An Industrial Tool–Driven Framework for Automated Architecture, Schematic, and Layout Design of Analog Integrated Circuits

A dissertation submitted towards the degree
Doctor of Philosophy (Ph.D.)
of the Faculty of Micro and Nano Electronics
of University of Pavia

by

Danish Kaleem Ansari

UNIVERSITÀ
DI PAVIA

Supervisor: Prof. Andrea Baschirotto
Industrial Supervisor: Capodivacca Giovanni
Ph.D. Coordinator: Prof. Piero Malcovati

Abstract

This dissertation presents an industrial tool-driven methodology for multi-level automation of analog integrated circuit (IC) design, spanning architecture exploration, schematic synthesis, optimization, and layout generation. While digital design flows are highly automated, analog IC design remains largely manual due to complex trade-offs, sensitivity to device matching, and strong dependence on layout quality.

At the architecture level, the proposed approach enables automated optimization of power-management circuits using Python-based algorithms integrated with industrial circuit simulators. A DC–DC buck converter is used as a benchmark to demonstrate efficient parameter sizing through constrained sweeps, achieving a peak efficiency of approximately 86% at 1 A load under realistic operating conditions while significantly reducing manual effort.

At the schematic level, parameterized and technology-independent generators are developed for automatic synthesis of analog and mixed-signal circuits. A resistor network connected to a comparator serves as a benchmark to demonstrate variation-aware optimization that minimizes layout area while maintaining voltage accuracy and robustness. Validation across more than 200 000 Monte Carlo samples achieves a 100% pass rate across all process corners, with a voltage error below 1%. The automatically generated schematic has been successfully implemented in a silicon prototype, confirming the practical deployability of the proposed methodology.

The methodology is further extended to hierarchical generation of SAR ADC sub-blocks, with key contributions including a three-stage evolution from single-ended to differential implementations, automated technology migration, and a dynamic symbol modification framework.

At the layout level, the approach incorporates constraint-driven placement and template-based routing to preserve analog symmetry. Comparative results show that template-based routing significantly improves matching and performance compared to unconstrained routing, with automated layout generation completed in under 5 minutes compared to 3–34 hours of manual development effort. The framework also supports technology porting through rule-based layer mapping, enabling the reuse of layout templates across different process nodes.

All stages are validated using industrial EDA tools, demonstrating scalability, design consistency, and significant reductions in design time. This work shows that systematic automation of analog IC design is achievable without compromising performance or layout quality.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, **Prof. Andrea Baschirotto**, for his continuous guidance, encouragement, and invaluable technical insights throughout the course of this PhD research. His expertise, critical feedback, and support were fundamental in shaping the scope, depth, and overall quality of this work.

I would also like to acknowledge **Prof. Piero Malcovati** for his formal role as co-supervisor and for his availability within the supervisory committee.

I would like to acknowledge the guidance and continuous support provided by my industrial supervisor, **Giovanni Capodivacca**. His industrial perspective and practical feedback were essential in aligning this research with real-world electronic design automation challenges. I also gratefully acknowledge **Infineon Technologies** for providing the industrial environment, tools, and technical support that enabled the practical validation of this work.

At the *architecture level*, I would like to thank **Stefano Orlandi** and **Dominik Duner** for their valuable technical discussions and insights, which contributed to the development and validation of the architecture-level automation methodologies.

At the *schematic generation and optimization level*, I would like to acknowledge the contributions and technical support provided by **Christos Konstantopoulos**, **Tobias Kaisermayer**, **Thomas Brandtner**, **Mirjana Videnovic-Misic**, **Florian Seim**, **Daniele Privato**, **Della Rovere Giovanni**, and **Andreas Fugger**. Their expertise and discussions, including collaborations with the **Infineon Austria** team, significantly contributed to the development, robustness, and scalability of the schematic generation and optimization framework.

At the *layout level*, I would like to express my sincere thanks to **Bon Nicholas**, **Petya Popova**, and **Habal Husni** for their valuable support, technical guidance, and discussions related to analog layout generation, symmetry constraints, and routing strategies.

This research was conducted in collaboration with the **University of Pavia**, the **University of Milano–Bicocca**, and **Infineon Technologies, Padova**. I am grateful for the stimulating research environment and the technical infrastructure provided by these institutions, which enabled effective collaboration between academic research and industrial practice.

I gratefully acknowledge the financial support provided through the **PNRR doc-**

toral program, which made this research and the associated international and industrial collaborations possible. The opportunity to work in a multidisciplinary and international research environment greatly enriched both the technical and professional aspects of this work.

I would also like to express my sincere gratitude to **Dr. Syed Arsalan Jawed** and **Dr. Najam Muhammad Amin**, my Master's supervisors, whose mentorship in analog and RF IC design laid the foundation for this doctoral research.

Finally, I would like to express my sincere thanks to my family and friends for their constant support, patience, and encouragement throughout this journey. Their understanding and motivation were essential in bringing this thesis to completion.

Contents

List of Figures

List of Tables

Introduction

Analog and Mixed-Signal (AMS) integrated circuits remain one of the most experience-driven and time-consuming processes in semiconductor development. Unlike digital design, where abstraction layers and automated tools have matured significantly, AMS design still depends heavily on manual methodologies, device-level insight, and iterative tuning. This reliance on manual intervention increases design time, introduces variability, and complicates technology porting. As modern systems demand higher performance and shorter development cycles, these manual workflows become increasingly time-consuming, error-prone, and difficult to scale.

Recent progress in scripting interfaces, simulation automation, and CAD tool programmability has enabled a shift toward more structured AMS design methodologies. However, most existing automation solutions address isolated steps—such as simulation or layout placement—without establishing continuity across architecture, schematic, and layout levels. This lack of integration limits productivity and hinders the creation of reusable, portable analog design flows.

A unified framework for AMS automation can significantly improve design productivity by enabling systematic exploration at the architectural stage, reproducible schematic generation, and constraint-aware layout construction. Such a framework reduces reliance on manual trial-and-error and supports scalable design methodologies suitable for industrial applications, validated using industrial EDA tools and design environments.

Motivation

Three industry-driven trends motivate the need for unified AMS automation:

- **Rising circuit complexity.** Designs increasingly require extensive multi-dimensional design-space exploration, which is difficult to perform manually in a systematic manner.

- **Fragmented design flows.** Architectural design, schematic implementation, and layout construction are often performed independently, relying on designer intuition rather than reproducible frameworks.

- **Demand for fast and portable design methodologies.** Industrial development cycles increasingly require rapid iteration, technology migration, and consistent design practices across teams and product lines.

Research Objectives

The objectives of this work are to:

- Develop a systematic methodology for automating architecture-level exploration in analog and mixed-signal circuits.
- Create generalized approaches for automated schematic generation and design-space evaluation of AMS building blocks.
- Initiate layout-automation techniques based on reusable templates and constraint-driven design principles.
- Integrate these levels into a coherent automation flow that improves design consistency, portability, and productivity.

Contributions of this Dissertation

This dissertation presents an integrated AMS design automation framework spanning architecture, schematic, and layout levels. The main contributions are:

- **Automated architecture exploration for DC–DC converters**, enabling systematic evaluation of design choices through simulation-driven parameter sweeps.
- **Automated optimization and schematic generation of resistor networks**, using a deterministic recursive flow integrated with industrial CAD tools, with the generated design successfully validated through silicon prototype implementation.
- **Hierarchical schematic generation of SAR ADCs**, supporting full top-down construction without optimization.
- **Initial layout automation strategies**, including reusable layout templates and constraint-aware placement and routing methods.

Thesis Organization

The remainder of this thesis is structured as follows:

- **Chapter 1** reviews AMS design flows and existing automation approaches.

- **Chapter 2** presents architecture-level automation for DC–DC converters.
- **Chapter 3** describes automated schematic generation and optimization for resistor networks.
- **Chapter 4** extends schematic automation to SAR ADCs.
- **Chapter 5** introduces template-based and hybrid layout automation techniques.
- The **Conclusion** summarizes the contributions and outlines future work.

Chapter 1

State of the Art and Background

1.1 Introduction to Analog and Mixed-Signal Systems

Analog and Mixed-Signal (AMS) integrated circuits play a critical role in modern semiconductor systems by bridging the physical world and digital processing engines. As system-on-chip (SoC) platforms evolve to support increasingly complex applications—including wireless communication, sensor interfaces, biomedical electronics, autonomous vehicles, and power management—AMS blocks such as data converters, regulators, oscillators, filters, and amplifiers have become indispensable [1, 2].

Unlike digital circuits, which rely on discrete logic levels and benefit from highly automated design flows, AMS circuits operate on continuous-time signals, are deeply influenced by device-level physics, and demand strong coupling between electrical behavior and physical layout. This coupling makes standard cell abstraction infeasible and necessitates simultaneous consideration of architecture, circuit behavior, device matching, parasitics, and layout geometry [3]. As CMOS technology continues to scale, variability, reduced intrinsic gain, and heightened parasitic effects further complicate AMS design.

1.1.1 Role of AMS Circuits in Modern SoCs

AMS circuits perform essential functions such as:

- Interfacing with sensors and actuators,
- Conditioning and converting analog signals,
- Regulating and distributing power,
- Providing timing and clock generation,

- Enabling wireless communication.

These blocks determine system-level performance metrics such as dynamic range, noise floor, signal integrity, battery life, and energy efficiency. Consequently, AMS design quality often constrains overall SoC capability.

Figure 1.1 illustrates how AMS circuits connect the physical world to digital processing through key building blocks such as power management, analog filters, amplifiers, and data converters. The diagram highlights the central role of AMS subsystems in bridging real-world signals and digital computation within a modern SoC.

The unique characteristics of AMS circuits—continuous operation, technology-dependent behavior, and high sensitivity to layout—have historically made AMS design an expert-driven, iterative, and time-intensive process. These challenges motivate enhanced design productivity through structured automation across multiple abstraction levels.

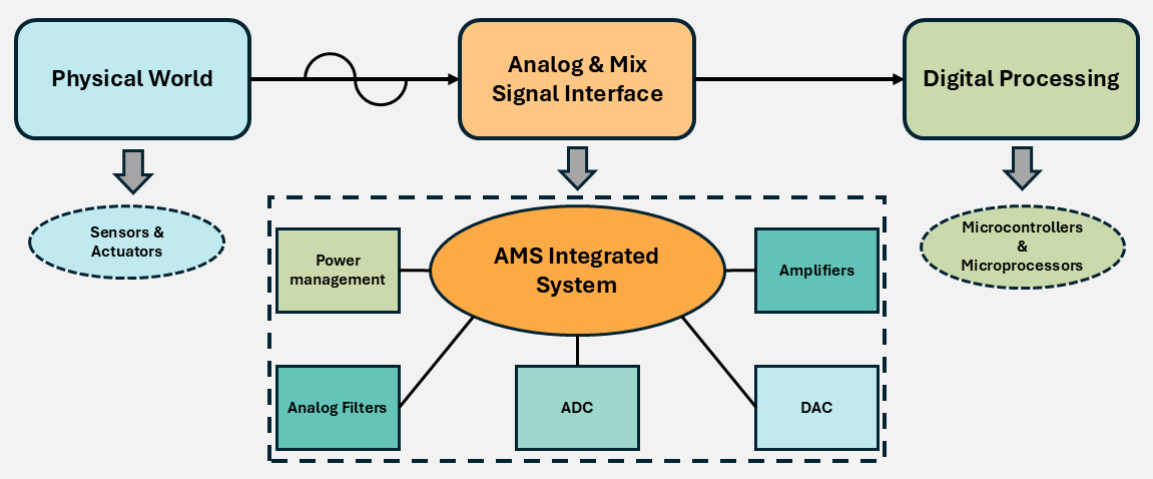


Figure 1.1: Overview of analog and mixed-signal (AMS) systems.

1.2 Overview of Analog & Mixed-Signal Design Flows

The design of analog and mixed-signal (AMS) integrated circuits follows a multi-stage workflow shown in Figure 1.2 that spans system-level specification, architectural exploration, circuit implementation, layout development, and final fabrication. Each stage requires specialized knowledge and often involves several iterations to meet performance, power, and area (PPA) targets.

1.2.1 System-Level Specification & Modeling

The process begins with defining system-level requirements, including resolution, gain, bandwidth, efficiency, noise, linearity, and stability margins. These specifications are typically evaluated using high-level modeling environments such as MATLAB, Simulink, or Python-based behavioral models to verify feasibility.

1.2.2 Architecture Exploration & Design

This stage focuses on selecting the appropriate architecture based on system targets. Examples include choosing buck versus boost converters for power management, folded-cascode versus differential pairs for amplifiers, and successive-approximation versus pipeline architectures for analog-to-digital converters. Architectural decisions strongly influence achievable performance and layout complexity.

1.2.3 Circuit Design & Verification

Transistor-level design involves device sizing, biasing strategies, optimization loops, and extensive simulation campaigns. Typical verification steps include corner analyses, Monte Carlo simulations, noise evaluations, and stability assessments. This stage is often the most expertise-intensive, requiring iterative refinement to meet design specifications.

1.2.4 Layout Design & Post-Layout Verification

Physical layout is critical to AMS performance due to device matching, parasitic sensitivity, isolation, and symmetry requirements. Layout tasks include placement, routing, shielding, and adherence to process design rules. Post-layout extraction (PEX) and parasitic-aware simulations validate the impact of layout-induced effects and may trigger additional design iterations.

1.2.5 Fabrication & Testing

Once design and verification are complete, the circuit proceeds to tape-out for fabrication. Silicon validation includes laboratory measurements to confirm functionality, performance, and robustness across operating conditions.

Across this workflow, several bottlenecks persist, including strong dependence on designer expertise, large iteration counts between schematic and layout stages, difficulty performing exhaustive design-space exploration, and limited portability across technology nodes. These challenges motivate ongoing research into AMS automation methodologies.

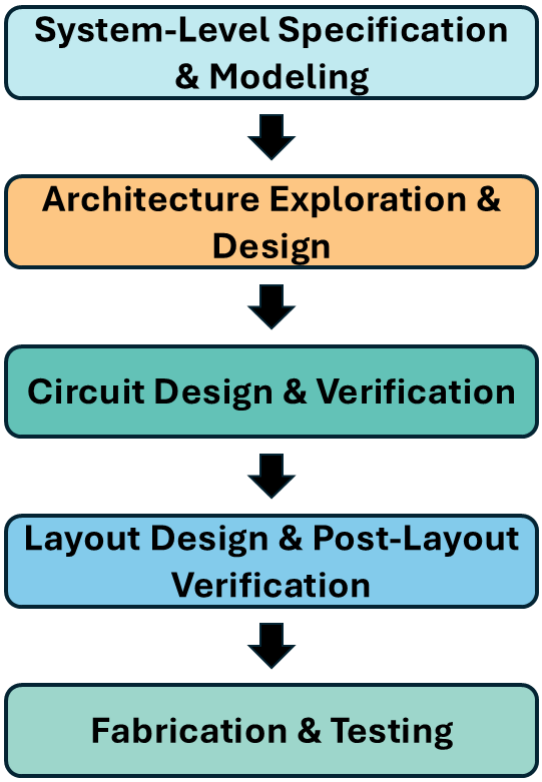


Figure 1.2: High-level design flow for AMS integrated circuits

1.3 Automation Levels in AMS Design

Automation in AMS design spans three primary abstraction levels: architecture-level automation, schematic-level automation, and layout-level automation. Each level faces distinct challenges but offers opportunities for productivity gains. A hierarchical view of these levels is shown in Figure 1.3.

1.3.1 Architecture-Level Automation

Architecture-level automation aims to evaluate circuit topologies and perform high-level design-space exploration. Methods include:

- behavioral modeling and simulation [3],
- topology generation and pruning,
- machine-learning–based design-space projection [4],
- evolutionary and heuristic search strategies [5].

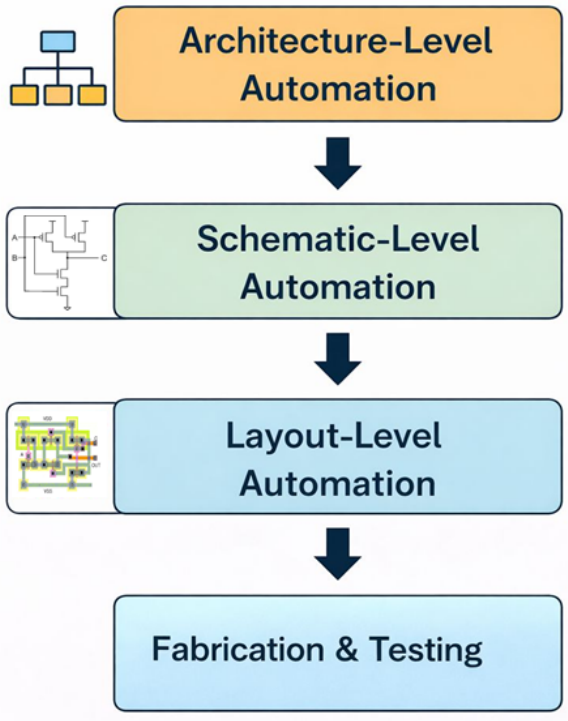


Figure 1.3: Three-Level of AMS Automation Hierarchy

However, most architecture-level tools stop before schematic generation, resulting in disconnected workflows.

1.3.2 Schematic-Level Automation

At the schematic level, automation technologies include:

- template-driven instance generation [6],
- automatic device sizing through geometric programming [7],
- graph-based circuit synthesis [8],
- simulation-based optimization loops.

These methods significantly reduce manual effort but often lack integration with layout-level constraints.

1.3.3 Layout-Level Automation

Layout automation is the least developed among the three. Key approaches include:

- constraint-aware placement engines [10],

- symmetry and matching rule enforcement,
- template-based layout reuse,
- analog routing algorithms.

Despite advancements, full analog layout automation remains an open challenge due to strong dependency on layout–circuit interactions.

1.4 State-of-the-Art in Architecture-Level Automation

Architecture-level automation addresses the earliest design stages where system specifications are translated into topology choices. Historically, this process has depended on designer expertise and manual exploration of circuit configurations. Recent advances have introduced algorithmic and simulation-driven techniques to improve efficiency, scalability, and consistency.

1.4.1 Topology Generation and Classification

Several works have focused on automatically generating and classifying circuit topologies. Early studies employed rule-based and template-based topology construction [6], while more recent approaches incorporate symbolic representations and graph grammars for generating feasible analog topologies [8]. These techniques enable systematic enumeration of circuit structures but often face scalability limitations as circuit complexity increases.

1.4.2 Simulation-Driven Architecture Exploration

Simulation-driven frameworks evaluate candidate topologies across a multidimensional parameter space. Tools such as LTspice, SIMPLIS, and simplified SPICE models facilitate rapid evaluation of power converters [11, 12], amplifiers, and data converters [5]. These methods enable:

- automated parameter sweeping,
- construction of Pareto-optimal performance fronts,
- pruning of infeasible configurations,
- early estimation of power, area, and efficiency.

However, architecture-level exploration tools typically stop short of generating transistor-level schematics or considering layout constraints, creating a discontinuity between exploration and implementation.

1.4.3 Machine Learning in Architecture Exploration

Machine learning (ML) has enabled novel techniques for design-space exploration, performance prediction, and topology selection. Neural-network–based regressors, Gaussian Process models, and reinforcement learning have been applied to predict circuit behavior, accelerate convergence, or propose candidate architectures [4]. While ML models improve efficiency, they often require large training datasets and may lack interpretability when applied to analog circuits.

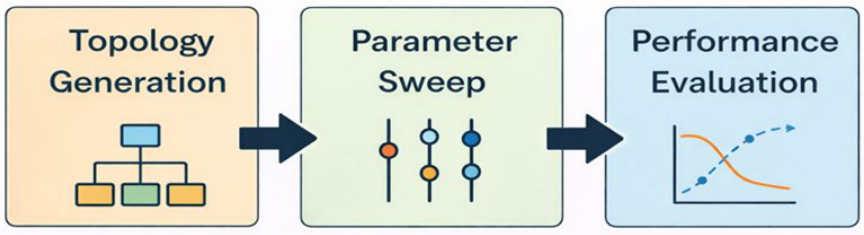


Figure 1.4: High-level view of architecture-level automation

Despite meaningful progress, none of the existing methods provides an integrated path that connects architecture-level results to schematic and layout automation.

Chapter 2 of this dissertation bridges this gap by introducing a Python-based integration framework for SIMPLIS/SIMetrix that enables systematic topology evaluation and design-space exploration for DC–DC converters.

1.5 State-of-the-Art in Schematic-Level Automation

Schematic-level automation has received considerable academic attention, with methods targeting topology generation, device sizing, netlist creation, and algorithmic design improvement. Still, most approaches lack the flexibility or integration required for real-world industrial flows.

1.5.1 Topology and Schematic Generation

Template-based methods construct schematics from predefined structural patterns. These methods support rapid instantiation of common blocks such as differential pairs, current mirrors, and operational amplifiers [6]. Graph-based synthesis techniques generate candidate analog circuits using graph transformations, automatically identifying feasible topologies that satisfy functional specifications [8].

Such approaches have demonstrated considerable promise but often suffer from:

- limited applicability across circuit classes,

- difficulty managing hierarchical designs,
- lack of tight integration with industrial CAD tools.

1.5.2 Device Sizing and Optimization

Device sizing automation has evolved across three major approaches:

- **Geometric programming and convex optimization**, widely used for analog circuits with amenable convex formulations [7].
- **Simulation-based optimization**, where simulated performance guides iterative tuning [20].
- **Metaheuristic algorithms**, including genetic algorithms, PSO, simulated annealing, and Bayesian optimization.

While effective in specific domains, sizing algorithms typically require circuit-specific models and do not inherently solve the schematic-generation problem. Chapter 3 of this thesis addresses passive network automation through resistor optimization and automatic schematic generation. The resulting design was integrated into a fabricated chip; while the physical layout was provided externally, the electrical design and schematic automation form the core contribution of this work.

Figure 1.5: High-level view of schematic-level design generation

1.6 State-of-the-Art in ADC Automation

Analog-to-digital converters (ADCs) are essential for interfacing between analog signals and digital processing units. Among various ADC architectures, SAR ADCs have gained prominence due to their excellent energy efficiency and scalability.

1.6.1 Behavioral-Level ADC Automation

Many works focus on behavioral modeling and architectural comparison. MATLAB, Simulink, and Python frameworks have been used to explore:

- capacitor DAC sizing,
- comparator performance estimation,
- digital control logic validation,
- settling and decision time analysis.

These tools accelerate architecture selection but do not generate transistor-level schematics.

1.6.2 Automation of SAR ADC Sub-Blocks

DAC optimization, bootstrapped switch sizing, comparator design, and SAR logic synthesis have been partially automated in literature [21]. Approaches include:

- monotonic capacitor array optimization,
- switch linearity optimization,
- behavioral comparator analysis.

However, these methods target isolated sub-blocks rather than full ADC integration.

1.6.3 Hierarchical ADC Schematic Generation

Very few works propose hierarchical transistor-level schematic automation for SAR ADCs. The majority of existing frameworks stop at high-level modeling or block-level sizing. Industrial adoption remains low because:

- sub-block integration depends on layout-sensitive interactions,
- analog and digital domains must be co-designed,
- technology independence is difficult to maintain.

The hierarchical SAR ADC generation introduced in this thesis fills this gap by providing a structured template-driven methodology capable of building complete SAR ADC schematics, as presented in Chapter 4.

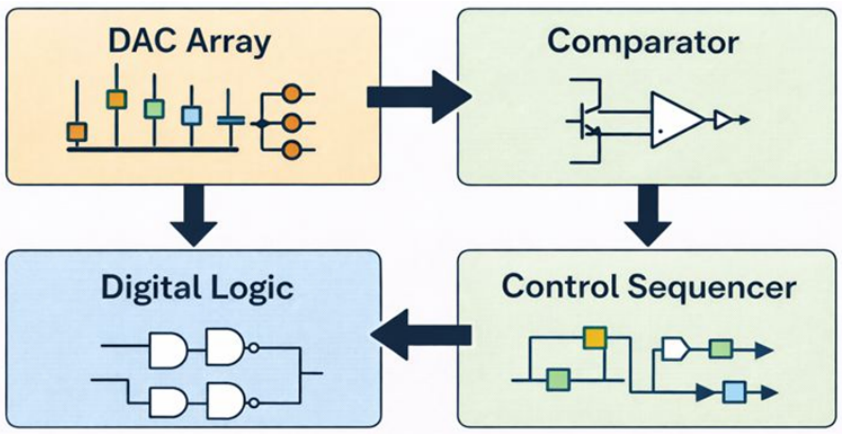


Figure 1.6: High-level block diagram of SAR ADC

1.7 State-of-the-Art in Layout Automation

Analog layout automation remains one of the most challenging domains in EDA research. Unlike digital layout—where standardized cells enable full automation—analog circuits require intricate placement strategies, careful routing, and strict adherence to symmetry and matching constraints.

1.7.1 Constraint-Aware Placement

Modern placement engines incorporate constraints such as:

- device matching,
- symmetry pairs,
- clustering,
- common-centroid placement,
- guard-rings and isolation.

Among the most prominent academic frameworks, BAG (Berkeley Analog Generator) provides a highly flexible, Python-based generator framework capable of producing layout-versus-schematic (LVS)-clean layouts from parameterized templates. However, it requires days of expert-level effort per design and is tightly coupled to a single process design kit (PDK), limiting its portability. ALIGN is an open-source analog layout automation system that accepts a SPICE netlist and automatically generates placement and routing solutions under symmetry and matching constraints, achieving runtimes of 10–30 minutes. MAGICAL (Machine Generated Analog IC Layout) similarly targets full automation of analog layout from a netlist, achieving runtimes of 1–5

minutes with layout quality comparable to manual routing. Despite these advances, both ALIGN and MAGICAL [30, 41] still require significant expert-level setup effort and remain limited to a single technology node, highlighting the need for multi-PDK compatible and template-reusable approaches such as the one proposed in this thesis.

1.7.2 Template-Based Layout Generation

Template reuse is widely employed in industry for current mirrors, differential pairs, OTAs, and ADC sub-blocks. Templates reduce design time and improve portability but require manual tuning across PDKs.

1.7.3 Routing and Parasitics

Routing remains the most difficult aspect of analog layout due to:

- parasitic coupling,
- shielding requirements,
- multi-layer constraints,
- differential routing rules,
- stringent design-rule checks.

Recent efforts incorporate machine learning to predict routing congestion or parasitic effects, but practical adoption is limited.

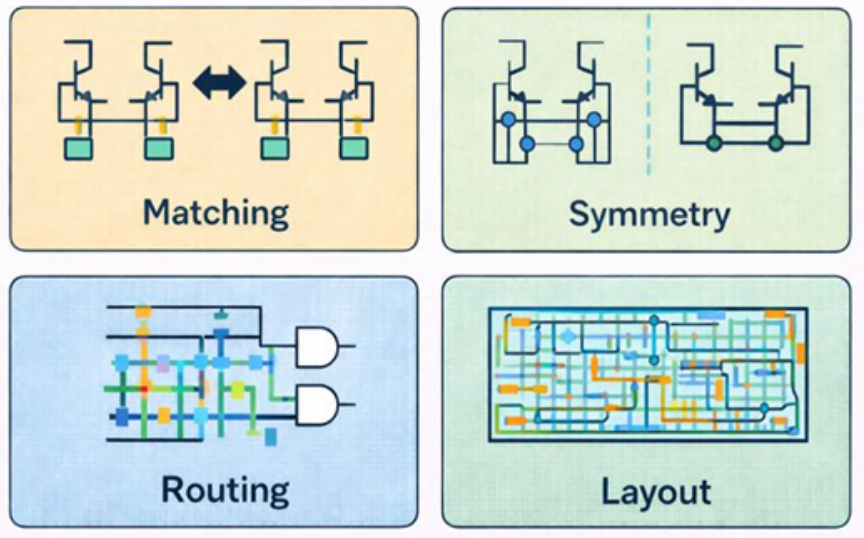


Figure 1.7: High-level view of template-based analog layout automation.

