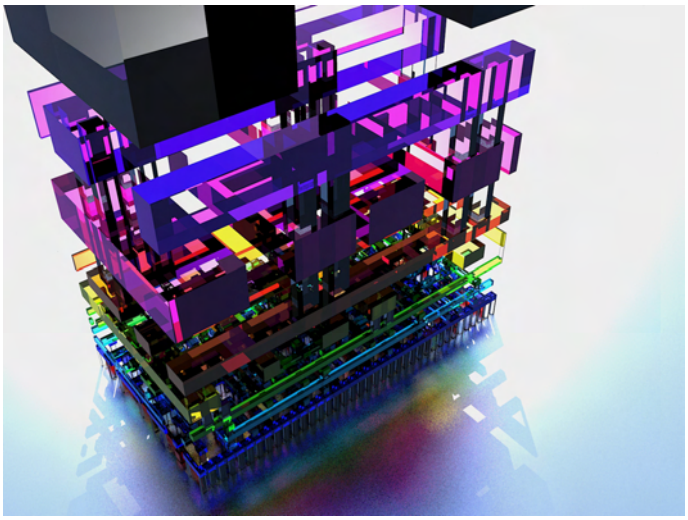
STATISTICAL STATIC TIMING ANALYSIS & PHYSICAL DESIGN

PANAGIOTIS - TAXIARCHIS GIANNAKOU



Master thesis
Electrical & Computer Engineering
ECE
University of Thessaly

November 2015 – version 1.0

[November 16, 2015 at 12:12 – classicthesis version 1.0]

SUPERVISORS:
George Stamoulis
Nestoras Eumorfopoulos
Panagiota Tsompanopoulou

Panagiotis - Taxiarchis Giannakou: *Statistical Static Timing Analysis & Physical Design*, , Master thesis, © November 2015

LOCATION:
Volos, Greece

TIME FRAME:
November 2015

To my family & friends

Dedicated to the loving memory of my father Apostolos Giannakou.
1958 – 2012

ABSTRACT

In modern nano-scale technology, a great deal of effort goes towards the optimization of the digital circuits in order to enhance the performance of modern design. Transistor level optimization has achieved remarkable results over the classic gate-level approach. Optimal transistor/device sizing is a very promising optimization method. However efforts to this end have been hindered by the sheer size of the optimization problem, modeling issues, and the DRC integration in the newly sized cells. In the resizing problem's domain layout resizing has proven to be one of the trickiest parts of the continuous transistor sizing procedure, as it involves layout manipulation which is constrained by a large number of layout design rules that must be taken into consideration and many standard cell libraries are hand-drawn. In this thesis, GDS2trim a layout manipulation tool for continuous transistor sizing is presented along with its related technology and theoretical basis. GDS2trim, by making use of key features such as minimized input/output operations and parallelism, implements an automated layout processing methodology both for design and cell layouts allowing globalized circuit optimization within exceptional performance margins.

Περίληψη

Στην σύγχρωνη τεχνολογία γίνεται μεγάλη προσπάθεια για την βελτιστοποίηση των ψηφιακών κυκλωμάτων προκειμένου να επιτευχθεί περεταίρω βελτίωση της απόδοσής τους. Η βελτιστοποίηση σε επίπεδο transistor έχει επιδείξει αξιόλογα αποτελέσματα σε σχέση με την κλασσική προσέγγιση σε επίπεδο πύλης. Η βελτιστοποίηση μεγεθών transistor είναι μια πολλά υποσχόμενη μέθοδος βελτιστοποίησης. Ωστόσο οι προσπάθειες πρός αυτή τη κατεύθυνση δυχαιρένονται απο το ίδιο το μέγεθος του προβλήματος, θέματα μοντελοποίησης, και θέματα ενσωμάτωσης των κανώνων DRC στα νέα κελιά.
Στο χώρο του προβλήματος κλιμάκωσης, η κλιμάκωση του σχηματκού αποδικνύεται οτι έιναι ένα απο τα πιο απαιτητικά τμήματα της διαδικασίας βελτιστοποίησης, καθώς εμπλέκει την διαχείρηση σχηματικής πληροφορίας, η οποία υπόκειται σε ενα μεγάλο αριθμό περιορισμών κανόνων σχεδίασης οι οποίοι πρέπει να ληφθούν υπ'όψιν και το γεγονός οατι πολλές βιβλιοθήκες τυποποιημένων κελιών δεν είναι αυτόματα παραγμένες, αλλα προϊόν ανθρώπινης σχεδίασης. Στα πλαίσια της παρούσας μεταπτυχιακής εργασίας, περιγράφεται το εργαλείο gds2trim, το οποίο κάνοντας χρήση πολύ συγκεκριμένων χαρακτηριστικών όπως ελαχιστοποιημένη είσοδο/έξοδο και παραλληλοποίηση, υλοποιεί μία πλήρως αυτοματοποιημένη διαδικασία διαχείρησης και επεξεργασίας σχηματικής πληροφορίας, για σχηματικά σχεδίασης και τυποποιημένων κελιών, κάνοντας έτσι δυνατή την ολική βελτιστοποίηση του κυκλώματος μέσα σε εξαιρετικά χρονικά πλαίσια.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— **?** [**?**]

ACKNOWLEDGEMENTS

First of all i would like to sincerely thank my family for the support and love that have shown me during good & rough times as well as my friends that stood for me and supported me.

Additionally, I would like to thank my supervisors George Stamoulis, Nestoras Eumorfopoulos, Panagiota Tsompanopoulou for providing guidance and valuable help throughout my entire course of studies here in the department of Electrical & Computer Engineering, University of Thessaly.

Last but not least, I would like to thank my colleagues in the Nanotrim project, for the exceptional cooperation and the advances we achieved through our work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

GDS Graphic Database system

API Application Programming Interface

PDK Process Design Kit

DRC Design Rule Checking

SREF GDSII Reference element

DEF Design Exchange Format

SPEF Standard Parasitics Exchange Format

1 INTRODUCTION

Abrief Introduction to continuous transistor sizing problem & layout manipulation

Modern integrated circuits are developed with several methodologies that span from fully-custom design techniques to automated standard-cell based design flows.

High-end integrated circuits often utilize fully-customized designs in order to achieve the desired levels of performance. This approach involves custom macro cells, dynamic logic circuits, transistor-level tools and handcrafted layouts. The former allow designers to tune the performance of the design exactly to the target requirements regarding the various performance metrics (area, delay and power). Nevertheless, despite its flexibility and potentials, this methodology requires a great deal of resources and design skills.

However many nanoscale integrated circuits, follow a different, more automated design procedure called standard-cell design methodology. This methodology relies on standard cell libraries, synthesis and place & route tools to implement the target design. Contrary to the aforementioned fully customized approach, standard-cell based design methodology is characterized by significantly shorter turn-around times, allowing designers to meet product goals under increased time-to-market time demands. Moreover designers can focus on the designing process itself in terms of target design definition and description. However, despite the former advantages, the implementation quality is far from optimal and the performance is limited, due to the generalized scope of the automated methods and the fact that this methodology relies on the availability of the standard-cell libraries and the various design tools.

1.0.1 *Standard-Cell based design flow*

A very abstract view of the basic design methodology is presented in Figure 1 and consists of the following steps:

- IC functionality is described in some Hardware Description Language (HDL) like VHDL, Verilog or System-C. In this level of abstraction the functionality of the system is validated either using testbenches (input series followed by the expected output series) or by formal methods. Timing accuracy is cycle accurate at best, and only a vague idea of the final circuit-level timing can be obtained. Power analysis is possible but still quite inaccurate for fine tuning the circuit. Only design choices that involve large swaths of the system can be analyzed for power.

- After HDL coding is completed and its functionality verified, the HDL code is translated into a circuit through:
  - An automated synthesis tool (like Synopsys Design Compiler), which uses predesigned logic gates to implement the logic behavior of the HDL description while satisfying constraints on the timing (and possibly power) of the circuit. The basic logic cells are already implemented both at the circuit level and the layout level and are part of a standard cell library
  - The standard cell library comprises the transistor (in Verilog and SPICE format) and layout descriptions (in LEF/DEF and GDSII formats) of a number of preselected logic gates, which are used exclusively for the implementation of the system described by the HDL. Furthermore, the standard cell library comprises timing and power descriptions of the aforementioned standard cells, usually in Liberty format. This data is used for the timing and power analysis of the circuit.
- The standard cell library comprises the transistor (in Verilog and SPICE format) and layout descriptions (in LEF/DEF and GDSII formats) of a number of preselected logic gates, which are used exclusively for the implementation of the system described by the HDL. Furthermore, the standard cell library comprises timing and power descriptions of the aforementioned standard cells, usually in Liberty format. This data is used for the timing and power analysis of the circuit.
- The standard cell library comprises the transistor (in Verilog and SPICE format) and layout descriptions (in LEF/DEF and GDSII formats) of a number of preselected logic gates, which are used exclusively for the implementation of the system described by the HDL. Furthermore, the standard cell library comprises timing and power descriptions of the aforementioned standard cells, usually in Liberty format. This data is used for the timing and power analysis of the circuit.
- The timing and power information is used to drive the next step of the physical implementation which is the placement and routing in the physical level, again using an automated platform like for example Cadence's SoC Encounter.
- Place and route significantly perturbs the timing and power characteristics of the circuit as all the steps until now have been interconnect oblivious or nearly. After place and route the actual interconnect is in place, and, thus, the actual parasitic loading on each individual net can be estimated. This has a significant

impact on both timing and power, and usually requires a few iterations before the process converges to an acceptable output.

- The final step is to export the design into GDSII format and ship it to the fabrication facility for production.

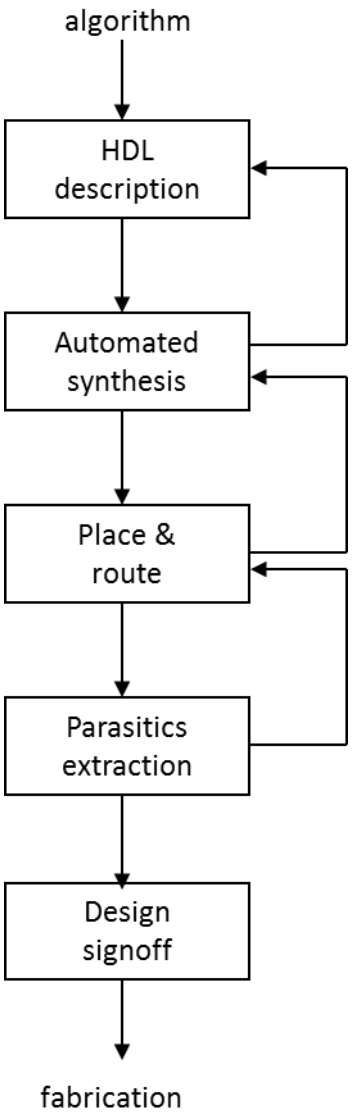


Figure 1: Standard-Cell based design flow

1.0.2 *Continuous transistor sizing*

Continuous transistor sizing is a key-enabler for the performance optimization of nanoscale integrated circuits and allows for significantly more power-efficient chips. Many efforts have been made in the field of continuous transistor sizing optimization problem as it closes the performance gap between a full-custom design circuit implementation and a standard cell based one.

A serious problem that rises during such optimization efforts is the layout resizing procedure. After defining the optimal transistor size for each cell used in the design, new standard cells must be implemented that incorporate the computed sizes. This is a rather complicated task, as each process design methodology is constrained by a large number of layout design rules. On top of that, many cells that belong to the cell libraries are custom designed in order to meet the desired performance metrics.

gds2trim *layout manipulation tool*

In the context of this thesis, a methodology for automated layout manipulation was designed and implemented, along with a layout manipulation tool, *gds2trim*. The implemented tool provides a fully automated layout manipulation solution that can be implemented in a circuit-wide scale and provide fully implementable optimized layouts without human intervention.

