

RTL Design using Verilog with SKY130 Technology



Table of contents:

Day 1 - Introduction to Verilog RTL design and Synthesis

Day 2 - Timing libs, hierarchical vs flat synthesis and efficient flop coding styles

Day 3 - Combinational and sequential optimizations

Day 4 - GLS, blocking vs non-blocking and Synthesis-Simulation mismatch

Day 5 – If, case for loop and for generate

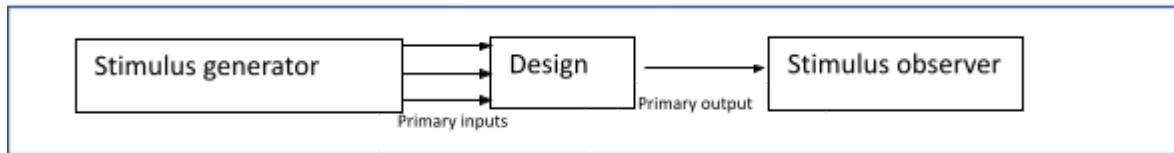
Commonly used commands.

Day 1 - Introduction to Verilog RTL design and Synthesis

As we all know hardware description languages are used to design, simulate and synthesize the digital electronic circuits. Any HDL simulator looks for changes on input. Whenever there is a change in primary inputs, the output is evaluated. If there is no change in input, then no output will be evaluated. So any design has primary inputs and primary outputs.

However, a test bench has no primary inputs and outputs. A test bench is used to check that whether the design behaves as expected or not. It has a stimulus generator corresponding to all input combinations for primary inputs of design and has a stimulus observer that observes the expected output collected from primary outputs of the design.

TEST BENCH



In this workshop SKY130 technology and iverilog software tool has been used to compile, simulate and synthesize Verilog codes. Following is the iverilog process flow:



We can see that in iverilog process flow, the design code and its test bench is given as input file which generates a VCD file (Value change dump format) file on simulation. And then using gtkwave one can see the waveforms generated after simulation.

VERILOG SIMULATION:

LAB SESSION:

Steps/commands:

1. load the library for all standard cells of sky130 and Verilog models files that contain all designs and test benches.

2. This can be done using cloning a github repo using command:

git clone [link of repo](#)

3. Load Verilog design file and test bench using following command:

iverilog good_mux.v tb_good_mux.v

Design file

Test bench

4. a.out file will be created which we need to execute using:

./a.out

5. This will generate a VCD file which can be used to generate waveforms using gtkwave.
gtkwave tb_good_mux.vcd

6. The command to read the Verilog file and see or edit the code is:

!gvim good_mux.v

Screenshot showing step no. 3,4 and 5:

```

ABC RESULTS: sky130_fd_sc_hd_o21a 1 cells: 1
ABC RESULTS: internal signals: 1
ABC RESULTS: input signals: 3
ABC RESULTS: output signals: 1
Removing temp directory.

yosys> write_verilog -noattr blocking_caveat.net.v
5. Executing Verilog backend.
Dumping module `blocking_caveat'.

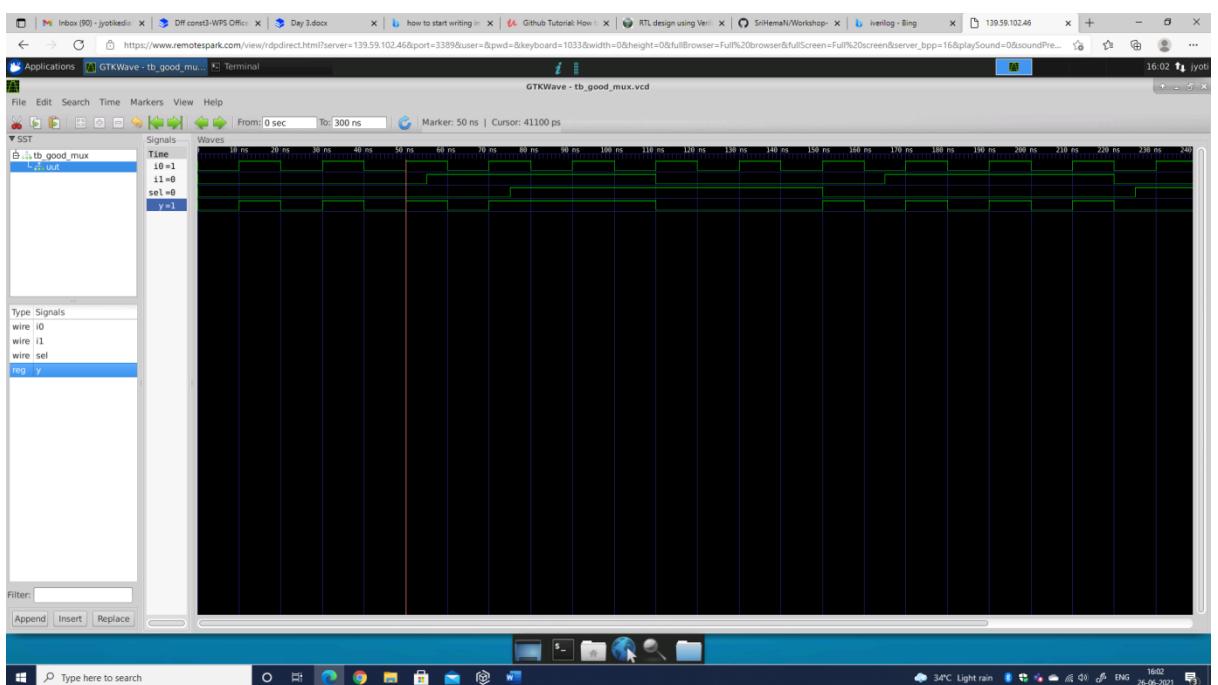
yosys> show
6. Generating Graphviz representation of design.
Writing dot description to `/home/jyoti/.yosys_show.dot'.
Dumping module blocking_caveat to page.
Exec: { test -f /home/jyoti/.yosys_show.dot.pid && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $$ >&3; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> exit

End of script. Logfile hash: d5cachdfc9, CPU: user 0.29s system 0.04s, MEM: 38.12 MB peak
Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O5)
Time spent: 55% 2x abc (0 sec), 25% 2x read Liberty (0 sec), ...
jyoti@rtlworkshop:~$ sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog good_mux.v tb_good_mux.v
jyoti@rtlworkshop:~$ ./tb_good_mux
VCD info: dumpfile tb_good_mux.vcd opened for output.
jyoti@rtlworkshop:~$ sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_good_mux.vcd

```

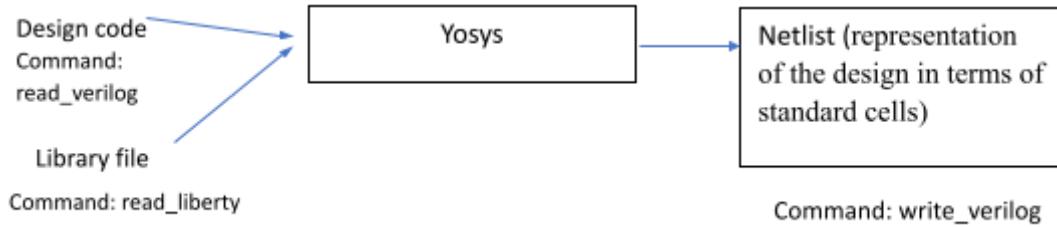
The terminal window shows the synthesis process using Yosys and the generation of waveforms using GTKWave. The output indicates the successful creation of a VCD file named `tb_good_mux.vcd`.

Screenshot showing waveforms of multiplexer:



SYNTHESIS:

A synthesizer converts RTL into netlist. The netlist is a representation of the design in terms of standard cells such as and, or, not gates. We have Yosys as a synthesizer tool. The flow of Yosys synthesizer is as follows:



LAB SESSION:

Steps/commands:

1. Read the library for all standard cells of sky130 using command:
read_liberty -lib path of library
2. Read the Verilog file which we want to synthesize using command:
read_verilog good_mux.v
3. Synthesize the top level entity in the given design using following command:
synth -top good_mux
4. Generate the netlist using:
abc -liberty path of library
5. The netlist can be written in readable format and then read using
 - a) **write_verilog -noattr good_mux_netlist.v**
 - b) **!gvim good_mux_netlist.v**
6. At the end the circuit synthesized can be viewed using command:
show

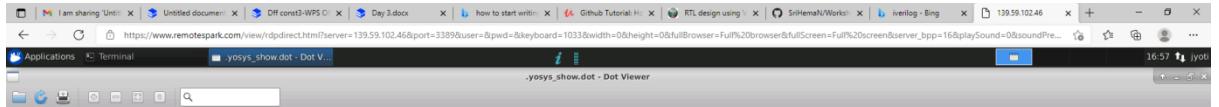
after synth command: shows statistics of how many number of blocks are there etc.

```

I am sharing Until... | Untitled document | Diff const3-WPS C... | Day 3.docx | how to start writing... | Github Tutorial H... | RTL design using... | SriHemaN/Works... | iverilog - Bing... | 139.59.102.46... | + - & x
Applications Terminal
File Edit View Search Terminal Tabs Help
Terminal
3.23.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module good_mux.
3.23.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module `good_mux'.
Removed a total of 0 cells.
3.23.3. Executing OPT_DFF pass (perform DFF optimizations).
3.23.4. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module `good_mux..
Removed 0 unused cells and 4 unused wires.
<suppressed -1 debug messages>
3.23.5. Finished fast OPT passes.
3.24. Analyzing HIERARCHY pass (managing design hierarchy).
3.24.1. Analyzing design hierarchy..
Top module: `good_mux'
3.24.2. Analyzing design hierarchy..
Top module: `good_mux'
Removed 0 unused modules.
3.25. Printing statistics.
==== good_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
3.26. Executing CHECK pass (checking for obvious problems).
Checking module good_mux...
Found and reported 0 problems.
yosys> 

```

This is the synthesized circuit. As we can see it is a 2 input multiplexer mapped with inputs and outputs of our design.



This is the netlist generated:

```

module good_mux(i0, i1, sel, y);
    wire _0;
    wire _1;
    wire _2;
    wire _3;
    input i0;
    input i1;
    input sel;
    output y;
    sky130_fd_sc_hd_mux2_1 _4_ (
        .A0(_0),
        .A1(_1),
        .S(_2),
        .X(_3)
    );
    assign _0 = i0;
    assign _1 = i1;
    assign _2 = sel;
    assign y = _3;
endmodule

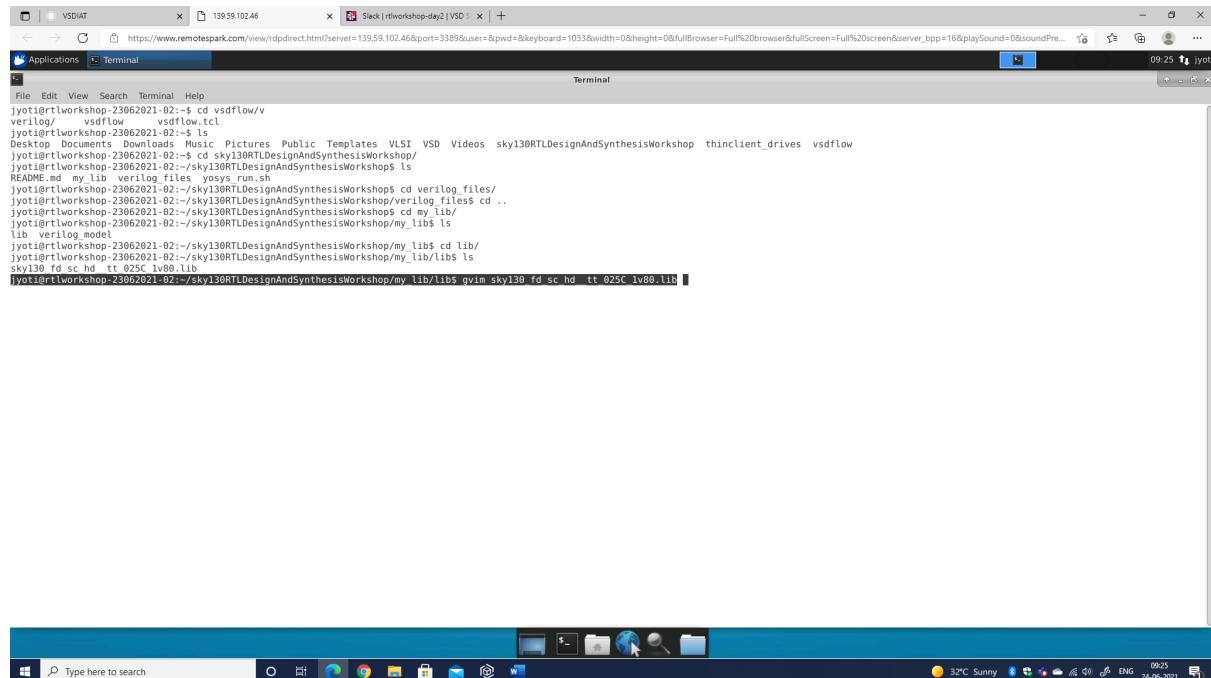
```

yosys> [redacted]

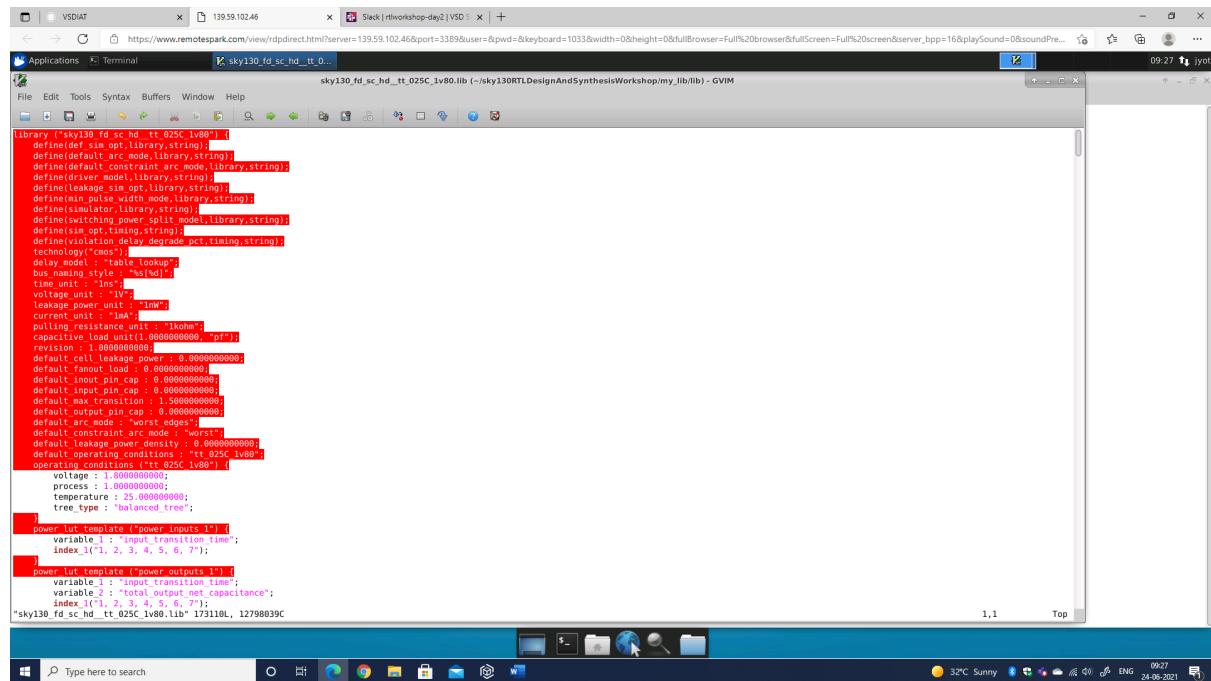
Day 2 - Timing libs, hierarchical vs flat synthesis and efficient flop coding styles

On Day 1 we understood that library contains all standard cells or basic digital modules which can be used to synthesize any given HDL code. Today we are going to view what is there inside the library file and we will also understand the significance of nomenclature of lib file.

Open the lib file using gvim command by specifying the relative path or going to that folder and open that to view.



The window that opens is



The first line in library specifies its name:

library ("sky130_fd_sc_hd_tt_025C_lv80"). Looking at a library file, three words coming into picture

P: Process

V: voltage

T: temperature

First of all, one should not edit this file.

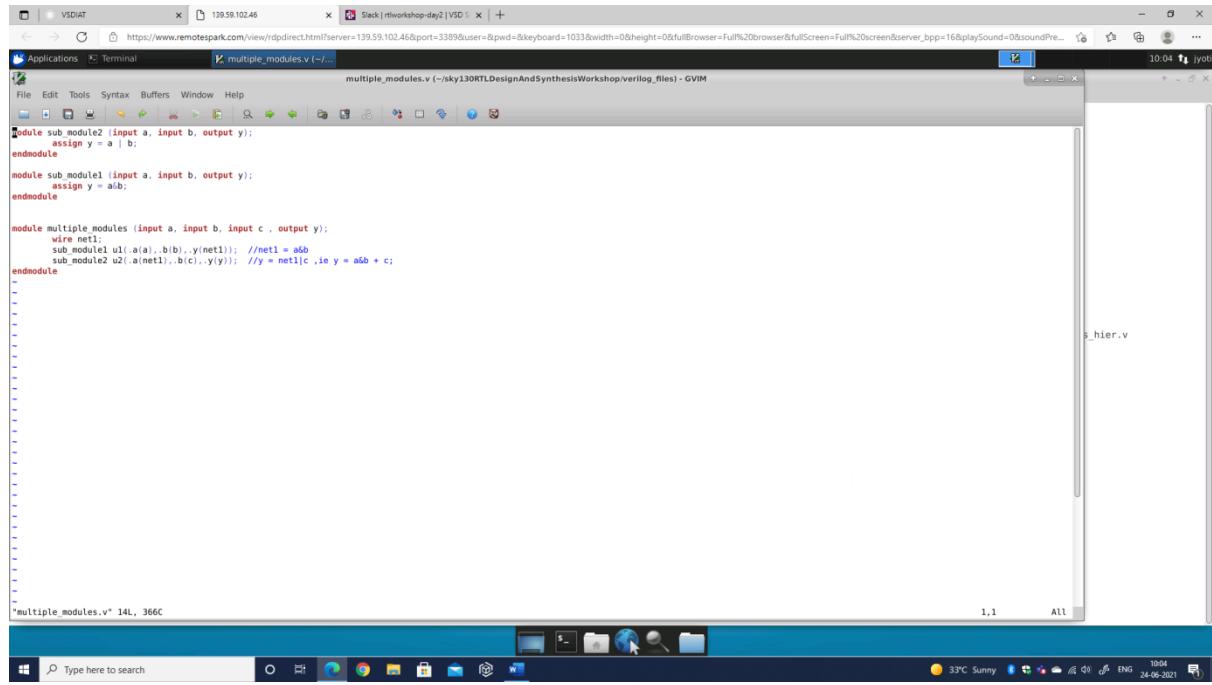
The library is created to model the variation possible in circuits due to process, voltage or temperature. The circuit behaviour is different when it is fabricated at different time, when there is different voltage or when temperature is different. So the design libraries develop their model to count upon all such variations in the circuit to model it as close to an actual circuit.

