INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

LAYGEN II – Automatic Layout Generation of Analog ICs based on Template Descriptions and Evolutionary Computation

Ricardo Miguel Ferreira Martins

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente: Prof. Marcelino Bicho dos Santos

Orientador: Prof. Nuno Cavaco Gomes Horta

Co-Orientador: Prof. Jorge Manuel Correia Guilherme

Vogal: Prof. João Carlos da Palma Goes

Maio de 2012

Abstract

The work presented in this report belongs to the scientific area of electronic design automation (EDA) and addresses the automatic generation of analog integrated circuit (IC) layout. An innovative design automation tool based on template descriptions and on evolutionary computation techniques, LAYGEN II, which stems from LAYGEN, was developed to validate the proposed approach giving special emphasis to the reusability of expert design knowledge and to the efficiency on retargeting operations. LAYGEN II intends to be integrated in the bottom-up physical synthesis path of an analog design automation process, along with an in-house tool, GENOM-POF, which performs automatic IC sizing in the top-down electrical synthesis path of the flow. The designer specifies the sized circuit-level structure, the required technology and, also, provides the technology independent high-level layout guidelines through an abstract layout description, henceforward called template. The generation proceeds in the traditional way, first placement and then routing. For placement, the topological relations present in the template are mapped to a non-slicing B*-tree layout representation, and the tool automatically merge devices and ensures that the design rules are fulfilled. The router optimization kernel consists of a modified version of the multi-objective evolutionary algorithm (MOEA), NSGA-II, and uses a built-in design rule check (DRC) as evaluation engine. The automatic layout generation is here demonstrated using the LAYGEN II tool for two selected typical analog circuit structures, namely, a fully-dynamic comparator and a single-ended folded cascode amplifier. The layouts were generated for two design processes, UMC 130 nm and AMS 350 nm, and the output provided is a GDSII stream format, a file standard for data exchange of IC layout. Automatic generation processes were performed in less than 5 minutes, which allow for the designer to quickly obtain a first cut solution. The results were validated using the industrial grade verification tool Calibre® to run DRC, layout versus schematic (LVS), and also extraction, in addition post-layout simulations were successfully performed.

Keywords

Analog Integrated Circuits Design

Automatic Layout Generation

Electronic Design Automation

Evolutionary Computation

Resumo

O trabalho apresentado neste relatório pertence à área científica de automação de projecto electrónico e foca a geração automática de *layout* de circuitos integrados analógicos. Para validar a metodologia proposta foi desenvolvida uma ferramenta, o LAYGEN II, que utiliza técnicas de descrição de *template* e computação evolutiva. O principal objectivo é reaproveitar o conhecimento do projectista e aumentar a eficiência das operações de migração para outras especificações ou tecnologias. Esta abordagem tem como ponto de partida uma implementação anterior, o LAYGEN. A ferramenta pretende ser integrada num processo automático de projecto de circuitos analógicos, juntamente com uma ferramenta de dimensionamento automático de circuitos, o GENOM-POF. Na metodologia proposta, o projectista começa por especificar a estrutura do circuito dimensionado, a tecnologia pretendida e desenvolve uma descrição de alto nível do *layout*, através do uso de *templates* independentes da tecnologia. A geração do *layout* decorre da forma tradicional, primeiro é feito o posicionamento dos blocos na área do chip e depois são efectuadas as ligações eléctricas entre eles. Para o posicionamento, as relações topológicas contidas no *template* são mapeadas para uma representação B*-tree, e a ferramenta realiza automaticamente a sobreposição de dispositivos e garante que as regras da tecnologia são impostas. Para as ligações é utilizado um núcleo de optimização multiobjectivo, baseado numa versão modificada do algoritmo NSGA-II, associado a um procedimento interno de verificação de regras como método de avaliação. A geração automática do *layout* é demonstrada para dois circuitos seleccionados, utilizando duas tecnologias de integração, *UMC* 130 *nm* e *AMS* 350 *nm*. O resultado gerado é um ficheiro GDSII, um formato *standard* para troca de informação de *layout* de circuitos integrados. As gerações automáticas dos *layouts* foram realizadas em menos de 5 minutos, o que permite ao projectista obter rapidamente a primeira iteração de projecto. Os resultados foram validados usando a ferramenta de verificação industrial Calibre®, e foram realizados os testes de DRC e LVS, assim como simulações sobre o circuito extraído do layout.

Palavras Chave

Projecto de Circuitos Integrados Analógicos

Geração Automática de *Layout*

Automação de Projecto Electrónico

Computação Evolutiva

Acknowledgements

I would like to acknowledge my supervisor Prof. Nuno Horta for all the support, trust and guidance since the first day, and also, the opportunity to develop my Master Thesis in the area of integrated circuit layout in the unique conditions available in the Instituto de Telecomunicações. Nuno Lourenço, for providing me the excellent tool he developed in his Master Thesis, LAYGEN. His never-ending work, support and motivation, allowed for this project to reach the level which I have the pleasure to present in this document.

I would also like to present a word of recognition to all involved in the AIDA project, Prof. Jorge Guilherme, Prof. João Goes and Prof. Nuno Paulino from CTS-UNINOVA, and Prof. Pedro Santos, whose valuable discussions and ideas have undoubtedly contributed for the progress of this work.

Now, to my friends, Hugo Rafael, Diogo Teixeira, Filipe Grou, João Vasco, my brothers from Instituto Superior Técnico, and many others, who supported me in the last, hard but rewarding months.

To my parents, for their support and life-time example of humbleness, to my sister and little Nádia. Finally, a word of gratitude to Nocas, who supported me since my first day in Instituto Superior Técnico.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASF Automatically Symmetric Feasible

AMS Analog and Mixed-Signal

BSG Bounded-sliceline Grid

CAD Computer-Aided-Design

CMOS Complementary Metal Oxide Silicon

DRC Design Rule Check

DSP Digital Signal Processing

EDA Electronic Design Automation

ERC Electrical-Rule Check

GA Genetic Algorithm

GUI Graphical User Interface

IC Integrated Circuit

LVS Layout versus Schematic

MOEA Multi-objective Evolutionary Algorithm

POF Pareto Optimal Fronts

SA Simulated Annealing

SoC System-on-a-Chip

SP Sequence Pair

TCG Transitive Closure Graph

VLSI Very Large Scale Integration

Chapter 1 **Introduction**

In the last years, the world has observed the increasing complexity of ICs, strongly triggered by the proliferation of consumer electronic devices. Thanks to the developments made in the last decades in the area of very large scale integration (VLSI) technologies, the designers have the means to build multimillion transistor ICs, meeting the needs of an ever-increasing microelectronics market. The design of complex systems-on-a-chip (SoCs) is emerging in telecommunications and multimedia applications, these systems merge analog or mixed-signal (AMS) blocks together with digital processors and memory blocks on the same chip [1][2].

It is known that most functions in today's ICs are implemented using digital or digital signal processing (DSP) circuitry. On the other hand, analog blocks constitute only a small fraction of the components on mixed-signal ICs and SoC designs, being essentially the link between digital circuitry and the continuous-valued external world, so are also integrated on the same die [3]. However, the development time of analog blocks is much higher when compared to the development time of the digital blocks. The two main reasons identified for the larger development cycle of analog blocks are the lack of effective computer-aided-design (CAD) tools for EDA, since analog design is less systematic, more knowledge-intensive and more heuristic in nature than the digital counterpart; and that analog circuits are being integrated using technologies optimized for digital circuits. For this reason, given the rampant growth of AMS systems, the economic pressure for high-quality yet cheap electronic products and time-to-market constraints, there is an urgent need for CAD tools that increase the analog design productivity and improve the quality of resulting ICs [4].

Analog ICs are known for its difficult re-utilization, so designers have been replacing analog circuit functions for digital computations, but there are some typical blocks referred as remaining analog forever [3]:

- On the input side of a system, signals from a sensor, microphone or antenna, must be sensed or received, amplified and filtered up to a level that allows digitalization with satisfying signal-to-noise and distortion ratio. Typical application of these circuits is in sensor interfaces, telecommunication receivers or sound recording;
- On the output side of a system, the signal from digital processing must be reconverted to analog and it has to be strengthened, so that it can drive outside load with low distortion. These circuits are used, e.g., in telecommunication transmitters and loudspeakers;
- Mixed-signal circuits like sample-and-hold, analog-to-digital converters, phase-locked loops and frequency synthesizers. These blocks establish the interface between input/output sides of a system and digital processing parts of a SoC;
- Voltage/current reference circuits and crystal oscillators offer stable and absolute references for the above mentioned circuitry;

- The last block of analog circuits are the extremely high-performance digital circuits. The prime example is state-of-the-art microprocessors that are custom sized like analog circuits, attempting to reach higher speed and lower power consumption.

Today's analog design is supported from circuit simulators, layout editing environments and verification tools, which maintain the design cycle for AMS ICs long and error-prone. These circuits suffer from diverse non-idealities and parasitic disturbances that, by not being weighted in the early stages of development, can be responsible for design errors and expensive re-design cycles, becoming the bottleneck of SoC and mixed-signal ICs design.

In Figure 1-1 is presented the V-Cycle of a design system architecture, which summarizes the differences between analog and digital design automation, in particular, the implementation path (left) and the verification path (right), according to the International Technology Roadmap for Semiconductors [2]. The green arrows indicate available and yellow arrows partially available elements of a state of the art design system environment, future requirements and current unavailable for design automation are indicated in red. A circuit design goes from a system-level description to lower and more detailed levels. In the digital domain, EDA is fairly well developed and establish a low-level design process almost fully automated. The principal lacks in digital design path (purple) of the design system are the tools and methodologies above the behavioral abstraction level. While on the other hand, the analog design path (blue) reveals that EDA is still in the early stages of completely close the V-Cycle, the analog level of automation is far from the "push-button" stage.

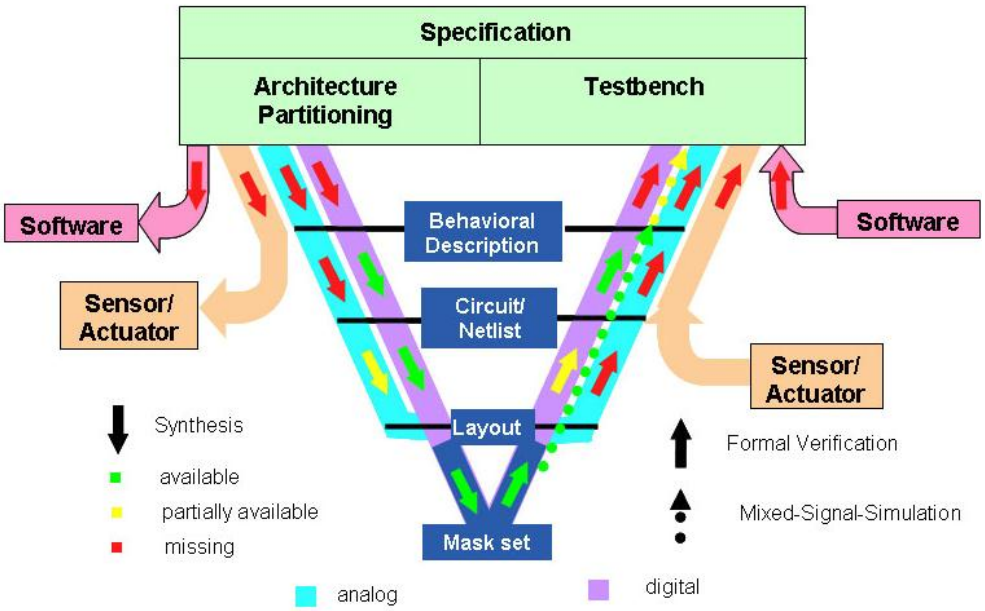


Figure 1-1 – The V-Cycle for design system architecture [2].

In order to understand the automation of the analog design, the steps in the analog design flow must be clear. After this brief introduction to the analog design paradigm, the next section of this chapter covers a systematic approach to the analog design flow proposed by Gielen and Rutenbar [3], which intends to ease design automation.

1.1 The Analog Design Flow

Different analog design flows are available in the literature, however the majority of the works developed in the last decade follow the design flow introduced by Gielen and Rutenbar [3]. This design flow for AMS IC circuits illustrated in Figure 1-2 consists of a series of top-down design steps repeated from the system level to the device-level, and bottom-up layout generation and verification. Adopting a hierarchical top-down design methodology is possible to perform system architectural exploration, obtaining a better overall system optimization at a higher level before starting more detailed implementations at device level. The problems are found early in the design flow and as a result have a higher chance of first-time success, with fewer or no overall time consuming redesign iterations [1]. The number of hierarchy levels depends on the complexity of the system being handled, and the steps between any two hierarchical levels are:

- The top-down electrical synthesis path includes topology selection, specification translation (or circuit sizing at lowest level) and design verification;
- Bottom-up physical synthesis path includes layout generation and detailed design verification (after extraction).

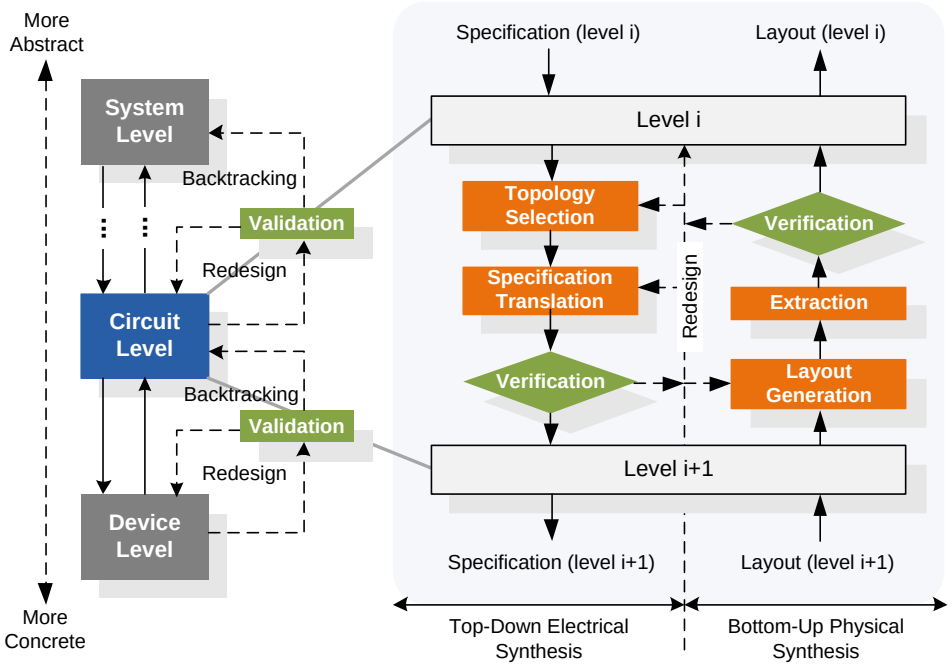


Figure 1-2 – Hierarchical level and design tasks of design flow architectures.

Topology selection is the step of determining the most appropriate circuit topology in order to meet a set of given specifications of the current hierarchy level. This topology can be chosen out of a set of available topologies, or created a new one.

Specification translation is then the step of mapping the high-level specifications for the selected topology block under design into individual specifications for each of the sub-blocks, at the lowest level the sub blocks are single devices and this task is reduced to circuit sizing. Specifications translation is

verified by means of simulations before proceeding down in the hierarchy. Since no device-level sizing is available at higher levels, behavioral simulations are needed, and electrical simulations are used at the lowest level in the design hierarchy. The specifications for each of the blocks are passed to the next level of the hierarchy and the process is repeated until the top-down flow is completed. Some recent works based on Pareto optimal fronts (POF) have been very successful exploring the tradeoff during synthesis [7], and already applied at system level sizing. In this approach, a set of non-dominated solutions are generated and the suitable solution is selected from the POF.

Several CAD tools, settled through the years in the industry, are fundamental to help the designer to successfully complete this task. They are used for IC design editing and evaluation, some of the tools available are: ADiT, Questa, Eldo [8]; HSPICE, nanosim, HSim [9]; Spectre [10]; ngspice [11] and SMASH [12].

Layout generation consists of creating the geometrical layout of the block under design at the lowest level in the design hierarchy, or place and route the layouts of the sub-blocks at higher levels. In the presented design flow, it is important to notice the presence of a detailed verification step over the extraction of the layout. In order to ascend to higher hierarchical levels is necessary that no potential problems are detected at the lowest levels and the layout meet the target requirements. When the topmost level verification is complete, the system is designed.

Some CAD tools are available for layout edition, e.g., IC Station Layout [8]; Galaxy Custom Designer LE [9] and Virtuoso Layout Editor [10]. Design rule verification and layout extraction can be performed for example in CALIBRE [8]; Hercules [9] and DIVA, Assura [10].

1.2 Motivation

In digital IC design several EDA tools and design methodologies are available to help the designers keeping up with the new capabilities offered by the integration technologies, while analog design automation tools are not keeping up with the new challenges created by technological evolution. This is one of the reasons why analog design is many technology nodes behind leading-edge digital [2][3]. Due to the lack of automation, analog designers keep exploring manually the solution space searching for a solution that fulfills the design specification. This method causes longer design times and allied to the non-reusable nature of analog IC design, make it a cumbersome task. After many years of stagnation due to heavy investment in digital domain, the once-sleepy analog design automation market is now evolving.

The methodology presented in this report focuses on the layout synthesis task of analog circuits, appointed as the critical part of the analog design flow [3][5]. In the last years a lot of works have emerged from universities, some of them even found their way into commercial EDA tools. Still, the available tools are far from perfect and lots of problems remain unsolved [4]. From an industrial point of view, the application of EDA tools in analog layout synthesis is still far away from being a reality.

The onset of more efficient and user-oriented tools is mandatory in order to boost analog designers' productivity and ease this time-consuming task.

Generally the complexity of designing analog circuits' layout it is not due to the number of devices, but from the countless interactions between them. Plus, for smaller technology nodes with the increasing complexity of design rules and physical effects these interactions even have a greater impact. Automated analog design tools should concentrate on settling an analog-specific layout synthesis process and do not currently take into account all precision matching needs for such designs [2].

1.3 Goals

The previous LAYGEN [13] implementation is used as starting point of the present work, which was intended to describe a methodology for automatic analog ICs layout generation, through the introduction of an abstraction level between technological details and the designer guidelines. The abstract layout template captures the designer knowledge independently of technology. The approach focuses on improving design reusability and retargetability once the template is available, introducing a new level of flexibility by supporting changes on device and modules specifications.

Design productivity is increased only if the target layout can be automatically generated in a process guided by the designer, and the result validated with a commercial tool, assuring the quality of the solution, which was not done in the previous implementation. Technology design rules are quite prohibitive, which forces reconsidering the whole previous approach. The main objective of the current implementation is to make the tool robust enough for industrial grade validation, performed in Mentor Graphics' Calibre® [8] DRC tool. Moreover, the internal module generator should generate parametric devices with equivalent quality to the commercial tools' parametric cells.

Fulfilling the restrictive design rules isn't the only objective of the actual implementation, but also improve the layout quality. The designer provides the high level floorplan and the tool instantiates and places the devices in the layout ensuring that the design rules are strictly respected. Automatic abutment and biasing considerations are mandatory.

A new and more versatile implementation of the router that only requires connectivity to generate DRC clean routing solutions must be developed, promoting automatic routing generation independently from the floorplan, and consequently allowing topological exploration. Simultaneously, it must allow the designer to introduce constraints of symmetry, sensitivity and power nets. In order to evaluate each layout solution without using an external tool, it is necessary to develop a powerful but lightweight internal validation procedure, as reliable as a commercial DRC tool.

These are the goals of the proposed methodology, and the robustness and retargetability characteristics of LAYGEN II will certainly justify its implementation. LAYGEN II does not intent to replace the designer in the layout generation task, but rather use the designer knowledge to perform an intelligent pruning of the design space, rapidly providing a solution to be used as a first cut design. Handmade designs are known for their robustness, and the tool provides the means for the designer

easily integrate his knowledge and lighten the efforts to accomplish layout design task, abstractly from the complex technological details. At this stage, the tool is still in development and the generated layouts do not intend to be competitive with layouts done by an experienced designer.

1.4 Achievements

The following achievements were obtained during the development of the methodology proposed in this dissertation:

- R. Martins, N. Lourenço, and N. Horta, “LAYGEN – Automatic Analog ICs Layout Generator based on a Template Approach”, *Genetic and Evolutionary Computation Conference (GECCO)*, Philadelphia, USA, 7-11 Jul 2012.

- N. Lourenço, R. Martins, M. Barros, and N. Horta, "Analog Circuit Design based on Robust POFs using an Enhanced MOEA with SVM Models," *Chapter in Analog/RF and Mixed-Signal Circuit Systematic Design*, Mourad Fakhfakh, Esteban Tlelo-Cuautle, Rafael Castro-Lopez, Springer, 2012 (estimated).

- "*Generating Analog IC Layouts with LAYGEN II,*" Master thesis accepted to be published in *SpringerBriefs in Computational Inteligence,* Springer, 2012 (estimated).

- Automatically generated fully dynamic Comparators sent for integration in a UMC 130 nm run, courtesy of CTS-UNINOVA.

1.5 Document Structure

This document is organized as follows:

- Chapter 2 presents a study of the available tools for layout design automation. Valuable information can be gathered from them, such as algorithms and supporting data structures.
- Chapter 3 introduces to the proposed automatic flow for analog IC design and particularly the general description of the proposed methodology providing more detail about its implementation, inputs and interfaces used by designer to generate and visualize the target layout.
- Chapter 4 describes the placer and the methods to place the modules in the floorplan, depicting each task implemented by the template-based approach.
- Chapter 5 sketches the router with emphasis on the evolutionary computational techniques used, along with details of the internal evaluation procedure used to depict if the layout solutions fulfill all the technology design rules and constraints.
- Chapter 6 presents the developed framework of the proposed methodology, addressing two case studies and illustrating the capabilities of the implementation.
- Chapter 7 shows the closing remarks and future directions for the continuous development of LAYGEN II are outlined.

Chapter 2 State-of-the-Art on Analog Layout Automation

In the past few years, several tools for the automation of the analog IC cell and system layout design, with application on both new and reused designs have emerged. Yet, most of the layout design is still handmade, this happens essentially because analog designers want to have total control over the different design options, and also, the current fully automated generators of analog IC layouts produce solutions that are no match for manually crafted ones.

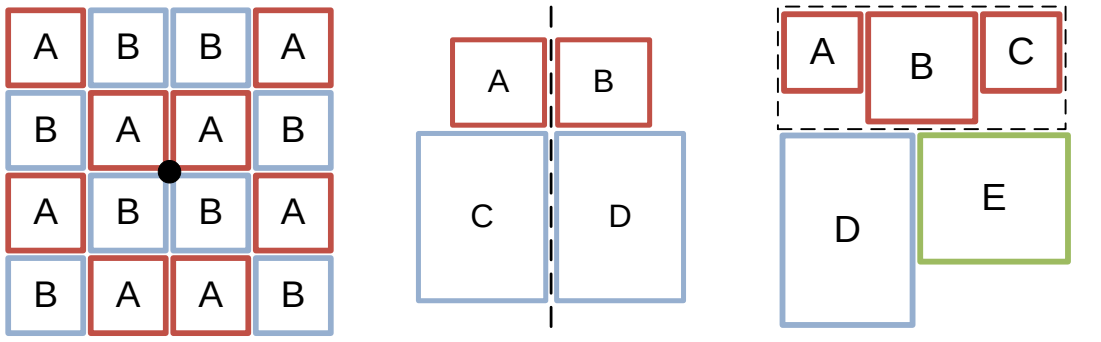
This chapter addresses various important concepts in understanding the developed work. The state-of-the-art on analog layout automation that follows reveals that after many years of stagnation, EDA market is evolving, creating more efficient and complementary approaches to the existing tools. In the next section the placement problem in EDA, providing a brief overview of the most recent placement tools developed, is addressed. In the following section the main references of automatic layout generation tools are presented. Then the recent advances in layout-aware analog synthesis approaches are discussed. Finally, the available commercial solutions for analog layout automation are outlined and the conclusions sketched.

2.1 Placement

Having the devices for the selected topology sized, they must be laid out in the chip, a common analog layout approach is to split the problem into two smaller problems, placement and routing. An automatic placement tool should produce analog device-level layouts similar in density and performance to the high-quality manual layouts. In order to achieve this, the capabilities to deal with layout constraints are mandatory.

2.1.1 Layout Constraints

To reduce the unwanted impact of parasitic, process variations, different operating conditions and improve the circuit performance, many topological constraints have been introduced into analog placement. The major topological constraints for analog placement are device matching, device symmetry and device proximity [5], as presented in Figure 2-1.



(a) Matching (common centroid). **(b) Symmetry.** **(c) Proximity (guard ring/well).**

Figure 2-1 - Analog layout constraints.

The symmetry constraint restricts devices to a mirrored placement, its used to offset geometric and electrical issues, and helps reducing the sensitivity to on-die thermal gradients and parasitic mismatches between two identical signal flows. The matching constraint forces a common gate orientation, common centroid or an interdigitized placement among devices, which improves the beneficial effects of the symmetry constraint by reducing the effect of process-induced mismatches. The proximity constraint limits devices to a specific placement so they can share a common substrate/well region, be surrounded by a common guard ring or place closely matched devices. Principally, it decreases the effect of substrate coupling, and also avoids mismatch and deviations during the fabrication process [5][14].

2.1.2 Chip Floorplan Representations

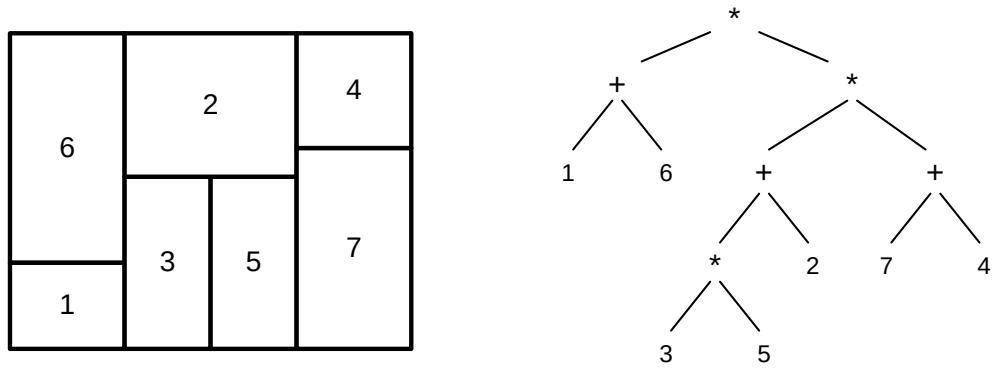
Each placement tool has its own strategy of representing the cells, two main different approaches to the chip floorplan representation have been used in the last years [15]: absolute representation (cells are represented by means of absolute coordinates) and topological representation (encoding the positioning relations between any pair of cells), the last one is further classified into slicing or non-slicing representations.

Absolute coordinates is the typical chip floorplan representation used by many CAD tools to solve device-level placement problems in analog layout, given the nature of this approach a huge search space is explored. This type of representation allows illegal overlaps during the moves (e.g., translations, changes of orientation), since no restriction is made referring to the relative position of a cell with respect to another cell. To circumvent this situation, a penalty cost term is associated with the total infeasible overlaps, and this penalty must be driven to zero in the, generally, simulated annealing (SA)-based [16] optimization engine [15]. The main disadvantages of using the absolute representation are the high running times, due to a larger number of moves necessary. It can generate low-quality and not physically achievable placement solutions, and also the need of an increased tuning effort due to the difficulty of predicting an appropriate weight for the overlap penalty.

Topological representations trade off a smaller number of moves for more complex-to-build feasible layouts. The first class of topological representation is the slicing model, where cells are organized in sets of slices which recursively bisect the layout horizontally and vertically. The direction and nesting of the slices can be recorded in a slicing tree or, equivalently, in a normalized Polish expression. The typical simulated annealing-based optimizer does not move cells explicitly, as it does when it operates with absolute layout representation. Instead, it alters the relative positions of cells by modifying the slicing tree or normalized Polish expression encoding the layout [5].

Figure 2-2 represents a slicing structure, which is obtained by recursively cut rectangles into smaller ones, the corresponding oriented rooted binary tree (slicing tree) is also presented. Each internal node of the tree is labeled with multiplication or plus symbol, respectively, vertical or horizontal cut [17]. Since not all the layout topologies have a slicing structure, this representation can degrade the density of the placement solutions, which is more remarkable when the cells of a layout are very different in aspect ratio, a common situation in analog circuits. Also, symmetry and matching constraints have to

be implemented in the cost function through the use of virtual symmetry axes, which is a less efficient solution.



(a) Slicing structure. **(b) Slicing Tree**

Figure 2-2 – Slicing example [17].

The weaknesses identified in the slicing model made it a bad choice for placement tools oriented to high-performance analog layout design, culminated in the emergence of several non-slicing topological representations. For these models, the degradation of layout density is no longer a matter of concern [15][18].

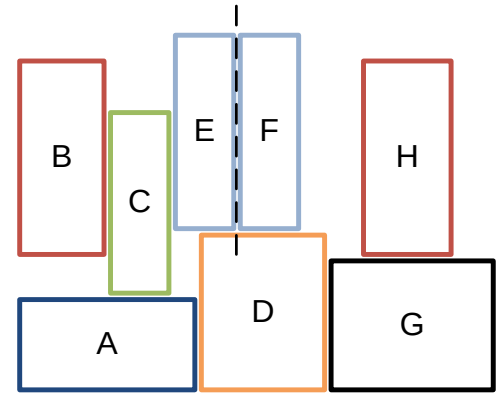
Within the set of non-slicing structures available nowadays one of the most popular is the sequence pair (SP), which encode the “left-right” and “up-down” positioning relations between cells [19], and the solution space can be effectively explored employing SA or genetic algorithms (GA). Symmetry and device matching constraints can be easily handled, and has a $O(n^2)$ complexity, where n is the number of placeable cells.

The bounded-sliceline grid (BSG) [20] also has a $O(n^2)$ complexity, it uses a meta-grid structure without physical dimensions, but introduces orthogonal relations of “right-of” and “above” unique for each pair of cells. Nevertheless, it poses a more intuitive packing that the sequence pair, although, exists a SP that is unconditionally mapped to an optimal packing, while in the BSG this is now always true and its support of symmetry constrains have not been proved yet.

The ordered tree (O-tree) [21] extended the binary tree to the representation of non-slicing structures, presenting a complexity even smaller than in the slicing floorplan. This method was presented to reduce the drawback of redundancies from SP and BSG representations and also, it needs fewer bits to describe the number of blocks than those methods. The run-time for transforming O-tree to its representing placement is linear to the number of blocks ($O(n)$), so one instance of O-tree will map into exactly one placement, no need for extra computation effort.

An efficient upgraded representation of binary trees, *B*-tree*, is also available, offers a $O(n \log n)$ packing for a binary-tree structure that supports cost evaluation, with no need for additional constraint graphs for cost computation, while the other methods above require them [22]. The correspondence between an admissible placement and its induced B*-tree is one-to-one, so no redundancy, with

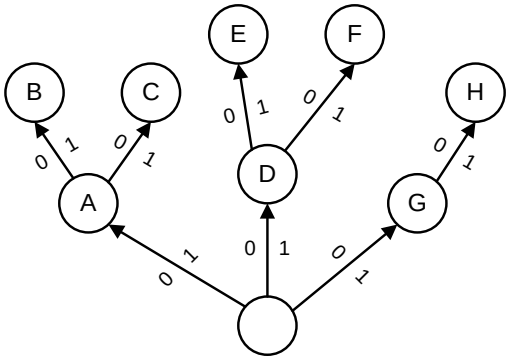
support for symmetry constraints. In Figure 2-3 an example of a placement with a symmetry constraint is presented, and their respective representation in SP, O-tree and B*-tree encodings.



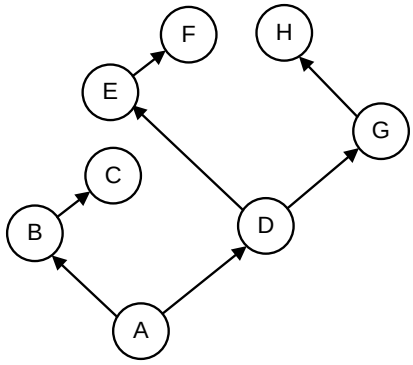
(a) Placement example.

{BCEAFDHG, ABCDEFGH}

(b) Sequence-pair.



(c) O-tree.



(d) B*-tree.

Figure 2-3 – Example of topological layout representations.

More recently, Lin et al. [23] introduced the concept of symmetry island, which keeps modules of the same symmetry group connected to each other. To model a specific placement within a symmetry island a structure based on the B*-tree is used, called Automatically Symmetric Feasible B*-tree (ASF-B*-tree), which also explores symmetry constraints in two dimensions, unlike the other approaches. The principal task of this algorithm is performed on a structure, a Hierarchical B*-tree (HB*-tree), to simultaneously optimize the placement with both symmetry islands and non-symmetry modules, and dynamically update the shape for the devices in a symmetry island. HB*trees are hierarchical oriented, although recent, already proved is high layout quality and runtime efficiency.

A transitive closure graph-based (TCG) [24] representation was proposed to combine the advantages of SP, BSG and B*-tree representations, guaranteeing a unique feasible packing for each representation and that doesn't need to construct additional constraint graphs for the cost evaluation during packing. This approach was quickly replaced for TCG-S [25] derived directly from TGC combined with SP, it presents the fast packing characteristic of SP while maintaining the TCG flexibility to handle placement with special constraints.

In Table 2-1 a summary of the advantages and disadvantages identified for each of the representations above is presented.

Table 2-1 – Classification of chip floorplan representations.

Representation			Advantages	Disadvantages
Absolute			Easier and quicker-to-build layout configurations, every possible placement can be described; Easiness of modeling positioning constraints.	Slow, solution space infinitely large, trade off a high number of moves; Allow illegal overlaps; Solutions not always physically achievable.
Topological	**Slicing**	Slicing Tree / Polish Expr. [17]	Smaller solution space; Moves are modifications of the placement code, altering the relative positions of the cells; Cells cannot overlap illegally, which lead to an improved efficiency.	Symmetry and matching constraints are difficult to maintain; Not all layout topologies have a slicing structure, this representation degrade the density of the solutions.
	Non-slicing		All the advantages presented for the slicing representation; No degradation of layout density.	
		Sequence Pair [19]	A placement configuration can be derived from any encoding; Handles symmetry and device matching constraints.	$O(n^2)$ complexity; $O(n \log n)$ extra effort for the computation to construct the placement from a SP;
		BSG [20]	More intuitive packing that the sequence pair.	$O(n^2)$ complexity; Cannot always represent the optimal packing for a determined group of cells; Deficient support of constrains.
		O-tree [21]	Smaller complexity and less redundancy than SP and BSG, also fewer bits needed to describe the number of blocks; Transforming O-tree to its representing placement is linear, $O(n)$ effort.	Less flexible than SP and BSG in representation; Tree structure is irregular, and thus some primitive tree operations (e.g., search, insertion) are not efficient.
		B*-tree [22]	Upgrades the O-tree both in processing as in efficiency; Smaller encoding cost; No need for additional constraint graphs;	Less flexible than SP and BSG in representation.
		HB*-tree [23]	Ability to handle symmetry constraints in 2D with Symmetry Islands and ASF-B*-trees; Possibility to combine Symmetry Islands with the rest of the modules.	
		TCG-S [25]	Combine the advantages of SP, BSG and B*-tree; Best area utilization, faster convergence speed; Flexibility to handle placement with special constraints.	Despite improvements, the evaluation complexity is still quadratic.

2.1.3 Approaches

Over the years different placement tools have explored the advantages of several chip floorplan representations and new ways of treating layout constraints, these tools had been integrated with more or less success in analog synthesis tools. Next are presented some standalone placement tools, developed recently, that somehow present new solutions or significant developments in the classic placement techniques. A more complete background of analog synthesis tools and respective placement approaches, are referred to the section 2.2 of this chapter.

In the area of device-level topological placement with layout constraints, Balasa et al. [26] presented an algorithm based on the exploration of symmetric-feasible SPs, a symmetry group corresponds to a subset of cells which them all share a common symmetry axis. This approach is powered by a $O(G.n.log\ n)$ complexity for each code evaluation. Koda et al. [27] created an improved method of symmetric placement, obtaining a constraint graph and a set of linear constraint expressions directly from SP. The later placement process is accomplished by linear programming.

In the past recent years, developments are being made in hierarchical placement with layout constraints, Lin et al. [28] were the first to present an algorithm for analog placement based on hierarchical module clustering, using HB*-trees. It deals with different constraints simultaneously and hierarchically, this is, if two or more devices are intended to satisfy one or more constraints, they are formed as a cluster, and these clustering constraints can be hierarchically specified to include other clusters. Interesting features for hierarchical symmetry and hierarchical proximity groups that often appear in analog circuits.

