

Introduction to Physical Implementation

Using Cadence Innovus and Stylus Common UI

This guide will take you through the place and route flow in Cadence Innovus.

1 Workspace Setup

The workspace setup is overviewed in the document “Overview of Project Workspace”, provided separately.

2 Getting Started with Innovus

Innovus is Cadence’s Place and Route tool that replaced “Encounter Design Implementation (EDI)”. This manual will introduce you to Innovus using the new **Stylus Common UI** commands. This means that anything that was written for Encounter “in legacy UI mode” will not work, so pay attention when reading documentation and support answers.

2.1 Starting Innovus

To start **Innovus** from a Linux command line:

```
~/workspace> innovus -stylus
```

The command prompt will now change to the Innovus Shell:

```
@[DEV] innovus 1>
```

2.1.1 Log files

Innovus will automatically create two log files:

- **innovus.log** : the log of all the messages printed to standard output.
- **innovus.cmd** : the log of all the command run in the current session.

You can change the name of the log file, using the “Log” option when invoking **innovus**.

Always search for **warnings** and **errors** in the log file to make sure you didn’t miss anything important. A good approach to do this is using the Linux **grep** command:

```
grep -ei "warning\|error" innovus.log | less
```

It is very useful to use the **.cmd** file to see which commands were run (especially if choosing a GUI menu option) and copying them to a script file.

Note that there is a GUI tool for viewing the log file that shows all the Errors and Warnings very clearly. To use it, type **view_log**

```
view_log -file_name innovus.log
```

2.1.2 Tool Command Language (TCL)

The Innovus Shell interprets TCL language. It is highly advised to learn some TCL, since you’ll be scripting quite a bit.

When sourcing an external TCL script file, by default, verbose logging is set. In other words, the whole script is printed to the log file. To turn this off change the **source_verbose** attribute to **false**.

```
set_db source_verbose false
```

2.1.3 Getting Help

To get information about a specific command, use the `man` command:

```
man <command name>
```

To find commands that match a certain string, use the `help` command:

```
help <glob string>
```

Note that help can also be run on an attribute!

2.1.4 Documentation

In addition, you can find the documentation under the `doc` directory of the installation path. The installation path is saved in the `$GENUSHOME` environment variable:

```
%> echo $INNOVUSHOME
/opt/cadence/INNOVUS/1714
%> acread $INNOVUSHOME/doc/UGcom/UGcom.pdf &
```

The most important documents are:

- Innovus Stylus CUI User Guide: [\\$INNOVUSHOME/doc/UGcom/UGcom.pdf](#)
- Innovus Stylus CUI Command Reference: [\\$INNOVUSHOME/doc/TCRcom/TCRcom.pdf](#)
- Innovus Stylus CUI Database Command Reference: [\\$INNOVUSHOME/doc/DBcom/DBcom.pdf](#)

To learn about an error/warning/info message:

```
man <message_name>
```

To open up Cadence `help` tool, from the Linux shell, type:

```
cdnshelp
```

Finally, you will find the best answers on the Cadence support site: support.cadence.com

3 Design Import

The first step, when moving from Synthesis to P&R is to import the design. This can be set up and run the first time with the **FILE→IMPORT DESIGN** form, but it is much better to run the step-by-step design import commands, as they will allow you to debug the design import along the way and afterwards, easily repeat design import after any changes.

Design import includes several parts:

1. Reading in the Multi-Mode Multi-Corner view file, which defines corners, libraries, etc.: `read_mmmc`
2. Reading in the technology definitions and LEF abstracts of the libraries: `read_physical -lef`
3. Reading in the post-synthesis netlist. `read_netlist`
4. Reading in the I/O definition file: `read_io_file` (only for a full-chip toplevel implementation)
5. Defining the global power nets: `set_db init_ground/power_nets`

The design import process also includes definition of advanced power intent, using CPF or UPF definitions, but this is beyond the scope of this course.

In the following subsections, the commands for each of these steps will be described. However, if you choose the GUI route, the design import form is shown in Figure 1 with some example values.

Figure 1: Design Import Form

3.1 Defining the Multi-Mode Multi-Corner (MMMC) settings

Multi-Mode Multi-Corner (MMMC) is the framework for defining the analysis views that will be used for setup and hold timing throughout the flow. This includes definition of design constraints (SDC), operating conditions (corners), timing libraries and more. A full explanation of MMC settings is given in Section

3.10. Once the MMMC settings are defined within a *view definition* file, they are read in with the `read_mmmc` command.

```
read_mmmc ../inputs/$TOP.view
```

This is set on the **DESIGN IMPORT** form in the **MMMC VIEW DEFINITION FILE** field.

Following the `read_mmmc` command, you can check the library consistency by running:

```
check_library
```

3.2 Defining the library cell abstracts and technology rules (LEF files)

Geometric information about both the technology and the hard macros is provided to Innovus as abstracts, most commonly in the .lef format. Two types of .lef information are required:

- **Tech-LEF:** The technology LEF provides information about the place and route constraints, such as routing layers, vias, and design rules. This is the information that Innovus uses to run DRC.
- **Library Cell Abstracts:** These are abstracts that include the size, pin location, and blockages in hard macros, such as standard cells, I/Os, memories, etc.

Each IP Library usually provides a separate .lef file. The Tech-LEF can either be provided in a separate file or as part of the standard cell .lef. A list of all the .lef files should be passed to Innovus with the `read_physical -lef` command:

```
read_physical -lef “../inputs/technology.lef ../inputs/standard_cells.lef”
```

Make sure that the first .lef file in the list is the Tech-LEF, or else Innovus will not understand the layers used in the library cell definitions.

After running this command, you can check the list of .lef files by querying the `init_lef_files` attribute.

This is set on the **DESIGN IMPORT** form in the **TECHNOLOGY/LEF FILES** field.

3.3 Reading in the post-synthesis netlist

Place and route is initialized after the behavioral RTL has already been synthesized into a structural gate-level netlist. This netlist is read in using the `read_netlist` command.

```
read_netlist ../exports/$TOP.postsyn.v
```

This is set on the **DESIGN IMPORT** form in the **NETLIST/VERILOG** field.

3.4 Reading in the I/O definition file

If you are implementing the top-level of a design (i.e., full-chip), you will want to provide the locations of the I/O cells, which often surround the perimeter of the chip. This can be done with the Cadence I/O definition file format, which is overviewed in Section 12.3. After writing an I/O definition file, read it in with the `read_io_file` command:

```
read_io_file ../inputs/$TOP.io
```

This is set on the **DESIGN IMPORT** form in the **IO ASSIGNMENT FILE** field.

3.5 Defining the global power nets

Power nets are special nets (referred to as **pg_nets** in Innovus) and they need to be defined as *global nets* during design initialization. Just to emphasize the need for such global nets, different library cells may have different internal names for the power nets (e.g., VDD, VCC, etc.), but when instantiating them, we must have a way of marking that they are all connected to the same net. There can be many power/ground nets in the design, but at least one power and one ground net, defined with the **init_ground_nets** and **init_power_nets** database attributes.

```
set_db init_power_nets "VDD1 VDD2"
set_db init_ground_nets "GND"
```

This is set on the **DESIGN IMPORT** form in the **POWER** field.

Note that if a power intent file is read in (CPF or UPF), the global nets are usually defined inside it.

3.6 Additional Design Import Options

Some additional design import options that you may not be using at this point include:

- **read_power_intent**: Loads a CPF or UPF power intent file.
- **read_def**: Loads a Design Exchange Format (DEF) file with pre-existing floorplan definitions.
- **set_db init_no_new_assigns true**: Ensures that no assigns are included in the netlist if some tool cannot support them.
- **init_design_uniquify**: If set to true, will uniquify the design during import.
- **init_delete_assigns**: Ensures that **assign** statements will not be created in the exported netlist.

3.7 Initializing the Design

After reading the MMMC file, LEFs, netlist and defining the global nets, we are ready to import the design with the **init_design** command:

```
init_design
```

Pressing **OK** on the **DESIGN IMPORT** form will run the **init_design** command.

3.8 Checking Design Import Results

3.8.1 Checking Warnings and Errors in the Log File

The **init_design** command initiates many operations and procedures, which run automatically. These include:

- Loading the LEF files
- Reading the Verilog netlist
- Loading the MMMC or equivalent CPF definitions, including
 - Loading the LIB files
 - Loading the extraction rules
 - Loading the design SDC

There are many potential mistakes that could have been made along the way, such as a syntax error in one of the definition scripts or the SDC, or perhaps a change in the path of a file. Therefore, it is *essential* to

carefully go over the logfile after design import finishes and rerun it after fixing any errors and important warnings, until the results are satisfactory.

3.8.2 The `check_legacy_design` command

Finally, run the `check_legacy_design` command:

```
check_legacy_design -all
```

The `check_legacy_design` command will create an HTML report that you can browse with Firefox:

```
%> firefox ../reports/init_design/my_design.main.htm
```

3.9 Global Net Connection

During design import, names were defined for the power and ground nets to be used in the design. These are also created in a power intent (CPF/UPF) flow. However, the virtual connection between a global net and the pins of a macro has to be defined separately. For example, if `init_power_nets` was set to `VDD` during design import, but the power pin of the standard cell library is called `vcc`, then the tool has to be told to connect the two. This is done with the `connect_global_net` command or the **POWER→CONNECT GLOBAL NETS** form, shown in Figure 2, (and can also be done within the CPF/UPF flows).

A sample command for connecting the `VDD` net to the `vcc` pins of a standard cell library would be:

```
connect_global_net VDD -type pgpin -pin_base_name vcc -instance_base_name *
```

Note that the GUI for this command is quite unintuitive. For example, to apply the above command with the GUI, you will need to follow these steps:

1. Choose **PIN**
2. **INSTANCE BASENAME**: *
3. **PIN NAME(S)**: `vcc`
4. **TO GLOBAL NET**: `VDD`
5. Now click **ADD TO LIST**
6. Now click **APPLY**.

For some technologies, *tie cells* are used to connect constants ('0', '1') to gate inputs. A global net connection command for such a case could be:

```
connect_global_net VDD -type tiehi
```

After running each `connect_global_net` command, go over the summary to see that the number of connections made makes sense. Alternatively, run with `-verbose` option to print out all connections made.

```
1750 new gnd-pin connections were made to global net 'VSS'.
1750 new pwr-pin connections were made to global net 'VDD'.
```

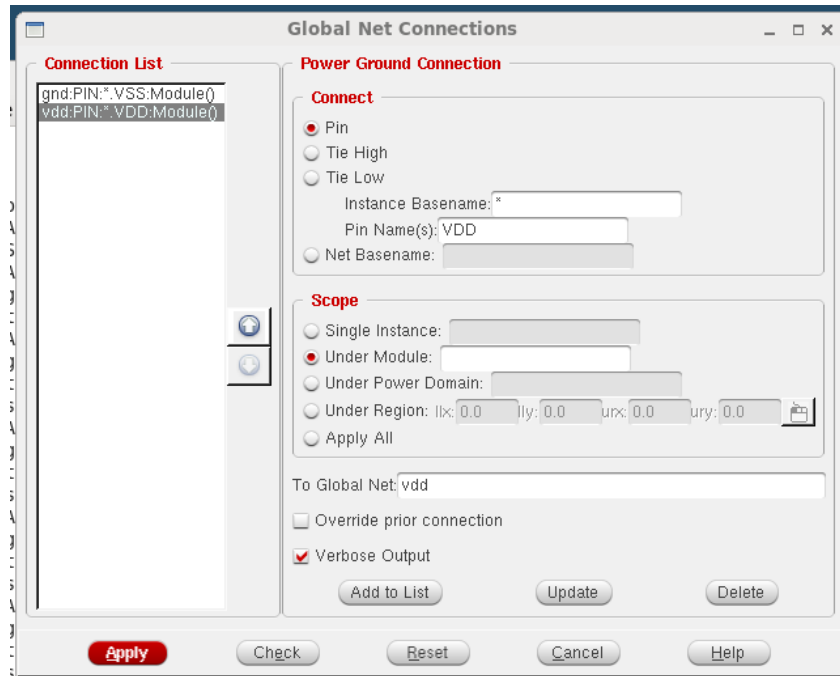


Figure 2: The Global Net Connections form

3.10 Restoring a saved design

In case you have saved a design using the `write_db` command, you can restore the design by using the `read_db` command:

```
read_db ../export/my_saved_design/
```

Note that restoring the design will probably not restore all the design variables that you set during the process, so you should read your design definitions, procedures, etc. again.