Despite advancements, fully automated analog layout remains an open challenge—further motivating the template-based and hybrid automation approaches developed in Chapter 5.

1.8 Research Gap Analysis

Despite extensive research in analog automation, several critical gaps remain:

1.8.1 Lack of End-to-End Automation

Most existing works address automation at only one abstraction level—architecture, schematic, or layout. Very few frameworks provide:

- consistent data flow across levels,
- reuse of design knowledge,
- hierarchical constraint propagation,
- unified optimization objectives.

This fragmentation forces designers to manually translate information between design stages, introducing errors and inefficiencies.

1.8.2 Limited Industrial Integration

Academic tools often lack compatibility with industrial design environments such as Cadence Virtuoso. As a result:

- schematic generators are not compatible with PCells,
- layouts cannot be validated against foundry rules,
- architectures are explored without considering implementation constraints.

A practical AMS automation framework must interface with existing industrial tools.

1.8.3 Insufficient Hierarchical Schematic Automation

Schematic automation research primarily focuses on single blocks (e.g., OTAs or filters). Complete hierarchical generation—such as full SAR ADCs—remains rare due to:

- complex block interactions,
- mixed-signal boundaries,
- technology-dependent structures,
- layout-driven decisions.

This leaves a significant gap in real-world applicability.

1.8.4 Analog Layout Automation Still Limited

Despite advances in placement engines and ML-based approaches, automated layout tools still face challenges with:

- matching and symmetry,
- routing parasitics,
- higher-layer routing complexity,
- template parameterization across technologies.

Realistic, reusable layout automation remains an open research problem.

1.8.5 Gap Summary

The literature lacks:

- an integrated multi-level flow,
- automation frameworks that work with real industrial tools,
- hierarchical schematic generators for large AMS circuits,
- portable layout-generation mechanisms.

This thesis directly addresses these gaps.

1.9 Summary and Research Contributions

This chapter presented a comprehensive overview of AMS design flows, automation methodologies, tool integration strategies, and state-of-the-art research across architecture, schematic, and layout domains. Although substantial progress has been made, the literature reveals several persistent challenges:

- architecture exploration is rarely integrated with transistor-level design,
- schematic-generation tools are often domain-specific,
- hierarchical AMS design generation remains limited,
- analog layout automation continues to require significant manual intervention,
- tool interoperability in industrial environments remains a major barrier.

These limitations highlight the need for unified frameworks capable of bridging architecture, schematic, and layout automation through scalable algorithms and industry-compatible tool integration.

The gaps identified in this chapter directly motivate the contributions of this thesis. Chapter 2 presents architecture-level automation for DC–DC converters. Chapter 3 introduces resistor optimization and automatic schematic generation for passive networks, validated through chip fabrication — the physical layout was provided externally, with electrical design forming the core contribution. Chapter 4 develops hierarchical SAR ADC schematic generation. Chapter 5 presents template-based layout automation. Together, these contributions form an integrated multi-level AMS automation framework.

The next chapter introduces the automated architecture-level framework developed in this thesis for DC–DC converters. This framework employs Python–SIMPLIS integration to perform systematic topology evaluation, parameter sweeping, and design-space exploration, laying the foundation for higher-level automation presented in subsequent chapters.

Chapter 2

Architecture level for DC-DC Converter

2.1 Introduction to DC–DC Converter Architectures

DC–DC switching converters are fundamental components in modern power-management systems, enabling efficient voltage conversion for a wide range of electronic applications, including embedded systems, SoCs, IoT devices, automotive electronics, and battery-powered platforms. Their primary function is to regulate an unregulated input supply and deliver a stable output voltage while maintaining high efficiency across varying load and line conditions.

At the architectural level, DC–DC converters can be classified into several widely used topologies such as buck, boost, buck–boost, SEPIC, and flyback converters. Each topology provides a specific voltage conversion relationship (step-down, step-up, or bidirectional) and is selected based on system-level requirements such as input range, output voltage, load current, cost, power density, and efficiency targets. Among these, the buck converter remains the most prevalent in low-voltage digital systems due to its simplicity, compact design, and inherently high efficiency.

Figure 2.1 shows a representative buck converter architecture consisting of a high-side switch, a low-side switch or diode, an inductor, an output capacitor, and a feedback control loop. Regardless of the specific control strategy or device implementation, the core energy-transfer mechanism remains the same: the switching network modulates the inductor current, and the output filter maintains a stable regulated voltage.

Designing a DC–DC converter at the architecture level involves navigating a multidimensional parameter space defined by parameters such as inductance, output capacitance, switching frequency, duty cycle, and load conditions. Because these factors interact nonlinearly, manual tuning of component values and simulation conditions

is highly time-consuming and often limited to a narrow region of the feasible design space. Architecture-level exploration, therefore, requires extensive transient simulations and parametric analyses before a topology or component set can be selected for schematic-level refinement.

This chapter focuses on the high-level architectural behavior of DC–DC converters and the automated parameter-exploration framework used to evaluate candidate designs across a wide range of operating conditions. A simplified representation of a generic regulated DC–DC converter architecture is provided in Fig. 2.1, highlighting the main energy-processing stages and the closed-loop feedback path required for stable voltage regulation. Parts of the architecture-level automation methodology presented in this chapter were previously published [18].

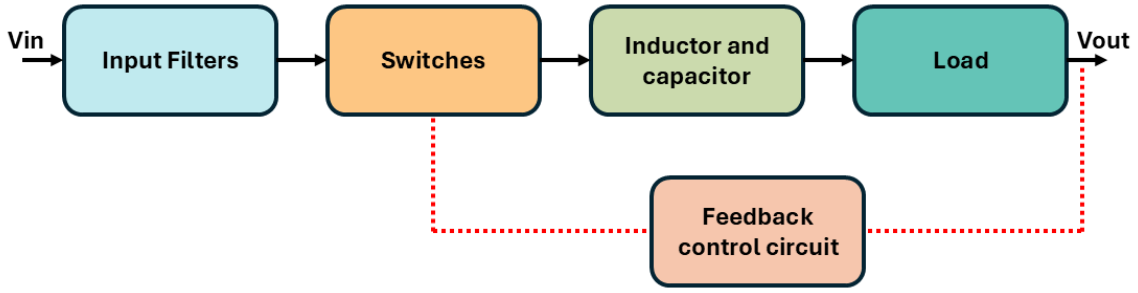


Figure 2.1: Representative architecture of a DC–DC buck converter.

.

2.2 Automated Design-Space Exploration Framework

Architecture-level design of DC–DC converters requires evaluating the impact of key design parameters on converter performance. Traditional manual design approaches rely heavily on designer intuition and iterative trial-and-error, which limits exploration of the full design space. To overcome these limitations, this work introduces an automated architecture-exploration methodology that systematically evaluates converter behavior across a wide range of component and operating conditions.

In a buck converter, several parameters strongly influence steady-state and transient performance:

- **Inductor value (L):** determines inductor ripple, peak current, transient response, and overall stability margin.
- **Capacitance (C) and ESR:** dictate output ripple, converter bandwidth, and load-transient behavior.

- **Switching frequency (f_{sw}):** controls the trade-off between conduction losses and switching losses, influencing efficiency.

- **Duty cycle (D):** governs the static conversion ratio and affects dynamic behavior under load variations.

- **Load resistance ($R_{\mathbf{load}}$):** impacts efficiency, current ripple, and loop stability.

Given the nonlinear interaction between these parameters, purely manual exploration is impractical for thorough optimization. This motivates the need for an automated approach capable of running large parameter sweeps and extracting performance metrics systematically.

Figure 2.2 illustrates the high-level automation flow. The process begins by defining the **design parameters**, which include component values and operating conditions to be explored. These parameters are passed to the **Python controller**, which programmatically generates the required SIMPLIS input files through an **Application Programming Interface (API)** layer. The API facilitates automated netlist creation and invocation of the SIMetrix/SIMPLIS simulation engine. For every parameter combination, the engine is invoked programmatically to perform transient analysis, after which waveform data—such as output voltage, inductor current, and switching-node behavior—are exported to Python for post-processing. The detailed sequence of this flow is further illustrated in Fig. 2.3.

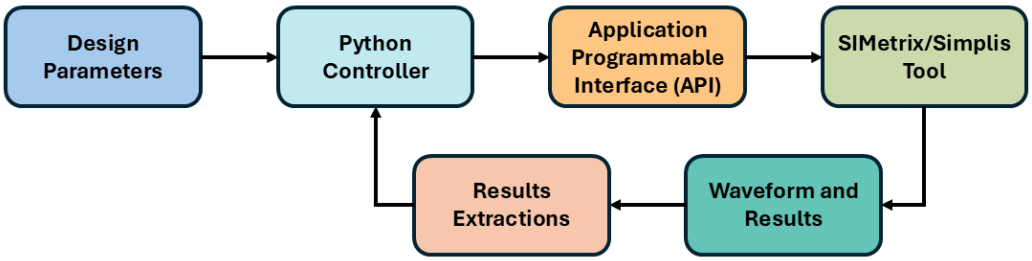


Figure 2.2: High-level automation flow

Key performance metrics, including output voltage ripple, peak inductor current, efficiency, and settling time, are then computed automatically. Each operating point is evaluated against design objectives, enabling infeasible designs to be filtered out and feasible regions of the design space to be identified. The aggregated results reveal performance trends, such as efficiency sensitivity to inductance or ripple variations with capacitance, that guide optimal component selection. By combining automated simulation with structured design-space exploration, the framework ensures reproducibility across all parameter combinations and provides meaningful insights into converter behavior across varying conditions.

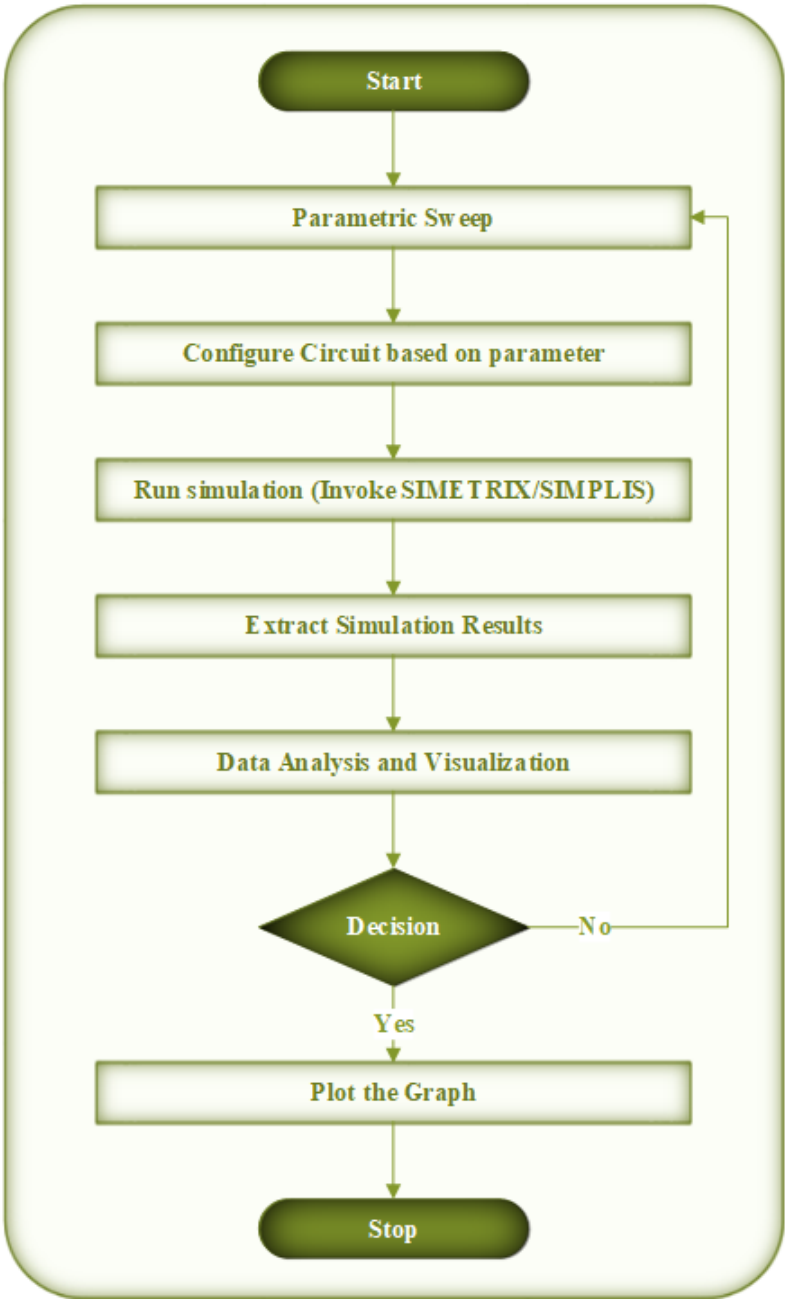


Figure 2.3: Automated simulation flow showing the parameter sweep, netlist configuration, SIMPLIS execution, result extraction, decision stage, and final visualization.

2.3 Schematic and Behavioral Modeling

The automated evaluation framework relies on two levels of circuit abstraction: a detailed behavioral model suitable for parameter sweeping and a high-level functional model that represents closed-loop behavior. Both models are implemented in SIMPLIS to capture the essential switching dynamics of the buck converter without the computational overhead of device-level SPICE simulations.

Figure 2.4 shows the annotated SIMPLIS schematic used as the core behavioral model. The schematic includes the switching devices, LC output filter, and current-measurement elements required to observe key waveforms such as inductor current and output voltage. Behavioral components—including variable blocks, RC filters, CCVS elements, POP trigger circuitry, and measurement probes—are highlighted because they define the configurability of the circuit model. These elements expose component values such as inductance, capacitance, and load to the simulation environment, enabling rapid modification of the schematic across operating points.

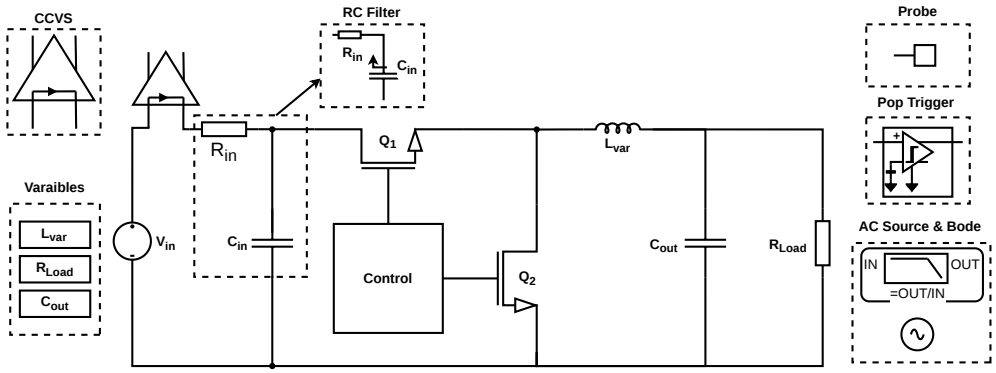


Figure 2.4: Annotated SIMPLIS schematic of the synchronous buck converter.

A high-level functional representation of the converter is also employed for abstraction and control-behavior evaluation. As shown in Fig. 2.5, this block-level model includes the power stage, PWM comparator, current-sense block, slope-compensation path, and the digital or mixed-signal logic that generates the switching signals. While it does not expose device-level detail, it accurately captures the loop dynamics required for transient and AC stability analysis.

Together, these two modeling abstractions—detailed behavioral and high-level functional—form the structural basis on which the automated optimization and analysis framework operates. These models define the circuit behavior being evaluated, while the automation methods described in the next section define *how* this evaluation is performed.

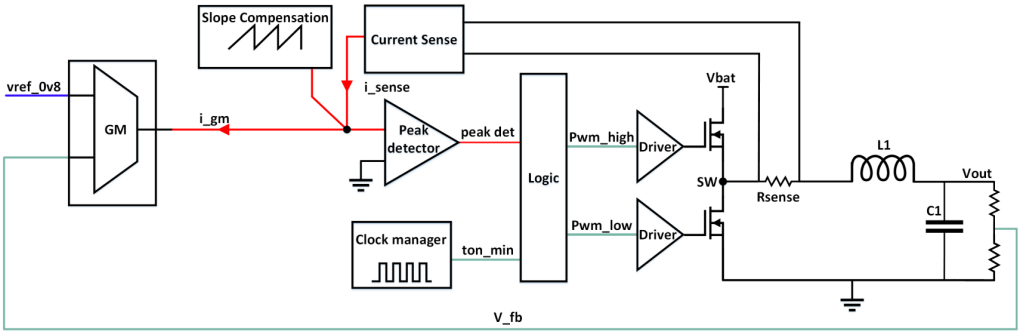


Figure 2.5: High-level block diagram of the synchronous buck converter used for control-behavior evaluation.

2.4 Optimization and Analysis Methods

The automated framework evaluates the converter models across a wide range of operating conditions using Python–SIMPLIS co-simulation. Once the modeling abstractions are defined (Section 2.4), optimization is performed through a combination of parameter sweeping, transient analysis, and frequency-domain AC analysis. These procedures are identical for both the open-loop synchronous buck converter and the high-level controlled model, enabling consistent evaluation across abstraction levels.

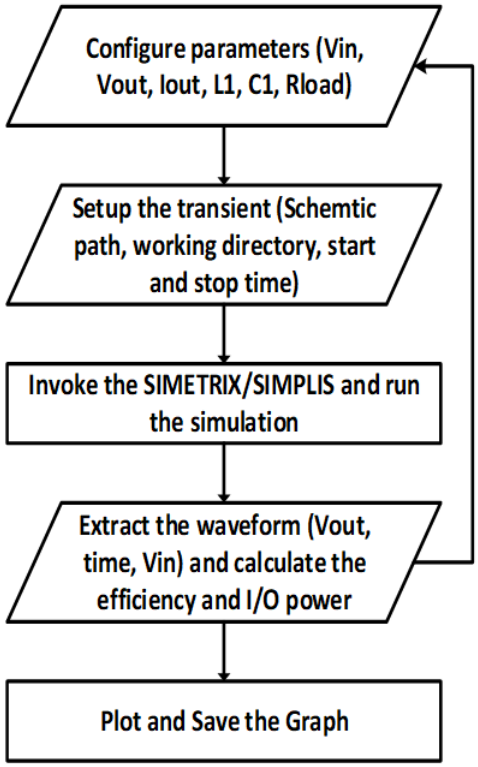
2.4.1 Parameter Sweep and Optimization Workflow

The optimization process begins by defining a set of design parameters to be swept, such as the inductor value, output capacitance, load resistance, and switching frequency. Python updates these parameters in the SIMPLIS schematic through exposed variable blocks before each simulation run. For every operating point, the simulator is invoked automatically, and waveform data are exported for further processing.

After each simulation completes, Python extracts performance metrics—such as efficiency, peak inductor current, output-voltage ripple, settling behavior, and switching characteristics. This automated extraction enables rapid identification of feasible or optimal design regions without manual intervention.

2.4.2 Transient Analysis Workflow

Figure 2.6 illustrates the transient-analysis algorithm used for evaluating each parameter combination. The algorithm initializes simulation parameters, iterates through all sweep points, performs transient simulations, and computes time-domain metrics from the resulting waveforms. A Python pseudocode representation of this workflow is provided in Fig. 2.6, demonstrating how the algorithm is translated into a programmable flow.



(a)

```
def parameters (Vin, Vout, Iout, L1, C1, ESR, Rload)
trans.options
    ( def schematic file,
     working directory for results,
     start and stop time)

Initialize arrays to store simulation results.
For each Vin value:
   For each Vout_req value:
      If Vout_req ≤ Vin, proceed:
         Update parameter and run simulation.
         Calculate output metrics:
               - V_load, Rl, I_out, Avg_power_out,
               - I_in, Avg_power_in, Efficiency, precision.
         Append results to respective arrays.
            Print simulation results for current
parameters.
            Plot efficiency vs. output current.
            Save the plot.
   Show the plot.
   Reset arrays for the next Vin value.
End loop.
```

(b)

Figure 2.6: (a) An algorithm for Transient Analysis (b) Python pseudocode

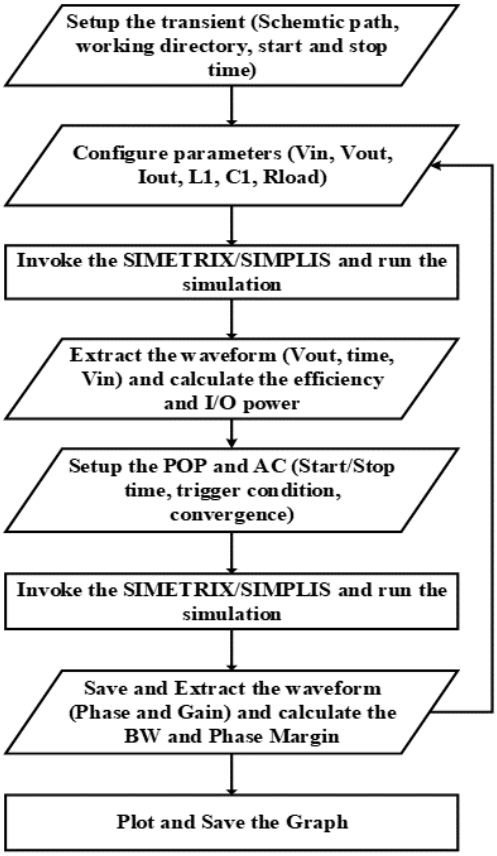
2.4.3 POP and AC Stability Analysis Workflow

To evaluate closed-loop stability characteristics, the framework performs a Periodic Operating Point (POP) analysis followed by small-signal AC analysis. POP analysis determines the steady-state switching solution, which is a prerequisite for accurate linearization. Once a valid POP solution is obtained, AC analysis computes the loop gain, phase response, crossover frequency, bandwidth, and phase margin.

The algorithmic steps for POP and AC analysis are summarized in Fig. 2.7. The corresponding Python pseudocode in Fig. 2.7 demonstrates how these operations are automated through scripted calls, allowing frequency-domain stability evaluation across all sweep points.

2.4.4 Summary

Together, the parameter sweep, transient-analysis workflow, and POP–AC stability evaluation create a unified optimization framework applicable across both converter




```
def parameters (Vin, Vout, Iout, L1, C1, ESR, Rload)
trans.options
    ( def schematic file,
      working directory for results,
      start and stop time)

Define options for POP and AC analysis:
   - Use the schematic file used for transient analysis
   - Use the directory used for transient analysis.
   - Set simulation parameters for the AC analysis
      • Stop time
      • Start frequency
      • Stop frequency
   - Set options for the POP analysis,
      • Including maximum period
      • Convergence criteria
      • Trigger gate and trigger condition

Start the simulator and run the AC analysis simulation.

Loop through parameters:
    Get Vout, pert, and freq.
    Calculate mag: mag = 20 * np.log10(np.abs(Vout / pert)).
    Calculate phase: phase = np.angle(Vout / pert, deg=True).
    Compute BW and PM.
    Plot phase and mag.
    Save the plot.
```

(b)

Figure 2.7: (a) An algorithm for POP and AC Analysis (b) Python pseudocode

abstractions. This enables comprehensive assessment of efficiency, ripple, dynamic behavior, and stability margins across a wide design space. The simulation results obtained from these automated procedures are presented in next section.

2.5 Simulation Results

2.5.1 Open-Loop Synchronous Buck Converter

The open-loop synchronous buck converter is evaluated across a wide range of inductor values, load currents, and parasitic parameters to understand its steady-state efficiency and dynamic behavior. Since no feedback loop is present, the results directly illustrate how component selection influences converter performance.

2.5.1.1 Efficiency as a Function of Inductor Value

Figure 2.8 shows the converter efficiency as the inductance is varied while keeping the input voltage, output voltage, and load fixed. The efficiency decreases monotonically with increasing inductance. Larger inductors reduce ripple but introduce higher copper losses and increased energy storage, leading to reduced efficiency. This trend is consistent with analytical predictions for open-loop converters, where conduction losses dominate at higher inductance values.

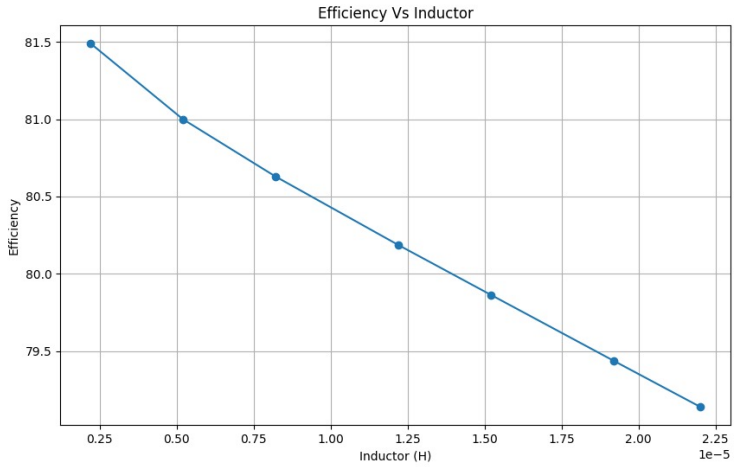


Figure 2.8: Efficiency versus inductor value for the open-loop buck converter.

2.5.1.2 Efficiency as a Function of Load Current

Figure 2.9 illustrates efficiency variation with output current. Efficiency increases sharply as the converter moves out of the light-load region, reaching a peak around 1–1.5 A, and then gradually decreases at higher load currents due to increased conduction losses. The shape of this curve is typical for synchronous buck converters operating open-loop, reflecting the trade-off between switching losses at light load and conduction losses at heavy load.

2.5.1.3 Impact of Inductor Series Resistance

To analyze parasitic effects, the inductor resistance (DCR) is swept while keeping inductance fixed. As shown in Fig. 2.10, even small increases in DCR lead to measurable efficiency degradation. This highlights the importance of selecting inductors with low series resistance, especially in designs targeting high efficiency.

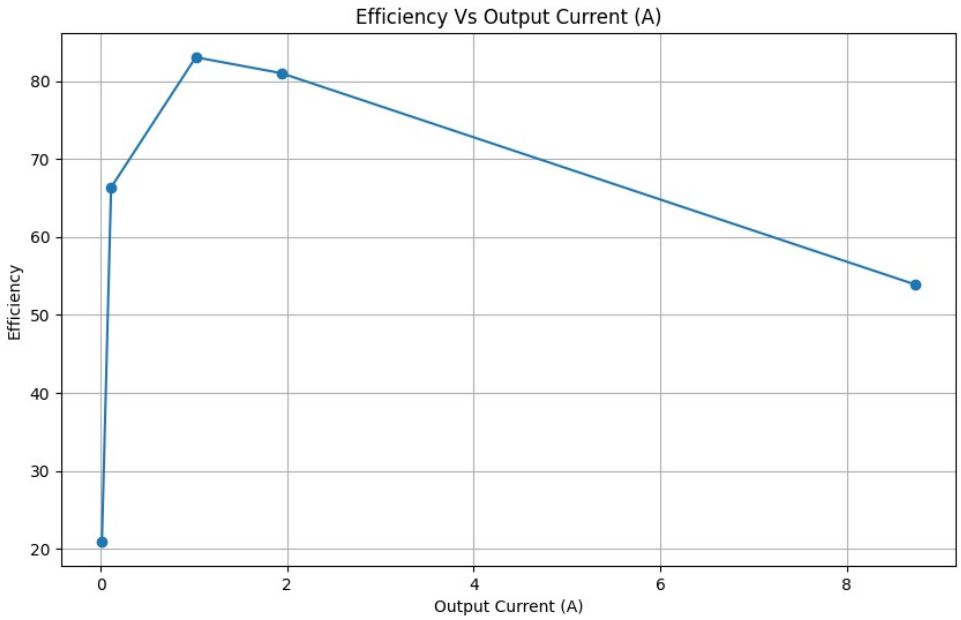


Figure 2.9: Efficiency versus output load current. Peak efficiency occurs near 1–1.5 A.