In a time dominated by the optimization algorithms, a full deterministic approach arises, Plantage [14], which is based on a hierarchically bounded enumeration of basic building blocks, using B*-trees. This approach is based on the principle that analog circuits show a hierarchical structure, so that hierarchy is used as a bound for the enumeration, aware that a complete enumeration of all possible placements is impracticable. The algorithm begins by generating all placements of the basic modules (leaf nodes of the hierarchy tree), then the results of the enumerations are combined, guided by the hierarchy tree, until a POF of placements with different aspect ratios for the whole circuit is obtained. Enhanced shape functions are used to store and combine modules efficiently, these functions consist of an ordered set of shapes which are classified through the process by aspect ratio and redundancy, modules considered suboptimal are removed for the sake of computational effort.

In a different direction from the other emerging works, Lin et al. [29] proposed a thermal-driven analog placement solution, to simultaneously optimize the placements of “power” and “non-power” (devices which consume much less power than those classified as “power”) devices, in an attempt to annihilate thermally-induced mismatches. It is known that the thermal impact from “power” devices can affect the electrical characteristics of the other thermally-sensitive modules, degrading AMS ICs performance. A thermal profile for a given circuit for better thermal matching of the matched devices is established, and the algorithm evolves until the desired thermal profile is achieved. This thermal profile consists

essentially in the even distribution of the heat for the whole chip and since the temperature at each point in the placement area must be known at each iteration.

Summary and further specifications of placement tools are presented in Table A-1 of Appendix A.

2.2 Layout Generation Tools

In this section will be reviewed some of the milestones in the analog layout generation, along with some recent tools. The earliest approaches, procedural module generation techniques coded the entire layout of a circuit in a software tool, which would generate the target layout for the parameters attained during sizing. This parametric representation of the layout is fully developed by the designer, either by a procedural language or a graphical user interface. ALSYN [30] employs fast procedural algorithms that are controlled through a database structures and attributes. A high-functionality pCell library independent of technologies can be found in [31]. Although fast processing, these methods may lack of flexibility, since the cost of introducing a new design task is relatively high and technology migrations may force complete cells redesign.

The use of template approaches, which define the relative position and interconnection of devices, is a common practice. A template-based generation is used by IPRAIL (Intellectual Property Reuse-based Analog IC Layout) [32] to automatically extract the knowledge embedded in an already made layout, and use it for retargeting. Layout retargeting is the process of generating a layout from an existing layout. The main target is to conserve most of the design choices and knowledge of the source design, while migrating it another given technology, update specifications or attempt to optimize the old design [5].

In order to retain the knowledge of the designer but without forcing an implicit definition, LAYGEN [13] uses a template-based approach to guide the layout generation. ALADIN [33] also allow designers to integrate their knowledge into the synthesis process. While ALG [34] uses the same knowledge-based principle, allowing the designer to interact with the tool in different phases, leaving to the discretion of the designer if the final layout is obtained almost full automatically or by designer directives. Zhang et al. [35] developed a tool that that automatically conducts performance-constrained parasitic-aware retargeting and optimization of analog layouts. Performance sensitivities with respect to layout parasitics are first determined, and then the algorithm applies a sensitivity model to control parasitic-related layout geometries, by directly constructing a set of performance constraints subject to maximum performance deviation due to parasitics.

The optimization-based layout generation approaches consist of synthesizing the layout solution using optimization techniques according to some cost functions, with a higher level of abstraction. The simulated annealing and genetic algorithms proved to be the most effective choice for solving analog device-level placement problems, beyond their flexibility in terms of incremental addition of new functionalities, they are relatively easy to implement [18]. In the area of device-level placement with

layout constraints there are some main references important to review. ILAC [36] uses simulated annealing operating over a topological slicing tree, used to limit the search space.

However, representing the cells by means of absolute coordinates proved to be the most practical solution to implement layout constraints, even though it allows for an infinitely large solution space. This is the approach found in KOAN/ANALGRAM II [37], LAYLA [38], Malavasi et al. [39] and ALDAC [40]. These methods are usually slow and not always produce optimal solutions in terms of area and performance.

In Table 2-2, a classification of the analog tools presented in this section based on generation techniques is presented, a summary of the vantages and disadvantages of each technique is also highlighted. A summary of the description and functional specifications of the referred tools is presented on Table A-2 of Appendix A, while on Table A-3 of Appendix A technical specifications and few observations are reported.

Table 2-2 – Classification of analog tools based on generation techniques.

	Layout Tool
Procedural	**ALSYN** [30]; **Jingnan** [31]. (+) Fast processing basic cells; (-) Lack of flexibility, technology migrations force complete cells redesign; high cost of the generation task.
Template	**IPRAIL** [32]; **LAYGEN** [13]; **ALADIN** [33]; **ALG** [34]; **Zhang** [35]. (+) Places modules in a short period of time; higher level than procedural; useful for small adjustments like technology migrations; (-) Still limits the search space; designer must add knowledge.
Optimization	**ILAC** [36]; **KOAN/ANAGRAM II** [37]; **LAYLA** [38]; **Malavasi** [39]; **ALDAC** [40]. (+) Higher level of abstraction; (-) Slow; not always optimal solutions in terms of area and performance.

2.3 Closing the Gap Between Electrical and Physical Design

In analog IC design, iterations between electrical and physical synthesis to counterbalance layout-induced performance degradations need to be avoided as much as possible [5][41]. The performance of a circuit need to be guaranteed pos-layout in the presence of layout parasitics, which prevent the circuit from realizing the estimated optimal values. This is usually achieved through time-consuming and unsystematic iterations between the electrical and physical design phases. One possible solution involves the integration of these two different phases, by including layout induced effects right into the electrical synthesis phase, or sizing at device-level.

This methodology can be found in recent literature with different designations like parasitic-aware, layout-aware and layout-driven synthesis (or sizing). This is a complex and hard to measure process, since there are geometric requirements whose effects on the resulting parasitics are very specific, so they are never included in the traditional electrical synthesis task. If predicted early in the synthesis process, the overestimation of layout-induced parasitics results in wasted power and area, while underestimation is very risky because may lead to completely malfunction [18]. Knowing the layout induced effects in the synthesis process ensures that performance solutions are attained, and the area minimization is done in a more realistic manner.

2.3.1 Layout-Aware Sizing Approaches

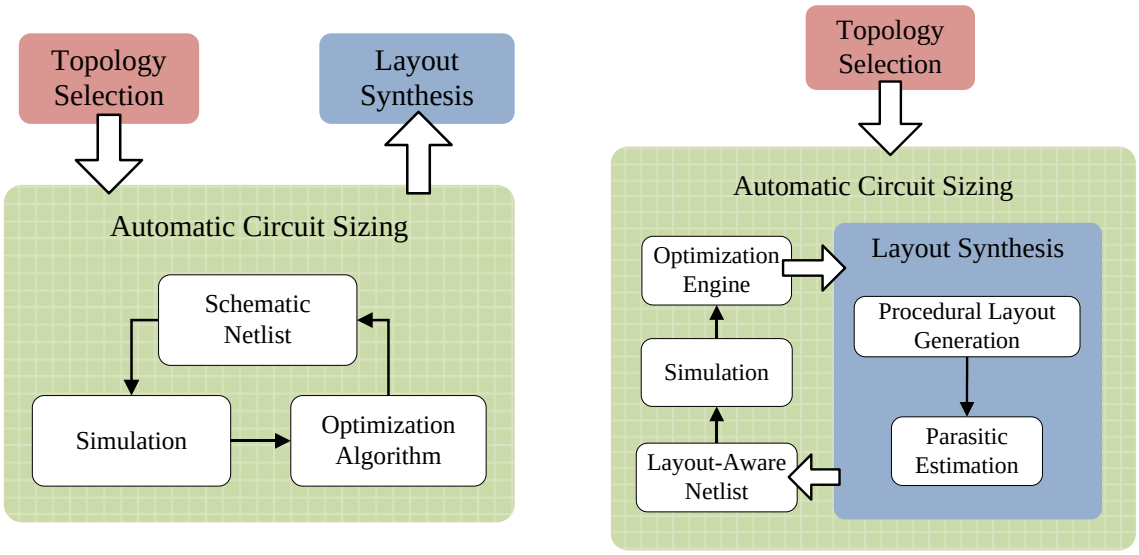
Layout-aware synthesis tools target a design process that avoids time consuming iterations, by bringing layout-related data into the sizing process, aware that layout generation at each iteration is an expensive process. Device parasitics include the bulk capacitances of analog devices, while non-device parasitics include area and coupling capacitances, also dependent of the final routing achieved. Next are presented some state-of-the-art layout-aware synthesis tools and their different ways of extracting layout-parasitics.

Initially, due to the fast processing of basic cells, procedural-based layout generation techniques were used. Vancorenland [42] used manually derived equations along with a procedural layout generation approach to find a suitable solution. Ranjan et al. [43] generates a parameterized layout using the module specification language system, it consists on a fixed template layout, which when provided with the circuit parameters produces a physical layout. Then, the extracted parasitic from the layout, along with the passive component values are passed to the precompiled symbolic performance models (symbolic equations in terms of circuit parameters), which predicts the circuit performance at each iteration avoiding numerical simulations.

Without actually generate a layout, Pradhan et al. [44] obtains a Pareto-optimal surface with good spread of points for conflicting performance objectives, and each solution contains the specific layout induced effects. The design space is initially sampled to generate circuit matrix models, which predict circuit performances at each iteration. For a uniform random number of design points, layout samples are generated by a procedural layout generator and device parasitics are modeled by linear regression.

Unlike the previous approaches, Castro-Lopez et al. [41] tackles both parasitic-aware and geometrically constrained sizing, which not only includes device parasitic-aware sizing, but also a solution for optimized area and shape. To bypass prohibitively long times for layout generation at each iteration, this approach supports on templates implemented by using the Cadence's pCells technology and SKILL programming. Since the predefined template is supported by a slicing tree and the block placement is fixed, area and shape optimization are obtained by finding the number of fingers of MOS transistors that yield optimal geometric features, or introduced as a constraint to obtain for example, area minimization or a certain aspect ratio. A parasitic estimation without layout generation has been equally implemented, using template sampling techniques and analytical equations.

Recently, Habal et al. [45] ruled out the use of templates given the few degrees of freedom they offer, investigating every possible layout for each device in the circuit using placement algorithm Plantage [14] (Section 2.1.3). The layouts with the best geometric features are kept, and only the final placement selected based on aspect ratio, area and electrical performance is routed. Designer knowledge is supplied by geometric circuit placement and routing constraints, then a deterministic nonlinear optimization algorithm is used for circuit sizing. In Figure 2-4, a traditional analog design flow with emphasis on circuit sizing is presented, versus the generalized layout-aware methodology proposed in [45].



(a) Traditional analog design flow. **(b) Layout-aware circuit sizing.**

Figure 2-4 – Traditional versus layout-aware circuit sizing flow [45].

Summary and further specifications of layout-aware sizing tools are remitted to Table A-4 of Appendix A.

2.4 Commercial Tools

Recently, some commercial solutions have emerged in the analog layout EDA market. Ciranova Helix™ [46] presents as a placement manager supported by a powerful and easy to use graphical user interface (GUI). The designer introduces the system hierarchy and each of the sub-blocks can be added independently from the remainder. This perspective is useful on an on-going system-level specifications translation, since the parasitics from the available blocks and estimated areas can be provided for the designer to optimize the circuit. For the automatic placement, the designer provides a set of constraints for a given circuit schematic and the tool automatically presents a set of possible minimum-spacing layout alternatives for that block. The tool explores the possible combinations deterministically. It produces DRC correct placement solutions for design rules at the nano-cmos design processes. The output is a standard OpenAccess database that can be edited in most of the layout editors. The general flow of Ciranova Helix™ is presented in Figure 2-5.

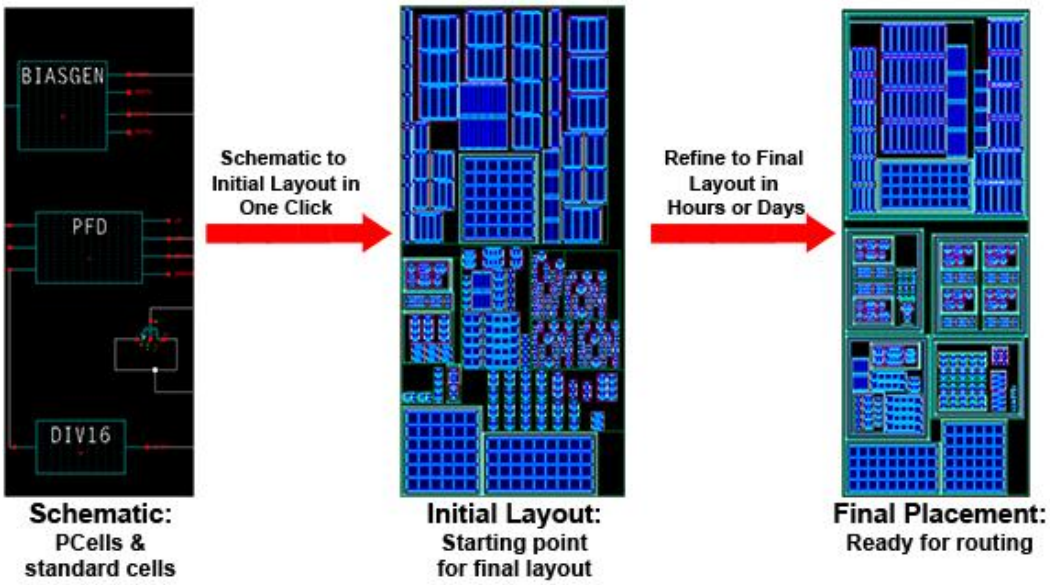


Figure 2-5 – Ciranova Helix™ general flow [46].

In Mentor Graphics IRoute and ARouter [8] the designer knowledge is used to manage the routing generation. The designer manually chooses the order in which the nets are routed, and the nets with a higher priority are routed more directly. The wires' width and spacing to other nets must be selected, and also, the designer sets the specific conductor to be used in each net and the transition points between layers. The tool provides markers and directions given the set of actual constraints, to interactively help the designer to manually draw the wires.

Calibre® YieldAnalyser [8] integrates process variability analysis using model-based algorithms, that automatically plug layout measurements into yield-related equations to identify the areas of the design that have higher sensitivity to process variations. Critical areas can be mathematically weighted by yield impact information to prioritize and trade-off the issues that have the biggest impact on chip yield. YieldAnalyzer performs critical area analysis on all interconnect layers to identify the areas of a layout with excess vulnerability to random particle defects. The tool runs analysis directly on most of the layout data files, e.g., GDSII, OASIS and OpenAccess design databases, and the information is presented in reports and graphs within the designer's layout environment.

Virtuoso® Layout Suite Family [10] eases the creation and navigation through complex designs, supported by a sturdy multi-window GUI with automatic assistants to aid the designer. These designers' directions guide the physical implementation process while managing multiple levels of design abstractions at device, cell, block, and chip levels, focusing on precision-crafting their designs without sacrificing time to repetitive manual tasks. The suite contains different levels of assistance: basic design-creation and implementation environment; assisted correct-by-construction wire-editing functionalities ensuring real time process-design–rule correctness; captures and drives common hierarchical design intent from schematic editor; and a set of advanced automated finishing tools to optimize the layout and rapidly realizing first time successful silicon.

Translates analog cell schematics into a full-custom optimized layout and capture the subtle electrical and geometrical constraints to enable analog layout reuse.

Tanner EDA [47] HiPer DevGen presents as a smart generator to accelerate the generation of standard cells. The tool analyzes the netlist and recognizes the current mirrors and differential pairs, and then automatically sends them to the generators. Designers have the control over generation options, layout, placement, and routing of these structures. For differential pairs there are multiple options to ensure matching, optimized parasitic, add dummy devices, guard rings, antenna effect diodes, etc. For current mirrors there are multiple outputs of different current strengths, options to ensure matching, add dummy devices, share diffusion, multiple finger with options for gate and bulk connections, and adjustments for well proximity effects. To configure a new technology only the design rules are required to present DRC and LVS clean standard cells. The generated primitives intend to be similar to those handcrafted. The Hyper DevGen GUI is presented in Figure 2-6.

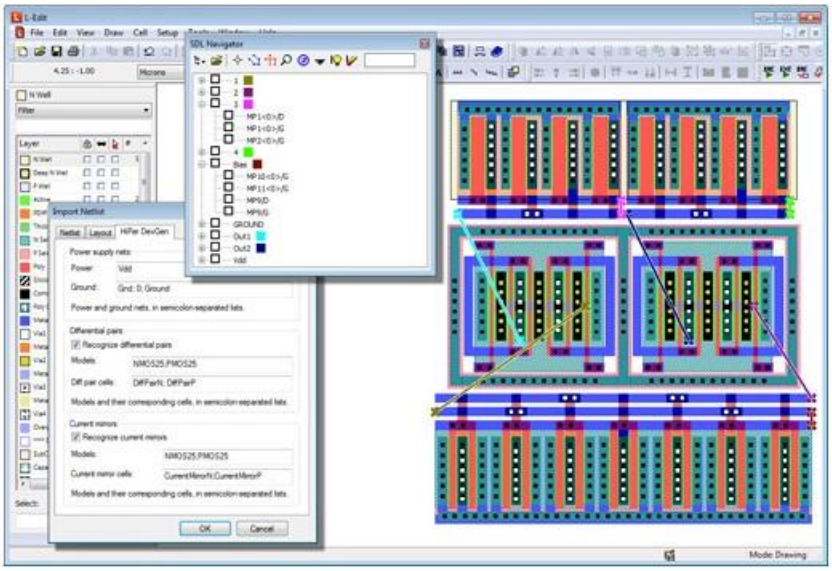


Figure 2-6 – Tanner EDA HiPer DevGen GUI [47].

2.5 Conclusions

In this chapter a set of tools applied to analog IC design automation were presented, with emphasis on the layout task. Although much has been accomplished in automatic analog layout generation, the fact is that automatic generators aren't yet used in the industrial design environment. Reviewed approaches are usually limited to some specific circuit or circuit class, and given the huge time required to create a new tool or prepare an existing one to support a new circuit, causes that analog designers continue to develop layouts manually.

Beyond the efforts made towards the generation of a full automatic layout, capable of competing with expert designers' handmade layouts, it is possible to notice that EDA tools are moving in a different direction from two decades ago. There is now a strong attempt to recycle the existing layouts, migrating them to new technologies or optimize the old design. Many of the circuits manufactured today are the same ones developed and implemented years ago, so it is extremely important to take

advantage of the knowledge embedded in the layout and follow the advances in the integration technologies, instead of going through all the design process once again.

At the same time, layout-aware sizing methodologies are spreading and represent an important part of the future of analog design automation, closing the gap between electrical and physical design for a unified synthesis process. Fast, flexible and as complete as possible layout generation techniques are required to include layout-related data into the sizing process, or eventually obtain a final layout simultaneously with the sizing. Most of the layout-aware solutions rely on procedural layout generations, which are known for their difficult reuse and lack of flexibility. The solution of [45] although avoids procedural generations, the computational times required to complete the automatic flow are superior to a dozen of hours.

Figure 2-7 establishes a chronological representation of the tools presented in this chapter, organized by the generation technique used.

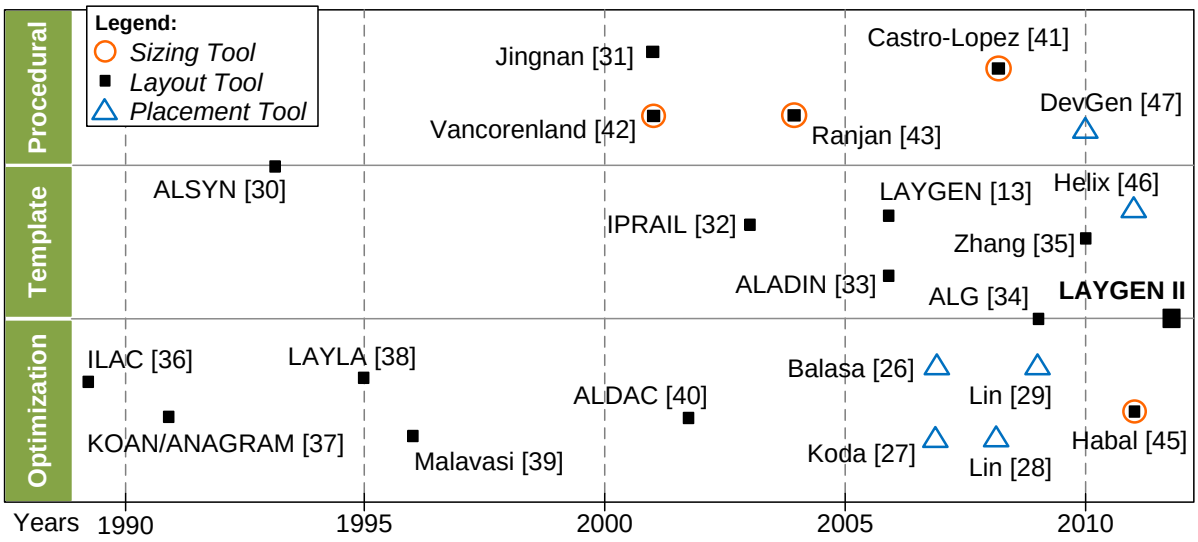


Figure 2-7 – Chronological representation of analog design tools.

The commercial tools presented, proved that only the approaches that allow for designers to integrate their knowledge into the synthesis process and offer control over the generation, found their way into the EDA market. Most of the available commercial solutions stand out because of the powerful GUIs provided and their characteristics as layout task managers, but lacking on the algorithmic complexity for automatic generation. These tools are used to speed up the manual design task by means of interactive and assisted-edition functionalities.

From the reviewed approaches, it is possible to notice that floorplan design automation, although far from perfect, are keeping up relatively well with the challenges imposed by new integration technologies. However, the routing task of the proceeding is where the most of the difficulties remain. This is clear when observing the limitations of the current approaches, and the completely lack of routing automation in commercial EDA.

The idea of parameterized model/template is present in the most recent successful approaches. Increasing the designer's active part in generation can't be seen as a drawback, since the inclusion of his knowledge to guide the process increases the layout quality, and consequently the automatic generation always presents a satisfying solution for the designer. LAYGEN II also allows for the designers to integrate their knowledge into the synthesis process, creating an abstraction layer between technological details and the designer guidelines. This design definition is inherently technology independent and allow for changes in circuit's specifications using the same template.

As mentioned, the previous LAYGEN [13] implementation is used as starting point of the present work. The general specifications of the previous and current implementation, LAYGEN II, are depicted in Table 2-3. Detailed implementation of the specifications outlined for placer, router and validation are presented through this document, in the proper sections.

Table 2-3 – General specifications: LAYGEN versus LAYGEN II.

	LAYGEN	LAYGEN II
Placer	**Template-based placer:** ▪ Macro-cell placer without overlap; ▪ Optimization-kernel used to decide the optimal combination from a set of different layout representations; ▪ B*-tree representation; ▪ Sets a fixed distance between devices;	**LAYGEN's placer, featuring:** ▪ Biasing considerations; ▪ Automatic merging; ▪ Places devices at the minimum distances allowed by target technology; ▪ Polygon guard rings;
Router	**Template-based router:** ▪ Fixed geometry and preferred layer; ▪ Fixed pins connecting; ▪ Limited geometric and layer shifting operators.	**Optimization-based router:** ▪ MOEA modified NSGA-II; ▪ Greedy initialization (pins, heuristics and layers); ▪ Geometric and layer shifting operators; ▪ Multiple pin search; ▪ Multiple contacts; ▪ Power lines; ▪ Symmetric wires; ▪ Noisy/Sensitive nets; ▪ Out of bounding box exploration.
Validation	▪ Internal short circuit checker.	▪ Multi-thread internal evaluation procedure: short circuit checker, design rule checker and electrical rule checker; ▪ Industrial grade parametric module generators; ▪ Calibre® DRC validation; ▪ Industrial validation up to 130 nm processes.
Other	▪ Graphical user interface oriented for visualization; ▪ The output provided is a GDSII stream format.	

Chapter 3 **Automatic Layout Generation**

This chapter starts with the description of the proposed automatic flow for analog IC design, with emphasis on the layout generation task of the proceeding. The following section provides a general description of the layout generation flow using LAYGEN II, referring in the further sections of this chapter more detail about its implementation, inputs and interfaces used by designer to generate and visualize the target layout.

3.1 Design Flow based on Automatic Generation

It is acknowledged that each designer/company has its own layout style but often this style is very regular for a large number of applications, even with some specifications or technological changes. The design guidelines for most common cells are kept the same. For simple cells, parametric generators are a valid solution to implement these guidelines, however, parametric generators are specific to a technology making them difficult to reuse. In addition, for cells that are more complex, the development of effective parametric generators has proven ineffective either on design-time or on design-reusability.

In order to cope with these limitations and increase design efficiency, LAYGEN II stores these design regularities in a layout meta-description that is independent of technology and parameters obtained during specification translation task. The template, together with LAYGEN II and a set of parametric module generators at device-level provide the designer with a technological and specification independent way of defining some of the most commonly used cells.

The generation starts from the coarse layout definition described in the template, and finishes at the optimized target layout. In addition, since technological details are treated automatically by LAYGEN II, the designer's efforts are focused on difficult layout issues and not on the technological dimensions details and so forth. This way, designer's efforts are better-used, increasing design productivity, efficiency and reusability.

As stated, this methodology introduces an abstraction level between the designer expertise and the technology details, which introduces some changes in the typical IC design flow. The proposed automatic design flow for analog ICs is shown in Figure 3-1. This work addresses how the guidelines are provided to LAYGEN II and the methods used to automatically generate the layout of analog ICs. Although the tool it's being developed to be integrated in the bottom-up physical synthesis path of an analog design automation process, it fits equally well as a standalone tool to be used by designers in the layout generation task.

The flow is triggered by a specification translation at system level or a sizing task at circuit level. When a set of device sizes is obtained for the desired specifications this information is included in the selected template, which encompasses the guidelines of the designer. Then layout is automatically

generated by LAYGEN II, being the remainder LVS and extraction steps necessary in order to perform post-layout simulation, to validate the intended specifications obtained during sizing task. If the specifications are not met re-design is necessary. The LVS validation verifies if a particular IC layout corresponds to the electrical schematic, this step is only performed if a successful DRC for the automatically generated layout was previously obtained. DRC validation is implied to layout generation task, as it will be shown later in this chapter with LAYGEN II's architecture. Although outside the scope of this work, the sizing task is briefly covered in the next subsection.

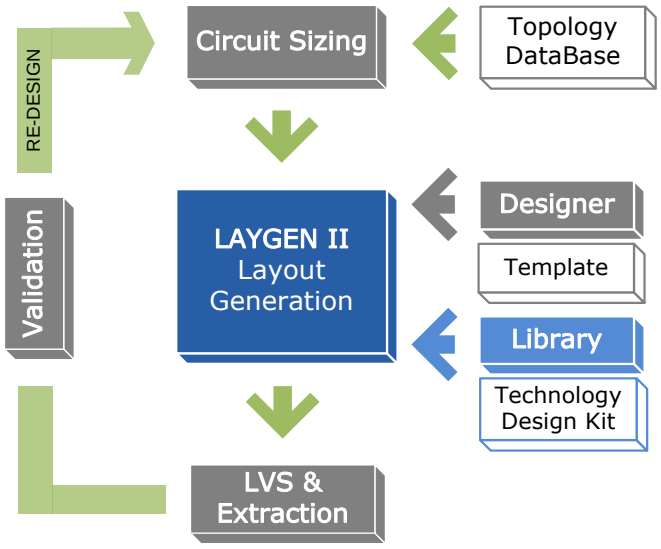


Figure 3-1 – Analog IC design flow: Closed loop.

3.1.1 Sizing Task

Before layout, the circuit must be sized. The designer must devise an architecture suitable to implement the system specifications and then all the blocks. From the high-level specifications to the devices sizes must be properly dimensioned. In industry this task is commonly done manually, the designers start by finding an approximate solution using simplified analytical expressions, and then iteratively, adjust the solution until it meets all specifications.

Another possibility for circuit sizing is the use of circuits' synthesizers. These tools are used to automate the circuit sizing and can be equation-based, i.e., the methods use analytic design equations to evaluate the circuit performance; numerical-simulation-based techniques that do not require the previous modeling, but present higher execution times than equation based due to the verification done using electrical simulations; and numerical-model-based tools that use macro models, like neural-networks or support vector machines to evaluate the circuit's performance. A general architecture for optimization based tools for automated circuit sizing is presented in Figure 3-2.

One of the critical problems in analog IC design is the process variability, i.e., devices designed to be equal are different after production. To verify if the design is robust enough special analysis techniques have to be employed, ensuring that the vast majority of the fabricated circuits will work according to specifications. The most common techniques for analog design centering are Monte Carlo simulation and Corner analysis. Monte Carlo technique executes many simulations by applying

random variations to the circuit's and process' parameters, making a stochastic sampling of the behavior of the circuit in real world conditions. Corner analysis is a worst-case approach where the circuit is simulated over multiple combinations of process parameters variations (power supply, temperature, etc.). Sizing must end with some sort of centering analysis to ensure design robustness to technological deviations, using these techniques the impact of, e.g., technology gradients and environmental conditions on circuit's performance can be balanced in the design.

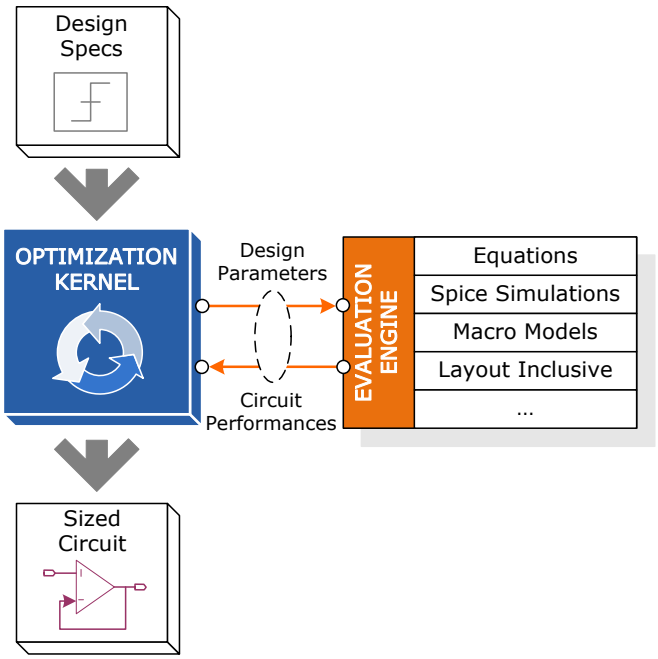


Figure 3-2 – Optimization based sizing.

During sizing task the designer should attempt to reduce the impact of process variations by applying robust circuit design techniques. For the complementary metal oxide silicon (CMOS) processes, a common technique consists of reducing the source and drain area of the devices by the use of interdigitized transistors. Reducing as minimum as possible the area of the diode source-substrate, or drain-substrate, will procure minimum parasitic coupling and minimum leakage current. Although matching is generally important, it's particularly essential in the design of current mirrors and differential pairs. Bad matching usually produces high offset so should be used layouts that optimize matching, which is achieved by providing the best symmetrical conditions possible. Transistors with different orientation match badly, moreover, a circuit can suffer mismatch even if the current in the transistors is flowing in opposite directions [48].

For the analog design flow of Figure 3-1, the problem of automatic specification translation at circuit level, circuit sizing, is performed by an in-house tool, GENOM-POF [49, 52]. This tool is based on an elitist multi-objective evolutionary optimization kernel [53] and uses an industrial grade simulator HSPICE® [9], to evaluate the performance of the design. GENOM-POF targets the design of robust circuits by allowing the consideration of corner cases during optimization. The inputs are the circuit netlist, testbench, the definition of the optimization variables, design constraints, objectives and the

corner cases. The output is a POF of the sized circuit that fulfills all the constraints and represents the feasible tradeoffs between the different optimization objectives.

3.2 Layout Generation Design Flow

LAYGEN II can support designer as a standalone tool in the iterative process of layout generation. The layout design flow using the tool as a low-level generator is illustrated in Figure 3-3. Taking into account that the desired technology design kit is already available, providing the high level floorplan, devices sizes and connectivity between modules the layout can be automatically generated.

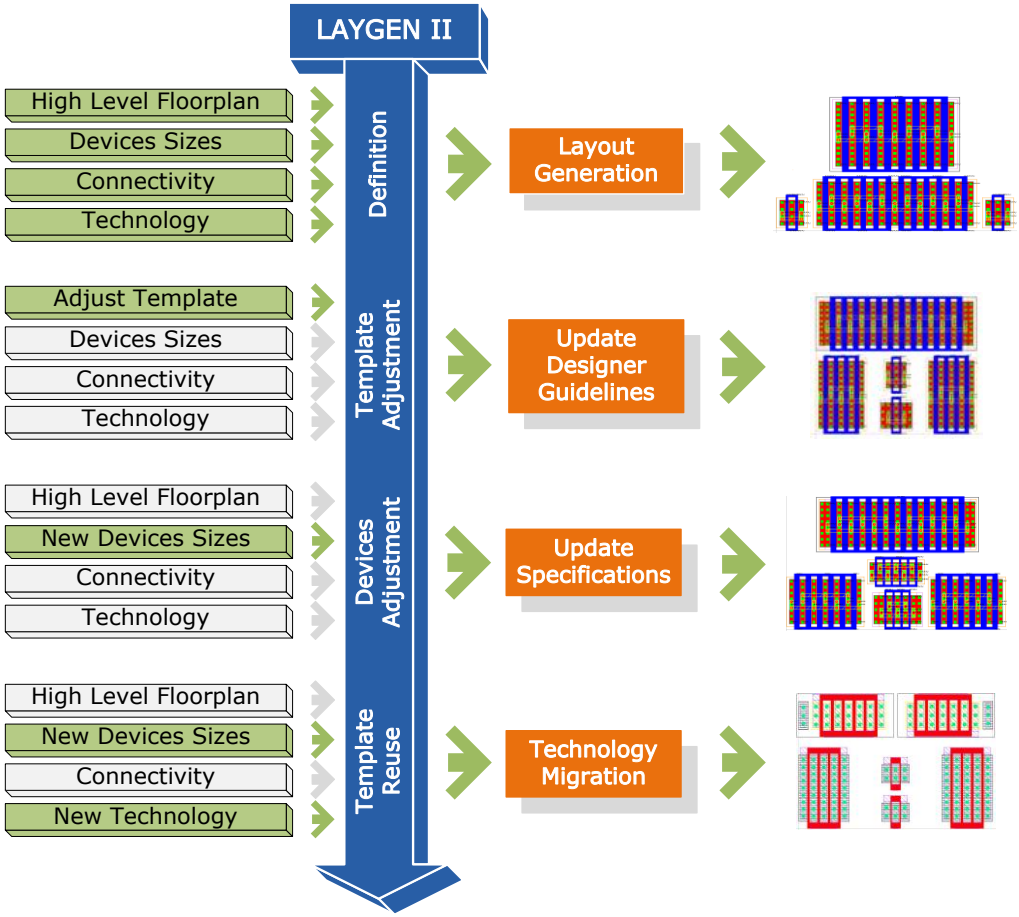


Figure 3-3 – Analog layout design flow using LAYGEN II.

After the first generation, the result can be iteratively re-generated by applying adjustments to the topological relations between cells in the high level floorplan provided in the template. This way, the designer can control the generation from a higher level and easily introduce new guidelines, until the desired solution is obtained for the current set of devices sizes.

Although in a typical manual IC design process the layout generation task is always performed after the specification translation, often designers may have to introduce some changes in the project even after the layout is concluded. Those changes may occur by different causes, for example, from a late adjustment to the previous specifications that consequently result in new devices sizes, or even from a sub-block replacement in system level design. In the manual process of designing layouts this

redesign may lead to partial or complete loss of the previous work, reflected in the total time necessary to obtain a satisfying solution. In LAYGEN II the information is reused because the guidelines and connectivity are kept in the layout meta-description.

The same template can be used to retarget a circuit for completely different specifications, being at the discretion of the designer if the old guidelines are still satisfactory for the new design. Although the template may require some adjustments, these changes in the high level floorplan take an almost negligible time when compared to a complete manual re-design. The reuse of old designs is a common practice of analog designers, both for technology retargeting, using the previous guidelines to generate the circuit for a different technology, as for reutilization of some circuit blocks in a system level design.

In summary, the input parameters can be iteratively changed by the designer, updating high level floorplan guidelines, devices sizes or choose new technologies to obtain the exact desired layout. This way, the designer is now producing technology and sizing independent layout descriptions that can be used for retargeting. The previous design may then be reused with few or no changes, yielding therefore a highly reusable and efficient design methodology.

