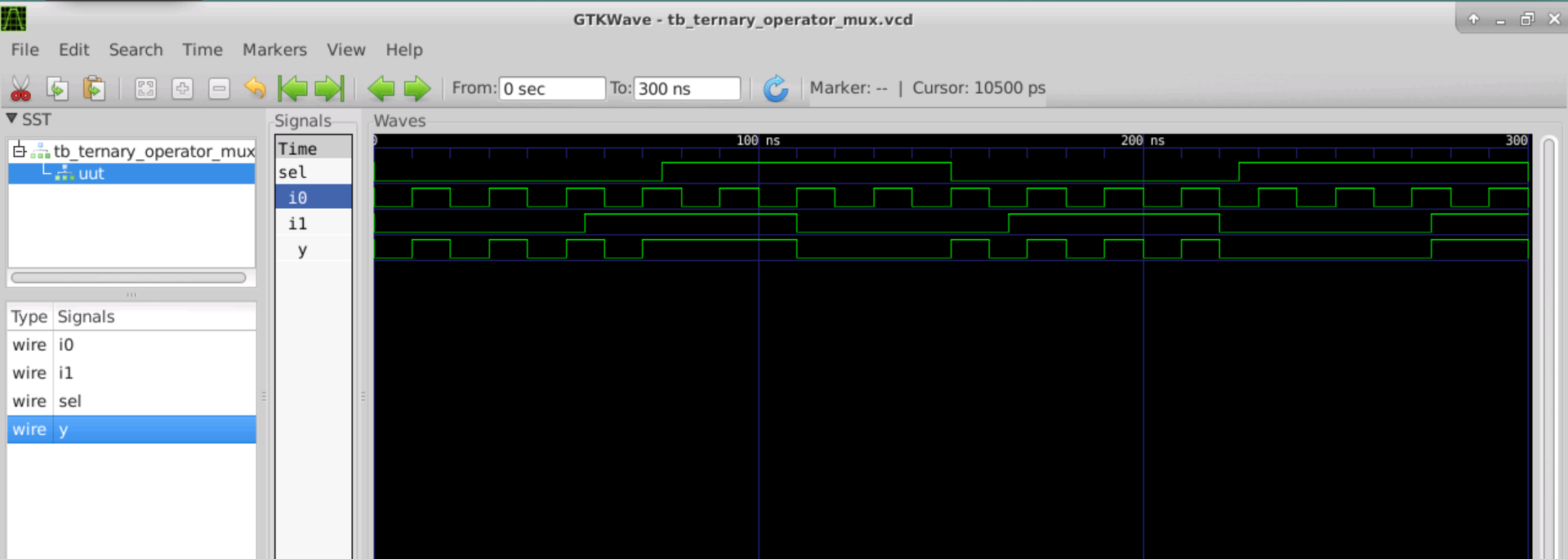
First design a MUX, the steps are as below as performed in the lab.



