

Automated Topology Synthesis for Analog Integrated Circuits

by

©Zhenxin Zhao

A dissertation submitted to the School of Graduate Studies
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

**Faculty of Engineering & Applied Science
Memorial University of Newfoundland**

Supervisory Committee

Dr. Lihong Zhang (Supervisor)

Dr. Tariq Iqbal

Dr. Antonina Kolokolova

May 2022

St. John's, Newfoundland

Abstract

Currently, except for circuit topology synthesis, all the other phases in the analog integrated circuit design procedure are equipped with electronic design automation (EDA) commercial tools to greatly facilitate the human laborious work and significantly improve the design productivity, even though they are still not as mature as digital EDA counterparts. This dissertation focuses on developing a circuit topology synthesis EDA tool for analog integrated circuits. In order to make the developed EDA tool commercializable, there are many challenges that have to be solved, including trustworthy solutions, innovative solutions, wide applicability, sound generalization capability, and affordable computation effort. This thesis proposes a graph-based generation method to automatically synthesize analog integrated circuits, which has partially solved some challenges. But one serious problem of this method is its unaffordable computation effort due to the time-consuming sizing process for a huge number of generated circuit structures. To address this problem, we propose a novel performance modeling method that can boost the sizing efficiency by more than 30 times with ignorable model building overhead, which is especially suitable for the circuit synthesis work that involves generating various circuit structures. With the assistance of the emerging machine learning advancement, EDA tools can be more efficient and effective. We have employed the deep reinforcement learning technique in this dissertation to synthesize analog integrated circuit structures. Its technical merits make it be able to address those pending challenges much better than the graph-based generation method. But it still suffers from a shortcoming, that is, the learning process has to be performed from scratch once the technology or design specification changes. In order to overcome this shortcoming, the transfer learning technique is applied to transfer the learned knowledge from a learning process to another in order

to largely save the learning effort. The experimental results exhibit strong efficacy and great applicability of our proposed methods.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Lihong Zhang for his continuous support in my PhD study, research, motivation, enthusiasm, immense knowledge and his assistance in writing papers and dissertation. I also intend to thank the other members of my supervisor committee, Dr. Iqbal and Dr. Kolokolova, for their guidance and suggestions.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, in part by Canada Foundation for Innovation, in part by the Research and Development Corporation of Newfoundland and Labrador through its Industrial Research and Innovation Fund and ArcticTECH R&D Award, and in part by the Memorial University of Newfoundland.

To my adorable parents

Table of Contents

Table of Contents

Chapter 1 Introduction	1
Chapter 2 Automated Analog Circuit Topology Synthesis, Challenges and Solutions.....	5
2.1. Challenges in Automated Analog Circuit Topology Synthesis	5
2.1.1. Wide Applicability.....	6
2.1.2. Strong Generalization Capability.....	7
2.1.3. Affordable Computation Effort.....	7
2.2. State-of-the-Art Analog Circuit Topology Synthesis Methods.....	8
2.2.1. Topology Selection Methods	9
2.2.2. Topology Generation Methods	10
2.2.3. Topology Refinement Methods	17
2.3. Summary	21
Chapter 3 Graph-Grammar-Based Topology Synthesis for Analog Integrated Circuits.....	22
3.1. Introduction	22
3.2. Circuit Topology Generation	24
3.2.1. Graph-Grammar-Based Tree Structure Generation (GTSG).....	24
3.2.2. Tree Structure Level Isomorphism	33
3.2.3. Circuit Formation.....	34
3.2.4. Circuit Topology Level Isomorphism.....	35
3.3. Fast Evaluation of Un-sized Circuits	38
3.3.1. Preliminaries	38
3.3.2. Low-order Symbolic Transfer Function Generator	40
3.3.3. Flow of Tests.....	42
3.4. Experimental Results.....	44
3.4.1. Predefined Building Block Library.....	46
3.4.2. Results of Circuit Topology Generation.....	46
3.4.3. Evaluation of LTFG.....	49
3.4.4. Results of Un-Sized Circuit Fast Evaluation	50

3.4.5.	Innovative Circuit Topology Synthesis	52
3.4.6.	Comparison with State-of-the-Art Tool.....	53
3.5.	Summary	55
Chapter 4	Efficient Performance Modeling for Automated Analog Circuit Synthesis	57
4.1.	Introduction	57
4.2.	Accurate Transistor Modeling.....	59
4.2.1.	Data Sampling and Preprocessing	61
4.2.2.	NN Design	62
4.2.3.	Model Segmentation	63
4.3.	DC Operating Point Computation.....	64
4.3.1.	Circuit Preprocessing.....	65
4.3.2.	Computation Methods.....	69
4.3.3.	Handling of Unsolvable Cases.....	76
4.4.	Circuit Performance Evaluation.....	78
4.5.	Experimental Results.....	80
4.5.1.	Evaluation of Transistor Modeling.....	81
4.5.2.	Comparison of DC Operating Point Computation Methods.....	84
4.5.3.	Analysis of Performance Modeling	86
4.5.4.	Comparison with the State-of-the-Art Methods	89
4.6.	Summary	91
Chapter 5	Deep-Reinforcement-Learning-Based Topology Synthesis for Analog Integrated Circuits.....	92
5.1.	Introduction	92
5.2.	The Proposed Framework	94
5.2.1.	Deep Reinforcement Learning.....	94
5.2.2.	DRL Framework for Circuit Topology Synthesis	97
5.3.	Specialized RL Environment	99
5.3.1.	Building Block (BB).....	99
5.3.2.	State.....	100
5.3.3.	Action.....	101
5.3.4.	Episode and Reward	104

5.4.	Circuit Topology Formation & Evaluation	106
5.4.1.	Hash Table	107
5.4.2.	State Decoding	110
5.4.3.	Unsize-Circuit Fast Evaluation	111
5.4.4.	Simulation-in-Loop Sizing.....	111
5.5.	Experimental Results.....	113
5.5.1.	Experimental Parameter Settings.....	114
5.5.2.	Analysis of the Learning Process.....	115
5.5.3.	Analysis of Synthesis Efficiency	116
5.5.4.	Analysis of Circuit Topology Synthesis	118
5.5.5.	Comparison with the State-of-the-Art Methods	122
5.6.	Summary	125
Chapter 6	Transfer Learning for Automated Analog Integrated Circuit Synthesis	127
6.1.	Introduction	127
6.2.	Training of DRL-CTSF	130
6.2.1.	Hierarchical Building Block Library	131
6.2.2.	Starting Sub-circuits Generation.....	133
6.3.	Training of DRL-CSS	138
6.3.1.	Preliminary DRL Framework	138
6.3.2.	Knowledge Transfer Among Design Specifications and Technology Process Nodes 139	
6.4.	Summary	140
Chapter 7	Conclusion and Future Work	141
References	147
Appendix A:	Published/Submitted Papers.....	156

List of Tables

Table 3.1. Allowed decompositions for different types of block nodes	27
Table 3.2. PBB library	45
Table 3.3. Results of Our Proposed Circuit Topology Synthesis Framework.....	48
Table 3.4. Example Results of LTFG	49
Table 3.5. Result of fast sizing and performance simulation test of the folded-Cascode OpAmp and a creative circuit topology	51
Table 3.6. Comparison between FEATS and GCTG.....	55
Table 4.1. Applied Training Data Sampling Scheme for the CMOS 65nm Technology	81
Table 4.2. Testing Results of the Transistor Models (Percentages of the Testing Data that Achieved the Given Accuracy Levels)	82
Table 4.3. Comparison of Various Transistor Modeling Methods (with Specified Size Ranges) 84	
Table 4.4. Results of DC Operating Point Computation and Symbolic Analysis of Two Properly Sized Circuits	85
Table 4.5. Optimization Results of Two OpAmp Circuits	87
Table 4.6. Building Blocks Used to Construct Circuits in Synthesis	88
Table 4.7. Results of Circuit Topology Synthesis	89
Table 4.8. Comparison of Different Performance Modeling Methods for the Two-Stage OpAmp	90
Table 5.1. Predefined Building Block Library (PBBL).....	114
Table 5.2. Evaluation Details of the Exemplary Six Processes	117
Table 5.3. Testing Results of the Effectiveness of Fast Evaluation Filter	117
Table 5.4. PBBL Extension	118
Table 5.5. Comparison with Other Circuit Topology Synthesis Methods.....	123
Table 5.6. Comparison among FEATS, GCTG, and DRL by using PBBL	124
Table 6.1. An example of hierarchical building block library	133

List of Figures

Fig. 3.1. Our proposed analog circuit synthesis framework.	23
Fig. 3.2. Example of some tree nodes and their possible contents.	25
Fig. 3.3. Normal node examples	26
Fig. 3.4. Decomposition operation examples.....	28
Fig. 3.5. Voltage-division decomposition.....	30
Fig. 3.6. An example of tree structure level isomorphism.....	34
Fig. 3.7. An example of circuit topology level isomorphism.	36
Fig. 3.8. GPDD data structure examples.....	39
Fig. 3.9. An example of synthesis of a three-stage OpAmp	47
Fig. 3.10. A synthesis example of a creative circuit	52
Fig. 3.11. An example of synthesis of an analog circuit.....	54
Fig. 4.1. Small-signal MOSFET model.	60
Fig. 4.2. Our NN structure.	63
Fig. 4.3. (a) The folded-Cascode OpAmp (FCO). (b) Its undirected bipartite graph (UBG) representation.	66
Fig. 4.4. The process of solving the variables in the folded-Cascode OpAmp.....	72
Fig. 4.5. Another two operational amplifiers as the test circuits.	75
Fig. 4.6. Parameter mappings in our proposed performance evaluation flow.	79
Fig. 5.1. Policy gradient neural network.....	96
Fig. 5.2. The proposed DRL framework for circuit topology synthesis.	97
Fig. 5.3. Examples of the six types of BBs.....	99
Fig. 5.4. An example state and its possible circuit topologies represented.	100
Fig. 5.5. Examples of the matched actions.	102
Fig. 5.6. (a) Two example episodes. (b) Their corresponding circuit topology construction processes.	105
Fig. 5.7. The proposed framework with detailed circuit topology evaluation process.	106
Fig. 5.8. Examples of unique state representations.....	108
Fig. 5.9. The mean reward of a batch during the learning process.	116
Fig. 5.10. Example of synthesized circuits.	120
Fig. 5.11. A novel circuit synthesized.....	122
Fig. 6.1. The interaction between users and the DRL-CSS.	128
Fig. 6.2. Training and using of a DRL framework.	130
Fig. 6.3. Training DRL-CTSF with hierarchical building block library.....	132
Fig. 6.4. An example of the perturbation operation.	135
Fig. 6.5. Preliminary DRL-CTSF.	138

List of Algorithms

Algorithm 3.1: Graph-Grammar-Based Tree Structure Generation	33
Algorithm 4.1: Performance Model Construction	69
Algorithm 4.2: Hybrid-Based Performance Evaluation.....	77
Algorithm 5.1: DRL-Based Circuit Topology Synthesis.....	112
Algorithm 6.1: Min-Occurrence-Driven Training Scheme	137

List of Abbreviations

Artificial Neural Network	ANN
Building Block	BB
Breadth-First Search	BFS
Complementary Metal-Oxide-Semiconductor	CMOS
Computer-Aided Design	CAD
Depth-first Search	DFS
Deep Reinforcement Learning	DRL
Electronic Design Automation	EDA
Genetic Programming	GP
Integrated Circuits	IC
Mixed Integer Nonlinear Programming	MINLP
Low-Noise Amplifier	LNA
Multi-Layer Perceptron	MLP
Metal–Oxide–Semiconductor Field-Effect Transistor	MOSFET
Mean Square Error	MSE
Operational Amplifier	OpAmp
Radio Frequency	RF
Root Mean Square Error	RMSE
Simulated Annealing	SA
Support Vector Machine	SVM
Undirected Bipartite Graph	UBG
Voltage Controlled Current Sources	VCCS

Chapter 1 Introduction

Due to continuous scaling of integrated circuit (IC) technologies and growing demands for high speed, small area, low power, and low cost, powerful electronic design automation (EDA) commercial tools are in great need in order to improve the design productivity for reduced time-to-market, facilitate the human laborious work, and enhance the design accuracy.

In the digital domain, the specifications of the digital circuits can be easily defined by using either Boolean functions or behavioral description in a hardware description language (e.g., VHDL or Verilog). Digital synthesis EDA tools can convert the behavioral level description to structural and subsequently to gate level descriptions. The validity of the design is checked through test platforms to ensure the satisfaction of the functional requirements specified by the system specifications as well as timing requirements. Then the layout tools map gate-level netlist to layout by performing floorplan, placement and routing, according to the specified technology. Finally, various verifications, including static timing analysis, functional equivalence checking, and physical design checking, are conducted along with certain iterations in the design flow as exhibited in Fig. 1.1(a) [1].

Unlike the digital computer-aided design (CAD) tools, which have long found prosperous in the semiconductor industry, the efforts are still largely needed to devise their analog counterparts to reach the same level of usability and reliability. The main reasons for this include the knowledge-intensive and heuristic nature of the analog design, high complexity and nonidealities involved within the design procedure, and conflicting requirements for satisfying the design specifications. This gets even more challenging since all of these factors keep changing from

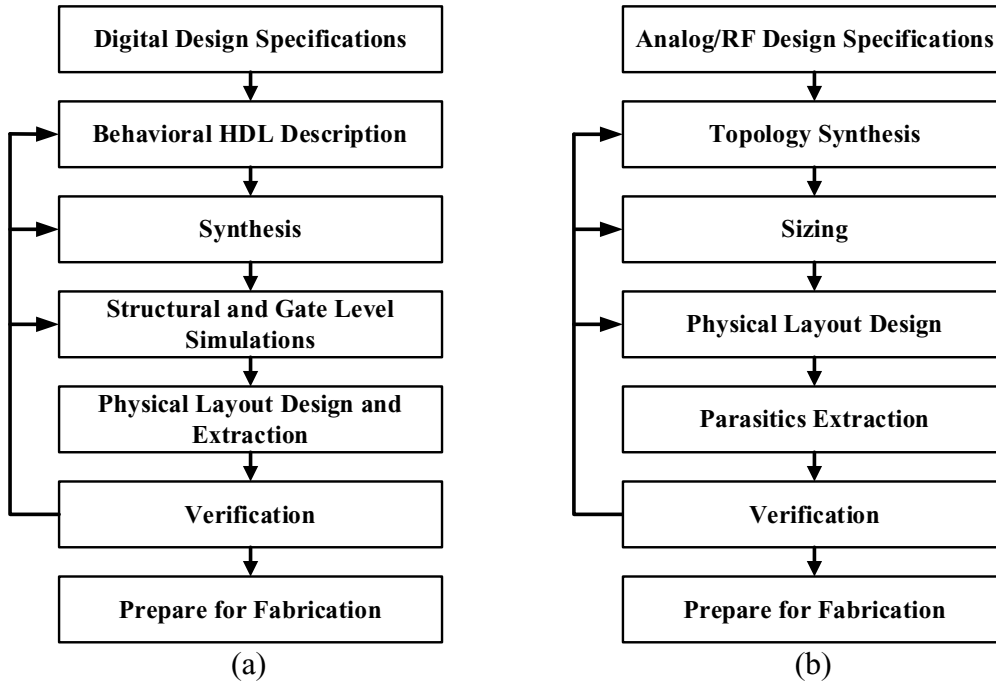


Fig. 1.1. Circuit design flows. (a) Digital design flow. (b) Analog/RF design flow.

process to process. As a matter of fact, these challenges have motivated us to focus on the research of analog EDA tools.

A widely accepted analog circuit design flow is illustrated in Fig. 1.1(b) [1]. As shown in this figure, after inputting the circuit specifications, the first stage is called “*Topology Synthesis*”. The output of this stage is a netlist that contains the connectivity information, geometry information, and electrical biases of the devices in the synthesized circuit. Device geometry, specifically in the CMOS technology, mainly refers to transistor width (W) and length (L) among others, and resistor/capacitor/inductor nominal values. The electrical bias may include circuit biasing voltage or current information. At this stage, the obtained device geometry and electrical biases can only guarantee that the synthesized circuit topologies meet the basic performance specification. Usually, designers have some special constraints and performance requirements, which are addressed in the “*Sizing*” stage. The objective of this stage is to determine the device geometries and electrical

biases to let the circuit fully satisfy the target constraints and requirements. Both the topology synthesis and sizing operations are normally categorized into the circuit-level design, while the other stages (i.e., “*Physical Layout Design*”, “*Parasitic Extraction*”, and “*Verification*”) fall in the layout-level design [1].

Physical layout design, also called “*Layout Synthesis*”, is typically comprised of module generation, placement, and routing. All the devices or functional building blocks in an analog circuit are first converted to their corresponding module layouts. This process is called module generation, which is to produce primitive units for the consequent operations [2]. Digital circuits normally use standard cell libraries for their modular units, while analog/RF circuits need more sophisticated modules to be customized for their special functionality in the context of various analog constraints. The placement process is to locate all the primitive modules to proper spots under demanded constraints. The major placement constraints are layout area and analog constraints, including symmetry, common centroid, proximity, regularity, etc. [3]. Some works consider wirelength as another constraint to decrease the interconnect parasitic impact on circuit performance. The analog routing process is to connect electrical terminals of the primitive modules by using optimal paths to meet the routing constraints for reaching the best performance. The second last stage of the analog circuit design flow is parasitic extraction that derives the parasitics of the layout for both interconnects and devices. Since analog circuits suffer from parasitic effects, it is essential to verify the functionality of the final layout after parasitic extraction. Thus, this verification process checks whether the circuit specifications are satisfied in the presence of parasitics. If so, the layout is ready for fabrication. Otherwise, certain iterations in the design flow are needed to address the drawbacks of the design.

As one can see, the “*Circuit Synthesis*” stage is quite important as it may significantly reduce the number of iterations needed in the design flow, thus greatly reducing the design cycle for sound time-to-market. However, due to the ongoing lack of a well-organized and efficient synthesis tool, the circuit topology synthesis is often largely simplified by manually selecting a familiar circuit topology from a predefined library, which needs to consider the tradeoffs among power, performance, area, yield, etc. However, the following sizing process can only produce a result as good as the selected topology allows. In order to meet the target specifications, other topology choices in the library may also be tried, which always leads to an expensive iteration. If necessary, a novel circuit topology has to be manually designed and verified. Therefore, this topology selection requires strong involvement from analog designers in terms of design knowledge and experience. But learning analog circuit design is a process that normally takes years to get started and decades to become a master. Thus, it is highly desirable for EDA tools to help designers in synthesizing circuit topology.

The rest of the thesis is organized as follows. Chapter 2 reviews the challenges that existed in the problem of analog circuit topology synthesis and the previous related works on this topic. Chapter 3 illustrates the graph-grammar-based topology synthesis method for analog integrated circuits. In Chapter 4, a performance modeling method that can boost the sizing efficiency by more than 30 times with ignorable model building overhead, which is especially suitable for the circuit synthesis work that involves generating various circuit structures, is explained. Chapter 5 describes a machine-learning-based method that utilizes the deep reinforcement learning technique to automatically synthesize analog integrated circuit topologies. Chapter 6 concludes this dissertation and discusses the future work.

Chapter 2 Automated Analog Circuit Topology Synthesis, Challenges and Solutions

Success in the manual analog circuit design, which is heavily dependent on the experience and intuition of the designers, has become increasingly difficult due to the continuous scaling of integrated circuit (IC) technologies and growing demands for high-performance low-power solutions. Thus, efficient automated analog circuit design is in great need in order to improve design productivity, facilitate human laborious work, and enhance design accuracy. Currently, almost all the stages in the analog design flow are equipped with electronic design automation (EDA) tools, even though they are still not as mature as digital EDA counterparts [4]. However, due to the high design complexity involved, huge design space searched, substantial design expertise required, and conflicting constraints traded off, there are still no widely accepted solutions to automated analog circuit topology synthesis even at the research level.

2.1. Challenges in Automated Analog Circuit Topology Synthesis

In order to make the analog circuit topology synthesis EDA tool commercializable, there are some challenges that have to be solved, including wide applicability, strong generalization ability, and affordable computation effort. Currently, all the existing research works are not able to perfectly address the abovementioned challenges.

2.1.1. Wide Applicability

Analog integrated circuits (ICs) mainly refer to integrated circuits that use analog circuits composed of capacitors, resistors, and transistors to process analog signals. The basic circuits of analog integrated circuits include current sources, single-stage amplifiers, filters, feedback circuits, current mirror circuits, etc. [5] The higher-level basic circuits composed of them are operational amplifiers and comparators, while the higher-level circuits include switched-capacitor circuits, phase-locked loop, ADC/DAC, etc. [6] According to the response relationship between the output and input signals, analog integrated circuits can be divided into two categories: linear integrated circuits and nonlinear integrated circuits [7]. In the linear ICs, the response between the output and input signals usually has a linear relationship. Thus, the shape of the output signal is similar to that of the input signal, but it is amplified or attenuated by a fixed coefficient. The response of the output signal in a nonlinear integrated circuit to the input signal has a nonlinear relationship, such as a square relationship, a logarithmic relationship, etc. So it is called a nonlinear circuit. Common nonlinear circuits include oscillators, timers, and phase-locked loop circuits.

As explained above, the range of analog integrated circuits is so wide, including different levels of implementation in terms of basic circuit components. Although hierarchical approaches are widely applied to address the problem of synthesizing circuits with multiple hierarchy levels, solving such a complex problem is still quite challenging. Even for the analog ICs at the same hierarchy level, such as operational amplifier and comparator, they usually have different structural characteristics and performance specifications. For instance, operational amplifiers care about gain while comparators concern about propagation delay.

Therefore, it is quite challenging to develop a circuit synthesizer that is able to synthesize such a wide range of analog ICs. Due to this reason, most of the existing works on this topic have only

focused on synthesizing either passive circuits or operational amplifiers. There is still a long way to go for the commercialization of circuit synthesis tools for analog ICs.

2.1.2. Strong Generalization Capability

As mentioned before, continuous scaling of integrated circuit technologies and growing demands for high-performance low-power solutions, which have led to increased difficulty in the design of analog ICs, are certainly also applied to circuit topology synthesis. In order to overcome these challenges, the developed circuit topology synthesizer has to have strong generalization capability, that is, it needs to be able to generalize to a new technology process and design specification from current ones with as little cost as possible. Here the design specification is composed of input-output specification and performance specification. However, due to their inherent natures, many existing works, such as rule-based methods, cannot guarantee that they can freely generalize to any technology processes or design specifications. Even though some methodologies, such as graph-based methods, has relatively better generalization capability, the cost for generalization is high. In addition, the developed synthesizer should have the capacity to freely generalize to synthesize large-size and innovative circuit topologies, which is one of the main shortcomings of manual analog circuit design due to the strong preference for known circuit structures.

2.1.3. Affordable Computation Effort

The most time-consuming part of circuit topology synthesis is to evaluate the feasibility of the produced circuit topologies within the synthesis process. Because the performance of a circuit not only depends on its topology (i.e., circuit structure) but also relies on its device sizes, a quite time-consuming sizing process has to perform on each topology to be evaluated to check whether it can

meet the performance specification or not. This feasibility information would be used to drive the circuit topology synthesis process in the right direction within the design search space. Furthermore, there is a tradeoff among applicability range, generalization capability, and computation complexity. In general, the wider the applicability range and the stronger the generalization capability are, the more computation efforts are needed. For instance, compared with synthesizing small-size circuits, the synthesizers that are able to produce large-size circuits usually require larger design search space at the cost of more computation effort.

There are two effective manners to reduce the computation effort. One way is to decrease the number of circuit topologies to be sized in the synthesis process. Another way is to speed up the sizing process. For the former scheme, except for the circuit topology synthesis algorithm itself, fast evaluation [8] and performance boundary exploration [9] are also widely used ways to reduce the number of circuit topologies to be sized. For the latter one, performance modeling [10] is proposed to substitute the computation-intensive and time-consuming SPICE simulator to reduce the sizing time, or some heuristic methods [11] are employed to shrink the sizing iterations. However, due to the high complexity of the problem itself, it is still challenging to find a solution that is able to efficiently synthesize circuit topologies.

2.2. State-of-the-Art Analog Circuit Topology Synthesis Methods

From the conceptual view, topology synthesis methods can be classified into three categories: 1) topology selection; 2) topology generation; and 3) topology refinement. In the following subsections, the methods in each category will be reviewed and discussed in detail.

2.2.1. Topology Selection Methods

The pioneer research work in the field of topology synthesis started to utilize the topology selection methods. In early attempts, desired topologies might be selected by users, and the behavioral models, which are in-depth analytic equations, can also be extracted by human designers. A library of well-established circuit topologies along with the corresponding behavioral models is the common feature of these methods [12] [13]. In spite of certain differences in the following steps to complete the design, they are all the same in the topology synthesis part, where the human designers directly make the selection with no automation involved to decide on the optimal topology. Thus, for junior designers, it would still be difficult to make a decision about the right topology or they would have to repeat the trial and error process for a long time.

Realizing the drawbacks above, the later methods have applied automation techniques mostly based on heuristic and design knowledge [14]. OASYS [15] manually creates templates of design styles in advance based on detailed design knowledge in order to facilitate its automated topology selection strategy. However, even though the selection process is done automatically, the time-consuming manual template design is unaffordable and hard to be generalized. AMGIE [16] solves the drawback of OASYS by automatically establishing a database at runtime through a large number of SPICE simulations, and then its proposed elimination strategy is applied to choose topologies based on the database. Unfortunately, this method only supports a few types of topologies, and building the database takes substantial computational effort. Integer programming is also an alternative method in [17] where a mixed integer nonlinear programming (MINLP) is deployed to optimize a set of analytic equations, derived from a super-circuit and a set of associated integers, which can result in optimal circuit topologies.

In general, those automated analog circuit synthesis methods in the category of topology selection have to address their time-consuming library setup challenge and poor capacity of generalization. Due to its intractable shortcomings, the stream of the topology selection synthesis has phased out over the years.

2.2.2. Topology Generation Methods

Topology generation methods, as the name indicates, aim to generate a topology from scratch. Instead of selecting a definitive structure from a library, the algorithm creates new circuit topologies by connecting together the basic components and sub-circuits, often provided from a library of basic building blocks. Different strategies have been devised over the past two decades to tackle the topology generation problem, which can mainly be classified into three categories: 1) evolutionary-algorithm (EA)-based; 2) graph-based; and 3) hybrid-based.

2.2.2.1. EA-based Topology Generation

The EA-based methods have become popular due to their inherent characteristics of automation, which means more independent of human effort. They are typically realized through population-based meta-heuristic optimization algorithms, such as genetic algorithm and evolutionary algorithm. Evolutionary operators, such as mutation and crossover, are utilized to incorporate more appropriate circuit components and building blocks at each step during the evolution. Early open-ended work in this field, which failed to combine any or many of circuit rules in the evolution flow, were less helpful, due to an unreasonable amount of generated circuits mostly meaningless from the vantage point of an analog designer. Sizing is often an integral part of the EA-based methods. It means that the sizing is done concurrently with the topology generation and its results are utilized as an evaluation factor to decide on the upcoming changes to

the topology. In EA-based circuit topology synthesis methods, the circuit topologies are encoded in various forms, such as graph, tree, and string.

Passive analog filters were synthesized in [18] by using a graph representation and clone selection algorithm, which is a type of EA. However, no active device synthesis can be considered in this work. Reference [19] followed a similar approach based on EA to generate multiple passive filters. Based on the fitness value, the best circuit was selected from each run and then three best fault-tolerant circuits were chosen. The outputs were combined by using a weighted summing function to help generate the robust ones. A similar EA-based method is utilized in [20] to synthesize a computational analog circuit for implementing cube root function with BJT transistors and resistors. The circuit is represented in a graph form and the information of the node connection and resistor parametric values is stored in the chromosomes, which are altered via EA operators in each generation. However, the method is quite open-ended with nothing mentioned about limiting of construction rules. It is believed that this method still needs some advanced techniques to handle the challenges of synthesizing more complex circuits. Similarly, Cohen *et al.* [21] proposed an open-ended GA-based method to simply combine circuit sub-units that were obtained by applying graph theory to a user-specified number of circuit components. For each circuit unit, two data including component position and its orientation need to be determined. Afterwards, GA is utilized to determine the optimal placement and routing among circuit units. Compared to the other works, this study tends to be based on some very basic rules of schematic-level analog circuit design. More advanced rules need to be appended to resolve the bottlenecks in the current analog EDA.

Sripramong and Toumazou [22] proposed a tree-based bottom-up genetic programming approach to generate topology structures. A design optimization method called current-flow analysis was utilized to correct the structures in each step. The circuits in the first generation are

the result of an embryonic structure (varying from a single wire to a complete OpAmp) evolved with mutation operation, which is then sent through current flow analysis. A series of current flow lists is then created to describe the component connections based on the current flow in the circuit. Any correction rule, including altering component interconnects, removing isolated parts, and removing the parts without any effect on performance, is done within the current flow lists. The remaining circuits are then evaluated for fitness values. Upon unsatisfactory results, an iterative process of producing new generations continues until some desired results are obtained. Despite incorporating some sort of design knowledge in this work, not all the runs could lead to desired outcomes as observed by the authors. In addition, some problems exist even for the successful runs where the proposed method for identifying transistor operating regions is not effective enough to discriminate between triode and saturation regions. Reference [23] proposed a rule-based GA-oriented method on top of a topology-reuse scheme, which took analog building blocks as inputs to generate an embryonic circuit. The evaluation of the generated circuits was done in two steps: a simple evaluation based on the behavioral models and a more precise simulation-based sizing. Although the method was examined on comparators, oscillators, and XOR logic gates, it had difficulty in dealing with some structures such as pass transistor logic, current bias input, and independent resistors. Generally speaking, this proposed method can just be applied to the circuits composed of only MOS transistors.

In ANTIGONE [24] [25], an embryonic circuit is chosen based on a set of equations describing the functionality of the design. The performance of the design is calculated and fed into an evaluator, which determines the satisfaction level as a number between 0 and 1. A global satisfaction level is also defined to determine the best design in the population along with execution termination of the algorithm. If the optimization continues, the designs with low satisfaction levels

will be removed so that the population would be simplified. To improve the satisfaction level, a series of transformations can be applied both on the architecture and parameter values in the form of knowledge-based, equation-based or statistical transformations within a tree structure. This process continues until the desired results are achieved. However, the proposed method seems quite computationally expensive and the repeated optimization-evaluation stages would even lengthen the process to reach the final result. Moreover, the reported experiments only cover data converters at the system level.

In [26] [27], the search space is a set of building blocks organized within predefined templates. Each point in this space is called an individual. These individuals are organized in a hierarchical way with a parameterized context free grammar as a vector tree of parameter values using genetic programming. These parameters are needed to instantiate the root block for choosing topology and setting specific device values. A method named ALPS (age-layered population structure) is utilized to prevent premature convergence due to stealth mutation. NSGAI is used in the sizing procedure of the problems with 2 – 3 objective functions and TAPAS [28] is utilized for the problems with more than 3 objective functions to preserve topology diversity. This approach results in a Pareto optimal set consisting of sized circuit topologies. There is no feedback process to repeatedly improve the results. Operational amplifiers were synthesized and no more than two stages were considered to avoid a low convergence rate for the algorithm. Nevertheless, the reported experimental results showed that the algorithm failed to reach a good convergence rate for more complicated structures. References [29] and [30] targeted delta-sigma modulators and determined the optimum topology as well as the specifications of the required building blocks such that the system specifications were satisfied for the lowest possible power consumption. The proposed methods were customized only for a single class of applications, namely, ADCs.

Reference [31] applies an open-ended GA method to construct linear passive filter topologies. A string representation, in which a circuit is mapped to a chromosome, namely an array of genes corresponding to circuit components, is used. The genes include information about component type and connection nodes. However, this method is only confined to passive circuits and suffers from lots of meaningless structures generated. In [32], circuit structures are constructed by using an automaton. The automaton is directed by a set of instructions, which identify the circuit components, i.e., resistors, capacitors, inductors, and transistors. These instructions are manipulated by GA and represented in the form of strings, containing both structure information and parameter values. Passive filters and active amplifiers consisting of BJT transistors are synthesized. However, the reported designs are not optimized in terms of structure. Analog genetic encoding (AGE) is another type of string-based genetic representation introduced in [33], which permits simultaneous synthesis of topology and component parameters. However, since no circuit connection rules are involved, this method is probably prone to excessive circuit generation. Reference [34] uses developmental encoding system introduced in [33] for encoding the generation process in GA. Each topology (i.e., candidate solution or chromosome) is represented by a dynamic data structure and the composed genes carry the information about components, their connections, and values. However, only passive filters can be generated by this work.

2.2.2.2. Graph-based Topology Generation

Graph-based topology synthesis methods use graphs for circuit representation. In this way, they can utilize graph theory concepts and techniques to tackle the topology synthesis problem. References [35] [36] [37] strive to generate a special class of circuits, e.g., wide-band low noise amplifiers (LNAs), within a bottom-up framework. Considering the transistors as voltage controlled current sources (VCCS), graph theory is applied to find any possible structure consisting

of only one or two VCCSs. Although some newly emerged structures were able to be successfully implemented on silicon, the only rule applied to generate the topology in this work is to identify and discard structures with dangling components. Therefore, many useless structures would be generated even with just two transistors. In addition, the proposed algorithm just generates the core topology, whereas biasing circuits and buffers have to be added by the users. Similar work in [38] demonstrated an indispensable need for embedding a source of analog design knowledge into the synthesis procedure. Otherwise, roughly connecting circuit components together would end up fruitlessly due to a huge number of structures generated by the algorithm. Therefore, in the later endeavors circuit design rules have been always embedded in the synthesis algorithms. Additionally, constructing a circuit at a higher level of abstraction is often desired as the increasingly larger topology justifies the introduction of hierarchy.

In [39] and [40], a graph-based hierarchical top-down topology generation method based on a library of basic analog building blocks is proposed. These basic blocks are then utilized in an abstract form with specified terminal characteristics. Each abstract building block may represent several basic blocks, provided that all of these basic circuits share the same characteristics in their input and output terminals. Construction rules indicating the essential conditions to generate a meaningful structure are employed. The final topology would be generated in an abstract form, which is then expanded to its circuit level. Out of a huge number of generated circuits, symbolic analysis [41] is utilized as a preliminary estimation of circuit behavior in order to eliminate the less promising ones. Some improvements on symbolic analysis were proposed in [42] to more accurately discard the redundant topologies and hence boost the efficiency of the method. However, this approach generates lots of asymmetric circuits, which may not be favored by some analog circuits such as OpAmps. To address this issue, [43] considers the topology generation problem

with the possibility of expanding the topology in a symmetric way. This will also help reduce the size of the design space. Thus, when symmetric expansion is applied, the same abstract building block would not be expanded into two different basic blocks.

The use of abstract building blocks instead of single transistors can greatly decrease the size of the design space, which is very helpful when dealing with large designs consisting of many transistors. To further deal with the downside of generating lots of duplicate structures, isomorphism is a method employed in [59] as a destructive method after topology expansion to eliminate the redundant structures. A thorough aggregation of the works above was presented in [44], which demonstrated the synthesis of an elliptic filter comprised of four OpAmp stages in a hierarchical way. Moreover, Ma *et al.* [45] introduced an even higher level of automation in which ASDex, a description language, was utilized to automate the generation of different information required for the circuit synthesis procedure, e.g., VHDL-AMS model, netlist template of target circuits, test-bench netlist, scripts for calculating properties, and simulation control files. However, the method for topology generation is the same as those in [46] [47].

2.2.2.3. Hybrid-based Topology Generation

The highly challenging nature of the topology synthesis problem and lack of definite solutions have urged the researchers to leverage the merits of different strategies by combining them together to overcome this puzzle.

The contributions in [48] [49] are some examples in this respect. Das and Vemuri proposed a tree-based top-down approach for generating and sizing circuit topologies where an analog circuit was treated as a graph. The synthesis procedure starts with a top-level black box and goes through a step-by-step decomposition process. The method is different from other EA-based circuit

synthesis methods, which often start from an embryonic circuit and evolve to the final topology through a bottom-up approach by managing a lower level of abstraction. In this method, the library of building blocks is updated in each run and new blocks based on the results from the previous runs are added to the library. Despite adding diversity to the library and taking some precautionary solutions such as ranking the building blocks inside the library, since unknown topologies would be generated in each generation, there is no guarantee that a correct topology will be added to the library. Accordingly, it is probable that the library might be spoiled by some generated untrustworthy building blocks. One possible solution to avoid injecting incorrect elements into the library is to update it only after a trustworthy design is obtained rather than every time during the synthesis without verification.

2.2.3. Topology Refinement Methods

In most of the recent literature, topology selection is viewed as an inferior choice that has stalled since the beginning years due to its severe weakness (e.g., high computational effort and limited beneficiary applications). However, the ongoing lack of a well-organized and efficient synthesis tool, which can satisfy the expectations at an industrial level, has intrigued the researchers to further investigate into the topology selection methods. Topology refinement is aimed to modify an existing structure with alternative building blocks to improve the performance of the current topology. Usually the researchers, who are motivated to walk along this path, believe that the process of analog circuit design is more like a decision-making process rather than a generation procedure from scratch. They believe that the blackbox methods are not an appropriate approach for analog circuit design; instead in every single synthesis step the trade-off among circuit parameters must be taken into consideration to guide the synthesis process towards the right

and meaningful direction. In this dissertation, this concept is treated as a complementary stream to topology selection.

Ferent and Doboli [50] [51] compared several circuits and extracted their common/different features to improve an existing design by incorporating useful features from other designs. To this end, a feature clustering method, called ordered node cluster representation (ONCR), is utilized. A library of circuits with common functionality is provided and their equivalent uncoupled building block behavioral (UBBB) models, which are actually directed signal flow graphs (SFG), are extracted. Therefore, each circuit is represented by a set of nodes and two sets of symbolic expressions to identify the poles and the couplings between nodes. Then the nodes are matched based on similar AC behavior considering the poles and the outgoing couplings. A simulated-annealing-based scheme helps identify the classification curves through circuit nodes, which can offer the highest similarities. Clusters of the matched nodes from different circuits would be generated during this matching procedure.

In [52] two low-voltage amplifiers are compared based on the concept of UBBB models. Following topology comparison, performance constraints of each circuit are extracted and analyzed. The transfer functions of common blocks are treated as constants and the impact of dissimilar blocks would be the criterion to analyze the different constraints. A knowledge base is tabulated and analyzed considering the trade-offs between different circuit specifications due to variations in small-signal parameters of the transfer function. The data gathered in the table, which is helpful in understanding the advantages and limitations of each circuit block, can be used in the topology refinement. Reference [53] offers almost the same reasoning-based concept by extracting behavioral models and in-depth reasoning to identify and address the design bottlenecks through proper alternatives. These alternatives are implemented in the design flow with a tree structure

description called concept structure, where nodes indicate topology solutions (modified circuits) and arcs state the constraints to be satisfied in the next level sub-trees (nodes).

