An Automatic Circuit Design Framework for Level Shifter Circuits

Jiwoo Hong, *Graduate Student Member, IEEE*, Sunghoon Kim, *Graduate Student Member, IEEE*, and Dongsuk Jeon, *Member, IEEE*

***Abstract*—Although design automation is a key enabler of modern large-scale digital systems, automating the transistor-level circuit design process still remains a challenge. Some recent works suggest that deep learning algorithms could be adopted to find optimal transistor dimensions in relatively small circuitry such as analog amplifiers. However, those approaches are not capable of exploring different circuit structures to meet the given design constraints. In this work, we propose an automatic circuit design framework that can generate practical circuit structures from scratch as well as optimize the size of each transistor, considering performance and reliability. We employ the framework to design level shifter circuits, and the experimental results show that the framework produces novel level shifter circuit topologies and the automatically optimized designs achieve 2.8×–5.3× lower power-delay product (PDP) than prior arts designed by human experts.**

***Index Terms*—Circuit design automation, deep learning, evolutionary algorithm, level shifter, reinforcement learning (RL).**

I. Introduction

With increasing hardware design complexity and variability of the fabrication process, design automation has been widely adopted in a large portion of the IC design process. For instance, various electronic design automation (EDA) tools are now available for designing digital blocks and System-on-Chips (SoCs). The EDA tools can generate a large block composed of millions of logic gates very efficiently using a standard cell library[1]. However, when it comes to designing integrated circuits, design automation remains a challenge. Most digital and analog circuits are still carefully designed by human experts due to high design complexity and reliability concerns [2].

Integrated circuit design automation can be decomposed into two design problems: 1) circuit topology selection and 2) transistor size optimization. It is crucial to choose a proper circuit topology in the first place since the topology mainly sets the limit on the performance and reliability a circuit can achieve. We also need to optimize the size of each transistor to realize its true potential. Various approaches have been reported for automatic circuit topology generation. For digital logic gates, the Boolean expression factoring method that generates series-parallel (SP) associations of transistors for the given function was suggested [3]. Possani *et al.* [4] proposed an improved graph-based method that creates a logic gate by introducing nonseries-parallel (NSP) arrangements into the SP structure, thus reducing the number of transistors. But these methods regard transistors as ideal switches and, hence, are only applicable to designing digital logic gates based on static operations.

There have also been several circuit topology synthesis approaches aimed at more general integrated circuits. The library-based methods [5], [6] select one of the predefined circuit structures (e.g., a two-stage amplifier) in the library based on the desired operating characteristics. However, one must construct a library containing all possible circuit structures in advance, which is a time-consuming process that also necessitates a considerable amount of human effort. Building-block-based methods [7]–[9] take a similar approach, but rely on a library of smaller building blocks, such as a current mirror and a differential input pair. They employ various algorithms to search for the best topology, such as the multiobjective evolutionary algorithm [7], framework for explorative analog topology synthesis method (FEATS) [8], and graph-grammar-based topology generation (GGTG) [9]. Since the library- or building-block-based approaches have relatively limited search space, they are suitable for the fast generation of integrated circuits using a well-established topology. However, the search space is constrained within the predefined set of circuit structures or building blocks and, hence, they are less adaptive to changes in design parameters or fabrication process. In addition, there is little possibility that they could generate a novel topology that has not been studied yet.

On the other hand, the transistor-based methods [10]–[13] do not rely on predefined components for topology generation; instead, they progressively construct a circuit by adding or removing a transistor in the topology. For instance, the circuit-constructing robot (CC-BOT) [11] starts with a single node and conditionally adds a transistor following an evolutionary algorithm. An active bot moves to a newly created node and continues adding transistors from there. The algorithm in [13] represents transistors and passive devices as a 3-node graph (*hypergraph*) and an edge, respectively. In each generation, it removes and adds multiple hypergraphs and edges, also following an evolutionary algorithm. The transistor-based

Manuscript received 25 June 2021; revised 10 December 2021; accepted 10 February 2022. Date of publication 1 March 2022; date of current version 22 November 2022. This work was supported in part by the National Research Foundation of Korea under Grant NRF-2020R1A4A4079177 and Grant NRF-2019R1C1C1004927, and in part by the IC Design Education Center. This article was recommended by Associate Editor P. Li. *(Corresponding author: Dongsuk Jeon.)*

The authors are with the Graduate School of Convergence Science and Technology, the Research Institute for Convergence Science, and the Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea (e-mail: djeon1@snu.ac.kr).

Digital Object Identifier 10.1109/TCAD.2022.3155444

approaches have a significantly larger search space and, as a result, are capable of generating an optimal circuit topology under different design constraints. These approaches do not require much prior knowledge on the target circuit, removing the need for the aid of human experts during the design process. However, they essentially rely on trial and errors; an inefficient search algorithm results in very slow search speed, requiring an extensive amount of SPICE simulations to evaluate candidate integrated circuits.

Transistor sizing is another crucial part of integrated circuit design automation since it directly affects the performance and reliability of a circuit. Liu *et al.* [14] proposed the multiobjective uncertain optimization with ordinal optimization LSS and parallel computation (MOOLP) optimizer based on a differential evolutionary algorithm, whereas other prior works suggest using particle swarm optimization [15], [16] or Bayesian methods [17], [18]. Recent works demonstrate promising results by applying deep learning algorithms to transistor sizing optimization. For instance, learning to design circuit (L2DC) [19] and AutoCkt [20] adopt reinforcement learning (RL) for optimizing transistor sizes in analog amplifiers. It was demonstrated that those RL-based approaches could successfully optimize the integrated circuit to meet the given design constraints, such as gain, bandwidth, and input-referred noise. While the RL-based methods achieve significantly faster convergence than conventional optimization algorithms, they still need numerous SPICE simulations during optimization. Also, both L2DC and AutoCkt utilize prior knowledge on the circuit topology during optimization (e.g., the signal path and tightly coupled transistors), limiting their applicability to other types of integrated circuits. To address these problems, Wang *et al.* [21] employed a graph convolutional network in RL (GCN-RL) and utilize transfer learning. They show that using a pretrained network can reduce the number of SPICE simulations in optimizing two-stage and three-stage transimpedance amplifiers. However, the initial training of the neural network still requires a large number of SPICE simulations, and the pretrained network is only effective when applied to another circuit with a similar structure.

Level shifter circuits are widely used in digital systems to convert the level of signals between voltage domains. The signals from the internal core must be boosted to communicate with external discrete components or the data between core blocks with different supply voltages should be level converted for proper operation. Various level shifter circuit topologies have been studied, and the optimal topology can vary greatly depending on the operating conditions, such as voltage conversion range (e.g., core to core or core to I/O) and power budget. For instance, differential cascode voltage switch (DCVS) exhibits better conversion speed and energy efficiency for conversion between core voltage domains, whereas the Wilson current mirror level shifter (WCMLS) and its variant are suitable for converting subthreshold voltage input due to relaxed contention [22]–[24]. Therefore, we may have to switch to a totally different topology and start the design process again when the design constraints and operating conditions change, making conventional circuit design automation frameworks unsuitable for level shifter design.

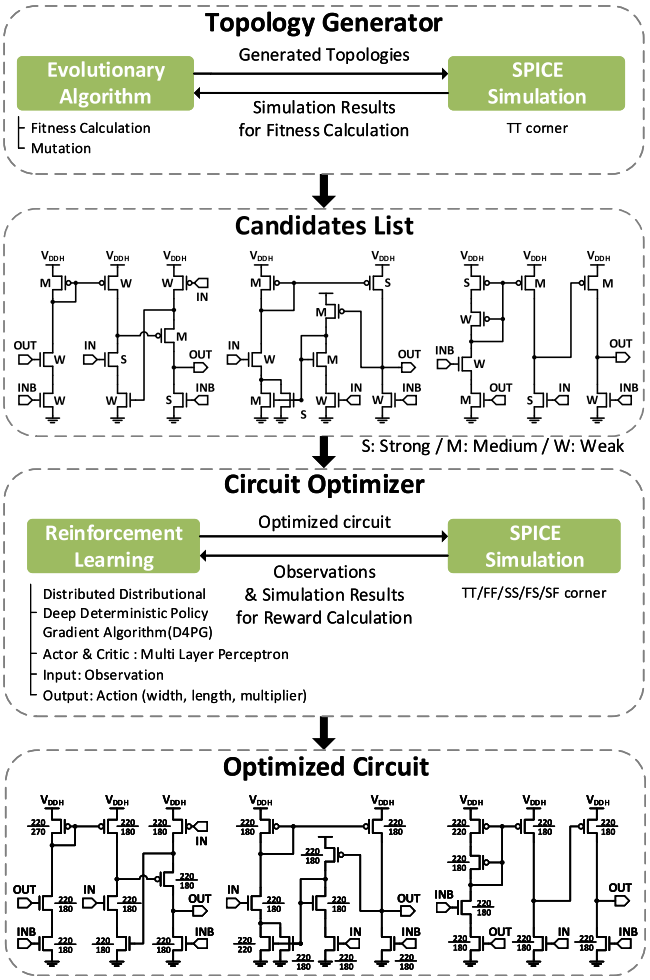


Fig. 1. Overview of the proposed circuit design framework.

