Deep Reinforcement Learning for Analog Circuit Structure Synthesis

Zhenxin Zhao and Lihong Zhang
Department of Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Canada
zz4376@mun.ca, lzhang@mun.ca

***Abstract*—This paper presents a novel deep-reinforcement-learning-based method for analog circuit structure synthesis. It behaves like a designer, who learns from trials, derives design knowledge and experience, and evolves gradually to eventually figure out a way to construct circuit structures that can meet the given design specifications. Necessary design rules are defined and applied to set up the specialized environment of reinforcement learning in order to reasonably construct circuit structures. The produced circuit structures are then verified by the simulation-in-loop sizing. In addition, hash table and symbolic analysis techniques are employed to significantly promote the evaluation efficiency. Our experimental results demonstrate the sound efficiency, strong reliability, and wide applicability of the proposed method.**

Keywords—deep reinforcement learning, hash table, analog circuit synthesis

I. Introduction

Continuous scaling of integrated circuit (IC) technologies and growing demands for high-performance low-power solutions result in manual analog circuit design becoming increasingly difficult [1]. Thus, efficient automated analog circuit design is in great need in order to improve design productivity, facilitate human laborious work, and enhance design accuracy. However, the automation of analog integrated circuit synthesis is still a challenging task with no widely accepted solutions yet. The circuit synthesis process includes two sub-processes, circuit structure (topology) synthesis [2] and circuit sizing [3]. Compared with circuit structure synthesis, a large amount of effort was devoted to circuit sizing research, contributing to a relatively developed area where commercial sizing tools have already been available for years. This fact has motivated us to focus on the frontier research of circuit structure synthesis.

Due to its intractable shortcomings, such as the poor generalization capacity, circuit structure synthesis through selection has gradually exited the stage two decades ago. Instead, circuit structure synthesis through generation, which automatically generates circuit structures from scratch, has attracted increasing research interests. The existing automated circuit structure generation works can be mainly divided into two categories: evolution-based [4] and graph-based [5]. However, both of them suffer from a common issue, that is, substantial computation effort would be wasted on generating and evaluating meaningless circuit structures. Instead of generating circuit structures from scratch, the structure reasoning methods [6] believe that the synthesis procedure is more like a decision-making process, where the distance to the predefined specification is calculated in each synthesis step and the decision is made based on the knowledge towards the right direction. However, even though it could efficiently produce reliable solutions, the extremely time-consuming knowledge mining process, which may need considerable expert intervention, is its severe shortcoming.

To address the drawbacks of the existing circuit structure synthesis methods, we propose a novel deep-reinforcement-learning (DRL)-based method, which integrates the merits of the structure generation and structure reasoning methods. DRL has been successfully applied in some EDA areas, such as analog circuit sizing [7] and analog layout placement [8]. Similar to the structure reasoning method, our proposed DRL-based method starts with a starting sub-circuit and gradually expands it with building blocks (BBs) via a decision-making process through selecting and executing actions. To improve the synthesis efficiency, hash table and symbolic analysis techniques are employed to significantly reduce the number of the produced structures to be sized during the synthesis process.

To the best of our knowledge, this is the first work that employs DRL to automate the structure 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 circuit structure generation methods.

II. The Proposed Framework

A. DRL Framework for Circuit Structure Synthesis

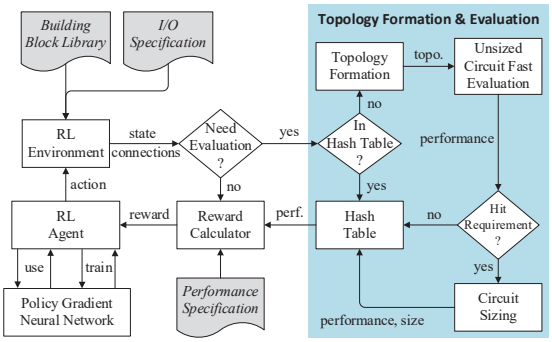


Fig. 1. The proposed DRL-based circuit structure synthesis framework.

