A Module-Linking Graph Assisted Hybrid Optimization Framework for Custom Analog and Mixed-Signal Circuit Parameter Synthesis

MOHSEN HASSANPOURGHADI, REZWAN A. RASUL, and MIKE SHUO-WEI CHEN,
University of Southern California

Analog and mixed-signal (AMS) computer-aided design tools are of increasing interest owing to demand for the wide range of AMS circuit specifications in the modern system on a chip and faster time to market requirement. Traditionally, to accelerate the design process, the AMS system is decomposed into smaller components (called *modules*) such that the complexity and evaluation of each module are more manageable. However, this decomposition poses an interface problem, where the module's input-output states deviate from when combined to construct the AMS system, and thus degrades the system expected performance. In this article, we develop a tool module-linking-graph assisted hybrid parameter search engine with neural networks (MOHSENN) to overcome these obstacles. We propose a module-linking-graph that enforces equality of the modules' interfaces during the parameter search process and apply surrogate modeling of the AMS circuit via neural networks. Further, we propose a hybrid search consisting of a global optimization with fast neural network models and a local optimization with accurate SPICE models to expedite the parameter search process while maintaining the accuracy. To validate the effectiveness of the proposed approach, we apply MOHSENN to design a successive approximation register analog-to-digital converter in 65-nm CMOS technology. This demonstrated that the search time improves by a factor of 5 and 700 compared to conventional hierarchical and flat design approaches, respectively, with improved performance.

CCS Concepts: • **Hardware → Data conversion**; **Analog and mixed-signal circuit synthesis**; Modeling and parameter extraction;

Additional Key Words and Phrases: CAD tool, deep neural network

ACM Reference format:
Mohsen Hassanpourghadi, Rezwan A. Rasul, and Mike Shuo-Wei Chen. 2021. A Module-Linking Graph Assisted Hybrid Optimization Framework for Custom Analog and Mixed-Signal Circuit Parameter Synthesis. *ACM Trans. Des. Autom. Electron. Syst.* 26, 5, Article 38 (May 2021), 22 pages.
https://doi.org/10.1145/3456722

1 INTRODUCTION

The future trends of electronic systems, such as the Internet of Things and 5G communications, demand high-performance **analog and mixed-signal (AMS) intellectual properties (IP)** [14]. Concurrently, the AMS IP design is becoming more expensive and time consuming due to more

Authors' address: M. Hassanpourghadi, R. A. Rasul, and M. S.-W. Chen, University of Southern California, 3737 Watt Way, Los Angeles, CA 90089-2560; emails: {mhassanp, rrasul, swchen}@usc.edu.

1084-4309/2021/05-ART38 $15.00
https://doi.org/10.1145/3456722

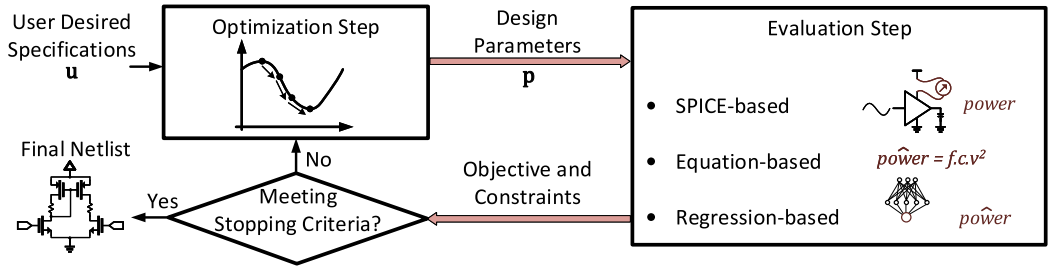


Fig. 1. Design of custom AMS parameter synthesis with optimization.

sophisticated technology design rules and increasing complexity of systems on a chip. Although sharing an IP among different applications can significantly reduce the design cost, this is an inefficient solution since the AMS circuit is application specific and only performs efficiently for a particular task. Accordingly, the ideal solution then is to optimize the same IP for different target specifications but with a significantly reduced design cost. Since AMS circuit design of a particular topology usually follows a similar procedure, the fixed-topology IP can be synthesized for different design specifications by a **computer-aided design (CAD)** tool with reduced design cost and development time [19].

One imperative step in the automated synthesis of a complex AMS system is parameter synthesis, which is the appropriate sizing of circuit elements. As discussed elsewhere [12, 19, 21], enhancing the performance of parameter synthesis can significantly improve layout generation and final verification in the AMS design process. Usually, the parameter synthesis of a complex AMS system is a two-step iterative process. The steps are evaluation and optimization, and they occur successively as presented in Figure 1. There are three main ways to perform the evaluation step: SPICE based, equation based, and regression based [19] as illustrated in Figure 1.

Performing the evaluation step of a complex AMS system in SPICE is accurate but time consuming [9]. As a result, it is infeasible to synthesize the entire AMS circuit with SPICE in-loop and find the appropriate design parameters that meet the specification. To speed up the evaluation step, the circuit's objectives and the constraints can be expressed as an analytical equation of the design parameters [4, 8, 23]. Consequently, the evaluation can take place much faster compared to using a SPICE simulation. However, it is complicated to derive accurate equations, and higher-order effects are often ignored for simplifications, limiting the accuracy of this method. To bypass manual equation derivation yet maintain evaluation speed, regression models are used. In this case, the accuracy depends on the number of system-level simulation data used for training the regression model [3, 6, 11, 16, 20, 25–27, 32, 34, 35].

Among various regression models, such as **support vector regression (SVR)** [3], **random forest (RF)** [35], and *k***nearest neighbors (kNN)** [32], the **neural network (NN)** [34] model provides a promising opportunity. NN models have shown excellent performance when approximating any non-linear function, given sufficient training data points [12, 16, 18, 34, 36]. Most previous works employed NN-based models to approximate a less complicated AMS circuit such as an amplifier with 5, 7, or 10 parameters per Wolfe and Vemuri [34], İslamoğlu et al. [36], and Li et al. [16], respectively. In Garitselov et al. [11], the NN model was used to approximate a **Phase-Locked Loop (PLL)** with 21 parameters; however, the scheme covered a small parameter range for training and searching because generating training data is expensive for a complete PLL. An AMS system often consists of a large number of parameters with wide ranges. As a result, generating a general and accurate NN model requires a huge training dataset for reasonable accuracy, incurring many time-consuming SPICE simulations.

Since modeling a complex AMS circuit altogether is challenging, an improved approach is to break the system down into multiple simpler modules. This method is called *hierarchical design* and improves the accuracy and maintains fast evaluation of the modules [9]. However, module breakdown poses a unique challenge known as the interface problem [7]. In other words, if the neighboring modules are not modeled with equal interfaces, their performance estimation may be inaccurate. Previous works have addressed this challenge by introducing a set of connectivity rules that help preserve the module input-output conditions when modeled separately [27]. However, these methods do not guarantee the equality of interfaces and can suffer from inaccuracies.

In this article, we propose MOHSENN, a **module-linking-graph (MLG)** assisted hybrid parameter search engine with NNs to alleviate the preceding challenges. We propose an MLG to address the interface issue, which forces equality on the shared circuit elements at the interface. To accelerate the design process and cover a wide design parameter range, we propose the hybrid search. In the first phase, the hybrid search exploits the adoption of NN regression models on the MLG in the global search, where it performs a fast and parallel gradient-based optimization on the design parameters. In the second phase, to attenuate the modeling inaccuracy, we perform a local search on the MLG using SPICE simulation. This step is further accelerated with the proposed gradient-based variable reduction technique that limits the number of selected design parameters for optimization. To prove the concept, we use MOHSENN for a **successive approximation register analog-to-digital converter (SAR ADC)** design in 65-nm CMOS technology, which achieves 5X and 700X faster search speed compared to the conventional hierarchical and flat approach, respectively, with relatively better ADC performance. Moreover, to demonstrate that MOHSENN is capable of realizing custom AMS IP, it is used to design three custom SAR ADCs of a fixed topology with diverse performance specifications.

2 PROPOSED MLG-ASSISTED HYBRID SEARCH ENGINE WITH NN

In the following sections, we will formulate the hierarchical design problem and then introduce the flow of MOHSENN and the enabling techniques.

2.1 Problem Statement

The objective of the AMS circuit sizing problem is to find the design parameter vector $\mathbf{p}$ of size n_p within a finite parameter space $\mathbb{P}$ to minimize and satisfy the system-level constraints set by the user desired specification vector $\mathbf{u}$. Circuit design parameters are transistor geometries, biases, or any other variables that can affect the circuit performance. User desired specifications $\mathbf{u}$ are high-level and simple measures of the AMS IP's performance understandable to non-expert users. For example, the number of bits or the bandwidth of an ADC can be $\mathbf{u}$.

In hierarchical design, an AMS circuit is divided into multiple smaller circuits, which are referred to as modules in this article. For each module, a **parameter-to-modules' metric (P2M)** function is defined that maps the module's design parameters to its metrics. For example, metric can be the gain of an amplifier or the delay of a D-flip-flop. For an AMS circuit consisting of N modules, we can express the P2M function of the i^{th} module as

$$\mathbf{m}^i = f^i(\mathbf{p}^i), \tag{1}$$

where $\mathbf{p}^i$ is the parameter vector, $\mathbf{m}^i$ is the metric vector, and $f^i(.)$ is the P2M function. We assume that the P2M function can be accurately characterized using SPICE simulation and foundry-provided device models. In a model-based method, a regression model is approximating the P2M function and is denoted by $\hat{f}^i$ that outputs the estimated modules' metrics $\hat{\mathbf{m}}^i$.

To design the entire AMS circuit, equations or behavioral models that map the modules' metrics to the system-level design objectives or constraints [9, 13] are necessary. With the system-level

objective function and constraint functions denoted by $obj()$ and $\mathbf{h}()$, respectively, the search problem can be expressed as follows [16]:

$$\begin{aligned} &Given\ \mathbf{u}, \\ &\arg\min_{\mathbf{p}\in\mathbb{P}} \quad obj(\mathbf{m}, \mathbf{p}^B), \\ &\quad s.t.: \quad h_j(\mathbf{m}, \mathbf{u}, \mathbf{p}^B) \geq 0,\ j \in \{1, 2, \ldots, n_h\}, \\ &\qquad\quad \mathbf{m} = [\mathbf{m}^1, \mathbf{m}^2, \ldots, \mathbf{m}^N],\ \mathbf{p} = [\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^N, \mathbf{p}^B], \mathbf{m}^i = f^i(\mathbf{p}^i), i \in \{1, 2, \ldots, N\}, \end{aligned} \tag{2}$$

where n_h is the number of constraints. In (2), $\mathbf{m}$ is derived from concatenating every modules' metrics, and $\mathbf{p}^B$ is the system-level parameter vector that is absent in any module but would affect the final performance. For example, $\mathbf{p}^B$ can be the number of modules required to drive a clock routing.