In this work, we propose a unified circuit design automation framework that can generate an optimal circuit topology from scratch as well as optimize the size of each transistor. Our key contributions are as follows.

1) A 2-stage circuit design framework that significantly speeds up the design process.
2) A new voltage-based graph representation of integrated circuits.
3) A fast circuit optimizer adopting a multiagent RL algorithm for faster convergence.
4) A process variation-aware optimization algorithm that results in a practical, robust design.

The framework was employed to design a level shifter circuit, and the resulting level shifter circuits are fabricated in a 180-nm CMOS process to validate the effectiveness of the proposed circuit design framework.

The remainder of the article elaborates on the proposed framework as follows: Section II describes the overall architecture of the framework and its distinct features. Section III discusses the experimental results, and Section IV concludes the article.

II. Proposed Circuit Design Framework

The overall flow of the proposed circuit design framework is shown in Fig. 1. Instead of relying on a single algorithm to

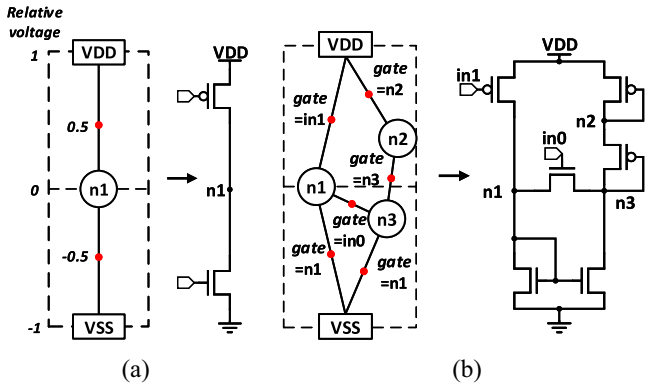


Fig. 2. Examples of proposed graph-based circuit representation. (a) Simple circuit with a pair of MOSFET devices. (b) Complex circuit where gates are connected to other nodes.

design a circuit, we propose to split the design process into two distinct stages. The first stage (*topology generator*) employs an evolutionary algorithm to search for candidate circuit structures quickly. The second stage (*circuit optimizer*) performs an RL-based transistor size optimization on the generated integrated circuits to maximize performance, while guaranteeing reliable operation under process variations. Each stage is described in detail in the following sections.

A. Topology Generator

In the topology generator, we represent each circuit topology as a graph and employ a graph generation algorithm to obtain candidate circuit structures. The graph-based method in [4] gives an example of expressing digital circuits as a graph, where pull-up and pull-down networks are generated separately, and each transistor corresponds to an edge in the graph. The two nodes connected by an edge define the source and drain, whereas the gate connection is defined as one of the node properties. However, this approach is not applicable to other types of integrated circuits that do not have separate pull-up and pull-down paths, where N-channel and P-channel MOSFET devices can be placed more arbitrarily. Therefore, we propose a generalized graph representation method suitable for a broader range of integrated circuits shown in Fig. 2. In our representation method, an edge (transistor) has *gate* and *size* properties, representing the net connected to the gate and transistor size. A Node has a *type* property that represents the net type (e.g., input port, output port, supply, ground, and internal net). Additionally, we introduce a new property *voltage* in the nodes. This property represents a relative voltage of each node and has a range of $[-1, 1]$. The voltage of an edge is obtained by averaging the voltages of the nodes on both ends (source and drain). The edges with positive voltage translate to P-channel MOSFET devices, whereas the edges having zero or negative voltage represent N-channel MOSFET devices. This method allows for generating more generalized circuit structures while preserving a common circuit property that P-channel MOSFET devices are typically placed near the power supply voltage, whereas N-channel MOSFET devices are biased at lower voltages to maximize operation range.

Algorithm 1 Topology Generator

Input: Population size N, Max Generations G, Mutation Probability
Output: Candidate Topologies

```
P: Population, C: Offspring, P_0: Initial Population
for g = 1, 2, ..., G do
    Simulate all C_i(i ∈ N) and Calculate Fitness
    Remove Stagnated Species and Extract Candidates
    Calculate Fitness of Species s_k ∈ S_g
    Calculate Reproduction Size R_k of each s_k
    for all s_k in S_g do
        Add Best Candidate in s_k to P_{g+1}
        Make Parent Pool with N' Top Candidates in s_k
        for j = 1, 2, ..., R_k do
            Crossover
            Mutate
            Add C_j to P_{g+1}
        end for
    end for
    Speciate P_{g+1}
end for
```

Since we aim to generate an optimal circuit topology without prior knowledge, we suggest employing an evolutionary algorithm in the topology generator. NeuroEvolution of Augmenting Topologies (NEAT) is a widely used evolutionary algorithm for exploring artificial neural network structures [25]. The algorithm starts from a simple network with a single fully connected layer, and the network evolves into more complex structures through crossover and mutation over generations. We modify the NEAT algorithm to make it suitable for circuit topology generation; we introduce the voltage concept into the NEAT algorithm, and the mutation functions and the properties of genes are heavily modified aimed at circuit topology generation.

Algorithm 1 details the proposed circuit topology generation algorithm. An offspring represents a candidate circuit topology and has node and connection genes. The node gene represents a node in the graph and has *type*, *voltage* and *innovation number* properties. The *type* property determines the type of the nodes (input port, output port, supply, ground, or internal net) and the *voltage* property represents its relative voltage, whereas the *innovation number* is a unique identifier. The connection genes represent edges in the graph with *in*, *out*, *size*, *gate*, and *innovation number* properties. The *in* and *out* properties define two endpoints of the edge (source and drain of the transistor), and the *size* property defines the relative strength of the transistor. As described above, since a transistor is a three-terminal device, the node to which the gate of the transistor is connected is defined by the *gate* property. The *innovation number* is a unique identifier. A population is a set of all offspring of the current generation. The population is divided into several species based on similarity. Each species has a base model, and the offspring close to the base model are included in the species.

The topology generator first creates an initial population P_0 which consists of offspring with only three node genes and

two connection genes: V_{DD}, V_{SS}, and an internal node connected by one *P*-channel MOSFET & *N*-channel MOSFET pair [Fig. 2(a)]. The gate of each transistor is randomly connected to the node except V_{DD} and V_{SS}. In each generation, evolution begins by calculating the fitness of the species in the current population. The algorithm converts each offspring into a netlist and runs a SPICE simulation to calculate the offspring's fitness based on the observed functionality and performance. Then, the fitness of the offsprings included in the species is averaged to obtain the species fitness. The number of offspring that can reproduce from each species to the next generation is determined in proportion to the species fitness. During circuit topology generation, simulations are performed only at the typical (TT) corner to scan large search spaces and find the best candidates quickly.

Before reproducing a new population, the algorithm observes whether the fitness of the best offspring in each species has been improved or not in the last few generations. If the fitness of a species does not improve any further for a certain number of generations, then the species is considered stagnant, and offsprings of that species are removed from the population. The evolution process is independently performed for each species. First, the offspring with the highest fitness in each species is automatically included in the population of the next generation. Next, a set of offspring with the highest fitness within each species is selected as a parent pool. Two offspring are randomly chosen from the pool and compared with each other, where the winner evolves through mutation and joins the population of the next generation.

The fitness function represents the performance and reliability of a circuit as a single value. We consider two types of design constraints for fitness calculation: 1) hard constraints and 2) soft constraints. The hard constraints are the set of design constraints that a circuit must satisfy (e.g., rail-to-rail output swing for level shifters), whereas the soft constraints indicate the design quality (e.g., power consumption and conversion delay of level shifters). The fitness of an offspring at the xth generation is calculated as

$$\text{fit}_x = \sum_{i\in H} \alpha_i f(q_{i,x}) + \left\{\prod_{i\in H} f(q_{i,x})\right\}\left\{\sum_{j\in S} \alpha_j f(q_{j,x})\right\} \quad (1)$$

where fit_x is the calculated fitness of an offspring, $q_{i,x}$ is the observed performance of the circuit in SPICE simulations corresponding to the ith constraint, $f(q_{i,x})$ is the score function for each constraint, α_i is the weight of the ith constraint, and H and S represent the sets of hard and soft constraints, respectively. This is similar to the reward function used in RL for circuit optimization in [19], but our approach has two distinct differences: 1) we use $\log(q_{i,x})$ instead of $q_{i,x}$ for the scores that have a large dynamic range and 2) the contribution of soft constraints in the fitness is regulated by the scores related to the hard constraints, instead of using a hyper-parameter manually tuned for a specific type of circuit. In early generations, it is highly likely that most offspring would fail to function properly. The scores related to the hard constraints would be very low, making the fitness largely dictated by the hard constraints. Hence, the algorithm focuses on finding feasible topologies that produce a desired output. Once the algorithm finds properly working circuit topologies, the scores related to the hard constraints saturate and do not affect the fitness. The remainder of the evolution process further modifies the topology to improve circuit performance.

