Template Based Automatic Generation of Runsets

Elena V. Ravve
Software Engineering Department
Ort Braude College
Karmiel, Israel
Email: cselena@braude.ac.il

***Abstract*—Layout of modern electronic devices consists of billions of polygons for chemical layers. There exist hundreds of design rules, defining how the polygons may be drowning. Design rule checkers (DRC) guarantee that the chip may be manufactured. Moreover, any manufacturing process allows a finite set of supported legal devices. Layout versus schematic (LVS) comparison determines one-to-one equivalency between a circuit schematic and its layout. The correctness of the DRC and LVS runsets are verified using test cases, which contain shapes, representing failing and passing conditions. Creation and maintenance of the complete set of runsets and the corresponding test cases is complicated and time consuming process that should be automatized. Usually almost all design rules may be divided into a set of categories: width, space/distance, enclosure, extension, coverage, etc. Moreover, the set of legal devices for any process may be divided into a set of technology independent categories: transistors, capacitors, resistors, diodes and so on. In this paper, we use these categories in order to define re-usable patterns. The integrator will use the pre-defined patterns in order to compose the design rule manuscript (DRM) rather than to write it. DRC and LVS runsets are then automatically generated using the DRM. Moreover, we use the patterns in order to automatically create the corresponding test cases.**

Keywords–Design Rule Manuscript; Design Rule Checker Runset; Layout versus Schematic Runset; Test Cases; Templates; Automatic Generation.

I. INTRODUCTION

Typical layout of modern electronic devices consists of billions of polygons for different chemical layers. For each such a layer, there exist dozens of design rules (DRs), which define how the polygons may be drowning. Any semiconductor manufacturing process/technology contains a set of physical DRs for geometrical configuration of available layers, wiring, placement and so on.

DRs are series of quantitative limitations, provided by semiconductor manufacturers that enable the designer to verify the correctness of a mask set. DRs have become increasingly more complex with each subsequent generation of semiconductor process. Every chip, which is expected to be manufactured in the given technology, must satisfy the limitations of the DRs. Design rule checking (DRC) runsets are provided by the manufacturer in order to guarantee that the given chip does not give the DR violations.

The document that contains all these rules: Design Rule Manuscript (DRM) has dozens of tables for each layer with free style description of the limitations. The fact leads to various problems, starting from inconsistency in the understanding of the meaning of the rules and going to lots of bugs in coding of the rules in DRC as well as poority of test cases in verification of the DRC runsets. On the other hand, according to our experience of the common work with TowerJazz foundry, usually almost all the DRs may be divided into a relatively small set of categories and sub-categories, such as width, space/distance, enclosure, extension, coverage, etc. In this paper, we use these categories in order to derive a set of patterns. These patterns are the basis of an environment that allows the integrator, who writes the DRM, to use the predefined patterns in order to compose the DRM rather than to write it. DRC runset is then automatically generated, based on the instantiations of the patterns in the DRM.

DRC runsets are provided in order to guarantee that the given chip does not give the design rule violations. The correctness and completeness of the DRC runsets are verified using test cases, which contain shapes of different chemical layers, representing various failing and passing conditions for each rule of the technology.

Creation, modification and maintenance of the complete set of test cases is complicated and time consuming process that should be automatized. Now, we enrich the derived set of patterns, used for DRC runset generation, by the option to create a set of test cases, which corresponds to the pass condition or to failures of the DRs. When the option of failures or passing is chosen, the particular type of the failure or of the passing is defined as well as the form of the report. In addition, particular subsets of the test cases, generated by the given pattern, may be chosen by the user, etc.

The set of the varied parameters for the test cases generator may be extended upon request. When all parameters are defined, the set of test cases would be created automatically. The complete set of the parametrized patterns may be (but not necessary) organized as a library. For any design rule for a given technology, one chooses the relevant parametrized pattern or set of patterns, provides the specific values of required parameters, and puts the obtained instances into the set of test cases, which corresponds to the technology.

The instantiation and(or) modification process may be automated as well. Using such a method, the complete set of test cases for the full set of DRs for the given technology may be created and easily maintained and(or) modified.

Any semiconductor manufacturing process allows a finite set of legal devices, supported and recognizable in the process. Layout versus schematic (LVS) comparison runsets determine one-to-one equivalency between an integrated circuit schematic and an integrated circuit layout. The correctness and completeness of the LVS runsets are verified using test cases, which contain shapes (with connectivity) representing failing and passing conditions for each legal device of the technology.

