PrimeECO User Guide

Version S-2021.06, June 2021

SYNOPSYS®

Contents

About This User Guide

The PrimeECO tool supports automatic and manual changes to fix timing, design rule, and noise violations, and to optimize power and parametric yield. It performs multi-scenario signoff timing, placement legalization, routing, and RC extraction in a single shell environment.

This guide is for engineers who use the PrimeTime, IC Compiler II, and StarRC tools for generating and implementing engineering change orders (ECOs).

The PrimeTime Suite documentation consists of the following documents:

- *PrimeTime User Guide* – Timing analysis using the PrimeTime, PrimeTime SI, PrimeTime ADV, and PrimeTime ADVP tools.
- *PrimeECO User Guide* (this user guide) – ECO timing, power, and DRC fixing with signoff-quality timing.
- *PrimePower User Guide* – Static and dynamic full-chip power analysis.
- *PrimeShield User Guide* – Parametric timing robustness analysis, cell robustness analysis, and critical path fast Monte Carlo HSPICE simulation.

This preface includes the following sections:

- New in This Release
- Related Products, Publications, and Trademarks
- Conventions
- Customer Support

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the PrimeECO Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the PrimeECO tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

You might also want to see the documentation for the following related Synopsys products:

- *PrimeTime User Guide* – Static timing analysis and ECO tool
- *IC Compiler II User Guide* – Placement and routing tool
- *StarRC User Guide* – Parasitic extraction tool

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing =` **`true`** `| false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

https://solvnetplus.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

Feedback

1

Introduction to PrimeECO

The PrimeECO tool supports automatic and manual engineering change orders (ECOs) to fix timing, design rule, and noise violations, and to optimize power and parametric yield. It performs multi-scenario signoff timing, placement legalization, routing, and RC extraction in a single shell environment. You can complete incremental ECO iterations and signoff without leaving the PrimeECO shell.

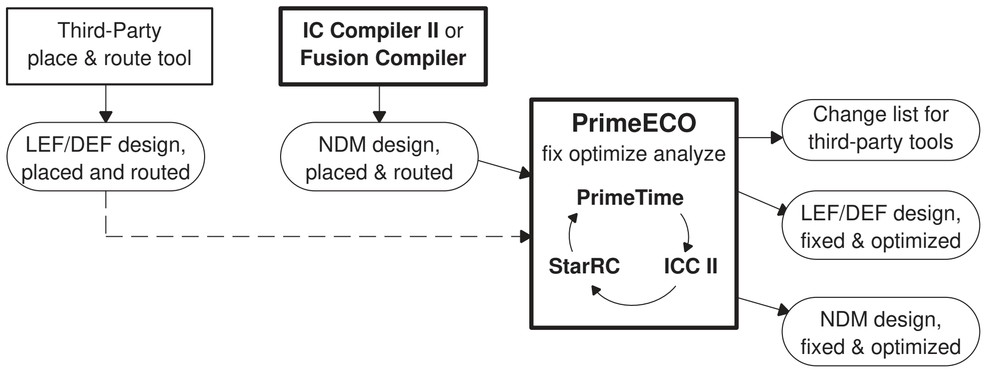
The following topics provide more information about the PrimeECO tool:

- The PrimeECO Tool
- Running the PrimeECO Tool
- PrimeECO Flow Options
- Levels of What-If Analysis

The PrimeECO Tool

The PrimeECO tool allows you to perform timing closure (extraction, STA, optimization, and physical implementation) in a single tool shell. Figure 1 shows how the PrimeECO tool works in the physical implementation flow.

Figure 1 PrimeECO in the Physical Implementation Flow



Third-Party place & route tool
IC Compiler II or Fusion Compiler
LEF/DEF design, placed and routed
NDM design, placed & routed
PrimeECO
fix optimize analyze
PrimeTime
StarRC
ICC II
Change list for third-party tools
LEF/DEF design, fixed & optimized
NDM design, fixed & optimized

Figure 2 and Figure 3 compare the discrete-tool ECO flow and the PrimeECO flow.

Figure 2 The Discrete ECO Flow

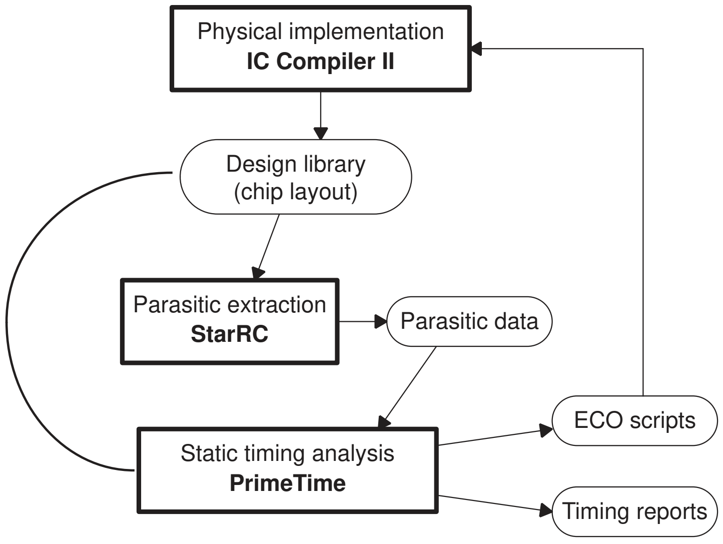
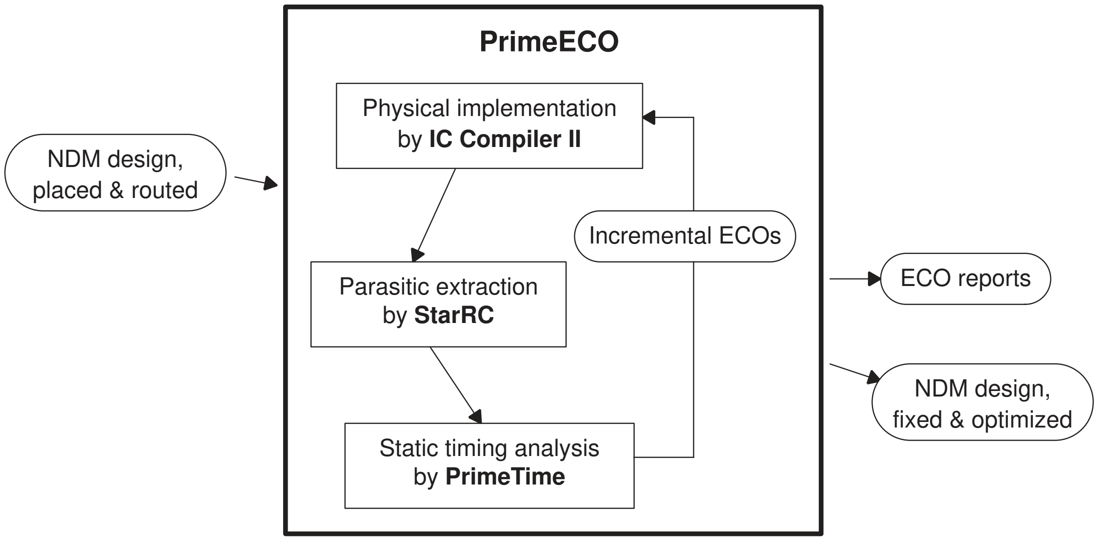


Figure 3 The PrimeECO Flow


PrimeECO
NDM design, placed & routed
Physical implementation by IC Compiler II
Incremental ECOs
Parasitic extraction by StarRC
ECO reports
NDM design, fixed & optimized
Static timing analysis by PrimeTime

The PrimeECO tool combines the timing analysis, physical implementation, and parasitic extraction ECO functions of the PrimeTime, IC Compiler II, and StarRC tools into a single integrated environment. The PrimeECO tool performs multiple ECO fixing iterations without the need for pausing between iterations or transferring data between tools, and without concern for keeping data files and libraries aligned between iterations.

The PrimeECO hybrid timing view handles an unlimited number of scenarios while using a limited number of machines, and finds optimal tradeoffs between compute resources and the required timing accuracy. The tool uses real-time updates on accuracy-critical scenarios while ensuring complete visibility of coverage-critical scenarios through efficient static views. This efficiency allows up to thousands of timing scenarios to be loaded into a single machine, allowing task completion with limited compute resources.

The PrimeECO tool offers the following features not available from the individual tools:

- Integrated ECO implementation and co-optimization using placement, legalization, routing, extraction, and multi-scenario timing analysis in one shell
- Single-machine ECO that performs timing ECOs using any number of scenarios
- Multiple iterations of incremental ECO timing analysis, physical implementation, and extraction to achieve ECO closure in a single operation
- Manual ECOs in the physical layout GUI with multi-scenario path-based timing analysis and advanced features such as legal site location display and cross-probing
- Access to both physical and logical databases at the same time, allowing operations such as region and layer queries of netlist elements

The PrimeECO tool supports the ECO flow only at the block level, without hierarchy.

Running the PrimeECO Tool

Using the PrimeECO tool requires a PrimeECO license and licenses for the underlying tools: IC Compiler II, StarRC, PrimeTime SI, and PrimeTime ADV-PLUS. The underlying tools must be installed and you need to know how to invoke them.

The PrimeECO tool runs in a standalone shell environment. To invoke a session, use the `eco_shell` command at the operating system prompt:

```
$ eco_shell
...
eco_shell>
```

At the eco_shell prompt, you can perform a new PrimeTime analysis or restore a previously saved PrimeTime session. However, running PrimeECO requires block-level physical design data: an NDM database from the IC Compiler II or Fusion Compiler tool or LEF/DEF files from a third-party layout tool.

The commands to perform a full ECO iteration, including implementation, extraction, and timing signoff, are similar to the PrimeTime commands that create an ECO change script.

The following example compares the ECO flow using discrete tools with the PrimeECO flow.

```
PrimeTime + ICC2 + StarRC Flow        PrimeECO Flow

set_eco_options \                     set_eco_options \
 -physical_icc2_lib my_NDM \           -physical_icc2_lib my_NDM \
 -physical_icc2_blocks CPU_routed      -physical_icc2_blocks CPU_routed

check_eco                             set_implement_options \
fix_eco_timing                         -icc2_exec icc2_shell \
report_global_timing                   -starrc_exec StarXtract ...

write_changes -format icc2tcl ...     check_eco
                                      fix_eco_timing
# IC Compiler II implementation       report_global_timing

# StarRC extraction                   implement_eco
                                      report_global_timing
# Repeat all for ECO closure          save_block -as CPU_after_eco
```

In the PrimeECO flow, the `implement_eco` command performs all implementation, extraction, timing analysis, and timing signoff steps in a single operation. Multiple change iterations occur in the background, using co-optimization to achieve a high quality of results.

For distributed multi-scenario analysis (DMSA), the commands are also similar:

```
PrimeTime + ICC2 + StarRC DMSA Flow    PrimeECO DMSA Flow

remote_execute -verbose {              remote_execute -verbose {
 set_eco_options \                      set_eco_options \
  -physical_icc2_lib my_NDM \           -physical_icc2_lib my_NDM \
  -physical_icc2_blocks CPU_routed }    -physical_icc2_blocks CPU_routed }

remote_execute {check_eco}             set_implement_options \
fix_eco_timing                          -icc2_exec icc2_shell \
report_global_timing                    -starrc_exec StarXtract ...

write_changes -format icc2tcl ...      check_eco
                                       fix_eco_timing
# IC Compiler II implementation        report_global_timing

# StarRC extraction                    implement_eco
                                       report_global_timing
# Repeat all for ECO closure           save_block -as CPU_after_eco
```

If you direct the PrimeECO tool to perform changes incrementally, the final results might vary slightly from full-design RC extraction and timing analysis. To ensure accurate signoff analysis, perform a full StarRC extraction and PrimeTime timing analysis.

During ECO analysis, the tool creates subdirectories named `eco`, `icc2`, `lef`, and `starrc` in the working directory specified by the `set_implement_options -work_dir` command. The tool uses these directories to store intermediate data files and log files.

If an error occurs during the ECO implementation step, the PrimeECO tool saves the physical implementation work so that you can correct the error and resume the flow from the point at which it stopped.

For a DMSA run, use the `remote_execute` command to run the `set_eco_options` command remotely, then run the `set_implement_options` command at the master process:

```
remote_execute {
  set_eco_options \
    -physical_icc2_lib Design.nlib \
    -physical_icc2_blocks Block_pre_eco
}

set_implement_options \
  -icc2_exec /myexec/icc2_shell \
  -starrc_exec /myexec/starrc \
  -starrc_cmd_files /mydata/mycmds.cmd \
  -work_dir $curr_dir/dcs_work_dir \
  -parasitic_corners { {setup1 typ_worst} {hold1 typ_best} }
```

The PrimeECO Flow

In general, a PrimeECO session consists of the following steps:

1. Specify the physical library and physical block:

   ```
   eco_shell> set_eco_options \
    -physical_icc2_lib my_NDM \
    -physical_icc2_blocks CPU_routed \
    ...
   ```

2. Set the implementation and extraction options:

   ```
   eco_shell> set_implement_options \
    -icc2_exec icc2_shell \
    -starrc_exec StarXtract \
    -starrc_cmd_files starrc.cmd \
    -work_dir $curr_dir/dcs_work_dir \
    ...
   ```

   You must specify the IC Compiler II and StarRC executable commands.

3. Run a check and initialization of the ECO setup:

   ```
   eco_shell> check_eco
   ```

This opens the block, creates a temporary working view of the block, and runs a StarRC extraction to establish a baseline for subsequent incremental extraction.

4. Enter the ECO fixing command:

```
eco_shell> fix_eco_timing ...
```

This can be the `fix_eco_timing`, `fix_eco_drc`, or `fix_eco_power` command. Use the fixing options you want in the command. Check the fixing results as you would in a PrimeTime session.

5. Implement the ECO changes:

```
eco_shell> implement_eco
```

This command performs incremental physical implementation, including placement, legalization and ECO routing, as well as RC extraction and timing analysis for all changes made by previous `fix_eco_timing`, `fix_eco_drc`, and `fix_eco_power` commands.

The default extraction is full extraction. You can specify incremental extraction by using the following command:

```
eco_shell> set_implement_options -starrc_mode incremental
```

6. Report the global timing to evaluate the quality of results:

```
eco_shell> report_global_timing ...
```

You can also perform a legality and routing check by using the `check_legality` and `check_route` commands.

7. Save the modified block to the design database:

```
eco_shell> save_block -as CPU_after_eco
```

PrimeECO Flow Options

At the start of the PrimeECO flow, set the physical implementation and extraction options using the `set_implement_options` command. You must at least specify the IC Compiler II and StarRC executable commands. You can optionally set several other implementation and extraction options.

```
set_implement_options

   -icc2_exec path
  [-icc2_max_cores core_count]
  [-icc2_post_link_script file]
  [-icc2_pre_legalize_script file]
```

```
 [-icc2_pre_route_script file]
 [-icc2_pre_extract_script file]
 [-icc2_eco_legalize_script file]
 [-icc2_eco_route_script file]

  -starrc_exec path
 [-starrc_cmd_files file_list]
 [-starrc_mode mode]
 [-parasitics_extractor mode]
 [-parasitic_corners list]

 [-insert_fillers]
 [-work_dir path]
```

For details, see the man page for the `set_implement_options` command.

Physical Implementation Options

If you specify filler cells using `set_eco_options -filler_cell_names`, the tool incrementally inserts filler cells in changed regions. The legalizer places ECO cells considering existing filler cells as empty spaces. After legalization is complete, the tool removes filler cells that overlap moved cells or ECO cells and inserts new filler cells in the changed regions.

The following is PrimeECO script example for incremental filler cell insertion.

```
# Specify filler lib cell names
set_eco_options -filler_cell_names $filler_cell_names

# Enable incremental filler cell insertion
set_implement_options -insert_fillers
```

The tool performs incremental metal fill insertion by running IC Validator In-Design inside the IC Compiler II executable. If the design has metal fill, the tool inserts metal fill incrementally by removing existing metal fills in the changed region and refills it incrementally.

If the design does not already have metal fill, you can set up the following script and specify it with the `-icc2_post_link_script` option of the `set_implement_options` command. Suppose the script icv_setup.tcl contains the following:

```
# Setup or run ICV
# This script is called right after ICC2 open_block is executed
#
set env(ICV_HOME_DIR) /global/apps/icv_2019.06-SP1-2
set target_arch "LINUX.64"
set env(PATH) "$env(ICV_HOME_DIR)/bin/$target_arch:$env(PATH)"
signoff_create_metal_fill -track_fill generic
```

You can specify this script file to perform initial metal fill with IC Validator tool:

```
# Run IC Validator before ECO
#
set_implement_options -icc2_post_link_script ./icv_setup.tcl
check_eco
```

The following IC Validator script, incr_metal_fill.tcl, performs incremental metal fill:

```
# Incremental threshold 20%
set_app_option \
  -name signoff.create_metal_fill.auto_eco_threshold_value -value 20
# Analyze changes and perform incremental metal fill
set percentage_change \
  [signoff_create_metal_fill -auto_eco true -track_fill generic]
#Rerun metal fill if design was changed more than 20% threshold
if {$percentage_change >= 20} {
   signoff_create_metal_fill -track_fill generic
}
```

If the area of change is greater than 20 percent, IC Validator abandons incremental metal fill and performs full metal fill instead. To configure the script to run in the PrimeECO tool:

```
# IC Compiler II Metal fill script invocation
#
set_implement_options -icc2_pre_extract_script ./incr_metal_fill.tcl
implement_eco
```

Perform incremental metal fill before extraction.

RC Extraction Options

The tool supports both incremental and full extraction. The default is full. Use the following command to specify incremental extraction:

```
eco_shell> set_implement_options -starrc_mode incremental
```

Full extraction is recommended for signoff timing accuracy after extensive changes made by running the `fix_eco_timing` or `fix_eco_power` commands. Incremental extraction is recommended when you make a small number ECO changes manually and you need quick feedback to assess the effects of those changes.

The PrimeECO RC extraction process uses the StarRC command file specified by the `-starrc_cmd_files` option of the `set_implement_options` command. However, the tool ignores certain StarRC commands and adds others before performing extraction. The changes are necessary to configure the extraction according to the PrimeECO operating mode, such as full or incremental extraction, and configure NDM-related information.

- The PrimeECO tool ignores the following StarRC commands in the StarRC command file:

  ```
  STAR_DIRECTORY
  GPD
  ECO_MODE
  NETLIST_INCREMENTAL
  NDM_DATABASE
  BLOCK
  COUPLING_REPORT_FILE
  SUMMARY_FILE
  ```

- The PrimeECO tool adds the following StarRC commands to the command file:

  ```
  NDM_DATABASE
  BLOCK
  ECO_MODE: YES
  NETLIST_INCREMENTAL: YES
  GPD
  ```

RC Extraction in a DMSA Run

In a DMSA run, you can configure PrimeECO fixing to use multiple StarRC command files. For example,

```
eco_shell> set_implement_options \
             -starrc_cmd_files {hot.corners.cmd cold.corners.cmd}
```

Each command file can be associated with one or more extraction corners:

```
$ grep SELECTED_CORNERS *.cmd
starrc.cold.corners.cmd: SELECTED_CORNERS: cold.BC.corrner
 starrc.hot.corners.cmd: SELECTED_CORNERS: hot.BC.corner hot.WC.corner
```

During fixing, the tool performs extraction at the required corners, grouped by the command files associated with each corner:

```
Information: Extraction starting at [Thu Jan  9 02:09:23 2020]
Information: Executing 'StarXtract .../starrc.hot.corners.cmd/starrc.cmd'
Information: Executing
 'StarXtract .../starrc.cold.corners.cmd/starrc.cmd'
...
Information: '.../ecoBlock_pid123_1.starrc.hot.corners.cmd.gpd' has been
  generated
Information: '.../ecoBlock_pid123_1.starrc.cold.corners.cmd.gpd' has been
  generated
...
Information: Reading parasitics from
  '.../ecoBlock_pid123_1.starrc.hot.corners.cmd.gpd'
  for scenario(s) 'hot.BC hot.WC' at [Thu Jan  9 02:13:26 2020]...
Information: Reading parasitics from
```

```
'.../ecoBlock_pid123_1.starrc.cold.corners.cmd.gpd'
for scenario(s) 'cold.BC' at [Thu Jan  9 02:13:31 2020]...
```

The resulting parasitics are used in the DMSA scenarios according to the mapping specified by the `-parasitic_corners` options of the `set_implement_options` command.

Levels of What-If Analysis

PrimeECO provides three levels of timing update to trade off accuracy and runtime, appropriate for different stages of ECO cycles:

1. Instant timing update on path collections

   This mode is enabled when the GUI is first invoked. It performs a fast timing update on the collection of paths from multiple scenarios, processing one million paths per second.

   This mode is useful for manual ECO changes. For example, you can insert a buffer or size a cell in the GUI and quickly assess timing path changes. If the change creates new violations, you can adjust the changes or go back to the original netlist.

2. Incremental and full timing update before placement and routing

   This is the traditional PrimeTime incremental and full timing update after ECO changes. It predicts timing without ECO placement and routing.

   When you are done with manual ECO changes, you can run a PrimeTime timing update for final signoff timing. If you discover unexpected timing changes, you can go back to level 1 to fix them.

3. Timing update after placement and routing

   When you have confidence in your ECO changes, you can perform full implementation, extraction, and signoff timing analysis.

Feedback

2

PrimeECO Technologies

The PrimeECO tool offers the following advanced ECO technology features for fast turnaround and high quality of results:

- Co-Optimization
- Incremental ECO Quality of Results
- Manual ECOs in the GUI
- Custom ECO Scripts
- Hybrid Timing View ECO

Co-Optimization

The PrimeECO timing analyzer, placement legalizer, and router work together as a single tool to perform several types of optimization. For example, cells with the most critical path-based timing have higher priority for legalization with minimal displacement. This allows buffer insertion in high-density regions with little timing degradation.

Another example is the optimization of wires with critical timing. The router uses layer promotion, increased wire spacing, and detour route removal to fix critical timing.

The PrimeECO tool places and legalizes ECO cells to minimize displacement and unexpected timing changes. If the tool cannot find good places for ECO cells, it moves neighboring cells that have positive slack to create the needed space. When neighboring cells are moved, the legalizer and timer work together to avoid creating new timing violations.

You can specify the maximum allowed cell displacement to control unexpected timing changes. For example, the following command limits cell movement to a distance no greater than 2.5 times the minimum site row height:

```
eco_shell> implement_eco -max_displacement_in_site_rows 2.5
```

This is a soft displacement limit. The tool tries to honor the limit but does not guarantee it.

Incremental ECO Quality of Results

The PrimeECO tool performs physical implementation, extraction, and timing analysis in multiple incremental iterations. It completes ECO closure in a single-shell environment, producing better results in less time and with greater reliability than using separate tools one at a time.

After ECO initialization, all internal steps are performed incrementally, including legalization, ECO routing, extraction, and timing analysis. Legalization operates only on changed cells and affected neighbor cells. The router works only on the nets changed during legalizations. Extraction works only on the parts of the design affected by routing changes. Timing analysis reads changed objects from the physical database, reads the incremental parasitics, and runs incremental timing updates.

Because the PrimeECO tool performs changes incrementally, the final results might vary slightly from performing a full-design RC extraction and timing analysis. To ensure a fully accurate analysis for final signoff, perform a full-design StarRC extraction and PrimeTime timing analysis.

Manual ECOs in the GUI

The PrimeECO GUI supports the same features as the PrimeTime GUI. In addition to exploring the timing effects of ECO changes, you can implement those changes in the physical design database.

The following script shows a typical PrimeECO GUI session with DMSA:

```
# Analysis portion of DMSA flow
create_scenario -name S1 ...
create_scenario -name S2 ...
set_host_options ...
start_hosts
current_session -all
current_scenario -all
report_global_timing

# Interactive ECO with implementation in DMSA using GUI
remote_execute {
  set_eco_options \
    -physical_icc2_lib Design.nlib \
    -physical_icc2_blocks Block_pre_eco }
set_implement_options ... -starrc_mode incremental

check_eco                 # Load physical database
load_distributed_design   # Load design data on DMSA master
update_timing             # Confirm timing is up-to-date
start_gui
```

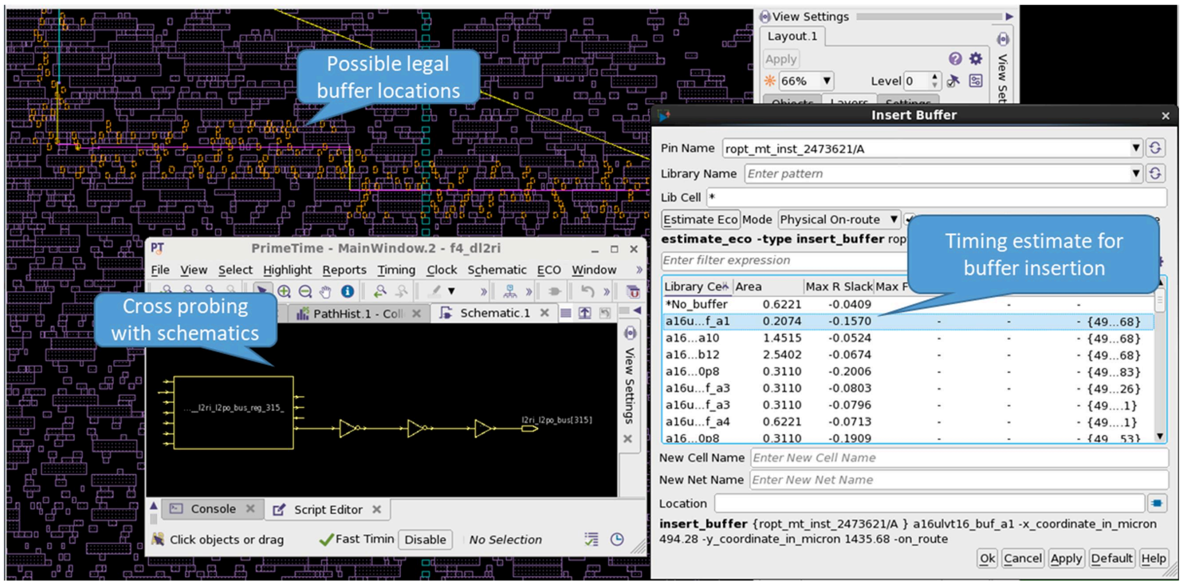
```
# Explore manual ECO options in GUI
report_timing -to $violating_endpoint
size_cell ...
insert_buffer ...
...
report_timing -to $violating_endpoint

# Implement changes; fast incremental extraction and timing update
implement_eco

# Save physical block
save_block -as Block_eco_1
```

During manual GUI ECO exploration, you can do the following:

- Perform ECO editing such as physical buffer insertion and cell sizing
- Highlight PBA timing paths analyzed in different scenarios in the chip layout
- Cross-probe nets and cells in the path analyzer, design schematic, and chip layout

The GUI shows possible legal locations for buffer insertion as well as timing estimates, as shown in Figure 4.