Reference [54] proposes a topology selection and refinement method based on the ONCR concept and a symbolic comparison method [55]. At the topology selection stage, based on the maximum amount of variance (maxdiff) from a reference circuit, the ONCR method is utilized to identify the proper topologies, which satisfy the maxdiff condition. The set of the selected circuits are then compared with the reference circuit and a trade-off profile will be generated to imply how structural differences affect the circuit performance. As a result, the candidates on the list are sorted to determine the most promising structure for improving the performance. At the topology refinement stage, the performance bottlenecks are identified through the trade-off analysis. The nodes, which are correlated to the specific devices that may cause bottlenecks, are recognized. Then the program tries to only locally modify the reference circuit to improve the bottlenecks.

Jiao *et al.* in [56] also benefited from a reasoning flow regarding the nature of the starting idea, that is, an analog circuit designer chooses to initiate a design. The proposed method mainly consists of two steps: 1) a procedure to select the starting idea, 2) a reasoning-based procedure to obtain the proper design based on the starting idea. Failure to begin the process with an appropriate starting idea may lead to performance defects in the final synthesis results [57]. A knowledge-mining technique over a library of 30 analog circuits is established to represent the analog circuit meta-knowledge [58], which continues and upgrades the main concepts of [55]. Different reasoning algorithms are considered for different starting ideas. This method is in the framework of divergent-convergent thinking expressed in cognitive sciences. They have grouped the synthesis flows into five categories based on the five starting ideas. Although these works are very appealing to revive the seemingly outdated topology selection methods, their proposed evaluation schemes

are more qualitative than quantitative, which may make it nontrivial for the readers to comprehend and reproduce the work. Moreover, little elaboration in these papers is provided to ensure the satisfaction of a design compared to the original specifications.

The feature-extraction-based method in [56] was further upgraded in [59] where in addition to automatically finding topological isomorphism of basic blocks, overlapping of different building blocks as well as recognizing repetitive structures of similar blocks, e.g., several connected Cascoded current sources or level shifters, were addressed in order to cover circuit schematics more efficiently. However, the proposed method was only applied to a small data set of 34 analog circuits. Moreover, the algorithm performance on run time is not clear from the paper. The core idea of [56] forms the basics of [60], which employs a three-level learning scheme, i.e., feature-, concept-, and constraint-level learnings for synthesis. A sequence of circuits as possible circuit solutions is analyzed in the feature-level learning. As a result, similar capabilities and limitations of different designs are detected and associated to common and distinct features. The concept-level learning determines the similar and opposing effects that each block may have on specific performance. The results of the concept-level learning specify the necessity to utilize or avoid using a block in the topology under synthesis. A novel low-voltage low-power amplifier was synthesized in [61] based on the strategy proposed in [56] to use a combination of abstract and physical features as the starting ideas in the synthesis procedure. To this end, a hierarchical graph-based knowledge structure at different abstraction levels including design features of low-voltage low-power OpAmps is utilized. Nevertheless, although good simulation results were achieved, the comparisons made in the paper show little fairness since simple simulation results in this work were compared with the results from some fabricated chips.

2.3. Summary

In this chapter, we have first discussed the main challenges that exist in the automation of circuit topology synthesis for analog ICs in the advanced technology era. Then, the previous automated analog circuit topology synthesis methods have been reviewed with their advantages and limitations pointed out. As can be seen, currently all the existing research works are not able to perfectly address the abovementioned challenges, which means there is still a long way to go in this research area.

In the next chapter, our proposed graph-grammar-based topology synthesis methodology will be detailed. It possesses not only wide applicability but also sound generalization capability, which can handle large-size and creative circuit topologies.

Chapter 3 Graph-Grammar-Based Topology Synthesis for Analog Integrated Circuits

3.1. Introduction

The challenges of the circuit topology synthesis lie on how to handle the following two key issues: 1) design space and 2) design expertise. As a matter of fact, the design space is so broad that exploring all the possibilities is impossible and unreasonable. Thus, shrinking the search space is extremely important. The design expertise is a double-edged sword. On the one hand, it is inevitably required to guide the synthesis process to meaningful directions within such a huge design space. On the other hand, an excessive amount of expert intervention would constrain the necessary exploration toward creative circuit topologies and obstruct the automation realization of the synthesis process. In this chapter, we propose a top-down hierarchical graph-construction-based approach to synthesize circuits, which can effectively reduce the design space and balance the design expertise embedded in the topology synthesis process.

The starting point of the top-down method is the final design, which is then decomposed into some appropriate sub-blocks until reaching the lowest level that contains only building blocks (BBs) [39] [48]. The abovementioned works treat resistors, capacitors, transistors, and subcircuits as BBs where only input–output terminal characteristics are known. The decomposition process is carried out by matching the input–output terminals of those BBs, which is guided by the defined decomposition rules. Because of utilizing BBs as the basic constructing components, the design space is largely reduced, the synthesis efficiency is greatly improved, and the synthesis result is more reliable. Similar to the other topology generation approaches [8], the proposed method also

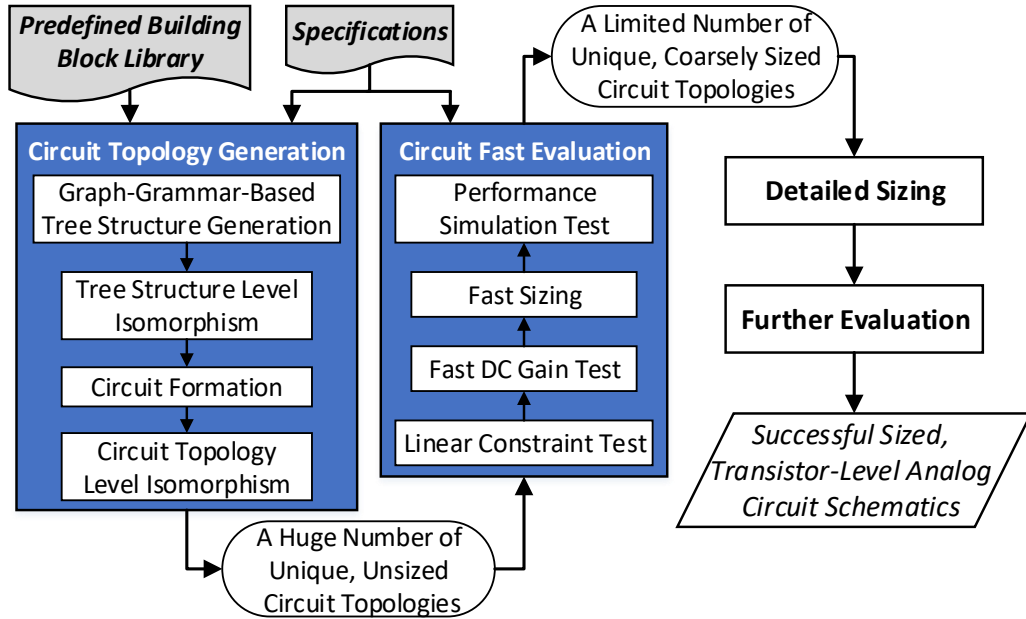


Fig. 3.1. Our proposed analog circuit synthesis framework.

experiences a common problem, that is, generating a huge number of un-sized circuit topologies as candidates, a large portion of which however does not actually satisfy the performance specification. This feature is beneficial for generating some novel topologies on the one hand, while it also shifts great computation burden and complex quality control to the subsequent evaluation process on the other hand. In order to efficiently evaluate the performance of these generated un-sized circuits, instead of directly applying the detailed sizing plus simulation to all of them, in this chapter we propose a novel fast evaluation method that can quickly filter the bad ones in terms of performance by trading accuracy for efficiency.

In the proposed circuit synthesis framework as depicted in Fig. 3.1, the detailed sizing, which can be readily carried out by any available commercial tools, is deliberately separated from the circuit topology synthesis. Two colored blocks, i.e., circuit topology generation and circuit fast evaluation, are the main focuses of this chapter. The key contributions of this chapter are listed as follows:

- An analog circuit topology synthesis framework, which greatly facilitates the design of analog integrated circuits.
- Graph-based circuit topology generator (GCTG).
- A novel fast evaluation method for un-sized circuits.
- Low-order symbolic transfer function generator (LTFG).

The research conducted on this topic has been mainly published in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [J1], and presented in 2018 IEEE International Symposium on Circuits and Systems (ISCAS) [C1] and 2019 IEEE International Symposium on Circuits and Systems (ISCAS) [C2].

3.2. Circuit Topology Generation

3.2.1. Graph-Grammar-Based Tree Structure Generation (GTSG)

In this section, generating circuit topologies is encoded as constructing graphs, with the construction process being controlled by the defined grammar. Formally, the graph grammar can be expressed as a 4-tuple $GTSG = (N_N, N_T, N_R, P)$, where the graph is composed of a finite set of normal nodes (N_N), a finite set of terminal nodes ($N_T, N_T \subseteq N_N$), and a root node ($N_R, N_R \subseteq N_N$), while the grammar consists of a finite set of production rules (P) that guide the graph construction process.

The graphs to be built by GTSG are actually trees, which are realized by iteratively splitting leaves starting from the root node. Each node in the tree, which can have up to two child nodes, represents a black block that is associated with only input and output terminals. The terminals can

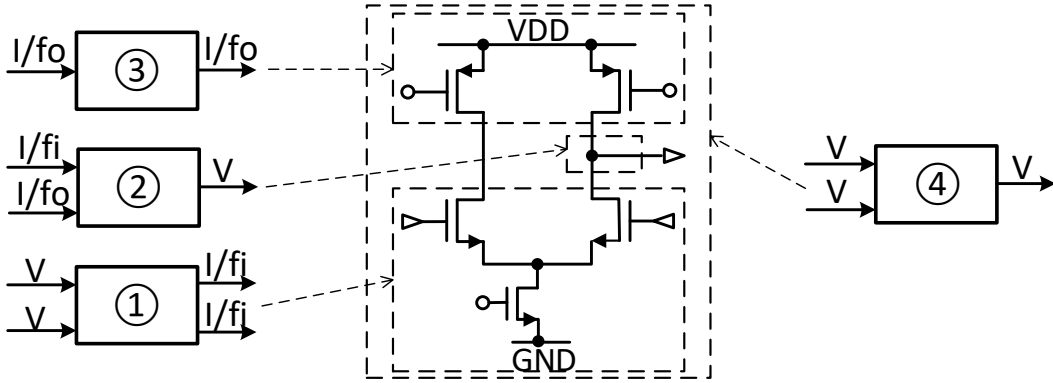


Fig. 3.2. Example of some tree nodes and their possible contents.

be classified into two types, voltage terminals and current terminals. For the current terminals, the direction of current, which should be included in the presentation, depends on whether the bias current (rather than the signal current) flows into or out of the block. Fig. 3.2 depicts one example with some tree nodes. One can see that input terminals always lie on the left side of a block while output terminals always stay on the right side. I/f_i or I/f_o represents a current terminal with bias current flowing into or out of the block, while V represents a voltage terminal. Thus, block ① might possibly be a differential pair with all their terminals matched in between. In addition, the detailed implementation of any block does not necessarily contain devices (e.g., transistors) as long as the terminal types are matched, as shown by block ② in Fig. 3.2.

Each tree structure contains a root node (N_R) that symbolizes the input-output specification of the represented circuits, as shown by block ④ in Fig. 3.2. For instance, for a two-stage operational amplifier (OpAmp) with the input-output specification of two voltage inputs and one voltage output, the encoded root node (N_R) would have two voltage input terminals on the left and one voltage output terminal on the right. If the terminals of the root node for tree-A match those of a non-root node within tree-B, tree-A can be treated as a sub-tree of tree-B. In this way, large and complex circuits can be divided into several components and then hierarchically implemented.

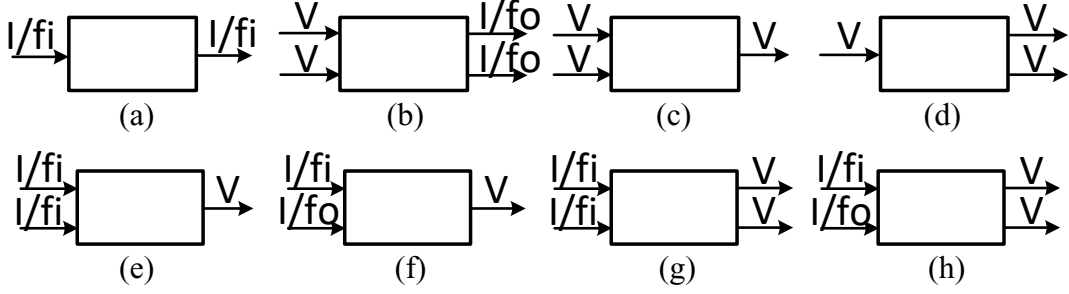


Fig. 3.3. Normal node examples: (a) One-signal-path block node. (b) Two-signal-paths block node. (c) Voltage-merger block node. (d) Voltage-splitter block node. (e) Identical-current-converto1 block node. (f) Distinct-current-converto1 block node. (g) Identical-current-converto2 block node. (h) Distinct-current-converto2 block node.

This is a major advantage of our proposed GTSG. The classification of normal nodes (N_N) and terminal nodes (N_T) is based on their functionality. Specifically, all the nodes in a tree structure are N_N , while only the leaf nodes that can be mapped onto circuit building blocks are N_T . It is worth noticing that not all the leaf nodes are necessarily N_T in a tree structure, but only the tree structures whose leaves are all composed of terminal nodes can be successfully translated into circuit topologies. For the example in Fig. 3.2, three block nodes ①-③ are terminal nodes since their mapped circuit BBs might be current mirrors, differential pairs, etc.

We have defined eight types of normal nodes in total. The names of these types are self-explained. Fig. 3.3 illustrates an example of each type of normal nodes. Generally, each input or output terminal can be either voltage, bias-current-flow-into, or bias-current-flow-out. For all the converto type block nodes, the input terminals must be current while the output terminal(s) must be voltage. For voltage-merger and voltage-splitter block nodes, both input terminals and output terminals must be voltages. We also have defined a set of production rules (P), which is composed of decomposition rule, structural symmetry rule, branch termination rule, deconstruction rule, and last-level constraint rule, to specify the way for constructing tree structures.

Table 3.1. Allowed decompositions for different types of block nodes

Type of Normal Node	Type of Decomposition Operation			
	<i>Horizontal</i>	<i>Semi-horizontal</i>	<i>Vertical</i>	<i>Voltage-division</i>
One-signal-path	√			√
Two-signal-paths	√		√	√
Voltage-merger	√			
Voltage-splitter	√			
Identical-current-convertor1	√	√		
Distinct-current-convertor1		√		
Identical-current-convertor2	√	√	√	
Distinct-current-convertor2		√		

1) *Decomposition Rule*

This rule guides how to split a leaf node into its leaf children, leading to a new tree structure. It consists of basic operations and optional operations. As mentioned in Section 3.1, the design space of the circuit topology generation problem is so huge that no method can explore all the possibilities. Therefore, we expect this decomposition rule to be an open rule, which means it is open for the users to add more reasonable optional operations to enhance the exploration for novel or specific interested circuit designs. In this regard, we have defined the following three basic decomposition operations, which are required to ensure reasonable construction of tree structures, and one optional operation for the listed different types of normal nodes as summarized in Table 3.1.

- Horizontal Decomposition

As depicted in Fig. 3.4(a), the left child keeps all its parent's input terminal(s), while the right child gets its parent's output terminal(s). Whether the left child owns one or two output terminal(s) depends on the type of its parent node. Specifically, for one-signal-path and two-signal-paths block nodes, their left children have one and two output terminal(s), respectively; for voltage merger or voltage splitter block node, its left child could have one or two output terminal(s). Furthermore,

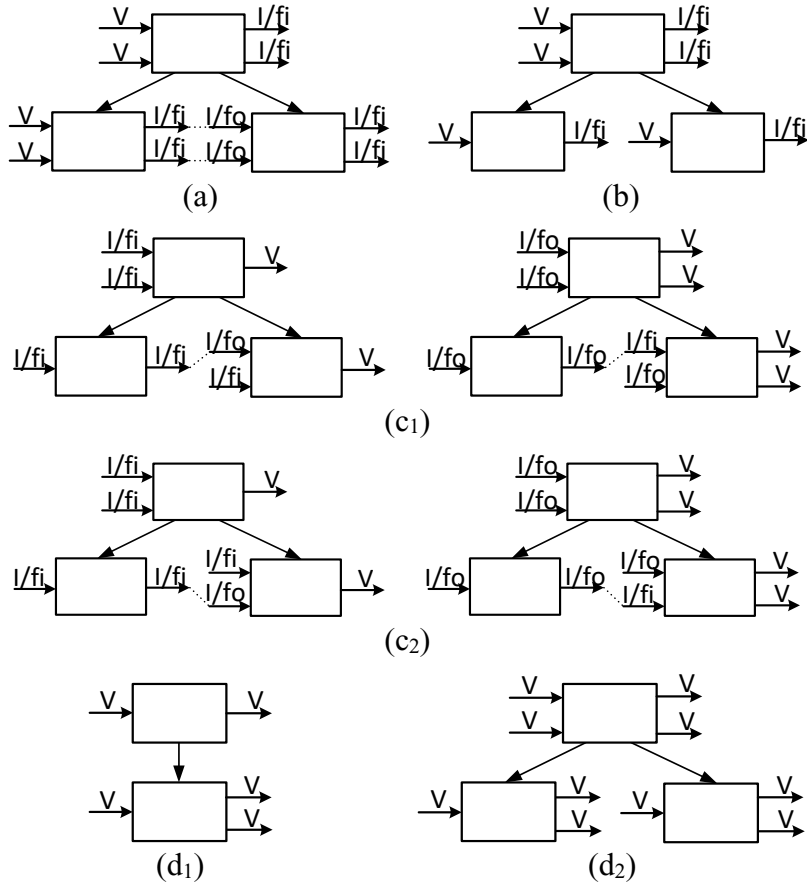


Fig. 3.4. Decomposition operation examples: (a) Horizontal decomposition. (b) Vertical decomposition. (c₁) Semi-horizontal decomposition case 1. (c₂) Semi-horizontal decomposition case 2. (d₁) Voltage-division decomposition for one-signal-path block nodes. (d₂) Voltage-division decomposition for two-signal-paths block nodes.

the output terminal(s) of the left child must be one-to-one paired with the input terminal(s) of the right child, as shown in Fig. 3.4(a). This pairing means that their terminals are physically connected. Such a pairing criterion encodes the reasonable connection rules between the two child block nodes, which requires that V must be paired with V , I/f_i with I/f_o , and vice versa.

- Vertical Decomposition

As depicted in Fig. 3.4(b), the first input and output terminals of the parent node are kept by the left child, while its second input and output terminals are reserved by the right child.

- Semi-horizontal Decomposition

We have defined this operation to include the following two cases. As illustrated individually in Fig. 3.4 (c₁) and (c₂), the left child holds its parent's one input terminal while the right child retains its parent's other input terminal and output terminal(s) are the commonplace of these two cases. The difference between them is that the left child takes the first input terminal of the parent node in Fig. 3.4(c₁) while the left child takes the second input terminal of the parent node in Fig. 3.4(c₂). Therefore, the output terminal of the left child is paired with the first input terminal of the right child in Fig. 3.4(c₁) and the second input terminal of the right child in Fig. 3.4(c₂).

- Voltage-division Decomposition

This is an optional operation that handles the situation of voltage division for exploring novel, complicated, or specific circuit designs. The voltage division may take place at one-signal-path or two-signal-paths block node with only voltage-type input(s) and output(s). For the one-signal-path case, it has only one child with identical input terminal and two duplicated output terminals of the parent node, as depicted in Fig. 3.4(d₁). For the two-signal-paths situation, as shown in Fig. 3.4(d₂), both the left child and the right child nodes have the same output terminals as the parent node, while the left child node keeps the first input terminal and the right child node maintains the second input terminal of the parent node.

Different from the three basic decomposition operations, the voltage division affects not only the node to be decomposed, but also other leaf nodes in the current tree structure. As depicted by an example in Fig. 3.5, the tree structure before the occurrence of voltage division is shown by all the red block nodes. The terminals with the same color mean that they are either inherited from the parent node, connected, or symmetric. After node ① experiences a voltage division operation, two more output terminals (i.e., the output terminals of its right child) are created. In order to

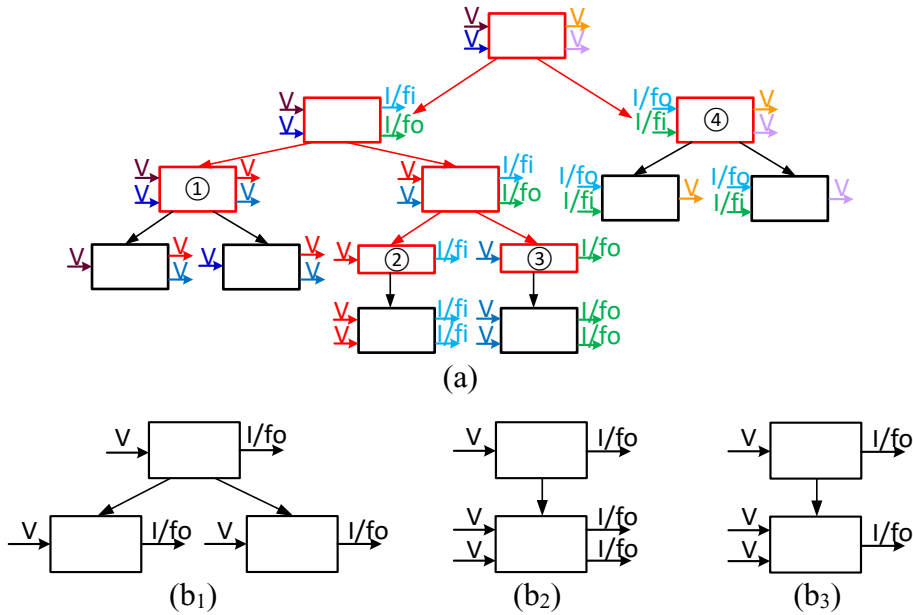


Fig. 3.5. Voltage-division decomposition: (a) Example of tree structure construction with voltage division. (b₁) The first case of internal signal division for one-signal-path block node. (b₂) The second case of internal signal division for one-signal-path block node. (b₃) Boundary signal division for one-signal-path block node.

receive these additional outputs, the leave(s) whose input(s) is (are) connected to them must also implement the operations that we call *signal division*, and so on until the last connected leaf. It is worthwhile to note that signal division is passively triggered by the incident of any voltage division for certain leaf nodes. Thus, we define that the signal division refers to the division operation that occurs to the triggered leaf nodes while the voltage division refers to the operation that only happens to the source leaf nodes.

According to the location where signal division happens, the division can be classified into boundary signal division and internal signal division, which occur to the last triggered (e.g., node ④) and the other triggered leaf nodes (e.g., nodes ② and ③), respectively. For one-signal-path block nodes, there are two possible cases to implement the internal signal division decomposition: the first case duplicates the parent's input and output terminals on different child nodes; the second

case copies the parent's input and output terminals twice to a two-signal-paths block child node, which are illustrated by the example of Fig. 3.5(b₁) and Fig. 3.5(b₂), respectively. When the internal signal division occurs to the other types of normal nodes, they have two child nodes whose input and output terminals are identical to their parent's, just like the first case of one-signal-path block nodes. Similarly, the boundary signal division decomposition doubles the parent' input terminals on the child node(s), but it keeps the number of the parent' output terminals on the child node(s), as depicted by the decomposition of node ④ in Fig. 3.5(a) and the example of Fig. 3.5(b₃).

2) *Structural Symmetry Rule*

This rule respects the necessary structural symmetry constraints (e.g., the first stage of OpAmps due to the preference of differential pairs) in the circuit design to reduce the chance of generating senseless circuit topologies [62]. Therefore, we require that when the vertical decomposition of a leaf node happens at the first stage, its two child nodes should be marked as symmetric. In order to track the stage depth during the tree structure construction process, we define that each leaf node owns an attribute of stage-depth, which is indicated by the appearance of current-to-voltage-converter block nodes and updated during the tree construction.

3) *Branch Termination Rule*

This rule halts a branch to be further disassociated in the tree structure construction process, which avoids producing a huge number of meaningless tree structures or asymmetric circuit topologies at the very early stage. Specifically, once two leaves are marked as being symmetric to each other, both are not allowed to be disassociated due to the concern that further splitting them would break or be difficult to maintain the symmetry property. For the same reason, when semi-horizontal decomposition happens at the first stage, the left child is forbidden to be further disassociated. It is worth noticing that this branch termination rule is only associated with the three

basic decomposition operations. In other words, when the optional voltage-division decomposition operation is applied, the branch termination rule would be disabled.

4) *Deconstruction Rule*

The decomposition process will never stop unless a termination constraint is applied. To avoid generating too many tree structures, the maximum number of leaves allowed in each tree structure can be controlled upon the user's input as listed in Algorithm 3.1. Furthermore, after the decomposition process is halted, any tree structures should be discarded if their leaves are not composed of terminal-only nodes.

5) *Last-Level Constraint Rule*

In order to improve the efficiency of the proposed GTSG algorithm, we apply the last-level constraint to the tree structures whose number of total leaves is $N-1$, where N is the maximum allowed number of leaves. For those structures, if any leaf node, except for the one to be disassociated, is not a terminal node, they should be discarded.

Algorithm 3.1 describes the whole tree structure generation, which is an explorative searching process. At each step, only one of the leaves is allowed to implement the decomposition. As explained in the decomposition operations, there are multiple possibilities to disassociate a leaf node. Each of these possibilities is treated as a forked version of the current structure and queued for further exploration. The defined decomposition rule (Line 13), structural symmetry rule (Line 15), branch termination rule (Lines 12 and 14), deconstruction rule (Lines 6 and 19), and last-level constraint rule (Line 9) interact with each other to control the quality and efficiency of the generation process.

Algorithm 3.1: Graph-Grammar-Based Tree Structure Generation

Input: Input-output specification S_{IO} ; A PBB library;
Maximum count of leaf nodes N .

Output: Candidate Tree structures.

1. Build a block node n_r that encodes S_{IO} ;
 2. Construct a tree structure T_{init} that treats n_r as root;
 3. Put T_{init} onto an empty list Q ;
 4. **While** (there is an un-fetched topology in Q)
 5. $T \leftarrow$ fetch the first unvisited topology in Q ; Mark T as fetched;
 6. **If** (T 's leaf count $n(T) \leq N-1$)
 7. **While** (there is an un-visited leaf node in T)
 8. leaf node $V \leftarrow$ extract the leftmost unvisited leaf of T ;
 9. **If** ($n(T) == N-1$ && any other leaf is a non-terminal node)
 10. Delete T from Q ;
 11. **Else**
 12. **If** (V is allowed to be decomposed)
 13. Decompose V ; Mark V as visited;
 14. Determine the termination property of V 's children;
 15. Determine the symmetry property of V 's children;
 16. Update the connection information in T ;
 17. Push each possible new topology T_{new} into Q ;
 18. Restore T to the one before disassociation;
 19. **If** (not all the leaves are terminal nodes)
 20. Delete T from Q ;
 21. Return Q ;
-

3.2.2. Tree Structure Level Isomorphism

During the tree structure generation process, there exist lots of chances to produce duplicate circuit topologies. Removing those duplicates at tree structure level can largely release the burden of further busyness. For the example in Fig. 3.6, even though (a) and (b) have distinct tree structure formations, since their leaves and sequence (from left to right) are exactly the same, we still consider (a) and (b) are isomorphic at the tree structure level because they will eventually produce the same circuit topology. It is difficult to manually detect this kind of isomorphism. However,

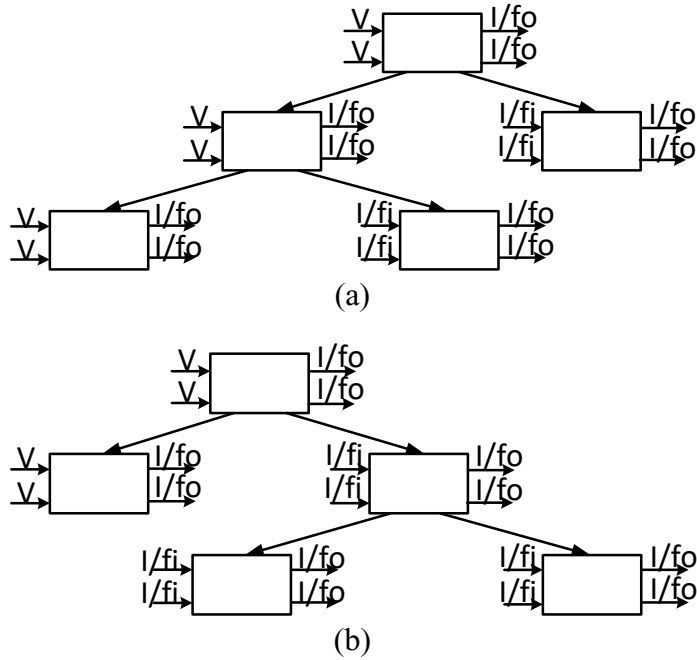


Fig. 3.6. An example of tree structure level isomorphism.

with the tree representation, automatically discovering the isomorphism is quite simple, which is another major advantage of our proposed GTSG.

Depth-first search (DFS) is a graph traversal algorithm with linear-time complexity. It is employed in our implementation to quickly extract the leaves from left to right. Once the extracted leaf sequences of two tree structures are exactly matched, the isomorphism at the tree structure level is confirmed, and then one of them has to be eliminated due to duplication.

3.2.3. Circuit Formation

The connection information and symmetry property among leaves are recorded and updated during the tree structure construction process. In addition, after applying the deconstruction rule, all the leaves of the generated tree structures should be terminal nodes that are able to be mapped to PBBs. With all the knowledge above, firstly the leaves of the tree structure are mapped to their

corresponding PBBs. Then transistor-level circuit topologies will be formed by connecting those PBBs according to the recorded connection information.

When decoding a tree structure, its leaf block node may be able to map to several PBBs, which depends on how the PBB library is defined. All these possibilities are preserved as forked versions of the current circuit topology. It is worth noting that if two leaf nodes are symmetric, they should be mapped to the same PBB for each possible choice. Furthermore, simply swapping input pins of an analog circuit may affect its performance. Due to this reason, we treat the circuits, which have exactly the same structure but swapped input pins, as different circuit topologies. All the formed circuit topologies should create copies of themselves with swapped input pins.

3.2.4. Circuit Topology Level Isomorphism

Although the previously applied tree structure level isomorphism test is powerful to eliminate all the duplicates, the later decoding process may also have a big chance to produce new duplicates that cannot be detected by the previous isomorphism check. Two example circuits that can be generated with the maximum-number-of-leaves of five are depicted in Fig. 3.7(b₁) and Fig. 3.7(b₂). Intuitively both circuits of Fig. 3.7(b₁) and Fig. 3.7(b₂) should be isomorphic, whereas their corresponding tree structures Fig. 3.7(a₁) and Fig. 3.7(a₂), whose connected terminals of leaves are indicated by the same colors, are not isomorphic due to distinct last leaves. Therefore, another isomorphism check at the circuit topology level is demanded to eliminate those new duplicates. The basic idea of this isomorphism test for analog circuit topologies was borrowed from [44], [47], and [63].

In those isomorphism algorithms, all the devices and nets are associated with labels. Initially, the nets that connect to pins will get special labels. Distinct types of pins will cause different labels.

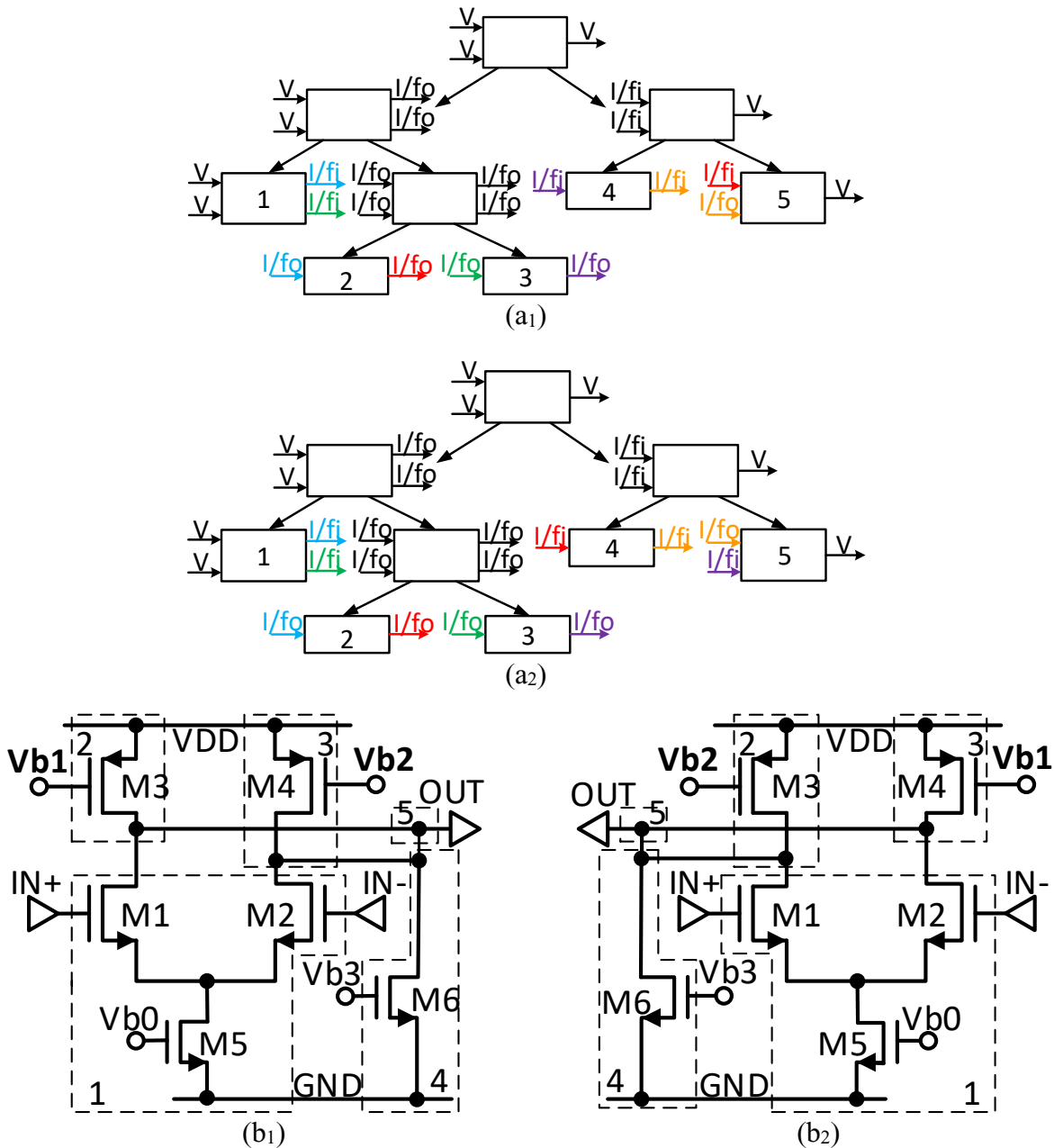


Fig. 3.7. An example of circuit topology level isomorphism.

For those nets that do not connect to pins will get labels according to their connections. Devices will get initial labels according to their types (e.g., NMOS or PMOS). Then all the devices and nets are partitioned into subsets according to their labels. If some subsets only contain one element, they are called *singletons*. Once those singletons are not matched for two circuits, isomorphism is

disapproved. If those singletons are matched, the labels of the devices and nets will be updated through a breath-first-search traversal. Then the newly generated singletons are compared. This process continues until all the subsets are matched singletons or isomorphism is disapproved.

However, we find that those published algorithms are still not reliable enough. In the example circuits of Fig. 3.7(b₁) and Fig. 3.7(b₂), since the gates of M_3 and M_4 connect to different biases, M_3 and M_4 would result in different initial labels that lead to a wrong check result. Actually, it is too difficult to give reasonable indexes for all the bias pins. Therefore, we believe that the indexes of the bias pins should not make the connected terminals different. In other words, all the biases had better be considered as the same type of pin just like the power or ground pin. However, even though the constraints of bias pins are given as mentioned above, the algorithm from [44] and [47] still fails to detect the isomorphism of Fig. 3.7(b₁) and Fig. 3.7(b₂) because (M_3, M_4) and (V_{b1}, V_{b2}) will always get the same labels, which make them always stay in the same subsets, respectively. Then the algorithm enters an infinite loop.

In this work, we have improved the existing algorithm as explained below. At each iteration of singleton-match checking, if there exist subsets that are not singletons, the device nodes (e.g., M_3 and M_4) from these subsets are first checked to see whether they have identical connections for all the terminals. If so, we consider their subset as singleton as long as the subset only contains these device nodes. Similarly, the net nodes (e.g., V_{b1} and V_{b2}) from these subsets are also checked to see whether they connect to the same type of terminals (e.g., drain, gate, source, and body) and the connected devices have identical connections for all the terminals or not. If so, we also mark their subset as singleton as long as the subset only contains these net nodes. For those subsets that fail in the above check, the labels of the devices and nets in them would be updated again through the breath-first traversal and the algorithm continues to enter the next iteration of singleton-match

checking. In this way, the circuits, which contain the devices that have identical connections for all the terminals, can be distinguished by our improved circuit-level isomorphism check scheme.

3.3. Fast Evaluation of Un-sized Circuits

3.3.1. Preliminaries

Our proposed work is based on Graph-Pair Decision Diagram (GPDD) algorithm [64] and GPDD-Reduction algorithm [65], a stream of the state-of-the-art symbolic analysis methods. To facilitate the understanding of our work, below we will give a brief review of both algorithms. Fig. 3.8(a) is a common source amplifier, whose small-signal model is given in Fig. 3.8(b), and GPDD data structure is shown in Fig. 3.8(c). Once a GPDD is constructed [64], the transfer function can be obtained from its topology as follows:

$$H(s) = \frac{(sC_{gd} - g_m)(g_1 + sC_l)}{(sC_{gd} + g)(g_1 + sC_l) + g_1sC_l}, \quad (3.1)$$

where $g = g_{ds} + g_2 = (R_{ds} + R_2) / (R_{ds} \cdot R_2)$ since GPDD processes all the impedance elements in the immittance form and parallel impedance elements as a lumped immittance.

