IC Validator Deck Creation and Validation Toolkit User Guide

Version T-2022.03-SP3, September 2022

SYNOPSYS®

Feedback

Contents

Feedback

About This User Guide

This user guide describes the *Deck Creation and Validation (DCV) Toolkit* utilities for the IC Validator tool, and is designed to enhance the knowledge that both beginning and advanced IC Validator users have of the DCV Toolkit.

This preface includes the following sections:

- New in This Release
- Related Products, Publications, and Trademarks
- Conventions
- Customer Support
- Statement on Inclusivity and Diversity

New in This Release

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

Related Products, Publications, and Trademarks

For additional information about the IC Validator 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:

- Synopsys® Custom Compiler™
- Synopsys IC Compiler™
- Synopsys IC Compiler™ II
- Synopsys Fusion Compiler™
- Synopsys StarRC™

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.

Statement on Inclusivity and Diversity

1

Introduction to Deck Creation and Validation

This chapter provides an introduction to IC Validator Deck Creation and Validation (DCV) Toolkit.

The DCV Toolkit is a set of utilities that helps you to create and validate DRC, LVS, and Fill runsets. You can configure and execute each of the tools in the toolkit, review the results, and validate or improve the quality and performance of your runsets.

For more information, see the following section:

- DCV Toolkit

DCV Toolkit

The DCV Toolkit consists of the following tools:

- DCV Analyzer

  Use this tool to analyze the performance of an IC Validator run. The tool creates organized reports that can help you to identify factors limiting overall performance.

  For more information, see Chapter 2, DCV Analyzer Tool.

- DCV Design Trimmer Tool

  This tool reduces test cases to smaller cases to aid in debugging.

  For more information, see Chapter 3, DCV Design Trimmer Tool

- DCV Results Compare

  This tool validates the quality of IC Validator output results. The tool compares two sets of error output results and identifies discrepancies between them.

  For more information, see Chapter 4, DCV Results Compare Tool.

- DCV Runset Coverage

  This tool evaluates the quality of DRC QA regressions by examining how well the regression data exercises a specified runset.

For more information, see Chapter 5, DCV Runset Coverage Tool.

- DCV Switch Optimizer

  This tool validates the syntax correctness of an IC Validator runset. The tool identifies and exercises a minimal set of runset preprocessor flow-control switches that exercise all of the code in the runset, and then reports any combinations that produce a runset syntax error.

  For more information, see Chapter 6, DCV Switch Optimizer Tool.

- DCV Test Case Compiler

  This tool generates non-functional layout database files for use in testing the IC Validator tool.

  For more information, see Chapter 7, DCV Test Case Compiler Tool.

2
DCV Analyzer Tool

This chapter explains how to run the DCV Analyzer tool and view the reports.

The IC Validator DCV Analyzer tool serves as a starting point for analyzing the performance of an IC Validator run.

For more information, see the following sections:

- Overview
- Running the Tool
- Analyzing and Comparing Performance
- Analyzing and Comparing Hierarchies

Overview

The DCV Analyzer tool takes the output of an IC Validator run and displays the data in a clear, organized way. You should use this tool as the starting point for IC Validator performance analysis.

You can use the DCV Analyzer tool to

- Analyze performance using the distributed log file from an IC Validator run
- Analyze hierarchy using a tree structure file from an IC Validator run
- Compare hierarchies using two tree structure files from the same or different IC Validator runs
- Compare performance using two distributed log files from different IC Validator runs

You can perform each of these tasks individually or perform any combination of them in a single DCV Analyzer run. The tool determines which operations to perform based on the input files that you specify.

- Distributed log files contain information about the commands that IC Validator executed during a run.
- Tree structure files contain information about the design hierarchy tree.

These IC Validator output files are located in the run_details directory. For more information about the files, see the *IC Validator User Guide*.

By default, the DCV Analyzer tool takes input from the files that you specify, stores the data in an SQLite database, analyzes the data, and provides output in report files. In general, the tool performs the following operations:

1. Parses the distributed log file or tree structure files and loads the input data into an SQLite database
2. Analyzes the data from the SQLite database and generates reports

In subsequent runs, the tool can reanalyze the data and regenerate or add reports from the same SQLite database.

For performance analysis, the tool provides output in a summary report file and a user-functions file. The summary report file contain runtime statistics, efficiency metrics, and tables highlighting discrete aspects of the run, such as the runset functions that took the longest or used the most memory. For performance comparisons, the tool provides additional output in a dplog comparison file named dplog_compare.txt.

Optional output includes individual table report files, in tab-separated-value format, that you can import into a third-party tool, such as the Microsoft® Excel® tool, for further analysis. You can also generate command chain files for long-pole analysis. In addition, you can customize the number of table rows in the reports .

For hierarchy analysis, the tool provides output in a tree summary report file named tree_summary.txt. Optional output can include cell and layer report files. For hierarchy comparisons, the tool provides additional output in a tree comparison file named tree_compare.txt.

Running the Tool

Before running the DCV Analyzer tool, make sure the `LM_LICENSE_FILE` environment variable and the *`ICV_HOME_DIR`* installation directory are set. The `dcv_analyzer` executable is located in the same bin directory as the `icv` executable. The tool creates the database and report files in your current working directory.

The DCV Analyzer command-line syntax is

```
dcv_analyzer [options] [runset.dp.log] [runset.dp.log] [top-cell.treeX]
 [top-cell.treeX]
```

where

- *options* are described inTable 1.
- *runset*.dp.log is the name of the distributed log file from an IC Validator run.
- *top-cell*.tree*X* specifies the name of a tree structure file from an IC Validator run.

  You can specify one tree structure file for hierarchy analysis or two tree structure files to compare the hierarchies. The files can come from the same or different IC Validator runs. In the tree structure file names, *X* is an integer from 0 to 999. The format of the tree structure files (Milkyway, NDM, LTL, GDSII, or OASIS) does not matter.

  The DCV Analyzer can also handle these files in gzipped format.

For more information, see the following sections:

- Command-Line Options
- Environment Variables

Command-Line Options

Table 1 describes the command-line options.

Table 1 DCV Analyzer Command-Line Options

Option	Description
`-c` *`count`* `| all`	Specifies the number of records for fields in the report. By default, the tool generates 10 records for most fields. The *count* must be a positive integer. Use `all` if you want the tool to return all records by the sort order.

`-C` *`cell-list`*
`-cell` *`cell-list`*
	Specifies a comma-separated list of cell names in *cell-list*. The tool provides cell statistics for each cell in cell information files named cell_*cellname*.txt. At least one tree structure input file (*top-cell*.tree*X*) must be provided.
	You can use the asterisk wildcard character (*) to specify multiple cells in a name pattern. When using wildcard characters, you should enclose the entire argument in quotation marks to prevent the shell from interpreting files in the current working directory.

Table 1 DCV Analyzer Command-Line Options (Continued)

Option	Description
`-e` `-export`	Provides the output data in table-based report files named *_table.tsv. You can import these tab-separated-values files into the Microsoft Excel tool for further analysis. All time-based metrics are in seconds, and all memory-based metrics are in GB.

`-f` *`command_key`*
`-full-cmdhist` *`command_key`*
	Specifies a comma-separated list of the command keys used to generate the full command dependency history. If the first element in a command key is a positive integer, the tool produces that number of worst-case graphs. The output file name format is fullHist_*command_key*.txt. Each full history file shows all dependencies which were run to reach the command key. These are shown in command execution order, not dependency order.

`-F` *`violation_comment`*
`-full_cmd-violation` *`violation_comment`*
	Produces the same fullHist_*command_key_rule*.txt files as the `-f` command-line option, but uses the *violation-list* comments as input. Enclose each comment in quotation marks and separate with commas. For example, `"*a*","b*"` (where `*` is a wildcard character).
	When comparing two log files, this command-line operation also generates a fullHist_compare_*rule*.txt file.

`-h` `-help`	Displays the usage message and exits.
`-intrinsicSummary`	Creates a summary file, which contains statistical performance information about each intrinsic in the run.

`-l` *`command_key`*
`-long-pole-hist` *`command_key`*
	Specifies a comma-separated list of the command keys used to generate the long-pole history. If the first element in a command key is a positive integer, the tool produces that number of worst-case graphs. The output file name format is cmdchain_*command_key*.txt. Each command chain file shows the longest serial path of dependencies to reach the command key.

Table 1 DCV Analyzer Command-Line Options (Continued)

Option	Description

`-L` *`layer-list`*
`-layer` *`layer-list`*
	Specifies layers in a comma-separated list of layer names, layer number and data type pairs, or both. The tool provides layer statistics for each layer in layer information files named layer_*layer-name*.txt for each layer name and *layer_num*:*data_type*.txt for each layer number and data type pair.
	You must provide at least one tree structure input file (*top-cell*.tree*X*) when you use this option. If *layer-list* contains layer number and data type pairs, the tree structure input file must be a single *top-cell*.tree0 file.
	You can use the asterisk wildcard character (*) to specify multiple layers in a layer name pattern. When using wildcard characters, enclose the entire argument in quotation marks to prevent the shell from interpreting files in the current working directory.
	Wildcard characters are not allowed in a layer number and data type pair. Specify each pair by using the form *`layer-number`*`:`*`data-type`*, where *layer-number* and *data-type* are integers.
	For example, `"M1,1:0,D*"` specifies layer name M1, layer number and data type pair 1:0, and layer name pattern D*. The quotation marks prevent the shell from expanding D*.

`-lpm` *`time-type`*
`-long-pole-method` *`time-type`*
	Changes what time is used to calculate long poles (not valid with the `-r` command-line option). Valid time measurements are: Elapsed, User, Check, None (disabled).

`-r` `-report-only`	Regenerates the reports using the data in the SQLite database from a previous run instead of parsing and analyzing the data from the distributed log file. At least one input file (dplog.db or tree.db) must be present in the current working directory. **Note:** Regeneration of the output file is skipped unless an option that changes the content of the file is used.
`-t` `-tabs`	Changes the output file name extensions to .tsv and the file format to tab separation instead of fixed-width printing. This option allows you to more easily import an output file into the Microsoft Excel tool or to parse specific fields in a file by using a script.

Table 1 DCV Analyzer Command-Line Options (Continued)

Option	Description
`-traceRules COMMANDKEY`	Operating similarly to the `-l` command-line option, this command-line option traces from the command key to the affected rules, which is helpful for determining how to select rules to run or avoid specific commands. The syntax is: `min line num:longest elapsed time commands,command key list` The first two parameters are optional. The *`min line num`* affects all #include files, and must be set to the lowest number for a command of interest on any of the files. **Note:** Using numerous commands generates a large output and can potentially take very long periods of time.
`-train`	Creates a new machine learning model for the `DCV_ML_MODELS_DIR` environment variable. The `CERBERUS_HOME` and `CERBERUS_PROJECT` must be set.

`-v` *`violation-list`*
`-violation-cmdchain` *`violation-list`*
	Produces the same cmdchain_*command_key_rule*.txt files as the `-l` command-line option, but uses the *violation-list* comments as input. Enclose each comment in quotation marks and separate with commas. For example, `"*a*","b*"` (where `*` is a wildcard character).

`-V` `-version`	Prints the tool version.

`-x` *`XREF`*
`-xref` *`XREF`*
	Changes the content of the field used to cross-reference the SQLite database. *XREF* can be `key`, `number`, or `line`. The default is `key`.
	• `key` prints the command key
	• `number` prints the command number
	• `line` prints the line number from the dp.log file
	In the following example, line 7105 from a distributed log file contains the command key 1337.0.0 and the command number 3163:
	`Host teralab-48(0) completes 1337.0.0, 3163:`

Environment Variables

When the DCV Analyzer tool detects an environment variable that affect its behavior, it notifies you in the standard output. For example:

```
11:15:44 Creating file: dplog_summary.txt
   INFO: Using Env Var: "DCV_USER_FUNCTION_VALUES" with value:
          "CUMULATIVE"
11:15:44 Creating file: userfunction_summary.txt
```

You can control the appearance of the standard output and log files by setting the following DCV Analyzer environment variables:

- `DCV_ML_MODELS_DIR`

  Specifies the directory from which to obtain the machine learning models. See the Runtime Predictions section for more information. The default is:

  `$ICV_HOME_DIR/etc/ML_models`

- `DCV_USER_FUNCTION_LOOP`

  Set this variable to control how loops are displayed in the User Function related reports. The `DISCRETE` setting splits all of the loop data into separate information for each iteration of the loop. The `CUMULATIVE` setting ignores all of the loop information at the top level and reports totals for all of the passes of loop data involved with the user functions. The `BOTH` setting prints two tables, one with each of the other values. The default is `DISCRETE`.

  Example 1 and Example 2 show the differences in the Slowest User Functions sorted by Check Time when the `DCV_USER_FUNCTION_LOOP` is `DISCRETE` (the default) versus `CUMULATIVE`.

Example 1 DCV_USER_FUNCTION_LOOP is DISCRETE

```
---------------------------------------------------------------------------
-----------
Slowest User Functions Sorted by Check Time:
---------------------------------------------------------------------------
-----------
User Function                                   Intrinsic   Avg      Total
                                                Count       Check    Time
                                                          Time(min) Time(m
in)
top_level_uf@analyzerDocs.rs:329/
             random_repeater@analyzerDocs.rs:224        3        0.1
       0.4
top_level_uf@analyzerDocs.rs:329/sub1@analyzerDocs.rs:225/
 sub2@analyzerDocs.rs:202                                2        0.0
  0.1
top_level_uf@analyzerDocs.rs:329                        4        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             random_repeater@analyzerDocs.rs:210        3        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             sub2@analyzerDocs.rs:211                   2        0.0
 0.0
copyOut@analyzerDocs.rs:331                             1        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-1 1        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-2 1        0.0
 0.0
```

```
top_level_uf@analyzerDocs.rs:329/
             sub1@analyzerDocs.rs:225                   1        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/
             no_intrinsic@analyzerDocs.rs:226           0        0.0
       0.0

DCV_USER_FUNCTION_LOOP: DISCRETE DCV_USER_FUNCTION_VALUES: DISCRETE


---------------------------------------------------------------------------
-----------
```

Example 2 DCV_USER_FUNCTION_VALUES is CUMULATIVE

```
---------------------------------------------------------------------------
-----------
Slowest User Functions Sorted by Check Time:
---------------------------------------------------------------------------
-----------
User Function                                   Intrinsic   Avg      Total
                                                Count       Check    Time
                                                          Time(min) Time(m
in)
top_level_uf@analyzerDocs.rs:329/
             random_repeater@analyzerDocs.rs:224      3          0.1
       0.4
top_level_uf@analyzerDocs.rs:329/sub1@analyzerDocs.rs:225/
             sub2@analyzerDocs.rs:202                 2           0.0
  0.1
top_level_uf@analyzerDocs.rs:329                      4           0.0
  0.0
copyOut@analyzerDocs.rs:331                           1           0.0
  0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230         2           0.0
  0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             random_repeater@analyzerDocs.rs:210      3          0.0
       0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             sub2@analyzerDocs.rs:211                 2          0.0
 0.0
top_level_uf@analyzerDocs.rs:329/
             sub1@analyzerDocs.rs:225                 1           0.0
  0.0

DCV_USER_FUNCTION_LOOP: CUMULATIVE DCV_USER_FUNCTION_VALUES: DISCRETE


---------------------------------------------------------------------------
-----------
```

- `DCV_USER_FUNCTION_VALUES`

  Set this variable to control how the values (Elapsed Time, Check Time, User Time, System Time, Memory and Intrinsic counts) from nested user functions are displayed in the User Function related reports. The `DISCRETE` setting reports only the values from intrinsics called natively in the user function. The `CUMULATIVE` setting includes values from all of the nested user functions called from the higher-level functions. The `BOTH` setting prints two tables, one with each of the other values. The default is `DISCRETE`.

Example 3 and Example 4 show the differences in the Slowest User Functions sorted by Check Time when the `DCV_USER_FUNCTION_VALUES` is `DISCRETE` (the default) versus `CUMULATIVE`.

The highlighted intrinsic count numbers of the `DISCRETE` example equal the highlighted intrinsic count number of the `CUMULATIVE` example.

Example 3   DCV_USER_FUNCTION_VALUES is DISCRETE

```
---------------------------------------------------------------------------
-----------
Slowest User Functions Sorted by Check Time:
---------------------------------------------------------------------------
-----------
User Function                                      Intrinsic   Avg      Total
                                                   Count       Check    Time
                                                             Time(min)  Time(m
in)
top_level_uf@analyzerDocs.rs:329/
             random_repeater@analyzerDocs.rs:224        3        0.1
       0.4
top_level_uf@analyzerDocs.rs:329/sub1@analyzerDocs.rs:225/
 sub2@analyzerDocs.rs:202                                2        0.0
       0.1
top_level_uf@analyzerDocs.rs:329                        4        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             random_repeater@analyzerDocs.rs:210        3        0.0
 0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             sub2@analyzerDocs.rs:211                   2           0.0
    0.0
copyOut@analyzerDocs.rs:3311                            1           0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-1 1           0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-2 1           0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             sub1@analyzerDocs.rs:2251                  1           0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             no_intrinsic@analyzerDocs.rs:226           0
         0.0      0.0

DCV_USER_FUNCTION_LOOP: DISCRETE DCV_USER_FUNCTION_VALUES: DISCRETE


---------------------------------------------------------------------------
-----------
```

Example 4 DCV_USER_FUNCTION_VALUES is CUMULATIVE

```
---------------------------------------------------------------------------
-----------
Slowest User Functions Sorted by Check Time:
---------------------------------------------------------------------------
-----------
User Function                                   Intrinsic   Avg       Total
                                                Count       Check     Time
                                                          Time(min)  Time(m
in)
top_level_uf@analyzerDocs.rs:329                      17         0.0
       0.5
top_level_uf@analyzerDocs.rs:329/
             random_repeater@analyzerDocs.rs:224        3            0.0
    0.4
top_level_uf@analyzerDocs.rs:329/
             sub1@analyzerDocs.rs:225                   3            0.0
    0.1
top_level_uf@analyzerDocs.rs:329/sub1@analyzerDocs.rs:225/
             ub2@analyzerDocs.rs:202                    2            0.0
    0.1
copyOut@analyzerDocs.rs:331                             1            0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-1 1            0.0
    0.0
top_level_uf@analyzerDocs.rs:329/
             esd_interact@analyzerDocs.rs:230.foreach-2 1            0.0
    0.0
top_level_uf@analyzerDocs.rs:329/

 no_intrinsic@analyzerDocs.rs:226          5         0.0      0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             random_repeater@analyzerDocs.rs:210        3            0.0
    0.0
top_level_uf@analyzerDocs.rs:329/no_intrinsic@analyzerDocs.rs:226/
             sub2@analyzerDocs.rs:211                   2            0.0
    0.0

DCV_USER_FUNCTION_LOOP: DISCRETE DCV_USER_FUNCTION_VALUES: CUMULATIVE


---------------------------------------------------------------------------
-----------
```

- `DCV_HEADER_FORMAT`

  Set this variable to control the alignment of the Runtime Summary, Run Information, and Efficiency Metrics sections in the dplog_summary.txt file. The valid settings are `PINCH`, `SPLIT`, `LEFT`, `RIGHT`, and `CENTERED`. The default is `CENTERED`.

- `DCV_PROGRESS_SECONDS`

  Set this variable to change the frequency of Progress Line updates. The value must be an integer. The default is `60`.

- `DCV_PROGRESS_PERCENT`

Set this variable to change the frequency of Progress Line percentage updates. The value must be an integer. The default is `10`.

- `DCV_UNICODE`

  Set this variable to `1` to allow the DCV Analyzer tool to print Unicode characters in the output report files. When you set this variable, the tool replaces `+INF` with the infinity character (∞).

  Note:

  Before setting this variable, make sure that everything you are using (shell, display, text editor, and so forth) supports Unicode characters.

The `DCV_PROGRESS_SECONDS` and `DCV_PROGRESS_PERCENT` variables work together. For example, if you set the seconds to `60` and the percent to `50`, and the run takes approximately three minutes, updates occur at 60 seconds, at 90 seconds (due to the 50 percent setting), at 120 seconds, and at 180 seconds.

In addition, setting these variables to artificially high values (more seconds than you think the job should take or a percentage equal to 100 or more) effectively disables the timer.

Analyzing and Comparing Performance

For performance analysis, the DCV Analyzer tool takes input from a distributed log file and stores the data in an SQLite database, named dplog.db, in the current working directory. Then, the tool analyzes the data and generates reports.

By default, the tool provides summary information for performance analysis in a summary report file, named dplog_summary.txt, and detailed information about the user functions from the runset in a user-functions file, named userfunction_summary.txt. The summary report file contains runtime statistics, efficiency metrics, and tables highlighting discrete aspects of the run, such as the runset commands that took the longest or used the most memory.

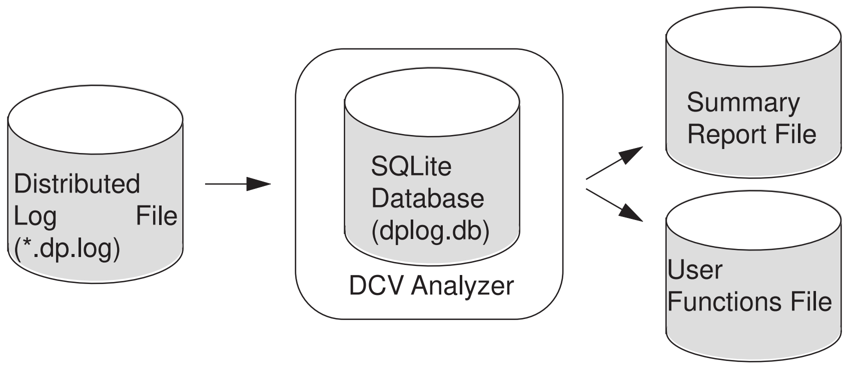
For example, you can enter the following command:

```
# dcv_analyzer runset1.dp.log
```

The tool generates the SQLite database, the summary report file, and the user-functions file in the current working directory. Most of the runtime involves creating the SQLite database. For subsequent runs with the same runset data, you can use the `-r` command-line option to reanalyze the data and regenerate or add reports from the same SQLite database.

Figure 1 illustrates this flow.

Figure 1 DCV Analyzer Performance Analysis Flow



For example, to increase the number of table rows to 15 in the summary report file, you can enter

```
# dcv_analyzer -c 15 -r
```

The tool generates additional output files depending on the options that you include on the `dcv_analyzer` command line.

- Table report files

  Use the `-e` command-line option to generate table-based report files, named *table-name*_table.tsv, that provide data from the database tables. You can import these tab-separated values files into the Microsoft Excel tool for further analysis.

- Command history files

  Use the `-l`, `-f`, `-v`, and `-F` command-line options to retrieve command dependency information for specified command keys. The tool generates an output file for each command key. Each file name is based on the following inputs: <*keyword*>_<*command_key*>_<*first_word*>.txt. See Table 1 for more information.

For example, to learn more about command key 1234.0.0 (a rule that maps to command key 5678.0.0), export the table-based reports in tab-separated-value files by entering the following command after loading dplog.db:

```
% dcv_analyzer -l 1234.0.0 -v "some description" -e -r
```

This creates the following new files: *dplog*_table.tsv, *cmdchain*_1234.0.0.txt, and *cmdchain*_5678.0.0_some description.txt

For more information about the performance analysis output files, see the following sections:

- Summary Report File
- User-Functions File

- Table Report Files
- Command Chain and Full History Files

Summary Report File

The summary report file is the main DCV Analyzer output file. This file contains the performance information highlighted by each run.

The top of this report contains summary information about

- The method and tool versions used to run the DCV Analyzer and IC Validator tools
- The elapsed time and the sum of the elapsed, user, and system times
- The peak and average memory experienced by each host, the number of CPUs used, and the total physical memory available on the host

  The average memory is calculated by adding, for each a periodic interval, the peak memory divided by the interval period. The memory is reported in gigabytes (GB).
- Run information that shows many basic statistics for the run

The IC Validator tool reports time measurements in elapsed time, check time, user time, and system time. Table 2 describes these time measurements.

Table 2 IC Validator Time Measurements

Type	Description
Elapsed time	The time that you can measure with a clock while the job runs (also referred to as the "wall time"). The tool retrieves elapsed times from an independent timer that prints timestamps during the run. The Sum of Command Elapsed Time value is the sum of all the individual commands. For a single-CPU run, the elapsed time is equal to the sum of elapsed times. For multiple-CPU runs, parallel processing makes the sum of elapsed times greater than the IC Validator elapsed time.
Check time	The time measured by a command and reported as `Check Time=`. Most commands report the check time; however, some commands use a different descriptor instead of `Check` and some commands do not report this time. Some engine commands may execute a second command that the master scheduler is not aware of (such as a hierarchical read), in these cases more than one "Check Time=" line is present. When this happens The DCV Analyzer tool attempts to store the main command in the main table and stores the other commands in a secondary table. It is possible for Elapsed and Check times to be quite different in these cases. In most other cases, the check and elapsed times should be within one second of each other, making them effectively the same.

Table 2 IC Validator Time Measurements (Continued)

Type	Description
User time	The time spent by the CPU doing work for the IC Validator tool. Threading makes this number higher than the elapsed time for a single command. These times are reported as `User Time=` for each command. The Sum of Command User Time value is the sum of this number for all commands.
System time	The time reported as `Sys=` for each command. System times represent the amount of time the operating system spends helping the IC Validator tool, for example, to copy a file. In general, system times are relatively low. High system times usually indicate that machine resource contention is affecting the IC Validator runtime. For example, the machine might be paging from high memory usage, or the CPU load average might exceed the amount of available processors on the machine. These problems can exist either because of the current IC Validator run or because of unrelated processes running on the machine.

In addition to the runtime and memory, the statistics for an IC Validator run can include information about the parse time, assigned layers, number of commands used in the run, number of DRC rules, and disk statistics. Table 3 describes these IC Validator run statistics.

Table 3 IC Validator Run Statistics

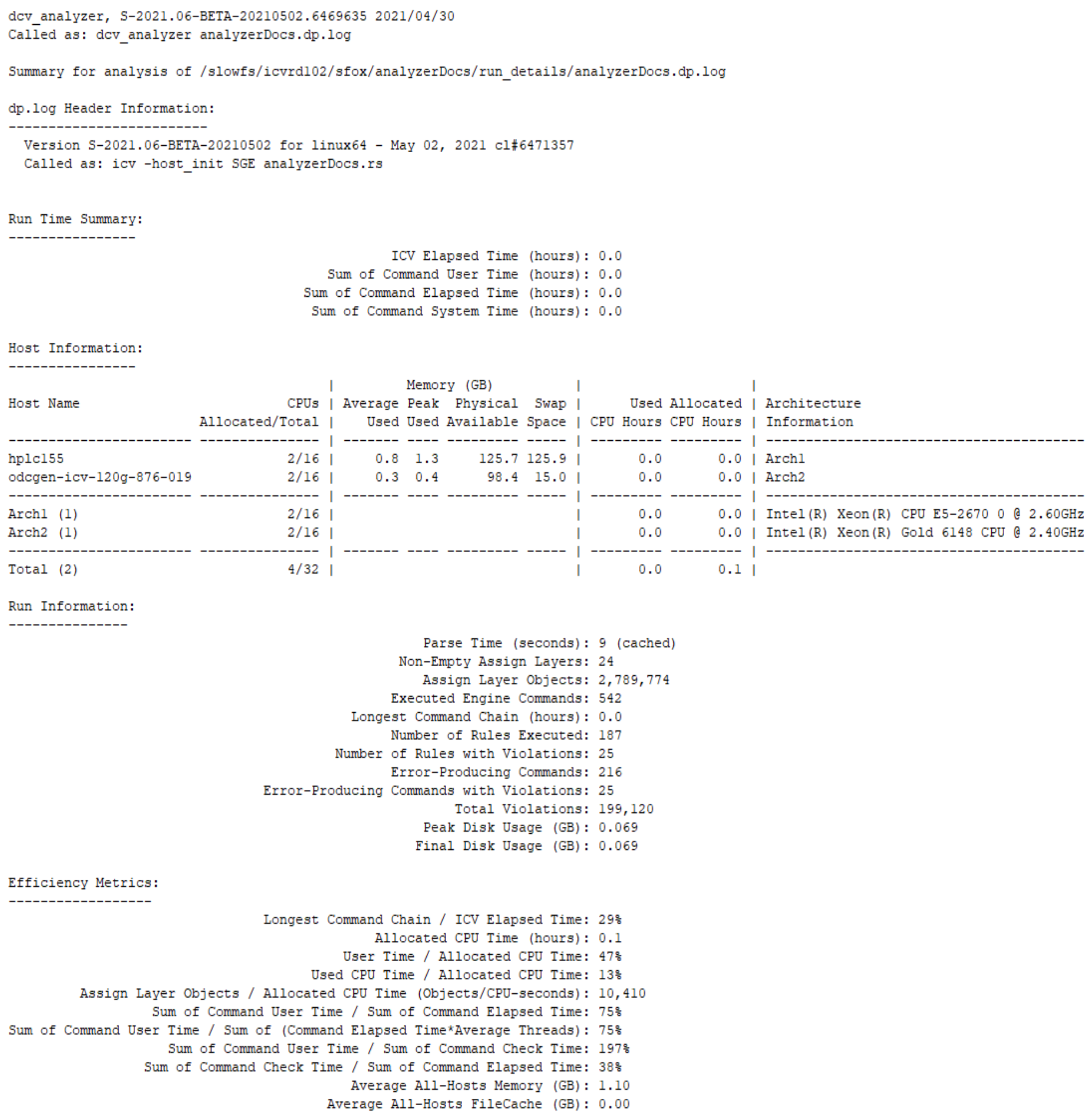
Type	Description
Parse time	The amount of time the IC Validator tool took to parse the runset. (Cache hits are shown in parentheses.)
Non-empty assign layers	The count of layers in `assign()` function calls that contain at least one object.
Assign layer objects	The sum of all the data read in from `assign()` function calls.
Executed engine commands	A count of the discrete processes the engine performed to complete the run. This count includes the generic processes that are independent of the runset and all of the processes directed by the runset. The same runset might produce different numbers depending on the command-line options that you use to run the IC Validator tool (the number of CPUs) and any preprocessor directives or other flow-control and empty-layer optimizations that the tool performs during the run.
Longest Command Chain	The longest serial path of dependencies in the run. This generally depicts the runtime performance threshold.

Table 3 IC Validator Run Statistics (Continued)

Type	Description
Number of rules executed	The number of unique text strings that can potentially write data to the LAYOUT_ERRORS file. A rule in PXL notation is defined as: `Rule1 @= { @ "some-description"; pxl_function( layers, constraints, options); }` These rules do not have to be proper DRC rules. They can be ERC checks or debug statements that you include in the runset.
Number of rules with violations	Shows the number of rules in the runset that produced violations.
Error-producing commands	The number of engine commands with the following comment in the distributed log file: `Comment: "some-description"` The number of error-producing commands might exceed the number of rules due to duplicate comments in the runset, looping, or other engine optimizations that can occur for more complex commands.
Violation	Any output shape reported by the error-producing commands. The DCV Analyzer tool shows counts for both the rules and error-producing commands and the total violations found.
Disk usage	Reports the amount of space at the peak and end of the IC Validator run.

All efficiency metrics should be well defined by the equations used in the file to derive them and should be consistent with the terms described in Table 3.

The following example shows the summary information at the top of the summary report file.



```
dcv_analyzer, S-2021.06-BETA-20210502.6469635 2021/04/30
Called as: dcv_analyzer analyzerDocs.dp.log

Summary for analysis of /slowfs/icvrd102/sfox/analyzerDocs/run_details/analyzerDocs.dp.log

dp.log Header Information:
-------------------------
  Version S-2021.06-BETA-20210502 for linux64 - May 02, 2021 cl#6471357
  Called as: icv -host_init SGE analyzerDocs.rs


Run Time Summary:
----------------
                                                ICV Elapsed Time (hours): 0.0
                                        Sum of Command User Time (hours): 0.0
                                     Sum of Command Elapsed Time (hours): 0.0
                                      Sum of Command System Time (hours): 0.0

Host Information:
----------------
                                        |         Memory (GB)          |                     |
Host Name                          CPUs | Average Peak Physical  Swap |      Used Allocated | Architecture
                        Allocated/Total |    Used Used Available Space | CPU Hours CPU Hours | Information
----------------------- --------------- | ------- ---- --------- ----- | --------- --------- | ----------------------------------------
hplc155                            2/16 |     0.8  1.3     125.7 125.9 |       0.0       0.0 | Arch1
odcgen-icv-120g-876-019            2/16 |     0.3  0.4      98.4  15.0 |       0.0       0.0 | Arch2
----------------------- --------------- | ------- ---- --------- ----- | --------- --------- | ----------------------------------------
Arch1 (1)                          2/16 |                              |       0.0       0.0 | Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
Arch2 (1)                          2/16 |                              |       0.0       0.0 | Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
----------------------- --------------- | ------- ---- --------- ----- | --------- --------- | ----------------------------------------
Total (2)                          4/32 |                              |       0.0       0.1 |

Run Information:
---------------
                                                    Parse Time (seconds): 9 (cached)
                                                 Non-Empty Assign Layers: 24
                                                    Assign Layer Objects: 2,789,774
                                                Executed Engine Commands: 542
                                           Longest Command Chain (hours): 0.0
                                                Number of Rules Executed: 187
                                         Number of Rules with Violations: 25
                                                Error-Producing Commands: 216
                                Error-Producing Commands with Violations: 25
                                                        Total Violations: 199,120
                                                    Peak Disk Usage (GB): 0.069
                                                   Final Disk Usage (GB): 0.069

Efficiency Metrics:
------------------
                                Longest Command Chain / ICV Elapsed Time: 29%
                                              Allocated CPU Time (hours): 0.1
                                          User Time / Allocated CPU Time: 47%
                                      Used CPU Time / Allocated CPU Time: 13%
         Assign Layer Objects / Allocated CPU Time (Objects/CPU-seconds): 10,410
                  Sum of Command User Time / Sum of Command Elapsed Time: 75%
Sum of Command User Time / Sum of (Command Elapsed Time*Average Threads): 75%
                    Sum of Command User Time / Sum of Command Check Time: 197%
                 Sum of Command Check Time / Sum of Command Elapsed Time: 38%
                                           Average All-Hosts Memory (GB): 1.10
                                        Average All-Hosts FileCache (GB): 0.00
```

The summary information can also include a Qualitative Analysis section that appears after the Efficiency Metrics section if the tool finds any of the following conditions. Consider the messages in this section first when analyzing the run.