Figure 4 Buffer Insertion in the PrimeECO GUI


Possible legal buffer locations
Cross probing with schematics
Timing estimate for buffer insertion
PrimeTime - MainWindow.2 - f4_dl2ri
File View Select Highlight Reports Timing Clock Schematic ECO Window
PathHist.1 - Coll
Schematic.1
Console
Script Editor
Click objects or drag
Fast Timin
Disable
No Selection
View Settings
Layout.1
Apply
66%
Level 0
Insert Buffer
Ok Cancel Apply Default Help

Pin Name: ropt_mt_inst_2473621/A
Library Name: Enter pattern
Lib Cell: *
Estimate Eco Mode: Physical On-route
estimate_eco -type insert_buffer rop
Enter filter expression

Library Cell	Area	Max R Slack	Max F			
*No_buffer	0.6221	-0.0409		-	-	
a16u...f_a1	0.2074	-0.1570	-	-	-	{49...68}
a16...a10	1.4515	-0.0524	-	-	-	{49...68}
a16...b12	2.5402	-0.0674	-	-	-	{49...68}
a16...0p8	0.3110	-0.2006	-	-	-	{49...83}
a16u...f_a3	0.3110	-0.0803	-	-	-	{49...26}
a16u...f_a3	0.3110	-0.0796	-	-	-	{49....1}
a16u...f_a4	0.6221	-0.0713	-	-	-	{49....1}
a16...0p8	0.3110	-0.1909	-	-	-	{49...53}

New Cell Name: Enter New Cell Name
New Net Name: Enter New Net Name
Location:

```
insert_buffer {ropt_mt_inst_2473621/A } a16ulvt16_buf_a1 -x_coordinate_in_micron
494.28 -y_coordinate_in_micron 1435.68 -on_route
```

When you apply an ECO change, the path timing is updated incrementally for immediate evaluation. If you want to see signoff timing, you can implement the change with the `implement_eco` command to get final signoff timing after ECO legalization and routing.

Custom ECO Scripts

You can develop customized ECO scripts with full access to the physical layout data as well as static timing data. For example, you can edit a block netlist and propagate the changes to other instances (multiply instantiated modules, or MIMs) and check for timing violations in other blocks.

The following command specifies custom configuration scripts for IC Compiler II setup, legalizing, and routing:

```
eco_shell> set_implement_options \
  -icc2_post_link_script ./my_icc2_config.tcl \
  -icc2_pre_leglize_script ./my_legalize_config.tcl \
  -icc2_pre_route_script ./my_router_config.tcl
```

When you run the `implement_eco` command, the tool sources the respective configuration scripts at the times indicated in the option names: after linking, before legalizing, and before routing.

Instead of allowing PrimeECO to control placement legalization and routing, you can direct the tool to use custom scripts to perform these steps, as shown in the following command:

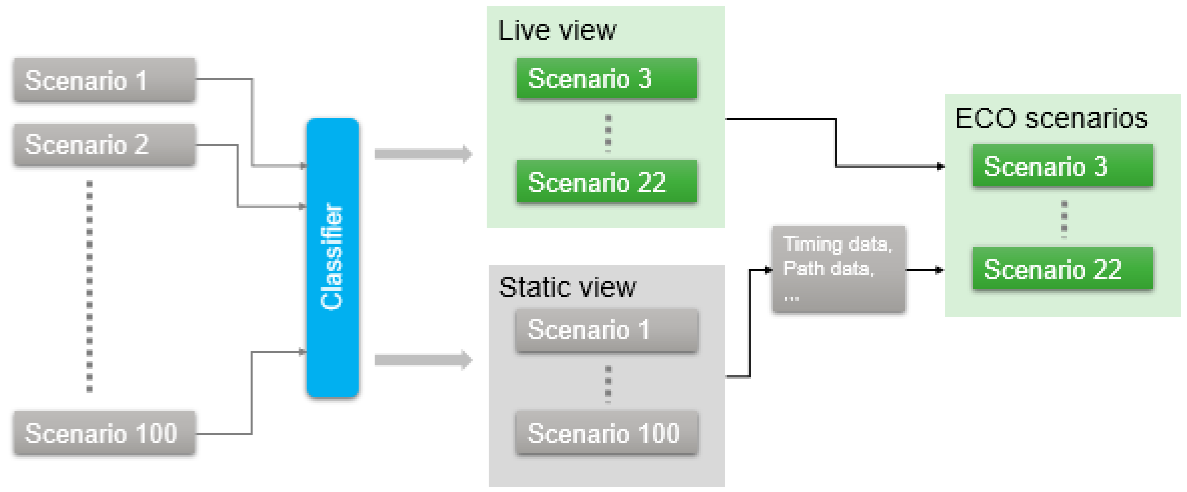
```
eco_shell> set_implement_options \
  -icc2_eco_legalize_script ./my_legalize.tcl \
  -icc2_eco_route_script ./my_route.tcl
```

Hybrid Timing View ECO

The PrimeECO hybrid timing view offers a scalable method to achieve accurate DMSA multi-scenario timing coverage using limited compute resources. Using this method, the tool divides the scenarios into live and static views, as follows:

- *Live view* scenarios run live and perform accuracy-critical ECO operations. The timing information is updated using PrimeTime signoff accuracy.
- *Static view* scenarios are inactive during ECO operations. The timing and violation information from these scenarios is merged into the live-view scenarios at intervals determined by the tool.

Figure 5 Live and Static Merging of Hybrid Timing Views



Suppose there are 100 scenarios, S1 to S100, and each scenario uses 10 GB of memory. To run all scenarios live, you would need 100 host processes and 1,000 GB of memory. The hybrid timing view feature allows you to reduce the number of live scenarios, which reduces the resource requirements.

The hybrid timing view feature supports setup, hold, maximum transition, and noise fixing. The resulting quality of results is very similar to using full DMSA ECO.

The following topics provide more information about the PrimeECO hybrid timing view feature:

- Exploring Live View Configurations
- Running ECO Fixing With Hybrid Timing Views
- Configuring Live Views by Target Coverage

Exploring Live View Configurations

In the PrimeECO hybrid timing view, the *live scenario coverage* indicates what percentage of an analysis type (such as setup, noise, DRC) is covered by a set of live scenario views, with the remainder being covered by the static views.

The following commands explore various configurations of live versus static views to evaluate memory and coverage tradeoffs.

```
eco_shell> report_eco_scenarios -num_live_views 10
eco_shell> report_eco_scenarios -num_live_views 20
eco_shell> report_eco_scenarios -num_live_views 30
```

Each report shows the total number of scenarios, the selected number of live ECO scenarios, the number of remaining scenarios, and the coverage resulting from this selection, as shown in the following examples:

```
eco_shell> report_eco_scenarios -num_live_views 10

ECO scenario summary
--------------------------------------------------------
Number of all scenarios                              :   100
Number of ECO scenarios                              :    10
Number of merged and filtered scenarios              :    90
ECO scenario live view setup timing coverage         :  56.9%
ECO scenario live view hold timing coverage          :  54.1%
...
ECO scenario static view setup timing coverage       :  43.1%
ECO scenario static view hold timing coverage        :  45.9%
...
Scenario reduction                                   :   7.8X

(runs in 128 GB machine)
 ...


eco_shell> report_eco_scenarios -num_live_views 20

ECO scenario summary
--------------------------------------------------------
Number of all scenarios                              :   100
Number of ECO scenarios                              :    20
Number of merged and filtered scenarios              :    80
ECO scenario live view setup timing coverage         :  95.1%
ECO scenario live view hold timing coverage          :  95.5%
...
ECO scenario static view setup timing coverage       :   4.9%
ECO scenario static view hold timing coverage        :   4.5%
...
Scenario reduction                                   :   3.9X

(runs in 256 GB machine)
...
```

The report shows that using 20 live views achieves 95 percent scenario coverage.

Running ECO Fixing With Hybrid Timing Views

The following commands start ECO operations using 20 live views:

```
eco_shell> set_host_options -num_processes 20 ...
...
eco_shell> start_eco_scenarios
...
```

The `start_eco_scenarios` command starts running PrimeECO fixing using 20 live views, providing 95 percent scenario coverage, with periodic merging of data from the remaining static view scenarios.

You might want to include specific scenarios in live views to make sure they are fully covered. The following example shows how to include scenarios S56 and S74 in live views:

```
eco_shell> set_host_options -num_processes 20 ...
...
eco_shell> start_eco_scenarios -include_scenarios {S56 S74}
...
```

The following session explores the usage of different numbers of live views, reporting the timing and other data coverage versus the machine resource usage. Then it performs ECOs using 20 live views running on 20 host processes:

```
remove_multi_scenario_design
stop_hosts
remove_host_options

create_scenario -name S1 \
  -specific_data {specific_S2.tcl} -eco_data S1/eco_data
create_scenario -name S2 \
  -specific_data {specific_S2.tcl} -eco_data S2/eco_data
create_scenario -name S3 \
  -specific_data {specific_S3.tcl} -eco_data S3/eco_data

report_eco_scenarios -num_live_views 15
report_eco_scenarios -num_live_views 20
report_eco_scenarios -num_live_views 30

set_host_options -num_processes 20
start_eco_scenarios
fix_eco_timing -type setup ...
```

At the beginning of this flow, remote execution of the `write_eco_scenario_data` command writes out the timing information using a different directory for each scenario. The hybrid timing ECO flow uses this information during ECO fixing.

Here is a comparison of conventional versus hybrid timing view ECO DMSA scripts:

```
Conventional DMSA script:            Hybrid timing view DMSA script:

# Create scenarios                   # Create scenarios
create_scenario -name S1 \           create_scenario -name S1 \
  -specific_data s1.tcl                -specific_data s1.tcl \
                                       -eco_data s1_data
create_scenario -name S2 \           create_scenario -name S2 \
  -specific_data s1.tcl                -specific_data s1.tcl \
                                       -eco_data s2_data
create_scenario -name S3 \           create_scenario -name S3 \
  -specific_data s3.tcl                -specific_data s3.tcl \
                                       -eco_data s3_data

                                     # Find the best live view count
                                     report_eco_scenarios -num_live 3
                                     report_eco_scenarios -num_live 2
                                     report_eco_scenarios -num_live 1

# Start multi scenario STA           # Start ECO scenarios
set_host_options -num_proc 3 \       set_host_options -num_proc 2
  -submit_command ...                start_eco_scenarios
start_hosts                          report_global_timing
current_scenario -all
report_global_timing

# ECO and write changes              # ECO and write changes
fix_eco_timing ...                   fix_eco_timing ...
write_changes -output pteco_pnr.tcl  write_changes -output pteco_pnr.tcl
```

To export the ECO changes to external tools, use the `write_implement_changes` command.

Configuring Live Views by Target Coverage

You can directly start the number of live view configurations required to meet a minimum coverage requirement by using the `-coverage_live_views` option of the `start_eco_scenarios` command:

```
eco_shell> start_eco_scenarios \
             -coverage_live_views 0.90  ;# 90% coverage
```

The tool starts the number of live scenarios required to achieve the coverage for each analysis type considered by the hybrid timing view analysis. In the preceding example, each of the default analysis types, which are `setup`, `hold`, `max_transition`, and `max_capacitance`, must meet 90% coverage.

You can also use the `-type` option to explicitly specify which violation types to consider during coverage evaluation:

```
eco_shell> start_eco_scenarios \
             -coverage_live_views 0.90 \
             -type {setup hold max_transition}
```

If you include the `noise` analysis type in live view selection, then the within-rail and beyond-rail coverages must both meet the requirement.

The tool starts only the number of hosts required for the specified coverage, adjusting the `set_host_options` configuration accordingly. For example, consider the following resource configuration:

```
set_host_options -name HOST_A -num_processes 10
set_host_options -name HOST_B -num_processes 10
set_host_options -name HOST_C -num_processes 10
```

The tool adjusts the configuration as follows:

- If fewer hosts are needed, the tool starts only the required number of hosts starting from the first `set_host_options` command.

  If only 14 hosts are needed, the configuration is adjusted as follows:

  ```
  set_host_options -name HOST_A -num_processes 10
  set_host_options -name HOST_B -num_processes 4
  ## (note that HOST_C is not used at all)
  ```

- If more hosts are needed, the tool increases the quantity of the last `set_host_options` command to meet the required total.

  If 34 hosts are needed, the configuration is adjusted as follows:

  ```
  set_host_options -name HOST_A -num_processes 10
  set_host_options -name HOST_B -num_processes 10
  set_host_options -name HOST_C -num_processes 14
  ```

3

ECO Flows and Fixing Methods

If your design has timing, design rule, or noise violations, or you want to optimize area or power, you can use the engineering change order (ECO) flow to fix the violations, implement the changes, and rerun timing analysis.

ECO Fixing Commands

In the PrimeECO tool, an engineering change order (ECO) is an incremental change in a chip design to fix timing violations or design rule constraint (DRC) violations, or to reduce power. The PrimeECO tool finds these issues and corrects them by sizing cells, replacing cells, or inserting buffers, and analyzes the results of these physical changes.

The commands to perform ECO fixing are:

```
fix_eco_drc
fix_eco_timing
fix_eco_power
fix_eco_wire
implement_eco
```

Before you attempt ECO fixing, ensure that the design is fully placed and routed, with generated clock trees.

For more information, see the following topics:

- DRC, Crosstalk, and Cell Electromigration Violation Fixing
- Timing Violation Fixing
- Power Recovery Fixing
- Wire Optimization
- Order of ECO Fixing Steps

DRC, Crosstalk, and Cell Electromigration Violation Fixing

To fix design rule constraint (DRC), noise, crosstalk delay, or cell electromigration violations, use the `fix_eco_drc` command. It fixes the violations reported by the following commands:

```
report_constraint -max_capacitance ...
report_constraint -max_transition ...
report_constraint -max_fanout ...
report_noise ...
report_si_bottleneck ...
report_cell_em_violation ...
```

The `fix_eco_drc` command attempts to fix violations by sizing cells and inserting buffers or inverter pairs, while minimizing the impact on area. By default, it does not consider timing effects.

The `fix_eco_drc` command has options to specify the following parameters:

- The type of violation to fix and the target amount of slack to achieve
- The scope of the design to fix (a list of pins or ports, or the whole design)
- The fixing methods (cell sizing, buffer insertion, or inverter pair insertion)
- The list of library cells to use for buffer or inverter pair insertion
- Whether to modify cells in data paths or clock networks
- Whether to consider timing constraints during fixing, and if so, the required setup and hold timing margins
- The physical placement and sizing mode (none, open site, occupied site, or freeze silicon)

For example, the following command fixes maximum transition time design rule violations using cell sizing:

```
eco_shell> fix_eco_drc -type max_transition -methods {size_cell}
```

The following command fixes noise violations using both cell sizing and buffer insertion.

```
eco_shell> fix_eco_drc -type noise \
           -methods {size_cell insert_buffer} \
           -buffer_list {BUFX1 BUFX2 BUFX3} \
           -physical_mode open_site
```

The command specifies the list of library cell buffers to use and selects the “open site” physically aware fixing mode, which restricts cell sizing and buffer insertion to occur only where there is already enough room for the change.

The `fix_eco_drc` command performs multiple fixing iterations until all violations are fixed or it determines that further fixing is not worth the runtime cost, based on the current quality of results and fixing option settings.

Upon completion, the `fix_eco_drc` command shows the remaining violations and the number of unfixable violations. To generate a report on the reasons for the unfixable violations, use the `-verbose` option.

Crosstalk Delta Delay Fixing

The `fix_eco_drc` command supports ECO reduction of crosstalk delta delays on victim nets. For example,

```
eco_shell> fix_eco_drc -type delta_delay -verbose \
 -delta_delay_threshold 0.05 -methods {size_cell insert_buffer} \
 -buffer_list {BUFX1 BUFX2 BUFX3} -physical_mode open_site
```

This command performs cell sizing and buffer insertion on victim nets to reduce crosstalk delays. It targets all victim nets that have a delta delay cost exceeding 0.05 time units and that belong to a timing path with negative slack.

Because there is no constraint that limits the allowable delta delay and therefore no “delta delay slack,” you must specify a delta delay threshold for fixing. Specify the threshold as a positive value greater than zero. The same threshold applies to both slowdown and speedup delta delays.

Delta Delay Fixing and Path Slack

By default, delta delay fixing considers only the nets that belong to paths with a negative timing slack. You can change this behavior by using the `-slack_lesser_than` option of the `fix_eco_drc` command:

```
eco_shell> fix_eco_drc -type delta_delay -verbose \
 -delta_delay_threshold 0.05 -methods {size_cell insert_buffer} \
 -buffer_list {BUFX1 BUFX2 BUFX3} -physical_mode open_site \
 -slack_lesser_than 0.05
```

The command targets nets for fixing that belong to paths with a slack less than the specified value. Specify a positive value to achieve a positive timing margin, or a negative value to perform less fixing.

SI Bottleneck Optimization and Reporting

The `fix_eco_drc -type delta_delay` command chooses the nets to fix based on SI bottleneck analysis. To preview the nets targeted for fixing, run the `report_si_bottleneck` command first. For example,

```
eco_shell> report_si_bottleneck -cost_type delta_delay \
 -min -max -slack_lesser_than 0.0 -all_nets
...
Bottleneck Cost:  delta_delay
net               scenario            cost
--------------------------------------------
DX23[17]          scen1               0.4597
REG_ADDR[2]       scen2               0.3671
REG_DATA[5]       scen1               0.2921
RE_RDY            scen1               0.0934
...
```

This command reports the highest-cost nets first, considering both max and min analysis types together. The “cost” value in the report shows the absolute value of the delta delay on the net (negative delta delays are shown as positive). For a DMSA analysis, the report shows the merged results across all scenarios.

You can use the `report_si_bottleneck` command to preview and check the results of delta delay fixing in the same way that you use `report_constraints` for DRC fixing or `report_timing` for timing fixing.

Cell Electromigration Violation Fixing

The `fix_eco_drc` command can target the cell-internal electromigration DRC violations that are reported by the `report_cell_em_violation` command.

Cell electromigration effects occur inside logic gates when excessive current densities cause a gradual displacement of metal atoms, which can eventually cause a short or open in the metal structure. The PrimePower tool uses a combination of library data, design data, and switching activity data (and physical data, if available) to identify and report high-risk cells as electromigration DRC violations.

Cell electromigration analysis is performed by the PrimePower tool and requires a PrimePower license. For details, see the “Cell Electromigration Analysis” chapter of the *PrimePower User Guide*.

To use this feature, configure the cell electromigration analysis and run the `update_power` command, then target the `cell_em` fixing type of the `fix_eco_drc` command:

```
# perform power/electromigration analysis
set_app_var power_enable_analysis true
set_app_var power_enable_em_analysis true
update_power

# report cell EM violations
report_cell_em_violation

# fix cell EM violations
fix_eco_drc -type cell_em -methods {size_cell insert_buffer}
```

The `cell_em` fixing type can use the sizing and buffering methods to fix cell electromigration violations.

The `cell_em` fixing type supports only a single iteration. To fix additional violations, run the `update_power` command, then rerun the `fix_eco_drc -type cell_em` command again.

Timing Violation Fixing

To fix setup or hold timing violations, use the `fix_eco_timing` command, which reduces timing violations by sizing cells and inserting buffers or inverter pairs. It attempts to fix the specified types of violations while minimizing the impact on area and power.

The `fix_eco_timing` command has options to specify the following fixing parameters:

- The type of timing violations to fix (setup or hold)
- The scope of the design to fix (from/to startpoints/endpoints, path groups, a path collection, or the whole design)
- The fixing methods (cell sizing, buffer insertion, or inverter pair insertion)
- The list of library cells to use for buffer or inverter pair insertion
- Whether to modify cells in data paths or clock networks
- The target amount of timing margin (slack) to achieve
- The graph-based analysis mode (graph-based, path-based, or path-based exhaustive)
- The physical placement and sizing mode (none, open site, occupied site, or freeze silicon)
- Whether to minimize power while fixing timing

For example, the following command fixes setup violations throughout the design:

```
eco_shell> fix_eco_timing -type setup
```

The following command fixes hold violations throughout the design using both cell sizing and buffer insertion, and using exhaustive path-based analysis for reduced timing pessimism:

```
eco_shell> fix_eco_timing -type hold -pba_mode exhaustive \
           -buffer_list {BUFX2 DLY1X2 DLY2X2}
```

Each time you use the `fix_eco_timing` command, you must specify the fixing type, either setup or hold, because of the different nature of these violations. By default, setup fixing uses cell sizing alone to reduce data path delays, whereas hold fixing uses both cell sizing and buffer insertion to increase data path delays. By default, fixing occurs only in data paths, not in clock networks.

The command performs multiple fixing iterations, starting with the worst violations. It repeats fixing iterations until all violations are fixed or it determines that further fixing is not worth the runtime cost, based on the current quality of results and fixing option settings.

Setup fixing seeks to avoid introducing design rule checking (DRC) violations, but it is allowed to introduce hold violations because setup violations are harder to fix. Hold fixing seeks to avoid introducing both setup and DRC violations.

Upon completion, the `fix_eco_timing` command shows the remaining violations and the number of unfixable violations. To generate a report on the reasons for the unfixable violations, use the `-verbose` or `-estimate_unfixable_reasons` option.

Power Recovery Fixing

The `fix_eco_power` command tries to recover power and area as much as possible by downsizing cells in paths with positive setup slack or by removing buffers from paths with positive hold slack, without introducing or worsening timing and DRC violations.

By default, the `fix_eco_power` command performs power and area recovery by downsizing cells in all paths with positive setup slack. The command options let you specify any one of the following power recovery methods:

- Replace cells to minimize area (the default)
- Replace cells to minimize a library cell numeric attribute
- Replace cells based on library cell preference, as specified by an explicit string priority list
- Replace cells to minimize switching, leakage, or total power using power analysis data generated by the PrimePower tool (`update_power` command)
- Remove buffers from paths with positive hold slack
- Replace cells in the clock network to specifically minimize dynamic power

Power Recovery Examples

The following command recovers power by downsizing both sequential and combinational cells, without introducing new timing violations:

```
eco_shell> fix_eco_power
```

The following command recovers leakage power by swapping out cells with higher leakage and replacing them with cells that have lower leakage, using the library cell name prefixes to determine which cells are preferred.

```
eco_shell> fix_eco_power -pattern_priority {HVT MVT LVT}
```

The following command recovers area and power by downsizing cells and analyzing the timing effects using path-based analysis.

```
eco_shell> fix_eco_power -pba_mode path
```

The following command recovers leakage power by replacing low-threshold-voltage cells with high-threshold-voltage cells, based on the library cell name prefixes “HVT” and “LVT.”

```
eco_shell> fix_eco_power -pattern_priority {HVT LVT} -pba_mode exhaustive
```

The following command removes buffers in paths with positive hold slack.

```
eco_shell> fix_eco_power -methods {remove_buffer}
```

You can apply multiple fixing methods by running the `fix_eco_power` command repeatedly with different option settings.

Power Recovery Operation

Power recovery is an iterative process. The `fix_eco_power` command starts by making the targeted changes and then analyzes the design for new timing and DRC violations (or worsening of existing violations) caused by the changes. It incrementally backs out of previous changes to meet the timing and DRC constraints, and also explores further changes for improved power recovery. It analyzes the design again and repeats this process until it determines that the cost of further progress exceeds the potential benefit.

For all of the cell replacement methods, replacement occurs only when the following conditions are met:

- The change does not introduce or worsen any timing violations or DRC violations
- The replacement cell has the same logical function as the original cell and meets any usage restrictions defined by the `eco_alternative_cell_attribute_restrictions` and/or `eco_alternative_cell_instance_based_restrictions` variables
- The replacement cell is preferred over the original cell in one of the following ways, depending on the option settings: less area, smaller power attribute value, higher-priority name or attribute string, or less power based on PrimePower analysis data

For more information, see Power Recovery.

Wire Optimization

When the PrimeECO tool modifies cells during ECO fixing operations, timing might be adversely affected due to changes in the routing of related wires. With wire optimization, the tool can analyze the routing and the timing of target nets and perform wire routing ECO changes.

Wire optimization performs the following functions:

- Provides wire ECO solutions by using timing analysis simultaneously with the router for selected nets
- Optimizes selected nets by using a combination of techniques, including wire spacing, wire widening, wire rerouting, and layer changes
- Allows the tool to degrade the timing of noncritical nets to improve the routing of critical nets
- Provides timing estimates for wire ECO fixing
- Implements automatic filtering for negative routing changes

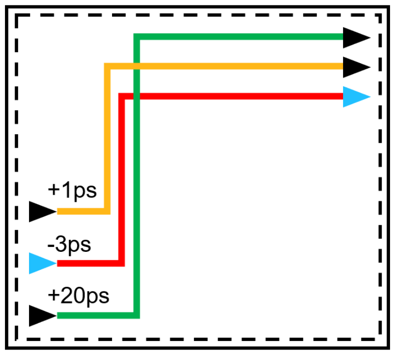
The `fix_eco_wire` command provides the wire ECO solutions to be invoked by a subsequent `implement_eco` command. You must specify the nets to evaluate with one of the following mutually exclusive options:

- Use the `-nets` option to specify nets explicitly.
- Use the `-path_selection_options` option to provide a string that serves as argument for the `get_timing_paths` command.

You can optionally specify the types of wire fixes to consider by setting the `-methods` option to a list of allowable methods. The default includes the `space_wire`, `widen_wire`, and `reroute_wire` settings. You can also enable layer changing by specifying the `change_layers` setting. If you select this method, you can restrict the layers to use for rerouting by using the `-min_routing_layer` and `-max_routing_layer` options. Without these options, the tool uses the available low-resistance layers.

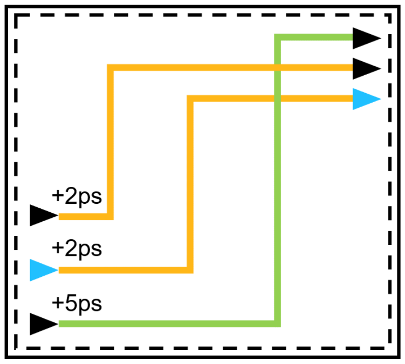
For example, Figure 6 contains a timing-critical path (shown in red) that has negative slack and is affected by crosstalk from neighboring nets. The neighboring nets have positive slack and are candidates for modification.

Figure 6 Before Wire Optimization



During wire optimization, the tool reroutes the neighboring nets to fix the timing violation on the critical net, as shown in Figure 7. As a result, the slack of one of the neighboring nets has been reduced, but is still positive.