The library file tells the technology, delay, power and area. Each logical cell has different versions with low power, medium power or high power, fast or slow etc. this is because circuits require all variety of gates or modules to design the circuits as per the specifications.

So this is all about the library file which we can understand better if you look at it carefully.

Next we're going to see that how we can synthesise a top level entity when we give command Synth-top. We are also going to see that if we have a hierarchical design that is using various other modules as a part of it that is if a design is instantiating a module of other block or other design then how we can synthesise it as a black box and we can go deep into the circuit by flattening it and going to the circuit level off the individual blocks included in the top most entity.

So we can see in the screenshot back the multiple module is instantiating two sub modules sub module one and 2 as a part of it and it is mapping its own signals to those of the sub modules. So let us see when we synthesise this circuit what do we get.



The screenshot shows a Windows desktop environment with several open windows. In the center, a GVIM window displays a Verilog source code named "multiple_modules.v". The code defines three modules: "sub_module2", "sub_module1", and "multiple_modules". The "sub_module2" module has inputs "a" and "b" and output "y", with the assignment `assign y = a | b;`. The "sub_module1" module has inputs "a" and "b" and output "y", with the assignment `assign y = a&b;`. The "multiple_modules" module has inputs "a", "b", and "c", and output "y". It contains a wire declaration `wire net1;`, and two instances of the "sub_module1" module: `sub_module1 u1(.a(a),.b(b),.y(net1));` and `sub_module2 u2(.a(net1),.b(c),.y(y));`. The "multiple_modules" module also includes the assignment `//y = net1|c .ie y = a&b + c;`. The GVIM window has a vertical scrollbar on the right. Below the GVIM window, a terminal window shows the command `"multiple_modules.v" 14L, 366C`. The desktop taskbar at the bottom includes icons for File Explorer, Task View, Edge browser, File Explorer, Mail, and File Explorer. The system tray shows the date and time as 24-06-2021, 10:04, and the weather as 33°C Sunny.

```
module sub_module2 (input a, input b, output y);
    assign y = a | b;
endmodule

module sub_module1 (input a, input b, output y);
    assign y = a&b;
endmodule

module multiple_modules (input a, input b, input c , output y);
    wire net1;
    sub_module1 u1(.a(a),.b(b),.y(net1)); //net1 = a&b
    sub_module2 u2(.a(net1),.b(c),.y(y)); //y = net1|c .ie y = a&b + c;
endmodule
```

```
File Edit View Search Terminal Help
jyoti@rtlworkshop-23062021-02:~$ cd vsdflow/v
jyoti@rtlworkshop-23062021-02:~$ cd vsdflow/
jyoti@rtlworkshop-23062021-02:~$ ls
Desktop Documents Downloads Music Pictures Public Templates VLSI VSD Videos sky130RTLDesignAndSynthesisWorkshop thinclient_drives vsdflow
jyoti@rtlworkshop-23062021-02:~$ cd sky130RTLDesignAndSynthesisWorkshop/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ ls
Readme.txt my lib sky130rtl_design_and_synthesis.v
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd verilog_files/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd ..
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd my_lib/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ ls
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd lib/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ ls
sky130_fd_hd.td sky130_fd_sc_hd.td
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ mv sky130_fd_hd.td sky130_fd_sc_hd.td
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ mv sky130_fd_sc_hd.td sky130_fd_sc_hd_tt_025C_lv80.lib
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd lib/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ mv sky130_fd_sc_hd_tt_025C_lv80.lib sky130_fd_sc_hd_tt_025C_lv80.lib
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ cd verilog_files/
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ gvim mul
mul2.net.v          mult2.v           mult8.v           multiple_gvims.mul
multiple_module.opt.v multiple_module_opt2.v multiple_modules.v      multiple_modules.flat.v multiple_modules.hier.v
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop$ gvim multiple_multiple.module.v
```



```
VSDIAT x 139.59.102.46 x Slack *rtworkshop-day|VSD | +  
Application Terminal multiple_modules v (~)  
File Edit View Search Terminal Help  
Terminal  
10:11 19 Jun 2021 10:11 19 Jun 2021  
Yosys 0.9+4081 (git sha1 862e4eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O0)  
  
yosys> exit  
End of script. Logfile hash: da39a3ee5e, CPU: user 0.01s system 0.01s, MEM: 10.30 MB peak  
Yosys 0.9+4081 (git sha1 862e4eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O0)  
Time spent in commands (ms):  
jyoti@jyotitl-workshop:~/sky130RTLDesignAndSynthesis/workshop/verilog/files$ gvim multiple_module.v  
jyoti@jyotitl-workshop:~/sky130RTLDesignAndSynthesis/workshop/verilog_files$ yosys  
  
/-----/  
| yosys -- Yosys Open Synthesis Suite  
| Copyright (C) 2012 - 2020 Claire Xenia Xenia Wolf <claire@yosyshq.com>  
|  
| Permission to use, copy, modify, and/or distribute this software for any  
| purpose with or without fee is hereby granted, provided that the above  
| copyright notice and this permission notice appear in all copies.  
|  
| THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES  
| WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF  
| MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR  
| ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES  
| WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN  
| 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+4081 (git sha1 862e4eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O0)  
  
yosys> read liberty -lib ..;/my/lib/lib/sky130_fd_sc_hd_tt_025C_lv80.lib  
1. Executing Liberty frontend.  
Imported 428 cell types from liberty file.  
  
yosys> read_verilog mul  
mul2.net.v      mult_2.v      mult_8.v      multiple_module_opt.v      multiple_module_opt2.v      multiple_modules.v      multiple_modules_flat.v      multiple_modules_hier.v  
yosys> read_verilog mul  
mul2.net.v      mult_2.v      mult_8.v      multiple_module_opt.v      multiple_module_opt2.v      multiple_modules.v      multiple_modules_flat.v      multiple_modules_hier.v  
yosys> read_verilog multiple_modules.v
```

```

VS2017 139.59.102.46 Slack | * rtworkshop-day2 | VSC | +
Applications Terminal multiple_modules.v (-/..) 10:13 jyoti ...
File Edit View Search Terminal Help
sub_module1 1
sub_module2 1
=====
sub module1 ===
Number of wires: 3
Number of wire bits: 3
Number of public wires: 3
Number of public wire bits: 3
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 1
$ AND_ 1
$ OR_ 1
=====
sub module2 ===
Number of wires: 3
Number of wire bits: 3
Number of public wires: 3
Number of public wire bits: 3
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 1
$ OR_ 1
=====
design hierarchy ===
multiple modules 1
sub module1 1
sub module2 1
Number of wires: 11
Number of wire bits: 11
Number of public wires: 11
Number of public wire bits: 11
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 2
$ AND_ 1
$ OR_ 1
3.26. Executing CHECK pass (checking for obvious problems).
Checking module multiple.modules...

```

```

VS2017 139.59.102.46 Slack | * rtworkshop-day2 | VSC | +
Applications Terminal multiple_modules.v (-/..) 10:18 jyoti ...
File Edit View Search Terminal Help
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp 4".
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlclkp 1" without logic function.
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlclkp 2" without logic function.
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlclkp 4" without logic function.
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp 1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp 2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxtp 1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxtp 2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxtp 3".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxtp 4".
ABC: Library "sky130_fd_sc_hd_tt_025C_lv80" from "/home/jyoti/sky130RTLDesign" has 334 cells (94 skipped: 63 seq; 13 tri-state;
10 IO; 1 func; 0 def used). Time = 0.25 sec
ABC: Memory = 16.00 MB, Time = 0.25 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd_fa"
ABC: + strash
ABC: + ifraig
ABC: + score
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + dc2
ABC: + dretime
ABC: + strass
ABC: + net -n
ABC: + Edch -f
ABC: + Snt
ABC: + Sput
ABC: + write_bif <abc-temp-dir>/output.blif
4.3.2. Re-integrating ABC results.
ABC RESULTS: sky130_fd_sc_hd_lpfw_inputisop_1 cells: 1
ABC RESULTS: internal signals: 0
ABC RESULTS: input signals: 2
ABC RESULTS: output signals: 1
Removing temp directory.
yosys> show
5. Generating Graphviz representation of design.
ERROR: For formats different than 'ps' or 'dot' only one module must be selected.
yosys show multiple.modules
6. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module multiple.modules to page 1.
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s -'/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>&3; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 

```

It is not showing the actual gates but is showing the names as sub module 1 and submodule 2. It is showing a hierarchy as the multiple modules has instantiated submodule 1 and 2. Also we can see at the net list back to sub-modules are instantiated inside the design multiple modules.

```

File Edit View Search
multiple_modules.v (~/. . . multiple_modules_hier...
File Edit Tools Syntax Buffers Window Help
Generated by Yosys 0.9+4081 (git sha1 862e94b, gcc 7.5.0-3ubuntu1-18.04 -FPIC -Os) */
ABC: Memory: 16.00 M
ABC: Timing: Detected
ABC: + strash
ABC: + ifraig
ABC: + sccor
ABC: + write_blf <abc-...
4.3.2. Re-integrating A
ABC RESULTS: sky130_fd_sc_hd_and2_0_3_
ABC RESULTS: int
ABC RESULTS: 
ABC RESULTS: 
ABC RESULTS: 
Removing temp directory
yosys> show
5. Generating Graphviz
ERROR: For formats different from 'ps' or 'dot' only one module must be selected.
yosys> show multiple_modules
6. Generating Graphviz
Writing dot description to '/home/jyoti/.yosys.show.dot'.
Dumping module multiple_modules to page 1.
Exec: { test -f '/home/jyoti/.yosys.show.dot.pid' && fuser -s '/home/jyoti/.yosys.show.dot.pid' 2> /dev/null; } || ( echo $$>63; exec xdot '/home/jyoti/.yosys.show.dot'; ) 3> '/home/jyoti/.yosys.show.dot.pid' &
yosys> write_verilog -noattr multiple_modules_hier.v
7. Executing Verilog backend.
Dumping module '\multiple_modules'.
Dumping module '\sub_module1'.
Dumping module '\sub_module2'.
yosys> igim multiple_modules_hier.v
8. Shell command: gvim multiple_modules_hier.v
yosys> igim multiple_modules_hier.v
9. Shell command: gvim multiple_modules_hier.v
yosys> flatten
10. Executing FLATTEN pass (flatten design).
Deleting now unused module sub_module1.
Deleting now unused module sub_module2.
<suppressed -2 debug messages>
yosys> write_verilog -noattr multiple_modules_flat.v
11. Executing Verilog backend.
Dumping module '\multiple_modules'.
yosys> igim multiple_modules_flat.v
12. Shell command: gvim multiple_modules_flat.v
yosys>

```

Now If we want to look at that what is there inside the sub modules we can do that by running command flatten. during synthesis if we use flatten we can see in the netlist that as compared to the hierarchical design net list there are no sub modules shown but only the lowest level circuit is listed there. We can also see in the circuit diagram that we get after synthesis, that it is no more showing sub module as the black boxes but it is showing the proper gates which were there inside the sub modules.

```

File Edit View Search Terminal Help
Removing temp directory.
yosys> show
5. Generating Graphviz representation of design.
ERROR: For formats different than 'ps' or 'dot' only one module must be selected.
yosys> show multiple_modules
6. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys.show.dot'.
Dumping module multiple_modules to page 1.
Exec: { test -f '/home/jyoti/.yosys.show.dot.pid' && fuser -s '/home/jyoti/.yosys.show.dot.pid' 2> /dev/null; } || ( echo $$>63; exec xdot '/home/jyoti/.yosys.show.dot'; ) 3> '/home/jyoti/.yosys.show.dot.pid' &
yosys> write_verilog -noattr multiple_modules_hier.v
7. Executing Verilog backend.
Dumping module '\multiple_modules'.
Dumping module '\sub_module1'.
Dumping module '\sub_module2'.
yosys> igim multiple_modules_hier.v
8. Shell command: gvim multiple_modules_hier.v
yosys> igim multiple_modules_hier.v
9. Shell command: gvim multiple_modules_hier.v
yosys> flatten
10. Executing FLATTEN pass (flatten design).
Deleting now unused module sub_module1.
Deleting now unused module sub_module2.
<suppressed -2 debug messages>
yosys> write_verilog -noattr multiple_modules_flat.v
11. Executing Verilog backend.
Dumping module '\multiple_modules'.
yosys> igim multiple_modules_flat.v
12. Shell command: gvim multiple_modules_flat.v
yosys>

```

```

Generated by Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -Os) +"
module multiple_modules(a, b, c, y);
  wire 1;
  wire 2;
  wire 3;
  wire 4;
  wire 5;
  input a;
  input b;
  input c;
  wire net1;
  wire u1.a;
  wire u1.b;
  wire u1.y;
  wire u2.a;
  wire u2.b;
  wire u2.y;
  output y;
  sub module u1 (
    .a(a),
    .b(b),
    .y(net1)
  );
  sub module u2 (
    .a(net1),
    .b(c),
    .y(y)
  );
endmodule

module sub module(a, b, y);
  wire 0;
  wire 1;
  wire 2;
  input a;
  input b;
  output y;
  sub module u1 (
    .a(1),
    .b(0),
    .y(2)
  );
  assign 4 = u2.b;
  assign 3 = u2.a;
  assign 2 = 1;
  assign u2.a = net1;
  assign u2.b = c;
  assign y = u2.y;
  assign u1.b = u1.a;
  assign u1.y = 2;
  assign u1.a = a;
  assign u1.b = b;
  assign net1 = u1.y;
endmodule

```

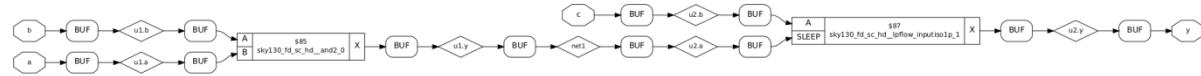
"multiple_modules_flat.v" 43L, 795C 1,1 All 6 46,13 12%

```

module sub module2(a, b, y);
  wire 0;
  wire 1;
  wire 2;
  input a;
  input b;
  output y;
  sub module u1 (
    .a(1),
    .b(0),
    .y(2)
  );
  assign 1 = b;
  assign 0 = a;
  assign y = 2;
endmodule

```

"multiple_modules.hier.v" 43L, 795C 1,1 All 6 46,13 12%



FLIP FLOPS

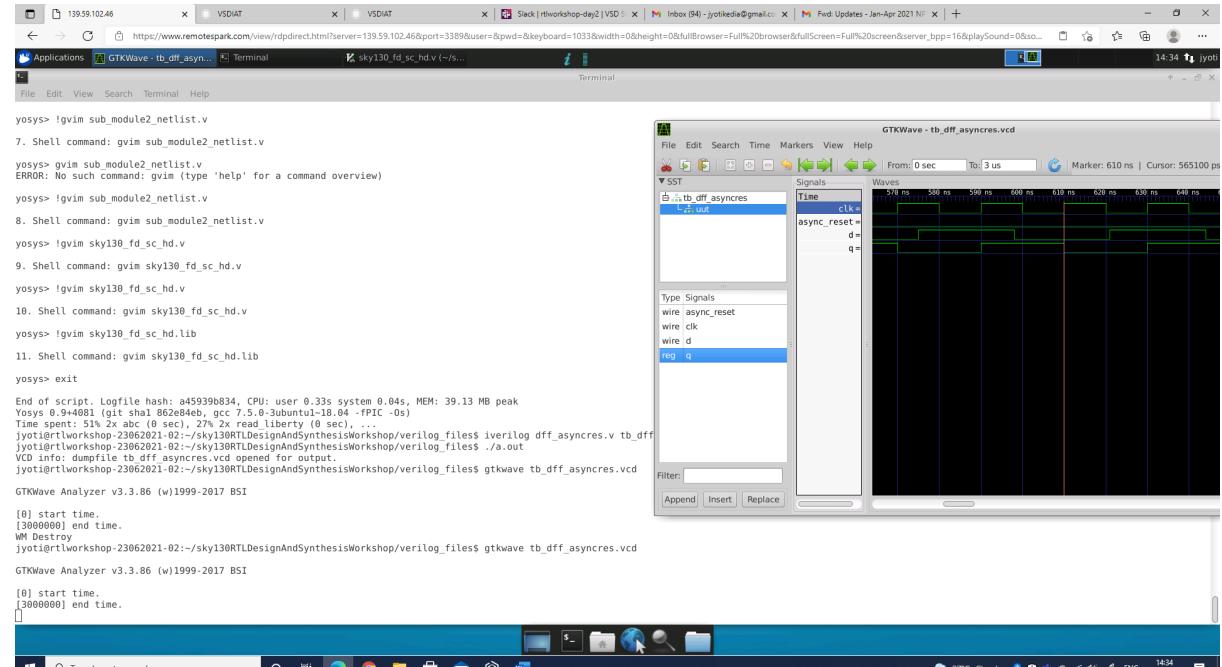
Let us look at the need of the flip flops in digital circuits. Whenever we design a combinational circuit the outputs are evaluated as soon as whenever there is a change in the input. So due to the delays that is propagation delay across the various gates sometimes the correct outputs are not evaluated because intermediate values propagate or reach to their final value only after the propagation delay and during that time if those intermediate values are used by the other gates cascaded then those cascaded gates are going to evaluate a wrong output for sometime. Although finally they will be reaching their exact or the correct value but momentarily the output of the combinational gates will be incorrect. This is known as glitch in order to avoid this glitching which may result into an altogether incorrect output

when we have a series of cascaded gates we can use a storage element in between these cascaded combinations which can hold onto their previous value for sometime.

So in this session we're going to look at the designs of D flip flop with various combinations of synchronous or asynchronous set or reset inputs. The set or reset inputs are nothing but the inputs which are used to initialize the flops.

