DAY 1: INTRO TO VERILOG AND RTL DESIGN AND SYNTHESIS
INTRODUCTION TO OPEN-SOURCE SIMULATOR IVERILOG

SKY130RTL D1SK1 L1 Introduction to iverilog design test bench

What is a simulator

- Tool used for simulating the design
- Rtl design is checked for adherence to spec by simulting the design
- We will use Iverilog

What is a design

- Actual verilog code or set of verilog codes which has intended functionality to meet desired specifications

Testbench

- Setup to apply stimulus (test_vectors) to design and and check its functionality

How simulator works

- Looks for changes in ip signal
- Upon change in ip, op is evaluated
  - If no change in ip, no change in op
- Simulator is looking for change in values of ip

Design may have 1 or more primary inputs, 1 or more primary outputs

TB doesn't have a primary input or primary outputs

IVERILOG BASED SIMULATION FLOW:

We have a design, a tb, we apply both of them to iverilog

- Simulator only looks for changes in input(any simulator for that matter)
- Output of the simulator will be a VCD FORMAT- Value change dump
  - To view this VCD, we use GTKwave which can be used to view the waveform to verify the functionality of the design
-

LABS USING IVERILOG AND GTKWAVE

SKY130RTL D1SK2 L1 Lab1 introduction to lab

Mkdir VLSI
Cd VLSI
VLSI# ls
Git clone https://github.com/kunalg123/vsdflow.git
Cd vsdflow/
Vsdflow# ls
Cd ..
 Git clone

/home/vsd/VLSI# (in this directory do the next git cloning, go to kunalg123 github, go to sky130rtldesignandsynthesisworkshop repo, copy the link)

/home/vsd/VLSI# git clone "enter the link"

- Upon hitting enter, a directory will be created

Look into the created directory

- Cd sky130rtldesignandsynthesisworkshop
- Ls (go into the dir)
    - We will see some files
    - "Cd my_lib" this mylib is going to contain all library files needed, it will have 2 folders which will have a lib and verilog model, lib has sky130 standard cell library which we will use for synthessis
    - The folder called verilog model creates all standard cells verilog models for all standard cells presnet inside the.lib
        - § This folder consists all our lab experiments, verilog source files, tb files which we will use in our labs.

SKY130RTL D1SK2 L2 Lab2 Introduction iverilog gtkwave part1

Open the verilog_files dir, here we will have lots of file names, and lots of files start with tb_some_name, hence for every file, there is an associated test bench, we have a bad latch, and a tb_bad_latch, there is a one-to one correspondence

Now we can load any design to i verilog using the command
  “Iverilog *designname*”

akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ iverilog goodmux.v tb_good_mux.v

Ls

A "a.out" file will get created here, execute it using "./a.out", when that is done, a vcd will get created

The output of a simulator is a vcd file, which we can load into our simulator, gtkwave

```
akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gtkwave tb_good_mux.vcd
```

Gtkwave will now open, there will be a tstbench, within which we will have uut- unit under test. The inputs can be dragged and dropped into the signals panel.
Timescale of the simulation is 1ps and 300ns is the time, which is very large, hence entire observation cannot be viewed in single obs without zoomfit, if we want to zoom into a particular region, we can zoom in,

The forward arrow looks for the forward transitions and vice versa.

We have implemented a mux in our design, 2x1.

This is how the design is loaded and its functionality is tested

SKY130RTL D1SK2 L3 Lab2 Introduction iverilog gtkwave part2

We can look into any particular file in the dir using the "gvim _filename_" command

```
sudo apt install vim-gtk3
```

```
akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesignAndSynthesisWorkshop/verilog_files$ gvim tb_good_mux.v -o good_mux.v
```

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

```
`timescale 1ns / 1ps
module tb_good_mux;
    // Inputs
    reg i0,i1,sel;
    // Outputs
    wire y;

        // Instantiate the Unit Under Test (UUT)
    good_mux uut (
        .sel(sel),
        .i0(i0),
        .i1(i1),
        .y(y)
    );

    initial begin
    $dumpfile("tb_good_mux.vcd");
    $dumpvars(0,tb_good_mux);
    // Initialize Inputs
    sel = 0;
    i0 = 0;
    i1 = 0;
    #300 $finish;
    end

always #75 sel = ~sel;
always #10 i0 = ~i0;
always #55 i1 = ~i1;
Endmodule
```

INTRODUCTION TO YOSYS AND LOGIC SYNTHESIS
6- SKY130RTL D1SK3 L1
INTRODUCTION TO YOSYS

SYNTHESISER

Tool used to convert RTL to Netlist

Yosys is the synthesiser used in this course

We have a design, a .lib, we will apply this to yosys, and get the netlist through yosys

Netlist is the representation of design in terms of standard cells present in .lib

HOW DO WE VERIFY IF OUR SYNTHESIS IS CORRECT!!?

How do we see if the tool has not damaged the design/changed the design during the process of synthesis,

We have a netlist, a tb, we have a tb, we feed these to iverilog, and get the vcd op from iverilog and load the waveform in gtkwave and we will be able to observe the stimulus

This stimulus should be same as the op observed during the RTL simulation for synthesis to be successful.

