Journal of Telecommunication, Electronic and Computer Engineering
ISSN: 2180 – 1843 e-ISSN: 2289-8131 Vol. 16 No. 4 (2024) 37 – 46
jtec.utem.edu.my
DOI: https://doi.org/10.54554/jtec.2024.16.04.006

AI-Driven Automation for Analog and Mixed-Signal Circuit Design: Schematic to GDSII Layout

Jiveya Sri Sockalingam[1], Yan Chiew Wong[1*], Ser Lee Loh[1], Ranjit Singh Sarban Singh[2], T. Joseph Sahaya Anand[3]
[1]Faculty of Electronics and Computer Technology and Engineering, Universiti Teknikal Malaysia Melaka (UTeM), 76100 Durian Tunggal Melaka, Malaysia.
[2]School of Engineering and Technology, Sunway University, Selangor, Malaysia.
[3]School of Computing, MIT Vishwaprayag University, Solapur, 413255, India.

Article Info

Article history:
Received Jul 1st, 2024
Revised Nov 5th, 2024
Accepted Dec 17th, 2024
Published Dec 20th, 2024

Index Terms:
Reinforcement Learning
Neural Network
Automatic analog circuit
Automated layout
Automated mixed signal
Op-amp
Ring Oscillator
ADC

Abstract

As technology propels forward and circuits evolve into intricate and complex designs, the traditional manual circuit design approach finds itself at a crossroads. With cutting-edge processes introducing many challenges, the journey from concept to creation has become increasingly arduous, demanding significant time investments. To overcome these challenges, automation emerges as a key innovation, accelerating product development while ensuring precision. This study explores analog circuit design by investigating the architecture of an analog and digital circuit generator and pioneering an automated synthesis method called "correct-by-construction". This innovative approach optimizes the design process while prioritizing accuracy from the outset. Additionally, this study evaluates the performance of analog generators, focusing on accuracy and circuit metrics using AutoCkt. Tools such as ALIGN for automated layout generation and OpenFASoC for digital design automation further enhance efficiency and accessibility in analog circuit design. The integration of these tools, alongside their compatibility with open-source CAD platforms, demonstrates significant advancements in automation. Furthermore, the development of a graphical user interface (GUI) provides a user-friendly platform for interacting with various functionalities related to circuit design and simulation, enhancing the overall design workflow.

*Corresponding Author: ycwong@utem.edu.my

I. INTRODUCTION

For decades, analog integrated circuit (IC) layouts have been considered as works of art due to their lack of standardized design styles, unlike their digital counterparts [1]. Crafting the correct analog layout is a complex and time-consuming task, demanding significant design effort despite the relatively small footprint of analog circuits. Traditionally, this process has been carried out using computer-aided design (CAD) tools, with designs either manually created by skilled designers or generated by computational tools. As technology advances, the challenges associated with innovating circuits within complex systems have increased. Detailed design rules and layout parasitic in advanced processes require substantial design time for circuits to reach the tape-out stage. This process has traditionally been managed by human designers, who iteratively adjust circuit parameters to meet design targets, relying heavily on their expertise to formulate equations and find solutions. To reduce time-to-market, it has become crucial to automate these time-consuming procedures in a simulation-efficient and accurate manner.

To meet the evolving requirements of modern System-on-Chip (SoC) design, automation is increasingly indispensable, particularly in the domains of analog circuit design, validation, and integration [2]. Manual circuit design, due to the myriad design criteria and specifications inherent in contemporary complex circuits and substantial process variability, is a challenging, time-consuming, and often inefficient endeavor [3]. Given these challenges, automated analog circuit generation emerges as a necessity. While ongoing initiatives aim to achieve fully automated end-to-end design using machine learning and other automation techniques, these efforts remain a work in progress and require verification for accuracy and practicality [4, 5]. The key processes in basic schematic circuit design involve determining the circuit's topology and the values of its elements, such as sizing to meet specific parameters. Model-based approaches stand out as prominent techniques in automated circuit sizing, encompassing non-convex polynomial optimization, deep neural networks (DNN), and geometric programming [6], [7].

In summary, previous work on automation presents promising and successful approaches to diverse aspects of the ASIC design process. Methodologies vary significantly based on the specific sub-problems they address, with distinctions in the level of automation, generalizability, accuracy, and interpretability of these approaches.

Aligned with these considerations, the primary objective of this paper is to develop an automatic circuit generator utilizing deep neural network (DNN) technology. This automatic circuit generator aims to create a comprehensive System-on-Chip (SoC) synthesis tool, guiding the process from user specifications to GDSII. This tool leverages innovative technology to automatically synthesize "correct-by-construction" Verilog descriptions for analog circuits, enabling a portable, single-pass implementation flow [4]. Additionally, the paper aims to evaluate and analyze the performance of analog generators in terms of accuracy and circuit performance.

II. LITERATURE REVIEW

Automated analog circuit optimization has progressed significantly, reducing the need for human expertise. Methods include reinforcement learning (RL) for parallel circuit design predictions [8], machine learning (ML) combining neural networks and RL for autonomous circuit sizing [9], and deep learning models for automated placement in circuit layout design [10]. Techniques like Generalized Differential Evolution 3 (GDE3) and Gaussian Processes efficiently optimize complex circuits with conflicting objectives [11]. Key steps in transforming a netlist into a layout involve partitioning the netlist [12][13][14], automated layout stitching [15], and customizable logic device layout generation [16]. Designing analog and mixed-signal (AMS) IP blocks on advanced technologies like FinFET or GAAFET is challenging due to the schematic-post-layout simulation gap, complex design rules, and reliability requirements. Recent research has focused on layout synthesis using ML advancements, with studies showing potential improvements using Boolean satisfiability-based routing algorithms and verifiable constraint languages [17].

Deep Reinforcement Learning (DRL) has been developed for designing analog-integrated circuits such as operational amplifiers. This method uses trial and error to find optimal designs, building circuits from basic components with custom rules and improving efficiency through hash tables and symbolic analysis, achieving effective designs in a few hours [18]. Similarly, reinforcement learning was used in AutoCkt to design two-stage operational amplifiers, showing efficiency in sparse subsampling and converging 25 times faster than standard methods, with a 40-fold speedup for subsequent circuits [19]. Another approach proposed a DNN-based framework for RL-inspired circuit optimization, demonstrating improved performance and reduced design time, although it relies on accurate DNN predictions and fine-tuning of hyperparameters [20].

The transformation of a netlist into a physical layout using electronic design automation (EDA) involves key steps such as partitioning, floor planning, placement, and routing. These processes are supported by advanced tools tailored for both digital and analog designs, ensuring functional correctness and compliance with manufacturing standards. Resolution enhancement techniques (RET) are also essential for refining layout quality [21]. Additionally, an open-source framework developed for automating analog layout generation using machine learning and graph-based recognition aimed to minimize human intervention but faced limitations due to proprietary Process Design Kits (PDKs) [22].

An Autonomous System-on-Chip (SoC) Design Framework introduced in 2021 promised faster design times and reduced Non-Recurring Engineering (NRE) costs. However, the framework was hindered by manual layout requirements and process-specific constraints [23].

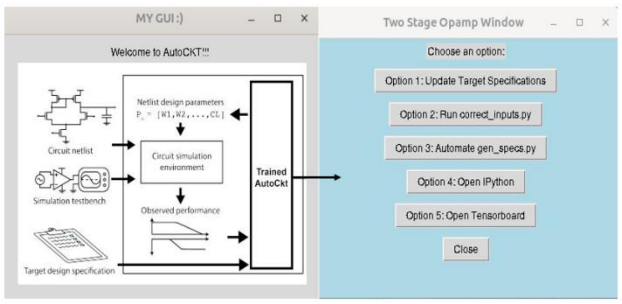
In this work, we propose an automated analog circuit generator leveraging multiple open-source Electronic Design Automation (EDA) tools such as AutoCkt, Xschem, ALIGN, and OpenFASOC. These tools were selected for their capabilities to incorporate deep neural networks, enabling the system to learn from experience and adapt to various circuit types. AutoCkt employs deep reinforcement learning to optimize circuit designs efficiently, while Xschem provides a versatile schematic capture tool. ALIGN facilitates analog layout automation, and OpenFASOC integrates machine learning techniques for circuit synthesis and optimization to automate analog mixed signals. Together, these tools enable a comprehensive and adaptive approach to analog circuit design, aligning with diverse design objectives and improving design efficiency and accuracy.

