DRVerify:
Verifying DRC decks and design rule specifications

Sage Design Automation, Inc.
Santa Clara, California, USA

Abstract

DRVerify is part of the iDRM design rule development platform. DRVerify addresses three problems that don't have an adequate solution today:

***a.** How to verify design rule check (DRC) decks and ensure they correctly, completely and accurately represent the design rule specification*

***b**. How to ensure a complex design rule description is specified correctly and evaluate its accurate meaning*

***c.** How to ensure design rule consistency and avoid potential conflicts between design rules*

Using iDRM's design rule definition pattern and its logical expression, DRVerify systematically generates variations of the design rule patterns and creates a comprehensive set of layout test cases that manifest all meaningful combinations of the design rule parameters that can make it pass or fail.

DRC deck programmers use DRVerify to automatically generate an exhaustive set of pass and fail test cases, and run their DRC code on the generated gds file to check that the code found all fail cases and did not flag any of the passing ones.

Design rule manual (DRM) teams use DRVerify to visualize their design rule expressions and check its boundary conditions to ensure the formal expression accurately reflects their intent. DRVerify will also indicate any possible conflict with other existing design rules.

DRVerify use cases in detail

A. **Verifying DRC Decks (AKA DRC Runsets)**

DRC runset implementation is a task performed by the PDK (Process Design Kit) team. Coding a DRC runset for an advanced process is a long and arduous task, and since the design rule descriptions are often ambiguous, the implementation is based on the runset programmer subjective interpretation. Creating a correct and accurate check of a complex design rule is close to impossible; the probability of making errors is so high and compounded by three levels of built-in ambiguity and inaccuracy: ① The design rule description in the DRM is informal and usually ambiguous to begin with, and therefore is subject to different individual interpretations. ② For any given interpretation of a design rule, it is hard to come up with a check that would cover all possible situations that might violate this rule and at the same time not mistakenly flag close-call legal cases. ③ An advanced process deck is a very complex low-level code with 100,000 lines or more, and therefore easily error-prone.

The inherent and most fundamental challenge is that there is no formal way or methodology to verify the DRC code against the design rule definition or any other golden reference, to ensure its correctness. It is hard to overstate the importance and severity of this problem since the DRC check plays such a critical role in qualifying any design for manufacturing and thus the DRC runset needs to be as clean as possible from bugs, misinterpretations and inaccuracies.

To try and minimize the risk and release higher quality runsets, PDK teams create QA test cases for each design rule. These test cases are layout snippets that manifest both violating (fail) and legal (pass) configurations. The developed DRC code is then run on these layout test cases and the developers check that their code flags an error in each of the "fail" test cases, and that it does not flag any of the "pass" cases.

These test cases are traditionally created either manually or by some automated layout tools and scripts. As for the manual approach: Humans usually have great intuition in creating meaningful test cases, but lack the ability to systematically search a formal rule specification and cover each and every possible combination of a large number of variables. We also tend to make mistakes or miss cases that are outside of our intuition zone. If one needs to create hundreds or thousands of intricate test case combinations of complex logical equations of multiple layers, shapes, polygon edges and parameters - it is very easy to err between passing and failing borderline conditions and it is unlikely to get it all perfectly correct; Scripts and automated layout tools are definitely quicker than manual layout but they are not comprehensive and systematic and are again driven by the user's interpretation of what tests to create. They are in fact a faster version of the manual intuitive method, and as such are neither systematic (covering all possible cases) nor necessarily correct in the interpretation of the design rule description. Furthermore they can create many redundant cases that do not display any new type of error, and yet take up more resources and time to manage and check.

Since all these traditional methods are not automatically derived from the design rule specification, they are subject to human interpretations and misconceptions and are error-prone. These methods can work for design rules that are relatively simple and are already well known from the past, but they don't solve the problem for new design rules and especially the complex ones that are required by advanced technology nodes.

DRVerify addresses these problems by offering a formal and automated methodology. Starting from the source, the design rule specification in iDRM, the DRVerify geometry engine exercises both topological properties of the design rule pattern and the numerical values of its logical expression, and does it in a systematic and comprehensive manner. The result is an exhaustive set of layout test cases that manifest all meaningful combinations of the design rule parameters that can make it pass or fail. Each test case is tagged and annotated with the specific combination of parameters that make it pass or fail. The entire collection of test cases is then written out to a gds file. In that file the collection is divided into two separate groups: all the violating test cases are grouped together and placed under a horizontal dividing line, and all the legal test cases are grouped together and placed above the line, to clearly distinguish between them.