As depicted in Fig. 1, in our work the automated circuit structure synthesis is realized through the proposed DRL-based framework. Trustworthy BBs are treated as the basic components to construct circuit structures, which are selected from a predefined building block library (PBBL) by taking actions. The input and output terminals of a BB 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 flow 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. 2 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.

The design specifications are composed of input-output (I/O) specification and performance specification. Among

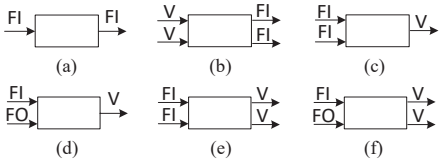


Fig. 2. Examples of the six types of BBs: (a) one-signal-path BB. (b) two-signal-paths BB. (c) identical-current converter with one output. (d) distinct-current converter with one output. (e) identical-current converter with two outputs. (f) distinct-current converter with two outputs.

them, the I/O specification works with the PBBL to form the specialized RL environment while the performance specification is utilized to calculate the reward. It is worth noticing that only a few states that satisfy certain conditions, which will be explained in detail later, need to be decoded into circuit structures for evaluation. The detailed evaluation process will be illustrated in Section III. For the states that do not meet the conditions, a reward will be directly assigned according to our defined reward function.

Due to the special environment of circuit structure synthesis, where circuit structure can only be evaluated after a valid structure has been constructed, the policy gradient method is employed in our work because it can wait until the end of an episode to calculate the reward. In policy-gradient-based DRL, the policy is modeled as a policy gradient neural network (PGNN), which takes a state as input and outputs the probability distribution over all actions.

B. State and Action

In the proposed DRL-based framework, states are in the representation of an array. The value at each slot in a state refers to the index of a BB in the PBBL if it does not equal zero. Otherwise, it means no BB (empty). In this way, each state encodes a sequence of BBs, making up a circuit structure. The size of a state is determined by the maximum number of BB allowed (a user-defined parameter) for constructing a circuit structure. An example state (state A) is given in Fig. 3(a). However, if the detailed connections among the represented BBs are unclear, one state may be able to be decoded into several circuit structures. For instance, state A may be decoded into two distinct three-stage OpAmps shown in Fig. 3(b_1) and Fig. 3(b_2). Thus, during the synthesis process, the connection information should be recorded.

In our proposed framework, taking action means selecting a BB from the PBBL to connect to the output terminal(s) of the current state-represented circuit structure. In order to largely avoid constructing meaningless circuit structures, we require that the connected terminals must be matched, which means that *V* must be matched with *V*, *FI* with *FO*, and vice versa. Based on the rules, the actions can be divided into three types: un-matched action, matched action, and terminational action. If any input terminal of the building block selected by an action cannot match with the output terminal that it is going to connect, it is an unmatched action. Otherwise, it is 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. Taking a terminational action would not select any BB to connect to the current structure but cause the termination of an episode.

C. Episode and Reward

The policy-gradient-based RL is carried out episodically. Each episode starts with a fixed initial state, which could be

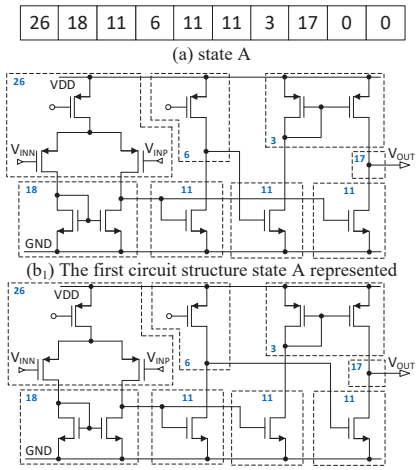


Fig. 3. An example state and its represented circuit structures.

an encoded BB or sub-circuit as long as its input terminal(s) meets the input specification. At each time step, the RL agent selects an action to take. After executing the action, the state turns into a new state, which has one more slot becoming non-zero. This process continues until this episode terminates.