- The number of allocated CPUs exceeds the number of available CPUs on the host
- The amount of memory used on a host exceeds the amount available

- A recommendation to increase or decrease the number of CPUs to optimize efficiency
- Deferred commands exceed 10 percent of the elapsed time on all hosts
- The log file contains a message about the disk filling up

  If this happens without messages in the log file, the DCV Analyzer tool cannot detect it.
- Any error messages reported from the distributed log file should be reported here as well.

Following the Qualitative Analysis section of the report, if a WARNING message is printed to stdout stating that the log file is incomplete, the tool assumes the job is still running, even if the job crashed or the log file was otherwise copied or edited. A table is printed that shows each host the job is running on as well as some basic statistics about memory and disk usage.

These issues concern only the IC Validator process for which the log is run. If the host is running other IC Validator tools, those tools are unknown. The second table printed in the log file shows all of the commands currently running, how long they have been running, and the amount of memory the IC Validator tool expects each of the commands to consume. The Elapsed time for each of these commands is calculated based on the timestamp of when the command started to the last recorded timestamp in the log file. If it has been a while since any timestamps were added to the log file, that time is not reflected in the Elapsed time calculation.

The primary part of the report contains a series of two types of tables that show detailed information about specific parts of the run. The number of rows in each table is controlled by the `-c` command-line option. The default is 10.

- Command-key tables highlight information about discrete commands and always contain a Command Key column. You can trace the data in this column to a specific instance of a command in the distributed log file.
- Grouped-row tables contain rows that are grouped for statistical analysis by a common quality, such as input layers or intrinsic types. These rows are aggregates between many commands and cannot be traced to specific instances.

The report also contains the following tables:

- Commands sorted by Check Time

  This table shows the discrete commands with the longest check times, as shown in Example 5.

Example 5 Table Highlighting Information About Discrete Commands

```
Commands sorted by Check Time:
---------------------------------------------------------------------------
-----------
```

```
Check Time User    Memory  Runset
 Command
  (min)    Time    (GB)    Text
 Key
           (min)
   0.3     0.3    0.2  large = external1(lay, distance <= 2 * dist, exte
 475.0.0
   0.2     0.2    0.1  internal1(M1, < 0.14, extension = RADIAL, relatio
 590.0.0
   0.1     0.2    0.1  m1e2_m1e31_x = enclose_edge(CO, M1, < 0.05, exten
 677.0.0
   0.1     0.1    0.1  external1(M1, < 0.14, extension = RADIAL, relatio
 591.0.0
   0.1     0.1    0.1  small = external1(lay, distance <= dist, extensio
 474.0.0
   0.1     0.1    0.1  via1e21_y = enclose_edge(VIA1, M1, < 0.05, extens
 687.0.0
   0.1     0.1    0.1  m1e2_m1e32_y = touching_edge(CO, m1e2_m1e31_x)
 680.0.0
   0.1     0.1    0.1 -=_read_library_segment Intrinsic, RunsetText N/A
 297.0.0
   0.1     0.1    0.1  output = internal_corner1(red, distance < 3.0, ty
 482.0.0
   0.0     0.0    0.1  M2 = assign({ { 12 } })
 358.0.0
---------------------------------------------------------------------------
-----------
```

- Commands sorted by User Time

  This table shows the discrete commands with the longest user times.

- Commands sorted by Memory

  This table shows the discrete commands with the highest peak memory.

- Commands sorted by Work

  This table shows the discrete commands that use the largest combination of peak memory and user time. These commands use the most overall computer resources.

  ```
  Work = peak memory * user time
  ```

- Runset Commands (including the dependent, automatically-inserted commands) sorted by Total Elapsed Time

  This table shows the total runtime driven by each line of runset text in the runset; the discrete, dependent commands affected are listed in the dependent, automatically-inserted Command Key column.

  Lines which are called multiple times (via loops or function reuse) are separated, as shown by the User Function column. This total includes any preprocessed commands the engine requires to run immediately before running the command, as well as any higher-level commands that are, by default, broken into more discrete intrinsics.

- Commands sorted by Gain

  This table shows commands in which the amount of output shapes have the largest proportional increase from the amount of input shapes to the command:

  ```
  Gain= Output-Objects/Input-Objects
  ```

  Output-Objects represents the sum of data being written to other group files for internal use and the violations sent to PYDB.

  Downstream User Time is the amount of CPU time consumed by all commands that depend on the command in question. This is also expressed as a percentage of the Total User Time of the run.

- Error-Producing commands sorted by Violation Count

  This table shows the error-producing commands that report the most violations based on the distributed log file. Because of hierarchy or error limit settings, the counts in this table might not match the counts in the *cell*.LAYOUT_ERRORS file. The long-pole times represent the slowest path through the dependent commands for each rule.

- Dangling Commands sorted by Long Pole Time

  This table shows the dangling commands, which are commands that do not have child commands and are not producing a violation comment or creating any data. Ideally, this table should not appear because these commands should be optimized away.

- Error-Producing Commands sorted by Long-Pole Time

  This table shows the error-producing commands that cause the run to go the slowest.

  Upstream User Time shows the total amount of CPU power required to process all the dependencies of the rule.

  Restarted Commands shows commands that the tool had to stop and restart in the given long-pole chain, most likely due to memory contention on the host, which the command started. Commands that are restarted retain the same command key.

- Data-Creating Commands sorted by Long-Pole Time

  This table shows the data-creating commands, which are commands without a violation comment that write a file to the output directory. This file can be a layout file, a database for the Synopsys StarRC™ tool, or an intermediate file for an LVS run.

- Error-Producing Commands sorted by Maximum Memory

  This table shows which error-producing commands have commands in their paths that consume the most memory.

- Slowest User Functions sorted by User Time

  Slowest User Functions sorted by Elapsed Time

  These two tables group runtimes by custom PXL functions defined in the runset. The tool sorts the tables based on the times used to group user functions defined by the runset coder. One table is sorted by the CPU (user) time; the other table is sorted by the elapsed time.

  The `DCV_USER_FUNCTION_LOOPS` and `DCV_USER_FUNCTION_VALUES` environment variables control how the information is displayed, and how they are printed in the footer of the each table. For more information, see the Environment Variables section. By default, nested user functions show discrete numbers, not cumulative numbers. In other words, if user function A contains user function B, the effects of the IC Validator intrinsics in function B are not cumulative for the report of function A. The numbers indicate which line of the runset they come from.

  For information about limitations and deciphering the function name, see the User-Functions File section.

The Memory Stats section of the report shows all of the commands that ran for each host when the host reached its peak memory.

Runset and input database MD5 checksums are tracked by the DCV Analyzer tool. This data is useful when you compare multiple runs to determine if the runset or input data has changed. If the checksums match, the input data is identical. If the checksums do not match, they are most likely different, although the manner in which they have changed may or may not be pertinent to the analysis that is being performed.

For all IC Validator runs using version M-2017.06 or later, runset checksums are printed for each file in the runset as well as a total checksum for all of the files in the runset.

The runset checksum is sensitive to changes in any evaluated IC Validator logic, and it changes with environment variables that are referenced in the PXL file. The runset checksum is insensitive to comments and whites-space changes.

Additionally, if the runset is configured using `run_options(report_streamfile_information={MD5SUM})` or the `-rsi MD5SUM` command-line option, the input GDSII or OASIS database checksum information is printed.

User-Functions File

The user-functions file, userfunction_summary.txt, contains detailed information about the user functions contained in the file. You can control how this information is displayed by using the `DCV_USER_FUNCTION_LOOPS` and `DCV_USER_FUNCTION_VALUES` environment variables. For more information, see the Environment Variables section.

Caution:

> The use of encryption or the published keyword in the runset obfuscates details in the distributed log file, and the DCV Analyzer tool is unable to present all of the information.

The file shows the hierarchy of all user functions contained within a runset. Each row shows statistical information for all native intrinsics in a user function. To get totals, you must manually add the nested functions to the level of interest. The format is

```
function_name@runset_file_name:line_number.loop_iteration
```

Table Report Files

The table report files, named *table-name*_table.tsv, contain raw data from the database tables used for all of the queries. These files are meant to be used only for debugging or advanced analysis. You can import these tab-separated values files into the Microsoft Excel tool.

Command Chain and Full History Files

The command chain files show the longest serial path of dependencies to reach a given command key. The full history files show every command that was executed (in order) to reach the command of interest (superset of equivalent command chain file). For each command, these dependencies include

- Elapsed time
- Gap time (present only in the command chain files)

  The gap time is the start time of the current command minus the end time of the previous command. For the first command, the gap time is the same as the start time because the previous end time is 0.
- Memory the command used
- Total of all the output geometric objects and the associated gain

The runset text, truncated at 75 characters with an ellipsis (...), and the parent command keys for each command in the chain appear on the right side of the report. Consider that input counts come from alternate parent commands, which are listed, and not just from the previous command key listed in the file. If you are interested in a parent command, you can regenerate its equivalent file by using the `-r` and `-l` command-line options in a subsequent run. The bottom of the file provides statistical information relative to the chain.

Example 6 shows an example of a command chain file.

Example 6   Command Chain File Example

```
Command    Elapsed Gap Time Memory      Output Gain Runset
 Parent
   Keys Time (min)     (min)   (GB) Object Count      Text
 Command Keys
297.0.0       4.21    0.08   0.31          0    0 -=_read_library_segment
                                                    Intrinsic, RunsetText N/A=
301.0.0       0.38    0.00   0.62           0   0 -=_create_hierarchy_tree
                                                    Intrinsic, RunsetText N/A=-
                                                                     297.0.0
355.0.0       2.77    0.16   0.75   30,222,080 +INF M1 = assign({ { 11 } })
                                                             301.0.0 297.0.0
356.0.0       0.53    6.05   0.30   30,236,670 1.00 M1 = assign({ { 11 } })
                                                             355.0.0 320.0.0
416.0.0       1.07    1.69   0.53   242,374 0.00 Generating associated layer
                                                   temp.associated
                                                             386.0.0 383.0.0
                                                             323.0.0 356.0.0
694.0.0       11.26   10.17  2.07   121,741,385 1.57 m1e2_m1e31_x =
 enclose_edge(
                                                     CO, M1, < 0.05,
                                                     extension = NONE,
                                                         look_thru ...
                                                             425.0.0 356.0.0
                                                             323.0.0 416.0.0
                                                                     383.0.0
696.0.0       0.74    10.96   0.71  186,539 0.00 Generating associated layer
                                                   m1e2_m1e32_y.associated
                                                             694.0.0 323.0.0
                                                                     383.0.0
697.0.0       5.11    4.12   1.03   121,780,908 0.72 m1e2_m1e32_y =
                                                   touching_edge(
                                                     CO, m1e2_m1e31_x
                                                   )
                                                             696.0.0 694.0.0
                                                             695.0.0 323.0.0
                                                                     383.0.0
698.0.0       0.65    0.43   0.53   20,919 0.00 Generating associated layer
                                                   temp.associated
                                                             697.0.0 323.0.0
699.0.0       0.80    0.00   0.16   0 0.00 length_edge(
                                                     m1e2_m1e32_y, < 0.001
                                                   )
                                                             697.0.0
 323.0.0
                                                                      698.
0.0

Sum of Chain Elapsed Times:          27.52  minutes  (includes Restarted
 Times)
Sum of Chain Gap Times:              33.66  minutes
Sum of Chain Restarted Times:         0.00  minutes
Sum of Chain User Times:             26.17  minutes
```

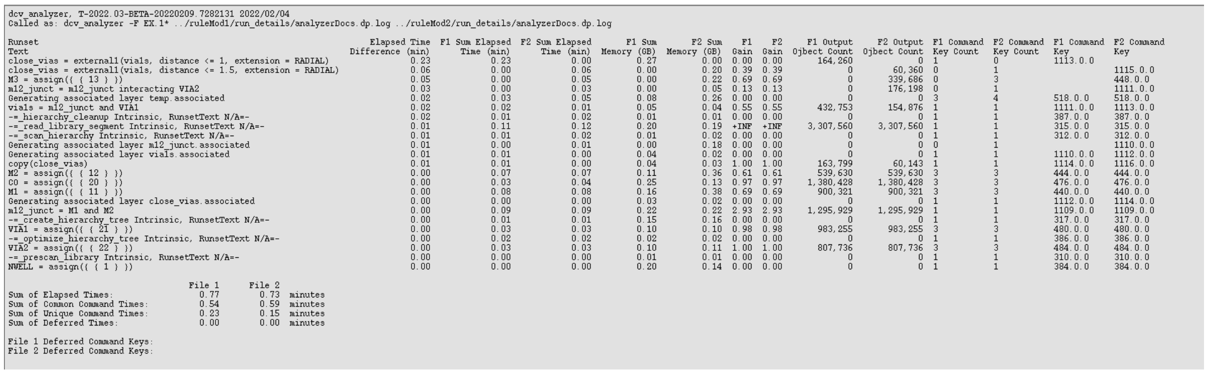
The data in the command chain file is similar to the data in the summary report file. However, the command chain file is organized by execution time rather than by statistics. Locating issues created by linear dependencies might be easier in this format.

Full History Compare File

You can generate a fullHist_compare_*rule*.txt file when you compare two distributed log files and use the `-F/-full_cmd-violation` command-line option in the tool. This file is similar to the fullHist_*cmdkey*.txt file, however it takes the RunsetText and compares and sorts it by the runtime difference. This is beneficial when comparing the performance or a rule or proposed runset change to a rule. The runtime, memory, and gain information is easily available. Commands that appear in only one of the runs have their respective values compared to 0 or an empty string. A summary of each of the Common and Unique command is printed at the bottom of the file, as well as any information about relevant deferred commands.

Notice that only one of these compare files is generated per run. This file contains all of the commands from the relevant rules. For example, `-F "M*"` takes every rule that starts with an uppercase M as well as all of the commands needed to generate and compare them. Similarly,`-F "M1*,M2*"` takes every rule that starts with M1 or M2 and does the same. Use longer strings to generate rule-specific files. Use `-r` to quickly generate multiple combinations of `-F`.

Figure 2 Full History Compare File



```
dcv_analyzer, T-2022.03-BETA-20220209.7282131 2022/02/04
Called as: dcv_analyzer -F EX.1* ../ruleMod1/run_details/analyzerDocs.dp.log ../ruleMod2/run_details/analyzerDocs.dp.log

Runset                                                                  Elapsed Time  F1 Sum Elapsed  F2 Sum Elapsed       F1 Sum       F2 Sum    F1    F2     F1 Output     F2 Output  F1 Command  F2 Command  F1 Command  F2 Command
Text                                                                Difference (min)      Time (min)      Time (min)  Memory (GB)  Memory (GB)  Gain  Gain  Ojbect Count  Ojbect Count  Key Count   Key Count   Key         Key
close_vias = external1(via1s, distance <= 1, extension = RADIAL)                0.23            0.23            0.00         0.27         0.00  0.00  0.00       164,260             0  1           0           1113.0.0
close_vias = external1(via1s, distance <= 1.5, extension = RADIAL)              0.06            0.00            0.06         0.00         0.20  0.39  0.39             0        60,360  0           1                       1115.0.0
M3 = assign({ { 13 } })                                                         0.05            0.00            0.05         0.00         0.22  0.69  0.69             0       339,686  0           3                       448.0.0
m12_junct = m12_junct interacting VIA2                                          0.03            0.00            0.03         0.00         0.05  0.13  0.13             0       176,198  0           1                       1111.0.0
Generating associated layer temp.associated                                     0.02            0.03            0.05         0.08         0.26  0.00  0.00             0             0  3           4           518.0.0     518.0.0
via1s = m12_junct and VIA1                                                      0.02            0.02            0.01         0.05         0.04  0.55  0.55       432,753       154,876  1           1           1111.0.0    1113.0.0
-=_hierarchy_cleanup Intrinsic, RunsetText N/A=-                                0.02            0.01            0.02         0.01         0.01  0.00  0.00             0             0  1           1           387.0.0     387.0.0
-=_read_library_segment Intrinsic, RunsetText N/A=-                             0.01            0.11            0.12         0.20         0.19  +INF  +INF     3,307,560     3,307,560  1           1           315.0.0     315.0.0
-=_scan_hierarchy Intrinsic, RunsetText N/A=-                                   0.01            0.01            0.02         0.01         0.02  0.00  0.00             0             0  1           1           312.0.0     312.0.0
Generating associated layer m12_junct.associated                                0.01            0.00            0.01         0.00         0.18  0.00  0.00             0             0  0           1                       1110.0.0
Generating associated layer via1s.associated                                    0.01            0.01            0.00         0.04         0.02  0.00  0.00             0             0  1           1           1110.0.0    1112.0.0
copy(close_vias)                                                                0.01            0.01            0.00         0.04         0.03  1.00  1.00       163,799        60,143  1           1           1114.0.0    1116.0.0
M2 = assign({ { 12 } })                                                         0.00            0.07            0.07         0.11         0.36  0.61  0.61       539,630       539,630  3           3           444.0.0     444.0.0
C0 = assign({ { 20 } })                                                         0.00            0.03            0.04         0.25         0.13  0.97  0.97     1,380,428     1,380,428  3           3           476.0.0     476.0.0
M1 = assign({ { 11 } })                                                         0.00            0.08            0.08         0.16         0.38  0.69  0.69       900,321       900,321  3           3           440.0.0     440.0.0
Generating associated layer close_vias.associated                               0.00            0.00            0.00         0.03         0.02  0.00  0.00             0             0  1           1           1112.0.0    1114.0.0
m12_junct = M1 and M2                                                           0.00            0.09            0.09         0.22         0.22  2.93  2.93     1,295,929     1,295,929  1           1           1109.0.0    1109.0.0
-=_create_hierarchy_tree Intrinsic, RunsetText N/A=-                            0.00            0.01            0.01         0.15         0.16  0.00  0.00             0             0  1           1           317.0.0     317.0.0
VIA1 = assign({ { 21 } })                                                       0.00            0.03            0.03         0.10         0.10  0.98  0.98       983,255       983,255  3           3           480.0.0     480.0.0
-=_optimize_hierarchy_tree Intrinsic, RunsetText N/A=-                          0.00            0.02            0.02         0.02         0.02  0.00  0.00             0             0  1           1           386.0.0     386.0.0
VIA2 = assign({ { 22 } })                                                       0.00            0.03            0.03         0.10         0.11  1.00  1.00       807,736       807,736  3           3           484.0.0     484.0.0
-=_prescan_library Intrinsic, RunsetText N/A=-                                  0.00            0.00            0.00         0.01         0.01  0.00  0.00             0             0  1           1           310.0.0     310.0.0
NWELL = assign({ { 1 } })                                                       0.00            0.00            0.00         0.20         0.14  0.00  0.00             0             0  1           1           384.0.0     384.0.0

                                 File 1      File 2
Sum of Elapsed Times:              0.77        0.73  minutes
Sum of Common Command Times:       0.54        0.59  minutes
Sum of Unique Command Times:       0.23        0.15  minutes
Sum of Deferred Times:             0.00        0.00  minutes

File 1 Deferred Command Keys:
File 2 Deferred Command Keys:
```

Runtime Predictions

The DCV Analyzer tool attempts to predict runtimes the IC Validator tool might encounter when running a job with the same runset and input database. These runtime predictions can be useful for targeting a particular end time with subsequent runs, or when trying to maximize CPU resources, as eventually diminishing returns are reached by continuing to scale the same job too many times.

There are three to six rows of data displayed based on the input number of CPUs used for the original IC Validator job. Estimates are calculated via machine learning models. Some generic models are provided with the IC Validator release. However, if you install the Synopsys Insight tool, you can create custom models. See the *Quick Start Guide for*

SYNOPSYS ON-SITE ML PLATFORM Installation on the Synopsys AI Central site for more information.

To create these models, set `dcv_analyzer -train` to automatically download the relevant data from the local Insight database and train the new models by placing them in your working directory. Specify the `DCV_ML_MODELS_DIR` environment variable to use these models.

Example 7 Runtime Predictions

```
-----------------------------------------------------------------------
Runtime Predictions for different host configurations (comparable
 hardware)
-----------------------------------------------------------------------
          Predicted
  CPU     Runtime (hrs)
*   4             0.02
    8             0.02
   12             0.02

*  actual results from this run
```

Comparing Performance

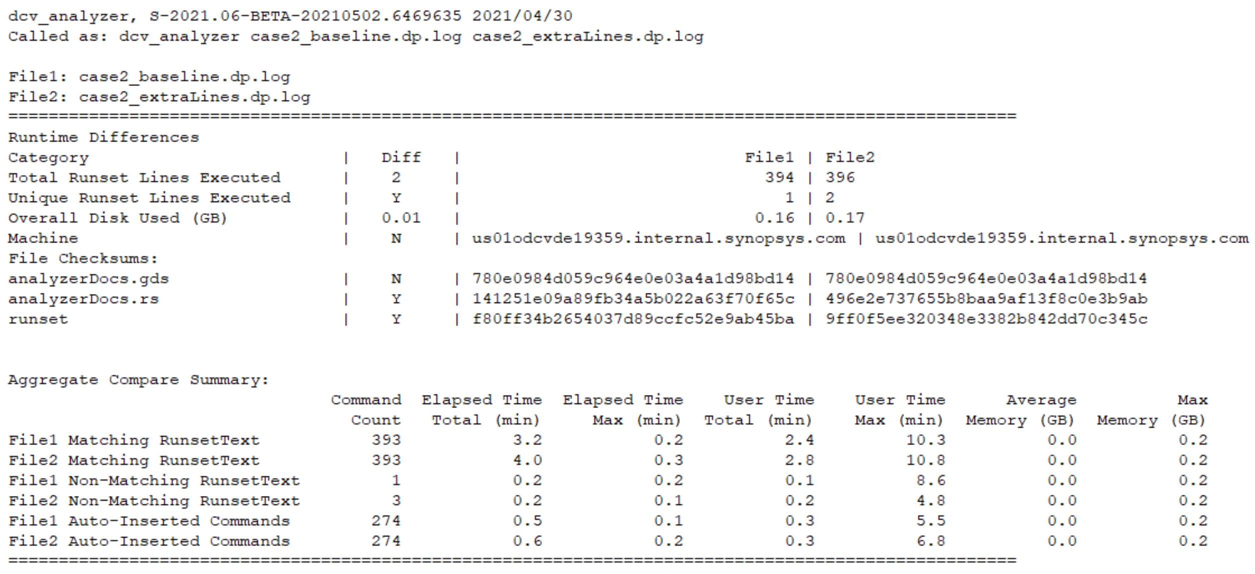
When two dplog files are provided as inputs to the DCV Analyzer tool, a performance comparison takes place. As the first step in this process, the DCV Analyzer tool produces the performance analysis output files normally generated, as if a single dplog file was specified. The difference is that output files associated with the second dplog file have a “2” suffix attached, for example, dplog_summary.txt becomes dplog_summary2.txt. This same suffix is applied to any additional output files associated with specific command-line options.

The second step in the process is the performance comparison. The DCV Analyzer tool generates a dplog_compare.txt file, which attempts to make more relevant comparisons than a simple diff of the output files typically generates. This text file provides comparison reports that can be used to determine differences in performance data...

The first table in the dplog_compare file contains summary information from each log file. This table can vary in size depending on the differences between the runs. Metrics that match or exist only in one file are not reported frequently. The Diff column contains a Y or N for string comparisons and an absolute difference for numerical differences. All available check sums (runset or layout database) are compared here as well. The aggregate compare summary has three sections used to compare statistics: the first section shows a comparison between runset lines that exists in both files, the second section shows runset lines that are unique to each file, and the third sections shows the automatically inserted IC Validator-processed commands that are not directly tied to a line in either runset. For runs in which similar results can be expected, this table helps clarify where differences might be found.

As the input runs become more divergent with things like runset options, input data, IC Validator version, and runset content, the ability to draw reasonable comparisons diminishes. Additionally, depending the differences, some tables might be more relevant than others.

In the following example, File 2 executes two additional lines of text. Notice that the checksums of the input GDS match, but the runset does not. Therefore, it appears a line might have changed in the runset since each one has a unique line or two executed. However, looking at the aggregate compare summary shows negligible differences in any runtime performance.



```
dcv_analyzer, S-2021.06-BETA-20210502.6469635 2021/04/30
Called as: dcv_analyzer case2_baseline.dp.log case2_extraLines.dp.log

File1: case2_baseline.dp.log
File2: case2_extraLines.dp.log
====================================================================================================
Runtime Differences
Category                         |   Diff   |                             File1 | File2
Total Runset Lines Executed      |    2     |                               394 | 396
Unique Runset Lines Executed     |    Y     |                                 1 | 2
Overall Disk Used (GB)           |   0.01   |                              0.16 | 0.17
Machine                          |    N     | us01odcvde19359.internal.synopsys.com | us01odcvde19359.internal.synopsys.com
File Checksums:
analyzerDocs.gds                 |    N     | 780e0984d059c964e0e03a4a1d98bd14 | 780e0984d059c964e0e03a4a1d98bd14
analyzerDocs.rs                  |    Y     | 141251e09a89fb34a5b022a63f70f65c | 496e2e737655b8baa9af13f8c0e3b9ab
runset                           |    Y     | f80ff34b2654037d89ccfc52e9ab45ba | 9ff0f5ee320348e3382b842dd70c345c


Aggregate Compare Summary:
                                 Command  Elapsed Time  Elapsed Time    User Time    User Time      Average          Max
                                   Count   Total (min)     Max (min)  Total (min)    Max (min)  Memory (GB)  Memory (GB)
File1 Matching RunsetText            393           3.2           0.2          2.4         10.3          0.0          0.2
File2 Matching RunsetText            393           4.0           0.3          2.8         10.8          0.0          0.2
File1 Non-Matching RunsetText          1           0.2           0.2          0.1          8.6          0.0          0.2
File2 Non-Matching RunsetText          3           0.2           0.1          0.2          4.8          0.0          0.2
File1 Auto-Inserted Commands         274           0.5           0.1          0.3          5.5          0.0          0.2
File2 Auto-Inserted Commands         274           0.6           0.2          0.3          6.8          0.0          0.2
====================================================================================================
```

The series of tables for the remainder of the file contains similar metrics to those used in a single log file. Keep the following in mind when reviewing this data:

- Each table distinguishes file1 or file2 in the header.

  The column names alias these as F1 and F2, respectively.

- Comparing discrete information between runs can be very difficult. For example, line numbers might not match and command keys can change as such more general comparison points are used.

  Runset text, for example, and each additional file, can have multiple uses of the same line. Therefore, all of these files are summarized for comparison; the count is displayed, and if a command key is given for a line with a count > 1, it is at random and is used only to manually look for more discrete information.

The remaining tables in this file highlight differences in the executed runset text between the files. Even when runsets match exactly, they can report differences due to flow control or empty layer optimizations. These tables are helpful for determining where changes might have occurred, if you expect them to match. If you know the runsets are different, they are not likely to add much value. These tables focus only on the text and line numbers and not the performance.

Analyzing and Comparing Hierarchies

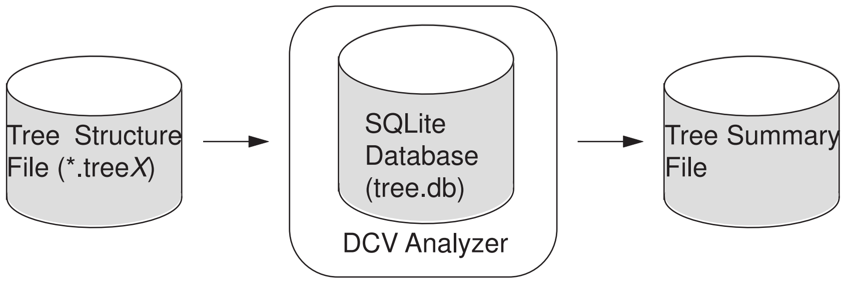
For hierarchy analysis, the DCV Analyzer tool takes input from a tree structure file that you specify and stores the data in an SQLite database, named tree.db, in the current working directory. Then the tool analyzes the hierarchy data and generates reports in the current working directory. By default, the tool provides hierarchy analysis information in a tree summary file named tree_summary.txt.

For example, you can enter the following command:

```
# dcv_analyzer mycell.tree0
```

The tool generates the SQLite database and the tree summary file in the current working directory. Most of the runtime involves creating the SQLite database. For subsequent runs with the same hierarchy data, you can use the `-r` command-line option to reanalyze the data and regenerate or add reports from the same SQLite database. Figure 3 illustrates this flow.

Figure 3 DCV Analyzer Hierarchy Analysis Flow



The hierarchy analysis operation is independent of the performance analysis operation; you can draw any linkage between the results that you want.

For hierarchy comparisons, the DCV Analyzer tool takes input from two tree structure files that you specify and stores the data in two SQLite databases, named tree.db and tree2.db, in the current working directory. By default, the tool provides the following information:

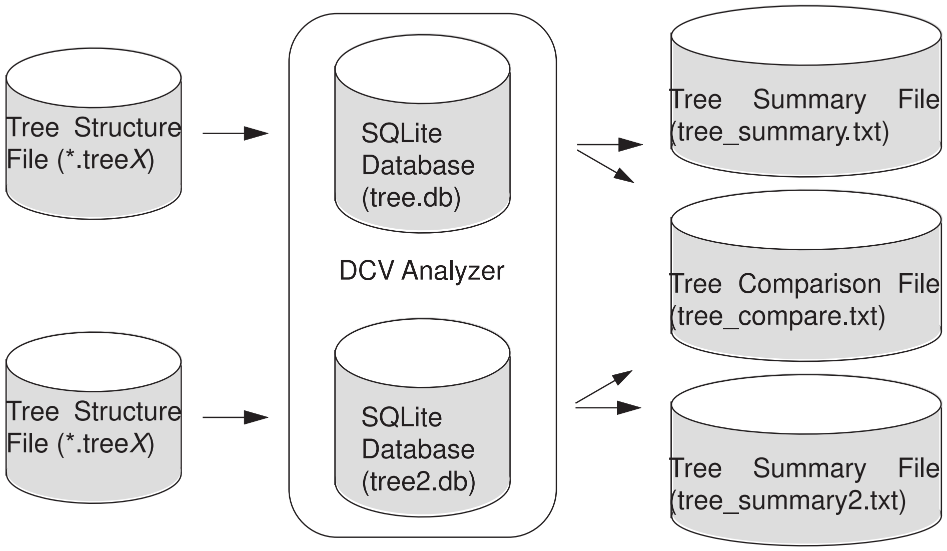
- Hierarchy analysis information in two tree summary files, named tree_summary.txt (from tree.db) and tree_summary2.txt (from tree2.db)
- Hierarchy comparison information in a tree comparison file, named tree_compare.txt, that highlights cross-referenced data between the two tree structure files

The tree structure files do not have to be from the same IC Validator run nor do they have to be of the same type (for example “tree0”). The DCV Analyzer tool can process any two tree structure files. For example, you can enter the following command:

```
# dcv_analyzer mycell.tree0 mycell.tree999
```

The tool generates the SQLite databases and the tree summary files in the current working directory. Most of the runtime involves creating the SQLite databases. For subsequent runs with the same hierarchy data, you can use the `-r` command-line option to reanalyze the data and regenerate or add reports from the same SQLite databases. Figure 4 illustrates this flow.

Figure 4 DCV Analyzer Hierarchy Comparison Flow


Tree Structure File (*.treeX)
SQLite Database (tree.db)
DCV Analyzer
Tree Structure File (*.treeX)
SQLite Database (tree2.db)
Tree Summary File (tree_summary.txt)
Tree Comparison File (tree_compare.txt)
Tree Summary File (tree_summary2.txt)

The tool also generates a reference cells file, reference_cells.txt, that compares only the differences found in the Cell Statistics sections of the tree structure files. When the statistics match for a cell, no data for that cell is included in the reference cells file. If the statistics for a cell appear in only one of the tree structure files, the data for that cell is omitted from the reference cells file.

The tool generates additional output files depending on the options that you include on the `dcv_analyzer` command line.

- Cell information files

  If you use the `-c` command-line option to specify a list of cell names, the tool produces a file named cell_*cell-name*.txt for each cell. These files contain statistical information about the cells.

- Layer information files

If you use the `-L` command-line option to specify a list of layer names or layer number and data type pairs, the tool produces a file named layer_*layer-name*.txt for each layer name and a file named *layer_num*:*data_type*.txt for each layer number and data type pair. These files contain statistical information about the layers.