If we just apply the GPDD algorithm to derive the I/O transfer function for even a simple analog circuit, the result is human unreadable since it contains a huge number of terms in both numerator and denominator. The GPDD-Reduction algorithm can largely reduce the number of terms through *0-operation* and *∞ -operation*. In (3.1), there are two ways to eliminate g_l (i.e., $1/R_l$) from the expression. The first one is to let $g_l = 0$ (equivalently $R_l = \infty$), by which the expression can be simplified as

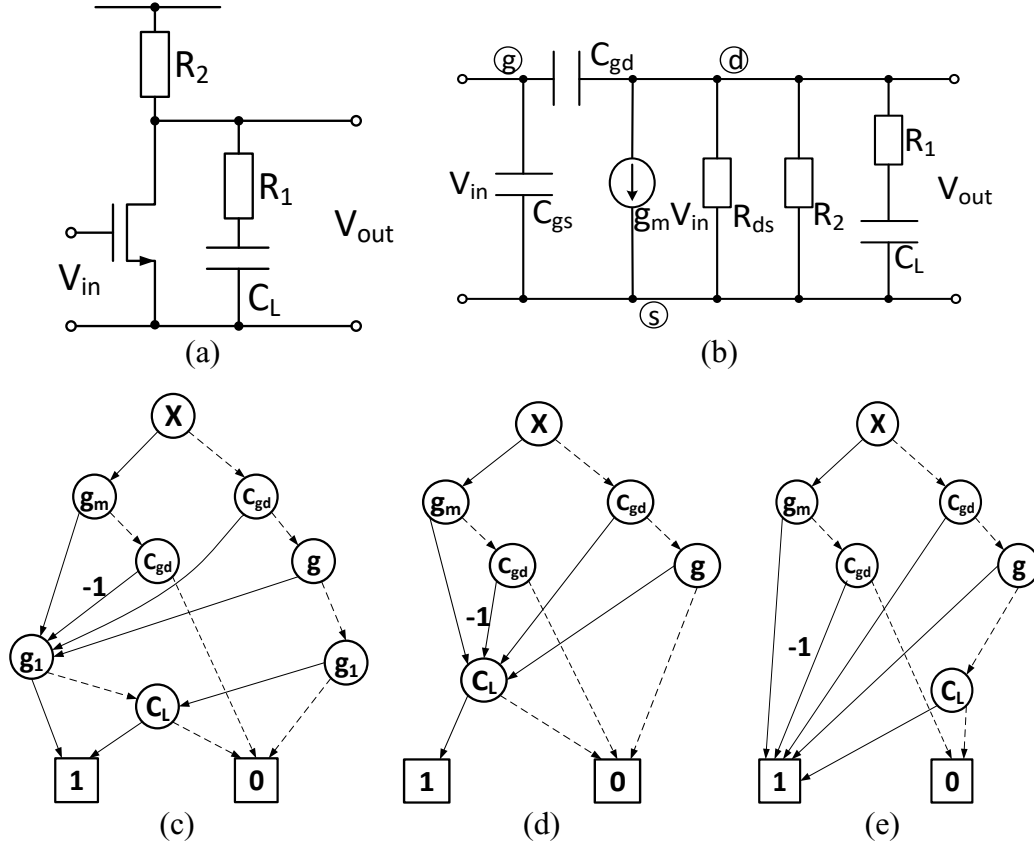


Fig. 3.8. GPDD data structure examples. (a) Common source amplifier. (b) Its small-signal model. (c) Its GPDD data structure. (d) Reduced GPDD after 0 -operation of g_1 . (e) Reduced GPDD after ∞ -operation of g_1 .

$$H(s)|_{g_1=0} = \frac{sC_{gd} - g_m}{sC_{gd} + g_{ds} + g_2}. \quad (3.2)$$

The other one is by letting $g_1 = \infty$ (equivalently $R_1 = 0$), the symbolic transfer function becomes

$$H(s)|_{g_1=\infty} = \frac{sC_{gd} - g_m}{s(C_{gd} + C_l) + g_{ds} + g_2}. \quad (3.2)$$

Both above have equivalent operations, which are called 0 -operation and ∞ -operation, in the GPDD-Reduction algorithm. Fig. 3.8 (d) and (e) show the reduced GPDD after eliminating element g_1 by 0 -operation and ∞ -operation, respectively. From these reduced GPDDs, the same symbolic transfer functions as shown by (3.2) and (3.3) can be extracted.

The GPDD-Reduction algorithm contains two phases. In the first phase, it firstly evaluates the

removing error of each symbol by applying both 0 -operation and ∞ -operation. Then the smaller error in between will be used to indicate the significance of the symbol, and the corresponding removing operation will be recorded. After that, the algorithm ranks all the symbols according to the increasing order of their significance. In the second phase, it sequentially removes the first K (a user-defined parameter) symbols on the ranked list.

3.3.2. Low-order Symbolic Transfer Function Generator

Symbolic analysis aims to derive human-interpretable expressions of circuit behavior. However, the interpretability of full-scale transfer functions is difficult to achieve. Therefore, a low-order symbolic transfer function, which is human readable as well as accuracy guaranteed, is of great interest. Thanks to the sequential reduction, the GPDD-Reduction algorithm can eliminate a large number of small-signal elements that do not play dominant roles in circuit performance. However, we found that during the reduction process, there existed a high possibility that some inner connections of the circuit are removed, leading to no transfer function to be extracted. This issue causes no trouble to the original GPDD-Reduction algorithm [65] because it aims to generate macromodel instead of symbolic transfer function. But this is not the case for our work.

Therefore, we have improved the algorithm to devise a robust low-order transfer function generator by adding invalidity and accuracy checking operations. In this regard, in order to avoid producing malfunction circuits, after ranking the significance of all the symbols, the ones with infinite removing errors should not be adopted in reduction [65]. Moreover, it is critical that the generated low-order transfer functions should bear an acceptable accuracy compared with the full-scale ones. To achieve this goal, we require that at each sequential reduction iteration, if removing one symbol causes an error larger than a certain percentage (user-defined parameter), that symbol

should be preserved in the final simplified GPDD structure. Compared with the original GPDD-Reduction algorithm, our improved reduction method in this work does not need to specify the number of symbols to be removed since it always keeps the minimum number of symbols under the requirement of certain accuracy. As a matter of fact, it is a tradeoff between degree of simplification and accuracy of the generated transfer function, which means if users require high accuracy, more complex symbolic transfer functions would have to be produced.

Moreover, the original GPDD-Reduction algorithm ranks the significance of all the symbols only once (before the reduction process) based on the primitive GPDD structure. In the process of sequential reduction, the GPDD structure may largely change after gradually reducing symbols. So, it is beneficial for the significance of the remaining symbols to be re-evaluated based on the reduced GPDD structure instead of always the primitive one. Otherwise, the ranked list would be highly inaccurate. Therefore, compared to the static decision method used in the original GPDD-Reduction algorithm, our proposed method dynamically determines which symbol has the least significance among the remaining symbols at each iteration when the GPDD structure is reduced.

After the low-order transfer function has been automatically extracted from the simplified GPDD structure, further effort is still needed to simplify it in the hierarchical way to reach the flat form. In addition, the device characteristics (e.g., g_m , g_{ds} , and intrinsic capacitances) of the symmetric transistors in some basic building blocks, such as differential pair, should have identical values. Thus, if those symmetric device characteristics are the common factors in both numerator and denominator, they should be eliminated from the final result. Through this post-processing operation, the final generated transfer function would be much cleaner and compacter.

3.3.3. Flow of Tests

The unique un-sized circuit topologies generated by the proposed circuit topology synthesizer above will be fast evaluated through a flow of tests, each of which acts as a filter to eliminate the circuit topologies that do not satisfy the requirements. The first one is the linear constraint test, which models the biasing constraints, symmetry constraints, and common constraints from a circuit topology as the linear equality or inequality constraints in the form of a linear programming problem. Once the modeled problem has a feasible solution, the circuit topology passes the test.

The survived circuit topologies will be further checked by the fast DC gain test. The DC gain of a circuit can be quickly numerically calculated through the GPDD algorithm [64], which requires the values of all the parameters in the small-signal model of the circuit as input. However, since the candidate circuits in our context are un-sized, it is impossible to get those values via SPICE simulation. What we know so far is that there exist value ranges for those small-signal model parameters in a certain technology. Therefore, after extracting such ranges for a specific technology, the center values of these ranges are employed to efficiently approximate the DC gain through the GPDD algorithm. In order to further speed up the operation, we employ a simplified small-signal model that only contains four parameters (g_m , g_{ds} , C_{gs} , and C_{gd}) for each MOSFET transistor in the circuits. Our experimental result indicates that even though center values are applied, the accuracy is still acceptable compared to the scenario of being fed with detailed parameter values. In practice, we can also deliberately lower the criterion of DC gain somehow to ensure that only the circuits with very bad performance would be discarded at this stage.

In order to further accurately evaluate the remaining un-sized circuits, we have proposed a new fast sizing method based on g_m/I_D (transconductance over drain current ratio) methodology. Recently the concept of g_m/I_D was employed in [66] to solve the parasitic-aware sizing problem.

The work of [67] further applied g_m/I_D and g_{ds}/I_D (drain conductance over current ratio), which are only related to the node voltages of V_{GS} , V_{DS} , and V_{TH} , to address the layout dependent effects during the analog sizing process. Moreover, the intrinsic parasitic capacitances C_{ij} mostly depends on $W*L$, where W and L are transistor's width and length, respectively; i and j are any of the drain, source, gate or bulk nodes of a transistor. It is known that I_D is proportional to W/L , thus

$$C_{ij}/I_D = L^2 f_{others}, \quad (3.4)$$

where f_{others} means the other effects that mostly come from oxide capacitance C_{ox} and gate overlap capacitance C_{ov} . Thus, if L is fixed, C_{ij}/I_D is also independent of transistor sizes.

As a summary of the explanation above, if L is fixed, g_m/I_D , g_{ds}/I_D , and C_{ij}/I_D can be directly expressed by only transistor node voltages. The curve-fitting technique is applied to reflect the relationship among them in the form of symbolic expressions. In this regard, after a batch of numerical simulations are performed on unit MOSFET transistor, the output sample data are fitted into analytic functions as follow:

$$\left(\frac{g_m}{I_D}, \frac{g_{ds}}{I_D}, \frac{C_{ij}}{I_D} \right) = f_k(V_{GS}, V_{DS})|L, \quad (3.5)$$

where k means that g_m/I_D , g_{ds}/I_D , and C_{ij}/I_D have different expressions. Moreover, the schematic-level circuit sizing can be modeled as a nonlinear programming (NLP) problem in the g_m/I_D form, with the objective function expressed as

$$obj = \alpha \sum_{i=1}^x \frac{\left(\frac{g_{m_i}}{I_{D_i}} \right) I_{ss}}{(V_{GS_i} - V_{TH_i})} + \beta I_{ss} V_{cc}, \quad (3.6)$$

where α and β are the weighting factors for total device area and power consumption, respectively [68]. Furthermore, the center values of the parameter ranges mentioned in the fast DC gain test

above are further used here to produce the low-order transfer function by our proposed LTFG tool as follows,

$$H(s) = h\left(g_{m_i}, g_{ds_i}, C_{gs_i}, C_{gd_i}\right). \quad (3.7)$$

By inserting (3.5) into (3.6) and (3.7), both the objective function and low-order transfer function can be expressed only in terms of bias parameters (e.g., V_{GS} , V_{DS} , and I_D). Nonlinear inequalities can be established between the transfer function and the predefined performance specifications. These nonlinear inequalities and the linear inequalities built in the previous test stages form the necessary constraints in our NLP model.

In this way, the NLP only contains the bias parameters of each MOSFET in the circuit as variables. The bias information is derived once the NLP problem is solved. Then by using the g_m/I_D theory or the look-up table search approach [67], W/L of all the transistors in the circuit can be derived. Since L can be pre-defined in our quick sizing process, W of each transistor would be readily calculated. With the size and bias information obtained from the solution of NLP, the candidate circuits are further directly evaluated via SPICE simulation. Once the performance of any circuits satisfies the predefined specifications within an acceptable error margin, they pass the performance simulation test and will be listed as the qualified output.

3.4. Experimental Results

This section is divided into six sub-sections. Sub-section 3.4.1 introduces the predefined building blocks used in our experiments. Sub-section 3.4.2 analyzes the results of circuit topology generation. Sub-sections 3.4.3 and 3.4.4 highlight the merits of our proposed low-order symbolic

transfer function generator (LTFG) and un-sized fast evaluation method, respectively. Sub-section 3.4.5 demonstrates that our proposed synthesis framework is able to synthesize innovative circuit topologies. Finally, our proposed work is compared with a state-of-the-art tool to show its advantages in Sub-section 3.4.6.

The whole framework was mostly implemented in C++, with the linear constraint test done by an LP solver, the fast sizing realized by the OPTI toolbox solver in MATLAB, and performance simulation test carried out by Cadence SPICE simulation tool. Our experiments were run on an Intel X86 1.2-GHz Linux workstation that has 64 GB of memory and the experiment was based on a CMOS 65-nm process, which can be readily replaced by any other CMOS technologies.

Table 3.2. PBB library

<i>Current Mirror</i>	<i>Wide Swing Cascode Current Mirror</i>	<i>Source-driven Current Splitter</i>	<i>Cascode Stage</i>	
<i>Common Source</i>	<i>Current Source</i>	<i>Differential Pair</i>	<i>One-Output Convertor</i>	<i>Two-Outputs Convertor</i>

3.4.1. Predefined Building Block Library

In GTSG, each terminal node can be mapped to at least one PBB. In this way, the tree structures comprised of terminal nodes can be decoded into the circuit topologies composed by basic BBs. As shown in Table 3.2, our defined PBB library contains 14 kinds of terminal nodes that can be mapped to 20 PBBs. Generally, each BB consists of nMOS/pMOS transistors, leading to distinct terminal types. Among them, one-output convertor and two-output convertor are two special PBBs due to only nets contained. Both have two kinds of terminal node correspondence, which is completely dependent on the order of the input terminals.

3.4.2. Results of Circuit Topology Generation

As illustrated in Fig. 3.1, our proposed GTSG algorithm takes input–output specification and PBB library as inputs. In our experiment, the input–output specification was defined as two voltage inputs and one voltage output with the PBB library provided as depicted in Table 3.2. Only the three basic decomposition rules were applied. Fig. 3.9 gives an example of synthesizing a three-stage OpAmp. Specifically, Fig. 3.9(a) illustrates how GTSG works to generate its tree structure with the maximum-number-of-leaves of twelve, while Fig. 3.9(b) shows the formation of its decoded circuit topology by mapping terminals from the tree structure above to their corresponding PBBs. It can be observed that a wider range of circuit topologies would be formed if given a more complicated PBB library.

In Fig. 3.9(a), leaves 0–6 belong to the first stage, leaves 7, 9, and 10 fall into the second stage, and leaves 8, 11, and 12 compose the third stage. As required by our symmetry constraint rule, leaf 5 is not allowed to be further split; otherwise, it is difficult to maintain the symmetry property of the first stage as depicted in Fig. 3.9(b). Since the parent of leaf 11 does not have this constraint,

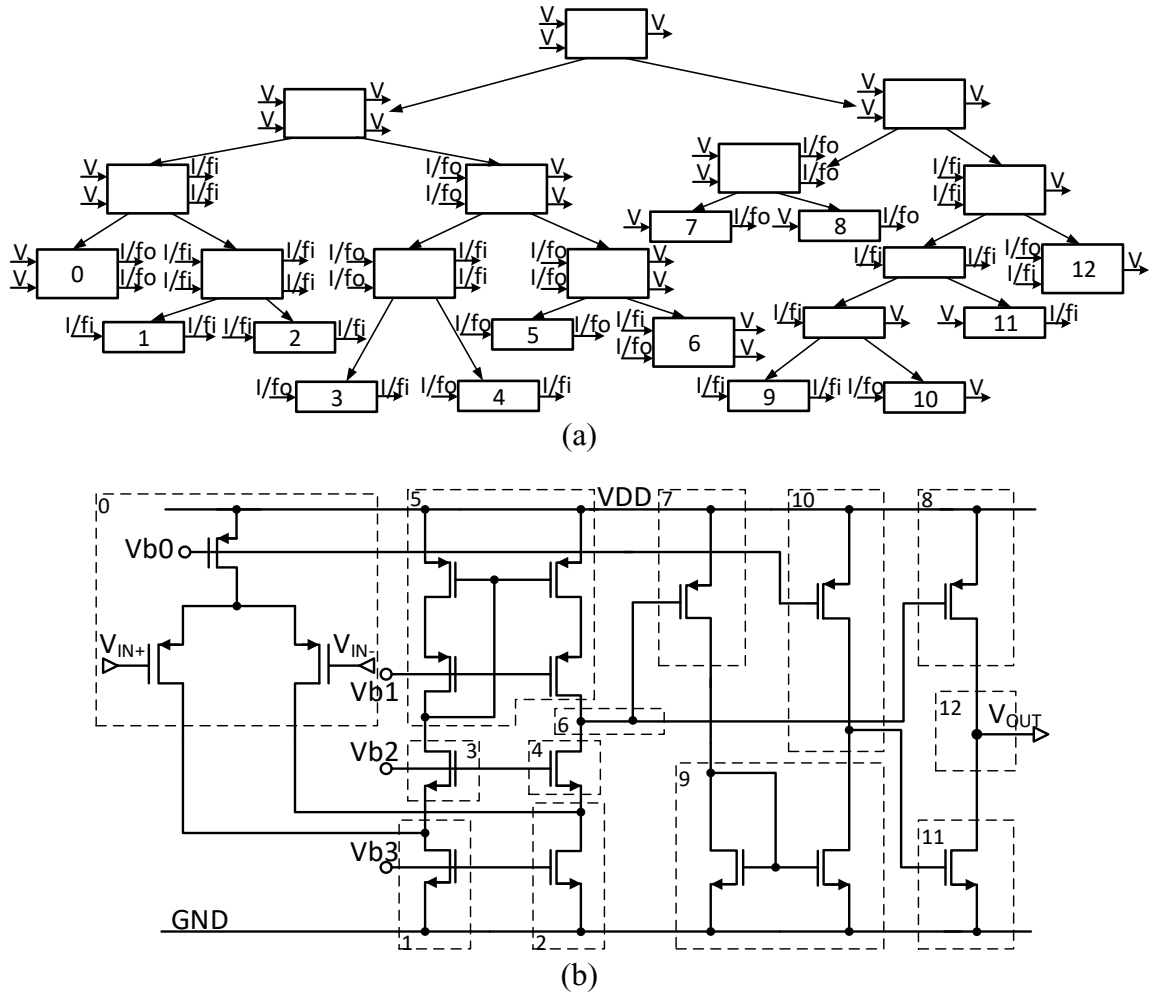


Fig. 3.9. An example of synthesis of a three-stage OpAmp. (a) Its tree structure. (b) Its circuit topology.

it can be further disassociated. Moreover, the significance of the last-level constraint rule is demonstrated by the experimental results listed in Table 3.6. Compared with the case without the last-level constraint rule, the efficiency of our proposed GTSG algorithm was greatly improved when the leaf number is larger than 4. It can be inferred that such an improvement becomes more significant as circuits get larger, since more leaves need to be processed.

The experimental results of the whole circuit topology synthesis are listed in Table 3.3. One can see that the tree structure level isomorphism is so powerful that a large number of duplicate

Table 3.3. Results of Our Proposed Circuit Topology Synthesis Framework

#Max Leaves	Circuit Topology Generation								Circuit Fast Evaluation							
	Tree Structures			Circuit Topologies					Whole Generation Time	LP Test		DC Gain Test		Total Time of LP and DC Gain Tests	Performance Simulation Test	
	Initially Generated	Tree-Level Isomorphism	Decoding	Swap Input Pins	Circuit-Level Isomorphism		#Left	%Pass		#Left	%Pass	#Left	%Pass			
	#	#Left	%Pass	#Generated	#Generated	#Left	%Pass	#Left		%Pass	#Left	%Pass	#Left		%Pass	
3	4	4	100%	12	24	12	50%	8.6ms	12	100%	0	0%	0.8s	-	-	
4	12	8	66.7%	24	48	22	45.8%	27.9ms	22	100%	0	0%	1.6s	-	-	
5	52	24	46.2%	108	216	100	46.3%	136.7ms	100	100%	24	24%	7.8s	18	75%	
6	388	64	16.5%	336	672	292	43.5%	3.9s	292	100%	120	41.1%	25.8s	72	60%	
7	3460	176	5.1%	1572	3144	1310	41.7%	68.2s	1310	100%	584	44.6%	139.5s	280	48%	

topologies are quickly filtered out, especially when the circuit size gets larger. When the maximum-number-of-leaves is 7, only 5.1% tree structures pass the check. Removing the duplicates at this level is of significance since it greatly reduces the subsequent workload of decoding and circuit level isomorphism, thus strongly improving the synthesis efficiency. As depicted in the PBB library, one terminal may be mapped to multiple PBBs. During the tree structure decoding process, all their combinations should be considered, while all the possibilities are treated as forked versions. Therefore, as shown in Table 3.3, the number of the generated circuit topologies after decoding is much larger than that of the tree structures. The circuit topology level isomorphism is also quite effective by eliminating at least half of the duplicated circuits at different circuit size levels.

The whole generation time in our experimental results as listed in Table 3.3, which is composed of the time consumed in tree structure construction, circuit topology formation, and two levels of isomorphism checking, clearly demonstrates the high efficiency of our propose GCTG. In GCTG, the most time-consuming parts are two isomorphism checks that only compare a pair of circuit topologies each time. When the number of the test circuits increases to n , the time consumed by the two tests would be increased to $O(n^2)$. That is, the main reason when the circuit size gets larger, the whole generation time, as shown in Table 3.3, increases dramatically.

Table 3.4. Example Results of LTFG

Two-Stage OpAmp				Folded-Cascode OpAmp			
Employing Values from Simulation Result (properly sized)				Employing Values from Simulation Result (properly sized)			
Simulation DC Gain	DC Gain by GPDD	DC Gain Calculated by LTF		Simulation DC Gain	DC Gain by GPDD	DC Gain Calculated by LTF	
64.15dB	64.96dB	65.08dB		65.62dB	65.08dB	69.92dB	
Simplified A_v	$A_v = -\frac{g_{m1}}{(g_{ds1} + g_{ds3})} * \frac{g_{m6}}{(g_{ds6} + g_{ds7})}$			Simplified A_v	$A_v = -\frac{g_{m2} * (g_{m9} + g_{ds9})}{(g_{ds2} + g_{ds11}) * g_{ds9}}$		
Employing Center Values Result (un-sized)				Employing Center Values Result (un-sized)			
g_m	g_{ds}	DC Gain by GPDD	DC Gain Calculated by LTF	g_m	g_{ds}	DC Gain by GPDD	DC Gain Calculated by LTF
0-200 μ S	0-2.5 μ S	63.98dB	64.08dB	0-200 μ S	0-2.5 μ S	66.65dB	70.21dB
0-300 μ S	0-5 μ S	58.94dB	59.08dB	0-300 μ S	0-5 μ S	61.68dB	65.25dB
0-400 μ S	0-10 μ S	51.83dB	52.04dB	0-400 μ S	0-10 μ S	54.68dB	58.28dB
Simplified A_v	$A_v = -\frac{g_{m1}}{(g_{ds1} + g_{ds3})} * \frac{g_{m6}}{(g_{ds6} + g_{ds7})}$			Simplified A_v	$A_v = -\frac{g_{m2} * (g_{m9} + g_{ds9})}{(g_{ds2} + g_{ds11}) * g_{ds9}}$		
Time Consumed to Generate LTF	0.27s			Time Consumed to Generate LTF	0.34s		

3.4.3. Evaluation of LTFG

For the evaluation of LTFG, we report the experimental results through two circuit schematics, two-stage OpAmp and folded-Cascode OpAmp, which are depicted in Table 3.4. Since the tool of LTFG was originally developed for properly sized circuits, we first employed sized circuits to analyze the accuracy and efficiency of LTFG. Then, LTFG was fed with center parameter values to test the feasibility and accuracy of its application to un-sized circuits.

After SPICE simulations were done on the sized circuits, the DC gain as well as small-signal model parameter values could be directly obtained. With those parameter values, the DC gain could be numerically calculated by using GPDD algorithm. In our experiments, it is slightly different (within 1dB) from the simulated results for both circuits due to the application of the simplified small-signal model. This result also exhibits the accuracy of our fast DC gain test. Then

the parameter values were fed into LTFG to generate the low-order transfer functions. From Table 3.4, one can see that the generated transfer functions for both circuits are highly simplified, whereas the accuracy is still maintained. The LTF generation time contains the time consumed in small-signal netlist generation, GPDD structure construction, GPDD structure reduction, and LTF extraction & post-processing. As depicted in Table 3.4, the LTF generation time for both circuits was within 0.4 seconds, which indicates high efficiency of our proposed LTFG.

In order to verify the applicability and accuracy of LTFG for un-sized circuits, we used three different ranges of g_m and g_{ds} with their center values fed to LTFG. As listed in Table 3.4 for both circuits, the generated low-order transfer functions (LTF) are the same not only for all three distinct ranges but also for the properly sized ones. This demonstrates the robustness and feasibility of LTFG for un-sized circuits. Compared with the DC gains calculated by the original GPDD, the ones calculated by LTF have at most $4dB$ difference for distinct ranges, which demonstrates the accuracy of LTFG for un-sized circuits. As discussed in Section 3.3, this accuracy can be fully controlled by the users through setting the allowed error margin (by default 10% in our experiment).

3.4.4. Results of Un-Sized Circuit Fast Evaluation

The experiment of un-sized circuit fast evaluation is the subsequent work after the earlier topology generation experiment. This part requires design performance specification in order to check the feasibility of those generated circuit topologies. In our experiment, only the performance attributes of DC gain and unit gain bandwidth are considered as the performance specification. Their requirements are listed in the top row of Table 3.5.

The evaluation experimental results are given in Table 3.3. It is interesting to observe that all the generated unique circuits passed the LP test. Although the LP test seems to have no effect in

Table 3.5. Result of fast sizing and performance simulation test of the folded-Cascode OpAmp and a creative circuit topology

Performance Specification			
<i>DC Gain</i>	55 dB	<i>Unit Gain Bandwidth</i>	10 MHz
Folded-Cascode OpAmp			
Results of Fast Sizing			
	<i>Width/Length</i>		<i>Width/Length</i>
<i>M1, M2</i>	32.15	<i>M4, M5</i>	15.75
<i>M6, M7</i>	11.1	<i>M8, M9</i>	6.83
<i>M10, M11</i>	44.25	<i>M3</i>	117.51
Results of Performance Simulation Test			
<i>DC Gain</i>	63.59 dB	<i>Unit Gain Bandwidth</i>	21.94 MHz
Creative Circuit Topology			
Results of Fast Sizing			
	<i>Width/Length</i>		<i>Width/Length</i>
<i>M1, M2</i>	10	<i>M3</i>	10
<i>M4, M5</i>	11	<i>M6, M7</i>	11
<i>M8, M9</i>	28.25	<i>M10</i>	31
<i>M11, M12</i>	5	<i>M13, M14</i>	5
Results of Performance Simulation Test			
<i>DC Gain</i>	70.31 dB	<i>Unit Gain Bandwidth</i>	66.48 MHz

filtering out circuits, it actually provides initial bias condition from its feasible solution to the NLP problem. On the other hand, the DC gain test is powerful by filtering out over half of input circuit topologies. The results of the fast sizing method are demonstrated through a generated circuit, the folded-Cascode OpAmp as depicted in Table 3.4. After modeling the circuit as an NLP problem and running the NLP solver, the size and bias information of all the transistors was derived. Feeding those parameter values to the SPICE simulation would produce the performance results. As shown in Table 3.5, since its performance attributes were satisfied with reference to the predefined specifications, this circuit passed the performance simulation test. The results of the performance simulation test for all the other candidate circuits are given in Table 3.3 (the last two columns), which indicates that only a portion of the generated circuit topologies retained as the final results.

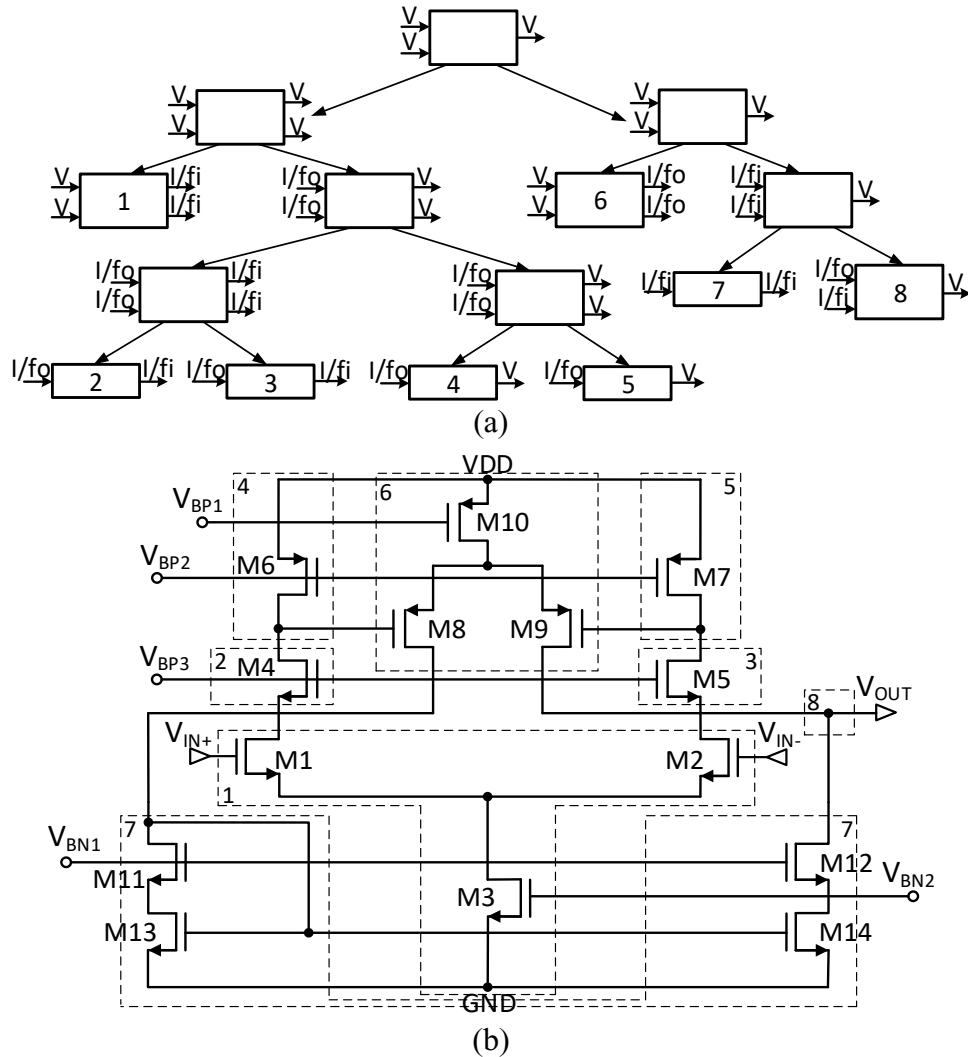


Fig. 3.10. A synthesis example of a creative circuit. (a) Its tree structure. (b) Its circuit topology.

3.4.5. Innovative Circuit Topology Synthesis

Technically, our proposed GCTG can exploit the following three means to potentially produce creative circuit topologies:

- 1) by enriching PBB library with creative building blocks;
- 2) by making creative connections among known building blocks;
- 3) by combining the two means above.

Fig. 3.10(b) depicts an innovative circuit topology that was synthesized by our GCTG with the maximum-number-of-leaves of eight. Its tree structure is shown in Fig. 3.10(a). In this creative circuit topology, there are six types of building blocks used, which are all known to us. But the creative connections of two differential pairs make this topology novel. As listed in Table 3.5, by applying our proposed fast sizing method and running SPICE simulation, the circuit passed the performance simulation test with the achieved DC gain and unit gain bandwidth of 70.31 dB and 66.48 MHz, respectively, which indicates high efficacy of such a generated circuit topology.

3.4.6. Comparison with State-of-the-Art Tool

In this part, we compare our work with a state-of-the-art academic circuit synthesis tool, FEATS [44]. In comparison, our proposed GCTG has the following main advantages to favor a wider scope of circuit topologies with more controllability than FEATS. Firstly, GCTG features a simple and accurate way to handle the circuit topology symmetry constraints. It is quite easy to locate the symmetric block nodes in a tree structure since only the vertical decomposition operation would form symmetric block nodes. Moreover, with the depth-track method, we can limit the symmetry constraints to only take place at a certain stage, which is necessary for designing multi-stage OpAmps.

Secondly, our proposed GCTG addresses the situations of current division and voltage division that are not covered by FEATS. Current division is handled by our defined source-driven current splitter building block as depicted in the PBB library. With this current splitter and our proposed stage depth tracking method, the circuits, such as the three-stage OpAmp depicted in Fig. 3.9(b) that cannot be generated by FEATS, can be synthesized by our GCTG. Besides the lack of defining any current splitters in the building block library, the fully symmetric expansion way

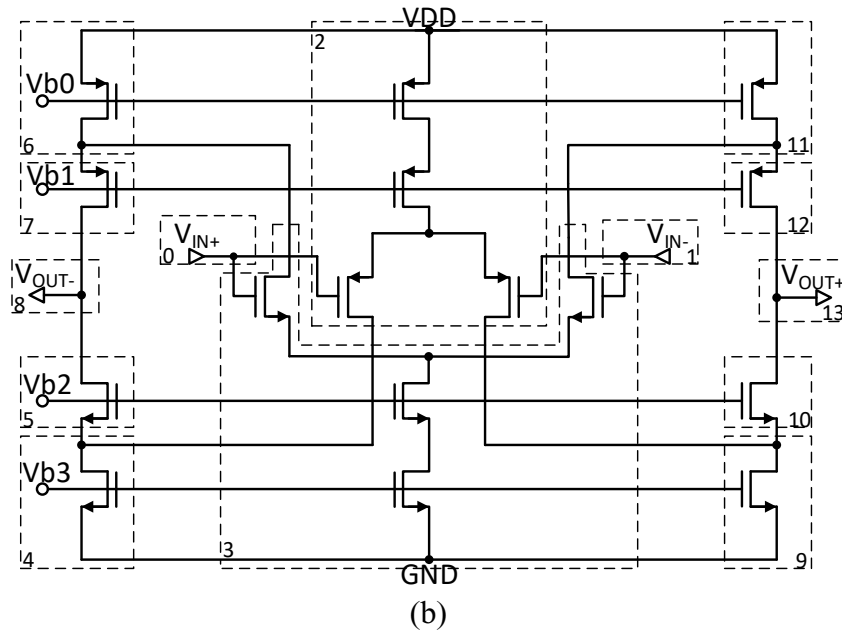
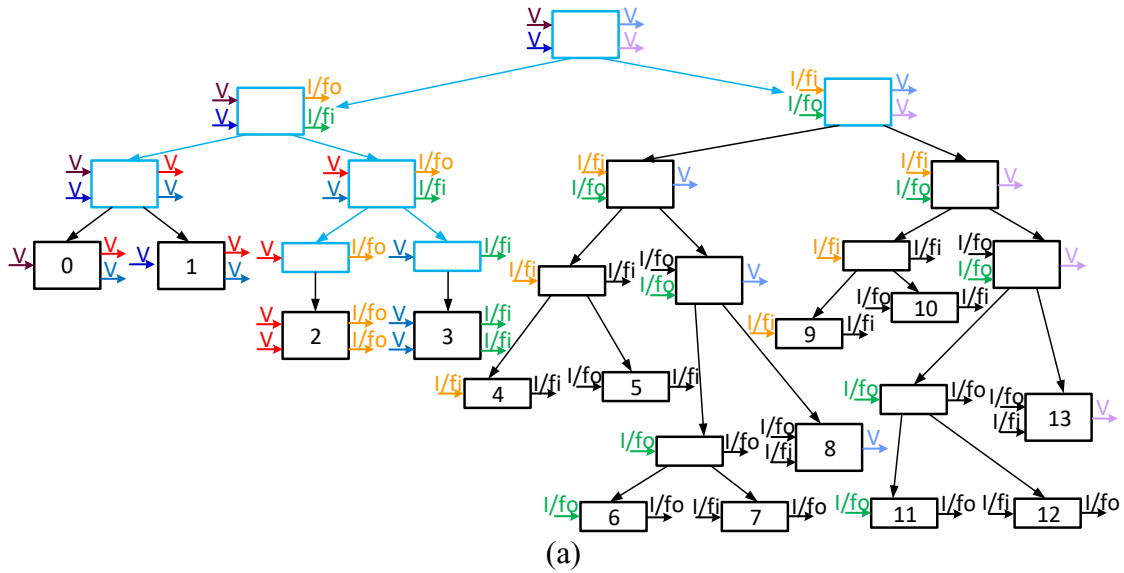


Fig. 3.11. An example of synthesis of an analog circuit. (a) Its tree structure. (b) Its circuit topology.

imposed by FEATS also impedes its generation of some circuits like the three-stage OpAmp, which only has symmetric relationship in certain stages. Voltage division is solved through our defined optional voltage-division decomposition operation. Fig. 3.11(a) describes how GCTG with the maximum-number-of-leaves of thirteen works with this operation to synthesize a nontrivial

Table 3.6. Comparison between FEATS and GCTG

# Max Leaves	#Topologies Generated Before Decoding		Time Consumed (μ s)		
			<i>FEATS</i>	<i>GTSG</i>	
	<i>FEATS</i>	<i>GTSG</i>		<i>Without Last-level Constraint Rule</i>	<i>With Last-level Constraint Rule</i>
3	4	4	1560	1158	1137
4	8	12	3636	3062	2856
5	28	52	4267	35920	6718
6	84	388	13021	843990	54152

analog circuit, which however cannot be generated by FEATS. In this figure, the terminals with the same color mean that they are either inherited from their parents, connected, or symmetric. The decoded circuit topology is depicted in Fig. 3.11(b).

Thirdly, GCTG possesses open topology generation rules, which means expanding the amount of design expertise embedded in the synthesis process can be readily realized by enriching the production rules or the PBB library. The basic three decomposition operations are sufficient to produce the most typical OpAmps, while the optional decomposition rules can help GCTG try more exploration in the design space, thus synthesizing more creative, complicated, or specific analog circuits. As the last note, our proposed GCTG has relatively lower efficiency as observed from Table 3.6. Nevertheless, it still can generate 388 tree structures within 0.06 seconds.

3.5. Summary

In this chapter, we presented a new framework for automatic analog circuit topology synthesis, which features wide applicability by its nature. It can be utilized to synthesize both voltage-type and current-type operational amplifiers (i.e., operational transconductance amplifiers). Although our proposed fast evaluation method can be treated as part of the established circuit topology

synthesis framework, they are actually not strongly coupled and can be deployed separately. Therefore, the proposed framework with certain adaptation can be also applied to synthesizing other analog circuits (e.g., analog comparators) or RF circuits (e.g., low-noise amplifiers (LNA)), followed by the symbolic-analysis-based fast evaluation of certain interested performance (e.g., propagation delay or noise figure). Our developed framework can significantly reduce the design time and provide fewer meaningful candidates for the designers to select. The final output circuit topologies would largely facilitate analog circuit design without intensive involvement of human designers in terms of design knowledge and experience.