Figure 7 After Wire Optimization



In the following example, the `fix_eco_wire` command specifies different fixing methods for several nets. The `reset_eco_wire` command resets netA to its pre-ECO state and the `report_eco_wire` command lists all of the changes specified by `fix_eco_wire` command.

```
fix_eco_wire -methods {reroute_wire} -nets {netA}
fix_eco_wire -methods {space_wire} -nets {netB netC}
fix_eco_wire -methods {space_wire widen_wire} -nets {netD}
fix_eco_wire -methods {reroute_wire change_layer} -nets {netE} \
   -min_routing_layer M5 -max_routing_layer M6
reset_eco_wire "netA"
report_eco_wire -verbose
```

In the following example, the `fix_eco_wire` command considers all fixing methods for the 10 worst violating paths and the `report_eco_wire` command lists the proposed changes.

The `report_global_timing` command estimates the global timing before and after ECO wire optimization occurs at the `implement_eco` command.

```
fix_eco_wire -methods {reroute_wire space_wire widen_wire change_layer} \
   -path_selection_options {-max_paths 10}
report_eco_wire -verbose
report_global_timing
implement_eco
report_global_timing
```

Specifying Routing Rules for Wire Optimization

You can control how the PrimeECO tool implements fixes for specific nets by using the `set_routing_rule` command.

Use the `-rule` option to specify a routing rule, which must already exist. Alternatively, use the `-default_rule` option to specify the default rule or the `-no_rule` option to remove any existing rules for the specified nets. These three options are mutually exclusive.

To specify the minimum and maximum routing layers, use the `-min_routing_layer` and `-max_routing_layer` options. Use the layer names from the technology file to specify the routing layers.

For example, to use layers M2 through M7 when routing the n1 net, use the following command:

```
eco_shell> set_routing_rule [get_nets n1] \
   -min_routing_layer M2 -max_routing_layer M7
```

By default, net-specific minimum layer constraints are soft constraints, while net-specific maximum layer constraints are hard constraints. You can change the default behavior as follows:

- To change the constraint strength for a single net-specific layer constraint, use the `-min_layer_mode` and `-max_layer_mode` options when you define the constraint with the `set_routing_rule` command.

  Set these options to `soft` to specify a soft constraint, `allow_pin_connection` to allow the use of lower layers only for pin connections, or `hard` to specify a hard constraint.

- To set the cost of violating soft constraints for a single net-specific layer constraint, use the `-min_layer_mode_soft_cost` and `-max_layer_mode_soft_cost` options when you define the constraint with the `set_routing_rule` command.

  Set these options to `low`, `medium` (the default), or `high`. The cost setting controls the effort expended by the router to avoid violations. The `high` setting can reduce violations at a cost of increased runtime. The `low` setting can reduce runtime at a cost of increased violations.

For example, the following command specifies that ECO fixes for netA and netB must use a routing rule named eco_ndr. In addition, the ECO fixes are limited to routing layers M8 and M9 and the fixing mode is `allow_pin_connection`.

```
eco_shell> set_routing_rule -rule eco_ndr "netA netB" \
   -min_routing_layer M8 -min_layer_mode allow_pin_connection \
   -max_routing_layer M9 -max_layer_mode allow_pin_connection
```

To report the nondefault routing rules present in the design, use the `report_routing_rules` command. To remove specific routing rule constraints, use the `set_routing_rule -clear` command. To reset all routing rules to their pre-ECO values, use the `reset_routing_rule` command.

Order of ECO Fixing Steps

The `fix_eco_*` commands and command options let you perform different types of ECOs. When deciding on the order in which to fix design rule constraint (DRC), setup, and hold violations, consider the status of your design and the ECO fixing goals.

Setup or hold fixing does not degrade DRC (max_capacitance and max_transition) violations, but DRC fixing can degrade setup or hold violations because DRC fixing has the highest priority. Also, hold fixing preserves setup slack, but setup fixing is allowed to introduce some hold violations, because setup is a harder problem to fix. Therefore, it is a good idea to fix DRC violations first, then setup violations, and finally hold violations.

If the number of changes is high and the changes disturb ECO route and legalization in IC Compiler or IC Compiler II, setup, hold, and DRC fixing can be done one at a time to reduce the disturbances that might occur in one pass. Therefore, you should first fix violations that cause more changes during ECO fixing.

To prioritize the fixing of timing violations over design rule violations, run the `fix_eco_timing` command with the `-ignore_drc` option. If you use this option, the command ignores and can potentially degrade the `max_transition`, `max_capacitance`, and `max_fanout` design rule violations. By default, the `fix_eco_timing` command does not fix timing violations on paths with design rule violations.

To optimize wire routing that might have been adversely affected by ECO cell changes, run the `fix_eco_wire` command after performing all other ECO fixing.

The following table summarizes the recommended flow for PrimeECO fixing.

Table 1 Recommended Order of PrimeECO Fixing

Steps	Commands	Fixing mechanisms	Precedence rules
Step1: Power recovery	`fix_eco_power`	Cell sizing, buffer removal	Does not introduce new timing or DRC violations

Table 1 Recommended Order of PrimeECO Fixing (Continued)

Steps	Commands	Fixing mechanisms	Precedence rules
Step 2: Fix DRC and noise violations	`fix_eco_drc`	Buffer insertion, cell sizing	Alters setup and hold slacks as needed
Step 3: Fix timing violations	`fix_eco_timing`	Buffer insertion, cell sizing	Setup fixing honors DRC and alters hold slack if needed; hold fixing honors setup slack and DRC
Step 4: Final leakage recovery	`fix_eco_power`	Threshold voltage swapping	Does not introduce new timing or DRC violations; does not change layout
Step 5: Wire fixing	`fix_eco_wire`	Wire spacing, wire widening, wire rerouting, layer changing	Does not introduce new timing or DRC violations

See Also

- DRC, Crosstalk, and Cell Electromigration Violation Fixing
- Timing Violation Fixing
- Power Recovery Fixing
- Setting the ECO Options

Setting the ECO Options

Before you run any ECO fixing commands, follow these steps to set the ECO options:

1. Import the design by reading the Verilog, SDC, physical data, and parasitic data files.
2. Set the `timing_save_pin_arrival_and_slack` variable to `true`.
3. Perform a full timing update (`update_timing` or `report_timing`).
4. Set the ECO variables that are relevant for your design or analysis conditions. For a complete list of all variables related to ECO, including the current and default settings, run this command:

```
eco_shell> report_app_var eco*
```

Table 2 Commonly Used ECO Variables

Condition	Relevant variables
Physical ECO	`eco_allow_filler_cells_as_open_sites` `eco_insert_buffer_search_distance_in_site_rows`
Design contains multiply instantiated modules (MIMs)	`eco_enable_mim`
Multi-scenario ECO with more scenarios than hosts	`eco_enable_more_scenarios_than_hosts`

5. Run the `set_eco_options` command with the relevant options for your design.

   To verify the ECO options, run the `report_eco_options` command. To reset the ECO options, run the `reset_eco_options` command.

Table 3 Commonly Used Options for the set_eco_options Command

To specify this information...	Use these set_eco_options command options

Data for physical ECO	`-physical_tech_lib_path`
	`-physical_lib_path`
	`-physical_design_path`
	`-physical_icc2_lib`
	`-physical_icc2_blocks`
	`-physical_constraint_file`
	`-physical_lib_constraint_file`

Handling of physical data	`-physical_enable_clock_data`
	`-physical_enable_all_vias`
	`-keep_site_names`
	`-power_ground_net_layer`
	`-enable_pin_color_alignment_check`
	`-allow_missing_lef`
	`-convert_sites`
	`-enable_via0_alignment_check`

Multiply instantiated module (MIM) groups	`-mim_group`
Filler cells	`-filler_cell_names`
Spare cells (freeze silicon flow)	`-programmable_spare_cell_names`

To specify this information...	Use these set_eco_options command options

Additional setup or hold timing margin	`-drc_setup_margin`
	`-drc_hold_margin`
	`-timing_setup_margin`
	`-timing_hold_margin`
	`-power_setup_margin`
	`-power_hold_margin`

Output log file	`-log_file`

Configuring the ECO for Multiple Libraries

If the tool must differentiate between multiple libraries with the same logical file name, specify how the tool records library cell names in the ECO change list by setting the `eco_write_changes_prepend_libname_to_libcell` and `eco_write_changes_prepend_libfile_to_libcell` variables. By default, these variables are set to `false`.

Excluding the Usage of Specific Library Cells (dont_use)

To exclude specific library cells from being used for ECO, set the `dont_use` attribute on those library cells by running the `set_dont_use` command:

```
set_dont_use lib_cell_list [true | false]
```

Preserving Parts of the Design (dont_touch)

To preserve parts of the design that were previously optimized, apply the `dont_touch` attribute to those parts. The ECO fixing commands do not make changes on objects that have the `dont_touch` attribute set to `true`.

Apply the `dont_touch` attribute by using the following commands.

Command	Description
`set_dont_touch`	Applies the `dont_touch` attribute to specific cells, nets, designs, or library cells.
`set_dont_touch_network`	Applies the `dont_touch` attribute on the transitive fanout of specific pins, ports, or clocks; recursively applies the attribute on all nets and combinational cells but stops at cells with setup or hold constraints, such as registers and latches.

To include reporting of the `dont_touch` attribute and similar attributes in timing reports, set the `timing_report_include_eco_attributes` variable:

```
eco_shell> set_app_var timing_report_include_eco_attributes true
true
```

```
eco_shell> report_timing ...
 ...
Attributes:
    d - dont_touch
    u - dont_use
    i - ideal_net
 ...

  Point                                  Incr    Path     Attributes
  -----------------------------------------------------------------
  clock clk (rise edge)                  0.00    0.00
  clock network delay (propagated)       3.30    3.30
  A1/B1/reg_out_reg_11_/CP (dfn)         0.00    3.30 r
  A1/B1/reg_out_reg_11_/Q (dfn)          0.87    4.18 f    i
  icc_route_opt6/ZN (inv7)               0.00    4.18 r    i d u
  icc_route_opt8/ZN (inv7)               0.07    4.25 f      d u
  reg_out[11] (out)                      0.01    4.25 f
  data arrival time                              4.25
```

Resizing Power Management Cells

By default, the ECO commands `fix_eco_timing`, `fix_eco_power`, and `fix_eco_drc` do not consider power management cells for possible resizing: level shifters, isolation cells, retention registers, and always-on cells.

To allow the tool to consider these power management cells for resizing during ECO generation, and to specify which types of cells to consider for resizing, set the `eco_allow_sizing_with_lib_cell_attributes` variable. For example,

```
eco_shell> set_app_var \
 eco_allow_sizing_with_lib_cell_attributes "is_level_shifter"
```

You can set the variable to a list of one or more of the following strings:

- `is_level_shifter` – Consider level shifters for resizing during ECO
- `is_isolation` – Consider isolation cells for resizing during ECO
- `is_retention` – Consider retention registers for resizing during ECO
- `always_on` – Consider always-on cells for resizing during ECO

When a library cell's attribute of the specified name (such as `is_isolation`) is set to `true`, that cell is considered for resizing.

Physical Data Files

For PrimeECO fixing, you need the following physical data files:

- Design Data Files – Design netlist, constraints, and timing information.
- Block-Level LEF Library Data – Library technology and physical cell information
- Physical Constraint File – Voltage areas and placement blockages in multivoltage designs (optional)
- Advanced Spacing Labels and Rules – Spacing labels and rules (optional)
- Parasitic Data From StarRC – Parasitic data with location information

After you prepare these files and set the ECO options (Setting the ECO Options), you can perform physical ECO fixing (Physical ECO).

Physical ECO

To perform physical ECO fixing:

1. Enable the reading of parasitics with location data by setting the `read_parasitics_load_locations` variable:

   ```
   eco_shell> set_app_var read_parasitics_load_locations true
   ```

2. Set the physical ECO options:

   - To enable the consideration of filler cells as open sites:

     ```
     set_app_var eco_allow_filler_cells_as_open_sites true
     ```

   - To specify the area in which the tool searches for an open site for buffer insertion:

     ```
     set_app_var eco_insert_buffer_search_distance_in_site_rows n
     ```

     where *n* is the number of site rows away in which to search (default 8)

   - To consider the presence of PG straps (typically vertical straps on bottom metal layers) and avoid overlaps with signal pins in the same layer:

     ```
     set_eco_options -power_ground_net_layer layer_name
     ```

3. Configure the physical design data location and configuration by using the `set_eco_options` command.

- For an NDM design database from the IC Compiler II or Fusion Compiler tool, the required `set_eco_options` command options are:

  ```
  set_eco_options
    -physical_icc2_lib icc2_lib_path
    -physical_icc2_blocks icc2_blocks
  ```

- For LEF/DEF files from a third-party layout tool, the required and optional `set_eco_options` command options are:

  ```
  set_eco_options
    -physical_tech_lib_path tech_LEF_files
    -physical_lib_path LEF_files
    -physical_design_path DEF_files
    -physical_constraint_file par_files
     || -physical_rules_file_path rules_file_path
    -tech_file_path tech_file
    -logical_design_path verilog_file
    -technology_node node_value
    -ndm_design_name design_name
    [-icc2_ref_lib ref_lib]
    [-routing_rule_file rule_file]
    [-upf_for_ndm upf_file]
  ```

- In both the NDM and LEF/DEF flows, you can use the following additional `set_eco_options` command options as needed:

  ```
  set_eco_options
    [-physical_constraint_file list]
    [-physical_lib_constraint_file list]
    [-log_file string]
    [-mim_group object]
    [-filler_cell_names list]
    [-programmable_spare_cell_names list]
    [-drc_setup_margin float]
    [-drc_hold_margin float]
    [-timing_setup_margin float]
    [-timing_hold_margin float]
    [-power_setup_margin float]
    [-power_hold_margin float]
    [-physical_enable_clock_data]
    [-physical_enable_all_vias]
    [-keep_site_names site_name_list]
    [-power_ground_net_layer layer_name_list]
    [-enable_pin_color_alignment_check]
    [-allow_missing_lef macro_name_list]
    [-convert_sites site_name_list]
    [-enable_via0_alignment_check]
    [-cell_em_profile string]
  ```

4. Specify the IC Compiler II and StarRC executables:

   ```
   set_implement_options \
     -icc2_exec icc2_shell \
     -starrc_exec StarXtract ...
   ```

5. (Optional) Preserve specific cells, nets, designs, and library cells by using the `set_dont_touch` command, which applies the `dont_touch` attribute to the specified objects.
6. Check your LEF/DEF files for missing, incomplete, or problematic data by running the `check_eco` command. If the `check_eco` command reports issues with the LEF/DEF files, resolve the issues, and rerun `check_eco` on the updated LEF/DEF files.
7. Run ECO fixing with these commands:
   - `fix_eco_drc -physical_mode ...`
   - `fix_eco_timing -physical_mode ...`
   - `fix_eco_power ...`

   To choose the `-physical_mode` setting, consider the design characteristics and ECO objectives:
   - `occupied_site` – Places or resizes cells even if no open sites are available; appropriate for an early-stage ECO or a high utilization design; results in higher fix rates due to more aggressive placement.
   - `open_site` – Places cells only in open sites; appropriate for a late-stage ECO or a low utilization design; results in smaller cell displacement due to more conservative placement.
   - `freeze_silicon` – Places new cells only on spare cells to preserve the silicon layers.
8. Generate the block-level and top-level ECO change list files by using the `implement_eco` command.

   ```
   implement_eco
   ...
   ```

   This initiates multiple cycles of physical implementation, extraction, and analysis.

For scripts that you can use as a starting point to implement the ECO flow, see the Synopsys Reference Methodology at https://solvnet.synopsys.com/rmgen.

Design Data Files

The PrimeECO tool requires the design data files shown in Table 4.

Table 4 Required Design Data Files for Physical ECO

Type of data	**Description**
Verilog netlist	Netlist written from IC Compiler II or Fusion Compiler, or from a third-party layout tool
Script and setup files	Files to configure SDC, UPF, DMSA, analysis setup

SPEF/GPD parasitics	Parasitics must have location information
	Use the following settings in the StarRC command file:
	`NETLIST_NODE_SECTION: YES`
	`REDUCTION: NO_EXTRA_LOOPS`

StarRC extraction script	Must be the same extraction script used to generate the initial parasitics in the preceding item
Physical design information (NDM or LEF/DEF)	NDM design database from IC Compiler II or Fusion Compiler, or LEF/DEF from a third-party layout tool (LEF/DEF must be consistent with netlist and parasitics files; reextract if necessary)
Additional physical constraints	Supplemental voltage area definitions and placement blockage constraints outside of NDM, if applicable

The design must be free of legality and routing violations. Otherwise, when the layout tool fixes these violations, it invalidates the initial state given to the PrimeECO tool. In the IC Compiler II or Fusion Compiler tool, use the following commands to check the design.

`check_legality -verbose`	All cells should be legal or fixed.
`check_route`	There should be no shorts or opens (or if so, all should be understood). All other DRC violations should be understood and expected to have no effect on ECO fixing.

To ensure that the design files are consistent with each other, generate them by following these steps *in this order*:

1. Open the block NDM in the IC Compiler II or Fusion Compiler tool.
2. Ensure that there are no legality or routing violations.

3. Write the Verilog netlist.

   This can modify netlist object names to meet Verilog name requirements. By performing this step first, the NDM block and the parasitics reflect any name changes.
4. Save the NDM block.
5. Extract the parasitics from the NDM saved in Step 4.

Specifying Physical Design (DEF) and Constraint Files

In a flat single-DEF flow, you can specify the DEF file—and optionally, a physical constraint file—in the global `set_eco_options` configuration command:

```
# configure everything in a single command
set_eco_options \
  -physical_tech_lib_path ... \
  -physical_lib_path ... \
  -physical_design_path TOP.def \
  -physical_constraint_file TOP_phys_cons.tcl
```

The `-physical_design_path` and `-physical_constraint_file` options are both applied to the physical block specified by the DESIGN statement inside the DEF file. In a hierarchical flow with multiple DEF design files, configure these options for each block using separate `set_eco_options` commands:

```
# configure global options
set_eco_options \
  -physical_tech_lib_path ... \
  -physical_lib_path ...

# configure top level with TOP* files
set_eco_options \
  -physical_design_path TOP.def \
  -physical_constraint_file TOP_phys.tcl

# configure block level with BLOCK* files
set_eco_options \
  -physical_design_path BLOCK.def \
  -physical_constraint_file BLOCK_phys.tcl
```

Later DEF and physical constraint file specifications overwrite previous file specifications applied to that same block:

```
set_eco_options \
  -physical_design_path BLOCK.def \
  -physical_constraint_file BLOCK_phys.tcl

# these NEW_* files are used instead of the original ones
set_eco_options \
  -physical_design_path NEW_BLOCK.def \
  -physical_constraint_file NEW_BLOCK_phys.tcl
```

Physical constraint file specifications remain until replaced or explicitly removed with an empty string (`{}`) specification:

```
set_eco_options \
  -physical_design_path BLOCK.def \
  -physical_constraint_file BLOCK_phys.tcl

# replace DEF, but keep constraint file
set_eco_options \
  -physical_design_path NEW1_BLOCK.def

# replace DEF and remove constraint file
set_eco_options \
  -physical_design_path NEW2_BLOCK.def \
  -physical_constraint_file {}
```

DEF to LEF Site Name Conversion

The physical ECO flow supports the mapping of DEF site names to LEF site names. For example,

```
eco_shell> set_eco_options -convert_sites {{unit CORE2T}} ...
```

This maps the site name “unit” in the DEF file:

```
ROW STD_ROW_324 unit 560 85680 FS ...;
ROW STD_ROW_325 unit 560 88200 N ...;
```

to the site name “CORE2T” in the LEF file:

```
SITE CORE2T
  CLASS CORE ;
  SYMMETRY X Y ;
  SIZE 0.14 BY 1.2 ;
END CORE2T
```

Missing LEF Files for Hierarchical Blocks

You can read in physical data in LEF/DEF format even if the DEF component for a hierarchical block lacks a LEF macro model, due to unavailable or incomplete LEF data. In these cases, the tool gets the size, shape, port/pin location, and routing obstruction data from the DEF description of the component.

Furthermore, you can specify a list of macro models in the design for which there is no LEF or DEF description available, and allow ECO operations to proceed without considering any instances of those models. Use the `set_eco_options` command as follows:

```
eco_shell> set_eco_options -physical_tech_lib_path $tech_lef_files \
                           -physical_lib_path $lef_files \
                           -physical_design_path design.def.gz \
                           -allow_missing_lef {X2 Y2}
```

In this example, physical ECO operations ignore macro models X2 and Y2. The ignored components are omitted from the GUI layout display.

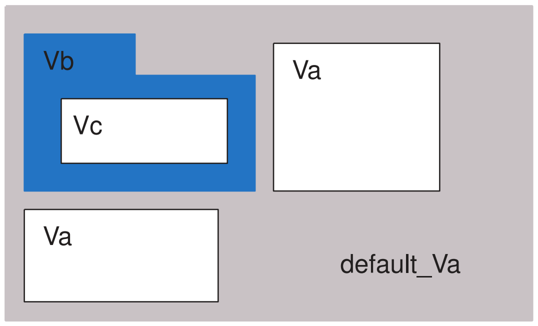
Because the tool ignores these components entirely, the physical spaces that they occupy are treated as empty. You can add placement blockages to these areas to prevent ECO commands from inserting or moving anything there. To report ignored macro models, use the `report_eco_options` command.

If you specify the names of macro models that have LEF or DEF data, the tool overrides your directive to ignore them, proceeds with using the data, and issues a warning message.

Physical Constraint File

When providing ECO guidance, the tool can consider physical information in multivoltage designs, such as disjoint and nested voltage areas.

Figure 8 Multivoltage Physical Constraint Example



- default_Va is the default voltage area
- Va is a disjoint voltage area
- Vc is nested inside Vb

The tool aims to fix late-stage violations with minimal changes to the existing layout and without generating new electrical rule checking (ERC) violations. To achieve this, you need a physical constraint file that defines voltage areas and placement blockages with these commands:

- `create_placement_blockage` – Creates a placement blockage.
- `create_voltage_area` – Creates a voltage area.

Advanced Spacing Labels and Rules

The physical ECO flow can consider advanced spacing rules to minimize displacements and timing degradation during ECO implementation. The tool can consider the following rules:

- Rules exported by the `export_advanced_technology_rules` command in the IC Compiler or IC Compiler II tool:
    - Minimum Vt rules
    - Jog (OD) rules
    - Trim poly (TPO) rules
    - Legal index rules
    - Vertical abutment rules
    - Advanced pairwise ECO physical-only cell spacing rules
- User-specified pairwise interlibrary cell spacing labels and rules

```
set_lib_cell_spacing_label
  -name label_name
  [-left_lib_cells {lib_cell_list}]
  [-right_lib_cells {lib_cell_list}]

set_spacing_label_rule
  -labels {label_name_list}
  { minimum maximum }
```

Parasitic Data From StarRC

The physical ECO flow uses the parasitic node locations to accurately determine the effects of placing an ECO cell on a net. For accurate computation, no loops should be present in the parasitic file. Therefore, set these options in the StarRC command file:

```
****************************************
*StarRC settings for physically-aware ECO
****************************************
REDUCTION: NO_EXTRA_LOOPS
NETLIST_NODE_SECTION: YES
```

Site-Aware Physical ECO Fixing

The physical ECO flow supports downsizing, upsizing, swapping, and insertion of single-height buffer cells, as well as and upsizing and insertion of multiple-height buffer cells for designs containing mixed single-height and double-height cells in the same placement region.

You can restrict usage of specific multiple-height ECO cells to specific site rows, for example, usage of cell `BUFFLVT` only in site rows of type `2Tunit`, as shown in the following figures.

Figure 9 Regular Single-Height and Double-Height Site Rows

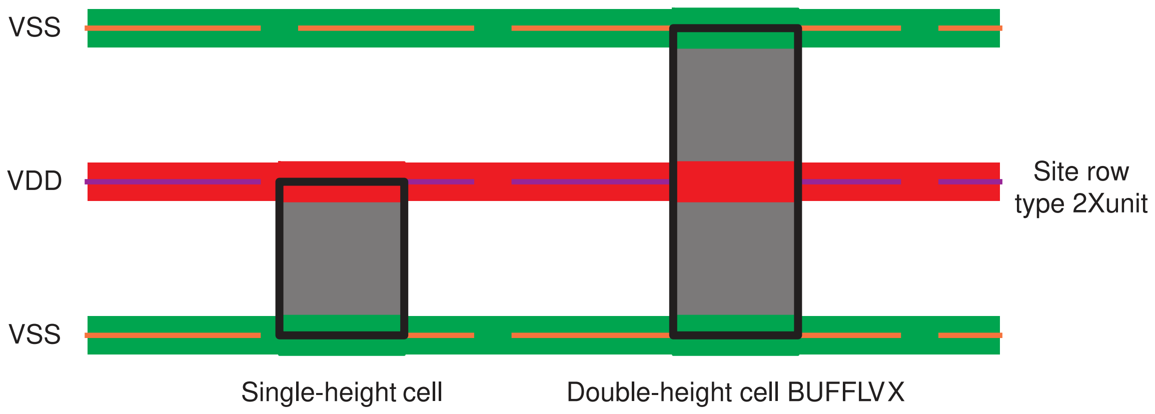
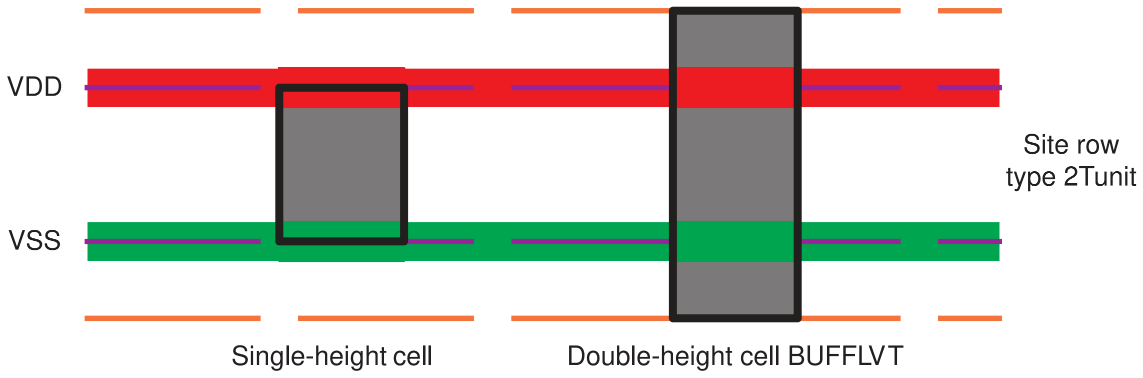


Figure 10 Offset Single-Height and Double-Height Site Rows



Note that physical ECO sizing only changes cells widths, not heights.

