Analog IC Layout Automation

by

Youssif Rabie
Mohamed Ewais
Nadeen Ibrahim

Thesis submitted to the
Department of Nanotechnology and Nanoelectronics Engineering
In partial fulfillment of the requirements
For the B.Sc. degree in
NANENG 599

Zewail City for Science and Technology
University of Science and Technology

Acknowledgements

We would like to express our sincere gratitude to our supervisor Dr. Hassan Mostafa for his continuous support, assistance, and motivation throughout the project. This project was partly supported by Si-Vision under the supervision of Eng. Soha Hamed who we sincerely thank for her support, encouragement, immense knowledge and motivation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General Introduction and Overview of the Topic

In 1940s, Bell Labs researchers John Bardeen, Walter Brattain, and William Shockley introduced the first point-contact transistor, an invention that significantly advanced the field of electronics. This technological breakthrough played a pivotal role in the subsequent emergence of the first integrated circuits, which first emerged in the late 1950s. Analog circuits constituted a crucial element of the initial integrated circuit designs, facilitating various applications such as audio amplification and signal processing. They were utilized in voltage regulators, oscillator circuits, and phase-locked loops.

The layout process starts after the end of the schematic design phase in the analog IC design flow. The layout process involves determining the physical arrangement, geometry, and material of the components and interconnects of a circuit. Moreover, layout involves defining the bond pads for the chip's connections to its peripherals. This includes the placement of transistors, resistors, capacitors, and other circuit elements, as well as the routing of interconnects between them. The layout process is the bottle neck in the analog design flow as it plays a critical role in determining the performance and reliability of the final IC design. Moreover, an optimized layout can reduce power consumption and increase the overall efficiency of the circuit. In addition, a well-designed layout improves the manufacturing yield of chips and increases the chip's tolerance to variations in the manufacturing process. A robust layout can help to minimize parasitic capacitances and resistances that degrade the circuit performance.

Analog IC layout process is a manual and takes timely tasks which requires skilled engineers in contrast to digital IC layout which is fully automated. However, as IC designs become more complex and the number of transistors on a chip increased exponentially over the years, manual design became impractical. In response, analog layout automation tools are now developed to help engineers create optimized layouts quickly and efficiently.

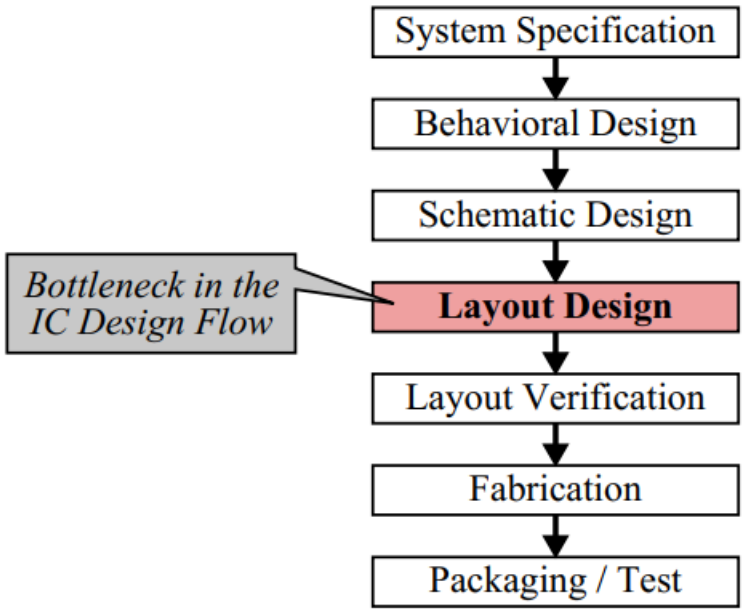


Figure 1.1: Simplified illustration of the integrated circuit design flow.

Today, analog layout automation is a critical component of the IC design process, and there is intensive research to develop software tools to help designers in creating high-performance, low-power analog circuits. Advanced algorithms are used to optimize layout parameters including placement, routing, and parasitic effects, to help designers achieve the best possible performance for their designs.

1.2 Problem Definition

Analog IC design cycle is much slower compared to digital IC design. The digital IC design cycle is now fully automated through scripts and software. The digital designer mainly focuses on system design, module partitioning, and writing HDL code for each module in the design. Afterwards, EDA tools synthesize the code into netlist, which is then converted into a full layout in a GDSI file. In contrast, analog design and layout lag far behind. This problem results from two main reasons.

1.2.1 challenges and limitations of the traditional manual layout

The first reason is related to the challenges and limitations of the traditional manual layout design process for analog integrated circuits.

The traditional manual layout design process for analog integrated circuits involves the placement and routing of circuit components to create a physical layout that meets the design specifications. This process is often time-consuming and requires expertise

and experience in circuit design, manufacturing processes, and design rules. The manual layout design process can be time-consuming and may require multiple iterations to achieve a satisfactory design. This can lead to delays in product development and time-to-market. Additionally, the layout design process requires expertise and experience in circuit design, manufacturing processes, and design rules. This makes it challenging for designers who lack this expertise or experience.

Manual layout design is error-prone, as designers may make mistakes or overlook design rules, leading to circuit malfunctions or poor performance. Moreover, the manual layout design process can result in variability in design quality, as different designers may have different design styles or interpretations of the design specifications. The manual layout design process may not be scalable for complex circuits or large designs, as it may require significant manual effort and time. Furthermore, the manual layout design process can be costly, as it may require specialized tools and expertise and may lead to design iterations and delays.

1.2.2 Challenges related to analog design its self

The second reason for the slow pace of analog IC design pertains to the design process itself. In contrast to digital design, where the primary objective is often the size of the chip, analog design focuses on optimizing circuit performance, matching, speed, and other functionality concerns. Unlike digital ICs, there are no fixed heights for standard cells or grids to guide placement. Additionally, the number of design rules governing analog ICs is significantly fewer. As such, designers are more concerned with how the circuit will perform, rather than meticulously ensuring that the cells fit together according to a predetermined set of guidelines. Analog layout designers prioritize four primary criteria when designing analog circuits, including design specifications, design objectives and restrictions, matching, and design hierarchy levels. Furthermore, the process encompasses device generation, floor planning, placement, and routing.

Design specifications, objectives and restrictions

Layout design is a complex optimization problem that involves searching for an optimal solution within an immense solution space. In this process, design restrictions establish a valid region within the solution space, while design objectives specify an optimum solution within that valid region. Design restrictions can be categorized into three main groups. Technological restrictions are intended to ensure the manufacturability of the integrated circuit and are formulated as geometrical design rules that can be very complex. However, they typically belong to one of the following categories: minimum width, minimum distance, minimum overlap, or minimum enclosure. Functional restrictions, also known as electrical restrictions, aim to ensure the proper electrical functioning of the circuit. They can be further divided into circuit-specific requirements (e.g., to prevent unwanted coupling effects) and process-specific requirements (such as limiting current density in electrical wires to prevent electromigration). Design-methodical restrictions are deliberately introduced to reduce the complexity of the layout design problem, making it more amenable to

computer-aided automation approaches. For example, layer-dependent wire directions are introduced to facilitate automated routing (e.g., metal1: horizontal, metal2: vertical).

While design restrictions are strict requirements that must be met, design objectives represent gradual optimization goals that designers pursue to the best of their ability. They can be broadly classified into economic optimization goals and functional optimization goals. Economic optimization goals include minimizing product costs (e.g., by reducing total chip area and the number of required metalization layers) and minimizing development costs (e.g., by reducing the design effort through automation). Examples of functional optimization goals include minimizing total wire length and optimizing heat dissipation to prevent critical hot spots in the chip.

Matching

Matching is an essential technique employed in analog design to achieve high accuracy of analog signals through "electrical symmetry". Due to the manufacturing tolerances exhibited by various steps in the IC fabrication process, the absolute exactness of a circuit's components is extremely poor. However, the parametrical deviation among circuit components of the same type is relatively good. Analog circuits are designed and laid out in a way that takes advantage of this relative exactness by "matching" certain components. Matching involves equalizing the electrical behavior of these components in relation to each other so that variations, including manufacturing tolerances, thermal gradients, and parasitic effects, do not affect their overall electrical functioning. Without this technique, achieving a functional analog integrated circuit would be virtually impossible. Therefore, obtaining a good matching is one of the most crucial tasks in the daily work of analog layout designers. The matching of circuit components requires that these components are of the same type and have equal dimensions, which are taken care of by the circuit designer. Further matching measures in the layout often involve placing these components with consistent orientations and aligning them in a compact, interdigitated, common centroid arrangement.

Design hierarchy levels

The arrangement of analog integrated circuit (IC) layouts is typically hierarchical, with design components nested within other components. The fundamental entities, such as transistors, are presented as primitive devices without subhierarchy. Higher-level design entities, which form functional units, can also be encapsulated as a hierarchical cell. These functional units become modular library components that can be instantiated in a layout, that is, inside another cell. To avoid any confusion when discussing cells, it is crucial to classify them based on their characteristics in the design hierarchy. This thesis proposes the following terminology for classifying cells: primitive devices, simple modules, advanced modules, blocks, and chip. Primitive devices are the most fundamental components in layout design, and they are made up of plain polygons. They typically represent transistors, resistors, and capacitors, and other structures such as wells and guardrings are

also frequently implemented as primitive devices. Simple modules are basic analog circuits with highly regular patterns. They consist of a set of identical primitive devices that are arranged in a strictly tiled, matrix-like arrangement. Advanced modules are also basic analog circuits, but they are composed of different components, including primitive devices and simple modules. The components' relative positions follow a well-established, often symmetric arrangement, regardless of their actual dimensions. Blocks are large hierarchical cells that perform high-level electrical functions. They contain unequal devices and modules, assembled in a rather irregular arrangement that must be determined individually by a layout expert in each case. The chip is the top-level entity of a layout, which is composed of hierarchical layout blocks. In the case of a mixed-signal chip, both analog and digital blocks are included. The chip may also contain single devices such as power transistors that realize high-current output stages. Although a family of chips may share topological similarities, each chip is always a unique design.

Device Generation, Floor planning, Placement, Routing

Device generation is the task of designing layouts for various components of an input circuit based on their size. This task was previously the responsibility of IC design teams, but today, vendors supply primitive devices as part of a process design kit (PDK) through procedural generators. Analog design requires layout variability to fulfill design challenges, and primitive devices have diverse layouts due to device folding. During floor planning, device generation is crucial for calculating the size of a layout block.

Floor planning involves minimizing overall layout space, total wire length, and optimizing power supply and current flow to reduce financial and electrical costs. The placement of components requires rotation and varying their arrangement without interfering with their electrical performance. Components must fit within the predetermined shape established during floor planning. Some blocks must be placed near the chip boundary to reduce wire length, while others require a specified minimum distance to prevent delicate signals from being disrupted by undesirable thermal and electrical impacts.

Routing space between layout components is required to facilitate their routing, and sensitive circuitry often prohibits electrical wires from being routed above these components. The number of metal layers available for routing must also be limited. Routing aims to reduce the number of vias and metal layers and homogenize the total wire density. The size of a via and the width of a wire segment must be determined according to the anticipated current load. The primary goals of routing are to reduce the number of vias, the number of metal layers, and to homogenize the total wire density.

1.3 Objectives

When attempting to automate the layout design of an analog circuit, circuit classification becomes an important tool. It is nearly impossible to create a scripted tool that can automate the layout design of all analog circuits, as they vary in their restrictions regarding

matching and floor planning depending on their functionality. For this study, we will be focusing on the folded OTA, which falls under the Low Frequency classification. The most crucial aspect of the OTA circuit is matching the passive and active elements. By using classification, we can narrow down the expected inputs for the scripting tool and improve the identification and automation process.

1.3.1 Pattern Recognition

The input for the automation script will be a SPICE netlist text file that includes all devices and nets in the schematic, along with their widths and lengths, and the number of fingers and multipliers for each device.The script will then identify the building blocks of the OTA circuit, which include various current mirrors, differential pairs, and active loads. A suitable matching pattern will be applied to each building block. Next, the floorplan and placement step will be executed, which can be run multiple times to obtain the best routable netlist. During placement, symmetry constraints across the entire device will be taken into account, and connected blocks will be placed in close proximity to one another.

1.3.2 Block and Global Routing

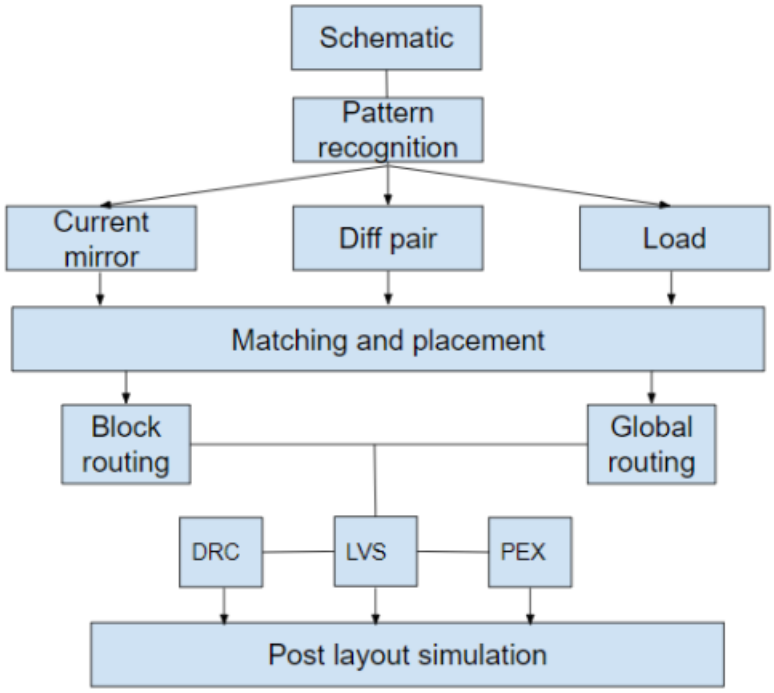


Figure 1.2: Diagram of the whole layout automation cycle.

The output of the floor planning step will provide the suitable coordinates for the matched devices, which will then be placed using SKILL scripts. The same script will be

used to perform internal routing for all blocks, as well as global routing between different blocks.