Although our proposed fast evaluation method can filter out a large percent of the generated circuit topologies, the number of the remaining ones to be sized is still large, especially when circuit size is large. To further improve the efficiency of the whole synthesis process, in the next chapter we will propose a novel performance modeling method that can boost the sizing efficiency by more than 30 times with ignorable model building overhead, which is especially suitable for the circuit synthesis work that involves generating various circuit structures.

Chapter 4 Efficient Performance Modeling for Automated Analog Circuit Synthesis

4.1. Introduction

The circuit sizing is normally formulated as a constrained nonlinear optimization problem and can be solved by well-developed optimization algorithms. The simulator-in-the-loop process has often been employed in those optimization algorithms, which directly utilize the SPICE simulation results to drive the optimization process. To improve the optimization efficiency, various performance modeling methods, which can estimate circuit performance significantly faster, have been proposed in the past two decades to replace the computation-intensive and time-consuming SPICE simulator. Although building performance models is generally difficult and time-consuming, the well-built models would greatly reduce the time associated with the whole optimization process, especially for the optimization methods that require the computation of performance for a large number of alternative circuit sizes, such as simulated annealing (SA) and evolutionary algorithm (EA) [69].

A generic flow for the generation of performance models is provided as follows. The SPICE netlist of a circuit extracted from the schematic or physical design is the input to the modeling flow. At the first step, the circuit parameters (e.g., transistor length and width) and target characteristics or figures-of-merit are identified. The netlist is then parameterized for these parameters to obtain a parameterized netlist, and each of the characteristics will have a model. A sampling technique is employed to obtain discrete samples of the design parameters. Then the parameterized netlist is simulated through SPICE simulator for the target characteristics with the

sampled values of the design parameters. Finally, these collected sample points and their corresponding characteristics are utilized for the regression or fitting of performance models. In order to achieve sufficient accuracy, a suitable sampling scheme should be selected and a large number of simulations have to be conducted. More important, the complexity of this sampling process grows exponentially versus the dimension of the design parameters. A variety of modeling techniques have already been successfully applied to develop unbound performance models, such as equations [70], posynomials [71], polynomials [72], kriging methods [73], circuit matrix [74], support vector machine (SVM) [75], artificial neural network (ANN) [76], sparse regression [77], Gaussian process [78], Bayesian process, and genetic programming [79].

However, for the circuit electrical synthesis that involves circuit topology (i.e., structure) synthesis (such as [27]) rather than simply selecting a known topology among alternatives, the conventional performance modeling methods are no longer applicable since they all suffer from a common issue: expensive model building cost per circuit topology. In order to address this challenge, this chapter proposes a novel performance modeling method, which features ignorable model building overhead essential for variant topologies, yet still can efficiently and accurately estimate circuit performance. To the best of our knowledge, this is the first work that possesses this unique feature among others.

The research conducted in this chapter has been published in IEEE Transactions on Very Large Scale Integration Systems [J2].

4.2. Accurate Transistor Modeling

The traditional square law transistor model provides a simple and coarse way to understand the characteristics of transistors. However, as the transistor channel length scales down, the second-order effects, such as body effect, channel length modulation, velocity saturation, and mobility degradation, affect the transistor characteristics and thus have to be seriously considered. These second-order effects derail the accuracy of the square law model and render it untruthful [80]. The curve-fitting technique was utilized to capture those second-order effects to generate more accurate transistor models [81]. Since even a small change in channel length would change the relationship between transistor characteristics and their dependent variables, this type of curve-fitting modeling methods suffers from a difficult tradeoff: too loose division of channel length would cause the model less accurate, whereas too dense categories would drive the model to have extremely complex segments and corresponding expressions.

In order to further improve the accuracy and address the common issues of the curve-fitting-based transistor modeling method, in this chapter we apply artificial neural network (ANN) to model transistor behaviors, which can not only accurately capture the second-order effects, but also effectively avoid the complex segmentation issue. In recent years, ANN has been utilized to model the characteristics of FET devices thanks to the advancement of machine learning. Xu and Root surveyed the application of ANNs to measurement-based modeling of active devices [82]. They envisioned parallel training and implementation speedup as an area likely to yield significant future benefits. But this is less concerned in our work for analog circuit performance evaluation since the transistor modeling is just a one-time job for one specific technology. Most recently, Wang *et al.* proposed an ANN-based compact modeling methodology in the context of advanced FET modeling for design-technology-cooptimization and pathfinding activities [83]. Their focus

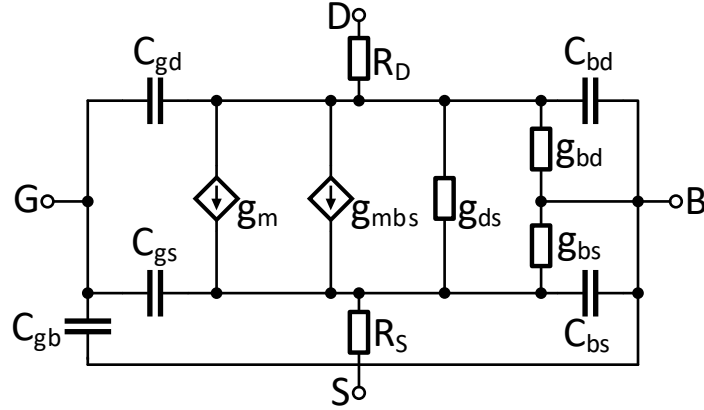


Fig. 4.1. Small-signal MOSFET model.

was to enhance the capabilities of ANN models (e.g., model retargeting and variability modeling) and to improve ANN training efficiency and SPICE simulation turn-around time. In contrast, our transistor modeling work in this chapter focuses on the MOSFET small-signal current-voltage and capacitance-voltage characteristics modeling for analog circuit performance analysis. In particular, we have developed various schemes below to improve the modeling accuracy of MOSFET small-signal parameters.

Our ANN-based model takes the values of a transistor's four terminal potentials, length, and width as input, outputs the values of the transistor drain-source current and the small-signal model parameters. The relationship between the output and input can be expressed as follows:

$$(I_{DS}, P_{SS}) = g(V_D, V_G, V_S, V_B, L, W), \quad (4.1)$$

where I_{DS} is the drain-source current; P_{SS} represents a vector of transistor small-signal model parameters as depicted in Fig. 4.1; V_D , V_G , V_S , and V_B are the bias potentials for drain, gate, source, and bulk terminals, respectively; L and W are the length and width of a transistor, respectively. Among the small-signal model parameters, R_S and R_D in Fig. 4.1, which are parasitic resistances

of metal routing wires to source and drain, are very small and thus ignored in our experimental implementation.

4.2.1. Data Sampling and Preprocessing

It is well known that a sufficient number of good quality training data are a necessary condition to produce an accurate regression model via ANN. The input and output pairs in (4.1) are sampled by performing SPICE simulation on MOSFET transistors within the design space. The piecewise uniform sampling scheme instead of the global uniform sampling scheme is applied in our work to collect data, considering the fact that transistor characteristics are more vulnerable when the channel length is short compared to the long channels. Therefore, for instance, if the SPICE simulations are carried out in a CMOS 65nm technology, more sampling points should be collected when the channel length is between 60nm and 200nm than that between 200nm and 400nm. The sampling process above is performed on both NMOS and PMOS transistors, and it is a one-time job for a specified technology process.

Since the values of input variables, as shown in (4.1), typically have very small units and quite different scales, we need to transform them to be in the same range and make them suitable as the input to neural network. Based on the requirements above, the values of each input variable are normalized as follows:

$$y = (x - min)/(max - min), \quad (4.2)$$

where x and y refer to the value before and after normalization, respectively, min and max are the minimum and maximum values of this variable in the dataset collected in the sampling process, respectively.

However, the normalization method above is not suitable to process the output variables since their values have too large variation ranges. For instance, the value of I_{DS} may vary from 10^{-9} to 10^{-2} A, making the small values extremely hard to be accurately fit since they contribute almost nothing to the training process. In order to solve this problem, we process the output variables with the following two steps: 1) scale all their values to be larger than 1 by multiplying a certain constant; 2) scale the output of Step-1 with logarithmic operation in order to largely shrink the variation range. The two steps above can be generally applied to the output variables whose values are always positive or negative. For the variables whose values could be both positive and negative, such as C_{bs} , their data has to be split into positive and negative sets, and then processed by the two steps above separately in each dataset.

4.2.2. NN Design

The mapping from input to output in (4.1) might be directly realized by building a neural network model, which contains 6 input variables and 11 output variables (i.e., I_{DS} and 10 small-signal model parameters as shown in Fig. 4.1). However, high accuracy is hard to achieve if so many output variables are fit simultaneously. In this work, we choose to fit them separately, which means one model just maps the inputs to one output variable. Since NMOS and PMOS transistors have distinct characteristics, two models will be built for each output variable of both transistor types, respectively.

Multi-layer perceptron (MLP) is capable of approximating any continuous multivariate functions to desired accuracy. Therefore, in this work we built all the models with the same MLP structure (i.e., three-layer perceptron), whose hidden layers have different numbers of neurons. As depicted in Fig. 4.2, each of our NN models is comprised of one passthrough input layer, one

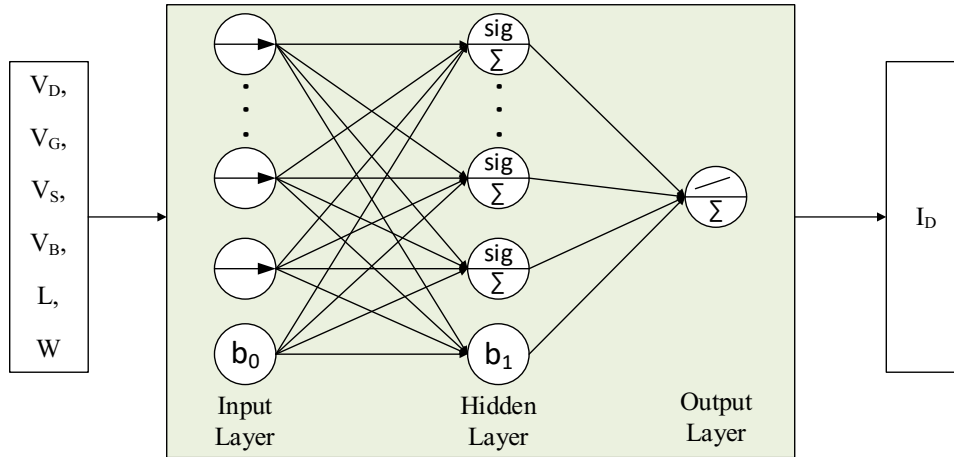


Fig. 4.2. Our NN structure.

hidden layer, and one output layer. The activation functions in the hidden and output layers are given by the widely used sigmoid function and linear function, respectively. For model performance evaluation, we select mean square error (MSE), which is a typical metric for regression problems. The connection weights in MLP are adjusted through the back-propagation learning, aiming at minimizing the MSE. Among various back-propagation learning algorithms, the Levenberg-Marquardt (LM) algorithm is selected thanks to its high efficiency and reliability.

4.2.3. Model Segmentation

Some output variables are difficult to build uniform NN models with high accuracy. This is mainly due to: 1) hard tracking data distribution trend; 2) quite large output variation range. The first issue can be addressed by sampling more training data, while the second problem can be overcome by model segmentation. In this part, we will discuss in more detail how to segment models to achieve high accuracy.

Segmentation of a model into several parts is an effective way to shrink the variation range of the output variables. For instance, if we segment the I_{DS} parameter model into two by its dependent

input variable V_{GS} from 0 to 0.6 V and from 0.6 V to 1.2 V, the variation range of I_{DS} would decrease from $[10^{-9}\text{A}, 10^{-2}\text{A}]$ to $[10^{-9}\text{A}, 10^{-4}\text{A}]$ and $[10^{-7}\text{A}, 10^{-2}\text{A}]$, respectively. According to our experimental studies, as long as the sampling data are sufficiently collected, a high level of accuracy tends to be readily achieved if a segmented model can limit the range of its associated I_{DS} to the scope of $10^{-(i+5)}$ to 10^{-i} (where i is a natural number). Otherwise, more segments are still needed to further decrease the parameter scope. In our segmentation scheme, one model is segmented into several parts according to its most sensitive input variable. In order to figure out which input variable is the most sensitive one to the model's output variable, sensitivity-analysis-based simulation trials are carried out to check whether the output variable variation range can be narrowed when the input variable is segmented [84]. The one that can shrink the range the most is treated as the most sensitive input variable.

4.3. DC Operating Point Computation

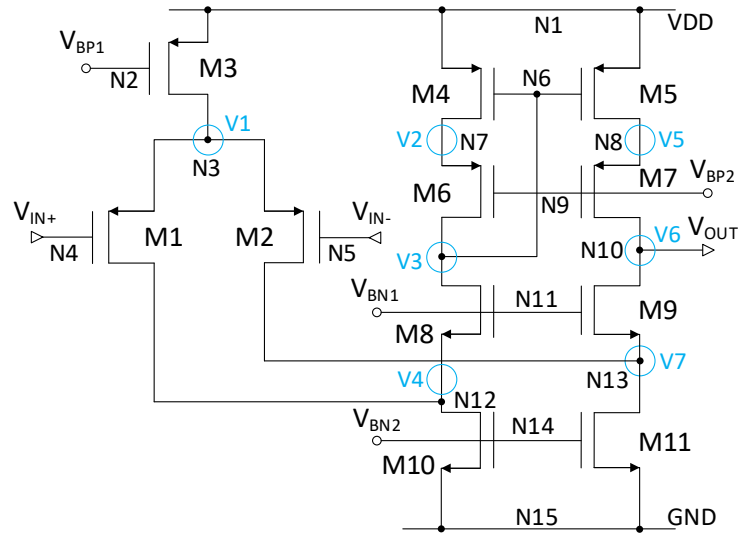
Based on the accurate transistor models derived in Section 4.2, we are ready to go to the circuit level to study the DC operating point of a target circuit by utilizing the established transistor models. The computation of the circuit DC operating points contains two phases, preprocessing phase and computation phase. The job of the preprocessing phase is to sort out the following three questions: 1) which net potentials are unknown; 2) the relationships among these net potentials and their bounds; 3) the relationships of I_{DS} among transistors. Learning these facts would greatly facilitate the work of the subsequent computation phase.

4.3.1. Circuit Preprocessing

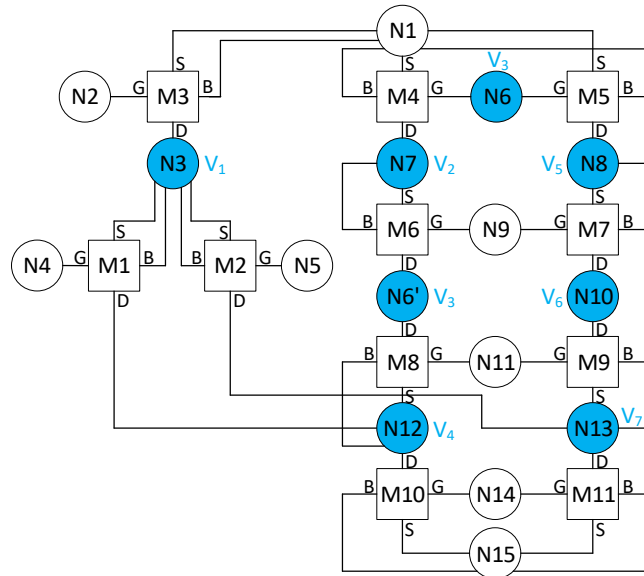
Certainly, the above-mentioned three questions can be answered through manual analysis of circuit netlists. However, we aim to automate this analytical process without human interaction, which is managed through the following two steps.

First of all, when reading a circuit netlist, an undirected bipartite graph (UBG) is simultaneously constructed with device nodes represented by squares forming one set of vertices, and net nodes represented by circles forming another set of vertices. Those device nodes and net nodes are connected via five types of edges: D , G , S , B , and P , where D , G , S , and B represent the connections to the drain, gate, source, and bulk terminals of transistor-type devices respectively, while P refers to the connection to passive-type devices (i.e., resistors, capacitors, or inductors). Each node possesses an attribute V to indicate its value: for a transistor-type device node, it refers to size; for a passive-type device node, it refers to the corresponding resistance, capacitance, or inductance; and for a net node, it refers to potential. During the UBG construction process, each node will get a value for its attribute V except for some net nodes that do not connect to circuit input pins. In this way, the unknown net potentials, which are treated as variables, are discovered. An example circuit (i.e., folded-Cascode OpAmp (FCO)) and its UBG are depicted in Fig. 4.3(a) and Fig. 4.3(b), respectively. In this UBG, we create an extra net node $N6'$ that represents the same one as node $N6$ to solely facilitate display. It is worthwhile to note that the connection mechanism for the transistor bulk terminal may vary in different designs. In this example, the bulk terminal and source terminal of any transistor connect to the same net node. Moreover, the net nodes with unknown potentials, marked as variables V_1 , V_2 , etc., are colored to illustrate their difference.

As the second step of the circuit preprocessing, the listed second and third questions above (i.e., relationships among the net potentials and among transistor I_{DS} 's) are efficiently extracted



(a)



(b)

Fig. 4.3. (a) The folded-Cascode OpAmp (FCO). (b) Its undirected bipartite graph (UBG) representation.

from the constructed UBG. Generally, analog circuits have to respect necessary symmetry constraints and biasing constraints in order to work properly and satisfy the design specifications. Among them, the symmetry constraints require that the symmetric transistors should have equivalent device characteristics (e.g., sizes, I_{DS} , g_m and intrinsic capacitances). Accordingly, from

(1), we know that as long as two transistors are symmetric and their any three terminals correspondingly connect to equal-potential net nodes, the net nodes that connect to their fourth terminals should have equal potential, too. The equivalent variables are treated as one variable to reduce the number of total variables. For the example UBG depicted in Fig. 4.3(b), all pairs of symmetric transistors, which are $(M1, M2)$, $(M4, M5)$, $(M6, M7)$, $(M8, M9)$, and $(M10, M11)$, are checked to see whether the condition above is met. Since all of them satisfy the condition, the equivalence of (V_2, V_5) , (V_3, V_6) , and (V_4, V_7) can be inferred. In this way, variables V_5 , V_6 , and V_7 can be removed so that only four independent variables (i.e., V_1, V_2, V_3, V_4) need to be solved in this circuit.

The biasing constraints, which can be specified by the designers, limit the operating regions of the transistors in the circuit. In our implementation, by default, we require that none of transistors is allowed to work in the cut-off region. This means in general all the transistors should follow the constraints below: $V_{DS} > 0$ and $V_{GS} > V_{TH}$ for NMOS, and $V_{SD} > 0$ and $V_{SG} > |V_{TH}|$ for PMOS, where the magnitude relationships among variables and their bounds can be derived. Since the threshold voltage (V_{TH}) of a transistor is determined by its size and biases that are still unknown, the transistors in the circuit might have distinct threshold voltages, which however are unclear before the detailed analysis. In order to address this issue, we uniformly assign a constant value to all transistors' V_{TH} , which is small enough to ensure to be less than the real various threshold voltages. Through breadth-first traversal (BFS) of the UBG starting from the net node connected to the power pin (e.g., node $N1$ in Fig. 4.3(b)), the upper and lower bounds of each variable can be efficiently confirmed by setting up the two inequalities above for all the visited transistor-type nodes. It is worth noticing that the bound of a variable could be another variable, which reflects the magnitude relationship between them. Learning this relationship can greatly help reduce the

search space of variables when they are being solved. For example, in Fig. 4.3(b), the relationships $VDD > V_2 > V_3 > V_4 > GND$ and $VDD > V_1 > V_4 > GND$ can be derived by applying the Kirchhoff's voltage law plus the constraint of $V_{DS} > 0$.

The relationships of I_{DS} among transistors can also be efficiently discovered with the aid of UBG. Specifically, all the sequences of transistor-type device nodes from the source node (i.e., the net node connected to the power pin) to the destination node (i.e., the net node connected to the ground pin) can be easily figured out with the aid of the BFS traversal algorithm when only D type and S type of edges are allowed to be visited, with the direction from source to drain for PMOS and from drain to source for NMOS, respectively. If two successive transistors only exist in one path, there is an equal relationship of I_{DS} between them. If more than one sequence contains the same transistor and the difference happens before or after such a transistor in the sequences, it means that there is a current merge or current division occurring around that transistor. For instance, there are four sequences in total can be extracted from Fig. 4.3(b): $M3 \rightarrow M1 \rightarrow M10$, $M3 \rightarrow M2 \rightarrow M11$, $M4 \rightarrow M6 \rightarrow M8 \rightarrow M10$, and $M5 \rightarrow M7 \rightarrow M9 \rightarrow M11$. Since $M4 \rightarrow M6$ and $M6 \rightarrow M8$ only exist in one path, $I_{DS,4} = I_{DS,6}$ and $I_{DS,6} = I_{DS,8}$ can be inferred. Since $M3$ appears in two sequences and the difference happens after it, there is surely a current division at $M3$. Similarly, since $M10$ and $M11$ also appear in two sequences and the differences happen before them, we can infer there are current mergences at $M10$ and $M11$. In this way, $I_{DS,3} = I_{DS,1} + I_{DS,2}$, $I_{DS,1} + I_{DS,8} = I_{DS,10}$, and $I_{DS,2} + I_{DS,9} = I_{DS,11}$ can be derived. It is worth mentioning that if the variables involved in one equation are all removed (because of symmetry), this equation should be removed as well. For instance, $I_{DS,5} = I_{DS,7}$, $I_{DS,7} = I_{DS,9}$, and $I_{DS,2} + I_{DS,9} = I_{DS,11}$ should be eliminated.

Algorithm 4.1 illustrates the proposed performance modeling construction process, which takes circuit netlist and design constraints (e.g., symmetry and biasing) as inputs. Lines 1-10 depict

Algorithm 4.1: Performance Model Construction

Input: Circuit netlist N ; Constraints C .

Output: Developed performance model.

1. Read N and construct its undirected bipartite graph (UBG);
 2. According to UBG and C , extract {
 3. the unknown net potentials V_i ($i=1 \dots n$);
 4. the relationships (R_V) among these V_i variables;
 5. the bounds (B_V) of these variables;
 6. the relationships of I_{DS} (R_I) among transistors; }
 7. Reduce the number of the variables in V_i according to R_V ;
 8. Build a polynomial equation set E of I_{DS} according to R_I ;
 9. Record the association of variables A_V in each equation in E ;
 10. Set the *Loss* function of E ;
 11. Construct the GPDD structure according to the UBG;
 12. **Return** $V_i, R_V, B_V, A_V, E, Loss$, and GPDD;
-

the circuit preprocessing for the input circuit while Line 11 describes the symbolic analysis model (i.e., GPDD structure) construction for performance estimation, which will be further explained in Section 4.5. Different from other performance modeling methods, our proposed performance model does not need developing or training time after it has been constructed. In other words, once the model is built, it is already a developed model. This unique characteristic makes our proposed performance modeling method feature ignorable model building overhead.

4.3.2. Computation Methods

To compute the values of those variables, the relationships of I_{DS} among transistors, which are automatically extracted in the preprocessing phase, are utilized to build an equation set that involves all the variables. For the folded-Cascode OpAmp depicted in Fig. 4.3, since the equivalent

variables are treated as one free variable, there are in total four variables (i.e., V_1, V_2, V_3, V_4), resulting in the following equation set:

$$\begin{cases} I_{DS,3} = 2 \cdot I_{DS,1} \\ I_{DS,4} = I_{DS,6} \\ I_{DS,6} = I_{DS,8} \\ I_{DS,8} + I_{DS,1} = I_{DS,10} \end{cases}, \quad (4.3)$$

where the subscript digits represent the transistor indexes in the circuit. If each transistor's I_{DS} has a symbolic expression, the solution is theoretically guaranteed since the number of variables is not greater than that of equations. However, our established transistor model for I_{DS} is numerical not analytical, leading to the equation set not being able to be directly solved out. Instead, we have to search for the proper values of all the variables by a small interval within their bounds, aiming to find the ones that satisfy all the equations in the set.

Nevertheless, no matter how small the interval is selected, it is almost impossible to always ensure to satisfy all these equations without fail. In order to address this issue, we define a loss function, which is the sum of the absolute difference between the left and right sides of all the equations in the set. Thus, the loss function for (4.3) can be defined as follows:

$$Loss = |I_{DS,3} - 2 \cdot I_{DS,1}| + |I_{DS,4} - I_{DS,6}| + |I_{DS,6} - I_{DS,8}| + |I_{DS,8} + I_{DS,1} - I_{DS,10}|. \quad (4.4)$$

In this way, directly solving the equation set is transferred to a problem of searching the values of all the variables within their bounds to identify the exact DC point that features the minimal loss. In this regard, we have developed four deterministic methods to solve such a problem.

1) *Brute Force*

The most straightforward method is to follow a brute-force search flow, which explores the possible value combinations of all the variables with a predefined interval within their bounds. The

solution is the point (i.e., value combination of variables) with the global minimal loss. By applying this method, the solution is deterministic, but the computation complexity is as high as $O(m^n)$, given n variables and m search steps for each variable. For example, assuming a circuit has five variables and each variable has 50 steps to search, the number of the total checking points would be 50^5 .

2) *Divide and Conquer*

Since solving an equation set means to deal with and satisfy all the equations altogether, the divide-and-conquer method is proposed aiming to shrink the search space of each variable by step-by-step solving all the equations in an equation set. It typically contains three steps: divide, conquer, and combine.

- 1) *Divide*: The original problem is divided into several sub-problems, each of which focuses on solving just one equation. In this way, the number of sub-problems is equal to the number of equations in the equation set.
- 2) *Conquer*: Each sub-problem (i.e., an equation) is solved by applying the above-mentioned brute-force method, aiming to derive the points that satisfy the absolute difference between the left side and the right side of the equation less than a very tiny threshold T (i.e., a user-defined constant).
- 3) *Merge*: The solved points of each variable are merged and uniquified, resulting in greatly reduced search space for each variable. These derived search spaces are utilized to discover the point with the global minimal loss through the brute-force method.

The detailed computation procedure of the folded-Cascode OpAmp is given in Fig. 4.4(a) by using the divide-and-conquer method. Firstly, in the divide step, the number of sub-problems is

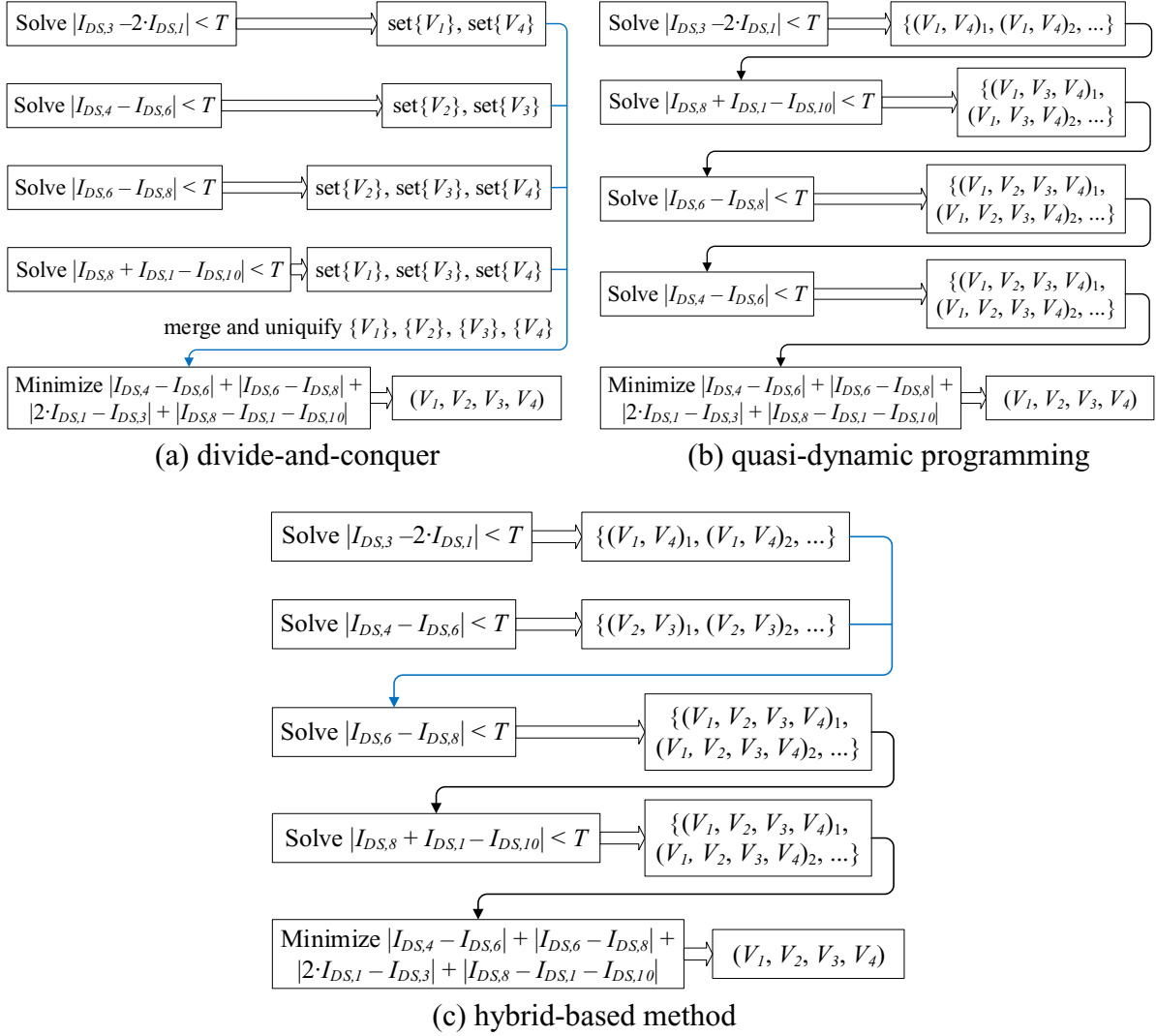


Fig. 4.4. The process of solving the variables in the folded-Cascode OpAmp.

confirmed to be four and each sub-problem one-to-one corresponds to an equation in (4.3). Then, in the conquer step, the sub-problems are separately solved by computing their corresponding inequalities (e.g., $|I_{DS,3} - 2 \cdot I_{DS,1}| < T$ and $|I_{DS,4} - I_{DS,6}| < T$). Finally, in the merge step, the satisfied points of variables (i.e., solution to each sub-problem) are combined and unquified. Provided that all the variables have reduced point sets, the brute-force method is employed to find the point with global minimal loss. Compared with the straightforward brute-force method, the computation complexity of this method is lower, based upon two main reasons as follows. First of

all, the number of variables involved in each sub-problem (i.e., equation) is less than that in the original problem. Secondly, in the merge step, the search space of variables is greatly reduced to derive the final solution.

3) *Quasi-Dynamic Programming*

The proposed quasi-dynamic programming method starts from solving one equation from the equation set that contains the least number of variables. But different from the divide-and-conquer method, the solved results are stored as tuples, each of which is a combination of the values of the involved variables. Then they are used to compute another equation that involves as few new variables as possible. This process continues until all the equations in the equation set are processed. The final resultant tuples, which should involve all the variables, will be used as search space to find the point with the global minimal loss (i.e., solution) of the entire circuit in the linear time.

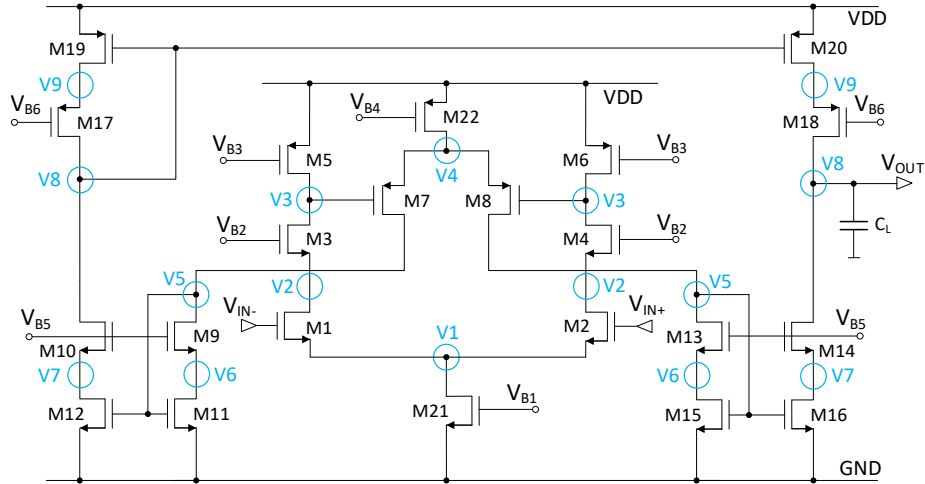
An exemplary quasi-dynamic programming process is illustrated in Fig. 4.4(b), which depicts a possible procedure to solve the variables in the folded-Cascode OpAmp. Firstly, it checks the association of variables in each equation (i.e., output A_V from Algorithm 4.1), starting with solving an equation that involves the least number of variables (e.g., $2 \cdot I_{DS,1} = I_{DS,3}$ that contains two variables, V_1 and V_4). Then, after checking the rest of the unsolved equations in the equation set in sequence, the solving process moves to an equation that involves a new variable (e.g., $I_{DS,8} + I_{DS,1} = I_{DS,10}$ and V_3) based on already involved variables. This process continues until all the variables are included. After that, it would continue to solve any unsolved equations in the set in sequence until all of them are solved to further refine the resultant tuples. During the whole process, the resultant tuples at the previous steps are fed to solve the next involved equation.

The computation complexity of the quasi-dynamic programming method is lower than the divide-and-conquer method, especially when the number of variables is large. The main reason for

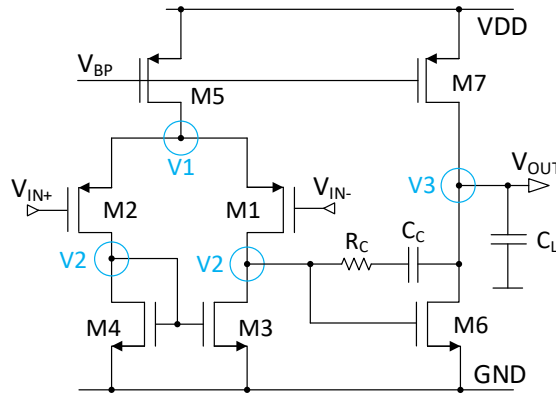
this is the solved results at each step are stored as tuples instead of singletons, contributing to the fact that the global minimal loss is searched in the linear time at the last step. But in the divide-and-conquer method, the global minimal loss has to be searched in the brute-force manner.

4) *Hybrid-Based Method*

This method is a hybrid of the divide-and-conquer method and the quasi-dynamic programming method, where the solving process is divided into several sub-processes and each of them is solved in the quasi-dynamic programming manner. Due to the characteristics of analog circuit structures, there always exist equations that contain only two variables (e.g., $I_{DS,4} = I_{DS,6}$ in Fig. 4.3(a), $I_{DS,18} = I_{DS,20}$ in Fig. 4.5(a), and $I_{DS,5} = 2 \cdot I_{DS,1}$ in Fig. 4.5(b)). The algorithm first identifies such equations, each of which is treated as the starting point of a sub-process for independent solving. Then the rest equations in the equation set are put onto an empty list Q . These equations would go through four levels of matching checks one after another. The first matching check examines whether the variables (V 's) contained in the equation to be checked are fully covered by the resultant tuples of one sub-process. If so, this equation will be solved along with the matched sub-process. The second matching check inspects whether V 's are fully covered by any two sub-processes. If so, these two sub-processes are merged after solving the matched equation with their resultant tuples. The third matching check verifies whether V 's contain one unvisited variable while the other variables are all in the resultant tuples of any sub-process. If so, this equation will be solved along with the matched sub-process. The fourth matching check discovers whether V 's contain an unvisited variable while the other variables are covered by any two sub-processes. If so, these two sub-processes are merged after solving the matched equation with their resultant tuples. If any equation fails in all these checks, it would be put to the end of list Q . Otherwise, this equation will be solved along with the matched sub-process(es). If this



(a) High-Gain OpAmp (HGO)



(b) Two-Stage OpAmp

Fig. 4.5. Another two operational amplifiers as the test circuits.

equation is unsolvable (i.e., the resultant tuples are none), the algorithm terminates to indicate no proper DC operating point existing for the current circuit setting. Otherwise, this equation is removed from the list to indicate it has been solved. This process continues until list Q is empty.

Fig. 4.4(c) give an example of the computation process by using this hybrid-based method for the folded-Cascode OpAmp. First of all, equations $I_{DS,3} = 2 \cdot I_{DS,1}$ and $I_{DS,4} = I_{DS,6}$ are identified as the starting point of two independent sub-processes. Then equation $I_{DS,6} = I_{DS,8}$ is found to fail in the first matching check but satisfy the second matching check. Thus, the two sub-processes above

are merged once this equation is solved with their resultant tuples. After that, equation $I_{DS,8} + I_{DS,1} = I_{DS,10}$ is identified to satisfy the first matching check. Thereby, this equation is solved along with the merged process. Since the variables associated with each equation (i.e., A_V output from Algorithm 4.1) and each sub-process are known, these four levels of matching checks can be done efficiently. Compared with the quasi-dynamic programming method, the hybrid-based method is more efficient when circuit structures become complex, such as the high-gain OpAmp (HGO) circuit depicted in Fig. 4.5(a). Since the solving operation of the quasi-dynamic programming is just a single process, it can only find the optimal solving sequence (i.e., the most efficient way) constrained by its starting point. However, the hybrid-based method has multiple processes with distinct starting points, which makes it be able to find the global optimal solving sequence by mutual merging. Thereby, the hybrid-based method can solve all the easily addressed sub-problems first and then extend the solving process step by step with minimal complexity increment, whereas the plain quasi-dynamic programming method cannot guarantee this smooth growth flow.

4.3.3. Handling of Unsolvable Cases

In Section 4.3.2, we only consider the situation where the DC operating point can be properly computed. As a matter of fact, any inappropriate combination of circuit biases and transistor sizes may cause the circuit to have improper DC operating points, further leading to abnormal performance results. In this part, we will complement our proposed DC operating point computation methods above by adding the handling of unsolvable cases. Once any of these cases are confirmed, the computation of the DC operating point should be terminated right away.

Specifically, before starting the DC operating point computation, all circuit biases are checked to see whether any transistor is unexpectedly working in the region that violates the given biasing

Algorithm 4.2: Hybrid-Based Performance Evaluation

Input: Output of Algorithm 4.1 (V_i , R_V , B_V , A_V , E , $Loss$, and GPDD);
Circuit biases V_B ; Device values V_D ; Constraints C ;
Transistor models M_T .

Output: Circuit performance.