III. METHODOLOGY

This methodology integrates a system to automate and optimize the design of analog circuits, analog layouts, and mixed-signal ICs. By combining advanced machine learning techniques with traditional Electronic Design Automation (EDA) tools, the system enhances design efficiency and effectiveness. The integrated system includes four key components: AutoCkt, Xschem, ALIGN, and OpenFASoC, each providing unique functionalities to the workflow.

A. *Automated Analog Circuit Design*

AutoCkt is a machine learning framework designed to solve analog circuit design problems. By training on a sparse sub-sample of the design space, AutoCkt significantly reduces convergence time and efficiently meets various new design specifications. It takes three inputs: a circuit netlist, a simulation testbench, and target design specifications. Using this information, AutoCkt determines the circuit parameters to ensure the design meets the specified requirements. The GUI for AutoCkt allows users to easily select between two-stage op-amp and ring oscillator circuits and includes a reset button and a help button to assist novice users.


MY GUI :)
Welcome to AutoCKT!!!
Netlist design parameters
P_c = [W1,W2,....,CL]
Circuit netlist
Circuit simulation environment
Simulation testbench
Observed performance
Trained AutoCkt
Target design specification
Two Stage Opamp Window
Choose an option:
Option 1: Update Target Specifications
Option 2: Run correct_inputs.py
Option 3: Automate gen_specs.py
Option 4: Open IPython
Option 5: Open Tensorboard
Close
Figure 1: Automated Analog Circuit Design

The graphical user interface of AutoCkt provides a seamless navigation experience, allowing users to easily transition between different functions. When selecting the Two Stage Opamp option, users are directed to the Two Stage Opamp Options Window, where they can access five options tailored to their needs, such as updating target specifications, running scripts, and accessing external tools like IPython and Tensorboard. This structured approach enhances productivity and user satisfaction by making it easy to perform tasks related to circuit design. The intuitive layout and clear labeling help users quickly locate and utilize the desired functionalities to optimize circuit parameters based on target design specifications.

B. AutoCkt Framework

In AutoCkt, the system comprises two main components: the RL agent, responsible for decision-making, and the circuit simulation environment, where decisions are executed and evaluated. Reinforcement Learning (RL) is a subset of machine learning that involves an agent interacting with its environment through trial and error, mimicking human learning. This approach uses a simulation-in-the-loop framework, requiring validation of outputs against a reliable simulation source.

At each step, the RL agent, equipped with a neural network, observes the state of the environment and selects an action from a probabilistic distribution. This process balances exploration and exploitation, enabling the agent to discover new possibilities while leveraging its existing knowledge to maximize rewards. After selecting an action, the environment transitions to a new state, and the reward for that action is calculated. This iterative process continues across a trajectory of multiple steps, with rewards accumulating until the target is achieved or a maximum step limit is reached.

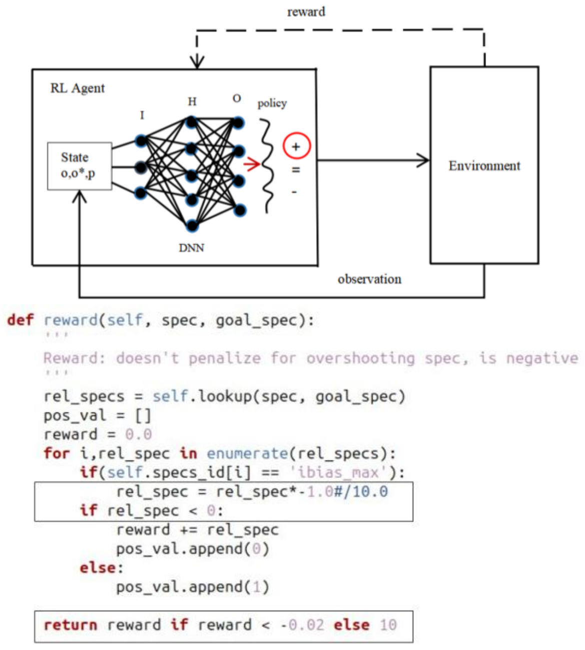
In this scenario, N refers to the array of parameters that need to be adjusted within a circuit, including transistor lengths and widths. M refers to the various target design specifications that the circuit aims to achieve post-training, such as gain and bandwidth. Together, these variables encompass a wide range of performance goals and requirements. The parameter space, denoted as $x \in Z^N$, and the target design specifications space, labeled as $y \in R^M$, are standardized to a predefined range, ensuring uniformity and comparability across different parameters and specifications. Initially spanning R^N, the parameter space is discretized into K grids: $\{x \in Z^{N:}\ 0 \le x_i \le K,\ i = 1,\ldots,N\}$. The system generates L trajectories, each with targets chosen from the set of M specifications. The reward for each trajectory is calculated by summing the rewards for each action, which are determined by the differences between the actual circuit performance (o) and its target specification (o*), normalized and bounded, and then multiplied by a scaling constant, β as shown below:

$$r_o = min\left(\beta \frac{o - o*}{o + o*}, 0\right) \tag{1}$$

$$r = \sum_{o \in o} r_o \tag{2}$$

For metrics being maximized, β is set to 1, while for metrics being minimized, β is set to -1. For minimized metrics that are not strict constraints, such as power, β is assigned a value within the range $-1 < \beta < 0$. More negative rewards are given for greater deviations below the target metrics being maximized or for exceeding the metrics being minimized. Over satisfying any one metric results in no reward for the agent, with $\boldsymbol{r_o}$ being set to zero in such cases.

Figure 2 presents a reward method that calculates a reward based on the differences between current specifications (‘spec’) and desired goal specifications (‘goal_spec’). It begins by computing these differences using a 'lookup' method and initializes an empty list ‘pos_val’ along with a reward variable set to 0.0. The method iterates through each relative specification, adjusting the handling for ‘ibias_max’ by inverting its sign. For negative relative specifications, their values are added to the reward, and 0 is appended to ‘pos_val’; otherwise, 1 is appended. Finally, if the computed reward is less than -0.02, it returns the reward, otherwise, it defaults to 10, ensuring no penalty for exceeding the specification in AutoCkt. This approach enhances the efficiency of the proximal policy optimization algorithm’s training process.



```
def reward(self, spec, goal_spec):
    '''
    Reward: doesn't penalize for overshooting spec, is negative
    '''
    rel_specs = self.lookup(spec, goal_spec)
    pos_val = []
    reward = 0.0
    for i,rel_spec in enumerate(rel_specs):
        if(self.specs_id[i] == 'ibias_max'):
            rel_spec = rel_spec*-1.0#/10.0
        if rel_spec < 0:
            reward += rel_spec
            pos_val.append(0)
        else:
            pos_val.append(1)

    return reward if reward < -0.02 else 10
```

Figure 2: Calculating Reward in AutoCkt

In AutoCkt, a three-layer neural network with 64 neurons per layer facilitates the mapping between states and actions in the reinforcement learning framework, as depicted in Figure 2. This architecture was chosen for its ability to manage the complexity of adjusting parameters to meet target specifications without fixating excessively on specific circuit components. Within this reinforcement learning algorithm, the network takes inputs including current performance (o), target specifications (o*), and current parameters (p). It outputs a probabilistic distribution from which samples are drawn to determine whether to increase, decrease, or maintain each circuit parameter. This approach enables adaptive parameter adjustments based on the circuit’s current state and desired performance goals.

When the agent is deployed, it is trained to create paths using distinct target requirements derived from o* (‘target design specification’). This training setting may differ from the simulation environment. After comparing the target design specification (o*) with the final specification o produced by the trajectory, the corresponding counter is incremented.

C. Automated Analog Layout Generation

ALIGN automates the layout generation process with a user-friendly graphical interface (GUI), as shown in Figure 3. This GUI integrates tools like Xschem for schematic capture, facilitating hierarchical circuit representation and netlist creation in formats such as VHDL, spice netlist, Verilog, or tEDAx. Users can select and run these tools via radio buttons on the main window, which also features a welcome message, option display, and navigation buttons for user convenience. This setup streamlines operations, reduces manual effort, and improves layout accuracy, ensuring a smooth and efficient user experience.

