VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh

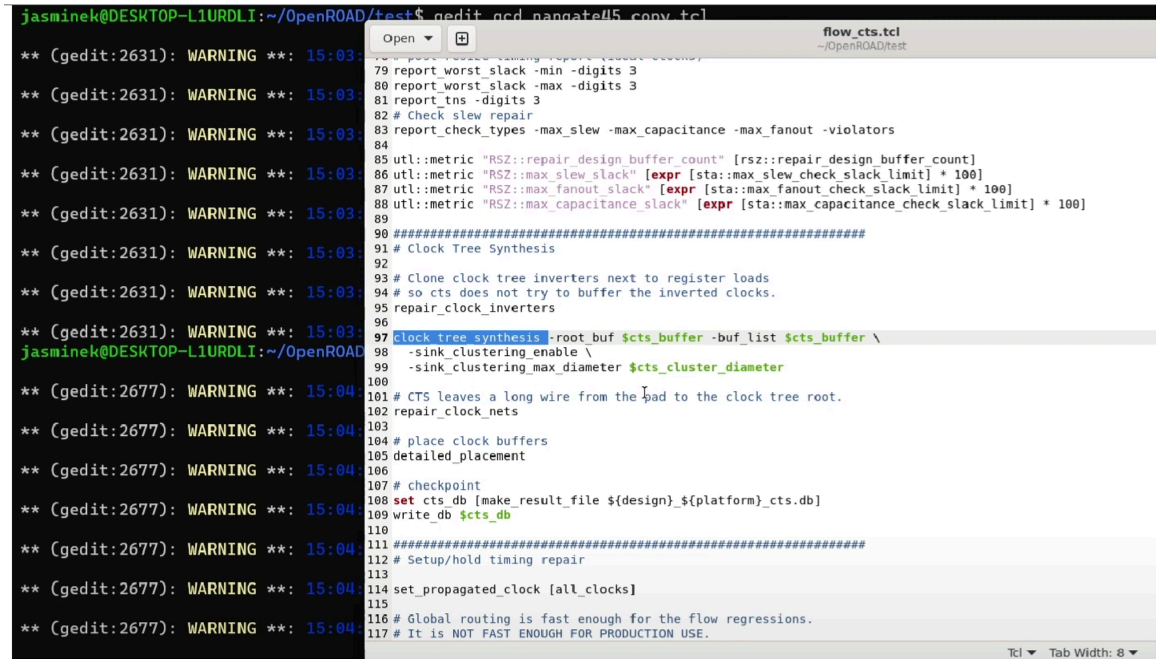
Department of Electronics and Communication Engineering
IIIT-Delhi

Lecture 54
Clock Tree Synthesis (CTS) and Routing

Hello everyone, my name is Jasmine Kaur. I am a PhD student at IIIT Delhi and I am your TA for the course VLSI Design Flow RTL to GDS. Welcome to tutorial 12. In the last tutorial, we have looked at the various steps of chip planning including the floor planning and power planning. Now after that we also looked at the placement step and now in this tutorial we will look at the next steps of physical design that are clock tree synthesis and routing and also after that we will perform parasitic extraction. In the previous tutorial, we looked at this gcd_nangate45.tcl file.

So again in this I have created two subscripts in which in one strict script I will be performing clock tree synthesis and the second script is for the complete flow from the clock tree synthesis and then the global routing and detailed routing step. So let us first look at the clock tree synthesis step. So in these firstly these are the steps that we already did in the last tutorial and now we will start with clock tree synthesis. So firstly we repair the clock inverters.

So this basically clones the clock tree inverters next to the resistor loads so that CTS does not try to buffer the inverted clocks. Now we are performing the clock tree synthesis and here we give the list of buffers and the sync clustering is enabled here and the diameter for sync clustering is given. Then we repair the clock nets. This is because when we perform clock tree synthesis there may be long wires from the pad to the clock tree root. So we need to insert buffers in these long wires.



```
jasminek@DESKTOP-L1URDLI:~/OpenROAD/test$ gedit gcd_nangate45_copy.tcl

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:

** (gedit:2631): WARNING **: 15:03:
jasminek@DESKTOP-L1URDLI:~/OpenROAD

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:

** (gedit:2677): WARNING **: 15:04:
```

```
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
# Check slew repair
report_check_types -max_slew -max_capacitance -max_fanout -violators

utl::metric "RSZ::repair_design_buffer_count" [rsz::repair_design_buffer_count]
utl::metric "RSZ::max_slew_slack" [expr [sta::max_slew_check_slack_limit] * 100]
utl::metric "RSZ::max_fanout_slack" [expr [sta::max_fanout_check_slack_limit] * 100]
utl::metric "RSZ::max_capacitance_slack" [expr [sta::max_capacitance_check_slack_limit] * 100]

################################################################
# Clock Tree Synthesis

# Clone clock tree inverters next to register loads
# so cts does not try to buffer the inverted clocks.
repair_clock_inverters

clock_tree_synthesis -root_buf $cts_buffer -buf_list $cts_buffer \
  -sink_clustering_enable \
  -sink_clustering_max_diameter $cts_cluster_diameter

# CTS leaves a long wire from the pad to the clock tree root.
repair_clock_nets

# place clock buffers
detailed_placement

# checkpoint
set cts_db [make_result_file ${design}_${platform}_cts.db]
write_db $cts_db

################################################################
# Setup/hold timing repair

set_propagated_clock [all_clocks]

# Global routing is fast enough for the flow regressions.
# It is NOT FAST ENOUGH FOR PRODUCTION USE.
```