For more information about these output files, see the following sections:

- Tree Summary Files
- Tree Comparison File
- Cell Information Files
- Layer Information Files

Tree Summary Files

When you specify a single tree structure input file, the DCV Analyzer tool creates a tree summary file named tree_summary.txt. If you specify a second tree structure input file, the tool also creates a file, named tree_summary2.txt, and a tree comparison file, named tree_compare.txt. The tree_summary2.txt file contains the same data for the second tree structure file that tree_summary.txt contains for the first tree structure file. The label "Summary for analysis of ..." appears at the top of each tree summary file to identify which tree structure file the data belongs to. The Design Statistics section in the tree structure file contains data from the header of the tree structure file. Table 4 provides definitions of these design statistics.

Table 4 Tree Structure Design Statistic Definitions

Statistic	Definition
Hierarchical levels	The number of levels in the design.
Unique cells	The number of unique cells in the design.
Hierarchical placements	The total number of placements (SREFs and AREFs) at each level in the design.
Total design placements	The total number of placements (SREFs and AREFs) in the design.
Exploded references	The total number of exploded references in the design.
Exploded placements	The total number of exploded placements in the design.
Total data count	The total number of all polygons, paths, edges, and rectangles in the design.
SREF placements	The total number of cell references in the design.

Table 4 Tree Structure Design Statistic Definitions (Continued)

Statistic	Definition
AREFs	The total number of array references in the design.
AREF placements	The total number of references in each array reference in the design.
Data	The number of data primitives in each cell of the design.
Text	The number of text primitives in each cell of the design.
Via	The number of via cells in the design.

Example 8 shows an example of the Design Statistics section.

Example 8 Design Statistics Section

```
Design Statistics:
-----------------
    Hierarchical Levels:  3
           Unique Cells:  25,101
Hierarchical Placements:  54,224,801
Total Design Placements:  54,224,801
    Exploded References:  0
    Exploded Placements:  0
       Total Data Count:  2,276,754,400
        SREF Placements:  54,224,801
                  AREFS:  0
        AREF Placements:  0
                   Data:  330,756,000
                   Text:  335,400
                    Via:  2,500

 Level  Placement Count  Path Count   Unique Cell Count
 -----  ---------------  ----------   -----------------
     0                1           1                   1
     1              100         100                 100
     2       54,224,700      25,000              25,000

 Depth       Cell Count
 -----       ----------
     0           25,000
     1              100
     2                1
```

The Placement Count is the total number of cells placed at each level of hierarchy.

The Path Count is the number of unique paths to a cell. For example, `Top - L1 - L2` counts as one path even if cell L2 is placed 500 times.

The Unique Cell Count is the number of unique cells that appear at each level of hierarchy. For example, suppose the hierarchy has the following tree structure:

```
TOP        Level=0  Count=   1
  L1A      Level=1  Count=   1
    L2     Level=2  Count= 500
  L1B      Level=1  Count=   1
    L2     Level=2  Count=   1
  L2       Level=1  Count=  10
```

Given this structure, the tree summary file shows the following placement counts:

```
 Level  Placement Count     Path Count   Unique Cell Count
 -----  ---------------     ----------   -----------------
     0                1              1                   1
     1               12              3                   3
     2              510              2                   1
```

Depth is defined as the furthest distance from the leaf cell, which is a cell without child cells. For example, if a design has 10 levels of hierarchy, the top cell always has a depth of 10.

Tree summary files contain the following tables:

- Cells sorted by Total Placements

  This table shows cells that are placed the most times. The placement count is for a specific location in the hierarchy (indicated by the Path), so a cell can appear multiple times in this table.

- Cells sorted by Data

  This table shows cells that contain the most geometric objects (all layers). Text counts are displayed in the table but not included in the sorting.

- Cells sorted by Flat Objects

  This table displays the cells that contribute the most data to the design based on a combination of the number of times they are placed and the amount of geometric objects contained within the cell.

  ```
  Flat Objects = FlatPlacements * (Data + Text)
  ```

- Layers within each Cell sorted by Data

  This table shows the layers by cell that contain the most data. The layer number and data type may not be available for certain data formats.

- Layers sorted by Flat Objects

  Layers sorted by Flat Data

Layers sorted by Flat Text

These tables display the layers that contribute the most data to the design based on a combination of the number of times the cells they are in are placed and the amount of geometric objects assigned to that layer.

```
Flat Objects = FlatPlacements * (DataCount + TextCount)
Flat Data = FlatPlacements * DataCount
Flat Text = FlatPlacements * TextCount
```

Tree Comparison File

When you specify two tree structure input files, the DCV Analyzer tool creates a tree comparison file named tree_compare.txt. You can compare two tree structure files from the same or different IC Validator runs. The format of the tree structure files (Milkyway, NDM, LTL, GDSII, or OASIS) does not matter.

The first two tables in the tree comparison file define the missing cells and extra cells from the second tree structure file, sorted by cell area. By default, only the 10 largest cells are shown, but the table headers indicate the total number of cells. You can use the `-c` command-line option to increase the number of rows in the tables.

Note:

Cells that appear in either the extra cells or missing cells table are not included in the majority of the other tables because they are not in both reports and the tool cannot perform a meaningful comparison.

Many of the tables in the tree comparison file contain the same information as the tables in the tree summary file, with the sorting column presented as a gain comparison from the second input file to the first input file. Because of this switching, the order of the input files on the command line might produce a different set of data for each table. Consider the order carefully before running the tool.

The tree comparison file contains the following tables:

- Matching Cells sorted by Total Placements Gain

  This table calculates the ratio of total placements between the files.

- Matching Cells sorted by Data Gain

  This table calculates the ratio of total data (all layers) within each cell between the files.

- Matching Cells sorted by Flat Objects Gain

  This table shows the ratio of total flat objects in each file.

```
Gain = (FlatPlacements2 * (Data2 + Text2))/( FlatPlacements1 * (Data1 +
 Text1))
```

- Layers in Matching Cells sorted by Data Difference

  This table shows the data by layer and cell difference between the files. The tool uses the absolute values, so the file order should not affect the results.

- Layers within each Cell sorted by Gain

  This table appears twice and shows the same comparison as the previous table, but with the relative differences rather than the absolute differences. The layer and cell names must be the same in both input files for the comparison to be valid. One copy of the table shows the values of the first file to second file comparison, and the other copy of the table shows the second file to first file comparison.

- Matching Layers Flat Objects sorted by Gain

  Matching Layers Flat Data sorted by Gain

  Matching Layers Flat Text sorted by Gain

  These tables show the gain (Flat Objects2/Flat Objects1) between the files. “Data” and “Text” are substituted for “Objects” in their respective tables.

  ```
  Flat Objects = FlatPlacements * (DataCount + TextCount)
  Flat Data = FlatPlacements * DataCount
  Flat Text = FlatPlacements * TextCount
  ```

- Cells sorted by Placement Difference

  This table shows the absolute difference between the two input files of the flat placement count of each cell. Because the tool uses absolute values, the order of the input does not matter. In addition, the missing and extra cells from the first tables should count as “0” in the calculations for this table.

- Cells with changed Status sorted by Flat Placements

  This tables shows matching cells between the input files in which the status does not match. The order is based on the sum of the number of times the cell is placed in each file.

- Status Changed Summary

  This table shows the number of unique pairs of how the status of a given cell might have changed between the input files.

Cell Information Files

When you use the `-c` command-line option to specify a list of cells, the DCV Analyzer tool creates a report file named cell_*cell-name*.txt for each cell in the cell list. If you specify a cell that is not in the data, the tool produces an empty file that contains only table headings.

If you include a wildcard character (*) to specify multiple cell names, you should enclose the entire argument in quotation marks to avoid having the shell interpret the wildcard character (to include files in the current working directory).

For example, `-C "NAND2X0,M*"` produces a file named cell_NAND2X0.txt and a file for each cell with a name that begins with an uppercase M (the cell names are case-sensitive). In this example, if no cells with a name that starts with M exist, the tool produces only the NAND2X0 file.

Cell information files contain the following sections:

- The first section of the report contains statistics, which are similar to the design statistics in the summary report file.
- The second section compares the layers in each cell and the amount of data and text on each layer. The differences reported in this section account for both text and data.
- The third section shows the names of all the immediate child cells with their placement count and status. In this section, the tool compares and calculates differences as appropriate.
- The final section shows all the available paths to the cell of interest in each file, which helps you to determine the parent cells.

You must specify at least one tree structure input file when you use the `-C` command-line option. The DCV Analyzer tool includes data from the input file in the cell information files. If you specify two tree structure files, the tool includes data from both input files side by side, along with a difference column. Differences can be absolute or relative depending on the data, or just an X for a string comparison (Extents and Status).

Layer Information Files

When you use the `-L` command-line option to specify a list of layers, the DCV Analyzer tool creates a layer information file named layer_*layer-name*.txt for each layer in the layer list. Each file contains a list of cells and the number of geometric objects per cell for that layer.

If you include a wildcard character (*) to specify multiple layer names, you should enclose the entire argument in quotation marks to avoid having the shell interpret the wildcard character (to include files in the current working directory).

You must specify at least one tree structure input file when you use the `-L` command-line option. If you specify two tree structure files, the tool includes two lists of cells in the layer information file, one from each tree structure file.

3
DCV Design Trimmer Tool

This chapter explains how to run the DCV Design Trimmer tool.

The IC Validator DCV Design Trimmer tool is a convenience utility that can be used for reducing test case data to aid debugging.

For more information, see the following sections:

- Overview
- Command-Line Options
- Combining Criteria
- Outputs

Overview

There are situations where debugging an issue that involves a huge, complex layout database, such as a full chip, becomes necessary. One iteration of running the database through DRC can take more than one hour to complete for one rule and many hours for a full set of rules. Identifying the source of the problem can take multiple iterations. In these situations, reducing the test case to a smaller case that reproduces the original issue can be both helpful and efficient for debugging purposes.

Although a reduced case can be created manually, this is a tedious process that can generally be automated. The DCV Design Trimmer tool, or Trimmer, provides an automated way to generate reduced test case data from the original test case based on user specifications.

Layout Reduction

The Trimmer provides an automated way to reduce a layout database while preserving the hierarchy of the original database in the reduced database. For the layout reduction, you can choose one criterion or any composition of criteria from the following:

- Select a window
- Select a cell and its subcells
- Unselect cells and their subcells
- Select layers
- Unselect layers
- Select assign layers used to check specific rules

Note:

"Select" and "unselect" are defined by IC Validator terminology. See the `select_window` argument of the `incremental_options()` function in the *IC Validator Reference Manual*, or `-svc`, `-uvc`, `-svn`, `-uvn`, `-sfn`, and `-ufn` in the Command-Line Options section of the *IC Validator User Guide* for examples.

The tool automatically detects the original format of the database. It supports the following formats, including gzipped files:

- GDSII
- OASIS

The tool generates the reduced database in the same format as the original database.

Runset Reduction

The Trimmer provides an automated way to reduce an IC Validator runset. For the runset reduction, you can choose one of the following criteria:

- Select runset content used for specific rules
- Select runset content used for rules that directly or indirectly depend on specific layers

Command-Line Options

The Trimmer command-line syntax is

```
dcv_trimmer [-i path] [-rs path] [options]
```

where

- `-i` is the layout database to reduce.
- `-rs` is the IC Validator runset to reduce.
- `options` are described in Table 5.

A valid command line specifies

1. `-i` or `-rs`.
2. At least one criterion that works on layout databases If it specifies `-i`.
3. At least one criterion that works on runsets if it specifies `-rs`.

If you specify both `-i` and `-rs`, the tool applies layout-compatible criteria on the layout database input and runset-compatible criteria on the runset input. Criteria that can work on both types of inputs are applied to both if both are specified.

Example 9 shows some valid and invalid command-line usage.

Example 9 Command-Line Usage

```
# OK. Apply -focus on layout.oas.
dcv_trimmer -i layout.oas -focus 0,0,1,1

# OK. Apply -svc on pxl.rs.
dcv_trimmer -rs pxl.rs -svc "M1.R.1.S*"

# OK. Apply -svc and -focus on layout.oas. Apply -svc on pxl.rs.
dcv_trimmer -i layout.oas -rs pxl.rs -svc "M1.R.1.S*" -focus 0,0,1,1

# ERROR. Cannot apply any of the specified criteria to pxl.rs.
dcv_trimmer -i layout.oas -rs pxl.rs -focus 0,0,1,1
```

Table 5 describes the command-line options.

Table 5 DCV Design Trimmer Tool Command-Line Options

Option	Description
`-I` *`path`* `[...]`	Specifies a preprocessor include directory for the runset input. For multiple include directories, specify this option repeatedly. For example, `-I INCLUDE/ -I ../other/INCLUDE/`
`-D` *`name[=value]`* `[...]`	Specifies a preprocessor define for the runset input, with an optional value. For multiple defines, specify this option repeatedly. For example, `-D d_AA -D d_BB=3`

Table 5 DCV Design Trimmer Tool Command-Line Options (Continued)

Option	Description
`-focus` *`x1,y1[,x2,y2]`*	Reduces the layout input by selecting one window. This option specifies the window's focal point or area. Units are in nanometers. Unless `-sc` is specified, the window is relative to the top cell. The `-sc` command-line option is required if the layout input has multiple top cells. See Figure 5 for a visualization of this option's role in selecting a window.
`-ambit side |` *`horizontal,vertical`*	Optionally used with `-focus` to reduce the layout input by window. This option defines the region surrounding the focal location in each window. Units are in nanometers. By default, zero ambit is added to each side of the focal location. See Figure 5 for a visualization of this option's role in selecting a window.
`-noclip`	Prevents data overlapping the boundaries of the window defined by `-focus` and `-ambit` from being otherwise clipped. See Figure 5 for a visualization of this option's role in selecting a window.

`-sld "`*`layer[-layerEnd][,dtype[-dtypeEnd]]`*` [...]"`	• Reduces the layout input by selecting layers.
	• Reduces the runset input by content needed for rules that directly or indirectly depend on the selected layers.
	The argument is a space-delimited list of layer-datatypes. It must be enclosed in quotation marks if it contains spaces.
	Example:
	`-sld "0-1 2 3,0-4 4-5,1-2"`
	`-sld 0-1`

`-uld "`*`layer[-layerEnd][,dtype[-dtypeEnd]]`*` [...]"`	Reduces the layout input by unselecting layers. The argument format is the same as `-sld`.
`-sc` *`cell`*	Reduces the layout input by selecting a cell. The specified cell becomes the top cell of the reduced layout. Its subcells are also included.
`-uc` *`cell`* `[...]`	Reduces the layout input by unselecting cells. Their subcells are also unselected. For multiple cells, specify this option repeatedly. For example, `-uc cellA -uc cellB -uc cellC`

Table 5 DCV Design Trimmer Tool Command-Line Options (Continued)

Option	Description

`-svc` *`violationComment`* `[...]`
`-svn` *`violationName`* `[...]`
	• Reduces the layout input by assign layers that are used to check the selected rules. Requires `-rs`.
	• Reduces the runset input by content that depend on the selected rules.
	The argument format and selection behavior follow IC Validator's `-svc` and `-svn` command line options. See the Command-Line Options section of the IC Validator User Guide for details.
	Example:
	`-svc "RULE1*" -svc "RULE2*"`

`-od` *`path`*	Overrides the directory the Design Trimmer outputs results to. By default, it is the current directory.
`-o` *`filename`*	Overrides the reduced layout's file name. The default file name is *`basename`*.trim.*`ext`*, where *`basename`* and *`ext`* are the original layout database's basename and extension respectively.
`-h` `-help`	Displays the help manual.
`-v` `-version`	Displays version.

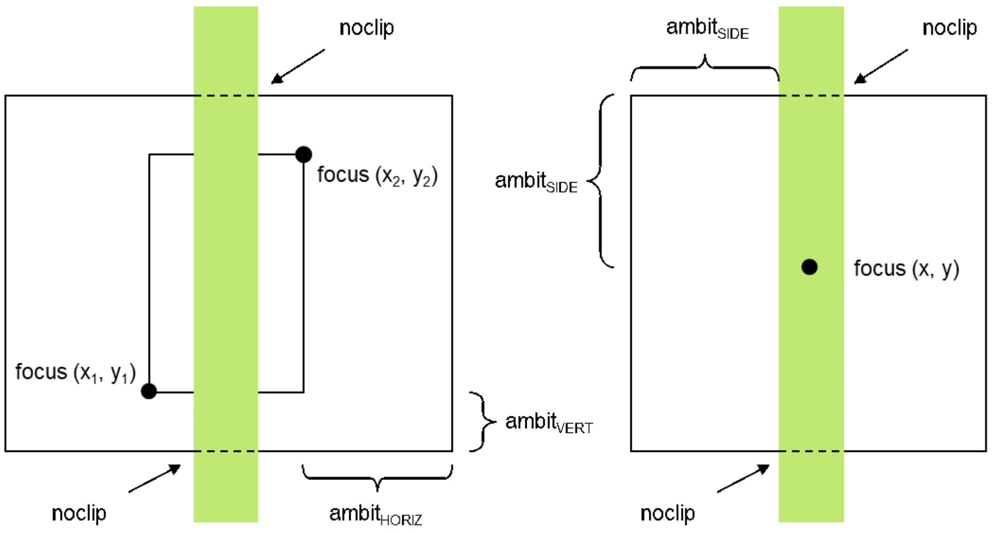


Figure 5 Selecting a Window in a Layout Database

Combining Criteria

You can specify multiple criteria in one run. The reduced test case generated by the tool is the intersection of the data selected by each criterion.

Note:

If `-sc` and `-focus` are combined in one run, the Trimmer assumes the coordinates of the select window are relative to the new top cell.

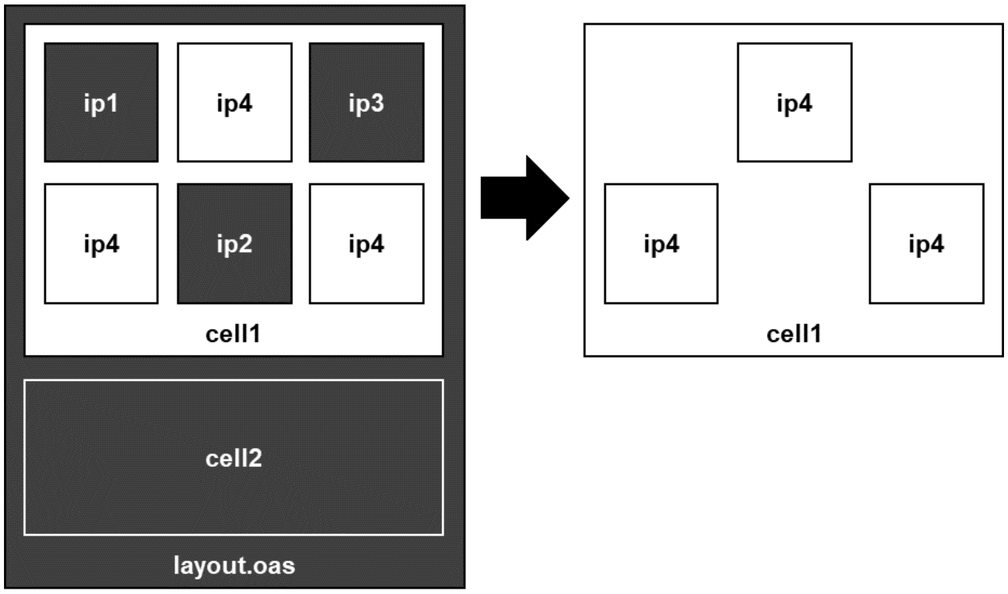
Note:

The `-sld` cannot be combined with `-svc` or `-svn` in one run.

Example 10 combines `-sc` and `-uc` to reduce a layout database. The left image represents the database input, and its dark shaded regions indicate data that are not selected by the combined criteria. The right image represents the reduced database output.

Example 10   Combining Criteria

```
$ dcv_trimmer -i layout.oas \
    -sc cell1 \
    -uc ip1   \
    -uc ip2   \
    -uc ip3
```



Outputs

The Trimmer generates the following results:

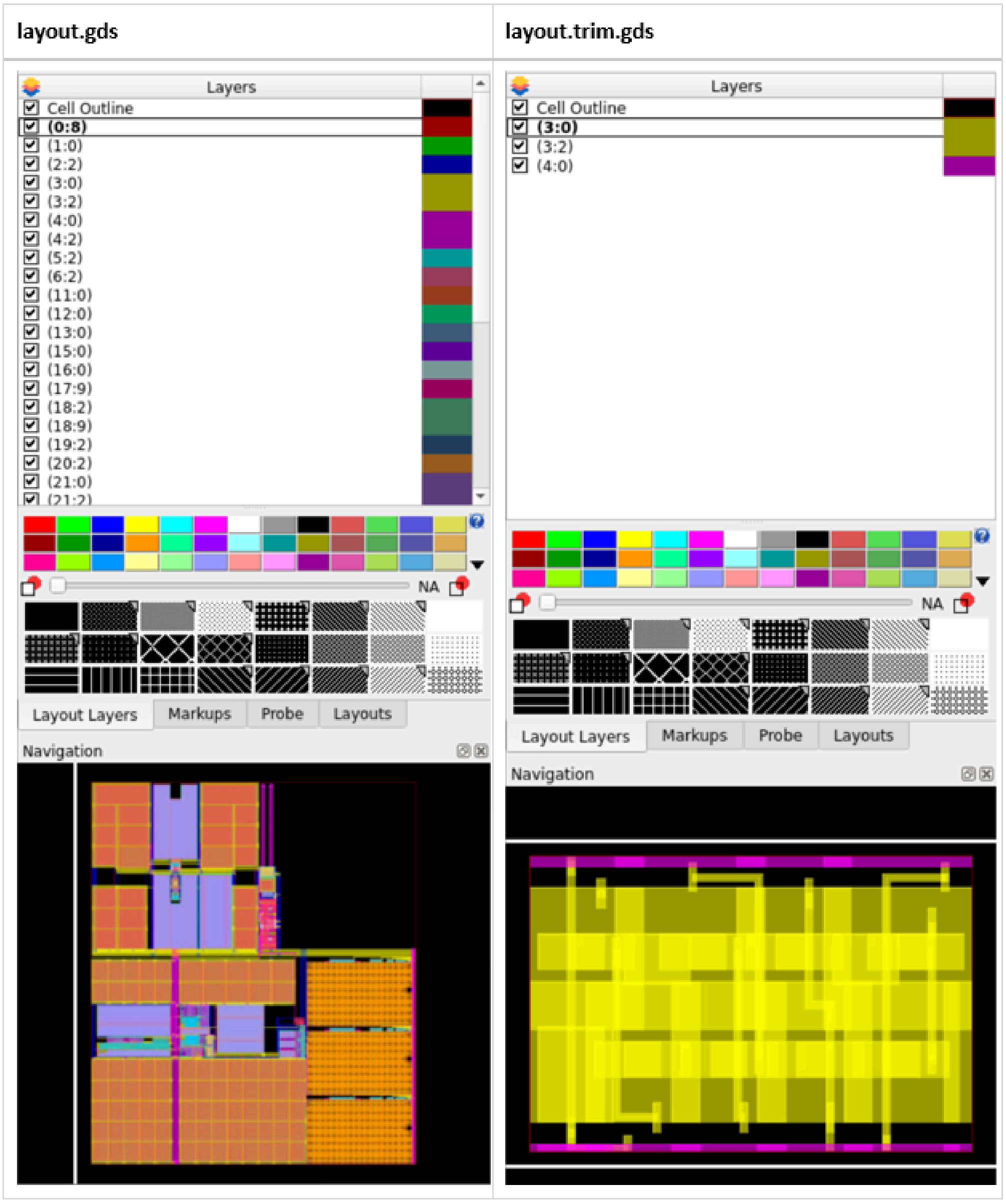
- A reduced layout database if a layout was specified as input. The file name is `basename.trim.ext`, where *basename* and *ext* are the input database file's basename and extension respectively. You can override this file name using the `-o` command-line option.
- A reduced runset if a runset was specified as input. The file name is `basename.trim.ext`, where *basename* and *ext* are the input runset file's basename and extension respectively.
- A summary log file for the run, named trimmer.log.

Figure 6, Figure 7, and Example 11 contain examples of each of the tool's outputs after running the following command line:

```
dcv_trimmer -i layout.gds -rs pxl.rs \
  -D AAA -D BBB -D CCC \
  -svn ruleb           \
  -sc cell1            \
  -focus -568.599,5590.276,6357.728,1637.4920
```

Figure 6 and Figure 7 also show the original database and runset file for comparison.

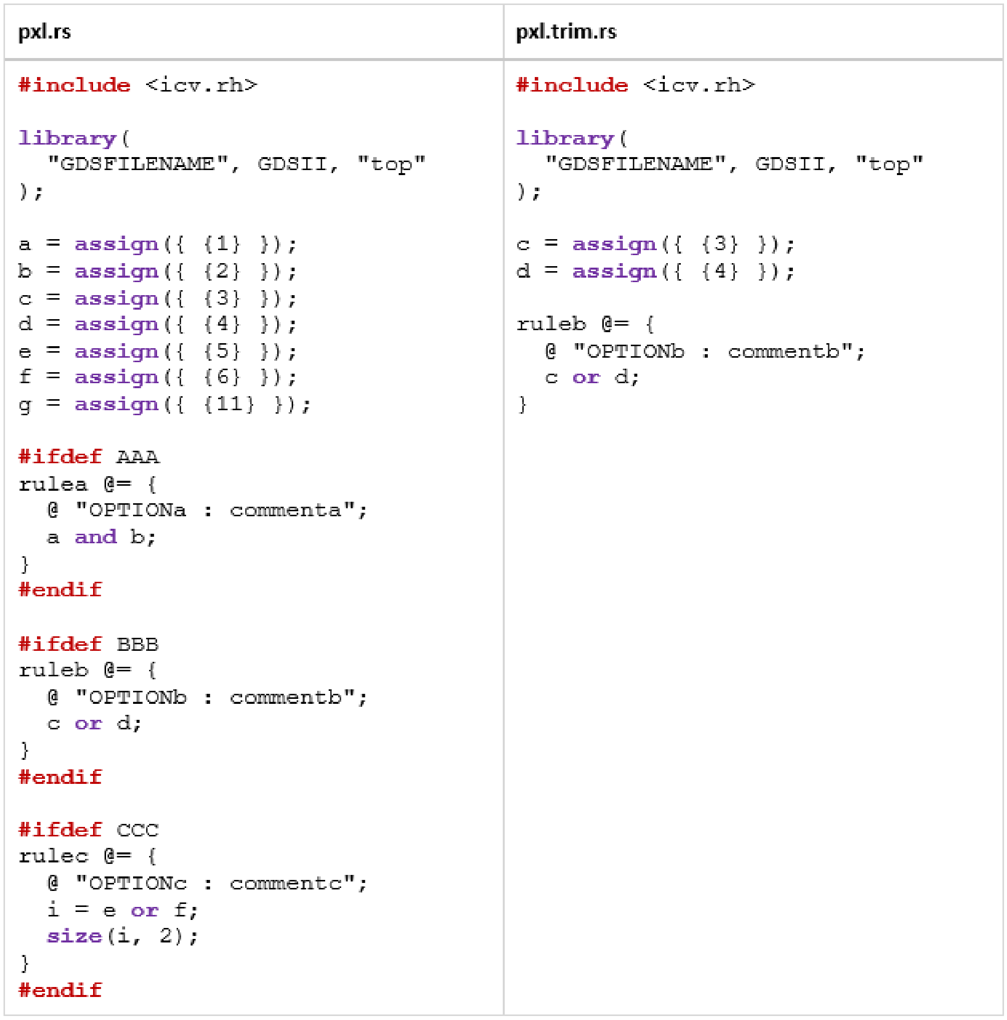
Figure 6 Original Versus Reduced Layout Database



layout.gds	layout.trim.gds
Cell Outline	Cell Outline
(0:8)	(3:0)
(1:0)	(3:2)
(2:2)	(4:0)
(3:0)	
(3:2)	
(4:0)	
(4:2)	
(5:2)	
(6:2)	
(11:0)	
(12:0)	
(13:0)	
(15:0)	
(16:0)	
(17:9)	
(18:2)	
(18:9)	
(19:2)	
(20:2)	
(21:0)	
(21:2)	

Feedback

Figure 7 Original Versus Reduced Runset



pxl.rs

```
#include <icv.rh>

library(
  "GDSFILENAME", GDSII, "top"
);

a = assign({ {1} });
b = assign({ {2} });
c = assign({ {3} });
d = assign({ {4} });
e = assign({ {5} });
f = assign({ {6} });
g = assign({ {11} });

#ifdef AAA
rulea @= {
  @ "OPTIONa : commenta";
  a and b;
}
#endif

#ifdef BBB
ruleb @= {
  @ "OPTIONb : commentb";
  c or d;
}
#endif

#ifdef CCC
rulec @= {
  @ "OPTIONc : commentc";
  i = e or f;
  size(i, 2);
}
#endif
```

pxl.trim.rs

```
#include <icv.rh>

library(
  "GDSFILENAME", GDSII, "top"
);

c = assign({ {3} });
d = assign({ {4} });

ruleb @= {
  @ "OPTIONb : commentb";
  c or d;
}
```

Example 11 Summary Log File

```
Called as: dcv_trimmer -i layout.gds -rs pxl.rs -D AAA -D BBB -D CCC -svn
 ruleb
-sc cell1 -focus -568.599,5590.276,6357.728,1637.4920


*******************
* Runset Trimming *
*******************
engine         := icv
runset input   := "pxl.rs"
defines        := AAA, BBB, CCC
runset output  := "/path/to/pxl.trim.rs"
svn            := block(ruleb)
```

```
Engine time = ~1s


***********************
* Reduced Runset Info *
***********************
AssignLayer: c = assign({{3}});
AssignLayer: d = assign({{4}});
RuleComment: @ "OPTIONb : commentb";


*******************
* Design Trimming *
*******************
engine        := icvwb
layout input  := "layout.gds"
layout output := "/path/to/layout.trim.gds"
sc            := cell1
sld           := 3  4
clip          := focus: { -568.5990000 5590.2760000 6357.7280000
1637.4920000 }, ambit: { -568.5990000 5590.2760000 6357.7280000
 1637.4920000 }

Engine time = ~2s
```

Feedback

4

DCV Results Compare Tool

This chapter explains how to run the DCV Results Compare tool and view the reports.

The IC Validator DCV Results Compare tool compares two sets of error data from different sources and produces a prioritized discrepancy report, and generates an output that you can view in the IC Validator VUE tool.

The DCV Results Compare tool compares can be used to perform the following functions:

- Overview
- Command-Line Options
- DRC Results Comparison
- LVS Results Comparison
- DPT Results Comparison
- Waiver Conversion Flow

Overview

The DCV Results Compare tool can be used to perform the following functions:

- DRC Results Correlation
- LVS Results Correlation
- Double Patterning (DPT) Results Correlation
- Design ECO Results Analysis
- Third-Party Waiver Database Conversion

DRC Results Correlation

The DCV Results Compare tool provides an automated, efficient way of correlating DRC error results from two different databases. These databases represent the golden, or reference error results (Baseline) and the test (Comparator) error results. The following database formats, commonly available as outputs from DRC tools, are supported:

- IC Validator PYDB
- IC Validator ASCII (.LAYOUT_ERRORS)
- 3rd Party ASCII
- GDSII
- OASIS
- Results Compare tool internal format

Note:

The formats of the input databases are automatically detected by the tool.

Note:

When IC Validator PYDB is provided as an input, it is necessary to keep the database file in a different directory than the DCV Results Compare tool working directory. It is recommended to maintain this database file in the original /pydb directory.

There are several different methods of comparison that can be chosen, depending on the exactness of the correlation required. You can choose from the following:

- Exact
- Fuzzy exact
- Overlap
- Overlap with abutment
- Overlap with abutment and point-touch
- Auto

The best compare method to choose varies and depends greatly on the database formats and the types of error results involved. For example, the IC Validator .LAYOUT_ERRORS file represents errors as a bounding-box. Therefore, it is not recommended to choose Exact when comparing against a database that represents errors results as-is, such as the IC Validator PYDB format. Auto mode enables the tool to choose the recommended compare method according to the input database formats.

The tool also supports two methods of processing the error data before correlating the results. Error results can either be merged or unmerged. The merge behavior (default) treats all error markers for a given rule that interact as a single entity for correlation. By contrast, the unmerge option behavior, which is selected by command-line option, treats each error marker individually for correlation.

The unmerge behavior provides a more precise comparison, and is generally recommended, but consideration must be taken when choosing the method of comparison to use.

See DRC Results Comparison for more information.

LVS Results Correlation

The DCV Results Compare tool provides an automated, efficient way of comparing the results from two different IC Validator LVS runs.

When correlating LVS results, it is important to note that an LVS run produces two sets of results: the extraction stage results and the compare stage results. These are correlated by running the DCV Results Compare tool separately for each result.

The extraction stage correlation process is run exactly the same as when correlating DRC results. The primary difference between the two is that there is more detailed evaluation of text and device extraction error content.

The compare stage correlation process focuses on identifying differences in the LVS netlist comparison results. The inputs to the DCV Results Compare tool for this process are the run_details directories for each LVS run, which contains the log file and compare summary files needed for correlation.

See LVS Results Comparison for more information.