An episode would terminate when any of the following three cases have appeared: 1) an unmatched action is taken; 2) state boundary is exceeded; 3) a terminational action is taken. Supposing *t* is the time step within an episode, and this episode terminates at the *T1*, *T2*, or *T3* time step due to the happening of the case 1), 2), or 3), respectively. The reward function $r(t)$ is accordingly defined as follow:

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

As the reward function indicated, all the non-terminal steps would receive a reward 0, while for the terminal step, a positive or negative reward is assigned to encourage or discourage the generation of corresponding circuit structure, respectively. For the terminations that occurred at *T3*, the constructed circuit structure has to be evaluated. If its performance meets the target specification, +1 is assigned. Otherwise, -1 is assigned. Furthermore, in order to speed up the learning process, when the terminations occur at *T3* and the evaluated performance meets the target specification, we go back to modify the reward received by taking the second last action to be +5 after completing this episode.

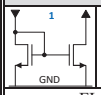
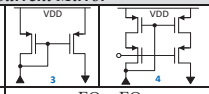
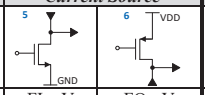
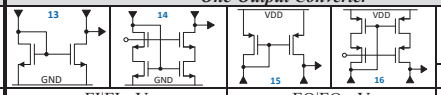
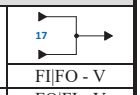
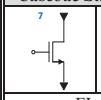
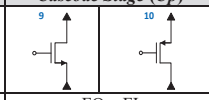
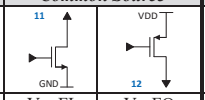
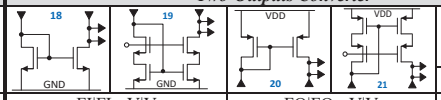
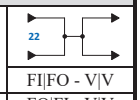
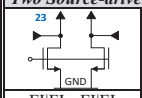
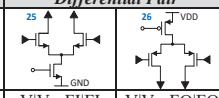
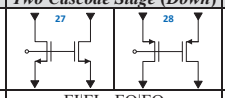
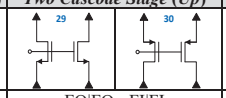
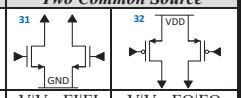
III. Circuit Structure Formation & Evaluation

The detailed evaluation process is illustrated in the shaded block in Fig. 1. 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 results are fetched and returned to the reward calculator. Otherwise, they need to be first decoded into a circuit structure 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.

A. Hash Table

The state to be evaluated is first calculated its hash via the

TABLE I. 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 Source-driven Current Splitter		*Differential Pair*		*Two Cascode Stage* (*Down*)		*Two Cascode Stage* (*Up*)		*Two Common Source*		
										
FI|FI - FI|FI	FO|FO - FO|FO	V|V - FI|FI	V|V - FO|FO	FI|FI - FO|FO		FO|FO - FI|FI		V|V - FI|FI	V|V - FO|FO	

following formula:

$$h(state) = \prod_{i=0}^{n} prime[j] \ \ mod\ (2^{32} - 5) \qquad (2)$$

where *i* is the index of a slot in *state*, *n* is the number of non-zero slots in *state*, *j* is the slot value in *state*, *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. The states and their evaluated performance results are put into buckets in the hash table according to their hash values.

B. State Decoding

Only when a terminational action is taken, the current state needs to be decoded and evaluated. The state to be decoded is firstly mapped to its represented BBs. Then transistor-level circuit structures will be formed by connecting those BBs according to the recorded connection information. Since simply swapping input pins of an analog circuit may affect its performance, we treat the circuits, which have exactly the same structure but swapped input pins, as different circuit structures. Thus, the decoded circuit structure should create a copy of itself with swapped input pins.