In (2), $\boldsymbol{f}$ generates the modules' metrics $\mathbf{m}$ through circuit-level simulations and both obj and $\mathbf{h}$ are system-level functions of $\mathbf{m}$. However, as opposed to objective function obj, the constraint functions $\mathbf{h}$ also require information from the user desired specification to set the bounds in the search problem. As an example, obj can be the power or area of an AMS circuit, which should be minimized via optimization. However, an example of $\mathbf{h}$ is the **signal-to-noise-ratio (SNR)** of an ADC that should be larger than $6 \times n_{bit} + 1.76$. In other words, $SNR - 6 \times n_{bit} - 1.76 \geq 0$, where $\mathbf{u} = [n_{bit}]$ is the user desired number of bits. A concrete example of constraint and objective function setup for optimizing a SAR ADC design will be provided in Section 4.1.3.

Conventionally, two different levels of optimization are utilized to solve (2). The first step is the system level, where the constraints on h_js are satisfied by finding modules' metrics. The second step is the module level, where each module's parameters are chosen individually to satisfy the found metrics at the system level. Designing modules separately exacerbates the interface problem since it ignores the interrelations between modules. In MOHSENN, we use the proposed MLG to design the dependent modules jointly. This could significantly increase the conventional optimization time; therefore, we propose a hybrid search to accelerate the design process.

2.2 MOHSENN Flow Overview

Figure 2 illustrates the MOHSENN design flow, which requires a one-time offline preparation of the MLG and the hybrid search.

In the preparation phase, after the module breakdown, we identify the necessary interfaces among modules that should be modeled in their P2M characterization functions. Subsequently, we propose to construct the MLG ($g(.)$) and incorporate the interfaces as the graph's vertices with a method that will be discussed in Section 3.1. The MLG accommodates a platform for obtaining the module metrics in the presence of interfaces from adjacent modules. Note that the MLG's output (modules' metrics) can be obtained either from fast NN models or more accurate SPICE simulations. This is crucial for the fast yet accurate exploration of the parameter space using the hybrid search.

The proposed hybrid search algorithm consists of a global and a local search phase. The global search phase utilizes NN models for evaluation, with the proposed **parallel Monte Carlo (Par-MC)** to select multiple random initial points to search over a broad region. This phase outputs a set of potential parameter candidates, one of which is selected for the local search. The proposed gradient-based variable reduction technique reduces the number of design parameters for the local search phase, resulting in a faster search result. This phase utilizes SPICE for evaluation to reduce the errors introduced by the regression models. The final output of the hybrid search is the design parameters for the given AMS system topology, which satisfies the user desired specification.

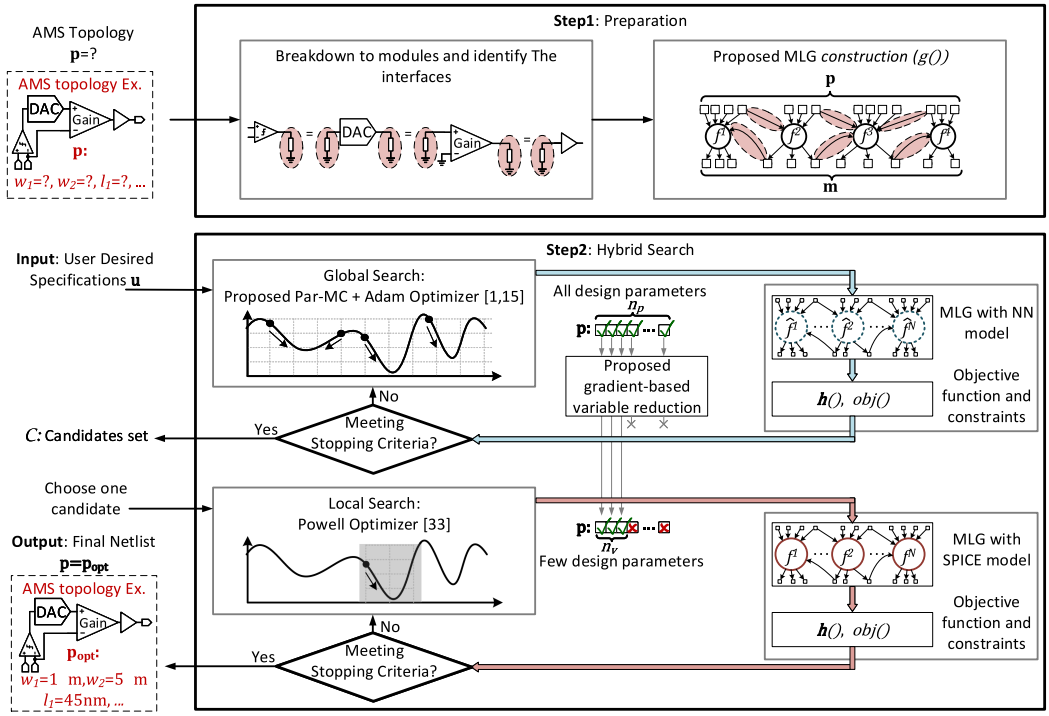


Fig. 2. Proposed MOHSENN flowchart, the preparation, and the hybrid search.

3 MOHSENN PREPARATION AND HYBRID SEARCH

In the following sections, we will describe several enabling techniques in the proposed MOHSENN flow. For simplification, we denote constraints by $\mathbf{h} = [h_1, h_2, \ldots, h_{n_h}]$, the SPICE model P2M functions by $\boldsymbol{f} = [f^1, f^2, \ldots, f^N]$, and for the NN models we use $\hat{\boldsymbol{f}} = [\hat{f}^1, \hat{f}^2, \ldots, \hat{f}^N]$.

3.1 MLG Construction

If the modules are designed individually without considering the interface present in the AMS system, modules' metrics estimated during the design may deviate from actual values, causing interface problems. In this article, we assume the interface is an inter-module variable that can simultaneously affect all physically connected modules. For example, this assumption includes the loading effect, where the load impedance of the driving module and the input impedance of the loading modules are the same variable and should be equated during the design stage.

Among the prior hierarchical design methodologies, contract-based design [5, 27] is one of the most successful methods that specifically address the interface problem. This method includes a set of rules named *contracts*, which determine the connectivity and the relation among modules in the system-level design. By using these contracts, the CAD tool shrinks the feasible range for the interfaces. Hence, the derived interfaces during module design are more accurately matched compared to conventional methods. However, contract-based design can only guarantee near (not full)-equality of the interfaces. This near-equality causes a small deviation in module behavior prediction in the design phase, which can slightly damage the AMS system final performance.

We use an MLG in this work, which enforces full equality of the interfaces among adjacent modules. The MLG, as shown in Figure 3, can be represented as a directed graph that consists

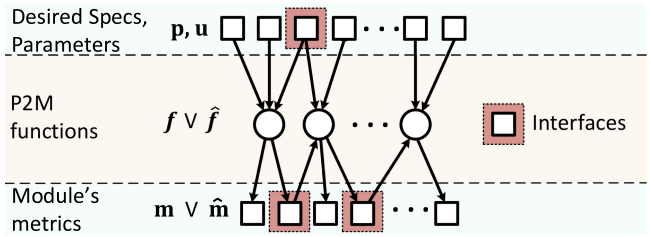


Fig. 3. Graphical depiction of MLG.

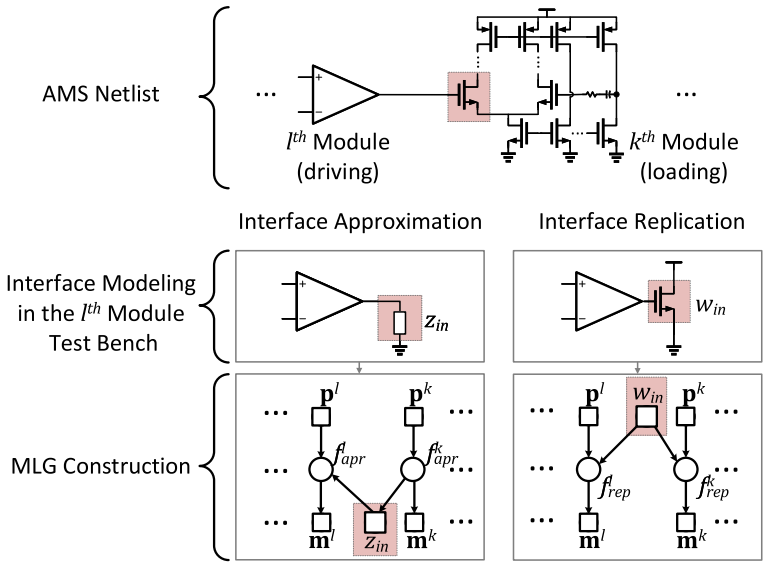


Fig. 4. Two types of interface modeling in module test benches.

of the design parameters $\mathbf{p}$, user desired specifications $\mathbf{u}$, P2M functions (f or $\hat{f}$), and modules' metrics ($\mathbf{m}$ or $\hat{\mathbf{m}}$) as the vertices. The functional representation of MLG is given by $[\mathbf{m} \vee \hat{\mathbf{m}}] = g\big(\mathbf{p}, \mathbf{u}, [f \vee \hat{f}]\big)$. In other words, MLG outputs the modules' metrics when the design parameters and desired specifications are provided. An interface between modules is modeled as a vertex in the graph, which either sources edges to or sinks edges from multiple f or $\hat{f}$ vertices as shown in Figure 3.

To construct the MLG, we should first depict each module's sub-graph representation, then connect those graphs with the interface vertices. The graphical representation of the i^{th} module sources only from its corresponding parameters ($\mathbf{p}^i$) into the P2M function (f^i), then sinks into its metrics vertices ($\mathbf{m}^i$). After constructing the module sub-graphs, we add the interfaces' nodes and connect them according to the interface characterization type.

MLG can be used with both conventional types of interface characterization (approximation and replication), as illustrated in the example of Figure 4. When interface approximation is used, the input impedance of the loading module should be characterized as its metric. Simultaneously, the load impedance of the driving module should be characterized as its design parameter. Since they are the same variable in the AMS system, they are represented as a single vertex (i.e., z_{in} in Figure 4), which sources an edge to the driving module and sinks an edge from the loading module.

