White Paper

SYNOPSYS®
Predictable Success

Enhancing the DRC Waiver Methodology for Layout Verification Productivity

May 2009

Author

Jason Puryear
Implementation Group,
Synopsys, Inc.

Executive Summary

AdvancesAs feature sizes shrink in advanced technology nodes, more complex and restrictive design rules are required for manufacturability. Design rules are conservative and companies on the leading edge of technology will knowingly waive design rules in order to get the most out of their designs as possible. As a result, DRC error management has become a larger issue for CAD departments and designers.

To manage design violations, CAD departments have employed a number of solutions to reduce the amount of violations needed to be checked by the physical verification engineer. Non-mask layers are added to the design to cover areas of the chips where known errors may occur, or entire cell based waivers can also be done. However, these solutions are limited in that they are labor intensive, difficult to synchronize with design changes, and waive real errors in certain situations. The need to retain violation classification information for a process, design or cell is required reduce design turnaround time This paper discusses ways to easily manage waiver information and a specific solution that is optimized for this purpose.

Introduction

While the general expectation that a completed design is clean of all design rule check (DRC) violations, there are often cases when such violations will be expected or even desired. Consider the following scenarios:

- Silicon-proven IP may not be redesigned to keep current with changing design rules.
- Providers of cell libraries work with technology providers such as foundries to violate certain design rules in exchange for better performance
- Certain designs can follow a different set of rules not available for general layout such as memory modules. Their inclusion into larger designs may lead to a major DRC violations unless the runset comprehends the inclusion of such designs.
- In new silicon manufacturing technology development, a lead technology vehicle (test chip) is designed to test all possible characteristics of the provided components. A typical test chip leads to thousands of violations that must be carefully analyzed and either is fixed or waived.

Currently these scenarios are typically addressed by a variety of ad hoc solutions. Cell Library or IP providers will provide a list of waivers for the intended DRC violations. These violations need to be carefully tracked for the duration of the cell library as waivers usually apply only to a particular shape position or dimension. New cell versions must be checked against given waivers. Old waivers must be transferred to the new layout and possible layout/waiver inconsistencies must be resolved. Because cell libraries or IP may include many designs, the process of tracking and confirming these waivers can be quite onerous.

For designs which follow a different set of rules from the general layout, the challenge is keeping the runset synchronized with the changes to the layout. A straightforward approach of checking the cells, summarizing the violations and updating runset code requires commitments from both the layout and the runset development teams. This approach necessitates the simultaneous release of both an updated cell library and a corresponding version of the DRC runset.

To address this situation, the runset may include a list of cells to be excluded from a particular check. This requires cell name synchronization and development of a separate runset release to check the excluded cells. An alternative to cell exclusion is for designers to use a drawing layer that would block certain violations. This approach presents other problems:

- Too general of a tool; inclusion or exclusion of a cell violation requires runset update
- Risk that this layer will be drawn in the wrong place;
- Addressing boundary areas where exclusion layer partially overlaps design features

Non-runset approaches usually involve a stand-alone system that allows a designer to export the DRC report then separately track and classify the errors. The downsides are obvious – This approach requires a separate copy of the DRC errors and synchronization between the layout changes and runset; this adds significant time to the design cycle.

During technology development, designers intentionally insert DRC violations to test exactly when the results are undesirable. For example, to ensure that metal lines on a given layer meet minimum spacing rules, a test structure is developed to violate spacing rule and provide an indication when a violation of such a rule leads to undesired behavior, in this case, a short between metal lines. The sheer number of violations makes it difficult to distinguish between intended violations versus actual violations. In addition, a single intended violation can spawn other violations, such as skew in a line width that triggers violations in via enclosure and line-to-line spacing rules. The potential of false violations is high due to the complexity of today's design rules and corresponding runset code.

These scenarios demonstrate the need for tools in error analysis, classification and tracking.

Solving the DRC Classification Dilemma

With the challenges outlined above, the need is imperative for a comprehensive solution to the DRC violation management system. There is a need to manage waivers through the design cycle with minimal impact and without having to modify the design rules or cell lists within the runsets.