ASYNCHRONOUS RESET

Here is an example of a D flip flop with asynchronous reset. Asynchronous reset means that output can be 0 whenever the reset input goes high irrespective of the clock pulse.



```

yosys> !gvim sub_module2.netlist.v
7. Shell command: gvim sub_module2.netlist.v
yosys> gvim sub_module2.netlist.v
ERROR: No such command: gvim (type 'help' for a command overview)
yosys> !gvim sub_module2.netlist.v
8. Shell command: gvim sub_module2.netlist.v
yosys> !gvim sky130_fd_sc_hd.v
9. Shell command: gvim sky130_fd_sc_hd.v
yosys> !gvim sky130_fd_sc_hd.v
10. Shell command: gvim sky130_fd_sc_hd.v
yosys> !gvim sky130_fd_sc_hd.lib
11. Shell command: gvim sky130_fd_sc_hd.lib
yosys> exit

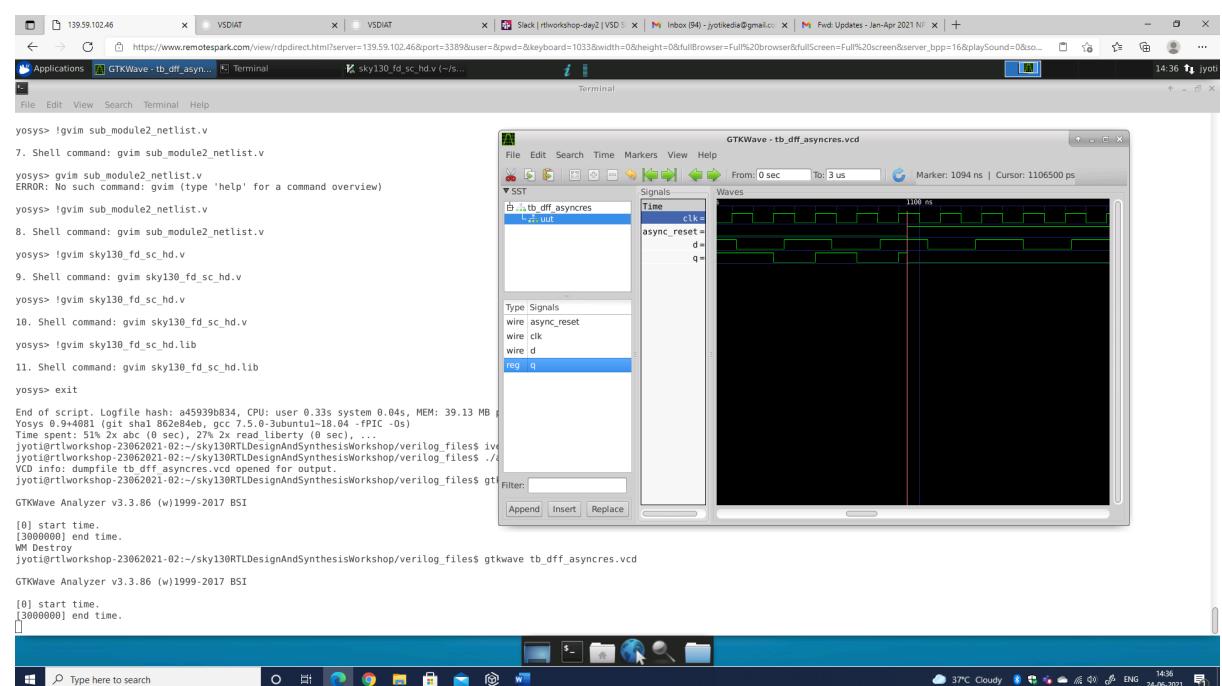
End of script. Logfile hash: a45939b834, CPU: user 0.33s system 0.04s, MEM: 39.13 MB peak
Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O5)
Time spent: 51s 2x abc (0 sec), 27s 2x read liberty (0 sec), ...
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog dff_asyncre.v tb_dff_asyncre.vcd
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out
VCD info: dumpfile tb_dff_asyncre.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_asyncre.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[30000000] end time.
WM Destroy
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_asyncre.vcd
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[30000000] end time.

```

```

yosys> !gvim sub_module2.netlist.v
7. Shell command: gvim sub_module2.netlist.v
yosys> gvim sub_module2.netlist.v
ERROR: No such command: gvim (type 'help' for a command overview)
yosys> !gvim sub_module2.netlist.v
8. Shell command: gvim sub_module2.netlist.v
yosys> !gvim sky130_fd_sc_hd.v
9. Shell command: gvim sky130_fd_sc_hd.v
yosys> !gvim sky130_fd_sc_hd.v
10. Shell command: gvim sky130_fd_sc_hd.v
yosys> !gvim sky130_fd_sc_hd.lib
11. Shell command: gvim sky130_fd_sc_hd.lib
yosys> exit

End of script. Logfile hash: a45939b834, CPU: user 0.33s system 0.04s, MEM: 39.13 MB peak
Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-18.04 -fPIC -O5)
Time spent: 51s 2x abc (0 sec), 27s 2x read liberty (0 sec), ...
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog dff_asyncre.v tb_dff_asyncre.vcd
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out
VCD info: dumpfile tb_dff_asyncre.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_asyncre.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[30000000] end time.
WM Destroy
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_asyncre.vcd
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

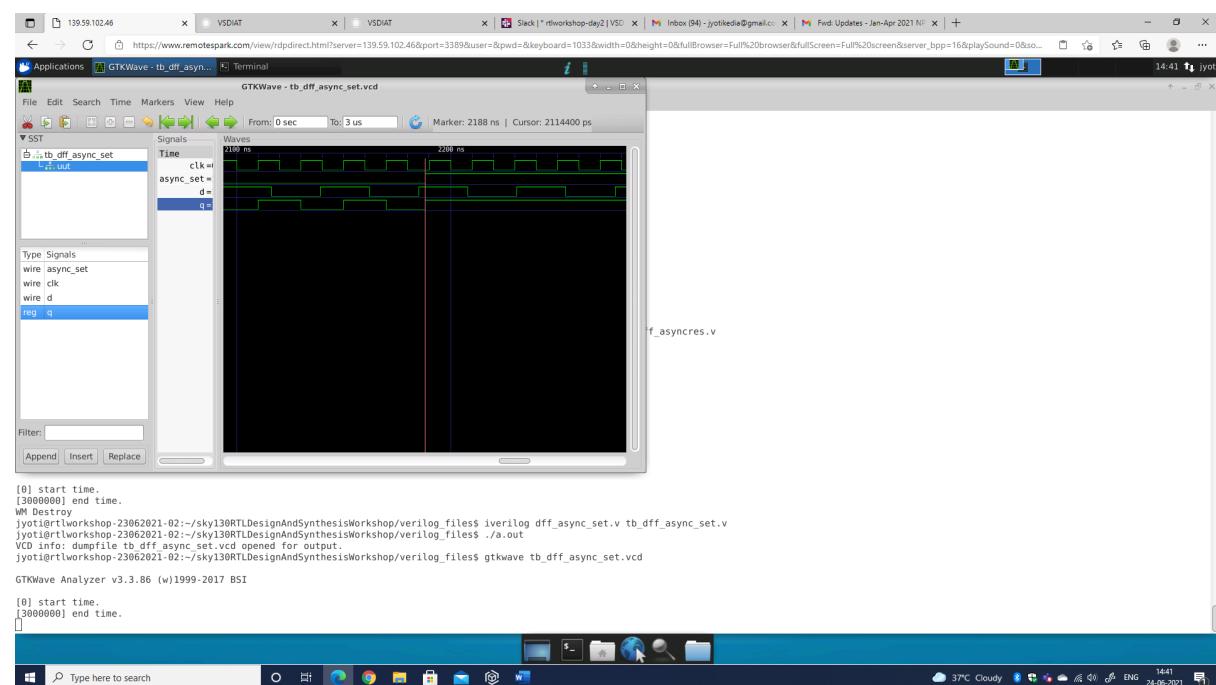
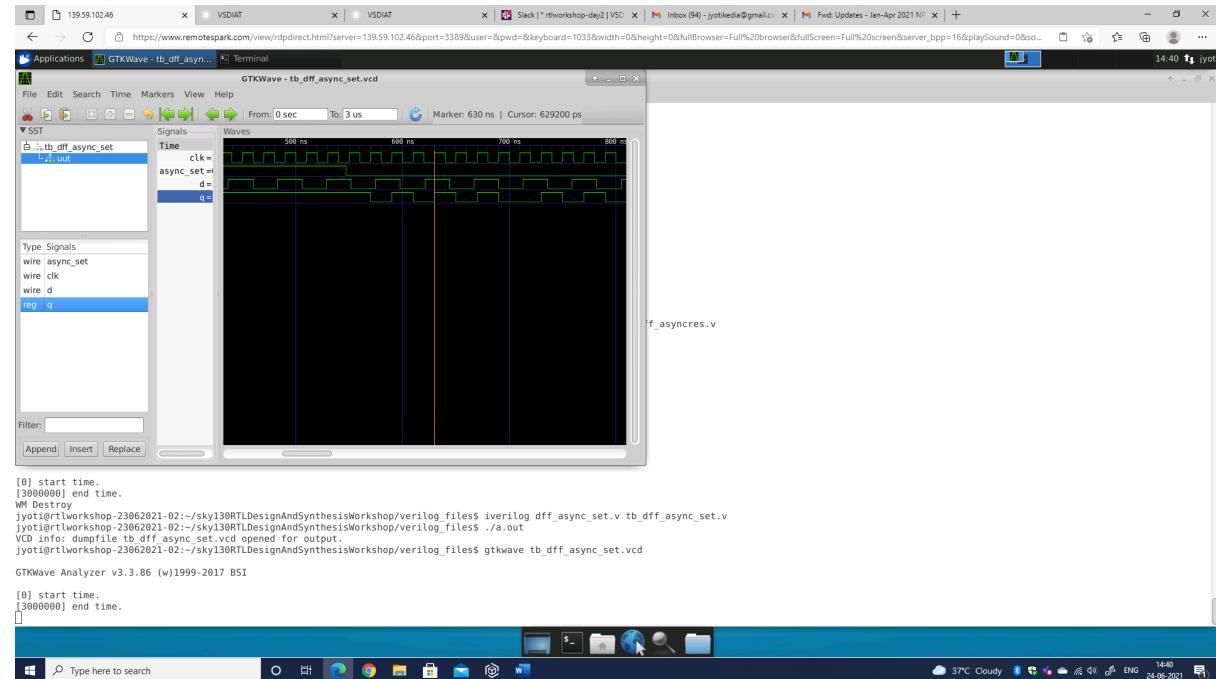
[0] start time.
[30000000] end time.

```

It is clearly visible in the above two wave forms that when reset is zero the output follows the input on the next clock pulse whenever it arrives but when reset input is high the output goes to zero irrespective of the clock pulse.

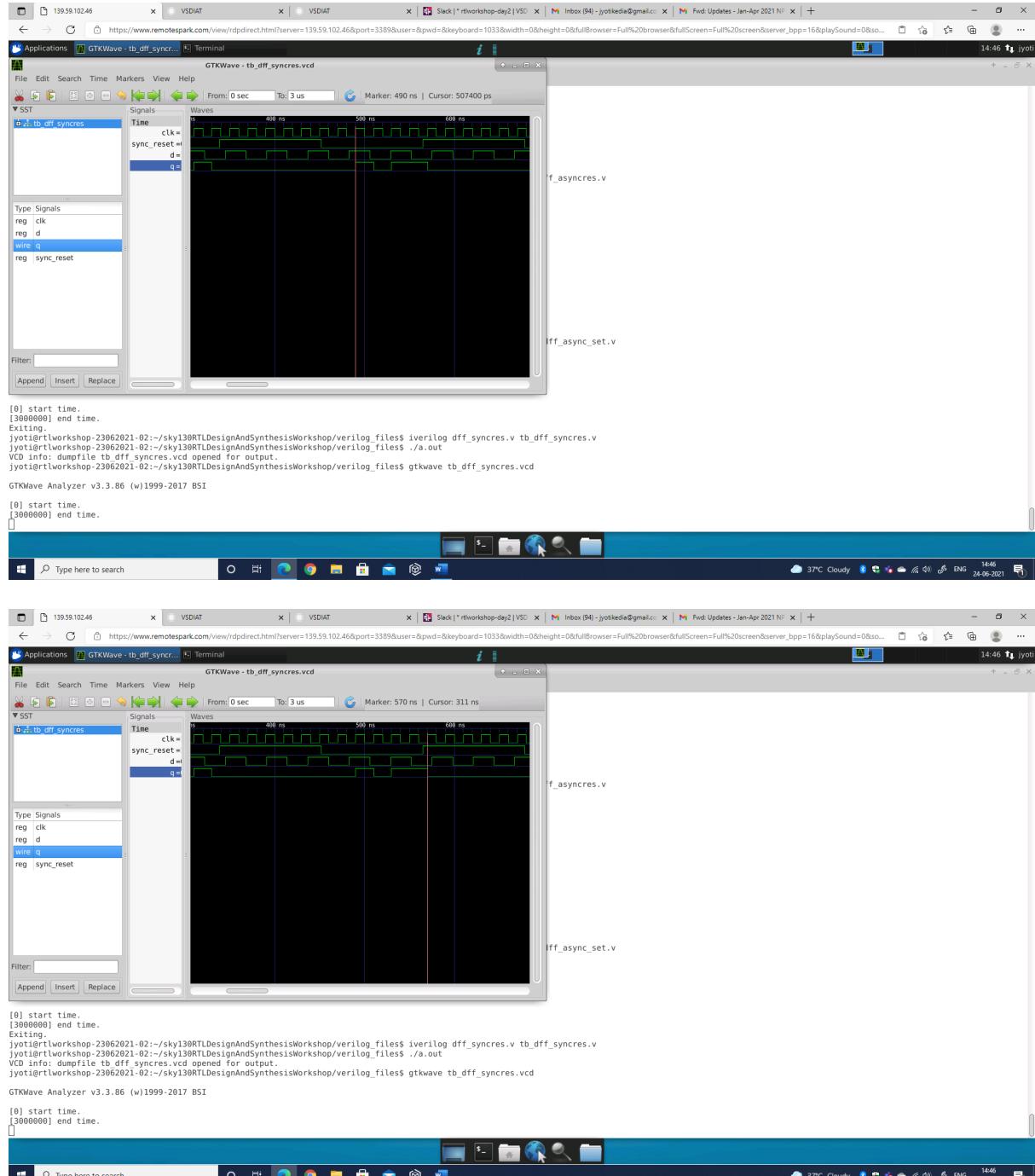
ASYNCHRONOUS SET

Similarly we have next example in which there is an asynchronous set input that sets the output of D flip flop to 1 whenever set input goes high. And if set input is low the output will follow the input on the next clock pulse arrival.



SYNCHRONOUS RESET

This is the example of synchronous reset. What happens here is that whenever reset goes to 1 the output goes to zero only when the new clock arrives since this reset input is synchronized with the clock. And further when this reset input goes low output will follow the input again on the next clock pulse arrival.



And in the next few snapshots we can see the synthesizer circuit of the flip flops with asynchronous/synchronous set or reset

SYNTHESIS OF ASYNCHRONOUS FLIP FLOP

```

139.59.102.46 VSDAT VSDAT Slack * rtwkshop-day2 | VSDAT | inbox (95) - jyotedia@gmail.com Not handling set and reset of a | + 15:05 jyoti ...
Applications Terminal .yosys_show.dot - Dot V... Terminal
File Edit View Search Terminal Help
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_4".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxbp_1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxbp_2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_4".
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdfflip_1" without function
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdfflip_2" without function
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdffxp_4" without function
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdffxp_1" without function
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdffxp_2" without function
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdffxp_3" without function
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdffxp_4" without function
ABC: Library "sky130_fd_sc_hd_tt_025C_1v80" from "/home/jyoti/sky130RTLDesign"
18 no func; 0 dont_use. Time = 0.22 sec
ABC: Memory = 16.00 MB. Time = 0.22 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd_fa")
ABC: + strash
ABC: + lraig
ABC: + scorrr
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + dftime
ABC: + strash
ABC: + Sget -n
ABC: + Sdch -f
ABC: + lraig
ABC: + Sput
ABC: + write blif <abc-temp-dir>/output.blif

5.1.2. Re-integrating ABC results:
ABC RESULTS: sky130_fd_sc_hd_cikinv_1 cells: 1
ABC RESULTS: internal signals: 0
ABC RESULTS: input signals: 1
ABC RESULTS: output signals: 1
Removing temp directory.

yosys> show
7. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module dff_asyncset to page 1.
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 
```

ASYNCHRONOUS SET

```

139.59.102.46 VSDAT VSDAT Slack * rtwkshop-day2 | VSDAT | inbox (95) - jyotedia@gmail.com Not handling set and reset of a | + 15:09 jyoti ...
Applications Terminal .yosys_show.dot - Dot V... Terminal
File Edit View Search Terminal Help
11.23.5. Finished fast OPT passes.
11.24. Executing HIERARCHY pass (managing design hierarchy).
11.24.1. Analyzing design hierarchy.
Top module: /dff_async_set
11.24.2. Analyzing design hierarchy.
Top module: /dff_async_set
Removed 0 unused modules.
11.25. Printing statistics.

*** dff_async_set ***
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
$ DFF_PP1 1

11.26. Executing CHECK pass (checking for obvious problems).
Checking module dff_async_set...
Found and reported 0 problems.

yosys> abc -liberty .../my_lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
12. Executing ABC pass (technology mapping using ABC).

12.1. Extracting gate netlist of module '\dff_async_set' to '<abc-temp-dir>/in
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs
Don't call ABC as there is nothing to map.
Removing temp directory.

yosys> show
13. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module dff_async_set to page 1.
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 
```

SYNCHRONOUS RESET

```

139.59.102.46 VSDIAT VSDIAT Slack | * rtworkshop-day2 | VSD | Inbox (95) - jyotedia@gmail.com Not handling set and reset of a... + Applications Terminal yosys_show_dot - Dot Viewer 15:15 jyoti
File Edit View Search Terminal Help
4.23.5. Finished fast OPT passes.
4.24.. Executing HIERARCHY pass (managing design hierarchy).
4.24.1. Analyzing design hierarchy..
Top module: \dff_syncres
4.24.2. Analyzing design hierarchy..
Top module: \dff_syncres
Removed 0 unused modules.
4.25.. Printing statistics.
*** dff_syncres ***
Number of wires: 5
Number of wire bits: 5
Number of public wires: 5
Number of public wire bits: 5
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 1
$_SDF_PP0_
4.26.. Executing CHECK pass (checking for obvious problems).
Checking module dff.syncres...
Found and reported 0 problems.
yosys> abc -liberty .../my_lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
5. Executing ABC pass (technology mapping using ABC).
5.1. Extracting gate netlist of module '\dff_syncres' to '<abc-temp-dir>/input'
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs
Don't call ABC as there is nothing to map.
Removing temp directory.
yosys> show
6. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys/show.dot'.
Dumping module 'dff_syncres' to 'show.dot'
Exec: test -f '/home/jyoti/.yosys/show.dot.pid' && fuser -s /home/jyoti/.yosys/show.dot.pid 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys/show.dot'; ) 3> '/home/jyoti/.yosys/show.dot.pid' &
yosys> [REDACTED]