The set of inputs/primary inputs will remain same bw rtl design and synthesised netlist
    Hence the same tb can be used as rtl tb as the inputs and outputs remain unchanged

7- SKY130RTL D1SK3 L2
INTRODUCTION TO LOGIC SYNTHESIS PART 1

RTL DESIGN
- Behavioral representation of the required representation
    - Specification in behavioral form written in verilog hdl

I dont want a code, i want a hw circuit, how do we map these 2? The answer is SYNTHESIS

SYNTHESIS
- RTL to gate level translation is known as synthesis
- Design is converted into gates and the connections are made bw the gates
- This is given out as a file known as a “NETLIST”

We have an RTL, we have a front end library, we will take these through synthesis and get an output known as the “NETLIST”

WHAT IS .LIB
- Collection of logical modules, contains all basic gates such as And,or not etc
- It may have different flavours of the same gate
    - Like 2 ip and gate,
        - Slow fast and medium
    - 3 ip and gate
        - Slow fast and medium
    - 4 ip and gate
        - Slow fast and medium
- Collection of all different flavours of functionalities of standard cells

May not be exhaustive of all components, but can implement any and all boolean logic functions

.lib contains std.cells to implement any boolean logic functionalities

WHY DO WE NEED DIFFERENT flavours OF GATES?

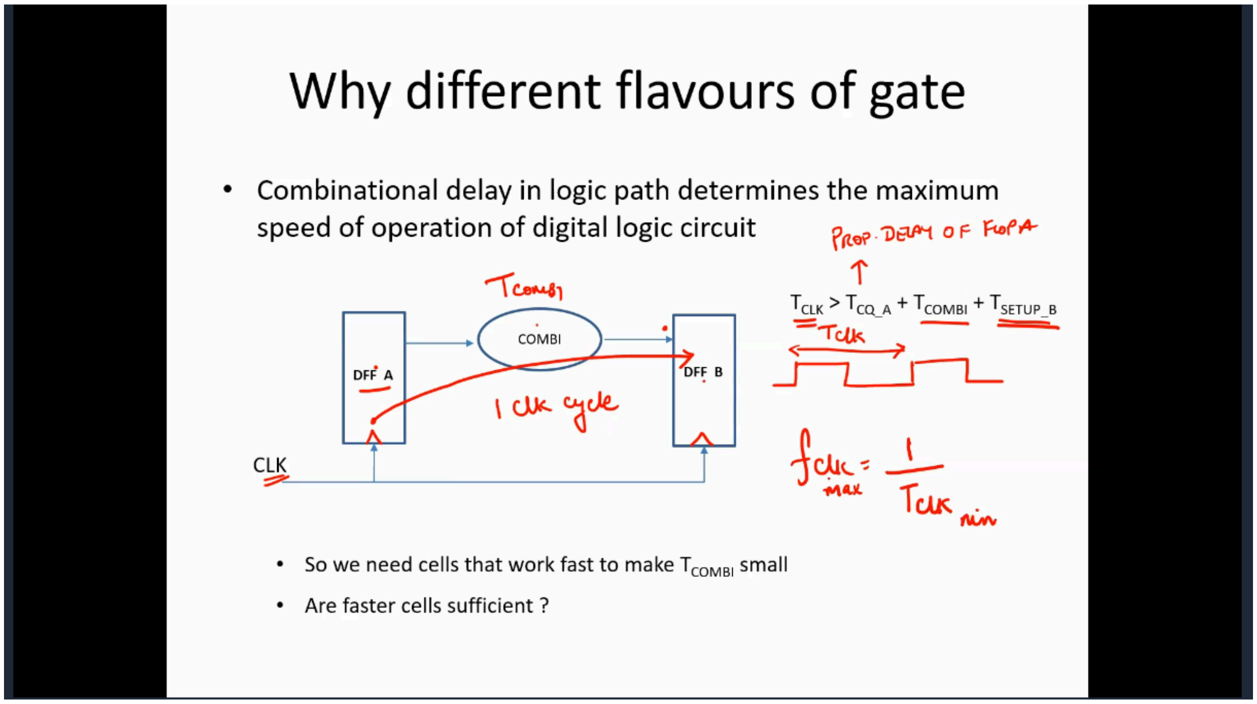
Combinational delay in logic path determines the maximum speed of operation of digital logic circuit

If we have ffa and ffb connected through a comb. Ckt, what is the max clock rate we can apply here? How big should t clock be?

It should be big enough, such that the time durationfor data to travel from flop a to flop b is achieved in 1 clock cycle

Hence propagation delay of flop is propagation delay of flop a ie. Tcq_a +propagation delay of comb. Ckt ie Tcombi + setup time(the time required for the flop to get ready to accept incoming data) ie Tsetup_b

This will be the minimum time period of the clock , hence 1/Tclkmin will be maximum frequency of the clock



Why different flavours of gate

- Combinational delay in logic path determines the maximum speed of operation of digital logic circuit

$$T_{CLK} > T_{CQ\_A} + T_{COMBI} + T_{SETUP\_B}$$

$$f_{clk_{max}} = \frac{1}{T_{clk_{min}}}$$

- So we need cells that work fast to make T_{COMBI} small
- Are faster cells sufficient ?