The DRC Classification system must have the following properties to be successful:

- It must be integrated directly with the layout editor and DRC viewer. Integration between these systems gives the highest performance to designers; a reported error must be viewed and classified simultaneously from the same interface. This also significantly reduces the possibility of a classification error. A designer must be allowed to classify an error – specify an error type (whether this error is intended or waived), capture reason for the classification, as well as information about the user performing the classification. Multiple users must be able to work simultaneously on the design.
- It must be complete – violations must be distinguished by their type, their location in the design hierarchy and their geometry. Developers must be assured that a waiver for a particular violation such as metal 1 spacing check in cell A found at X,Y coordinates is not falsely applied to other detected violations.
- Error classification must be easily reversible, to deal with cases when the waiver is revoked.
- Error classifications can be tracked. Reasons for classifications, who made the classification, and when the classification is made must be stored and be made available for inspection.

- The flow of DRC classification must be flexible. The system allows for classification information from several sources to be referenced within a run. Each classification source should have control over how violations matched to classification data within it are processed. For example, a particular run on a 32nm design would require the 32nm process classification information found during test-chip analysis, IP classification information provided by the vendor to protect performance and timing, individual design classification information found during previous iterations. The process and IP classification information might be suppressed from all error output to reduce the amount of debugging needed by an end user. The design classification information might not be suppressed, but the matched violations are pre-classified as waived or have extra information attached to aide in fixing the violation if it occurs.
- It must allow several levels of error description. It must permit several types of error classification such as waive, must-fix, watch, etc, and provide the user with a way to reverse the classification of an error.
- Lastly, the classification must be persistent. Once an error has been classified, the classification information needs to be safely stored and automatically retrieved as long as the error appears in the design. The requirement for the error persistence ensures that error classification does not have to be repeated multiple times for the same violation. Once the waiver is specified, it persists across both the layout modifications and application of different DRC checks. Consider two different scenarios. In the first case, a design is completed and, an error waiver is granted and classified. Later on, the design gets changed such that the classified error disappears. At this point, DRC was checked but it was decided to revert back to original design. Once DRC is checked on the design, the DRC classification system must ensure that previously given error classification will still be available. In the second scenario, the same layout goes through a series of different DRC checks such as poly check, metal check, etc, and during each check certain violations are classified and stored. At the final stage, a full check is applied to the design. The DRC error classification system ensures that all previously created classification for different layers are available.

Synopsys' IC Validator Classification Overview

All DRC violations in Synopsys' IC Validator are written to a standard MySQL database (PYDB). The schema includes tables specifically for error classification. These tables may be populated using IC Validator VUE, command line utilities provided with IC Validator, or a user's custom script can be used to access and modify the error database tables directly. Errors are classified with one of the following classes: Error, Watch, Waive, Ignore, and Fixed. Classifications are made on an individual basis or for an entire cell or violation tag. Once classified, the violations can be exported to a special classification error database known as cPYDB. This database may be used as input to subsequent IC Validator runs. Violations found during the next run that match (cell, violation tag, and polygon) those within the cPYDB maintain the classification information.

An IC Validator run may use more than one classification error database as an input allowing for flexibility with classifications from several sources such as cell library groups, process developers, and designers.