```

OPTIMIZATION

Now let us look at few interesting optimizations in the design. Although the detailed sequential and combinational optimizations are shown in the next session. But here we see very commonly used and interesting optimizations. For example let us say we want to multiply a given 3 bit number by two. If we write down the truth table showing input as a 3 bit number and an output as a multiple of input number by 2. We will observe that output is having its 3 most significant bits same as that of the input and the LSB is zero throughout. This means that we need not to have any circuitry to multiply a given number by two rather we can just copy the input number and append a zero as a LSB. so this kind of optimization can lead to a great reduction of this circuit which is anticipated while writing a code.

```

module mul2 (input [2:0] a, output [3:0] y);
    assign y = a * 2;
endmodule

```

```

139.59.102.46 VSDIAT VSDIAT Slack | rtworkshop-day2 | VSD | Inbox (95) - jyotkedia@gmail.co... Not handling set and reset of a ... + 15:51 jyoti ...
Applications Terminal mult_2.v (~/sky130RTL...) Terminal
File Edit View Search Terminal Help
Finding identical cells in module '\mul2'.
Removed a total of 0 cells.

6.23.3. Executing OPT_DFF pass (perform DFF optimizations).
6.23.4. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \mul2..
6.23.5. Finished fast OPT passes.
6.24. Executing HIERARCHY pass (managing design hierarchy).
6.24.1. Analyzing design hierarchy..
Top module: \mul2
6.24.2. Analyzing design hierarchy..
Top module: \mul2
Removed 0 unused modules.

6.25. Printing statistics.
==== mul2 ====
Number of wires: 2
Number of wire bits: 7
Number of public wires: 2
Number of public wire bits: 7
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 0

6.26. Executing CHECK pass (checking for obvious problems).
Checking module mul2...
Found and reported 0 problems.
yosys> abc -liberty ../my/lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
7. Executing ABC pass (technology mapping using ABC).
7.1. Extracting gate netlist of module '\mul2' to 'abc-temp-dir>/input.blif'..
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.

yosys> [REDACTED]

```

[REDACTED]

```

139.59.102.46 VSDIAT VSDIAT Slack | rtworkshop-day2 | VSD | Inbox (95) - jyotkedia@gmail.co... Not handling set and reset of a ... + 15:51 jyoti ...
Applications Terminal mult_2.v (~/sky130RTL...) yosys_show.dot - Dot Viewer Terminal
File Edit View Search Terminal Help
7.2. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module mul2 to page 1
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $$ >&3; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> [REDACTED]

```

[REDACTED]

```

graph LR
    a((a)) --> B[2:0 - 3:1<br/>0 -> 0:0]
    B --> y((y))
    style B fill:#fff,stroke:#000,stroke-width:1px
    style a fill:#fff,stroke:#000,stroke-width:1px
    style y fill:#fff,stroke:#000,stroke-width:1px

```

Netlist

The screenshot shows a Windows desktop environment. In the center, there is a terminal window titled 'Terminal' with the command prompt 'yosys>'. The window displays the following log output:

```

6.25. Printing statistics.
== mul2 ==
Number of wires: 2
Number of wire bits: 7
Number of public wires: 2
Number of public wire bits: 7
Number of memory wires: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 0

6.26. Executing CHECK pass (checking for obvious problems).
Checking module mul2...
Found and reported 0 problems.

yosys abc -liberty .../my_lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
7. Executing ABC pass (technology mapping using ABC).

7.1. Extracting gate netlist of module `mul2` to `<abc-temp-dir>/input`
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0
Don't call ABC as there is nothing to map.
Removing temp directory.

yosys> show
8. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module mul2 to page 1.
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' }

yosys> write_verilog -noattr mul2_net.v
9. Executing Verilog backend.
Dumping module `mul2`.

yosys> !gvim mul2_net.v
10. Shell command: gvim mul2_net.v

yosys>

```

To the right of the terminal, a GVIM window is open with a Verilog file named 'mul2.net.v'. The code in the GVIM window is:

```

module mul2(a, y);
    input [2:0] a;
    output [3:0] y;
    assign y = { a, 1'h0 };
endmodule

```

So like we can see in the synthesised circuit and the generated netlist that there is no circuit at all but the input is directly going as output and a zero is appended at the rightmost bit. The same thing can also be verified in the netlist generated.

DAY 3: COMBINATIONAL AND SEQUENTIAL OPTIMIZATIONS

As we know we have two types of circuits in digital logic design one is combinational and other is sequential so whenever we design any circuit in VLSI we always look for where we can design our circuit optimising on area and power so that is why whenever we are making an RTL design we need to optimise

The techniques of combinational optimisation:

1. constant propagation
 - Direct Optimisation technique
2. Boolean logic Optimisation : a more efficient simplified Boolean logic expression using techniques like
 - K map
 - Quin McCluskey algorithms

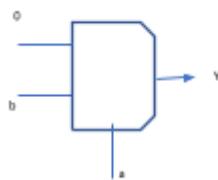
So let us take the first example. As it can be seen from the verilog code that it is trying to design a two input mux in which output why will be assigned the value be when a input is 1. And output will get a value of zero if a input is zero. What is expected out of the code is as shown in the diagram below. But when we analyze it we find that the boolean expression for Y is

$$Y = ab + a'0$$

If you look at the boolean expression carefully we can see that the second term becomes zero and what we are left with is Y is equal to a and b. it means that we can realise or synthesise the given code only using an and gate instead of a multiplexer. Let us check this optimization using our simulator and synthesiser.

This optimization can be done using the following command:

```
opt_clean -purge
```




```

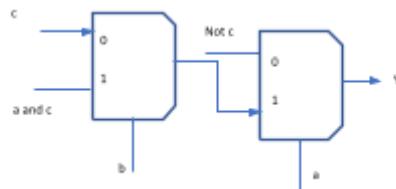
VS-DAT 139.59.102.46 99 - jyotiheda@gmail.com | Invitation: RTL Design Workshop | + 
Applications Terminal mult_2.v (~/sky130RTL) yosys_show.dot - Dot V... 11:04 11:04 ...
File Edit View Search Terminal Help
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_4".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxbp_1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxbp_2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sdfxtp_4".
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdclck_2" without
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdclck_4" without
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxtp".
ABC: Library "sky130_fd_sc_hd_tt_025C_lv80" from "/home/jyoti/sky130RTLDesign"
18 no_func; 0 dont_use. Time = 0.23 sec
ABC: Memory usage: 16.00 MB. Time = 0.23 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd_fa
ABC: + strash
ABC: + lraig
ABC: + scor
ABC: Warning: The network is combinational (run "lraig" or "lraig_sweep").
ABC: + dftime
ABC: + strash
ABC: + Sget -n
ABC: + Sdch -f
ABC: + lraig
ABC: + Sput
ABC: + write_bif <abc-temp-dir>/output.blif

5.1.2. Re-integrating ABC results.
ABC RESULTS: sky130_fd_sc_hd_and2_0 cells: 1
ABC RESULTS: internal signals: 0
ABC RESULTS: input signals: 2
ABC RESULTS: output signals: 1
Removing temp directory.

yosys> show
6. Generating Graphviz representation of design.
Writing dot description to "/home/jyoti/yosys.show.dot".
Dumping module opt_check to page 1
Exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 
```

So it can be clearly seen in the synthesizer circuit that it is not a two input multiplexer but the design can be realised only using an AND gate.

Here is another example optimization check. As we can see again from the code, hello let it can be two cascaded multiplexers having control input as A and B. The circuit that is expected is as follows



When we try to find out the boolean expression for output it minimises to a XNOR c . Now let us observe the synthesise circuit for the given design.

```

module opt_check4 (input a , input b , input c , output y);

assign y = a?(b?(a & c ):c):(!c);

endmodule

```

File Edit View Search Terminal Help

```

ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp_4".
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlc1p_1" without logic function.
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlc1p_2" without logic function.
ABC: Scl.LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdlc1p_4" without logic function.
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp_1".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp_2".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp_3".
ABC: Scl.LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxbp_4".
ABC: Library "sky130_fd_sc_hd_tt_025C_lv80" from "/home/jyoti/sky130RTLDesign/rtl/sky130_fd_sc_hd_tt_025C_lv80" has 334 cells (94 skipped: 63 seq; 13 tri-state;
18 no func; 0 dont use). Time = 0.22 sec
ABC: Memory usage: 16.00 MB. Time = 0.22 sec
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd_fa")
ABC: + strash
ABC: + ifraig
ABC: + scorr
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + dftime
ABC: + dretime
ABC: + strash
ABC: + Sget -n
ABC: + Sget -f
ABC: + Sntf
ABC: + Sput
ABC: + write_bif <abc-temp-dir>/output.bif

5.1.2. Re-integrating ABC results.
ABC RESULTS: sky130_fd_sc_hd_xnor2_1 cells: 1
ABC RESULTS: internal signals: 3
ABC RESULTS: input signals: 3
ABC RESULTS: output signals: 1
ABC RESULTS: 1

Removing temp directory.

yosys> show
6. Generating Graphviz representation of design.
Writing dot description to '/home/jyoti/.yosys_show.dot'.
Dumping module opt_check4 to page 1.
Exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s -'/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> write_verilog -noattr opt_check4_nelist.v
7. Executing Verilog backend.
Dumping module `opt_check4'.

yosys> !gvim[]
```

```

graph LR
    a((a)) -- BUF --> A(( ))
    b((b)) -- BUF --> B(( ))
    c((c)) -- BUF --> C(( ))
    A --- AND[opt_check4]
    B --- AND
    C --- AND
    AND -- Y --> y((y))
    A --> a
    B --> b

```

File Edit Tools Syntax Buffers Window Help

opt_check4_nelist.v (~\sky130RTLDesignAndSynthesisWorkshop\verilog_files) - GVIM1

```

Generated by Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-10.04 -fPIC -Os) */

module opt_check4(a, b, c, y);
    wire 0;
    wire 1;
    wire 2;
    wire 3;
    wire 4;
    wire 5;
    wire 6;
    input a;
    input b;
    input c;
    output y;
    sky130_fd_sc_hd_xnor2_1 _7_ (
        .A(0),
        .B(3),
        .Y(6)
    );
    assign 0 = c;
    assign 3 = a;
    assign 4 = b;
    assign y = 6;
endmodule

opt_check4_nelist.v
module opt_check4(input a, input b, input c, output y);
    assign y = (a&b)&(c)|(b&c);
endmodule

```

File Edit View Search Terminal Help

11:51 jyoti

File Edit View Search Terminal Help

11:51 jyoti

As we can clearly observe from the net list as well as from the synthesizer circuit that the function optimized is $Y = a \text{xnor} c$ instead of two 2 input multiplexers. this is how we can optimize the various circuits to minimise up an area as well as power.

Multiple module optimization

Similar example of combinational circuit optimization is given below in which we have a hierarchical module that instantiates for sub modules inside itself. However when we look at the expected circuit

we see that output is just tied to 1 and all other inputs are not contributing anyhow to the output. When we synthesise the circuit, we found expected results.

The screenshot shows a Windows desktop environment with several windows open:

- Terminal Window:** Shows the Yosys tool being used to synthesize a Verilog module named `multiple_module_opt2.v`. The terminal output includes ABC library loading logs and synthesis results. It shows that the module has 12.1 nodes and 9.14 top-level nodes. The synthesized circuit consists of four sequential cells (labeled U1, U2, U3, U4) each containing a BUF gate followed by a tri-state output node. The inputs are labeled a, b, c, d, and the outputs are labeled y1, y2, y3, y4.
- GIMP Window:** A Dot Viewer window titled ".yosys_show.dot - Dot Viewer" displays the synthesized circuit as a graph. The graph shows four nodes (U1.y, U2.y, U3.y, U4.y) connected to four corresponding nodes (y1, y2, y3, y4). Each node is represented by a diamond shape with a BUF gate inside. The connections are shown as directed edges between the nodes.
- System Tray:** Shows standard system icons like battery level, signal strength, and volume.

SEQUENTIAL OPTIMIZATIONS

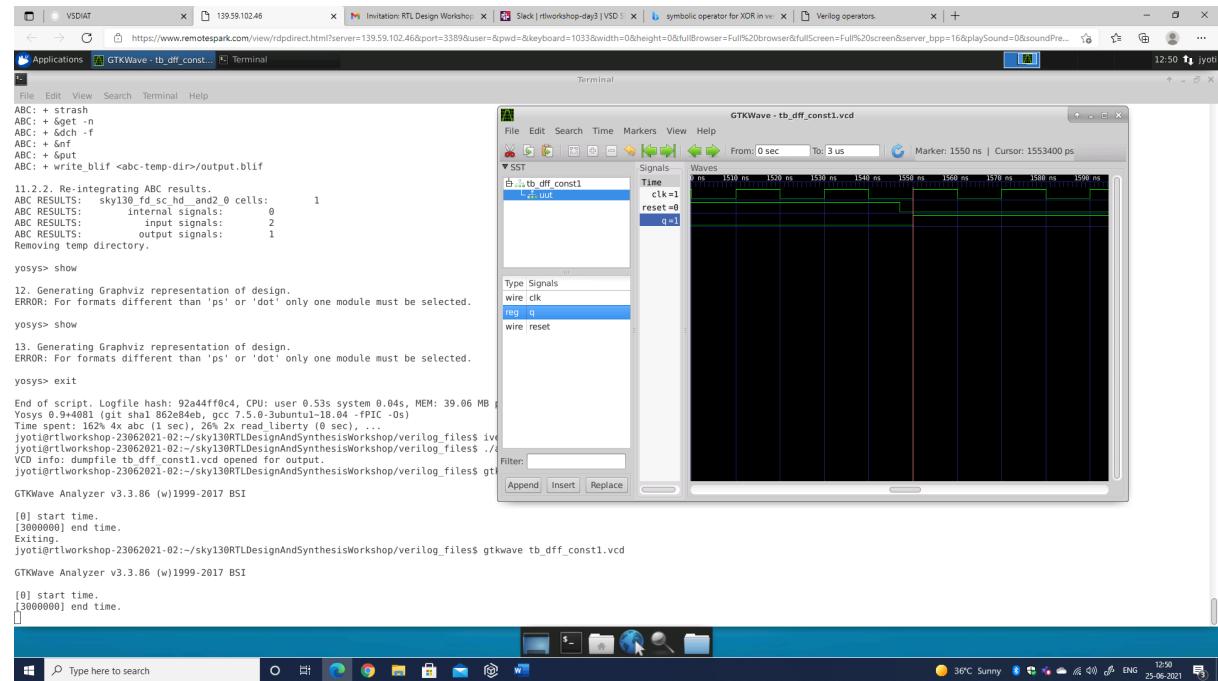
Now let us look at the optimization in sequential circuits. In sequential circuits sometimes on observing the code we find that output is sometimes set to a particular value either one or zero or to anyone of the control input irrespective of the clock pulse. Such flip flops are known as sequential constants. Such a behaviour can be observed from the simulation of the sequence circuit. Is the output of a sequential circuit is set to a constant value irrespective of the clock pulses or any given input then

we understand that there is no need to connect a flop or latch to represent that circuit. So we're going to see several examples in this regard.

Example1:

```
module dff_const1(input clk, input reset, output reg q);
always @ (posedge clk, posedge reset)
begin
    if(reset)
        q <= 1'b0;
    else
        q <= 1'b1;
end
endmodule
```

In the first example the code is such that when reset is set to one the output is 0 else output is assigned a constant value one.



When reset is 0, the q waits for next clock to go 1 or to follow d input. It means that the output is not having a constant value throughout. Therefore, we need to have this flip flop. This cannot be optimized as sequential constant.

Example 2:

```
module dff_const2(input clk, input reset, output reg q);
always @ (posedge clk, posedge reset)
begin
    if(reset)
        q <= 1'b1;
    else
        q <= 1'b1;
end
endmodule
```

In this example the reset input of the entity is connected to the set input of latch that gives the value of one to the output when reset is set to 1. And when reset goes to zero the output gets a constant value of 1.

```

VSIM[1]: VSDIAT <--> 139.59.102.46 <--> Invitation RTL Design Workshop | Slack | rtwkshop-dej3 | VSD | symbolic operator for XOR in vc | Verilog operators
File Edit View Search Terminal Help
ABC RESULTS:      input signals:      2
ABC RESULTS:      output signals:     1
Removing temp directory.

yosys> show
12. Generating Graphviz representation of design.
ERROR: For formats different than 'ps' or 'dot' only one module must be selected.

yosys> show
13. Generating Graphviz representation of design.
ERROR: For formats different than 'ps' or 'dot' only one module must be selected.

yosys> exit
End of script. Logfile hash: 92a44ff8c4, CPU: user 0.53s system 0.04s, MEM: 39.006 MB peak
Yosys 0.9+4081 (git 1ab11862e84ed, gcc 7.5.0, binutils 2.31, SPICE -0s)
Time spent: 162.4 ns (4 abc 1.862e84ed sec), 260.2x read liberty (0 sec).
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out
VCD info: dumpfile tb_dff_const2.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI
[0] start time.
[3000000] end time.
Exiting.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI
[0] start time.
[3000000] end time.
WM Destroy
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out
VCD info: dumpfile tb_dff_const2.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_const2.vcd
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI
[0] start time.
[3000000] end time.

```

The screenshot shows a terminal window with the output of a Verilog simulation and a GTKWave window displaying the waveform for the tb_dff_const2.vcd file. The terminal shows the ABC tool running and the GTKWave window showing a waveform for three signals: clk, reset, and q. The clk signal is a square wave. The reset signal starts at 1 and then goes to 0. The q signal remains at 1 while reset is 1, and then remains at 0 after reset goes to 0.

when set is 1, the q is 1 and when set goes to 0 it remains 1 and waits for next clock to take input d which is again 1. So q remains 1 throughout. This is an example of sequential constant.

As we have seen in the above two examples that how a sequential constant can be observed using the wave forms. Further the behavior can be verified by synthesising the circuits. Following are the snapshots it shows the synthesised circuit corresponding to example one and two respectively.