In this paper, we briefly explain how our general approach may be extended to the case of automatic generation of LVS

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-513-5

runsets and sets of test cases in order to verify them. The proposed innovation is based on the fact that again the set of legal devices for any process or technology may be divided into final set of technology independent categories and sub-categories such that transistors, capacitors, resistors, diodes and so on.

The environment that partially implements the approach is provided. We restricted ourselves to the case of automatic generation of a DRM and a DRC runset, which define and verify limitations, related to width of different layers, as well as the automatic generation of the corresponding set of test cases. The complete tool would produce automatically the DRM, the DRC/LVS runsets and the testcases to test them in a uniform way for all layers and legal devices.

There are several benefits of the presented invention:

- Common methodological basis for different processes, technologies and verification tools;
- Formal approach to DRM composition that allows precise and consistent formulation of physical design rules and description of legal devices for different processes, technologies and verification tools;
- Human independent accumulation of knowledge;
- Significant reduction of human factor and manual writing;
- Total elimination of manual coding and re-use of patterns;
- Better quality and confidence level of the delivered DRM, DRC/LVS runsets and test cases;
- Significant reduction of time and effort to implement DRM, DRC/LVS runsets and test cases;
- Full coverage of all physical design rules and legal devices and the corresponding test cases;
- Integrator does not learn any new programming language;
- Effective, consistent and safe way to change, update and maintain DRM and the corresponding DRC/LVS runsets as well as test cases for all verification tools;
- Detection and correction of mistakes and bug at earliest stages of the flow;
- Effective, consistent and safe way of bug fixes.

The paper is structured in the following way. In Section II, we consider the previous results in the field under investigation. Section III is central in our paper and describes our general approach to solve the problem. In Section IV, we describe in great details a particular implementation of our general approach for creation of a DRC runset for verification of width related DRs. In Section V, we provide the implementation details. Method of automatic generation of test cases for verifying DRC/LVS runsets, using process independent pre-defined generic set of parametrized patterns is described in Section VI. Section VII summarizes the paper.

II. REVIEW OF PREVIOUS WORKS

These exist a lot of attempts to improve the process of creation of DRC and LVS runsets. They start at least from [1], where a process flow representation was proposed in order to create a single, unified wafer processing representation in order to facilitate the integration of design and manufacturing. Hardware assisted DRC was considered in [2][3][4] but quickly returned back to software based solutions [5]. There exist a lot of patents, which attack the same problem. We take the description of the most relevant patents almost verbatim.

In [6], a method for generating test patterns for testing digital electronic circuits, is defined. It fully specifies some primary inputs, while other primary inputs are specified in accordance with selected series of codes. The test pattern template is then repeatedly converted into a stimulus pattern, using different integers in the selected series of codes, and fault simulation is performed on a circuit under test using each stimulus pattern. A stimulus pattern is then saved for subsequent testing of the circuit under test whenever fault simulation using that stimulus pattern shows that fault coverage has increased.

Another close approach was proposed in [7], which considers automatic generation of DRC runsets, using templates per verification tools. The main idea of the invention is that instead of a user creating runsets in a language of a specific verification tool (also called ”native language”), the user expresses the DRC rules in a high level programming language (also called ”meta language”) that is independent of the native language. The meta language includes, in addition to normal constructs of the high level programming language, a set of keywords that identify DRC rules from an abstract viewpoint, unrelated to any native language.

In [8], an approach to deal with programming language, such as C, C++, Perl or Tcl was proposed. In addition, DRC templates of the type described herein capture the expertise of the template author for use by numerous novice users who do not need to learn the native language of a verification tool. In our approach, we eliminate the need to use any (either experienced or novice) user/programmer in order to write the DRC/LVS runsets. In order to reach the target, we propose to force the DRM composer (who is assumed to remain in the game in any case) to instantiate the relevant pre-defined generic patterns rather than to write the DRM as a free-style document. When these patterns are instantiated and the relevant information is extracted and stored in the suitable way, we use the patterns for DRC runsets generation and similar (new proposed) patterns for LVS runsets generation for any particular verification tool.