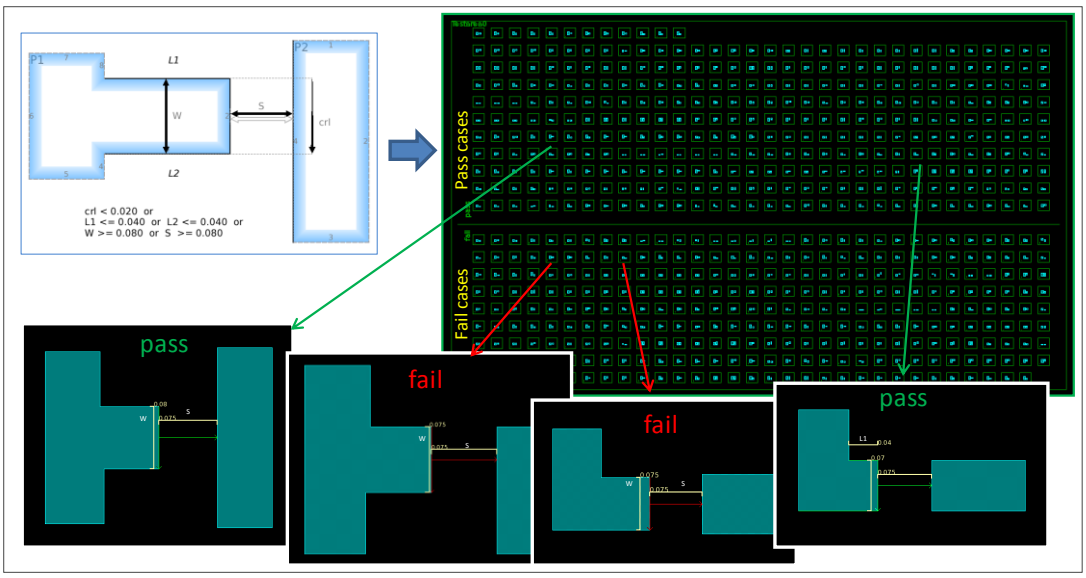

Fig 1. Clear and formal design rule description in iDRM generates hundreds of DRC test cases

The DRC deck under test is then run on this gds file. A correct DRC deck must flag as errors all the test cases in the fail group below the dividing line, and should not flag any of the passing test cases above the line. Any other result indicates a discrepancy between the deck and the design rule specification. The specific cases in question need to be further evaluated and the DRC code be corrected. The specific value annotations generated by DRVerify for each test case pinpoint the specific discrepancy details and help the user to debug the DRC code and correct it.

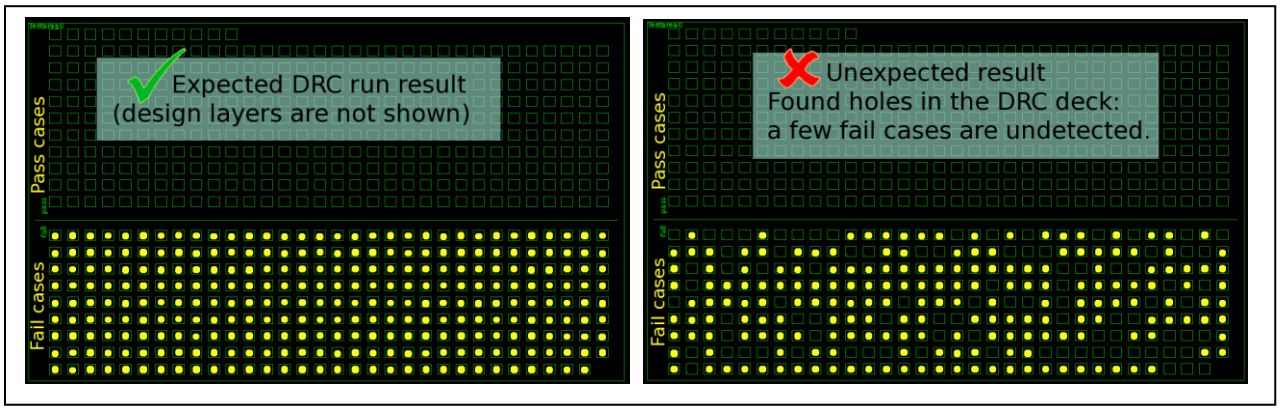

Fig2. Verifying DRC deck using the generated test cases (only error marker layer shown)

B. Design Rule Manuals

B1. Validating Design Rule Descriptions: The problems actually start with the design rule description as specified in the DRM (Design Rule Manual). Design rule descriptions are written today in various formats using drawings, tables and free form text. In advanced technologies (20nm and below) design rules have become so complex that describing them can occupy full pages of diagrams, tables and textual

expressions. Since the description is not formal, there is no methodology today to verify that such a description is a complete and accurate representation of the design rule intent.

To address this problem, DRVerify uses the iDRM formal design rule description and automatically creates layout embodiment snippets of how this rule can manifest itself given the specified drawing and logical conditions. In this use case, DRVerify acts like an animator that acts out the written spec and creates a set of design rule instances that visualize the design rule expression and highlight the possible corner cases and boundary conditions that can flip each test case from pass to fail and vice versa.