4 Multi-Mode Multi-Corner Settings

Multi-Mode Multi-Corner (MMMC) is a framework for defining timing conditions for a design, taking into account the various corners and modes in the design spec. Both timing reports and timing optimization can take into account the many corners and modes according to the MMMC definitions.

4.1 MMMC Command Hierarchy

The MMMC framework is defined according to a quite complex hierarchy, as illustrated in Figure 3.

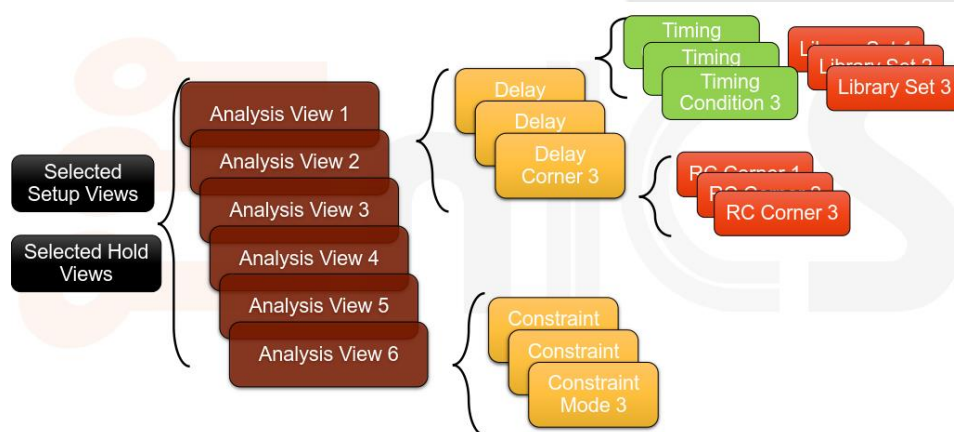


Figure 3: MMMC Hierarchy

From top to bottom, the hierarchy consists of:

- **Selected Setup/Hold Views:** These are the **Analysis Views** that will be considered when running setup/hold timing optimization and reports.
- **Analysis Views:** These are the top level views that contain all the corner and operating mode conditions and constraints. **Analysis Views** consist of **Constraint Modes** and **Delay Corners**.
- **Constraint Modes:** These are the settings that define the operating modes of the design.
- **Delay Corner:** These are the settings that define the operating conditions of the design. **Delay Corners** consist of **RC Corners** and **Timing Conditions**.
- **RC Corners:** These are the settings that define the rules for parasitic extraction.
- **Timing Conditions:** These are the setting that define how the timing for each partition of the design should be taken into account. **Timing Conditions** are defined with **Library Sets**.
- **Library Sets:** These are groups of libraries (.lib files) that correspond with a certain operating condition (i.e., corner)

4.2 Defining and Selecting Analysis Views

Analysis Views are the top level entities in MMMC, which define what the mode of the design is and the corner for extraction and timing. Each pair of mode and corner is defined in a separate **Analysis View**, and then any number of **Analysis Views** is selected for setup checking (a.k.a. Max Delay) and hold checking (a.k.a., Min Delay). An **Analysis View** consists of a **Constraint Mode** and a **Delay Corner** and is defined with the `create_analysis_view` command:

```
create_analysis_view -name turbo \
    -constraint_mode turbo_mode -delay_corner slow_corner_vdd12
```



```
create_analysis_view -name low_power \
    -constraint_mode low_power_mode -delay_corner slow_corner_vdd05
create_analysis_view -name turbo_hold \
    -constraint_mode turbo_mode -delay_corner fast_corner_vdd13
```

- A **Delay Corner** tells the tool how the delays are supposed to be calculated. Therefore, it contains timing libraries and extraction rules.
- A **Constraint Mode** is basically the relevant SDC commands/conditions for the particular operating mode.

After creating the **Analysis Views**, one or more views can (separately) be selected for setup timing and for hold timing. These selections can be changed throughout the flow by running the [set_analysis_view](#) command:

```
set_analysis_view -setup {turbo low_power} -hold {turbo_hold}
```

4.3 Constraint Modes

A **Constraint Mode** is simply a list of relevant SDC files. When you move between analysis views, the STA tool will automatically apply the relevant constraints to the design.

```
create_constraint_mode -name turbo_mode -sdc_files {turbo.sdc}
create_constraint_mode -name low_power_mode -sdc_files {low_power.sdc}
```

4.4 Delay Corners

A **Delay Corner** is a bit more complex. It comprises a **Timing Condition**, an **RC Corner** and a few other things that we won't discuss right now.

```
create_delay_corner -name slow_corner_vdd12 \
    -rc_corner {RCmax} -timing_condition {ss_1p2V_125C}
create_delay_corner -name slow_corner_vdd05 \
    -rc_corner {RCmax} -timing_condition {ss_0p5V_125C}
create_delay_corner -name fast_corner_vdd13 \
    -rc_corner {RCmin} -timing_condition {ff_1p3V_m40C}
```

4.5 Timing Conditions

A Timing Condition is a collection of library sets to be used for a certain power domain. For now, we will just automatically connect a **Timing Condition** to a **Library Set**.

```
create_timing_condition -name tc_ss_1p2V_125C \
    -library_sets ss_1p2V_125C
```

4.6 Library Sets

A **Library Set** is a collection of the .lib characterizations that should be used for timing the relevant gates. This includes the standard cells and other macros, such as RAMs and I/Os. There also may be special "SI" characterizations for noise.

```
create_library_set -name ss_1p2V_125C \
    -timing [list ${sc_libs}/ss_1p2V_125.lib ${mem_libs}/ss_1p2V_125.lib \
        ${io_libs}/ss_1p8V_125.lib] -si ${sc_libs}/ss_1p2V_125.si
```

4.7 RC Corners

An **RC Corner** is a collection of the rules for RC extraction. There may be a **capacitance table** for quick extraction and a **QRC techfile** for accurate extraction. The temperature is also defined in the **RC Corner**, but it is taken into account in the .lib file, as well.

```
create_rc_corner -name RCmax -cap_table ${tech}/RCmax.CapTbl} -T {125} \  
-qx_tech_file ${tech}/RCmax.qrctech
```

4.8 Reviewing your MMMC Configuration

To review your MMMC configuration, use the [report_analysis_views](#) command:

```
report_analysis_views
```

5 Floorplanning

After design import, the tool will be in “Floorplan View”, which will show you (see Figure 4):

- *Center*: a default floorplan with the locations of the I/Os (if an I/O file was provided)
- *Left*: A hierarchical view of the Verilog modules, sized according to estimated standard cell area.
- *Right*: A box for each hard macro (such as SRAMs) according to their LEF defined shape.

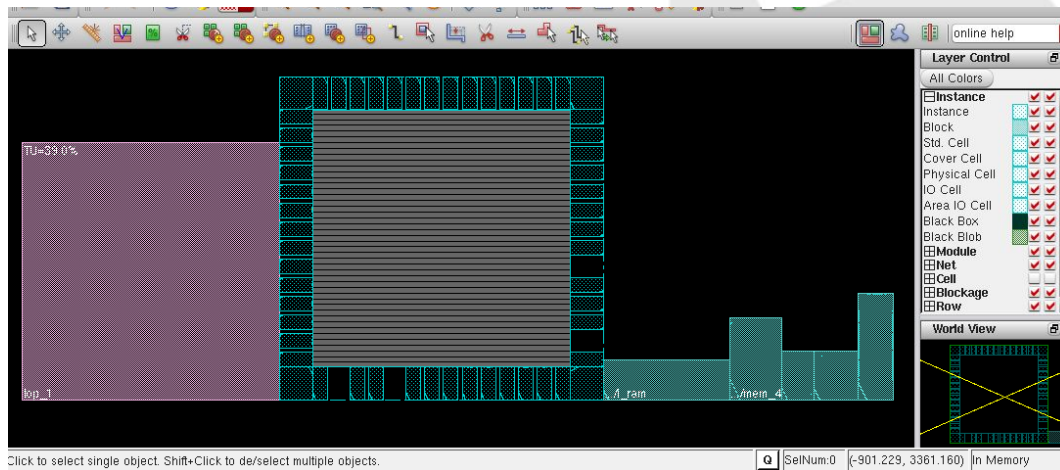


Figure 4: Floorplan View after Import Design

5.1 Specify Floorplan

The floorplan box at this stage is set either according to the I/O configuration file or based on a default utilization (70%) with a 1:1 ratio. However, you should setup the floorplan according to your specific intent. This is done with the `create_floorplan` command or the **FLOORPLAN→SPECIFY FLOORPLAN** form, shown in Figure 5.

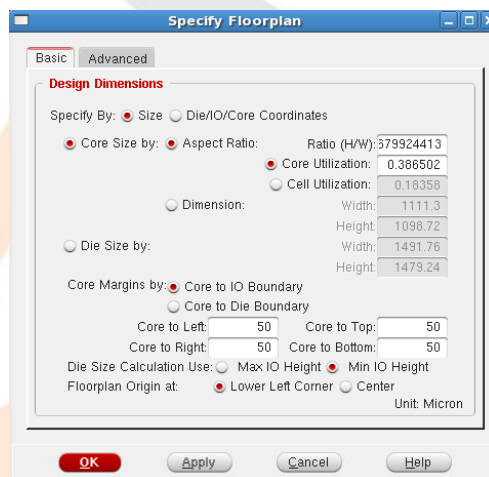


Figure 5: Specify Floorplan Form

You can specify the floorplan according to various parameters, such as:

- Coordinates
- Width and Height

- Aspect ratio and Initial Utilization (based on gate size from the netlist)

In addition, you can specify the distance between the borders of the design and the core area (where the standard cell rows are created). If you have I/Os then this distance will usually be from the inner boundary of the I/O ring to the core area. This space is usually used to make rings for the power and ground signals and leave room from routing to the I/Os.

An example command for specifying a floorplan with a square ratio, a target utilization of 70% and 5 microns between the boundary and the standard cell rows (with the standard cell SITE called "CORE") is:

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

For additional details see the user manual or run `create_floorplan -help`.

5.2 Reading in a previously saved Floorplan

If you have previously saved a floorplan with the `write_floorplan` command, you can read it in either with the **FILE→LOAD→FLOORPLAN** form or with the `read_floorplan` command:

```
read_floorplan top.fp
```

5.3 Define Pin Placement

If you are working on a block, rather than the full chip, you will not define an I/O file during design import, and therefore, the ports to your design will be simple pins, i.e., pieces of metal around the boundary of the floorplan. After initializing your design, the pins all be on top of each other at coordinate (0,0) in the bottom left corner, as shown in Figure 6.