Minimum the clk period, maximum the clock speed, hence better the performance

Hence delays should be as less as possible to enhance performance, hence we need our cells to work very fast

BUT IF FASTER CELLS ARE TRULY SUFFICIENT, WHY DO WE NEED SLOW AND MEDIUM CELLS IN OUR LIBRARY FILE!?

8- SKY130RTL D1SK3 L3
INTRODUCTION TO LOGIC SYNTHESIS PART 2

WHY DO WE NEED SLOW CELLS?

Like we looked at setup, there is something called hold

Whatever a is launching in first cycle, we want b to capture the data in the next cycle, hence we dont want their catch and throw cycle to not overlap

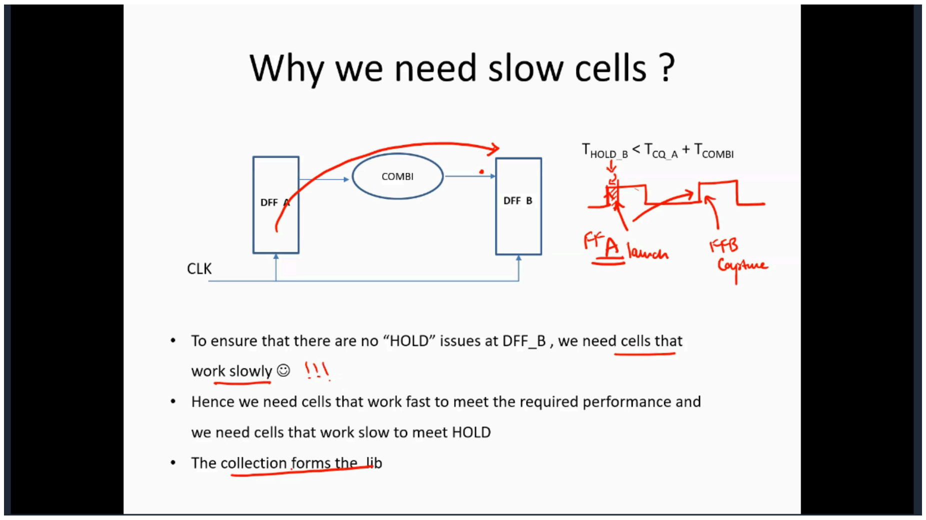
Hence we need some minimum delay between the 2

The fastest change that can happen at b should be after the minimum required time so that the previous clock cycle data is captured

I want this circuit to work fast, but it should not be too fast to cause hold faults, hence we want some cells to work slowly too

This creates a contradictory condition, i want some cells to work fast to meet setup condition and some to work slow to create hold condition

Hence we need a variety of cells to meet these requirements, and this is why we have multiple sets of cells in .lib file.



Why we need slow cells ?

$T_{HOLD\_B} < T_{CQ\_A} + T_{COMBI}$

- To ensure that there are no "HOLD" issues at DFF_B , we need cells that work slowly ☺ !!!
- Hence we need cells that work fast to meet the required performance and we need cells that work slow to meet HOLD
- The collection forms the .lib