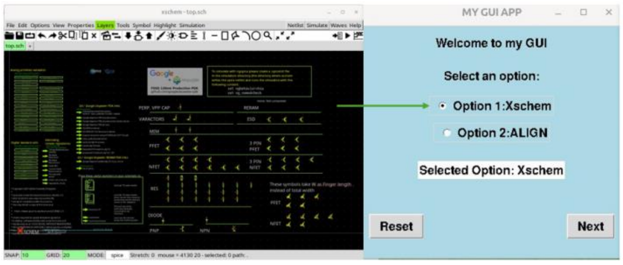
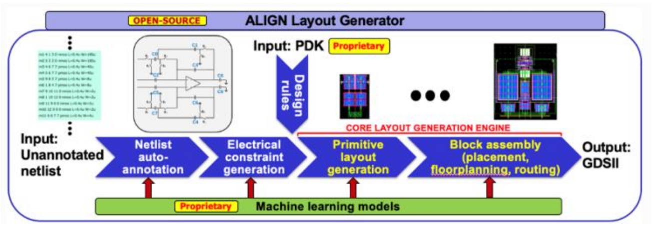


Figure 3: Automated Layout Generation based on Xschem and ALIGN.

D. ALIGN Framework

ALIGN is an open-source automatic layout generator for analog circuits. Figure 4 shows the five key components that make up the ALIGN flow:


OPEN-SOURCE
ALIGN Layout Generator
Input: PDK
Proprietary
Design rules
Input: Unannotated netlist
Netlist auto-annotation
Electrical constraint generation
CORE LAYOUT GENERATION ENGINE
Primitive layout generation
Block assembly (placement, floorplanning, routing)
Output: GDSII
Proprietary
Machine learning models
Figure 4: Key Modules of ALIGN [24]

- Design Rule Capture translates the proprietary Process Design Kit (PDK) into guidelines for the layout generator. Simplified versions for advanced process nodes (10 nm, 7 nm, 22 nm) enable layout tools to understand PDK features, including enforced grid stops, minimum length design rules, metal spacing, and width/spacing grids for each layer.
- Netlist auto-annotation in ALIGN identifies geometric constraints for each block and organizes passives and transistors into building blocks. Representing the netlist as a graph, it uses machine learning to recognize conventional structures, mimicking the pattern recognition abilities of experienced designers.
- Electrical Constraint Generation translates performance limitations, such as maximum route lengths, into layout constraints for sub-blocks. These guidelines define parameters like route lengths and parasitic requirements, ensuring compliant and optimized layouts at all hierarchical levels.
- Parameterized Layout Generation at the lowest ALIGN hierarchy level automatically generates layouts for primitives based on variables like transistor size, MOM capacitance, and serpentine resistance. Parameterized templates, adhering to PDK grids, ensure design rule compliance and correct PDK abstraction interfacing.
- Block Assembly organizes blocks according to the design hierarchy, with primitives using fixed-shape placement methods to generate multiple layout options. Higher levels use flexible forms and placement algorithms to create compact layouts, while adhering to geometric and electrical constraints.

E. Mixed Signal IC Design

Open-Source Fully Autonomous SOC, also known as OpenFASOC, is an open-source initiative revolutionizing semiconductor design by automating the entire flow for digital and mixed-signal systems-on-chip (SoCs). It focuses on generating analog components from high-level specifications using a combination of machine learning and traditional EDA techniques. OpenFASOC includes a graphical user interface (GUI), as shown in Figure 5, that simplifies tasks such as file uploading, viewing Verilog files, logs, results, and generated GDS files from ALIGN processes with a single click. This interface enhances efficiency and productivity in circuit design workflows, and is designed to be user-friendly for individuals with minimal technical expertise.

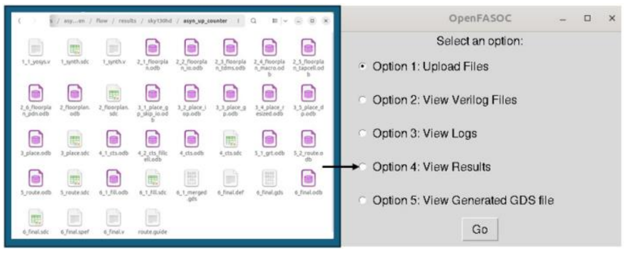


Figure 5: Automated Mixed Signal Circuit based on OpenFASoC.

F. OpenFASoC Framework

Figure 6 provides a high-level overview of the OpenFASoC framework. The initial setup involved creating aux-cells and generator models for the process design kit (PDK), a step performed once. The process began by translating high-level user intent into analog specifications that adhered to user constraints. This included generating the SoC layout by assembling the design components, executing an automatic place and route flow, and utilizing block generators as needed. This comprehensive process transformed user requirements into a finalized SoC layout efficiently within the automated framework of OpenFASoC.

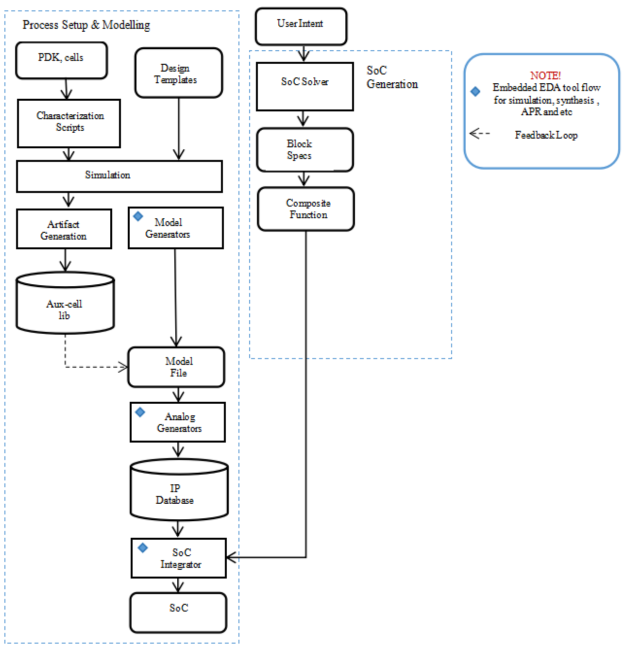


Figure 6: FASoC Architecture

OpenFASoC employed a synthesisable cell-based approach to create analog blocks, significantly reducing manual layout and verification efforts, as depicted in Figure 7. These small analog circuits, called aux-cells, augmented the standard cell library with essential analog capabilities required by the generators. The process integrated PDK characterization scripts with design templates to streamline aux-cell creation. The templates encapsulated the exact behavior of each aux-cell without PDK-specific data, while characterization scripts extracted technology-specific parameters to adjust template knobs. These knobs controlled device type, transistor size, and other circuit variables within the template. The resulting aux-cell generation included essential files like timing libraries, netlists, and layouts, facilitating traditional synthesis and automatic place and route (APR) seamlessly.

The analog generators in OpenFASoC used models to predict performance and determine design parameters for optimized block designs that met user-defined requirements. These models relied on parameterized templates containing aux-cells as their foundation. Each generator utilized a distinct model, crafted through a combination of design space exploration, machine learning, and mathematical formulas. This modeling process was executed once per PDK, and the outcomes were stored in respective model files, as illustrated in Figure 7.

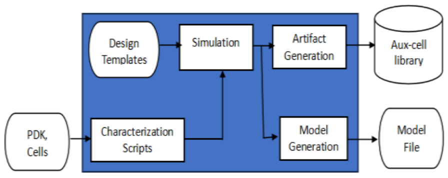


Figure 7: Process Setup and Modelling in OpenFASoC

Block designs and composite designs in OpenFASoC were stored in the IP-XACT format for comprehensive description. Additional analog data, simulation, and verification information were recorded in an expanded format [22]. The SoC integrator began by assembling the composite design and converting it into a structural Verilog format compatible with computer simulation programs. The final validated GDS file was generated by processing this structural Verilog through the embedded tool flow, incorporating all necessary artifacts from the database. This standardized flow was employed universally by all generators (aux-cell, model, and analog) across the framework.

