RTL2GDS Demo Part 4:

# Place and Route on a simple block

Prof. Adam Teman

EnICS Labs, Bar Ilan University

EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs

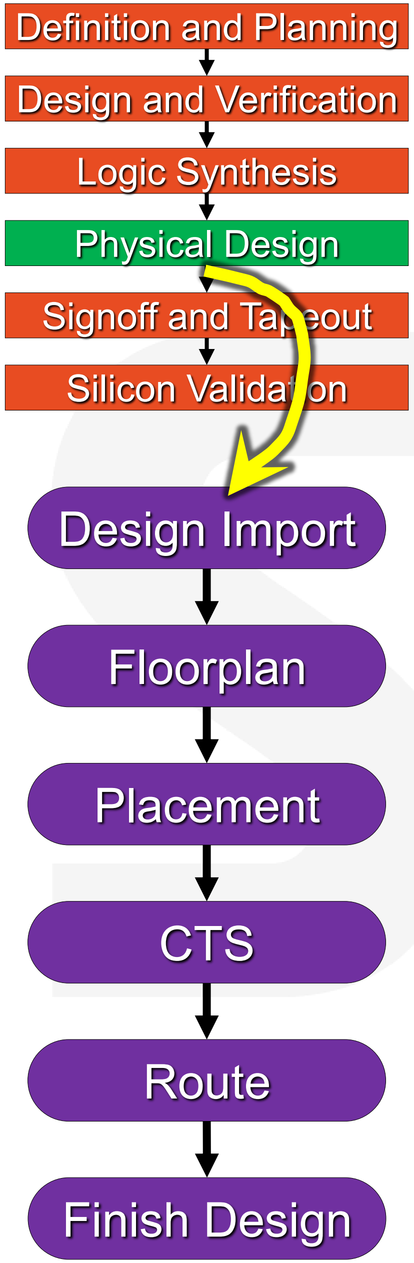
The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

# Introduction

- **At this point we have:**
  - Written our RTL
  - Performed (at least very basic) functional verification
  - Synthesized our design
  - Run gate-level simulation and power estimation

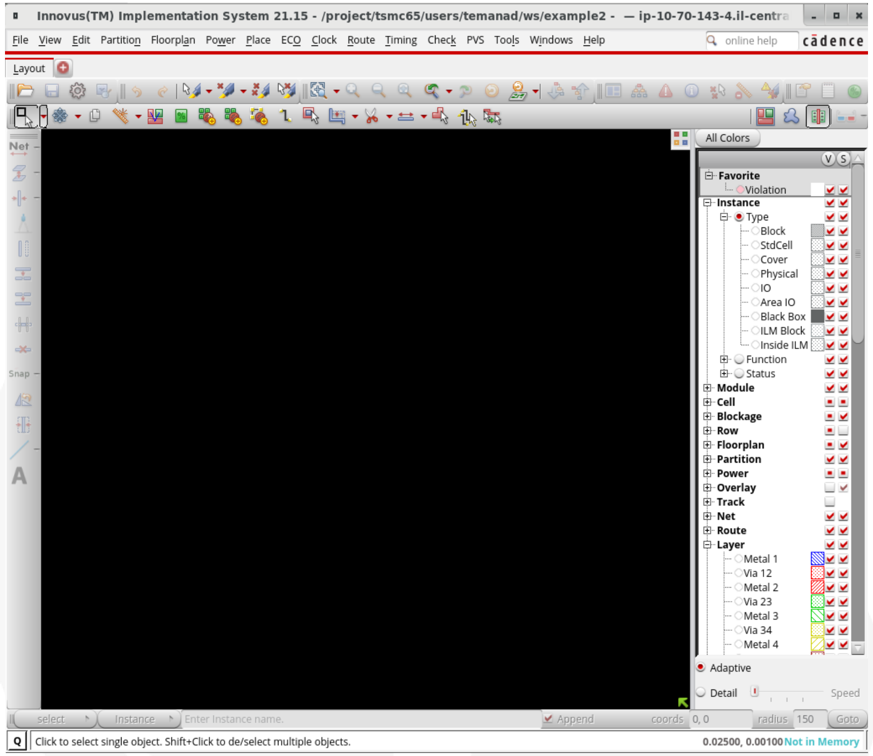
- **It's now time to move on to the physical domain!**

- **And for this, we will be using Cadence Innovus.**



# Cadence Innovus

- Innovus is Cadence's Place and Route Tool, encapsulating several other Cadence Tools:
  - GigaPlace – for timing-driven placement
  - CCOpt – for timing-driven clock tree synthesis
  - NanoRoute – for timing-driven Routing
  - QRC – for signoff quality parasitic extraction
  - Tempus – for signoff quality static timing analysis
  - Voltus – for EM/IR and power estimation



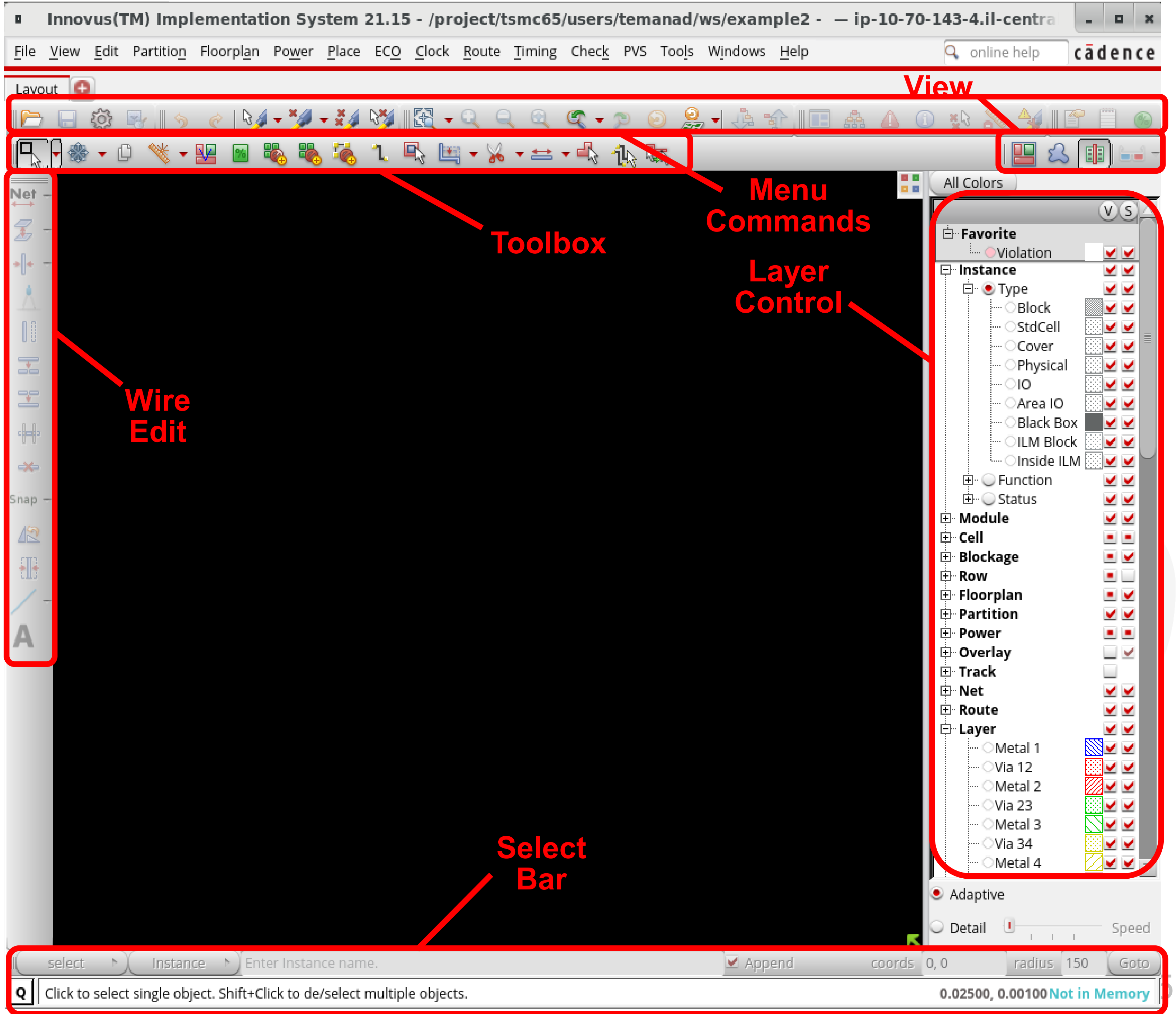

- To start Innovus in Stylus CommonUI mode:

```
%> innovus -stylus
```

- Two files will be created in your working directory:
  `innovus.log` and `innovus.cmd`