1.4 Functional Requirements/product specification

Analog layout design is a crucial aspect of analog design that involves placing and routing components to meet the design specifications. This process is usually time-consuming and requires high levels of expertise, which makes it a bottleneck in the analog design process. Therefore, it is important to explore ways of automating this process to accelerate the design process. To achieve this goal, it is important to understand the challenges that need to be addressed. One of the main challenges is the need to recognize the schematic of the design and extract the required data from it. This is essential for automating the layout process, as it allows for the tool to have a clear understanding of the design. Another challenge is the need for accurate floor planning. Floor planning involves estimating the area of the design and organizing the components in a way that meets the design specifications. This is important as it determines the overall layout of the design and affects the performance of the final product. The placement step is also critical as it involves placing all components in the layout to meet the design specifications. This step requires a high level of expertise and can be time-consuming, especially for complex designs. Finally, the routing step involves connecting the placed components to create a complete circuit. This step is divided into detailed routing and global routing. Detailed routing involves connecting the components within a small area, while global routing involves connecting components over a larger area. Both of these routing techniques are important for creating an efficient and reliable circuit.

1.5 Report Organization

The structure of this graduation thesis is organized into the following sections. Section 2 presents the standards that will be utilized throughout the study. In Section 3, a review of the literature and market trends regarding placement and routing will be discussed. Section 4 outlines the proposed work that will be carried out as part of this research project.

Chapter 2

Standards to be used

The main objective of this graduation project is to enhance an existing tool and create a comprehensive layout automation framework capable of handling all stages of the layout process. This includes pattern recognition from schematics, generating placement patterns, optimizing them, auto-routing parameterized cells (PCs), performing global routing among main blocks, and conducting verification checks such as Design Rule Checking (DRC) and Layout vs. Schematic (LVS).

Our automation tool will focus on testing the Recycle Folded Operational Transconductance Amplifier (RFOTA), a fundamental analog circuit application, which is a key component of the Analog to Digital Converter (ADC). To achieve this, we will divide the tool into two primary software scripting components: a Python script for pattern recognition, optimization, and placement pattern generation, and a Skill scripting component for PC routing, global routing, and verification.

To ensure that our software meets industry standards, we will utilize the ISO/IEC/IEEE 12207:2017 standard, which is an international standard for software lifecycle processes that defines all the necessary steps for developing and maintaining software systems, including the outcomes and/or activities of each process. Throughout the development process, we will adhere to these standards, from software acquisition to the design of the Graphic User Interface (GUI). The software for the automation tool is the primary component of this project, and we will strive to maintain the highest standards throughout all phases of its development.

Chapter 3

Market and Literature Review

Although there are various proposed automated methods for analog layout, the full analog design till that day is done manually. It is believed that the existing automated solutions could not yet reach the high level of efficiency that human designers guarantee. However, as will be explained, there are many tools that have been recently introduced that are capable of partially helping in the automation of analog layout design but with some limitations that require additional solutions. Hence, the possibility of finding a tool for layout automation is really promising.

3.1 Placement

The first step of layout design is placement, where the physical floor plan of the circuit is being constructed. It is quite challenging to perform placement in an automated manner as there are many constraints that need to be considered in the process, such as minimum area, DRC rules and parasitic effect. Additionally, thermal gradient and stress gradient on the die are issues that need to be eliminated by applying the concepts of matching, symmetry, and proximity. Matching is an essential step to prevent variation in device parameters as a results of process variation, lithography variation, and temperature gradient on the chip, it could be applied as common centroid, common gate, or interdigitated pattern [1]. Placing devices symmetrically around a common axis, in some cases could be more than one common axis, is crucial especially for differential analog circuits as any mismatch could cause problems like degraded power supply rejection ratio and high offset voltages. Moreover, devices placed symmetrically are less prone to thermal gradients and undesired oscillations [2]. Proximity problem such as well proximity is most prominent for current mirrors, where the threshold voltage of a device significantly changes shifting the saturation point. Proximity is highly affected by blocks separation and guard rings insertion [3].

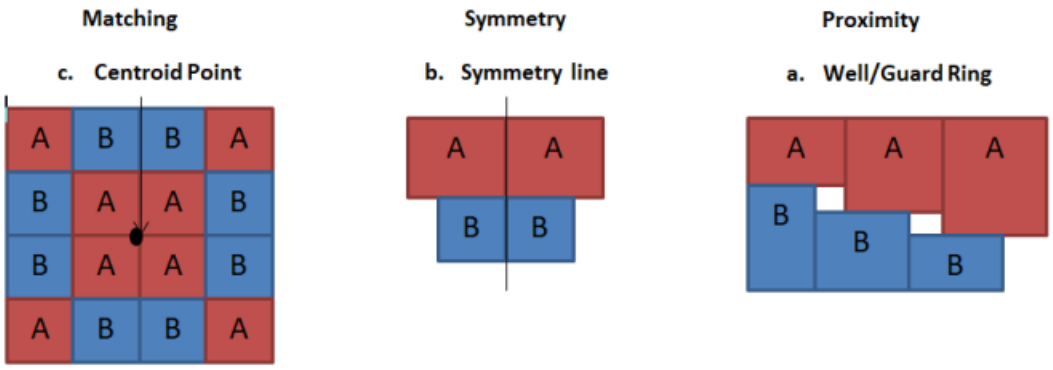


Figure 3.1: a) proximity, b) symmetry, c) Matching.

3.1.1 The Absolute Representation

The main feature of the absolute representation is that modules are assigned to an absolute coordinate on a plane with no grid, where any manipulation required on the module is done by changing these coordinates. Nevertheless, the absolute method could result in placing overlapped modules which requires an additional post processing step to resolve this problem. Therefore, the absolute method requires a relatively long computation time when it is compared to other methods [2].

3.1.2 The Topological Representation

The topological representation is different from the absolute representation, the cells are not recognised in terms of their fixed coordinates but rather in terms of the topological relations between the other cells. In other words, the positions of the cells are identified in a relative manner [1]. There are two types of topological representation, slicing representation and non-slicing representation.

3.1.3 Slicing Representation

A slicing representation consists of horizontal and vertical lines that slices the main rectangular area into smaller rectangles where each rectangle is called a room that can acquire set of modules of designed circuits. A slicing floor plan can be represented as a slicing tree as shown in figure 2 or as a polish expression as shown in the expression below [4]. Polish Expression: 2 3 * 1 + 4 5 + 6 7 * + *

3.1.4 Non- Slicing Representation

The non-slicing representation is known for the usage of constraints graphs, the vertical relations of modules are associated with the vertical constraints, while the horizonal ones

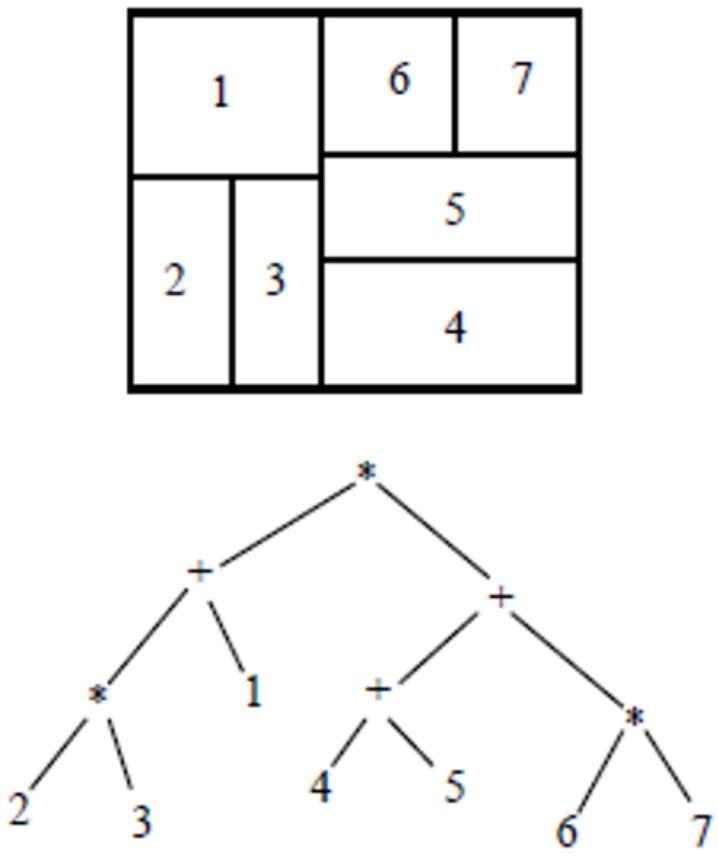


Figure 3.2: Slicing floor plan, slicing tree.

are associated with the horizontal constraints. There are various algorithms developed for non-slicing representation, such as bounded slicing grid structure (BSG), corner sequence (CS), sequence pair (SP), the ordered tree, B*Tree, and transitive closure graph based (TCG).

3.1.5 Routing

Routing is a crucial and complex stage in the layout of analog circuits due to their performance dependency on circuit parasitics. Stray resistance in interconnects causes voltage drops, which necessitates the use of short and wide wires or higher metals. In order to prevent crosstalk and coupling effects, it is essential to leave routing spaces between sensitive circuits that do not permit electrical wires above them. Crosstalk results from unwanted communication between signals caused by the coupling between two parallel nets. Digital sections of the circuit generate coupling noise that is coupled into analog circuits through power supplies and over the substrate, leading to performance degradation. Parasitic resistance and capacitance may also result in significant signal delays. Some signals are so

sensitive that they require extra protection through shielding, such as the differential signal in the differential pair [5].

The way metal is selected for routing is another factor to consider, and it is generally agreed upon to use the first two available routing metals in lower design levels and use the remaining metal layers for routing the top-level. To connect different metal segments, vertical interconnect access (via) is used. The current load determines the wire width and via size. It is crucial to decide on the wire width to avoid electromigration, which occurs when material moves due to the gradual transport of ions in conducting wires, causing voids or hillocks that may result in an open or short circuit.

The main objectives of routing are to reduce the number of metal layers, to reduce the number of photolithographic masks, homogenize the overall wire density, minimize the number of vias in fear of metal layer crossing, and fully verify routing with minimal parasitics. Some nets require symmetric routing. This project also aims to develop a technology-independent automated routing skill code. In this section, the analog routing problem definition is further discussed, along with state-of-the-art routers and suggested solutions. Digital solutions are also included since digital and analog routers face similar challenges in specific parts [6].

3.1.6 Problem Definition of Analog Routing

The analog routing problem deals with finding suitable paths among the electrical nets terminals, while meeting the routability and analog constraints. The problem involves a three-dimensional graph with terminals and non-terminals as its vertices. The terminals include device and input/output pins, and every electrical net has terminals connected with one another. The non-terminals are the intermediate vertices that connect the terminal in the graph. The netlist is the input of the analog routing problem, while the paths among the terminals in the netlist are the output.

There are two main types of analog routers: general-purpose routers and analog constraint specific (ACS) routers. General-purpose routers attempt to satisfy multiple constraints simultaneously, while ACS routers focus on a particular constraint such as wire length, electromigration, IR-drop, and mismatch effect. The sequential analog routers route the nets one after another, while the ACS routers aim to satisfy a distinct purpose.

Academic layout synthesis tools started the development of general-purpose analog routers. ANAGRAM II is an example of such tools, which uses an algorithm for wire expanding to look for the path that achieves the lowest constraints such as wire length and crosstalk. This tool was updated to become one of the first commercial tools in analog layout [7].

Many routers were introduced to handle several constraints, like exact length and topology matching and symmetry, using integer linear programming (ILP). However, their design had low efficiency because of the high complexity of LP. A configurable design was proposed that uses a unified constraint method from schematic to layout, and the layout was recursively partitioned according to the driven constraints from the placement. A*

algorithm and maze router were used to wire critical and general nets. The maze routing algorithm is a low overhead method that finds possible paths between any two nodes, while the A* algorithm is a search algorithm that finds the shortest path between the two nodes [8].

Electromigration [1] is becoming more challenging to avoid as the feature size shrinks. Many post-processing algorithms have been proposed to promote the layout reliability against EM failure. Another idea for solving EM-aware routing was proposed using violation and reserved-path finder schemes to set up LP formulation so that a minimum routing area could be achieved [9].

3.1.7 Routing Algorithms

Using abstraction, the devices is assumed to get presented by points and routings to be lines connecting these points, concepts and algorithms of graph theory can be used in analog layout designing [10]. One of the widely used algorithms is the Maze Routing algorithm [11], also known as Lee's Algorithm. Its primary objective is to find the shortest path between two locations within a maze, namely the entry and exit points. Furthermore, it can identify the number of paths between these two nodes. To accomplish this, the algorithm employs a priority queue that scans cells in ascending order of distance from the starting point to the endpoint. Once the final point is reached, the path of cells is checked in reverse order back to the starting point to determine the shortest path efficiently [12]. The major disadvantage of this algorithm is its long runtime. However, optimization is possible at the cost of increased complexity.

Due to the fact that routing is an open problem, more than one algorithm may be used depending on the design. The second is the Dijkstra Algorithm, which is used to identify the shortest paths between nodes in a graph. A commonly used variant of this algorithm is used to identify the shortest paths from a selected source point to all other points in the graph. During each step of the algorithm, the distance from each node to the source node is stored and updated until the shortest path is reached. Once the shortest path between the source node and another node is identified, that node is marked as "visited" and added to the path [13]. This process continues until the shortest path between the source and final nodes is found. However, the Dijkstra Algorithm [14] fails when dealing with negative edges, and it cannot handle 3D graphs, which are its main drawbacks.

Chapter 4

Project Design

4.1 Project purpose and constraints

4.1.1 Project purpose

For many years, analog layout designers have grappled with the arduous task of managing multiple tools for simulation and verification across an array of designs. This complexity is exacerbated by the requirement to wield expertise in several programming languages, such as TCL, Python, SKILL, and Shell, each with their own unique syntax and operational quirks. This convoluted workflow can often result in inefficiencies and errors, and thus, there is a compelling need for a more streamlined, integrative solution.

In response to these challenges, the idea of creating a simple yet powerful unified platform - a Python-based Linux desktop application - has been conceived. This proposed application serves as a holistic platform to manage the various stages involved in analog layout design. By adopting Python as the underlying programming language, the platform capitalizes on the extensive libraries and versatile capabilities of Python, promoting greater ease of use and accessibility for designers.

The innovation of this platform extends beyond its unified nature and Python basis. This application will have the ability to convert a schematic to a layout automatically, eliminating the time-consuming and error-prone process of manual conversion. This feature offers designers a unique opportunity to expedite the early stages of design, allowing them to devote more time and energy to critical aspects of the design process.

Additionally, the platform incorporates a novel approach to post-layout simulations. By simply clicking a mouse button, designers will be able to initiate and run these simulations. This feature further simplifies the design process, making the task of simulation, which is typically a complex and laborious process, more user-friendly and efficient. In doing so, this platform promises to revolutionize the workflow of analog layout designers, potentially leading to significant improvements in productivity and design quality.