However, when interface replication is used, both the driving and the loading modules should be characterized with the input transistor of the loading module, making it a design parameter for both. Then, a single vertex in the MLG would represent this interface variable, sourcing two edges to the modules (i.e., w_{in} in Figure 4).

Interface replication is preferable over approximation for MLG. When approximation is employed, the evaluation of an adjacent module's P2M functions inside the MLG must be serialized. In the previous example, the loading module's input impedance (z_{in}) should be derived first before evaluating the driving module. In the case of replication, the evaluation can be processed in parallel as there is no such dependency, resulting in accelerated metric estimation by parallel processing. Interface approximation also causes multiple layers in the MLG, which makes gradient estimation more difficult. In the same example, estimating the gradient of the driving module's metrics with respect to the loading module's parameters ($\frac{\partial \hat{\mathbf{m}}^l}{\partial \mathbf{p}^k}$) requires back-propagation through two P2M functions ($\frac{\partial \hat{\mathbf{m}}^l}{\partial \hat{\mathbf{z}_{in}}} and \frac{\partial \hat{\mathbf{z}}_{in}}{\partial \mathbf{p}^k}$) according to the chain rule. However, the same gradient ($\frac{\partial \hat{\mathbf{m}}^l}{\partial w_{in}}$) requires back-propagation through a single P2M function in case of replication. Another drawback of interface approximation arises in systems where the modules cyclically affect each other, such as in feedback-based circuits (i.e., delta-sigma modulators or PLL). In such circuits, a loop may be created inside MLG, which slows down the inference and gradient estimation.

Besides ensuring equality of the interface, MLG is also recyclable. This means that the same MLG can be used to design an AMS system across different corners and CMOS technologies, provided its topology is fixed. The only parts of MLG that are a function of technology and corner cases are the P2M functions. This implies that incorporating the new technology or corner (e.g., typical, slow, or fast) in the test bench to generate the P2M model is sufficient, requiring no MLG alteration. When the topology of a specific module changes, a slight MLG alteration is required. For example, when replacing a cascode amplifier module with a folded cascode amplifier [29], the interface models do not vary, the MLG remains almost unchanged, and only the design parameters vertices of the new module need to be replaced. This feature of MLG helps accelerate the design flow using MOHSENN across different technologies, corners, and module topologies.

3.2 Hybrid Search

Unlike hierarchical design, MOHSENN uses monolithic optimization on both the system level and module level to solve (2). This is obtained by the composition of MLG $g()$ and the system-level objective function obj and constraints $\mathbf{h}$. Then we can formulate MOHSENN's search problem as

$$\begin{aligned} &Given\ \mathbf{u}, \\ &\underset{\mathbf{p}\in\mathbb{P}}{\arg\min} \quad obj\left(g(\mathbf{p},\mathbf{u},[f \vee \hat{f}]),\mathbf{p}^B\right), \\ &\quad s.t.: \quad h_j\left(g(\mathbf{p},\mathbf{u},[f \vee \hat{f}]),\mathbf{u},\mathbf{p}^B\right) \geq 0,\ j \in \{1,2,\ldots,n_h\}. \end{aligned} \tag{3}$$

To solve (3), we propose to use the NN model for global optimization (global search), followed by a local optimization with SPICE model (local search). The NN model evaluation time is usually less than milliseconds, whereas the SPICE model is of the order of seconds to hours. Further, global optimization spends an order of magnitude larger number of function evaluations for proper exploration compared to the local one. The example of Section 4 complies with this statement, which shows almost 100,000 AMS circuit evaluations for global optimization, compared to less than 1,000 for local ones. Therefore, by sacrificing the accuracy and using NN models instead of SPICE simulations in global optimization, we can avoid several days of convergence, reducing it to minutes. This feature can help us to vary the optimizer's hyper-parameters to achieve better results. The

global optimization submits a set of design parameter candidates (denoted by C) for the user to choose one as the initial point of the local search. Precision is crucial in the local search; hence, we use a precise SPICE model. Due to NN model utilization in the global search, its results may not be even sub-optimal or satisfactory. The local optimization helps to improve the results by moving the candidate toward the nearest optimal point. This combination of the two optimizations helps to search over a wide range and find appropriate results within a short time.

By transforming the constraint satisfactory problem of (3) into optimization problem, we can use the hybrid search. In hybrid search, instead of design parameters $\mathbf{p}$ as variables, we use a mapped version of design parameters denoted by $\mathbf{x}$ to remove the variable bounds. This step is necessary for the type of the optimizers used in MOHSENN. $\mathbf{p}$ has bounds in its domain $\mathbb{P}$ (i.e., $\mathbf{p}_{min} \leq \mathbf{p} \leq \mathbf{p}_{max}$). So, instead of $\mathbf{p}$, we use mapped parameter $\mathbf{x}$ transformed by the saturation function defined by

$$sat : \mathbb{R}^{n_p} \longrightarrow \mathbb{P}, \tag{4}$$
$$sat(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}} \times (\mathbf{p}_{max} - \mathbf{p}_{min}) + \mathbf{p}_{min}.$$

Clearly, we can derive the design parameters at any iteration from the mapped variables by $\mathbf{p} = sat^{-1}(\mathbf{x})$. By constructing MOHSENN's cost function $cost(.)$ we can formulate (3) as an optimization problem expressed by

$$\mathbf{opt}: \underset{\mathbf{x}\in\mathbb{R}^{n_p}}{\arg\min} \quad cost\left(\mathbf{x}, [f \vee \hat{f}]\right) = w^{obj} \times obj\left(g\Big(sat(\mathbf{x}), \mathbf{u}, [f \vee \hat{f}]\Big), \mathbf{p}^B\right) \tag{5}$$
$$+ \sum_{j=1}^{n_h} elu\Bigg(- w_j^h \times h_j\left(g\Big(sat(\mathbf{x}), \mathbf{u}, [f \vee \hat{f}]\Big), \mathbf{u}, \mathbf{p}^B\right)\Bigg),$$

where $\mathbf{w^h} \geq 0$ and $w^{obj} \geq 0$ are the positive weights vector for normalization and $elu()$ is the non-linear **exponential linear unit (ELU)** defined by

$$elu(x) = \begin{cases} e^x - 1 & x \leq 0 \\ x & 0 \leq x \end{cases}. \tag{6}$$

The ELU function imposes a high gradient when the specification is not met and exponentially reduces it if otherwise. Owing to its exponential part, ELU also helps to optimize the specifications further, even after they are satisfied.

3.2.1 Global Search Phase. In the global search phase, MOHSENN uses a **Gradient-Based Optimizer (GBO)** on the NN regression models in combination with the proposed Par-MC scheme to find multiple design parameter candidates. Compared to conventional global optimizers such as **simulated annealing (SA)** or the genetic algorithm, GBOs require less optimization iterations to converge. However, they are more prone to getting stuck in local minimums if the problem is non-convex. In this work, we utilize the Adam optimizer in the global search phase, which has a higher probability of hopping over the local minimums compared to the conventional GBOs such as the Gradient Descent optimizer [15], leading to better overall performance. We employ two hyper-parameters to terminate the search process: the maximum number of iterations (k_{max}) and the tolerance ($ftol$). The optimizer stops when the number of iteration crosses k_{max} or the cost function reaches a value smaller than $ftol$.

Par-MC performs considerably faster than conventional sequential MC schemes if combined with machine learning tools such as TensorFlow [1]. The conventional sequential MC technique picks uncorrelated random initial point for the optimizer in each MC experiment for n_{mc} times. Unfortunately, this approach requires full convergence of each optimization for n_{mc} experiences

ALGORITHM 1: Proposed gradient-based variable reduction

Input: C, n_v
Output: n_v variables
1: Choose one candidate from C named $\mathbf{x}_{og}$
2: Find $||\nabla_{\mathbf{x}} cost(\mathbf{x})|| = [||\frac{\partial cost}{x_1}||, ||\frac{\partial cost}{x_2}||, \ldots, ||\frac{\partial cost}{x_{np}}||]$ at $\mathbf{x} = \mathbf{x}_{og}$
3: Sort $||\nabla_{\mathbf{x}} cost(\mathbf{x})||$ and choose n_v top variables
4: **return** n_v top variables of $\mathbf{x}$

(each time with different initialization), which slows down the search engine. Par-MC, however, optimizes only once by paralleling the experiments, which significantly increases the search speed. In this technique, the variables from different MC experiments are concatenated, and the optimization is performed on the aggregation of all resulting cost functions. We construct the MC's concatenated variable matrix denoted by $\mathbf{X}$ with the size of $[n_p, n_{mc}]$, and we formulate the Par-MC optimization as

$$\textbf{opt}: \underset{\mathbf{X}\in\mathbb{R}^{n_{mc}\times n_p}}{\arg\min} \quad cost_{mc}(\mathbf{X}, \hat{f}) = \sum_{i=1}^{n_{mc}} cost(\mathbf{X}_{\mathbf{i},:}, \hat{f}), \tag{7}$$

where $cost()$ is the cost function from (5) and $\mathbf{X}_{\mathbf{i},:}$ is the i^{th} column of the $\mathbf{X}$. The optimization begins with a randomly picked starting point $\mathbf{X}^{(0)}$ and results in $\mathbf{X}_{\mathbf{og}}$. We stack the columns of $\mathbf{X}_{\mathbf{og}}$ and their corresponding $cost()$ value in C for the user to choose.

The final results of Par-MC are almost equivalent to the sequential MC if a GBO is used for optimization. In a GBO, the optimization direction is only dependent on the gradients and the starting points. Since the derivatives of Par-MC's cost function $cost_{mc}$ with respect to columns of $\mathbf{X}$ are independent of one another, the optimization direction is only related to the columns' initialization. Therefore, if the initialization of $\mathbf{X}_{\mathbf{i},:}$s is equivalent to the sequential MC initialization for each experiment, the two methods' final results should be the same. Note that since n_{mc} cost functions are summed up in the Par-MC scheme, the stopping criterion of tolerance should also be multiplied by this factor. Therefore, the search result of the Par-MC scheme may vary slightly from its sequential counterpart for the same initial point.

The proposed scheme reduces the total number of optimization iterations compared to the conventional scheme of optimizing multiple initial points sequentially. However, it comes at the cost of optimizing more variables per iteration. Thanks to the TensorFlow [1] tool used in this work, the cost becomes negligible. TensorFlow can handle millions of variables while training deep learning models, implying that it can easily operate with n_{mc} times more variables in an iteration even for a sufficiently large value of n_{mc}. As a result, the search space exploration is greatly accelerated.