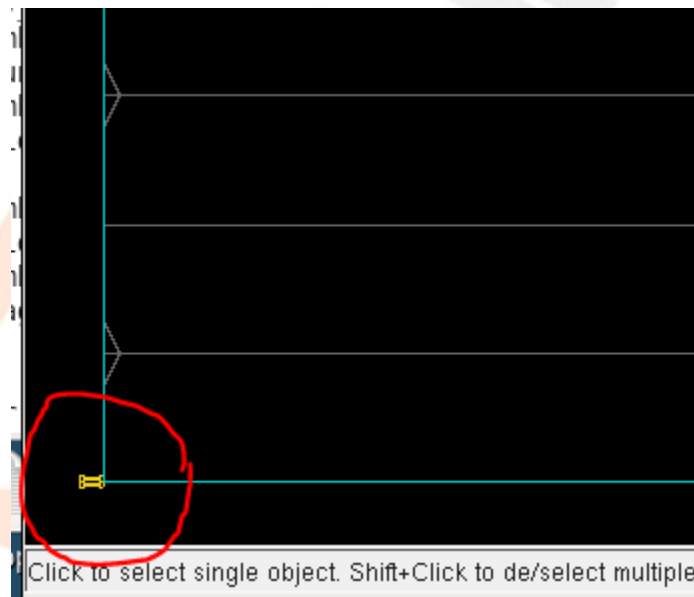


Figure 6: Default pin location after import design

This is obviously bad. It is also not recommended to let the placement algorithm move the pins to wherever it feels best, since the pins have a significant influence on the placement and the location of the pins is important for connection outside the block. There are several methods for providing a location for pins, including:

- **Dragging and dropping:** Just click on a pin and click on the **MOVE/RESIZE/RESHAPE** button (Shift+R), then click a pin, and drag it wherever you want it.
- **Command Line:** Change the location, size, metal layer, etc. of a pin or of a group of pins, using the **edit_pin** command. For example, to move the **clk** pin to the top side of the macro on M3 at the coordinate (0.0, 3.9):

```
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
```

Or, for example, to spread a list of pins along the entire bottom edge of the macro in M2:

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

- **Pin Editor Form:** A big form for editing pins is the pin editor form, shown in Figure 7, accessible from the GUI at **EDIT→PIN EDITOR**.

Figure 7: Pin Editor Form

- **Pin Groups:** You can define pin groups through the form found at **EDIT→EDIT PIN GROUP**.
- **Pin Guides:** You can define pin guides through the form found at **EDIT→EDIT PIN GUIDE**.

5.4 Place Macro Cells

There are three basic ways to place macro cells:

1. **Drag and Drop:** In the Floorplan View, drag the boxes on the right and manually set them inside the floorplan area.
2. **Specify coordinates for the macro:** Use the `set_obj_floorplan_box` command. An example placement would be:

```
set_obj_floorplan_box Instance SRAM1 {10 20 30 40}
```

3. **Use relative floorplanning:** Use the **FLOORPLAN→RELATIVE FLOORPLAN→EDIT CONSTRAINT** form to define a relative placement for an instance, such as x and y from the design corner or from another instance.

5.5 Add Rings around Macros

It is usually a good idea to add power/ground rings around macros in order to enable easy connection to the power stripes and connection to the power/ground pins of the macros themselves. As usual, this can be done through the command line, using the `add_rings` command or through the GUI with the

POWER→POWER PLANNING→ADD RING form, shown in Figure 8.

When defining rings, you will need to set the nets (vdd, gnd, etc.), the layer for the horizontal (top and bottom) and vertical (left and right) edges, the widths of the rings, the spacing between them, etc. An example command for adding VDD and GND rings in M1 and M2 around a selected macro would be:

```
select_inst ${TOPLEVEL}/SRAM1

add_ring -around selected -nets {VDD GND} -type block_rings \
        -layer {bottom metal1 top metal1 right metal2 left metal2} \
        -width 4 -spacing {bottom 0.24 top 0.24 right 0.28 left 0.28} \
        -offset 0.31

deselect_all
```

Figure 8: The Add Rings Form

5.6 Add Halos around Macros

Good practice is to add placement blockage around macros, so the placer won't put any standard cells in the immediate area around the macro, potentially blocking routing to the pins or causing other congestion.

Note that some macros, especially SRAMs, are designed to efficiently connect to the power/ground stripes in high layers without needing pins in order to save area.

A special command, `create_place_halo` can add this blockage, or it can be achieved with the **FLOORPLAN→EDIT FLOORPLAN→EDIT HALOS** form, shown in Figure 9. An example command to add a 14 micron halo around a memory instance would be:

```
create_place_halo -halo_deltas 14 14 14 14 ${TOPLEVEL}/SRAM1
```

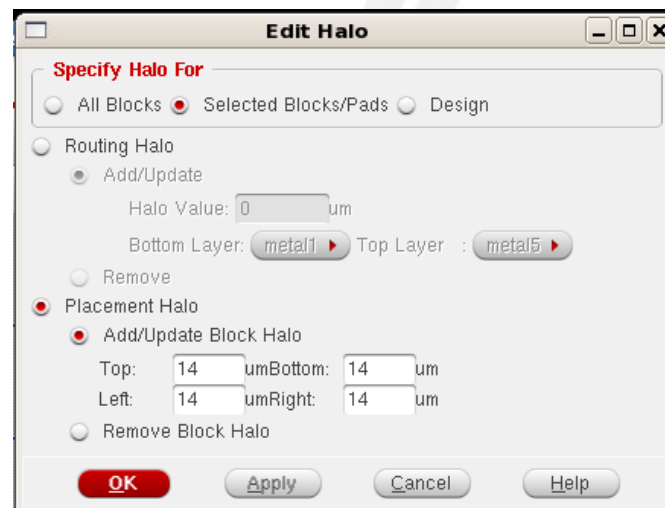


Figure 9: Edit Halo Form

5.7 Create FullChip Rings

Good practice is to create nice wide power and ground rings around the chip (between the I/Os and the core) to distribute the power evenly from the power/ground pads and to make connecting to the power/ground nets easier for power stripes and follow pins. This is done with the same `add_rings` command or the **POWER→POWER PLANNING→ADD RING** form, used in Section 5.5, only this time select the **AROUND CORE BOUNDARY** option or the `-type core_rings` option for command line. An example command to add I/O Rings would be:

```
addRing -type core_rings -nets {VDD GND} \
  -layer {bottom metal1 top metal1 right metal2 left metal2} \
  -width 20 -spacing 1 -offset 0.31
```

An example of what it looks like after successfully adding a set of power/ground rings around the Core Boundary is shown in Figure 10.

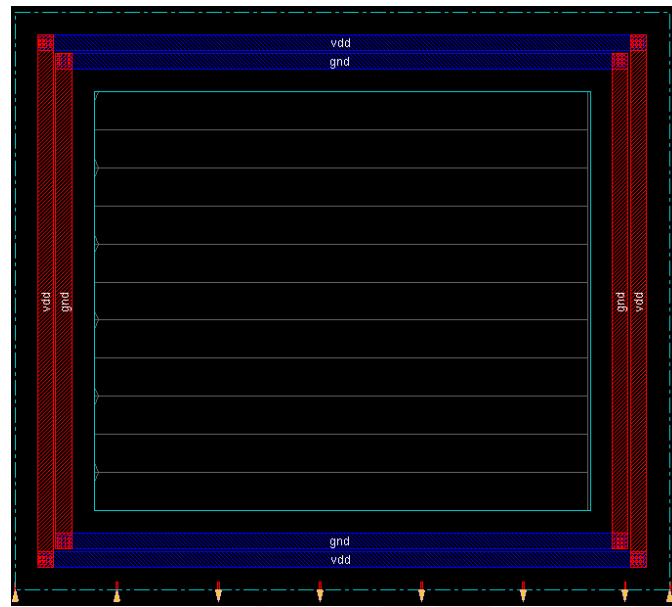


Figure 10: Example of Ring around Core Boundary

5.8 Create Placement Regions/Guides

Placement guides enable several levels of constraining instances to a certain area, according to the following hierarchy:

- **Soft Guide:** The tool will try to place the instances close to each other without an actual location constraint.
- **Guide:** The tool will try to place the instances within the defined area.
- **Region:** The tool *has to* place the instances within the defined area, but other instances can also be placed inside.
- **Fence:** The tool *has to* place the instances within the defined area, and not other instances are allowed to be placed inside.
- **Density:** Set a different utilization target for a specified area than the global utilization target.

The way to define these guides is a bit awkward: first create a “group” with a constraint (such as region or density) and then *add instances* to the group. This is done with the following commands:

```
create_inst_group <name> [-guide|-region|-fence-|soft_guide <coords>|-density <val>]
update_inst_group -name <groupName> -add (<hInstName | instName | groupName>)
```

5.9 Create Placement Blockages

You can create full or partial placement blockages with the `create_place_blockage` command. The types of placement blockage are:

- **hard:** The area cannot be used to place blocks or cells. This is the default.
- **soft:** The area cannot be used to place blocks during standard cell placement, but can be used during in-place optimization, clock tree synthesis, or ECO Placement.
- **partial:** Sets a percentage of the area that is available for placement.

A few examples:

- To create placement blockages around every macro:

```
create_place_blockage -all_macros
```

- To create a partial blockage (limited utilization) in a specific area:

```
create_place_blockage -area "$llx $lly $urx $ury" -density 50 -type partial
```

5.10 Pre-place Physical Cells

5.10.1 Add Well Taps

In most modern technologies, the standard cells do not include well taps (body connections) inside the cell itself. The DRC of the technology states the minimum distance between taps to protect from latchup, and usually this is around 20 microns. Therefore, the library vendor provides special well tap cells that should be spread in every other row (only one tap needed per continuous well) with a spacing of less than the DRC requirement. An example of *staggered* well taps is shown in Figure 11.

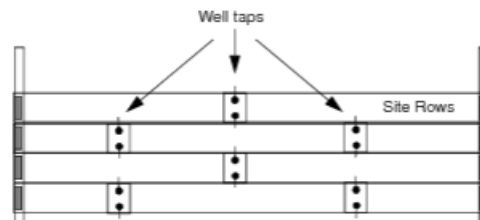


Figure 11: Well Taps placed in a "staggered" fashion

This is done with the `add_well_taps` command or with the **PLACE → PHYSICAL CELL → ADD WELL TAPS** form from the GUI, shown in Figure 12. For example, to add well taps every other row, every 10 micron, starting at 3 microns from the left, the command would be:

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

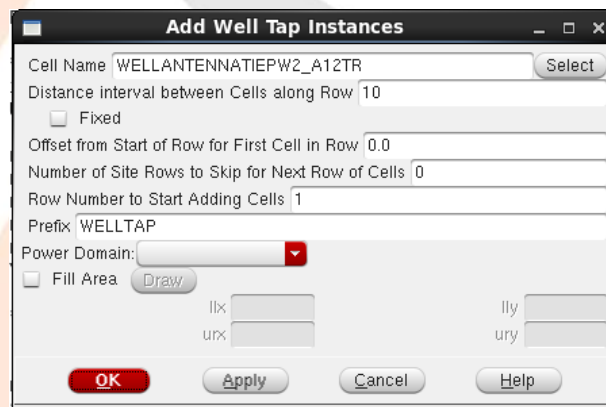


Figure 12: Add Well Taps form

To check that there is a sufficient number of well taps in the design, use the `check_well_taps` command, where the `-max_distance` option sets the allowed distance between taps:

```
check_well_taps -max_distance 20

Finished check_well_taps, 0 violations found.
```

5.10.2 End Caps

In some technologies, End Cap cells are required. The definition of these can vary, but they are usually cells that “finish” the row well in a “clean” way, especially if deep N-Wells are used. There can be end cap cells meant for all sides of the core area, but usually left and right end caps are sufficient. Figure 13 shows an example of end caps placed at the ends of each standard cell row.