3.3 Tool Architecture

The proposed functional architecture is shown in Figure 3-4 that depicts the principal tasks performed by LAYGEN II. The designer defines a template yielding a high-level circuit description, featuring placement and routing guidelines. The template, the sized components and the target technology design kit, that comprises the design rules and a set of parametric module generators, are used by the tool to automatically deal with the exact placing and routing. This automatic generation is performed attending the specific design rules of the target technology and the device sizes specific to the target application. The generation proceeds in the traditional way, first placement and then routing, lastly a final DRC validation step is performed. This architecture follows a hybrid solution of correct-by-construction and optimization-based methodology.

The output provided is a GDSII stream format, a file standard in the microelectronics industry for data exchange of IC layout. GDSII is a binary file format composed of several structures that are hierarchically related and a set of elements for each structure, a detailed description is presented in the Appendix B. The physical validation of the result is performed in Mentor Graphics' Calibre® DRC tool, a main reference in the ICs design when the development is intended for fabrication.

Placer encompasses all tasks from instantiation until the floorplan is obtained. During the instantiation phase parametric devices are generated using the module generators, custom cells are loaded into the database and sub-templates are generated by the same set of operations of its own modules. When all the device level layouts are available, the placement is performed using the topological guidelines from the template through a set of procedures, automatically merging and ensuring that the

design rules are fulfilled, which will improve the floorplan solution. The placer follows a template-based approach and the details of this stage are dealt in Chapter 4 of this document.

In turn, router uses the obtained floorplan as the starting point. Then, the connectivity and a set of constraints of symmetry and sensitivity are used to guide an evolutionary optimization kernel. In this process may be used multiple sequential executions of optimizations kernels, being that the last execution must result in the detailed and final routing. The routing solution fits in the previously obtained floorplan, ensuring that the technology design rules are strictly respected and fulfills the set of constraints defined by the designer. Although connectivity is provided in the form of template, LAYGEN II's router follows a highly flexible optimization-based generation approach, on which more detail is presented in Chapter 5.

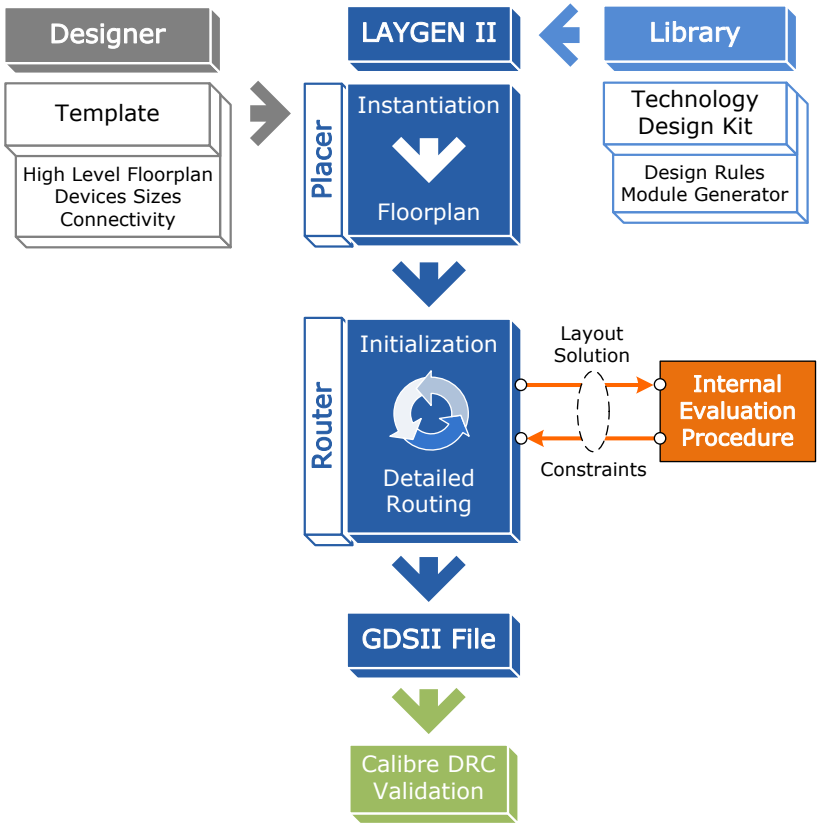


Figure 3-4 – LAYGEN II general architecture and interfaces.

Inside the optimization cycles of router an internal evaluation procedure is used to evaluate each layout solution, mainly because commercial tools' execution times are prohibitive. Still, the parsing of the commercial tool report would mean an extra work when settling a new design kit, deteriorating LAYGEN II's modulation, since a new design kit would require a different parser. The internal evaluator provides LAYGEN II the means required to predict if a layout will be successfully validated and thus guide the optimization. After the detailed routing is obtained and the GDSII generated, there is a step of validation with Calibre® DRC tool to assure that the output complies with the design rules. Nevertheless, the final verification in a sturdy commercial tool is required to ensure industrial level compliance not only with all physical design rules but also electrical-rule checking (ERC).

LAYGEN II has a complete graphical user interface (GUI), so there is no need for the designer to use an external layout design tool to accurately visualize the results. Since GUI doesn't allow editing but only visualization, if the designer intents to manually edit some details over the automatically generated layout, the output GDSII file should be imported to a proper editor, e.g., Virtuoso Layout Editor. The GDSII file provided is an industry standard supported by the majority of technology vendors and compatible with nearly all EDA software, from commercial to free tools.

In the next sub-sections some details are provided for the GUI implemented and the technology design kits supported, along with a brief introduction to the input parameters and hierarchical capabilities of the high level cell description used by LAYGEN II.

3.3.1 Graphical User Interface

LAYGEN II's framework and GUI are implemented in Java™ 1.6, which is platform independent programming language. Though efficiency is an important issue, since automatic layout generation using evolutionary computation techniques is under constant development, it is important to keep the modularity and flexibility of the implementation.

As mentioned the tool offers a visualization engine, in Figure 3-5 is displayed a screenshot of the GUI. This GUI provides a simple and fast way for the designer to check the evolution of the automatic generation. It is also particularly important in the development of the tool, since the impact of introducing new features is easily evaluated in all stages of the generation. Black-box generators are not suited for multiple reporting facilities, and given the graphical nature of layout generation justifies the use of an own interface.

This template representation is quite suitable for graphical display. It is provided a graphical view of the topological relations between cells of the designers' defined templates, this way the errors in the template definition are easy to identify. The template viewer depicted in Figure 3-5 (a) provides scroll and zoom functionalities, launchers for the placer and router are also available.

The layout viewer uses the display settings associated to the layout's technology design kit to define the graphical properties, such as color, drawing pattern and z-axis value for each layer. In Figure 3-5 (b) is presented an automatically generated layout. The layout viewer also implements scroll and zoom functionalities, it provides automatic zoom adjustment to the displaying window when is maximized or resized, and it is also possible to select or unselect a layer for display. The internal evaluation procedure of router automatically places markers in the layout reporting the errors detected, this is essential for the designer to indentify the errors, but also for the developer debug the tool.

The GUI can also be used as GDSII viewer. The files can be imported and a library browser is provided to easily explore the current layout hierarchy. In Figure 3-5 (c) an imported hand-made design and the associated library browser are presented.

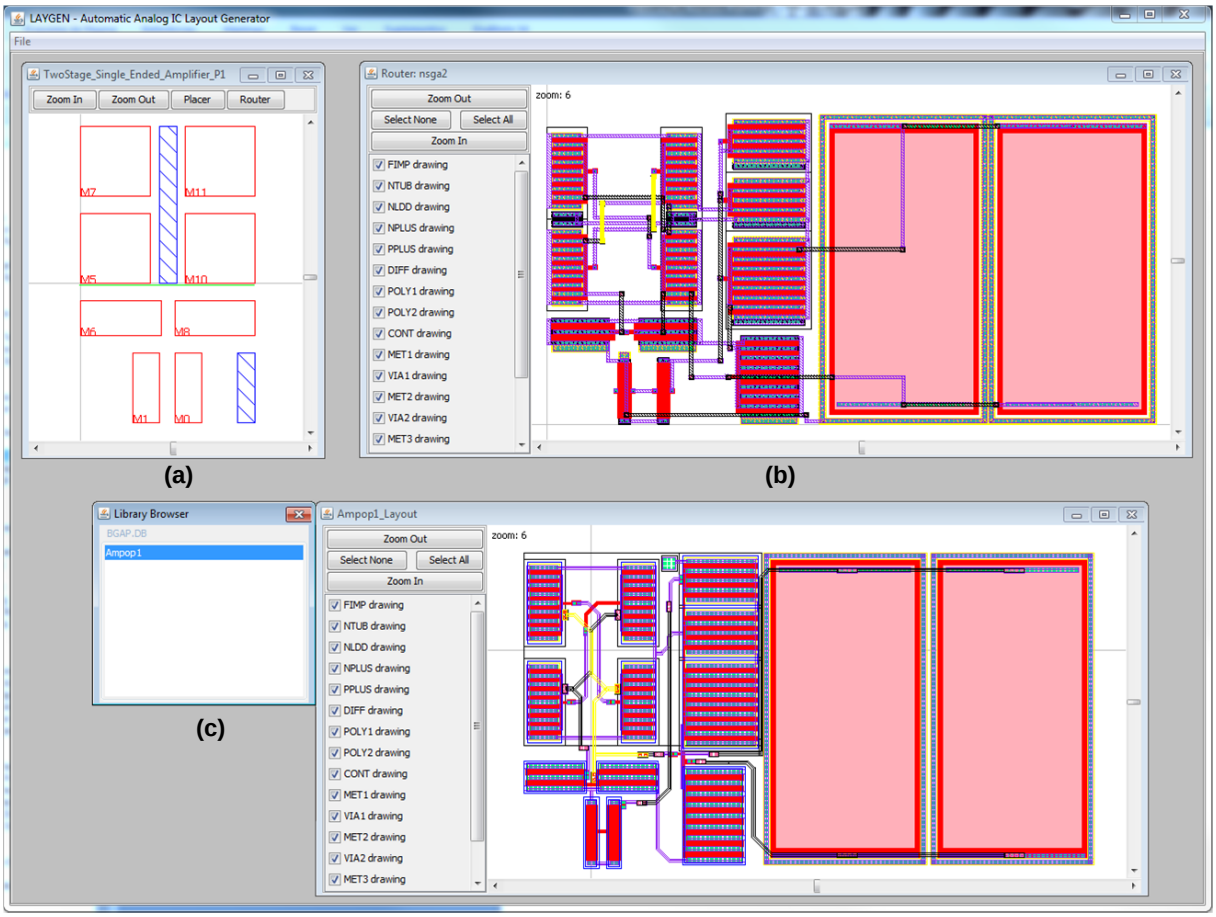


Figure 3-5 – LAYGEN II GUI: (a) Template viewer, (b) Layout viewer and (c) Library browser.

3.3.2 Technology Design Kit

The modules used in the tool can be generated by parametric module generators from the target technology design kit or sub-templates that will be generated during the instantiation phase of the main generation. Custom hand-made layout cells can also be used as modules. However, special handling is required since it is necessary to properly identify the terminals and pins of a cell, in a consistent manner with the current implementation. The use of an industry standard allows the use of previous designs directly as modules whenever suitable. There is no limit on each cell complexity, a cell can be as simple as transistors and as complex as amplifiers, these devices are used as macro-cells by LAYGEN II.

To define an internal technology design kit it is required a layer structure with the enumeration of the layers and vias available; a layer map, containing the GSDII number and type of each layer; the GUI display settings, which describes the colors and patterns used by the layout viewer; the design rules that the technology must comply; and the parametric module generator. The tool was initially developed for a 350 nm process, but in the course of the project that LAYGEN II is part of, the technology design kit was settled to a 130 nm process detailed in Appendix C. This is design kit used over this document to generate the examples and the majority of the test cases.

The parametric module generator is usually used to instantiate the basic available devices. These modules are described in terms of technology design rules in order to facilitate porting between technologies. Even though there might be large differences between technologies, which are more remarkable when changing to a different factory or distant technology nodes, much of the effort spent when defining the parametric modules can be reused when settling a new design kit. This reuse may signify only some hours of work, while defining a new module generator could require some days to perform. At the time of generation or migration, modules should be validated in Calibre® DRC for a large different number of device sizes, this way it is possible to ensure that all parametric modules comply with the design rules.

Each module can have multiple terminals, and each terminal can have any number of non-overlapping pins. These pins represent the exact points to connect the wires and should be inserted in the physical layer where the corresponding shape is. During the instantiation of the modules, after the rotations (if required), the topological labels are assigned to the pins. These labels provide the absolute location and layer information for the pins composing that terminal in the module being generated.

3.3.3 Hierarchical High Level Cell Description

To reduce unwanted impact of parasitic and process variations, many knowledge-intensive constraints have been considered in analog design. Device matching, symmetry and proximity, current density in interconnects and thermal effects are just some of the factors that analog designers have to consider while planning an analog layout project. While designer's expertise is essential in this phase, this type of knowledge does not require being aware of the particular details of a given technology, e.g., minimum distances or enclosures allowed between layers, etc.

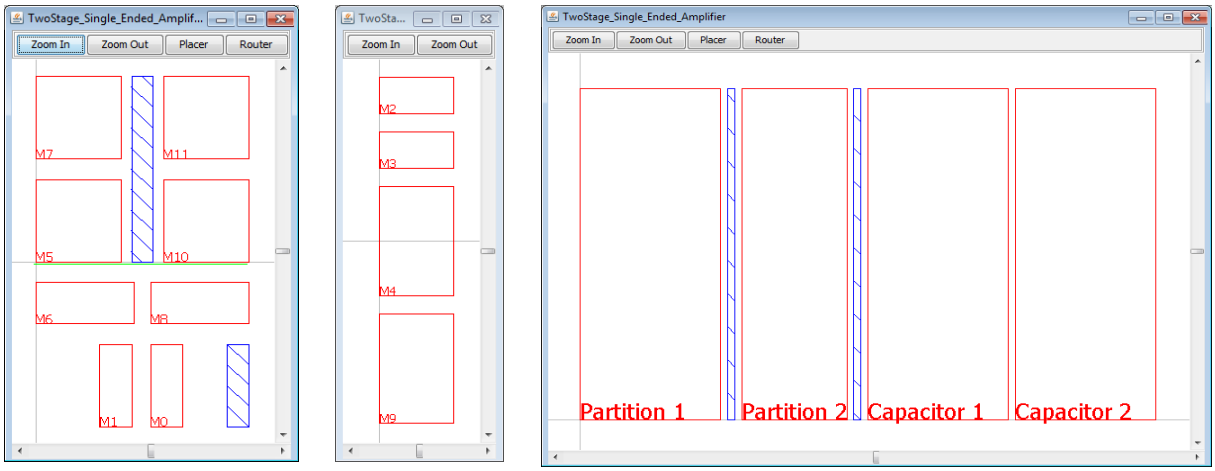
Designer's expertise is caught into the technology independent template and used to guide the automatic layout generation. These guidelines will provide to LAYGEN II the necessary information to increase the layout quality and achieve a satisfying solution for the designer. The template must allow the tool to generate the target layout in a technology and specification independent way. An extensible markup language (XML) [54] description is used to define the template

The information used for placement is the devices sizes and the high level floorplan contained in the template, namely the topological relations between cells and a set of symmetry and matching requirements. The topological relations are used to generate a B-Tree layout representation, which the extraction and packing procedure is described together with placement in Chapter 4. For routing the designer provides the connectivity between devices and set of symmetry and sensitivity constraints, if desired. This information is used to automatically generate a routing solution that complies with the design rules, in a process detailed in Chapter 5.

Templates can also be used as modules in a hierarchically manner, allowing the designer to use templates for simpler cells in the definition of more complex ones, splitting the complexity of the space search into different executions of the optimization kernels. The hierarchy is explored as a bottom-up approach, where the physical representation of the lowest levels must be available or generated

before proceeding to the automatic generation of the higher ones in the hierarchy. It is also possible to perform routing in the sub-templates from a top template, considering the hierarchy in the description of the connectivity. In Figure 3-6 it is provided an example of a hierarchical template definition, each of the blocks represented may have any number of sub-templates, and so on.

Despite the advantages, the placement constraints enforced in the template, which are defined by the designer, may inhibit the performance of the target layout. For wide specification changes the topological relations between cells may not yield a suitable layout, as some new arrangement of modules may be required. For its part, the connectivity provided for routing doesn't have to be modified through any changes performed in high level floorplan.



(a) Partition 1. **(b) Partition 2.** **(c) Top partition.**

Figure 3-6 – Example of a hierarchical template, partition 1 and 2 are the sub-templates.

3.4 Conclusions

Analog IC layout design is complex and yields long development times. In this chapter the LAYGEN II tool architecture was presented, that aims to reducing the design flow time by the introduction of a design methodology that is inherently technology and specification translation independent. This template description creates an abstraction level between physical representation and designer's knowledge. The designer has a way to control the process at a higher level, being the LAYGEN II responsible for dealing with the exact placement and routing, while attending the specific design rules of the target technology and the device sizes specific to the target application. In addition, the support for hierarchically defined templates allows the designer to define complex cells at expense of simpler ones.

LAYGEN II presents itself as a tool to assist the designer in the generation of layouts and the solution obtained from the intelligent pruning of the design space can be used as a first cut solution. The methodology focuses on the efficiency of retargeting operations, introducing new guidelines, update specifications or migration of designs to different technologies are now tasks easier to perform in a project supported by LAYGEN II. The detailed description of the techniques used for computing placement and routing is found in the next chapters.

Chapter 4 **Placer**

In this chapter the methods used by the placer to process and place the modules in the floorplan following the designer guidelines embedded in the template are presented. The first section addresses the general architecture of the placer. The following section covers the high level floorplan described in the template. Finally, the generation procedure for the floorplan, depicting each task implemented in LAYGEN II's template-based placer is presented.

4.1 Placer Architecture

The proposed generation architecture is shown in Figure 4-1, which depicts the principal tasks performed by the template-based placer. This process is performed in three main stages, instantiation, pre-processing and post-processing. Between these stages are performed steps denoted as packing. Before packing, in the extraction process the topological relations present in the template are mapped to a non-slicing B*-tree layout representation, on which the $O(n\ log\ n)$ packing algorithm presented in [56] is used to obtain a compact placement. In this phase, the important information read from the template are the topological relation between modules and not modules sizes or absolute positioning.

The usage of a template that incorporates expert knowledge about the high level floorplan allows LAYGEN II to always present a satisfying solution for the designer. The placer acts as a macro cell placer without overlap, whose functions are to instantiate and process the modules in different ways, increasing the solution quality but always fulfilling the requirements described in the template. Even though placer doesn't create any nets or wires, it is necessary to have connectivity available in order to trigger the pre-processing tasks detailed in section 4.3.3.

In the previous LAYGEN implementation [13] an optimization-based kernel, mostly using SA, was used to decide the optimal combination of modules that minimized the effective area occupied, or if desired, restricting the obtained placement to a certain aspect ratio. The placer explored the alternative placements by selecting one of the alternative modules for each device and packing the layout. These alternative modules are a set of different layout representations provided in the template, for example, a range of possible number of fingers for the same transistor or different aspect ratios for a capacitor.

Despite the actual LAYGEN II's implementation still supports the SA optimization kernel, after pre-processing, he quickly revealed obsolete. The truth is that designers want to have total control over the devices. For example, the transistor models for electrical simulators consider the number of fingers so they have to be weighted during sizing task, an alternative module may not be electrically equivalent. Since this functionality is no longer used, its implementation is not presented in this document. Furthermore, an entirely new floorplan processing, which is covered in this chapter, is introduced, as summarized in Table 2-3 of Chapter 2.

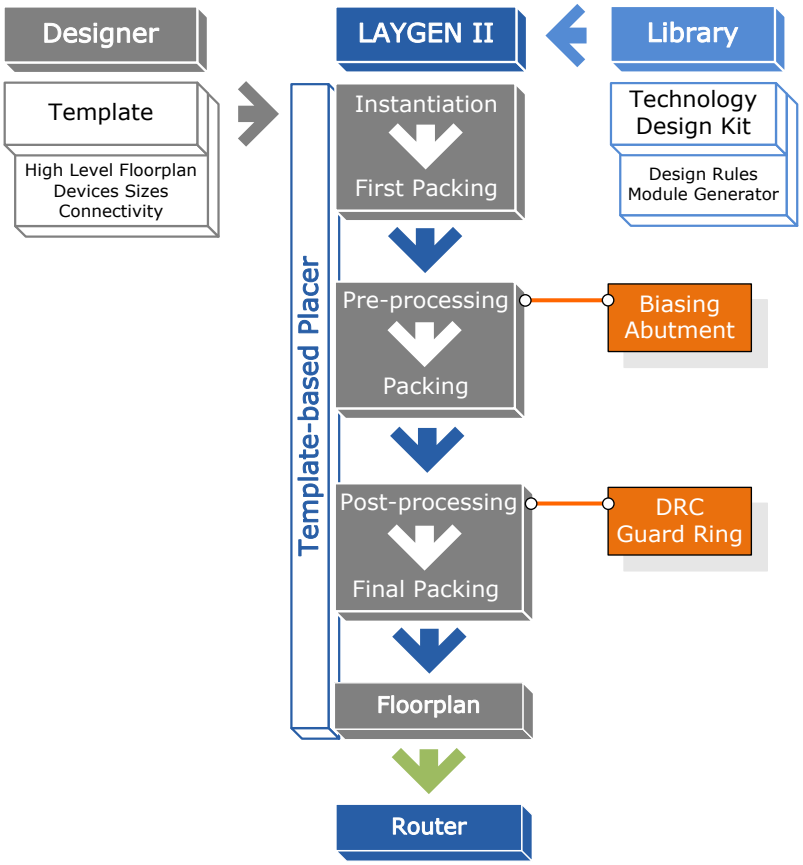
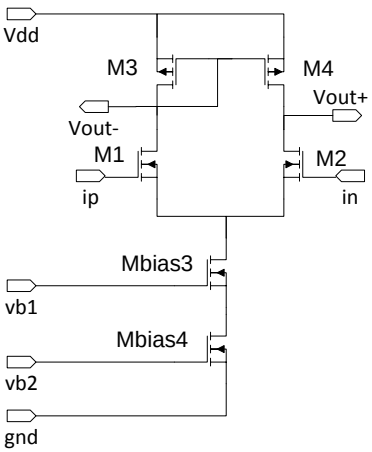


Figure 4-1 – Template-based placer architecture.

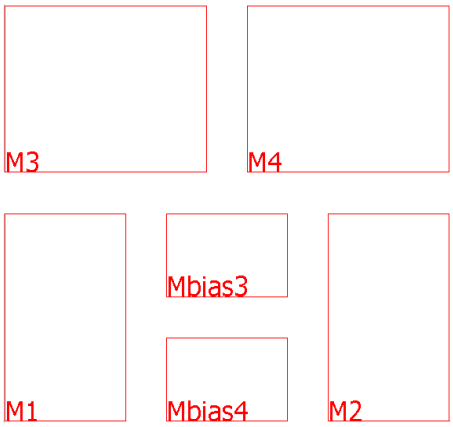
4.2 Template

As mentioned, the template information used for placement is the type and relative placement of the cells; the symmetry and matching requirements; and the devices sizes obtained during a previous sizing task. The floorplan building blocks are the cells described by theirs template bounding-box, the template sizes have no meaning, only the relative positioning regarding the other cells is of concern. Besides the cells' description, the designer also provides other design guidelines, like bulk information and guard rings. Although the template description is relatively long, this task only has to be performed once for the target circuit and all subsequent changes in the designer guidelines result only from small adjustments to the respective fields.

Figure 4-2 (c) presents a possible template description for the layout generation of the differential amplifier with a current mirror load [55] of Figure 4-2 (a). The template view for the provided topological relations between cells is shown in Figure 4-2 (b). This example will be used through this chapter as a demonstration example of the implementations performed by the placer. A previous sizing task was performed by GENOM-POF whose optimization objectives were minimizing the area and power, and maximizing the gain. The first point of the obtained POF, i.e., the sizing solution which minimizes the area was selected to be used in this chapter. The selected sizing solution is presented in Table 4-1 and all the layouts presented over this chapter were generated for the CMOS design process of 130 nm addressed in Appendix C.



(a) Diff-amp with a current mirror load.



(b) Template view.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Template SYSTEM "template2.dtd">
<Template name="Diff_amp_with_current_mirror_load" cell_dir="..\"
        technology="UMC_013">

<CellList>
        <Cell name="M3" symGroupId="1" symCellId="1">
                <Box x="0000" y="3000" w="2500" h="2000" />
                <ParametricCellView device="MOSFET" type="P" bulk="vdd"
                width="17.2" length="0.350" m="6" angle="0" />
                <Match cell="M4"/></Cell>

        <Cell name="M4" symGroupId="1" symCellId="1">
                <Box x="3000" y="3000" w="2500" h="2000" />
                <ParametricCellView device="MOSFET" type="P" bulk="vdd"
                width="17.2" length="0.350" m="6" angle="0" />
                <Match cell="M3"/></Cell>

        <Cell name="M1" symGroupId="1" symCellId="2">
                <Box x="0000" y="0000" w="1500" h="2500" />
                <ParametricCellView device="MOSFET" type="N" bulk="none"
                width="17.6" length="0.270" m="4" angle="0" />
                <Match cell="M2"/></Cell>

        <Cell name="M2" symGroupId="1" symCellId="2">
                <Box x="4000" y="0000" w="1500" h="2500" />
                <ParametricCellView device="MOSFET" type="N" bulk="none"
                width="17.6" length="0.270" m="4" angle="0" />
                <Match cell="M1"/></Cell>

        <Cell name="Mbias3" symGroupId="1" symCellId="-1">
                <Box x="2000" y="1500" w="1500" h="1000" />
                <ParametricCellView device="MOSFET" type="N" bulk="none"
                width="3.4" length="0.120" m="2" angle="0" />
                <Match cell="Mbias4"/></Cell>