Double Patterning (DPT) Results Correlation

The DCV Results Compare tool provides an automated, efficient way to compare double patterning post-decomposition checking results. The results being correlated are odd-cycle, even-cycle, and anchor conflicts, not the resulting colored data layers.

This is a specialized flow that applies ONLY when there are specific DRC rules created to identify these decomposition conflicts (odd-cycle, even-cycle, and anchor).

See DPT Results Comparison for more information.

Design ECO Results Analysis

The DCV Results Compare tool provides an automated, efficient way of analyzing DRC error results from two different PYDB databases. These databases represent the DRC results before (baselines) and after (comparator) a layout design change, commonly referred as ECO.

See Design ECO Results Analysis for more information.

Third-Party Waiver Database Conversion

The DCV Results Compare tool provides an automated, efficient way to convert third-party waiver GDSII databases to an equivalent IC Validator error classification database (cPYDB). The cPYDB is constructed based on errors from a corresponding IC Validator DRC run that match the waiver shapes data from the third-party waiver GDSII database.

The waiver conversion process is run almost exactly the same as you would for correlating DRC results. The only difference is that a mapping file is needed to identify the layer;datatype assignments identifying waiver shapes, comments, user names, and dates. See Waiver Conversion Flow for more information.

Command-Line Options

Before running the DCV Results Compare tool, make sure the `LM_LICENSE_FILE` environment variable and your *`ICV_INSTALL_DIRECTORY`* installation directory are set. The `dcv_rct` executable is located in the same bin directory as the `icv` executable. The tool creates the output database and report files in your current working directory.

The DCV Results Compare tool command-line syntax is

```
dcv_rct
     overlap | exact | fuzzy_exact | auto
     baseline_data
     comparator_data
     layout_data
     [options]
```

where

- *`baseline_data`* specifies the results file that contains the reference error data
- *`comparator_data`* specifies the error file that contains the error data for comparison with the reference error data
- *`layout_data`* specifies either the original layout database used to generate the error data or a skeleton database that contains just the layout hierarchy data

The DCV Results Compare tool command-line options allow you to

- Specify custom rule mappings in a rule-map file
- Select rules that you want to include or exclude

- Waive individual discrepancies by specifying an error classification database
- Specify layer mapping for mixed mode correlation.

Note:

All command-line options must be specified after *`layout_data`*. Exceptions to this restriction are the `-lvs`, `-convert`, and `-skeleton` command-line options, which use different command-line syntax.

Table 6 describes the command-line options.

Table 6 DCV Results Compare Tool Command-Line Options

Option	Description
`-64`	Runs the DCV Results Compare tool with 64-bit coordinates support. See “Support for 64-Bit Coordinates” in the *IC Validator User Guide* for more information. This option is available with Elite or Base licensing. See “Which IC Validator Licenses Do I Need?” in the *IC Validator User Guide* for more information about licensing.
`-all_rules`	Disables rule filtering. By default, the tool excludes rules from the rule map file for which the baseline and comparator data contain no violations and does not include these rules in the output report.
`-convert`	Converts error data from any supported input format to the internal DCV error map format. The syntax is: `dcv_rct -convert error_data layout_data`
`-cpydb_name` *`error-db`*	Specifies the name of the error classification database (cPYDB) that contains error waivers. You must use the `-cpydb_path` option to specify the path to the database.
`-cpydb_path` *`path-name`*	Specifies the path to the error classification database (cPYDB) that contains waivers for comparison discrepancies. You must also use the `-cpydb_name` option to specify the name of the database.
`-csv`	Saves the summary report in a comma separated value (CSV) format file, in addition to the comparison report file. You can import this CSV file into a third-party tool, for further analysis.
`-dbname` *`database`*	Specifies the error database (PYDB) that the tool reads for error input.
`-delta` *`value`*	Specifies the tolerance that the tool uses to expand the baseline and comparator data for the `fuzzy_exact` comparison type. The tool uses this value as a multiplier of the input database resolution. The maximum value is `10`. The default is `5`.

Table 6 DCV Results Compare Tool Command-Line Options (Continued)

Option	**Description**
`-dpt` *`mapfile`*	Specifies a DPT map file, which contains the rules and layers the tool uses to apply the DPT comparison flow for selected rules other than the default DRC comparison flow.

`-eco`	Executes the design ECO results analysis flow.
	The syntax is:
	`dcv_rct -eco -base_error_db <pydb input> -base_library <baseline layout> \`
	`-comp_error_db <pydb input> -comp_library <gds/oasis>`

`-flat_error`	Reports total flat violation counts from the baseline and comparator error databases.
`-g`	Generates a rule-map file containing default rules that you can modify and use as input for a subsequent run. The tool determines these default rules based on the data it is comparing when you do not specify an input rule-map file with the `-m` option.
`-h` `-help`	Displays the usage message and exits.
`-host_init` *`hostname`* `|` *`#cores`* `|` *`hostname:#cores`*	Starts a job on a single host or multiple hosts. For more information, see “Specifying Host Names and CPU Counts” in the *IC Validator User Guide*.
`-host_login`	Specifies the binary/script used to log in to remote hosts. For more information, see “General Command-Line Options” in the *IC Validator User Guide*.
`-host_memory` *`Memory_Value`* `M|G|T`	Sets a target memory for each host in the run. For more information, see “General Command-Line Options” in the *IC Validator User Guide*.

`-i edge | all`
`-include edge | all`
	Specifies the types of error marker interactions that represent a match when you specify the `overlap` comparison type. By default, the tool reports discrepancies for error markers that do not interact in any way. Use `edge` to report discrepancies for error markers that do not interact except for edge abutments or point touches. Use `all` to report discrepancies for error markers that do not interact except for point touches only.

`-keep`	Keeps the rct_comparator.gds and rct_baseline.gds files in the DCV Results Compare tool working directory after the dcv_rct run.
`-layer_map` *`map-file`*	Specifies a user-supplied layer map file to be used for results correlation between raw-shapes (GDS/OASIS) and non-raw-shapes error databases.

Table 6 DCV Results Compare Tool Command-Line Options (Continued)

Option	Description
`-lvs`	Executes the LVS compare stage results comparison flow. The syntax is: `dcv_rct -lvs baseline_input_directory comparator_input_directory` The input directory is the run_details directory from the original IC Validator run without modification.

`-m` *`map-file`*
`-map` *`map-file`*
	Specifies a rule-map file containing the rules that the tool uses to compare results. If you do not specify this file, the tool determines default rules based on the data it is comparing. To generates a default rule-map file that you can modify and use for subsequent runs, specify the `-g` option.

`-merge`	Directs the DCV Results Compare tool to correlate error data (edges/polygons) by merging interacting data for the same rule together. This is the default behavior for the tool.
`-missing_rules`	Explicitly identifies rules that are present in one input, but are completely absent from the other. By default, the DCV Results Compare tool identifies these missing rules in the regular summary with a RCT_gen* prefix. This command-line option creates two additional sections in the report identifying all missing rules by their original name. `-missing_rules` example for more information. **Note:** The `-missing_rules` command-line option is ignored for GDSII or OASIS inputs.
`-oasis`	Temporarily converts the GDSII layout database to OASIS format for internal processing by the tool. **Note:** When used with the `-keep` command-line option, the intermediate rct_baseline and rct_comparator databases will be in OASIS format.
`-rci`	Directs the DCV Results Compare tool to correlate rule names by making all characters uppercase while creating the rule map. This option does not apply to comments specified in the `-svc` and `-uvc` command-line options.

Table 6 DCV Results Compare Tool Command-Line Options (Continued)

Option	Description
`-skeleton`	Creates a layout database that contains only the layout hierarchy data from the original layout database, which is required input when you use this option. For subsequent runs, you can specify this skeleton layout database as input instead of the original layout database. The syntax is: `dcv_rct -skeleton layout_data`
`-sort` *`criteria`*	Sorts the report summary based on a specific criteria. The following values are supported: • rule. Sorts alphabetically based on the rule name. • layer. Sorts based on the layer number and datatype (only applicable for raw error-shape comparison). • severity (default). Sorts based on the severity of results. For example: `dcv_rct` *`req args`* `-sort severity` **Note:** This command-line option is supported only in the DRC comparison flow. Neither LVS nor waivers support this command-line option.
`-srg` *`rule-group-list`*	Specifies one or more rule groups, from the rule-map file, that the tool uses to compare the results. Separate individual rule groups with commas.

`-svc` *`rule-name`*
`-uvc` *`rule-name`*
	(`-svc`) or suppresses (`-uvc`) violations in either the baseline or comparator inputs by rule *names* of `"RuleName : RuleDescription"` – not by rule comments. The arguments
	• Can include the wildcard * and range expressions using [ ].
	• Must be enclosed in quotation marks if they use a wildcard *. (This enclosure prevents processing by the UNIX shell.)
	These options take only one argument but you can specify multiple options on a command line. For example,
	`%` **`dcv_rct req args -svc "*96A1*" -svc "*11B16*"`**
	If the final selection of violations is empty, the run terminates with an error. Usage of the `-svc` and `-uvc` command-line options is allowed only when the baseline/comparator formats are text- or PYDB-based.

`-unmerge`	Directs the DCV Results Compare tool to correlate error data (edges/polygons) individually without merging interacting data together. By default, error data for the same rule is merged.

Table 6 DCV Results Compare Tool Command-Line Options (Continued)

Option	Description
`-urg` *`rule-group-list`*	Specifies one or more rule groups, from the rule-map file, that the tool excludes from the results comparison. Separate individual rule groups with commas.

DRC Results Comparison

See the following sections on comparing the results from different IC Validator DRC runs:

- Running the Tool
- Comparison Results File

Running the Tool

The DCV Results Compare tool compares errors from two data files and reports discrepancies based on the comparison type that you specify.

To compare error data, you specify the type of comparison you want to perform, the files that contain the errors you want to compare, and the layout data. For example, to compare the results from a baseline file named TOP.LAYOUT_ERRORS and a comparator file named TOP.db with the original layout data in a file named TOP_input.gds, and report discrepancies for errors that do not match exactly, enter

```
% dcv_rct exact TOP.LAYOUT_ERRORS TOP.db TOP_input.gds -unmerge
```

For more information on the available comparison types, see Setting the Comparison Type. For information about the file formats for the input files, see Input File Formats.

You can specify either the original layout database used to generate the results or a skeleton database that contains just the layout hierarchy data. The tool uses this data to resolve hierarchical differences in the output results if either the baseline or comparator results are in a text file format.

The DCV Results Compare tool performs a rule-based error-versus-error comparison of the error data and creates a comparison report listing the discrepancies sorted by severity. The tool generates both a report file, *top_cell*.RCT.report, and a VUE-compatible file, *top_cell*.RCT.vue, where *top_cell* is the name of the top cell in the design.

You can also save the report in a comma-separated value (CSV) format file by using the `-csv` command-line option. You can import this file into a third-party tool (such as the Microsoft Excel tool) for further analysis.

For information about the report file, see Comparison Results File. For information about the IC Validator VUE tool, see the *IC Validator VUE User Guide*.

For more information, see the following sections:

- Specifying Custom Rule Mappings
- Including or Excluding Rules
- Waiving Errors With an Error Classification Database
- Mixed-Mode Comparison

Choosing the Process Method

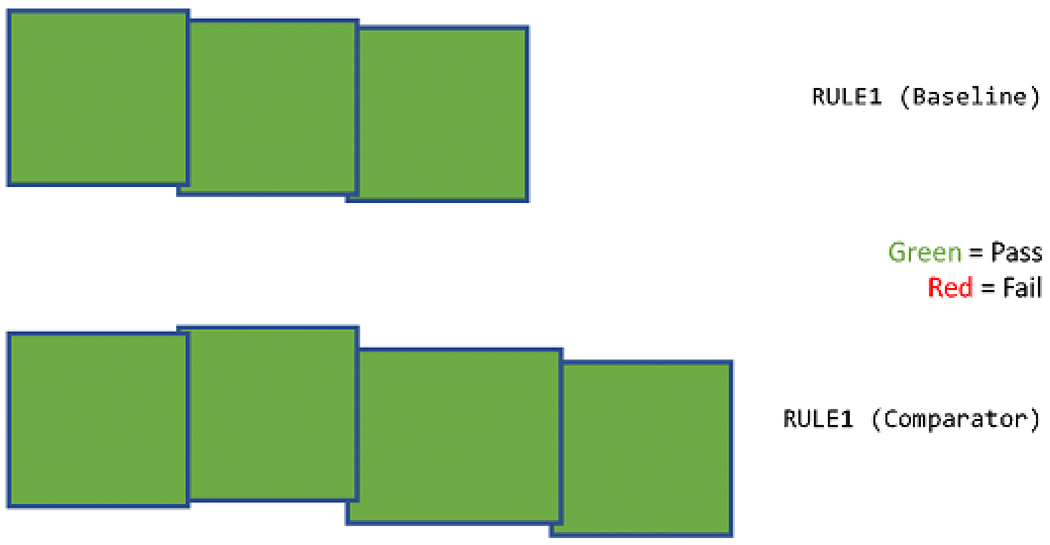
There are two methods of processing data within the DCV Results Compare tool: merge and unmerge.

The merge method is the default method of correlating data in the DCV Results Compare tool. With this method, input error data is merged before correlation for the same rule or the same layer;datatype in the case of GDS or OASIS error databases that interact.

By contrast, the unmerge method correlates error data without merging. Each data element in the input error databases is correlated individually.

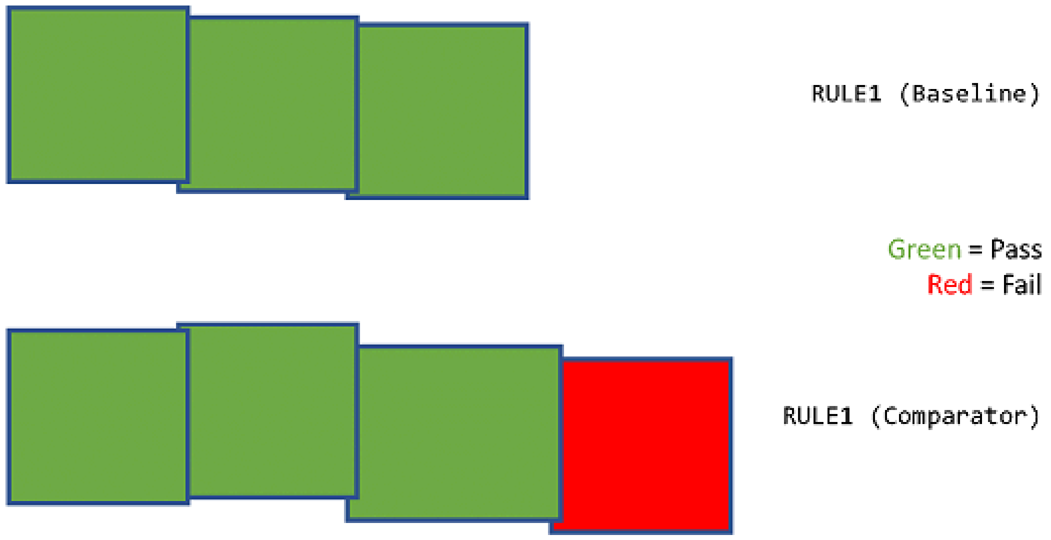
The merging of error data can result in unexpected correlation results in cases where multiple error markers from violations to the same rule are expected to interact. See Figure 8, Figure 9, Figure 10, and Figure 11 for the differences in correlation results between the two processing methods, using overlap and exact compare modes respectively for an illustration.

Figure 8 Merge Processing With Compare Mode Set to Overlap



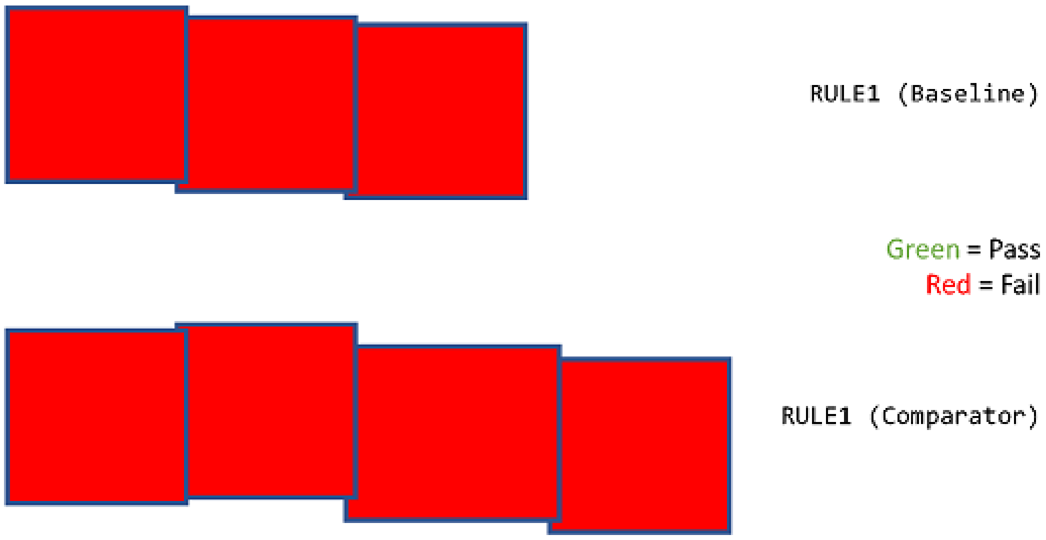
```
Rule     Baseline   Comparator   Result     Total      Total     Missed      False
Group   Rule Name    Rule Name           Baseline Comparator     Errors     Errors
    1       RULE1        RULE1     PASS         3          4          0          0
```

Figure 9 Unmerge Processing With Compare Mode Set to Overlap



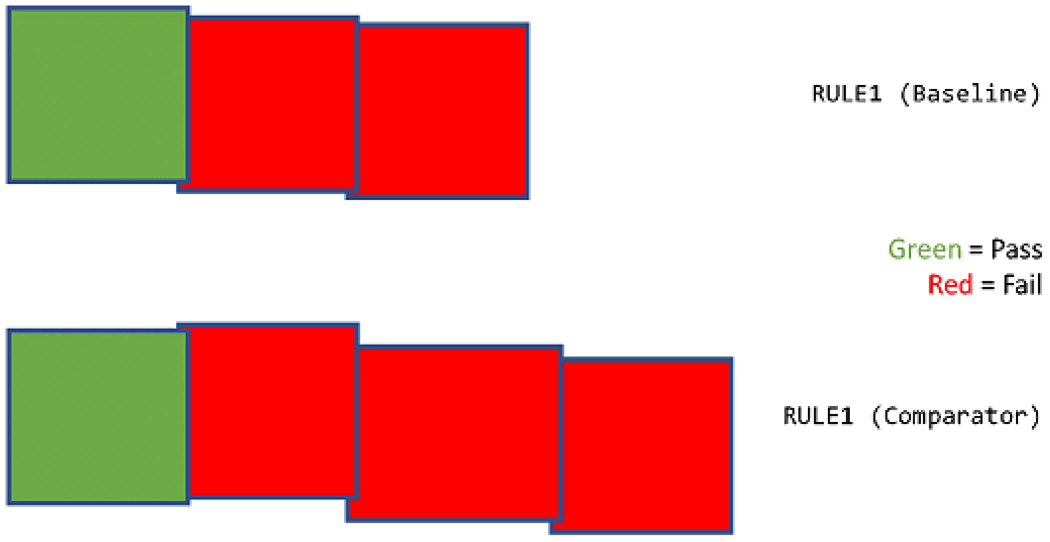
```
Rule     Baseline   Comparator   Result      Total      Total    Missed      False
Group   Rule Name    Rule Name            Baseline Comparator    Errors     Errors
    1       RULE1        RULE1     FAIL          3          4         0          1
```

Figure 10   Merge Processing With Compare Mode Set to Exact



```
Rule     Baseline   Comparator   Result      Total       Total     Missed      False
Group   Rule Name    Rule Name            Baseline  Comparator     Errors     Errors
    1       RULE1        RULE1     FAIL          3           4          1          2
```

Figure 11 Unmerge Processing With Compare Mode Set to Exact



```
 Rule     Baseline   Comparator   Result     Total      Total     Missed    False
Group    Rule Name    Rule Name            Baseline Comparator    Errors    Errors
    1        RULE1        RULE1     FAIL          3          4         2         3
```

Setting the Comparison Type

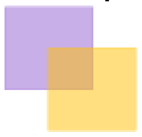
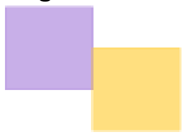
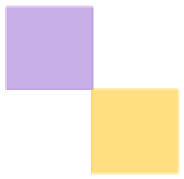
The comparison type sets the criteria for matches between baseline and comparator error markers. Valid comparison types are `overlap`, `exact`, `fuzzy_exact`, or auto. The tool performs an automated results-versus-results comparison of the error data and reports PASS for errors that match and FAIL for errors that fail the comparison.

Use `overlap` to match error markers that overlap. You can include edge abutments and corner touches by using the `-i` command-line option.

- To report a match for error markers that have an edge abutment or a corner touch but do not overlap, use the `-i edge` command-line option.
- To report a match for error markers that have a corner touch but do not overlap or have an edge abutment, use the `-i all` command-line option.

Table 7 shows the truth tables for the `overlap` comparison type.

Table 7 Interactive Matches

Comparison type	**Overlap** 	**Edge abutment** 	**Point touch** 
`overlap`	PASS	FAIL	FAIL
`overlap -i edge`	PASS	PASS	FAIL
`overlap -i all`	PASS	PASS	PASS

Use `exact` to match error markers that match exactly.

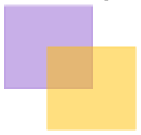
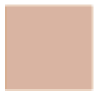
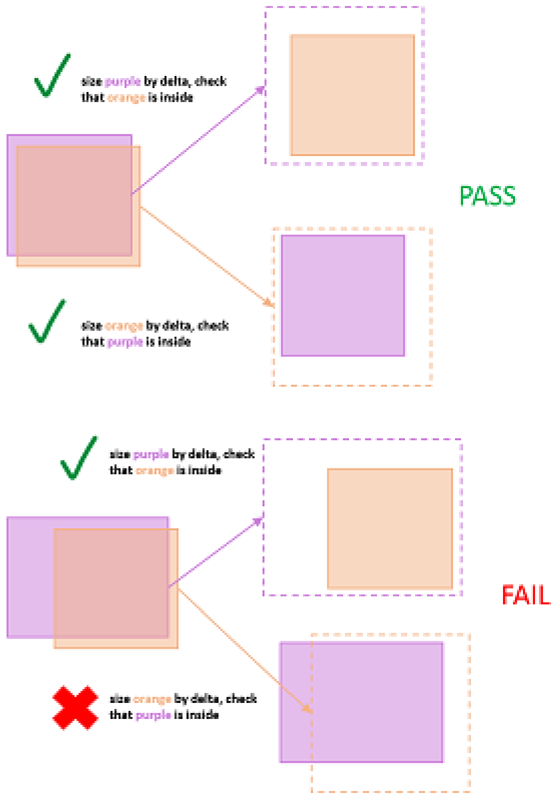
Use `fuzzy_exact` to match error markers that match within an expanded region. You can define a tolerance value for this region by using the `-delta` command-line option to specify a multiplier of the input database resolution. The maximum value is `10`. The default is `5`.

Use `auto` to enable the tool to choose the recommended compare method according to the input database formats.

For example, if the input database resolution is 0.001 um, the default tolerance value for the expanded region is 0.005 um. The tool expands the error markers from the baseline and comparator data, and then evaluates them to determine if they match within these expanded regions.

Table 8 shows the truth tables for the `exact` and `fuzzy_exact` comparison types.

Table 8 Exact Matches

Comparison type	Overlap	Exact	Fuzzy exact
			 size purple by delta, check that orange is inside size orange by delta, check that purple is inside PASS size purple by delta, check that orange is inside size orange by delta, check that purple is inside FAIL
`exact`	FAIL	PASS	FAIL
`fuzzy_exact`	FAIL	PASS	PASS

Input File Formats

The DCV Results Compare tool automatically detects the input data formats. The baseline and comparator data do not have to be in the same format. The valid formats for the baseline and comparator data files are

- IC Validator LAYOUT_ERRORS file
- IC Validator error database (PYDB)
- DCV error map database
- GDSII

- OASIS
- Third-party ASCII error file

Note:

When the baseline or comparator data is in the IC Validator error database format, use the `-dbname` command-line option to specify the PYDB file name.

The input data does not need to be flattened because the tool preserves the hierarchy of the input data during a comparison.

The valid formats for the original layout data file are

- GDSII
- OASIS

Gzipped input error data, except those originally in PYDB format, and layout data are also valid.

Note:

For text-based comparison, at least one of the error formats (baseline or comparator) must be of the following type:

- IC Validator ASCII error format
- IC Validator PYDB error format
- DCV error map format

Mixed-Mode Comparison

By default, the DCV Results Compare tool can compare a raw shapes (GDSII/OASIS) error database directly to another raw shapes error database by mapping between identical layer;datatype assignments. You can compare a raw shapes error database to a different format by using mixed mode comparison.

Mixed-mode comparison is triggered simply by specifying the input databases, as usual, and supplying a layer-map file using the`-layer_map` command line option.

The following example compares results between a GDSII error database (rct_baseline.gds) and an IC Validator PYDB error database (comparator.pydb).

```
dcv_rct overlap rct_baseline.gds comparator.pydb layout.gds -layer_map
baseline.layer.map
```

Layer-Map File

The layer-map file is required in mixed-mode comparison to define the mapping from the layer;datatype assignments of a GDS/OASIS error database to the Rule Descriptions available in other database formats See Input File Formats for more information.

Example 12 shows the format of a layer-map file with three layer and datatype pairs. Table 9 describes the fields in the map file.

Example 12 Layer-Map File Format

```
L;D  ,  RuleName : RuleComment
0;0  ,  "RuleA"
1;0  ,  "RuleB : Comment B"
2;0  ,  "RuleC : Comment C"
```

Note:

In a layer map file, the layer and datatype pair assignments do not need to be in order.

Table 9 Fields in a Layer-Map File

Field	Description
L;D	Identifies a layer;datatype pair from the GDSII or OASIS error file.
Rule Description (RuleName : RuleComment)	Identifies the rule. This descriptive comment typically corresponds to the Process Design Manual rule name or rule descriptor, for example, GRMx.S.2 : Mx minimum space.

The syntax requirements for the layer-map file are:

- Fields in the layer-map file cannot be empty
- An empty string ("") is not allowed for a rule description
- Assignments must be 1:1, that is one layer;datatype assignment is allowed for one rule description and vice versa

Layer;datatypes that exist in the input GDSII/OASIS database that are not specified in the layer-map file have generic Rule Descriptions generated. The descriptions are for the form: `RCT_L<`*`layer`*`>D<`*`datatype`*`>`.

Example 13 shows the error databases and layer-map file used in mixed-mode comparison and a summary of the outcome results.

Example 13 Mixed-Mode Comparison With Baseline GDSII Database

```
Layer;Datatypes in Baseline GDS: {0;0} {1;0}, {2;0}
Rules in Comparator Database: RuleA, RuleB, RuleC
```

```
Layer-Map File:
       L;D  ,  RuleName : RuleComment
       0;0  ,  "RuleA"
       1;0  ,  "RuleB : Comment B"

Report SUMMARY

            ------------------
            COMPARISON SUMMARY ( Total PASS = 2  Total FAIL = 2 )
          ------------------

Rule     Baseline    Comparator  Result Total     Total        Missed    False
Group    Rule Name   Rule Name          Baseline  Comparator   Errors    Errors
   1     RuleA       RuleA         PASS    1         1           0         0
   2     RuleB       RuleB         PASS    1         1           0         0
   3     RCT_L2D0  RCT_genC_RCT_L2D0 FAIL 1         0           1         0
   4  RCT_genB_RuleC RuleC         FAIL    0         1           0         1
```

Note:

While the layer-map file requires 1:1 mapping between layer;datatype assignments and rule descriptions, it is possible to achieve M:N rule mappings in mixed-mode comparison by combining the layer-map file with an input rule-map file. See Rule-Map File for more information. Example 14 shows how this can be done.

Example 14   Baseline Layer-Map File With Input Rule-Map File

```
Layer;Datatypes in Baseline GDS: {0;0} {1;0}, {2;0}
Rules in Comparator Database: RuleA, RuleB, RuleC1, RuleC2

Layer-Map File:
L;D  ,  RuleName : RuleComment
0;0  ,  "RuleA"
1;0  ,  "RuleB : Comment B"
2;0  ,  "RuleC : Comment C"

Input Rule-Map File:

Rule   Baseline    Baseline(L;D) Comparator Comparator(L;D) Description
Group   Rule                      Rule
 1  ,   RuleA  ,     0;0  ,       RuleA  ,    0;0  ,  "RuleA"
 2  ,   RuleB  ,     1;0  ,       RuleB  ,    1;0  ,  "RuleB    Comment B"
 3  ,   RuleC  ,     2;0  ,       RuleC1 ,    2;0  ,  "RuleC    Comment 1"
 3  ,          ,          ,       RuleC2 ,    3;0  ,  "RuleC    Comment 2"
```

Specifying Custom Rule Mappings

By default, the DCV Results Compare tool creates rule mappings based on a 1:1 correspondence of matching rule names extracted from the baseline and comparator error input. These default 1:1 rule mappings are sometimes inadequate. For example, you might need to map multiple rules together in an *N*:1, 1:*M*, or *N*:*M* mapping. You might also encounter cases where rule names do not match and the tool creates placeholder rule names.

You can provide customized rule mappings in a rule-map file, which you specify by using the `-m` command-line option. For example, to specify a rule-map file named TOP.RCT.rule_map, enter

```
% dcv_rct exact TOP.LAYOUT_ERRORS TOP.db TOP_input.gds -m TOP.RCT.rule_map
```

The tool generates a default rule-map file when you use the `-g` command-line option. You can modify this file and use it as input in a subsequent run. Use this option if you know in advance that the default 1:1 rule mappings are inadequate.

```
% dcv_rct exact TOP.LAYOUT_ERRORS TOP.db TOP_input.gds -g
```

For more information about rule-map files, see Rule-Map File.

Including or Excluding Rules

You can limit the comparison to a subset of the available rules by adding rules to an empty selection or by subtracting rules from a universal selection. You specify the rules by using their group identifiers in the rule-map file. The tool selects the entire rule group associated with an identifier to ensure a correct comparison.

To include rules, use the `-srg` command-line option to specify the rules or rule groups. For example, to use only rule groups 1, 2, and 4, enter

```
% dcv_rct exact TOP.LAYOUT_ERRORS TOP.db TOP_input.gds -m TOP.RCT.rule_map
 -srg
1,2,4
```

To exclude rules, use the `-urg` command-line option to specify the rules or rule groups. For example, to exclude rule groups 2 and 5, enter

```
% dcv_rct -m TOP.RCT.rule_map -urg 2,5 exact TOP.LAYOUT_ERRORS TOP.db
TOP_input.gds
```

The Results Compare tool automatically filters rules for which the data does not contain violations by default. To disable filtering, use the `-all_rules` command-line option.

```
% dcv_rct exact TOP.LAYOUT_ERRORS TOP.db TOP_input.gds -all_rules
```

Waiving Errors With an Error Classification Database

The DCV Results Compare tool allows you to specify waivers in an error classification database that prevent the tool from reporting known results comparison discrepancies. The tool marks these waived discrepancies in the reports.

To specify the error classification database (cPYDB), use the `-cpydb_name` and `-cpydb_path` command-line options.

Error classification for the DCV Results Compare tool is similar to the IC Validator error classification flow. The only difference is that you specify the error classification database on the DCV Results Compare tool command line by using the `-cpydb_name` and `-cpydb_path` options.

In general, follow these steps to set waivers for comparison discrepancies:

1. Run the DCV Results Compare tool on the input error data and layout data.
2. Classify the comparison discrepancies that you want to waive and save them in an error classification (cPYDB) database by using the IC Validator VUE tool or the pydb_report utility.

   You can merge new classifications into an existing error classification database. The tool appends the newly classified errors to the database and updates existing errors that have changed.

   For more information about classifying errors, see the *IC Validator VUE User Guide*. For information about error classification databases, see the *IC Validator User Guide*.
3. Rerun the DCV Results Compare tool with the error classification database to preclassify the discrepancies.

   Use the `-cpydb_name` and `-cpydb_path` command-line options to specify the name and location of the error classification database.

Rule-Map File

The rule-map file specifies how the DCV Results Compare tool attempts to correlate results between the baseline and comparator error databases. This is an optional input.

If no map file is provided, the tool creates one using a default 1:1 mapping based on matching rule identifier names (or matching layer;datatype assignments for GDSII/OASIS inputs). This file is maintained after the run and can be modified for subsequent use. If rule names do not match, or a rule in one database is not in the other, the tool automatically generates a placeholder rule name of the form RCT_genB_*rule-name* or RCT_genC_*rule-name*, where B indicates baseline data, C indicates comparator data, and rule-name is the generated rule. You can create your own rule-map file by editing the default file provided from a previous run of the DCV Results Compare tool, or you can generate one using the following rules:

- Every line in the file contains six comma-separated fields
- Every line has at least one baseline rule identifier or comparator rule identifier
- Every line specifies a rule group ID. Rules with the same rule group ID will be merged for correlation.
- Rules assigned to the same rule group ID are grouped together

- Every unique rule group identifier has at least one baseline rule identifier and at least one comparator rule identifier
- A baseline or comparator rule identifier is not used in more than one rule group

Note:

The rule descriptions in the last field of each line are optional, but are highly recommended. The tool uses them when generating the output report, and having meaningful descriptions helps debug the correlation results.