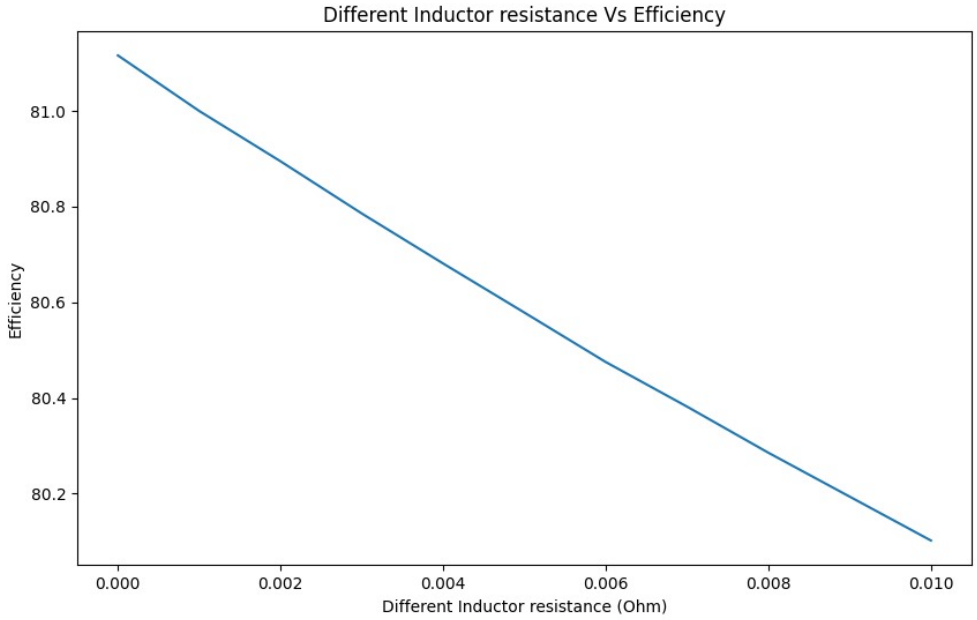


Figure 2.10: Efficiency degradation with increasing inductor series resistance.

2.5.1.4 Transient Response

Figure 2.11 presents the transient inductor current and load-voltage response for various inductor values. Higher inductance reduces current ripple but slows down dynamic response, increasing settling time. Conversely, smaller inductors settle faster but exhibit larger peak current and higher ripple. These observations confirm the classical L–C trade-off in open-loop power stages.

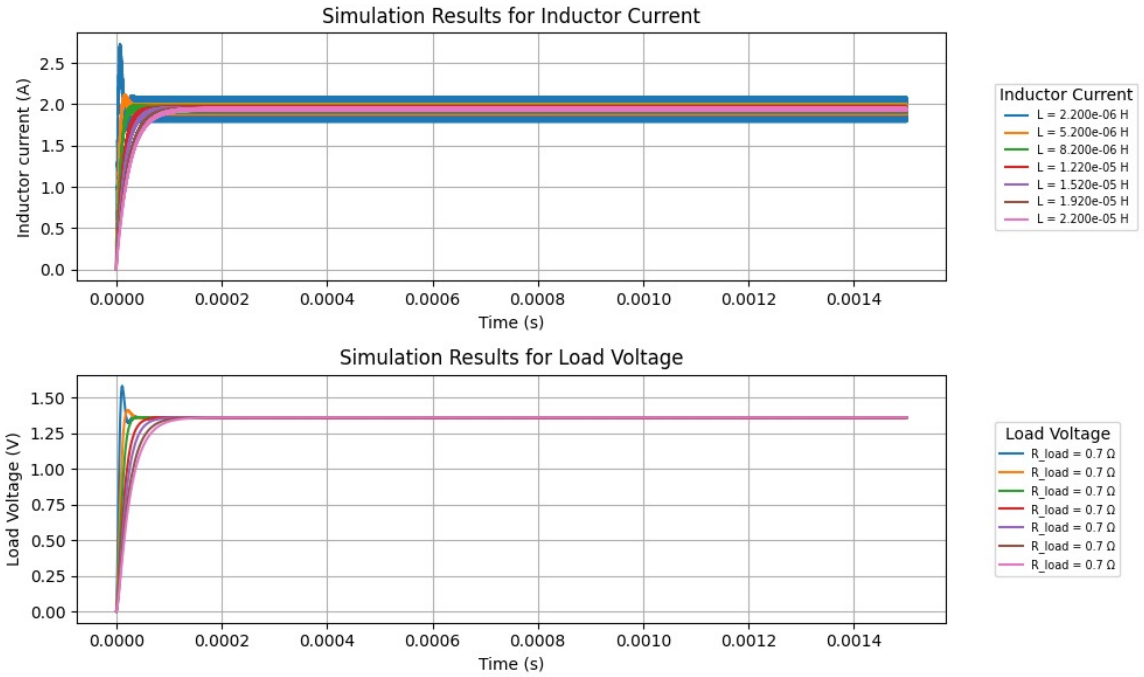


Figure 2.11: Transient simulation results showing inductor current (top) and load voltage (bottom) for different inductance values.

Overall, these results demonstrate how passive-component choices strongly influence the efficiency and dynamic behavior of an open-loop synchronous buck converter. These findings form the baseline for the closed-loop evaluation presented in Section 2.5.2.

2.5.2 Results of the High-Level Controlled Buck Converter

In addition to the open-loop parameter exploration, the automated framework was applied to a high-level controlled buck converter model representative of current-mode control (CMC). This abstraction captures the dynamic interaction between the power stage and the control loop, enabling automated evaluation of transient performance, regulation quality, and loop stability.

Figure 2.12 shows the output-voltage transient response under multiple load currents ranging from 0.6 A to 3.5 A. Across all operating points, the converter settles cleanly to the regulated 3.3 V level with minimal overshoot. The fast rise of the inductor current followed by controlled convergence reflects the characteristic behavior of peak CMC, confirming correct operation of the PWM comparator, slope compensation, and logic stages.

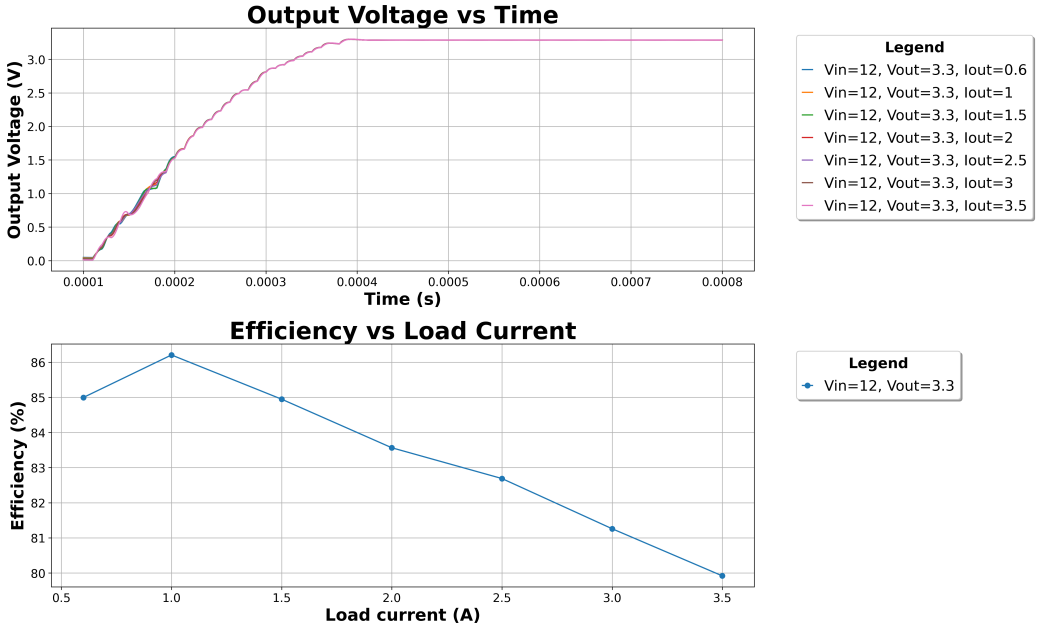


Figure 2.12: Transient output-voltage response and corresponding efficiency for various load currents in the controlled buck converter.

The efficiency profile across load currents is also included in Fig. 2.12. The converter achieves a peak efficiency of approximately 86% at a load current of 1 A, obtained with the optimized passive component values ($L = 2.2$ μH, $C = 44$ μF). Efficiency decreases gradually at heavier loads due to conduction losses and at lighter loads due to fixed switching losses, consistent with expected behavior in synchronous buck converters.

Loop stability for the controlled model was evaluated using automated POP (Periodic Operating Point) and AC analysis. Figure 2.13 illustrates the measured phase margin as the inductor value is varied. Although the phase margin decreases with increasing inductance, it remains above 57° for all tested values, exceeding the industry-standard 45° requirement for stable CMC operation. This confirms that the automated workflow reliably captures frequency-domain behavior for design verification.

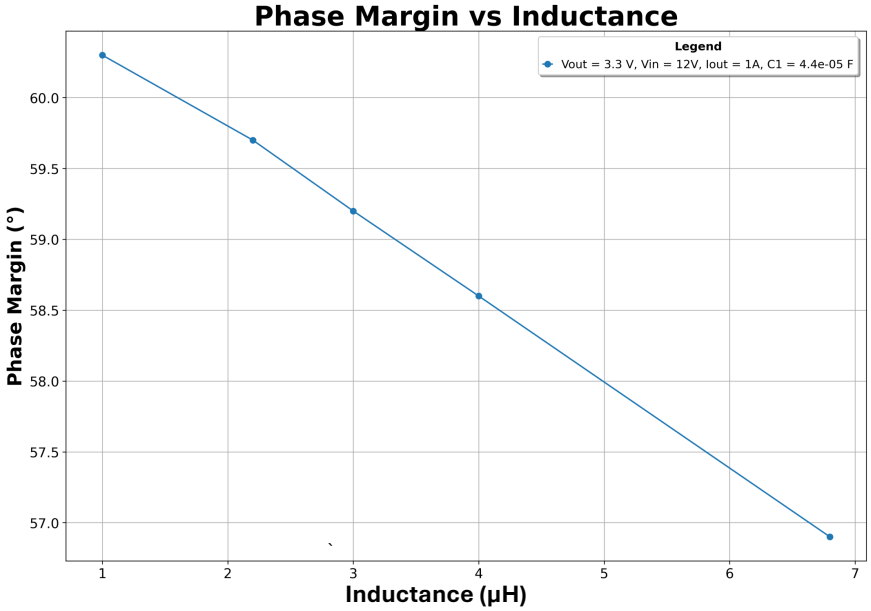


Figure 2.13: Phase margin of the controlled buck converter across inductance values using automated POP–AC analysis.

Further efficiency trends for multiple output-capacitance values are shown in Fig. 2.14. All curves exhibit a clear efficiency optimum around $L = 2.2\ \mu$H, while remaining relatively flat (85.5% $\pm$ 0.4%) for inductances between 2.2 μH and 5 μH. This demonstrates that the converter maintains robust efficiency even when component tolerances or BOM variations occur.

Overall, the high-level controlled results validate the automated framework's ability to assess transient response, stability margins, and efficiency trends without manual intervention. The combination of parameter sweeps, transient simulations, and POP–AC analysis provides a comprehensive foundation for evaluating converter performance across a wide design space.

Table 2.1: List of Parameters and Values Used in Automated Sweeps

Parameter	**Given**	**Swept Range**
f_{osc} (MHz)	2	2
V_{in} (V)	12	3.7, 12, 45
V_{out} (V)	3.3	3.3
I_{out} (A)	2	0.6, 1, 1.5, 2, 2.5, 3, 3.5
L_1 (μH)	2.2	1, 2.2, 3, 5, 6.6
C_1 (μF)	44	10, 22, 44, 100, 200

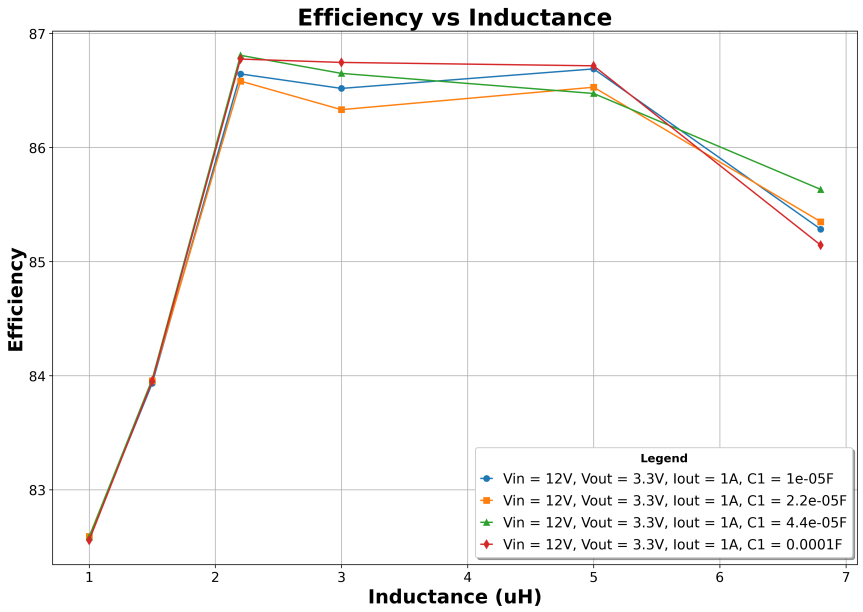


Figure 2.14: Efficiency variation with inductance for multiple capacitor values in the controlled buck converter.

Table 2.2: Comparison of Targeted and Achieved System Performance

Parameters	***Target***	***Achieved***
$Efficiency(\eta)(\%)$	83	85 @ 0.6A 86.2 – 80 @ 1A– 3.5A 86.2 @1A 83.5 @ 2A
$PhaseMargin(^\circ)$	45	60.5-56.8 @ L1 = 1.0 – 6.8 μH

2.6 Placement of Published Work and Transition to Schematic-Level Automation

Parts of the architecture-level automation framework presented in this chapter were previously published in our earlier work [18], where Python–SIMPLIS integration was used to construct an automated benchmarking and optimization environment for DC–DC converters. That publication demonstrated the feasibility and efficiency of automating parametric sweeps, transient analysis, and POP–AC stability evaluation for power converters. The present thesis extends that work significantly by embedding the same automation principles into a broader AMS design framework spanning schematic- and layout-level automation.

The methods introduced in Chapter 2 established:

- A parameterized behavioral model suitable for high-speed evaluation,
- An API-driven Python controller for netlist modification and simulation execution,
- Automated extraction of key metrics such as efficiency, ripple, transient response, and phase margin,
- Validation of both open-loop and high-level controlled buck converter abstractions.

These results confirm that architecture-level decisions—such as component sizing, operating points, and control behavior—can be explored efficiently without manual intervention. However, architecture-level evaluation alone is insufficient for full AMS design automation. Once a feasible design region is identified, the next step is to translate the architectural choices into detailed transistor-level schematics suitable for circuit-level validation and optimization.

Chapter 3 builds directly on this foundation. It introduces automated schematic generation methodologies used to construct analog building blocks from Python, interfacing with industrial CAD tools. Beginning with resistor-network generation and optimization, the chapter demonstrates how schematic structures, device parameters, and connectivity can all be produced automatically, establishing the second major layer of the proposed multi-level AMS automation framework.

Chapter 3

Automatic Optimization and Schematic Generation for Resistor Networks

3.1 Introduction to Automated Analog Block Generation

Analog and mixed-signal (AMS) design remains a labor-intensive process in which circuit optimization, schematic capture, and validation are typically handled as loosely connected steps. Designers often compute circuit parameters using calculations, spreadsheets, or scripts, and then manually implement schematics in electronic design automation (EDA) tools. This fragmented flow introduces inefficiencies, increases the likelihood of transcription errors, and prolongs the overall design cycle.

Chapters 1 and 2 highlighted that, unlike digital flows, AMS schematic and layout design still rely heavily on manual effort. Existing schematic-generation techniques [45–49] introduced rule-based or template-driven flows that reduce repetitive editing and allow partial automation. Later works [50–52] improved modularity through parameterized templates and hierarchical generator-based environments. More recent developments [53–55] explored reinforcement learning and procedural construction for block-level synthesis.

Despite these advances, several key limitations remain:

- optimization, variation analysis, and schematic generation are rarely integrated into a single coherent flow;
- many generators remain template-based without incorporating technology-aware sizing or mismatch-driven robustness checks;
- tight integration with industrial CAD environments is limited, leaving a gap between algorithmic results and schematic implementation.

To address these gaps, this chapter presents a Python-based framework that unifies optimization, variation-aware validation, and automatic schematic generation within a single flow. While the methodology is general, resistor networks are chosen as the benchmark demonstration because they are widely used in LED drivers, bias ladders, monitoring circuits, voltage references, and DAC arrays. Although conceptually simple, modern ICs often require networks using hundreds of unit resistors, making manual design inefficient, error-prone, and difficult to scale across process corners and specifications.

The framework introduced here efficiently determines resistor values, validates mismatch robustness, and automatically generates the corresponding schematic in an industrial CAD environment. This closes the gap between numerical optimization and schematic instantiation, forming the schematic-level automation pillar of the multi-level AMS design flow developed in this thesis.

It is worth noting that the use of resistor dividers is not isolated to stand-alone bias or monitoring networks. In Chapter 2, resistor-based sensing and feedback paths were introduced as part of the system-level design of switching power converters, where output regulation and threshold generation frequently rely on divider ratios rather than absolute resistance values. Motivated by the same need for fast re-targeting across specifications and corners, this chapter formalizes the resistor-network design task into an automated, technology-aware flow that bridges numerical optimization directly to CAD-ready schematic generation.

3.2 Design Principles and Use Cases of Resistor Networks

Resistor networks are fundamental building blocks in AMS systems. They are used to generate bias voltages, implement voltage dividers, realize ladder structures in DACs, and define thresholds for comparators in monitoring and protection circuits. In many such applications, the absolute accuracy of resistance values is less critical than the ratio between them, but process variation, mismatch, and layout-dependent effects can significantly affect performance.

In the context of this work, the target use case is a resistor network connected to a comparator input, used to generate programmable voltage thresholds. The network typically consists of three or more taps that define different divider ratios, while an additional resistor may serve as a bleed or current-limiting element. The main design objectives are:

- accurate DC voltage ratios across process and temperature.
- robustness against mismatch-induced variations.
- minimized layout area.

- compliance with technology constraints such as minimum width, length, and spacing.

Traditional design flows for such networks proceed as follows:

- Compute ideal resistance ratios based on the desired voltages.
- Select a unit resistor geometry and sheet resistance.
- Manually decompose each resistance into series and parallel combinations of unit devices.
- Implement the network in the schematic editor.
- Perform circuit verification (e.g., Monte Carlo) and iterate if constraints are not met.

This process is time-consuming and error-prone, especially when the number of units is large or when multiple design corners must be validated. Moreover, any change in system-level specifications can force designers to repeat most steps manually.

The framework introduced in this chapter addresses these challenges by:

- automatically determining feasible resistor values and unit decompositions.
- validating variation robustness through automated Monte Carlo analysis before schematic finalization.
- programmatically generating the resistor network in the CAD environment.
- and producing a final verification report summarizing DC, transient, and statistical results.

In this work, the entire optimization, schematic creation, and verification flow including Monte Carlo simulations—is automated through a Python-driven interface. Only the layout implementation is handled separately using a template-based approach for the resistor units.

3.3 Python-Based Framework Architecture

The proposed framework provides an automated, technology-aware flow for the design of resistive networks. It integrates circuit optimization, variation analysis, unit-device construction, and schematic generation within a single Python environment interfaced to an industrial CAD tool (e.g., Cadence Virtuoso) via an application programming interface.

Figure 3.1 illustrates the overall architecture and workflow, starting from user specifications and ending with an automatically generated schematic.

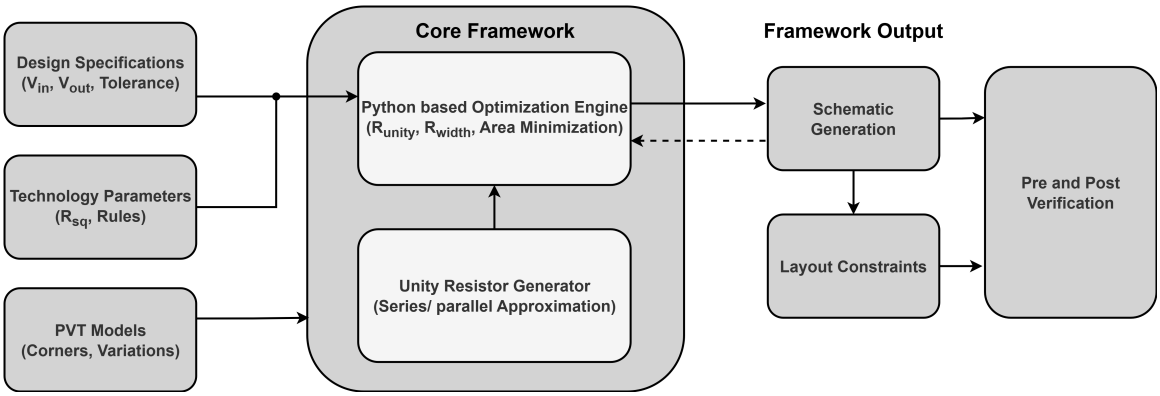


Figure 3.1: Overall architecture of the proposed resistor-network framework, showing the flow from user specifications to automatic schematic generation.

3.3.1 Workflow Overview

The framework begins with user-defined electrical and technological specifications that describe the circuit design objectives and applicable process constraints. Typical inputs include:

- target input and output voltage levels;
- maximum permissible voltage error and variation at each tap;
- sheet resistance, minimum geometry, and mismatch parameters of the resistor technology;
- optional bounds on total area or number of unit devices.

In Fig. 3.1: The framework processes these inputs through four main modules:

1. **Optimization Engine**, which determines feasible resistance values and selects the optimal configuration based on layout area, electrical accuracy, and mismatch constraints.
2. **Variation-Aware Validation**, which evaluates mismatch sensitivity using first-order uncertainty propagation models and filters out configurations that violate statistical robustness limits.
3. **Unit-Device Synthesis**, which converts ideal resistor values into realizable series–parallel combinations of unit resistors while respecting technology minimum dimensions.

4. **Automatic Schematic Generator**, which instantiates the validated configuration in the target CAD environment using a Python-to-EDA API (e.g., Python–SKILL), assigning device parameters and connectivity automatically.

Each module communicates through standardized data structures, allowing the flow to progress smoothly from numerical optimization to schematic creation without manual intervention. The modular design also enables reuse of the same framework for other passive-dominated AMS structures, such as DAC ladders and bias networks.

3.4 Deterministic Recursive Optimization of the Resistor Network

This section presents a deterministic and constraint-driven methodology for synthesizing a resistive divider network with minimum physical layout area. The proposed approach is fully parametric and expressed in symbolic form, ensuring applicability across technologies while preserving confidentiality.

Unlike brute-force enumeration or stochastic heuristics, the optimization framework exploits analytical relations between divider ratios, unit-resistor geometry, and physical realizability constraints. These relations are used to aggressively prune infeasible solutions, resulting in a bounded and predictable search process suitable for automation.

3.4.1 Problem Formulation

The design objective is to determine a physically realizable resistor network that minimizes total layout area while satisfying electrical feasibility and variation robustness constraints. This can be expressed as:

$$\begin{aligned} &\text{minimize} && A_{\text{layout}} \\ &\text{subject to} && |\Delta V_{\text{out}}| < \varepsilon_{\max}, \\ &&& \sigma_{V_{\text{out}}} < \sigma_{\text{limit}}, \end{aligned} \tag{3.1}$$

where A_{layout} is the estimated layout area, ΔV_{out} is the static output error, and $\sigma_{V_{\text{out}}}$ captures mismatch-driven output variation.

3.4.2 Electrical Target Formulation

The divider network is designed to generate accurate reference voltages in the presence of non-ideal parasitic resistances introduced by switches or interconnect. Let R_{par} denote the equivalent parasitic series resistance and let R_1 represent the primary divider resistor.

The electrical relationships are expressed symbolically in terms of voltage levels, leading to intermediate resistance targets:

$$R_2' = \frac{V_{\text{th}}(R_1 + R_{\text{par}})}{V_{\text{L}} - V_{\text{th}}}, \qquad R_3' = \frac{V_{\text{th}}(R_1 + R_{\text{par}})}{V_{\text{H}} - V_{\text{th}}}, \tag{3.2}$$

which are compactly denoted as

$$k_1 = R_2', \qquad k_2 = R_3'. \tag{3.3}$$

3.4.3 Configurable Compensation Resistor

To enable discrete, layout-aware optimization, a configurable compensation resistor R_c is introduced. The resistor is constructed from identical unit elements combined in series and parallel form:

$$R_c = N_{sc} \cdot R_{\text{unit}} + \frac{R_{\text{unit}}}{N_{pc}}, \tag{3.4}$$

where R_{unit} is the unit-resistor value and N_{sc} and N_{pc} denote the number of series and parallel units, respectively.

The effective divider resistances are then obtained as

$$R_2 = k_1 - R_c, \tag{3.5}$$

$$R_3 = \frac{R_2(k_2 - R_c)}{-k_2 + R_2 + R_c}, \tag{3.6}$$

subject to the physical realizability constraints

$$R_2 > 0, \qquad R_3 > 0. \tag{3.7}$$

Configurations violating these constraints are deterministically pruned during the search.

3.4.4 Area Model and Optimization Objective

Each resistor is implemented using identical unit devices with width W and sheet resistance R_{sq}. The corresponding unit length is

$$L_{\text{unit}} = \frac{R_{\text{unit}}}{R_{\text{sq}}} \cdot W. \tag{3.8}$$

The total number of unit resistors in the network is

$$N_{\text{tot}} = N_{R1} + N_{R2} + N_{R3} + N_{sc} + N_{pc}. \tag{3.9}$$

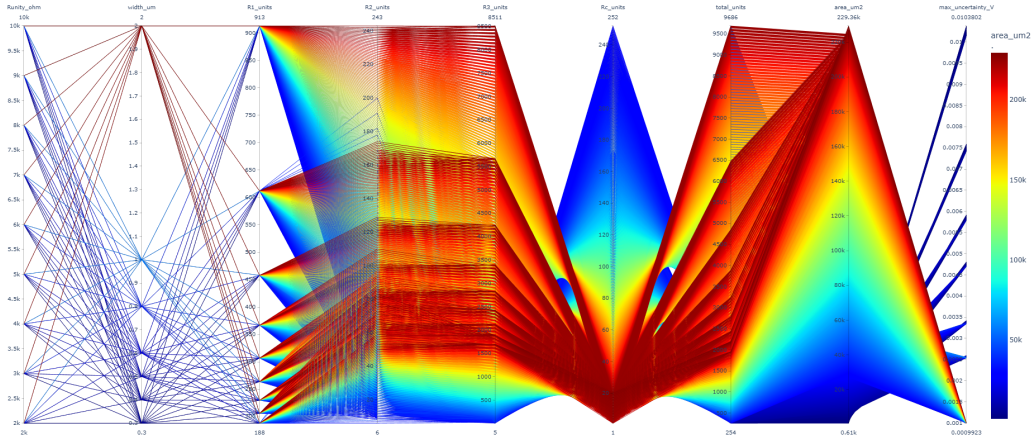


Figure 3.2: Parallel-coordinate visualization of feasible resistor-network configurations generated by the deterministic optimization process for total layout area.

