Day-4

GLS, Blocking vs Non-blocking and Synthesis Simulation Mismatch

Gate Level Simulation (GLS)

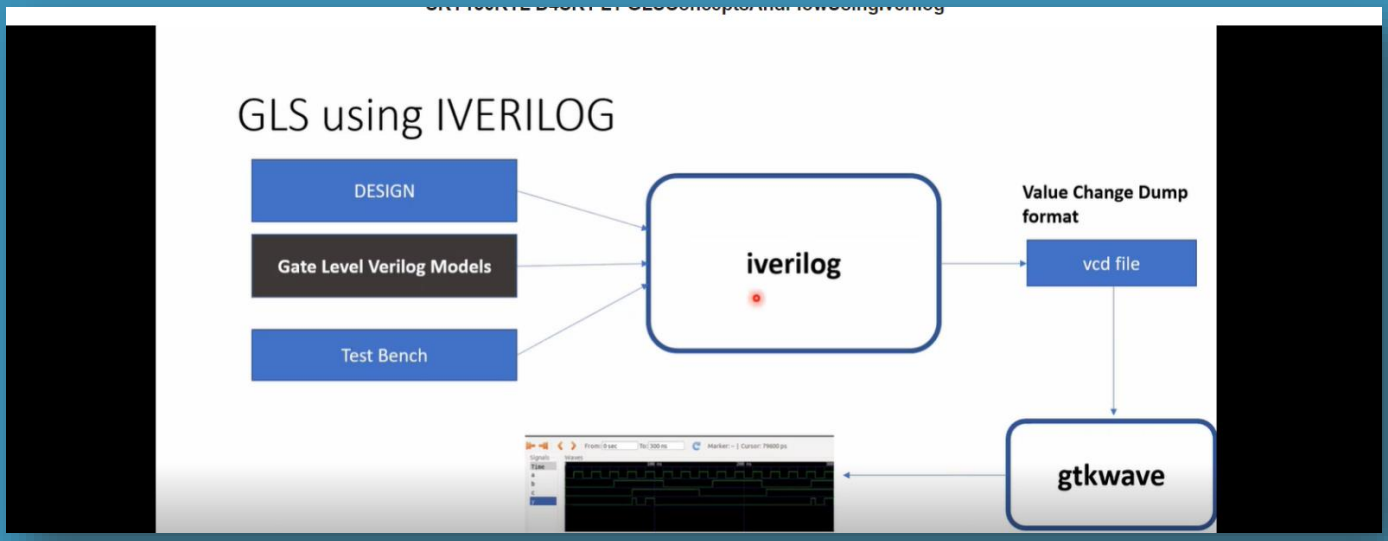
What is GLS?

Running the test bench with Netlist as Design Under Test. As Netlist is logically same as RTL Code same Test Bench will align with the design. So, we plug the Netlist in place of RTL code and run the simulation.

Why?

In order to verify the logical correctness of the design after synthesis we perform GLS. As design might not be synthesizing correct logic. GLS also ensured timing design is met for which GLS needs to be run with delay annotation.

GLS Flow



Gate Level Verilog Models contain information regarding gates in the netlist. These models can be either Timing or Functional Aware. If only Functional aware then we can validate only the functionality and if Timing aware then we can validate both Functionality and Timing of the gates.

Synthesis Simulation Mismatch

Synthesis and Simulation mismatch can occur due to the following:

- Missing Sensitivity List
- Blocking V/s Non-Blocking Assignment
  Inside always block:
  = Blocking assignment statements are executed one after the another in the order in which they are written.
  <= Non-Blocking assignment statements executes all RHS statements parallelly and assign then to the LHS
  Incorrect use of Blocking statements to create sequential logic can result in Simulation and Synthesis Mismatch, so it is always recommended to use NON-Blocking Assignment statements for Sequential Logic.