FASTER CELLS VS SLOWER CELLS

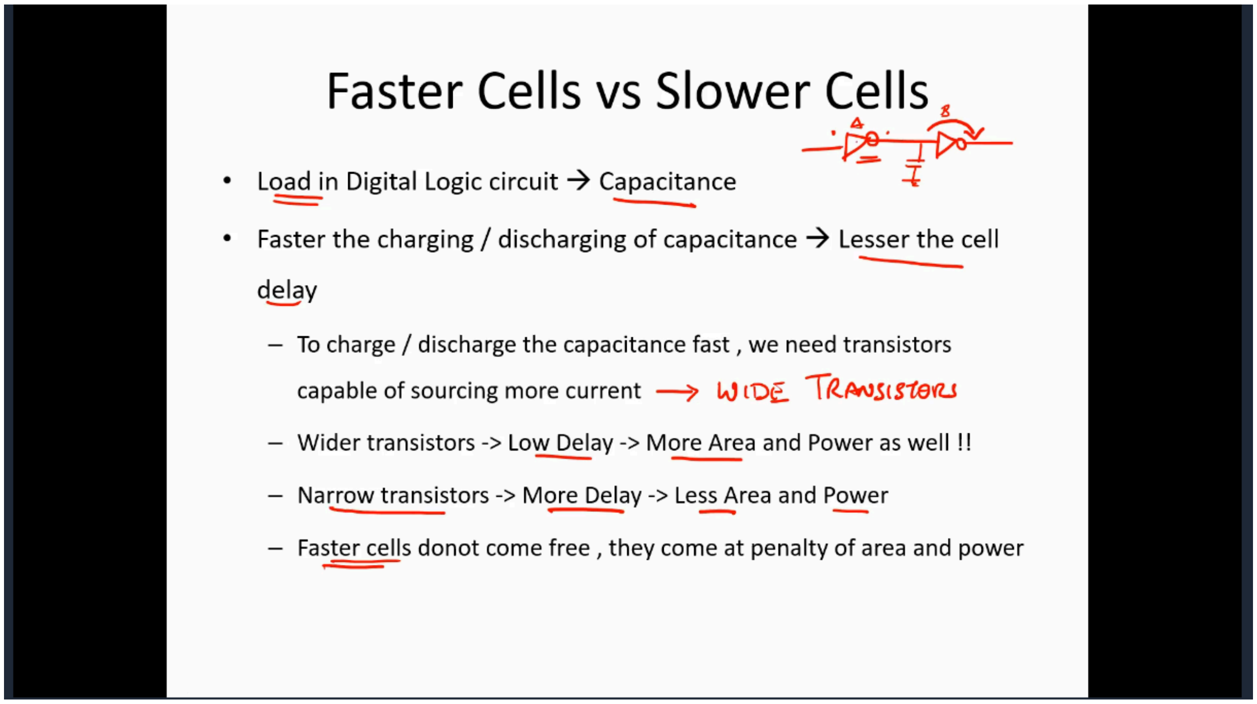
Load in digital circuit- capacitance

Faster the charging/discharging of capacitance - lesser the cell delay

To charge/discharge the capacitance fast, we need transistors capable of sourcing more current, and for sourcing more current, we will need wider transistors

Wider transistors - low delay - more area and power as well!!!
Narrow transistors - more delay - less area and power

THEREFORE FASTER CELLS DO NOT COME FREE, THEY COME AT PENALTY OF AREA AND POWER!!



Faster Cells vs Slower Cells

- Load in Digital Logic circuit → Capacitance
- Faster the charging / discharging of capacitance → Lesser the cell delay
  - To charge / discharge the capacitance fast , we need transistors capable of sourcing more current ⟶ WIDE TRANSISTORS
  - Wider transistors -> Low Delay -> More Area and Power as well !!
  - Narrow transistors -> More Delay -> Less Area and Power
  - Faster cells donot come free , they come at penalty of area and power

IF capacitance is large, b will drive a slowly, and vice versa

SELECTION OF CELLS

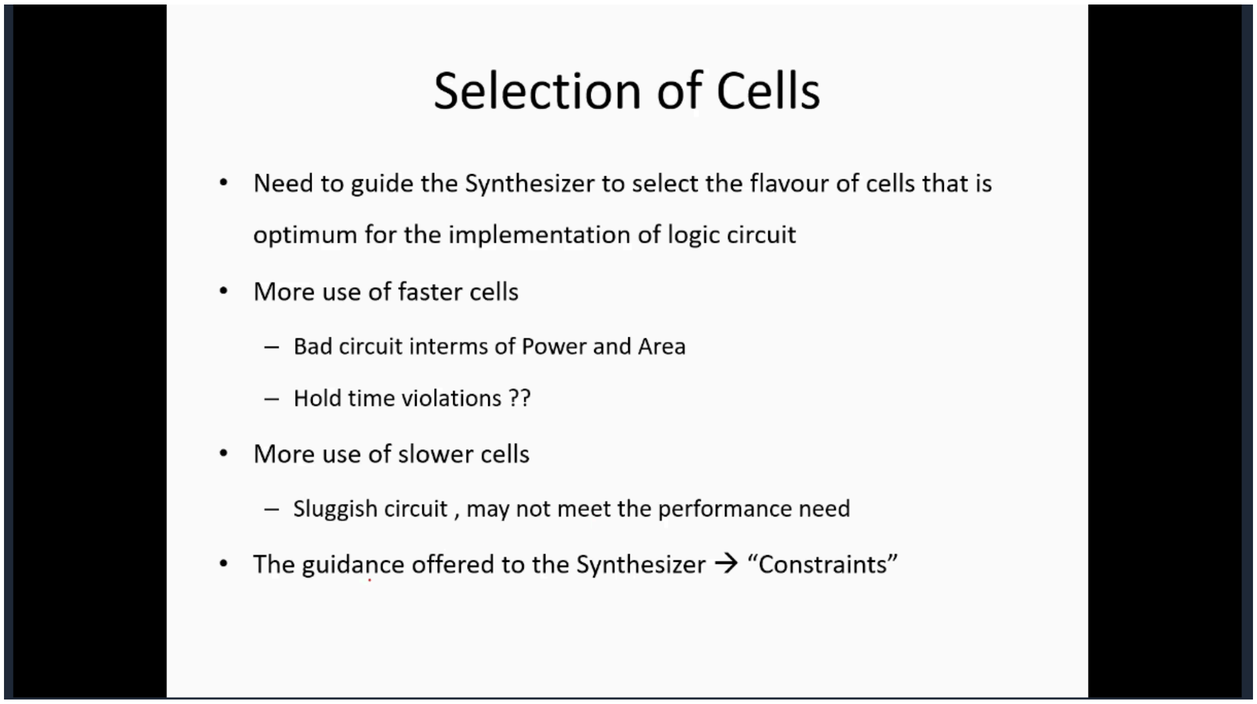
We need to guide the synthesis to select the falvour of cells that is optimum for implementation of logic circuit