The topology generator employs various mutation functions so that it can cover a wide range of circuit topologies. Note that the nodes without any connection (i.e., floating nodes) can be generated as a result of mutation. Hence, we label the nodes with one or more connections as active nodes, and only active nodes are selected for mutation. The types of mutations are discussed as follows.

1) Add Connection: This mutation randomly chooses two active nodes and connects them by adding a new edge. Since an edge corresponds to a transistor in the actual circuit, it links the gate of the new edge to one of the existing active nodes by updating the *gate* property.

2) Add Node: This inserts a new node in one of the edges. In other words, a single transistor is replaced with two stacked transistors. The gates of the stacked transistors are connected to the same node to which the gate of the original transistor was connected. This process is often used when designing a circuit to increase output resistance or minimize leakage current.

3) Add P-Channel MOSFET and N-Channel MOSFET: Pair A *P*-channel MOSFET and *N*-channel MOSFET pair makes a new connection between V_{DD} and V_{SS}. If a single *P*-channel or *N*-channel MOSFET transistor is placed between V_{DD} and V_{SS}, this will be just a current leaking path. Therefore, we place transistors as a pair of *P*-channel and *N*-channel MOSFET devices when making a new connection between the supply rails.

4) Change Gate: The gate of a transistor is connected to a different active node except for V_{DD} and V_{SS} nodes.

5) Remove Connection: This mutation randomly removes one of the connections, which allows for removing transistors from the current topology. This prevents the circuit from continuously becoming larger.

6) Change Size: The size of the connection genes represents the relative size (strength) of a transistor. Since our goal is to quickly go through a variety of circuit topologies and find promising candidates, we define each transistor's strength in only three steps: 1) strong; 2) medium; and 3) weak. During mutation, the transistor size randomly changes in each connection gene independently.

7) Change Output Port: This mutation changes the location of the output port. One of the active nodes is selected as an output.

In the original NEAT algorithm, each mutation function is randomly selected in each mutation. Hence, multiple types of mutations may be performed simultaneously. However, this may result in an excessive amount of change in a circuit. For instance, removing a transistor from the circuit and changing the gate connection of another transistor would produce a circuit with entirely different characteristics. Hence, we limit the mutation process to select only one of the add, remove or gate change mutations (mutations 1 through 5 above). In addition, other minor mutations (mutations 6 and 7) are independently

introduced with a certain probability. Let P_{addNode}, P_{addCon}, P_{addPair}, $P_{\text{changeGate}}$, and P_{rmCon} denote the probability of mutations 1 through 5 above. Then, the mutation process follows the equation below:

$$P_{\text{addNode}} + P_{\text{addCon}} + P_{\text{addPair}} + P_{\text{changeGate}} + P_{\text{rmCon}} = 1 \quad (2)$$

During topology exploration, we do not want the algorithm to keep adding transistors indefinitely. Otherwise, the number of transistors in a circuit may explode, and the resulting circuit would be far from what we desire. For instance, an ideal analog amplifier or level shifter circuit typically has tens of transistors at most. Therefore, we balance the expected number of removed and added transistors in each mutation by enforcing the relationship as follows:

$$2P_{\text{addNode}} + P_{\text{addCon}} + 2P_{\text{addPair}} - P_{\text{rmCon}} = 0 \quad (3)$$

since adding a node (a net in the circuit) adds two transistors, whereas adding or removing a connection adds or removes a single transistor in the circuit.

After a new generation is obtained by mutating all the offspring of the current generation, the newly generated offspring are grouped again into a set of species. Each offspring is compared to the base models of existing species. If the number of differences in the connection genes is below the threshold for one or more existing species, then the offspring joins the closest species. Otherwise, the offspring constitutes a new species and becomes its base model. After the grouping process is done, the population undergoes another iteration of the mutation process to obtain the next generation. This process continues until it reaches the maximum number of generations defined by the user.

The topology generator selects candidate topologies both during and at the end of the evolution. When a stagnant species is removed during evolution, the offspring with the best fitness in that species is selected and added to the candidate list if it meets all the given design constraints. When the algorithm finishes the last iteration, the same operation is performed on all the remaining species. Note that there may exist floating nodes and floating paths as a result of mutation. Before adding an offspring to the candidate list, the topology generator finds and removes the floating nodes and paths.

B. Circuit Optimizer

The topology generator is aimed at quickly finding promising circuit topologies. Hence, each transistor is only roughly sized during exploration (e.g., strong, medium, or weak). This accelerates the search process by significantly limiting the search space, but the size of each transistor must be further tuned for optimal performance. For this purpose, we employ an additional circuit optimizer as the second stage in the proposed circuit design framework.

The circuit optimizer adopts a RL algorithm to optimize candidate integrated circuits. Various RL algorithms have been used for circuit optimization. L2DC [19] and GCN-RL [21] are based on deep deterministic policy gradient (DDPG) [26], and AutoCkt [20] adopts proximal policy optimization (PPO) [27]. DDPG has an actor–critic structure and generally works well in continuous or high-dimensional action spaces. An agent collects and saves a sample into a replay memory. Then, a minibatch is randomly selected from the replay memory to train the network. While PPO also has an actor–critic structure suitable for training in continuous or high-dimensional action space, it does not have a replay memory. Instead, N agents collect samples in parallel during an episode which consists of T time steps, and a minibatch is constructed using the collected samples and used for training the algorithm. Then, all the samples are discarded. DDPG exhibits slower convergence during training since it only uses one agent contrary to PPO, but has the advantage of being able to reuse the samples stored in the replay memory. PPO trains the model more quickly by using multiple agents, but it only uses the samples collected in the current episode for training, which reduces sample efficiency. In circuit optimization, samples are obtained by running time-consuming SPICE simulations. Therefore, it is crucial to maximize sample efficiency (i.e., reduce the number of samples required for algorithm convergence) to speed up the circuit optimization process. To resolve this issue, we adopt distributed distributional DDPG (D4PG) [28] algorithm in the circuit optimizer. D4PG supports both multiagent training and sample reuse by using a replay memory. Unlike DDPG and PPO which express future rewards as a single scalar value, D4PG expresses rewards as a probability distribution. It models the inherent uncertainty imposed by function approximation in a continuous environment, resulting in better gradients and improving the training performance compared to DDPG. It also shows more stable performance when multiple agents are used [28].

In the RL algorithms using actor–critic structure, two different neural networks are typically employed: an actor network and a critic network. The actor network takes a state vector as an input and produces an action vector, whereas the critic network takes state and action vectors as inputs and predicts the reward value an agent is expected to receive as a result of the current and future actions. The RL algorithm trains those neural networks on the observed samples. As the complexity of the neural network increases with the dimension of input vectors, it is important to minimize the dimension of the input vector for faster optimization. Since the action vector represents relative size changes of all the transistors in the circuit, its dimension is fixed. Hence, we aim to optimize the critic network by reducing the dimension of the state vector. Specifically, we use the simulated circuit performance (e.g., power consumption and delay) and area as a state, instead of feeding each transistor's size or other characteristics (e.g., V_{th}, V_{sat}, and μ_0) as did in prior works [19]–[21]. Therefore, the dimension of the state vector is independent of the number of transistors in the topology and the optimization process can be efficiently accelerated when the target circuit topology consists of many transistors.

The actor network creates an action based on the state obtained by SPICE simulations. An action represents a relative change in the size (width, length, and multiplier) of each transistor. If the target topology has N transistors in total, the dimension of the action vector would be $3N$. Each dimension of the action vector has a value in $[-1, 1]$. Then, the amount

of change in the size of the ith transistor is

$$\Delta\text{Size} = \text{round}\left(\text{Action} \cdot \frac{\text{size}_{\max} - \text{size}_{\min}}{L_{\text{maxStep}}}\right) \tag{4}$$

where ΔSize is the amount of change in transistor size, Action is the output of the actor network, L_{maxStep} is the number of steps in one episode, and $\text{size}_{\max}$ and $\text{size}_{\min}$ are the allowable maximum and minimum transistor sizes. This translates to the maximum size change that can occur in one episode equal to $\text{size}_{\max} - \text{size}_{\min}$. The size values are real numbers, so they are rounded to the closest values allowed in the given process before converted to an actual circuit.