The multiple-height site row feature requires an exact match between the library cell LEF site name and the corresponding design DEF site row name. For example, here is a library cell definition in a LEF file that specifies the site name `2Tunit` for the cell:

```
MACRO BUFLVT1
  CLASS CORE ;
  ORIGIN 0 0 ;
  SIZE 1.632 BY 0.768 ;
  SYMMETRY X Y ;
  SITE 2Tunit ;
  ...
```

Here are site row definitions in a design DEF file that specify the same `T2unit` name:

```
ROW _row_1 unit 1500 1488 FS DO 4080 BY 1 STEP 96 0 ;
ROW _row_2 unit 1500 1872 N DO 4080 BY 1 STEP 96 0 ;
...
ROW 2Xrow_1 2Tunit 1500 1680 N DO 4080 BY 1 STEP 96 0 ;
ROW 2Xrow_2 2Tunit 1500 2448 N DO 4080 BY 1 STEP 96 0 ;
ROW 2Xrow_3 2Tunit 1500 3216 N DO 4080 BY 1 STEP 96 0 ;
...
```

To invoke site-aware ECO, set the `eco_physical_match_site_row_names` variable:

```
eco_shell> set_app_var eco_physical_match_site_row_names true
```

When the variable is set to `true`, ECO cell sizing and buffer insertion place the `BUFLVT1` macro cell only in `2Tunit` type rows.

You can change the `eco_physical_match_site_row_names` variable setting at any time to enable or disable the feature for successive open site ECO fixing operations. In the following example, ECO fixing is performed first in default mode using single-height cells, followed by site-aware ECO fixing performed on double-height cells in `2Tunit` rows.

```
set_eco_options \
  -physical_tech_lib_path technology.lef.gz \
  -physical_lib_path library.lef.gz \
  -physical_design_path design.def.gz \
  -physical_lib_constraint_file icc2_adv_tech_rule.gz \
  -log_file lef_def.log

# Keep 2Tunit site name data when reading LEF/DEF physical data
set_eco_options -keep_site_names {2Tunit}

# Check and read in LEF/DEF physical data
report_eco_options
check_eco

# Run default ECO with single-height cells, $SINGLE_BUF buffer list
set_app_var eco_physical_match_site_row_names false
fix_eco_drc -type max_transition -physical_mode open_site \
  -buffer_list $SINGLE_BUF -verbose
fix_eco_timing -type setup -physical_mode open_site -verbose
fix_eco_timing -type hold -physical_mode open_site \
  -buffer_list $SINGLE_BUF -verbose

# Run site-aware ECO with double-height cells, $DOUBLE_BUF buffer list
set_app_var eco_physical_match_site_row_names true
fix_eco_drc -type max_transition -physical_mode open_site \
  -buffer_list $DOUBLE_BUF -verbose
```

```
fix_eco_timing -type setup -physical_mode open_site -verbose
fix_eco_timing -type hold -physical_mode open_site \
  -buffer_list $DOUBLE_BUF -verbose
```

The site-aware feature can be used only when the ECO fixing command is invoked with the `-physical_mode open_site` option.

ECO Fixing Methods

You control ECO changes by using the appropriate ECO command, `fix_eco_timing`, `fix_eco_drc`, or `fix_eco_power`; and command options, such as `-type` and `-methods`. These are some of the fixing methods:

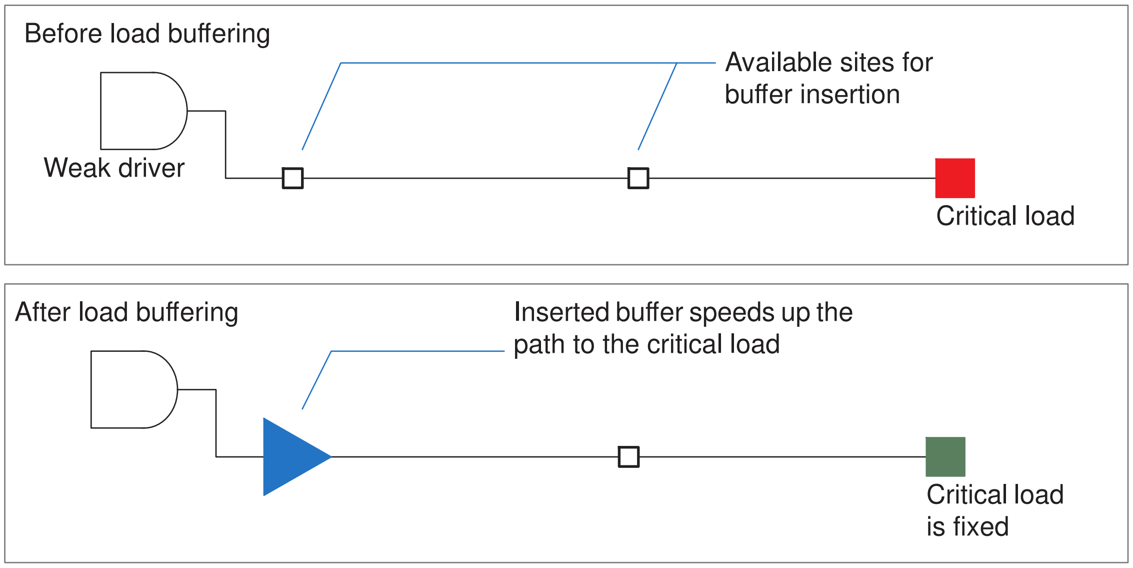
- Load Buffering and Load Shielding
- Side-Load Cell Sizing
- Clock Network ECO Fixing
- ECO Hold Fixing Using Load Capacitance Cells
- Power Recovery
- ECOs With Multiply Instantiated Modules (MIMs)

Load Buffering and Load Shielding

The tool can fix setup violations by performing physical load buffering and load shielding, resulting in more fixed violations.

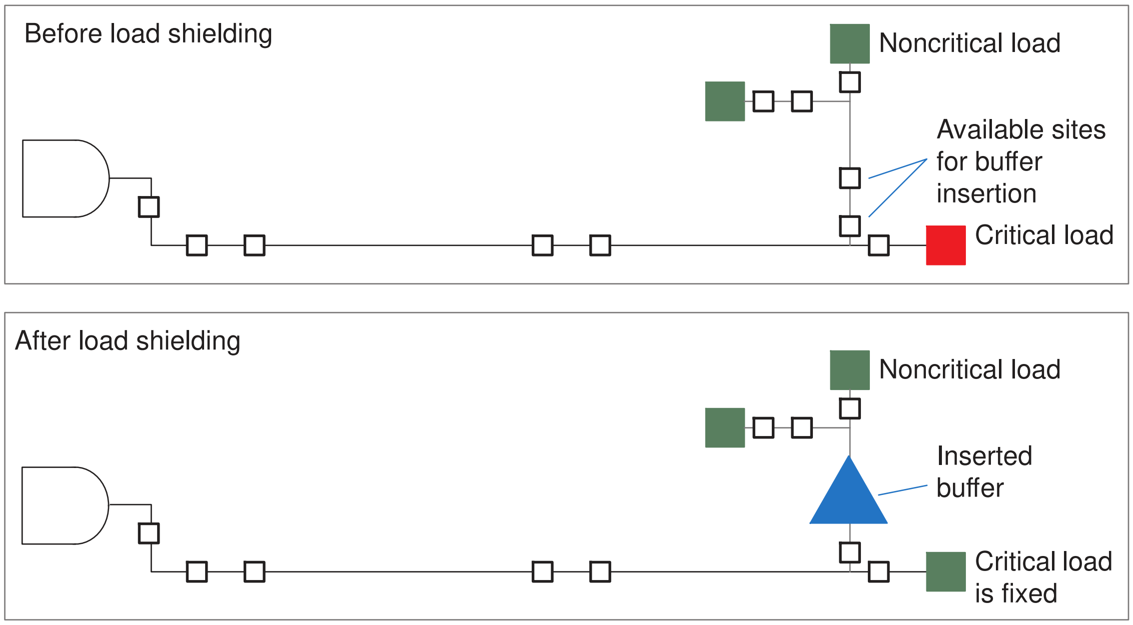
In load buffering, the tool inserts buffers to strengthen weak drivers. The tool automatically chooses the best available location for buffer insertion while considering placement blockages, as shown in the following example.

Figure 11 Load Buffering Example


Before load buffering
Available sites for buffer insertion
Weak driver
Critical load
After load buffering
Inserted buffer speeds up the path to the critical load
Critical load is fixed

In load shielding, the tool inserts buffers to shield critical loads from noncritical loads. The tool automatically chooses the best available location for buffer insertion while considering placement blockages, as shown in the following example.

Figure 12 Load Shielding Example



To fix setup violations with load buffering and shielding, use the `fix_eco_timing -type setup` command with the `-methods insert_buffer` option:

```
fix_eco_timing -type setup
  -methods insert_buffer
  -buffer_list {list_of_buffers}
  ...
```

You can use the `insert_buffer` and `size_cell` methods at the same time. If you use them separately for setup fixing, it is recommended to use `size_cell` first, and then `insert_buffer` and other methods to gain incremental improvements.

Side-Load Cell Sizing

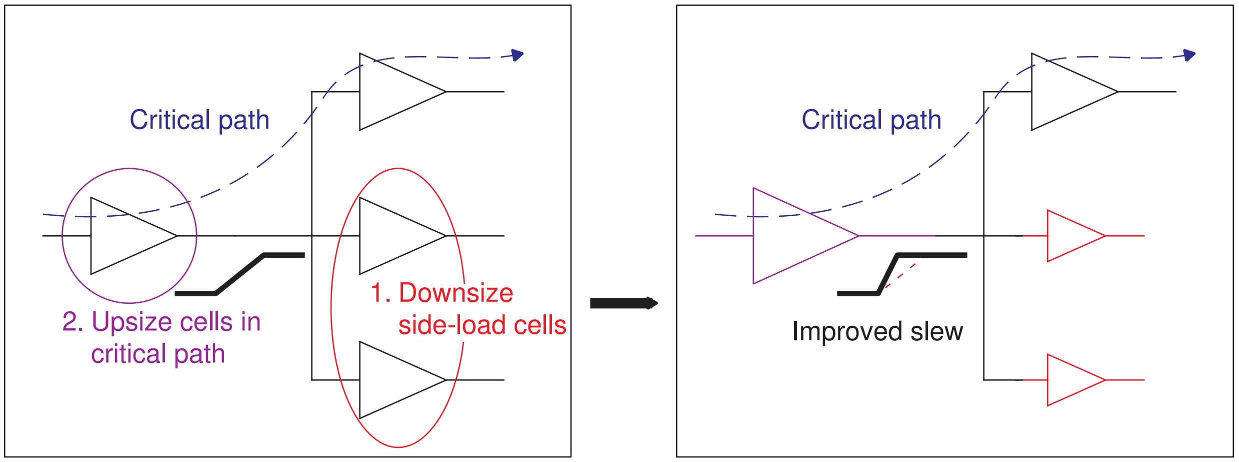
The `-methods` option of the `fix_eco_timing` command specifies the types of ECO changes performed to fix timing violations. The supported methods are cell sizing, general buffer insertion, buffer insertion at load pins, and side-load cell sizing.

The `size_cell_side_load` method first downsizes side-load cells in noncritical paths to reduce parasitic capacitance and then upsizes the cells in the critical path. This method works only with the `-type setup` and `-cell_type combinational` options. For example,

```
eco_shell> fix_eco_timing -type setup -methods {size_cell_side_load} ...
```

This command performs setup timing fixing as shown in the following figure.

Figure 13 Side-Load Cell Sizing Example



Clock Network ECO Fixing

By default, ECO fixing modifies only cells in data paths. However, you have the option to perform fixing in clock networks. Using this feature, you can fix setup and hold violations by changing clock arrival delays in the clock network, resolving hard-to-fix timing issues more efficiently; and you can fix DRC violations in the clock network.

Clock network ECO fixing is supported only in the physical ECO flow. Before you read in the LEF/DEF or IC Compiler II data, set the following ECO option to enable reading of physical data for the clock network:

```
set_eco_options -physical_enable_clock_data ...
```

Use the following ECO command options to perform clock network ECO fixing:

```
fix_eco_timing
 -type setup | hold
 -cell_type clock_network
 [-clock_fixes_per_change integer]
 [-clock_max_level_from_reg integer]

 [-methods {size_cell | insert_buffer | bypass_buffer \
                        | insert_inverter_pair}]
 ...

fix_eco_drc
 -type max_transition | max_capacitance | max_fanout
 -cell_type clock_network
 [-methods {size_cell | insert_buffer | insert_inverter_pair}]
 ...

fix_eco_power
 -power_mode dynamic
 -cell_type clock_network
 ...
```

Clock network ECO fixing can affect clock arrival skew in clock trees. To minimize changes to the arrival skew, perform data path ECO fixing first, then apply clock network ECO fixing to fix the remaining violations. For example,

```
eco_shell> fix_eco_timing -type setup -cell_type combinational ...
...
eco_shell> fix_eco_timing -type setup -cell_type clock_network ...
...
```

Fixing Timing Violations by Modifying Clock Networks

When you choose clock network ECO fixing, the `fix_eco_timing` command fixes setup or hold violations by modifying the clock paths to change the clock arrival times. For example,

```
eco_shell> set_eco_options -physical_enable_clock_data ...
...
eco_shell> fix_eco_timing -type setup \
           -methods {size_cell insert_buffer} \
           -buffer_list {buf2 buf4 buf6 buf8} \
           -cell_type clock_network      # Setup fixing in clock paths
...
```

With the `-cell_type clock_network` option, the command fixes setup violations by resizing and inserting buffers throughout the clock network. For example, it can upsize buffers in the launch clock path and insert or downsize buffers in the capture clock path.

Restricting Clock Network Fixing

You can restrict clock network ECO fixing by using additional options:

```
eco_shell> fix_eco_timing -type setup \
           -methods {size_cell insert_buffer} \
           -buffer_list {buf2 buf4 buf6 buf8} \
           -cell_type clock_network \
           -clock_fixes_per_change 4 \     # At least 4 fixes per change
           -clock_max_level_from_reg 6     # Up to 6 levels from register
```

- `-clock_fixes_per_change` – This option specifies a minimum number of violations to be fixed per change, such as 4 in this example. That means each change must fix at least four violations. For example, resizing a cell that fans out through buffers to four register clock pins can fix four timing violations. The default is 1.

  Setting a larger limit results in fewer changes, and the changes occur at higher levels in the clock tree, closer to the source and farther from the sequential cells.

- `-clock_max_level_from_reg` – This option specifies the maximum number of buffer cells away from the sequential cell where fixing can occur, such as 6 in this example. That means each buffer resizing or buffer insertion must occur no more than six buffers away from the register's clock pin. The default is no limit.

  Setting a smaller number limits the scope of the clock tree affected by each ECO change. For example, setting the limit to 1 limits each change to the immediate driver of the register clock pin. To restore the default (no limit), set the number to 0.

Clock Network Fixing With Buffer Bypassing and Insertion

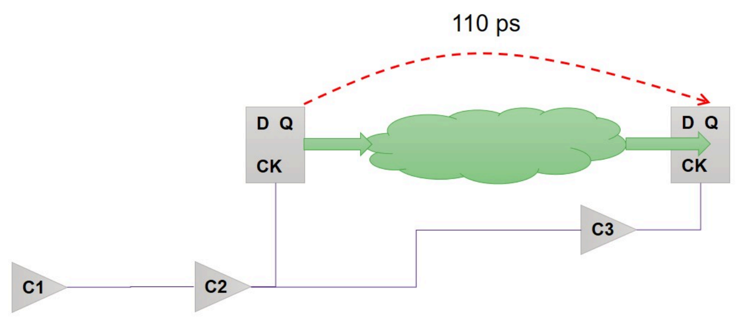
The `fix_eco_timing` command provides the `bypass_buffer` method for setup and hold fixing in clock networks. This method is valid only when the `-cell_type` option is set to `clock_network`.

With the `bypass_buffer` method, the tool adjusts the launch or capture path delay by bypassing one or more buffers that exist in the launch or capture paths.

For example, the circuit in Figure 14 has a setup violation with a slack of -10 ps. The following command is applied:

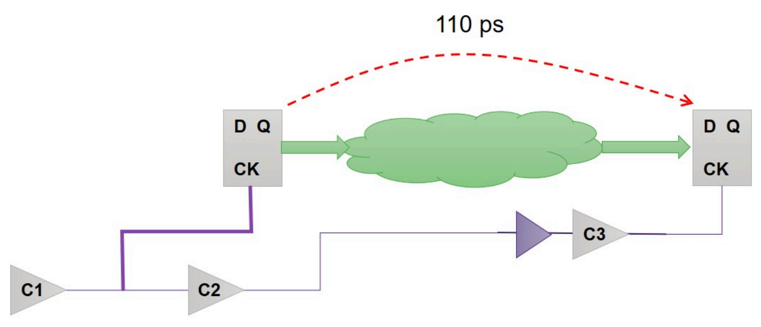
```
eco_shell> fix_eco_timing -type setup -cell_type clock \
                     -methods {insert_buffer bypass_buffer}
```

Figure 14 Setup Violation Before ECO Fixing



With the `bypass_buffer` method, the tool inserts a net in the launch path that bypasses buffer cell C2, which reduces the launch path delay. With the `insert_buffer` method, the tool inserts a new buffer in the capture path, which increases the capture path delay. The changes are shown in Figure 15. As a result, the setup slack improves to +1 ps.

Figure 15 Improved Setup Slack After ECO Fixing



The `bypass_buffer` method uses the `disconnect_net` and `connect_net` commands to accomplish the changes. These commands appear in the output from the `write_changes` command.

Because of the changes in the net topology, the original parasitics for the changed nets are no longer valid. The PrimeECO tool estimates parasitics using the Manhattan distances, annotates the estimated capacitances on the changed nets, and estimates timing using the new parasitics. You can see the annotated capacitances by using the `report_timing` command. After the changes are implemented with the `implement_eco` command, perform another iteration of extraction and timing analysis for final signoff.

The following reasons for unfixable problems appear in the `fix_eco_timing` report:

- C1: New net length after net switching is too long
- C2: Bypassed delay is too large after net switching
- C3: No nets available to switch
- C4: No nets to switch because parent nets have more violations in their fanout
- C5: The current net has pins across hierarchy and cannot be changed.

TNS-Driven Clock Network Fixing

The `fix_eco_timing` command has options to make clock network changes that fix some violations while allowing others to become worse, while targeting total negative slack (TNS):

```
fix_eco_timing
 [-target_violation_type endpoint | tns]
 [-wns_limit float]
```

The target violation type settings operate as follows:

- `endpoint` – Specifies the default targeting method, the same as omitting the `-target_violation_type` option. Changes to the clock network are not allowed to worsen any endpoint violation.
- `tns` – Specifies TNS-driven fixing. Changes to the clock network are allowed to worsen some violations while fixing others, while targeting total negative slack. The worst negative slack (WNS) is limited to either the existing WNS or a new value specified by the `-wns_limit` option.

A PrimeTime-ADV-PLUS license is required for TNS violation targeting.

Use the following command for TNS-driven clock network timing fixing:

```
eco_shell> fix_eco_timing -type ... \
 -cell_type clock_network \
 -target_violation_type tns
```

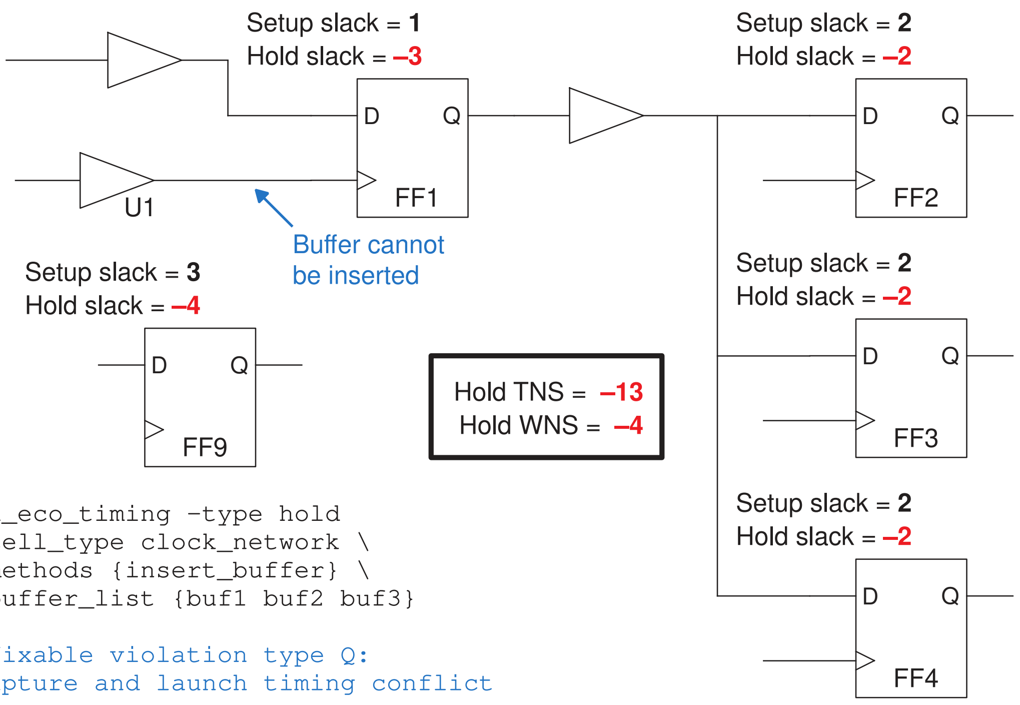
This type of fixing has the following results:

- Total negative slack (TNS) is reduced.
- The slack of some paths may become worse.
- The worst negative slack (WNS) is not worsened, or if the `-wns_limit` option if used, the WNS is no worse than the specified limit.

The following examples demonstrate TNS targeting options:

- Figure 16: Negative hold violations cannot be fixed by default ECO command
- Figure 17: Negative hold fixing targeting TNS, new WNS limited to existing WNS
- Figure 18: Negative hold fixing targeting TNS, new WNS limited to a specific value

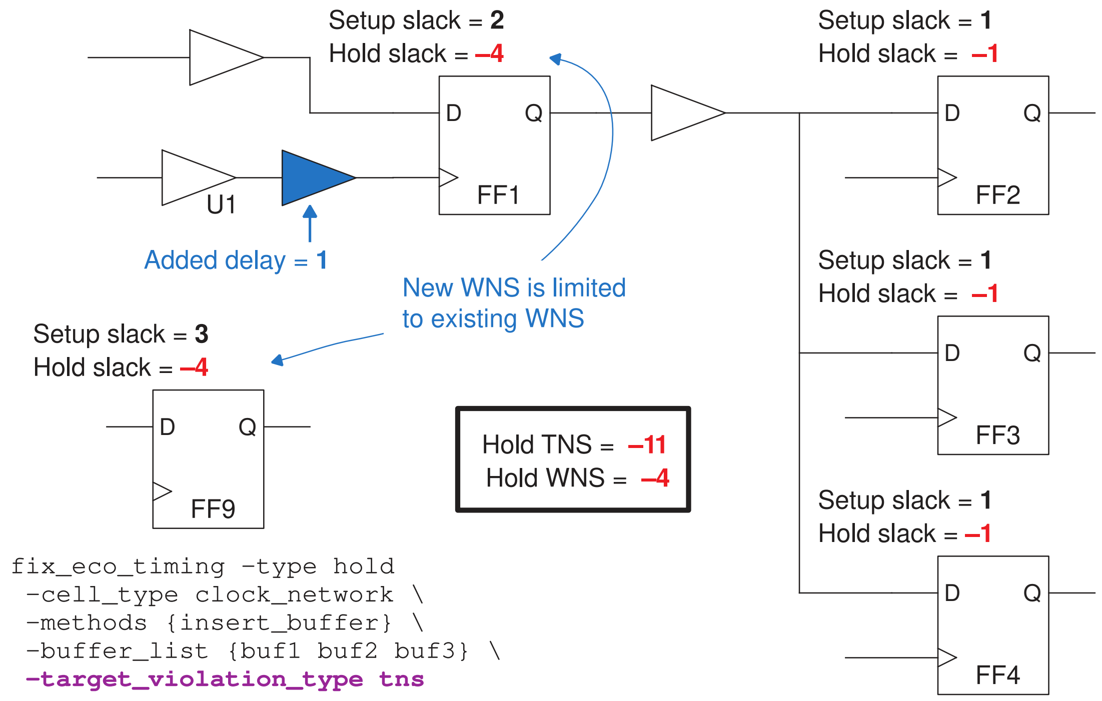
Figure 16 Negative Hold Slack Example



```
fix_eco_timing -type hold
 -cell_type clock_network \
 -methods {insert_buffer} \
 -buffer_list {buf1 buf2 buf3}
```

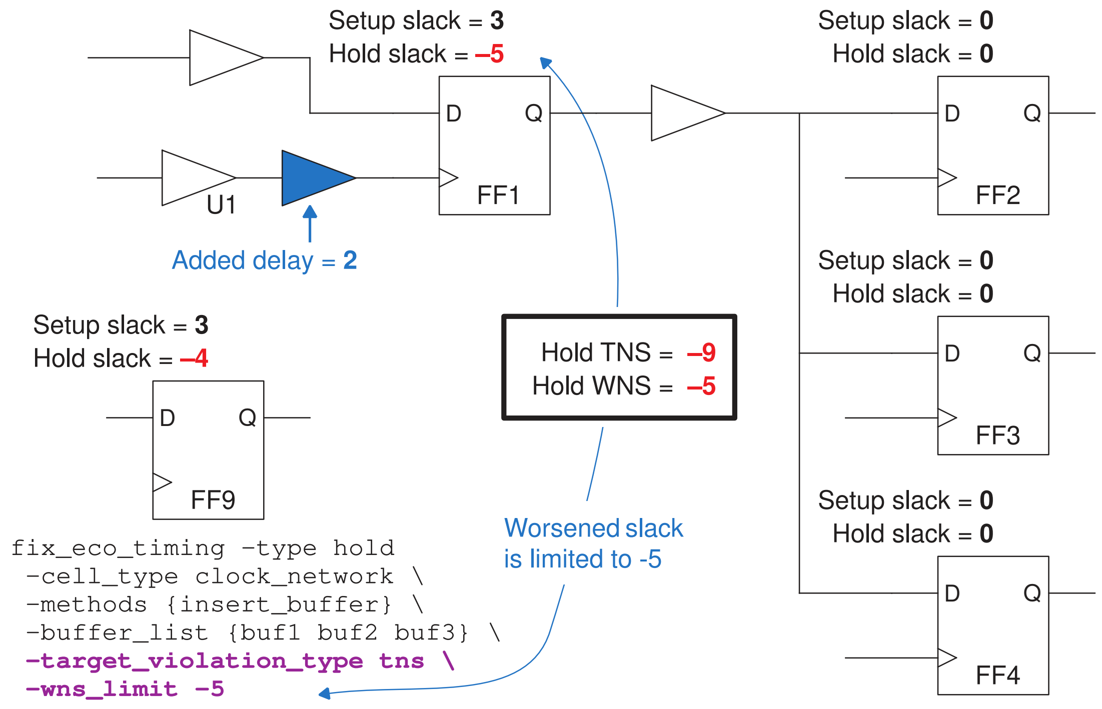
```
Unfixable violation type Q:
 Capture and launch timing conflict
```

Figure 17 TNS-Driven Hold Fixing With Default Slack Degradation Limit



```
fix_eco_timing -type hold
 -cell_type clock_network \
 -methods {insert_buffer} \
 -buffer_list {buf1 buf2 buf3} \
 -target_violation_type tns
```

Figure 18 TNS-Driven Hold Fixing With Explicit Slack Degradation Limit



```
fix_eco_timing -type hold
 -cell_type clock_network \
 -methods {insert_buffer} \
 -buffer_list {buf1 buf2 buf3} \
 -target_violation_type tns \
 -wns_limit -5
```

In Figure 16, the three hold violations at FF2, FF3, and FF4 could be fixed by inserting a single buffer in the clock network at inverter U1. However, this would worsen the hold violation at FF1, which is not allowed by default.