More use of faster cells
Bad circuit in terms of power and area
Possibility of hold time violations

More use of slower cells

Sluggish circuit, may not meet performance needs

Constraints- the guidance offered to the synthesiser



Selection of Cells

- Need to guide the Synthesizer to select the flavour of cells that is optimum for the implementation of logic circuit
- More use of faster cells
  - Bad circuit interms of Power and Area
  - Hold time violations ??
- More use of slower cells
  - Sluggish circuit , may not meet the performance need
- The guidance offered to the Synthesizer → “Constraints”

ILLUSTRATION OF SYNTHESIS

First synthesiser will do a syntactical check of the code

Then it will start mapping

First the ports are mapped(input and output)

Here, assign becomes a mux, and output of mux is connected to input of the flop and clock and reset connections is made.

This is the conversion of the RTL in terms of std cell cells present in the .LIB

This is given out as a netlist

9- SKY130RTL D1SK4 L1 LAB3 Yosys 1 good mux Part 1

We will see how to invoke yosys and how to synthesis our design.

INVOKING YOSYS-

In the verilog_files directory, we will Type “yosys” and use the my_lib file for this project

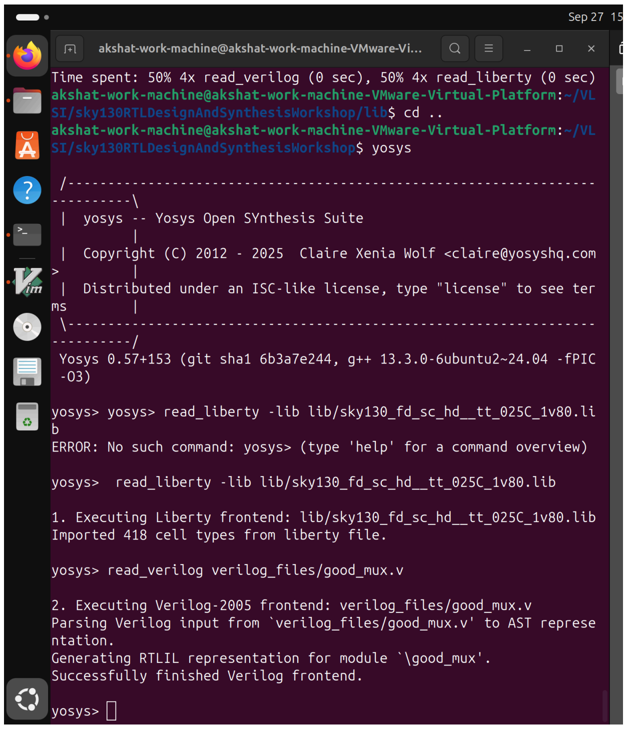
```
akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesig
nAndSynthesisWorkshop$ yosys

yosys>  read_liberty -lib lib/sky130_fd_sc_hd__tt_025C_1v80.lib
```

NOW WE WILL READ THE DESIGN

Command is “read_verilog” and in this case we will read the good mux, hence “read_verilgo good_mux.v”

In case our design spans more than one file, we will have to read all of them, we will see such an example later

```
read_verilog verilog_files/good_mux.v
```



```
Time spent: 50% 4x read_verilog (0 sec), 50% 4x read_liberty (0 sec)
akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesignAndSynthesisWorkshop/lib$ cd ..
akshat-work-machine@akshat-work-machine-VMware-Virtual-Platform:~/VLSI/sky130RTLDesignAndSynthesisWorkshop$ yosys

 /----------------------------------------------------------------------------\
 |  yosys -- Yosys Open SYnthesis Suite                                       |
 |  Copyright (C) 2012 - 2025  Claire Xenia Wolf <claire@yosyshq.com>         |
 |  Distributed under an ISC-like license, type "license" to see terms        |
 \----------------------------------------------------------------------------/
 Yosys 0.57+153 (git sha1 6b3a7e244, g++ 13.3.0-6ubuntu2~24.04 -fPIC -O3)

yosys> yosys> read_liberty -lib lib/sky130_fd_sc_hd__tt_025C_1v80.lib
ERROR: No such command: yosys> (type 'help' for a command overview)

yosys>  read_liberty -lib lib/sky130_fd_sc_hd__tt_025C_1v80.lib

1. Executing Liberty frontend: lib/sky130_fd_sc_hd__tt_025C_1v80.lib
Imported 418 cell types from liberty file.

yosys> read_verilog verilog_files/good_mux.v

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

yosys> 
```

NOW WE WILL SYNTHESISE THIS DESIGN

NEXT step is "synth -top", this tells the tool, what module we re trying to synthesise,
"Synth -top (module name we have to synthesise) good_mux"

NOTE: This is a cobinational design, had this been sequential, we wouldve needed another command.