Furthermore in this stage of the overall flow, the optimization changes should be performed at the schematic representation of the input design. This means that the input design has already been synthesized, physically placed and routed and meets all the performance standards that are defined by the designer. On top of that having the schematic representation of a circuit (which is the final stage of the standardized industrial IC design flow) means that the design is already checked for timing consistency as well as signal integrity, meets all the power and timing constraints and has passed through the final signoff analysis.

As a logical inference, all the changes performed by the developed program should be of minimum impact on the schematic representation of the input design so as not to violate any of the circuit design rules and to avoid the potential need of replacement and rerouting of the entire design. In other words the optimization changes should take place at the building blocks of the IC design the standard cells, to avoid major changes in the design's schematic representation.

1.1 PROBLEM DESCRIPTION & PREVIOUS WORK

Layout manipulation is a rather tricky procedure due to the large number of layout design rules and the fact that most standard cell libraries are hand-drawn. Even though there is a number of layout generation tools for standard cells, none is "automated enough" to be

used in a large scale operation, as most of the results require partial human intervention (e.g. Sagantec15 Cadabra16). This has led to approaches in which only a few cells are continuously resized, usually on the critical path of the circuit, and then they would be manually drawn and inserted in the standard cell library (Zenasis17, OpenSilicon18). Clearly this approach does not permit globalized circuit optimization and certainly leaves significant margins for further improvement.

The main challenge is the creation of a fully automated approach regarding the layout manipulation process. The methodology should be able to incorporate the new device sizes into the physical representation of the standard cell, automatically creating the necessary schematic files, as well as update the overall design with the proper definitions of customized standard cells that reflect the results of the resizing procedure.

1.2 *gds2trim* APPROACH

The developed layout resizing methodology allows transistor sizes to vary only between the values that fit within the original standard cell footprint, constraining the optimality of the solution somewhat in favor of ease of layout manipulation. By constraining transistor sizes within this range, manipulation becomes tractable by an automated approach as only a small fraction of the design rules need to be accounted for. Linear variations in transistor sizes are easy implementable and do not require human intervention. Thus, they can be implemented in a circuit-wide scale and be readily transferred to the actual layout. Moreover, this approach is compatible with a 'metal respin' production process, whereby a fabricated IC can be optimized by changes in metal masks only, greatly reducing the cost compared to a full silicon re-spin, and enabling interesting business models for exploiting the proposed technology.

The key target of the overall processing by the utility developed, is the reduction of the diffusion areas of the transistors within the rudimental building blocks of an integrated circuit design, the standard cells, in order to achieve the desired power and timing optimizations.

The main criterion which indicates the magnitude of the desired sizing is determined by the analysis on the input design, which is performed by the main resizing algorithm (not in this thesis' context).

The resizes the transistors in the physical level (i.e. layout) and incorporates the resized modules within the chip-level GDSII file. The resizing is done according to layout directives provided by the gate modeling module in order to accelerate the relayout process and prevent design rules errors in a correct by construction approach. This module can handle both planar transistors and FinFETs as the utility can downsize the diffusion area and remove fingers, if necessary.

Inputs to this module are:

1. The cell scaling file, which includes the scaling factor for all gates
2. The layout directives file, which guides the physical implementation utility to generate a correct by construction, DRC-clean layout.
3. The global GDSII file for annotation
4. The standard cells' GDSII files for resizing

This utility provides the finalized layout for both the chip-level and the individual resized gates, so its outputs are:

1. The chip-level GDSII file annotated with the new cell names referring to the resized gates
2. The GDSII files with the layout description of each resized cell.

1.3 GDSII STANDARD

The design schematic representation that will be used as input to the program will be in the GDSII format. GDSII stream format (Graphic Database System) is the major standard in the semiconductor industry for the schematic description of the IC design layouts.

Basic properties of the GDSII stream file format:

Integer Database

GDSII is an integer database. The basic unit of measurement is a nanometer (10-9 meter).Since four byte signed integers are used to describe a coordinate then the integer coordinates can range from minus 232 to plus 232-1. (One bit must be reserved for the plus/minus sign.)

Hierarchical

GDSII is organized in a hierarchical fashion. That is to say, that a number of elements are grouped into a cell or structure, and then that structure is used (or instanced or placed) many times. Since digital IC's are extremely repetitive, the database matches the physical layout very well. Cells can be nested with no limitation as to how deep the nesting goes (though I have yet to see nesting more than 9 levels deep.) It is this nesting and hierarchy that allow one to describe an IC with one billion polygons using a database on the order of 5 GB. Unfortunately, when one needs to compute the actual position of the polygonal entities, one must "reverse" this nesting; for large databases this turns out to be a difficult computation to do quickly.

Binary

The database is binary for compactness. This means that any software for reading or writing GDSII has to be able to extract each byte and interpret the bits. There is no official ASCII equivalent to the binary format.

Record Based

GDSII is divided into "records." Only a few record types make up the great majority of the GDSII data.

The tool should effectively derive the schematic information from the GDSII files provided as input as well as produce GDSII files at the end of the process that are fully compliant with the GDSII file format.

Alongside with the GDSII definitions of the input design and the library standard cells, GDSII mapping files are used to resolve the mapping between GDSII layer entities and their physical counterparts (diffusion area layers, metal layers, poly-silicon layers etc.).

After the production of the resized design, the DEF file of the input design must be updated to contain the new cell type definitions. This must be done for completeness of the design process as well as verification purposes.

The resizing process should have the minimum impact on the design flow. This means that the overall logic, placement, routing cannot be affected by the changes made during this phase of the processing. Moreover the fundamental processing of the standard cell individually must take up the minimum possible runtime as the amount of the total of cell instances to be resized can be arbitrarily large. As a consequence, the tool must have fast access to various data structures provided by its environment that are effectively organized to facilitate this purpose and should also edit the minimum amount of information possible in order to achieve the physical resizing.

2

DESIGN RULE CHECKING (DRC) MANAGEMENT

2.1 LAYOUT GENERATION

The physical implementation of the resized cells from the already existing layout of the cells included in the standard cell library is a key capability, on which rests the success or failure of the entire approach described in this thesis. This is due to the fact that this is the most time consuming part of the tool flow along with the precharacterization procedure. Even with parallelization, special care should be taken for designs of the order of 1 million gates. Furthermore, layout generation has been the bane of similar approaches that have been proposed in the past. A characteristic example is the Cadabra ATL tool coupled with AMPS[*E Yoneno and P Hurat, 2001, Power and Performance Optimization of Cell-Based Designs with Intelligent Transistor Sizing and Cell Creation, IEEE/DATC Electronic Design Processes Workshop, pp. 155-162.*], which resized all cells except sequential and special purpose ones (i.e. tri-state buffers, clock tree buffers). The total number of cells that was actually implemented in the physical layer was 187 out of a total of 70,000 with an average time of 192 seconds per cell. Even though computing resources have improved significantly, runtimes are right at the 60-90 second range even in most recent implementations.[*AI Reis and OC Andersen, 2013, Sizing a Cell Library, US Patent No. 8,615,726*] Even with extreme parallelism these approaches would require weeks to implement all the cells in the tool's target group.

Therefore,our approach was to implement a very fast layout resizing utility but we chose not to implement every new cell from scratch. Rather, we opted to generate the new layout (especially given the fact that no upsizing is permitted) from the already existing standard cell layouts based on a set of design directives unique to each standard cell. These design directives can be generated by experienced designers who overview each standard cell layout in the library and provide the layout resizing utility with explicit instructions on how to achieve the scaling factor calculated from the transistor sizing utility, while taking into account all the design rules pertinent to the transformations applied to the original cell layout, resulting in a correct-by-construction methodology. Thus, generating a resized layout becomes a matter of the order of 0.1secs, which brings the total relayout time to less than a day, if we take into account moderate parallelization of the process on a multicore workstation. The subset of design rules, only for the affected layers (active, contact, and in some case metal 1) are incorporated into the design directives and the layout manip-

ulation utility is simply required to move or delete polygons and be concerned with the correctness of the design, which has been already taken care of. The end result is a flow that permits the completion of the resizing process without DRC (Design Rule Check) errors and in an acceptable timeframe for the users.

A set of design directives for the Nangate 45nm library has been developed for the purposes of the tool's development, in which after studying the topology and the connectivity of standard cell, the design directives were generated, that implement without design rule violations the resized cell at every scaling factor within the acceptable range for the specific cell. The acceptable range for each cell is between 1 (maximum) and a minimum scaling factor. This set of design directives can be used on every design implemented in the Nangate 45nm library.

2.2 DESIGN RULES

Layout design rules are provided by the foundry to designers and intend to ensure the correct and reliable implementation of the design on silicon. The same holds true for the process Nangate's 45nm standard cell library was created for. The entire set of design rules is 80 (see Appendix 1 for a complete list). However, due to the described methodology's limited intervention methodology only a fraction of this set needs to be taken into account while generating the design directives. More specifically, as *gds2trim* targets the active area only, the pertinent design rules are shown in Table 1, since by resizing the active area, evidently, active area and implant layers are affected along with the possible removal of contacts that end up outside the active area after the resizing has been done. This limits the design rules to be taken into account to 12, a number that is manageable for a designer.

Furthermore, in some cases where moving a contact and stretching the associated metal can provide significantly lower minimum scaling factor for the cell, and extra set of another 12 design rules need to be considered, as they have to do with the interactions of metal 1 and contacts. This set is shown in Table 2

The rest of the layers of the physical design are not affected as no design rules can be violated by the Nanotrim methodology. More specifically:

(a) metal layers 2 and up are not affected as they do not appear in the Nangate standard cells

(b) metal 1 design rules are applied only in a subset of cases of contact migration

(c) well rules are automatically satisfied as the active area can only be shrunk, and, therefore, the minimum well enclosure of the active area is already there.

(d) via rules are not pertinent as they do not appear in the design of the standard cells of the specific library, in the same fashion as (a).

During design directive generation special care is taken so that the design rules are observed over the entire range of scaling factors, leaving no room for design rule violations after the automated layout manipulation utility has finished resizing the cell.

2.3 SCALING METHODOLOGY

Based on the principles of sections 1 and 2, the tool's scaling methodology relies heavily on the mask designers generating the appropriate design directives in order to enable the timely and correct-by-construction implementation of the layout of the resized cells. The major points of the methodology are as follows:

(a) each design directive is applied to a single object in the polygon database (GDSII) of the integrated circuit. For example, if the shrinking of an active area results in the need to remove a specific contact, this is explicitly stated as an active area edge move directive and a contact removal directive. There are no implicit directives as this would make the entire approach slower.

(b) scaling of the p-type and n-type active areas is performed independently even though it is by the same scaling factor since the contact pattern in the two active areas might be different and contact removal directives need to be issued at different scaling factors for each active area. This is also due to the fact that it is safe and desirable from a layout standpoint not to remove a contact unless we absolutely have to. Furthermore, it is easier during the design directive generation process to handle separately the p-part and the n-part of a gate and then merge the directives onto a single scaling scale.

(c) each scaling factor interval is self-contained (i.e. the are no "global" statements) in order to speed up the scaling process and to minimize the probability of error.

(d) standard cell scaling is performed exclusively by shrinking the active areas and by the removal of contacts that may exist within the removed active area. Of course, electrical connectivity of the circuit must be maintained at all times and design rule violations must be avoided.

(e) it was observed in certain cases that moving one or a few contacts (and on occasion extending the metal line to make sure that the metal overlap of a contact meets the design rule specifications) leads to significantly lower minimum scaling factor than of the move did not occur. In these few cases, contact moves were permitted and metal lines were slightly extended. This could have been avoided if the standard cell layout had been constructed specifically to enable the Nanotrim design flow.