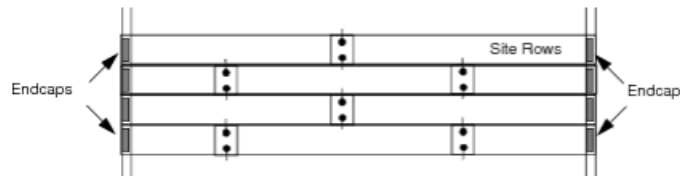


Figure 13: End Caps placed at the ends of each site row.

To add end caps, use the `add_endcaps` command or from the GUI **PLACE→PHYSICAL CELL→ADD END CAPS**. To define the specific cells to use on each edge, use the end cap database attributes, for example:

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

5.11 Connecting Power Nets with Special Route

This is a horrible name for a set of operations that all use the same command, `route_special`, and with the same form **ROUTE→SPECIAL ROUTE**, shown in Figure 15. Depending on the values added with the `-connect` option, various nets can be routed, as follows:

- `route_special -connect block_pin` **BLOCK PINS**: Connects the power/ground pins of macros to either rings around the macros or to power stripes or pins of I/Os.
- `route_special -connect pad_pin` **PAD PINS**: Connects the power/ground pins of I/O to either rings around the chip or to power stripes or to pins of macros.
- `route_special -connect pad_ring` **PAD RINGS**: Connects the internal power ground rings that are inside the I/Os. If you have a complete pad ring, this should be already achieved by abutment, but this at least tells the tool they are virtually connected.
- `route_special -connect core_pin` **FOLLOW PINS**: Connects the power/ground rails of the standard cell rows, also known as “follow pins”. These, also, should be connected through abutment (after adding fillers) but this virtually connects them and also connects them to the vertical rings around the boundary. An example of a floorplan with follow pins connecting to the rings is shown in Figure 14.
- `route_special -connect floating_stripe` **FLOATING STRIPES**:
Note that the order of running the `route_special` command can be important. For example, if you connect follow pins before creating an I/O ring, the follow pins may not find a target to connect to.
- `route_special -connect secondary_power_pin` **SECONDARY POWER PINS**: Connects secondary power pins, which could be:
 - Body Bias connections.
 - Second power connection of Level shifters.

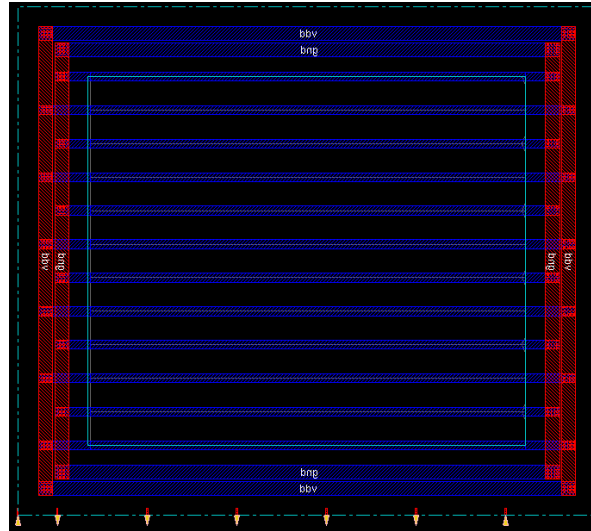


Figure 14: Example of floorplan with rings after adding Follow Pins

The `route_special` command has many many options that can be applied as flags to either the `route_special` command, itself, or to the `route_special` database attributes. These include options that tell the tool what nets to route, what layers to use, and many more. Good luck!

The setting for the `route_special` database attributes can be defined through the GUI by pressing the **MODE SETUP** button on the Special Route form.

Here is an example `route_special` command, but do this first with the GUI and copy the command that was run after you get it to work.

```
route_special -connect { block_pin pad_pin } -nets {VDD GND} \
  -layer_change_range { metal1 metal5 } -allow_jogging 1 \
  -block_pin_target { ring stripe nearest_target } \
  -pad_pin_port_connect { all_port one_geom } -allow_layer_change 1 \
  -crossover_via_layer_range {metal1 metal5} \
  -target_via_layer_range {metal1 metal5} -block_pin use_lef
```

Figure 15: The Special Route Form

5.12 Adding Power Stripes

Now that we have a ring around the chip and horizontal follow pins to create/connect the standard cell rows, it is important to provide low resistance paths to every point in the floorplan. This is usually done by creating a power grid, or in other words, routing vertical and horizontal stripes in different metal layers to carry current to the standard cells and other power consuming instances.

Stripes are created with the `add_stripes` command or with the **POWER→POWER PLANNING→ADD STRIPES** form, shown in Figure 16. There are many options for creating stripes, starting with the nets to route, the layer to use, the width and spacing between the stripes, the distance between sets of stripes, and the target to connect to, such as the core ring or the internal pad ring. An example command for creating 5 micron wide VDD and GND stripes in M2 every 100 microns could be:

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

Figure 16: Add Stripes Form

An example of stripes added in M2 with successful connections to both rings and follow pins is shown in Figure 17.

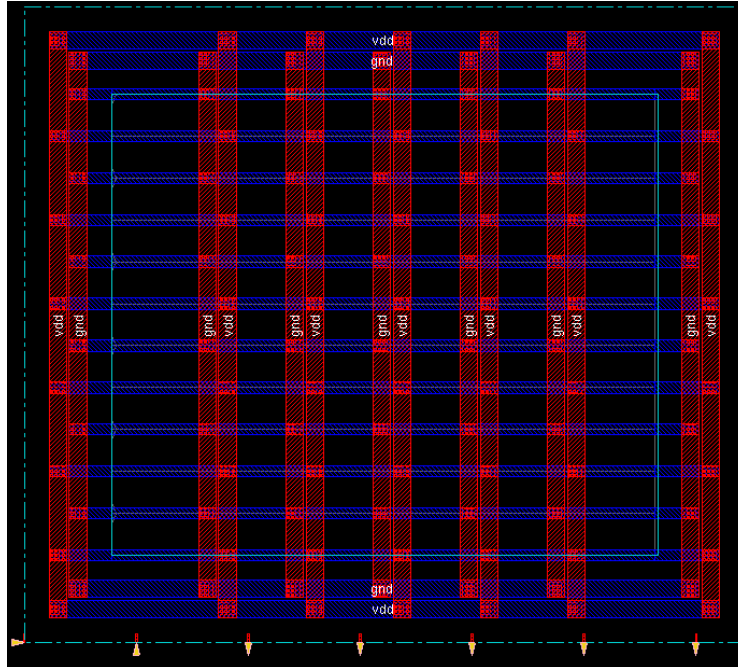


Figure 17: Example of floorplan with rings and follow pins after adding Stripes in M2

5.13 Additional Tweaks

5.13.1 Trimming excess power stripes

You may trim redundant PG stripes and vias using an existing power grid pattern based on IR drop analysis reports with the `edit_trim_routes` command. This eliminates the need to manually adjust the power structure to reduce power grid pessimism for area/timing/power improvement in a design.

5.13.2 Eliminating congestion in routing channels

You can automatically adjust the channel width between objects to prevent potential routing congestion, using the `resize_floorplan_channel` command.

An additional setting that can be useful if the design has routing congestion in channels between blocks is to automatically add soft placement blockages in these areas:

```
set_db place_global_auto_blockage_in_channel {none soft partial}
```

5.13.3 Adding Padding

Sometimes, in order to reserve space during placement, padding can be added to cells or modules. The space saved by padding can be used for timing optimization (e.g., hold buffers), clock tree synthesis, adding decaps next to clock buffers, etc.

To define padding for a specific type of leaf cell, use the `set_cell_padding` command. To define the padding for a particular instance, use the `set_inst_padding` command. Padding factor is given in number of sites.

5.14 Verifying the Floorplan

After creating the floorplan and finishing the power routing, it is essential to run DRC and LVS verification. This should probably be run with an external DRC tool (such as Assura or Calibre DRC), but at the very least, you should run Innovus's internal DRC and LVS checks.

5.14.1 Running DRC with `check_drc`

Innovus's internal tool for running DRC is called `check_drc`. You can also invoke the tool from the GUI at

CHECK→CHECK DRC.

```
@[DEV]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) ***
```

5.14.2 Running LVS with `check_connectivity`

Innovus's internal tool for running LVS is called `check_connectivity`, and it can be invoked from the GUI at **CHECK→CHECK CONNECTIVITY**. This is one of the tougher tools to get to work correctly, since as opposed to standard LVS, which compares two spice netlists (the schematic and the GDS), this tool is comparing the virtual connectivity of the database with the LEF abstracts of the macros and the routing data. Therefore, you can expect many errors that can be a result of factors, such as incomplete global net definitions, opens due to lack of fillers, strange LEF definitions of macros, etc.

To try and get a clean LVS at this stage:

- Add filler cells prior to running `check_connectivity` and then erase them after.
- Run `check_connectivity` with the `-type special` option to check only special routes

```
@[DEV]innovus 79> 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)
```

5.14.3 Checking the Floorplan

An overall useful tool to run is `check_floorplan`, which checks the quality of the floorplan to detect potential problems before the design is passed on to other tools.

```
@[DEV]innovus 80> check_floorplan
Checking routing tracks.....
Checking other grids.....
Checking FINFET Grid is on Manufacture Grid.....

Checking core/die box is on Grid.....

Checking snap rule .....

Checking Row is on grid.....

Checking AreaIO row.....
Checking routing blockage.....
Checking components.....
Checking constraints (guide/region/fence).....
Checking groups.....

Checking Preroutes.....
No. of regular pre-routes not on tracks : 0
```

5.14.4 Checking the Design

Run the `check_design` command with the `-type place` option to check for macros that are not fixed prior to placement. You must resolve the issue by updating the `place_status` to `fixed` before running placement.

```
check_design -type all
```

6 Placement

Following the creation of a floorplan, we can now move on to standard cell placement

6.1 Setup Placement Options

As you have already seen, or will see from now on, almost every “big” command in Innovus has a set of corresponding database attributes for setting up mode options (using `set_db`). The category for placement database attributes is called `place`. Therefore, you can see a list of the current values of all relevant attributes for placement using `get_db -category place *`:

```
@[DEV]innovus 89> get_db -category place *
Object: root:/
  place_cell_edge_spacing: {}
  place_design_floorplan_mode: false
  place_design_refine_place: true
  place_detail_allow_border_pin_abut: false
```

You can also access these attributes through the GUI either directly by **TOOLS→GLOBAL** and choosing **CATEGORY: PLACE** or by clicking on the **GLOBAL** button on the **PLACE→PLACE STANDARD CELL** form. The form is shown in Figure 18.