The global search result $cost(\mathbf{X}_{\mathbf{i},:\mathbf{og}}, \boldsymbol{f})$ can be far from the actual global optimal point if the NN models' inaccuracy is too large. Modules' NN models have regression errors $\epsilon(\hat{\mathbf{m}})$ that cause deviation in the cost function estimation leading to global search performance degradation. The deviation in the cost function is proportional to the regression errors and derivatives with respect to the modules' metric. Hence, to reduce the cost function estimation (i.e., $||cost(\mathbf{X}_{\mathbf{i},:\mathbf{og}}, \hat{\boldsymbol{f}}) - cost(\mathbf{X}_{\mathbf{i},:\mathbf{og}}, \boldsymbol{f})||$) by half, the regression errors should be reduced by half. Therefore, considering this difference, one can determine the tolerable regression errors and tune the corresponding hyper-parameters (e.g., dataset size) accordingly [18].

3.2.2 Local Optimization Phase. In the local search phase, we use a **gradient-free optimizer (GFO)** [30] and SPICE model, which is more precise than NN models. The initial point of the GFO is one of the parameter candidates C picked by the user. Since each optimization step with SPICE

takes a significant amount of time (depending on the module test bench), we want as few steps as possible for the optimization. Therefore, we enlarge $ftol$ for the local search. In addition, we use the proposed gradient-based variable reduction technique illustrated in Algorithm 1 to decrease the number of the optimizer's variables.

In this technique, we sort the design parameters based on their significance in affecting the cost function and then keep n_v most sensitive parameters as the variables for the local optimization. We set the sorting criteria to be $||\frac{\partial cost(\hat{f})}{\partial x_i}||$ obtained from the NN model in decreasing order, in which x_i is the i^{th} element of the variable vector $\mathbf{x}$.

We know that the initial point for the local search phase is one of the columns of $\mathbf{X}$ derived from Adam optimization. Let us call it $\mathbf{x}_{og}$. We also know that Adam obtains this value from the gradients and evaluations performed on the NN model, which suffers from a regression error. The regression error deviates these partial derivatives for the i^{th} variable by ϵ_d^i; therefore, the last partial derivative with respect to the i^{th} design parameter is expressed as

$$\frac{\partial cost(\hat{f})}{\partial x_i} \times \left(1 + \frac{\epsilon_d^i}{\frac{\partial cost(\hat{f})}{\partial x_i}}\right)|_{\mathbf{x}=\mathbf{x}_{og}}. \tag{8}$$

This shows that the cost function is more deviated by the i^{th} variable if either ϵ_d^i is larger or the absolute partial derivative along this variable is smaller. Therefore, we propose to sort the parameters in $\mathbf{x}$ according to the absolute partial derivatives at $\mathbf{x}_{og}$ in decreasing order and choose the top n_v variables.

There is a trade-off between the choice of n_v and the optimization time. Because regression errors and their effects on the update function are stochastical, we suggest considering n_v as a design hyper-parameter. For this work, we use $n_v = 40\%n_p$ as a rule of thumb to enhance the local search speed by a factor of more than 2. Another method can be to set a maximum bound d_{max} for partial derivatives, and parameters are chosen that bring $||\frac{\partial cost(\hat{f})}{\partial x_i}|| < d_{max}$.

After choosing the n_v most sensitive variables, we can run a local optimization with the few remaining parameters. We suggest using the Powell optimization algorithm, which practically converges faster than other typical GFOs for circuit design. We name the optimization result $\mathbf{x}_{ol}$, and we find the design parameter by $\mathbf{p}_{opt} = sat^{-1}(\mathbf{x}_{ol})$. Finally, we send $\mathbf{p}_{opt}$ to the output, which concludes the algorithm.

4 CASE STUDY OF A SAR ADC

This section illustrates an example of our proposed MOHSENN algorithm to design a SAR ADC. The algorithm runs on the TensorFlow platform on a machine running Red Hat 6.10 OS with an Intel Xeon E5-2630 CPU and 32 GB of memory. The block-level verification tests are performed with Spectre on a machine running Red Hat 6.10 OS and with an Intel Xeon E7-4870 CPU and 256 GB of memory.

The SAR ADC is a popular data conversion topology choice due to its power efficiency and wide range of applications [10, 24, 31]. This section illustrates the design of a SAR ADC proposed in Liu et al. [17] by the proposed MOHSENN. Figure 5 shows the ADC's top-level block diagram, including four main modules: a comparator (COMP), a track-and-hold with a capacitive DAC (THCDAC), a SAR sequential logic for driving THCDAC (SEQ1), and a SAR sequential logic for clocking the comparator (SEQ2). We set the user desired specification as $\mathbf{u} = [n_{bit}, freq_S]$, which holds the number of bits and sampling frequency to cover a wide range of operations for the SAR ADC. In this case study, we use 65-nm CMOS technology and the design is at the schematic level.

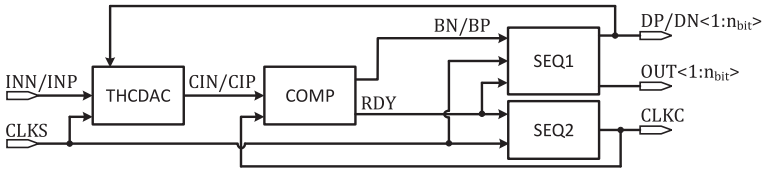


Fig. 5. SAR ADC top level.

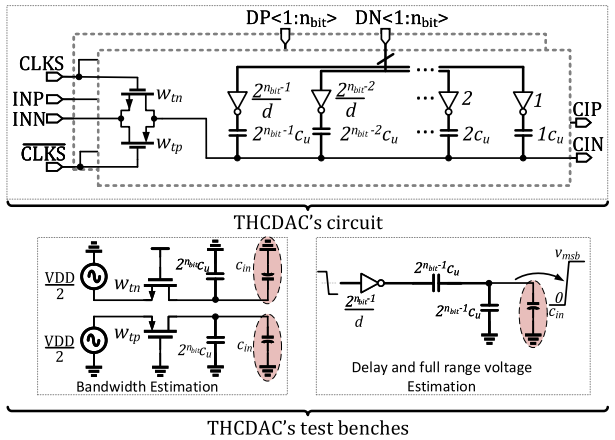


Fig. 6. THCDAC architecture and the test bench.

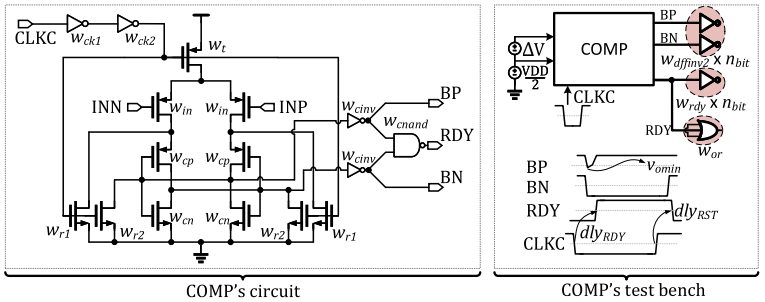


Fig. 7. COMP's architecture and the test bench.

4.1 Preparation of MOHSENN for SAR ADC Design

4.1.1 SPICE P2M Functions. Modules in the SAR ADC load each other cyclically. Therefore, if we use interface approximation, the MLG will contain a loop. To avoid this problem, we use interface replication to model the SAR logic interface with the CDAC and the comparator. There is no significant addition to the number of module parameters due to this choice. However, the comparator's loading effect on the CDAC is approximated with interface approximation for the purpose of this study.

Table 1 indicates the module parameters with their ranges and brief illustrations. It also contains the modules' metrics, their illustrations, and the required simulation type to derive them. Figure 6, Figure 7, Figure 8, and Figure 9 show the architecture and the test benches of THCDAC, COMP, SEQ2, and SEQ1, respectively. Moreover, in each figure, the design parameters are shown as part of the architecture, whereas the interface modeling is presented in the corresponding test bench. The

Table 1. Design Parameters, Their Ranges, and Modules' Metrics of the SAR ADC

Module	p	[p_{min}, p_{max}]	Description	m	Description	Sim. Type
THCDAC	c_u	[0.5fF,5.0fF]	Unit capacitance	bw_n	NMOS Sw. bandwidth	AC
	w_{tn}	[2,40]	Sw. NMOS size	bw_p	PMOS Sw. bandwidth	AC
	w_{tp}	[2,60]	Sw. PMOS size	v_{msb}	CDAC output swing	Tran
	d	[2,16]	Division factor	dly_{DAC}	DAC settling time	Tran
	n_{bit}	[3,11]	ADC resolution			
	c_{in}	[2fF,30fF]	Interface approx.			
COMP	w_{ck1}	[1,10]	Inv. size	pw_c	Power	Tran
	w_{ck2}	[1,20]	Inv. size	dly_{RDY}	CLK-RDY delay	Tran
	w_{cin}	[1,20]	Input PMOS size	dly_{RST}	Reset delay	Tran
	w_{cn}	[1,40]	NMOS size	v_{omin}	Output common mode	Tran
	w_{cp}	[1,40]	PMOS size	c_{in}	Input Capacitance	AC
	w_t	[2,80]	Tail PMOS size	$noise_c$	Input referred noise	Noise
	w_{r1}	[1,8]	Reset NMOS size		—	
	w_{r2}	[1,8]	Reset NMOS size		—	
	w_{cinv}	[1,40]	Inv. load size		—	
	w_{cnand}	[1,8]	NAND load size		—	
	n_{bit}	[3,11]	ADC resolution		—	
	$w_{dffinv2}$	[1,16]	Interface replicat.		—	
	w_{rdy}	[1,48]	Interface replicat.		—	
	w_{or}	[1,10]	Interface replicat.		—	
SEQ1	w_{inv1}	[1,12]	Inv. size	pw_{s1}	Seg.#1 power	Tran
	w_{inv2}	[2,24]	Inv. size	pw_{s2}	Seg.#2 power	Tran
	w_{inv3}	[2,96]	Inv. size	dly_{s1}	RDY-DP delay	Tran
	w_{nand}	[1,16]	NAND size	dly_{s2}	RDY-CLKQ delay	Tran
	$w_{dffinv1}$	[2,16]	Seg.#1 DFF Inv. size		—	
	w_{dffck1}	[1,10]	Seg.#1 DFF Inv. size		—	
	$w_{dffinv2}$	[1,16]	Seg.#2 DFF Inv. size		—	
	w_{dffck2}	[1,16]	Seg.#2 DFF Inv. size		—	
	$w_{dffnand}$	[2,32]	Seg.#2 DFF NAND size		—	
	n_{bit}	[3,11]	ADC resolution		—	
	d	[2,16]	Division factor		—	
SEQ2	w_{or}	[1,10]	OR size	pw_{or}	OR gate power	Tran
	w_{ck1}	[1,12]	Interface replicat.	pw_{buf}	Buffer power	Tran
				pw_{drv}	Driver power	Tran
				dly_{or}	OR gate delay	Tran
				dly_{buf}	Buffer delay	Tran
				dly_{drv}	Driver delay	Tran

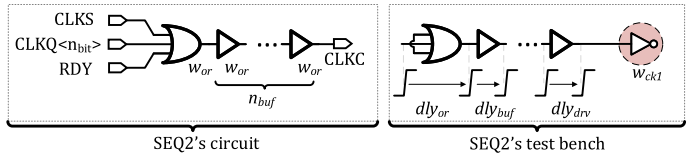


Fig. 8. SEQ2 architecture and the test bench.