It is not required to specify the layer;datatype assignments. The tool automatically assigns these to unique settings.

A rule-map file can be a superset or subset of the content actually present in the baseline and comparator databases. The behavior of each is described in later examples.

The rule-map file consists of six comma-separated fields, as shown in Example 15.

Example 15   Rule-Map File Format

```
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D),
 Description
     1  ,        F   ,        0;0  ,       F          ,    0;0 ,    "Rule F"
     2  ,        F.1 ,        1;0  ,       F_1        ,    1;0  ,   "Rule F.1"
     3  ,        B   ,        2;0  ,       RCT_genC_B ,    2;0 ,    "Rule B"
```

Note:

In a real rule-map file, the *layer*;*datatype* assignments might not be this well ordered.

Table 10   Rule-Map File Fields

Field	Description
RuleGroup	Determines which baseline results will be compared to which comparator results. Results with the same group ID are merged for correlation.
BaselineRule	Identifies the process ground rule from the baseline results, for example, GRMx.S.2.
Baseline(L;D)	Identifies the baseline *layer*;*datatype* assigned for the generation (default case) of the baseline error shapes. For GDSII or OASIS error files, this data corresponds to the *layer*;*datatype* assigned to the error output.
ComparatorRule	Identifies the process ground rule from the comparator results, for example, GR.Mx.S.2.

Table 10 Rule-Map File Fields (Continued)

Field	Description
Comparator(L;D)	Identifies the *layer*;*datatype* assigned for the generation (default case) of the comparator error shapes. For GDSII or OASIS error files, this data corresponds to the *layer*;*datatype* assigned to the error output.
Description	Identifies the rule. This descriptive comment typically corresponds to the Process Design Manual rule name or rule descriptor, for example, GRMx.S.2 : Mx minimum space.

By default, the DCV Results Compare tool creates rule groups with 1:1 mappings based on matching layers and datatypes in the baseline and comparator data. When rule names do not match, or a rule in one file is not in the other file, the tool automatically generates a placeholder rule name of the form RCT_genB_*rule-name* or RCT_genC_*rule-name*, where B indicates baseline data, C indicates comparator data, and *rule-name* is the generated rule name.

Note:

A rule identifier that appears in more than one rule group causes an error.

You can create *M*:*N* rule mappings by modifying the 1:1 mappings that the tool generates. Make sure you use the same rule group for all of the rules that are part of a particular *M*:*N* mapping.

In Example 16, rule groups 1 and 2 are 1:1 maps, group 3 is a 1:3 map, and group 4 is a 2:4 map:

Example 16 Rule Mappings Organized by Rule Groups

```
Rule  Baseline Baseline Comparator Comparator Description
Group  Rule     (L;D)      Rule       (L;D)
   1 ,  M1.S.2,  100;0,   M1_S_2,    15;0,      "M1 minimum space"
   2 ,  Mx.S.1,  101;0,   M1_S_1,    101;0,     "M1 minimum width"
   3 ,  Bx.R.3,  102;0,   B1_R_3,      1;0,     "B1 touching C"
   3 ,        ,       ,   B2_R_3,      1;1,     "B2 touching C"
   3 ,        ,       ,   B3_R_3,      1;2,     "B3 touching C"
   4 ,  GRD.R.1, 103;0,  GRD_R_1a,   200;0,     "Grid check 1"
   4 ,        ,       ,  GRD_R_1b,   201;0,     "Grid check 1"
   4 ,  GRD.R.2,  103;1,  GRD_R_2a,  202;0,     "Grid check 2"
   4 ,        ,        ,  GRD_R_2b,  202;0,     "Grid check 2"
```

Rule correspondence is controlled through the rule group identifier. In Example 17, rule groups 3 and 4 contain a rule name mismatch and rule groups 5 through 7 contain a 1:*N* rule mapping, which in this example is a single baseline rule matched by two rules in the comparator data.

Example 17 Rule Correspondence

```
Rule  Baseline   Baseline  Comparator  Comparator  Description
Group  Rule      (L;D)      Rule        (L;D)
  1 , F    ,      0;0   ,   F ,          0;0   ,     "Rule F"
  2 , F.1  ,      1;0   ,   F_1 ,        1;0   ,     "Rule F.1"
  3 , B    ,      2;0   ,   RCT_genC_B , 2;0   ,      "Rule B"
  4 , RCT_genB_BB , 3;0 ,   BB ,         3;0   ,      "Rule BB"
  5 , Fx ,        4;0  ,    RCT_genC_Fx , 4;0 ,       "Rule Fx"
  6 , RCT_genB_x1 , 5;0 ,   x1 ,         5;0   ,      "Rule x1"
  7 , RCT_genB_x2 , 6;0 ,   x2 ,         6;0   ,      "Rule x2"
```

Example 18 shows a customized version of the same rule-map file.

Example 18 Updated or Edited Rule-Map File

```
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
     1  ,       F      ,      0;0   ,          F      ,          0;0   ,  "Rule F"
     2  ,       F.1    ,      1;0   ,          F_1    ,          1;0   ,  "Rule F.1"
     3  ,       B      ,      2;0   ,         "BB"    ,          3;0   ,  "Rule B"
     4  ,       "Fx"   ,      4;0   ,         "Fx1" ,            5;0   ,  "Rule Fx1"
     4  ,              ,            ,         "Fx2" ,            6;0   ,   "Rule Fx2"
```

When you modify a rule-map file, ensure that

- Every line in the file contains six comma-separated fields, specifies a rule group ID, and has at least one baseline rule identifier or comparator rule identifier.
- Every rule identifier specifies a corresponding *layer*;*datatype* assignment.

  For example, if you specify a baseline rule identifier, you must also specify a *baseline-layer*;*datatype* assignment.
- Every unique rule group identifier has at least one baseline rule identifier and at least one comparator rule identifier.

Note:

Although the rule descriptions in the last field of each line are optional, you should include rule descriptions because the tool needs them to generate meaningful report information.

Example 19 shows the behavior of the `-missing_rules` command-line option.

Example 19 --missing_rules Command-Line Option

```
  ------------------
  COMPARISON SUMMARY
  ------------------

Rule     Baseline     Comparator  Result Total     Total        Missed     False
Group    Rule Name    Rule Name          Baseline Comparator   Errors     Errors
    1       RULE_1    RCT_gen_RULE_1 FAIL     12           0         10          0
    2       RULE_2    RULE_2         PASS     1            1          0          0
  ----------------------
  MISSING BASELINE RULES
  ----------------------

          NONE
  ------------------------
  MISSING COMPARATOR RULES
  ------------------------

          RULE_1
```

Note:

By default, rules with zero violations do not appear in the comparison summary, but are still reported in the missing baseline or comparator rules sections if the “zero violation rule” does not appear in one of the inputs.

Superset/Subset Rule-Map Files

You can supply a rule-map that is either a subset or a superset of the rules in the input error databases.

Subset Rule Map

When the rule map is a subset of the rules that exist in the input error databases, the DCV Results Compare tool uses the user-specified mappings and adds those mappings for the unmapped rules using the default 1:1 mapping behavior based on rule name. Example 20 illustrates a subset rule map and the resulting tool behavior.

Example 20 Subset Rule-Map File

```
Baseline Database Contents: F, F.1, B, Fx
Comparator Database Contents: F, F_1, BB, x1, x2

User-supplied Subset Rule-Map File:
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10   ,         F    ,        5;0   ,        F    ,           5;0 ,      "Rule F"
11   ,         F.1  ,        6;0   ,        F_1  ,           6;0 ,      "Rule F.1"
12   ,         B    ,        7;0   ,        B    ,           7;0 ,      "Rule B"
13   ,         B    ,        8;0   ,        BB   ,           8;0 ,      "Rule BB"
```

The system-generated rule-map file is shown in Example 21.

Example 21 System-Generated Rule-Map File

```
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10   ,      F    ,          5;0   ,      F     ,          5;0 ,     "Rule F"
11   ,      F.1  ,          6;0   ,      F_1   ,          6;0 ,     "Rule F.1"
12   ,      B    ,          7;0   ,      B     ,          7;0 ,     "Rule B"
13   ,      B    ,          8;0   ,      BB    ,          8;0 ,     "Rule BB"
1   ,       Fx ,            0;0   ,      RCT_genC_Fx ,    0;0 ,     "Rule Fx"
2   ,       RCT_genB_x1 ,   1;0   ,      x1     ,         1;0 ,     "Rule x1"
3   ,       RCT_genB_x2 ,   2;0   ,      x2     ,         2;0 ,     "Rule x2"
```

Superset Rule Map

When the rule map is a superset of the rules, the DCV Results Compare tool limits correlation to the rules that exist in the databases and ignores the remaining mappings. The ignored mappings are logged in a separate file generated by the tool. Example 22 illustrates a superset rule map and the resulting tool behavior.

Example 22   Superset Rule-Map File

```
Baseline Database Contents: F, F.1, B, Fx
Comparator Database Contents: F, F_1, BB, x1, x2

User-supplied Superset Rule-Map File:
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10   ,        F    ,          5;0    ,         F    ,       5;0   ,       "Rule F"
11   ,        F.1  ,          6;0    ,         F_1  ,       6;0   ,       "Rule F.1"
12   ,        B    ,          7;0    ,         B    ,       7;0   ,       "Rule B"
13   ,        B.B  ,          8;0    ,         BB   ,       8;0   ,       "Rule BB"
14   ,        Fx   ,          9;0    ,         Fx   ,       9;0   ,       "Rule Fx"
15   ,        x1   ,          10;0   ,         x1   ,       10;0  ,       "Rule x1"
16   ,        x2   ,          11;0   ,         x2   ,       11;0  ,       "Rule x2"
17   ,        Ub1  ,          12;0   ,         Uc1  ,       12;0  ,       "Extra Rule 1"
18   ,        Ub2  ,          13;0   ,         Uc2  ,       13;0  ,       "Extra Rule 2"
```

With these inputs, Example 23 shows the rule-map file generated by the DCV Results Compare tool for internal correlation.

Example 23   System-Generated Rule-Map File

```
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10   ,        F    ,          5;0    ,         F    ,       5;0   ,       "Rule F"
11   ,        F.1  ,          6;0    ,         F_1  ,       6;0   ,       "Rule F.1"
12   ,        B    ,          7;0    ,         B    ,       7;0   ,       "Rule B"
13   ,        B.B  ,          8;0    ,         BB   ,       8;0   ,       "Rule BB"
14   ,        Fx   ,          9;0    ,         Fx   ,       9;0   ,       "Rule Fx"
15   ,        x1   ,          10;0   ,         x1   ,       10;0  ,       "Rule x1"
16   ,        x2   ,          11;0   ,         x2   ,       11;0  ,       "Rule x2"
17   ,        Ub1  ,          12;0   ,         Uc1  ,       12;0  ,       "Extra Rule 1"
18   ,        Ub2  ,          13;0   ,         Uc2  ,       13;0  ,       "Extra Rule 2"
```

Note:

The RuleGroups 17 and 18 are ignored by the correlation process because the rule results do not exist in either database. However, the entries are preserved in the system-generated rule-map file to allow you the opportunity to modify this map file for subsequent correlation activity.

Subset/Superset Combination

It is possible to have a rule-map file that has a combination of subset and superset mappings. That is, the rule-map file might not contain mappings for all of the rules in the input error databases (subset) but can also contain mappings for rules that are not present in the input error databases (superset). Example 24 shows the result of this mapping and the resulting tool behavior.

Example 24 Combined Subset and Superset Rule-Map File

```
Baseline Database Contents: F, F.1, B, Fx
Comparator Database Contents: F, F_1, BB, x1, x2

User-supplied Superset Rule-Map File:
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10  ,        F    ,        5;0    ,      F    ,      5;0  ,      "Rule F"
11  ,        F.1  ,        6;0    ,      F_1  ,      6;0  ,      "Rule F.1"
12  ,        B    ,        7;0    ,      B    ,      7;0  ,      "Rule B"
13  ,        B.B  ,        8;0    ,      BB   ,      8;0  ,      "Rule BB"
17  ,        Ub1 ,         12;0   ,      Uc1  ,      12;0 ,      "Extra Rule 1"
18  ,        Ub2 ,         13;0   ,      Uc2  ,      13;0 ,      "Extra Rule 2"
```

With these inputs, Example 25 shows the rule-map file generated by the DCV Results Compare tool for internal correlation.

Example 25 System-Generated Rule-Map File

```
RuleGroup BaselineRule Baseline(L;D) ComparatorRule Comparator(L;D)
 Description
10  ,        F    ,        5;0    ,      F    ,       5;0   ,      "Rule F"
11  ,        F.1  ,        6;0    ,      F_1  ,       6;0   ,      "Rule F.1"
12  ,        B    ,        7;0    ,      B    ,       7;0   ,      "Rule B"
13  ,        B.B  ,        8;0    ,      BB   ,       8;0   ,      "Rule BB"
17  ,        Ub1 ,         12;0   ,      Uc1  ,       12;0 ,      "Extra Rule 1"
18  ,        Ub2 ,         13;0   ,      Uc2  ,       13;0 ,      "Extra Rule 2"
1   ,        Fx  ,         0;0    ,      RCT_genC_Fx , 0;0   ,    "Rule Fx"
2   ,      RCT_genB_x1   ,  1;0    ,     x1    ,      1;     ,    "Rule x1"
3   ,      RCT_genB_x2   ,  2;0    ,     x2     ,     2;0    ,    "Rule x2"
```

Note:

As in the superset example, RuleGroups 17 and 18 are ignored by the correlation, but the entries are preserved in the system-generated rule-map file.

Comparison Results File

The DCV Results Compare tool produces a comparison report that describes how the comparator results correlate with the baseline results. This report contains a report header, a comparison summary, and a detailed comparison results section.

- The report header contains information about the tool release version, the total number of errors (from the baseline and comparator input), and the matched errors and unmatched errors (discrepancies) per rule.
- The comparison summary section summarizes each rule, indicating whether the rule has passed the comparison criteria or failed due to discrepancies.
- The detailed comparison results section gives detailed information about each rule. The section includes rule comments, rule identifiers, and any missed or false errors (discrepancies).

The report file name takes the form *top_cell*.RCT.report, where *top_cell* is the name of the top cell in the design.

For more information, see the following sections:

- Report Header
- Comparison Summary
- Detailed Comparison Results

Report Header

The report header contains the following information:

- Banner Information, which contains the tool release version
- Tool input information such as the baseline, comparator, and layout files and the tool called as information

Comparison Summary

The comparison summary section summarizes each rule, indicating whether the rule has passed the comparison criteria or failed due to discrepancies. The summary includes

- The rule group numbers, baseline and comparator rule identifiers, results (PASS or FAIL), total baseline errors, and total comparator errors.
- The number of discrepancies in the comparator results that are not in the baseline results (false errors).
- The number of discrepancies in the baseline results that are not in the comparator results (missed errors).

The results in this section are organized in a decreasing order of priority:

- Rules with the largest number of missed errors and no false errors
- Rules with the largest number of false errors and no missed errors

- Rules with the greatest difference between missed and false errors
- Rules with no discrepancies (passing results)

Two or more rules that have equal priorities are ordered based on their rule-group IDs.

Example 26 shows an example of a comparison report summary. The summary results are sorted based on severity. The Missed Errors column shows the number of error results in the baseline data that are not in the comparator data. The False Errors column shows the number of error results in the comparator data that are not in the baseline data.

Example 26   Comparison Report Summary Example

```
          ------------------
          COMPARISON SUMMARY          ( Total PASS=2 Total FAIL=13 )
          ------------------

Rule  Baseline  Comparator Result Total    Total      Missed False
Group Rule Name  Rule Name        Baseline Comparator Errors Errors

   3   RULE_2     RULE_2   FAIL   2739     3528        850        0
  14   RULE_4     RULE_4   FAIL    851      460        209        0
  48   RULE_F     RULE_F   FAIL    228       96        120        0
   9   RULE_A1    RULE_A1  FAIL      3        2         50        0
  18   RULE_B1    RULE_B1  FAIL      9        9         24        0
  32   RULE_C     RULE_C   FAIL     11        1          5        0
  26   RULE_6b    RULE_6b  FAIL  12467  1589096          0   125670
   5   RULE_8     RULE_8   FAIL   2735   633190          0    96500
  16   RULE_H     RULE_H   FAIL    417    64417          0     3015
  17   RULE_SS    RULE_SS  FAIL    328     1314         39      589
  21   RULE_SX    RULE_SX  FAIL     16       16         12       64
  22   RULE_S2    RULE_S2  FAIL    370      371         30        1
  20   RULE_Se    RULE_Se  FAIL  18497      481        108      124
  23   RULE_Se3   RULE_Se3 PASS    299      299          0        0
   2   RULE_9     RULE_9   PASS     46       42          0        0
```

In this example,

- The first six rule groups in the example have errors in the baseline data but none in the comparator data. The rules are sorted by number of discrepancies.
- The next three rule groups have errors in the comparator data but not in the baseline data. The rules are sorted by number of discrepancies.
- The next four rule groups have discrepancies in both the baseline and comparator data. The rules are sorted by largest delta to smallest delta.
- For the last two rule groups, the comparison matched. These results are sorted by the number of errors in the baseline data (Total Baseline).

The total number of violations reported for the baseline and comparator databases is, by default, a hierarchical count. If the `-flat_error` command-line option is specified, the total violations will be replaced by a flat count calculated based on the number of cell placements in the original layout hierarchy. See Example 27 for an illustration.

Note:

Missed and false errors are still reported hierarchically regardless of whether the `-flat_error` command-line option is specified.

Example 27 Comparison Report Summary With -flat_error

```
          ------------------
          COMPARISON SUMMARY          ( Total PASS=2 Total FAIL=13 )
          ------------------

Rule           Baseline     Comparator Result     Total      Total    Missed
   False
Group         Rule Name       Rule Name         Baseline Comparator    Errors
  Errors

   3          RULE_2       RULE_2    FAIL     8217     10584       850         0
  14          RULE_4       RULE_4    FAIL      851       460       209         0
  48          RULE_F       RULE_F    FAIL      456       192       120         0
   9          RULE_A1      RULE_A1   FAIL        9         4        50         0
  18          RULE_B1      RULE_B1   FAIL       18        18        24         0
  32          RULE_C       RULE_C    FAIL       44         4         5         0
  26          RULE_6b      RULE_6b   FAIL    18932   1842091         0    125670
   5          RULE_8       RULE_8    FAIL     7920   1000236         0     96500
  16          RULE_H       RULE_H    FAIL      683     89412         0      3015
  17          RULE_SS      RULE_SS   FAIL      656      2628        39       589
  21          RULE_SX      RULE_SX   FAIL       16        16        12        64
  22          RULE_S2      RULE_S2   FAIL      370       371        30         1
  20          RULE_Se      RULE_Se   FAIL    25269       490       108       124
  23          RULE_Se3     RULE_Se3  PASS      438       438         0         0
   2          RULE_9       RULE_9    PASS       92        84         0         0
```

It is important to note that flat error counts should never be used to indicate whether the passing or failing correlation results are correct. It is quite possible for different flat error counts to produce a PASS result, and it is equally possible for matching flat error counts to produce a FAIL result. There are many factors that can result in different error counts that have nothing to do with whether correlation results should match, including:

- Different code/functions used to generate the errors
- Different output data types specified to represent the errors
- Different tool/engine behaviors
- The comparison type chosen for the Results Compare tool run
- The processing method chosen for the Results Compare tool run

Detailed Comparison Results

The detailed comparison results section gives detailed information about each rule, including rule comments, rule identifiers, and any missed or false errors. The results are similar to the presentation of errors in a LAYOUT_ERRORS file; the main difference is the details provided with the discrepancies.

Note:

The output report format might change depending on the command-line options that you specify.

The report displays basic rule group information and the results for each missed and false error for each rule group ID in ascending order. Example 28 shows the detailed results for a rule group with both missed and false errors.

Example 28 Comparison Report Detailed Results Section

```
        ---------------------------
        DETAILED COMPARISON RESULTS
        ---------------------------

Rule Group: 1
Baseline Rule Names: Rule_ABC
Comparator Rule Names: RCT_Rule_ABC
Description: rule_comment_or_description
Compare Result: FAIL
-------------------------------------------------------------------------

Missed errors: 8
-----------------------------------------------------------
    RCT (lower left x, y) (upper right x, y)
-----------------------------------------------------------
    (683.8800, 1864.2140) (683.9600, 1864.9060)
    (688.6800, 1864.2140) (688.7600, 1864.9060)
    (683.8800, 1858.4540) (683.9600, 1860.4800)
    (679.0800, 1858.9200) (679.1600, 1860.4800)
    (678.7600, 1857.1200) (678.8400, 1858.6800)
    (664.3600, 1864.2140) (664.4400, 1864.9060)
    (663.4000, 1858.4540) (663.4800, 1860.4800)
    (661.1600, 1858.4540) (661.2400, 1860.4800)

False errors: 6
-----------------------------------------------------------
    RCT (lower left x, y) (upper right x, y)
-----------------------------------------------------------
    (683.8800, 1864.2140) (683.9600, 1864.9060)
    (688.6800, 1864.2140) (688.7600, 1864.9060)
    (683.8800, 1858.4540) (683.9600, 1860.4800)
    (679.0800, 1858.9200) (679.1600, 1860.4800)
    (678.7600, 1857.1200) (678.8400, 1858.6800)
    (664.3600, 1864.2140) (664.4400, 1864.9060)
```

If you specify a waiver database, the report contains additional sections for waived comparison discrepancies. Example 29 shows an example of waived comparison discrepancies.

Example 29 Comparison Report Detailed Results for Waived Discrepancies

```
            --------------------
            WAIVED DISCREPANCIES
            --------------------

    Rule Group: 21
    Baseline Rule Names: Rule_BCD
    Comparator Rule Names: RCT_Rule_BCD
    Description: rule_comment_or_description
    Compare Result: WAIVED
    -------------------------------------------------------------------------

    Missed errors waived: 2
    -----------------------------------------------------------
        RCT (lower left x, y) (upper right x, y)
    -----------------------------------------------------------
        (588.2150, 451.7400) (588.2200, 451.7450)
        (588.2100, 451.7350) (588.2150, 451.7400)

    False errors waived: 0
```

LVS Results Comparison

LVS results comparison consists of two stages of results: the extraction stage results and the compare stage results. The DCV Results Compare Tool compares them separately.

Running the Tool

The syntax and input for these stages are different. See the following sections for details about how to run the tool:

- Extraction Stage Results Comparison
- Compare Stage Results Comparison

The extraction stage results comparison is similar to the DRC flow, but provides a more detailed comparison of the text and device extraction errors. Unlike DRC rules comparison, the LVS extract stage compares more attributes than just coordinates.

The compare stage results comparison provides an overview of the differences of two LVS runs. You can see the differences in LVS results (equivalence type, compare options, error type, warning type, and statistics) from the output, which includes a summary for all equivalence points and a detailed report for each equivalence point.

Extraction Stage Results Comparison

The LVS extraction stage comparison results are similar to the DRC comparison results, but the LVS extraction stage results have more detailed support for text and device extraction errors.

The LVS extraction stage results comparison command-line syntax is

```
dcv_rct
    overlap | exact | fuzzy_exact
    baseline_data
    comparator_data
    layout_data
    [options]
```

For example,

`dcv_rct exact test1.LAYOUT_ERRORS test2.LAYOUT_ERRORS test.gds`

The syntax and usage is the same as the DRC Results Comparison. See Running the Tool. For text and device extraction errors, only `exact` is supported regardless of the setting in the command line. However, the user-defined setting is applied to the other DRC rules.

Note:

Only IC Validator versus IC Validator comparison is supported.

The DCV Results Compare tool performs a runset command-based error-versus-error comparison of the error data and creates a comparison report for text and device extraction errors. The tool generates a report file, top_cell.RCT.report, where top_cell is the name of the top cell in the design.

Compare Stage Results Comparison

The DCV Results Compare tool provides a well-constructed output report to quickly and easily find the difference in LVS compare stage results.

The LVS compare stage results comparison syntax is

`dcv_rct -lvs <baseline run directory> <comparator run directory>`

Use the run_details directory of the original IC Validator run to execute the compare stage comparison. Using this syntax, you can compare the input _lvs.log file and compare directory that are generated by the IC Validator tool. The input directory must have the following structure:

```
input_directory/
      layout_equiv_cell/
             sum.block.block
```

```
        layout_equiv_cell/
               sum.block.block
```

The RCT_LVS_result output directory, created by the DCV Results Compare tool, contains all of the output reports for compare stage results.

Comparison Results File

The DCV Results Compare tool produces several comparison reports for LVS Comparison that describe how the comparator results correlate with the baseline results.

For extraction stage results, the report contains a report header, a comparison summary, and a detailed comparison results section.

- The report header contains information about the tool release version, the total number of errors (from the baseline and comparator input), and the matched errors and unmatched errors (discrepancies) per rule.
- The comparison summary section summarizes each rule, indicating whether the rule has passed the comparison criteria or failed due to discrepancies.
- The detailed comparison results section gives detailed information about each rule. The section includes, rule identifiers, and any missed or false errors (discrepancies).

The report file name is *top_cell*.RCT.report, where top_cell is the name of the top cell in the design.

For the compare stage results, the output is in a directory created by the DCV Results Compare tool, with the following structure:

```
RCT_LVS_result/
    lvs_RCT.report
    equivs/
           block.block.report
           block.block.report
           block.block.report
              …
```

For more information, see the following sections:

- Report Header
- Extraction Stage Comparison Report
- Compare Stage Comparison Report

Report Header

The report header contains the following information:

- Banner Information, which contains the tool release version
- Tool input information such as the baseline, comparator, and layout files and the tool called as information

Extraction Stage Comparison Report

The DCV Results Compare tool report has two sections: the comparison summary and the detailed comparison summary results.

There are seven columns, five of which are the same as the DRC results report. Two new columns, Command and Error type, contains IC Validator command name and error type in the LAYOUT_ERRORS file or error database (PYDB).

Note:

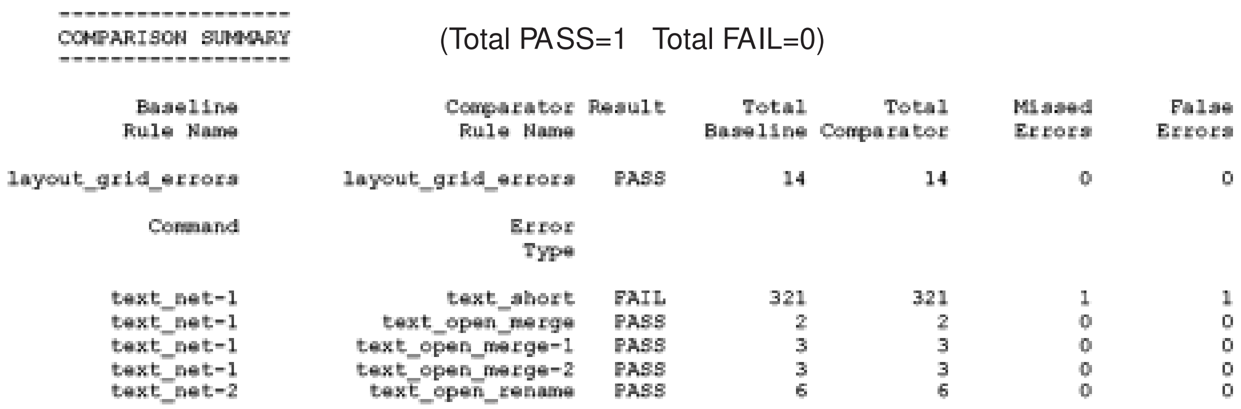
The header will not be generated if no corresponding rules or commands are found.

Runsets can contain multiple error producing text commands (for example, `text_net()`) and each of those text functions can have multiple types of error results (for example, short, opens, and so on).

To properly distinguish these results, a number ID is added to the end of the command name, as shown in Figure 12. The number ID is based on the order of the commands in the runset.

If a given text command produces multiple errors of the same type (for example, `text_open_merge`), a number ID is added at the end of subsequent error types, as shown in Figure 12.

Figure 12 Comparison Summary Results Number ID



```
      ------------------
      COMPARISON SUMMARY             (Total PASS=1   Total FAIL=0)
      ------------------

          Baseline               Comparator Result   Total       Total    Missed   False
         Rule Name                Rule Name        Baseline Comparator   Errors   Errors

layout_grid_errors       layout_grid_errors  PASS        14          14        0        0

           Command                    Error
                                       Type

         text_net-1              text_short  FAIL       321         321        1        1
         text_net-1         text_open_merge  PASS         2           2        0        0
         text_net-1       text_open_merge-1  PASS         3           3        0        0
         text_net-1       text_open_merge-2  PASS         3           3        0        0
         text_net-2        text_open_rename  PASS         6           6        0        0
```

For these error producing text commands, the Detailed Summary Results contents are slightly modified from typical DRC results comparison. An example of this is shown in Figure 13. The rule group identifier is removed and a “Compare Result” is included to indicate if the result is missing from the Comparator or Baseline.

Figure 13 Detailed Summary Results

```
------------------------------------------------------------------------------
Command: text_net-1
Error Type: text_short
Compare Result: Missing in comparator
------------------------------------------------------------------------------
EPEnMRAM_cheetah  11  *  LM_TNOR_ENB  143  0  (-246.658000,1278.254000)  met[mlyr_name_list[c]]  LAYER
                            TNOR_ENB  142  0  (-246.658000,1831.634000)  met[mlyr_name_list[c]]  LAYER
```

Compare Stage Comparison Report

The LVS compare stage comparison output is in a directory with the following structure:

```
RCT_LVS_result/
    lvs_RCT.report
    equivs/
           block.block.report
           block.block.report
           block.block.report
              …
```

The lvs_RCT.report file is higher level summary of the Compare Stage results comparison (see Figure 14 for an illustration). The report contains two sections. The first section is a comparison of the runset level, for example, global compare options settings. The second section contains an overview of the comparison for each sum.*block*.*block* file.

Figure 14 LVS Compare Summary Report

```
-----------------------------------------------------
                    COMPARE OPTIONS
-----------------------------------------------------

Options                       Baseline Settings                  Comparator Settings
-------                       -----------------                  -------------------
schematic global_nets          {}                                  {EXAMPLE}
power nets
   schematic                   {}                                  {EXAMPLE}

-----------------------------------------------------
             LVS RESULT COMPARE SUMMARY
-----------------------------------------------------

Compare     Equiv       Compare     Error       Warning     Netlist Stats  Netlist Stats  Equivs

Result      Type        Options     Messages    Messages    -Schematic     -Layout        schematic = layout

--------    --------    --------    --------    --------    -------------  -------------  -----------------------------------

OK          OK          UNMATCHED   UNMATCHED   OK          UNMATCHED      OK             sub = sub

OK          OK          UNMATCHED   OK          OK          UNMATCHED      UNMATCHED      top = top
```

The equivs directory contains individual*block*.*block*.report files, which are the detailed compare results for each sum.*block*.*block* file. See Figure 15 for an example of a *block*.*block*.report file.

Each *block*.*block*.report file is broken up into the following sections: EQUIV POINT, COMPARE RESULT, EQUIV TYPE, ERROR MESSAGES, WARNING MESSAGES, COMPARE OPTIONS, NETLIST STATISTICS - LAYOUT, and NETLIST STATISTICS - SCHEMATIC. These sections correspond to the blocks of information compared within each sum.*block.block* file.

If all results for a section match, a “MATCHED” result will be shown, except for the EQUIV POINT, ERROR MESSAGES, and WARNING MESSAGES sections. For these, all of the information will be presented, whether they match or not. This is also illustrated in Figure 15.

Figure 15 Detailed Summary Results

```
-----------------------------------------------------
                     EQUIV POINT
-----------------------------------------------------

schematic = layout
------------------
sub = sub

-----------------------------------------------------
                     COMPARE RESULT
-----------------------------------------------------

MATCHED

-----------------------------------------------------
                      EQUIV TYPE
-----------------------------------------------------

MATCHED


-----------------------------------------------------
                     ERROR MESSAGES
-----------------------------------------------------

Baseline Errors                         Count          Comparator Errors                       Count
---------------                         -----          ----------------                        -----
one-connection non-port layout net      1               one-connection non-port layout net     1
one-connection non-port schematic ne@   1
```

EQUIV POINT

The EQUIV POINT section of the report displays the cell name in the schematic design and layout, as shown in Figure 16.

Figure 16 EQUIV POINT Section

```
-----------------------------------------------------
                     EQUIV POINT
-----------------------------------------------------

schematic = layout
------------------
sub = sub
```

ERROR MESSAGES

The ERROR MESSAGES and WARNING MESSAGES sections list all of the errors and warnings whether or not they are different. You must determine if the errors or warnings are the same by reviewing the original LVS report. For example, in Figure 17, the first error message is the same for both LVS runs, but the detail of this error can be different.

Figure 17 ERROR MESSAGES Section

```
-----------------------------------------------------
                     ERROR MESSAGES
-----------------------------------------------------

Baseline Errors                         Count         Comparator Errors                       Count
---------------                         -----         ----------------                        -----
one-connection non-port layout net      1              one-connection non-port layout net     1
one-connection non-port schematic ne@   1
```

COMPARE OPTIONS

The COMPARE OPTIONS section is similar to section in the summary report, but it compares the LVS compare options for this equivalence instead of the runset setting, as shown in Figure 18.