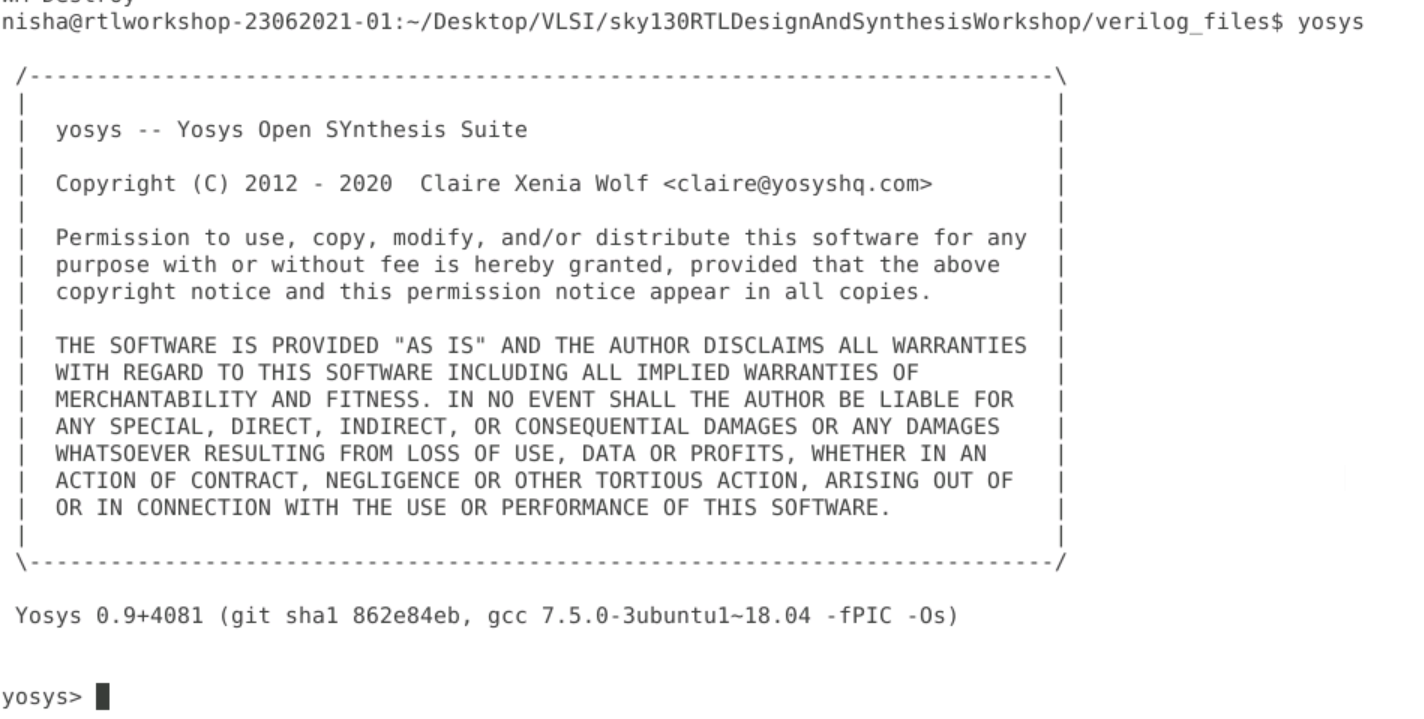


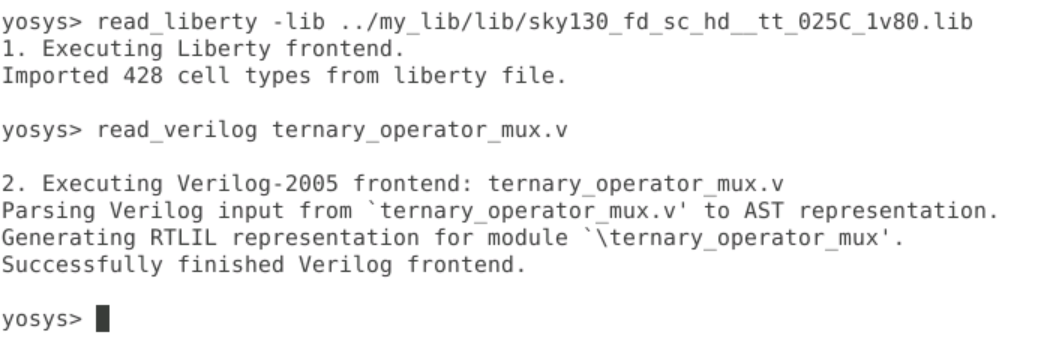


Check for the RTL simulation in GTKWave as below.

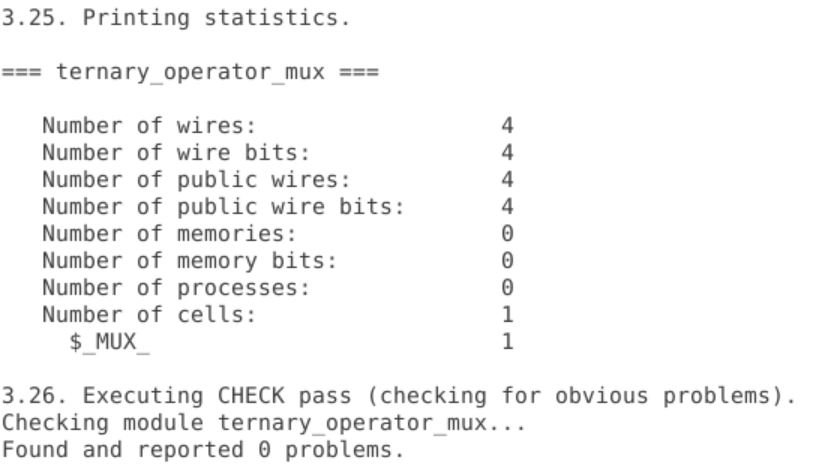


Yosys Lab steps for Synthesis:



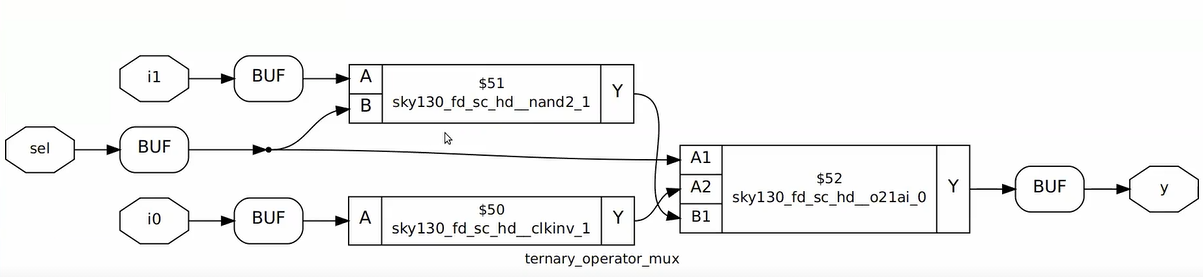




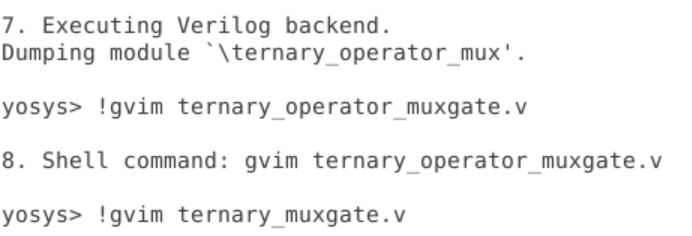




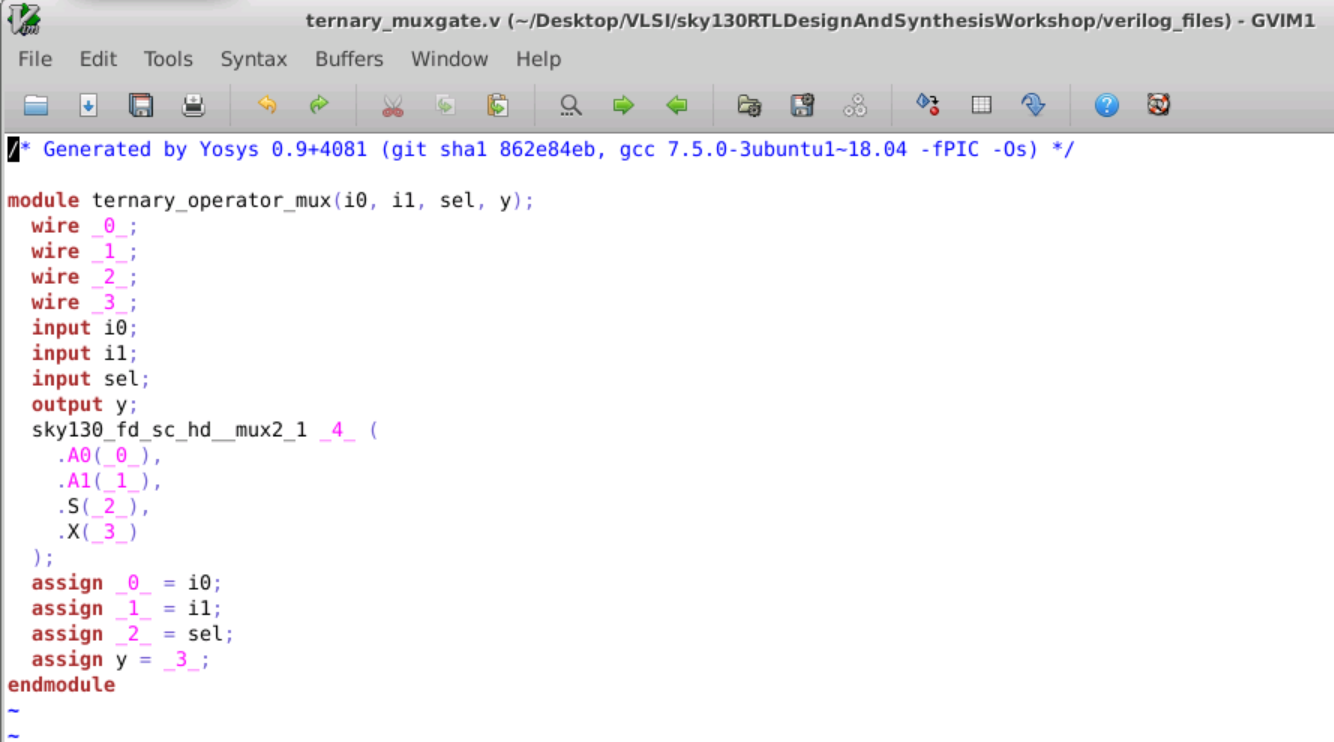
Below is the MUX design after synthsis in Yosys as expected using the gates.



The gatelevel netlist from Yosys tool can be generated as below:



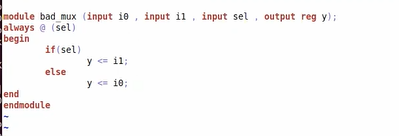
Below is the snippet of the gatelevel netlist:

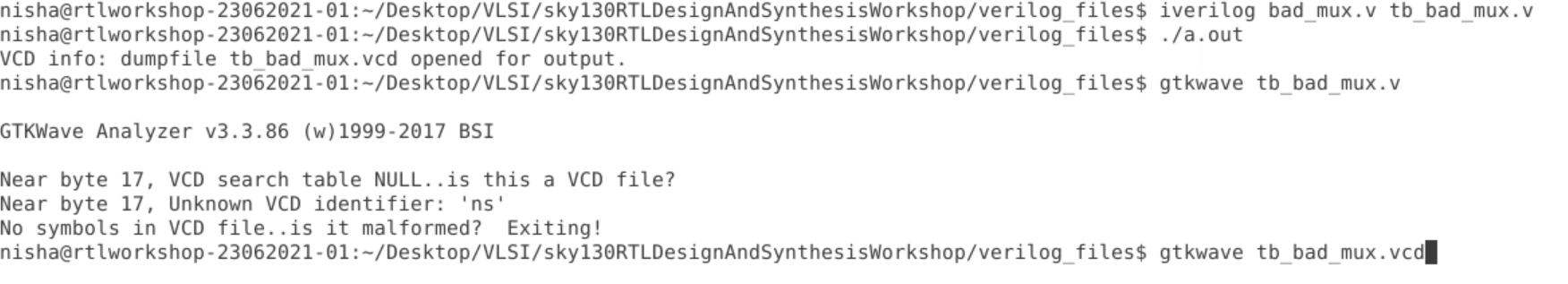


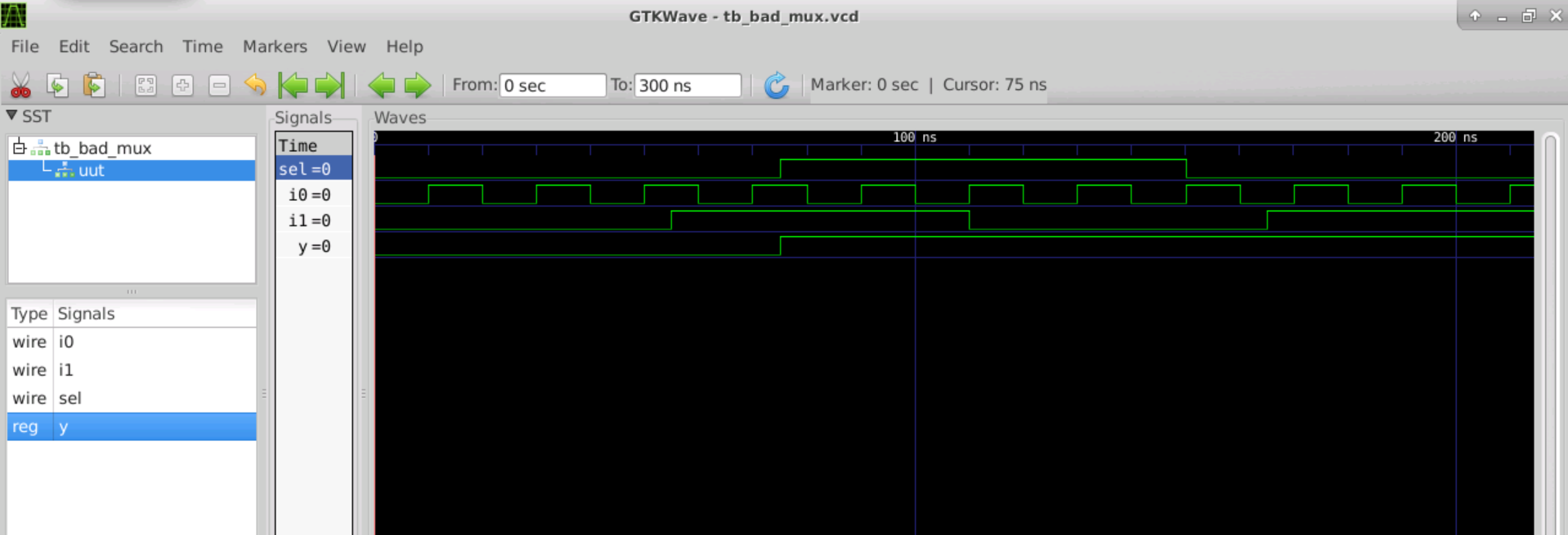
Simulation Mismatch(RTL Simulation v/s GLS):

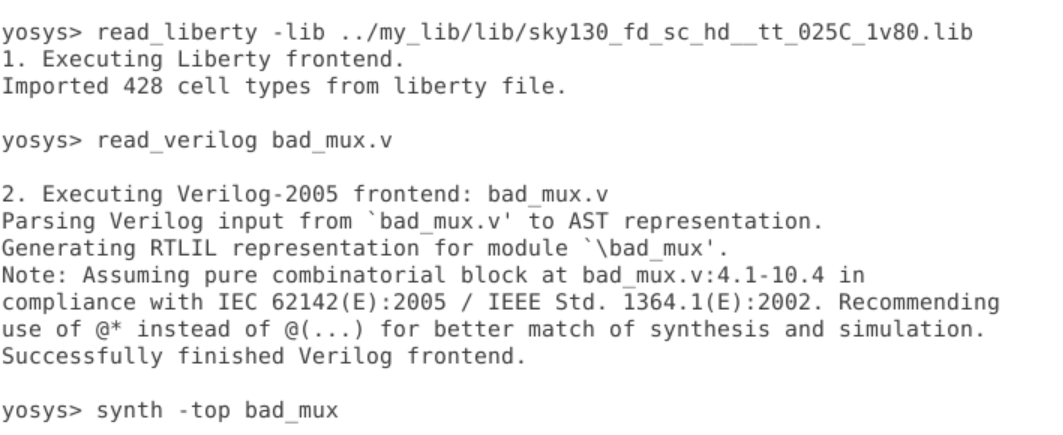
1. Missing Sensitivity List:

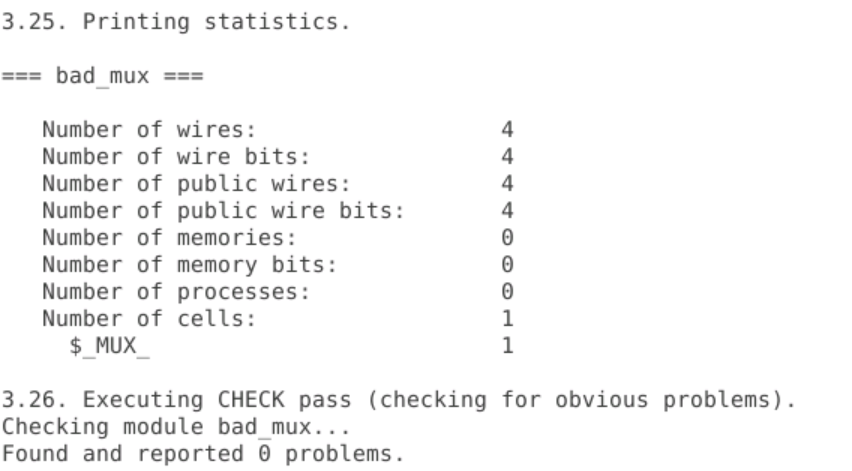
During the GLS the simulator will check for the sensitivity list, however during synthesis the tool is interested only in functionality not the sensitivity list, so basically GLS checks for the proper working of a design. Below is the example of a simulation mismatch in the RTL design and GLS for MUX due to missing sensitivity list. The design are referred as bad and good MUX as below.