Synthesis of Example 1

```

VS-DIAT  x  139.59.102.46  x  Invitation: RTL Design Workshop...  x  Slack | rtiworkshop-day3 | VSD  x  symbolic operator for XOR in ve...  x  Verilog operators  x  +
File Edit View Search Terminal Help
File Edit View Search Terminal Help
3.23.5. Finished fast OPT passes.
3.24. Executing HIERARCHY pass (managing design hierarchy).
3.24.1. Analyzing design hierarchy..
Top module: \dff_const1
3.24.2. Analyzing design hierarchy..
Top module: \dff_const1
Removed 0 unused modules.
3.25. Printing statistics.
*** dff_const1 ***
Number of wires: 3
Number of wire bits: 3
Number of public wires: 3
Number of public wire bits: 3
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 1
$_DFF_PP0_ 1
3.26. Executing CHECK pass (checking for obvious problems).
Checking module dff_const1...
Found and reported 0 problems.
yosys> abc -liberty ..\my_lib\lib\sky130_fd_sc_hd_tt_025C_1v80.lib
4. Executing ABC pass (technology mapping using ABC).
4.1. Extracting gate netlist of module '\dff_const1' to <abc-temp-dir>/input.
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.
yosys> show
5. Generating Graphviz representation of design.
Writing dot description to /home/jyoti/.yosys_show.dot.
Dumping module dff_const1 to page 1.
exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 

```

Synthesis of Example 2

```

VS-DIAT  x  139.59.102.46  x  Invitation: RTL Design Workshop...  x  Slack | rtiworkshop-day3 | VSD  x  symbolic operator for XOR in ve...  x  Verilog operators  x  +
File Edit View Search Terminal Help
File Edit View Search Terminal Help
7.23.5. Finished fast OPT passes.
7.24. Executing HIERARCHY pass (managing design hierarchy).
7.24.1. Analyzing design hierarchy..
Top module: \dff_const2
7.24.2. Analyzing design hierarchy..
Top module: \dff_const2
Removed 0 unused modules.
7.25. Printing statistics.
*** dff_const2 ***
Number of wires: 3
Number of wire bits: 3
Number of public wires: 3
Number of public wire bits: 3
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 0
7.26. Executing CHECK pass (checking for obvious problems).
Checking module dff_const2...
Found and reported 0 problems.
yosys> abc -liberty ..\my_lib\lib\sky130_fd_sc_hd_tt_025C_1v80.lib
8. Executing ABC pass (technology mapping using ABC).
8.1. Extracting gate netlist of module '\dff_const2' to <abc-temp-dir>/input.
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.
yosys> show
9. Generating Graphviz representation of design.
Writing dot description to /home/jyoti/.yosys_show.dot.
Dumping module dff_const2 to page 1.
exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
yosys> 

```

The synthesizer circuits are showing the same behavior that has been observed with simulations. Like example one is showing the circuit as flip flop since it was not an example of sequential constant however the Second Circuit given by example two is certainly an example of sequential constant.

Example 3

```

module dff_const3(input clk, input reset, output reg q);
reg q1;

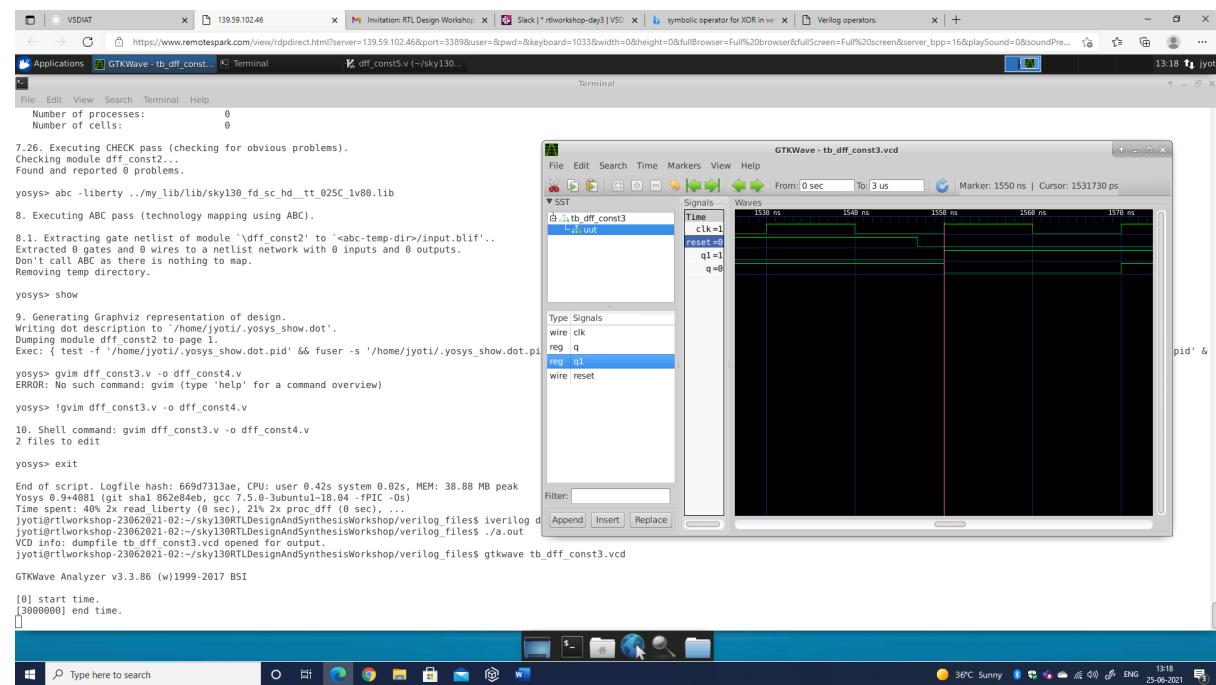
```

```

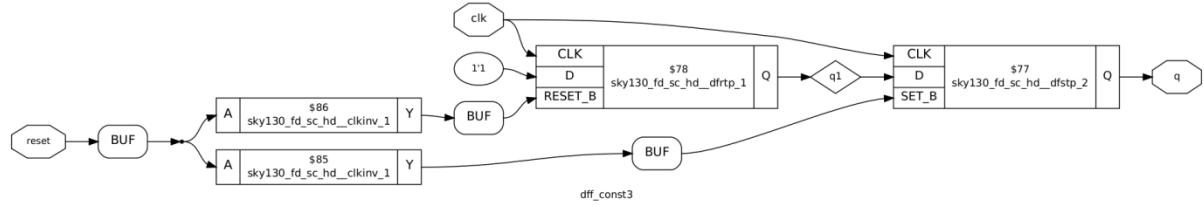
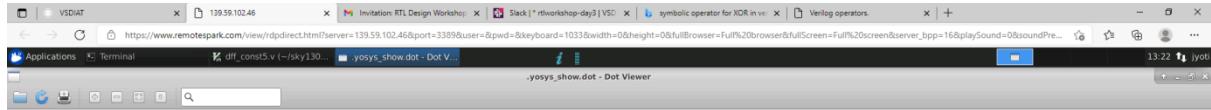
always @ (posedge clk, posedge reset)
begin
    if(reset)
        begin
            q <= 1'b1;
            q1 <= 1'b0;
        end
    else
        begin
            q1 <= 1'b1;
            q <= q1;
        end
    end
endmodule

```

As we can note from the waveforms that when we set is 1 the output of Q1 is zero and output of Q is 1 since we had seen in the verilog code that the reset input of the block is connected to the set input of flip flop and reset of Q1 flip-flop. Therefore the Q is set to 1 and q one is reset to zero however, when reset goes low then at the next clock input q1 goes to 1 because it input is connected to one. However the output of Flip-flop Q goes to 0 as it is following Q1 at that instant. The Q will get the new input of Q1 in the next clock pulse. So if we synthesize this code we will be needing a flip flop and it is not an example of the synchronous sequential constant.



Synthesis



Example 4

```
module dff_const4(input clk, input reset, output reg q);
reg q1;

always @ (posedge clk, posedge reset)
begin
    if(reset)
        begin
            q <= 1'b1;
            q1 <= 1'b1;
        end
    else
        begin
            q1 <= 1'b1;
            q <= q1;
        end
end
endmodule
```

Now like we can see in the waveform for dff constant 4. When reset is 1, the reset input is connected to the set input of both flip flops which sets the output of the flip flops to 1. As we can see in the waveform q and q 1 both are 1 however when reset goes to 0, q1 is connected to the one at the input therefore q1 remains one and so is q. this ia an example of sequential constant. When we synthesise this circuit we will get that we need not any flip flop to get connected as q and q 1 both are one throughout irrespective of the reset or clock which can be verified from the synthesized circuit.

File Edit View Search Terminal Help

```

Unmapped dff cell: $ DFF_P
Unmapped dff cell: $ DFF_N
`sky130_fd_sc_hd_dffbn_1_DFFSR_NNN_(.CLK_N(C), .D(D), .Q(Q), .Q_N(~Q), .RESET_B(R), .SET_B(S));
unmapped dff cell: $ DFFSR_NNP
unmapped dff cell: $ DFFSR_NPN
`sky130_fd_sc_hd_dffbn_1_DFFSR_PNN_(.CLK(C), .D(D), .Q(Q), .Q_N(~Q), .RESET_B(R), .SET_B(S));
unmapped dff cell: $ DFFSR_PNP
unmapped dff cell: $ DFFSR_PPN
unmapped dff cell: $ DFFSR PPP

```

4.1. Executing DFFLEGALIZE pass (convert FFs to types supported by the target).
Mapping DFF cells in module `dff_const4`:

```

yosys> abc -liberty .../my/lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
5. Executing ABC pass (technology mapping using ABC).
```

5.1. Extracting gate netlist of module `dff_const4` to `<abc-temp-dir>/input.blif`.
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.

```

yosys> show
6. Generating Graphviz representation of design.  

Writing dot description to `/home/jyoti/.yosys_show.dot`.
Dumping module dff const4 to page 1.
Exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot' ) 3> '/home/jyoti/.yosys_show.dot.pid'
yosys> exit
Warnings: 26 unique messages, 234 total
End of script. Logfile hash: fcac27624, CPU: user 0.49s system 0.02s, MEM: 39.04 MB peak
Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu1-18.04 -FPIIC -O8)
Time spent: 35% 2x read liberty (0 sec), 33% 1x dfllibmap (0 sec), ...
jyoti@rlworkshop-23062021-02-~:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog dff_const4.v
jyoti@rlworkshop-23062021-02-~:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out
VCD info: dumpfile tb_dff_const4.vcd open for output.
jyoti@rlworkshop-23062021-02-~:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_dff_const4.vcd

```

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time
[3000000] end time.

File Edit View Search Terminal Help

```

Warning: Found unsupported expression 'D6|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_ddrxtp_4' - skipping.
Warning: Found unsupported expression 'D6|SCE|SCDE65CE|10&|DE65|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_sedfbap_1' - skipping.
Warning: Found unsupported expression 'D6|SCE|SCDE65CE|10&|DE65|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_sedfxhp_2' - skipping.
Warning: Found unsupported expression 'D6|SCE|SCDE65CE|10&|DE65|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_sedfxtp_1' - skipping.
Warning: Found unsupported expression 'D6|SCE|SCDE65CE|10&|DE65|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_sedfxtp_2' - skipping.
Warning: Found unsupported expression 'D6|SCE|SCDE65CE|10&|DE65|SCE|SCDS65CE' in pin attribute of cell 'sky130_fd_sc_hd_sedfxtp_4' - skipping.
Final dff cell mapping:
Unmapped dff cell: $ DFF_N
`sky130_fd_sc_hd_dfftp_1_DFF_P_(.CLK(C), .D(D), .Q(0));
`sky130_fd_sc_hd_dfftn_1_DFF_NN0_(.CLK_N(C), .D(D), .Q(Q), .RESET_B(R));
unmapped dff cell: $ DFF_NN1
unmapped dff cell: $ DFF_P
unmapped dff cell: $ DFF_NP
`sky130_fd_sc_hd_dfftp_1_DFF_PN_(.CLK(C), .D(D), .Q(Q), .RESET_B(R))
`sky130_fd_sc_hd_dfftn_1_DFF_PN1_(.CLK(C), .D(D), .Q(Q), .SET_B(R))
unmapped dff cell: $ DFF_PPB
unmapped dff cell: $ DFF_PPB
`sky130_fd_sc_hd_dffbn_1_DFFSR_NNN_(.CLK_N(C), .D(D), .Q(Q), .Q_N(~Q));
unmapped dff cell: $ DFFSR_NNP
unmapped dff cell: $ DFFSR_NPN
`sky130_fd_sc_hd_dffbn_1_DFFSR_PNN_(.CLK(C), .D(D), .Q(Q), .Q_N(~Q));
unmapped dff cell: $ DFFSR_PNP
unmapped dff cell: $ DFFSR_PPN
unmapped dff cell: $ DFFSR PPP

```

4.1. Executing DFFLEGALIZE pass (convert FFs to types supported by the target).
Mapping DFF cells in module `dff_const4`:

```

yosys> abc -liberty .../my/lib/lib/sky130_fd_sc_hd_tt_025C_1v80.lib
5. Executing ABC pass (technology mapping using ABC).
```

5.1. Extracting gate netlist of module `dff_const4` to `<abc-temp-dir>/input.blif`.
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.

```

yosys> show
6. Generating Graphviz representation of design.  

Writing dot description to `/home/jyoti/.yosys_show.dot`.
Dumping module dff const4 to page 1.
Exec { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>63; exec xdot '/home/jyoti/.yosys_show.dot' ) 3> '/home/jyoti/.yosys_show.dot.pid'
yosys> exit

```

File Edit View Search Terminal Help

GTKWave - tb_dff_const4.vcd

File Edit Search Time Markers View Help

SST

Time: 1500 ns 1600 ns

Waves

clk=1
reset=1
q=q1
q1=q1
wire reset

Signals

Type Signals
Wire clk
Reg q
Reg q1
Wire reset

Filter Append Insert Replace

.yosys_show.dot - Dot Viewer

reset

1'1

BUF

q1

1'1

BUF

q

clk

dff_const4

Example 5

```
module dff_const5(input clk, input reset, output reg q);
reg q1;
```

```
always @ (posedge clk, posedge reset)
begin
```

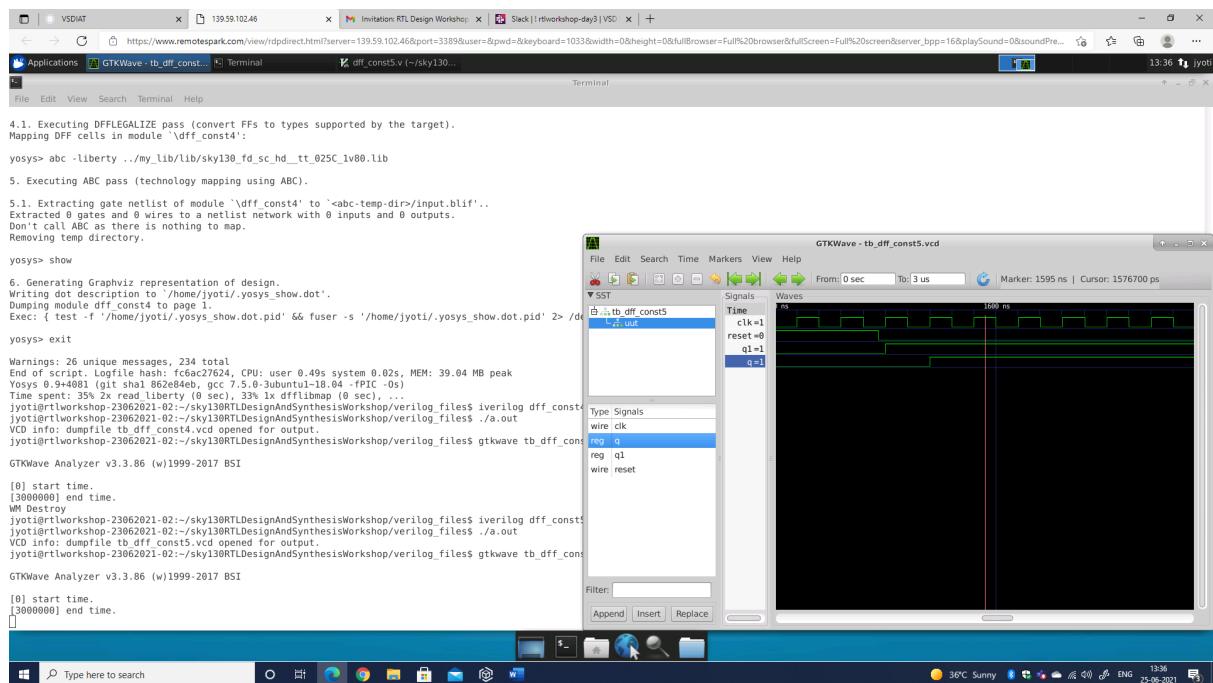
```
if(reset)
begin
    q <= 1'b0;
    q1 <= 1'b0;
end
else
```

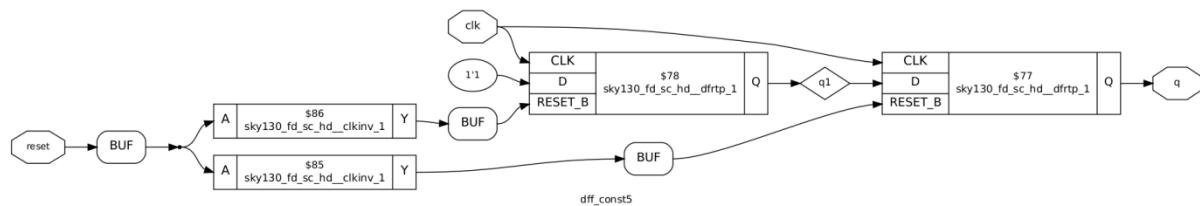
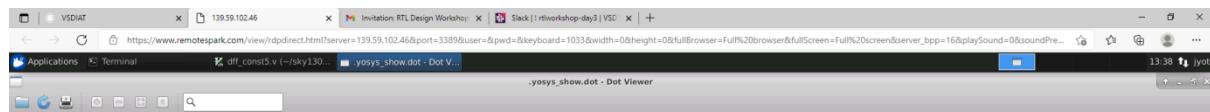
```

begin
    q1 <= 1'b1;
    q <= q1;
end
endmodule

```

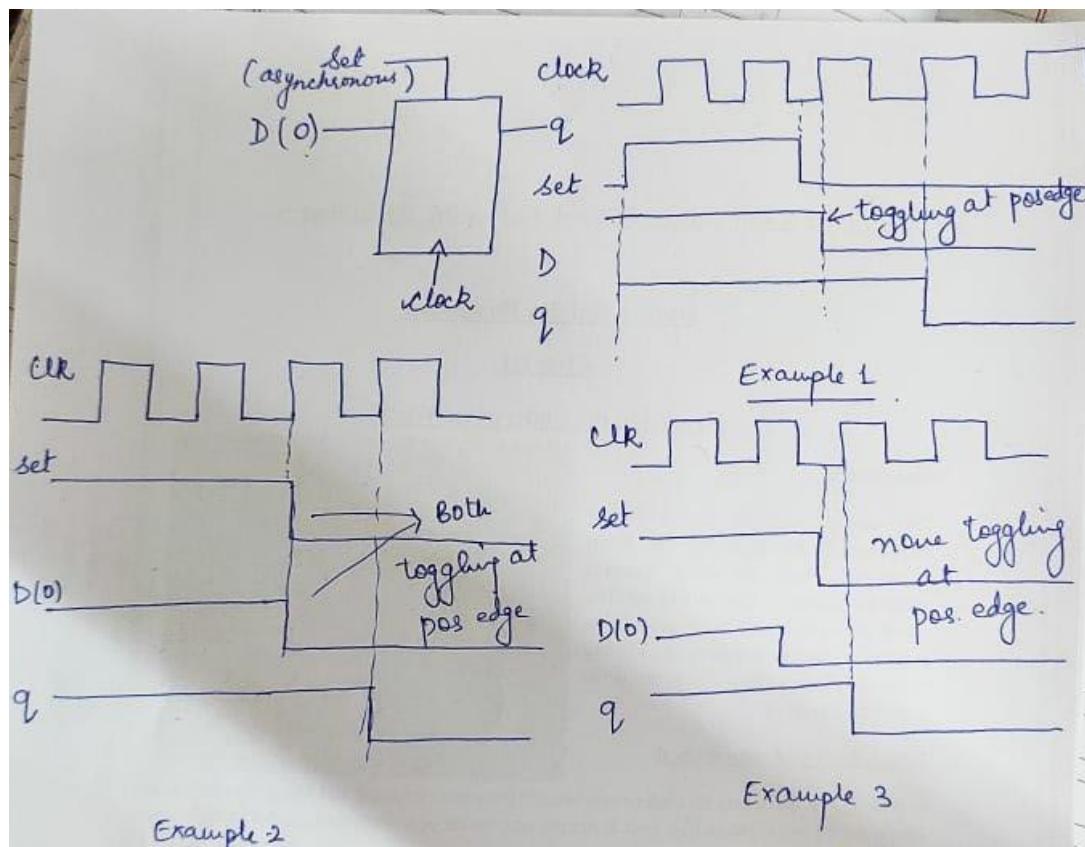
Now in this design we can see that when reset is 1 the output of both the flip flops is reset to 0 however when reset goes to zero the output of q1 goes to one because the input of Flip-flop q1 is getting an input of one during that same clock pulse the q flip flop is getting an input of q1 that is zero at that moment so it continues to be zero until next clock pulse. During next clock, it identifies q1 as one and q also goes to one at that time so we can again see this is not an example of sequential constant.





