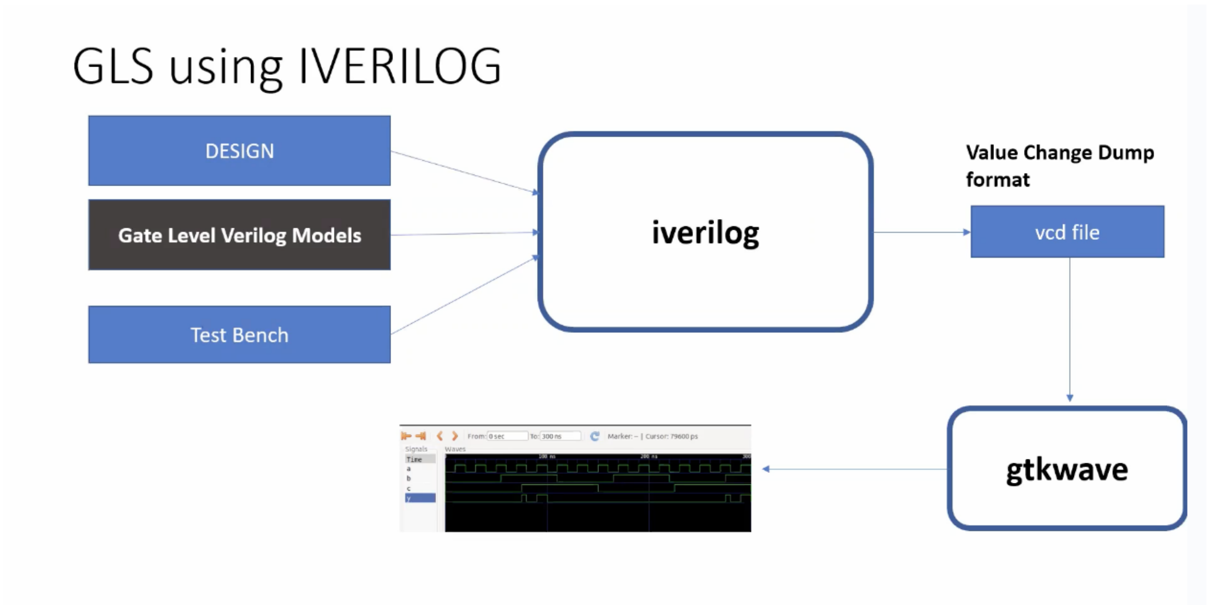
Why is Gate Level Simulation (GLS) necessary?

- Verify the correctness of the design after synthesis
- Ensure the timing of the design is met which is done with delay annotation (timing aware)



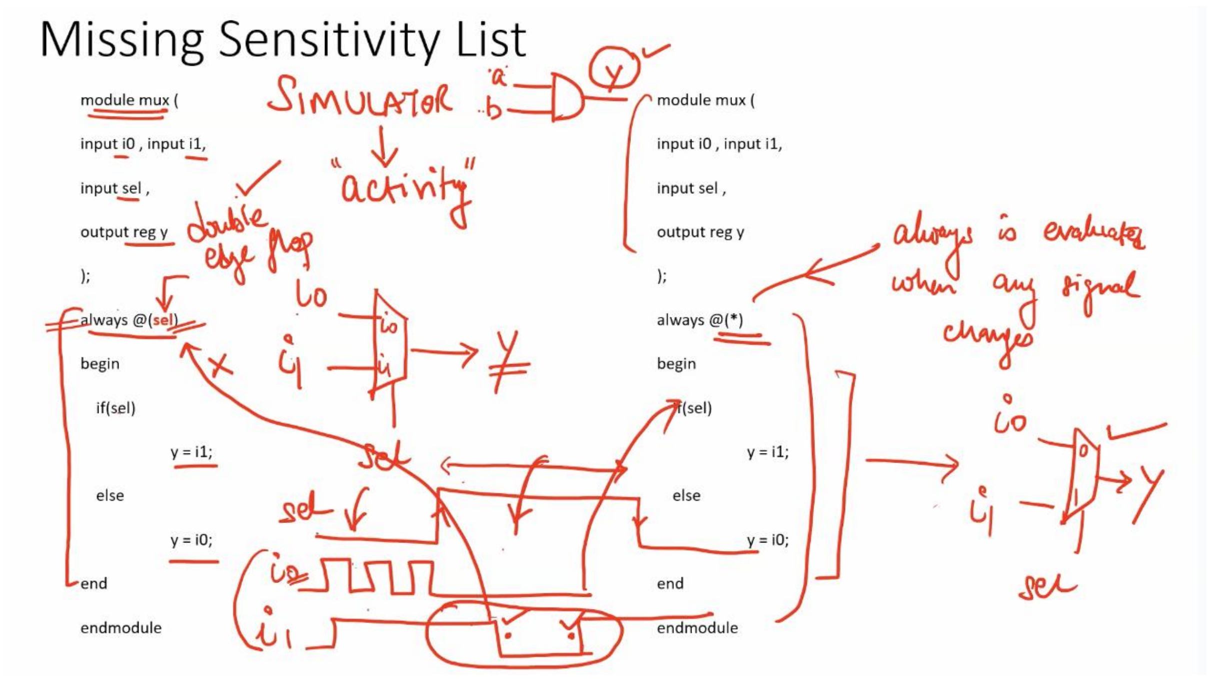
Synthesis Simulation Mismatches

It happens because of the following reasons

- Missing sensitivity list
- Blocking vs non-blocking assignments
- Non-standard verilog coding

(1) Missing sensitivity list

As shown in the screenshot below, always block is evaluated only when sel is changing. So output y is not evaluated when sel is not changing although i0 and i1 are changing. Rather it acts like a latch. The code on the right side represents the correct design coding for mux. In this case always is evaluated for any signal changes.



Missing Sensitivity List

```
module mux (
input i0 , input i1,
input sel ,
output reg y
);
always @(sel)
begin
  if(sel)
        y = i1;
  else
        y = i0;
end
endmodule
```

```
module mux (
input i0 , input i1,
input sel ,
output reg y
);
always @(*)
begin
  if(sel)
        y = i1;
  else
        y = i0;
end
endmodule
```

(2) Blocking vs Non-blocking Assignments

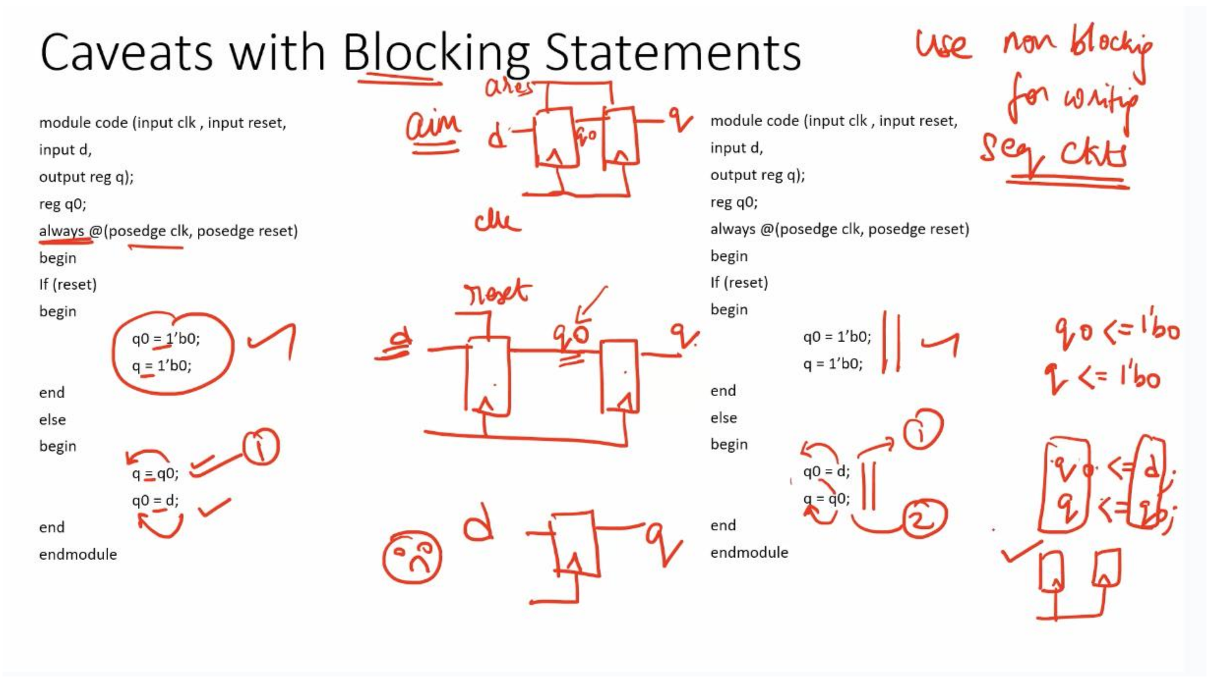
Blocking Statements

- Represented by =
- Executes the statements in the order it is written inside always block
- So the first statement is evaluated before the second statement

Non-Blocking Statements

- Represented by <=
- Executes all the RHS when always block is entered and assigns to LHS
- Parallel execution

The left side of the screenshot below gives us the correct execution. While the right side can lead to serious issues as d is assigned to q directly. ***So choosing non-blocking statements is best practice*** (highlighted in the screenshot below).



