Optimizing Lithography Weakpoints Classification with Pattern Matching Technology

Yongfu Li, I-Lun Tseng, Valerio Perez, Vikas Tripathi, Zhao Chuan Lee, Yoong Seang Jonathan Ong

GLOBALFOUNDRIES
Singapore

https://www.globalfoundries.com

ABSTRACT
Lithography simulation has been the physical verification tool for checking lithography weak-
points, but at the cost of extremely high computational complexity and runtime. To achieve
fast turn-around time for our customers, pattern matching based physical verification has been
generally adopted in place of lithography simulations. This technology has helped overcome
the limitations of DRC in dealing with multi-dimensional constraints and precisely identifies
more localized problematic areas in the layout. The accuracy and runtime heavily depend on
the bounding region of a weakpoint. A smaller bounding region is adopted in practice because
any escapes of real weakpoints can be devastating during manufacturing. However, as a result,
having to fix false violations can increase development time and degrade timing performance
at critical paths. In this paper, we propose a two-level pattern classification methodology to
improve the accuracy of each violation markers. We have characterized a list of “false viola-
tion” and “Known weakpoint” peripheral patterns using lithography simulation. Hence, when
we perform the pattern matching simulation through Synopsys ICV, the enhanced classification
provides more critical information for design fixing or violation waiver.
Table of Contents

1 Introduction ..................................................... 3
  1.1 Problem Statement ........................................... 3
  1.2 Contributions ............................................... 4

2 Architecture and Implementation ................................. 5
  2.1 Pattern Classification (Lithography Simulation) ............ 5
  2.2 Pattern Learning ............................................. 6
  2.3 Pattern Matching ............................................. 8

3 Results and Discussion ........................................... 11
  3.1 User Case I: Unit Testing .................................... 11
  3.2 Pattern Matching Verification ............................... 12
    3.2.1 Pattern Matching using Synopsys ICV command-line Option .... 12
  3.3 Pattern Matching using Synopsys ICV VUE in IC WorkBench ........ 13

4 Conclusions ...................................................... 16

References .......................................................... 16

Table of Figures

Figure 1. Layout images for standard cell A and B with Pattern #1. ............ 4
Figure 2. Our proposed pattern matching workflow with PXL functions. .......... 5
Figure 3. (a) Sample layout for unit testing (b) “central” pattern in the library (c)&(d) hit and false “peripheral” patterns identified through lithography simulation. .... 12
Figure 4. (a) Graphical user interface of the IC WorkBench (b) ICV error result in ICV VUE interface. ..................................................... 14
Figure 5. (a) ICV layout error report (b) Location of error markers in the unit test-case. 15

Table of Tables
1 Introduction

After many years of continuous scaling on the CMOS technology without the use of extreme ultraviolet lithography tool, the limits of the 193i optical lithography tool has caused the printing of smaller geometrical sizes to suffer serious degradations in resolution. The classical rule-based Design-Rule-Check (DRC) approach is no longer sufficient to guarantee 100% pattern printability. Design-for-Manufacturability (DFM) compliance checking is required to identify manufacturing weak-points and to prevent catastrophic errors such as open (necking) and shorts (bridging) issues.

Every foundry uses their in-house proprietary model-based lithography simulation tool to obtain accurate printed chip’s images that detect any pattern-related defects (lithography weak-points). With this great accuracy, it comes at a cost of extremely high computational complexity and runtime. Therefore, chip-level lithography simulations are usually performed only during the tapeout phase. As a result, designers are given only a short period of time to modify the design to eliminate any production failures.

GLOBALFOUNDRIES offers a quick-lithography verification solution, named DRC+, which is based on pattern matching technology [1]. The pattern library is built from the fab-learned critical lithography weakpoint patterns. Then, the entire design layout is scanned, and locations with layout topologies identical to one of the pattern in the library patterns are identified. Thus, the tool detects lithographic weakpoints very effectively and it is scalable to large chip verifications.