4.1.2 Constraints

Constraints for First Component

The initial module-pattern acknowledgment and generation- is mainly layout depend upon the concept of obtaining the optimum matching pattern for below obstructs to lessen the symmetry, matching, as well as proximity issues of pattern generation. The function was as getting the netlist info of schematic, making use of terminals of every device to specify the sorts of subblocks, standard layout formula to obtain optimum area, producing matching pattern for each and every sub obstructs- usual centroid or interdigitated, and also finally composing the output in a text documents in a standard format to be prepared for the second module-placement and routing. The main constraints in this module were working with different PDK technologies, estimation of area due to the difference of tool properties from schematic to format, and generation the names and also dummies to be matched with the Cadence virtuoso procedure.

Constraints for Second Module

The restraints of this job can be divided into two main groups. The first group has the formalized constraints while the second group includes the non-formalized ones. The defined constraints are easier to deal with as they are distinct, do not need to be interpreted as they are straight, can be checked using mathematical or logical models, and they are usually written clearly in a file stored in the used PDK. The non-formalized constraints are related to the design itself so they might contain some unpredictable conditions if the circuit layout is not well understood. They depend on the human experience so they usually are educated by the layout engineer verbally or by agreed tags on the schematic. In the SKILL device, The DRC and LVS constraints were followed for the chip to be able to get produced. The non-formalized constraints can be split into two parts; the matching and the routing parts.

Matching is in the pattern generation component which is in the Python code. Matching is performed in every element of the OTA circuit to minimize the differences in device specifications due to distance, lithography variations, process variations, rotation, biasing, and temperature gradients on the chip . There are rules that must be followed to get better matching which are:

1. Putting the transistors in close proximity.
2. Positioning the transistors in low-stress gradients area.
3. The layout should be as compact as possible.
4. The transistors positioning should be in the same direction.

For the current mirror pattern, common centroid matching pattern is used as it reduces the stress and temperature gradient as much as possible. Its main object is that the

centroids of each device should coincide . Inter-digitization is used in differential pair as all the transistors are in an interleaved pattern. The load pattern is also inter-digitization as they consist of 2 transistors each like the differential pair.

For the routing constraints, routing is matched for a single device so that each transistor in a device is affected by the same parasitics and stress. Electromigration is also considered as the length of each route is determined based on the current being carried by it. In the differential pair, the input (Differential signal) is routed close to each other with no other signal merging with them in order to cancel out any noise. No routing was done across any device to reduce the parasitics such as merging effects. To achieve the matching requirements, the same number of metals is used in each row of a single device. Source sharing is obtained for each device besides the PMOS load as no terminal is shared. Source sharing was used in order to reduce area and parasitics. For devices with odd number of multipliers, an extra dummy is added to preserve the matching pattern. The bulk connections were made to have near minimum length as there is no large current passing through the body of the transistors. Dummies are added at both sides of each device as different undercut due to the etching can be greater on the sides than in the center.

4.2 Project Technical Specifications

This module was mainly written in Python 3.11 language using the online compiler provided by pycharm educational liscence, which adheres to the ISO/IEC/IEEE 12207:2017 standard requirements. This choice was made due to the difficulty of executing this component using skill code, as it involved a significant number of computations and formulas. The technological requirements were primarily focused on the code itself, and this will be demonstrated.

4.2.1 Software Program Requirements Specifications

The software application requirements specifications focus on the functionality and organization of the code. The quality of the output is closely tied to the design quality aspect. The software specifications will be discussed in detail in the following subsections.

Product Features

The product features of this project are outlined below for each block:

- **A. Current mirror**
    1. Positioning
    2. Source and drain routing
    3. Gate transmission

Technical specifications	**How it was achieved**
Accessibility	Python code can be run easily with uploading the netlist file and only one click on run. Adding a demo video is attached with tool.
Availability	The code can be run on Google colab at any time freely.
Data quality	The input must be in text format and also the output.
Human error	The assumption that tool will be worked with professional engineer, the human error is be limited by only error in number of figures and multipliers, else any error the error massage will be appeared.
Interoperability	The code can be worked with any netlist file from number of technologies are assigned in consideration before.
Maintainability	The code is clean with free errors and do not need any modification to do its task.
Performance	The average time is depended on the number of multipliers are assigned, but it not be greater than 50 seconds in worst case and 18 seconds in best case.
Productivity	The output is automated and updated itself with any number of runs.
Standards	The code is following the ISO/IEC/IEEE 12207:2017 standard requirements for compiling and execution.

Table 4.1: Technical specifications and how they were achieved

4. Dummy routing
5. Mass tie and guard ring routing
6. Pin placement

Similar features with different specifications are utilized in the differential pair and the load. These features will be elaborated upon in Section 4.5.

Operating Environment

The operating environment utilized in this project is Cadence Virtuoso. The positioning and routing parts are implemented using the Skill language. However, the pattern recognition and pattern generation parts were implemented using Python to facilitate the utilization of complex formulas and achieve improved outcomes.

Constraints on Design and Implementation

The design and implementation process is subject to certain constraints imposed by the coding environment. These constraints, along with the requirements of the design tool, are summarized in the table provided below.

User Interface

The tool should allow users to specify certain required inputs, such as the facet ratio used to determine the positioning of the blocks and the source current required to determine the width of the routing wires. Additionally, it should display the output format once the code execution is completed.

Performance Requirements

The following performance requirements are expected:

1. Short runtime (in seconds).
2. Low space complexity.
3. Error-free execution when applied to different technologies.

4.2.2 Layout Quality

The generated output layout quality of the code consists of two key aspects: capability and consistency, as illustrated in Figure 13. These aspects vary according to the circuit's specifications. In this project, these facets are developed for the current mirrors, differential pair, and the load. The specific facets will be discussed in detail in the subsequent subsections.

4.2.3 Functionality

Functionality of an Integrated Circuit (IC) is contingent upon the adherence to capability requirements, ensuring the fabricated IC derived from the outcome format of the code operates correctly. It considers three primary factors: precision, reliability, and yield, which are not explicitly defined in the designed code and depend on the design engineer's experience. Therefore, the implementation of such elements will likely differ across different blocks.

Precision denotes the ability of the IC to operate within an acceptable range. To achieve this, it is imperative for the generated format to generate robust and accurate signals, capable of functioning adequately under minor or severe conditions. This requirement involves devising the best matching strategy in placement and routing for each block.

Maintaining the same orientation is crucial to ensure all block signals are influenced by an identical gradient.

To enhance signal precision, dummies should be added on both sides of each device to shield it from Layout Dependent Effects (LDE) such as well proximity effect and diffusion length. Consideration must also be given to the bulk contacts, ensuring a uniform equidistant bulk for all devices, which will provide a consistent substrate bias across the entire IC, thereby producing accurate results. To minimize the parasitics as much as possible, channel routing is employed over device routing, leading to increased reliability and precision.

Reliability measures the ability of the chip to endure constant cycles that may cause electromigration and failure. To prevent electromigration, which is discussed in Section 3.2, the width of routing wires must be calculated considering the current capacity of each metal layer, without using minimum values. This is to mitigate future electromigration issues.

If the calculated width exceeds the maximum available wire width, stacking strategies are used, employing two or more metal layers until the current is less than the maximum width. Repeated cycles may also lead to Minority Carrier Injection in the substrate, causing power leakage and potential device lockup. This issue can be mitigated by adding guard rings.

Furthermore, if the circuit is subjected to high voltage, the dummy devices could suffer breakdown as their gates are often connected to GND or VDD. To preclude this issue, the dummy gates should be connected to connect low or connect high terminals, rather than directly to the power rails.

The yield factor pertains to the percentage of properly functioning manufactured chips relative to the total chips produced. Certain design practices could adversely affect the yield and should be avoided. For instance, reducing the number of vias and using more than one via decreases the chance of electromigration, hence enhancing reliability. This also lessens the chance of via issues causing chip defects. Dual vias are often employed to circumvent these problems.

4.2.4 Consistency

Consistency is a requisite attribute, ensuring the generated layout is compatible with an analog design tool harmonious with Cadence tools and verification methodologies. The code should be versatile enough to be used after Process Design Kit (PDK) updates, with different PDKs, and in various projects. To articulate the concept of consistency, "Maintainability" is introduced. It deals with managing the layout along the toolchain, either in combination with other elements in the tool such as a floor planner, or during verification.

Achieving consistency requires modularizing certain layout structures. This modularity facilitates PDK updates and the management of technology-dependent constraints without

altering the code. The tool should manage the devices as components susceptible to input changes, thus constants should not be used.

For the tool to pass Layout versus Schematic (LVS), the inclusion of dummies must be considered in the schematic to make it compatible with the layout. The tool should be technology-independent to be used effectively without necessitating changes in the code. However, it is only compatible with Cadence, as it is written in SKILL.

The interface with Python should be established efficiently to enable seamless data exchange between the layout tool and Python.

4.2.5 Layout Alternatives and Justification

Throughout the project duration, we encountered multiple instances where various design options were presented. These scenarios required an evaluation of the advantages and disadvantages of each alternative, leading us to select the most suitable design based on this assessment. A few of these instances are briefly discussed in the ensuing subsections.

Design Alternatives and Justification for the First Module

The first module, encompassing pattern recognition and generation, presented numerous layout alternatives or strategies for achieving optimal output. The selection depended heavily on the experience-based opinions from both the industrial and academic supervisors. The specifics of these alternatives are discussed in the following subsections.

Design Alternatives for Pattern Recognition The initial approach proposed involved obtaining the types of subblocks directly from the user. This method was limited as it required an in-depth understanding of the layout and was consequently only applicable to a subset of users. The second method allowed the user to choose the matching patterns and devices, although this could lead to significant errors and routing complications if improper input was given.

Following an extensive literature review in the area of analog pattern recognition, a tool utilizing the concept of complexity level was introduced. This tool employed the sharing of device net terminals to identify the block type, with further details provided in Section 4.4.1.3.

Design Alternatives for Pattern Generation The first proposed approach involved taking the number of rows from the user. However, this method was somewhat limited and proved ineffective on a larger scale. After conducting several trials and consultations with experts, an alternative method was adopted. This approach used the device bounding box information and the current requirement to gauge the required routers, provided in the netlist data. An optimal algorithm was chosen to estimate the area and determine the number of rows for each subblock, as discussed in Section 4.4.1.

Design Alternatives and Justification for the Second Module

Matching Technique Used and Justification In this topology, structures such as differential pairs and current mirrors were used, where matching of device characteristics is crucial. The goal of these matching techniques was to minimize differences in device parameters due to distance, lithography variations, rotation, process variations, biasing, and temperature gradients on the chip. By using dummies on the sides, all multipliers were ensured to undergo the same environmental conditions, reducing etching variations between edge and center transistors.

In the differential pair block, inter-digitization was employed for matching, with all transistors in an interleaved pattern (Figure 14). The main reason for choosing this strategy was to maintain equal path lengths and thereby minimize mismatching effects. Conversely, in current mirror blocks, a common centroid strategy was used for matching (Figure 15).

In differential pair devices, shielding was used around the paths carrying the input signal to isolate sensitive signals from other routing, reduce noise impact, and prevent crosstalk. Lastly, dummies were added to counteract variations from over-etching at the edges (Figure 16).

Detailed Horizontal Routing Paths and Justification In the differential pair and P-load blocks, the horizontal paths of the inputs at gates could be placed in two locations: at the end of the block or between the rows. The first approach facilitates global routing, providing easier and shorter connections with other devices. However, it also presents disadvantages that can affect the performance of the matching.

The second approach addresses this issue by positioning devices further away from other routes, reducing the impact of noise and crosstalk. Additionally, it achieves better matching performance as both input routes have almost the same length. Moreover, it results in a shorter route, thereby minimizing parasitic effects. Hence, the second approach is chosen to enhance the performance

of the differential pair.

Detailed Vertical Routing Paths and Justification Two strategies were examined during routing: "over device routing" and "channel routing". The first strategy's advantages are smaller routing area and simpler global routing. However, it increases parasitics since routes pass over the device, potentially leading to failure under certain conditions.

Conversely, channel routing addresses these issues by confining routes to only pass over the metal terminals, achieving the least possible parasitics. Although it may result in a larger routing area, this approach was chosen for this project to ensure the best performance of crucial device matching.

4.2.6 Description of the Selected Design

Overview of How the Subsystems Interact

When the graphical user interface (GUI) is activated within a specified directory, it first initializes the Cadence Virtuoso environment, a critical precursor to the operational functionality of the platform. Following this initiation, the GUI presents a window, prompting the user to make a selection between two options: creating a new schematic view or selecting from an existing schematic view. These options offer flexibility to the user, accommodating both the inception of new design projects and the continuation of existing ones.

Should the user opt to create a new schematic view, the GUI responds by presenting a dialog box. This interactive element invites the user to provide specific details, such as the library name and the cell view name, necessary for the creation of a new schematic view. This user-driven input mechanism ensures that the new schematic is tailored to the user's precise specifications, encouraging a more efficient and personalized design process.

Alternatively, if the user elects to choose from existing schematics, the GUI facilitates this process by presenting a comprehensive list of available libraries, each containing multiple cell views. The user is then able to peruse these options and make an informed selection of the cell view that best suits their current needs.

Once a cell view has been selected, whether newly created or pre-existing, the GUI undertakes the task of layout generation and post-layout verification. This encompasses comprehensive checks including, but not limited to, Design Rule Checks (DRC) and Layout Versus Schematic (LVS) checks.

These advanced features, which are readily accessible at a single mouse click, ensure the accuracy of the layout generation and the integrity of the post-layout verification process. This GUI logic ensures that users can proceed with their design tasks confident in the knowledge that the platform is taking appropriate steps to maintain design quality and uphold adherence to established design rules.

4.3 Block Diagram and Functions of the Subsystems

Placement of Blocks (4.5.2.1)

In this action, the output of the previous step (gadgets' collaborates) is read and utilized. Using features in skill, all instances are placed at their determined positions according to their selected matching pattern. The transmitting location is estimated in the positioning stage, which may lead to unrouteable layout if inadequate area was designated, or area loss if the routing location is overestimated. Likewise, guard ring positioning is done after establishing the minimal area between the routes, blocks, and bulk ties to pass DRCs and LVS.

Inputs:

- Innovation parameters
- A list of all gadgets' coordinates and names

Results:

- All tools positioned and matched to be prepared for routing.

Routing Blocks (4.5.2.2)

Vertical Routing and Via Positioning

Inputs:

- Placed tools
- Technology PDK and DRCs

Results:

- All S/D terminals of the same gadget are connected and encompassed the predefined coordinates of the horizontal rails using network routing strategy.
- All vertical entrance paths and connections are done. Additionally, vias are added on all terminals and at the junction points of vertical and horizontal paths if any.

