VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh

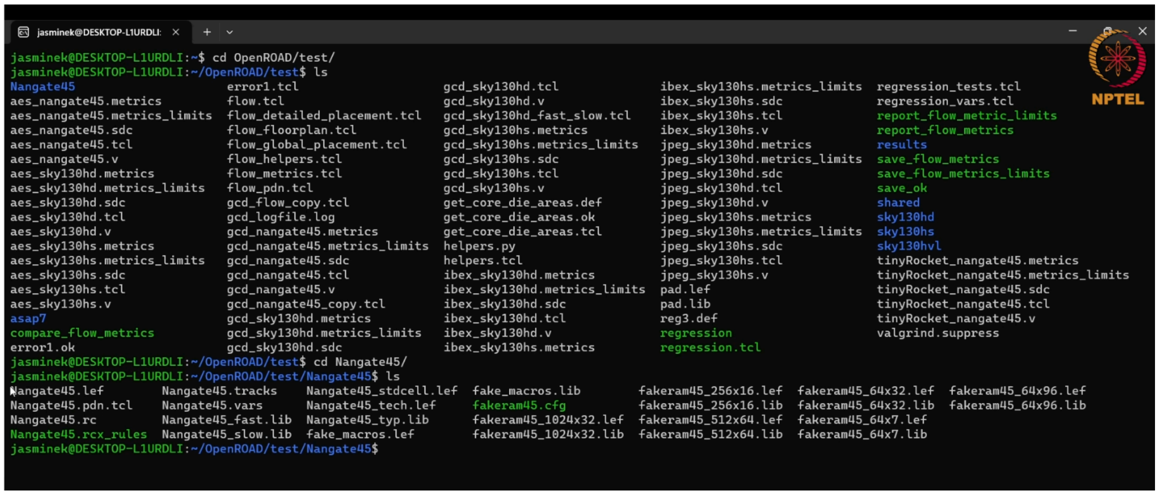
Department of Electronics and Communication Engineering
IIIT-Delhi

Lecture 7
Chip Planning and Placement

Hello everyone. My name is Jasmine Kaur and I am a PhD student at IIIT Delhi. I am your TA for the course VLSI Design Flow RTL to GDS. Welcome to tutorial 11. In this tutorial, we will look at the various stages of chip planning that include floor planning, power planning and placement. So for this, we will be using the Open Road app that we installed in the last tutorial.

So now let us see what are the inputs that will be required by the Open Road tool to perform the chip planning. So these are the files and folders, example files and folders that were there in the Open Road test folder that we saw in the last tutorial. So in this tutorial, we will be looking at the gcd example using the nangate45 library and this is the Verilog Netlist that we will be using. And the gcd_nangate45.sdc file is the constraint file that we will be using.