Pattern matching is categorized into exact pattern matching and fuzzy pattern matching techniques. In the exact pattern matching, the weakpoint is reported if and only if there is an exact same pattern in the library. Hence, the exact pattern matching is similar to string searching algorithm, which has been highly-optimized for performance. However, the exact pattern matching is weak in recognizing previously unseen hotspot patterns. To overcome part of the problem, fuzzy pattern matching technique is adopted to identify similar weakpoint pattern in the layout.

1.1 Problem Statement

The window size of the pattern affects the accuracy of weakpoint detection and computation runtime. In our previous work, we proposed a methodology to optimize the window size of each pattern [2]. However, the technique is not suitable in our advanced technology due to an increasing adoption of fuzzy pattern matching and smaller window size to identify unseen lithography weakpoints.

For example, as shown in Figure 1, “Pattern #1” appears in both standard cells A and B with differing pattern topologies at the periphery of the clipping window. “Pattern #1” in standard cell A is a lithography weakpoint but the “Pattern #1” in standard cell B is a “false violation”. In this paper, we propose a two-level pattern classification methodology to improve the classification of our fuzzy patterns.
1.2 Contributions

Our contributions are briefly summarized as follows:

1. We propose a robust and efficient two-level hotspot pattern matching tool utilizing both central and peripheral pattern information.

2. We demonstrate our implemented prototype system in Section 2.

3. We provide the relevant scripts for our users to adopt our methodology.

The remainder of this paper is structured as follows. We present our proposed two-level hotspot pattern classification system and the implementation of our enhanced pattern matching system using the Synopsys Programmable, Extensible Language (PXL) in Section 2. In Section 3 we provide our verification results on a design testcase through Synopsys IC Validator (ICV) command-line option and Synopsys IC WorkBench. We conclude our work in Section 4.
2 Architecture and Implementation

In this section, we introduce the concept and implementation of our two-level pattern matching system. We use our existing pattern library to search for matching patterns in a given input design and extract the pattern topologies with a larger window size. Then, we use our model-based full lithography simulation to classify the location of each window clip either as lithography weakpoint or false violation. The original patterns and the new patterns with extended window size are labelled as central and peripheral patterns. The peripheral pattern library is built to separate the “false violation” and “known weakpoint” from all the identified weakpoints. The remaining unclassified error markers will be classified as “new violation”. Such an approach greatly reduces the false positives and therefore reduces the layout area to be examined by designers. With the proposed pattern matching system, we are able to continuously improve its accuracy over time.

The implementation of our enhanced pattern matching prototype system using several C shell scripts, Tcl scripts and Synopsys IC Validator (ICV) runsets, where the ICV runsets are written with the Synopsys Programmable, Extensible Language (PXL) [3, 4]. As shown in Figure 2, our implementation are accomplished through pattern classification (Lithography Simulation), pattern learning, and pattern matching, which will be discussed in details in the following sections.

![Figure 2. Our proposed pattern matching workflow with PXL functions.](image)

2.1 Pattern Classification (Lithography Simulation)

We use our foundries sign-off lithography tool to determine the accuracy of all the error markers reported by our existing pattern matching runset. Locations with lithography weakpoint will be classified as “known violation” while the remaining locations will be classified as “false violation”. We optimize and determine the window size for each violation and clip out the images for pattern learning phase. The optimization process is required to minimize pattern redundancy and improve the runtime efficiency.
2.2 Pattern Learning

As shown in Algorithm 1, we use Synopsys ICV to compile the new pattern layout into our ICV pattern library. We use the Synopsys utility, pdb_utility.pl to convert the pattern library into layout for review. Our Synopsys ICV runset (Algorithm 2) comprises of eight sections, which are summarized as follows:

1. User-defined environment variables
2. Call library() to define the design library used
3. Call error_options()
4. Call pattern_options() to define the pattern library path
5. Layer Assignment
6. Define the library information
7. Pattern Learning
8. Output layout database

where the pattern_options() function defines the path for the pattern library and the pattern_learn() function creates a new pattern library or updates an existing pattern library with source patterns. We have partitioned all the patterns into three different libraries based on the pattern classification result. Our original patterns are compiled into the “CENTRAL” library. The “known violation” and “false violation” patterns are compiled into “PERIPHERAL_HIT” and “PERIPHERAL_FALSE” libraries, respectively.

```
#!/bin/csh -f

setenv PATTERN_LIBRARY "./pattern.gds"
setenv FORMAT GDSII;
setenv PM_LIB_PATH "./pattern_library";
setenv PM_LIB_NAME "PERIPHERAL_HIT";
setenv LEARNING_RUNSET "./pattern_learning.runset";

icv -f $FORMAT \
   -i $PATTERN_LIBRARY \
   -c "." \ 
   -turbo \
   -verbose \ 
   -64 \ 
   $LEARNING_RUNSET $ICV_HOME_DIR/contrib/pdb_utility.pl \
$PM_LIB_PATH \ 
$PM_LIB_NAME \ 
-read ${PM_LIB_PATH}/${PM_LIB_NAME}/central.gds
```

Algorithm 1. Synopsys ICV command-line option for pattern learning
```c
#include <icv.rh>

// Step 1: User-defined environment variables
#pragma envvar default PM_LIB_PATH "./pattern_library"
#pragma envvar default PM_LIB_NAME "MET"

// Step 2: Call library() to define the design library used
library(
    format = GDSII,
    library_name = "TOP.gds",
    cell = "*"
);

// Step 3: Call error option()
error_options (error_limit_per_check = ERROR_LIMIT_UNLIMITED);

// Step 4: Pattern Option
pattern_options (
    pattern_library_path = $PM_LIB_PATH
);

// Step 5: Layer Assignments
metal : polygon_layer = assign( {{1,0}} );
pattern_marker : polygon_layer = assign( {{2,0}} );
pattern_extent : polygon_layer = assign( {{3,0}} );
edge_tolerance_layers : polygon_layer = assign( {{4,0}} );
ignore_region_layers : polygon_layer = assign( {{5,0}} );
pattern_text_id : text_layer = assign_text( {{6,0}}, use_exploded_text={
    cells="*", text = {"*"}});

error_list : list of write_error_map_s = {};

// Step 6: Define the library information
library = pattern_library (library_name = $PM_LIB_NAME,
    library_path = $PM_LIB_PATH);

// Step 7: Pattern Learning
violation @= {
    pattern_learn(
        pattern_library_handle = library,
        pattern_layers = {metal},
        pattern_marker = pattern_marker,
        pattern_text_id = pattern_text_id,
        pattern_extent = pattern_extent,
        ignore_region_layers = ignore_region_layers,
        edge_tolerance_layers = edge_tolerance_layers,
        match_ambit = {0.2,0.2,0.2,0.2},
    )

```

Page 7

Optimizing Lithography Weakpoints Classification with Pattern Matching Technology
ambit_mode = PM_MARKER_CENTER,
pattern_fuzziness = PM_EDGE_UNIFORM,
uniform_fuzzy_size = 0.004,
edge_jog_size = 0,
pattern_reflect = true,
pattern_rotate = true,
ignore_extra_polygons = false,
output_pattern_xml = false,
anchor = FIRST_PATTERN_LAYER,
pattern_type = PM_TWO_DIMENSIONAL,
pattern_naming_mode = PM_NAME_SEQUENTIAL,
pattern_anchor_optimization = PM_ALL
);

// Step 8: Output layout database
error_list.push_back((violation,{0,0}));
output_lib = gds_library("out.gds");
write_gds(
    output_library=output_lib,
    errors=error_list);

Algorithm 2. An example of Synopsys ICV runset for pattern learning

2.3 Pattern Matching

Our proposed Synopsys ICV runset (Algorithm 3) comprises of eight sections, which are summarized as follows:

1. User-defined environment variables
2. Call library() to define the design library used in the pattern matching check
3. Call error_options() for error filtering
4. Call pattern_options() to define the pattern library path
5. Layer Assignment
6. Call pattern_match() to perform pattern matching
7. Perform the boolean between central marker and peripheral marker
8. Output layout database

As described in Algorithm 3 we invoke pattern_match() function multiple times within our runset to capture patterns on a design that match the central and peripheral patterns in our pattern library. Then, we perform layer boolean operations on the marker shapes for pattern classification. In addition, we used error_options() function to classify the error types into “known weakpoint”, “false violation” and “new violation”. We propose to use different pattern
library names to maintain the different class of our peripheral patterns. For physical inspection of the error makers, we invoke gds_library() and write_gds() functions to generate an overlay layout with all these marker shapes.

```c
#include <icv.rh>

// Step 1: User-defined environment variables
#pragma envvar default PM_LIB_PATH "./pattern_library"
#pragma envvar default PM_TOP_MET "10"

// Step 2: Call library() to defines the design library
// used in the pattern matching check.
library(
    format = GDSII,
    library_name = "TOP.gds",
    cell = "*"
);

// Step 3: Call error_option() for error filtering
error_options(
    error_limit_per_check = ERROR_LIMIT_UNLIMITED,
    classify_by_layer = {
        {violation_comments = {"*NEW*"},
         classification = "Watch"},
        {violation_comments = {"*FALSE*"},
         classification = "Ignore"}
    }
);

// Step 4: Call pattern_option() to define the pattern library path
pattern_options(
    pattern_library_path = $PM_LIB_PATH
);

// Step 5: Layer Assignment
MET1 : polygon_layer = assign ( {{15,0}} );
MET2 : polygon_layer = assign ( {{17,0}} );
metal_layer_list : list of polygon_layer = {MET1, MET2};
metal_error_list : list of write_error_map_s = {};

// Step 6: Call pattern_match() to perform pattern matching
for(i = 0 to strtoi($PM_TOP_MET)-1) {
    layer_index : integer = i + 1;
    central_error = pattern_match(
```
pattern_library_name = $PM_CENTRAL_LIB_NAME,
    pattern_layers = {metal_layer_list[i]},
    report_orientations = ALL);
central_layer = marker_merge( layer1 = central_error);
false_error = pattern_match(
    pattern_library_name = $PM_PERIPHERAL_FALSE_LIB_NAME,
    pattern_layers = {metal_layer_list[i]},
    report_orientations = ALL);
false_layer = marker_merge( layer1 = false_error);
hit_error = pattern_match(
    pattern_library_name = $PM_PERIPHERAL_HIT_LIB_NAME,
    pattern_layers = {metal_layer_list[i]},
    report_orientations = ALL);
hit_layer = marker_merge( layer1 = hit_error);
pm_hit_layer = interacting(
    layer1 = central_layer,
    layer2 = hit_layer,
    include_touch = ALL,
    count_parity = ALL,
    count_by = SHAPE,
    name = "known weakpoints");
pm_false_layer = interacting(
    layer1 = central_layer,
    layer2 = false_layer,
    include_touch = ALL,
    count_parity = ALL,
    count_by = SHAPE,
    name = "false violations");
peripheral_layer = or(
    layer1 = hit_layer,
    layer2 = false_layer);
new_violation_layer = not_interacting(
    layer1 = central_layer,
    layer2 = peripheral_layer,
    include_touch = ALL,
    count_parity = ALL,
    count_by = SHAPE,
    name = "new weakpoints");

// Step 7: Perform the boolean between central marker and peripheral marker
pm_hit_violation @= {
  @"METAL"+layer_index+" KNOWN PM RESULT: ";
  copy(
    layer1 = pm_hit_layer,
    name = "known weakpoints");
};
pm_false_violation @= {
  @"METAL"+layer_index+" FALSE PM RESULT: ";
  copy(

Algorithm 3. An example of Synopsys ICV runset for pattern matching

3 Results and Discussion

The experiments were carried out in two parts. In the first experiment, we verify our prototype pattern matching system on our sample design. We also present two methods of executing Synopsys ICV runsets, namely (1) Synopsys ICV command-line option and (2) Synopsys ICV VUE [4]. In the second experiment, we perform pattern matching verifications on several chip-level large designs to help us understand the performance trade-offs for our new prototype system (To be disclosed after final submission).

