NanoRoute® Technology Reference

Product Version 7.1
October 2007

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

About This Manual

The NanoRoute® router is a high-speed, high-capacity concurrent routing and optimization system. It performs simultaneous routing, extraction, and signal integrity optimization of a placed design, and supports both multi-threading and distributed routing. This reference guide describes the baseline functionality of the NanoRoute technology engine that is found in NanoRoute Ultra SoC routing solution, Nano Encounter® implementation system for flat designs, and SoC Encounter™ hierarchical RTL-to-GDSII physical implementation solution.

Audience

This reference guide describes the NanoRoute commands and options. It is intended for experienced designers of digital integrated circuits. Such designers must be familiar with signal routing and have a solid understanding of UNIX and Tcl/Tk programming.

For additional information about NanoRoute technology, contact Cadence sales and technical support personnel.

Chapter Summary

This manual is organized into the following chapters:

- Chapter 1, “Introduction”

  Describes NanoRoute features and design flow.

- Chapter 2, “Getting Started”

  Describes supported platforms, license requirements, installation, support file formats, NanoRoute syntax, and additional general information about starting and using the standalone NanoRoute router.

- Chapter 3, “Using the NanoRoute Router”

  Describes common tasks you can perform with the standalone NanoRoute router.

- Chapter 4, “Command Reference”

  Describes the NanoRoute commands.

- Chapter 5, “Option Reference”

  Describes the NanoRoute options.

- Appendix A, “Timing Constraint Compatibility”

  Describes NanoRoute router support for timing commands.

Conventions Used in This Manual

`text`	Indicates text that you must type exactly as shown. For example: `analyze_connectivity -analyze all`
`text`	Indicates information for which you must substitute a name or value. In the following example, you must substitute the name of a specific file for *`configfile`*: `wroute filename` *`configfile`*
text	Indicates the following: ■ Text found in the graphical user interface (GUI), including form names, button labels, and field names ■ Terms that are new to the manual, are the subject of discussion, or need special emphasis ■ Titles of manuals
`[ ]`	Indicates optional arguments. In the following example, you can specify none, one, or both of the bracketed arguments: `command [-arg1] [arg2` *`value`*`]`
`[ | ]`	Indicates an optional choice from a mutually exclusive list. In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one: `command [arg1 | arg2 | arg3 | arg4]`
`{ | }`	Indicates a required choice from a mutually exclusive list. In the following example, you must specify one, and only one, of the arguments: `command {arg1 | arg2 | arg3}`

{ }	Indicates curly braces that must be entered with the command syntax. In the following example, you must type the curly braces: `command arg1 {x y}`
. . .	Indicates that you can repeat the previous argument.
. . .	Indicates an omission in an example of computer output or input.
Command – Subcommand	Indicates a command sequence, which shows the order in which you choose commands and subcommands from the GUI menu. In the following example, you choose *Floorplan* from the menu, then *Power Planning* from the submenu, and then *Add Rings* from the displayed list: *Floorplan – Power Planning – Add Rings* This sequence opens the Add Rings form.

Related Documents

For more information about NanoRoute, consult the following Cadence documents. You can access these and other documents with the CDSDoc online documentation system.

- *Encounter Database Access Command Reference*

  Lists all of the Encounter database access commands and provides a brief description of syntax and usage.

- *Encounter Known Problems and Solutions*

  Describes important Product Change Requests (PCRs) for the Encounter family of products, including solutions for working around known problems.

- *Encounter Library Development Guide*

  Describes library development guidelines for the independent tools that make up the Encounter family of products.

- *Encounter Menu Command Reference*

  Provides information specific to the forms and commands available from the Encounter graphical user interface.

- *Encounter Text Command Reference*

  Describes the Encounter text commands, including syntax and examples.

- *Encounter Timing Closure Guide*

  Addresses issues related to timing closure for challenging designs. It presents a recommended task flow and provides tips on how to resolve or avoid common timing closure problems typically found in a design.

- *Encounter User Guide*

  Describes how to install, configure, and use the Encounter graphical user interface (GUI). This document also provides strategies for performing specific tasks with the Encounter software, using both the GUI and text commands.

- *LEF/DEF Language Reference*

  Describes Library Exchange Format (LEF) and Design Exchange Format (DEF) syntax.

- *What's New in Encounter*

  Provides information about new and changed features in this release of the Encounter family of products.

- README file

  Contains installation, compatibility, and other prerequisite information, including a list of product change requests (PCRs) that were resolved in this release. You can read this file online at downloads.cadence.com.

For a complete list of documents provided with this release, see the CDSDoc library.

10/8/07

1

Introduction

Features

The NanoRoute® router performs concurrent signal integrity, timing-driven, and manufacturing aware routing (SMART routing) of cell, block, or mixed cell and block level designs. The router is optimized for routing designs with the following features:

- More than 300K instances or nets and at least five routing layers
- 180 nanometer or smaller process technology
- Signal integrity critical
- Timing critical
- Detailed-model (full-model) abstracts

The router provides the following features:

- Concurrent management of *S*ignal integrity, *M*anufacturing *A*wareness, *R*outability, and *T*iming issues during routing (SMART routing)
- 65 nm, 90 nm,130 nm, and 180 nm process rule support
- Distributed routing on multiple workstations running multi-threading (Superthreading)
- Complex wide wire spacing rule support, density checking and fill, minimum area, and notch (donut) avoidance
- Via stacking, diagonal spacing, and density rule support
- LEF file optimization
- Automatic track generation
- Manufacturing aware routing support, including wire spreading and support for redundant vias
- Open timing architecture, including support for Encounter common timing engine (CTE) and external timing engines such as PrimeTime
- Support for third-party data files, including customized LEF technology files and Astro files, and older-syntax LEF files

What the NanoRoute Router Does

The router performs simultaneous routing, extraction, and signal integrity optimization of multi-layer placed designs and can be multiprocessed on a multiple-CPU machine. It can

handle multimillion instance designs for cell-based, mixed standard-cell and block, and top-level SoC based designs.

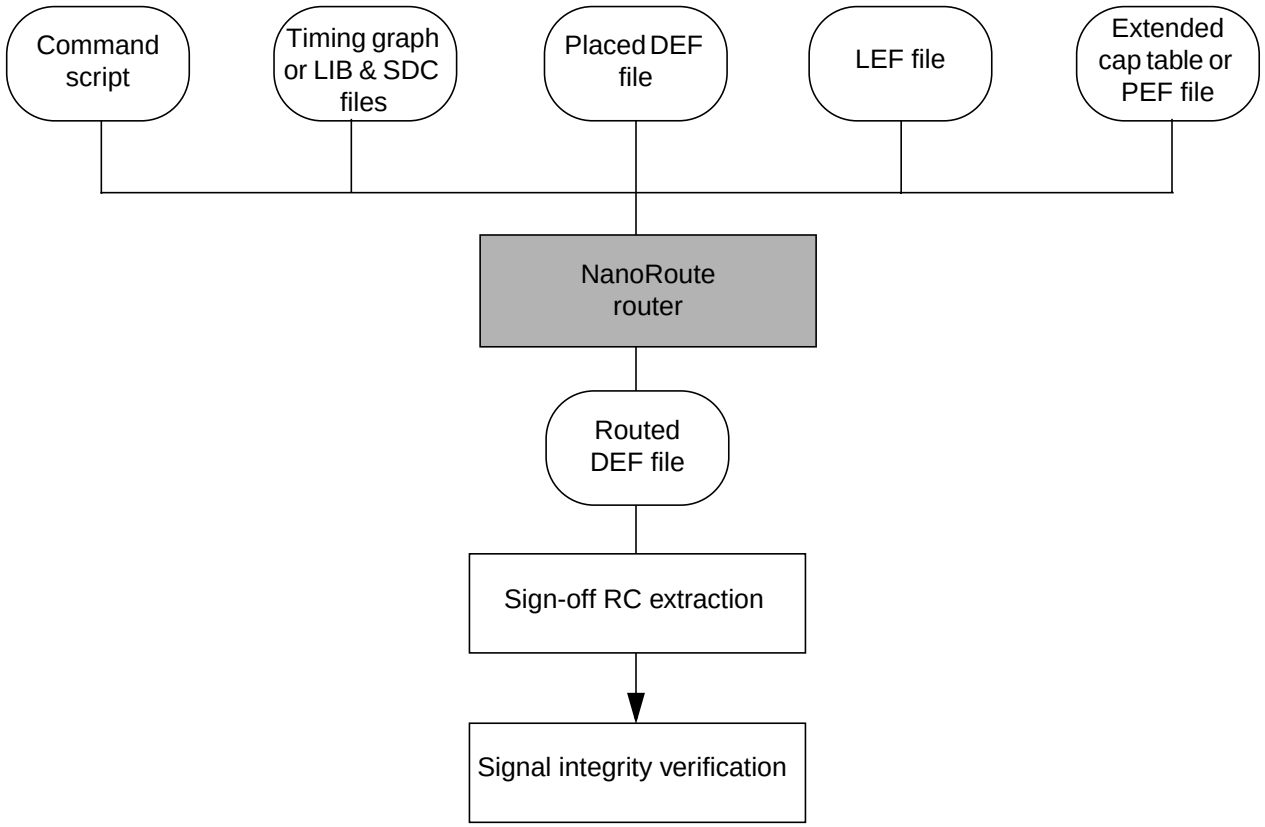
The NanoRoute router performs most effectively on designs with four or more metal layers.

You can use the router in either a graphical environment or batch mode. During routing, it routes to avoid design rule violations and minimizes wiring resources such as wire lengths and vias. The router also honors the preexisting routing of partially or fully routed special nets. A complete routing process performed by the NanoRoute router includes global and detailed routing.

The routed design is output to routed DEF. If you are running timing analysis with the internal NanoRoute timing engine, you can also output an SDF file to use in timing verification.

Routing Flow

The following figure shows the place-and-route design flow with the standalone NanoRoute router.



Terminology

The following terms are used throughout this manual.

Block

A block refers to the geometric and electrical description of a custom-designed macro circuit, such as a full-custom circuit block or memory mega-cell.

Blockage

Blockage refers to interconnect layer geometries that are not part of the pins. They are created to provide guidance for routing layers. It also corresponds to the `OBS` (obstruction) descriptions of a `MACRO` in LEF.

Cell

A cell refers to the geometric and electrical description of a primitive in the library. It can represent a logical gate, a complex logic gate, or other macro circuits designed to be used as a standard library primitive.

Congestion

Congestion is an indication of routability of the design. A congestion map can be generated at the global routing stage. The severity of congestion is based on the number of tracks needed by the global router and how many are available in each subarea of routing (grid).

Ggrid

The ggrid defines how the routing area of the chip is divided into small subareas at the global routing stage. It corresponds to the `GCELLGRID` definition in DEF.

Instance

The instance refers to a placed objects in general, such as a cell or a block. It generally refers to the components described in the `COMPONENTS` section in DEF.

Interconnect

Interconnect refers the collection of structures that propagate signals between the pins of cell instances. Interconnect should not include the structures that occur as part of the fixed layout of a cell.

Library

Library refers to a collection of circuit functions, implemented in a particular integrated circuit technology, which a circuit designer or synthesis application can select in order to implement or analyze a design.

Net

A net is two or more pins that need to be connected. A net includes both the geometric and logical descriptions. It generally refers to the nets described in the `NETS` section in a DEF file.

Pin

The pin object refers to a port or terminals of a cell or a block. It also corresponds to the `PIN` descriptions of a `MACRO` in LEF.

Row

A row refers to a physical area to which a group of standard cells with the same height can be placed. It corresponds to the `ROW` definition in DEF.

Special Net

A special net is a net that is used for a specific purpose, such as a power or a ground net. It generally corresponds to a `SPECIALNETS` description in DEF. The regular signal routing does not route special nets. Special nets are usually treated as prerouting in regular routing.

Track

The track defines the routing grid for detailed routing. It corresponds to the `TRACK` definition in DEF. The grid size (track spacing) is the pitch size of each layer, which is defined by `PITCH` value in LEF.

View

A view is a version of the design. It can be a placed design, a detailed routed design, or a global routed design, and so on. A design can have a single design name with multiple views.

Violation

A violation refers to design rule and LVS violations after routing. If violations exist after routing, you can run detailed routing again and perform the search-and-repair process to correct them.

Wire

A wire is a physical description of a wire segment of a net. It includes the physical location, layer, and width of the wire segment.

2

Getting Started

Supported Operating Systems

The NanoRoute® router runs on AIX, HP-UX, Redhat Linux, and Solaris operating systems. For more information, see the Encounter® README file.

License Requirements

You need a license for one of the following products to use the router:

- NanoRoute Ultra nanometer ultra router to GDSII system (FE 150)
- SoC Encounter XL hierarchical RTL-to-GDSII physical implementation system (FE 200 GPS)
- Nano Encounter® flat netlist-to-GDSII implementation system (FE 300)
- SoC Encounter™ L demand-based savings (DBS) system (FE 300 DBS)
- SoC Encounter GXL advanced hierarchical RTL-to-GDSII physical implementation system (FE 850)
- Virtuoso® Digital Implementation Option (FE 3001)

You do not need an additional license to run NanoRoute signal integrity options. You do need additional licenses for multi-threading and Superthreading. For information, see "Getting Started," in the *Encounter User Guide*.

For more information on starting the router, see page 24.

Installation

For information on setting router environment variables, running the license daemon, and installing the software, see the *Cadence Installation Guide*, the *Cadence License Manager*, and the README file included on your CD.

Log File

The log file, `nanoroute.log`, records all messages during the run process. If you have problems using the router, save the log file for reference when you contact Cadence.

You can use the log file to generate a NanoRoute command script. To generate a script, issue one of the following commands at the UNIX prompt:

```
egrep ^pdi nanoroute.log
egrep ^pdi nanoroute.log > run.tcl
```

Supported File Formats

The router supports the following input and output file formats.

Input Files

Note: You must import the library files before you import the design files.

- Library files, including the following:
  - Library Exchange Format (LEF) file (version 5.3 or higher), containing library and technology information. You can import more than one LEF file for a design.

    The router uses the first definition it reads for pins and obstructions defined for a macro. This is consistent with the behavior of Encounter™ and Silicon Ensemble® software also.
  - Extended capacitance table
  - Synopsys Liberty (LIB), containing library timing, power, and other device information
- Design files, including the following:
  - Design Exchange Format (DEF) (versions 5.3 or higher), containing design netlist, placement, and routing control information such as tracks, global routing grid, power routing, and so on.

    Note: Avoid mixing `ROUTED` and `FIXED` status of prewires in your DEF file. The mixed status of prewires might cause problems for the router during wire extraction.
  - Synopsys Design Constraints (SDC) (version 1.3 or higher), containing design-specific timing constraints
- Tcl-based NanoRoute command script
- Externally generated timing graph, to drive timing-driven routing

Note: The router can read gzipped LEF and DEF files.

Output Files

- DEF, containing the routed design
- Standard Delay Format (SDF) file, containing postroute path-delay information
- NanoRoute log file, containing routing statistics, status, and run-time information
- Capacitance report, which shows the capacitance values for all nets
- Wirelength report, which shows the wirelength and via count for all nets
- Shielding statistics report, which shows statistics for the shielded signal wires in the design.

Note: The router can export gzipped LEF and DEF files.

Customizing the Display

To change the colors and patterns that the router uses to display objects, use *Layer – Layer Configuration* in the GUI or the pdi `set_attribute -layer` command.

For information, see "pdi set_attribute" on page 110.

Starting the NanoRoute Router

You can run the router in a graphical or non-graphical environment, and in batch or interactive mode. The command syntax is:

```
nanoroute
     [-accel_any_lic]
     [-batch]
     [-help]
     [-init init_fileName]
     [-lic {nanou | nanoe | soce | socegps | socegxl}]
     [-log log_fileName]
     [-version]
```

`-accel_any_lic`

Searches for available NanoRoute Ultra, Nano Encounter, SoC Encounter, SoC-GPS, and Route Accelerator licenses to use for multi-threading. After completion, the router returns the checked-out licenses.

If you do not specify this parameter, you must have available Route Accelerator licenses for each additional thread.

`-batch` Runs the router in text (nongraphic) mode.

`-help` Displays the router online command help.

`-init` *`init_fileName`*

Runs the router in text interactive mode with a specified initialization script.

`-lic {nanou | nanoe | soce | socegps | socegxl}`

Specifies the license to check out. The router automatically searches for a license in the following order: NanoRoute (standalone), NanoRoute Ultra, Nano Encounter, SoC Encounter. You can limit the license search by specifying one of the following:

`nanou`	Searches for a NanoRoute (standalone) or NanoRoute Ultra license only.
`nanoe`	Searches for a Nano Encounter license only.
`soce`	Searches for a SoC Encounter license only.
`socegps`	Searches for a SoC Encounter Global Physical Synthesis (GPS) license only.
`socegxl`	Searches for a SoC Encounter GXL license only.

`-log` *`log_fileName`*

Specifies the NanoRoute log file. If you do not specify a filename, the router saves the log file as `nanoroute.log`.

`-version` Displays the router version number.

Running the Router in a Graphical Environment

Enter the following command:

```
nanoroute
```

Important

The NanoRoute GUI cannot be brought up on the Solaris CDE environment when the `DTHOME` environment variable is set.

Running the Router in Batch Mode

Enter the following command:

```
nanoroute -batch < run.tcl
```

Note: To stop the router in batch mode, use the UNIX command `kill -9 process_identification_number`

Running Multi-Threading

To enable multi-threading, you need additional licenses. For more information see "Running Multiple-CPU Processing" in the *Encounter User Guide*.

Running Superthreading

Superthreading combines multi-threading and distributed routing to further increase throughput during detailed routing. For more information, see "Running Multiple-CPU Processing" in the *Encounter User Guide*.

Running the Router With an Initialization Script

To run the router with an initialization script, and then run additional commands, enter the following command:

```
nanoroute -init init_fileName
```

The router starts and runs the script. After the script executes, you can enter additional commands.

Using UNIX Commands

Prefix UNIX commands with `exec` to ensure the correct execution of the command. To read back results from external commands, redirect the command to a local file and read the data back with Tcl.

For example,

```
exec mv .nano_eco_diode.list run1ECOdiode.list
exec myScript.pl file1 < file2 > file3
```

Viewing the Router Version Number

To view the version number of the router, enter the following at the UNIX prompt:

```
nanoroute -version
```

The router prints the version number to the screen and exits.

For example,

```
NanoRoute Ultra Version 2.1.s044 NR021204-2055/MT (database version 2.30) (compiled
on 12/04/2002)
```

Command Syntax

The router text commands are based on Tcl and follow Tcl syntax. The following example provides the syntax:

```
pdi command_name [-option [argument]] [command_argument ...]
```

`pdi`	Abbreviation for Programming Design Interface. You must type `pdi` before every text command.
`command_name`	Specifies the command.
`-option`	Specifies how the commands should run. A hyphen (`-`) precedes an option name.
`argument`	Specifies an argument to an option. If the argument is a list, you must enclose the list in double quotation marks (`""`) or curly braces (`{}`) and separate the list items with spaces.
`command_argument`	Specifies an argument to a command.

Important

Options are persistent throughout a session; that is, once set, they retain the current value during the router run unless you reset the value during that session. You can reset the value of an option to its default value by using `default` as the argument.

Case Sensitivity

Unless otherwise specified, command names, option names, and arguments are case-sensitive.

Multiple-Line Commands

You can split a long command onto more than one line using a backslash (\).

```
pdi cal_noise -report_file noise_timingsi.rpt \
    -error_file noise_timingsi.err
```

Online Help

After you start the router, you can list commands by typing the following command:

```
pdi help
```

The router displays a list of valid commands.

You can get help on a specific command by doing one of the following:

- Type the command name using the `-help` option. For example,

  ```
  pdi import_lib -help
  ```

- Type `pdi help` and then the command name. For example,

  ```
  pdi help import_lib
  ```

The router displays the command and arguments, plus a brief description.

```
#help for command pdi import_lib:
#pdi import_lib [ -lef | -tlib | -pef ] file_name
##Description: read library and technology file
##Arguments: 1 - 10
```

3

Using the NanoRoute Router

Overview

This chapter describes how to perform basic and advanced tasks using the NanoRoute® router in both graphical and batch mode.