So let us see the library files. So in the Nangate45 folder, we have these library files. So Nangate45_typ.lib we will be using and the physical information of the technology files and the standard cells will be in these Nangate45_stdcell.lef and Nangate45_tech.lef files. So the script that will be running is this gcd_nangate45.tcl file. So now let us look at this script file gcd_nangate45.tcl .



```
jasminek@DESKTOP-L1URDLI:~$ cd OpenROAD/test/
jasminek@DESKTOP-L1URDLI:~/OpenROAD/test$ ls
Nangate45                     error1.tcl                     gcd_sky130hd.tcl               ibex_sky130hs.metrics_limits  regression_tests.tcl
aes_nangate45.metrics         flow.tcl                       gcd_sky130hd.v                 ibex_sky130hs.sdc             regression_vars.tcl
aes_nangate45.metrics_limits  flow_detailed_placement.tcl    gcd_sky130hd_fast_slow.tcl     ibex_sky130hs.tcl             report_flow_metric_limits
aes_nangate45.sdc             flow_floorplan.tcl             gcd_sky130hs.metrics           ibex_sky130hs.v               report_flow_metrics
aes_nangate45.tcl             flow_global_placement.tcl      gcd_sky130hs.metrics_limits    jpeg_sky130hd.metrics         results
aes_nangate45.v               flow_helpers.tcl               gcd_sky130hs.sdc               jpeg_sky130hd.metrics_limits  save_flow_metrics
aes_sky130hd.metrics          flow_metrics.tcl               gcd_sky130hs.tcl               jpeg_sky130hd.sdc             save_flow_metrics_limits
aes_sky130hd.metrics_limits   flow_pdn.tcl                   gcd_sky130hs.v                 jpeg_sky130hd.tcl             save_ok
aes_sky130hd.sdc              gcd_flow_copy.tcl              get_core_die_areas.def         jpeg_sky130hd.v               shared
aes_sky130hd.tcl              gcd_logfile.log                get_core_die_areas.ok          jpeg_sky130hs.metrics         sky130hd
aes_sky130hd.v                gcd_nangate45.metrics          get_core_die_areas.tcl         jpeg_sky130hs.metrics_limits  sky130hs
aes_sky130hs.metrics          gcd_nangate45.metrics_limits   helpers.py                     jpeg_sky130hs.sdc             sky130hvl
aes_sky130hs.metrics_limits   gcd_nangate45.sdc              helpers.tcl                    jpeg_sky130hs.tcl             tinyRocket_nangate45.metrics
aes_sky130hs.sdc              gcd_nangate45.tcl              ibex_sky130hd.metrics          jpeg_sky130hs.v               tinyRocket_nangate45.metrics_limits
aes_sky130hs.tcl              gcd_nangate45.v                ibex_sky130hd.metrics_limits   pad.lef                       tinyRocket_nangate45.sdc
aes_sky130hs.v                gcd_nangate45_copy.tcl         ibex_sky130hd.sdc              pad.lib                       tinyRocket_nangate45.tcl
asap7                         gcd_sky130hd.metrics           ibex_sky130hd.tcl              reg3.def                      tinyRocket_nangate45.v
compare_flow_metrics          gcd_sky130hd.metrics_limits    ibex_sky130hd.v                regression                    valgrind.suppress
error1.ok                     gcd_sky130hd.sdc               ibex_sky130hs.metrics          regression.tcl
jasminek@DESKTOP-L1URDLI:~/OpenROAD/test$ cd Nangate45/
jasminek@DESKTOP-L1URDLI:~/OpenROAD/test/Nangate45$ ls
Nangate45.lef        Nangate45.tracks     Nangate45_stdcell.lef  fake_macros.lib          fakeram45_256x16.lef  fakeram45_64x32.lef  fakeram45_64x96.lef
Nangate45.pdn.tcl    Nangate45.vars       Nangate45_tech.lef     fakeram45.cfg            fakeram45_256x16.lib  fakeram45_64x32.lib  fakeram45_64x96.lib
Nangate45.rc         Nangate45_fast.lib   Nangate45_typ.lib      fakeram45_1024x32.lef    fakeram45_512x64.lef  fakeram45_64x7.lef
Nangate45.rcx_rules  Nangate45_slow.lib   fake_macros.lef        fakeram45_1024x32.lib    fakeram45_512x64.lib  fakeram45_64x7.lib
jasminek@DESKTOP-L1URDLI:~/OpenROAD/test/Nangate45$
```

So in this file, firstly, we are sourcing some files. So this helpers.tcl and flow_helpers.tcl file contains the helper functions and this Nangate45.vars file contains the variable values defined in this. Then we are defining some variables like the design variable for the name of the design, which is gcd, the top module name that is gcd and we are giving input synthesized Verilog file and input constraint file using these variables. Then we are giving the die area and core area. This is given in LX, LY, UX, UI format.

So what is this LX, LY is it is the bottom most bottom left corner coordinates and UX, UY are the top right coordinates. So knowing these two coordinate values, we can find the die area and the core area. Then finally we are sourcing flow.tcl files. So this file contains a script for complete physical design flow. So let us see what is there in this file.

So in this file we can see firstly we are reading the library files using read libraries. So this is defined in the flow_helpers.tcl file. This is a procedure defined in this file. Then we are reading the Verilog file and link design is used to flatten the design. Then we are reading the sdc file here.

And next we can see floor planning is being done in this file. In IO placement, macro placement, global placement, clock tree synthesis, detailed placement, global routing. So the complete flow of the physical design is being done in this flowfloor.tcl file. So what I have done is I have broken down this file into subscripts so that we can see step by step how things are happening. So let us see I have created a gcd_nangate45_copy.tcl file.

In this we will be sourcing the different script files turn by turn. So firstly we are doing floor planning. So let's source this flow_floorplan.tcl file. So in this file firstly we are initializing the floor plan. And this site here defines the list of sites to make rows.

So basically this is the unit tile that has the minimum height and width of a cell to be placed. So cells are placed in multiples of these sites. Then we are defining the die area and core area. And then we are writing the def file. So this is a design exchange format file that contains the physical design information.