In [9], use of patterns for improving design checking was proposed but in another context. Moreover, one aspect of the present invention includes a method for generating functional testcases for multiple boolean algorithms from a single generic testcase template. The method includes the preliminary step of creating a generic testcase template containing user-entered mask levels shapes and grouping the shapes within each mask level of the template. Next, testcase generation code comprising mask build language is developed to copy and rename the mask levels from the template into the desired input levels necessary to test a mask build operation. Finally, testcase generation code is executed to generate a testcase. The testcase generation code can be easily modified as necessary to change the mask levels. Additionally, shape interactions for new mask level builds can be added into the generic testcase template, allowing the patterns to be reused to generate additional testcases, see also [10].

A more general approach to use patterns was proposed in [11]. During the design of semiconductor products which incorporates a user specification and an application set, the application set being a partially manufactured semiconductor platform and its resources, a template engine is disclosed which uses a simplified computer language having a character whereby data used in commands identified by the character need only be input once, either by a user or by files, and that data, after it has been verified to be correct, is automatically allocated to one or more templates used to generate shells for the specification of a final semiconductor product. Data must be correct and compatible with other data before it can be used within the template engine and the generated shells; indeed the template engine cooperates with a plurality of rules and directives to verify the correctness of the data. The template engine may generate one or more of the following shells: an RTL shell, a documentation shell, a timing analysis shell, a synthesis shell, a manufacturing test shell, and/or a floorplan shell.

In [12], an automatic LVS rule file generation apparatus, which includes a definition file generating unit and a rule file generating unit, was proposed. The definition file generating unit generates definition files used for a layout verification based on first data and templates that are used for the layout verification in a layout design of a semiconductor apparatus. The rule file generating unit automatically generates a LVS rule file based on the definition rule files. The templates includes first parameters indicating three-dimensional structures of the semiconductor apparatus. The definition files includes second data with respect to the first parameters. However, unlike our approach, a template for an automatic LVS rule file generation is used for generating a LVS rule file that indicates a rule for a layout verification of a layout design.

In [13], a method for comprehensively verifying design rule checking runsets was proposed. It seems to be the most relevant patent to our test cases generation approach. The patent describes a system and method for automatically creating testcases for design rule checking, which comprises first creating a table with a design rule number, a description, and the values from a design rule manual. Next, any design specific options are derived that affect the flow of the design rule checking, including back end of the line stack options. Then, the design rule values and any design specific options are extracted into testcases. Next, the testcases are organized such that there is one library with a plurality of root cells, further comprising one root cell for checking all rules pertaining to the front end of the line, and another root cell for checking design specific options including back end of the line stack options. Finally, the DRC runset is run against the testcases to determine if the DRC runset provides for design rule checking. However, while the patent deals with the general flow of testcase creation for a particular technology, we propose a general method for instantiations of technology independent generic patterns.

In [14], a system and method for automatically creating testcases for design rule checking was proposed. The method first creates a table with a design rule number, a description, and the values from a design rule manual. The design rule values and any design specific options are extracted into testcases. Finally, the DRC runset is run against the testcases to determine if the DRC runset provides for design rule checking. Other methods for verifying design rule checking were proposed in particular in [15] and [16].

One more techniques for verifying error detection of a design rule checking runset was introduced in [16]. Another method for verifying design rule checking software was proposed in [15]. One more technique for verifying error detection of a design rule checking runset was introduced in [16]. However, all the mentioned methods and approaches do not reach our level of generality. Moreover, they do not use sets of pre-defined patterns in the consistent way.

III. A SYSTEMATIC APPROACH TO AUTOMATIC GENERATION OF DRC AND LVS RUNSETS AND THE CORRESPONDING TEST CASES

A DR set specifies certain geometric and connectivity restrictions to ensure sufficient margins to account for variability in semiconductor manufacturing processes. DRC is a major step during physical verification signoff on the design. Each process allows a finite list of legal devices, which may be used and recognizable in the process. LVS comparison runsets determine one-to-one equivalency between an integrated circuit schematic and an integrated circuit layout. DRM may contain hundreds of physical design rules and definitions of dozens legal devices.

Like each physical DR must be implemented in DRC runsets, each legal device must be recognized by LVS runsets. Wafer foundry must provide customers with DRC and LVS runsets, implemented in all required verification tools and languages. Creation, modification and maintenance of the complete set of DRC and LVS runsets is a complicated and time consuming process that should be automatized.

The proposed approach is based on the fact that the set of physical design rules for any process or technology usually may be divided into a final set of technology independent categories such that width, space, enclosure and so on. Moreover, the set of legal devices for any process or technology may be divided into a final set of technology independent categories such that transistors, capacitors, resistors, diodes and so on.