Now we need to write the verilog file, we have read the design and liberty file that is the library, now it is time to generate the netlist

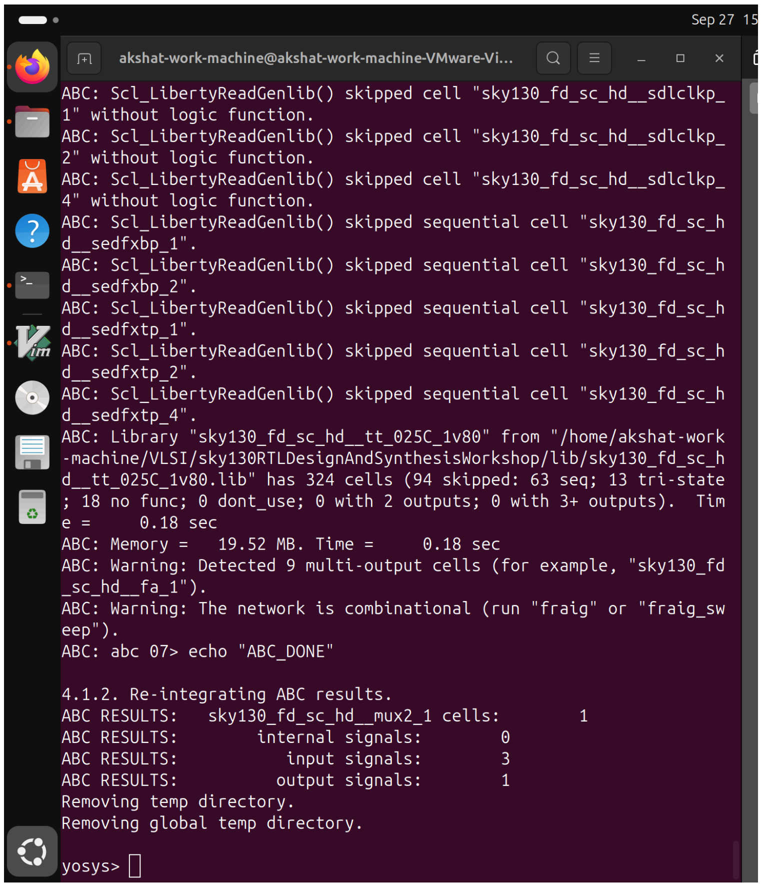
Command for netlist

"Abc -liberty (path to .lib)

"yosys> abc -liberty lib/sky130_fd_sc_hd__tt_025C_1v80.lib"

Abc command converts our tl file into the gates, and what gates it has to link to is specified in the library , the logic of good_mux will be realised using the standard cells in the earlier specified library

This info has to be paid attention to, it has identified 3 input signals, one putput and 0 internal signals
Because in our original



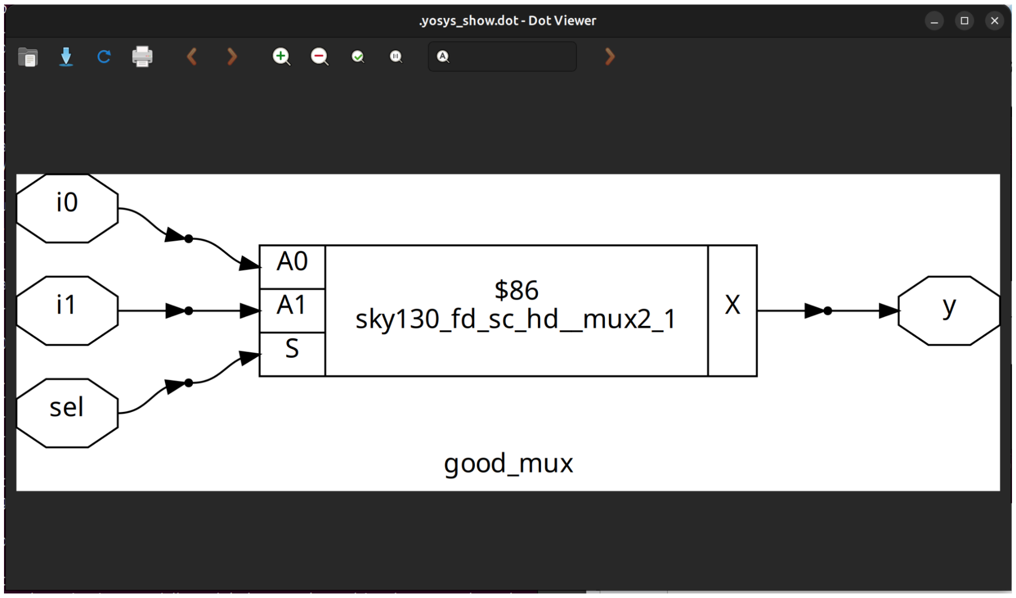
```
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_
1" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_
2" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__sdlclkp_
4" without logic function.
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_h
d__sedfxbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_h
d__sedfxbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_h
d__sedfxtp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_h
d__sedfxtp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_h
d__sedfxtp_4".
ABC: Library "sky130_fd_sc_hd__tt_025C_1v80" from "/home/akshat-work
-machine/VLSI/sky130RTLDesignAndSynthesisWorkshop/lib/sky130_fd_sc_h
d__tt_025C_1v80.lib" has 324 cells (94 skipped: 63 seq; 13 tri-state
; 18 no func; 0 dont_use; 0 with 2 outputs; 0 with 3+ outputs).  Tim
e =     0.18 sec
ABC: Memory =   19.52 MB. Time =     0.18 sec
ABC: Warning: Detected 9 multi-output cells (for example, "sky130_fd
_sc_hd__fa_1").
ABC: Warning: The network is combinational (run "fraig" or "fraig_sw
eep").
ABC: abc 07> echo "ABC_DONE"

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.
Removing global temp directory.

yosys> 
```