(f) in cases where there are fingered transistors within the resized cell, scaling can occur either by shrinking the active area along the vertical axis or along the horizontal axis (in this case removing parallel transistors). Obviously floating contacts are also removed.

(g) the incorporation of the layout design rules is implemented by the designer during design directive creation. Therefore, the layout manipulation utility does not perform any DRC check as it is assumed that the generated layout is correct by construction. Of course this will be validated multiple times during the testing of Nanotrim's units and of the system as a whole.

(h) no design directives will be generated for cells that were deemed unscalable by the library analysis, i.e. they have minimum scaling factor of 1.0.

The result of the activity described in this chapter is a file (with *.rdif* extension) that contains all the design directives for all the cells in the library and for every permissible scaling factor such that all the cases produced by the transistor sizing utility can be handled. It should also be noted that the design directives interact with the layout manipulation utility to produce a correct-by-construction layout for the resized cells. AN example of a complete set of resizing directives for a specific type of cell is listed in Figure

Rule	Value	Description
IMPLANT.1	70 nm	Minimum spacing of nimplant/ pimplant to channel
IMPLANT.2	25 nm	Minimum spacing of nimplant/ pimplant to contact
IMPLANT.3/4	45 nm	Minimum width/ spacing of nimplant/ pimplant
IMPLANT.5	none	Nimplant and pimplant must not overlap
ACTIVE.1	90 nm	Minimum width of active
ACTIVE.2	80 nm	Minimum spacing of active
ACTIVE.3	55 nm	Minimum enclosure/spacing of nwell/pwell to active
ACTIVE.4	none	saveDerived: active must be inside nwell or pwell
CONTACT.1	65 nm	Minimum width of contact
CONTACT.2	75 nm	Minimum spacing of contact
CONTACT.3	none	saveDerived: contact must be inside active or poly or metal1
CONTACT.4	5 nm	Minimum enclosure of active around contact

Table 1: Design rules pertinent to gds2trim's design directives methodology (adapted from http://www.eda.ncsu.edu/wiki/FreePDK45:Contents)

Rule	Value	Description
CONTACT.5	5 nm	Minimum enclosure of poly around contact
CONTACT.6	35 nm	Minimum spacing of contact and gate
CONTACT.7	90 nm	Minimum spacing of contact and poly
METAL1.1	65 nm	Minimum width of metal1
METAL1.2	65 nm	Minimum spacing of metal1
METAL1.3	35 nm	Minimum enclosure around contact on two opposite sides
METAL1.4	35 nm	Minimum enclosure around via1 on two opposite sides
METAL1.5	90 nm	Minimum spacing of metal wider than 90 nm and longer than 900 nm
METAL1.6	270 nm	Minimum spacing of metal wider than 270 nm and longer than 300 nm
METAL1.7	500 nm	Minimum spacing of metal wider than 500 nm and longer than 1.8um
METAL1.8	900 nm	Minimum spacing of metal wider than 900 nm and longer than 2.7 um
METAL1.9	1500 nm	Minimum spacing of metal wider than 1500 nm and longer than 4.0 um

Table 2: Design rules pertinent to contact move and metal stretch operations (adapted from http://www.eda.ncsu.edu/wiki/FreePDK45:Contents)

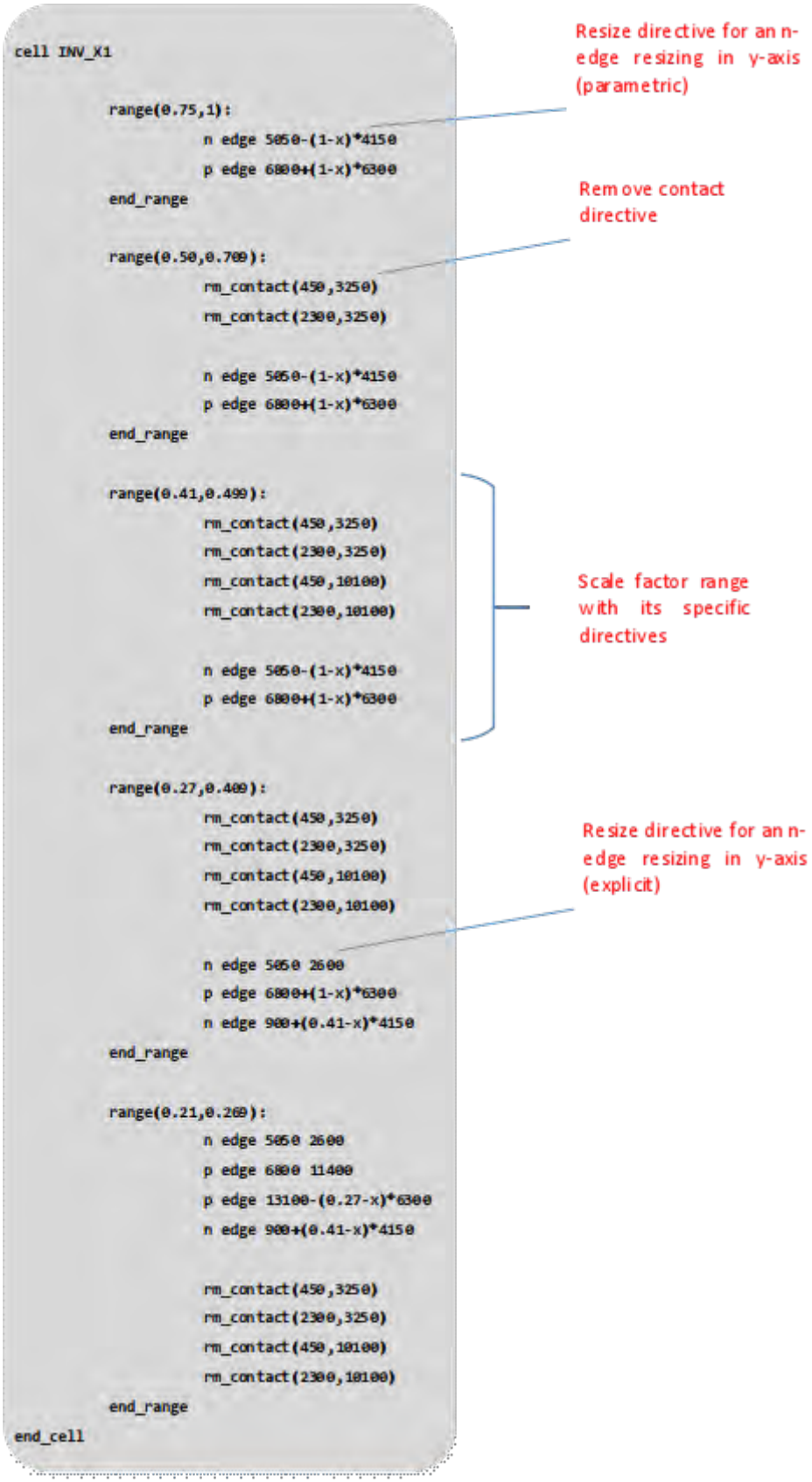


Figure 2: rdif file format sample (cell INV_X1)

3

IMPLEMENTATION

This section contains the full description of the implementation plan of the GDSII manipulation utility along with the methodology followed for the GDSII schematics processing. The whole flow is described from the perspective of:

- input
- procedure
- output

3.1 PLANNING

According to the specifications set in the previous section and in order our program to conform to the state of the art standard cell design flow with the minimum impact, the various widely used commercial tools which participate in the above flow should be examined and their outputs to be analyzed and evaluated. This would allow the definition of the exact point at which the developed tool should intervene and perform the desired optimizations.

More specifically a set of designs widely used for benchmarking purposes by the EDA community, were used for synthesis, physical placement and routing in order to produce their finalized schematic representations in GDSII format files. After that, the GDSII stream file formats were inspected both by a schematic viewing tool, and a hexadecimal editing tool in order to specify the exact method that the commercial hardware development tools structure the GDSII file of an input design. Additionally, an ASCII text version of the GDSII files was produced for further inspection by the viewing tool, to ensure the integrity of the produced results.

The inspection of the above revealed the following results:

The produced stream from the commercial tool for physical placement & routing contains all the placed standard cells as references to the GDSII schematics of the standard cells contained in the PDK that is provided to the tool, using the SREF type record. The matching of the standard cells to their corresponding detailed descriptions is achieved by using the exact library cell name as the name attribute of the reference record (i.e. AND2_X1, INV_X1 for the used PDK)

All the vias are described as structures before the description of the overall design and are later referenced with SREF records throughout the design description.

All the other records are in strict compliance with the GDSII stream file format specifications. A detailed description of the SREF record is as follows:

SREF SNAME XY or SREF STRANS SNAME XY

Where SNAME is the record label containing the cell name, XY the record containing the coordinates that denote the exact placement of the cell and finally STRANS contains the description of the transformations made to the cell.

As a logical inference to the above, it is clear that the schematic information (boundaries, paths etc.) of the finalized design should not necessarily be modified by the developed utility. The exact point at which the developed utility should alter the design information is the reference records of the standard cells and the altering of the original schematic description of the standard cells (also GDSII file format) according to the needs of every input design by creating modified copies of them in order to be used in the resized design. In other words, as the overall design description contains references to the placed cell schematics it is clear that the resizing should be performed in the individual schematic description of the cells contained in the library

This kind of intervention is of minimum impact because modifying the inner characteristics of a standard cell does not affect or alter the physical dimensions of the standard cell. This allows the design's boundaries to remain unchanged during the sizing optimization, and leads to no hazard of geometry or other functional violation that would possibly demand replacement, rerouting and recheck of the entire design.

Furthermore, according to the developed approach every custom layout construction should be transformed and characterized as a standard cell first and then treated by the manipulation utility for resizing.

3.2 METHODOLOGY

3.2.1 *General Flow*

After taking into account the formal GDSII specifications as well as all the other requirements and the conclusions that were reached in the former sub-section the main parts of the manipulation utility are implemented as shown in figure 3:

The implemented approach takes as inputs the original (unsized) GDSII layout representation of the input design, the provided by the PDK standard cell layout files (in GDSII format), a configuration file that contains the scaling factors as well as directives for the resizing of each cell and the standard cells GDSII layout map file provided