For all the tasks, the basic procedure is as follows:

1. Read libraries (LEF).
2. Read placed design (DEF).
3. Perform global routing.
4. Perform detailed routing.
5. Verify design rules.
6. Output routed design (DEF).

Both sections describe how to use the router to route a design using LEF and DEF files as input. In the first section, you use the software in a graphical environment; in the second section, you use it in batch mode.

Using the Router in a Graphical Environment

Files Required

- LEF files containing descriptions of all cells and blocks used in the design and technology information
- DEF file containing the placed design

Before Starting

Make sure the software and the license file are installed.

Starting NanoRoute

➤ At the UNIX prompt, type `nanoroute` and press `Return`.

NanoRoute starts in graphical mode.

Reading the Library

1. From the menu, choose *Library – Import – LEF...*

   A dialog window appears.

2. Select the LEF file you are using from the *Files* list, then click *OK* or press `Return`.

   NanoRoute imports the LEF file that contains the library and technology information into the system's memory.

Reading the Placed Design

1. From the menu, choose *Design – Import – DEF...*

   A dialog window appears.

2. In the *Files* list, select the DEF file you are using, then click *OK* or press `Return`.

   NanoRoute imports the DEF file that contains the design netlist and placement into the database.

Performing Global Routing

➤ From the menu, choose *Route – Global Route*.

NanoRoute starts global routing. The log window where you started NanoRoute displays messages that tell you the global routing status. Global routing is finished when the *Complete Global Routing* message is displayed, followed by a summary of the routing layer and length information, the run time, and memory used.

Performing Detailed Routing

➤ From the pull-down menu, select *Route – Detail Route – Design*.

NanoRoute starts detailed routing. You can route the entire design, an area of the design, a selected net, or a selected net in an area of the design.

The log window where you started NanoRoute displays messages that tell you the detailed routing status. detailed routing is finished when the *Complete Detail Routing* message is displayed, followed by a summary of the routing layer and length information, number of violations, the run time, and memory used.

Verifying the Design

➤ From the menu, choose *Verify – All*.

NanoRoute verifies all design rules specified in the LEF files.

Alternatively, you can verify DRC rules only, LVS only, or process antenna rules only by choosing *Verify – DRC*, *Verify – LVS*, or *Verify – Process Antenna*.

Outputting Files

1. From the menu, choose *Design – Export – Full DEF...*

   A dialog window appears.

2. In the *Files* field, enter the name of the routed DEF file, and click *OK* or press `Return`.

   The routed DEF file that NanoRoute outputs contains all the design information, including placement and routing.

   Alternatively, you can output a DEF file that contains net routing information only by choosing *Design – Export – DEF (Routing Only)...*

Using the Router in Batch Mode

Files Required

- LEF files containing descriptions of all cells and blocks used in the design and technology information
- DEF file containing the placed design

Before Starting

Make sure the software and the license file are installed.

Writing the Command Script

➤ Prepare the following Tcl-based NanoRoute command script file, and save it as `sample.tcl`:

```
# Read library and technology file sample.lef
pdi import_lib -lef file_name.lef
# Read placed design file_name.def
pdi import_design -def file_name.def
# Perform global routing
pdi global_route
# Perform detail routing on the entire design
pdi detail_route
# Verify all design rules
pdi verify
# Output the entire design to DEF
pdi export_design -def -full sample_routed.def
# Exit NanoRoute Ultra
pdi exit
```

Executing the Script

➤ In the UNIX window, type the following and press `Return`:

```
nanoroute -batch < sample.tcl
```

NanoRoute executes the command script and exits. Running status is displayed during execution.

4

Command Reference

File Commands

Execute Script... (pdi exe_file)

`pdi exe_file` *`file_name`*

Runs a Tcl script file that contains NanoRoute® commands.

Argument

`file_name`	Specifies the Tcl script to run.

Example

```
pdi exe_file example.tcl
```

Runs the script file named `example.tcl`.

Save All (pdi save_lib)

`pdi save_lib [`*`file_name`*`]`

Saves the library and technology data onto disk. The data must have been previously imported with `pdi import_lib`. If you use the *`file_name`* argument to give a new name to the library, the command creates a new directory in the database using the new name. When you execute the command, the data is saved to the new directory.

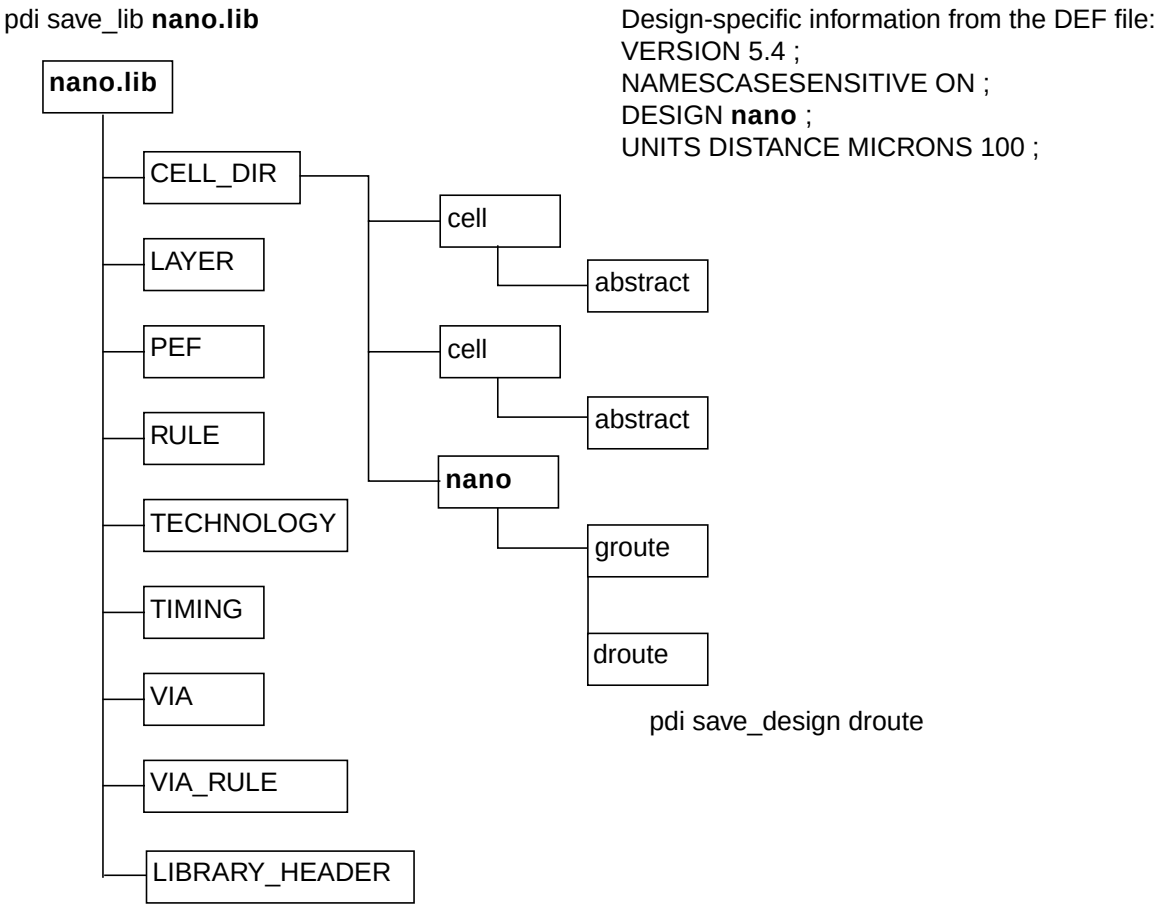
- To save only the current design or view, use the GUI command, *Save*.
- To save the current design or view with a new name, use the GUI command, use *Save As*.

Argument

`file_name`	Specifies the library. If you do not specify a name, NanoRoute uses `LIBRARY.lib` for the library name. If the design has been read by the `pdi import_design` command, the library name is the name specified in the DEF file in the `DESIGN` section with extension `.lib`.

The following figure shows the binary directory structure. The `CELL_DIR` directory contains all cell abstract definitions, including the current design. The other directories contain the technology information.

All designs saved in `CELL_DIR` use this technology information.



Example

```
pdi save_lib nano.lib
```

Related Commands

`pdi import_design`

`pdi import_lib`

`pdi load_lib`

`pdi load_design`

`pdi save_design`

Clear All (pdi clear_all)

```
pdi clear_all
```

Clears and removes all data in memory.

Note: This command does not remove designs or libraries that have been saved by the `pdi save_design` and `pdi save_lib` commands.

Argument

None

Example

```
pdi clear_all
```

Related Commands

`pdi clear_design`

`pdi import_lib`

`pdi import_design`

`pdi load_design`

`pdi save_lib`

`pdi save_design`

Exit (pdi exit)

```
pdi exit
```

Exits NanoRoute.

Save your design prior to executing the Exit command—if you have not saved your design, Exit does not ask you whether you want to save it.

Argument

None

Example

```
pdi exit
```

Related Commands

None

Library Commands

Import (pdi import_lib)

```
pdi import_lib [-lef | -tlib | -pef] file_name
```

Reads library, technology, and process data in LEF or Synopsys Liberty (`.lib`) format, and extraction data. If your design has more than one LEF or `.lib` file, you must invoke this command separately for each file.

Arguments

`-lef`	Reads the Library Exchange Format (LEF) file you specify. NanoRoute can import gzipped LEF files.
`-tlib`	Reads the Synopsys Liberty (LIB) file you specify.
`-pef`	Reads the extraction file you specify. Use the Encounter® `generateCapTbl` command to generate an extended capacitance table. For information, see `generateCapTbl` in the “RC Extraction Commands” chapter in the *Encounter Text Command Reference*.
`file_name`	Specifies the file to read.

Example

```
pdi import_lib -lef test.lef
pdi import_lib -tlib max.lib
pdi import_lib -tlib min.lib
pdi import_lib -pef abc.capTbl
pdi import_design -def placed.def
```

Imports a LEF file, minimum and maximum timing libraries, an extraction file, and a DEF file.

```
pdi import_lib -lef test.lef
pdi import_lib -lef test2.lef
```

Imports two LEF files.

```
pdi import_lib -lef lib.lef_mod.gz
```

Imports a LEF file in GZIP format.

Related Commands

`pdi export_design`

`pdi import_design`

Load (pdi load_lib)

```
pdi load_lib [file_name]
```

Loads the library and technology data from disk. The library and technology data must have been previously imported with `pdi import_lib` and saved onto the disk with `pdi save_lib`.

Saved databases are not compatible with all releases of the software.

Argument

file_name	Specifies a library name. If you do not specify a library, the software uses the default library, `LIBRARY.lib`.

Example

```
pdi load_lib abc_routed_lib
```

Loads the file called `abc_routed_lib`. NanoRoute displays the following as the file is loaded:

```
#Start load_lib on Fri Nov  12 15:20:40 2002
#
#    loading LIBRARY_HEADER ...
#    loading TECHNOLOGY ...
#    loading 14 LAYERS ...
#    loading 11 VIAS ...
#    loading 2 RULES ...
#    loading TIMING ...
#Number of layers = 14
#Number of routing layers = 6
#Number of vias = 11
#Number of rules = 2
#Number of cell = 0
```

Related Commands

`pdi import_design`

`pdi import_lib`

`pdi load_design`

`pdi save_design`

```
pdi save_lib
```

Save (pdi save_lib)

```
pdi save_lib
```

Saves the library and technology data onto disk.

For more information, see “Save All (pdi save_lib)” on page 38.

Save As (pdi save_lib)

`pdi save_lib` *`file_name`*

Renames and saves the library.

When you use the *`file_name`* argument to give a new name to the library, the command creates a new directory in the database using the new name. When you execute the command, the data is saved to the new directory.

For more information, see "Save All (pdi save_lib)" on page 38.

Design Commands

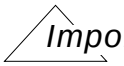
Import (pdi import_design)

```
pdi import_design [-def | -eco | -sdc] file_name
```

Reads placed DEF or SDC files.

If NanoRoute reads a DEF ECO file, it compares the file to the existing design database and updates the database. It removes the routing information from all changed nets.

This command ignores properties, property definitions, placement blockages, groups, and regions in the DEF file.



Important

Import DEF files before you import SDC files. SDC files are not incremental. When you import SDC files, they are saved with the design database.

Arguments

`-def`	Imports a DEF file. DEF files must include basic netlist and component placement. They can also include floorplan and special net routing information, via definitions, and design preroutes. NanoRoute can import gzipped DEF files. NanoRoute can import multiple DEF files. Make sure to include the blockage information in the main DEF file, instead of in a separate DEF file. If the files contain conflicting information, NanoRoute stores the information from the last file it reads.
`-eco`	Imports a DEF ECO file.
`-sdc`	Imports a Synopsys Design Constraints (SDC) file. For information on NanoRoute support for SDC syntax, see "Timing Constraint Compatibility" on page 215.
`file_name`	Specifies the file to import.

Example

To run ECO routing, complete the following steps:

1. Load a routed design into NanoRoute. For example,

   ```
   pdi load_design mydesign final
   ```

   Note: Do not use a new design name.

2. Import the new design information. For example,

   ```
   pdi import_design -eco new_design.def
   ```

3. Import the new constraints. For example,

   ```
   pdi import_design -sdc new_constraints.sdc
   ```

4. Reroute the design. For example,

   ```
   pdi global_detail_route
   ```

NanoRoute reads in the files. It compares the net names and pin instance names in the old and new databases, and copies all unchanged nets and pin instances from the old database to the new database. If it finds new placed cells, it honors the placement information in the DEF ECO file.

Related Commands

`pdi import_lib`

`pdi load_design`

`pdi load_lib`

`pdi save_design`

Load (pdi load_design)

`pdi load_design` *`cell_name view_name`*

Loads the design and constraints from disk. The design must have been previously imported with `pdi import_design` and saved into the disk with `pdi save_design`.

Saved databases are not compatible with all releases of the software. Read the LEF and DEF files if there is a compatibility problem.

Arguments

`cell_name`	Specifies the design name, which corresponds to the name defined in the `DESIGN` statement in the DEF file.
`view_name`	Specifies the view name of the design.

Example

```
pdi load_design mydesign init
```

Related Commands

`pdi import_design`

`pdi import_lib`

`pdi load_lib`

`pdi save_design`

Save (pdi save_design)

```
pdi save_design [view_name] [-ref my_design.ref]
```

Saves the design onto disk. The design must have been imported with `pdi import_design` or created by performing routing.

Argument

view_name	Specifies the view name of the design. If you do not specify a view name, NanoRoute uses the last view opened.
-ref *-my_design*.ref	Saves an ASCII format representation of the congestion map that the Encounter software can read.

Example

The following command saves the design to an ASCII file named `routed.ref`.

```
pdi save_design -ref routed.ref
```

The .ref file lists each global routing cell (gcell) on a separate line, in the following format:

```
gcell_1 (llx, lly) (x_tracks_used / x_tracks_total, y_tracks_used /
y_tracks_total)
gcell_2 (llx, lly) (x_tracks_used / x_tracks_total, y_tracks_used /
y_tracks_total)
```

For example,

```
GRC(2504.28, 19.68) (5/29, 10/20)
GRC(24.6, -0.08) (1/0) (1/0, 1/19)
```

Related Commands

`pdi import_design`

`pdi load_design`

`pdi clear_design`

Save As (pdi save_design)

For information about the syntax of this command, see “Save (pdi save_design)” on page 52.

Saves the design with a new name.

Clear (pdi clear_design)

```
pdi clear_design
```

Removes the current design from memory. Does not remove designs that have been saved into the database and disk by the `pdi save_design` command.

Argument

None

Example

```
pdi clear_design
```

Related Commands

`pdi clear_all`

`pdi import_design`

`pdi load_design`

Export (pdi export_design)

```
pdi export_design
     {-def [-full] [-dbunit {100 | 1000 | 200 | 2000}]
     [-wire_extension_format]
     [-no_lef_via file_name.def]
     | sdf
     | -aef -rule rule_file_name}
     file_name
```

Outputs the routed design in DEF, SDF, or AEF. The design must be loaded before it can be exported.

NanoRoute uses 3-D capacitance if a PEF or extended capacitance table file is loaded, otherwise it uses one-dimensional capacitance.

This command ignores properties, property definitions, placement blockages, groups, and regions in the DEF file.

Note: Before 3-D extraction or exporting SDF information, you must run *Report Capacitance*. For information see "Report Capacitance (pdi report_capacitance)" on page 89.

Arguments

`-aef`	Specifies the output in Astro/Apollo Extension Format (AEF). NanoRoute maps only the routing data, so it maps layers and LEF vias only; it does not map DEF vias or special routing. For more information, see "Routing Your Design With NanoRoute" in the *Encounter User Guide*.
`-dbunit`	Specifies the database unit value for the DEF file, if different from the `UNITS` statement defined in the LEF file.
`-def`	Specifies the output in DEF. Without additional arguments, outputs the DEF `NETS` section only, without the logical information. The default database unit for output DEF is 1,000. **Note:** NanoRoute can export gzipped DEF files.
`file_name`	Specifies the output filename.
`-full`	Specifies that the output file contain the floorplan, vias, special nets, components, pins, and nets. Specify `-full` after timing optimization with NanoRoute.

`-no_lef_via` *`file_name`*`.def`

Removes redundant vias from the output DEF file.

`-rule` *`rule_file_name`*

Specifies the AEF rule file. The rule file specifies the location of the Milkyway technology file. You can also include user-created rules for additional mapping.

`-sdf` Specifies the output in Standard Delay Format (SDF).

`-wire_extension_format`

Specifies the wire extension format for the exported file.

Examples

To read a rule file named `tfo.map` and output a file named `MyOutputFile`:

```
pdi export_design -aef -rule tfo.map MyOutputFile
```

Note: If you do not need any additional mapping, the only line in `tfo.map` is

```
techfile apollo.tf
```

To output a gzipped DEF file named `mydesign.def`:

```
pdi export_design -def -full -no_lef_via mydesign.def.gz
```

Related Commands

`pdi import_design`

`pdi report_capacitance`

View Commands

Zoom In (pdi zoomin)

`pdi zoomin {`*`x1 y1 | x1 y1 x2 y2`*`}`

Zooms in the viewable window.

- If you do not specify arguments, the software zooms in the current window by two times the magnification.
- If you specify x1 and y1 coordinates, the software zooms in to the point defined by the coordinates.
- If you specify all four coordinates, the software zooms in the area defined by the coordinates by two times the current magnification.

Tip

`pdi zoomin` *`x1 y1 x2 y2`* is equivalent to `pdi pan` *`x1 y1 x2 y2`*

Arguments

`x1 y1`	Specifies the x and y coordinates of the lower left corner of the zoom window or a point. The coordinates are in microns.
`x1 y1 x2 y2`	Specifies all four coordinates of the zoom window. The coordinates are in microns.

Bindkey

- Press z to zoom in by two times.
- Press the right mouse button at *`x1 y1`* and drag to *`x2 y2`*.

Example

```
pdi zoomin 0.0 0.0 500 300
```

Zooms in the window bounded by 0.0, 0.0 and 500, 300.

Related Commands

`pdi fit`

`pdi pan`

```
pdi redraw

pdi zoomout
```

Zoom Out (pdi zoomout)

```
pdi zoomout
```

Zooms out from the viewable window by two times.

Arguments

None

Bindkey

Z

Example

```
pdi zoomout
```

Related Commands

`pdi fit`

`pdi pan`

`pdi redraw`

`pdi zoomin`

Fit (pdi fit)

```
pdi fit
```

Fits the entire design in the window.

Arguments

None

Bindkey

```
f
```

Example

```
pdi fit
```

Related Commands

```
pdi pan
```

```
pdi redraw
```

```
pdi zoomin
```

```
pdi zoomout
```

Redraw (pdi redraw)

```
pdi redraw
```

Redraws the display window.

Arguments

None

Bindkey

```
r
```

Example

```
pdi redraw
```

Related Commands

```
pdi fit
```

```
pdi pan
```

```
pdi zoomin
```

```
pdi zoomout
```

Pan (pdi pan)

`pdi pan {-up | -down | -left | -right | x1 y1 x2 y2}`

Scrolls the viewing area by one-half screen up, down, left, or right. Alternatively, `pdi pan` can move a viewing point to a new location.