Caveats with Blocking Statements

```
module code (input clk , input reset,
input d,
output reg q);
reg q0;
always @(posedge clk, posedge reset)
begin
If (reset)
begin
        q0 = 1'b0;
        q = 1'b0;
end
else
begin
        q = q0;
        q0 = d;
end
endmodule
```

```
module code (input clk , input reset,
input d,
output reg q);
reg q0;
always @(posedge clk, posedge reset)
begin
If (reset)
begin
        q0 = 1'b0;
        q = 1'b0;
end
else
begin
        q0 = d;
        q = q0;
end
endmodule
```

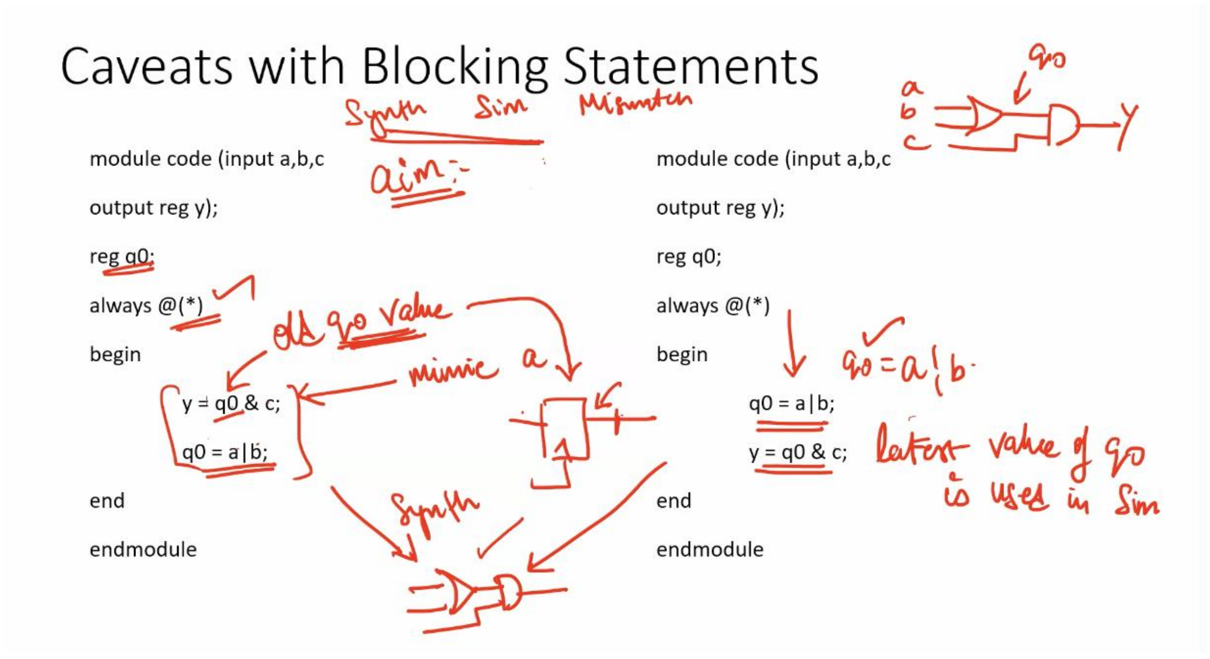
Use non blocking for writing Seq ckts
aim
d
q
q0
clk
reset
① ②
q0 <= 1'b0
q <= 1'b0
q0 <= d;
q <= q0;

Blocking Statements Leading to Synthesis Simulation Mismatch

In the code shown below, y gets the old q0 value. This will mimic delay or flop. But when you synthesize, there will be no flop. If the order is changed (right side code), latest value of q0 is assigned to y.

When synthesized, both will lead to the same circuit. However, simulation will result in different behavior. For the left side of the code, y gets the old q0 value and for the right side of the code, y gets the latest q0 value leading to a synthesis simulation mismatch.

This issue is resolved by using ***non-blocking statements***.



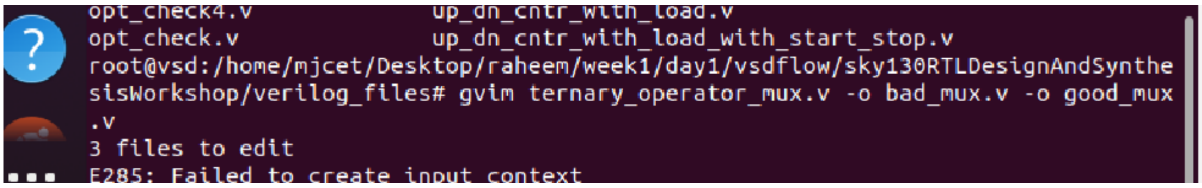
Caveats with Blocking Statements

```
module code (input a,b,c
output reg y);
reg q0;
always @(*)
begin
        y = q0 & c;
        q0 = a|b;
end
endmodule
```

```
module code (input a,b,c
output reg y);
reg q0;
always @(*)
begin
        q0 = a|b;
        y = q0 & c;
end
endmodule
```

Synth Sim Mismatch
aim:-
old q0 value
mimic a
Synth
a b c
q0
y
q0 = a|b.
latest value of q0 is used in Sim

Labs on GLS and Synthesis-Simulation Mismatch



```
opt_check4.v                up_dn_cntr_with_load.v
opt_check.v                 up_dn_cntr_with_load_with_start_stop.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# gvim ternary_operator_mux.v -o bad_mux.v -o good_mux
.v
3 files to edit
E285: Failed to create input context
```

Ternary operator MUX (ternary_operator_mux.v)

The Verilog code of ternary_operator_mux.v

module ternary_operator_mux (input i0 , input i1 , input sel , output y);

assign y = sel?i1:i0;