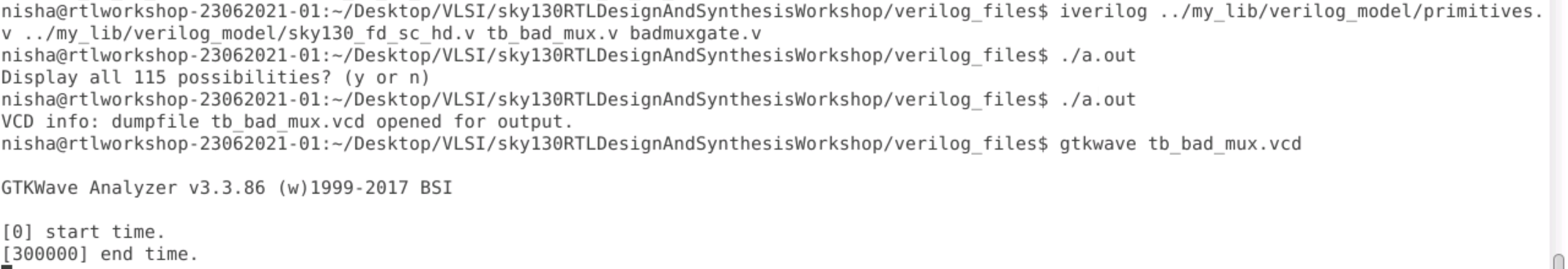


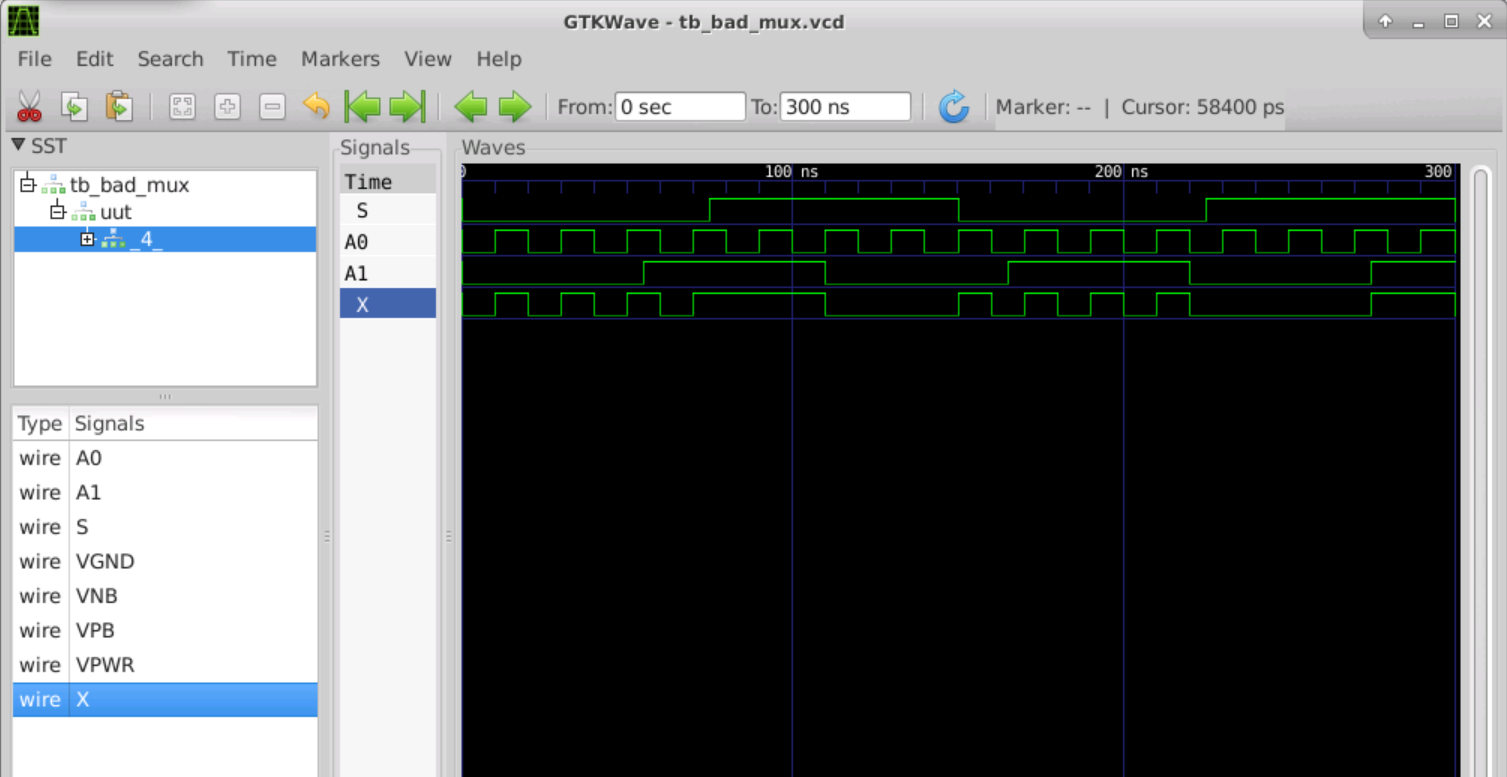






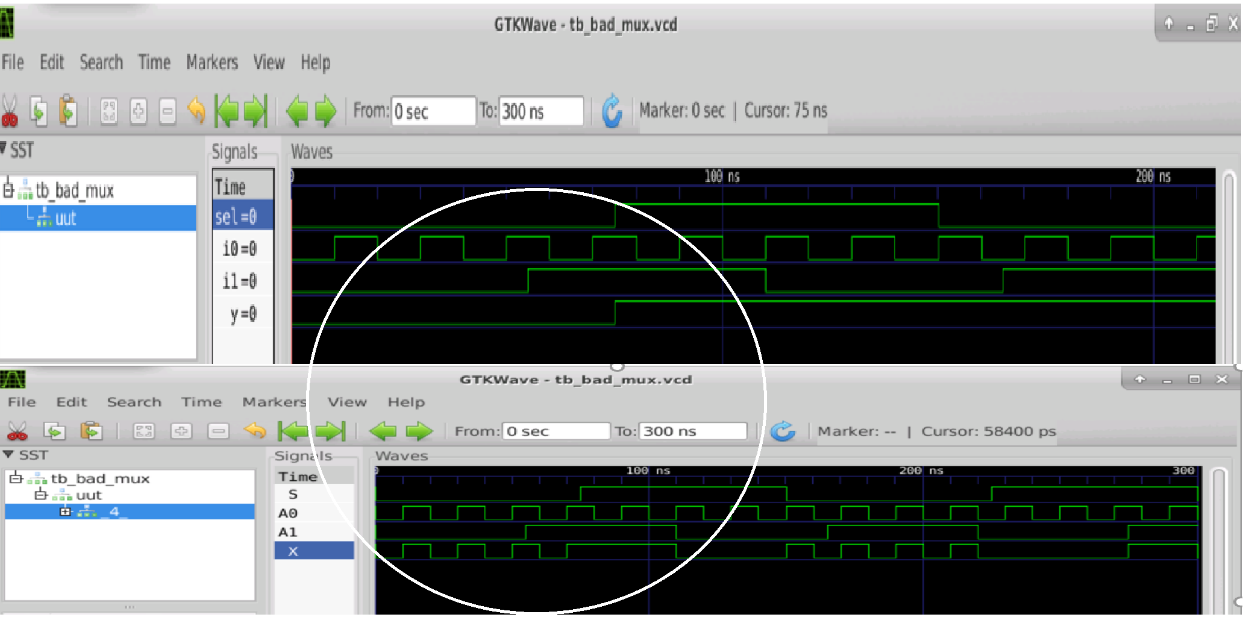






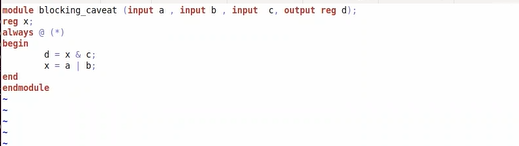
CONCLUSION:

We can clearly see the GLS and RTL simulation mismatch for bad\_mux design and the difference are shown below marked.



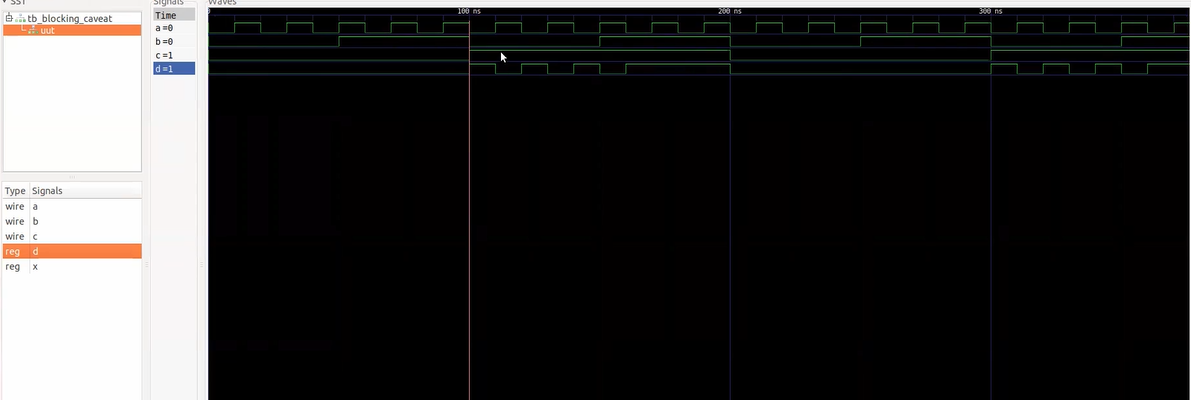
1. Blocking v/s Non-Blocking Assignment:

* Inside Always block
  + **“=”** 🡪 Blocking
    - Executes the statement in the order it is written
    - So the first statement is evaluated before the second statement
  + **“<=”** 🡪 Non Blocking
    - Executes all the RHS when always block is entered and assigns to LHS
    - Parallel evaluation