In D4PG, when a sample collected by one of the agents is stored in the replay memory, a minibatch is created by randomly choosing samples from the replay memory. However, as the training progresses, the amount of samples stored in the memory becomes larger; thus, only a fraction of stored samples is used to generate a minibatch, reducing sample efficiency. In addition, the learner updates the network only once in each time step, and the SPICE simulation to obtain a new sample becomes the processing bottleneck. To address these issues, we propose to adopt a multiupdate technique that has been used for unbiased learning. When a sample is obtained by the agent and stored in the replay memory, unlike the conventional method of circuit optimization that creates one minibatch from the stored samples, we create several minibatches and update the critic and actor networks multiple times. This accelerates the circuit optimization process without time overheads since multiple updates could be performed while SPICE simulations are running. This scheme also allows for unbiased learning through random sampling that removes correlation between minibatches, reducing the possibility of overfitting.

At the beginning of training, the actor network tends to generate the same action even if the state changes gradually in each step. In other words, the size of a transistor continues to increase or decrease regardless of the current state. This is because the output is close to either 1 or -1 in most cases when the actor network weights are randomly initialized. The actor network typically uses the tanh function as the activation function. In a randomly initialized network, the output of the network, which is the input to the final tanh activation function, typically has an absolute value of 2 or larger, rendering the final output close to ± 1. This effect is amplified by the fact that circuit performance is converted to a state using a logarithmic function. Even if the state changes, the sign of the action which determines size change direction (increase or decrease) is likely to stay the same. In addition, the weights of the actor network in each agent are updated only when an episode ends, and they remain fixed for all the steps within an episode. Therefore, in the first few episodes, the sizes of many transistors just move to the minimum or maximum value. This severely hinders circuit optimization by moving the design far from the initial point, which is already a near-optimal design found in the circuit topology generator. To solve this problem, we propose an episode early stopping technique that limits the number of steps in an episode in the early stage of training. As the learning progresses, it gradually increases the number of steps in each episode, and the episode finally proceeds with the maximum number of steps defined by a hyperparameter. This technique allows the network to learn more stably while acquiring more meaningful samples near the initial point in early episodes.

Algorithm 2 Circuit Optimizer

Learner

Input: Number of Steps in Episode N, Batch Size M, Replay Memory Size R, Learning Rates α_0 and β_0, Multi-Update Parameter U

```
Determine Network Size by Analyzing Netlist
Initialize Network Weights with Kaiming Initialization
for i = 1, 2, ..., N do
    Wait for Samples from Agents
    for j = 1, 2, ..., U do
        Randomly Choose M Samples from Replay Memory
        Compute Updates of Actor and Critic Networks Using Samples
        Update Network Parameters
    end for
end for
```

Agent

Input: Number of Steps in Episode N, Number of Actors P, Episode Early Stopping Interval T

```
repeat
    Initialize Episode
    Copy Actor Network from Learner
    for step = 0, ..., K do
        Get Action from Actor Network and Change Size
        Simulate and Calculate State (s) and Reward (r)
        Send Sample to Learner
    end for
    Increase K every T Episodes
until Learner Finishes
```

Algorithm 2 details the proposed circuit optimizer. The exploration agent gets an action by entering the current state into the actor network in each step. The algorithm uses the network output (action) to change the size of transistors with the corresponding action values and runs a SPICE simulation to obtain the reward and the state. The reward function in the circuit optimizer is identical to the fitness function (1) employed in the topology generator, except that the scores are obtained at different process corners, as explained later in this section. The current state, the action, the next state, and the calculated reward constitute a single sample and are written to the replay memory. Each time a new sample is sent to the replay memory, the optimizer creates multiple minibatches to update the neural networks. This update process continues until it reaches a user-defined maximum number of steps. Then, the best set of the parameters found in the course of training is selected as the final design.

L2DC [19] uses a recursive neural network (RNN) in the actor network and multilayer perceptron (MLP) as the critic

network. However, RNN is typically hard to train due to the vanishing and exploding gradient problems [29]. Also, the state is composed of the observed values (e.g., g_m and V_{th}) of each transistor, and the order is determined by the signal path of the circuit, necessitating manual examination of the circuit topology. Instead, we use an MLP as the actor as did in AutoCkt [20], where the specifications of topology are combined into a state vector in an arbitrary order. Also, we initialize the weights of the MLP following the method in [30].

While the circuit optimizer primarily focuses on maximizing circuit performance, it is also very important to guarantee that the circuit properly operates under process variations. Contrary to prior works on circuit optimization [19]–[21], we run SPICE simulations at five different process corners (TT, FF, SS, FS, and SF). The optimizer constantly observes if the circuit meets the hard constraints at all corners during optimization. On the contrary, the scores related to the soft constraints are only measured at TT corner. This allows the circuit to exhibit maximum performance at the corner of most concern while still guaranteeing proper functionality in the worst cases. Note that the Monte-Carlo analysis better captures the robustness of a circuit under process variation. However, since the size of each transistor continues to change during optimization, adopting the Monte-Carlo analysis will require a large number of SPICE simulations in each time step, incurring a large time overhead. On the contrary, the corner analysis requires only a few simulations for each design point and, hence, is more suitable for fast optimization.

III. Experiment Result

In the previous section, we presented an unified circuit design framework that automatically generates appropriate circuit topologies and further optimizes each design through finding an optimal size of each transistor. In this section, we experimentally verify the proposed circuit design framework. By employing the framework to design level shifter circuits, we demonstrate that the topology generator produces novel level shifter topologies, and the circuit optimizer successfully improves the design. Finally, the resulting level shifter designs are fabricated and compared against prior arts designed by human experts. All experiments are conducted on a workstation running CentOS 7.4 with two Intel E5-2687W v4 processors, 128-GB DRAM, and an Nvidia GTX Titan X GPU. The topology generator only uses the processors whereas the circuit optimizer uses both the processors and GPU.

A. Level Shifter Design

We choose a level shifter circuit as a test vehicle for our framework since it is an active research area where new circuit topologies are continuously developed. There are many different topologies, and an optimal topology varies with the design constraints [24]. Therefore, the effectiveness of our framework that is capable of finding optimal circuit topologies could be verified more clearly. In addition, level shifter circuits share common properties both with digital and analog circuits. For instance, level shifters operate on a rail-to-rail input signal and produce a rail-to-rail output in a higher voltage, similar to digital circuits [22]. On the other hand, the internal operation is similar to that of analog circuits such as amplifiers. In experiments, we adopt the framework to design level shifter circuits in a 180-nm CMOS process, and the resulting circuits are compared to prior designs reported in the literature.

TABLE I
Experimental Setup for Level Shifter Design

		Topology Generator	Circuit Optimizer
Generation & Step		400	350,000
Process Corners		TT	TT/FF/SS/FS/SF
Hard Constraint	Swing Ratio	$q_{i,x}$ / V_{DDH}	$q_{i,x}$ / V_{DDH}
Soft Constraints	Delay	$-logq_{i,x} + b$	$-logq_{i,x} + b$
	P_{total}	$-logq_{i,x} + b$	$-logq_{i,x} + b$
	P_{static}	$-logq_{i,x} + b$	$-logq_{i,x} + b$
	Area	$1 - \frac{max(q_{i,x}-b,0)}{slope}$	$-logq_{i,x} + b$

A level shifter circuit converts a low-voltage (V_{DDL}) digital signal to a high-voltage (V_{DDH}) signal. Level shifters must generate a rail-to-rail swing between the ground and V_{DDH} at the output. Therefore, we use output signal swing as a hard constraint in the framework. Because level shifters are typically expected to operate with high conversion speed and low power consumption with minimal footprint [31], we use the delay, total power (P_{total}), static power (P_{static}), and area as soft constraints. The circuit area is calculated as the number of transistors in the topology generator, whereas the circuit optimizer uses the total active area.

Table I shows the score functions used in each step. The scores related to the soft constraints are calculated as -log(qi,x) except for the area in the topology generator, whereas the score for the hard constraint (output swing) is calculated as the swing observed in simulation divided by V_{DDH}. In topology generation, the area is calculated as the number of transistors in the circuit. When the number of transistors exceeds a threshold (b in Table I), the score is divided by a slope which is a hyperparamter. In the circuit optimizer, we use the worst values across all process corners when calculating the score for the hard constraint. Soft constraint scores are obtained at the TT corner.