Example 6:

Here is another example which is just shown by the analysis. When set input is 1 which is asynchronous, the output is set to 1. When set goes to zero the output follows the input during the next clock pulse. As we can see in the three different combinations or possibilities shown regarding the time during which the set input is reset or the D input toggles from 1 to zero. In all the cases we can see that the output can never be said same as that of the set input.



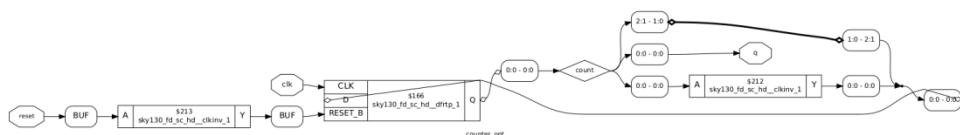
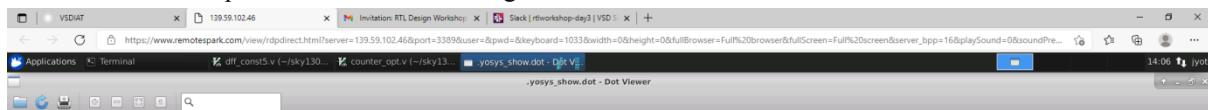
Unused output optimization

```
module counter_opt (input clk , input reset , output q);
reg [2:0] count;
assign q = count[0];

always @ (posedge clk , posedge reset)
begin
    if(reset)
        count <= 3'b000;
    else
        count <= count + 1;
end

endmodule
```

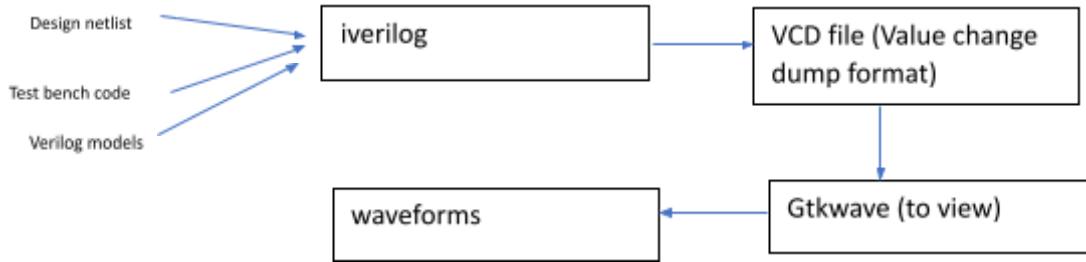
Here we are looking at the design of a counter which is three bit so it can count from zero to 7. We can see in the code that output q is only getting the LSB of the count. Rest of the two bits of count are not utilised at the output at all. So what we expect from the counter design that if it is a 3 bit counter there will be



Day 4 - GLS, blocking vs non-blocking and Synthesis-Simulation mismatch

GLS stands for gate level simulation. Gate level simulation is verification of the netlist generated by the design under test during simulation. In GLS, we run the test bench with netlist as design under test. Netlist is logically same as RTL code as netlist is the standard cell implementation of RTL code so it must align with the design.

The need of GLS is to verify the logical correctness of design after synthesis. It is also used to ensure the timing of design that the delay annotations are matched or not.



Carrying out GLS verifies that there is any synthesis simulation mismatch. This mismatch can happen due to two things:

1. Missing sensitivity list
2. Blocking versus nonblocking assignments

so, the above two methods result in the synthesis simulation mismatch because of the use of non-standard verilog coding.

Blocking and nonblocking statements in verilog

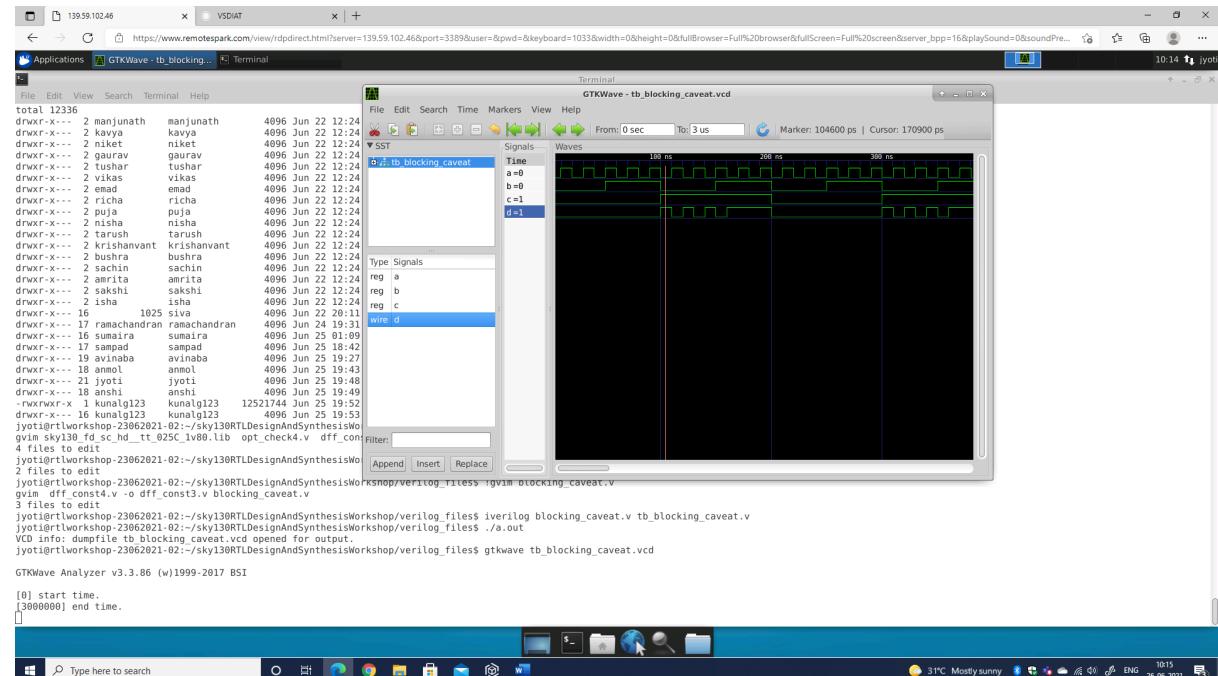
What is a blocking statement? Blocking and nonblocking is coming only to the picture when we are using always block. Inside an always block if I am using equal to assignment, we call that is blocking state. It executes in order it is written. First statement is evaluated first, and the second statement is evaluated next. The behaviour is like a C program. In case of a non-blocking it executes parallel. whenever I am using a non-blocking, the moment they enter the always block, all the RHS will be evaluated simultaneously. Evaluation order doesn't matter.

Following is the example of a blocking statement in which there is a synthesis simulation mismatch.

```

Itna tezpurmodule blocking_caveat (input a , input b , input c, output reg d);
reg x;
always @ (*)
begin
    d = x & c;
    x = a | b;
end
  
```

endmodule



As we can see in the waveforms that when a is zero and b is zero the output of or gate should be 0 and when it is ended with C equal to 1 the output of and Gate should be 0 where as we can see in the waveform that d is equal to 1 it means that it is picking up the previous value of x in which x was 1. This one value of x is ended with the 1 value of C which is giving final output D equals to 1 so this is an example of the blocking statement for synthesis and simulation mismatch. It clearly indicates that it is picking up the last value of x it means it is behaving as a flop or I can say it is latching the previous value of the output intermediate output Let Us see that what it gives after synthesis.

139.59.102.46 x VSDIAT x Gmail x Slack | New messages from Pre... x Slack | rtwkshop-day4 | VSDIAT | +

Applications GTKWave - tb_blocking... Terminal bad_mux.v (~/sky130... .yosys_show.dot - Dot V... Terminal

File Edit View Search Terminal Tabs Help Terminal

```
ABC: Scl_LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdclkp_1" without logic function.  
ABC: Scl_LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdclkp_2" without logic function.  
ABC: Scl_LibertyReadGenLib() skipped cell "sky130_fd_sc_hd_sdclkp_3" without logic function.  
ABC: Scl_LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxp_1".  
ABC: Scl_LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxp_2".  
ABC: Scl_LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxp_3".  
ABC: Scl_LibertyReadGenLib() skipped sequential cell "sky130_fd_sc_hd_sedfxp_4".  
ABC: Library "sky130_fd_sc_hd_tt_02c.lvl8" from "/home/sky130_fd_sc_hd_tt_02c.lvl8"  
ABC: Memory = 15.66 MB, Time = 0.23 sec  
ABC: Warning: Detected 9 multi-output gates (for example, "sky130_fd_sc_hd_fa...  
ABC: + strash  
ABC: + ifraig  
ABC: + tr  
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").  
ABC: + dc2  
ABC: + dretime  
ABC: + strash  
ABC: + tr  
ABC: + &dcn -f  
ABC: + &nf  
ABC: + &put  
ABC: + write_bif <abc-temp-dir>/output.blif
```

4.1.2. Re-integrating ABC results.

```
ABC RESULTS: sky130_fd_sc_hd_o2la_1 cells: 1  
ABC RESULTS: internal signals: 1  
ABC RESULTS: input signals: 3  
ABC RESULTS: output signals: 1  
Removing temp directory.
```

```
yosys> write_verilog -noattr blocking_caveat.net.v
```

5. Executing Verilog backend.

```
Dumping module 'blocking_caveat'.
```

```
yosys> show
```

6. Generating Graphviz representation of design.

```
Writing dot description to '/home/jyoti/.yosys_show.dot'.  
Dumping module blocking_caveat to page 1.  
Exec: { test -f '/home/jyoti/.yosys_show.dot.pid' && fuser -s '/home/jyoti/.yosys_show.dot.pid' 2> /dev/null; } || ( echo $>&3; exec xdot '/home/jyoti/.yosys_show.dot'; ) 3> '/home/jyoti/.yosys_show.dot.pid' &
```

```
yosys> [REDACTED]
```

139.59.102.46 x VSDIAT x Gmail x Slack | New messages from Pre... x Slack | rtwkshop-day4 | VSDIAT | +

Applications GTKWave - tb_blocking... Terminal bad_mux.v (~/sky130... .yosys_show.dot - Dot V... Terminal

File Edit View Search Terminal Tabs Help Terminal

```
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog ..//my_lib/verilog_model/primitives.v ..//my_lib/verilog_model/sky130_fd_sc_hd.v tb_bad_mux.v bad_mux.net.v  
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out  
VCD info: dumpfile tb_bad_mux.vcd opened for output.  
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_bad_mux.vcd
```

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

```
[0] start time.  
[30000000] end time.  
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog ..//my_lib/verilog_model/primitives.v ..//my_lib/verilog_model/sky130_fd_sc_hd.v tb_blocking_caveat.v  
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./a.out  
VCD info: dumpfile tb_blocking_caveat.vcd opened for output.  
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_blocking_caveat.vcd
```

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

```
[0] start time.  
[30000000] end time.  
[REDACTED]
```

139.59.102.46 x VSDIAT x Gmail x Slack | New messages from Pre... x Slack | rtwkshop-day4 | VSDIAT | +

Applications GTKWave - tb_blocking_caveat.vcd Terminal

File Edit Search Time Markers View Help

GTKWave - tb_blocking_caveat.vcd

File Edit Search Time Markers View Help

Time From: 0 sec To: 3 us Marker: 100 ns Cursor: 45300 ps

Signals Waves

tb_blocking_caveat.b1_uut

Type Signals

- wire _1
- wire _2
- wire _3
- wire _4
- wire a
- wire b
- wire c
- wire d

Filter: Append Insert Replace

Waves

11:44 26-06-2021

As we can see in the synthesis above that it has taken or and gate. It has not taken as any kind of latch in case of synthesizer output if we see however if we look at the GLS waveforms we can clearly see that there is no lashed or lag output it is just taking the current or instantaneous value that when A is Zero B is zero and she is 1 then output of and OR gate is 0 not one ok so we can see that this is a clearly case of synthesis simulation mismatch. This mismatch is caused by the blocking statement.

TERNARY OPERATOR

As we can see in the code that this is a 2 input multiplexer and for which the output is i0 when select is low and output is i1 when select is high so lets simulate it and see the waveforms. As we can see in the waveforms that output is following I0 when select is 0 and output is following I1 when select is one. So we can clearly see in the sentences that it is a 2 to 1 mux with I0 and I1 as input with select input and output as Y. The output of the simulation and GLS matches exactly.

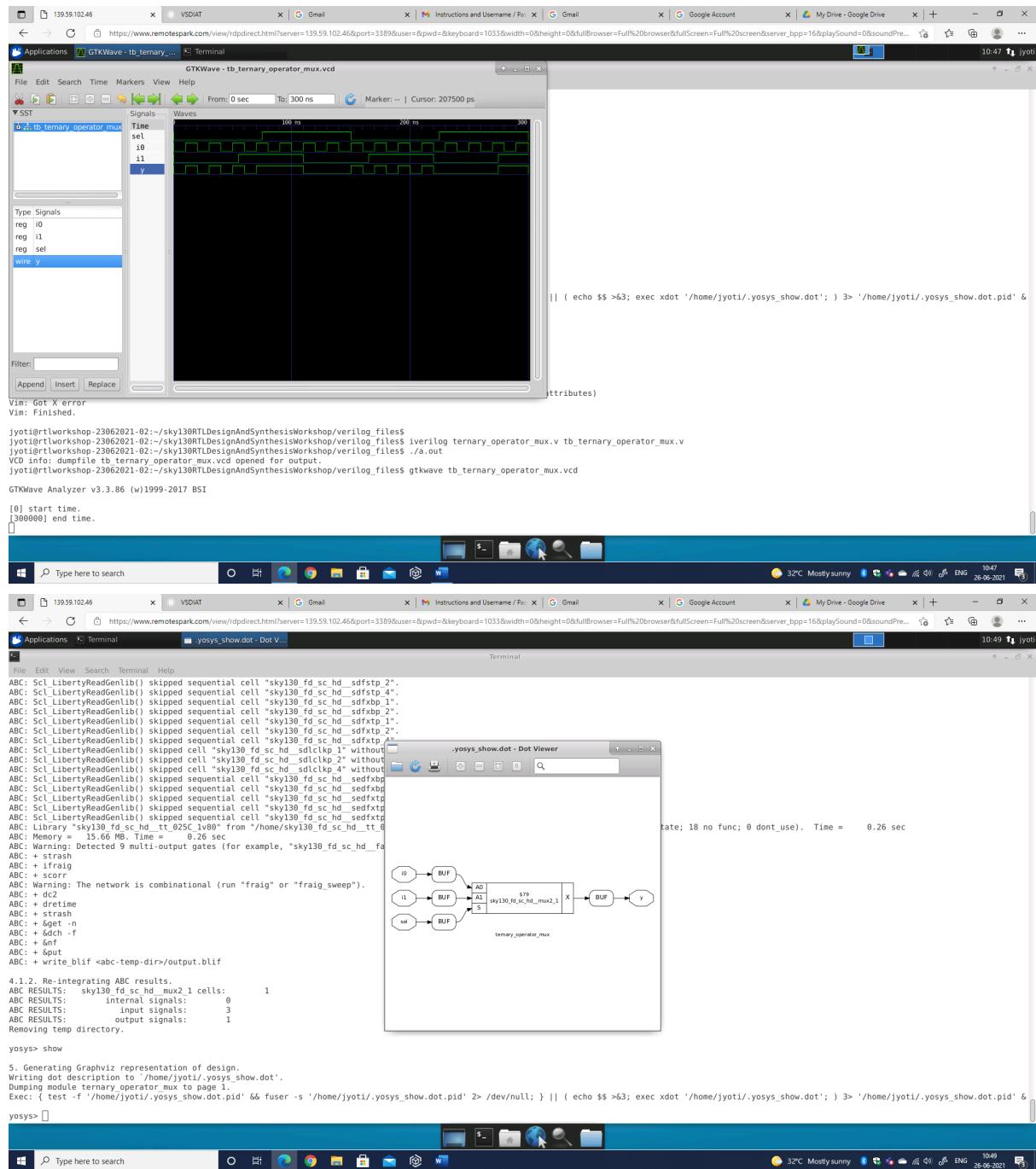
```

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

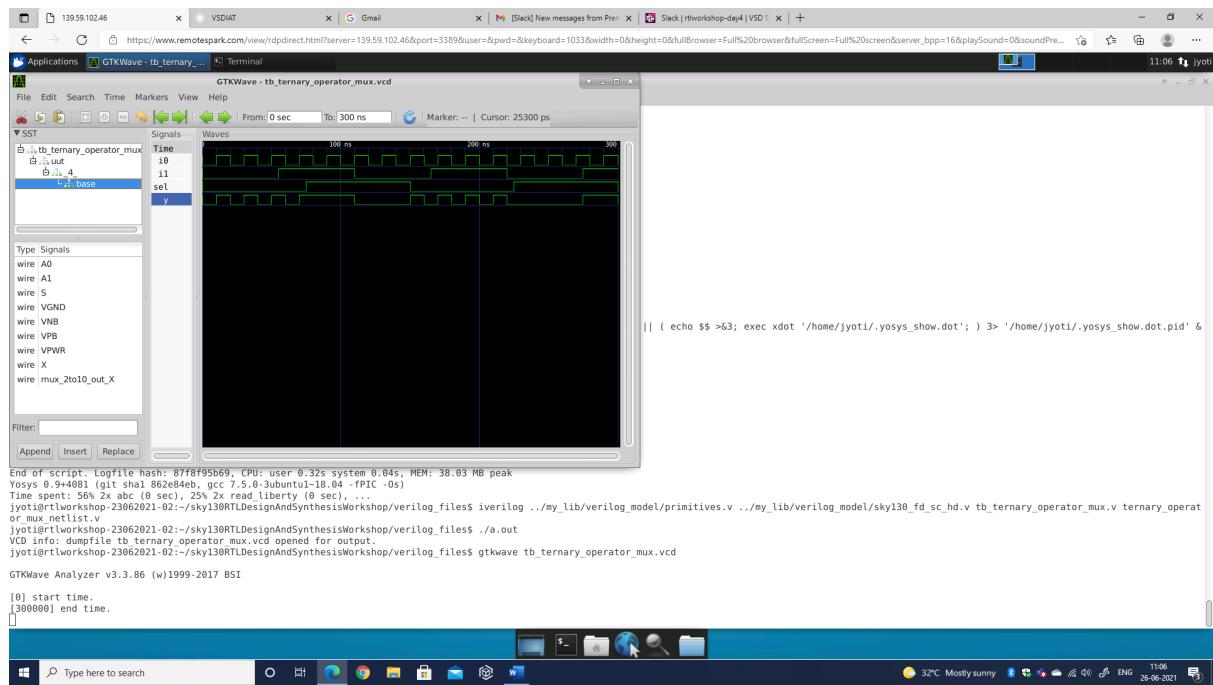
    assign y = sel?i1:i0;

endmodule