Arguments

`-up`	Moves the view up.
`-down`	Moves the view down.
`-left`	Moves the view left.
`-right`	Moves the view right.
`x1 y1 x2 y2`	Moves the view to the area defined by the coordinates.

Tip

`pdi pan` *`x1 y1 x2 y2`* is equivalent to `pdi zoomin` *`x1 y1 x2 y2`*

Bindkeys

Up, down, left, and right arrow keys pan up, down, left, and right, respectively.

Example

```
pdi pan 0.0 0.0 300.0 200.0
pdi pan -up
```

Related Commands

`pdi fit`

`pdi redraw`

`pdi zoomin`

`pdi zoomout`

Ruler Toggle (pdi display_ruler)

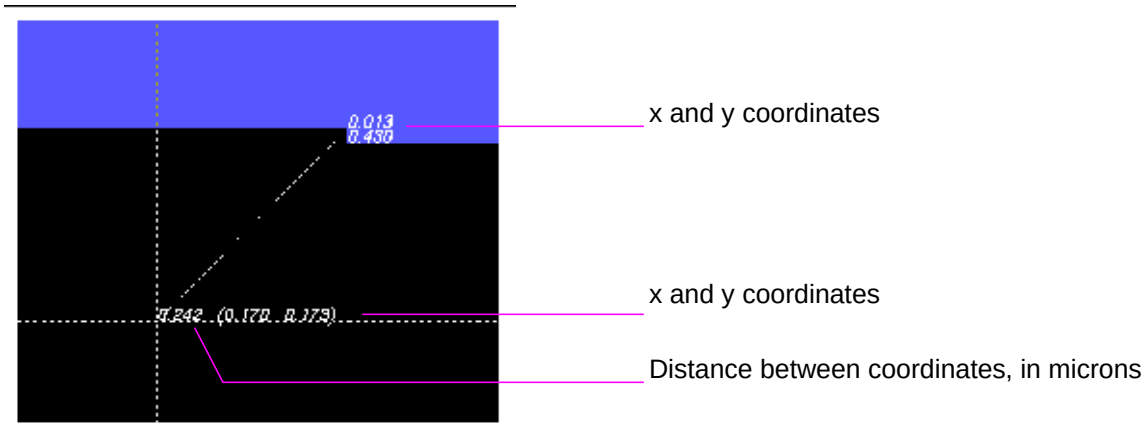
```
pdi display_ruler {-on | -off | -toggle | -stop | -add x y}
```

Creates or deletes a ruler for distance measurements. The measurements are in microns. The ruler can measure distance or size from any angle. All ruler values are removed when the ruler is turned off.

Note: When you use this command, operations dependent on the left mouse button are temporarily disabled.

Arguments

`-on`	Turns on the ruler and changes the cursor into a *cross* shape.
`-off`	Turns off the ruler and deletes existing rulers from the display.
`-toggle`	Turns the ruler on or off. If the ruler is set to `-on`, this option changes it to `-off`; if the ruler is set to `-off`, this option changes it to `-on`.
`-stop`	Stops at the end point of the ruler.
`-add` *x y*	Adds a ruler at x and y location. This arguments works only when the ruler is set to `-on`.



Bindkeys

- k turns ruler mode on or off.
- Click the left mouse button to add a ruler.

- Double-click the left mouse button to stop at the ruler end point.

Example

To use the ruler, complete the following steps:

1. Type k on the keyboard to start ruler mode.
2. Using the left mouse button, click the first point.
3. Double-click the left mouse button on another point to complete the rule.

Related Commands

`pdi fit`

`pdi pan`

`pdi redraw`

`pdi zoomin`

`pdi zoomout`

Edit Commands

Add Blockage (pdi add_blockage)

`pdi add_blockage {`*`layer_name x1 y1 x2 y2`* `|` `"`*`list_of_layer_names`*`"` *`x1 y1 x2 y2`*`}`

Creates a routing blockage on the specified layers.

Arguments

`layer_name`	Specifies the layer for the blockage. *`layer_name`* is defined in the LEF file.
`"`*`list_of_layer_names`*`"`	Specifies a list of layer names. Enclose the list in double quotation marks.
`x1 y1 x2 y2`	Specifies the x, y coordinates of the bounding box.

Examples

The following command adds a routing blockage to `m3`:

```
pdi add_blockage m3 10.5 20.5 11.0 21.0
```

The following command adds a routing blockage to `m1`, `m2`, and `m3`:

```
pdi add_blockage "m1 m2 m3" 10.5 20.5 11.0 21.0
```

Related Commands

`pdi delete_blockage`

Delete Blockage (pdi delete_blockage)

```
pdi delete_blockage {layer_name x1 y1 x2 y2} |
     "list_of_layer_names" x1 y1 x2 y2}
```

Deletes a routing blockage on the specified layers.

Arguments

-layer_name	Specifies the layer for the blockage. *layer_name* is defined in the LEF file.
"list_of_layer_names"	Specifies a list of layer names. Enclose the list in double quotation marks.
x1 y1 x2 y2	Specifies the x, y coordinates of the bounding box.

Example

The following command deletes a routing blockage from m3:

```
pdi add_blockage m3 10.5 20.5 11.0 21.0
```

The following command deletes routing blockages from m1, m2, and m3:

```
pdi add_blockage "m1 m2 m3" 10.5 20.5 11.0 21.0
```

Related Commands

`pdi add_blockage`

Delete ... (pdi delete_wire)

`pdi delete_wire {-selected | -violated | -net | -snet | -all | x1 y1 x2 y2}`

Deletes routed wires after global or detailed routing. Lets you delete selected nets, all nets with violations, all signal nets, all special nets such as power or ground nets, all nets, or all nets within a bounding box specified by lower left coordinates (*x1* and *y1*) and upper right coordinates (*x2* and *y2*).

Note: To reconnect the wires, run `pdi global_detail_route`. For information on `pdi global_detail_route`, see "Global and Detail Route (pdi global_detail_route)" on page 84.

Arguments

`-selected`	Deletes the selected nets.
`-violated`	Deletes nets with violations.
`-net`	Deletes all signal nets.
`-snet`	Deletes all special nets.
`-all`	Deletes all nets in the design.
x1 y1 x2 y2	Specifies the x and y coordinates of a bounding box from which to delete nets.

Example

```
pdi delete_wire -violated
```

Deletes nets with violations.

Related Commands

`pdi global_route`

`pdi global_detail_route`

`pdi detail_route`

`pdi select`

Delete Violation (pdi delete_violation)

```
pdi delete_violation [-drc | -process_antenna | -all]
```

Deletes specified types of violations after routing.

Arguments

`-drc`	Deletes DRC violations.
`-process_antenna`	Deletes process antenna violations.
`-all`	Deletes DRC and process antenna violations.

Default

`-all`

Example

```
pdi delete_violation -process_antenna
```

Deletes nets with process antenna violations.

Object and Layer Commands

Object or Layer (pdi toggle_viewable)

```
pdi toggle_viewable {-row | -instance | -net | -snet | -pin | -marker
     | -violation | -blockage | -ggrid | -congestion | -via | -track
     | -selected | -unselected | -layer layer_name | -masterslice
     | -metal1 | -metal2 | -metal3 | -metal4 | -metal5| -metal6
     | -metal7 | -metal8 | -metal9 | metal10}
```

Resets the viewable objects. An object that is viewable becomes unviewable when you toggle it; an object that in unviewable becomes viewable when you toggle it. You must redraw the window to change the display.

When you start NanoRoute and load a design, instances are the only objects that are viewable.

You can also use `pdi set_viewable` to make an object viewable or not viewable.

Tip

To quickly make all objects not viewable, then choose a few objects to be viewable, use the Tcl `proc` command to define a procedure. The following procedure makes all objects not viewable, then makes rows, instances, and nets viewable.

```
proc alloff {} {
  pdi set_viewable -row 0
  pdi set_viewable -instance 0
  pdi set_viewable -net 0
}
```

Arguments

`-blockage`	Controls whether blockages are viewable.
`-congestion`	Controls whether congestion is viewable.
`-ggrid`	Controls whether the global routing grid is viewable.
`-instance`	Controls whether instances are viewable
`-layer` *`layer_name`*	Controls where a specific layer is viewable. Layer names are defined in the LEF file.
`-marker`	Controls whether markers are viewable.
`-masterslice`	

	Controls whether the masterslice layers (poly and diffusion layers defined in the LEF file) are viewable.
`-metal1 ... -metal9`	
	Controls whether the specified routing layer is viewable.
`-net`	Controls whether nets are viewable.
`-pin`	Controls whether pins are viewable.
`-row`	Controls whether rows are viewable.
`-selected`	Controls whether the selected objects are viewable.
`-snet`	Controls whether special nets are viewable.
`-track`	Controls whether the routing tracks are viewable.
`-unselected`	Controls whether the unselected objects are viewable.
`-via`	Controls whether the vias are viewable.
`-violation`	Controls whether violations are viewable.

Interpreting the Congestion Map

In the congestion map, in general, blue indicates an acceptable level of congestion; white indicates an unacceptable level. You can also use the track under-congestion and over-congestion numbers to assess congestion. However, this depends largely on your design. For example, a design that is mostly uncongested might have small areas that are highly congested. You must look at the overall congestion to assess routability.

The table that follows shows the meaning of the colors and the over-congestion and under-congestion numbers when you set congestion to be viewable.

Color	Congestion Number	Explanation
Black	+2 or greater	No congestion: You have at least two tracks that are under-used.
Blue	+1	No congestion: You probably have one track that is under-used.
Green	0	No congestion: All the tracks are used.
Yellow	-1	Low congestion: You probably have one track that is over-used.

Color	Congestion Number	Explanation
Red	-2	Some congestion: You probably have two tracks that are over-used.
Magenta	-3	Moderate congestion: You probably have three tracks that are over-used.
White	-4 or less	High congestion: You probably have at least four tracks that are over-used.

Bindkeys

```
b                pdi toggle_viewable -blockage
g                pdi toggle_viewable -ggrid
i                pdi toggle_viewable -instance
m                pdi toggle_viewable -masterslice
1                pdi toggle_viewable -metal1
2                pdi toggle_viewable -metal2
3                pdi toggle_viewable -metal3
4                pdi toggle_viewable -metal4
5                pdi toggle_viewable -metal5
6                pdi toggle_viewable -metal6
7                pdi toggle_viewable -metal7
8                pdi toggle_viewable -metal8
9                pdi toggle_viewable -metal9
n                pdi toggle_viewable -net
p                pdi toggle_viewable -pin
S                pdi toggle_viewable -snet
t                pdi toggle_viewable -track
V                pdi toggle_viewable -via
v                pdi toggle_viewable -violation
```

Example

```
pdi toggle_viewable -congestion
pdi toggle_viewable -metal2
```

Related Commands

`pdi set_viewable`

Select Commands

Select (pdi select)

```
pdi select {object_name | x1 y1}
```

Selects objects based on the object name or *x1* and *y1* coordinates. In the GUI, selecting by coordinates accepts the next point as input, selecting by *object_name* prompts you to make a choice.

You can use the following wildcard characters when you select objects:

- * matches zero or more characters.
- ? matches one character.
- @CLOCK selects all nets marked + CLOCK in the DEF file.

Make sure the object is selectable. See "pdi set_selectable" on page 103 for the objects you can select.

Arguments

object_name	Specifies the object to select.
x1 y1	Specifies the x and y coordinates of the point at which an object is to be selected.

Bindkeys

- Type s, then click the object.
- Double-click the object with the mouse.

Example

```
pdi select 200.0 300.0
pdi select abc*xyz
```

Related Commands

`pdi deselect`

`pdi set_selectable`

Deselect (pdi deselect)

```
pdi deselect
```

Deselects all selected objects.

Argument

None

Bindkey

d

Example

```
pdi deselect
```

Related Commands

`pdi select`

Route Commands

Global Route (pdi global_route)

```
pdi global_route
```

Performs global routing on the entire design or selected nets. During global routing, NanoRoute does the following:

- Plans global interconnect.
- Plans detailed routing.
- Produces a congestion map.
- Runs timing optimization

NanoRoute creates the routing plans by routing the signal nets at the user-specified global cell (gcell) level. The goal of global routing is to minimize congestion and wire length, and optimize signal timing.

Tip

After global routing, check the congestion map. Areas on the map that have a lot of red, magenta, and white are very congested and indicate that your design is not routable.

For information on the congestion map, see Interpreting the Congestion Map on page 73.

Arguments

None

Example

```
pdi set_attribute -net net1 -is_selected true
pdi set_option route_selected_net_only true
pdi global_route
```

Globally routes `net1`.

Related Commands

`pdi global_detail_route`

`pdi detail_route`

```
pdi select

pdi set_attribute
```

Detail Route (pdi detail_route)

`pdi detail_route [-select] [x1 y1 x2 y2]`

Performs detailed routing and search-and-repair on the entire design, an area of the design specified by the bounding box, or selected nets. If you do not specify an argument, NanoRoute routes the entire design. If you specify both the bounding box and selected nets, it routes only selected nets within the bounding box.

During detailed routing, NanoRoute implements the nets according to design rules. It considers DRC violations, timing violations and signal integrity (or coupling capacitance) issues concurrently, and makes small adjustments to space nets, change layers, and minimizing detours to improve the overall timing slack.

In contrast to other routers that perform search and repair as postroute operations, NanoRoute performs search and repair concurrently with detailed routing. NanoRoute determines automatically when to stop search and repair.

Arguments

`-select`	Performs routing on selected nets.
x1 y1 x2 y2	Specifies the x and y coordinates of the bounding box.

Example

The following example shows the steps in the basic routing strategy with NanoRoute:

1. Global routing, including saving the design for graphical congestion analysis.:

```
pdi global_route
pdi save_design groute
```

2. Initial detailed routing (iteration 0 does not include a search-and-repair step), including saving the design for analysis:

```
pdi set_option start_iteration 0
pdi set_option droute_end_iteration 0
pdi detail_route
pdi save_design droute0
```

3. First search-and-repair iteration, including saving the design for analysis:

```
pdi set_option droute_start_iteration 1
pdi set_option droute_end_iteration 1
pdi detail_route
pdi save_design droute
```

4. Second to nineteenth search-and-repair iterations, including saving the design for analysis. The switch box grows as the iteration number grows.

```
pdi set_option droute_start_iteration 2
pdi set_option droute_end_iteration 19
pdi detail_route
pdi save_design droute19
```

5. Postroute optimization (`droute_end_iteration default`) and additional search-and-repair operations, saving the design:

```
pdi set_option droute_start_iteration 20
pdi set_option droute_end_iteration default
pdi detail_route
pdi save_design droute
```

Related Commands

`pdi global_route`

`pdi global_detail_route`

`pdi select`

`pdi set_attribute`

Global and Detail Route (pdi global_detail_route)

```
pdi global_detail_route
```

Performs both global and detailed routing with a single command.

Argument

None

Related Commands

`pdi global_route`

`pdi detail_route`

Verify Commands

This chapter describes the following commands:

Verify (pdi verify)

`pdi verify [-drc | -lvs | -process_antenna | -all] [x1 y1 x2 y2]`

Checks signal (regular) nets in a routed design or an area in a routed design for the specified types of violations.

When it flags a violation on a via, NanoRoute places the violation marker on the lower metal layer of the via, whether the actual violation is due to a problem on the lower layer or the upper layer. To repair the violation, rerun detailed routing. NanoRoute finds and repairs the violation, even when the marker was reported on the incorrect layer.

Every time you start a new routing session and run the `pdi verify` command on a routed design, ensure that you reset all options to the values that were used the last time `pdi verify` was run. Because NanoRoute resets options to their default values every time it starts a new routing session, you might get false violations if any options are not set to the same values both times you run `pdi verify`.

If you specify `pdi set_option route_selected_net_only true`, and run `pdi verify`, NanoRoute only checks the selected nets for violations. To check for violations on all nets, reset `route_selected_net_only` to `false`, and rerun `pdi verify`. For information on `route_selected_net_only`, see "route_selected_net_only" on page 164.

Important

Verify your design with the Encounter `verifyGeometry` command or another verification command. The NanoRoute `pdi verify` command does not check for the following conditions, and is not recommended in the 4.1 or 4.2 design flow:

- Double-cut via usage
- Minimum-cut violations
- Minimum enclosed area violations

For information on the Encounter verification commands, see "Verification Commands" in the *Encounter Text Command Reference*.

Arguments

`-drc`	Performs design rule checking.
`-lvs`	Performs LVS checking.
`-process_antenna`	

	Performs process antenna rule checking and generates a process antenna report named `default.antenna.rpt`.
`-all`	Performs DRC, LVS, and process antenna rule checking.
`x1 y1 x2 y2`	Specifies the x and y coordinates of the area for which you want to run verification.

Default

`-all`

Example

```
pdi verify -drc
```

Reports DRC violations.

Related Commands

`pdi global_detail_route`

`pdi detail_route`

Report Commands

This chapter describes the following commands:

Report Capacitance (pdi report_capacitance)

```
pdi report_capacitance [-one_d] [-select]
     [-c_total | -wire_c_total | -coupling_c_total | -ratio]
     [-number_top_cap value] [-report_file file_name]
     [-set_load_file file_name]
```

Extracts the parasitics after routing, reports the results, and stores the results in the database. Saves the report in the run directory. The default filename for the report is *cellname_viewname*_extract.rpt.

If you do not provide an extended capacitance table or a PEF file, NanoRoute uses 1-D extraction. If you provide an extended capacitance table or PEF file, you can use the -one_d argument to override 3-D extraction. For more information on extended capacitance files, see the *Encounter User Guide*.

Arguments

-one_d	Reports 1-D capacitance values.
-select	Reports the capacitance of selected nets only.
-c_total	Reports the capacitance sorted from large to small by total net capacitance.
-wire_c_total	Reports the capacitance sorted from large to small by wire capacitance.
-coupling_c_total	Reports the capacitance sorted from large to small by coupling capacitance.
-ratio	Reports the capacitance sorted from large to small by the ratio of coupling capacitance to total capacitance.
-number_top_cap *value*	Reports the specified number of the nets with the highest capacitance. By default, NanoRoute reports all nets or selected nets.
-report_file *file_name*	Specifies a filename for the capacitance report.
-set_load_file *file_name*	

Generates a lump capacitance file that an external timing engine can use in static timing analysis.

Default

`-c_total`

Example

To extract and report capacitance sorted by coupling capacitance from the largest to the smallest, type

```
pdi report_capacitance -coupling_c_total
```

The command generates the following report:

```
Report of Net Capacitance Sorted by total capacitance for All 8423 Nets (Unit: pf):
```

c_total	wire_c_total	coupling_c_total	ground_c_total	coupling_c_total	c_total net_name
15.723323	3.034733	1.530254	14.193069	0.097324	(ph1)
0.733395	0.697080	0.344868	0.388528	0.470234	(N0003)
0.649434	0.613119	0.334046	0.315388	0.514365	(N0004)
0.639260	0.566630	0.268566	0.370694	0.420120	(N0005)
0.625798	0.553168	0.271141	0.354657	0.433272	(N0002)
0.347411	0.347411	0.184799	0.162612	0.531933	(N0007)
0.330070	0.242604	0.130265	0.199805	0.394658	(N6801)

```
...
```

Related Commands

`pdi report_timing`

`pdi cal_noise`

Report (pdi report_design)

```
pdi report_design [-summary | -violation | -wire]
     [-selected_net_only]
```

Generates design, routing, violation, and wire length information.

Arguments

`-summary`	Creates a design summary report that includes design statistics, total wire length, and total violations. NanoRoute outputs the report to the log file.
`-violation`	Creates a detailed report of violations. The software saves the report in the run directory named *`cellname_viewname_violation`*`.rpt`.
`-wire`	Creates a report that shows the detailed wire length for each net. The report is sorted from the longest to the shortest net lengths. The software saves the report in the run directory with the name *`cellname_viewname`*`_wire.rpt`.
`-selected_net_only`	Reports only the selected nets. **Note:** `-selected_net_only` is available only when you run NanoRoute in batch mode.

Default

`-summary`

Example

```
pdi report_design -violation
```

Generates a report of routing violations for all signal nets.