Figure 18 COMPARE OPTIONS Section

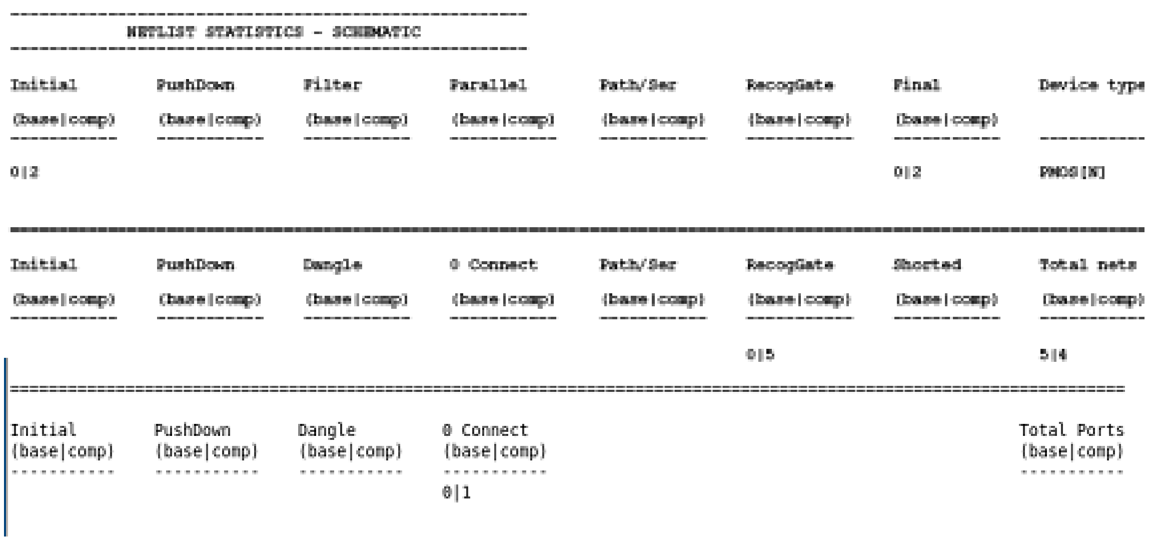
```
-----------------------------------------------------
                    COMPARE OPTIONS
-----------------------------------------------------

Options                      Baseline Settings                  Comparator Settings
-------                      -----------------                  -------------------
print_messages
   pre_merge_stats            false                               true
```

NETLIST STATISTICS

The NETLIST STATISTICS section lists the differences of entries about devices, nets, and ports in the statistics report; the entries without differences are not listed. Schematic and layout statistic differences are reported in three sections, as shown in Figure 19.

Figure 19 NETLIST STATISTICS- SCHEMATIC Section



```
------------------------------------------------------------
             NETLIST STATISTICS - SCHEMATIC
------------------------------------------------------------

Initial        PushDown       Filter         Parallel       Path/Ser       RecogGate      Final          Device type

(base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)
-----------    -----------    -----------    -----------    -----------    -----------    -----------    -----------

0|2                                                                                       0|2            PMOS[N]

=====================================================================================================================

Initial        PushDown       Dangle         0 Connect      Path/Ser       RecogGate      Shorted        Total nets

(base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)    (base|comp)
-----------    -----------    -----------    -----------    -----------    -----------    -----------    -----------

                                                                           0|5                           5|4

====================================================================================================================

Initial        PushDown       Dangle         0 Connect                                                   Total Ports
(base|comp)    (base|comp)    (base|comp)    (base|comp)                                                 (base|comp)
-----------    -----------    -----------    -----------                                                 -----------
                                             0|1
```

DPT Results Comparison

The DPT flow is applied to post-decomposition rules defined in the DPT map file. Post-decomposition rules are compared to respective odd cycles, as opposed to the comparison of baseline and comparator post-decomposition rules, which are used in the default DRC flow. The primary goal of this flow is to generate accurate DPT post-decomposition rule comparison results. Therefore, a two-step process is used: the odd cycle is compared first by using the OVERLAP mode, and the post-decomposition rules are compared to the respective odd cycle by the OVERLAP mode. In addition, the even cycle and anchor conflicts are used to enlarge the region of the odd cycle after the first step, if they are specified in the map file.

You must use a map file with this flow. The map file contains the following eight columns:

- METAL. The metal is colored. The metal name must be the same for the baseline and comparator, as the tool uses metal to link the baseline and comparator.
- ODD CYCLE RULE. Rule name of the rule used to report the odd cycle.
- ANCHOR CONFLICT RULE. Rule name of the rule used to report anchor conflicts.
- EVEN CYCLE RULE. Rule name of the rule used to report the even cycle.
- POST DECOMPOSITION RULES. Rule names of the rules based on post-decomposition results.
- COLORLESS LAYER. The layer and data type of colorless metal.
- PRE-COLOR1. The layer and data type of pre-colored 1 metal.
- PRE_COLOR2. The layer and data type of pre-colored 2 metal.

An example of the rule map file is displayed in Example 30.

Example 30 Map File Example

```
METAL    ODD      ANCHOR      EVEN     POST              COLORLESS    PRE-
  PRE-
         CYCLE    CONFLICT    CYCLE    DECOMPOSITION     LAYER        COLOR1
 COLOR2
         RULE     RULE        RULE     RULES
BASELINE:
  M1     A.OC;   A.ANCHOR;    A.EC;    {A.PD.1,A.PD.2};  15:0;        15:235;
 15:236;
  M2     B.OC;   N/A;         N/A;     {B.PD.1,B.PD.2};  17:0;        17:235;
 17:236;
  M3     C.OC;   C.ANCHOR;    C.EC;    C.PD;             19:0;        19:235;
 19:236;
COMPARATOR:
  M1     A.OC;   A.ANCHOR;    A.EC;    {A.PD.1,A.PD.2};  15:0;        15:235;
 15:236;
  M2     B.OC;   N/A;         N/A;     {B.PD.1,B.PD.2};  17:0;        17:235;
 17:236;
```

```
  M3      C.OC;    CC.ANCHOR;    C.EC;     CC.PD;             19:0;        19:235;
 19:236;
```

The map file must also meet these requirements:

- Each column must end with a semicolon. Use the contents exactly as they are in the header file.
- Multiple post-decomposition rules are provided in the {rule1,rule2}; format, as shown in the Example 31. The order of post-decomposition rules in the baseline and comparator must be matched, as rules are associated by their order in the map file. If the number of rules in the baseline and comparator are different, an error message occurs.

Example 31 Map File With Multiple Decomposition Rules

```
METAL    ODD      ANCHOR      EVEN     POST              COLORLESS     PRE-
  PRE-
         CYCLE    CONFLICT    CYCLE    DECOMPOSITION     LAYER        COLOR1
 COLOR2
         RULE     RULE        RULE     RULES
BASELINE:
  M1     A.OC;   A.ANCHOR;    A.EC;    {A.PD.1,A.PD.2};  15:0;        15:235;
 15:236;
COMPARATOR:
  M1     A.OC;   A.ANCHOR;    A.EC;    {A.PD.1,A.PD.2};  15:0;        15:235;
 15:236;
```

- Rule names can be different in the baseline and comparator, and in these cases, “BASELINE:” and “COMPARATOR:” must explicitly be listed. Baseline and comparator results are associated by the metal name. For the metal listed only in the baseline, the tool uses the baseline rule name as the rule name for the comparator.
- EVEN CYCLE RULE and ANCHOR CONFLICT RULE are defined as “N/A”, if they are not applicable.

  Note:

  The pre-color1 and pre-color2 layers must always be listed, even they are empty.

The DPT flow generates two new sections in the report: “DPT COMPARISON SUMMARY” and “DPT DETAILED COMPARISON RESULTS”. Example 32 shows the summary. In this example, each metal has a subsection. The comparison result of the odd cycle is listed at the beginning of this section. A warning message is provided if the odd cycle does not match.

Example 32 Detailed Report

```
              ------------------
              DPT COMPARISON SUMMARY
              ------------------
Metal            Baseline              Comparator              Result   Missed
  False
```

```
Name            Rule Name              Rule Name                       Errors
  Errors

 M1(15:0)    (odd_cycle)ODD_CYCLE_1  (odd_cycle)ODD_CYCLE_1    FAIL      1
     0
 WARNING: The following M1 post-decomposition rules comparison may not be accu
rate.
 M1(15:0)        R.Metal1.A             R.Metal1.A             FAIL      1
     1

 M1(15:0)    (odd_cycle)ODD_CYCLE_1  (odd_cycle)ODD_CYCLE_1    PASS      0
     0
 M2(17:0)        R.Metal2.A             R.Metal2.A             PASS      0
     0
 M2(17:0)        R.Metal2.B             R.Metal2.B             PASS      0
     0
```

Example 33 provides a detailed report including information on mismatching, which is similar to the DRC detailed results.

Example 33   DRC Comparison Results Example

```
            ---------------------------
            DPT DETAILED COMPARISON RESULTS
            ---------------------------

    Metal: M1(2:0)
    Baseline Rule Names: (odd_cycle)ODD_CYCLE_1
    Comparator Rule Names: (odd_cycle)ODD_CYCLE_1
    Compare Result: PASS
    -------------------------------------------------------------------------

    Missed errors: 1
    -----------------------------------------------------------
        RCT (lower left x, y) (upper right x, y)
    -----------------------------------------------------------
        (683.8800, 1864.2140) (683.9600, 1864.9060)

    False errors: 0


    -------------------------------------------------------------------------
    Metal: M1(2:0)
    Baseline Rule Names: R.Metal1.A
    Comparator Rule Names: R.Metal1.A
    Compare Result: PASS
    -------------------------------------------------------------------------

    Missed errors: 1
    -----------------------------------------------------------
        RCT (lower left x, y) (upper right x, y)
    -----------------------------------------------------------
        (661.1600, 1858.4540) (661.2400, 1860.4800)

    False errors: 1
```

```
---------------------------------------------------------
    RCT (lower left x, y) (upper right x, y)
---------------------------------------------------------
    (683.8800, 1858.4540) (683.9600, 1860.4800)
```

Design ECO Results Analysis

You can analyze DRC error results from two different PYDB database using the ECO flow. These databases represent the DRC results before(baselines) and after(comparator) a layout design change, commonly referred as ECO.

Running the Tool

The ECO flow (`-eco`) evaluates the effects of a layout design change in the DRC results. The before and after DRC results (PYDB) databases and the corresponding before and after design databases are specified as inputs for the ECO flow to identify. For example:

- Introduced errors (New)
- Resolved errors (Fixed)
- Unresolved errors (Unchanged)

In this flow, the before (baseline) and after (comparator) layout databases can have different hierarchies.

Use the same DRC runset and IC Validator engine versions for optimal before and after results. The ECO flow evaluates design change effects. Results analysis involving runset and engine changes must be done using DRC Results Comparison.

The syntax

```
dcv_rct -eco -base_error_db pydb input -base_library baseline layout \
-comp_error_db pydb input -comp_library gds/oasis
```

where

Syntax	Description
`-base_error_db`	Specifies DRC results database from the design before ECO. Currently, only PYDB is supported.
`-base_library`	Specifies the design before ECO.
`-comp_error_db`	Specifies the DRC results database from the design after ECO. Currently, only PYDB is supported.
`-comp_library`	Specifies the design after ECO.

Analysis Results Files

The ECO flow produces a summary report and VUE files for graphical debug of results.

Summary Report

A summary report sorts the violation counts into three categories: New, Fixed, and Unchanged. The file created in your RCT working directory is *top_cell_name*.RCT.report.

Example 34 Summary Report

```
-------------------------------------------------------------------------
Rule     New Violation     Fixed Violation     Unchanged Violation
-------------------------------------------------------------------------
Rule1     1                 0                    0
Rule2     0                 3                    3
Rule3     0                 1                    3
```

While reporting violation counts, the hierarchy of the new design after ECO is used except for the following cases:

- The violation is a fixed violation.
- Not all instances of a single violation are matched by baseline. For example, after the ECO flow, one placement of a cell with an error is deleted, however, the other five placements still contain the same violation. Whether the input PYDBs contain the violation in the top cell, or their own cell, one fixed violation and five unchanged violations are reported in the top cell.

  Note:

  The results for these two cases are reported in the top cell.

Graphical Debugging

The ECO flow creates multiple VUE files to provide for easy access to different permutations of the analysis results. The following list shows the VUE files created:

VUE File	Results
top_cell_name.RCT.vue	All (New, Fixed, Unchanged) violation results
*top_cell_name*_top.RCT.vue	All (New, Fixed, Unchanged) violation results in the top cell
*top_cell_name*_unchanged.RCT.vue	Unresolved (Unchanged) violation results
*top_cell_name*_unchanged_top.RCT.vue	Unresolved (Unchanged) violation results in the top cell
*top_cell_name*_fixed.RCT.vue	Resolved (Fixed) violation results

VUE File	Results
*top_cell_name*_fixed_top.RCT.vue	Resolved (Fixed) violation results in the top cell
*top_cell_name*_new.RCT.vue	Introduced (New) violation results
*top_cell_name*_new_top.RCT.vue	Introduced (New) violation results in the top cell

All PYDB files used by VUE files are created in the rct_details directory, however, those files are not standard PYDB files. Using these files without using the VUE files generated by the Results Comparison tool is not recommended.

Waiver Conversion Flow

The DCV Results Comparison Tool can be used to convert third-party waiver GDSII files to an equivalent IC Validator error classification database (cPYDB). The tool performs a rule-based correlation of the third-party waiver GDS shapes against a corresponding IC Validator PYDB error database. All waiver shapes, and associated waiver comments that correlate to a matching IC Validator error shape are written to the cPYDB.

Executing the cPYDB From the Results Compare Tool

To use the error classification database (cPYDB), set these arguments in the `error_options()` function:

```
match_errors_processing_mode = HIERARCHICAL

comment_match_delimiter = ":"
```

The `comment_match_delimiter` argument is required when running the waiver flow. If it is not set, violations might not be waived properly by the cPYDB created by the DCV Results Compare Tool.

Running the Tool

The waiver conversion flow command-line syntax is

```
dcv_rct
    overlap | exact | fuzzy_exact
    IC Validator PYDB
    3rd party Waiver GDSII
    layout_data
    [options]

    -m <Waiver Description File>
```

To convert the waivers from the file named Waiver.gds, the corresponding IC Validator PYDB database named PYDB_TOP, the original layout data in a file named TOP_input.gds, and the waiver description file named waiver_input.txt, type the following at the command line:

```
% dcv_rct exact Waiver.gds ./PYDB_TOP TOP_input.gds -m waiver_input.txt
```

Input Database Formats

For the waiver conversion flow, the supported database formats are limited to the following:

- IC Validator PYDB database
- Third-party waiver GDSII file

The tool automatically detects the formats so the databases can be specified in any order.

Waiver Description File

For the waiver conversion flow, you must specify a waiver description file. This file provides information about the mapping of the layer;datatype assignments in the third-party waiver GDSII data. The file must contain the following:

- Layer;Datatype assignment for the waiver shapes. Specifies the assignment of the waiver polygon marker layers.
- Layer;Datatype assignment for the waiver comments. Specifies the assignment of the waiver comment text. These comments are typically used to describe the justification for granting a waiver.
- Layer;Datatype assignment for the user name. Specifies the assignment for the name of the person who approved of the waiver.
- Layer;Datatype assignment for the date layer. Specifies the assignment for the date when the waiver was approved.

The format for specifying information is illustrated in the following example:

Example 35 waiver_input.txt

```
WAIVER SHAPES :  (164;99)
WAIVER COMMENTS :  (164;2)
USER NAME  :  (164;5)
DATE   :  (164;3)
```

Note:

The terminology and specification of information is based on the standard content supported by the third-party waiver GDSII database for classification and documentation of approved waivers.

Output

The DCV Results Compare tool waiver conversion flow produces the following output:

- IC Validator Error Classification Database (cPYDB)
- Correlation Report File

IC Validator Error Classification Database

The error classification database contains all the violations from the IC Validator PYDB that successfully correlated with the corresponding third-party waiver data. In addition to the correlated shapes data, all waiver comments, including user names, and dates are included in the output cPYDB. You can then use this automatically generated error classification database with follow-on IC Validator runs to apply the same waivers that were applied in the third-party DRC run.

Correlation Report File

The DCV Results Compare tool produces a report that specifies whether the conversion is successful. The report contains the header, correlation summary, and a detailed results section. See the Report Header section for more information.

Waiver Correlation Summary

The correlation summary section summarizes each rule, indicating whether the rule from the IC Validator PYDB has completely correlated with the third-party waiver or not. See Figure 20.

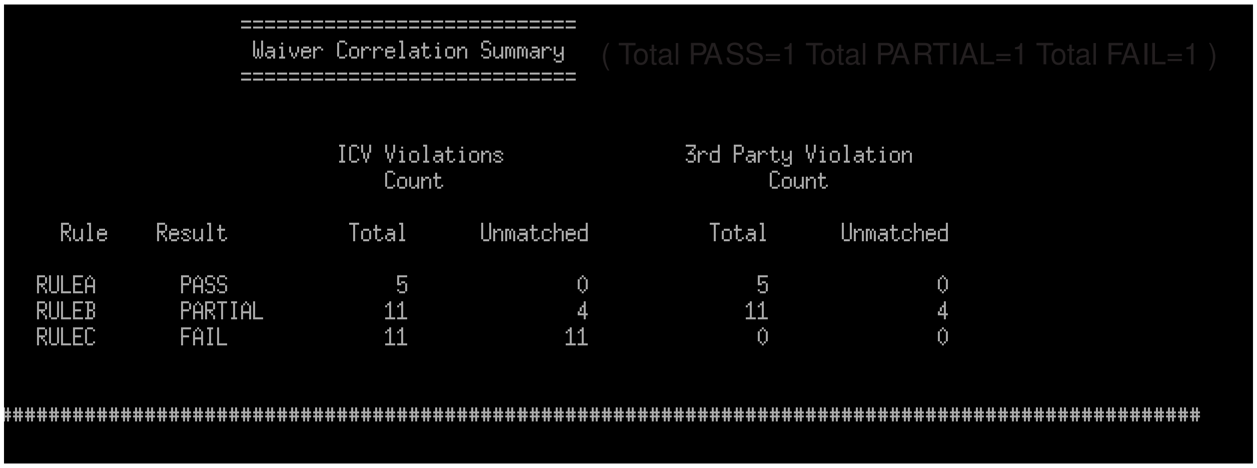
The summary includes the following:

- Rule name, correlation result for that rule, total violation counts for IC Validator and third-party inputs, and total unmatched violation counts for IC Validator and third-party inputs.

There are three possible correlation results:

- PASS. All of the IC Validator violations successfully correlated with all third-party waiver data for a given rule.
- PARTIAL. Some of the IC Validator violations successfully correlated with third-party waiver data but there are also some uncorrelated results. Successfully correlated results are written to the output cPYDB error classification database.
- FAIL. There was no successful correlation between IC Validator violations and third-party waiver data.

Figure 20 Waiver Correlation Summary



```
                  ============================
                    Waiver Correlation Summary    ( Total PASS=1 Total PARTIAL=1 Total FAIL=1 )
                  ============================


                            ICV Violations              3rd Party Violation
                                Count                          Count

   Rule     Result          Total        Unmatched         Total        Unmatched

 RULEA        PASS              5                0             5                0
 RULEB        PARTIAL          11                4            11                4
 RULEC        FAIL             11               11             0                0


######################################################################################################
```

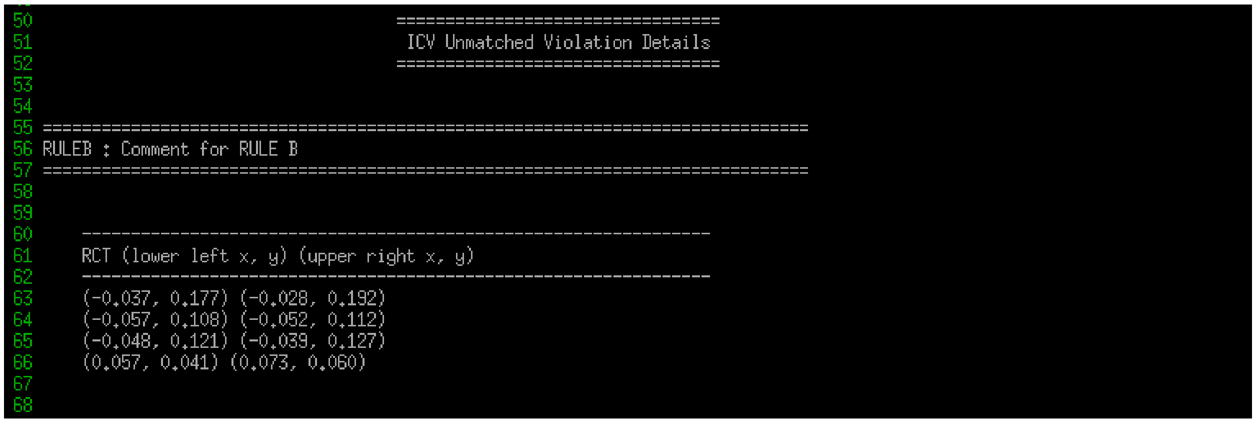
Detailed Results

The detailed results section provides detailed information about each unmatched rule, including rule comments and original cells in which the violation was reported, wherever applicable.

This section is separated into two components: one containing the unmatched violations from the IC Validator PYDB input and the other containing information from third-party waiver input.

The results are similar to the presentation of errors in a LAYOUT_ERRORS file; the primary difference is shown in Figure 21 and Figure 22.

Figure 21 Unmatched Violation Details



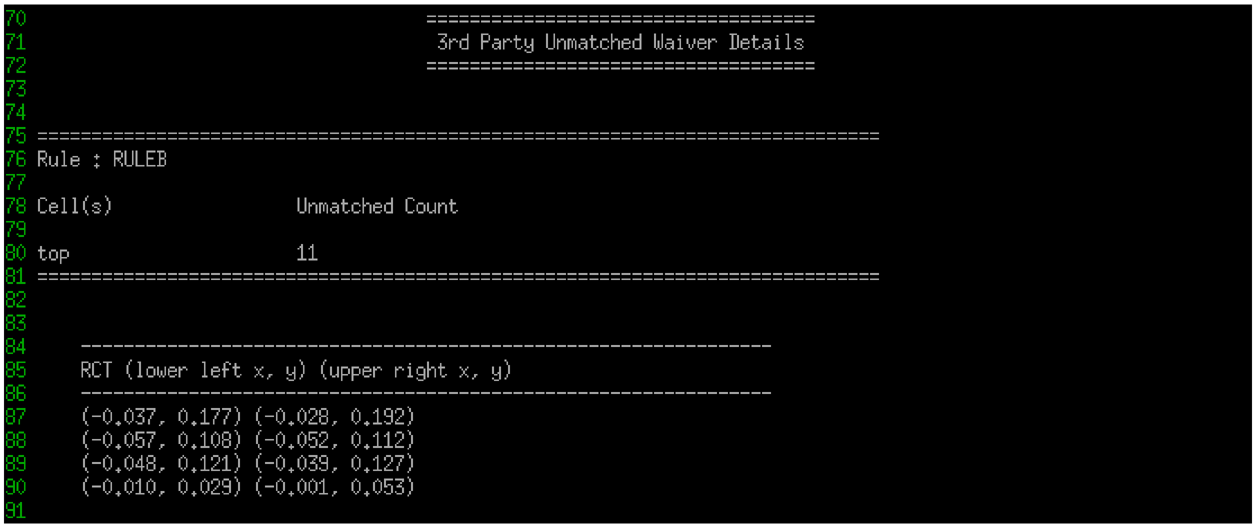
```
                                   ================================
                                    ICV Unmatched Violation Details
                                   ================================


============================================================================
RULEB : Comment for RULE B
============================================================================


    ---------------------------------------------------------------
    RCT (lower left x, y) (upper right x, y)
    ---------------------------------------------------------------
    (-0.037, 0.177) (-0.028, 0.192)
    (-0.057, 0.108) (-0.052, 0.112)
    (-0.048, 0.121) (-0.039, 0.127)
    (0.057, 0.041) (0.073, 0.060)

```

Figure 22 Third-Party Unmatched Waiver Details



```
                                     ====================================
                                      3rd Party Unmatched Waiver Details
                                     ====================================


==============================================================================
Rule : RULEB

Cell(s)                 Unmatched Count

top                     11
==============================================================================


    ---------------------------------------------------------------
    RCT (lower left x, y) (upper right x, y)
    ---------------------------------------------------------------
    (-0.037, 0.177) (-0.028, 0.192)
    (-0.057, 0.108) (-0.052, 0.112)
    (-0.048, 0.121) (-0.039, 0.127)
    (-0.010, 0.029) (-0.001, 0.053)

```

5

DCV Runset Coverage Tool

This chapter explains how to run the DCV Runset Coverage tool and view the reports.

The IC Validator DCV Runset Coverage tool evaluates the quality of DRC QA regressions by examining how well the regression data exercises a specified runset.

For more information, see the following sections:

Overview of the DCV Runset Coverage Tool

The DCV runset tool ensures that QA regressions are properly exercising the runset code. You use the DCV runset tool to test complex DRC runsets by constructing layout test cases (QA regression data).

When you want to evaluate the quality of DRC QA regressions and examine how well the regression data exercises a specified runset, use a DRC runset file and QA regression data as input to generate reports that provide the following information:

- Rule coverage
- Line coverage
- Empty assign layers
- Empty derived layers
- Rules depending on empty input layers

- Unexercised lines
- Unexercised rules

The tool also supports a batch processing mode to completely analyze DRC runsets containing conditional switches in a single process.

Command-Line Options

The DCV Runset Coverage tool command-line syntax is

```
dcv_runcov [options] runset-file input-design -i path
```

where

- *runset-file* specifies the DCV runset file to analyze.
- *input-design* specifies the path (directory) that contains the QA regression data (in GDSII or OASIS format).
- `-i` *path* specifies the path to the file containing iteration options for batch processing. See the Batch Processing Input File section for more information.

Table 11 describes the command-line options.

Table 11 DCV Runset Coverage Tool Command-Line Options

Argument	Description
`-empty_layers`	Outputs only information about empty layers in the runset.
`-h` `-help`	Displays the usage message and exits.
`-grid` *type*	Computes grid system type for distributed processing. The following are supported: • `SGE` (Sun or Univa Grid Engine) • `LSF` (Load Sharing Facility) • `NB` (Intel Netbatch) By default, the Runset Coverage tool is configured for an `SGE` system.
`-individual`	Disables merging of QA regression data.
`-line_coverage`	Outputs only information about line coverage.
`-rule_coverage`	Outputs only information about rule coverage.

Table 11 DCV Runset Coverage Tool Command-Line Options (Continued)

Argument	Description
`-slots` *`number`*	Allows the tool to run iterations in a compute grid. *number* is the maximum number of grid slots this tool can use. Specify `-1` for no user limit.
`-submit_cmd` *`command`*	Submits worker processes to a compute grid for distributed processing. "qsub -P bnormal" is the default command. **Note:** This command-line option applies to distributed processing only, for example, when `-slots` is specified.
`-tsv`	Outputs rule coverage information in a TSV-formatted file.

Batch Processing Input File

The DCV Runset Coverage tool supports using a batch processing file to exercise DRC runsets with multiple combinations of switch settings. This allows for separate runset coverage analysis for each combination and provides a summary of the coverage for all of the combined iterations. This coverage is enabled via a batch preprocessing input file.

Each line of the input file must be in this format:

```
<iteration> : <list of ICV command-line options>
```

The *<iteration>* identifies a run to apply the *<list of ICV command-line options>* to. The output reports refer to the results of that run by this identifier name.

A special iteration identifier named *GLOBAL*can be used to apply a list of command-line options to ALL iterations. If used, this identifier must be the first line of the input file. It is equivalent to specifying the same list of options for each *<iteration>*.

The *<list of ICV command-line options>* is the list of IC Validator command-line options that are normally specified for a DRC run. The format for this list is exactly the same as how these options appear normally on a command line. The DCV Runset Coverage tool uses these command-line options to run the IC Validator tool for that iteration.

The following example shows the batch processing input file:

```
GLOBAL : -dp16 -D FOO -D XYZ
RUN_ABC : -D BEOL_STACK=ABC -D …
RUN_DEF : -D BEOL_STACK=DEF -D …
RUN_QRS : -D BEOL_STACK=QRS -D …
```

This tells the DCV Runset Coverage tool to execute the following IC Validator runs:

```
icv -dp16 -D FOO -D XYZ -BEOL_STACK=ABC -D…
icv -dp16 -D FOO -D XYZ -BEOL_STACK=DEF -D…
icv -dp16 -D FOO -D XYZ -BEOL_STACK=QRS -D…
```

Distributed Processing

By default, the DCV Runset Coverage tool runs iterations serially. It is possible, however, to distribute processes to a compute grid so that iterations are run in parallel.

To enable distributed processing, specify the `-slots` command-line option as well as the maximum number of grid slots the tool can use. If `-slots` is `-1`, the maximum is the number of iterations.

When distributed processing is enabled, the tool submits worker processes to a Sun or Univa Grid engine via a “qsub -P bnormal” request by default. To change the grid engine and submit command, supply the `-grid` and `-submit_cmd` command-line options respectively.

Note:

The Runset Coverage tool assumes the specified compute grid system is set up.

Summary Report Files

The DCV Runset Coverage tool generates summary reports that provide the following:

- **Rule coverage**. Ratio of rules that produce error results, including “partial” results, to total number of rules. The presence of partial results is indicated by two asterisk characters after the ratio.
- **Line coverage**. Ratio of lines with commands that produce nonempty results to total lines with commands.

  Note:

  The following lines are not counted:

  - Unused assign layers.
  - Commands with no direct output, for example, `library()`, `error_options()`, `resolution_options()`, `write_gds()`, `write_oasis()`, and so on.
- **Empty Assign Layers**. Number and list of empty assign layers that are not used.

- **Empty Derived Layers**. Number and list of data creation layers (that is, layers that do not write results to the error database) that are empty.
- **Rules Depending on Empty Input Layers**. Number and list of rules that have one or more input layers (assign layers or derived layers) that are empty. These rules might have produced error results, but the fact that there are empty layers input to these rules indicates a situation where the regression data is not completely exercising those rules.
- **Rules With No Output**. List of rules that produce no error result.
- **Rules With Partial Output**. List of rules that produce “partial” error results. Under each listed rule is the ratio of error-generating commands that produce results to total error-generating commands and the list of error-generating commands producing no results.

Example 36 shows an example of a summary report file.

Example 36 Summary Report File

```
Called as: dcv_runcov runset.rs testcase/ -i input.txt

Input Runset : runset.rs
Input Layout Data : testcase/ (14 files evaluated)
Total Iterations : 2

                   ----------------------------
                     OVERALL COVERAGE SUMMARY
                   ----------------------------

Rule Coverage : 135/138 (97.8%)

Line Coverage : 2383/2572 (92.7%)
     Empty Assign Layers  : 6
     Empty Derived Layers : 3

Rules Depending on Empty Input Layers : 2 (1.4%)

** Includes "partially covered" rules. See "Rules with Partial Output."


                   ----------------------------
                         ITERATION SUMMARY
                   ----------------------------

Iteration Name    Line Coverage   Percentage     Rule Coverage   Percentage
RUN1              2383/2572       92.7%          135/138 **      97.8%
RUN2              3/3             100.0%         1/1             100.0%

Please check RunCov_iteration_reports/<IterationName>.report for details.

** Includes "partially covered" rules. See "Rules with Partial Output" in
RunCov_iteration_reports/<IterationName>.report.


                   ----------------------------
                     OVERALL COVERAGE DETAILS
                   ----------------------------

-------------------
```

```
Empty Assign Layers
-------------------

        LAYER_JoT
        LAYER_XOn
        LAYER_HyN
        LAYER_Ky1
        LAYER_gdC
        LAYER_RS3


--------------------
Rules With No Output
--------------------

        RULE_FCLX : Place high ...
        RULE_FVXH : Space to ...
        RULE_Fjlo : Width of VI ...


-------------------------
Rules With Partial Output
-------------------------

           Rule :  RULE_ALCX : No overlapping ...
       Coverage :  4/6 (66.7%)
    Zero Errors :  [runset.rs:1291] interacting(LAYER_153, ...);
                   [runset.rs:1292] interacting(LAYER_154, ...);


           Rule :  RULE_EXOR : External spacing ...
       Coverage :  1/2 (50.0%)
    Zero Errors :  [runset.rs:1076] external1(LAYER_198, ...);


-------------------------------
Rules Dependent on Empty Layers
-------------------------------

        RULE_F4vX : Core P+ ...
                                        -----------------
                                           Empty Layers
                                        -----------------
                                            LAYER_435
                                            LAYER_RS3

        RULE_FOPx : Floating gate is ...
                                        -----------------
                                           Empty Layers
                                        -----------------
                                            LAYER_JoT
                                            LAYER_478


--------------------
Empty Derived Layers
--------------------

+---------------------+---------------------------+------------------+
| Runset File (Line)  | Call Stack                | Runset Text      |
+---------------------+---------------------------+------------------+
| runset.rs (47)     | func_ptn@runset.rs:918   | LAYER_10 = ...; |
| runset.rs (552)    | @runset.rs:552.foreach-2 | LAYER_78 = ...; |
| runset.rs (625)    |                          | LAYER_35 = ...; |
+--------------------+--------------------------+-----------------+
```

The metrics for all iterations are summarized in the *runset*.RunCov.report, where *runset* is the base name of the runset file. Metrics for a specific iteration are reported in RunCov_iteration_reports/*iteration*.report, where *iteration* is the identifier name of the iteration.