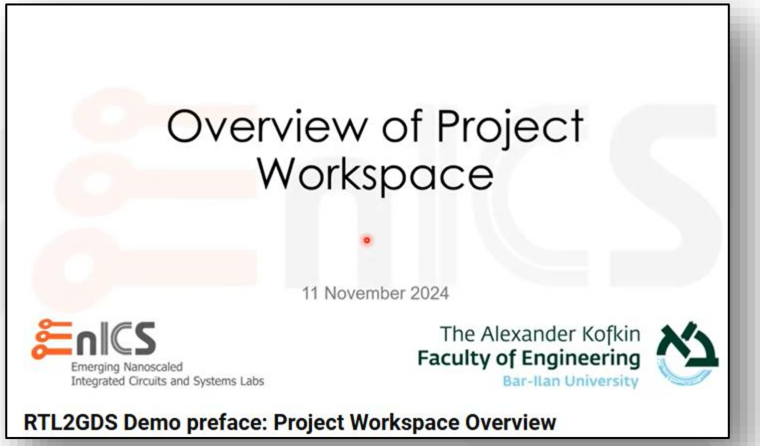
# Innovus GUI

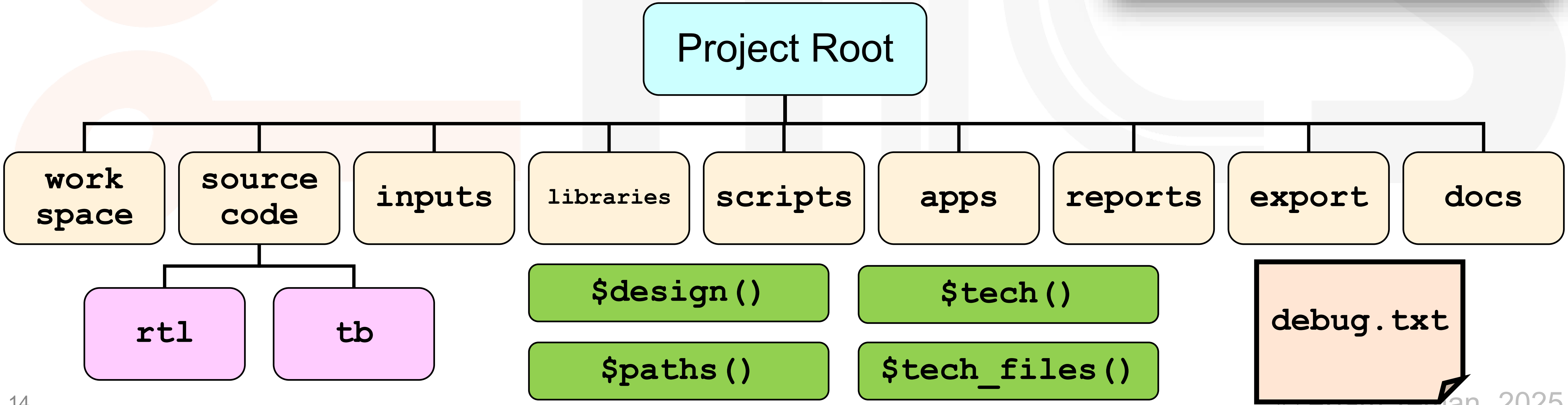
- **Menu Commands**
- **View:**
  - Floorplan
  - Amoeba
  - Placement
- **Layer Control**
- **Toolbox**
- **Select Bar**
- **Wire Edit**



# Reminder: Project Structure

- **As a reminder, our project workspace consists of:**
  - Directories and files, e.g., `inputs`, `libraries`, `scripts`
  - Tcl arrays, e.g., `$design`, `$tech`, `$tech_files`
  - The `debug.txt` file
- **And we run everything from the `workspace/` directory**



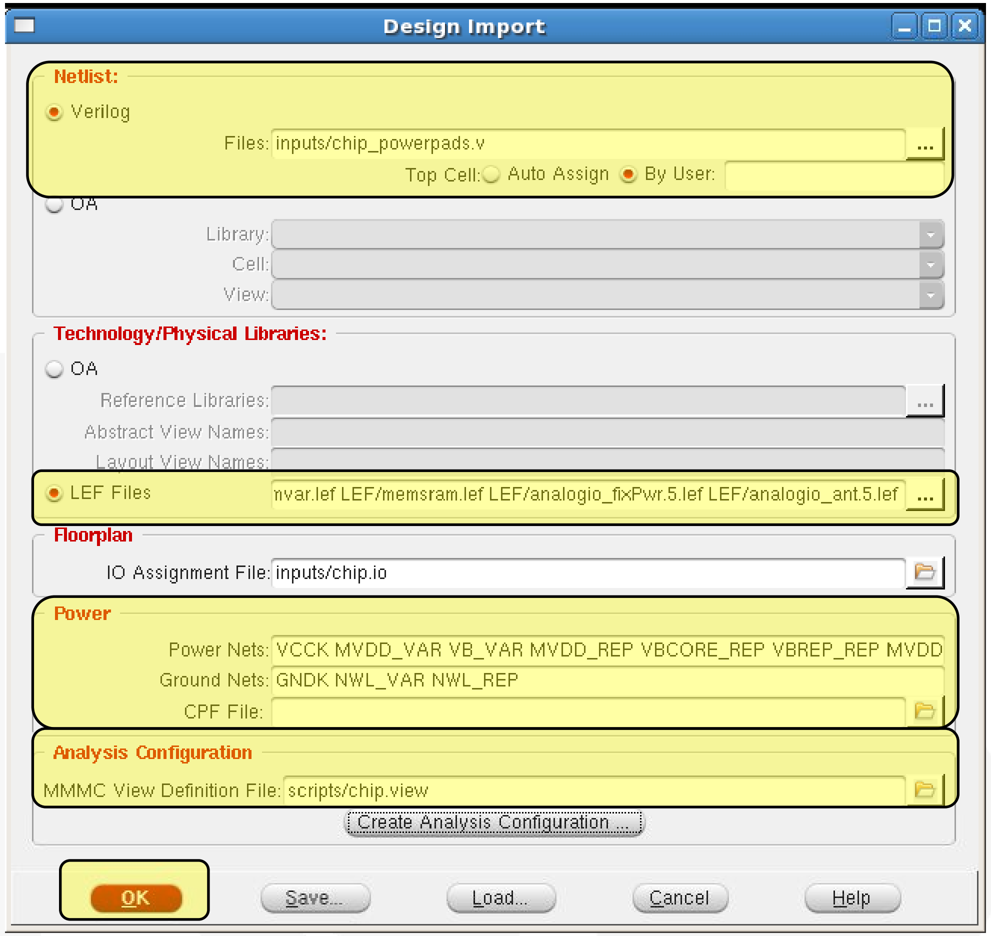
# What do we need to start Place & Route?

| | |
|---|---|
| Design | Our design is the Gatelevel netlist exported from Synthesis |
| Constraints | Design constraints are defined in the SDC files |
| Process Technology | Process Technology is defined in the techlef |
| Process Extraction | Extraction rules provided in the QRCtech or Cap Tables |
| Library Definition | Libraries are defined in .lib files |
| Library Physical Data | Libraries physical data is provided in the .lef files |
| Power/Ground definitions | We need to tell the tool what the power and ground nets are |