IT HAS REALISED THE LOGIC USING A 2 TO 1 MUX

4.1.2. Re-integrating ABC results.

ABC RESULTS: sky130_fd_sc_hd__mux2_1 cells: 1

USING the “show” command, we get a graphical representation of our design

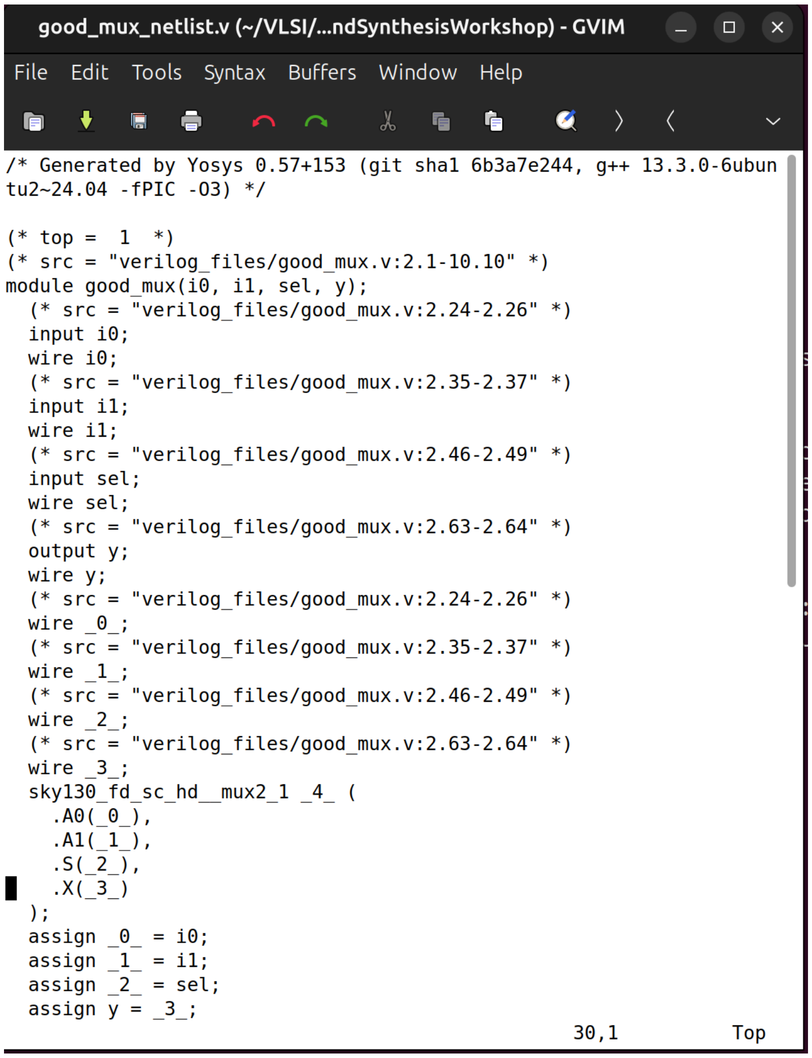


10,11- SKY130RTL D1SK4 L1 LAB3 Yosys 1 good mux Part 2,3

WRITING THE NETLIST

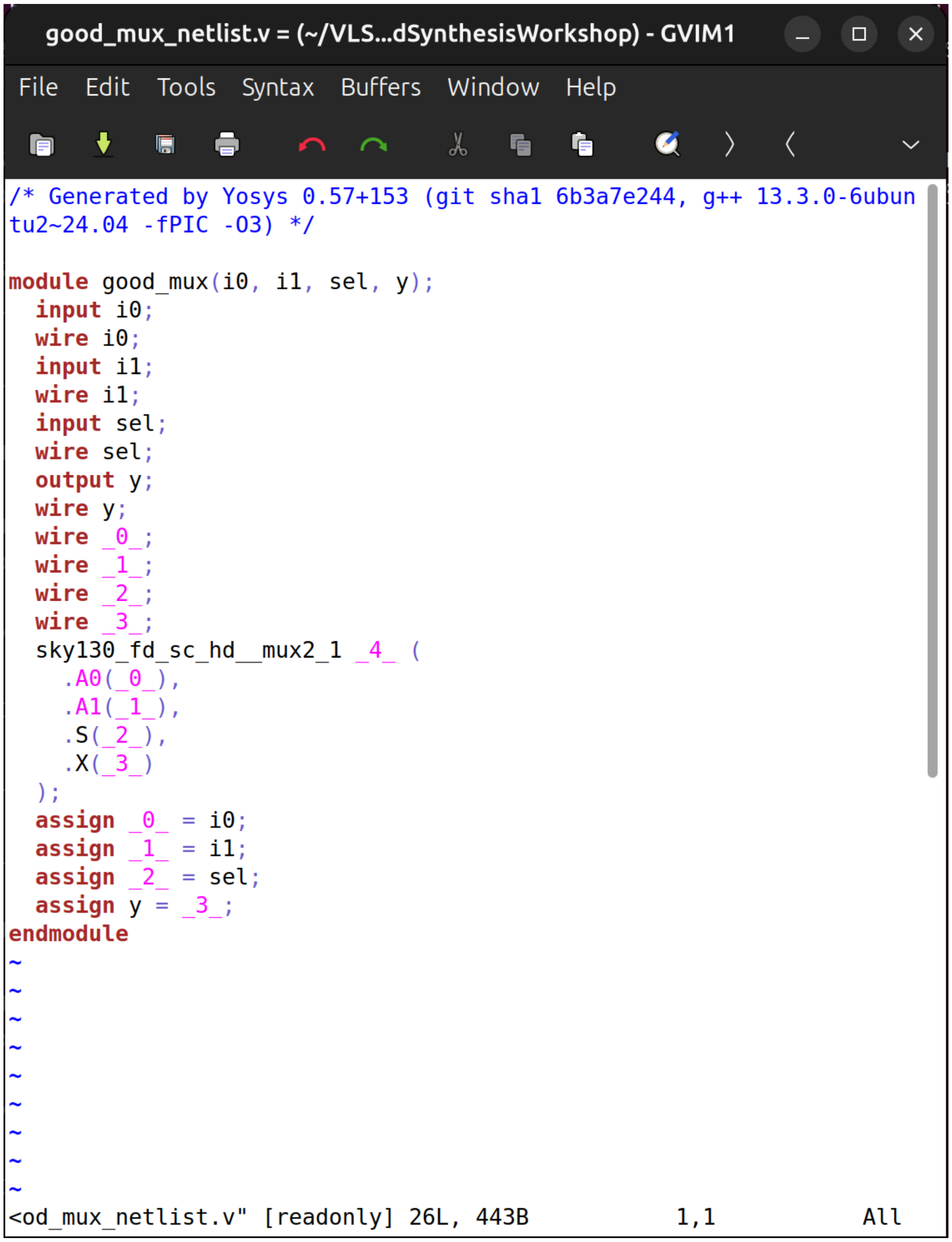
Command to write the netlist is “write_verilog good_mux_netlist.v” instead of good_mux_netlist any good name can be given

!gvim good_mux_netlist.v


good_mux_netlist.v (~/VLSI/...ndSynthesisWorkshop) - GVIM
File Edit Tools Syntax Buffers Window Help

```
/* Generated by Yosys 0.57+153 (git sha1 6b3a7e244, g++ 13.3.0-6ubun
tu2~24.04 -fPIC -O3) */

(* top =  1  *)
(* src = "verilog_files/good_mux.v:2.1-10.10" *)
module good_mux(i0, i1, sel, y);
  (* src = "verilog_files/good_mux.v:2.24-2.26" *)
  input i0;
  wire i0;
  (* src = "verilog_files/good_mux.v:2.35-2.37" *)
  input i1;
  wire i1;
  (* src = "verilog_files/good_mux.v:2.46-2.49" *)
  input sel;
  wire sel;
  (* src = "verilog_files/good_mux.v:2.63-2.64" *)
  output y;
  wire y;
  (* src = "verilog_files/good_mux.v:2.24-2.26" *)
  wire _0_;
  (* src = "verilog_files/good_mux.v:2.35-2.37" *)
  wire _1_;
  (* src = "verilog_files/good_mux.v:2.46-2.49" *)
  wire _2_;
  (* src = "verilog_files/good_mux.v:2.63-2.64" *)
  wire _3_;
  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_;
                                                  30,1          Top
```

But this has a lot of unnecessary info, so we will add another command
“write _verilog -noattr good_mux_netlist.v”


good_mux_netlist.v = (~/VLS...dSynthesisWorkshop) - GVIM1
File Edit Tools Syntax Buffers Window Help

```
/* Generated by Yosys 0.57+153 (git sha1 6b3a7e244, g++ 13.3.0-6ubun
tu2~24.04 -fPIC -O3) */

module good_mux(i0, i1, sel, y);
  input i0;
  wire i0;
  input i1;
  wire i1;
  input sel;
  wire sel;
  output y;
  wire y;
  wire _0_;
  wire _1_;
  wire _2_;
  wire _3_;
  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
~
~
~
~
~
~
~
~
~
<od_mux_netlist.v" [readonly] 26L, 443B           1,1           All
```