endmodule

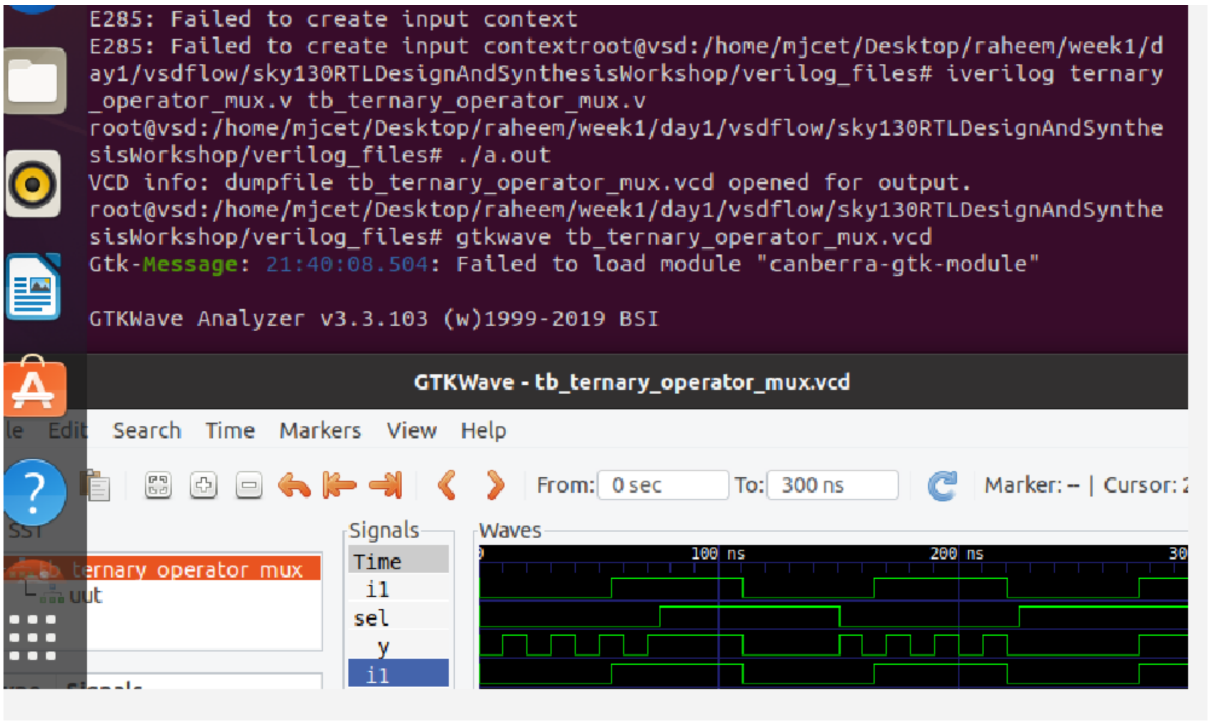
The command to run HDL simulation

iverilog ternary_operator_mux.v tb_ternary_operator_mux.v

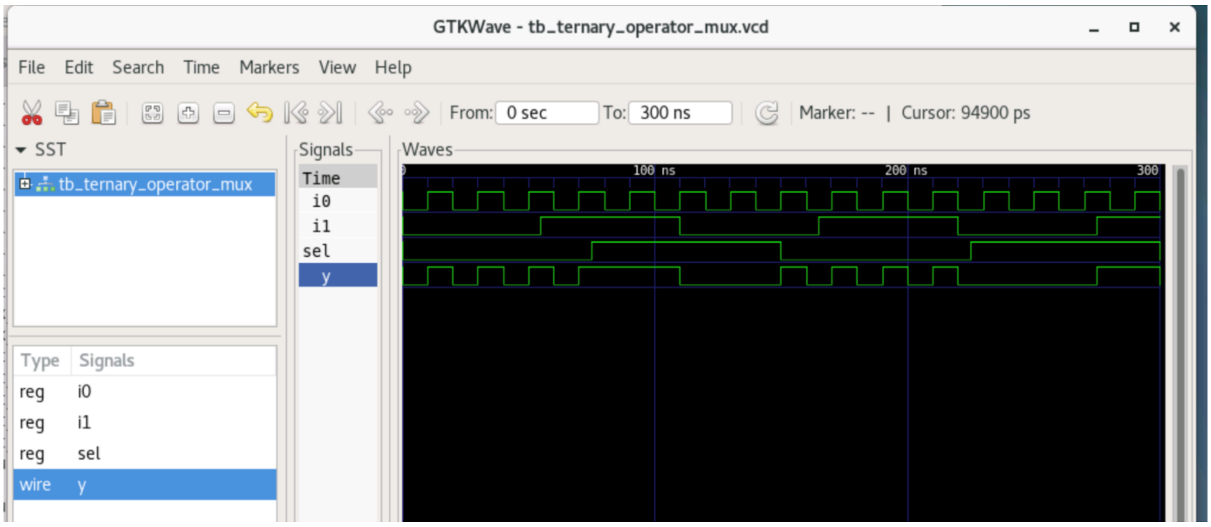
./a.out

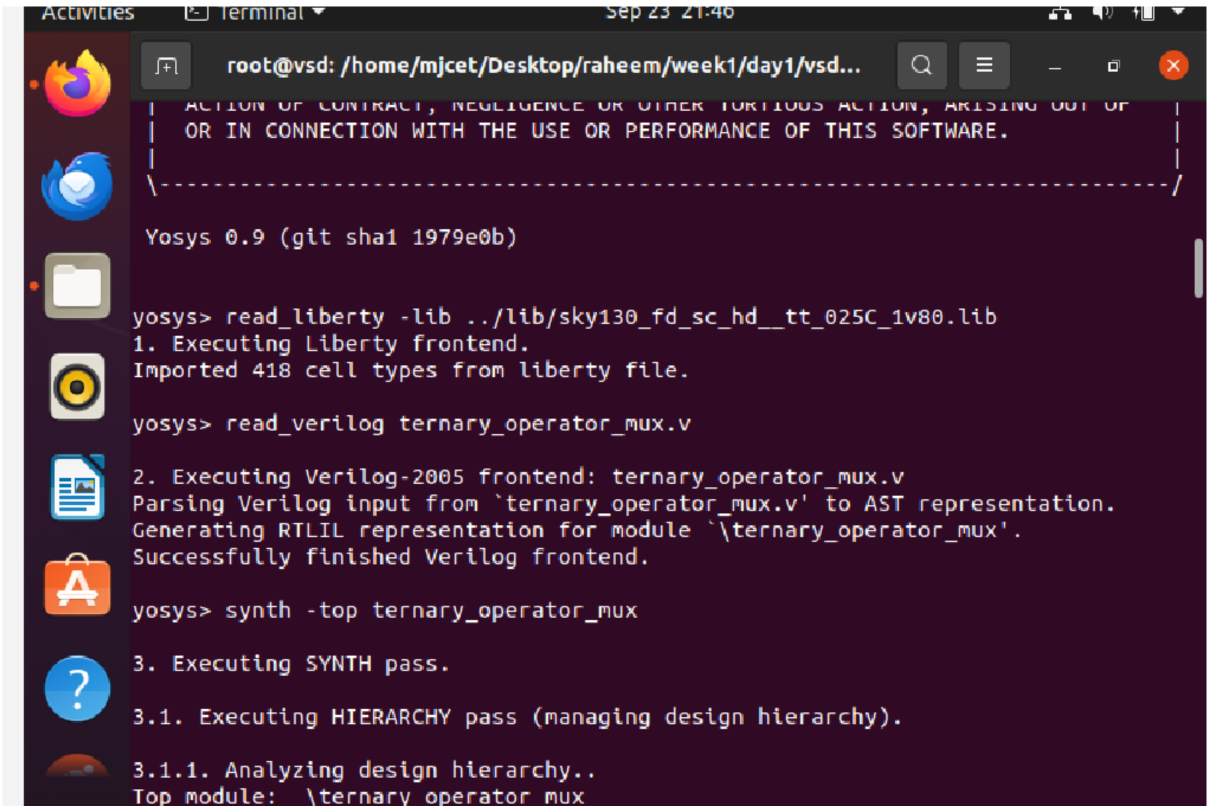
gtkwave tb_ternary_operator_mux.vcd

HDL Simulation waveform of ternary_operator_mux.v is shown in the screenshot below



```
E285: Failed to create input context
E285: Failed to create input contextroot@vsd:/home/mjcet/Desktop/raheem/week1/d
ay1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# iverilog ternary
_operator_mux.v tb_ternary_operator_mux.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# ./a.out
VCD info: dumpfile tb_ternary_operator_mux.vcd opened for output.
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# gtkwave tb_ternary_operator_mux.vcd
Gtk-Message: 21:40:08.504: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```





```
|  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF  |
|  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.           |
|                                                                            |
\----------------------------------------------------------------------------/

 Yosys 0.9 (git sha1 1979e0b)


yosys> read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib
1. Executing Liberty frontend.
Imported 418 cell types from liberty file.

yosys> read_verilog ternary_operator_mux.v

2. Executing Verilog-2005 frontend: ternary_operator_mux.v
Parsing Verilog input from `ternary_operator_mux.v' to AST representation.
Generating RTLIL representation for module `\ternary_operator_mux'.
Successfully finished Verilog frontend.

yosys> synth -top ternary_operator_mux

3. Executing SYNTH pass.

3.1. Executing HIERARCHY pass (managing design hierarchy).

3.1.1. Analyzing design hierarchy..
Top module:  \ternary_operator_mux
```

The commands to run the synthesis for ternary_operator_mux.v

read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

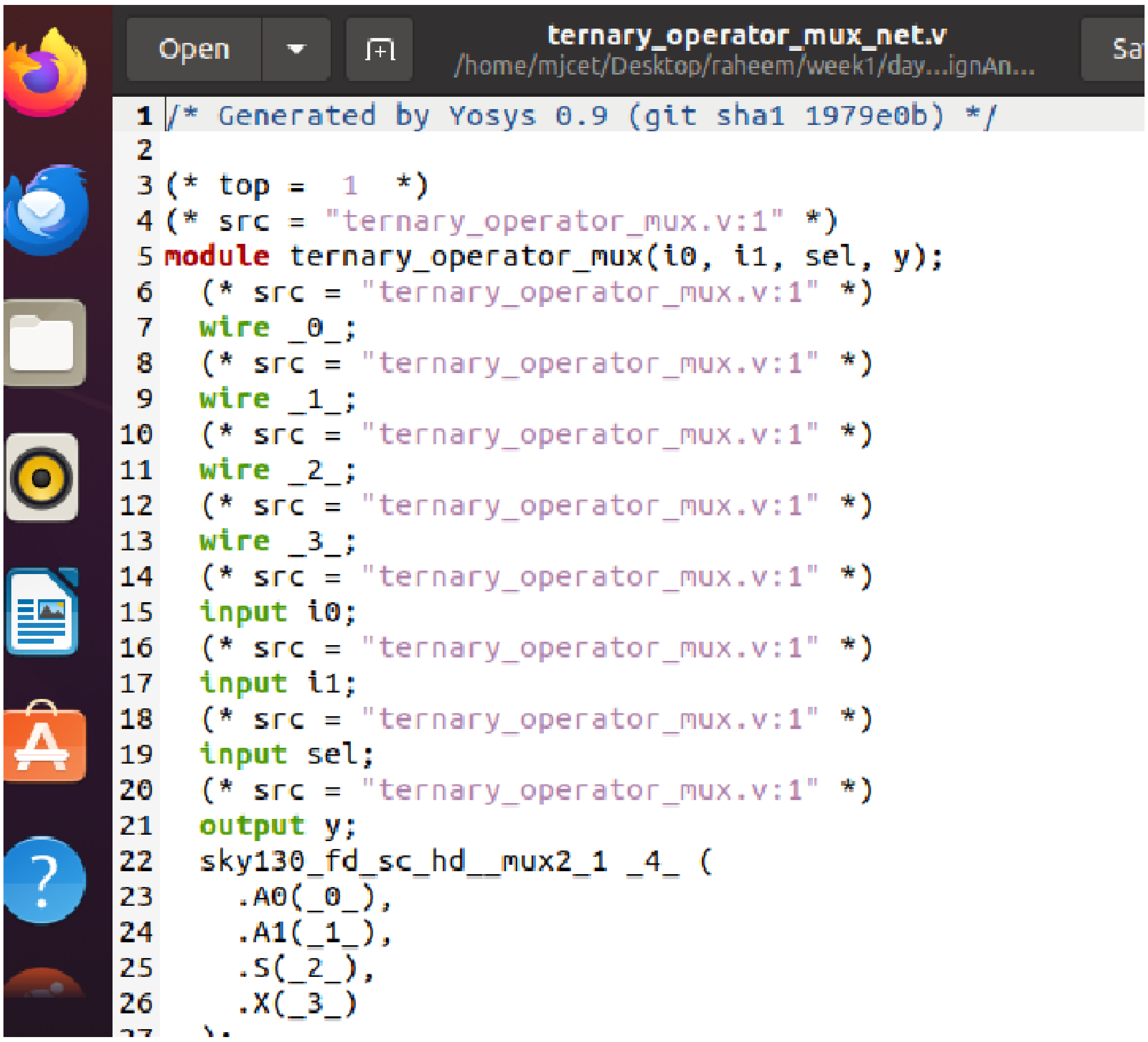
read_verilog ternary_operator_mux.v

synth -top ternary_operator_mux

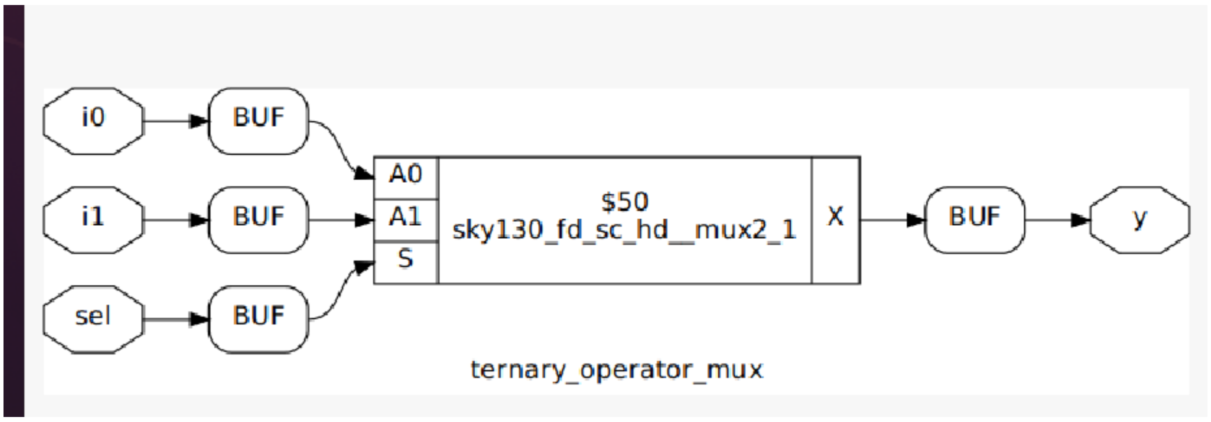
abc -liberty ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

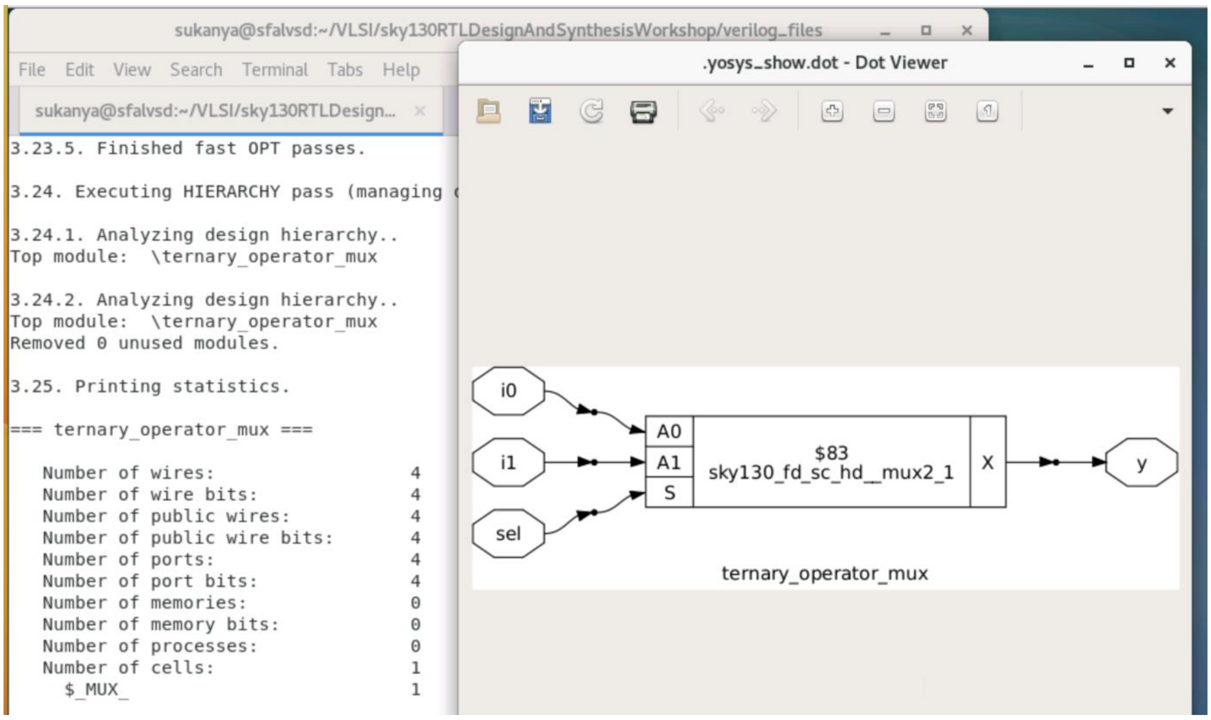
show

write_verilog ternary_operator_mux_net.v


Open
ternary_operator_mux_net.v
/home/mjcet/Desktop/raheem/week1/day...ignAn...