MMMC

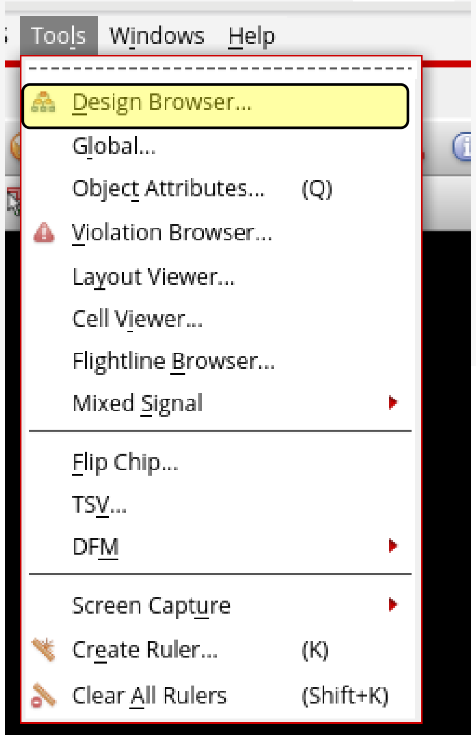
# Importing our Design

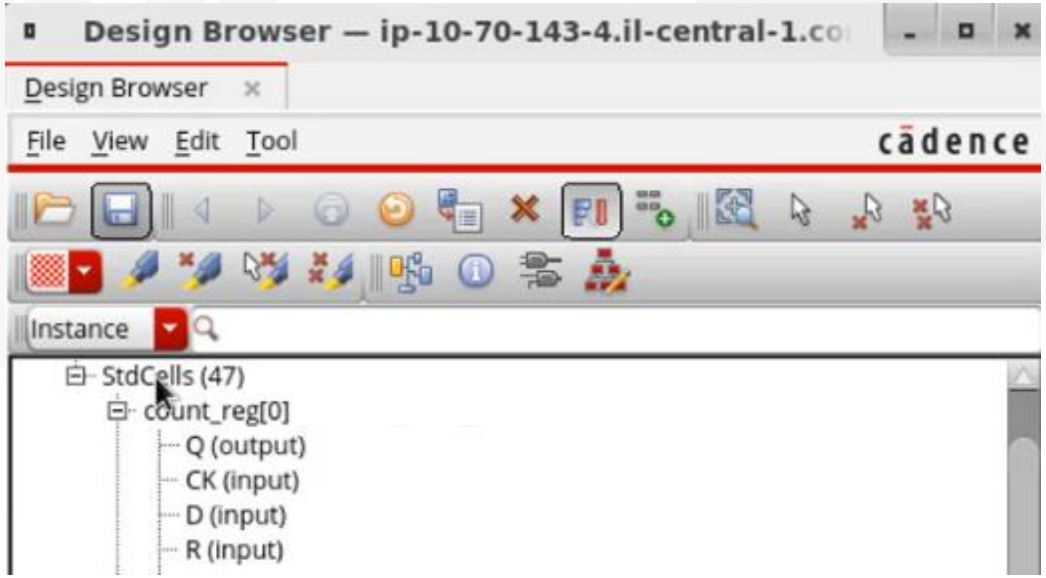
- **All of this data is defined in the Design Import form or by the following commands:**
  - `read_netlist` – reads in the post-synthesis netlist
  - `read_physical` – reads in the `.lef` abstracts of the libraries and techlef.
  - `init_gnd_nets` and `init_pwr_nets` – define global PG net names.
  - For a complicated power management setup, read in `UPF`/`CPF` files, as well.
  - `read_mmmc` – reads in the `.mmmc` file, which includes pointers to the `SDC` file, extraction rules of the process (`QRCtech` files) and `.lib` files of the libraries.
- **And then we can run `init_design` to process all of this data.**



# Design Browser

- **The Design Browser is a very useful tool for browsing your netlist.**
  - After `read_netlist` it will be populated with your design.
- **The Design Browser has many options:**
  - Browse connectivity
  - Select Instances/Nets
  - Zoom to Instances/Nets
  - Highlight Instances/Nets
  - Show Schematics
  - …and more

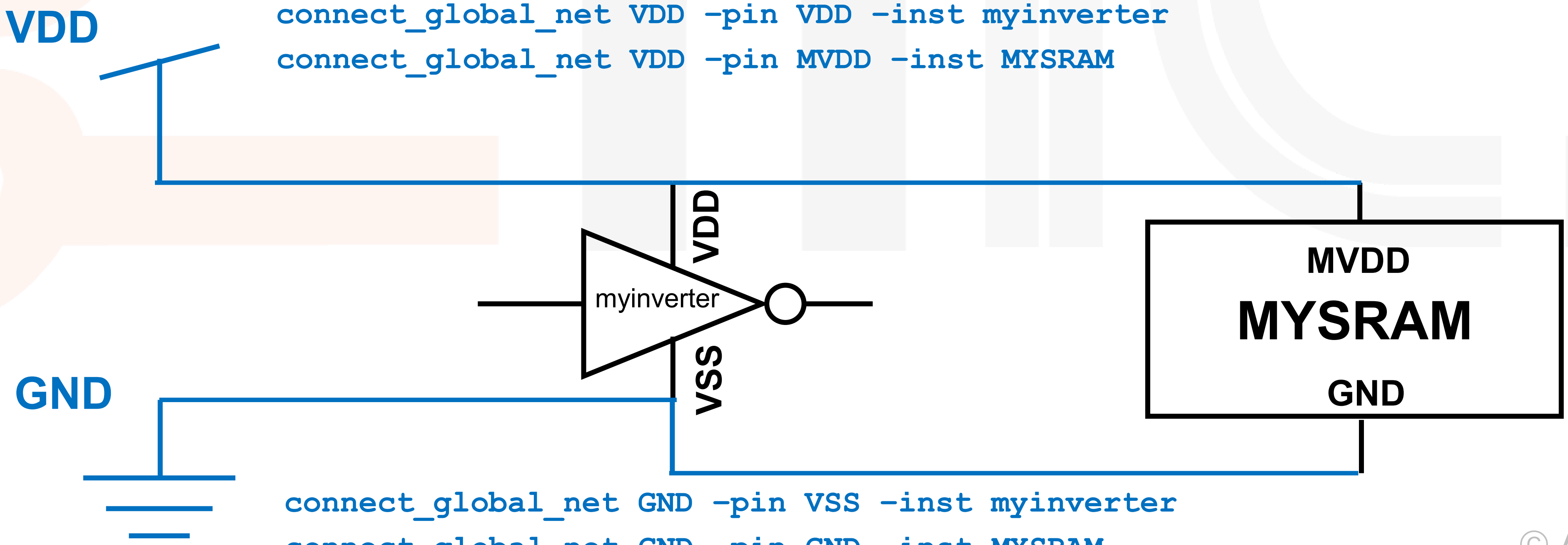


# Global Nets

- **Another very important (and new) concept in physical design is “Global Nets”**
  - During `init_design`, we defined the names of the power and ground nets with the `init_pwr_nets`, `init_gnd_nets` attributes.
  - This created “**PG nets**” but so far, they are not connected to anything.
  - We need to tell the tool which pins on our leaf cells connect to these nets.

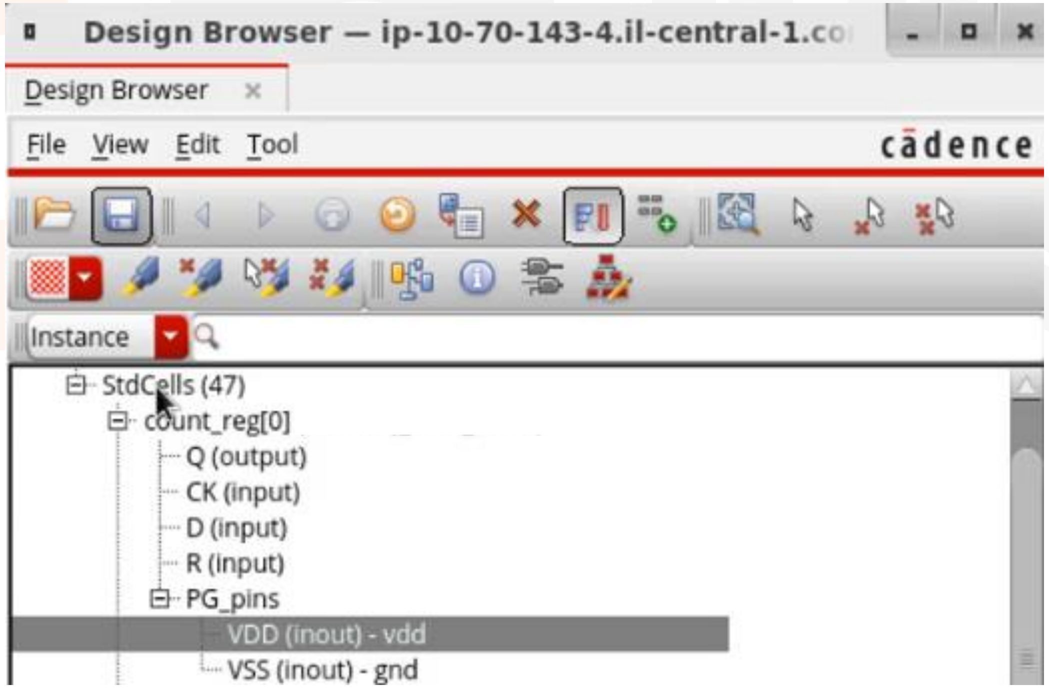
```
connect_global_net VDD -pin VDD -inst myinverter
connect_global_net VDD -pin MVDD -inst MYSRAM
```




```
connect_global_net GND -pin VSS -inst myinverter
connect_global_net GND -pin GND -inst MYSRAM
```