  Horizontal Routing

  Inputs:

  - Positioned gadgets
  - Required path length for current capacity
  - Technology PDK and DRCs

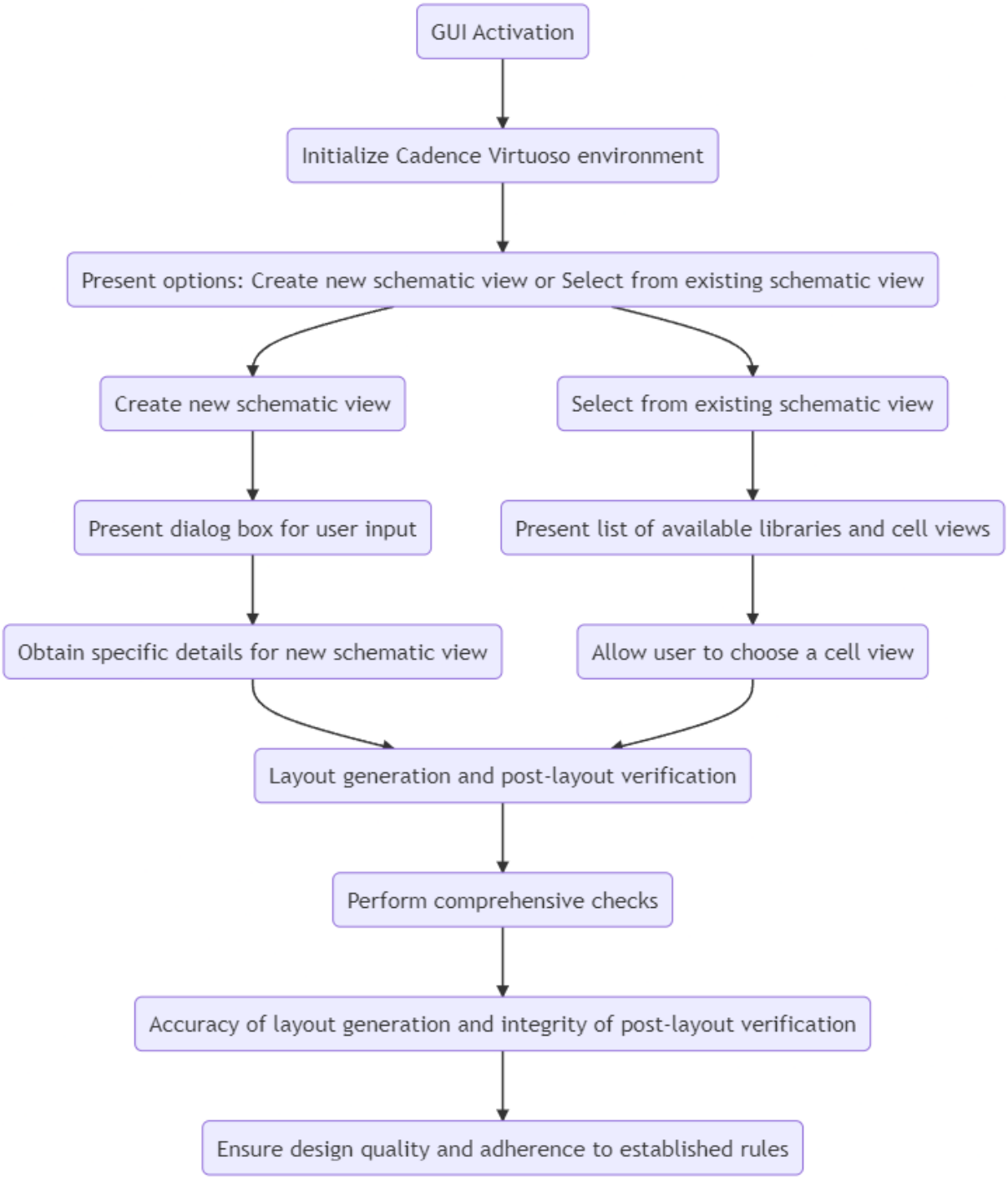


Figure 4.1: Flow chart of system actions and decisions.

Results:

- The horizontal routes of S/D/G terminals, with the appropriate determined width, are positioned at the minimum distance possible to pass DRC.
- Protection of input routes is done if needed, for example, at the differential pair block.

Gate Routing

According to each type of gadgets (current mirror, differential pair, or load), the gate connections will be executed differently. For example, at current mirror devices, the routing of diode-connected devices "gate shorted with drain" needs to be done. However, in differential pair devices, each device should have its own gate routing as they have different input signals.

Inputs:

- Positioned gadgets
- Required path length for current capacity
- Technology PDK and DRCs

Results:

- All gates are routed (horizontally or vertically) according to the configuration of the devices.

Dummies Routing

To have fewer parasitics, mos-cap dummies are used, where all terminals (S/D/G) are shorted and connected with the same point according to the configuration of each device.

Inputs:

- Positioned gadgets and their paths
- Technology PDK and DRCs

Results:

- All dummies are connected as a mos-cap according to the configuration of the devices.

5.1.3)]N/P Well Extension (4.

5.1.3)

According to the type of devices and the number of rows in each block, the extension of Nwell/P-well is performed to pass the DRC and LVS.

Inputs:

- Positioned devices and their paths
- Technology PDK and DRCs

Results:

- All blocks with multiple rows are connected to the same N/P-well according to the devices' type.

Global Routing (4.5.1.4)

Routing all blocks together is a challenging task that requires considering various considerations such as space, detailed routing of blocks, placement of VDD/Tie high and GND/Tie low rails, and the locations of blocks and pins to communicate with other devices.

Inputs:

- Placed tools and their routes
- Technology PDK and DRCs

Results:

- Global routing is achieved through three main stages: block routing, input/output routes and pins, and VDD/Tie high and GND/Tie low rails and routing.

Pattern Recognition and Generation (4.5.1.5)

The first module of pattern recognition and generation can be divided into subsystems. The parsing design is used to convert the input netlist into a standardized format. The second version of pattern recognition utilizes the concept of connectivity terminals. The third version of layout estimation incorporates all elements in the floorplan, including devices, routing, and DRC areas. The fourth version of pattern generation focuses on generating the pattern according to the type of block. The final version involves preparing the result in a fixed-layout text file for multiplier placements and schematic modification, including the addition of dummies and updating device properties.

Chapter 5

Project Design

5.1 Description of Each Subsystem

5.1.1 Graphical User Interface

GUI Logic

When the graphical user interface (GUI) is activated within a specified directory, it first initializes the Cadence Virtuoso environment, a critical precursor to the operational functionality of the platform. Following this initiation, the GUI presents a window, prompting the user to make a selection between two options: creating a new schematic view or selecting from an existing schematic view. These options offer flexibility to the user, accommodating both the inception of new design projects and the continuation of existing ones.

When the graphical user interface (GUI) is activated within a specified directory, it first initializes the Cadence Virtuoso environment, a critical precursor to the operational functionality of the platform. Following this initiation, the GUI presents a window, prompting the user to make a selection between two options: creating a new schematic view or selecting from an existing schematic view. These options offer flexibility to the user, accommodating both the inception of new design projects and the continuation of existing ones.

Should the user opt to create a new schematic view, the GUI responds by presenting a dialog box. This interactive element invites the user to provide specific details, such as the library name and the cell view name, necessary for the creation of a new schematic view. This user-driven input mechanism ensures that the new schematic is tailored to the user's precise specifications, encouraging a more efficient and personalized design process.

Alternatively, if the user elects to choose from existing schematics, the GUI facilitates this process by presenting a comprehensive list of available libraries, each containing multiple cell views. The user is then able to peruse these options, and make an informed selection of the cell view that best suits their current needs.

Once a cell view has been selected, whether newly created or pre-existing, the GUI undertakes the task of layout generation and post-layout verification. This encompasses comprehensive checks including, but not limited to, Design Rule Checks (DRC) and Layout Versus Schematic (LVS) checks.

These advanced features, which are readily accessible at a single mouse click, ensure the accuracy of the layout generation and the integrity of the post-layout verification process. This GUI logic ensures that users can proceed with their design tasks confident in the knowledge that the platform is taking appropriate steps to maintain design quality and uphold adherence to established design rules.

Initialization of cadence directory

Upon activation, the Graphical User Interface (GUI) executes a critical preliminary check to determine whether the Cadence initialization files exist within the operating directory. In the absence of these files, the GUI initiates the creation of three primary files: `'cds.lib'`, `'.cdsinit'`, and `'.cdsenv'`, each of which serves a distinct purpose within the Cadence Virtuoso environment.

The function `cdsenv()` carries the responsibility of creating or overwriting a `.cdsenv` file in the current working directory. This file is instrumental in the configuration of specific settings for Cadence Virtuoso. To facilitate this, the function writes the string `"ddserv.ciw promptOnExit boolean nil\n"` into the `.cdsenv` file, effectively disabling the prompt that typically requests exit confirmation within Cadence Virtuoso. In cases where a `.cdsenv` file already exists, its existing content is overwritten; if absent, a new file is generated. If an IOError is encountered during the write operation, the function raises an exception.

The second function, `cdslib()`, focuses on the creation or overwriting of a `cds.lib` file in the current directory. This file is crucial in defining libraries that are utilized within Cadence Virtuoso, notably the TSMC 65 nm library. To achieve this, the function compiles the content of the `cds.lib` file by concatenating several text lines, incorporating the path to the TSMC 65 nm library. The behavior of this function mirrors `cdsenv()`, overwriting any pre-existing `cds.lib` file, or establishing a new file if one does not already exist.

The third function, `cdsinit()`, orchestrates the creation of a `.cdsinit` file in the current working directory. This file is paramount for customizing the Cadence Virtuoso environment as it serves as a startup file, loaded each time the environment is initiated. The content of the `.cdsinit` file comprises a significant amount of SKILL code, tasked with setting various environment variables, defining bind keys, loading libraries, and performing other configurations. Like its counterparts, if a `.cdsinit` file is already present, it is overwritten; if not, a new file is created. The function also raises an exception if an IOError occurs during file creation.

The function `init_dir()` is essentially the primary point of entry for the script. This function sequentially invokes `cdsinit()`, `cdslib()`, and `cdsenv()` to establish the necessary configuration files for the initiation of Cadence Virtuoso in a new directory.

Design rules check

The function `drc_check_tsmc65nm(cadence_directory, drc_rules_path, library_name, top_Cell)` is designed to execute a Design Rule Check (DRC) on a given design specified by `top_Cell` from a library specified by `library_name` using the 65nm TSMC technology. This function employs the Calibre tool from Mentor Graphics. The procedure executed by this function includes:

1. **Exporting Layout to GDS:** The function first exports the layout of the top cell in the given library to a GDSII (GDS) format file. GDSII is a database file format which is a standard for data exchange of IC layout artwork. The function `export_layout_to_gds` performs this task. It navigates to the Cadence directory where the library is defined and executes the `strmout` command to output the GDS file.
2. **Creating DRC Directory:** Next, a new directory named "drc" is created in the current working directory. This directory will be used to store the results of the DRC.
3. **Editing DRC Rule File:** The function `edit_calibre_drc` is then used to edit the DRC rule file by updating the paths of the layout file (GDS file), the top cell, and the DRC results database. The path of the edited DRC rule file is returned.
4. **Running DRC:** The function `run_calibre_drc` runs the DRC using the Calibre tool. It takes the path of the edited DRC rule file as an argument. The DRC rule file contains all the design rules to be checked on the IC layout. Any `stdout` or `stderr` messages are printed.
5. **Running Calibre RVE:** Finally, the function `run_calibre_drc_rve` is used to launch the Calibre RVE tool. Calibre RVE is used to review the results of the DRC. The DRC results are stored in a database file. The path to this database file is passed to the `run_calibre_drc_rve` function.

After running this function, the user should be able to see the DRC results in the "drc" directory that was created. If any design rule violations were detected, they will be indicated in the results.

Layout vs Schematic

The function `lvs_check_tsmc65nm(cadence_directory, lvs_rules_path, library_name, cell_view_name)` performs a Layout Versus Schematic (LVS) check on the specified design. It compares the physical layout and the schematic of the design to verify that they match. Here's a breakdown of what it does:

1. **Initializing the SI Environment File:** This function starts by generating an SI environment file in the specified Cadence directory. The function

`init_si_env_file` performs this task. It writes certain variables and their values into the SI environment file, such as the library name, cell view name, and various simulation settings.

2. **Converting the Schematic to SPICE:** The function `schematic_to_spice` is then used to convert the schematic of the design to a SPICE netlist. The SPICE netlist represents the schematic in a format that can be read by circuit simulation tools. The conversion is done using the `si` command-line tool from Cadence.
3. **Editing the LVS Rule File:** After generating the SPICE netlist, the function `edit_calibre_lvs` is called to edit the LVS rule file. The paths of the layout file (GDS file) and the SPICE netlist, as well as the name of the top cell, are updated in this file. The path of the edited LVS rule file is returned.
4. **Running LVS:** The function `run_calibre_lvs` is then used to run the LVS using the Calibre tool. It takes the path of the edited LVS rule file as an argument. The LVS rule file contains the instructions and rules for the LVS check. Any `stdout` or `stderr` messages are printed.
5. **Launching the Calibre RVE:** Finally, the function `run_calibre_lvs_rve` is used to launch the Calibre RVE tool. Calibre RVE is a review tool that provides an interface for viewing the results of the LVS.

Once this function has completed, the LVS check results can be reviewed in the Calibre RVE tool. Any discrepancies between the layout and the schematic will be flagged.

5.1.2 SKILL modifications

P-Load Block Description

The P-Load block in the OTA circuit comprises two PMOS devices denoted as M8 and M9 as shown in Figure 1. The gates of these two transistors are interconnected. The drain of M8 is connected with the drain of the N_load transistor M10 while the drain of M9 is associated with the drain of the N_load transistor M11 and the output node. As there are no common source or drain connections between these devices in the P_load sub-block, the matching process becomes more challenging.

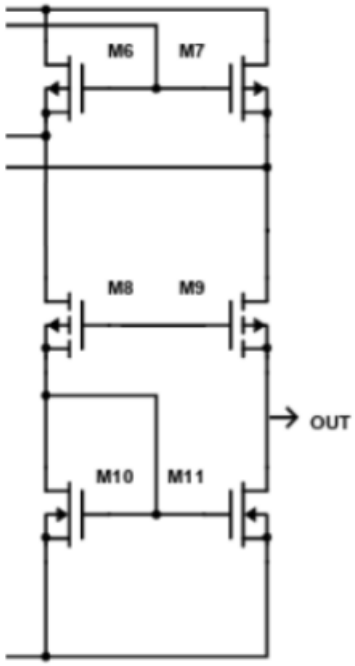


Figure 5.1: Schematic of Loads and P_current mirror

A viable solution is to utilize the "matching in space" technique. This method is based on placing devices as close to each other as possible, ensuring that the matching space distance is the same between all devices to minimize process variations. Despite its promising potential, this method introduces parasitic capacitance and resistance, which can lead to a degree of mismatching between devices.