        <Cell name="Mbias4" symGroupId="1" symCellId="-1">
                <Box x="2000" y="0000" w="1500" h="1000" />
                <ParametricCellView device="MOSFET" type="N" bulk="gnd"
                width="3.4" length="0.120" m="2" angle="0" />
                <Match cell="Mbias3"/></Cell>
</CellList>
```

(c) Template description.

Figure 4-2 – Template example.

Table 4-1 – Devices sizes and objectives attained during sizing task, using GENOM-POF for a 130 nm process.

Objectives	Devices	Sizes	
		Width	Length
Estimated Area = 4,7286 μm^2 Power = 1,181 mW DC gain = 30,47 dB	M1, M2	17,6 µm	270 nm
	M3, M4	17,2 µm	350 nm
	Mbias3, Mbias4	3,4 µm	120 nm

Symmetry is specified by two properties, symmetry group that defines a set of cells that share the same symmetry axe; and symmetric cell, the cell that is placed symmetrically in relation to the symmetry axe. The topological constraints are inferred from the template placement directly. Matching defines a pair of cells that designer considers that are suited to be matched. As detailed in section 4.3.3.2, in the current implementation matching consists of replacing two cells by a cell with common source or common drain merged, but the same process could be used for interdigitized or common centroid cells. These matching solutions depend only on the parametric modules available on the module generator of the target technology design kit. Bulk information should identify the net which has the desired biasing potential for the device, essential for biasing considerations in a process detailed in section 4.3.3.1.

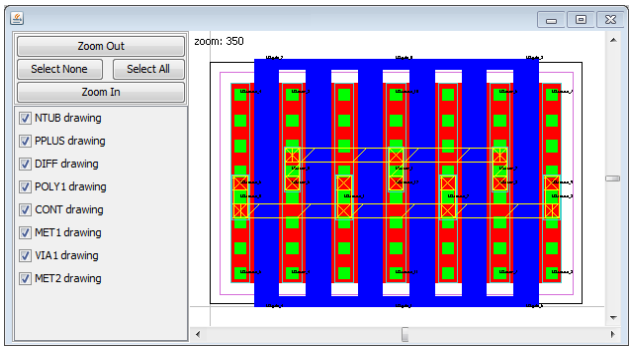
As addressed in section 4.3.4.1 of this chapter, the minimum spacing allowed by target technology is forced between cells in placement ensuring that the technology design rules are strictly respected. However, those distances may prove insufficient space for routing. Therefore, to allow the designer to force some gaps between cells, routing channels can also be defined in the template. The routing channel is treated during placement as any other cell, except that its layout representation is an empty box. The usage of routing channels eliminates routing consideration during placement.

4.3 Template-based Generation Procedure

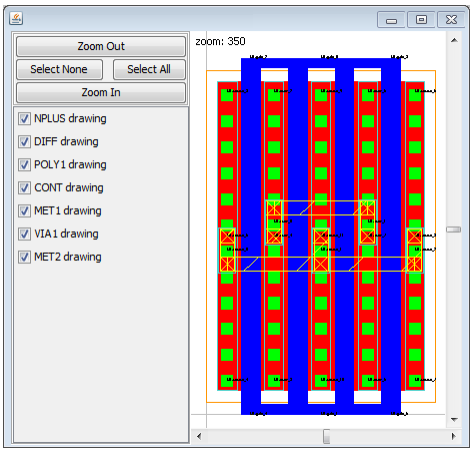
In the next sub-sections, some more detail is provided about the instantiation step of the proceeding and the implementation of the extraction and packing of the non-slicing data structure. Then, the main tasks performed by LAYGEN II's template-based placer, pre and post layout processing.

4.3.1 Instantiation

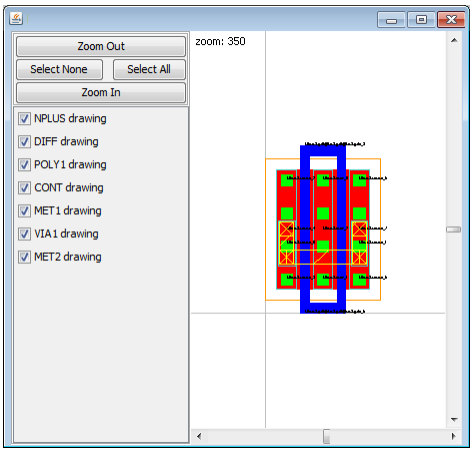
In the instantiation step, all modules contained in the template are substituted by their physical layout representation. The modules can be internal procedural generators from the target technology design kit, sub-templates that will be generated during the instantiation phase of the main generation or custom hand-made layouts imported in GDSII format. All the devices from template example in Figure 4-2 are generated from the parametric module generator of the current 130 nm design kit. The three different modules instantiated are shown in Figure 4-3.



(a) Transistor M3/M4 (W = 17.2 µm, L = 350 nm).



(b) Transistor M1/M2 (W = 17.6 µm, L = 270 nm).



(c) Transistor Mbias3/Mbias4 (W = 3.4 µm, L = 120 nm).

Figure 4-3 – Instantiation of devices in the template example.

4.3.2 B*-Tree Representation

The binary tree representation imposes vertical and horizontal positioning constraints: (a) each device in the left sub-tree is above its parent device and (b) if the y projections of the two devices are overlapping, the device of the node visited first in a pre-order traversal of the tree (visit any node before its left and right sub-trees) is to the left of the device whose node is visited the second [22].

The template provided by the designer is processed to extract the binary tree that encodes the specified constraints. In the extraction procedure, first, all cells are placed in a list, the bottom-left cell is added to the root of the B*-Tree and removed from the list, this procedure is repeated until the list is empty. When the y-projections of the cell being inserted and the cell of the current node are overlapping, if the cell is to the right of the node's cell, the cell is added to the right sub-tree, if the cell is to the left of the node's cell, the cell replaces the node's cell, and the node's cell and the cells in its sub-trees are added to the current sub-tree. When the cell is above the node's cell, there are two scenarios. The cell is above with some x-projection overlapping, in this case the cell is placed in the left sub-tree. Alternatively, the cell is to the right of the current node, in this case there are two possible B*-Tree encoding. The tree is copied, and the cell is placed in the left sub-tree in one copy and in the right sub-tree in the other.

To generate the target placement, the B*-Tree is packed using the sizes of the selected modules. The packing is performed in two steps, the y-coordinate of the cells are calculated in one pre-order transversal of the B-Tree, each cell positioning is set by knowing the position of its left parent. Then, the x-coordinate of each cell is computed using the Red-Black interval tree algorithm [56]. With the y-coordinates already assigned, the modules are placed in the smallest available x-coordinate that do not yield any overlap. Since multiple different B*-trees may be extracted, each one must be packed for the current devices sizes and the one that represents the smallest area is selected.

The layout obtained after the first packing is presented on Figure 4-4. Whenever are performed changes to the template information, a new B*-tree must be extracted before processing packings.

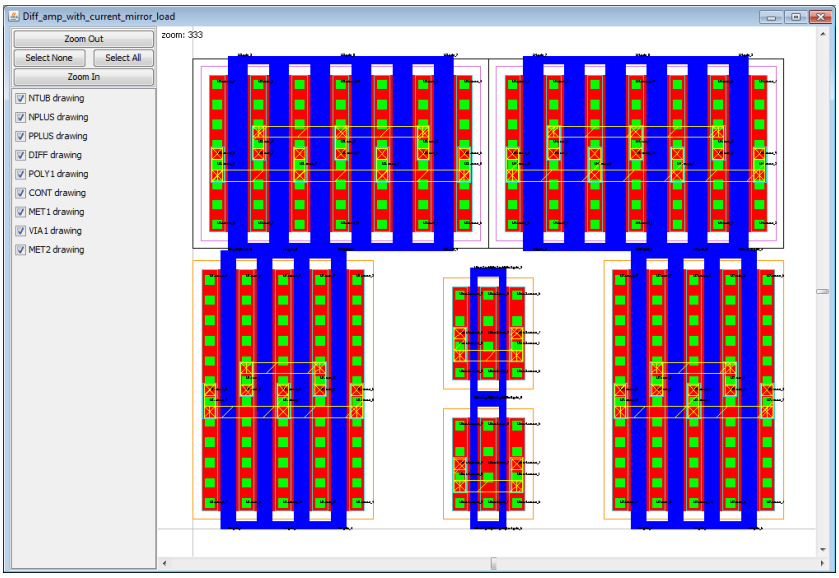


Figure 4-4 - Layout after first packing.

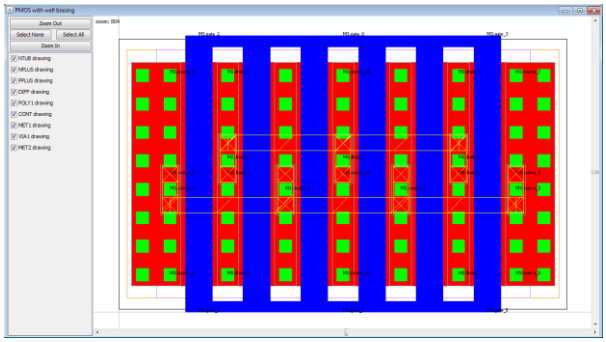
4.3.3 Pre-processing

The pre-processing tasks are performed over the layout obtained after first packing. They treat problems related to the substitution of the current devices from different ones and consequently handling the alterations in the nets. Pre-processing intends to automatically implement common techniques employed by the designers. These techniques are applied only for modules generated from the parametric module generator of the technology design kit, and are not suited for imported GDSII layouts.

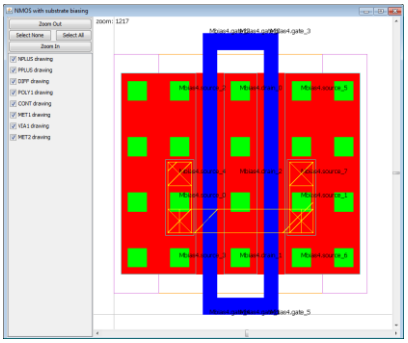
4.3.3.1 Biasing

In a layout design, it must be ensured that the biasing is as close as possible to the active devices. Any noisy signal affecting the substrate or the well should be sunk by the biasing and should not affect the circuit itself. Typically, any possible silicon space should be used for biasing purposes [48]. From the viewpoint of manual design, it is possible for a certain way to visualize where the biasing considerations will be placed, often using guard rings for this purpose.

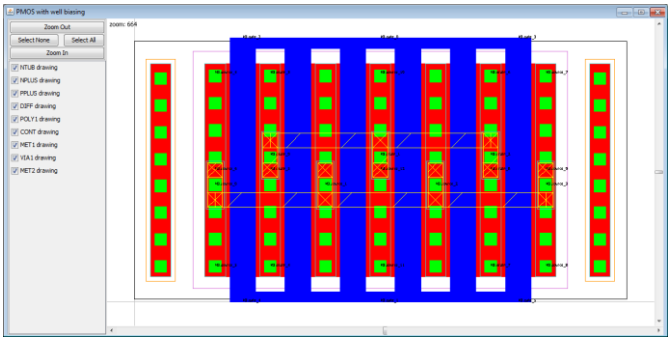
For the automatic layout generation, it is necessary to ensure that there is enough space for biasing. If the biasing considerations are performed after the modules placement, in some layout topologies may cause that biasing is placed too far from the active devices. While undertakes performance, may even violate technology design rules. The solution used by LAYGEN II is to include biasing directly on module generation. In the actual module generator from the 130 nm process it is possible to perform biasing with PMOS transistors, NMOS transistors and guard rings. For the current technology it is not yet being considered triple well. In Figure 4-5 are presented some examples of PMOS and NMOS transistors featuring well and substrate contacts, accordingly. As the remaining modules in the current parametric module generator, each device is self-symmetric. For the transistors of Figure 4-5 (a) and (b) the well/substrate contacts are merged with the source active area of the transistors.



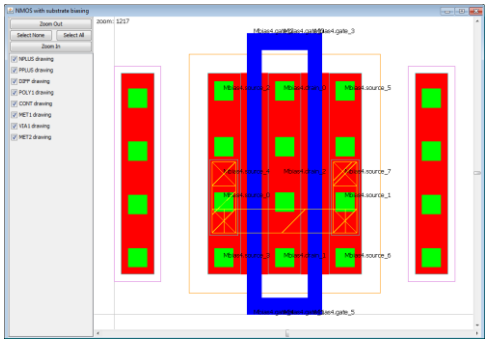
(a) PMOS with well biasing.



(b) NMOS with substrate biasing.



(c) PMOS with well biasing.



(d) NMOS with substrate biasing.

Figure 4-5 – Transistors with well/substrate contacts: (a)-(b) merged and (c)-(d) separate.

In a first phase, placer uses the bulk information provided in the template and analyzes the connectivity provided to verify if the net assigned to bulk is the same net assigned to source terminal. The connectivity from all template hierarchy should be considered, since this connection may exist in higher level templates. The modules that have the bulk at equal potential of source allow for the use of topologies of Figure 4-5 (a) and (b). These modules are essential for routing because there is no need to add bulk terminals in the device, it is only necessary to keep the connection to the device source. All modules that suit these requirements are substituted into the template for a module with biasing.

If the net assigned to bulk isn't the same net assigned to source terminal, the module is substituted by one of the topologies of Figure 4-5 (c) and (d). Those modules possess two additional bulk terminals, so it is necessary to process the connectivity for all template hierarchy and then add a connectivity

term between the bulk terminals and the correct potential net. The connectivity to these bulk terminals may be defined in the template by the designer, overriding these automatic considerations.

Since numerous transistors may be substituted in this process, the designer can set the bulk information as *none* in the template description. For minimum distances considerations it is assumed by default that the bulk potential is *gnd* or *vss* in case of NMOS transistors, and *vdd* in case of PMOS.

4.3.3.2 Abutment

Merged devices appear separate from one another in schematics but they can be combined in the layout. Abutment creates an overlapped connection between two cells without introducing design rule violations or connectivity errors, which not only saves space and reduces the wiring length, but in some cases also improves performance by decreasing parasitics. It is the designer's task to weigh the benefits of mergers against the possibility of introducing unexpected interactions between merged devices [57].

The B*-Tree although very efficient is not suited for merging devices. One possible solution is to deceive the packing algorithm of the B*Tree, by providing sizes that are smaller than the actual layout for a cell. However, this creates some problems, one immediate consequence is that it may create undesired overlaps, contradicting the macro-cell placer without overlap nature of the current approach. So, the design rules would have to be verified internally to the modules, to ensure they were correctly placed. Another approach is to extend the parametric module generator to support merging devices instead of using the standard basic cells, the tool automatically indentifies in a layout the cells suited to be merged and replaces them for a specific parametric module.

If the two cells are placed together (without any cells between) and both are marked as matched in the template they are suited to be merged. The placer must analyze the connectivity provided and verify is the two transistors are connected in the same net, both by source or by the drain. As above, the connectivity from all template hierarchy should also be considered. If the two share the same source, a common-source topology is adopted, otherwise, a common-drain one. The gate connectivity is simultaneously verified, if exists, the module is generated already featuring connection between the two transistor gates, this feature is important to automatically create pins between the two sides of the abutted transistor, preserving symmetry. If no terminals are shared, the cells are kept the same, and none changes are performed.

If the two transistors are marked to be abutted, the two modules in the template are substituted by a single module, as depicted in Figure 4-6. The new module is formed by the two previous modules bounding boxes, as well as their names. This process is repeated for all pairs to be merged. At this stage it is necessary to process the connectivity and constraints for all template hierarchy. As some modules for which the designer provide connectivity have been substituted, and hence the devices names and terminals, it is necessary to ensure that connectivity is updated to the new devices designations. Beyond saving space, abutment also reduces the length of the interconnect wiring, since in this process some wires disappear and thereby obtaining a simpler layout.

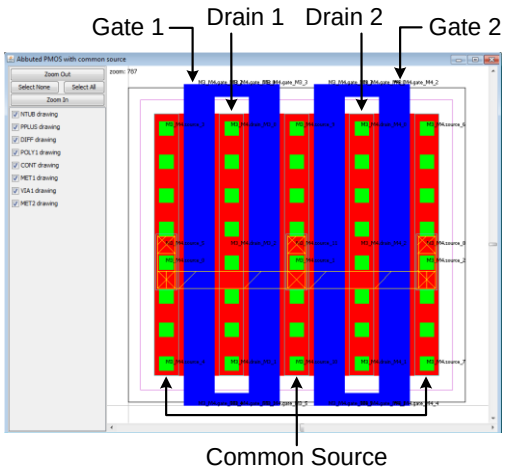


Figure 4-6 – Two PMOS transistors with the source merged.

For the example of Figure 4-2 the template obtained after applying this processing is presented in Figure 4-7. Although transistor M1 and M2 are matched and possess the sources connected to the same net, the transistors between prevent them from being merged. Even for a small circuit like the differential amplifier used, with just the abutment of transistors M3 and M4 it was possible to remove two wires from connectivity, namely, the connections between the two transistors sources and the two gates. Besides improving the layout quality, for a circuit with large dimensions this reduction in the number of wires will significantly increase the performance of the optimization kernel of the router.

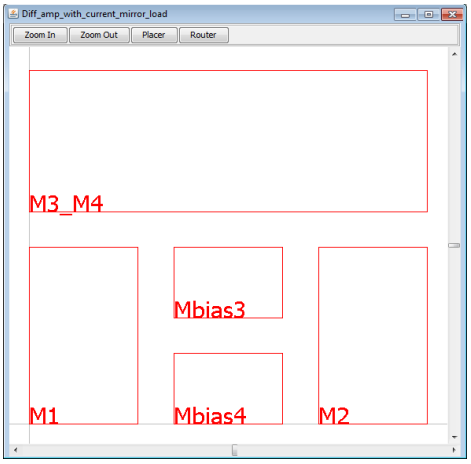


Figure 4-7 – Template after abutment processing.

This process only allows merging physically identical transistors so that symmetry is kept. The feasibility and advantages of merging devices with different number of fingers or even different widths, should be studied in a future implementation. The same analogy of this section may be utilized to interdigitized or common centroid transistors, if the proper cells are available in the module generator of the target technology design kit.

Since in this pre-processing the template cells are changed, it is necessary to perform a new B*-tree extraction before packing a new layout. Figure 4-8 presents the obtained layout for the current example after the pre-processing tasks.

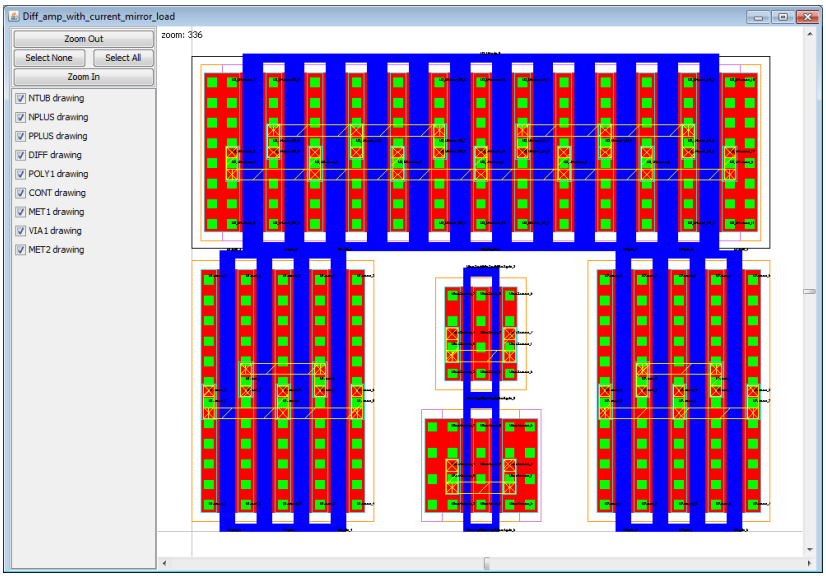


Figure 4-8 – Layout after the packing from pre-processing (GUI).

4.3.4 Post-processing

The post-processing tasks are performed over the layout obtained after pre-processing and consequently second packing. In this phase it is not performed module substitutions neither changes to the nets, but rather the current layout is processed in way to verify technology design rules, as depicted next. Then, the guard rings are added if desired by designer.

4.3.4.1 Minimum Distances

As it is possible to observe in all layouts presented until this section, the floorplan modules are all placed together and the minimum distances required by target technology are not ensured. These minimum distances rely on the type, biasing properties and effective location of the devices, so they can only be set when all the modules are known.

This process assumes that cells in the floorplan will not suffer further changes. Each cell in the floorplan verifies the distances between her and all the cells placed above and at right. If the minimum distance is not ensured, the cell's bounding is expanded until the value that fulfills the technology design rule is imposed. This way, the cells are placed exactly at the minimum distances allowed by target technology and no space is occupied beyond the necessary. Since the floorplan provided for post-processing present all modules placed together, some special cases are taken into account without changing the current cells bounding boxes, e.g., wells at the same potential are not separated.

After all the cells are verified, the floorplan is again packed for the new bounding boxes. Each cell is placed at the bottom left of their own expanded bounding box, thus only the need of verifying the cells conflicting at top and right. This task assumes that all modules instantiated were previously validated by Calibre® DRC, so there is no need to verify internally in the module the minimum distances, enclosures or extensions between layers.

4.3.4.2 Guard Ring

Off all the many types of failures that plague ICs, none is more frustrating and elusive as latchup. Devices that operate properly in one circuit latch up the moment they are inserted into another, simulation rarely uncovers latchup problems and neither do most forms of testing. The smaller dimensions of modern CMOS processes made them more prone to latchup. Although not totally solve the problems, the use of guard ring try to enhance immunity to this possible circuit failure and should be used by designers as suited [57].

The possibility of generating guard rings with different shapes, when supported, is a feature commonly used by the designers in the layout edition tools. In LAYGEN II is equally possible to automatically generate guard rings with different geometries. In order to define which guard ring geometry must be drawn, the solution found was to take all the four edges from the bounding boxes of all devices in the floorplan, already considering the minimal distances between each cell, and applied an algorithm of convex hull.

In the current planar case, the convex hull for a set of finite points is the minimal convex polygon containing all the points. The method of computing the convex hull used is the Graham's scan [58], which is a method of computing the convex hull in a plane with complexity of O(n log n). For all non 90º segments obtained from connecting two followed points of the polygon, a third point is added to perform only rectangular structures. Some guard ring examples of the versatility provided by the current implementation are show in Figure 4-9.

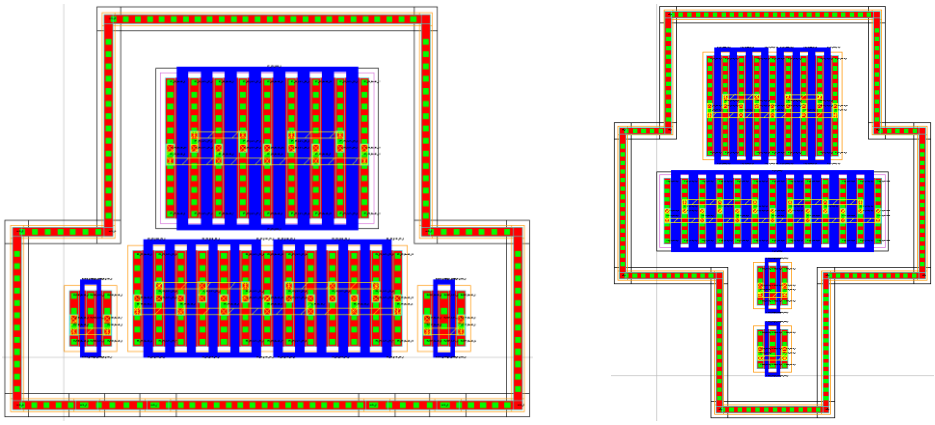


Figure 4-9 – Examples of the automatic guard ring adjustment to the obtained floorplan.

There are two available types of guard rings to be selected by designer, namely the P plus-based ring or N Plus-based ring. The P plus-based ring is relative shallow, and his ability to correctly blind the involving circuit is compromised since it can only intercept a fraction of the carriers. The N Plus-based ring although it's placed inside an N-well still has its limitations since most of the carriers flow down to the substrate instead of laterally to the guard ring. For a sturdy layout should be used a combination of different types of guard rings both to suppress most forms of latchup, and as usually used by designers to correctly bias the circuit.

The actual implementation places the guard ring around the circuit defined by a template. So, if designer only wants one transistor surrounded by a guard ring, it should defined a sub-template for that module, taking advantage of the hierarchical capabilities of LAYGEN II. Also, it is important to notice that each guard ring possesses a bulk terminal that should be properly connected during the automatic routing task.

In Figure 4-10 it is presented the layout obtained for the current example after the post-processing tasks, and consequently the final floorplan which will be provided for routing.

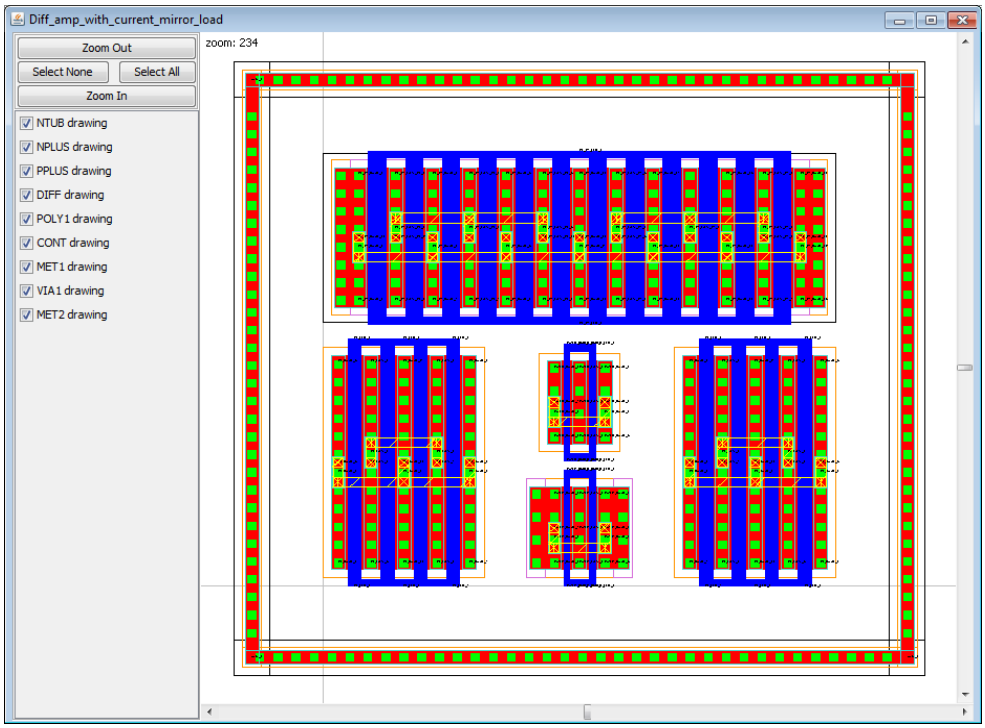


Figure 4-10 – Floorplan obtained.

4.4 Conclusions

In this chapter, the template information required for placement and the techniques used to generate it were properly introduced. The designer provides the high level floorplan and the tool automatically instantiates and places the devices in the layout, automatically abutting and ensuring that the design rules are fulfilled. Lastly, the tool automatically adapts guard rings to the floorplan obtained. A simple amplifier was used through the chapter to explain the proceedings.

Design rules were only mentioned in the post-processing of placer, as LAYGEN II's placer is a macro-cell placer without overlap. Therefore knowing that the instantiated cells comply with the design rules and assuming that the circuit is properly biased, maintaining the distance between devices is the only operation required to verify the technology design rules.

Chapter 5 **Router**

The router uses the obtained floorplan solution, the routing connectivity and a set of symmetry and sensitivity constraints contained in the template as inputs. The solution space is then explored ensuring that technology design rules and designer constraints are respected. Unlike the placer, the router must ensure that contacts, vias and wires do not violate any of the design rules. In addition, care must also be taken to avoid unwanted contacts between electrically connected layers, wires shunts or connecting a wire to unwanted shapes of the cells underneath. All of these design rule validations and the huge search space make the routing algorithm extremely complex, and computationally more expensive than placer.

The first section of this chapter covers the general description of router architecture. Then, the information necessary provided by the designer, namely, the connectivity and routing constraints for the circuit is presented. Even though router follows an approach characteristic of the fully-automatic generators, the connectivity and constraints are provided as template designation, in order to keep uniformity with placer. The following section addresses the generation procedure for the routing, depicting each task implemented in LAYGEN II's optimization-based router, with emphasis on the evolutionary computational techniques used. Then is detailed the internal evaluation procedure used to depict if the routing solutions fulfill all the technology design rules and constraints.

5.1 Router Architecture

The proposed generation architecture is shown in Figure 5-1, which depicts the principal tasks performed by the optimization-based router. This process is performed in two main stages, marked as Phase I routing optimization kernel and the Detailed Routing optimization kernel. Before the phase I there is an initialization step detailed in section 5.3.2.2. The inputs required are the connectivity and constraints presented above, and the floorplan obtained from placer.

LAYGEN II performs a flexible and easy to setup optimization-based routing. The connectivity and constraints although obviously dependent from the circuit, are provided independently from the floorplan attained and the processing tasks performed by placer. This is, the designer guidelines for placement may change, along with the devices sizes or even the target technology, but the template provided for routing remains valid. It isn't necessary to perform any kind of modification to the connectivity and sensitivity, and only if the topological relations sharply change may be required to delete some symmetries.

In the previous LAYGEN implementation [13] the router followed a strict template-based approach where the designer provided for each wire in the template the two exact pins to be connected, a geometric shape for the wire defined by a set of points, and the preferred conductor layer. All this information made the process of defining the template extremely long. Additionally, the fixed position of the pins and the limited operators, which were made not to disregard the designer definitions,

introduced great limitations on the routing flexibility during generation. Moreover, any change in the topological relations between cells could compromise all the connectivity and geometries, forcing template re-design. Although suited to abstract designer from technology details, the strict template-based approach did not yield physically achievable layouts for wide specification changes.

During the development of the actual full-automatic router, the possibility for the designer to totally define a wire was kept. The solution could be generated from any range of template-based wires to automatic generated wires, at designer's criteria. However, template wires proved to lack flexibility and the automatic wires were strongly conditioned by the enforced geometries, requiring the use of more conductors layers even for a fairly simple routing. The exploration of different topologies in placement carried long setup time of the template-wires, while the connectivity for automatic wires does not require any modification. For the reasons outlined, template-based routing is to be discontinued and is not present on this document.

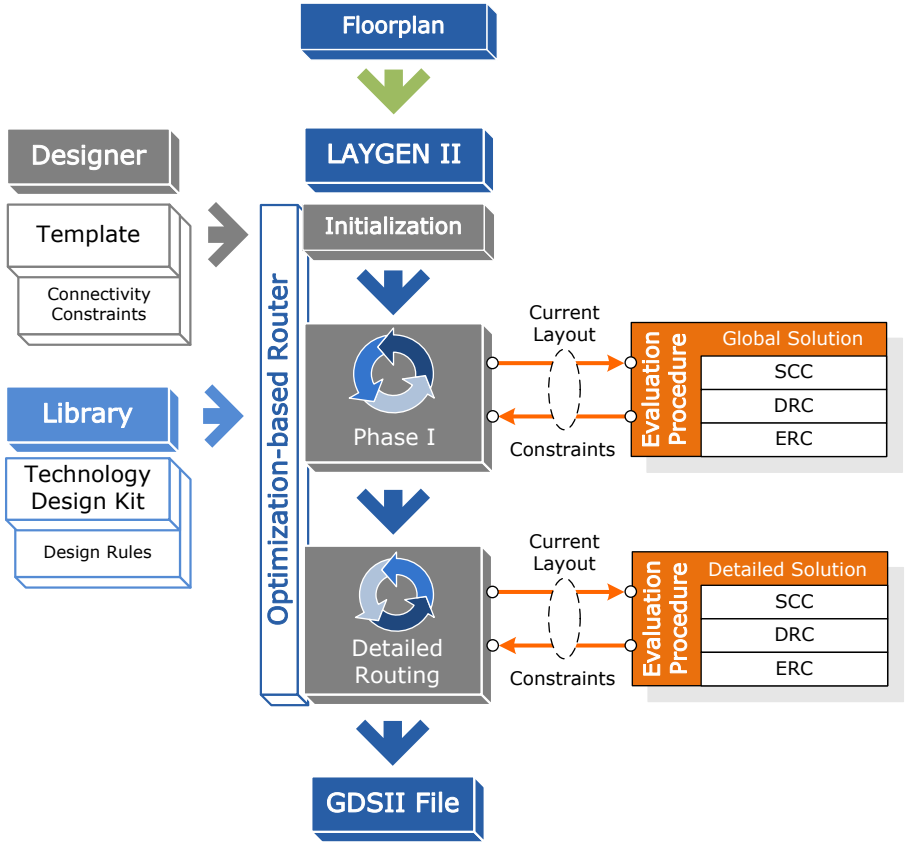


Figure 5-1 – Optimization-based router architecture.

The violations of the design rules are used as constraints during the routing generation, which the evolutionary algorithm must drive to zero. The evaluation is performed by a powerful internal evaluation procedure which possesses three different types of validation, short circuit check (SCC), DRC and ERC, detailed in section 5.4. Other qualitative measures are used as objectives for the evolutionary algorithm, for example the total wires length, the number of conductors or contacts used for routing, and other objectives more oriented to the processing of special nets, such as distance between noisy and sensitive.

5.2 Template

Each net is divided into a set of wires, each one connecting two and only two contact points, or pins. Internally, each wire is formed by any number of linked segments. For its part, a segment refers to the connection of two different points in the space, whose values in the x axis or in the y axis are the same. The routing connectivity is defined using the same cells designations used for the template of placer, and the terminal designations associated with the modules. The designer only has to concern on defining the set of nets and the respective wires, each one connecting two and only two terminals.

In a cell, the pins of a terminal are identified with a label. The parametric module generators should provide that same label placed over the shapes that are part of the terminal. Even though each terminal may have any number of pins, the router deals automatically with them, so designer only has to concern with the correct terminals. In order to clarify, when referring to the terminals of a cell, e.g., a transistor source, the term terminal is used. When referring the possible contact points of a terminal, e.g., the exact location over the stripe of metal where the connection can be made, the term used is pin. The next paragraphs provide more detail of the elements used to describe the routing template.

Figure 5-2 presents a possible connectivity and constraints description for the routing generation of the differential amplifier with a current mirror load of Figure 4-2 (a).

```
<NetList>
        <PowerNet id="vdd" mode="top" pin="true" width_pc="50">
                <Connect term="M3.source"/>
                <Connect term="M4.source"/>
                <Connect term="bulk"/>
        </PowerNet>

        <PowerNet id="gnd" mode="bottom" pin="true" width_pc="50">
                <Connect term="Mbias4.source" />
        </PowerNet>

        <Net id="id01" pin="false">
                <Wire source="M1.source" sink="M2.source" />
                <Wire source="M1.source" sink="Mbias3.drain" />
        </Net>

        <Net id="id02" pin="false">
                <Wire source="Mbias3.source" sink="Mbias4.drain" />
        </Net>

        <Net id="Vout-" pin="true">
                <Wire source="M3.gate" sink="M4.gate" />
                <Wire source="M3.gate" sink="M3.drain" />
                <Wire source="M3.drain" sink="M1.drain" />
       </Net>

        <Net id="Vout+" pin="true">
                <Wire source="M4.drain" sink="M2.drain" />
        </Net>

        <Symmetric net1="id01.3" net2="vout+.1" />
        <Symmetric net1="id02.1"  net1="id02.2" />
</NetList>
```

Figure 5-2 – Connectivity and constraints example for the circuit of Figure 4-2 (a).

Here two power nets, *vdd* and *gnd*, were defined that will instantiate a conductor stripe above, on the left, on the right or bellow the circuit according to the designer discretion, similar to what it's done in digital layout design. Unlike other wires, the terminals connecting to a power net can be defined sequentially.

Any pair of two wires may be denoted as symmetric. Since those two lines are physically identical except one is mirrored, if they do not belong to the same net it is important to ensure that those two lines will not cross each other in the layout, in order not to cause an inevitable short circuit. It is also possible to denote nets as noisy or sensible, to be treated differently from the other nets during the evolution of the optimization as detailed ahead in section 5.4.3. The pin flag is activated by the designer if he intends to save the net label in the GDSII file.

The lack of electrical measures is compensated by the constraints provided by the designer in the template. It can be defined a certain percentage of current density for a net, resulting that each wire of that net will have a width equal to the minimum width allow by target technology to the current conductor, plus a percentage of that value. For the current template the net *vdd* and *gnd* were defined to be 50% wider than the minimum width allowed by target technology to the current conductor used. This parameter will affect in the same manner the width of the conductor stripe of the power net.

The connectivity and constraints are provided always taking into account the initial elements from the electric schematic, and are provided simultaneously with the template for placement. As detailed in section 4.3.3 the pre-processing tasks perform some changes in the modules and consequently nets. In Figure 5-3 the changes automatically performed by placer are depicted, these are due to the merging of transistor M3 and M4 sources and gates.

```
<PowerNet id="vdd" mode="top" pin="true">
        <Connect term="M3_M4.source"/>
        <Connect term="bulk"/>
 </PowerNet>

 <Net id="id01" pin="false">
        <Wire source="M3_M4.gate" sink="M3_M4.drain_M3" />
        <Wire source="M3_M4.drain_M3" sink="M1.drain" />
 </Net>

 <Net id="out" pin="true">
        <Wire source="M3_M4.drain_M4" sink="M2.drain" />
 </Net>