```



After GLS



BAD MUX

```

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

```

Now we have simulated the verilog model for bad mux in which we need to see that is there any simulation of synthesis mismatch . now we can see in the waveforms that although there are activities on I 0 that it is changing from 1 to0, 0 to 1 but it is not reflected at the output because there has been no activity on select input which was the only sensitivity list put in the always block. So we can clearly see that the output is not following the input because of the incorrect way of writing the sensitivity list in the verilog module.

```

ABC: Warning: The network is combinational (run "frag" or "frag_sweep").
ABC: + dc2
ABC: + dretime
ABC: + strash
ABC: + set -n
ABC: + Edcr -f
ABC: + Snf
ABC: + Sput
ABC: + write blif <abc-temp-dir>/output.blif

4.1.2. Re-integrating ABC results.
ABC RESULTS: sky130_fd_sc_hd_mux2_1 cells: 1
ABC RESULTS: internal signals: 0
ABC RESULTS: input signals: 3
ABC RESULTS: output signals: 1
Removing temp directory.

yosys> write_verilog -noattr bad_mux.net.v
5. Executing Verilog backend.
Dumping module `bad_mux'.

yosys> exit

End of script. Logfile hash: f4d2a8e90f, CPU: user 0.36s system 0.01s, MEM: 38.01 MB peak
Yosys 0.9+4081 (git sha1 862e84eb, gcc 7.5.0-3ubuntu-18.04 -FPIC -O5)
Time spent: 52% 2x abc (0 sec), 23% 2x read liberty (0 sec), ...
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_bad_mux

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[300000] end time.
WM Destroyed.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog bad_mux.v
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./out
VCD info: dumpfile tb_bad_mux.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_bad_mux

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[300000] end time.

```

Now let's see how we get the output after synthesizing and generating GLS. So, we can see from the GLS waveform that output is following I not corresponding to whatever is a select input at that time which if we compared with the simulation output was missing so this output after synthesis is correct we can see here that the simulation and synthesis waveforms are not matching

```

jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog ..//my_lib/verilog_model/primitives.v ..//my_lib/verilog_model/sky130_fd_sc_hd.v tb_bad_mux.v bad_mux_net.v
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ ./out
VCD info: dumpfile tb_bad_mux.vcd opened for output.
jyoti@rtlworkshop-23062021-02:~/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_bad_mux.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

[0] start time.
[300000] end time.


```

Day 5 – If, case for loop and for generate

In this session we are going to see about if and case statement and there is a danger with the case statement that also we will see. “If” is mainly used to create priority logic.

The syntax will be as shown below. if <condition> the write the code between begin and end. so clearly this if <condition> portion has a Priority.

If <condition>

---statement

Elsif <condition2>

--- statement

Elseif <condition3>

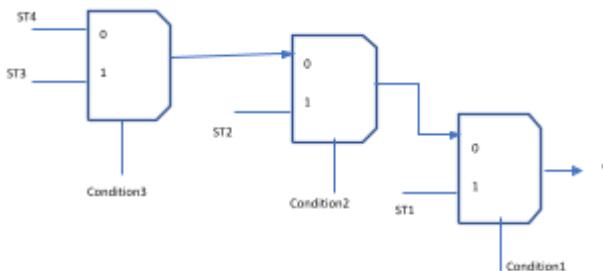
--- statement

Else

--- statements

What will this mean in terms of hardware? If condition here gets the highest priority because if this condition matches, all other conditions will not be executed. When first condition is not met condition 2 will be evaluated. Only when these two are not met condition 3 will be evaluated and further only when all these are not met then else will be evaluated this clearly create a Priority logic.

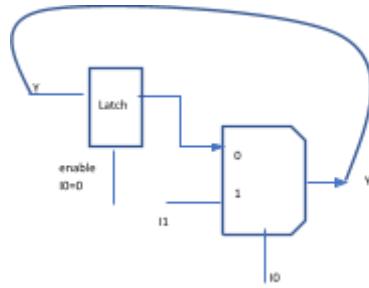
Let us look at the hardware.



Now let us look at an incomplete if statement. As given in the code below we have written that if I0 is one then why gets the value of I1 however there is no else statement specified. Now let us see what hardware is expected out of this program.

INCOMPLETE IF

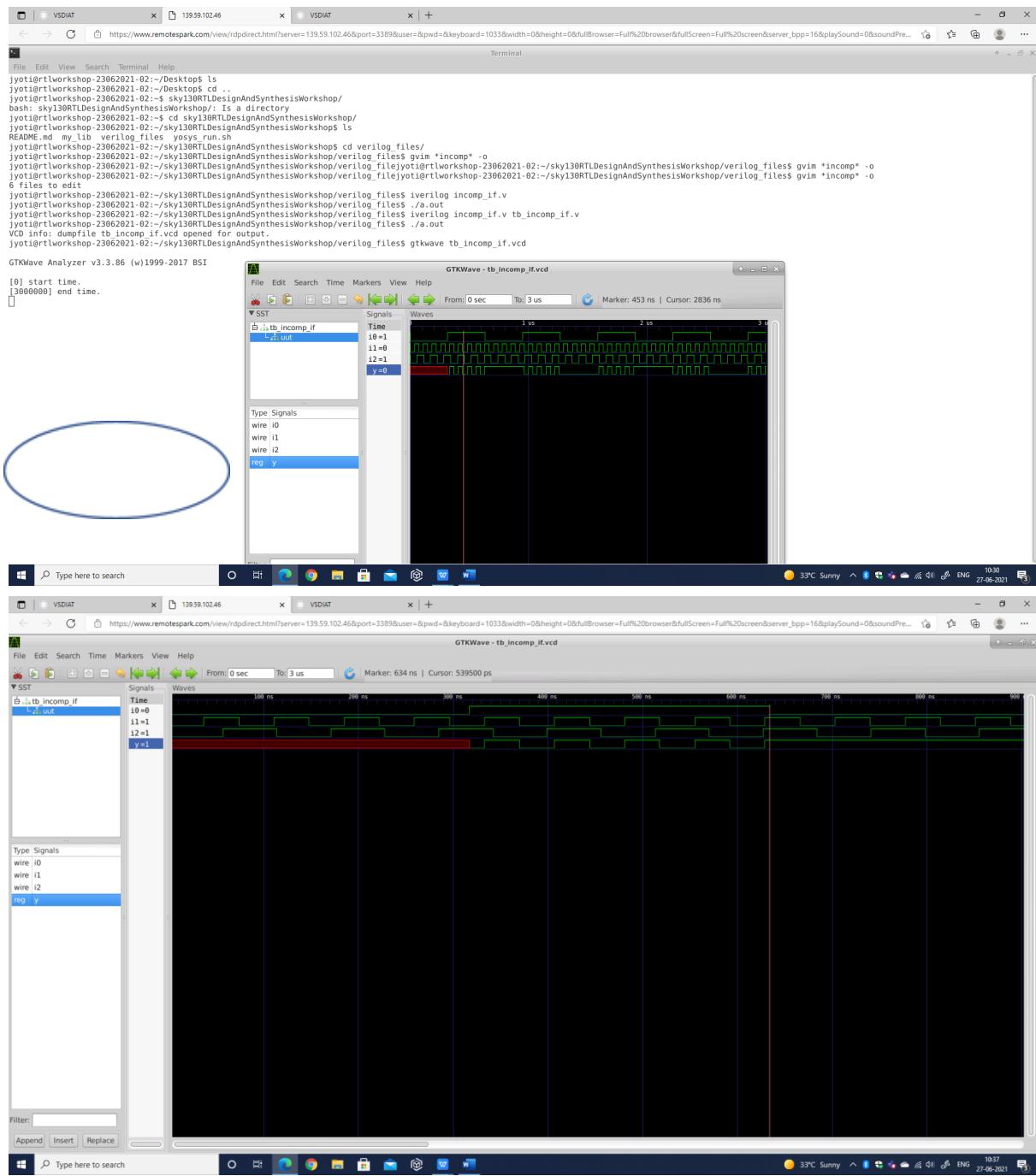
```
module incomp_if(input i0 , input i1 , input i2 , output reg y);
always @ (*)
begin
  if(i0)
    y <= i1;
end
endmodule
```



So as we can see here in the hardware that seems else statement is not specified output why is going to retain its old value it means it is going to behave as a latch. So this is a combination loop. To avoid this what the tool will do is it will put a Latch. This is the inferred latch. The tool will inform it as a latch and connect here. So whatever value is stored, will be driven here this is called inferred latch coming because of incomplete if.

How we will write the counter? A count get count + 1. If there is no enable, it should latch on to the previous value. here if no enable a counter should latch on to the previous value. So this is perfectly fine. this is the intended behaviour however in previous case this is not the intended behaviour in a combinational circuit I cannot have a inferred latch in a combinational circuit.

Now let us simulate this incomplete if statement and see what wave forms do we get. As we can see, When i_0 is 1 the output y is following i_1 one input but When our i_0 is going low , the output y is latching onto some previous value, it's either 1 or 0 permanently. there is no change absolutely . thus clearly the latching action is taking place. Lets us see in synthesis.



Screenshot of a terminal window showing synthesis logs for 'incomp_if' module:

```

3.22.1. Extracting gate netlist of module '\incomp_if' to '<abc-temp-dir>/input.blif'..
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.

3.23. Executing OPT pass (performing simple optimizations).
3.23.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module incomp_if.

3.23.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module '\incomp_if'.
Removed a total of 0 cells.

3.23.3. Executing OPT_DFF pass (perform DFF optimizations).

3.23.4. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \incomp_if.

3.23.5. Finished fast OPT passes.

3.24. Executing HIERARCHY pass (managing design hierarchy).
3.24.1. Analyzing design hierarchy..
Top module: \incomp_if
3.24.2. Analyzing design hierarchy..
Top module: \incomp_if
Removed 0 unused modules.

3.25. Printing statistics.

== incomp_if ==
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
$_DLATCH_P_ 1

3.26. Executing CHECK pass (checking for obvious problems).
Checking module incomp_if...
Found and reported 0 problems.

yosys>

```

Screenshot of a Windows taskbar showing the 'yosys show dot - Dot Viewer' application running.

Diagram illustrating the synthesized logic for the 'incomp_if' module:

incomp_if

There we can see in synthesis that we are getting a latch instead of a multiplexer which we were aiming to design when we were writing the code. So it is clearly a case of incomplete if statement which needs to be taken care.

Similarly in incomplete if2:

```

module incomp_if2 (input i0 , input i1 , input i2 , input i3, output reg y);
always @(*)
begin
  if(i0)
    y <= i1;

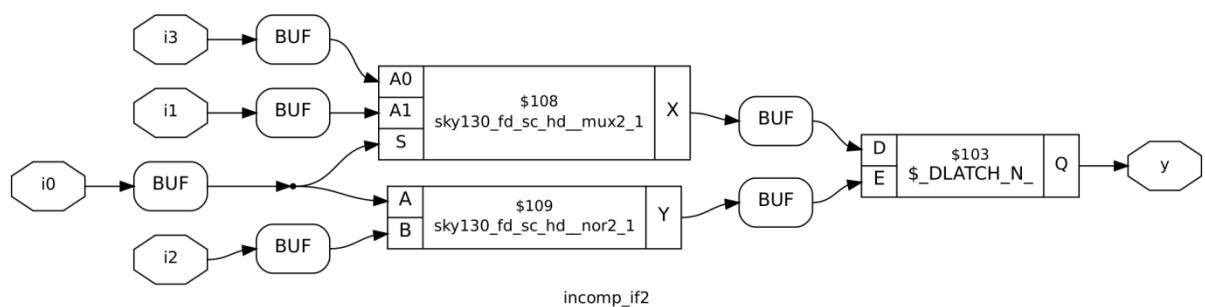
```

```
else if(i2)
```

```
    y <= i3;
```

```
end
```

```
endmodule
```



Is another example of incomplete if statement we can see in the waveforms that when I0 is one Y is following I1 when I0 is 0 and I2 is 1 then output is following I3 But when both I0 and I2 is 0 it is holding onto its previous value like a latch.

So as expected we can see in the circuit that is implemented or synthesized that there is a latch having enabled by the combination of I₀ and I₂ both are going into the NOR gate so it means the latch is an active low enabled and whenever the latch is enabled the input is a combination of I₀, I₁ and I₃ that is going through a mux so this is what is expected.

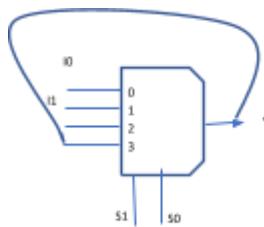
INCOMPLETE CASE STATEMENT

if and case are used Inside always block, so the rule in verilog is whatever variable you are trying to assign in case or if should be a register variable. case statement is also going to be like a mux. so effectively what it is going to do. What are the important case caveats, let us look at that. Case statement is very very dangerous to use without having understanding.

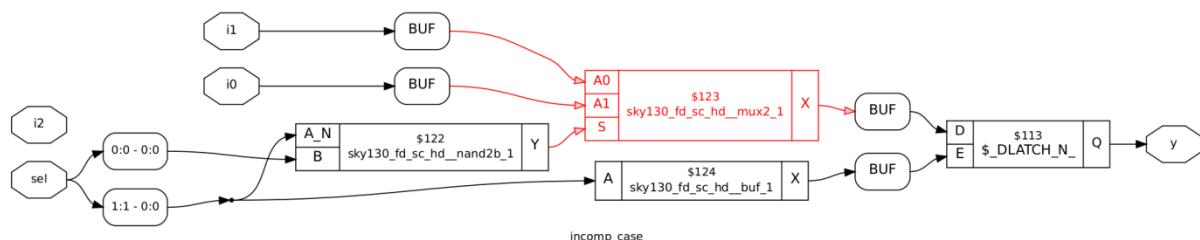
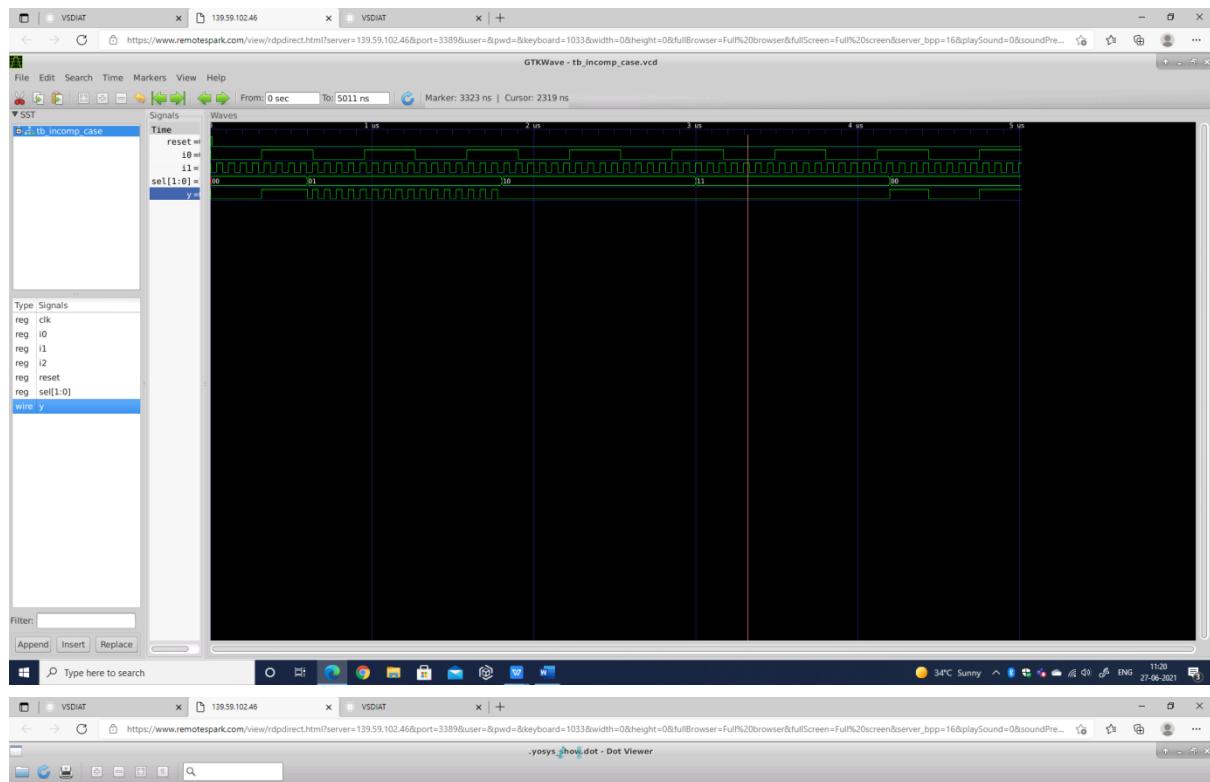
1. Incomplete case statement : Incomplete case statement will lead to inferred latches. For example now when select is zero you are giving a C₁ and when select is one you are giving C₂. What should happen when select is 3 or 4, it is not told. What is going to cause is an incomplete case Or inferred latches. This is very dangerous so to avoid incomplete cases the solution is code case with default.