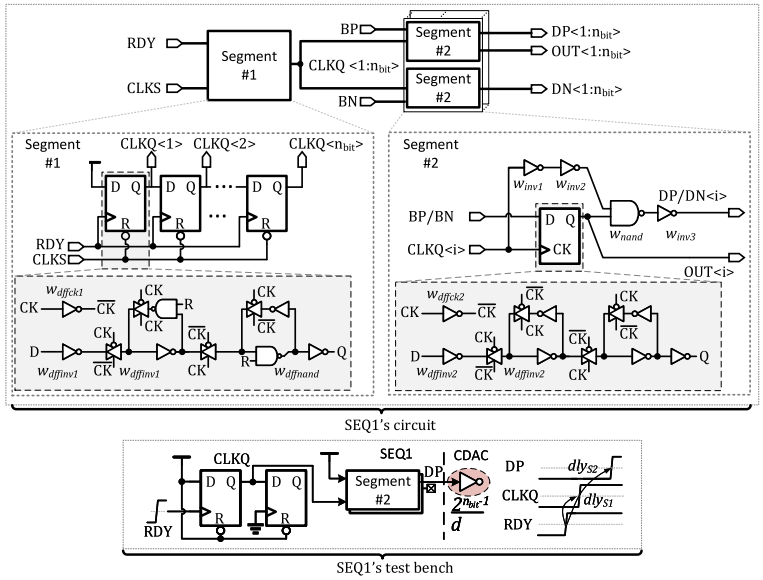


Fig. 9. SEQ1 architecture and the test bench.

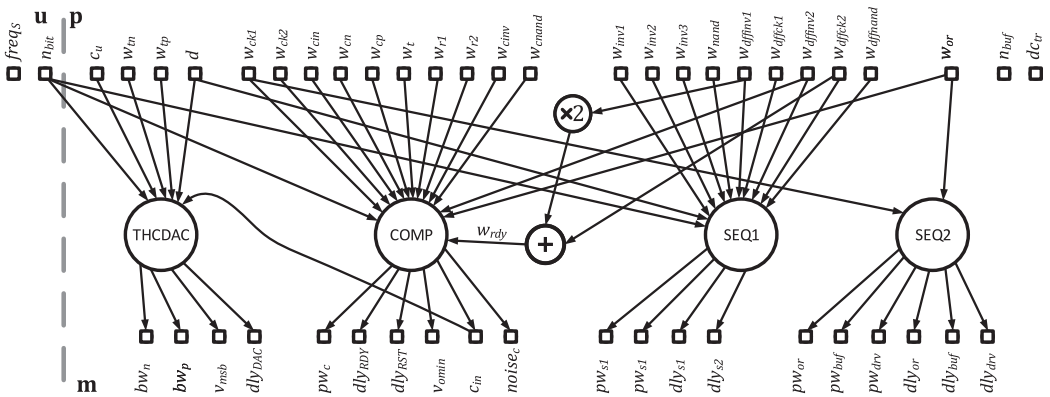


Fig. 10. SAR ADC MLG.

SAR ADC modules contain 26 design parameters, with 24 of them belonging to modules gathered in Table 1. It has two system-level parameters, which are $\mathbf{p}^B = [n_{buf}, dc_{tr}]$. The first is the buffer number used in SEQ2, and the second is the duty cycle of the sampling clock.

4.1.2 MLG Construction. After module breakdown, we construct the MLG. As mentioned previously, $g(\mathbf{p}, \mathbf{u}, [f \vee \hat{f}])$ is a function of user desired specifications, module parameters, and module test benches (P2M functions). Here, the only specification causing an effect is n_{bit}, which is a parameter used in THCDAC, COMP, and SEQ1. The SAR ADC also has two block-level parameters $\mathbf{p}^b = [n_{buf}, dc_{tr}]$, which are the number of buffers used in SEQ2 and THCDAC's tracking time duty cycle. The details related to both of these parameters are shown in Table 1. By knowing all of the parameters, metrics, and interfaces, we can construct the MLG shown in Figure 10. The shared vertices of d, w_{ck2}, $w_{dffinv1}$, $w_{dffinv2}$, w_{dffck2}, w_{or} show the interface replication graph construction as discussed earlier in Section 3.1. For the specific case of $w_{rdy} = 2w_{dffinv1} + w_{dffck2}$,

Table 2. NN Topology and Hyper-Parameters Used in SAR ADC Design

Module	Neurons per Layer	ϕ	$\lvert D_{train}\rvert$	$\lvert D_{test}\rvert$	Training MAE Loss	Test MAE Loss
THCDAC	[6, 64, 256, 64, 4]	Sigmoid	7,500	2,500	0.013	0.0140
COMP	[14, 256, 512, 256, 6]	Sigmoid	7,500	2,500	0.013	0.014
SEQ1	[11, 256, 512, 256, 4]	Sigmoid	7,500	2,500	0.011	0.014
SEQ2	[2, 50, 50, 6]	Sigmoid	120	120	0.0003	0.0003

we placed the relationship in the graph. The only interface approximation modeling is c_{in} connecting COMP's metric to THCDAC's parameter.

4.1.3 System-Level Design Objective and Constraints. For the SAR ADC design, we construct the objective function obj and constraints $\mathbf{h}$ by

$$\begin{aligned} obj &= power, \\ h_1 &= digital_{dly} - analog_{dly} \geq 0 \\ h_2 &= \frac{1}{freq_S} - t_{tr} - t_q \geq 0 \\ h_3 &= SNR - 6n_{bit} - 11.76 \geq 0 \\ h_4 &= 2\pi t_{tr} \min(bw_n, bw_p) - n_{bit} \ln 2 \geq 0 \\ h_5 &= 4 - 2\pi t_{tr} \min(bw_n, bw_p) + n_{bit} \ln 2 \geq 0 \\ h_6 &= v_{omin} - 0.6 \geq 0, \end{aligned} \tag{9}$$

where

$$\begin{aligned} digital_{dly} &= dly_{or} + n_{buf} \times dly_{buf} + dly_{drv} \\ analog_{dly} &= dly_{s1} + dly_{s2} + dly_{DAC} \\ t_q &= n_{bit}\left(dly_{RDY} + dly_{RST} + 2digital_{dly}\right) \\ t_{tr} &= dc_{tr}/freq_S \\ SNR &= 10\log\left(\frac{(4v_{msb})^2}{4KT/(2^{n_{bit}}c_u + c_{in}) + (noise_c)^2}\right) \\ power &= n_{bit}\left(pw_{s1} + pw_{s2} + pw_{or} + n_{buf} \times + pw_{buf} + pw_{drv} + pw_c\right). \end{aligned} \tag{10}$$

In (9), h_1 and h_2 hold timing constraints imposed by two different SAR ADC's loop. h_3 holds the signal to noise ratio requirements, whereas h_4 and h_5 indicate THCDAC's bandwidth requirement. h_6 sets constraints on COMP's output common mode voltage, and obj is the power consumption.

4.1.4 NN Regression Models. We gathered information regarding the NN models for the four different modules, as presented in Table 2. For COMP, THCDAC, and SEQ1, we used an NN regression model with five layers, whereas for SEQ2, we used four layers. The number of neurons for each layer is shown in Table 2. To determine the number of layers and neurons per layer, we used the method illustrated in Wolfe and Vemuri [34]. The activation function of all layers except the last one is the Sigmoid function (i.e., $\phi(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$). The last layer uses a linear activation function. Other than SEQ2's model, we trained the NN models with a dataset of $\lvert D_{train}\rvert$ =7,500 and tested it with $\lvert D_{test}\rvert$ =2,500 sample points generated by the previously described test benches. Generating the whole dataset took almost 5 hours and 50 minutes. For training, we used the Adam algorithm, and the loss function is the **mean absolute error (MAE)**. The training and testing loss for the given datasets are presented in Table 2.