Name	Category	Value	Description
place_global_enable_distributed_place	place	FALSE	enable distributed placement
place_global_group_flop_to_macro	place	FALSE	group flops to Macro
place_global_group_flop_to_macro_level	place	1	groupFlopToMacroLevel value
place_global_group_flop_to_macro_list	place		List of macros for which nets con
place_global_ignore_scan	place	true	ignore scan net during placement
place_global_ignore_spare	place	FALSE	discard spare cell connections du
place_global_max_density	place	-1.0	placement strives to not let densi
place_global_module_aware_spare	place	FALSE	Spare insts are placed randomly a
place_global_module_padding	place		define the padding factor for the
place_global_place_io_pins	place	FALSE	place IO Pins concurr
place_global_reorder_scan	place	TRUE	turn on reorder scan during place
place_global_soft_guide_strength	place	low	level of effort for user defined se
place_global_timing_effort	place	medium	level of effort for timing driven glo
place_global_uniform_density	place	false	enable even cell distribution for d
place_hard_fence	place	TRUE	honor fence and region constrain
place_opt_post_place_tcl	place		a tcl script to be sourced after init
place_opt_run_global_place	place	full	place opt run global place
place_spare_update_timing_graph	place	TRUE	update timing graph during place

Figure 18: Placement Mode Options Form

As with all mode setting commands, there are many options that can do anything but make you coffee, so you will have to open the manual or [Cadence Support](#) to try and figure out what all of them are. But here are a few important ones:

- Be aware of clock gates

```
set_db place_global_clock_gate_aware true
```

- Try to eliminate congestion problems around macros

```
set_db place_global_auto_blockage_in_channel soft
```

- Add padding (extra space) around cells in a certain module

```
set_db <module> .place_global_module_padding <factor>
```

6.2 Run Placement

The super command for running placement in Innovus is **place_opt_design**. This command actually runs concurrent placement and optimization:

```
place_opt_design
```

As an alternative, you can run placement without optimization using the **place_design** command, and later run optimization using **opt_design -pre_cts**. This can also be accessed from the GUI through

PLACE→PLACE STANDARD CELL. The **PLACE** menu is shown in Figure 19.

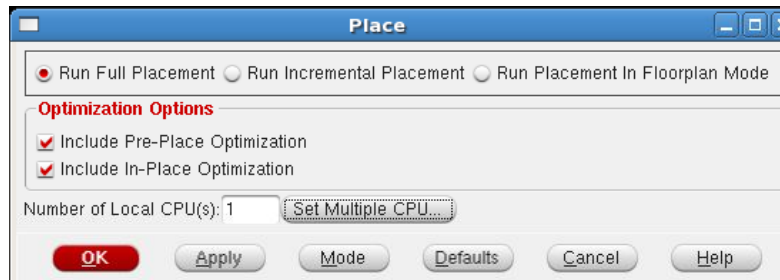


Figure 19: Place Menu

6.2.1 Useful placement attributes

- To reduce congestion:

```
set_db place_global_cong_effort high
```

- If the design has local congestion, rerun placement with:

```
set_db place_global_module_padding module factor
```

- To reduce to switching power on power-critical nets:

```
set_db place_global_clock_power_driven true
```

6.3 Manual cell placement/movement

If for some reason, you need to make manual changes in the placement, you can do so either through the GUI by selecting the move icon, or with the `place_inst` command. However, unless you carefully calculated the move to fit on the placement grid and not overlap with another cell, you will need to legalize the placement. This is achieved with the `place_detail` command:

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

6.4 Tie Cell Placement

Tie cells are gates that do nothing other than connect to VDD and GND to provide a '1' or '0' signal that is tied to some inputs. They are not required in all processes, but some libraries require the use of them. Since these are constant signals that do not drive any current, there is essentially no timing constraint on them, but driving a high fanout with a single cell is not a good idea. Therefore, it is recommended to erase any automatically placed tie cells, define constraints for them (maximum fanout, maximum distance to the gate, etc.) and replace them. This is achieved with the following commands (assuming the tie cells in the library are called "TIE1" and "TIE0"):

```
delete_tieoffs -cell {TIE1 TIE0}
set_db add_tieoffs_cekks "TIE0 TIE1"
set_db add_tieoffs_max_fanout 20
set_db add_tieoffs_max_distance 250
set_db add_tieoffs_prefix tieoff
add_tieoffs
place_detail
```

Alternatively, tie cells can be placed through the **PLACE→TIE HI/LO→ADD** form.

6.5 Post Placement Checks/Reports

6.5.1 Check Placement

The **check_place** command, also accessible from **PLACE→CHECK PLACEMENT...**, checks the placement with all kinds of preset checks:

```
check_place
Begin checking placement ... (start mem=1151.5M, init mem=1151.5M)
*info: Placed = 40          (Fixed = 2)
*info: Unplaced = 0
Placement Density:48.89%(201/411)
Placement Density (including fixed std cells):49.77%(208/419)
Finished checkPlace (cpu: total=0:00:00.0, vio checks=0:00:00.0; mem=1151.5M)
```

6.5.2 Check Density

The design density can be reported in several ways:

- The **report_place_density** command **PLACE→QUERY DENSITY→QUERY PLACE DENSITY**

```
Average module density = 0.853.
Density for the design = 0.853.
    = stdcell_area 842 sites (404 um^2) / alloc_area 987 sites (474 um^2).
Pin Density = 0.246.
    = total # of pins 227 / total Instance area 922.
```

- The **report_density_map** command will return a (text) histogram of the density of your design:

```
default core: bins with density > 0.75 = 25 % ( 1 / 4 )
bin size: 120 sites by 10 row(s). total 4 bins ( 2 by 2 )
density range    #bins    %
  0.9 - 0.95      1       25
total            1       25
```

- The **PLACE→DISPLAY→DENSITY MAP** menu will show you the density map with a color scheme on the floorplan itself.
- Pressing the percentage icon and then selecting an area on the screen will return the placement density in the selected area:



```
StdInstArea/freeSpace = 87.3%(442.56/506.88)
macroInstArea/totArea = 0.0%(0.00/823.68)
powerMetalArea/totArea = 38.5%(316.80/823.68)
PlacementObsArea/totArea = 0.0%(0.00/823.68)
Utilization in area (-1672 -1475) (84071 74398) = 87.31%
```

6.5.3 Check Timing

To check timing:

```
report_timing
```

Or run the **time_design** super command, which will dump a whole set of timing reports:

```
timeDesign -pre_cts -ideal_clock
```


6.6 Placement Optimization

After running placement, or actually any stage of the design, you can run an optimization iteration. If you used `place_opt_design`, this was already run along with the placement, but sometimes you may change a constraint or want the tool to work even harder on optimization at this stage. The command for running optimization is `opt_design` or it can be accessed through the GUI at **ECO→OPTIMIZE DESIGN**, as shown in Figure 20. In the case of post placement optimization, the command is:

```
opt_design -pre_cts ; # "pre_cts" because Clock Tree Synth is next!
```



Figure 20: Optimization Form

This can also be achieved by running `place_opt_design` with the `-incremental` option:

```
place_opt_design -incremental
```

It is not clear what the difference between the two optimization commands is, but the user guide recommends the second option (`place_opt_design -incremental`)

To check timing after placement optimization, again, you can run the `time_design` super command:

```
time_design -pre_cts -ideal_clock -path_report -drv_report \
-slack_report -num_paths 50 -prefix pre_cts -report_dir reports/pre_cts
```

6.6.1 Repairing Congestion

If you see congestion problems at the post placement or post-CTS stages, try the `place_fix_congestion` command:

```
place_fix_congestion
```

7 Clock Tree Synthesis

Clock tree synthesis is an important and quite complex stage in the physical implementation of a chip. Cadence provides two engines for implementing the clock tree:

- **Skew Balanced Clock Tree Synthesis (Traditional CTS):** This is the traditional clock tree implementation, targeting a zero skew clock tree that attempts to maintain the ideal clock timing features of the design (setup and hold) after adding a real propagated clock tree. Skew balanced CTS is run with the `clock_design` command.
- **Clock Concurrent Optimization (CCOpt):** This is an alternative approach to clock tree synthesis, which targets meeting timing constraints (setup and hold), while achieving improved performance, power and area, by iterating between clock synthesis and timing optimization, as well as applying useful skew. Clock concurrent optimization is run with the `ccopt_design` command.

An in depth description of the clock tree synthesis procedure and various features is provided in the document [08-Clock Tree Synthesis](#). This section will briefly introduce the main commands required for the flow, without going into very many deviations from the standard.

7.1 Create Automatically Generated CCOpt Configuration

To automatically generate a clock tree specification file for CCOpt, use the `create_clock_tree_spec` command. I recommend doing this the first time or after any significant change in your SDC, but then manually edit your own configuration file (`../inputs/${TOPLEVEL}.ccopt`), which includes the settings that you really want.

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

7.2 Define Clock Trees and Skew Groups

The basic (most simple) clock tree configuration will have a *clock tree* for each clock defined in the SDC and a *skew group* of the same name that includes all of the *clock tree*'s sinks. These are created, as follows:

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

The rest of the commands below require more knowledge about clock tree synthesis, and I will not explain them in this document. But I have included the most relevant commands for reference.

7.3 Define Special Pins

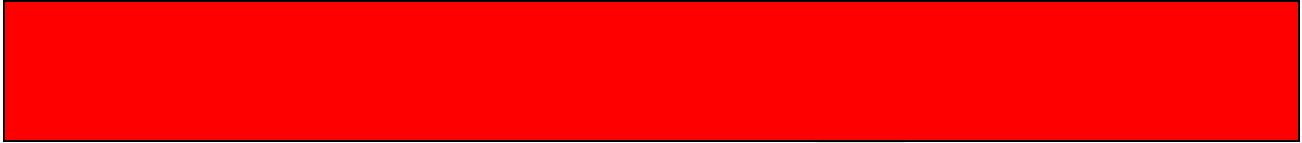
7.3.1 Stop Pin

A *Stop Pin* is a sink that should be considered part of a *skew group*. The `-auto_sinks` option of `create_skew_group` will traverse the clock tree and automatically mark all sinks (clock pins) as *Stop Pins*. However, if for some reason, you would like to mark an additional pin in your design as a *Stop Pin*:

```
set_db pin:$pin .cts_sink_type stop
```

In addition, you can add sinks to an existing *skew group* with the `update_skew_group` command:

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



7.3.2 Ignore Pin

An *ignore pin* is a pin that should not be considered for skew balancing/analysis, but you would still like to build your clock tree up to this pin (i.e., fix DRV violations up to the pin). An example of an ignore pin is the clock input to a clock divider.

Note that I am careful not to write that a Stop Pin is considered for skew balancing. This is because CDCm doesn't necessarily attempt to balance skew, just to make sure there are no setup and hold violations. In CTS Mode, all Stop Pins of a skew group are considered for skew balancing.

```
set_db pin:$pin .cts_sink_type ignore
```

7.3.3 Exclude Pin

An *exclude pin* is similar to an *ignore pin*, but it is also removed from the clock tree. Honestly, I am not sure how this actually affects us, other than that it will not be displayed in the reports and clock tree debugger...

```
set_db pin:$pin .cts_sink_type exclude
```

7.3.4 Macro Clock-Input Pin

The clock input pins of macros (.lib model) must usually be earlier than other sinks, which means they will have a lesser clock arrival time to take account of the internal clock path inside the macro. This is demonstrated in Figure 21, showing that the macro clock should arrive X before the insertion delay of the other leaf cells.