- Non-Standard Verilog Coding

Hence, we should check the behaviour of the circuit with GLS

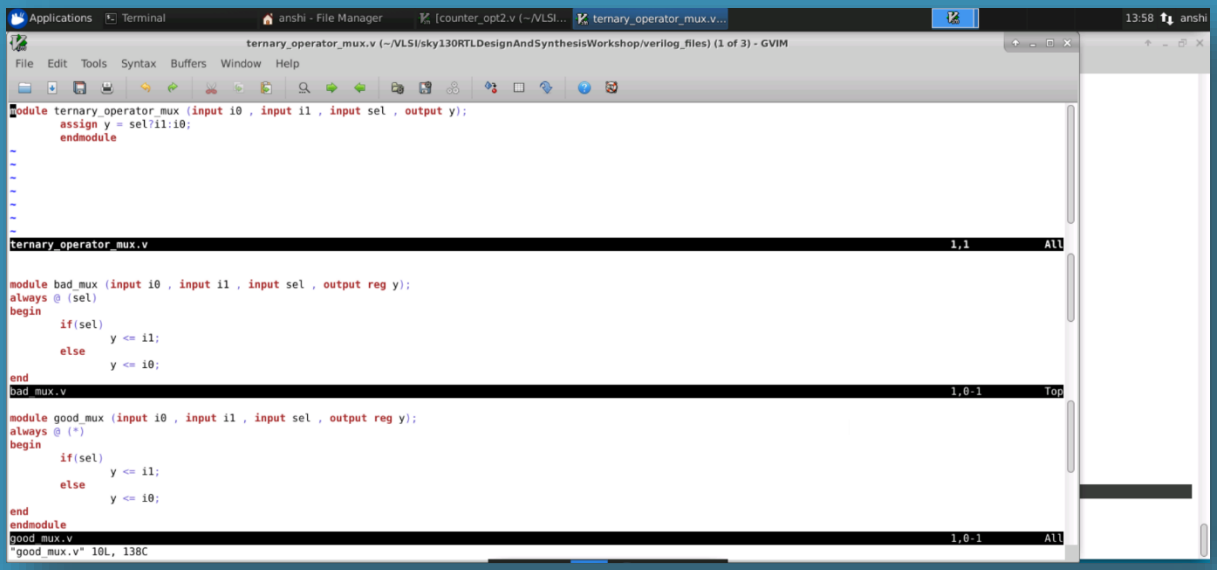
Lab Examples

1.Ternary_Operator_mux

2.Bad_mux

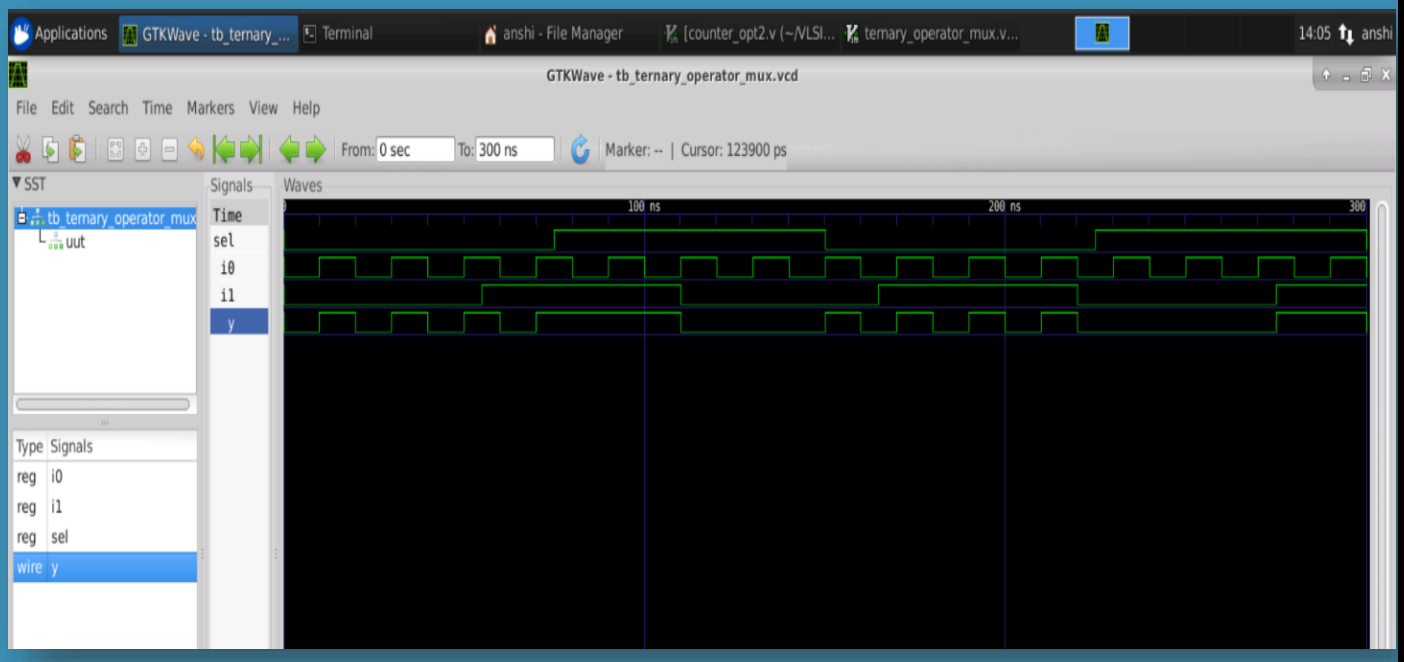
3.Good_mux

Following is the Verilog Code:



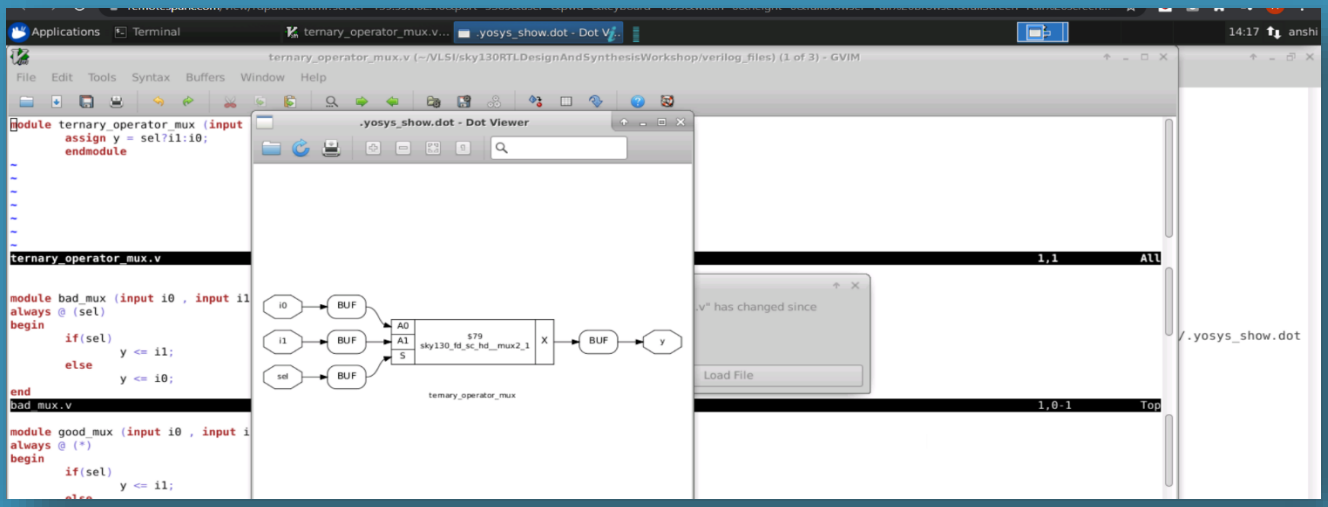
1.Ternary_Operator_mux

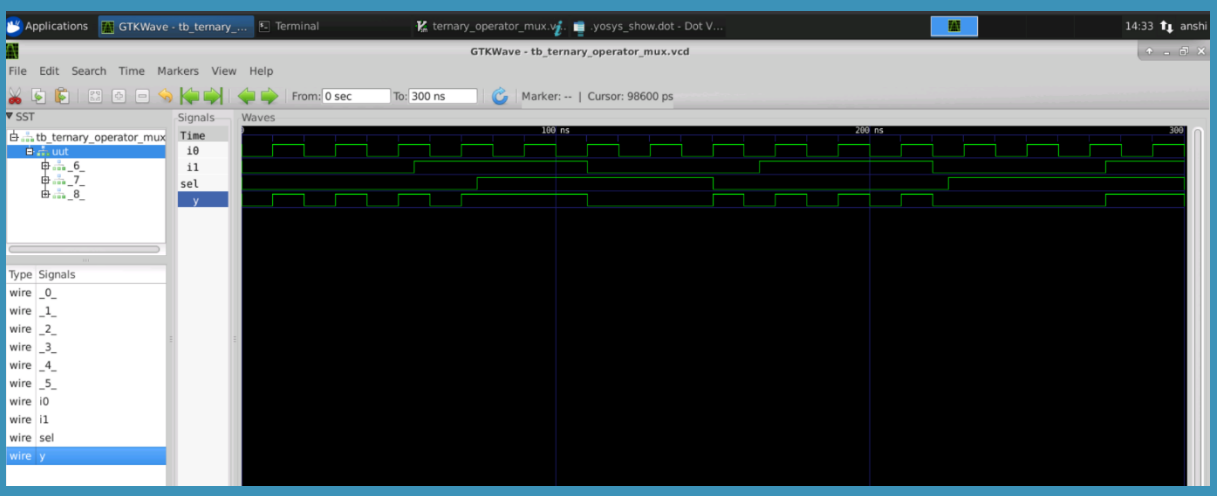
The below waveform after simulation clearly indicates the behaviour of 2:1 Mux.



```
Finding unused cells or wires in module \ternary_operator_mux..
Removed 0 unused cells and 4 unused wires.
<suppressed ~1 debug messages>

5.23.5. Finished fast OPT passes.

5.24. Executing HIERARCHY pass (managing design hierarchy).

5.24.1. Analyzing design hierarchy..
Top module:  \ternary_operator_mux

5.24.2. Analyzing design hierarchy..
Top module:  \ternary_operator_mux
Removed 0 unused modules.

5.25. Printing statistics.

=== ternary_operator_mux ===

   Number of wires:                 4
   Number of wire bits:             4
   Number of public wires:          4
   Number of public wire bits:      4
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 1
     $_MUX_                         1

5.26. Executing CHECK pass (checking for obvious problems).
Checking module ternary_operator_mux...
Found and reported 0 problems.
```

Hence, a 2:1 MUX is synthesized.