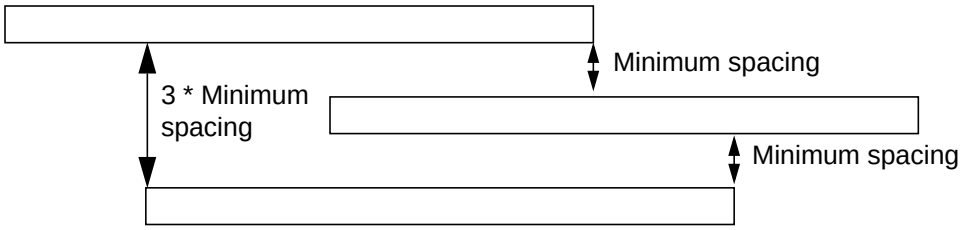
pdi report_dfm_metric

```
pdi report_dfm_metric [-extend]
```

Calculates and reports the parallel run length of adjacent wires on each routing layer.

Reports the parallel run length for each layer, including subtotals for adjacent wires within multiples of the minimum spacing rule. Reports the values in microns and as percentages of the total parallel run length for the layer. Does not include the parallel run length of routes in the non-preferred direction in the calculations or report.

Adjacent parallel wires that are separated by the minimum spacing are more likely to have short violations than wires that are separated by more than the minimum spacing. This command reports the spacing between adjacent parallel wires on each layer, so you can determine whether to separate them further and decrease the possibility of shorts. For example, if the router skips a track between wires, the spacing distance increases to approximately three times the minimum spacing and the likelihood of shorts decreases.

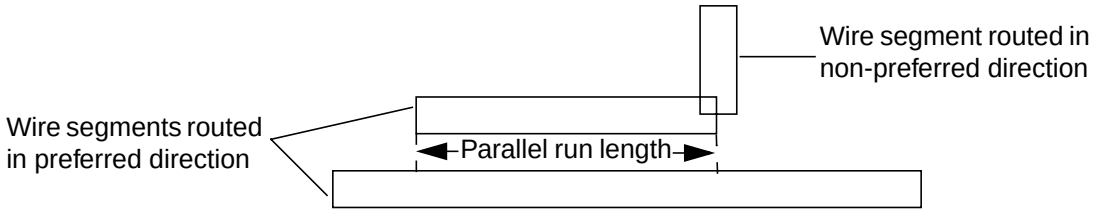


Arguments

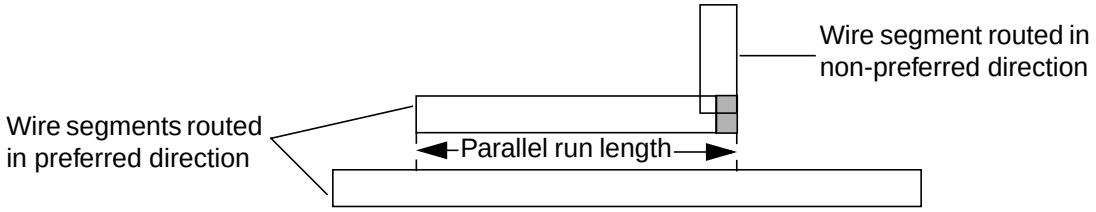
`-extend`	Includes calculations for one-half the default rule wire width, in addition to the other calculations. Use this argument to calculate parallel run length for routing layers with wire segments that have same-layer turns. When you specify this argument, `report_dfm_metric` calculates slightly longer parallel run lengths, depending on the number of same-layer turns.

Examples

The following figure shows the parallel run length considered by `pdi report_dfm_metric` without using `-extend`:



The following figure shows the parallel run length considered by `pdi report_dfm_metric -extend`:



About the DFM Metric Report

The report is in table format and includes the following information from the LEF file:

- Layer name
- Pitch

  The pitch is sometimes larger than the sum of the default width and the spacing for the layer.
- Default rule wire width for the layer
- Default rule wire-to-wire spacing for the layer

In addition to the total parallel run length for each layer, the report includes subtotals in separate columns. The subtotals are based on adjacent wires' edge-to-edge distance, as follows:

- `<=spacing`

  No spacing to one time the default spacing rule
- `<=2*spacing`

More than one time the default spacing rule to two times the default spacing rule

- `<=3*spacing`

  More than two times the default spacing rule to three times the default spacing rule

- `>3*spacing`

  More than three times the default spacing rule to infinite spacing

Examples

The following examples show reports for the same design with and without using the `-extend` argument. This design uses 0.13µm libraries.

Without using `-extend`***:***

```
#Start report_dfm_metric on Tue Jul 27 16:50:42 2004
#
#           DFM Metric (Categorized by edge-to-edge distance in multiples of min spacing)
#-------------------------------------------------------------------------------------------------------
#   Layer  Pitch Width Spacing    <=spacing     <=2*spacing     <=3*spacing      >3*spacing     Total
#-------------------------------------------------------------------------------------------------------
#   METAL1   820   320  360        0 (  0%)      540 (2.0%)        0 (  0%)    26315 ( 98%)     26856
#   METAL2   920   400  420       99 (  0%)   178729 ( 29%)       54 (  0%)   433128 ( 71%)    612011
#   METAL3   820   400  420   683656 ( 54%)        0 (  0%)   330543 ( 26%)   255784 ( 20%)   1269983
#   METAL4   920   400  420        0 (  0%)   282949 ( 36%)        0 (  0%)   496509 ( 64%)    779458
#   METAL5   820   400  420   208275 ( 36%)        0 (  0%)   197182 ( 34%)   179429 ( 31%)    584885
#   METAL6   920   400  420        1 (  0%)   114918 ( 29%)        0 (  0%)   275564 ( 71%)    390482
#-------------------------------------------------------------------------------------------------------
#   Total                     892031 ( 24%)   577136 ( 16%)   527779 ( 14%)  1666729 ( 45%)   3663675
#-------------------------------------------------------------------------------------------------------
#Cpu time = 00:00:42
#Elapsed time = 00:00:42
#Increased memory = 36.60 (Mb)
#Total memory = 127.54 (Mb)
#Peak memory = 127.54 (Mb)
#Number of warnings = 0
#Total number of warnings = 24
#Number of fails = 0
#Total number of fails = 0
#Complete report_dfm_metric on Tue Jul 27 16:51:25 2004
```

With `-extend`***:***

```
#Start report_dfm_metric on Tue Jul 27 16:51:25 2004 #
#           DFM Metric (Categorized by edge-to-edge distance in multiples of min spacing)
#-------------------------------------------------------------------------------------------------------
#   Layer  Pitch Width Spacing    <=spacing     <=2*spacing     <=3*spacing      >3*spacing     Total
#-------------------------------------------------------------------------------------------------------
#   METAL1   820   320  360        0 (  0%)      604 (2.0%)        0 (  0%)    29065 ( 98%)     29669
#   METAL2   920   400  420      240 (  0%)   200832 ( 31%)      108 (  0%)   454200 ( 69%)    655380
#   METAL3   820   400  420   724516 ( 55%)       36 (  0%)   334780 ( 26%)   251385 ( 19%)   1310718
#   METAL4   920   400  420        7 (  0%)   295140 ( 37%)        1 (  0%)   497863 ( 63%)    793011
#   METAL5   820   400  420   213828 ( 36%)        3 (  0%)   198156 ( 34%)   179305 ( 30%)    591291
#   METAL6   920   400  420        2 (  0%)   116499 ( 30%)        0 (  0%)   276015 ( 70%)    392517
#-------------------------------------------------------------------------------------------------------
#   Total                     938594 ( 25%)   613113 ( 16%)   533045 ( 14%)  1687833 ( 45%)   3772586
#-------------------------------------------------------------------------------------------------------
#The wire lengths in the above table were measured with wire segments were extended in both end.
#Cpu time = 00:00:43
#Elapsed time = 00:00:43
#Increased memory = 0.00 (Mb)
#Total memory = 127.54 (Mb)
#Peak memory = 127.54 (Mb)
#Number of warnings = 0
#Total number of warnings = 24
#Number of fails = 0
#Total number of fails = 0
#Complete report_dfm_metric on Tue Jul 27 16:52:08 2004 #
```

Report DRV (pdi report_drv)

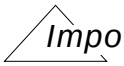
```
pdi report_drv file_name
```

Reports maximum capacitance and maximum transition violations.

The report of all maximum capacitance and maximum transition violations includes the following:

- The number of nets with maximum capacitance and maximum transition constraints
- The number of violating nets and the specific violation on each net

Note: Load maximum capacitance and maximum transition constraints with the `pdi import_design` command. For information on `pdi import_design`, see "Import (pdi import_design)" on page 49.



Important

You must run `pdi report_capacitance` and `pdi report_timing` prior to executing this command. For information, see "Report Capacitance (pdi report_capacitance)" on page 89 and "Report Timing (pdi report_timing)" on page 100.

Arguments

file_name	Specifies the report filename.

Examples

To report a design's maximum capacitance and maximum transition violations in a file named `drv.rpt`, type

```
pdi report_timing drv.rpt
```

This command generates the following report:

```
Max Capacitance Violation Report:
  7127 nets have max_cap constraints.
  10 nets have max_cap violations.

net     total_cap     max_cap      pin_cap  total_cap/max_cap  num_pins
-------------------------------------------------------------------------
N1752
         0.114311     0.076465     0.008234         1.49         2
```

```
N1754
        0.105417     0.076465     0.008234         1.38         2
N1750
        0.144498     0.107444     0.021837         1.34         3
N0515
        0.069977     0.052786     0.008234         1.33         2
N4296
        0.108450     0.082615     0.017472         1.31         2
N1583
        0.092518     0.082615     0.008234         1.12         2
N3580
        0.168851     0.156453     0.036162         1.08         4
N0729
        0.089065     0.082615     0.016847         1.08         3
N5724
        0.072137     0.069955     0.009968         1.03         2
N7021
        0.076888     0.076465     0.022356         1.01         3

Max Transition Violation Report:
  0 nets have max_tran constraints.
  0 nets have max_tran violations.
```

Related Commands

`pdi report_capacitance`

`pdi report_timing`

Report Noise (pdi cal_noise)

```
pdi cal_noise [-select | -violate | -all] [-report_file file_name]
     [-error_file file_name] [-post_fixing_file file_name]
```

Analyzes crosstalk noise for each receiver pin of a victim net from aggressive capacitive coupled nets, and generates a noise report file and noise error report file. The noise calculation must be done after detailed routing.

Arguments

`-select`	Calculates and reports noise for selected nets only.
`-violate`	Reports only nets that have noise errors.
`-all`	Calculates and reports noise for all nets.
`-report_file` *`file_name`*	Specifies the noise report file. The default filename is *`view_name_design_name`*`_noise.rpt`.
`-error_file` *`file_name`*	Specifies the noise error file. The default filename is *`view_name_design_name`*`_noise.err`.
`-post_fixing_file` *`file_name`*	Generates a Tcl script for postroute signal integrity fixing. The default name for the script is *`view_name_design_name`*`_noise.tcl.`

Default

`-violate`

Example

To create a report file named `mynoise.rpt` for selected nets with noise violations, type

```
pdi cal_noise -select -report_file mynoise.rpt
```

This command generates the following error report:

```
Report of Crosstalk Noise Errors
```

```
Total number of nets: 8423; Number of violated nets: 8

Victim net N1752 noise information:
peak_noise_ratio  noise_margin  length(um)  c_total(p)  coupling_c_total(p) state
0.262391          0.150000      693.000000  0.113623    0.070717            f


peak_noise_ratio coupling_cap(p) transition  driver_pin            aggressor_net
0.170070         0.042263        0.310967 I3519(std12_nand2_1,abstract)y  (N1750)
0.092321         0.017058        0.176538 I1682(std12_inv_2,abstract)y    (N6988)
-                0.002789        -            -(-)-                       (N6049)
-                0.001675        -            -(-)-                       (N4418)
-                0.001464        -            -(-)-                       (N6801)
-                0.001328        -            -(-)-                       (N7140)
-                0.000948        -            -(-)-                       (N1993)
-                0.003191        -            -(-)-                    other_nets

...
```

The command also generates the following noise report:

```
Report of Crosstalk Noise for All Violated Nets

peak_noise_ratio  length(um)  c_total(p)  coupling_c_total(p)  state   net_name
0.262391          693.000000  0.113623    0.070717             f       (N1752)
0.241095          416.640000  0.073056    0.051606             r       (N8133)
0.213141          848.400000  0.144016    0.074521             r       (N1750)
0.186461          467.040000  0.121164    0.060064             r       (N6988)
0.186164          441.840000  0.089771    0.047532             r       (N0809)
0.185230          304.080000  0.063320    0.040112             f       (N0613)
0.174399          600.600000  0.111157    0.072838             f       (N0703)
0.155785          723.240000  0.135707    0.060802             f       (N5156)
```

Related Commands

`pdi import_design -sdc`

`pdi import_lib -lib`

`pdi report_capacitance`

Report Timing (pdi report_timing)

```
pdi report_timing [-nworst n]
     file_name
```

Generates reports from the timing analyzer used for detailed routing. The timing analyzer is compatible with Synopsys SDC format, and recognizes false paths, multicycle paths, case statements, and operating conditions (K-factor). This command reports setup and hold violations.

To use information from extraction, you must run `pdi report_capacitance` prior to executing this command. For information, see "Report Capacitance (pdi report_capacitance)" on page 89.

Tip

To run a quick timing analysis on the input data, run this command before global routing. This is equivalent to doing a zero-wireload timing analysis in Encounter™ or Cadence® Physically Knowledgeable Synthesis.

Arguments

`-nworst n`	Reports the worst violations. For example, to see the 20 worst violations, specify `-nworst 20`.
`file_name`	Specifies the report filename.

Related Commands

`pdi cal_noise`

`pdi import_design -sdc`

`pdi import_lib -lib`

`pdi report_capacitance`

`pdi report_drv`

Help Commands

Help (pdi help)

`pdi help [`*`command_name`*`]`

Provides command syntax information about the requested command. If you do not type any arguments, returns the list of valid `pdi` commands.

Argument

`command_name`	Specifies the command for which you want help.

Example

To see help for the `delete_violation` command, type

```
pdi help delete_violation
```

NanoRoute displays the following information:

```
#pdi delete_violation (0-1) arguments
```

pdi set_selectable

```
pdi set_selectable {-row | -instance | -net | -snet | -pin | -violation}
```

Sets objects to be selectable. You can select only one object type at a time.

Arguments

`-row`	Makes rows selectable.
`-instance`	Makes instances selectable.
`-net`	Makes nets selectable.
`-snet`	Makes special nets selectable.
`-pin`	Makes pins selectable.
`-violation`	Makes violations selectable.

Example

To make nets selectable, type the following:

```
pdi set_selectable -net
```

Related Commands

`pdi select`

pdi set_viewable

```
pdi set_viewable {[-row] [-instance] [-net] [-snet] [-pin] [-violation]
     [-blockage] [-ggrid] [-track] [-congestion] [-via] [-layer layer_name]
     [-marker] [-masterslice] [-metal1] [-metal2] [-metal3] [-metal4]
     [-metal5] [-metal6] [-metal7] [-metal8] [-metal9] [-metal10]}
     {0 | 1 | true | false}
     [selected | unselected]
```

Sets objects to be viewable. You can specify more than one type of object to be viewable or not viewable at a time.

You can also use `pdi toggle` viewable to make an object viewable or not viewable.

Tip

To quickly make all objects not viewable, then choose a few objects to be viewable, use the Tcl `proc` command to define a procedure. The following procedure makes all objects not viewable, then makes rows, instances, and nets viewable.

```
proc alloff {} {
  pdi set_viewable -row 0
  pdi set_viewable -instance 0
  pdi set_viewable -net 0

}
```

Arguments

`-blockage`	Makes blockages viewable or not viewable.
`-congestion`	Makes the congestion map viewable or not viewable.
`-ggrid`	Makes global routing grids viewable or not viewable.
`-instance`	Makes instances viewable or not viewable.
`-layer` *`layer_name`*	Makes a layer viewable. *`layer_name`* is defined in the LEF file.
`-marker`	Makes markers viewable or not viewable.
`-masterslice`	Makes masterslice layers (poly and diffusion layers that are defined in LEF) viewable or not viewable.
`-metal1`	Makes the first routing layer viewable or not viewable.
`-metal2`	Makes the second routing layer viewable or not viewable.

`-metal3`	Makes the third routing layer viewable or not viewable.
`-metal4`	Makes the fourth routing layer viewable or not viewable.
`-metal5`	Makes the fifth routing layer viewable or not viewable.
`-metal6`	Makes the sixth routing layer viewable or not viewable.
`-metal7`	Makes the seventh routing layer viewable or not viewable.
`-metal8`	Makes the eighth routing layer viewable or not viewable.
`-metal9`	Makes the ninth routing layer viewable or not viewable.
`-metal10`	Makes the tenth routing layer viewable or not viewable.
`-net`	Makes signal nets viewable or not viewable.
`-pin`	Makes pins viewable or not viewable.
`-row`	Makes rows viewable or not viewable.
`-selected`	Sets selected objects only viewable or not viewable.
`-snet`	Makes special nets viewable or not viewable.
`-track`	Makes tracks viewable or not viewable.
`-unselected`	Makes unselected objects viewable or not viewable.
`-via`	Makes vias viewable or not viewable.
`-violation`	Makes violations viewable or not viewable.
`0`	Specifies that the selected objects are not viewable.
`1`	Specifies that the selected objects are viewable.
`true`	Specifies that the selected objects are viewable.
`false`	Specifies that the selected objects are not viewable.

Example

To make the congestion map viewable, type:

```
pdi set_viewable -congestion true
```

Note: For information on the congestion map, see Interpreting the Congestion Map on page 73.

To make *metal2* viewable, type

```
pdi set_viewable -metal2 true
```

To make all selected objects viewable, type

```
pdi set_viewable -selected true
```

Bindkeys

For a list of bindkeys you can use, see “Quick Reference for Bindkeys” on page 128.

Related Commands

```
pdi toggle_viewable
```

Attributes and Options

pdi get_attribute

```
pdi get_attribute [-net net_name | -cell cell_name | -pin pin_name
     |-layer layer_name {-purpose object}{-color color}
     {-pattern fill_pattern}]
```

Shows the attributes attached to a database object. Apply attributes with `pdi set_attribute`. For information, see "pdi set_attribute" on page 110.

Arguments

`-cell` *`cell_name`*	Specifies that you are getting the attributes for a cell and the name of the cell.
`-net` *`net_name`*	Specifies that you are getting the attributes for a net and the name of the net.
`-layer` *`layer_name`* `{-purpose` *`object`*`} {-color` *`color`*`} {-pattern` *`fill_pattern`*`}`	Specifies that you are getting the attributes for a layer. You can get attributes for the layer purpose, color, and pattern.
`-pin` *`pin_name`*	Specifies that you are getting the attributes for a pin and the name of the pin.

Example

```
pdi get_attribute -net net1
```

Displays the attributes attached to `net1`.

Related Commands

`pdi set_attribute`

pdi get_option

```
pdi get_option [option_name]
```

Lists the value of a NanoRoute option for this run. You can use the * or ? wildcards for the option names. If you do not supply an argument, NanoRoute lists all option settings.

For more information on NanoRoute options, see "Option Reference" on page 131.

Argument

option_name	Specifies the name of the option to list.