Since each unit resistor length is determined by its aspect ratio as $L_{\text{unit}} = \frac{R_{\text{unit}}}{R_{\text{sq}}} \cdot W$, the area per unit becomes $W^2 \cdot \frac{R_{\text{unit}}}{R_{\text{sq}}}$, and the total layout area is therefore

$$A_{\text{total}} = N_{\text{tot}} \cdot W^2 \cdot \frac{R_{\text{unit}}}{R_{\text{sq}}}. \tag{3.10}$$

Figure 3.2 visualizes the feasible design space generated by the deterministic optimization process. The structured bands confirm that the search is guided by analytical constraints rather than brute force.

A high-level flow of the deterministic optimization and pruning process is shown in Fig. 3.3. Only electrically feasible and constraint-compliant solutions are stored and forwarded to the statistical validation stage.

3.5 Variation-Aware Validation

Once the optimization engine confirms electrical feasibility, each candidate configuration is evaluated for robustness against device mismatch. The goal of this stage is to guarantee that the divider meets a specified multi-sigma accuracy requirement without resorting to Monte Carlo simulations during the optimization stage.

3.5.1 Resistor Mismatch Model

Mismatch in a unit resistor follows the inverse-area relationship:

$$\sigma_{R_{\text{unit}}} = \frac{A_R}{\sqrt{W \cdot L_{\text{unit}}}} \tag{3.11}$$

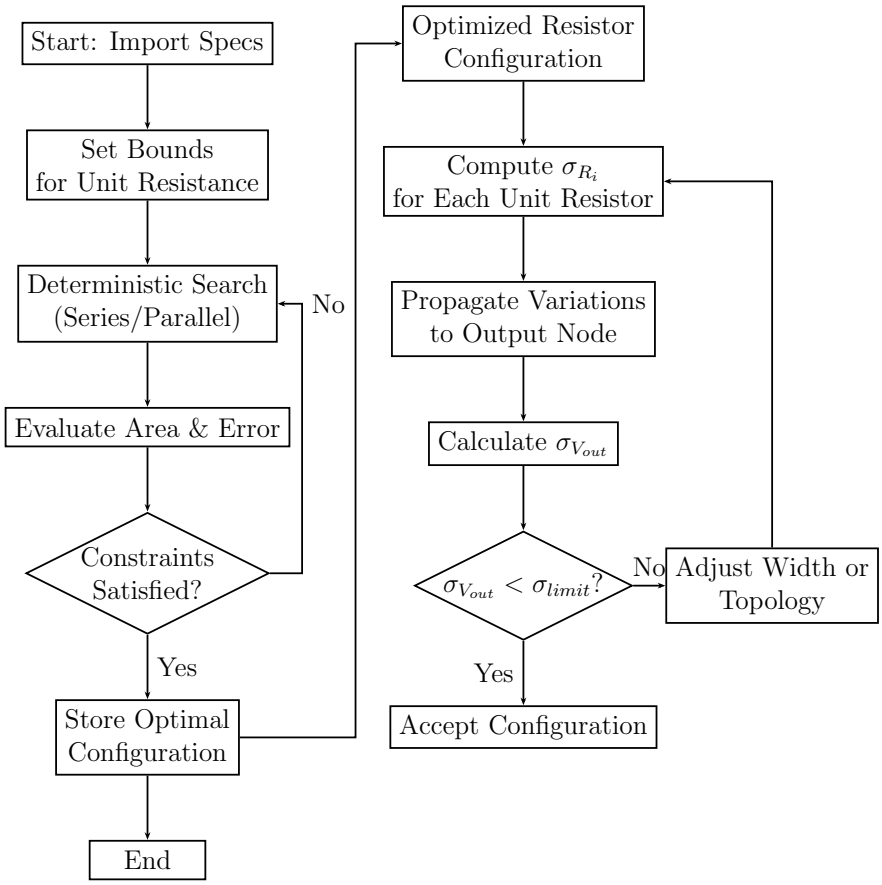


Figure 3.3: Deterministic optimization flow and variation validation flow.

indicating that increased physical area reduces statistical variability, where A_R is the technology-specific Pelgrom mismatch coefficient for the resistor flavour used in this work. Since $L_{\text{unit}} \propto W$ for a fixed $R_{\text{unit}}/R_{\text{sq}}$, increasing width increases unit area and typically improves mismatch performance.

3.5.2 Output Voltage Uncertainty Propagation

The fractional output-voltage uncertainty is estimated using first-order sensitivity analysis:

$$\frac{\sigma_V}{V} = \sqrt{\left(\frac{\partial V}{\partial R_1}\frac{\sigma_{R_1}}{V}\right)^2 + \left(\frac{\partial V}{\partial R_2}\frac{\sigma_{R_2}}{V}\right)^2}. \tag{3.12}$$

In practice, the worst-case fractional uncertainty across all relevant output nodes is considered.

3.5.3 Constraint Enforcement and Optimal Selection

Only configurations whose worst-case fractional uncertainty remains within the specified multi-sigma accuracy margin are considered statistically robust. Among these candidates, the framework selects the configuration with minimum total layout area.

Figure 3.4 illustrates the area–uncertainty trade-off, while Fig. 3.5 shows that tightening the accuracy requirement leads to a smooth and predictable contraction of the feasible design space.

It should be noted that the area values reported in Figure 3.4 represent the calculated resistor area excluding layout spacing and routing overhead. The total layout area, including spacing constraints, is summarized in Table 3.2.

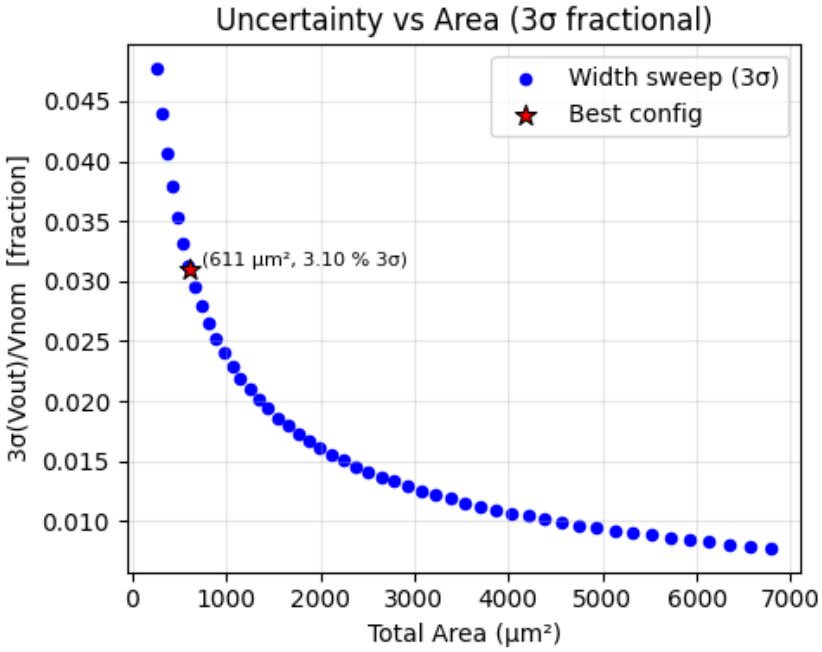


Figure 3.4: Trade-off between total resistor-network area and fractional output uncertainty. The highlighted point indicates the selected optimal configuration.

The smooth evolution of feasible solutions confirms that the proposed deterministic optimization framework remains stable and well behaved under changing statistical constraints.

3.6 Automated Schematic Construction Flow

The final stage of the flow translates the optimized resistor configuration into a design-ready schematic in the industrial CAD environment. This step is crucial for bridging

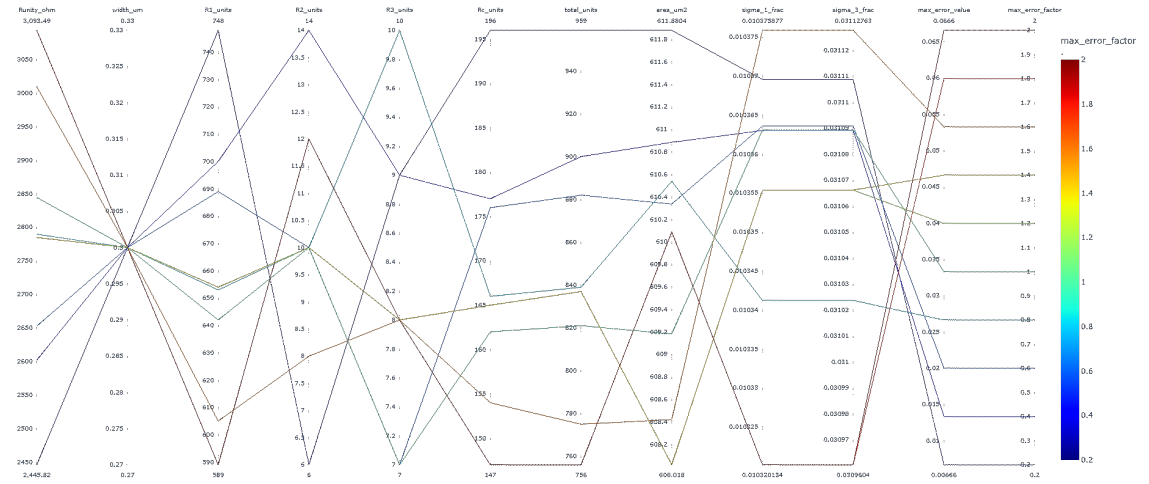


Figure 3.5: Sensitivity of the feasible design space under progressively tightened accuracy requirements. Color encodes the applied error bound.

algorithmic design and practical implementation, ensuring that no information is lost or corrupted between optimization and circuit creation.

3.6.1 Objectives and Design Philosophy

The schematic generator is designed with the following objectives:

- **Reproducibility**: identical optimization data always produces an identical schematic.
- **Consistency**: device naming, parameters, and connectivity directly correspond to optimization results.
- **Automation-readiness**: no manual editing of device parameters or net connections is required.
- **Design integrity**: the generated schematic remains compatible with simulation, LVS, and subsequent layout generation.

In the implementation used in this work, Python acts as the control layer and communicates with Cadence Virtuoso through a SKILL-based API to instantiate and parameterize devices, create nets, and connect pins according to the optimized network description.

3.6.2 Generation Workflow

The schematic generation flow operates as follows:

1. Parse the validated optimization output, which includes:
   - the chosen unit resistance R_{unity},
   - series–parallel decomposition strings for R_1, R_2, R_3, and R_C,
   - unit counts and geometry parameters,
   - connectivity information for the network and comparator interface.
2. Open a schematic template cellview that contains only high-level symbols (e.g., comparator, supply, and resistor placeholders).
3. Programmatically instantiate each resistor unit by duplicating a base device, assigning its resistance value and geometry, and connecting it into the appropriate series or parallel branch using SKILL functions.
4. Remove placeholder elements and verify that all required nets are connected and named correctly.
5. Save the finalized schematic cellview and trigger a netlisting step to enable immediate simulation.

Figure 3.6 illustrates the functional topology of the comparator-driven resistor network and the automatically generated schematic after unit expansion.

The comparator itself is not part of the optimization loop but provides a realistic context for integrating the resistor network into a mixed-signal system. The generated schematic is thus directly usable for DC, transient, and Monte Carlo simulations, as well as for layout handoff.

3.7 Schematic-Level Verification and Statistical Validation

Once the resistor network was generated automatically, the framework produced a simulation-ready testbench, allowing the designer to validate the electrical behavior of the network without manual netlisting or schematic editing. Nominal simulations confirmed that the generated divider ratios matched the analytical targets obtained from the optimization stage.

Statistical verification was performed using large-scale Monte Carlo analysis and multi-corner simulations (TT, FF, SS, FS, SF) in a single-click manner. The verification flow was fully automated: the framework configured the simulator, executed all statistical samples, and post-processed the data to extract the output voltage distribution at each tap. More than 200,000 Monte Carlo samples were evaluated across all defined specifications, with a 100% pass rate observed. Detailed statistical distributions and parameter values are omitted due to industrial confidentiality; however,

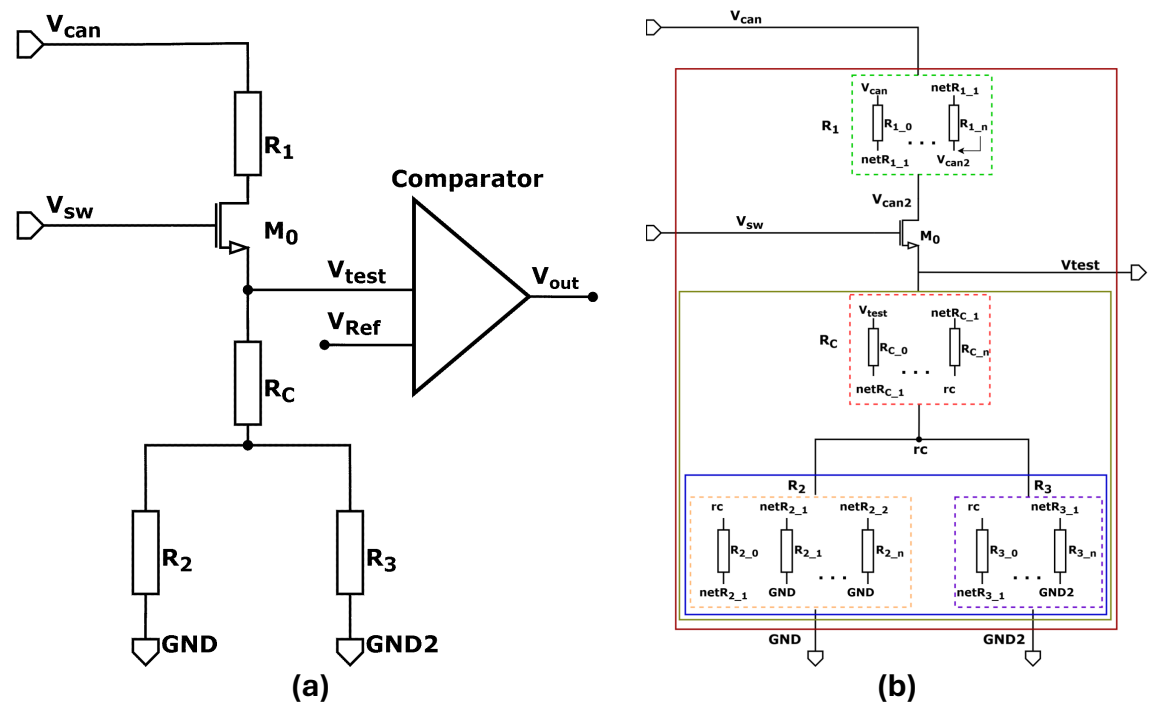


Figure 3.6: (a) Functional topology of the comparator-driven resistor network, and (b) automatically generated schematic with series–parallel unit-resistor expansions for R_1, R_2, R_3, and R_C.

the results confirm the robustness and correctness of the automatically generated schematic.

Table 3.1: Summary of Automated Schematic-Level Verification

Verification Type	**Passed / Total**	**Pass Rate**
Monte Carlo (Statistical)	>200K samples	100%
Process–Voltage–Temperature Corners	All evaluated	100%

The automated verification flow also generated a consolidated summary report containing nominal results, corner simulations, and statistical metrics. Although detailed numerical results and internal limits are not reported here due to confidentiality constraints, the overall verification confirmed that the optimized resistor network meets the design requirements across all simulated process and mismatch conditions.

3.8 Layout Implementation

Following schematic generation and verification, the optimized resistor network was implemented at the layout level using parameterized unit-resistor cells. The goal of this stage is to translate the series–parallel topology identified during optimization into a physically realizable and variation-robust layout while respecting technology design rules.

3.8.1 Layout Construction Strategy

The layout implementation follows a structured methodology designed to preserve matching, minimize parasitics, and ensure consistency with the schematic-level configuration:

- **Unit-Cell Replication**: Each resistor element is constructed using a single parameterized unit cell. The framework computes the required number of series and parallel units for R_1, R_2, R_3, and R_C, and these units are instantiated accordingly.
- **Symmetric Array Placement**: To improve matching and gradient immunity, the resistor units are arranged in structured arrays that enforce symmetry, equal spacing, and uniform orientation. This is particularly important for achieving predictable variation behavior at the tap nodes.
- **Contact and Routing Optimization**: Shared contact regions, merged diffusion segments, and minimal-length interconnect routing are used to reduce parasitic resistance and area. Routing is performed with uniform metal layers to maintain consistency across series and parallel segments.
- **Design Rule Compliance**: All device geometries follow minimum width, length, and spacing rules. Parallel branches are placed with mirrored orientation to minimize systematic mismatch.

A representative implementation of the automatically generated layout is shown in Fig. 3.7. The highlighted section corresponds to the optimized resistor array, while the comparator cell is placed alongside to illustrate complete block integration.

3.8.2 Layout Consistency and Physical Validation

After completing the layout, a design rule check (DRC) and layout-versus-schematic (LVS) check were performed. The extracted resistor values matched the schematic values with less than 1% deviation, confirming:

- correct translation of series–parallel topology,

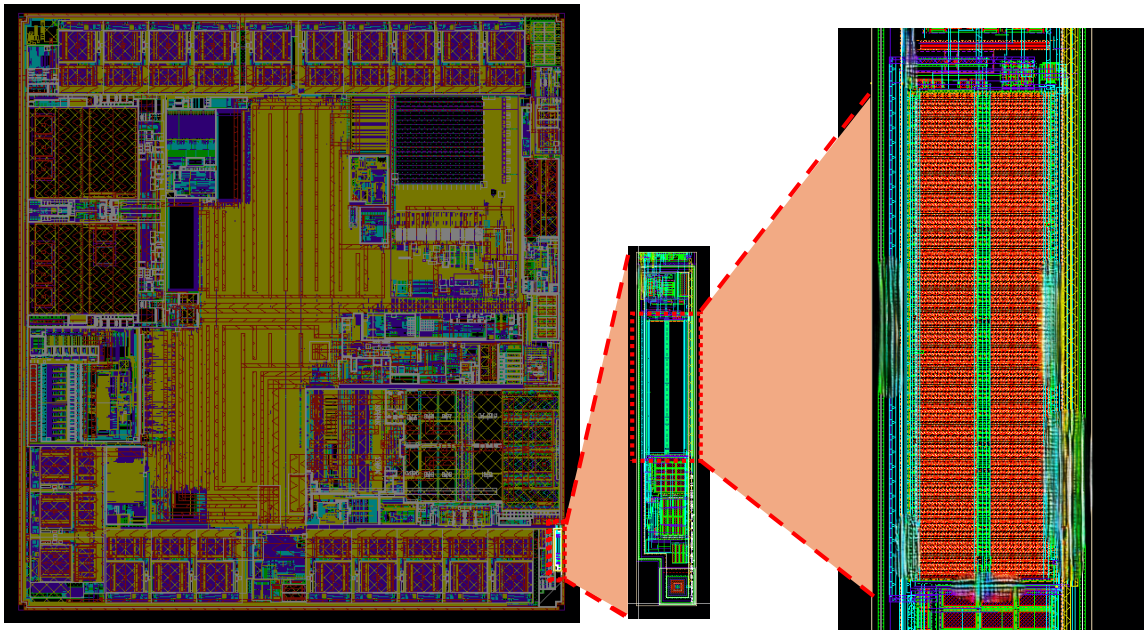


Figure 3.7: Automatically generated layout of the resistor network with an integrated comparator. The highlighted region corresponds to the optimized resistor array.

- proper device sizing and orientation,

The layout validation confirms that the optimization and schematic generation framework is physically realizable and maintains its accuracy.

Table 3.2: Summary of Optimization and Layout Results for the Resistor Network

Parameter	**Value**
R_{unity}	2.8 kΩ
Total Area	2400 μm^2
Total Unit Count	850
Voltage Error ΔV_{out}	$< 1\%$
Output Variation $\sigma_{V_{\text{out}}}$	$< 3.34\%$ (3σ)

3.9 Summary

This chapter presented a complete and fully automated framework for the design and generation of resistor networks used in analog and mixed-signal systems. The flow integrates optimization, mismatch-aware validation, schematic construction, and ver-

ification within a unified Python–CAD environment. The key contributions demonstrated in this chapter are summarized below:

- A **deterministic recursive optimization algorithm** was developed to identify the optimal unit resistance and series–parallel decomposition while minimizing layout area and satisfying voltage-accuracy constraints.
- A **variation-aware validation stage** was incorporated using first-order mismatch propagation. This ensures that only statistically robust configurations proceed to schematic generation, eliminating the need for repeated manual checks.
- A **fully automated schematic-generation engine** was implemented, where Python communicates with the CAD environment to instantiate and connect hundreds of unit resistors, ensuring consistency, correctness, and reproducibility across design iterations.
- A **schematic-level verification flow** was executed, including transient simulations, DC checks, and statistical Monte Carlo analysis across all corners to confirm that the automatically generated design satisfies performance and robustness requirements.
- A **template-based layout methodology** was introduced to construct the physical implementation of the optimized network. The resulting layout preserves symmetry and matching requirements while remaining consistent with the schematic-level configuration.

Overall, this chapter demonstrates that the entire resistor-network design process—from numerical specification to schematic creation and verification—can be automated in a structured and scalable manner. Although resistor networks are relatively simple circuits, they serve as an ideal benchmark for validating core automation principles such as hierarchical generation, deterministic optimization, statistical evaluation, and CAD tool integration.

The next chapter extends these ideas to a significantly more complex mixed-signal system: the successive-approximation-register (SAR) analog-to-digital converter. Chapter 4 will show how the same automation philosophy scales from passive building blocks to complete data-converter architectures, enabling automated construction of capacitive DAC arrays, sampling networks, comparator blocks, and hierarchical top-level schematics.

Chapter 4

Hierarchical Automatic Schematic Generation for SAR ADC

4.1 Introduction to SAR ADC

Successive Approximation Register (SAR) analog-to-digital converters have become the preferred architecture for low-power and medium-resolution applications such as IoT sensor interfaces, biomedical instrumentation, and battery-powered mixed-signal systems. Their widespread adoption is driven by excellent energy efficiency, the absence of power-hungry operational amplifiers, and strong compatibility with advanced CMOS technologies due to their predominantly digital control structure.

A SAR ADC operates by iteratively resolving the digital output code through a binary search process. The architecture is composed of a sampling network that acquires the analog input signal, a capacitive digital-to-analog converter (CDAC) that generates binary-weighted reference levels, a comparator that performs bit-wise decisions, and a successive approximation register (SAR) logic block that controls the conversion sequence and updates the CDAC configuration.

From an automation perspective, SAR ADCs are particularly attractive because their architecture is highly regular and hierarchical. The CDAC is constructed from repetitive capacitor arrays, the sampling and switching networks follow rule-based connectivity rules, the comparator is typically instantiated from a fixed template, and the SAR logic can be integrated as a reusable digital block. These characteristics make SAR ADCs well suited for rule-based and template-based schematic generation.

The selection of SAR ADC as the benchmark architecture in this work is driven by industrial requirements. The framework was developed in collaboration with Infineon Technologies, where SAR ADCs represent a key circuit class in the product portfolio for low-power mixed-signal applications. This industrial context motivated the choice of SAR ADC as the target architecture, ensuring that the proposed automation methodology addresses a practically relevant and commercially deployed circuit topology.

In this work, a Python-driven automation framework — implemented using the AnaGen schematic generation environment integrated with Cadence Virtuoso — is developed to automatically generate parameterised SAR ADC schematics based on high-level design requirements. Given a set of user-defined specifications, the framework instantiates the required sub-blocks, configures the CDAC array, connects the sampling and switching network, integrates the comparator and SAR logic, and assembles a complete hierarchical SAR ADC schematic suitable for further verification or layout generation. The framework evolved through three distinct development stages: an initial single-ended implementation, a technology-ported variant for a different channel configuration, and a final differential implementation integrating all blocks into a complete hierarchical SAR ADC core.

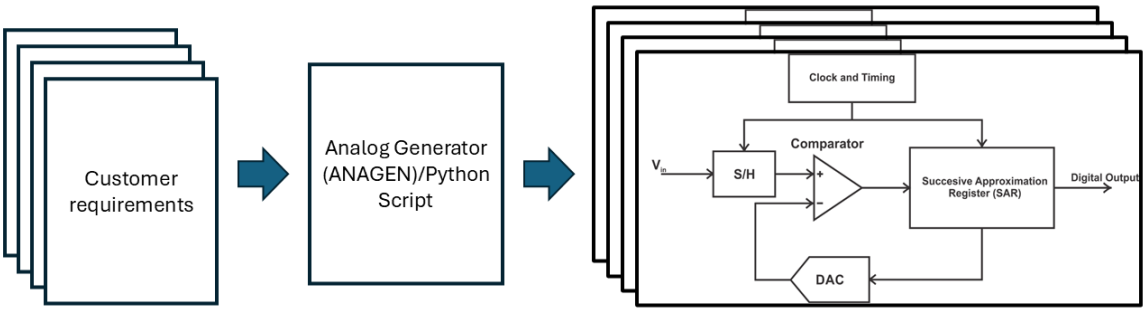


Figure 4.1: High-level SAR ADC automation concept.

4.2 Decomposition of the SAR ADC into Functional Blocks

To enable systematic and scalable schematic automation, the SAR ADC architecture is decomposed into a set of modular functional blocks with well-defined electrical and hierarchical interfaces. This decomposition is a key step in transforming a traditionally hand-crafted mixed-signal circuit into an automation-friendly design.

From an implementation perspective, each block is designed to be instantiated, parameterised, and connected independently through Python-based control logic. The overall SAR ADC schematic is then assembled hierarchically by integrating these sub-blocks according to rule-based connectivity rules. Figure 4.2 illustrates the block-level structure used in this work.

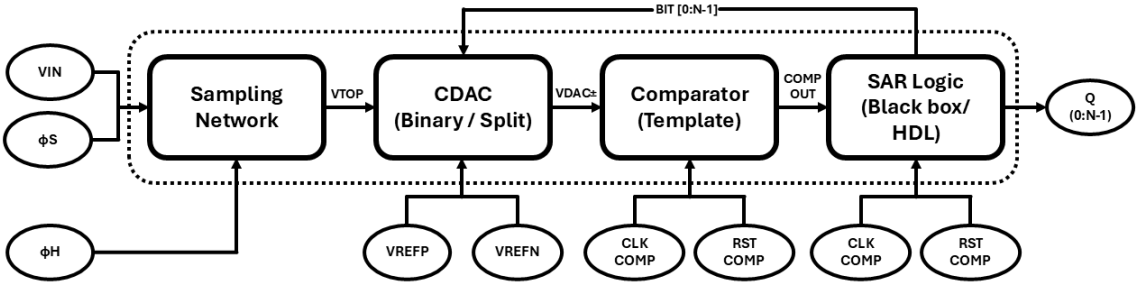


Figure 4.2: Functional decomposition of the SAR ADC architecture for schematic automation.

Table 4.1: Summary of automation modules for SAR ADC schematic generation.

Module	**Block Type**	**Key Contribution**
Sampling Capacitor Array	Sampling Network	Automated capacitor array with dummy and error cells
LV Switch Generator	Low Voltage Switch	LV CMOS switch generation with channel routing
MV Switch Generator	Medium Voltage Switch	MV switch generation with shared-node routing
MV NMOS Switch Generator	MV NMOS Switch	NMOS MV switch with ready signal handling
HV Switch Generator	High Voltage Switch	HV switch generation for high voltage domain
Input Multiplexer	Input Multiplexer	Multi-channel analog mux with symbol generation
Level Shifter Schematic	Level Shifter	Technology migration via automated pin/bus renaming
Level Shifter Symbol	Level Shifter Symbol	Automated symbol pin renaming for ported designs
Differential Sampling Network	Differential Sampling	Evolution: single-ended to differential architecture
Top-Level ADC Core Assembler	Top-Level ADC Core	Hierarchical assembly of complete SAR ADC core
Bus Name Generator	Utility Module	Non-contiguous bus name generation algorithm
Symbol Pin Modifier	Symbol Modifier	Dynamic symbol pin addition, removal, and renaming

The complete list of automation modules developed in this work is summarised in Table 4.1.