3.1 User Case I: Unit Testing

Figures 3(a) and (b) illustrate the layout view of our sample design and one of our lithography weakpoint pattern in the library, respectively. In this experiment, we label the lithography weakpoint pattern as “central” pattern. As shown in Figure 3(a), there are a total of three “central” markers highlighted in the layout. Two highlighted layout topologies have previously observed in our past lithography simulations. We have characterized these layout topologies as hit and false “peripheral” patterns, as shown in Figures 3(c) and (d). The “peripheral” patterns are built into our prototype pattern matching software for verification.
Figure 3. (a) Sample layout for unit testing (b) “central” pattern in the library (c) & (d) hit and false “peripheral” patterns identified through lithography simulation.

3.2 Pattern Matching Verification

3.2.1 Pattern Matching using Synopsys ICV command-line Option

Our C shell script used for executing the Synopsys ICV command-line option is shown in Algorithm 4. We used shell environment variables to define the three following pattern library, namely (1) “CENTRAL”, (2) “PERIPHERAL_HIT” and (3) “PERIPHERAL_FALSE”. The result is stored in the run_detail directory.

```bash
#!/bin/csh -f

setenv LAYOUT_PATH "./testcase.gds" ;
setenv FORMAT GDSII;
setenv PM_LIB_PATH "./pattern_library";
setenv PM_CENTRAL_LIB_NAME "CENTRAL";
setenv PM_PERIPHERAL_FALSE_LIB_NAME "PERIPHERAL_FALSE";
setenv PM_PERIPHERAL_HIT_LIB_NAME "PERIPHERAL_HIT";
setenv MATCHING_RUNSET "./pattern_matching.runset";

icv \
  -f $FORMAT \
  -i $LAYOUT_PATH \
  -turbo \
"./pattern_matching.runset";
```

Page 12 Optimizing Lithography Weakpoints Classification with Pattern Matching Technology
3.3 Pattern Matching using Synopsys ICV VUE in IC WorkBench

Synopsys pattern matching can be executed through Synopsys graphical user interface softwares such as IC WorkBench, IC compiler, IC compiler II and custom compiler. In this paper, we demonstrate the pattern matching verification in the IC WorkBench. We need to define the shell environment variable “ICWBEV_USER=IC64” before starting the software. Figure 4(a) illustrates the graphical user interface of the Synopsys IC WorkBench with out unit test-case. The pattern matching runset is loaded into the ICV VUE and the result can be viewed through the interface as shown in Figures 5. From Figure 5(b), we can clearly see that our proposed pattern matching system is able to classify the error into three categories, namely (1) “Error”, (2) “Ignore” and (3) “Watch”. Using the ICV VUE interface, we are able to annotate the errors back to the layout viewer, as shown in Figure 4(b).
Figure 4. (a) Graphical user interface of the IC WorkBench (b) ICV error result in ICV VUE interface.
Figure 5. (a) ICV layout error report (b) Location of error markers in the unit test-case.
4 Conclusions

In this work, we adopt an innovative approach to optimize the pattern matching tool through our proposed two-level hotspot pattern classification flow utilizing both central and peripheral pattern information. We have demonstrated a working model and provided the relevant software code to enhance our users learning. Since Synopsys ICV supports command-line option, we are able to automate the entire process and the system is able to continuously identify new unique patterns. Therefore, the enhanced pattern matching system is able to improve its accuracy over time.

References