We create one set of parametrized patterns for DRC purposes, such that one pattern (or rather sub-set of patterns) corresponds to a DRC category. In addition, we create another set of parametrized patterns for LVS purposes, such that one pattern (or rather sub-set of patterns) corresponds to a LVS category. The parameters of the patterns may contain in particular (but not limited to) involved layout layers, specific design values, connectivity, additional constrains, etc. The set of parameters may be enriched upon request. While earlier proposed methods involve the patterns in pretty late stages of the runsets generation, we propose to force the DRM composer to fulfill the templates, defined by the patterns, (in any relevant way, for example, using GUI) instead of free-style writing of the document.

For any design rule or legal device for a given technology, the DRM composer chooses the relevant parametrized pattern or set of patterns, provides the specific values of required parameters or (preferably) chooses them from a choice list. The obtained information is transformed and stored as a data structure that will be used for different purposes, such that automatic generation of DRM itself as well as DRC and LVS runsets in particular verification tools and so on. All devices

of the process are put in the list of legal devices with their description in DRM.

Moreover, any verification tool uses different commands, key words and options for features. When free style is used for DRM writing, different interpretations and further implementations of sentences are allowed that may lead to unexpected results in runs of DRC/LVS runsets. In addition, when different formulations are used for definitions of derived layers as well as special options, hardly detectable effects in DRC/LVS runsets may be produced.

The following flow is proposed. We start first from precise definitions of all derived layers or options, which are expected to be used in physical design rules or descriptions of legal devices. The definitions lead to a final fixed set of key words, which are allowed in physical DRs or descriptions of legal devices. The set contains, for example, entries for definition of such notions as $GATE$, $HOT\ NWELL$, $NTAP$, $BUTTED\ DIFFUSION$ and so on, as well as information, extracted from the technology file, such that names and purposes of layout layers, value of grid and so on. In addition, the set contains key words to choose between minimal, maximal, exact options for the values and so on. The set of key words may be divided into sub-sets, such that only values from a particular sub-set are allowed in certain fields of certain templates.

When the set of allowed key words is fixed and stored as the relevant data structure, the DRM composer may pass to the stage of filling the fields in the pre-defined set of templates for physical design rules or descriptions of legal devices. Any field that is aimed to contain a value from the (sub-)set of key words, either is checked on-the-fly for its correctness or is presented as a choice list.

Only fields for the numerical values (for example, the particular value of the width) will not be so. Moreover, many other checking procedures may be involved at this step. For example, check precision on the given numeric values against grid, etc. The precise information, obtained as the result of the filling of the templates is stored as a relevant data structure and will be re-used for particular patterns for further generation of DRM as well as DRC and LVS runsets, implemented in particular languages or tools. Moreover, the information will be used also for the automatic generation of the corresponding test cases for the DRC and LVS runsets.

IV. From rule definition to DRM and DRC

In fact, typically, every DR is constructed from the rule number, the rule parameter such as width, space, overlap, etc., and the layer name, followed by the description of the rule. The last thing is the size, which is typically maximum or minimum. Moreover, a rule may be exclusive for a specific voltage, devices, combination of layers or purpose.

In order to demonstrate how our general approach works, we decided to treat all width rules of a particular existing DRM of Tower-Jazz foundry. We collected all the width rules for all the layers. Then we transformed every rule to a set of short expressions. We proved that an integrator, who writes DRM, may compose any width rule as detailed as she/he wants by composing the expressions without having to add anything manually.

We had a lot of meetings with the target audience of the tool that implements our approach: the integrators. We wanted to understand what is the best way to build the user interface and where we may encounter difficulties. After summing these meetings, we understood that our Achilles heel of the traditionally used practice is the inconsistency of the rule writing. In fact, adding a rule without considering the previous rules or the way that they were written may cause inconsistency in DRM. Moreover, they mostly used to patch new phrase to the old rule, which describes the new need, without changing all the rule from scratch. In order to overcome the obstacle, we analyzed every width rule and divided it into its components in one long table, taking in account what is the purpose of each one as well as what are the corresponding constraints.