In OpenFASoC, the initial stage involved feeding the Process Design Kit (PDK), cells, models, and block specifications into the Verilog generation module, as depicted in Figure 8. This module generated a synthesizable Verilog description of the block that met input specifications, utilizing models for accuracy. It also produced guidance data in a vendor-independent format. The macro generation phase created macros suitable for integration into larger SoC designs, forwarding the Verilog and guidance data to a digital flow. Here, processes like synthesis, automatic place and route (APR), design rule check (DRC), and layout versus schematic (LVS) verification were executed. Finally, macro validation ensured comprehensive verification and reporting of the created block, including parasitic extraction, SPICE simulations, and requirement checks across the entire circuit.

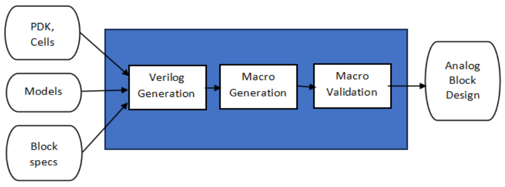
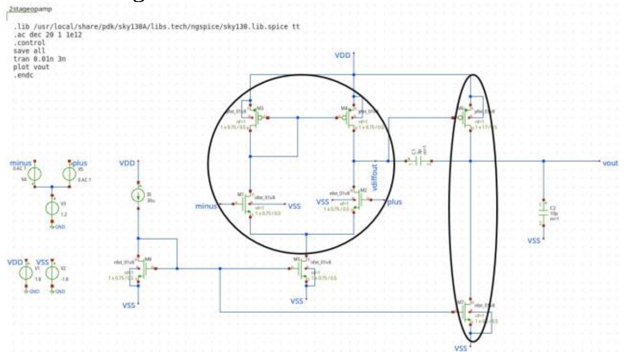


Figure 8: SoC Generator in OpenFASoC.

IV. RESULT AND DISCUSSION

A. Optimizing 2-stage Op-amp in Automated Analog Circuit Design

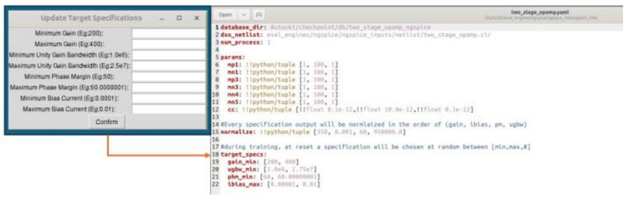
An automated analog circuit design was evaluated through the implementation and testing of a 2-stage operational amplifier using Xschem, as shown in Figure 9. The schematic depicts the operational amplifier circuit that includes a differential stage, which comprises a differential pair and a current mirror. Additionally, it features an amplification stage to enhance the gain.



```
.lib /usr/local/share/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.ac dec 20 1 1e12
.control
save all
tran 0.01n 3n
plot vout
.endc
```

Figure 9: Schematic of two stage Operational Amplifier created by using Xschem.

When users selected Option 1 in the Two Stage Op-amp window, they encountered an "Update Target Specifications" dialog box as shown in Figure 10. This feature allowed users to input specific parameters aligned with their requirements. It provided the capability to customize and fine-tune target specifications for their circuit designs directly. By enabling direct user input, the interface supported real-time adjustments, empowering users to refine their design criteria precisely. This interactive capability enhanced the design process by providing flexibility for iterative refinement based on evolving needs and preferences.

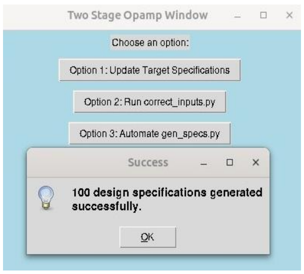


Minimum Gain (Eg:200):
Maximum Gain (Eg:400):
Minimum Unity Gain Bandwidth (Eg:1.0e6):
Maximum Unity Gain Bandwidth (Eg:2.5e7):
Minimum Phase Margin (Eg:60):
Maximum Phase Margin (Eg:60.0000001):
Minimum Bias Current (Eg:0.0001):
Maximum Bias Current (Eg:0.01):

```
database_dir: Autockt/checkpoint/db/two_stage_opamp_ngspice
dsn_netlist: eval_engines/ngspice/ngspice_inputs/netlist/two_stage_opamp.cir
num_process: 1

params:
  mp1: !!python/tuple [1, 100, 1]
  mn1: !!python/tuple [1, 100, 1]
  mp3: !!python/tuple [1, 100, 1]
  mn3: !!python/tuple [1, 100, 1]
  mn4: !!python/tuple [1, 100, 1]
  mn5: !!python/tuple [1, 100, 1]
  cc: !!python/tuple [!!float 0.1e-12,!!float 10.0e-12,!!float 0.1e-12]

#Every specification output will be normlaized in the order of (gain, ibias, pm, ugbw)
normalize: !!python/tuple [350, 0.001, 60, 950000.0]

#during training, at reset a specification will be chosen at random between [min,max,#]
target_specs:
  gain_min: [200, 400]
  ugbw_min: [1.0e6, 2.75e7]
  phm_min: [60, 60.00000001]
  ibias_max: [0.0001, 0.01]
```

Figure 10: Update Target Specification for two stage op-amp.

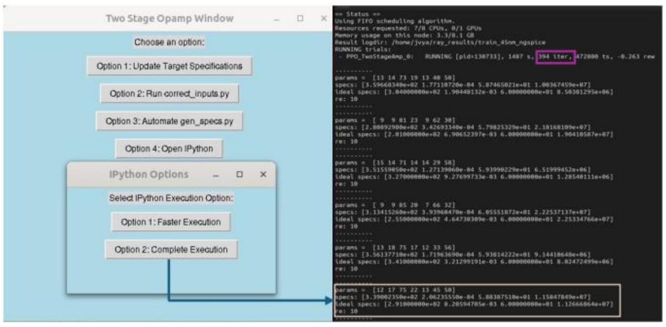
Selecting Option 2 in the Two Stage Op-amp window triggered the automatic execution of the script correct_inputs.py, which specifically involved navigating to the circuit netlist directory for training purposes. Moreover, if Option 3 was chosen in the Two Stage Op-amp window, it prompted the user to input the desired number of design specifications. By confirming with a click on OK, the system updated the specifications accordingly, as shown in Figure 11. Upon completion, a message indicated whether the generation process was successful or not.


Two Stage Opamp Window
Choose an option:
Option 1: Update Target Specifications
Option 2: Run correct_inputs.py
Option 3: Automate gen_specs.py
Success
OK

100 design specifications generated successfully.

Figure 11: Generating Design Specifications for Two Stage Op-amp.

Option 4 in the Two Stage Op-amp window allowed users to choose between two execution modes: “Faster Execution” and “Complete Execution”, as depicted in Figure 12. Opting for Faster Execution triggered a streamlined process with 50 training iterations, providing a quick overview of AutoCKT’s functionality. This mode automatically ended after the specified iterations, offering a brief insight into the process. In contrast, selecting “Complete Execution” initiated a comprehensive cycle that continued until the system achieved the desired specifications and parameters. This mode enabled a thorough exploration of AutoCKT’s capabilities, allowing users to monitor its performance until achieving the desired outcome.