 <Symmetric net1="id01.2" net2="out.1" />
```

Figure 5-3 – Template nets changed by the pre-processing from placer.

5.3 Optimization-based Generation Procedure

In the next sub-section before introducing the routing chromosome and the genetic operators, it is necessary to explain how LAYGEN II deals with the transitions between different conductors. Then, some detail is provided for the optimization kernel used, including the chromosome structure,

initialization and operators. After, the differences between the different routing optimization phases are depicted, finally, the internal evaluation procedure is explained.

5.3.1 Multiple Contacts

It is a designer's common practice the use of multiple contacts not only on top of the source and drain regions to avoid parasitic transversal drop voltages, but also in any transition between conductors. Usually multiple contacts are placed at a minimum distance instead of using a single large contact. Many contacts placed to each another make the surface of metal connections smoother than when using only one contact, this prevents microcracks in the metal that can be a source failure [48]. From the standpoint of automatic generation, the use of multiple contacts is easily achieved for the active regions of the devices, but for small design processes greatly limit the capability of the automatic routing to perform transitions between conductors.

The contacts, vias and connection plates necessary to perform a valid connection between conductors can cause the emergence of more violations of the design rules, hindering the algorithm convergence, so it is necessary to ensure that are placed in the best way possible. For LAYGEN II's router each transition is performed with a minimum of two contacts or vias. In Figure 5-4 are depicted the six different arrangements of contacts with respect to a connection point, marked in the figure with the black dot. The contact plate can be placed horizontally or vertically, and centered or shifted to the right or left. The same analogy can be used for two or more contacts or vias. It is important to notice that the connection point, marked with a dot, may refer to a pin of a device terminal or the point of transition between two different conductors inside a wire structure

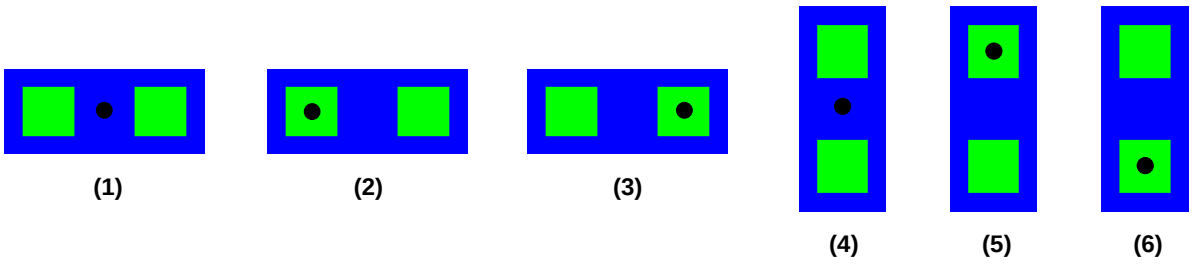


Figure 5-4 – Different arrangements of a contact or via with respect to a connection point.

When optimizing, the router only places the contacts under the wiring connecting. So, for each shape formed with two segments connecting exists just some valid arrangements from all the presented above. When a contact orientation is randomized for a given connection point, this random choice is made within a solution space which contains only the valid orientations, and not all the available, ensuring that a desirable arrangement is enforced.

For each transition between a vertical segment and a horizontal segment, there are two valid possible arrangements, as depicted in Figure 5-5 (a.1). For a transition between two segments with the same orientation, Figure 5-5 (a.2), there are three different possible arrangements that satisfy the connection, (1) to (3) if segments are horizontal and (4) to (6) if vertical. In Figure 5-5 (b) some illegal contact dispositions are illustrated. For the connections between a wire and a terminal of a device, the

pin label possesses the information that determines if the contact orientation must be horizontal or vertical.

(a.1) Segments with different orientation. **(a.2) Segments with the same orientation.**

(a) Example of legal contact dispositions.

(b) Examples of illegal contact dispositions.

Figure 5-5 – Contact arrangements of a transition between two different conductors.

5.3.2 Evolutionary Algorithm

During the initial development of LAYGEN II's router, it was used a classical GA approach where the constraints were modeled simultaneously with the objectives in the fitness function. Given the need to deal with constraints separately from objectives, the kernel was moved to the modified NSGA-II [53], an elitist MOEA. Although it is not being yet explored the full potential of the algorithm, this implementation opens up a wide range of future implementations, which are remitted to the appropriate section 7.2. The distinctive characteristic of the evolutionary kernel implementation is the chromosome structure. It must be adequate to support complex routing problems and still allow the use of common genetic operators, as detailed in next section.

The constraints are not enforced during the generation of the routing solution, ensuring design correctness for every wire, yielding only feasible solutions. This strategy yields large computation times, and all the infeasible points are treated equally, regardless if there is one violation of the minimum spacing between two wires, or hundreds of design rules violations and unwanted shunts. Instead, router follows an optimization approach where the population should converge to feasible, as constraints are driven to zero during the optimization procedure.

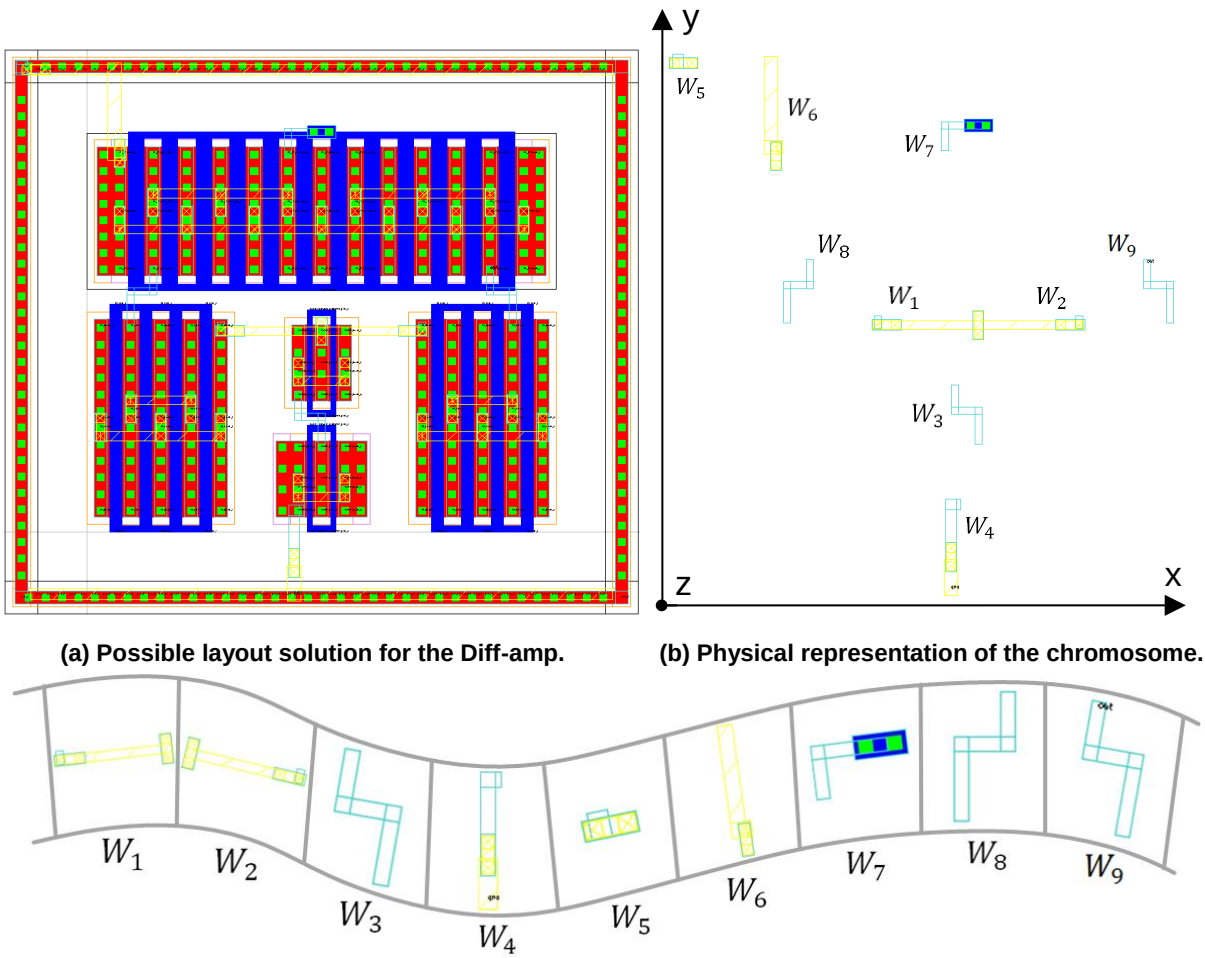
When the physical implementable solutions are obtained, i.e., that does not violate any of the constraints and hence will successfully validate in Calibre® DRC, the optimization objectives are taken into account. These objectives may be, for example, the total wiring length or the total number of contacts (or vias) used, in order to minimize them during the optimization. Each conductor layer has an associated cost pre-defined in the technology design kit, commonly the lowest conductor levels are associated with the lowest costs. So, the wiring length can be computed together with the conductor

cost of each segment, in a way to minimize not only the paths but also favors the use of the lowest conductor levels for a given technology.

Symmetric routing can be used if the terminals of the wires being router are symmetric with respect to a symmetry axis. The symmetry is handled at chromosome level. In a symmetric pair of wires, only one is operated during optimization, the other is generated deterministically from the first.

5.3.2.1 Chromosome

Each element in the population, chromosome, encodes the information of a different routing solution, corresponding each gene to one wire. So, each chromosome has a fixed number of genes equal to the number of wires present in the circuit. In Figure 5-6 (a) a possible layout solution generated for the connectivity and constrains provided in the example of Figure 5-2 is presented. Two different representations of the chromosome formed by nine genes (wires) are presented in Figure 5-6 (b) and (c).



(a) Possible layout solution for the Diff-amp.

(b) Physical representation of the chromosome.

(c) Abstract representation of the chromosome.

Figure 5-6 – Chromosome used for routing optimization.

For its part, each wire (gene) of the chromosome has the same even number of segments, which facilitates the implementation of the genetic operators. A segment may be defined as horizontal or vertical; if vertical, the one following must be horizontal, and vice-versa. In order to clarify the gene structure, in Table 5-1 the fields contained in a wire structure with four segments are presented. At any

time in the optimization, if a segment is defined with a delta length equal to zero obviously does not have physical representation. A layer is always associated with a segment, so a transition between two layers is always related to a transition between two segments. Since the current wire structure allows any number of segments with a null length, this permits layer transitions between two segments with the same orientation.

For the examples of this chapter the number of segments in a wire was set to four. While in Chapter 6, given the increasing complexity of the addressed problems and hence the routing solution, each wire was set to be constituted by six segments. In addition to the physical wire structure represented on Table 5-1, the wire also has the information of source and sink terminal, due to perform different pin search in mutation (section 5.3.2.3).

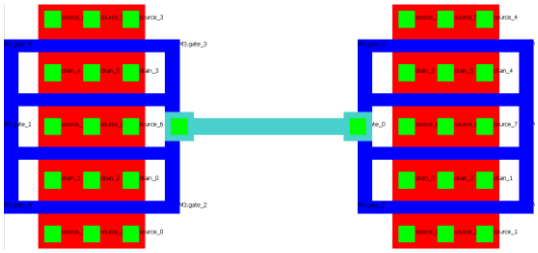
Table 5-1 – Wire structure composed by four segments.

Wire Structure	
Contact orientation connecting source pin to the first non-zero segment.	
Line Segment 1	Vertical segment
	Δy length
	Associated layer
Contact orientation connecting segment 1 to segment 2.	
Line Segment 2	Horizontal segment
	Δx length
	Associated layer
Contact orientation connecting segment 2 to segment 3.	
Line Segment 3	Vertical segment
	Δy length
	Associated layer
Contact orientation connecting segment 3 to segment 4.	
Line Segment 4	Horizontal segment
	Δx length
	Associated layer
Contact orientation connecting the last non-zero segment to sink pin.	

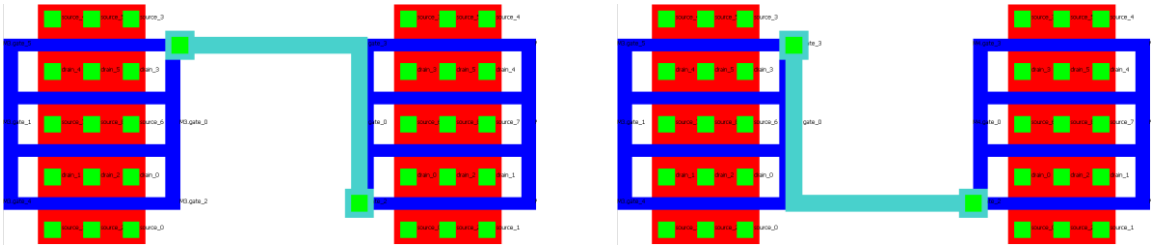
5.3.2.2 Initialization

Initialization step consists of generating the initial population for the evolutionary kernel of Phase I. For each wire of the chromosome only the source and sink terminals are known. The initial pins of the terminals connecting are selected following a greedy approach, the distance between each pin of the source terminal and each pin of the sink terminal is computed, the pair which represents the minimum distance between the two is selected. If implementable they represent the optimal solution, so are used as optimization starting point.

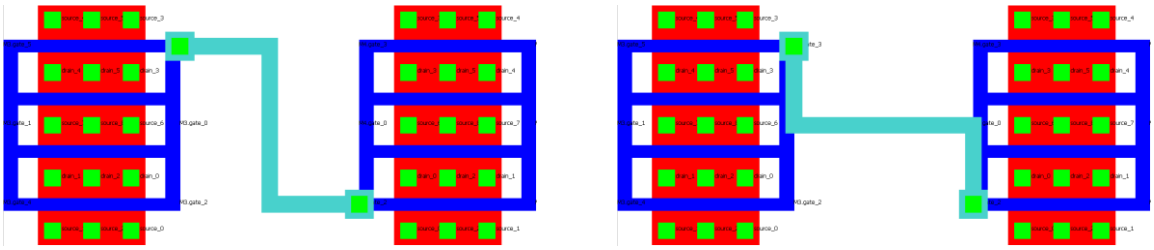
Although the actual chromosome structure allows fully random generation of the wire segments, it has been found that the use of pre designed shapes, henceforward called heuristics, present a better starting point for the connections between pins. So, each wire is randomly generated through a set of available heuristics; the possible geometries are presented in Figure 5-7. It is important to notice that in heuristics (c) and (d), the segments inside the wire structure, i.e., those not connected to source pin neither sink pin, can take any position in the range of x axis or y axis allowed by the distance between terminal pins, sharply increasing the solution space. Since the only restriction is the effective pins location, initialization step only has to ensure connectivity without validating technology design rules.



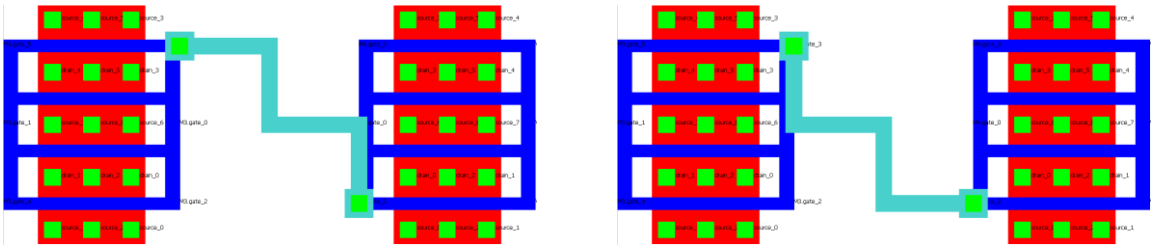
(a) One segment.



(b) Two segments.



(c) Three segments.



(b) Four segments.

Figure 5-7 – Different heuristics for generation of wires.

Although the current wire structure may have an even fixed number of line segments higher than four, the use of heuristics with more than four segments has not proved necessary. After the heuristics set, each wire, and hence segment, it is initialized with the conductor with the lowest associated cost in the current technology design kit. After the heuristic initialization and correction, each contact orientation

of the gene is randomly set within the valid arrangements, ensuring that no connection plate is placed outside the boundaries of the two line segments connecting.

After each wire generation, it is performed a *remove small segments* validation to remove the segments considered to be too small to be included in the solution. If a segment presents a delta smaller than his width, the segment is deleted. Obviously, the removal of this small segment will cause that connectivity is no longer verified, so it is necessary to perform a *rescale* step. This processing equally rescales all the deltas from the remainder non-zero segments until the missing value is restored to the wire and verifies connectivity.

In summary, initialization step settles the ideal conditions for the current routing problem. A routing performed with greedy connections using the same lowest cost conductor for all connections would be great to achieve, however it is unlikely to be possible without violating design rules. The removal of the violation rules from the greedy initial population is the optimization problem in question.

5.3.2.3 Genetic Operators

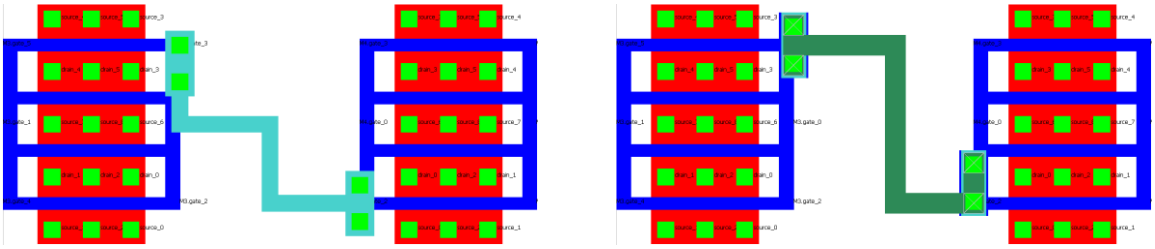
Although the evolutionary algorithm follows the NSGA-II flow, innovative genetic operators were developed to deal with the specific routing chromosome presented above. Next, are provided the detailed implementations of crossover and mutation operators.

Crossover

At each new generation, each pair of parents is selected by tournament to generate two offspring that present a combination of their information. The parents' contribution refers not only to segments length, but also the combination of layers and contacts orientation used. A multi-point crossover is used as presented in Table 5-2, for the two random parents depicted in Figure 5-8. Since the offspring represents a random combination of segment lengths from the two parents, marked in the table as red, it is common that the resulting wire fails the connectivity.

After each crossover, it is performed three different validations. A *connect* task must be performed to randomly add (or remove) the missing (or excess) delta x or delta y value to a valid segment, in order to ensure connectivity. Then, a *remove inverted segments* validation is made to assure that doesn't exist two horizontal or vertical followed segments with opposite directions. The inverted segment must be removed from the wire structure to guarantee that there are no overlapping segments in a wire. Finally, a *verify contacts orientation* task is performed to ensure that the actual contact orientations are valid for the new combination of deltas. If not, it must be randomly assigned a new contact orientation within the valid possible arrangements, ensuring that no connection plate is placed outside the boundaries of the two line segments connecting. The exceptions are the source and sink contact orientations since those depend on the orientation of the device.

After the referred corrections, the offspring obtained is marked at green in the Table 5-2.



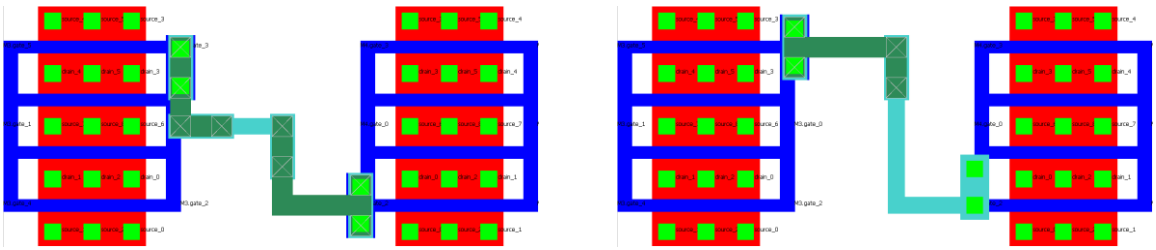
(a) Parent 1. **(b) Parent 2.**

Figure 5-8 – Example of two possible parents.

Table 5-2 – Example of crossover, parents (P1 and P2) and offspring (O1 and O2).

	Segment 1 Vertical			Segment 2 Horizontal			Segment 3 Vertical			Segment 4 Horizontal			Δ Total	
	Ct	Δ	L	Ct	Δ	L	Ct	Δ	L	Ct	Δ	L	Ct	Δx	Δy
P1	5	-156	M1	2	176	M1	5	-156	M1	2	176	M1	4	352	-312
P2	4	0	M3	1	199	M3	5	-312	M3	6	153	M3	6	352	-312
O1	5	-156	M3	1	199	M1	5	-156	M3	2	176	M3	4	375	-312
O2	4	0	M1	2	176	M3	5	-312	M1	6	153	M1	6	329	-312
O1	5	-156	M3	2	199	M1	5	-156	M3	2	153	M3	4	352	-312
O2	4	0	M1	2	199	M3	5	-312	M1	6	153	M1	6	352	-312

Ct – Contact orientation. **Δ – Delta length.** **L – Associated Layer:**
M1 – Level 1 conductor
M3 – Level 3 conductor



(c) Offpring 1. **(d) Offpring 2.**

Figure 5-9 – Example of two possible offspring generated, after correction tasks.

Mutation

A mutation ratio is applied to each of the chromosomes of the offspring population. There is a set of operators to be applied to the wires, which will introduce diversity for the further evaluations. They are classified in two categories: geometric operations and layer shifting operations. In the Figure 5-10 (b) to (f) different mutation operators were applied to the heuristic presented in Figure 5-10 (a). These operations are applicable to any wire, and the geometric operations are the following:

(1) Randomly select a new source pin from the source terminal (Figure 5-10 (b));
(2) Randomly select a new sink pin from the sink terminal;
(3) Randomize a new heuristic for the wire;
(4) Vertical slide on a horizontal segment ;

(5) Horizontal slide on a vertical segment (Figure 5-10 (c));

(6) Delete a vertical segment (Figure 5-10 (d));

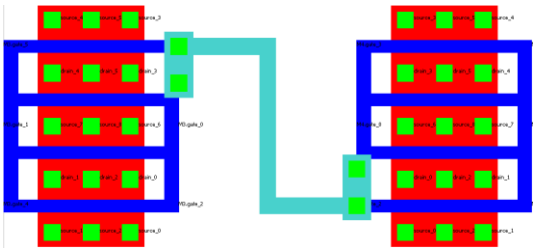
(7) Delete a horizontal segment;

(8) Randomize contact orientation (Figure 5-10 (e).

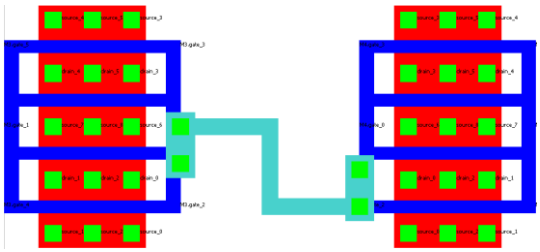
The layer shifting operations are:

(1) Move all line segments to a random conductor (Figure 5-10 (f));

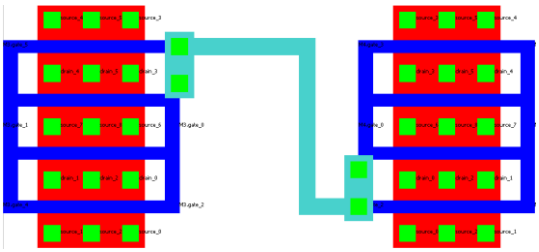
(2) Move a line segment to a random conductor (Figure 5-10 (g)).



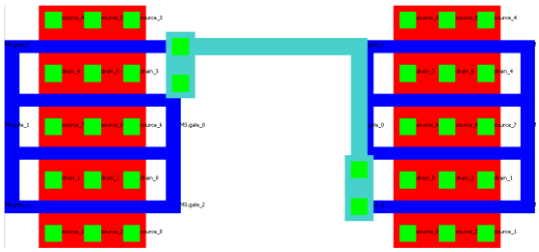
(a) Initial heuristic of three segments.



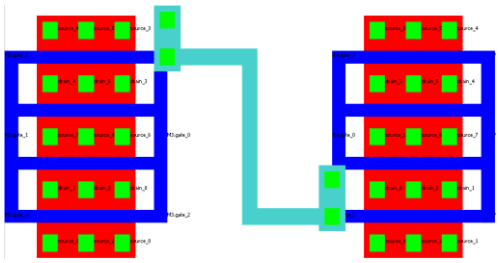
(b) New source pin.



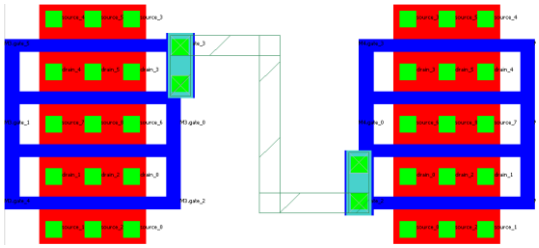
(c) Vertical Slide.



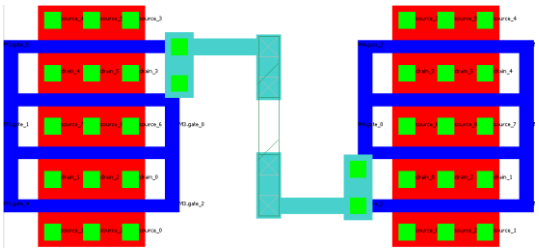
(d) Vertical segment deleted.



(e) New contact orientation



(f) Move all line segments to a random conductor.



(f) Move a line segment to a random conductor.

Figure 5-10 – Examples of mutation operators.

Since the router follows a greedy approach in the initialization, every wire is set to connect the two closest possible pins. The two change pins (source and sink) operators are essential to introduce

some diversity in the population since the closest connection may not be physically achievable. Using all the available locations to connect the wires will result in better routings. As the randomize source and sink pin operators, the layer shifting operators are fundamental to circumvent the initial greedy approach where all wires from all nets were set to the same lowest cost conductor.

Deleting and sliding segments allow exploring endless combinations between wires geometries. These geometric operations allow using more effectively the lowest levels of conductors for a given technology, causing the wires adapt to each other without having the need of requiring higher level conductors to avoid short circuits and fulfilling design rules. The randomize heuristic operator exist to add fresh information to the gene pool, as the other operators are incremental, making less aggressive changes to the wire structure.

After each mutation, since numerous changes may have been performed over each wire and the connectivity disrupted, it is necessary to perform a full set of the correction tasks presented in the operators above. As in the initialization by heuristics, namely a *remove small segments* validation followed by a *rescale* step to ensure connectivity. And then as made after crossover, a *remove inverted segments* validation and finally a *verify contacts orientation* task.

As it is possible to observe in the Figure 5-7, the initial heuristics always create wires confined in the rectangle defined between the two pins (edges). The genetic operators allow for out of bounding box exploration, this is, the wire geometry may depart as much as necessary away from the initial rectangle, as long as the connectivity is guaranteed.

5.3.3 Optimization Phases

Any thread of phases may be defined. No matter what is the phase sequence selected, the Phase I population always receives a greedy initialization detailed at section 5.3.2.2. For the current architecture presented in Figure 5-1 there are two phases represented. Phase I and Detailed Routing represent the optimization stages of the router. The existence of two optimization stages, is due to the fact that when the exact physical representation of the wire is generated the number of layout elements being handled greatly increase, and the evaluation process becomes much slower because that its complexity is at best $O(n^2)$.

It was noticed that if the contacts, vias, and respective enclosing shapes were not generated, the evaluation was much faster. For this reason, in Phase I only the main shape of each segment is generated and evaluated. This makes the processing faster and allows the exploration the general shape of the wires, obviously, the results might be infeasible, but they are obtained much faster. Both SCC, DRC as ERC validations are performed in each one of the phases.

Detailed Routing uses the output of phase I as initial population, and performs a more local and detailed optimization, taking into account the exact shape of the wire, considering every contact, via and conductor plates. When the solution is found, the generated routing is added to the placement and the target layout is saved in a GDSII file.

The routing for the template of Figure 5-2 was automatically generated using the floorplan of Figure 4-10. Given that no net was marked as noisy or sensitive, the optimization constraints considered were the number of short circuits and the number of design rules violations. The optimization objectives were the total wiring length computed together with the conductor cost associated and the number of contacts used. The layout was generated for a population of 128 elements, for 100 generations, along with a mutation rate of 3% and a crossover rate of 90%, in approximately 18 seconds. The detailed routing obtained is presented on Figure 5-11.

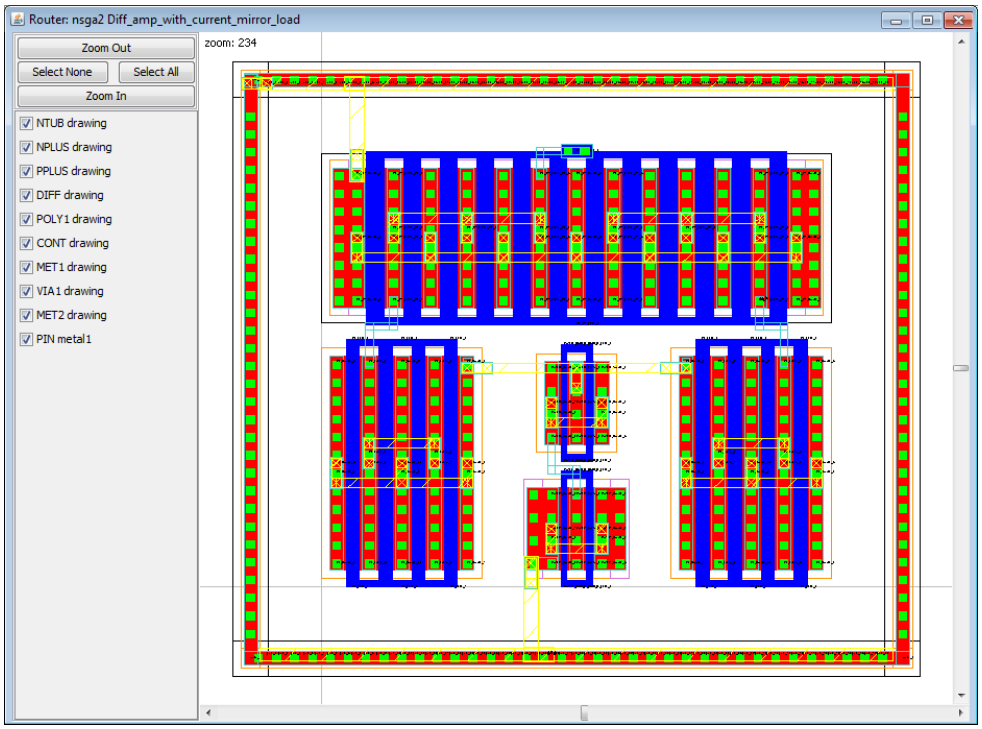


Figure 5-11 – Detailed routing obtained.

As depicted in the figure above, each defined power net is automatically adjusted to the width or height of the guard ring, depending on the position defined by the designer in the guidelines. Since the guard ring is biased with the same potential of *vdd* power net, the stripe of conductor defining the power net is drawn with the same conductor.

5.4 Internal Evaluation Procedure

In order to evaluate efficiently the correctness of the circuit, an internal evaluation procedure was implemented that performs lightweight, but accurate design rule check, short circuits check, and some electrical rules check. The violations of the rules are used as constraints during optimization, and must be driven to zero.

To further increase the efficiency of the evaluation, it is done using one thread per available CPU, splitting among them the individuals being evaluated. Given the nature of evolutionary algorithms and the independence of the individual in the population the use of multi-threading in the evaluation procedure is greatly simplified. In addition, since nowadays most of the workstations have more than

one CPU, a local thread pool can be used to accelerate the evaluation without many of complications associated with remote multi-threading or clustering parallelization techniques. For the current implementation, the use of three threads instead of a single thread, generated solutions in about 95% smaller computational times for the evaluation of Phase I, and around 135% smaller for the evaluation of Detailed Routing.

The use of an internal evaluation procedure rather than executing and then parsing the reports from an external tool, has many advantages when comparing the computational times. However, the commercial tool is more reliable than a custom made evaluator. This internal evaluation is constantly subjected to an exhaustive debug to ensure that the validations are being made correctly. The detection of false errors may difficult the convergence of the optimization kernels. The reliability is not compromised since the results will be either way validated in Calibre® DRC tool.

5.4.1 Short Circuit Checker

The way the wires are operated during the optimization ensures the desired connectivity is present in the output, however is does not ensures that unwanted connections between nets are not created. The short circuit checker is used to count the number of unwanted connections between nets, and this must be reduce to zero during generation. The pseudo code for the SSC is presented in Figure 5-12.

```
short_circuits = 0;
for (each Wire W in Layout) {
        List Wire_shapes = Wire.get_Physical_Representation;
        //Check for Short Circuits with Placement
        for (each different Layer L in Wire_shapes)
                List Layer_shapes_wire = Wire_shapes.get(Shapes from Layer L);
                List Layer_shapes_placement = Placement.get(Shapes from Layer L);

                Layer_shapes_placement.remove(Wire W Source Shapes);
                Layer_shapes_placement.remove(Wire W Sink Shapes);

                for (each shape sh1 in Layer_shapes_wire)
                        for (each shape sh2 in Layer_shapes_placement)
                                if (sh1 intersects sh2)
                                        short_circuits ++;
                                        add markers in layout viewer;

        //Check for Short Circuits with other Nets
        for (each other Net N not containing Wire W)
                for (each different Layer L in Wire_shapes)
                        List Layer_shapes_wire = Wire_shapes.get(Shapes from Layer L);
                        List Layer_shapes_net = Net N.get(Shapes from Layer L);