Table 3. Metrics Introduced in Table 1, the Statistical Parameters, and the Regression Error by kNN, RF, SVR, and NN Modeling with Different Sizes of Training Dataset $|D_{train}|$

$\mathbf{m}$	$\mu(\mathbf{m})$	$\sigma(\mathbf{m})$	kNN $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 7,500	RF $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 7,500	SVR $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 7,500	NN $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 1,000	NN $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 2,500	NN $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 5,000	NN $\epsilon(\hat{\mathbf{m}})$ $\lvert D_{train}\rvert$ = 7,500
THCDAC Metrics									
bw_n [GHz]	17.33	32.19	6.80	3.20	3.03	1.12	0.58	0.33	0.17
bw_p [GHz]	8.17	15.62	3.46	1.46	1.53	0.50	0.26	0.15	0.13
v_{msb} [mV]	475.55	39.93	8.24	3.77	1.85	1.04	0.35	0.26	0.21
dly_{DAC} [pS]	241.86	172.52	30.05	15.63	7.70	9.87	7.78	7.45	6.92
COMP Metrics									
pw_c [μW]	118.93	46.63	13.61	10.77	5.79	9.25	6.24	3.10	1.36
dly_{RDY} [pS]	320.49	226.78	68.24	46.75	40.68	38.0	24.6	15.5	7.77
dly_{RST} [pS]	170.90	116.6	38.10	27.69	21.93	26.8	17.7	9.85	4.58
v_{omin} [mV]	745.8	229.0	91.4	47.43	39.48	34.7	21.0	15.8	8.32
c_{in} [fF]	17.1	8.0	2.7	0.87	0.17	0.15	0.12	0.10	0.06
$noise_c$ [μV^2]	367.8	255.5	84.51	29.41	52.12	54.7	35.7	30.9	13.8
SEQ1 Metrics									
pw_{s1} [μW]	189.5	606.8	181.47	31.30	119.13	43.7	20.0	13.0	5.68
pw_{s2} [μW]	12.5	7.1	1.59	0.45	0.37	0.21	0.12	0.08	0.05
dly_{s1} [pS]	159.5	258.7	72.12	29.18	44.68	30.0	14.5	9.65	6.04
dly_{s2} [pS]	59.1	25.3	9.80	2.60	5.52	1.70	0.90	0.66	0.50
SEQ2 Metrics, $\lvert D_{train}\rvert$ = 120 for All Cases									
pw_{or} [μW]	10.2	6.9	0.60	0.11	0.08	—	—	—	0.01
pw_{buf} [μW]	3.06	1.88	0.164	0.031	0.028	—	—	—	0.004
pw_{drv} [μW]	3.43	1.26	0.121	0.025	0.024	—	—	—	0.005
dly_{or} [pS]	31.60	1.85	0.355	0.008	0.148	—	—	—	0.007
dly_{buf} [pS]	12.71	1.62	0.286	0.007	0.094	—	—	—	0.006
dly_{drv} [pS]	14.76	4.33	0.580	0.018	0.185	—	—	—	0.014

With the available dataset, we approximated all modules' metrics with four different non-linear regression models that have been used in circuit design tools (SVR, kNN, RF, and NN), and their corresponding regression errors are presented in Table 3. For SVR, kNN, and RF, we used the Python package of scikit-learn [28], and for NN, we used TensorFlow. For SVR, we used the non-linear rbf kernel, tol of 10^{-3}, and epsilon of 10^{-12}, and the remaining hyper-parameters are scikit-learn's default. For kNN, we set the number of neighbors to 100, and for RF, we set the number of estimators to 100. We chose these hyper-parameters based on manual tuning to achieve the minimum loss. We included the average $\mu()$ and standard deviation $\sigma()$ of metrics from the training dataset in Table 3 to be compared with regression errors. Comparatively, among all regression types, we can observe that NN illicits fewer regression errors for all metrics when the number of training sample points ($|D_{train}|$) are equal to 7,500. We varied the $|D_{train}|$ to examine its effect on the NN model accuracy and presented the results in Table 3. As mentioned before, the NN model regression error can be reduced significantly by augmenting the dataset.

4.2 Search Engine Results

After preparation, the CAD tool with the MOHSENN algorithm can receive $\mathbf{u} = [n_{bit},\ freq_S]$ from the user to generate the corresponding circuit. In this work, we start from a design of

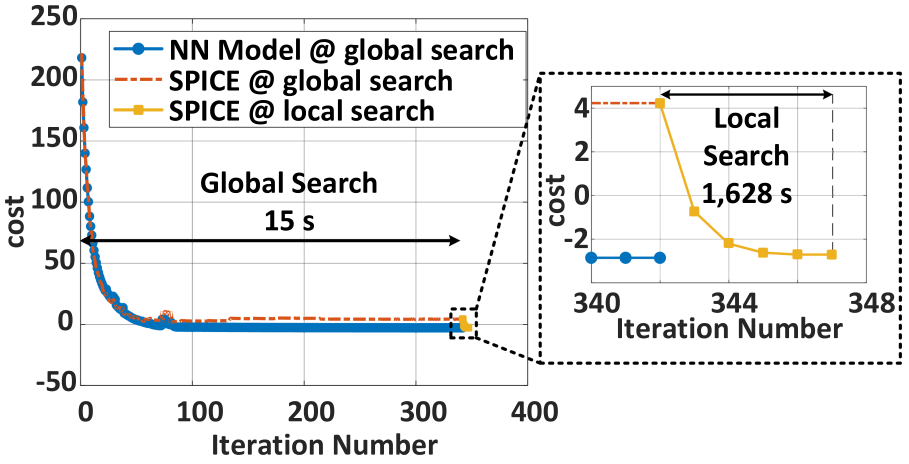


Fig. 11. *cost* vs. optimization iteration in global search phase, and local search phase.

$\mathbf{u}$ =[6, 500MS/s]. We show the global and local search phases when n_{mc} is 1 (single-run Adam optimization) and when the stopping criterion is restrictive $tol = 0.001$ for local optimization. In the next experiment, we change n_v to test whether reducing the dimensions with the proposed gradient-based variable reduction method can still produce reliable results. We also examine the efficiency of the Par-MC and Adam algorithm, and we compare it to SA optimization and the conventional sequential MC method. Next, we gathered three user desired specifications for $\mathbf{u}$ =[10, 200MS/s], $\mathbf{u}$ =[8, 340MS/s], and $\mathbf{u}$ =[6, 500MS/s] designed by MOHSENN. Finally, we will compare MOHSENN methodology to other circuit sizing algorithms.

We tested MOHSENN's hybrid search performance on SAR ADC design. The SAR ADC's cost function curve vs. optimization iterations for both global and local searches are presented in Figure 11. For this test in the global search, the hyper-parameters were the learning rate $\gamma = 0.02$, $ftol = 10^{-5}$, and $n_{mc} = 1$, and it took 342 iterations for convergence. In the local search, the hyper-parameters are $ftol = 0.001$ and $n_v = 10$, and it took 337 SPICE evaluations within six optimization iterations for successful convergence. It can be seen that the global search improves the initial starting point significantly from $cost(\mathbf{x}^{(0)}, \hat{f}) = 218.05$ to $cost(\mathbf{x}_{og}, \hat{f}) = -2.85$ based on NN modeling. Because of the fast NN evaluation, this step converges within 15 seconds. We evaluated every iteration of the global search with the SPICE model to examine the cost function trend without the regression errors. During the global search, the cost function is initially $cost(\mathbf{x}^{(0)}, f) = 221.66$, which is minimized to $cost(\mathbf{x}_{og}, f) = 4.22$ with the SPICE models. In the local search, the Powell algorithm conducts many SPICE evaluations per iteration. Therefore, for only six iterations, convergence takes more than 1,600 seconds. The final found point is $cost(\mathbf{x}_{ol}, f) = -2.7985$, which is evaluated by the SPICE models.

We compared the proposed Par-MC/Adam optimization ($n_{mc} = 50$) with the SA optimizer to examine the efficiency of the proposed scheme for a global search. The results are presented in Figure 12. SA optimization is stochastic and global, and conventionally used for circuit design problems [9, 25, 27]. During one SA optimization iteration, several hundred function evaluations are performed. We used an available open source package of SciPy [33] and the function "scipy.optimize.dual_annealing." Figure 12(a) and Figure 12(b) show $cost(\mathbf{x}, \hat{f})$ vs. the optimization iteration and the optimization time, respectively. These two figures show that SA can achieve the cost function of $cost(\mathbf{x}_{og}, \hat{f}) = -3.00$ after 960 iterations. The Adam optimizer with Par-MC

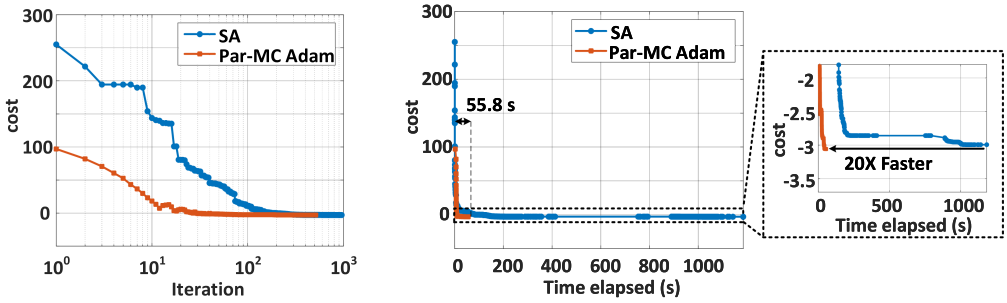


Fig. 12. Global search phase with Par-MC/Adam optimization (n_{mc} = 20) vs. the SA optimizer in number of iterations (a) and required time (b).

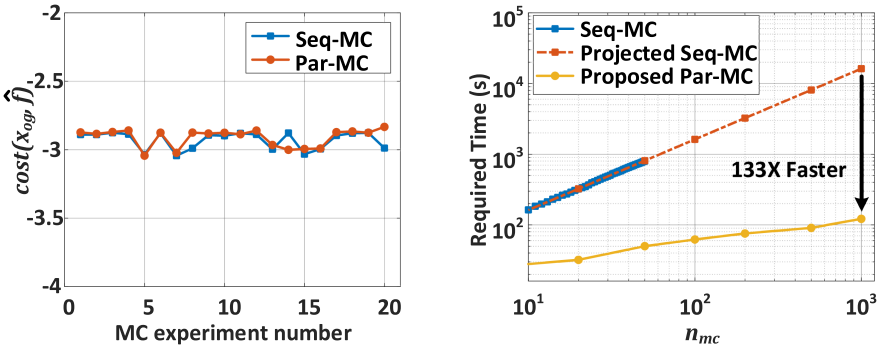


Fig. 13. Comparison between Par-MC and conventional sequential MC. (a) The optimization result for 20 MC samples with identical initialization. (b) The required time for different n_{mc}.

requires 515 iterations to achieve $cost(\mathbf{x}_{og}, \hat{f}) = -3.06$. Par-MC with Adam converges to the optimized value within only 56 seconds, whereas it takes 1,185 seconds for SA to converge. This time difference is as result of SA's more function evaluations per iteration relative to the Par-MC method. This experiment shows that Par-MC with Adam optimization can reach the same optimized point derived by the conventional SA algorithm at least 20 times quicker.

Figure 13 shows a comparison between conventional sequential MC vs. Par-MC. In Figure 13(a), we show the optimization results for 20 different initial points. It can be seen that the final results between the sequential MC and Par-MC are similar, as claimed in Section 3.2.2. Interestingly, all derived results from the Adam optimizer for this problem are approximately close to the competitive point of $cost(\mathbf{x}_{og}, \hat{f}) = -3.06$. It infers that even without MC, the Adam optimizer can achieve respectable results for SAR ADC's design problem. Figure 13(b) presents a comparison of Par-MC with sequential MC in terms of speed. In sequential MC, the required time is linearly proportional to n_{mc}. Therefore, we tested sequential MC for n_{mc} between 10 and 50, then we linearly projected the possible required time for n_{mc}=1,000, where it took almost 16,000 seconds (≈4 hours). However, the same number of samples for Par-MC only requires 121.5 seconds. In Section 3.2.2, we claimed that Par-MC's completion time does not increase linearly with time on the TensorFlow platform. In this example, we can achieve more than 130 times faster speed when using Par-MC (when n_{mc} = 1,000).