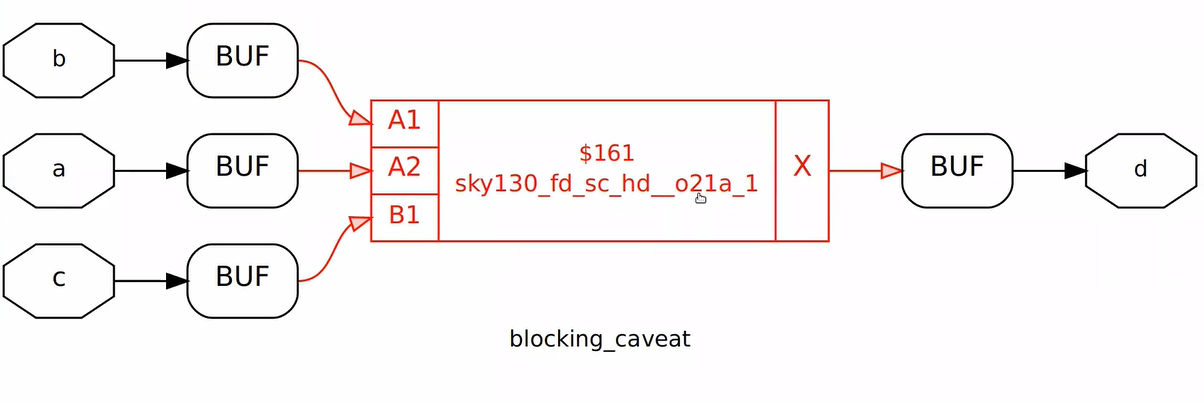


* After analyzing the above code ‘x’ looks like a flopped output

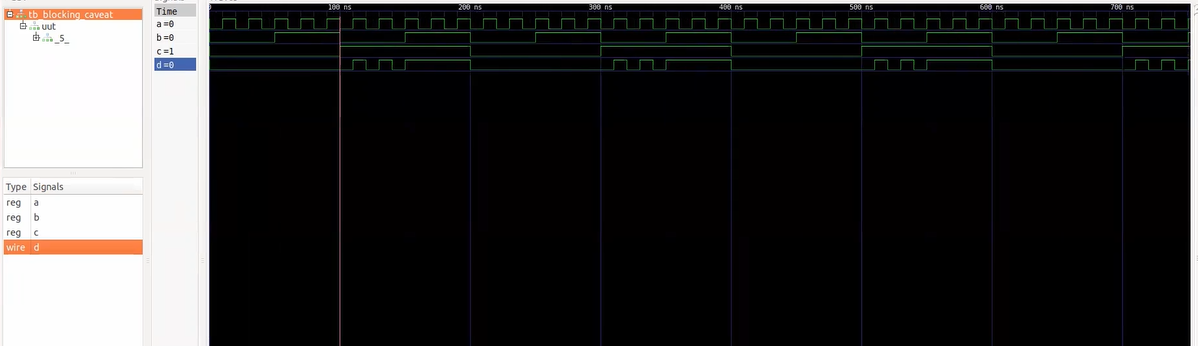
The simulation for the above design is as below and the discrepancy is marked.



Logic after synthesis is as below:-



After synthesis the GLS is as below.



CONCLUSION: -

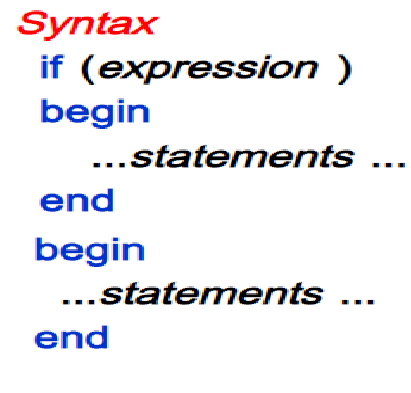
We can clearly see the GLS and RTL simulation mismatch due to blocking statement for blocking\_caveat design and the difference are shown above marked. Always be careful while you use the blocking statement