Two Stage Opamp Window
Choose an option:
Option 1: Update Target Specifications
Option 2: Run correct_inputs.py
Option 3: Automate gen_specs.py
Option 4: Open IPython
IPython Options
Select IPython Execution Option:
Option 1: Faster Execution
Option 2: Complete Execution

```
== Status ==
Using FIFO scheduling algorithm.
Resources requested: 7/8 CPUs, 0/1 GPUs
Memory usage on this node: 3.3/8.1 GB
Result logdir: /home/jvya/ray_results/train_45nm_ngspice
RUNNING trials:
 - PPO_TwoStageAmp_0:   RUNNING [pid=130733], 1487 s, 394 iter, 472800 ts, -0.263 rew
-----------
params =  [13 14 73 19 13 40 50]
specs: [3.59668340e+02 1.77110720e-04 5.87465021e+01 1.00367459e+07]
ideal specs: [3.04000000e+02 1.90448132e-03 6.00000000e+01 8.50381295e+06]
re: 10
-----------
-----------
params =  [ 9  9 81 23  9 62 30]
specs: [2.80892980e+02 3.42693340e-04 5.79825329e+01 2.18168109e+07]
ideal specs: [2.01000000e+02 6.90652397e-03 6.00000000e+01 1.90410587e+07]
re: 10
-----------
-----------
params =  [15 14 71 14 14 29 50]
specs: [3.51559850e+02 1.27139060e-04 5.93990229e+01 6.51999452e+06]
ideal specs: [3.27000000e+02 9.27699733e-03 6.00000000e+01 1.28540111e+06]
re: 10
-----------
-----------
params =  [ 9  9 85 28  7 66 32]
specs: [3.13415260e+02 3.93960470e-04 6.05551872e+01 2.22537137e+07]
ideal specs: [2.55000000e+02 4.64738309e-03 6.00000000e+01 2.25334766e+07]
re: 10
-----------
-----------
params =  [13 18 75 17 12 33 56]
specs: [3.56137710e+02 1.71963690e-04 5.93814222e+01 9.14418648e+06]
ideal specs: [3.41000000e+02 3.21299191e-03 6.00000000e+01 8.02472499e+06]
re: 10
-----------
-----------
params =  [12 17 75 22 13 45 50]
specs: [3.39002350e+02 2.06235550e-04 5.88387510e+01 1.15847849e+07]
ideal specs: [2.91000000e+02 8.20594705e-03 6.00000000e+01 1.12666864e+07]
re: 10
-----------
```

Figure 12: Design parameters, specs and ideal specs achieved during training.

Figure 12 also shows the sequence of parameters obtained for a 2-stage op-amp after optimization. The optimized parameters revealed that the width of the PMOS transistor “mp1” was 12, the width of the NMOS transistor “mn1” was 17, and then the width of the PMOS transistor “mp3” was 75. Furthermore, additional optimized parameters included widths of 22 for transistor “mn3”, 13 for transistor “mn4”), 45 for transistor “mn5”, and 50 for the compensation capacitor “cc” within the circuit.

The GUI’s performance was evaluated based on the speed of integrated processes and the efficiency of design optimization workflows. Users found that the GUI facilitated rapid execution of essential tasks, such as updating target specifications and running validation scripts. The option to choose between faster and complete IPython execution provided flexibility, with the faster option delivering quick results within a limited number of iterations. Automated processes, including generating design specifications and running validation scripts, streamlined the circuit design process. Integration of external tools like Tensorboard enabled users to monitor the RL agent’s performance in ongoing simulations and track optimization progress using saved checkpoint files, as shown in Figure 13.

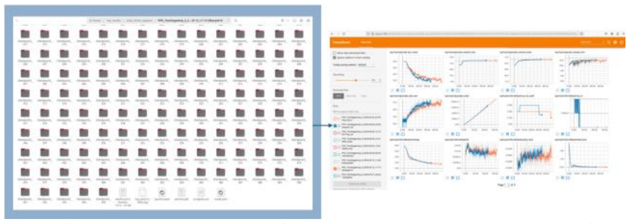
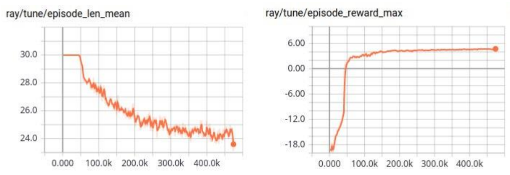


Figure 13: RL agent’s performance monitoring in Tensorboard.

For instance, Figure 14 (a) shows the mean reward over total environment steps. At the beginning of the training process, the agent required a significant number of steps to complete circuit optimization. As training progressed, the agent learned to optimize circuit parameters more efficiently, requiring fewer steps. As a result, the average episode length decreased. This decreasing trend indicated that the agent was improving in its decision-making capabilities for optimizing circuit parameters and was meeting the design specifications faster than through the manual steps. Figure 14(b) shows the maximum reward obtained by the reinforcement learning agent during the training. Initially, the reward increased as the agent learned how to optimize the circuit parameters to meet target design specifications. Subsequently, the reward stabilized, indicating that the agent has become proficient in the optimization process.



(a) Ray episode length mean (b) Ray episode reward maximum

Figure 14: Performance Analysis of AutoCkt

B. *3-stage Ring Oscillator in Automated Layout Design*

Selecting Xschem directed users to a new window with additional sub-options such as browsing and selecting schematics or SPICE files to review their contents and analyze pre- and post-layout simulation results through graphical representations. Each sub-option was tailored to improve users’ ability to visualize and analyze various aspects of their circuit designs.

When users selected the option to view the schematic in the Xschem window, the GUI displayed an image of the schematic, specifically a ring oscillator, as shown in Figure 15. From the schematic, users were able to view the generated netlist for the ring oscillator. This visual representation offered users a detailed overview of the circuit's design, making it easier to understand and verify the components and connections. Direct visualization of the schematic enabled users to swiftly identify potential issues or areas for improvement in their design, thereby improving the overall design and simulation workflow.

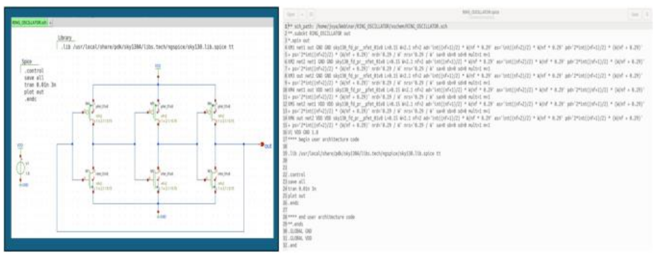


Figure 15: Viewing 3-stage Ring Oscillator’s schematic and generated netlist in Xschem

Following this, when users chose to view folders or files in the Xschem window, the GUI presented a file browsing interface, as depicted in Figure 16. This interface allowed users to navigate their directory structure and select files. Upon double-clicking a file, its contents were displayed in a text widget, enabling users to review detailed information such as circuit design specifics and simulation parameters. This feature ensured users could verify that the correct files were selected and that their content aligned with design expectations. It enhanced file management and verification efficiency, thereby improving the overall effectiveness of the design and simulation process.

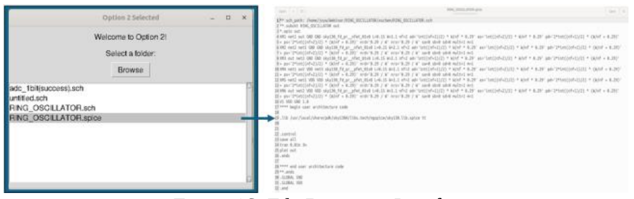
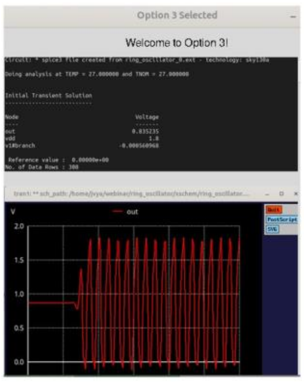
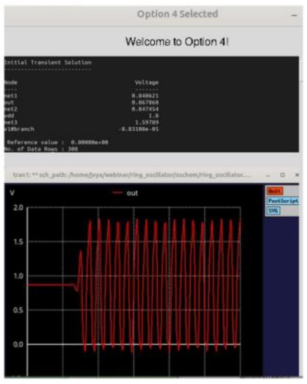


Figure 16: File Browsing Interface

Figure 17 (a) demonstrated how selecting the option to view pre-layout results in the Xschem window presented simulation outcomes before generating the physical layout. Users were able to visualize waveform images and other graphical data illustrating the circuit's behavior under test conditions, such as the transient response or frequency characteristics of a ring oscillator. These results validated the circuit's functionality in its schematic form, ensuring it met specifications before proceeding to layout, thereby saving time. Subsequently, option 4 allowed users to view post-layout results, as shown in Figure 17 (b), indicating simulation outcomes after completing the physical layout. Like pre-layout results, these visuals provided critical insights into the circuit's performance in its final form, verifying compliance with design specifications. They also helped identify issues caused by layout parasitic or other layout-related factors that may not have been evident during pre-layout simulation. Moreover, the consistency in periodic simulation output between pre-layout and post-layout analyses further supported the robustness and reliability of the oscillator design.