In Figure 17, ECO fixing inserts a buffer with a delay of 1, which improves the TNS from -13 to -11 while allowing the negative slack at FF1 to worsen from -3 to -4. The WNS is not allowed to become any worse than the existing WNS, -4.

In Figure 18, the WNS limit is explicitly set to -5, so ECO fixing inserts a buffer with a delay of 2. This improves the TNS to -9 while allowing the negative slack at FF1 to worsen to -5.

You can set the `-wns_limit` option to a value larger or smaller than the existing WNS. Clock network timing fixing respects the specified limit in either case.

Fixing DRC Violations in Clock Networks

When you choose clock network ECO fixing of DRC violations, the `fix_eco_drc` command fixes maximum transition, maximum capacitance, or maximum fanout violations in the clock network. For example,

```
eco_shell> set_eco_options -physical_enable_clock_data ...
...
eco_shell> fix_eco_drc -type max_transition \
           -methods {size_cell insert_buffer} \
           -buffer_list {buf2 buf4 buf6 buf8} \
           -cell_type clock_network       # DRC fixing in clock paths
...
```

With the `-cell_type clock_network` option, the command fixes maximum transition violations in the clock network by upsizing buffers and inserting buffers. The default behavior is to fix violations only in data paths.

To enable hold fixing for paths that use clock signals as data, set the following variable:

```
eco_shell> set_app_var eco_enable_fixing_clock_used_as_data true
```

Dynamic Power Recovery in Clock Networks

The dynamic power consumption of clock networks is often the largest contributor to total power consumption. Clock network cells toggle every clock cycle, except when gated. Clock network cells closest to the leaf level drive fewer registers, but they consume the most power as a category due to their number.

The PrimeECO tool can specifically target power recovery within clock networks. To do this, specify the `clock_network` cell type when running dynamic power recovery:

```
eco_shell> fix_eco_power -power_mode dynamic -cell_type clock_network
```

Using this feature requires a PrimeECO license.

The `clock_network` cell type targets clock network buffer and inverter cells with high dynamic power consumption, downsizing them to lower internal-power cells when the timing can be preserved in the transitive fanout. In particular, the algorithm targets areas where the clock network is locally over-designed, such that downsizing the cell has less effect on delay and output transition.

The algorithm considers both setup and hold slack at registers in the transitive fanout. Note that a timing change at a clock network cell has different (and opposing) effects on the launched and captured setup and hold paths at the fanout registers.

The following example shows clock network dynamic power recovery:

```
eco_shell> fix_eco_power \
             -verbose -power_mode dynamic -cell_type clock_network \
             -dynamic_scenario $SCENARIO_NAME
Information: Starting dynamic power recovery on clock-network cells at ...

Information: Analyzing power...
Information: Found dynamic power data... (PTECO-089)
Initial cell usage:
Cell Group            Count                      Area
-----------------------------------------------------------
Combinational         16868 ( 97%)         18477.84 ( 89%)
Sequential              288 (  2%)          2022.00 ( 10%)
Clock                   267 (  2%)           361.22 (  2%)
Others                    0 (  0%)             0.00 (  0%)
-----------------------------------------------------------
```

```
Total                 17423 (100%)        20861.06 (100%)

Information: Clock cells will be sized.
Information: Starting iteration 1 at [ date ]...

...

Final ECO Summary:
------------------------------------------------------------
Number of size_cell commands                              99
Initial clock cell area                               361.22
Final clock cell area                                 295.12
Total clock cell area decreased                        66.10
Percentage of clock cell area decreased                18.3%
```

Note the following requirements and limitations:

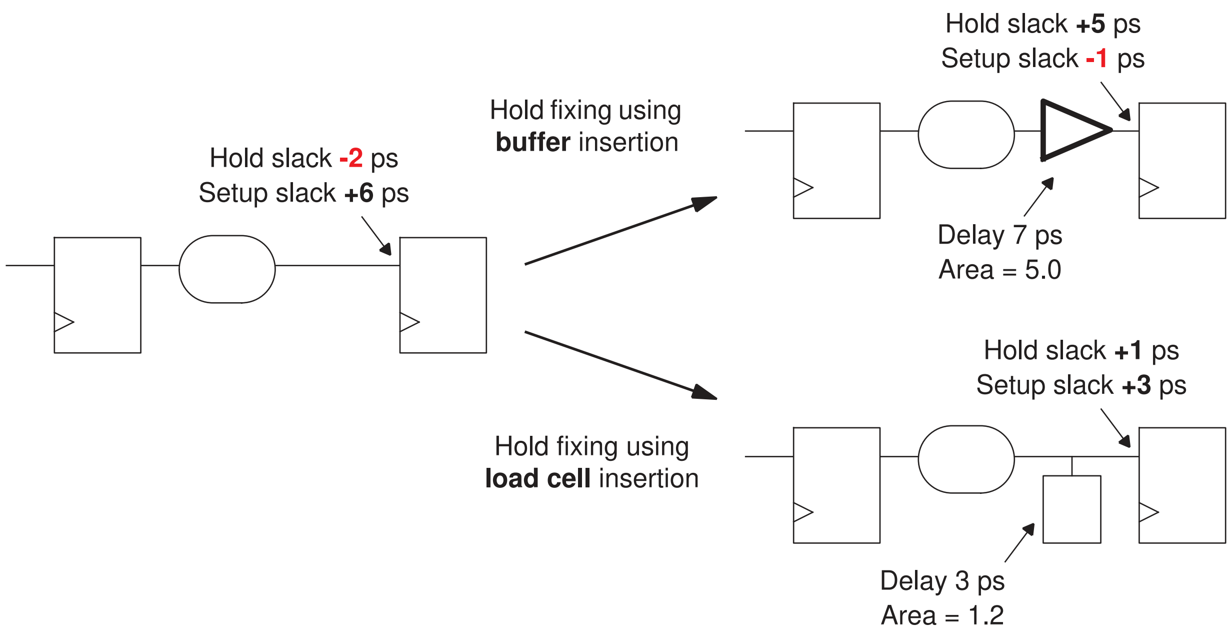
- The power mode must be `dynamic`; other values are not supported.
- The `clock_network` cell type must be targeted independently; it cannot be targeted together with the `combinational` or `sequential` cell types.
- For the `-pba_path` option, only values of `none` and `exhaustive` are supported. If the `path` or `ml_exhaustive` value is specified, the tool issues a message and converts the value internally to `exhaustive`.

ECO Hold Fixing Using Load Capacitance Cells

Hold fixing using the `fix_eco_timing` command has an option to insert load capacitance cells in the data path. Inserting a load cell has the same delay effect as inserting a buffer but offers a better way to fix small hold violations of about 5 ps or less.

In the following figure, the small hold violation of 2 ps can be fixed by inserting the smallest available buffer, which increases the data path delay by 7 ps. However, this over-fixes the hold violation and introduces a setup violation of 1 ps.

Figure 19 Hold Fixing Using Buffer Insertion and Load Cell Insertion



Instead, inserting a load cell that increases the path delay by 3 ps, resulting in a positive hold slack of 1 ps while maintaining a positive setup slack. An additional benefit is the smaller area of the load cell compared to the buffer.

To insert load cells, use the `fix_eco_timing -type hold` command with the `-load_cell_list` option. You can fix small hold violations using load cells and then fix the remaining hold violations using buffers, as shown in the following script:

```
# Fix small hold violations by inserting load cells
fix_eco_timing -type hold -methods insert_buffer -load_cell_list $clist \
 -slack_lesser_than 0.000 -slack_greater_than -0.003

# Fix remaining hold violations by inserting buffers
fix_eco_timing -type hold -methods insert_buffer -buffer_list $bflist
```

Alternatively, you can allow the tool to insert both load cells and buffers in a single operation:

```
# Fix hold violations by inserting load cells and buffers
fix_eco_timing -type hold -methods insert_buffer \
 -load_cell_list $clist -buffer_list $bflist
```

In this case, the tool automatically fixes small hold violations by inserting load cells, and larger hold violations by inserting buffers.

For load cell insertion by the `fix_eco_timing` command, the library must have load cells available. These cells have the following attributes:

Attribute Name	Type	Value
function_id	string	unknown
is_black_box	boolean	true
is_combinational	boolean	true
number_of_pins	int	1

In the physical ECO flow, the `fix_eco_timing` command limits the search area for an available site by considering the `eco_insert_buffer_search_distance_in_site_rows` variable setting, for both buffer insertion and load cell insertion.

Power Recovery

The `fix_eco_power` command tries to recover power and area by downsizing cells in paths with positive setup slack or by removing buffers from paths with positive hold slack, without introducing or worsening timing violations or DRC violations. The command options let you specify any one of the following power recovery methods:

- Replace cells to minimize area (the default)
- Replace cells to minimize a library cell numeric attribute (use the `-power_attribute` option)
- Replace cells based on library cell preference, as specified by an explicit string priority list (use the `-pattern_priority` option)
- Replace cells to minimize switching, leakage, or total power using power analysis data generated by the PrimePower `update_power` command (use the `-power_mode` option)
- Remove buffers from paths with positive hold slack (use the `-methods remove_buffer` option)
- Replace cells in the clock network to specifically minimize dynamic power (use the `-cell_type clock_network` option)

To perform power recovery, use the following procedure.

1. (Optional) Guide power recovery with these controls:
    - Restrict alternative library cells to specific attributes by setting the `eco_alternative_cell_attribute_restrictions` variable
    - Exclude unconstrained cells as candidates for swapping by setting the `eco_power_exclude_unconstrained_cells` variable to `true`

2. (Optional) List the alternative library cells for ECO by using the `report_eco_library_cells` command. For example:

   ```
   eco_shell> report_eco_library_cells

   ****************************************
   Report : eco_library_cells
   ...
   ****************************************
   Alternative library cells:

     Attributes:
         u - dont_use or pt_dont_use
         d - dont_touch

   Group Library_cell                  Area Attributes
   ----------------------------------------------------
   [   0] core_typical/hvt_buf_1       4.50
         core_typical/hvt_buf_2        5.40
         core_typical/hvt_buf_3        6.30
         core_typical/hvt_dly2         8.10 u
   [   1] core_typical/hvt_inv_1       3.60
         core_typical/hvt_inv_2        4.50
         core_typical/hvt_inv_3        5.40
   ```

3. (Optional) Report the cells used in the design with the `report_cell_usage` command. For example:

   ```
   eco_shell> report_cell_usage

   ****************************************
   Report : cell_usage
   ...
   ****************************************
   Cell Group                Count                   Area
   ----------------------------------------------------------
   Combinational       3977357 ( 85%)      5496541.50 ( 22%)
   Sequential           667986 ( 14%)      9514767.00 ( 38%)
   Clock                 42891 (  1%)      2975476.75 ( 12%)
   Others                  973 (  0%)      6873247.50 ( 28%)
   ----------------------------------------------------------
   Total               4689207 (100%)     24860032.00 (100%)
   ```

4. Perform power recovery by using the `fix_eco_power` command:

   ```
   fix_eco_power
     [-methods { ... }]
     [-power_attribute ...]
     [-pattern_priority { ... }]
     [-power_mode ...]
     ...
   ```

5. Report the power recovery results by using the PrimePower `report_power` command. For example:

```
eco_shell> set_app_var power_enable_analysis true
eco_shell> report_power -groups {register combinational sequential}

                     Internal  Switching  Leakage  Total
Power Group          Power     Power      Power    Power(%)      Attrs
----------------------------------------------------------------------
register             2.986e-03 1.733e-03  0.0235   0.0282 (21.23%)
combinational        8.213e-03    0.0374  0.0590   0.1046 (78.77%)
sequential              0.0000    0.0000  0.0000   0.0000 ( 0.00%)
  Net Switching Power = 0.0391   (29.46%)
  Cell Internal Power = 0.0112   ( 8.43%)
  Cell Leakage Power  = 0.0825   (62.12%)
                        -------
Total Power           = 0.1328  (100.00%)
```

   To quantify the power reduction, compare the `report_power` results before and after the using `fix_eco_power` command.

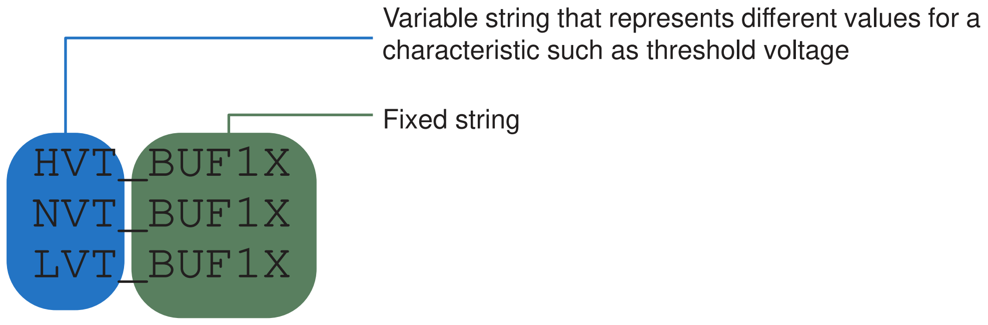
The following topics describe additional power recovery features:

- Power Recovery Based on Library Cell Names
- Power Recovery Based on a Library Cell String Attribute
- Power Recovery Based on a Library Cell Leakage Attribute
- Power Recovery Based on PrimePower Data
- Accelerated Power Recovery Using Machine Learning
- Excluding I/O Paths From Power Recovery

Power Recovery Based on Library Cell Names

For replacing higher-power cells based on the library cell names, the cells must following a naming convention staring with a variable string and ending with a fixed string, as shown in the following example.

Figure 20 Library Cell Naming Example for Power Recovery Cell Swapping


Variable string that represents different values for a characteristic such as threshold voltage
Fixed string
HVT_BUF1X
NVT_BUF1X
LVT_BUF1X

For example, these buffer cell names indicate high, normal, and low threshold voltages.

Library	Library cells
HVT	HVT_BUF1X, HVT_BUF2X, HVT_BUF4X, HVT_BUF8X, HVT_DLY
NVT	NVT_BUF1X, NVT_BUF2X, NVT_BUF4X, NVT_BUF8X, NVT_DLY
LVT	LVT_BUF1X, LVT_BUF2X, LVT_BUF4X, LVT_BUF8X, LVT_DLY

To perform cell replacement based on these library cell names, use the `fix_eco_power` command with the `-pattern_priority` option. For example:

```
fix_eco_power -pattern_priority {HVT NVT LVT}
```

Be sure to list the variable name prefixes in order of priority, lowest-power cells first.

Power Recovery Based on a Library Cell String Attribute

If the library cell names do not follow a naming convention, you can still perform cell swapping based on a user-defined library cell string attribute. For example:

Library cell	Value of user-defined attribute to specify threshold-voltage swap priority
INV1XH	INV1X_best
INV1XN	INV1X_ok
INV1X	INV1X_worst

To perform cell replacement based on a priority list for a user-defined string attribute, use commands like the following:

```
define_user_attribute vt_swap_priority -type string -class lib_cell
set_user_ attribute -class lib_cell lib/INV1XH \
  vt_swap_priority INV1X_best
set_user_attribute -class lib_cell lib/INV1XN \
  vt_swap_priority INV1X_ok
set_user_ attribute -class lib_cell lib/INV1X \
  vt_swap_priority INV1X_worst
...
fix_eco_power -pattern_priority {best ok worst}
  -attribute vt_swap_priority
```

Be sure to list the attribute strings in order of priority, lowest-power cells first.

Power Recovery Based on a Library Cell Leakage Attribute

Leakage power may increase when cells are upsized to improve setup timing. By default, the `fix_eco_timing` command performs optimization to meet timing requirements while minimizing the increase in area, without considering leakage power.

If leakage power is more important than area for setup timing optimization, use the `-power_attribute` option of the `fix_eco_timing` command. Specify the name of a user-defined lib_cell attribute that defines the leakage power for the cell. For example,

```
# Create a user-defined attribute "leak_attr" for lib_cell object
define_user_attribute leak_attr -type float -class lib_cell

# Assign values that reflect leakage of each lib_cell at worst corner
set_user_attribute -class lib_cell \
      [get_lib_cells LIB_H/INV1X_HVT] leak_attr  1.0
set_user_attribute -class lib_cell \
      [get_lib_cells LIB_H/INV2X_HVT] leak_attr  2.0
set_user_attribute -class lib_cell \
      [get_lib_cells LIB_L/INV1X_LVT] leak_attr 10.0
set_user_attribute -class lib_cell \
      [get_lib_cells LIB_L/INV2X_LVT] leak_attr 15.0
...
# Perform setup timing fixing with leakage power consideration
fix_eco_timing -power_attribute leak_attr
```

In this example, the `fix_eco_timing` command chooses library cells that minimize leakage power. It only replaces existing cell instances that have the leak_attr attribute with other library cells that also have the leak_attr attribute; it ignores cells that do not have the attribute.

Instead of assigning the leakage values explicitly, you can import the attribute values from the cell library database. For details, see SolvNet 2371111, “Extracting Leakage Power for Library Cells”; and SolvNet 2040404,”PrimeTime J-2014.12 Update Training,” Module 8: ECO - Power Recovery.

In distributed multi-scenario analysis, the respective library cells in different libraries must be assigned matching leakage power values.

Power Recovery Based on PrimePower Data

The PrimePower tool performs comprehensive power analysis using actual switching activity and library-defined power data such as dynamic and leakage power of each cell. The tool performs a power analysis when you use the `update_power` or `report_power` command at the eco_shell prompt.

To use PrimePower analysis data in power recovery, use the `fix_eco_power` command with the `-power_mode` option set to `dynamic`, `leakage`, or `total`. Then the power recovery process modifies the design to minimize the dynamic (switching) power, leakage power, or total (dynamic plus leakage) power as measured by the `update_power` command.

With the `-power_mode total` option, the tool chooses actions to reduce the total power the most, which can vary with local conditions in the design. For example, where not much timing slack is available, it can choose to recover power by either downsizing the cell or by increasing the cell threshold voltage, the choice depending on the local switching activity. High switching activity favors downsizing (to reduce switching power), whereas low switching activity favors increasing the threshold voltage (to reduce leakage). Where plenty of timing slack is available, it can do both.

Power Recovery ECO Options

The `fix_eco_power` command has the following options to support optimized power recovery:

```
fix_eco_power
  ...
 [-power_mode total | dynamic | leakage]   (Specifies power mode)
 [-dynamic_scenario scenario_name]   (Specifies dynamic power scenario)
 [-leakage_scenario scenario_name]   (Specifies leakage power scenario)
```

The `-power_mode` option specifies the types of power recovery to consider: dynamic power only, leakage power only, or both (`total`). The `-power_mode` option cannot be used with the `-power_attribute` or `-pattern_priority` option.

In distributed multi-scenario analysis (DMSA) flow, the tool gets its dynamic power data from exactly one scenario, which you specify with the `-dynamic_scenario` option. Similarly, it gets its leakage power data from exactly one scenario, which you specify with the `-leakage_scenario` option. These options are used only in a DMSA flow.

In general, you should specify the scenario showing the worst dynamic power and worst leakage power, respectively, for these two options. They could be the same scenario or two different scenarios. You should continue to analyze all scenarios for their timing and DRC constraints, which are honored by the power recovery command.

Single Scenario Power Recovery ECO Example

The following script example demonstrates the ECO power recovery process.

```
restore_session Session1                # Restore analysis session
set_app_var power_enable_analysis true  # Enable PrimePower
set_app_var power_clock_network_include_register_clock_pin_power false
read_saif or read_vcd activity_file     # Read switching activity data
 ...                                    # Set up power analysis
update_power                            # Power analysis
report_power                            # Report power
fix_eco_power -power_mode total         # Dynamic/static power recovery
report_power                            # Report power improvement
```

If you set the `power_clock_network_include_register_clock_pin_power` variable to `false`, the PrimePower tool accounts for the internal power of register cells at the clock input pin as part of the register cell and not as a member of the clock network, so the power recovery is better reflected as part of the data path. The power recovery process can potentially replace register cells to reduce clock pin power (it does not modify the clock network).

Multi-Scenario Power Recovery ECO Example

In the following DMSA example uses six scenarios. Scenario S1 has the most leakage power and scenario S3 has the most dynamic power. To perform total power recovery, use a script similar to the following:

```
current_session {S1 S2 S3 S4 S5 S6}     # Apply timing/DRC constraints
                                           from all scenarios, S1-S6
current_scenario {S1 S3}                # update_power is only needed
 for
                                           power scenarios, S1 and S3
remote_execute {
  set_app_var power_enable_analysis true
  set_app_var power_clock_network_include_register_clock_pin_power false
  read_saif or read_vcd activity_file   # Read switching activity data
   ...
  update_power
}
current_scenario -all
fix_eco_power -power_mode total \
  -leakage_scenario S1 \
  -dynamic_scenario S3
```

Accelerated Power Recovery Using Machine Learning

The PrimeECO tool offers machine learning to speed up power recovery performed by the `fix_eco_power` command. Power recovery can be a computation-intensive process, especially when using path-based analysis, because the tool must analyze the timing, topology, and power in detail to find the optimum cell replacements.

If the machine learning feature is enabled, when the `fix_eco_power` command finishes power recovery, it saves its cell replacement decisions in a file called the training data file. The decision data includes the cell instance replacements, as well as decisions not to replace, and the associated timing and circuit topology conditions that led to the decisions.

The next time you perform power recovery, the `fix_eco_power` command reads the previous training data files, trains the machine learning model, and looks for cells and subcircuits that match the conditions of the earlier sessions that can be provided to the trained model.

Where matching conditions are found, the tool makes the same cell replacement decisions based on the machine learning model, avoiding the time-consuming detailed analysis already performed.

Using this feature requires a PrimeTime-ADV-PLUS license.

Machine Learning Runtime Benefit

The amount of runtime reduction depends on the similarity between the current design and the designs used during training, with respect to timing environment and circuit topology.

For example, you are likely to see a large runtime benefit under the following conditions:

- Same design or similar design type
- Same or similar clock characteristics
- Same or similar layout and parasitic data

On the other hand, you are likely to see little or no runtime benefit under the following conditions:

- Different design type (CPU versus modem; latches versus flip-flops)
- Different clock characteristics (period, uncertainty)
- Different operating conditions (voltage, temperature)
- Different technologies or process nodes

ECO Machine Learning

To enable machine learning for faster power recovery by the `fix_eco_power` command, set the `training_data_directory` variable to the name of a directory. No other user action is needed. For example,

```
eco_shell> set_app_var training_data_directory ./train_dir
```

The `fix_eco_power` command uses the specified directory to retrieve the training data from previous power recovery sessions and to store the learning data from the current session. The `fix_eco_power` command performs these actions automatically when the `training_data_directory` variable is set to a directory name.

The first time you do power recovery with the machine learning enabled, there is no training data available, so power recovery takes the same amount of time as usual. However, upon completion, the `fix_eco_power` command writes out what it has learned to a file in the training directory.

```
eco_shell> read_verilog my_design1.v
...
eco_shell> set_app_var training_data_directory ./train_dir
...
eco_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
...
Training coverage  0.0%
...
Information: Completed
Information: Writing training data train_dir/tr1.td
Information: Elapsed time [ 4280 sec]
```

The next time and subsequent times that you do power recovery, the tool reads in all training data from the directory and applies what it can.

```
eco_shell> read_verilog my_design2.v
...
eco_shell> set_app_var training_data_directory ./train_dir
...
eco_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
Information: Reading training data train_dir/tr1.td
...
Machine Learning Summary:
-------------------------------------------------------
Total number of cells                          10272492
Cells eligible for power recovery               4788129
Trained cells                                   4106599
Adjusted cells                                     2109
Untrained cells                                  679421
Training coverage                                 85.0%
Training adjustment                                0.0%
Retraining scope                                  15.0%
...
Information: Completed
Information: Writing training data train_dir/tr2.td
Information: Elapsed time [ 998 sec]
```

In the “Machine Learning Summary” report:

- `Training coverage` = (Trained cells) / (Cells eligible for power recovery)
- `Training adjustment` = (Adjusted cells) / (Cells eligible for power recovery)
- `Retraining scope` = (Untrained cells) / (Cells eligible for power recovery)

Each usage of the `fix_eco_power` command reads all of the machine learning files from the specified directory and writes out any new learned data to a new file in the directory.

Training Coverage

When you use training data, the `fix_eco_power` command reports the usability of the available data as the “training coverage.” This is the percentage of cells eligible for power recovery that use the learned optimization actions. For example,

```
eco_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
Information: Reading training data train_dir/tr1.td
...
Training coverage              85.0%
...
```

In this example, the training coverage is 85%, which means that 85% of the cells eligible for power recovery matched the timing and topology conditions available from the training data generated by previous runs, and the tool applied the learned optimization actions to those cells. The remaining 15% of these cells did not match previous conditions and were analyzed in detail for power recovery, the same as using no training data.

If you perform power recovery on a design and save the training data, performing another power recovery on the same design under the same timing conditions results in 100% coverage and greatly increased performance, for example, 5X faster. On the other hand, performing power recovery on a very different type of design or a very different clock period may result in 0% coverage and no runtime benefit.

The best runtime benefit comes from training coverage in the 90% to 100% range. You might see very little runtime benefit for training coverage below 50%.

Training Data Files

The `fix_eco_power` command reads training data files from the directory specified by the `training_data_directory` variable and writes out new training data to the same directory.

The file format is binary and you cannot read or edit the contents. However, you can copy and delete these files to modify the training data set. The files are named according to the design from which the data originated and the date and time at which they are generated.

Generally, you should allow the `fix_eco_power` command to read all of the files in the training directory, so that it can find the most suitable data to match the current design. The runtime and memory overhead for reading many files is minor. However, if you wish to read only a subset of the files in the directory, you can do so by using the `-training_data` option of the `fix_eco_power` command.

When to Enable Machine Learning

There is never any harm in enabling machine learning because it can only help the runtime; it cannot hurt. If the `fix_eco_power` command cannot use the available data, it simply ignores that data and performs power recovery using normal detailed analysis, and it still stores what it learns in the current session for possible use in future sessions.