```
/* Generated by Yosys 0.9 (git sha1 1979e0b) */

(* top =  1  *)
(* src = "ternary_operator_mux.v:1" *)
module ternary_operator_mux(i0, i1, sel, y);
  (* src = "ternary_operator_mux.v:1" *)
  wire _0_;
  (* src = "ternary_operator_mux.v:1" *)
  wire _1_;
  (* src = "ternary_operator_mux.v:1" *)
  wire _2_;
  (* src = "ternary_operator_mux.v:1" *)
  wire _3_;
  (* src = "ternary_operator_mux.v:1" *)
  input i0;
  (* src = "ternary_operator_mux.v:1" *)
  input i1;
  (* src = "ternary_operator_mux.v:1" *)
  input sel;
  (* src = "ternary_operator_mux.v:1" *)
  output y;
  sky130_fd_sc_hd__mux2_1 _4_ (
    .A0(_0_),
    .A1(_1_),
    .S(_2_),
    .X(_3_)
```




sukanya@sfalvsd:~/VLSI/sky130RTLDesignAndSynthesisWorkshop/verilog_files
File Edit View Search Terminal Tabs Help
sukanya@sfalvsd:~/VLSI/sky130RTLDesign...
.yosys_show.dot - Dot Viewer
i0
i1
sel
A0
A1
S
$83
sky130_fd_sc_hd__mux2_1
X
y
ternary_operator_mux

```
3.23.5. Finished fast OPT passes.

3.24. Executing HIERARCHY pass (managing

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

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

3.25. 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 ports:                 4
   Number of port bits:             4
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 1
     $_MUX_                         1
```

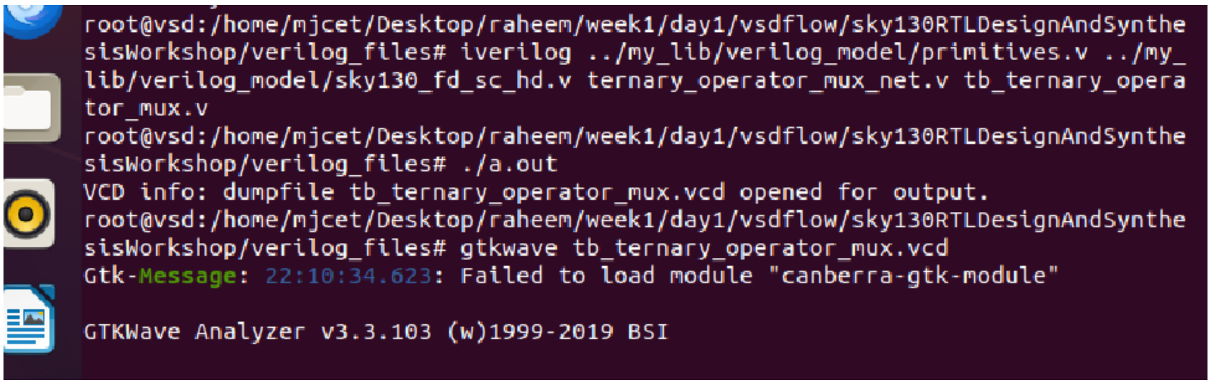
The commands to do GLS for ternary_operator_mux.v

iverilog ../my_lib/verilog_model/primitives.v ../my_lib/verilog_model/sky130_fd_sc_hd.v ternary_operator_mux_net.v tb_ternary_operator_mux.v

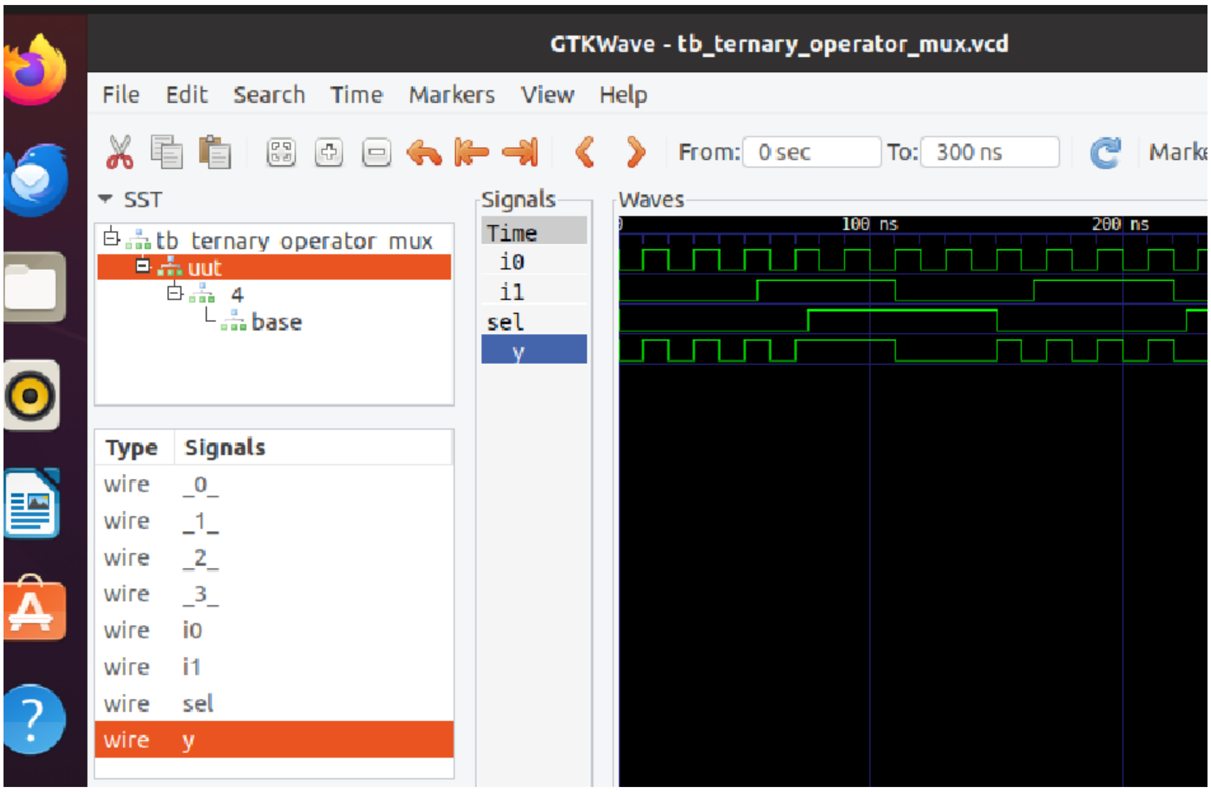
./a.out

gtkwave tb_ternary_operator_mux.vcd

The GLS output is shown



```
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# iverilog ../my_lib/verilog_model/primitives.v ../my_lib/verilog_model/sky130_fd_sc_hd.v ternary_operator_mux_net.v tb_ternary_operator_mux.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# ./a.out
VCD info: dumpfile tb_ternary_operator_mux.vcd opened for output.
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# gtkwave tb_ternary_operator_mux.vcd
Gtk-Message: 22:10:34.623: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```

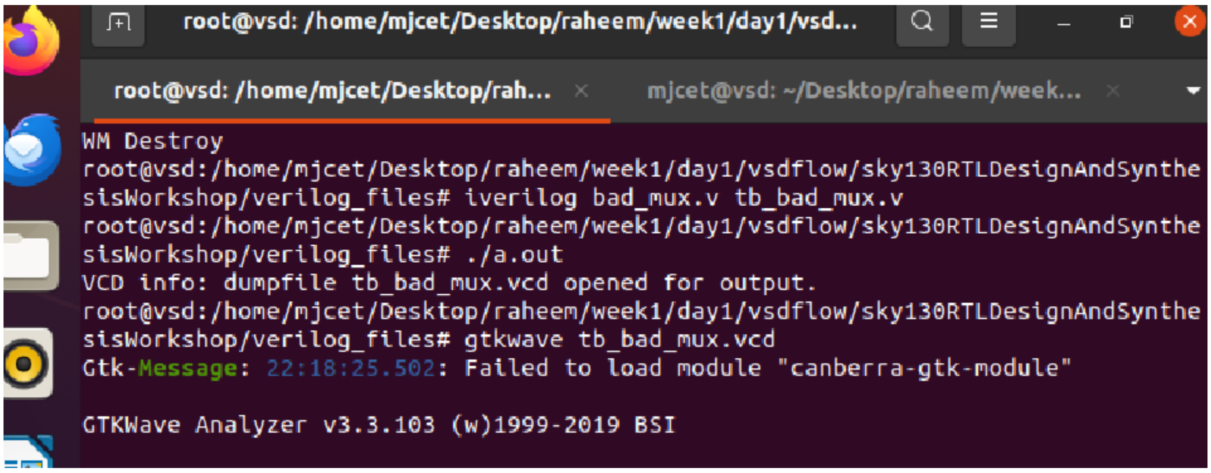


.