And then we are sourcing the tracks_file type. So in this source tracks_file we are adding routing tracks to the floor plan. And then remove buffers are used to remove the buffers that are inserted by the synthesis tool. Now we will be doing IO placement using the place pins command. Here we are doing random pin placement.

And we are defining the horizontal and vertical layers that will be used. Then macro placement. So firstly we will be checking if there are macros then we will place them using global placement. And then we will place the macros and we will specify the hello and channel. Then the next step is tap cell insertion.

So tap cells and cap cells are added here which prevent latch up and guard against the manufacturing damage to cell gates that are close to the border. And then we are writing the def file. So let us run this first script. openroad -gui for GUI option. Then we can create a log file using the -log option.

And then we give the input script that needs to be run. So gcd_nangate45_copy_tcl file . So here we can see that we have a core area and we have a die area. Then these are the standard cell rows. So we can zoom in and these are the standard cell rows in which the standard cells have to be placed.

If we zoom in further, so this unit tile here, this is the site. And then these are the IO pins and these IO pins are random for now. So we have placed them randomly in this step. Then also we can see no macros have been created because the tool did not find any macro blocks in this design. Then after that tap cell insertion.

So we can have a look at the tap cells from this option instance physically. So these are the tap cells or we can save well taps. And we can see that these are created at the boundary of the core. Let's check that. So we can see this is the core and they are at the boundary of the core.

So next let us go to the next step which is power distribution network. So let us look at the flow_pdn.tcl file. So in this file we are firstly sourcing the pdn_cfg file. So what is this file? This file defines the global connections which are VDD, VSS. It defines the voltage domains, defines the power grids for standard cells and macros.

And finally this pdngen it builds the power grid. Then we are writing the def file for this. So now let us run this. So here we can see that power grids and the power distribution network has been created. So we can see that in nets option here that no signal and no clock nets are created yet.

And these are the power nets and these are the ground nets. So this is the complete power distribution network. Now moving on to the next step for global placement. So let's source the global placement flow script. So let us see what is happening there.

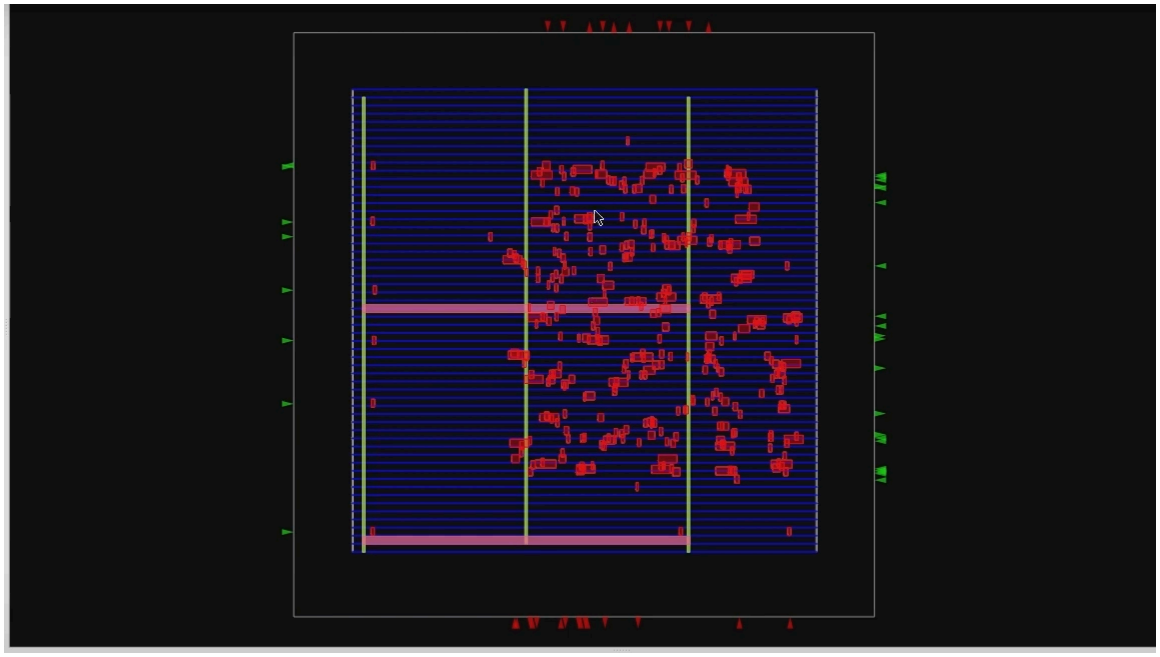
So firstly we are setting global routing layer adjustment here. So why is it being done is it is used by the global router during placement to estimate and avoid the congestion. So it is basically a congestion estimate being done here before we can do the global placement. Then we are setting the routing layers for signal and clock. And here we are setting the macro extension.