Example

To see the values of all the options that start with `pdi get_option droute_`, type

```
pdi get_option droute_*
```

NanoRoute displays the following:

```
pdi get_option droute_*
#droute_fix_antenna    true (bool, default setting)
#droute_end_iteration    (integer, dynamic default setting)
#droute_start_iteration    (integer, dynamic default setting)
#droute_auto_stop    true (bool, default setting)
#droute_use_min_spacing_for_blockage    true (bool, default setting)
#droute_search_and_repair    true (bool, default setting)
{droute_fix_antenna true} {droute_end_iteration} {droute_start_iteration}
{droute_auto_stop true} {droute_use_min_spacing_for_blockage true}
{droute_search_and_repair true}
```

For each option, NanoRoute displays the following:

- Name
- Current value
- Type (boolean, string, float, or integer)
- How current value was set (user, default, dynamic)

Related Commands

`pdi set_option`

pdi set_attribute

```
pdi set_attribute
     {-net net_name {[-weight {n | default}] |
        [-shield_net special_net_name] |
        [-pattern {default | steiner | trunk}] |
        [-top_preferred_routing_layer {layer | default}] |
        [-bottom_preferred_routing_layer {layer | default}] |
        [-preferred_extra_space {pitch | default}] |
        [-preferred_routing_layer_effort {low | medium | high}
        [-avoid_detour {true | false}] |
        [-is_selected {true | false}] |
        [-skip_routing {true | false}] |
        [-skip_antenna_fix {true | false default}] |
        [-non_default_rule rule_name] |
        [-si_post_route_fix {true | false}]}
        [-max_cap max_cap_value] |
        [-usage clock]
     -pin pin_name {[-rise_noise_margin float_value] |
        [-fall_noise_margin float_value]
        [-noise_margin_relative {true | false}]}
     -cell cell_name -pin pin_name {[-rise_noise_margin float_value] |
        [-fall_noise_margin float_value]
        [-noise_margin_relative {true | false}]} |
     -layer layer_name -purpose object {-color color | -pattern fill_pattern}}
```

Attaches attributes to nets and other database objects. Attaching the attributes allows NanoRoute to route the objects following specific requirements. The attributes are persistent; that is, throughout the routing process, from global routing to optimization, NanoRoute honors the attributes.

Following are some of the ways you can use attributes during routing:

- You can specify that NanoRoute route critical nets first.

  Setting the `-weight` attribute sets the routing priority. The larger the weight, the earlier the nets are routed. Within each switch box, NanoRoute routes the nets with the highest weight first, then the next highest, and so on.

- You can improve timing by restricting certain nets to higher or lower layers.

  Setting a `-top_preferred_routing_layer` and `-bottom_preferred_routing_layer` allows the router to route certain nets using mostly the specified layers, while simultaneously routing other nets with all routing layers.

- You can specify that the router must not touch some preroutes.

  Setting the `-skip_routing` attribute to `TRUE` tells NanoRoute to ignore nets with this attribute during global and detailed routing. However, skipping routing on some nets

might cause violations that can only be repaired by rerouting the skipped nets. If this is the case, you might not be able to repair the violations on some nets while `skip_routing` is set to `TRUE`.

- You can tell the router to ignore process antenna fixing for some nets.

  Setting the `-skip_antenna_fix` attribute tells the detailed router to skip certain nets during process antenna repair. This is useful if you have already provided diodes to fix the process antenna violations.

- You can space some nets, such as clock nets, farther apart if they are very sensitive to coupling capacitance.

  Setting the `-preferred_extra_space` attribute for sensitive nets tells the router to space those nets apart from other wires. The router bases the spacing on the default spacing rule plus the extra pitch defined by this attribute. The extra space is not a hard rule, however, so some nets might be routed closer than the value specified by the attribute.

- You can specify a value for maximum capacitance and set an option to optimize routing to correct capacitance violations on nets whose capacitance exceeds the value.

  Setting the `-max_cap` attribute for a net and setting the option `optimize_fix_max_cap` ensures no net drives more capacitance than the specified value after optimization.

- You can customize color and pattern settings for objects on specified layers.

  Setting the `-layer` attribute to a specified object and layer, and color or patterns, customizes the display. Put the custom settings into a TCL file and source the file in NanoRoute.

  Note: You can also customize color and patterns for objects using *Layer – Layer Configuration* in the GUI.

Arguments

`-net` *`net_name`*

Sets attributes on a net. Accepts the * and ? wildcards. You can also use the following keywords:

- `@CLOCK` specifies all clock nets
- `@ECO` specifies all prewires that can be modified during ECO routing
- `@PREROUTE` specifies all prewires that must not be modified during ECO routing
- `@RULE` specifies all nets governed by the nondefault rule in the LEF file

`-avoid_detour`

Routes critical nets as short as possible. When you set this attribute, NanoRoute does not allow a net to go beyond the minimum bounding box of all of its pins.

During postroute optimization, this attribute is disabled in order to resolve DRC issues in highly congested areas. Set the `-skip_routing` attribute for nets that must not detour or resolve the congestion that requires postroute optimization.

`-bottom_preferred_routing_layer` *`layer`*

Specifies the lowest routing layer. NanoRoute might still use a layer below the specified layer, if necessary.
If you do not set `-top_preferred_routing_layer`, NanoRoute uses all the available layers from the `-bottom_preferred_routing_layer` and up.
Default: Lowest available routing layer
Range: `1-15`

`-is_selected` Selects a net.
Default: `false`

`-max_cap`
: Specifies a global or per-net value for maximum capacitance during optimization so that, after optimization, no instance (or no net) drives more than the value you specify. To set a global value, use the asterisk (*) wildcard character. To set a per-net value, set this attribute for individual nets.
Default: `0.0` (no constraint)

`-non_default_rule` *`rule_name`*
: Identifies the nondefault routing rule to use with the specified net. Define nondefault rules in the LEF or DEF file.
To reset the value of this attribute to default, specify `-non_default_rule default`.

`-pattern {default | steiner | trunk}`
: Specifies the routing pattern. Select one of the following:

  `default`
  : Uses the value set in the `NETS` section of the DEF file for `+ PATTERN`.

  `steiner`
  : Minimizes net length.

  `trunk`
  : Routes the net from each pin directly to power and ground trunks.

    NanoRoute can connect multiple pins of the same net to a clock pin. To do trunk routing for a net, specify the net in the DEF `SPECIALNETS` section, using the following syntax:
    `-clockNetName` `+ PATTERN TRUNK ;`

    In the `PIN` section for the top-level instance, mark the pins `+ SPECIAL`.

`-preferred_extra_space` *`pitch`*

Gives additional pitch spacing to the specified net. This attribute sets a soft limit. Use `-preferred_extra_space 1` to specify that a net gets 1 extra pitch spacing, compared to other nets.
Default: `0`
Range: `0-3`

`-preferred_routing_layer_effort {low | medium | high}`

Determines how flexible NanoRoute is when setting layer limits for routing specified nets. Use this attribute with the `-top_preferred_routing_layer` and `-bottom_preferred_routing_layer` attributes. Specify `medium` or high to make NanoRoute less flexible (that is, to honor the specified routing layer range more strictly).
Default: `low` (more flexible)

`-shield_net` *`special_net_name`*

Specifies a special net to use as a shield. Use this attribute to shield critical nets or high-speed nets from other nets. Specify an attribute for the critical nets and assign power nets to shield them. Typically, you route shielded nets before other nets.

`-si_post_route_fix`

Fixes crosstalk violations on nets with problems. After you run noise analysis, use this attribute in combination with the option `route_with_si_post_route_fix`.

`-skip_antenna_fix`

Prevents the router from repairing violations on nets with process antenna violations. Specify `true` when nets have preassigned antenna diode cells, so the router does not need to perform antenna repair for those nets.
Default: `false`

`-skip_routing`

Prevents the router from routing or rerouting unrouted or partially routed nets.

When you specify `-skip_routing` for a net, the router treats the net as if you had marked it `+ FIXED` in the DEF file and it becomes an obstruction for other nets.

If the net is partially routed, the router does not complete the routing.
Default: `false`

`-top_preferred_routing_layer` *`layer`*

Specifies the highest routing layer for specified nets. This attribute sets a soft limit; that is, NanoRoute might still use a higher layer if necessary to complete routing. If you do not set `-bottom_preferred_routing_layer`, NanoRoute uses all the available layers up to the layer specified by `-top_preferred_routing_layer`.
Default: Highest available routing layer
Range: `1-15`

`-usage` Attaches the clock attribute to a net. Attaching the attribute from the command line has the same result as marking the net `+ USE CLOCK` in the `NETS` section of the DEF file.

`-weight` *`n`* Specifies a relative weight for routing nets. In each switch box, NanoRoute routes nets with the highest weight first, then the next highest weight, and so on. Specify a number higher than 2 to ensure that, within a switch box, a net is routed before other nets.
Default: 2

`-pin` *`pin_name`* Sets attributes on an instance pin. Accepts the * and ? wildcards.

`-fall_noise_margin` *`float_value`*

Specifies the fall noise margin for the `cal_noise` command.

`-noise_margin_relative`

Specifies whether the rise or fall noise margin is relative to the high voltage value or the absolute voltage value.

`-rise_noise_margin` *`float_value`*

Specifies the rise noise margin for the `cal_noise` command.

`-cell` *`cell_name`* `-pin` *`pin_name`*

Sets attributes on a cell master pin. Accepts the * and ? wildcards.

`-fall_noise_margin` *`float_value`*

Specifies the fall noise margin for the `cal_noise` command.

`-noise_margin_relative`

Specifies whether the rise or fall noise margin is relative to the high voltage value or the absolute voltage value.

`-rise_noise_margin` *`float_value`*

Specifies the rise noise margin for the `cal_noise` command.

`-layer` *`layer_name`*

Specifies color or fill attributes for an object on a specified layer. Select a layer and purpose, and a color or pattern.

`-color` *`color`*

Choose from the following colors: `aquamarine`, `azure`, `beige`, `bisque`, `black`, `blue`, `brown`, `burlywood`, `chartreuse`, `chocolate`, `coral`, `cornsilk`, `cyan`, `firebrick`, `gainsboro`, `gold`, `goldenrod`, `green`, `grey`, `honeydew`, `ivory`, `khaki`, `lavender`, `linen`, `magenta`, `maroon`, `moccasin`, `navy`, `orange`, `orchid`, `peru`, `pink`, `plum`, `purple`, `red`, `salmon`, `seashell`, `sienna`, `snow`, `tan`, `thistle`, `tomato`, `turquoise`, `violet`, `wheat`, `white`, `yellow`.

`-pattern` *`fill_pattern`*

Choose from the following fill patterns: `solid`, `hline`, `vline`, `backslash`, `slash`, `dots25`, `dots50`, `dots 75`, `brick`, `nw`, `ne`, `sw`, `se`, `square`, `diamond`, `x`.

`-purpose` *`object`*

Choose from the following objects: `net`, `snet`, `via`, `pin`, `blockage`, `track`, `violation`.

Examples

```
pdi set_attribute -net @RULE -is_selected true
```

Selects all nondefault rule nets.

```
pdi set_pdi set_attribute -net * -is_selected true
pdi set_attribute -net @clock -is_selected false
```

Selects all nets except clock nets.

```
pdi set_attribute -net @clock -weight 3 -is_selected false
pdi global_detail_route
```

Routes clock nets first by setting a higher net weight for clock nets than the rest of the nets in the design, then routes all nets.

```
pdi set_attribute -net @clock -is_selected true
pdi set_option route_selected_net_only true
pdi global_detail_route
pdi set_option route_selected_net_only false
pdi global_detail_route
```

Routes clock nets first by routing the clock nets, then rerunning routing on the rest of the design.

```
set netweight 3
set toplayer 6
set botlayer 5
set NETFILE [open "list_of_nets"]
foreach i [read $NETFILE] {
     pdi set_attribute -net $i -weight $netweight \
                       -top_preferred_routing_layer $toplayer \
                       -bottom_preferred_routing_layer $botlayer
}
close $NETFILE
pdi global_route
pdi detail_route
pdi save_design routeall
pdi verify
pdi save_design verify
```

Sets attributes for nets in a file, using a Tcl script. The name of the file is `list_of_nets`. NanoRoute uses the attributes as a guide to route the specified nets first because they have the `-weight` attribute of 3. It routes the specified nets mostly in layers 5 and 6 as the bottom and top layers respectively. All the routing is done simultaneously with the nets that do not have these attributes.

```
pdi set_attribute -layer M3 -purpose net -color coral -pattern slash
pdi set_attribute -layer M3 -purpose snet -color coral -pattern backslash
pdi set_attribute -layer M3 -purpose via -color coral -pattern dots25
pdi set_attribute -layer M3 -purpose pin -color coral -pattern dots50
pdi set_attribute -layer M3 -purpose blockage -color coral -pattern brick
pdi set_attribute -layer M3 -purpose track -color coral -pattern solid
pdi set_attribute -layer M3 -purpose violation -color coral -pattern solid
pdi redraw
```

Sets color and pattern attributes for objects on the *metal3* layer and redraws the design. You can put the settings into a TCL file and source it. To set colors and patterns using the GUI, select *Layer – Layer Configuration*.

Related Commands

`pdi global_route`

`pdi global_detail_route`

`pdi detail_route`

`pdi get_attribute`

pdi set_option

`pdi set_option` *`option_name value`*

Sets run-time options for NanoRoute commands. The options are persistent; that is, during each session of NanoRoute they keep the same value throughout the routing process, unless you reset the value to the another value or to the default value. Clearing the design, using `pdi_clear_design`, or `pdi clear_all` does not reset the value.

For more information on options, see "Option Reference" on page 131.

Arguments

`option_name`	Specifies the name of the option.
`value`	Specifies the value of the option. Use `default` to reset the value of the option to its default value.

Examples

The following examples show option settings for several options. In the last line, the value of `route_top_routing_layer_limit` is reset to its default value.

```
pdi set_option route_selected_net_only true
pdi set_option droute_search_and_repair false
pdi set_option route_top_routing_layer 5
pdi set_option route_top_routing_layer default
```

Related Commands

`pdi get_option`

Non-GUI Commands

pdi add_marker

```
pdi add_marker [-layer_name] [x1 y1 x2 y2] "comment"
```

Adds markers that allow you to view reports from external verification tools within NanoRoute.

Argument

-layer_name	Specifies the layer for the marker. *layer_name* is defined in the LEF file.
x1 y1 x2 y2	Specifies the coordinates for the marker.
comment	Specifies a comment about the marker.

Example

```
pdi add_marker 10.5 20.5 11.0 21.0 "m4 spacing violation"
```

Related Commands

`pdi delete_marker`

pdi *command* -help

```
pdi command -help
```

Displays online help for the specified command.

Argument

command	Specifies the name of a command.

Example

```
pdi find_design -help
```

Displays online help for `pdi find_design`.

NanoRoute displays the following:

```
#help for command pdi find_design:
#pdi find_design cell_name view_name
##Description find design in the database
##Argument 2 - 2
```

Note: In the last line, the first value is the minimum number of arguments you can set for the command. The second value is the maximum number of arguments you can set.

pdi delete_marker

`pdi delete_marker [-layer_name] [x1 y1 x2 y2] "comment"`

Deletes a marker.

Argument

`-layer_name`	Specifies the layer for the marker. `layer_name` is defined in the LEF file.
`x1 y1 x2 y2`	Specifies the coordinates for the marker.
`"comment"`	Specifies a comment about the marker.

Related Commands

`pdi add_marker`

Example

```
pdi delete_marker 10.5 20.5 11.0 21.0 "m4 spacing violation"
```

pdi find_design

```
pdi find_design cell_name view_name
```

Finds the design in the database. If two views of the designs are loaded, you can toggle between the views quickly using this command.

Arguments

cell_name	Specifies the design that has been saved in the database. *cell_name* corresponds to the design name specified by DESIGN in the DEF file.
view_name	Specifies the view name of the design. A design may have multiple views. A view is a version of the design that can be saved on the disk. For example, a design can have a placed view, a global-routed view, or a detail-routed view, and so on.

Example

To toggle between the `init` view and the `groute` view of the design `abc_design`, type the following:

```
pdi find_design abc_design init
pdi find_design abc_design groute
```

Quick Reference for Commands

The table below is an alphabetical list of NanoRoute commands, and gives a brief description of each command. For more information on a command, see the listing for the command earlier in this chapter.

Command	Description
`pdi add_blockage`	Creates a routing blockage on the specified layer.
`pdi add_marker`	Adds markers that allow you to view reports from external verification tools within NanoRoute.
`pdi cal_noise`	Analyzes crosstalk noise for each receiver pin of a victim net from aggressive capacitive coupled nets, and generates a noise report file and noise error report file.
`pdi clear_all`	Clears and removes all data in memory.
`pdi clear_design`	Removes the current design from memory.
`pdi delete_blockage`	Deletes a routing blockage on the specified layer.
`pdi delete_marker`	Deletes a marker.
`pdi delete_violation`	Deletes specified types of violations after routing.
`pdi delete_wire`	Deletes routed wires after global or detailed routing.
`pdi deselect`	Deselects all selected objects.
`pdi detail_route`	Performs detailed routing and search-and-repair on the entire design, an area of the design specified by the bounding box, or selected nets.
`pdi display_ruler`	Creates or deletes ruler for distance measurements.
`pdi exe_file`	Executes a Tcl script file that contains NanoRoute commands.
`pdi exit`	Exits NanoRoute.
`pdi export_design`	Outputs the routed design in DEF or SDF.
`pdi find_design`	Finds the design in the database.
`pdi fit`	Fits the entire design in the window.
`pdi get_attribute`	Shows the attributes attached to a database object.
`pdi get_option`	Lists the value of a NanoRoute option for the current run.

Command	Description
`pdi global_detail_route`	Performs both global and detailed routing with a single command.
`pdi global_route`	Performs global routing on the entire design.
`pdi help`	Provides command syntax information about the requested command.
`pdi import_design`	Reads placed DEF or SDC files.
`pdi import_lib`	Reads library, technology, and process data in LEF or Synopsys Liberty (`.lib`) format, and extraction data.
`pdi load_design`	Loads the design and constraints from disk.
`pdi load_lib`	Loads the library and technology data from disk.
`pdi pan`	Scrolls the viewing area by one-half screen up, down, left, or right.
`pdi redraw`	Redraws the display window.
`pdi report_capacitance`	Extracts the parasitics after routing, reports the results, and stores the results in the database.
`pdi report_design`	Generates design, routing, violation, and wire length information.
`pdi report_dfm_metric`	Calculates the parallel run length of wires on each routing layer.
`pdi report_drv`	Reports maximum capacitance and maximum transition violations.
`pdi report_timing`	Generates reports from the core timing analyzer used for detailed routing.
`pdi save_design`	Saves the design onto disk.
`pdi save_lib`	Renames and saves the library.
`pdi select`	Selects objects based on the object name or *`x1`* and *`y1`* coordinates.
`pdi set_attribute`	Attaches attributes to objects, such as nets.
`pdi set_option`	Sets run-time options for NanoRoute commands.

Command	Description
`pdi set_selectable`	Sets objects to be selectable.
`pdi set_viewable`	Sets objects to be viewable.
`pdi toggle_viewable`	Resets the viewable objects.
`pdi verify`	Checks signal (regular) nets in a routed design or a portion of a routed design for the specified types of violations.
`pdi zoomin`	Zooms in the viewable window.
`pdi zoomout`	Zooms out from the viewable window.

Quick Reference for Bindkeys

The table below lists NanoRoute bindkeys.

Bindkey	Equivalent Command
1	pdi toggle_viewable -routing_layer -metal1
2	pdi toggle_viewable -routing_layer -metal2
3	pdi toggle_viewable -routing_layer -metal3
4	pdi toggle_viewable -routing_layer -metal4
5	pdi toggle_viewable -routing_layer -metal5
6	pdi toggle_viewable -routing_layer -metal6
7	pdi toggle_viewable -routing_layer -metal7
8	pdi toggle_viewable -routing_layer -metal8
9	pdi toggle_viewable -routing_layer -metal9
b	pdi toggle_viewable -blockage
d	pdi deselect
f	pdi fit
g	pdi toggle_viewable -ggrid
i	pdi toggle_viewable -instance
k	pdi display_ruler
m	pdi toggle_viewable -masterslice
n	pdi toggle_viewable -net
p	pdi toggle_viewable -pin
r	pdi redraw
s	pdi select
S	pdi toggle_viewable -snet
t	pdi toggle_viewable -track
v	pdi toggle_viewable -violation
V	pdi toggle_viewable -via
z	pdi zoomin

Bindkey	Equivalent Command
Z	`pdi zoomout`
Press down arrow key	`pdi pan -down`
Press left arrow key	`pdi pan -left`
Double click left mouse button	`pdi select`
Press right arrow key	`pdi pan -right`
Press and drag right mouse button	`pdi zoomin`
Press up arrow key	`pdi pan -up`

5

Option Reference

General Options

db_report_wire_extraction

```
pdi set_option db_report_wire_extraction fileName
```

Reports extraction information in the specified file. This option is useful for ECOs.

db_skip_analog

```
pdi db_skip_analog {true | false | default}
```

Skips routing pins or nets marked + USE ANALOG in the DEF file. Set this option before reading in DEF files or loading the design.