If the `-individual` command-line option of the DCV Runset Coverage tool is specified, the file name of a specific iteration's report is *testcase_iteration*.report, where *testcase* is the file name of the individual layout test case used to exercise the runset. Additionally, the tool generates reports summarizing the metrics for all iterations from each layout test case. The report for each layout test case is found in RunCov_block_reports/*testcase*.report.

-svc and -uvc Command-Line Options

By default, reported metrics are based on all exercised lines in the runset. However, metrics for iterations that use the `-svc` or `-uvc` IC Validator command-line options are based on the exercised lines the selected rules are dependent upon.

TSV Rule Coverage File

The DCV Runset Coverage tool outputs rule coverage information in a standalone TSV-formatted file, which enables you to perform additional custom analysis. To generate this file, specify the `-tsv` command-line option. The generated file is called *runset*.RunCov.tsv, where *runset* is the base name of the runset file. This file is generated in addition to the usual run coverage report files.

The following examples show the rule coverage file for the default flow, and the file when the `-individual` command-line option is specified. Table 12 describes each of the fields in the file.

Examples

Example 1: TSV Rule Coverage File Format, Default Flow

```
Layout          Rule     RUN1  RUN2 RUN
 RUNCOVTOP     "Rule1"    1    -1    1
 RUNCOVTOP     "Rule2"   -1    -1    1
 RUNCOVTOP     "Rule3"    1    -1    1
```

Example 2: TSV Rule Coverage File Format, -individual Flow

```
Layout          Rule     RUN1  RUN2 RUN3
 top1.gds      "Rule1"     -1   -1     1
```

```
top2.gds      "Rule1"      1     -1    -1
top1.gds      "Rule2"     -1     -1     1
top2.gds      "Rule2"     -1     -1    -1
top1.gds      "Rule3"      1     -1     1
top2.gds      "Rule3"      1     -1     1
```

Table 12 Fields in a TSV Rule Coverage File

Field Name	Description
`Layout`	Identifies the layout database used. For the default flow, this is the top cell of the internally merged layout database, as shown in Example 1. For the `-individual` flow, this is the name of the individual regression layout database, as shown in Example 2.
`Rule`	Rule.

`<iteration 1>`
`<iteration 2>`
`<iteration 2>`
.
.
.
`<iteration N>`
	These fields provide the coverage information for each row (Layout/Rule). The name of each field corresponds to the iteration identifiers provided in the your Batch Processing Input File.
	Each field contains an integer with two possible values:
	• 1 indicates the rule is covered by the layout.
	• -1 indicates the rule is not covered by the layout.

Unexercised Report Files

It is possible that some runset content might not exercised by any iteration. In these cases, the DCV Runset Coverage tool generates an unexercised lines report named RunCov.unexercised.lines. The following example shows an unexercised lines report file.

Examples

Unexercised Lines Report File

```
File name      Line number       Line content
runset.rs      21                Rule3 @={ @ "BBB : comments3";
runset.rs      22                        C or B;
runset.rs      23                        C or D;
runset.rs      24                        }
runset.rs      25                Rule4 @={ @ "CCC : comments4";
runset.rs      26                        C not B;
runset.rs      27                        C not D;
runset.rs      28                        }
```

If the unexercised runset content contains rules, the tool also generates an unexercised rules report named RunCov.unexercised.rules. The following example shows an unexercised rules report file.

Unexercised Rules Report File

```
File Name      Line Number    Line Content
runset.rs      21             Rule3 @={ @ "BBB : comments3";
runset.rs      25             Rule4 @={ @ "CCC : comments4";
```

6

DCV Switch Optimizer Tool

This chapter explains how to run the DCV Switch Optimizer tool and view the reports.

The IC Validator DCV Switch Optimizer tool exercises combinations of runset preprocessor flow control switches to identify runset compile errors.

For more information, see the following sections:

Overview

Many DRC runsets consist of switch blocks that are complex in nature. The manual process of identifying the necessary switch combinations that exercise the entire runset code and detecting syntax errors in the runset is often inefficient.

The DCV Switch Optimizer tool reduces manual intervention and allows you to exercise the entire runset code optimally and qualify a runset free of syntax errors. The tool exercises a minimal number of user-defined flow control switch combinations to identify possible runtime syntax errors. This contrasts with manual runset validation processes that can be error prone (unexpected combinations) or overly conservative, resulting in more runset syntax-testing iterations than necessary.

The DCV Switch Optimizer tool takes input from a DRC runset file. The tool scans the runset file to identify all of the switches present in the runset and identifies *a minimal* set of switch combinations that exercise all of the runset code. The tool then parses the runset using those switch combinations and produces an output summary report. If there are failures for any iteration, the tool produces a separate log file for each failing iteration.

The tool can also take input from an optional switch information file. You use this file to provide context information about the switch types that guide the analysis. The context information identifies the following switch cases:

- Mutually exclusive switch cases
- Default and default list switch cases
- Ignore switch cases
- Optional values switch cases

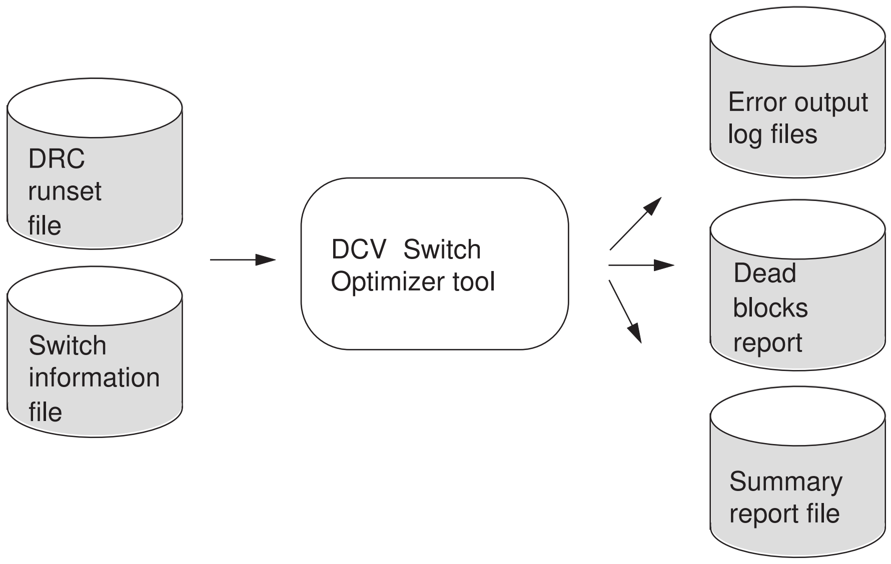
For more information, see the Runset Information File section.

The DCV Switch Optimizer tool performs the following tasks in its main flow:

1. Examines the entire runset and identify all code blocks and the switches or switch combinations that exercise all code
2. Determines the minimum set of switch combinations that fully cover the runset code
3. Runs IC Validator iteratively with each set of switch combinations
4. Produces an output summary report and, (when applicable) a report identifying dead code blocks, and log files with error details for failing iterations

Figure 23 illustrates this flow.

Figure 23 DCV Switch Optimizer Tool Flows



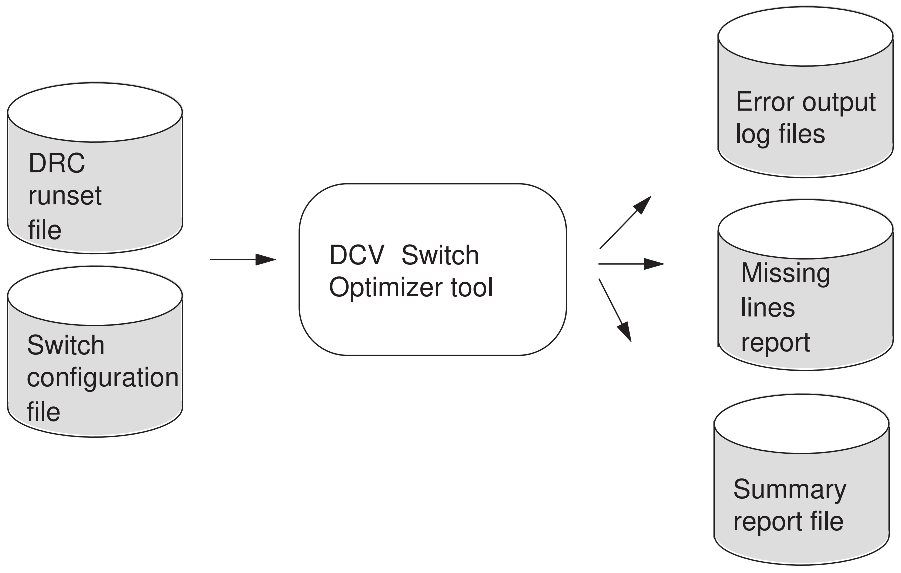
The DCV Switch Optimizer tool supports an alternative method of analysis where you specify your own set of switch combinations to run instead of having the combinations generated. This is accomplished by providing a separate switch configuration file. For more information, see the Switch Configuration File section.

The DCV Switch Optimizer tool performs the following tasks in this flow:

1. Runs IC Validator iteratively with each set of switch combinations fro the Switch Configuration file
2. Detects lines are not covered in the set of switch combinations from the Switch Configuration File
3. Produces an output summary report and, (when applicable) a report identifying missed lines, and log files with error details for failing iterations

Figure 24 illustrates this flow.

Figure 24 DCV Switch Optimizer Tool Flows



Command-Line Options

The DCV Switch Optimizer tool command-line syntax is

`dcv_swop` *`runset-file`* `[command-line-`*`options`*`]`

where *`runset_file`* specifies the DCV runset file.

Table 13 describes the command-line options.

Table 13 DCV Switch Optimizer Tool Command-Line Options

Argument	Description
`-h` `-help`	Displays the usage message and exits.
`-gen_sc`	Creates a switch configuration file to report all iterations.
`-i` `-input`	Specifies the path to the runset switch description file.

Table 13 DCV Switch Optimizer Tool Command-Line Options (Continued)

Argument	Description
`-I path`	Specifies the preprocessor include directories. This command-line option takes only one path, but you can use multiple `-I` options on a command line. When you specify multiple paths, order matters. For example, you have the file aaa.rh in both my_path1 and my_path2. • When you specify `-I my_path1 -I my_path2`, the aaa.rh file from my_path1 is used. • When you specify `-I my_path2 -I my_path1`, the aaa.rh file from my_path2 is used. Include directories are searched after the current working directory is searched. If an include directory is not in the install directory, then for the `-I` command-line option, you must specify the absolute path of the directory.
`-np`	Set the number of processes for the Switch Optimizer run (maximum is 16, default is 8).
`-pex_off`	Disables runtime syntax error checking for PEX maps.
`-report_switches`	Reports all switch names.
`-sc`	Specifies the path to the user-specified switch configuration file.
`-V` `-version`	Prints the tool version.

Runset Information File

The DCV Switch Optimizer tool takes input from a DRC runset file and an optional runset switch description file. You should provide a runset switch description file as input if the DRC runset file contains any special switch types.

For example, to specify mutually exclusive switch types, assume that you have the following switches:

```
SwitchA
SwitchB
SwitchC
SwitchD
SwitchE
SwitchF
SwitchG
SwitchH = dx_YES
```

```
SwitchI
SwitchJ = dx_NO
```

These switches have the following mutually exclusive conditions:

- `SwitchA`, `SwitchB`, and `SwitchC` are mutually exclusive with each other
- `SwitchB` is mutually exclusive with `SwitchD` and `SwitchE`
- `SwitchE` and `SwitchF` are mutually exclusive with each other
- `SwitchH = dx_YES` and `SwitchJ = dx_NO` are mutually exclusive with each other

Use the following format to describe these mutually exclusive switch conditions:

```
#MUTEX START
       SwitchA {SwitchB, SwitchC}
       SwitchB {SwitchD}
       SwitchB {SwitchE}
       SwitchE {SwitchF}
       SwitchH = dx_YES {SwitchJ = dx_NO}
#MUTEX END
```

Note:

The switches within curly braces should always be on the same line in the runset switch description file.

For more information about the special switch types that you can describe in this file, see the following sections:

- Mutually Exclusive Switch Cases
- Default and Default List Switch Cases
- Ignore Switch Cases
- Optional Values Switch Cases

Mutually Exclusive Switch Cases

Use the mutually exclusive switch case to identify switches and values that should not be allowed at the same time. For example, if the following switches are turned on at the same time, a syntax error results:

```
#ifdef LAYOUT_FORMAT_GDS
   aFFO = assign ( {{ 1, 0 }} );
#endif

#ifdef LAYOUT_FORMAT_OASIS
   aFFO = assign ( {{ 100, 0 }} );
#endif
```

You can avoid having to specify mutually-exclusive switches in the runset switch description file by adding an error detection case. For example:

```
#ifdef LAYOUT_FORMAT_GDS
   #ifdef LAYOUT_FORMAT_OASIS
      #error "Can't do this"
   #endif
#endif
```

The DCV Switch Optimizer tool automatically detects these error cases and prevents corresponding switches from being active at the same time.

Default and Default List Switch Cases

Use the default switch case to identify switches that must be set to some value every time the runset is used. These switches fall into one of the following categories:

- Single default value
- List of default values

Single default value switches must be set to a specific value for every iteration. This is typically done to establish a known startup state and is not meant to be modified by the runset users. For example:

```
#DEFAULT START
ABC = ON
FFO = ON
BRR = TOT
PQR = OFF

#DEFAULT END
```

The result is a runset with the following switch settings for each iteration:

- `#define ABC`
- `#define FFO`
- `#define BRR TOT`
- `#undef PQR`

Note:

You do not need to specify entries for `#define` statements used to set variables to a specific value (for example, macro expansions).

Switches that use a list of default values must be set to a value for every iteration, and each value must come from a defined list of valid choices. The DCV Switch Optimizer tool chooses a random value from the list and sets the switch accordingly. A new value is chosen for every iteration. For example:

```
#DEFAULT START
ABC = {value_1, value_2, value_3}
#DEFAULT END
```

If you specify either an invalid value or a missing value for the default list of switches in the user-input file, the tool reports a warning message to STDOUT. For example:

```
WARNING: Invalid value(s) specified.
         Please check the <user-input> file.
         Switch = PQR , Value = GREEN
         Switch = ABC , Value = RED

WARNING: Missing value(s) detected for the following switch(es).
         Please check the <user-input> file.
         Switch = PQR, Value = BAR
         Switch = ABC , Value = BLUE
         Switch = ABC , Value = FOO
```

Note:

Default switches cannot also be specified in the mutually exclusive section.

Ignore Switch Cases

The ignore switch case refers to switches that are to be left alone by the DCV Switch Optimizer tool. These switch settings, and their resulting behavior, are left as coded in the original runset.

This type of switch can be used to control the inclusion of control of the inclusion of common blocks of PXL code, as shown by the following example:

```
#ifndef ASSIGN_RS
        #include assign.rs
        #define ASSIGN_RS
#endif
```

In this example, the `ASSIGN_RS` switch is used to make sure that the assign.rs file is included only one time. This coding style relies on the switch not being turned on at the start.

If the DCV Switch Optimizer tool tries to set the `ASSIGN_RS` switch to `ON` as part of testing all of the runset code blocks, the assign.rs file are not included. This results in a syntax error because the layers that the tool expects subsequent code to define in the file are not be created.

Example 37 Ignore Syntax Example

```
#IGNORE START
ASSIGN_RS
#IGNORE END
```

Note:

An alternative way to handle this example is to use the Single Value Default case to set `ASSIGN_RS` to `OFF`.

```
#DEFAULT START
ASSIGN_RS = OFF
#DEFAULT END
```

Optional Values Switch Cases

Use the optional values switch case to provide optional values to the switches. You can apply the values in this switch case to any combination the switches are present in. For example:

```
#OPTIONAL START
ABC = {val}
PQR = {val2}
#OPTIONAL END
```

Note:

Optional switches cannot have multiple values.

Switch Configuration File

You can specify a set of switch combinations by using switch configuration file along with your runset file. These user-specified combinations replace the iterations normally produced by the Switch Optimizer Tool. When you set these switch combinations, runset code blocks might be missed. The Switch Optimizer tool reports these lines in the Missing Lines Report File.

This switch configuration file is in CSV or JSON format. Example 38 shows the switch configuration file in CSV format with four user-specified iterations. Example 39 shows the switch configuration file in JSON format.

Example 38 DCV Switch Configuration File in CSV Format

```
id,SwitchA,SwitchB,SwitchC,SwitchD
SWOP_DEFINE,SWOP_DEFINE,BB,DD,SWOP_DEFINE
2,SWOP_UNDEFINE,CC,DD,SWOP_DEFINE
3,SWOP_UNDEFINE,BB,EE,SWOP_UNDEFINE
4,SWOP_DEFINE,CC,SWOP_UNDEFINE
```

Example 39 DCV Switch Configuration File in JSON Format

```
{
    "iter_1": {
        "SwitchA": true,
        "SwitchB": "BB",
        "SwitchC": "DD",
        "SwitchD": true
    },
    "iter_2": {
        "SwitchA": false,
        "SwitchB": "CC",
        "SwitchC": "DD",
        "SwitchD": true
    },
    "iter_3": {
        "SwitchA": false,
        "SwitchB": "BB",
        "SwitchC": "EE",
        "SwitchD": false
    },
    "iter_4": {
        "SwitchA": true,
        "SwitchB": "CC",
        "SwitchC": false,
        "SwitchD": false
    }
}
```

Both of these examples create the following switch combinations:

```
SwitchA ON,  SwitchB == BB, SwitchC == DD and SwitchD ON
SwitchA OFF, SwitchB == CC, SwitchC == DD and SwitchD ON
SwitchA OFF, SwitchB == BB, SwitchC == EE and SwitchD OFF
SwitchA ON,  SwitchB == CC, SwitchC    OFF and SwitchD OFF
```

In CSV format, the first line of the switch configuration file starts with the switch ID followed by the switch names separated by a comma. Each subsequent line starts with the iteration number followed by the value of each of the switches mentioned in the first row.

For Boolean switch values, use `SWOP_DEFINE` to turn the switch `OFF` and `SWOP_UNDEFINE` to turn the switch `ON`. For all other switches, the switch values must *not* be enclosed in double quotation marks (`"`).

In JSON format, each iteration must be an element with key value pairs to indicate switch and its value. For Boolean switch values, use `true` to turn the switch `ON` and `false` to turn the switch `OFF`.

Specify the `-gen_sc` command-line option to let the Switch Optimizer Tool generate a switch configuration file, iterations.sc for the current run without an iteration parse check. For any Switch Optimizer Tool run, iterartions.sc can be found at the end of the run in the current working directory.

Error Output Log Files

When the DCV Switch Optimizer tool identifies a set of switch options that produces a syntax error, the tool produces an error log file. This file contains the IC Validator output for the run and describes why the iteration failed. The DCV Switch Optimizer tool creates an error log file for each iteration that has a syntax error. The file names have the format ParserIteration_*number*.log, where *number* is the iteration number where the error occurs.

Summary Report File

The tool produces a report file that summarizes all of the iterations. The report contains the following information:

- Total number of passed iterations
- Total number of failed iterations
- Results of all of the iterations
- Detailed descriptions of iterations that result in a syntax error

The report includes the following sections:

- *Parser Results Summary*

  This section lists the iteration numbers and results. The results are either PASS or FAIL. For failed iterations, the report also includes the path to the iteration-specific log file.

  The following example lists an iteration that failed and an iteration that passed:

```
Parser Iteration 1: FAIL - Check ParserIteration_1.log file for more
 info...
Parser Iteration 2: PASS
```

- *Parser Results Details*

  This section contains a detailed description of each iteration, including those that result in syntax errors. This information includes:

  - switch settings corresponding to the iteration
  - environment variables before running the iteration
  - any PEX map runtime syntax errors
  - an overall PASS or FAIL indicator
  - an IC Validator command-line call to reproduce the result

The following example shows the description of an iteration that failed:

```
###########################
# Switch Combination 1 :  #
###########################

TURN ON :
            d_11M_3MX_4FX_2HX

TURN OFF :
            d_CELL_FINE_SS_YES

Switch values :
            d_SRULES_MERGED=dx_NO

Environment variables :
            export d_ENV_11M_FD_9DF_VER=1
            unset  d_ENV_11M_FD_8DF

Evaluate to TRUE :
            SVALUE_CHECK(d_SRULES_MERGED)

Evaluate to FALSE :
            SVALUE_CHECK(d_11M_3MX_4FX_2HX)
            VERSION_GT(2016, 12, 0, 0)

ICV call : icv -nro -c swop_test_top_cell -i
/remote/us54home1/user/empty/swop_details/dcv_swop_test.oas -f OASIS
 -D
d_11M_3MX_4FX_2HX -D d_SRULES_MERGED=dx_NO runset.rs

ICV PARSE RESULT : FAIL
PEX MAP RESULT   : FAIL
[Error 1] [/remote/us54home1/user/case/pxl.rs] [Line: 160] : Layer
 "M3" is
used in pex_conducting_layer_map() but not in any active connect()
 database
or resistor() device body
[Error 2] [/remote/us54home1/user/case/pxl.rs] [Line: 86] : Layer "M1"
 is
used in an active connect() database but not in any pex_*_layer_map()
function

RESULT : FAIL
```

Note:

Reported environment variables must be set before running the IC Validator command-line to reproduce the result.

Note:

The IC Validator command-line call might be incomplete for some iterations. The command line does not factor in switch inputs that are required for macros in conditional directives to evaluate to true or false.

Example 40 shows the DCV Switch Optimizer report file.

Example 40   DCV Switch Optimizer Report File

```
Runset Switch Combinations - Parser Results Summary
===================================================

Results:
--------
Parser Iteration 1 : FAIL - Check ParserIteration_1.log file for more info...
Parser Iteration 2 : PASS
Parser Iteration 3 : PASS
Parser Iteration 4 : PASS
Parser Iteration 5 : PASS
Parser Iteration 6 : PASS

Total no of. PASSED Iterations : 5
Total no of. FAILED Iterations : 1

Runset Switch Combinations - Parser Results Details
===================================================

###########################
# Switch Combination 1 :  #
###########################

TURN ON :

            SWITCH_ABC
            SWITCH_CDE
            SWITCH_PQR
            SWITCH_XYZ

TURN OFF :

            SWITCH_LMN
            SWITCH_FFO

Switch values :

            SWITCH_BRR=Mx_NO
            SWITCH_TOT=Mx_YES

Evaluate to TRUE :
            VERSION_GT(2016, 12, 0, 0)

ICV call : icv -nro -c swop_test_top_cell -i /path/to/the/dcv_swop_test.gds
-D SWITCH_ABC -D SWITCH_CDE -D SWITCH_PQR -D SWITCH_XYZ -D SWITCH_BRR=Mx_NO
-D SWITCH_TOT=Mx_YES
runset.rs

ICV PARSE RESULT : FAIL
PEX MAP RESULT   : N/A
RESULT : FAIL
```

```
*****
```

Missing Lines Report File

When you provide your own set of switch combinations using the switch configuration file, it is possible that some runset content will not be exercised. In these cases, the Switch Optimizer tool generates a missing lines report.

Example 41 shows the format of the file name, *summary_report*_missing_lines.

Example 41 DCV Switch Optimizer Missing Lines Report File

```
path_1: /my_runset/Include
path_2: /my_runset/Include/pex_map

File name          Line number<s>     First Line
path_1/runset.rs   291                pex_remove_layer_map(pex_matrix,gSD_S...
path_2/runset.rs   295-300            pex_remove_layer_map(pex_matrix,gSE_S...
```

Dead Block Report File

In runset development, it is possible to inadvertently create sections of code that cannot be exercised regardless of the switch combinations chosen. These blocks are referred as dead code blocks.

Example 42 shows a runset with dead code blocks.

Example 42 Runset With Dead Code Blocks

```
#ifdef SwitchA
       #undef SwitchB
#endif
#ifdef SwitchA
       Code; // block1
       #ifdef SwitchB
              Code; // block2
              Code line 2;
              # include "runset.rs"
       #endif
#endif
```

In this case, code block2 is never exercised because any time that SwitchA is turned ON, SwitchB is turned OFF. The DCV Switch Optimizer tool generates the following dead code blocks report file,*summary_report*_dead_blocks.

Example 43 shows a dead code blocks report file.

Example 43 DCV Switch Optimizer Dead Code Blocks Report File

```
path_1: /control_switch/
path_2: /control_switch/Include/
```

```
File name         First Line number   First Line
path_1/main.rs    7                   Code; // block
path_2/runset.rs  1                   pex_remove_layer_map(pex_matrix,gSE_S...
```

This report file consists of two parts:

- Parent directories for all files containing dead code blocks.
- List of all dead code blocks including their filename and path, relative to the parent directory path. Only the first line of each dead block is reported.

Feedback

7

DCV Test Case Compiler Tool

This chapter explains how to run the DCV Test Case Compiler tool.

The IC Validator DCV Test Case Compiler tool generates non-functional layout database files for use in testing IC Validator runsets.

For more information, see the following sections:

- Overview
- Prerequisites
- Running the Tool
- Router Capabilities

Overview

The DCV Test Case Compiler tool (`dcv_tcc`) can be used to create a variety of layouts for testing a new IC Validator runset for both functional and performance purposes. The DCV Test Case Compiler tool takes an abstract description via a configuration file and creates a layout database (GDSII or OASIS) to use in testing the IC Validator runset. These layouts are non-functional, but ideally realistic enough to be useful for testing various aspects of the IC Validator runset, such as DRC, LVS, FILL, and so on.

Prerequisites

The DCV Test Case Compiler tool (`dcv_tcc`) requires an IC Validator WorkBench (ICVWB) license to run.

Running the Tool

Before running the DCV Test Case Compiler tool, make sure the `LM_LICENSE_FILE` environment variable and your *`ICV_INSTALL_DIRECTORY`* installation directory are set. The `dcv_tcc` executable is located in the same bin directory as the `icv` executable. All output, that is, logs and new layout, are generated in the current working directory.

The DCV Test Case Compiler tool command-line syntax is

```
dcv_tcc [options]
```

or

```
dcv_tcc [config_file]
```

where

- *options* are described in Table 14 section.
- config_file is described in the Configuration File section.

The recommended flow is to generate a configuration file using the `-g` command-line option, edit that file, and resubmit it to the tool to generate the final GDSII or OASIS file.

Note:

More hierarchical structures can be created by running the tool iteratively. For more information, see Iterative Execution.

For more information, see the following sections:

- Command-Line Options
- Environment Variables
- Configuration File
- Iterative Execution

Command-Line Options

Table 14 describes the command-line options.

Table 14 DCV Test Case Compiler Command-Line Options

Option	Description
`-g -genConfig`	Prints an example configuration file with comments, which can be edited and re-input into the tool.
`-V` `-version`	Prints the tool version.
`-h` `-help`	Displays the usage message and exits.

Table 14 DCV Test Case Compiler Command-Line Options (Continued)

Option	Description
`-pyRule`	Generates a rules.py file. See Router Capabilities for more information.

Environment Variables

You can control the appearance of the standard output and log files by setting the following DCV Test Case Compiler environment variables:

- `DCV_PROGRESS_SECONDS`

  Set this variable to change the frequency of Progress Line updates. The value must be an integer. The default is `60`.

- `DCV_TCC_ICVWB_PATH`

  This is the directory containing the `icvwb` executable. If the `icvwb` executable is in your path before launching the run, the tool uses it, and this environment variable is not needed. If this is set, it overrides the path in your regular environment. It is recommended that you use a comparable version to the version of IC Validator being used. An IC Validator WorkBench license is required to run this tool.

- `DCV_TCC_WORKDIR`

  Set this variable to dictate where any intermediate files go for the run. If not set, the files print to the current working directory.

Configuration File

The configuration file contains all of the specific information about how the DCV Test Case Compiler tool generates the final GDSII or OASIS file. Use the `-g` command-line option to generate an example configuration file, which can be edited and resubmitted to the tool.

The configuration file is setup in a *keyword=value* context. Lines beginning with a pound sign (#) are considered comment lines. Many of the keywords are binary in nature, in which case 0 is considered false and 1 is considered true. Keywords and descriptions are:

Table 15 describes the DCV Test Case Compiler configuration files.

Table 15 DCV Test Case Compiler Configuration Files

Keyword	Description
`tccMode` *`ENUM`*	Available `tccMode` enumerators: `cellRepeater`: Takes a list of cells and other directives and creates a new layout with the specified number of copies. `addWires`. Creates a coarse wiring grid inside an existing GDSII database. repeat&Route. Runs the cellRepeater keyword followed by the `addWires` keyword. **Note:** The keyword precedence for `cellRepeater` and `repeat&Route` is applied when forcibly merging cell names or making them unique: `depth` `addSuffix` `removePrefix` `mergeLeafCells` `retain` `rename`.
`preventCollisions` *`ENUM`*	Available `preventCollisions` enumerators: `Never`: Does not rename cells. The last cell is selected if cells from different files have the same name. `AsNeeded`: Renames only those cells that match a previously used cell in a prior input file. The renamed cell will have a “File#_” prefix. This includes top cells, which you can anticipate by adding the appropriate prefix to your `repeatCells` list. `Always`: Renames all cells with a “File#_” prefix. You must anticipate this in your `repeatCells` list.
`inputLayout` *`gds/oasis file list`*	Specifies the input GDSII or OASIS comma-separated file list. Merges only those cells in the `repeatCells` list and takes the cell from the last file if there is a conflict. This keyword is required.
`repeatCells` *`cell-list`*	Specifies a comma-separated list of cells to repeat (A,B,C), and accepts Tcl-based regular expression examples: • .* for all cells • .*INV.* for anything with INV • .*[^.*INV.*].* for anything without INV **Note:** You can create uneven ratios of cells A,A,B, which places two instances of cell A for every instance of cell B. All cells are considered regular expressions. A warning message is emitted if the regular expression cannot be found in any `inputLayout`, if none are found the job will fail. This keyword is required.
`repeaterMode` *`ENUM`*	Specifies `count` to place a specific number of instances, `size` to fill a specific area, or `eachTopCell` to place one of each topCell from the input data. This keyword is required.

Table 15 DCV Test Case Compiler Configuration Files (Continued)

Keyword	Description
`routingRules`	Specifies the path to the file used by the `addWires` and `repeat&Route` keywords.
`count` *`INT`*	Specifies the number of instances placed in the output file when `repeaterMode=count`. This keyword is required if `repeaterMode=count`.
`xDim` *`REAL`*	Specifies the dimension in millimeters of the x-direction of space to fill when `repeaterMode=size`. This keyword is required if `repeaterMode=size`.
`yDim` *`REAL`*	Specifies the dimension in millimeters of the y-direction of space to fill when `repeaterMode=size`. This keyword is required if `repeaterMode=size`.
`rotate` *`Binary`*	Specifies whether the cell is rotated.
`mirror` *`Binary`*	Specifies whether the cell is mirrored (flipped).
`xOff` *`REAL`*	Specifies offset in the x-direction in DB (database unit), typically 1e-9. Positive numbers create buffer space and negative overlap.
`yOff` *`REAL`*	Specifies offset in the y-direction DB (database unit), typically 1e-9. Positive numbers create buffer space and negative overlap.
`depth` *`INT`*	Specifies depth level (from the top). To add a suffix or remove a prefix at the cell depth, `addSuffix` or `removePrefix` must be `>0`.
`addSuffix` *`Binary`*	Adds a unique suffix to cells, and goes down to the specified hierarchy depth.
`removePrefix` *`Binary`*	Removes prefixes from cell names to merge them, and goes down to the specified hierarchy depth.
`mergeLeafCells` *`Binary`*	Restores all leaf cells to their original names.
`retain` *`cell-list`*	Specifies a comma-separated list of cells to forcibly remove all prefixes and suffixes.
`retainSubCells` *`Binary`*	Specifies that when set to `1`, all child cells from the `retain` options are added to the retain list. This command-line option retains names that are not under the original instance.
`rename` *`cell-list`*	Specifies a comma-separated list of cells to forcibly remove or add a suffix.

Table 15 DCV Test Case Compiler Configuration Files (Continued)

Keyword	Description
`prefix` *`string`*	Specifies the prefix to use (appended with #_).
`suffix` *`string`*	Specifies the suffix to use (appended with #).
`fillSpace` *`Binary`*	When specifying multiple cells, abuts as many small cells in the same orientation to fill the footprint of the largest specified cell, but might still have gaps depending on size ratios.
`cleanup` *`Binary`*	Cleans up unused cells.
`overwriteOriginal` *`Binary`*	Overwrites output file, if it exists (copies to original file; otherwise, any original files will be overwritten).
`outputTop` *`string`*	Specifies the output file name. If omitted, the output file name is *outputTop*.oasis.
`outputFile` *`string`*	Specifies the output file name. If omitted, the output file name is *outputTop*.oasis.
`terminateTimeMinutes` *`REAL`*	Terminates the `dcv_tcc` run after the designated minutes have expired if `>0`.
`terminateMemoryGB` *`REAL`*	Terminates the `dcv_tcc` run if the designated amount of memory in GB is exceeded (no limit if `<=0`).
`terminateMemoryFreeGB` *`REAL`*	Terminates the `dcv_tcc` run if the amount of free memory in GB on the machine drops below the specified amount (no limit if `<=0`).
`terminateMemoryPercent` *`REAL`*	Terminates the `dcv_tcc` run if the run exceeds the percentage of total memory on the machine from which it is running (no limit if `<=0`).
`terminateDirSizeGB` *`REAL`*	Terminates the `dcv_tcc` run if the directory size in GB changes by more than the specified amount (no limit if `<=0`). **Note:** The `terminate*` options are not required, but exist to monitor the job since it is potentially easy to set options along with input data that might exceed the available hardware limitations.