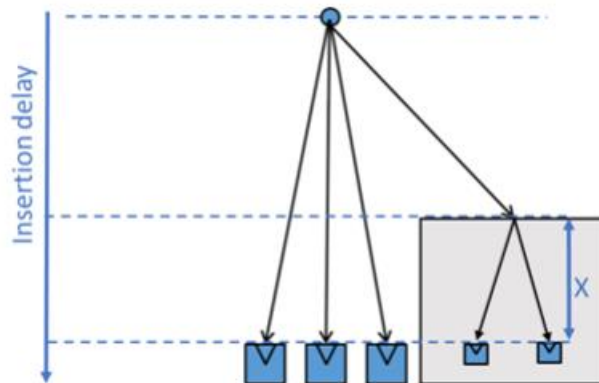


Figure 21: Macro Clock-Input Pin

To define this, use the `.cts_pin_insertion_delay` attribute, for example:

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

Note that a positive value is how long *before* the average insertion delay that the clock should arrive at the macro clock-input.

7.4 Define CTS Constraints

7.4.1 Clock Period

Tell the CTS engine what the clock period is:

```
set_db port:CLK .cts_clock_period $PERIOD
```

7.4.2 Max Transition

```
set_db cts_target_max_trans $max_clock_trans
```

7.4.3 Max Fanout

```
set_db cts_max_fanout $max_fanout
```

7.4.4 Max Capacitance

```
set_db cts_target_max_capacitance $max_cap
```

7.4.5 Case Analysis

```
set_db port:TEST_MODE .cts_case_analysis 0
```

7.5 Define Library Cells

7.5.1 Driving Cell for Clock Net

```
set_db clock_tree:clock .cts_source_driver BUFFX8/Z
```

7.5.2 Define Buffers and Inverters for building a clock tree

```
set_db cts_buffer_cells $CLOCK_BUF_LIST
```

```
set_db cts_inverter_cells $CLOCK_INV_LIST
```

7.5.3 Recommendations for CTS Library Cells

- Use low threshold (LVT) cells. The resulting insertion delay will be lower leading to less impact of OCV timing derates, therefore reducing the datapath dynamic and leakage power increase from timing optimization.
- For many, but not all, low geometry processes inverters result in lower insertion delay and lower power than buffers. In older technologies, buffers may be more efficient. The exact preference here is technology, library, and design target dependent.
- Permitting the largest library cells to be used may be undesirable for electromigration reasons and can increase clock power.
- Very weak cells, for example, X3 and below in many libraries, are usually undesirable due to poor cross-corner scaling characteristics and are sensitive to detailed routing jogs and changes.
- Limiting the number of library cells to no more than 5 per cell type may help reduce run time.
- Do not exclude small cells, such as X4, as otherwise CTS will be forced to use larger more power and area consuming cells to balance skew or implement the useful skew schedule.
- Include always-on buffers and inverters in designs with multiple power domains.

7.6 Define Non-Default Rules for Clock Routing

7.6.1 Define Non-Default Rules

A Non-Default Rule (NDR) is a routing rule, such as “double-space, double-width”, that is applied to specific nets (such as clock nets) during routing. NDRs (called “routing rules” in Innovus) are either defined in the TechLEF file or using the `create_route_rule` command:

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

7.6.2 Define Route Types

A route type binds a non-default routing rule, preferred routing layers, and a shielding specification together.

```
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
```

7.6.3 Specify Route Type usage on Clock Nets

CTS uses the concept of top, trunk and leaf nets, as shown in Figure 22 and explained below:



Figure 22: CTS Route Types

- Leaf nets – Any net that is connected to one or more clock tree sinks is a leaf net. By default, CTS will insert buffers so that no buffer drives both sinks and internal nodes.
- Trunk nets – Any net that is not a leaf net is by default a trunk net.
- Top nets – If you configure the `cts_routing_top_fanout_threshold` attribute, then any trunk net that has a transitive fanout sink count higher than the configured count threshold will instead be a top net. For example, if the attribute is set to 10,000, then any trunk net that is above (in the clock tree fan-in cone) 10,000 or more sinks will be a top net.

To specify that the route types to be used for `leaf`, `trunk`, and `top` nets, set the following database attributes:

```
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
```

7.6.4 Routing Rule Recommendations

Clock net routing rules are crucial to obtaining low insertion delay and avoiding signal integrity problems. Especially for small geometry process nodes, the following recommendations should be considered:

- Configure the **trunk** net type to use **double width double spacing** and **shielding**. Prefer **middle to higher layers** subject to the power grid pattern. **Double width** is recommended to reduce resistance and permit use of bar shape vias, with **double spacing** to reduce the capacitance impact of shielding. **Shielding** is essential to avoid aggressors impacting clock trunk net timing, as impact on clock trunk timing is often significant for both WNS and TNS.
- Configure the **leaf** net type to use **double width** and prefer **middle layers**. **Double width** is recommended to reduce resistance. Extra spacing is desirable, but extra spacing and/or shielding may consume too much routing resource.
- Try to arrange for each net type (**leaf**, **trunk**, **top**) to use a single layer pair, one horizontal and one vertical, which have the same pitch, width, and spacing. This increases the correlation accuracy between routing estimates before clock nets are routed and actual routed nets.

7.7 Running Clock Tree Synthesis

There are two modes to run CTS in Innovus – Clock Concurrent Optimization and Skew Balanced (Traditional) Clock Tree Synthesis.

7.7.1 Clock Concurrent Optimization

To run Clock Concurrent Optimization with heavy doses of useful skew, use the **ccopt_design** super command:

```
ccopt_design
```

There is no need to run timing optimization after running Clock Concurrent Optimization.

7.7.2 Skew Balanced Clock Tree Synthesis

To create a traditional, skew-balance clock tree, run the **clock_design** command. However, first you will need to define skew targets for your skew groups:

```
set_db skew_group:clock .target_skew 0.2
```

Then you can run **clock_design**:

```
clock_design
```

And finally run post CTS timing optimization for both setup and hold:

```
opt_design -post_cts
```

```
opt_design -post_cts -hold
```

7.7.3 Minimum Insertion Delay Clock Tree Synthesis

You may want to run a buffer tree insertion algorithm on the clock tree that just fixes DRVs without actually considering skew or timing. In other words, build a minimum insertion delay clock tree. This, in fact, is the first stage of the CCOpt engine. An example for a reason to do this would be to quickly see what is going on in your clock tree (i.e., debugging) to fix configurations before moving on to a much longer CCOpt run, which could take many hours.

This mode of running CCOpt is known as “cluster mode” and it is set using:

```
set_db clock_tree:clock .cts_opt_ignore true
```


7.8 CCOpt Clock Tree Debugger

The CCOpt Clock Tree debugger is a GUI feature that enables you to graphically display your clock tree and see all the sources, sinks and their skew. This can be used before CTS to see that your clock trees and skew groups have been properly defined, and after CTS to analyze the results. The debugger can be opened from the command line with the `gui_open_ctd` command or from **CLOCK → CCOPT CLOCK TREE DEBUGGER...**

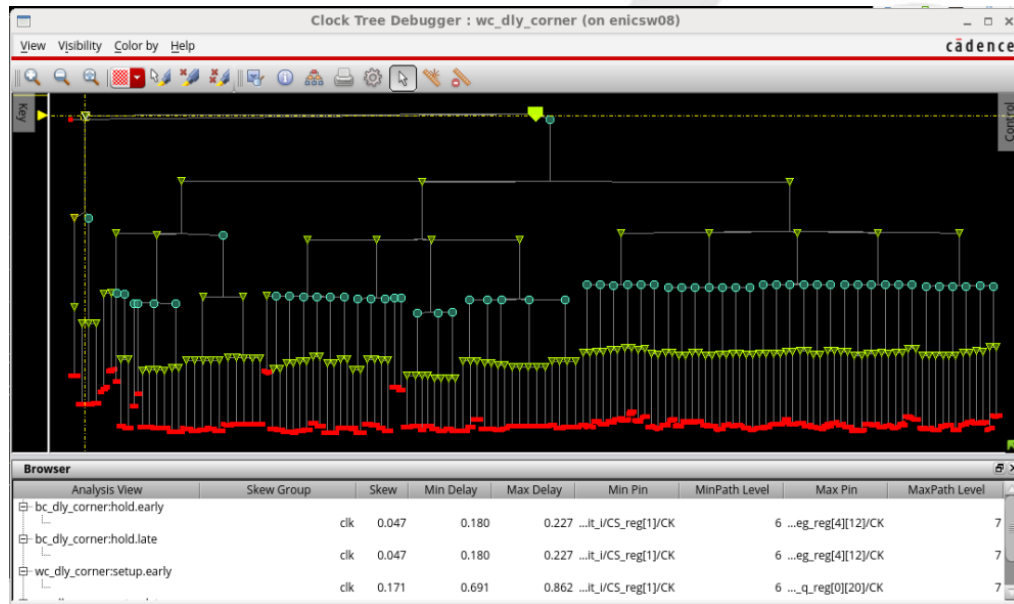


Figure 23: Example of CCOpt Clock Tree Debugger after CTS

7.9 Clock Tree Reports

7.9.1 Post CTS Timing

As usual, use the `time_design` super command, but now check *hold*, as well as *setup*:

```
time_design -post_cts -path_report -drv_report -slack_reports \
    -num_paths 50 -prefix postCTS -report_dir ../reports/cts

time_design -post_cts -hold -path_report -drv_report -slack_reports \
    -num_paths 50 -prefix postCTS -report_dir ../reports/cts
```

7.9.2 Reporting Clock Trees

To see a complete report about the clock trees, use the `report_clock_trees` command.

```
report_clock_trees
```

7.9.3 Reporting Skew Groups

To see a complete report about the skew groups, insertion delays, etc. use the `report_skew_groups` command.

```
report_skew_groups
```

8 Route

Routing in Innovus is achieved with the NanoRoute engine. This engine concurrently runs timing-driven and signal integrity (SI) driven routing. Additionally, it can perform multi-cut via insertion, wire widening and spacing.

8.1 Setting Route Options

Route options are set with the `route` database attributes, which can also be accessed or through the Global and Attribute options forms, show in Figure 24 and Figure 25.

Figure 24: The Route Attributes Form – Accessed from the NanoRoute Form, Attributes button

Name	Category	Value	Description
route_design_high_freq_search_repair	route	false	run search and repair to remove
route_design_high_freq_shield_trim_length	route	0.0	specify minimum length of shield
route_design_honor_power_domain	route	FALSE	honor power domain routing
route_design_ignore_antenna_top_cell_pin	route	TRUE	ignore antenna check on block I/O
route_design_ignore_follow_pin_shapes	route	FALSE	ignore follow pin via shapes
route_design_number_fail_limit	route	0	set limit for number of fails
route_design_number_thread	route	1	set the number of processors to t
route_design_number_warning_limit	route	0	set limit for number of warnings
route_design_process_node	route		Specify the process node
route_design_rc_extraction_corner	route		specific which RC extraction corne
route_design_relaxed_route_rule_spacing_to_pg_nets	route	none	relax the spacing requirement fro
route_design_reserve_space_for_multi_cut	route	FALSE	Reserves space to insert multicut
route_design_reverse_direction	route		reverse routing direction in area:
route_design_route_clock_nets_first	route	TRUE	route clock nets first
route_design_selected_net_only	route	FALSE	route selected net only
route_design_shield_crosstie_offset	route		Specifies the offset in terms of nu
route_design_skip_analog	route	FALSE	skip routing nets or pins marked
route_design_strict_honor_route_rule	route	false	Strictly enforce non-default rules

Figure 25: The Route Database Attributes – Accessed from the NanoRoute Form, Global button

8.1.1 NanoRoute Mode Defaults

By default, both timing-driven and SI-driven are set, when using the `route_design` command, i.e.:

```
set_db route_design_with_timing_driven true
set_db route_design_with_si_driven true
```

In addition, clock nets are unfixed to enable ECOs for post_cts optimization and for fixing DRVs, i.e.:

```
set_db route_design_fix_clock_nets false
```

These defaults can be changed before running `route_design`.