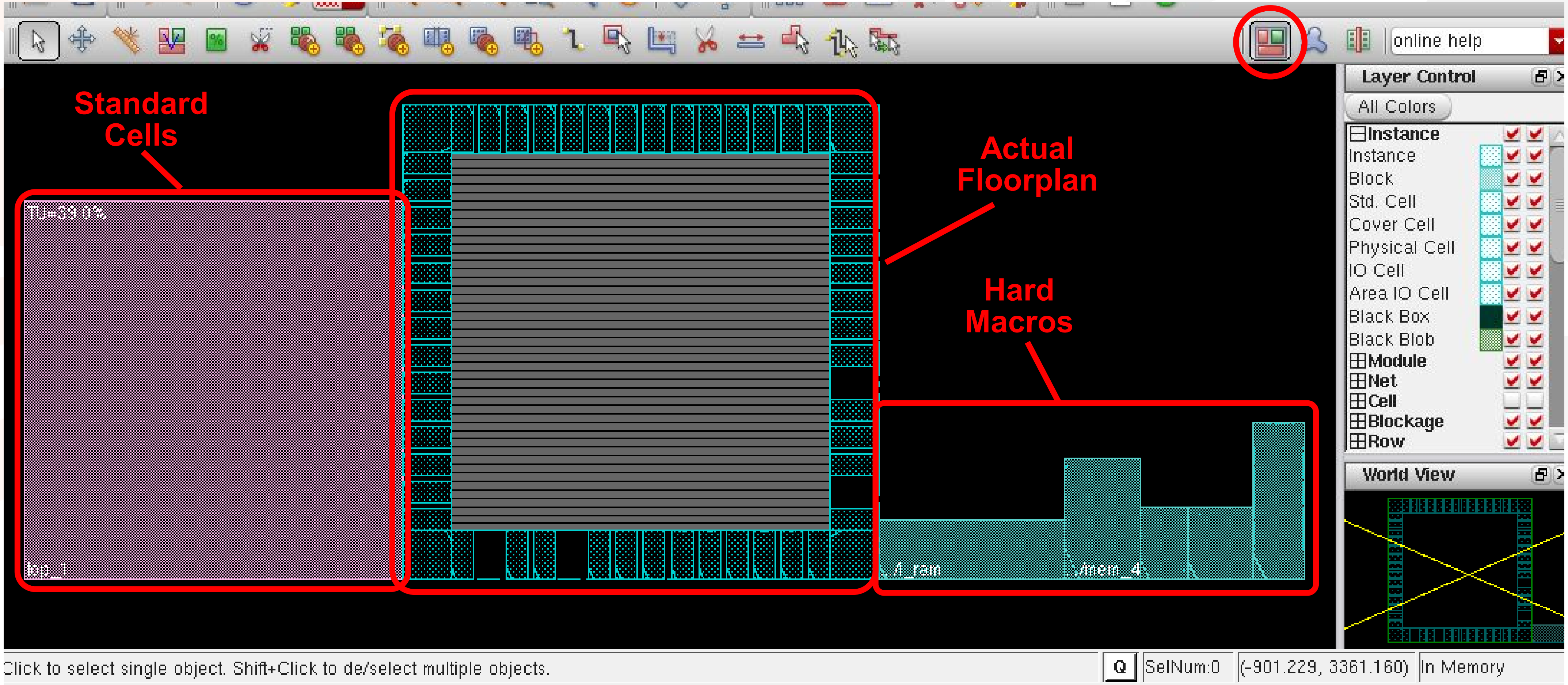
# PG Connections in the Design Browser

- **After init_design, PG_pins will appear in the Design Browser.**
  - At first, these will not be connected
- **The PG_pins are connected by `connect_global_nets`.**
  - Mak sure each instance connects to the correct global nets.



# Floorplanning

- At this point, we can look at the GUI under the "Floorplan View"



# Specify Floorplan

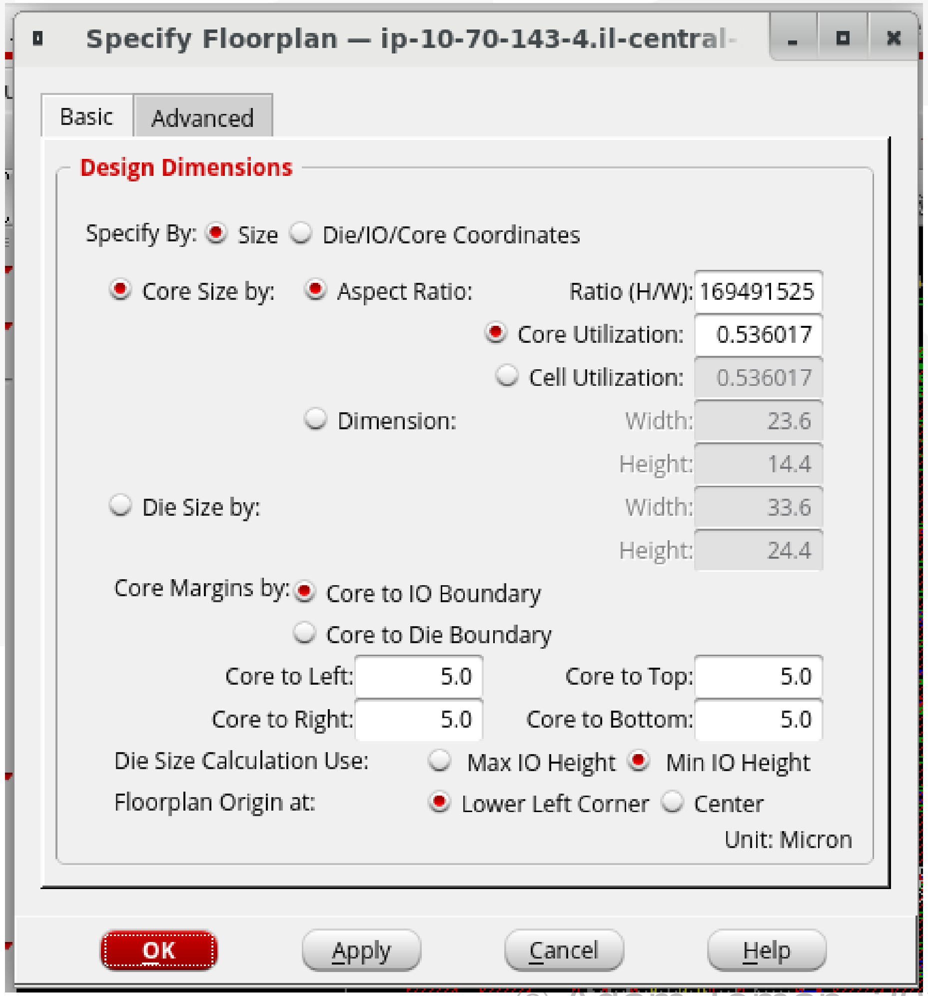
- The `create_floorplan` command or Specify Floorplan form are used to set the size and target utilization of the floorplan.

```
create_floorplan -site CORE -match_to_site \
    -core_density_size 1.0 0.7 5 5 5 5
```

Aspect Ratio

Target Utilization

Core to Boundary

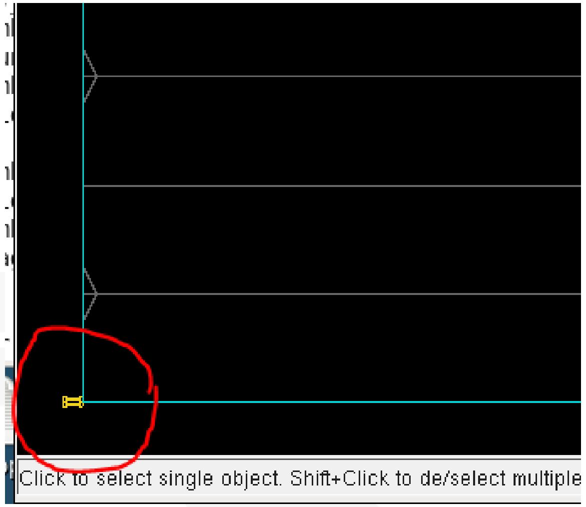


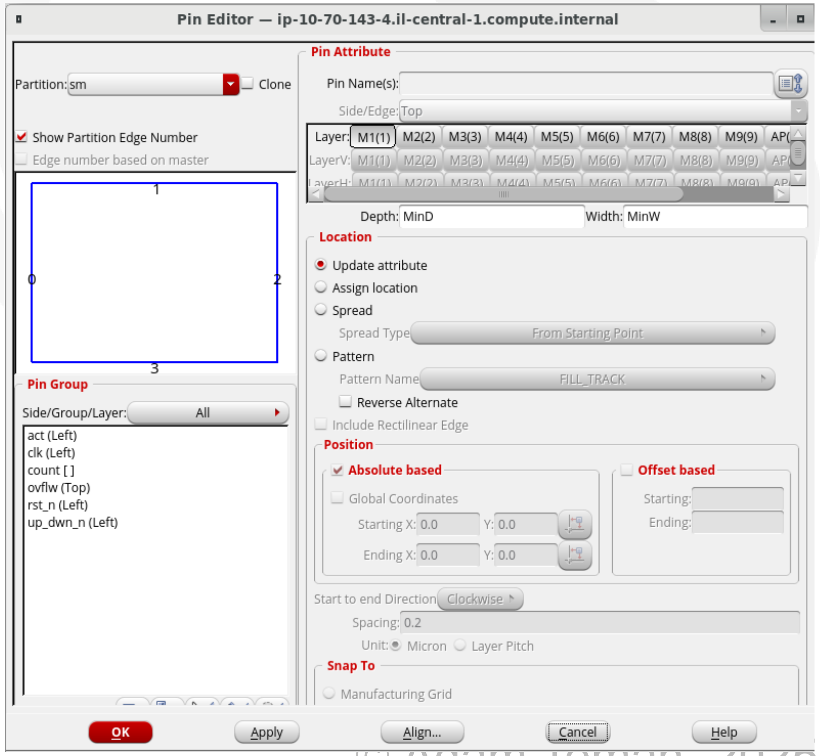
# Pin Location

- **As we saw in the Placement lecture, pin (port) locations have a very significant effect on the Analytic Placement solution.**
  - But where do the locations come from?
  - Well, the default is to put them all in the corner…
- **We have several ways to locate the pins:**
  - Manually move them
  - Use the Pin Editor
  - Use the `edit_pin` command
  - and more functions like "Pin Groups"

```
edit_pin -pin clk -use CLOCK -assign 0.0 3.9 -fix_overlap 1 \
      -side Top -layer 3 -pin_width 0.1 -pin_depth 0.1
```

```
edit_pin -pin $PINS -spread_type side -side Bottom -layer 2 \
      -spread_direction clockwise -fix_overlap 1
```