For example: if we use nMOS transistors and the gate layer with 3.3V voltage then we approve one value of minimum width. Unlikely, if we use pMOS transistors for the same gate layer with 5V voltage then we approve an absolutely different value of minimum width. We concluded with the help of Tower-Jazz's experts that we may map all the additions to the width DRs into six main categories:

1) Rules for special layers like marking layers;
2) Rules for layer under other layers;
3) Device dependent rules;
4) Voltage dependent rules;
5) Area despondent rules; For example, two layers are used to define thick gate oxide 5V for mask generation and device recognition. **AREA2** defines area with thick oxide either 3.3V. **AREA6** marks thick oxide as 5V for DRC, LVS and MDP purposes.
6) Purpose dependent rules.

In order to translate all these short sentences into one rule, we got help from Mentor Graphics experts with profound knowledge how *Calibre* works. For example: The most comment and basic example to write a width rule will be coded as follows:

XP.W.1 {

@XP.W.1: XP width, min. $\mathbf{0.XX}$ (XP.W.1)
internal XP$<$ 0.8 region singular abut$>$ 0 $<$ 90

}

Let us consider a more complicated example. A metal width rule **MI.W.2** for I=2,.,6 is formulated as follows:

MI.W.2

Minimal width of MI line, connected to a *wide* MI.

In order to better understand the above, we look at the rule's layout in Fig. 1. Now, we see that layer M2 is connected to a *wide* M2. The metal is *wide* if it dimensions are equal or bigger then 35um. Note, that the definition unfortunately does not appear at all in the original formulation in the rule and it is expected to be *known* from the *common knowledge* of the integrators' team. However, the narrow metal, according to the DRs, is approved to be minimum 1um. Otherwise, if it is smaller, our runset must report the violation. In this case, these details are hidden in the original formulation of the rule and must be extracted from other sources of knowledge.

The main problem in the maintenance of DRMs and the corresponding runsets is that, as a rule, the well defined, consistent and well supported source of the knowledge does

not exist at all and it is replaced by some common local folklore, transferred verbally in the integrators' community. Our approach starts from precise definitions of all such short-cuts, which are reviewed by the corresponding experts and supported in a uniform way.

In our particular case, we have, for example, **M2NRW** shortcut that actually means in *Calibre* coding:

M2NRW = ((M2MS or (M2slits interact M2MS)) interact M2WIDE) not M2WIDE.

Now, let us code the rule in *Calibre* for M2.

- The first thing, to be written in the runset file, is the rule name, followed by {. In our specific example, it should be:
  M2.W.2 {
  In this way, we know where this rule begins.
- Next, usually, we want to write comments for this rule to make it easier maintained. We start the comment with sign @. That leads us to the next line in the runset:
  @M2.W.2: Width of Narrow Metal, Connecting to Wide Metal min. $0.YY$ (M2.W.2)
- Now we put the body of the rule for constraints, which are interpreted as violations for this specific layer:
  X2=not outside edge M2NRW M2WIDE
  EX2=expand edge X2 by 0.01
  area EX2 $<$ 0.02
- Sign } finishes the composition of the rule, so that we determine where it ends.

As the result of our coding, we receive the following automatically generated piece of the runset:

M2.W.2 {

```
@M2.W.2: Width of Narrow Metal, Connecting to
Wide Metal min. 1 ( M2.W.2 ).
X2=not outside edge M2NRW M2WIDE
EX2=expand edge X2 by 0.01
area EX2 < 0.02
```

}

The considered example represents a single rule of dozens of rules, while each such a rule has dozens of layers. Eventually, each rule must be translated into DRC statements. We have shown how the coding may be automatized.

V. Implementation Details

In this section, we show in great details, how our general approach is implemented in a particular toy-tool. We start from a complete snapshot of the GUI, see Fig. 2, then, we explain each step.

A. Let us start

The user(integrator) is expected to provide her/his password, when starting the tool, see Fig. 2 step 1.

B. What about the process?

Using the tool, the user may add a new process, remove an existing process or use a stored process, see Fig. 2 step 2.