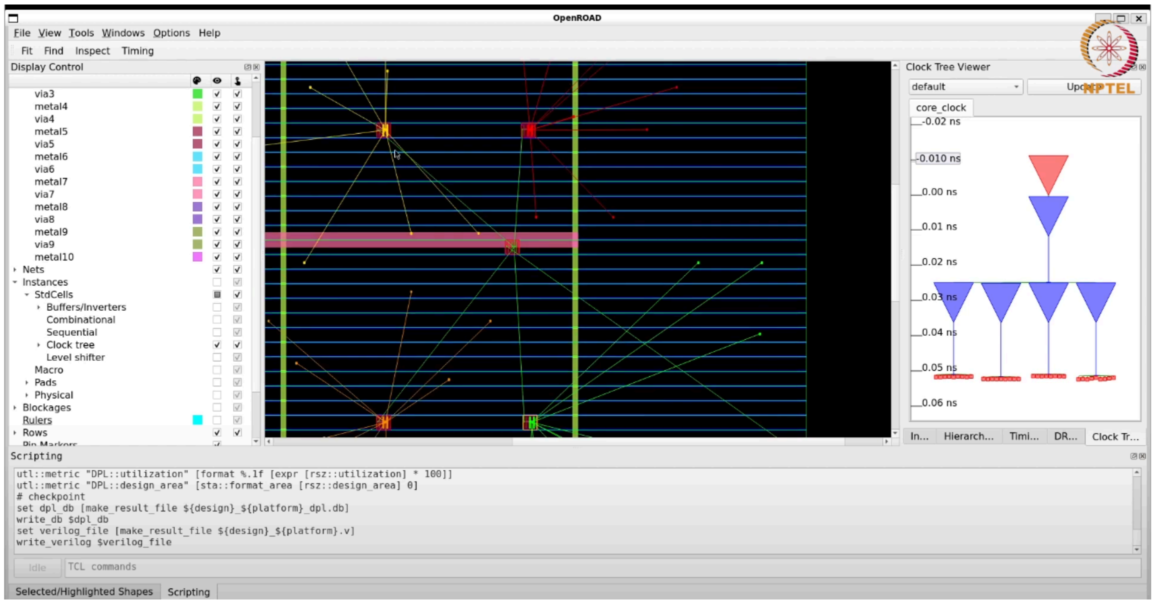
So this is next we are doing this detailed placement. So placement of these clock buffers that are placed in this clock tree synthesis and then finally we are creating DB files. Then we perform a set of whole timing repairs. Now after the clock tree synthesis we have actual latencies and actual timing of the clocks. So we need to again check the setup and hold and repair the design.

So firstly what we are doing is we are estimating the parasitics here. So either we can do that using global routing or we can just estimate on the basis of the placement. But since global routing is not fast enough for production use so we will be using placement based parasitic estimation. And after we have the parasitics we will perform repair timing. So in this the setup and hold violations are repaired by downsizing of the cells or using high VT cells.

Then we are reporting the different timing reports. The worst slack for hold and set up time and total negative slack and other things. Now after that we will perform detailed placement after we have done the resizing and we have to place the CTS, clock tree synthesis. Then we are here creating DB files and writing a Verilog file. Now we will run this.

So this is how the layout looks after the clock tree synthesis step. So let us see in this instance we have this clock tree option. So here we can see the instances that are inserted after the clock tree synthesis. So we can see this is a clock buffer. So all these cells that are here are the clock buffer cells that are inserted in the clock tree synthesis step.