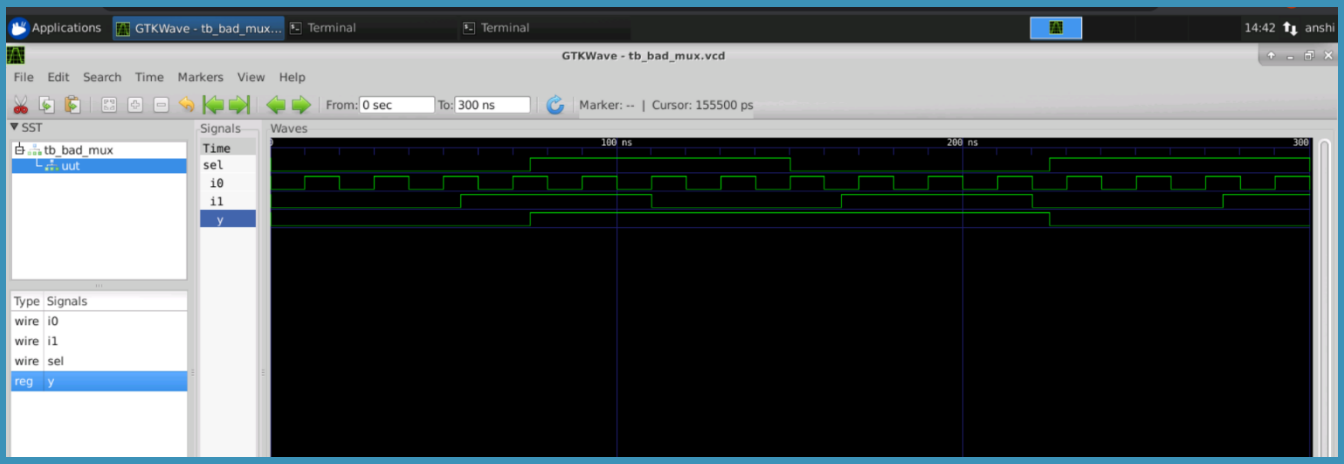




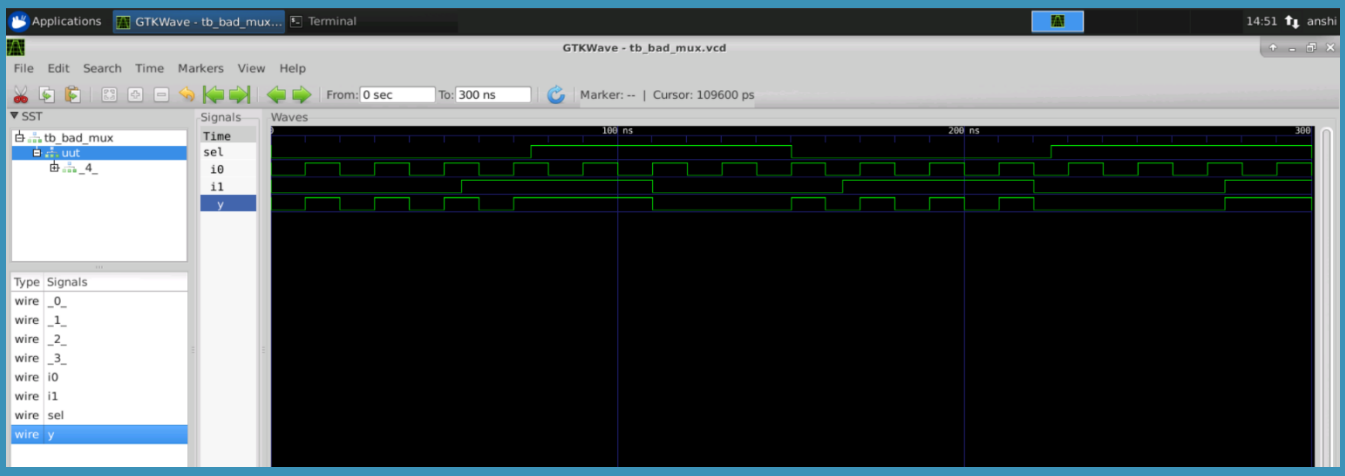
Clearly the waveform after simulation and synthesis matches.

2.Bad_mux.v

Simulation Waveform-Here y is not following