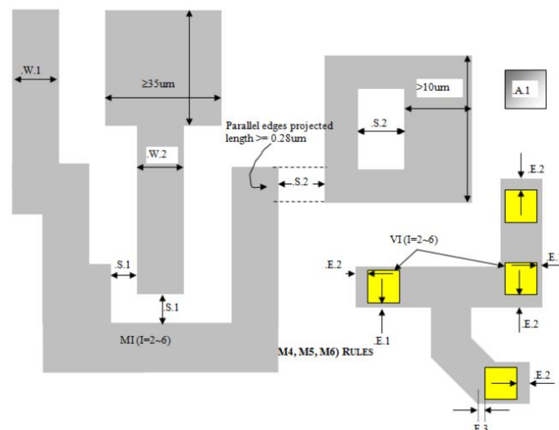

Figure 1. **M2.W.2** rule

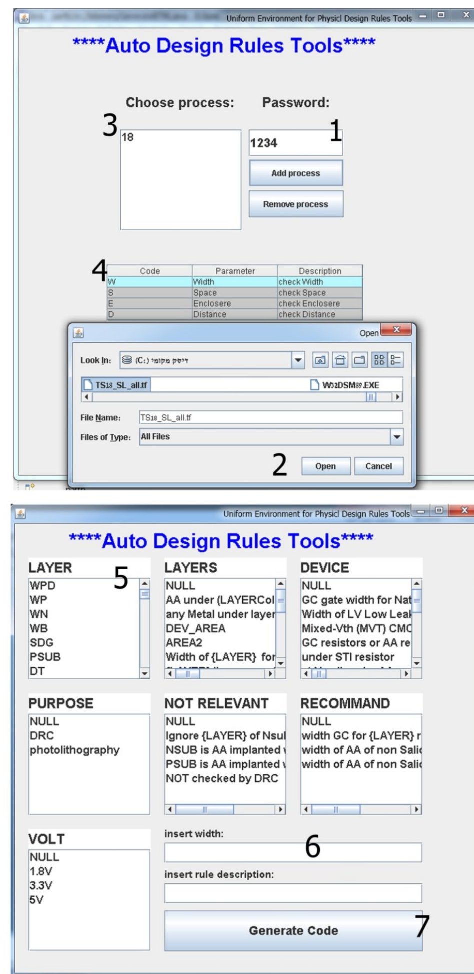

Figure 2. Complete snapshot of the GUI

C. Which layer?

When the process is chosen, see Fig. 2 step 3. The techfile of the chosen process is used in order to access the list of the available layers.

D. Composing a rule

When the layer is chosen, the user gets the list of all available categories of the rule. By double-clicking on the desired category, the user gets all the pre-defined sub-categories, available in order to compose the new rule, see Fig. 2 step 4. The sub-categories include in our particular case (but not limited in the general case to):

- the list of all layers from the techfile as well as special layers, like marking layers;
- the list of purposes and recommended options;
- the list of not relevant cases and available devices;
- the list of voltages.

The user may choose any allowed combination of the sub-categories for the new rule, see Fig. 2 step 5. If some combination of the sub-categories is not allowed then the fact is checked automatically by the tool and the user is updated accordingly.

Now, the user should insert the value of the rule: width in our particular case, as well as a free style comment, see Fig. 2 step 6. These are the only values, which are inserted and not chosen from pre-defined options. Then, the corresponding DR is put to its place in the DRM.

E. Generating the code of the rule

It remains to choose the corresponding tool: Calibre in our example, see Fig. 2 step 7. The corresponding code is generated automatically by the tool.

F. Testing the generated code of the rule

In order to test the generated code, we composed a layout with the corresponding DRC violation. The violation was found and reported by the automatically generated runset.

VI. Method of automatic generation of test cases for verifying DRC/LVS runsets, using process independent pre-defined generic set of parametrized templates

In general, dozens of test cases per a design rule should be provided in order to guarantee correctness and completeness of all DRC runsets implemented in all tools and all languages. Moreover, different test cases should be created for failing and passing conditions per each design rule. In addition, all the test cases must be maintained and modified according to any relevant change in DR. As for now, both code of DRC runsets and test cases are manually created and maintained. All the above justifies that automated methodology and system should be proposed for these tasks.

We propose a new approach to the automated test cases generation for DRC runsets again based on the fact that there exists a finite fixed set of categories, which may be defined at once, such that the categories cover all (or most) design rules for any given process or technology. The set again contains such categories as width rules, spacing rules, enclosure rules etc. Then we propose to re-use the parametrized patterns, defined for DRM generator, for each category such that, the pattern may be calibrated to the particular testing purposes by assignment of the corresponding parameters.