Below is shown the example of an incomplete case statement. What is expected out of here is a four input marks with four inputs, two select lines and an output.

```
module incomp_case (input i0 , input i1 , input i2 , input [1:0] sel, output reg y);
always @ (*)
begin
    case(sel)
        2'b00 : y = i0;
        2'b01 : y = i1;
    endcase
end
endmodule
```



So we can see in the diagram that the inputs for the possible values of select input as 10 and 11 is not specified. So it will try to retain its old value and as we have looked in an incomplete if statement it will be inferring this incomplete information in case statement also as a latch. This weekend easily verify through simulation and synthesis as shown in the snapshots below.



SOLUTION TO INCOMPLETE CASE:

There is a solution to the incomplete case statement that if we use the default value for the left out options in case of case statement then there will be no latches. This has been shown by modifying the code of the cases statement written below using default and then subsequent to that are shown the correct wave forms and the synthesis circuit. So we can see that there are no more latches after synthesis in the circuit.

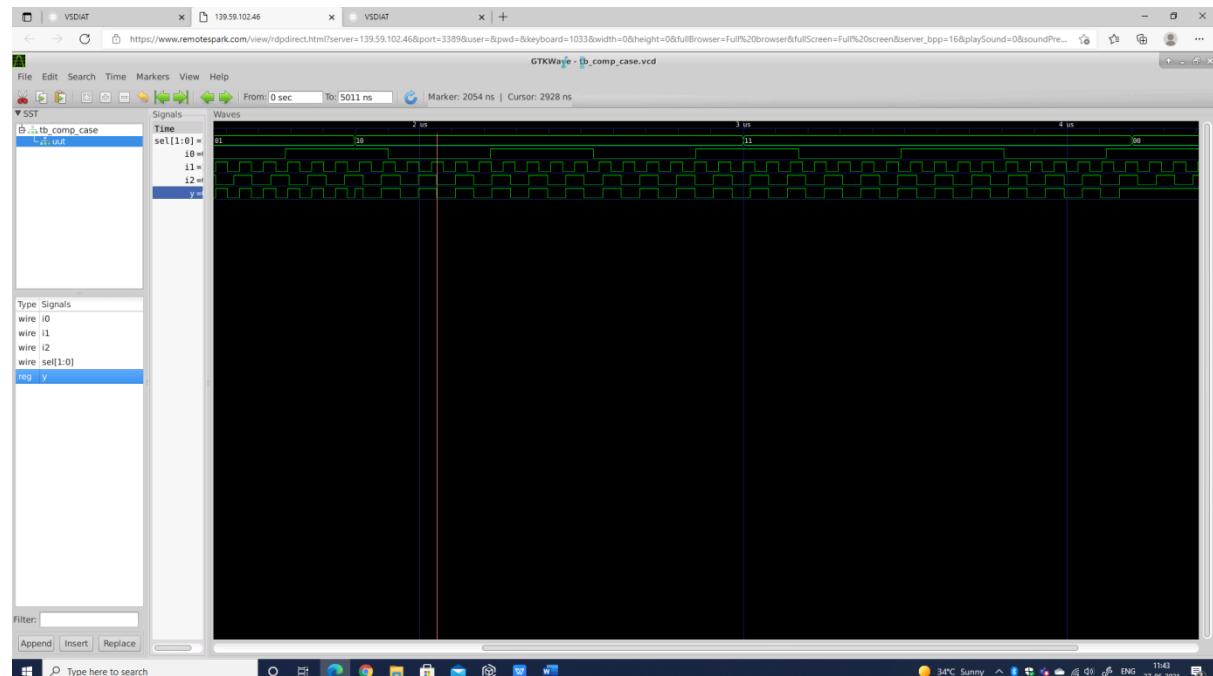
```
module comp_case (input i0 , input i1 , input i2 , input [1:0] sel, output reg y);
```

```
always @(*)
```

```

begin
    case(sel)
        2'b00 : y = i0;
        2'b01 : y = i1;
        default : y = i2;
    endcase
end
endmodule

```



```

8.23.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module comp_case.

8.23.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells or registers in module `comp_case'.
Removed a total of 0 cells.

8.23.3. Executing OPT_DFF pass (perform DFF optimizations).

8.23.4. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module `comp_case'.
Removed 0 unused cells and 13 unused wires.
<suppressed -i debug messages>

8.23.5. Finished fast OPT passes.

8.24. Executing HIERARCHY pass (managing design hierarchy).

8.24.1. Analyzing design hierarchy..
Top module: `comp_case

8.24.2. Analyzing design hierarchy..
Top module: `comp_case
Removed 0 unused modules.

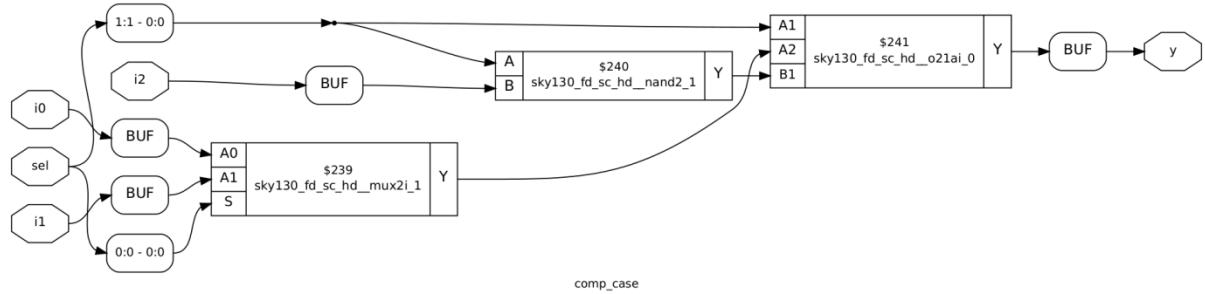
8.25. Printing statistics.

==== comp_case ====
Number of wires: 11
Number of wire bits: 12
Number of public wires: 5
Number of public wire bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 7
$_ANDNOT_
$_AND_
$_MUX_
$_ORNOT_
$_OR_
2
1
1
2

8.26. Executing CHECK pass (checking for obvious problems).
Checking module comp_case...
Found and reported 0 problems.

yosys> 

```



PARTIAL ASSIGNMENT: CASE STATEMENT

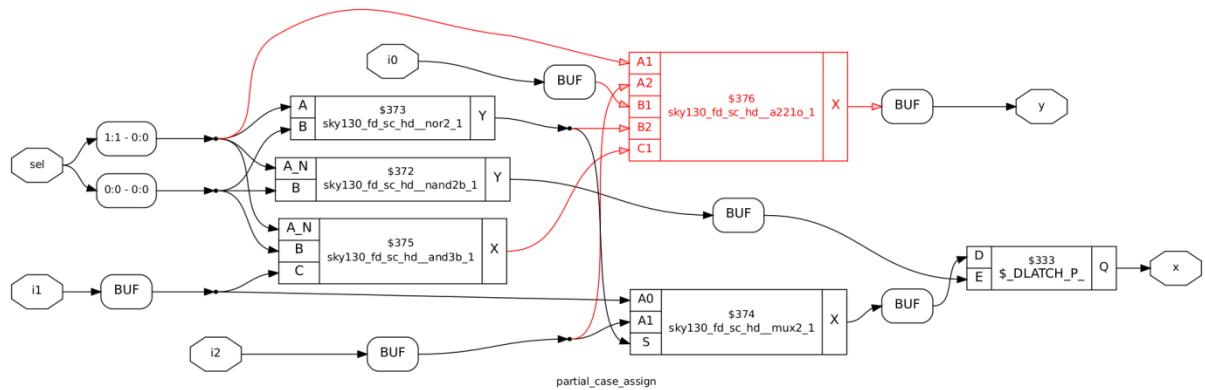
Now there are two output called X and Y so that will be too muxes. So we think that we have added default statement so everything is done correctly but that doesn't guarantee anything. When select is 0 0 x is getting i2 and Y is getting i0. When select is 0 1 is getting we see that y is getting i1 but nothing is assigned for x. So during this case of selecting 0 1, X will be latched however for rest of the two input combinations it is mentioned that X is getting i1 and why is getting i2. So, this is going to create and inferred latch.

```
module partial_case_assign (input i0 , input i1 , input i2 , input [1:0] sel, output reg y , output reg x);
always @ (*)
begin
    case(sel)
        2'b00 : begin
            y = i0;
            x = i2;
        end
        2'b01 : y = i1;
        default : begin
            x = i1;
            y = i2;
        end
    endcase
end
endmodule
```

```

endcase
end
endmodule

```



OVERLAPPING CONDITIONS

And the last caveat in case and if statement is if we compare If in case statement then there is clearly a Priority list in if statement like if statement condition is true it is not going to go to the rest of the conditions and it will come out of it however if this condition is not met then it is going to the next else if clause or else clause only one portion of the if statement will be executed and whichever is executed it will leave rest of the cases and come out of the if statement. However, such is not the case in case of case statement. Like we can see in the given example if we have overlapping case such as in option 3 and 4 since case statement goes sequentially when it matches the option 10 for the select input it will execute those statements and then it will also go to the next option which also matches because of it is 1x, so clearly it is going to create problem for the hardware. So, in case statement we should not have any overlapping case.

So let us look at the code below. This is a case when we are having overlapping options for case statement. Now we're going to see the output of the simulation then synthesis and GLS simulation also.

```

module bad_case (input i0 , input i1, input i2, input i3 , input [1:0] sel, output reg y);
always @(*)
begin
    case(sel)
        2'b00: y = i0;

```

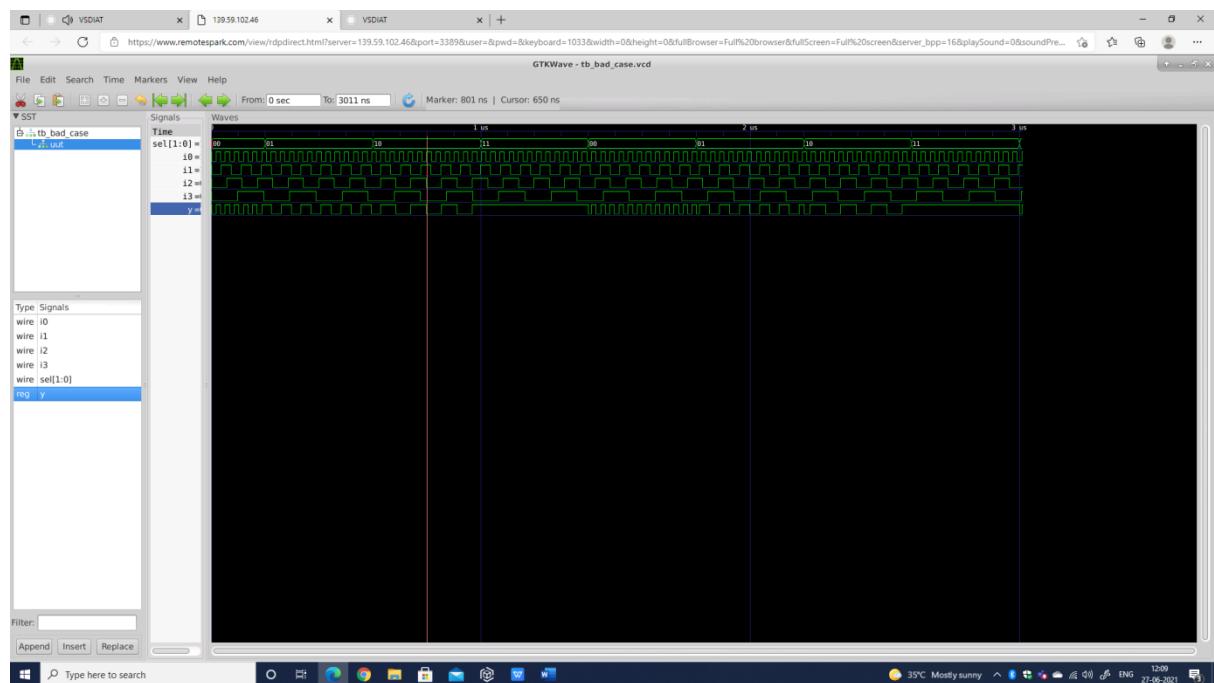
```

2'b01: y = i1;
2'b10: y = i2;
2'b1?: y = i3;
//2'b11: y = i3;

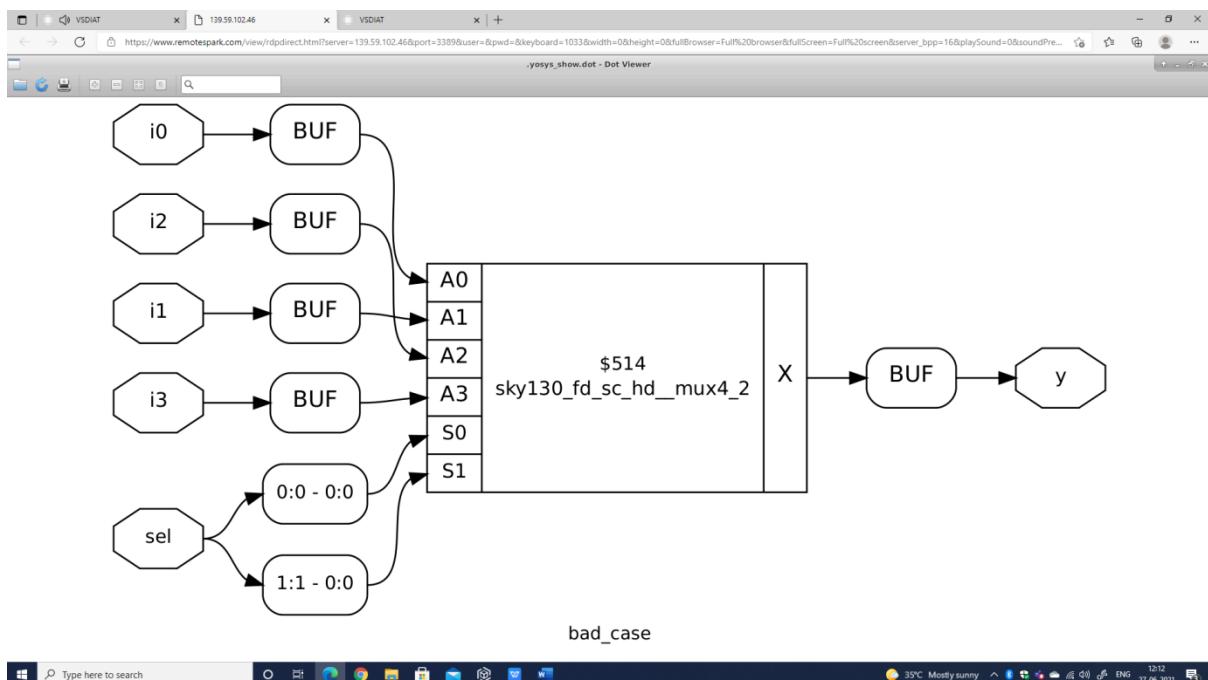
endcase
end

```

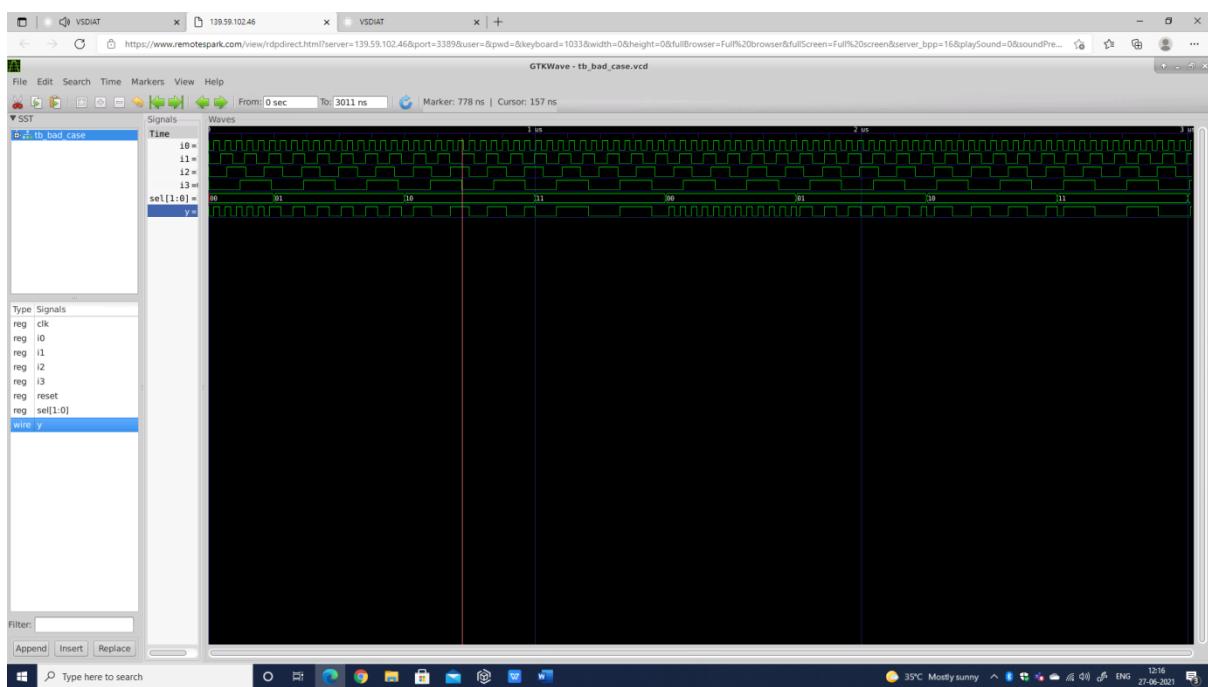
endmodule



As we can see in the wave forms above that when input is 10 the output is following I 2 but when input is 11 it is not following any input what is randomly held on to logic 1. However when we look at the synthesis details, there are no latches. it means there are no inferred latches and when we look at the GLS waveform we can see that output is correctly getting input I3 when select input is 11. It means that this is a case of simulation synthesis mismatch.



GLS



There is synthesis and simulation mismatch.