Now we can see the clock tree using this clock tree viewer that is there in this windows tab and pressing this update button we can see the clock tree of the fly lines of the clock tree. So this is coming from the clock pin and going to this buffer and this main buffer is supplying the clocks to these four buffers. And finally that goes to sequential cells. Here also we can see this is the clock pin going to the one buffer and then that is given to the four buffers and then finally to the standard cells. The next step.



```
utl::metric "DPL::utilization" [format %.1f [expr [rsz::utilization] * 100]]
utl::metric "DPL::design_area" [sta::format_area [rsz::design_area] 0]
# checkpoint
set dpl_db [make_result_file ${design}_${platform}_dpl.db]
write_db $dpl_db
set verilog_file [make_result_file ${design}_${platform}.v]
write_verilog $verilog_file
```

Let us look at the next step of routing. So we will source this flow dot tcl file in the script file that we are using. So gedit flow dot tcl file in this. So the first step here is global routing. So in this firstly we are giving the pin access.

So here we define the routing layers for accessing the pins of the standard cells. Then we are setting the route guide output file and finally we perform the global routing. Here we perform the global routing and the number of congestion iterations to check for a congestion overflow is 100 here. Then we set the verilog file and write the verilog file for the global routing step. Here we are checking the antennas if any antenna violations are there and after that we do the filler placement.

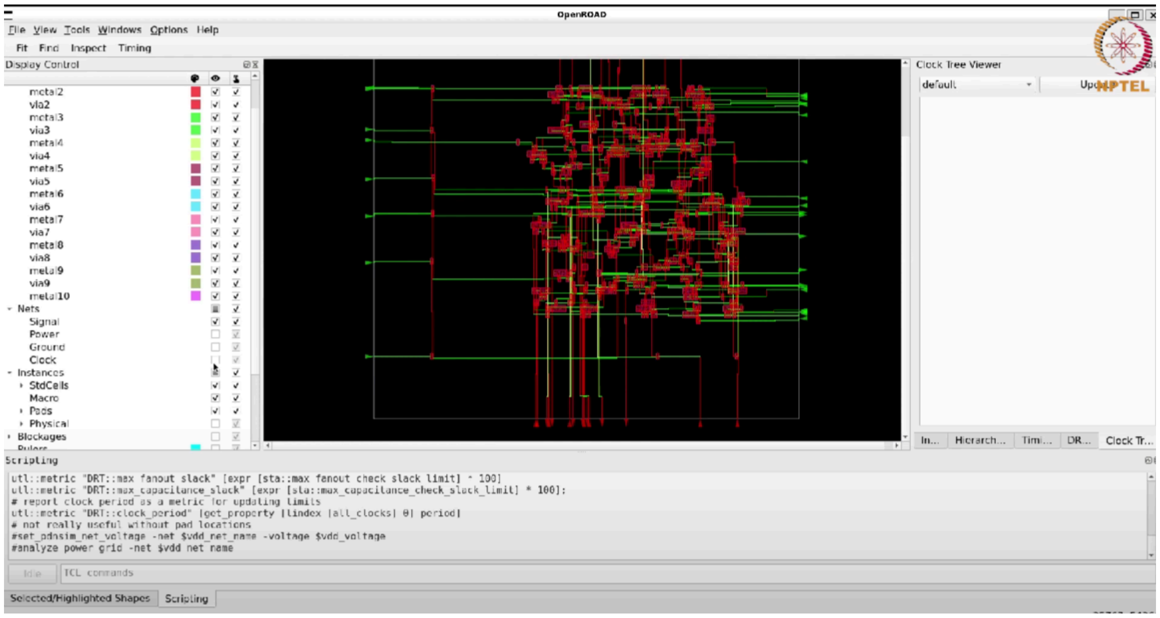
So the filler cells are placed in this step and then we check for the legality of the placement of these filler cells if they are legal or not. Then here we are writing the db file. Then the next step is detailed routing. In this again we are giving the routing layers for the pin access. Then we are doing detailed routing.

For this we are first using the set thread count to find the number of processors and the number of processors available. So the detailed routing step will be done parallelly within multiple multi thread options. Then this is a detailed routing step in which we are giving these different options. We are generating the output DRC file and the different options are given here that you can look at. Then we are writing the guides file and then again we are checking the antenna violations if there are any antenna violations.

And here we don't have any antenna and there will be no antenna violations in our design. So if there are any antenna violations we can repair that using the repair design option. And then we are writing the db and def files. Then the final step is parasitic extraction to find the RC values. So for that firstly we need to have a RC file that is given in the NANGATE 45 library folder and in that we are extracting the parasitics and then writing the standard parasitic extraction format file.

If we don't have these RC files then we estimate the parasitics based on the global routing. And after we have these parasitics we generate the final reports in which we generate the reports for worst slack for hold and set up total negative slack and different reports for different paths are generated here. And we are checking the power, the clock skew, the floating nets and the design area. So to run this. So this is the layout that is generated after the complete flow.