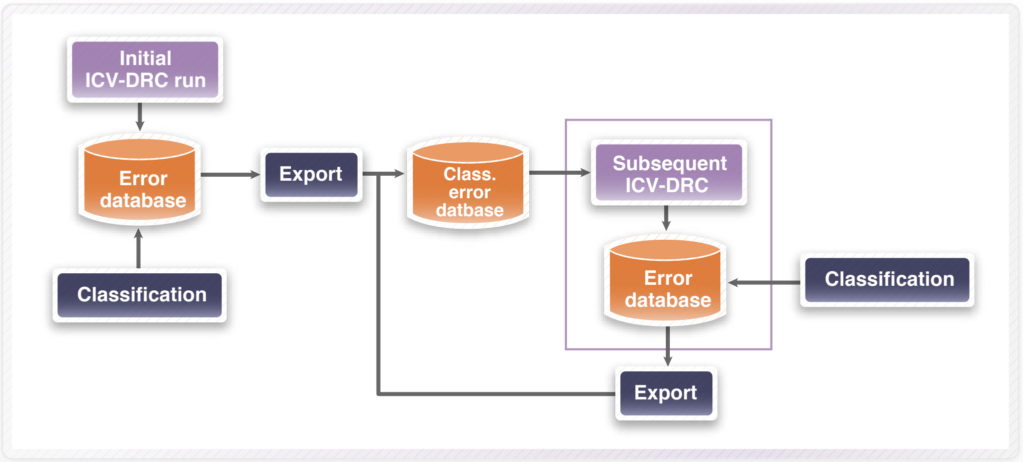

Figure 1: ICV DRC classification flow

The user controls the permissions for each cPYDB referenced within the run, deciding whether matched errors are written to the output PYDB and LAYOUT_ERRORS file, how conflicts are handled, and if the classification data may be amended or overwritten via password protection.

Integrating IC Validator DRC Classification within the Design Flow

IC Validator's DRC classification system has been deployed by two groups at a major semiconductor company. The tool code base was not modified when the engineers integrated the classification system into their existing DRC checking flows. The updated DRC flow provides capabilities that a stand-alone tool does not. Such as:

- Searching for predefined master/project/user level cPYDB and inserting their locations into the combined runset code. Search paths are defined by the project environment. This setup allows consistent application of error filtering and classification for all designs in the project.
- Allowing the selection of error classification to be enabled in selected flows such as general DRC, while restricting its use in other flows
- Checking to see if error classification has occurred and automatically exporting the error classification at the beginning of the DRC flow to prevent an unintentional loss of data
- Retrieving the error output (number of errors, error type, and error classification) from the PYDB with a Perl script after the DRC run for storage in a project tracking database system.

The typical DRC work flow has two stages:

In the first stage, the designer runs DRC on a design. A new PYDB is created that may contain some master process /project level error classification already applied to it; which means that there are fewer numbers of violations being reported requiring fixes to the physical layout. The designer uses IC Validator VUE to view and classify errors as intended or waived. If the design serves as a general-usage cell, the violations will be waived and be filtered out of subsequent error reports. The designer can use the 'Export Classification Information' feature in VUE to create a cPYDB, then configure the DRC flow to use this cPYDB for project-level error filtering. If the cell is to be used only once, no data export is required. After an error classification event, the tool flags the cell PYDB as updated. Figure 2 shows a workflow for this stage.

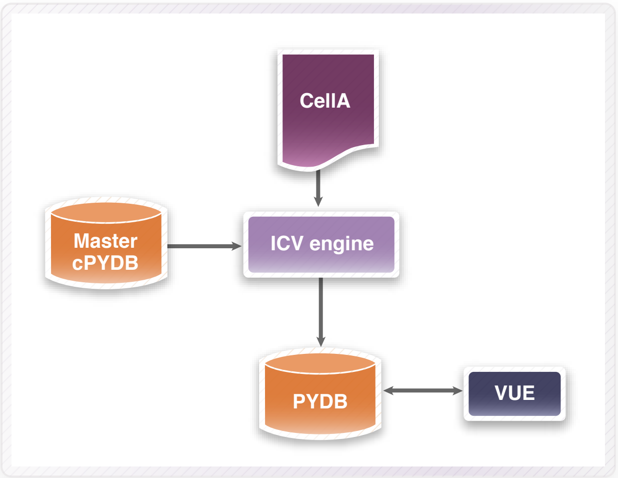

Figure 2: Initial DR check and error classification

For the second stage, once edits to the design are complete, the designer executes another DRC run. At the start, the execution flow checks whether the cell's PYDB contains a "classification updated" flag. If the flag exists, the cPYDB export operation is performed creating a new stand-alone cPYDB unit. An effective runset code is created that now includes the master process and project level cPYDBs and the new created user level cPYDB. IC Validator starts DRC the run and filters the incoming DRC violations according to the user's classification. Project level waived errors are suppressed from all error output to allow the user to focus only on real violations. Upon the completion of the design rule check, the results summary is exported to a format suitable for project tracking. When the designer starts VUE to check results, VUE displays all classification data available from the cPYDB used in the run to display the errors according to the stored classification. Figure 3 illustrates the data flow for this stage.