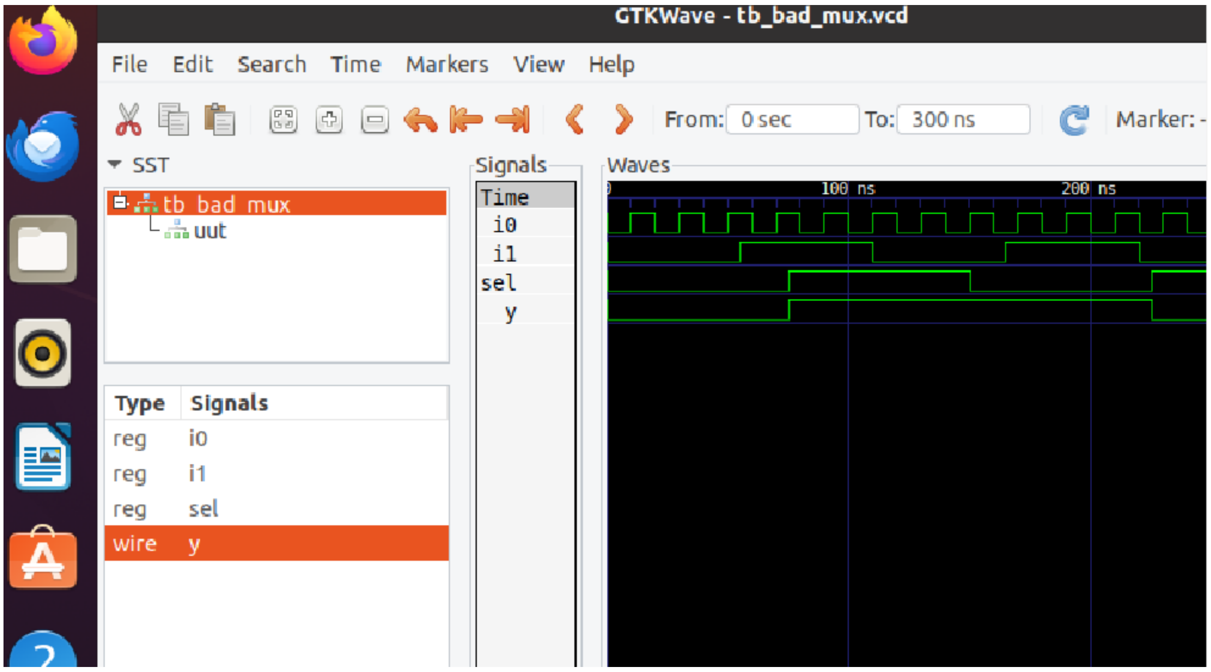
Bad MUX (bad_mux.v)

The always block is executed only at sel signal. It works like a flop rather than mux. The Verilog code of bad_mux.v

```
module bad_mux (input i0 , input i1 , input sel , output reg y);

always @ (sel)

begin

        if(sel)

                y <= i1;

        else

                y <= i0;

end

endmodule
```



```
WM Destroy
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# iverilog bad_mux.v tb_bad_mux.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# ./a.out
VCD info: dumpfile tb_bad_mux.vcd opened for output.
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# gtkwave tb_bad_mux.vcd
Gtk-Message: 22:18:25.502: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```



Type	Signals
reg	i0
reg	i1
reg	sel
wire	y

read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

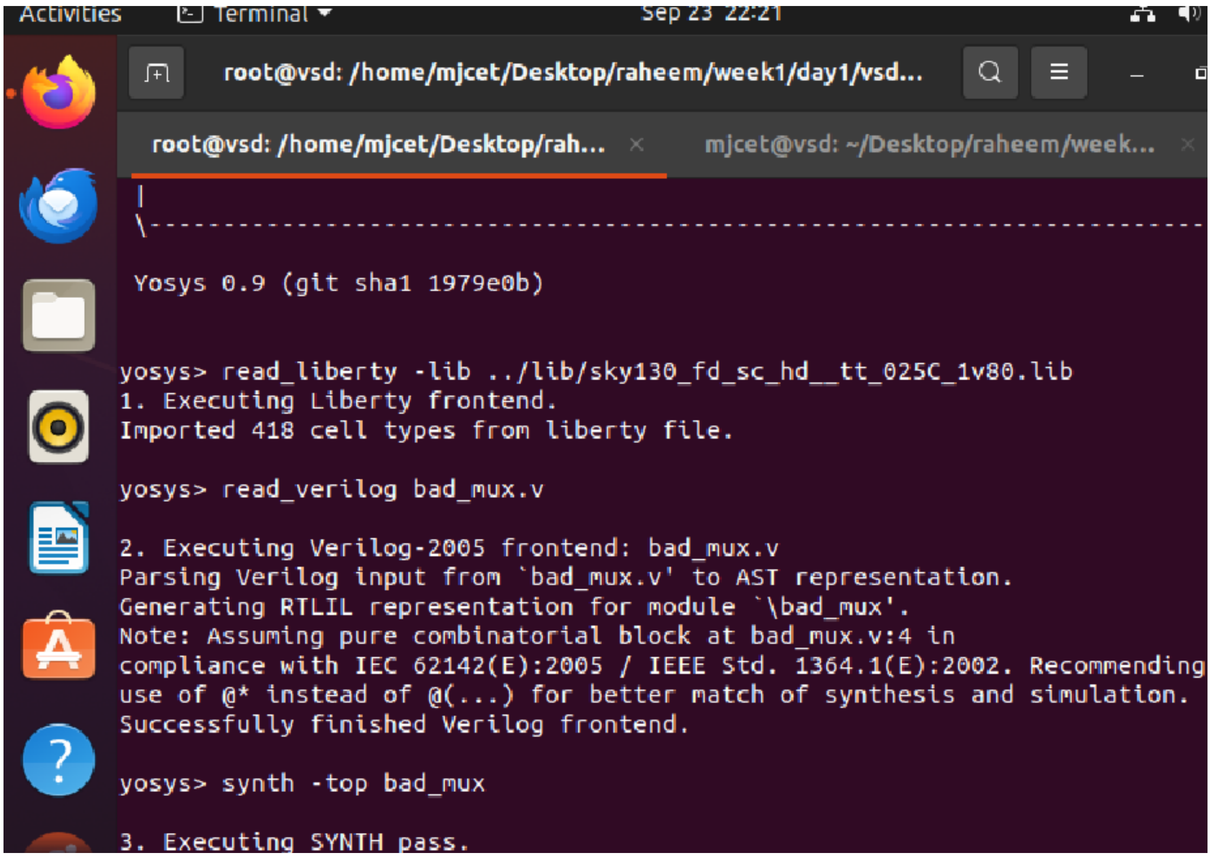
read_verilog bad_mux.v

synth -top bad_mux

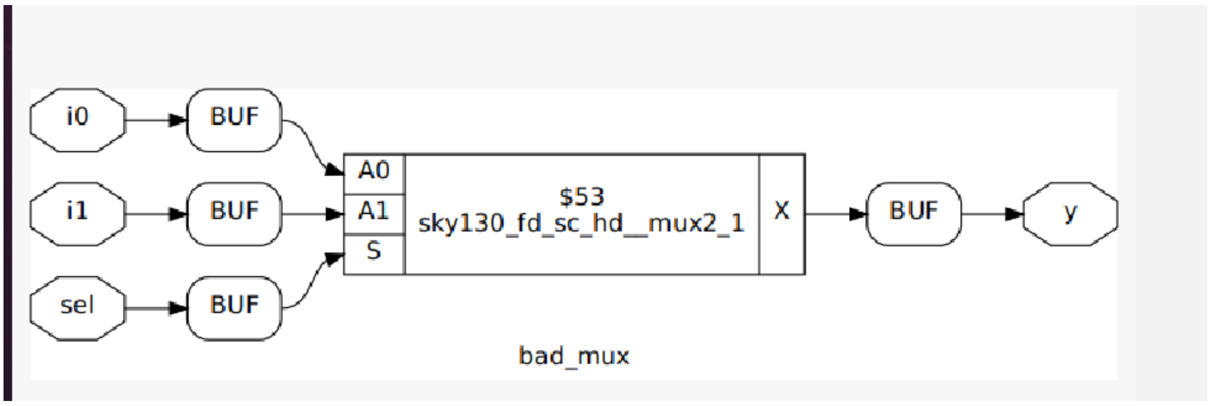
abc -liberty ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

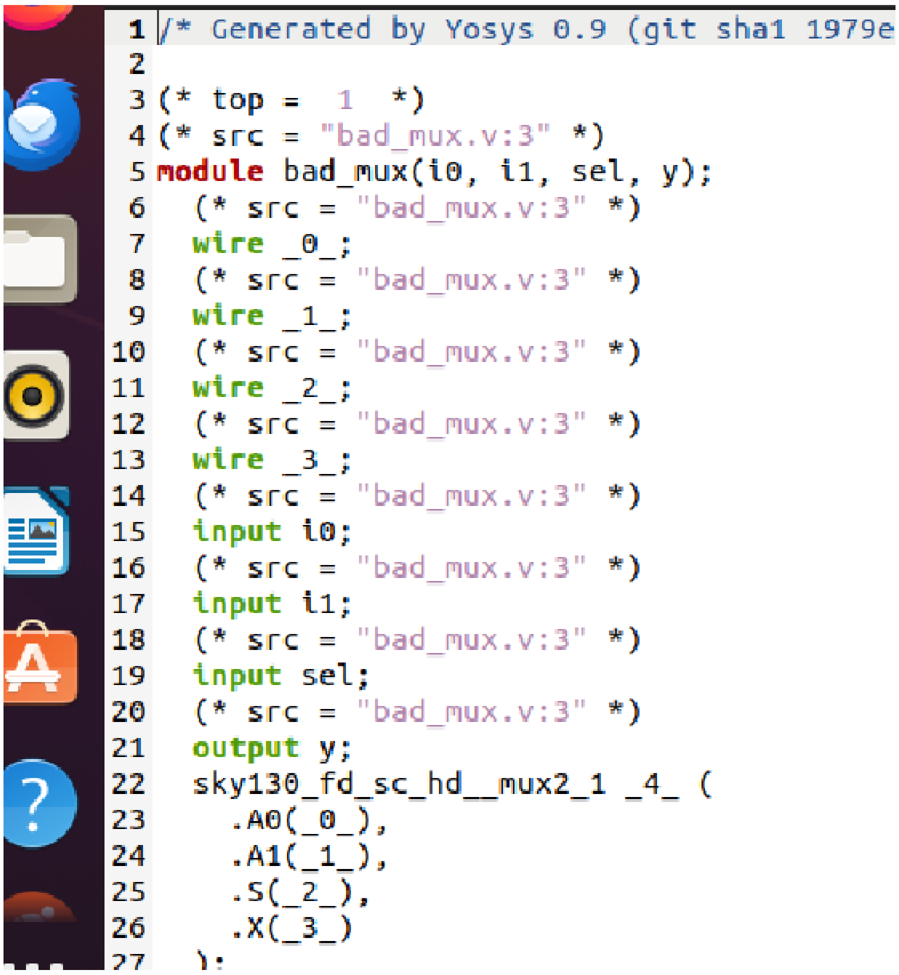
show

write_verilog bad_mux_net.v

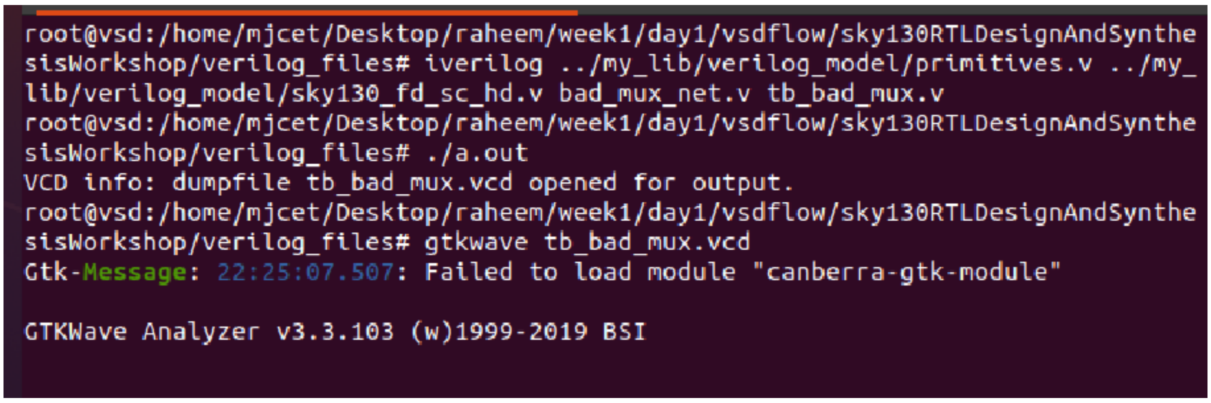

Activities
Terminal
Sep 23 22:21
root@vsd: /home/mjcet/Desktop/raheem/week1/day1/vsd...
root@vsd: /home/mjcet/Desktop/rah...
mjcet@vsd: ~/Desktop/raheem/week...