1. **If** (circuit biases V_B do not satisfy C)
 2. **Return** Zeros;
 3. **Else**
 4. Identify equations e_i in E that contain the least variables;
 5. Treat each of e_i as the starting point of sub-process p_i ;
 6. Solve each of e_i based on V_B , V_D , B_V , R_V , and M_T ;
 7. Store the resultant values as sets of tuples $S_{T,i}$ for each p_i ;
 8. **If** any $S_{T,i}$ is empty, **return** Zeros;
 9. Put the rest of unsolved equations onto an empty list Q ;
 10. **While** (Q is not empty) {
 11. Fetch the first equation e in Q ;
 12. Apply the four levels of matching checks to e ;
 13. **If** e fails in all the checks, put it onto the end of Q ;
 14. **Else if** e is unsolvable, **return** Zeros;
 15. **Else**, delete e from Q ;
 16. Traverse the final resultant tuples to find the tuple T that has the minimal $loss$;
 17. **End If**;
 18. Calculate the small-signal model parameter values according to M_T , circuit biases V_B , device values V_D , and the values in T ;
 19. Update the small-signal model parameter values in GPDD;
 20. Calculate the circuit performance P with GPDD;
 21. **Return** P ;
-

constraints. If so, the circuit DC operating point is considered improper from the perspective of our proposed methods. Otherwise, a further check is needed to examine whether all the equations are satisfied, that is to say, whether there exist variable values to make the absolute difference between the left side and right side of each equation smaller than the threshold T . If any equation is unsatisfied, the DC operating point is treated improper. These two levels of checks can be easily integrated into the four deterministic computation methods presented in Section 4.3.2.

Algorithm 4.2 illustrates the performance evaluation process with the integration of unsolvable case checking, which takes the output of Algorithm 4.1, well-trained transistor models, and circuit biases and sizes as inputs, and outputs the evaluated performance of the circuit. In this algorithm, the hybrid-based method is adopted to compute the DC operating point, which is described from Line 1 to Line 17. The checking of circuit biases is conducted before computing the DC operating points (Lines 1-2). Then, at each equation solving step, the resultant tuple set is checked to see whether it is empty (Line 8 and Line 14). If so, it means the DC operating point is improper and thus the computation should be terminated. Lines 18-20 present the performance evaluation via GPDD. It is worthwhile to note that in this algorithm, solving an equation means to find the combination values of the involved variables meeting the condition that the absolute difference between the left and right sides of the equation is less than a tiny threshold value. Setting this threshold value needs to consider the trade-off between reliability and efficiency. In general, a smaller threshold value would degrade the reliability as it can easily cause the equation unsatisfied. In the meantime, such a smaller threshold value would enhance the efficiency since the smaller the threshold value is, the narrower the search space of the variables involved in the equation to be solved would become. On the other hand, a larger threshold value would produce the opposite effects.

4.4. Circuit Performance Evaluation

Our proposed performance model takes the values of circuit biases and transistor sizes as input, and outputs the determined amounts of the circuit performance attributes. The relationship between the output and input can be expressed as:

$$P = f(V_{bias_1} \dots V_{bias_x}, L_1 \dots L_y, W_1 \dots W_y), \quad (4.5)$$

where L and W refer to the length and width of each transistor, respectively; V_{bias} is the voltage bias; P represents circuit performances. Here we assume in total there are x voltage bias inputs and y transistors in the circuit.

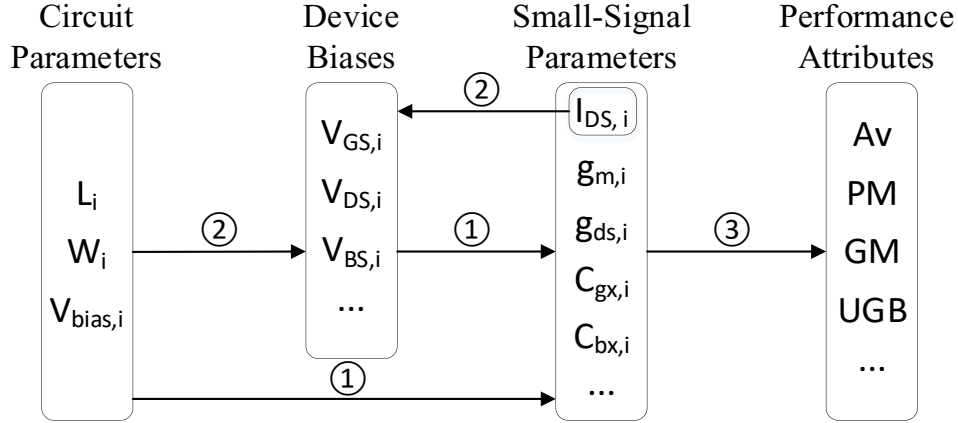


Fig. 4.6. Parameter mappings in our proposed performance evaluation flow.

Fig. 4.6 depicts the successive parameter mappings in the proposed performance evaluation flow. As reflected by mapping ①, the values of small-signal model parameters (e.g., g_m , g_{ds} , C_{gd} , C_{gs}) of each transistor in the circuit depend on its biases and sizes. Although the sizes of all the transistors have been provided as input, the biases of some transistors are still unknown. To solve those unknown biases, the relationship of I_{DS} among transistors is utilized to facilitate the DC operating point computation. This process is illustrated by mapping ②. Once the values of small-signal model parameters of all transistors have been derived, the circuit performance attributes (e.g., gain (A_v), phase margin (PM), gain margin (GM), unity gain bandwidth (UGB), etc.) can be computed out by using the symbolic analysis technique. In Fig. 4.6, subscript i is between 1 and y . And $C_{gx,i}$ covers $C_{gs,i}$, $C_{gd,i}$, and $C_{gb,i}$, while $C_{bx,i}$ represents $C_{bs,i}$ and $C_{bd,i}$.

As described in Sections 4.2 and 4.3, mapping ① and mapping ② are realized through the proposed transistor modeling method and circuit DC operating point computation methods, respectively. As a matter of fact, mapping ③ can be carried out by any symbolic analysis techniques, either in the formulation manner or in the graphic way (e.g., DDD and GPDD) [85]. In this work, we have selected GPDD as the symbolic analysis engine in our experimental implementation thanks to its sound efficiency and reliability. A GPDD is a bottom-up recursive computation data structure constructed upon the small-signal model of a circuit. Once it is constructed, the AC analysis can be performed for a given number of frequency points to derive circuit performance. Therefore, as listed in Algorithm 4.1, the model building overhead for a new circuit topology in our work is just the time used for circuit preprocessing and GPDD construction. By utilizing the undirected bipartite graph, the circuit preprocessing can be done very efficiently since most of the operations are completed in linear time. According to our experimental studies, the construction of GPDD is also very efficient, at most on the scale of seconds. However, for the other performance models as mentioned in Section 4.1, training or fitting a model typically needs some time of minutes. In addition, generally the sample data collection time would be significantly more than the training time. Therefore, our model building time is virtually ignorable compared to the other alternatives.

4.5. Experimental Results

This part is divided into four sub-sections. Sub-section 4.5.1 reports the evaluation of transistor modeling while Sub-section 4.5.2 compares the DC operating point computation

methods. Subsection 4.5.3 analyzes the proposed performance modeling, which is compared with the state-of-the-art methods in Sub-section 4.5.4.

We implemented our proposed performance modeling method in C++ and Matlab. Specifically, the SPICE simulation was realized by running Cadence Spectre simulator. The transistor modeling was carried out by using the Matlab Neural Network Fitting tool and then translated into C++ through the Matlab Coder tool. The symbolic analysis was implemented by using the GPDD technique in C++. Our experiments were run on an Intel X86 1.2-GHz Linux workstation that has 64 GB of memory. The experiments were conducted in the CMOS 65nm and 90nm technologies, which can be readily replaced by other similar CMOS technologies.

4.5.1. Evaluation of Transistor Modeling

Table 4.1. Applied Training Data Sampling Scheme for the CMOS 65nm Technology

W	<i>Range</i>	120nm – 500nm	1 μ m - 10 μ m	15 μ m – 80 μ m
	<i># Points</i>	5	10	14
L	<i>Range</i>	60nm - 200nm		300nm – 1 μ m
	<i># Points</i>	15		8
V_{DS}	<i>Range</i>	0 – 1.2V (NMOS) / -1.2V – 0 (PMOS)		
	<i># Points</i>	25		
V_{GS}	<i>Range</i>	0 – 1.2V (NMOS) / -1.2V – 0 (PMOS)		
	<i># Points</i>	61		

Table 4.1 lists part of our devised sampling division for transistor modeling in the CMOS 65nm technology, where every range is a closed interval with boundary points included. To mainly address the second-order effects that are hard to be accurately modeled in the previous works, in our experiments we focused more on the short channel transistors with transistor length varying from 60nm to 1 μ m and transistor width changing from 120nm to 80 μ m. The collected sampling data was processed with the preprocessing method discussed in Section 4.2.1, and then split into the training and testing datasets with the proportions of 80% and 20%, respectively. The transistor

Table 4.2. Testing Results of the Transistor Models (Percentages of the Testing Data that Achieved the Given Accuracy Levels)

Unsegmented Transistor Models (CMOS 65nm Technology)																	
Accuracy Level	99%	97%	95%	90%	99%	97%	95%	90%	Accuracy Level	99%	97%	95%	90%	99%	97%	95%	90%
Parameter	NMOS (%)				PMOS (%)				Parameter	NMOS (%)				PMOS (%)			
I_{DS}	77.96	98.96	99.93	100	84.73	99.52	99.94	100	C_{gd}	98.31	99.96	100	100	98.15	99.96	100	100
g_m	85.21	99.57	99.97	100	86.03	99.63	99.98	100	C_{gs}	99.56	99.97	100	100	99.85	100	100	100
g_{ds}	66.58	97.78	99.72	99.99	64.08	95.9	99.22	99.99	C_{gb}	92.67	99.93	100	100	95.86	99.98	100	100
g_{mbs}	81.69	98.77	99.91	100	85.24	99.6	99.95	100	g_{bd}	99.89	100	100	100	100	100	100	100
C_{bs}	23.78	54.43	73.91	91.64	24.34	59.46	76.97	92.11	g_{bs}	100	100	100	100	100	100	100	100
C_{bd}	18.87	51.38	72.32	91.46	29.89	70.59	86.74	97.09									
Segmented Transistor Models (CMOS 65nm Technology)																	
Accuracy Level	99%	97%	95%	90%	99%	97%	95%	90%	Accuracy Level	99%	97%	95%	90%	99%	97%	95%	90%
Parameter	First Segment								Parameter	Second Segment							
	NMOS (%)				PMOS (%)					NMOS (%)				PMOS (%)			
	$0 < V_{GS} \leq 0.42V$				$-0.5V \leq V_{GS} < 0$					$0.42V \leq V_{GS} \leq 1.2V$				$-1.2V \leq V_{GS} \leq -0.5V$			
I_{DS}	99.8	100	100	100	99.72	100	100	100	I_{DS}	99.99	100	100	100	99.65	100	100	100
g_m	99.57	100	100	100	99.87	100	100	100	g_m	98.93	100	100	100	99.89	100	100	100
g_{ds}	95.25	99.91	100	100	97.88	99.97	100	100	g_{ds}	88.08	99.5	99.94	100	83.12	99.51	99.97	100
g_{mbs}	99.8	100	100	100	99.69	100	100	100	g_{mbs}	98.88	100	100	100	99.91	100	100	100
Parameter	$60nm \leq L \leq 200nm$				$60nm \leq L \leq 200nm$				Parameter	$200nm \leq L \leq 1\mu m$				$200nm \leq L \leq 1\mu m$			
C_{bs}	44.01	81.4	91.93	97.47	45.71	83.52	93.37	97.97	C_{bs}	59.36	90.93	95.76	98.45	71.73	94.93	97.37	98.83
C_{bd}	27.24	65.09	83.08	94.15	42.15	85.47	95.65	99.34	C_{bd}	80.98	98.53	99.3	99.76	86.03	98.81	99.6	99.94

modeling was conducted in both the CMOS 65nm and 90nm technologies. To simplify our presentation in the chapter, below we just discuss the modeling results obtained from the CMOS 65nm technology, while the CMOS 90nm technology shows similar trends of the transistor modeling.

The testing results of the well-trained unsegmented parameter models are listed in Table 4.2, which compares the predicted and real output values of the testing data with the relative errors indicated. As one can see, regardless of NMOS or PMOS type, the C_{gd} , C_{gs} , C_{gb} , g_{bd} , and g_{bs} models are highly accurate, which can achieve 95% and 99% accuracy for 100% and over 90% of their testing data respectively, whereas the other parameter models are not that accurate. From our observation of experiments, we found that the variation range of C_{gd} , C_{gs} , C_{gb} , g_{bd} , and g_{bs} are narrow enough, contributing to their high accuracy. For the other parameters, as mentioned in Section 4.2.3, the segmentation operation can help improve their modeling accuracy. Among them,

I_{DS} , g_m , g_{ds} and g_{mbs} are most sensitive to V_{GS} while C_{bs} and C_{bd} are most sensitive to L . In our experiments, those parameter models are split into two according to their most sensitive input variable. The detailed division points and the testing results of the well-trained segmented models are given in Table 4.2. By comparing with the testing results of the unsegmented models, we can easily observe that the accuracy of the segmented models is significantly improved. Specifically, except for the C_{bs} and C_{bd} models, all the other segmented parameter models have improved to 90% accuracy for 100% of their testing data, and most of them even achieved over 99% accuracy for more than 90% of their testing data. The accuracy of C_{bs} and C_{bd} models is not as high as the others, but it is worth mentioning that their errors tend to have tiny impact on the circuit performance evaluation since their values are usually much smaller than C_{gd} , C_{gs} , and C_{gb} . It is expected that if more segments are utilized, the accuracy of the segmented C_{bs} and C_{bd} models would be further enhanced but at the cost of more time consumption for training the models.

These experimental results can exhibit the high accuracy of the NN-based transistor modeling method, which offers a strong foundation for the subsequent circuit performance evaluation. In order to further demonstrate the advantages of the proposed NN-based transistor modeling method, we compare it with the traditional square-law modeling method and the curve-fitting-based modeling method by following the procedure proposed in [81]. Here we only report the comparison of two important MOSFET parameters (i.e., I_{DS} and g_m). In our experiments, data were sampled by performing SPICE simulations on the CMOS 90nm technology, with the width range from 10 μ m to 50 μ m and length range from 400nm to 600nm (working in the saturation region). The sampled data were then split into the training and testing datasets with the proportions of 80% and 20%, respectively. For the square-law models, only testing data was used to calculate their average (*Avg.*) error as given in Table 4.3. The derived curve-fitting models of two split ranges

Table 4.3. Comparison of Various Transistor Modeling Methods (with Specified Size Ranges)

CMOS 90nm Technology				
Parameter	Square-Law Model	Curve-Fitting Model		NN Model
	L (400nm - 600nm)	L (400nm - 500nm)	L (500nm - 600nm)	L (400nm - 600nm)
NMOS				
I_{DS}	$0.5\mu_n C_{ox}(W/L)(V_{GS} - V_{TH})^2$	$0.0027WL^{-0.769}V_{GS}^{2.47}V_{DS}^{1.16}$	$0.0008WL^{-0.854}V_{GS}^{2.5}V_{DS}^{1.18}$	–
Avg. Error (testing)	23.92%	42.24%	41.17%	0.074%
g_m	$\mu_n C_{ox}(W/L)(V_{GS} - V_{TH})$	$0.0125WL^{-0.7239}V_{GS}^{1.07}V_{DS}^{0.537}$	$0.0018WL^{-0.8562}V_{GS}^{1.115}V_{DS}^{0.561}$	–
Avg. Error (testing)	38.51%	25.89%	24.67%	0.042%
PMOS				
I_{DS}	$0.5\mu_p C_{ox}(W/L)(V_{SG} - V_{TH})^2$	$3.27W^{0.242}L^{0.088}V_{SG}^{0.88}V_{SD}^{5.176}$	$0.053W^{0.276}L^{-0.229}V_{SG}^{0.87}V_{SD}^{5.1}$	–
Avg. Error (testing)	84.07%	56.1%	50.09%	0.36%
g_m	$\mu_p C_{ox}(W/L)(V_{SG} - V_{TH})$	$1.23 \times 10^{-5}WL^{-0.88}V_{SG}^{-0.84}V_{SD}^{-0.19}$	$1.12 \times 10^{-5}WL^{-0.77}V_{SG}^{-0.836}V_{SD}^{-0.2}$	–
Avg. Error (testing)	157.03%	95.17%	100.28%	0.99%

(i.e., L of 400nm-500nm and 500nm-600nm) are listed in Table 4.3. As one can see, the NN models of I_{DS} and g_m , which reach more than 99% average accuracy for the testing data, perform much better than the other two transistor modeling methods.

4.5.2. Comparison of DC Operating Point Computation Methods

For the evaluation of the DC operating point computation methods, we report the experimental results through two properly sized circuits, folded-Cascode OpAmp (Fig. 4.3) and high-gain OpAmp (Fig. 4.5(a)), which have four (i.e., V_{I-4}) and nine (i.e., V_{I-9}) distinct variables, respectively.

The computation results of the proposed four methods are compared in Table 4.4, where methods BF, DC, QDP, and HB refer to the brute-force, divide-and-conquer, quasi-dynamic programming, and hybrid-based method, respectively. As one can observe, all four methods can efficiently compute the variables for the folded-Cascode OpAmp, whereas only QDP and HB are applicable to solving larger circuits like the high-gain OpAmp. Since the computation complexity of BF and DC is exponential to the number of the involved variables, their runtime is more than 1

Table 4.4. Results of DC Operating Point Computation and Symbolic Analysis of Two Properly Sized Circuits

Folded-Cascode OpAmp (CMOS 65nm Technology)						High-Gain OpAmp (CMOS 90nm Technology)											
DC Operating Point Computation (Properly Sized)						DC Operating Point Computation (Properly Sized)											
SPICE Simulation Result						SPICE Simulation Result											
V_1 (V)	V_2 (V)	V_3 (V)	V_4 (V)			V_1 (V)	V_2 (V)	V_3 (V)	V_4 (V)	V_5 (V)	V_6 (V)	V_7 (V)	V_8 (V)	V_9 (V)			
0.9328	1.091	0.8548	0.1959			0.0141	0.2995	0.3193	0.6223	0.2665	0.2153	0.2293	0.8732	1.017			
Computation Result						Computation Result											
Search Interval	0.01V		Threshold Value	3 μ A		Search Interval	0.01V		Threshold Value	3 μ A							
Method	Run Time	V_1 (V)	V_2 (V)	V_3 (V)	V_4 (V)	Method	Run Time	V_1 (V)	V_2 (V)	V_3 (V)	V_4 (V)	V_5 (V)	V_6 (V)	V_7 (V)	V_8 (V)	V_9 (V)	
BF	1.49s	0.93	1.09	0.86	0.19	BF	>1 hour	-	-	-	-	-	-	-	-	-	
DC	0.39s	0.93	1.09	0.85	0.19	DC	>1 hour	-	-	-	-	-	-	-	-	-	
QDP	0.038s	0.93	1.09	0.85	0.19	QDP	0.13s	0.014	0.31	0.33	0.63	0.27	0.22	0.23	0.87	1.02	
HB	0.023s	0.93	1.09	0.85	0.19	HB	0.034s	0.014	0.3	0.32	0.62	0.26	0.22	0.23	0.88	1.01	
DC Operating Point Computation Analysis (Random Size Perturbation)						DC Operating Point Computation Analysis (Random Size Perturbation)											
RMSE with Unsegmented I_{DS} Model (μ A)						RMSE with Unsegmented I_{DS} Model (μ A)											
$I_{DS,1}$	$I_{DS,3}$	$I_{DS,4}$	$I_{DS,6}$	$I_{DS,8}$	$I_{DS,10}$	$I_{DS,2}$	$I_{DS,4}$	$I_{DS,6}$	$I_{DS,8}$	$I_{DS,13}$	$I_{DS,14}$	$I_{DS,15}$	$I_{DS,16}$	$I_{DS,18}$	$I_{DS,20}$	$I_{DS,21}$	$I_{DS,22}$
0.31	0.29	0.14	0.21	0.39	0.11	2.65	1.88	2.2	0.93	2.33	2.36	1.74	2.38	2.05	1.58	1.7	1.38
RMSE with Segmented I_{DS} Model (μ A)						RMSE with Segmented I_{DS} Model (μ A)											
$I_{DS,1}$	$I_{DS,3}$	$I_{DS,4}$	$I_{DS,6}$	$I_{DS,8}$	$I_{DS,10}$	$I_{DS,2}$	$I_{DS,4}$	$I_{DS,6}$	$I_{DS,8}$	$I_{DS,13}$	$I_{DS,14}$	$I_{DS,15}$	$I_{DS,16}$	$I_{DS,18}$	$I_{DS,20}$	$I_{DS,21}$	$I_{DS,22}$
0.29	0.23	0.076	0.13	0.11	0.11	1.71	1.46	1.01	0.96	2.6	1.9	2.32	1.75	2.06	1.35	1.44	1.08
Performance Evaluation (Properly Sized)						Performance Evaluation (Properly Sized)											
	A_V (dB)	PM ($^\circ$)	GM (dB)	UGB (MHz)				A_V (dB)	PM ($^\circ$)	GM (dB)	UGB (MHz)						
SPICE Simulation	58.8	78.19	51.46	24.31			SPICE Simulation	91.29	118.4	64.29	143.3						
Symbolic Analysis	59.66	81.75	51.56	22.91			Symbolic Analysis	90.79	115.5	61.8	154.7						
Relative Error	1.46%	4.55%	0.19%	5.76%			Relative Error	0.55%	2.45%	3.87%	7.96%						
Time Consumption Analysis						Time Consumption Analysis											
SPICE Simulation			1.4s			SPICE Simulation			1.5s								
Performance Model Building			0.0049s + 0.22s			Performance Model Building			0.0057s + 0.597s								
Performance Model Evaluation			0.039s			Performance Model Evaluation			0.046s								

hour, which is well out of our interest in the experiments. Compared with the SPICE simulation results, the variable computation results achieve high accuracy for both circuits. Among these four methods, the most efficient one is the hybrid-based method that could solve all the variables within 0.023 and 0.034 seconds for the folded-Cascode OpAmp and high-gain OpAmp, respectively. Thanks to its high efficiency, this method has been selected as our proposed DC operating point computation engine for the circuit performance evaluation process, whose runtime is also reported in Table 4.4. In our experiments, 0.01V and 3 μ A are set as the search interval value and threshold value, respectively. In practice, if this search interval value cannot ensure enough accuracy for solving out the DC operating point of some special circuits, smaller search interval values are suggested to utilize but at the cost of extended computation time.

We have also analyzed the impact of I_{DS} model on the accuracy of the computed DC operating points through the comparison of using the unsegmented I_{DS} model and the segmented I_{DS} model to compute the DC operating points. In the experiments, we randomly varied the circuit sizes and calculated the root mean square errors (RMSE) of each transistor's I_{DS} with reference to the SPICE simulation results, which are provided in Table 4.4. As shown, by utilizing the more accurate segmented I_{DS} model, better accuracy on the computation results of each transistor's I_{DS} was generally achieved, which demonstrates beneficial contribution from our proposed model segmentation scheme.

4.5.3. Analysis of Performance Modeling

The performances of the two properly sized circuits are evaluated through both SPICE simulation and symbolic analysis (GPDD). The results are listed in Table 4.4. As one can see, compared with the SPICE simulation, the accuracy of the symbolic analysis is high, achieving less than 5% relative error for DC gain (A_v), phase margin (PM), and gain margin (GM), and less than 8% relative error for unity gain bandwidth (UGB). In Table 4.4, we express the performance building time (i.e., the runtime of Algorithm 4.1) by two parts. The first part is the circuit preprocessing time while the second part is the GPDD structure construction time. As shown in the table, the circuit preprocessing times for both circuits are around 0.005 seconds, and the GPDD structure construction times for the folded-Cascode OpAmp and high-gain OpAmp are 0.22 and 0.597 seconds, respectively. The performance model evaluation times (i.e., the runtime of Algorithm 4.2) of the two circuits are 0.039 and 0.046 seconds, respectively. For the SPICE-based simulations, there is only evaluation time taken by circuit simulator Spectre. As reported in Table 4.4, compared with the SPICE simulation, the evaluation efficiency is improved by more than 30 times for our proposed method. As a matter of fact, due to the handling of unsolvable cases

proposed in Section 4.3.3, the evaluation runtime would be further shortened when the DC operating point of the circuit is improper. This is because if the circuit input is an inappropriate combination of circuit biases and transistor sizes, the evaluation process would be automatically terminated.

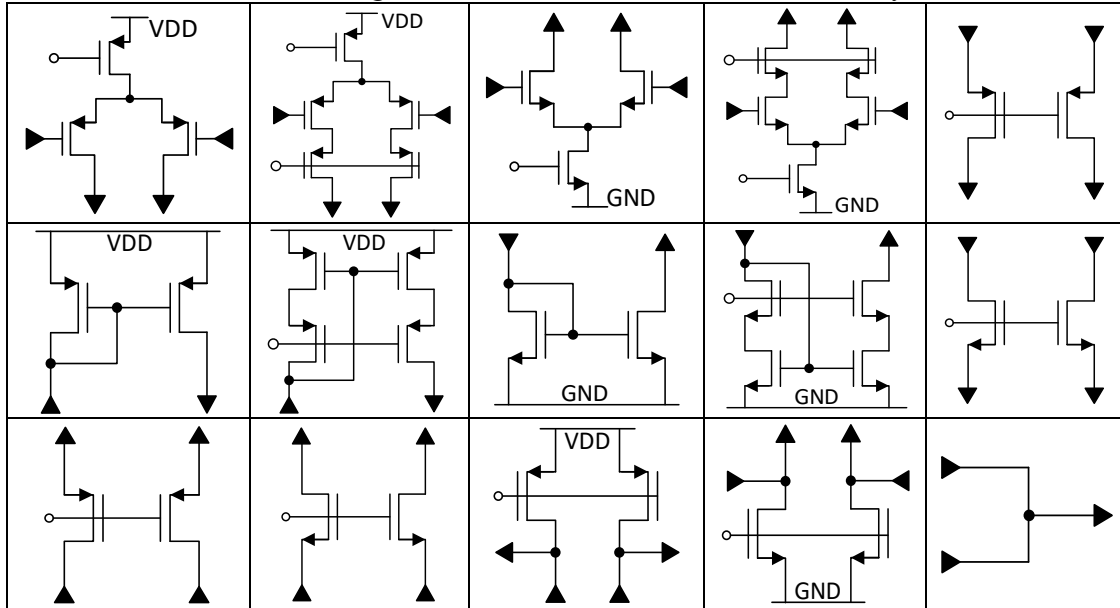
Table 4.5. Optimization Results of Two OpAmp Circuits

Circuit	Parameter	Spec.	Model		SPICE
			OPT. Result	Verification	OPT. Result
FCO	A_v (dB)	max	64.95	64.18	64.98
	PM ($^\circ$)	> 60	73.55	69.4	73.21
	GM (dB)	> 10	47.99	45.35	52.12
	UGB (MHz)	> 10	36.4	35.95	34.49
	<i>Runtime</i>		$\approx 17s$		$\approx 603s$
HGO	A_v (dB)	max	98.83	97.43	96.1
	PM ($^\circ$)	> 60	103.9	106.1	90.2
	GM (dB)	> 10	37.97	35.94	25.13
	UGB (MHz)	> 10	131.5	121.1	148.5
	<i>Runtime</i>		$\approx 40s$		$\approx 1285s$

In order to further demonstrate our proposed performance model can be robustly applied to the circuit sizing work, we employed the well-known multi-objective genetic algorithm NSGA-II to optimize two example circuits in our experiments [86]. The population size was set to 20, while the number of generations was set to 20 for the folded-Cascode OpAmp (FCO) and 40 for the high-gain OpAmp (HGO), respectively. The sizing optimization results are given in Table 4.5. For each circuit, both our proposed performance model and SPICE were used to size the circuit for comparison. The gain (i.e., A_v) is used as the design objective, while the others (i.e., phase margin (PM), gain margin (GM), and unity gain bandwidth (UGB)) are treated as design constraints. The final sizing results obtained from our proposed performance model were later validated by using SPICE simulation as shown in the fifth column of Table 4.5. As shown in the table, the

performances of the final sizing results from our proposed model are equivalent to those obtained by using SPICE simulation, but the efficiency was boosted by over 30 times.

Table 4.6. Building Blocks Used to Construct Circuits in Synthesis



To demonstrate the unique feature of almost ignorable model building overhead for our proposed performance modeling method, we have utilized it to aid in synthesizing circuit topologies by following the procedure presented in [J1], which involves a large number of topology variations. With a simplified building block library illustrated in Table 4.6 plus the specification and constraint listed in Table 4.7, 64 unique circuit topologies were generated. Among them, 32 topologies passed the fast evaluation filtering test, which is an automated stage consisting of linear constraint test, DC gain test, and fast sizing test [J1]. But they still need detailed sizing for further feasibility checking. In this regard, NSGA-II was utilized to optimize (i.e., size and verify) these 32 candidate topologies. Both the population size and the number of generations were set to be 20, which means 400 evaluations in total for each topology. As shown in Table 4.7, by using our proposed model to evaluate circuit performance, we could obtain exactly the same

Table 4.7. Results of Circuit Topology Synthesis

CMOS 65nm Technology					
Performance Specification					
A_V (dB)	PM ($^\circ$)	GM (dB)	UGB (MHz)		
60	60	10	10		
Constraint			#Max Leaves = 5		
#Topologies Generated	#Topologies Left after Fast Evaluation	#Topologies Left after Optimization		Total Optimization Time	
		SPICE	Our Model	SPICE	Our Model
64	32	8	8	≈ 320 mins	≈ 10 mins

synthesis result as that by using SPICE simulations (i.e., the same 8 topologies satisfying the target specifications and constraints stood out through the optimization), while our proposed method could speed up the optimization process by more than 30 times.

4.5.4. Comparison with the State-of-the-Art Methods

So far we have shown high efficiency, acceptable accuracy, strong robustness, and efficient model development of our proposed performance modeling method. In this sub-section, we will exhibit some comparisons with other state-of-the-art performance modeling methods. Among those methods, CAFFEINE [79] aims to build symbolic performance models via canonical-form functions and genetic programming while the works of [75], [74], [77], and [87] focus on building support vector machine (SVM), circuit matrix (CM), sparse regression (SR), and coupling sparse regression (CSR) models to fit circuit performances, respectively. As mentioned in Section 4.1, since those performance models directly take a large number of circuit simulation samples as training data, the time-consuming simulation-based sampling process has to be re-performed when modeling any different circuits.

Table 4.8. Comparison of Different Performance Modeling Methods for the Two-Stage OpAmp

Methods	A_V	GM	PM	UGB	# SPICE	Circuit Modeling Time	Eval. Time
	Average Error (%)						
<i>CAFFEINE</i>	2.95	–	–	–	129	20mins	–
<i>SR</i>	2.5 ~ 3	–	3.5 ~ 4	–	400	–	–
<i>CM</i>	0.03	0.09	0.05	–	2000	3.7mins	0.033s
<i>Our Method</i>	2.73	4.02	7.47	9.63	0	0.054s	0.007s
	RMSE						
<i>SVM</i>	0.047	–	0.003	–	1300	–	0.61ms
<i>CSR</i>	0.0369	–	0.015	0.015	100000	–	5s
<i>Our Method</i>	1.26	0.62	8.3	5.41	0	0.054s	0.007s

However, different from those performance modeling works, our proposed method computes circuit performances based on the regression of transistor models, which is just a one-time job for a specific technology. Therefore, as listed in Table 4.8, the number of SPICE simulations needed to obtain training data to train the performance model for our proposed method is zero whereas those are at least hundreds for the other five methods. The Circuit Modeling Time reported in Table 4.8 is the time consumed to build or train the model for the entire circuit, which excludes the time consumption for the collection of training data. As one can see, the model building time is only 0.054 seconds for our proposed method versus several digits in the unit of minutes for the other modeling methods. Although transistor modeling is needed for our proposed method, it is just a one-time job for a specific technology, which can always take place in advance of circuit synthesis and optimization. For the evaluation of modeling accuracy, CAFFEINE, SR, and CM report average errors while SVM and CSR give RMSE. For the comparison purpose, we randomly varied biases and transistor sizes in the two-stage OpAmp (as depicted in Fig. 4.5(b)) for 500 perturbations and accordingly derived the average error values compared with SPICE simulation results. As listed in Table 4.8 for DC gain (A_V), our proposed method offers better accuracy than

CAFFEINE, similar accuracy to SR, but worse accuracy than CM, SVM, and CSR. For phase margin (PM), our proposed method was 4% worse than SR and 7%-8% worse than CM and SVM. Nevertheless, it can preferably achieve an evaluation runtime of 0.007 seconds and less than 10% average performance evaluation error, which indicates its superior fit for the analog circuit synthesis and optimization tasks.

4.6. Summary

In this chapter, we presented a novel performance modeling method that can not only achieve high efficiency, acceptable accuracy, and strong robustness, but also feature a unique metric of almost ignorable model building overhead. It is comprised of advanced neural-network-based transistor modeling, circuit preprocessing through undirected bipartite graph, deterministic circuit DC operating point computation, and symbolic analysis. Compared with other performance modeling methods, it moderately sacrifices the accuracy but gains significant generality. This is commonly preferable since the main goal of using performance models is to save simulation time with a trade-off of reasonable accuracy loss, especially within the context of circuit structural synthesis and optimization.

As mentioned in Section 2.2.3, there are two main streams to reduce the computation effort of the work of automated analog circuit topology synthesis. In the next chapter, we will propose a novel method from another stream, which utilizes the merits of machine learning, to significantly shrink the computation effort of the whole synthesis process.

Chapter 5 Deep-Reinforcement-Learning-Based Topology Synthesis for Analog Integrated Circuits

5.1. Introduction

Since the genetic operations are typically performed randomly without clear direction, the evolution-algorithm (EA) based methods often suffer from substantial computation effort wasted on generating and evaluating meaningless circuit topologies. Furthermore, there is no means to flexibly extend its search space once the rules of genetic operators have been designed, leading to only a narrow range of circuits that can be synthesized. Compared with the EA-based generation methods, the design search space of the graph-based generation methods can be flexibly extended (even to infinitely large) depending on the allowed graph size (a user-defined parameter) and complexity of the predefined BB library. On one hand, this feature is beneficial for generating a wider range of circuits, including some novel circuits. On the other hand, it would cause a severe problem, that is, topology explosion (i.e., a huge number of topologies generated in the synthesis process), because the graph-based approaches would explore all the possibilities in a brute-force manner within their allowed search space. Checking the feasibility of such a huge number of circuit topologies would be an extremely time-consuming computation task in practice.

To address the shortcomings of the existing topology synthesis approaches, in this chapter we propose a novel deep-reinforcement-learning-based circuit topology synthesis methodology. It not only integrates the merits together of both topology generation methods and topology refinement methods, but also is capable of synthesizing large-size and innovative circuit topologies. Deep reinforcement learning (DRL) is a machine learning technique known to be able to solve complex

tasks. It has been successfully applied in some EDA areas to facilitate circuit design, such as analog circuit sizing [88] [89] and analog layout placement [90]. Similar to the topology refinement methods, our proposed method starts with an initial sub-circuit and gradually expands topologies with BBs via a decision-making process. But this process is realized by DRL with the produced topologies and expanded BBs, which are encoded into states and actions, respectively. Besides using BBs as the basic components, the composition and decomposition rules employed in the graph-based topology generation methods are effectively defined in our proposed DRL-based method as rule-based actions, which function in a similar way to reasonably construct circuit topologies in the synthesis process.

The performance of a circuit not only depends on its topology (i.e., circuit structure), but also relies on its device sizes. In our proposed method, we justify the feasibility of a produced circuit topology by checking whether it is able to meet a certain performance specification through a simulation-in-loop sizing operation rather than seeking for an optimum performance. As long as the performance of a produced topology meets the specification requirement, the DRL agent will receive a positive reward; otherwise, it will get a negative reward. Through successive trials, the DRL agent is continuously learning from the rewards and improving its means to reach a solution (i.e., a feasible topology) by figuring out which BBs to expand is the best choice at each construction step. Moreover, hash table and symbolic analysis [85] are employed in our work to reduce the number of produced circuit topologies to be sized during the synthesis process, which would significantly boost the whole synthesis efficiency. In addition, the parallel computing technique is adopted to speed up the time-consuming circuit sizing process.

To the best of our knowledge, this is the first work that employs the reinforcement learning technique to automate the topology synthesis of analog integrated circuits. It has the following remarkable features:

- 1) The synthesis process is a decision-making process;
- 2) Design search space is flexible and controllable;
- 3) Design knowledge is automatically learned;
- 4) Considerably less computation effort compared with the conventional topology generation methods;
- 5) Output of trustworthy solutions with detailed device sizes.

The research work conducted on this topic has been submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [J3] (with major revision now) and accepted by 2022 ACM/IEEE Design, Automation & Test in Europe Conference (DATE) [C7].

5.2. The Proposed Framework

5.2.1. Deep Reinforcement Learning

Reinforcement learning (RL) is built upon an agent that iterates to function in an environment using a trial-and-error process that mimics learning in humans, where agent is defined as an imagined learner (i.e., the algorithm itself) and environment is deemed as the world where the agent operates. There are three basic elements of RL, which are *state*, *action*, and *reward*. All of them are associated with the RL environment, which is based on the problem to be solved. Specifically,

state is the observation of the status for the environment, *action* is the operation that the agent performs in the environment, and *reward* is the feedback that the agent receives from the environment after making an action.

DRL combines RL and deep learning (DL), as reflected by the fact that the RL agent is connected with a neural network (NN). At each time step, the RL agent observes the current state of the environment, consults the NN by inputting the current state, infers the action to be taken based on the output of NN, and takes the selected action. The environment then returns a new state that is used to calculate the reward as a result of taking that particular action. In the meantime, the current state, selected action, and resultant reward are collected as training data to train the neural network through DL, which will help the RL agent make better decisions subsequently. This continuous RL loop would eventually help the agent to learn to act in a way (i.e., finding a sequence of actions) that will maximize its expected cumulative reward, which is the objective of RL. The expected cumulative reward at time step t can be written as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad (5.1)$$

where r is the reward received at each time step; and $\gamma \in (0, 1]$ is called the *discount factor*, which means future rewards are worth less than immediate rewards due to the common sense that rewards coming sooner are more likely to take place.

There are several schemes that have been proposed to carry out the DRL process, such as Q-learning, policy gradient, and advantage actor-critic [91]. The circuit topology synthesis process, which we have to deal with in this work, features the following special situation and in turn the uncommon DRL environment: any circuit topology can only be evaluated after a valid structure, which satisfies the circuit input-output specification, has been constructed. Compared with either

Q-learning or advantage actor-critic method, the policy gradient method shows its best fit to our work because this method can wait to calculate the reward until the end of an *episode*, which is defined as a sequence of states, actions, and rewards from an initial state to a termination state.