The following subsections describe the role of each block from an automation perspective rather than a detailed circuit-theoretic viewpoint.

4.2.1 Sampling Network

The sampling network is responsible for acquiring the input signal and transferring it onto the CDAC during the sampling phase. In typical SAR ADC implementations, this block consists of transmission-gate or bootstrapped switches driven by non-overlapping clock signals.

From an automation standpoint, the sampling network exhibits a predictable structure. Switch devices follow fixed connectivity patterns, clock signals are globally distributed, and node naming conventions (e.g., input, top-plate, bottom-plate) can be standardised. These properties allow the sampling network to be instantiated from a parameterised template, with switch sizing and clock polarity defined through high-level inputs.

4.2.2 Capacitive Digital-to-Analog Converter (CDAC)

The CDAC is the most structurally repetitive block in the SAR ADC and therefore the primary target for automation. It is typically implemented as a binary-weighted or split-capacitor array connected to a reference-switching network.

In the proposed framework, CDAC automation includes algorithmic computation of capacitor values, systematic instantiation of unit capacitors, assignment of bit-indexed nodes, and systematic wiring of the switching matrix. The regular geometry and hierarchical nature of the CDAC enable its construction entirely through rule-based generation without manual intervention.

4.2.3 Comparator

The comparator performs the decision-making operation during each conversion step. Although various comparator architectures exist, the schematic structure is typically fixed once a design is selected.

For automation purposes, the comparator is treated as a reusable schematic template. The automation framework instantiates the comparator cell, connects its differential inputs to the CDAC outputs, and binds the required clock and control signals. Optional parameterisation of device sizes or bias conditions can be supported without modifying the core topology.

4.2.4 Successive Approximation Register (SAR) Logic

The SAR logic implements the digital binary-search algorithm that controls the conversion process. In practice, this block is often provided as a digital macro, behavioral model, or HDL-based component.

Within the automation flow, the SAR logic is integrated as a black-box module with a well-defined interface. The Python controller binds the comparator output, clock and reset signals, and bit-control outputs to the corresponding CDAC switches. This clear separation between analog and digital domains simplifies integration and improves reusability.

4.2.5 Hierarchical Assembly

Once all functional blocks are generated, the automation framework creates the top-level SAR ADC schematic by hierarchically assembling the sampling network, CDAC, comparator, and SAR logic. Connectivity is established using predefined interface rules, ensuring consistent naming, correct signal polarity, and verification-ready structure.

This block-wise decomposition enables flexible configuration of SAR ADC variants while maintaining a consistent schematic-generation flow, forming the foundation for the automation framework introduced in the next section.

4.3 Automation Pipeline for SAR ADC Schematic Generation

Based on the block decomposition introduced in Section 4.2, a unified automation pipeline was developed to generate complete SAR ADC schematics in a systematic and configurable manner. The generator is implemented in Python and translates a compact set of user-defined specifications into a fully connected, hierarchical SAR ADC schematic suitable for simulation and downstream integration.

A key design principle of the proposed approach is **hierarchical generation**: instead of directly rewriting transistor-level connectivity for each new variant, the Python controller orchestrates the instantiation of reusable block templates and applies rule-based interconnection rules through standardised interfaces. This abstraction improves reproducibility, reduces integration errors, and enables rapid reconfiguration across SAR ADC variants.

Figure 4.3 summarises the high-level automation pipeline and its corresponding hierarchical output structure.

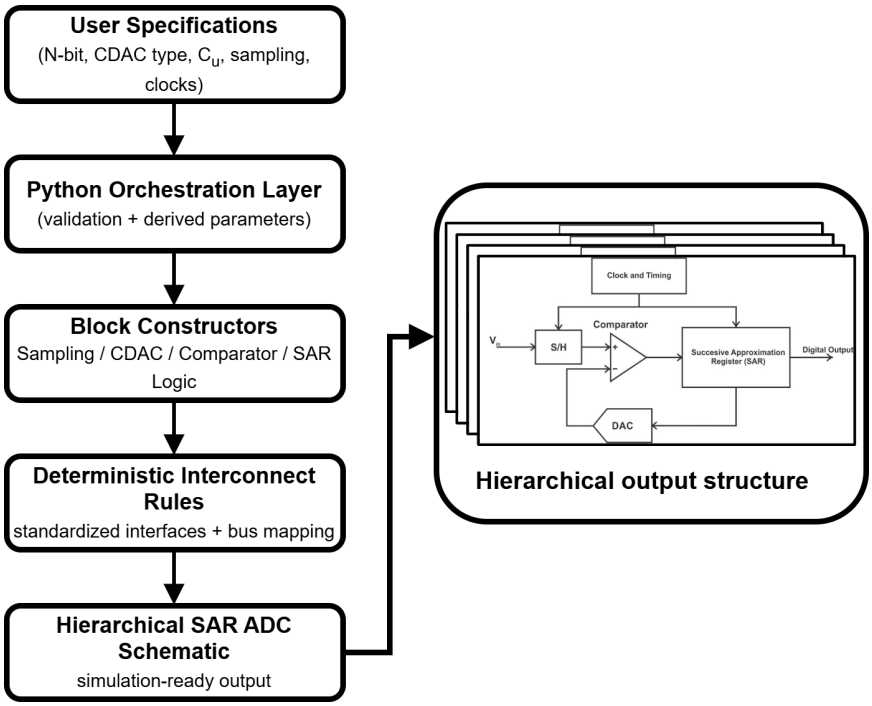


Figure 4.3: Python-based automation pipeline illustrating the transformation from user specifications to a hierarchical SAR ADC schematic and its generated output structure.

The automation pipeline is organised into three core stages.

- Specification Processing
- Block Construction Through Python Modules
- Hierarchical Schematic Synthesis and Output

4.3.1 Specification Processing

The generation process begins from a user-defined configuration that specifies the SAR ADC resolution, CDAC architecture options, unit capacitor sizing, sampling method, reference rails, clocking scheme, and the interface used for SAR logic integration. During this stage, the Python controller validates the specification set, computes all derived parameters, and prepares an internal block-level representation of the design. Typical derived values include the binary-weighted capacitor list, bus widths (e.g., `BIT[0:N-1]`), and the required clock/reset distribution signals.

This stage ensures that architectural modifications — such as changing resolution or switching between CDAC variants — propagate consistently through the entire design without manual intervention.

The complete set of user-defined input parameters accepted by the generation framework is summarized in Table 4.2. These parameters serve as the sole entry point to the automation pipeline: all derived quantities, block configurations, and hierarchical connectivity rules are computed internally by the Python controller without further manual input.

Table 4.2: User-defined input specifications for the SAR ADC schematic generation framework.

Parameter	**Description**	**Example Value**
`channel_map`	Channel-to-output-node assignment map	`ch0:0, ch1:1, ch2:1, ...`
`voltage domain`	Switch voltage domain selection	`Low, Medium, High`
`bus width`	ADC resolution, determines bus indices	`8`
`library name`	Target Cadence Virtuoso library	`SAR lib`
`c16_count`	Number of standard (LV) capacitor units	`4`
`c122_count`	Number of medium-voltage capacitor units	`4`
`differential mode`	Selects single-ended or differential path	`True | False`

From these inputs, the Python controller derives all secondary parameters required for block construction. Specifically, `bus_width` determines the bit-indexed node names and bus ranges (e.g., `BIT[0:N-1]`), `channel_map` controls the number of switch instances and output node merging, and `differential_mode` triggers duplication of all signal paths and symbol pin pairs in the subsequent block construction stage. This single-entry specification model ensures that architectural modifications such as changing the ADC resolution or switching between voltage domains propagate automatically through the entire generation pipeline without any manual schematic intervention.

4.3.2 Block Construction Through Python Modules

In the second stage, each SAR ADC functional block is generated using a dedicated Python module that encapsulates a reusable schematic template and a set of connection rules. The generator operates primarily at the **block and hierarchy level**: templates are instantiated and parameterised, while block interfaces remain stable.

Concretely, this stage includes (i) algorithmic CDAC instantiation from the derived capacitor list, including bit-indexed node assignment and switch-matrix wiring;

(ii) sampling-network instantiation with configurable switch sizing and clock binding; (iii) comparator integration using a fixed template cell with standardised clock/reset pins; and (iv) SAR logic integration as a black-box/HDL-based module whose outputs are bound to the CDAC control signals.

Because block construction follows rule-based procedures, the generated schematic is reproducible and does not depend on manual wiring choices. The top-level generation pseudocode is presented in Algorithm 1.

Algorithm 1 Top-Level SAR ADC Generation Pipeline

Require: *channel_map, voltage_domain, switch_type, bus_width, library_name*
Ensure: Complete SAR ADC schematic in Cadence Virtuoso

Stage 1: Specification Processing
Validate input specifications
$bus\_width \leftarrow$ `len`($channel\_map$)
$pin\_names \leftarrow$ `GenerateBusNames`($channel\_map$)
$node\_count \leftarrow$ `CountUniqueOutputNodes`($channel\_map$)
Open schematic editor in target library

Stage 2: Block Construction
`GenerateSamplingCapacitor`($c16\_count$, $c122\_count$)
`GenerateCDAC`($resolution$, $unit\_cap$)
`GenerateSwitchBlock`($channel\_map$, $voltage\_domain$)
if $voltage\_domain$ = Low Voltage **then**
    `GenerateLVSwitch()`
else if $voltage\_domain$ = Medium Voltage **then**
    `GenerateMVSwitch()`
else if $voltage\_domain$ = High Voltage **then**
    `GenerateHVSwitch()`
end if
`GenerateInputMux`($channel\_map$)
`GenerateLevelShifter`(old_bus, new_bus)
if $differential\_mode$ = True **then**
    `GenerateDifferentialVariant()`
end if

Stage 3: Hierarchical Assembly
Connect sub-blocks via rule-based wiring procedures
Rename nets and pins to match bus specifications
Perform connectivity verification
Generate and save symbol view

4.3.3 Hierarchical Schematic Synthesis and Output

In the final stage, the Python controller assembles the full SAR ADC by composing all generated sub-blocks into a top-level schematic. This step programmatically creates top-level nets and buses, connects block interfaces according to predefined mapping rules, and generates the required I/O pins for simulation and integration. Consistency checks are performed to prevent missing connections, duplicated nets, or interface mismatches.

The final output is a fully hierarchical SAR ADC schematic that is **simulation-ready** and **reproducible by a single execution** of the generator. Design variants — such as different resolutions or CDAC configurations — are produced through specification changes rather than manual schematic edits, enabling fast iteration and consistent generation across projects.

4.4 Block-Level Schematic Generation

This section details the automated generation procedure for each functional block, derived from the Python modules developed within the AnaGen framework.

4.4.1 Sampling Capacitor Network

The sampling capacitor network is the most structurally repetitive block in the SAR ADC. It consists of two types of unit capacitor arrays — standard low-voltage units and medium-voltage units — along with dummy, error, and end-rail cells. The generation algorithm iterates over the defined array sizes and places each cell at a computed grid position, drawing wires to connect the analog input, virtual ground, and current output pins automatically. The generation procedure is summarised in Algorithm 2.

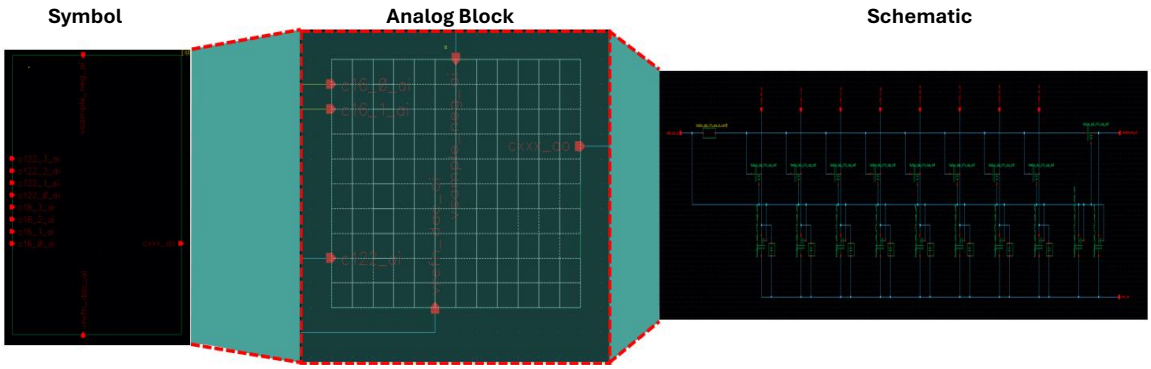


Figure 4.4: Automatically generated sampling capacitor network — symbol (left), analog block view (centre), and schematic (right) — produced by the AnaGen-based Python framework.

Algorithm 2 Sampling Capacitor Array Generation

Require: $c16\_count = 4$ *(standard capacitor units)*, $c122\_count = 4$ *(medium voltage capacitor units)*

Ensure: Connected capacitor array schematic with symbol

Declare pins: `vrefn_dac_ai`, `vsample_neg_ai`, `cxxx_ao`
Instantiate analog-to-current converter block
Connect `vrefn_dac_ai` to converter input

for $i = 0$ **to** $c16\_count - 1$ **do**
    Instantiate `cap_unit` at grid $(x,\ y)$
    Instantiate `cap_unit_err` at $(x + 0.5,\ y)$
    Instantiate `cap_unit_dummy` at $(x - 0.5,\ y + 2)$
    Declare input pin: `c16_{i}_ai`
    Connect `c16_{i}_ai` $\rightarrow$ `cap_unit.ana_ai`
    Draw wire: `cap_unit.vagnd_ai` $\rightarrow$ virtual ground
    Draw wire: `cap_unit.cur_ao` $\rightarrow$ current bus
    Connect `cap_unit_err` in parallel branch
    $x \leftarrow x + 2$
end for

for $i = 0$ **to** $c122\_count - 1$ **do**
    Instantiate `mv_cap_unit` at grid $(x,\ y)$
    Instantiate `mv_cap_unit_err` and `mv_dummy`
    Declare input pin: `c122_{i}_ai`
    Connect `c122_{i}_ai` $\rightarrow$ `mv_cap_unit.ana_ai`
    $x \leftarrow x + 2$
end for

Instantiate `end_rail`, `split_cap`, dummy termination cells
Connect `vsample_neg_ai` to final dummy cell
Connect `cxxx_ao` output pin
Create block symbol with pin groups (W, N, S, E)

4.4.2 Multi-Voltage Switch Block Generation

The switch generation framework supports three voltage domains: low voltage, medium voltage, and high voltage. All three domains use the same structural generation algorithm but instantiate different technology cells from the toolbox library. The channel routing logic handles both new output node creation and shared-node merging based on a user-defined channel connectivity map, as described in Algorithm 3.

Algorithm 3 Multi-Voltage Switch Block Generation

Require: $channel\_map = \{ch0 : 0,\ ch1 : 1,\ ch2 : 1,\ ch3 : 0,\ ...\}$, $voltage\_domain \in$ {Low, Medium, High Voltage}
Ensure: Parameterised switch schematic with routed outputs

Declare power pins: `VDD`, `VSS`
Declare bus pins: `sample_ch_p_i<N:0>`, `sample_ch_n_i<N:0>`
if *voltage_domain* = High Voltage **then**
    Declare HV analog input pins
end if
if *voltage_domain* = Medium Voltage **then**
    Declare `ready_n` control signal
end if

created_signals ← [ ]
output_nodes ← [ ]

for $i = 0$ **to** $switch\_count - 1$ **do**
    channel_id ← *channel_map*[*ch_i*]
    Instantiate `switch_cell` at position $(x,\ y)$
    Connect `sample_ch_p_i<i>` → `switch.select_p`
    Connect `sample_ch_n_i<i>` → `switch.select_n`
    Connect analog input pin → `switch.ainp_ai`
    Connect `VDD`, `VSS` supply pins
    if *channel_id* ∉ *created_signals* **then**
        Create new output pin: `sw_{N}_ao`
        Connect `switch.sw_out` → new output pin
        Append *channel_id* to *created_signals*
    else
        Route `switch.sw_out` → existing output wire
        $y \leftarrow y - 2$
    end if
end for

Generate block symbol:
    `PinGroup(VDD→North)`, `PinGroup(VSS→South)`
    `PinGroup(inputs→West)`, `PinGroup(outputs→East)`

4.4.3 Input Multiplexer Generation

The input multiplexer aggregates all voltage-domain switch outputs into a single configurable analog input block. Its schematic is generated by instantiating the switch

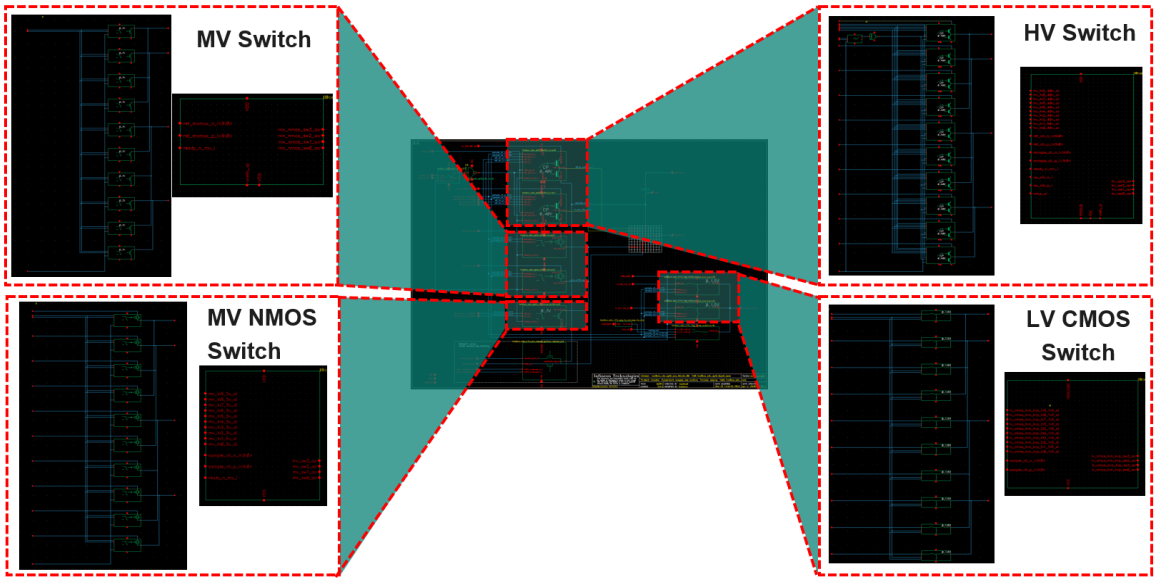


Figure 4.5: Automatically generated switch block schematics and symbols for all voltage domains.

sub-blocks for each voltage domain and connecting their outputs through a standardised interface. The corresponding symbol exposes all channel input pins grouped by voltage domain, enabling clean hierarchical integration into the top-level ADC core.

4.4.4 Technology Migration via Automated Pin and Net Renaming

A critical component of the framework is the ability to automatically migrate existing schematic templates to new channel configurations without recreating the design from scratch. This is achieved through automated pin renaming at both schematic and symbol levels, as well as net renaming for internal signal routing. Two implementation strategies were developed: generation from scratch and template-based migration (Algorithm 4).

4.4.5 Intelligent Bus Name Generation

A key utility module developed during this work is the bus name generation algorithm, which handles non-contiguous pin index sets. In standard EDA tools, bus names imply a contiguous range. However, when specific channel indices are selected from a larger pool, the bus representation must encode non-contiguous ranges. Algorithm 5 handles this automatically and produces valid EDA tool bus strings.

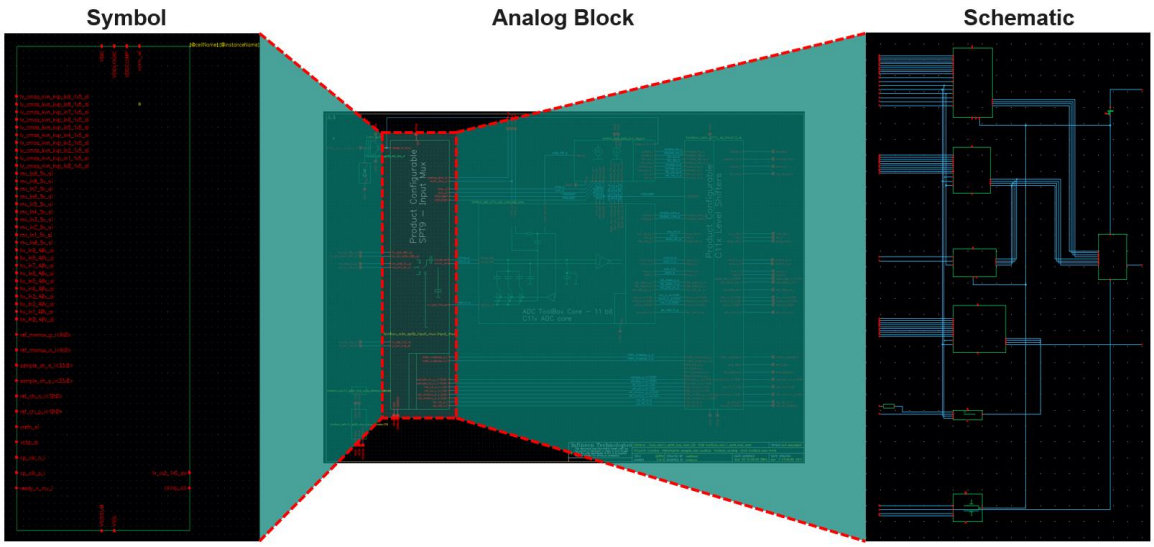


Figure 4.6: Automatically generated input multiplexer — symbol (left), analog block view (centre), and detailed schematic (right) — integrating all voltage-domain switch blocks through AnaGen.

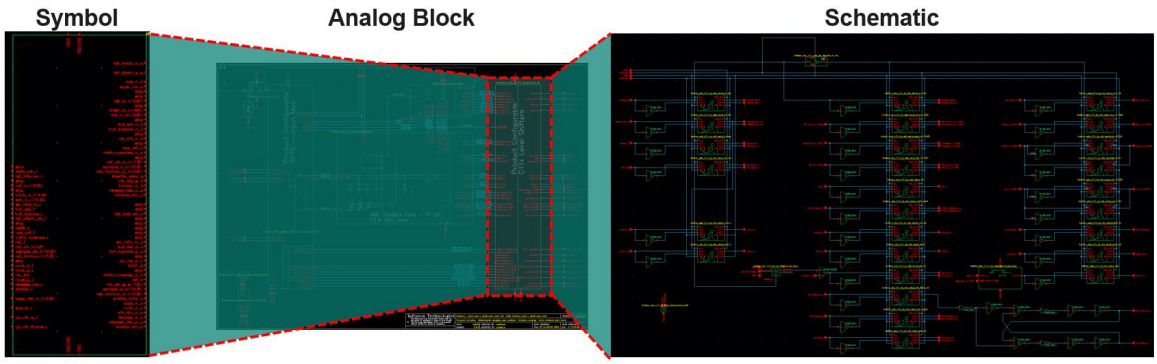


Figure 4.7: Level shifter block generated from scratch — symbol (left), analog block context (centre), and automatically generated schematic (right).

4.5 Framework Evolution: Single-Ended to Differential Implementation

The schematic generation framework was developed through a structured three-stage evolution. In the first stage, single-ended implementations were created for each switch voltage domain individually. In the second stage, the framework was migrated to a new channel configuration using the automated pin and net renaming procedures

Algorithm 4 Automated Pin Renaming for Technology Migration

Require: $old\_config = \{pins : [\ldots old\ bus\ names \ldots]\}$, $new\_config = \{pins : [\ldots new\ bus\ names \ldots]\}$

Ensure: Updated schematic and symbol with new bus widths

Step 1: Schematic Pin Renaming

for all (*old_pin*, *new_pin*) pairs **do**
    Locate terminal by name in schematic cellview
    Extract: position, orientation, pin direction
    Delete old pin figure from cellview
    Rename associated wire label
    Create new pin at same position with new name
    Run schematic integrity check (`schCheck`)
end for

Step 2: Internal Net Renaming

for all (*old_net*, *new_net*) pairs **do**
    Rename all wire segments: $old\_net \rightarrow new\_net$
end for

Step 3: Instance Renaming

for all (*old_instance*, *new_instance*) pairs **do**
    Update instance name to reflect new bus index
end for

Step 4: Symbol Pin Renaming

for all (*old_pin*, *new_pin*) pairs **do**
    Open symbol cellview in append mode
    Find terminal by old name in symbol
    Update label text in all pin figures
    Run symbol integrity check (`schVIC`)
    Save and close symbol cellview
end for

Step 5: Finalisation

Save and verify complete updated schematic

described in Section 4.4. In the third stage, a differential implementation was developed that handles positive and negative signal paths simultaneously and integrates all sub-blocks into a complete SAR ADC analog core.

This evolution represents a significant engineering contribution: rather than re-

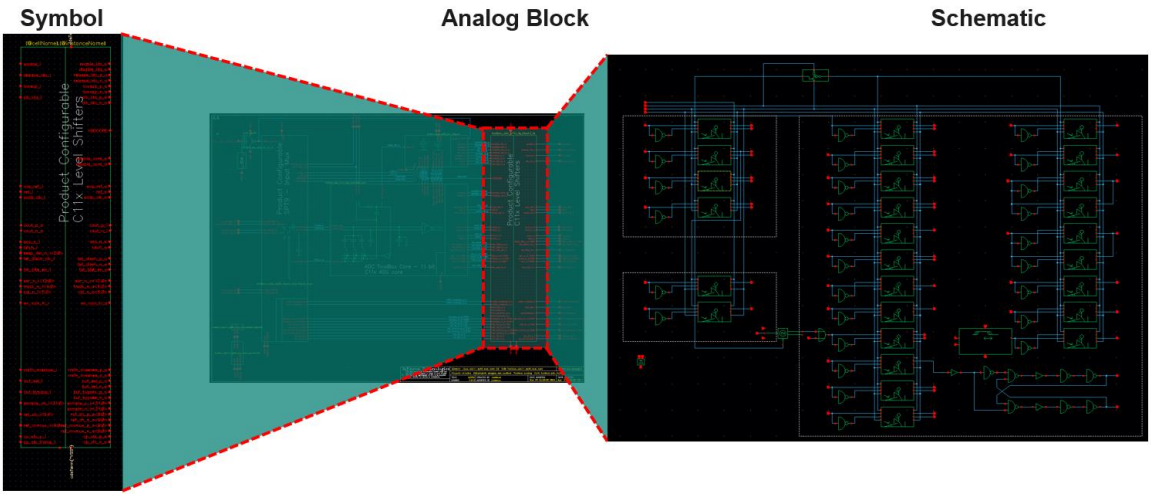


Figure 4.8: Level shifter block generated through AnaGen template migration — updated symbol (left), analog block (centre), and ported schematic with renamed bus widths (right).

designing each block manually for the differential variant, the automation framework allowed rapid reconfiguration of all sub-blocks through parameter changes and targeted script modifications. Algorithm 6 summarises the three-stage evolution.

4.6 Configurable Block-Based Schematic Generation

The proposed SAR ADC automation framework adopts a configurable, block-based approach to schematic generation. Rather than modifying individual transistors or manually editing subcircuits, the complete SAR ADC is constructed by instantiating and interconnecting a set of predefined block-level schematic templates. Each major functional block — including the sampling network, capacitive DAC (CDAC), comparator, and SAR logic — is implemented as a reusable schematic template whose configuration and integration are controlled programmatically through Python.

This methodology decouples *circuit architecture* from *circuit generation*. Architectural choices such as ADC resolution, DAC structure, sampling strategy, comparator variant, and clocking scheme are captured within stable schematic templates, while Python acts as an orchestration layer that governs block selection, parameter propagation, and hierarchical connectivity. As a result, the generated SAR ADC schematic automatically adapts to new specifications without requiring manual schematic editing.