Default

`false`

env_dont_use_lics_for_threadings

```
pdi set_option env_dont_use_lics_for_threadings {"[nano][nanor][nedbs][soce]
        [socegps]"}
```

Excludes licenses from use for Superthreading or multi-threading. Enclose the arguments in double quotation marks if you specify more than one type of license. Set this parameter before the first global or detailed route command.

Important

Starting in Encounter 7.1, use the Encounter common interface for Superthreading and multi-threading. For more information, see "Running Multiple-CPU Processing" in the *Encounter User Guide*. In this release, you can still use the NanoRoute Superthreading and multi-threading options, but they are overwritten by the Encounter multiple-CPU processing commands and options.

Arguments

`nano`	Nano Encounter®
`nanor`	NanoRoute® Ultra
`nedbs`	Nano Encounter DBS (Used for multi-threading only)
`soce`	SOC Encounter™
`socegps`	SOC Encounter GPS

Example

To exclude using licenses for SoC Encounter and SoC GPS for Superthreading or multi-threading (to use only available Nano Encounter, NanoRoute Ultra, and Nano Encounter DBS licenses), specify the following:

```
pdi set_option env_dont_use_lics_for_threadings "soce socegps"
```

Related Options

`pdi set_option env_number_processor`

`pdi set_option env_superthreading`

env_number_fail_limit

```
pdi set_option env_number_fail_limit {num_failures | default}
```

Specifies the maximum number of errors by the current command. NanoRoute commands continue running if the number of errors is less than the *num_failures*. If you specify `default`, NanoRoute halts if it encounters a single error.

Arguments

num_failures	Specifies the number of failures allowed per command.
`default`	Specifies a limit of 1.

Example

```
pdi set_option env_number_fail_limit 4
```

Sets the failure limit to `4`.

Related Options

`pdi set_option env_number_warning_limit`

env_number_processor

```
pdi set_option env_number_processor {num_processors | default}
```

Specifies the number of additional processors that one workstation can use for multi-threading. Generally, using more processors decreases the run time. NanoRoute uses the maximum number of available processors.

Note: When you start NanoRoute, you can specify the `-accel_any_lic` argument to search for unused NanoRoute Ultra, Nano Encounter®, SoC Encounter, SoC Encounter GPS, and Route Accelerator licenses. If you do not specify `-accel_any_lic`, NanoRoute searches only for Route Accelerator licenses.

Starting in Encounter 7.1, use the Encounter common interface for Superthreading and multi-threading. For more information, see "Running Multiple-CPU Processing" in the *Encounter User Guide*. In this release, you can still use the NanoRoute Superthreading and multi-threading options, but they are overwritten by the Encounter multiple-CPU processing commands and options.

Arguments

num_processors	Specifies the number of processors that one workstation can use.
`default`	Specifies 1 processor.

Example

```
pdi set_option env_number_processor 4
```

Specifies 4 processors. You must have three additional licenses available.

Related Options

`pdi set_option env_superthreading`

Related Commands

`pdi detail_route`

`pdi global_route`

```
pdi global_detail_route

pdi verify

pdi report_capacitance
```

env_number_warning_limit

```
pdi set_option env_number_warning_limit {num_warnings | default}
```

Specifies the maximum number of times NanoRoute outputs a specific warning to the log file. NanoRoute stops issuing the warning if the number of warnings exceeds *num_warnings*. If you specify `default`, NanoRoute stops issuing the warnings after it issues 20 warnings.

Arguments

num_warnings	Specifies the maximum number of times to output the warning.
`default`	Specifies 20 warnings.

Default

`20`

Example

```
pdi set_option env_number_warning_limit 200
```

Sets the warning limit to `200`.

Related Options

`pdi set_option env_number_fail_limit`

env_superthreading

```
pdi set_option env_superthreading
          {"RSH {machineName number_CPUs nanoroute_executable} ..." |
           "LSF {number_CPUs nanoroute_executable [bsub_options]} ..." |
           "SGE {number_CPUs nanoroute_executable [qsub_options]} ..." |
           "SSH {machineName number_CPUs nanoroute_executable} ..." |
           "<option_configuration_file"}
```

Distributes detailed routing among several machines that can each have several processors. This parameter accepts a configuration file or an option string containing RSH, LSF, SGE, or SSH as arguments. Use an option string if you plan to use variables that you want the Tcl interpreter to parse. Use a configuration file if your option string contains Tcl meta characters and you do not want the Tcl interpreter to parse it. Specify this parameter before running any routing commands.

Important

Starting in Encounter 7.1, use the Encounter common interface for Superthreading and multi-threading. For more information, see "Running Multiple-CPU Processing" in the *Encounter User Guide*. In this release, you can still use the NanoRoute Superthreading and multi-threading options, but they are overwritten by the Encounter multiple-CPU processing commands and options.

Arguments

bsub_options

Specifies the `bsub` options. For information on `bsub` options, see the LSF documentation.

`LSF`

Specifies Load Sharing Facility. Client logs are sent by LSF through email. If you do not want to see the logs, specify the `-o` option. For more information, see the LSF documentation.

machineName

Specifies a client machine.

nanoroute_executable

Specifies the path to the NanoRoute executable file.

number_CPUs

Specifies the number of CPUs to use in the specified *machineName*.

option_configuration_file

	Specifies a configuration file containing the LSF, RSH, SGE, or SSH parameters.
qsub_options	
	Specifies the qsub options. For information on qsub options, see the SGE documentation.
RSH	Specifies the rsh (remote shell) command. You must be able to use remote shell from server to client machines without a password prompt. NanoRoute creates a log directory for debugging. The name of the log file is *machine*.log. For more information see the rsh documentation.
SGE	Specifies the Sun Grid Engine. For more information, see the SGE documentation.
SSH	Specifies the Secure Shell. For more information, see the SSH documentation.

Example

- The following command runs Superthreading using rsh on 2 processors on my_linux_machine and 4 processors on my_sun_machine:

```
pdi set_option env_superthreading "RSH
     {my_linux machine 2 /home/bin/linux/nanoroute}
     {my_sun_machine 4 /home/bin/solaris/nanoroute}
"
```

- The following command reads a configuration file named my_superthreading_configuration_file to run Superthreading:

```
pdi set_option env_superthreading "<my_superthreading_configuration_file"
```

  - Using RSH, the file could contain the following:

```
RSH
{my_linux_machine 2 /home/bin/linux/nanoroute}
{my_sun_machine 4 /home/bin/solaris/nanoroute}
```

  - Using LSF, the file could contain the following:

```
LSF
{2 /home/bin/linux/nanoroute -q fast_queue -m linux_machine_group
{4 /home/bin/solaris/nanoroute -q multi_cpu_queue -m sun4v_machine_group}
```

Related Options

pdi set_option env_number_processor

Related Commands

`pdi detail_route`

Global and Detailed Route Options

- route_with_via_only_for_standard_cell_pin on page 174

route_allow_power_ground_pin

```
pdi set_option route_allow_power_ground_pin {true | false | auto}
```

Allows the NanoRoute software to connect tie-high and tie-low nets to nearby straps, ring pins, or core rings.

- When `false`, connects tie nets to nearby straps, ring pins, or core rings.
- When `true`, connects tie nets to the same targets as when `false`, plus macro (power and ground) pins, cover macro pins, and standard cell followpins. I/O pad pins and pad ring pins are excluded from tie-net connections.

Default

`auto` (If the design contains straps, the behavior is the same as when `false`. If the design does not contain straps, the behavior is the same as when `true`.)

Related Options

`pdi set_option route_stripe_layer_range`

route_antenna_cell_name

```
pdi set_option route_antenna_cell_name {"cellName" | "list_of_cellNames"}
```

Specifies antenna diode cells to use during postroute optimization. The cells must have the same LEF SITE definition as the standard cells. Use this parameter with `-route_insert_antenna_diode`.

Arguments

`{"cellName" | "list_of_cellNames"}`

Specifies the cells to use. You define antenna cells in the MACRO statement of the LEF file.

Default

Core macro with type ANTENNACELL in the LEF MACRO statement

Example

```
pdi set_option route_antenna_cell_name "ANTENNA"
pdi set_option route_insert_antenna_diode true
pdi global_detail_route
```

Inserts antenna diode cells named ANTENNA and routes them during postroute optimization.

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_pin_limit`

`pdi set_option route_fix_top_layer_antenna`

`pdi set_option route_ignore_antenna_top_cell_pin`

`pdi set_option route_insert_antenna_diode`

`pdi set_option route_insert_diode_for_clock_nets`

`pdi set_option route_use_existing_antenna_cell`

route_antenna_pin_limit

`pdi set_option route_antenna_pin_limit` *`num_pins`*

Specifies the maximum number of pins on any net to consider for antenna repair.

Arguments

`num_pins`	Specifies the maximum number of pins on any net to be considered for antenna repair.

Default

`1000`

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_cell_name`

`pdi set_option route_delete_antenna_reroute`

`pdi set_option route_fix_top_layer_antenna`

`pdi set_option route_ignore_antenna_top_cell_pin`

`pdi set_option route_insert_antenna_diode`

`pdi set_option route_insert_diode_for_clock_nets`

`pdi set_option route_use_existing_antenna_cell`

route_auto_ggrid

```
pdi set_option route_auto_ggrid {true | false | default}
```

Overwrites the `GCELLGRID` defined in the DEF file with a grid that is optimized for NanoRoute. Using this option helps resolve congestion and over-the-macro violations if you see routability issues due to hot spots.

When you specify `pdi set_option route_with_eco true`, NanoRoute automatically sets the value of this option to `true`, in order adjust the global routing grid automatically and reduce the memory usage.

Default

`true`

Example

```
pdi set_option route_auto_ggrid true
pdi global_route
```

Generates a global routing cell (gcell) grid that is optimal for NanoRoute, even if a different gcell grid is defined in the DEF file. Runs global routing using the grid generated by this option.

Related Options

`pdi set_option route_with_eco`

route_bottom_routing_layer

```
pdi set_option route_bottom_routing_layer {num_routing_layer | default}
```

Specifies a lower limit for global and detailed routing. Setting this option eliminates the need to change the technology files to limit the routing layers.

If the design has pins below the layer specified by this option, and if the tracks for the in-between layers are aligned to allow stacked vias to be dropped directly on top of the pins, NanoRoute uses stacked vias. However, if the tracks are not aligned, NanoRoute adds a short piece of wire before dropping a via to go to the next layer.

To set a limit for routing specific nets, use `pdi set_attribute -net -bottom_preferred_routing_limit`. For information on `pdi set_attribute`, see "pdi set_attribute" on page 110.

Arguments

num_routing_layer	Specifies the lowest layer the global and detailed routers will use. *Range:* `1-15`
`default`	`1`

Example

```
pdi set_option route_bottom_routing_layer 2
```

Sets the bottom routing layer to *metal2*.

Related Options

`pdi set_option route_selected_net_only`

`pdi set_option route_top_routing_layer`

Related Command

`pdi deselect`

`pdi global_route`

```
pdi global_detail_route

pdi detail_route

pdi set_selectable

pdi verify
```

route_delete_antenna_reroute

```
pdi set_option route_delete_antenna_reroute {true | false}
```

Deletes and reroutes nets with process antenna violations.

This option is not enabled if the design has more than 100 DRC violations, as this might introduce more DRC violations, or if you are using LEF 5.3.

Note: If the design is congested, this option runs for a long time because it deletes and reroutes the nets.

Default

`true`

Related Options

```
pdi set_option droute_antenna_eco_list_file

pdi set_option droute_fix_antenna

pdi set_option route_antenna_cell_name

pdi set_option route_antenna_pin_limit

pdi set_option route_fix_top_layer_antenna

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_antenna_diode

pdi set_option route_insert_diode_for_clock_nets

pdi set_option route_use_existing_antenna_cell
```

route_eco_only_in_layers

`pdi set_option route_eco_only_in_layers "`*`bottom_layer_number`*`:`*`top_layer_number`*`"`

Specifies the bottom and top routing layers to use for ECO routing. NanoRoute does not make changes to the routing on layers outside the specified range.

Arguments

`bottom_layer_number`
: Specifies the lowest metal layer for ECO routing.

`top_layer_number`
: Specifies the highest metal layer for ECO routing.

Related Options

`pdi set_option route_with_eco`

Example

The following command specifies ECO routing on layers 3, 4, and 5:

```
pdi set_option route_eco_only_in_layers "3:5"
```

In this example, during the ECO routing, a `via23` can access a pin on *metal2* only if the pin completely encloses the *metal2* geometry of the via.

route_extra_via_enclosure

```
pdi set_option route_extra_via_enclosure distance
```

Specifies a greater overhang value in order to connect tie-high and tie-low nets to the center of power and ground wires.

Note: OVERHANG is defined by wire direction, not by layer. It is not possible to have different overhangs for different routing layers. You must set the overhang in both directions to the distance needed by the layer that needs the largest overhang. For more information on specifying OVERHANG value, see the "LEF Syntax" chapter in the *LEF/DEF Language Reference*.

Arguments

distance	Specifies the preferred minimum distance in microns.

Default

None

Example

```
pdi set_option route_extra_via_enclosure 0.01
```

Leaves a via overhang of 0.01 μm when possible.

route_fix_top_layer_antenna

```
pdi set_option route_fix_top_layer_antenna {true | false | default}
```

Reserves the top routing layer for repairing process antenna violations.

When this option is set to `true`, NanoRoute globally routes nets that do not have diode protection, such as nets without output pin protection, without using the top routing layer. During detailed routing, it repairs antenna violations by jumping to the highest layer. This is useful for routing feedthrough nets.

Default

`true`

Related Options

```
pdi set_option droute_antenna_eco_list_file

pdi set_option droute_fix_antenna

pdi set_option route_antenna_cell_name

pdi set_option route_antenna_pin_limit

pdi set_option route_delete_antenna_reroute

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_antenna_diode

pdi set_option route_insert_diode_for_clock_nets

pdi set_option route_use_existing_antenna_cell
```

route_ignore_antenna_top_cell_pin

```
pdi set_option route_ignore_antenna_top_cell_pin {true | false | default}
```

Ignores antenna violations on top-level I/O block pins, but repairs antenna violations elsewhere.

Default

`true`

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_cell_name`

`pdi set_option route_antenna_pin_limit`

`pdi set_option route_delete_antenna_reroute`

`pdi set_option route_fix_top_layer_antenna`

`pdi set_option route_insert_antenna_diode`

`pdi set_option route_insert_diode_for_clock_nets`

`pdi set_option route_use_existing_antenna_cell`

route_honor_power_domain

```
pdi set_option route_honor_power_domain {true | false | default}
```

Routes nets without crossing power domains. You must specify a power domain before using this parameter.

- If a net connects pins that all belong to one power domain, the software tries to route the net within that power domain. If completing the routing is not possible without breaking the rules, due to congestion for example, completes the routing and issues a warning message.
- If a net connects pins that belong to different power domains, the software tries to route the net within the involved power domains, without crossing over into other power domains that do not include any of the pins in the net. If completing the routing is not possible without breaking this rule, completes the routing and issues a warning message.

Default

`false`

route_insert_antenna_diode

```
pdi set_option route_insert_antenna_diode {true | false | default}
```

Inserts and places antenna diode cells during postroute optimization if there are placement locations available. Multiple passes with antenna diode insertion add additional diodes to the net.

NanoRoute inserts antenna cells even if filler cells are already placed. It can swap filler cells with antenna diode cells, and fill the gap automatically if a diode cell is not the same size as the filler cell it replaced.

- To specify the antenna diode cells, also use `pdi set_option route_antenna_cell_name`.

  If you do not specify an antenna cell, NanoRoute uses a core macro of type `ANTENNACELL` defined in the `MACRO` statement

- To use preplaced antenna diode cells, use `pdi set_option route_use_existing_antenna_cell`.

Default

`false`

Example

```
pdi route_re_filler_cell_list fillers
pdi set_option route_antenna_cell_name "ANTENNA"
pdi set_option route_insert_antenna_diode true
pdi global_detail_route
```

Inserts antenna diode cells named `ANTENNA` and routes them during postroute optimization.

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_cell_name`

`pdi set_option route_antenna_pin_limit`

`pdi set_option route_delete_antenna_reroute`

```
pdi set_option route_fix_top_layer_antenna

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_diode_for_clock_nets

pdi set_option route_use_existing_antenna_cell
```

route_insert_antenna_in_vertical_row

```
pdi set_option route_insert_antenna_in_vertical_row {true | false | default}
```

Allows the router to insert antenna diode cells in vertical rows.

Default

`false`

route_insert_diode_for_clock_nets

```
pdi set_option route_insert_diode_for_clock_nets {true | false | default}
```

Inserts diodes to repair process antenna violations on clock nets that are in the regular net section of the DEF file.

Default

`false`

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_cell_name`

`pdi set_option route_antenna_pin_limit`

`pdi set_option route_delete_antenna_reroute`

`pdi set_option route_fix_top_layer_antenna`

`pdi set_option route_ignore_antenna_top_cell_pin`

`pdi set_option route_insert_antenna_diode`

`pdi set_option route_use_existing_antenna_cell`

route_merge_special_wire

```
pdi set_option route_merge_special_wire {true | false | default}
```

Merges overlapping same-net special wires to create large rectangles when calculating spacing requirements. Merging might be required if the result of merging produces shapes that fall on a different design-rule width range than the original shapes.

Note: Use of this parameter is recommended only for designs that require special wire merging. Using this parameter requires additional CPU time. Contact your Cadence representative for more information.

Default

`false`

route_min_shield_via_span

`pdi set_option route_min_shield_via_span` *`float`*

Specifies the preferred minimum distance in microns between via connections on a shield wire. This parameter sets a soft limit.

Argument

`distance`	Specifies the preferred minimum distance in microns.

Default

`-1` (infinite) NanoRoute will only drop one via.

route_reverse_direction

```
pdi set_option route_reverse_direction "(x1 y1 x2 y2) (x3 y3 x4 y4) ..."
```

Reverses the preferred routing direction in the specified areas. You can specify one or more areas.

> **Note:** To route a specified net in the nonpreferred direction in the specified area, you must also use the following option:
>
> ```
> setNanoRouteMode -route_selected_net_only
> ```

Use this parameter to resolve pin access problems for macro cell pins that meet the following conditions:

- They have only one routing layer access
- They do not have via access
- They are placed on the edges of the macro, parallel with the preferred routing direction (that is, they must use wrong-way routing access)

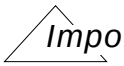
You can also use this parameter to increase the utilization of routing channels where some routing layers are blocked by power routing. For example, in a narrow routing channel, the long way usually has more traffic than the crosswise direction. If the preferred direction for power routing is the long way, the power routing might use too many routing resources and cause congestion. Reversing the preferred routing direction in this channel, that is, using routing layers without power wires for long way routing, might increase available routing resources.

route_selected_net_only

```
pdi set_option route_selected_net_only {true | false | default}
```

Specifies whether the router routes all nets at once or routes selected nets only.

To route critical nets, such as clock nets, as short as possible, select the critical nets and set this option to `true`. The global and detailed routers will route only the selected nets.



Important

If you specify `pdi set_option route_selected_net_only true`, and run the *Verify* command, NanoRoute only checks the selected nets for violations. To check for violations on all nets, you must reset `route_selected_net_only` to `false`, and rerun the *Verify* command. For more information on *Verify*, see "Verify (pdi verify)" on page 86.

Default

`false`

Example

```
pdi set_option route_selected_net_only true
pdi global_detail_route
pdi set_option route_selected_net_only false
pdi global_detail_route
```

Runs global and detailed routing for selected nets only, then reruns global and detailed routing for all nets.

```
pdi set_attribute -net @clock -weight 4 -avoid_detour true \
   -bottom_preferred_routing_layer 3
pdi set_option route_selected_net_only true
```

Gives priority to routing clock nets, and routes them without detours if possible. Tries to complete routing without using the lower two metal layers.

Related Commands

`pdi deselect`

`pdi detail_route`

`pdi global_route`

```
pdi global_detail_route

pdi set_attribute

pdi set_selectable
```

route_strictly_honor_non_default_rule

```
pdi set_option route_strictly_honor_non_default_rule {true | false}
```

Specifies that NanoRoute strictly obey nondefault spacing rules. By default, NanoRoute treats nondefault spacing rules as soft rules.