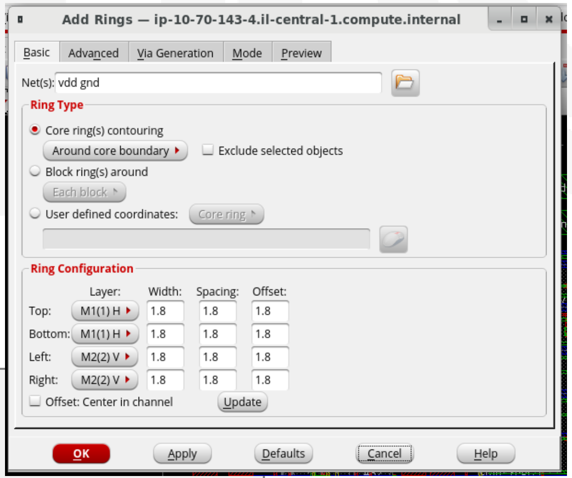


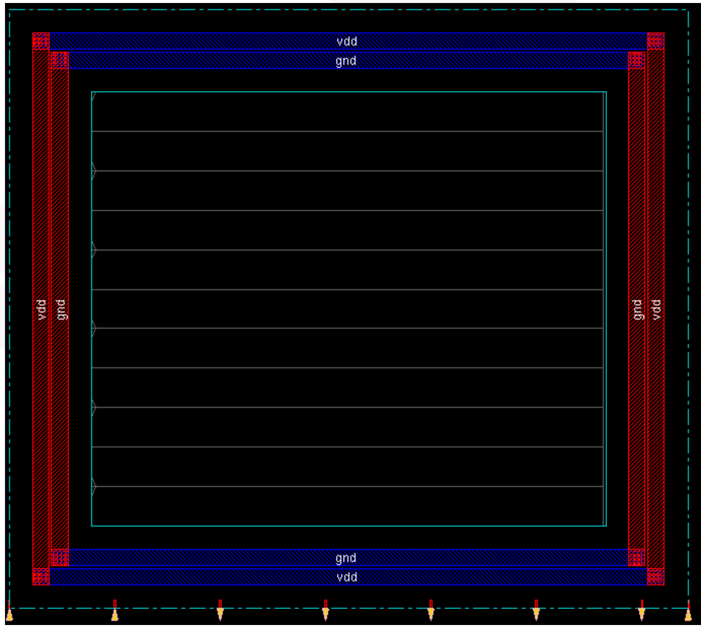



# Power Rings

- **Power distribution is the most significant challenge of floorplanning:**
  - How do we distribute the PG voltages evenly throughout the design?
  - How do we robustly connect (short) all wires of a certain PG net?
- **One common method is to use rings:**
  - Distribute the net around the block or a macro.
  - Provide easy targets for connecting stripes and PG pins.

- **Use the `add_rings` command or form.**



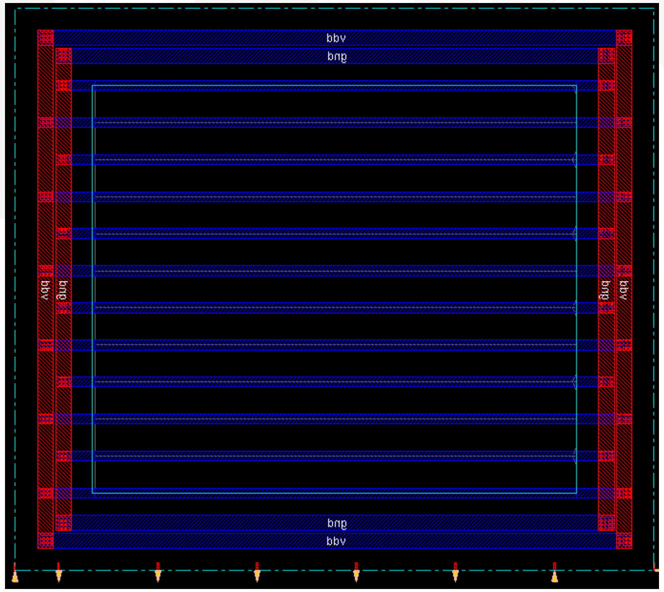



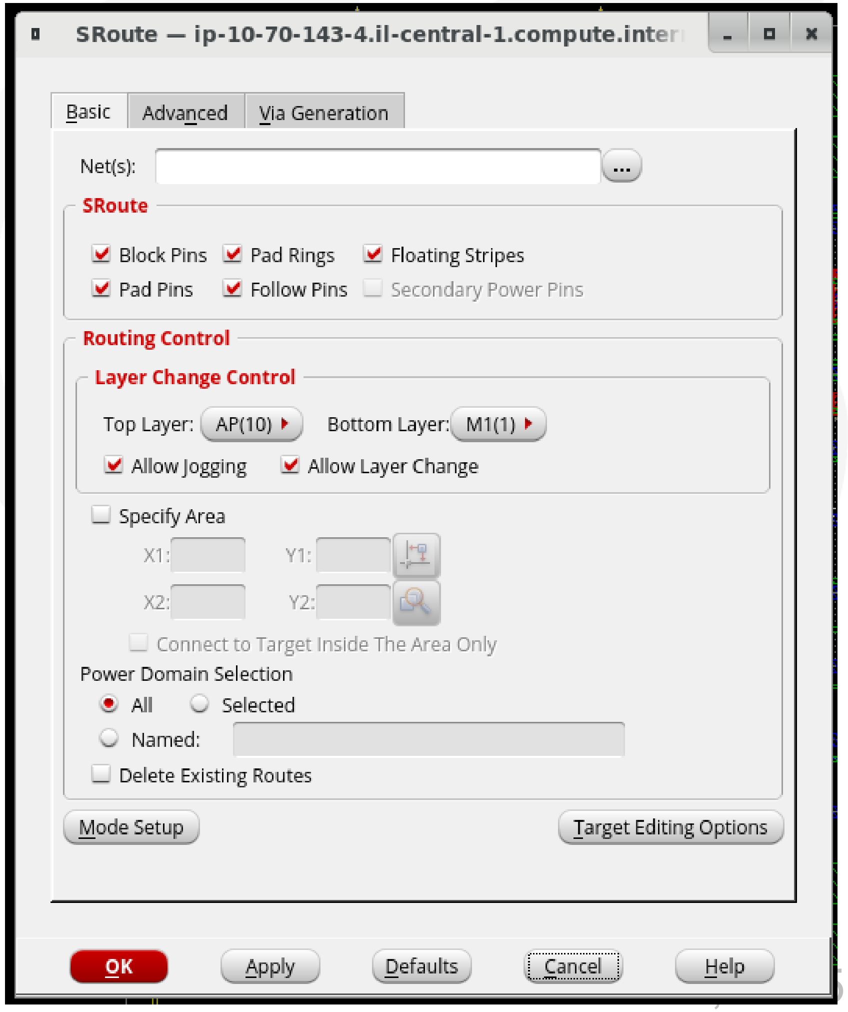

```
add_ring -type core_rings -nets {VDD GND} \
      -width 20 -spacing 1 -offset 0.31 \
      -layer {bottom m1 top m1 right m2 left m2}
```

# Connecting PG Pins

- **The connection of PG Pins in Innovus is called "Special Route":**
  - Routing Follow Pins (M1 VDD/GND rails)
  - Connecting PG Ports to VDD/GND nets.
  - Connecting macro PG pins to VDD/GND nets.
- **This is done with the SRoute form or the `route_special` command**

```
route_special \
      -connect core_pin \
      -nets {VDD GND}
```