B. Topology Generation

The topology generator runs seven SPICE simulations in parallel only at the TT corner for fast topology search. The input inverter of level-shifter is implemented using low threshold voltage (V_{th}) devices, whereas the other transistors are standard V_{th} devices. We use a minimum-sized transistor with 180-nm channel length and 220-nm channel width as a weak device. Medium and strong devices have 2× and 4× larger channel width, respectively. The initial population size is set to 450, and the population evolves for 600 generations, which takes approximately 5 h 30 min. In addition, we experiment with varying the soft constraint weights in the fitness function to observe how the topology generator performs under different design constraints. The generated circuit topologies are displayed in Fig. 3. Specifically, three cases are tested: 1) all the constraints have the same weight [C1 in Fig. 3]; 2) only the weight of static power is lowered (C2 and C3); and

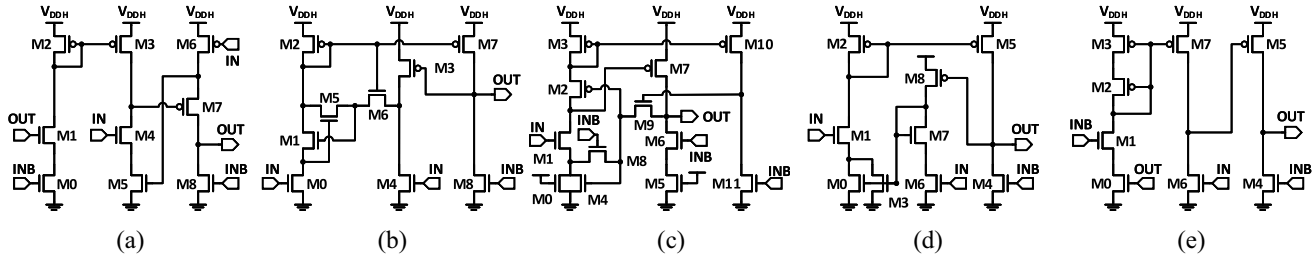


Fig. 3. Level shifter circuit topologies generated by topology generator (a) C1. (b) C2. (c) C3. (d) C4. (e) C5.

TABLE II
RESULTS OF TOPOLOGY GENERATION

V_{DDL}=0.4V @ 1MHz						
	Swing Ratio	P_{total} (nW)	P_{static} (nW)	Delay (ns)	Transistors	Fitness
C1	1.00	47.3	0.10	22.2	9	28.39
C2	0.98	52.2	0.22	11.9	9	26.43
C3	1.00	50.5	0.70	17.8	12	26.36
C4	0.99	58.0	0.71	16.6	9	27.38
C5	1.00	39.8	0.65	15.0	8	27.75

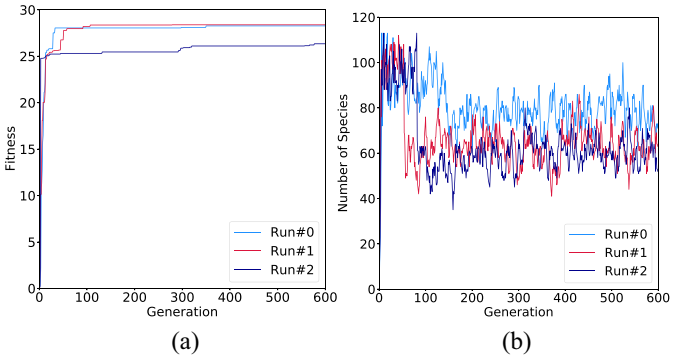


Fig. 4. Experimental results of topology generation. (a) Fitness. (b) Number of species.

3) the weight of delay is increased while the weights of static and total powers are decreased (C4 and C5). Table II summarizes the performance and fitness of the generated circuits (C1–C5). Simulation results show that the circuits generated with lower static power weight (C2 and C3) exhibit higher static power than C1. In addition, the circuits optimized for delay (C4 and C5) achieve lower delay than C1 at the expense of power consumption increases.

We also perform three independent runs of circuit topology generation to estimate algorithm stability. Fig. 4 shows the fitness and the number of species as the evolution proceeds. The best fitness, which is the fitness of the best circuit in the population, rapidly increases in the first 7–9 generations, and then gradually improves through fine tuning of the circuit topology. Note that the value of fitness is not capped at a fixed value. While the fitness of a circuit can have an arbitrary value, the generated circuits exhibit fitness values less than 30 in our experiments. The number of species is nearly constant during evolution except in the first few generations, suggesting that stagnant species are replaced with a similar number of new species.

C. Circuit Optimization

In experiments, we use an MLP with three hidden layers and 200 nodes in each layer as the actor network. The critic network has the same structure but has two hidden layers. First, we evaluate each of the proposed RL optimization techniques using WCMLS circuit, which is widely adopted in level shifter designs [22]–[24]. Experiments are performed using a total of seven actors, where one of them is used to estimate the performance of the optimization algorithm in real time (evaluation actor). Thirty thousand SPICE simulations are run across all the actors except the evaluation actor, which takes 2 h 20 min. Since the RL algorithm has some degree of randomness, we test each configuration on three independent runs to observe its reliability. Fig. 5 summarizes the experimental results. Fig. 5(a) shows that conventional D4PG fails to converge in two of the three runs. However, when the multiupdate technique with $U = 10$ is applied, the algorithm successfully finds a correct optimization direction and properly biases transistors in the circuit after about 7000 SPICE simulations [Fig. 5(c)]. Fig. 5(e) shows the optimization results when the episode early stopping method is also employed. Initially, an episode stops only after four steps, and the episode length increases by two after every five episodes in each exploration agent until it reaches the maximum length of 20. This method reduces the number of SPICE simulations required to capture the bias points from 7000 to 5000, suggesting that this technique accelerates RL training in the early stage. Note that the algorithm shows more fluctuation during optimization when the early stopping method is adopted. We suspect that the conventional approach is exposed to more "bad" samples, which are far from the initial nearly optimized design from the topology generator, in the early stages of training. Those samples exhibit very low rewards as they do not meet the hard constraints. As a result, the actor is trained to be more conservative, and once the design enters the near-optimal region where the hard constraints are satisfied, the algorithm tends to stay near that point only with fine tuning to avoid a large drop in the reward value. On the contrary, the episode early stopping method allows the design to enter the near-optimal region quickly, significantly reducing the number of bad samples during initial training. When the design approaches an optimal point during optimization, the algorithm now searches for better design points more aggressively. In other words, the algorithm is less reluctant to depart from the local optima, which helps find a global optimum.

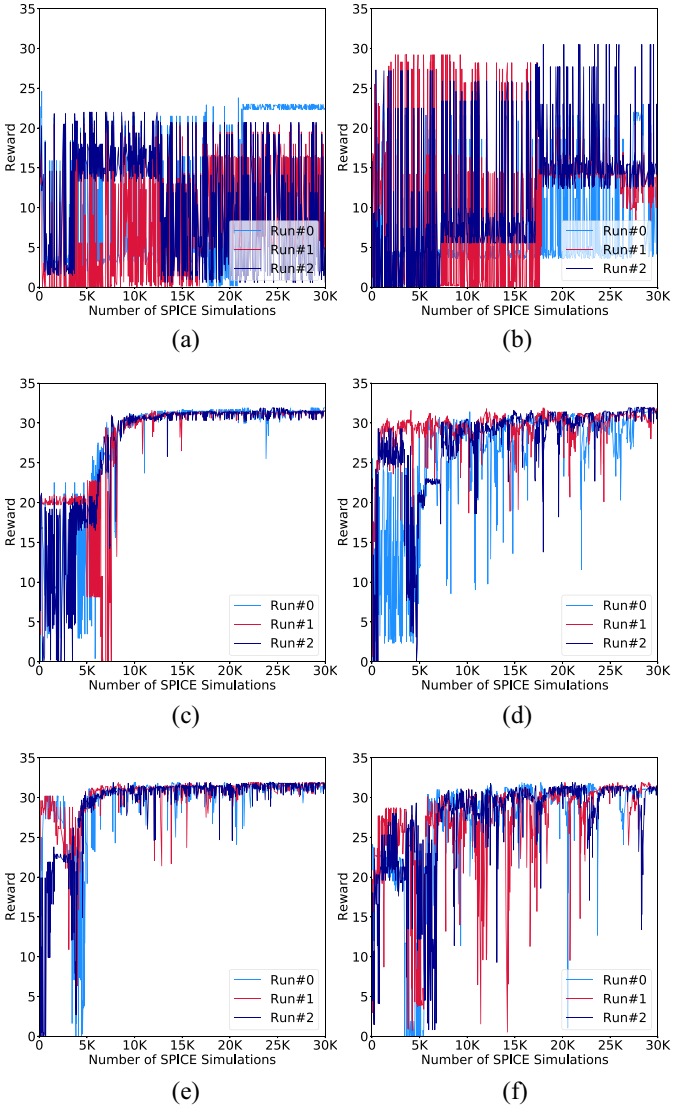


Fig. 5. Trends of reward improvement with and without proposed algorithmic optimization techniques. Proposed techniques enable a faster and more stable optimization process. (a) D4PG. (b) DDPG. (c) D4PG with multiupdate. (d) DDPG with multiupdate. (e) D4PG with multiupdate and episode early stopping. (f) DDPG with multiupdate and episode early stopping.