This function is very useful and valuable, as both the authors and consumers of the design rule description can immediately see what the design rule spec actually means, what cases are passing and what are failing and where the possible hidden 'gotcha' situations are.

By quickly viewing the generated test cases, the user can immediately spot unexpected aberrations, such as a layout configuration that clearly violates the rule and yet is placed in the "passing" set, or conversely a legal layout instance that based on some missing or overly restrictive conditions was placed in the "failing" set.

B2. Design Rules Consistency: Another tough problem with developing a DRM is inter-rule consistency: making sure that design rules do not conflict with each other. Advanced technology DRMs hold thousands of design rules, some of them very complex. Many layout configurations are subject to multiple design rules which can be overlapping, meaning that certain objects or distances in the layout need to satisfy different conditions that are specified in multiple design rules. When capturing a new design rule one needs to ensure that no part of it is in conflict with parts of other design rules. This is almost humanly impossible to do, and such conflicts may therefore be found only later on, when the DRM is already used by designers.

DRVerify addresses this problem as well. When a design rule is being exercised by DRVerify, the tool keeps all the other background design rules where it is relevant. The user can select which background rules to keep and which not to keep. When DRVerify generates all its different variations for the rule under development, it may violate it to intentionally create "fail" cases for the specific rule, but the selected background rules will not be violated. If a specific test case requires setting values that conflict with any of the background rules, such instance will not be made and DRVerify will issue a warning indicating such conflict. The user can then review the conflicting conditions between the rules and determine how to resolve it. Note that conflicting conditions are not necessarily the result of two or more different design rules; a single complex design rule can have an inherent inconsistency within itself. DRVerify will of course detect such conflicts as well and will issue the proper warnings.

Conclusion

Today there is little automation or tools to assist the work of design rule development, design rule documentation and design rule check development teams. The huge number of design rules and their complexity present a very difficult challenge for these teams. Ambiguity, incompleteness and errors in both design rule descriptions and DRC checks are therefore impossible to avoid. The consequences of

finding them later on by designers are costly and painful, as these issues begin to impact a wide community of design rule consumers and an entire design enablement eco-system.

DRVerify is part of the iDRM design rule compiler platform. iDRM uses a formal and clear design rule description as a single source for everything: the design rule specification, the checker and the test structures. With iDRM a design rule is defined only once, and all its other facets and derivative manifestations are generated automatically, correct by construction and are always in sync.

DRVerify enables the development of higher quality DRC decks, much faster and with less effort. In addition it can also be used to develop clear, unambiguous and consistent design rule specifications.

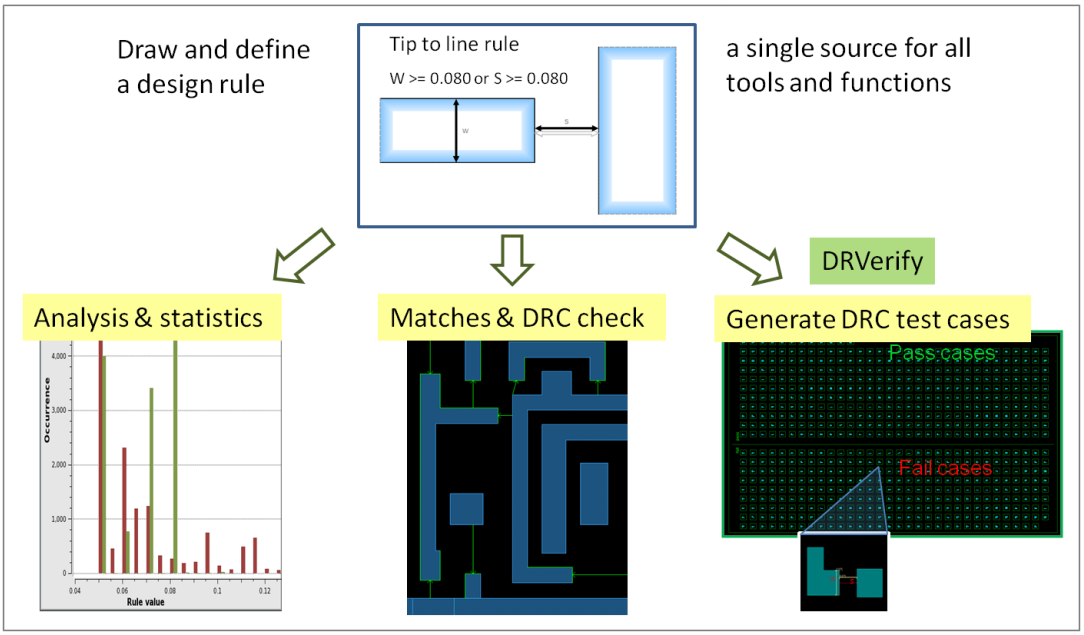

Fig 3: DRVerify is part of the iDRM Platform