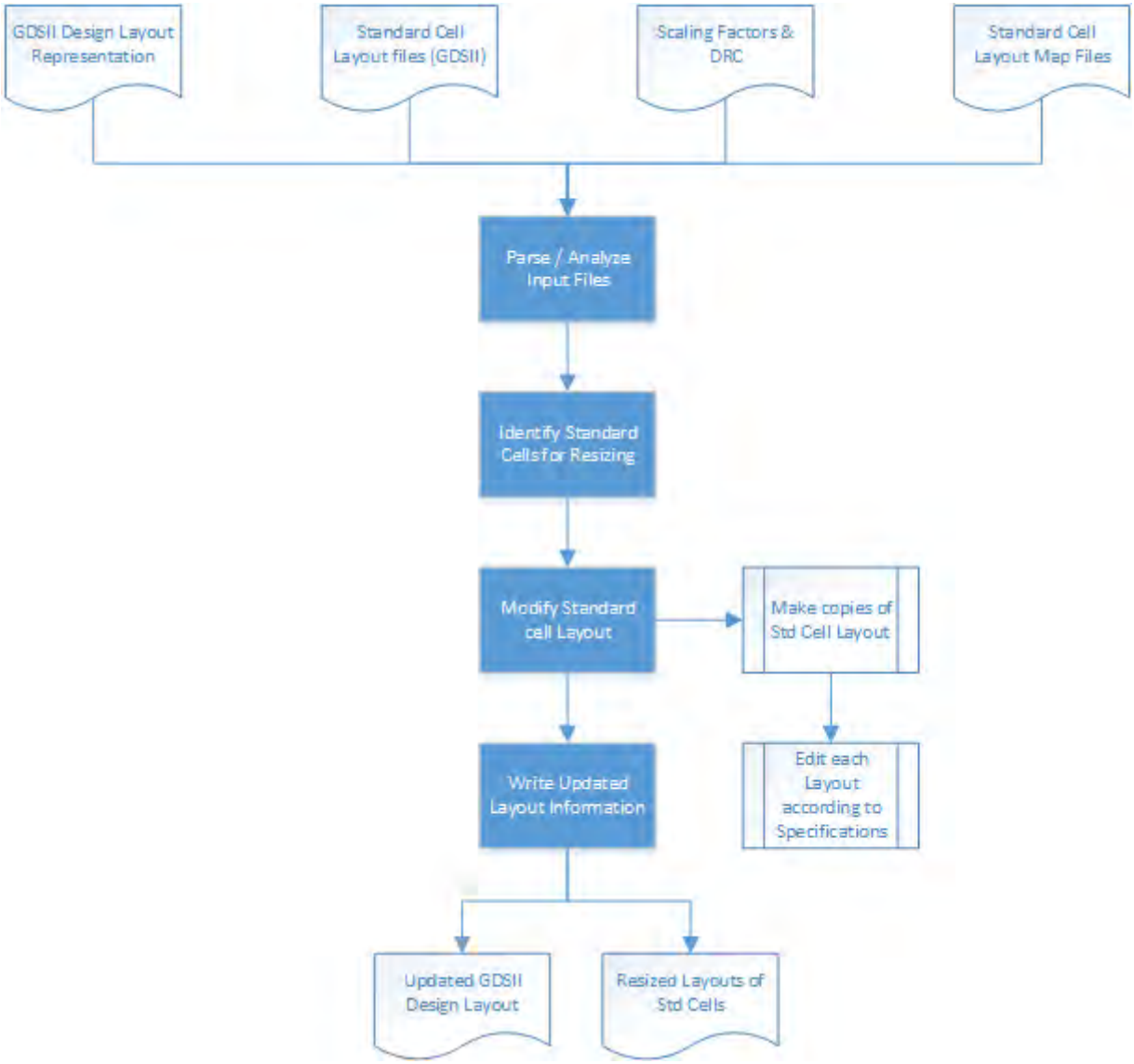


Figure 3: GDSII Manipulation Utility Flow

by the PDK. On the next step, the analysis of the above files follows, in order all the available information to be efficiently stored and organized so as to be easily recalled and processed. More specifically, the input design layout is parsed and the various structures and references are identified. The standard cell references are the points of interest regarding the presented implementation. The standard cell layout files, the specification file containing the DRC rules and the scaling factors, as well as the standard cell layout map file are used in later steps when the editing of the standard cell layouts takes place. Moreover, the data types containing the information contained by the former three file types are mapped to each standard cell library name in order to be easily searchable and accessible.

Subsequently, all the standard cell instances denoted by the references (SREF fields - as described in the previous subsection) in the original design layout are processed for resizing. For every standard cell instance in the input design the resizing procedure takes place as follows:

The utility searches the exact name of the reference name field in the standard cell library (PDK) for the corresponding layout representation. Once found, the standard cell layout is copied and opened for processing (resizing).

Utilizing the information held in the standard cell layout map file, the implemented utility identifies the GDS layers that correspond to the diffusion areas of the transistors of the cell. The utility isolates the corresponding boundaries of the GDS layer and shrinks them to the desired size. The exact way and criteria of the boundaries resizing will be explained in following subsection of this chapter.

After resizing of the boundaries, the modified standard cell layout is written in a new GDSII format file with the proper cell name that corresponds to the resized cell uniquely. This is very important in order to create correct references in the input design in order to include the resized standard cell layouts as references.

Subsequently the utility modifies the name of the reference in the design file that referred to the initial standard cell in order to reflect the newly created resized standard cell.

Once this process is completed for every standard cell reference of the input design, the latter will no longer contain references to the initial standard cell layouts provided by the standard cell library, but will contain references to the resized versions of them.

At this point it is important to make clear that the resizing is performed exactly on the cell types contained in the input design and no other cell type (of the same logic function) is chosen to be resized.

3.2.2 *Cell Resizing Details*

The area of concern as pointed out in previous sections of this report is the diffusion area of the transistors of the standard cell to be resized. In the GDSII layout representation, all the schematic information is organized in layers that reflect the actual structural layers of the cell. By using the information contained in the standard cell layout map file, the layer containing the diffusion area description can be isolated and manipulated according to the directions provided by the specification file, which contains the scaling factors and the DRC directives for every standard cell of the library. The diffusion areas are a set of boundaries that belong to the same GDS layer the id number of which is derived from the map file.

A schematic representation of a standard cell is shown in Figure 4. The diffusion areas to be isolated and resized are marked with the

red dash. In Figure 5 the isolated diffusion areas to be resized are shown.

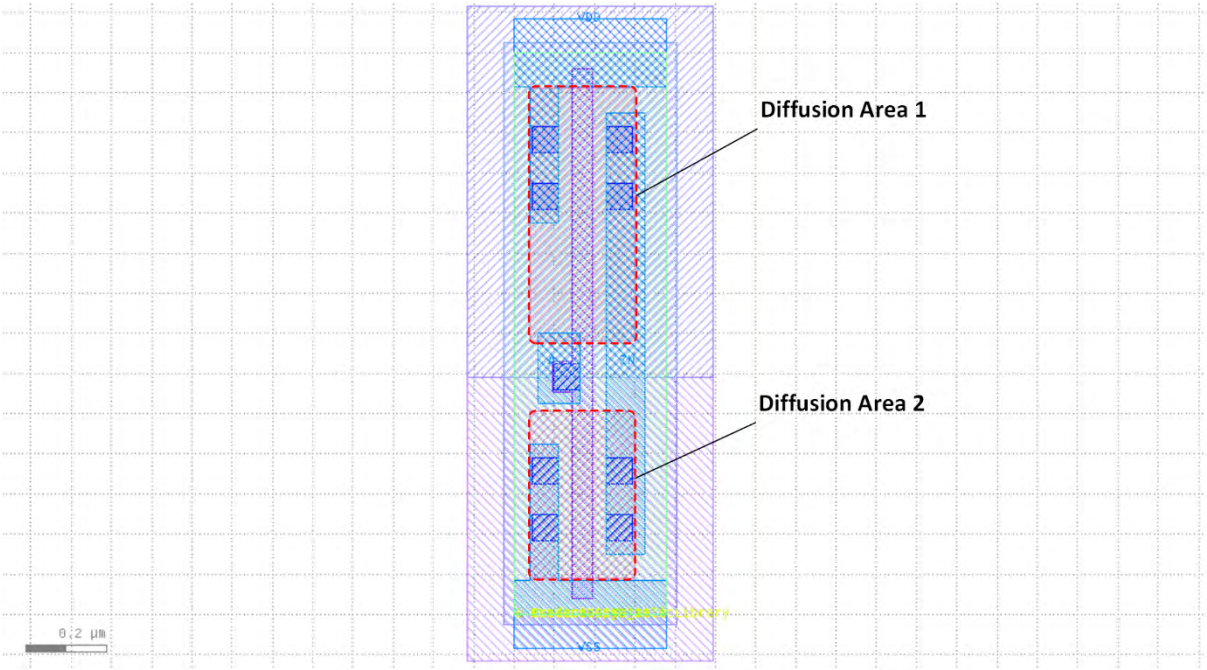


Figure 4: Top View of the standard cell INV_X1

After the isolation step, every boundary of the layer is examined for resizing. More specifically for each boundary the lower left coordinate's pair are treated as its key. According to this key the manipulation utility can resolve the exact way of shrinking the boundaries of the diffusion areas by looking up the directives database which is derived by the resizing directives file. The resizing of a boundary can be performed both along the x and y axis.

For every diffusion boundary there are two possible ways of resizing in y - axis:

1. Shrink the boundary from the bottom edge up.
2. Shrink the boundary from the top edge down.
3. Both of the above

The manipulation utility computes the actual magnitude of resizing by finding the vertical dimension of the boundary (height) and scaling it by the scaling factor. According to this logic the new coordinates of the cell are computed and written back into the boundary attributes. It is important to point that this procedure takes place for every boundary that belongs to the diffusion layer. The computation of the new coordinates (following the annotation of Figure 7) is carried out as follows:

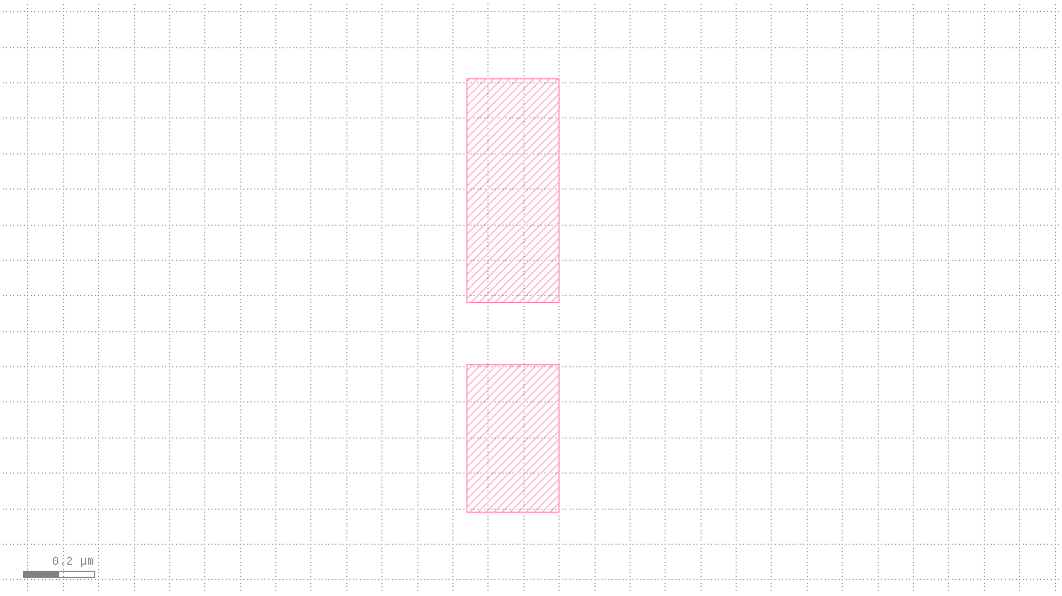


Figure 5: Isolated Diffusion Areas of cell INV_X1

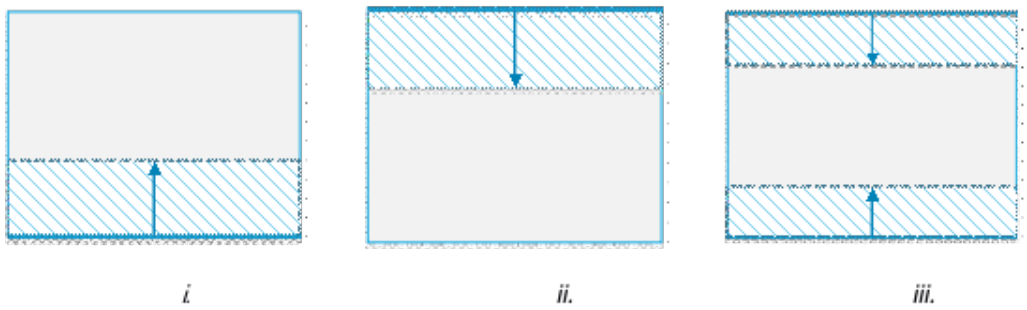


Figure 6: Isolated Diffusion Areas of cell INV_X1

The y-axis resizing directives, according to the specification described in Â§1.1 can be either parametric or explicit. For the parametric directives the new y dimension of the boundary is computed. Both in parametric and in explicit resizing, the new boundary coordinates must be substituted in the GDSII boundary definition. Let yi' denote the y coordinate of the boundary after resizing