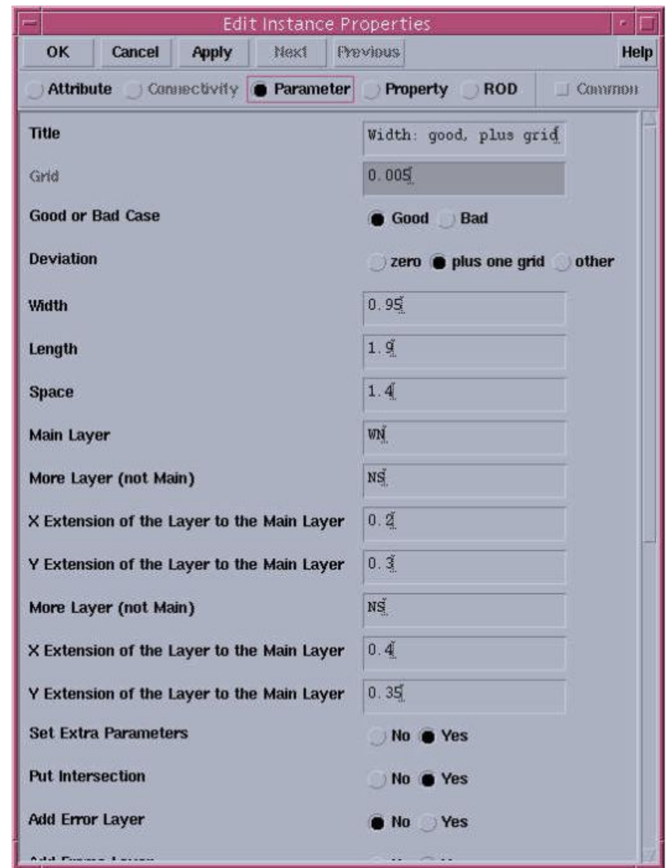

Figure 3. Menu to generate test cases

Fig. 3 illustrates the concept. The example shows some part of the technology parameters such as the layout layers and purposes as they are defined in the technology file, different values taken from DRM, as well as parameters, related to the testing purpose such that the failing or passing case and its particular version.

In addition, the corresponding report format may be defined using, for example, error layers and so on. All the parameters (or any part of them) may be assigned either manually or in some automated way. The assignment procedure leads to creation of a particular instance of the template that corresponds to the chosen pattern, testing purpose, etc.

Fig. 4 illustrates one of the possible implementation of such instantiation. The particular test case generator was written in SKILL and it is included as an integrated part in the proposed tool.

This approach may be extended to the case of automatically created testcases for LVS checking as well. In fact, the list of legal devices of the process as well as their detailed description is available in DRM. DRM may contain dozens of legal devices such that their final list for the process may be combined from different sub-sets, according to additional options or limitations. LVS runsets are implemented, using different tools

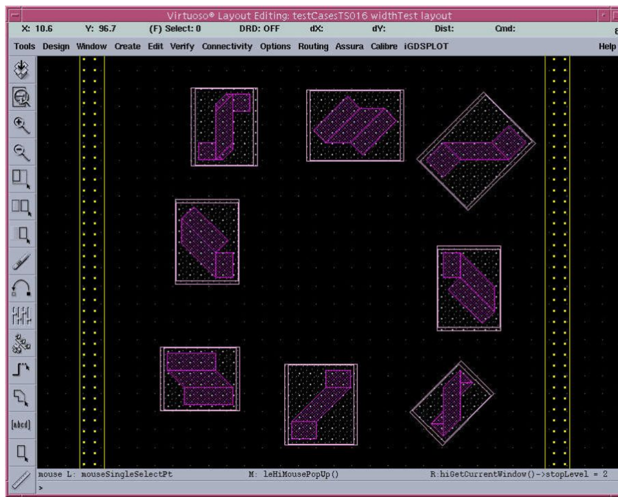

Figure 4. Automatically generated test cases

and program languages, each one with its own algorithms and particular implementations of checking procedures for different features.

Hundreds of test cases per a legal device should be provided in order to guarantee correctness and completeness of all LVS runsets, implemented in all tools and languages. Moreover, different test cases should be created for failing and passing conditions per each legal device and/or their combination. In addition, all the test cases must be maintained and modified according to any relevant change in DRM.

Our method comprises first of all creating of a data structure (say, a table) with a device identifier, its description (including involved layers and connectivity), and the corresponding values from DRM. The data structure contains all legal devices for the process. Any design specific options or limitations, which affect the recognition process, may be added.

Then, the device descriptions and design specific options are implemented into a set of test cases. The implementation is expected to be automatic for both failing and passing conditions. Next, the testcases are organized in a data structure (say, a library) that is suitable for the further run of LVS checkers. Finally, the LVS runset is run against the testcases to determine if the LVS runset is correct and complete. The LVS test case generator is still not included in the implemented tool.