Another solution entails matching these devices with dummy devices, implying the insertion of dummy devices between those that are neither common source nor common drain. This configuration forms a continuous bridge between devices, which helps reduce the process variations previously mentioned. However, the dummy connections must ensure that the dummy device remains off to prevent short circuits in the main block routing.

The matching pattern chosen for this block to accommodate the routing of devices

along with the dummy devices is interdigitation. This pattern involves inserting a dummy device between each consecutive device.

Pattern and Names Generation

Two Python functions were developed to apply the matching pattern described above. The first function determines the positioning pattern of each device and dummy, while the second generates names to be parsed later by the skill code for further use in the layout.

Matching Pattern

The following list represents the matching pattern: [D_P_load81 , M08.1, D_P_load9, M09.1, D_P_load8, M08.2, D_P_load9, M09.2, D_P_load91].

Name Generation The positions of the devices correspond to the coordinates of the bottom-left corner of the gate of each specific device. To get the full length of the device, one must add the bounding box, the gate length, and the source/drain terminal length, as illustrated in Figure 2.

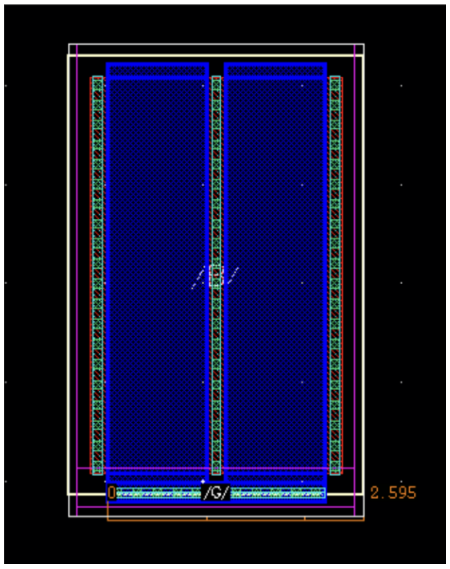


Figure 5.2: Single device representation

A starting point representing the beginning of the block is required to get the positions of all devices. This starting point is the position of the end of the last device in the adjacent block (either vertically or horizontally), plus the minimum spacing multiplied by a safety factor. This methodology helps determine a starting point for the X and Y positions.

To calculate the position of the next device in the X direction, the ending position of the device is computed and added to the gate length of the dummy since source

and drains are shared with adjacent devices. For the Y positions, the position of the device ending in the vertical direction is calculated and added to the minimum spacing multiplied by a safety factor, along with the bounding box of the device.

The X position of the dummy devices placed between the devices is determined as the position of the previous device (M8/M9) plus the gate length and source/drain length of the device. For the first dummy at the right, it is inserted separately, and its position is the position of the first device subtracted from the gate length and source/ drain length of the device itself.

The output of this operation is four lists: one for the X positions of devices and dummies combined, one for the Y positions, one for the X positions of devices, and one for dummy devices. All lists have the same length corresponding to a single row of multipliers.

First, the names of the device list and the dummy device list are generated separately. For M8 and M9, the naming follows the pattern "device name.index" (M8.1, M9.1, M8.2, M9.2, etc.), while the dummy devices are named in the form of D_P_load(index) (D_P_load1, D_P_load2, etc.).

To determine the required matching pattern, the device names need to be combined in a single list alternately. This task is accomplished by the following algorithm:

Algorithm 1 Device Name Combination

1: Determine the dimensions for a new 2D list:
   - Get the size of 'device_names'
   - Get the size of 'x_list_p_load_with_dummy' (positions of devices and dummy devices)
2: Create a new 2D list 'device_names_with_dummy', initializing all elements to 0
3: **for** each entry in 'device_names' **do**
4: **for** each entry in 'x_list_p_load_with_dummy' **do**
5: **if** index is even **then**
6: Assign the corresponding entry from 'dummy_names' to 'device_names_with_dummy'
7: **else**
8: Assign the corresponding entry from 'device_names' to 'device_names_with_dummy'
9: **end if**
10: **end for**
11: **end for**

Input File and Schematic Modification

There are four main categories of dummy devices in the P_load block, each differing from the others in terminal connections. To simplify routing and schematic modifica-

tion, each category was assigned a unique naming method, allowing for differentiation between these devices and their connections.

For each row, the first dummy on the left is named D_P_load81(index), while the last dummy on the right is named D_P_load91(index). The dummy before the M9 device is named D_P_load9(index), while that before M8 is named D_P_load8(index).

The names of these dummy devices were updated according to the criteria explained above using the following logic when writing to the input file:

Algorithm 2 Dummy Devices Name Update

```
for each device name do
  for each element in 'x_list_p_load_with_dummy' do
    if index is 0 and the name contains "D_p_load" then
      Replace "D_p_load" with "D_p_load.81" in the name
      Write the updated name, corresponding element from 'x_list_p_load_with_dummy', and corresponding y-value to the file with spaces in between
    else if index is the last in the list and the name contains "D_p_load" then
      Replace "D_p_load" with "D_p_load.91" in the name
      Write the updated name, corresponding element from 'x_list_p_load_with_dummy', and corresponding y-value to the file
    else if name contains "D_p_load" then
      if next name contains "M08" then
        Replace "D_p_load" with "D_p_load8" in the name
        Write the updated name, corresponding element from 'x_list_p_load_with_dummy', and corresponding y-value to the file with spaces in between
      else if next name contains "M09" then
        Replace "D_p_load" with "D_p_load9" in the name
        Write the updated name, corresponding element from 'x_list_p_load_with_dummy', and corresponding y-value to the file with spaces in between
      end if
    else
      Write the current name, corresponding element from 'x_list_p_load_with_dummy', and corresponding y-value to the file with spaces in between
    end if
  end for
end for
```

In the schematic modification, new dummy devices, along with their information including net connections, technology, transistor type, length, width, number of fingers, and number of multipliers, are written in a file named "Schematic Modification".

```
D_p_load8<1:14> <*14>M7_net <*14>VDD <*14>M6_net <*14>M6_net tsmcN65 pch 1.0u 4.0u 4.0u 1 1 1
D_p_load9<1:16> <*16>M6_net <*16>VDD <*16>M7_net <*16>M7_net tsmcN65 pch 1.0u 4.0u 4.0u 1 1 1
D_p_load81<1:2> <*2>M6_net <*2>VDD <*2>M6_net <*2>M6_net tsmcN65 pch 1.0u 4.0u 4.0u 1 1 1
D_p_load91<1:2> <*2>M7_net <*2>VDD <*2>M7_net <*2>M7_net tsmcN65 pch 1.0u 4.0u 4.0u 1 1 1
```

Figure 5.3: Schematic Modification for P_Load

The input file and schematic modifications are two output files extracted from the Python program and are given as inputs to the skill code. The first part to be executed in the skill code will be the schematic modification, which adds the new dummy devices of the P_Load to the schematic file with proper connections to pass the LVS test.

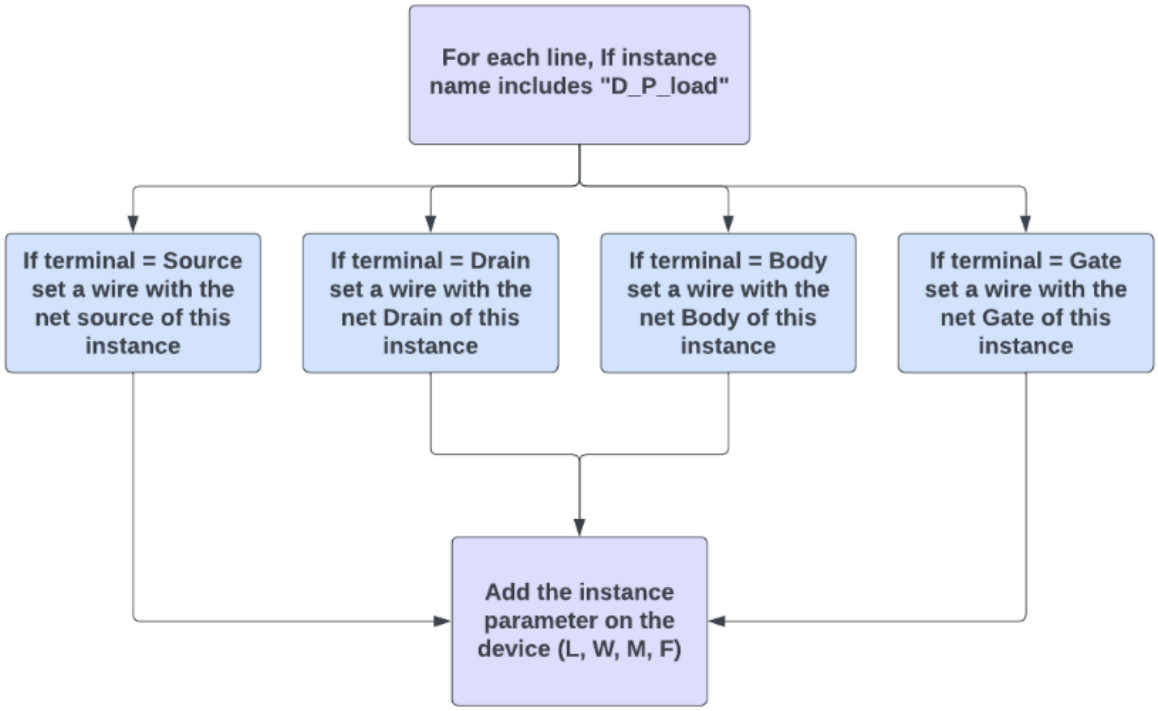


Figure 5.4: Schematic Modification Logic

Routing

Drain Connection

Each device's drain is linked to a unique horizontal rail, as illustrated in Figure 5.6, because they each connect to separate blocks. Consequently, a pair of horizontal rails are established at the block's upper section, aiding the subsequent stage of global routing.

Figure 5.5: Schematic Modification Output

The P-load drains are guided using appropriate metal strata, with their width determined based on the user's input current and the metal's current capacity, which varies according to the utilized technology.

Devices constituting the P-load are initially pinpointed and stored in variables, allowing for the individual routing of each device. Horizontal pathways, created beneath the P-load block, employ a suitable metal layer. Vertical pathways stem from the drain terminals of each device, connecting to one of the horizontal paths via vias. The count of vias bridging the vertical and horizontal routes is contingent on the width of the horizontal pathways, calculated to span the width of the horizontal route with two columns of cuts. If stacking is utilized, the second and fourth metal layers are employed for vertical routing, while the third and fifth metal layers are used for horizontal routing. Stacked vias are used in lieu of standard vias. Labels are appended as required for future stages in global routing.

As described earlier, there are 4 categories of dummy devices, where the only difference between these categories is the connections of the nets.

The first dummy in the row is the dummy on the very left of the row , this dummy device, regardless of which row it exists, has the three terminals (source, drain, gate) connected to the first signal rail. This dummy is named as $D\_P\_Load81(index)$.

The second dummy is the one having the M8 device before it and M9 device after it, so it is sharing a drain with the source of M9 connected to the second signal rail and a source with the source of M8 connected to the first signal rail. The gate is connected to the source of the dummy in this case, which is the first signal rail.

The third dummy is the one having the M9 device before it and M8 device after it, so it is sharing a drain with the source of M8 connected to the second signal rail and a

source with the source of M9 connected to the first signal rail. The gate is connected to the source of the dummy in this case, which is the second signal rail.

The fourth dummy is the dummy on the very right of the row. This dummy device, regardless of which row it exists, has the three terminals (source, drain, gate) connected to the second signal rail. This dummy is named as $D\_P\_Load91(index)$.

Note that the gates of all dummies are connected to their own sources, which is the source of the previous device as well. This ensures that V_{GS} of the dummy device is always zero, making sure the dummy is off.

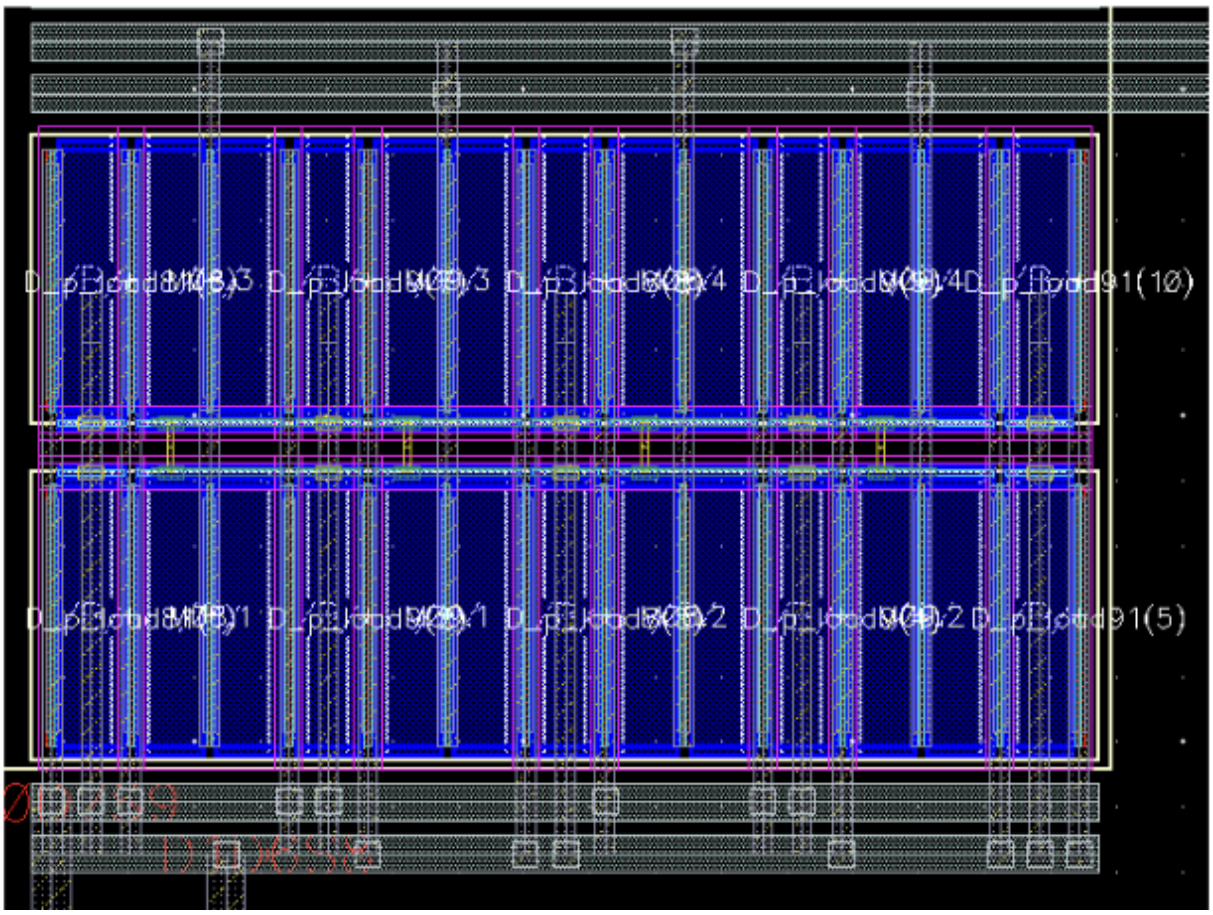


Figure 5.6: P_Load layout

In order to output the connections seen in Figure 5.6, the skill code logic flow is as follows for the P_Load Gate part.