- For top-down resizing:

  y2′ : computed, y1′ = y1, y4′ = y4, y3′ = y2′

- For bottom-up resizing:

  y1′ : computed, y2′ = y2, y4′ = y1′, y3′ = y3

- For both sides resizing:

  y1′ : computed, y2′ : computed, y4′ = y1′, y3′ = y2′

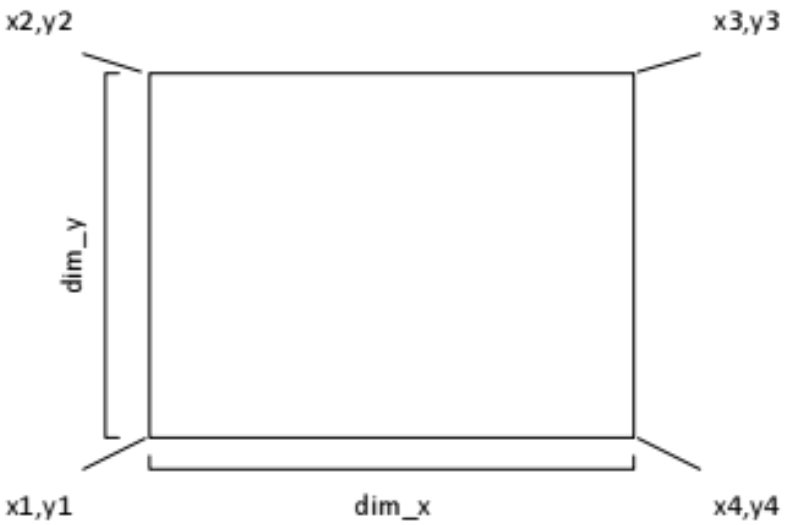


Figure 7: Boundary with coordinates and dimensions explained

The tool also supports resizing in x-axis by explicit directives. The resizing can take place in two directions: a) right to left and b) left to right. Let xi' be the resized x coordinates then:

- For left to right resizing:

  x1' : computed, x2 = x1', x3' = x3, x4' = x4

- For right to left resizing:

  x4' : computed, x2 = x2', x3' = x4', x1' = x1

After the resizing of the diffusion area boundaries, the contact layer must be examined for possible contact removal. Every contact marked for removal is identified by its lower left corner in the directives file. If a contact (which is also e boundary) is to be removed, its definition is excluded in the resized GDS schematic.

In order to enhance the resizing capabilities of the tool, the addition and the removal of whole box boundaries is supported. The layer of interest is specified first and then the removal or addition is performed. In case of box removal, the methodology is the same with the contact removal case. In the case of boundary addition, the lower left and upper right corners of the box are provided by the directive. According to that, the dimensions of the box are calculated and a boundary definition layer is created in the resized version of the standard cell schematic.

Finally, when all the boundaries are resized and updated with the new coordinates as attributes, a new (resized) standard cell is created by writing the new updated information (as well as all the other information contained in the original standard cell that was not modified) into a new GDSII file.

3.2.3 *Design Layout editing*

The described methodology in the former sub-section lies in the core of the overall physical resizing mechanism described in this report. The main functionality controlling the physical resizing procedure prior to resizing the standard cells of the design, must first analyze the top level schematic representation of the input design. By looking up the GDSII mapping file that accompanies the schematic file, the tool identifies the layer that contains the references of the various cells the design is constructed of. By looking up the scale factors file and the DEF file of the input design the tool can correctly identify every cell and map the correct provided scale factor to it. After that identification/mapping procedure the core functionality described in subsection 3.2.2 is put to work for the actual resizing to take place. After every cell of the design is examined for resizing and the new resized versions are created, the tool writes each resized cell layout to GDSII files and updates the reference records of the top level schematic. Finally, the top level schematic is written back to an updated GDSII file.

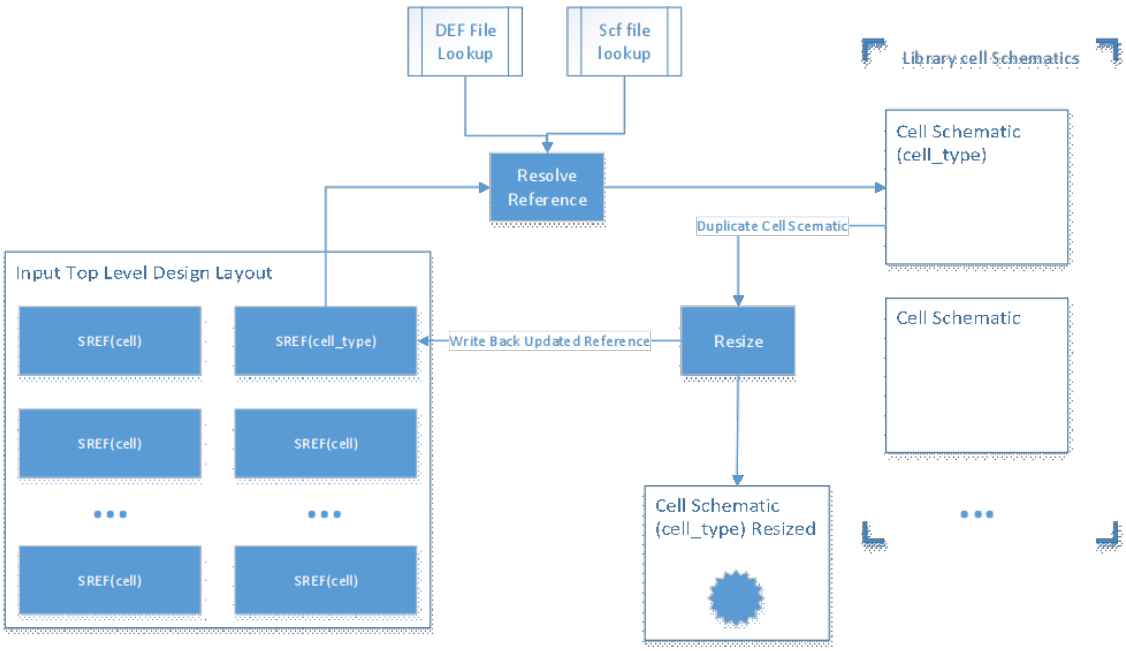


Figure 8: Design Layout editing Overview

3.3 IMPLEMENTATION

Below are described the main implementation technical details of the manipulation utility. First the overall system architecture is presented. Then a more detailed description of the implemented mechanisms and functionalities is presented.

3.3.1 *Utility Architecture*

According to the described specifications and methodology, the architecture of the inner memory representation of the layout information is organized as shown in Figure 9:

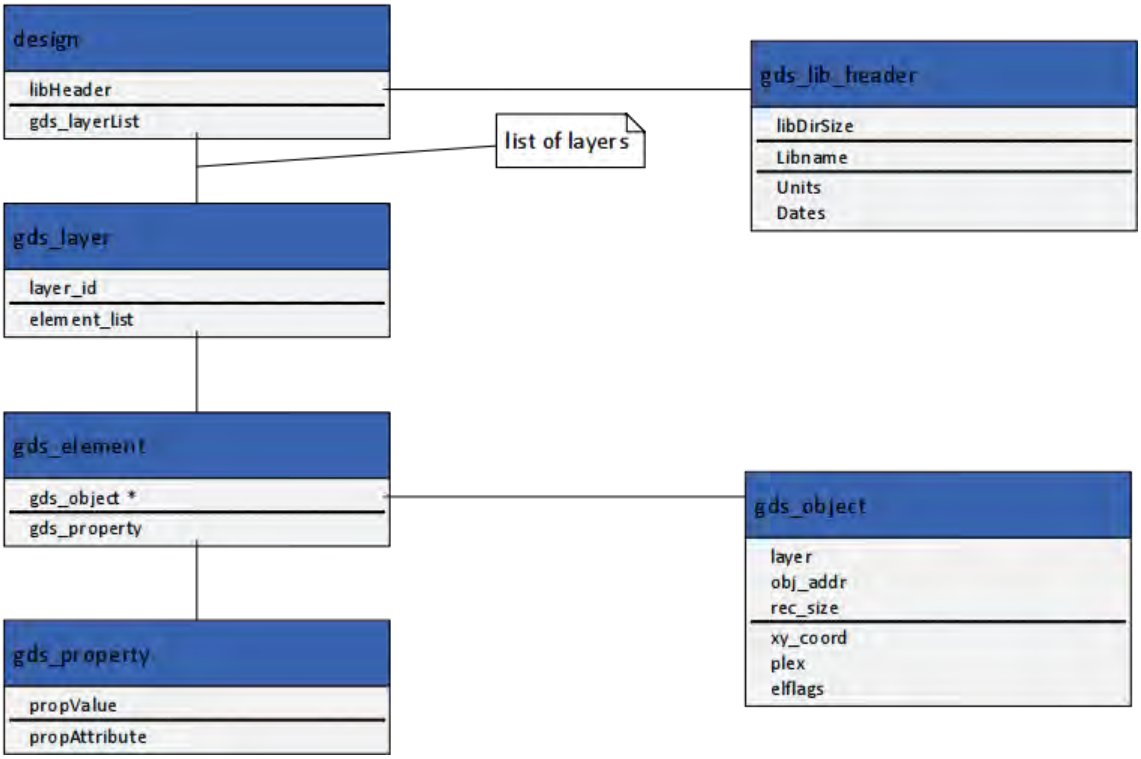


Figure 9: GDSII Data Representation

The aforementioned data types form a generic GDSII data representation scheme and can be used for storing both the input design information and the standard cell layouts information in corresponding object instances of course. The gds_object is a base class and all the other element types described in the GDSII specs are derived from this base class. More specifically all the other compounds of a gds_element are derived from the gds_object class as shown in Figure 10.

The utility stores all the layout files necessary in different design objects which are then read or processed depending on the performed task in each step as described in the methodology section. For the input design the manipulation utility performs a search on the layer list in order to find the layer in which the standard cell references are stored. Then, the references are read in order to identify which standard cell layouts should be resized. After that the design objects that contain the proper standard cells are recalled in order to be processed and resized. After the processing the Libname field of these objects is updated in order to reflect/identify the resized cell layout and the information contained in them is written in new GDSII files. Finally,

the utility updates the reference names of the input design in order to contain the right names of the newly written GDSII layouts.

The overall mechanism that provides the functionality described in this report (will be referred as engine from this point on) is enclosed in a top level object that organizes the main data types used by the various sub processes and also provides a set of wrapping functions that control and execute the various phases of the presented in 3.2.3 mechanism.

The main phases of the engine workflow are:

Engine Initialization This phase is executed first and its main target is to set various engine attributes, initialize the data structures required for the engine's proper function as well as call the constructors of other classes in order to recursively initialize all the necessary underlying objects and functionalities.

Input reading Refers to the reading of the various input files as described in previous sections of this report. More specifically the following

the following files are read and stored into the tools memory in proper data structures:

1. GDSII Design layout file
2. GDSII Library cells layout files
3. Scaling Factors (.scf) files
4. Resizing directives (.rdif) files
5. Design's DEF file
6. GDSII mapping files for design and library cells

The parsing system of the GDSII format tailored for the needs of this project will be explained in later sub-section of this report. The interfacing between the engine and the other inputs are explained in detail in work package 5.2 report. There a planning of the engine's interfacing as well as the structures and functionalities that facilitate the data extraction and accessing are thoroughly specified and analyzed.