**Session 5**

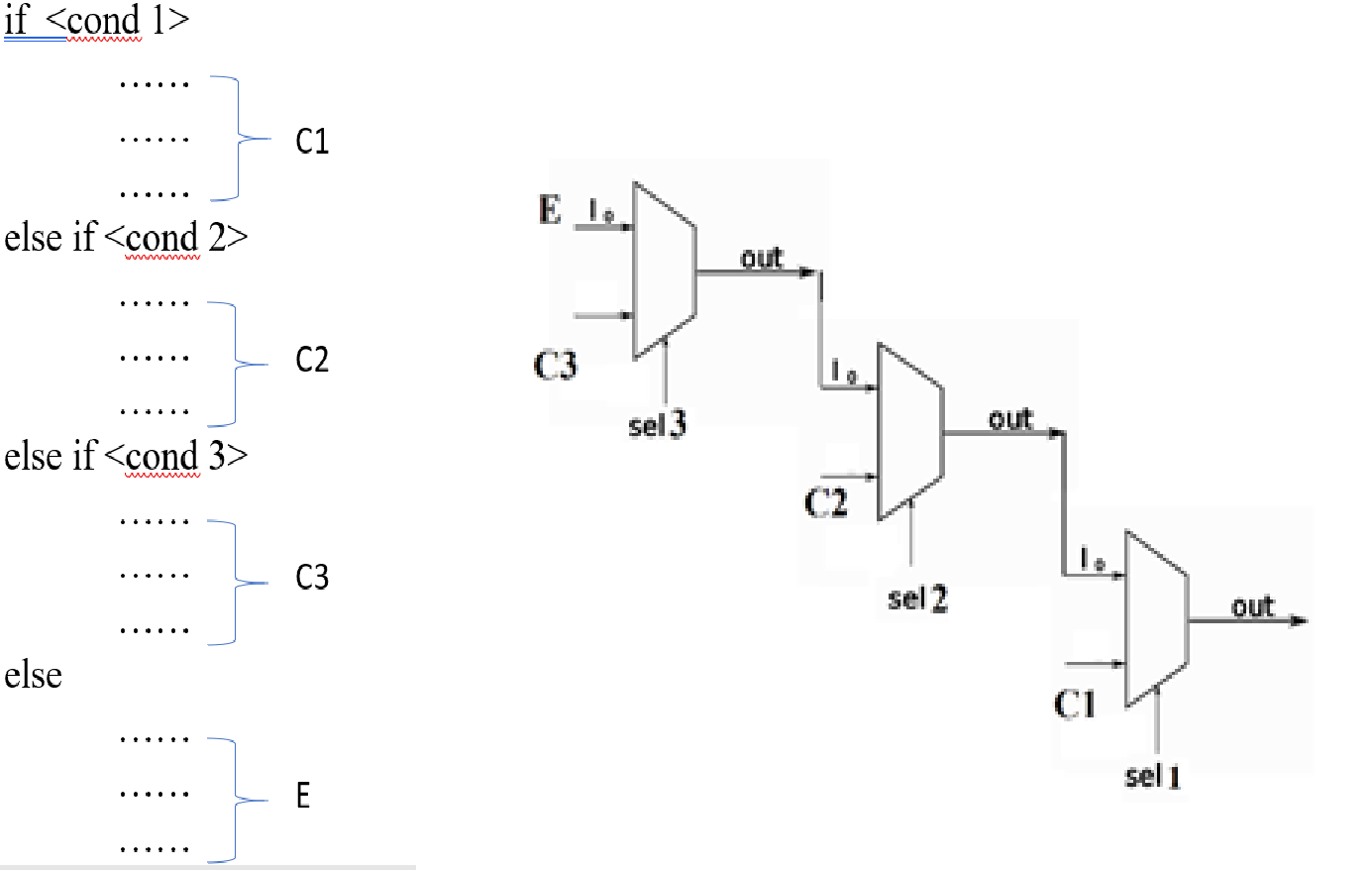
if, case, for loop and for generate

1. if, case

* “if ” is used to create priority logic.
* The syntax of “if” is as below.

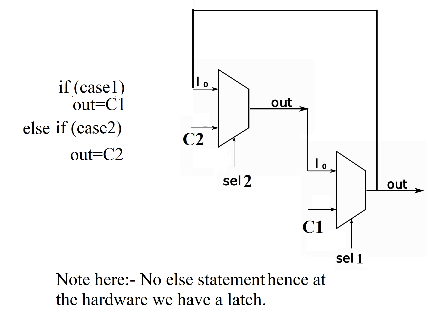


* While you use the if, case statement the hardware realized is a MUX as below.

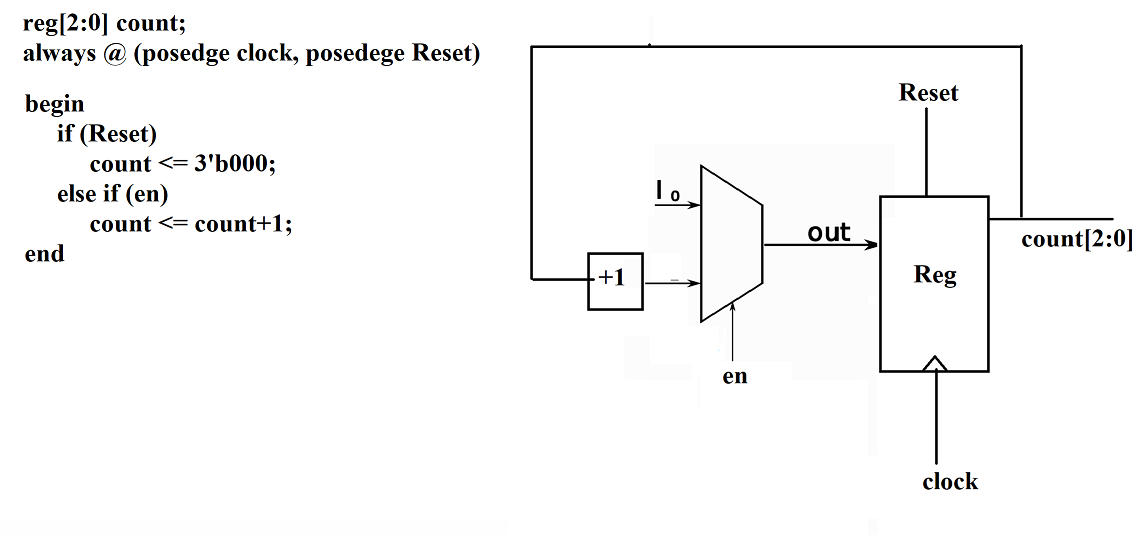


Care to be taken while you use the “if, case” statement.

* + The “if” statement should be complete, if incomplete it leads to INFERRED LATCHES. This leads to bad coding style. The example is as below.



* + Above statement cannot be generalized for counter design as below, where you use incomplete if, case statement, where if no “en” count should latch on to previous value, hence depends on the intent of the design. Always draw the design before you code.



* + If, case statement is used inside the always block
  + Code for the default statement and avoid partial assignments in “case” else it will lead to inferred latches in hardware. Assign all the outputs in all segments of case