VLSI Design Flow: RTL To GDS (NPTEL Course)

Lab Tutorial 12

Objective: To gain a hands-on experience on **CTS and Routing** using OpenROAD app.

OpenROAD: OpenROAD is an integrated chip physical design tool that takes a design from synthesized Verilog to routed layout.

1. **Yosys:** for performing Logic Synthesis.

2. Floorplanning through Detailed Routing: using **OpenROAD**

References:

1. Ajayi, Tutu, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim et al. "Toward an open-source digital flow: First learnings from the openroad project." In Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1-4. 2019.
2. Ajayi, Tutu, and David Blaauw. "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain." In Proceedings of Government Microcircuit Applications and Critical Technology Conference. 2019.

Website:

https://theopenroadproject.org/

OpenRoad Community:

https://gitter.im/The-OpenROAD-Project/community

OpenRoad Documentation:

To see the available options for any OpenROAD command, you can type the following command in the openroad shell:

openroad> **help [pattern]**

Example 1:
openroad> **help macr***
This will generate a comprehensive listing of all relevant commands and their corresponding options that match with the pattern "macr*"
Example 2:
openroad> **help macro_placement**
This will list all the options of the openroad command macro_placement.

However, if you want to see the description of any command and their options, you can refer to the link provided below. On the left-hand side of the page, distinct sections are organized based on various stages, each delineating specific commands pertinent to that stage:

https://openroad.readthedocs.io/en/latest/

Activity: *Take any chip planning tool that is available to you (I am using OpenROAD tool) and carry out the following activities. Load a Verilog netlist. You should give the timing constraints, technology library, and physical information (possibly using LEF files) as inputs.*

TASK 1: Perform CTS (Clock Tree Synthesis)
Timing Optimization

TASK 2: Perform Global Routing
Filler cell insertion
Check antenna violations
Perform detailed routing

TASK 3: Parasitic extraction
Signing-off timing reports

Script used to execute activity:

gcd_nangate45.tcl
(Location: OpenROAD/test/gcd_nangate45.tcl)

Inputs to the OpenRoad Tool:

1. RTL Netlist: gcd_nangate45.v
(Location: OpenROAD/test/gcd_nangate45.v)
2. SDC file: gcd_nangate45.sdc
(Location: OpenROAD/test/gcd_nangate45.sdc)
3. Library file: Nangate45_typ.lib
(Location: OpenROAD/test/Nangate45/Nangate45_typ.lib)
4. LEF file
    A. Technology Lef: Nangate45_tech.lef
    (Location: OpenROAD/test/Nangate45/Nangate45_tech.lef)
    B. Standard Cell Lef: Nangate45_stdcell.lef
    (Location: OpenROAD/test/Nangate45/Nangate45_stdcell.lef)

NOTE:
a. Technology LEF file contains information about technology site, available layers for routing, layer's physical information (pitch, width, spacing, mask, direction etc.), DRC rules, VIA definition, Non-Default Rules (NDR).
b. Cell LEF file contains information about abstract view of macro and standard cells

1. gcd_nangate45.tcl

```
# gcd flow pipe cleaner
source "helpers.tcl"
source "flow_helpers.tcl"
source "Nangate45/Nangate45.vars"

set design "gcd"
set top_module "gcd"
set synth_verilog "gcd_nangate45.v"
set sdc_file "gcd_nangate45.sdc"
set die_area {0 0 100.13 100.8}
set core_area {10.07 11.2 90.25 91}

source -echo "flow.tcl”
# You can divide the flow.tcl file into smaller sub-files to systematically analyze the distinct stages of the Physical Design process as demonstrated in the tutorial.
```

2. flow.tcl

(Location: OpenROAD/test/flow.tcl)

```
# read design file (post_detailed_placement) from Tutorial 11
################################################################
# 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.
set repair_timing_use_grt_parasitics 0
if { $repair_timing_use_grt_parasitics } {
  # Global route for parasitics - no guide file requied
  global_route -congestion_iterations 100
  estimate_parasitics -global_routing
} else {
  estimate_parasitics -placement
}

repair_timing

# Post timing repair.
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
report_check_types -max_slew -max_capacitance -max_fanout -violators -digits 3

utl::metric "RSZ::worst_slack_min" [sta::worst_slack -min]
utl::metric "RSZ::worst_slack_max" [sta::worst_slack -max]
utl::metric "RSZ::tns_max" [sta::total_negative_slack -max]
utl::metric "RSZ::hold_buffer_count" [rsz::hold_buffer_count]

################################################################
# Detailed Placement

detailed_placement

# Capture utilization before fillers make it 100%
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

################################################################
# Global routing

pin_access -bottom_routing_layer $min_routing_layer \
        -top_routing_layer $max_routing_layer

set route_guide [make_result_file ${design}_${platform}.route_guide]
```

```
global_route -guide_file $route_guide \
  -congestion_iterations 100

set verilog_file [make_result_file ${design}_${platform}.v]
write_verilog -remove_cells $filler_cells $verilog_file

################################################################
# Antenna repair

check_antennas
utl::metric "GRT::ANT::errors" [ant::antenna_violation_count]

################################################################
# Filler placement

filler_placement $filler_cells
check_placement -verbose

# checkpoint
set fill_db [make_result_file ${design}_${platform}_fill.db]
write_db $fill_db

################################################################
# Detailed routing

# Run pin access again after inserting diodes and moving cells
pin_access -bottom_routing_layer $min_routing_layer \
        -top_routing_layer $max_routing_layer

set_thread_count [exec getconf _NPROCESSORS_ONLN]
detailed_route -output_drc [make_result_file "${design}_${platform}_route_drc.rpt"] \
            -output_maze [make_result_file "${design}_${platform}_maze.log"] \
            -no_pin_access \
            -save_guide_updates \
            -bottom_routing_layer $min_routing_layer \
            -top_routing_layer $max_routing_layer \
            -verbose 0

write_guides [make_result_file "${design}_${platform}_output_guide.mod"]
set drv_count [detailed_route_num_drvs]
utl::metric "DRT::drv" $drv_count

check_antennas
utl::metric "DRT::ANT::errors" [ant::antenna_violation_count]

set routed_db [make_result_file ${design}_${platform}_route.db]
write_db $routed_db
```

```
set routed_def [make_result_file ${design}_${platform}_route.def]
write_def $routed_def

################################################################
# Extraction

if { $rcx_rules_file != "" } {
  define_process_corner -ext_model_index 0 X
  extract_parasitics -ext_model_file $rcx_rules_file

  set spef_file [make_result_file ${design}_${platform}.spef]
  write_spef $spef_file

  read_spef $spef_file
} else {
  # Use global routing based parasitics inlieu of rc extraction
  estimate_parasitics -global_routing
}

################################################################
# Final Report

report_checks -path_delay min_max -format full_clock_expanded \
  -fields {input_pin slew capacitance} -digits 3
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
report_check_types -max_slew -max_capacitance -max_fanout -violators -digits 3
report_clock_skew -digits 3
report_power -corner $power_corner

report_floating_nets -verbose
report_design_area

utl::metric "DRT::worst_slack_min" [sta::worst_slack -min]
utl::metric "DRT::worst_slack_max" [sta::worst_slack -max]
utl::metric "DRT::tns_max" [sta::total_negative_slack -max]
utl::metric "DRT::clock_skew" [expr abs([sta::worst_clock_skew -setup])]

# slew/cap/fanout slack/limit
utl::metric "DRT::max_slew_slack" [expr [sta::max_slew_check_slack_limit] * 100]
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]
```