That means we are setting the number of g cells or routing means that need to be added to the blockages boundaries from the macros. And now we are doing the global placement which is routability driven. Here we are defining the density, the target placement density and we are adding the paddings to the left and right of the cells. Now we do the IO placement. Now this time the IO placement is optimized.

And now we can write a db file and a def file. Then in this step we are repairing the maximum slew capacitance fan out violations and normalizing the slews. So firstly we source the layer_rc file This is nangate45_rc file which contains the resistance and capacitance of each metal layer. Then we set the wire rc for the signal and clock nets.

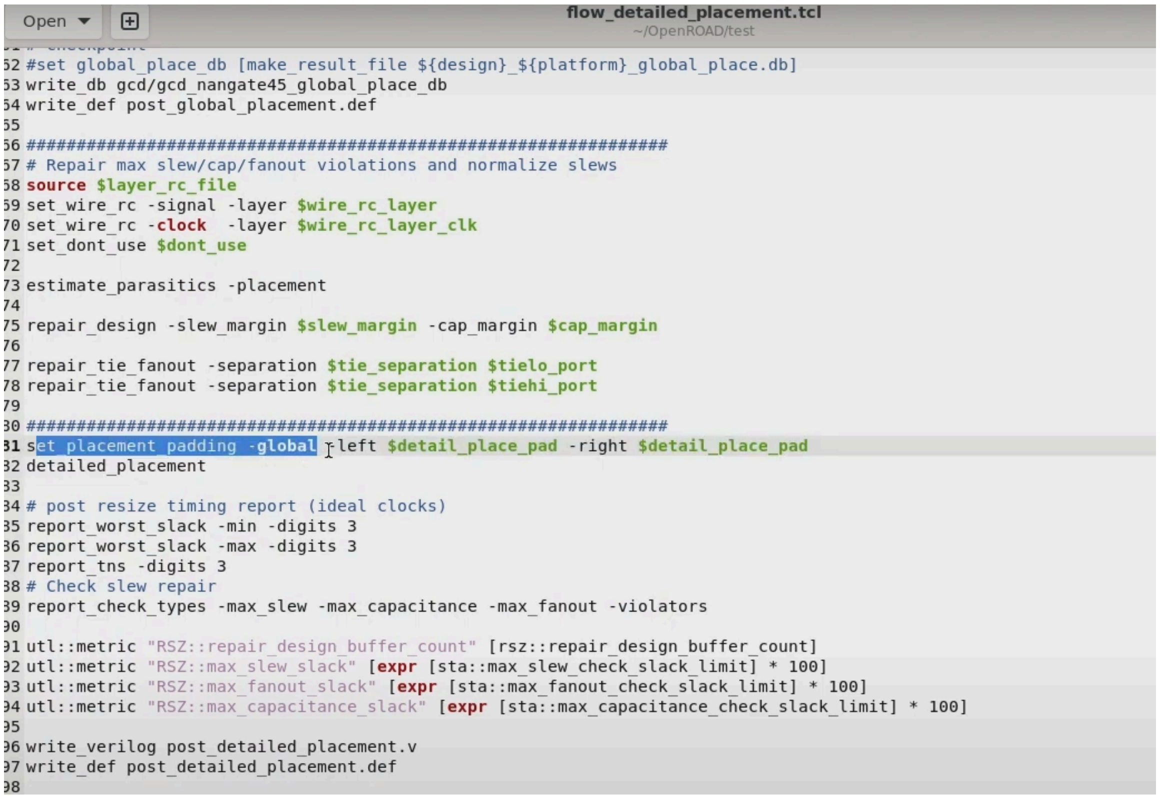
This is used for delay calculation. Then finally we are setting don't use cells which are used to remove from the consideration by resizer. Then we are estimating the parasitics after the placement. These will be considered during the repairing of the design which we are doing in this repair design and we are giving the slew margin and capacitance margin. So what is this repairing of design? What happens in this is we insert the tool inserts buffers on the nets to repair the maximum slew, maximum capacitance, maximum fan out violations and also on long wires to reduce the rc delay and it also resizes gates to normalize the slew.