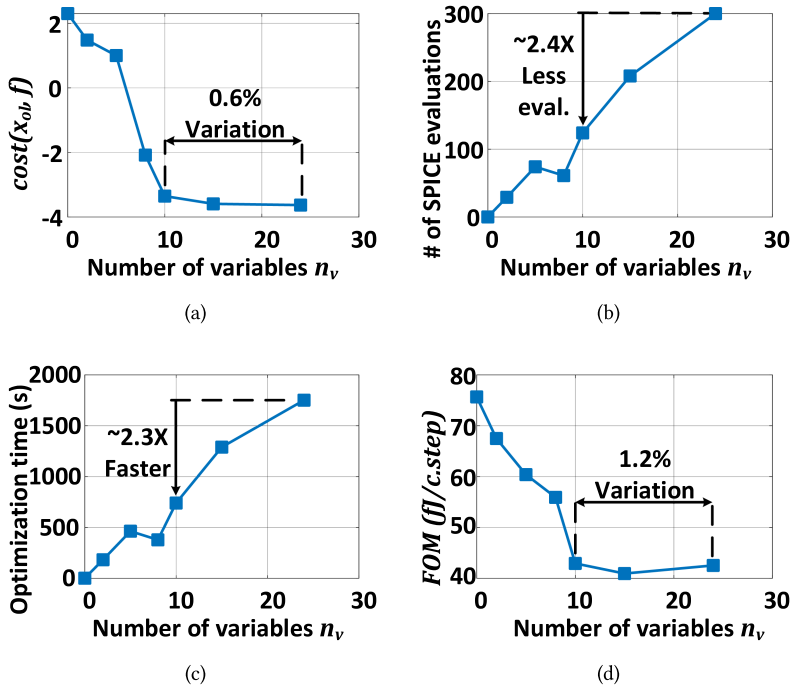
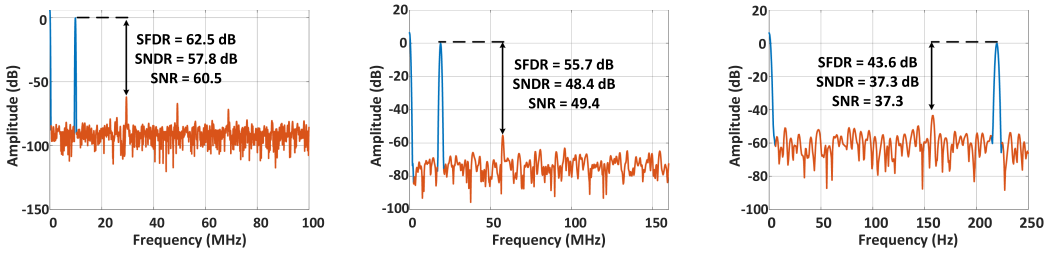


Fig. 14. The local optimization results of 6-bit, 500 MS/s ADC vs. the number of variables n_v cost function (a), number of iterations (b), optimization time (c), and ADC's FOM (d).

We gathered the local search results with SPICE models and the Powell algorithm when altering n_v values to examine its effects on the final optimization results. For this test, the initial cost function is $cost(\mathbf{x}_{og}, f)$ =2.30. The Powell algorithm from the SciPy package [33] in "scipy.optimize.minimize" was used with a stopping criteria of $tol = 0.1$. Figure 14 summarizes the effect of optimization result vs. n_v. Figure 14(a) shows the final achieved optimized value $cost(\mathbf{x}_{ol}, f)$ vs. n_v. This clearly shows that after n_v =10, there is no significant improvement over the cost function. In other words, the other 16 variables with higher gradients do not contribute to the local optimization. We can observe the benefit of reducing the variables in Figure 14(b) and Figure 14(c), which show the required number of iterations and optimization time, respectively. Both of these increase linearly with n_v, which indicates that reducing the variables can significantly help to accelerate the search. For each local optimization with different n_v, we derived $\mathbf{p}_{opt}$ and tested them in the ADC verification test bench. Figure 14(d) shows the measured FOM of the SAR ADC derived from the SPICE simulation, and there is no significant improvement of FOM if $n_v \geq$10. For this example, with the introduction of a variable reduction technique by only considering n_v =10 variables, we can achieve the same FOM as n_v =26 with almost 2.3 times faster convergence time.

4.2.1 Different User Desired Specifications. We examined the diverse design capabilities of MOHSENN by three different user desired specifications of $\mathbf{u}$ =[10, 200 MS/s], $\mathbf{u}$ =[8, 340MS/s], and $\mathbf{u}$ =[6, 500 MS/s]. For this test, we use MOHSENN's default hyper-parameters. The global search uses the Adam optimizer with $\gamma = 0.02$, $n_{mc} = 500$, $ftol = 10^{-5}$, and $k_{max} = 10,000$. The local search uses the Powell optimizer with $ftol = 0.1$ and $k_{max} = 1,000$. For all three cases, we simulated the SAR ADC with MOHSENN's offered $\mathbf{p}_{opt}$. We tested the ADCs with a monotone



(a) (b) (c)

Fig. 15. The output spectrum of the MOHSENN-designed SAR ADC when input is a single tone sine wave for [10-bit, 200 MS/s] (a), [8bit, 340 MS/s] (b), and [6-bit, 500 MS/s] (c).

Table 4. MOHSENN Result of the SAR ADC with Given $\mathbf{u}$

Desired Spec. (u)	**[10, 200MS/s]**	**[8, 340MS/s]**	**[6, 500MS/s]**
Global Search iter (#)	1,149,500	360,500	282,000
Global Search Time	394s	124s	91s
Local Search iter. (#)	49	171	89
Local Search Time	311s	1,224s	739s
Verification Time	5h 35m 23s	1h 39m 47s	45m 18s
SNDR (Global/Local) [dB]	57.8/57.8	48.9/48.4	37.0/37.3
SNR (Global/Local) [dB]	60.6/60.5	49.4/49.4	37.0/37.3
SFDR (Global/Local) [dB]	62.0/62.5	59.8/55.7	50.1/43.6
ENOB (Global/Local) [bit]	9.34/9.18	7.86/7.75	5.87/5.92
Power (Global/Local) [mW]	2.70/2.40	2.91/2.58	2.22/1.30
FOM (Global/Local) [fJ/c.step]	20.8/18.5	36.7/35.5	75.9/42.9

sine wave input; their output spectra are shown in Figure 15, and the results are presented in Table 4. For the 10- and 8-bit ADC design cases, there is no significant improvement with the local search. This means that the global search result is already close to the nearest optimal point, and there is no need for further optimization. So the local optimization time could be avoided for these two cases. However, for the 6-bit ADC, the FOM changes from 67.0 to 43.7 fJ/c.step, which is a significant improvement.

4.2.2 Comparison to Conventional Methods. To examine the efficiency of MOHSENN, we compared it to two other algorithms. The first is a conventional top-down hierarchical design [2, 20, 22, 27] illustrated in Section 2.1. In this method, for the system level we conduct a global search of MOHSENN for the system-level optimization yielding $\mathbf{p}_{og}$, and thus feasible metrics $\hat{\mathbf{m}}_{og}$. In the module level, we optimize each module separately to render $f^i(\mathbf{p}^i) = \hat{\mathbf{m}}^i_{og}$, or make $f^i(\mathbf{p}^i)$ better than $\hat{\mathbf{m}}^i_{og}$. For the module-level optimization, we use SA optimization and SPICE modeling for accurate results. However, this approach designs modules separately (causing interface problems), as illustrated in Section 2.1.

The other algorithm is to directly use SPICE for the global search and use SA for the optimization. In other words, the optimization problem is the same as (5) with $\boldsymbol{f}$. We call this approach the *Flat-SPICE method*. Since it is a global optimization directly on the overall AMS circuit functional model with SPICE $obj(g(., \boldsymbol{f}))$ and $\mathbf{h}(g(., \boldsymbol{f}))$, we expect to find the global optimal point with no regression errors. However, the optimization time is prolonged.

Table 5. Design Parameters and the Search Space of the SAR ADC

Design Method	Proposed MOHSENN	Hierarchical Design	Flat-SPICE (fast SA)	Flat-SPICE (slow SA)
Dataset Generation Time	5h 50m	5h 50m	0s	0s
NN Evaluations (#)	282,000	282,000	0	0
NN Search Time	91s	91s	0s	0s
SPICE Evaluations (#)	89	4,178	27,000	103,302
SPICE Search Time	12m 19s	1h 9m 38s	1d 22h 12m 4s	7d 12h 2m 47s
Total Search Time	13m 50s	1h 11m 17s	1d 22h 12m 4s	7d 12h 2m 47s
Total Required Time	6h 3m	7h 1m	1d 22h	7d 12h 2m
SNDR	37.30 dB	37.50 dB	24.07 dB	37.76 dB
Power	1.30 mW	2.00 mW	1.77 mW	1.67 mW
FOM	43 fJ/c.step	67 fJ/c.step	269 fJ/c.step	52 fJ/c.step

We designed a 6-bit, 500-MS/s ADC with these two methods and then inserted the parameter candidate to the SAR ADC verification test bench. The result is shown in Table 5. MOHSENN uses its default hyper-parameters. In the hierarchical design, the first step is similar to MOHSENN's global search, which uses NN modeling and MLG. However, the second step involves designing each module separately (without MLG). For the second step of hierarchical design, we used SA in the SciPy [33] library with a default format, and we only changed the maximum number of iterations. For the comparator design the number is 50, and for the rest it is 10. Further, for the second-level design, we do not take $\mathbf{p}^B$ as variables, and they are set by system-level optimization. In Table 5, the number of SPICE simulations is the summation of running each module test bench. As indicated in Table 5, the hierarchical approach converges after 70 minutes, whereas MOHSENN converges within approximately 14 minutes. Accordingly, the proposed algorithm's search engine is five times faster than the conventional hierarchical method while achieving a better FOM for the SAR ADC.