8.1.2 Create Shielding for a certain Net

To create shields, use the `set_route_attributes` command:

```
set_route_attributes -nets net1 -shield_nets vss
route_global_detail
```

There is also this set of commands (the manual is not clear what does what...)

```
set_db net:clock .shield_nets VDD
setAttribute -net CLK -shield VDD
route_add_shield
```

8.2 Running NanoRoute

The `route_opt_design` command is a super command that runs both global and detail routing, followed by post route optimization:

```
route_opt_design
```

Alternatively, the older `route_design` command can be used to run global and detail routing without running optimization:

```
route_design
```

8.3 DRC/LVS Checking

8.3.1 Run DRC

To run an initial DRC, use the `check_drc` command or from the GUI, select **CHECK→CHECK DRC**. The Verify DRC form is shown in Figure 26, and the command can be run as:


```
check_drc
```



Figure 26: The Verify DRC Form

The `check_drc` command will divide the design into sub-areas and can distribute the DRC tasks over multiple CPUs. Depending on the size of your design and the complexity of your process, these checks can take some time. At the end, you will get a summary of the violations found, categorized with names, such as "SameNet", "Antenna", "Short" and "Overlap". These violations can be accessed through the Violation

Note that check_drc is not a "sign-off" tool. It only checks for rule violations. It should not be used to sign off DRC tool, such as the "sign-off" tool.

Browser at **TOOLS→VIOLATION BROWSER** or pressing the  icon. A sample state of the violation browser is shown in Figure 27. This form enables operations, such as automatic zoom in to the area of the violation.

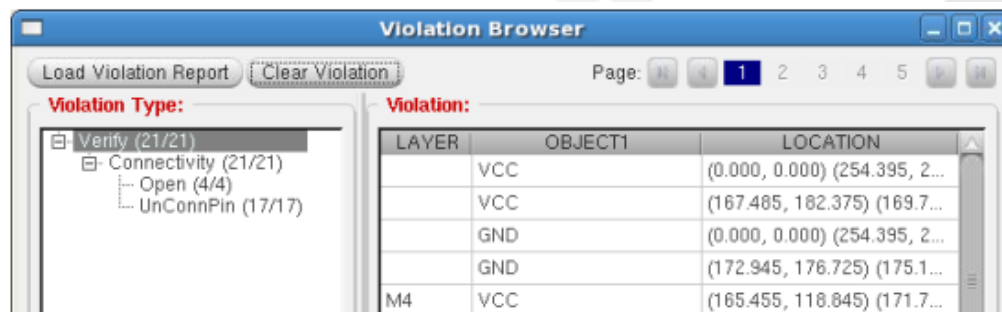


Figure 27: Violation Browser

Pay attention that the `check_drc` command is affected by options database attributes in the `check_drc` category.

```
get_db -category check_drc *
```

For example, using these settings, you can limit `check_drc` to only check a certain area (for a much faster incremental DRC run).

```
set_db check_drc_area { 83.831 87.597 831.406 830.806 }
check_drc
```

8.3.2 Run LVS

To run an initial LVS, use the `check_connectivity` command or from the GUI, select **CHECK→CHECK**

CONNECTIVITY. Violations are debugged with the Violation Browser, accessible from **TOOLS→VIOLATION**

Similarly, for `check_drc`, the `check_connectivity` command is nowhere near a sign-off LVS tool.

You should use a more or PVS to achieve this level of LVS.

```
check_connectivity
```

8.3.3 Check for Antenna Violations

To check for antenna violations, use the `check_process_antenna` command:

```
check_process_antenna
```

8.3.4 Loading Violations from Calibre

If you run DRC/LVS from a sign off tool, such as Calibre, you can load the markers back into Innovus. You just have to figure out the offset from the origin that is seen in the GDS of the chip/block. If the offset is (\$x, \$y) and the Calibre database is saved at \${TOPLEVEL}.drc.db then you would load the markers with:

```
read_marker ${TOPLEVEL}.drc.db -type Calibre -x_offset $x -y_offset $y
```

8.3.5 Clearing Violations

To clear up all the markers left by previous verification checks, run:

```
delete_drc_markers
```

8.4 Fixing Post Route Problems

This is always a headache, and it can be very hard to do. Sometimes, you have to go down the painful path of exporting your design to Virtuoso and fixing the DRCs by hand. However, EDI provides many ways to fix DRCs systematically, so you should try them first (this is by no means an exhaustive list).

8.4.1 Instance Overlaps

One major problem that looks like a routing problem is actually illegal placement. This can be caused by some utilization/congestion or other floorplanning problem. For example, if the floorplan is too dense, the placer will put cells on top of each other (this includes CTS cells or new buffers during optimization) and they will cause a million shorts that will screech out during DRC check.

This is a relatively “easy” or at least “straightforward” fix.

- Run `place_detail -eco true`. If it can, the legalization algorithm will shift the cells to remove overlaps.
- If `place_detail` fails, then there are **FIXED** cells or not enough room. Change the status and/or manually move the cells and run `place_detail` again.
- If there just is no room to move the cells, tough luck. Go back and redo your floorplan – make it bigger, reduce the utilization in that area, apply placement blockages, or use one of the other methods we’ve discussed.

8.4.2 Try and fix violations

There are some automatic commands to try and fix violations, such as these two commands:

- Delete routes with DRCs and re-route them:

```
delete_routes_with_violations
```

- Run and ECO route that tries to fix DRC violations:

```
route_eco -fix_drc
```

8.4.3 Reroute a problematic area

There are various ways to make the router try harder or incrementally route a certain area. Since routing can take really really long, making it focus on a certain area is a very useful approach. The easiest way in my opinion is to open the **ROUTE→NANOROUTE→ROUTE...** form, select **AREA ROUTE** and click on **SELECT AREA AND ROUTE**, as shown in Figure 28. Then select a box around the problematic area, wait until it finishes, and see if your DRCs disappeared.

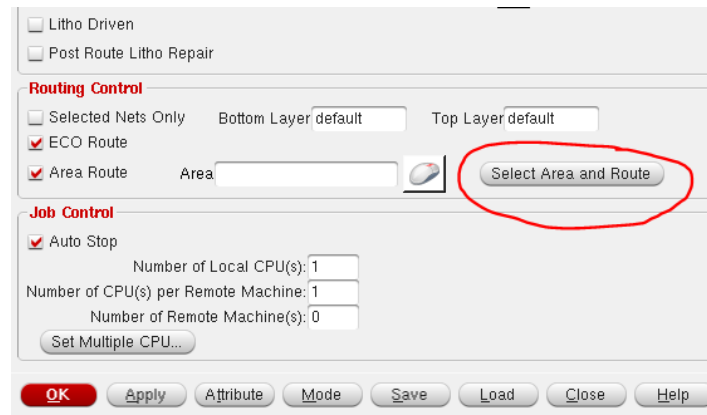


Figure 28: Fixing DRCs with Select Area and Route

8.4.4 FIXED Nets and Instances

Often the problem is due to a wire being marked as **FIXED**. For example, if the wire was routed during CTS, it is probably considered **FIXED** and shouldn't be tampered with (since the whole CTS analysis used the extraction of this particular routing). However, it seems that the router used during CTS sometimes overlooks some DRCs, causing a problem that will not automatically be taken care of.

To solve this problem, change the status of the wire to **ROUTED** and make an incremental route.

Similar problems can be caused by pre-routes, such as power routing, which may be somewhere that physically blocks the router from achieving a DRC clean route, and furthermore, some cells may be pre-placed or **FIXED** where a pin is blocked by a **FIXED** net or something similar. Garbage-In, Garbage-Out.

Note that after placement, the default settings of Innovus will not move clocked elements (i.e., registers). This can cause, such unroutable situations, where a pin of a **FIXED** flip flop is inaccessible or something. Just move the problematic instance...

8.4.5 Brute Force It!

When nothing else works, roll up your sleeves and use the trustworthy Brute Force technique. In other words:

- 1) Select a bunch of wires in the problematic area, hit delete, and then reroute the area.
- 2) If that doesn't work, select a few standard cells around the DRC violation, move them, legalize the placement ([place_detail](#)) and reroute the area. Works like a charm!

8.4.6 Fix it manually... In Innovus

Well, good luck with this one, but Innovus provides tools for push polygon manual routing. It's very painful, but if you get the hang of it, it could be scripted and stay internal to the tool, rather than going out to Virtuoso and possibly coming back...

8.5 PostRoute Parasitic Extraction and Timing

8.5.1 RC Extraction Mode

Parasitics extraction is achieved by Innovus with the [extract_rc](#) command, which is run as part of many commands, such as [report_timing](#) (if an updated extraction isn't present). To make sure you are getting accurate post-route timing, adjust the [extract_rc](#) database attributes:

```
set_db extract_rc_engine post_route
```



```
set_db extract_rc_effort_level <low | medium | high | signoff>
set_db extract_rc_coupled true
```

The `effort_level` setting is very important at this time. It defaults to `low`, which uses the native (fast) extraction engine, but for more accurate extraction, use the `medium` (Quantus QRC), `high` (Integrated QRC) or `signoff` (Standalone QRC) levels.

Note that if you are using a third-party tool, the resulting SPEF file can be imported for each corner, as follows:

```
read_spec rc_corner1.spef -rc_corner rc_corner1
```

8.5.2 Analysis Mode

Timing analysis is controlled by the `timing_analysis` database attributes:

- **On Chip Variation (OCV):** OCV analysis derates the launch path and accelerates the capture path during setup check and applies the opposite during hold checks. To enable OCV analysis:

```
set_db timing_analysis_type ocv
```

- **Clock Pessimism Recovery Removal (CPRR):** CPRR brings timing a bit back to reality by removing the derating on the part of the clock path that is shared by both the launch and capture clocks. To apply CPRR analysis to both hold and setup checks:

```
set_db timing_analysis_cprr both
```

- **SI CPRR:** Additionally, you can apply signal integrity CPRR with the following global variable:

```
set_db timing_enable_si_cprr true
```

8.5.3 Delay Calculation Mode

If that wasn't enough, an additional mode setting affects the timing calculation, controlled by the `delaycal` database attributes.

- For SI-Aware delay calculation:

```
set_db delaycal_enable_si true
```

8.6 Post Route Optimization

In general, post-route optimization is run concurrently with global and detail route if the `route_opt_design` command is used. The following discusses the independent flow (`route_design` followed by `opt_design -post_route`); however, the concepts and settings are relevant for the optimization run during `route_opt_design`, as well.