Default

`false`

route_stripe_layer_range

`pdi set_option route_stripe_layer_range "`*`bottomLayerNum`*`:`*`topLayerNum`*`"`

Specifies the range of routing layers for vias.

Arguments

`bottomLayerNum`	Specifies the lowest routing layer for vias.
`topLayerNum`	Specifies the highest routing layer for vias.

Related Options

`pdi set_option route_power_ground_pin`

route_top_routing_layer

```
pdi set_option route_top_routing_layer {num_routing_layer | default}
```

Specifies an upper limit for global and detailed routing. For example, setting the value to 4 means no wires will be routed on any layer above *metal4*: The global and detailed routers will use *metal1*, *metal2*, *metal3*, and *metal4*. Setting this option eliminates the need to change the technology files to limit the routing layers.

Use this option to set a limit for the highest routing layer for block-level routing. This reserves the remaining routing layers for top-level or chip-level routing.

Note: This option sets a hard limit. To set a soft limit for routing specific nets, use `pdi set_attribute -net -top_preferred_routing_limit`. For information on `pdi set_attribute`, see "pdi set_attribute" on page 110.

Arguments

num_routing_layer	Specifies the highest layer the global and detailed routers will use. *Range:* 1-15
default	The highest routing layer specified in the LEF file.

Example

```
pdi set_option route_top_routing_layer 4
```

Sets the top routing layer to *metal4*.

Related Option

`pdi set_option route_bottom_routing_layer`

`pdi set_option route_selected_net_only`

Related Command

`pdi deselect`

`pdi global_route`

```
pdi global_detail_route

pdi detail_route

pdi set_selectable

pdi verify
```

route_use_blockage_for_auto_ggrid

```
pdi set_option route_use_blockage_for_auto_ggrid {true | false}
```

Generates global routing cells that fit around blockages, macros, and I/O pads. The default setting is useful when libraries have blockages that cover macros from edge to edge. When blockages and macros are not aligned, specify `true` for this parameter to avoid creating violations. Also, when this parameter is `true`, NanoRoute considers blockages of I/O pads when it generates the global routing grid.

Default

`false`

Related Option

`pdi set_option route_auto_ggrid`

route_use_existing_antenna_cell

```
pdi set_option route_use_existing_antenna_cell "cell_name"
```

Uses existing diodes to repair process antenna violations. This parameter is useful during ECO routing, when you already have placed diode cells in the design.

Arguments

"*cell_name*"	Specifies a preplaced antenna cell to use during process antenna repair.

Related Options

```
pdi set_option droute_antenna_eco_list_file

pdi set_option droute_fix_antenna

pdi set_option route_antenna_cell_name

pdi set_option route_antenna_pin_limit

pdi set_option route_delete_antenna_reroute

pdi set_option route_fix_top_layer_antenna

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_antenna_diode

pdi set_option route_insert_diode_for_clock_nets
```

route_with_eco

```
pdi set_option route_with_eco {true | false | default}
```

Performs engineering change order (ECO) routing by completing partial routes with added logic, while maintaining the existing wire segments as much as possible.

ECO routing requires a complete DEF file with the original partial routes and a new ECO DEF file.

Default

`false`

Related Commands

`pdi global_detail_route`

`pdi import_design -def`

Related Options

`pdi set_option route_eco_only_in_layers`

route_with_via_in_pin

```
pdi set_option route_with_via_in_pin {true | false | default}
```

Encloses via geometries completely inside standard cell pins. Specify `true` for this option to avoid `MINSTEP` violations for standard cell pin access.

Default

`false`

Related Options

`pdi set_option route_with_via_only_for_standard_cell_pin`

route_with_via_only_for_standard_cell_pin

```
route_with_via_only_for_standard_cell_pin
     {true | false | bottomLayerNum:topLayerNum | default}
```

Allows via access only to standard cell pins. Does not allow planar access.

Default

`false`

Arguments

`true`	Always use vias for standard cell pin access on all layers.
`false`	Use vias or planar access for standard cell pins on all layers.
`bottomLayerNum``:`*`topLayerNum`*	Always use vias for pin access in the specified layer range. ■ `1:1` Always use vias for access to *metal1* standard cell pins. *metal2* and above can use vias or planar access. ■ `2:5` Always use vias for access to standard cell pins on *metal2*, *metal3*, *metal4*, and *metal5*. *metal1* and *metal6* (and above) can use vias or planar access.

Related Options

`pdi set_option route_with_via_in_pin`

Detailed Route Options

droute_antenna_eco_list_file

`pdi set_option droute_antenna_eco_list_file` *`file_name`*

Specifies the file generated during process antenna repair that contains the list of diodes that were inserted. If you insert diodes during more than one detailed routing pass, NanoRoute creates a separate file for each pass and gives it a unique name.

Default

`.nano_eco_diode.list`

Example

```
pdi set_option droute_antenna_eco_list_file nano_ant_diode.list
```

NanoRoute creates the following files:

- After the first detailed routing pass: `nano_ant_diode.list`
- After the second detailed routing pass: `nano_ant_diode.list1`
- After the third detailed routing pass: `nano_ant_diode.list2`

Related Options

```
pdi set_option droute_fix_antenna

pdi set_option route_antenna_cell_name

pdi set_option route_antenna_pin_limit

pdi set_option route_delete_antenna_reroute

pdi set_option route_fix_top_layer_antenna

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_antenna_diode

pdi set_option route_insert_diode_for_clock_nets

pdi set_option route_use_existing_antenna_cell
```

droute_auto_stop

```
pdi set_option droute_auto_stop {true | false | default}
```

Controls whether the router continues routing if there are many violations. In highly congested designs, the router stops if the number of violations is too high. Specify `false` to continue routing until there is no improvement.

If your design is very congested and this option is set to `true` or `default`, NanoRoute halts after building the initial detailed routing database.

Default

`true`

Related Options

`pdi set_option droute_search_and_repair`

Related Commands

`pdi detail_route`

droute_check_minstep_on_top_level_pin

```
pdi set_option droute_check_minstep_on_top_level_pin {true | false | default}
```

Checks for minimum step violations where a wire connects to a DEF pin shape. Set this parameter to `true` if the DEF pin shapes in the design are not the same width as the wires.

Default

`false`

droute_elapsed_time_limit

`droute_elapsed_time_limit` *`minutes`*

Specifies a run-time limit for detailed routing. The time-limit is based on the elapsed time (the wall-clock time), not the CPU time. If the router reaches the limit during initial routing iteration, it stops when it finishes the routing iteration and keeps the routing result. If it reaches the limit during search and repair or postroute optimization, it stops immediately and keeps the routing result at that time.

Default

`0` (no time limit)

droute_end_iteration

```
pdi set_option droute_end_iteration {num_iteration | default}
```

Specifies the last pass in a detailed routing step. If NanoRoute has globally routed the design, but has not made any detailed routing passes, set `droute_start_iteration` to `0` (iteration `0` is the initial detailed route) to build the database for detailed routing.

Iterations after `0` do not run routing. Instead, they run search and repair. If you delete a net and globally route the net, you must reset `droute_start_iteration` to `0`.

- `droute_start_iteration` specifies the first pass in a routing step.
- `droute_end_iteration` specifies the last pass in a routing step.

To run postroute optimization, specify the following values for `droute_start_iteration` and `droute_end_iteration`:

```
droute_start_iteration 20
droute_end_iteration default
```

Note: Setting `droute_end_iteration` to a value higher than 20 generally does not improve results.

Arguments

`num_iteration`	Specifies the last pass in a detailed routing step.
`default`	Specifies postroute optimization.

Example

```
pdi global_route
pdi save_design groute
pdi set_option droute_end_iteration 0
pdi set_option droute_end_iteration 0
pdi detail_route
pdi save_design droute0
pdi set_option droute_start_iteration 1
pdi set_option droute_end_iteration 1
pdi detail_route
pdi save_design droute1
pdi set_option droute_start_iteration 2
pdi set_option droute_end_iteration 19
pdi detail_route
pdi save_design droute19
pdi set_option droute_start_iteration 20
pdi set_option droute_end_iteration default
pdi detail_route
pdi save_design droute
```

In the example above, NanoRoute does the following:

1. Globally routes the design and saves it as `groute`.
2. Runs initial detailed routing and saves the design as `droute0`.
3. Runs search and repair and saves the design after iteration 1 and iteration 19.
4. Runs postroute optimization.
5. Saves the fully routed database as `droute`.

Related Option

`pdi set_option droute_start_iteration`

Related Command

`pdi detail_route`

droute_fix_antenna

```
pdi set_option droute_fix_antenna {true | false | default}
```

Determines whether to repair process antenna violations by antenna stapling (also called layer hopping or jumping layers).

Note: In general, NanoRoute gives the highest priority to repairing DRC violations. It repairs process antenna violations only if it can do so without creating DRC violations.

During metallization and etching, the lower-level metal layers can create process antennas, because the connections have not been completed yet. The antennas can build up charge, which can damage the gate of the transistors if the antennas are connected to the input pins. The amount of charge is directly proportional to the metal area connected to the gate, and can be diffused when the router makes the connections to the top layer.

You can also repair antenna violations by doing the following:

1. Set `droute_fix_antenna false`.
2. Finish detailed routing
3. Set `droute_fix_antenna true`.
4. Rerun detailed routing to repair the process antenna violations.

Note: The LEF antenna keywords are not the same for versions 5.3, 5.4, and 5.5. In your LEF file, do not mix keywords from the LEF 5.3, 5.4, and 5.5. If you are using LEF 5.3, NanoRoute assumes only one of `ANTENNAAREAFACTOR` and `ANTENNALENGTHFACTOR` is specified as non-zero for a given layer. If neither one is specified or both are `0`, NanoRoute does not check antenna violations for that layer. If you specify a value for only one of `ANTENNAAREAFACTOR` and `ANTENNALENGTHFACTOR`, NanoRoute assumes the value of the other to be `0`.

Default

`true`

Related Options

`pdi set_option droute_antenna_eco_list_file`

`pdi set_option droute_fix_antenna`

`pdi set_option route_antenna_cell_name`

```
pdi set_option route_antenna_pin_limit

pdi set_option route_delete_antenna_reroute

pdi set_option route_fix_top_layer_antenna

pdi set_option route_ignore_antenna_top_cell_pin

pdi set_option route_insert_antenna_diode

pdi set_option route_insert_diode_for_clock_nets

pdi set_option route_use_existing_antenna_cell
```

Related Commands

```
pdi detail_route
```

droute_min_length_for_wire_spreading

`droute_min_length_for_wire_spreading` *`length_in_microns`*

Specifies the minimum length in microns for a wire segment of a net before a jog is necessary to avoid OPC issues. Spreading wires farther apart (creating jogs in the wires) helps avoid design rule violations, signal integrity problems, and yield loss.

Default

2

Related Options

`pdi set_option droute_post_route_spread_wire`

droute_min_length_for_wire_widening

```
pdi set_option droute_min_length_for_wire_widening length_in_microns
```

Specifies the minimum length in microns for a wire segment of a net before the segment is a candidate for widening. Wires are widened by an amount defined by a nondefault rule. They are not widened unless they meeting the following criteria:

- Timing is not adversely affected
- Routing resources are available
- No overflow is created
- No design rule or process antenna violations are created

Widening wires helps avoid design rule violations, signal integrity problems, and yield loss.

Note: To enable this option, you must also include a wide wire nondefault rule in the LEF file and specify the `-droute_post_route_widen_wire_rule true` option.

Default

`1`

Related Options

```
pdi set_option droute_post_route_widen_wire
pdi set_option droute_post_route_widen_wire_rule
```

droute_min_slack_for_wire_optimization

`droute_min_slack_for_wire_optimization` *`time_in_nanoseconds`*

Specifies the minimum slack on pins associated with nets that are candidates for wire spreading or wire widening. Using this parameter reduces the timing impact of wire spreading and wire widening, and helps reduce both congestions and the yield loss due to shorts and opens.

Default

0.0

droute_minimize_via_count

```
pdi set_option droute_minimize_via_count {true | false | default}
```

Minimizes the number of vias in a fully routed design.

Set this option to `true` and `droute_end_iteration` to `default`.

Note: Do not use this option and `-droute_use_multi_cut_via` together.

Default

`false`

Related Options

`droute_use_multi_cut_via_effort`

droute_no_taper_on_output_pin

```
pdi set_option droute_no_taper_on_output_pin {true | false | default}
```

Prohibits NanoRoute from tapering at standard cells, macro cells, and block output pins.

Default

`false`

droute_post_route_spread_wire

```
pdi set_option droute_post_route_spread_wire {true | false | default}
```

Turns postroute wire spreading on (`true`) or off (`false`). Specifying `true` for this parameter might increase the overall routing run time up to 150 percent, but helps avoid design rule violations, signal integrity problems, and yield loss.

Note: Specify the minimum length for wires to spread with `-drouteMinLengthForWireSpreading`.

Default

`false`

Related Options

`pdi set_option droute_min_length_for_wire_spreading`

droute_post_route_swap_via

```
pdi set_option droute_post_route_swap_via {multiCut | singleCut | none}
```

Swaps single-cut vias for multiple-cut vias or reverses swapping on critical nets, so that multiple-cut vias are swapped for single-cut vias on those nets. Does not swap multiple-cut vias in nondefault rule routing.

A net with a pin on a critical path is considered a critical net if the pin slack, plus the margin defined by the `droute_min_slack_for_wire_optimization` option, is less than or equal to 0.

Default

`none`

Arguments

`multiCut`	Specifies the first pass in a detailed routing step.
`none`	Does not swap vias.
`singleCut`	During timing-driven routing, swaps vias on critical nets only; otherwise, might swap any multiple-cut via for a single-cut via. Swapping multiple-cut vias for single-cut vias can improve timing and relieve congestion as a postroute process.

Related Options

`pdi set_option -droute_min_slack_for_wire_optimization`

droute_post_route_widen_wire

```
pdi set_option post_route_widen_wire {widen | shrink | none}
```

Applies the rule specified by the -`droute_post_route_widen_wire_rule` parameter to widen or shrink wires.

Arguments

`none`	During timing-driven routing, widens only non-critical nets; otherwise widens as many nets as possible that are covered by the nondefault rule for widening wires.
`shrink`	During timing-driven routing, shrinks only critical nets; otherwise shrinks as many nets as possible that are covered by the nondefault rule for widening wires.
`widen`	Turns off this parameter, so the router does not widen or shrink wires.

Default

`none`

Related Options

`pdi set_option droute_post_route_widen_wire_rule`

droute_post_route_widen_wire_rule

`pdi set_option droute_post_route_widen_wire_rule` *`ruleName`*

Specifies the nondefault rule for widening wires. The rule is specified in the LEF file.

Related Options

`pdi set_option droute_post_route_widen_wire`

droute_search_and_repair

```
pdi set_option droute_search_and_repair {true | false | default}
```

Runs a search-and-repair step after the initial detailed routing.

When you want to determine quickly if the design is routable, set this option to `false`, run initial detailed routing without search and repair, and see the number of violations. This is a quick indication of whether routing can achieve a DRC-clean design.

Note: In general, NanoRoute gives the highest priority to repairing DRC violations. It repairs process antenna violations only if it can do so without creating DRC violations.

Default

`true`

Example

```
pdi set_option droute_search_and_repair false
```

Related Options

`pdi set_option droute_auto_stop`

Related Commands

`pdi detail_route`

droute_start_iteration

```
pdi set_option droute_start_iteration {num_iteration | default}
```

Specifies the first pass in a detailed routing step. Runs detailed routing in steps.

By default, during detailed routing, NanoRoute sets `droute_start_iteration 0` and `droute_end_iteration default`. During this step, NanoRoute builds the detailed routing database and runs initial detailed routing.

Iterations after `0` do not run routing. Instead, they run search and repair. If you delete a net then run globally routing, you must reset `droute_start_iteration` to `0`.

If you set `droute_start_iteration` to `20`, and `droute_end_iteration` to `default`, NanoRoute runs postroute optimization.

Arguments

num_iteration	Specifies the first pass in a detailed routing step.
`default`	`0`

Example

```
pdi global_route
pdi save_design groute
pdi set_option droute_end_iteration 0
pdi set_option droute_end_iteration 0
pdi detail_route
pdi save_design droute0
pdi set_option droute_start_iteration 1
pdi set_option droute_end_iteration 1
pdi detail_route
pdi save_design droute1
pdi set_option droute_start_iteration 2
pdi set_option droute_end_iteration 19
pdi detail_route
pdi save_design droute19
pdi set_option droute_start_iteration 20
pdi set_option droute_end_iteration default
pdi detail_route
pdi save_design droute
```

In the example above, NanoRoute does the following:

1. Globally routes the design and saves it as `groute`.
2. Runs initial detailed routing and saves the design as `droute0`.
3. Runs search and repair and saves the design after iteration 1 and iteration 19.

4. Runs postroute optimization.
5. Saves the fully routed database as `droute`.

Related Options

`pdi set_option droute_end_iteration`

Related Commands

`pdi detail_route`

droute_use_bigger_overhang_via_first

```
pdi set_option droute_use_bigger_overhang_via_first {true | false}
```

Specifies that NanoRoute choose the vias with the bigger metal overhang first during via optimization. NanoRoute chooses from fat single and fat double-cut vias. All vias must be predefined or automatically generated before via optimization.

Default

`false`

droute_use_multi_cut_via

```
pdi set_option droute_use_multi_cut_via {true | false}
```

Inserts multiple-cut vias concurrently with routing.

Default

`false`

droute_use_multi_cut_via_effort

```
pdi set_option droute_use_multi_cut_via_effort {low | medium | high}
```

Specifies the effort level toward increasing the ratio of double-cut vias to single-cut vias and minimizing the total number of vias concurrently with routing. Increasing the effort level increases the double-cut vias ratio and decreases the total number of vias in the design, which means that it also decreases the number of single-cut vias. The lower the number of single-cut vias, the better the yield will be. After routing with this parameter specified, you can add more double-cut vias or larger overhang vias by running another pass of `detailRoute` with the `droute_post_route_swap_via` parameter specified.

Arguments

`low`	Specifies normal routing, so the router uses single-cut vias only.
`medium`	Balances the need for a high double-cut via ratio with run time and congestion. Specifying this parameter increases the run time somewhat compared with the `low` parameter. The router inserts some double-cut vias, although not as many as if the high parameter were specified.
`high`	Specifies the highest yield possible for vias, as the router puts its best effort toward achieving the highest possible double-cut via ratio at the expense of run time and congestion.

Default

`low`

droute_via_on_grid_only

```
pdi set_option droute_via_on_grid_only {true | false | default}
```

Centers vias on routing grids by dropping them at track intersections.

Default

`false`

Signal Integrity Options

route_si_effort

```
pdi set_option route_si_effort {min | normal}
```

Prevents or reduces crosstalk by spreading wires, layer hopping, and other changes to the topology. When both timing-driven and signal integrity-driven routing are specified, NanoRoute repairs crosstalk violations, minimizes congestion, and resolves timing violations concurrently.

Arguments

`min`	Minimizes crosstalk without running noise analysis. Does not run postroute signal integrity repair.
`normal`	Runs postroute noise analysis and repair of violations.

Default

`min`.

Related Options

`pdi set_option route_with_si_driven`

`pdi set_option route_with_si_post_route_fix`

Related Commands

`pdi import_design`

`pdi import_lib`

Example

```
pdi set_option route_si_effort normal
pdi global_detail_route
```

Runs global and detailed routing, with signal integrity avoidance, analyzes noise, and repairs noise violations.

route_with_si_driven

```
pdi set_option route_si_driven {true | false | default}
```

Prevents or reduces crosstalk by spreading wires, layer hopping, and minimization of long parallel wires. When both timing-driven and signal integrity-driven routing are specified (SMART routing), NanoRoute repairs crosstalk violations, minimizes congestion, and resolves timing violations concurrently.