                        for (each shape sh1 in Layer_shapes_wire)
                                for (each shape sh2 in Layer_shapes_net)
                                        if (sh1 intersects sh2)
                                                short_circuits ++;
                                                add markers in layout viewer;
}
```

Figure 5-12 – Short circuit check algorithm.

For each wire in the layout, its' shapes are checked against all the shapes from the devices (except the ones connected to the wire source/sink) and from other nets. Only shapes in the metals and poly

layers are checked, because when contacts or vias are instantiated, the enclosing metal shape (above and below) are always added.

5.4.2 Design Rule Checker

To ensure that the generated routing complies with the process design rules, a mixture of correct by construction and optimization is used. Some rules, like the enclosing of vias by metals or minimum sizes are not validated. In these cases as all shapes that are created within the application must be correct. If during the Calibre® DRC verification any of these rules is violated, it means that there is a bug in the generation code or design kit, not in the verification procedure.

However, in the way the problem is modeled, some rules may be violated during the generation, they must be verified and their occurrence must be driven to zero. The rules that are verified using the internal design rule checker are:

- Minimum spacing between conductors;
- Minimum spacing between contacts or vias.
- Minimum and maximum widths of conductors and vias.

In some technology design kits, the vias have a fixed size that must be verified because even though different shapes are created with the correct size, overlapping two vias may create an illegal shape (only if they totally overlap the layout is correct).

Each segment or shape as an associated bounding box. The minimum distances validation procedure is essentially an interception check between the expanded bounding box of the actual shape and every other shape in the same conductor. The initial bounding box was expanded in each direction by a size equal to the minimum distance allowed by target technology. However, a shape may have valid connections to any number of other shapes and still has to verify a fixed distance with the remaining shapes. Obtaining this special expanded bounding box is where the main computational effort of design rule check lies.

Because the layout of the placed devices is correct, those shapes do not need to be verified, however for each wire all its' shapes should be verified against all the placement shapes and the shapes from the other wires. The verification approach used was to iteratively add the shapes of each wire to the layout and compute the number of violations when comparing the wire shapes against each other, and against the shapes already present in the layout.

5.4.3 Electrical Rule Checker

For the current implementation, the electrical rule checker is used essentially to identify the constraints violated by the noisy and sensitive signals, as detailed next. As the tool is being adapted for smaller design processes, the treatment of other important electrical events must be taken into account.

5.4.3.1 Noisy and Sensitive Signals

In an AMS ICs, analog signals are far more-noise sensitive than digital counterpart, but not all analog signals are equally sensitive. The most sensitive nodes are those that carry very low-level signals at high impedance levels, e.g., the input of an amplifier is far more noise sensitive than its output, because any signal present at the usually high-impedance input is amplified by the gain of the amplifier to the low-impedance output. The following types of signals are among the most noise sensitive [57]: inputs to high-gain amplifiers, precision comparators and analog-to-digital converters; outputs of precision voltage references; analog ground lines to high-precision circuitry; precision high-value resistor networks; very low-level signals, regardless of impedance; and very low-current circuitry of any sort.

Usually, noisy and sensitive signals are identified during the specification translation task to be properly treated during the routing task, in which exist a set of common techniques used by designers. The identification of these lines is a difficult task in the absence of electrical measures, and most layout designers do not possess the knowledge and experience required to correctly identify all of them in a complicated analog circuit. As stated for LAYGEN II, designer uses the template to identify all the noisy and sensitive nets, and the router deals automatically with them.

Typically noisy circuitry is placed as far away as possible from sensitive circuitry. In some floorplan dispositions noisy circuitry may even occupy one portion of the die and sensitive circuitry another. It is known that noisy should not run on top of sensitive signals, or vice versa. If a crossing must occur the area of intersection should be minimized and an electrostatic shield technique used. The usual method of constructing such shielding is to run one signal in one conductor and the other signal in a conductor two levels higher. A plate of conductor connecting to a low-impedance node, e.g. ground line, is interposed in the conductor between the two signals and acts as an electrostatic shield. Also, in order to run noisy signals adjacently to sensitive, a shield line must be used between the two signals specifically to shield the sensitive signal from the noisy one. The shield line used is generally a low-noise, low-impedance signal such as extra ground line, supply line or even the output of digital logic gates [57].

While the techniques listed can somehow be used with more or less success in the manual design, from the standpoint of automatic generation the use of extra lines connected to ground or power, either to perform electrostatic shields as shield lines, would mean an abrupt increase of the solution's complexity. As LAYGEN II's router uses the wiring length as objective, is ensured that the sensitive signals are always as short as possible, reducing the opportunities for noise coupling. To assure the best possible treatment for the nets denoted as noisy or sensitive without unduly increase the solution complexity, any crossing between any wire of a net denoted as noisy and another denoted as sensitive is considered an ERC error.

It is know that sensitive signals should not pass through other circuit blocks, the same can be applied to noisy signals if the sensitive devices of a given circuit block where not correctly identified. In order to reduce the possible capacitive coupled noise in the optimization, each noisy and sensitive net

running on top of the active area of the devices is marked as an ERC violation. Simultaneously with the verification of the possible interceptions, it is computed the minimum distance between any segment of a noisy line and a sensitive one. This computed distance may be used as objective during the evolutionary optimization in order to maximize it, and place the two types of nets as far away as possible.

The internal evaluator also marks individually each wire as he contains any constraint violation or not. In the next generation and subsequent mutation, wires that fulfill all the constraints have a smaller mutation ratio than the wires violating constraints. This way the wires without errors suffer fewer changes and are preferably kept in the solutions, the ones containing errors need to evolve to adapt to the remaining and become valid.

5.5 Conclusions

In this chapter, the optimization-based router was described and the simple amplifier presented in the previous chapter was used to demonstrate the results. A summary of the constraints and objectives used by the MOEA are depicted in Table 5-3, other constraints or objectives can be easily added as suited for the designer. The optimization algorithm and hence the chromosome structure and genetic operators were properly detailed. Along with the internal evaluation procedure that allows LAYGEN II to avoid the need of external evaluation tools.

Table 5-3 – Summary of constraints and objectives.

Constraint	Target	Description
SCC	$= 0$	Short circuits.
DRC	$= 0$	Minimum distance violations.
ERC	$= 0$	Crossing between noisy and sensitive nets, or running on top of devices.
Objective	**Target**	**Description**
Wiring Lengh	minimize	Total wiring length computed with the associated conductors cost.
Contacts	minimize	Number of contacts or vias used.
Distance	maximize	Distance between noisy and sensititve nets.

Unlike the placer where a strict template-based approach is followed, in the router, LAYGEN II replaces the obligation to choose the exact routing, by a set of constraints that guide the automatic generation to solution according to the designer wishes. The routing constraints are independent of the floorplan and can be valid even for different placement topologies. Plus, the connectivity is immutable no matter changes performed in the floorplan, since they only depend on the existence of the terminals of the devices.

This extremely versatile approach, allows for the designer to provide the connectivity, and a set of symmetry and sensitivity constraints, and the tool automatically starts a optimization proceeding that will lead to a solution that strictly complies with the target technology design rules. The solution is generated and validated even if only connectivity is provided.

Chapter 6 **Results**

This chapter presents the design tasks required for the application of the proposed design flow to practical examples. In order to use the implemented platform, the first design task is the development of the technology design-kits. Since the presented UMC 130 nm CMOS design kit and its parametric module generator accompanied the development of the tool, it is difficult to establish a development time. It will be used as main reference for the design of the layouts presented in this chapter.

The framework of the proposed methodology for the automatic generation of analog ICs layout, based on template descriptions and on evolutionary computation techniques, has been coded in JAVA and is running, for the two presented examples, on an Intel® Core™ 2 Quad CPU 2.4 GHz with 6 GB of RAM. Are being used three threads (cores) to perform the evaluation procedure of each population, at each generation. The code automatically generates the GDSII file required by Calibre® tool.

6.1 Case Study I – Fully-Dynamic Comparator

The first circuit used as test case is a fully-dynamic comparator which is part of a ΔΣ Modulator [59], courtesy of the group from CTS-UNINOVA, along the AIDA project. The comparator schematic is presented in Figure 6-1 and was designed for UMC 130 nm CMOS technology. The handmade layout of Figure 6-2 was designed for the devices sizes provided in Figure 6-3.

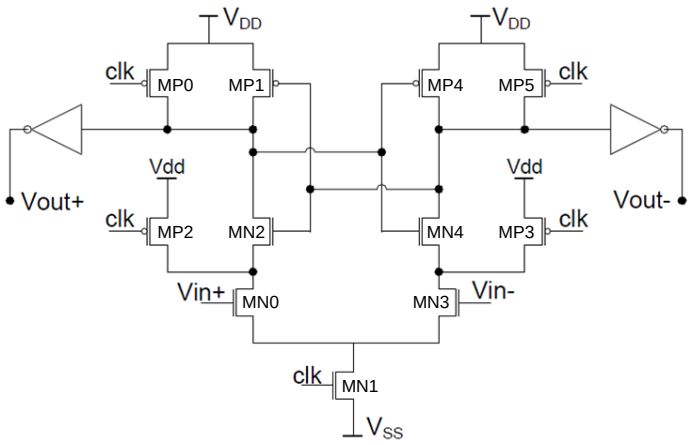


Figure 6-1 – Electrical schematic of the fully-dynamic comparator.

Table 6-1 – Comparator sizing (130 nm process).

Devices	Sizes		
	Width	Length	Gates
Inverter MN0	4 µm	120 nm	2
Inverter MP0	12 µm	120 nm	4
MN0, MN3	4 µm	260 nm	2
MN1	8 µm	640 nm	4
MN2, MN4	6 µm	120 nm	4
MP0, MP2, MP3, MP5	820 nm	120 nm	1
MP1, MP4	12 µm	120 nm	4

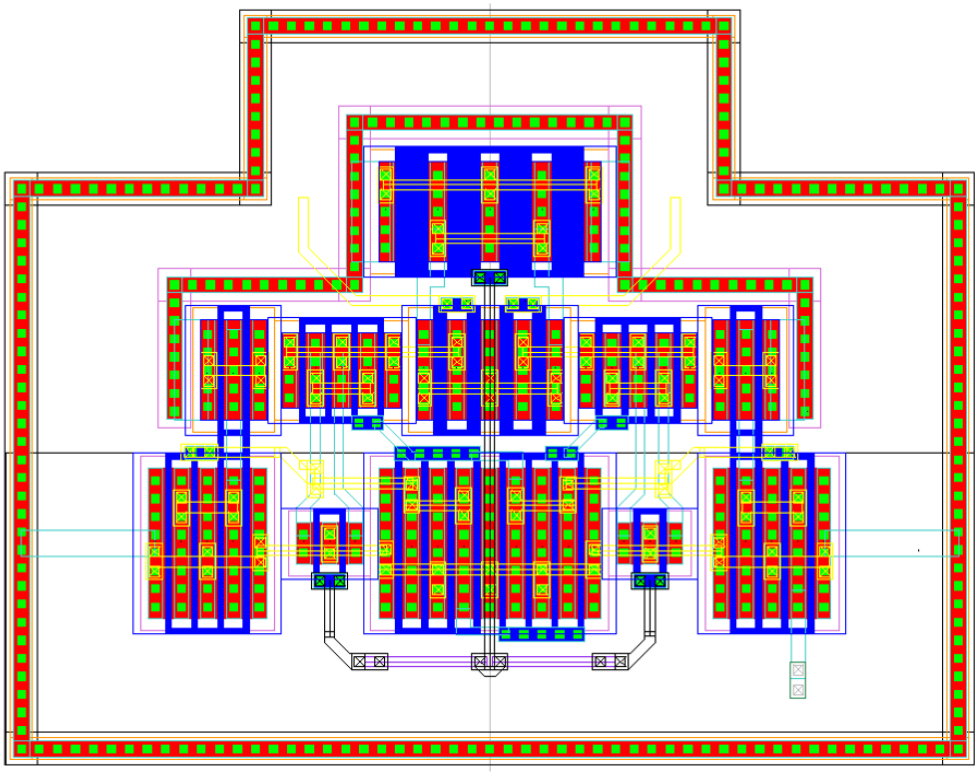


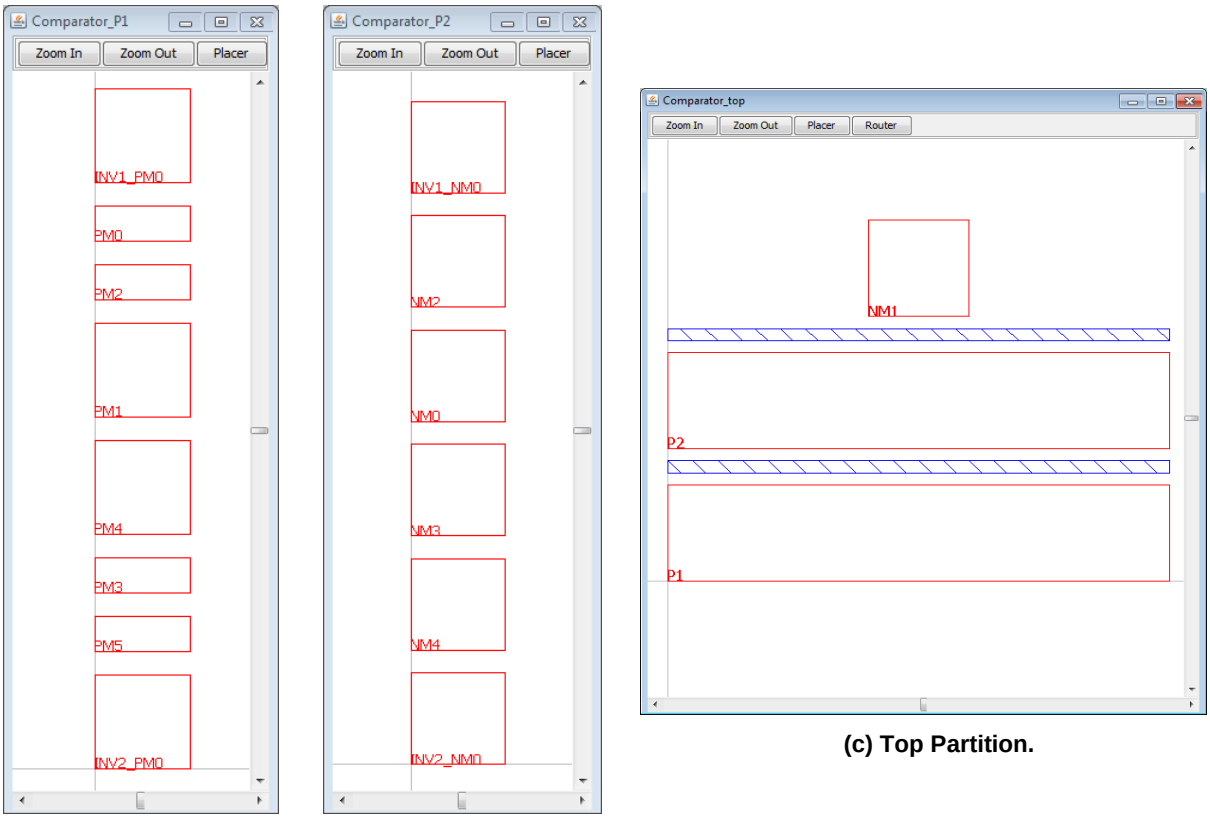
Figure 6-2 – Comparator handmade layout (UMC 130 nm process).

6.1.1 Template

The proposed design flow starts with the template's definition. To incorporate design strategies used by designer when performing handmade layout, the template guidelines were derived from handmade layout of Figure 6-2. Besides being the support to extract the expert's knowledge, the handmade layout also provided an evaluation measure for the quality of the target layout. Using these inputs, the proper design-kit and template, LAYGEN II is used to automatically generate the layout.

The first step in defining the template is to identify possible inner-templates. In this circuit, the hierarchy partitioning used is the one depicted in Figure 6-3. The partition 1 includes all the PMOS transistors with sources connecting to *vdd* potential, while partition 2 includes the remaining NMOS transistors with the exception of transistor MN1, which was moved to top cell for symmetry purposes. Guard ring is only requested for the top cell.

The routing connectivity was equally provided, and all nets suited to be symmetric were marked as such. The *clock* was labeled as noisy and, without using any measures that proved it, the *gnd* net and connections from transistor MN1 to transistors MN0 and MN3 were marked as sensitive. The previous attributions were used only to identify the behavior of optimization kernel in the presence of special nets. Also, two power nets for *vdd* potential, placed bellow the circuit and *gnd* potential, placed above were defined. All necessary routing of the sub-templates and intra-templates is performed in the top cell, in total the connectivity for 36 wires was provided, distributed by 9 different nets.



(a) Partition 1. **(b) Partition 2.** **(c) Top Partition.**

Figure 6-3 – Comparator template hierarchy.

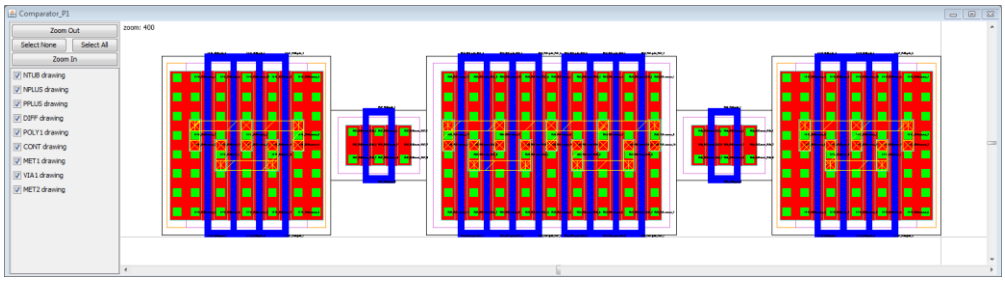
6.1.2 Layout Generation

First all placement task are performed, from the inner templates to the top cell, and only then the routing tasks are executed.

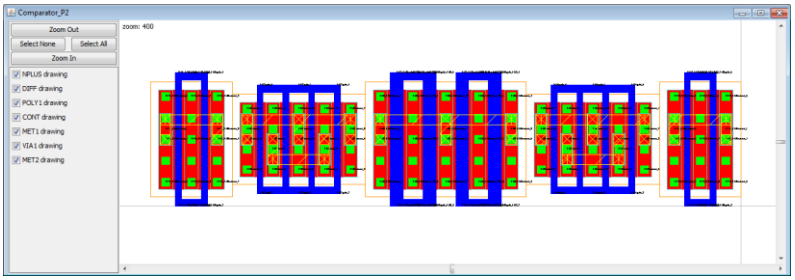
6.1.2.1 Placer

Using the template partitioning of Figure 6-3 and the parametric module generator of Appendix C the placer starts by generating the inner templates first. The layout generated for partition 1 is shown in Figure 6-4 (a), while partition 2 in Figure 6-4 (b) and, finally, the top cell in Figure 6-4 (b), which is placed only after all the sub partitions are available. All the pre and post processing task are performed.

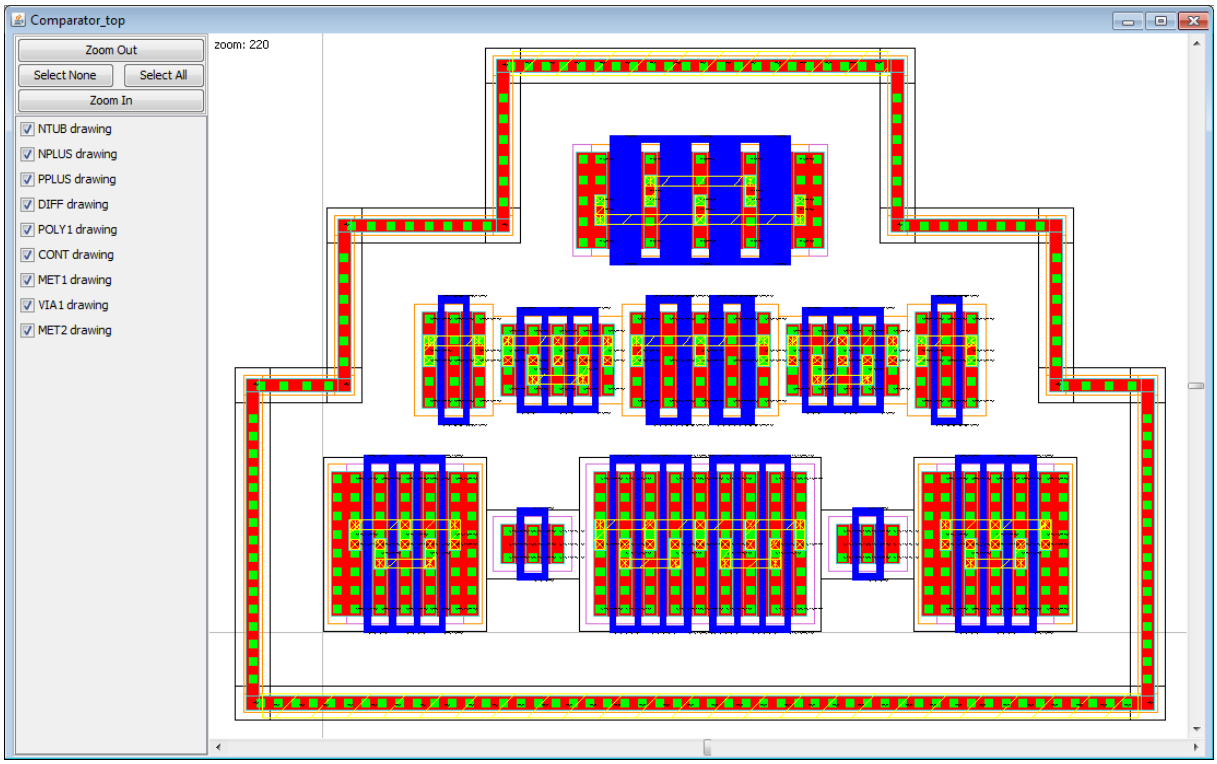
In partition 1 transistors pairs PM0 and PM2, PM1 and PM4, and also PM3 and PM5 were automatically merged; and for the partition 2 the pair NM0 and NM3. The placement is totally symmetric, even the biasing considerations included in the transistors. Although partition 1 and 2 were firstly generated vertically, since the layout representation structure only allow for vertical symmetry axis, the partitions were then rotated to fit correctly in the top cell. A guard ring with N-Well was automatically adjusted to the obtained floorplan.



(a) Partition 1.



(b) Partition 2.



(c) Top Partition.

Figure 6-4 – Floorplans obtained (UMC 130 nm process).

6.1.2.2 Router

The floorplan of Figure 6-4 (c) is used as starting point for routing optimization. The optimization kernels have a population of 256 elements, both the Phase I as the Detailed Routing were optimized for 200 generations. The convergence of the algorithm strongly depends on the greedy initializations performed for each element. So, populations with higher number of elements present more diversity and consequently fewer generations required to achieve feasibility. For complex routing problems, the number of elements should be kept above the 128 elements, allied to a considerable number of

generations. As characteristic from the current algorithm, the objectives are only taken into account when all the constraints are fulfilled, so there has to be ensured that the evolutionary kernel optimize the solutions for a substantial number of generations after the feasibility is attained.

The results were generated for a mutation rate of 3% and a crossover rate of 90%. The mutation rates for the current optimization problem superior to 5% proved to disperse the elements of the population, which hinders the algorithm convergence, so this value need to be kept significantly low. For its part, the crossover rates can be raised up to 100% without deteriorating the performance of the algorithm.

Four constraints were used in the optimization, namely short circuits, design rule violations, illegal crossing between noisy and signal, and finally noisy or signal nets running on top of the active area of the devices. The three objectives used were minimizing the wiring length considering the conductor cost associated to each segment, minimizing the number of contacts used and maximizing the distance between noisy and signal nets. The obtained solution and consequently final layout is presented on Figure 6-5.

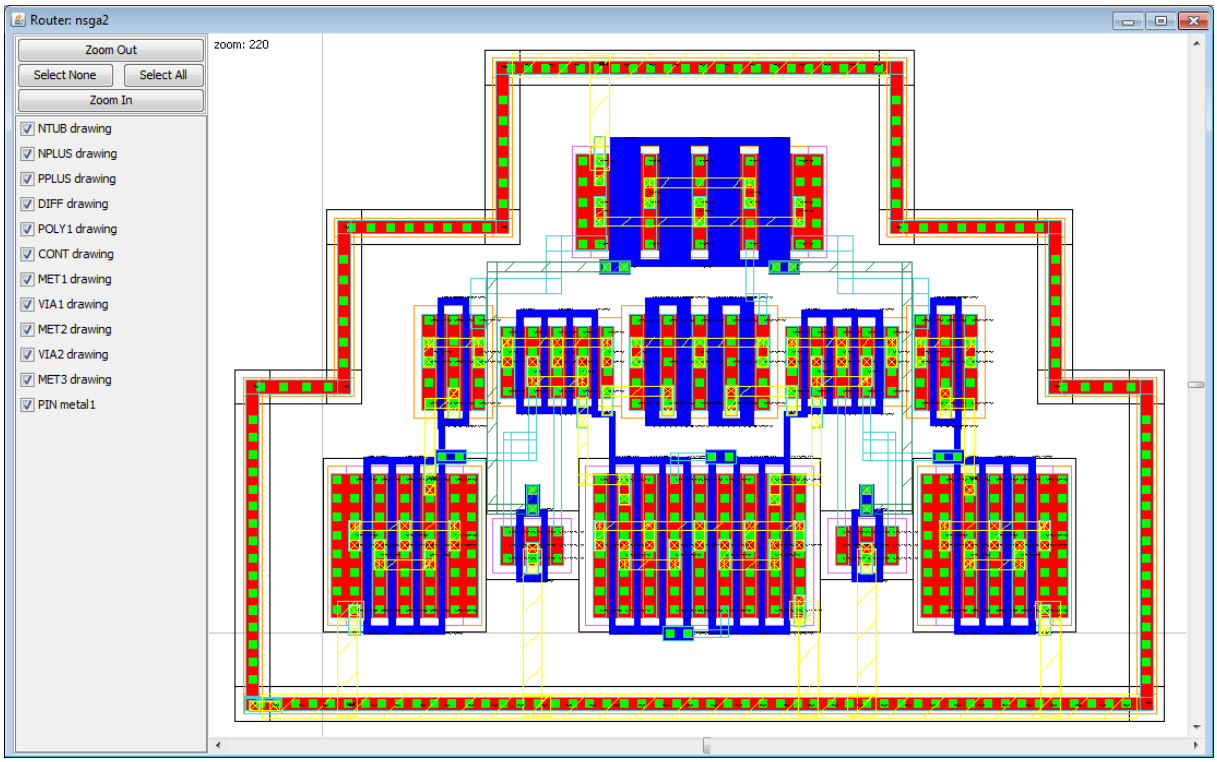


Figure 6-5 – Automatically generated layout (UMC 130 nm process).

Table 6-2 summarizes the execution times for all the template hierarchy of the comparator, its ease to see that the placer execution times are almost instantaneous when compared to the router, which dominated more than 99% of the computation time. The automatic generation times (approximately 281 seconds) are obviously negligible when compared to the manual design, for a fair comparison is necessary to estimate the complete design of a circuit, which encompasses the template setup and guidelines adjustment times. The complete design of this fully-dynamic comparator is difficult to measure since his development accompanied the development of the tool. The new implementations

have been iteratively integrated and tested in the current circuit, leaving the considerations about complete design times to the next test case presented in this chapter.

Table 6-2 – Execution times summary.

Template	Placement time	Routing time		Total
		Phase I	Detailed Routing	
Partition 1	83 ms	Not performed.		83 ms
Partition 2	39 ms	Not performed.		39 ms
Top Partition	50 ms	97,571 s	183,286 s	280,907 s

A direct loss when comparing to the manual design is the lack of 45 degrees wires. Although they might not be essential in the current circuit for a 130 nm process, for radio frequency circuits and smaller design processes the 45 degrees wires are a particular request of the designers, so they are therefore addressed as future implementation in the proper section. Nevertheless it does not invalidate the conclusions about the validity of the proposed design approach.

It is important to notice the symmetry nets in the obtained layout and also many of the initial greedy considerations were kept. Only the three first levels of metals from the eight available in the current technology were used in the generated solution, a detailed perspective is presented in Figure 6-6.

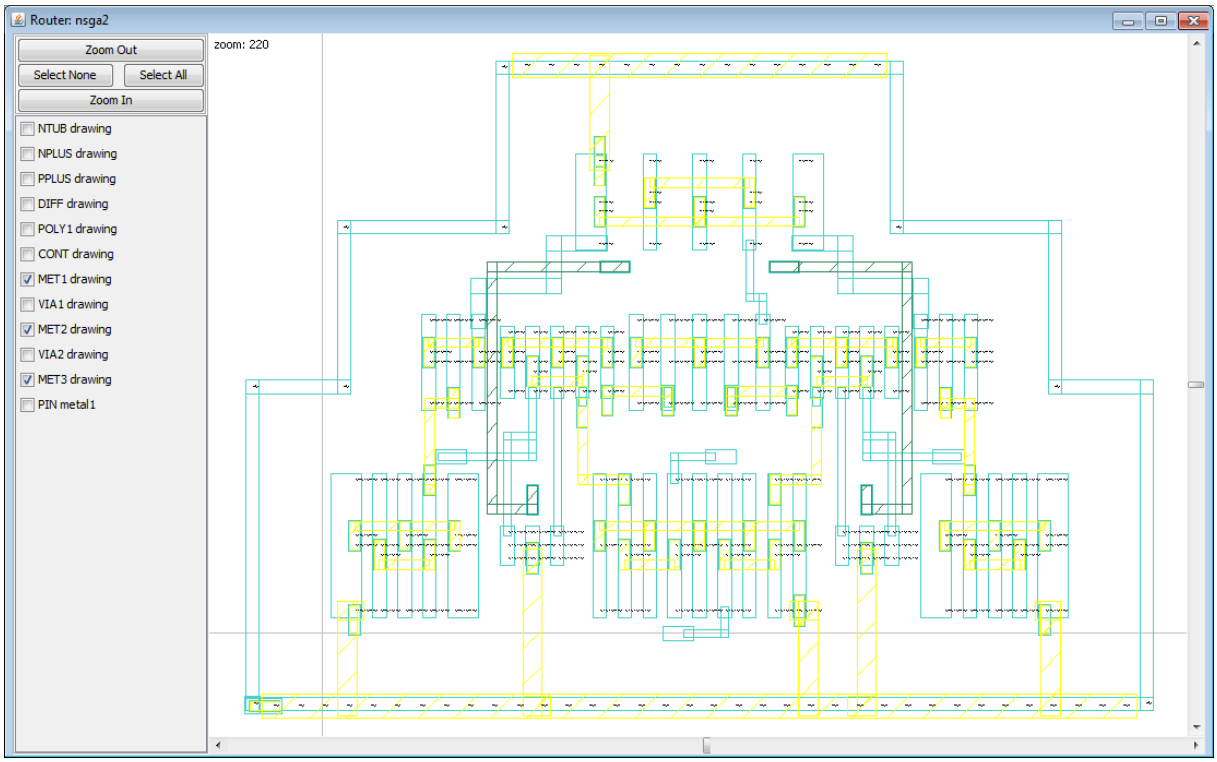


Figure 6-6 – Layout obtained with only metal 1, metal 2 and metal 3 layers set as visible (UMC 130 nm process).

6.1.3 Validation

The GDSII file was generated for the top cell and the results were successfully validated by Calibre® DRC tool, the obtained report for front-end-of-line (FEOL) layout rules is presented on Figure 6-7. The only result obtained is due to not having been considered filling over the generated layout. The back-end-of-line (BEOL) layout rules verification report was obtained, equally presenting the coverage minimum rates error, and die corner rules for metals, which can be ignored.

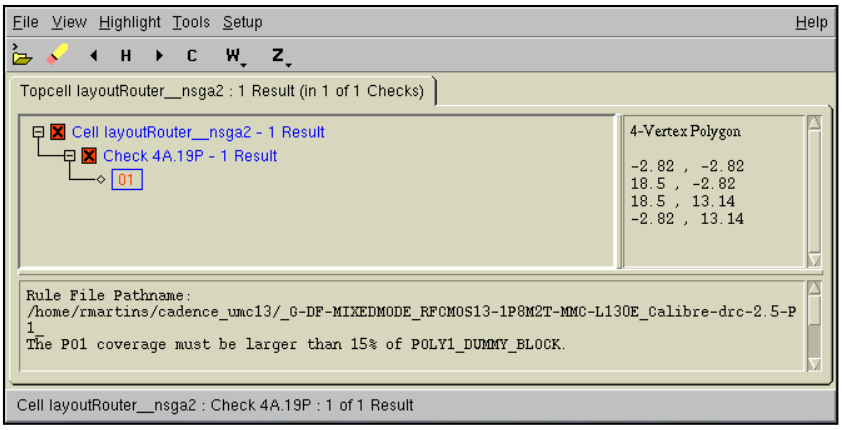


Figure 6-7 – Calibre® DRC report for the automatically generated comparator.

Once the DRC verification is concluded, the next step is to verify if the layout matches the electrical schematic. Calibre® LVS was performed and the obtained report is presented on Figure 6-8.

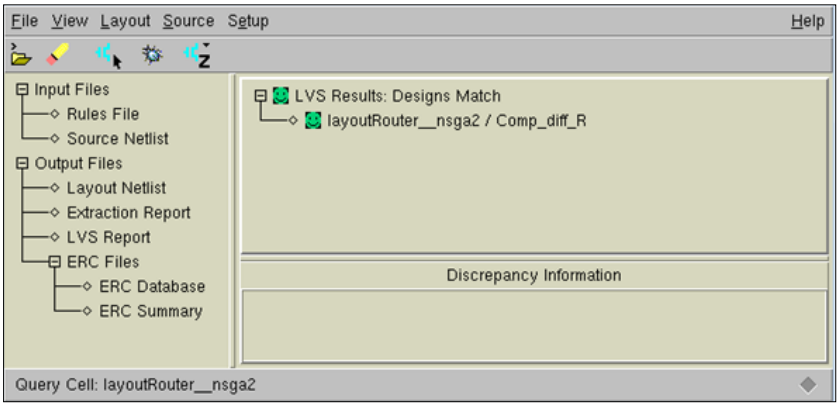


Figure 6-8 – Calibre® LVS report for the automatically generated comparator.

With the DRC and LVS verified, it is necessary to create an extracted view of the layout including the parasitic elements, e.g., resistances/capacitances of layout traces and coupling capacitances, to perform post-layout simulations. Transient simulations for the electrical schematic, extracted netlist from the handmade layout, and extracted netlist from the automatically generated layout were performed and the outputs are presented in Figure 6-9.

The presented result is the output potential of the comparator for a state change in a rising edge of the clock. It is relevant to notice that both handmade and automatically generated layouts present similar response times, however, it is not intended to take any conclusions about performances with the simulation of Figure 6-9. The objective was to prove the use of LAYGEN II to automatically generate

layouts, that can be validated in an industrial grade DRC and LVS tool, and successfully perform post-layout simulations. The achieved extracted comparator would require a complete characterization, in order to confirm the performance results of all specifications of the original design. This characterization is out of the scoop of the work presented in this report.

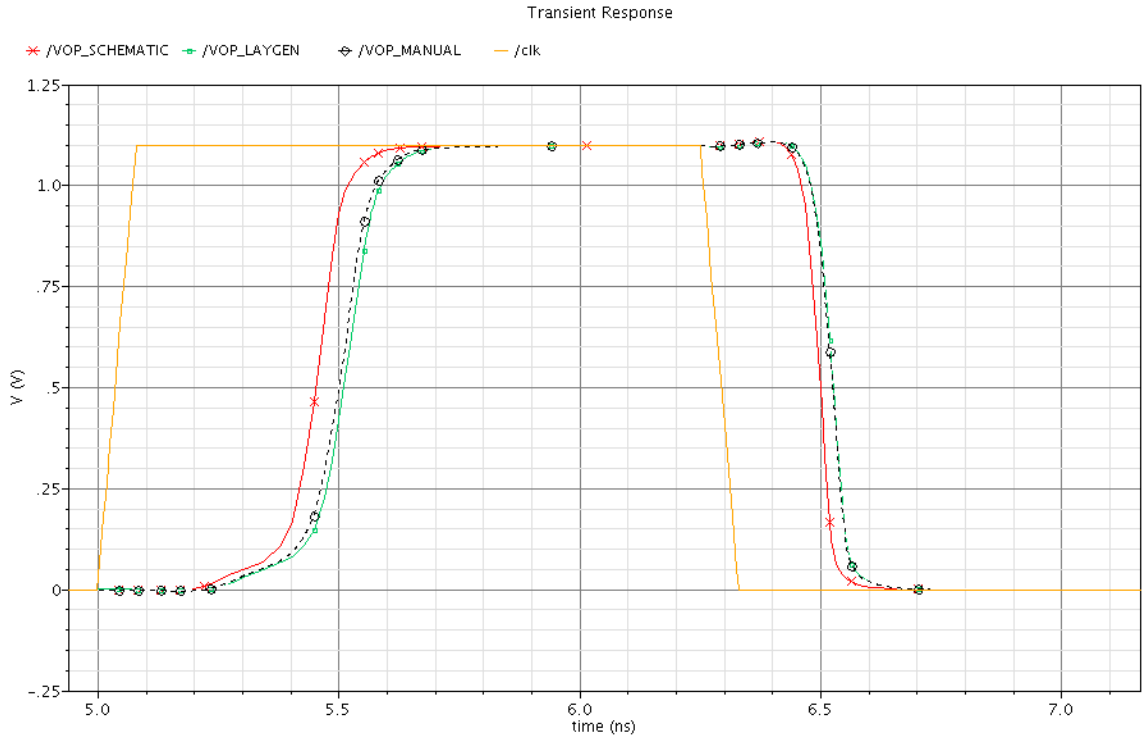


Figure 6-9 – Comparator simulation: Schematic (red), Handmade layout (black), LAYGEN II layout (green).

6.2 Case Study II – Single-Ended Folded Cascode Amplifier

The second case study of this chapter is a single ended folded cascode OpAmp tested with FRIDGE synthesis tool [60], the circuit schematic is shown in Figure 6-10.

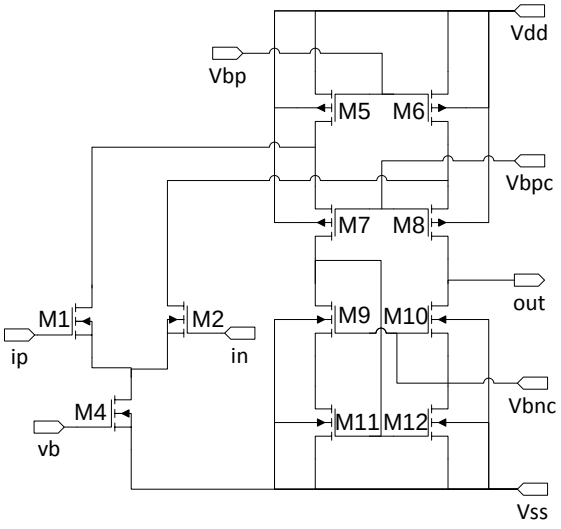


Figure 6-10 – Electrical schematic of the single-ended folded cascode amplifier.

A previous sizing task was performed by GENOM-POF whose optimization objectives were minimizing the area and maximizing the gain. The gain versus area POF obtained is presented in Figure 6-11 and three sizing solutions were selected. The sizing solution which minimizes the area was selected as the first point from the obtained POF. A second solution was randomly selected from the remaining solution space, with the only restriction that the area was substantially greater than the area of the first solution (about 60% higher). The third and last solution is the solution that maximizes the area, more than 4 times the relative area from the first sizing selected. Those sizing solutions are presented in Table 6-3.

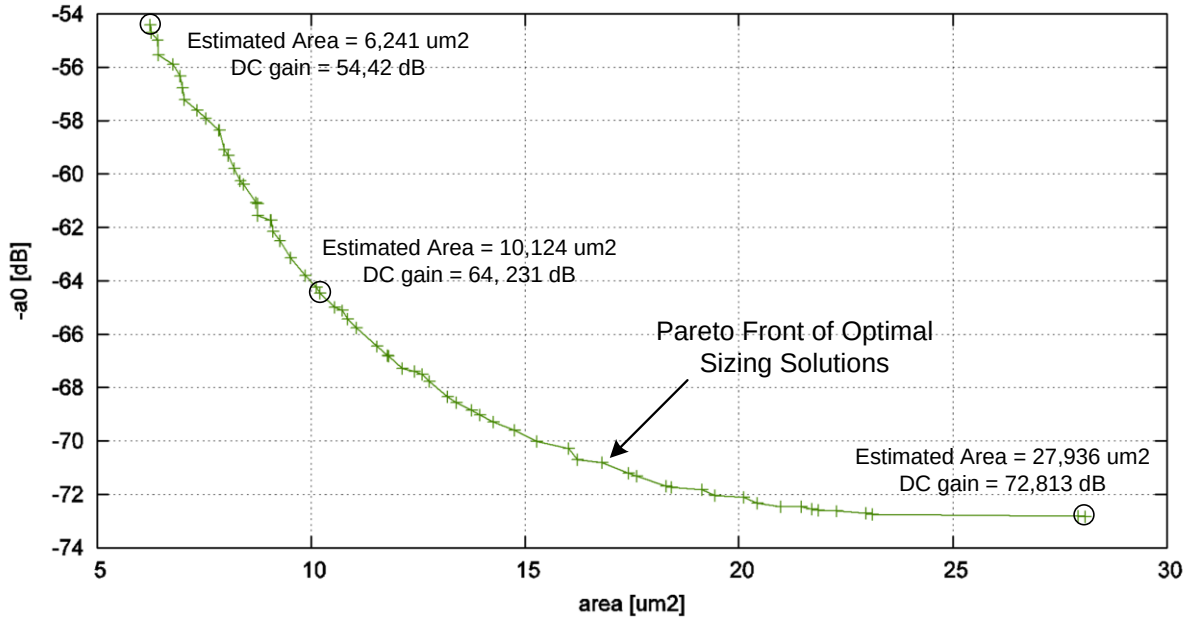


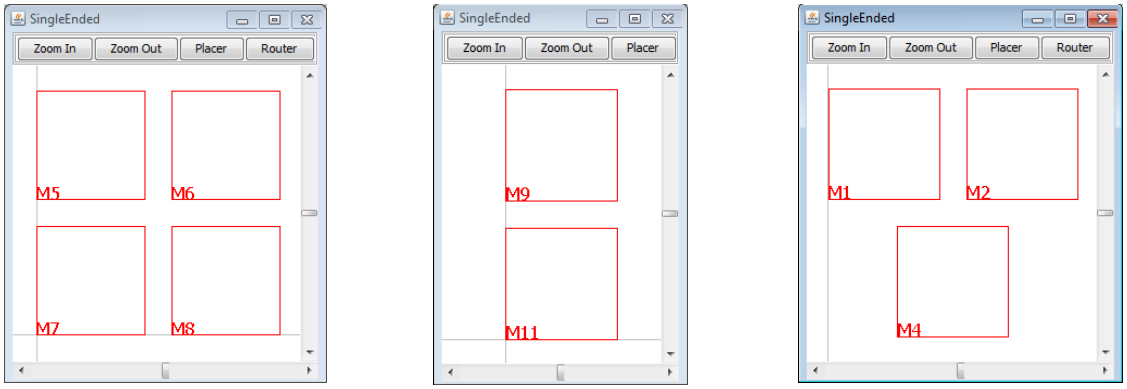
Figure 6-11 – POF obtained during sizing task, gain [dB] versus estimated area [µm²].

Table 6-3 – Devices sizes and objectives attained during sizing task for the amplifier, using GENOM-POF for a 130 nm process.

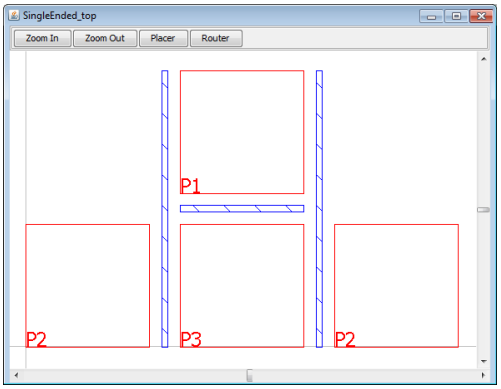
Objectives	Devices	Sizes	
		Width	Length
Estimated Area = 6,241 μm^2 DC gain = 54,42 dB	M1, M2	14,67 µm	480 nm
	M4	3,84 µm	530 nm
	M5, M6	13,4 µm	140 nm
	M7, M8	17,77 µm	370 nm
	M9, M10	5,72 µm	310 nm
	M11, M12	2,53 µm	470 nm
Estimated Area = 10,124 μm^2 DC gain = 64, 231 dB	M1, M2	13,2 µm	490 nm
	M4	11,33 µm	400 nm
	M5, M6	27,29 µm	290 nm
	M7, M8	36,86 µm	530 nm
	M9, M10	17,61 µm	540 nm
	M11, M12	7,61 µm	730 nm
Estimated Area = 27,936 μm^2 DC gain = 72,813 dB	M1, M2	41,22 µm	760 nm
	M4	69,09 µm	670 nm
	M5, M6	153,37 µm	570 nm
	M7, M8	249,84 µm	780 nm
	M9, M10	55,29 µm	790 nm
	M11, M12	12,21 µm	800 nm

6.2.1 Template Hierarchy

The first set of devices sizes of Table 6-3 was used to define the guidelines of the floorplan. The partition 1 encompasses all the PMOS transistors of the circuit; partition 2 includes two PMOS of the cascode, while partition 3 corresponds to the differential pair. Partition 2 is instantiated twice since the cascode requires two plus two symmetric NMOS transistors. The hierarchy partitioning used is the one depicted in Figure 6-12.



(a) Partition 1. **(b) Partition 2.** **(c) Partition 3.**



(c) Top Partition.

Figure 6-12 – Comparator template hierarchy.

The routing connectivity was equally provided, and all nets suited to be symmetric were marked as such. Two power nets for *vdd* potencial, placed above the circuit and *gnd* potential, placed bellow were defined. All necessary routing of the sub-templates is performed in the top cell, in total the connectivity for 25 wires was provided, divided into 12 different nets.

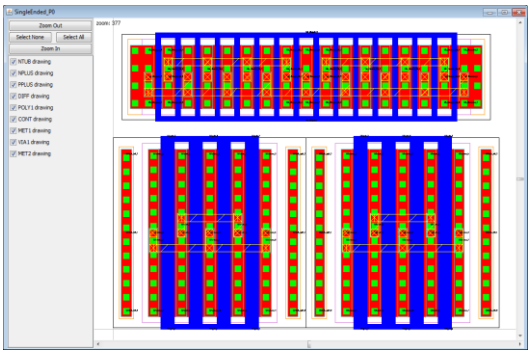
6.2.2 Layout Generation

As stated, first all bottom-up placement tasks are performed and then routing task is executed.

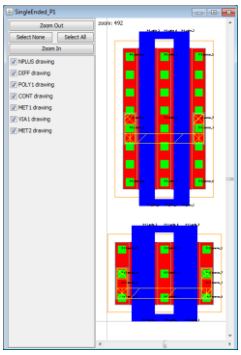
6.2.2.1 Placer

Using the template partitioning of Figure 6-12 and the parametric module generator of Appendix C the placer starts by generating the inner templates first. The layout generated for partition 1 is shown in Figure 6-13 (a), while partition 2 in Figure 6-13 (b) and partition 3 in Figure 6-13 (c). The layout for the top partition is presented in Figure 6-13 (d), which is placed only after all the sub partitions are available. All the pre and post processing task are performed.

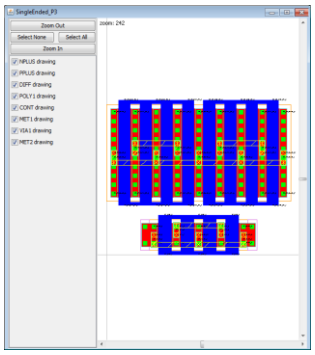
In partition 1 transistors pairs M5 and M6 were automatically merged and for the partition 3 the pair M1 and M2. The placement obtained is totally symmetric and a guard ring with N-Well was automatically adjusted to the obtained floorplan.



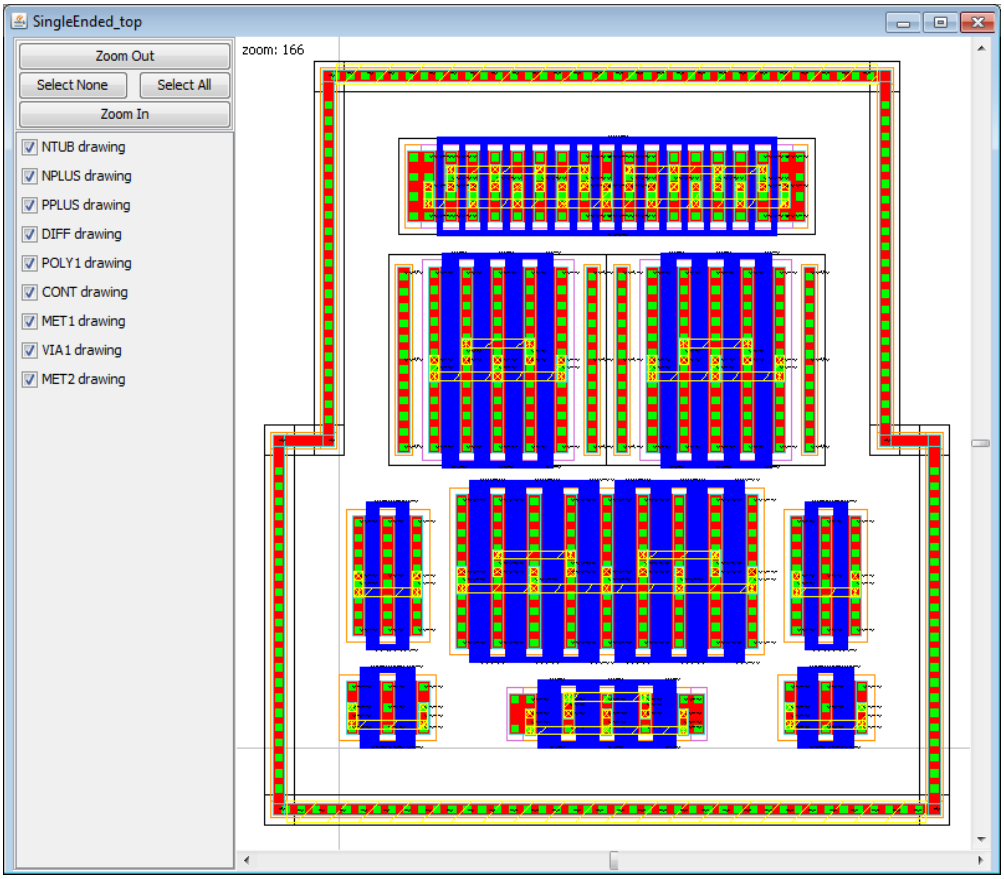
(a) Partition 1.



(b) Partition 2.



(c) Partition 3.



(d) Top Partition.

Figure 6-13 – Floorplan obtained for the top partition (UMC 130nm process).

6.2.2.2 Router

The floorplan of Figure 6-13 is used as starting point for routing optimization. The optimization kernels have a population of 128 elements, both the Phase I as the Detailed Routing were optimized for 200 generations. The results were generated for a mutation rate of 3% and a crossover rate of 90%. The obtained solution and consequently final layout is presented on Figure 6-14. Table 6-4 summarizes the execution times for all the template hierarchy of the circuit. Two constraints were used in the

optimization, namely short circuits and design rule violations, the two objectives used were minimizing the wiring length considering the conductor cost associated to each segment and the number of contacts used.

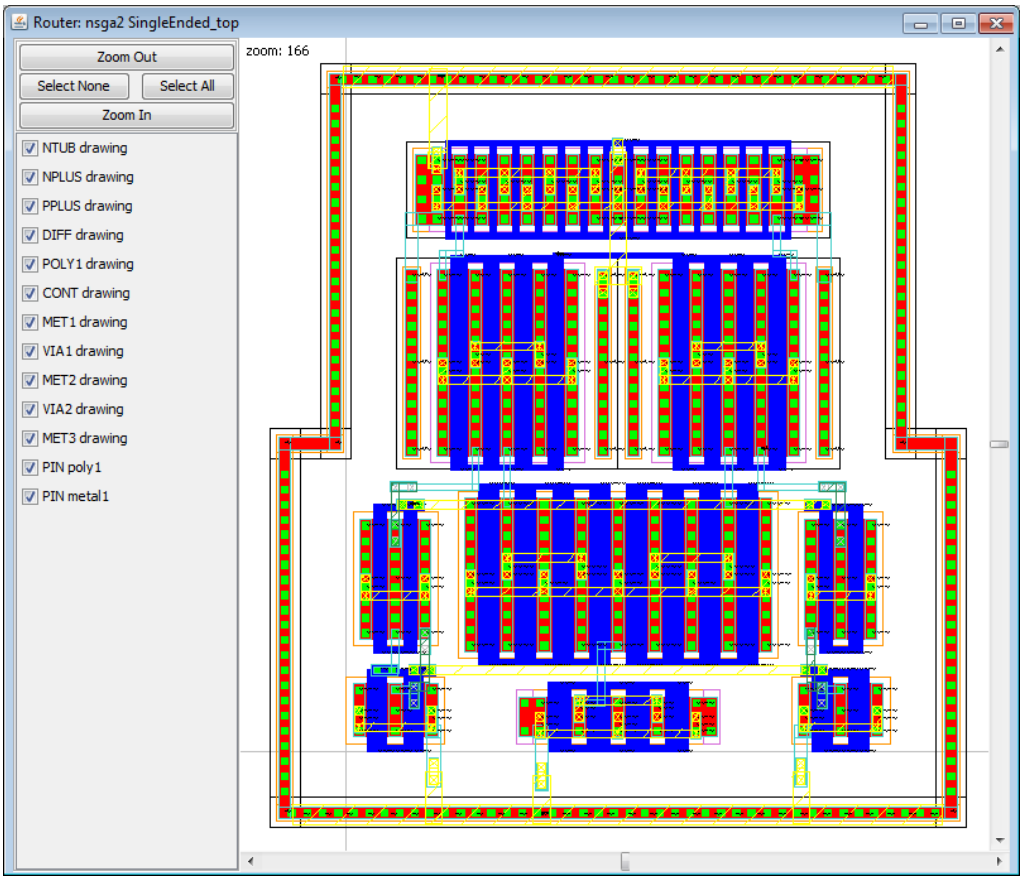


Figure 6-14 – Automatically generated layout for the first sizing solution (UMC 130 nm process).

Table 6-4 – Execution times summary.

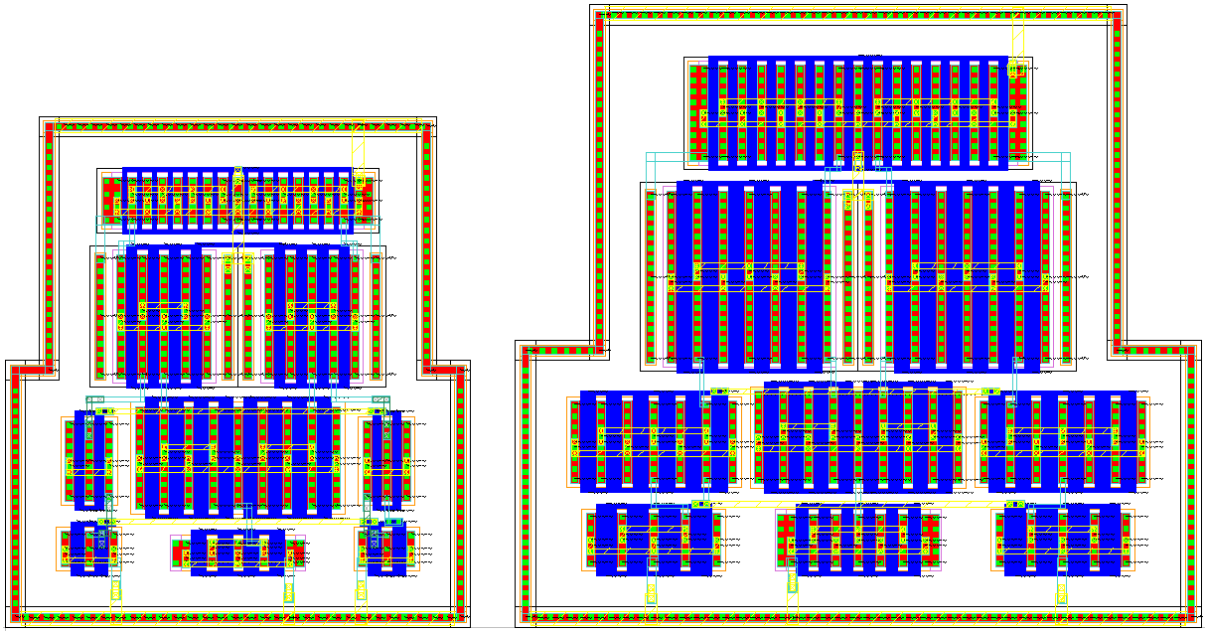
Template	Placement time	Routing time		Total
		Phase I	Detailed Routing	
Partition 1	64 ms	Not performed.		64 ms
Partition 2	16 ms	Not performed.		16 ms
Partition 3	15 ms	Not performed.		15 ms
Top Partition	41 ms	40,438 s	71,060 s	111,539 s

The total template description was done in approximately 2 hours, including the iterative process of refining the guidelines and the automatic generation was performed in less than 112 seconds. The presented results were successfully validated in Calibre® DRC tool.

6.2.3 Retargeting for Different Sizes

The low-level design flow proposed in this work is intended to increase the design reusability, without loss of the designer expertise, but also without overwhelming the designer. In this section the same template was used, but with different sizes for the devices. The new sizes are the second solution presented in Table 6-3. This example took approximately the same computational time than the

previous example to be generated and the result is presented in Figure 6-15 (b). The complete retargeting operation took less than 15 minutes to perform, including the computational effort to generate the solution. The only change in the template was made to reflect the new sizes of the modules, no changes were performed in the routing template (connectivity and constraints).

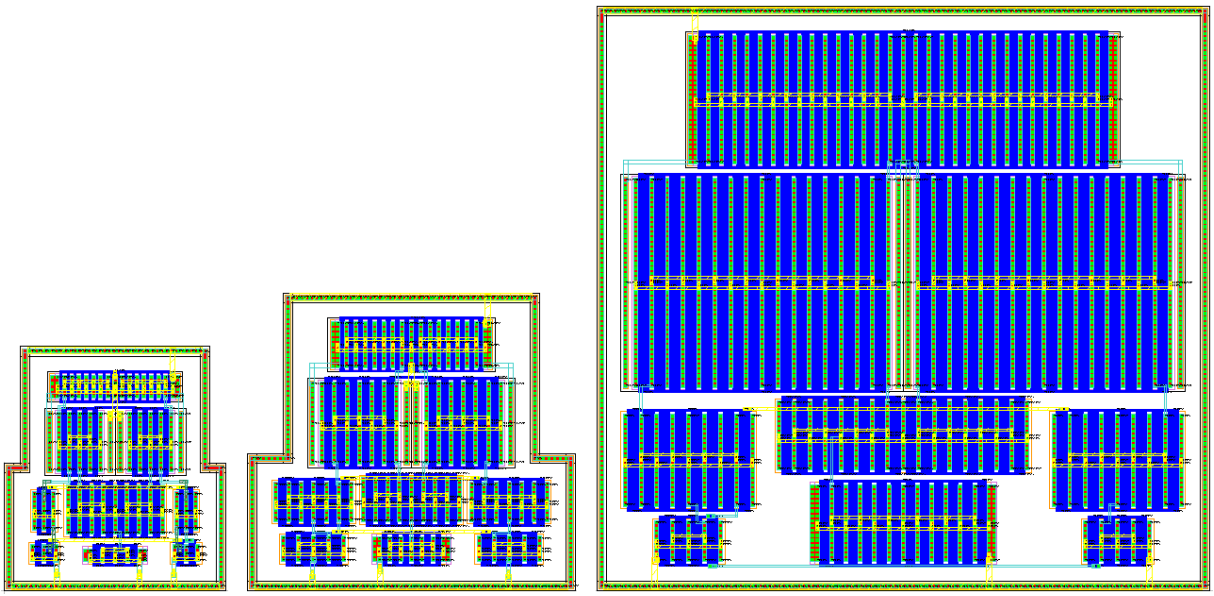


(a) Automatically generated layout for the first sizing solution.

(b) Automatically generated layout for the second sizing solution.

Figure 6-15 – Amplifier retargeting (UMC 130 nm process).

In Figure 6-16 (b) is presented the retargeting for the third sizing solution of Table 6-3, which represents a huge change in the devices sizes comparatively to the other solutions. The layouts automatically generated for the two previous sizing solutions are placed at the same scale in Figure 6-16 (a). Again, no changes were performed in the template except the devices sizes.



(a) Layouts of the first and second sizing solutions.

(b) Automatically generated layout for the third sizing solution.

Figure 6-16 – Amplifier retargeting (UMC 130 nm process).

Considering that no change was performed in the high level floorplan, connectivity or constraints, for any of the retargeting operations above, the results are promising. However, the template was optimized for the first sizing solution and different topological relations between cells could be more suited for the remainder sizing solutions. The same template may not yield the best topological relations between cells as devices sizes are changed, thought as demonstrated, the connectivity is always valid.

In the following section, this same template is going to be used for retargeting, but for a different technology.

6.2.4 Retargeting for Different Technology

The ability to support multiple technologies is mandatory if one attempts to achieve the maximum flexibility on retargeting operations. To demonstrate the technology independence of the proposed design flow, the template is going to be retargeted for the 350 nm process of Appendix D. A previous sizing task was performed by GENOM-POF whose optimization objectives were minimizing the area and power, and maximizing the gain. The sizing solution which minimizes the area was selected from the obtained POF, and is presented in Table 6-5.

Table 6-5 – Devices sizes and objectives attained during sizing task for the amplifier, using GENOM-POF for a 350 nm process.

Objectives	Devices	Sizes	
		Width	Length
Estimated Area = 6,8407 μm^2 Power = 0,347 mW DC gain = 49,17460 dB	M1, M2	20 µm	520 nm
	M4	12 µm	350 nm
	M5, M6	8 µm	420 nm
	M7, M8	11 µm	410 nm
	M9, M10	2 µm	480 nm
	M11, M12	2 µm	870 nm

The only changes in the templates were made to reflect the new sizes of the modules, and the XML files header changed to the designation of the different technology. No changes were performed in routing template and this retargeting task was performed in less than 10 minutes. This process obviously assumes that the desired technology design kit is available. If available, the migration process is actually pretty simple. The obtained result from this retargeting operation is presented in Figure 6-17.

Although the technology design kit is outdated when compared to the 130 nm technology design kit, lacking on merged transistors and symmetry in all devices, it does not invalidate the obtained results to proof the concept. The presented results were successfully validated in Calibre® DRC tool.

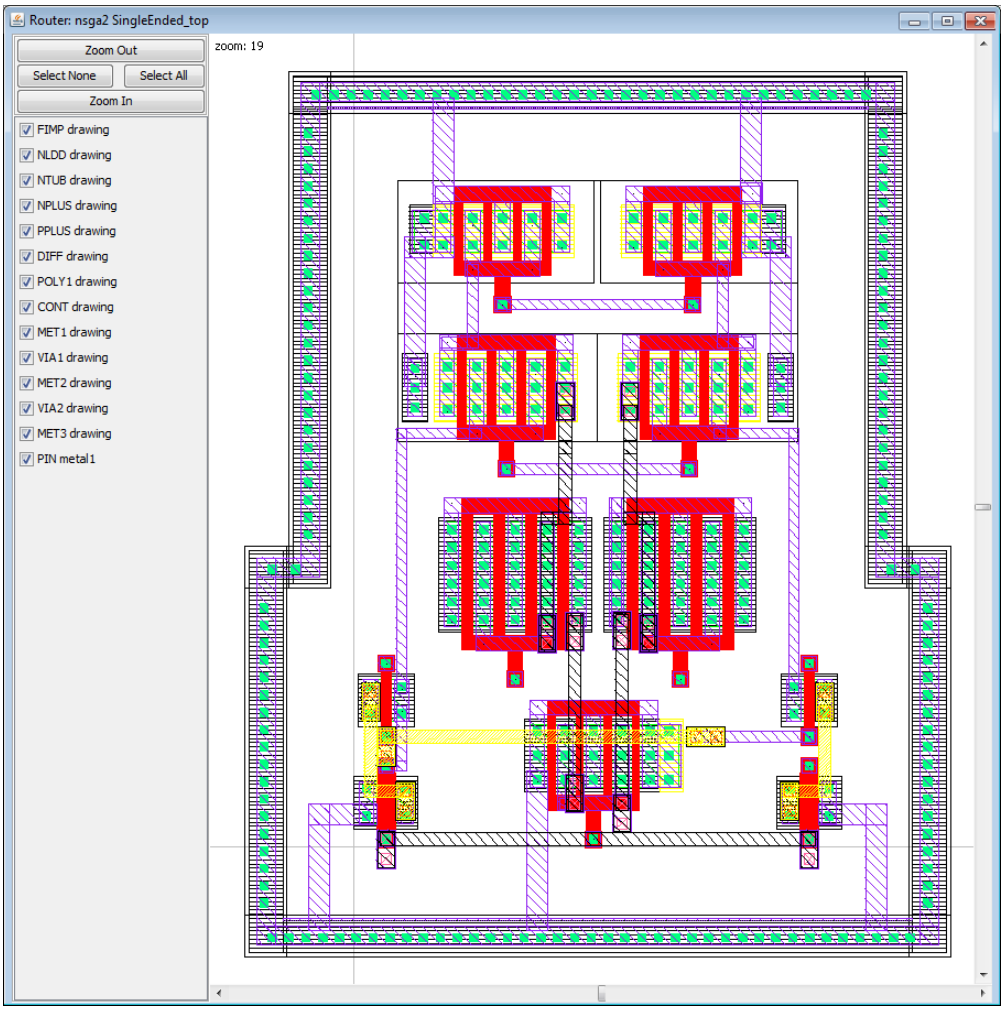


Figure 6-17 – Amplifier retargeting for different technology (AMS 350 nm process).

6.3 Conclusions

In this chapter, two test cases were addressed to show the capabilities of the tool. The first example, a fully-dynamic comparator, was used to compare the LAYGEN II results with a handmade layout and perform a set of validations (DRC, LVS and post-layout simulation). The second example, a single-ended folded cascade amplifier, was used to explore the retargetability characteristics of the proposed methodology.

In the first test case the template was easier to design, because it was only required to translate the information from the handmade layout to the template. Designing the template from scratch and refining the high level floorplan is obviously a more time consuming task. When compared to manual design, the use of LAYGEN II implies some additional initial work to setup the template. However as shown in the second example, after the template is available the proposed design flow highly increases the reusability of the design.

The design tasks that are more time consuming are the development of the technology design kits. A retargeting for a 350 nm process was performed to show the versatility of technology migrations. Once

the template and the target technology design kits are available, technology migration processes are performed for the same or different specifications within minutes of computational time.

Chapter 7 Conclusions and Future Work

This chapter presents the closing remarks, and then future directions for the continuous development of LAYGEN II are outlined.

7.1 Conclusions

The proposed methodology for the automatic generation of analog IC layouts was proved by the implementation of a tool, LAYGEN II, which is able to generate robust layout solutions. LAYGEN II outstands from the remaining tools presented in the state-of-the-art on analog design automation of Chapter 2, by automatically generating flexible routing solutions, using only connectivity, that are validated in a commercial tool widely accepted in the industry, Calibre® DRC.

The tool is a combination of template-based and optimization-based approach, allowing the designer to provide layout guidelines that are used as a first cut solution allowing an intelligent pruning of the design space and, therefore, reducing the overall computational effort required by the evolutionary optimization kernel. Moreover, the use of a technology independent template, that creates an abstraction level between physical representation and designer's knowledge, introduce flexibility to the high level guidelines and easies the migration of designs to different IC technologies.

The tool potential has been proved for the two test cases presented. The hierarchical and modular nature of the developed approach allows the generation of large circuits layout by scaling the problem into different sub-templates, thus, dividing the problem into smaller ones. The introduction of an automatic generation during routing, independently from the floorplan attained, allows the designer to explore different layout topologies without the effort of defining new templates. The small computational times achieved for each automatic generation, reinforce the integration of the tool in the bottom-up physical synthesis path of an automatic analog design flow.

As this implementation stage, it is unlikely that automatically generated layouts would achieve better performance than handmade layouts. This limits the usage of the automatic tool in cells where the performance is extremely high, but for simpler cells or macro-cell place and route, this approach presents a highly effective design flow. The generated target layout must pass DRC and LVS validations, and should not introduce extreme parasitic effects. While the parasitic are not handled at layout level, they are fed to an automatic circuit synthesizer that will re-size the circuit components to compensate layout parasitics, closing this way the traditional analog IC design flow.

Analog IC layout design automation is not a trivial matter and although LAYGEN II can automatically generate layouts, there is still much to evolve. This implementation focused on first settling an industrial-level layout synthesis process, and do not promptly take into account all precision parameters characteristics of the physical designs. Some limitations of the current implementation and

relevant future improvements were identified. In the next section, these future enhancements are discussed and some concrete solutions presented.

7.2 Future Work

Starting with LAYGEN II's main purpose of creating an abstraction level between designer and technology details, the introduction of other automated abstraction levels above may increase the design automation level. The current implementation is gradually moving away from the template-based approach, but never tacking from the designer the ability to control the automatic generation.

The introduction of routing spaces constraints during placement reduce routing limitations that could arise from placement, however the generation flow placement-then-routing, still lacks the introduction of routing criteria during placement. Sometimes the minimum distances allowed by target technologies to place the devices may not be enough to router many wires. The use of a procedure which includes some measures to estimate the placement influence in routing may be considered in a future implementation.

As mentioned, a template developed for a current set of devices sizes may not yield the best layout if those devices are changed. The currently NSGA-II algorithm used opens a range of possible different implementations, which the most interesting is undoubtedly the extension of the placer module to allow topology exploration rather than strict template based generation. The designer guidelines should continue to be respected, such as symmetry, matching and proximity, but obtaining a POF of possible placements, instead of a single solution drawn directly from the topological relations between cells. An example of the possible POF generated is presented on Figure 7-1. Since the routing connectivity and guidelines are provided independently from the obtained floorplan, the routing can be easily generated for each one of the solutions without requiring more effort by the designer.

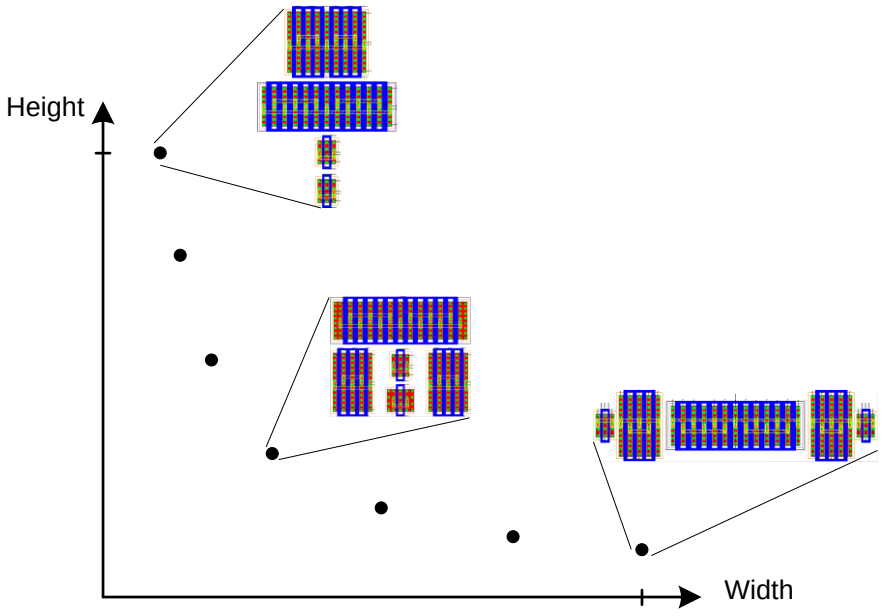


Figure 7-1 – Example of a POF of placements.

The current B*-Tree layout representation although fits relatively well in the current template-based approach, for the topology exploration should be changed to a BSG or TSG-S representation. The possibility for the optimizer to know the relation between any pair of two cells is useful to set more compact floorplans, instead of the rectangular nature of the B*-Tree. It was demonstrated in the previous chapter that the computational times of placer are not of concern.

Since it is desired to perform routing to each solution of the presented POF, an advanced version of router could identify patterns from the previous obtained solutions and use them as the starting point for the following evolutionary algorithms. This could mean a relative improvement on the computational times, since only the optimization phases of the first floorplans routed would take the total execution time, but the following routing tasks would converge faster having a previous optimized solution as the starting point. Theoretically, this would improve significantly the routing quality of the solutions achieved. Since every solution of the POF is composed of the same devices, the topological relation patterns are common between the floorplan solutions.

The parametric module generator of technology design kits should also be extended to interdigitized and common centroid cells. The processing of matched cells is already implemented, only the specific parametric cells are missing. In placement processing, the convex hull approach used for guard ring processing may be adapted for creating unique wells that contain different transistors. At this moment, if the respective wells from transistors are packed together the minimum distances processing do not depart them, but the lack of a unique well for those transistors that do not join was noticed.

For router, the lack of 45 degrees wires was identified and should be implemented in future versions of LAYGEN II. The actual wire structure and connectivity supports 45 degrees segments, as genetic operators can be easily adapted to chromosomes containing these wires. However, the main obstacle is the internal evaluation procedure. The actual implementation of the internal evaluator uses the bounding boxes of shapes contained in the layout to verify the intersections and distances, and these bounding boxes are only rectangles. The introduction of non 90 degree segments in the layout force different geometrical considerations, and computing distances or crossings between lines must be converted to a point-to-point approach.

The endnotes of future developments shall be the improvements required to LAYGEN II be compliant with deep nanometer technologies. The inclusions of transistor dummies in the current modules and to consider more specific ERC, e.g., well proximity effect or time-dependent dielectric breakdown, are just some examples of the challenges that have to be considered as it is being dealt with smaller design processes. Although being applicable to any design process, they become more relevant for the process variability of the sub-100 nm ICs era. While still under development, LAYGEN II has proved capable to assist the designer to obtain a robust first cut design and so is intended to maintain competitive for the deep nanometer processes.

In Table 7-1, a summary of the specifications for the future enhancements suggested is presented, the LAYGEN II specifications are intentionally replicated from Table 2-3. These are identified implementations for the recent future. The use of optimization techniques for analog ICs layout

generation has been demonstrated, however, the potential for further developments is still large. LAYGEN II project is not closed, far from it, this work served to validate the concept and it provides support for further developments. New people will bring new insights in the analog IC layout generation problem, and it is hoped that LAYGEN II will converge to an application suitable for industrial uses. The fact that is being developed so it can be embedded in an automatic analog IC design flow follows the indications of the recent papers published, suggest that soon viable unified sizing/layout solutions arise and settle in the EDA market.

Table 7-1 – General specifications: LAYGEN II versus future enchancements.

	LAYGEN II	Future Enhancements
Placer	**LAYGEN's placer, featuring:** ▪ Biasing considerations; ▪ Automatic merging; ▪ Places devices at the minimum distances allowed by target technology; ▪ Polygon guard rings;	**Optimization-based placer:** ▪ MOEA modified NSGA-II; ▪ Interdigitized and common-centroid cells; ▪ Topology exploration, POF of placements; ▪ Estimation of placement impact on routing;
Router	**Optimization-based router:** ▪ MOEA modified NSGA-II; ▪ Greedy initialization (pins, heuristics and layers); ▪ Geometric and layer shifting operators; ▪ Multiple pin search; ▪ Multiple contacts; ▪ Power lines; ▪ Symmetric wires; ▪ Noisy/Sensitive nets; ▪ Out of bounding box exploration.	**LAYGEN II's router, featuring:** ▪ 45 degrees wires; ▪ Pattern identification from previous optimized routing solutions;
Validation	▪ Multi-thread internal evaluation procedure: short circuit checker, design rule checker and electrical rule checker; ▪ Industrial grade parametric module generators; ▪ Calibre® DRC validation; ▪ Industrial validation up to 130 nm processes.	▪ Non-rectangular internal evaluation procedure; ▪ Internal electrical rule checker extended to specific ERC (WPE, TDDB, etc.); ▪ Extended validation up to 65 nm processes (transistor dummies, etc.).

References

[1] G. G. E. Gielen, “CAD tools for embedded analogue circuits in mixed-signal integrated systems on chip,” *IEE Proceedings on Computers and Digital Techniques,* vol. 152, no. 3, pp. 317 – 332, May 2005.

[2] “International Technology Roadmap for Semiconductors 2009 Edition,” http://public.itrs.net/.

[3] G. G. E. Gielen, and R. A. Rutenbar, “Computer-aided design of analog and mixed-signal integrated circuits,” *Proceedings of the IEEE,* vol. 88, no. 12, pp. 1825-1852, Dec. 2000.

[4] R. A. Rutenbar, “Analog layout synthesis: What’s missing?,” in *Proc. ACM/SIGDA ISPD*, p. 43, Jan. 2010.

[5] H. E. Graeb, *Analog Layout Synthesis: A Survey of Topological Approaches,* Springer, 2010.

[6] H. Habal, and H. Graeb, “Constraint-Based Layout-Driven Sizing of Analog Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 8, pp. 1089-1102, Aug. 2011.

[7] E. Roca, R. Castro-Lopez, and F. V. Fernandez, “Hierarchical synthesis based on pareto-optimal fronts,” *in European Conference on Circuit Theory and Design*, pp. 755-758, Aug. 2009.

[8] "Mentor Graphics," http://www.mentor.com.

[9] "Synopsis," http://www.synopsys.com.

[10] "Cadence Design Systems Inc," http://www.cadence.com.

[11] “gEDA Project,” http://www.gpleda.org.

[12] "Dolphin Integration," http://www.dolphin.fr.

[13] N. Lourenço, M. Vianello, J. Guilherme, and N. Horta, “LAYGEN - Automatic Layout Generation of Analog ICs from Hierarchical Template Descriptions,” in *Conference on Ph.D. Research in Microelectronics and Electronics*, pp 213-216, Jun. 2006.

[14] M. Strasser, M. Eick, H. Gräb, U. Schlichtmann, and F. M. Johannes, “Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 306-313, Nov. 2008.

[15] F. Balasa, S. C. Maruvada, and K. Krishnamoorthy, “On the exploration of the solution space in analog placement with symmetry constraints,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 23, no. 2, pp. 177-191, Feb. 2004.

[16] B. Suman, and P. Kumar, “A survey of simulated annealing as a tool for single and multiobjective optimization,” *Journal of the Operational Research Society,* vol. 57, pp. 1143-60, 2006.

[17] D. F. Wong, and C. L. Liu, “A new algorithm for floorplan design,” in *Proc. 23th ACM/IEEE Design Automation Conference (DAC)*, pp. 101–107, Jun. 1986.

[18] Graeb, H., Balasa, F., Castro-Lopez, R., Chang, Y.-W., Fernandez, F.V., Lin, P.-H., and Strasser, M., “Analog Layout Synthesis - Recent Advances in Topological Approaches,” *Proceedings on Desing, Automation & Test in Europe*, pp- 274 – 279, 2009.

[19] F. Balasa, and K. Lampaert, “Symmetry within the sequence-pair representation in the context of placement for analog design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 19, no. 7, pp. 721-731, HJul. 2000.

[20] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, “Module packing based on the BSG-structure and IC layout applications,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 519–530, June 1998.

[21] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, “An O-tree representation of nonslicing floorplan and its applications,” in *Proc. 36th ACM/IEEE Design Automation Conference (DAC)*, pp. 268–273, 1999.

[22] Y.-C. Chang, Y.-W. Chang, G.-M.Wu, and S.-W.Wu, “B*-trees: A new representation for nonslicing floorplans,” in *Proc. 37th ACM/IEEE Design Automation Conference (DAC)*, pp. 458–463, 2000.

[23] P.-H. Lin, and S.-C. Lin, “Analog placement based on novel symmetry-island formulation,” in *Proc.44th Design Automation Conference (DAC)*, pp. 465–470, 2007.

[24] L. Jai-Ming, and C. Yao-Wen, "TCG: a transitive closure graph-based representation for non-slicing floorplans," in *Proc. 38th Design Automation Conference (DAC)*, pp. 764-769, 2001.

[25] L. Lin, and Yao-Wen Chang, “TCG-S orthogonal coupling of P-admissible representations for general floorplans,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 5, pp. 968-980, 2004.

[26] K. Krishnamoorthy, S. C. Maruvada, and F. Balasa, “Topological placement with multiple symmetry groups of devices for analog layout design,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2032–2035, May 2007.

[27] S. Koda, C. Kodama, and K. Fujiyoshi, “Linear programming-based cell placement with symmetry constraints for analog IC layout,” *IEEE Transactions on Computer-Aided Design*, vol. 26, no. 4, pp. 659–668, April 2007.

[28] P.-H. Lin, and S.-C. Lin, “Analog placement based on hierarchical module clustering,” in *Proc. 45th ACM/IEEE Design Automation Conference (DAC)*, pp. 50–55, June 2008.

[29] P.-H. Lin, H. Zhang, M. Wong, and Y.-W. Chang, “Thermal-driven analog placement considering device matching,” in *Proc. 46th ACM/IEEE Design Automation Conference (DAC)*, pp. 593–598, Jul. Jul. 2009.

[30] V. Meyer, “ALSYN: Flexible rule-based layout synthesis for analog ICs,” *IEEE J. Solid-State Circuits*, vol. 28, no. 3, pp. 261–268, Mar. 1993.

[31] Xu Jingnan, J. Vital, and N. Horta, "A SKILLTM - based Library for Retargetable Embedded Analog Cores," *Proceedings on Design, Automation and Test in Europe*, pp. 768–769, Mar. 2001.

[32] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C. Shi, “IPRAIL—Intellectual property reuse-based analog IC layout automation,” *Integration, VLSI J.*, vol. 36, no. 4, pp. 237–262, Nov. 2003.

[33] L. Zhang, U. Kleine, and Y. Jiang, “An Automated Design Tool for Analog Layouts,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 8, pp. 881–894, Aug. 2006.

[34] Y. Yilmaz, and G. Dundar, “Analog Layout Generator for CMOS Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 32–45, Jan. 2009.

[35] L. Zhang, and Z. Liu, “A Performance-Constrained Template-Based Layout Retargeting Algorithm for Analog Integrated Circuits,” in *Proc. 47th ACM/IEEE Design Automation Conference,* pp. 293 – 298, Jan. 2010.

[36] J. Rijmenants, J. Litsios, T. Schwarz, and M. Degrauwe, “Ilac: An automated layout tool for analog cmos circuits,” *IEEE Journal of Solid-State Circuits SC*, vol. 24, no. 2, pp. 417–425, April1989.

[37] Cohn, John M., Garrod, David J., Rutenbar, Rob A., and Carley, L. Richard, "KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing," *IEEE Journal of Solid-State Circuits*, vol 26, no.3, pp. 330 - 342, Mar. 1991.

[38] K. Lampaert, G. Gielen, and W. Sansen, “A performance-driven placement tool for analog integrated circuits,” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 7, pp. 773–780, July 1995.

[39] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, “Automation of IC layout with analog constraints”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 923–942, Aug. 1996.

[40] P. Khademsameni, and M. Syrzycki, "A tool for automated analog CMOS layout module generation and placement," in *IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 416 - 421, May 2002.

[41] R. Castro-Lopez, O. Guerra, E. Roca, and F. Fernandez, “An integrated layout-synthesis approach for analog ICs,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 7, pp. 1179–1189, Jul. 2008.

[42] P. Vancorenland, G. V. der Plas, M. Steyaert, G. Gielen, and W. Sansen, “A layout-aware synthesis methodology for RF circuits,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 358 – 362, Nov. 2001.

[43] M. Ranjan, W. Verhaegen, A. Agarwal, H. Sampath, R. Vemuri, and G. Gielen, “Fast, layout inclusive analog circuit synthesis using pre-compiled parasitic-aware symbolic performance models,” in *Design Automation Conference and Test in Europe Conference (DATE)*, vol. 1, pp. 604 – 609, Feb. 2004.

[44] A. Pradhan, and R. Vemuri, “Efficient synthesis of a uniformly spread layout aware Pareto surface for analog circuits,” in *22nd International Conference on VLSI Desing*, pp. 131–136, Jan. 2009.

[45] H. Habal and H. Graeb, “Constraint-Based Layout-Driven Sizing of Analog Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1089-1102, 2011.

[46] “Ciranova,” http://www.ciranova.com/.

[47] “Tanner EDA” http://www.tannereda.com/.

[48] F. Maloberti, *Analog Design for CMOS VLSI Systems*, Kluwer Academic Publishers, 2001.

[49] M. Barros, J. Guilherme, and N. Horta, “GA-SVM feasibility model and optimization kernel applied to analog IC design automation,” in *ACM Great Lakes symposium on VLSI*, pp 469-472, Mar 2007.

[50] M. Barros, J. Guilherme, and N. Horta, *Analog circuits and systems optimization based on evolutionary computation techniques*, Studies in computational intelligence, vol. 294. Springer, 2010.

[51] M. Barros, J. Guilherme, and N. Horta, “Analog circuits optimization based on evolutionary computation techniques,” *Integration, VLSI J.*, vol. 43, no. 1, pp. 136-155, Jan. 2009.

[52] N. Lourenço and N. Horta, “GENOM-POF: Multi-Objective Evolutionary Synthesis of Analog ICs with Corners Validation,” *Genetic and Evolutionary Computation Conference (GECCO)* 2012, USA., in press.

[53] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, Apr. 2002.

[54] “XML,” http://www.xml.com/.

[55] R. Jacob Baker, *CMOS Circuit Design, Layout and Simulation,* IEEE Press, New York, 2005.

[56] F. Balasa, S. C. Maruvada, and K. Krishnamoorthy, "Using Red-Black Interval Trees in Device-Level Analog Placement with Symmetry Constraints," *Proceedings of the Asian and South Pacific – Design Automation Conference*, pp. 777 - 782, Jan. 2003.

[57] A. Hastings, *The Art of Analog Layout*, Prentice Hall, 2nd edition, 2005.

[58] R. L. Graham, “An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set,” *Information Processing Letters*, vol. 1, pp. 132-133, Jan. 1972.

[59] B. Nowacki, N. Paulino, and J. Goes, “A 1.2 V 300 μW second-order switched-capacitor ΔΣ modulator using ultra incomplete settling with 73 dB SNDR and 300 kHz BW in 130 nm CMOS,” in *Proceedings of the ESSCIRC*, pp.271-274, Out. 2011.

[60] F. Medeiro, F. V. Fernandez, R. Dominguez-Castro, and A. Rodriguez-Vazquez, “A Statistical optimization-based approach for automated sizing of analog cells,” in ACM/IEEE International Conference on Computer-Aided Design, pp. 594-597, Nov. 1994.

Appendix A Overview of Electronic Design Automation Tools

Table A-1 – Overview of placement tools.

Placement Tool	Year	Specifications				Observations
		Description	Techniques	Level / FR[1]	Constraints	
Multiple Symmetry Groups [26]	2007	Exploration of symmetric-feasible SPs, the presence of a symmetry group is directly taken into account. Keeps the symmetry groups tightly on their axes, producing always optimal solutions in terms of area.	Optimization -based with Simulated Annealing	Device-level, Sequence Pair	Symmetry	(+) Complexity of $O(G.n.log\ n)$ per code evaluation, G number of symmetry groups.
Linear Programming [27]	2007	Linear programming used to solve linear expressions derived from a constraint graph; graph codes a placement that satisfies symmetry constraints and topology constraints imposed by a SP.	Device-level, Linear Programming		Symmetry	(+) The number of linear expressions is decreased by substituting the expressions for dependent variables.
Module Clustering [28]	2008	Deals with different constraints simultaneously and hierarchically. Devices intended to satisfy a constraint are formed as a cluster; clusters may contain not only devices in the same level, but also other clusters.	Optimization -based with Simulated Annealing	Hierarchical HB*-tree (Symmetry Islands)	Matching Symmetry Proximity	First work handling floorplanning with the clustering constraint using HB*-trees.
Plantage [14]	2008	Bottom-up approach; placements of the basic modules are generated by enumeration and then combined (stepping up the hierarchy tree) using enhanced shape functions. Suboptimal modules are removed.	Deterministic	Hierarchical B*-tree	Matching Symmetry Proximity	(+) The result is a POF of placements with different aspect ratios, instead of a single layout; avoids randomness; (-) High computational times.
Thermal-Driven Placement [29]	2009	Establishes a thermal profile for better thermal matching of the devices of a circuit; reduce thermally-induced mismatches. Uses look-up tables to store the thermal profile of each device, easy thermal profile computation.	Optimization -based with Simulated Annealing	Hierarchical HB*-tree (Symmetry Islands)	Symmetry Matching (common-centroid)	One of the few works studying the impact of power devices on thermally-sensitive devices

1 – Chip Floorplan Representation.

Table A-2 – Overview of layout generation tools, part I.

Layout Tool	Year	Specifications				
		Description	Tech.	M[1]	P[2]	Input/Output Data
ILAC [36]	1989	Macro-cell Place and Route; placement and routing algorithms inspired by those used for digital design. Not limited to circuits in the input library.	CMOS	✓	✓	In: Netlist, user-specified constraints on cell height; specs. Out: CIF file.
KOAN/ANAG II [37]	1991	Macro-cell Place and Route; uses a pre-defined small module generators data-base. The fusion of two classical tools, placer KOAN and router ANAGRAM.	CMOS BiCMOS	✓	✓	In: Spice netlist with annotation to control place/route; Out: Magic file.
ALSYN [30]	1993	Procedural modules controlled though a user-defined database of rules.	CMOS	✓		In: Circuit netlist; rule sets.
LAYLA [38]	1995	Similar tools. Macro-cell Place and Route. Constraint-driven layout, the degradation of the performance due to due to interconnect parasitic and device mismatches is weighed, combines this with geometrical optimization.	CMOS	✓	✓	In: Circuit netlist; list of performance specifications.
Malavasi [39]	1996			✓	✓	
Jingnan [31]	2001	Automatic generation and reusability of physical layouts; high-functionality pCells	Independ	Procedural layout generation.		
ALDAC [40]	2002	Generate full-stacked layout modules and performs module placement and local routing. Stacks can be performed either fully-automatically or user controlled.	CMOS	✓		In: Design Rules; ALDAC specific netlist. Out: CIF file.
IPRAIL [32]	2003	Automatically creates a template from an existing expertise-embedded layout, and then imposes new device sizes and technology design rules on template.	Independ	✓		In: CIF file; original and target technology rules. Out: CIF file.
LAYGEN [13]	2006	Includes expert knowledge as placement and routing constraints. Designer provides a high level template description and layout is automatically generated.	Independ	✓		In: Selected template; Technology Design Kit.
ALADIN [33]	2006	Designers can develop and maintain technology- and application-independent module generator for relatively complex sub circuits.	Independ	✓	✓	In: Cells and Netlist; Interative association between them.
ALG [34]	2009	User can interact with the tool in each automation step to enhance/polish the layout in order to meet performance specifications. Performance-aware operation provided by a layout adviser tool, YASA.	CMOS	✓	✓	In: Specifications; designer's interaction at different levels.
Zhang [35]	2010	Parasitic-aware retargeting; performance sensitivities with respect to parasitics are first determined; automation in a single process without users intervention.	CMOS	✓	✓	In: Existing layout; original and target technology design rules.

1 – Matching and symmetry constraints; 2 – Proximity constraint.

Table A-3 – Overview of layout generation tools, part II.

Specifications					
Layout Tool	**Placement**		**Router**	**Development Environment**	**Observations**
	Optimization	**Floorplan**			
ILAC [36]	S. Annealing	Slicing Tree	Best-first maze search.	Pascal	(-) Slicing representation.
KOAN II [37]	S. Annealing	Absolute	Re-routing, over-the-device wiring, crosstalk avoidance.	C code	(-) High dimensionality of the solution space.
ALSYN [30]	Deterministic	Slicing Tree	Maze router with crosstalk avoidance.	C code	(+) Combines the concept of easy-to-write rules with fast procedural placement.
LAYLA [38]	Simulated Annealing	Absolute	Take into account variable wire widths.	C++ in UNIX	(+) Optimize solution quantifying the performance degradation due to impact of parasitic; more optimum solutions found.
Malavasi [39]			Maze router.	OCTTOOLS	
Jingnan [31]	Procedural layout generation.			Cadence SKILL	Parameterized cells organized hierarchically
ALDAC [40]	S. Annealing	Absolute	Local routing with two metal layers.	C++ on PC	(+) Post-layout simulation of multiple layouts
IPRAIL [32]	Linear programming and graph-based methods.			-	(-) Undervalues performance.
LAYGEN [13]	S. Annealing	B*-tree	Adapts the template routing to the created placement;	Java	Speeds up retargeting or technology migration
ALADIN [33]	Two-stage technique: 1) Genetic approach with simulated annealing and half-perimeter routing. 2) Very fast reannealing placement algorithm and global routing.			C++, Cadence, SKILL, Tcl/Tk	(-) Can only handle small or medium size circuits.
ALG [34]	Different from custom to automated mode, combined with global and local routing steps.			Java	User may choose the level of automation.
Zhang [35]	Mixed-integer nonlinear programming and graph-based methods.			C/C++	(+) Retargeting with less area and CPU time.

Table A-4 – Overview of layout-aware sizing tools.

Layout-Aware Sizing Tool	Year	Specifications			Observations
		Description	Estimation method for Parasitics	LG[1]	
Layout-aware for RF [42]	2001	The sizing engine uses basic performance models and evolutionary algorithms.	Dimensions obtained from the generated layout are used to calculate analytical models.	Yes	(-) Performance estimated and the information of parasitics is very limited.
Symbolic Models [43]	2004	Efficient parameterized layout generation; symbolic performance models are used to predict the circuit performance.	Device parasitics extracted from the templates are incorporated into symbolic equation performance models.	Yes	(-) Limited to small-signal performances; geometric constraints are not considered.
Integrated Layout-Synthesis Approach [41]	2008	Parasitic-aware sizing and geometrically constrained sizing. Uses a combination of simulation based optimization, procedural layout generation and exhaustive geometric evaluation algorithms.	Calculation of the MOS transistor diffusion, areas and perimeters by analytic equations.	No	(-) Storage requirements for the lookup table may be exceedingly large
			Geometric methods for transistors, 3-D Analytical and Geometric methods for interconnects and other devices.	Yes	(+) Extraction very accurate; performance evaluator is HSPICE. (-) Longer simulation times.
Layout-Aware Pareto Surface [44]	2009	Performance is predicted using circuit matrix models formulated with spice simulations data. Pareto tradeoff between performances is explored using a multi objective simulated annealing algorithm.	Sample layouts are obtained by procedural layout generation. Device parasitics are estimated using polynomial models with the known bias, diffusion area and perimeter for devices.	No	(+) Fast; result is a pareto optimal surface inclusive of layout effects; (-) While device parasitics are approximated; geometric constraints and matching are barely considered.
Constraint-Based Layout-Driven Sizing [45]	2011	Uses algorithm Plantage [14] (Section 2.1.3) to generate placement solutions. A deterministic algorithm is used for circuit sizing; DC electrical constraints are employed to ensure correct CMOS operating region and device matching.	Parasitic coupling capacitance is extracted directly by an integral equation field solver without any modeling or approximation. The effect of routing congestion is considered.	Yes	(+) Numerical (SPICE) simulation for performance evaluation. (-) Slow, complexity of the sizing method increases with the number of devices, routing constraints and design parameters

1 – Layout Generation

Appendix B GDSII File Format

GDSII Stream format is the standard file format for transferring/archiving 2D graphical design data. It contains a hierarchy of structures, and each structure contains different elements (boundary/polygon, path/polyline, structure references, array of structures, text, node or box). The elements are placed on layers. GDSII is a binary format that is platform independent, because it uses internally defined formats for its data types. While reading GDSII files, the GDSII internal data types (like float, integers, etc.) need to be converted to the platform data types that are used. The format is a sequential list of records; each record contains a header to tell what information is in the record. The order in which the records are sorted is relevant, because of the strict organization it is relatively easy to parse. The maximum number of vertices is officially only 200 x,y pairs, but many packages can read up to the absolute maximum of 64k/2=32k. Because 32 Kbytes is the maximum record length that can be specified (two bytes).

B.1 Record Structure

The Stream format output file is composed of variable length records. Record length is measured in bytes. The minimum record length is four bytes. Within the record, two bytes (16 bits) is a word. The 16 bits in a word are numbered 0 to 15, left to right.

The first four bytes of a record compose the record-header. The first two bytes of the record-header contain a count (in eight-bit bytes) of the total record length, so the maximum length is 65536 (64k). The next record starts immediately after the last byte of the previous record. The third byte of the header is the record type. The fourth byte of the header identifies the type of data contained within the record. The fifth until count bytes of a record contain the data. The fourth byte in the record header contains the data type for the rest of the record. The record length is used to find the number of items of the specified data type. The used records are listed in Table B-1.

Table B-1 – GDS records header.

File Header Records	Bytes 3 and 4	Parameter Type
HEADER	0x0002	2-byte integer
BGNLIB	0x0102	12 2-byte integers
LIBNAME	0x0206	ASCII string
REFLIBS	0x1F06	2 45-character ASCII strings
FONTS	0x2006	4 44-character ASCII strings
ATTRTABLE	2306	44-character ASCII string
GENERATIONS	2202	2-byte integer
FORMAT	3602	2-byte integer
MASK	3706	ASCII string
ENDMASKS	3800	No data
UNITS	0305	2 8-byte floats

File Tail Records	Bytes 3 and 4	Parameter Type
ENDLIB	0400	No data
Structure Header Records	**Bytes 3 and 4**	**Parameter Type**
BGNSTR	0502	12 2-byte integers
STRNAME	0606	Up to 32-characters ASCII string
Structure Tail Records	**Bytes 3 and 4**	**Parameter Type**
ENDSTR	0700	No data
Element Header Records	**Bytes 3 and 4**	**Parameter Type**
BOUNDARY	0800	No data
PATH	0900	No data
SREF	0A00	No data
AREF	0B00	No data
TEXT	0C00	No data
NODE	1500	No data
BOX	2D00	No data
Element Contents Records	**Bytes 3 and 4**	**Parameter Type**
ELFLAGS	2601	2-byte integer
PLEX	2F03	4-byte integer
LAYER	0D02	2-byte integers
DATATYPE	0E02	2-byte integer
XY	1003	Up to 200 4-byte integer pairs
PATHTYPE	2102	2-byte integer
WIDTH	0F03	4-byte integer
SNAME	1206	Up to 32-character ASCII string
STRANS	1A01	2-byte integer
MAG	1B05	8-byte float
ANGLE	1C05	8-byte float
COLROW	1302	2 2-byte integers
TEXTTYPE	1602	2-byte integer
PRESENTATION	1701	2-byte integer
ASCII STRING	1906	Up to 512-character string
NODETYPE	2A02	2-byte integer
BOXTYPE	2E02	2-byte integer

B.2 Data Types

In the GDSII Stream Format Manual v6.0 there are seven data-types listed, the first is "No data present". The code is 0x00. This means that the entire record is 4 bytes long. An example of an element with no data would be ENDLIB, which marks the end of a library.

The second is called a "Bit array". The code is 0x01. It is simply two bytes. The meaning of each bit depends on the record type that the bit array is found in.

The third data type is a "Two-Byte Signed Integer". The code is 0x02. It is an integer between -32768 and 32767. It is stored in twos complement format, with the most significant byte first.

The fourth data type is a "Four-Byte Signed Integer". The code is 0x03. Same basic thing as a two-byte integer, but with four bytes.

The fifth data type is the "Four-Byte Real". The code is 0x04. This one seems to have never been used, so the eight-byte real will be described in a bit more detail. However, the first bit is the sign (1 = negative), the next 7 bits are the exponent. You have to subtract 64 from this number to get the real value. The next three bytes are the mantissa, divide by 2^24 to get the denominator.

$$value = \frac{mantissa}{2^{24}} . 16^{exponent-64} \quad \text{(B-1)}$$

In the above, we use the actual values of the fields in the stream file for the mantissa and exponent.

The sixth data type is the "Eight Byte Real". The code is 0x05. This one gets a little more use. The first (most significant) bit of the first byte is the sign, one means negative, 0 means positive. The 7 least significant bits of the first byte are the exponent in "excess 64" notation. The remaining 7 bytes are the mantissa, with a binary point to the left of the most significant figure. The formula below uses the unsigned integer value of these 7 bytes as the numerator of a fraction.

$$value = \frac{mantissa}{2^{56}} . 16^{exponent-64} \quad \text{(B-2)}$$

The seventh and final data type is the "ASCII String". The code is 0x06. The length of this string is always equal to the length of the record minus the four bytes used for the record header. If this number is not even, a NULL character (0x00) is added to the end. This is another artifact of the 16-bit words that the stream file format assumes.

B.3 Library Head and Tail

A GDS II file header always begins with a HEADER record the parameter of which contains the GDSII version number used to write the file. For example, the bytes [0, 6, 0, 2, 0, 1] at the start of the file constitute the header record for a version-1 file. Following the HEADER, comes a BGNLIB record that contains the date of the last modification and the date of the last access to the file. Dates take six 2-byte integers to store the year, month, day, hour, minute, and second. The third record of a file is the LIBNAME, which identifies the name of this library file. For example, the bytes 0, 8, 2, 6, "C", "H", "I", "P" define a library named "CHIP." Following the LIBNAME record there may be any of the optional header records: REFLIBS to name up to two reference libraries, FONTS to name up to four character fonts, TTRTABLE to name an attribute file, GENERATIONS to indicate the number of old file copies to keep, and FORMAT to indicate the nature of this file. The strings in the REFLIBS, FONTS, and ATTRTABLE records must be the specified length, padded with zero bytes. The parameter to FORMAT has the value 0 for an archived file and the value 1 for a filtered file. Filtered files contain only a subset of the mask layers and that subset is described with one or more MASK records followed by an ENDMASK record. The string parameter in a MASK record names layers and sequences of layers; for example, "1 3 5-7." The final record of a file header is the UNITS record and it is not optional. The parameters to this record contain the number of user units per database unit (typically less than 1 to allow granularity of user specification) and the number of meters per database unit (typically much less than 1 for IC specifications). After the last structure has been defined, the file

terminates with a simple ENDLIB record. Note that there is no provision for the specification of a root structure to define a circuit; this must be tracked by the designer.

B.4 Structure Head and Tail

Each structure has two header records and one tail record that contains an arbitrary list of elements. The first structure header is the BGNSTR record, which contains the creation date and the last modification date. Following that is the STRNAME record, which names the structure using any alphabetic or numeric characters, the dollar sign, or the underscore. The structure is then open and any of the seven elements can be listed. The last record of a structure is the ENDSTR. Following it must be another BGNSTR or the end of the library, ENDLIB.

B.5 Elements

There are seven kinds of elements: boundary, defines a filled polygon; path, defines a wire; structure reference, invokes a subcell; array reference, invokes an array of subcells; text is for documentation; node, defines an electrical path; and box places rectangular geometry.

B.5.1 Boundary

The boundary element defines a filled polygon. It begins with a BOUNDARY record, has an optional ELFLAGS and PLEX record, and then has required LAYER, DATATYPE, and XY records.

The ELFLAGS record, which appears optionally in every element, has two flags in its parameter to indicate template data (if bit 16 is set) or external data (if bit 15 is set). This record should be ignored on input and excluded from output. Note that the GDS II integer has bit 1 in the leftmost or most significant position so these two flags are in the least significant bits.

The PLEX record is also optional to every element and defines element structuring by aggregating those that have common plex numbers. Although a 4-byte integer is available for plex numbering, the high byte (first byte) is a flag that indicates the head of the plex if its least significant bit (bit 8) is set.

The LAYER record is required to define which layer (numbered 0 to 63) is to be used for this boundary. The meaning of these layers is not defined rigorously and must be determined for each design environment and library.

The DATATYPE record contains unimportant information and usually its argument is zero, nevertheless CADENCE uses the DATATYPE to identify the layer purpose.

The XY record contains anywhere from four to 200 coordinate pairs that define the outline of the polygon. The number of points in this record is defined by the record length. Note that boundaries must be closed explicitly, so the first and last coordinate values must be the same.

B.5.2 Path

A path is an open figure with a nonzero width that is typically used to place wires. This element is initiated with a PATH record followed by the optional ELFLAGS and PLEX records. The LAYER record must follow to identify the desired path material. In addition, a DATATYPE record must appear and an XY record to define the coordinates of the path. From two to 200 points may be given in a path.

Prior to the XY record of a path specification there may be two optional records called PATHTYPE and WIDTH. The PATHTYPE record describes the nature of the path segment ends, according to its parameter value. If the value is 0, the segments will have square ends that terminate at the path vertices. The value 1 indicates rounded ends and the value 2 indicates square ends that overlap their vertices by one-half of their width. The width of the path is defined by the optional WIDTH record. If the width value is negative, then it will be independent of any structure scaling (from MAG records, see next section).

B.5.3 Structure Reference

Hierarchy is achieved by allowing structure references (instances) to appear in other structures. The SREF record indicates a structure reference and is followed by the optional ELFLAGS and PLEX records. The SNAME record then names the desired structure and an XY record contains a single coordinate to place this instance. It is legal to refer to structures that have not yet been defined with STRNAME.

Prior to the XY record, there may be optional transformation records. The STRANS record must appear first if structure transformations are desired. Its parameter has bit flags that indicate mirroring in x before rotation (if bit 1 is set), the use of absolute magnification (if bit 14 is set), and the use of absolute rotation (if bit 15 is set). The magnification and rotation amounts may then be specified in the optional MAG and ANGLE records. The rotation angle is in counterclockwise degrees.

B.5.4 Array of Structures

For convenience, an array of structure instances can be specified with the AREF record. Following the optional ELFLAGS and PLEX records comes the SNAME to identify the structure being arrayed. Next, the optional transformation records STRANS, MAG, and ANGLE give the orientation of the instances. A COLROW record must follow to specify the number of columns and the number of rows in the array. The final record is an XY with three points: the coordinate of the corner instance, the coordinate of the last instance in the columnar direction, and the coordinate of the last instance in the row direction. From this information, the amount of instance overlap or separation can be determined. Note that flipping arrays (in which alternating rows or columns are mirrored to abut along the same side) can be implemented with multiple arrays that are interlaced and spaced apart to describe alternating rows or columns.

B.5.5 Text

Messages can be included in a circuit with the TEXT record. The optional ELFLAGS and PLEX follow with the mandatory LAYER record after that. A TEXTTYPE record must then appear. An optional PRESENTATION record specifies the font in bits 11 and 12, the vertical presentation in bits 13 and 14 (0 for top, 1 for middle, 2 for bottom), and the horizontal presentation in bits 15 and 16 (0 for left, 1 for center, 2 for right). Optional PATHTYPE, WIDTH, STRANS, MAG, and ANGLE records may appear to affect the text. The last two records are required: an XY with a single coordinate to locate the text and a STRING record to specify the actual text.

B.5.6 Node

Electrical nets may be specified with the NODE record. The optional ELFLAGS and PLEX records follow and the required LAYER record is next. A NODETYPE record must appear, followed by an XY record with one to 50 points that identify coordinates on the electrical net. The information in this element is not graphical and does not affect the manufactured circuit.

B.5.7 Box

The last element of a GDS II file is the box. Following the BOX record are the optional ELFLAGS and PLEX records, a mandatory LAYER record, a BOXTYPE record, and an XY record. The XY must contain five points that describe a closed, four-sided box. Unlike the boundary, this is not a filled figure. Therefore, it cannot be used for IC geometry.

Appendix C **UMC 130 nm CMOS Technology Design Kit**

In order to support the methodology proposed, the developed framework must provide support for multiple technologies and should be easy to extend to new technologies. Most commercial tools use a layer map file that associate the layer number and data type number present in the binary file with the layer name and purpose. Figure C-1 shows the layer map file for the UMC 130 nm technology design kit used by LAYGEN II.