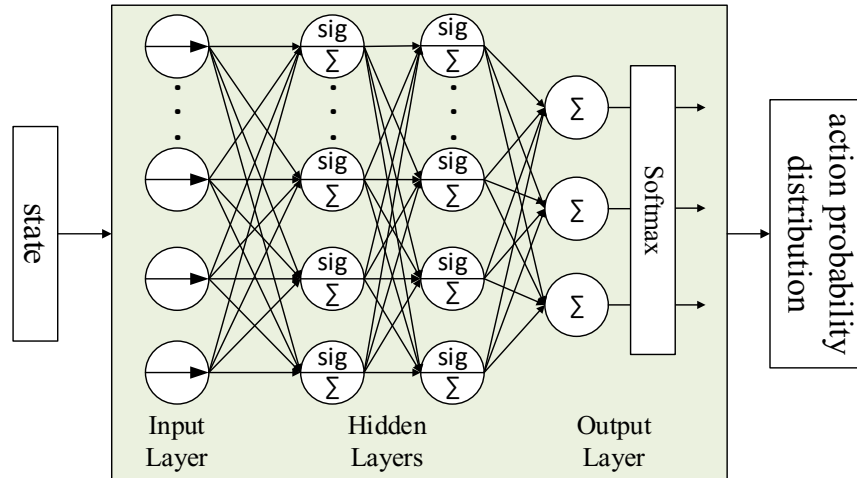


Fig. 5.1. Policy gradient neural network.

The policy-gradient-based method attempts to directly find a policy that can form a circuit topology meeting the target specifications. Here *policy* is defined as the strategy that an agent employs to determine the next action based on the current state. In the policy-gradient-based DRL, the policy (π) is modeled as a policy gradient neural network (PGNN) with parameterized weights θ . As depicted in Fig. 5.1, the PGNN is actually a supervised classification model, which takes a state (s) as input and outputs the probability distribution over all actions (a):

$$\pi_{\theta}(a|s) = P(a|s). \quad (5.2)$$

The architecture of PGNN is a fully connected artificial neural network (ANN), which is comprised of a passthrough input layer, several hidden layers, and an output layer. The activation functions applied in the hidden and output layers are the widely used sigmoid function and softmax function, respectively.

5.2.2. DRL Framework for Circuit Topology Synthesis

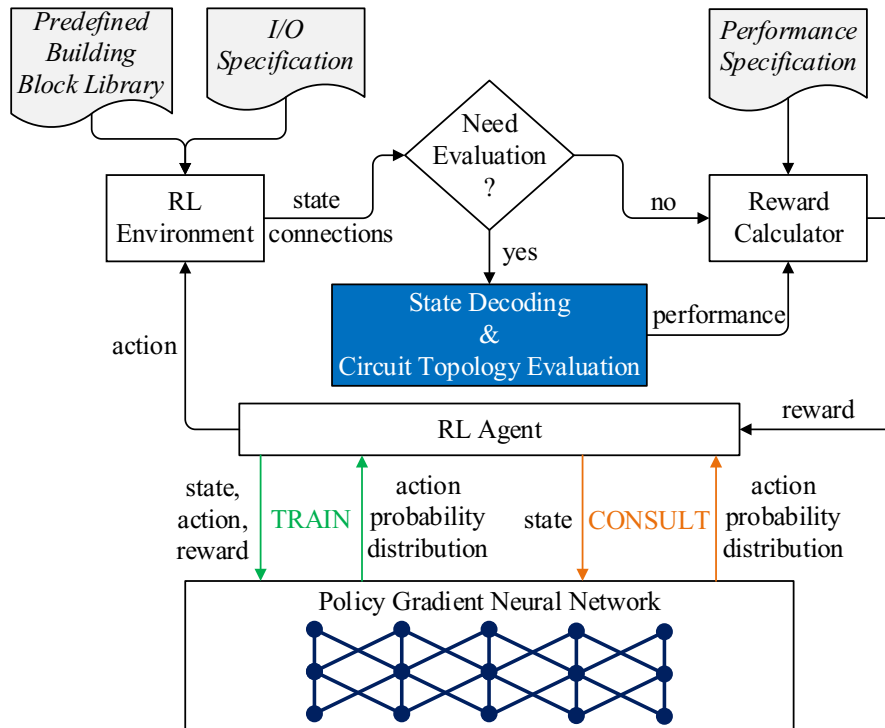


Fig. 5.2. The proposed DRL framework for circuit topology synthesis.

As depicted in Fig. 5.2, in our work the automated circuit topology synthesis is realized through the proposed DRL framework. Building blocks are utilized as the basic components, which are selected from a predefined building block library (PBBL) by taking actions, to construct circuit topologies. The design specifications are composed of input-output specification and performance specification. Among them, the input-output specification works with the PBBL to form the specialized RL environment for circuit topology synthesis, while the performance specification is utilized to calculate the reward for the RL agent. It is worth noticing that only the states that satisfy certain conditions, which will be explained in more detail in Section 5.4.2, need to be decoded into circuit topologies for evaluation. For the states that do not meet the conditions, a reward will be directly assigned according to our reward function as defined in Section 5.3.4.

Unlike the consulting of PGNN that occurs at all the time steps, the training of PGNN is carried out only at the end of an episode. Thereby, the training data contains the collected states, actions, and rewards of one or multiple episodes. The gradient ascent method is employed to iteratively find the best weights that improve the policy in the direction of maximizing the expected future reward with the following update rule:

$$\nabla\theta = \alpha \left(\sum_{t=0}^{T-1} \log\pi_{\theta}(a_t|s_t) \right) \left(\sum_{t'=t+1}^T \gamma^{t'-t-1} r_{t'} \right), \quad (5.3)$$

where α is the learning rate, and T is the total time steps of an episode. Through continuous training along the RL loop, the agent gradually learns the essential synthesis knowledge, which is stored in the PGNN. This episodic training process terminates when the gradient (i.e., Eq. (5.3)) equals zero, which means the weights of PGNN are not able to be further optimized.

As illustrated in Fig. 5.2, the consulting and training of PGNN are carried out simultaneously during the whole learning process. Specifically, the consulting of PGNN will get feedback from the environment via rewards, which would be used to train the PGNN in return to let it learn from the trial results and be able to make better decisions subsequently based on the learned knowledge. After the PGNN has been well trained, the RL agent can always derive a solution in its first episode trial.

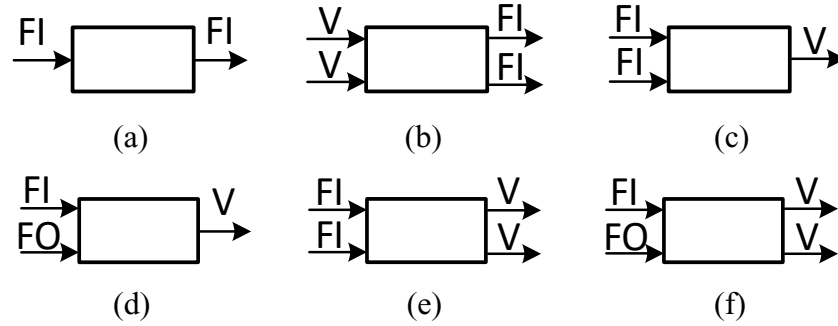


Fig. 5.3. Examples of the six types of BBs: (a) One-signal-path BB. (b) Two-signal-paths BB. (c) Identical-current converter with one output. (d) Distinct-current converter with one output. (e) Identical-current converter with two outputs. (f) Distinct-current converter with two outputs.

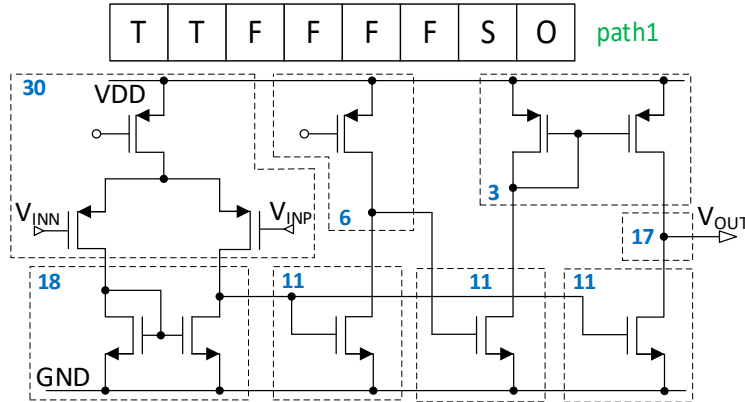
5.3. Specialized RL Environment

5.3.1. Building Block (BB)

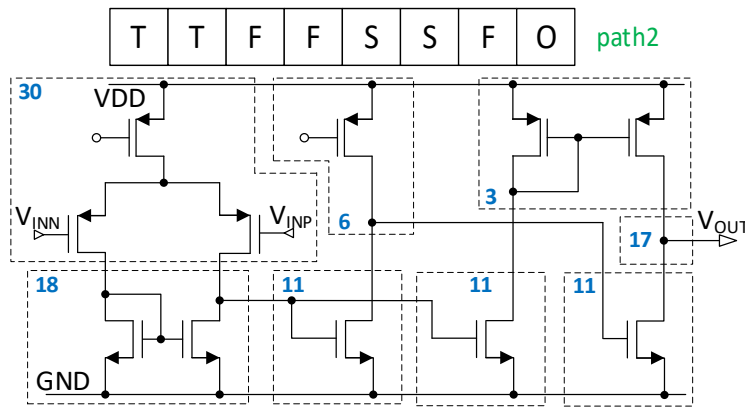
As mentioned in Sections 5.1 and 5.2, our proposed work utilizes BBs as the basic components to construct circuit topologies. The input and output terminals of a building block can be categorized into three types: V , FI , and FO . Among them, V means the corresponding input or output terminal is a voltage terminal, while FI or FO represents a current terminal with bias current flowing into or out of a BB, respectively. Based on the counts and types of input and output terminals, we have defined six types of BBs. The names of these types are self-explained. Fig. 5.3 illustrates an example of each type of BB, where the input and output terminals always lie on the left side and right side of a block, respectively. Generally, each input or output terminal can be either V , FI , or FO . But for all the converter-type BBs, the input terminals must be current while the output terminal(s) must be voltage. In addition, the two-signal-paths BBs must have the same type of input terminals as well as the same type of output terminals.

← Max Number of BB Allowed →									
30	18	11	6	11	3	11	17	0	0

(a) state A



(b₁) The circuit topology represented by state A and path1



(b₂) The circuit topology represented by state A and path2.

Fig. 5.4. An example state and its possible circuit topologies represented.

5.3.2. State

In our proposed DRL-based framework, states are represented by the data structure of array. The value at each slot in the array refers to the index of a BB in the PBBL if it is not equal to zero. Otherwise, it means no BB (i.e., empty) available in this slot. In this way, each state encodes a sequence of BBs, making up a circuit topology. An example state (state A) is given in Fig. 5.4(a).

The size of a state is determined by the maximum number of BB allowed (a user-defined parameter) for constructing a circuit topology. This constraint is necessary to practically limit the design search space. According to our experiments, this useful constraint works well with the extendable PBBL, contributing to flexible and controllable design search space in our proposed circuit topology synthesis methodology.

However, if the detailed connections among the represented BBs are unclear, one state may be able to be decoded into several circuit topologies. To address this issue, we define an attribute called *path*, which is also an array, to track the signal path which each BB in the slot belongs to. There are four types of *path* in total: *F*, *S*, *T*, and *O*, which means that the added BB belongs to the first signal path, the second signal path, both signal paths, and only one signal path, respectively. During the synthesis process, the path information is recorded in order to assist the decoding of a state because the combination of a pair of state and path would lead to a determined circuit topology correspondence. Fig. 5.4(b₁) and Fig. 5.4(b₂) illustrate two possible paths and their represented circuit topologies when being combined with state A.

5.3.3. Action

In our proposed framework, taking an action means selecting a BB from the PBBL to connect with the output terminal(s) of the current-state-represented circuit structure. Although the DRL has the ability to automatically learn the optimal selection strategy through its continuous trial-and-error process, making the DRL process follow some basic design rules would largely avoid constructing meaningless circuit structures, contributing to significant speed-up of the learning process. These basic design rules force the connected terminals to be matched. Specifically, *V* must be matched with *V*, *FI* with *FO*, and vice versa. Based on these rules, the actions can be divided

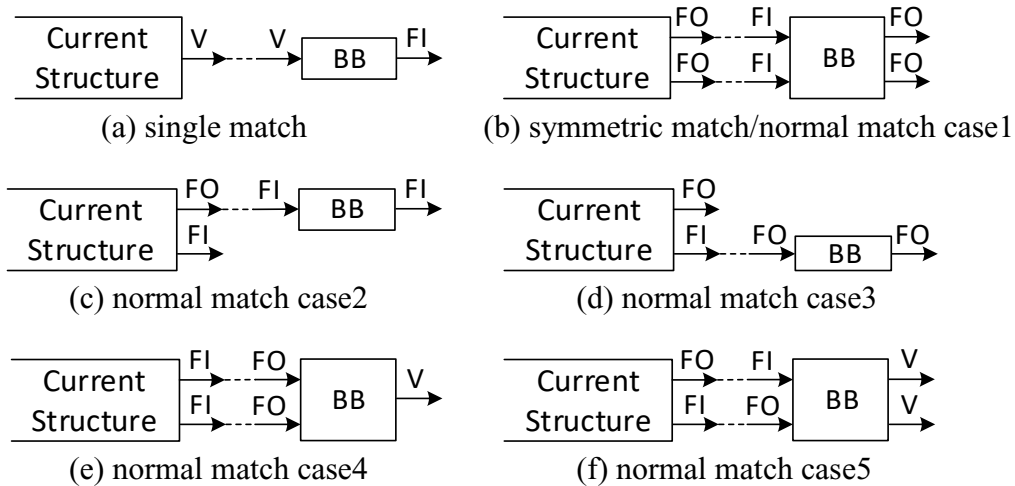


Fig. 5.5. Examples of the matched actions.

into three types: un-matched action, matched action, and terminational action. In this regard, if any input terminal of the BB selected by an action cannot match with the output terminal that it is going to connect, this action is called an unmatched action. Otherwise, it is named as a matched action. The terminational action is a special type of action, which is only allowed to happen when the output terminal(s) of the current structure has (have) the same counts and types as the output specification required. Executing a terminational action would not select any BB to connect with the current structure, but cause the termination of an episode.

Based on the types of BBs selected by actions, the matched actions can be categorized into single match, symmetric match, and normal match. Besides the *path* information, knowing the type of a match would greatly facilitate the process of figuring out the detailed connections between the current structure and the added BB. An example of each type of matched action is illustrated in Fig. 5.5. Their formal descriptions are listed below:

- *Single match*: connect a one-signal-path BB to a one-output structure.

- *Symmetric match*: connect a two-signal-paths BB to a two-identical-outputs structure that is symmetric.
- *Normal match case1*: connect a two-signal-paths BB to a two-identical-outputs structure that is asymmetric.
- *Normal match case2*: connect a one-signal-path BB to the first output terminal of a two-outputs structure.
- *Normal match case3*: connect a one-signal-path BB to the second output terminal of a two-outputs structure.
- *Normal match case4*: connect an identical-current converter to a two-identical-outputs structure.
- *Normal match case5*: connect a distinct-current converter to a two-distinct-outputs structure.

In order to further reduce the chance of generating senseless circuit topologies, it is essential to respect necessary structural symmetry constraints in circuit design, such as the first stage of OpAmps (due to the preference of differential pairs) [62]. In the proposed work, we keep tracking the symmetry property of the structure being constructed. At the beginning of an episode, symmetric-match actions are allowed. However, once a normal-match action is taken, which would break the symmetry property of the structure being constructed, the symmetry-match actions are no longer allowed to perform within this episode. Thereby, although the normal match case1 shared the same connection way as the symmetric match, they are different since the normal match case1 is still allowed to take when the symmetry property has already been broken.

5.3.4. Episode and Reward

As mentioned in Section 5.2, the policy-gradient-based RL is carried out episodically. Within one learning task, all the episodes start with the same initial state. This initial state could be an encoded BB or a sub-circuit as long as its input terminal (or terminals) meets (or meet) the input specification. At each time step, the RL agent selects an action to take. After executing the action, the state transforms into a new state, which has one more slot becoming non-zero. This process continues until entering a termination state, which indicates the ending of an episode. In our work, we have defined a uniform format for the termination state, in which all the slots have the same value of 99. An episode would go to the termination state when any of the following three cases has occurred: 1) an unmatched action is taken; 2) state boundary is exceeded; 3) a terminational action is taken.

Fig. 5.6(a) and Fig. 5.6(b) depict two example episodes and their encoded circuit topology construction processes, respectively. The two episodes both start with the 30th BB in PBBL. After taking actions a_1 and a_2 , the 24th and 16th BBs in the PBBL are added in sequence, as shown by one more slot becoming non-zero in state at each time step. Now, the output of the structure being constructed becomes one voltage. Assuming it meets the output specification, then the terminational-type actions are allowed to take at this point. If an unmatched action or a terminational action (e.g., action a_3 in red on the lower branch in Fig. 5.6(a)) is taken, the state turns into the termination state and this episode is completed. Otherwise, a matched action (e.g., action a_3 in green on the upper branch) is made and the episode continues. After taking this matched action, the 11th BB is added into the state, which reaches the state boundary (i.e., all the slots in the state are filled with non-zeros). Then, taking any type of further action (e.g., action a_4

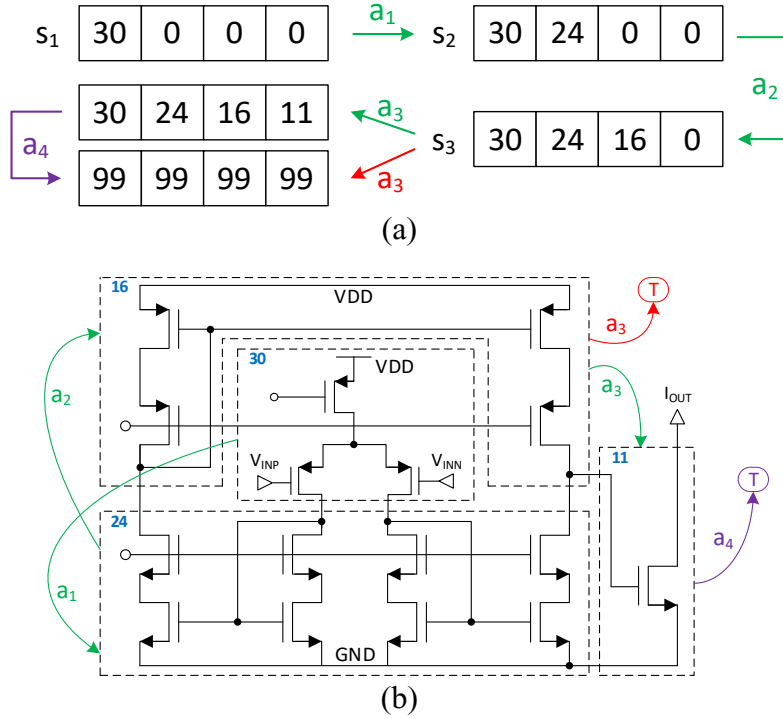


Fig. 5.6. (a) Two example episodes. (b) Their corresponding circuit topology construction processes.

in purple) would let the episode end with the termination state due to the exceeding of the state boundary.

Assume t is the time step within an episode, and this episode terminates at the $T1$, $T2$, or $T3$ step due to occurrence of case 1), 2), or 3), respectively. The reward function $r(t)$ is accordingly defined as follows:

$$r(t) = \begin{cases} 0, & \text{if } t < \min(T1, T2, T3) \\ -5, & \text{elif } t = T1 \\ -3, & \text{elif } t = T2 \\ 1, \text{ if meeting spec.} \\ -1, \text{ otherwise} \end{cases} \quad \text{elif } t = T3 \quad (5.4)$$

As the reward function indicates, all the non-termination steps would receive a reward of 0, while for the termination step, a positive or negative reward is assigned to encourage or discourage the generation of the corresponding circuit topology, respectively. For the termination that occurs

at $T3$, the constructed circuit topology has to be evaluated. If its performance meets the target specification, +1 is assigned. Otherwise, -1 is assigned.

However, like the example shown in Fig. 5.6, even though taking actions a_1 and a_2 is in the correct direction to form a circuit topology that meets the target specifications, if action a_3 is an unmatched action, -1 will be received to make the agent feel that all the selected actions in this episode are generally in the wrong direction. Learning exactly which action is good or bad is quite slow, which may require a huge number of iterations to figure out. In order to speed up such a learning process, we need to tell the agent that actions a_1 and a_2 are actually good. Therefore, when a termination occurs at $T3$ and the evaluated performance meets the target specification, we go back to modify the reward received from taking the second last action (e.g., a_2 in Fig. 5.6 for the case with a_3 in red leading to the termination state) to be +5 after completing this episode.

5.4. Circuit Topology Formation & Evaluation

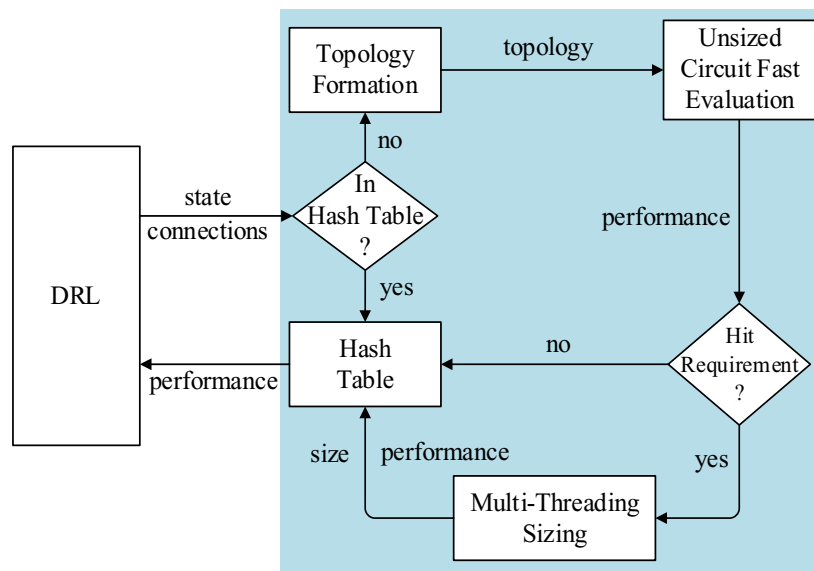


Fig. 5.7. The proposed framework with detailed circuit topology evaluation process.

During the RL process, when the produced states need to be evaluated, the simulation-in-loop sizing is applied in our work to verify their decoded circuit topologies. This is a time-consuming operation per se. In order to boost the evaluation efficiency, the hash table, symbolic analysis, and parallel computing techniques are employed in our work. The evaluation process is illustrated in Fig. 5.7, which is the detailed implementation of the shaded block in Fig. 5.2. Specifically, the state to be evaluated and its associated connections are fed to the hash table first to check its existence. If they already exist, their performance attributes are fetched and returned to the reward calculator. Otherwise, they need to be first decoded into a circuit topology and then evaluated by the proposed fast evaluation filter. Only the survived ones will go to the subsequent sizing phase to check the performance feasibility. Finally, the evaluated results of both fast evaluation and sizing are stored in the hash table.

5.4.1. Hash Table

The hash table guarantees that one circuit topology only needs to be evaluated once. In this way, the total evaluation time for the whole learning process is significantly reduced. However, one circuit topology may be represented by many combinations of state and connections if both normal-match-case2 and normal-match-case3 have taken place during the construction process. For instance, the circuit topology depicted in Fig. 5.8(b) can be represented by either the combination of state A and path1 or the combination of state B and path2 illustrated in Fig. 5.8(a). The difference between them is the occurrence order of the normal-match-case2 and normal-match-case3, which indicates that the BBs are added to the first signal path (F -path) and the second signal path (S -path) at distinct time steps. The following lemma is to answer whether their occurrence order will affect the produced circuit structure or not.

path	T	T	F	F	S	S	F	F	O	
state A	30	18	11	3	11	3	5	11	17	0
path	T	T	S	F	F	F	S	F	O	
state B	30	18	11	11	3	5	3	11	17	0
path	T	T	F	F	F	F	O	S	S	
ustate	30	18	11	3	5	11	17	11	3	0

(a)

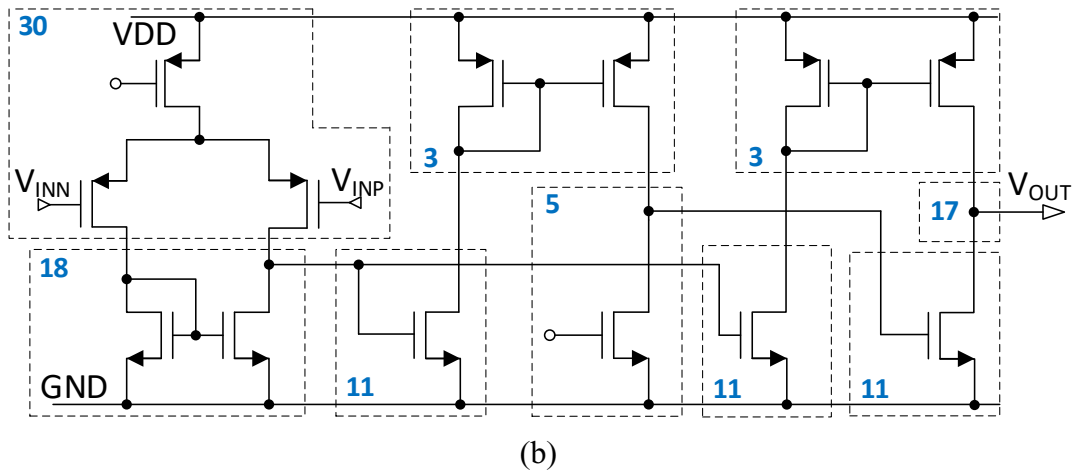


Fig. 5.8. Examples of unique state representations.

Lemma: If both normal-match-case2 and normal-match-case3 take place within a circuit construction process, the occurrence order between these two types of matches will not affect the constructed circuit structure as long as the orders among each type of match are maintained.

Proof: 1) It is a fact that at each time step, the order of adding an F -path-type or S -path-type BB to the current state will not affect the produced circuit structure if both of them have to take place. Thus, it can be inferred that adding an F -path-type (or S -path-type) BB can be postponed after adding one or multiple S -path-type (or F -path-type) BBs without affecting the produced circuit structure, as long as it is allowed at that time step. For example, in Fig. 5.8, the two S -path-type BBs are located at different slots in state A and state B but still represent the same circuit

structure. 2) If the occurrence order of the same path-type BBs is not preserved, the input and output terminals of the connected BBs may no longer be matched, leading to an invalid circuit structure. For instance, exchanging the two values in the fourth and sixth slots of state A (i.e., both 3) in Fig. 5.8 will not alter it but cause unmatched terminals among connected BBs, resulting in an invalid circuit structure. Based on the two analyses above, the lemma can be proved. ■

In order to save the size of the hash table, we need to convert the states due to those combinations, which are actually mapped to the same circuit topology, to a unique representation. That is to say, we need to set up a one-to-one correspondence between state representation and circuit topology. Here we will first define an operation called *state equivalent moving operation*: in a state, moving all the *S*-path-type BBs to the position after the last non-zero BB while still preserving their original relative occurrence order.

Theorem: The state equivalent moving operation is able to convert any states, which represent the same circuit structure, to a unique state representation.

Proof: The multiple-to-one correspondence between state representation and circuit topology is actually caused by the confusion due to multiple signal paths existing in the circuit topology construction. All the combinations that satisfy the Lemma described above can be viewed as equivalency in terms of the represented circuit structure. Thus, after such a moving operation, all the equivalent state representations should be converted into the same one, which is unique from others because the confusion source has been eliminated. Thus, the theorem must hold. ■

For example, as depicted in Fig. 5.8, after the BB equivalent moving operation, state A and state B are converted into the same representation (marked as *ustate*), which is unique. After converting a state to-be-evaluated to its unique representation *ustate*, its hash can be calculated via the following formula:

$$h(ustate) = \prod_{i=0}^n prime[j] \text{ mod } (2^{32} - 5), \quad (5.5)$$

where i is the index of a slot in $ustate$, n is the number of non-zero slots in $ustate$, j is the slot value (i.e., the index of the corresponding BB) in $ustate$, $prime$ is an array that stores all the prime numbers starting from 1 in the increasing order with the size being equal to the number of BBs in PBBL, and $2^{32} - 5$ is known to be the largest 32-bit unsigned prime number. Each circuit topology is put into a bucket according to its hash. For the circuit topologies that have only been fast evaluated, their DC gain results are stored in the hash table. For the ones that have been sized, their performance results and sizes are stored in the hash table.

5.4.2. State Decoding

As mentioned in Section 5.3.2, only when a terminational action is taken, the second last state (e.g., s_3 on the lower branch in Fig. 5.6(a)) rather than the termination state needs to be decoded and evaluated. The states that satisfy this condition only occupy a very small portion of the total states generated in the synthesis process, which is one of the main reasons that our proposed DRL-based topology synthesis method features good efficiency.

The state to be decoded is firstly mapped to its represented BBs. Then transistor-level circuit topologies will be formed by connecting those BBs according to the recorded connection information. In practice, simply swapping input pins of an analog circuit may affect its performance. Due to this fact, we treat the circuits, which have exactly the same structure but swapped input pins, as different circuit topologies. Therefore, the decoded circuit topology should create a copy of itself with swapped input pins.

5.4.3. Unsized-Circuit Fast Evaluation

As shown in Fig. 5.7, the decoded circuit topologies will first be fast evaluated, which can roughly assess quality of the topologies through symbolic analysis. In our proposed work, the GPDD algorithm [64] is employed to numerically calculate the DC gain of an unsized circuit, which requires the values of all the parameters in the small-signal model of the circuit as input. However, since the circuit to be evaluated in our context is un-sized, it is impossible to get those values via SPICE simulation. What we know so far is that there exist value ranges for those small-signal model parameters in a certain technology. Therefore, after extracting such ranges for a specific technology, the center values of these ranges are employed to efficiently estimate the DC gain through the GPDD algorithm. Furthermore, in order to further speed up the operation, we employ a simplified small-signal model that only contains four parameters (g_m , g_{ds} , C_{gs} , and C_{gd}) for each MOSFET transistor in a circuit. Due to approximate calculation, we deliberately lower the requirement of this quality filter compared to the given performance specification, which ensures that only the circuit topologies with very bad performance will not proceed to the subsequent sizing phase. Our experiment results in Section 5.5 will show high effectiveness of this proposed fast evaluation filter.

5.4.4. Simulation-in-Loop Sizing

Once the circuit topologies pass the fast evaluation filter, they have to be sized to check their performance feasibility. In this work, we employ the well-known multi-objective genetic algorithm NSGA-II to size circuits, which utilizes the SPICE simulation to evaluate circuit performance. Since our purpose of sizing is to check the performance feasibility instead of optimizing the circuit, we have slightly modified the NSGA-II algorithm. Specifically, the sizes of population and

Algorithm 5.1: DRL-Based Circuit Topology Synthesis

Input: Input-output specification S_{IO} ; performance specification S_P ;
trustworthy building block library $PBBL$; initial state s_0 .

Output: Circuit topology with device sizes.

1. Build the specialized RL environment based on S_{IO} and $PBBL$;
 2. Build the PGNN, gradient function ∇_{θ} , and hash table T ;
 3. **While** (true)
 4. Make a batch {
 5. Start an episode with s_0 :
 6. choose action a ; perform action a ;
 7. receive reward r and new state s ;
 8. store s , a , and r into sets S , A , and R , respectively;
 9. **If** (episode terminates)
 10. calculate the discounted reward R_d ;
 11. store R_d into set G_t ;
 12. break;
 13. }
 14. Train the PGNN with S , A and G_t ; Calculate ∇_{θ} ;
 15. **If** ($\nabla_{\theta} == 0$) break;
 16. **End While**
 17. Starting an episode with s_0 , get last state s_f before termination;
 18. Decode s_f into circuit netlist C ;
 19. Extract the performance P and size S of s_f from T ;
 20. **Return** C , P , and S ;
-

generation are set to 20 and 50, respectively. During the algorithm running, if the evaluated performance of any individual meets the target performance specification, the algorithm terminates right away and returns the performances and device sizes to the hash table. Otherwise, after executing all 50 generations, the best performances achieved so far and the corresponding device sizes are stored in the hash table. At each time, there are two circuit structures with swapped input pins to be sized. To boost the sizing efficiency, we employ the parallel computation technique to size the circuits.

The whole DRL-based circuit topology synthesis process is illustrated in Algorithm 5.1. Specifically, Lines 1-2 set up the specialized RL environment for circuit topology synthesis and the PGNN for learning synthesis knowledge, Lines 4-13 collect the training data by interacting with the environment, and Line 14 utilizes the training data to train the PGNN. The collecting and training processes are repeated until the gradient function equals zero. After that, Lines 17-19 extract the solution, which includes circuit netlist, performance results, and corresponding sizes, from the trained DRL-based framework.

5.5. Experimental Results

In Section 5.5.1, our experimental parameter settings are explained. Sections 5.5.2 and 5.5.3 analyze the learning process and synthesis efficiency of deep reinforcement learning in the work of circuit topology synthesis, respectively. Section 5.5.4 reports the circuit topologies synthesized by the proposed method. Finally, Section 5.5.5 compares the proposed work with the state-of-the-art circuit topology synthesis tools.

The proposed circuit topology synthesis framework was mostly implemented in Python, with the fast evaluation (GPDD algorithm) realized in C++, the NSGA-II sizing carried out in C, and the SPICE simulations conducted by the Cadence tool. Our experiments were run on an Intel X86 1.2-GHz Linux workstation that has 64 GB of memory. All the experiments were conducted in a CMOS 65nm technology process, which can be readily replaced by any other CMOS technologies.

5.5.1. Experimental Parameter Settings

As shown in Table 5.1, our defined PBBL contains 36 BBs, each of which has a unique index. All our experiments were carried out based on this PBBL, which includes 12 one-signal-path BBs, 14 two-signal-path BBs, and 10 converter-type BBs. The one-signal-path BBs are composed of well-known current mirrors, Cascode current mirrors, current sources, Cascode stages, and common sources. Besides the differential pairs, the two-signal-paths BBs are basically made up of two copies of one-signal-path BBs. Among the converter-type BBs, there are two special BBs that only contain nets, while the others are the variants of the (Cascode) current mirrors.

Each type of BB could be composed of NMOS/PMOS transistors, leading to distinct terminal types. For different types of BBs, their input and output terminals are expressed in distinct formats as follow: 1) One-signal-path type BBs: *input – output*; 2) Two-signal-paths type BBs: *input1|input2 – output1|output2*; 3) Converter type BBs: *input1|input2 – output* or *input1|input2 – output1|output2*. It is worth mentioning that the two special nets-contained-only BBs (i.e., the 17th and 22nd BB) both have two possible combinations of input-output terminals, which are completely

Table 5.1. Predefined Building Block Library (PBBL)

One-Signal-Path Building Blocks				Converter Building Blocks			
(Cascode) Current Mirror		Current Source		One-Output Converter			
FI - FI	FO - FO	FI - V	FO - V	FI FI - V	FO FO - V	FO FI - V	
Cascode Stage (Down)		Cascode Stage (Up)		Common Source			
						Two-Outputs Converter	
FI - FO	FO - FI	V - FI	V - FO	FI FI - V V	FO FO - V V	FO FI - V V	
Two-Signal-Paths Building Blocks							
Two (Cascode) Current Mirror				Two Source-driven Current Splitter			
FI FI - FI FI	FO FO - FO FO	FI FI - FI FI	FO FO - FO FO				
Differential Pair		Two Cascode Stage (Down)		Two Cascode Stage (Up)		Two Common Source	
V V - FI FI	V V - FO FO	FI FI - FO FO	FO FO - FI FI	V V - FI FI	V V - FO FO		

dependent on the order of the input terminals. In addition, in order to facilitate the job of evaluating the produced circuit topologies, we require that except for the differential pairs (i.e., the 29th and 30th BBs in the PBBL), all the transistors within a BB have the same length and the same width.

In our experiments, the size of a state was set to be 10, which is the maximum number of BBs allowed to construct circuit topologies. The size of action was set to be 20, which equals the maximum number of possible choices for expanding a BB. For the PGNN, we defined two hidden layers, with the first and second layers containing 300 and 200 neurons, respectively. The learning rate, discount factor, and batch size were set to be 0.002, 0.95, and 200, respectively.

5.5.2. Analysis of the Learning Process

To synthesize OpAmp circuits, we set the starting sub-circuit as a differential pair made of PMOS (i.e., the 30th BB), the input-output specification as two voltage inputs and one voltage output, and the performance specification as 60dB DC gain (A_v), 60° phase margin (PM), 10dB gain margin (GM), and 10 MHz unity-gain bandwidth (UGB). In order to fairly illustrate the learning process of the RL agent in the proposed framework of circuit topology synthesis, we ran the algorithm six times and received six corresponding learning processes, which are depicted in Fig. 5.9.

As shown in the diagram, these six processes terminate at different iterations. Specifically, process2, process4, and process6 finished within 800 iterations, while process1, process3, and process5 required around 2000, 1300, and 1600 iterations, respectively. However, all these six learning processes share almost the same learning trend. Specifically, in the beginning, all the processes got the mean reward around -5, which means that their episodes within a batch were mainly terminated due to unmatched actions taken. Then, the mean of the received rewards in a

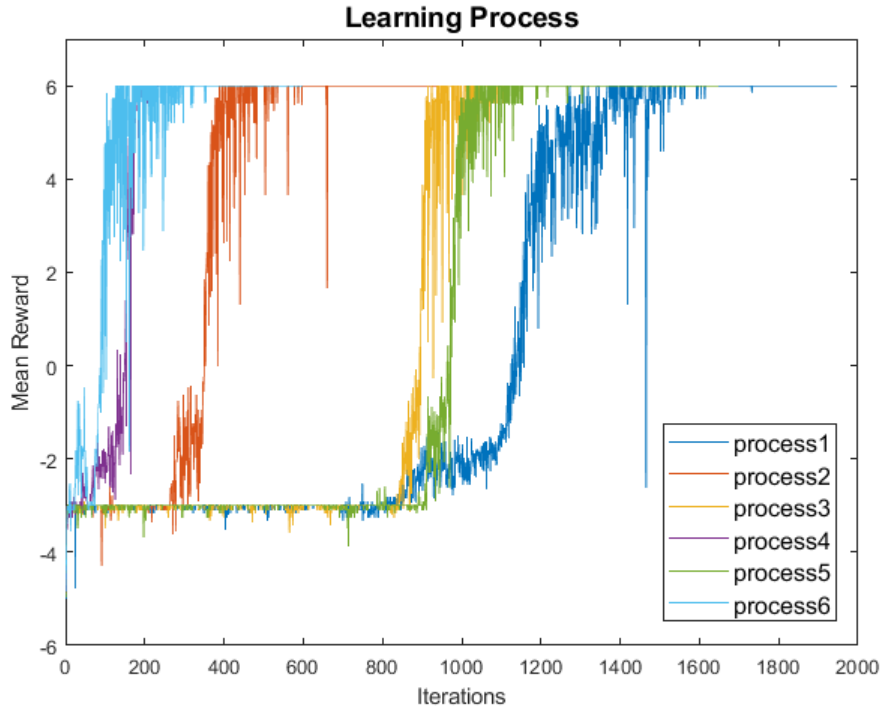


Fig. 5.9. The mean reward of a batch during the learning process.

batch became -3. It indicates that the RL agents figured out taking unmatched actions was bad and tried to avoid taking them, but they were still struggling to find a solution before exceeding the state boundary. After that, the received mean rewards became vibrating around -2, which means many episodes within a batch terminated because of taking terminational actions rather than only exceeding the state boundary. After a longer course of learning, all the RL agents gradually found a way to produce solutions, thus getting the rewards of +6 eventually.