C. Circuit Structure Evaluation

The decoded circuit structures will first be fast evaluated through symbolic analysis. In our proposed work, the GPDD algorithm is employed to numerically calculate the DC gain of an unsized circuit [9]. 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 structures with very bad performance will not proceed to the subsequent sizing phase.

Once the circuit structures pass the fast evaluation filter, they have to be sized to check their performance feasibility. In this work, we employ the NSGA-II to size circuits, which uses 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 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 performance and size to the hash table. Otherwise, after executing all 50 generations, the best performance achieved so far and the corresponding size 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 utilize the parallel computing technique to size the circuits.

IV. EXPERIMENTAL RESULTS

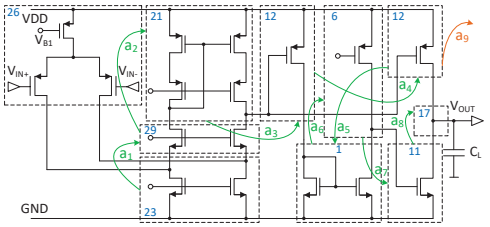
A. Predefined Building Block Library (PBBL)

As shown in TABLE I, our defined PBBL contains 32 BBs, and each BB has a unique index. All our experiments were carried out based on this PBBL. 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 noticing that the two special nets-contained-only BBs both have two possible combinations of I/O terminals, which are completely dependent on the order of the input terminals. In addition, in order to facilitate the job of evaluating the produced circuit structures, we require that except for differential pairs, all the transistors within a BB have the same length and width.

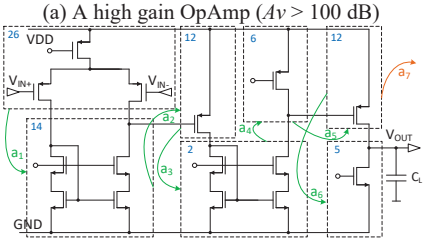
B. Analysis of Circuit Structure Synthesis

In our experiments, the state size and action size were set to be 10 and 20, respectively. For the PGNN, there are two hidden layers, with the first and second layers containing 300 and 200 neurons, respectively. The learning rate α, discount factor γ, and batch size were set to be 0.002, 0.95, and 200, respectively. The SPICE simulation was conducted in a CMOS 65nm technology process. The starting sub-circuit (initial state) was set as a differential pair made of PMOS transistors (i.e., the 26th BB). The I/O specification was two voltage inputs and one voltage output.

Fig. 3(b_1) and Fig. 3(b_2) have already depicted the circuit structures synthesized by our proposed framework. Since high-gain OpAmps and high-bandwidth OpAmps always attract more academic and industrial interests, in this part we will show that these two categories of circuits can be readily synthesized by our proposed work with distinct performance specification settings. In our experiments, the high gain OpAmps are synthesized with the specification of 100dB *Av*, 60° *PM*, 10dB *GM*, and 10MHz *UGB*, while the high bandwidth OpAmps are generated with the specification of 60dB *Av*, 60° *PM*, 10dB *GM*, and 1GHz *UGB*. Fig. 4(a) depicts a synthesized high gain circuit with *Av* larger than 100 dB, which is actually a three-stage OpAmp. Fig. 4(b)



(a) A high gain OpAmp (Av > 100 dB)



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

Fig. 4. Example of synthesized circuits.

illustrates a high unity-gain-bandwidth circuit synthesized by the proposed work, which contains seven BBs and needs seven actions to build it. It can easily reach more than 1 GHz *UGB* with power consumption around 0.6 mW, which demonstrates high efficacy of our proposed methodology.

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

In this part, we will show some comparisons with other state-of-the-art circuit structure synthesis methods. Among these methods, the work of [6] utilizes design knowledge mining and reasoning (KMR) to synthesize circuit structures while MOJITO [4], FEATS [5], and GCTG [10] use genetic programming, graph composition, and graph decomposition methods to generate circuit structures, respectively.