the changes in i1 or i0, only changing its value at the rising edge of sel.



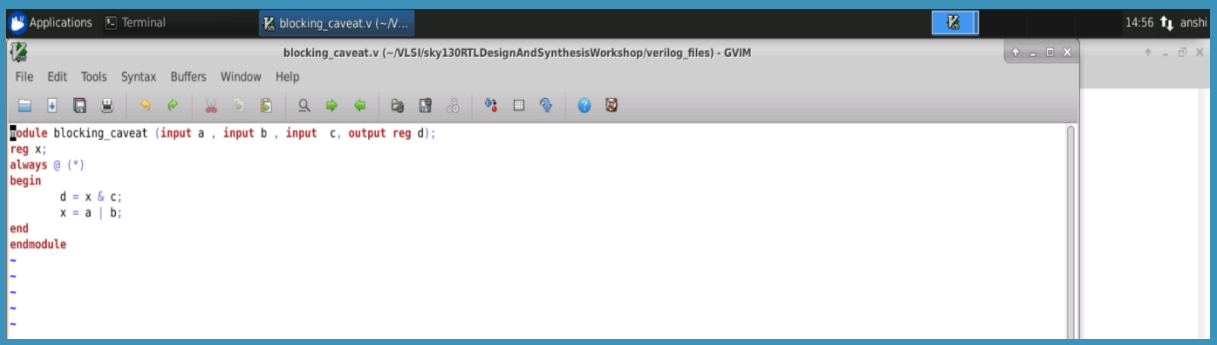
Waveform after Synthesis Using Netlist-Here y is following the changing in i1 and i0.