Each functional block is generated using a dedicated Python generator that ac-

Algorithm 5 Non-Contiguous Bus Name Generation

Require: *pin_names* (list of pin names with embedded indices)
Ensure: *bus_string* encoding contiguous or non-contiguous ranges

Extract indices from *pin_names* by parsing name strings

if $\max(indices) - \min(indices) + 1 = \texttt{len}(indices)$ **then**
    return `'<'` + $\max(indices)$ + `':'` + $\min(indices)$ + `'>'` ▷ Simple contiguous bus
else
    Sort *indices* in descending order
    ranges ← [ ]
    current_range ← [*indices*[0]]
    for $i = 1$ **to** $\texttt{len}(indices) - 1$ **do**
        if $indices[i] = indices[i-1] - 1$ **then**
            Append *indices*[*i*] to *current_range*
        else
            Finalise *current_range* → add to *ranges*
            current_range ← [*indices*[*i*]]
        end if
    end for
    Finalise last *current_range* and add to *ranges*
    return `'<'` + `','.join`(*ranges*) + `'>'`
end if

Example:
Input : *indices* = [1, 3, 7, 8, 9, 32]
Output: `'<32,9:7,3,1>'` *(one contiguous sub-range 9:7, remaining indices isolated)*

cepts a well-defined set of configuration parameters. For example, the CDAC generator derives the required capacitor array and switching structure from the target resolution and unit capacitance; the sampling block adapts to different switch topologies and clock definitions; and the SAR logic interface automatically scales its control and output bus widths to match the ADC resolution. Importantly, Python does not directly modify individual transistor instances. Instead, it instantiates validated schematic templates and configures their interfaces, preserving design correctness while enabling rapid reconfiguration.

After block generation, the framework performs automatic top-level schematic assembly. All blocks are placed within a predefined hierarchical structure to ensure consistent organisation across generated designs. Rule-based interconnection is

Algorithm 6 Framework Evolution: Single-Ended to Differential

Stage 1: Single-Ended Implementation

for all *voltage_domain* ∈ {Low, Medium, High Voltage} **do**
    `GenerateSwitchBlock`(*channel_map*, *voltage_domain*)
end for
`GenerateSamplingCapacitor`(*c*16_*count*, *c*122_*count*)
`GenerateInputMux`(*channel_map*)
    → Output: Block schematics for single-ended signal path

Stage 2: Technology Migration

for all *block* ∈ *single_ended_blocks* **do**
    `MigrateSchematic`(old_bus_config, new_bus_config)
    `MigrateSymbol`(old_pin_config, new_pin_config)
end for
    → Output: Ported schematics for updated channel configuration

Stage 3: Differential Implementation

for all *block* ∈ *ported_blocks* **do**
    Add positive path pins: `sample_p` signals
    Add negative path pins: `sample_n` signals
    Duplicate instantiation loops for both paths
    Update symbols with differential pin pairs
end for

`AssembleTopLevel()`:
    Instantiate all differential sub-blocks
    Connect inter-block differential signal paths
    Rename all internal nets to differential naming
    Generate top-level symbol for complete ADC core
    → Output: Complete differential SAR ADC core schematic

then applied to establish all required signal paths, including connections between the CDAC and comparator, bit-control routing from the SAR logic to the DAC switching network, distribution of sampling and comparator clocks, and assignment of reference rails. All nets, buses, and pins are created automatically using systematic naming conventions, eliminating common sources of schematic errors such as floating nodes or mismatched bus widths.

The final output of the automation process is a fully hierarchical, simulation-ready SAR ADC schematic. All external interfaces — including analog inputs, reference nodes, clock signals, and digital output buses — are automatically generated and exposed, allowing immediate integration with mixed-signal simulation environments.

Structural consistency checks are performed prior to export to ensure complete connectivity and configuration coherence. No manual schematic modification is required at any stage of the process.

To illustrate the result of the proposed automation flow, the block-level outputs include the sampling capacitor network (Fig. 4.4), the multi-voltage switch blocks (Fig. 4.5), the input multiplexer (Fig. 4.6), and the level shifter blocks generated from scratch and via template migration (Fig. 4.7 and Fig. 4.8, respectively). The complete top-level assembly is shown in Fig. 4.9 and Fig. 4.10. Detailed circuit-level design analysis and performance evaluation are outside the scope of this chapter.

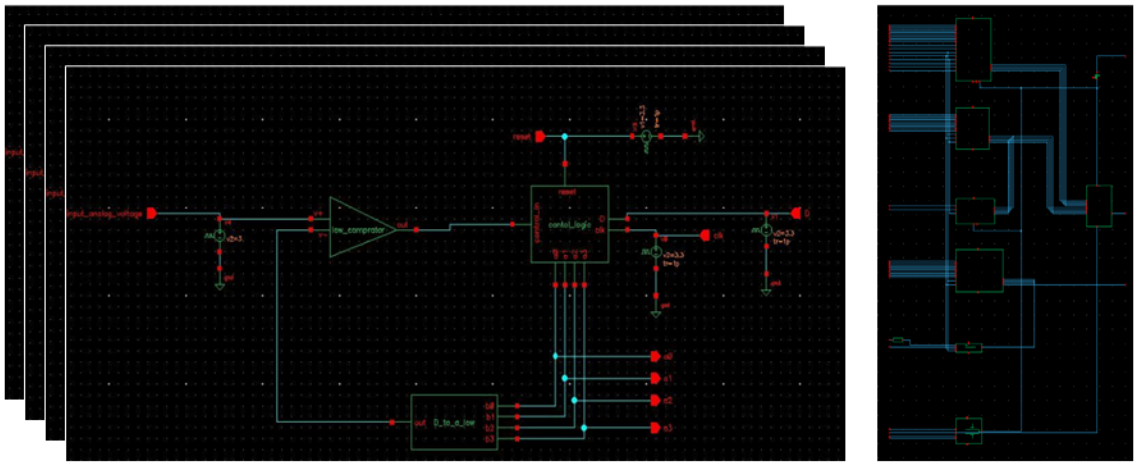


Figure 4.9: Partial view of an automatically generated SAR ADC schematic highlighting hierarchical block instantiation and deterministic inter-block connectivity.

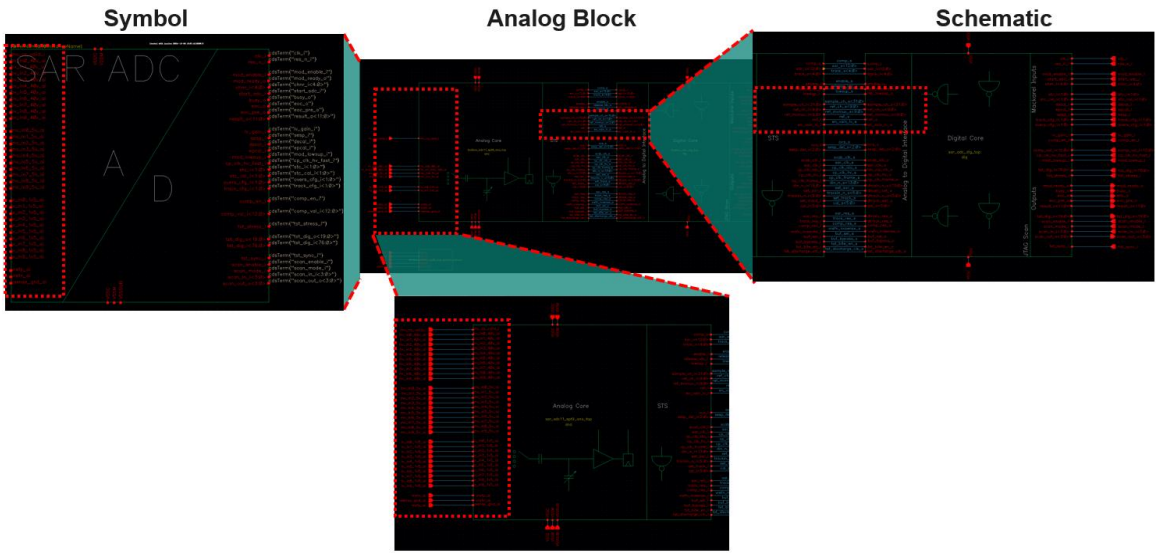


Figure 4.10: Automatically generated top-level SAR ADC analog core — symbol (left), hierarchical block view (centre), and complete assembled schematic (right) — confirming successful end-to-end generation via AnaGen.

4.7 Structural Verification

Following automatic schematic generation, structural verification was performed on all generated blocks to confirm correct connectivity and interface consistency. The verification checks performed are summarised in Table 4.3.

Table 4.3: SAR ADC Sub-block Verification Checks.

Verification Check	**Method**	**Result**
Net connectivity check	schCheck() call	Passed — all blocks
Pin-to-wire mapping	Automated wire trace	No floating nodes detected
Bus width consistency	Pin name verification	Consistent across all levels
Symbol-schematic match	VIC integrity check	All symbols verified
Inter-block interface	Net name matching	Correct propagation confirmed

It is important to note that the objective of this chapter is limited to automated

schematic generation. Circuit-level performance evaluation, transistor sizing, and simulation-based verification of the SAR ADC are outside the scope of this work. The primary contribution demonstrated here is the structural automation pipeline — its correctness, scalability, and reproducibility across block variants and technology configurations.

4.8 Summary

This chapter presented a Python-based automation framework for the schematic generation of Successive Approximation Register (SAR) analog-to-digital converters, implemented using the AnaGen schematic generation environment integrated with Cadence Virtuoso. The proposed approach targets the SAR ADC core and exploits its inherently modular and repetitive structure to enable systematic, scalable, and configurable schematic generation. The framework was developed through a structured three-stage evolution: initial single-ended implementation, technology migration via automated pin and net renaming, and final differential implementation with top-level hierarchical assembly.

The chapter first reviewed the operating principle of SAR ADCs and decomposed the architecture into well-defined functional blocks, including the sampling network, capacitive DAC, comparator, and SAR logic. This decomposition established a clear structural foundation for automation by defining standardised block interfaces and hierarchical boundaries.

A structured automation pipeline was then introduced, in which Python serves as the central orchestration layer. User-defined specifications are translated into derived design parameters, after which block-level schematic templates are instantiated and interconnected using rule-based procedures. The automation operates exclusively at the block level, preserving validated circuit templates while enabling rapid adaptation to changes in resolution, DAC architecture, sampling strategy, comparator selection, and clocking scheme.

The key contributions demonstrated in this chapter are as follows. A parameterised sampling capacitor array generator was developed that automatically instantiates unit cells, error cells, and dummy cells at computed grid positions with systematic wiring. A unified multi-voltage switch generation algorithm was implemented supporting low, medium, and high voltage domains through a shared channel routing framework that handles both new output node creation and shared-node merging. An intelligent bus name generation algorithm was developed to handle non-contiguous pin index sets, enabling correct EDA tool representation of partial channel selections. An automated technology migration procedure was implemented enabling bulk pin renaming, net renaming, and symbol modification without manual schematic editing. Finally, all blocks were integrated into a complete top-level hierarchical SAR ADC core through automated assembly and interface verification.

Overall, this chapter establishes a robust schematic-generation layer within the

multi-level automation framework presented in this thesis. By eliminating manual schematic construction while preserving architectural flexibility, the proposed method provides a critical foundation for scalable analog design automation. The next chapter builds upon this work by addressing automated layout generation, completing the transition from architecture-level specification to physical implementation.

Chapter 5

Template-Based and Hybrid Layout Automation

5.1 Introduction to Automated Layout Generation

Analog and mixed-signal layout remains one of the most time-consuming and expertise-driven stages of the integrated circuit design flow. Unlike digital place-and-route, analog layout requires strict control over device matching, symmetry, orientation, and routing parasitics. These constraints are typically enforced manually by experienced designers, making the process difficult to scale and challenging to reuse across different designs or technology nodes.

With the increasing demand for rapid design iterations, customer-specific variants, and technology portability, manual analog layout has become a major bottleneck in modern industrial design flows. This motivates the need for automation approaches that preserve analog layout quality while significantly reducing development effort.

In this chapter, a **template-based and rule-driven analog layout automation methodology** is presented. The proposed approach is built around an in-house **Analog Generator (AnaGen)**, which encodes analog layout knowledge—such as matching rules, symmetry constraints, and routing preferences—into reusable and parameterized layout templates. Instead of generating layouts from scratch for each design, these templates can be automatically instantiated across different circuits and technology nodes.

The overall layout automation framework adopted in this work is illustrated in Fig. 5.1. The framework combines schematic-derived constraints, reusable layout templates, and technology rule layers to perform automated placement and routing within a unified flow.

The framework is evaluated on a set of representative analog building blocks commonly used in mixed-signal systems:

- an inverter,

- a single-ended amplifier,
- a dynamic comparator,
- an Ibias generator with and without trimming capability.

These blocks were intentionally selected to cover a wide range of layout challenges, ranging from simple connectivity to symmetry-critical and matching-sensitive structures.

Within the same automation framework, multiple routing strategies are explored:

- **AnaGen template-based routing**, enforcing analog-quality constraints,
- **VSR (Virtuoso Space-based Router)**, enabling extremely fast and compact routing,
- **Hybrid routing**, combining VSR-based connectivity with AnaGen refinement,
- **Manual routing**, used as a baseline for comparison.

In addition, the framework supports **technology porting** through rule-based abstraction of layout layers, routing constraints, and device parameters. This allows the same placement and routing templates to be reused across different CMOS technology nodes with minimal manual intervention.

This chapter presents the complete layout automation flow, including template-based placement, routing strategy selection, quantitative comparison of layout methods, and technology-porting experiments. Together, these results demonstrate how analog layout automation can significantly improve productivity while maintaining layout quality comparable to expert manual designs.

5.2 Layout Automation Framework

This section describes the operational flow of the proposed layout automation framework. Rather than focusing on individual analog layout rules, the emphasis here is on how schematic information, reusable templates, routing strategies, and technology rules are orchestrated to automatically generate a complete analog layout.

The framework follows a modular and template-based methodology built around the in-house **Analog Layout Generator (AnaGen)**. The overall flow, previously illustrated in Fig. 5.1, transforms a schematic-level design into a DRC-clean layout through a sequence of well-defined automation stages.

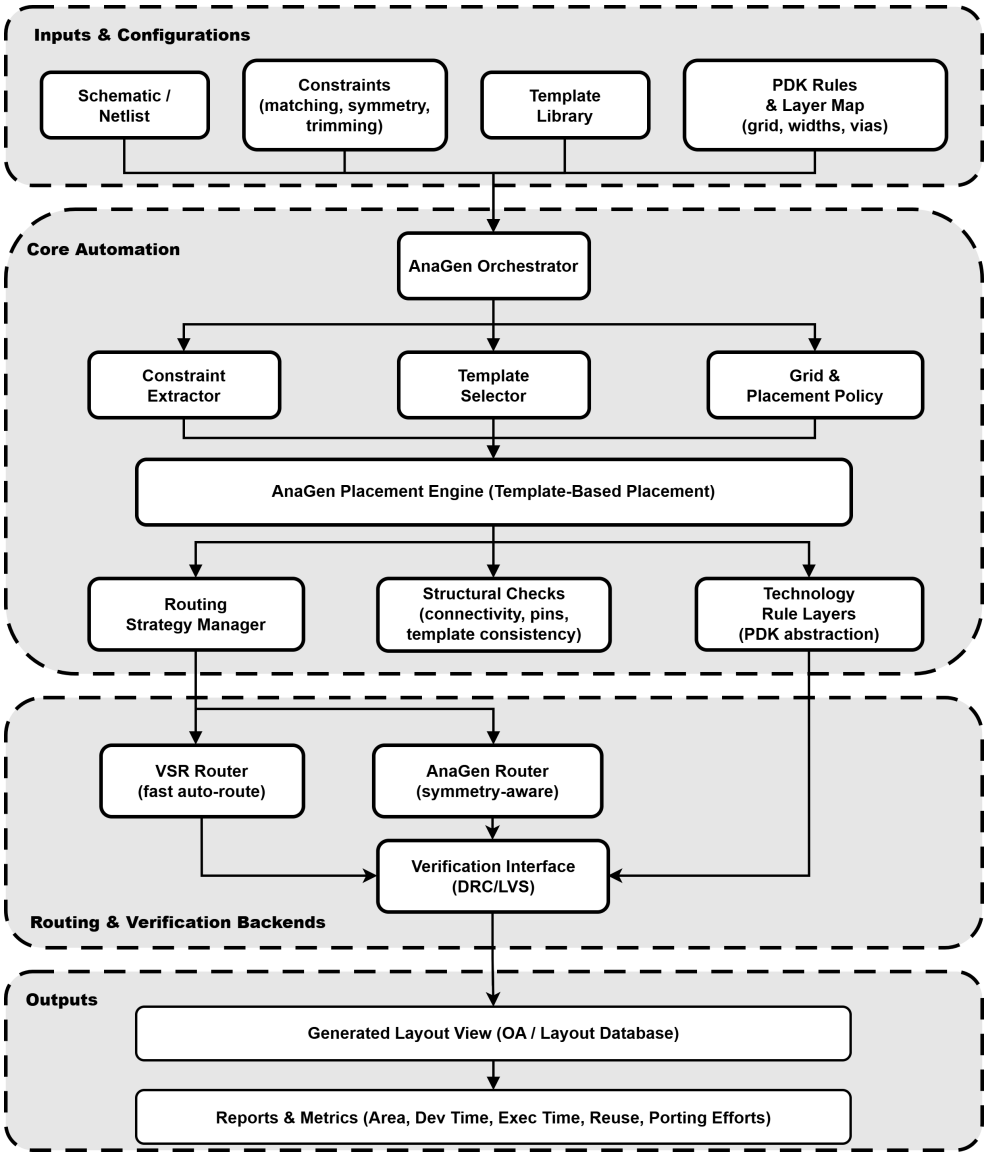


Figure 5.1: Template-based analog layout automation framework using AnaGen for placement and routing strategy selection.

5.2.1 Schematic-Driven Constraint Extraction

The layout generation process begins with schematic-driven analysis. Structural and connectivity information is extracted directly from the schematic or netlist representation of the circuit. This step identifies the elements required for template instantiation without embedding detailed analog layout knowledge at this stage.

The extracted information includes:

- device instances and hierarchical block boundaries,

- electrical connectivity between devices and sub-blocks,
- identification of matched device groups and replicated structures,
- classification of critical versus non-critical nets.

This information serves as a structural description of the circuit and forms the input to the template-selection stage.

5.2.2 Template Selection and Configuration

Based on the extracted schematic structure, the framework selects appropriate layout templates from the AnaGen template library. Each template represents a predefined placement and routing structure suitable for a specific class of analog blocks, such as inverters, amplifiers, comparators, or bias generators.

At this stage:

- The type of analog block determines the selected template,
- Template parameters (e.g., device count, array dimensions) are configured,
- Hierarchical composition of templates is established when required.

This step decouples layout generation from device-level customization and enables high reuse across different designs.

5.2.3 Template-Based Device Placement

Once a template is selected, AnaGen instantiates the corresponding placement structure. Device coordinates, orientations, and relative ordering are determined entirely by the template definition.

Placement generation at this stage:

- follows a deterministic, template-driven procedure,
- produces a complete and consistent placement for the block,
- remains independent of the selected routing strategy.

By separating placement from routing decisions, the framework allows multiple routing approaches to be evaluated on the same underlying placement.

5.2.4 Routing Strategy Selection

After placement generation, a routing strategy is selected for each block based on its structural complexity and sensitivity to layout parasitics. The framework supports multiple routing engines within the same flow:

- **VSR routing**, prioritizing execution speed and area efficiency,
- **AnaGen template-based routing**, prioritizing layout regularity and consistency,
- **Hybrid routing**, combining fast connectivity with template-based refinement.

The routing strategy is treated as a configurable parameter, allowing the same placement template to be routed using different methodologies without modifying the template itself.

5.2.5 Technology Rule Application and Porting

Technology portability is achieved by abstracting process-specific information into separate rule files. These rules define:

- metal layers and preferred routing directions,
- width and spacing constraints,
- via definitions and enclosure rules,
- grid resolution and alignment constraints.

By applying these rules during placement and routing, the same AnaGen templates can be reused across different CMOS technology nodes with minimal modification.

5.2.6 Layout Generation and Validation

The final stage of the framework generates the physical layout database view. Structural consistency checks are performed to ensure:

- complete connectivity between devices and blocks,
- consistency between schematic and layout hierarchy,
- compliance with template and technology constraints.

The resulting layout is exported in a format suitable for standard DRC and LVS verification flows.

Section 5.2 focused on the operational flow of the layout automation framework, while Section 5.3 details the analog layout constraints encoded within the templates.

5.3 Device-Level Placement Constraints

High-quality analog layout requires strict enforcement of device-level placement constraints in order to minimize mismatch, preserve symmetry, and control layout-induced parasitics. Unlike digital layouts, where placement is largely abstracted, analog circuits are highly sensitive to geometric relationships between devices. As a result, these constraints must be explicitly encoded and consistently enforced during layout generation.

In this work, device-level placement constraints are captured directly within reusable AnaGen templates and applied uniformly across all evaluated blocks, including the inverter, single-ended amplifier, dynamic comparator, and Ibias generators with and without trimming. The constraints described in this section represent domain-specific analog layout knowledge and are enforced deterministically through template definitions, rather than manual heuristics.

Figure 5.2 conceptually illustrates the key device-level placement constraints enforced by AnaGen templates, including common-centroid, interdigitated, and symmetry-aware arrangements. These constraints form the foundation of the template definitions described in the following subsections.

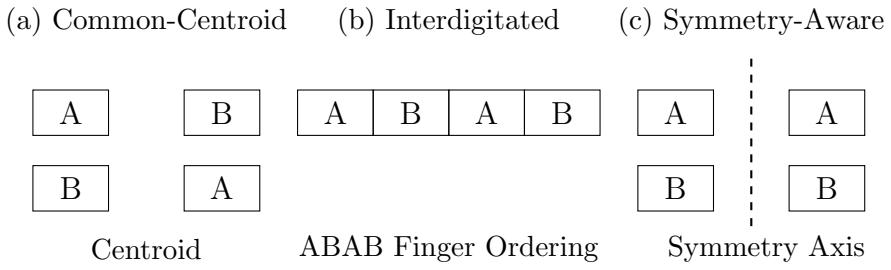


Figure 5.2: Conceptual illustration of device-level placement constraints enforced by AnaGen templates: (a) common-centroid placement for mismatch reduction, (b) interdigitated finger ordering to average process gradients, and (c) symmetry-aware placement around a defined symmetry axis for balanced parasitics.

5.3.1 Matching and Device Grouping

Matched devices in analog circuits are highly sensitive to variations in processing, stress, and local layout environment. To reduce systematic mismatch, devices that are electrically intended to match must be placed in close proximity and share identical surroundings.

AnaGen enforces the following matching-related placement constraints:

- explicit grouping of matched transistor sets,

- equal spacing between devices within a matched group,
- shared local environment for all devices in the group,
- alignment to a common placement grid to ensure consistent parasitics.

These constraints reduce both random and systematic mismatch across amplifier stages, comparator input pairs, and Ibias current mirrors.

5.3.2 Common-Centroid Placement

Process gradients across the layout area can introduce systematic offset in matched devices. Common-centroid placement is a well-established technique for cancelling such gradients by spatially interleaving matched devices around a geometric center.

AnaGen templates support multiple common-centroid configurations, including:

- ABBA, AABBBAA, and higher-order centroid patterns,
- horizontal and vertical symmetry axes,
- multi-row centroid structures for larger device arrays,
- centroid-based ordering for trimming-enabled Ibias arrays.

These patterns are used extensively in comparator input stages, differential amplifier structures, and trimming circuits to improve offset and matching robustness.

This concept is illustrated in Fig. 5.2(a), where matched devices are arranged around a geometric centroid to cancel first-order process gradients.

5.3.3 Interdigitated Arrays

Interdigitated placement is employed when equalizing parasitic loading and local environment is more critical than strict centroid symmetry. This is particularly useful for multi-finger devices and replicated bias elements.

AnaGen supports interdigitated placement through:

- alternating ABABAB ordering of device fingers,
- symmetric extension for odd and even device counts,
- controlled interleaving to balance parasitic capacitance and resistance,
- awareness of routing channels within interdigitated structures.

This approach is applied to Ibias generators and selected amplifier input stages where balanced parasitics are essential.

An example of interdigitated finger ordering is shown in Fig. 5.2(b), demonstrating alternating device placement to equalize parasitic loading and local variations.

5.3.4 Orientation and Mirroring Rules

Device orientation directly affects well proximity, stress distribution, and parasitic coupling. Inconsistent orientation within matched structures can therefore introduce systematic mismatch.

AnaGen enforces strict orientation rules, including:

- a limited set of allowed orientations (R0, R180, MX, MY),
- mirrored placement of symmetric devices in amplifiers and comparators,
- consistent orientation across matched Ibias branches,
- prevention of disallowed mirroring in trimming-sensitive devices.

These rules ensure that matched devices experience comparable physical environments, improving robustness across process corners.

5.3.5 Symmetry Enforcement

Symmetry is a fundamental requirement in precision analog blocks such as comparators and amplifiers. Asymmetric placement can lead to unequal parasitic paths, degrading offset, gain, and timing behavior.

AnaGen templates explicitly encode symmetry constraints by enforcing:

- vertical or horizontal symmetry axes at the block level,
- mirrored placement of corresponding devices across the symmetry axis,
- symmetric routing entry points for critical nets,
- equal interconnect lengths for delay-sensitive paths.

Symmetry enforcement is applied consistently during placement generation and preserved during routing refinement.

Figure 5.2(c) illustrates symmetry-aware placement, where devices are mirrored across a defined symmetry axis to preserve balanced parasitic paths.

5.3.6 Grid and Alignment Constraints

To ensure predictable routing behavior and DRC compliance, all devices are placed on a regular and technology-aware grid. Grid-based placement also improves template reuse and technology portability.

AnaGen applies the following alignment constraints:

- uniform device row height across the placement region,

- aligned diffusion sharing for matched transistors,
- regular routing rails aligned with the placement grid,
- track-based alignment compatible with both VSR and AnaGen routing engines.

These constraints contribute to consistent layout quality across different blocks and technology nodes.

5.3.7 Summary of Placement Constraints

The device-level placement constraints described in this section capture essential analog layout knowledge required for matching, symmetry, and parasitic control. By encoding these constraints within reusable AnaGen templates, the proposed framework achieves consistent placement quality across a diverse set of analog blocks, while enabling scalable, automated, and technology-portable layout generation.

5.4 Routing Techniques

Routing has a strong influence on the electrical performance and robustness of analog layouts. Interconnect parasitics, routing symmetry, shielding, and metal usage directly affect matching, offset, noise, and timing behavior. In this work, multiple routing techniques are explored within a unified template-based automation framework to address the diverse routing requirements of analog building blocks with different sensitivity levels.

Rather than performing a direct one-to-one comparison between individual routing tools, the objective is to analyze how different routing strategies occupy distinct positions in the speed–quality–reuse design space. The conceptual differences between the routing strategies explored in this work are summarized in Fig. 5.3, highlighting trade-offs in automation effort, analog routing quality, and template reuse.

The investigated routing strategies include manual routing as a reference baseline, fast automatic routing using VSR, constraint-driven template-based routing using AnaGen, and a hybrid approach that combines rapid connectivity with analog-aware refinement. Each strategy is applied selectively depending on the analog sensitivity of the target block, ranging from simple connectivity in inverters to symmetry- and matching-critical routing in comparators and trimming-enabled Ibias generators.