Design analysis In this face the GDSII layout information are analyzed and optimally stored in various data structures. The proper objects are created and organized in order to be ready for processing. Moreover during this phase associative data structures are made up in order to expose the input data relations. This is very useful for the correct identification of the cell instances on the GDSII schematic, according to the DEF file and the scf file in order the proper scale factor to be applied. A detailed explanation will follow in later section.

Resizing process This is the main task of the tools engine. In this stage the cells are identified and are processed for resizing. All the cells that are to be resized are processed according to the previously described methodology. The code that performs the cell resizing will be referred to as engine core.

Update definitions After the resizing of the individual cells, the overall design must be updated in order its cell instance references to point the correct (resized) definitions of the cells. Additionally a new DEF file is created with the COMPONENT section updated to include the new cell types' definitions. The GDSII map, rdif and scf files are not modified in any way.

Write output The final stage of the engine's execution is the production of the output files. This means that all the updated components that are stored in tool's memory are flushed in the proprietary files. More specifically the new GDSII files are produced (both an updated version of the overall schematic and the updated library cell layouts) and the new DEF file is written.

Reports/ Display This section of the engine is an individual mechanism that is responsible for printing out various reports and summaries during the execution of the engine. The kind and amount of the displayed information can be adjusted according to the needs of the designer.

Runtime/API The engine can be configured to perform its tasks through various ways. First, the whole engine operation can be explicitly configured and executed by using its interfacing functionalities. This is very useful when the engine is part of an automated toolset. In this case the toolset via the engines API can directly set the data structures that control the engines execution. Secondly, the engine can be manually configured via a simple script that provides the necessary information for the engine to perform its tasks. Finally it can be manually directed through a runtime environment via command line interface.

3.3.2 *Implementation detailed specifications*

3.3.2.1 *GDSII Parsing and analysis mechanism*

In order the resizing tool to meet its goals of minimum runtime requirements and minimum interaction with the original layout files a special way of accessing the layout information should be used. Additionally the observation that only specific areas of the original layout files should be edited, led to the use of alternate file handling techniques. Specifically, the method of memory mapped files was used for performing IO with the GDSII files. This technique offers minimized IO activity as the file contents are can be read in one or more

chunks and be directly mapped into the main memory of the system only once. Moreover this technique is parallel-safe as many processes can map the same memory region into their local memories. Once the file is mapped all the contents of it are available to the processes' memory. In a few words this means that for files that do not exceed the systems physical memory (such as the library cell layout files) the program does not need to perform IO operations (runtime costly) and the accessing costs are reduced to reading and writing the main memory of the program. For the memory mapping integration the C++ Boost Interprocess library was used.

The file handling of the GDSII files is performed by the gds_FileHandler class. This class is responsible for memory mapping the GDSII files and flushing them back into the proper files once the editing is complete. The class contains a vector of the GDSII file names provided as input and a map that connects every filename to its memory mapped region. The file names are given as parameter during the construction of the handler object. The function map_files() performs the memory mapping of the files that are given to the handler object while the flush_mem() function flushes the contents of the memory regions back to the files. The flush_mem is defined to flush only one file at a call. As it can be observed, the gds_SchematicAnalyzer - the class responsible for the layout analysis is declared as a friend class for ease of data accessing.

After the memory mapping, the mapped regions (which can be accessed as tables of binary data) are analyzed by the gds_SchematicAnalyzer. The gds_SchematicAnalyzer class provides functionalities for accessing the binary data with the proper order in order to derive the written information. This class also initializes the GDS data structures described in the architecture section in order to store the derived data. This class also provides functionalities for updating the GDS definitions after the resizing of the cells. This class reads the binary data of the GDS files and is responsible for converting any updated information into its binary data representation according to the specifications of the GDSII File Format.

Listing: gds_FileHandler definition

```
1       class gds_FileHandler
        {
        private:
                vector <char *> *designFiles;
                mapped_mem mmap_stats;
6               map <string, mapped_region> *regions;

        public:
                gds_FileHandler (vector <char *> *files);
                mapped_mem get_mapped_mem () {return mmap_stats;}
11
```

```
            void map_file ();       //memory mapping function
            void flush_mem (char *file);    //flushing
                function

            friend class gds_SchematicAnalyzer;
16      };
```

The function map_file() first copies the input file and renames the copy. The mapping and the processing in later stages are performed on the copy of the GDS file. This methodology is followed for processing the library cell GDSII layout files.

In order the new cell reference names to be correctly written in the GDS file of the resized input design, apart from the copying of the file, a pre-processing of the GDS records is carried out. More specifically, the GDS records are scanned and the references are traced. If a reference is listed as a cell that belongs to the target input library, then the string type record of the reference is expanded. The purpose of this is to provide the exact extra space required within the record so that the updated name can be written without the hazard of a possible truncation of the string. For example, the resized cell's reference name AND2_X1_0.56 requires 5 character space more than the original AND_X1 reference name. This additional space is provided through the mapping mechanism.

A pre-processing is applied in the GDS files of the library-cells, in order to provide the sufficient extra characters' space in the text of the structure name record, in order the cell to be in coherence with the resizing mechanism standards.

The use of the GDS file pre-processing is mandatory due to the limitations imposed by the used memory-mapping library. In specific, the boost interporcess memory map functionality does not allow any changes in the memory mapped region, once the file is mapped. This implies that there is no feasible way to provide the extra space required during the schematic information processing without modifying the characteristics of the memory region.

Finally it should be noticed that the copying of the files was carried out utilizing the sendfile() function and not through the conventional read-write mechanism. The sendfile() function allows the copy procedure to be done at kernel space memory and not in the conventional process memory, speeding up the whole procedure and minimizing the IO cost.

FEATURE ENHANCEMENT: There is no straightforward way to represent the identity information of each cell reference in the design. That happens because the GDS standard is a schematical database

and does not contain any relational information capabilities or mapping data structures to facilitate a straightforward identity resolving. In consequence, the identity representation of the cell references is often chosen by the designer of the physical implementation tool that generates the GDS representation of the design. In order the layout manipulation tool to have enhanced compatibility with the existing EDA workflows, two main identity representation schemas are supported. These come from the most widely used physical implementation tools used in the industrial community, namely, the Cadence SoC Encounter physical implementation tool and the Cadence Custom IC Design system. The first uses a triplet of a TEXT - REFERNCE - BOUNDARY records. The text contains the cell instance id, the reference contains the actual cell reference while the boundary record denotes the physical area that the cell takes up on the design.

Rather than triplets, Custom IC Design System utilizes the PROPATTR and PROPVAL attribute records that supplement a GDS reference element to mark the id of the cell instance the reference belongs to. A convention is made that the first attribute of the reference element i.e PROPATTR = 1, PROPVAL = <name>, is used to hold the id. This convention is easily configurable in the parsers code, so that any attribute number can be set as the id attribute.

The memory access operations are both parametric. For memory read operation, starting from a certain base address, the memory is read and interpreted as a data type specified each time, for an amount specified too. Then the base pointer is moved by the same offset in order to point in the first unread byte:

```
//      READ_MEM macro mem copies to the dest the contents
//      of the the memory region defined by the base and size
//      parameters and updates the base to point to the next
//      unread address.

#define READ_MEM( dest, base, type, ptr_type, offs )    \
memcpy(dest, base, offs * sizeof(type));        \
base = static_cast<ptr_type>(base) + offs;
```

For memory write, there is no base pointer moving. Only the memory is set according to a base address and an offset.

```
#define SET_MEM( source, base, type, ptr_type, offs )   \
memcpy(base, source, offs * sizeof(type));      \
```

3.3.2.2 *Engine Core Implementation*

This section describes the implementation details of the core resizing utility, namely the algorithm that modifies the layouts of the library standard cells. The overall control of the engine core is carried out

by the class *trim_engineCore*. This class initializes every necessary object and mechanism in order to provide the complete functionality of opening, resizing and writing back a cell layout file. This lies in the center of the layout manipulation utility and in the fundamental functionality that the tool provides. This class also provides a set of functions that process boundaries. This class processes the internally stored layout information. The definition of the class is as follows:

The engine core takes as inputs the GDSII layout information, the resizing directives information and the scale factors and performs the designated resizing of the cells. The resizing engine can support boundaries with four or more edges. It can also support unsorted coordinate sets in the boundaries description.

Egnine Core implementation

```
1
//
//      Engine Core's main object. Offers the core
//      functionalities to resize a standard cell
//      The correct directives must be priorly resolved
6 //      by the core's environment and be provided to it.
//

class trim_engineCore
{
11
private:

gds_FileHandler                 *_local_handler;
gds_SchematicAnalyzer   *_local_loader;
16
gds_Design                              *_design;
cell_directives_t               *_directives;
os_fileSysManager               *_file_manager;

21 string  cell_name;
float   scale_f;
int     diff_layer;
int     contact_layer;

26 void orderPolygon(coordinates_t &poly_in);

void calculatePolygonHeight (gds_Object *obj_in, vector <
    ydim_dir * > directives,
pair < long int, long int > ll, pair < long int, long int > ur );

31 void calculatePolygonWidth      (gds_Object *obj_in, xdim_dir *
    x_d);

public:
trim_engineCore();

36 void loadCell   (string name);
```

```
void setupCore  ( int diff_layer_in, int contact_layer_in, float
    factor, os_fileSysManager *file_mngr);

cell_directives_t * resolveDirectives(directives_set_t *set,
    string cell_name);

void trimCell ();
void rmContacts_zero_write(vector <coord_pair_t> contacts);

};
```

The trim_engineCore class is accompanied by a display class that is responsible for displaying various reports and summaries of the process. Currently the implemented report function produces a summary of the read layout and the objects that consists of.

3.3.2.3 *GDS2trim Engine Implementation*

The engine object of the tool is the central object of the tool's software architecture. This object is the realization of the tools internal flow. It utilizes all the previously described mechanisms and classes in order to perform the overall layout manipulation. The tool's engine connects and coordinates all the other objects and functionality in a way that a robust outcome may be produced according to the toolset's specification. The description of the trim_engine class is as follows:

Tool's engine implementation definition

```
class trim_Engine
{
private:

//input arguments
string                                  _rdif_filename;
string                                  _top_gds_filename;
string                                  _scf_filename;
string                                  _top_level_name;
string                                  _lib_gds_dir;
string                                  _work_dir;

//for future use
design_container_t              *_lib_layouts;
design_container_t              *_design_layouts;

//for current use
gds_Design                              *_design;

map<layer_type_t, int>  *_layer_lookup;

trim_directivesHandler  *_directives_dmn;
directives_set_t                *_directives_top;

```

```
scale_factors_t                 _scale_factors;

std::set<string>                _cell_types;
cell_type_map_t                 _cell_instances;

os_fileSysManager               *_file_manager;
gds_FileHandler                 *_local_handler;
gds_SchematicAnalyzer   *_local_loader;



public:
trim_Engine();

void setArgs(string rdif_n, string gds_n, string scf_n, string
    top_str, string lib_dir, string work_dir);
void initializeEngine();
void initializeDirectives(const char *file_name);
void initializeScaleFactors(const char *scf);


des_container_iter_t resolveLibLayout(string cell_name);

int loadTopLevelDesigns();

void testCore();
void execCore(float sc_factor, string type, string full_name);

void trimDesign();
void writeGDS();

void reportTopInstances ();

~trim_Engine();
};
```

This object is instantiated in the main function of the program and its member functions are responsible for the proper execution of the tools resizing flow. As it can be observed, this class contains all the necessary information (i.e. program arguments, internal top-level data structures) that allow the tool's various objects to communicate efficiently and at a minimum cost.