The runtime overhead for reading and saving training data is considered negligible. Therefore, it is recommended that you leave the feature enabled every time you use the `fix_eco_power` command, so you can gather data from the widest possible range of design styles, timing conditions, and topologies. The more data you have, the more likely that the data will be useful in future power recovery sessions.

Using the machine learning feature reduces runtime, with little effect on quality of results. You can expect to see similar power recovery results whether or not you use the feature.

Comprehensive Training Data Coverage

If you generate training data to cover all stages of your design cycle, you can gain the benefit of machine learning all the time in future sessions. For example, you can run one or more setup, hold, and DRC fixing operations in parallel to reach different states of the design, and run the `fix_eco_power` command to create training data from each of these stages:

```
# Initial run with DRC, setup, and hold fixing
...
update_timing
save_session after_update      # save session after timing update
fix_eco_drc
save_session after_drc_fix     # save session after DRC fixing
fix_eco_timing -type setup
save_session after_setup_fix   # save session after setup fixing
fix_eco_timing -type hold
save_session after_hold_fix    # save session after hold fixing

# Training Run 1 on host 1
set_app_var training_data_directory ./all_training_data
restore_session after_update
fix_eco_power

# Training Run 2 on host 2
set_app_var training_data_directory ./all_training_data
restore_session after_drc_fix
fix_eco_power

# Training Run 3 on host 3
set_app_var training_data_directory ./all_training_data
restore_session after_setup_fix
fix_eco_power

# Training Run 4 on host 4
```

```
set_app_var training_data_directory ./all_training_data
restore_session after_hold_fix
fix_eco_power
```

After all four runs finish, the training data directory contains four different sets of training data created from different stages of the design cycle. When you start working on a new revision of the design, you are likely to get high training data coverage due to matching of current timing and topology conditions with one or more of the previous training sessions, resulting in greatly reduced runtime during power optimization.

User-Scripted ECO Machine Learning

If you have created custom scripts to perform power optimization, you can direct the tool to learn from these strategies and apply them in future execution of the `fix_eco_power` command. To do this, use the `record_training_data` and `write_training_data` commands as shown in the following example.

```
# Machine learning session
...
record_training_data                    # Start recording training data
...
source my_eco_changes.tcl               # User’s power optimization script
...
write_training_data -output my_training.td     # Generate training data
...
```

In a later session, the `fix_eco_power` command uses the training data when you specify the training data directory with the `-training_data` option.

```
# Machine learning usage
...
fix_eco_power -training_data my_training.td   # Use training data
...
```

In this machine learning flow, the PrimeECO tool does not check or evaluate the quality of the power optimization script. It simply accepts the changes made by the user script as beneficial and saves the timing and topological conditions surrounding the changes performed by the script. It writes the training data to the specified file.

When you later use the `fix_eco_power` command with the `-training_data` option, the command reads in the training data and applies the changes where it finds cells with similar timing and topological conditions, without performing a detailed analysis. Where there is no match with the learned timing and topological conditions, the command performs normal power optimization using detailed analysis, the same as using no machine learning.

Excluding I/O Paths From Power Recovery

The `fix_eco_power` command can perform power recovery on internal (register-to-register) paths only, while leaving I/O paths untouched.

To use this feature,

- Set the -start_end_type option to reg_to_reg.
- Set the -pba_mode option to path or exhaustive. This is required.

For example,

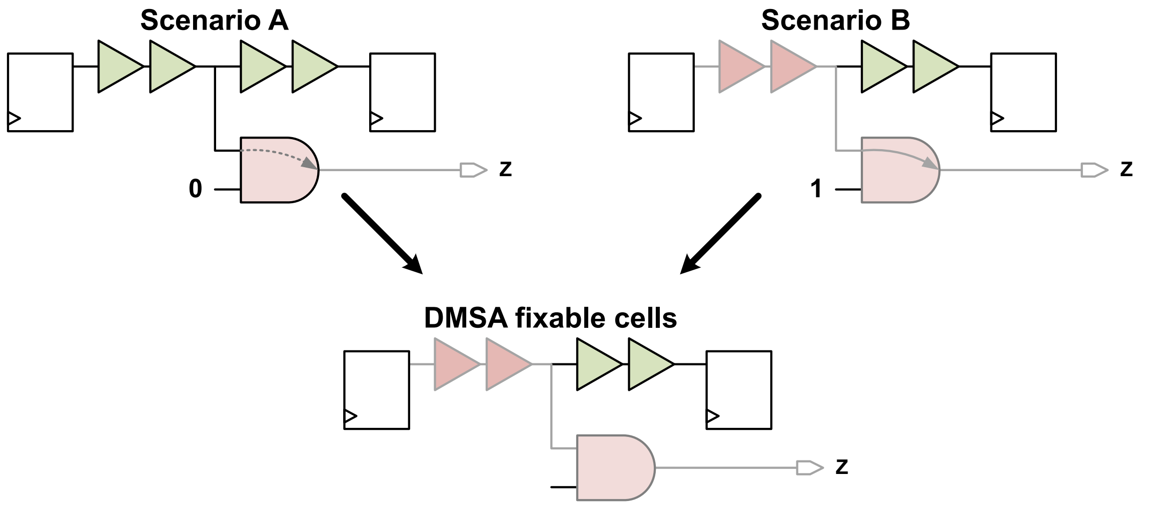
```
eco_shell> fix_eco_power \
            -start_end_type reg_to_reg \
            -pba_mode exhaustive
```

The -start_end_type option also exists on the report_timing and get_timing_paths commands. However, the only valid value for the fix_eco_power command is reg_to_reg.

For power reduction, a cell is considered “internal” if there are no paths, constrained or unconstrained, that reach the cell from an I/O port. Case analysis and disabled timing arcs are considered during the check.

In a DMSA analysis, the cell is excluded if it reaches an I/O port in any scenario, as shown in Figure 21.

Figure 21 Merging the Set of Unfixable I/O Cells in DMSA fix_eco_power



Cells that reach I/O ports are left untouched, even if their slack is less than the setup or hold margin used for fixing. They are reported with reason code “V” in the unfixable reasons report.

ECOs With Multiply Instantiated Modules (MIMs)

When you set the `eco_enable_mim` variable to `true`, the tool performs the same ECO changes across each set of MIMs and writes out a single change list file for each set of MIMs.

In physical ECO fixing, the tool recognizes a set of MIM instances when they share the same DEF file; in logic-only fixing, it recognizes them when they share the same parasitics file according to the `-path` option in the `read_parasitics` command.

To use multiply instantiated modules in the ECO flow, follow these steps:

1. Set the `eco_enable_mim` variable:

   ```
   eco_shell> set_app_var eco_enable_mim true
   ```

2. (Optional) Define groups of multiply instantiated modules by using the `set_eco_options` command with the `-mim_group` option. To specify multiple groups, use the command multiple times:

   ```
   eco_shell> set_eco_options -mim_group {CPU1 CPU2}
   eco_shell> set_eco_options -mim_group {CPU3 CPU4}
   ```

   Note:

   Using the `-mim_group` option might use more runtime and memory during ECO because it separately analyzes each MIM group.

   To report the defined MIM groups, use the `report_eco_options` command:

   ```
   eco_shell> report_eco_options
   ...
   Instance groups:
   Group Module Instances
   ---------------------------
   1     CPU     CPU1 CPU2
   2     CPU     CPU3 CPU4
   ```

3. Perform ECO fixing by using the `fix_eco_drc`, `fix_eco_timing`, and `fix_eco_power` commands.

HyperTrace ECO Fixing

HyperTrace is a technology that accelerates path-based analysis (PBA). The PrimeECO tool can use this technology to speed up ECO fixing when the `-pba_mode` option of the `fix_eco_timing` or `fix_eco_power` command is used. Graph refinement data is automatically updated during the ECO fixing process.

This feature requires a PrimeECO license.

The following topics provide more information:

- HyperTrace ECO Fixing With fix_eco_timing
- HyperTrace ECO Fixing With fix_eco_power

HyperTrace ECO Fixing With fix_eco_timing

The `fix_eco_timing` command supports HyperTrace ECO fixing for the following path-based analysis modes:

- `-pba_mode exhaustive`
- `-pba_mode ml_exhaustive`

Internally, the `fix_eco_timing` command calls HyperTrace reporting to accelerate the exhaustive PBA search. As a result, the graph refinement variables used by HyperTrace reporting are also used during ECO fixing:

```
timing_refinement_max_slack_threshold
timing_refinement_min_slack_threshold
timing_refinement_maximum_critical_pin_percentage
```

In particular, you must ensure that the refinement slack thresholds bound the implicit or explicit `-slack_lesser_than` value of the `fix_eco_timing` command.

To use HyperTrace ECO fixing with the `fix_eco_timing` command,

1. Enable HyperTrace ECO fixing:

   ```
   eco_shell> set_app_var eco_enable_graph_based_refinement true
   ```

   Note that this variable setting (for HyperTrace ECO fixing) is independent from the `timing_enable_graph_based_refinement` variable (for HyperTrace reporting).

2. If you are fixing setup violations to a slack value other than zero (positive or negative), set the HyperTrace max-delay slack threshold to your target slack value:

   ```
   eco_shell> # the default max-delay threshold is 0
   eco_shell> set_app_var timing_refinement_max_slack_threshold 0.05
   ...
   eco_shell> fix_eco_timing -type setup -slack_lesser_than 0.05
   ```

3. (Optional) If you are fixing hold violations, note that HyperTrace is disabled for min-delay paths by default because hold violations are typically short paths that do not benefit from HyperTrace acceleration.

If your design has complex violating min-delay logic cones (such as I/O or memory interfaces) that benefit from HyperTrace acceleration, set the min-delay slack threshold to your target slack value:

```
eco_shell> # the default min-delay threshold is 'disabled'
eco_shell> set_app_var timing_refinement_min_slack_threshold 0
...
eco_shell> fix_eco_timing -type hold
```

4. (Optional) Look for the following message to confirm that HyperTrace ECO fixing is being used:

```
Information: Using HyperTrace to accelerate PBA-based ECO fixing.
(PTECO-116)
```

HyperTrace ECO Fixing With fix_eco_power

The `fix_eco_power` command supports HyperTrace ECO fixing for the following path-based analysis modes:

- `-pba_mode path`
- `-pba_mode exhaustive`
- `-pba_mode ml_exhaustive`

The `fix_eco_power` command incorporates HyperTrace algorithms and data directly into its ECO fixing algorithms. As a result, the graph refinement variables used by HyperTrace reporting are not used or needed:

```
timing_refinement_max_slack_threshold
timing_refinement_min_slack_threshold
timing_refinement_maximum_critical_pin_percentage
```

To use HyperTrace ECO fixing with the `fix_eco_power` command,

1. Enable HyperTrace ECO fixing:

```
eco_shell> set_app_var eco_enable_graph_based_refinement true
```

Note that this variable setting (for HyperTrace ECO fixing) is independent from the `timing_enable_graph_based_refinement` variable (for HyperTrace reporting).

2. Run one of the following ECO fixing commands:

   ```
   fix_eco_power -pba_mode path ...
   fix_eco_power -pba_mode exhaustive ...
   fix_eco_power -pba_mode ml_exhaustive ...
   ```

3. (Optional) Look for the following message to confirm that HyperTrace ECO fixing is being used:

   ```
   Information: Running HyperTrace-based ECO fixing (PTECO-115)
   ```

The following options of the `fix_eco_power` command are unsupported for HyperTrace ECO fixing:

```
fix_eco_power -start_end_type reg_to_reg
fix_eco_power -pba_path_selection_options
```

Reporting Unfixable Violations and Unusable Cells

When you perform ECO timing or DRC fixing, you can choose to report information about why violations could not be fixed. Similarly, for ECO power fixing, you can choose to report information about why cells could not be resized for power reduction.

These features are described in the following topics:

- Reporting Unfixable Timing and DRC Violations
- Reporting Unusable Cells for Power Reduction

Reporting Unfixable Timing and DRC Violations

The `fix_eco_timing` and `fix_eco_drc` commands offer an “unfixable violations” reason feature that reports why a violation could not be fixed.

To use this feature, first set the `eco_report_unfixed_reason_max_endpoints` variable to the number of unfixable violations to report:

```
eco_shell> set_app_var eco_report_unfixed_reason_max_endpoints 50
```

You can then include the reasons in the command output, write them to a file, or estimate the reasons without fixing (for timing ECO fixing only):

Table 5 Reporting Unfixable Timing and DRC Violations

To do this	Use this command
Estimate the unfixable violations without fixing (`fix_eco_timing` only)	`fix_eco_timing -estimate_unfixable_reasons`

Table 5 Reporting Unfixable Timing and DRC Violations (Continued)

To do this	Use this command

Run fixing, and include the unfixable violations in the command output	`fix_eco_timing | fix_eco_drc`
	`-verbose`

Run fixing, and write the unfixable violations to a log file	`fix_eco_timing | fix_eco_drc`
	`-unfixable_reasons_prefix file_name_prefix`
	`[-unfixable_reasons_format text | csv]`

Run fixing, include the unfixable reasons in the output, and write them to a log file	`fix_eco_timing | fix_eco_drc`
	`-verbose`
	`-unfixable_reasons_prefix file_name_prefix`
	`[-unfixable_reasons_format text | csv]`

For example,

```
eco_shell> set_app_var eco_report_unfixed_reason_max_endpoints 50
50
eco_shell> fix_eco_timing -type setup -estimate_unfixable_reasons
Information: Using option -estimate_unfixable_reasons. ...
Information: The design will not be changed.
Information: 22 violating endpoints located... (PTECO-022)
Information: 22 endpoints are being considered for fixing... (PTECO-027)
 ...
Unfixable violations:
  A - There are available library cells outside area limit
  B - Delay improvement is too small to fix the violation
  C - The violation is in clock network
  I - Buffer insertion with given library cells cannot fix the violation
  S - Cell sizing with alternative library cells cannot fix the violation
  T - Timing margin is too tight to fix the violation
  U - UPF restricts fixing the violation
  V - Net or cell is invalid or has dont_touch attribute
  W - Fixing the violation might degrade DRC violations

  Violation                                          Reasons  Prio/slk
  -----------------------------------------------------------------
  S:I_ORCA_TOP/I_PCI_CORE/pad_en_reg/Q
    U7/ZN                                              A B W        P8
    U62/ZN                                             A B W        P8
    U63/Z                                              A B W        P8
    U63ASTipoInst495/Z                                 A B W        P5
    U63ASTipoInst494/Z                                 A B W        P3
    pad_iopad_1/PAD                                      B W        P9
  E:pad[1]                                                      -1.157
  ...
  C:U54/Z
    pc_be_iopad_1/PAD                                    B W        P9
  E:pc_be[1]                                                    -0.017
```

```
1
Final ECO Summary:
-------------------------------------------------------
Number of size_cell commands                        276
Total number of commands                            276
Area increased by cell sizing                    128.50
Total area increased                             128.50

Information: Elapsed time ...
Information: Completed at ...
```

For detailed information about unfixable violation reports, see the man page for the `fix_eco_timing` or `fix_eco_drc` command.

For timing ECO fixing, the estimation report can guide you in fine-tuning the command options without actually performing ECO changes. This report provides guidance on how to address the unfixable violations. For example, the report can help you decide between modifying the buffer list, relaxing area constraints, or relaxing margins before you perform actual fixing. Generating the report is faster than actual fixing.

Estimation of unfixable violation reasons is compatible with all the other `fix_eco_timing` command options. For example, to restrict the report to specific paths, specify those paths using `-from` or `-to`, as in the following example.

```
eco_shell> fix_eco_timing -type setup -estimate_unfixable_reasons \
           -from I_ORCA_TOP/I_PCI_CORE/pad_en_reg/Q -to pad[1]
```

Reporting Unusable Cells for Power Reduction

The `fix_eco_power` command offers an “unusable cells” reason feature that reports why a cell could not be downsized for power reduction.

The `fix_eco_timing` and `fix_eco_drc` commands have a similar “unfixable reasons” feature. However, there is no concept of a violation with power reduction, so the terminology is somewhat different for the `fix_eco_power` command.

To use this feature, first set the `eco_power_report_max_unusable_cells` variable to the number of unusable violations to report:

```
eco_shell> set_app_var eco_power_report_max_unusable_cells 100
```

You can then include the reasons in the command output, write them to a file, or both:

Table 6 Reporting Unusable Cells for Power Reduction

To do this	Use this command
Run recovery, and include the unusable reasons in the command output	`fix_eco_power -verbose`

Table 6 Reporting Unusable Cells for Power Reduction (Continued)

To do this	Use this command
Run recovery, and write the unusable violations to a log file	`fix_eco_power -unusable_reasons_prefix file_name_prefix`
Run recovery, include the unusable reasons in the output, and write them to a log file	`fix_eco_power -verbose -unusable_reasons_prefix file_name_prefix`

For example,

```
eco_shell> set_app_var eco_power_report_max_unusable_cells 8
8
eco_shell> fix_eco_power -verbose
...

Unusable Cells:
  B - Power benefit is too small
  L - Physical constraints restrict sizing
  S - Cell has no alternate library cell with better power
  T - Sizing cell might degrade timing
  U - UPF restrictions prevent sizing
  V - Cell is invalid or has dont_touch attribute
  W - Sizing cell might degrade DRC
  X - Cell is unusable for ECO
  Z - Cell is sized

  Cell name                         Lib_cell name             Reasons
  -------------------------------------------------------------------
  ff1                                 SDFFARX1_HVT                  W
  ff2                                 SDFFARX2_HVT                  T
  ff3                                 SDFFARX1_HVT                  U
  ff4                                 SDFFARX2_HVT                  L
  U1676                                 AO22X1_HVT                X V
  U1674                                 OA21X1_HVT                  S
  U1655                                 AO22X1_HVT                  W
  U1654                                 OA21X1_HVT              T W Z

  Unusable cells summary:
  ------------------------------------------------------------
    Total number of cells checked:                          8
    Total number of cells with reason code S :              1
    Total number of cells with reason code L :              1
    Total number of cells with reason code T :              2
    Total number of cells with reason code U :              1
    Total number of cells with reason code V :              1
    Total number of cells with reason code W :              3
    Total number of cells with reason code X :              1
    Total number of cells with reason code Z :              1
```

The reported cells are ordered by (and thus truncated to the limit by) the type of power fixing being performed:

Power fixing type	Ordering and truncation method
Area-based downsizing	In order of `[get_cells -hier *]`
Leakage swapping	In order of pattern priority (lowest to highest), with the name as a tiebreaker
Attribute-based downsizing	By decreasing order of library cell attribute value, with the name as the tiebreaker
Power-mode downsizing	By decreasing order of instance cell power In total power mode for DMSA, ordering is determined by the `-dynamic_scenario` scenario.

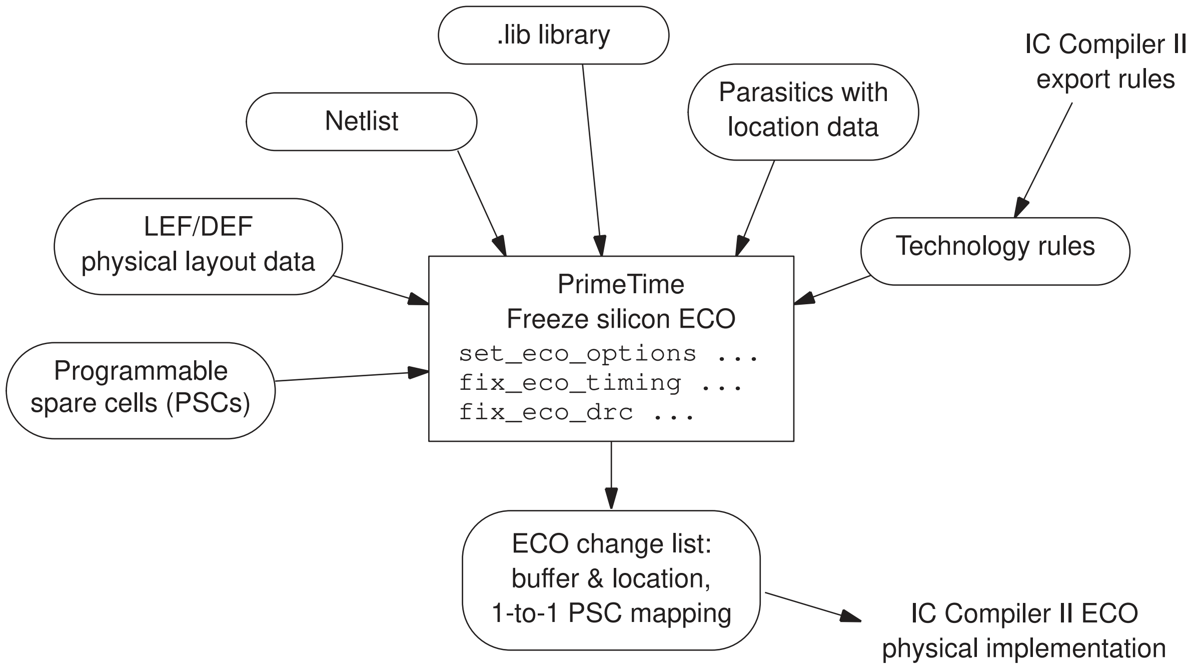
Freeze Silicon ECO Flow

In the freeze silicon flow, the IC Compiler II tool implements ECOs by modifying only the metal and via layers, leaving the silicon-level layers unchanged, thereby saving the time and expense of generating new masks for the implant, diffusion, and poly layers. This flow requires the placement of spare cells throughout the chip layout, allowing the IC Compiler II tool to implement ECOs by connecting the spare cells into the design using metal routes.

The following diagram shows the types of data typically used in the freeze silicon flow.

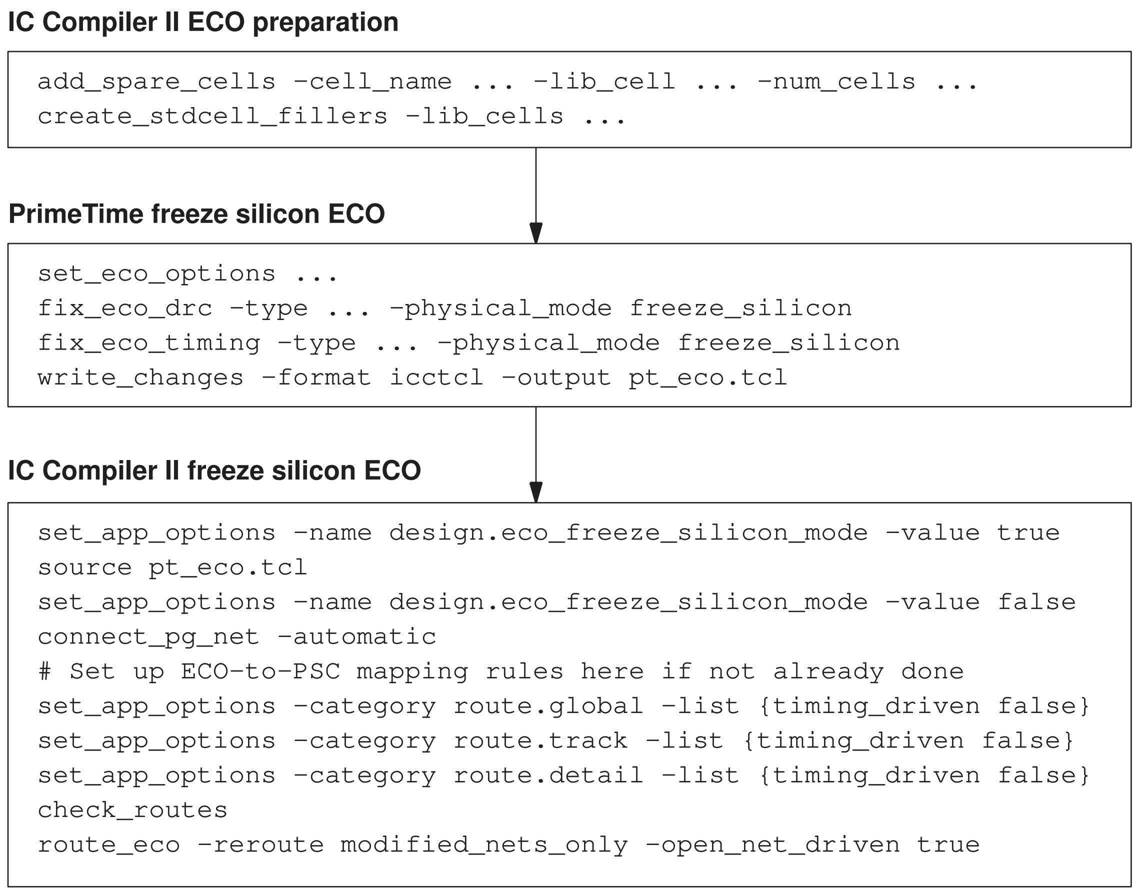
Feedback

Figure 22 Freeze Silicon Data Flow


.lib library
Netlist
Parasitics with location data
IC Compiler II export rules
LEF/DEF physical layout data
Technology rules
PrimeTime
Freeze silicon ECO
set_eco_options ...
fix_eco_timing ...
fix_eco_drc ...
Programmable spare cells (PSCs)
ECO change list: buffer & location, 1-to-1 PSC mapping
IC Compiler II ECO physical implementation

The following figure shows the commands typically used in the IC Compiler II and PrimeTime tools in the freeze silicon flow.

Figure 23 IC Compiler II and PrimeTime Commands in the Freeze Silicon Flow



IC Compiler II ECO preparation

```
add_spare_cells -cell_name ... -lib_cell ... -num_cells ...
create_stdcell_fillers -lib_cells ...
```

PrimeTime freeze silicon ECO

```
set_eco_options ...
fix_eco_drc -type ... -physical_mode freeze_silicon
fix_eco_timing -type ... -physical_mode freeze_silicon
write_changes -format icctcl -output pt_eco.tcl
```

IC Compiler II freeze silicon ECO

```
set_app_options -name design.eco_freeze_silicon_mode -value true
source pt_eco.tcl
set_app_options -name design.eco_freeze_silicon_mode -value false
connect_pg_net -automatic
# Set up ECO-to-PSC mapping rules here if not already done
set_app_options -category route.global -list {timing_driven false}
set_app_options -category route.track -list {timing_driven false}
set_app_options -category route.detail -list {timing_driven false}
check_routes
route_eco -reroute modified_nets_only -open_net_driven true
```

Preparing for the Freeze Silicon ECO Flow

The following requirements apply to the freeze silicon ECO flow:

- Spare cells must be available in the chip layout. For details, see “ECO Flow” and “Freeze Silicon ECO Flow” in the *IC Compiler II Implementation User Guide*.
- The spare cells must be defined in the .lib library.
- The library-based and instance-specific layout properties of the spare cells must be available in a set of LEF/DEF files, which you can generate from the IC Compiler II tool.