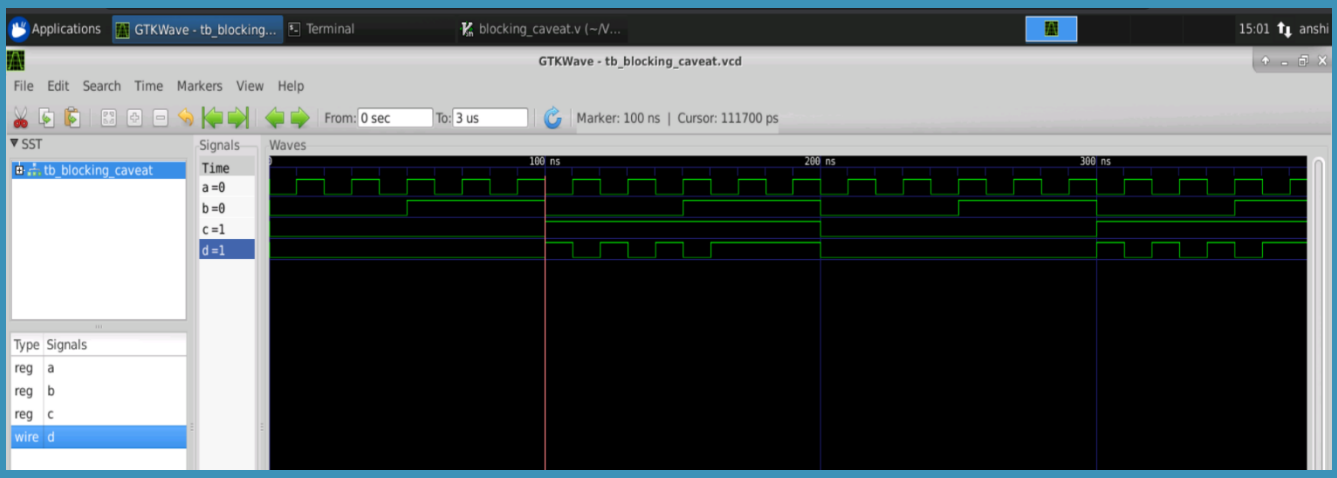
Both the waveforms obtained after simulation and synthesis are different, this clearly indicated Simulation and Synthesis Mismatch

Example- Synthesis and Simulation Mismatch because of Blocking Statements

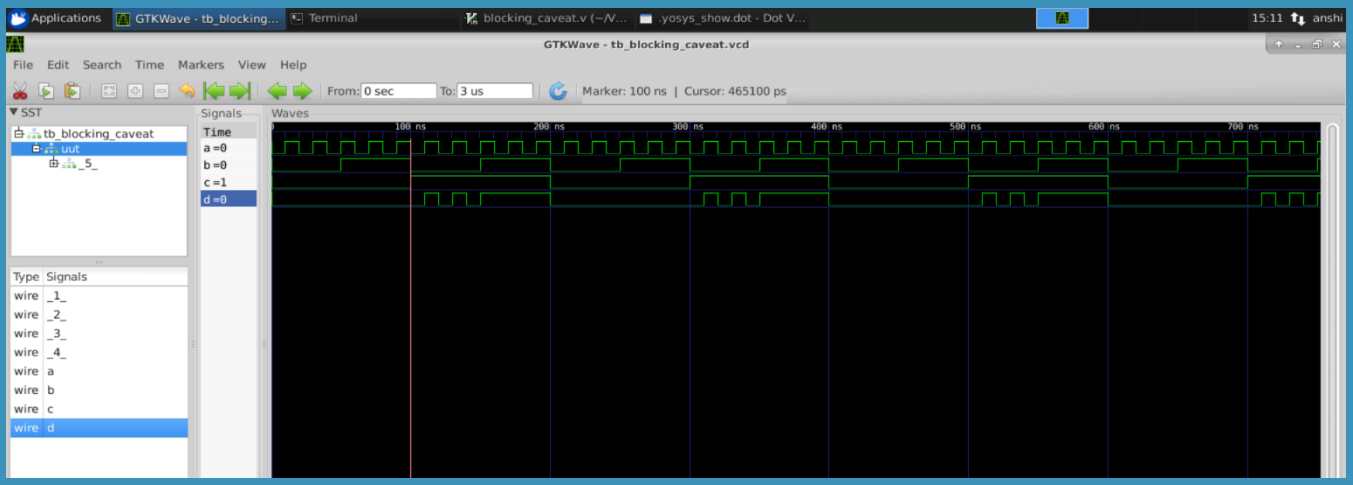
Verilog Code



At the pointer, we can see that past values a and b are considered and anded with c to give the present value of d. It seems as if a and b are flopped



After Synthesis-y is evaluating at present value od a and b.



Evaluating correct.

So, this is clearly simulation and synthesis mismatch because of blocking statements

So, it is advisable to use Non-Blocking Statements.