```
#########################################################
#            DF Layer             |    GDSII  Layer
# Name                  Purpose    |   Number  DataType
#--------------------------------------------------------
NTUB                    drawing          3      0
DIFF                    drawing          1      0
POLY1                   drawing         41      0
NPLUS                   drawing         12      0
PPLUS                   drawing         11      0
POLY2                   drawing         41      1
CONT                    drawing         39      0
MET1                    drawing         46      0
VIA1                    drawing         47      0
MET2                    drawing         48      0
VIA2                    drawing         49      0
MET3                    drawing         50      0
VIA3                    drawing         51      0
MET4                    drawing         52      0
VIA4                    drawing         53      0
MET5                    drawing         54      0
VIA5                    drawing         55      0
MET6                    drawing         56      0
VIA6                    drawing         57      0
MET7                    drawing         58      0
VIA7                    drawing         59      0
MET8                    drawing         60      0
PIN                     poly1          100     0
PIN                     metal1         101     0
PIN                     metal2         102     0
PIN                     metal3         103     0
PIN                     metal4         104     0
PIN                     metal5         105     0
PIN                     metal6         106     0
PIN                     metal7         107     0
PIN                     metal8         108     0
ERROR                   laygen         200     1
MARKER1                 laygen         200     2
MARKER2                 laygen         200     3
DRCERROR                laygen         200     4
#--------------------------------------------------------
```

Figure C-1 – UMC 130 nm layer map.

The layer structure specifying the conductors and vias organization is described using an array where the lowest index indicates the deepest layer. For the current technology the layer structure is described in Figure C-2.