```
 |
 \------------------------------------------------------------------------

 Yosys 0.9 (git sha1 1979e0b)


yosys> read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib
1. Executing Liberty frontend.
Imported 418 cell types from liberty file.

yosys> read_verilog bad_mux.v

2. Executing Verilog-2005 frontend: bad_mux.v
Parsing Verilog input from `bad_mux.v' to AST representation.
Generating RTLIL representation for module `\bad_mux'.
Note: Assuming pure combinatorial block at bad_mux.v:4 in
compliance with IEC 62142(E):2005 / IEEE Std. 1364.1(E):2002. Recommending
use of @* instead of @(...) for better match of synthesis and simulation.
Successfully finished Verilog frontend.

yosys> synth -top bad_mux

3. Executing SYNTH pass.
```





```
/* Generated by Yosys 0.9 (git sha1 1979e

(* top =  1  *)
(* src = "bad_mux.v:3" *)
module bad_mux(i0, i1, sel, y);
  (* src = "bad_mux.v:3" *)
  wire _0_;
  (* src = "bad_mux.v:3" *)
  wire _1_;
  (* src = "bad_mux.v:3" *)
  wire _2_;
  (* src = "bad_mux.v:3" *)
  wire _3_;
  (* src = "bad_mux.v:3" *)
  input i0;
  (* src = "bad_mux.v:3" *)
  input i1;
  (* src = "bad_mux.v:3" *)
  input sel;
  (* src = "bad_mux.v:3" *)
  output y;
  sky130_fd_sc_hd__mux2_1 _4_ (
    .A0(_0_),
    .A1(_1_),
    .S(_2_),
    .X(_3_)
  );
```

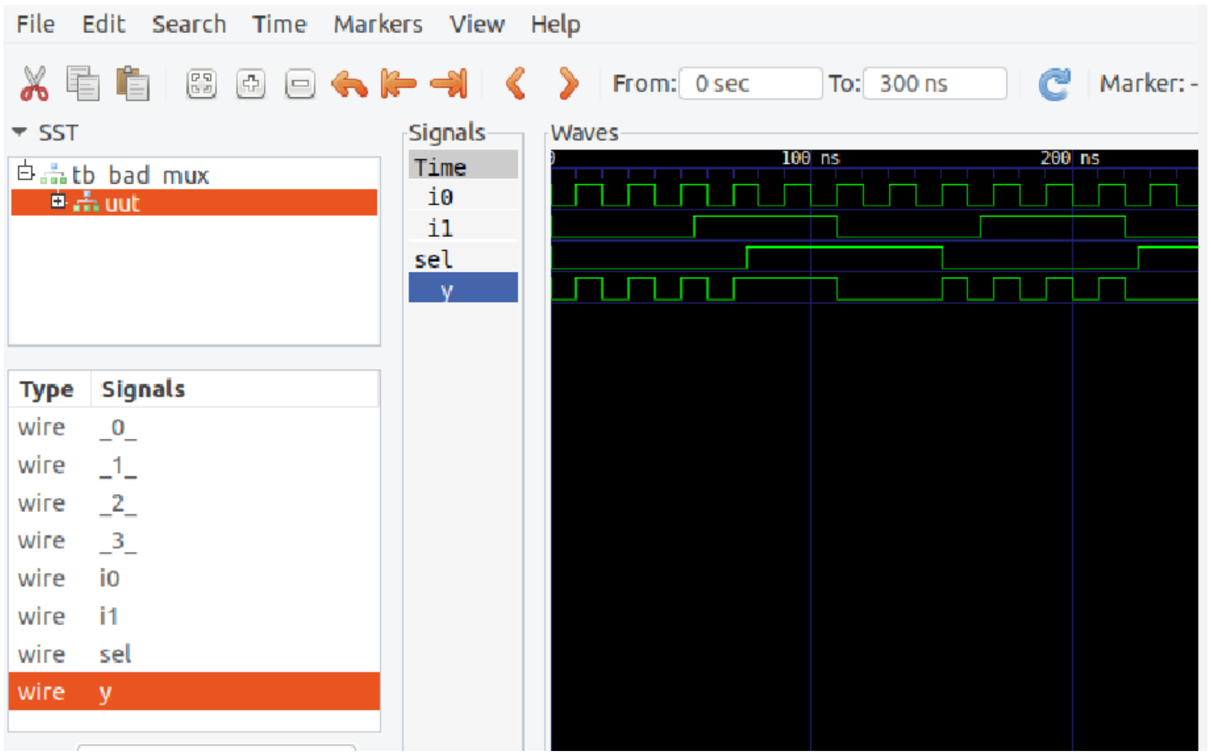


```
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# iverilog ../my_lib/verilog_model/primitives.v ../my_
lib/verilog_model/sky130_fd_sc_hd.v bad_mux_net.v tb_bad_mux.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# ./a.out
VCD info: dumpfile tb_bad_mux.vcd opened for output.
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# gtkwave tb_bad_mux.vcd
Gtk-Message: 22:25:07.507: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```

iverilog ../my_lib/verilog_model/primitives.v ../my_lib/verilog_model/sky130_fd_sc_hd.v bad_mux_net.v tb_bad_mux.v