Then we are repairing the tie fanout . So when some netlist is any when a netlist pin is connected to logic 0 or 1 then we can insert a tie cell whose gate is always tied to VDD or VSS. So that is the purpose of this here and now let us run this. So we can see these red blocks. These are the standard cells. So global placement of standard cells is done and we can see there is some overlapping of cells for now because we have just done the global placement and not legalized the placement.



So there is some overlapping and we can also see that these cells are not inside the standard cell rows that are there. Now you can see these pins have also changed their position and now they are placed according to the optimizations. They are optimized locations. Moving on to the next step. So the next step is for today's tutorial. The last step is the flow of detailed placement.

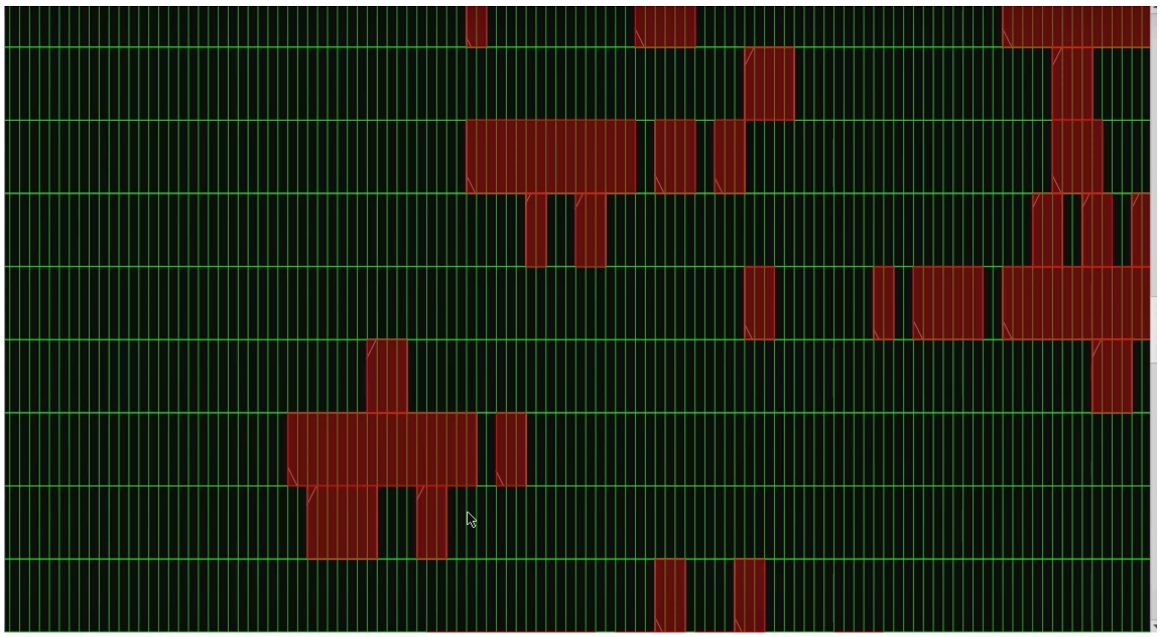
So let us see what is happening in this detailed placement. So first is setting placement padding. So in this placement pads are added on the left and right. This is done to leave room for routing in the further steps and then finally we do the detailed placement and then we are reporting the different setup hold slacks and total negative slack. So different timing reports we are generating and then we are checking the report check types and we are finally writing the def files and verilog files.



flow_detailed_placement.tcl
~/OpenROAD/test

```
#set global_place_db [make_result_file ${design}_${platform}_global_place.db]
write_db gcd/gcd_nangate45_global_place_db
write_def post_global_placement.def

################################################################
# Repair max slew/cap/fanout violations and normalize slews
source $layer_rc_file
set_wire_rc -signal -layer $wire_rc_layer
set_wire_rc -clock  -layer $wire_rc_layer_clk
set_dont_use $dont_use

estimate_parasitics -placement

repair_design -slew_margin $slew_margin -cap_margin $cap_margin

repair_tie_fanout -separation $tie_separation $tielo_port
repair_tie_fanout -separation $tie_separation $tiehi_port

################################################################
set_placement_padding -global -left $detail_place_pad -right $detail_place_pad
detailed_placement

# post resize timing report (ideal clocks)
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]

write_verilog post_detailed_placement.v
write_def post_detailed_placement.def
```

Now let us run this step. So now as we zoom in we can see now the standard cells there are no overlapping there and they are now placed inside the legal standard cell rows.



And we can see the ground and power nets as well. So this was all for today's tutorial. We will cover the next steps in physical design that is clock tree synthesis and the routing step in the next tutorial. Thank you very much. Thank you.