Iterative Execution

To create a more complex example with deeper hierarchy from simple cells, do the following:

1. Type: `dvc_tcc -g`

   Generates an example configuration file

2. Edit the configuration file

   Point to the input GDSII file and specify the cells to be replicated

   Change any other options as needed

   Specify values for outputTop and outputFile (use these names in subsequent runs, as this will be easier than using the automatically generated names)

3. `Type: dcv_tcc` *`edited_config_file`*

   Creates the new data

4. Create a new configuration file

   Repeat steps 1 and 2 or copy the file from step 3 and edit

   Include outputTop in the repeatCells list and outputFile in the GDSII/OASIS inputLayout list (you can include other files as well)

5. `Type: dcv_tcc` *`new_edited_config_file`*

   Creates a larger version of the data from step 3 with a deeper hierarchy

6. Repeat steps 4 and 5 as often as necessary depending on the overall target size and depth of the required hierarchy in the final layout

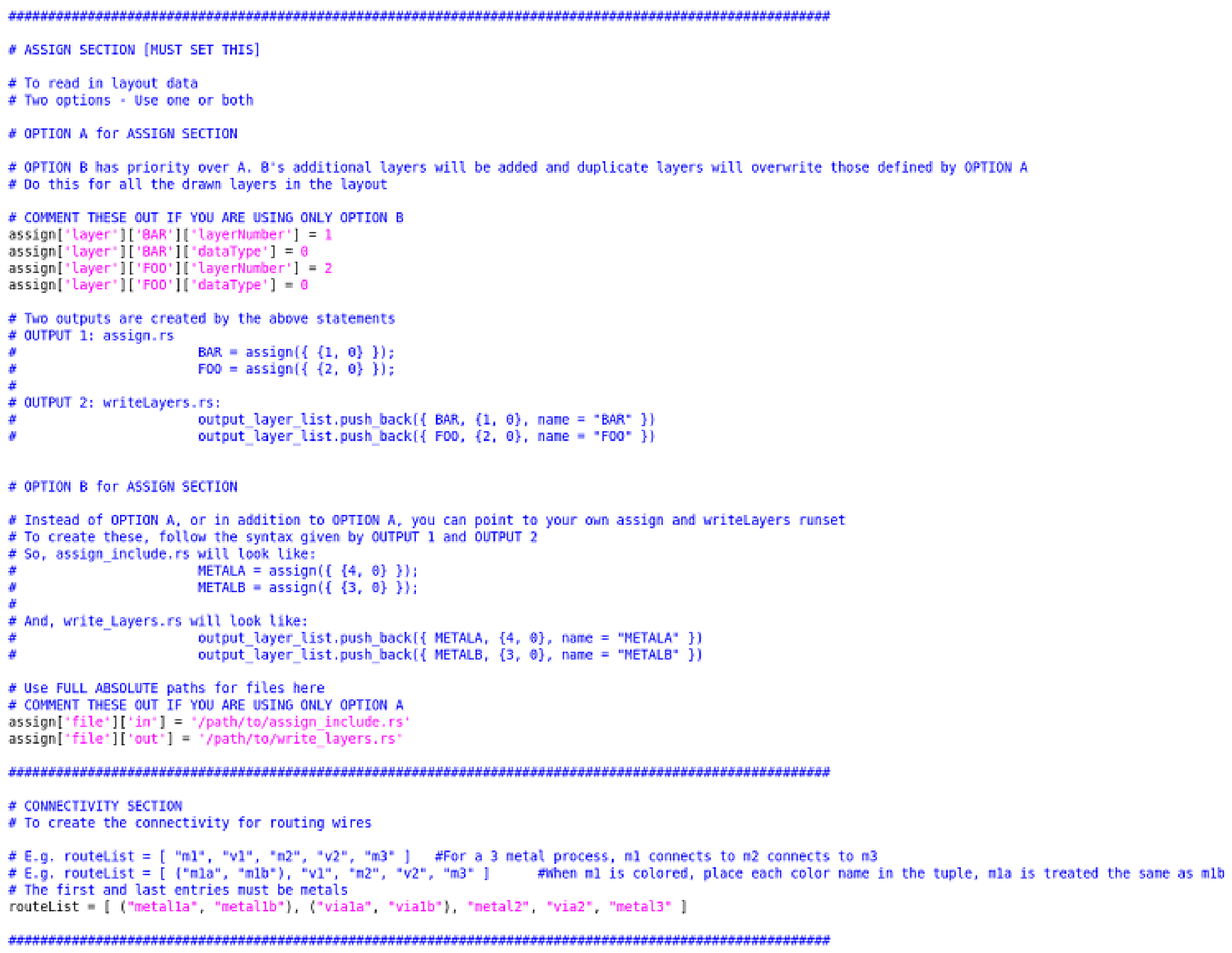
Router Capabilities

The DCV Test Case Compiler routing functionality allows you to define a set of rules to add routing layers to preexisting data. Data can be created by growing vertical stack and filling in the gaps between existing metals based on routing configurations.

The `-pyRule` command-line option invokes the IC Validator tool to create routing layers in a rules.py file.

The rules.py example file is commented with help text to properly fill out. The IC Validator tool creates the layers a runset uses to populate this file. The member functions in the runset are described in the following sections, but basic assign information must be given to the runset. The rules.py file uses information stored in python dictionaries to create a PXL runset. No knowledge of python is required to use this tool, however the file is interpreted by the tool so that you might use python to populate the data structures, as shown in Figure 25.

Figure 25 rules.py File



```
#########################################################################################################

# ASSIGN SECTION [MUST SET THIS]

# To read in layout data
# Two options - Use one or both

# OPTION A for ASSIGN SECTION

# OPTION B has priority over A. B's additional layers will be added and duplicate layers will overwrite those defined by OPTION A
# Do this for all the drawn layers in the layout

# COMMENT THESE OUT IF YOU ARE USING ONLY OPTION B
assign['layer']['BAR']['layerNumber'] = 1
assign['layer']['BAR']['dataType'] = 0
assign['layer']['FOO']['layerNumber'] = 2
assign['layer']['FOO']['dataType'] = 0

# Two outputs are created by the above statements
# OUTPUT 1: assign.rs
#                       BAR = assign({ {1, 0} });
#                       FOO = assign({ {2, 0} });
#
# OUTPUT 2: writeLayers.rs:
#                       output_layer_list.push_back({ BAR, {1, 0}, name = "BAR" })
#                       output_layer_list.push_back({ FOO, {2, 0}, name = "FOO" })


# OPTION B for ASSIGN SECTION

# Instead of OPTION A, or in addition to OPTION A, you can point to your own assign and writeLayers runset
# To create these, follow the syntax given by OUTPUT 1 and OUTPUT 2
# So, assign_include.rs will look like:
#                       METALA = assign({ {4, 0} });
#                       METALB = assign({ {3, 0} });
#
# And, write_Layers.rs will look like:
#                       output_layer_list.push_back({ METALA, {4, 0}, name = "METALA" })
#                       output_layer_list.push_back({ METALB, {3, 0}, name = "METALB" })

# Use FULL ABSOLUTE paths for files here
# COMMENT THESE OUT IF YOU ARE USING ONLY OPTION A
assign['file']['in'] = '/path/to/assign_include.rs'
assign['file']['out'] = '/path/to/write_layers.rs'

#########################################################################################################

# CONNECTIVITY SECTION
# To create the connectivity for routing wires

# E.g. routeList = [ "m1", "v1", "m2", "v2", "m3" ]   #For a 3 metal process, m1 connects to m2 connects to m3
# E.g. routeList = [ ("m1a", "m1b"), "v1", "m2", "v2", "m3" ]      #When m1 is colored, place each color name in the tuple, m1a is treated the same as m1b
# The first and last entries must be metals
routeList = [ ("metal1a", "metal1b"), ("via1a", "via1b"), "metal2", "via2", "metal3" ]

#########################################################################################################
```

When filling out the rules dictionary, note that strings must be quoted numbers must not be. it is recommended to be in this format:

```
rule[LAYER][FUNCTION][INDEX][PARAMETER] = VALUE

WHERE:
   LAYER = 'layer_name' or ('layer_color1', 'layer_color2')
   FUNCTION = 'function_name'
   INDEX = integer
   PARAMETER = 'function_parameter'
   VALUE = 'string' or number
```

The router capability functions are described in these sections:

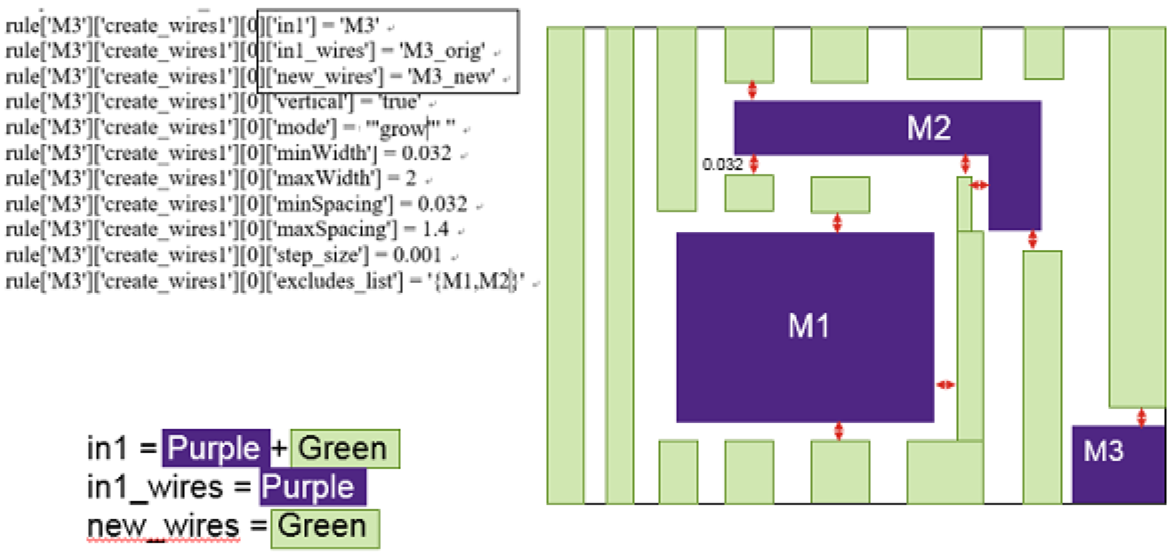
- `create_wire1()`
- `create_wires2()`
- `insert_vias()`
- `insert_vias2()`
- `remove_branch()`
- `remove_branch2()`
- `fill_notch()`
- `fill_gap()`
- `remove_vias_near_plate()`

---

create_wire1()

The `create_wire1()` function creates wires for single color layers between the existing metal spaces based on specified rules.

Figure 26 create_wire1() Function



```
rule['M3']['create_wires1'][0]['in1'] = 'M3'
rule['M3']['create_wires1'][0]['in1_wires'] = 'M3_orig'
rule['M3']['create_wires1'][0]['new_wires'] = 'M3_new'
rule['M3']['create_wires1'][0]['vertical'] = 'true'
rule['M3']['create_wires1'][0]['mode'] = '"grow"" "
rule['M3']['create_wires1'][0]['minWidth'] = 0.032
rule['M3']['create_wires1'][0]['maxWidth'] = 2
rule['M3']['create_wires1'][0]['minSpacing'] = 0.032
rule['M3']['create_wires1'][0]['maxSpacing'] = 1.4
rule['M3']['create_wires1'][0]['step_size'] = 0.001
rule['M3']['create_wires1'][0]['excludes_list'] = '{M1,M2}'
```

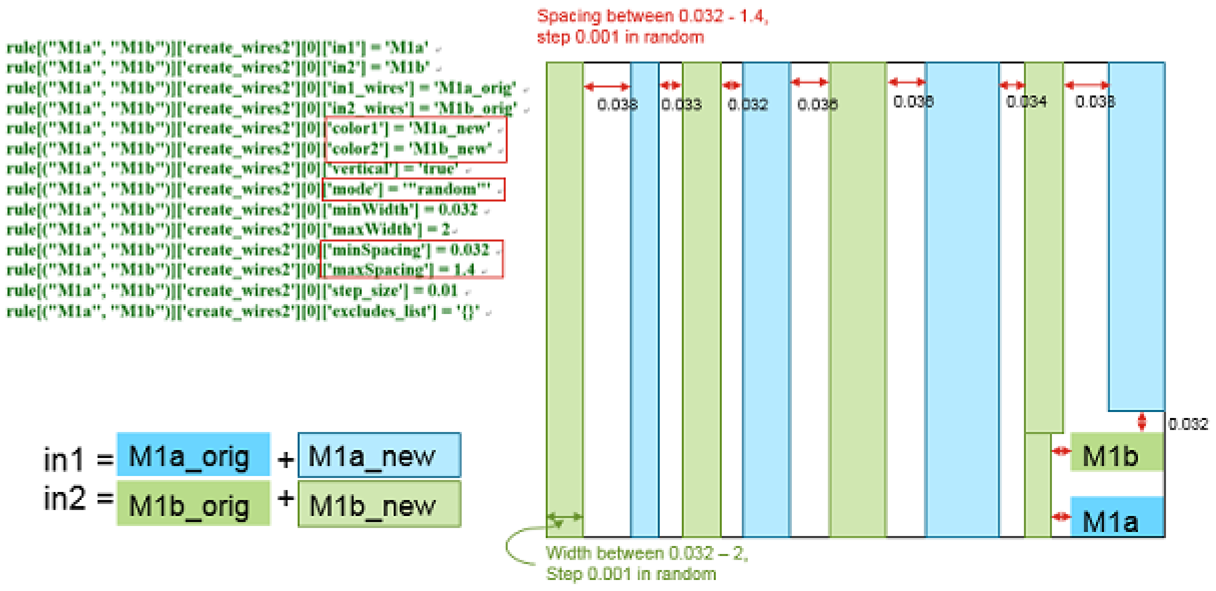
in1 = Purple + Green
in1_wires = Purple
new_wires = Green

Feedback

create_wires2()

The `create_wires2()` function creates wires for dual color layers between the existing metal spaces based on specified rules.

Figure 27   create_wires2() Function

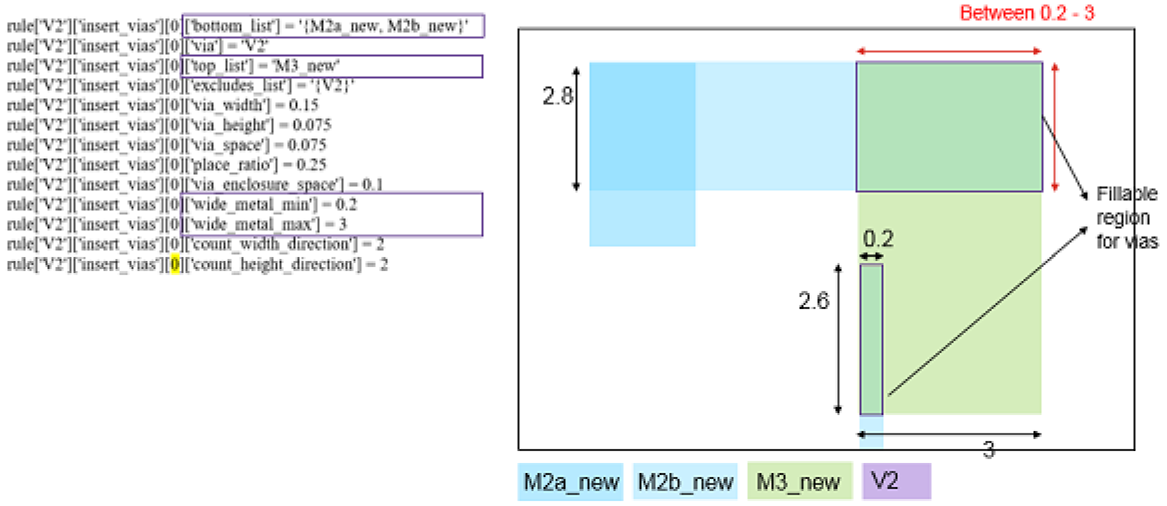


```
rule[("M1a", "M1b")]['create_wires2'][0]['in1'] = 'M1a'
rule[("M1a", "M1b")]['create_wires2'][0]['in2'] = 'M1b'
rule[("M1a", "M1b")]['create_wires2'][0]['in1_wires'] = 'M1a_orig'
rule[("M1a", "M1b")]['create_wires2'][0]['in2_wires'] = 'M1b_orig'
rule[("M1a", "M1b")]['create_wires2'][0]['color1'] = 'M1a_new'
rule[("M1a", "M1b")]['create_wires2'][0]['color2'] = 'M1b_new'
rule[("M1a", "M1b")]['create_wires2'][0]['vertical'] = 'true'
rule[("M1a", "M1b")]['create_wires2'][0]['mode'] = '"random"'
rule[("M1a", "M1b")]['create_wires2'][0]['minWidth'] = 0.032
rule[("M1a", "M1b")]['create_wires2'][0]['maxWidth'] = 2
rule[("M1a", "M1b")]['create_wires2'][0]['minSpacing'] = 0.032
rule[("M1a", "M1b")]['create_wires2'][0]['maxSpacing'] = 1.4
rule[("M1a", "M1b")]['create_wires2'][0]['step_size'] = 0.01
rule[("M1a", "M1b")]['create_wires2'][0]['excludes_list'] = '{}'
```

insert_vias()

The `insert_vias()` function inserts single color vias between the specified layers (can be _new or _orig).

Figure 28 insert_vias() Function



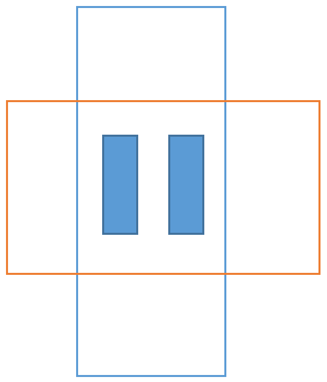
```
rule['V2']['insert_vias'][0]['bottom_list'] = '{M2a_new, M2b_new}'
rule['V2']['insert_vias'][0]['via'] = 'V2'
rule['V2']['insert_vias'][0]['top_list'] = 'M3_new'
rule['V2']['insert_vias'][0]['excludes_list'] = '{V2}'
rule['V2']['insert_vias'][0]['via_width'] = 0.15
rule['V2']['insert_vias'][0]['via_height'] = 0.075
rule['V2']['insert_vias'][0]['via_space'] = 0.075
rule['V2']['insert_vias'][0]['place_ratio'] = 0.25
rule['V2']['insert_vias'][0]['via_enclosure_space'] = 0.1
rule['V2']['insert_vias'][0]['wide_metal_min'] = 0.2
rule['V2']['insert_vias'][0]['wide_metal_max'] = 3
rule['V2']['insert_vias'][0]['count_width_direction'] = 2
rule['V2']['insert_vias'][0]['count_height_direction'] = 2
```

The `via_height` and `via_width` parameters are relative to the orientation of checking for the insertion of a minimum-sized via area with the `count_width_direction` and `count_height_direction` parameters.

Figure 29 shows a minimum 1x2 array. In this figure, the width direction is the x-axis, so it will place an array where you flip either the `via_width`/`via_height` or the `count_width_direction`/`count_height_direction` parameters, and where no vias will be placed. If you flip both numbers, you will get the same result except that the height direction (for this placement) is now on the x-axis.

Figure 29 insert_vias() Function Example 2

```
rule['layer']['insert_vias'][0]['via_width'] = 1
rule['layer']['insert_vias'][0]['via_height'] = 2
rule['layer']['insert_vias'][0]['count_width_direction'] = 2
rule['layer']['insert_vias'][0]['count_height_direction'] = 1
```

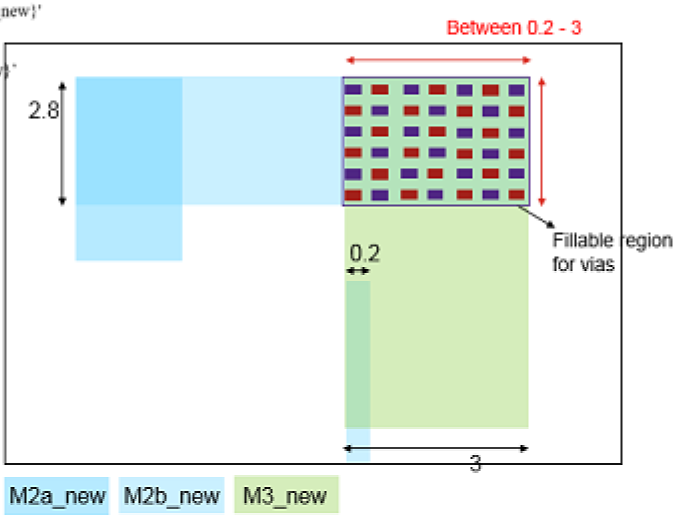


insert_vias2()

The `insert_vias2()` function inserts dual color vias between the specified layers (can be _new or _orig).

Figure 30 insert_vias2() Function

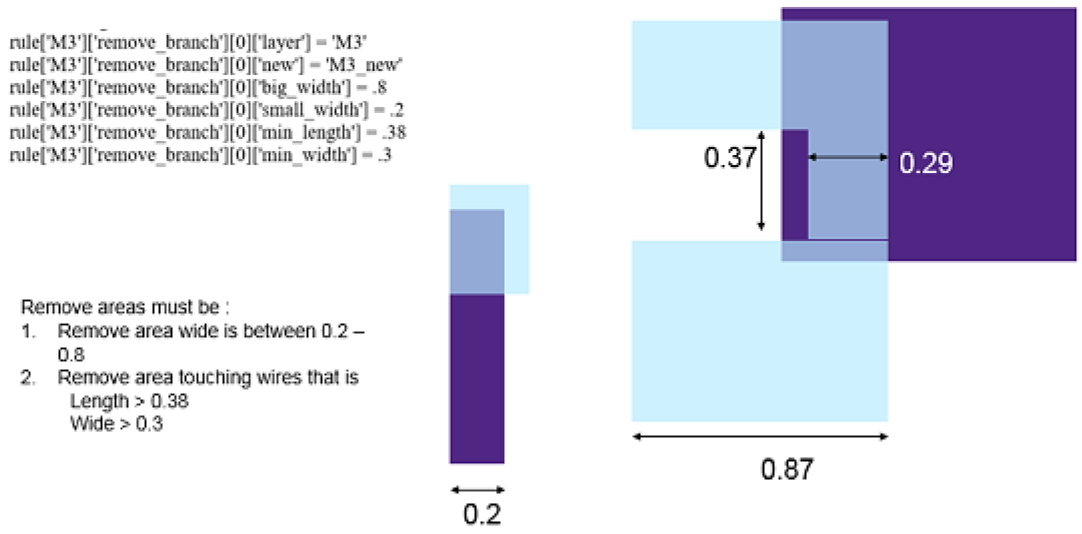
```
rule[("V1a", "V1b")]['insert_vias2'][0]['bottom_list'] = '{M1a_new, M1b_new}'
rule[("V1a", "V1b")]['insert_vias2'][0]['color1'] = 'V1a'
rule[("V1a", "V1b")]['insert_vias2'][0]['color2'] = 'V1b'
rule[("V1a", "V1b")]['insert_vias2'][0]['top_list'] = '{M2a_new, M2b_new}'
rule[("V1a", "V1b")]['insert_vias2'][0]['excludes_list'] = '{}'
rule[("V1a", "V1b")]['insert_vias2'][0]['via_width'] = 0.075
rule[("V1a", "V1b")]['insert_vias2'][0]['via_height'] = 0.075
rule[("V1a", "V1b")]['insert_vias2'][0]['via_space'] = 0.075
rule[("V1a", "V1b")]['insert_vias2'][0]['place_ratio'] = 0.5
rule[("V1a", "V1b")]['insert_vias2'][0]['via_enclosure_space'] = 0.001
```



remove_branch()

The `remove_branch()` function removes branch metals of a certain size that are connected to other metals of a certain size.

Figure 31 remove_branch() Function



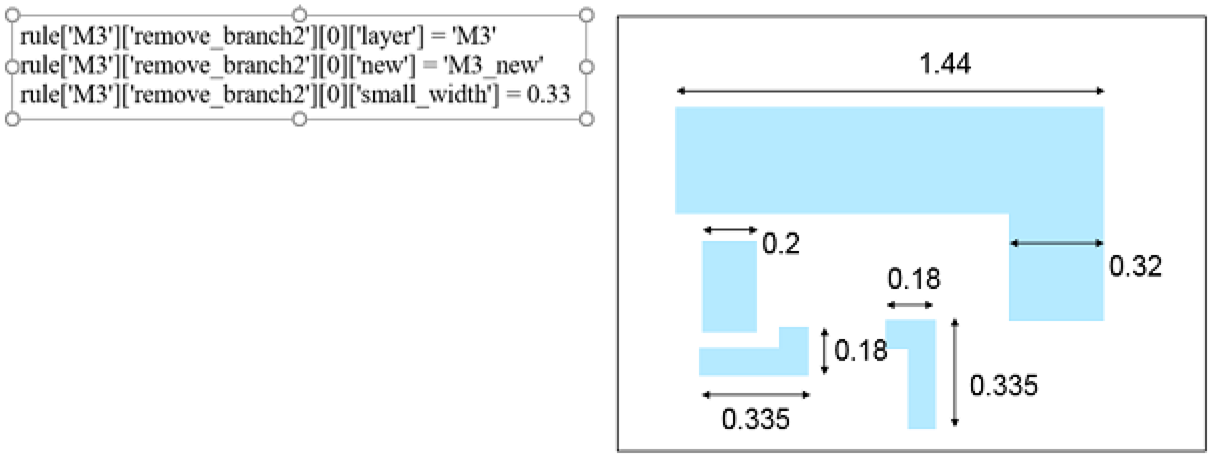
```
rule['M3']['remove_branch'][0]['layer'] = 'M3'
rule['M3']['remove_branch'][0]['new'] = 'M3_new'
rule['M3']['remove_branch'][0]['big_width'] = .8
rule['M3']['remove_branch'][0]['small_width'] = .2
rule['M3']['remove_branch'][0]['min_length'] = .38
rule['M3']['remove_branch'][0]['min_width'] = .3
```

Remove areas must be :
1. Remove area wide is between 0.2 – 0.8
2. Remove area touching wires that is
   Length > 0.38
   Wide > 0.3

remove_branch2()

The `remove_branch2()` function removes branch metals that are below the minimum feature size.

Figure 32 remove_branch2() Function

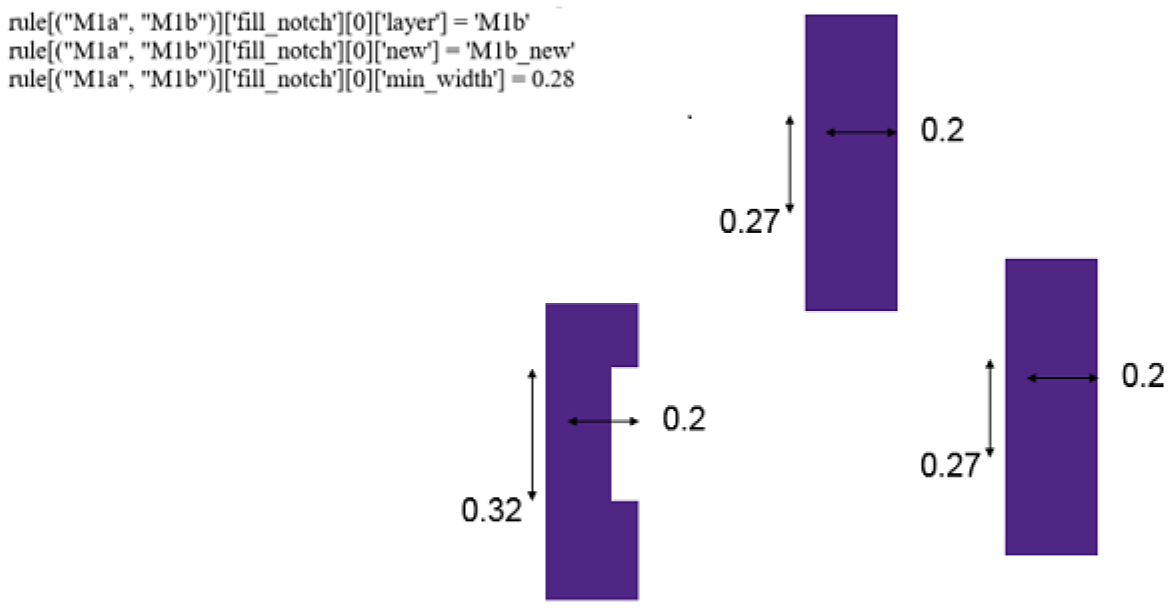


```
rule['M3']['remove_branch2'][0]['layer'] = 'M3'
rule['M3']['remove_branch2'][0]['new'] = 'M3_new'
rule['M3']['remove_branch2'][0]['small_width'] = 0.33
```

fill_notch()

The `fill_notch()` function fills in corners of metals if both corners are below the specified feature size.

Figure 33 fill_notch() Function



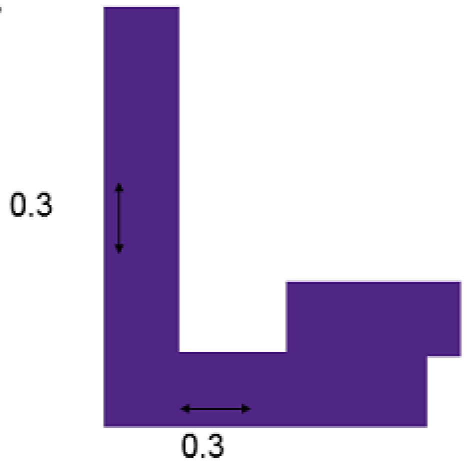
```
rule[("M1a", "M1b")]['fill_notch'][0]['layer'] = 'M1b'
rule[("M1a", "M1b")]['fill_notch'][0]['new'] = 'M1b_new'
rule[("M1a", "M1b")]['fill_notch'][0]['min_width'] = 0.28
```

fill_gap()

The `fill_gap()` function fills in gaps of metals that are below a defined feature size.

Figure 34 fill_gap() Function

```
rule[("M1a", "M1b")]['fill_gap'][0]['layer'] = 'M1b'
rule[("M1a", "M1b")]['fill_gap'][0]['new'] = 'M1b_new'
rule[("M1a", "M1b")]['fill_gap'][0]['min_space'] = 0.31
```



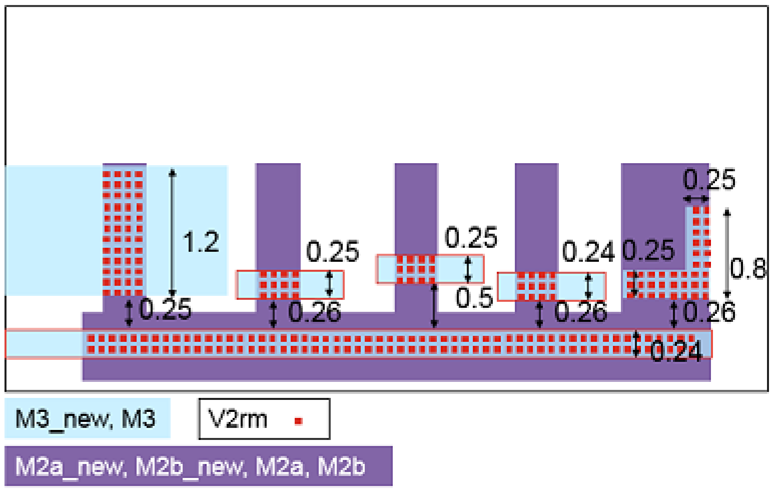
remove_vias_near_plate()

The `remove_vias_near_plate()` function removes vias in a narrow metal that are near a wider metal; the other metal list being the complement to the via of the metal with the “plate” (can be above or below). If necessary, you can clear both calls to this function twice with the switched layers.

Figure 35 remove_vias_near_plate() Function

```
rule['V2']['remove_vias_near_plate'][0]['plate_metal_list'] = '{M3_new, M3}'
rule['V2']['remove_vias_near_plate'][0]['via'] = 'V2rm'
rule['V2']['remove_vias_near_plate'][0]['other_metal_list'] = '{M2a_new, M2b_new, M2a, M2b}'
rule['V2']['remove_vias_near_plate'][0]['plate_width'] = 0.25
rule['V2']['remove_vias_near_plate'][0]['distance'] = .26
rule['V2']['remove_vias_near_plate'][0]['via_width'] = 0.04
rule['V2']['remove_vias_near_plate'][0]['via_height'] = 0.04
rule['V2']['remove_vias_near_plate'][0]['via_side'] = 0.04
```

Removes vias of size 0.04*0.04 on metal (M3_new and M3)below width of 0.25 within distance of 0.26 of metal of a larger size


1.2
0.25
0.25
0.26
0.25
0.5
0.24
0.26
0.25
0.25
0.8
0.26
0.24
M3_new, M3
V2rm
M2a_new, M2b_new, M2a, M2b