8.6.1 Post Route Timing Optimization

The post route timing optimization engine can cut wires, insert buffers and locally update the timing graph to evaluate the change. Post-route optimization is run with:

```
opt_design -post_route
```

Since hold needs to be optimized after route, as well, repeat with the `-hold` option:

```
opt_design -post_route -hold
```

Or run them together...

```
opt_design -post_route -setup -hold
```

- **Multi-VT Optimization:** To allow cell swapping between multi-VT options at the post-route stage:

```
set_db opt_allow_only_cell_swapping true
```

8.6.2 Post Route Wire Optimization

NanoRoute has several heuristics for improving the wiring for better yield (DFM), reducing resistance and capacitance, and improving SI (lowering coupling capacitance). These are controlled with the following options:

- Wire Spreading:

```
set_db route_design_detail_post_route_spread_wire true
```

- Via reduction:

```
set_db route_design_concurrent_minimize_via_count_effort high
```

- Apply multi-cut vias for vias that cannot be eliminated:

```
set_db route_design_detail_use_multi_cut_via_effort high
```

Incremental routing for wire optimization is then run with the `-wire_opt` option:

```
set_db route_design_with_timing_driven false  
route_design -wire_opt
```

8.6.3 Via Optimization

```
routeDesign -via_opt
```

8.6.4 Fix Antenna Violations

To fix antenna violations during post route ECO, define the following settings:

```
set_db route_design_antenna_cell_name <CELLNAME>  
set_db route_design_detail_fix_antenna true  
set_db route_design_antenna_diode_insertion true
```

9 Finalize Design

9.1 Insert Fillers

To add filler cells, either use the `add_fillers` command and its associated `add_fillers` database attributes to set the command options, or run through the GUI at **PLACE→PHYSICAL CELL→ADD FILLER**, as shown in Figure 29.

```
add_fillers -base_cells <list of filler cells> -prefix FILLER -fix_drc
```

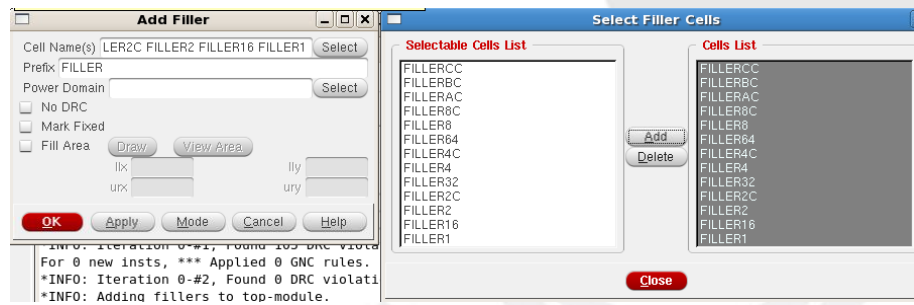


Figure 29: Add Fillers Form with Select Filler Cells Form

Not automatically remove the fillers, perform optimization, and re-insert them.

9.2 Insert Decap Cells

DeCap cells can add MOSCAPs between VDD and GND to provide better local voltage stability. These are added with the `add_decaps` command.

```
add_decaps -cells <decap list> -prefix FILLER_DECAP -area {x1 y1 x2 y2}
```

Some recommendations for adding DeCaps could be:

- Add DeCaps adjacent to Clock Buffers in order to improve the resilience to voltage droops that would affect the clock.
- Same principle but next to Flip Flops to make sure the setup and hold times are more stable.
- Add DeCaps instead of regular fillers to the entire design. But beware that in some processes, the leakage through DeCaps is substantial.

9.3 Add Metal Density Fill

Innovus has an internal tool to insert Metal Density Fill. However, the rules may be complex and you may need to use an external tool (such as Calibre DRC) to reach signoff level for density fillers, and especially for front-end masks (such as poly and diffusion fillers).

9.3.1 Adding Metal Fill in Innovus

To add fillers in Innovus, use the `add_metal_fill` and `add_via_fill` commands or from the GUI **ROUTE→METAL FILL→ADD**. In general, the metal fill rules should be defined in the LEF file. However, if you want to override these rules, use the `set_metal_fill` and `set_via_fill` commands or from the GUI **ROUTE→METAL FILL→SETUP**.

```
set_via_fill -layer "Via23" -window_size 50 50 -window_step 25 25 \
-min_density 0.005 -max_density 30
```

```
add_via_fill -layer "Via23" -mode floatingOnly -area "0 0 100 100"  
set_metal_fill -layer 1 -window_size 200 200 -window_step 100 100  
add_metal_fill -layers {1 2 3} -area 100 200 300 400 -nets {VDD GND}
```

9.3.2 Verifying Metal Density

After inserting metal fill, use the `check_metal_density` command to verify the DRC requirements are met.

```
check_metal_density
```

9.3.3 Using an external tool for metal fill

To add metal fill with an external tool, stream out the GDS and apply the metal density fill in the external tool. For a Cadence PVS flow, this can be done from within EDI:

```
write_stream -map_file gds_map -output_macros gds_file  
eval_legacy {run_pvs_metal_fill -ruleFile PVS_RULE_DECK -defMapFile map_file \  
-gdsFile gds_file -cellgds_top_cell}
```

9.3.4 Trimming the metal fill for timing

Adding metal fill can add unexpected timing problems due to increased parasitics. To trim back the metal layers close to critical nets, use the `trim_metal_fill_near_net` command or from the GUI

ROUTE→METAL FILL→TRIM:

```
trim_metal_fill_near_net -slack_threshold $slack1 -spacing value \  
-spacing_bove value -spacing_below value -min_trim_density value
```

10 Generating Reports

10.1 Design Summary

You can automatically generate a summary report that can be exported as a text file or in a very convenient HTML file that can be viewed with a web browser. This is done with the `report_summary` command or from the GUI at **FILE→REPORT→SUMMARY**.

10.2 Netlist Statistics

To get statistics, such as number of cells, fanout distribution of nets, number of instances of each standard cell, etc., use the `report_netlist_statistics` command or from the GUI **FILE→REPORT→NETLIST STATISTICS**.

```
report_netlist_statistics
```

10.3 Gate Count

To get a gate count and area report for each module, use the `report_gate_count` command or from the GUI **FILE→REPORT→GATE COUNT**.

```
report_gate_count -level 5 -limit 100
```

10.4 Check Timing

```
check_timing -verbose
```

10.5 Report Timing

Either use the `time_design` super command or directly use the `report_timing` command.

11 Exporting Your Design

11.1 Export Your Netlist

Use the `write_netlist` command or from the GUI, **FILE→SAVE→NETLIST**.

- For SDF Back Annotation

```
write_netlist -exclude_leaf_cells ../export/${TOPLEVEL}.PLS.v
```

- For LVS

```
write_netlist -include_phys_cells -exclude_leaf_cells ../export/${TOPLEVEL}.LVS.v
```

11.2 Export an SDF for Back Annotation

Use the `write_sdf` command:

```
write_sdf -view analysis_view_wc ../export/${TOPLEVEL}.sdf
```

11.3 Export Your GDS

Use the `write_stream` command or from the GUI, **FILE→SAVE→GDS/OASIS**.

```
write_stream ../export/${TOPLEVEL}.gds -map_file $MAPFILE \  
-lib_name ${TOPLEVEL} -structure_name ${TOPLEVEL} \  
-merge $GDS_FILES -units 1000 -mode ALL
```


12 Example Files

12.1 SDC File

```
create_clock -name clock -period 10.0 [get_ports clock]
set_clock_uncertainty 0.2 clock
set_input_delay 1.0 -clock clock [all_inputs]
set_driving_cell -lib_cell PRDWUW0408SCDG_HI -pin PAD [all_inputs]
set_output_delay 1.0 -clock clock [all_outputs]
set_load 0.3 [all_outputs]
set_max_delay 5.0 -from [all_inputs] -to [all_outputs]
```

12.2 MMMC View Definition File

```
# Constraint Modes #
# ----- #
create_constraint_mode \
    -name functional_mode \
    -sdc_files $design(functional_sdc)

# RC Corners #
# ----- #
create_rc_corner \
    -name bc_rc_corner \
    -T $tech(TEMPERATURE_BC) \
    -qrc_tech $tech_files(QRC TECH_FILE_BC)

create_rc_corner \
    -name tc_rc_corner \
    -T $tech(TEMPERATURE_TC) \
    -qrc_tech $tech_files(QRC TECH_FILE_TC)

create_rc_corner \
    -name wc_rc_corner \
    -T $tech(TEMPERATURE_WC) \
    -qrc_tech $tech_files(QRC TECH_FILE_WC)

# Library Sets #
# ----- #
create_library_set \
    -name bc_libset \
    -timing $tech_files(ALL_BC_LIBS)

create_library_set \
    -name tc_libset \
    -timing $tech_files(ALL_TC_LIBS)

create_library_set \
    -name wc_libset \
    -timing $tech_files(ALL_WC_LIBS)

# Timing Conditions #
# ----- #
create_timing_condition \
    -name bc_timing_condition \
    -library_sets bc_libset
```

```

create_timing_condition \
    -name          tc_timing_condition \
    -library_sets tc_libset

create_timing_condition \
    -name          wc_timing_condition \
    -library_sets wc_libset

# Delay Corners #
# ----- #
create_delay_corner \
    -name bc_dly_corner \
    -timing_condition bc_timing_condition \
    -rc_corner bc_rc_corner

create_delay_corner \
    -name tc_dly_corner \
    -timing_condition tc_timing_condition \
    -rc_corner tc_rc_corner

create_delay_corner \
    -name wc_dly_corner \
    -timing_condition wc_timing_condition \
    -rc_corner wc_rc_corner

# Analysis Views #
# ----- #
create_analysis_view \
    -name bc_analysis_view \
    -constraint_mode functional_mode \
    -delay_corner bc_dly_corner

create_analysis_view \
    -name tc_analysis_view \
    -constraint_mode functional_mode \
    -delay_corner tc_dly_corner

create_analysis_view \
    -name wc_analysis_view \
    -constraint_mode functional_mode \
    -delay_corner wc_dly_corner

# Selected Analysis Views #
# ----- #
set_analysis_view \
    -setup $design(selected_setup_analysis_views) \
    -hold  $design(selected_hold_analysis_views)

```

12.3 I/O Definition File

This file is used to provide a location for the I/O cells around the periphery of the chip. The exact syntax is described in the user manuals, and an I/O file can be exported from EDI, but for your convenience, an example is provided below:

```
(iopad
```

```

(topleft
  (inst name="pad_corner1"      orientation=R90 )
)
### left side (W, bottom to top)
(left
  (inst name="pad_vcc_p2"      offset=140.12 ) # pin no: 14
  (inst name="pad_DIxDI_0"    offset=220.10 ) # pin no: 12
  (inst name="pad_DIxDI_1"    offset=300.08 ) # pin no: 12
  (inst name="pad_DIxDI_2"    offset=380.06 ) # pin no: 11
  (inst name="pad_DIxDI_3"    offset=460.04 ) # pin no: 10
  (inst name="pad_mvdd_VAR"    offset=540.02 ) # pin no: 9
  (inst name="pad_gnd_c1"      offset=620.00 ) # pin no: 8
  (inst name="pad_vcc_c1"      offset=699.98 ) # pin no: 7
  (inst name="pad_WAddrxDI_0"  offset=779.96 ) # pin no: 6
  (inst name="pad_WAddrxDI_1"  offset=859.94 ) # pin no: 5
  (inst name="pad_StartBISTxSI" offset=939.92 ) # pin no: 4
  (inst name="pad_ResetxSI"    offset=1019.90 ) # pin no: 3
  (inst name="pad_ClkxCI"      offset=1099.88 ) # pin no: 2
  (inst name="pad_gnd_p1"      offset=1179.86 ) # pin no: 1
)
(bottomleft
  (inst name="pad_corner2"      orientation=R180 )
)

```