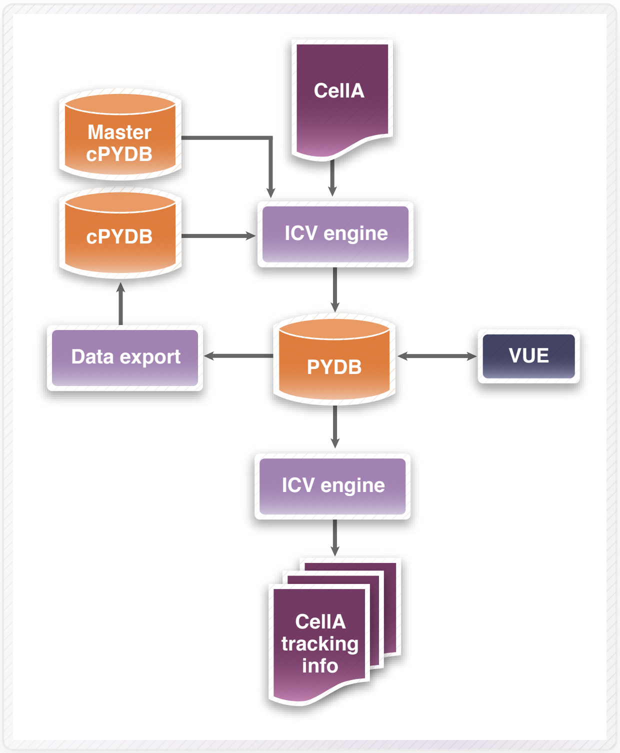

Figure 3: Incremental DRC check, user level error filtering

Details on Process Improvement and Turnaround Time

The application of DRC classification impacts both the storage space required to the host design's PYDB and cPYDB data. It also adds and a small amount of process time to DRC execution for matching the errors as they appear after the runset checks are applied. The data storage requirement is dependent upon how the user processes matched violations. Matched errors which are not written to an output database will reduce the size of the pydb. If matched errors are retained, the output pydb will be slightly larger due to the extra information written. The following table summarizes an impact of DRC classification on a size of error database for a design with large number (~30K) of violations, most of which (~28K) were classified as 'waived' with error explanation ("cPYDB test") added during classification. This situation shows a worst case scenario with large number of errors and their almost complete classification.

The difference in post-classified and post-export PYDBs indicates that user-level DRC classification applies to every unique violation.

	PYDB (MB)	cPYBD (MB)	Difference with initial PYDB size
Pre classification	5.3	0	0
Post classification	5.5	0	3.7%
Export classification	5.5	4.0	79% (PYDB +cPYDB)
Subsequent run	7.5	4.0	116% (PYDB with cPYDB)

Table 1 Impact of DRC classification on data size

For the worst case scenario described above DRC check time increased from 192 to 270 seconds (40% increase) when matching this case. This however is not the typical usage and the increase in runtime averages about 10%.

The runtime performance impact however is more than offset by the increased productivity of designers throughout the design process.

- The time used for analyzing an error occurs only once for a design instead of once for each run.
- Waiver management is streamlined as information is easily obtained from the pydb at any time.
- Errors may be reviewed the second they are written to the database while IC Validator is running. Error Classification may also be done during the run.
- Multiple users may debug the design at the same time using the same error database. Classifications added by one user will be seen by other users the minute they load that cell.

Summary

DRC error management is a major part of the design cycle, from initial technology development to the chip tapeout. Each technology node introduces complex waiver methodologies and the standard management methods do not maintain the tight schedules required for a design project. IC Validator's DRC classification system addresses this problem by tightly integrating an error management system within their error database. This allows designers to perpetually manage errors from initial technology development, project level design management, IP integration, and designer input.

To find out more about Synopsys and IC Validator, visit www.synopsys.com/tools/implementation