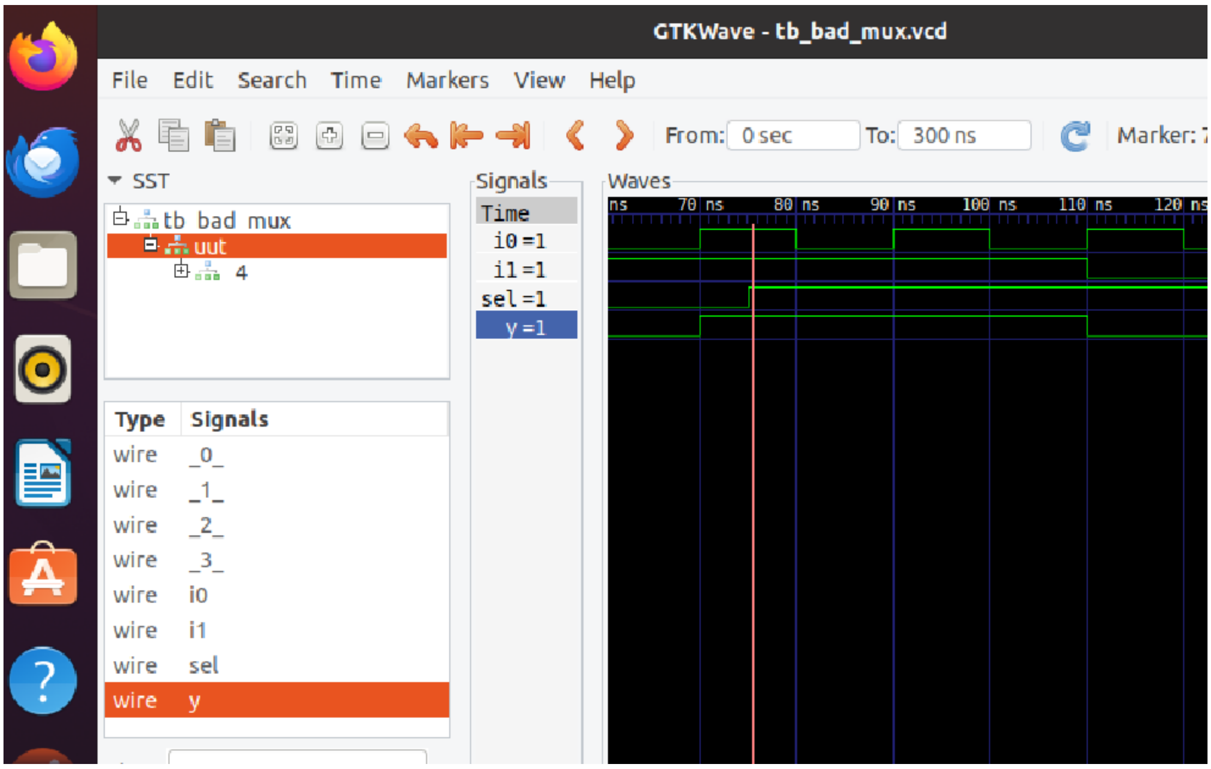
./a.out

gtkwave tb_bad_mux.vcd


File Edit Search Time Markers View Help
From: 0 sec To: 300 ns Marker: -
SST
tb_bad_mux
uut
Signals
Time
i0
i1
sel
y
Waves
100 ns
200 ns

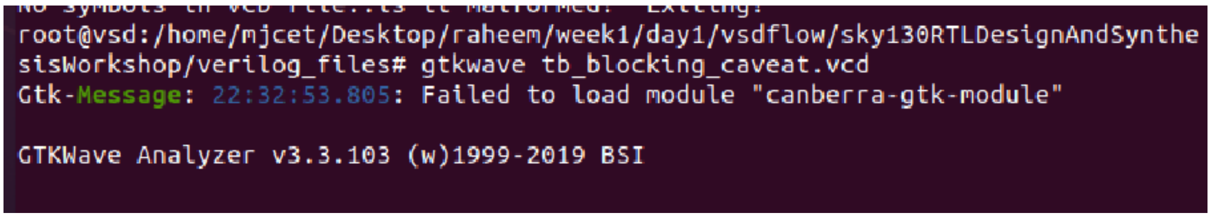
Type	Signals
wire	_0_
wire	_1_
wire	_2_
wire	_3_
wire	i0
wire	i1
wire	sel
wire	y

Synthesis Simulation Mismatch

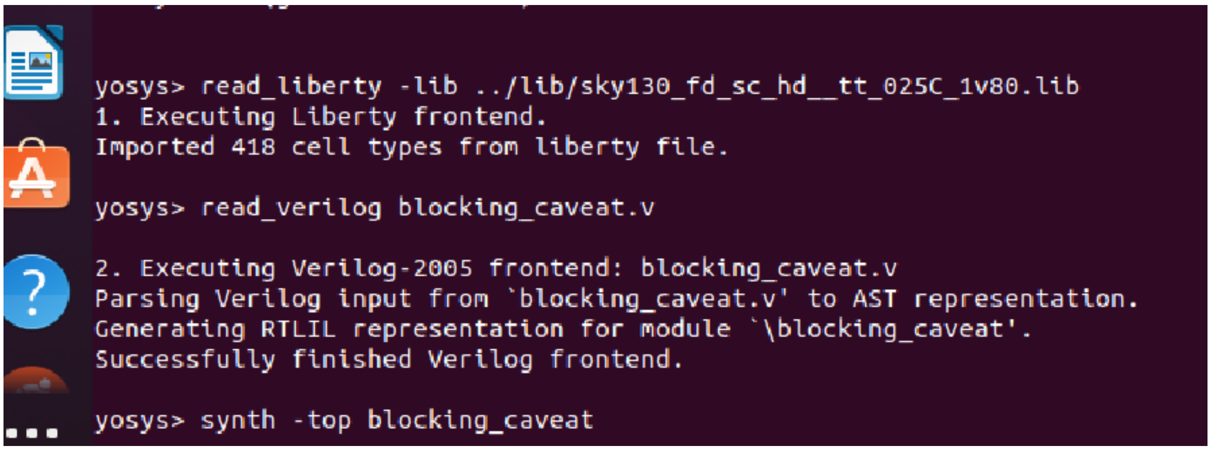


Type	Signals
wire	_0_
wire	_1_
wire	_2_
wire	_3_
wire	i0
wire	i1
wire	sel
wire	y

```
module blocking_caveat (input a , input b , input  c, output reg d);

reg x;

always @ (*)

begin

        d = x & c;

        x = a | b;

end

endmodule
```

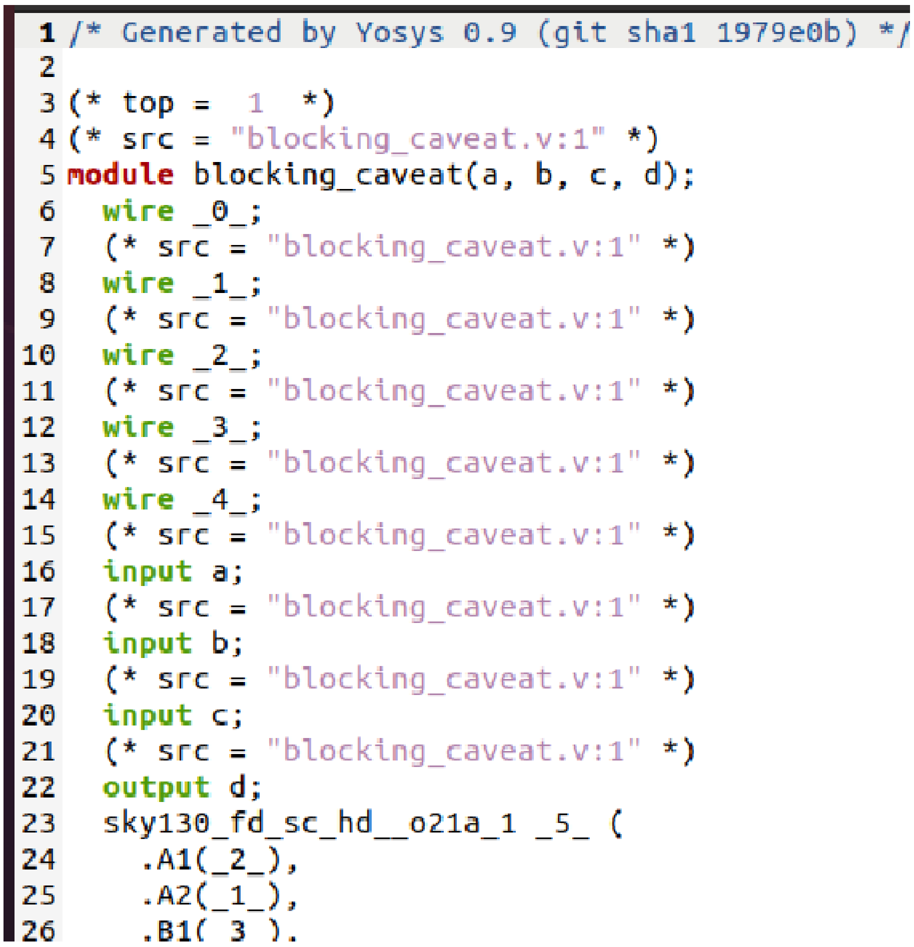


```
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthesisWorkshop/verilog_files# gtkwave tb_blocking_caveat.vcd
Gtk-Message: 22:32:53.805: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```

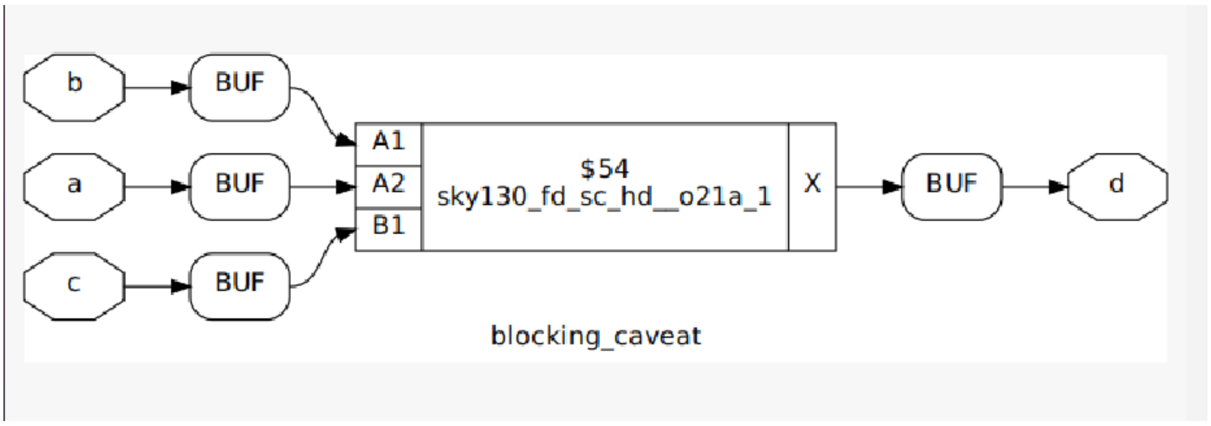


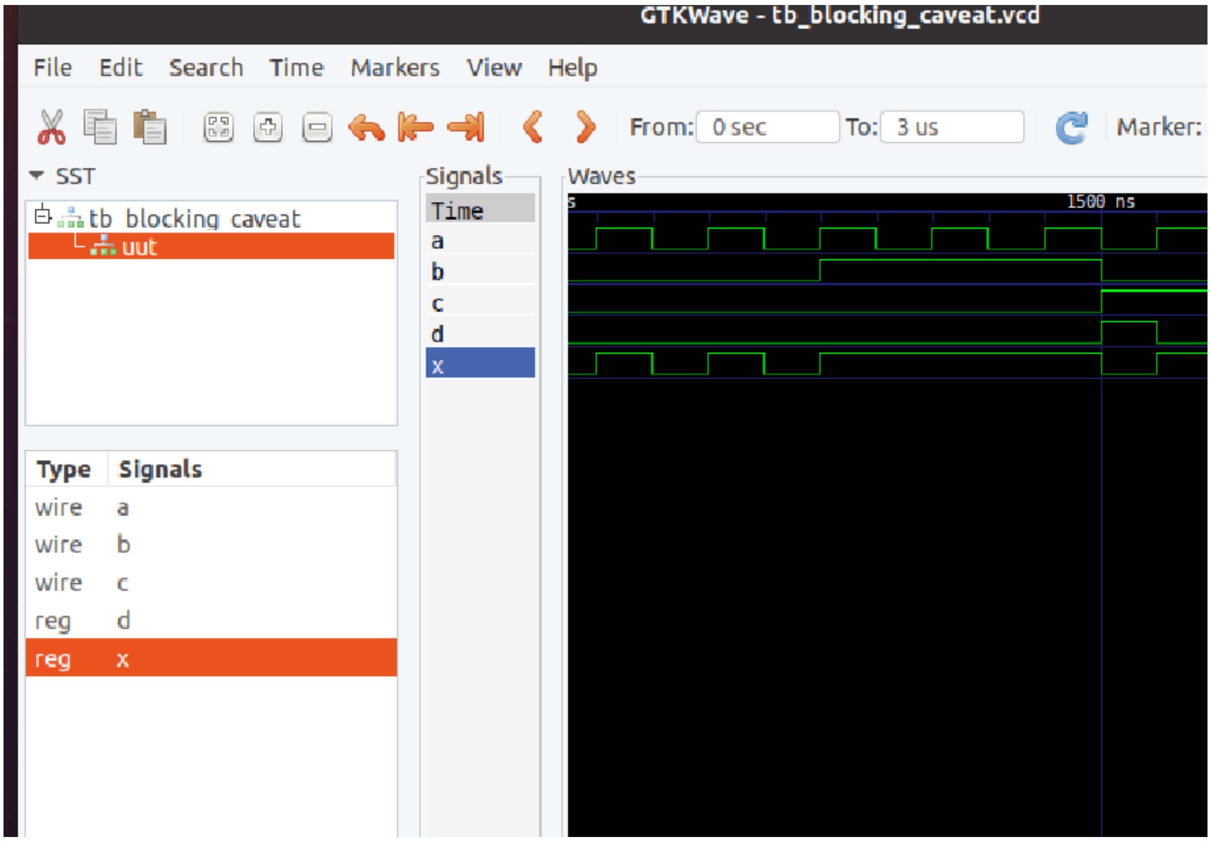
```
yosys> read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib
1. Executing Liberty frontend.
Imported 418 cell types from liberty file.