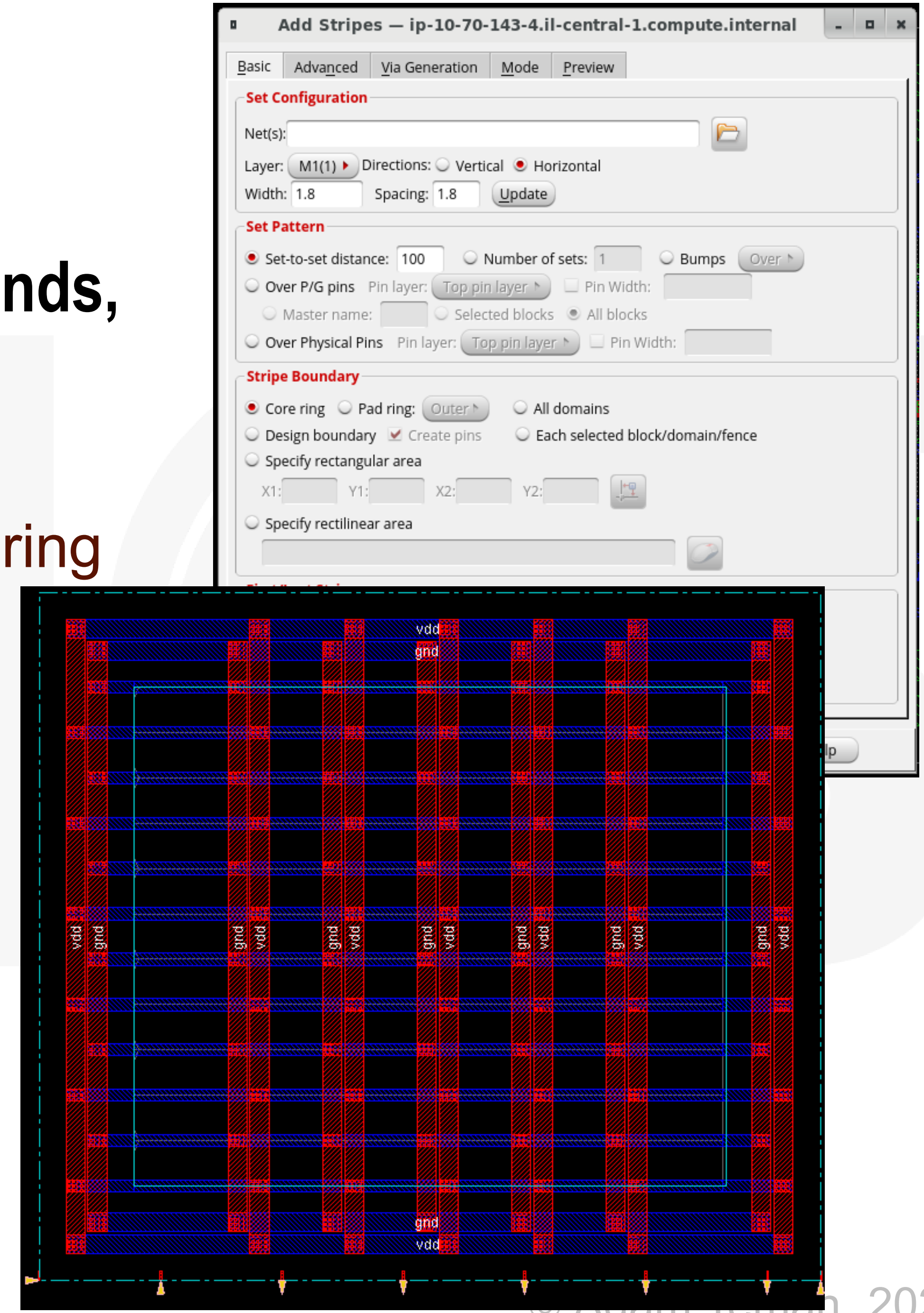




# Creating a Power Grid

- **After running the various Special Route commands, the power and ground rails are connected, e.g.:**
  - Core ring is connected to toplevel PG Pins
  - Follow pins are routed and connected to core ring
- **But for robust power distribution, we need to build a power grid (power mesh).**
- **This is done with the `add_stripes` command**

```
add_stripes -nets {VDD GND} \
      -layer metal2 \
      -width 5 -spacing 0.32 \
      -start_from_left \
      -start_offset 100 \
      -set_to_set_distance 100
```



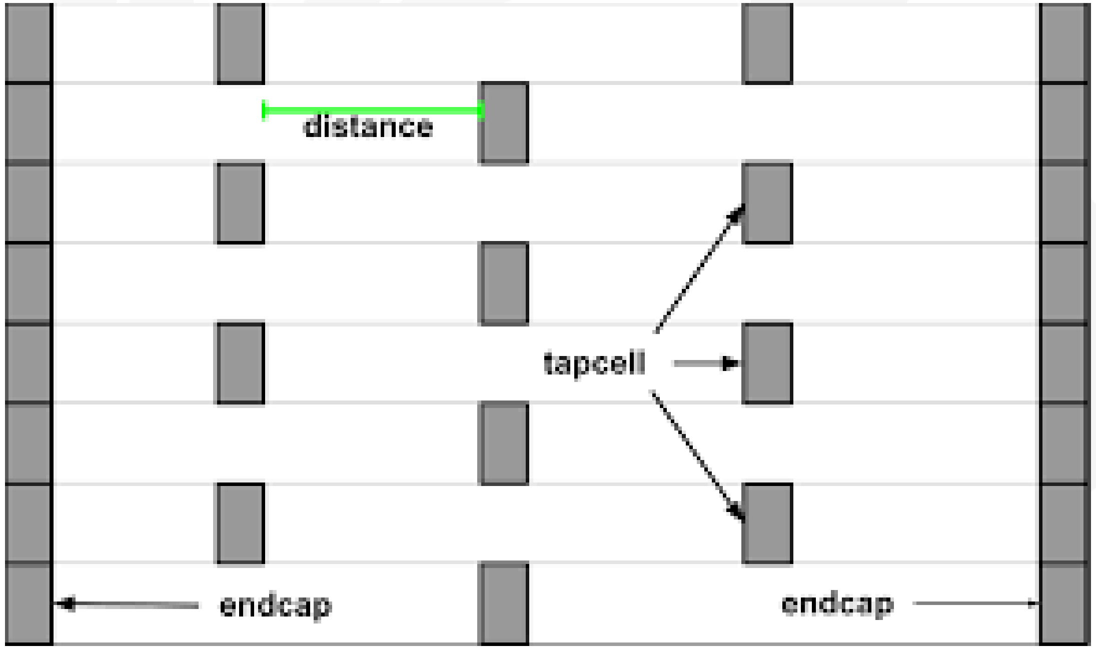
# Some notes about Power Grids

- **The `add_stripes` command is a very powerful command:**
  - It can create PG stripes according to many parameters
  - It will automatically drop via arrays between stripes on different layers
  - It will connect to macro PG pins when possible
- **But how do we know how much power routing is needed?**
  - Which layers? How wide? What offset?
- **These are (*multi-*) million dollar questions…**
  - *Golden answer*: As many PG stripes as possible while enabling signal routing.
  - *Reality*: Guess and Iterate.
    - Create a power mesh and run through the flow checking routability.
    - Run EM/IR analysis (e.g., with Voltus) to expose problems.

# End Caps and Well Taps

- **End Caps (or boundary cells):**
  - Are cells required at the end of standard cell rows (and in advanced processes, at the top, bottom, corners, etc.)
  - Include POLY, OD, NWELL and other layers for DRC and density.
- **Well Taps (or Tap Cells):**
  - Are cells that connect VDD/GND to NWELL/PWELL to prevent latch up.
  - DRC rules require tap cells every several microns.

```
set_db add_endcaps_left_edge $ENDCAP_LEFT
set_db add_endcaps_right_edge $ENDCAP_RIGHT
add_endcaps -prefix ENDCAP
```




*Source: OpenRoad*

```
add_well_taps -cell $WELLTAPCELL \
        -skip_row 1 -prefix WELLTAP \
        -in_row_offset 3 -cell_interval 10
```

```
check_well_taps -max_distance 20
```

# Check DRC

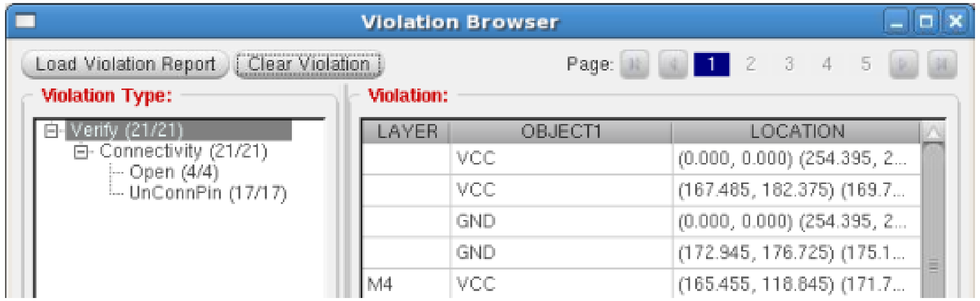
- **Innovus has a built in DRC checker, activated with the `check_drc` command or Verify DRC form.**

```
@innovus 77> check_drc
 *** Starting Verify DRC (MEM: 858.2) ***

  VERIFY DRC ...... Starting Verification
  VERIFY DRC ...... Initializing
  VERIFY DRC ...... Deleting Existing Violations
  VERIFY DRC ...... Creating Sub-Areas
  VERIFY DRC ...... Using new threading
  VERIFY DRC ...... Sub-Area: {0.000 0.000 31.800 29.200} 1 of 1
  VERIFY DRC ...... Sub-Area : 1 complete 4 Viols.

  Verification Complete : 4 Viols.

 *** End Verify DRC (CPU: 0:00:00.0  ELAPSED TIME: 0.00  MEM: 0.0M) ***
```


  - Note: DRC checking is based on the TechLEF. It is not signoff quality DRC checking!
  - Explore DRCs with the Violation Browser
  - Clear DRCs with `delete_drc_markers`

```
delete_drc_markers
```

# Check Connectivity

- **Similarly, Innovus has a type of LVS checker, called `check_connectivity`**
  - Primarily reports opens and shorts
  - Based on abstracts (LEFs) so only checks pin connectivity
  - Can sometimes report confusing errors. Make sure appropriate flags are used.

```
@innovus 78> check_connectivity -type special
VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Sun Nov  4 22:17:59 2018

Design Name: sm
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (31.8000, 29.2000)
Error Limit = 1000; Warning Limit = 50
Check specified nets

Begin Summary
  Found no problems or warnings.
End Summary
End Time: Sun Nov  4 22:17:59 2018
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)
```

# Placement

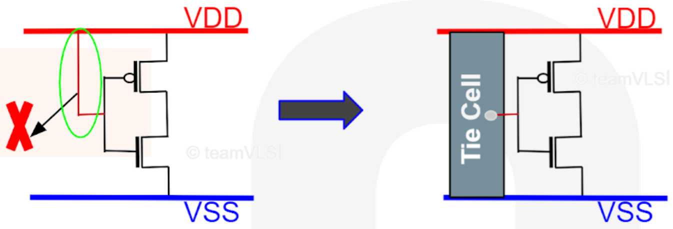
- **At this point, our floorplan is ready:**
  - Block (or chip) dimensions and pins (or I/Os) are set.
  - Macros (primarily SRAMs) are fixed.
  - Power grid is routed.
  - Additional guides (e.g., fences, feedthroughs, pre-routes) are defined.
- **We can now proceed to placement**
  - Innovus runs concurrent placement and timing optimization with "GigaPlace"

```
place_opt_design
```

- **Placement runs early global route (a.k.a. Trial Route) for parasitic estimation.**
  - At this stage you should check congestion to see expected hotspots.