- The applicable spacing rules used for cell placement must be available in a file. You can provide this file in any of the following ways:
    - In the IC Compiler II tool, use the `export_advanced_technology_rules` command, which writes out the technology rules in a binary encrypted format.
    - Write a script file containing a set of `set_lib_cell_spacing_label` and `set_spacing_label_rule` commands that define the rules.
    - Provide the rules in Synopsys technology file format, as described in the *Synopsys Technology File and Routing Rules Reference Manual*.

In the PrimeTime tool, use the `set_eco_options` command to choose the freeze silicon ECO flow, specify the spare cell names, specify the spacing-rule constraint file, and specify the physical design (LEF/DEF) files. For example,

```
eco_shell> set_eco_options \
 -programmable_spare_cell_names {PSC1 PSC2 PSC3 PSC4 PSC5 PSC6} \
 -physical_lib_constraint_file ../tech_data/my_spacing_rules \
 -physical_tech_lib_path {../phy_data/my_tech.lef} \
 -physical_lib_path {../phy_data/my_lib.lef ../phy_data/my_design.def} \
 -log_file my_lef_def.log \
 ...
```

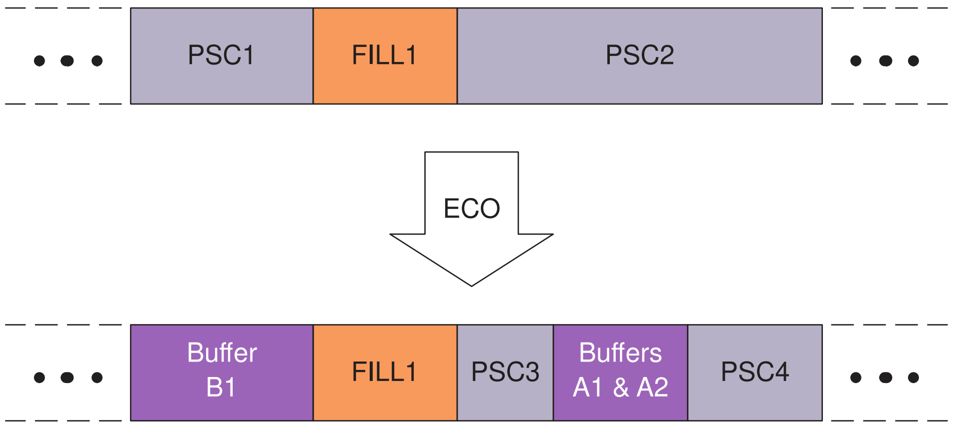
The `-log_file` option causes the tool to write out LEF/DEF processing messages (errors and warnings) into the specified log file.

Programmable Spare Cells

The freeze silicon ECO flow supports the use of *programmable spare cells* (PSCs). Instead of performing only a single fixed logic function, a PSC can be programmed to perform any of various logic functions. The cell is made up of transistor-level parts that can be “wired up” in a variety of ways, allowing great flexibility in implementing ECO changes. If a particular ECO uses only part of a PSC, the remaining part of the cell is still available for later ECOs.

The following figure shows how an ECO replaces spare cells with functional cells. At the top, you can see part of a standard-cell row in the chip layout after cell placement, as displayed in the IC Compiler II tool, before applying an ECO. The boxes labeled PSC1 and PSC2 are unconnected PSC-type spare cells. Because this is a freeze silicon flow, no cells can be moved, but the PSC cells can be wired to perform different logical functions.

Figure 24 Freeze Silicon Site Row Cell Replacement



As a result of applying an ECO, the PSC1 spare cell is replaced entirely by a large buffer, B1. The space originally occupied by the PSC2 spare cell is used to implement two buffers, A1 and A2. Only part of this space is used, so the tool backfills the remaining available parts of the original PSC2 spare cell with two new spare cells, PSC3 and PSC4, which are available to be used in later ECOs.

LEF/DEF Descriptions of PSCs

The library-based and instance-specific layout properties of the spare cells must be available in a set of LEF/DEF files, which you can generate from the IC Compiler II tool by using the `write_lef` and `write_def` commands.

The description of a PSC in a LEF file is similar to the description for an ordinary fill cell. However, for ordinary fill cells, the `MACRO` section must contain `CLASS CORE FEEDTHRU` or `CLASS CORE SPACER`; this is optional for PSCs. For example,

```
# LEF Programmable Spare Cell construct, cell name = e0hd_cap12x
MACRO  e0hd_cap12x
 CLASS CORE ;
 ORIGIN 0 0 ;
 SIZE 0.816 BY 0.42 ;
 ...
END  e0hd_cap12x
```

In the description of a PSC instance in a DEF file, the `COMPONENTS` section must contain `SOURCE DIST` (because the instance is not defined in the netlist) and `PLACED` (because it has a physical location); and it cannot contain `FIXED` or `COVER`. For example,

```
# DEF instantiation of Programmable Spare Cell
-xofiller!ECO_DECAP_FILLER_!e0hd_cap12x!179638
e0hd_cap12x + SOURCE DIST + PLACED (264300 624100) N ;
```

You can check the validity of LEF/DEF descriptions of PSCs by using the `-log_file` option of the `set_eco_options` command. For example,

```
set_eco_options \
  -programmable_spare_cell_names {PSC1 PSC2 PSC3 PSC4 PSC5 PSC6} \
  -physical_lib_constraint_file my_spacing_rules \
  -physical_tech_lib_path {../phy_data/my_tech.lef} \
  -physical_lib_path {my_lib.lef my_design.def} \
  -log_file lef_def_info.log
fix_eco_timing -type hold -physical_mode freeze_silicon \
  -buffer_list $hold_buffer_list
```

After you run the `fix_eco_timing` command, the lef_def_info.log file shows PSC identification messages:

```
...
Information: Reading LEF file my_lib.lef
Information: Information: Identified programmable spare cell PSC1 at
 lineNumber 1238.
Information: Information: Identified programmable spare cell PSC2 at
 lineNumber 1279.
...
Information: Reading DEF file  my_design.def
Physical Information Summary:
Identified programmable spare cells PSC1 ...
...
```

Exporting DEF and Advanced Spacing Rules From IC Compiler II

To generate the required DEF files and technology spacing rules from the IC Compiler II tool, you can use the `write_def` and `export_advanced_technology_rules` commands. At the icc2_shell prompt, execute a script similar to the following:

```
set myblocks "block1 block2 block3"
set mylib "design_A"
open_lib $mylib                     # include all required libs
foreach blockx $myblocks {
 open_block $blockx                 # open the design
 check_legality
 link -force
 write_def $blockx.def                            # write block DEF file
 export_advanced_technology_rules $blockx_rules.tcl     # and tech rules
 close_blocks $blockx
}
```

Freeze Silicon ECO in PrimeTime

To perform a freeze silicon ECO in the PrimeTime tool, use the following commands:

- `set_eco_options` to specify the ECO parameters
- `fix_eco_timing` or `fix_eco_drc` command to generate the ECO changes

The following script performs freeze silicon ECO hold fixing in multiple scenarios.

```
# Create scenarios
...
set_app_var read_parasitics_load_locations true
# Update timing with pin slack and arrival info
remote_execute {
 set_app_var timing_save_pin_arrival_and_slack true;
 update_timing -full }

# Configure Physical Information needed for ECO
remote_execute { set lef_files [ glob ./*.lef]
set_eco_options \
 -physical_tech_lib_path $tech_lef_files \
 -physical_lib_path $lef_files \
 -physical_design_path ./design.def.gz \
 -physical_lib_constraint_file $rule_files \
 -programmable_spare_cell_names $pscs \
 -log_file ./lef_def.log }

# Alternative to LEF/DEF: direct access to IC Compiler II data
# set_eco_options \
#   -physical_icc2_lib $icc2_lib_path \
#   -physical_icc2_blocks $icc2_blocks \
#   ...
# Fix hold violations
set buffer_list { BUF_1 BUF_2 ... BUF_N}
set_app_var eco_report_unfixed_reason_max_endpoints 50
fix_eco_timing -type hold -physical_mode freeze_silicon \
 -buffer_list $buffer_list -verbose

write_changes -format icctcl -output pt_eco.tcl
```

Automated ECO implementation is not supported in the freeze silicon ECO flow. Instead, use the `write_changes` command to write out the ECO change list for IC Compiler II. For details, see the freeze silicon ECO flow documentation in the “ECO Flow” chapter of the *PrimeTime User Guide*.

Manual Netlist Editing

You can manually edit the netlist and analyze how the changes affect the timing. This can be useful late in a design cycle when only small edits can be made. However, if you need to fix many violations or make substantial changes, you should use PrimeECO fixing commands.

To manually edit the netlist, follow these steps:

1. If you need to replace or resize cells, find alternative library cells by running the `get_alternative_lib_cells -base_names` command:

   ```
   eco_shell> get_alternative_lib_cells -base_names U135
   AN2 AN2i
   ```

2. Edit the netlist by using one or more of these commands.

Table 7 Netlist Editing Commands

Object	Task	Command
Buffer	Insert a buffer Remove a buffer	`insert_buffer` `remove_buffer`

Cell	Change the size (or drive strength) of a cell
	Create a cell
	Rename a cell
	Remove a cell
		`size_cell`
		`create_cell`
		`rename_cell`
		`remove_cell`

Net	Add a new net
	Connect nets to pins
	Disconnect nets from pins
	Rename a net
	Remove a net
		`create_net`
		`connect_net`
		`disconnect_net`
		`rename_net`
		`remove_net`

3. Generate a change list file by using the `write_changes` command.

"What-If" Incremental Analysis

For a faster manual ECO fixing flow, you can perform "what-if" analysis of certain design changes entirely within PrimeECO. For analyzing these changes, PrimeECO uses a fast *incremental* analysis, which takes just a fraction of the time needed for a full analysis because it updates only the portion of the design affected by the changes.

To fix violations, you can increase the sizes of the driving cells with the `size_cell` command or insert buffers with the `insert_buffer` command. To fix crosstalk violations, you can separate adjacent victim and aggressor nets with the `set_coupling_separation` command. You can also perform other custom netlist operations such as removing the cell and reconnecting a new cell with the `remove_cell`, `create_cell`, and `connect_net` commands.

During incremental analysis, you can use these commands:

- `size_cell`
- `insert_buffer`, `remove_buffer`
- `set_coupling_separation`, `remove_coupling_separation`
- `connect_net`, `disconnect_net`, `remove_net`
- `create_cell`, `remove_cell`

If you use other netlist editing commands, the `update_timing` command performs a full (not incremental) timing update.

For an incremental update, PrimeECO does the following:

1. Identifies the nets that are directly affected by the "what-if" changes, and their aggressors.
2. Performs electrical filtering of the affected nets.
3. Performs the first update iteration on the fanout cone of the affected nets, with the depth of the update cone limited to changes in slew.
4. Performs the second and any subsequent iterations on the nets in the affected cone.

After you decide on a set of changes, you can generate a change list file for the physical implementation tool.

The results of a "what-if" crosstalk analysis can be slightly different from a full analysis because of the iterative nature of the analysis and the interdependence of timing windows and crosstalk delay results. For final signoff of the design, perform a full analysis using complete GPD or SPEF parasitic data and netlist data from the physical implementation tool.

Automatic Uniquifying of Blocks

If you edit a hierarchical block that is one of multiple instances of the same block, you make that block unique. The PrimeECO tool "uniquifies" that block for you automatically. For example, if the edited block is an instance of BLOCK, the tool renames it BLOCK_0 or some similar name, avoiding any names already used.

Making an instance of a design in PrimeECO unique is somewhat different from other Synopsys tools because a new design is not created at the time the block becomes unique. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by using the `link_design -remove_sub_designs` command.

To list designs created by automatic uniquifying, use the `list_designs -all` command. These designs become real only when you relink the design.

When a block is edited, it is uniquified, along with all blocks above it, up to the first singly-instanced block. Messages are generated when uniquifying occurs. For example, if you use the `size_cell` command to change the size of cell i1/low/n1, it causes multiple cells to be uniquified:

```
eco_shell> size_cell i1/low/n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

As soon as you edit a block, its parent and ancestor blocks up to the top level are edited. This information is available from a Boolean attribute, `is_edited`, which is available on a design and on hierarchical cells. In the foregoing example, after you use the `size_cell` command, the `is_edited` attribute is true on i1/low block:

```
eco_shell> get_attribute [get_cells i1/low] is_edited
true
```

Resolving Library Cells

Many of the netlist editing commands pass a library cell as an argument, which can be a library cell object or the name of a library cell.

To get library cell objects, use the `get_lib_cells` command, for example,

```
eco_shell> get_lib_cells */*
```

The name of a library cell can be either the following:

- The full name of the library cell, such as mylib1/AND2, in which case there is no ambiguity about what to link the cell to
- The base name of the library cell, such as AND2, in which case the library cell base name must be resolved to a library

Library cell resolution for netlist editing commands is used primarily for netlist editing in distributed multi-scenario analysis, where you cannot use full names of library cells because each scenario might have a different set of libraries. However, you can also use this feature in single-scenario analysis. For more information, see Netlist Editing in Multiple Scenarios.

You can resolve library cell base names from the cell's current library or from a specific library by using the `-current_library` or `-library` option, respectively:

- Use the `-current_library` option with the `size_cell`, `get_alternative_lib_cells`, and `report_alternative_lib_cells` commands. This option instructs the tool to resolve the library cell base name using the currently linked library.
- To specify a list of libraries to resolve a specified library cell name, use the `-libraries` option of the `size_cell`, `insert_buffer`, and `create_cell` commands. The tool searches the list of libraries in the order that you specify.

In the absence of these options, the PrimeECO tool resolves library cell base names from libraries in the following order:

- Link library of the current cell for the `size_cell`, `get_alternative_lib_cells`, or `report_alternative_lib_cells` command
- Libraries specified by the `link_path_per_instance` variable
- Libraries specified by the `link_path` variable

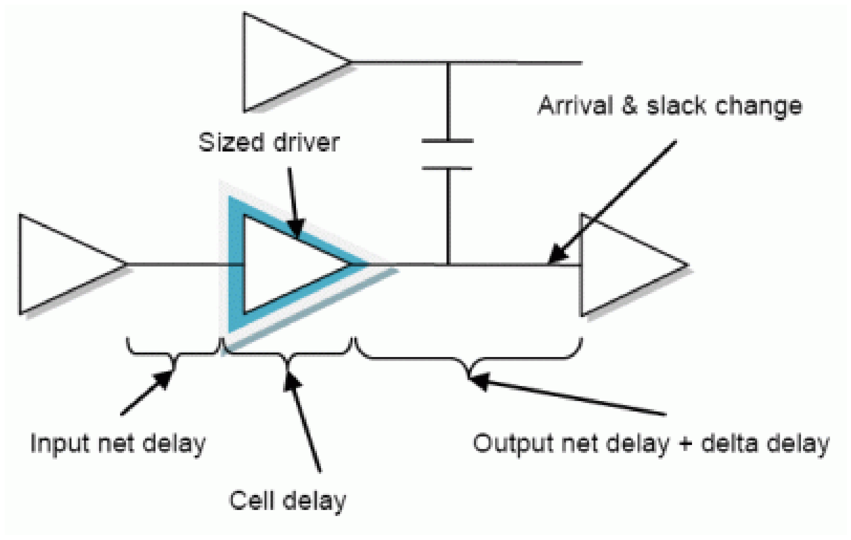
Use the `-base_names` option with the `get_alternative_lib_cells` command to return the alternative library cells as a list of library cell base names (strings) rather than a collection.

Estimating Delay Changes

When performing ECO design modifications, it is important to understand the timing effects. Reporting the effects of a change with the `report_timing` command incurs the cost of an incremental timing update.

For faster analysis of proposed `size_cell` and `insert_buffer` changes, use the `estimate_eco` command. This command quickly computes and displays the ECO alternatives based on current timing values of a particular stage. The estimate considers the incoming transition times, output loading, fanout loading, detailed parasitics if present, and driver characteristics of the candidate cells, as shown in the following diagram.

Figure 25 Parameters Considered by ECO Timing Estimation



The `estimate_eco` command is fast but the results are not guaranteed to be exact, so you should use this command only to explore different ECO options. For full accuracy, analyze the final results using the `report_timing` command.

When using the `estimate_eco` command, follow these steps:

1. Show the available options.

   In this stage, you evaluate possible ECO options. For example,

   ```
   eco_shell> estimate_eco -type size_cell -max -rise CPU2/U93

   delay type : max_rise
   lib cell              area stage_delay   arrival      slack
   -----------------------------------------------------------
   mylib90/INVD24       21.17     0.0265    0.0265     0.1671
   mylib90/INVD20       18.35     0.0291    0.0291     0.1645
   mylib90/CKNXD8       14.11     0.0610    0.0610     0.1326
   mylib90/INVD3         3.53     0.1295    0.1295     0.0212
   *mylib90/INVD2        2.82     0.1936    0.1936    -0.0013
   mylib90/CKNXD1        2.82     0.4271    0.4271    -0.3214
   ```

   This report shows alternative library cells and the timing that would result from using each library cell. The current library cell is marked with as asterisk (*).

2. Estimate the stage delay improvements.

After you deciding on the change, analyze the delay effects at the stage where the netlist change will occur. For example, to insert a buffer at a specified pin and estimate the stage delay change, use this command:

```
eco_shell> estimate_eco -type insert_buffer -inverter_pair \
           -max -rise -verbose -lib_cells {INVD2} CPU2/U25/Y

cell name : CPU/U25
current lib cell: mylib90/INVD1
buffer lib cell : mylib90/INVD2

max_rise                    current      estimate
-----------------------------------------------------------------
area                          0.925        1.270
input net delay               0.102        0.052
stage delay                   0.000        0.000
  cell delay                  0.371        0.175
  output net delay            0.206        0.110
  buffer delay                0.023        0.013
  delta delay                 0.140        0.051
arrival time                  0.371        0.175
slack                         0.304        0.499
```

3. Commit changes and verify.

   If you are sure about the fix, run the `size_cell` or `insert_buffer` command, and then run the `update_timing` command to see the actual timing changes with full accuracy.

   Note:

   The timing update does not take into account the changes in parasitics that could result from new placement and routing.

To customize the types of information reported by the `estimate_eco` command, set the `eco_estimation_output_columns` variable. In addition to the default information, you can also report the transition time, stage min/max transition, stage min/max capacitance, and stage max fanout.

Sizing Cells

To replace an existing cell with a new cell of a different size, use the `size_cell` command.

A library might contain several alternative library cells. For example:

```
eco_shell> get_alternative_lib_cells CPU2/reg318
{"class/FD2P1", "class/FD2P2", "class/FD2P3"}
```

If you do not know which cell to use for replacement, you can obtain the slack at the outputs of the leaf cell by using this command:

```
eco_shell> report_alternative_lib_cells CPU2/reg318
...
Report : alternative_lib_cells
        CPU2/reg318
        -delay_type max
...
Alternative                     Slack
Library Cells
------------------------------------------
class/FD2P3                      1.88(r)
class/FD2 *                     -1.02(r)
class/FD2P1                     -2.88(r)
class/FD2P2                     -2.98(r)
```

The report indicates that the class/FD2P3 cell would produce a positive slack at the output of the leaf cell. Therefore, you would choose class/FD2P3 to size the leaf cell:

```
eco_shell> size_cell CPU2/reg318 class/FD2P3
Information: Sized 'CPU2/reg318' with 'class/FD2P3'
1
```

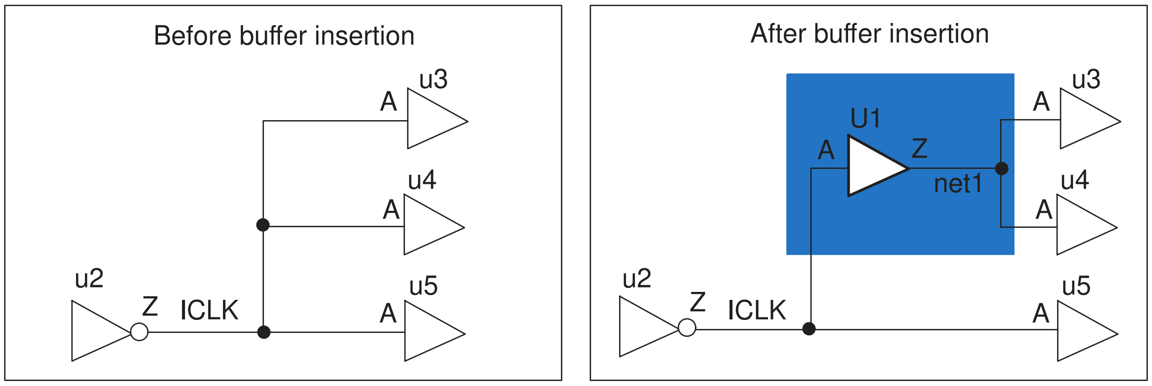
Inserting and Removing Buffers

To add a buffer at one or more pins in the netlist, use the `insert_buffer` command. To remove a buffer from the netlist, use the `remove_buffer` command.

For example, to insert buffers at pins u3/A and u4/A using library cell class/B1I:

```
eco_shell> insert_buffer {u3/A u4/A} class/B1I
```

Figure 26   Buffer Insertion Example



The `insert_buffer` command creates a new buffer “U1” and a new net “net1.” The input of the new buffer is the existing net “ICLK” and the output is the new net “net1.”

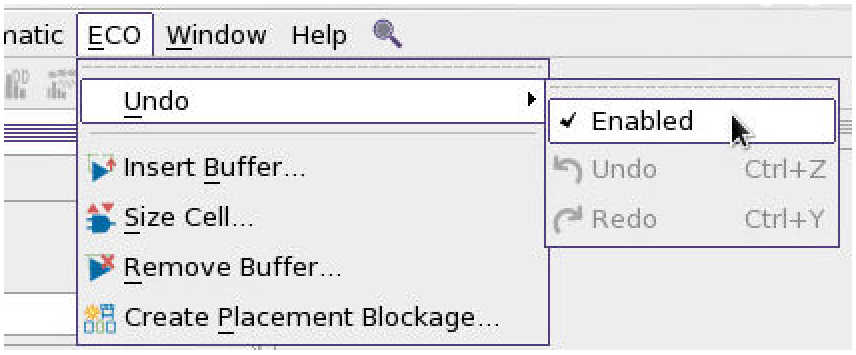
To change the prefix name of newly inserted buffers, set the `eco_instance_name_prefix` variable. By default, this variable is set to U.

Sizing Cells and Inserting Buffers in the GUI

The PrimeECO GUI provides menu commands for inserting buffers, removing buffers, and sizing cells. A PrimeTime-ADV-PLUS license is required to perform ECO actions in the GUI. After you make a change, you can easily undo it.

To perform an ECO operation in the GUI, use the menu options under ECO, shown in the following figure.

Figure 27 ECO Menu in the GUI



To enable the undo feature, make sure the Enabled option has a check mark. If you do not plan to use the feature, you can disable it to save memory.

Swapping Cells

To replace the cells in the specified cell list with a different design or library cell, use the `swap_cell` command.

Note:

> The `swap_cell` command is intended for swapping complex cells or hierarchical cells. When cells are swapped, a full timing update is incurred. For simple cell resizing, use the `size_cell` command instead.

Before performing the swap, the tool verifies that the pins of the replacement cell are equivalent to the pins of the original cell. For every pin of the cell you are swapping out, there must be a pin with the same name in the cell you are swapping in.

The `swap_cell` command always relinks the part of the design that has been replaced. Do not use the `link_design` command after you have used a `swap_cell` command; doing so often undoes the work of the `swap_cell` command.

The PrimeECO data model is instance based. When you use the `swap_cell` command, an instance is modified. For example, `U2/U0` is an instance of design `X`, and you swap `U2/U0` for a different design. PrimeECO did not modify the design `X`; it modified the instance `U2/U0`. Therefore, an initial link reverts to the old design.

PrimeECO does not save information about cells that were swapped; that information is maintained only until the next link. However, the change list for the design records the fact that the swap occurred, and you can export this information using the `write_changes` command.

By default, the `swap_cell` command restores the constraints that were on the design before the swap occurred. If you are doing multiple swaps, saving the constraints using the `write_script` command is far more efficient, and then use the `swap_cell` command with the `-dont_preserve_constraints` option.

When you have completed the swaps, restore the constraints by sourcing your script that you had written with the `write_script` command. If you are swapping one library cell for another, consider using the `size_cell` command, which is much more efficient. For example, to replace cells u1, u2, and u3 in TOP with the A design (in A.db), enter

```
eco_shell> read_verilog A.v
eco_shell> current_design TOP
eco_shell> swap_cell {u1 u2 u3} A.db:A
```

To replace cells u1, u2, and u3 in TOP with an LC3 lib_cell in the misc_cmos library, enter

```
eco_shell> swap_cell {u1 u2 u3} [get_lib_cells misc_cmos/LC3]
```

Renaming Cells or Nets

To change the name of a cell or net, use the `rename_cell` or `rename_net` command. When renaming a cell or a net using hierarchical names, ensure that the old and new names are at the same level of hierarchy. Reference any cell or net below the current instance by its full hierarchical name.

This example renames cellA to cellB in the current instance, middle.

```
eco_shell> rename_cell cellA cellB
Information: Renamed cell 'cellA' to 'cellB' in design 'blkA'. (NED-008)
1
```

This example renames a net at a level below the current instance. In this example, u1 and u1/low are instances of designs that have multiple instances, so they are uniquified as part of the editing process.