Option 3 Selected
Welcome to Option 3!
tran1: ** sch_path: /home/jvya/webinar/ring_oscillator/xschem/ring_oscillator...
V
out
Quit
PostScript
SVG
2.0
1.5
1.0
0.5
0.0

```
Circuit: * spice3 file created from ring_oscillator_0.ext - technology: sky130a

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000

Initial Transient Solution
--------------------------

Node                                   Voltage
----                                   -------
out                                   0.835235
vdd                                        1.8
v1#branch                         -0.000560968

 Reference value :  0.00000e+00
No. of Data Rows : 308
```


Option 4 Selected
Welcome to Option 4!
tran1: ** sch_path: /home/jvya/webinar/ring_oscillator/xschem/ring_oscillator...
V
out
Quit
PostScript
SVG
2.0
1.5
1.0
0.5
0.0

```
Initial Transient Solution
--------------------------

Node                                   Voltage
----                                   -------
net1                                  0.848621
out                                   0.867868
net2                                  0.847454
vdd                                        1.8
net3                                   1.59789
v1#branch                         -8.83108e-05

 Reference value :  0.00000e+00
No. of Data Rows : 308
```

(a) Pre-layout simulation (b) Post-Layout simulation

Figure 17: Pre-and Post layout simulation results

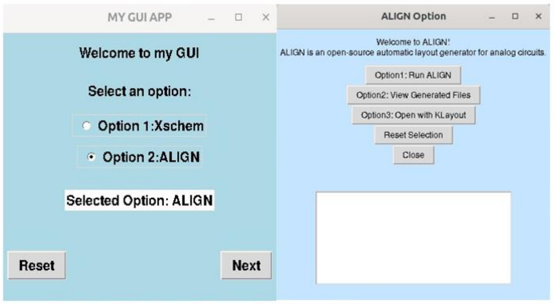


Figure 18: ALIGN options

Similarly, choosing the ALIGN option opened a dedicated window for layout generation and visualization tasks, as depicted in Figure 18. This interface introduced ALIGN's capabilities, enabling users to execute the tool, select a circuit, generate layout designs using Docker, and view the resulting .gds and .lef files directly. These files were accessible for detailed examination in KLayout without requiring manual intervention. The interface also provided options to reset file selections and close the window, enhancing workflow efficiency and user experience.

Upon clicking "Choose Circuit" in Figure 19, a directory selection window opened, showing the contents of '/home/jvya/mygui/ALIGN-public/ALIGN-pdk /sky130/examples'. Users could select a specific circuit folder from this interface, which updated the path displayed at the bottom of the window. This selected path was then dynamically integrated for running the ALIGN tool with the correct directory.

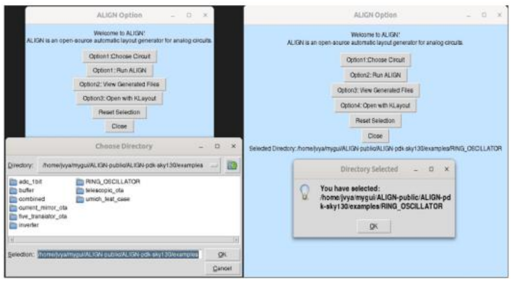
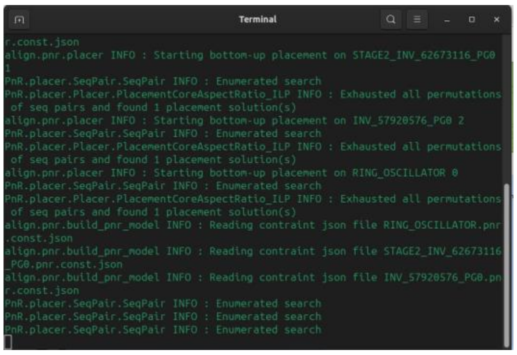

ALIGN Option
Welcome to ALIGN!
ALIGN is an open-source automatic layout generator for analog circuits.
Option1:Choose Circuit
Option1:Run ALIGN
Option2: View Generated Files
Option3: Open with KLayout
Reset Selection
Close
Choose Directory
Directory: /home/jvya/mygui/ALIGN-public/ALIGN-pdk-sky130/examples
adc_1bit
buffer
combined
current_mirror_ota
five_transistor_ota
inverter
RING_OSCILLATOR
telescopic_ota
umich_test_case
Selection: /home/jvya/mygui/ALIGN-public/ALIGN-pdk-sky130/examples
OK
Cancel
ALIGN Option
Welcome to ALIGN!
ALIGN is an open-source automatic layout generator for analog circuits.
Option1:Choose Circuit
Option2: Run ALIGN
Option3: View Generated Files
Option4: Open with KLayout
Reset Selection
Close
Selected Directory: /home/jvya/mygui/ALIGN-public/ALIGN-pdk-sky130/examples/RING_OSCILLATOR
Directory Selected
You have selected: /home/jvya/mygui/ALIGN-public/ALIGN-pdk-sky130/examples/RING_OSCILLATOR
OK

Figure 19: Choosing Circuit in ALIGN

One of the significant enhancements in this GUI application was its seamless integration with ALIGN. Users were able to execute ALIGN processes with a single click, facilitating the generation of GDS and LEF files, as shown in Figure 20. This integration simplified the process of running ALIGN and offered immediate access to the generated files, which could be viewed and analyzed using KLayout. This streamlined workflow supported efficient GDS layout generation and inspection, essential for advanced circuit design and analysis.

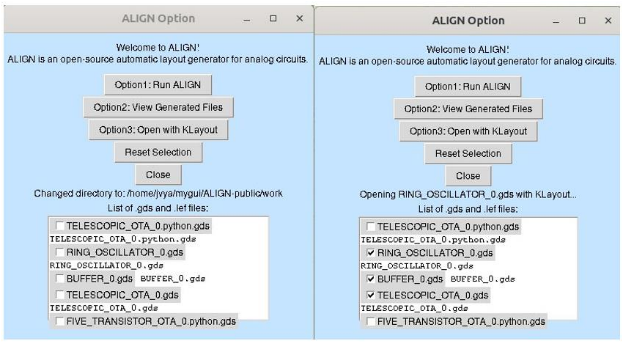


```
r.const.json
align.pnr.placer INFO : Starting bottom-up placement on STAGE2_INV_62673116_PG0
1
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
PnR.placer.Placer.PlacementCoreAspectRatio_ILP INFO : Exhausted all permutations
 of seq pairs and found 1 placement solution(s)
align.pnr.placer INFO : Starting bottom-up placement on INV_57920576_PG0 2
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
PnR.placer.Placer.PlacementCoreAspectRatio_ILP INFO : Exhausted all permutations
 of seq pairs and found 1 placement solution(s)
align.pnr.placer INFO : Starting bottom-up placement on RING_OSCILLATOR 0
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
PnR.placer.Placer.PlacementCoreAspectRatio_ILP INFO : Exhausted all permutations
 of seq pairs and found 1 placement solution(s)
align.pnr.build_pnr_model INFO : Reading contraint json file RING_OSCILLATOR.pnr
.const.json
align.pnr.build_pnr_model INFO : Reading contraint json file STAGE2_INV_62673116
_PG0.pnr.const.json
align.pnr.build_pnr_model INFO : Reading contraint json file INV_57920576_PG0.pn
r.const.json
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
PnR.placer.SeqPair.SeqPair INFO : Enumerated search
```

Figure 20: Running ALIGN

Users could select multiple generated GDS and LEF files conveniently using checkboxes in the GUI, as illustrated in Figure 21, and open them simultaneously in KLayout for visualization by clicking the “Open with KLayout” button. This streamlined process allowed users to efficiently inspect and compare layout designs, ensuring accuracy and consistency across different versions. With all selected files readily available in KLayout, users were able to perform thorough analyses, verify connections, and validate their designs with ease. By integrating file selection and visualization within the GUI, this feature enhanced usability, accelerated the design validation process, and contributed to improved productivity and design quality.

Overall, this GUI seamlessly integrated file browsing, content display, and visualization features, simplifying the management and validation of design and simulation processes. The capability to open generated files directly in KLayout, as depicted in Figure 22, enhanced usability by providing a comprehensive environment for analog circuit design and layout generation. This streamlined workflow enabled a thorough review of design schematics and simulation results, promoting efficiency and productivity throughout the design process.