```
/*Metals by order*/
conductors_[0] = P1;
conductors_[1] = M1;
conductors_[2] = M2;
conductors_[3] = M3;
conductors_[4] = M4;
conductors_[5] = M5;
conductors_[6] = M6;
conductors_[7] = M7;
conductors_[8] = M8;

/*Vias by order*/
vias_[0] = CT;
vias_[1] = V1;
vias_[2] = V2;
vias_[3] = V3;
vias_[4] = V4;
vias_[5] = V5;
vias_[6] = V6;
vias_[7] = V7;
```

Figure C-2 – UMC 130 nm layer structure.

To define the layout Look & Feel in the GUI, a specific file maps the layers to a color, design pattern, and relative depth. The technology display settings are presented in Figure C-3.

#layer-num	type-num	Laygen-color	Laygen-pattern	layer-order
1	0	16711680	FillPattern	40
3	0	0	none	10
11	0	14053594	none	30
12	0	16749824	none	20
41	0	255	FillPattern	50
41	1	65535	FillPattern	60
39	0	65280	FillPattern	70
46	0	4772300	none	80
47	0	16776960	CrossPattern	90
48	0	16776960	RightLinePattern	100
49	0	12500670	CrossPattern	110
50	0	2984535	RightLinePattern	120
51	0	13882323	CrossPattern	130
52	0	9445616	RightLinePattern	140
53	0	0	CrossPattern	150
54	0	0	RightLinePattern	160
55	0	0	CrossPattern	170
56	0	0	RightLinePattern	180
57	0	0	CrossPattern	190
58	0	0	RightLinePattern	200
59	0	0	CrossPattern	210
60	0	0	RightLinePattern	220
100	0	255	RightLinePattern	230
101	0	13422920	RightLinePattern	240
102	0	16776960	RightLinePattern	250
103	0	2984535	RightLinePattern	260
104	0	9445616	RightLinePattern	270
105	0	0	RightLinePattern	280
106	0	0	RightLinePattern	290
107	0	0	RightLinePattern	300
108	0	0	RightLinePattern	310
200	1	454545	CrossPattern	320
200	2	16776960	CrossPattern	330
200	3	13422920	CrossPattern	340
200	4	-454545	CrossPattern	350

Figure C-3 – UMC 130 nm display settings.

The design rules associate the values for the supported rules with layer or layers. The technology design rule map can be found in Figure C-4.

```
// all values in nano  manufacturing_grid_ = 5;
/*-------------------------------------------
 *  AA rules LAYGEN II
 *-------------------------------------------*/
//Minimum Diffusion width NMOS and PMOS
width_.put(AA, Δ /manufacturing_grid_);
//Minimum Diffusion to Diffusion spacing
spacing_.put(AA_AA, Δ /manufacturing_grid_);
//Minimum N Well enclosure of P+ Diffusion
enclosure_.put(NW_PD, Δ /manufacturing_grid_);
//Minimum N Well enclosure of N+ Diffusion
enclosure_.put(NW_ND, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  NW rules LAYGEN II
 *-------------------------------------------*/
width_.put(NW, Δ /manufacturing_grid_);
//Minimum N Well to N Well notch 0
spacing_.put(NW_NW, Δ /manufacturing_grid_);
//Minimum N Well to N Well spacing or notch
spacing_.put(NW_NW2, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  NP rules LAYGEN II
 *-------------------------------------------*/
width_.put(NP, Δ /manufacturing_grid_);
//Minimum N+ implant enclosure of N+ Diffusion
enclosure_.put(NP_ND, Δ /manufacturing_grid_);
//Minimum N+ implant overhang of N+ Diffusion
extension_.put(NP_ND, Δ /manufacturing_grid_);
//Minimum N+ implant enclosure of N+ Contact
enclosure_.put(NP_CT, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  PP rules LAYGEN II
 *-------------------------------------------*/
width_.put(PP, Δ /manufacturing_grid_);
//Minimum P+ implant to P+ implant spacing
spacing_.put(PP_PP, Δ /manufacturing_grid_);
//Minimum P+ implant overhang of P+ Diffusion
extension_.put(PP_PD, Δ /manufacturing_grid_);
//Minimum P+ implant enclosure of P+ Diffusion
enclosure_.put(PP_PD, Δ /manufacturing_grid_);
//Minimum P+ implant enclosure of P+ Contact
enclosure_.put(PP_CT, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  CT rules LAYGEN II
 *-------------------------------------------*/
width_.put(CT, Δ /manufacturing_grid_);
//Minimum contact to contact spacing
spacing_.put(CT_CT, Δ /manufacturing_grid_);
//Minimum DIFFUSION Contact to Poly1 spacing
spacing_.put(P1_CT, Δ /manufacturing_grid_);
//Minimum P+ DIFFUSION enclosure of P+ Diff Ct
enclosure_.put(PD_CT, Δ /manufacturing_grid_);
//Minimum N+ DIFFUSION enclosure of N+ Diff Ct
enclosure_.put(ND_CT, Δ /manufacturing_grid_);
//M1 line end enclosure of Contact
enclosure_.put(M1_CT, Δ /manufacturing_grid_);
//Minimum POLY1 enclosure of Contact
enclosure_.put(P1_CT, Δ /manufacturing_grid_);
//Minimum CPOLY enclosure of POLY2CON
enclosure_.put(CPOLY_CT, Δ /manufacturing_grid_);
//Minimum POLY1CON to CPOLY spacing
spacing_.put(CT_CPOLY, Δ /manufacturing_grid_);
/*-------------------------------------------
 * P1 rules
 *-------------------------------------------*/
//Minimum POLY1 with for NMOS/PMOS (1.2V device)
width_.put(P1, Δ /manufacturing_grid_);
//Minimum POLY1 to POLY1 spacing
spacing_.put(P1_P1, Δ /manufacturing_grid_);
//Minimum POLY1 overhand of DIFFUSION
extension_.put(P1_AA, Δ /manufacturing_grid_);
```

```
/*-------------------------------------------
 *  M1 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M1, Δ /manufacturing_grid_);
spacing_.put(M1_M1, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V1 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V1, Δ /manufacturing_grid_);
spacing_.put(V1_V1, Δ /manufacturing_grid_);
enclosure_.put(M1_V1, Δ /manufacturing_grid_);
enclosure_.put(M2_V1, Δ);
/*-------------------------------------------
 *  M2 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M2, Δ /manufacturing_grid_);
spacing_.put(M2_M2, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V2 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V2, Δ /manufacturing_grid_);
spacing_.put(V2_V2, Δ /manufacturing_grid_);
enclosure_.put(M2_V2, Δ);
enclosure_.put(M3_V2, Δ);
/*-------------------------------------------
 *  M3 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M3, Δ /manufacturing_grid_);
spacing_.put(M3_M3, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V3 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V3, Δ /manufacturing_grid_);
spacing_.put(V3_V3, Δ /manufacturing_grid_);
enclosure_.put(M3_V3, Δ);
enclosure_.put(M4_V3, Δ);
/*-------------------------------------------
 *  M4 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M4, Δ /manufacturing_grid_);
spacing_.put(M4_M4, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V4 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V4, Δ /manufacturing_grid_);
spacing_.put(V4_V4, Δ /manufacturing_grid_);
enclosure_.put(M5_V4, Δ);
enclosure_.put(M4_V4, Δ);
/*-------------------------------------------
 *  M5 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M5, Δ /manufacturing_grid_);
spacing_.put(M5_M5, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V5 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V5, Δ /manufacturing_grid_);
spacing_.put(V5_V5, Δ /manufacturing_grid_);
enclosure_.put(M6_V5, Δ);
enclosure_.put(M5_V5, Δ);
/*-------------------------------------------
 *  M6 rules LAYGEN II
 *-------------------------------------------*/
width_.put(M6, Δ /manufacturing_grid_);
spacing_.put(M6_M6, Δ /manufacturing_grid_);
/*-------------------------------------------
 *  V6 rules LAYGEN II
 *-------------------------------------------*/
width_.put(V6, Δ /manufacturing_grid_);
spacing_.put(V6_V6, Δ /manufacturing_grid_);
enclosure_.put(M7_V6, Δ);
enclosure_.put(M6_V6, Δ);
```

Δ – Values omitted due to restricted rights .

Figure C-4 – UMC 130 nm design rules.

In order to complete the design kit, parametric module generators for basic devices must be provided. The module generator encompasses two types of functions, devices generators and composed shapes that depend of the technology design rules. Devices generators include parametric cells, e.g., transistors and capacitors. Composed geometric generators include, e.g., multilayer contacts and guard rings. Figure C-5 shows the available parametric module generators for the current technology design kit. More devices and composed forms will be introduced as the tool matures, but to demonstrate the feasibility of the techniques presented in this work, this subset will suffice.

```
//Component Generators
NMOS(int width, int length, int m, int angle, String bulk, Boolean sideways);

PMOS(int width, int length, int m, int angle, String bulk_, Boolean sideways);

NMOS_merged(int width, int length, int m_, int angle, String bulk, Boolean sideways, Boolean
            common_source, Boolean gates_connect, String name_P1, String name_P2);

PMOS_merged(int width, int length, int m_, int angle, String bulk_, Boolean sideways, Boolean
            common_source, Boolean gates_connect, String name_P1, String name_P2);

Capacitor(int width, int length, int n, int m, int angle);

//Composed Geometric Generators
Contact(double center_x, double center_y, Layer.ID top, Layer.ID bottom, double w_width);

MultipleContact(double center_x, double center_y, Layer.ID top, Layer.ID bottom, int orientation);

Guard_Ring(Layout l, LinkedList<Point> bounds, Boolean nwell);

Power_Strip(Layout layout, String pin_name, int min_width, PowerNetMode mode,
            List<IShape> ignore_shapes, HashMap<String, Point> bounds);
```

Figure C-5 – UMC 130 nm module generator.

Appendix D AMS 350 nm CMOS Technology Design Kit

To test the ability to retarget for a different technology, the AMS 350 nm technology design kit is used. Figure D-1 shows the layer map file for the AMS 350 nm technology design kit used by LAYGEN II.

```
#########################################################
#            DF Layer              |    GDSII  Layer
# Name                    Purpose   |   Number  DataType
#-------------------------------------------------------
NTUB                      drawing          5      0
FIMP                      drawing          8      0
DIFF                      drawing         10      0
POLY1                     drawing         20      0
NLDD                      drawing         21      0
PLDD                      drawing         22      0
NPLUS                     drawing         23      0
PPLUS                     drawing         24      0
POLY2                     drawing         30      0
CONT                      drawing         34      0
MET1                      drawing         35      0
VIA1                      drawing         36      0
MET2                      drawing         37      0
VIA2                      drawing         38      0
MET3                      drawing         39      0
VIA3                      drawing         41      0
MET4                      drawing         42      0
text                      drawing         61      0
PIN                       poly1           61      1
PIN                       metal1          61      2
PIN                       metal2          61      3
PIN                       metal3          61      4
PIN                       metal4          61      5
ERROR                     laygen          200     1
MARKER1                   laygen          200     2
MARKER2                   laygen          200     3
DRCERROR                  laygen          200     4
#-------------------------------------------------------
```

Figure D-1 – AMS 350 nm layer map.

For the current technology the layer structure is described in Figure D-2.

```
/*Metals by order*/
conductors_[0] = P1;
conductors_[1] = M1;
conductors_[2] = M2;
conductors_[3] = M3;
conductors_[4] = M4;

/*Vias by order*/
vias_[0] = CT;
vias_[1] = V1;
vias_[2] = V2;
vias_[3] = V3;
```

Figure D-2 – AMS 350 nm layer structure.

The technology display settings are presented in Figure D-3.

```
#layer-num      type-num        Laygen-color    Laygen-pattern          layer-order
5               0               0               none                    10
8               0               16749824        none                    10
21              0               0               none                    10
23              0               0               HorizontalLinePattern   20
24              0               16776960        HorizontalLinePattern   30
10              0               65280           LeftLinePattern         40
20              0               16711680        FillPattern             50
30              0               16756922        FillPattern             60
34              0               65407           FillPattern             70
35              0               9445616         LeftLinePattern         80
36              0               16738724        CrossPattern            90
37              0               0               LeftLinePattern         100
38              0               16729344        CrossPattern            110
39              0               16776960        RightLinePattern        120
41              0               16749824        CrossPattern            130
42              0               2984535         LeftLinePattern         140
61              1               16711680        FillPattern             150
61              2               9445616         FillPattern             150
61              3               0               FillPattern             150
61              4               16776960        FillPattern             150
61              5               2984535         FillPattern             150
200             1               454545          CrossPattern            160
200             2               16776960        CrossPattern            170
200             3               13422920        CrossPattern            180
200             4               -454545         CrossPattern            180
```

Figure D-3 – AMS 350 nm display settings.

The technology design rule map is similar to the one presented in Figure C-4 for the UMC 130 nm technology design kit, however the manufacturing grid for the current technology is 25 nm instead of 5 nm. Finally, figure D-4 shows the available parametric module generators for the current technology design kit.

```
//Component Generators
NMOS(int width, int length, int m, int angle, String bulk, Boolean sideways);

PMOS(int width, int length, int m, int angle, String bulk_, Boolean sideways);

Capacitor(int width, int length, int n, int m, int angle);

//Complex Geometric Generators
Contact(double center_x, double center_y, Layer.ID top, Layer.ID bottom, double w_width);

MultipleContact(double center_x, double center_y, Layer.ID top, Layer.ID bottom, int orientation);

Guard_Ring(Layout l, LinkedList<Point> bounds, Boolean nwell);

Power_Strip(Layout layout, String pin_name, int min_width, PowerNetMode mode,
            List<IShape> ignore_shapes, HashMap<String, Point> bounds);
```

Figure D-4 – AMS 350 nm module generator.