VII. CONCLUSION AND OUTLOOKS

In this paper, we propose a general approach that allows automatized generation of a design rule manuscript, based on a final set of pre-defined patterns. The particular instantiations of the patterns in the DRM generator are then used for automatic generation of DRC and LVS runsets as well as the corresponding test cases.

The approach is based on the fact that usually almost all design rules may be divided into relatively small set of categories: width, space/distance, enclosure, extension, coverage, etc. Moreover, the set of legal devices for any process or technology may be divided into final set of independent categories: transistors, capacitors, resistors, diodes and so on.

The environment that partially implements the approach is provided.

We restricted ourselves to the case of automatic generation of a DRM and a DRC runset, which defines and verifies limitations, related to width of different layers, as well as the automatic generation of the corresponding set of test cases. The complete tool would produce automatically the DRM, the DRC/LVS runsets and the testcases to test them in a uniform way for all layers and all legal devices.

The approach may be extended to automatic generation of other runsets, say, antenna runsets and the corresponding test cases. In general, the approach may be applied in a uniform way to all steps of the of masks' generation and verification.

Acknowledgments

We are would like to thank T. Estrugo (Tower-Jazz) for valuable discussions, general support and his many suggestions. We are would like to thank U. Krispil (Mentor Graphics) for his technical assistance. We also appreciate the effort of our students M. Ankonina and N. Mazuz, who implemented the tool.

REFERENCES

[1] E. Ünver, Implementation of a Design Rule Checker for Silicon Wafer Fabrication, ser. MTL memo. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1994.

[2] L. Seiler, A Hardware Assisted Methodology for VLSI Design Rule Checking, ser. MIT/LCS/TR-. Mass. Inst. of Technology, Laboratory for Computer Science, 1985.

[3] T. Blank, M. Stefik, and W. vanCleemput, "A parallel bit map processor architecture for DA algorithms," in Proceedings of the 18th Design Automation Conference, ser. DAC '81. Piscataway, NJ, USA: IEEE Press, 1981, pp. 837–845.

[4] R. M. Lougheed and D. L. McCubbrey, "The Cytocomputer: A practical pipelined image processor," in ISCA, J. Lenfant, B. R. Borgerson, D. E. Atkins, K. B. Irani, D. Kinniment, and H. Aiso, Eds. ACM, 1980, pp. 271–277.

[5] D. Wittenmyer, Offline Design Rule Checking for VLSI Circuits. University of Toledo., 1992.

[6] K. Bowden, "Method for generating test patterns," Apr. 18 2000, uS Patent 6,052,809.

[7] G. Richardson and D. Rigg, "Method and system for automatic generation of DRC rules with just in time definition of derived layers," Aug. 26 2003, US Patent 6,611,946.

[8] D. Shei and J. Cheng, "Configuration management and automated test system ASIC design software," Dec. 30 1997, US Patent 5,703,788.

[9] S. O'Brien, "Methods and systems for performing design checking using a template," Aug. 4 2009, US Patent 7,571,419.

[10] P. Selvam, "Method for generating integrated functional testcases for multiple boolean algorithms from a single generic testcase template," Feb. 24 2009, US Patent 7,496,876.

[11] T. Youngman and J. Nordman, "Language and templates for use in the design of semiconductor products," Oct. 11 2011, US Patent 8,037,448.

[12] K. Okuaki, "Automatic LVS rule file generation apparatus, template for automatic LVS rule file generation, and method for automatic lvs rule file generation," Oct. 6 2005, US Patent App. 11/093,100.

[13] D. Shei and J. Cheng, "Configuration management and automated test system ASIC design software," Dec. 30 1997, US Patent 5,703,788.

[14] J. Crouse, T. Lowe, L. Miao, J. Montstream, N. Vogl, and C. Wyckoff, "Method for comprehensively verifying design rule checking runsets," May 4 2004, US Patent 6,732,338.

[15] W. DeCamp, L. Earl, J. Minahan, J. Montstream, D. Nickel, J. Oler, and R. Williams, "Method for verifying design rule checking software," May 16 2000, US Patent 6,063,132.

[16] J. Lawrence, "Techniques for verifying error detection of a design rule checking runset," Jul. 23 2009, US Patent App. 12/017,524.