yosys> read_verilog blocking_caveat.v

2. Executing Verilog-2005 frontend: blocking_caveat.v
Parsing Verilog input from `blocking_caveat.v' to AST representation.
Generating RTLIL representation for module `\blocking_caveat'.
Successfully finished Verilog frontend.

yosys> synth -top blocking_caveat
```

```
read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

read_verilog blocking_caveat.v

synth -top blocking_caveat

abc -liberty ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

show

write_verilog blocking_caveat_net.v
```



```
/* Generated by Yosys 0.9 (git sha1 1979e0b) */

(* top =  1  *)
(* src = "blocking_caveat.v:1" *)
module blocking_caveat(a, b, c, d);
  wire _0_;
  (* src = "blocking_caveat.v:1" *)
  wire _1_;
  (* src = "blocking_caveat.v:1" *)
  wire _2_;
  (* src = "blocking_caveat.v:1" *)
  wire _3_;
  (* src = "blocking_caveat.v:1" *)
  wire _4_;
  (* src = "blocking_caveat.v:1" *)
  input a;
  (* src = "blocking_caveat.v:1" *)
  input b;
  (* src = "blocking_caveat.v:1" *)
  input c;
  (* src = "blocking_caveat.v:1" *)
  output d;
  sky130_fd_sc_hd__o21a_1 _5_ (
    .A1(_2_),
    .A2(_1_),
    .B1(_3_),
```




GTKWave - tb_blocking_caveat.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 3 us Marker:
SST
tb blocking caveat
uut
Type Signals
wire a
wire b
wire c
reg d
reg x
Signals
Time
a
b
c
d
x
Waves
1500 ns

read_liberty -lib ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

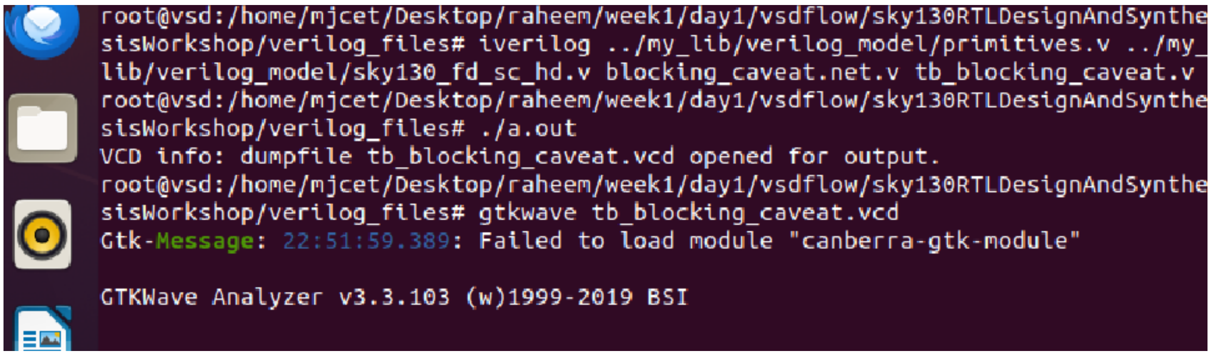
read_verilog blocking_caveat.v

synth -top blocking_caveat

abc -liberty ../lib/sky130_fd_sc_hd__tt_025C_1v80.lib

show

write_verilog blocking_caveat_net.v

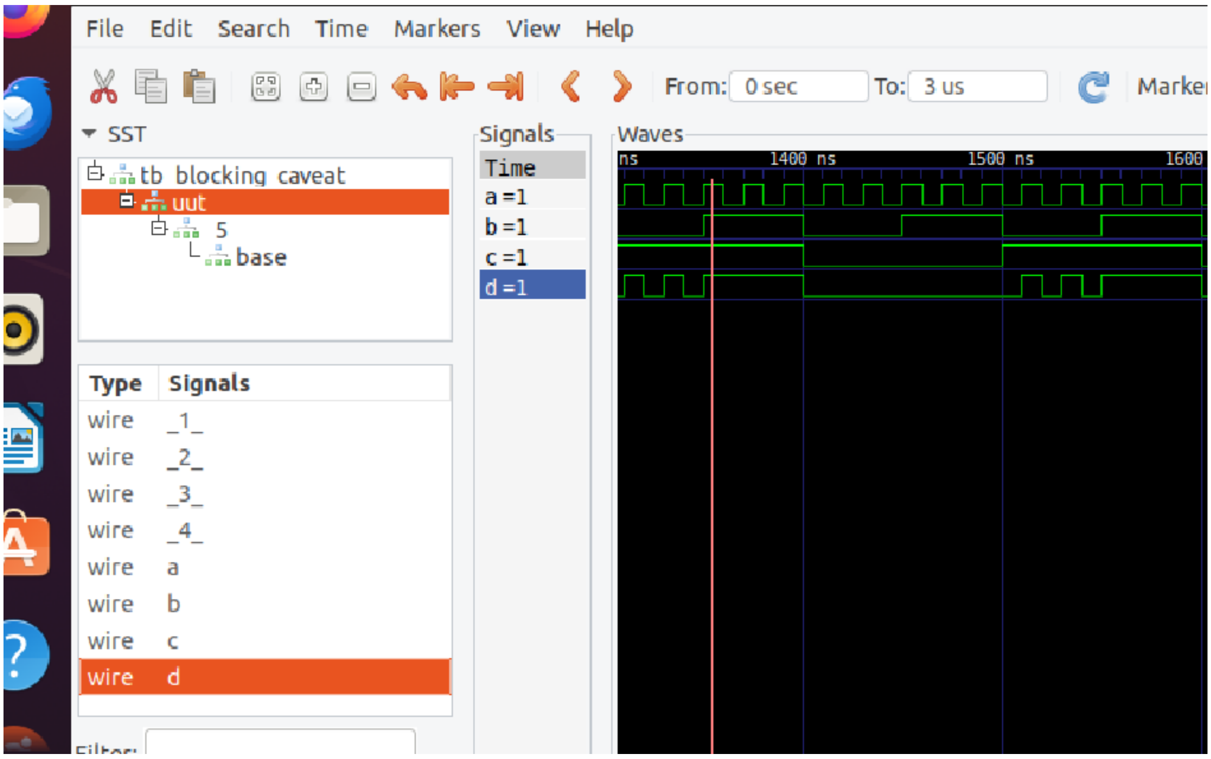


```
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# iverilog ../my_lib/verilog_model/primitives.v ../my_
lib/verilog_model/sky130_fd_sc_hd.v blocking_caveat_net.v tb_blocking_caveat.v
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# ./a.out
VCD info: dumpfile tb_blocking_caveat.vcd opened for output.
root@vsd:/home/mjcet/Desktop/raheem/week1/day1/vsdflow/sky130RTLDesignAndSynthe
sisWorkshop/verilog_files# gtkwave tb_blocking_caveat.vcd
Gtk-Message: 22:51:59.389: Failed to load module "canberra-gtk-module"

GTKWave Analyzer v3.3.103 (w)1999-2019 BSI
```

iverilog ../my_lib/verilog_model/primitives.v ../my_lib/verilog_model/sky130_fd_sc_hd.v blocking_caveat_net.v tb_blocking_caveat.v

./a.out

gtkwave tb_blocking_caveat.vcd



```
tb_blocking_caveat
  uut
    5
      base
```

Type	Signals
wire	_1_
wire	_2_
wire	_3_
wire	_4_
wire	a
wire	b
wire	c
wire	d

```
Time
a =1
b =1
c =1
d =1
```