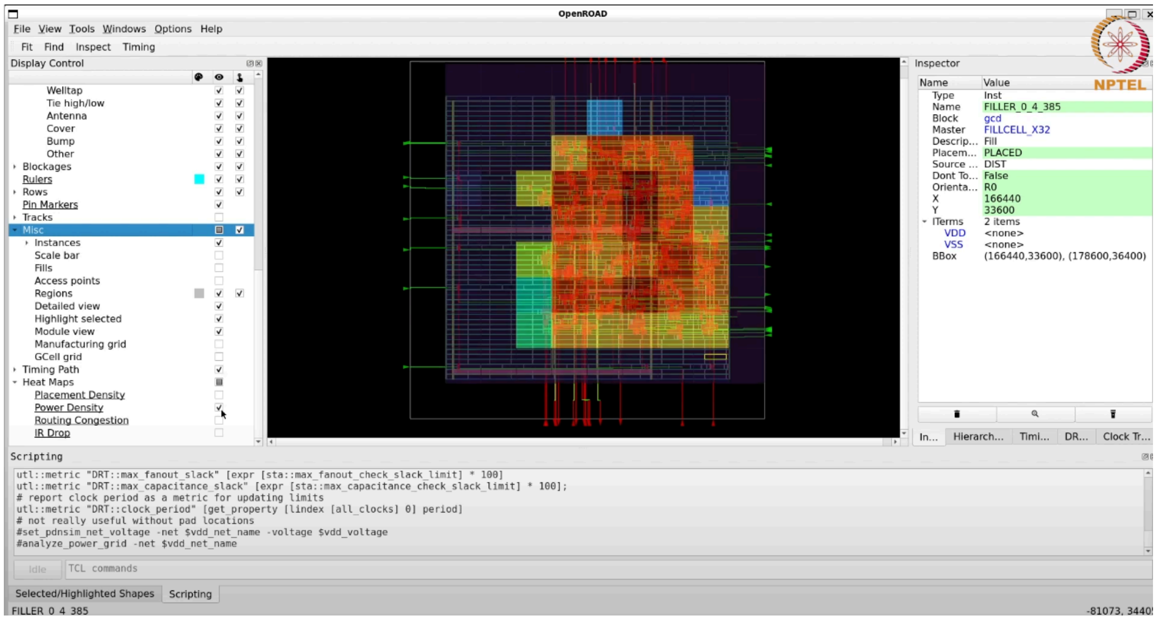
So here we can see for the signal nets the routing is done or not using this signal nets option. Firstly let us include the instances as well. And I have just taken the standard cells and now these are the signal routes that are going from the signal nets that are going from the pins to the different cells. And then these are the clock nets.



```
utl::metric "DRT::max fanout slack" [expr [sta::max fanout check slack limit] * 100]
utl::metric "DRT::max_capacitance_slack" [expr [sta::max_capacitance_check_slack_limit] * 100];
# report clock period as a metric for updating limits
utl::metric "DRT::clock_period" [get_property [lindex [all_clocks] 0] period]
# not really useful without pad locations
#set_pdnsim_net_voltage -net $vdd_net_name -voltage $vdd_voltage
#analyze power grid -net $vdd net name
```

These are the clock nets. So these are all the complete layout again with the power and ground nets. And in instances let us check for these filler cells that we included. So these are the filler cells that are included in the design everywhere they are included. So these are the different options that you can explore. Then in this heat maps option we can look at the placement density.

So we can say the dark blue area is where the density is more. This is the power density. So the red areas are having more power. This is the routing congestion. And then we can see the clock tree view here.



Name	Value
Type	Inst
Name	FILLER_0_4_385
Block	gcd
Master	FILLCELL_X32
Descrip...	Fill
Placem...	PLACED
Source ...	DIST
Dont To...	False
Orienta...	R0
X	166440
Y	33600
ITerms	2 items
VDD	<none>
VSS	<none>
BBox	(166440,33600), (178600,36400)

```
utl::metric "DRT::max_fanout_slack" [expr [sta::max_fanout_check_slack_limit] * 100]
utl::metric "DRT::max_capacitance_slack" [expr [sta::max_capacitance_check_slack_limit] * 100];
# report clock period as a metric for updating limits
utl::metric "DRT::clock_period" [get_property [lindex [all_clocks] 0] period]
# not really useful without pad locations
#set_pdnsim_net_voltage -net $vdd_net_name -voltage $vdd_voltage
#analyze_power_grid -net $vdd_net_name
```

FILLER 0 4 385 -81073, 3440

So this was all about today's tutorial. So we have covered the complete physical design flow and you can look at the different scripting commands that I use and their options in the open road github. I will provide you with the link to this and that's all for today. Thank you.