For comparison, we experiment with the DDPG algorithm adopted in prior work [19] using the same environment. D4PG, which is employed in our framework, is similar to DDPG except that it uses multiple agents in parallel, and the output of the critic network is represented as the probability distribution. The DDPG algorithm is trained for 30 000 SPICE simulations in total, and the total running time is 14 h 30 min. This is more than six times longer than the time required for our approach to process the same number of SPICE simulations, which confirms the effectiveness of the multiagent training of D4PG. The experimental results are displayed in Fig. 5. We experimented with a vanilla DDPG algorithm [Fig. 5(b)], DDPG with multiupdate [Fig. 5(d)], and DDPG with both techniques [Fig. 5(f)]. Experimental results show that DDPG exhibits larger variations between runs and unstable training convergence compared to our approach. To observe how the type of critic affects the training performance, we experimentally

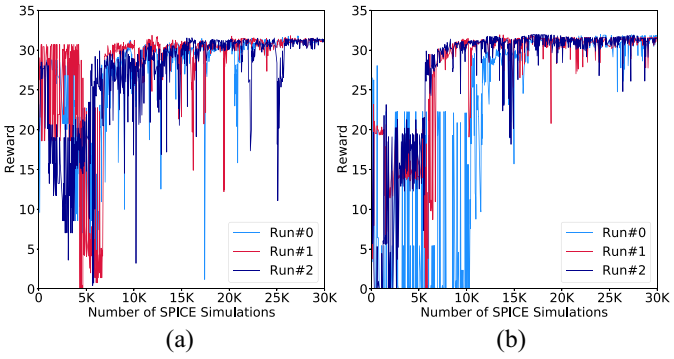


Fig. 6. Reward trends of alternative approaches for comparisons. (a) Scalar value critic. (b) Action scaling.

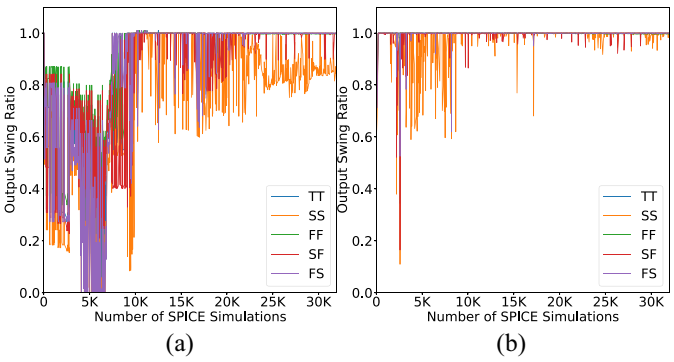


Fig. 7. Trends of output swing ratio when the circuit is optimized (a) at TT corner only and (b) at all process corners. Considering process corners significantly improves reliability.

apply the scalar value critic used in DDPG to our algorithm. With both multiupdate and episode early stopping applied, Fig. 6(a) shows that using the scalar value critic results in more unstable training convergence compared to Fig. 5(e).

The episode early stopping method effectively limits the agent's exploration capability in the early stages of training, and a similar effect could be achieved by scaling the output of the actor network. We conducted additional experiments in which we multiplied the output of the actor network with a scaling factor before passing it to the environment. The scaling factor is set to 0.2 at first and is increased by 0.1 every five epochs, which translates to the maximum amount of size change in each episode identical to the episode early stopping method. Experimental results are displayed in Fig. 6(b). It can be seen that this scaling method results in a slower convergence. We suspect that this is because the actor network is not properly trained in early episodes due to the continuously changing scaling factor. More specifically, the actor network is trained in a way to generate the best action for the current state. However, the output of the actor network is scaled before being applied to the environment and, hence, the actor should take this into account during training. Since we are now changing the scaling factor, the actor should be trained in different directions as the optimization process continues, hindering proper training.

During optimization, our framework considers multiple process corners to make sure the circuit properly works under process variations. Fig. 7 compares our approach to the

TABLE III
EXPERIMENTS WITH DIFFERENT WEIGHTS IN CIRCUIT OPTIMIZATION

V_{DDL}=0.4V @ 1MHz						
Prioritized Metric	Swing Ratio	P_{total} (nW)	P_{static} (nW)	Delay (ns)	Area (μm^2)	Weight*
None	0.97	27.5	1.05	22.5	1.06	1.0/1.0
P_{total}	0.98	26.2	1.05	22.3	1.07	1.6/0.8
	0.99	25.1	0.93	29.6	1.26	2.3/0.6
	0.97	25.0	0.93	27.9	1.18	2.9/0.4
	0.96	24.7	0.89	31.9	1.24	3.1/0.3
Delay	0.97	26.6	1.09	21.1	1.00	1.6/0.8
	0.99	26.6	1.10	20.8	1.03	2.3/0.6
	1.00	27.5	1.16	20.7	1.20	2.9/0.4
	1.00	62.6	6.41	17.9	4.32	3.1/0.3

* Weight of prioritized metric / weight of other metrics

TABLE IV
RESULTS OF OPTIMIZING GENERATED CIRCUITS

V_{DDL}=0.4V @ 1MHz						
	Swing Ratio	P_{total} (nW)	P_{static} (nW)	Delay (ns)	Area (μm^2)	RW
C1	1.00	26.2	0.30	11.4	0.38	33.35
C2	1.00	34.2	0.16	9.7	0.45	33.49
C3	1.00	34.3	0.36	12.5	0.68	33.02
C4	0.99	37.3	0.32	9.8	0.37	33.28
C5	1.00	26.0	0.43	11.3	0.32	33.25

conventional method that observes the circuit performance at the TT corner only. When the circuit is optimized only at the TT corner, the relative output voltage swing reaches 0.95 at the same corner, but the design may produce much smaller swing at different corners [Fig. 7(a)]. On the other hand, if we obtain the score related to the hard constraint at the worst corner during optimization, the resulting circuit achieves >0.95 output swing at all the corners.

Similar to topology generator, we also experiment with changing the weights of the soft constraints. The sum of the weights is fixed, and their values are allocated differently in each case. Table III summarizes experimental results for optimization with 32 000 SPICE simulations. The last column shows the actual weights of the soft constraint of interest and the others. As expected, increasing the weight for total power consumption further reduces power consumption during optimization while sacrificing delay and area since the circuit is subject to a tradeoff between delay, power, and area. Similarly, using a higher weight for delay produces a faster level shifter circuit at the expense of power and area increase.

Finally, we apply our circuit optimizer to the circuits generated by the topology generator (C1–C5). Similar to previous experiments, we use seven actors in the RL algorithm, where one of them is used as an evaluation actor. For each circuit topology, 38 000 SPICE simulations were performed except the evaluation actor, and the multiupdate constant U was set to 13. The optimizer successfully improved all the generated circuit topologies, which is verified by comparing the results in Table IV to the results in Table II. Note that the area represents the total active area, not the actual layout size.

D. Test Chip Fabrication

To validate level shifter circuits designed by our framework, we fabricated the generated and optimized circuits C1–C5 in a 180-nm process. Since the framework only provides a netlist as the output, the layout was manually drawn, as shown in Fig. 8. The input inverter supplied by V_{DDL} is included in the layout. It is difficult to measure the conversion delay of a level shifter accurately, since parasitic components (e.g., I/O cell, PCB trace, and bond wire) also contribute to the delay. Hence, we adopt the dual-path measurement method in [32]. Two different paths with and without a level shifter are implemented, and the conversion delay is indirectly measured by subtracting their delays as depicted in Fig. 9(a). The V_{DDL} inverter (colored gray in the figure) converts a high-voltage input to a low-voltage signal, which is later converted back to V_{DDH} by the level shifter. Each level shifter has a dedicated power supply rail to measure its power consumption. A different level shifter can be selected by a control signal to the multiplexer and demultiplexer. The conversion delay is measured as the difference in arrival times of OUT and REF signals. Fig. 9(b) shows the top-level layout of the test chip, and Fig. 9(c) is the chip micrography.

Table V displays measurement results and comparisons against recent level shifter circuits reported in the literature (Fig. 10). Note that the performance of the baseline circuits (B1–B5) are simulation results obtained from [31]. In

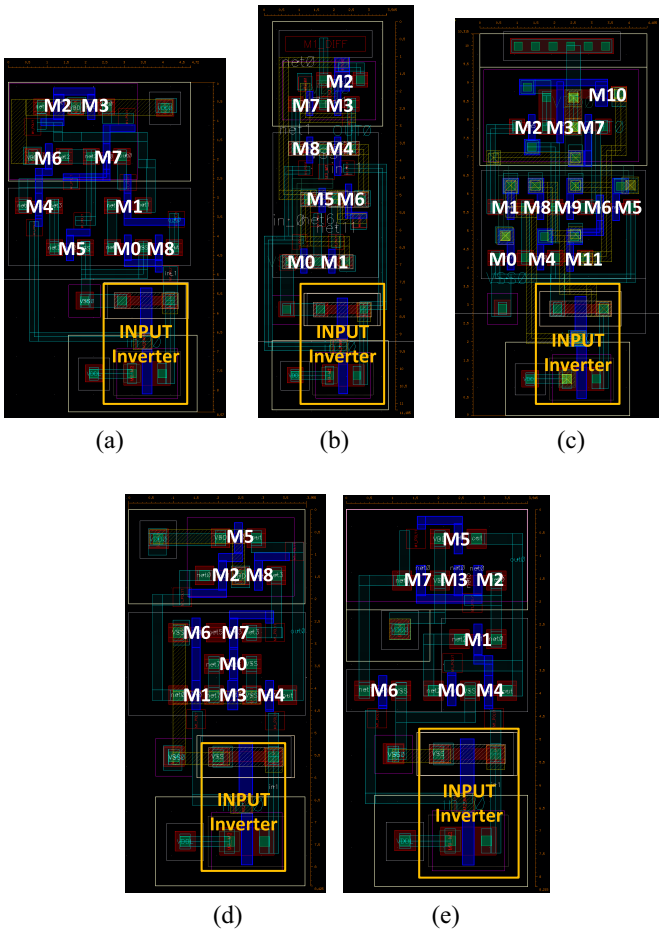


Fig. 8. Layout of generated circuits and their size. (a) C1 (4.7 μm × 8.6 μm). (b) C2 (3.5 μm × 11.2 μm). (c) C3 (4.5 μm × 10.3 μm). (d) C4 (4.0 μm × 8.4 μm). (e) C5 (4.0 μm × 8.2 μm).