5.4.1 Manual Routing (Reference Baseline)

Manual routing represents the traditional approach to analog layout design and serves as a reference baseline in this study. In this flow, routing decisions are made entirely by the designer, allowing precise control over wire paths, parasitic coupling, and

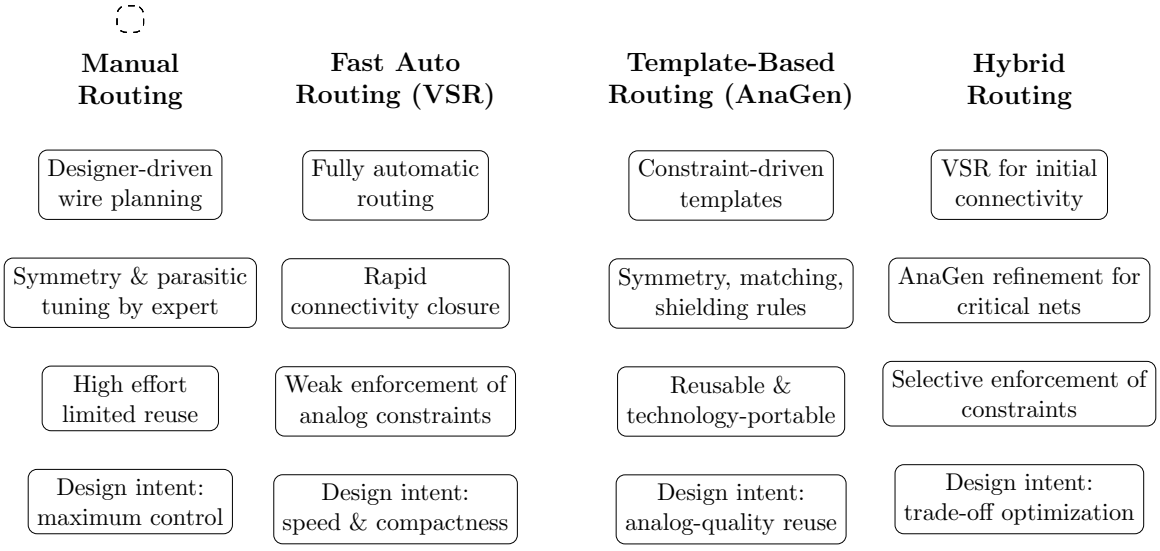


Figure 5.3: Conceptual comparison of routing strategies explored in this work, highlighting automation level, analog constraint awareness, and reuse intent.

symmetry enforcement. This approach is commonly used for highly sensitive analog blocks where layout quality is critical.

Manual routing was used to establish a qualitative benchmark for routing quality, symmetry, and parasitic control. It enables fine-grained optimization of differential paths, bias networks, and high-impedance nodes, particularly in comparators and trimming-enabled Ibias circuits.

However, manual routing requires significant design effort and does not scale well. Each layout must be recreated from scratch, making reuse and technology porting difficult. As a result, manual routing is impractical for rapid design iterations or large sets of related designs, motivating the need for automated alternatives.

5.4.2 VSR Routing (Virtuoso Space-based Router)

VSR is a lightweight automatic routing technique optimized for extremely fast layout generation with minimal setup overhead. It prioritizes execution speed and area compactness over strict enforcement of analog constraints, making it suitable for blocks with relatively simple connectivity and low sensitivity to routing asymmetry.

In this work, VSR was primarily applied to simple or moderately sensitive blocks such as inverters and Ibias generators without trimming. For these circuits, VSR was able to generate complete, LVS-clean layouts within a few seconds, enabling rapid design iteration and early verification.

Due to the absence of symmetry enforcement and matched routing control, VSR is not well suited for precision-critical blocks such as comparators. Its role in the overall

flow is therefore focused on speed and early-stage layout generation rather than final high-accuracy routing.

5.4.3 AnaGen Template-Based Routing

AnaGen provides a template-based routing engine designed specifically for analog layout quality. Routing constraints such as symmetry, shielding, preferred metal layers, and controlled parasitic loading are encoded directly into reusable templates. This allows routing decisions to be made in a consistent and rule-driven manner.

AnaGen routing was applied to analog blocks with high sensitivity to mismatch and parasitics, including comparators, single-ended amplifiers, and trimming-enabled Ibias generators. The routing templates ensure symmetric paths for differential signals, balanced parasitic environments, and predictable interconnect behavior.

While the initial development of routing templates requires additional effort, AnaGen provides high-quality and reusable routing structures. Once created, these templates can be reused across multiple designs and technology nodes, making this approach well suited for scalable analog IP development.

5.4.4 Hybrid Routing (VSR + AnaGen)

Hybrid routing combines the speed of VSR with the analog-aware refinement capabilities of AnaGen. In this approach, VSR is first used to establish complete connectivity quickly, after which AnaGen templates are applied to refine critical nets and enforce analog layout constraints.

This strategy was particularly effective for blocks with mixed sensitivity levels, such as single-ended amplifiers and Ibias generators with trimming. VSR enables rapid initial routing, while AnaGen selectively improves symmetry, matching, and parasitic control for sensitive signal paths.

Hybrid routing provides a practical compromise between development time and layout quality, allowing designers to benefit from automation speed without fully sacrificing analog performance requirements.

5.4.5 Routing Strategy Selection Across Blocks

The choice of routing strategy depends strongly on the analog sensitivity and structural complexity of each block. Based on the experimental evaluation in this work, the following observations were made:

- **Inverter:** VSR routing is sufficient due to simple topology and low sensitivity to routing asymmetry.

- **Single-ended amplifier:** Hybrid routing provides a balanced trade-off between speed and analog-quality refinement.
- **Comparator:** AnaGen routing is required to preserve symmetry and matching in critical differential paths.
- **Ibias (non-trimming):** VSR routing enables fast and compact layouts.
- **Ibias (with trimming):** Hybrid or AnaGen routing is necessary to maintain matching and trimming accuracy.

These results demonstrate that no single routing technique is universally optimal. Instead, routing strategies must be selected based on circuit sensitivity, accuracy requirements, and development-time constraints, highlighting the flexibility of the proposed template-based automation framework.

5.5 Application of the Proposed Layout Automation Framework

This section presents the schematics and corresponding generated layouts of the analog building blocks used to evaluate the proposed template-based layout automation framework. The selected blocks include an inverter, a single-ended differential amplifier, a comparator, and Ibias generators with and without trimming capability. These circuits were chosen to represent a wide spectrum of analog layout challenges, ranging from simple connectivity to symmetry-critical and matching-sensitive structures.

It is important to note that the layouts shown in this section were generated using different routing strategies within the same template-based automation framework. Depending on the analog sensitivity and structural complexity of each block, VSR routing, AnaGen template-based routing, hybrid routing, or manual routing was selectively applied. This demonstrates the flexibility of the proposed approach, where a common template-based placement methodology is preserved while routing strategies are adapted to meet circuit-specific requirements.

For each block, the schematic is shown alongside one or more layout variants to illustrate how the same automation framework supports different routing techniques without altering the underlying placement templates.

5.5.1 Inverter

The inverter represents the simplest building block considered in this study and serves as a baseline for evaluating layout compactness and routing efficiency. Due to its low analog sensitivity and simple connectivity, template-based placement combined with automated routing is sufficient to generate compact and DRC-clean layouts.

Fig. 5.4 illustrates the inverter implementation using the proposed framework, showing (a) the schematic, (b) the placement template with vertical and horizontal configurations, and (c) the automatically generated layout. The result highlights that the proposed template-based approach can efficiently handle simple analog cells with minimal development effort.

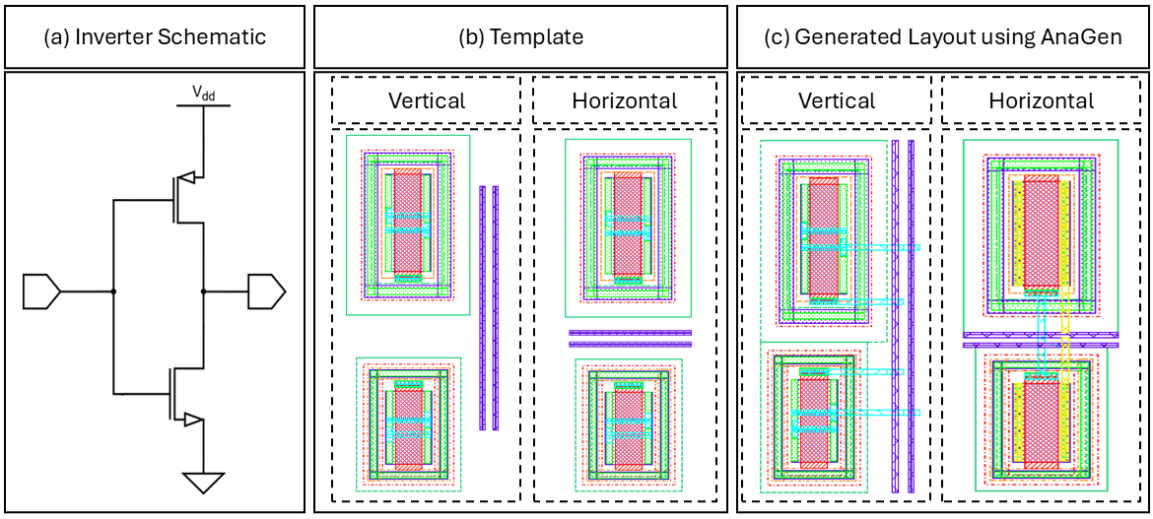


Figure 5.4: Inverter implementation using the proposed template-based layout automation framework.

Having established the baseline performance of the proposed framework on a simple block, the following section examines a more constraint-intensive structure where matched device placement and parasitic control become critical.

5.5.2 Single-Ended Differential Amplifier

The single-ended differential amplifier introduces stronger analog constraints, including matched devices and sensitivity to routing parasitics. To address these requirements, a hybrid routing strategy was employed, combining fast connectivity generation with constraint-driven refinement.

Fig. 5.5 presents the single-ended differential amplifier implementation using the proposed framework, showing (a) the schematic, (b) the placement template with vertical and horizontal configurations, and (c) the automatically generated layout. The result demonstrates how template-based placement ensures matched device arrangement, while routing refinement improves symmetry and parasitic predictability for critical signal paths.

Building on the hybrid routing results of the differential amplifier, the next section evaluates the Ibias generator, which introduces replicated branches and trimming-sensitive structures requiring careful matching across routing strategies.

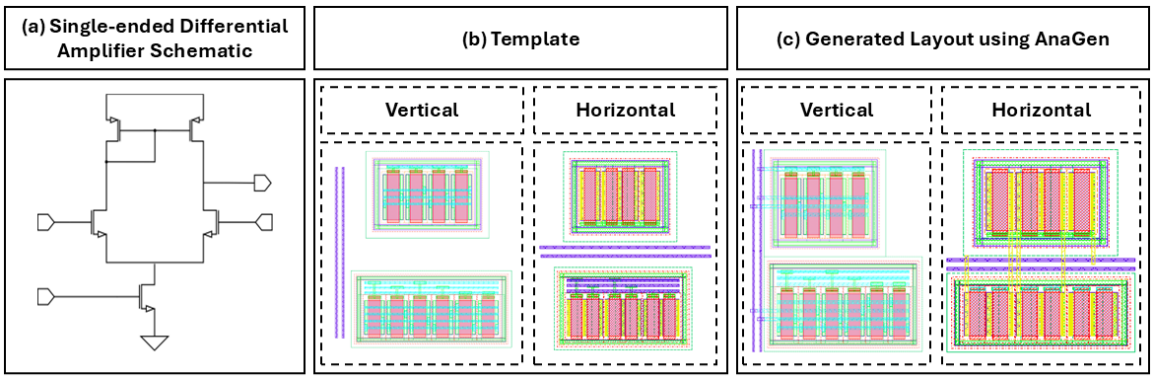


Figure 5.5: Single-ended differential amplifier implementation using the proposed template-based layout automation framework.

5.5.3 Ibias Generator (With and Without Trimming)

Ibias generators are particularly sensitive to matching and, in the trimming-enabled case, require careful routing of replicated branches and control elements. Fig. 5.6 shows the schematics used for layout automation. Fig. 5.6(a) presents the basic Ibias generator without trimming, while Fig. 5.6(b) shows the trimming-enabled variant, which introduces selectable branches for fine-grained current adjustment. Both variants share the same fundamental architecture, allowing a direct comparison of routing strategies and layout robustness under increasing complexity.

Fig. 5.7 compares different layout realizations of the non-trimming Ibias generator. Fig. 5.7(a) shows the reusable placement template defining device groupings and routing channels. Fig. 5.7(b) presents the AnaGen template-based routed layout, emphasizing structured routing and consistent parasitic control. Fig. 5.7(c) shows the hybrid routing result, where VSR provides fast connectivity and AnaGen refines selected nets. Fig. 5.7(d) shows the manually routed layout, used as a reference baseline for quality comparison. Fig. 5.7(e) illustrates the VSR-only routed layout, which achieves the most compact area and fastest execution. For the non-trimming Ibias generator, VSR routing was sufficient to produce a compact and electrically acceptable layout. The absence of replicated trimming branches reduces sensitivity to routing asymmetry, making fast automatic routing an effective choice.

The trimming-enabled Ibias generator imposes significantly stronger layout constraints due to the presence of multiple replicated branches that must remain well matched.

Fig. 5.8 shows the corresponding layout results. Fig. 5.8(a) presents the placement template, where trimming branches are arranged symmetrically to preserve matching. Fig. 5.8(b) shows the AnaGen-routed layout, which enforces consistent routing topology across all trimming paths. Fig. 5.8(c) illustrates the hybrid routing result,

combining fast connectivity with template-based refinement. Fig. 5.8(d) presents the manually routed layout used as a reference for symmetry and matching quality. Fig. 5.8(e) shows the VSR-only routed layout, highlighting its limited suitability for trimming-critical structures due to weak constraint enforcement.

These results demonstrate that while VSR routing is effective for simple bias circuits, trimming-enabled structures require either template-based or hybrid routing to preserve matching and ensure predictable electrical behavior.

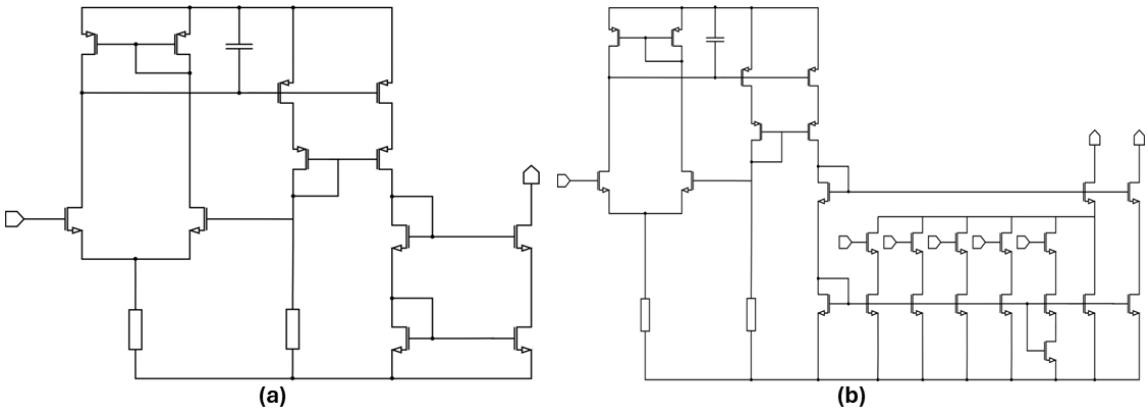


Figure 5.6: Schematic of the Ibias generator used for layout automation: (a) basic Ibias topology without trimming and (b) trimming-enabled Ibias variant with selectable branches for current adjustment.

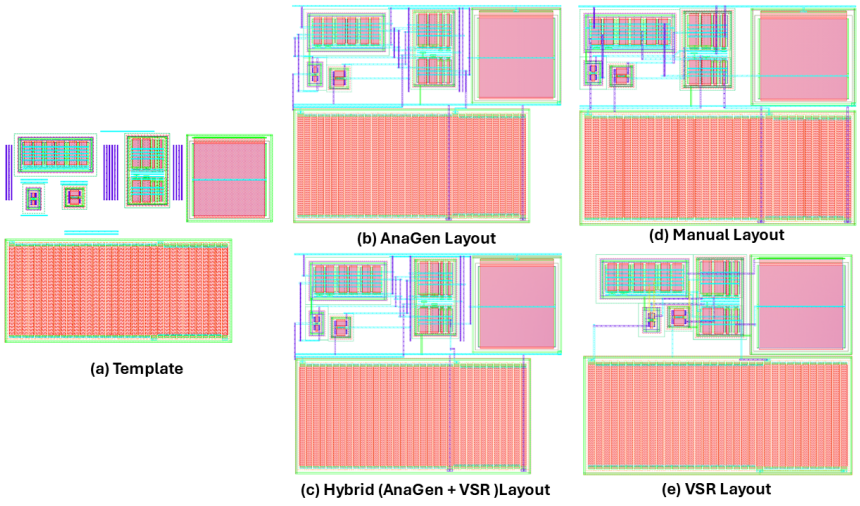


Figure 5.7: Generated layouts of the Ibias generator without trimming.

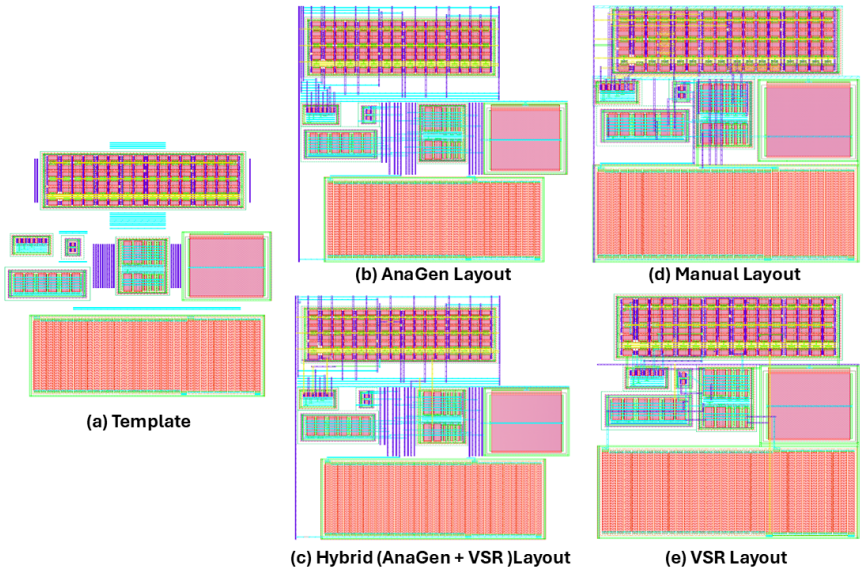


Figure 5.8: Generated layouts of the trimming-enabled Ibias generator.

5.5.4 Comparator

The comparator represents the most symmetry-critical block evaluated in this work. Strict symmetry and matched routing paths are required to ensure proper offset, decision accuracy, and timing behavior. Any imbalance in device placement or routing can directly translate into systematic offset and degraded dynamic performance.

Fig. 5.9 presents the schematic of the dynamic comparator used in this study. The circuit includes multiple differential paths, regeneration devices, and biasing structures, making it highly sensitive to both placement and routing-induced parasitics.

Fig. 5.10 compares the layouts generated using different routing strategies. Fig. 5.10(a) shows the reusable placement template defining device grouping, symmetry axes, and routing channels. Fig. 5.10(b) illustrates the layout generated using AnaGen template-based routing, where symmetry, matching, and controlled routing are explicitly enforced. Fig. 5.10(c) presents the hybrid routing result, combining fast VSR connectivity with AnaGen refinement for critical nets. For comparison, Fig. 5.10(d) shows a manually routed layout created by an experienced designer, while Fig. 5.10(e) depicts a fully VSR-routed layout.

The comparison demonstrates that AnaGen routing preserves structural symmetry and balanced parasitic paths comparable to manual routing, while offering significantly higher reusability. In contrast, VSR routing, although fast, fails to maintain consistent symmetry and matching, making it unsuitable for highly sensitive blocks such as comparators.

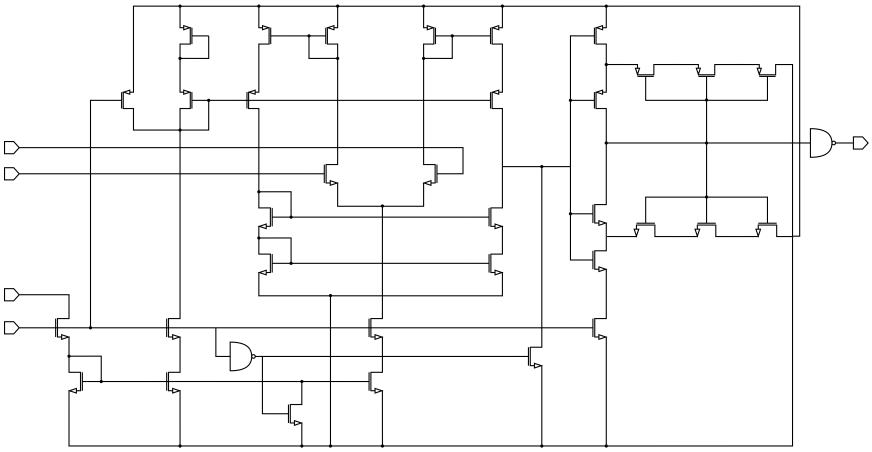


Figure 5.9: Schematic of the comparator used for layout automation experiments.

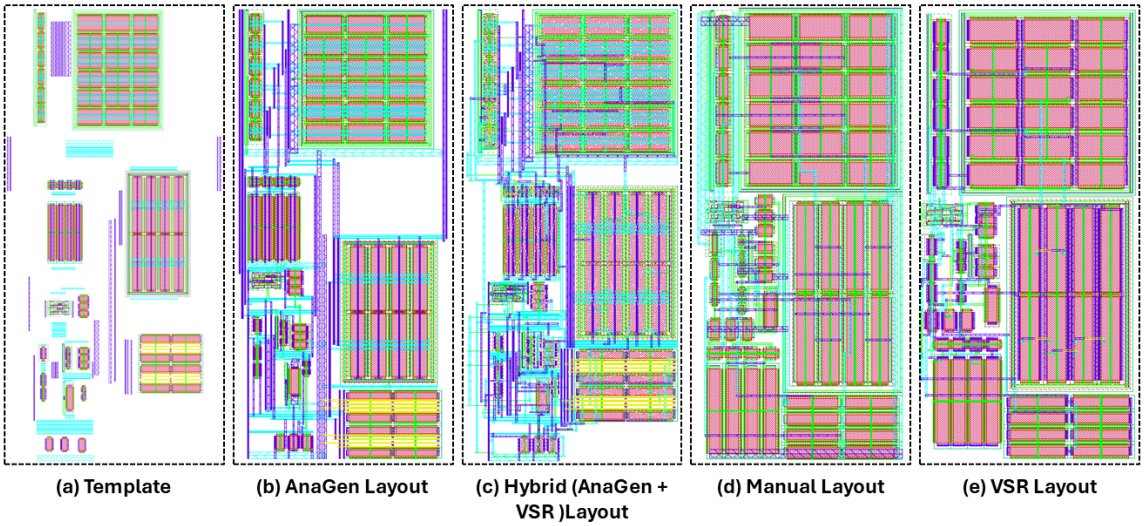


Figure 5.10: Generated layouts of the comparator using different routing techniques.

5.6 Quantitative Comparison of Layout Methods

This section presents a quantitative evaluation of the layout development and routing strategies investigated in this work. The comparison focuses on development effort, routing execution time, generated layout area, and overall suitability across analog blocks with different sensitivity levels.

Rather than a direct tool-to-tool benchmark, the objective is to analyze how different routing strategies occupy distinct points in the trade-off space between layout quality, design effort, execution speed, and reusability within a common template-based layout automation framework.

Table 5.1: Routing strategy comparison across benchmark circuits.

Circuit / Metric	**Manual**	**VSR**	**AnaGen**	**Hybrid**
		Comparator		
Dev. Time	–	$<$ 1 min	26–34 h	**6–7 h**
Exec. Time	16–18 h	$<$ 2 min	5–10 min	$<$ **5 min**
Area (μm^2)	1295	$\approx$1295	2115	**1795**
		Ibias (No Trim)		
Dev. Time	–	$<$ 1 min	3–4 h	**1–2 h**
Exec. Time	1.5–2 h	$\sim$5 s	3–5 min	$\sim$**5 s**
Area (μm^2)	1510	1475	1715	**1655**
		Ibias (With Trim)		
Dev. Time	–	$<$ 1 min	7–8 h	**3–4 h**
Exec. Time	3–4 h	$<$ 10 min	5–10 min	**5–10 s**
Area (μm^2)	2080	1930	2915	**2560**

In this analysis, **manual routing is treated as the reference baseline**, as it represents the traditional approach used by experienced analog designers and provides a benchmark for layout quality and compactness. Automated methods are evaluated relative to this baseline in terms of efficiency, scalability, and ability to preserve analog constraints.

The initial template development represents a one-time investment of approximately 3–34 hours depending on block complexity. Once templates are available, layout generation for new designs or technology nodes can be completed within minutes, confirming the scalability of the proposed framework.

5.6.1 Area Efficiency Analysis

The generated layout area varies significantly depending on the routing strategy and the level of analog constraints enforced. As shown in Table 5.1, manual routing consistently produces compact layouts and therefore serves as a strong baseline for area comparison.

For the comparator, manual routing achieves the smallest area (1295 µm^2), reflecting careful human optimization and symmetry-aware routing. AnaGen routing produces a significantly larger area (2115 µm^2), which is expected due to conservative placement spacing, explicit symmetry enforcement, and structured routing tracks. The hybrid routing approach reduces the area to 1795 µm^2, demonstrating that partial automation can recover a large fraction of the compactness of manual layouts while preserving analog robustness. Notably, VSR routing achieves an area comparable to manual routing (approximately 1295 µm^2), confirming its compactness for blocks where strict analog constraints are not required.

A similar trend is observed for the Ibias generator without trimming. Manual

routing results in an area of 1510 µm^2, while VSR routing achieves a slightly smaller area of 1475 µm^2 by aggressively minimizing wire length and spacing. AnaGen routing increases the area to 1715 µm^2, whereas hybrid routing provides an intermediate result of 1655 µm^2.

For the Ibias generator with trimming, the impact of routing constraints becomes more pronounced. AnaGen routing produces the largest area (2915 µm^2) due to the need for balanced trimming structures and matched routing paths. Manual routing achieves a more compact layout (2080 µm^2), while VSR routing reduces area further to 1930 µm^2 at the cost of weaker constraint enforcement. Hybrid routing offers a practical trade-off at 2560 µm^2, balancing compactness with matching accuracy.

5.6.2 Development and Execution Time Analysis

Routing execution time and development effort show clear distinctions among the evaluated strategies. VSR routing is by far the fastest approach, requiring less than one minute of setup and only a few seconds for execution. This makes it particularly useful for rapid prototyping and early-stage design exploration.

Manual routing, while offering maximum control and compact layouts, requires substantial execution time, typically between one and two hours per block. This significantly limits scalability and reuse, especially in industrial design flows with frequent design iterations.

AnaGen routing requires a higher initial development effort (3–34 hours) due to the definition of placement and routing templates. In addition, full AnaGen routing can require several hours of execution time for complex blocks. However, this cost is largely amortized through reuse: once templates are available, layout regeneration can be completed within minutes, enabling efficient design portability and customer-specific variants.

The hybrid routing strategy reduces development effort compared to full AnaGen routing while maintaining acceptable analog quality. By combining fast VSR-based connectivity with selective AnaGen refinement, the hybrid approach offers a favorable balance between execution speed and layout robustness.

5.6.3 Suitability Across Analog Blocks

The quantitative results confirm that routing strategy selection should be guided by the analog sensitivity of the target block rather than by a single universal method. VSR routing is well suited for simple blocks such as inverters and non-trimming Ibias generators, where routing parasitics and symmetry constraints are limited.