# Post Placement

- **Now we have provided a location to all standard cells, ran global route for parasitic estimation and applied timing optimization.**
- **To finalize the placement stage, we need to:**
  - Add Tie Cells

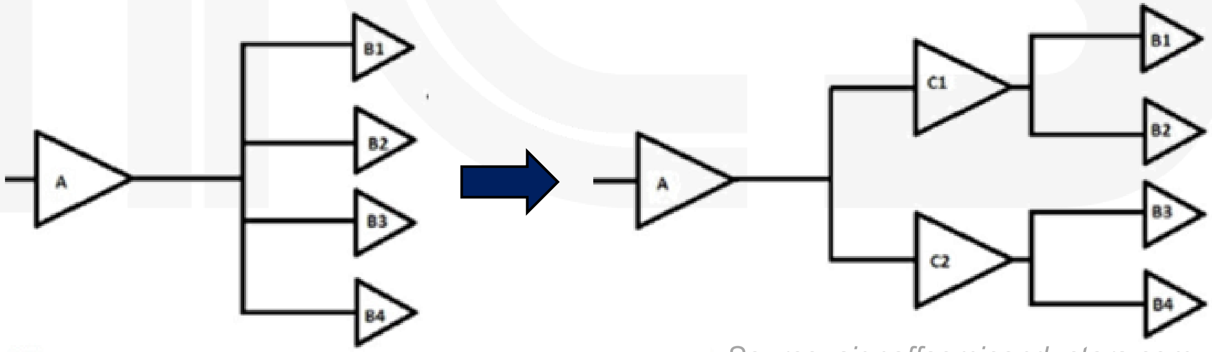



Source: Team VLSI

```
set_db add_tieoffs_cells "TIE0 TIE1"
set_db add_tieoffs_max_fanout 20
set_db add_tieoffs_max_distance 250
add_tieoffs
place_detail
```

  - Buffer High Fanout Nets (Reset Tree)
  - Additional timing optimization

```
set_db opt_fix_fanout_load true
opt_design -pre_cts
```



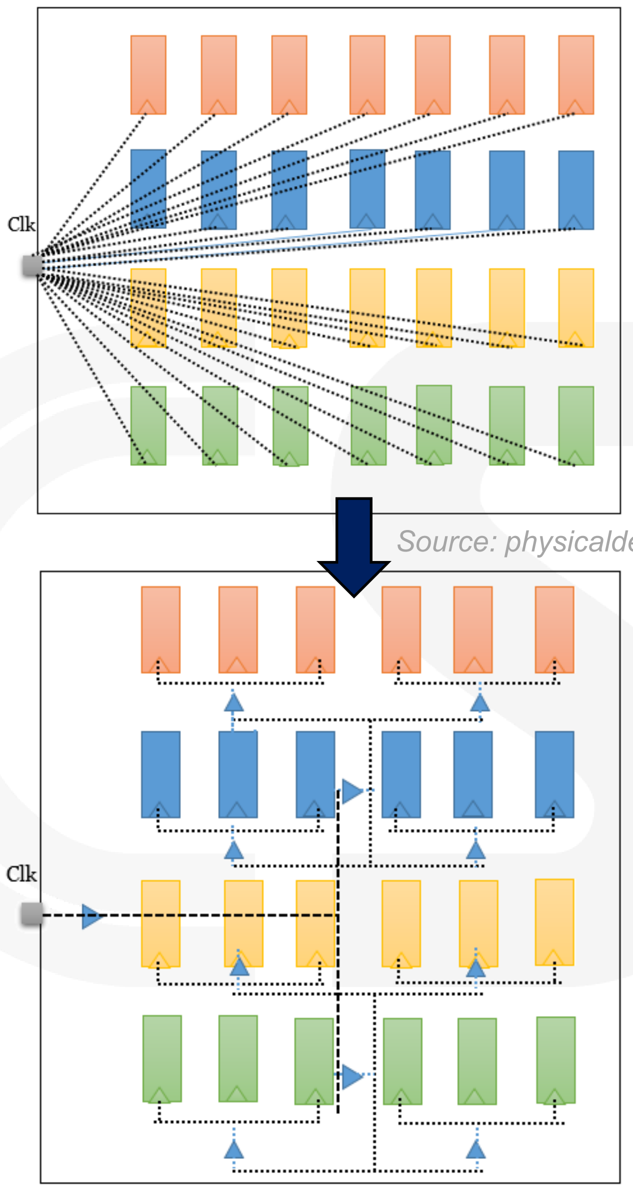

Source: signoffsemiconductors.com

- **Any changes in cell location require incremental Placement Legalization**

```
place_inst G123 34.0 52.0 -fixed
place_detail -eco true
```

# Clock Tree Synthesis

- **At this point, the design is fully placed, which means:**
  - Location of the clock source(s) is fixed.
  - Location of clock pins of macros is fixed.
  - Location of clock pins of registers (flip flops) is fixed.
  - Clock elements (e.g., ICGs, clock muxes) are placed.
- **Therefore, we can continue to clock tree synthesis.**
- **CTS has many definitions and constraints:**
  - These are highly correlated with SDC definitions.
  - e.g., clock source, clock frequency, constants
- **Therefore, we can automatically generate an initial clock tree spec script.**



Source: physicaldesign4u

```
create_clock_tree_spec -out_file <filename>
```

# Clock Tree Synthesis

- **Specifically, the clock tree spec includes (among others):**
  - Definition of "clock trees" and "skew groups"

```
create_clock_tree -name clock -source CLK -no_skew_group
create_skew_group -name clock -sources CLK -auto_sinks
```

  - Definition of CTS DRVs

```
set_db cts_max_fanout 20
set_db cts_target_max_transition_time 0.1
set_db cts_target_max_capacitance 0.1
```

  - Adding additional sinks to a skew group

```
update_skew_group -skew_group clock -add_sinks $pins
```

  - Defining stop, ignore and exclude pins

```
set_db pin:$pin .cts_sink_type stop / ignore / exclude
```

  - Defining insertion delay (float) pins

```
set_db pin:mem1/CK .cts_pin_insertion_delay 1.2ns
```

# Non-Default Rules for Clock Routing


- Clock nets get priority by routing them during CTS
- Non-default rules (NDRs), such as double-spacing, double-width and shielding are often applied to clock nets.
- For NDRs, Innovus divides clock nets into three categories:
  - Top: The net driven by the main clock input/source
  - Trunk: The major clock nets with a large fanout
  - Leaf: The minor clock nets with a small fanout
- NDRs are defined with three commands:
  - `create_route_rule`
  - `create_route_type`
  - `set_db cts_route_type_`

```
create_route_rule \
    -name CTS_2S2W \
    -spacing_multiplier 2 \
    -width_multiplier 2
```

```
set_db cts_top_fanout_threshold 10000
set_db cts_route_type_leaf leaf_rule
set_db cts_route_type_trunk trunk_rule
set_db cts_route_type_top top_rule
```