ALIGN Option
Option1: Run ALIGN
Option2: View Generated Files
Option3: Open with KLayout
Reset Selection
Close
ALIGN Option
Option1: Run ALIGN
Option2: View Generated Files
Option3: Open with KLayout
Reset Selection
Close

```
Welcome to ALIGN!
ALIGN is an open-source automatic layout generator for analog circuits.
Changed directory to: /home/jvya/mygui/ALIGN-public/work
List of .gds and .lef files:
☐ TELESCOPIC_OTA_0.python.gds
TELESCOPIC_OTA_0.python.gds
☐ RING_OSCILLATOR_0.gds
RING_OSCILLATOR_0.gds
☐ BUFFER_0.gds  BUFFER_0.gds
☐ TELESCOPIC_OTA_0.gds
TELESCOPIC_OTA_0.gds
☐ FIVE_TRANSISTOR_OTA_0.python.gds
```

```
Welcome to ALIGN!
ALIGN is an open-source automatic layout generator for analog circuits.
Opening RING_OSCILLATOR_0.gds with KLayout...
List of .gds and .lef files:
☐ TELESCOPIC_OTA_0.python.gds
TELESCOPIC_OTA_0.python.gds
☑ RING_OSCILLATOR_0.gds
RING_OSCILLATOR_0.gds
☑ BUFFER_0.gds  BUFFER_0.gds
☑ TELESCOPIC_OTA_0.gds
TELESCOPIC_OTA_0.gds
☐ FIVE_TRANSISTOR_OTA_0.python.gds
```

Figure 21: View Generated Files

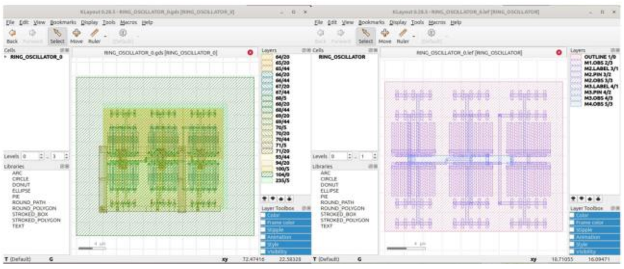


Figure 22: Ring Oscillator’s GDS Layout and Lef View in KLayout

C. *Integrating 3-Stage Ring Oscillator and 1-Bit ADC using Automated Mixed Signal IC Design.*

When users select Option 1 Upload Files, a new window opened, providing functionalities to upload various types of files essential for the OpenFASOC workflow. First, uploading GDS and LEF files allowed users to select and upload respective GDS and LEF files to the specified directory, facilitating the design process as shown in Figure 23. These LEF files were essential for layout and design rules. Additionally, the Upload Verilog Files feature provided a straightforward method for uploading Verilog files, which are critical for circuit design in OpenFASOC flow. This feature simplified file management within the OpenFASOC environment, ensuring that all necessary files were readily accessible for further processing.

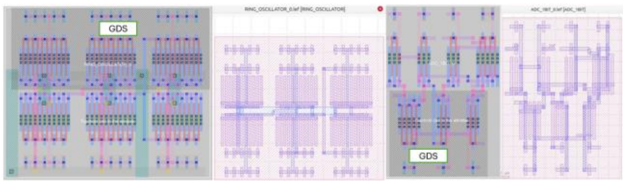
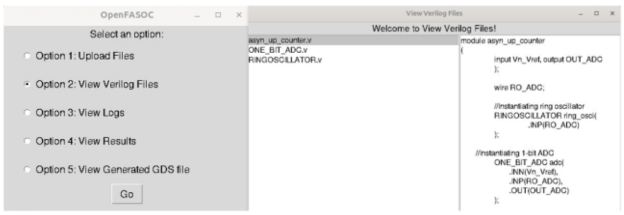


Figure 23: Upload GDS and LEF Files of both Ring Oscillator and ADC

Selecting the “View Verilog Files” option opened a new window where users could browse and view the contents of Verilog files, as illustrated in Figure 24. This window featured a listbox that listed all Verilog files located in the specified directory. Upon selecting a file from the list, its contents were displayed in a text widget within the same window. This functionality enabled users to efficiently inspect and review their Verilog code, facilitating debugging and validation processes.

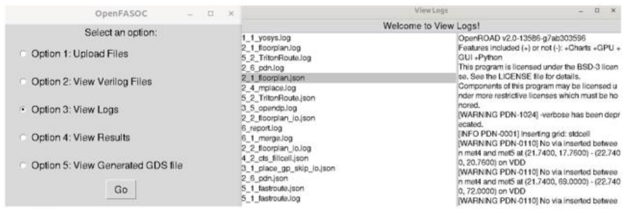


```
asyn_up_counter.v
ONE_BIT_ADC.v
RINGOSCILLATOR.v
```

```
module asyn_up_counter
(
        input Vn_Vref, output OUT_ADC
        );

        wire RO_ADC;

        //Instantiating ring oscillator
        RINGOSCILLATOR ring_osci(
                .INP(RO_ADC)
        );

    //Instantiating 1-bit ADC
        ONE_BIT_ADC adc(
            .INN(Vn_Vref),
            .INP(RO_ADC),
            .OUT(OUT_ADC)
        );
```

Figure 24: View Verilog Files Window

The View Logs option opened a window where users could view the logs generated during the design and simulation processes. By selecting a log file from the list box, users could read its content in the text widget, as shown in Figure 25. This feature was crucial for diagnosing issues and ensuring that the design process proceeds smoothly.

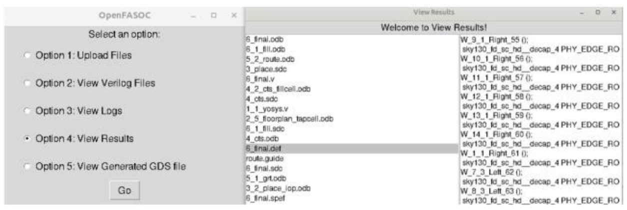


```
1_1_yosys.log
2_1_floorplan.log
5_2_TritonRoute.log
2_6_pdn.log
2_1_floorplan.json
2_4_mplace.log
5_2_TritonRoute.json
3_5_opendp.log
2_2_floorplan_io.json
6_report.log
6_1_merge.log
2_2_floorplan_io.log
4_2_cts_fillcell.json
3_1_place_gp_skip_io.json
2_6_pdn.json
5_1_fastroute.json
5_1_fastroute.log
```

```
OpenROAD v2.0-13586-g7ab303596
Features included (+) or not (-): +Charts +GPU +
GUI +Python
This program is licensed under the BSD-3 licen
se. See the LICENSE file for details.
Components of this program may be licensed u
nder more restrictive licenses which must be ho
nored.
[WARNING PDN-1024] -verbose has been depr
ecated.
[INFO PDN-0001] Inserting grid: stdcell
[WARNING PDN-0110] No via inserted betwee
n met4 and met5 at (21.7400, 17.7600) - (22.740
0, 20.7600) on VDD
[WARNING PDN-0110] No via inserted betwee
n met4 and met5 at (21.7400, 69.0000) - (22.740
0, 72.0000) on VDD
[WARNING PDN-0110] No via inserted betwee
```

Figure 25: View logs Window

When users selected the “View Results” option, as depicted in Figure 26, they accessed comprehensive results generated by OpenFASOC, covering the synthesis to routing stages. Similar to the “View Logs” feature, this window enabled users to browse, select, and examine result files effortlessly. It offered valuable insights into the outcomes of design and simulation processes, aiding in the validation of design effectiveness and correctness. This streamlined capability ensured a smooth and efficient user experience by facilitating a thorough review of all relevant files.



```
6_final.odb
6_1_fill.odb
5_2_route.odb
3_place.sdc
6_final.v
4_2_cts_fillcell.odb
4_cts.sdc
1_1_yosys.v
2_5_floorplan_tapcell.odb
6_1_fill.sdc
4_cts.odb
6_final.def
route.guide
6_final.sdc
5_1_grt.odb
3_2_place_iop.odb
6_final.spef
```

```
W_9_1_Right_55 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_10_1_Right_56 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_11_1_Right_57 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_12_1_Right_58 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_13_1_Right_59 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_14_1_Right_60 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_1_1_Right_61 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_7_3_Left_62 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
W_8_3_Left_63 ();
 sky130_fd_sc_hd__decap_4 PHY_EDGE_RO
```

Figure 26: View Results Window

The Option 5: View Generated GDS File feature enabled users to view the final GDS layout using KLayout, as illustrated in Figure 27. Upon selection, the application automatically opened the specified GDS file in KLayout, providing a detailed visualization of the circuit layout. This integration with KLayout enhanced the user's ability to inspect and analyze the final design of a 4-bit asynchronous up counter, ensuring it met the required specifications.