The other approach (Flat-SPICE) is a single-run SA optimization. Since one optimization convergence would take a considerable amount of time, we demonstrated this method for two cases. One is SA with a maximum number of iterations of 100, which we call *fast Flat-SPICE*. For the other case, this number is 500, and we call it *slow Flat-SPICE*. The fast technique took almost 2 days to converge, and the results were not satisfactory. However, the slow Flat-SPICE converged to an excellent point for the SAR ADC (i.e., a power consumption of 1.67 mW and SNDR of 37.76 dB). However, this required more than 7 days to achieve this point, and yet MOHSENN submits an approximately better result within total required time of 6 hours and 3 minutes including the initial training and testing dataset generation. This experiment shows that MOHSENN is capable of designing a SAR ADC as effectively as the Flat-SPICE method while converging 30 times faster.

5 CONCLUSION

This article proposes a method for automatic AMS parameter synthesis. The method uses a novel MLG methodology to enforce equality between the interfaces of adjacent modules. This leads to more accurate characterization of modules and, as a result, a better AMS design. Moreover, this method accelerates the parameter search by the proposed hybrid search. During global search with NN models, we used the proposed Par-MC technique with Adam optimization, which reduced the convergence time to minutes. In the local search with SPICE models, we introduced a novel variable reduction technique based on the gradients achieved from the NN regression models, which significantly accelerated the search time. The effectiveness of this approach was tested on

a SAR ADC with 26 parameters, where it achieved almost five times faster speed and better final performance compared to the conventional hierarchical approach.

ACKNOWLEDGMENTS

The work is supported by Defense Advanced Research Projects Agency (DARPA) under the ERI POSH program. The authors would like to thank Prof. Pierluigi Nuzzo, Prof. Anthony F. J. Levi, and Prof. Sandeep K. Gupta from the University of Southern California for valuable technical discussions.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2015. TensorFlow:Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] António Manuel Lourenço Canelas, Jorge Manuel Correia Guilherme, and Nuno Cavaco Gomes Horta. 2020. *AIDA-C Variation-Aware Circuit Synthesis Tool*. Springer International, Cham, Switzerland, 155–177. DOI: https://doi.org/10.1007/978-3-030-41536-5_5

[3] V. Ceperic and A. Baric. 2004. Modeling of analog circuits by using support vector regression machines. In *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits, and Systems (ICECS'04)*.IEEE, Los Alamitos, CA, 391–394.

[4] W. Daems, G. Gielen, and W. Sansen. 2003. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 5 (May 2003), 517–534. DOI: https://doi.org/10.1109/TCAD.2003.810742

[5] F. De Bernarclinis, S. Gambini, R. Vincis, F. Svelto, A. Sangiovanni Vincentelli, and R. Castello. 2004. Design space exploration for a UMTS front-end exploiting analog platforms. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)*. IEEE, Los Alamitos, CA, 923–930.

[6] F. De Bernardinis and A. Sangiovanni Vincentelli. 2005. Efficient analog platform characterization through analog constraint graphs. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'05)*. IEEE, Los Alamitos, CA, 415–421. DOI: https://doi.org/10.1109/ICCAD.2005.1560104

[7] R. De Bernardinis, P. Nuzzo, and A. Sangiovanni Vincentelli. 2005. Mixed signal design space exploration through analog platforms. In *Proceedings of the 2005 42nd Design Automation Conference*. IEEE, Los Alamitos, CA, 875–880. DOI: https://doi.org/10.1145/1065579.1065808

[8] M. del Mar Hershenson. 2002. Design of pipeline analog-to-digital converters via geometric programming. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer Aided Design (ICCAD'02)*. IEEE, Los Alamitos, CA, 317–324.

[9] T. Eeckelaert, T. McConaghy, and G. Gielen. 2005. Efficient multiobjective synthesis of analog circuits using hierarchical Pareto-optimal performance hypersurfaces. In *Proceedings of the Design, Automation, and Test in Europe Conference*. IEEE, Los Alamitos, CA, 1070–1075. DOI: https://doi.org/10.1109/DATE.2005.129

[10] Q. Fan and J. Chen. 2019. A 1-GS/s 8-Bit 12.01-fJ/conv.-step two-step SAR ADC in 28-nm FDSOI technology. *IEEE Solid-State Circuits Letters* 2, 9 (Sept. 2019), 99–102. DOI: https://doi.org/10.1109/LSSC.2019.2934351

[11] O. Garitselov, S. P. Mohanty, and E. Kougianos. 2012. Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization. In *Proceedings of the 2012 25th International Conference on VLSI Design*. IEEE, Los Alamitos, CA, 316–321.

[12] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović. 2019. BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. IEEE, Los Alamitos, CA, 1–8.

[13] M. Hassanpourghadi and M. Sharifkhani. 2013. Fast static characterization of residual-based ADCs. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60, 11 (Nov. 2013), 746–750. DOI: https://doi.org/10.1109/TCSII.2013.2281908

[14] Y. Huo, X. Dong, and W. Xu. 2017. 5G cellular user equipment: From theory to practical hardware design. *IEEE Access* 5 (2017), 13992–14010.

[15] D. Kingma and J. Ba. 2014. Adam: A Method for Stochastic Optimization. arxiv:cs.LG/1412.6980

[16] Y. Li, Y. Wang, Y. Li, R. Zhou, and Z. Lin. 2020. An artificial neural network assisted optimization system for analog design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2640–2653.

[17] C. Liu, S. Chang, G. Huang, and Y. Lin. 2010. A 10-bit 50-MS/s SAR ADC with a monotonic capacitor switching procedure. *IEEE Journal of Solid-State Circuits* 45, 4 (April 2010), 731–740. DOI: https://doi.org/10.1109/JSSC.2010.2042254

[18] J. Liu, M. Hassanpourghadi, Q. Zhang, S. Su, and M. S. W. Chen. 2020. Transfer learning with Bayesian optimization-aided sampling for efficient AMS circuit modeling. In *Proceedings of the 2020 IEEE/ACM International Conference on Computer Aided Design (ICCAD'20)*. IEEE, Los Alamitos, CA, 1–9.

[19] R. Lourenço, N. Lourenço, and N. Horta. 2015. *AIDA-CMK: Multi-Algorithm Optimization Kernel Applied to Analog IC Sizing*. Springer International. DOI: https://doi.org/10.1007/978-3-319-15955-3

[20] N. Lourenço, E. Afacan, R. Martins, F. Passos, A. Canelas, R. Póvoa, N. Horta, and G. Dundar. 2019. Using polynomial regression and artificial neural networks for reusable analog IC sizing. In *Proceedings of the 2019 16th International Conference on Synthesis, Modeling, Analysis, and Simulation Methods and Applications to Circuit Design (SMACD'19)*. IEEE, Los Alamitos, CA, 13–16.

[21] N. Lourenço, R. Martins, A. Canelas, R. Póvoa, and N. Horta. 2016. AIDA: Layout-aware analog circuit-level sizing with in-loop layout generation. *Integration* 55 (2016), 316–329.

[22] R. Martins, N. Lourenço, R. Póvoa, A. Canelas, N. Horta, F. Passos, R. Castro-López, E. Roca, and F. Fernández. 2017. Layout-aware challenges and a solution for the automatic synthesis of radio-frequency IC blocks. In *Proceedings of the 2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD'17)*. IEEE, Los Alamitos, CA, 1–4. DOI: https://doi.org/10.1109/SMACD.2017.7981577

[23] T. McConaghy and G. G. E. Gielen. 2009. Template-free symbolic performance modeling of analog circuits via canonical-form functions and genetic programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 8 (2009), 1162–1175.

[24] J. Nam, M. Hassanpourghadi, A. Zhang, and M. S. Chen. 2018. A 12-bit 1.6, 3.2, and 6.4 GS/s 4-b/cycle time-interleaved SAR ADC with dual reference shifting and interpolation. *IEEE Journal of Solid-State Circuits* 53, 6 (June 2018), 1765–1779. DOI: https://doi.org/10.1109/JSSC.2018.2808244

[25] P. Nuzzo, F. De Bernardinis, and A. Sangiovanni-Vincentelli. 2006. Platform-based mixed signal design: Optimizing a high-performance pipelined ADC. *Analog Integrated Circuits and Signal Processing* 49, 3 (Dec. 2006), 343–358. DOI: https://doi.org/10.1007/s10470-006-9067-8

[26] P. Nuzzo, F. De Bernardinis, P. Terreni, and A. Sangiovanni Vincentelli. 2005. Enriching an analog platform for analog-to-digital converter design. In *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems*. IEEE, Los Alamitos, CA, 1286–1289.DOI: https://doi.org/10.1109/ISCAS.2005.1464830

[27] P. Nuzzo, A. Sangiovanni-Vincentelli, X. Sun, and A. Puggelli. 2012. Methodology for the design of analog integrated interfaces using contracts. *IEEE Sensors Journal* 12, 12 (Dec 2012), 3329–3345. DOI: https://doi.org/10.1109/JSEN.2012.2211098

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[29] B. Razavi. 2000. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

[30] Luis Rios and Nikolaos Sahinidis. 2009. Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization* 56 (Nov. 2009), 1247–1293. DOI: https://doi.org/10.1007/s10898-012-9951-y

[31] A. Samiei and H. Hashemi. 2019. A chopper stabilized, current feedback, neural recording amplifier. *IEEE Solid-State Circuits Letters* 2, 3 (March 2019), 17–20. DOI: https://doi.org/10.1109/LSSC.2019.2916754

[32] X. Tang and A. Xu. 2016. Multi-class classification using kernel density estimation on *K*-nearest neighbours. *Electronics Letters* 52, 8 (2016), 600–602.

[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, et al. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods* 17 (2020), 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2

[34] G. Wolfe and R. Vemuri. 2003. Extraction and use of neural network models in automated synthesis of operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 2 (Feb. 2003), 198–212. DOI: https://doi.org/10.1109/TCAD.2002.806600

[35] T. Wu, C. Alkan, and T. W. Chen. 2009. Complexity reduction for analog circuit performance models using random forests. In *Proceedings of the 2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC'09)*. IEEE, Los Alamitos, CA, 29–34.

[36] G. İslamoğlu, T. O. Çakici, E. Afacan, and G. Dündar. 2019. Artificial neural network assisted analog IC sizing tool. In *Proceedings of the 2019 16th International Conference on Synthesis, Modeling, Analysis, and Simulation Methods and Applications to Circuit Design (SMACD'19)*. IEEE, Los Alamitos, CA, 9–12. DOI: https://doi.org/10.1109/SMACD.2019.8795293

Received August 2020; revised February 2021; accepted March 2021