```
eco_shell> rename_net [get_nets u1/low/w1] u1/low/netA
Uniquifying 'u1/low' (low) as 'low_0'.
Uniquifying 'u1' (inter) as 'inter_0'.
Information: Renamed net 'w1' to 'netA' in 'blkA/u1/low'. (NED-009)
1
```

The `write_changes` command writes out the name changes using the `rename_cell` and `rename_net` commands.

Library Cell Functional Equivalency

PrimeECO operations can replace existing cells with functionally equivalent cells that have different area, drive strength, and power. A functionally equivalent cell must have the same logical function, same number and direction of I/O pins, and same timing arcs.

For some complex library cells, you might need to create a library cell attribute named `user_function_class` to describe the functional behavior of the cell. For details, see SolvNet article 020292, “What are the ECO Cell Equivalency Rules for PrimeTime?” The foundry can provide information about function classes to ensure that the proper cells are chosen for upsizing.

Netlist Editing in Multiple Scenarios

When editing the netlist to fix violations, you should verify that the changes do not negatively affect the timing results in all scenarios. To do this, use distributed multi-scenario analysis (DMSA) to evaluate the effects and get merged reporting on the tested scenarios.

With DMSA, you can run any number of netlist editing commands using a varying command focus. You can also execute netlist editing commands directly at the master. However, complex nested command structures are not supported in this mode of operation.

In DMSA, you can use the netlist editing commands in Table 7. You cannot use the `swap_cell` command in the DMSA master, but you can execute it through the `remote_execute` command.

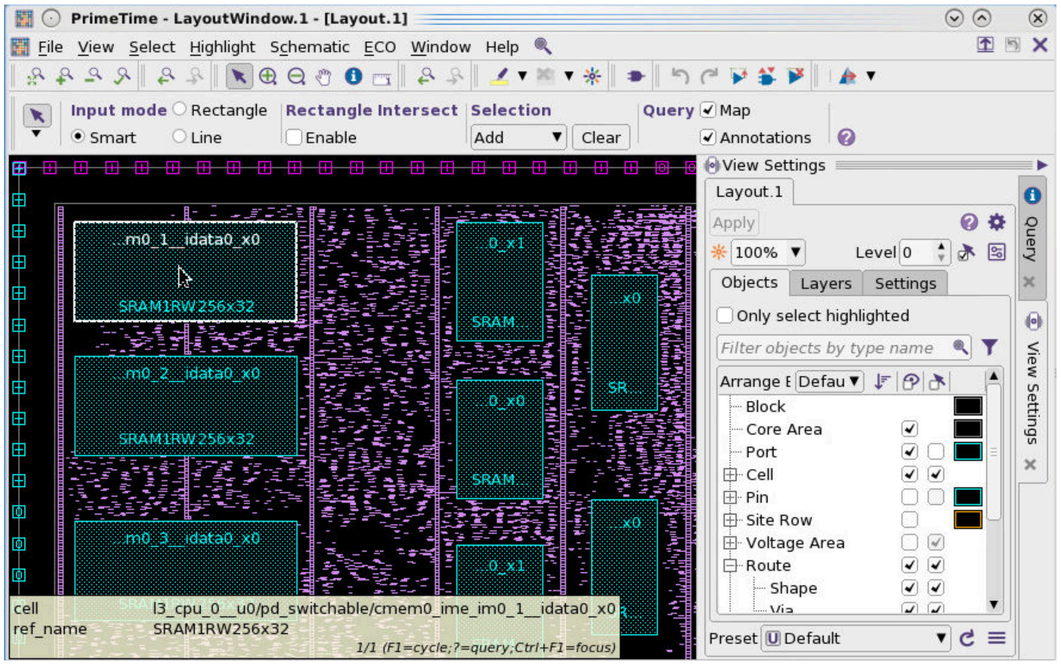
By using the `remote_execute` command, you can execute the following commands in a worker process and apply them to all scenarios in the command focus:

- `set_coupling_separation` – Creates a separation constraint on nets in all the scenarios in command focus.
- `remove_coupling_separation` – Removes the coupling separation from all scenarios in command focus.
- `read_parasitics -eco` – Reads StarRC ECO parasitic files to all scenarios in command focus.

Layout View and ECOs in the GUI

To view the chip layout in the GUI in preparation for engineering change orders (ECOs), from the main GUI window, choose Window > Layout window. This opens the layout window, as shown in the following figure.

Figure 28 Layout View in the GUI

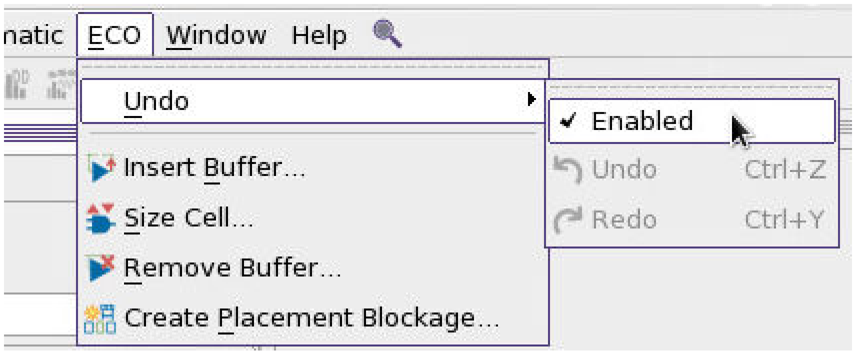


To use the layout view, you need to specify the physical data by using the `set_eco_options` command. The physical data can be in IC Compiler II or LEF/DEF format. To check for the presence of valid physical data, use the `check_eco` command.

You can generate and view ECOs in the GUI, including inserting buffers, removing buffers, and sizing cells. A PrimeTime-ADV-PLUS license is required. After you make a change, you can easily undo it.

To perform an ECO operation in the GUI, use the menu options under ECO, shown in the following figure.

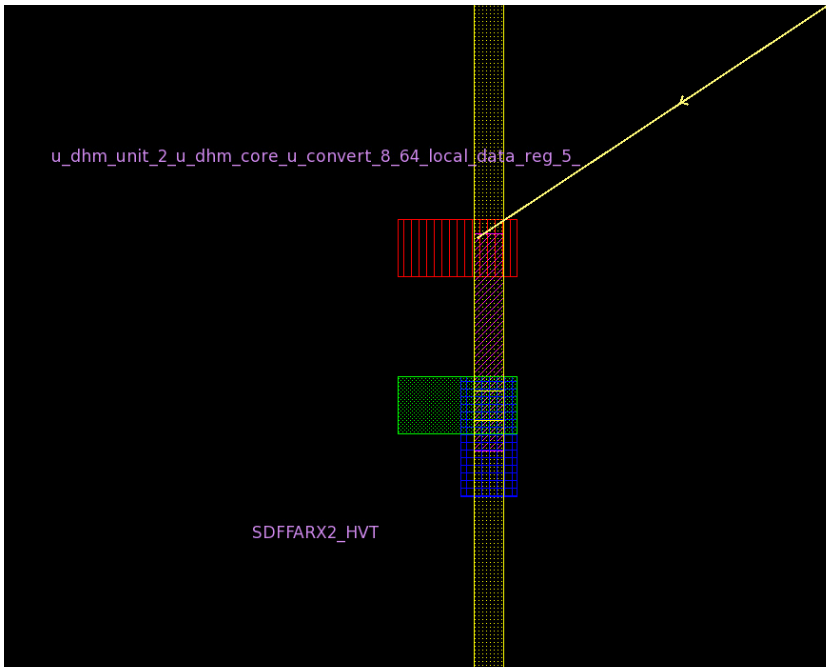
Figure 29 ECO Menu in the GUI



To enable the undo feature, make sure the Enabled option has a check mark. If you do not plan to use the feature, you can disable it to save memory.

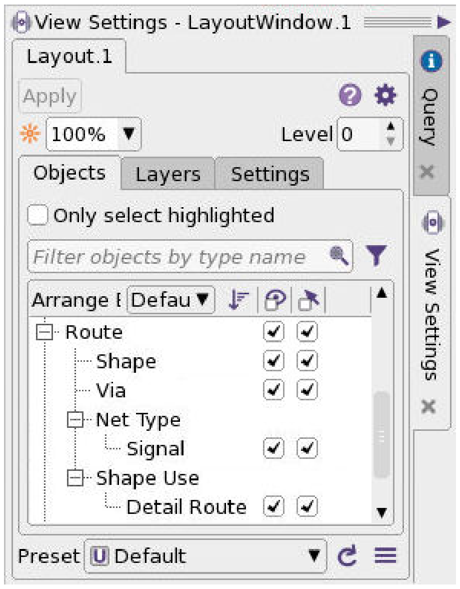
You can highlight a timing path and show only the actual net shapes traversing the path, including relevant vias, without including the unrelated fanout of each driver in the path.

Figure 30 Net Shape Display



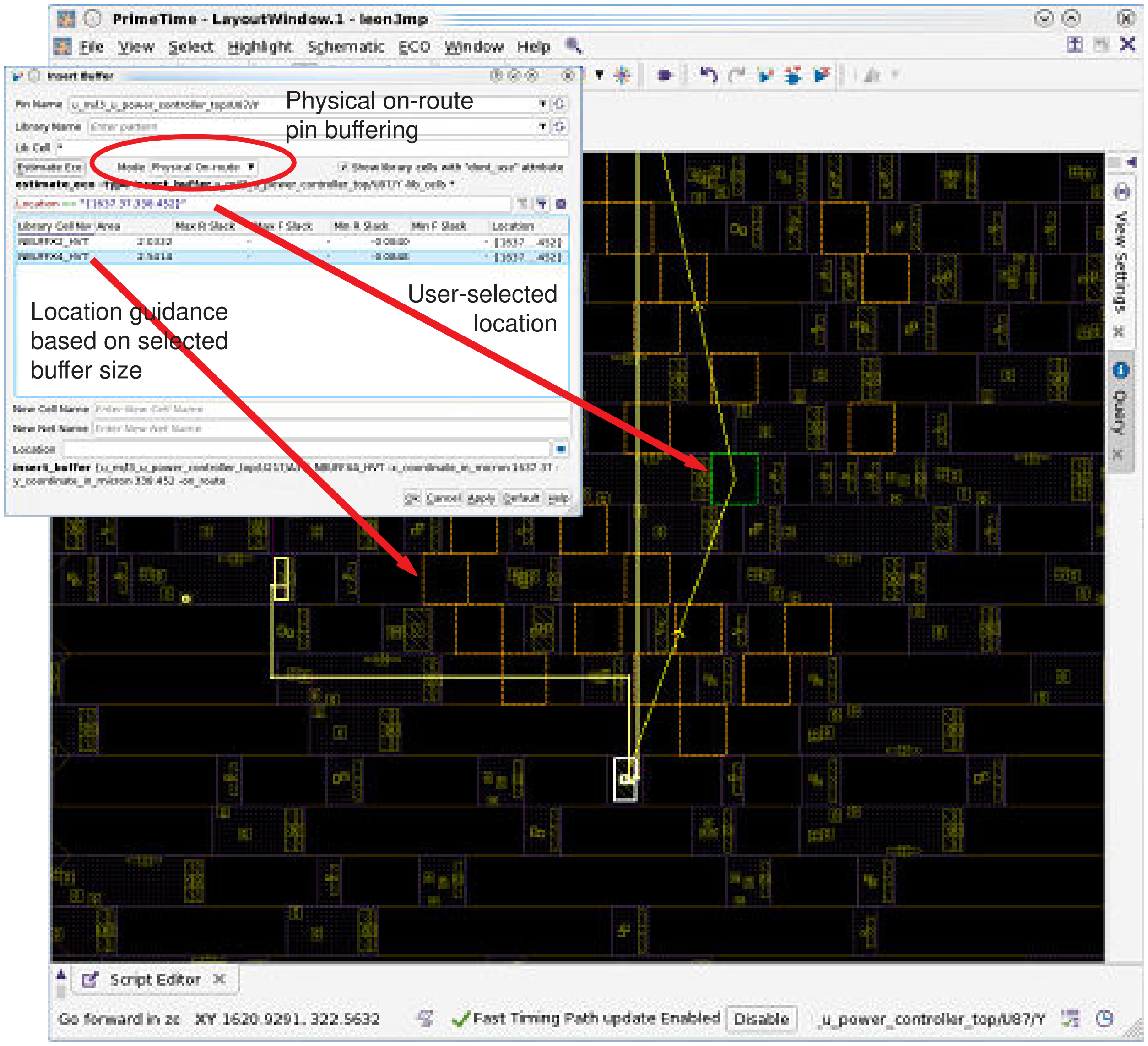
To control the net-shape-only path display mode, use the option settings in the View Settings box under the Objects tab.

Figure 31 Net Shape Display Option Controls



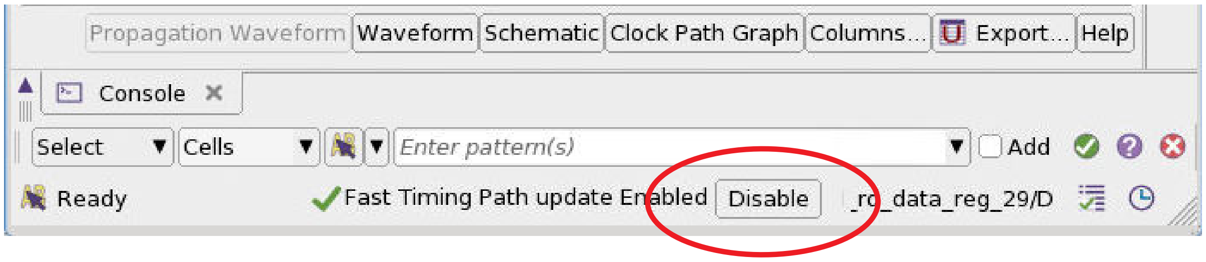
When you perform manual ECO operations using the ECO > Insert Buffer or ECO > Size Cell command (`insert_buffer` or `size_cell`), the tool automatically performs an incremental path-only timing update in memory so that you can immediately determine the timing effects of the change.

Figure 32 Manual ECO Operations in the GUI


PrimeTime - LayoutWindow.1 - leon3mp
File View Select Highlight Schematic ECO Window Help
Insert Buffer
Physical on-route pin buffering
Mode Physical On-route
User-selected location
Location guidance based on selected buffer size
OK Cancel Apply Default Help
View Settings
Query
Script Editor
Fast Timing Path update Enabled
Disable

Placement blockages are honored during on-route buffer guidance.

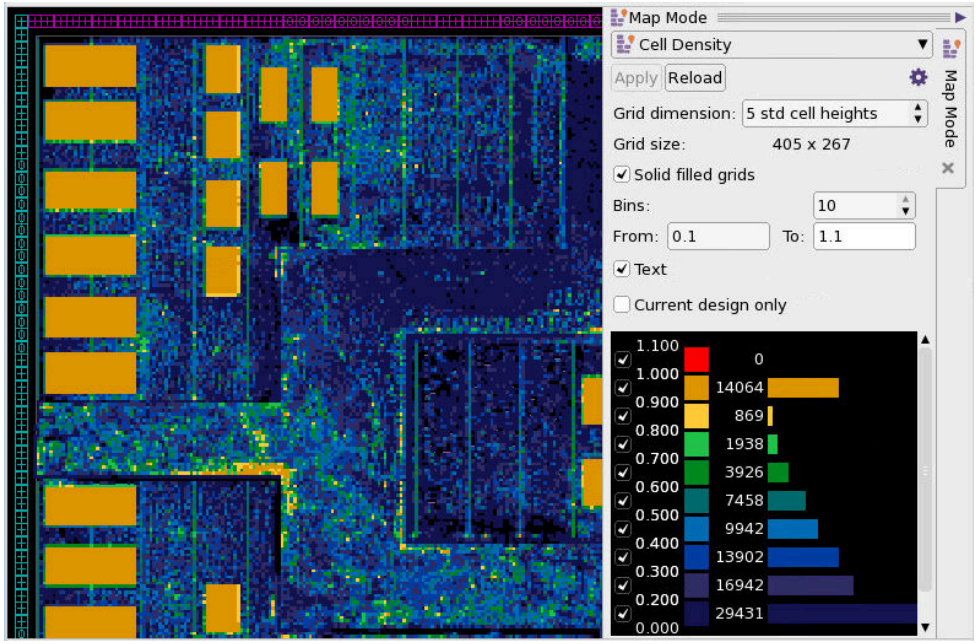
To disable the automatic incremental path-only timing update, click the Disable button at the bottom of the GUI window next to the label “Fast Timing Path update Enabled.”



With this feature disabled, you can initiate a regular incremental or full timing update when needed, and using the `estimate_eco` command or querying design attributes triggers a timing update.

To display color-coded cell density maps and pin density maps in the layout window, choose View > Map > Cell Density or View > Map > Pin Density. A PrimeTime-ADV-PLUS license is required.

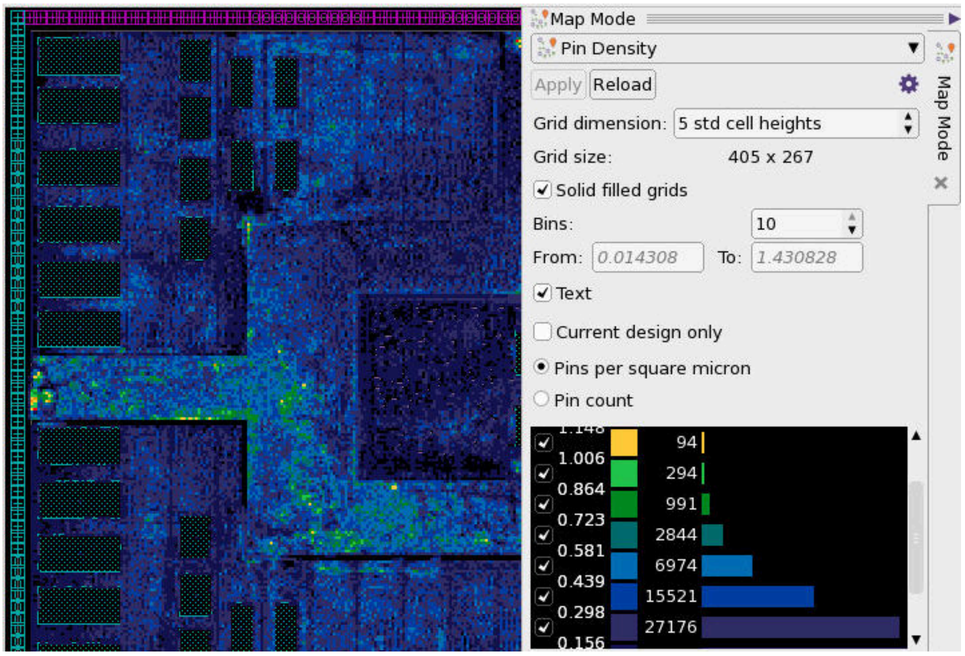
Figure 33 View > Map > Cell Density in Layout View



Grid dimension: 5 std cell heights
Grid size: 405 x 267
✓ Solid filled grids
Bins: 10
From: 0.1 To: 1.1
✓ Text
Current design only

✓	1.100	🟥	0
✓	1.000	🟧	14064
✓	0.900	🟨	869
✓	0.800	🟩	1938
✓	0.700	🟩	3926
✓	0.600	🟩	7458
✓	0.500	🟦	9942
✓	0.400	🟦	13902
✓	0.300	🟪	16942
✓	0.200	🟦	29431
	0.000		

Figure 34 View > Map > Pin Density in Layout View



Pin Density

Grid dimension: 5 std cell heights

Grid size: 405 x 267

✓ Solid filled grids

Bins: 10

From: 0.014308 To: 1.430828

✓ Text

☐ Current design only

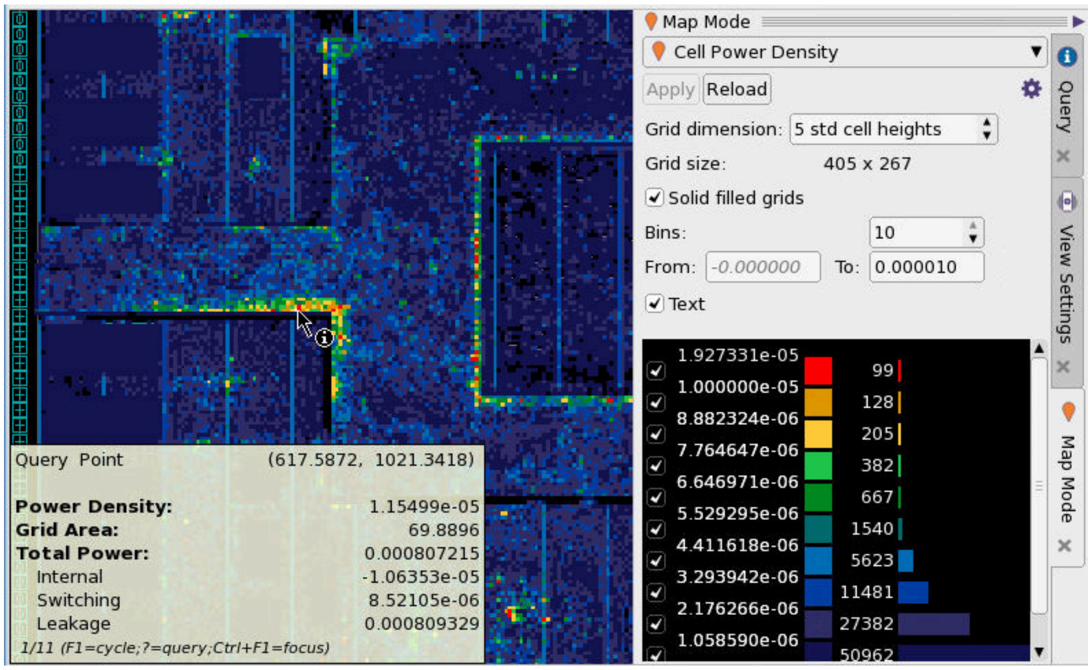
● Pins per square micron

○ Pin count

✓	1.148	🟨	94
✓	1.006	🟩	294
✓	0.864	🟩	991
✓	0.723	🟦	2844
✓	0.581	🟦	6974
✓	0.439	🟦	15521
✓	0.298	🟪	27176
	0.156		

To display a color-coded power density map in the layout window, choose View > Map > Cell Power Density. Viewing the power map requires PrimePower analysis data generated by the `update_power` or `report_power` command.

Figure 35 View > Map > Cell Power Density in Layout View



Grid dimension: 5 std cell heights
Grid size: 405 x 267
✓ Solid filled grids
Bins: 10
From: -0.000000 To: 0.000010
✓ Text

✓	1.927331e-05	🟥	99
✓	1.000000e-05	🟧	128
✓	8.882324e-06	🟨	205
✓	7.764647e-06	🟩	382
✓	6.646971e-06	🟩	667
✓	5.529295e-06	🟩	1540
✓	4.411618e-06	🟦	5623
✓	3.293942e-06	🟦	11481
✓	2.176266e-06	🟦	27382
✓	1.058590e-06	🟦	50962

```
Query Point                       (617.5872, 1021.3418)

Power Density:                         1.15499e-05
Grid Area:                                 69.8896
Total Power:                           0.000807215
  Internal                             -1.06353e-05
  Switching                             8.52105e-06
  Leakage                              0.000809329
1/11 (F1=cycle;?=query;Ctrl+F1=focus)
```

Writing ECO Change Lists for Third-Party Place-and-Route Tools

In a PrimeECO flow, you can write change lists for third-party place-and-route tools by using the `-format aprtcl` option of the `write_changes` command.

The `aprtcl` format (short for Automatic Place-and-Route) writes the changes as a set of pseudo-Tcl commands that are not specific to a particular tool. These commands can be parsed and incorporated into third-party place-and-route tools.

Table 8 lists the pseudo-Tcl commands specific to this format. Commands not shown in the table are equivalent to the `-format icc2tcl` format.

Table 8   Pseudo-Tcl Commands Used by the write_changes -format aprtcl Command

ECO operation	Pseudo-Tcl command syntax

Insert buffer (pin-based)	`insert_buffer`
	`  new_lib_cell`
	`  -new_net_names net_names`
	`  -new_cell_names buff_names`
	`  [-location {x y}]          # buffers`
	`  [-location {x1 y1 x2 y2}]  # inverter pairs`
	`  [-orientation dir]`
	`  [-inverter_pair]`
	`  {pin_or_port_list}`

Insert buffer (on-route)	`insert_buffer`
	`  new_lib_cell`
	`  -on_route`
	`  -new_net_names net_names`
	`  -new_cell_names buff_names`
	`  [-location {x y}]          # buffers`
	`  [-location {x1 y1 x2 y2}]  # inverter pairs`
	`  [-orientation dir]`
	`  [-inverter_pair]`
	`  [-route_cut_location {x y}]          # buffers`
	`  [-route_cut_location {x1 y1 x2 y2}]  # inverter pairs`
	`  {pin_or_port_list}`

Resize cell	`size_cell`
	`  leaf_cell`
	`  new_lib_cell`
	`  [-overlap]`
	`  [-location {x y}]`
	`  [-orientation dir]`

Create cell (for load cells)	`create_cell`
	`  load_cell_name`
	`  new_lib_cell`
	`  [-location {x y}]`
	`  [-orientation dir]`

Connect net (for load cells)	`connect_net`
	`  net_name`
	`  pin_or_port_list`

Remove buffer	`remove_buffer`
	`  cell_name`

For details, see the `write_changes` man page.

Example 1 shows an example change list that creates a buffer tree to drive four load pins, and adds a load cell to fix a hold violation. The buffer insertion order and load pin lists re-create the required tree structure.

Example 1   Example Change List From write_changes -format aprtcl

```
########################################################################
######
# Change list, formatted for Third Party Compiler
#
#
#
# aprtcl_file_syntax_version : 1.0
#
########################################################################
######
current_instance
current_instance {blk/subblk3}
insert_buffer -on_route BUFFD4 \
  -new_net_names {net1} -new_cell_names {BUF1} \
  -location {x1 y1} -route_cut_location {x1’ y1’} \
  {sink1/I sink2/I sink3/I sink4/I}
insert_buffer -on_route BUFFD2 \
  -new_net_names {net2} -new_cell_names {BUF2} \
  -location {x2 y2} -route_cut_location {x2’ y2’} \
  {sink1/I sink2/I sink3/I}
insert_buffer -on_route BUFFD4 \
  -new_net_names {net4} -new_cell_names {BUF4} \
  -location {x4 y4} -route_cut_location {x4’ y4’} \
  {sink4/I}
insert_buffer -on_route BUFFD4 \
  -new_net_names {net3} -new_cell_names {BUF3} \
  -location {x3 y3} -route_cut_location {x3’ y3’} \
  {sink1/I sink2/I}
insert_buffer -on_route BUFFD2 \
  -new_net_names {net5} -new_cell_names {BUF5} \
  -location {x5 y5} -route_cut_location {x5’ y5’} \
  {sink3/I}
create_cell {U_LOAD_CAP_CELL_1} {CLOAD1_LVT} \
  -location {150.6320 61.8640} -orientation N
connect_net {design_ack_signal} {U_LOAD_CAP_CELL_1}
```

The header of the change list file includes a version string so that external parsers can adapt to changes in the format over time. Currently, the only possible version value is `1.0`. It is recommended that parser scripts check this version for an expected value, so that future syntax upgrades do not cause unexpected behavior.

Feedback

A

Legacy Flow Topics

This appendix includes older topics that documented deprecated legacy flows:

- Block-Level LEF Library Data
- DEF Files or IC Compiler II Database Files

Block-Level LEF Library Data

From the IC Compiler II tool, write out the block-level Library Exchange Format (LEF) data for all standard cells as well as all macros. Only one LEF file needs to include the technology information for the design. For example,

```
open_lib lib1
write_lef -include {cell tech} lib1.lef
close_lib

open_lib lib2
write_lef -include {cell} lib2.lef
close_lib

open_lib lib3
write_lef -include {cell} lib3.lef
close_lib
```

DEF Files or IC Compiler II Database Files

A Design Exchange Format (DEF) file contains the placement and routing information of the design. Physical ECO fixing needs the hierarchical top-level and block-level DEF files (version 5.7 or higher) or IC Compiler II database files to find available sites and place buffers near the original routes.

Here is an example of an IC Compiler script, called write_def.tcl:

```
open_mw_lib design.mw
open_mw_cel post_route_design
current_design design
link
write_def -all_vias -output file_com.def
```

To include via definitions that might not be part of the via definitions in the technology LEF, use the `write_def -all_vias` command.

For more information, see the “Writing the Floorplan to a DEF File” section in the *IC Compiler Design Planning User Guide*.