For the drain and source connections

of the dummy devices, they all have common sources and drains with the sources of the adjacent devices, so no new connections were established in this section. However, the source of the first dummy in the row and the drain of the last dummy in the row are not connected since there exist no devices to their other side.

Figure 5.7: Dummy logic in Skill

In the drain section of the skill code, a check condition is added: if the instance name is *D_P_Load*81, which means that this is the leftmost dummy in any row, a vertical path is created connecting all vertical devices and a via connecting this path with the first signal rail.

In the source section of the skill code, a check condition is added: if the instance name is *D_P_Load*91, which means that this is the rightmost dummy in any row, a vertical path is created connecting all vertical devices and a via connecting this path with the second signal rail.

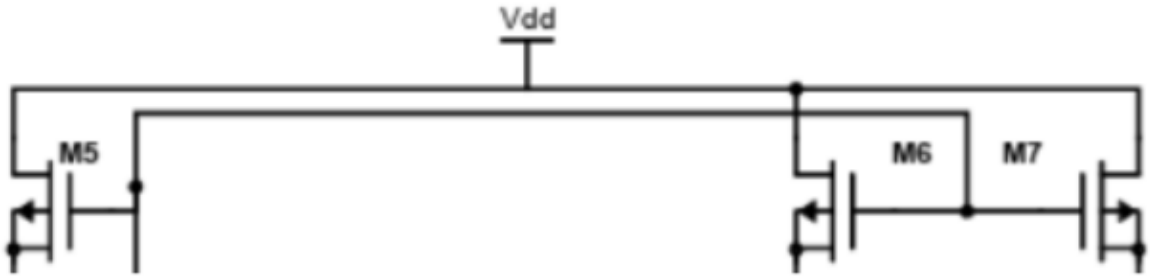
P and N Current Mirrors

For the P_Current Mirror, it consists of three PMOS transistors (M5, M6, M7) where the gates of the three transistors are connected to the drain of the M5 device, and the sources are pulled up to V_{DD} while drains of each device are connected to a separate signal according to the OTA main schematic. While for the N_Current mirror, it consists of three NMOS transistors (M2, M3, M4) where the gates of the three devices are connected together along with the drain of M2 that is connected to the IBIAS signal, and the sources of all devices are pulled down to ground. Each device's drain is connected to a separate signal.

The matching pattern associated with these blocks is common centroid, as it acts as the most reliable method for matching the three devices of the current mirror. In previous designs, if the input of any of the device's multipliers is an odd number, the python algorithm would change the width of the unit cell in order to adjust the multiplier to be an even number. Making sure that the product of (multipliers $\times$ fingers $\times$ width) is constant, the number of multipliers could change accordingly to be an even number. An even number of multipliers is required for easier and better matching. However, designers may find it very restrictive to change the width of the

devices in some applications. For this reason, an option was encountered where if the number of multipliers is odd, instead of changing width, the new algorithm would add a number of dummies as to round up the number of multipliers to the next 2^n. Extra dummy devices are added to enhance matching and remove the mentioned restriction. Nevertheless, the pattern by which these extra dummy devices are added is very crucial; it must serve the matching purpose without altering block operation. Not to mention that these dummy devices would be required to be off. Figure 5.8 shows the schematic of the P_Current mirror block, while Figure 5.9 shows the schematic of the N_Current mirror.

Figure 5.8: Schematic of P_Current Mirror



Pattern and names generation

The detailed explanation in this section is performed on the P_Mirror, however, the exact algorithms go applied to N_Mirror. For these two blocks, the

user is allowed to choose the option of whether they prefer to add dummy devices to the odd number of multipliers or just change the width and perform the previous algorithm. The matching pattern for the P_Current mirror for a single row is [D_P_mirror, M7.1, M7.2, M6.1, M5.1, M5.2, M6.2, M7.3, M7.4, D_P_mirror]. Let's assume that the M7 device has an odd number of multipliers; therefore, the matching pattern would instead be [D_P_mirror, M7.1, D_P_mirrorM7(1), M6.1, M5.1, M5.2, M6.2, M7.3, M7.4, D_P_mirror], where $D\_P\_mirrorM7(1)$ is the added dummy to compensate for the missing multiplier in the row. Accordingly, for the N_Current mirror for the same case, the matching pattern would be [D_N_mirror, M4.1, D_N_mirrorM2(1), M3.1, M2.1, M2.2, M3.2, M4.3, M4.4, D_N_mirror].

The following pseudo code shows how the coordinates of the devices' positions are generated along with the dummy devices' positions.

The previous flow chart explains how the positions of devices are calculated and handled if there exist extra dummies that need to be inserted. It is worth noting that until this step, the dummy is treated as a normal device in terms of positioning; it is not yet differentiated from the other devices. The variable $center\_x3$ is the starting point from which the very first device in the row would be located. This

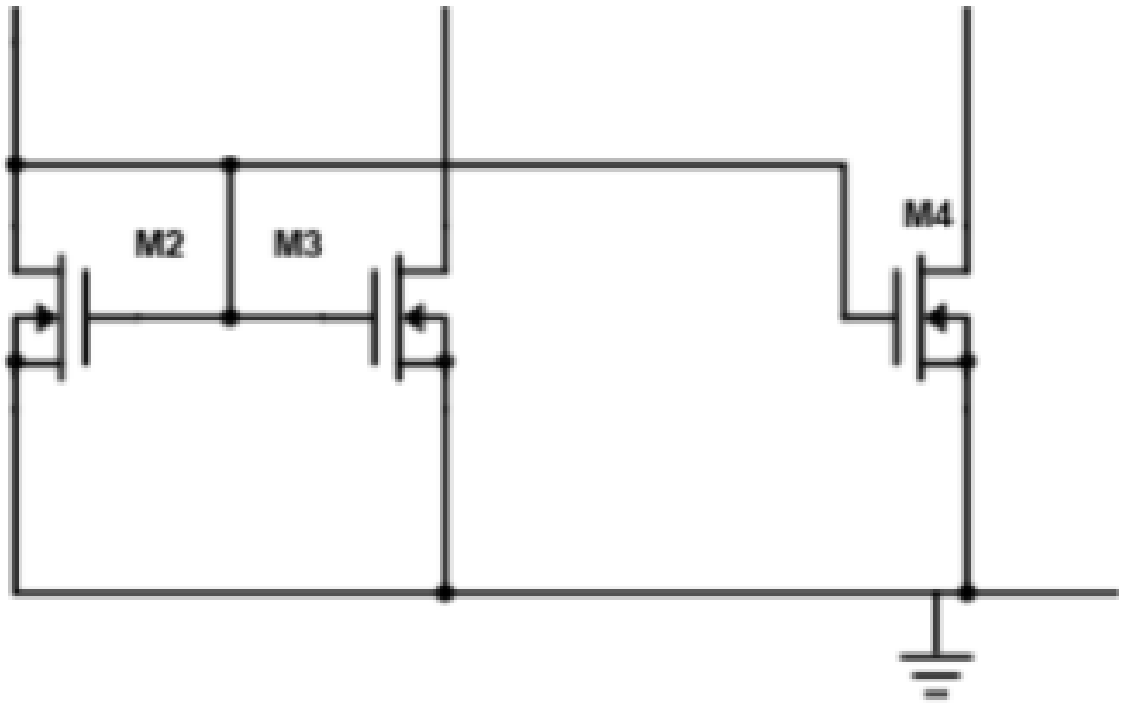


Figure 5.9: Schematic of N_Current Mirror

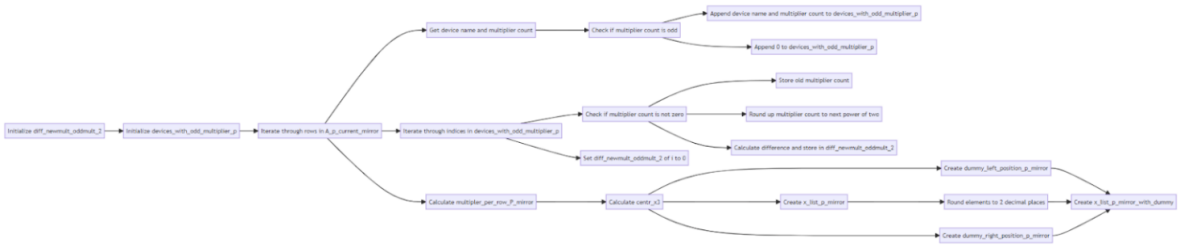
method is used for the X positions. On the other hand, for the Y positions, the following pseudo code explains the process.

1. Initialize an empty list $y\_list\_p\_mirror$.
2. Iterate through the range of $n\_row\_p\_mirror\_p\_load$.
   (a) Calculate the value of y for each iteration using the formula: $i \times (min\_device\_spacing \times 1.5 + b\_box\_oxide \times 2 + x\_6\_2) + y\_2$.
   (b) Append the calculated y value to the $y\_list\_p\_mirror$ list.
3. The resulting $y\_list\_p\_mirror$ will contain the calculated y values for each iteration.

Where $n\_row\_p\_mirror\_p\_load$ is the number of rows of the P_mirror block, x_6_2 is the vertical dimension of the device itself, and Y_2 is the starting position of the first block.

In order to define the required pattern, the generated coordinates should be assigned to correct names that correspond to each device. In this section, the dummy devices will be differentiated from the other devices. The dummy devices of each device are named after it, as this naming method will be essential for schematic modification for connecting nets as well as for routing.

Figure 5.10: Flow chart of position generation



The following flow chart describes the name degeneration of both dummies and devices.

Let's assume M7 has a number of multipliers equal to 5, therefore, 3 dummies need to be added.

Devices_names	Dummies_names	M7_left	M7_right
M7.1	D_P_LoadM7(1)	M7.1	M7.2
M7.3	D_P_LoadM7(3)	M7.3	M7.4

Table 5.1: Outputs of each list of the algorithm M7

sectionInput file and Schematic modification

5.1.3 Input file

In order to obtain the pattern shown in figure 11, the skill code should run taking proper input and schematic modification files. Therefore, the dummies added should be written to the input file. The following pseudo code shows how this operation takes place.

```
1. Iterate over each row in arr\_pattern\_p\_mirror using the outer loop varia
2. Within each row, iterate over each element using the inner loop variable j.
3. Check if the substring of arr\_pattern\_p\_mirror[i][j] before the first oc
4. If the condition is true, perform the following sub-steps:
    4.1 Write the concatenated string of arr\_pattern\_p\_mirror[i][j], x\_lis
    4.2 Repeat steps 2-4 for all elements in each row of arr\_pattern\_p\_mirr
5. Repeat steps 1-5 for all rows in arr\_pattern\_p\_mirror.
```

Note that the list arr_pattern_p_mirror is the final concatenated list just obtained, this example is on M7, same goes for all other devices of both blocks.

Schematic modification The extra dummies should be inserted in the schematic modification file.

Dummy connection For each block, there exists three categories of dummies, one for each device. It is important to realize that the three categories have different naming and that it's essential to assign each category with the correct nets connections according to the device it is associated with. The dummy device would be sharing sources and with the sources of the adjacent devices so no modification done in this part. Note that this dummy would have 2 fingers like the device itself. The dummy is treated as a device and its gate is connected to the gates of the rest of the devices. Since we need the dummy to be off for proper operation the drain must be connected to VDD to incur a VDS of zero and an off state. The VDD is a really strong and stable single, therefore, it is acceptable to have a value for the VGS, the device will never be on due to the high stability of the VDD that will ensure zero VDS.

5.1.4 Project Testing and Evaluation

Finally, the code was tested on several cases in different technologies and all test results were reviewed and evaluated; all aims of the code such as constraints, DRC LVS, non-formal constrains, layout quality and consistency of the layout generated structures were achieved successfully. These requirements are met by using good matching techniques and shielding that enhance immunity to noise, accurate coordinates and pattern generation, etc. For reliability, metal widening was done to overcome electromigration, double cut vias was used in all the structures even the big ones to increase yield. Furthermore, the LUP issue was solved by adding a guard ring strips in the layout to isolate the unwanted low impedance path between supply and ground. In addition, bulk contacts were created at top and bottom of the chip to achieve minimum area and to enhance reliability. The runtime of the skill code is less than 1.5 min.