3.3.2.4 *OS (Operating System) integration*

In order to successfully cope with the increased IO demands and the large scale of file handling and new file creation, the OS integration module was created. This module handles all the necessary file naming and listing operations required by the resizing procedure. This module is implemented through the os_fileSysManager class. This class provides functionalities such as path and file assertions, cell li-

brary scanning for the available library cells, resizable cells checking, cells excluded from resizing listing and workspace configuration. Additionally, it holds all the necessary information regarding the paths and the path configuration the tool uses. The definition of the aforementioned class is listed below:

Tool's OS integration mechanism

```
class os_fileSysManager
{

private:

string          _workspace_path_arg;
string          _path_arg;
path_container_t        *_lib_cell_paths;
std::set<string>        *_lib_cell_list;
std::set<string>        *_exclude_cell_list;


void            initExcludeList();

public:

os_fileSysManager();
os_fileSysManager(string path_in);

void            setPath (string arg)    { this->_path_arg = arg;
    }
string          getPath()       { return this->_path_arg; }

void            setWorkspacePath (string work);
string          getWorkspacePAth()      { return this->
    _workspace_path_arg; }

void            addCellExclusion (string arg);
std::set<string>        *getExclusionList() { return this->
    _exclude_cell_list; }

char *          queryFile(string filename);
fs_type_t       queryPath(string path);
path_container_t        *locateFiles(vector<string> key_list);
std::set<string>        *mapLibCells();
bool            is_ResizableCell(string name);

~os_fileSysManager ();

};
```

FUNCTIONALITY: The class is initialized with the cell library location path and the workspace location path. The latter is the path that will contain the resized design file along with the resized library cell files. Then the available library cells list is created by scanning the

provided library location path. The excluded cell list is created, which contains the cells that are excluded from the resizing procedure (such as flip-flops or multiplexers). The created object is then passed to the created filehandler objects and created by the engine and the engine core instances. The object's methods are called during the resizing procedure, to assert correct path validity and check whether the cell is excluded from resizing or not.

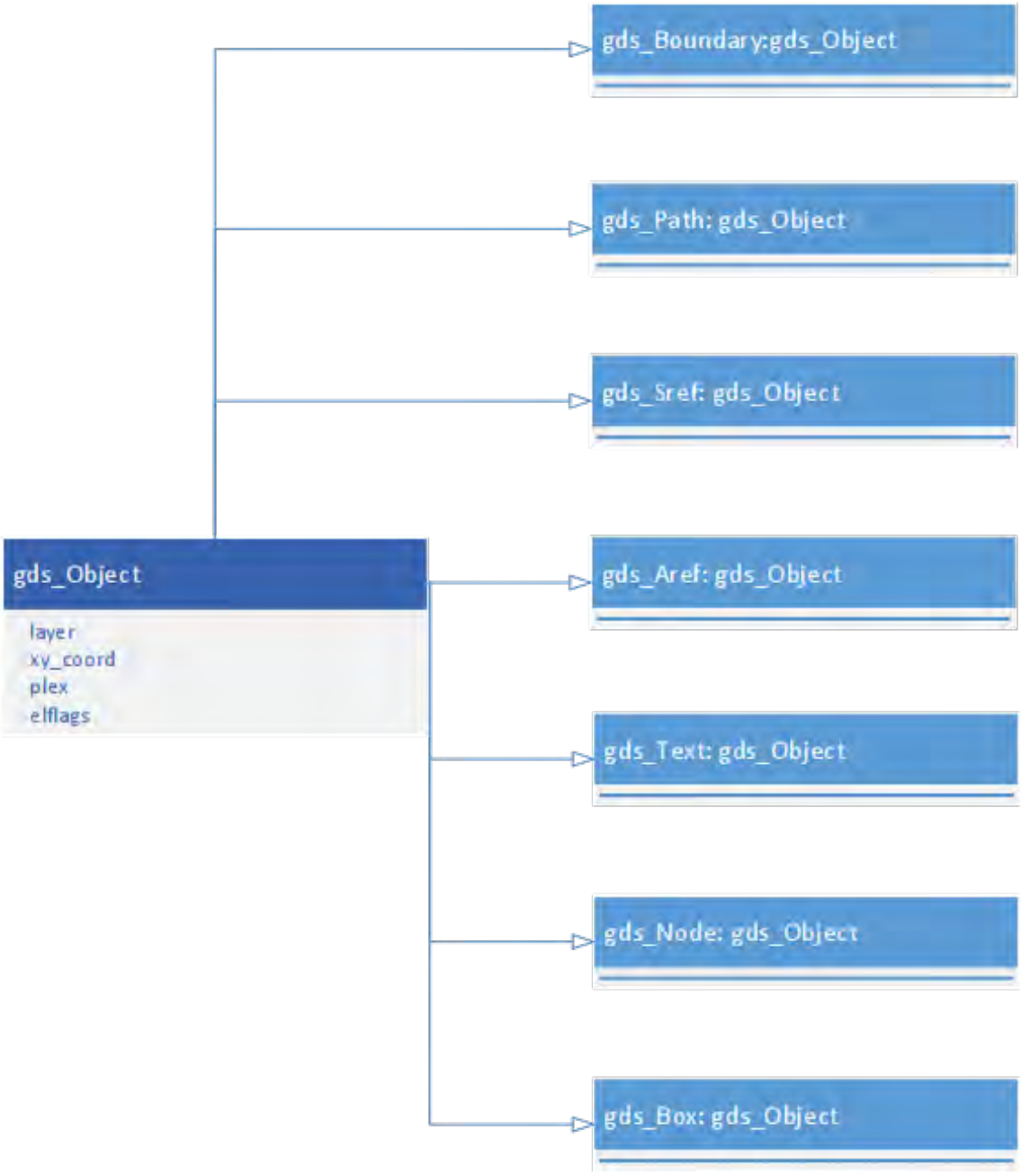


Figure 10: gds_Object inheritance

4

INTERFACE TO TIMING ANALYSIS

The resizing algorithm used to suuply the scale factors to the gds2trim tool performs the ULE method for delay evaluation and minimization in paths composed of CMOS logic gates. Each path, that will be evaluated, can be chosen arbitrarily from a pool of paths, but this technique will not yield the best results. A choosing criterion must be used in order to select the most suitable path every time. That criterion would ideally be a combination of metrics, which is power consumption and delay. The developed algorithm only uses the delay as a filter when examining paths.

Timing analysis must be performed in order to sort the available paths according to their criticality, that is extracting the paths with the worst or best delay (Late / Early timing analysis). Since the resizing algorithm is very conservative, the timing analysis will only be performed for the worst delay paths. There are different approaches for timing analysis, such as static timing analysis, dynamic timing analysis, statistical timing analysis, and each of them uses a different methodology to analyze the circuit.

The static timing analysis procedure requires both cell and interconnect delay information in order to perform produce timing information for a given input design. In order the delay of a standard cell to be computed, the capacitive load this cell drives must be known, prior to this computation. Thus the capacitive load of both cells' input pins and interconnect wires is required. Finally, all the possible transitions, given a certain combination of input-output of a cell must be defined (unateness).

Moreover, apart from the delay calculation for the standard cells and interconnect wires, it is necessary the arrival times, input/output transitions at the design's primary inputs as well as the capacitive load the design drives at its primary outputs must be determined.

Static timing analysis verifies circuit timing by "adding up propagation delays along paths between clocked elements" in a circuit. It checks the delays along each path against the specified timing constraints for each circuit path and reports any existing timing violations.

Dynamic timing analysis verifies circuit timing by applying test vectors to the circuit. This approach is an extension of simulation and ensures that circuit timing is tested in its functional context. This method reports timing errors that functionally exist in the circuit and avoids reporting errors that occur in unused circuit paths.

Statistical timing analysis is a variation of the static timing analysis which replaces the normal deterministic timing of gates and interconnects with probability distributions, and gives a distribution of possible circuit outcomes rather than a single outcome.

Every method has its own advantages and disadvantages, but for the purposes of the resizing algorithm, static timing analysis was selected, to provide an estimate of the worst paths, with the least algorithm complexity.

In the context of the Nanotrim continuous transistor sizing toolkit, a timing analysis tool was developed in order to supply the required timing information to the resizing algorithm in order to compute the proper cell sizes and export the cell scaling factors that are subsequently used by the gds2trim tool. In the context of this thesis the interface to the timing analysis tool is elaborated.

All the relevant information for the delay, input pins capacity and the set of possible input-output transitions for each cell, is available in the timing library file. The most widely used format for timing information files is the *Synopsys Liberty Timing Library format* denoted by the extension *.lib*. The wire delay is computed form the distributed RC interconnection network, described in the *Standard Parasitic Exchange Format* file (.spef). Finally, he initialization information of the timing analysis tool that includes the arrival times at the primary inputs and its transitions as well as the capacitancies driven by the design's primary outputs, are included in the timer's configuration file.

4.1 .LIB INTERFACE

In the listing that follows a .lib timing information section is shown for the cell INV_X0.759036

```
1
cell ("INV_X0.759036") {
...

        pin (A) {
6               capacitance : 1.025479; %input capacitance
                direction : "input";    %of pin A
                ...
        }

11      pin (ZN) {
                direction : "output";
                ...
                timing () {
                        %A-Z transition. Negative unativity
16                      related_pin : "A";
                        timing_sense : "negative_unate";
                        ...
```

```
                        cell_fall ("tmg_ntin_oload_7x7") {
                                index_1(...);
                                index_2(...));
                                values(...)
                                ...
                        }

                        %rise delay
                        cell_rise ("tmg_ntin_oload_7x7") {
                                index_1(...);
                                index_2(...);
                                values(...)
                                ...
                        }

                        %fall time @ pin Z
                        fall_transition ("tmg_ntin_oload_7x7") {
                                index_1(...);
                                index_2(...);
                                values(...)
                                ...
                        }
                        rise_transition ("tmg_ntin_oload_7x7") {
                                index_1(...);
                                index_2(...);
                                values(...))
                                ...
                        }
                }
        }
}
/* end of cell */
```

4.1.1 .*SPEF Interface*

A part of the SPEF file of the c17 physically implemented design of the ISCAS 85 bennchmarking suit. In this example the net N19's distributed RC interconnection network is listed.

```
*NAME_MAP

*1 N23
*2 N22
*3 N7
*4 N6
*5 N3
*6 N2
*7 N1
*8 VDD
*9 VSS
```

```
*10 N16
*11 NAND2_6
*12 NAND2_5
*13 NAND2_
*14 N19
*15 NAND2_4
*16 N10
*17 NAND2_1
*18 N11
*19 NAND2_2

...

*D_NET *14 0.1575969    #total capacitance of net 14
...

*CAP

1 *11:A2 0.0
2 *15:ZN 0.01060115
3 *14:3 0.02944625
4 *14:4 0.04005637      %distributed capacitance
5 *14:5 0.02683567
6 *14:6 0.04005637
7 *14:7 0.01060115

*RES
1 *14:3 *11:A2 22.5
2 *14:4 *14:5 22.5
3 *14:6 *14:7 22.5      %distributed resistance
4 *14:7 *15:ZN 22.5
5 *14:3 *14:5 0.7175533
6 *14:6 *14:4 2.026962

*END
```

An optical representation of the above interconnection network can be shown in Figure 11:

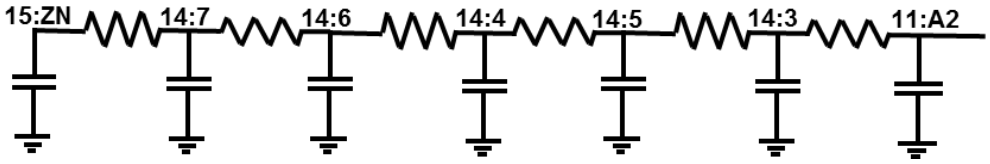


Figure 11: N19 Interconnection network

5 RESULTS

5.1 OVERVIEW

In this part of the project the main objective is to implement the physical layout manipulation system. First a relative search on the field should be carried out in order to examine all the details of the physical implementation phase of the standard cell design flow and how it is carried out by the state of the art tools that are used widely academically and in the industry. The research that was carried out by carefully exploring the tools that provided physical implementation solutions, their configuration options, and the data relationships that existed through the various file formats that needed or should be produced. A number of well specified file formats was inspected in order to determine the way the resizing tool would efficiently extract all the necessary information for the correct resizing procedure. A set of third party tools was used in order to analyze the produced information and thoroughly inspect the structure and information of all the layout files produced by the physical implementation tools and how this structure changes as various parameters are altered. Moreover emphasis was given not to interfere in any unnecessary stage of the known physical implementation flow as well as to find out which files should be edited and updated alongside with the GDSII files in order to provide a well specified physical implementation of the design in order to be actually constructed later.