As claimed as one of the features of our proposed work, the design search space is flexible and controllable, which is realized by extending PBBL or changing state size. FEATS and GCTC have a similar feature by extending the defined BB library or changing the maximum number of BBs allowed for synthesis. However, there is no means for MOJITO to control its design search spaced, 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.

Due to the stochastic-driven synthesis, MOJITO needs to evaluate around 101,904 structures in the process to finally produce 512 unique structures, which took 7 days. KMR has to go through a quite timing-consuming data mining process because the similarity of circuits is extracted by comparing them in pairs. Thus, both of them suffer from almost unaffordable computation efforts to synthesize circuits. TABLE II illustrates the comparison among FEATS, GCTC, and our proposed DRL by using the same BB library as listed in TABLE I (i.e., the PBBL) and the same I/O specification. As one of the table headers in TABLE II, structure 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 II are the average outputs from ten runs of our proposed algorithm. Each algorithm run would take 2 to 9 hours to complete the learning process.

TABLE II. COMPARISON AMONG FEATS, GCTG, AND DRL

Structure Size	Number of the Unique Structures 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 structure size is 5, FEATS and GCTC generate around 1,000 unique circuit structures after applying their isomorphism checks, whereas our proposed DRL only produces 20 unique circuit structures during the synthesis process. When the structure 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. The reason for it is that both FEATS and GCTC build circuit structures in the brute-force explorative way. However, this structure explosion issue can be readily addressed by our proposed DRL methodology. As shown in TABLE II, the number of the unique circuit structures generated by DRL only increases a bit when the structure size grows.

V. CONCLUSIONS

In this paper, we presented a novel DRL-based method to synthesize analog ICs 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 has wide applicability to handle distinct input-output and performance specifications. Thanks to the simulation-in-loop involvement, it has the ability to verify the output solutions. Compared with the other state-of-the-art circuit structure synthesis methods, it can not only address their commonly known shortcomings, but also achieve the least computation effort.

ACKNOWLEDGMENT

This work was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), Provincial Government of Newfoundland and Labrador, and Memorial University of Newfoundland.

REFERENCES

[1] Z. Zhao and L. Zhang, "Efficient performance modeling for automated CMOS analog circuit synthesis," in *IEEE Trans. on VLSI Systems*, vol. 29, no. 11, pp. 1824-1837, Nov. 2021.

[2] Z. Zhao and L. Zhang, "Graph-grammar-based analog circuit topology synthesis," in *Proc. ISCAS*, pp. 1-5, 2019.

[3] T. Liao and L. Zhang, "An LDE-aware gm/ID-based hybrid sizing method for analog integrated circuits," in *IEEE Trans. on CAD of Integrated Circuits and Systems,* vol. 40, no. 8, pp. 1511-1524, 2021.

[4] 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," in *IEEE Trans. on Evolutionary Computation,* 15(4), pp. 557-570, 2011.

[5] M. Meissner and L. Hedrich, "FEATS: Framework for explorative analog topology synthesis," in *TCAD,* vol. 34, no. 2, pp. 213-226, 2015.

[6] F. Jiao, et al., "Analog circuit design knowledge mining: discovering topological similarities and uncovering design reasoning strategies," in *TCAD,* vol. 34, no. 7, pp. 1045-1058, 2015.

[7] Z. Zhao and L. Zhang, "Deep reinforcement learning for analog circuit sizing," in *Proc. ISCAS*, pp. 1-5, 2020.

[8] M. Ahmadi and L. Zhang, "Analog layout placement for FinFET technology using reinforcement learning," in *Proc. ISCAS*, 2021.

[9] Z. Zhao, T. Liao and L. Zhang, "Fast performance evaluation for analog circuit synthesis frameworks," in *Proc. ISCAS*, pp. 1-5, 2018.

[10] Z. Zhao and L. Zhang, "An automated topology synthesis framework for analog integrated circuits," in *TCAD*, 39(12), pp. 4325-4337, 2020.