First test case

Block	**Length (um)**	**Width(um)**	**Fingers**	**Multipliers**
M5	1	4	2	16
M6	1	4	2	15
M7	1	4	2	9
M8	1	4	2	16
M9	1	4	2	16
M2	2	2	2	32
M3	2	2	2	16
M4	2	2	2	16
M10	2	2.5	2	8
M11	2	2.5	2	8
M0	2	2.5	2	32
M1	2	2.5	2	32

Table 5.2: first test case

Second test case

Block	Length (um)	Width(um)	Fingers	Multipliers
M5	1	5	2	8
M6	1	5	2	8
M7	1	5	2	15
M8	1	4.4	2	16
M9	1	4.4	2	16
M2	2	2	2	32
M3	2	2	2	16
M4	2	2	2	17
M10	2	2.5	2	8
M11	2	2.5	2	8
M0	2	2.5	2	32
M1	2	2.5	2	32

Table 5.3: Second test case

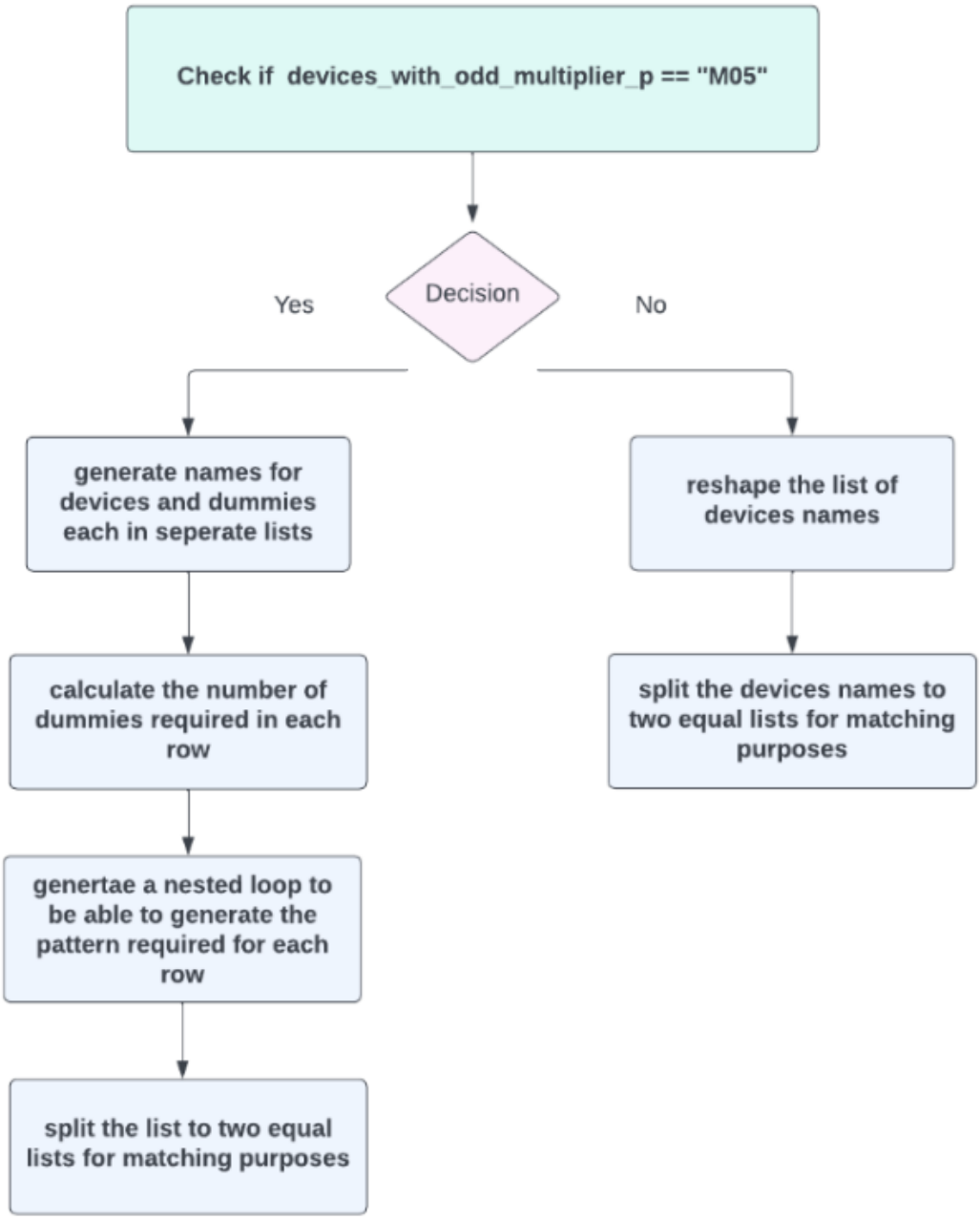


Figure 5.11: Name degeneration flow chart

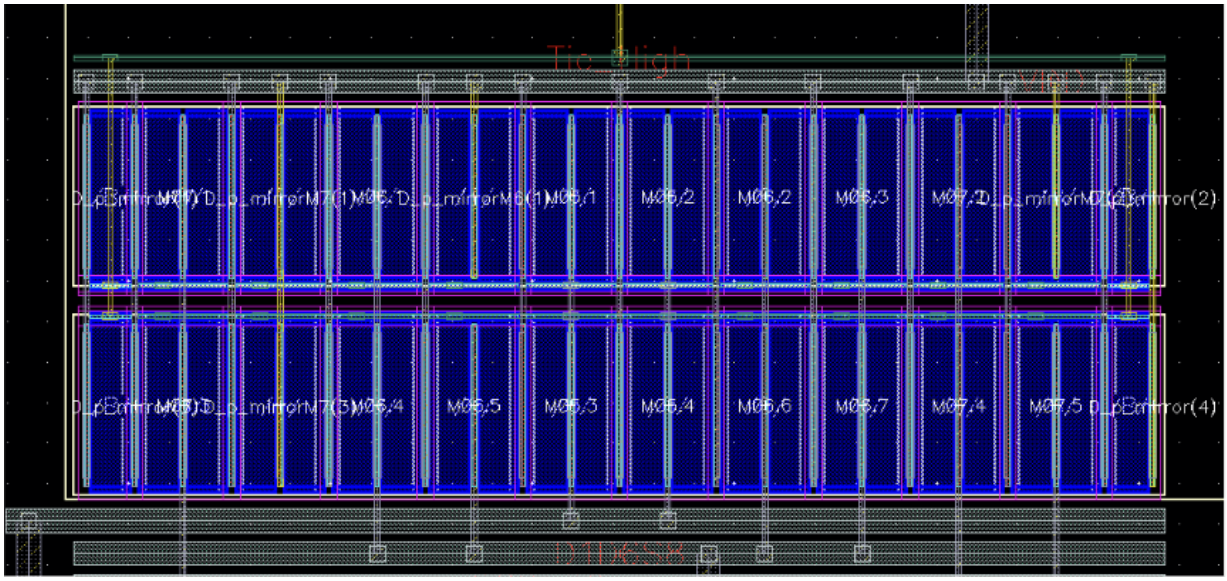


Figure 5.12: Placement of P$_{C}urrentmirror$

```
D_n_mirror<1:8> <*8>GND <*8>GND <*8>tie_Low <*8>GND tsmcN65 nch 2.0u 2.0u 2.0u 1 1 1
D_p_mirror<1:4> <*4>VDD <*4>VDD <*4>tie_High <*4>VDD tsmcN65 pch 1.0u 4.0u 4.0u 1 1 1
D_p_mirrorM6<1:1> <*1>VDD <*1>VDD <*1>PCM_gate <*1>VDD tsmcN65 pch 1.0u 4.0u 4.0u 2 1 2
D_p_mirrorM7<1:3> <*3>VDD <*3>VDD <*3>PCM_gate <*3>VDD tsmcN65 pch 1.0u 4.0u 4.0u 2 1 2
```

Figure 5.13: Schematic modification file of P_mirror

Figure 5.14: Schematic modification in schematic file

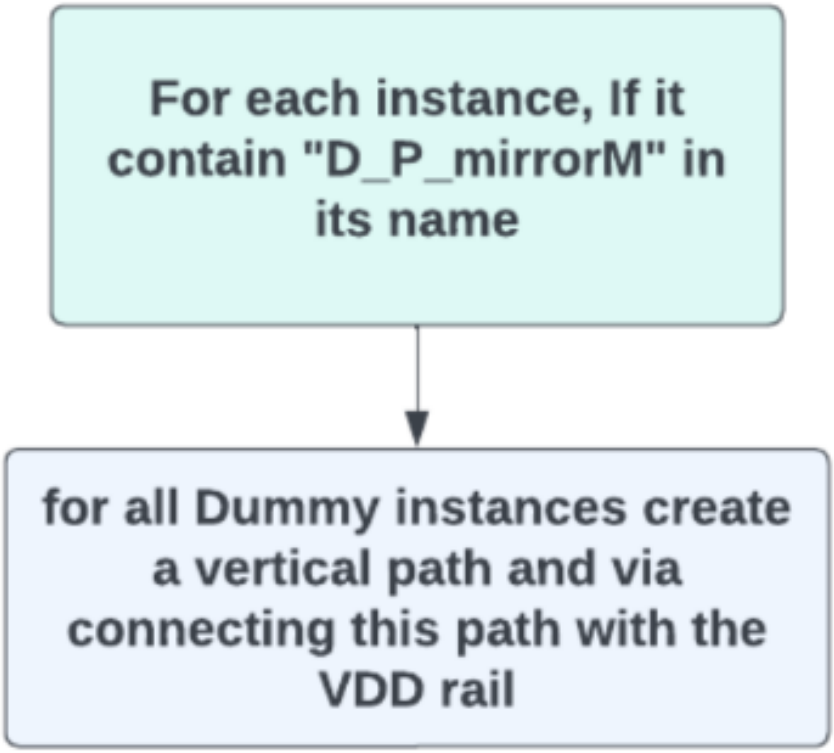


Figure 5.15: Dummy logic in skill

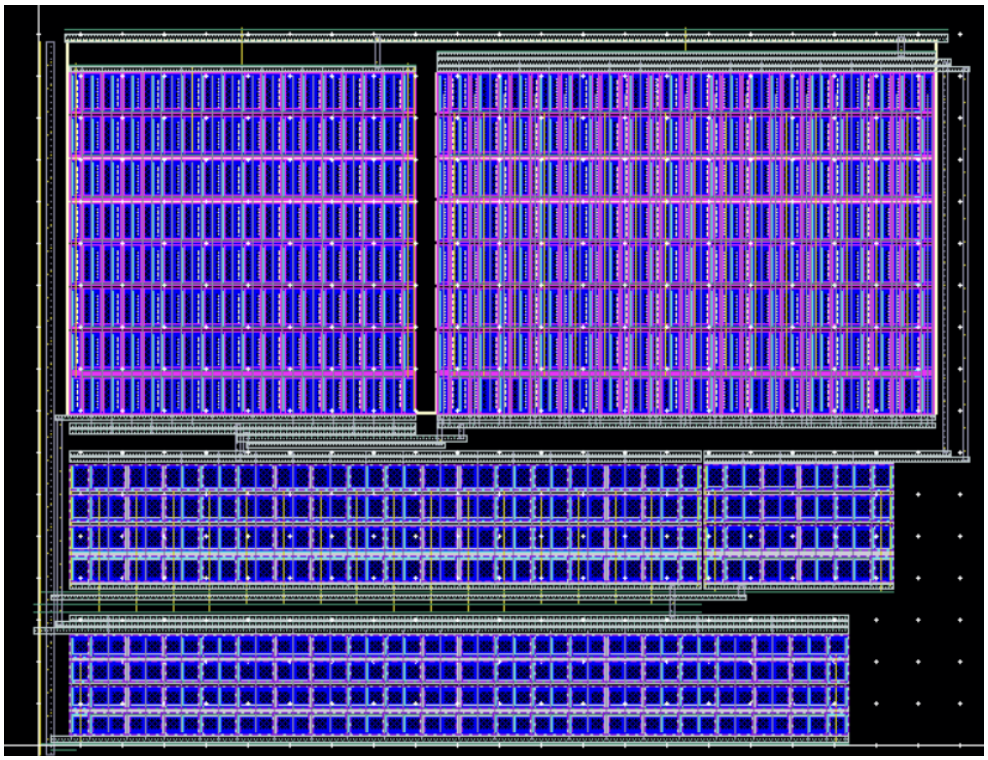


Figure 5.16: Layout

Check / Cell	Results	Flat
Check DOD.R.1	1	1
Check PO.DN.1H	1	1
Check PO.R.8	260	384
Check DPO.R.1	1	1
Check M1.DN.1L	1	1
Check M5.DN.1L	1	1
Check M6.DN.1L	1	1
Check M7.DN.1L	1	1
Check CSR.R.1.NWi	1	1
Check CSR.R.1.PPi	10	10
Check CSR.R.1.NPi	8	8
Check CSR.R.1.COi	1000	1000
Check CSR.R.1.M1i	1000	1000
Check CSR.R.1.M1_real	1000	1000
Check CSR.R.1.M2i	457	457