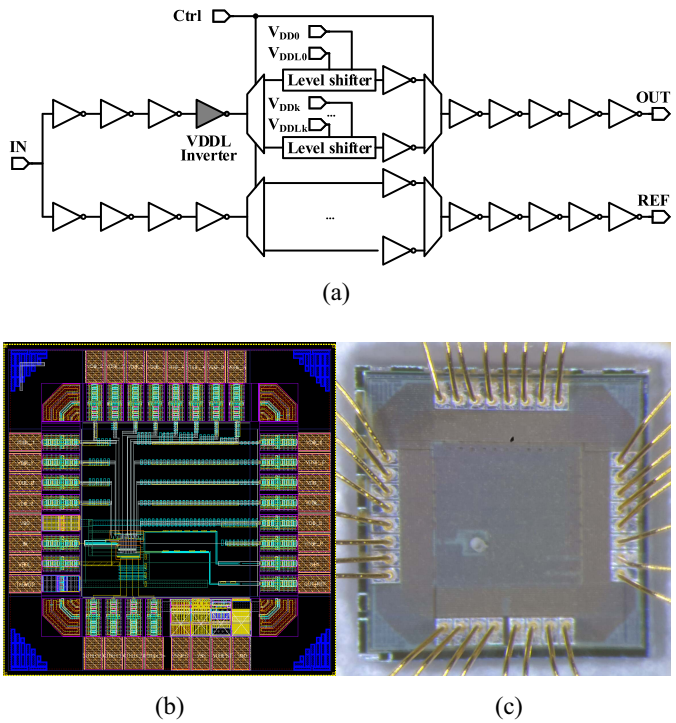


Fig. 9. (a) Delay measurement circuit, (b) test chip layout, and (c) test chip micrography.

TABLE V
MEASUREMENT RESULTS OF GENERATED CIRCUITS

	V_{DDL}=0.4V @ 1MHz					
	V_{DDLmin} (mV)	P_{total} (nW)	P_{static} (nW)	Delay (ns)	Area (μm^2)	PDP
B1*	400 / –	2654	0.98	61	69.1	161894
B2*	370 / –	584	0.18	36	74.4	21024
B3*	380 / –	290	0.22	54	99.8	15660
B4*	370 / –	320	0.13	35	103.5	11200
B5*	360 / –	327	0.13	31	120.9	10137
C1	320 / 42	69.8	0.44	51.6	40.3	3604
C2	280 / 41	85.4	0.47	30.7	39.2	2620
C3	310 / 34	113.2	0.45	31.2	46.3	3533
C4	320 / 70	76.3	1.01	39.2	33.1	2991
C5	270 / 48	62.0	0.95	30.8	32.4	1911

* Simulation results reported in [31]

measurements, all of the generated circuits (C1–C5) successfully perform level conversions. Measurement results show that our designs consume much smaller power consumption during conversion with similar or lower conversion delay. More specifically, our designs exhibit 2.6×–4.7× lower total power consumption than the design with the lowest power consumption (B3) and 1.0×–1.7× larger conversion delay than the fastest design (B5). In addition, our designs occupy 1.5×–2.1× smaller area than the smallest design (B1). The power-delay product (PDP) is a metric commonly used for comparing level shifter circuits [24], [31], [33], and the generated circuits achieve 2.8×–5.3× lower PDP than the baseline circuits.

V_{DDLmin} represents the minimum input voltage that a level shifter can convert to a high voltage signal. V_{DDLmin} was first measured for the input with 1-MHz frequency. Generated circuits (C1–C5) achieve 320 mV or lower V_{DDLmin},

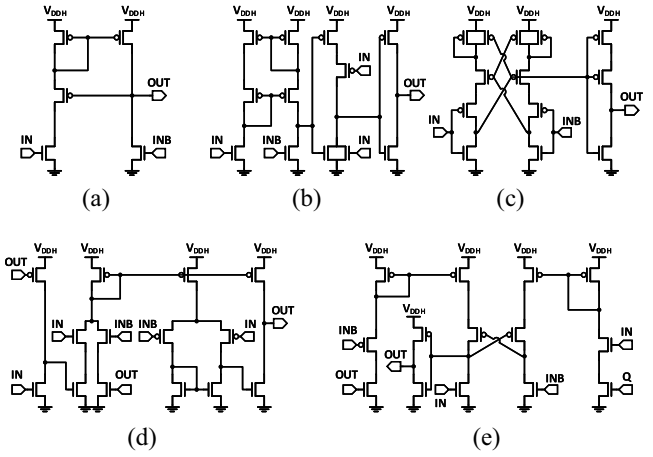


Fig. 10. Baseline level shifter designs from prior work. (a) B1 [22]. (b) B2 [23]. (c) B3 [34]. (d) B4 [35]. (e) B5 [36].

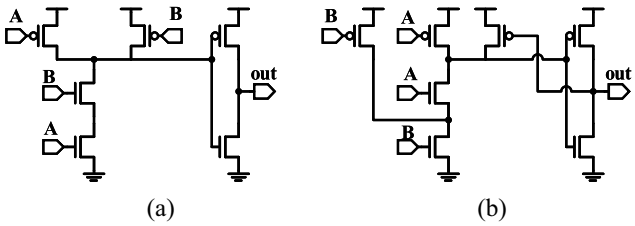


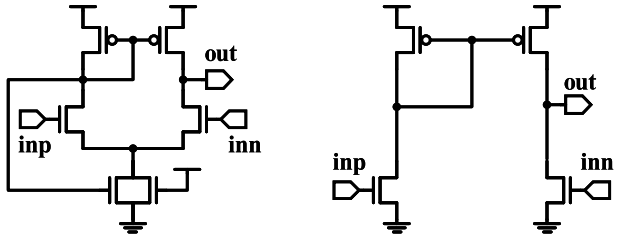
Fig. 11. AND gates generated by topology generator.

outperforming baseline circuits. To determine the lowest possible voltage that the level shifters could handle, we also experimented with a 100-Hz input signal and checked if the output shows full swing. In this case, the generated level shifters achieve significantly lower V_{DDLmin} less than 100 mV.

E. Applicability of Topology Generator

We conduct further experiments to observe if the proposed topology generator could be used for designing other types of circuits. For experiments, the topology generator is tested on both digital (AND gate) and analog (differential amplifier) circuits. In both cases, the algorithm starts with a *P*-channel MOSFET and *N*-channel MOSFET pair as the initial offspring and an initial population size of 600. For AND gate, the population evolves for 300 generations. We use a minimum-sized transistor with 180-nm channel length and 220-nm channel width as a weak device. Medium and strong devices have 2× and 4× larger channel width, respectively. The topology generator successfully produces a standard AND gate composed of a NAND gate and an inverter as shown in Fig. 11(a). The left part of the circuit in Fig. 11(b) is similar to a standard NAND gate, but the output is not fully pulled up since one of the pMOS devices is connected to an internal node. However, the additional pMOS keeper fully pulls up the output node, providing a rail-to-rail output.

For the amplifier design, the population evolves for 600 generations. Since analog circuits often require proper biasing, a bias node that supplies a dc voltage is introduced in the algorithm. In addition, five sizing options are used for topology



(a) (b)

Fig. 12. Differential amplifiers generated by topology generator.

generation. We use a transistor with 720-nm channel length and 220-nm channel width as a baseline. Two stronger devices have 2× and 4× larger channel width, respectively, whereas two weaker devices have 2× and 4× larger channel length, respectively. The topology generator successfully generates circuit topologies that are similar to widely used amplifier circuits. The amplifier circuit in Fig. 12(a) is a self-biased 5T OTA (Operational Transconductance Amplifier) circuit [37], and the circuit in Fig. 12(b) is a low-voltage pseudodifferential amplifier [38], [39].