After the research phase the goals and specifications of the resizing tool implementation were set and the tool planning took place according to them. The data structure organization was specified alongside with the various mechanisms that the tool will support to carry out its primary tasks. The data structures used facilitate the ease of access, searching and the associativity between data of the various inputs of the tool

After the total of the tool's architecture and functionalities was specified the phase of development commenced. The used programming language is C++. All the abstracted planning was modeled and implemented using data containers and data types from verified and optimally implemented libraries such as STL and Boost C++ libraries. Emphasis is given in the object oriented programming principles as well as the ease of maintenance and further extension of the code.

The engine core mechanism-the fundamental resizing functionality and the integration with the workflow mechanisms were successfully fully implemented and various unit tests were carried out in order

to ensure every component of the engine core flow works flawlessly. Various versions of the directive file formats were used (gradually more detailed), in order to enhance the resizing capabilities of the tool. In the current phase with the engine core ready the main engine mechanism of the tool is being under development and testing. All the data structures are being well specified and mapped to actual C++ data structures and objects. The various functionalities that the tool will provide as a system are being developed and thoroughly tested.

5.1.1 *Results*

The layout manipulation tool was tested in various cases of input designs. The tool's behavior and performance was inspected and analyzed. The tool's runtime testing was performed in two stages. First, the engine core module was tested and profiled. Then, the whole tool was executed with various input designs in order to evaluate its performance. The two faces of the analysis are described in the following sub-sections.

5.1.2 *Engine Core Evaluation*

In this step, the performance of the engine core was evaluated. This was the first section of the tool to be profiled and analyzed as it is the core functionality of the tool. Additionally, for the aforementioned reason, the performance of the engine core mechanism is crucial for the performance closure of the whole manipulation mechanism, as it represents the majority of the workload carried out during the resizing procedure. It is a functionality that may potentially apply to the majority of a design's cells, in contrast , i.e, to the main design parsing or the scale factors/directives parsing mechanisms that are evoked only once for every execution. This makes the engine core performance a key value for the whole tool's performance evaluation.

A large set of input library cell GDS files were provided as inputs to the engine core module. The latter was configured to resize the aforementioned cells to various scale factors. Emphasis was given so that complex representative cases to be chosen as well (i.e. scale factor regions that require the largest number of layout modifications for a certain type of standard cell), in order to obtain a thorough image of the module's performance.

The Nangate 45nm OpenCell Library was used for the tests. The evaluation took place on a system with the following characteristics:

- Intel Core i7 4-core @ 3.60GHz
- 16GB RAM DDR3
- 500 Gb Hard Disk

The resized cell generation performance for representative cases is shown in the table 4:

Std Cell	Scale factor	Produced Cell	Execution Time (sec)
AND2_X1	0.79	AND2_X1_0.79	0.000166
AND2_X1	0.50	AND2_X1_0.50	0.000330
INV_X1	0.50	INV_X1_0.50	0.000293
INV_X1	0.35	INV_X1_0.35	0.000280
INV_X1	0.90	INV_X1_0.90	0.000309
OAI211_X1	0.55	OAI211_X1_0.55	0.000264
OAI211_X1	0.90	OAI211_X1_0.90	0.000465
AOI222_X1	0.37	AOI222_X1_0.37	0.000100
AOI222_X1	0.60	AOI222_X1_0.60	0.000213

Table 3: Cell Generation Runtimes

As it can be observed by Table 1, the engine core module shows an overall exceptional performance as the execution time is below 0.5 millisecond and generally there was no case that caused an execution time larger than 1 the one millisecond, a performance that allows thousands of cells to be resized within seconds. This is of utmost importance in modern designs, in which the potential number of cells can be in the scale of millions.

Another useful observation is that the execution times differ, even among the resized instances of the same cell times. This happens due to the fact that the various scale factors that belong to different range sets engage a different set of resizing directives within the same cell type.

The produced GDS files are fully functional, as they can be opened, read and processed by GDS layout editing and viewing tools. Additionally, they are in appliance with the library's design rules. A graphical representation of a resizing case is given in Figure 12.

As it can be observed, the active regions are shrunk and two of the contacts on the n-mos region are removed. This is an expected result, according to the resizing directives provided for the specific resizing configuration.

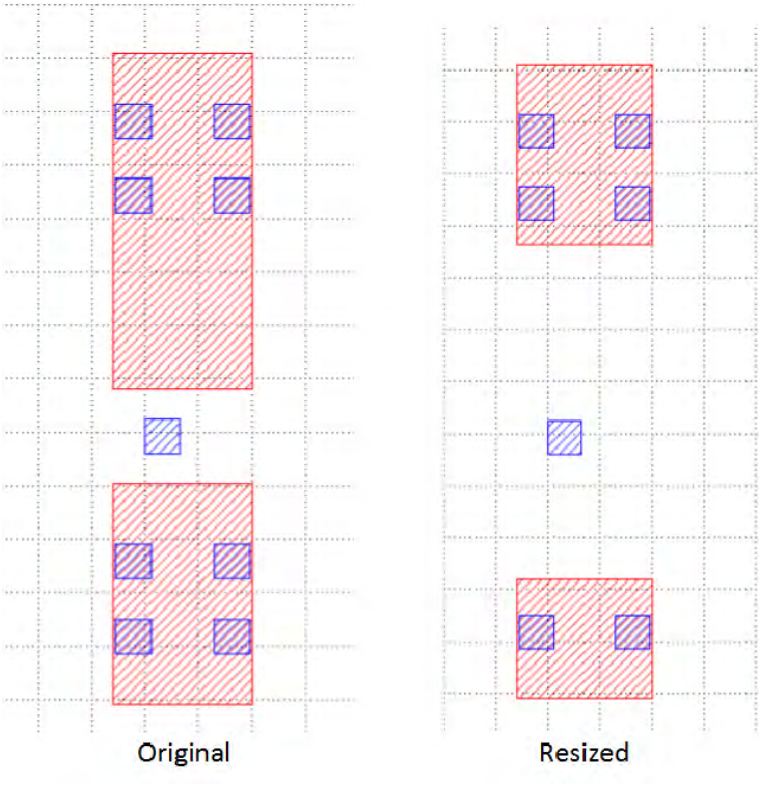


Figure 12: Resizing result of cell INV_X1 to INV_X1_0.55

5.1.3 gds2trim *Evaluation*

In this section the whole system's evaluation in terms of performance is described. The main designs provided as inputs to the tool are the ISCAS benchmarking circuits along with additional large scale benchmarks used by the EDA community. Moreover, a synthetic benchmark was constructed in order to evaluate the tool's performance in the case of a very large input set.

Moreover the results from the synthetic benchmarks are as listed in Table .

Design	Cells	Runtime
ac97_ctrl	13375	1s
aes_core	22403	1,6s
aes_core_oneM	1M	120s

Table 4: Synthetic Benchmarks

5.2 KEY CHARACTERISTICS

In the context of the tool's development, key characteristics were utilized to enhance performance and render the tool highly configurable and flexible for further future development. These features play a critical role in the overall good performance of the tool, and allow it to alleviate modern techniques in order to cope with its highly demanding task. A brief inspection of such key characteristics used by the tool is:

MINIMIZED INPUT / OUTPUT DEMANDS The tool's operations are highly I/O demanding. This relies to the fact that reading and updating GDS stream data involves multiple read and write operations within a single file. In the scale of an actual input design, this cost is multiplied thousands, even millions of times, depending on the total number of resizable cells the input design contains. This fact can impose a significant performance barrier and even in cases of massive input sets, render the whole resizing operation prohibiting in terms of execution time and resources. This is tackled by using the technology of memory mapped files in order to process the GDS files. This technique requires only one read and write operation, at the beginning and at the end of the file editing respectively. This brings the I/O requirements of the tool to a minimum, as the minimum interaction with the I/O subsystem is achieved. It must be mentioned that the I/O subsystem is a usual bottleneck in I/O intensive applications. This allows the tool's performance to be faster by orders of magnitude, compared to the traditional I/O handling because all the processing is carried out in machine's main memory.

PARALLELISM The tool's architecture is designed so that every resizing operation to be totally data-independent from another. This is a key design concept that allows the tool to execute multiple resizing operations in parallel. Every resizing operation can be modeled as a standalone task along with the main design processing that is the initial task. This allows the utilization of both task-level parallelism and data-level parallelism, making the tool versatile to be accelerated in various perspectives and platforms such as servers or cloud computing machines.

BOOST INTERPROCESS The tool makes use of the boost interprocess library. The latter is a versatile modern library for complex, high performance memory operations. It offers mechanisms for memory management, memory mapping and memory sharing. By utilizing this library, the tool has enhanced versatility and extensibility in terms of memory handling and management, a critical aspect of the tools performance limits. By using this library, gds2trim can be easily ex-

tended to support multi-process execution with shared process memory models, a key feature for additional speedup.

Benchmark	Total Cell No.	Resized Cells	Resized Percentage	GDS2Trim Runtime
c17	6	0	0.00%	0.3ms
c432	168	18	10.00%	2.5ms
c499	210	13	6.10%	2.783ms
c880	383	17	4.40%	3.694ms
c1355	554	34	6.10%	5.749ms
c1908	932	28	3.00%	5.811ms
c2670	1278	51	3.90%	10.173ms
c3540	1719	21	1.20%	11.361ms
c5315	2332	80	3.40%	14.223ms
c6288	2416	28	1.10%	12.963ms
c7552	3573	118	3.30%	19.883ms
s27	13	0	0.00%	1.99ms
s298	136	13	9.50%	1.573ms
s344	175	10	5.70%	1.843ms
s349	176	0	0.00%	0.7ms
s382	183	18	9.80%	2.245ms
s400	188	11	5.80%	1.762ms
s420	237	19	8.00%	2.515ms
s510	217	12	5.50%	pending
s526	222	0	0.00%	7.03ms
s641	398	24	6.00%	3.231ms
s713	412	0	0.00%	1.239ms
s820	306	0	0.00%	1.043ms
s953	424	0	0.00%	1.133ms
s1196	549	0	0.00%	1.825ms
s1238	528	0	0.00%	1.335ms
s1423	733	0	0.00%	2.4ms
s1488	659	0	0.00%	2.361ms
s5378	2970	88	3.00%	18.506ms
s9234	5833	303	5.20%	40.917ms
s13207	8605	0	0.00%	27.27ms
s15850	10348	0	0.00%	31.32ms
s35932	17793	0	0.00%	74.139ms
s38417	23835	355	1.50%	pending

Figure 13: Benchmark Runtimes

DECLARATION

I confirm that this Master's thesis is my own work and I have documented all sources and material used.

This thesis was not previously presented to another examination board and has not been published.

Volos, Greece, November 2015

Panagiotis - Taxiarchis
Giannakou, November 16,
2015