Default

`false`

Related Options

`pdi set_option route_si_effort`

`pdi set_option route_tdr_effort`

`pdi set_option route_with_timing_driven`

Related Commands

`pdi detail_route`

`pdi global_route`

`pdi global_detail_route`

route_with_si_post_route_fix

```
pdi set_option route_with_si_post_route_fix {true | false | default}
```

Changes the routing topology, including layer assignments, spacing, and location, to avoid coupling to other nets. This option can be used only with nets that have the `si_post_route_fix` attribute set to `true`. For information on setting attributes, see "pdi set_attribute" on page 110.

Default

`false`

Related Options

`pdi set_option route_with_si_driven`

`pdi set_option route_si_effort`

Related Commands

`pdi detail_route`

`pdi global_route`

`pdi global_detail_route`

`pdi set_attribute`

Timing and Timing Optimization Options

route_tdr_effort

`pdi set_option route_tdr_effort` *`integer`*

Specifies the effort for timing-driven routing. A higher value increases the effort toward meeting timing constraints and decreases the effort toward relieving congestion.

Default

`0`

Range

`0-10`

Related Options

`pdi set_option route_with_timing_driven`

`pdi set_option route_si_effort`

route_with_timing_driven

```
pdi set_option route_with_timing_driven {true | false | default}
```

Minimizes the number of timing violations during routing. During timing-driven routing NanoRoute tries to maintain routability and avoid creating timing violations. The following list includes some of the operations NanoRoute performs when it optimizes wires on critical nets: avoiding detours, reducing coupling and wire resistance, and routing to the most critical sink pin first.

The timing-driven routing flow requires the following input files:

- Externally generated timing graph if you are using the Encounter® common timing engine (CTE) or another external timing engine or a timing library in `.lib` format and timing constraints in `.sdc` format if you are using the internal NanoRoute timing engine
- Physical libraries in LEF
- Placed design in DEF
- Extended capacitance table from the Encounter software

For more information on timing-driven routing, see "Report Timing (pdi report_timing)" on page 100.

Default

`false`

Related Options

`pdi set_option timing_engine`

`pdi set_option route_tdr_effort`

Related Commands

`pdi report_timing`

`pdi detail_route`

`pdi global_route`

`pdi global_detail_route`

timing_engine

`pdi set_option timing_engine {Nano | externalTimingGraphFile}`

Specifies the timing engine or timing graph file that drives timing analysis.

Arguments

`externalTimingGraphFile`

Specifies the timing graph file.

Default

`Nano`

Related Options

`pdi set_option route_with_timing_driven`

`pdi set_option route_with_timing_opt`

`pdi set_option route_tdr_effort`

Quick Reference for Options

For more information on an option, see the listing for the option earlier in this chapter.

Option	Description
`db_report_wire_extraction`	Reports wire extraction information in the specified file.
`db_skip_analog`	Skips routing pins or nets marked + `USE ANALOG` in the DEF file.
`droute_antenna_eco_list_file`	Specifies the file generated during process antenna repair that contains the list of diodes that were inserted.
`droute_auto_stop`	Controls whether the router continues routing if there are too many violations.
`droute_check_minstep_on_top_level_pins`	Checks for minimum step violations where a wire connects to a DEF pin shape.
`droute_elapsed_time_limit`	Specifies a run-time limit for detailed routing.
`droute_end_iteration`	Specifies the last pass in a detailed routing step.
`droute_fix_antenna`	Determines whether to repair process antenna violations by antenna stapling (also called layer hopping or jumping layers).
`droute_min_length_for_wire_spreading`	Specifies the minimum length in microns for a wire segment of a net before a jog is necessary to avoid OPC issues.
`droute_min_length_for_wire_widening`	Specifies the minimum length in microns for a wire segment of a net before the segment is a candidate for widening.
`droute_min_slack_for_wire_optimization`	Specifies the minimum slack on pins associated with nets that are candidates for wire spreading or wire widening.

Option	Description
`droute_minimize_via_count`	Minimizes the number of vias in a fully routed design.
`droute_no_taper_on_output_pin`	Prohibits NanoRoute from tapering at standard cells, macro cells, and block output pins.
`droute_search_and_repair`	Runs a search-and-repair step after the initial detailed routing.
`droute_post_route_spread_wire`	Turns postroute wire spreading on (`true`) or off (`false`).
`droute_post_route_swap_via`	Swaps single-cut vias for multiple-cut vias or reverses swapping on critical nets, so that multiple-cut vias are swapped for single-cut vias on those nets.
`droute_post_route_widen_wire`	Turns postroute wire widening on (`true`) or off (`false`).
`droute_post_route_widen_wire_rule`	Specifies the nondefault rule for widening wires. The rule is specified in the LEF file.
`droute_start_iteration`	Specifies the first pass in a detailed routing step.
`droute_use_bigger_overhang_via_first`	Specifies that NanoRoute choose the vias with the bigger metal overhang first during via optimization.
`droute_use_multi_cut_via`	Inserts multiple-cut vias concurrently with routing.
`droute_use_multi_cut_via_effort`	Specifies the effort level toward increasing the ratio of double-cut vias to single-cut vias and minimizing the total number of vias concurrently with routing.
`droute_via_on_grid_only`	

Option	Description
	Centers vias on routing grids by dropping them at track intersections.
`env_dont_use_lics_for_threadings`	Excludes licenses from use for Superthreading or multi-threading.
`env_number_fail_limit`	Specifies a limit for the number of errors generated by the current command.
`env_number_processor`	Specifies the number of processors that one workstation can use during multi-threading.
`env_number_warning_limit`	Specifies the maximum number of times NanoRoute outputs a specific warning to the log file.
`env_superthreading`	Distributes detailed routing among several machines that can each have several processors.
`route_allow_power_ground_pin`	Allows NanoRoute to connect tie-high and tie-low nets to power and ground pins of standard cells.
`route_antenna_cell_name`	Routes antenna diode cells with the specified names during postroute optimization.
`route_antenna_pin_limit`	Specifies the maximum number of pins on any net to be considered for antenna repair.
`route_auto_ggrid`	Overwrites the `GCELLGRID` defined in the DEF file with a grid that is optimized for NanoRoute.
`route_bottom_routing_layer`	Specifies a lower limit for global and detailed routing.

Option	Description
`route_delete_antenna_reroute`	Deletes and reroutes nets with process antenna violations.
`route_eco_only_in_layers`	Specifies the bottom and top routing layers to use for ECO routing.
`route_extra_via_enclosure`	Specifies a greater overhang value in order to connect tie-high and tie-low nets to the center of power and ground wires.
`route_fix_top_layer_antenna`	Reserves the top routing layer for repairing process antenna violations.
`route_honor_power_domain`	Routes nets without crossing power domains.
`route_ignore_antenna_top_cell_pin`	Ignores antenna violations on top-level I/O block pins, but repairs antenna violations elsewhere.
`route_insert_antenna_diode`	Inserts and places antenna diode cells during postroute optimization if there are placement locations available.
`route_insert_antenna_in_vertical_row`	Allows the router to insert antenna diode cells in vertical rows.
`route_insert_diode_for_clock_nets`	Inserts diodes to repair process antenna violations on clock nets that are in the regular net section of the DEF file.
`route_merge_special_wire`	Merges overlapping same-net special wires to create large rectangles when calculating spacing requirements.
`route_min_shield_via_span`	Specifies the preferred minimum distance in microns between vias connections on a shield wire.

Option	Description
`route_reverse_direction`	Reverses the preferred routing direction in the specified areas. You can specify one or more areas.
`route_selected_net_only`	Specifies whether the router routes all nets at once or routes selected nets only.
`route_si_effort`	Reduces coupling concurrently with routing to decrease crosstalk.
`route_strictly_honor_non_default_rule`	Forces NanoRoute to honor nondefault rules.
`route_stripe_layer_range`	Specifies the range of routing layers for vias.
`route_tdr_effort`	Specifies the effort for timing-driven routing.
`route_top_routing_layer`	Specifies an upper limit for global and detailed routing.
`route_use_blockage_for_auto_ggrid`	Generates global routing cells that fit around blockages, macros, and I/O pads.
`route_use_existing_antenna_cell`	Uses preplaced diodes to repair process antenna violations.
`route_with_eco`	Performs engineering change order (ECO) routing by completing partial routes with added logic, while maintaining the existing wire segments as much as possible.
`route_with_si_driven`	Spreads wires to reduce coupling whenever possible without sacrificing routability.

Option	Description
`route_with_si_post_route_fix`	Changes the routing topology, including layer assignments, spacing, and location, to avoid coupling to other nets.
`route_with_timing_driven`	Minimizes the number of timing violations during routing.
`route_with_via_in_pin`	Encloses via geometries completely inside standard cell pins.
`route_with_via_only_for_standard_cell_pin`	Allows via access only to standard cell pins. Does not allow planar access.
`timing_engine`	Specifies the timing engine or timing graph file that drives timing analysis.

A

Timing Constraint Compatibility

For more information, see the following documents:

- For information on Synopsys syntax, see the appropriate Synopsys documentation.
- For information on SDC syntax that is supported by the Encounter® product family, including the Encounter common timing engine (CTE) see Appendix A, "Data Preparation," in the *Encounter User Guide*.
- For information on SDC syntax that is supported by Cadence® Physically Knowledgeable Synthesis, see the *SDC Support Guide*.

Important

The table in this appendix lists commands and arguments that are known by Cadence at the time this document is published. There might be additional commands or arguments that are not included in this appendix.

Overview

The NanoRoute internal timing engine reads timing constraints in SDC format. The constraints describe the design intent, including timing and area requirements, and are used by NanoRoute during timing analysis, and to share information with the Synopsys Design Compiler and PrimeTime tools.

If your timing constraints are in PrimeTime or Design Compiler format, you must translate them to SDC format before NanoRoute can read them.

Note: The parser can read several timing constraint commands that are not in SDC format. These commands are included in the table in the following section.

Cadence recommends that you use the CTE-based timing-driven routing flow. The information in this appendix is for backward compatibility, and describes the internal NanoRoute timing engine timing constraint support. For information on CTE timing constraint support, see the *Encounter User Guide*.

Timing Constraint Support Table

The table in this section lists timing constraint commands in alphabetical order, and whether they are supported by NanoRoute.

- Supported commands

  If no arguments are listed, then all arguments are supported.

- Unsupported commands

  No arguments are listed for unsupported commands.

Note: NanoRoute supports the use of * and ? wildcards in object names in the *`object_list`*. For example, both of the following object names are supported:

```
a/b/c/*
/inst*_??/
```

Commands	Support
`add_to_collection`	Unsupported
`all_clocks`	Supported

Commands	Support	
`all_connected`		Unsupported
`all_inputs`	Supported	
`-clock` *`clock_name`*		Unsupported
`-edge_triggered`		Unsupported
`-level_sensitive`		Unsupported
`all_outputs`	Supported	
`-clock` *`clock_name`*		Unsupported
`-edge_triggered`		Unsupported
`-level_sensitive`		Unsupported
`all_registers`		Unsupported
`create_clock`	Supported	
`-add`	Supported	
`-domain` *`domain_name`*		Unsupported
`-name` *`clock_name`*	Supported	
`-period` *`period_value`*	Supported	
`-waveform` *`edge_list`*		Unsupported
`source_objects`		
none (virtual clock)	Supported	
`pins`	Supported	
`ports`	Supported	
`create_generated_clock`	Supported	
`-add`		Unsupported
`-divide_by_factor` *`factor`*	Supported	
`-domain` *`domain_name`*		Unsupported
`-duty_cycle` *`percent`*		Unsupported
`-edge_shift` *`shift_list`*	Supported	
`-edges` *`edge_list`*	Supported	

Commands	Support
-invert	Supported
-multiply_by *factor*	Supported
-name *clock_name*	Supported
-source *master_pin*	Supported
source_objects	
pins	Supported
ports	Supported
current_design	Supported
design	Unsupported
current_instance	Unsupported
define_design_mode	Unsupported
derive_clocks	Unsupported
expr	Unsupported
filter	Unsupported
filter_collection	Unsupported
find	Supported
-flat	Unsupported
-hierarchy	Unsupported
search_pattern	Supported
object_type	
cell	Supported
clock	Supported
cluster	Unsupported
design	Unsupported
file	Unsupported
implementation	Unsupported
lib_cell	Unsupported

Commands	Support
lib_pin	Unsupported
library	Unsupported
module	Unsupported
multibit	Unsupported
net	Unsupported
operator	Unsupported
pin	Supported
port	Supported
reference	Unsupported
foreach_in_collection	Unsupported
get_cells	Supported
-exact	Unsupported
-filter *expression*	Unsupported
-hierarchical	Unsupported
-hsc *separator*	Unsupported
-nocase	Unsupported
-of_objects *objects*	Unsupported
-quiet	Unsupported
-regexp	Unsupported
patterns	Unsupported
get_clocks	Supported
-filter *expression*	Unsupported
-nocase	Unsupported
-quiet	Unsupported
-regexp	Unsupported
patterns	Unsupported
get_designs	Unsupported

Commands	Support
get_generated_clocks	Unsupported
get_lib_cells	Supported
-exact	Unsupported
-filter *expression*	Unsupported
-hsc *separator*	Unsupported
-nocase	Unsupported
-of_objects *objects*	Unsupported
-quiet	Unsupported
-regexp	Unsupported
patterns	Unsupported
get_lib_pins	Unsupported
get_libs	Supported
-exact	Unsupported
-filter *expression*	Unsupported
-hsc *separator*	Unsupported
-nocase	Unsupported
-of_objects *objects*	Unsupported
-quiet	Unsupported
-regexp	Unsupported
patterns	Unsupported
get_nets	Supported
-exact	Unsupported
-filter *expression*	Unsupported
-hsc *separator*	Unsupported
-nocase	Unsupported
-of_objects *objects*	Unsupported
-quiet	Unsupported

Commands	Support
`-regexp`	Unsupported
`patterns`	Unsupported
`get_pins`	Supported
`-exact`	Unsupported
`-filter` *`expression`*	Unsupported
`-hsc` *`separator`*	Unsupported
`-nocase`	Unsupported
`-of_objects` *`objects`*	Unsupported
`-quiet`	Unsupported
`-regexp`	Unsupported
`patterns`	Unsupported
`get_ports`	Supported
`-exact`	Unsupported
`-filter` *`expression`*	Unsupported
`-hsc` *`separator`*	Unsupported
`-nocase`	Unsupported
`-of_objects` *`objects`*	Unsupported
`-quiet`	Unsupported
`-regexp`	Unsupported
`patterns`	Unsupported
`get_unix_variable`	Unsupported
`group_path`	Unsupported
`list`	Supported
Note: You can enclose `list` arguments in curly braces (`{`) or double quotation marks (`"`).	
`remove_from_collection`	Unsupported
`set`	Unsupported
`set_capacitance`	Unsupported

Commands	Support	
`set_case_analysis`	Supported	
`set_clock_gating_check`		Unsupported
`set_clock_latency`	Supported	
`-early`	Supported	
`-fall`	Supported	
`-late`	Supported	
`-max`	Supported	
`-min`	Supported	
`-rise`	Supported	
`-source`	Supported	
`delay`	Supported	
`object_list`		
`clocks`	Supported	
`ports`		Unsupported
`pins`		Unsupported
`set_clock_skew`		Unsupported
`set_clock_transition`	Supported	
`set_clock_uncertainty`	Supported	
`-from` *`from_clock`*	Supported	
`-to` *`to_clock`*	Supported	
`-from_edge`	Supported	
`-to_edge`	Supported	
`-rise`	Supported	
`-fall`	Supported	
`-setup`	Supported	
`-hold`	Supported	
`uncertainty_value`	Supported	

Commands	Support	
object_list		
cells		Unsupported
clocks	Supported	
pins		Unsupported
ports		Unsupported
set_current_design_mode		Unsupported
set_data_check		Unsupported
set_disable_timing	Supported	
-from *from_pin_name*	Supported	
-restore		Unsupported
-to *to_pin_name*	Supported	
object_list		
cells	Supported	
lib_cells	Supported	
lib_pins	Supported	
pins	Supported	
ports	Supported	
set_dont_touch	Supported	
set_dont_touch_network		Unsupported
set_dont_use	Supported	
set_drive	Supported	
set_drive_resistance	Supported	
set_driving_cell	Supported	
-fall	Supported	
-from_pin *from_pin_name*	Supported	
-input_transition_fall *fall_time*		Unsupported

Commands	Support
`-input_transition_rise` *`rise_time`*	Unsupported
`-lib_cell` *`lib_cell_name`*	Supported
`-max`	Supported
`-min`	Supported
`-multiply` *`factor`*	Unsupported
`-no_design_rule`	Unsupported
`-pin` *`pin_name`*	Supported
`-rise`	Supported
`object_list`	
`input_ports`	Supported
`set_false_path`	Supported
`-setup`	Supported
`-hold`	Supported
`-hold`	Supported
`-rise`	Supported
`-fall`	Supported
`-from` *`from_list`*	Supported
`-to to_list`	Supported
`-through` *`through_list`*	Supported
Note: Multiple `-through` arguments are not supported.	
`-reset_path`	Unsupported
`set_fanout_load`	Unsupported
`set_hierarchy_separator`	Unsupported
`set_ideal_net`	Unsupported
`set_input_delay`	Supported
`-add_delay`	Supported
`-clock` *`clock_name`*	Supported

Commands	Support	
-clock_fall	Supported	
-fall	Supported	
-level_sensitive		Unsupported
-max	Supported	
-min	Supported	
-network_latency_included		Unsupported
-rise	Supported	
-source_latency_included		Unsupported
delay_value		
object_list		
input_ports	Supported	
internal_pins		Unsupported
set_input_transition	Supported	
-clock *clock_name*		Unsupported
-clock_fall		Unsupported
-fall	Supported	
-max	Supported	
-min	Supported	
-rise	Supported	
transition	Supported	
object_list		
input_ports	Supported	
internal_pins		Unsupported
set_load	Supported	
set_logic_dc		Unsupported
set_logic_one	Supported	
object_list		

Commands	Support
pins	Unsupported
ports	Supported
set_logic_zero	Supported
object_list	
pins	Unsupported
ports	Supported
set_max_area	Unsupported
set_max_capacitance	Supported
value	Supported
object_list	
current_design	Supported
Note: Current design only	
pins	Supported
ports	Supported
set_max_delay	Unsupported
set_max_dynamic_power	Unsupported
set_max_fanout	Unsupported
set_max_leakage_power	Unsupported
set_max_time_borrow	Unsupported
set_max_transition	Supported
value	Supported
object_list	
current_design	Supported
Note: Current design only	
pins	Supported
ports	Supported
set_min_capacitance	Unsupported

Commands	Support
`set_min_delay`	Unsupported
`set_min_porosity`	Unsupported
`set_min_pulse_width`	Unsupported
`set_min_transition`	Unsupported
`set_mode`	Unsupported
`set_multicycle_path`	Supported
`-end`	Supported
`-fall`	Supported
`-from` *`from_list`*	Supported
`-group_path` *`group_name`*	Unsupported
`-hold`	Supported
`-reset_path`	Supported
`-rise`	Supported
`-setup`	Supported
`-start`	Supported
`-through` *`through_list`*	Supported
Note: Multiple `-through` arguments are not supported.	
`-to` *`to_list`*	Supported
`path_multiplier`	Unsupported
`set_operating_conditions`	Supported
`set_output_delay`	Supported
`-clock clock_name`	Supported
`-clock_fall`	Supported
`-level_sensitive`	Unsupported
`-rise`	Supported
`-fall`	Supported
`-max`	Supported

Commands	Support	
-min	Supported	
-add_delay		Unsupported
-network_latency_included		Unsupported
-source_latency_included		Unsupported
delay_value	Supported	
object_list		
output_ports	Supported	
internal_pins		Unsupported
set_port_fanout_number		Unsupported
set_propagated_clock	Supported	
object_list		
clocks	Supported	
ports		Unsupported
pins		Unsupported
cells		Unsupported
set_resistance		Unsupported
set_scan_configuration		Unsupported
set_scan_element		Unsupported
set_switching_activity		Unsupported
set_switching_element		Unsupported
set_timing_derate		Unsupported
set_timing_ranges		Unsupported
set_wire_load		Unsupported
set_wire_load_min_block_size		Unsupported
set_wire_load_mode		Unsupported
set_wire_load_model		Unsupported
set_wire_load_selection_group		Unsupported

Commands	Support
`source`	Unsupported

Index

L

M

N

O

P

R

S

T