5.5.3. Analysis of Synthesis Efficiency

Since the time-consuming sizing has to be performed on each generated circuit structure to check its feasibility, the most challenging part of circuit topology synthesis is the evaluation of the produced circuit structures no matter what synthesis strategies are applied. This is especially true for our proposed DRL-based method because training the PGNN needs a huge number of training

Table 5.2. Evaluation Details of the Exemplary Six Processes

	# Evaluations	# Fast Evaluation	# Sizing	Runtime
<i>process1</i>	28,469	174	86	9h:23min
<i>process2</i>	14,952	133	65	8h:22min
<i>process3</i>	10,553	86	38	4h:27min
<i>process4</i>	9,645	89	20	2h:38min
<i>process5</i>	26,053	73	35	3h:48min
<i>Process6</i>	12,685	73	36	3h:15min

Table 5.3. Testing Results of the Effectiveness of Fast Evaluation Filter

Filtering Requirement	# Fast Evaluation Failure	# Sizing Failure	Correct Rate
> 50 dB	146	101	69%
> 40 dB	138	117	85%
> 30 dB	128	127	99%

data. Table 5.2 depicts the evaluation details of the above-mentioned six processes. As one can see, almost at least ten thousand evaluations are performed for each process. If the simulation-in-loop sizing is directly applied as the means for evaluation, the synthesis time would be by far unaffordable.

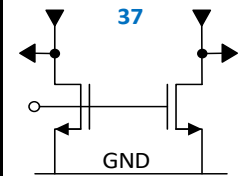
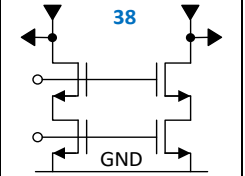
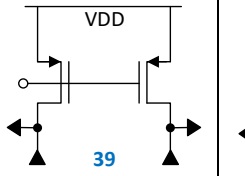
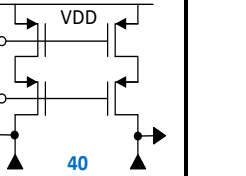
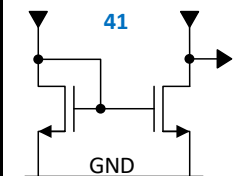
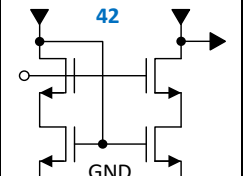
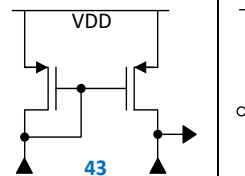
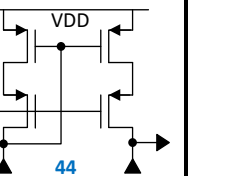
As demonstrated by the number of circuit topologies going to the fast evaluation stage in Table 5.2, after employing the hash table, the number of evaluations has dramatically decreased to just hundreds from at least near ten thousand, which means most of the evaluations were actually performed on the same structures. The reason for this is that a substantial number of episodes produced in the process were repeated, due to batch-size training and the stochastic policy strategy that always makes a decision based on the probability distribution over actions. Therefore, with the help of the hash table, evaluating the generated circuit topologies becomes realizable. However, we are still not clear about whether the circuits that fail the fast evaluation filter are all unable to meet the performance specification through sizing. To test it, we ran the algorithm ten times with

distinct filtering requirements. All the fast evaluation failed circuits were recorded for detailed sizing to check the effectiveness of our fast evaluation filter. The average results are depicted in Table 5.3. As one can see, when we set the filtering requirement as DC gain larger than 30 dB, 99% of the fast evaluation failed circuits would fail the detailed sizing check as well. Therefore, in our work, this value is set as the filtering threshold.

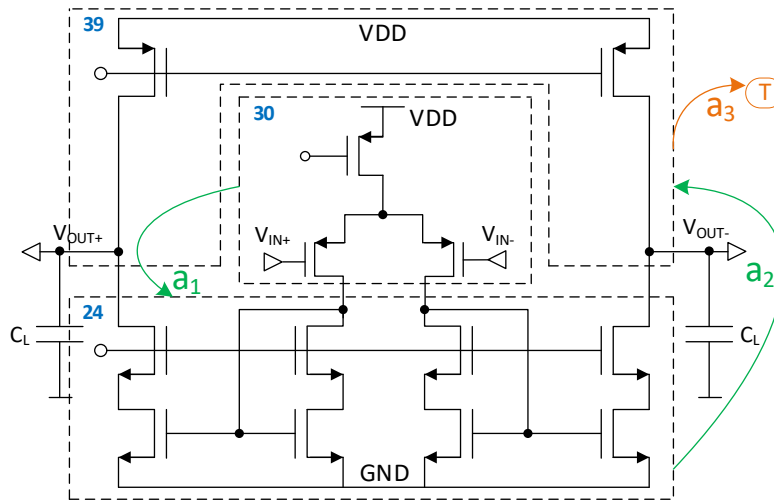
As reflected by the number of circuit topologies entering the sizing stage, the fast evaluation filter eliminated at least half of the topologies to be sized, which significantly improved the evaluation efficiency. The runtime of each learning process is provided in the last column of Table 5.2, which varies from 2 to 9 hours. It is worth noting that greater number of generated topologies to be sized does not necessarily mean more runtime. For instance, process6 took less runtime than process5 but with more topologies to be sized. This is because some sizing jobs of process5 need more time (i.e., generations) to complete.

5.5.4. Analysis of Circuit Topology Synthesis

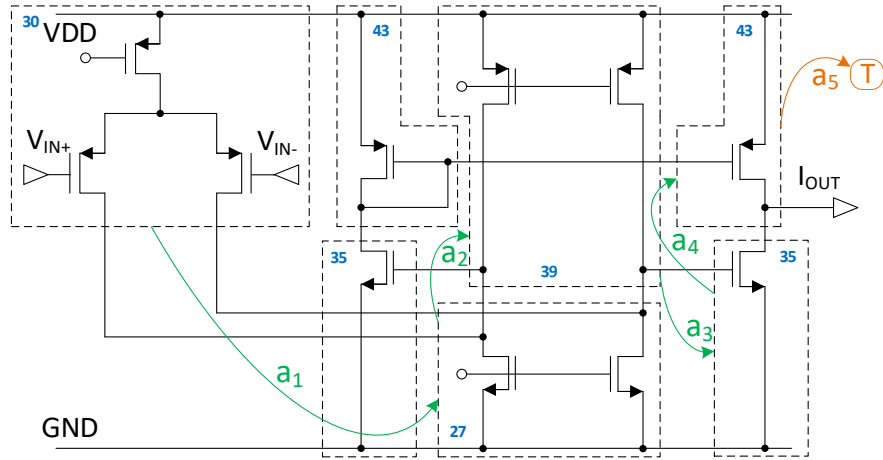
Table 5.4. PBBL Extension

Extension of Converter Building Blocks			
			
FI FI - V V		FO FO - V V	
Current Merger Building Blocks			
			
FI FI - FO		FO FO - FO	

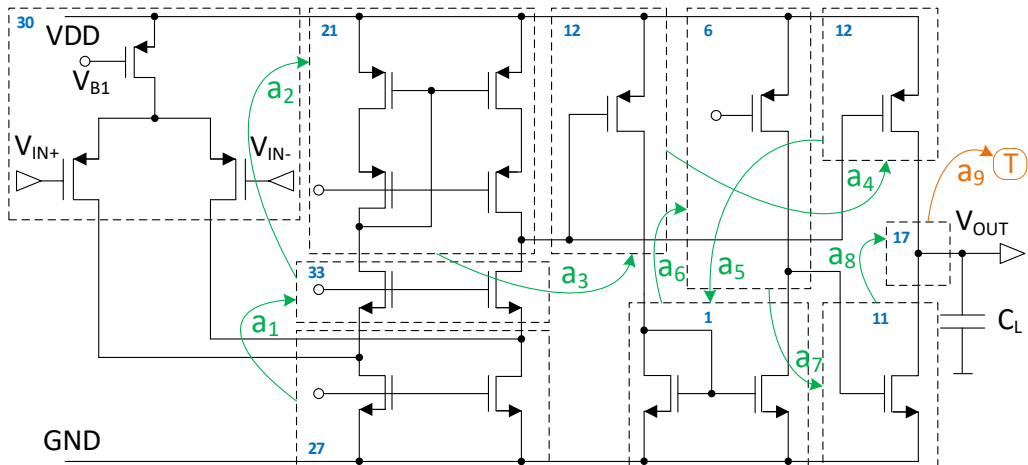
In this part, besides showing some good circuit topologies that can be synthesized by the proposed work, we will also demonstrate strong reliability and wide applicability of our methodology by using distinct output specifications and performance specifications to synthesize circuits. Fig. 5.4, Fig. 5.6, and Fig. 5.8 have already depicted the circuit topologies produced by applying the input-output specification as two voltage inputs and one voltage output. To synthesize circuit structures with distinct output specifications, the PBBL defined in Table 5.1 has to be extended. Table 5.4 lists some exemplary extension BBs. Among them, the extended converter-type BBs or current merger BBs are used to address output specification as two voltages or one current, respectively. Fig. 5.10(a) depicts a synthesized differential OpAmp, which is similar to the circuit shown in Fig. 5.6 except for the last BB selected by a_2 . In the differential OpAmp, the 39th BB from the extension BB library is chosen by action a_2 . Fig. 5.10(b) shows an operational transconductance amplifier (OTA) synthesized by our proposed method. It is synthesized by starting with the 30th BB (a PMOS-type differential pair) and taking four actions if excluding the terminational action a_5 shown in a different color. Action a_2 and a_4 select the 39th and 43rd BB from the extension BB library, respectively.



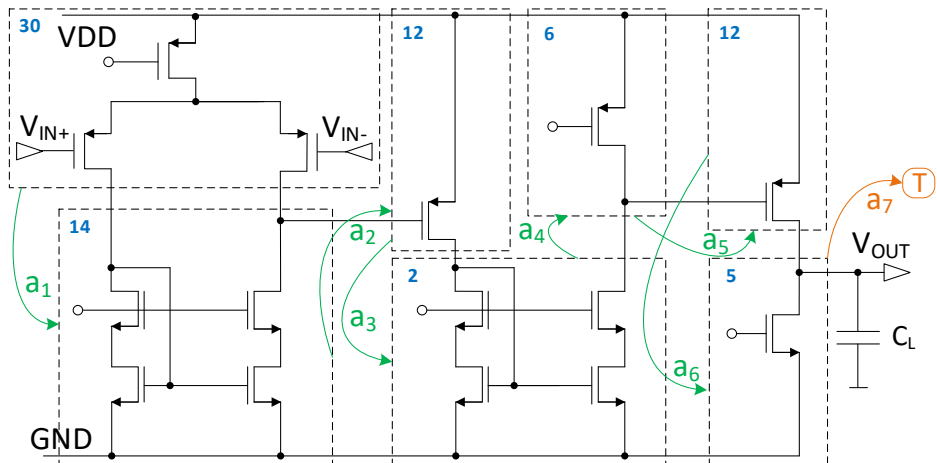
(a) A differential OpAmp



(b) An operation transconductance amplifier (OTA)



(c) A high gain OpAmp ($A_v > 100$ dB)



(d) A high unity-gain bandwidth OpAmp ($UGB > 1$ GHz)

Fig. 5.10. Example of synthesized circuits.

High gain OpAmps and high bandwidth OpAmps always attract more academic and industrial interests. These two categories of circuits with distinct performance specification settings can be readily synthesized by our proposed work. In our experiments, the high gain OpAmps are synthesized with the specification of 100dB A_v , 60° PM , 10dB GM , and 10MHz UGB , while the high bandwidth OpAmps are generated with the specification of 60dB A_v , 60° PM , 10dB GM , and 1GHz UGB . Fig. 5.10(c) depicts a synthesized high gain circuit, which is actually a three-stage OpAmp. The synthesis starts with a sub-circuit composed of a PMOS-type differential pair and an NMOS-type current mirror, and takes nine actions to construct the circuits. As mentioned in Section 5.4.1, if both normal-match-case2 and normal-match-case3 take place during the construction process, the difference of their occurring time steps will not affect the produced structure if the relative orders of the BBs belonging to the same type of match are preserved. Thereby, in Fig. 5.10(c), as long as action a_4 is taken after action a_3 but before action a_8 , its occurring time step will not affect the synthesized circuit structure. Fig. 5.10(d) illustrates a synthesized high bandwidth circuit, which contains seven BBs and needs seven actions to build. It can easily achieve more than 1 GHz UGB with power consumption around 0.6 mW, which demonstrates high efficacy of our proposed methodology.

As already demonstrated, even though the state size is only set to 10, the proposed work is still able to synthesize large analog circuits such as three-stage OpAmps. It can be inferred that with a larger size of state and PBBL, more complicated circuits can be synthesized by the proposed method due to strong scalability of the neural networks. In addition, our proposed work can synthesize innovative circuits. It has three means to potentially generate novel circuits: 1) by enriching PBBL with creative BB; 2) by making creative connections among known BBs; 3) by combining the two means above. Fig. 5.11 depicts a novel circuit synthesized by our work via the

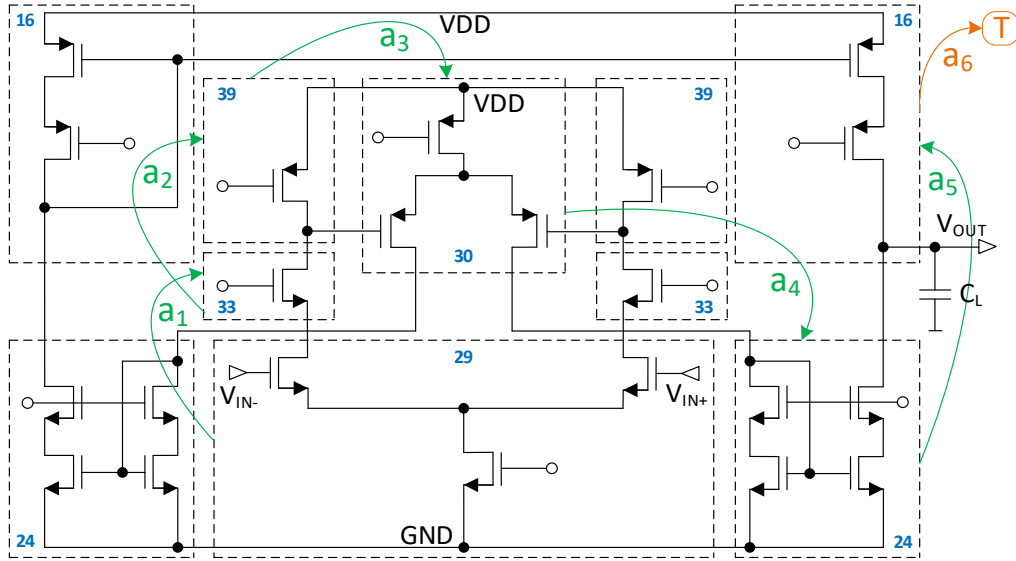


Fig. 5.11. A novel circuit synthesized.

second means. It is composed of six known BBs, while the creative connections of two differential pairs make this circuit innovative. It can not only reach more than A_v of 100dB, but also satisfy the other basic performance requirements specified in Section 5.5.2.

5.5.5. Comparison with the State-of-the-Art Methods

So far, we have demonstrated good efficiency, strong reliability, and wide applicability of our proposed DRL-based circuit topology synthesis method. In this part, we will illustrate some comparisons with other state-of-the-art circuit topology synthesis methods. Among these methods, the work of [54] utilizes design knowledge mining and reasoning (KMR) to synthesize circuit topologies while MOJITO [26], FEATS [44], and GCTG [J1] use genetic programming, graph composition, and graph decomposition methods to generate circuit topologies, respectively. Table 5.5 summarizes the characteristics of the above-mentioned four methods as well as our proposed DRL-based method.

Table 5.5. Comparison with Other Circuit Topology Synthesis Methods

	MOJITO	KMR	FEATS	GCTG	DRL
<i>Result Trustworthy?</i>	yes	yes	yes	yes	yes
<i>Design Search Space Flexible and Controllable?</i>	no	no	yes	yes	yes
<i>Large-Size Circuits Generalizable?</i>	no	no	yes	yes	yes
<i>Creative Circuits Capable?</i>	yes	–	yes	yes	yes
<i>Design Knowledge Automatically Learned?</i>	no	no	no	no	yes
<i>Computation Effort Affordable?</i>	no	no	no	no	yes

Since all these five methods utilize simulation-in-loop sizing to verify the produced circuit topologies, their synthesized solutions are trustworthy. There is no means for MOJITO to control its design search space, and its defined crossover and mutation operations are only workable for a narrow range of circuits, typically one-stage or two-stage OpAmps shown in its experimental results. Since KMR has to mine the design knowledge beforehand and then apply it to synthesize circuits in a reasoning way, it is difficult to flexibly extend its design search space and freely generalize its synthesis to another class of circuits that have not been learned. FEATS and GCTC are able to flexibly extend the design search space to address large-size circuit designs by extending the defined BB library or increasing the maximum number of BBs allowed for synthesis. Similar to them, our proposed DRL-based method can flexibly extend the design search space and easily generalize to large-size circuit design by extending PBBL or increasing state size.

Except for KMR, all the other methods claim that they are able to synthesize less-known circuits. The design knowledge is encoded in the predefined crossover/mutation operations, composition rules, and decomposition rules in the works of MOJITO, FEATS, and GCTC, respectively. Thus, there is no knowledge learning involved in these works. Since the design

knowledge learning process of KMR involves a great amount of human intervention, it is not deemed as automatic learning. However, our proposed work utilizes DRL to automatically learn design knowledge, which is stored in the neural network.

Due to the stochastic-driven synthesis, MOJITO needs to evaluate around 101,904 topologies in the process to finally produce 512 unique topologies, which took 7 days. Thereby, MOJITO suffers from almost unaffordable computation effort to synthesize circuits. KMR has to go through a quite timing-consuming data mining process because the similarity of circuits is extracted by comparing them in pairs. Due to the brute-force explorative nature of both FEATS and GCTC, their computation effort is also unaffordable especially if the size of the BB library is large. Table 5.6 illustrates the comparison among FEATS, GCTC, and our proposed DRL by using the same BB library as listed in Table 5.1 (i.e., the PBBL) and the same input-output specification (i.e., two voltage inputs and one voltage output). As one of the table headers in Table 5.6, topology size refers to the maximum block number, maximum leave number, and maximum state size allowed in FEATS, GCTC, and DRL, respectively. Since the DRL process features stochastic nature, its results in Table 5.6 are the average outputs from ten runs of our proposed algorithm.

Table 5.6. Comparison among FEATS, GCTG, and DRL by using PBBL

Topology Size	Number of the Unique Topologies Generated		
	FEATS	GCTG	DRL
5	1,056	832	20
6	6,780	6,152	43
7	43,175	45,776	61

As one can see, when the topology size is 5, FEATS and GCTC generate around 1,000 unique circuit topologies after applying their isomorphism checks, whereas our proposed DRL only produces 20 unique circuit topologies during the synthesis process. It is worth mentioning that such a significant number contrast should not be deemed as capability inferiority of circuit

topology construction for the DRL method. Instead by comparing the nature of the three methods above, we can see that FEATS and GCTC are only dedicated to the construction of circuit topologies in the brute-force way without concerning the quality of the generated topologies. Thus, a large majority of their generated circuit topologies turn to be fruitless. However, our proposed DRL method can smartly learn from the construction experience and only selectively generate increasingly more promising circuit topologies. More important, the number of the unique topologies produced by FEATS and GCTC grows exponentially in terms of the allowed topology size. For instance, as listed in Table 5.6, when the topology size increases to only 7, FEATS and GCTG would generate more than 40,000 unique circuit structures for evaluation, which definitely leads to a barrier to their industrial applications in practice. However, this topology explosion issue can be readily addressed by our proposed DRL methodology thanks to strong scalability of the neural networks. As shown in Table 5.6, the number of the unique circuit topologies generated by DRL only increases a bit when the topology size grows.

5.6. Summary

In this chapter, we presented a novel DRL-based method to synthesize analog circuits in a smart trial-and-error process that mimics the self-learning manner of humans, where the learned intelligence is stored in a neural network. It is able to manage large-size and innovative circuit synthesis. It also has wide applicability to deal with distinct input-output and performance specifications. Thanks to simulation-in-loop involvement and other speed-up schemes, it has the ability to accurately and efficiently verify the output solutions. Compared with the other state-of-the-art circuit topology synthesis methods, it can not only address their commonly known

shortcomings, but also achieve the least computation effort. Furthermore, the trained model can always be reused, which is able to provide a solution with the same input within one second.

However, this method is still suffers from an issue, that is, the learning process has to be performed from scratch once the technology or design specification changes. In the next chapter, we will introduce the transfer learning technique to transfer the learned knowledge from one learning process to another to largely save the learning effort of related tasks.

Chapter 6 Transfer Learning for Automated Analog Integrated Circuit Synthesis

6.1. Introduction

As already demonstrated in Chapter 5, compared with the traditional automated topology generation methods, our proposed DRL-based circuit topology synthesis framework (DRL-CTSF) can robustly produce a trustworthy solution with detailed device sizes within several hours. This is a significant improvement thanks to much less computation effort and more sensible application in general. However, in practice, the designers may have various I/O and performance specifications as well as distinct requirements of technology nodes for circuit design. To address these demands, a smart DRL-based circuit synthesis system (DRL-CSS) is proposed to be able to simultaneously and immediately provide solutions to the users as long as their inputs (i.e., design specification and technology requirement) are reasonable.

Such a proposed DRL-CSS is depicted in Fig. 6.1, which contains an interface for communication with the users and multiple clusters of DRL-CTSFs, each of which deals with a specific technology node (e.g., CMOS 90nm). Within a cluster of DRL-CTSFs, each DRL-CTSF would address a unique design specification, including I/O specification and performance specification. Once this system is well trained, the users can always get a solution immediately after inputting their design specifications and technology requirements. If an input given by the users has never been learned, the system will automatically initiate a learning process and then return a solution to the users. The learned knowledge is stored in the neural network for easy

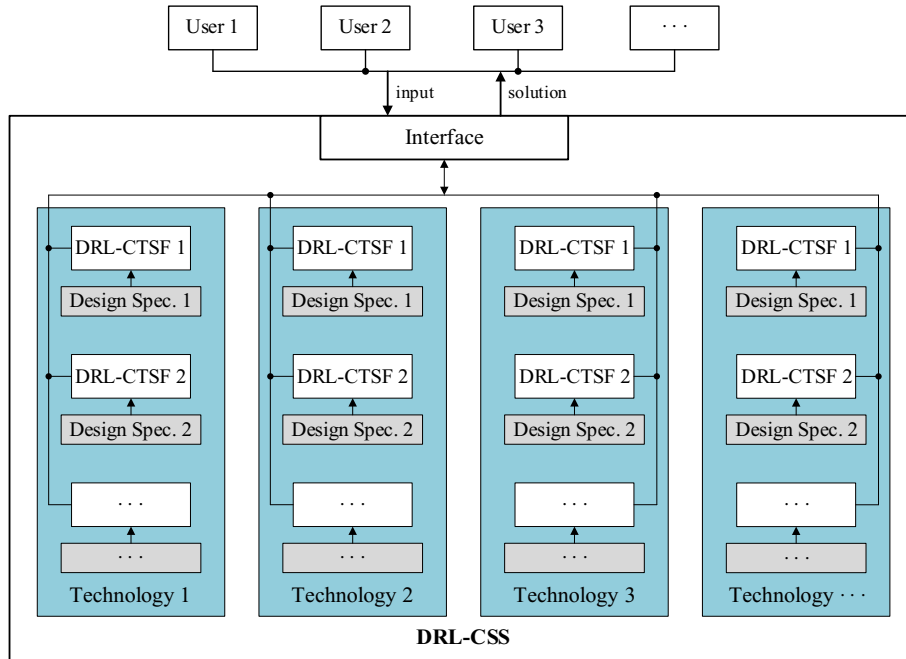


Fig. 6.1. The interaction between users and the DRL-CSS.

reusing. Therefore, it is a smart system that has the ability to continuously learn and evolve so that the more users there are, the smarter this system becomes.

Due to the inherent nature of reinforcement learning, the topology synthesis process of the proposed DRL framework always begins with a starting sub-circuit and then goes through a decision-making process to eventually produce a solution. This means that different starting sub-circuits provided by the users may cause distinct solutions. Therefore, training just one DRL framework to a sufficient extent is a quite time-consuming task since there are so many possible starting sub-circuits with various sizes in terms of the number of devices included. For instance, if the number of possible starting sub-circuits is just 100 and the training/learning time for each starting sub-circuit is 5 hours, then the total model training time would be more than 20 full days with day-and-night training. This fact indicates that training such a complex system, which includes multiple DRL frameworks to a fine-grained extent is practically an unaffordable task in

terms of computation effort. More important, if the target technology process node moves from one to another, the whole training process has to be re-performed from scratch, which would be an extremely painful task.

Transfer learning is a new technology in the field of artificial intelligence (AI) [92]. It is proposed to solve the bottleneck of the traditional machine learning technology, that is, limited data resources and computing resources, because it can automatically draw inferences from one another to transfer the learned knowledge from one domain to another [92]. It has already been successfully applied to some EDA areas to facilitate circuit design, such as mixed-signal circuit modeling [93], analog circuit sizing [94], and analog layout placement [95]. To address the issue of unaffordable computation effort that was painfully experienced in training the proposed DRL-CSS, in this chapter we will employ the transfer learning technique to transfer the learned knowledge from one trained DRL-CTSF to another. This will help contribute to significant reduction of the total training time so that our proposed work can be used in practice. To the best of our knowledge, this is the first work that applies transfer learning technique to synthesize analog integrated circuits in the EDA area. As the transfer learning itself, as one of the most popular machine learning techniques at the moment, is still evolving, in this chapter we only document the major concepts that we have used thus far to develop our methodology for acute analog topology synthesis. More enrichment and refinement work are expected as the future work for this research.

The research conducted in this chapter is under preparation for submission to IEEE Transactions on Very Large Scale Integration Systems [JR1].

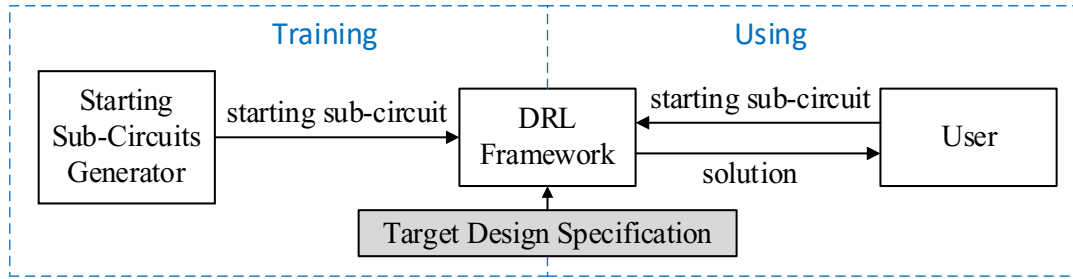


Fig. 6.2. Training and using of a DRL framework.

6.2. Training of DRL-CTSF

As mentioned in the previous introduction section (Section 6.1), in our proposed DRL-CSS, each DRL-CTSF would deal with a fixed design specification and thus various starting sub-circuits may be provided by the users. The experimental results in Section 5.5 have already demonstrated that the proposed DRL-CTSF has the ability to robustly produce a trustworthy solution for a target design specification as long as the users provide a valid starting sub-circuit. The validity of a starting sub-circuit will be explained later in this Section. Therefore, in order to ensure that a trained DRL framework can always return a solution immediately instead of experiencing a time-consuming training/learning process whenever the users input a valid starting sub-circuit, this DRL framework should be well trained with almost all possible valid starting sub-circuits.

Fig. 6.2 depicts a block diagram for an intuitive understanding of the training and using of a DRL-CTSF, which can be carried out simultaneously, but usually, the *training* part is done to some extent before the *using* part. The starting sub-circuits generator would automatically generate various starting sub-circuits for training a DRL-CTSF, which is bound with a specific design specification. However, after training within the limited number of iterations (i.e., a user-defined parameter), not all the generated starting sub-circuits could successfully derive solutions within the allowed design search space. Only those starting sub-circuits that can make it are called *valid*

starting sub-circuits. Although some starting sub-circuits are not valid under the current environment setting, it does not mean that training with such starting sub-circuits would always fail (i.e., cannot produce a solution). If the design search space is enlarged (e.g., by increasing the state size), those starting sub-circuits may become valid. It is worth noticing that only the valid starting sub-circuits are allowed to train a DRL-CTSF. For the starting sub-circuits that are confirmed as invalid, we need to restore the weights of the PGNN in the DRL-CTSF to be the ones before training them.

6.2.1. Hierarchical Building Block Library

As a matter of fact, the proposed DRL-CTSF is driven by the BB library since it treats BBs as the basic components to construct circuit structures, which means distinct BB libraries would contribute to different solutions being produced. Therefore, if some BBs naturally share similar structural characteristics and have close performance capability, their similarity can be utilized for subsequent relational-knowledge transfer in order to save the training time. For instance, according to our experimental studies, when replacing a BB of a circuit already satisfying a certain performance requirement with its Cascode-version BB, most likely this modified circuit would still satisfy the original performance requirement.

Based on the relationship among BBs, we can stratify the BB library into multiple layers, by which some more complex BBs are put into higher layers. In this way, instead of training the DRL-CSTF with the whole library from scratch, we can train it with the BBs layer by layer, as depicted in Fig. 6.3. Specifically, we can first train the DRL-CSTF with only the BBs in the first layer. After training to a good extent, the knowledge about similarity among BBs and the learned knowledge from previous training will be used to continue to train it with BBs in both the first and second

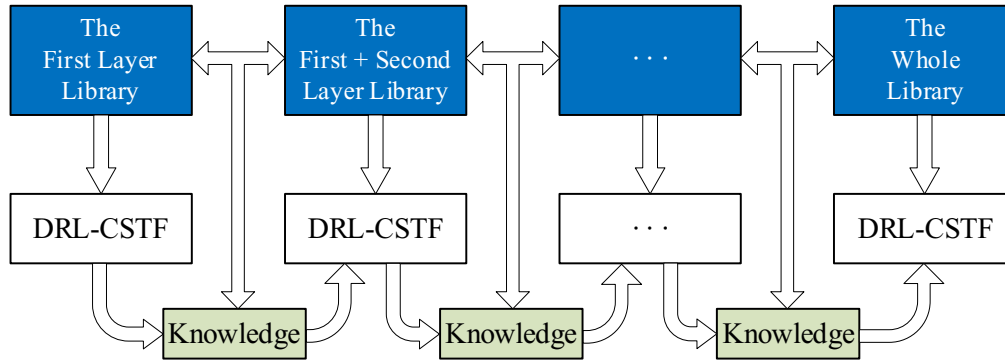


Fig. 6.3. Training DRL-CTS with hierarchical building block library.

layers. This process continues until all the layers in the library are included. Compared with the direct training with the whole library, training in the way of layer-by-layer would largely reduce the training time. There are two main reasons behind this. First of all, compared with the whole library, there are fewer BBs in the first layer, contributing to much smaller design space to search and much less computation effort to spend in order to train the DRL-CSTF to a sufficiently good extent with the BBs only in the first layer. Secondly, the knowledge obtained from the training in the previous phases and the similarity among the BBs in different layers would greatly facilitate the training in the later phase.

Table 6.1 illustrates an example hierarchical BB library. As one can see, this library contains 38 BBs in total, with 22, 8, and 8 BBs in the first, second, and third layer, respectively. Specifically, the first layer is made up of some basic one-signal-path and two-signal-path types of BBs. The second layer extends the first layer with four two-signal-path type BBs and four two-output type converters. The third layer is typically composed of Cascode BBs, each of which has a corresponding non-Cascode version in the first or second layer. For instance, the 31st and 32nd BBs are the Cascode version of the 1st and 2nd BBs (i.e., NMOS-type and PMOS-type current source),

Table 6.1. An example of hierarchical building block library

First Level											
Current Mirror		Current Source		Cascode Stage				Common Source		One-Output Converter	
$FI-FI$	$FO-FO$	$FI-V$	$FO-V$	$FI-FO$	$FO-FI$	$V-FI$	$V-FO$	$FI/FI-V$	$FO/FO-V$		
Differential Pair		Two Source-driven Cur. Splitter		Two Cascode Stage				Two Current Source			
$V/V-FI/FI$	$V/V-FO/FO$	$FI/FI-FI/FI$	$FO/FO-FO/FO$	$FI/FI-FO/FO$	$FO/FO-FI/FI$	$FI/FI-V/V$	$FO/FO-V/V$				
Second Level											
Two Current Mirror				Two Common Source				Converter			
$FI/FI-FI/FI$	$FO/FO-FO/FO$	$V/V-FI/FI$	$V/V-FO/FO$	$FI/FI-V/V$	$FO/FO-V/V$	$FI/FO-V$	$FO/FI-V$	$FI/FO-V/V$	$FO/FI-V/V$		
Third Level											
Cascode Current Mirror		Two Cascode Current Mirror		One-Output Converter		Two-Outputs Converter					
$FI-FI$	$FO-FO$	$FI/FI-FI/FI$	$FO/FO-FO/FO$	$FI/FI-V$	$FO/FO-V$	$FI/FI-V/V$	$FO/FO-V/V$				

respectively. Furthermore, in this library each BB has a unique index and each type of BB (e.g., current source) could be composed of either N-type or P-type MOSFET transistors, leading to distinct I/O terminal types. There are two special BBs (i.e., the 29th and 30th BB), which are composed of only nets. Both of them have two possible combinations of I/O terminals, which depends on the order of the input terminals.

6.2.2. Starting Sub-circuits Generation

In order to automatically train a DRL-CTSF to a satisfactory level, a suitable starting sub-circuit generation scheme is in demand. In this regard, we have proposed three methods to automatically generate the starting sub-circuits.

1) Brute Force

The most straightforward method is to follow a brute-force search flow, which explores all the possible starting sub-circuits within the design search space bound by the state size and the building block library. However, due to the brute-force nature, the number of the generated starting sub-circuits is very large while training each of them is a time-consuming process (e.g., several hours), leading to low efficiency of this method. More important, since it is uncertain what kind of starting sub-circuits are valid, a large percentage of those generated starting sub-circuits might be infeasible, causing a huge amount of computation effort is wasted on evaluating the invalid starting sub-circuits.

2) *Simulated Annealing*

In the process of an algorithm run, two hash tables, $T_{invalid}$ and T_{valid} , are built to record all the generated valid and invalid starting sub-circuits, respectively. The hash is calculated via (5.5). During an algorithm run, if a generated starting sub-circuit can successfully derive a solution after training, it is inserted into T_{valid} . Otherwise, this starting sub-circuit is inserted into $T_{invalid}$. For those valid starting sub-circuits recorded in T_{valid} , the number of their occurrences during an algorithm run is recorded as their associated values. Thereby, each time an starting sub-circuit is put into T_{valid} , its associated value is increased by 1 if it is already existed in T_{valid} . Otherwise, its associated value is set to be 1. In summary, T_{valid} is utilized to drive the SA algorithm while $T_{invalid}$ is used to avoid re-training those confirmed invalid starting sub-circuits to improve the efficiency of the algorithm.

The SA algorithm starts with a valid starting sub-circuit, which is treated as the current solution and put into the hash table T_{valid} . After successful training, a valid circuit that satisfies the design specification would be generated. At each iteration, the derived valid circuit from the current solution would be perturbed to produce a new starting sub-circuit, which will replace the

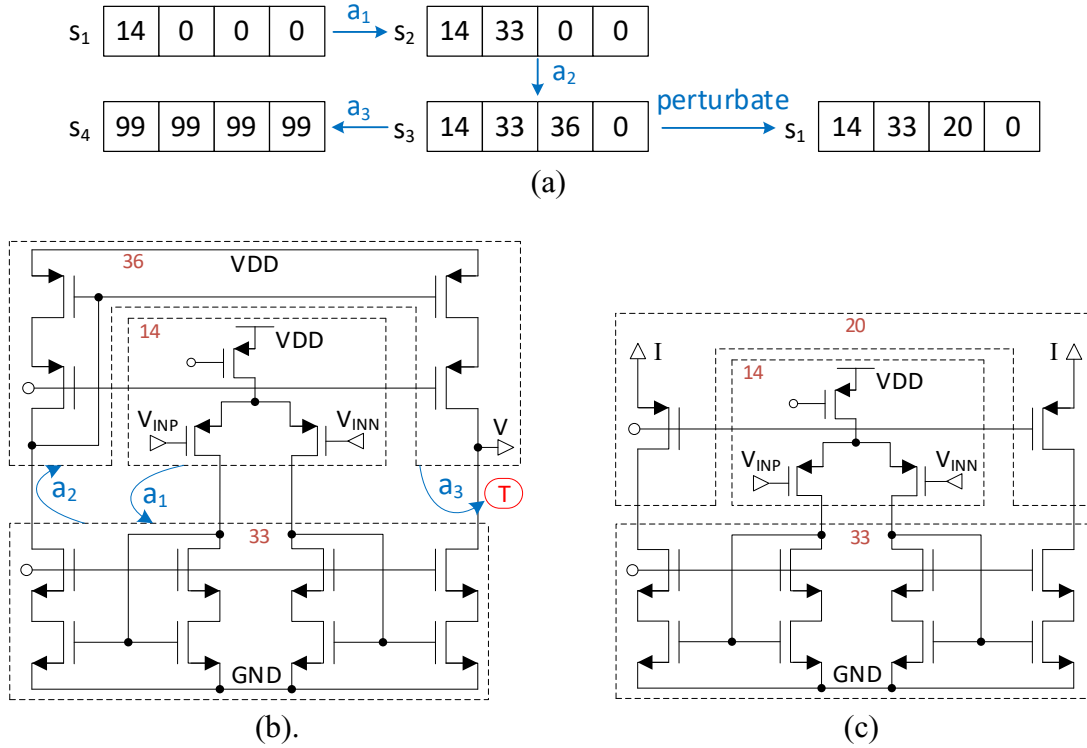


Fig. 6.4. An example of the perturbation operation.

current solution if it is confirmed as valid and has a smaller value (i.e., the number of its occurrences) than that of the current solution. Even though its value is larger than that of the current solution, substituting the current solution is still possible due to the property of SA algorithm. Once the solution is updated, the temperature goes down a bit. This iterative process continues until reaching the lowest limit of temperature or reaching the predetermined number of iterations.