Figure 5.17: DRC

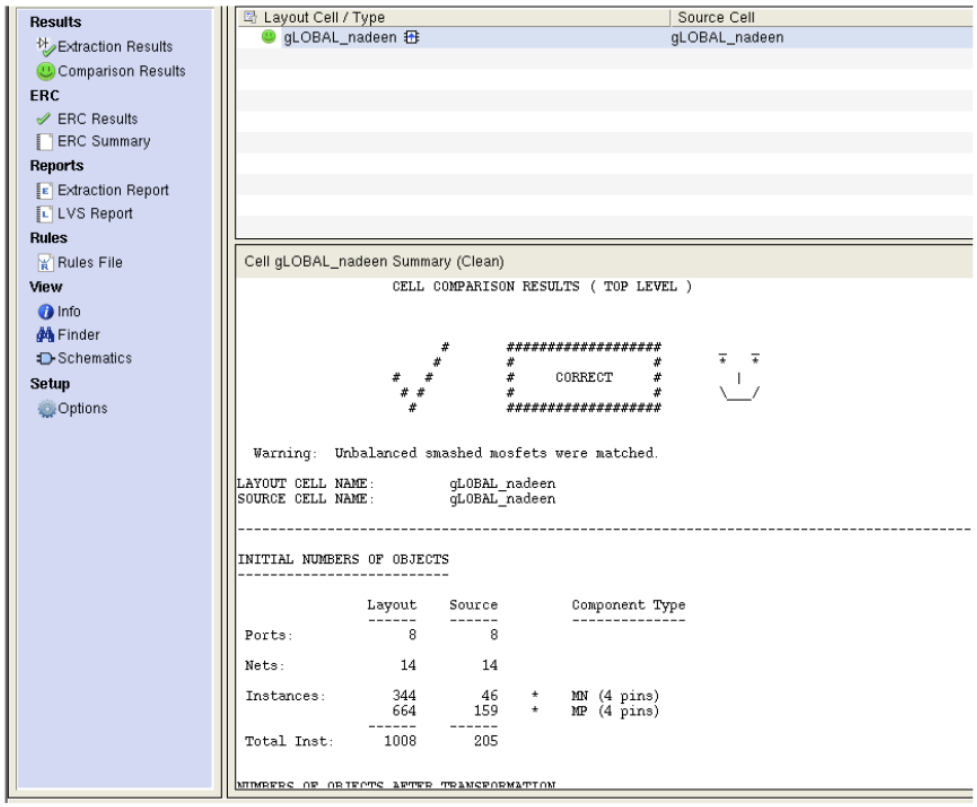


Figure 5.18: LVS

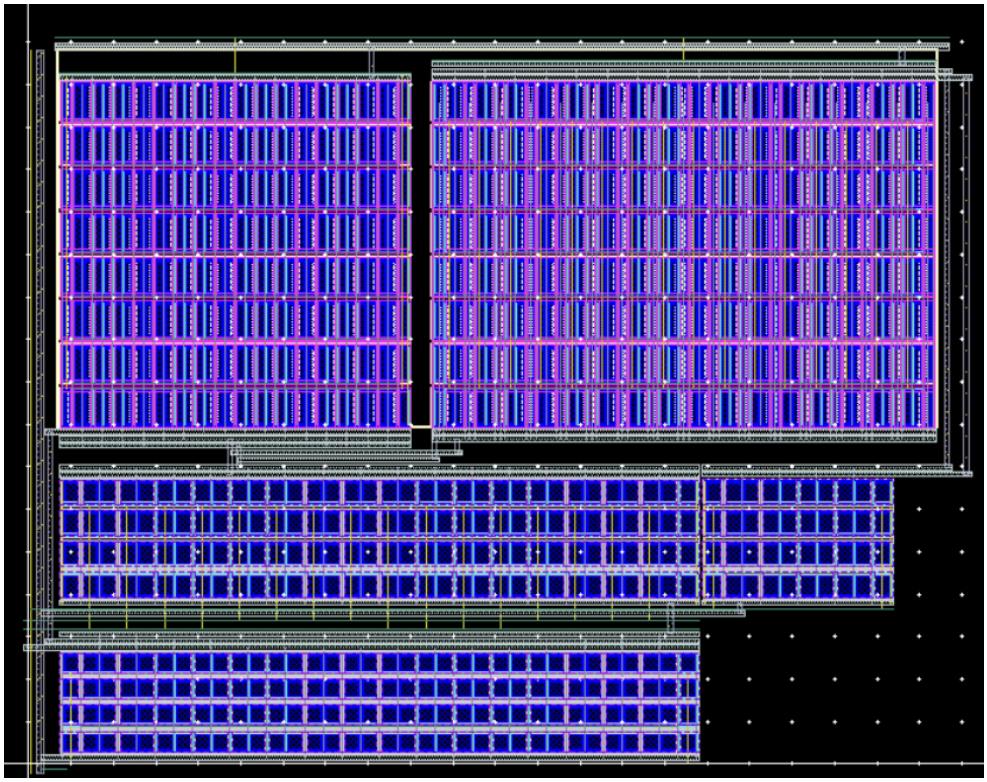


Figure 5.19: Layout

Check / Cell	Results	Flat
Check DOD.R.1	1	1
Check PO.DN.1H	1	1
Check PO.R.8	260	384
Check DPO.R.1	1	1
Check M1.DN.1L	1	1
Check M5.DN.1L	1	1
Check M6.DN.1L	1	1
Check M7.DN.1L	1	1
Check CSR.R.1.NWi	1	1
Check CSR.R.1.PPi	10	10
Check CSR.R.1.NPi	8	8
Check CSR.R.1.COi	1000	1000
Check CSR.R.1.M1i	1000	1000
Check CSR.R.1.M1_real	1000	1000
Check CSR.R.1.M2i	435	435

Figure 5.20: DRC

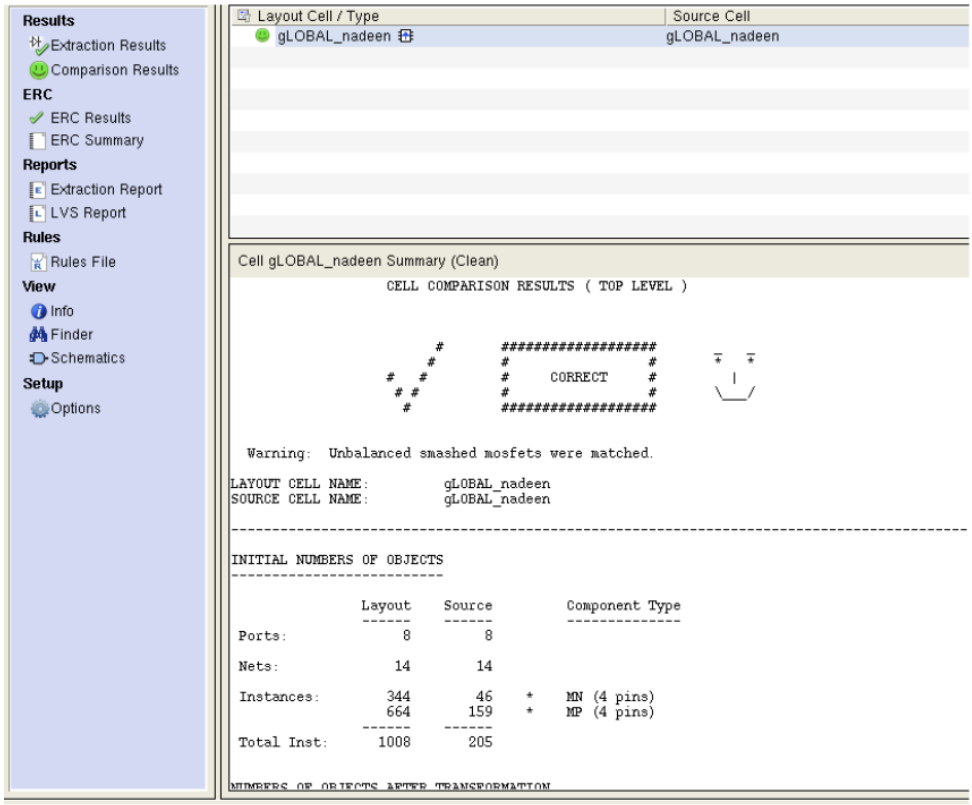


Figure 5.21: LVS

Chapter 6

Cost Analysis

Cost Analysis

Cost

Recurring expenditures account for the majority of ongoing costs. On the other hand, non-recurring expenditures are associated with the ongoing maintenance and functionality enhancement of the code. However, these expenditures are insignificant compared to the cost of manual layout design, as it consumes more time and effort. Furthermore, layout automation technologies may provide time savings, allowing engineers to dedicate more time to optimization rather than manually routing and rectifying Design Rule Check (DRC) and Layout Versus Schematic (LVS) problems.

Another cost factor is the Computer-Aided Design (CAD) software that will be used to test the design and/or build the integration. In our situation, Zewail City provides the Cadence Virtuoso license and student Process Design Kit (PDK).

Environmental Impact

The tool primarily consists of software and does not have a direct impact on the environment. However, as our tool makes it easier for fabrication houses to increase yield and improve the fabrication process, the environment may be negatively affected if the operation of fabrication in industries is not well controlled, leading to an increase in the pollution rate.

Manufacturability

The developed tool ensures DRC and LVS clean layouts, saving several hours of work, as layout engineers usually spend a significant amount of time repairing unintentional DRC violations during the placement and routing process. The tool creates a matching pattern for the devices while also allowing the designer to change the locations of any of them, ensuring flexibility and saving time in creating a matching pattern

for frequently used blocks. As engineers encounter fewer DRC and LVS issues, the company may reduce the number of licensed products it purchases, especially when human modifications to the generated design are required.

Ethics

The company can potentially reduce the number of licensed products it purchases, especially when human modifications to the generated design are required.

Social and Economic Impact

The social and economic impact can be clearly observed in the design cycle. Analog layout automation, in general, shortens the design cycle time and effort, which may encourage businesses to include more analog components in their devices. In the long run, analog layout automation may impact the manual analog layout business, with certain cases transitioning to a more engineering-supervised process or even full automation.

Sustainability

Sustainability and working with a wide range of technologies pose real challenges to our work. However, the code is designed to be technology independent. Although it has been tested and achieved 100

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In conclusion, the automation of analog layout design is a critical area of research and development in the field of integrated circuit design. The historical advancements in transistor technology and the emergence of integrated circuits have paved the way for the need to automate the layout process, especially for analog circuits.

The layout process plays a crucial role in determining the performance, reliability, and manufacturing yield of analog integrated circuits. Manual layout design, while effective, has become impractical due to the increasing complexity of IC designs and the need for faster turnaround times. This has led to the development of automated analog layout tools that leverage advanced algorithms to optimize layout parameters and improve overall circuit performance.

By automating the layout process, designers can achieve optimized layouts more efficiently, reduce power consumption, increase circuit efficiency, and enhance the manufacturing yield of chips. Automation also helps minimize the impact of parasitic effects and enables designers to focus more on the performance and functionality of the circuit rather than the intricate details of layout design.

However, there are still challenges and limitations that need to be addressed in the field of automated analog layout design. These include the need for adaptable and technology-agnostic tools, addressing the complexities of advanced technology nodes, improving user-friendliness, handling parasitic effects, and integrating power optimization capabilities.

Despite these challenges, ongoing research and development efforts continue to push the boundaries of analog layout automation. As technology continues to advance, automated analog layout design tools will play an increasingly vital role in enabling the creation of high-performance, low-power analog circuits. With the advancements in machine learning, cloud-based collaboration, and power optimization, the future of automated analog layout design looks promising, offering designers more efficient and effective tools to bring their analog circuit designs to life.

Extended Future Work for Automated Analog Layout Program

There are several avenues of future work to enhance the functionality of the automated analog layout program for OTA circuit design. Firstly, there is potential to evolve the graphical user interface into a technology-agnostic script, bolstering its adaptability across diverse technology platforms. Secondly, the objective of creating a parameterized cell could be set as a milestone, contributing to the establishment of a valuable intellectual property.

The future scope could also involve the strategic optimization of the area consumed by incorporating features such as routing over devices and employing a variable global routing approach. This would potentially result in a more compact design, thus enhancing the efficiency of the layout. Additionally, making the tool adaptable to handle different aspect ratios could offer greater flexibility in design choices.

The idea of developing a mechanism that rectifies designer errors in real-time, enhancing the robustness of the tool, could also be pursued. In the realm of supply current circuits, advancements could be made to the PMOS and NMOS current mirrors, including biasing the differential pair and the amplifier. The intention is to aim for a complete automation of the current mirror design specifications and its layout generation.

A more user-centric approach can be adopted by enabling the user to choose the matching method. This could enhance the customization and user-friendliness of the tool. Additionally, the tool could be designed to handle more than one finger, offering more flexibility in the design process.

Finally, an additional feature could be created for designs with an odd number of multipliers. This feature could provide two options to the designer: working matching with dummies or adjusting the design specifications. This would enable the user to make a choice based on their specific needs and design constraints, thus making the tool more versatile and adaptive.

Furthermore, exploring the integration of machine learning algorithms into the automated analog layout program holds great potential for future advancements. By leveraging the power of machine learning, the program can learn from previous designs and layout iterations, allowing it to generate more optimized layouts with minimal human intervention. This could significantly reduce the time and effort required for manual layout adjustments and fine-tuning.

Another avenue for future work involves enhancing the program's ability to handle parasitic effects. By incorporating advanced modeling techniques and algorithms, the program can accurately account for parasitic capacitances, resistances, and inductances that affect circuit performance. This would enable the tool to provide more accurate estimations of circuit behavior and improve the overall design quality.

Moreover, extending the program's functionality to support advanced technology nodes, such as FinFET or nanoscale CMOS, would be a significant step forward.

These advanced technologies pose unique layout challenges due to their complex structures and manufacturing processes. Adapting the program to handle these challenges would open up new possibilities for designing high-performance analog circuits using cutting-edge technologies.

To foster collaboration and knowledge sharing among analog circuit designers, incorporating a cloud-based platform for the automated analog layout program could be explored. This would allow multiple designers to work on the same project simultaneously, facilitating teamwork and enabling real-time feedback and collaboration. Additionally, the cloud platform could serve as a centralized repository of design libraries, templates, and best practices, further enhancing productivity and design quality.

Lastly, considering the increasing demand for energy-efficient and low-power analog circuits, integrating power optimization capabilities into the automated layout program would be highly valuable. This could involve developing algorithms that analyze the circuit's power consumption and suggest layout modifications to minimize power dissipation while maintaining performance requirements. By incorporating power optimization as an integral part of the layout process, the program would contribute to the development of energy-efficient analog designs.

In conclusion, the future work for the automated analog layout program involves expanding its adaptability to different technology platforms, optimizing layout area and flexibility, improving robustness and user-friendliness, exploring machine learning integration, addressing parasitic effects, supporting advanced technology nodes, introducing cloud-based collaboration, and incorporating power optimization capabilities. By pursuing these avenues, the program can evolve into a comprehensive tool that empowers designers to create high-performance analog circuits efficiently and effectively.

References

[1] “Analog Layout Synthesis” (2011). Available at: https://doi.org/10.1007/978-1-4419-6932-3.

[2] Placement with symmetry constraints for analog . (Accessed: March 4, 2023).

[3] Implications of proximity effects for analog design (no date). Available at: https://picture.iczhiku (Accessed: March 4, 2023).

[4] L. Minghorng, D.G. Wong, Slicing tree is a complete floorplan representation, inproceedings on Design, Automation and Test in Europe (DATE), Mar 2001, pp. 228–232.

[5] Sajid, Khusro Carothers, Jo Rodriguez, Jefrey Holman, W.. (2001). Global routing methodology for analog and mixed-signal layout. Microporous and Mesoporous Materials - MICROPOROUS MESOPOROUS MAT. 442 - 446. 10.1109/ASIC.2001.954742.

[6] Marolt, Daniel. (2019). Layout automation in analog IC design with formalized and non formalized expert knowledge.

[7] Torabi, M., (2018). Analog Layout Design Automation: ILP-Based Analog Routers.

[8] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang, ”Nonuniform multilevel analog routing with matching
constraints,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 12, pp. 1942-1954, 2014.

[9] S. Bigalke and J. Lienig, ”Load-aware redundant via insertion for electromigrationavoidance,” in Proc. of International Symposium on Physical Design - ISPD ’16, pp. 99- 106, 2016.

[10] Electromigration in metals - IOPscience - Institute of Physics. Available at: https://iopscience.iop.org/article/10.1088/0034-4885/52/3/002/meta (Accessed: March 4, 2023). [11] Fast maze router - janders.eecg.utoronto.ca (no date). Available at: https://janders.eecg.utoronto.ca/1387/readings/soukup.pdf (Accessed: March 4, 2023).

[12] Sedgewick, Robert (2002), Algorithms in C++: Graph Algorithms (3rd ed.)

[13] Gass, Saul; Fu, Michael (2013). Gass, Saul I; Fu, Michael C (eds.). ”Dijkstra’s Algorithm”. Encyclopedia of Operations Research and Management Science. doi:10.1007/978-1-4419-1153-7

[14] A method for the shortest path search by extended Dijkstra algorithm ... . Available at: https://ieeexplore.ieee.org/abstract/document/886462/citations (Accessed: March 4, 2023).