IV. Conclusion

In this work, we proposed an automatic circuit design framework for level shifter circuits. To design a circuit without preconstructed building blocks and prior knowledge, the framework implements a two-step design process using the topology generator and the circuit optimizer. We first proposed a new graph-based circuit representation, and the topology generator employs an evolutionary algorithm to search for possible circuit topologies quickly, considering the given design constraints. Then, the circuit optimizer utilizes RL to fine-tune the size of each transistor, where we adopt various algorithmic optimizations, such as multiagent training, process variation-aware optimization, multiupdate, and episode early stopping to improve sample efficiency. In experiments, the framework was applied to designing level shifter circuits. The topology generator produced novel level shifter topologies, and they are successfully optimized by the circuit optimizer. Fabricated in a 180-nm CMOS process, the test chip demonstrates that the automatically designed circuits achieve 2.8×–5.3× lower PDP than manually designed level shifter circuits reported in the literature.

References

[1] L. Lavagno, L. Scheffer, and G. Martin, *EDA for IC Implementation, Circuit Design, and Process Technology*. Boca Raton, FL, USA: CRC Press, 2016.

[2] O. Aaserud and I. R. Nielsen, "Trends in current analog design—A panel debate," *Analog Integr. Circuits Signal Process.*, vol. 7, no. 1, pp. 5–9, 1995.

[3] M. C. Golumbic, A. Mintz, and U. Rotics, "An improvement on the complexity of factoring read-once Boolean functions," *Discr. Appl. Math.*, vol. 156, no. 10, pp. 1633–1636, May 2008.

[4] V. N. Possani, V. Callegaro, A. I. Reis, R. P. Ribas, F. De Souza Marques, and L. S. Da Rosa, "Graph-based transistor network generation method for supergate design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 692–705, Feb. 2016.

[5] H. Y. Koh, C. H. Séquin, and P. R. Gray, "OPASYN: A computer for CMOS operational amplifiers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 2, pp. 113–125, Feb. 1990.

[6] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Proc. 32nd Annu. ACM/IEEE Des. Autom. Conf.*, 2002, pp. 139–144.

[7] T. McConaghy, P. Palmers, M. Steyaert, and G. G. Gielen, "Variation-aware structural synthesis of analog circuits via hierarchical building blocks and structural homotopy," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 1, pp. 1281–1294, Sep. 2009.

[8] M. Meissner and L. Hedrich, "FEATS: Framework for explorative analog topology synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 2, pp. 213–226, Feb. 2015.

[9] Z. Zhao and L. Zhang, "Graph-grammar-based analog circuit topology synthesis," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.

[10] J. R. Koza, F. Dunlap, F. H. Bennett, M. A. Keane, J. Lohn, and D. Andre, "Automated synthesis of computational circuits using genetic programming," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1997, pp. 447–452.

[11] J. D. Lohn and S. P. Colombano, "A circuit representation technique for automated circuit design," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 205–219, Sep. 1999.

[12] Y. Sapargaliyev and T. G. Kalganova, "Unconstrained evolution of analogue computational 'QR' circuit with oscillating length representation," in *Proc. Int. Conf. Evolvable Syst.*, Sep. 2008, pp. 1–10.

[13] J. Slezák and J. Petržela, "Evolutionary synthesis of cube root computational circuit using graph hybrid estimation of distribution algorithm," *Radioengineering*, vol. 23, no. 1, p. 549, 2014.

[14] B. Liu, Q. Zhang, F. V. Fernàndez, and G. G. E. Gielen, "An efficient evolutionary algorithm for chance-constrained bi-objective stochastic optimization," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 786–796, Dec. 2013.

[15] P. P. Prajapati and M. V. Shah, "Two stage CMOS operational amplifier design using particle swarm optimization algorithm," in *Proc. IEEE UP Section Conf. Electr. Comput. Electron.*, 2015, pp. 1–5.

[16] R. A. Thakker, M. S. Baghini, and M. B. Patil, "Low-power low-voltage analog circuit design using hierarchical particle swarm optimization," in *Proc. Int. Conf. VLSI Des.*, 2009, pp. 427–432.

[17] W. Lyu *et al.*, "An efficient Bayesian optimization approach for automated optimization of analog circuits," *IEEE Trans. Circuits Syst. I, Reg. Paper*, vol. 65, no. 6, pp. 1954–1967, Jun. 2018.

[18] B. He, S. Zhang, F. Yang, C. Yan, D. Zhou, and X. Zeng, "An efficient Bayesian optimization approach for analog circuit synthesis via sparse Gaussian process modeling," in *Proc. Design Autom. Test Eur. Conf. Exhib.*, 2020, pp. 1463–1468.

[19] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," 2018, *arXiv:1812.02734*.

[20] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. Design Autom. Test Eur. Conf. Exhibition*, 2020, pp. 490–495.

[21] H. Wang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. Design Autom. Conf.*, 2020, pp. 1–6.

[22] S. Lütkemeier and U. Rückert, "A subthreshold to above-threshold level shifter comprising a Wilson current mirror," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 9, pp. 721–724, Sep. 2010.

[23] S.-C. Luo, C.-J. Huang, and Y.-H. Chu, "A wide-range level shifter using a modified wilson current mirror hybrid buffer," *IEEE Trans. Circuits Syst. I, Reg. Paper*, vol. 61, no. 6, pp. 1656–1665, Jun. 2014.

[24] S. Kabirpour and M. Jalali, "A power-delay and area efficient voltage level shifter based on a reflected-output Wilson current mirror level shifter," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 2, pp. 250–254, Feb. 2020.

[25] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[26] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," Sep. 2015, *arxiv:1509.02971*.

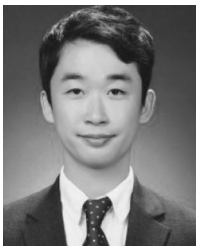
[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Jul. 2017, *arxiv:1707.06347*.

[28] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients," Apr. 2018, *arxiv:1804.08617*.

[29] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[31] S. R. Hosseini, M. Saberi, and R. Lotfi, "A high-speed and power-efficient voltage level shifter for dual-supply applications," *IEEE Trans. VLSI Syst.*, vol. 25, no. 3, pp. 1154–1158, Mar. 2017.
[32] R. Lotfi, M. Saberi, S. R. Hosseini, A. R. Ahmadi-Mehr, and R. B. Staszewski, "Energy-efficient wide-range voltage level shifters reaching 4.2 fJ/transition," *IEEE Solid-State Circuits Lett.*, vol. 1, no. 2, pp. 34–37, Feb. 2018.
[33] S. Kabirpour and M. Jalali, "A low-power and high-speed voltage level shifter based on a regulated cross-coupled pull-up network," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 6, pp. 909–913, Jun. 2019.
[34] M. Lanuzza, P. Corsonello, and S. Perri, "Fast and wide range voltage conversion in multisupply voltage designs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 2, pp. 388–391, Feb. 2015.
[35] Y. Osaki, T. Hirose, N. Kuroki, and M. Numa, "A low-power level shifter with logic error correction for extremely low-voltage digital CMOS LSIs," *IEEE J. Solid-State Circuits*, vol. 47, no. 7, pp. 1776–1783, Jul. 2012.
[36] S. R. Hosseini, M. Saberi, and R. Lotfi, "A low-power subthreshold to above-threshold voltage level shifter," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 10, pp. 753–757, Oct. 2014.
[37] B. A. Chappell *et al.*, "Fast CMOS ECL receivers with 100-mV worst-case sensitivity," *IEEE J. Solid-State Circuits*, vol. 23, no. 1, pp. 59–67, Feb. 1988.
[38] C. J. A. Gomez, H. Klimach, E. Fabris, and O. E. Mattia, "High PSRR nano-watt MOS-only threshold voltage monitor circuit," in *Proc. IEEE Symp. Integr. Circuits Syst. Design (SBCCI)*, 2015, pp. 1–6.
[39] A. Shankar, J. Silva-Martínez, and E. Sánchez-Sinencio, "A low voltage operational transconductance amplifier using common mode feedforward for high frequency switched capacitor circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 1, 2001, pp. 643–646.



Jiwoo Hong (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2019, where he is currently pursuing the M.S. degree with the Graduate School of Convergence Science and Technology.

His current research interests include electronic design automation and circuit design automation with machine learning.

Sunghoon Kim (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2018, where he is currently the Ph.D. degree with the Graduate School of Convergence Science and Technology.

His current research interests include radiation hardening circuit, mixed-signal machine learning hardware, and computation in memory.

Dongsuk Jeon (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2009, and the Ph.D. degree in electrical engineering from the University of Michigan at Ann Arbor, Ann Arbor, MI, USA, in 2014.

From 2014 to 2015, he was a Postdoctoral Associate with the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently an Associate Professor with the Graduate School of Convergence Science and Technology, Seoul National University. His current research interests include hardware-oriented machine learning algorithms, hardware accelerators, and low-power circuits.