Since all the starting sub-circuits in T_{valid} have already been learned, updating a starting sub-circuit from T_{valid} will always derive a determined valid circuit. From our observation of experiments, if we continuously perturb the same valid circuit, it becomes increasingly hard to obtain a valid starting sub-circuit that has never been seen, leading to low training efficiency. Due to this observation, the number of occurrences of starting sub-circuits in T_{valid} is utilized to drive

the proposed SA algorithm, and the starting sub-circuit with a small number of occurrences is preferred as the updated one.

An example of the above-mentioned perturbation operation is illustrated in Fig. 6.4. After training with a valid starting sub-circuit (state representation of [14, 0, 0, 0] in Fig. 6.4(a)), a circuit topology depicted in Fig. 6.4(b) (state representation of [14, 33, 36, 0] in Fig. 6.4(a)) is derived. The perturbation operation will randomly choose a non-zero slot in its state representation to change to any BB, excluding itself, from the building block library as long as they share the same input port(s). The slots after the chosen slot become zeros to indicate these seats are empty. In this example, the perturbation operation selects the third slot to change (i.e., the 36th BB becomes the 20th BB). After the perturbation operation, a new starting sub-circuit shown in Fig. 6.4(c) with the state representation of [14, 33, 20, 0] is produced.

Although the tricky part (i.e., what kind of starting sub-circuits are valid) is still uncertain, generally perturbing a valid circuit is more likely to derive another valid one, according to our experiments. Thus, the SA-based method is more efficient than the brute-force method that normally constructs circuits from scratch. The rationale behind this is that the preserved part (i.e., the new starting sub-circuit after perturbation) has been proved to be reasonably constructed and so it is more likely to succeed. Thereby, the proposed SA-based method is more desirable than the brute-force method.

3) *Minimum-Occurrence-Driven Method*

The two hash tables and perturbation operation employed in the SA-based method are also used in this method. The difference is that this method will always replace the current starting sub-circuit with the one that has the minimum number of occurrences in T_{valid} when it fails the training (i.e., confirmed as invalid) while the SA-based method cannot guarantee that. Such an updating

Algorithm 6.1: Min-Occurrence-Driven Training Scheme

Input: A untrained DRL-CTSF; Predetermined iteration
A valid starting sub-circuit s_0 ; Hash tables T_{valid} and
Output: A well trained DRL-CTSF.

1. Train DRL-CTSF with s_0 to derive a valid circuit C ;
 2. **While** (the size of $T_{valid} < N$):
 3. Perturb C to derive a new starting sub-circuit s ;
 4. **If** (s exists in T_{valid} or $T_{invalid}$):
 5. Get the starting sub-circuit with the minimum
 of occurrences from T_{valid} to replace s ;
 6. **Else:**
 7. Save the weights of the PGNN in DRL-CTSF as W ;
 8. Train DRL-CTSF with s ;
 9. **If** (training is not successful):
 10. Get the starting sub-circuit with the minimum
 of occurrences from T_{valid} to replace s ;
 11. Load W back to the PGNN; Insert s into $T_{invalid}$;
 12. **Else:**
 13. Insert s into T_{valid} and add its associated value by
 14. **End If;**
 15. **End If;**
 16. Use DRL-CTSF with s to derive a valid circuit to
 17. **End While;**
 18. **Return** DRL-CTSF;
-

feature will help this method do more exploitation compared with the SA-based method, contributing to better training efficiency, which will be demonstrated in the experimental results.

The details of this method are listed in Algorithm 6.1. As one can see, Lines 5 and 10 indicate that if the derived starting sub-circuit after a perturbation operation (Line 3) has already existed in the hash tables or confirmed as training failure, it will be replaced by a certain one with the minimal number of occurrences in T_{valid} . If it is confirmed as training success, no replacement operation is needed. At each iteration, the final determined starting sub-circuit will be utilized to derive a valid

circuit via DRL-CTSF immediately since this starting sub-circuit has already been learned, as shown in Line 16. This process continues until the predetermined iteration count, which is also the size of T_{valid} , is reached.

6.3. Training of DRL-CSS

6.3.1. Preliminary DRL Framework

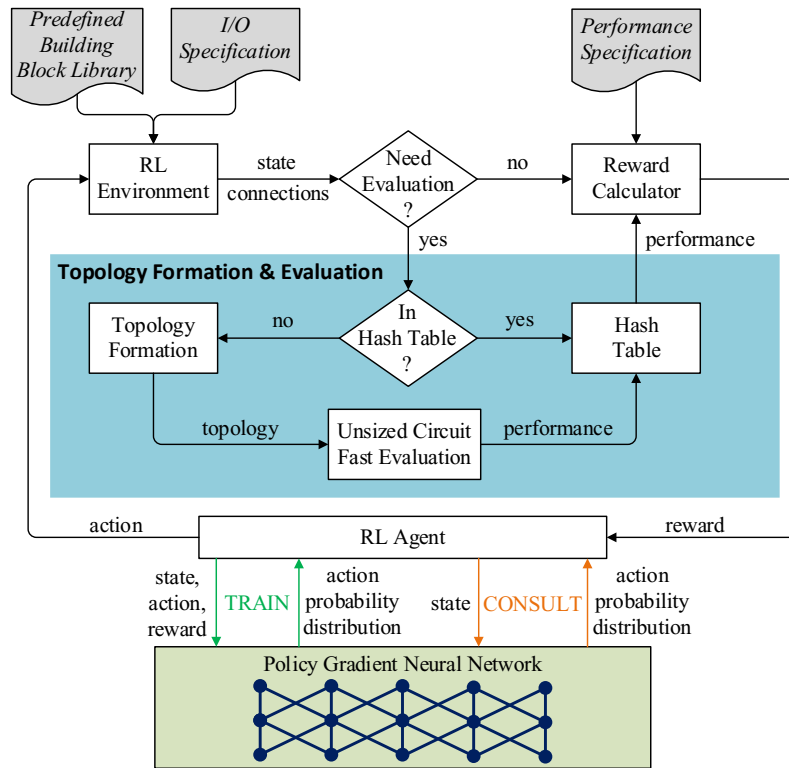


Fig. 6.5. Preliminary DRL-CTSF.

For the first stage of training the proposed DRL-CSS, instead of directly training a comprehensive DRL-CTSF, we opt to utilize the concept of transfer learning by first training its preliminary version, which is a DRL-CTSF excluding the simulation-in-loop sizing phase in its circuit topology formation & evaluation process as depicted in Fig. 6.5. Compared with training a

comprehensive DRL-CTSF, training a preliminary DRL-CTSF would be significantly faster since the most time-consuming operation within a learning process in DRL-CTSF is circuit sizing. This means that we promote to trade accuracy for efficiency at this level of training. Here, inaccuracy refers to the derived circuits from the trained model may not actually meet the target specification. After training the preliminary DRL-CTSF to a certain satisfactory level, the synthesis knowledge is stored in its PGNN model. Even though this knowledge may not be that accurate, it still can greatly reduce the overhead of training any comprehensive DRL-CTSF with the same environmental settings. Later by coping the trained PGNN from the preliminary DRL-CTSF to the comprehensive DRL-CTSF, any further training can get much smoother.

6.3.2. Knowledge Transfer Among Design Specifications and Technology Process Nodes

After completing the training of a DRL-CTSF, its learned knowledge can be transferred to train another DRL-CTSF with distinct design specifications but in the same technology node. Some design specifications are strongly related. For instance, one specification requires 60dB dc gain while another needs 80dB dc gain. For the two DRL-CTSFs that deal with these two specifications, the knowledge obtained from a trained one can be used to train the other one by applying the transfer learning technique since they share the same domain but address different tasks. Even though some design specifications might not be strongly related, such as high gain and high bandwidth, they may still share the same requirements for the other performance attributes (e.g., gain margin and phase margin). For this situation, transfer learning is still helpful to reduce the training overhead. A typical strategy used via transfer learning is to freeze some lower layers of the neural network model, which means the weights of the frozen layers are not allowed to update when training it for a new task. Since the neural network model contained in our DRL-

CTSF has several hidden layers, the number of the layers that need to be frozen to get the best training service has to be explored through experiments.

Although circuits may behave quite differently in distinct technologies, it is mainly caused by circuit biases and sizes. Through our experimental studies, we found that the same circuit structure had similar performance capability in different technologies. This observation has inspired us to consider transfer learning as a suitable medium for conveying the knowledge about circuit structure synthesis among the DRL-CSTFs in the different technologies. Thereby, after finishing the training of a cluster of DRL-CSTFs that process distinct specifications for the same technology, transfer learning is also helpful to facilitate the training of another cluster of DRL-CSTFs that deal with a different technology.

6.4. Summary

In this chapter, we have proposed a preliminary smart DRL-CSS. It can immediately return a solution to the users if their inputs have already been learned. Otherwise, it will go through a learning process and then return a solution. It can address various requests of design specifications and technology nodes from the user side. It has the ability to continuously learn and evolve so that the more users there are, the smarter this system becomes. A transfer learning scheme is also proposed to significantly reduce the overhead of training DRL-CTSFs from scratch, thus greatly cutting down the computation effort of training DRL-CSS.

In the next chapter, we will conclude the works studied in this dissertation and discuss the future work.

Chapter 7 Conclusion and Future Work

In this dissertation, all our research works are aimed at developing a circuit synthesis EDA tool for analog integrated circuits, which still has no widely accepted commercial solutions even at the research level. In order to make the developed EDA tool commercializable, there are many challenges that have to be solved, including wide applicability, sound generalization capability, and affordable computation effort. Currently, all the existing research works are not able to perfectly address the abovementioned challenges.

Chapter 3 proposed an automated graph-grammar-based topology synthesis framework for analog ICs. The circuit topology synthesis is efficiently realized by encoding circuit topology generation process as tree structure construction. Each graph node in the tree structure represents a black block that is associated with only input and output terminals. The tree construction process is realized by iteratively splitting leaves starting from the root node, which symbolizes the input-output specification of the represented circuits. The splitting operation is guided by our defined decomposition rules, which would largely reduce generating meaningless circuit topologies by matching the input-output terminals of the connected block nodes. Besides the decomposition rules, we have also defined structural symmetry rule, branch termination rule, deconstruction rule, and last-level constraint rule to help improve the efficiency and quality of the tree structure construction process. Then the leaves of the produced tree structures are mapped to their corresponding basic BBs in the predefined BB library. Finally, transistor-level circuit topologies will be formed by connecting those BBs according to the recorded connection information during the tree structure construction process. To ensure only unique circuit topologies to be generated, two levels of isomorphism checks are performed at both tree structure level and circuit topology

level. The experimental results demonstrate that more than half of the produced circuit topologies would be eliminated due to duplication.

As mentioned in Section 2.1.3, one of the main challenges of automated circuit topology synthesis is its efficiency. Evaluating such a huge number of circuit topologies produced in the synthesis process is practically a task with unaffordable computation challenge, if each circuit topology has to go through a quite time-consuming sizing process for performance feasibility checking. To address this issue, in Chapter 3 we proposed a novel fast evaluation method to roughly evaluate un-sized circuits, which trade accuracy for efficiency. In this way, only a small portion of produced circuit topologies can pass the fast evaluation stage and go to the detailed sizing stage. The fast evaluation consists of three tests, each of which acts as a filter to eliminate the circuit topologies that do not satisfy the requirements. The first one is the linear constraint test, which models the biasing constraints, symmetry constraints, and common constraints from a circuit topology as the linear equality or inequality constraints in the form of a linear programming problem. Once the modeled problem has a feasible solution, the circuit topology passes this test. The second one is the DC gain test, which utilizes the estimated center values of small-signal model parameters to numerically calculate the DC gain through the GPDD symbolic analysis algorithm. This test can largely avoid the ill-constructed circuit topologies. The third test is the fast sizing & simulation test, which integrates symbolic analysis, gm/ID methodology, and nonlinear programming technique together to fast and coarsely size the circuit topologies and then check the performance feasibility via SPICE simulation.

Compared with the state-of-the-art topology synthesis methods, this proposed method possesses wider applicability because it can address the situation of current division and voltage division, which have not been solved by any other methodologies before. It also achieves higher

synthesis efficiency due to the proposed fast evaluation method that can largely reduce the circuit topologies to be sized. However, since this method will explore all the possibilities in a brute-force manner within its allowed design search space, the number of circuit topologies that pass the fast evaluation stage is still large, especially when the circuit size is large. To further boost the synthesis efficiency, in Chapter 4, a novel performance modeling method is proposed to substitute the computation-intensive and time-consuming simulator in the sizing process.

To improve the sizing efficiency, a variety of performance models, which can estimate circuit performance significantly faster compared with simulators, have been proposed in the past two decades. A generic flow for the generation of performance models is provided as follows. The SPICE netlist of a circuit extracted from the schematic or physical design is the input to the modeling flow. At the first step, the circuit parameters (e.g., transistor length and width) and target characteristics or figures-of-merit are identified. The netlist is then parameterized for these parameters to obtain a parameterized netlist, and each of the characteristics will have a model. A sampling technique is employed to obtain discrete samples of the design parameters. Then, the parameterized netlist is simulated through the SPICE simulator for the target characteristics with the sampled values of the design parameters. Finally, these collected sample points and their corresponding characteristics are utilized for the regression or fitting operation of performance models. In order to achieve sufficient accuracy, a suitable sampling scheme should be selected, and a large number of simulations have to be conducted. More important, the complexity of this sampling process grows exponentially versus the dimension of the design parameters.

Therefore, building performance models is generally difficult and time-consuming. Moreover, this time-consuming process has to restart from scratch for every new circuit. This severe drawback makes all the existing performance modeling approaches cannot be applied to the

topology synthesis works, which would produce a large number of various circuit structures during the synthesis process. To overcome this problem, in Chapter 4, we proposed a novel performance modeling method that does not need to re-extract sampling data for new circuits, making it be able to rapidly adapt to any variation of circuit topology in the same technology. Different from the existing modeling methods that attempt to directly model circuit behaviors, our proposed method focuses on modeling transistor behaviors and then efficiently computes circuit performances based on the established transistor models.

In Chapter 4, we take advantage of the advanced neural network (NN) regression technique, which is naturally suitable to process the complex and nonlinear relationship between input variables and output performances, to generate transistor models that can accurately fit these functions. Through an undirected bipartite graph (UBG), which is constructed when reading a circuit netlist, circuit topology information can be automatically and efficiently extracted. By integrating the established transistor models and extracted circuit topology information, the transistor DC operating points can be efficiently computed. Once they are confirmed, the small-signal model parameter values of all the transistors in the circuit can be directly derived through the established transistor models. Finally, the symbolic analysis technique is employed to efficiently calculate the circuit performances in terms of small-signal model parameters. The experimental results indicate that the proposed performance modeling method can boost the sizing efficiency by more than 30 times with ignorable model building overhead, making it especially suitable for circuit synthesis works that involve generating various circuit structures.

Although promoting sizing efficiency is an effective way to improve the whole circuit topology synthesis efficiency, the efficiency of the proposed graph-grammar-based topology synthesis method is still low after employing the proposed performance modeling method in the

sizing process, in that the number of circuit topologies produced in the synthesis process is too big. For instance, if at each explorative step there are only 6 possible choices to map and the maximum allowed exploring step is 8, the number of topologies generated would be 6^8 , which means more than one million topologies would be generated. To further improve the synthesis efficiency to be computation affordable, in Chapter 5, we have proposed a novel deep-reinforcement-learning-based topology synthesis method with distinct synthesis mechanisms, which treat the synthesis process as a decision-making learning process. With the help of artificial intelligence, the synthesis algorithm would become increasingly smarter, that is, gradually learn the design knowledge to make the optimum decision at each synthesis step. Due to this synthesis mechanism, in general, the deep-reinforcement-learning-based topology synthesis method would produce much fewer unique circuit structures to find a solution compared with the graph-grammar-based topology synthesis method, which means considerably less computation effort consumed.

To speed up the learning process, we have defined some basic design rules to let the DRL process follow, which would largely avoid constructing meaningless circuit structures even though the DRL has the ability to automatically learn the optimal selection strategy through its continuous trial-and-error process. In addition, hash table and symbolic analysis are employed in this work to reduce the number of produced circuit topologies to be sized during the synthesis process. The parallel computing technique is adopted to speed up the time-consuming circuit sizing process. As the experiments depicted in Section 5.5.5, the computation effort of the DRL-based topology synthesis method is much less than the graph-grammar-based topology synthesis method thanks to much fewer unique circuit topologies to be sized in the synthesis process.

Although the circuit topology synthesis system proposed in chapter 5 is promising, which is able to provide solutions to the users immediately as long as their input design specifications are

reasonable, training this system to a satisfactory level is quite challenging. Its training time consumed and computation resources needed are unaffordable. To solve this challenge, transfer learning, one of the most popular machine learning techniques, is applied for training this system in chapter 6. Transfer learning can automatically draw inferences from one another to transfer the learned knowledge from one trained DRL framework to greatly reduce the overhead of training another DRL framework from scratch. With the help of transfer learning, the total training time is significantly reduced, contributing to the proposed system can be used in practice.

There is still a lot of room for improving the DRL-based circuit topology synthesis method, which can be treated as future works. For instance, this method still applies the SPICE simulator to evaluate circuit performance, which can be replaced by the proposed performance modeling method to further enhance the synthesis efficiency. This method does not address the current division and voltage division situations, leading to only a narrow range of circuits to be synthesized. Thus, widening its applicability is also a direction to go in future research. In addition, since the learning process has to be performed from scratch once the technology or design specification changes, addressing this issue should also be a good topic to study in the future.

As the success of circuit topology synthesizer relies on the degree of solving the three challenges mentioned in Section 2.1, there are generally three main directions to go for future works. For instance, making the synthesizer have the ability to produce RF circuits and mixed-signal circuits would widen its applicability. Furthermore, some OpAmps circuits contain compensation sub-circuits, but no existing works are able to automatically synthesize this kind of circuits, which provides an opportunity for the next-generation topology synthesizer to go.

References

- [1] S. Sorkhabi and L. Zhang, "Automated topology synthesis of analog and RF integrated circuits: a survey," *Integration, the VLSI Journal*, vol. 56, 2017, pp. 128-138.
- [2] E. Yilmaz and G. Dunder, "Analog layout generator for CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, 2009, pp. 32-45.
- [3] M. P. H. Lin, Y. W. Chang, and C. M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macau, pp.617-622, 2016.
- [4] L. Zhang, in *VLSI circuit layout*, Hoboken, NJ, USA, Wiley Encyclopedia Computer Science and Engineering, 2009, pp. 3034-3044.
- [5] B. Razavi, *Fundamentals of microelectronics*, Hoboken, NJ: John Wiley & Sons, 2021.
- [6] J. H. a. W. S. R.J. van de Plassche, *Analog Circuit Design: High-Speed Analog-to-Digital Converters, Mixed Signal Design; PLLs and Synthesizers*, Boston: Springer Science & Business Media.
- [7] P. R. Gray, P. J. Hurst, S. H. Lewis, and R.G. Meyer, *Analysis and Design of Analog Integrated Circuits*, 5-th Edition, New York: John Wiley & Sons, Inc., 2009.
- [8] Z. Zhao, T. Liao, and L. Zhang, "Fast performance evaluation for analog circuit synthesis frameworks," in *Proc. IEEE International Symposium on Circuits and Systems*, Florence, Italy, pp. 1-5, 2018.
- [9] G. Stehr, H. E. Graeb and K. J. Antreich, "Analog Performance Space Exploration by Normal-Boundary Intersection and by Fourier–Motzkin Elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 10, 2007, pp. 1733-1748.
- [10] S. P. Mohanty, *Nanoelectronic mixed-signal system design*, New York: McGraw-Hill Education, 2015.

- [11] A. Girardi, F. P. Cortes, and S. Bampi, "A tool for automatic design of analog circuits based on gm/ID methodology," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4643-4646, 2006.
- [12] R. Harjani, R.A. Rutenbar, and L.R. Carley, "A prototype for knowledge-based analog circuit synthesis," in *Proc. Proceedings of the 24th Conference on Design Automation*, pp. 42-49, 1987.
- [13] I. O'Connor and A. Kaiser, "Automated design of switched-current cells," in *Proc. IEEE Custom Integrated Circuits Conference*, pp.477-480, 1998.
- [14] H.Y. Koh, C.H. Sequin, and P.R. Gray, "OPASYN: a compiler for CMOS operational," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 2, 1990, pp. 113-125.
- [15] R. Harjani, R.A. Rutenbar, and L.R. Carley, "OASYS: a framework for analog circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 12, 1989, pp. 1247-1266.
- [16] G.V. d. Plas, G. Debyser, et al., "AMGIE-A synthesis environment for CMOS analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, 2001, pp. 1037-1058.
- [17] P.C. Maulik, R. Carley, and R. Rutenbar, "Integer programming based topology selection of cell-level analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 4, 1995, pp. 401-412.
- [18] Z. Gan, A. Yang, T. Shang, T. Yu, and M. Jiang, "Automated synthesis of passive analog filters using graph representation," *Expert Syst. Appl.*, vol. 37, no. 3, 2010, pp. 1887-1898.
- [19] K.J. Kim and S.B. Cho, "Automated synthesis of multiple analog circuits using evolutionary computation for redundancy-based fault-tolerance," *Appl. Soft Comput.*, vol. 12, no. 4, 2012, pp. 1309-1321.
- [20] J. Slezak and J. Petrzela, "Evolutionary synthesis of cube root computational circuit using graph hybrid estimation of distribution algorithm," *Radioengineering*, vol. 23, no. 1, 2014, pp. 549-558.
- [21] M.W. Cohen, M. Aga, and T. Weinberg, "Genetic algorithm software system for analog circuit design," *Procedia CIRP*, vol. 36, 2015, pp. 17-22.

- [22] T. Sripramong and C. Toumazou, "The invention of CMOS amplifiers using genetic programming and current-flow analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, 2002, pp. 1237-1252.
- [23] T.R. Dastidar, P.P. Chakrabarti, and P. Ray, "A synthesis system for analog circuits based on evolutionary search and topological reuse," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 2, 2005, pp. 211-224.
- [24] E. Martens and G. Gielen, "ANTIGONE: top-down creation of analog-to-digital convertor architecture," *VLSI Journal Integration*, vol. 42, no. 1, 2009, pp. 10-23.
- [25] E. Martens and G. Gielen, "Top-Down heterogeneous synthesis of analog and mixed-signal systems," in *Proc. Design, Automation and Test in Europe (DATE)*, 2006.
- [26] T. McConaghy, P. Palmers, M. Steyaert, and G.G.E. Gielen, "Trustworthy genetic programming based synthesis of analog circuit topologies using hierarchical domain specific building blocks," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, 2011, pp. 557-570.
- [27] T. McConaghy, P. Palmers, M. Steyaert, and G.G.E. Gielen, "Variation-aware structural synthesis of analog circuits via hierarchical building blocks and structural homotopy," *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 9, 2009, pp. 1281-1294.
- [28] T.M.P. Palmers, M. Steyaert, and G. Gielen, "Massively multi-topology sizing of analog integrated circuits, design, automation & test," in *Proc. Europe Conference Exhibition (DATE)*, pp. 706-711, 2009.
- [29] K. Francken, P. Vancorenland, and G. Gielen, "DAISY: A simulation-based high-level synthesis tool for delta-sigma modulators," in *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 188-192, 2000.
- [30] K. Francken and G.G.E. Gielen, "A high level synthesis environment for delta-sigma modulators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 8, 2003, pp. 1049-1061.
- [31] J.B. Grimbleby, "Automatic analogue network synthesis using genetic algorithms," in *Proc. International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 53-58, 1995.
- [32] J.D. Lohn, S.P. Colombano, "A circuit representation technique for automated circuit design," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, 1999, pp. 205-219.

- [33] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, 2007, pp. 596-607.
- [34] A.T. Soto, E.E. d.L. Sentí, A.H. Aguirre, and E.D.D.M.D. Soto, "A robust evolvable system for the synthesis of analog circuits," *Comput. Y Sist.*, vol. 13, no. 4, 2010, pp. 409-421.
- [35] E.A.M. Klumperink, F. Bruccoleri, and B. Nauta, "Finding all elementary circuits exploiting transconductance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 48, no. 11, 2001, pp. 1039-1053.
- [36] F. Bruccoleri, E. A. M. Klumperink, and B. Nauta, "Generating all 2-transistor circuits leads to new wide-band CMOS LNAs," in *Proc. 26rd European Solid-State Circuits Conference (ESSCIRC)*, pp. 316-319, 2000.
- [37] F. Bruccoleri, E.A.M. Klumperink, and B. Nauta, "Generating all two-MOS-transistor amplifiers leads to new wide-band LNAs," *IEEE J. Solid-State Circuits*, vol. 36, no. 7, 2001, pp. 1032-1040.
- [38] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Proc. 32nd Conference on Design Automation (DAC)*, pp. 433-438, 1995.
- [39] X. Wang and L. Hedrich, "An approach to topology synthesis of analog circuits using hierarchical blocks and symbolic analysis," in *Proc. Asia and South Pacific Conference on Design Automation*, pp. 700-705, 2006.
- [40] X. Wang and L. Hedrich, "Hierarchical exploration and selection of transistor-topologies for analog circuit design," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1467-1470, 2006.
- [41] X. Wang and L. Hedrich, "Hierarchical symbolic analysis of analog circuits using two-port networks," in *Proc. 6th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing*, pp. 21-26, 2007.
- [42] O. Mitea, M. Meissner, and L. Hedrich, "Topology synthesis of analog circuits with yield optimization and evaluation using pareto fronts," in *Proc. 19th International Conference on VLSI and System-on-Chip (VLSI-SoC)*, pp. 78-81, 2011.
- [43] O. Mitea, M. Meissner, L. Hedrich, and P. Jores, "Automated constraint-driven topology synthesis for analog circuits, design, automation & test," in *Proc. Europe Conference & Exhibition (DATE)*, pp. 1-4, 2011.

- [44] M. Meissner and L. Hedrich, "FEATS: Framework for explorative analog topology synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 34, no. 2, 2015, pp. 213-226.
- [45] M. Ma, M. Meissner, and L. Hedrich, "A case study: automatic topology synthesis for analog circuit from an ASDeX specification," in *Proc. International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 9-12, 2012.
- [46] O. Mitea, M. Meissner, L. Hedrich, and P. Jores, "Automated constraint-driven topology synthesis for analog circuits, design, automation & test," in *Proc. Europe Conference & Exhibition (DATE)*, pp. 1-4, 2011.
- [47] M. Meissner, O. Mitea, L. Luy, and L. Hedrich, "Fast isomorphism testing for a graph-based analog circuit synthesis framework, design, automation & test," in *Proc. Europe Conference & Exhibition (DATE)*, pp. 757-762, 2012.
- [48] A. Das and R. Vemuri, "Topology synthesis of analog circuits based on adaptively generated building blocks," in *Proc. 45th ACM/IEEE Design Automation Conference (DAC)*, pp. 44-49, 2008.
- [49] A. Das and R. Vemuri, "A graph grammar based approach to automated multiobjective analog circuit design," in *Proc. ACM/IEEE Design, Automation & Test in Europe Conference (DATE)*, pp. 700-705, 2009.
- [50] C. Ferent and A. Daboli, "A symbolic technique for automated characterization of the uniqueness and similarity of analog circuit design features," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-4, 2011.
- [51] C. Ferent and A. Daboli, "Analog circuit design space description based on ordered clustering of feature uniqueness and similarity," *VLSI Journal Integration*, vol. 47, no. 2, 2014, pp. 213-231.
- [52] C. Ferent and A. Daboli, "Systematic comparison of two low-voltage amplifiers using topology matching and performance constraints," in *Proc. IEEE 10th International New Circuits and Systems Conference (NEWCAS)*, pp. 225-227, 2012.
- [53] C. Ferent, S. Montano, and A. Daboli, "A prototype framework for conceptual design of novel analog circuits," in *Proc. International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 13-16, 2012.

- [54] C. Ferent and A. Dobioli, "Novel circuit topology synthesis method using circuit feature mining and symbolic comparison," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1-4, 2014.
- [55] C. Ferent, A. Dobioli, "Symbolic matching and constraint generation for systematic comparison of analog circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 4, 2013, pp. 616-629.
- [56] F. Jiao, S. Montano, and A. Dobioli, "Knowledge-intensive, casual reasoning for analog circuit topology synthesis in emergent and innovative applications," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1144-1149, 2015.
- [57] F. Jiao and A. Dobioli, "Causal reasoning mining approach to analog circuit verification, integration," *VLSI Journal Integration*, vol. 55, 2016, pp. 376-383.
- [58] F. Jiao, S. Montano, C. Ferent, A. Dobioli, and S. Dobioli, "analog circuit design knowledge mining: discovering topological similarities and uncovering design reasoning strategies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 34, no. 7, 2015, pp. 1045-1058.
- [59] H. Li, F. Jiao, and A. Dobioli, "Analog circuit topological feature extraction with unsupervised learning of new sub-structures," in *Proc. Europe Conference & Exhibition on Design Automation & Test in (DATE)*, pp. 1509-1512, 2016.
- [60] F. Jiao and A. Dobioli, "Three learning methods for reasoning-based synthesis of novel analog circuits," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2411-2414, 2016.
- [61] F. Jiao and A. Dobioli, "A low-voltage, low-power amplifier created by reasoning-based, systematic topology synthesis," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2648-2651, 2015.
- [62] L. Zhang, Y. Zhang, Y. Jiang, and C.-J. R. Shi, "Symmetry-aware placement with transitive closure graphs for analog layout design," *International Journal of Circuit Theory and Application*, vol. 38, no. 3, 2010, pp. 221-241.
- [63] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "SubGemini: Identifying subCircuits using a fast subgraph isomorphism algorithm," in *Proc. ACM/IEEE Design Automation Conference*, Dallas, TX, USA, pp. 31-37, 1993.

- [64] G. Shi, "Graph-pair decision diagram construction for topological symbolic circuit analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 2, 2013, pp. 275-288.
- [65] G. Shi, H. Hu, and S. Deng, "Topological approach to automatic symbolic macromodel generation for analog integrated circuits," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 3, 2017, pp. 1-25.
- [66] T. Liao and L. Zhang, "Parasitic-aware gm/ID-based many-objective analog/RF circuit sizing," in *Proc. IEEE Int. Symp. Qual. Electron. Design*, Santa Clara, CA, USA, pp. 100-105, 2018.
- [67] T. Liao and L. Zhang, "Layout-dependent effects aware gm/iD-based many-objective sizing optimization for analog integrated circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, pp. 1-5, 2018.
- [68] Y.-F. Cheng, L.-Y. Chan, Y.-L. Chen, Y.-C. Liao, and C.-N. J. Liu, "A bias-driven approach to improve the efficiency of automatic design optimization for CMOS OP-Amps," in *Proc. Asia Symp. Qual. Electron. Design (ASQED)*, Penang, Malaysia, pp. 59-63, 2012.
- [69] M. Barros, J. Guilherme and N. Horta, "Analog circuits optimization based on evolutionary computation techniques," *Integration*, vol. 43, no. 1, pp. 136-155, 2010.
- [70] G. G. E. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825-1854, 2000.
- [71] W. Daems, G. Gielen, and W. Sansen, "Simulation-based automatic generation of signomial and posynomial performance models for analog integrated circuit sizing," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, San Jose, pp. 70-74, 2001.
- [72] S.H. Lui, H.K. Kwan, and N. Wong, "Analog circuit design by nonconvex polynomial optimization: two design examples," *International Journal of Circuit Theory and Applications*, vol. 38, no. 1, 2010, pp. 25-34.
- [73] G. Yu and P. Li, "Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, San Jose, pp. 464-469, 2007.
- [74] A. Pradhan and R. Vemuri, "Regression based circuit matrix models for accurate performance estimation of analog circuits," in *Proc. International Conference on Very Large Scale Integration*, Atlanta, 2007.

- [75] D. Boolchandani, A. Ahmed, and V. Sahula, "Efficient kernel functions for support vector machine regression model for analog circuits' performance evaluation," *Analog Integrated Circuits and Signal Processing*, vol. 66, no. 1, 2011, pp. 117-128.
- [76] G. Wolfe and R. Vemuri, "Extraction and use of neural network models in automated synthesis of operational amplifiers," *IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems*, vol. 22, no. 2, 2003, pp. 198-212.
- [77] M. B. Alawieh, F. Wang, R. Kanj, X. Li, and R. Joshi, "Efficient analog circuit optimization using sparse regression and error margining," in *Proc. International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, 2016.
- [78] I. Guerra-Gomez, T. McConaghy, and E. Tlelo-Cuautle, "Study of regression methodologies on analog circuit design," in *Proc. Latin-American Test Symposium (LATS)*, Puerto Vallarta, pp. 1-6, 2015.
- [79] T. McConaghy and G. G. E. Gielen, "Template-free symbolic performance modeling of analog circuits via canonical-form functions and genetic programming," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 8, 2009, pp. 1162-1175.
- [80] K. Jeppson, "Learning tool MOSFET model: A stepping-stone from the square-law model to BSIM4," in *Proc. International Workshop on Power and Timing Modeling, Optimization and Simulation*, 39-44, 2013.
- [81] P. Rajeswari, G. Shekar, S. Devi, and A. Purushothaman, "Geometric programming-based power optimization and design automation for a digitally controlled pulse width modulator," *Circuits, Systems, and Signal Processing*, vol. 37, no. 9, 2018, pp. 4049-4064.
- [82] J. Xu and D. E. Root, "Advances in artificial neural network models of active devices," in *Proc. IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, Ottawa, Canada, pp. 1-3, 2015.
- [83] J. Wang, Y. -H. Kim, J. Ryu, C. Jeong, W. Choi and D. Kim, "Artificial Neural Network-Based Compact Modeling Methodology for Advanced Transistors," *IEEE Transactions on Electron Devices*, vol. 68, no. 3, 2021, pp. 1318-1325.
- [84] T. Liao and L. Zhang, "Efficient parasitic-aware hybrid sizing methodology for analog and RF integrated circuits," *Integration VLSI Journal*, 2018, pp. 301-313.
- [85] G. Shi, S. X.-D. Tan, and E. Tlelo-Cuautle, *Advanced symbolic analysis for VLSI systems*, Berlin: Springer, 2014.

- [86] X. Dong and L. Zhang, "EA-based LDE-aware fast analog layout retargeting with device abstraction," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 4, 2019, pp. 854-863.
- [87] Y. Wang, M. Orshansky, and C. Caramanis, "Enabling efficient analog synthesis by coupling sparse regression and polynomial optimization," in *Proc. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014.
- [88] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "Autockt: Deep reinforcement learning of analog circuit designs," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, pp. 490-495, 2020.
- [89] H. Wang et al., "GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, pp. 1-6, 2020.
- [90] M. Ahmadi and L. Zhang, "Analog Layout Placement for FinFET Technology Using Reinforcement Learning," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5, 2021.
- [91] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Cambridge: MIT Press, 2018.
- [92] H. Liang, W. Fu and F. Yi, "A Survey of Recent Advances in Transfer Learning," in *Proc. IEEE 19th International Conference on Communication Technology (ICCT)*, Xi'an, China, pp. 1516-1523, 2019.
- [93] J. Liu, M. Hassanpourghadi, Q. Zhang, S. Su and M. S. -W. Chen, "Transfer Learning with Bayesian Optimization-Aided Sampling for Efficient AMS Circuit Modeling," in *Proc. IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, San Diego, CA, pp. 1-9, 2020.
- [94] H. Wang et al., "GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, pp. 1-6, 2020.
- [95] M. Liu et al., "Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning," in *Proc. Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, pp. 496-501, 2020.

Appendix A: Published/Submitted Papers

- **Journal Papers Published or Accepted**

- [J3] Zhenxin Zhao and Lihong Zhang, "Automated Analog Integrated Circuit Topology Synthesis with Deep Reinforcement Learning," Accepted by *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, to appear in 2022.
- [J2] Zhenxin Zhao and Lihong Zhang, "Efficient Performance Modeling for Automated CMOS Analog Circuit Synthesis," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 11, 2021, pp. 1824-1837.
- [J1] Zhenxin Zhao and Lihong Zhang, "An Automated Topology Synthesis Framework for Analog Integrated Circuits," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, 2020, pp. 4325-4337.

- **Journal Papers Under Preparation**

- [JR1] Zhenxin Zhao and Lihong Zhang, "Transfer Learning for Automated Analog Integrated Circuit Synthesis," *under preparation for submission to IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

- **Conference Papers Published or Accepted**

- [C7] Zhenxin Zhao and Lihong Zhang, "Deep Reinforcement Learning for Analog Circuit Structure Synthesis," accepted by *2022 ACM/IEEE Design, Automation & Test in Europe Conference (DATE)*.
- [C6] Hohammad Hajijafari, Lihong Zhang, Mehrnaz Ahmadi and Zhenxin Zhao, "Reinforcement-Learning-based Mixed-Signal IC Placement for Fogging Effect Control," accepted by *2022 International Symposium on Quality Electronic Design (ISQED)*.
- [C5] Lian He, Zhenxin Zhao, Yuanzhu Chen and Lihong Zhang, "Advanced Transitive-Closure-Graph-Based Placement Representation for Analog Layout Design," in *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Glasgow, UK, pp. 1-4, 2020.
- [C4] Lian He, Zhenxin Zhao, Yuanzhu Chen and Lihong Zhang, "Placement with Sequence-Pair-Driven TCG for Advanced Analog Constraints," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, London, ON, Canada, pp. 1-4, 2020.
- [C3] Zhenxin Zhao and Lihong Zhang, "Deep Reinforcement Learning for Analog Circuit Sizing," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Sevilla, Spain, pp. 1-5, 2020.
- [C2] Zhenxin Zhao and Lihong Zhang, "Graph-Grammar-Based Analog Circuit Topology Synthesis," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, Japan, pp. 1-5, 2019.

- [C1] Zhenxin Zhao, Tuotian Liao, and Lihong Zhang, "Fast Performance Evaluation for Analog Circuit Synthesis Frameworks," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, pp. 1-5, 2018.