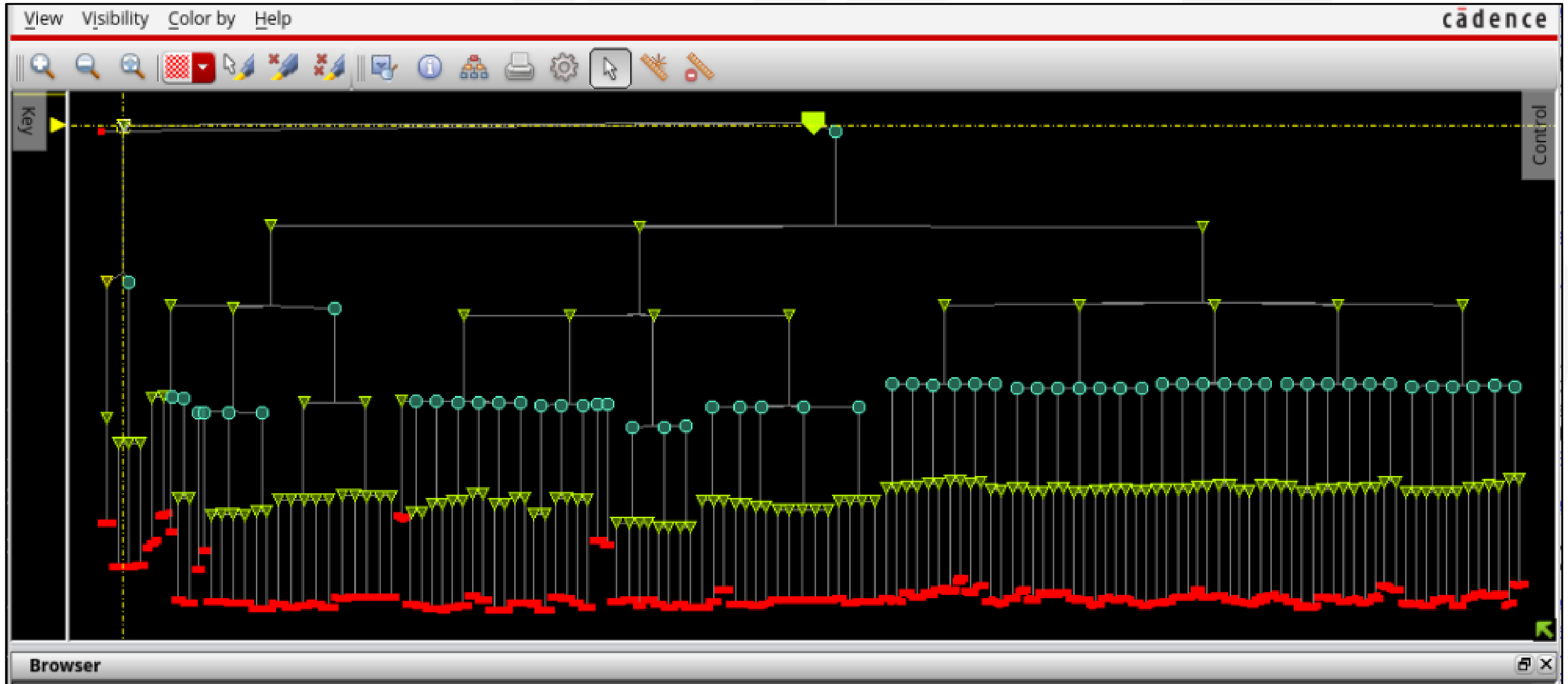
```
create_route_type -name leaf_rule -non_default_rule CTS_2W1S \
    -top_preferred_layer M5 -bottom_preferred_layer M4
create_route_type -name trunk_rule -non_default_rule CTS_2W2S \
    -top_preferred_layer M7 -bottom_preferred_layer M6 \
    -shield_net VSS -bottom_shield_layer M6
create_route_type -name top_rule -non_default_rule CTS_2W2S \
    -top_preferred_layer M9 -bottom_preferred_layer M8 \
    -shield_net VSS -bottom_shield_layer M8
```

# Running and Debugging CTS

- CTS is run with the `ccopt_design` command:

```
reset_ccopt_config
source my_clock_tree_spec.tcl
ccopt_design
```

- After running CCopt, use the clock tree debugger to analyze the clock tree:




| Analysis View | Skew Group | Skew | Min Delay | Max Delay | Min Pin | MinPath Level | Max Pin | MaxPath Level |
|---|---|---|---|---|---|---|---|---|
| bc_dly_corner:hold.early | | | | | | | | |
| | clk | 0.047 | 0.180 | 0.227 | ...it_i/CS_reg[1]/CK | 6 | ...eg_reg[4][12]/CK | 7 |
| bc_dly_corner:hold.late | | | | | | | | |

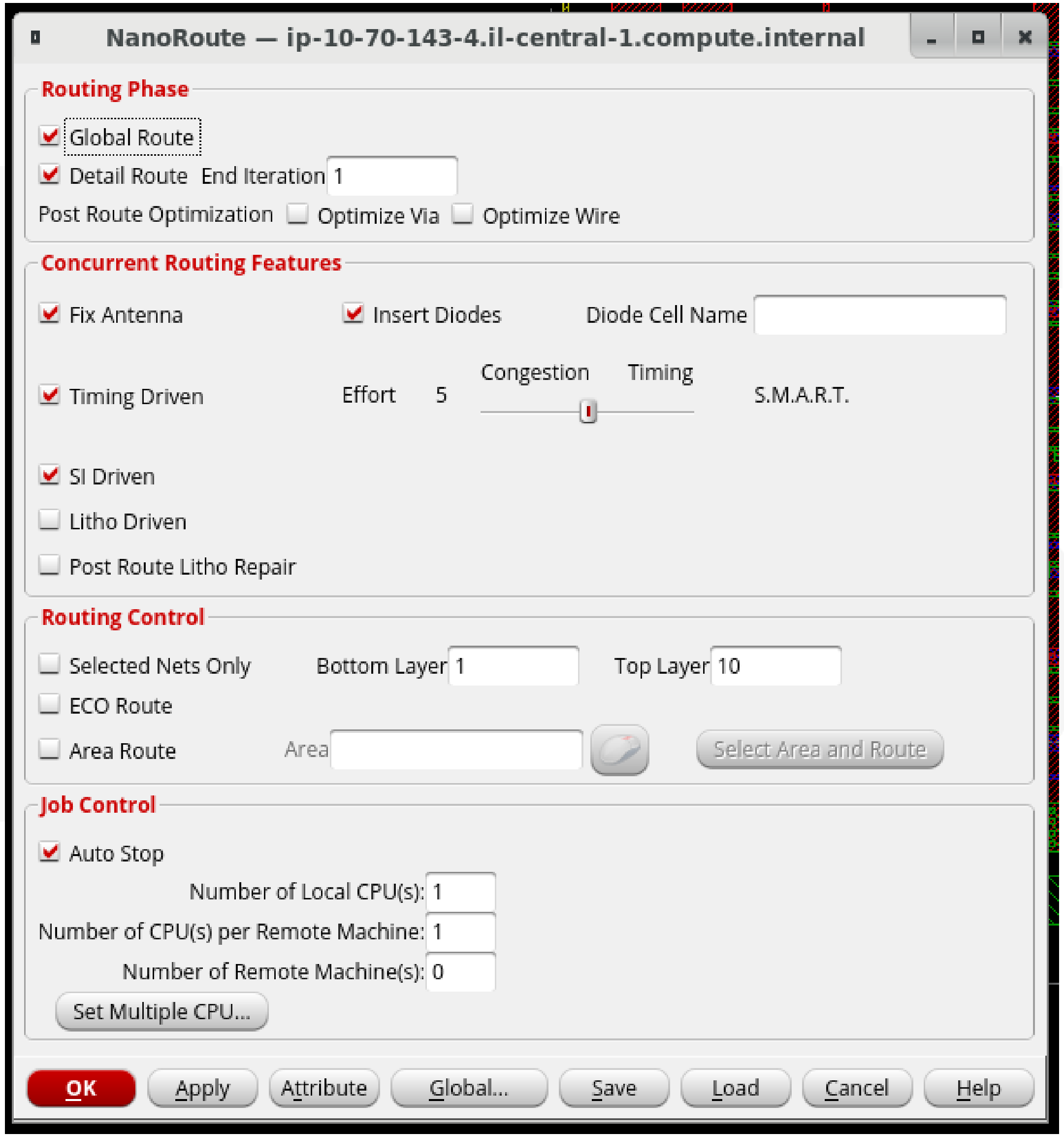
- Hold optimization can be run following CTS:

```
opt_design -post_cts -hold
```

# Routing

- **At this point:**
  - All cells have been placed.
  - Clock buffers have been inserted and clock tree has been routed.
  - Timing (setup and hold) is evaluated based on global route parasitics.
- **It is now time to route all of the signal nets:**

```
set_db route_with_timing_driven true
set_db route_with_si_driven true
route_opt_design
```



# Post-Route and Signoff

- **At this point we can run some post-route optimizations for timing and DFM:**
  - Post-route timing optimization: `opt_design -post_route -setup -hold`
  - Wire optimization (straightening, widening, spreading) `route_design -wire_opt`
  - Via optimization (via reduction, multi-cut) `route_design -via_opt`
- **Prepare for signoff:**
  - Add fillers `add_fillers;    route_eco -fix_drc`
  - DRC/LVS `check_drc;    check_connectivity`
- **Export Design**
  - Export Verilog netlist `write_netlist final_netlist.v`
  - Export SDF `write_sdf final.sdf`
  - Export GDS `write_stream final.gds`

# Summary

- **In this demonstration, we covered a full block-level Place and Route flow:**
  - Imported the design to Innovus
  - Created a floorplan
  - Ran standard cell placement
  - Synthesized the clock tree
  - Routed the design
  - Ran post-route and signoff
- **This was a simple design, but we showed a lot of the functionalities of the place and route tool and demonstrated the complete flow.**
- **Next, we go over a larger, more complex design to show more capabilities.**