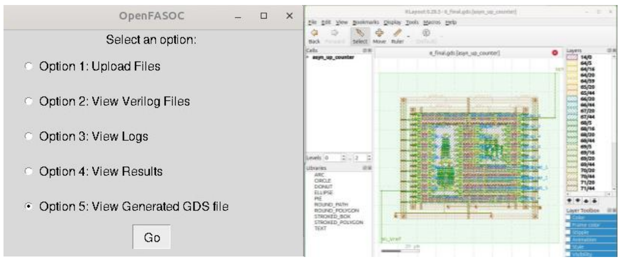


Figure 27: View Generate GDS File

V. CONCLUSION

The work has successfully integrated different intelligent modules such as AutoCkt, Xschem, ALIGN, and OpenFASOC to achieve automated analog and mixed-signal design from schematic to GDSII layout. AutoCkt optimizes circuits such as the two-stage operational amplifier with a 93.8% success rate using reinforcement learning. Xschem facilitated schematic capture, ALIGN automated layout generation, and OpenFASOC efficiently integrated designs. Successful simulations of the 1-bit ADC, ring oscillator, and 4-bit asynchronous up counter confirmed stable performance and functionality. ALIGN ensured precise placement and routing, while OpenFASOC supported seamless integration and manufacturing readiness, despite challenges in integration and verification. However, limitations were identified, including Xschem's lack of advanced simulation capabilities, AutoCkt's resource-intensive optimization, ALIGN's difficulty with custom layouts, and OpenFASOC's limited technology support and challenges with cloud deployment. Future improvements are proposed to enhance Xschem's simulation capabilities, optimize AutoCkt's algorithms, improve ALIGN's flexibility, expand OpenFASOC's technology support, and develop a unified interface for smoother design transitions. These enhancements aim to streamline the workflow for better efficiency and user-friendliness.

ACKNOWLEDGMENT

The authors acknowledge the technical and financial support by the Ministry of Higher Education, Malaysia, under research grant no. FRGS/1/2024/ICT02/UTEM/02/3, and Universiti Teknikal Malaysia Melaka (UTeM).

CONFLICT OF INTEREST

Authors declare that there is no conflict of interests regarding the publication of the paper.

AUTHOR CONTRIBUTION

The authors confirm contribution to the paper as follows: study conception and design: Yan Chiew Wong, Jiveya Sri Sockalingam; data collection: Jiveya Sri Sockalingam; analysis and interpretation of findings: Jiveya Sri Sockalingam, Ser Lee Loh, Yan Chiew Wong; draft manuscript preparation: Jiveya Sri Sockalingam, Ranjit Singh Sarban Singh, T. Joseph Sahaya Anand. All authors had reviewed the findings and approved the final manuscript.

REFERENCES

[1] M. T. Nguyen et al., "Direct delta-sigma receiver: analysis modelization and simulation" in Proc. of 2013 IEEE International Symposium on Circuits and Systems (ISCAS), 2013, pp. 1035-1038.
[2] R. Dutta, A. James, S. Raju, Y. -J. Jeon, C. S. Foo and K. T. C. Chai, "Automated deep learning platform for accelerated analog circuit design," in Proc. of 2022 IEEE 35th International System-on-Chip Conference (SOCC), 2022, pp. 1-5.
[3] T. Ajayi et al., "An open-source framework for autonomous SoC design with analog block generation," in Proc. of 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), 2020, pp. 141-146.
[4] O. Aiello, P. Crovetti and M. Alioto, "Fully synthesizable low-area analogue-to-digital converters with minimal design effort based on the dyadic digital pulse modulation," IEEE Access, vol. 8, pp. 70890-70899, 2020.
[5] B. Khailany et al., "Accelerating chip design with machine learning," IEEE Micro, vol. 40, no. 6, pp. 23-32, 2020.
[6] K. Kunal et al., "ALIGN: Open-source analog layout automation from the ground up," in Proc. of the 56th Annual Design Automation Conference 2019 (DAC '19), 2019, pp. 1–4.
[7] A. Hammoud, V. Shankar, R. Mains, T. Ansell, J. Matres and M. Saligane, "OpenFASOC: An open platform towards analog and mixed-signal automation and acceleration of chip design," in Proc. of 2023 International Symposium on Devices, Circuits and Systems (ISDCS), 2023, pp. 1-4.
[8] S. -H. Lui, H. -K. Kwan, and N. Wong, "Analog circuit design by nonconvex polynomial optimization: two design examples," Int. J. circuit Theory Appl., vol. 38, no. 1, pp. 25-43, 2010.
[9] Y. Uhlmann, M. Brunner, L. Bramlage, J. Scheible, and C. Curio, "Procedural- and reinforcement-learning-based automation methods for analog integrated circuit sizing in the electrical design space," Electronics, vol. 12, no. 2, 2023.
[10] A. Gusmão, P. Alves, N. Horta, N. Lourenço, and R. Martins, "Differentiable constraints' encoding for gradient-based analog integrated circuit placement optimization," Electronics, vol. 12, no. 1, pp. 110-110, 2022.
[11] W. Cao, M. Benosman, X. Zhang, and R. Ma, "Domain knowledge-infused deep learning for automated analog/radio-frequency circuit parameter optimization, in Proc. of the 59th ACM/IEEE Design Automation Conference (DAC '22), 2022, pp. 1015–1020.
[12] Y. Wei, J. Liu, D. Sun, G. Su, and J. Wang, "From netlist to manufacturable layout: an auto-layout algorithm optimized for radio frequency integrated circuits," Symmetry, vol. 15, no. 6, pp. 1272–1272, 2023.
[13] D. Meng and Y. -L. Zheng, "Circuit partitioning for PCB netlist based on net attributes," in Proc. of 2022 International Conference on Machine Learning and Cybernetics (ICMLC), 2022, pp. 31-36.
[14] J. Lienig and J. Scheible, "Steps in physical design: from netlist generation to layout post processing," Springer eBooks, pp. 165–211, 2020.
[15] F. Castejón, E. J. Carmona, "Automatic design of analog electronic circuits using grammatical evolution", Applied Soft Computing, Volume 62,2018, Pages 1003-1018,
[16] S. Yaldiz, "Analog layout automation on advanced process technologies," in Proc. of the 2023 International Symposium on Physical Design (ISPD '23), 2023, pp. 103.
[17] Z. Zhao and L. Zhang, "Analog integrated circuit topology synthesis with deep reinforcement learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 12, pp. 5138-5151, 2022.
[18] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in Proc. of 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2020, pp. 490-495.
[19] A. F. Budak, P. Bhansali, B. Liu, N. Sun, D. Z. Pan, and C. V. Kashyap, "DNN-Opt: An RL inspired optimization for analog circuit sizing using deep neural networks," in Proc. of 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 1219–1224.
[20] J. Lienig and J. Scheible, "Steps in physical design: from netlist generation to layout post processing," Springer eBooks, pp. 165–211, 2020.

[21] K. Kunal, J. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. S. Sapatnekar, “A general approach for identifying hierarchical symmetry constraints for analog circuit layout,” arXiv (Cornell University), 2020.

[22] T. Ajayi et al., “An open-source framework for autonomous soc design with analog block generation,” in Proc. of 2020 IFIP/IEEE 28th International Conference on Very Large-Scale Integration (VLSI-SOC), 2020, pp. 141-146.

[23] T. Dhar et al., “ALIGN: A system for automating analog layout,” IEEE Design & Test, vol. 38, no. 2, pp. 8-18, 2021.

[24] R. Dreslinski et al., “Fully autonomous soc synthesis using customizable cell-based synthesizable analog circuits,” GOMACTech Conference, Tech. rep., University of Michigan Ann Arbor United States, pp. 1-21, 2019.