For single-ended amplifiers, the hybrid routing approach provides an effective compromise, preserving critical analog constraints while reducing development time. For symmetry-critical circuits such as comparators, AnaGen routing is essential to ensure balanced parasitics and robust operation. Similarly, Ibias generators with

trimming networks benefit from AnaGen or hybrid routing to maintain matching accuracy.

5.6.4 Summary of Comparative Findings

This quantitative comparison demonstrates that no single routing strategy is optimal across all analog blocks. Manual routing provides compact, high-quality layouts but does not scale. VSR routing excels in speed and compactness but lacks analog awareness. AnaGen routing prioritizes robustness, symmetry, and reusability at the cost of area and initial effort. Hybrid routing offers a practical middle ground, combining fast automation with analog-quality refinement.

Overall, the results highlight the key strength of the proposed template-based layout automation framework: it enables flexible routing strategy selection while maintaining consistency, scalability, and analog design integrity across diverse circuit types and design objectives.

5.6.5 Comparison with State-of-the-Art

Table 5.2 compares the proposed framework against representative state-of-the-art analog layout automation tools in terms of runtime, manual effort, PDK support, and layout quality.

BAG [57] provides a highly flexible generator-based framework but requires days of expert effort per design and is tightly coupled to a single PDK. ALIGN [58] and MAGICAL [59] significantly reduce runtime to the range of minutes and achieve layout quality comparable to manual routing, but still require expert-level setup and remain limited to a single technology node. LayoutCopilot [60] further reduces runtime to under one minute, though it similarly demands expert involvement and targets a single PDK.

The proposed AnaGen-based framework achieves competitive runtime of under five minutes while requiring only a few hours of effort from a new user, owing to the reusable and parameterized nature of its templates. Critically, it is the only approach in this comparison that natively supports multi-PDK deployment through rule-driven technology porting, enabling layout reuse across different CMOS technology nodes without modifying placement or routing templates. Layout quality remains comparable to manual routing, as demonstrated by the quantitative results in Table 5.1.

5.7 Technology Porting

A key advantage of a template-based layout methodology is the ability to migrate an already-verified layout structure across different CMOS technology nodes with

Table 5.2: Comparison with state-of-the-art analog layout automation approaches.

Tool	Runtime	Manual Effort	PDK Support	Layout Quality
BAG [57]	Hours–days	Days (expert)	Single	Designer-dependent
ALIGN [58]	10–30 min	Hours (expert)	Single	Comparable to manual
MAGICAL [59]	1–5 min	Hours (expert)	Single	Comparable to manual
LayoutCopilot [60]	<1 min	Hours (expert)	Single	Comparable to manual
Proposed	**<5 min**	**Hours (new user)**	**Multi-PDK**	**Comparable to manual**

minimal manual intervention. In contrast to manual layouts—where device placement patterns, routing topology, and even cell organization often require redesign—AnaGen preserves the *structural intent* of the layout and localizes technology dependency into PDK-specific rule updates. In this way, layout migration becomes a rule-driven transformation problem: the same template structure is replayed under a new set of layer, grid, PCell, and routing constraints.

Fig. 5.11 summarizes the adopted technology-porting flow. Starting from an *existing template-based layout* (OA/GDS plus template metadata), the designer provides the target porting intent (e.g., target PDK, metal-stack choices, and routing/EM classes) together with the corresponding PDK rule files. A centralized *technology porting controller* derives a technology-abstraction view and a transformation plan that drives four main adaptation steps: (i) layer/via mapping, (ii) grid and pitch scaling, (iii) PCell substitution, and (iv) routing rule adaptation. The ported layout is then obtained by template re-instantiation, followed—when required—by an optional re-routing/adjustment stage and a final export to OA/GDS.

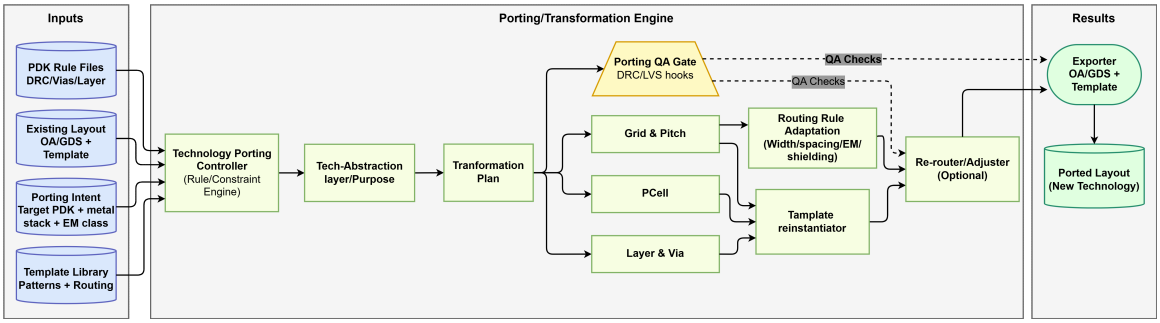


Figure 5.11: Rule-driven technology porting flow for a template-based analog layout in AnaGen

5.7.1 Rule-Based Layer and Via Mapping

Different PDKs provide different metal stacks, via definitions, and enclosure rules. In the proposed flow (Fig. 5.11), this dependency is captured by a rule-driven *layer and via mapping* stage that translates the original layout layers into the target technology through mapping tables. These tables specify: (i) layer naming and purpose equivalence, (ii) minimum width/spacing constraints, (iii) via type selection and enclosure requirements, and (iv) preferred routing directions when applicable. Consequently, porting typically requires updating the technology rule file rather than modifying the template structure.

5.7.2 Scaling of Device and Routing Grids

Technology migration often impacts grid resolution, track pitch, and placement alignment constraints. AnaGen therefore expresses template coordinates in *grid units* rather than absolute physical values, allowing the *grid and pitch* stage in Fig. 5.11 to scale placement rows/columns and routing tracks according to the target PDK. This preserves relative placement geometry—such as symmetry axes, centroid patterns, and interdigitation—while adapting the absolute dimensions to the new technology.

5.7.3 Device Abstraction Through PCells

Device portability is achieved by instantiating technology-specific parameterized cells (PCells) in the target PDK while keeping the template-level device roles unchanged. As shown in Fig. 5.11, the *PCell substitution* step maps each device in the template to the corresponding PCell in the new technology and adapts key parameters (e.g., W/L, finger count, multipliers, and well/guard-ring options). This avoids manual device replacement and enables consistent reuse of the same template description across nodes.

5.7.4 Routing Rule Adaptation

Routing constraints vary significantly across technologies due to differences in minimum width/spacing, via rules, density constraints, and electromigration limits. In the proposed flow, these effects are managed through *routing rule adaptation* (Fig. 5.11), where technology-specific rules are updated for: (i) layer-dependent width/spacing, (ii) EM-aware sizing for bias networks, (iii) shielding/spacing constraints for sensitive nets, and (iv) via selection and redundancy constraints. After rule adaptation, the layout can be re-instantiated and, if required, an optional re-router/adjuster can refine connectivity under the new constraints.

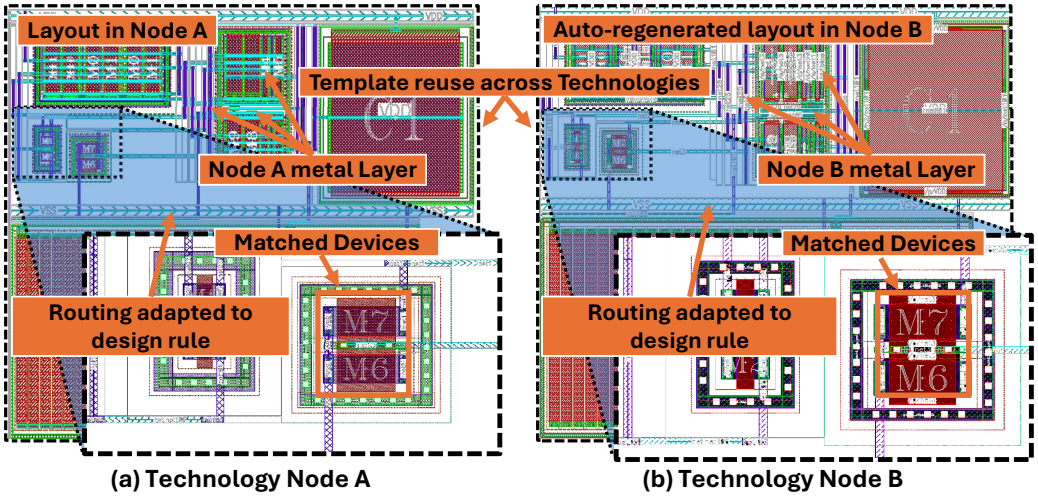


Figure 5.12: Illustration of template-based technology porting across two CMOS technology nodes.

5.7.5 Validation Across Analog Blocks

Fig. 5.12 demonstrates the outcome of the rule-driven porting flow applied to a representative analog block. The left side shows the original layout realized in Technology Node A, where matched devices, metal layer assignments, and routing paths are defined according to the source PDK design rules. The right side presents the auto-regenerated layout in Technology Node B, produced by replaying the same AnaGen template under the target PDK constraints. As evident from the comparison, matched device placement is preserved across both nodes, while metal layers and routing geometry are automatically adapted to the new technology rules — confirming that the proposed framework achieves structural portability without manual redesign. The porting procedure was exercised on the analog blocks investigated in this chapter: the inverter, the single-ended differential amplifier, the comparator, and the Ibias generator (with and without trimming). Across these blocks, the template structure and placement patterns remained unchanged, while only the PDK-dependent rule files required updates. Symmetry-critical structures (e.g., comparator cores) preserved their intended topology through grid/mapping consistency, while trimming-enabled Ibias layouts reused the same array organization with technology-specific routing adjustments when needed. These results confirm that the proposed approach enables structural portability of template-based layouts across technologies.

5.7.6 Summary of Technology Porting Benefits

Overall, the proposed rule-driven porting flow (Fig. 5.11) enables (i) reuse of placement structures across multiple PDKs, (ii) reduced effort when migrating to new processes, (iii) preservation of analog layout intent through template replay, and (iv)

scalable reuse for customer- or product-variant generation. By isolating technology-specific dependencies into compact rule updates, AnaGen reduces the overhead typically associated with analog layout migration while maintaining structural consistency across nodes.

5.8 Summary

This chapter proposed and validated a complete template-based layout automation methodology for analog and mixed-signal circuits. The presented framework, centered around the AnaGen (Analog Generator) engine and orchestrated through Python-driven control logic, enables rapid, scalable, and technology-portable layout generation across a wide range of analog building blocks.

The methodology was demonstrated on four representative circuits: an inverter, a single-ended differential amplifier, a comparator, and an Ibias generator (with and without trimming). These blocks span diverse placement sensitivities and routing constraints, providing a comprehensive evaluation of the framework's capabilities.

The chapter detailed the automated layout flow starting from schematic-driven constraint extraction, followed by template-based device placement and multiple routing strategies, including VSR routing, AnaGen routing, hybrid routing, and manual routing. Quantitative and qualitative comparisons highlighted the inherent trade-offs between development effort, execution time, layout area, and analog quality. AnaGen routing consistently delivered symmetry-aware and analog-robust layouts; VSR routing achieved the fastest execution and most compact areas, while the hybrid approach offered an effective compromise for blocks requiring both speed and precision. These results confirm that no single routing strategy is universally optimal, and that flexible strategy selection is essential for practical analog automation.

Technology porting experiments further validated the portability of the proposed approach. By isolating technology dependencies into rule files, existing AnaGen templates were migrated across different PDKs without modifying placement structures or routing topologies. This demonstrates that the framework supports scalable reuse and significantly reduces the overhead associated with migrating analog layouts to new technologies.

In summary, this chapter demonstrates that template-based layout automation provides a robust and industrially viable solution for accelerating analog layout design while preserving quality, consistency, and portability. The proposed methodology supports rapid iteration, systematic reuse, and efficient adaptation across circuits and technology generations, establishing a strong foundation for advanced analog automation and further research in layout synthesis and optimization.

Chapter 6

Conclusion

This thesis presented a comprehensive, industrial tool-driven framework for multi-level automation of analog integrated circuit (IC) design, spanning architecture exploration, schematic synthesis, optimization, and layout generation. Despite extensive research in electronic design automation, analog IC design continues to rely heavily on manual expertise due to complex performance trade-offs, sensitivity to device matching, and strong dependence on layout quality. The work presented in this thesis demonstrates that these challenges can be systematically addressed through a structured, constraint-aware, and reusable automation flow validated on multiple representative analog design benchmarks.

At the architecture level, the thesis demonstrated automated optimization of analog system-level designs using Python-based algorithms tightly integrated with industrial circuit simulators. A DC–DC buck converter was employed as a benchmark to validate the proposed approach, where constrained parametric sweeps enabled automatic sizing of key design parameters such as inductance and capacitance values. The results showed that a peak efficiency of approximately 86% could be achieved at 1 A load, remaining stable across a load range of 0.6 A to 3.5 A, while significantly reducing manual optimization time. This architecture-level automation enables rapid design-space exploration and supports early-stage design decisions without manual schematic development.

At the schematic level, the thesis introduced parameterized, technology-independent generators for the automatic synthesis of schematic diagrams for analog and mixed-signal building blocks. A resistor network connected to a comparator was used as a benchmark to demonstrate deterministic and variation-aware optimization, targeting layout area minimization while preserving voltage accuracy and statistical robustness. The framework was validated through large-scale Monte Carlo analysis exceeding 200 000 samples across all process corners, achieving a 100% pass rate. Post-layout validation confirmed close agreement between calculated and extracted values, with a voltage error below 1% and output variation within 3.34% (3σ), demonstrating the reliability of the proposed schematic-level automation. Notably, the automatically

generated schematic was successfully implemented in a silicon prototype, confirming the practical deployability of the proposed methodology. The framework was further extended to the automatic generation of configurable SAR ADC architectures, where block-level schematic templates enabled rapid assembly of complete converters based on user-defined specifications. This approach highlights the scalability and reusability of the framework across different circuit classes.

At the layout level, the thesis addressed one of the most critical challenges in analog design automation: enforcing symmetry, matching, and routing balance. The proposed framework integrates schematic-driven constraint extraction with analog-aware placement and template-based routing techniques. Particular emphasis was placed on symmetry-critical blocks, such as comparators, where matched device placement and balanced routing paths are essential for offset and timing performance. Comparative layout experiments demonstrated that unconstrained routing fails to preserve analog symmetry and matching, whereas template-based routing consistently achieves analog-quality layout results. For the comparator benchmark, automated layout generation is completed in under 5 minutes, compared to an initial manual development effort of 3–34 hours, while achieving a layout area of 1795 μm^2 with the hybrid routing strategy comparable to the manual reference of 1295 μm^2.

All stages of the proposed framework were implemented and validated using industrial EDA tools and design environments, ensuring practical relevance and deployability in real design environments. The experimental results confirm that the framework enables scalable reuse, consistent design quality, and substantial reductions in overall design time. By combining architecture-level optimization, schematic-level automation, and layout-aware constraint enforcement within a unified flow, this work demonstrates that systematic, multi-level automation of analog IC design is achievable without compromising performance or layout integrity.

Future work may extend the proposed framework toward closed-loop optimization flows, integration of machine learning techniques for constraint inference and layout refinement, and expansion to more complex mixed-signal and RF systems. The contributions of this thesis represent a significant step toward bridging the gap between academic research and industrial practice in analog integrated circuit design automation.

Bibliography

[1] J. M. Rabaey, *Low Power Design Essentials*. Springer Science & Business Media, 2009.

[2] W. M. Sansen, *Analog Design Essentials*. Springer Science & Business Media, 2007.

[3] G. G. E. Gielen and W. M. C. Sansen, "Modeling of the power-supply interactions of CMOS operational amplifiers using symbolic computation," in *Proc. IEEE Int. Workshop on Advances in Sensors and Interfaces (IWASI)*, 2019, pp. 29–34.

[4] L. Liu, M. Chen, W. Huang, X. Liao, and J. Xu, "A high current-efficiency rail-to-rail class-AB op-amp with dual-loop control," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 69, no. 11, pp. 4218–4222, 2022.

[5] A. Gupta and S. Singh, "Design of two-stage CMOS op-amp with high slew rate and high gain in 180 nm," in *Proc. 2nd Int. Conf. I-SMAC*, 2018, pp. 341–345.

[6] R. Harjani, "Systematic design of high-frequency gm-C filters," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 1999.

[7] S. P. Boyd, S.-J. Kim, and S. Mohan, "Geometric programming and its applications to EDA problems," DATE 2005 Tutorial.

[8] M. Ershov *et al.*, "Statistical analysis of random telegraph noise variability and extraction method for industrial ultra-scaled CMOS technologies," *IEEE Trans. Electron Devices*, vol. 61, no. 7, pp. 2075–2082, 2014.

[9] E. Lauwers and G. Gielen, "Systematic design of high-frequency gm-C filters," in *Analog Circuit Design*, Springer, 2002, pp. 21–45.

[10] Y. Yang, "Design of high-performance CMOS two-stage op-amps," *IEEE J. Solid-State Circuits*, vol. 29, no. 9, 1994.

[11] A. Andreta, L. F. Lavado Villa, Y. Lembeye, and J.-C. Crebier, "A novel automated design methodology for power electronics converters," *Electronics*, vol. 10, no. 3, p. 271, 2021.

[12] A.-T. Grăjdeanu, C. Răducan, C.-S. Pleşa, B. S. Kirei, and M. Neag, "Comparison of four design environments employed to analyze a switched-capacitor DC–DC converter," in *Proc. IEEE ISETC*, 2016, pp. 190–193.

[13] A. J. M. Cardoso, "Power electronics design methods and automation in the digital era," *IEEE Power Electronics Magazine*, vol. 7, no. 2, pp. 36–40, 2020.

[14] A. Bindra and A. Mantooth, "Modern tool limitations in design automation," *IEEE Power Electronics Magazine*, vol. 6, no. 1, pp. 28–33, 2019.

[15] Wei-Bing Bao and Jian-Yu Bao (2010). "Modeling and simulation of multilevel current source inverter based on SIMetrix/SIMPLIS," in *2010 International Conference on Computer Application and System Modeling (ICCASM)*, Taiyuan, China, pp. V4-466–V4-470, doi: 10.1109/ICCASM.2010.5620416.

[16] G.-R. Li, K. A. Kim, A. Roy, and T. G. Wilson (2021). "Examining Power Factor Correction Boost Converter Feedback Control Using SIMPLIS," in *2021 IEEE International Future Energy Electronics Conference (IFEEC)*, Taipei, Taiwan, pp. 1–6, doi: 10.1109/IFEEC53238.2021.9662029.

[17] J. Amanor-Boadu and E. Hammond (2024). "Leveraging SIMPLIS to Better Predict Server Platform Power Delivery Performance," in *2024 IEEE 28th Workshop on Signal and Power Integrity (SPI)*, Lisbon, Portugal, pp. 1–4, doi: 10.1109/SPI60975.2024.10539218.

[18] D. K. Ansari, G. Capodivacca, S. U. Amin, and A. Baschirotto, "Python and SIMETRIX/SIMPLIS based automated platform for the DC–DC converter with current mode control," in *Proc. IEEE DCIS*, 2024, pp. 1–4.

[19] M. Flaibani, C. Garbossa, E. Orietti, and A. Vecchiato, "System and method for controlling a switched-mode power supply," U.S. Patent 8,810,227, Aug. 19, 2014.

[20] M. del Mar Hershenson, S. P. Boyd, and T. H. Lee, "GPCAD: A tool for CMOS op-amp synthesis," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, 1998, pp. 296–303.

[21] Murmann, B. (2015). "Automation of Analog Circuit Design: Challenges and Opportunities," *IEEE Transactions on Circuits and Systems II: Express Briefs.*

[22] Yang, H. G., et al. (1994). "Design, Measurement and Analysis of CMOS Polysilicon TFT Operational Amplifiers," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 6, pp. 727–735.

[23] Hanna, R., Natarajan, A., and Murmann, B. (2021). "An Automated Design Flow for Analog Layout using Deep Reinforcement Learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 937–942.

[24] Li, J. et al. (2022). "An Open Source Automated End-to-End SAR ADC Compiler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[25] Ding, M. et al. (2018). "A hybrid design automation tool for SAR ADCs in IoT," *IEEE Transactions on Circuits and Systems I: Regular Papers*.

[26] Nikolić, B., et al. (2020). "Performance Comparison of BAG and Custom Generated Analog Layout for Single-Tail Dynamic Comparator," in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Glasgow, UK.

[27] Hammoud, A., Goyal, C., Pathen, S., Dai, A., Li, A., and others (2024). "Human Language to Analog Layout Using GLayout Layout Automation Framework," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA.

[28] Liu, B., Zhang, H., and others (2024). "LayoutCopilot: An LLM-Powered Multiagent Collaborative Framework for Interactive Analog Layout Design," arXiv:2406.18873 [cs.LG].

[29] Y. Wei, Y. Xu, K. Zhu, and Y. Lu (2024). "Universal Process Migration Solution of MAGICAL for Analog IC Layout Automation," in *2024 Conference of Science and Technology for Integrated Circuits (CSTIC)*, Shanghai, China, pp. 1–3, doi: 10.1109/CSTIC61820.2024.10531849.

[30] H. Chen et al. (2021). "MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII," *IEEE Design & Test*, vol. 38, no. 2, pp. 19–26, doi: 10.1109/MDAT.2020.3024153.

[31] B. Xu et al. (2019). "MAGICAL: Toward Fully Automated Analog IC Layout Leveraging Human and Machine Intelligence," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, CO, USA, pp. 1–8, doi: 10.1109/ICCAD45719.2019.8942060.

[32] H. Chen et al. (2021). "MAGICAL 1.0: An Open-Source Fully-Automated AMS Layout Synthesis Framework Verified With a 40-nm 1GS/s $\Delta\Sigma$ ADC," in *2021 IEEE Custom Integrated Circuits Conference (CICC)*, Austin, TX, USA, pp. 1–2, doi: 10.1109/CICC51472.2021.9431521.

[33] K. Zhu, H. Chen, M. Liu, and D. Z. Pan (2023). "Tutorial and Perspectives on MAGICAL: A Silicon-Proven Open-Source Analog IC Layout System," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 2, pp. 715–720, doi: 10.1109/TCSII.2022.3172869.

[34] Basso, D., et al. (2024). "Effective Analog ICs Floorplanning with Relational Graph Neural Networks and Reinforcement Learning," arXiv:2411.15212 [cs.AR].

[35] Della Rovere, S. J., Basso, D., Bortolussi, L., Videnovic-Misic, M., and Habal, H. (2025). "Enhancing Reinforcement Learning for the Floorplanning of Analog ICs with Beam Search," arXiv:2505.05059 [cs.AR].

[36] Basso, D., Bortolussi, L., Videnovic-Misic, M., and Habal, H. (2024). "Fast ML-driven Analog Circuit Layout using Reinforcement Learning and Steiner Trees," in *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, arXiv:2405.16951 [cs.AR].

[37] X. Xu, B. Cline, G. Yeric, B. Yu, and D. Z. Pan (2015). "Self-Aligned Double Patterning Aware Pin Access and Standard Cell Layout Co-Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 699–712, doi: 10.1109/TCAD.2015.2399439.

[38] M. Bio, S. M. Sondón, J. Sturm, and H. Pretl (2025). "AICL Co-Pilot: An Interactive Analog Integrated Circuit Layout Generator," in *2025 Austrochip Workshop on Microelectronics (Austrochip)*, Linz, Austria, pp. 65–68, doi: 10.1109/Austrochip67945.2025.11183684.

[39] X. Liu et al. (2023). "A Digital LDO in 22-nm CMOS With a 4-b Self-Triggered Binary Search Windowed Flash ADC Featuring Analog Layout Generator Framework," *IEEE Solid-State Circuits Letters*, vol. 6, pp. 101–104, doi: 10.1109/LSSC.2023.3265956.

[40] C.-M. Huang and S.-Y. Fang (2016). "Overlay-aware layout legalization for self-aligned double patterning lithography," in *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Hsinchu, Taiwan, pp. 1–4, doi: 10.1109/VLSI-DAT.2016.7482574.

[41] T. Dhar et al. (2020). "The ALIGN Open-Source Analog Layout Generator: v1.0 and Beyond," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, San Diego, CA, USA, pp. 1–2.

[42] K. Kunal et al. (2019). "INVITED: ALIGN – Open-Source Analog Layout Automation from the Ground Up," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, NV, USA, pp. 1–4.

[43] J. Poojary, R. S, S. S. Sapatnekar, and R. Harjani (2023). "Exploration of Design / Layout Tradeoffs for RF Circuits using ALIGN," in *2023 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, San Diego, CA, USA, pp. 57–60, doi: 10.1109/RFIC54547.2023.10186141.

[44] H. Hsu and Y. Peng (2025). "PosterO: Structuring Layout Trees to Enable Language Models in Generalized Content-Aware Layout Generation," in *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, pp. 8117–8127, doi: 10.1109/CVPR52734.2025.00760.

[45] Jehng, Yeu-Shen, Liang-Gee Chen, and Tai-Ming Parng. "ASG: Automatic schematic generator." *Integration* 11.1 (1991): 11-27.

[46] Kim, Nam-Hoon, et al. "RightTopologizer: an efficient schematic generator for multi-level optimization." *Proceedings of 13th Annual IEEE International ASIC/SOC Conference (Cat. No. 00TH8541)*. IEEE, 2000.

[47] Wu, Yuping. "Novel method of analog circuit schematic synthesis." *2009 IEEE 8th International Conference on ASIC*. IEEE, 2009.

[48] Garg, Bikram, Amarpal Singh, and Ashish Agrawal. "Efficient undo-redo with incremental netlist to schematic generation." *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, 2010.

[49] Meissner, Markus, and Lars Hedrich. "FEATS: Framework for explorative analog topology synthesis." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.2 (2014): 213-226.

[50] Greif, Matthias, Daniel Marolt, and Juergen Scheible. "gPCDS: An interactive tool for creating schematic module generators in analog IC design." *2016 12th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2016.

[51] Wittmann, Juergen, Carsten Wegener, and Fabio Rigoni. "Automatic Analog-on-Top Chip-Level Schematic Generation Based on Wire-by-Name Methodology Juergen Wittmann, Carsten Wegener." *ANALOG 2018; 16th GMM/ITG-Symposium*. VDE, 2018.

[52] Passerini, F., et al. "Anagen: A methodology for analog circuit generation." *IEEE CICC*. 2021.

[53] Hsu, Hung-Yun, and Mark Po-Hung Lin. "Automatic analog schematic diagram generation based on building block classification and reinforcement learning." *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*. 2022.

[54] Demiri, David, et al. "A procedural generator for the sizing and physical synthesis of a mosfet low-side driver." *2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, 2023.

[55] Emilia, Gheorghiță, et al. "A Procedural Sizing Approach for LDO Modules Using AnaGen Methodology." *2024 International Semiconductor Conference (CAS)*. IEEE, 2024.

[56] D. J. Walters and P. Johnson, “Optimal Resistor Networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 8, pp. 3015–3026, Aug. 2024.

[57] J. Wiley Crossley, "BAG: A designer-oriented framework for the development of AMS circuit generators," EECS Dept., Univ. California at Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-195, 2014.

[58] Dhar, Tonmoy, et al. "ALIGN: A system for automating analog layout." IEEE Design & Test 38.2 (2020): 8-18.

[59] Xu, Biying, et al. "Magical: Toward fully automated analog ic layout leveraging human and machine intelligence. In 2019 IEEE." ACM International Conference on Computer-Aided Design (ICCAD).

[60] Liu, Bingyang, et al. "LayoutCopilot: an LLM-powered multiagent collaborative framework for interactive analog layout design." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 44.8 (2025): 3126-3139.